Data Pre-processing and Cleaning Using Python

Team: Yisi Lu,
Kenny Lei, Alex Fan

Data Collection

Why we chose the data:

About the Data:

- Sales Transactions from a Cafe
  - Rows: 10,000
  - Columns: 8

Missing / Error Values

- Contains Missing or Invalid Values
  - Errors, Unknowns, etc

Duplicates

- Contains Duplicate rows

Inconsistent Data

- Contains inconsistencies in Data Types
- Contains Inconsistent Date Formats

8 Data Cleaning Steps:

1. Check and Remove Duplicate Rows
   - Data consisted of Duplicate Rows leading to inaccurate results if left alone
2. Convert Data Types
   - All columns were stored as object strings which made it impossible to perform calculations
3. Fill Missing Numeric + Categorical Values with Median and Mode
   - Data had missing values that needed to be filled. Numeric filled with Median, Categorical filled with Mode
4. Strip and Standardize Text Values
   - Consisted of inconsistent text formats such as capitalization and extra white spaces
5. Convert Transaction Date to DateTime
   - Transaction Date had inconsistent formats. Converted all to DateTime Format
6. Drop Rows with Invalid Dates
   - Rows contained invalid dates and needed to be deleted
7. Reset Index
   - Need to Reset the Index due to dropped rows
8. Recalculate Total Spent
   - For some rows, total spent was not calculated correctly. We corrected these values

Final Dataset Summary

Before: 10,000 rows
8 columns
Improper data types

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 8 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   Transaction ID    10000 non-null  object
 1   Item              9667 non-null   object
 2   Quantity          9862 non-null   object
 3   Price Per Unit    9821 non-null   object
 4   Total Spent       9827 non-null   object
 5   Payment Method    7421 non-null   object
 6   Location          6735 non-null   object
 7   Transaction Date  9841 non-null   object
dtypes: object(8)
memory usage: 625.1+ KB
Original dataset information: None
```

After: 9540 rows
8 columns
Suitable data types

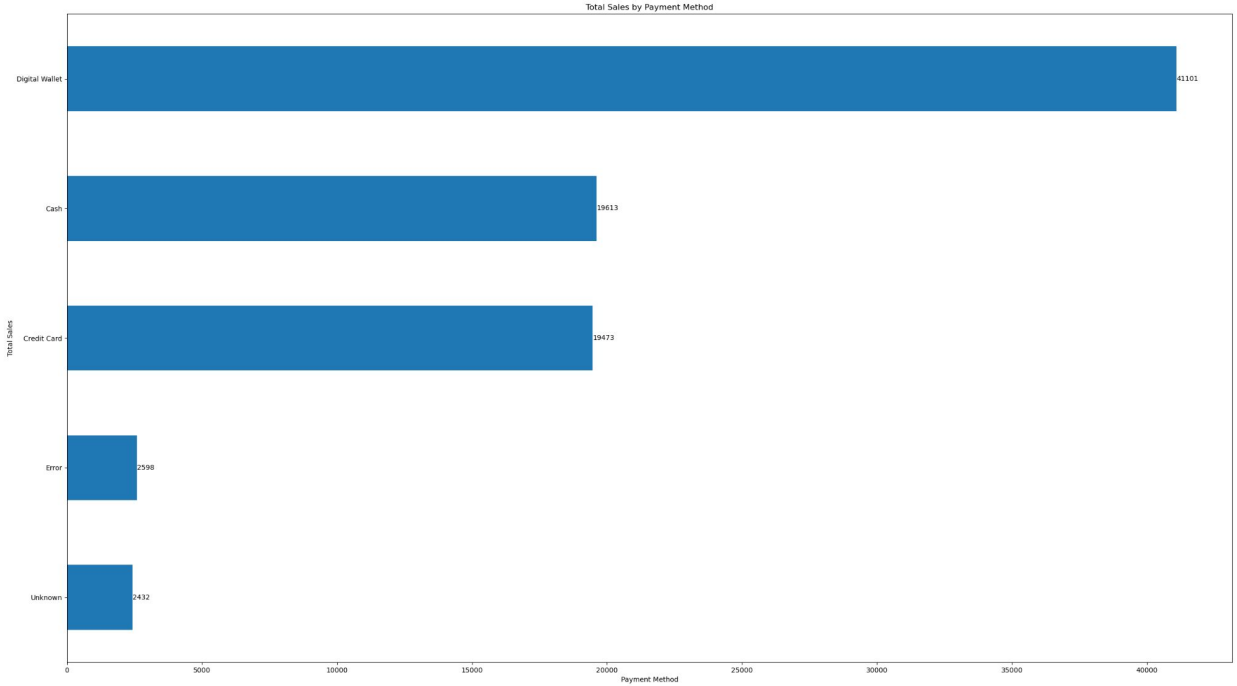
```
Cleaned dataset information: None
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9540 entries, 0 to 9539
Data columns (total 8 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   Transaction ID    9540 non-null   object
 1   Item              9540 non-null   object
 2   Quantity          9540 non-null   float64
 3   Price Per Unit    9540 non-null   float64
 4   Total Spent       9540 non-null   float64
 5   Payment Method    9540 non-null   object
 6   Location          9540 non-null   object
 7   Transaction Date  9540 non-null   datetime64[ns]
dtypes: datetime64[ns](1), float64(3), object(4)
memory usage: 596.4+ KB
```

Cleaning Implementation


```python
#Import required libraries
import pandas as pd
import numpy as np

#Check the current working dictionary
import os
os.getcwd()

#Load the dataset
df = pd.read_csv("dirty_cafe_sales.csv")

#==== Data Cleaning ====
#Check if there is null data in the orginal dataset
print("Original dataset information:", df.info())

#Step 1: Check and remove duplicate rows
print(df.duplicated())
df.drop_duplicates(inplace=True)
print(df.to_string())
print(df.info())

#Step 2: Convert numeric columns properly for further analysis
df['Quantity'] = pd.to_numeric(df['Quantity'], errors='coerce')
df['Price Per Unit'] = pd.to_numeric(df['Price Per Unit'], errors='coerce')
df['Total Spent'] = pd.to_numeric(df['Total Spent'], errors='coerce')
print(df.info())

#Step 3: Fill missing numerical values with median
df['Quantity'].fillna(df['Quantity'].median(), inplace=True)
df['Price Per Unit'].fillna(df['Price Per Unit'].median(), inplace=True)
df['Total Spent'].fillna(df['Total Spent'].median(), inplace=True)
#Fill missing categorical columns with mode
df['Item'].fillna(df['Item'].mode()[0], inplace=True)
df['Payment Method'].fillna(df['Payment Method'].mode()[0], inplace=True)
df['Location'].fillna(df['Location'].mode()[0], inplace=True)
print(df.info())
```

```python
#Step 4: Strip and standardize text values
df['Item'] = df['Item'].str.title().str.strip()
df['Payment Method'] = df['Payment Method'].str.title().str.strip()
df['Location'] = df['Location'].str.title().str.strip()

#Step 5: Convert Transaction Date to datetime
df['Transaction Date'] = pd.to_datetime(df['Transaction Date'], errors='coer

#Step 6: Drop rows with invalid dates
df = df.dropna(subset=['Transaction Date'])
print(df.info())

#Step 7: Recalculate total spent where needed
df['Expected Total'] = df['Quantity'] * df['Price Per Unit']
wrong_total = np.abs(df['Expected Total'] - df['Total Spent']) > 1
df.loc[wrong_total, 'Total Spent'] = df['Expected Total']
df.drop(columns='Expected Total', inplace=True)
#Final shape after cleaning
print("Cleaned dataset information:", df.info())

#Step 8: Reset the index after cleaning to ensure it's sequential
df.reset_index(drop=True, inplace=True)
print(df.info())

#Export the cleaned dataset
df.to_csv('CleanedCafeSales.csv',index=False)
print("File saved at:", os.path.abspath("CleanedCafeSales.csv"))
```

Plot using Pandas




```
#==== Plotting ====
#Plot total sales by payment method using pandas plot
plot_df = df.groupby('Payment Method')['Total Spent'].sum().sort_values()
ax = plot_df.plot(kind='barh', figsize=(10, 4), title="Total Sales by Payment Method", xlabel="Payment Method", ylabel="Total Sales", legend=False)
ax.bar_label(ax.containers[0], fmt='%.0f', label_type='edge')
```

Challenges and Solutions

Challenges

1. Type Conversions can crash script if invalid data is encountered
2. Detecting and Correcting specific column
3. Caught only the rows where Expected Total is greater than Total Spent
4. Summarizing data per category gets messy
5. Cleaned dataset has been stored to another location

How We Addressed Them (Solutions)

1. Add Errors = 'coerce' for safe conversions
   a. This avoids runtime errors and ensures the script doesn't break on bad input.
2. Use built-in pandas commands instead of a loop to improve efficiency of the code
3. Logical Masking with np.abs() for Targeted Row Correction
4. Group by + Aggregation for Summary Stats
   a. Gives sorted totals by group
5. Check the saving path with os.path.abspath and locate the cleaned csv file