Data Pre-Processing and Cleaning Using Python Report

1. **Data Collection**

We selected a data set from Kaggle called Cafe Sales. The Cafe Sales dataset represents sales transactions at a cafe. We decided to use this data set because it has 10,000 rows and 8 columns. The columns consisted of numerical, categorical, and date/time data. This data representing sales transactions is filled with missing values, duplicate rows, error or invalid values, incorrect data types, inconsistencies in formatting as well as data. Some of the calculations and formulas were incorrect, specifically for the total spent category.

2. **Data Cleaning Process**

We applied 8 distinct data cleaning processes to this data set. Our steps include checking and removing duplicate rows, converting data types to the required ones, standardizing text values, converting transaction date to datetime format, drop rows with invalid dates, replacing values in a column with new calculated value and resetting the index.

Before the data cleaning process, we load the dataset named `dirty_cafe_sales.csv` into a pandas DataFrame named `df`.

```
df = pd.read_csv("dirty_cafe_sales.csv")
```

We first check the summary of the dataset and have a overview of column data types, non-null counts, and memory usage.

```
print("Original dataset information:", df.info())
```

We then started the process of data cleaning. First, we checked for duplicate records and removed them directly from the dataframe.

```
print(df.duplicated())
df.drop_duplicates(inplace=True)
```

Second, we converted the columns ('Quantity', 'Price Per Unit', 'Total Spent') into numeric types using `pd.to_numeric`. This is crucial because these fields are required to be numerical for calculations. The `errors="coerce"` argument is added to ensure that any invalid data is replaced with the same format.

```
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')
```

Step 3 handles missing values. For numeric columns, the missing values are filled with the median of each column using `fillna(..., inplace=True)`. For categorical columns ('Item', 'Payment Method', 'Location'), the mode is used, so they are resistant to outliers and preserve the natural distribution of the data.

```
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)

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)
```

Step 4 standardizes text data by capitalizing each word and stripping whitespace from the strings in 'Item', 'Payment Method', and 'Location' columns. This makes values more uniform for analysis, by converting to title case and removing any leading/trailing spaces.

```
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()
```

In Step 5, the script converts 'Transaction Date' to datetime format. Any value that cannot be converted becomes NaT (Not a Time).

```
df['Transaction Date'] = pd.to_datetime(df['Transaction Date'], errors='coerce')
```

Step 6 drops rows where 'Transaction Date' could not be parsed and is thus null. This ensures all records have valid dates.

```
df = df.dropna(subset=['Transaction Date'])
```

In step 7, we verified if 'Total Spent' matches 'Quantity' multiplied by 'Price Per Unit'. We then created a new column 'Expected Total' as the product of 'Quantity' and 'Price Per Unit'. Then it creates a boolean mask (wrong_total) using np.abs(...) to identify rows where the difference between the expected and actual 'Total Spent' exceeds 1unit. These rows are corrected by assigning the expected value to 'Total Spent'. The temporary 'Expected Total' column is dropped afterward.

```
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)
```

We reset the dataframe index. This to ensure it is sequential and removes the old index.

```
df.reset_index(drop=True, inplace=True)
```

The cleaned dataframe is then saved and the full path is printed to show where the file is stored.

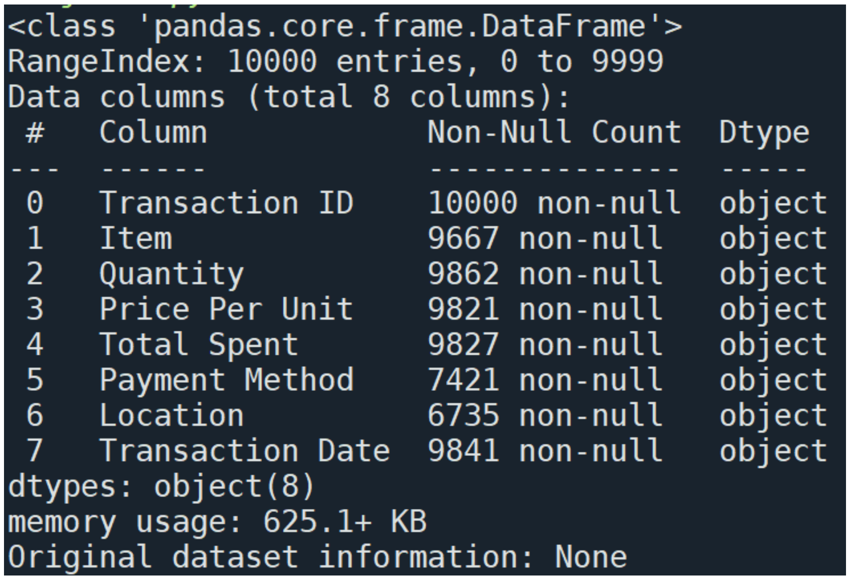
```
df.to_csv('CleanedCafeSales.csv',index=False)
print("File saved at:", os.path.abspath("CleanedCafeSales.csv"))
```

In the final section, a plot is generated to show total sales grouped by payment method using the cleaned data. The cleaned dataset is grouped using df.groupby('Payment Method')['Total Spent'].sum() and sorted. The resulting series is plotted as a horizontal bar chart using pandas' built-in plot() function which uses matplotlib under the hood. The ax.bar_label(...) line adds value labels to the ends of the bars, formatted as whole numbers.

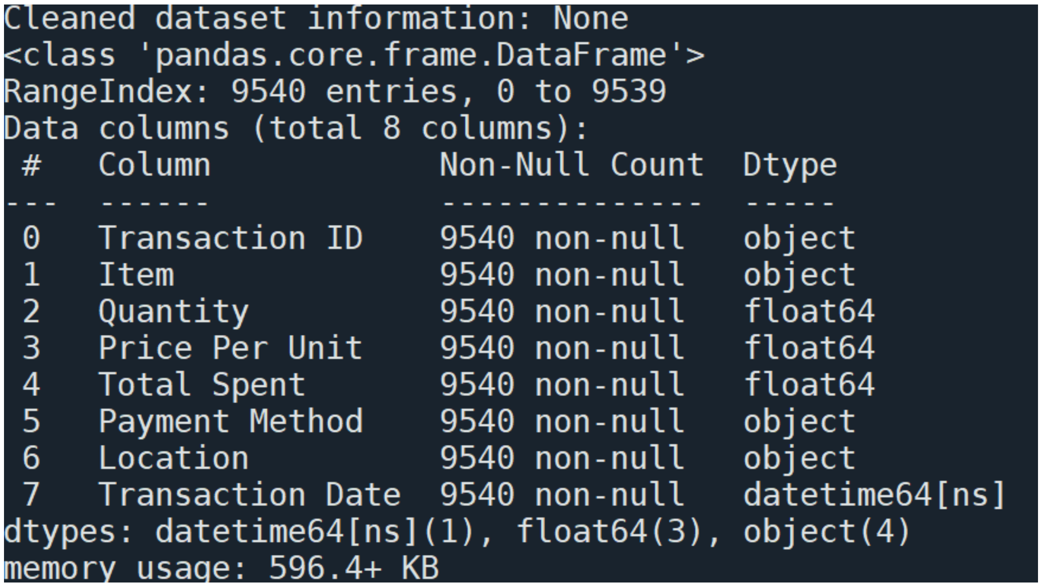
```
plot_df = df.groupby('Payment Method')['Total Spent'].sum().sort_values()
ax = plot_df.plot(kind='barh', figsize=(10, 4), title="Total Sales by Payment Method",
xlabel="Payment Method", ylabel="Total Sales", legend=False)
ax.bar_label(ax.containers[0], fmt='%.0f', label_type='edge')
```

3. **Final Dataset Summary**

Our dataset originally contained 10,000 rows, 8 columns, and improper data types. After our 8 step data cleaning process, our new dataset contains 9540 rows, 8 columns, and suitable data types. We were able to get rid of inconsistencies and instead of every category having a different amount of attributes, every column now has 9540.

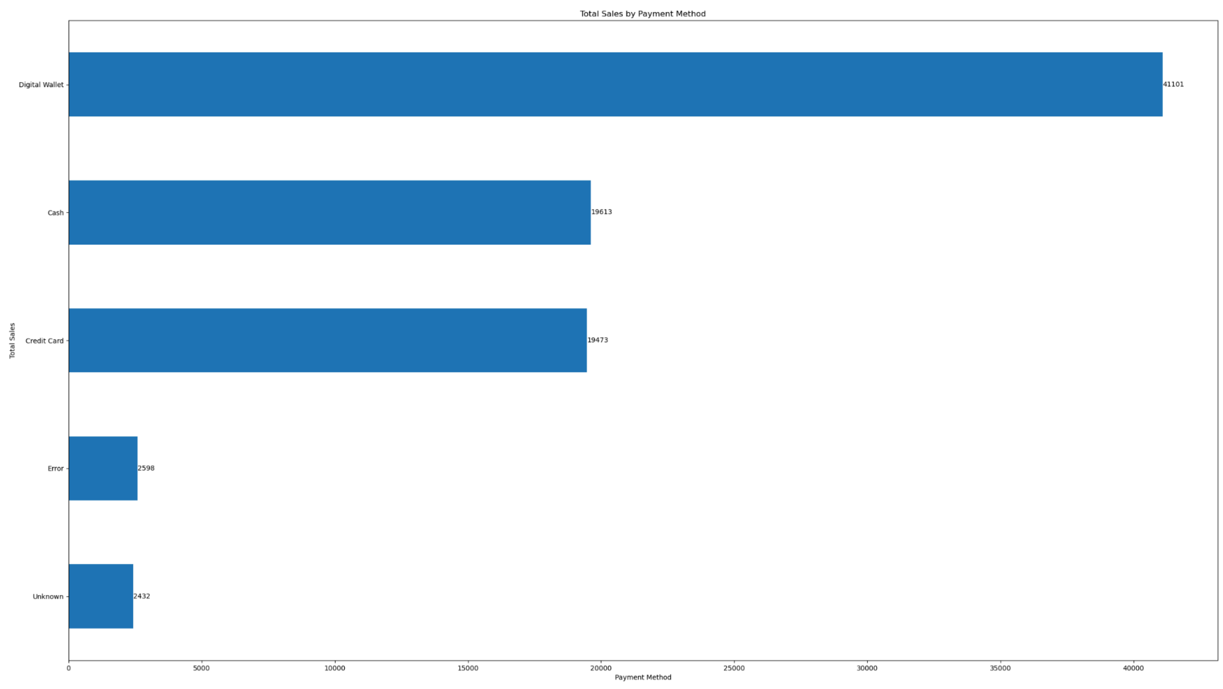


```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 8 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   Transaction ID    10000 non-null  object
 1   Item              9667 non-null   object
 2   Quantity          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
```



```
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
```

4. **Plot the Clean Data: Using Pandas:**



5. **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