universität wien

MASTERARBEIT | MASTER'S THESIS

Titel | Title

Measuring Hierarchy Depth in the Aviation and Automotive Industries

verfasst von | submitted by

Christoph Obernosterer BSc (WU) BSc MSc (WU)

angestrebter akademischer Grad | in partial fulfilment of the requirements for the degree of

Master of Science (MSc)

Wien | Vienna, 2026

Studienkennzahl lt. Studienblatt | Degree programme code as it appears on the student record sheet:	UA 066 915
Studienrichtung lt. Studienblatt | Degree programme as it appears on the student record sheet:	Masterstudium Betriebswirtschaft
Betreut von | Supervisor:	Univ.-Prof. Dipl.-Chem. Dr. Markus Georg Reitzig MBR

Abstract (English)

This thesis develops a systematic, rule-based coding approach to measure hierarchical depth in the aviation and automotive industries based on job titles. Prior research, most notably Lee (2021) in the video game start-up context, demonstrated how job title data can map hierarchical structures. However, empirical investigations of vertical differentiation within organizational structures remain scarce, although vertical structure has been associated with performance, coordination, information flow, and innovation. Reliable measures of hierarchical depth are unavailable, as organizations rarely disclose detailed organizational charts. Building on Lee's (2021) approach, this study extends the systematic measurement of hierarchical depth to the aviation and automotive industries and adapts the method to their distinct functional structures.

To enable this, this study develops a Python-based classification system that can process LinkedIn profile data through a multi-stage pipeline: preprocessing, reshaping of nested JSON structures, industry filtering, and both horizontal and vertical categorization of job titles. Individuals are assigned to standardized hierarchical levels within their respective organizations and major functional subunits. Industry expert interviews support the development and validation of the two industry-specific classification frameworks.

The results indicate that major U.S. airlines typically operate across 7–8 hierarchical layers, while the largest automotive manufacturers exhibit 8–9. Beyond descriptive insights, the frameworks provide a basis for future research on how hierarchical depth relates to corporate goals. By introducing horizontal categorization for industries with highly heterogeneous job structures – particularly in aviation – this work facilitates differentiation between complex organizations characterized by a wide variety of specialized job profiles.

Abstract (German)

Diese Masterarbeit entwickelt einen regelbasierten Code zur Messung der Hierarchietiefe in der Luftfahrt- und Automobilindustrie auf Basis von Jobtiteln. Bisherige Forschung, insbesondere die von Lee (2021) im Kontext von Start-ups in der Videospielbranche, hat gezeigt, dass sich mithilfe von Jobtiteldaten hierarchische Strukturen abbilden lassen. Dennoch sind empirische Untersuchungen zur vertikalen Differenzierung von Organisationsstrukturen bislang selten, obwohl Hierarchietiefe mit Unternehmenserfolg, Koordination, Informationsfluss und Innovation in Verbindung gebracht wird. Verlässliche Messungen fehlen, da Unternehmen detaillierte Organigramme in der Regel nicht veröffentlichen. Aufbauend auf Lee (2021) erweitert diese Arbeit die Messung der Hierarchietiefe auf die Luftfahrt- und Automobilindustrie und passt die Methode an deren besondere funktionale Strukturen an.

Hierfür wird ein Klassifikationssystem entwickelt, das auf Python basiert. Es verarbeitet LinkedIn-Profildaten in mehreren Schritten: Datenbereinigung, Auflösung verschachtelter JSON-Strukturen, Branchenfilterung sowie horizontale und vertikale Kategorisierung von Jobtiteln. Den Personen werden standardisierte Hierarchieebenen innerhalb der jeweiligen Organisation und den zentralen funktionalen Teilbereichen zugeordnet. Experteninterviews aus beiden Branchen unterstützen die Entwicklung und Validierung der beiden branchenspezifischen Klassifikationssysteme.

Die Ergebnisse zeigen, dass große US-Airlines typischerweise über 7–8 Hierarchieebenen verfügen, während die größten Automobilhersteller 8–9 Ebenen aufweisen. Neben den deskriptiven Ergebnissen bieten die Klassifikationssysteme eine Grundlage für zukünftige Forschung zur Untersuchung von Zusammenhängen zwischen Hierarchietiefe und Unternehmenszielen. Durch die zusätzliche Entwicklung einer horizontalen Kategorisierung für Branchen mit stark differenzierten Tätigkeitsprofilen – insbesondere in der Luftfahrt – ermöglicht der hier präsentierte Code eine detaillierte Analyse komplexer Organisationen mit einer Vielzahl spezialisierter Berufsbilder.

Table of Contents

1. Introduction

Organizational structure is a central design dimension of firms. Hierarchical depth reflects the vertical differentiation of authority, decision rights, and coordination mechanisms within organizations. Research on organizational theory has long emphasized that hierarchy is neither inherently beneficial nor detrimental. There are trade-offs between control, coordination, and flexibility (Mintzberg, 1989; Burton & Obel, 2004; Van de Ven et al., 2013). Research indicates that steep power differences within teams may undermine performance by discouraging open communication and amplifying status competition (Greer et al., 2018; Kilduff et al., 2016); in general, the hierarchical structures significantly influence a firm's ability to balance exploration and exploitation (Puranam, 2018). Moreover, Reitzig (2022) underlines that flattening an organization is no end in itself but may be very successful with major strategic alignments, such as a rethought task division and allocation, or novel processes for escalation, remuneration, and information flows.

Despite its theoretical relevance, hierarchical depth remains difficult to measure empirically. Corporations rarely disclose detailed organizational charts, and newly established enterprises often struggle to establish formalized organizational structure at an early stage (Stinchcombe, 1965; Davila et al., 2019). Furthermore, there is only limited availability of large-scale, reliable data sets, particularly for entrepreneurial ventures (Burton et al., 2019). Nevertheless, Lee (2021) achieved a major methodological success by demonstrating that large-scale job title data can be used to reconstruct hierarchical structures in video game start-ups. Through a systematic analysis of job titles, Lee provided a scalable measure of hierarchical layering for a specific industry.

However, job titles describe different job profiles in the array of business sectors. Therefore, each of them requires its customized modifications and validation. For example, aviation and

automotive manufacturing are characterized by complex, heterogeneous occupational structures and both industries are largely dominated by a few corporations with workforces surpassing 100,000 (Stock Analysis, 2026). Airlines integrate cockpit crews, cabin staff, ground operations, corporate management, and often even technical maintenance within a single organizational entity. Automotive manufacturers combine research and development, large-scale production systems, global supply chains, and sales. This pronounced functional differentiation complicates vertical classification and requires industry-specific coding logic which leads to the following research question:

"How can hierarchical depth in airlines and automotive manufacturers be measured reliably using large-scale job title data?"

Building on Lee's (2021) methodological approach, this study develops a systematic, rule-based coding framework to empirically reconstruct hierarchical depth in two established industries. Rather than testing specific hypotheses, the objective is to provide a reliable measurement instrument. Using millions of LinkedIn profiles, a multi-stage Python-based data pipeline is implemented. It includes the preprocessing of raw files, the restructuring of nested job histories, industry filtering, horizontal (functional) categorization, and vertical (hierarchical) classification. Expert interviews from both industries support the development and validation of the two industry-specific classification systems.

This thesis mainly contributes by extending hierarchical measurement beyond start-ups of video games to two structurally complex and economically significant industries. Additionally, it introduces horizontal categorization as a necessary complement to vertical classification in industries with pronounced functional heterogeneity, particularly aviation. At last, it establishes a scalable empirical basis for future research investigating how hierarchical depth relates to

organizational outcomes such as but not limited to performance, coordination efficiency, innovation capacity, and power distribution, among others.

The structure of this thesis looks as follows: Section 2 reviews the literature on hierarchical depth, power disparities, and organizational design. Section 3 outlines the methodological framework and coding logic in detail. Section 4 presents the empirical results and the fifth section discusses how the measurement of hierarchies compares to Lee's (2021), scientific implications, further possible extensions of the analysis of a large dataset and limitations. The last section concludes before – as a matter of transparency – in the annex the transcribed expert interviews and the Python codes can be found.

Note: Artificial intelligence tools were used to facilitate complex coding procedures and to check spelling and grammar.

2. Literature Review

To organize a company, drawing an organizational chart is far from enough and in no way a trivial task that simply includes connecting boxes with names. It depicts the strategy of a company, and the way it is structured substantially influences its future success by creating the foundation of achieving company goals. The ideal structure helps to find the ideal balance between exploitation and exploration (Csaszar, 2012).

Companies can be structured in a variety of styles with the two most extreme forms being a self-managing organization, a so-called holacracy, that has radically eliminated reporting duties between superiors and subordinates (Lee & Edmundson, 2017); on the other end of the spectrum there is the fully authoritarian enterprise in which one single person has the entire command. In between, the horizontal and vertical differentiation may vary; both strongly depend on one another according to Van de Ven (1976). The latter depends on a company's strategy, goals, and capabilities (Burton & Obel, 2004). Adding to this, one of those capabilities is to make various agents operating within its boundaries to collaborate in a way that their individual outputs contribute in the most efficient way towards performance targets (Puranam, 2018).

Some believed that tall hierarchies have lost their appeal and would be marginalized by flat, egalitarian ventures (Sutton & Rao, 2014). Particularly start-ups should be flat; a belief that was based on several success stories like GitHub or Valve (Burton et al., 2017; Puranam & Håkonsson, 2015). However, the reality cannot be described in a single trend, and a flat hierarchy serves no self-purpose. Furthermore, merely eliminating hierarchical layers does not make a company flat; it requires fundamental changes in delegation (Reitzig, 2021).

2.1. Conceptual Foundations of Hierarchical Depth and its Relevance for the Airline and Car Manufacturing Industries

To investigate in-depth why companies outperform others, the hierarchical depth may pose an important factor. Lee (2021) has successfully mapped hierarchies with a large set of data, including job titles, for

start-ups in video games. While many companies are obliged to publish their financial statements, companies hardly ever publish organizational charts. By expanding Lee's (2021) model the possibility arises to shed light on the hierarchical depth of further industries and, at a later stage, determine the ideal hierarchical depth. "Ideal" may ensure the highest profits or outperforming competitors in other measurements of success.

In the following work the two investigated industries will be the automotive and the aviation industries. Particularly the airlines are notoriously difficult to operate profitably, but also OEMs have reported declines in profit margins in the past years (Damodaran, 2025; Stricker et al., 2025). Applying Porter's Five Forces framework, the dynamics become apparent when considering the competitive rivalry and buyer power (Porter, 1979). Both industries have seen major consolidations and mergers to gain profitability from economies of scale, synergies, and lower competition (Orchinik & Remer, 2023; Serrano, 2025). Hence, an agile organization might have the power to differentiate from its competitors which mostly all do the same thing: either transport people and freight from A to B in metal tubes or constructing metal chassis with wheels which also transport people and goods from A to B, all by relying on the same kind of fuel. Given the tremendous size of both industries with global revenues in the trillions, there is reasonable interest to find out whether hierarchical depth plays a role in their success (IATA, 2025; IBISWorld, 2024).

An explanation why this had not been done before is the difficulty to gain large data sets of employees as well as measuring hierarchies from the outside (Burton et al., 2019; Keum & See, 2017). There is no univocal measure of hierarchical depth; instead, the problem can be approached from different perspectives. For instance, Keum & See (2017) describe that besides looking at organizational charts, the number of "yeses" until a decision is made is a way of measuring the depth. The researchers also point out how difficult it is to measure the number of ideas that have never reached the surface, since those barely ever make it into any report. Furthermore, the two authors describe the span of control as a potential way of measurement, i.e., how many subordinates report to a single superior. The latter can be increased by standardizing work processes to reduce the coordination effort, thus cutting the time needed between manager and subordinate as well as diminishing the number of manager interventions. This

also explains why highly standardized operational jobs, also in the aviation and automotive industries, usually see much greater spans of control than, for instance, an R&D department within the very same corporation. However, besides standardization, the management style as well as the support a manager receives from his/her company in leading his/her own team, are two more influential aspects on the span of control (Burton et al., 2020).

Mintzberg (1989) introduced an exceptionally interesting theory about how power is not distributed by coincidence within a company but there are various streams that pull into different directions, i.e., span of control, standardizations of skills, outputs, magnitude of formalization, etc. Those aspects must be carefully designed as they depend on one another. Speaking of the various streams, he argues that the operational core favors professionalization and the standardization of skills, such as pilots' professional flying expertise, which defines how tasks are to be performed and thereby strengthens their positions in wielding influence over managerial decisions. Those middle managers, of course, aim for the most possible autonomy granting them maximum power over their own departments. Looking further up to the top management, the rationale is centralization, allowing them to exert maximum influence over the entire company. Technocrats like analysts and planners will advocate for formally designed processes with standardized output, giving them, as process architects, the greatest influence and power of control. On the other hand, support functions like HR and IT endorse coordination, since their power lies within expertise rather than formalized power. At last, people in favor of standardization of norms want all employees to be sworn in to the same goals and norms which is a foundation of very flat collaborative communities, like open-source projects (Burton & Obel, 2018).

2.2.Hierarchical Power Disparities and Their Consequences

While hierarchies can solve complex tasks (Zhou, 2013), they do not inherently perform better. Many hierarchies are even just maintained because of habit or due to unknown alternatives (Anderson & Brown, 2010). Greer et al. (2018) add to this that on average, hierarchy is negatively correlated with team performance plus its viability. Employees generally see steep hierarchies as a demotivator, making them more likely to leave their company (Anderson & Brown, 2010; Wassermann, 2012). Moreover,

Kilduff et al. (2016) find a substantial discrepancy between workers' actual position in the hierarchy and the status they believe their efforts warrant. Claiming *their* places reduces the overall team output, particularly if they see chances to advance within the hierarchy (Greer et al., 2018). Furthermore, workers at the lowest level in the hierarchy often feel underrepresented and unheard, which reduces their level of personal dedication to their work (Cowherd & Levine, 1992). Plus, they fear that someone else higher up will gain the laurels for their efforts (Pfeffer & Langton, 1993).

Barnard (1964) even points out that subordinates often overestimate the competence of their leaders and weigh their bosses' potentially wrong opinions and assessments more than they should. This happens even though the capability of the person in power to put themselves in their subordinate's shoes reduces in steep hierarchies (Galinsky et al., 2006). In an experiment by Tarakci et al. (2016) the results show that only 55% of the teams with a high power disparity initially choose the most competent person as their leader or as Anderson & Brown (2010, p. 16) state: "[…] groups often fail to select benevolent, collectively-minded individuals to lead them, and even might systematically select selfish individuals." The wrong person on the top strongly deteriorates the overall performance since they follow the wrong goals and alternatives are suppressed.

Strictly lived hierarchies can even pose concrete safety threats as Milanovich et al. (1998) found out; in 80% of civil aviation incidents, the co-pilots did not want to question wrong decisions made by their captain. Additionally, overconfidence leads to further costly commission errors, for example through unprofitable acquisitions (Malmendier & Tate, 2008; Cheng, 2007). Adding to this, in hierarchies dominant alpha characters often beat their more intelligent, wiser but calmer counterparts when it comes to climbing up the career ladder (Anderson & Berdahl, 2002; Anderson & Kilduff, 2009). Tarakci et al. (2016) sum up these aspects by arguing that complete inequality is impossible to achieve. Yet, highly skilled teams with little power disparity perform significantly better than if the wrong leader is on top of a strict hierarchy.

2.3.Hierarchies, Information Flow and Idea Distortion

Hierarchical organizations tend to generate homogeneous groups, as managers often hire and promote individuals with similar educational backgrounds and career paths. This reduces the diversity of perspectives and limits idea generation (Kossinets & Watts, 2009). In addition, hierarchies act as highly selective filters: while they can improve selection, they also limit the number of ideas that are voiced. Each additional hierarchical layer decreases information flow by roughly ten percent, and employees often self-censor as they fear rejection or loss of control over what was once their idea (Reitzig & Maciejovsky, 2015). Moreover, in hierarchical settings, dominance – communicated both verbally and non-verbally – can outweigh competence in discussions, discouraging employees from sharing potentially valuable ideas. Tost et al. (2013) show that even though managerial awareness reduces these effects, it does not eliminate them entirely.

Therefore, it seems apparent that fewer hierarchies stimulate informal communication, which is beneficial to idea generation. Nonetheless, filters are less structured compared to hierarchical settings; consequently, which ideas will be realized is more a matter of chance – rather than strategy. At the same time, if an idea must pass through multiple hierarchical layers until implementation, it automatically gets repeated inaccurately and often misinterpreted until it reaches the final decision-maker, costing time and jeopardizing the original initiative (Csaszar, 2012; Reitzig & Sorenson, 2013). Adding to this, long discussions and internal approvals make companies less responsive in their way of finding the *right* projects, even though in a fast-paced world, agility and innovation often outperform scale and stability (Van de Ven et al., 2013).

2.4.Implementing Flat Organizational Structures

After many preceding pro-flat arguments, it must be mentioned that going flat is not an end in itself but a company must make and follow this *strategic* decision in order to reach certain goals. Hence, the strategy must be aligned; just changing organizational charts is certainly not enough (Reitzig, 2021). While not every enterprise may be suitable for going fully flat, even if at least one of the following five aspects is changed in comparison to a classic hierarchy, some sort of novelty is created which can

provide a strategic advantage: Task division (splitting up work into processable chunks) as well as the task allocation of those work packages can be decided among peers rather than by a manager. Next, reward distribution can be peer-reviewed in preference to a manager's decision. Moreover, information exchange, which hierarchical organizations funnel through controlled channels and are thus pivotal for the organizational steering, can occur in a lateral and informal way (Puranam, 2014; Reitzig, 2022). Here, novel communication systems must substitute the previous hierarchies to make sure all employees have access and gain access to all relevant information. This is even more crucial for newly established firms as human resources are their major capital (Lee, 2021). Conveniently, new information technology not only lowers communication costs but also provides ways for structured approaches and real-time control through process mining. This facilitates the real-time visualization and consequently also gives a holistic picture of the possibilities of task disaggregation as well as aggregation (Kretschmer & Khashabi, 2020). At last, exception management poses a major challenge. Hierarchies do have a "stop-sign" to end conflicts – at least on the surface –, while flat organizations might struggle to find a consensus through "lateral escalation" (Puranam, 2014). This becomes evident as the number of potential conflicts rises quadratically with the number of employees. Either way, for managers, going flat represents a shift in "authority"; away from designing and assigning tasks to becoming an architect of collaboration.

"The right question may not be whether one should dismantle the hierarchy or not, but instead where to do so (and how much), […]" (Puranam, 2014, p. 17). "Too flat" can leave the top management overwhelmed with too many decisions resulting in a decision gridlock. The organization must fit the business environment, internal norms, the strategy and must be adapted if it no longer fits (Burton et al., 2002). Hierarchical depth is not set in stone but is a constant development process (Van de Ven, 2013). There is no need to completely abolish hierarchies (Puranam et al., 2014) and not everything must be rethought. Already novel approaches to one of the five previously mentioned aspects can exert substantial impact (Puranam, 2014).

2.5.The Organizational Ambivalence of Hierarchy

Several studies have shed light on pros and cons of hierarchies, which will be briefly described in the following paragraphs. Lee (2021) provides evidence that while more hierarchical layers do slightly reduce idea creation, they also lead to higher commercial success. Keum & See (2017) particularly focused on the aspect how ideas find their way through hierarchies and flat organizations. While hierarchies act as a strong filter and choosing the "right" project is a core competence, the authors also state that they hamper creative heads from speaking out their most radical new approaches, fearing they might not be taken seriously. By contrast, in very flat structures, employees tend to exhibit a so-called "self-promotion bias", whereby they push their very own ideas while irrationally denigrating those of peers, which jeopardizes a key advantage of flat organizations (Van de Ven et al., 2013). Importantly, this behavior can be prevented through the presence of persons of authority (Keum & See, 2017).

Beyond differences in idea generation and selection, another source of ambiguity between flat and hierarchical structures concerns how organizations deal with different types of decision-making errors, i.e., their risk of committing omission and commission errors. The former describes the opportunity costs when missing out on a potential gain, the latter refers to the costs of a decision that should have been avoided. Large companies with lots of hierarchical layers and hence also lots of filters are – perhaps counterintuitively – not more prone to omission errors, since they have enormous resources to explore in various directions while maintaining a comprehensive oversight through professional portfolio management. This is particularly applicable if there is effective communication between managers to share ideas.

Anyhow, the cost of commission errors should define the number of decision makers as Csaszar (2012) references the example of a court, where commission errors have the highest possible costs. Consequently, many people are involved before a verdict is announced. Although there should be more than just two eyes, Eisenhardt (1989) points out that in very authoritarian structures commission errors often last longer, since no one dares to speak out the unadorned truth to their CEO after this person took that decision alone. The opposite is the case: often the information transported upwards is either simply

wrong or fundamental details are deliberately concealed. Moreover, authoritarian structures often limit the downward information flow to cement power, but this exacerbates commission errors, since employees cannot understand why things go wrong or why a decision was made (Liao et al., 2010). By contrast, commission errors are usually faster to correct. An unprofitable route can be terminated quickly; a flopping car model may swiftly be discontinued. However, not investing in a major new business opportunity can pose an existential threat (Bouwman et al., 2014).

2.6.The Trade-offs Between Routine, Innovation, and Control

Taken together, the aforementioned findings suggest that neither flat nor hierarchical structures prevail across all dimensions but rather address different organizational problems. Hierarchies are particularly good at following routines and at avoiding total flops. By contrast, hits are rarely produced since radical ideas do not make it through the layers. Moreover, flat structures do not just generate hits but because of their flexibility, they are better at coping with complex, novel tasks. Nonetheless, scientists point out that a hierarchical process should be implemented for the decision-making, to avoid self-promotion and find the best idea among all (Keum & See, 2017).

This ambivalence is further reinforced when considering the role of informal communication and relationships. These two important yet often overlooked aspects are particularly utilized by top-level managers to entrench their power. Moreover, formal and informal communication cannot be treated apart from one another since they are deeply intertwined. Across all hierarchical levels, informal communication and relationships can ease the weaknesses of formal structures, overextend them by turning even simple tasks into collective endeavors, and balance them by enabling horizontal and lateral connections. At the same time, they may also sabotage formal structures through hidden networks that are notoriously difficult to detect, let alone to dissolve (McEvily et al., 2014).

These tensions are particularly visible in organizations that have either experimented extensively with flat structures or still stick to them: GitHub used to be an organization without any classical hierarchies. Two employees were sufficient to initiate a project. Their happiness was deemed the most valuable aspect to make them productive. Conflicts were to be resolved through consensus. However, as more

employees increased the chances of disputes, finding a consensus became increasingly unlikely. This became especially visible around 2013/14, when GitHub had already grown far beyond its early start-up phase during which the flat structures succeeded. Coordination across projects became more demanding, and decisions increasingly needed to be made faster. At the same time, the company faced a changing market environment, in particular rising expectations from corporate customers and external stakeholders. Under these conditions, purely consensus-based decision-making began to slow down the organization. In response, GitHub did not abandon its core principles, but gradually introduced classical hierarchical elements, including formal leadership roles and clearer decision-making authority to ensure smooth coordination and faster decision-making processes (Burton et al., 2017).

Puranam (2014) investigated the example of the US-based video game developing company VALVE with more than 400 employees which shows that being flat can attract employees that are capable of self-initiation of creative projects – all without any fancy job titles. Nonetheless, an interviewed employee stated that "Hierarchical management bottlenecks innovation through the people at the top of the hierarchy." (Puranam & Håkonsson, 2015, p. 2) Despite many success stories, the risk that employees look out for "safe" projects still prevails as taking up riskier projects that might fail can consequently cost them their peer-reviewed bonus. Moreover, employees can still be overruled by their own peers since the owner does not interfere in daily business. That, however, does not feel better than a rejection by a superior (Puranam & Håkonsson, 2015).

Looking at these two real-life examples, one can sum up that self-managing organizations thrive in environments without fierce competition or in separate departments where real-time information flows are not needed. In certain industries like video game development, flat hierarchies are the standard rather than the exception (DeSantola & Gulati, 2017). Nevertheless, in cases of strong market pressure, hierarchies can realize efficiencies (Burton et al., 2017). This tension reflects a conflict of objectives inherent in organizational design (Burton et al., 2017) or as Gibbons (2005, p. 206) states, "The cost of control is the loss of initiative".

2.7.Hierarchies as a Coordination Mechanism

Hierarchies are not a clean-sheet design but must fit the organizational environment and corporate culture in which they operate. As Joseph et al. (2018) point out, it is illusory to assume that hierarchies are purely rational arrangements. Nevertheless, they serve central functions, particularly in structuring processes and allocating resources. There is no ideal depth of hierarchy; instead, organizations face a continuous challenge of finding the best fit for their current context (Van de Ven et al., 2013). Even completely central coordination is justified when its benefits outweigh the costs (Hart & Moore, 1999).

One key reason why hierarchies persist lies in their essential coordination mechanisms for highly complex tasks that are difficult to decompose. In this sense, organizational structure helps connect interdependencies while at the same time separating activities that do not require close coordination (Zhou, 2013). Self-evidently, more modular tasks can be managed with fewer layers, still, full holacracies often struggle with coordination and information processing, and with uncoordinated information flows (Ferro, 2016). Real-time KPIs and their monitoring are another advantage of bureaucratic systems, enabling a variety of quick responses to upcoming problems which can be quickly identified (Eisenhardt, 1989).

Research further suggests that within flat organizations, conflicts are often "resolved" through dominance or personal attractiveness rather than competence. This leads to productivity losses and as the focus shifts away from core activities (Greer & van Kleef, 2010). Decision-making may be reduced to the lowest common denominator, as consensus becomes harder to achieve with multiple equally leveled employees involved – but in the absence of clear authority (Tarakci et al., 2016; Meier et al., 2019). Hierarchies can avoid such minimal consensus by efficiently managing exceptions. Galbraith (1974) argues that exceptions should remain exactly that – *exceptions*. Most organizational processes function without managerial intervention. Hence, hierarchy should be activated only when problems arise to avoid unnecessary escalations. Routine decisions should therefore be made at the lowest possible level. Galbraith emphasizes that task forces with lateral collaboration relieve the hierarchy.

Rather than being purely bureaucratic, hierarchies can provide light but effective guidance. Clément and Puranam (2017) argue that networks tend to emerge within organizational structures but decay without them. A light hierarchy can offer structure and orientation without becoming overly bureaucratic. Given the results of Slade Shantz et al. (2020), a certain structure helps, because even when formal hierarchies are minimized, informal hierarchies tend to emerge. These are often based on status, sympathy, or personal networks rather than formal responsibility. Such informal structures can disadvantage capable but introverted personalities and promote "I-thinking" over "We-thinking" (Slade Shantz et al., 2020). Start-ups may even benefit from some hierarchical elements, as clear roles and responsibilities help define ownership and accountability (Savage, 2015). To safeguard against opportunistic behavior in hierarchies, Burton & Obel (2018) advocate for a multidivisional rather than functional organizational structure.

2.8.The Case of Start-Ups

Start-ups are one type of business for which data is particularly limited. So far, only limited research has been done (Burton et al., 2019; Keum & See, 2017). Start-ups face a major difficulty: while growth at the beginning is very important, it is equally challenging. A start-up tries to be innovative; thinking about organizational charts is not something entrepreneurs usually pursue as their primary task. The challenge of finding a balance between growth and internal organization is huge (Bhidé, 2000; Baron & Hannan, 2002; Davila et al., 2010).

Yet, if done right, entrepreneurial ventures can vastly outperform established firms in terms of growth (Kirchhoff, 1993). Moreover, due to their mostly innovative character, they can profit from a higher willingness to pay by early adopters. This underlines the necessity to innovate and explains an increasing disparity between top and flop, with little being left in the middle (Rietveld & Eggers, 2018). What must be considered is that growth is multidimensional, for example, the number of customers, financial funds, production capacity, geographical outreach, revenue, employee count, but, of course, also profit (Josefy et al., 2015).

The shape of a start-up in terms of organization at the very beginning has a long-lasting influence on the future growth path of the venture (Beckman & Burton, 2011). Self-evidently, the initial business environment forms the behavior and structure (Stinchcombe, 1965). Growth is so eminently essential for start-ups because the competition is usually intense in newly emerging market environments. All strive to be highly innovative and fast-moving, resulting in fierce competitive pressure. Not growing at the beginning poses the risk of sinking into insignificance (Rindova et al., 2010). Thus, coping with internal and external growth concurrently forms a major capability (Hiatt & Sine, 2014) but is indeed a key structural dilemma (Bhidé, 2000). Not being able to do so causes many "early deaths" of start-ups within the first five years (Dahl & Sorenson, 2012).

Although management systems do not just add bureaucratic hurdles but can boost growth (Davila et al., 2010), start-ups often introduce hierarchies *reactively* after a crisis and usually only *proactively* if the founder had some experience in large corporations and she/he has hence internalized hierarchical systems. Sometimes an external motivation is also the reason as company partners often want to speak to the Vice President (Van de Ven et al., 2013).

After having achieved successful initial expansion, founders of start-ups often struggle to step aside, handing over the steering of their company to an experienced manager. Personal management style, however, clearly hits its limits, particularly if a company reaches a size of 50-80 employees (Davila, 2005). According to Boeker & Wiltbank (2005), at this moment in time, founders should depart from general management to allow for more professional management. Venture capital firms frequently complement founders by offering the introduction of professional management structures (Burton et al., 2019). Assisting them with managers frees their time allowing them to focus on research and development. Nonetheless, for this "assistance" to be successful, founders must be open to a change in management and hand over the reins. Noteworthy, Lee & Kim (2024) indicated that an early shift within the first few months to professional management negatively influences the success of a start-up. This might seem counterintuitive given the advanced skills, however professional managers are very expensive for small ventures, plus they make them inflexible and less innovative since managers tend to be exploitive rather than exploratory. Therefore, they usually commit to the first ideas and defer

novelties, since they try to earn money with what already exists. A U-shape in terms of exploration has also been described for larger firms by Peretti & Negro (2006). Young employees and high-level managers are the most innovation-driven ones. The first group is brave, the other faces less objections than middle managers.

Besides the growth topic, hierarchies are not necessarily negative for start-ups. They can function as a protective skeleton that offers structure and guidance. Despite that, formal structures are often not widely accepted internally, also because authority has not settled yet. Hence, the hierarchy works less efficiently due to the lower internalization and consequently a limited assertive power. For that reason, Stinchcombe (1965) speaks of a "Liability of Newness". Also, he states that hierarchies are no clean sheet design for start-ups either, but are shaped by dependencies, not just efficiency alone.

Baron & Hannan (2002) describe 5 different leadership models of start-ups. The "authoritarian" model, where the founder directly monitors his/her employees, the "stars" model in which highly skilled workers are motivated by autonomy and the "engineering" model, where the work itself is so motivating that employees challenge and control themselves as peers. Furthermore, the authors describe the "commitment" model, where the personal commitment of the workers motivate them in their work as well as the "bureaucratic" model. There, clear formal structures provide guidance and control work processes. What is worth mentioning is that the latter model requires three times more middle managers than the commitment model, since that model features internalized goals and employees who have committed themselves to them. While this is more flexible, it can be bothersome for future growth compared to the bureaucratic model as a change in the model particularly alienates long-term employees. Hence, the "blueprint" of the founder has a long-lasting influence, even long after he/she has left the company (Baron & Hannan, 2002).

Another decision from the beginning with an influence far into the future is the remuneration. An "egalitarian" remuneration model can make young companies laggy and this condition is difficult to change later due to opposition of the employees (Burton et al., 2019).

3. Method

The method used to answer the research question *"How can hierarchical depth in airlines and car manufacturers be measured effectively?"* is a rule-based code. Rather than merely scanning the LinkedIn dataset, the Python scripts systematically preprocess, filter, and parse the data to enable hierarchical analysis. Profiles are mapped horizontally to primary departments and, most importantly, classified vertically according to their position within the chain of command. Transforming raw data into a finalized, verified master spreadsheet required several steps. Otherwise, intermediary checks and numerous iterations would not have uncovered substantial flaws in the implemented code. Despite the sophistication of the script, industry expert interviews were a key component in determining an accurate hierarchical classification system. Notably, the two Python pipelines share similarities; nonetheless, they are clearly no copies from one another merely with replaced job titles given the differing nature of the input data as well as structural differences between the two industries. Hence, some information on the automotive industry might be repetitive but it is preferable that both methodological sections can be read and understood independently.

3.1.Interviews

To get in-depth insights into both industries, airlines and car manufacturers, experts were interviewed. These form the basis of the Python code and enabled the building of a hierarchical framework, where each job title description is assigned to the correct level of hierarchy.

As agreed, no semi-structured interview guide was used to explore the departments of the individual interviewees in greater depth. That yielded the opportunity to receive more precise information for both operational and managerial jobs. The goal was to learn about all the job titles within the interviewee's line of command as well as from other departments if the interviewee possessed the relevant information. In addition, information about the general setup of the various organizations was particularly valuable and sharing the information was highly encouraged.

All but two interviews were conducted in German, but the interviewees were given the choice. One interview was held in Dutch, the other one in English. The interviews lasted between 15 and 45 minutes

and were recorded for the sole reason of enable subsequent transcription which can be found in Annex B: Transcripts. The interviewees were made aware of this before the interview. All agreed but requested anonymity to avoid disciplinary action and inadvertently revealing company details. Afterwards, the German and Dutch interviews were translated and transcribed, as can be seen in the annex. When finding interview partners from airlines I benefited from a professional network through my job, however I emphasized that this work is entirely independent of my employer, even if I used the IT infrastructure for some interviews, i.e., Microsoft Teams.

3.2. Online Research

Despite conducting nine interviews, minor ambiguities remained within the hierarchical framework, particularly in the automotive industry. Online sources like theorg.com, zoominfo.com, and were used to supplement missing information, however, the data there does not always match the content of LinkedIn profiles. Furthermore, some information could not be verified with privately obtained organizational charts. Consequently, commercial websites only offered limited value for the development of the rule-based classification system and were only used to bridge residual gaps or to find out, which titles usually describe the same positions or professional roles with comparable hierarchical levels (Apollo, 2026; Orgio Inc., 2026;zoominfo, 2026).

3.3.Airlines

The complexity and the large-scale initial dataset volume of around 160 GB required a multi-stage process. Each of these processes is described in detail for comprehensibility and transparency. The five main parts are: preprocessing, reshaping, filtering, horizontal categorization and vertical categorization. All of which can be seen in Annex A: Python Codes.

3.3.1. Preprocessing (airline)

The initial aviation data set comprised 40 .csv-files with a size of around 4 GB each. After loading the files into the working directory and creating an output folder, the core part of this step began using the Pandas library:

Out of 117 initial columns only 18 were kept for potential further analysis. While it was apparent that not all of the 18 columns would be used in ensuing steps, keeping more to be on the safe side was preferred. These are the columns retained:

> "id", "current_company", "education", "experience", "position", "experience_company", "experience_industry", "experience_title", "experience_duration", "crunchbase_company", "num_employees", "type", "crunchbase_industries", "contacts", "current_employees", "num_contacts", "legal_name", "industry_dummy".

Furthermore, this Python script started with an initial cleaning of the data. It removes null bytes, invisible to humans but not to computers, as those could end the analysis abruptly. It also deletes replacement characters, often symbolized by a black diamond shape with a question mark in it. This ensures Microsoft Excel does not crash or show those signs which cannot be analyzed properly. For the same reason, line breaks within a cell are replaced by a simple space, so that line breaks do not unwittingly overwrite information in the line(s) below.

Last but not least, all .csv-files were transformed into .xlsx-files which was facilitated using the OpenPyXL engine.

Afterwards, all 40 files were saved in a newly created folder. Each file was marked with a clear indicator to distinct new files from old iterations by adding individual time stamps: YYYY.MM.DD.HH.MM.

In total, the first step reduces the file size of the 40 initial files to approximately 3% of their original size, i.e., roughly 120 MB per file.

3.3.2. Reshape (airline)

The "Reshaping" step aims to extract data from so-called JSONs (JavaScript Object Notation), a data format that nests data into a single cell, figuratively comparable to a Russian Matryoshka doll, where one doll is nested into a slightly larger one. Therefore, the script needs to extract the nested data to analyze it at a subsequent stage.

To do so, the script reads the file paths of the previously preprocessed .xlsx-files from a .txt-file. It utilizes the Pandas library to load each preprocessed Excel file into a high-speed data structure called a DataFrame. As a next step, duplicates of the IDs are eliminated. Only one single line per ID is retained. This constitutes an essential prerequisite before the data transformation, since the data was stored in multiple lines with a logic comparable to the game "I'm going on holiday and I'm bringing …". More specifically, the lowest line of each ID only included the last job experience, in the second lowest line the last and the second-to-last experience, etc. Therefore, the coding will select the line, which includes *all* previous job experiences and omits all others where data is incomplete.

It transforms the nested JSON data into a flat structure. Thereby, the data is broken down into six columns:

exp_1_title
exp_1_company
exp_1_start
exp_1_end
exp_1_industry
exp_1_duration_years

There is a cap of maximum 30 experiences to be flattened. The script sorts these six column blocks side-by-side, starting with the newest experience, which is the most relevant one for future analysis. Additionally, the Python script intelligently cleans the data using Regular Expressions (Regex). It fixes

common formatting errors in the JSON text, such as missing quotes or incorrect brackets, which would otherwise prevent the data from being read.

Moreover, as a possible extension for future research, the script also adds a "total_experience_years-column", where all experiences of the individuals are summed up by subtracting the start date from the end date. However, many entries had two simultaneous experiences. The prevent someone from having, e.g., >100 years of experience, whenever a new job was commenced, the previous one was truncated. For instance, someone has worked from 2021 until 2023 as a consultant and from 2022 until now as a senior consultant, then the code changes the end date of consultant to the end of 2021, to forestall an overlap and thus double counting of the years 2022 and 2023. Jobs without a time stamp are always omitted.

At last, the XlsxWriter engine was utilized for the final data export because of the extreme width of the spreadsheet with almost 200 columns. The data of all 40 reshaped Excel input files are saved in another output folder. Deleting the unnecessary lines and transforming the JSONs reduces the data volume of all 40 files by approximately another 65% to around 40–50 MB per file.

3.3.3. Filter, Merge, Delete (airline)

The following step comprises three sub-steps which accurately filter unnecessary data. For this purpose, the preprocessing and reshaping act as a precondition that ensures that the filter can function as intended. Only after the filtering step is it reasonable to merge all 40 preprocessed, reshaped, and industry filtered Excel files into one Excel master file. Finally, further clearly unwanted data entries that have passed through the industry filter but cannot be analyzed any further are omitted to provide a "sanitized" spreadsheet for the final two major steps.

3.3.3.1.Industry Filter (airline)

Firstly, the script uses another .txt-file with all input paths of the previously reshaped Excel files. As an initial step, the company names are normalized to lower-case letters only, and suffixes like "Ltd." or "Corp." are stripped utilizing Regular Expressions (Regex).

Next, fuzzy matching is applied with a threshold of 90% against the whitelist. The pipeline employs the "thefuzz" library, which is based on the Levenshtein Distance algorithm and also finds near-perfect matches like "United Airlins" instead of "United Airlines".

The whitelist of airlines can be found in the annex under chapter 0. Moreover, an industry check is executed, controlling whether the industry contains "airlines", or "aviation". The whitelist and the industry filter act as an "or-filter", so the entries must comply with at least one of two criteria.

As a last filtering step, a blacklist is applied which is absolute. Any entry with an airline name that is on the blacklist as the most recent experience is omitted. The blacklist allows omitting non-relevant companies, including bankrupt or merged carriers like Pan Am or US Airways, as well as aerospace manufacturing and maintenance firms, thereby narrowing the focus to active commercial airlines.

This step reduces the size of the 40 files once again. New file sizes range between 0.2–1.2 MB, corresponding to a data reduction of well beyond 90%.

3.3.3.2.Merge (airline)

This comparatively simple step aggregates the 40 preprocessed, reshaped, and filtered files to a single Excel master file. By using Pandas' concat function it stacks all the Excel files from the input folder on top of one another. Not only does this function prevent the repetition of headers, it also re-indexes the entire dataset into a single numerical sequence to facilitate future analysis. Furthermore, the script indicates potential faulty files with an error message. Therefore, if a single file might be unreadable and thus cannot be included in the Excel master file, transparency is given.

This consolidated master file has a size of roughly 13.8 MB.

3.3.3.3.Delete (airline)

After reviewing the merged master file, it became apparent that a further deletion of unwanted entries must be executed before moving on to further analysis. This intermediary step ought to precisely remove data which was previously accepted by the filters but disqualifies them from further analysis. This

ensures that the master spreadsheet is as free of inconsistencies as possible. To reach this goal, the script uses “Deletion Patterns” powered by Regular Expressions (Regex) to identify non-eligible rows. This includes temporal exclusions like “former”, “retired”, “future”. This can be exemplified by “former manager” or “retired pilot”. Also, occupations like actor or musician sometimes make it through the filters, if the specific person marked their job as being part of the airline industry and must be excluded. Furthermore, job seekers like “future pilot”, “aspiring flight attendant”, or even personal titles like “flying mom”, etc. are excluded. If no job title can be found the entire line is deleted. After the deletion, the script shows how many entries are eliminated.

The deletion reduces the file size from 13.8 MB to ca. 12.8 MB, so by approx. another 8%.

3.3.4. Horizontal Categorization (airline)

Before this major procedural advancement, the interviews were essential to develop an appropriate hierarchical framework that works for most airlines.

In general, this is the first step that aims to mimic a human HR specialist and horizontally categorizes job titles according to their functional departments. It flattens tens of thousands of job titles into dozens of categories. However, besides flattening, it does not yet level the hierarchies but aims to divide the structurally heterogeneous jobs of airlines into several core departments. These are: cockpit, cabin, technical & maintenance, ground operations and general management jobs, which cannot be attributed to a single operational department. In a new column, each line receives an additional entry with information about the core department they are assigned to.

For the robust categorization, it uses Regular Expressions (Regex), for example, both “captain” as well as “B757-captain” are categorized in the joint cockpit department. Moreover, an “ordered dictionary” ensures a specific sequence of keyword search: Operational jobs are prioritized over non-specific management roles. For instance, an engine manager gets assigned to maintenance, rather than to general management.

Furthermore, it already pre-analyzes hierarchical titles like director or vice president. While the script does provide a very rough leveling, it is not final yet. Instead, the purpose is to get a better overview of the data before the next step.

Additionally, the script manages to "categorize" job titles like "staff" or "employee" to an uncategorized section, since such titles do not offer any possibility for reliable analytical basis. Also, by context-aware Regular Expressions, it takes care of so-called negative lookaheads, for example, a potentially misleading title like "Assistant to the COO" who is clearly at a different level compared to the "COO". Among other numerous sophisticated logics of this almost 600-line script, it also manages to not assign specialized positions like a "duty manager" to general management but, with the help of industry experts, it was possible to correctly assign those workers to ground operations. One more example would be adding the "clean_text" processing layer to ensure that an engineer is never accidentally classified as an Engine Mechanic, although the first six letters are "engine–". This would negatively affect the upcoming hierarchical leveling.

The addition of this horizontal categorization increased the file size of the newly created Excel master file to 15.5 MB.

3.3.5. Vertical Categorization (airline)

The next and final step is the vertical categorization of jobs; in other words: assigning each individual a level within the hierarchical framework. Horizonal categorization is performed as a 4th step because hierarchical levels of, for example, cockpit, ground ops and management are difficult to compare and a one-size-fits-all linear hierarchy does not seem to be advantageous after consulting experts and incorporating personal professional experience.

To avoid losing the top-tier managers, level 00 (supervisory board), level 01 (CEO), and level 02 (EVPs and C-Suite executives) are vertically categorized right away. As written before, no assistant to COO will be captured here by means of negative lookaheads. Higher and middle management positions of levels 03 to 06 are analyzed next. Iterations showed that separating the core departments already at level

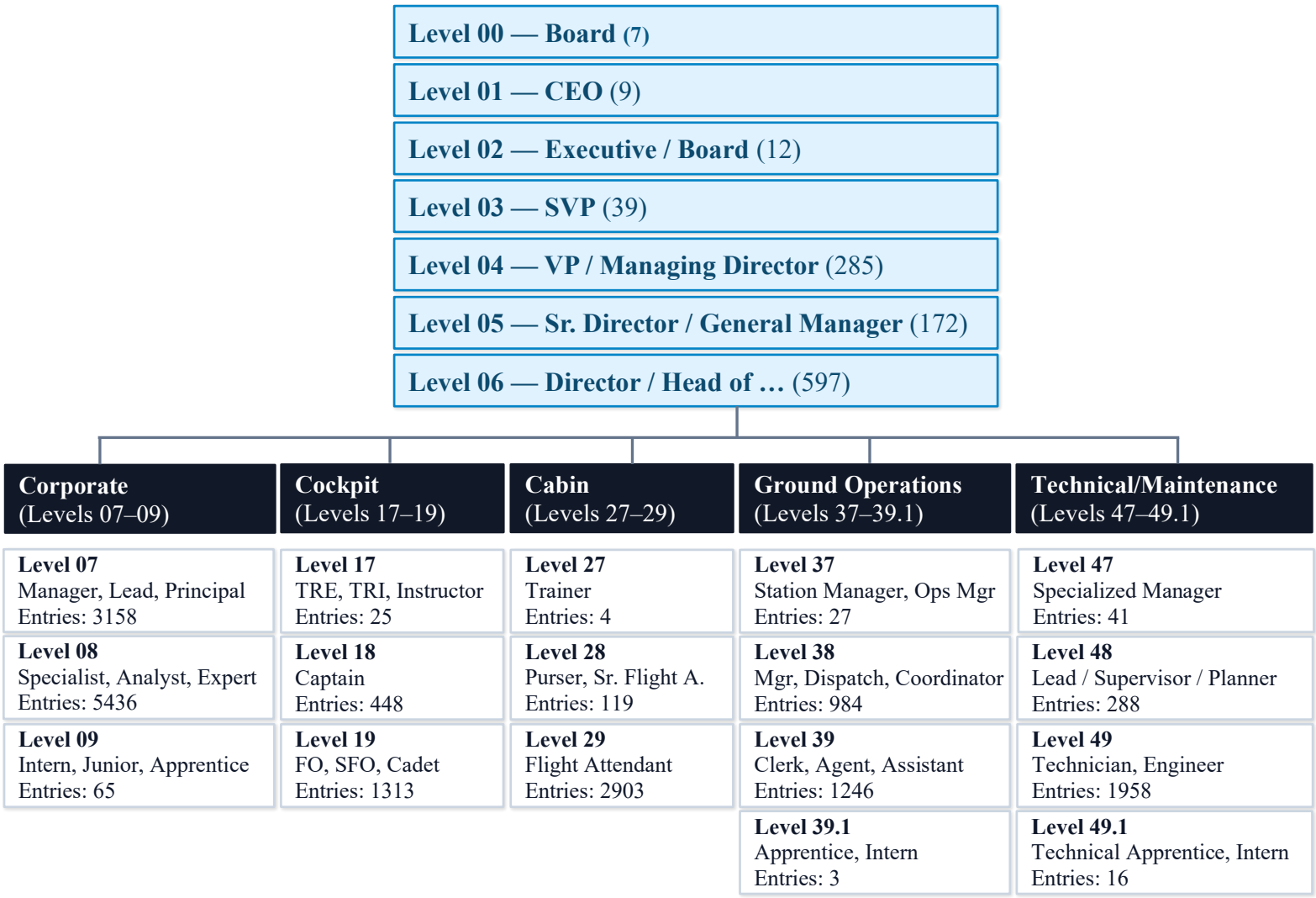
06, for example with a director of inflight, is not efficient and results in statistically insignificant clusters per hierarchical level. Since such a position clearly involves managerial duties, this logic is justified. Hence, level 07 marks the point at which the horizontal categories diverge.

Therefore, this script then investigates the previous horizontal categories in this order: cockpit, cabin, ground ops, technical/maintenance and, as a last step, the remaining general and lower management functions. After scanning each of the core operational departments, it assigns the respective hierarchical levels. For cockpit and cabin, three hierarchical layers turned out to be sufficient, those are 17-19 for cockpit and 27-29 for cabin. Ground operations as well as technical/maintenance require four levels, thus 37-39 plus 39.1 for ground ops and 47-49 plus 49.1 for technical/maintenance respectively. Of course, lower-level management titles are also sorted into levels 07-09.

Because of strict filtering and strict deletion in the third step of this Python-based data processing pipeline, barely any entries remain unclassified. The residuals, however, are assigned to level 13 and will not be analyzed any further (less than 2%).

An important aspect of this implementation is that even if a clear operational description like director of flight operations is present, it considers the hierarchical job description, which is more relevant for this work. Moreover, the previous horizontal categorization turned out to be particularly helpful here: a cabin lead, also known as purser, is on level 28, rather than 07 which would be the level assigned for a "lead" in management – thus one level higher. Also, a shift lead in maintenance is not the same as a team lead in corporate, as a shift lead usually does not have disciplinary power over employees in general, apart from the shift for which they work.

3.3.6. Airline Hierarchy Overview



3.4.Automotive Industry

While most parts of the analysis of the hierarchical depth are largely analogous to the airline industry, there are some differences because the initial data was provided in considerably heterogeneous file sizes. The horizontal categorization turned out to be less important when compared to airlines. However, this computationally complex component was kept for future analysis.

3.4.1. Preprocessing (automotive)

The first preprocessing script preprocesses all 1,200 files at once which were provided. Again, the files contain millions of scraped LinkedIn profiles with connection to the automotive industry. The size of the .csv-files ranged from 2.1 MB to 1.3 GB totaling 355 GB.

After importing the files and creating the output folder, the script reduces the number of columns from 137 down to 18, namely:

> "id", "current_company", "education", "experience", "position", "experience_company", "experience_industry", "experience_title", "experience_duration", "crunchbase_company", "num_employees", "type", "crunchbase_industries", "contacts", "current_employees", "num_contacts", "legal_name", "industry_dummy".

It was apparent that not all the columns would be used for the later analysis, but only columns deemed definitely dispensable should be deleted to mitigate the risk of redoing the preprocessing again as this step requires a processing time of several hours. Speaking of the processing time: to ensure this hours-long computation is not interrupted by a single corrupted file, the script includes specific exception handling to skip empty files or missing paths without crashing the entire batch. However, to preclude any crashes of the files in the future, the script checks if there are any forbidden characters in strings and removes them entirely. It also uses the low_memory=FALSE function which ensures that all columns are treated the same way by telling Pandas not to “guess” the file format chunk-by-chunk for the entire file. While using more RAM, this promotes a higher quality of the output data.

Additionally, the script immediately marks every output file with a complete timestamp of YYYY.MM.DD.HH.MM to reduce the likelihood of any overwriting and to facilitate the search of the newest available file after several iterations. All input files are also transformed into .xlsx-files using the OpenPyKL engine.

Due to the wide range of input file sizes, a reliable absolute size of the preprocessed files cannot be stated here, however, like in the airline script, the file size reduced to approximately 3% of the initial size. The largest 1.3 GB file was reduced to 36 MB.

3.4.2. Reshape (automotive)

This code imports all 1,200 files, defines an output folder, and adds a new timestamp to every reshaped file.

The most important column is the experience column where crucial data is stored as a JSON (JavaScript Object Notation). This format, as described in chapter 3.3.2 Reshape (airline) is computationally cumbersome to analyze since the entire lifetime experience of a person (job title, start and end date, industry, employer) is stored in a "Matryoshka-doll-style" within just one single cell which ought to be reshaped for a future analysis. To this end, the JSONs have to be checked for illegal characters which are corrected or replaced by spaces. Thereby, the particularly important job title is extracted. The same maximum cap of 30experiences is applied and the same columns are extracted:

exp_1_title
exp_1_company
exp_1_start
exp_1_end
exp_1_industry
exp_1_duration_years

This script applies the identical calculation of years of experience as in the 3.3.2 Reshape (airline) script. Likewise, each ID is allowed only once. Hence, it uses the same deduplication logic described in the corresponding script for airlines. This concerns profiles that have multiple lines whose only difference lies in the number of experiences. Therefore, the coding will select the line with the most job experiences and omit all others.

This step shrinks the size of the largest Excel file from 36 MB down to 16.8 MB – comparable to the respective airline counterpart.

3.4.3. Filter (automotive)

This filtering step is divided into two major intermediate phases due to the large number of files and a large amount of data that is not required for the analysis of hierarchical depth in the automotive industry. Moreover, multiple iterations demonstrated that this two-step sequence is more reliable.

3.4.3.1.Industry Filter & Batching (automotive)

This phase incorporates two major sub-steps within one script which is designed to batch the hundreds of files and to reduce the data to relevant industries as there is a lot of noise in the data, i.e., many entries are not associated with car manufacturers like restaurant employees. The first sub-step involves a hard-coded industry whitelist to systematically scan all the lines for industries such as Manufacturing, Automotive, Motor Vehicle Manufacturing, etc. In addition, a hard-coded blacklist is implemented for industries like: Hospitality, Insurance, Real Estate etc. Those entries are removed.

The second sub-step of this script consolidates the 1000+ files by always merging 50 files in one. This significantly reduces the number of files to make their handling easier.

Naturally, this step leads to a substantial decrease in data volume. After step 2 (reshape), 1,137 files totaled 4.45 GB. These are batched into 23 files with a cumulative volume of 382 MB, representing a reduction of more than 90%.

3.4.3.2.OEM-filter (automotive)

In the second intermediate filtering phase, the script will filter for profile whose most recent employer is car manufacturer (OEM). To do so, a list of the 90 biggest car producers was explicitly defined into the Python script. The latter does not require exact matching but uses fuzzy matching to prevent the omission of minor spelling mistakes. Moreover, it strips company names of their suffixes like “Ltd.”, or

“AG”. Thus, Volkswagen AG or Volkswagen Group are standardized to “Volkswagen”, which is included in the hard-coded OEM whitelist.

As an output, there are still 23 batched files totaling 52.6 MB, down from 382 MB (a reduction of 86%).

3.4.4. Horizontal Categorization & Data Reduction (automotive)

This part represents the first analytical step of the empirical process. Using the Pandas library, the script imports the previously filtered files to perform a multi-stage cleaning and horizontal classification.

The process begins with a “hard delete” of irrelevant profiles. Using Regular Expressions (Regex), the script scans the most recent job titles (exp_1_title) and immediately excludes lines containing keywords for inactive or non-professional roles, such as ‘student’, ‘retired’, ‘future’, or ‘unemployed’. A significant obstacle involved entries where users entered a company name (e.g., “BMW”) instead of a functional role like “engineer”. Because such entries provide no data for hierarchical analysis, they were systematically omitted.

The core analytical function of this script is the functional mapping of profiles into departments such as R&D/Engineering, Production/Assembly, Sales, Logistics/SCM, and HR/Recruitment, among others. The rationale for this step lies in the different vertical requirements of departments like R&D compared to Production/Assembly, albeit the interviews showed that the disparity is not as pronounced compared to airlines with four tremendously distinct operational departments. In addition, for car manufacturers, the only truly operative jobs are those in manufacturing. Notably, the workforce of Production/Assembly is often outsourced which restricts manufacturers’ direct supervisory authority over factory workers.

To address complex titles, a Functional Priority Logic was developed. Instead of a simple keyword search, the script utilizes a top-down hierarchy where technical pillars are prioritized over generic management keywords. For example, a “Senior Manager Engineering” is correctly captured within the Engineering domain rather than being lost in a generic management cluster. Every profile is assigned a new department attribute in a dedicated column; those who do not trigger a keyword match are labeled as “uncategorized.”

This stage successfully reduces the dataset to a manageable 29.2 MB by stripping auxiliary data and excluding analyticall non-viable entries, creating a clean foundation for the vertical hierarchy measurement.

3.4.5. Vertical Categorization & Merge (automotive)

Building on the departmental foundation, this stage introduces the core empirical logic: placing thousands of diverse jobs into their respective hierarchical position on a standardized "hierarchy shelf". To ensure absolute consistency across the entire population, the previously decentralized 23 processed batch files were programmatically consolidated into a single, unified master spreadsheet.

This consolidation is executed through a structured Pandas pipeline: all Excel files generated in the previous step are detected, imported into internal memory, and merged into a single unified table (DataFrame). This eliminates any inconsistencies between batches and guarantees that the hierarchical classification is applied to the entire population simultaneously.

The accuracy of this 13-tier ladder is driven by the following technical interventions: This script utilizes a mandatory clean_text preprocessing layer that standardizes job titles before classification. To clean data it converts common foreign terms, e.g., "ingénieur" and common typos like "ingenier" into standardized English keywords: "engineer". This correction enables the successful classification of many data points that previously slipped through the rule-based classification framework.

Furthermore, contextual intelligence and negative lookaheads have been incorporated: Simple keyword searches often fail on complex titles like "Assistant to the CEO." The script employs negative lookahead logic to distinguish between high-level strategic roles and their support staff. If an executive keyword is modified by terms like 'assistant', 'intern', or 'junior', the script automatically ignores the high-level match and continues searching for the correct lower rank. Furthermore, it filters out roles that include the keyword "management" but, according to the interviewed experts, whose roles usually do not encompass disciplinary authority. These include, for example, Parts Managers, Project Managers, or Process Managers, who are placed at level 10 based on the interview insights.

The script does not infer on vague titles. Entries that remain unclassifiable after these passes are funneled into level 13 (Exclusion). This allows for transparent reporting of data quality: 91.4% of the entries that reached this stage were successfully captured and categorized.

Finally, the processed master dataset is exported via the XlsxWriter engine as a time-stamped file, ensuring strict version control and methodological reproducibility of the classification process.

3.4.6. Automotive Hierarchy Overview

Level 00 – Owner, Founder, Chairman (41)

Level 01 – CEO, President (98)

Level 02 – C-Suite (excl. CEO) (113)

Level 03 – EVP, Global Head (20)

Level 04 – Managing Director, SVP (30)

Level 05 – VP, General Manager (695)

Level 06 – Senior/Executive Director (49)

Level 07 – Director, Senior Manager, Head of (1408)

Level 08 – Manager, Regional Sales, … (6054)

Level 09 – Lead, Supervisor, Principal (1809)

Level 10 – Project Manager, Engineer, Expert (10462)

Level 11 – Professional, Skilled Worker (9218)

Level 12 – Junior, Entry, Trainee, Apprentice (15)

Level 13 — Exclusion, Blank, Invalid Title (2824)

4. Results

4.1. Results (airlines)

	All	United Airlines	American Airlines	Delta	Southwest	Alaska
Level 00, Board	7	4	1			
Level 01, CEO	9	3			1	
Level 02, Executive / Board of Management	12	3			2	
Level 03, SVP	39	7		1	4	
Level 04, VP/MD	285	139	21	9	20	6
Level 05, Sr. Director / GM	172	31	2	12	23	1
Level 06, Director /Head of	597	277	27	8	37	11
Level 07, Manager / Lead / Principal	3158	1580	164	84	352	37
Level 08, Specialist / Analyst / Expert	5436	2861	252	151	867	42
Level 09, Intern / Junior / Apprentice	65	34			7	
Level 13, Exclusion / Blank / Invalid Title	*176*	*85*	*5*	*4*	*24*	*2*
Level 17, TRE / TRI / Instructor	25	15			1	2
Level 18, Captain	448	195	12	8	18	6
Level 19, FO / SFO / Cadet	1313	416	48	63	113	20
Level 27, Trainer	4	2			1	
Level 28, Purser / Sr FA	119	81	3	2	15	1
Level 29, Flight Attendant	2903	1653	66	46	362	35
Level 37, Station Manager/Ops Mgr	27	2			10	
Level 38, Mgr / Dispatcher / Coordinator	984	508	27	6	147	9
Level 39 Clerk / Agent / Assistant	1246	658	37	25	230	10
Level 39.1 Ground Ops / Apprentice / Intern	3	3				
Level 47 Specialized Manager	41	23	4	1		
Level 48, Lead / Supervisor / Planner	288	125	17		45	6
Level 49 Technician / Engineer / Licensed	1958	916	104	84	281	28
Level 49.1, Technic / Apprentice / Intern	16	3		1	5	

Of the 19,331 LinkedIn profiles remaining in the final step, almost all could be assigned to a specific hierarchical level. The low amount of residual noise can be explained by the strict filter for specific company names in the third step. This approach is justifiable, as the airline industry is dominated by a few large players. As the table shows, most major airlines in the United States are organized across 7–8 hierarchical levels. Although the initial aim was to categorize junior positions such as interns or apprentices, this approach did not turn out to be meaningful since only a few entries matched the criteria of this category. The horizontal categorization, however, turned out to be more valid. While the levels 39.1 & 49.1 contained very few observations and should be omitted in future analysis, it indeed shows that level 47, i.e., specialized managers of technicians are at least common at United Airlines and also occur in ground operations (level 37). On the contrary, for the levels 17 and 27, i.e., managing positions of pilots and flights attendants, hardly any LinkedIn profiles could be identified – also because they usually also work as pilots and flight attendants and their job titles often overlap with those of regular flight crew.

4.2.Results (automotive)

Level	All	Ford	Tesla	General Motors	Chrysler
Level 00, Board / Owner / Founder / Chairman	41	1		15	
Level 01, CEO / President	98	8	4	4	5
Level 02, C-Suite (excl. CEO)	113	13	5	6	1
Level 03, Executive Vice President / Group President / Global Head	20	3	1	5	2
Level 04, Managing Director / Senior Vice President	30			1	
Level 05, Vice President / VP / General Manager	695	63	15	16	25
Level 06, Senior Director / Executive Director / Function Owner	49	2	2	26	
Level 07, Director / Senior Manager / Head	1408	100	81	246	51
Level 08, Manager / Senior Role / Regional Sales	6054	425	733	1083	255
Level 09, Lead / Supervisor / Principal	1809	126	252	548	91
Level 10, Specific Manager / Engineer / Expert	10462	724	1820	3224	432
Level 11, Professional / Skilled Worker	9218	625	919	2507	270
Level 12, Junior / Entry / Trainee / Apprentice	15	1	3	7	
Level 13, Exclusion / Blank / Invalid Title	2824	185	257	807	156

32,836 profiles were vertically matched in the last step; 8.6% of those could not be assigned to a specific hierarchical layer. Results of the automotive industry show that most car manufacturers use nearly all hierarchical levels defined in the framework. Only the proposed level 04 is uncommon and indicates that car manufacturers rarely use the job titles of SVPs or MDs. For some, even Senior Directors cannot ever or only very infrequently be found. What is special though, is the large number of engineers (level 10) which exceeds the number of employees classified at level 11. This can mostly be explained by LinkedIn statistics, where profiles with at least a bachelor's degree clearly predominate (Servar, 2024). Therefore, the results indicate that 8-9 hierarchical layers are common in the automotive industry.

5. Discussion

The core goal of this thesis is the expand the framework of Lee (2021) to other industries, namely the huge aviation and automotive industries. These are characterized by a few, but very big players that share the market. In both industries some niche enterprises exist, but the vast majority of the market is concentrated among a few companies. The highly consolidated market substantially differs from the market investigated by Lee's (2021) model, researching on the viability of start-ups. The small number of companies in aviation and car manufacturing makes it more difficult to obtain reasonably large sample to test which KPIs might be influenced by hierarchical depth. Therefore, collecting more industry-specific data, yet with a much broader geographic scope seems advisable for future research when using the previously presented framework. This applies to both investigated industries. The large scale of the companies also provides a reasonable explanation why so many hierarchical layers are needed to ensure an effective corporate coordination. In aviation, 7–8 vertical levels seem to be common, while in the car manufacturing industry 8-9 levels seem to prevail. Moreover, since the companies in these two industries employ such a substantial workforce, many different skill sets are needed which require very different types of formation. Particularly in the aviation industry, there is considerable disparity between the core operational departments. While the training of a pilot usually lasts around two years, flight attendants can finish their training within just six weeks. Comparing them on the same hierarchical level can be done, but this model extends a simple chain of command into various departments which was enabled by industry experts as well as personal professional experience.

Since this study does not yet provide any further insights into the implications of the number of hierarchical levels, further research needs to be done building on this framework.

5.1.Further Outlook

The framework can be expanded for example by a list of first names. During the creation of this thesis, trials were conducted using a long list of names, which can be divided into clearly male, female, or unisex names. This would allow researchers to examine whether, for example, gender diversity on various hierarchical levels positively or negatively influences specific KPIs.

Moreover, already implemented in the Python script is a sophisticated calculation of the total work experience according to the individual's LinkedIn profiles. This feature ensures a reliable calculation by avoiding counting simultaneous work experience, as this would have caused many individuals to have more than 100 years of experience. With the help of this reliable calculation of the experience, it can be determined, whether there is a correlation between younger or older individuals in the lower, middle, and upper managements and the performance of the companies. Furthermore, it is also possible to expand the experience calculator in a way that it determines the number of years of experience within the industry and from outside. This could answer the question whether managers from other industries are beneficial or if long-lasting experiences in the same industry is the way to the highest profits.

5.2.Limitations

Airlines and car manufacturers that consolidate under one roof but maintain their brands are very hard to analyze and often, the exact organization is very difficult to be theoretically reconstructed as an external. For example, Lufthansa Group has a Group-CEO und Group-Board, however, all individual airlines have their own top-level management meaning there are CEOs and functional Chief Officers at various hierarchical levels. The same applies for large car conglomerates like Stellantis or the Volkswagen Group. Moreover, certain operational parts of these large companies are often outsourced. Car manufacturers often outsource their workforce in factories. Some airlines outsource their crew, but particularly maintenance is

mostly done by companies outside of the corporation. MRO (Maintenance, Repair, Overhaul) could be seen as its own independent industry.

Another difficult aspect of this thesis was the integration of General Managers and Managing Directors. Interview partners were unsure where to put them in the hierarchy and when comparing companies, those titles are used at different levels of hierarchy. Moreover, the title "Head of … " is a tricky one. At my employer this title is used internally for any position between the SVP and a Senior Manager. This, of course, makes it very hard to categorize these individuals precisely.

Only one interview was conducted with a major North American airline. Further interviews on that continent could help to get an improved, nuanced coding. At least from an airline perspective, this one North American airline showed a much steeper hierarchy than its European counterparts, even though 3 layers had been crossed very recently.

Last but not least, the available data for American Airlines and Delta Airlines is comparatively limited given the similar size of these two of the three major US network carriers. The other one is United Airlines with way more data available. Opening the filter did increase the number of entries by around 20%, but the data quality deteriorated as an unwanted side effect. Since many companies have "Delta" and even more "American" in their names, a less strict filter causes a lot of noise, rather than high-quality data. Therefore, the only way is to filter is with additional parts of their names like, "airlines" "air lines", etc.

6. Conclusion

This thesis sets out to provide a classification code to measure hierarchical depth in airlines and automotive manufacturers using large-scale job title data. To achieve this goal, a Python-based data processing pipeline was developed. The approach needs several sequential steps: the preprocessing of raw data files, reshaping nested job histories, industry filtering, and the horizontal and vertical categorization of job titles. The first three stages of the pipeline contribute by reducing noise in the data and ensuring that the remaining observations can be interpreted within a standardized hierarchical framework. The introduction of a horizontal categorization is advantageous for industries with highly heterogeneous occupations. Separating functional domains before assigning hierarchical levels allows for a more reliable and clearer interpretation of the hierarchical structure. Still, the latter is approximated by the pivotal last step: the vertical categorization, i.e., the core part of this thesis.

The results indicate that large US airlines typically operate across approximately 7–8 hierarchical layers, while the largest automotive manufacturers are organized in around 8–9levels. These values are no exact blueprints, as such measurements would require official organizational charts rather than self-reported LinkedIn data. However, they do provide an indication of the hierarchical setup in the two investigated industries and, most importantly, the code enables systematic analysis of other companies using the same methodology.

Therefore, the primary contribution of this work lies in the methodological framework itself. Building on Lee's (2021) study of video game startups, this thesis demonstrates that job titles can be used to reconstruct hierarchical structures in the aviation and automotive industries. Extending the approach required substantial methodological adjustments, particularly due to the variety of roles within these sectors.

Despite the carefully designed methodology, the approach has several limitations. LinkedIn data poses the risk of a representation bias in the dataset, as highly educated professionals and managers are more likely to publicly display their academic and professional merits in comparison to purely operational workers. Furthermore, job titles alone cannot perfectly depict authority structures within firms, as titles may differ across companies or contain ambiguous descriptions. Although the rule-based coding

framework attempts to meet these challenges, some degree of misclassification cannot be ruled out entirely.

Still, the developed approach can be applied to other datasets of job titles within the same two industries. Moreover, it enables further research on how hierarchical depth relates to aspects like coordination efficiency, innovation capacity, or other performance measures like profitability or growth rate.

To conclude, this study demonstrated that hierarchical depth can be empirically estimated using large-scale professional profile data when supported by a systematically designed code validated by industry experts. In doing so, it helps bridge the gap between organizational design theory and the empirical measurement of hierarchical structures in real world examples.

Bibliography

Anderson, C., & Berdahl, J. L. (2002). The experience of power: Examining the effects of power on approach and inhibition tendencies. *Journal of Personality and Social Psychology*, *83*(6), 1362–1377. https://doi.org/10.1037/0022-3514.83.6.1362

Anderson, C., & Brown, C. E. (2010). The functions and dysfunctions of hierarchy. *Research in Organizational Behavior*, *30*(1), 55–89. https://doi.org/10.1016/j.riob.2010.08.002

Anderson, C., & Kilduff, G. J. (2009). The Pursuit of Status in Social Groups. *Current Directions in Psychological Science*, *18*(5), 295–298. https://doi.org/10.1111/j.1467–8721.2009.01655.x

Apollo. (2026). Apollo.io: Data-First Sales Platform. Apollo. https://www.apollo.io

Article, R., & Yildirim, C. (2021). *Internal Factors Affecting the Profitability: Evidence from the Global Aviation Industry*. *11*(1), 415–428. https://doi.org/10.30783/nevsosbilen.621188

Barnard, C. I. (1968). *The functions of the executive* (Vol. 11). Harvard university press.

Baron, J. N., & Hannan, M. T. (2002). Organizational Blueprints for Success in High-Tech Start-Ups: Lessons from the Stanford Project on Emerging Companies. *California Management Review*, *44*(3), 8–36. https://doi.org/10.2307/41166130

Beckman, C. M., & Burton, M. D. (2011). Bringing Organizational Demography Back In: Time, Change, and Structure in Top Management Team Research. *The Handbook of Research on Top Management Teams*. https://doi.org/10.4337/9780857933201.00009

Bhidé, A. (2000). *The origin and evolution of new businesses*. Oxford University Press.

Boeker, W., & Wiltbank, R. (2005). New Venture Evolution and Managerial Capabilities. *Organization Science*, *16*(2), 123–133. https://doi.org/10.1287/orsc.1050.0115

Bouwman, H., Carlsson, C., Carlsson, J., Nikou, S., Sell, A., & Walden, P. (2014). *How Nokia failed to nail the Smartphone market*. Www.econstor.eu; Calgary: International Telecommunications Society (ITS). https://www.econstor.eu/handle/10419/101414

Burton, M. D., Colombo, M. G., Rossi-Lamastra, C., & Wasserman, N. (2019). The organizational design of entrepreneurial ventures. *Strategic Entrepreneurship Journal*, *13*(3), 243–255. https://doi.org/10.1002/sej.1332

Burton, R. M., Håkonsson, D. D., Nickerson, J., Puranam, P., Workiewicz, M., & Zenger, T. (2017). GitHub: exploring the space between boss-less and hierarchical forms of organizing. *Journal of Organization Design*, *6*(1). https://doi.org/10.1186/s41469-017-0020-3

Burton, R. M., Lauridsen, J., & Obel, B. (2002). Return on Assets Loss from Situational and Contingency Misfits. *Management Science*, *48*(11), 1461–1485. https://doi.org/10.1287/mnsc.48.11.1461.262

Burton, R. M., & Obel, B. (2004). *Strategic organizational diagnosis and design: The dynamics of fit* (4th ed.). Springer.

Burton, R. M., & Obel, B. (2018). The science of organizational design: fit between structure and coordination. *Journal of Organization Design*, *7*(1), 1–13. Springeropen. https://doi.org/10.1186/s41469-018-0029-2

Burton, R., Håkonsson, D. D., Larsen, E. R., & Obel, B. (2020). New trends in organization design. *Journal of Organization Design*, *9*(1). https://doi.org/10.1186/s41469-020-00072-1

Cheng, P. Y. K. (2007). The Trader Interaction Effect on the Impact of Overconfidence on Trading Performance: An Empirical Study. *Journal of Behavioral Finance*, *8*(2), 59–69. https://doi.org/10.1080/15427560701377232

Clement, J., & Puranam, P. (2018). Searching for Structure: Formal Organization Design as a Guide to Network Evolution. *Management Science*, *64*(8), 3879–3895. https://doi.org/10.1287/mnsc.2017.2807

Cowherd, D. M., & Levine, D. I. (1992). Product Quality and Pay Equity Between Lower-Level Employees and Top Management: An Investigation of Distributive Justice Theory. *Administrative Science Quarterly*, *37*(2), 302. https://doi.org/10.2307/2393226

Csaszar, F. A. (2012). Organizational structure as a determinant of performance: Evidence from mutual funds. *Strategic Management Journal*, *33*(6), 611–632. https://doi.org/10.1002/smj.1969

Dahl, M. S., & Sorenson, O. (2012). Home Sweet Home: Entrepreneurs' Location Choices and the Performance of Their Ventures. *Management Science*, *58*(6), 1059–1071. https://doi.org/10.1287/mnsc.1110.1476

Damodaran, A. (2025). *Operating and Net Margins*. Nyu.edu. https://pages.stern.nyu.edu/~adamodar/New_Home_Page/datafile/margin.html?utm_source=chatgpt.com

Davila, A., Foster, G., & Jia, N. (2010). Building Sustainable High-Growth Startup Companies: Management Systems as an Accelerator. *California Management Review*, *52*(3), 79–105. https://doi.org/10.1525/cmr.2010.52.3.79

Davila, T. (2005). An exploratory study on the emergence of management control systems: formalizing human resources in small growing firms. *Accounting, Organizations and Society*, *30*(3), 223–248. https://doi.org/10.1016/j.aos.2004.05.006

DeSantola, A., & Gulati, R. (2017). Scaling: Organizing and Growth in Entrepreneurial Ventures. *Academy of Management Annals*, *11*(2), 640–668. https://doi.org/10.5465/annals.2015.0125

Eisenhardt, K. M. (1989). Making Fast Strategic Decisions In High-Velocity Environments. *Academy of Management Journal*, *32*(3), 543–576. https://doi.org/10.5465/256434

Ferro, S. (2016, January 26). *What Happened When This Major Company Got Rid Of All Its Bosses*. HuffPost. https://www.huffpost.com/entry/why-you-need-a-boss_n_569fdddbe4b0fca5ba765409

Galbraith, J. R. (1974). Organization Design: An Information Processing View. *Interfaces*, *4*(3), 28–36. https://doi.org/10.1287/inte.4.3.28

Galinsky, A. D., Magee, J. C., Inesi, M. E., & Gruenfeld, D. H. (2006). Power and Perspectives Not Taken. *Psychological Science*, *17*(12), 1068–1074. https://doi.org/10.1111/j.1467-9280.2006.01824.x

Gibbons, R. (2005). Four formal(izable) theories of the firm? *Journal of Economic Behavior & Organization*, *58*(2), 200–245. https://doi.org/10.1016/j.jebo.2004.09.010

Greer, L. L., Bart, J., Schouten, M. E., & Dannals, J. E. (2018). Why and when hierarchy impacts team effectiveness: A meta-analytic integration. *Journal of Applied Psychology*, *103*, 6.

Greer, L. L., & Kleef, van. (2010). Equality versus differentiation: The effects of power dispersion on group interaction. *Journal of Applied Psychology*, *95*, 6.

Hart, O., & Moore, J. (2005). On the Design of Hierarchies: Coordination versus Specialization. *Journal of Political Economy*, *113*(4), 675–702. https://doi.org/10.1086/431794

Hiatt, S. R., & Sine, W. D. (2013). Clear and present danger: Planning and new venture survival amid political and civil violence. *Strategic Management Journal*, *35*(5), 773–785. https://doi.org/10.1002/smj.2113

IATA. (2025). *Airline Profitability Stabilizes with 3.9% Net Margin Expected in 2026*. Iata.org. https://www.iata.org/en/pressroom/2025-releases/2025-12-09-01/

IBISWorld. (2024). *IBISWorld - Industry Market research, reports, and Statistics*. Www.ibisworld.com. https://www.ibisworld.com/global/industry-trends/biggest-industries-by-revenue/

Josefy, M., Kuban, S., Ireland, R. D., & Hitt, M. A. (2015). All Things Great and Small: Organizational Size, Boundaries of the Firm, and a Changing Environment. *Academy of Management Annals*, *9*(1), 715–802. https://doi.org/10.5465/19416520.2015.1027086

Joseph, J., Baumann, O., Burton, R., & Srikanth, K. (2018). Reviewing, Revisiting, and Renewing the Foundations of Organization Design. *Advances in Strategic Management*, *40*, 1–23. https://doi.org/10.1108/s0742-332220180000040012

Keum, D. D., & See, K. E. (2017). The Influence of Hierarchy on Idea Generation and Selection in the Innovation Process. *Organization Science*, *28*(4), 653–669. https://doi.org/10.1287/orsc.2017.1142

Kilduff, G. J., Willer, R., & Anderson, C. (2016). Hierarchy and Its Discontents: Status Disagreement Leads to Withdrawal of Contribution and Lower Group Performance. *Organization Science*, *27*(2), 373–390. https://doi.org/10.1287/orsc.2016.1058

Kirchhoff, B. (1993). *Entrepreneurship and Dynamic Capitalism*. Bloomsbury Publishing USA.

Kossinets, G., & Watts, Duncan J. (2009). Origins of Homophily in an Evolving Social Network. *American Journal of Sociology*, *115*(2), 405–450. https://doi.org/10.1086/599247

Kretschmer, T., & Khashabi, P. (2020). Digital Transformation and Organization Design: An Integrated Approach. *California Management Review*, *62*(4), 86–104. https://doi.org/10.1177/0008125620940296

Lee, M. Y., & Edmondson, A. C. (2017). Self-managing organizations: Exploring the limits of less-hierarchical organizing. *Research in Organizational Behavior*, *37*(1), 35–58. https://doi.org/10.1016/j.riob.2017.10.002

Lee, S. (2021). The Myth of the Flat Start-up: Reconsidering the Organizational Structure of Start-ups. *Strategic Management Journal*, *43*(1). https://doi.org/10.1002/smj.3333

Lee, S. (Ronnie), & Kim, J. D. (2024). When do startups scale? Large-scale evidence from job postings. *Strategic Management Journal*, *45*(9). https://doi.org/10.1002/smj.3596

Liao, C., Chuang, S.-H., & To, P.-L. (2011). How knowledge management mediates the relationship between environment and organizational structure. *Journal of Business Research*, *64*(7), 728–736. https://doi.org/10.1016/j.jbusres.2010.08.001

Malmendier, U., & Tate, G. (2008). Who Makes acquisitions? CEO Overconfidence and the market's Reaction☆. *Journal of Financial Economics*, *89*(1), 20–43. https://doi.org/10.1016/j.jfineco.2007.07.002

McEvily, B., Soda, G., & Tortoriello, M. (2014). More Formally: Rediscovering the Missing Link between Formal Organization and Informal Social Structure. *Academy of Management Annals*, *8*(1), 299–345. https://doi.org/10.5465/19416520.2014.885252

Meier, S., Stephenson, M., & Perkowski, P. (2019). Culture of trust and division of labor in nonhierarchical teams. *Strategic Management Journal*, *40*(8). https://doi.org/10.1002/smj.3024

Milanovich, D. M., Driskell, J. E., Stout, R. J., & Salas, E. (1998). Status and cockpit dynamics: A review and empirical study. *Group Dynamics: Theory, Research, and Practice*, *2*, 3.

Mintzberg, H. (1989). The Structuring of Organizations. *Readings in Strategic Management*, 322–352. https://doi.org/10.1007/978-1-349-20317–8_23

Orchinik, R., & Remer, M. (2023). What's the Difference? Measuring the Effect of Mergers in the Airline Industry. *SSRN Electronic Journal*. https://doi.org/10.2139/ssrn.3602765

Orgio Inc. (2026, February 17). *The Org: Transparency starts here*. The Org. https://theorg.com

Perretti, F., & Negro, G. (2006). Filling Empty Seats: How Status and Organizational Hierarchies Affect Exploration Versus Exploitation in Team Design. *Academy of Management Journal*, *49*(4), 759–777. https://doi.org/10.5465/amj.2006.22083032

Pfeffer, J., & Langton, N. (1993). The Effect of Wage Dispersion on Satisfaction, Productivity, and Working Collaboratively: Evidence from College and University Faculty. *Administrative Science Quarterly*, *38*(3), 382. https://doi.org/10.2307/2393373

Phanish Puranam. (2018). *The Microstructure of Organizations* (1st ed.). Oxford University Press. C. https://scholar.google.com/scholar?hl=de&as_sdt=0%2C5&q=Puranam%2C+P.+%282018%29.+The+microstructure+of+organizations.+Oxford+University+Press.&btnG=

Porter, M. E. (1979). How Competitive Forces Shape Strategy. *Readings in Strategic Management*, *57*(2), 133–143. https://doi.org/10.1007/978-1-349-20317–8_10

Puranam, P. (2014, March 3). *Managing Without Authority: Notes on the Romance and Reality of "Boss-Less" Firms*. Papers.ssrn.com. https://papers.ssrn.com/sol3/papers.cfm?abstract_id=2495910

Puranam, P., Alexy, O., & Reitzig, M. (2014). What's "New" About New Forms of Organizing?. *Academy of Management Review*, *39*(2), 162–180. https://doi.org/10.5465/amr.2011.0436

Puranam, P., & Håkonsson, D. D. (2015). Valve's Way. *Journal of Organization Design*, *4*(2), 2. https://doi.org/10.7146/jod.20152

Reitzig, M. (2022a). *Get Better at Flatter*. Springer International Publishing. https://doi.org/10.1007/978-3-030-89254-8

Reitzig, M. (2022b). How to get better at flatter designs: considerations for shaping and leading organizations with less hierarchy. *Journal of Organization Design*, *11*(11), 5–10. https://doi.org/10.1007/s41469-022-00109-7

Reitzig, M., & Maciejovsky, B. (2014). Corporate hierarchy and vertical information flow inside the firm-a behavioral view. *Strategic Management Journal*, *36*(13), 1979–1999. https://doi.org/10.1002/smj.2334

Reitzig, M., & Sorenson, O. (2013). Biases in the selection stage of bottom-up strategy formulation. *Strategic Management Journal*, *34*(7), 782–799. https://doi.org/10.1002/smj.2047

Rietveld, J., & Eggers, J. P. (2018). Demand Heterogeneity in Platform Markets: Implications for Complementors. *Organization Science*, *29*(2), 304–322. https://doi.org/10.1287/orsc.2017.1183

Rindova, V., Ferrier, W. J., & Wiltbank, R. (2010). Value from gestalt: how sequences of competitive actions create advantage for firms in nascent markets. *Strategic Management Journal*, *31*(13), 1474–1497. https://doi.org/10.1002/smj.892

Savage, C. (2015, October 15). *Ditching Flat: How Structure Helped Us Move Faster*. Wistia. https://wistia.com/learn/culture/ditching-flat

Serrano, F. (2025, June). *Financial and strategic impact of mergers and acquisitions in the automotive industry: a multi-case study*. Upc.edu; Universitat Politècnica de Catalunya. https://upcommons.upc.edu/entities/publication/8a4b6a3d-7238-49a7-b5e5-52c7d1f7b71b

Servar, M. (2024, August 16). 2024 LinkedIn Demographics and Statistics that Matter for Your Business. 2024 LinkedIn Demographics That Matter for Your Business | Linked Helper Blog. https://www.linkedhelper.com/blog/linkedin-demographics/?utm_source=chatgpt.com

Slade Shantz, A. F., Kistruck, G. M., Pacheco, D. F., & Webb, J. W. (2020). How Formal and Informal Hierarchies Shape Conflict within Cooperatives: A Field Experiment in Ghana. *Academy of Management Journal*, *63*(2), 503–529. https://doi.org/10.5465/amj.2018.0335

Stinchcombe, A. L. (1965). Social structure and organizations. In *Advances in Strategic Management* (Vol. 17, pp. 229–259). Emerald. https://doi.org/10.1016/s0742-3322(00)17019-6

Stock Analysis. (2026). *List of The Biggest Companies by Number of Employees*. Stock Analysis. https://stockanalysis.com/list/most-employees/

Stricker, K., Correa, P., & Stein, I. (2025, January 15). *Automotive Profitability: How OEM and Supplier Margins Are Faring*. Bain & Company. https://www.bain.com/insights/automotive-profitability-how-oem-and-supplier-margins-are-faring-interactive/?utm_source=chatgpt.com

Tarakci, M., Greer, L. L., & Patrick. (2016). When does power disparity help or hurt group performance? *Journal of Applied Psychology*, *101*, 3.

Tost, L. P., Gino, F., & Larrick, R. P. (2013). When Power Makes Others Speechless: The Negative Impact of Leader Power on Team Performance. *Academy of Management Journal*, *56*(5), 1465–1486. https://doi.org/10.5465/amj.2011.0180

Van de Ven, A. H. (1976). A Framework For Organization Assessment. *Academy of Management Review*, *1*(1), 64–78. https://doi.org/10.5465/amr.1976.4408765

Van de Ven, A. H., Ganco, M., & Hinings, C. R. (BOB). (2013). Returning to the Frontier of Contingency Theory of Organizational and Institutional Designs. *Academy of Management Annals*, *7*(1), 393–440. https://doi.org/10.5465/19416520.2013.774981

Wasserman, N. (2012). *The Founder's Dilemmas : Anticipating and Avoiding the Pitfalls That Can Sink a Startup.* Princeton University Press. https://www.torrossa.com/en/resources/an/5579875

Zhou, Y. M. (2013). Designing for Complexity: Using Divisions and Hierarchy to Manage Complex Tasks. *Organization Science*, *24*(2), 339–355. https://doi.org/10.1287/orsc.1120.0744

zoominfo. (2026). ZoomInfo: B2B Database | Business Leads & Company Contacts. ZoomInfo. https://zoominfo.com

Annex A: Python Codes

A.1. Aviation Industry

A.1.1. Preprocessing

```
import pandas as pd
from datetime import datetime
import os
import re

# Define the directory containing the input CSV files
input_folder = "/Users/Christoph/Desktop/MasterarbeitUniWien/0inputfiles_aviation"
# Define the directory for the processed Excel output files
output_folder = "/Users/Christoph/Desktop/MasterarbeitUniWien/1processed_aviation"

# Ensure the output directory exists
os.makedirs(output_folder, exist_ok=True)

# Define the subset of columns to be extracted from the source data
columns_to_keep = [
    "id", "current_company", "education", "experience", "position",
    "experience_company", "experience_industry", "experience_title",
    "experience_duration", "crunchbase_company", "num_employees", "type",
    "crunchbase_industries", "contacts", "current_employees",
    "num_contacts", "legal_name", "industry_dummy"
]


def clean_string_for_excel(x):
    """
    Removes non-printable control characters from string values
    to ensure compatibility with the Excel file format.
    """
    if isinstance(x, str):
        return re.sub(r"[\x00-\x1F\x7F-\x9F]", " ", x)
    return x


# Iterate through all files in the specified input directory
for filename in os.listdir(input_folder):
    # Process only files with a .csv extension
    if filename.endswith(".csv"):
        input_path = os.path.join(input_folder, filename)

        # Create a timestamp for the output filename
        timestamp = datetime.now().strftime("%Y.%m.%d.%H.%M")
        # Define the output path using the original filename and the current timestamp
        base_name = os.path.splitext(filename)[0]
        output_path = os.path.join(output_folder, f"1pre_{base_name}_{timestamp}.xlsx")
```

```
        try:
            print(f"Processing: {filename}")

            # Load the CSV data, filtering for relevant columns
            df = pd.read_csv(
                input_path,
                usecols=lambda col: col in columns_to_keep,
                low_memory=False,
                memory_map=True
            )

            # Apply regex-based cleaning to all object (string) columns
            df_cleaned = df.apply(
                lambda col: col.map(clean_string_for_excel) if col.dtype == "object" else col
            )

            # Export the processed DataFrame to an Excel file
            df_cleaned.to_excel(output_path, index=False, engine="openpyxl")
            print(f"Successfully saved: {output_path}")

        except Exception as e:
            # Log errors encountered during the processing of individual files
            print(f"Error processing {filename}: {e}")

print("Data preprocessing completed.")
```

A.1.2. Reshape

```
import pandas as pd
import json
from datetime import datetime
from dateutil.relativedelta import relativedelta
import os
import re
from tqdm import tqdm # Import tqdm for progress bar

# === KONFIGURATION ===
# Path to the TXT file containing the full paths of your 1pre_combined_*.xlsx files
# IMPORTANT: Ensure this path is correct on YOUR LOCAL SYSTEM.
INPUT_PATHS_TXT_FILE =
"/Users/Christoph/Desktop/MasterarbeitUniWien/2025.05.11/1processed_excel_paths.txt"

# Directory where the 2reshaped_ files will be saved
# IMPORTANT: Ensure this path is correct on YOUR LOCAL SYSTEM.
OUTPUT_RESHAPED_DIR =
"/Users/Christoph/Desktop/MasterarbeitUniWien/2reshaped_excel"

# Ensure the output directory exists
os.makedirs(OUTPUT_RESHAPED_DIR, exist_ok=True)

# Max positions to flatten from experience JSON (adjust as needed)
```

```
MAX_POSITIONS_TO_FLATTEN = 30

# Mapping of month abbreviations to numerical values for date conversion
month_map = {
   "Jan": 1, "Feb": 2, "Mar": 3, "Apr": 4,
   "May": 5, "Jun": 6, "Jul": 7, "Aug": 8,
   "Sep": 9, "Oct": 10, "Nov": 11, "Dec": 12
}


def parse_date(date_str):
   """
   Parses a date string into a datetime object. Handles 'Present' and various 'Mon.YYYY'
formats.
   """
   # Return None if the input is not a string
   if not isinstance(date_str, str):
      return None
   # If the date is 'present', use today's date
   if date_str.strip().lower() == "present":
      return datetime.today()

   # Look for patterns like 'Jan 2020' or 'Jan. 2020' using regular expressions
   match = re.match(r"(\w{3})\.? (\d{4})", date_str.strip())
   if match:
      month_str, year = match.groups()
      month = month_map.get(month_str)
      if month:
         # Create a datetime object for the first day of that month and year
         return datetime(int(year), int(month), 1)

   # If only a four-digit year is provided, default to January 1st of that year
   if re.match(r"^\d{4}$", date_str.strip()):
      try:
         return datetime(int(date_str.strip()), 1, 1)
      except ValueError:
         pass  # Continue to return None if year-only parse fails

   return None


def months_between(d1, d2):
   """
   Calculates the number of months between two datetime objects.
   """
   # Return 0 if either date is missing
   if not d1 or not d2:
      return 0
   # Ensure the first date is the earlier one to avoid negative results
   if d1 > d2:
      d1, d2 = d2, d1
```

```
    # Calculate the difference in years, months, and days
    delta = relativedelta(d2, d1)
    # Convert years to months and add remaining months, adjusting for day count
    return delta.years * 12 + delta.months + (delta.days > 15)


def parse_experience_entry(entry_str, row_id=None):
    """
    Parses a JSON string from the 'experience' column, handles various malformations,
    and flattens nested 'positions' lists into a list of standardized dictionaries.
    """
    # Return an empty list if there is no string to parse
    if not isinstance(entry_str, str):
        return []

    # Return an empty list if the string is empty or indicates no data
    if not entry_str.strip() or entry_str.strip() in ['[]', '{}', 'None']:
        return []

    processed_str = entry_str
    try:
        # Clean the string to make it valid JSON format
        processed_str = processed_str.replace("\\", "\\\\")  # Handle backslashes
        processed_str = processed_str.replace("\n", "\\n").replace("\r", "\\r").replace("\t",
                                                                "\\t")  # Handle whitespace
        processed_str = processed_str.replace("'", '"')  # Convert single quotes to double quotes

        # Use regular expressions to put quotes around dictionary keys that lack them
        processed_str = re.sub(r'([{,]\s*)([a-zA-Z_]\w*)\s*:', r'\1"\2":', processed_str)

        # Fix structural issues where objects are next to each other without a comma
        processed_str = re.sub(r'}\s*{', '},{', processed_str)

        # Ensure the string is wrapped in square brackets so it is treated as a list
        if not processed_str.strip().startswith('['):
            processed_str = '[' + processed_str + ']'

        # Convert the cleaned string into a Python list of dictionaries
        parsed_entries = json.loads(processed_str)

        # Ensure the final object is a list; wrap it if it is a single dictionary
        if not isinstance(parsed_entries, list):
            if isinstance(parsed_entries, dict):
                parsed_entries = [parsed_entries]
            else:
                return []

        parsed_list_of_positions = []
        # Iterate through each company or job entry in the list
        for entry in parsed_entries:
```

```
if isinstance(entry, dict):
    company = entry.get("company_name") or entry.get("company")
    industry = entry.get("industry")

    # If the entry contains a sub-list of specific job positions
    positions_list = entry.get("positions")
    if isinstance(positions_list, list) and positions_list:
        for p in positions_list:
            if isinstance(p, dict):
                # Extract job details and convert date strings to date objects
                title = p.get("title")
                start_date_str = p.get("start_date")
                end_date_str = p.get("end_date")

                start = parse_date(start_date_str)
                end = parse_date(end_date_str)

                # Ignore entries where the start date is after the end date
                if start and end and start > end:
                    continue
                # Ignore entries that have an end date but no start date
                elif not start and end:
                    continue

                # Add the standardized job data to the list
                parsed_list_of_positions.append({
                    "company": company,
                    "title": title,
                    "start": start,
                    "end": end,
                    "industry": industry
                })
    else:
        # If there is no sub-list, treat the main entry as the job position
        title = entry.get("title")
        start_date_str = entry.get("start_date")
        end_date_str = entry.get("end_date")

        start = parse_date(start_date_str)
        end = parse_date(end_date_str)

        # Apply same date validation rules
        if start and end and start > end:
            continue
        elif not start and end:
            continue

        parsed_list_of_positions.append({
            "company": company,
            "title": title,
            "start": start,
```

```
                "end": end,
                "industry": industry
            })
        return parsed_list_of_positions
    except json.JSONDecodeError as e:
        # Handles cases where string cleaning failed to create valid JSON
        return []
    except Exception as e:
        # Handles any other unexpected errors during parsing
        return []


def reshape_and_save_single_file(input_filepath, output_dir,
max_positions=MAX_POSITIONS_TO_FLATTEN):
    """
    Reshapes a single input Excel file and saves the result.
    """
    print(f"\n--- Reshaping {os.path.basename(input_filepath)} ---")

    try:
        # Load the Excel file into a pandas DataFrame
        df_input = pd.read_excel(input_filepath)
        print(f"  Loaded {len(df_input)} rows.")
        # Identify and remove rows with duplicate IDs to ensure data uniqueness
        df_input = df_input.drop_duplicates(subset="id", keep="first")
        print(f"  Removed duplicates. {len(df_input)} unique rows remaining.")
    except FileNotFoundError:
        print(f"  Error: Input file not found at {input_filepath}. Skipping.")
        return None
    except Exception as e:
        print(f"  An error occurred while loading {input_filepath}: {e}. Skipping.")
        return None

    all_records = []

    # Process each row of the spreadsheet individually
    for index, row in tqdm(df_input.iterrows(), total=df_input.shape[0], desc="  Reshaping
rows"):
        row_id = row.get("id", f'Row_{index}')
        # Extract and standardize job history from the JSON-formatted experience column
        flat_positions = parse_experience_entry(row.get("experience"), row_id)

        # Sort the jobs by start date (newest first) and remove any entries missing a start date
        flat_positions = sorted([p for p in flat_positions if p["start"]], key=lambda x: x["start"],
reverse=True)

        # Logic to fix overlapping dates so time is not double-counted
        for i in range(1, len(flat_positions)):
            # Adjust previous end date if it overlaps with the current start date
            if flat_positions[i - 1]["end"] and flat_positions[i]["start"] and flat_positions[i -
1]["end"] > \
```

```
                flat_positions[i]["start"]:
            flat_positions[i - 1]["end"] = flat_positions[i]["start"]
        # Adjust current end date if it overlaps with the previous start date
        if flat_positions[i]["end"] and flat_positions[i - 1]["start"] and flat_positions[i]["end"]
> \
                flat_positions[i - 1]["start"]:
            flat_positions[i]["end"] = flat_positions[i - 1]["start"]

    # Variable to hold total calculated work experience
    total_months = 0

    # Define time periods for each job entry
    periods = []
    for p in flat_positions:
        if p["start"] and p["end"]:
            periods.append((p["start"], p["end"]))
        elif p["start"] and not p["end"]:  # Handle jobs listed as currently active
            periods.append((p["start"], datetime.today()))

    # Sort job periods chronologically
    periods.sort()

    # Combine overlapping periods into single continuous blocks of time
    merged_periods = []
    if periods:
        current_start, current_end = periods[0]
        for i in range(1, len(periods)):
            next_start, next_end = periods[i]
            if next_start <= current_end:  # If periods touch or overlap, merge them
                current_end = max(current_end, next_end)
            else:  # If there is a gap, save the current block and start a new one
                merged_periods.append((current_start, current_end))
                current_start, current_end = next_start, next_end
        merged_periods.append((current_start, current_end))  # Save the final block

    # Calculate the total duration in months across all merged time blocks
    for start, end in merged_periods:
        total_months += months_between(start, end)

    # Convert total months to years
    total_years = round(total_months / 12, 2)

    # Convert the original row to a dictionary and add the new total experience value
    base = row.to_dict()
    base["total_experience_years"] = total_years

    # Create new columns for each individual job (up to the defined maximum)
    for i, p in enumerate(flat_positions[:max_positions]):
        idx = i + 1
        base[f"exp_{idx}_company"] = p["company"]
        base[f"exp_{idx}_title"] = p["title"]
```

```
            base[f"exp_{idx}_start"] = p["start"]
            base[f"exp_{idx}_end"] = p["end"]
            base[f"exp_{idx}_industry"] = p["industry"]
            base[f"exp_{idx}_duration_years"] = round(months_between(p["start"], p["end"]) /
12, 2)

        # Add the completed record to the collection
        all_records.append(base)

    # Convert the list of dictionaries back into a pandas DataFrame
    df_reshaped = pd.DataFrame(all_records)
    print(f"  Reshaped dataframe to {len(df_reshaped.columns)} columns and
{len(df_reshaped)} rows.")

    # Generate the output filename based on the input name and current time
    filename = os.path.basename(input_filepath)
    match = re.match(r"1pre_(combined_\d+)_(\d{4}\.\d{2}\.\d{2}\.\d{2}\.\d{2})\.xlsx",
filename)
    file_identifier = match.group(1) if match else "unknown_file"
    current_process_timestamp = datetime.now().strftime('%Y.%m.%d.%H.%M')

    output_file_name = f"2reshaped_{file_identifier}_{current_process_timestamp}.xlsx"
    output_path = os.path.join(output_dir, output_file_name)

    print(f"  Saving reshaped data to: {output_path}")
    try:
        # Save the final processed data to a new Excel file
        df_reshaped.to_excel(output_path, index=False, engine='xlsxwriter')
        print("  Reshaping complete and data saved successfully.")
        print(f"  Output saved to: {output_path}")
    except Exception as e:
        print(f"  An **ERROR** occurred while saving reshaped data: {e}.")
        print(f"  Please check output directory '{output_dir}' and permissions.")
    return df_reshaped

# MAIN BATCH EXECUTION FOR RESHAPE

if __name__ == "__main__":
    # Start of the script execution
    print(f"Starting batch reshaping. Reading input file paths from:
{INPUT_PATHS_TXT_FILE}")

    preprocessed_files_paths = []
    try:
        # Open and read the text file containing paths to Excel files
        with open(INPUT_PATHS_TXT_FILE, 'r') as f:
            for line in f:
                path = line.strip()
                # Only add paths that exist and are Excel files
                if path and path.endswith(".xlsx") and os.path.exists(path):
                    preprocessed_files_paths.append(path)
```

```
            elif path:
                print(f"Warning: File path '{path}' from '{INPUT_PATHS_TXT_FILE}' does
not exist or is not an Excel file. Skipping.")
        print(f"Found {len(preprocessed_files_paths)} valid input file paths in the text file.")
    except FileNotFoundError:
        print(f"Error: Input paths TXT file not found at {INPUT_PATHS_TXT_FILE}. Please
ensure the path is correct. Exiting.")
        exit()
    except Exception as e:
        print(f"An unexpected error occurred while reading the input paths TXT file: {e}.
Exiting.")
        exit()

    # Process each file found in the text file
    if not preprocessed_files_paths:
        print("No valid input file paths found in the TXT file. Exiting.")
    else:
        # Iterate through the list of files in alphabetical order
        for full_input_path in sorted(preprocessed_files_paths):
            # Perform the reshaping logic on each individual file
            reshape_and_save_single_file(full_input_path, OUTPUT_RESHAPED_DIR)

    print("\nBatch reshaping execution finished for all files.")
```

A.1.3. Filter

```
import pandas as pd
import os
from tqdm import tqdm
from thefuzz import fuzz
from datetime import datetime
import re

# === KONFIGURATION ===
# Path to the TXT file containing the full paths of your 2reshaped_*.xlsx files
# IMPORTANT: Ensure this path is correct on YOUR LOCAL SYSTEM.
INPUT_PATHS_TXT_FILE =
"/Users/Christoph/Desktop/MasterarbeitUniWien/2025.05.11/2reshaped_excel_paths.txt"

# Define the output directory for filtered files
# IMPORTANT: Ensure this path is correct on YOUR LOCAL SYSTEM.
OUTPUT_DIR = "/Users/Christoph/Desktop/MasterarbeitUniWien/3filtered_excel"
os.makedirs(OUTPUT_DIR, exist_ok=True) # Ensure the output directory exists

# Paths for whitelist and blacklist files
# IMPORTANT: Ensure these files (final_airline_whitelist.xlsx, airline_blacklist.txt)
# are located in the same directory as this script, or provide their full paths.
WHITELIST_PATH =
"/Users/Christoph/Desktop/MasterarbeitUniWien/2025.05.11/final_airline_whitelist.xlsx"
BLACKLIST_PATH =
"/Users/Christoph/Desktop/MasterarbeitUniWien/2025.05.11/airline_blacklist.txt"
```

```
# === HELPER FUNCTIONS ===
def clean_text(text):
    """Cleans text by converting to lowercase, removing common suffixes, and stripping
whitespace."""
    # Return an empty string if the input is missing (NaN)
    if pd.isna(text):
        return ""
    # Convert text to lowercase and remove standard corporate designations
    return str(text).lower().replace("inc.", "").replace("ltd.", "").replace("llc", "").strip()

def fuzzy_match(company, reference):
    """Performs a fuzzy match, with special handling for airline suffixes."""
    # Standardize both strings using the cleaning function
    company = clean_text(company)
    reference = clean_text(reference)
    # Define common words found at the end of airline names
    suffixes = ["airlines", "airways", "air"]
    for suffix in suffixes:
        # If both names end with the same suffix, compare only the main brand name
        if company.endswith(suffix) and reference.endswith(suffix):
            base_company = company.rsplit(suffix, 1)[0]
            base_ref = reference.rsplit(suffix, 1)[0]
            # Return True if the brand names are at least 80% similar
            return fuzz.ratio(base_company.strip(), base_ref.strip()) > 80
    # Otherwise, perform a standard similarity check on the full names
    return fuzz.ratio(company, reference) > 80


# === LOAD WHITELIST AND BLACKLIST ===
print("Loading whitelist and blacklist...")
try:
    # Read the allowed airline list from an Excel file
    wl = pd.read_excel(WHITELIST_PATH)
    print(f"  Whitelist loaded from: {WHITELIST_PATH}")
except FileNotFoundError:
    print(f"Error: Whitelist file not found at {WHITELIST_PATH}. Please check the path and
file name.")
    exit()
except Exception as e:
    print(f"An error occurred while loading the whitelist: {e}")
    exit()

try:
    # Read the excluded airline list from a text file, ignoring empty lines
    bl = [line.strip().lower() for line in open(BLACKLIST_PATH, "r").read().splitlines() if
line.strip()]
    print(f"  Blacklist loaded from: {BLACKLIST_PATH}")
except FileNotFoundError:
```

```
    print(f"Error: Blacklist file not found at {BLACKLIST_PATH}. Please check the path and
file name.")
    exit()
except Exception as e:
    print(f"An error occurred while loading the blacklist: {e}")
    exit()

def in_whitelist(company):
    """Checks if a company is in the whitelist using fuzzy matching."""
    # Return False if the company name is empty
    if pd.isna(company):
        return False
    # Compare the company against every entry in the whitelist
    return any(fuzzy_match(company, ref) for ref in wl["airline_brand"].dropna())

def in_blacklist(company):
    """Checks if a company is in the blacklist using fuzzy matching."""
    # Return False if the company name is empty
    if pd.isna(company):
        return False
    # Compare the company against every entry in the blacklist
    return any(fuzzy_match(company, ref) for ref in bl if isinstance(ref, str))


# === FILTER FUNCTION WITH STRICT INDUSTRY AND COMPANY LOGIC ===
def is_relevant(row):
    """
    Determines if a row is relevant based on industry keywords and company
whitelist/blacklist.
    Checks exp_1_company and exp_1_industry.
    """
    # Retrieve the company name and industry of the most recent job
    company = row.get("exp_1_company", "")
    industry = str(row.get("exp_1_industry", "")).lower()

    # Check if the industry description contains relevant keywords
    industry_matches = ("airline" in industry or "aviation" in industry)

    # Check if the company is authorized (on whitelist and not on blacklist)
    company_matches = (in_whitelist(company) and not in_blacklist(company))

    # A row is kept only if both the industry and the company name are relevant
    return industry_matches and company_matches

def filter_and_save_single_file(input_filepath, output_dir, wl_df, bl_list):
    """
    Filters a single input Excel file and saves the result.
    """
    print(f"\n--- Filtering {os.path.basename(input_filepath)} ---")

    try:
```

```
        # Load the reshaped data into a DataFrame
        df = pd.read_excel(input_filepath)
        print(f"  Loaded {len(df)} rows.")
    except FileNotFoundError:
        print(f"  Error: Input file not found at {input_filepath}. Skipping.")
        return None
    except Exception as e:
        print(f"  An error occurred while loading {input_filepath}: {e}. Skipping.")
        return None

    # === APPLY FILTER WITH PROGRESS BAR ===
    print("Applying filter to dataset (this may take a while)...")
    tqdm.pandas(desc="  Filtering rows")

    # Identify which rows meet the criteria and count them for verification
    relevance_mask = df.progress_apply(is_relevant, axis=1)
    true_count = relevance_mask.sum()
    false_count = len(relevance_mask) - true_count
    print(f"\n  --- DEBUG: Filter Relevance Summary ---")
    print(f"  Number of rows evaluated as RELEVANT: {true_count}")
    print(f"  Number of rows evaluated as NOT RELEVANT: {false_count}")
    print(f"  --- END DEBUG ---\n")

    # Create a new DataFrame containing only the relevant rows
    filtered_df = df[relevance_mask].copy()

    print(f"Filtering complete. {len(filtered_df)} relevant rows found.")

    # === EXPORT ===
    # Parse the original filename to maintain consistent naming conventions
    filename = os.path.basename(input_filepath)
    match = re.match(r"2reshaped_(combined_\d+)_(\d{4}\.\d{2}\.\d{2}\.\d{2}\.\d{2})\.xlsx",
filename)
    file_identifier = match.group(1) if match else "unknown_file"
    # Create a timestamp for the current filtering process
    current_process_timestamp = datetime.now().strftime('%Y.%m.%d.%H.%M')

    # Construct the final output filename
    output_file_name =
f"3filtered_{file_identifier}_with_industry_strict_filter_{current_process_timestamp}.xlsx"
    output_path = os.path.join(output_dir, output_file_name)

    print(f"Saving filtered data to: {output_path}")
    try:
        # Write the filtered results to a new Excel file
        filtered_df.to_excel(output_path, index=False, engine='xlsxwriter')
        print("Filtered data saved successfully.")
        print(f"Output saved to: {output_path}")
    except Exception as e:
        print(f"An **ERROR** occurred while saving the filtered data:")
        print(f"Error Type: {type(e).__name__}")
```

```
        print(f"Error Message: {e}")
        print(f"Please check if the output directory '{output_dir}' exists and you have write permissions, or if the file is open elsewhere.")
    return filtered_df

# MAIN BATCH EXECUTION FOR FILTER

if __name__ == "__main__":
    print(f"Starting batch filtering. Reading input file paths from: {INPUT_PATHS_TXT_FILE}")

    reshaped_files_paths = []
    try:
        # Open and read the text file containing paths to the reshaped Excel files
        with open(INPUT_PATHS_TXT_FILE, 'r') as f:
            for line in f:
                path = line.strip()
                # Verify that each path exists and points to an Excel file
                if path and path.endswith(".xlsx") and os.path.exists(path):
                    reshaped_files_paths.append(path)
                elif path:
                    print(f"Warning: File path '{path}' from '{INPUT_PATHS_TXT_FILE}' does not exist or is not an Excel file. Skipping.")
        print(f"Found {len(reshaped_files_paths)} valid input file paths in the text file.")
    except FileNotFoundError:
        print(f"Error: Input paths TXT file not found at {INPUT_PATHS_TXT_FILE}. Please ensure the path is correct. Exiting.")
        exit()
    except Exception as e:
        print(f"An unexpected error occurred while reading the input paths TXT file: {e}. Exiting.")
        exit()

    # Execute the filtering process if valid files were found
    if not reshaped_files_paths:
        print("No valid input file paths found in the TXT file. Exiting.")
    else:
        # Process each file in alphabetical order
        for full_input_path in sorted(reshaped_files_paths):
            # Apply the industry and company filters to the file
            filter_and_save_single_file(full_input_path, OUTPUT_DIR, wl, bl)

    print("\nBatch filtering execution finished for all files.")
```

A.1.3.1. Merging

```
import pandas as pd
import os
from datetime import datetime

# --- CONFIGURATION ---
```

```
# Define the source directory where the individual batch files are stored
input_folder = '/Users/Christoph/Desktop/MasterarbeitUniWien/3filtered_excel'

# Define the destination directory for the single consolidated output file
output_folder =
'/Users/Christoph/Desktop/MasterarbeitUniWien/3.5_filtered_excel_combined'


def combine_excel_files(input_dir, output_dir):
    """
    Reads all Excel files from a directory, combines them into a single DataFrame,
    and saves the result to a new Excel file.
    """
    # Ensure the target output directory exists before proceeding
    os.makedirs(output_dir, exist_ok=True)

    # Initialize an empty list to collect data from each individual file
    all_dataframes = []

    print(f" Reading all Excel files from: {input_dir}")

    # Iterate through every file in the specified input directory
    for filename in os.listdir(input_dir):
        # Target only Excel spreadsheet formats for processing
        if filename.endswith('.xlsx') or filename.endswith('.xls'):
            file_path = os.path.join(input_dir, filename)
            try:
                # Load the current Excel file into a temporary DataFrame
                df = pd.read_excel(file_path)
                # Store the DataFrame in the collection list
                all_dataframes.append(df)
                print(f" Successfully read {filename}")
            except Exception as e:
                # Catch and report errors for specific files to avoid stopping the entire script
                print(f" ERROR: Could not read {filename}. Skipping. Error: {e}")

    # Verify that data was successfully collected before attempting to merge
    if not all_dataframes:
        print("\n No Excel files were found in the specified directory. Exiting.")
        return

    # Merge all collected DataFrames into a single continuous table
    combined_df = pd.concat(all_dataframes, ignore_index=True)

    # Generate a unique filename using the current system date and time
    timestamp = datetime.now().strftime('%Y.%m.%d.%H.%M')
    output_filename = f'combined_filtered_data_{timestamp}.xlsx'
    output_path = os.path.join(output_dir, output_filename)

    print(f"\n All files combined successfully! Total rows: {len(combined_df)}.")
    print(f" Saving the combined data to: {output_path}")
```

```
    # Export the consolidated DataFrame to a single master Excel file
    combined_df.to_excel(output_path, index=False)

    print("\n File combination script finished successfully!")


# Entry point for the script execution
if __name__ == "__main__":
    # Execute the combination process using the folders defined in the configuration
    combine_excel_files(input_folder, output_folder)
```

A.1.3.2. Deletion

```
import pandas as pd
import os
import re
from datetime import datetime

# --- CONFIGURATION ---
# Path to the specific master file generated in the previous consolidation step
input_file_path =
'/Users/Christoph/Desktop/MasterarbeitUniWien/3.5_filtered_excel_combined/combined_filt
ered_data_2025.08.30.19.50.xlsx'
# Directory where the final cleaned dataset will be stored
output_folder = '/Users/Christoph/Desktop/MasterarbeitUniWien/3.6_delete_combines_all'
# Potential column names to search for professional titles
TITLE_COLUMNS = ['exp_1_title', 'job_title', 'position', 'title', 'titel']

# Create the output directory if it is not already present
os.makedirs(output_folder, exist_ok=True)

# --- DELETION RULES (PERMANENTLY REMOVE ROWS) ---
# Regular expression patterns used to identify and exclude rows with irrelevant or non-active
roles
DELETION_PATTERNS = [
    r'\b(former|retired|past|previous|ex)\b',
    r'\b(actor|actress|comedian|musician)\b',
    r'\b(mom|dad|mother|father)\b',
    r'\badventure\b',
    r'\bbadass\b',
    r'\b(airlines?/aviation)\s+professional\b',
    r'\b(dad|mom)\s+of\b',
    r'\b(looking)\b',
    r'\b(cook|chef|kitchen)\b',
    r'\b(Open to work)\b',
    r'\b(visionary)\b',
    r'\b(Self Employed)\b',
    r'\b(Hello)\b',
    r'\b(Dance Educator)\b',
    r'\b(Aviation)\b',
```

```
    r'\bpilot\b',
]

# Specific characters or strings that signal data noise and should trigger row removal
STRINGS_TO_DELETE = ['- -', 'â']


def delete_rows_from_file(input_path: str, output_folder: str,
title_columns=TITLE_COLUMNS):
    """
    Analyzes a dataset and removes rows that meet specific exclusion criteria based on job
titles.
    """
    timestamp = datetime.now().strftime('%Y.%m.%d.%H.%M')

    # Verify that the master file exists before attempting to load it
    if not os.path.exists(input_path):
        print(f"\n ERROR: Input file not found at {input_path}.")
        return

    print(f" Processing the combined file for row deletion: {os.path.basename(input_path)}")

    try:
        # Load the spreadsheet and convert all column headers to lowercase for consistency
        df = pd.read_excel(input_path)
        df = df.rename(columns={c: c.lower() for c in df.columns})

        # Locate the first available column from the preference list that exists in the file
        title_col = next((c for c in title_columns if c in df.columns), None)
        if not title_col:
            print(f" No title column found from {title_columns}. Cannot proceed with deletion.")
            return

        initial_row_count = len(df)

        # Rule 1: Exclude rows where the title field is null or contains only empty spaces
        df = df[df[title_col].astype(str).str.strip() != '']

        # Rule 2: Exclude rows where the title matches any of the defined negative regex
patterns
        for pattern in DELETION_PATTERNS:
            df = df[~df[title_col].astype(str).str.contains(pattern, case=False, na=False,
regex=True)]

        # Rule 3: Exclude rows containing specific noise strings or corrupted characters
        for string in STRINGS_TO_DELETE:
            # Remove exact matches
            df = df[df[title_col].astype(str).str.strip() != string]
            # Remove rows where the string appears as part of a larger text block
            df = df[~df[title_col].astype(str).str.contains(string, case=False, na=False,
regex=False)]
```

```
        # Calculate and report the impact of the cleaning process
        rows_deleted = initial_row_count - len(df)
        print(f" Deleted {rows_deleted} rows based on exclusion rules.")
        print(f"Total rows remaining: {len(df)}.")

        # Construct the final filename including the current timestamp
        output_filename = f'cleaned_data_{timestamp}.xlsx'
        output_path = os.path.join(output_folder, output_filename)

        print(f" Saving the cleaned data to: {output_path}")
        # Write the final cleaned results to a new Excel file
        df.to_excel(output_path, index=False)
        print(" Script finished successfully!")

    except Exception as e:
        # Log any errors encountered during the reading or cleaning phase
        print(f" ERROR: Could not process {os.basename(input_path)}: {e}.")


# Standard execution block
if __name__ == "__main__":
    # Start the deletion process using the configured file paths
    delete_rows_from_file(input_file_path, output_folder)
```

A.1.4. Horizontal Categorization

```
import pandas as pd
import os
import re
from collections import OrderedDict
from datetime import datetime
from typing import Union, Optional

# --- CONFIGURATION ---
# Define the file path for the input Excel data
input_file =
'/Users/Christoph/Desktop/MasterarbeitUniWien/3.6_delete_combines_all/cleaned_data_202
5.08.31.11.04.xlsx'
# Define the folder where the processed file will be saved
output_folder = '/Users/Christoph/Desktop/MasterarbeitUniWien/4flatten'
# Define a list of possible column names that might contain job titles
TITLE_COLUMNS = ['exp_1_title', 'job_title', 'position', 'title', 'titel']
# Define the name of the new column that will store the processed categories
FLATTENED_COLUMN = 'category_neo'
# Define the file path for the list of approved airlines
AIRLINE_WHITELIST_PATH =
'/Users/Christoph/Desktop/MasterarbeitUniWien/2025.05.11/final_airline_whitelist.xlsx'

# Create the output directory if it does not already exist
os.makedirs(output_folder, exist_ok=True)
```

```
# --- SHARED HELPERS ---
# Helper function to convert text to lowercase and handle non-string values
def _lower(s): return "" if not isinstance(s, str) else s.lower()


# Helper function to remove extra whitespace and clean up text
def _ws(s):    return re.sub(r"\s+", " ", s).strip()


# Logic to determine if a category should be updated (only if empty or marked
'Uncategorized')
def _should_update(cur: Union[str, None]) -> bool:
    """
    Only update category_flat if it is empty/NaN or 'Uncategorized'.
    """
    cur = "" if cur is None else str(cur)
    return cur.lower() in ("", "nan", "uncategorized")


# Ensure the target column exists in the spreadsheet and set default values
def _ensure_flat_col(df: pd.DataFrame, out_col: str, insert_at: int = 6):
    """Ensure the output category column exists and is of string type."""
    if out_col not in df.columns:
        insert_at = insert_at if insert_at <= len(df.columns) else len(df.columns)
        df.insert(insert_at, out_col, "")
    df[out_col] = df[out_col].astype(str).replace('', 'Uncategorized')


# ============================================================
# COCKPIT FLATTENING LOGIC
# ============================================================
# Standardize cockpit-related titles by removing punctuation and expanding abbreviations
def cockpit_normalize(title: str) -> str:
    t = _lower(title)
    t = re.sub(r"[——\-_/|]+", " ", t)
    t = re.sub(r"[(){}\[\],;:]", " ", t)
    t = re.sub(r"\bcpt\b", " captain ", t)
    t = re.sub(r"\bcapt\b", " captain ", t)
    t = re.sub(r"\bf\s*/\s*o\b", " first officer ", t)
    t = re.sub(r"\bf\.?o\.?\b", " first officer ", t)
    t = re.sub(r"\bsfo\b", " senior first officer ", t)
    t = re.sub(r"\bpic\b", " pilot in command ", t)
    t = re.sub(r"\btri\b", " type rating instructor ", t)
    t = re.sub(r"\btre\b", " type rating examiner ", t)
    t = re.sub(r"\bsfi\b", " synthetic flight instructor ", t)
    return _ws(t)


# List of keywords that strongly indicate a cockpit-related role
```

```
COCKPIT_WHITELIST = [
    r"\bcaptain\b", r"\bpilot in command\b", r"\baircraft commander\b",
    r"\b(first\s+officer|co-?\s?pilot|copilot|flight officer)\b",
    r"\bsenior\s+first\s+officer\b",
    r"\b(type rating|line training|flight
training|simulator|check|line|standards)\s+(instructor|examiner|captain|pilot|airman)\b",
    r"\b(tri|sfi|tre)\b",
]

# List of keywords to exclude to avoid misclassifying finance or office roles as cockpit
COCKPIT_BLACKLIST = [
    r"\bcfo\b", r"\bchief financial officer\b",
    r"\bfront office\b", r"\bfinance officer\b",
    r"\bflight operations\b(?!\s*(supervisor|manager|director)\b)",
    r"\bcaptain\s+america\b",
]

# Pattern to identify if a title is simply 'pilot' without further detail
PILOT_ALONE = re.compile(r"^\s*(airline\s+)?pilot\s*$")

# Hierarchical list of patterns to categorize cockpit roles into specific levels
COCKPIT_PATTERNS = [
    # NEW PATTERN - SIMPLIFIED AND PRIORITIZED
    ("5Cockpit, First Officer", [r"\bfirst\s+officer\b", r"\bfirst-?officer\b", r"\bfirstofficer\b",
r"\bfo\b"]),
    ("3Cockpit, Captain", [r"\bcaptain\b"]),
    ("4Cockpit, Senior First Officer",
     [r"\bsenior\s+first\s+officer\b", r"\bsenior\s+first-?officer\b", r"\bsenior\s+firstofficer\b",
      r"\bsenior\s+fo\b"]),
    ("1Cockpit, Examiner",
     [r"\btype rating examiner\b", r"\bline check captain\b", r"\bline check pilot\b", r"\bcheck
captain\b",
      r"\bcheck airman\b", r"\bproficiency check examiner\b", r"\bexaminer pilot\b",
r"\bstandards captain\b",
      r"\btre\b", r"\bcheck pilot\b"]),
    ("2Cockpit, Instructor",
     [r"\bflight instructor\b", r"\bsimulator instructor\b", r"\bline training captain\b",
r"\btraining captain\b",
      r"\binstructor pilot\b", r"\bflight training captain\b", r"\bsimulator check instructor\b",
      r"\bline instructor\b", r"\bpilot instructor\b", r"\bsynthetic flight instructor\b",
      r"\btype rating instructor\b", r"\btri\b", r"\bsfi\b", r"\binstructor\b"]),
    ("6Cockpit, Cadet", [r"\bcadet\b", r"\btrainee\s+pilot\b"]),
]


# Check if a title matches cockpit criteria and is not on the blacklist
def is_confident_cockpit(t: str) -> bool:
    return any(re.search(p, t) for p in COCKPIT_WHITELIST) and not any(re.search(n, t) for n
in COCKPIT_BLACKLIST)
```

```
# Assign a specific cockpit category label based on the job title
def classify_cockpit_flat(title_raw: str) -> Optional[str]:
    t = cockpit_normalize(title_raw)
    if PILOT_ALONE.match(t):
        return None
    if not is_confident_cockpit(t):
        return None
    for label, pats in COCKPIT_PATTERNS:
        for pat in pats:
            if re.search(pat, t):
                return label
    return None


# Iterate through the data and apply cockpit classification to eligible rows
def add_flat_cockpit(df: pd.DataFrame, title_col="position", out_col="category_flat") ->
pd.DataFrame:
    _ensure_flat_col(df, out_col, insert_at=5)
    titles = df[title_col].astype(str).tolist()
    flat = df[out_col].astype(str).tolist()

    for i, (t_raw, cur) in enumerate(zip(titles, flat)):
        if _should_update(cur):
            lab = classify_cockpit_flat(t_raw)
            if lab is not None:
                df.at[i, out_col] = lab
    return df


# ============================================================
# CABIN FLATTENING LOGIC
# ============================================================
# Standardize cabin-related titles by removing punctuation and expanding abbreviations
def cabin_normalize(title: str) -> str:
    t = _lower(title)
    t = re.sub(r"[——\-_/|]+", " ", t)
    t = re.sub(r"[(){}\[\],;:]", " ", t)
    t = re.sub(r"\bf\s*/\s*a\b", " flight attendant ", t)
    t = re.sub(r"\bfa\b(?=\s|$)", " flight attendant ", t)
    t = re.sub(r"\bflight\s*attend(ent|and|ent)?\b", " flight attendant ", t)
    return _ws(t)


# List of keywords that strongly indicate a cabin crew or inflight service role
CABIN_WHITELIST = [
    r"\bflight attendant\b", r"\bcabin crew\b", r"\bcabin\s+(team|service)\b",
    r"\bsteward(ess)?\b", r"\binflight\b|\bin[- ]?flight\b|\binflight\s+crew\b",
    r"\bpurser\b",
r"\b(inflight|cabin)\s+(supervisor|manager|trainer|instructor|coordinator|safety|duty|base)\b",
    r"\bflight attendant in charge\b|\bfa in charge\b|\blead (flight|cabin) attendant\b",
    r"\b(international|corporate)\s+flight attendant\b",
```

```
    r"\b(purser|steward(ess)?|cabin\s+crew|flight\s+attendant|crewmember)\b",
]

# List of keywords to exclude to avoid misclassifying general customer service or office roles
as cabin crew
CABIN_BLACKLIST = [
    r"\bcustomer service\b(?!.*\b(gate|station|airport)\b)",
    r"\b(call\s*center|csr)\b",
    r"\bfront office\b",
    r"\bfinance|financial\b",
    r"\bfacilities?\b",
    r"\bsecurity\b(?!.*\b(cabin|inflight)\b)",
    r"\bmanager\b(?!.*\b(cabin|inflight)\b)",
    r"\bservice attendant\b(?!.*\b(flight|cabin|inflight)\b)",
    r"\bcrew\b(?!.*\b(cabin|flight)\b)",
    r"\b(bachelor of arts|b.a.)\b",
    r"\b(attorney|aviation)\b"
]

# Hierarchical list of patterns to categorize cabin roles into specific levels
CABIN_PATTERNS = [
    ("5Cabin, Flight Attendant", [r"\bflight attendant\b", r"\bcabin crew\b|\bcrew member\b",
r"\bsteward(ess)?\b",
                                  r"\binflight crew\b|\binflight\b|\bin[- ]?flight\b", r"\bcrewmember\b"]),
    ("4Cabin, Purser", [r"\bpurser\b", r"\bflight service leader\b|\binflight service leader\b",
                        r"\bchief flight attendant\b|\bchief fa\b",
                        r"\blead flight attendant\b|\blead cabin attendant\b"]),
    ("3Cabin, Team Lead / Trainer", [r"\b(inflight|cabin)\s+(supervisor|duty manager|base
supervisor)\b",
                                     r"\bcabin (team lead|team leader|service supervisor|appearance
supervisor)\b",
                                     r"\b(inflight|cabin)\s+(instructor|trainer)\b",
                                     r"\b(cabin|inflight)\s+safety\s+(coordinator|trainer)\b",
                                     r"\bfa in charge\b|\bflight attendant in charge\b",
                                     r"\blead (flight|cabin) attendant\b"]),
    ("2Cabin, Manager", [r"\b(cabin|inflight)\s+(manager|operations manager|duty
manager|base manager)\b",
                         r"\bmanager,\s*inflight (training|services?)\b", r"\b(inflight|cabin)\s+service
manager\b",
                         r"\bcabin crew manager\b"]),
    ("1Cabin, Head", [r"\bhead of (cabin|inflight)\b", r"\bdirector (of )?(cabin|inflight)\b",
                      r"\bsenior manager (cabin|inflight) (ops|operations)?\b",
r"\b(inflight|cabin)\s+director\b"]),
    ("6Cabin, Cadet", [r"\b(cabin|flight attendant)\s+trainee\b", r"\binflight\s+trainee\b",
r"\bcadet\b"]),
]


# Check if a title matches cabin criteria and is not on the blacklist
def is_confident_cabin(t: str) -> bool:
```

```
    return any(re.search(p, t) for p in CABIN_WHITELIST) and not any(re.search(n, t) for n in
CABIN_BLACKLIST)


# Assign a specific cabin category label based on the job title
def classify_cabin_flat(title_raw: str) -> Optional[str]:
    t = cabin_normalize(title_raw)
    if not is_confident_cabin(t):
        return None
    for label, pats in CABIN_PATTERNS:
        for pat in pats:
            if re.search(pat, t):
                return label
    return "Cabin, Flight Attendant"


# Iterate through the data and apply cabin classification to eligible rows
def add_flat_cabin(df: pd.DataFrame, title_col="position", out_col="category_flat") ->
pd.DataFrame:
    _ensure_flat_col(df, out_col, insert_at=5)
    titles = df[title_col].astype(str).tolist()
    flat = df[out_col].astype(str).tolist()

    for i, (t_raw, cur) in enumerate(zip(titles, flat)):
        if _should_update(cur):
            lab = classify_cabin_flat(t_raw)
            if lab is not None:
                df.at[i, out_col] = lab
    return df


# ============================================================
# NEW GROUND OPS HIERARCHY (Levels 4 -> 1)
# ============================================================
# Standardize ground operations titles by cleaning text and expanding acronyms
def _normalize_ground(text: str) -> str:
    t = _lower(text)
    t = re.sub(r"[——\-_/|]+", " ", t)
    t = re.sub(r"[(){}\[\],;:]", " ", t)
    # common acronyms / typos
    t = re.sub(r"\bcsa\b", " customer service agent ", t)
    t = re.sub(r"\bcsr\b", " customer service representative ", t)
    t = re.sub(r"\bgsa\b", " guest service agent ", t)
    t = re.sub(r"\bpsa\b", " passenger service agent ", t)
    t = re.sub(r"\bpsr\b", " passenger service representative ", t)
    t = re.sub(r"\bcheckin\b", " check in ", t)
    t = re.sub(r"\bticketting\b", " ticketing ", t)
    t = re.sub(r"\bcustomer\s+care\b", " customer care ", t)
    t = re.sub(r"\bloadmaster\b", " loadmaster ", t)
    return _ws(t)
```

```
# List of patterns for entry-level ground operations roles
GROUND_LV4_PATTERNS = [  # Agents
    r"\b(customer|passenger|guest)\s+service\s+(agent|representative)\b",
    r"\b(service|ticket|gate|boarding|check\s*in|crew)\s+agent\b",
    r"\b(passenger|airport)\s+services?\s+(agent|representative)\b",
    r"\bstation\s+agent\b", r"\bops?\s+agent\b",
    r"\b(baggage|ramp)\s+(agent|handler)\b", r"\bcargo\s+agent\b",
    r"\bconcierge\b|\bambassador\b|\blounge\b|\bwheelchair\b|\bskycap\b|\bporter\b",
    r"\blost\s*&?\s*found\b|\bbaggage\s+services?\b",
    r"\bairport\s+agent\b",
    r"\bloadmaster\b",
]
# List of patterns for ground operations leadership roles
GROUND_LV3_PATTERNS = [  # Team Leads, Duty Manager
    r"\bteam\s*lead(er)?\b|\blead\b",
    r"\bsupervisor\b",
    r"\b(duty|shift)\s+manager\b",
    r"\bcustomer\s+service\s+manager\b",
]
# List of patterns for station-level management roles
GROUND_LV2_PATTERNS = [  # Station Manager
    r"\bstation\s+manager\b",
    r"\b(airport|terminal)\s+operations?\s+manager\b",
    r"\bmanager\s+airport\s+operations\b",
]
# List of patterns for regional-level ground operations management
GROUND_LV1_PATTERNS = [  # Regional Manager
    r"\bregional\s+(ground\s+)?operations?\s+manager\b",
    r"\bregional\s+station\s+manager\b",
    r"\bhead\s+of\s+region\b|\bregion\s+head\b",
]


# Match ground operations titles to their respective hierarchy levels
def classify_ground_ops_levels(title: str) -> Optional[str]:
    t = _normalize_ground(title)
    for pat in GROUND_LV1_PATTERNS:
        if re.search(pat, t): return "1Ground, Regional Manager"
    for pat in GROUND_LV2_PATTERNS:
        if re.search(pat, t): return "2Ground, Station Manager"
    for pat in GROUND_LV3_PATTERNS:
        if re.search(pat, t): return "3Ground, Team Lead / Duty Manager"
    for pat in GROUND_LV4_PATTERNS:
        if re.search(pat, t): return "4Ground, Agent"
    return None


# ============================================================
# NEW IT CATEGORY
# ============================================================
```

```
# List of keywords and technical terms that identify information technology roles
IT_PATTERNS = [
    r"\bdeveloper\b", r"\bsoftware\s+developer\b", r"\bsoftware\s+engineer\b",
    r"\bprogrammer\b", r"\bcoder\b",
    r"\bapplication\s+developer\b", r"\bapp\s+developer\b", r"\bmobile\s+developer\b",
    r"\b(full\s*stack|frontend|front[- ]end|backend|back[- ]end)\s+(developer|engineer)\b",
    r"\bjava\b", r"\bpython\b", r"\bscala\b", r"\bc\+\+\b", r"\bc#\b", r"\b.net\b",
r"\bnode(\.js)?\b",
    r"\bphp\b", r"\bruby\b", r"\bkotlin\b", r"\bswift\b",
    r"\bweb\s+developer\b", r"\bweb\s+engineer\b", r"\bwordpress\b", r"\bdrupal\b",
    r"\bcloud\b", r"\baws\b", r"\bazure\b", r"\bgcp\b", r"\bkubernetes\b", r"\bdocker\b",
    r"\bdevops\b", r"\bsite\s+reliability\s+engineer\b", r"\bsre\b",
    r"\bdatabase\s+administrator\b", r"\bdba\b", r"\bdata\s+engineer\b",
    r"\bETL\b", r"\bdata\s+integration\b",
    r"\bIT\s+support\b", r"\bhelp\s*desk\b", r"\bIT\s+technician\b",
    r"\bIT\s+administrator\b", r"\bsystem\s+administrator\b", r"\bsysadmin\b",
    r"\bnetwork\s+engineer\b", r"\bnetwork\s+administrator\b", r"\bnetwork\s+architect\b",
    r"\binfrastructure\b", r"\bsystems?\s+engineer\b", r"\bplatform\s+engineer\b",
    r"\bsecurity\s+engineer\b", r"\bIT\s+security\b", r"\bcyber\s+security\b",
    r"\binformation\s+security\b", r"\binfosec\b", r"\bpenetration\s+tester\b",
r"\bpen\s*tester\b",
    r"\bsecurity\s+analyst\b", r"\bSOC\s+analyst\b", r"\bsecurity\s+consultant\b",
    r"\bIT\s+architect\b", r"\bsolution\s+architect\b", r"\benterprise\s+architect\b",
    r"\bIT\s+consultant\b", r"\btechnical\s+consultant\b",
    r"\bbusiness\s+analyst\b(?=.*\bIT\b)", r"\bIT\s+business\s+analyst\b",
    r"\bERP\b", r"\bSAP\b", r"\bOracle\b", r"\bMicrosoft\s+Dynamics\b",
    r"\bapplication\s+support\b", r"\bIT\s+operations\b",
    r"\bIT\s+project\s+manager\b", r"\bIT\s+program\s+manager\b",
    r"\bIT\s+specialist\b", r"\bIT\s+expert\b", r"\bIT\s+coordinator\b",
    r"\bquality\s+assurance\b", r"\bQA\b", r"\btest\s+engineer\b", r"\btester\b",
    r"\bUI\b", r"\bUX\b", r"\bUI/UX\b", r"\buser\s+experience\b", r"\buser\s+interface\b",
    r"\bproduct\s+owner\b", r"\bproduct\s+manager\b(?=.*\bIT\b)",
    r"\bIT\s+trainer\b", r"\bIT\s+instructor\b",
    r"\bdesktop\s+engineer\b",
    r"\bsystem\s+architect\b", r"\bsoftware\s+architect\b", r"\bsolutions\s+architect\b"
]


# Check if a title matches the defined IT patterns
def classify_it_flat(title: str) -> Optional[str]:
    t = _lower(title)
    if any(re.search(pat, t) for pat in IT_PATTERNS):
        return "IT"
    return None


# ============================================================
# NEW TECHNICAL/MAINTENANCE CATEGORY
# ============================================================
# List of keywords related to aircraft maintenance, engineering, and technical operations
TECHNIK_PATTERNS = [
```

```
    # Mechaniker allgemein
    r"\bmechanic\b", r"\bmaintenance\b", r"\btechnician\b", r"\btech\b",
    r"\bline\s+maintenance\b", r"\bbase\s+maintenance\b",
    r"\bMRO\b", r"\bhangar\b",

    # Ingenieure allgemein
    r"\bengineer\b", r"\bengineering\b", r"\baviation\s+engineer\b",
    r"\bmaintenance\s+engineer\b", r"\breliability\s+engineer\b",
    r"\bflight\s+engineer\b", r"\bfleet\s+engineer\b",
    r"\baircraft\s+engineer\b", r"\blicensed\s+engineer\b",

    # Lizenzen & Zulassungen
    r"\bB1\b", r"\bB2\b", r"\bB1\/B2\b", r"\bCAT\s*B1\b", r"\bCAT\s*B2\b",
    r"\bEASA\s*B1\b", r"\bEASA\s*B2\b",
    r"\bavionics\b", r"\bstructures?\s+engineer\b",

    # Spezifische Rollen
    r"\bAME\b", r"\baircraft\s+maintenance\s+engineer\b",
    r"\bCAMO\b", r"\bcontinuing\s+airworthiness\b",
    r"\bengine\s+engineer\b", r"\bpowerplant\b", r"\bA&P\b", r"\bA\&P\b",
    r"\bpropulsion\b", r"\bfuel\s+systems?\b",
    r"\bstructures?\s+mechanic\b", r"\bcomposite\s+mechanic\b",
    r"\bsheet\s+metal\b", r"\bNDT\b", r"\bnon[- ]destructive\b",

    # Führungsrollen Technik
    r"\bmaintenance\s+manager\b", r"\bmaintenance\s+supervisor\b",
    r"\bmaintenance\s+planner\b", r"\bmaintenance\s+controller\b",
    r"\bengineering\s+manager\b", r"\bchief\s+engineer\b",
    r"\bVP\s+engineering\b", r"\bdirector\s+engineering\b",
    r"\bdirector\s+maintenance\b", r"\bhead\s+of\s+engineering\b",
]


# Check if a title matches technical or maintenance-related criteria
def classify_technical(title: str) -> Optional[str]:
    t = _lower(title)
    if any(re.search(pat, t) for pat in TECHNIK_PATTERNS):
        return "Technical/Maintenance"
    return None


# ============================================================
# CORPORATE MANAGEMENT LEVELS (-1 -> 6)
# ============================================================
# Standardize corporate titles by expanding management-related abbreviations
def _normalize_corporate(text: str) -> str:
    t = _lower(text)
    t = re.sub(r"[——\-_/|]+", " ", t)
    t = re.sub(r"[(){}\[\],;:]", " ", t)
    # unify common abbreviations (English only)
    t = re.sub(r"\bsr\.?\b", " senior ", t)
```

```
    t = re.sub(r"\bsnr\.?\b", " senior ", t)
    t = re.sub(r"\bjr\.?\b", " junior ", t)
    t = re.sub(r"\bsvp\b", " senior vice president ", t)
    t = re.sub(r"\bevp\b", " executive vice president ", t)
    t = re.sub(r"\bvp\b", " vice president ", t)
    t = re.sub(r"\bmd\b", " managing director ", t)
    t = re.sub(r"\bcfo\b", " chief financial officer ", t)
    t = re.sub(r"\bcoo\b", " chief operating officer ", t)
    t = re.sub(r"\bcto\b", " chief technology officer ", t)
    t = re.sub(r"\bcio\b", " chief information officer ", t)
    t = re.sub(r"\bcmo\b", " chief marketing officer ", t)
    t = re.sub(r"\bcco\b", " chief commercial officer ", t)
    t = re.sub(r"\bcro\b", " chief revenue officer ", t)
    t = re.sub(r"\bceo\b", " chief executive officer ", t)
    t = _ws(t)
    return t


# Guard pattern to prevent ground-based staff from being labeled as corporate management
GROUND_CONTEXT_GUARD = re.compile(

r"\b(station|duty|ramp|baggage|gate|ticket(ing)?|check\s*in|passenger\s+service|customer\s+se
rvice|ops?\s+agent|"

r"wheelchair|skycap|porter|apron|pushback|tug|load\s*control|weight\s*and\s*balance|airport\
s+operations?)\b",
    re.IGNORECASE
)

# Hierarchical list of corporate levels from board members down to interns
CORP_LEVELS = [
    ("-1Management, Advisory Board", [
        r"\badvisory\s+board\b", r"\bboard\s+of\s+advisors\b",
        r"\bsupervisory\s+board\b", r"\bboard\s+member\b",
        r"\bnon[-\s]?executive\s+director\b", r"\bboard\s+of\s+directors?\b",
        r"\bboard\s+(chair|chairman|chairwoman)\b", r"\btrustee\b"
    ]),
    ("0Management, C-Level / Executive", [

r"\bchief\s+(executive|financial|operating|technology|information|marketing|human\s+resourc
es|people|"

r"commercial|security|strategy|sustainability|legal|administrative|risk|revenue)\s+officer\b",
        r"\bpresident\b(?!\s*(of|student))", r"\bexecutive\s+committee\b",
r"\bexecutive\s+board\b"
    ]),
    ("1Management, Senior Vice President", [
        r"\bsenior\s+vice\s+president\b", r"\bexecutive\s+vice\s+president\b",
        r"\bgroup\s+senior\s+vice\s+president\b", r"\bglobal\s+senior\s+vice\s+president\b",
        # NEW: Head of ... (generic, kept here as Level 1)
        r"\bhead\s+of\b",
```

```
        r"\bhead\b\s*(?:-|,|\sof\b)",
    ]),
    ("2Management, Vice President", [
        r"\bvice\s+president\b", r"\bassistant\s+vice\s+president\b",
r"\bassociate\s+vice\s+president\b"
    ]),
    ("3Management, Director / Senior Manager", [
        r"\bsenior\s+director\b", r"\bmanaging\s+director\b", r"\bexecutive\s+director\b",
        r"\bregional\s+director\b", r"\barea\s+director\b", r"\bglobal\s+director\b",
        r"\bdeputy\s+director\b",
        r"\bsenior\s+manager\b", r"\bdeputy\s+manager\b", r"\bdirector\b"
    ]),
    ("4Management, Manager / Lead / Coordinator / Supervisor", [
        r"\bmanager\b", r"\bteam\s+lead(er)?\b", r"\blead\b", r"\bco[-
\s]?ordinator\b|\bcoordinator\b",
        r"\bsupervisor\b", r"\bofficer\b(?!\s*first)", r"\brepresentative\b",
r"\bsubject\s+matter\s+expert\b"
    ]),
    ("5Management, Professional / Staff", [
        r"\bgeneralist\b", r"\bspecialist\b", r"\banalyst\b", r"\bscheduler\b|\bscheduling\b",
        r"\bplanner\b|\bplanning\b", r"\bdispatch(er)?\\b",
        r"\bassistant\b|\bassistance\b|\bassociate\b",
r"\brecruiter\b|\btalent\s+acquisition\b|\bsourcer\b",
        r"\bpurchaser\b|\bbuyer\b|\bprocurement\b", r"\bfacilitator\b", r"\bauditor\b",
        r"\bdeveloper\b", r"\bcontroller\b|\bcontrolling\b", r"\bconsultant\b",
        r"\bsupport\b|\badmin(istrator)?\b|\badministrative\s+assistant\b",
r"\bstaff\b|\bemployee\b",
        r"\badvisor\b",
        r"\bengineer\b"
    ]),
    ("6Management, Junior / Entry", [
        r"\bintern(ship)?\b", r"\bapprentice(ship)?\b", r"\btrainee\b", r"\bjunior\b",
        r"\bworking\s+student\b|\bstudent\s+assistant\b|\bstagiaire\b"
    ]),
]


# Match corporate titles to their management levels while avoiding ground-ops
misclassification
def classify_corporate_levels(title: str) -> Optional[str]:
    t = _normalize_corporate(title)
    # avoid corporate if clear ground/station context is present
    if GROUND_CONTEXT_GUARD.search(t):
        return None
    for label, pats in CORP_LEVELS:
        for pat in pats:
            if re.search(pat, t):
                return label
    return None
```

```
# ============================================================
# INTEGRATOR (run in order): Cockpit -> Cabin -> Ground/IT/Technical/Corporate
# ============================================================
# Main orchestration function to apply all classification rules in a specific priority order
def flatten_airline_roles(df: pd.DataFrame,
                          title_col="position", in_cat_col="category", out_col="category_flat") ->
pd.DataFrame:
    """
    Applies Cockpit, then Cabin, then the new Ground Ops, IT, and Corporate Management
flattening.
    """
    # Ensure the output column exists
    _ensure_flat_col(df, out_col, insert_at=6)

    # 1. Apply Cockpit and Cabin roles first (these are very specific and should take
precedence)
    df = add_flat_cockpit(df, title_col=title_col, out_col=out_col)
    df = add_flat_cabin(df, title_col=title_col, out_col=out_col)

    # 2. Now, iterate and apply the new Ground Ops, IT, and Corporate logic
    titles = df[title_col].astype(str).tolist()
    flat_categories = df[out_col].astype(str).tolist()

    for i, (raw_title, current_cat) in enumerate(zip(titles, flat_categories)):
        if _should_update(current_cat):
            # Check for Ground Ops levels first
            ground_label = classify_ground_ops_levels(raw_title)
            if ground_label:
                df.at[i, out_col] = ground_label
                continue

            # If not Ground Ops, check for IT roles
            it_label = classify_it_flat(raw_title)
            if it_label:
                df.at[i, out_col] = it_label
                continue

            # If not IT, check for Technical/Maintenance roles
            tech_label = classify_technical(raw_title)
            if tech_label:
                df.at[i, out_col] = tech_label
                continue

            # If not Technical, check for Corporate Management levels
            corporate_label = classify_corporate_levels(raw_title)
            if corporate_label:
                df.at[i, out_col] = corporate_label
                continue

    # Finally, replace any remaining 'Uncategorized' values with the original 'category' column
```

```
    df[out_col] = df.apply(lambda row: row[in_cat_col] if row[out_col] == 'Uncategorized'
else row[out_col], axis=1)

    return df


# Load the spreadsheet, perform the flattening process, and save the result
def process_single_file(input_file, output_folder, title_columns=TITLE_COLUMNS):
    timestamp = datetime.now().strftime('%Y.%m.%d.%H.%M')
    file_path = input_file
    filename = os.path.basename(file_path)

    print(f" Processing file: {filename}")
    try:
        # Read the Excel file and make all column names lowercase for consistency
        df = pd.read_excel(file_path)
        df = df.rename(columns={c: c.lower() for c in df.columns})

        # Identify which column contains the job titles
        title_col = next((c for c in title_columns if c in df.columns), None)
        if not title_col:
            print(f" Skipping {filename}: No title column found from {title_columns}")
            return

        # Check for the 'category' column and create it if it doesn't exist
        if 'category' not in df.columns:
            print(" 'category' column not found. Creating a new 'category' column with
'Uncategorized' values.")
            df['category'] = 'Uncategorized'

        # Step 2: Apply the flattening logic from your script
        # The new logic is now a part of the single flatten_airline_roles function.
        df = flatten_airline_roles(df, title_col=title_col, out_col=FLATTENED_COLUMN,
in_cat_col='category')

        # Generate the output filename and save the final spreadsheet
        output_filename = f'flattened_airline_jobs_neo_{timestamp}.xlsx'
        output_path = os.path.join(output_folder, output_filename)

        print(f"\n All files processed successfully. Total rows processed: {len(df)}.")
        print(f" Saving the flattened data to: {output_path}")
        df.to_excel(output_path, index=False)
        print(" Script finished successfully!")

    except Exception as e:
        # Provide feedback if the file processing fails
        print(f" ERROR: Could not process {filename}: {e}. Skipping.")


# Execute the script if run as the main program
if __name__ == "__main__":
```

```
    process_single_file(input_file, output_folder)
```

A.1.5. Vertical Categorization

```
import pandas as pd
import os
import re
from datetime import datetime
from typing import Tuple, Any

# CONFIGURATION

# 1. Input: The flattened file from your previous step (REQUIRED CHANGE)
# This variable stores the path to your source Excel data
INPUT_FILE_PATH =
r'/Users/Christoph/Desktop/MasterarbeitUniWien/4flatten/flattened_airline_jobs_neo_2025.1
0.23.12.12.xlsx'

# 2. Output Directory (REQUIRED CHANGE)
# This variable defines where the processed file will be saved
OUTPUT_DIR = r'/Users/Christoph/Desktop/MasterarbeitUniWien/5levels_aviation'

# Column Names (DO NOT CHANGE THESE)
# These identify the specific columns in your Excel sheet for titles, categories, and the new
levels
TITLE_COLUMN = 'exp_1_title'
CATEGORY_COLUMN = 'category_neo'
LEVEL_COLUMN = 'airline_level_matrix_classified'  # Standard column name

# Creates the output directory if it doesn't exist yet
os.makedirs(OUTPUT_DIR, exist_ok=True)

# HELPERS & KEYWORDS

def clean_text(text: Any) -> str:
    """Standardizes text for regex matching."""
    # Converts text to lowercase, removes special characters, and adds padding for matching
    if not isinstance(text, str):
        return ""
    text = str(text).lower().strip()
    # Remove punctuation/symbols for safer matching, keeping only word chars and spaces
    text = re.sub(r'[^\w\s]', ' ', text)
    text = re.sub(r'\s+', ' ', text).strip()
    return " " + text + " "  # Pad with spaces for word boundary matching


# Universal keywords for the new X.1 Junior/Trainee tiers
# Keywords used to identify entry-level or student positions
JUNIOR_KEYWORDS = [
    ' intern ', ' apprentice ', ' trainee ', ' cadet ', ' student ', ' praktikum '
]
```

```
# Manual Exclusions (Per User Request)
# Specific words that signal a job should be ignored or marked unclassified
MANUAL_EXCLUSION_KEYWORDS = [
    ' non flying ', ' non-flying ', ' back office ', ' back-office ',
    ' ground staff ', ' recruitment '
]

# Technic/Maintenance specific keywords (L45-L49 track)
# List of terms related to aircraft engineering and maintenance
TECHNIC_KEYWORDS = [
    ' mechanic ', ' maintenance ', ' technician ', ' tech ', ' line maintenance ',
    ' base maintenance ', ' mro ', ' hangar ', ' engineer ', ' engineering ',
    ' aviation engineer ', ' maintenance engineer ', ' reliability engineer ',
    ' flight engineer ', ' fleet engineer ', ' aircraft engineer ', ' licensed engineer ',
    ' b1 ', ' b2 ', ' b1 b2 ', ' cat b1 ', ' cat b2 ', ' easa b1 ', ' easa b2 ',
    ' avionics ', ' structures engineer ', ' ame ', ' camo ', ' continuing airworthiness ',
    ' engine engineer ', ' powerplant ', ' a&p ', ' a & p ', ' propulsion ', ' fuel system ',
    ' sheet metal ', ' ndt ', ' non destructive ', ' spares ', ' parts ', ' inventory ',
    ' technical operations ', ' load planner ', ' load planning '
]

# LOGIC: TRACK DETECTION

def get_job_track(title_clean, category_clean):
    """
    Determines job track: Technic, Cockpit, Cabin, Ground, or Corporate.
    Priority order: Corporate Override (The Fix) > Technic > Flight Operations > Ground Ops
> Corporate
    """
    # Override 0: FORCE CORPORATE for specific support/universal titles, overriding
category.
    # THIS IS THE PRIMARY FIX TO PREVENT L39 SPILLAGE.
    if any(x in title_clean for x in [' administrative ', ' executive assistant ', ' recruiter ', ' human
resource ',
                                      ' administrator ', ' technology ', ' it ', ' support specialist ', ' analyst ',
                                      ' planner ', ' developer ', ' financial ', ' sales ', ' quality ', ' legal ',
                                      ' paralegal ', ' accounting ', ' procurement ', ' business partner ',
                                      ' compliance ',
                                      ' auditor ', ' technical writer ', ' product owner ', ' lead ', ' manager ',
                                      ' coordinator ']):
        # Check for strong Ground Ops terms to prevent over-filtering legitimate Ground Ops
Managers/Leads
        if not any(x in title_clean for x in
                   [' station manager ', ' ramp manager ', ' ground handling manager ', ' duty officer ',
' shift manager ', ' shift lead ']):
            return "Corporate"

    # 1. TECHNIC/MAINTENANCE TRACK
    if any(x in title_clean for x in TECHNIC_KEYWORDS):
        return "Technic/Maintenance"
```

```
    if 'technic' in category_clean or 'maintenance' in category_clean:
        return "Technic/Maintenance"

    # 2. COCKPIT TRACK
    if any(x in title_clean for x in
           [' captain ', ' first officer ', ' second officer ', ' pilot ', ' aviator ', ' tre ', ' tri ',
            ' type rating examiner ', ' type rating instructor ', ' flight instructor ', ' simulation
instructor ', ' sim instructor ']):
        return "Cockpit"
    if 'cockpit' in category_clean or 'pilot' in category_clean:
        return "Cockpit"

    # 3. CABIN TRACK
    if any(x in title_clean for x in
           [' flight attendant ', ' stewardess ', ' steward ', ' cabin crew ', ' purser ', ' inflight manager
']):
        return "Cabin"
    if 'cabin' in category_clean or 'flight attendant' in category_clean or 'inflight' in
category_clean:
        return "Cabin"

    # 4. GROUND OPS TRACK
    ground_keywords = [
        ' dispatcher ', ' flight dispatcher ', ' operations controller ', ' ops controller ',
        ' station manager ', ' station supervisor ', ' duty manager ',
        ' ramp ', ' gate ', ' ticket agent ', ' check in ', ' baggage ',
        ' loadmaster ', ' turnaround ', ' ground handling ', ' customer service agent ',
        ' airport operations ', ' airport services ', ' station officer ', ' operations agent '
    ]
    if any(x in title_clean for x in ground_keywords):
        return "Ground Ops"
    if 'ground' in category_clean or 'airport' in category_clean or 'station' in category_clean:
        return "Ground Ops"

    # 5. DEFAULT: CORPORATE
    return "Corporate"


# LOGIC: LEVEL ASSIGNMENT


def assign_airline_matrix_level(row) -> Tuple[str, str]:
    """
    Assigns a level (00-49.1) based on Title, Category, and Track logic.
    Returns: (Level Code, Derived Track)
    """
    title_raw = row.get(TITLE_COLUMN, '')
    title = clean_text(title_raw)
    category = clean_text(row.get(CATEGORY_COLUMN, ''))
```

```
    # PHASE 0: Blank/Invalid Titles & Manual Exclusions

    # Level 13: Exclusion/Blank/Invalid Title
    if not title or pd.isna(title_raw) or str(title_raw).strip() == '':
        return "Level 13 (Exclusion/Blank/Invalid Title)", "Unclassified"

    # Level 13: Non-aviation Instructor Logic
    if " instructor " in title and not any(x in title for x in [' flight ', ' sim ', ' type ', ' tri ', ' tre ']):
        return "Level 13 (Exclusion/Blank/Invalid Title)", "Unclassified"

    # Check for manual exclusions (General words like back-office etc.)
    if any(x in title for x in MANUAL_EXCLUSION_KEYWORDS):
        return "Level 13 (Exclusion/Blank/Invalid Title)", "Unclassified"


    # PHASE 1: Universal Top Management (Levels 00 - 06)


    # LEVEL 00: Board of Directors
    if any(x in title for x in [' board of ', ' chairman ', ' chairperson ', ' supervisory board ']):
        if ' assistant ' in title: return "Level 08 (Specialist/Analyst/Expert)", "Corporate"
        return "Level 00 (Board)", "Universal"

    # LEVEL 01: CEO & President
    if any(x in title for x in [' ceo ', ' chief executive officer ', ' president ']):
        if ' assistant ' in title: return "Level 08 (Specialist/Analyst/Expert)", "Corporate"
        if not any(x in title for x in [' svp ', ' vp ', ' vice president ']):
            return "Level 01 (CEO)", "Universal"

    # LEVEL 02: Executive/Board of Mgmt
    if any(x in title for x in [' cfo ', ' coo ', ' cto ', ' cmo ', ' cio ', ' evp ',
                                ' executive vice president ']) and ' assistant ' not in title:
        return "Level 02 (Executive/Board of Mgmt)", "Universal"

    # LEVEL 03: SVP
    if any(x in title for x in [' senior vice president ', ' svp ', ' senior vp ', ' sr vp ', ' sr vice
president ']):
        if ' assistant ' in title: return "Level 08 (Specialist/Analyst/Expert)", "Corporate"
        return "Level 03 (SVP)", "Universal"

    # LEVEL 04: VP/MD (Universal)
    if any(x in title for x in [' vp ', ' vice president ', ' managing director ']):
        if ' assistant ' in title: return "Level 08 (Specialist/Analyst/Expert)", "Corporate"
        return "Level 04 (VP/MD)", "Universal"

    # LEVEL 05: Senior Director (Includes Head of ground/flight/maintenance)
    if any(x in title for x in [' senior director ', ' sr director ', ' sr. director ', ' general manager ',
                                ' head of maintenance ', ' head of flight ', ' head of ground ']):
        if ' assistant ' in title: return "Level 08 (Specialist/Analyst/Expert)", "Corporate"
        return "Level 05 (Sr. Director/GM)", "Universal"
```

```
    # LEVEL 06: Director / Regional Manager
    if any(x in title for x in [' director ', ' head of ', ' regional manager ', ' country manager ']):
        if ' assistant ' in title: return "Level 08 (Specialist/Analyst/Expert)", "Corporate"
        return "Level 06 (Director/Head of)", "Universal"


    # PHASE 2: Track-Specific Logic

    track = get_job_track(title, category)

    # >>> TRACK: TECHNIC / MAINTENANCE <<<
    if track == "Technic/Maintenance":
        # L47: Specialized Manager / Shift Lead / Tech Ops Fix
        if (any(x in title for x in [' manager ', ' mngr ', ' lead ', ' shift ', ' supervisor ])) and \
           (any(x in title for x in [' camo ', ' airworthiness ', ' line maintenance ', ' airframe ', '
engine ',
                                    ' structure ', ' avionics ', ' technical fleet ', ' spares ', ' parts ',
                                    ' inventory ', ' technical operations ', ' load planner ', ' load planning ])):
            return "Level 47 (Specialized Manager)", track
        if 'maintenance manager' in title or 'engineering manager' in title:
            return "Level 47 (Specialized Manager)", track

        # L48: Shift Lead/Supervisor/Planner/Controller
        if any(x in title for x in [' shift lead ', ' shift manager ', ' supervisor ', ' maintenance
controller ',
                                    ' maintenance planner ', ' auditor ']):
            return "Level 48 (Lead/Supervisor/Planner)", track

        # L49.1: Apprentice/Intern
        if any(x in title for x in JUNIOR_KEYWORDS):
            return "Level 49.1 (Technic Apprentice/Intern)", track

        # L49: Technicians, Engineers, Licensed Roles
        return "Level 49 (Technician/Engineer/Licensed)", track

    # >>> TRACK: COCKPIT <<<
    elif track == "Cockpit":
        # L17: TRI/TRE/Instructor
        if any(x in title for x in [' tre ', ' tri ', ' type rating examiner ', ' type rating instructor ',
                                    ' examiner ', ' flight instructor ', ' simulation instructor ', ' sim instructor
']):
            return "Level 17 (TRE/TRI/Instructor)", track
        # L18: Captain
        if any(x in title for x in [' captain ', ' cpt ', ' commander ', ' pic ', ' lead pilot ']):
            return "Level 18 (Captain)", track
        # L19: FO/SFO/Cadet
        return "Level 19 (FO/SFO/Cadet)", track

    # >>> TRACK: CABIN <<<
    elif track == "Cabin":
        # L27: Trainer
```

```
        if ' trainer ' in title:
            return "Level 27 (Trainer)", track
        # L28: Purser
        if any(x in title for x in [' purser ', ' senior flight attendant ', ' cabin manager ', ' csm ', '
supervisor ']):
            return "Level 28 (Purser/Sr FA)", track
        # L29: Flight Attendant
        return "Level 29 (Flight Attendant)", track

    # >>> TRACK: GROUND OPS <<<
    elif track == "Ground Ops":
        # L37: Station Manager/Ops Manager
        if ' station manager ' in title or ' station lead ' in title or ' operations manager ' in title:
            return "Level 37 (Station Manager/Ops Mgr)", track
        # L38: Supervisor/Coordinator
        if any(x in title for x in [' duty manager ', ' shift manager ', ' dispatcher ', ' coordinator ', '
supervisor ', ' auditor ']):
            return "Level 38 (Mgr/Dispatcher/Coordinator)", track
        # L39.1: Ground Apprentice
        if any(x in title for x in JUNIOR_KEYWORDS):
            return "Level 39.1 (Ground Ops Apprentice/Intern)", track
        # L39: Standard Ground Agent
        return "Level 39 (Clerk/Agent/Assistant)", track

    # >>> TRACK: CORPORATE <<<
    else:
        # Specialist/Analyst check for Assistants
        if ' assistant ' in title:
            return "Level 08 (Specialist/Analyst/Expert)", "Corporate"
        # L07: Manager / Lead
        if any(x in title for x in [' manager ', ' lead ', ' principal ', ' assistant manager ']):
            if any(x in title for x in [' project manager ', ' product manager ']):
                return "Level 08 (Specialist/Analyst/Expert)", "Corporate"
            return "Level 07 (Manager/Lead/Principal)", "Corporate"
        # L09: Corporate Junior
        if any(x in title for x in JUNIOR_KEYWORDS) or ' junior ' in title:
            return "Level 09 (Intern/Junior/Apprentice)", "Corporate"
        # L08: Default Specialist
        return "Level 08 (Specialist/Analyst/Expert)", "Corporate"

# MAIN EXECUTION

def main():
    """Execution entry point that handles file reading and processing logic."""
    timestamp = datetime.now().strftime("%Y.%m.%d_%H.%M")

    # Check if the input file exists
    if not os.path.exists(INPUT_FILE_PATH):
        print(f" Error: Input file not found: {INPUT_FILE_PATH}")
        print("Please check and update the 'INPUT_FILE_PATH' variable in the configuration
section.")
```

```
        return

    try:
        print(f"📖 Reading file: {os.path.basename(INPUT_FILE_PATH)}...")
        df = pd.read_excel(INPUT_FILE_PATH)
        df.columns = [c.lower().strip() for c in df.columns]

        # Check for column existence
        if TITLE_COLUMN not in df.columns:
            print(f" Error: Missing column '{TITLE_COLUMN}'. Available columns:
{list(df.columns)}")
            return

        print(" Applying Refined Matrix Logic...")
        # Apply the logic row by row
        results = df.apply(assign_airline_matrix_level, axis=1)

        # Unpack results into two new columns
        level_values = [r[0] for r in results]
        track_values = [r[1] for r in results]

        # --- COLUMN INSERTION (RETAINED VERBOSE LOGIC) ---
        # Inserts the new track and level information back into the data
        try:
            target_index = df.columns.get_loc(TITLE_COLUMN)
            df.insert(target_index, 'derived_track_final', track_values)
            df.insert(target_index + 1, LEVEL_COLUMN, level_values)
        except Exception:
            df['derived_track_final'] = track_values
            df[LEVEL_COLUMN] = level_values

        # --- VALIDATION STATS (RETAINED VERBOSE LOGIC) ---
        # Calculates summary statistics for the classification process
        total = len(df)
        unclassified = len(df[df[LEVEL_COLUMN].str.contains('Level 13')])
        classified = total - unclassified
        classification_rate = (classified / total) * 100 if total > 0 else 0

        # Calculate the new L39 count for comparison
        new_l39_count = len(df[df[LEVEL_COLUMN].str.contains('Level 39 ')])

        print("\n============================================================")
        print(f" CLASSIFICATION COMPLETE")
        print(f"Total Entries: {total}")
        print(f"Entries Classified (Operational + Corporate): {classified}")
        print(f"Entries Set to L13 (Exclusion/Blank/Invalid): {unclassified}")
        print(f"Classification Rate (Excl. L13): {classification_rate:.2f}%")
        print(f" Ground Ops L39: {new_l39_count}")
        print("============================================================")

        # --- CORRECTED FILENAME FORMAT ---
```

```
        output_filename = f'5levels_aviation_{timestamp}.xlsx'
        output_path = os.path.join(OUTPUT_DIR, output_filename)

        print(f" Saving classified data to: {output_path}")
        df.to_excel(output_path, index=False)
        print(" Script finished successfully! Check your output directory for the Excel file.")

    except Exception as e:
        print(f" CRITICAL ERROR during execution: {e}")


if __name__ == '__main__':
    main()
```

A.1.6. Whitelist

airline_brand	matched_company	company_name
abx air	abx air, inc.	ABX Air, Inc.
alaska airlines	alaska airlines	Alaska Airlines
allegiant air	allegiant air	Allegiant Air
american airlines	american airlines	American Airlines
american eagle	american eagle	American Eagle
atlas air	atlas air	Atlas Air
cape air	cape air	Cape Air
delta air lines	delta air lines, inc	Delta Air Lines, Inc
delta air lines	delta air lines	Delta Air Lines
envoy air	envoy air	Envoy Air
expressjet	expressjet airlines	Expressjet Airlines
fedex express	fedex express	FedEx Express
frontier airlines	frontier airlines	Frontier Airlines
gojet airlines	gojet airlines	GoJet Airlines
hawaiian airlines	hawaiian airlines	Hawaiian Airlines
horizon air	horizon air	Horizon Air
jetblue	jetblue airways	JetBlue Airways
jetblue	jetblue airways corporation	JetBlue Airways Corporation
jsx	jsx	JSX
netjets	netjets	NetJets
piedmont airlines	piedmont airlines	Piedmont Airlines
psa airlines	psa airlines	PSA Airlines
psa airlines	psa airlines, inc.	PSA Airlines, Inc.
ravn alaska	ravn alaska	Ravn Alaska
skywest	skywest	Skywest
skywest airlines	skywest airlines	SkyWest Airlines
southwest airlines	southwest airlines/customer service	Southwest Airlines/customer Service
southwest airlines	southwest airlines	Southwest Airlines
spirit airlines	spirit airlines	Spirit Airlines
sun country airlines	sun country airlines	Sun Country Airlines

united airlines	united airlines	United Airlines
united express	commutair dba united express	CommutAir dba United Express
Republic Airways	republic	republic airways
Northern Pacific Airways	northern pacific	northern pacific airways
Ups Airlines	ups	ups airlines
Commutair	commutair	commutair
Air Wisconsin	wisconsin	air wisconsin
Southern Airways Express	southern express	southern airways express
Usa Jet Airlines	usa jet	usa jet airlines
Breeze Airways	breeze	breeze airways
Northern Air Cargo	northern cargo	northern air cargo
Key Lime Air	key lime	key lime air
Global Crossing Airlines	global crossing	global crossing airlines
World Atlantic Airlines	world atlantic	world atlantic airlines
Amerijet International	amerijet international	amerijet international
Eastern Airlines	eastern	eastern airlines
Jetstream Aviation	jetstream aviation	jetstream aviation
Empire Airlines	empire	empire airlines
Everts Air	everts	everts air
Mesa Airlines	mesa	mesa airlines
Ameriflight	ameriflight	ameriflight
Iaero Airways	iaero	iaero airways
Air Transport International	transport international	air transport international
Kalitta Air	kalitta	kalitta air
Aloha Air Cargo	aloha cargo	aloha air cargo
National Airlines	national	national airlines
Western Global Airlines	western global	western global airlines
Cargojet Airways	cargojet	cargojet airways
Avelo Airlines	avelo	avelo airlines
Omni Air International	omni international	omni air international

A.1.7. Blacklist

360 Aviation
ABL Aviation
ALTO Aviation
AerCap
Aero Design Labs
Aero Dynamix
Aero Star Aviation
AeroParts Now
Aerospace Technologies Group
Aery Aviation
Airways Aviation
Albinati Aeronautics
Alerion Aviation
Alliance Aviation Services
Altitude Aerospace
Anderson Aeromotive
Apollo MedFlight
Archer Aviation
Av8 Group
AvAir
Avia Solutions
Avion Graphics
Avionica
Avioserv
Avmats
BAS Part Sales
Barnes Aerospace
BizJet International
C&L Aviation Group
Constant Aviation
Contour Aviation
Cutter Aviation
Duncan Aviation
East West Aeronautical
ExecuJet MRO Services
FLYHT
Falcon Aviation Services
Field Aerospace
Flight Research
Flying Colours Corp
GA Telesis
GlobalParts.aero
Gulfstream Aerospace
Hawker Pacific
Heritage Aviation
IAero Thrust
Jet Aviation
Jet Midwest
Jett Pro
King Aerospace
Liberty Jet
MRO Holdings
Mid Continent Instruments and Avionics
Mingo Aerospace
NORDAM
Pentastar Aviation
Pratt & Whitney
Precision Aviation Group
RBR Aviation
STS Aviation Group
StandardAero
Textron Aviation
Thrust Tech Accessories
West Star Aviation
airtran airways
braniff international airways
continental airlines
eastern air lines
flybe
midway airlines
monarch airlines
pan am
pan american world airways
trans world airlines
twa
us airways
wow air

A.2. Automotive Industry

A.2.1. Preprocessing

```
import pandas as pd  # Imports pandas for data manipulation and analysis
import os  # Imports os to interact with the computer's file system (folders/files)
from datetime import datetime  # Imports datetime to create timestamps for filenames
import re  # Imports regular expressions for advanced text cleaning

# --- CONFIGURATION FOR STEP 1 ---
# Defines the specific folder where raw, modified LinkedIn CSVs are stored
raw_csv_input_folder = '/Volumes/CO_MSc/06 Modified CSVs Automotive'
# Defines the target folder where the cleaned Excel files will be saved
preprocessed_output_folder = '/Volumes/CO_MSc/1preprocessed_autoindu'

# Checks if the output folder exists; if not, it creates it automatically
os.makedirs(preprocessed_output_folder, exist_ok=True)

# A list of the 18 specific columns you want to extract from the raw data
columns_to_keep_step1 = [
    "id", "current_company", "education", "experience", "position",
    "experience_company", "experience_industry", "experience_title", "experience_duration",
    "crunchbase_company", "num_employees", "type", "crunchbase_industries",
    "contacts", "current_employees", "num_contacts", "legal_name",
    "industry_dummy"
]

# --- FUNCTION FOR STRING CLEANING ---
def clean_string_for_excel(x):
    """
    Removes non-printable control characters from strings for Excel compatibility.
    """
    if isinstance(x, str):  # Only runs if the data point is a piece of text (string)
        # Uses regex to strip characters like null, bell, or escape that break Excel
        return re.sub(r"[\x00-\x1F\x7F-\x9F]", "", x)
    return x  # Returns the data as-is if it is not a string (e.g., numbers, empty cells)

# --- MAIN EXECUTION FOR STEP 1 ---
print(f"Starting Step 1: Preprocessing files from: {raw_csv_input_folder}")

# Scans the input folder and makes a list of files starting with 'chunk_' and ending with
'_modified.csv'
file_list = [f for f in os.listdir(raw_csv_input_folder) if f.endswith('_modified.csv') and
f.startswith('chunk_')]

if not file_list:  # If the code finds 0 matching files, it stops here
    print(f"No '_modified.csv' files found in {raw_csv_input_folder}. Please check the path
and file names. Exiting.")
else:
    print(f"Found {len(file_list)} files to preprocess.")
```

```
    # Loops through every file found in the 'file_list'
    for i, filename in enumerate(file_list):
        print(f"\n--- Processing file {i+1}/{len(file_list)} for Step 1: {filename} ---")

        # Combines the folder path and filename to get the full location of the CSV
        current_csv_input_path = os.path.join(raw_csv_input_folder, filename)

        # Creates a unique timestamp (Year.Month.Day.Hour.Minute) for the new filename
        timestamp = datetime.now().strftime("%Y.%m.%d.%H.%M")
        # Removes the '.csv' extension from the original filename
        base_filename_without_ext = os.path.splitext(filename)[0]
        # Combines prefix, name, and timestamp into a new .xlsx filename
        preprocessed_excel_filename =
f"1pre_{base_filename_without_ext}_{timestamp}.xlsx"
        # Defines the full save path for the final Excel file
        preprocessed_excel_output_path = os.path.join(preprocessed_output_folder,
preprocessed_excel_filename)

        try:
            print(f" Loading CSV: {filename}")
            # Reads the CSV file, but only imports the 18 columns defined earlier
            df_raw = pd.read_csv(current_csv_input_path, usecols=columns_to_keep_step1,
low_memory=False)
            print(f"Loaded {len(df_raw)} rows.")

            print(" Cleaning strings for Excel compatibility…")
            # Applies the cleaning function to every cell in columns that contain text
            df_cleaned = df_raw.apply(lambda col: col.map(clean_string_for_excel) if col.dtype
== "object" else col)

            print(f" Saving preprocessed data to: {preprocessed_excel_output_path}")
            # Exports the cleaned data to an Excel file using the openpyxl engine
            df_cleaned.to_excel(preprocessed_excel_output_path, index=False,
engine="openpyxl")
            print(" Preprocessing complete for this file.")

        except pd.errors.EmptyDataError:  # Handles files that have headers but no data
            print(f" WARNING: File {filename} is empty or contains no data. Skipping this file.")
        except FileNotFoundError:  # Handles cases where a file disappears or the path is broken
            print(f" ERROR: Input file {filename} not found. Please check the path. Skipping this
file.")
        except Exception as e:  # Catches any other unexpected errors so the whole loop doesn't
crash
            print(f" ERROR: An unexpected error occurred while processing {filename}: {e}.
Skipping this file.")

print("\n--- Step 1: All files preprocessed. ---")
print(f"Preprocessed files saved to: {preprocessed_output_folder}")
```

A.2.2. Reshaping

```
import pandas as pd
import os
from datetime import datetime
import re
import json
from dateutil.relativedelta import relativedelta # Make sure to install: pip install python-dateutil

# --- CONFIGURATION FOR STEP 2 ---
# Here the paths for the input and output folders are defined
preprocessed_input_folder = '/Volumes/CO_MSc/1preprocessed_autoindu'
reshaped_output_folder = '/Volumes/CO_MSc/2reshaped_autoindu'

# Here the system creates the output folder if it does not already exist
os.makedirs(reshaped_output_folder, exist_ok=True)

# Here a dictionary is created to translate month abbreviations into numbers
month_map = {
    "Jan": 1, "Feb": 2, "Mar": 3, "Apr": 4,
    "May": 5, "Jun": 6, "Jul": 7, "Aug": 8,
    "Sep": 9, "Oct": 10, "Nov": 11, "Dec": 12
}

# --- FUNCTIONS FOR RESHAPING LOGIC ---
def parse_date(date_str):
    """
    Parses a date string into a datetime object. Handles 'Present' and various 'Mon.YYYY' formats.
    """
    # Here the function checks if the input is actually text
    if not isinstance(date_str, str):
        return None
    # Here 'Present' is converted to the current date and time
    if date_str.strip().lower() == "present":
        return datetime.today()

    # Here regex is used to find patterns like 'Jan 2020' or 'Jan. 2020'
    match = re.match(r"(\w{3})\.? (\d{4})", date_str.strip())
    if match:
        month_str, year = match.groups()
        month = month_map.get(month_str)
        if month:
            # Here a date object is created for the first day of that month
            return datetime(int(year), int(month), 1)

    # Here the code tries to parse the string if it only contains a 4-digit year
    if re.match(r"^\d{4}$", date_str.strip()):
        try:
            return datetime(int(date_str.strip()), 1, 1)
        except ValueError:
            pass
```

```
    return None


def months_between(d1, d2):
    """
    Calculates the number of months between two datetime objects.
    """
    # Here the code ensures both dates exist before calculating
    if not d1 or not d2:
        return 0
    # Here the code swaps the dates if the start date is after the end date
    if d1 > d2:
        d1, d2 = d2, d1

    # Here the difference is calculated in months, adding a month if there are more than 15
extra days
    delta = relativedelta(d2, d1)
    return delta.years * 12 + delta.months + (delta.days > 15)


def parse_experience_entry(entry_str, row_id=None):
    """
    Parses a JSON string from the 'experience' column, handles various malformations,
    and flattens nested 'positions' lists into a list of standardized dictionaries.
    """
    # Here the function validates that the entry is a non-empty string
    if not isinstance(entry_str, str):
        return []

    if not entry_str.strip() or entry_str.strip() in ['[]', '{}', 'None']:
        return []

    processed_str = entry_str
    try:
        # Here common characters like backslashes and newlines are escaped for JSON
compatibility
        processed_str = processed_str.replace("\\", "\\\\")
        processed_str = processed_str.replace("\n", "\\n").replace("\r", "\\r").replace("\t", "\\t")
        processed_str = processed_str.replace("""", "")

        # Here regex adds double quotes to any keys that are missing them
        processed_str = re.sub(r'([{,]\s*)([a-zA-Z_]\w*)\s*:', r'\1"\2":', processed_str)

        # Here missing commas between JSON objects are fixed
        processed_str = re.sub(r'}\s*{', '},{', processed_str)

        # Here the code ensures the string is enclosed in square brackets as a list
        if not processed_str.strip().startswith('['):
            processed_str = '[' + processed_str + ']'

        # Here the cleaned string is converted into a Python list
```

```
parsed_entries = json.loads(processed_str)

if not isinstance(parsed_entries, list):
    if isinstance(parsed_entries, dict):
        parsed_entries = [parsed_entries]
    else:
        return []

parsed_list_of_positions = []
# Here the code loops through the parsed companies and positions
for entry in parsed_entries:
    if isinstance(entry, dict):
        company = entry.get("company_name") or entry.get("company")
        industry = entry.get("industry")

        positions_list = entry.get("positions")
        # Here the code handles entries that have a nested list of positions
        if isinstance(positions_list, list) and positions_list:
            for p in positions_list:
                if isinstance(p, dict):
                    title = p.get("title")
                    start_date_str = p.get("start_date")
                    end_date_str = p.get("end_date")

                    start = parse_date(start_date_str)
                    end = parse_date(end_date_str)

                    # Here invalid date sequences are skipped
                    if start and end and start > end:
                        continue
                    elif not start and end:
                        continue

                    parsed_list_of_positions.append({
                        "company": company,
                        "title": title,
                        "start": start,
                        "end": end,
                        "industry": industry
                    })
        # Here the code handles simple entries where position info is at the top level
        else:
            title = entry.get("title")
            start_date_str = entry.get("start_date")
            end_date_str = entry.get("end_date")

            start = parse_date(start_date_str)
            end = parse_date(end_date_str)

            if start and end and start > end:
                continue
```

```
                elif not start and end:
                    continue

                parsed_list_of_positions.append({
                    "company": company,
                    "title": title,
                    "start": start,
                    "end": end,
                    "industry": industry
                })
        return parsed_list_of_positions
    except json.JSONDecodeError as e:
        # print(f"ERROR (ID: {row_id}): JSON Decode Error for original entry:
'{entry_str[:200]}...' Error: {e}")

        return []

    except Exception as e:

        # print(f"ERROR (ID: {row_id}): General Error parsing experience entry:
'{entry_str[:200]}...' Error: {e}")

        return []


def reshape_dataframe(df_input, max_positions=30):
    """
    Reshapes the input DataFrame by flattening 'experience' JSON data into new columns,
    calculating total experience, and handling date overlaps.
    """
    all_records = []

    # Here the code begins iterating through every row in the spreadsheet
    for index, row in df_input.iterrows():
        row_id = row.get("id", f"Row_{index}")

        # Here the JSON experience data is converted into a usable list of jobs
        flat_positions = parse_experience_entry(row.get("experience"), row_id)

        # Here jobs are sorted by date so the most recent ones appear first
        flat_positions = sorted([p for p in flat_positions if p["start"]], key=lambda x: x["start"],
reverse=True)

        # Here the code adjusts job dates so they do not overlap chronologically
        for i in range(1, len(flat_positions)):
            if flat_positions[i - 1]["end"] and flat_positions[i]["start"] and flat_positions[i -
1]["end"] > \
                    flat_positions[i]["start"]:
                flat_positions[i - 1]["end"] = flat_positions[i]["start"]
            if flat_positions[i]["end"] and flat_positions[i - 1]["start"] and flat_positions[i]["end"]
> \
```

```
                flat_positions[i - 1]["start"]:
            flat_positions[i]["end"] = flat_positions[i - 1]["start"]

    # Here the code prepares a list of job periods to calculate total work time
    total_months = 0
    periods = []
    for p in flat_positions:
        if p["start"] and p["end"]:
            periods.append((p["start"], p["end"]))
        elif p["start"] and not p["end"]:
            periods.append((p["start"], datetime.today()))

    periods.sort()

    # Here overlapping time periods are merged together to avoid double-counting
experience
    merged_periods = []
    if periods:
        current_start, current_end = periods[0]
        for i in range(1, len(periods)):
            next_start, next_end = periods[i]
            if next_start <= current_end:
                current_end = max(current_end, next_end)
            else:
                merged_periods.append((current_start, current_end))
                current_start, current_end = next_start, next_end
        merged_periods.append((current_start, current_end))

    # Here the final count of months is converted into a total years value
    for start, end in merged_periods:
        total_months += months_between(start, end)

    total_years = round(total_months / 12, 2)

    # Here the original row data is preserved while adding the new experience calculation
    base = row.to_dict()
    base["total_experience_years"] = total_years

    # Here the list of jobs is broken out into individual columns (up to 30)
    for i, p in enumerate(flat_positions[:max_positions]):
        idx = i + 1
        base[f"exp_{idx}_company"] = p["company"]
        base[f"exp_{idx}_title"] = p["title"]
        base[f"exp_{idx}_start"] = p["start"]
        base[f"exp_{idx}_end"] = p["end"]
        base[f"exp_{idx}_industry"] = p["industry"]
        base[f"exp_{idx}_duration_years"] = round(months_between(p["start"], p["end"]) /
12, 2)

    all_records.append(base)
```

```
    # Here the list of processed records is converted back into a DataFrame
    return pd.DataFrame(all_records)


# --- MAIN EXECUTION FOR STEP 2 ---
print(f"Starting Step 2: Reshaping files from: {preprocessed_input_folder}")

# Here the code searches the folder for all Excel files starting with '1pre_'
file_list = [f for f in os.listdir(preprocessed_input_folder) if f.endswith('.xlsx') and
f.startswith('1pre_')]

if not file_list:
    print(f"No '1pre_*.xlsx' files found in {preprocessed_input_folder}. Please ensure Step 1
was run. Exiting.")
else:
    print(f"Found {len(file_list)} files to reshape.")

    # Here the code begins the loop to process each file individually
    for i, filename in enumerate(file_list):
        print(f"\n--- Processing file {i+1}/{len(file_list)} for Step 2: {filename} ---")

        current_excel_input_path = os.path.join(preprocessed_input_folder, filename)

        # Here the timestamp and new filename are generated for the output
        timestamp = datetime.now().strftime("%Y.%m.%d.%H.%M")
        base_filename_part = '_'.join(filename.split('_')[1:-1])
        reshaped_output_filename = f"2reshaped_{base_filename_part}_{timestamp}.xlsx"
        final_reshaped_output_path = os.path.join(reshaped_output_folder,
reshaped_output_filename)

        try:
            print(f" Loading Excel for reshaping: {filename}")
            # Here the Excel file is loaded into memory
            df_step2_input = pd.read_excel(current_excel_input_path)
            print(f"Loaded {len(df_step2_input)} rows.")

            # Here duplicate person IDs are removed to ensure data quality
            df_step2_input = df_step2_input.drop_duplicates(subset="id", keep="first")
            print(f"Removed duplicates. {len(df_step2_input)} unique rows remaining.")

            print(" Reshaping dataframe...")
            # Here the main reshaping logic is triggered
            df_reshaped = reshape_dataframe(df_step2_input)
            print(f"Reshaped dataframe to {len(df_reshaped.columns)} columns and
{len(df_reshaped)} rows.")

            print(f" Saving reshaped data to: {final_reshaped_output_path}")
            # Here the final reshaped data is saved to a new Excel file
            df_reshaped.to_excel(final_reshaped_output_path, index=False, engine='xlsxwriter')
            print(" Reshaping complete for this file.")
```

```
        except pd.errors.EmptyDataError:
            print(f" WARNING: File {filename} is empty or contains no data. Skipping this file.")
        except FileNotFoundError:
            print(f" ERROR: Input file {filename} not found. Please check the path. Skipping this
file.")
        except Exception as e:
            print(f" ERROR: An unexpected error occurred while processing {filename}: {e}.
Skipping this file.")

print("\n--- Step 2: All files reshaped. ---")
print(f"Reshaped files saved to: {reshaped_output_folder}")
```

A.2.3. Filter

```
import pandas as pd
import os
from tqdm import tqdm
from datetime import datetime

# === KONFIGURATION ===
# Source directory containing the reshaped input files
INPUT_DIR = "/Users/Christoph/Desktop/MasterarbeitUniWien/2reshaped_autoindu"

# Target directory for the filtered and bundled output files
OUTPUT_DIR = "/Users/Christoph/Desktop/MasterarbeitUniWien/3filtered_autoindu"
os.makedirs(OUTPUT_DIR, exist_ok=True)  # Ensure the output directory exists

# Number of individual files to merge into a single batch output
AGGREGATION_CHUNK_SIZE = 50

# === WHITELIST AND BLACKLIST ===
# Industries explicitly included in the auto-industry scope
WHITELIST = {
    "Appliances, Electrical, and Electronics Manufacturing",
    "Automation Machinery Manufacturing",
    "Automotive",
    "Electrical/Electronic Manufacturing",
    "Industrial Machinery Manufacturing",
    "Logistics and Supply Chain",
    "Machinery Manufacturing",
    "Manufacturing",
    "Mechanical or Industrial Engineering",
    "Motor Vehicle Manufacturing",
    "Oil & Energy",
    "Oil and Gas",
    "Semiconductors",
    "Transportation, Logistics and Storage",
    "Truck Transportation"
}

# Industries explicitly excluded from the scope to reduce noise
```

```
BLACKLIST = {
    "Biotechnology Research",
    "Building Materials",
    "Computer Software",
    "Consumer Goods",
    "Financial Services",
    "Food and Beverage Services",
    "Higher Education",
    "Hospitality",
    "Hospitals and Health Care",
    "Insurance",
    "Internet Publishing",
    "Law Practice",
    "Leasing Real Estate",
    "Medical Equipment Manufacturing",
    "Pharmaceutical Manufacturing",
    "Real Estate",
    "Restaurants",
    "Retail",
    "Software Development",
    "Staffing and Recruiting",
    "Telecommunications",
    "Wellness and Fitness Services"
}

# === FILTER FUNCTION ===
def is_relevant(row):
    """
    Evaluates row relevance based on industry categories.
    Validates entries against the whitelist while ensuring they aren't blacklisted.
    """
    industry = str(row.get("exp_1_industry", "")).strip()
    normalized_industry = industry.lower()

    # Perform case-insensitive membership check for both lists
    return (normalized_industry in {i.lower() for i in WHITELIST} and
            normalized_industry not in {i.lower() for i in BLACKLIST})

#                         MAIN BATCH EXECUTION

if __name__ == "__main__":
    print(f"Starting batch filtering and aggregation.")
    print(f"Reading input files from: {INPUT_DIR}")

    # Get all .xlsx files in the input directory
    all_files = [os.path.join(INPUT_DIR, f) for f in os.listdir(INPUT_DIR) if
f.endswith('.xlsx')]
    all_files.sort()  # Sort to ensure consistent processing order

    if not all_files:
        print("No Excel files found in the input directory. Exiting.")
```

```
    else:
        batch_dataframes = []
        batch_count = 0
        file_counter = 0

        # Iterate through files with a visual progress bar
        for full_input_path in tqdm(all_files, desc="Processing files"):
            try:
                # Load file and identify rows matching industry criteria
                df = pd.read_excel(full_input_path)
                relevance_mask = df.apply(is_relevant, axis=1)
                filtered_df = df[relevance_mask].copy()

                # Collect non-empty filtered data for aggregation
                if not filtered_df.empty:
                    batch_dataframes.append(filtered_df)

                file_counter += 1

                # Trigger batch save when the chunk size limit is reached
                if file_counter % AGGREGATION_CHUNK_SIZE == 0:
                    if batch_dataframes:
                        combined_df = pd.concat(batch_dataframes, ignore_index=True)
                        timestamp = datetime.now().strftime('%Y.%m.%d.%H.%M.%S')
                        output_file_name = f"3filtered_autoindustry_batch_{batch_count +
1}_{timestamp}.xlsx"
                        output_path = os.path.join(OUTPUT_DIR, output_file_name)

                        combined_df.to_excel(output_path, index=False, engine='xlsxwriter')
                        print("Batch saved successfully.")

                        # Reset for the next batch
                        batch_dataframes = []
                        batch_count += 1

            except Exception as e:
                print(f"An error occurred while processing {os.path.basename(full_input_path)}:
{e}. Skipping.")

        # Finalize the remaining data that didn't fill a complete chunk
        if batch_dataframes:
            combined_df = pd.concat(batch_dataframes, ignore_index=True)
            timestamp = datetime.now().strftime('%Y.%m.%d.%H.%M.%S')
            output_file_name = f"3filtered_autoindustry_batch_{batch_count +
1}_final_{timestamp}.xlsx"
            output_path = os.path.join(OUTPUT_DIR, output_file_name)
            combined_df.to_excel(output_path, index=False, engine='xlsxwriter')
```

A.2.3.1. Industry Filter & Batching

```
import pandas as pd
```

```
import os
from tqdm import tqdm
from thefuzz import fuzz
from datetime import datetime
import re
from typing import Set

# === CONFIGURATION ===
# Input files are the output from the previous industry filter
INPUT_DIR = "/Users/Christoph/Desktop/MasterarbeitUniWien/3filtered_autoindu"

# Define the CORRECTED output directory for the new, filtered files
OUTPUT_DIR = "/Users/Christoph/Desktop/MasterarbeitUniWien/3filter_autoindu_oem"
os.makedirs(OUTPUT_DIR, exist_ok=True)  # Ensure the output directory exists

# The column containing the company name to be filtered
COMPANY_COLUMN = 'exp_1_company'

# The fuzzy match score threshold (0-100).
FUZZY_THRESHOLD = 90

# === OEM BRAND LIST ===
# Your list of 93 Car Brands/OEMs from whitelist_oem_top_brands_2025.10.07.17.54.txt
OEM_BRANDS_RAW: Set[str] = {
    "Acura", "Aiways", "Alfa Romeo", "Alpine", "Aptera", "Arrival", "Audi", "BAIC",
"Bentley", "BMW", "BMW i",
    "Bugatti", "Buick", "BYD", "Cadillac", "Canoo", "Changan", "Chery", "Chevrolet",
"Chrysler", "Citroën",
    "Cupra", "Dacia", "Daihatsu", "Datsun", "Dodge", "Dongfeng", "Faraday Future", "FAW",
"Fiat", "Fisker",
    "Ford", "GAC", "Geely", "General Motors", "Genesis", "GMC", "Great Wall", "Haval",
"Honda", "Hyundai",
    "Infiniti", "Iran Khodro", "Isuzu", "JAC", "Jaguar", "Jeep", "Kia", "Lamborghini", "Land
Rover", "Lexus",
    "Lincoln", "Lotus", "Lucid", "Lynk & Co", "Mahindra", "Maserati", "Mazda", "Mercedes-
Benz", "MG", "Mini",
    "Mitsubishi", "NIO", "Nissan", "Opel", "Peugeot", "Polestar", "Porsche", "Proton", "Ram",
"Renault",
    "Rimac", "Rivian", "Roewe", "Rolls-Royce", "SAIC", "Saipa", "Seat", "Smart",
"Stellantis", "Subaru",
    "Suzuki", "Tata", "Tesla", "Toyota", "Vauxhall", "Volkswagen", "Volkswagen Commercial
Vehicles", "Volvo",
    "WEY", "Xpeng", "Zeekr", "Škoda"
}


# Pre-clean and normalize the OEM list for faster matching
def normalize_text(text: str) -> str:
    """Removes non-alphanumeric/whitespace chars and converts to lowercase."""
    if not isinstance(text, str):
        return ""
```

```
    # Remove special characters (like periods and commas), keeping letters and numbers
    text = re.sub(r'[^a-zA-Z0-9\s]', '', text)
    return text.lower().strip()


# Create a set of normalized OEM brands
NORMALIZED_OEM_BRANDS = {normalize_text(brand) for brand in
OEM_BRANDS_RAW}


# === FILTER FUNCTION: FUZZY MATCHING ===
def is_oem_relevant(company_name: str) -> bool:
    """
    Checks if a company name is a fuzzy match for any OEM brand.
    Uses fuzz.partial_ratio to handle subsets (e.g., 'Seat' in 'Seat GmbH').
    """
    normalized_company = normalize_text(company_name)
    if not normalized_company:
        return False

    for oem in NORMALIZED_OEM_BRANDS:
        # Use partial_ratio to check if the OEM brand is a strong subset match
        score = fuzz.partial_ratio(normalized_company, oem)
        if score >= FUZZY_THRESHOLD:
            return True  # Match found above threshold

    return False


def filter_and_save_single_file(input_filepath, output_dir):
    """
    Filters a single input Excel file and saves the result to a new folder.
    """
    filename = os.path.basename(input_filepath)
    print(f"\n--- Filtering {filename} ---")

    try:
        df = pd.read_excel(input_filepath)
        # Ensure column names are lowercase for consistent access
        df = df.rename(columns={c: c.lower() for c in df.columns})

        if COMPANY_COLUMN not in df.columns:
            print(f"  ⚠ Skipping {filename}: Company column '{COMPANY_COLUMN}' not
found. Check your file structure.")
            return None

        print(f"  Loaded {len(df)} rows. Applying OEM filter...")

    except Exception as e:
        print(f"  An error occurred while loading {input_filepath}: {e}. Skipping.")
        return None
```

```
    # === APPLY FILTER WITH PROGRESS BAR ===
    tqdm.pandas(desc="  Fuzzy filtering rows")

    # Apply the fuzzy matching logic to the company column
    relevance_mask = df[COMPANY_COLUMN].progress_apply(is_oem_relevant)

    filtered_df = df[relevance_mask].copy()

    print(f"Filtering complete. {len(filtered_df)} relevant OEM rows found.")

    # === EXPORT ===
    timestamp = datetime.now().strftime('%Y.%m.%d.%H.%M.%S')
    # Use a simpler naming convention based on the original filename
    output_file_name = f"3.5_oem_filtered_{filename.replace('.xlsx', '')}_{timestamp}.xlsx"
    output_path = os.path.join(output_dir, output_file_name)

    print(f"Saving filtered data to: {output_path}")
    try:
        filtered_df.to_excel(output_path, index=False, engine='xlsxwriter')
        print("Filtered data saved successfully.")
    except Exception as e:
        print(f"An **ERROR** occurred while saving the filtered data: {e}. Check
permissions/file lock.")
    return filtered_df

#                         MAIN BATCH EXECUTION

if __name__ == "__main__":
    print(f"Starting batch filtering on OEM brands.")
    print(f"Reading input files from: {INPUT_DIR}")

    # Get all .xlsx files in the input directory
    all_files = [os.path.join(INPUT_DIR, f) for f in os.listdir(INPUT_DIR) if
f.endswith('.xlsx')]
    all_files.sort()  # Sort to ensure consistent processing order

    if not all_files:
        print("No Excel files found in the input directory. Exiting.")
    else:
        print(f"Found {len(all_files)} total files to process.")

        for full_input_path in all_files:
            filter_and_save_single_file(full_input_path, OUTPUT_DIR)

    print("\nBatch filtering execution finished for all files.")
```

A.2.4. Horizontal Categorization & Data Reduction

```
import pandas as pd
import os
```

```
import re
from datetime import datetime
from typing import Set, List, Union

# === CONFIGURATION ===
# These variables tell the script where to find your data and where to put the results.
# The folder path where your existing Excel files are stored.
INPUT_DIR = '/Users/Christoph/Desktop/MasterarbeitUniWien/3filter_autoindu_oem'
# The folder path where the script will save the newly cleaned and categorized files.
OUTPUT_DIR = '/Users/Christoph/Desktop/MasterarbeitUniWien/4analysis_autoindu_flat'
# A list of possible column names for job titles; the script checks these in order of priority.
TITLE_COLUMNS = ['exp_1_title', 'job_title', 'position', 'title', 'titel']
# The name of the new column that will be created to store the final department/category.
FLATTENED_COLUMN = 'category_flat_final'

# Automatically creates the output folder on your computer if it doesn't exist yet.
os.makedirs(OUTPUT_DIR, exist_ok=True)

# --- MISSPELING CORRECTION MAP ---
# This is a dictionary used to standardize text. It finds common typos or variations (like
'enginer')
# and replaces them with a clean, standard version ('engineer') to ensure accuracy during
sorting.
MISSPELING_MAP: dict[str, str] = {
    "enginer": "engineer", "engneer": "engineer", "enginner": "engineer", "enginerr":
"engineer",
    "engneering": "engineer", "enginering": "engineer", "engeneer": "engineer", "engenier":
"engineer",
    "engieer": "engineer", "engieering": "engineer", "engenerring": "engineer", "ingénieur":
"engineer",
    "asambly": "assembly", "asembley": "assembly", "assemly": "assembly", "assambly":
"assembly",
    "assambli": "assembly", "asemblaje": "assembly", "asemblar": "assembly", "maneger":
"manager",
    "managar": "manager", "managor": "manager", "mngr": "manager", "manger": "manager",
    "tecnician": "technician", "techician": "technician", "tehcnician": "technician",
    "tekniker": "technician", "technik": "technician", "consulant": "consultant",
    "consultent": "consultant", "specialest": "specialist", "specialst": "specialist",
    "spezialist": "specialist", "proyect": "project",
    "projekct": "project", "projeto": "project", "operater": "operator",
    "oprator": "operator", "direktor": "director", "directer": "director", "dirctor": "director",
    "dirécteur": "director", "analist": "analyst",
    "anallist": "analyst", "analyste": "analyst", "interne": "intern", "interno": "intern",
    "stagiaire": "intern", "eng": "engineer_abbr", "engr": "engineer_abbr", "engen":
"engineer_abbr",
    "engnr": "engineer_abbr", "ing": "engineer_abbr"
}

# --- DELETION LISTS ---
# This is a list of "exclusion" words. If a job title contains any of these, the script assumes
# the person is a student, retired, or in an irrelevant role, and deletes that row entirely.
```

```
DELETION_KEYWORDS: Set[str] = {
    # Non-Active/Pre-Employment Status: Removes people who aren't currently working in the role.
    "retired", "former", "ex-", "past", "alumnus", "alumna", "previously",
    "volunteer", "student", "candidate", "seeking", "aspiring", "unemployed",
    "looking for", "phd", "intern", "trainee", "apprenticeship", "studying",
    "seeking", "looking", "unemployed", "none", "future", "trust me", "always working",
    "together we can change the world", "customer service oriented", "automotive professional",
    "high school diploma", "attended", "enthusiast", "come what may be", "outdoors, fitness & business enthusiast",
    "willing to learn new things", "networking", "music producer", "multimedia artist",
    "graduate university of phoenix", "ngk ceramics lab assistant", "head basketball coach",
    "artist", "fourth year student", "empty", "at", "advisor",

    # Spanish titles/words: Removes non-English entries to keep the dataset consistent.
    "ejecutivo", "ejecutiva", "gerente", "responsable", "técnico", "consultor",
    "consultante", "especialista", "analista", "becario", "ventas", "proyecto",
    "proyeto", "operador", "diretora", "director técnico",

    # Irrelevant high-frequency titles: Removes general or unrelated jobs identified in previous runs.
    "valet", "millwright", "employee", "contract specialist", "solar installer",
    "geophysicist", "pipefitter", "service", "global supply management", "technical writer",
    "delivery specialist", "owner", "technical support specialist", "senior technical writer",
    "gm", "superintendent", "safety specialist", "laborer", "dock worker", "general laborer",
    "measurement specialist", "it", "ai/ml scientist", "sdet", "helicopter pilot",
    "digital scholar", "self employed", "dre", "technical fellow", "partner", "petrophysicist",
    "technologist", "quality control specialist", "union rep", "used car mgr", "air specialist",
    "roadside agent", "used vehicle quality specialist",
    "auto tech", "company owner", "autoworker", "bdc", "sme", "title specialist",
    "site surveyor", "timekeeper", "service desk specialist ii", "solar roofer",
    "journeyman pipefitter", "trader", "service support specialist", "co op", "sr. sdet",
    "senior ehs specialist", "team specialist", "lube tech", "supplier technical assistance",
    "delivery experience specialist", "diemaker", "wsm", "master scheduler",
    "field technical support specialist ii", "interconnection specialist", "franchise owner",
    "pm", "industrial hygienist", "senior specialist", "quality", "gsm",
}

# A list of car brands. If a job title is ONLY the brand name (e.g., just "Audi"), it is deleted.
OEM_BRANDS: Set[str] = {
    "acura", "aiways", "alfa romeo", "alpine", "aptera", "arrival", "audi", "baic", "bentley",
    "bmw", "bmw i", "bugatti", "buick", "byd", "cadillac", "canoo", "changan", "chery",
    "chevrolet", "chrysler", "citroën", "cupra", "dacia", "daihatsu", "datsun", "dodge",
    "dongfeng", "faraday future", "faw", "fiat", "fisker", "ford", "gac", "geely",
    "general motors", "genesis", "gmc", "great wall", "haval", "honda", "hyundai",
    "infiniti", "iran khodro", "isuzu", "jac", "jaguar", "jeep", "kia", "lamborghini",
    "land rover", "lexus", "lincoln", "lotus", "lucid", "lynk & co", "mahindra",
    "maserati", "mazda", "mercedes-benz", "mg", "mini", "mitsubishi", "nio", "nissan",
    "opel", "peugeot", "polestar", "porsche", "proton", "ram", "renault", "rimac",
    "rivian", "roewe", "rolls-royce", "saic", "saipa", "seat", "smart", "stellantis",
```

```
    "subaru", "suzuki", "tata", "tesla", "toyota", "vauxhall", "volkswagen",
    "volkswagen commercial vehicles", "volvo", "wey", "xpeng", "zeekr", "škoda"
}


# --- SHARED HELPERS ---
# Converts any text to lowercase for easier matching.
def _lower(s): return "" if not isinstance(s, str) else s.lower()

# Removes extra spaces from the beginning, end, and middle of a string.
def _ws(s):    return re.sub(r"\s+", " ", s).strip()

# Cleans up punctuation and expands common job abbreviations like 'Sr.' or 'VP'.
def _normalize_title(text: str) -> str:
    """
    Cleans text by converting to lowercase, simplifying common phrases,
    and applying cross-language standardizations.
    """
    t = _lower(text)
    t = re.sub(r"[——\-_/|]+", " ", t) # Replaces dashes and slashes with a space.
    t = re.sub(r"[(){}\[\],;:]", " ", t) # Removes brackets and semicolons.
    t = re.sub(r"\"+", "", t)  # Removes all double quotes.

    # Standardizes common corporate abbreviations into full words.
    t = re.sub(r"\bsr\.?\b", " senior ", t)
    t = re.sub(r"\bvp\b", " vice president ", t)
    t = re.sub(r"\bmd\b", " managing director ", t)
    t = re.sub(r"\bld\b", " lead ", t)

    return _ws(t)

# Splits the title into words and fixes any misspellings found in the MISSPELING_MAP.
def _apply_stemming(text: str) -> str:
    """Replaces single-word misspellings/variants with their correct, normalized
counterparts."""
    if not isinstance(text, str):
        return ""

    t_normalized = _normalize_title(text)
    words = t_normalized.split()

    corrected_words = []
    for word in words:
        # Check if the word is a known typo; if not, keep the original word.
        corrected_words.append(MISSPELING_MAP.get(word, word))

    return " ".join(corrected_words)


# ============================================================
# 1. FLAT CATEGORIES AND KEYWORDS (Functional Priority)
```

```
# ============================================================
# This is the master list for sorting. The script looks for these keywords in order.
FLAT_CATEGORIES: List[tuple[str, List[str]]] = [
    # 1. R&D / ENGINEERING / IT: Highest priority for technical roles.
    ("R&D_Engineering_IT", [
        "r&d", "research", "development", "engineer", "software", "hardware", "validation",
        "simulation", "test engineer", "product engineer", "systems engineer", "powertrain",
        "battery", "embedded", "adas", "autonomous", "infotainment", "controls", "calibration",
        "mechatronics", "hvac engineer", "integration engineer", "devops engineer",
        "vehicle dynamics", "e-drive", "electric drive", "thermal systems", "functional safety",
        "engine calibration", "automated driving", "chassis engineer", "software developer",
        "quality systems specialist", "visualization", "technical artist", "corrosion",
        "technician", "ing", "engr", "designer", "developer", "architect", "systems analyst",
        "scrum master", "technical specialist", "solutions architect", "system designer",
        "planner", "ai/ml scientist", "energy specialist", "petrophysicist", "technologist",
        "scientist", "digital sculptor", "senior digital modeler",
        "sap", "it ", "information technology", "crm", "erp", "cloud", "system admin",
        "network", "security", "cybersecurity", "infrastructure", "data warehouse",
        "digital workplace", "digital transformation", "digitalization", "data engineer",
        "business intelligence", "bi developer", "ui visual designer", "back end developer",
    ]),

    # 2. PRODUCTION & ASSEMBLY: Focuses on the factory floor and quality control.
    ("Production_Assembly", [
        "production", "manufacturing", "assembly", "plant", "line", "press shop", "foundry",
        "operator", "fabrication", "assembler", "line worker", "production associate",
        "paint shop", "body shop", "welding", "machining", "shopfloor", "robot operator",
        "process coach", "machinist", "smt", "assembley", "auto worker", "painter",
        "quality manager", "quality supervisor", "quality assurance", "quality assurance
manager",
        "quality assurance tester", "team member", "production technician", "quality technician",
        "quality lead", "vehicle dimensional quality", "fall hazard control", "body launch
manager",
        "safety supervisor", "field quality", "warranty reduction specialist",
        "ehs specialist", "quality control", "installer", "welder", "quality control specialist",
        "vehicle readiness specialist", "technical sourcer", "hes specialist",
        "toolmaker", "construction superintendent"
    ]),

    # 3. SCM: Roles related to moving parts, logistics, and buying.
    ("SupplyChain_Logistics", [
        "supply chain", "logistics", "warehouse", "material planner", "inventory",
        "shipping", "receiving", "procurement", "buyer", "purchasing", "distribution",
        "import export", "scm", "supply planner", "material handling", "transportation",
        "sourcing", "supplier quality", "packaging engineer", "parts logistics", "parts",
        "parts and accessory consultant", "part specialist",
        "supply manager", "fleet manager", "os&d manager", "launch wpi specialist",
    ]),

    # 4. SALES / CUSTOMER SERVICE: Roles dealing with selling and supporting
customers.
```

```
    ("Sales_CustomerService", [
        "sales", "dealer", "client advisor", "customer", "retail", "service advisor",
        "product advisor", "delivery advisor", "owner advisor", "client experience",
        "brand ambassador", "salesperson", "account executive", "sales consultant",
        "customer experience", "showroom", "fleet sales", "advocate", "driver",
        "service manager", "store manager", "used car manager", "cashier", "service greeter",
        "wholesale market area manager", "wholesale manager", "f&i manger", "guide",
        "automotive biller", "service department", "social media specialist", "warranty specialist"
    ]),

    # 5. MAINTENANCE & REPAIR: Focuses on fixing and maintaining vehicles.
    ("Maintenance_Repair", [
        "mechanic", "maintenance", "repair", "diagnostic", "troubleshooting",
        "auto body", "service technician", "foreman", "tire technician", "shop foreman",
        "service writer", "shop",
        "electrician", "industrial electrician", "field specialist", "automotive painter",
        "detailer", "porter", "service", "technical trainer"
    ]),

    # 6. CORPORATE: Support roles like HR, Finance, Legal, and Marketing.
    ("Corporate", [
        "finance", "accounting", "controller", "hr", "human resources", "recruiter", "payroll",
        "talent", "communication", "pr", "public relations", "legal", "lawyer", "tax",
        "compliance", "internal audit", "admin", "assistant", "reception", "office manager",
        "corporate affairs", "strategy", "sustainability", "environment", "learning", "training",
        "facilities", "real estate", "cleaning", "project manager", "consultant", "marketing",
        "event", "auditor", "public policy", "diversity", "inclusion", "people operations",
        "organizational development", "health and safety", "bookkeeper", "tax preparer", "csr",
        "treasury", "benefits", "staffing", "claims", "vendor management", "risk management",
        "case specialist", "contracts manager", "acquisition specialist", "accounts receivable
specialist",
        "financial analyst", "business analyst", "analyst", "accountant", "accounts payable",
        "counsel", "title clerk", "geologist", "earth scientist", "digital enablement analyst",
        "billing analyst", "brand specialist", "claims adjuster", "contract manager",
        "treasury analyst", "benefits specialist", "staffing specialist", "case specialist",
        "business manager", "account manager", "bmw genius", "regional digital performance
manager",
        "government contracts manager", "remote technical specialist", "bim", "financial
services",
        "brand engagement", "system specialist", "contracts clerk", "business planner",
        "business specialist", "tesla adviser", "desktop support specialist", "economist",
    ]),

    # 7. GENERIC MANAGEMENT: A catch-all for titles that imply leadership but no
specific department.
    ("Management", [
        "ceo", "chief", "cfo", "coo", "cto", "cmo", "executive", "president", "founder",
        "co-founder", "board member", "vp", "vice president", "managing director",
        "general manager", "director general", "leadership", "head of", "global head",
        "leader", "chief of staff", "country manager", "site head", "supervisor",
        "lead", "group lead", "district manager", "area manager", "regional manager",
```

```
        "design manager", "construction manager", "managing partner", "floor manager",
        "zone manager", "manager", "director(?! of )", "management"
    ])
]


# ============================================================
# DELETION LOGIC (The Critical Step)
# ============================================================
# Decides whether to keep a row or delete it based on the keywords and brand names.
def _is_valid_position(title: str) -> bool:
    """Checks if a position should be kept. Returns True if valid (keep), False if invalid
(delete)."""
    t_processed = _apply_stemming(title)
    t_stripped = _ws(t_processed)

    # 1. If any deletion keyword (like 'intern') is in the title, return False (delete).
    if any(keyword in t_processed for keyword in DELETION_KEYWORDS):
        return False

    # 2. If the title is exactly a car company name (like 'Toyota'), return False (delete).
    if t_stripped in OEM_BRANDS:
        return False

    return True


# ============================================================
# INTEGRATOR FUNCTION (Classification)
# ============================================================
# The logic that assigns a single department name to a valid job title.
def classify_flat_role(title: str) -> str:
    """
    Applies the flat functional classification to a single, ALREADY VALIDATED title.
    Order: Function (Priority) -> Management (Fallback) -> Uncategorized.
    """
    if not isinstance(title, str):
        return "Uncategorized"

    # Uses the cleaned and spell-checked title for categorization.
    t = _apply_stemming(title)

    # 1. Check departmental categories first (Priority Check).
    for category_label, keywords in FLAT_CATEGORIES[:-1]:
        if any(keyword in t for keyword in keywords):
            return category_label

    # 2. If no department matched, check if it's a general management role.
    generic_management_keywords = FLAT_CATEGORIES[-1][1]
    if any(keyword in t for keyword in generic_management_keywords):
        return "Management"
```

```
    # 3. If it matches nothing in the list, label it as 'Uncategorized'.
    return "Uncategorized"


# This function applies the deletion and sorting logic to an entire Excel table.
def process_auto_oem_roles_flat(df: pd.DataFrame, title_col: str, out_col: str) ->
pd.DataFrame:
    """
    Applies the deletion logic first, then the classification logic to the DataFrame.
    """
    if title_col not in df.columns:
        raise ValueError(f"Required title column '{title_col}' not found in DataFrame.")

    # --- Step A: Physical Deletion ---
    # Log how many rows we started with.
    print(f"   Pre-Deletion Count: {len(df)} rows")
    # Identify which rows to keep using the gatekeeper function.
    mask_to_keep = df[title_col].apply(_is_valid_position)
    df_cleaned = df[mask_to_keep].copy()

    # Calculate and print how many rows were thrown away.
    deleted_count = len(df) - len(df_cleaned)
    print(f"   Deleted {deleted_count} rows based on exclusion criteria (student, Spanish, and
omissions)")

    # --- Step B: Classification on Remaining Rows ---
    # Prepare the new output column for the results.
    if out_col not in df_cleaned.columns:
        df_cleaned.insert(min(6, len(df_cleaned.columns)), out_col, "")
    df_cleaned[out_col] = df_cleaned[out_col].astype(str).replace('', 'Uncategorized')

    # Assign the final category name to each remaining row.
    df_cleaned.loc[:, out_col] = df_cleaned[title_col].apply(classify_flat_role)
    print(f"   Post-Deletion Count: {len(df_cleaned)} rows classified.")

    return df_cleaned


# The main loop that finds and processes every Excel file in your input directory.
def process_all_files(input_dir, output_dir, title_columns=TITLE_COLUMNS):
    """Iterates through all files in the input directory and processes them."""
    # List all Excel files in the input folder.
    all_files = [os.path.join(input_dir, f) for f in os.listdir(input_dir) if f.endswith('.xlsx')]
    all_files.sort()

    if not all_files:
        print(f"No Excel files found in input directory: {input_dir}. Exiting.")
        return
```

```
    print(f"Starting batch processing ({len(all_files)} files) with **Final Refined
Categorization**.")

    for file_path in all_files:
        filename = os.path.basename(file_path)

        # Looks for the 'batch_#' number in the filename to preserve it.
        match = re.search(r'batch_(\d+)', filename)
        base_name = f'batch_{match.group(1)}' if match else 'combined_data'

        # Gets the current date and time to put in the final filename.
        timestamp = datetime.now().strftime('%Y.%m.%d.%H.%M')

        print(f"\n Processing file: {filename}")
        try:
            # Load the Excel file into memory.
            df = pd.read_excel(file_path)
            # Make column headers lowercase for easier searching.
            df = df.rename(columns={c: c.lower() for c in df.columns})

            # Figure out which column contains the job titles.
            title_col = next((c for c in title_columns if c in df.columns), None)
            if not title_col:
                print(f" Skipping {filename}: No title column found from {title_columns}")
                continue

            # Run the actual cleaning and sorting logic.
            df_final = process_auto_oem_roles_flat(df, title_col=title_col,
out_col=FLATTENED_COLUMN)

            # Create the final name for the output file.
            output_filename = f'4_flat_oem_{base_name}_{timestamp}.xlsx'
            output_path = os.path.join(output_dir, output_filename)

            # Save the results to your computer.
            print(f" Saving final filtered and categorized data to: {output_path}")
            df_final.to_excel(output_path, index=False)

        except Exception as e:
            # If an error happens (e.g., file is open elsewhere), report it and move to the next file.
            print(f" ERROR: Could not process {filename}: {e}. Skipping.")

    print("\n Batch processing finished successfully! The cleaned files are ready for final
analysis.")


# This tells the computer to run the 'process_all_files' function when the script is started.
if __name__ == "__main__":
    process_all_files(INPUT_DIR, OUTPUT_DIR)
```

A.2.5. Vertical Categorization & Merge

```
import pandas as pd
import os
import re
from datetime import datetime
from typing import Tuple, Any

# CONFIGURATION

# 1. Input: The flattened file from the previous step (REQUIRED CHANGE)
# This variable stores the path to your source Excel data
INPUT_FILE_PATH =
r'/Users/Christoph/Desktop/MasterarbeitUniWien/4flatten/flattened_airline_jobs_neo_2025.1
0.23.12.12.xlsx'

# 2. Output Directory (REQUIRED CHANGE)
# This variable defines where the processed file will be saved
OUTPUT_DIR = r'/Users/Christoph/Desktop/MasterarbeitUniWien/5levels_aviation'

# Column Names (DO NOT CHANGE THESE)
# These identify the specific columns in your Excel sheet for titles, categories, and the new
levels
TITLE_COLUMN = 'exp_1_title'
CATEGORY_COLUMN = 'category_neo'
LEVEL_COLUMN = 'airline_level_matrix_classified'  # Standard column name

# Creates the output directory if it doesn't exist yet
os.makedirs(OUTPUT_DIR, exist_ok=True)

# HELPERS & KEYWORDS

def clean_text(text: Any) -> str:
    """Standardizes text for regex matching."""
    # Converts text to lowercase, removes special characters, and adds padding for matching
    if not isinstance(text, str):
        return ""
    text = str(text).lower().strip()
    # Remove punctuation/symbols for safer matching, keeping only word chars and spaces
    text = re.sub(r'[^\w\s]', ' ', text)
    text = re.sub(r'\s+', ' ', text).strip()
    return " " + text + " "  # Pad with spaces for word boundary matching


# Universal keywords for the new X.1 Junior/Trainee tiers
# Keywords used to identify entry-level or student positions
JUNIOR_KEYWORDS = [
    ' intern ', ' apprentice ', ' trainee ', ' cadet ', ' student ', ' praktikum '
]
```

```
# Manual Exclusions (Per User Request)
# Specific words that signal a job should be ignored or marked unclassified
MANUAL_EXCLUSION_KEYWORDS = [
    ' non flying ', ' non-flying ', ' back office ', ' back-office ',
    ' ground staff ', ' recruitment '
]

# Technic/Maintenance specific keywords (L45-L49 track)
# List of terms related to aircraft engineering and maintenance
TECHNIC_KEYWORDS = [
    ' mechanic ', ' maintenance ', ' technician ', ' tech ', ' line maintenance ',
    ' base maintenance ', ' mro ', ' hangar ', ' engineer ', ' engineering ',
    ' aviation engineer ', ' maintenance engineer ', ' reliability engineer ',
    ' flight engineer ', ' fleet engineer ', ' aircraft engineer ', ' licensed engineer ',
    ' b1 ', ' b2 ', ' b1 b2 ', ' cat b1 ', ' cat b2 ', ' easa b1 ', ' easa b2 ',
    ' avionics ', ' structures engineer ', ' ame ', ' camo ', ' continuing airworthiness ',
    ' engine engineer ', ' powerplant ', ' a&p ', ' a & p ', ' propulsion ', ' fuel system ',
    ' sheet metal ', ' ndt ', ' non destructive ', ' spares ', ' parts ', ' inventory ',
    ' technical operations ', ' load planner ', ' load planning '
]

# LOGIC: TRACK DETECTION

def get_job_track(title_clean, category_clean):
    """
    Determines job track: Technic, Cockpit, Cabin, Ground, or Corporate.
    Priority order: Corporate Override (The Fix) > Technic > Flight Operations > Ground Ops
> Corporate
    """
    # Override 0: FORCE CORPORATE for specific support/universal titles, overriding
category.
    # THIS IS THE PRIMARY FIX TO PREVENT L39 SPILLAGE.
    if any(x in title_clean for x in [' administrative ', ' executive assistant ', ' recruiter ', ' human
resource ',
                                      ' administrator ', ' technology ', ' it ', ' support specialist ', ' analyst ',
                                      ' planner ', ' developer ', ' financial ', ' sales ', ' quality ', ' legal ',
                                      ' paralegal ', ' accounting ', ' procurement ', ' business partner ',
                                      ' compliance ',
                                      ' auditor ', ' technical writer ', ' product owner ', ' lead ', ' manager ',
                                      ' coordinator ']):
        # Check for strong Ground Ops terms to prevent over-filtering legitimate Ground Ops
Managers/Leads
        if not any(x in title_clean for x in
                   [' station manager ', ' ramp manager ', ' ground handling manager ', ' duty officer ',
' shift manager ', ' shift lead ']):
            return "Corporate"

    # 1. TECHNIC/MAINTENANCE TRACK
    if any(x in title_clean for x in TECHNIC_KEYWORDS):
        return "Technic/Maintenance"
    if 'technic' in category_clean or 'maintenance' in category_clean:
```

```
        return "Technic/Maintenance"

    # 2. COCKPIT TRACK
    if any(x in title_clean for x in
           [' captain ', ' first officer ', ' second officer ', ' pilot ', ' aviator ', ' tre ', ' tri ',
            ' type rating examiner ', ' type rating instructor ', ' flight instructor ', ' simulation
instructor ', ' sim instructor ']):
        return "Cockpit"
    if 'cockpit' in category_clean or 'pilot' in category_clean:
        return "Cockpit"

    # 3. CABIN TRACK
    if any(x in title_clean for x in
           [' flight attendant ', ' stewardess ', ' steward ', ' cabin crew ', ' purser ', ' inflight manager
']):
        return "Cabin"
    if 'cabin' in category_clean or 'flight attendant' in category_clean or 'inflight' in
category_clean:
        return "Cabin"

    # 4. GROUND OPS TRACK
    ground_keywords = [
        ' dispatcher ', ' flight dispatcher ', ' operations controller ', ' ops controller ',
        ' station manager ', ' station supervisor ', ' duty manager ',
        ' ramp ', ' gate ', ' ticket agent ', ' check in ', ' baggage ',
        ' loadmaster ', ' turnaround ', ' ground handling ', ' customer service agent ',
        ' airport operations ', ' airport services ', ' station officer ', ' operations agent '
    ]
    if any(x in title_clean for x in ground_keywords):
        return "Ground Ops"
    if 'ground' in category_clean or 'airport' in category_clean or 'station' in category_clean:
        return "Ground Ops"

    # 5. DEFAULT: CORPORATE
    return "Corporate"


# LOGIC: LEVEL ASSIGNMENT


def assign_airline_matrix_level(row) -> Tuple[str, str]:
    """
    Assigns a level (00-49.1) based on Title, Category, and Track logic.
    Returns: (Level Code, Derived Track)
    """
    title_raw = row.get(TITLE_COLUMN, '')
    title = clean_text(title_raw)
    category = clean_text(row.get(CATEGORY_COLUMN, ''))


    # PHASE 0: Blank/Invalid Titles & Manual Exclusions
```

```
    # Level 13: Exclusion/Blank/Invalid Title
    if not title or pd.isna(title_raw) or str(title_raw).strip() == '':
        return "Level 13 (Exclusion/Blank/Invalid Title)", "Unclassified"

    # Level 13: Non-aviation Instructor Logic
    if " instructor " in title and not any(x in title for x in [' flight ', ' sim ', ' type ', ' tri ', ' tre ']):
        return "Level 13 (Exclusion/Blank/Invalid Title)", "Unclassified"

    # Check for manual exclusions (General words like back-office etc.)
    if any(x in title for x in MANUAL_EXCLUSION_KEYWORDS):
        return "Level 13 (Exclusion/Blank/Invalid Title)", "Unclassified"


    # PHASE 1: Universal Top Management (Levels 00 - 06)


    # LEVEL 00: Board of Directors
    if any(x in title for x in [' board of ', ' chairman ', ' chairperson ', ' supervisory board ']):
        if ' assistant ' in title: return "Level 08 (Specialist/Analyst/Expert)", "Corporate"
        return "Level 00 (Board)", "Universal"

    # LEVEL 01: CEO & President
    if any(x in title for x in [' ceo ', ' chief executive officer ', ' president ']):
        if ' assistant ' in title: return "Level 08 (Specialist/Analyst/Expert)", "Corporate"
        if not any(x in title for x in [' svp ', ' vp ', ' vice president ']):
            return "Level 01 (CEO)", "Universal"

    # LEVEL 02: Executive/Board of Mgmt
    if any(x in title for x in [' cfo ', ' coo ', ' cto ', ' cmo ', ' cio ', ' evp ',
                                ' executive vice president ']) and ' assistant ' not in title:
        return "Level 02 (Executive/Board of Mgmt)", "Universal"

    # LEVEL 03: SVP
    if any(x in title for x in [' senior vice president ', ' svp ', ' senior vp ', ' sr vp ', ' sr vice
president ']):
        if ' assistant ' in title: return "Level 08 (Specialist/Analyst/Expert)", "Corporate"
        return "Level 03 (SVP)", "Universal"

    # LEVEL 04: VP/MD (Universal)
    if any(x in title for x in [' vp ', ' vice president ', ' managing director ']):
        if ' assistant ' in title: return "Level 08 (Specialist/Analyst/Expert)", "Corporate"
        return "Level 04 (VP/MD)", "Universal"

    # LEVEL 05: Senior Director (Includes Head of ground/flight/maintenance)
    if any(x in title for x in [' senior director ', ' sr director ', ' sr. director ', ' general manager ',
                                ' head of maintenance ', ' head of flight ', ' head of ground ']):
        if ' assistant ' in title: return "Level 08 (Specialist/Analyst/Expert)", "Corporate"
        return "Level 05 (Sr. Director/GM)", "Universal"

    # LEVEL 06: Director / Regional Manager
```

```
    if any(x in title for x in [' director ', ' head of ', ' regional manager ', ' country manager ']):
        if ' assistant ' in title: return "Level 08 (Specialist/Analyst/Expert)", "Corporate"
        return "Level 06 (Director/Head of)", "Universal"


    # PHASE 2: Track-Specific Logic

    track = get_job_track(title, category)

    # >>> TRACK: TECHNIC / MAINTENANCE <<<
    if track == "Technic/Maintenance":
        # L47: Specialized Manager / Shift Lead / Tech Ops Fix
        if (any(x in title for x in [' manager ', ' mngr ', ' lead ', ' shift ', ' supervisor '])) and \
           (any(x in title for x in [' camo ', ' airworthiness ', ' line maintenance ', ' airframe ', '
engine ',
                                     ' structure ', ' avionics ', ' technical fleet ', ' spares ', ' parts ',
                                     ' inventory ', ' technical operations ', ' load planner ', ' load planning ']):
            return "Level 47 (Specialized Manager)", track
        if 'maintenance manager' in title or 'engineering manager' in title:
            return "Level 47 (Specialized Manager)", track

        # L48: Shift Lead/Supervisor/Planner/Controller
        if any(x in title for x in [' shift lead ', ' shift manager ', ' supervisor ', ' maintenance
controller ',
                                    ' maintenance planner ', ' auditor ']):
            return "Level 48 (Lead/Supervisor/Planner)", track

        # L49.1: Apprentice/Intern
        if any(x in title for x in JUNIOR_KEYWORDS):
            return "Level 49.1 (Technic Apprentice/Intern)", track

        # L49: Technicians, Engineers, Licensed Roles
        return "Level 49 (Technician/Engineer/Licensed)", track

    # >>> TRACK: COCKPIT <<<
    elif track == "Cockpit":
        # L17: TRI/TRE/Instructor
        if any(x in title for x in [' tre ', ' tri ', ' type rating examiner ', ' type rating instructor ',
                                    ' examiner ', ' flight instructor ', ' simulation instructor ', ' sim instructor
']):
            return "Level 17 (TRE/TRI/Instructor)", track
        # L18: Captain
        if any(x in title for x in [' captain ', ' cpt ', ' commander ', ' pic ', ' lead pilot ']):
            return "Level 18 (Captain)", track
        # L19: FO/SFO/Cadet
        return "Level 19 (FO/SFO/Cadet)", track

    # >>> TRACK: CABIN <<<
    elif track == "Cabin":
        # L27: Trainer
        if ' trainer ' in title:
```

```
            return "Level 27 (Trainer)", track
        # L28: Purser
        if any(x in title for x in [' purser ', ' senior flight attendant ', ' cabin manager ', ' csm ', '
supervisor ']):
            return "Level 28 (Purser/Sr FA)", track
        # L29: Flight Attendant
        return "Level 29 (Flight Attendant)", track

    # >>> TRACK: GROUND OPS <<<
    elif track == "Ground Ops":
        # L37: Station Manager/Ops Manager
        if ' station manager ' in title or ' station lead ' in title or ' operations manager ' in title:
            return "Level 37 (Station Manager/Ops Mgr)", track
        # L38: Supervisor/Coordinator
        if any(x in title for x in [' duty manager ', ' shift manager ', ' dispatcher ', ' coordinator ', '
supervisor ', ' auditor ']):
            return "Level 38 (Mgr/Dispatcher/Coordinator)", track
        # L39.1: Ground Apprentice
        if any(x in title for x in JUNIOR_KEYWORDS):
            return "Level 39.1 (Ground Ops Apprentice/Intern)", track
        # L39: Standard Ground Agent
        return "Level 39 (Clerk/Agent/Assistant)", track

    # >>> TRACK: CORPORATE <<<
    else:
        # Specialist/Analyst check for Assistants
        if ' assistant ' in title:
            return "Level 08 (Specialist/Analyst/Expert)", "Corporate"
        # L07: Manager / Lead
        if any(x in title for x in [' manager ', ' lead ', ' principal ', ' assistant manager ']):
            if any(x in title for x in [' project manager ', ' product manager ']):
                return "Level 08 (Specialist/Analyst/Expert)", "Corporate"
            return "Level 07 (Manager/Lead/Principal)", "Corporate"
        # L09: Corporate Junior
        if any(x in title for x in JUNIOR_KEYWORDS) or ' junior ' in title:
            return "Level 09 (Intern/Junior/Apprentice)", "Corporate"
        # L08: Default Specialist
        return "Level 08 (Specialist/Analyst/Expert)", "Corporate"

# MAIN EXECUTION

def main():
    """Execution entry point that handles file reading and processing logic."""
    timestamp = datetime.now().strftime("%Y.%m.%d_%H.%M")

    # Check if the input file exists
    if not os.path.exists(INPUT_FILE_PATH):
        print(f" Error: Input file not found: {INPUT_FILE_PATH}")
        print("Please check and update the 'INPUT_FILE_PATH' variable in the configuration
section.")
        return
```

```
    try:
        print(f"📖 Reading file: {os.path.basename(INPUT_FILE_PATH)}...")
        df = pd.read_excel(INPUT_FILE_PATH)
        df.columns = [c.lower().strip() for c in df.columns]

        # Check for column existence
        if TITLE_COLUMN not in df.columns:
            print(f" Error: Missing column '{TITLE_COLUMN}'. Available columns:
{list(df.columns)}")
            return

        print(" Applying Refined Matrix Logic...")
        # Apply the logic row by row
        results = df.apply(assign_airline_matrix_level, axis=1)

        # Unpack results into two new columns
        level_values = [r[0] for r in results]
        track_values = [r[1] for r in results]

        # --- COLUMN INSERTION (RETAINED VERBOSE LOGIC) ---
        # Inserts the new track and level information back into the data
        try:
            target_index = df.columns.get_loc(TITLE_COLUMN)
            df.insert(target_index, 'derived_track_final', track_values)
            df.insert(target_index + 1, LEVEL_COLUMN, level_values)
        except Exception:
            df['derived_track_final'] = track_values
            df[LEVEL_COLUMN] = level_values

        # --- VALIDATION STATS (RETAINED VERBOSE LOGIC) ---
        # Calculates summary statistics for the classification process
        total = len(df)
        unclassified = len(df[df[LEVEL_COLUMN].str.contains('Level 13')])
        classified = total - unclassified
        classification_rate = (classified / total) * 100 if total > 0 else 0

        # Calculate the new L39 count for comparison
        new_l39_count = len(df[df[LEVEL_COLUMN].str.contains('Level 39 ')])

        print("\n============================================================")
        print(f" CLASSIFICATION COMPLETE")
        print(f"Total Entries: {total}")
        print(f"Entries Classified (Operational + Corporate): {classified}")
        print(f"Entries Set to L13 (Exclusion/Blank/Invalid): {unclassified}")
        print(f"Classification Rate (Excl. L13): {classification_rate:.2f}%")
        print(f" Ground Ops L39: {new_l39_count}")
        print("============================================================")

        # --- CORRECTED FILENAME FORMAT ---
        output_filename = f'5levels_aviation_{timestamp}.xlsx'
```

```
        output_path = os.path.join(OUTPUT_DIR, output_filename)

        print(f" Saving classified data to: {output_path}")
        df.to_excel(output_path, index=False)
        print(" Script finished successfully! Check your output directory for the Excel file.")

    except Exception as e:
        print(f" CRITICAL ERROR during execution: {e}")


if __name__ == '__main__':
    main()
```

Annex B: Transcripts

B.1. Automotive Industry Interview 1

Interviewer:
Hello [Name], thanks for taking the time and for sharing so openly. As I have written to you in my message on LinkedIn, I am trying to develop a python code that can measure the hierarchical depth with scraped, publicly available LinkedIn data. The interview will be done anonymously, as you wished.

Interviewee:
Hallo Christoph, happy to help. Not compromising the name of the company would be great, even though I am just in the process of leaving it. Otherwise, happy to help!

Interviewer:
Of course! Let me start by sharing the screen with you. Here you can already see quite an extensive list of potential titles. For sure, not every company will have every job title, for example, a 'senior director'. Then this hierarchy level just stays. Therefore, I'd like to ask you if you can share with me the line of command of your department – so from the intern until the CEO of the Group

Interviewee:
Yes, I can tell if for you from top to bottom?

Interviewer:
Whatever you prefer. I'm fine either way.

Interviewee:
So, if we start with the CEO – hm, I was in sales. So then came the Chief Marketing Officer. Then there was the VP or "Principal Department Leader" [Hauptabteilungsleiter]. Afterwards came the Director. And then there were 'Teams' which can be compared to small departments. Our department or team, whichever term you prefer, was called "Sales Projects". Then came the Head of Business Development and Head of Strategy.

Interviewer:
Okay, thanks. In fact, the "Head of" are the ones giving me the greatest headache since some call themselves "Head of Marketing" and are indeed CMOs, the other ones are team leaders.

Interviewee:
That I fully understand. For us, they were team leads. So there were two teams deeply intertwined, but yes, one was called "Head of Marketing", the other one "Head of Strategy". Within the teams there were managers, project leads, project coordinators and then also the interns and working students.
Interviewer:
Did I understand correctly that the managers and projects leads were within the same team on the same level?

Interviewee:
More or less; the managers were in a different renumeration scheme, in terms of their work, they were very similar to the projects leads. On the other hand, if it is about hierarchy, it was the manager who guided the project leads, but that was not official.

Interviewer:
And then below the working students and intens?

Interviewee:
Yes, indeed. And the project coordinators are like projects leads “light” in terms of responsibility. But mostly they are much younger?

Interviewer:
So, the projects leads are more of the seniors and the coordinators more the juniors?

Interviewee:
Yes, that’s correct. Not official, but unofficially indeed. And in terms of salary as well.

Interviewer:
Okay, that is something I have seen now multiple time, that there is, for example, an analyst and a senior analyst, but in terms of hierarchy they are on the same level. Great, thanks already!

Interviewee:
What I want to add. This concept was very similar in all main departments. We had HR, procurement, finance, sales and IT.

Interviewer:
So, there were VPs for all these departments?

Interviewee:
No, these were all part of the board. Procurement and Logistics were combined; finance and IT were independent – as well as HR. Distribution and marketing too. Plus, I believe, let me think; … oh yes, research and development which split up in model ranges but on a VP-level.

Interviewer:
How was it with production?

Interviewee:
Hm, I believe they were part of procurement but let me check that. … [checking an internal site]. Oh, I am sorry, production and logistics are combined; procurement is independent. Yes, definitively, procurement is independent?

Interviewer:
Do you maybe happen to know if the factories had VPs each?

Interviewee:
Hm, that is a really good question?

Interviewer:
Yes, it is. Particularly it is so interesting because that is were so many people work, for example, a welder or an electrician.

Interviewee:

That is very hard to answer since in the automotive industry a lot is done through subcontracted labor and labor leasing. But I believe the factories are organized by models. Hm, yes, that makes sense. I am pretty sure, that the VPs of a model line are also responsible for the respective production factories.

Interviewer:
So, looking back, that was the most part I needed. Also interestingly, there are not that many levels. If I count right a maximum of seven.

Interviewee:
Yes, that could be it. Also, if you have any further questions, I am always happy to answer and will try to find some more research!

Interviewer:
The let me thank you sincerely for taking your time and for sharing so openly. You really helped me a lot! Have a great rest of the day!

Interviewee:
Anytime! Have a great day to and best of success!

Interviewer:
Thanks a lot! Good-bye!

B.2. Automotive Industry Interview 2

Interviewer:
Hello [name]! Thanks for taking your time!

Interviewee:
Hello! Yes, [name] told me you're looking for an interview partner with automotive industry expertise. Happy to help!

Interviewer:
Thanks a million, in advance! May I just introduce the topic very shortly?

Interviewee:
Oh yes indeed. [Name] just told me it's about hierarchies, but in which sense?

Interviewer:
That's correct. It's about measuring hierarchies to be more exact. Compared to financial data, which is publicly available, organizational charts are not. The goal is to find out, how successful business in certain industries are given with their hierarchical depth. In other words, are more levels between the intern and the CEO more profitable or maybe even less.

Interviewee:
Sounds interesting, happy to help you out here. But I am asking myself, how I can help you on this?

Interviewer:

I'm trying to find out, which position titles refer to which level of a hierarchy. For example, if there are directors, VPs, managers, managing directors and so on. I am building a python script that will look for certain keywords and then put them in – how can I say this – a shelf of hierarchies on the correct level.

Interviewee:
Hm, I can tell you about the department where I used to work for a long time, would that help?

Interviewer:
Yes indeed, that is exactly what I need. Just one line of command would already immensely help me.

Interviewee:
Well, then let's do then example where I have worked and we'll try to map the way all the way up to the CEO.

Interviewer:
Yes, please, amazing!

Interviewee:
So, until the end of my occupation at [company name] I've worked for research and development of motors and gears. In this development environment of motors and gears, the design engineer and the constructor were the lowest level.

Interviewer:
Okay, good. I've you hear me typing, I am just taking the notes.

Interviewee:
Okay. The design engineer was a subordinate of a team lead. For example, the team lead of development. But if you look at the motor you can divide into various segments. And there, one team lead of development would be, for example, 'crankcase' or 'crank assembly'.

Interviewer:
Good, thanks for the example. Do I understand it correctly, that for each of the segments there are team leads?

Interviewee:
Indeed, team leads for developments. Within those groups there are so-called DSEs, Design Release Engineers. Those engineers, those DSEs, together with the constructors. So those were in a development team. Another example would be the 'valve gear', that docks to the crankcase. For that, there was also an own team lead with the subordinates called DSEs and constructors.

Interviewer:
The team lead of crankcases and valve gears were on the same hierarchical level?

Interviewee:
Yes, development leader crankcase and valve gear were on the same level. Those were not only responsible for 1 type of motors, but for various motors which were produced in Vienna,

Hungary, Kaiserslautern and at other places. So – important – they were responsible for different motor families, but *only* for valve gears, nothing else.

Interviewer:
So, summarizing. The team leads are responsible for small segments of the motors at various geographical places, correct?

Interviewee:
Yes, that is correct.

Interviewer:
How large can I imagine such a team? Or in other words, how many subordinates below one team lead? 10, or maybe even 40?

Interviewee:
Roundabout 10 is realistic. Yes, and above the team leads, there was already the director. But there could be a parallel hierarchal level. There was always a responsible person for projects – for one single motor. That person was a subordinate of director of motor development.

Interviewer:
So there was a director of motor development and below that there was one person responsible for one specific motor.

Interviewee:
After the director, who was next in line?

Interviewer:
Yes, above that there was the responsible person for the powertrain. Hm, how did we call this person? Director, no, no. That was the Vice President powertrain.

Interviewer:
Powertrain is the correct term?

Interviewee:
Yes, that includes the motor plus the gear. Yes, indeed, that one was responsible for all motors and gears of [company name]. Potentially there was also a position between the team leads and the director. But, important to know, the director had subordinates not only in development but also validation. So next to development there was also test and validation.

Interviewer:
But those persons of test and validation still reported to the director of motor development?

Interviewee:
Yes indeed, that is correct. But even more, for example, the calculation of product costs, so actually even my old department, product costing. And what because of the matrix structure of a project, where a responsible project manager for a specific, also procurement was part of it. Even though procurement was at a different hierarchical stream. So from a hierarchical perspective, the projects were similarly set up.
Hm, let me give you an example. The project procurer of the motor families that were produces in Vienna, that person was part of a project team with his/her, for example,

purchasing agents for, for instance, a cylinder piston. Do you get what I mean? I am only trying to explain you the hierarchy for the development of the motor.

Interviewer:
Oh yes, that I get and I am happy that you can give me such a lot of details for the part of the company where you had worked in.
Do you know, after the Vice President, who came next?

Interviewee:
Well, the exact title I am not sure. Next to the Vice President powertrain there was also the Vice President for the platform of the vehicle. That includes, for example, wheel suspension. So everything from wheels to wheel suspension.

Interviewer:
The base plate too?

Interviewee.
Yes, that as well.

Interviewer:
The Chassis as well?

Interviewee:
No, that is part of the design and exterior.

Interviewer:
Okay, I get that. Very good.

Interviewee:
But if we think all that together. The main responsible for the entire vehicle. Hm, how did we call that person? Difficult to say. I cannot recall at the moment.

Interviewer:
Well, that is totally okay. Better than any wrong information. I'll not that there is a responsible person for the entire vehicle. There are numerous titles, like managing director, a Senior Vice President, or straight away the Chief Technology Executive.

Interviewee:
Managing director sounds very familiar to me.

Interviewer:
And then came the CEO?

Interviewee:
Yes, so the managing directors, of which I am not hundred per cent sure about the title were already part of the board. Hm, talking about this, I do think there was a layer in between. It was not that quick between the ordinary designer to the CEO. But I do not know at the moment.

Interviewer:

Okay, no worries. I have some raw data of General Motors and also PSA, maybe I can find the correct position name there. Otherwise, that seems very logical. It matches the data I have. So, in fact, that was already it. I do not have any further questions.

Interviewee:
Oh, that was already it?

Interviewer:
Oh yes, it does not have to be super long. I am genuinely very thankful that you took the time and that you were so honestly speaking, also about what you know and where you were uncertain. That helps me to know where to dig deeper.

Interviewee:
Nothing to thank for. I am happy to help. Just in case you cannot get the data from anywhere else, I can offer to retain some more data from old files, but that might take a while.

Interviewer:
For the moment I am fine. If you could find an organization chart, I would be very thankful, otherwise I am also fine.

Interviewee:
Yes, I'll try to find that.

Interviewer:
Other than that, thank you once again and enjoy your lunch!

Interviewee:
You too! And all the best for your work. Greetings to our mutual friend! Goodbye!

Interviewer:
Will do! Goodbye!

B.3. Automotive Industry Interview 3

Interviewer:
Hi NAME, thanks a million for taking the time and for helping me out here! I highly appreciate it!

Interviewee:
Hi Christoph! Anytime, happy to help!

Interviewer:
Thanks! I've given you a small introduction to the topic; do you want me to repeat the most important tasks?

Interviewee:
Yes, that would be great since our last messages were some weeks ago.

Interviewer:

Sure! So, for quite some time it seemed to be en vogue to have the flattest possible hierarchies. Now, some studies showed, that this is not always the case and not suitable for every company. Thus, as part of a bigger team, we are trying to measure the hierarchies of companies in various business branches. My topics are airlines and car manufacturers. For the latter, you could really help me out if you map the line of command that you have in your company.

Interviewee:
Thanks, that helps! In fact, we do not have that much of a hierarchy. I am working for the European Technical Center of the motor group of our big mother company that is located in South Korea. We're specialized on development and there are also several other parts, like an own company for steel.
For our Motor Group there is the production and the development and design department – that's where I work. We're responsible for Europe entirely and we are an own limited company within the group. So we are rather small, only 500 employees.

Interviewer:
So, you do research and design?

Interviewee:
Yes, our core task is the development and design of existing as well as new cars. So, we adjust the products of the mother company for the European market.

Interviewer:
Thanks for the background information. For this thesis it would be really helpful how you organize 500 employees. For example, from the apprentice to the highest-ranking person of your company. Or even, who at your company is even above your company.

Interviewee:
Fine, here I definitely have some information.

Interviewer:
Great! Here you can see already some parts of the coding already, but of course your input will make it more concrete and accurate. I've already tried to map some positions with some help of online sources, like chairmen, CEOs, SVPs, VPs, specialist, senior specialist, juniors, apprentices and so on. However, I do need to validate and potentially correct this. That is exactly where your help is needed.

Interviewee:
So, we do not have any apprentices, but we do have students. There are several tracks, but they are all organized the same way. I'm in the career track for administrative tasks. There are also the IT, the design, and the R&D tracks. For my part there is the specialist, the advanced specialist and the senior specialist. For R&D there's the engineer, the advanced engineer and the senior engineer. Same for design: designer, advanced designer, and senior designer. However, these are not really different hierarchies. If I get promoted, I'd become the advanced specialist, but this are kind of the levels of employees, but there is no command over others added to these positions.

Interviewer:
Noted, thanks already!

Interviewee:
Afterwards, there's the manager. And then, hm – let me have a look! Oh yes, the next in line is the senior manager, followed by the head of department and then director. Afterwards there's the managing director.

Interviewer:
Am I correct that the managing director is then the highest level of the limited sub-company for which you work?

Interviewee:
Correct, he's the highest executive of our company. Maybe what is also interesting for you, we have five different divisions, but only three of which have a director. For the two other divisions it ends with the head of department. Those report directly to the managing director. For those with the director, that is the normal line of command.

Interviewer:
Thanks, that is already super helpful. Also, that you say that a senior engineer is not an own hierarchical level.

Interviewee:
However, even though there is no hierarchical layer, it is impossible to promote from a specialist to a team leader.

Interviewer:
Just to make sure, the team lead is the same as the manager?

Interviewee:
Oh, yes indeed.

Interviewer:
Do you maybe know how comes next after the managing director? Maybe, there is a head of global design, or someone completely different, like the CTO?

Interviewee:
Oh, let me try to find that out on the intranet. Wait a minute please.

Interviewer:
No worries, take your time.

Interviewee:
So, the boss of our managing director is called 'Head of Research and Development'. But to be honest I am not 100% sure. There are the Motor Group and the Motor Company, so it could also be the 'head of vehicle development'. That makes it really complicated. So, I cannot tell you with absolute uncertainty.

Interviewer:
I totally understand the complexity. We also have a group CEO and an airline CEO and it is very hard to understand who reports to whom. Also, with all the 'Head of' titles, it gets quite complex.

Interviewee:

Indeed, but usually we the ‘Head of’ are department heads.

Interviewer:
Do you happen to know, who’s above the head of R&D.

Interviewee:
I think that should be the CTO. Then there would be the CEO.

Interviewer:
Okay, great, sounds great!

Interviewee:
Yes, so usually we just talk to our team lead, and sometimes with the department lead. Then there’s already the director and then the managing director.

Interviewer:
That’s really not a lot of levels. Do you maybe have an organizational chart of a production line? That I am still missing, and it is quite hard to get one.

Interviewee:
I am afraid, I don’t. I already tried to get one before the interview, but we do not have any as we’re at least on paper an independent company.

Interviewer:
No worries, you have already helped me a lot! I am really, really thankful for your expertise!

Interviewee:
No need to thank.

Interviewer:
Oh yes indeed! So, without further ado, may I wish a great rest of the day and thank you!

Interviewee:
Any time! Goodbye!

Interviewer:
Goodbye!

B.4. Automotive Industry Interview 4

Interviewer:
Hallo [name]! Thanks for taking the time!

Interviewee:
Hello Christoph! Anytime! My nephew told me, that you’re still looking for interview partners! I know how hard this can be, so happy to help!

Interviewer:
Thanks a lot! Yes, particularly for the automotive industry it was quite difficult to me. That’s why I am so thankful that you share your expertise! A little bit of background information: I

am developing a Python code that can be fed with LinkedIn data. The goal is, to find out how hierarchical a company is since companies usually do not publish organizational charts. This code will then help, to find out, whether, for example, it's better to have more or less hierarchical levels between the CEO and the intern to be successful in certain industries. For me, these industries are car manufacturers and airlines.

Interviewee:
Well, if you're looking for someone from a car manufacturer, it seems we've good match! How can I help you?

Interviewer:
If you can give me an overview how your company is structured, it would really help. Particularly if you know the job titles, because the script will look for those. Also, the line of command in your department would be extremely interesting.

Interviewee:
Sure! The structure that is in our main department is the same for all main departments. I am part of "Concepts, Architecture and Integration" and then this splits up into departments and then later in groups.
This structure, from division, main department, department and groups, that one is everywhere very similar, so in fact 5 hierarchical layers. This really is all departments very much the same.

Interviewer:
So, within the group there is the engineer, a group leader, a department leader, the main department leader, the division leader and then the CEO?

Interviewer:
Yes, indeed. Here in this organizational chart, you can see the division leader [name], then the main department leader [name] who oversees four departments – but this number varies. Same as the number of groups per department. Actually, this varies a lot. We have a minimum size for a group of – let me think. Hm, I have to put this in other words. One organizational unit, no matter whether it's a group or a main department, there must be at least at least eight subordinates for one hierarchical boss. We even had the case, that there was no other structure below a main department and they were just eight people. Why was this done? These were very important, but very specialized departments which needed to act independently.

Interviewer:
Thanks, that is very interesting information and new for me!

Interviewee:
Well wait, this is the – how may I say that – the line function. However, we also work in this company with V-models? Do you know this from academia?

Interviewer:
I'm afraid I don't.

Interviewee:
No worries, that is the cooperation between line functions and product line functions. What you have just seen, those are the line functions, also called centers of competence. But if you

look here, there are some abbreviations which stand for the project organization. For example, these two letters stand for the project function of 3 rather similar car models. Although the structure is similar to the line functions, there are way less employees encompassed. The projects use the people from the line functions.

Interviewer:
That means one can work in the line function and in the project function simultaneously.

Interviewee:
Yes, indeed, you can be a specialist in the construction of side frames in the line function, so you're organized in the car body, but you've a project, for example, this specific car model, where you will be technically leaded within the project and from a disciplinary point of view you're still in your line function department. That is why we call it V-model because you have two bosses, one disciplinary boss and one functional boss.

Interviewer:
Oh, I understand that. Kind of a matrix organization. I have the same with my two bosses even working for two separate subsidiary companies in two different countries.

Interviewee:
Yes, indeed. The project organization is in this division, but the worker can be from a different division. Let's have a look at this division of the board, because you have, of course, the main departments, but they do not only include the development but also the factories. So that bord member oversees the development and the factories. A factory manager is the same as a leader of a main department.

Interviewer:
That is very interesting, since I have not yet received a lot of information on factories. If it is possible for you, can we map the hierarchy of a factor, from the welder, someone who simply drives screws or operates a machine to the factory manager. That would be utterly interesting to me.
Interviewee:
The hierarchies are quite simple.

Interviewer:
Is there something like a shift lead?

Interviewee:
Well, here you really need to pay attention. The big difference is there is a factory manager, below there is the assembly manager. The responsibilities are then divided by the various assembly lines. For instance, there's the chassis assembly, there's the finish, etc. There are various assembly lines with different status of the cars. Here you have the specialties – take the finish – there're the groups. Let me check this out in our system; hm, maybe not the best example. Let's look at a differently assembly line: if you zoom in here to this special assembly line, then you see the approximately 200 workers for one assembly manager. So officially all of one group are below one assembly manager. But self-speaking, it is impossible to manage so many people. Therefore, you can see various team structures: plant operators, foremen, master craftsmen. Those are in charge of leading employees. But there is no official hierarchy for that. So, it is very common to have a team structure even within a team. Also, in our development team, it is very common to have such structures. Nonetheless, there is no disciplinary responsibility. That is clearly at the group lead level.

Interviewer:
Oh, that is clear. I understand that and it's well noted. Thanks for those insights!

Interviewee:
Yes, so summarizing, disciplinary leading is always at the team lead, but from a functional point of view it is common to have more subdivisions, which are then the team leads' responsibilities.

Interviewer:
Thanks, lots of information. Just give me some seconds to finish typing.

Interviewee:
Yes, take your time. I was just a bit surprised why I could not see all the factory workers in our intranet. For our plant in [location], it was clear to me, for the rest I also do not know by hard.

Interviewer:
Luckily, I have already visited that plant! I'd like to ask you one more thing: in my company internal and external titles vary. For example, for the outside there are VPs, Senior Directors, Senior Managers; internally they are all called "Head of …". How is that in your company?

Interviewee:
No, not that I would know of. I just now the division lead, main department lead, department lead, group lead plus the unofficial team leads. Group leads are normally non-tariff employees and are also called lower-segment leader. Department leads and sometimes also main department leads, those are a bit nebulous, those are so-called middle-segment leads, and the other main department leads, and division leads are so-called higher-segment leaders. Those are again divided into those with and without power of attorney. That is more less the structure, also of the non-tariff employees.

Interviewer:
Oh, so you also have this non-tariff system. I thought only my company does that.

Interviewee:
No, we also have that. There are even some super highly skilled employees that are non-tariff, but they try to reduce that and only allow that for employees who have direct supervisory responsibility.

Interviewer:
Thank you, [Name]! That was extremely helpful!

Interviewee:
I hope the interview matched your expectations?

Interviewer:
Oh yes, even more than that. I am extremely grateful for your insights and will handle them with care.

Interviewee:
Oh yes, no worries. The information is quite easy to obtain.

Interviewer:
Well, yes! But only if you have access to the intranet.

Interviewer:
Oh yes, that I believe. Therefore, I am so thankful that you took your time!

Interviewee:
Then I wish you best of success for your thesis. If you have any questions, you can always shoot me an email. But please keep it confidential and anonymous.

Interviewer:
Of course! That is what we agreed upon and I can guarantee you, I will stick to that! Then, let me wish you a great evening and thanks again!

Interviewee:
Anytime, all the best! Goodbye!

Interviewer:
Thank you! Goodbye!

B.5. Aviation Industry Interview 1

Interviewer:
Hallo [name]! Thanks a million for taking your time.

Interviewee:
Hallo Christoph! Anytime and happy to help you with your thesis!

Interviewer:
As asked by you, we will keep this interview anonymous. I will try to map the line of command within our company. Just a short recap: this master thesis aims to develop a python coding that, with the help of scraped publicly available LinkedIn data, measure the hierarchical depth of companies. In the end, it is going to be an important step in finding out, whether more or less hierarchical level benefit or hamper companies in various business branches.

Interviewee:
Sounds like an interesting topic and a question my team asks itself every day!

Interviewer:
Do you know how many employees work in the entire Airline Group?

Interviewee:
I cannot tell you the exact number, but there’re roundabout 100,000.

Interviewer:
That number sounds familiar. And your current position is “Strategy and Organizational Development”?

Interviewee:
Not exactly, that is the department above mine, I work for “organizational development”.

Interviewer:
Seems to be the perfect position for what I need to know.

Interviewee:
Yes, my team tries to find the ideal organization for the group as well as the different airlines. We are divided by the various resort, e.g. for operation, human resources, …

Interviewer:
So, any changes to the organization are done by you?

Interviewee:
Yes. Think of a leadership position that must be newly created or the opposite, two positions merged into one, then I am responsible for that. We then supervise everything from the idea until detailed changes, for example, changes in the process. Also, we evaluate the renumeration of this position.

Interviewer:
Thanks already, let me just quickly take some notes. Do you also have some supervisory duties and to whom do you report them?

Interviewee:
Let me show you the organizational charts. So below the CEO, there is the Chief Organizational Officer, to whom I report directly. I am in the third layer below the board. However, only team leads are represented in the organizational charts, therefore, you cannot see my name there.

Interviewer:
Great, thanks! Besides all the HQ functions that there are, do you also have an overview on pilots, cabin crew, ground ops crew or technicians?

Interviewee:
Not really, because those are not depicted in our organizational charts. Nevertheless, for example, we do see the flight chiefs, who are responsible various aircraft models like 747s or the A320 family. But those below, the first officers for instance, that is not within our scope. That you need to ask in the specific divisions how the career paths are. We also see leaders of cabin crew, but no one below, as, again, they are not depicted in the organizational charts.

Interviewer:
Okay, I will try to find that out accurately elsewhere. For other functions like HR, Pricing, Legal, Revenue, etc.: those are depicted quite well in our organizational charts. Is it always organized the same way? Levels: N1, N2, N3 and below the specialists? Or are there any levels in between?

Interviewee:
No, there is nothing in between. N0 is the board of the corporation, so the group. Next come N1 to N3 and below usually there’s immediately the employee. Roundabout 10 years ago a new rule was implemented, that below the board, there is a maximum of two management levels for all corporate functions and in all operative functions a maximum of three

management levels below the head. However, management functions are not only the Leadership Circle functions 1, 2, and 3, but also the team leads. There is only one exception: in the very large operative divisions like the Technical Department, which were sourced in several years ago to the airline itself, those have deeper structures because the work in a 24/7 system with shifts, something that doesn't exist in the office jobs. Therefore, they have various other functions like a shift lead, however, those do not have any disciplinary duties, but they are responsible for the shift. So, they will not utter any disciplinary warnings or even terminations. Consequently, these have deeper structures, otherwise it usually ends on the level of specialized employees on the N4 layer.

Interviewer:
Okay, I'll note that. One more question, there are quite some persons have operational and managerial duties, for example, pilots. Is this interlocking deliberate or are rumors correct, that this causes inefficacies and frictions and should be abandoned if possible.

Interviewee:
I know those management pilots. Whether there's a trend towards ending such mixed positions I do not know, but I could give you the contact if highly relevant. The organization is part of the strategy, so my team, the persons doing the job are selected by a different department, also if it's a double position and if we aim to get workers with operational experience into management positions.

Interviewer:
Great, thanks. If necessary, I'll ask you for the contact. If one question about the supervisory board. Can it be called on own hierarchical layer?

Interviewee:
The supervisory is definitely above the board. You could even say the supervisory board is the boss of the CEO and also appoints the CEO. That is the same for all supervisory boards of publicly traded companies.

Interviewer:
Good to know. I thought so, but I wanted to reassure from the expert. Sorry for the huge jump, but we go all the way to our beginners like an apprentice. What that be even below the N4 employees?

Interviewee:
No, from a hierarchical point of view, they are normal employees. Of course, if we look at renumeration, there are differences. But from a reporting view, they are on the same level.

Interviewer:
Do you know how it is on the airplane with cockpit and cabin crew? Is a captain on the same level as a purser? Or what about a First Officer? It does not seem correct, to put them below a purser.

Interviewee:
While I am not 100% sure I have a very strong assumption. I do believe it is the captain, because he/she also has the 'householder's right' on the aircraft. But as I said, if you want to be certain you'd have to ask in the operational departments.

Interviewer:

Different topic: do we see a problem in the so-called “title inflation”? In other words: everyone wants to be a manager on LinkedIn.

Interviewee:
Internally we have a regulation that everyone is called “Head of”. That is our internal solution. Externally, we differentiate between the job titles for all the leadership circles. LC3 or N3, that’s the same: then you may call yourself ‘Senior Director. If you are LC2 you may call yourself ‘Vice President’ and if you are LC1, you can call yourself ‘Senior Vice President’. If there are exceptions, which we already had, it something that must be agreed upon between the respective individual and the responsible department. But that is extremely rare and we have put a lot of effort into that to standardize the job titles.

Interviewer:
That really helps! Those are exactly the terms I will look for with my script to define the role in the hierarchy.

Interviewee:
Good to hear, just be aware that there might be mistakes. We cannot control for that and I assume, there are many inflated titles on LinkedIn. Someone from HR would have to check for that, but I do not think anyone does that. But at least, what I told you before, that is the company official regulation.

Interviewer:
How do you deal with projects? Someone could be a project manager, although this employee does not manage any subordinates. Do they get then an internal title?

Interviewee:
There are no official project hierarchies. How can I say that? Projects are not the little brother of line functions. If there is a project, there is the same process as a line function. Depending on the scope it will be evaluated if it is a LC1, LC2, or LC3 position. Then they also possess the title. But there is not internal cannibalization of titles because of projects. For big projects there are own jobs and those people do not have line functions; on the contrary, small projects are simply done within the normal line functions.

Interviewer:
A question to the matrix organization. Am I right that we only have matrix functions in the office?

Interviewee:
We only have matrix position in the so-called A-functions. There are also C-functions; “C” for corporate. There are “G-functions”, “G” for group where we try to leverage synergies. And there are “A-functions”, “A” standing for Airline, which only includes the hub airlines. Only those A-functions have the matrix left.

Interviewer:
Oh, was there more matrix in former times?

Interviewee:
Yes, I think that was changed in 2018. For five years we had the matrix in all corporate functions and support functions, but this did not prove successful and that’s why this was reversed.

Interviewer:
Oh interesting! Thanks for sharing these insights. I'm in a matrix functions, but honestly, I like that; working with so many countries and corporate cultures is an interesting challenge.

Interviewee:
I understand that. Me too.

Interviewer:
Different topic again: are there also "team leads"? Are these automatically LC3 or below?

Interviewee:
Team leads are below LC3.

Interviewer:
One more question that I am interested in: there are disciplinary and functional supervisors. Are the tasks strictly divided?

Interviewee:
Yes, they are. But I must add, many leads have difficulties with the matrix because of its complexity. Once I am a functional manager, one moment later I am disciplinary managers – that is difficult for many, but not for all.

Interviewer:
Will we stick to the matrix though?

Interviewee:
Yes, for sure. The synergies outweigh this complexity by a large scale. When we stopped the matrix for corporate functions, there was a clear will to keep it for airline function. It also does not mean that support functions do what they want, there are still aligned processes. But that the group functions co-decided on every new employee in [different location than HQ] was completely over the top and rightfully abolished.

Interviewer:
Good, actually that was it already! As agreed, I will keep it anonymous and will not write the company.

Interviewee:
Yes, I'd be very thankful for that. I am not sure what may be shared externally and what needs to stay secret.

Interviewer:
I am totally fine with that! Once, again: let me genuinely thank you for your expertise and wish you a great rest of the day!

Interviewee:
Anytime Christoph! If more question come up, do not hesitate to write me an email. Have a great rest of the day, goodbye!

Interviewer:
Thanks for the offer! Goodbye!

B.6. Aviation Industry Interview 2

Interviewer:
Hallo [name]! Thanks for taking your time and for sharing the information! Since we have so far only collaborated between our companies, this interview is completely independent from my employer, and I will not share any data and keep your data anonymous – as agreed.

Interviewee:
Hi Christoph! Yes, got that! No worries, happy to take some time. I'm sorry you had to wait so lang until we could find a time frame.

Interviewer:
No worries. I am super thankful! As already stated in the call when I ask you for this interview, I am trying to program a python code to measure the hierarchical depth of airlines and car manufacturers using publicly available, scraped LinkedIn Data. For that, I need to map job titles of both branches and therefore, I need interviews with experts like you, who know how the titles within your company are called. So let's try to find out the line of command together.

Interviewee:
I'll do my best! Happy to help!

Interviewer:
You're station manager, right?

Interviewee:
Yes, that's right.

Interviewer:
Is there something like a duty or a shift manager?

Interviewee:
No, everything was sourced out, I the last woman standing so to say.

Interviewer:
And who is above you?

Interviewee:
I've a regional manager above me. We are divided into areas. Our region is Europe, but there are various subdivisions called areas. My area includes Switzerland, Austria, Eastern Europe and Southeastern Europe. There is one manager for this area for operative aspects. There is also an area sales manager. Do you need that as well?

Interviewer:
If you have that information, I'd be happy to gain as much information as possible.

Interviewee:
Sure, to my best knowledge, the area for sales is congruent with the operations area. However, below the area manager, for sales, it's about the countries. That is different to the operational area managers, who are subdivided by airports as, for example, the airport

manager of Prague also represents some airports in Southern Poland. That would not be possible for sales.
Interestingly, my regional operations manager is a subordinate of the regional sales manager, even though from a functional point of view, the operations manager is below the Group headquarter. That is really quite complex.

Interviewer:
So, above your operational manager there is the sales manager of the area.

Interviewee:
Exactly, I don't know why but that's the way it is. The sales manager is above, but again, this is just the disciplinary hierarchy, but the functional hierarchy goes to head office.

Interviewer:
Do you happen to know the title of this sales manager or the operational manager? Like Vice President, etc.?

Interviewee:
Oh, for sure not Vice President, but let me look what title this sales area manager has. I don't even know. Let me check, because this gentleman is all new to this position. Wait a minute please. […]
He's called "General Manager". So, he's the sales manager. He's the highest person in our area.

Interviewer:
Good to know! And *your* regional manager? Is he also a general manager?

Interviewee:
No, he's just called "Regional Station Manager". By the way, the sales guy for Austria is called "Country Sales Manager".

Interviewer:
Do you know who is above the General Manager?

Interviewee:
Yes, the general manager is a subordinate of the Senior Vice President of Europe and Latin America.

Interviewer:
That's interesting and coincides very much with my previous mapping out of other interviews. Happy to hear that.

Interviewee:
Sounds good!

Interviewer:
And next in line of command after the Senior Vice President is the COO?

Interviewee:
No, the next would be the EVP Chief Commercial Officer. And then is the CEO.

Interviewer:
EVP and Chief Commercial Officer is the same person?

Interviewee:
Yes, that is correct. And that EVP is a subordinate of the Group CEO – of the holding.

Interviewer:
Do the single airlines also have their own CEOs?

Interviewee:
Well, the Group CEO is also the CEO of one of the group airlines. The other big group airline has an own CEO.

Interviewer:
Ah, that's the way it is. Even though it doesn't make my life easier since the code will look for 'CEOs', but the group CEO is of course above the airline CEO. But yes, I'll deal with it! Thanks! You massively helped me and that would already be it.

Interviewee:
Well, that was an easy task!

Interviewer:
For you for sure, but it's very difficult to get these insights from the outside. Therefore, many thanks! Unless you have any more questions, I'd wish you a great rest of the day!

Interviewee:
I wish you the best of success for your thesis! Have a great day!

Interviewer:
Thanks, you too! Goodbye!

B.7. Aviation Industry Interview 3

Interviewer:
Hi there! Long time, no see!

Interviewee:
Yes, true, the last time was in Brussels! Happy to see you again!

Interviewer:
True that! Thanks for helping out here! Just to make sure, this interview has nothing to do with my employer. It's a private project and I only do it for university

Interviewee:
Okay, I thought so! No worries. Just keep it anonymous.

Interviewer:
Of course, that's what I promised. I've already written you a short introduction, but I am trying to establish a python model that analysis the number of hierarchical levels of airlines and car manufacturers. Therefore, I am trying to find industry experts to get to know the

various job titles within the respective industry. Therefore, I am particularly interested about the job titles of the people in the line of command where you work.

Interviewee:
Yes, I can remember from your message on LinkedIn! No worries, happy to help.

Interviewer:
Great, so let's get started with the line of command of the department where you work–if that's okay?

Interviewee:
Yes, of course, so let me think. I am an analyst and the next in line is my manager. She oversees five workers including me. We are rather small, but as you know, we are a rather small subsidiary. We have a lot of airplanes with around 60, but we they are all rather small for feeder traffic to our hub.

Interviewer:
Yes, that I know. But there is a lot of data in my sample from feeder airlines, so that makes it actually very interesting how you are organized compared to the big players on the market – or, let's say, as a part of a very big player. It's the same in the US with United Connect or American Eagle. But their planes are even smaller, sometimes with just 3-abreast seats.

Interviewee:
Oh, sounds good. I've just found an organizational chart so that I do not tell you something wrong as I was not sure for all positions. So, after by boss, who is a manager, her boss calls himself "Director". My department is too small, but for some larger departments like cabin crew or cockpit crew, there is also a senior director. Above those, there is always a VP, which is almost the end of the line I think, let me check.

Interviewer:
Take your time. Sometimes there is a senior vice president.

Interviewee:
Those, we don't have. But it's quite complicated. So. above the VP there is our EVP, who is the top of our subsidiary airline.

Interviewer:
Okay, interesting. Well noted.

Interviewee:
Maybe you know there is another subsidiary airline for low-cost traffic in both our operating countries?

Interviewer:
You mean [company name]?

Interviewee:
Yes, so they are organized pretty much the same way like us. Their CEO is also an EVP.

Interviewer:
Makes sense, but I forgot about that subsidiary to be honest.

Interviewee:
No worries. Anyways, above that there is the executive committee for the group. There are executives for cabin, cockpit, MRO, flight ops, passenger business, cargo; hm maybe more, but I cannot recall. Just the very big departments are represented in the executive committee.

Interviewer:
Thanks for sharing. Would the next one then be the CEO?

Interviewee:
Yes, so to be exact, the CEO of the airline group.

Interviewer:
Oh yes of course! Do you know the line of command from any other department as well?

Interviewee:
Hm, not really. Maybe cabin, since I had done a project there when I was a trainee, but I am not 100% sure.

Interviewer:
No worries, any information is valuable. I'll try to compare and validate it then with online data and with the hierarchy from my airline.

Interviewee:
Good, so there are the normal flight attendants and above those pursers. We also have senior pursers when they have been working for us for a very long time; not sure if they are an own hierarchy. Those are organized by a cabin crew manager. Then there should be another one and I believe even one more, I think the highest cabin manager of our subsidiary. But again, not 100% sure. What I know, there is someone in the executive committee for cabin crew.

Interviewer:
Thanks, that really helps! Actually, that was exactly the information that I needed. Thanks for sharing.

Interviewee:
That was quick. Happy to help! If you need anything else, then just write me an email. We could do another call.

Interviewer:
Thanks a million! In case I need something more, I'll text you. Until then, may I wish you a great rest of the day?

Interviewee:
Happy to answer. Have a great day! Bye!

Interviewer:
Thanks! Goodbye!

B.8. Aviation Industry Interview 4

Interviewer:
Hi Stefan, sorry for doing it remotely, I did not make it home on time.

Interviewee:
Hi there, no worries.

Interviewer:
Thanks for taking the time. As I have written to you in my short request, I'm trying to develop a Python code to determine the hierarchical depth of airlines. The reason for that is to later, so after my work, find out which form of organization is more successful in the various industries.

Interviewee:
Yes, I can remember!

Interviewer:
Great, for that I am using LinkedIn data, where you I analyze the job title and try to assign them a hierarchical level. For that, I appreciate your expertise in this field. With that I can optimize my coding and avoid and wrong leveling of the various hierarchies; to give a plain example, I do not want to level an FO higher than a Captain.

Interviewee:
Got it, yeah, that would not be correct. Happy to help!

Interviewer:
Amazing, so let's start! I am also happy to you work for a smaller airline that does not belong to a large group. I have not had any interview with someone from such an airline; so, thanks again! To get started, I'd be super interest in how many positions there are between an intern and the CEO, or, for example, Junior First Officer – if that exists – and the CEO. I've tried to separate ops and management, since they function quite differently.

Interviewee:
That is indeed correct and was a smart move. Are you more interested in management hierarchy?

Interviewer:
Actually, I'm highly interested in both sectors. Since you're a pilot, I'm sure you can also tell me about positions like TRI, TRE and so on. I'm very curious on finding that out and will need it in my thesis.

Interviewee:
If we look at the flight itself, there is a single "commander", since particularly in safety critical situations, it must 100% clear, who is in command. That's always the one in the front on the left seat. Only in case there is a training flight, that I might sit on the right seat, but I am still the commander. Like in a driving school. Also if two captains or even two training pilots fly together, only one is the commander. On long-haul flights there is even a Senior First Officer. So, there's the Commander, the Senior First Officer, the First Officer and then the Purser. At last, there are the flight attendant. In case a purser would become sick during the flight, then the flight attendant with the highest seniority would be in charge.

Interviewer:

Okay, makes sense.

Interviewee:
There's more here though: In case there are two captains and an FO on a long-haul flight, it depends on the seniority. However, qualification beats seniority. So, I am a TRI (Type Rating Instructor) and even if my colleague happens to be a captain with a way higher seniority, my TRI gives me the right to choose. But that's very special cases.

Interviewer:
Yes, indeed. It's also not possible to analyze such edge cases with my data. However, am I correct that an FO and an SFO may sit in the cockpit alone during a long-haul flight if the captain sleeps in the crew rest?

Interviewee:
Yes, you're right here. If the SFO works for the captain, he sits on the left seat. If he flies together with the captain, he sits on the rights.

Interviewer:
Noted. Another question I have. You're a TRI. Is the TRE from a hierachical perspective above or below you?

Interviewee:
That position is above mine.

Interviewer:
Above those, are there any team leads or groups how all the pilots are organized.

Interviewee:
No, there's only the fleet chief and his deputy.
Interviewer:
Do they have any fancy title like Vice President, or Senior Vice President?

Interviewee:
Hm, not that I would know of. But let me check. I've a lot of organizational charts prepared here. Just looking for the right line. Give me a second.

Interviewer:
Take your time.

Interviewee:
Here it is: no, he's only called fleet chief.

Interviewer:
Okay, great, thanks. Do you know who's above the fleet chief?

Interviewee:
Afterwards there is the Director of Flight Operations. That one is very important because this position is mandated by the German Aviation Authority. So, my boss is the fleet chief, then the next level is the Director of Flight Operations. So the boss of my boss. The Director of Flight Operations is a direct subordinate of the COO.

Interviewer:
Okay, and the COO is one position below the CEO I assume.

Interviewee:
Yes, correct. So, for the flight operations that's already it.

Interviewer:
Do you know it also about other operational departments? Like Ground Ops. Is there maybe a hub manager?

Interviewee:
Ground operations are difficult, since that is completely outsourced. However, we do have a newly established position of a hub manager, but if you ask me, they themselves do not really know what to do until now. HR I could offer, does that help?

Interviewer:
Oh yes, indeed! For example, how many positions here between the intern in the HR department and the CEO.

Interviewee:
Yes, I can tell you from the organizational chart. But, of course, I am not in that department. You have the Director HR, and below him there is the Head of HR Flight Staff. That one is divided between HR Cockpit and HR Cabin.

Interviewer:
Good to know. Below HR Cockpit and Cabin, are there any team leads or how do you report?

Interviewee:
No, no team leads, but there are three positions HR Cockpit and six for HR Cabin. Which person you are assigned to depends on the first letter of your surname. When I started, there was an HR Cockpit for Captains and one for FOs, but that has changed. The cabin staff have always been organized by their surname.

Interviewer:
Okay, I understand the concept.

Interviewee:
So, that was HR. But there is also the training department reporting to the COO; no actually the Director of Flight Operations and then then to the COO. Also, the Nominated Person Crew Training reports to the Director of Flight Operations. But that person has several managers, for example, for projects as well as for the A320s, the A330s, ground courses.

Interviewer:
Okay, give me a second please to write that down in detail. Quite complicated I must say.

Interviewee:
Relatable. By the way, I also report to the training manager of the A320, since I also work as a TRI. Afte that training manager, next in line is the head of training and that person reports to the COO too.

Interviewer:

Oh yes, such double reporting lines are quite common. But it makes it quite complex.

Interviewee:
Yes, but overall, my airline is quite flat. Actually there only four people until the CEO. The fleet chief, the Director of Flight Operations, the COO and then already the CEO – fairly simple. Same for training: training manager, head of training, COO, CEO. Even the safety manager, that reports directly to the COO, but the safety manager also his safety managers.

Interviewer:
Great thanks, that really helped! Thanks for those in-depth insights.

Interviewee:
Always the happy to help! Also happy to welcome you in the cockpit soon!

Interviewer:
That would be amazing!

Interviewee:
One more comment: Flat hierarchies sound really good, but in aviation, particularly in the airplane itself it can very complicated. If everyone reports to everyone you lose a lot of time which, often, you really don't have.

Interviewer:
You're absolutely correct. Flat hierarchies, unless organized wisely, can make information sharing extremely complex and long-lasting. A good path must be found out. I'm very excited for the results. So far, airlines in the U.S. tend to have more hierarchies than in Europe. But let's see.

Interviewee:
Interesting! Just in case any more questions come up, you can always shoot me a message. Happy to help!

Interviewer:
Thanks a million, and thanks for taking the time! Have a great rest of the day!

Interviewee:
You too! Goodbye!

B.9. Aviation Industry Interview 5

Interviewer:
Hello! Hi there!

Interviewee:
Hey, how are you?

Interviewer:
Great! How about you? Surviving the snowstorm?

Interviewee:

Oh, it's really bad here! Even for Canadians, but we'll get through it!

Interviewer:
All the best for that! Anyhow, thanks a lot for taking the time! Is already written in my email, I will keep this interview anonymous and it is completely independent from my employer and a private project. I am trying to understand better hierarchical layers of airlines and have conducted interviews with several European airlines already. However, since most of the data I have is from North America, I am highly interested about the hierarchical structure in your airline, so that I can improve the Python code I already have and make it more accurate.

Interviewee:
That's fine! Happy to help! Absolutely!

Interviewer:
Can you tell me about the hierarchy in your department, or in other words: how many positions between the intern and the chairman. The reason behind that is to find out whether more or less levels of hierarchy are beneficial for companies of various business branches.

Interviewee:
Great, that comes at a very interesting time because we've just done a major reorganization in the last months to get rid of hierarchies. The management thought there are too many ladders to get to the top and they have cut it to remove some layers. May I show you my screen because I am more of a visual guy? Let me know when you see my screen.

Interviewer:
Absolutely! I can already see it.

Interviewee:
I'll map out the order and show it to you from top to bottom.

Interviewer:
That's perfect! Please just also tell me the titles of these jobs like Vice President, Director and so on as these job titles are what my code reads out of the huge data set.

Interviewee:
Will do! Let's start at the CEO who is on top of the food chain and then we have executive vice presidents. So, they basically all have a specific type of pillar. So, there's a COO, who is also an Executive Vice President basically. He oversees operations. And then you have revenue management, network planning, sales. We also have marketing, IT, AeroPlan, which is basically our miles program, legal, HR, safety, maintenance and finance. Within all those branches we have senior vice president who all report to executive vice presidents. Below there are vice presidents. Hm, below there are MDs, so managing directors. We also have senior directors; we have directors and principles. The latter ones are basically senior managers. Below we also have partners or specialists. Basically, they have the same level, but specialists are more SMEs (subject matter experts) and partners are more of a business partner. Then we have product owners and product managers. They are both within the same band, which is like your hierarchical levels. Below that there're advisors, and last but not least analysts.

Interviewer:
So, advisors are above analysts?

Interviewee:
Yeah, that's how it is. Some departments, however, only have a VP and a director, some others have a VP, MDs, Senior Directors and so. It really depends by the branch. But in fact, that is how it from the CEO down to bottom.

Interviewer:
Actually, that's exactly what I need. It is way more titles that I have seen at any European airline, and I understand now your reorganization.

Interviewee:
[laughs]

Interviewer:
Below senior manager there's usually only one more level – at least for office staff. Also, how EVPs are such a common role! I've not seen that in Europe except for the CEO already of one airline in an airline group.

Interviewee:
Yeah, this is the whole picture. How as I said, they have cut a lot of titles. Now it's basically the VPs and the directors. We really had too many layers and middlemen. Some departments now don't have level 5 or 6, but that's okay.

Interviewer:
Good! Actually, that was already all the information I needed. As I had written to you, it should not take too long. I'm super thankful for your insights as this information is super hard to obtain.

Interviewee:
Absolutely, the org charts are usually not published. You see a lot of titles on LinkedIn, but it's hard to determine where someone is in the food chain. Anyways, feel free to contact me should you need any further information! I would try to gather it.

Interviewer:
Great, that's all from my side. If I have any further questions, I'll reach out to you! Thanks a lot!

Interviewee:
Alright, goodbye! Good luck!

Interviewer:
Goodbye, thanks!