Jupyter Notebook Documentation Report

1. Introduction

This report documents the end-to-end workflow implemented in two Jupyter Notebooks that operate on a retail sales dataset obtained from Kaggle. The notebooks focus on data preprocessing, exploratory data analysis (EDA), feature engineering, and predictive modeling of weekly store-level sales. Each code cell is described in plain English to help readers understand the purpose and functionality of every step in the pipeline, even if they are not familiar with the original notebook environment.

Data source: Kaggle – Retail store weekly sales dataset (insert the specific competition or dataset URL in the placeholder below).

Dataset URL placeholder: [INSERT_KAGGLE_DATASET_LINK_HERE]

2. Step-by-Step Breakdown

Notebook 1 – PREPROCESSING & VISUALIZATION

Notebook 1 – PREPROCESSING & VISUALIZATION – Cell 1

Description: This cell imports the core Python libraries required for data manipulation, visualization, and numerical computation.

Code:

```
import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

import numpy as np
```

Notebook 1 – PREPROCESSING & VISUALIZATION – Cell 2

Description: This cell mounts Google Drive in the Colab environment to make the Kaggle data files accessible.

Code:

```
from google.colab import drive

drive.mount('/content/drive')



train = pd.read_csv('/content/drive/My Drive/Colab Notebooks/shared data/train.csv')

features = pd.read_csv('/content/drive/My Drive/Colab Notebooks/shared data/features.csv')

stores = pd.read_csv('/content/drive/My Drive/Colab Notebooks/shared data/stores.csv')

customers_train = pd.read_csv('/content/drive/My Drive/Colab Notebooks/shared data/customer_train.csv')
```

Notebook 1 – PREPROCESSING & VISUALIZATION – Cell 3

Description: This cell performs intermediate data manipulation, feature engineering, or visualization that contributes to the overall preprocessing, exploration, or modeling workflow.

Code:

```
train['Date'] = pd.to_datetime(train['Date'])

features['Date'] = pd.to_datetime(features['Date'])
```

Notebook 1 – PREPROCESSING & VISUALIZATION – Cell 4

Description: This cell merges multiple DataFrames to create a single training dataset that combines store information, engineered features, and sales history.

Code:

```
train_merged = train.merge(features, on=['Store','Date','IsHoliday'], how='left')
```

Notebook 1 – PREPROCESSING & VISUALIZATION – Cell 5

Description: This cell merges multiple DataFrames to create a single training dataset that combines store information, engineered features, and sales history.

Code:

```
train_merged = train_merged.merge(stores, on='Store', how='left')
```

Notebook 1 – PREPROCESSING & VISUALIZATION – Cell 6

Description: This cell aligns and concatenates the customer-level training data with the main training set so both can be analyzed together.

Code:

```
print(f"طول train_merged: {len(train_merged)}")

print(f"طول customer_train: {len(customers_train)}")

min_len = min(len(train_merged), len(customers_train))


train_trimmed = train_merged.iloc[:min_len].reset_index(drop=True)

customers_trimmed = customers_train.iloc[:min_len].reset_index(drop=True)


train_full = pd.concat([train_trimmed, customers_trimmed], axis=1)
```

Notebook 1 – PREPROCESSING & VISUALIZATION – Cell 7

Description: This cell prints the first few rows of the combined dataset to provide an initial view of its structure and values.

Code:

```
print(train_full.head())
```

Notebook 1 – PREPROCESSING & VISUALIZATION – Cell 8

Description: This cell prints a concise summary of the DataFrame, including column counts and data types, to assess data completeness and schema.

Code:

```
print(train_full.info())
```

Notebook 1 – PREPROCESSING & VISUALIZATION – Cell 9

Description: This cell outputs descriptive statistics for the numeric variables (such as mean, standard deviation, and quartiles) to understand value distributions.

Code:

```
print(train_full.describe())
```

Notebook 1 – PREPROCESSING & VISUALIZATION – Cell 10

Description: This cell computes the number of missing values in each column to identify where imputation or cleaning may be required.

Code:

```
print(train_full.isnull().sum())
```

Notebook 1 – PREPROCESSING & VISUALIZATION – Cell 11

Description: This cell checks for and/or counts duplicated rows in the dataset in order to prevent repeated observations from biasing the analysis.

Code:

```
print(train_full.duplicated().sum())
```

Notebook 1 – PREPROCESSING & VISUALIZATION – Cell 12

Description: This cell performs intermediate data manipulation, feature engineering, or visualization that contributes to the overall preprocessing, exploration, or modeling workflow.

Code:

```
numeric_cols = train_full.select_dtypes(include=['float64', 'int64']).columns


plt.style.use('seaborn-v0_8-darkgrid')


for col in numeric_cols:
    plt.figure(figsize=(8, 4))
    sns.boxplot(x=train_full[col], color='skyblue')
    plt.title(f'Boxplot for {col}', fontsize=14)
    plt.xlabel(col)
    plt.grid(axis='x', linestyle='--', alpha=0.6)
```

```
    plt.show()
```

Notebook 1 – PREPROCESSING & VISUALIZATION – Cell 13

Description: This cell aggregates weekly sales by a time-based key (such as date, month, or week) to analyze temporal trends and seasonality.

Code:

```
sales_by_date = train_full.groupby('Date')['Weekly_Sales'].sum()


plt.figure(figsize=(14,6))
sales_by_date.plot()
plt.title('Total Weekly Sales Over Time')
plt.xlabel('Date')
plt.ylabel('Weekly Sales')
plt.grid()
plt.show()
```

Notebook 1 – PREPROCESSING & VISUALIZATION – Cell 14

Description: This cell aggregates weekly sales by a time-based key (such as date, month, or week) to analyze temporal trends and seasonality.

Code:

```
train_full['Month'] = train_full['Date'].dt.strftime('%b %Y')
train_full['Date'].dt.strftime('%b')
monthly_sales = train_full.groupby(train_full['Date'].dt.strftime('%b'))['Weekly_Sales'].mean()


month_order = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun',
               'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec']
monthly_sales = monthly_sales.reindex(month_order)


plt.figure(figsize=(10,5))
monthly_sales.plot(kind='bar')
plt.title('Average Weekly Sales by Month')
plt.xlabel('Month')
plt.ylabel('Average Weekly Sales')
plt.show()
```

Notebook 1 – PREPROCESSING & VISUALIZATION – Cell 15

Description: This cell compares average weekly sales between holiday and non-holiday periods to quantify the impact of holidays on demand.

Code:

```
holiday_sales = train_full.groupby('IsHoliday')['Weekly_Sales'].mean()


plt.figure(figsize=(6,4))
holiday_sales.plot(kind='bar', color=['skyblue', 'lightcoral'])
plt.title('Average Weekly Sales: Holiday vs. Non-Holiday')
plt.xlabel('Is Holiday')
plt.ylabel('Average Weekly Sales')
plt.xticks(rotation=0)
plt.show()
```

Notebook 1 – PREPROCESSING & VISUALIZATION – Cell 16

Description: This cell creates a scatter plot to explore the relationship between fuel prices and weekly sales.

Code:

```
sns.scatterplot(data=train_full.sample(5000), x='Fuel_Price', y='Weekly_Sales')
plt.title('Fuel Price vs Weekly Sales')
plt.show()
```

Notebook 1 – PREPROCESSING & VISUALIZATION – Cell 17

Description: This cell calculates a correlation matrix for numeric features and visualizes it as a heatmap to highlight linear relationships between variables.

Code:

```
numeric_data = train_full.select_dtypes(include=['int64', 'float64'])


corr_matrix = numeric_data.corr()


plt.figure(figsize=(14,10))
sns.heatmap(
    corr_matrix,
    cmap='coolwarm',
    annot=True,
    fmt=".2f",
```

```
    linewidths=0.5
)
plt.title('Correlation Heatmap', fontsize=16)
plt.show()
```

Notebook 1 – PREPROCESSING & VISUALIZATION – Cell 18

Description: This cell imports the core Python libraries required for data manipulation, visualization, and numerical computation.

Code:

```
# @title
import pandas as pd
from jupyter_dash import JupyterDash
from dash import dcc, html, Input, Output
import plotly.express as px




columns = [col for col in train_full.columns if col not in ['Weekly_Sales']]


app = JupyterDash(__name__)


app.layout = html.Div(
    style={'backgroundColor': '#0f172a', 'color': '#f8fafc', 'fontFamily': 'Arial, sans-serif', 'padding':
'20px'},
    children=[
        html.H1("□ Weekly Sales Insights Dashboard",
                style={'textAlign': 'center', 'color': '#38bdf8', 'marginBottom': '40px'}),


        html.Div([
            html.Label("Select a feature to compare with Weekly_Sales:",
                       style={'fontSize': '20px', 'color': '#f1f5f9', 'marginBottom': '10px'}),
            dcc.Dropdown(
                id='feature-dropdown',
```

```
                options=[{'label': col, 'value': col} for col in columns],
                value='Temperature',
                style={'width': '70%', 'margin': 'auto', 'color': '#0f172a'}
            )
        ], style={'textAlign': 'center', 'marginBottom': '40px'}),

        html.Div([
            dcc.Graph(id='sales-graph', style={'width': '60%', 'display': 'inline-block'}),
            dcc.Graph(id='hist-graph', style={'width': '38%', 'display': 'inline-block'})
        ], style={'textAlign': 'center'})
    ]
)

@app.callback(
    [Output('sales-graph', 'figure'),
     Output('hist-graph', 'figure')],
    Input('feature-dropdown', 'value')
)
def update_graphs(selected_feature):
    if train_full[selected_feature].dtype in ['float64', 'int64']:
        fig1 = px.scatter(train_full, x=selected_feature, y='Weekly_Sales',
                          title=f'Relationship between {selected_feature} and Weekly_Sales',
                          trendline="ols", opacity=0.7, color_discrete_sequence=['#38bdf8'])
    else:
        fig1 = px.box(train_full, x=selected_feature, y='Weekly_Sales',
                      title=f'Relationship between {selected_feature} and Weekly_Sales',
                      color_discrete_sequence=['#10b981'])

    fig1.update_layout(template="plotly_dark", title_font=dict(size=20, color='#f1f5f9'),
                       paper_bgcolor='#0f172a', plot_bgcolor='#0f172a')

    fig2 = px.histogram(train_full, x='Weekly_Sales', nbins=50,
```

```
                        title="Weekly_Sales Distribution", color_discrete_sequence=['#f43f5e'])
    fig2.update_layout(template="plotly_dark", title_font=dict(size=20, color='#f1f5f9'),
                       paper_bgcolor='#0f172a', plot_bgcolor='#0f172a')


    return fig1, fig2


app.run(mode='inline')
```

Notebook 1 – PREPROCESSING & VISUALIZATION – Cell 19

Description: This cell discretizes the continuous fuel price variable into categories to simplify downstream analysis and visualization.

Code:

```
train_full['Fuel_Category'] = pd.cut(train_full['Fuel_Price'], bins=10)




fuel_sales = train_full.groupby('Fuel_Category')['Weekly_Sales'].mean()


plt.figure(figsize=(10,6))
plt.bar(fuel_sales.index.astype(str), fuel_sales.values, color='blue', width=0.6)
plt.title('Average Weekly Sales vs Fuel Price Ranges', fontsize=14)
plt.xlabel('Fuel Price Range', fontsize=12)
plt.ylabel('Average Weekly Sales', fontsize=12)
plt.grid(axis='y', linestyle='--', alpha=0.6)
plt.xticks(rotation=45)
plt.show()
```

Notebook 1 – PREPROCESSING & VISUALIZATION – Cell 20

Description: This cell plots average weekly sales against temperature to investigate how weather conditions relate to sales performance.

Code:

```
temp_sales = train_full.groupby('Temperature')['Weekly_Sales'].mean().sort_index()


temp_sales_sampled = temp_sales[::50]
```

```
plt.figure(figsize=(10,6))

plt.plot(temp_sales_sampled.index, temp_sales_sampled.values, color='royalblue', linewidth=2)

plt.title('Average Weekly Sales vs Temperature',fontsize=14)

plt.xlabel('Temperature', fontsize=12)

plt.ylabel('Average Weekly Sales', fontsize=12)

plt.show()
```

Notebook 1 – PREPROCESSING & VISUALIZATION – Cell 21

Description: This cell identifies and visualizes the top-performing departments based on total weekly sales.

Code:

```
top_depts = train_full.groupby('Dept')['Weekly_Sales'].sum().sort_values(ascending=False).head(10)

top_depts.plot(kind='bar', figsize=(12,5), title='Top 10 Departments by Total Sales')
```

Notebook 1 – PREPROCESSING & VISUALIZATION – Cell 22

Description: This cell derives promotion-related features from the MarkDown columns and examines how promotional activity correlates with weekly sales.

Code:

```
markdown_cols = ['MarkDown1','MarkDown2','MarkDown3','MarkDown4','MarkDown5']

corrs = train_full[markdown_cols + ['Weekly_Sales']].corr()['Weekly_Sales'].sort_values(ascending=False)


plt.figure(figsize=(8,4))

bars = corrs.drop('Weekly_Sales').plot(kind='bar', color='teal')

plt.title('Correlation of MarkDowns with Weekly Sales')

plt.xlabel('MarkDown Feature')

plt.ylabel('Correlation Coefficient')

plt.ylim(0, 1)

plt.xticks(rotation=0)


for i, v in enumerate(corrs.drop('Weekly_Sales')):

    plt.text(i, v + 0.025, f"{v:.3f}", ha='center', fontsize=10)


plt.show()
```

Notebook 1 – PREPROCESSING & VISUALIZATION – Cell 23

Description: This cell derives promotion-related features from the MarkDown columns and examines how promotional activity correlates with weekly sales.

Code:

```
train_full['HasPromotion'] =
train_full[['MarkDown1','MarkDown2','MarkDown3','MarkDown4','MarkDown5']].sum(axis=1) > 0
```

Notebook 1 – PREPROCESSING & VISUALIZATION – Cell 24

Description: This cell performs intermediate data manipulation, feature engineering, or visualization that contributes to the overall preprocessing, exploration, or modeling workflow.

Code:

```
sns.barplot(data=train_full, x=train_full['Date'].dt.strftime('%b'), y='Weekly_Sales', hue='HasPromotion',
order=month_order)

plt.title('Average Sales by Month (With and Without Promotions)')

plt.legend(title='Has Promotion', loc='lower left')

plt.show()
```

Notebook 1 – PREPROCESSING & VISUALIZATION – Cell 25

Description: This cell derives promotion-related features from the MarkDown columns and examines how promotional activity correlates with weekly sales.

Code:

```
train_full = train_full.drop_duplicates()



train_full[markdown_cols] = train_full[markdown_cols].fillna(0)



cols = [

    'Weekly_Sales', 'Temperature', 'MarkDown1', 'MarkDown2', 'MarkDown3',

    'MarkDown4', 'MarkDown5', 'Unemployment',

    'Num_Customers', 'Avg_Spend_per_Customer', 'Loyalty_Avg'

]




for col in cols:

    q1 = train_full[col].quantile(0.25)

    q3 = train_full[col].quantile(0.75)

    iqr = q3 - q1

    upper = q3 + 1.5 * iqr

    lower = q1 - 1.5 * iqr
```

```
    before_outliers = ((train_full[col] < lower) | (train_full[col] > upper)).sum()
    train_full[col] = train_full[col].clip(lower, upper)
    #after_outliers = ((train_full[col] < lower) | (train_full[col] > upper)).sum()

    print(f"🔹 {col}: {before_outliers} outliers were clipped to within bounds")

print("تمت المعالجة train_full")
```

Notebook 1 – PREPROCESSING & VISUALIZATION – Cell 26

Description: This cell performs intermediate data manipulation, feature engineering, or visualization that contributes to the overall preprocessing, exploration, or modeling workflow.

Code:

```
plt.style.use('seaborn-v0_8-darkgrid')

for col in numeric_cols:
    plt.figure(figsize=(8, 4))
    sns.boxplot(x=train_full[col], color='skyblue')
    plt.title(f'Boxplot for {col}', fontsize=14)
    plt.xlabel(col)
    plt.grid(axis='x', linestyle='--', alpha=0.6)
    plt.show()
```

Notebook 1 – PREPROCESSING & VISUALIZATION – Cell 27

Description: This cell computes the number of missing values in each column to identify where imputation or cleaning may be required.

Code:

```
print(train_full.isnull().sum())
```

Notebook 1 – PREPROCESSING & VISUALIZATION – Cell 28

Description: This cell engineers calendar-based features (year, month, week, day, and day-of-week) from the original date column to better capture seasonality and calendar effects.

Code:

```
train_full.drop('Month', axis=1, inplace=True)

train_full['Year'] = train_full['Date'].dt.year
```

```
train_full['Month'] = train_full['Date'].dt.month

train_full['Week'] = train_full['Date'].dt.isocalendar().week

train_full['Day'] = train_full['Date'].dt.day

train_full['DayOfWeek'] = train_full['Date'].dt.dayofweek
```

Notebook 1 – PREPROCESSING & VISUALIZATION – Cell 29

Description: This cell sorts the dataset by store, department, and date to ensure a consistent temporal ordering of observations.

Code:

```
train_full = train_full.sort_values(['Store', 'Dept', 'Date'])
```

Notebook 1 – PREPROCESSING & VISUALIZATION – Cell 30

Description: This cell casts boolean indicator variables such as IsHoliday (and potentially HasPromotion) to integer format (0/1) required by many modeling algorithms.

Code:

```
train_full['IsHoliday'] = train_full['IsHoliday'].astype(int)

train_full['HasPromotion'] = train_full['HasPromotion'].astype(int)



train_full.drop('Date', axis=1, inplace=True)
```

Notebook 1 – PREPROCESSING & VISUALIZATION – Cell 31

Description: This cell prints the first few rows of the combined dataset to provide an initial view of its structure and values.

Code:

```
print(train_full.head())
```

Notebook 1 – PREPROCESSING & VISUALIZATION – Cell 32

Description: This cell performs one-hot encoding on categorical features so that they can be supplied as numeric inputs to machine learning models.

Code:

```
train_full = pd.get_dummies(train_full, columns=['Type', 'Fuel_Category'])

train_full = train_full.astype({col: 'int' for col in train_full.select_dtypes('bool').columns})



#train_full.to_csv('FData.csv', index=False)
```

Notebook 1 – PREPROCESSING & VISUALIZATION – Cell 33

Description: This cell prints the first few rows of the combined dataset to provide an initial view of its structure and values.

Code:

```
print(train_full.head())
```

Notebook 1 – PREPROCESSING & VISUALIZATION – Cell 34

Description: This cell is empty and does not perform any operation.

Code:

```
[No content in this cell]
```

Notebook 1 – PREPROCESSING & VISUALIZATION – Cell 35

Description: This markdown cell provides narrative documentation or section headings that structure the notebook for readability.

Code:

```
## Feature Engineering Summary


The following steps were performed to prepare the dataset for modeling:


1. **Datetime Features**
   - Extracted temporal features such as:
     - Month (seasonality)
     - Year (trend)
     - Week of Year
   - These features help models learn repeating patterns over time.


2. **Promotion Feature**
   - Created a binary feature `HasPromotion` from MarkDown1–MarkDown5.
   - Indicates whether any promotional markdown occurred during the week.


3. **Categorical Encoding**
   - Store Type (A, B, C) encoded into a numerical representation.


4. **Data Merging**
   - Combined Train/Test with Features and Store metadata using keys:
     `Store`, `Dept`, `Date`.


5. **Scaling / Normalization**
```

```
   - Applied scaling to continuous variables (e.g., Temperature, Fuel Price, CPI, Unemployment, Store Size) when needed for algorithms sensitive to feature magnitude (e.g., linear models, neural networks).

   - This ensures all numerical features contribute proportionally during model training.


5. **Feature Selection**

   - All original sales-related variables were retained except those with extremely low or random signal.


Overall, the engineered features capture seasonal, promotional, and store-related effects — key drivers of weekly retail sales.
```

Notebook 1 – PREPROCESSING & VISUALIZATION – Cell 36

Description: This cell is empty and does not perform any operation.

Code:

```
[No content in this cell]
```

Notebook 1 – PREPROCESSING & VISUALIZATION – Cell 37

Description: This markdown cell provides narrative documentation or section headings that structure the notebook for readability.

Code:

```
## EDA Summary


- The data shows strong **seasonal patterns**, with sales peaking in November and December.

- **Holiday weeks** significantly increase weekly sales.

- **Store Type A** has the highest average sales, followed by Types B and C.

- **Store size** correlates positively with sales volume.

- **Top Performing Departments:** Sales are concentrated in specific departments, with Departments **92, 95, and 38** generating the highest total sales across all stores.

- **Temperature has little to no clear relationship with weekly sales**, as the trend fluctuates without a consistent pattern. This suggests temperature is **not a strong predictive feature** for sales forecasting.

- **Markdown features** show weak correlation with Weekly Sales, suggesting promotional effects may be localized or require more advanced modeling.

- Economic factors (CPI, Fuel Price, Unemployment) vary across time and stores, suggesting potential long-term trend influence.
```

Notebook 1 – PREPROCESSING & VISUALIZATION – Cell 38

Description: This cell is empty and does not perform any operation.

Code:

```
[No content in this cell]
```

Notebook 2 – PREPROCESSING & MODELING

Notebook 2 – PREPROCESSING & MODELING – Cell 1

Description: This cell imports the core Python libraries required for data manipulation, visualization, and numerical computation.

Code:

```
import pandas as pd

import numpy as np

from sklearn.model_selection import RandomizedSearchCV, TimeSeriesSplit

from sklearn.linear_model import LinearRegression

from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor

from xgboost import XGBRegressor

from lightgbm import LGBMRegressor

from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score, mean_absolute_percentage_error

from sklearn.preprocessing import MinMaxScaler
```

Notebook 2 – PREPROCESSING & MODELING – Cell 2

Description: This cell defines an XGBoost regression model that will be used to predict weekly sales based on the engineered features.

Code:

```
#!pip install xgboost lightgbm
```

Notebook 2 – PREPROCESSING & MODELING – Cell 3

Description: This cell mounts Google Drive in the Colab environment to make the Kaggle data files accessible.

Code:

```
from google.colab import drive

drive.mount('/content/drive')
```

Notebook 2 – PREPROCESSING & MODELING – Cell 4

Description: This cell reads one or more CSV files from storage into pandas DataFrames that will be used throughout the analysis pipeline.

Code:

```
train = pd.read_csv('/content/drive/My Drive/Colab Notebooks/shared data/train.csv')

features = pd.read_csv('/content/drive/My Drive/Colab Notebooks/shared data/features.csv')

stores = pd.read_csv('/content/drive/My Drive/Colab Notebooks/shared data/stores.csv')

customers_train = pd.read_csv('/content/drive/My Drive/Colab Notebooks/shared data/customer_train.csv')
```

Notebook 2 – PREPROCESSING & MODELING – Cell 5

Description: This cell performs intermediate data manipulation, feature engineering, or visualization that contributes to the overall preprocessing, exploration, or modeling workflow.

Code:

```
train['Date'] = pd.to_datetime(train['Date'])

features['Date'] = pd.to_datetime(features['Date'])
```

Notebook 2 – PREPROCESSING & MODELING – Cell 6

Description: This cell merges multiple DataFrames to create a single training dataset that combines store information, engineered features, and sales history.

Code:

```
train_merged = train.merge(features, on=['Store','Date','IsHoliday'], how='left')
```

Notebook 2 – PREPROCESSING & MODELING – Cell 7

Description: This cell merges multiple DataFrames to create a single training dataset that combines store information, engineered features, and sales history.

Code:

```
train_merged = train_merged.merge(stores, on='Store', how='left')
```

Notebook 2 – PREPROCESSING & MODELING – Cell 8

Description: This cell aligns and concatenates the customer-level training data with the main training set so both can be analyzed together.

Code:

```
print(f"طول train_merged: {len(train_merged)}")

print(f"طول customer_train: {len(customers_train)}")

min_len = min(len(train_merged), len(customers_train))


train_trimmed = train_merged.iloc[:min_len].reset_index(drop=True)

customers_trimmed = customers_train.iloc[:min_len].reset_index(drop=True)


train_full = pd.concat([train_trimmed, customers_trimmed], axis=1)
```

Notebook 2 – PREPROCESSING & MODELING – Cell 9

Description: This cell performs intermediate data manipulation, feature engineering, or visualization that contributes to the overall preprocessing, exploration, or modeling workflow.

Code:

```
# train_full['Month'] = train_full['Date'].dt.strftime('%b %Y')

# train_full['Date'].dt.strftime('%b')
```

Notebook 2 – PREPROCESSING & MODELING – Cell 10

Description: This cell discretizes the continuous fuel price variable into categories to simplify downstream analysis and visualization.

Code:

```
train_full['Fuel_Category'] = pd.cut(train_full['Fuel_Price'], bins=10)
```

Notebook 2 – PREPROCESSING & MODELING – Cell 11

Description: This cell derives promotion-related features from the MarkDown columns and examines how promotional activity correlates with weekly sales.

Code:

```
train_full['HasPromotion'] =
train_full[['MarkDown1','MarkDown2','MarkDown3','MarkDown4','MarkDown5']].sum(axis=1) > 0
```

Notebook 2 – PREPROCESSING & MODELING – Cell 12

Description: This cell derives promotion-related features from the MarkDown columns and examines how promotional activity correlates with weekly sales.

Code:

```
train_full = train_full.drop_duplicates()



markdown_cols = ['MarkDown1','MarkDown2','MarkDown3','MarkDown4','MarkDown5']

train_full[markdown_cols] = train_full[markdown_cols].fillna(0)



cols = [

    'Weekly_Sales', 'Temperature', 'MarkDown1', 'MarkDown2', 'MarkDown3',

    'MarkDown4', 'MarkDown5', 'Unemployment',

    'Num_Customers', 'Avg_Spend_per_Customer', 'Loyalty_Avg'

]




for col in cols:

    q1 = train_full[col].quantile(0.25)

    q3 = train_full[col].quantile(0.75)

    iqr = q3 - q1

    upper = q3 + 1.5 * iqr

    lower = q1 - 1.5 * iqr
```

```
    before_outliers = ((train_full[col] < lower) | (train_full[col] > upper)).sum()
    train_full[col] = train_full[col].clip(lower, upper)
    after_outliers = ((train_full[col] < lower) | (train_full[col] > upper)).sum()

    print(f"✂ {col}: {before_outliers} outliers were clipped to within bounds")

print("تمت المعالجة train_full")
```

Notebook 2 – PREPROCESSING & MODELING – Cell 13

Description: This cell engineers calendar-based features (year, month, week, day, and day-of-week) from the original date column to better capture seasonality and calendar effects.

Code:

```
train_full['Year'] = train_full['Date'].dt.year
train_full['Month'] = train_full['Date'].dt.month
train_full['Week'] = train_full['Date'].dt.isocalendar().week
train_full['Day'] = train_full['Date'].dt.day
train_full['DayOfWeek'] = train_full['Date'].dt.dayofweek
```

Notebook 2 – PREPROCESSING & MODELING – Cell 14

Description: This cell sorts the dataset by store, department, and date to ensure a consistent temporal ordering of observations.

Code:

```
train_full = train_full.sort_values(['Store', 'Dept', 'Date'])
```

Notebook 2 – PREPROCESSING & MODELING – Cell 15

Description: This cell casts boolean indicator variables such as IsHoliday (and potentially HasPromotion) to integer format (0/1) required by many modeling algorithms.

Code:

```
train_full['IsHoliday'] = train_full['IsHoliday'].astype(int)
train_full['HasPromotion'] = train_full['HasPromotion'].astype(int)



train_full.drop('Date', axis=1, inplace=True)
```

Notebook 2 – PREPROCESSING & MODELING – Cell 16

Description: This cell performs one-hot encoding on categorical features so that they can be supplied as numeric inputs to machine learning models.

Code:

```
train_full = pd.get_dummies(train_full, columns=['Type', 'Fuel_Category'])

train_full = train_full.astype({col: 'int' for col in train_full.select_dtypes('bool').columns})
```

Notebook 2 – PREPROCESSING & MODELING – Cell 17

Description: This cell performs intermediate data manipulation, feature engineering, or visualization that contributes to the overall preprocessing, exploration, or modeling workflow.

Code:

```
data=train_full

data.head()
```

Notebook 2 – PREPROCESSING & MODELING – Cell 18

Description: This cell performs intermediate data manipulation, feature engineering, or visualization that contributes to the overall preprocessing, exploration, or modeling workflow.

Code:

```
data.columns = data.columns.str.replace('[^A-Za-z0-9_]+', '_', regex=True)


X = data.drop('Weekly_Sales', axis=1)

y = data['Weekly_Sales']


scaler_X = MinMaxScaler()

scaler_y = MinMaxScaler()


split_index = int(len(X) * 0.8)

X_train, X_test, y_train, y_test = (

    X.iloc[:split_index],

    X.iloc[split_index:],

    y.iloc[:split_index],

    y.iloc[split_index:]

)


# Scaling X data
```

```
X_train_scaled = scaler_X.fit_transform(X_train)

X_test_scaled = scaler_X.transform(X_test)


# Scaling y data

y_train_scaled = scaler_y.fit_transform(y_train.values.reshape(-1, 1))

y_test_scaled = scaler_y.transform(y_test.values.reshape(-1, 1))


# Transforming data to dataframe and series

X_train = pd.DataFrame(X_train_scaled, columns=X_train.columns, index=X_train.index)

X_test  = pd.DataFrame(X_test_scaled,  columns=X_test.columns,  index=X_test.index)


y_train = pd.Series(y_train_scaled.flatten(), index=y_train.index, name='Weekly_Sales')

y_test = pd.Series(y_test_scaled.flatten(), index=y_test.index, name='Weekly_Sales')


# Sampling the train data

sample_idx = y_train.sample(frac=0.1, random_state=42).index


df_xsample_train = X_train.loc[sample_idx]

df_ysample_train = y_train.loc[sample_idx]
```

Notebook 2 – PREPROCESSING & MODELING – Cell 19

Description: This cell defines an XGBoost regression model that will be used to predict weekly sales based on the engineered features.

Code:

```
xgb_model = XGBRegressor(objective='reg:squarederror', eval_metric='rmse')

xgb_params = {

    'n_estimators': [200, 500, 800],

    'max_depth': [3, 5, 7],

    'learning_rate': [0.01, 0.05, 0.1],

    'subsample': [0.7, 0.9, 1.0],

    'colsample_bytree': [0.5, 0.8, 1.0]

}
```

```
lgb_model = LGBMRegressor()
lgb_params = {
    'n_estimators': [200, 500, 800],
    'learning_rate': [0.01, 0.05, 0.1],
    'num_leaves': [15, 31, 63],
    'max_depth': [-1, 5, 10],
    'subsample': [0.7, 0.9, 1.0],
    'colsample_bytree': [0.7, 1.0]
}



rf_model = RandomForestRegressor()
rf_params = {
    'n_estimators': [200, 500, 800],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}



gbr_model = GradientBoostingRegressor()
gbr_params = {
    'n_estimators': [200, 500],
    'learning_rate': [0.01, 0.05, 0.1],
    'max_depth': [2, 3, 5]
}



models = [xgb_model, rf_model, gbr_model, lgb_model]
parameters = [xgb_params, rf_params, gbr_params, lgb_params]
model_names = ['xgb', 'rf', 'gbr', 'lgb']
```

```
all_models = []


def smape(y_true, y_pred):
    return np.mean(2 * np.abs(y_pred - y_true) / (np.abs(y_true) + np.abs(y_pred))) * 100
```

Notebook 2 – PREPROCESSING & MODELING – Cell 20

Description: This cell defines a standard linear regression model that will serve as a baseline for comparison against more complex models.

Code:

```
linear_model=LinearRegression()

linear_model.fit(df_xsample_train, df_ysample_train)

preds=linear_model.predict(X_test)


# Evaluation

mae = mean_absolute_error(y_test.values, preds)

rmse = np.sqrt(mean_squared_error(y_test.values, preds))

r2 = r2_score(y_test.values, preds)

mape = mean_absolute_percentage_error(y_test.values, preds)

smape_val = smape(y_test.values, preds)

accuracy = 100 - smape_val

print("MAE:", mae)

print("RMSE:", rmse)

print("R2:", r2)

print("MAPE:", mape)

print("SMAPE:", smape_val)

print("Approx Accuracy:", accuracy, "%")
```

Notebook 2 – PREPROCESSING & MODELING – Cell 21

Description: This cell executes hyperparameter tuning and/or cross-validation in order to optimize the configuration of the machine learning models.

Code:

```
for model, params, name in zip(models, parameters, model_names):

  tscv = TimeSeriesSplit(n_splits=5)

  random = RandomizedSearchCV(
```

```
  estimator=model,
  param_distributions=params,
  cv=tscv,
  scoring='neg_mean_absolute_error',
  n_iter=10,
  n_jobs=4,
  verbose=2,
  random_state=42
)

random.fit(df_xsample_train, df_ysample_train)
best_model = random.best_estimator_
best_params = random.best_params_
all_models.append(best_model)

preds = best_model.predict(X_test)

mae = mean_absolute_error(y_test.values, preds)
rmse = np.sqrt(mean_squared_error(y_test.values, preds))
r2 = r2_score(y_test.values, preds)
mape = mean_absolute_percentage_error(y_test.values, preds)
smape_val = smape(y_test.values, preds)
accuracy = 100 - smape_val
print('For', name + ':')
print("MAE:", mae)
print("RMSE:", rmse)
print("R2:", r2)
print("MAPE:", mape)
print("SMAPE:", smape_val)
print("Approx Accuracy:", accuracy, "%")
print()
print()
```

```
print(best_params)

print()
```

3. Conclusion

The two notebooks collectively implement a complete workflow for forecasting store-level weekly sales using a Kaggle retail dataset. The preprocessing and visualization notebook focuses on loading and merging raw data files, conducting exploratory data analysis, engineering calendar- and promotion-related features, and preparing a clean modeling dataset. The modeling notebook then leverages this processed data to define baseline and advanced regression models, perform feature transformations and encodings, and evaluate predictive performance using appropriate metrics. Together, these steps demonstrate a practical approach to building a demand forecasting solution that can be adapted to similar retail datasets.