ACC 424 Accounting Information System

Auditing data sets with prompt engineering
Python notes for 5.1-5.4

Section 001 MW 9:30 AM – 10:45 AM at Rm. 257 over Jan 13 – May 07
Section 002 MW 11:00 AM – 12:15 AM at Rm. 127 over Jan 13 – May 07

5.1. Auditing data sets

Data auditing is the process of **examining** and **verifying** data to ensure it is **accurate**, **complete**, and **consistent**. Think of it like **proofreading** a document, but for data. Auditing data sets helps find and fix **errors**, like missing numbers, duplicates, or incorrect values that don't make sense.

Why are Auditing Data Sets important?

1. **Reliability**: Ensures the data should well reflect real-world values. For example, if a company's sales data says they made 1,000,000 in a month, but the real number is 1,000,000 *in a month, but the real number is*100,000, that's a big problem!
2. **Compliance**: Many industries (like accounting and healthcare) have strict rules about data. Auditing helps ensure the data **follows** these rules.
3. **Help strategic decision-making**: Businesses rely on **data** to make strategic decisions. If the data is wrong, the decisions will be wrong too.
4. **Fraud Detection**: Auditing can help find **suspicious** activity, like fake transactions or unusual patterns.

Data Quality Dimensions

When auditing data, we check for the following **key data qualities** to ensure the data is reliable and useful:

1. **Correctness**: Is the data correct? Incorrect data can lead to wrong decisions. For example, if a sales transaction is recorded as 1,000 due to a typo instead of 1,00 (actual value), it could mislead managers about firm performance.
2. **Completeness**: There is no **missing** data. **All** required fields should have values.

3. **Consistency**: The data is **uniform** across different files, systems, or time periods.
4. **Timeliness**: The data is **up-to-date** and available when needed. Outdated data can lead to poor decisions. For example, if last month's sales data is missing, you can't accurately analyze trends.
5. **Validity**: The data should follow **predefined rules** or formats. Invalid data can cause errors in data processing. For example, if account numbers must be 6 digits, a 5-digit number is **invalid**.

Common Data Problems

- **Duplicates**: The same data appears more than **once** in a dataset. Duplicates can skew analysis and may lead to incorrect conclusions. For example, if a sales transaction is recorded twice, it might look like sales are higher than they really are.
- **Missing Values**: Missing values occur when data is **incomplete** (e.g., blank cells or fields). Missing data can make analysis inaccurate. For example, if customer names are missing, you can't identify who made a purchase.
- **Outliers**: Numbers that are **too high** or **too low** compared to the remaining data. Outliers can **distort** analysis and indicate errors or fraud. For example, a $1,000,000 sale in a small business might be a mistake or a fraudulent transaction.
- **Formatting Errors**: Formatting errors occur when data is in the **wrong format**. Formatting errors can prevent data from being processed correctly. For example, dates should be written as numbers in Date 1 format (see below). And if dates are written as text in Date 2 format (see below), you can't sort them chronologically.

  Date 1: 2023-03-01 (correct format)

Date 2: March 1, 2023 (text format)

5.2. Prompt Engineering

Prompt engineering is the process of writing **clear** and **specific** instructions (called **prompts**) for Generative AI tools (like ChatGPT, DeepSeek, Claude) to get useful answers. Think of it like giving directions to a **robot**. AI tools don't "understand" things the way humans do. They rely on the words you use to generate responses. If your prompt is vague, the AI's answer might be vague too. So, the better your instructions, the better the robot's work.

How does prompt engineering work?

1. You write a **prompt**: A prompt is a question or instruction telling the AI what to do.
2. The AI reads the prompt and generates a response: The AI uses a **large language model** (like GPT, DeepSeek, or Claude) to analyze your prompt. The model is trained on a **gigantic** amount of text data, so it understands **language patterns**. The model looks for patterns in its **training** data that match your prompt and generates **responses** based on that knowledge.
3. You use the response to solve a problem or get insights. If the response isn't what you wanted, **tweak** your prompt and try again.

Tips for writing effective prompts

1. **Be Specific**: Clearly state what you want the AI to do.

   - Example: Instead of “Tell me about sales,” say “Summarize the total sales for each product category in Q1 2023.”

2. **Provide Context**: Give the AI background information to help it understand your request. More details to be given is always fine.
   - Example: “This is a dataset of monthly sales. Analyze the trends and identify the top-selling products.”
3. **Use Clear Language**: Avoid ambiguous terms unless you define them clearly.
   - Example: Instead of “Find the anomalies,” say “Find all transactions where the amount is more than $10,000.”

Why use DeepSeek for Prompt Engineering?

While there are many Generative AI tools available (like ChatGPT and Claude), DeepSeek stands out for several reasons:

1. **Open Source**: DeepSeek code is publicly available. The code is hosted on platforms like **GitHub**, where anyone can access it. Our users can see how the software works.
2. **Free to Use**: DeepSeek is **free**, making it accessible to individuals, students, and organizations with limited budgets (**key advantage** for our students). Unlike some AI tools that require monthly subscriptions, DeepSeek provides powerful AI capabilities at **no cost**.
3. **Customizability**: This allows developers and users to **customize** and **improve** the tool to **suit** their specific needs. DeepSeek allows users to **modify** and **fine-tune** the model for specific tasks, such as auditing data or data analysis.

5.3. Hands-on auditing data sets with Python and prompt engineering

https://anaconda.cloud/share/notebooks/bfbbbb2d-00f9-45c7-825c-fcaf1357f0ad/overview

Please use the above weblink to download the Python coding of **Auditing data sets with prompt engineering.**

We'll dive into auditing data sets using **prompt engineering** to write **Python code**. We'll create and import a larger dataset, define **predefined rules** for auditing this dataset, and use prompt engineering to write Python that **automates** the auditing process.

Step 1: Importing the Dataset

We'll start by importing a sample dataset into Jupyter Notebook.

```
import pandas as pd
```

This line code imports the **Pandas** library and gives it the alias pd. We will use this library to convert data into a Pandas **DataFrame**, which is a **table-like** structure used for data analysis.

```
data = {
    "Transaction ID": [1001, 1002, 1003, 1004, 1005, 1006, 1007, 1008, 1009, 1010, 1011, 1012, 1
013, 1014, 1015, 1016, 1017, 1018, 1019, 1020],
    "Date": ["2023-01-01", "2023-01-02", "2023-01-03", "2023-01-04", "2023-01-05", "2023-01-06",
    "2023-01-07", "2023-01-08", "2023-01-09", "2023-01-10", "2023-01-11", "2023-01-12", "2023-01
-13",
    "2023-01-14", "2023-01-15", "2023-01-16", "2023-01-17", "2023-01-18", "2023-01-31", "2023-01
-20"],
    "Amount": [3744, 9508, 7323, 5981, 1568, 100000, 1568, 6776, 5612, 1555, 6755, 8430, 1098, 7
263, 1234, 1001, 9876, 5432, 2023, 8765],
    "Customer Name": ["Alice", "Bob", "Charlie", "Diana", "Eve", "Alice", "Bob", "Charlie", "Dia
na", "Eve", None, "Alice", "Bob", "Charlie", "Diana", "Eve", "Alice", "Bob", "Charlie", "Dian
a"],
    "Region": ["West", "North", "South", "East", "West", "North", "South", "East", "West", "Nort
h", "South", "East", "West", "North", "South", "East"].
}
```

We then import a sample dataset of financial transactions. We create a **dictionary** called data containing the sample dataset. Each **key** in the dictionary (e.g., "Transaction ID", "Date") represents a **column name**, and the corresponding value is a **list** of data for that column.

- "Transaction ID": A list of unique IDs for each transaction.
- "Date": A list of dates for each transaction.
- "Amount": A list of transaction amounts.
- "Customer Name": A list of customer names.
- "Region": A list of regions where the transactions occurred.

```
df = pd.DataFrame(data)
```

This line code converts the data dictionary into a **Pandas DataFrame** called df. A DataFrame is a 2-dimensional **table** with rows and columns, like an Excel spreadsheet.

```
df
```

This line displays the **DataFrame** df in the notebook. When you run this code in a Jupyter Notebook, it will show the dataset as a table.

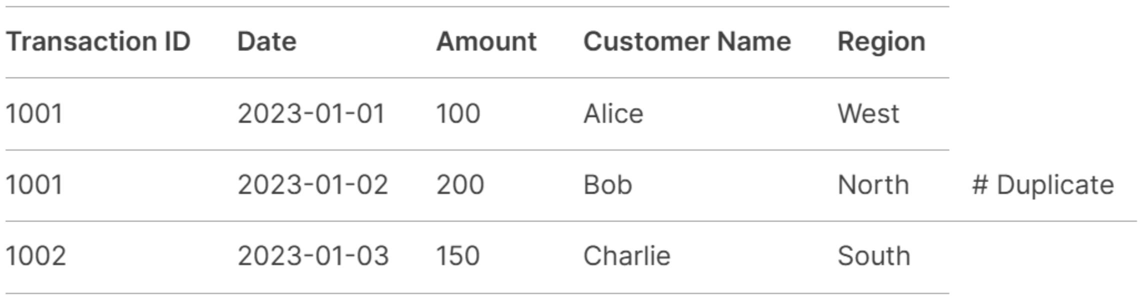
Step 2: Defining predefined rules for auditing datasets

Before auditing datasets, it's essential to establish **predefined rules** to ensure the data meets **quality standards**. These rules act as guidelines to identify and resolve data quality issues.

1. No **Missing** Values: All required **fields** (e.g., Customer Name; Region) must have **values**. Missing values can lead to incorrect conclusions. For example, if the Customer Name is missing, you can't identify who made a purchase.
   - **Valid**: Customer Name = "Alice"
   - **Invalid**: Customer Name = None (missing value)

2. No **<u>Duplicates</u>**: Each Transaction ID must be unique. Each transaction should have a **<u>unique identifier</u>** (Transaction ID) to distinguish it from all other transactions. For example, if a transaction is recorded twice, it might look like sales are higher than they really are.

   - Valid: Transaction ID 1001 refers to a specific sale of $100 on January 1, 2023.
   - Invalid: Transaction ID 1001 refers to two **<u>different</u>** sales (e.g., 100 and 200).



Transaction ID	Date	Amount	Customer Name	Region	
1001	2023-01-01	100	Alice	West	
1001	2023-01-02	200	Bob	North	# Duplicate
1002	2023-01-03	150	Charlie	South	

3. No **<u>Outliers</u>**: Outliers are values that are too high or too low compared to the rest of the data. We set the rule that the amounts should be between 100 and 10,000, otherwise they are outliers.

   - Valid: Amount = 5000 (within the range)
   - Invalid: Amount = 100000 (**<u>outlier</u>**)

4. Correct Date Format: Dates must be in YYYY-MM-DD format. This is a standardized format that ensures consistency and makes it easier to sort and analyze dates. Incorrect date formats can prevent data from being processed. For example, if dates are written as **<u>text</u>**, you can't sort them **<u>chronologically</u>**.

   - Valid: Date = "2023-01-01" (correct format)

- Invalid: Date = "January 1, 2023" (incorrect format)

Step 3: Using prompts to generate Python code

If you don’t know how to write Python code or find it time-consuming to write on your own, you can use **prompt engineering** to generate the necessary code for auditing tasks. Prompt engineering involves writing clear and specific instructions (called **<u>prompts</u>**) for Generative AI tools (like ChatGPT, DeepSeek, or Claude) to produce the desired coding.

Before writing a prompt, we need to clearly identify the **<u>auditing task</u>** we want to perform.

- Finding missing values in the ‘Customer Name’ column.
- Identifying duplicate Transaction IDs.
- Detecting outliers in the Amount column.
- Validating date formats.

We should craft a prompt that clearly explains the task to AI. The more **<u>SPECIFIC</u>** your prompt, the better the generated code will be. Submit your prompt to the AI tool, and it will generate the corresponding Python code. Here’s an example of how this works:

Generating code for finding missing values

- **Prompt**: “Write Python code to find all rows in the dataset where the ‘Customer Name’ column is missing.”
- **AI-Generated Code**:

```python
# Find missing values in the 'Customer Name' column
missing_values = df[df["Customer Name"].isnull()]
print("Rows with missing 'Customer Name':")
print(missing_values)
```

Generating code for identifying duplicates in Transaction ID

- **Prompt**: "Write Python code to find all duplicate entries in the 'Transaction ID' column."
- **AI-Generated Code**:

```python
# Find duplicate Transaction IDs
duplicates = df[df.duplicated("Transaction ID", keep=False)]
print("Duplicate Transactions:")
print(duplicates)
```

Generating code for detecting outliers in the Amount column

- **Prompt**: "Write Python code to find all transactions where the amount is less than 100 **or** greater than 10,000."
- **AI-Generated Code**:

```python
# Find outliers in the 'Amount' column
outliers = df[(df["Amount"] < 100) | (df["Amount"] > 10000)]
print("Outliers:")
print(outliers)
```

Generating Code for validating Date formats

- **Prompt**: "Write Python code to check if all values in the 'Date' column are in the correct format (YYYY-MM-DD). If not, list the invalid entries."
- **AI-Generated Code**:

```python
# Check for incorrect date formats
incorrect_dates = df[~pd.to_datetime(df["Date"], errors="coerce").notna()]
print("Rows with incorrect date formats:")
print(incorrect_dates)
```

Once the AI generates the Python code, you can copy and paste it into your Python environment (e.g., Jupyter Notebook) and run it to perform the auditing tasks.

5.4. Debugging AI-generated Python code

Even though AI tools like ChatGPT or DeepSeek can generate Python code, the code might not always run perfectly on the first try. When you run the code and it fails, Python will usually display an **error message**. This message is your first clue to understanding what went wrong.

Common Error Types:

1. **SyntaxError**: The code has a typo or is written incorrectly (e.g., missing a colon or parenthesis).
   - Example: print("Hello world" (missing closing parenthesis).
2. **NameError**: The code uses a variable or function that hasn't been defined.
   - Example: print(x) (if x is not defined).
3. **TypeError**: The code tries to perform an operation on incompatible data types.
   - Example: "5" + 5 (trying to add a string and a number).
4. **KeyError**: The code tries to access a column or key that doesn't exist in the dataset.
   - Example: df["CustomerName"] (if the column is named Customer Name instead of CustomerName).

If the error message isn't clear, check the code for these common issues:

Missing Imports

- AI-generated code might assume certain libraries (e.g., pandas) are already imported. If you see an error like NameError: name 'pd' is not defined, you need to **import** the library.
- **Fix**:

```
import pandas as pd
```

Incorrect Column Names

- AI-generated code might use column names that don't match your dataset. For example, if your dataset has a column named Customer Name but the code uses CustomerName, it will throw a **KeyError**.
- **Fix**: Double-check the column names in your dataset and update the code accordingly.

```
# Incorrect
df["CustomerName"]
# Correct
df["Customer Name"]
```

Data Type Issues

- AI-generated code might assume the data is in a specific format (e.g., numeric or date). If the data is in the wrong format, the code will fail.
- **Fix**: Check the **data types** of your columns using df.dtypes and convert them if necessary.

```
# Convert 'Date' column to datetime
df["Date"] = pd.to_datetime(df["Date"], errors="coerce")
```

Missing Data

- If your dataset has **missing** values, some operations (e.g., mathematical calculations) might fail.

- **Fix**: Handle missing values before running the code.

Finally, if you still can’t fix the code, refine your prompt and ask the AI to <u>**regenerate**</u> it. Be more <u>**SPECIFIC**</u> about the issue you’re facing.