# Task 1 Part I - III

## Data Profiling – Profile Data by doing

**By David J Kelley**

(dkelle24@wgu.edu)

### Introduction

In this assignment, we will review the data dictionary and the data provided for this assignment. This includes two initial files: a Word document [Employee Turnover Considerations and Dictionary.docx] and an Excel spreadsheet [Employee Turnover Dataset.xlsx], which I converted to CSV [Employee Turnover Dataset.csv] for programmatic analysis and processing in this assignment.

### Part I: Item A.1 Item A. Description

The request is to describe the general characteristics of the initial dataset (e.g., number of rows, columns). It is an Excel spreadsheet in \*.xlsx format, which is an XML-based format consisting of unnormalized data in 10,199 rows and 15 columns with labels. Scanning it quickly, I can see that there are floating-point values, as well as dollar values and what appear to be integers. Additionally, there is no PII data as such, but rather other data that would typically not be public.

### Part I: Item A.1 Item B. Data Types and Subtypes

The following request was "indicate the data type and data subtype for each variable." As suggested in the last item, the data set includes basically numeric values and what should be reference ID values, at least from the standpoint of wanting the data normalized in, say, Third Normal Form.

Among the numeric values are what appear to be integers, floating-point values, and dollar amounts, whereas the reference contains duplicated data. Of the numeric values, some are inconsistent in terms of being floats and integers in the same column. Some numeric values also have conflicting values that do not align with the rest of the data, such as N/A, which could represent nulls or other representations of empty values.

To be clear these “Identified Data Quality Issues” are somewhat problematic as it relates to extrapolating data types and sub types include:

* Mixed data types within a single column – For example, the HourlyRate field contains both integer and float values, and sometimes dollar signs (e.g., 25, 25.00, $25). This inconsistency complicates numeric analysis and requires standardization to a consistent numeric format without symbols.
* Inconsistent categorical values – Several categorical fields, such as PaycheckMethod, have variations in spelling or phrasing (e.g., "Mail Check" vs "Mailed Check"). This can cause inaccurate groupings or aggregations.
* Nulls represented inconsistently – Some missing values appear as N/A, others as blanks, and some as null. These should be standardized to a single null representation to ensure accurate filtering and calculations.
* Mismatched column names between source and data dictionary – The order and spelling of some columns in the extract differ from the data dictionary (e.g., "DrivingCummuterDistance" in the extract vs "Driving Commuter Distance" in the dictionary). This could cause mapping errors in ETL processes.
* Potential ID/reference misuse – Fields like EmployeeNumber are numerical but represent identifiers, not quantities, and should be treated as categorical keys rather than numeric measures.

I would extend the data dictionary like this in order as it appears in the source:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Column** | **File Index** | **DataDictionary Label** | **Index DD** | **Data Type (Implied)** | **Data Subtype** |
| EmployeeNumber | 1 | Employee Number | 1 | Integer/ probably an index | Identifier (Discrete, Nominal) |
| Age | 2 | Age | 2 | Integer | Numeric – Discrete |
| Tenure | 3 | Tenure | 3 | Integer | Numeric Discrete |
| Turnover | 4 | Turnover | 4 | String/Reference | Categorical – Nominal |
| HourlyRate | 5 | Hourly Rate | 6 | Float/Currency | Numeric – Continuous |
| HoursWeekly | 6 | Hours Weekly | 7 | Integer | Numeric – Discrete |
| CompensationType | 7 | Compensation Type | 5 | String/Reference | Categorical – Nominal |
| AnnualSalary | 8 | Annual Salary | 8 | Float | Numeric – Continuous |
| DrivingCummuterDistance | 9 | Driving Commuter Distance | 9 | Integer | Numeric – Discrete |
| JobRoleArea | 10 | Job Role | 10 | String/Reference | Categorical – Nominal |
| Gender | 11 | Gender | 11 | String/Reference | Categorical – Nominal |
| MaritalStatus | 12 | Marital Status | 12 | String/Reference | Categorical Nominal |
| NumCompaniesPreviouslyWorked | 13 | Number of Companies Worked | 13 | Integer or null? "n/a"? | Numeric - Discrete |
| AnnualProfessionalDevHrs | 14 | Annual Professional Development Hours | 14 | Integer or null? "n/a"? | Numeric – Discrete |
| PaycheckMethod | 15 | Paycheck Method | 15 | String/Reference | Categorical – Nominal |
| TextMessageOptIn | 16 | Text Message Opt-In | 16 | String or Boolean? /reference | Categorical – Nominal (Binary) |

Just by looking at the differences between the titles of columns in the extract and the data dictionary, we can see that both the order and spelling are different or not exact matches. Although it seems to be something we can deduce, validation would probably be needed to check if this is going to be an ongoing ETL process or something else. Also, data types seem somewhat forward-looking without too many inconsistencies, but might need to be mapped to some reference data if the underlying DBMS is normalized. Additionally, there are some inconsistencies in the data, such as the difference between "Mail Check" and "Mailed Check."

If I'm going to clean this data up, I would "normalize" categorical or reference data. Remove inconsistencies and remove formatting like dollar signs, and fix 'null' values like 'N/A'

### Part I: Item A.1 Item c. Sample Values

Next, we are asked to "provide a sample of observable values for each variable." Here is a chart of some interesting examples:

|  |  |  |
| --- | --- | --- |
| **Column(s)** | **Sample Values** | **Data Type (Implied)** |
| EmployeeNumber | 1 659 688 | Integer/ probably an index |
| Age | 59  36  21 | Integer |
| Tenure | 1  20  17 | Integer |
| Turnover | Yes  No | String/Reference |
| HourlyRate | $24.37  $87.22  $39.97 | Float/Currency |
| HoursWeekly | 40 | Integer |
| CompensationType | Salary | String/Reference |
| AnnualSalary | 50689.6  284641.6  287720 | Float |
| DrivingCummuterDistance | 89  0  -11 | Integer |
| JobRoleArea | Research  Information\_Technology  Human\_Resources  HumanResources | String/Reference |
| Gender | |  | | --- | | Female | | Male | | Prefer Not to Answer | | String/Reference |
| MaritalStatus | Single  Divorced  Married | String/Reference |
| NumCompaniesPreviouslyWorked | 3  6  N/A | Integer or null? "n/a"? |
| AnnualProfessionalDevHrs | 7  8  N/A | Integer or null? "n/a"? |
| PaycheckMethod | Mail Check Mailed Check  Direct\_Deposit DirectDeposit | String/Reference |
| TextMessageOptIn | Yes No N/A | String or Boolean? /reference |

Examining the same values reveals additional inconsistencies in the source data. For example, the Paycheck Method column. It would probably be best to aggregate the variances using a script, such as Python or SQL, to analyze or count the discrepancies in the data.

### Part II: Item B Item 1. Explain how examined

### My Python analysis script (D599Task1PartBnCScript.py) was designed to systematically examine the task CSV file for common data quality issues. The goal of this approach was to create a repeatable, automated method for identifying problems early, before the data is used for analysis, reporting, or machine learning. Instead of manually inspecting spreadsheets for errors, this script provides an overview of the dataset's health by analyzing five key dimensions: duplicates, missing values, categorical consistency, formatting issues, and numeric outliers. While I could write a script to resolve these issues, if I'm building an ETL process that will examine the same types of files, understanding data structure issues is more critical to address upfront.

### To begin the process, the script reads the CSV into a DataFrame and immediately cleans up the data. This includes trimming whitespace, standardizing representations of missing values (e.g., converting "N/A" and empty strings into proper nulls), and cleaning currency or number-like fields by removing symbols such as $ or commas so they can be safely converted into numeric types. This early normalization step ensures all later checks operate on clean, comparable values.

### Once the data is cleaned, the script performs several targeted inspections. It checks for both full-row duplicates and repeated values in the primary key column EmployeeNumber, which should uniquely identify each record. It counts missing values per column and compares the contents of categorical columns (such as Gender, Turnover, or PaycheckMethod) against predefined sets of expected values to detect typos or inconsistencies. It also flags formatting red flags, such as embedded currency symbols or trailing spaces, which are often overlooked but can cause parsing errors later. Finally, the script scans all numeric columns for outliers using the interquartile range (IQR) rule, which does not assume a normal distribution.

### As an example of issues detected during this inspection, the duplicate-check logic identified multiple EmployeeNumber values appearing more than once, confirming that primary key uniqueness was violated. In the categorical consistency check, the PaycheckMethod field contained inconsistent entries such as "Direct Deposit", "Mail\_Check", and "MailedCheck", showing that the dataset lacked a controlled vocabulary. These concrete findings illustrate that the inspection process not only automated the review but also surfaced tangible problems to be addressed in later ETL and cleaning stages.

### Part II: Item B Item 2. List my findings

Take a look at the output from my Python file [data\_quality\_report.txt]:

DATA QUALITY REPORT: Employee Turnover Dataset.csv

============================================================

DUPLICATES

Total duplicate rows : 99

Duplicate EmployeeNumbers : 99

First 5 duplicate rows index numbers:

10100, 10101, 10102, 10103, 10104

MISSING VALUES

NumCompaniesPreviouslyWorked : 665

AnnualProfessionalDevHrs : 1969

TextMessageOptIn : 2266

INCONSISTENT CATEGORICAL ENTRIES

PaycheckMethod : ['Direct Deposit', 'Mail\_Check', 'MailedCheck']

FORMATTING ISSUES

Embedded currency symbols in: ['HourlyRate ']

NUMERIC OUTLIERS (IQR rule)

AnnualSalary : 544 outliers

DrivingCommuterDistance : 245 outliers

Based on this inspecting of the dataset with a custom Python script, I identified several key data quality issues. First, I found 99 duplicate rows, all sharing the same EmployeeNumber values, such as indices 10100 through 10104. This violates the uniqueness required for a primary key and would cause referential integrity errors in a relational database. My ETL plan will include a de-duplication stage that either retains the first occurrence, merges duplicates when differences exist, or rejects and logs the extra records. I will also investigate the upstream system to determine why duplicate exports were possible and prevent recurrence.

### Second, there are substantial missing-value rates in three columns: NumCompaniesPreviouslyWorked is missing in 665 rows, AnnualProfessionalDevHrs in 1969 rows, and TextMessageOptIn in 2266 rows. These nulls reduce completeness and can cause failures if the data is used without defaults. The remediation plan is to first determine whether each field is optional or required by downstream systems. If optional, I will document the null percentages; if required, I will implement default values—such as setting TextMessageOptIn to “No” when blank—and log any remaining exceptions for review.

### Third, categorical data suffers from inconsistency. For example, PaycheckMethod contains variations such as “Direct Deposit,” “Mail\_Check,” and “MailedCheck.” Such differences will fragment categories in BI tools and cause joins to fail. I will create a normalization mapping during transformation so that all variants are converted to a canonical value (e.g., “Mail Check”) and will request that the source system enforce a controlled vocabulary to prevent future drift.

### Fourth, I identified formatting artifacts in numeric fields, most notably in HourlyRate, where embedded currency symbols (“$”) and occasional trailing spaces are present. This prevents direct numeric parsing. My resolution plan is to remove all non-numeric characters except the decimal point (for example, using a regular expression such as re.sub(r'[^0-9.]', '', value)), then convert the cleaned values to floats and round to two decimal places for consistency. I will also validate that resulting rates fall within a realistic range, such as $10.00 to $500.00 per hour, and flag outliers for review. Similar cleansing will be applied to other currency-related fields such as AnnualSalary, stripping symbols and commas before conversion. Finally, I will update extraction specifications to ensure that these fields are delivered as pure numeric values from the source whenever possible.

### Lastly, my outlier analysis, based on the IQR rule, revealed 544 AnnualSalary outliers and 245 DrivingCommuterDistance outliers. While these do not block loading, they can skew statistical models and may indicate entry errors such as extra zeros or incorrect units. My plan is to flag these records and decide—based on business rules—whether to cap, winsorize, or leave them untouched, while routing extreme anomalies to an “exceptions” table for manual inspection.

### By systematically removing duplicates, enforcing missing-value rules, normalizing categorical variables, applying robust numeric field cleaning, and flagging outliers, I will improve both the immediate usability and long-term reliability of the dataset. These steps will also feed back into upstream processes to prevent recurrence of the same issues in future extracts.

### Part II: Item C Item 1. How we modify the data based on Item B

After running the test script [D599Task1PartBnCScript.py] and creating a cleanup data script [D599Task1PartCETLScript.py], every problem that the quality scan surfaced is addressed in a concrete, rule-based way, allowing the data set to move safely into my theoretical ETL process.

**Duplicate primary‑key rows were eliminated first**

Whenever two or more records shared the same EmployeeNumber, the script kept the earliest instance and removed the rest. Each discarded row was written to "employee\_data\_issues.csv" with the note "Duplicate PK," ensuring I have a complete audit trail while restoring one‑to‑one integrity between the key and the employee.

**Numeric columns were sanitized, imputed, and "winsorised"**

Embedded dollar signs or commas were stripped out, values were coerced to floats, and any empty numeric cells were filled with the column median—our chosen, distribution‑agnostic default. The script then applied the IQR rule to cap extreme salaries or commuter‑distance figures at the upper or lower winsor limits. Outliers weren't discarded; they were brought back into a plausible range and flagged as "Outlier capped (winsor)" for later review.

**Categorical fields were harmonized and validated**

Known spelling variants (e.g., DirectDeposit, Mail\_Check, MailedCheck, or the space‑separated Direct Deposit) were converted to canonical tokens (Direct\_Deposit, Mail Check, Mailed Check). Any category value that still fell outside the approved "whitelist" was left unchanged but logged as "Unexpected category" so domain owners can decide whether to expand the vocabulary or correct upstream entry errors.

**Systematic filling of required text fields reduced sparsity**

Columns in my downstream model that might not tolerate empty values, such as TextMessageOptIn, NumCompaniesPreviouslyWorked, and AnnualProfessionalDevHrs, were given explicit defaults (No, 0, 0). Each autofill was recorded in the issues file, turning silent gaps into transparent, documented assumptions.

**Low‑level formatting flaws were removed silently.**

All object columns were trimmed of leading or trailing spaces as soon as the file loaded, nipping hidden mismatches ("Sales " vs "Sales") in the bud. No functional information was lost, so these trims were treated as routine hygiene and not individually logged.

The end result is two new artefacts:

1. employee\_data\_cleaned.csv – a clean dataset with unique keys, consistent categories, numeric fields free of currency symbols, no critical nulls, and extreme values safely bounded.
2. employee\_data\_issues.csv – a row‑by‑row ledger of every duplicate dropped, value imputed, category normalized, and outlier capped, giving me full traceability for governance or root‑cause follow‑ups.

By correcting issues instead of merely reporting them, the script transforms raw extracts into analysis-ready inputs while preserving a transparent record of every change—a best-practice balance between data quality, auditability, and ETL efficiency.

### Part II: Item C Item 2. Why did I do it this way?

The honest answer is that I know Python, and it's one of the choices available for the class, which is why I did it this way. Setting that reason aside and considering it from a theoretical point of view, I did it this way as I would normally do when building an ETL pipeline used repeatedly with large groups of CSV extract data. Generally, given a choice, I would probably use C# if the ETL were running in Azure, and Python, as I actually did here. After using the test script, I would use the later cleanup script repeatedly as needed via a lambda or other function. Say something that is watching an S3 bucket for the CSV file.

In line with this approach, automatically detecting, correcting, and logging issues before data enters an ETL pipeline, theoretically improving data quality, is valuable because it ensures reliability, consistency, and transparency across the entire data lifecycle. By cleaning and validating the dataset at the earliest stage, I prevent common errors from cascading into downstream systems, such as databases, analytics dashboards, and machine learning models. When issues such as duplicates, missing values, inconsistent categories, and outliers go unchecked, they can compromise join logic, distort metrics, and erode trust in reporting outputs. Addressing these problems upfront minimizes those risks and creates a clean foundation for all subsequent processes.

Unlike tools or reports that merely identify data issues, this method actively corrects them. Duplicates are removed, formatting artifacts like dollar signs or extra spaces are stripped, missing values are intelligently filled, and inconsistent categories are normalized to match the expected taxonomy. Numeric outliers are bounded using a robust IQR-based winsorization technique that reduces the influence of extreme values without discarding data. These corrective actions transform raw data into clean, consistent records that can be safely loaded into production systems and maintain referential integrity.

A critical advantage of this workflow approach is its inherent transparency. Every modification made to the data is logged in a separate file (which would likely be done in the database in a real version, where the log would have a trigger and other automated processes to notify someone). This log captures which row was affected, the issue, and how it was fixed. This audit trail supports data governance and accountability by showing a complete history of changes. Whether I am debugging an analytics discrepancy or answering a regulatory compliance question, I have concrete evidence of how the data was processed.

Additionally, the script's structure is highly adaptable. Business rules, category definitions, and thresholds are all configurable, so the system can evolve as requirements change. This makes the approach sustainable and scalable over time, rather than rigid or one-off. By automating these checks and corrections, organizations can handle incoming data more efficiently and consistently, reducing reliance on manual reviews and ad hoc cleaning scripts.

Ultimately, this approach helps build confidence in the data. Clean, trustworthy input reduces complexity in transformation logic, improves the stability of pipelines, and supports better decision-making at every level. It turns what is often a tedious and error-prone task into a repeatable, auditable, and high-impact part of my data engineering workflow.

### Part II: Item C Item 3 and 4. Anything else and downsides

In addition to the core benefits already discussed, several other advantages of this data quality approach enhance its utility in real-world data workflows. One significant advantage is that the script can be easily integrated into automated data pipelines or CI/CD systems. Because it's fully scripted and requires no manual interaction, it can serve as a quality gate for batch jobs, API ingestions, or scheduled workflows. This enables teams to consistently and immediately enforce data quality standards whenever new data arrives, without slowing down delivery.

Another advantage is that this approach accelerates the onboarding of new data sources. When new datasets are received from vendors, partners, or internal departments, they often vary significantly in format and quality. The script provides a centralized, reusable mechanism for validating and cleaning data against business rules, eliminating the need to build custom cleaning logic for each source. Moreover, by exporting an itemized issues log it supports efficient collaboration between data engineers and analysts. Instead of manually inspecting the whole dataset, teams can focus their attention on just the flagged records, speeding up root-cause analysis and resolution.

Despite its strengths, this approach has limitations. One key limitation is that the validation rules are static and domain-specific (let's talk about the 'taxonomy team at Microsoft to showcase problems like this). Category definitions, default fill values, and outlier thresholds must be manually configured. If business needs evolve. For example, new job titles or compensation types are introduced; the script requires updates to reflect these changes. It does not automatically learn or adapt, which can introduce maintenance overhead as data complexity increases.

Another limitation is how outliers are handled. While the use of interquartile range (IQR) capping is a reasonable general-purpose strategy, it may not always align with business intent. Some extreme values might be valid, such as unusually high salaries for executives or long tenures for founding employees. Automatically capping those values could distort meaningful signals in downstream analysis. Additionally, the script operates on a column-by-column basis and doesn't account for relationships across fields. It cannot catch issues such as salaries not matching the product of hourly rates and work hours, or inconsistencies between tenure and age.

Lastly, while the script logs all changes in a separate CSV, it doesn't support version control of datasets or track the downstream impact of changes on business metrics or models. In regulated or mission-critical environments, full reproducibility and lineage tracing may be required, which this lightweight approach does not offer out of the box.

In summary, this method provides an efficient and transparent solution to data quality challenges, especially in early-stage ETL or exploratory environments. However, it should be viewed as a foundational layer within a broader data quality framework, supplemented by schema validation, context-aware checks, and human oversight where necessary.

Now that said, given the current state of things, there are more robust ways of doing this sort of thing that overcome these limitations, such as cloud-based cognitive architecture that dynamically writes ETL processes and can modify them on the fly without human intervention. I've done a lot of research [Kelley] in this area and will reference my most recent paper below but basically given that if the ETL process is going to use a scripting language like Python then I could also use a JavaScript methodology in some kind of works flow engine that a large scale system could manipulate to do this kind of data cleaning on its own. Granted, this is still somewhat speculative, but I've seen it at work in limited cases. Lastly, I'll note that if performance is the bigger issue, I would write the process in C++, but that is very much an atypical need in most business cases.

### Part II: Item D Item 1 through 4

The data cleaning report, noted earlier as needed for items B and C, is also included as a text file, titled 'data\_quality\_report.txt'.

I created two Python files, one that generates the analysis report titled "D599Task1PartBnCScript.py" and the other titled "D599Task1PartCETLScript.py" both of which are annotated with comments.

There are three data set files included, if we count the txt file above, but also the cleaned data set titled: "employee\_data\_cleaned.csv", and the issues log titled "employee\_data\_issues.csv". Both of these are CSV files.

The Panopto view is included separately, but here is a link: <https://wgu.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=9a011f9b-903e-4ec0-ab90-b309005423d0>

### References

Kelley, D.; "Problem-Solving and Learning Strategies within the Independent Core Observer Model (ICOM) Cognitive Architecture;" BICA 2024 / AGI Conference 2024; DOI: 10.13140/RG.2.2.17125.82406; Springer Nature BICA Proceedings Dec 2024; <https://www.researchgate.net/publication/381375310_Problem-Solving_and_Learning_Strategies_within_the_Independent_Core_Observer_Model_ICOM_Cognitive_Architecture> ; Proceedings Release Version: <https://link.springer.com/book/10.1007/978-3-031-76516-2?sap-outbound-id=0A34C0BAE61B956A4CA1DD5C48793788C0964CD7>

No other resources where used.