**SQLite Database Connection, Data Preprocessing, and Validation**

1. **SQLite Database Connection**

A connection was established to the AdventureWorks SQLite database using the sqlite3 library in Python. This allowed for querying relevant data and creating DataFrames for each table. The connection was the same for all the data transformations performed on the various CSV files.

**Database Path**:

The database file path for the connection was as follows:

C:\Users\Oluwafunmilayo Basil\Documents\EBUNOLUWA\3signet\AdventureWorks.db

**Connection Code**:

python

import sqlite3

conn = sqlite3.connect(db\_path)

**Confirmation:**

A message was printed to confirm the database connection for each file processed:

python

print(f"Connected to the database at: {db\_path}")

2. **Data Extraction and Loading of CSV Files**

Data was extracted from the SQLite database for different tables and stored as CSV files for the following datasets:

AdventureWorks\_Calendar.csv

AdventureWorks\_Customers.csv

AdventureWorks\_Product\_Categories.csv

AdventureWorks\_Products.csv

AdventureWorks\_Product\_Subcategories.csv

AdventureWorks\_Returns.csv

AdventureWorks\_Sales\_2015-2017.csv

AdventureWorks\_Territories.csv

The query for each table extracted all available data and converted it into a Pandas DataFrame.

SQL Query Example:

sql

SELECT FROM <Table\_Name>

**Loading Data into Pandas**:

The extracted data was loaded into a DataFrame for processing:

python

df = pd.read\_sql\_query(query, conn)

3. **Data Preprocessing**

The same data preprocessing steps were applied across all CSV files to ensure consistency and data integrity.

**Step 1**: Handling Missing Values

Each DataFrame was inspected for missing values to determine where gaps existed in the data.

Missing Value Inspection:

python

missing\_values = df.isnull().sum()

print(missing\_values)

Missing values were handled using the forward fill method (ffill()), which ensured that any gaps in the dataset were filled based on prior valid data.

Handling Missing Data:

python

df\_cleaned = df.ffill()

**Step 2**: Removing Duplicates

The data was checked for duplicate rows to avoid redundancy, ensuring that all records were unique across the different datasets.

Duplicate Removal:

python

duplicates = df\_cleaned.duplicated().sum()

df\_cleaned = df\_cleaned.drop\_duplicates()

print(f"Number of duplicate rows removed: {duplicates}")

**Step 3**: Data Type Conversion

The columns containing date information were converted to datetime format. This allowed for accurate timebased analysis and avoided potential errors due to incompatible data types.

Date Conversion:

python

df\_cleaned['Date'] = pd.to\_datetime(df\_cleaned['Date'], errors='coerce')

**Step 4**: Data Normalization

Normalization was applied to numerical columns, where necessary, to standardize the range of values. This ensured that all numerical features had comparable scales for potential future analysis.

Normalization Using MinMaxScaler:

python

from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()

df\_normalized = pd.DataFrame(scaler.fit\_transform(df\_cleaned), columns=df\_cleaned.columns)

4. **Data Saving**

After preprocessing and validation, the cleaned data was saved back into CSV format for each file. This enabled further analysis or integration with other systems.

CSV Export:

The cleaned and transformed DataFrame was saved for each dataset:

python

output\_csv\_path = r'C:\path\to\cleaned\_file.csv'

df\_cleaned.to\_csv(output\_csv\_path, index=False)

Output Files:

The cleaned data was saved as:

Calender.csv

cleaned\_customer\_data.csv

cleaned\_product\_category\_data.csv

cleaned\_product\_data.csv

cleaned\_product\_subcategory\_data.csv

cleaned\_sales\_territory\_data.csv

cleaned\_sales\_20152017.csv

The above files were afterwards imported to excel as a single file, with each table in different sheets.

python

import pandas as pd

csv\_files = {

'Calender.csv': 'Calendar',

'cleaned\_customer\_data.csv': 'Customer Data',

'cleaned\_product\_category\_data.csv': 'Product Category',

'cleaned\_product\_data.csv': 'Product Data',

'cleaned\_product\_subcategory\_data.csv': 'Product Subcategory',

'cleaned\_sales\_territory\_data.csv': 'Sales Territory'

}

# Output Excel file path

excel\_file\_path = r'C:\Users\Oluwafunmilayo Basil\Documents\EBUNOLUWA\3signet\combined\_data.xlsx'

# Create a new Excel writer object

with pd.ExcelWriter(excel\_file\_path, engine='xlsxwriter') as writer:

# Loop through the CSV files and write each to a separate sheet

for csv\_file, sheet\_name in csv\_files.items():

# Read the CSV file into a DataFrame

df = pd.read\_csv(csv\_file)

# Write the DataFrame to an Excel sheet

df.to\_excel(writer, sheet\_name=sheet\_name, index=False)

# The Excel file with multiple sheets has been created at the specified location

print(f"Excel file saved at: {excel\_file\_path}")

5. **Data Validation Documentation**

The following processes ensured the validity of the data across all files:

Missing Data: Missing values were handled using forward fill, which helped maintain continuity in the dataset without introducing new biases.

Duplicate Removal: Duplicate rows were identified and removed across all datasets, ensuring data quality.

Date Conversion: All date columns were successfully converted into datetime format to facilitate future temporal analysis.

Normalization: Numerical columns were normalized using the MinMaxScaler, ensuring consistent data scaling across features.

**Conclusion**

This report serves as a comprehensive guide to the data preprocessing and validation steps performed for all listed CSV files, outlining the entire process from database connection to data transformation, ensuring clarity and reproducibility for future work.