Walmart is the biggest retail store in the United States. Just like others, they have been expanding their e-commerce part of the business. By the end of 2022, e-commerce represented a roaring $80 billion in sales, which is 13% of total sales of Walmart. One of the main factors that affects their sales is public holidays, like the Super Bowl, Labour Day, Thanksgiving, and Christmas.

In this project, I have been tasked with creating a data pipeline for the analysis of supply and demand around the holidays, along with conducting a preliminary analysis of the data. I will be working with two data sources: grocery sales and complementary data. I have been provided with the `grocery_sales` table in `PostgreSQL` database with the following features:

`grocery_sales`

- `"index"` - unique ID of the row
- `"Store_ID"` - the store number
- `"Date"` - the week of sales
- `"Weekly_Sales"` - sales for the given store

Also, I have the `extra_data.parquet` file that contains complementary data:

`extra_data.parquet`

- `"IsHoliday"` - Whether the week contains a public holiday - 1 if yes, 0 if no.
- `"Temperature"` - Temperature on the day of sale
- `"Fuel_Price"` - Cost of fuel in the region
- `"CPI"` – Prevailing consumer price index
- `"Unemployment"` - The prevailing unemployment rate
- `"MarkDown1"`, `"MarkDown2"`, `"MarkDown3"`, `"MarkDown4"` - number of promotional markdowns
- `"Dept"` - Department Number in each store
- `"Size"` - size of the store
- `"Type"` - type of the store (depends on `Size` column)

I will need to merge those files and perform some data manipulations. The transformed DataFrame can then be stored as the `clean_data` variable containing the following columns:

- `"Store_ID"`
- `"Month"`
- `"Dept"`
- `"IsHoliday"`
- `"Weekly_Sales"`
- `"CPI"`
- "`"Unemployment"`"

After merging and cleaning the data, I will have to analyze monthly sales of Walmart and store the results of my analysis as the `agg_data` variable.

Finally, I will save the `clean_data` and `agg_data` as the csv files, and using `pandas` for this project.

Projects Data DataFrame as `grocery_sales`

```sql
-- SQL query to retrieve data
SELECT * FROM grocery_sales
```

Hidden output

```python
import pandas as pd
import os

# Extraction function
def extract(store_data, extra_data):
    extra_df = pd.read_parquet(extra_data)
    merged_df = store_data.merge(extra_df, on = "index")
    return merged_df

# Call the extract() function and store it as the "merged_df" variable
merged_df = extract(grocery_sales, "extra_data.parquet")
```

```python
# Clean and preprocess the raw data. The transform() function with one parameter: "raw_data"
def transform(raw_data):
    # Fill missing numerical values with the column mean
    raw_data.fillna(raw_data.mean(numeric_only=True), inplace=True)

    # Extract the month from the 'Date' column (if exists)
    if 'Date' in raw_data.columns:
        raw_data['Month'] = pd.to_datetime(raw_data['Date']).dt.month

    # Filter rows where "Weekly_sales" is greater than 10K
    if 'Weekly_Sales' in raw_data.columns:
        raw_data = raw_data[raw_data['Weekly_Sales'] > 10000]

    #Drop unnecessary columns
    #drop_columns = ['index', 'Temperature', 'Fuel_Price', 'MarkDown1']
    #raw_data.drop(columns=[col for col in drop_columns if col in raw_data.columns], inplace=True)

    # List of columns to keep
    columns_to_keep = ["Store_ID", "Month", "Dept", "IsHoliday", "Weekly_Sales", "CPI", "Unemployment"]

    # Select only the columns to keep and update the DataFrame
    raw_data = raw_data[columns_to_keep]

    return raw_data
```

```python
# Call the transform() function and pass the merged DataFrame
clean_data = transform(merged_df)

print(clean_data.head())
```

```
   Store_ID  Month  Dept  IsHoliday  Weekly_Sales         CPI  Unemployment
0         1    2.0     1          0      24924.50  211.096358      8.106000
1         1    2.0    26          0      11737.12  211.096358      8.106000
2         1    2.0    17          0      13223.76  211.096358      8.106000
5         1    2.0    79          0      46729.77  211.096358      7.500052
6         1    2.0    55          0      21249.31  211.096358      7.500052
```

```python
# The avg_weekly_sales_per_month function that takes in the cleaned data from the last step
def avg_weekly_sales_per_month(clean_data):
    # check for weekly_sales column
    if 'Weekly_Sales' not in clean_data.columns:
        raise ValueError("The dataset must contain 'Weekly_Sales' column.")

    # group by 'Month' and calculate the average
    avg_sales = clean_data.groupby('Month')['Weekly_Sales'].mean().reset_index()

    # rename the column for clarity
    avg_sales.rename(columns={'Weekly_Sales': 'Avg_Sales'}, inplace=True)

    # round the 'Avg_Weekly_Sales' to two decimal places
    avg_sales['Avg_Sales'] = avg_sales['Avg_Sales'].round(2)

    return avg_sales

agg_data = avg_weekly_sales_per_month(clean_data)
```

```python
# Call the avg_weekly_sales_per_month() function and pass the cleaned DataFrame

avg_sales_per_month = avg_weekly_sales_per_month(clean_data)
print(avg_sales_per_month.head())
```

```
   Month  Avg_Sales
0    1.0   33174.18
1    2.0   34333.33
2    3.0   33220.89
3    4.0   33392.37
4    5.0   33339.89
```

```python
# The load() function that takes in the cleaned DataFrame and the aggregated one with the paths where they are going to be stored
def load(full_data, full_data_file_path, agg_data, agg_data_file_path):
    """
    Saves the cleaned dataset and aggregated dataset to specified file paths.

    Parameters:
    - full_data (DataFrame): The cleaned full dataset.
    - full_data_file_path (str): The file path to save the full dataset.
    - agg_data (DataFrame): The aggregated dataset (average weekly sales per month).
    - agg_data_file_path (str): The file path to save the aggregated dataset.

    Returns:
    - None
    """
    try:
        # Save the cleaned full dataset
        full_data.to_csv(full_data_file_path, index=False)
        print(f"Full cleaned data saved to {full_data_file_path}")

        # Save the aggregated dataset
        agg_data.to_csv(agg_data_file_path, index=False)
        print(f"Aggregated data saved to {agg_data_file_path}")

    except Exception as e:
        print(f"Error saving files: {e}")
```

```python
# Call the load() function and pass the cleaned and aggregated DataFrames with their paths
load(clean_data, "clean_data.csv", avg_sales_per_month, "avg_sales_per_month.csv")
```

```
Full cleaned data saved to clean_data.csv
Aggregated data saved to avg_sales_per_month.csv
```

```python
# The validation() function with one parameter: file_path - to check whether the previous function was correctly executed
def validation(file_path):
    """
    Validates whether the file exists at the given file path.

    Parameters:
    - file_path (str): The file path to check.

    Returns:
    - bool: True if the file exists, False otherwise.
    """
    if os.path.exists(file_path):
        print(f"File successfully saved at {file_path}")
        return True
    else:
        print(f"Error: File not found at {file_path}")
        return False
```

```python
# Call the validation() function and pass first, the cleaned DataFrame path, and then the aggregated DataFrame path
# Save agg_data to a CSV file
agg_data_file_path = "agg_data.csv"
agg_data.to_csv(agg_data_file_path, index=False)

clean_data_file_path = "clean_data.csv"
agg_data_file_path = "avg_sales_per_month.csv"

validation(clean_data_file_path)  # Validate cleaned data file
validation(agg_data_file_path)  # Validate aggregated data file
```

```
File successfully saved at clean_data.csv
File successfully saved at avg_sales_per_month.csv
```

```
True
```