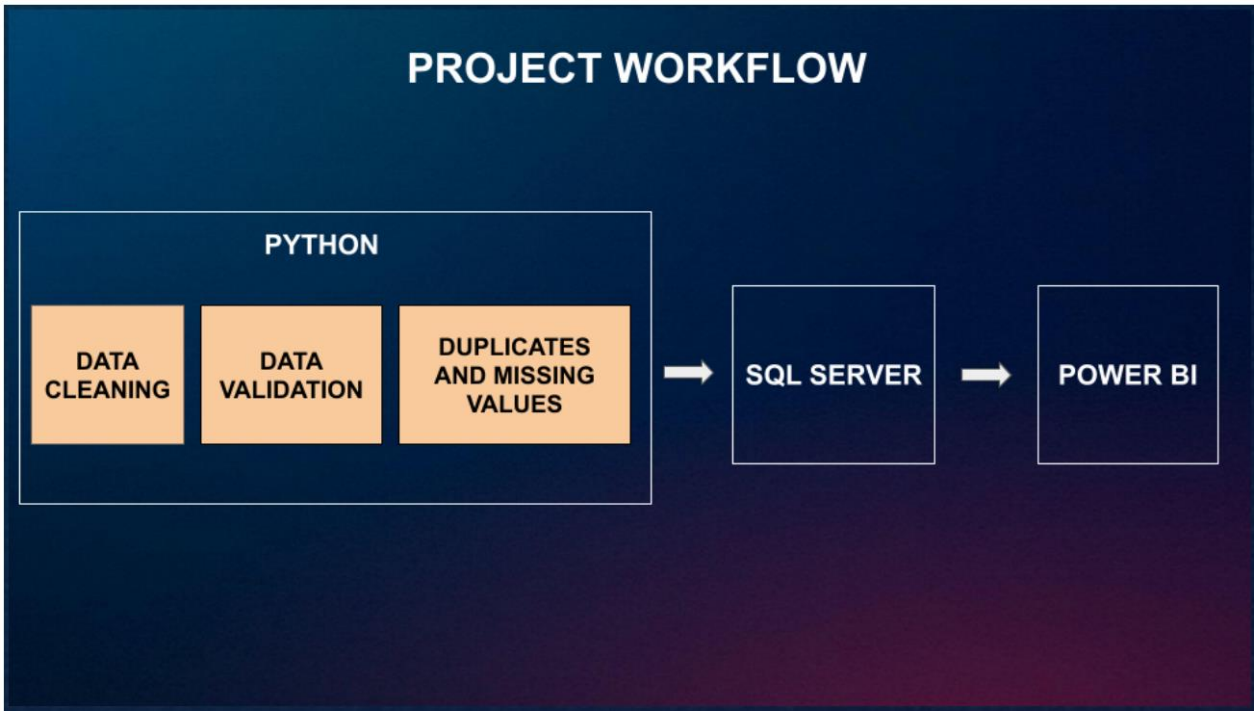
Amazon Product Analysis

Barathan Rangarajan
Ashpak Sheikh
Kousik Dutta
Raghavendra Betageri

Introduction:

The Amazon Product Analysis project aimed to analyze and gain insights from a dataset of Amazon product details. The objective was to extract meaningful information from the data and provide valuable insights to aid decision-making, product improvements, and marketing strategies. The project involved several stages, including data collection, data cleaning and preprocessing, exploratory data analysis, and visualizations.

Project Workflow:



1. Data Cleaning and Validation Workflow:

a. Import the necessary Python libraries for data cleaning and validation.
b. Load the raw data into a pandas DataFrame.

c. Checking Data Consistency.
d. Perform data validation.
- Address any validation errors by applying corrective actions
e. Handle duplicates:
- Decide on an approach to handle duplicates (e.g., removal, consolidation)
f. Missing Value Analysis:
- Decide on a strategy to handle missing values (e.g. imputation, deletion).

2. Storing Cleaned Data in SQL Server Workflow:

a. Connect to the SQL Server database using Python libraries like `pyodbc`
b. Create a new table or identify an existing table to store the cleaned data.
c. Importing the data from local with the help of Bulk Insert method to SQL Server.
d. Execute SQL queries as per the problem statements.

3. Visualizing Cleaned Data using Power BI Workflow:

a. Launch Power BI and establish a connection to the SQL Server database containing the cleaned data.
b. Import the required tables/views from the database into Power BI.
c. Design and create visualizations (charts, graphs, dashboards) based on the data requirements and objectives.
d. Apply data transformations or calculations within Power BI to enhance the visualizations.
e. Customize the visualizations with formatting, colors, labels, and interactions.

Libraries:

The code begins by importing necessary libraries for data manipulation, web scraping, and data validation:

- To find the matching patterns in string we have imported **re** module -
  import re
- To access the data source in the local file system we have used **OS** module -
  import os
- For data cleaning, transformation and analysis we have used **NumPy** and **pandas** module -
  import numpy as np
  import pandas as pd

Path Variables:

The code defines the folder paths and file paths to read and write the data. Update the folder path, cleaned folder, and path variables to match your system's folder structure.

- Project Folder Path -

```
project_path = r"C:\\Users\\Futurense\\Desktop\\Team Project - Amazon"
```

- Let's define folder path
```
folder_path = f"{project_path}\\data"
```

- Clean File Name
```
clean_file = "amazon_csv_cleaned.csv"
```

- driver = "ODBC Driver 18 for SQL Server"
```
driver = "SQL Server"
server = "DESKTOP-TOEPTEF\SQL_SERVER"
port = 1433
```

- Database Name
```
db_name = "AmazonDB"
```

- Table name
```
table_name='amazon_product_v1'
```

Files Check Inconsistency:

The code checks if the "Amazon-Products.csv" file contains the full data of all other files in the folder. It compares the content of each file with the "Amazon-Products.csv" file and identifies any differences. If any differences are found, it displays the file name and the indices of the differing rows.

We concatenated all the rows to a single column and used it to compare with other data frames, instead of checking every element of the row.

The code we have used in our project is -

```
def create_concat_col(df: pd.DataFrame) -> pd.DataFrame:
    """
    Concat all rows to new column and sort them
    """
    df["sorter"] = df.apply(lambda x: concat_series(x), axis=1)
    df = df.sort_values("sorter").reset_index(drop=True)
    return df

my_df["is_match"] = my_df["sorter"] == DF["sorter"]
DF["is_match"] = my_df["sorter"] == DF["sorter"]
print(f"The No of non-matcing : {len(my_df[~my_df['is_match']])}")
```

Read Data:

The code reads the data from the "Amazon-Products.csv" file into a pandas Data Frame named df.

```
df = pd.read_csv(f"{folder_path}\Amazon-Products.csv", index_col=0)
```

Data Consistency:

The code performs consistency checks on the data, specifically for numeric and categorical columns.

- **Numeric Data Inconsistency:**

  The code defines a function format_numbers to format numeric strings by removing commas and currency symbols. It then applies this function to numeric columns (discount_price, actual_price, no_of_ratings, ratings) in the DataFrame.

```
def format_numbers(string: str, is_price: bool=False ) -> str:
    '''Replacing ',' and '₹' symbols to a proper numberic string.'''
    if isinstance(string, float) and np.isnan(string):
        return string

    res = string.replace(",", "")
    if is_price:
        res = res.replace("₹", "")
    return res


def covert_numeric_cols(df: pd.DataFrame, cols: list) -> None:
    ''' will convert string to number and overwrite the DataFrame '''
    for col in cols:
        is_price = False
        if 'price' in col:
            is_price = True
        df[col] = pd.to_numeric(df[col].apply(lambda x: format_numbers(x, is_price)),
'coerce')
```

- **Categorical Data Inconsistency:**

  The code defines a function cat_word_consistency to standardize text by capitalizing words and replacing specific patterns. It applies this function to categorical columns (sub_category, main_category) in the DataFrame.

```
def cat_word_consistency(x: str)->str:
  """
  Text Standardization.
  """
  x=x.title().replace("Tv", "TV").replace("&", "And").replace("'S", "'s")
  return x
```

Data Validation:

The code performs data validation checks on the DataFrame, including numeric data validation and URL validation.

- **Numeric Data Validation:**

  The code defines a function numeric_data_validation to limit numeric values to a specific range (0-5) and a function negative_value_validation to convert negative values to zero. It applies these functions to the ratings and no_of_ratings columns, respectively.

```
df["is_valid_no_of_ratings"] = (df["no_of_ratings"] >= 0)
df["is_rating_valid"] = (df["ratings"].between(1, 5) | ( (df["ratings"] == 0) ) & (
(df["no_of_ratings"] == 0) ) )
```

- **Price Dependent Validation:**

The code checks for price dependency issues by validating that the actual_price is greater than 0, discount_price is greater than or equal to 0, and actual_price is greater than discount_price. It creates a new column is_price_dependency_valid to store the validation result.

```
df["is_price_dependency_valid"] = (df["actual_price"] > 0) \
& (df["discount_price"] >= 0) \
& (df["actual_price"] > df["discount_price"])
```

Duplicates:

The code removes duplicate rows from the DataFrame using the drop_duplicates method.

Example: -
We have modified the link by removing the variable part after 'ref', as it does not affect the functionality of the link. The updated link is: https://www.amazon.in/Lloyd-Inverter-Convertible-Anti-Viral-GLS18I3FWAMC/dp/B0BRKXTSBT. Although the 'ref' portion may make the link appear unique, it does not impact its usability or purpose.

Codes used -

```
col = ["name", "main_category", "sub_category", "link", "no_of_ratings",
"ratings", "actual_price", "discount_price"]
df = df.drop_duplicates(subset=col)

col = ["name", "link", "no_of_ratings", "ratings", "actual_price",
"discount_price"]
df = df.drop_duplicates(subset = col)
```

Missing Values:

The code checks for missing values in the DataFrame and provides information about the columns with missing values. There can be products in amazon where no ratings have been given to it yet so it is reasonable to fill the null values with zero.

```
df[["no_of_ratings", "ratings"]] = df[["no_of_ratings", "ratings"]].fillna(0)
```

Price Missing:

The code checks for missing values in the discount_price columns. If any missing values are found, it replaces them with 0. And for actual_price we are dropping the data

```
df.loc[:, "discount_price"] = df["discount_price"].fillna(0)
df = df[~df["actual_price"].isna() & (df["actual_price"] > 0)]
```

Save Cleaned Data:

The code saves the cleaned DataFrame to a new CSV file named "Cleaned-Amazon-Products.csv" in the specified cleaned_folder path.

```
df.to_csv(f"{project_path}\\{clean_file}", index=None)
```

Python SQL Connector:

The exported file is used to load the data from Python to SQL server using pyodbc.

Power BI:

We connect the SQL data to Power BI and used PowerBI to analyze and visualize data to gain insights.

Note: - The Power BI visualization is shared along with this document

Summary:

The code provides a summary of the cleaning process, displaying the total number of rows, the number of rows before cleaning, and the number of rows after cleaning.

This code aims to clean and validate Amazon product data stored in CSV files. It performs consistency checks, data validation, removes duplicates, handles missing values, and to extract additional product information. The final cleaned data is saved to a new CSV file and then use pyodbc to load data to SQL server.