Sales Forecasting and Optimization Project

Team members

TAWFEK MOHAMED

AHMED KANDIL

MOHAMED MOUSA

RANA MAHER

KHOLOUD M.AMER

AYA ALASSEY

SPECIAL GREETINGS TO:

Dr.Sayed Haggag

SPECIAL GREETINGS TO:

Eng.Omar Ahmed

CONTENTS

1. Talking About general view of the project's purpose and Dataset
2. About code
3. Deployment

Talking about

Sales Forecasting and Optimization Project

Sales forecasting and Optimization System is designed to help businesses make smarter decisions by accurately forecasting future sales and optimizing inventory management. It leverages historical sales data and advanced machine learning techniques to predict upcoming demand, reduce costs, and improve overall operational efficiency.

🎯 Project Purpose:

- Predict future sales using time-series forecasting models.
- Provide data-driven insights through an interactive dashboard.
- Enhance decision-making for sales, marketing, and finance teams.
- Deploy a scalable system using Streamlit and MLOps best practices.

DataSet

A dataset containing historical sales data (e.g., daily or weekly sales data from retailor e-commerce platforms). and relevant features like sales amount, date, promotions. we do feature engineering to generate holidays column .

Store Sales Forecasting Dataset

From Data to Dollars: Unraveling Store Sales Predictions

Data Card | Code (13) | Discussion (1) | Suggestions (0)

About Dataset

This dataset offers a valuable resource for businesses operating in the retail furniture sector. By analyzing historical sales data from the superstore dataset, users can gain insights into future sales patterns and trends. This information can be utilized to optimize inventory management strategies, anticipate customer demand, and enhance overall operational efficiency. Whether for retail managers, analysts, or data scientists, this dataset provides a foundation for informed decision-making, helping businesses maintain stability and drive sustained growth in the dynamic retail environment.

Usability

10.00

License

Apache 2.0

Expected update frequency

Annually

Tags

Data Visualization

Data Cleaning

Reading and Explore the Data

[2]:
```
date_columns = ["Order Date", "Ship Date"]
df = pd.read_csv("data.csv", encoding="ISO-8859-1", parse_dates=date_columns)
```

[3]:
```
df.head()
```

[3]:

	Row ID	Order ID	Order Date	Ship Date	Ship Mode	Customer ID	Customer Name	Segment	Country	City	...
0	1	CA-2016-152156	2016-11-08	2016-11-11	Second Class	CG-12520	Claire Gute	Consumer	United States	Henderson	...
1	2	CA-2016-152156	2016-11-08	2016-11-11	Second Class	CG-12520	Claire Gute	Consumer	United States	Henderson	...
2	4	US-2015-108966	2015-10-11	2015-10-18	Standard Class	SO-20335	Sean O'Donnell	Consumer	United States	Fort Lauderdale	...
3	6	CA-2014-115812	2014-06-09	2014-06-14	Standard Class	BH-11710	Brosina Hoffman	Consumer	United States	Los Angeles	...
4	11	CA-2014-	2014-06-09	2014-06-14	Standard Class	BH-11710	Brosina Hoffman	Consumer	United States	Los Angeles	...

Import necessary libraries

Seaborn and Matplotlib: for visualizing and analyzing data.

Pandas and Numpy: for data manipulation and analysis.

```
[1]: import numpy as np
     import pandas as pd
     import matplotlib.pyplot as plt
     import seaborn as sns

     sns.set_theme(context="notebook", style="darkgrid", palette="pastel")

     import holidays

     from xgboost import XGBRegressor
     from sklearn.metrics import mean_squared_error, mean_absolute_error

     from skopt import BayesSearchCV
     from skopt.space import Real, Integer
```

XGBoost is a High-performance gradient boosting library optimized for speed and accuracy.

- Use in project: Main regression model for predicting future sales based on historical patterns.

Scikit-Learn Provides machine learning tools and evaluation metrics.

- Use in project:
  - Metrics: Evaluating model performance using Mean Squared Error (MSE) and Mean Absolute Error (MAE).

```python
date_columns = ["Order Date", "Ship Date"]
df = pd.read_csv("data.csv", encoding="ISO-8859-1", parse_dates=date_columns)
```

```python
df.describe(include="object")
```

Python

	Order ID	Ship Mode	Customer ID	Customer Name	Segment	Country	City	State	Region	Product ID	Category	Sub-Category	Product Name	holiday_name
count	2121	2121	2121	2121	2121	2121	2121	2121	2121	2121	2121	2121	2121	168
unique	1764	4	707	707	3	1	371	48	4	375	1	4	380	17
top	US-2015-129007	Standard Class	SV-20365	Seth Vernon	Consumer	United States	New York City	California	West	FUR-FU-10004270	Furniture	Furnishings	KI Adjustable-Height Table	Christmas Day
freq	4	1248	15	15	1113	2121	192	444	707	16	2121	957	18	18

```python
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2121 entries, 0 to 2120
Data columns (total 28 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Row ID         2121 non-null   int64
 1   Order ID       2121 non-null   object
 2   Order Date     2121 non-null   datetime64[ns]
 3   Ship Date      2121 non-null   datetime64[ns]
 4   Ship Mode      2121 non-null   category
 5   Customer ID    2121 non-null   object
 6   Customer Name  2121 non-null   object
 7   Segment        2121 non-null   object
 8   Country        2121 non-null   object
 9   City           2121 non-null   object
 10  State          2121 non-null   object
 11  Postal Code    2121 non-null   int64
 12  Region         2121 non-null   object
 13  Product ID     2121 non-null   object
 14  Category       2121 non-null   category
 15  Sub-Category   2121 non-null   object
 16  Product Name   2121 non-null   object
 17  Sales          2121 non-null   float64
 18  Quantity       2121 non-null   float64
 19  Discount       2121 non-null   float64
 20  Profit         2121 non-null   float64
 21  is_holiday     2121 non-null   bool
 22  holiday_name   158 non-null    object
 23  Order_Year     2121 non-null   int32
 24  Order_Month    2121 non-null   int32
 25  Order_Day      2121 non-null   int32
 26  Order_Weekday  2121 non-null   int32
 27  Shipping_Days  2121 non-null   int64
dtypes: bool(1), category(2), datetime64[ns](2), float64(4), int32(4), int64(3), object(12)
memory usage: 387.8+ KB
```

1

2

```python
print(df.describe())  # Summary statistics
```

```
            Row ID                     Order Date  \
count  2121.000000                           2121
mean   5041.643564  2016-04-30 03:54:13.748231680
min       1.000000            2014-01-06 00:00:00
25%    2568.000000            2015-05-26 00:00:00
50%    5145.000000            2016-06-20 00:00:00
75%    7534.000000            2017-05-14 00:00:00
max    9991.000000            2017-12-30 00:00:00
std    2885.740258                            NaN

                           Ship Date   Postal Code         Sales  \
count                           2121   2121.000000  2.121000e+03
mean   2016-05-04 01:54:44.299858432  55726.556341 -1.172513e-17
min              2014-01-10 00:00:00   1040.000000 -6.916522e-01
25%              2015-05-31 00:00:00  22801.000000 -6.019055e-01
50%              2016-06-23 00:00:00  60505.000000 -3.331903e-01
75%              2017-05-18 00:00:00  90032.000000  1.696279e-01
max              2018-01-05 00:00:00  99301.000000  8.083201e+00
std                              NaN  32261.888225  1.000236e+00

           Quantity      Discount        Profit
count  2.121000e+03  2.121000e+03  2.121000e+03
mean   1.172513e-17  1.122262e-16 -1.028461e-15
min   -1.237182e+00 -9.582280e-01 -1.375570e+01
25%   -7.929525e-01 -9.582280e-01 -1.584236e-01
50%   -3.487232e-01  1.436732e-01 -6.797135e-03
75%    5.397355e-01  6.946237e-01  1.840008e-01
max    4.537800e+00  2.898426e+00  7.384565e+00
std    1.000236e+00  1.000236e+00  1.000236e+00
```

Missing values & Duplicatation

```python
print("Duplicate Rows:", df.duplicated().sum())

df = df.drop_duplicates()

print("Duplicate Rows After Removal:", df.duplicated().sum())
```

```
Duplicate Rows: 0
Duplicate Rows After Removal: 0
```

```python
df.isna().sum()
```

```
Row ID            0
Order ID          0
Order Date        0
Ship Date         0
Ship Mode         0
Customer ID       0
Customer Name     0
Segment           0
Country           0
City              0
State             0
Postal Code       0
Region            0
Product ID        0
Category          0
Sub-Category      0
Product Name      0
Sales             0
Quantity          0
Discount          0
Profit            0
is_holiday        0
holiday_name   1963
dtype: int64
```

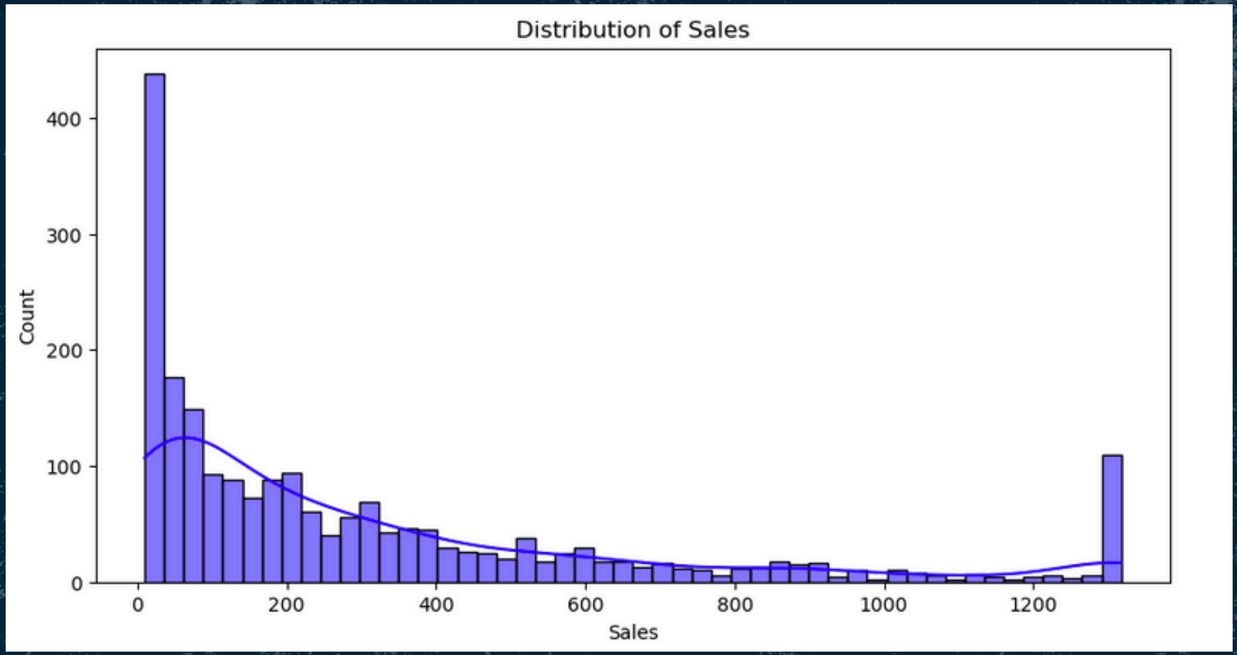
Data analysis

Exploratory Data Analysis (EDA)

1. **Sales & Profit Distribution** (Histogram)
2. **Sales vs Profit** (Scatter Plot)
3. **Missing Values** (Heatmap)
4. **Sales Trend Over Time** (Line Chart)
5. **Data Distribution**
6. **Category Distribution**
7. **Correlation Heatmap**
8. **Sales by Sub-Category**
9. **Sales and Profit by Sub-Category**
10. **Discount vs Profit** (Scatter Plot with Color)
11. **Sales by Month** (Based on Ship Date)
12. **Holiday vs Non-Holiday Sales** y Sales**

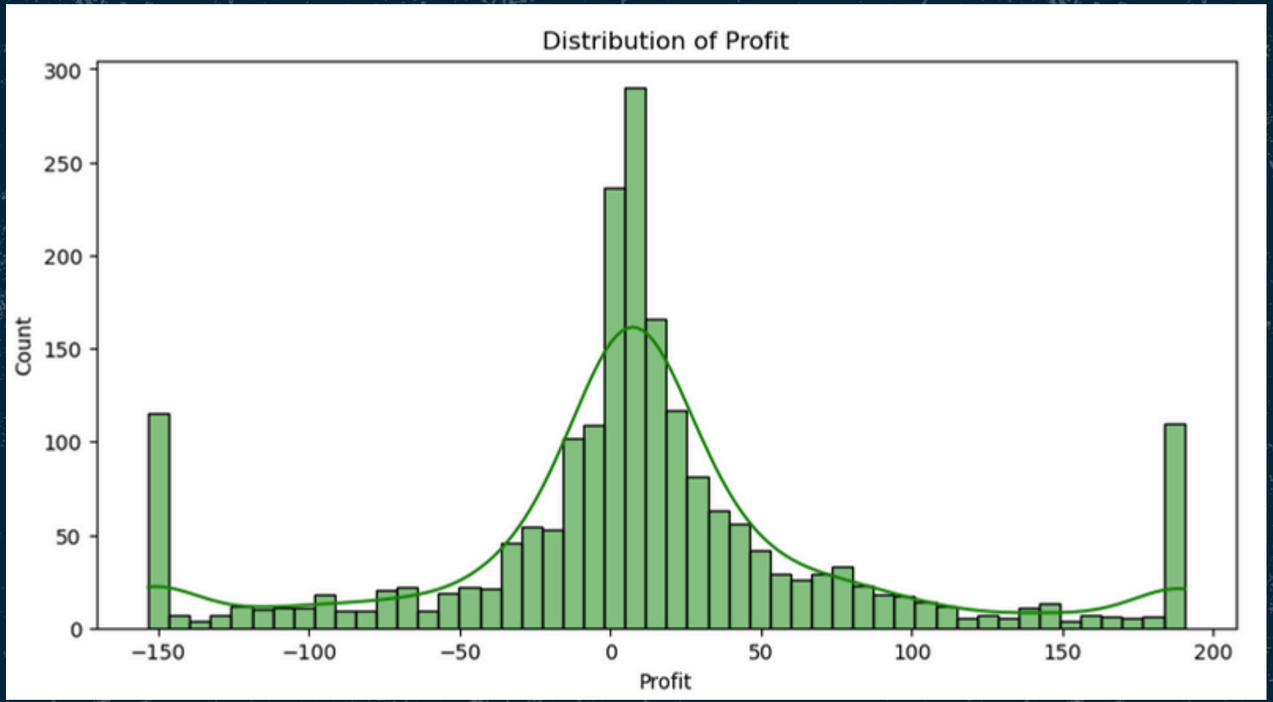
1️⃣ Sales & Profit Distribution (Histogram)

```
import matplotlib.pyplot as plt
import seaborn as sns

# Histogram for Sales
plt.figure(figsize=(10, 5))
sns.histplot(df['Sales'], bins=50, kde=True, color='blue')
plt.title("Distribution of Sales")
plt.xlabel("Sales")
plt.ylabel("Count")
plt.show()
```



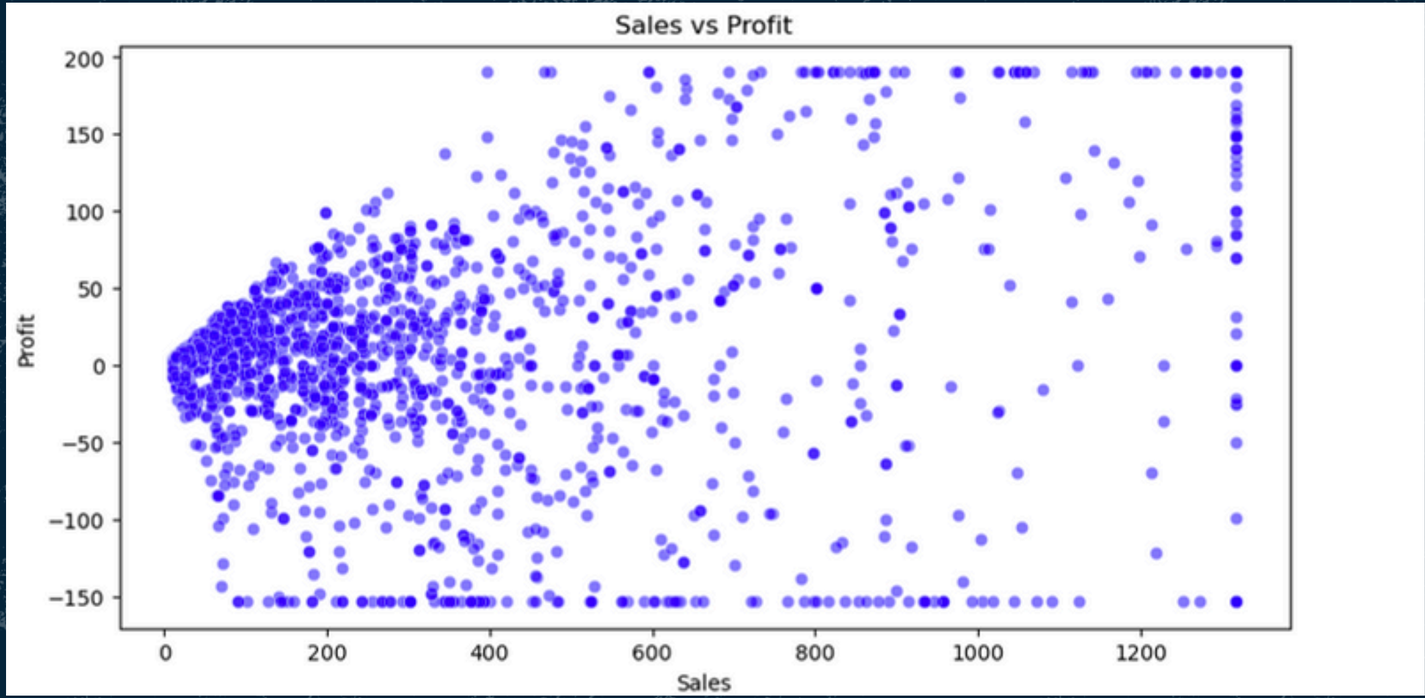
1 Sales & Profit Distribution (Histogram)

```python
# Histogram for Profit
plt.figure(figsize=(10, 5))
sns.histplot(df['Profit'], bins=50, kde=True, color='green')
plt.title("Distribution of Profit")
plt.xlabel("Profit")
plt.ylabel("Count")
plt.show()
```



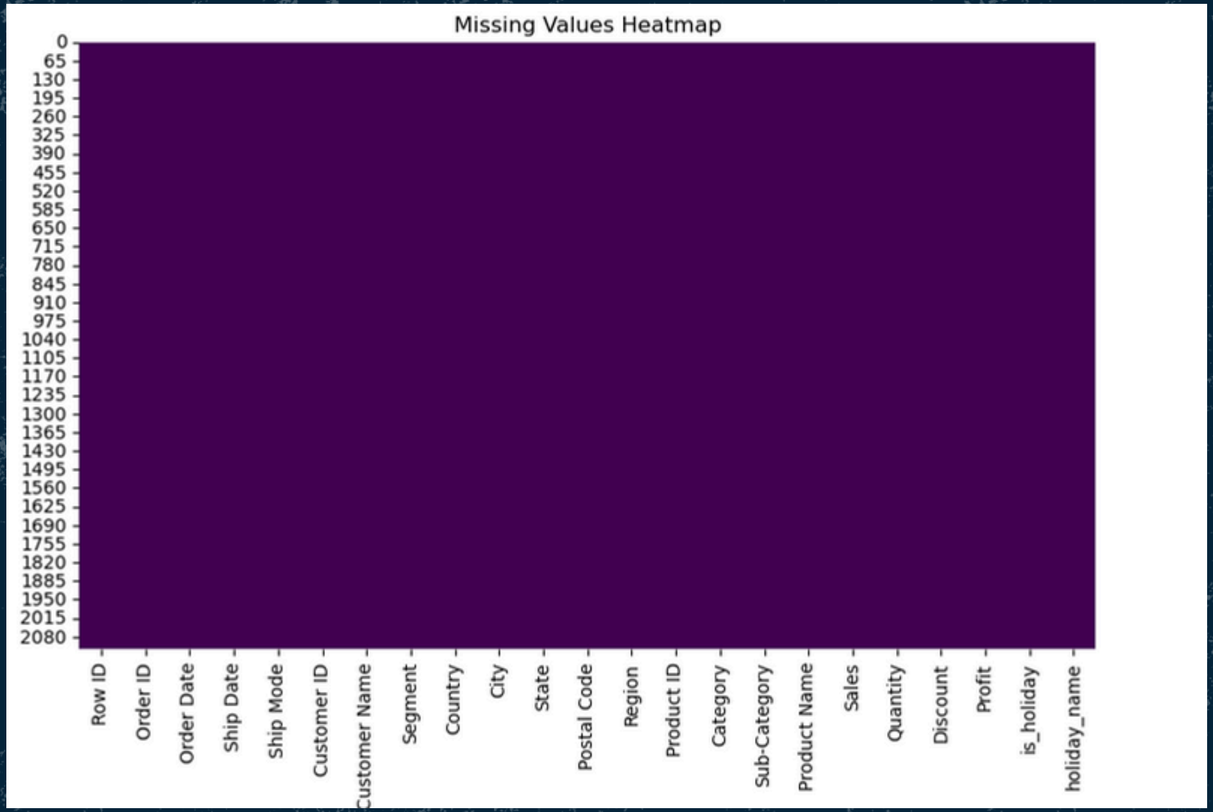
2️⃣ Sales vs Profit (Scatter Plot)

```
plt.figure(figsize=(10, 5))
sns.scatterplot(x=df['Sales'], y=df['Profit'], alpha=0.5, color='blue')
plt.title("Sales vs Profit")
plt.xlabel("Sales")
plt.ylabel("Profit")
plt.show()
```



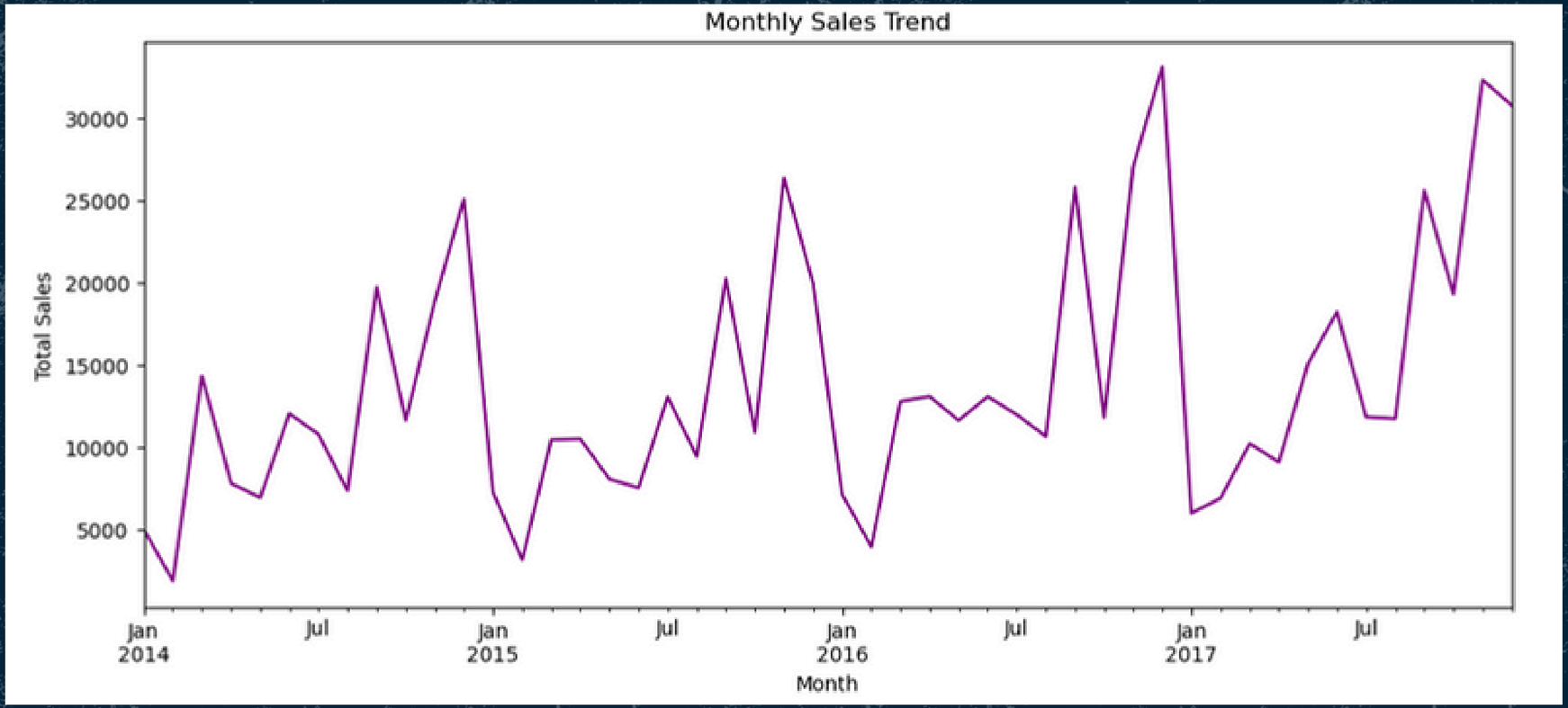
3️⃣ Missing Values Heatmap

```
import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 6))
sns.heatmap(df.isnull(), cmap="viridis", cbar=False)
plt.title("Missing Values Heatmap")
plt.show()
```



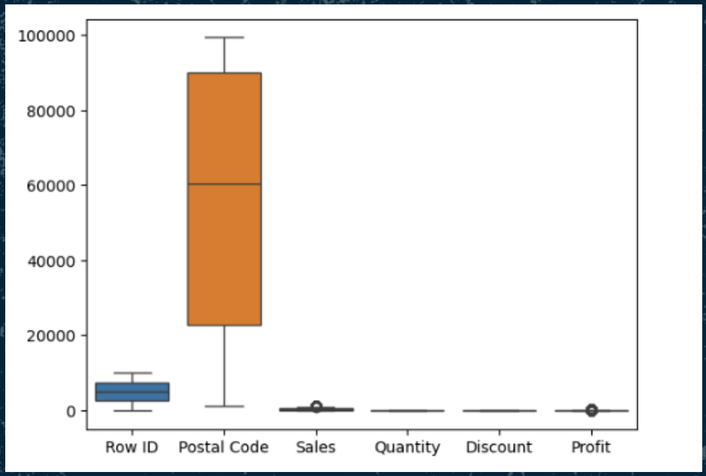
4️⃣ Sales Trend Over Time (Line Chart)

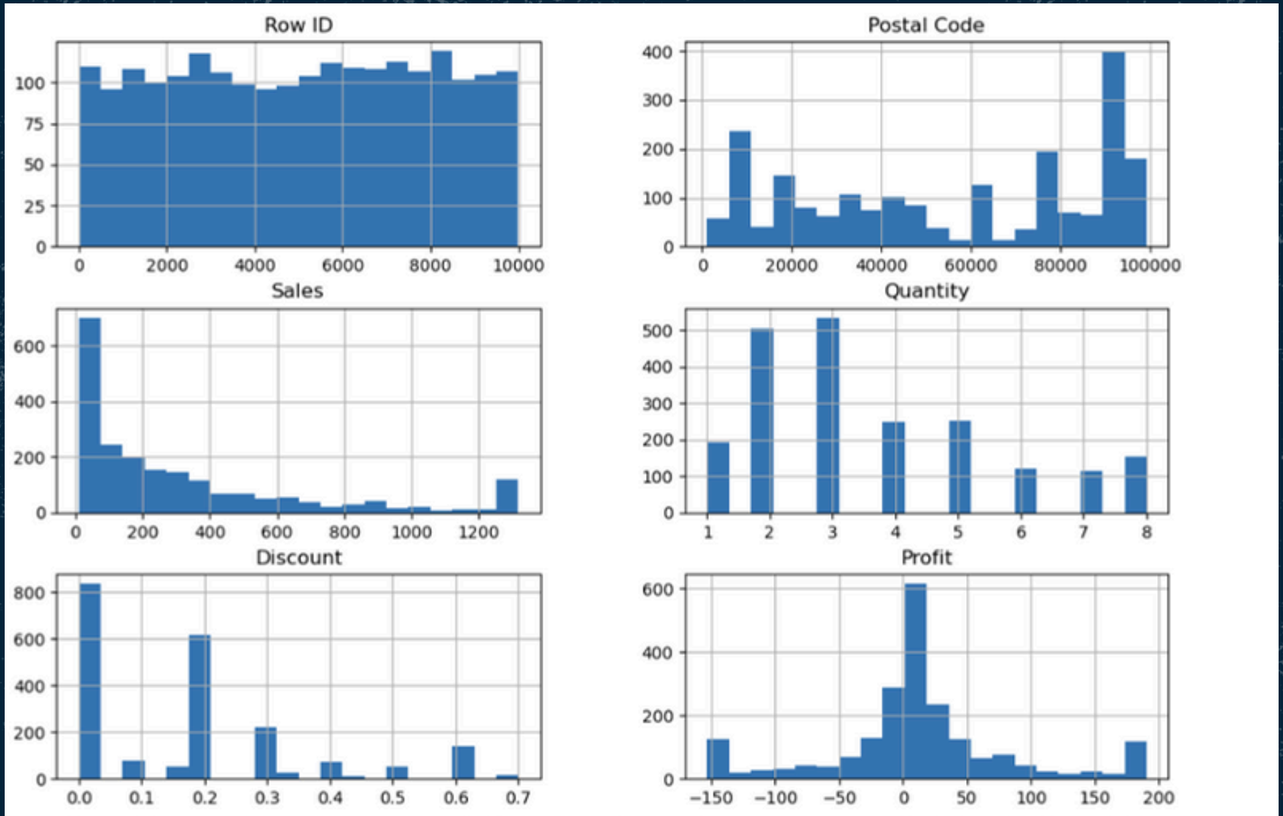
```
df['Order Date'] = pd.to_datetime(df['Order Date'])
df.set_index('Order Date', inplace=True)
df['Sales'].resample('M').sum().plot(figsize=(12, 5), color='purple')
plt.title("Monthly Sales Trend")
plt.xlabel("Month")
plt.ylabel("Total Sales")
plt.show()
```



5️⃣ Data Distribution

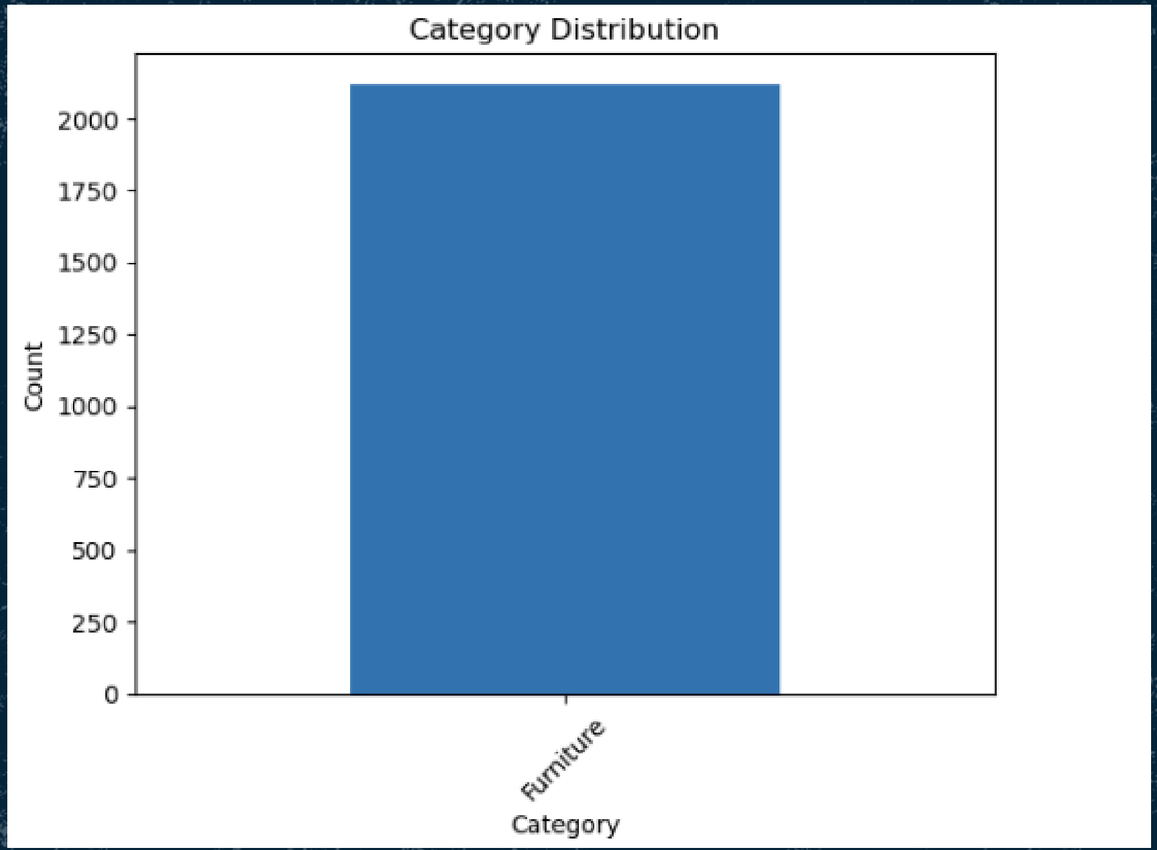
```
df.hist(figsize=(12, 8), bins=20)
plt.show()

# Boxplots to check for outliers
sns.boxplot(data=df.select_dtypes(include=['number']))
plt.show()
```





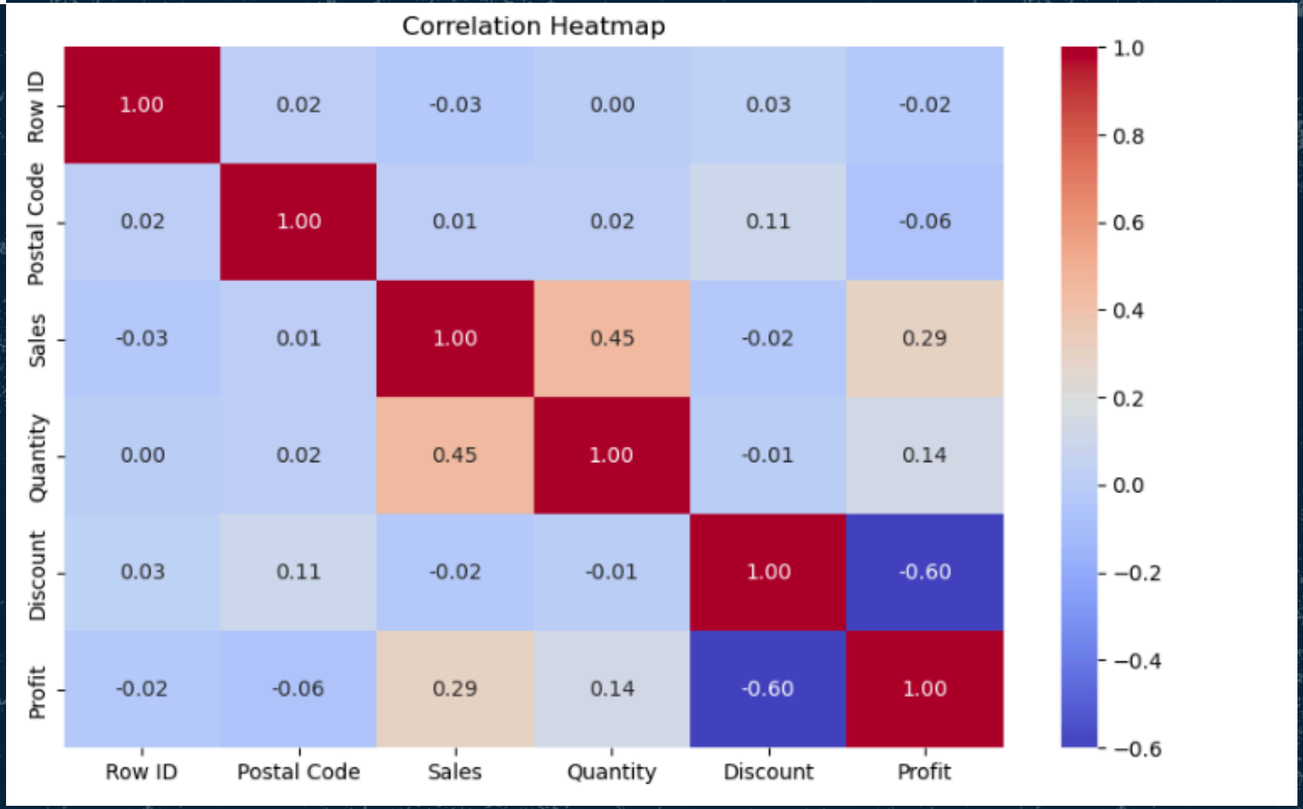
6 Category Distribution

```python
import matplotlib.pyplot as plt

df['Category'].value_counts().plot(kind='bar')
plt.title("Category Distribution")
plt.xlabel("Category")
plt.ylabel("Count")
plt.xticks(rotation=45)
plt.show()
```



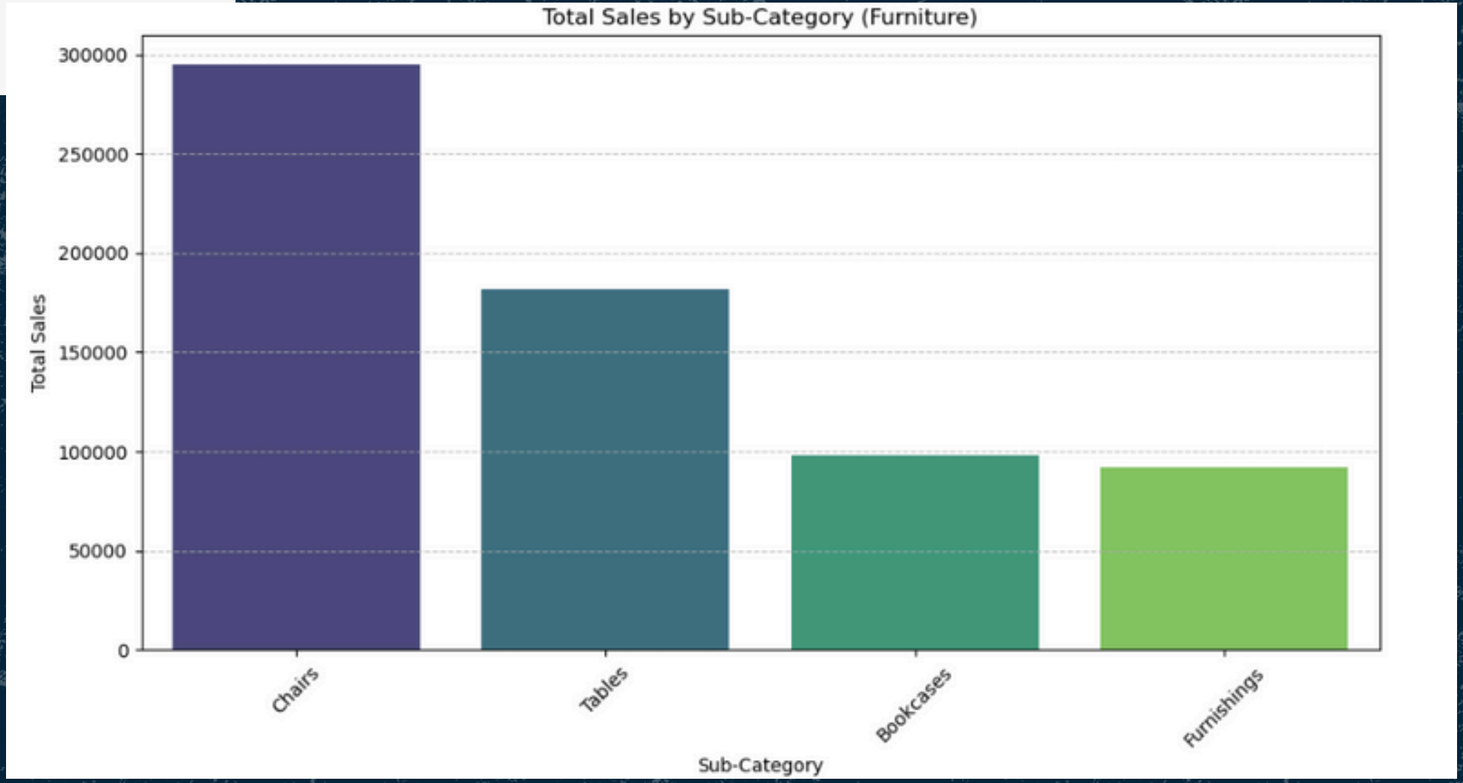
7️⃣ Correlation Heatmap

```python
import seaborn as sns
import matplotlib.pyplot as plt

# Select only numeric columns
numeric_df = df.select_dtypes(include=['number'])

# Plot the correlation heatmap
plt.figure(figsize=(10, 6))
sns.heatmap(numeric_df.corr(), annot=True, cmap="coolwarm", fmt=".2f")
plt.title("Correlation Heatmap")
plt.show()
```



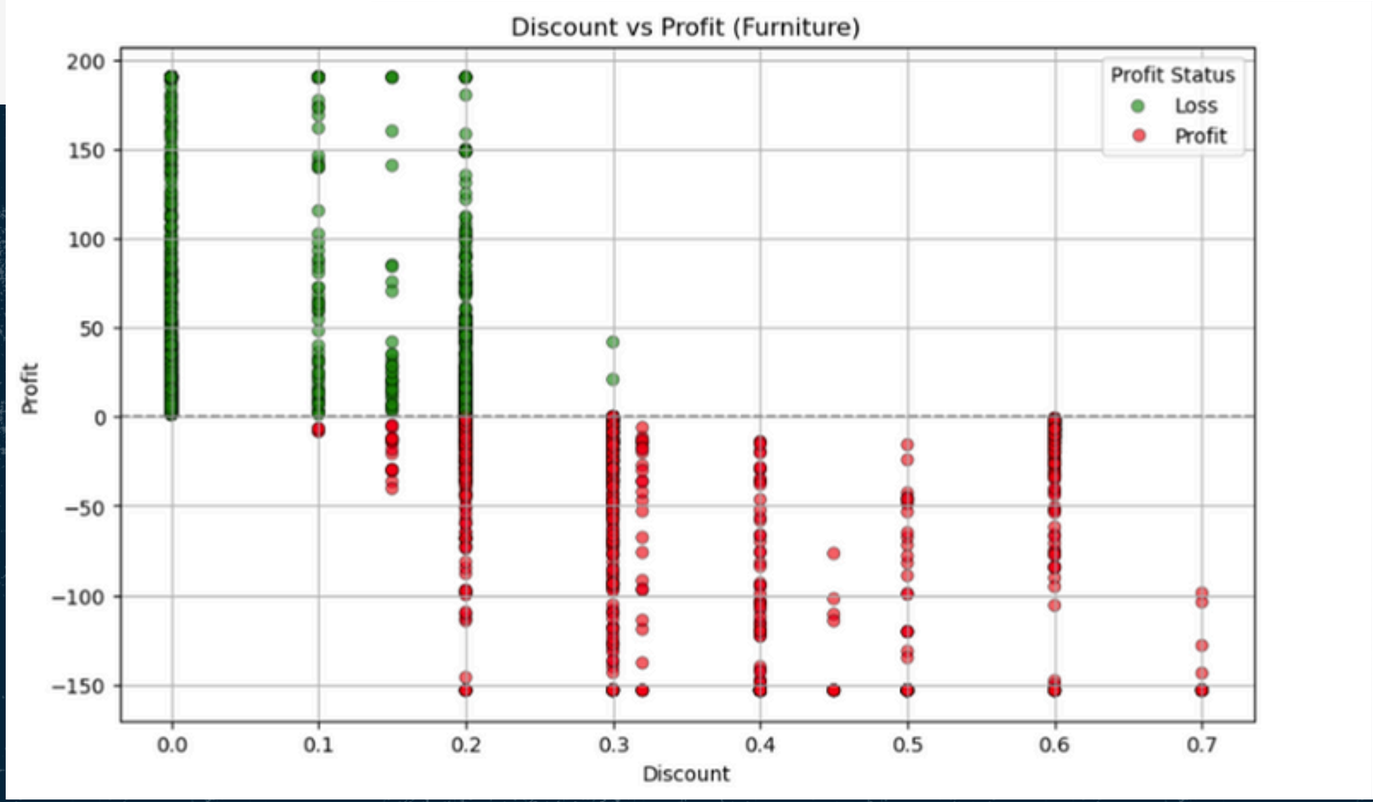
8️⃣ sales by sub-category

```python
# Aggregate sales by sub-category
sub_category_sales = df.groupby('Sub-Category', as_index=False)['Sales'].sum()

# Sort by sales
sub_category_sales = sub_category_sales.sort_values(by='Sales', ascending=False)

# Plot sales per sub-category
plt.figure(figsize=(12, 6))
sns.barplot(x='Sub-Category', y='Sales', data=sub_category_sales, palette='viridis')
plt.title('Total Sales by Sub-Category (Furniture)')
plt.xlabel('Sub-Category')
plt.ylabel('Total Sales')
plt.xticks(rotation=45)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```



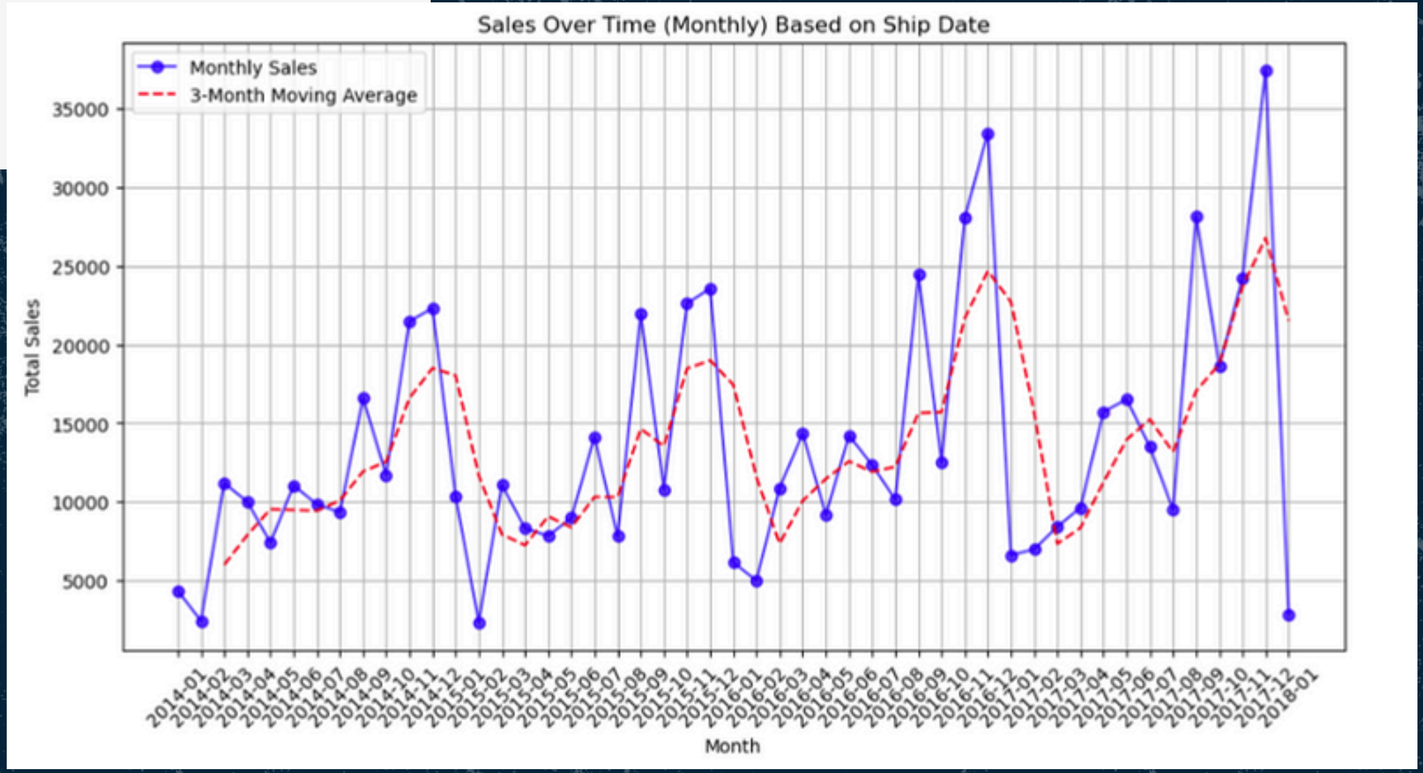
🔟 scatter plot for Discount vs Profit with color distinction

```python
# Create scatter plot for Discount vs Profit with color distinction
plt.figure(figsize=(10, 6))
sns.scatterplot(
    x=df['Discount'], y=df['Profit'], hue=df['Profit'] > 0, palette={True: 'green', False: 'red'},
    alpha=0.6, edgecolor='black'
)

# Add reference line at zero profit
plt.axhline(0, color='gray', linestyle='--', linewidth=1.2)

plt.title('Discount vs Profit (Furniture)')
plt.xlabel('Discount')
plt.ylabel('Profit')
plt.legend(title="Profit Status", labels=["Loss", "Profit"])
plt.grid(True)
plt.show()
```



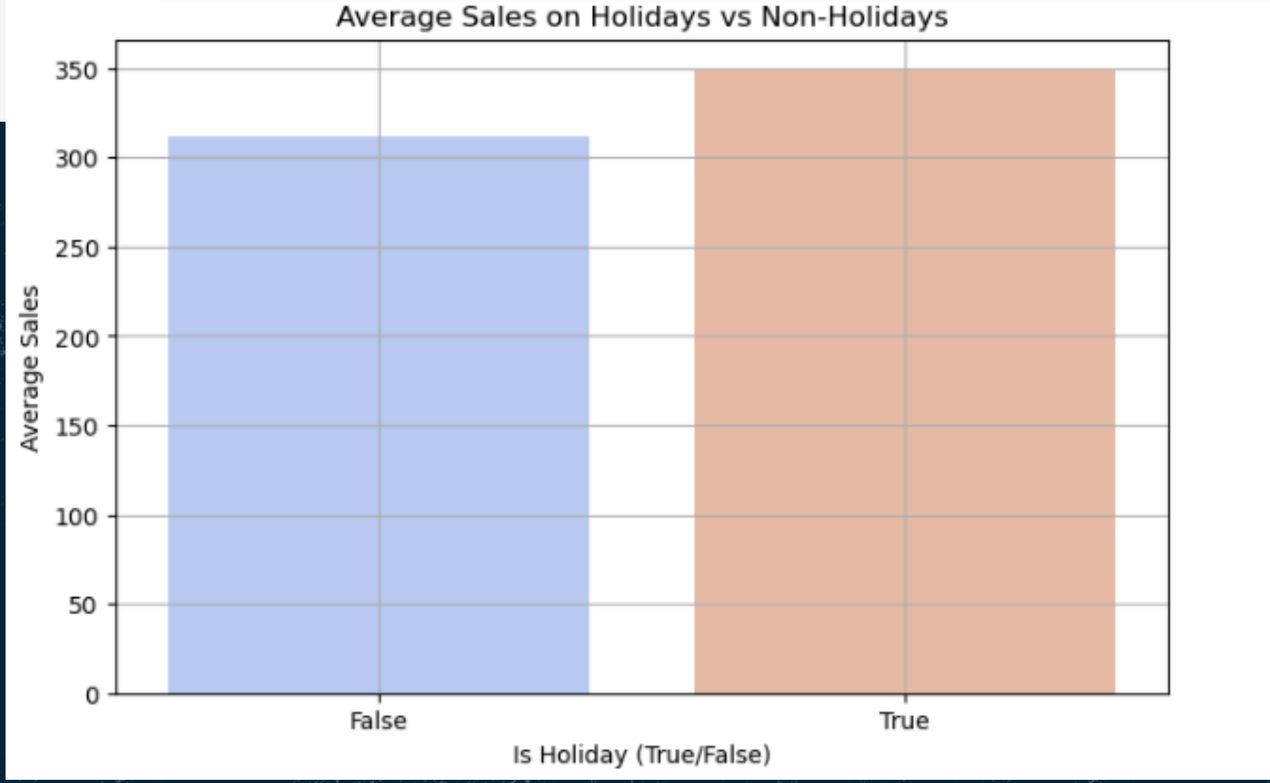
1️⃣ 1️⃣ sales by month based on Ship Date

```python
# Convert Ship Date to datetime
df['Ship Date'] = pd.to_datetime(df['Ship Date'])

# Aggregate sales by month based on Ship Date
df['Month'] = df['Ship Date'].dt.to_period('M')
monthly_sales = df.groupby('Month')['Sales'].sum()

# Plot Monthly Sales Trend
plt.figure(figsize=(12, 6))
plt.plot(monthly_sales.index.astype(str), monthly_sales, marker='o', linestyle='-', color='b', alpha=0.7, label='Monthly Sales')

# Add moving average to smooth the trend
rolling_avg = monthly_sales.rolling(window=3).mean()
plt.plot(monthly_sales.index.astype(str), rolling_avg, linestyle='--', color='r', alpha=0.9, label='3-Month Moving Average')

plt.title('Sales Over Time (Monthly) Based on Ship Date')
plt.xlabel('Month')
plt.ylabel('Total Sales')
plt.xticks(rotation=45)
plt.legend()
plt.grid(True)
plt.show()
```



1️⃣ 2️⃣ sales for holidays vs non-holidays

```
# Compute average sales for holidays vs non-holidays
avg_sales = df.groupby('is_holiday')['Sales'].mean().reset_index()

# Create a bar plot to compare average sales
plt.figure(figsize=(8, 5))
sns.barplot(x='is_holiday', y='Sales', data=avg_sales, palette='coolwarm')

plt.title('Average Sales on Holidays vs Non-Holidays')
plt.xlabel('Is Holiday (True/False)')
plt.ylabel('Average Sales')
plt.grid(True)
plt.show()
```



Key Insights

1. Heavy discounts negatively impact profit.
2. Chairs are the top-selling product.
3. Sales increase during holidays.
4. There is a clear seasonal pattern in sales.
5. Most orders have low value, but a small number of high-value orders exist.

Feature Creation

In this step, I prepared new features to help the model better understand the time-related patterns behind sales behavior

What I Did

Aggregated daily sales Grouped the data by order date and set the date as the index for time-based analysis

Extracted time-based features from Order Date:

Day of Week
Day of Month
Month (with Sine Transformation)
Day (with Sine Transformation)
Day of Year
Quarter
Year

Example of Feature Creation in Code

```python
def add_time_features(df):
    df = df.copy()

    df["Day of Week"] = df.index.dayofweek

    df["Month"] = df.index.month
    df["Month Sin"] = np.sin(df["Month"] * (2 * np.pi / 12))

    df["Day"] = df.index.day
    df["Day Sin"] = np.sin(df["Day"] * (2 * np.pi / 31))

    df["Day of Year"] = df.index.dayofyear
    df["Quarter"] = df.index.quarter
    df["Year"] = df.index.year
```

Added lag-based features:

Sales Day Before

Sales Day After

Marked special days:

Holiday

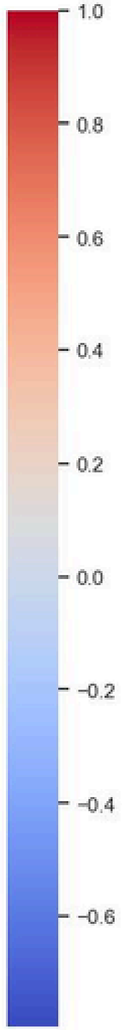
Weekend

Example of Feature Creation in Code

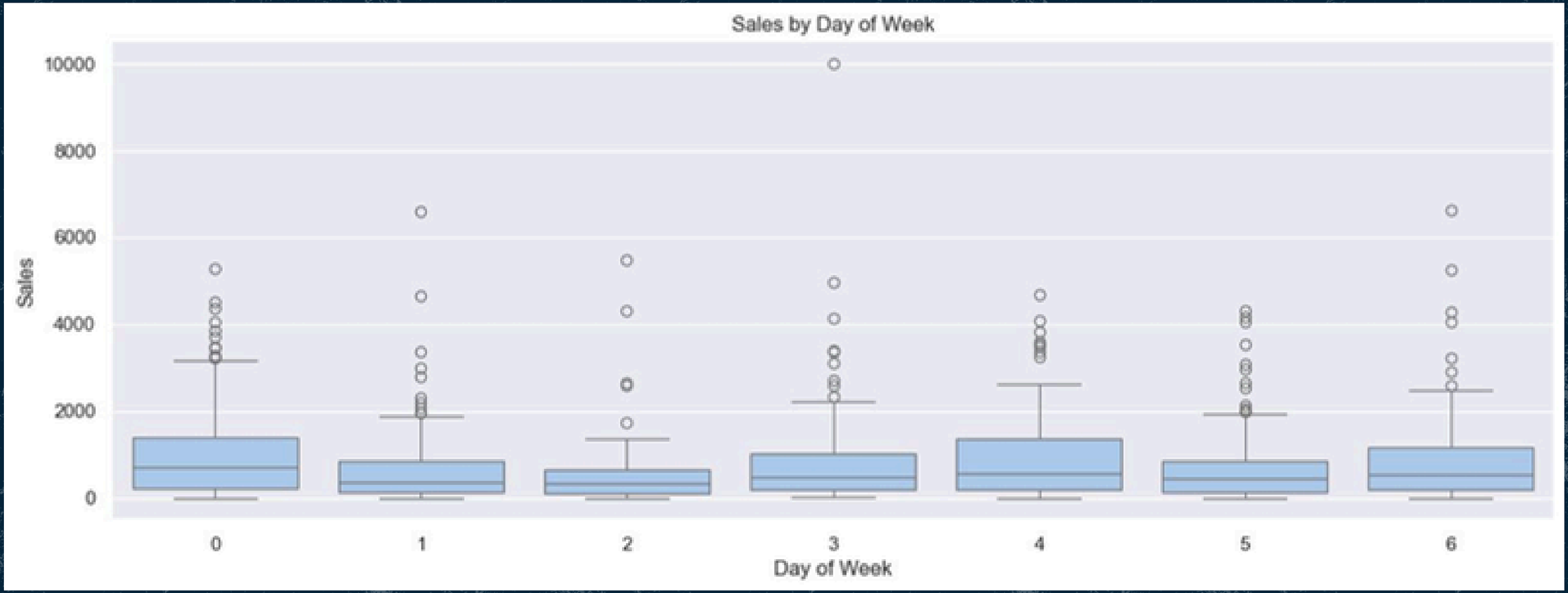
```
    df["Sales Day Before"] = df["Sales"].shift(1)
    df["Sales Day After"] = df["Sales"].shift(-1)

    return df


def add_free_days(df):
    df = df.copy()

    years = df.index.year.unique()
    us_holidays = holidays.UnitedStates(years=years)

    df["Holiday"] = df.index.isin(us_holidays).astype(int)
    df["Weekend"] = (df.index.dayofweek >= 5).astype(int)

    return df
```

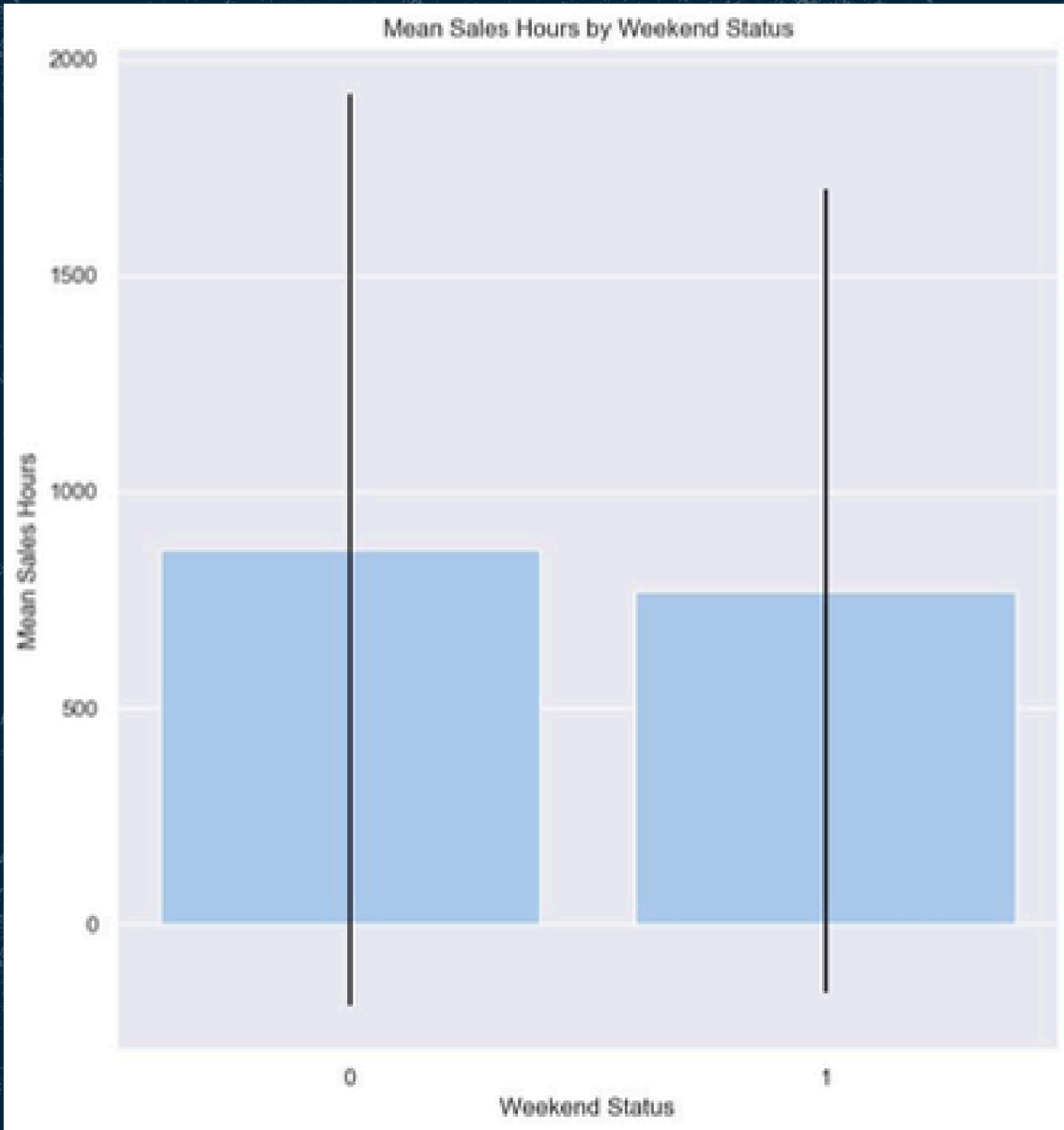
Feature / Target Relationship

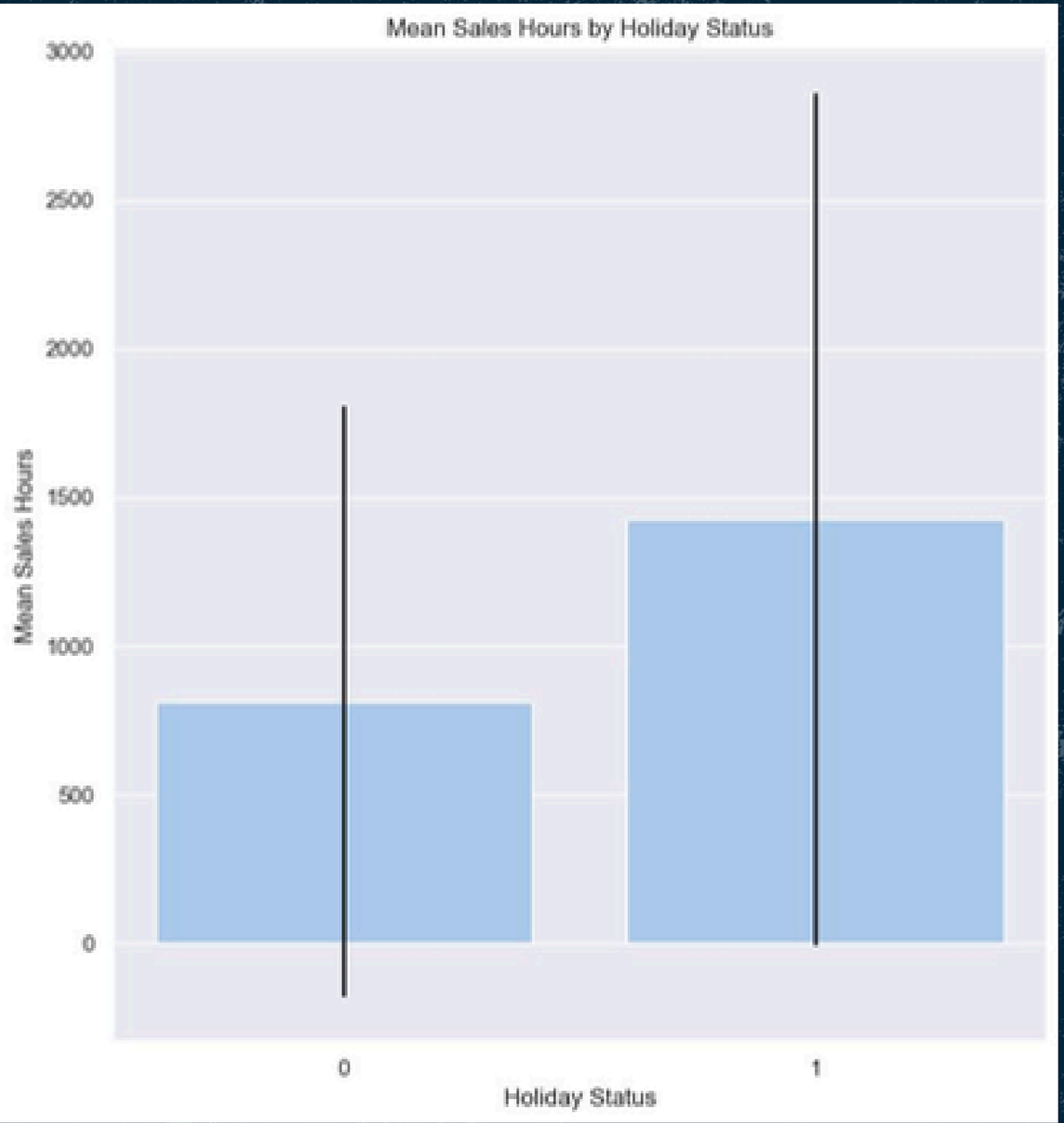
	Sales	Day of Week	Month	Month Sin	Day	Day Sin	Day of Year	Quarter	Year	Sales Day Before	Sales Day After	Holiday	Weekend
Sales	1	-0.038	0.22	-0.12	-0.011	-0.0052	0.22	0.2	0.015	0.012	0.012	0.11	-0.044
Day of Week	-0.038	1	-0.0058	0.011	-0.02	0.0065	-0.0075	0.0025	0.042	-0.023	0.055	-0.15	0.78
Month	0.22	-0.0058	1	-0.73	-0.012	-0.00039	1	0.97	-0.023	0.23	0.22	0.042	-0.016
Month Sin	-0.12	0.011	-0.73	1	0.01	-0.0086	-0.73	-0.75	0.037	-0.12	-0.14	-0.03	0.016
Day	-0.011	-0.02	-0.012	0.01	1	-0.79	0.072	-0.013	-0.01	-0.032	0.026	0.0059	-0.021
Day Sin	-0.0052	0.0065	-0.00039	-0.0086	-0.79	1	-0.068	0.0044	-0.006	0.0042	-0.011	-0.0089	-0.0019
Day of Year	0.22	-0.0075	1	-0.73	0.072	-0.068	1	0.97	-0.022	0.22	0.23	0.042	-0.017
Quarter	0.2	0.0025	0.97	-0.75	-0.013	0.0044	0.97	1	-0.018	0.19	0.2	0.054	-0.0042
Year	0.015	0.042	-0.023	0.037	-0.01	-0.006	-0.022	-0.018	1	0.016	0.019	0.032	0.015
Sales Day Before	0.012	-0.023	0.23	-0.12	-0.032	0.0042	0.22	0.19	0.016	1	0.014	-0.018	-0.01
Sales Day After	0.012	0.055	0.22	-0.14	0.026	-0.011	0.23	0.2	0.019	0.014	1	0.0056	0.046
Holiday	0.11	-0.15	0.042	-0.03	0.0059	-0.0089	0.042	0.054	0.032	-0.018	0.0056	1	-0.092
Weekend	-0.044	0.78	-0.016	0.016	-0.021	-0.0019	-0.017	-0.0042	0.015	-0.01	0.046	-0.092	1



Sales by Year
10000
8000
6000
4000
2000
0
Sales
2014
2015
2016
2017
Year


Sales by Day of Week
Sales
10000
8000
6000
4000
2000
0
0
1
2
3
4
5
6
Day of Week


Mean Sales Hours by Weekend Status
Mean Sales Hours
2000
1500
1000
500
0
0
1
Weekend Status


Mean Sales Hours by Holiday Status
Mean Sales Hours
3000
2500
2000
1500
1000
500
0
0
1
Holiday Status

start to Training Model

Sales Data Split

In this part, the dataset is divided into two parts based on a specific date. All the data before this date is used for training the model, while the data from this date onward is used for testing and validation. This approach helps to simulate real-world situations where the model learns from past information and is tested on future data it hasn't seen before.
After splitting, the sizes of both the training and validation sets are printed, showing how many samples and features each set contains. This helps confirm that the data was divided as intended.

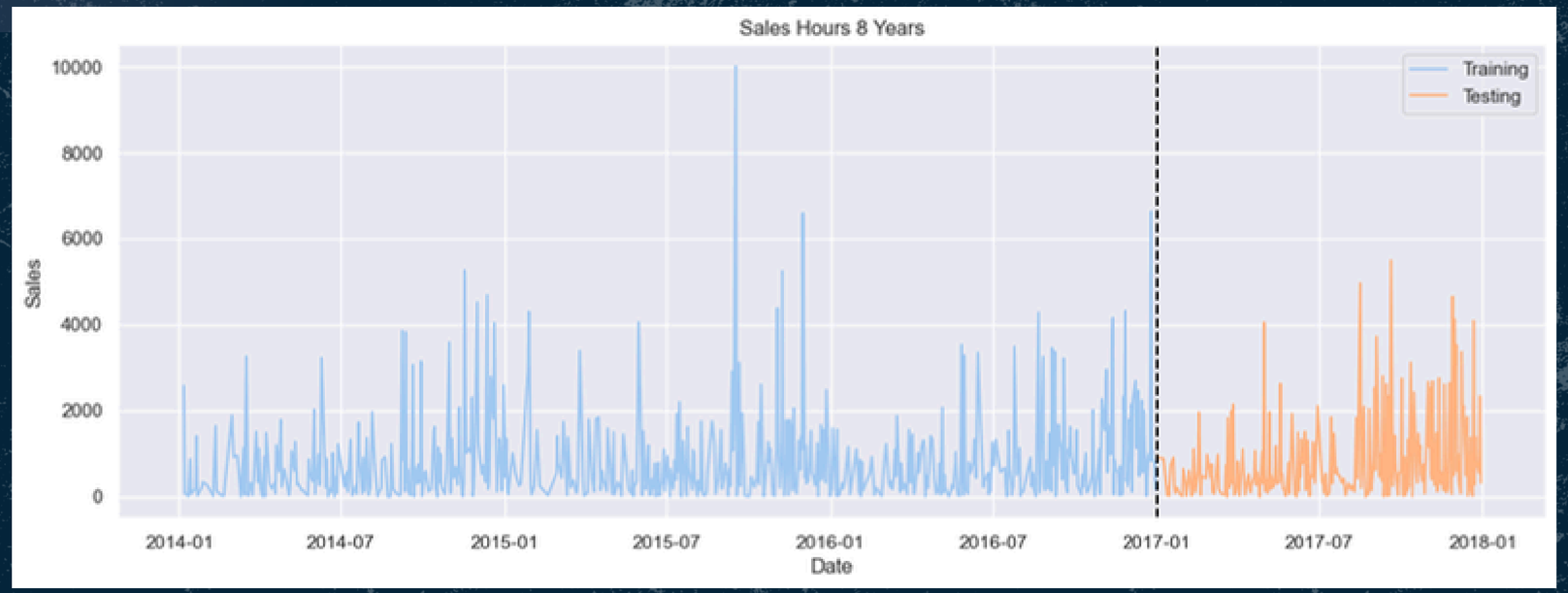
```
train_df = df.query("index < @split_year")
test_df = df.query("index >= @split_year")
```

```
print(f"Train set size: {train_df.shape}")
print(f"Validation set size: {test_df.shape}")
```

```
ain set size: (631, 13)
lidation set size: (258, 13)
```

Sales Data Split

The plot shows sales data over 8 years, split into training (blue) and testing (orange) sets by a vertical dashed line. Data before the line is used to train the model, and data after is for testing. This simulates real-life by training on past data and validating on future data. The shaded areas highlight the two sets clearly.



Model Training and Hyperparameter Tuning

Here, the features and target columns are selected from the training data. The data is split into training and test sets accordingly.
An XGBoost regression model is created with a fixed random seed. A range of hyperparameters like number of trees, learning rate, and tree depth is defined.
Bayesian Search with 5-fold cross-validation is used to find the best combination of these parameters efficiently.
The model is then trained on the training data and evaluated on the test data to optimize its performance for better predictions on new data.

```
FEATURES = train_df.columns[1:]
TARGET = "Sales"

train_features, train_target = train_df[FEATURES], train_df[TARGET]
X_test, y_test = test_df[FEATURES], test_df[TARGET]
```

[32]

Train the Model

```
model = XGBRegressor(random_state=42)

params = {
    "n_estimators": Integer(2000, 3000),
    "early_stopping_rounds": Integer(5, 15),
    "learning_rate": Real(0.01, 0.1),
    "max_depth": Integer(2, 5),
}

bayes_search = BayesSearchCV(
    estimator=model, search_spaces=params, cv=5, n_jobs=-1, random_state=42
)
```

[33]

```
%%time
bayes_search.fit(train_features, train_target,
                 eval_set=[(X_test, y_test)],
                 verbose=None);
```

[34]

```
CPU times: user 32.7 s, sys: 1.69 s, total: 34.4 s
Wall time: 44.1 s
```

Model Evaluation Results

This part shows the best model found after tuning its settings. The chosen parameters like learning rate, tree depth, and number of trees are listed. The model is tested on new data, and three error measures are calculated: RMSE, MAE, and MAPE. These numbers show how close the predictions are to the real values. Lower values mean better performance.

```
best_model = bayes_search.best_estimator_
best_params = bayes_search.best_params_
best_params
```

[35]

```
OrderedDict([('early_stopping_rounds', 7),
             ('learning_rate', 0.05965543339494371),
             ('max_depth', 3),
             ('n_estimators', 2418)])
```

Evaluate the Model

```
y_pred = best_model.predict(X_test)

rmse = np.sqrt(mean_squared_error(y_test, y_pred))
mae = mean_absolute_error(y_test, y_pred)
mape = np.mean(np.abs((y_test - y_pred) / y_test))

print(f"RMSE: {rmse:.2f}")
print(f"MAE: {mae:.2f}")
print(f"MAPE: {mape:.2%}")
```

[36]

```
RMSE: 937.20
MAE: 663.74
MAPE: 860.99%
```

Feature Importance Visualization and Analysis

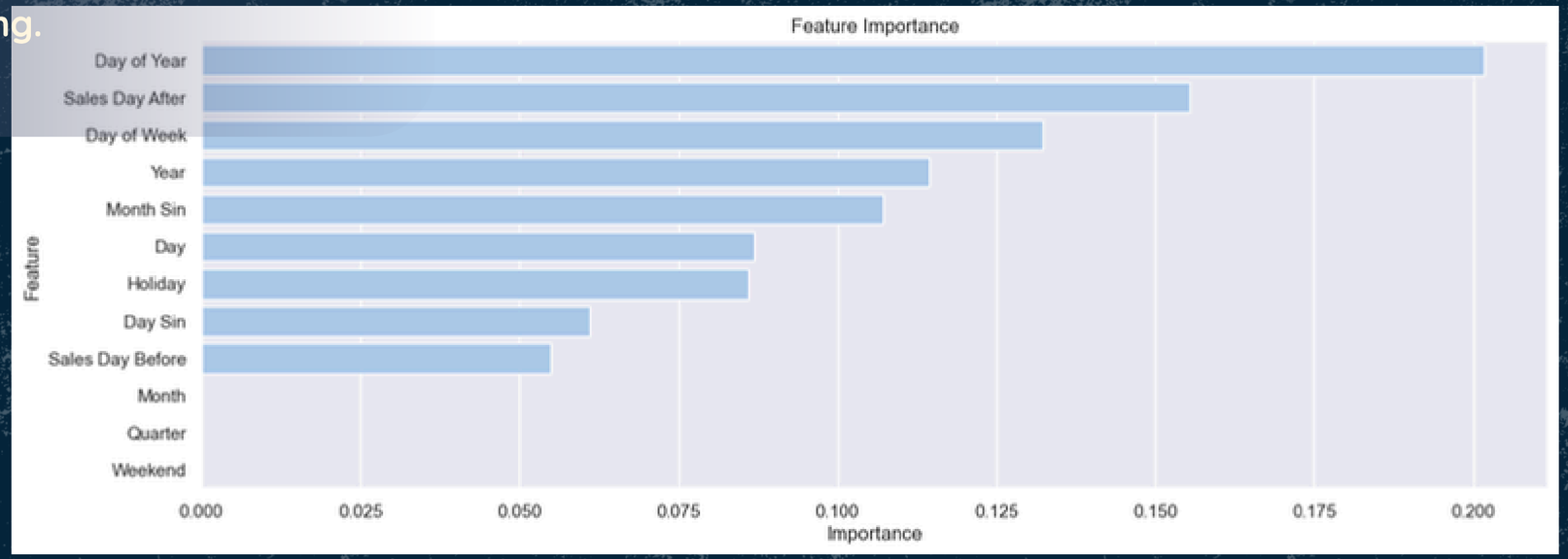
This part of the process involves analyzing which features of the dataset have the most influence on the model's predictions. The importance scores of the features are extracted from the best-performing model and organized into a structured format for better readability.
A bar plot is then created to visually represent these importance scores, showing the relative significance of each feature in descending order. The plot is labeled clearly with titles and axis names to ensure it is easy to interpret.
This analysis helps in understanding which features the model relies on most, providing insights that can be valuable for feature engineering, model refinement, or domain understanding.

```
importances = best_model.feature_importances_
features = best_model.feature_names_in_

features_df = pd.DataFrame(
    {"Feature": features, "Importance": importances}
).sort_values("Importance", ascending=False)
```

[37]

```
plt.figure(figsize=(15, 5))
sns.barplot(features_df, x="Importance", y="Feature")

plt.title("Feature Importance")
plt.xlabel("Importance")
plt.ylabel("Feature")

plt.show()
```

[38]



How the model works:

The model uses XGBoost, which builds decision trees to predict sales. Each tree corrects the errors of the previous one. The final prediction is made by combining all tree outputs. This approach helps the model handle complex data and make accurate predictions.

(Optimization Techniques)

We applied Bayesian Optimization using BayesSearchCV to efficiently search for the optimal hyperparameters. Additionally, we utilized 5-Fold Cross-Validation to ensure the model's performance is consistent and not prone to overfitting or random fluctuations.

Summary of what was explained:

1

Collected and cleaned sales data
Handled missing values and created time-based features like day, month, and holidays.

2

Explored the data visually
Used charts to understand sales trends, category performance, and discount impact.

3

Trained an XGBoost model
Tuned it with Bayesian Optimization and evaluated using MAE & MSE

4

Built a simple forecasting app
Using Streamlit to let users input dates and get instant sales predictions.

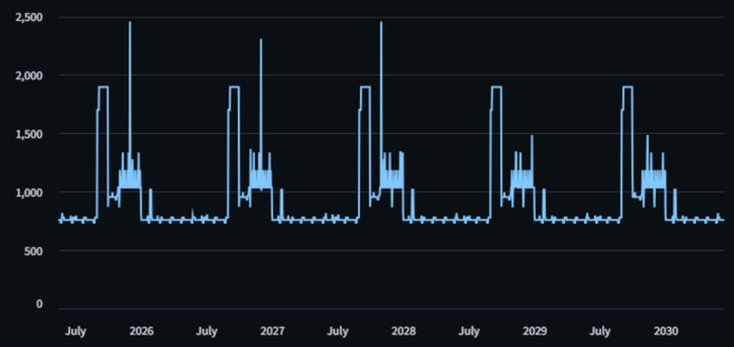
Model deployment

Single Date Prediction

Predictions for Date Range

Predictions for Date Range

	Prediction
2025-06-01 00:00:00	757.1524
2025-06-02 00:00:00	757.1524
2025-06-03 00:00:00	757.1524
2025-06-04 00:00:00	757.1524
2025-06-05 00:00:00	757.1524
2025-06-06 00:00:00	757.1524
2025-06-07 00:00:00	757.1524
2025-06-08 00:00:00	757.1524
2025-06-09 00:00:00	757.1524
2025-06-10 00:00:00	757.1524
2025-06-11 00:00:00	757.1524



Postprocessing: Time features

This part shows the time-based features added to the dataset in the add_time_features function. The features include the Day of the Week, Month, Day, Day of the Year, Quarter, Year, and Sales Day Before/After. The Month and Day features are transformed using sine functions to capture cyclical patterns. These time-related features help the model recognize seasonal, weekly, and yearly trends, improving the accuracy of predictions, particularly for sales forecasting.

```python
def add_time_features(df):
    df = df.copy()

    df["Day of Week"] = df.index.dayofweek

    df["Month"] = df.index.month
    df["Month Sin"] = np.sin(df["Month"] * (2 * np.pi / 12))

    df["Day"] = df.index.day
    df["Day Sin"] = np.sin(df["Day"] * (2 * np.pi / 31))

    df["Day of Year"] = df.index.dayofyear
    df["Quarter"] = df.index.quarter
    df["Year"] = df.index.year

    if "Sales" in df.columns and len(df) > 1:
        df["Sales Day Before"] = df["Sales"].shift(1).fillna(0)
        df["Sales Day After"] = df["Sales"].shift(-1).fillna(0)
    else:
        df["Sales Day Before"] = 0
        df["Sales Day After"] = 0

    return df
```

Postprocessing: Free Days

This part shows the features added by the add_free_days function to account for holidays and weekends. The function creates a Holiday feature to indicate whether a date is a holiday, based on the United States holiday calendar, and a Weekend feature to mark whether a date falls on a weekend. These features help the model capture the impact of non-regular patterns, like holidays and weekends, on sales or other business metrics.

```python
def add_free_days(df):
    df = df.copy()

    years = df.index.year.unique()
    us_holidays = holidays.UnitedStates(years=years)
    holiday_dates = pd.to_datetime(list(us_holidays.keys()))

    df["Holiday"] = df.index.isin(holiday_dates).astype(int)
    df["Weekend"] = (df.index.dayofweek >= 5).astype(int)

    return df
```

Streamlit Tabs: Single Data Prediction

The Streamlit app includes two main tabs: Single Date Prediction and Date Range Prediction.
Single Date Prediction: This tab allows users to select a specific date and get a sales prediction for that day. The app generates the necessary time-based features and free day features, then uses the trained model to make the prediction.

```python
with tabs[0]:
    st.header("Single Date Prediction")

    selected_date = st.date_input("Select a date", value=pd.to_datetime("today"))

    df_single = pd.DataFrame(index=[pd.to_datetime(selected_date)])
    df_single = add_time_features(df_single)
    df_single = add_free_days(df_single)

    if st.button("Predict", key="single"):
        X_single = df_single[feature_cols]
        prediction = model.predict(X_single)[0]

        st.subheader("Predicted Sales")
        st.write(f"For {selected_date}, the predicted sales is: **{prediction:.2f}**")

with tabs[1]:
    st.header("Date Range Prediction")

    start_date = st.date_input("Start Date", value=pd.to_datetime("today"))
    end_date = st.date_input(
        "End Date", value=pd.to_datetime("today") + pd.DateOffset(days=30)
    )
```

Streamlit Tabs: Date Range Prediction

The Streamlit app includes two main tabs: Single Date Prediction and Date Range Prediction.
Date Range Prediction: This tab enables users to choose a start and end date for a date range and get predictions for each day within that range. The app generates predictions for the entire range, displays them in a table, and also visualizes the predictions through a line chart.

```python
with tabs[0]:
    st.header("Single Date Prediction")

    selected_date = st.date_input("Select a date", value=pd.to_datetime("today"))

    df_single = pd.DataFrame(index=[pd.to_datetime(selected_date)])
    df_single = add_time_features(df_single)
    df_single = add_free_days(df_single)

    if st.button("Predict", key="single"):
        X_single = df_single[feature_cols]
        prediction = model.predict(X_single)[0]

        st.subheader("Predicted Sales")
        st.write(f"For {selected_date}, the predicted sales is: **{prediction:.2f}**")

with tabs[1]:
    st.header("Date Range Prediction")

    start_date = st.date_input("Start Date", value=pd.to_datetime("today"))
    end_date = st.date_input(
        "End Date", value=pd.to_datetime("today") + pd.DateOffset(days=30)
    )
```

The End

THANK YOU