# Forecasting Sales for Rossmann Stores

ADS1002 Tuesday Group 3

| **Name** | **Monash ID** | **Contribution** | **Description** |
|---|---|---|---|
| Chung Ling Lai | 33067856 | 39% | EDA, Research, Modelling, PowerPoint, Prediction, Report, Presentation |
| Dowon Yi | 33394326 | 39% | Data Wrangling, EDA, Research, Modelling, Prediction, Presentation, Report |
| Simin Liang | 33499179 | 18% | PowerPoint, Time Series Analysis, Report, Presentation |
| Ruchir Mandar Kajrolkar | 33045917 | 3% | Presentation |
| Jiazhe Han | 33072493 | 1% | Presentation |

## Contents

# Introduction

The project centres on a comprehensive analysis of a dataset encompassing 1,115 Rossmann drug stores across 7 European countries spanning the years from 2013 to 2015. This extensive dataset comprises a staggering 1,017,209 rows of daily records, providing a rich source of information for our investigation.

The dataset features a variety of critical variables such as Customers, Sales, Day of the Week, and information about whether the store is open on a particular day, Promotions, Competition, as well as data on school and state holidays. The dataset also provides insight into specific attributes of the stores, detailing their store type and assortment offerings. Here are the detailed descriptions for each column:

- Sales - Revenue for any given day (the target variable we aim to predict).
- Store - A unique ID for each store.
- DayOfWeek - Represents the day of the week, from 1 (Monday) to 7 (Sunday).
- Customers - The number of customers on a given day.
- Open - Indicates whether the store was open: 0 means closed, 1 means open.
- Promo - Indicates if a store is running a promotion on that day: 0 = no promo, 1 = promo.
- StateHoliday - Indicates a state holiday. Most stores are closed on state holidays, with a few exceptions. Note: all schools are closed on public holidays and weekends. Values: a = public holiday, b = Easter holiday, c = Christmas, 0 = None.
- SchoolHoliday – Indicates closure of public schools: 0 = not a school holiday, 1 = school holiday.
- StoreType - Differentiates between four store models: a, b, c, d.
- Assortment - Describes the assortment level: a = basic, b = extra, c = extended.
- CompetitionDistance - Distance to the nearest competitor store.
- CompetitionOpenSince[Month/Year] - Indicates the approximate year and month the nearest competitor opened.
- Promo2 - Continuous and consecutive promotion for some stores: 0 = no participation, 1 = participation.

- Promo2Since[Year/Week] - Indicates the year and calendar week when the store started participating in Promo2.
- PromoInterval - Describes the intervals when Promo2 restarts, listing the months the promotion begins again. For example, "Feb, May, Aug, Nov" means each round starts in February, May, August, November of any given year for that store.

The primary goals of this project are twofold: firstly, to derive actionable insights from the data to boost sales, and secondly, to create a predictive model for forecasting sales. In our pursuit of these objectives, we will utilise sophisticated data analysis methods, including machine learning, to deliver valuable findings and concrete recommendations for Rossmann. This report will methodically outline our processes in data preparation, analysis, and modelling.

# Preprocessing

Prior to loading the data into Python, it was essential to ensure the data was well-organised, curated, and saved in a structured format such as CSV. Once confirmed, we then imported the necessary Python libraries for subsequent processing.

Library Importation: To support the extensive data preprocessing and analysis requirements of the project, we began by importing a suite of essential Python libraries:

- Basic Libraries:
  - numpy: Offers support for arrays (including mathematical operations).
  - pandas: Provides structures for data manipulation and analysis.

- Visualisation Libraries:
  - seaborn & matplotlib: Allows for sophisticated data visualisations.

- Machine Learning Libraries:
  - From sklearn, we imported:
    - Regression models: LinearRegression, Ridge, DecisionTreeRegressor, and RandomForestRegressor.
  - Model selection tools: train_test_split.
  - Metrics: r2_score, mean_squared_error, mean_absolute_error.

- Warning Management:
  - \`warnings\`: Used to suppress any warning messages to keep our outputs clean.

In conclusion, data preprocessing plays an indispensable role in data analysis projects. Through meticulous data preprocessing, we ensured that the datasets were in the appropriate format. With the datasets in order, we imported the necessary Python libraries, paving the way for further exploratory data analysis and modelling

# Data Wrangling

In the Data Wrangling phase, we loaded two datasets: the train dataset and the store dataset. The pivotal step in this phase was merging the train dataset with the store dataset based on the store number using the 'inner' merge method. This ensured a seamless alignment of data from both sources, creating a unified dataset for subsequent analysis.

After creating a unified dataset, we turned our attention to the investigation of missing values. Our initial focus was on the “CompetitionDistance” column, and we identified three stores with missing values. Notably, these stores also lacked entries for "CompetitionOpenSinceMonth" and "CompetitionOpenSinceYear". Such consistent omissions suggest they might not be merely accidental. There's a possibility that Rossmann chose not to document competitors located beyond 75,860 metres, deeming those distances as placing them in a different economic zone with negligible impact on sales. Given the rarity of these missing values and the underlying complexities, we refrained from imputing the competition distance values. This stance is bolstered by the fact that the distribution of Rossmann stores is shaped by the number and size of cities across countries. Such a distribution inherently influences sales, underscoring the need for caution and making straightforward imputation ill-advised.

We discovered seemingly anomalous values for "CompetitionOpenSinceYear" — specifically, 1900 and 1961. Although Rossmann was founded in 1972, we have records indicating competitors opening in 1900 and 1961. While these data points might be considered outliers or mistaken entries, it's also possible that competing stores opened prior to Rossmann's establishment, and Rossmann documented these later upon investigation. In the absence of definitive information to warrant their exclusion, we opted to retain these data points.

We confirmed that the missing values for "CompetitionOpenSinceMonth" and "CompetitionOpenSinceYear" occur concurrently. The most likely explanations for the absence of these two data points are that the competing store predated the establishment of the Rossmann store, or there was a lapse in data recording. While these datasets contain some missing columns, they also offer valuable information in other columns. As such, we've chosen to retain them.

For "Promotion2SinceWeek" and "Promotion2SinceYear," missing values were directly related to the absence of Promotion 2. Therefore, these missing values were retained, as they held significance in indicating the absence of this promotion.

Notably, for other columns, no missing values were identified, ensuring the completeness and integrity of the dataset.

# Exploratory Data Analysis

In the Data Analysis section, we thoroughly examined the dataset to uncover key insights. Initially, we conducted Exploratory Data Analysis on two subsets: the top 20% of sales data and the overall sales data. This approach was intended to identify the factors that amplified sales and to offer valuable guidance to store managers.

Notably, the dataset for the top 20% sales indicates that "customers" and "sales" exhibit the strongest correlation, with a coefficient of 0.71, highlighting their significant relationship. However, the other features do not exhibit strong correlations with one another. The analysis of sales data reveals that the mean sales figure stands at 11209.83, with a standard deviation of 2908.04.

From our analysis, stores with an 'extra' assortment size (Figure 3.2) led in sales, and those classified as 'type b' stores (Figure 3.3) closely followed. While individual promotions (Figure 3.4) significantly boosted sales, there was a noticeable decrease in sales when consecutive and continuous promotions (Promotion 2) were implemented (Figure 3.5). This clearly indicates that overwhelming customers with ongoing promotions can have a counterproductive effect on sales.

Additionally, the data indicates that sales tend to increase during school holidays (Figure 3.6), suggesting a potential opportunity for strategic marketing. Interestingly, in the absence of state holidays (Figure 3.7), there were generally higher sales figures.

Analysis of competition distance (Figure 3.8) reveals that shorter distances were associated with higher sales, aligning with the phenomenon of competitive businesses often clustering together[1] (Talwalkar, 2012).

The time series analysis, spanning Figures 3.9 to 3.14, unveiled remarkably consistent trends. Notably, it revealed an uptrend in sales while the number of customers dropped significantly (Figure 3.9). Sales consistently peaked during the fourth quarter, primarily attributed to the holiday season, which includes notable events like Thanksgiving in November and Christmas in December (Figure 3.10). Additionally, December

[1] Presh Talwalkar (October 23, 2012) Fast food location game theory

consistently registered as the month with the highest sales (Figure 3.11), with a noticeable peak in sales every two weeks (Figure 3.12). Moreover, opening stores on Sundays resulted in a significant surge in sales (Figure 3.13, Figure 3.14). These insights offer valuable information on sales patterns, guiding informed decision-making for Rossmann, especially concerning promotions and store operations.

An analysis of the start dates of competitions, by year and month (Figures 3.15 and 3.16), showed a consistent trend irrespective of the time frame, suggesting that the onset of competitions did not markedly influence sales. Such insights are instrumental for Rossmann when formulating strategies concerning competitors.

# Modelling

In the Modelling phase, we adopted a systematic approach to build and assess sales prediction models. Initially, we focused on the top 20% of sales records for a more targeted analysis. The 'DayOfWeekName' column was discarded as redundant. Our Exploratory Data Analysis (EDA) confirmed that 'CompetitionOpenSinceYear' and 'CompetitionOpenSinceMonth' didn't significantly influence sales, so they were removed. Furthermore, the 'Store' column, merely identifying stores uniquely, was deemed irrelevant for sales forecasting and was thus excluded.

To handle categorical data, one-hot encoding was applied to the columns 'DayOfWeek,' 'StateHoliday,' 'StoreType,' and 'Assortment,' making them suitable for regression modelling. The dataset was then divided into training data, comprising 75% of the records, and testing data, containing the remaining 25%. This segregation was essential for effective model training and evaluation.

We utilised two variations of the top 20% sales data: one included the 'Customers' feature, while the other omitted it to minimise dependency on the sales-customer correlation. For each dataset, we developed three models: multivariate regression, normalised multivariate regression, decision tree regression, and random forest regression, resulting in a total of eight models.

In Figure 4.0, the Multivariate Linear Regression model with the 'Customers' variable yielded training and testing R-squared scores of 0.7008 and 0.6989, respectively. The Normalised Linear Regression posted scores of 0.7133 (training) and 0.7127 (testing). While the Decision Tree Regressor showcased an outstanding training score of 0.9978, its testing score settled at 0.8310. However, the Random Forest Regressor outshone the others with scores of 0.9850 (training) and 0.9015 (testing). Notably, even without the 'Customers' variable, the Random Forest model remained impressive. Given its highest testing R-squared score, the Random Forest Regressor was chosen as the optimal model for sales prediction.

To illustrate the model's practical application, consider the following scenario: on a given day with 3443 customers, no ongoing promotions, neither state holiday nor school holiday, a competition distance of 840m, a Friday, an 'extra' assortment, and store type 'b,' the model predicts sales amounting to $11,120.

The model was subsequently applied to predict sales for May of 2015. Figure 4.1 demonstrates the magnitudes of predicted sales compared to actual sales, affirming the model's effectiveness. The mean relative error between actual and predicted sales was calculated at 0.04, with a standard deviation of 6.73, demonstrating the model's precision. The magnitude of error remained below 3.84%, further attesting to the model's reliability. Figure 4.2 provides insights into the distribution of relative errors, and Figure 4.3 presents a box and whisker plot of these errors, further validating the model's performance. These findings serve as a solid foundation for sales predictions and informed decision-making for Rossmann.

# Conclusion

In summary, this data science project focused on leveraging data analysis and predictive modelling to drive recommendations for increasing sales for Rossmann. Based on our data analysis, key factors that appear to significantly impact sales include offering an 'extra' assortment, belonging to store type 'b', employing a single promotion, scheduling promotions when the sales are low, and reducing competition distances.

To facilitate these recommendations, we constructed a Random Forest Regressor, which emerged as the most reliable model for predicting sales. By applying this model to real-world scenarios, we can better understand and anticipate the sales outcomes, enabling Rossmann to make informed and strategic decisions aimed at boosting revenue.

This project underscores the power of data-driven insights and predictive models in shaping the future of sales strategies for Rossmann, offering a valuable tool for enhancing overall business performance.

# Figures

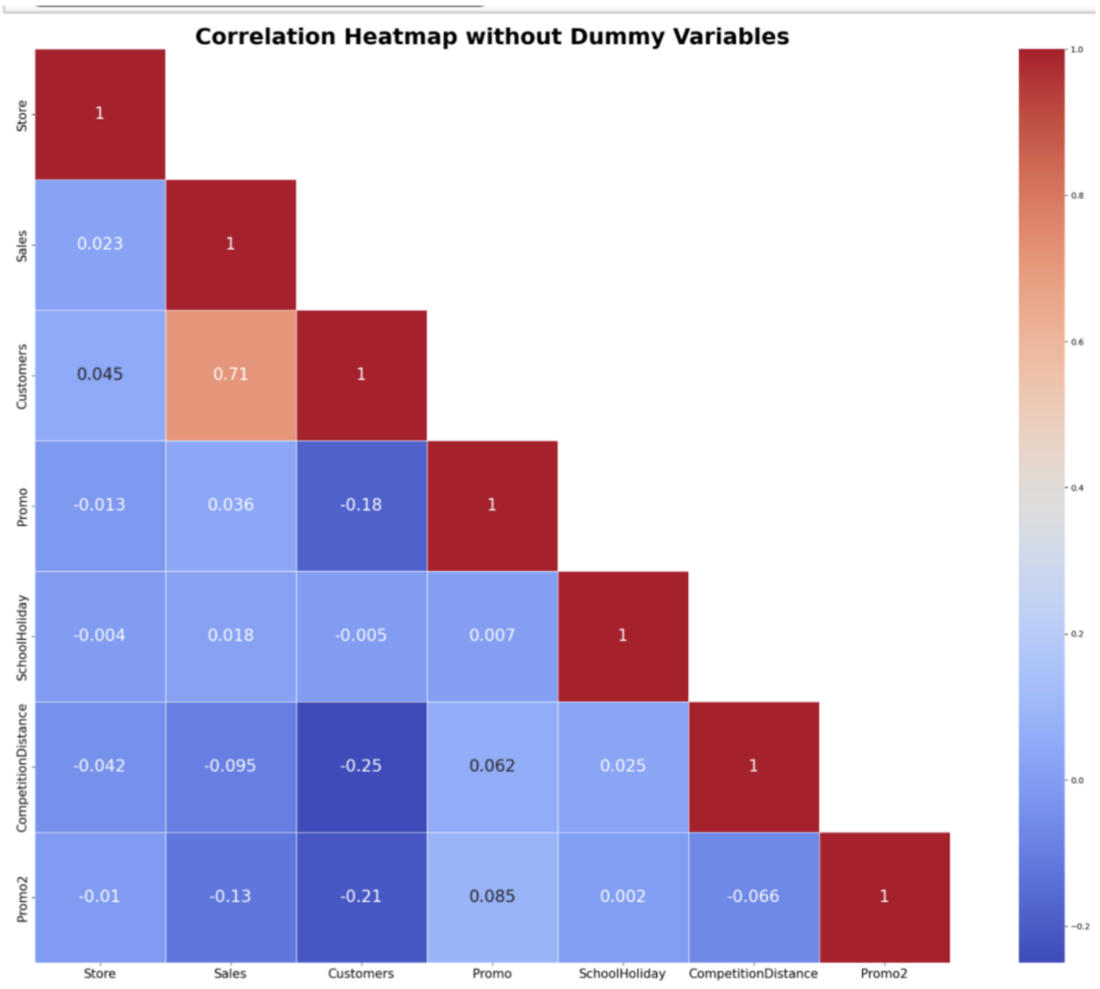


Figure 3.1 – Correlation Heatmap

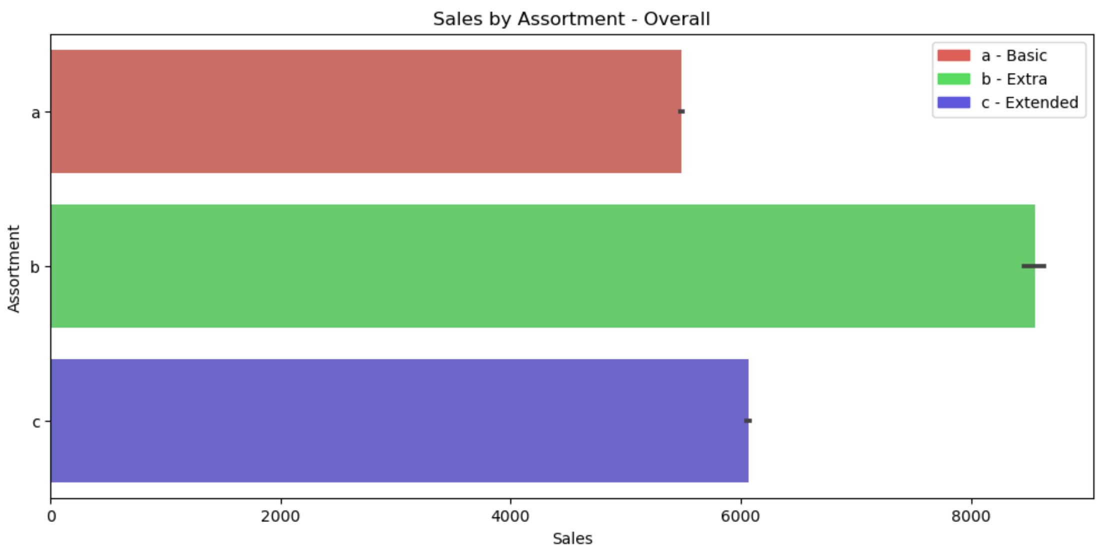


Figure 3.2 - Assortment

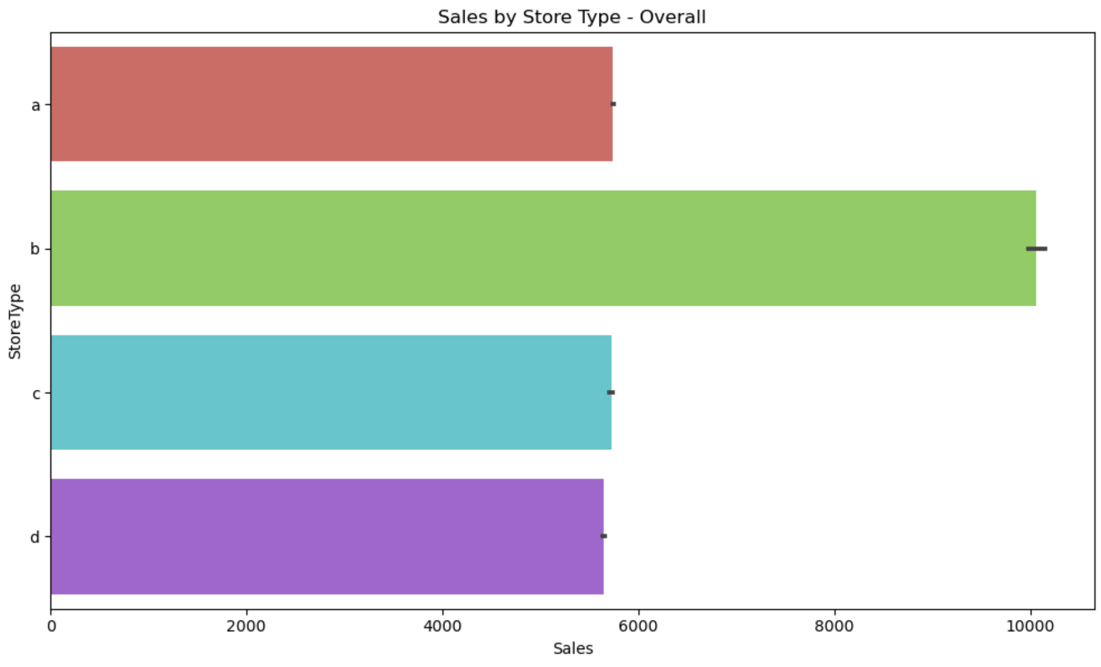


Figure 3.3 – Store Type

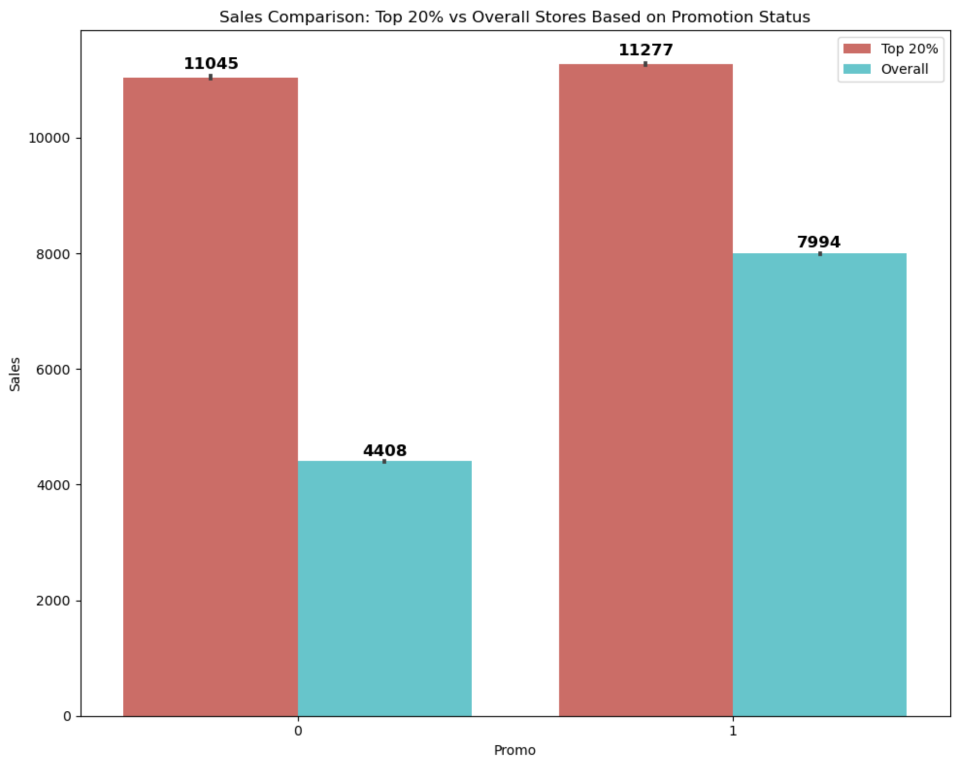


Figure 3.4 - Promotion

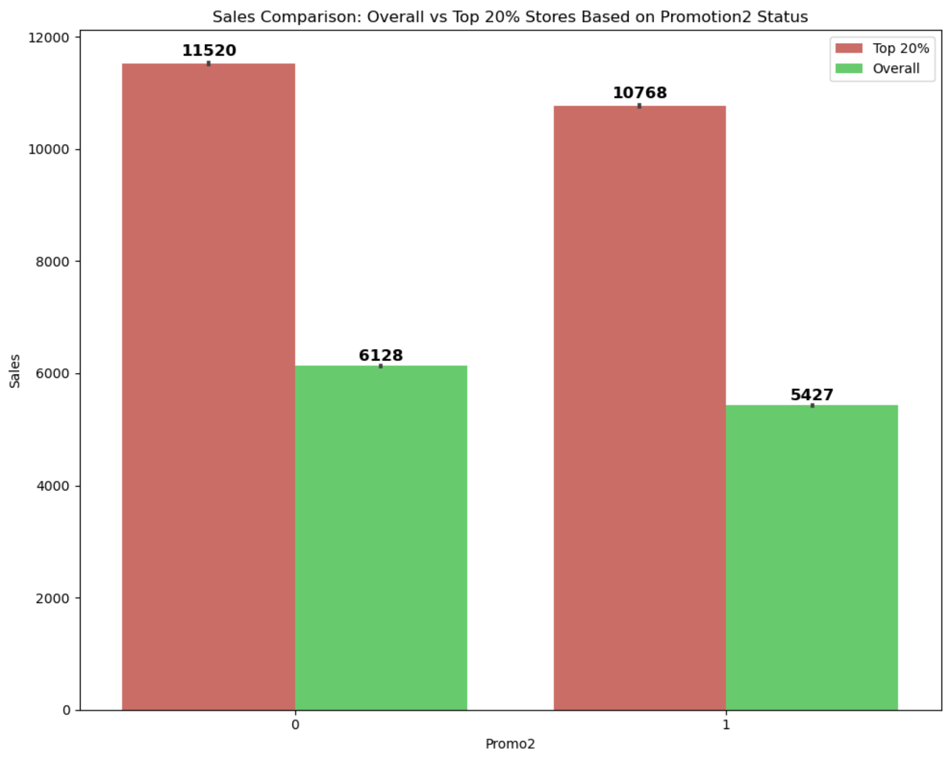


Figure 3.5 – Promotion2

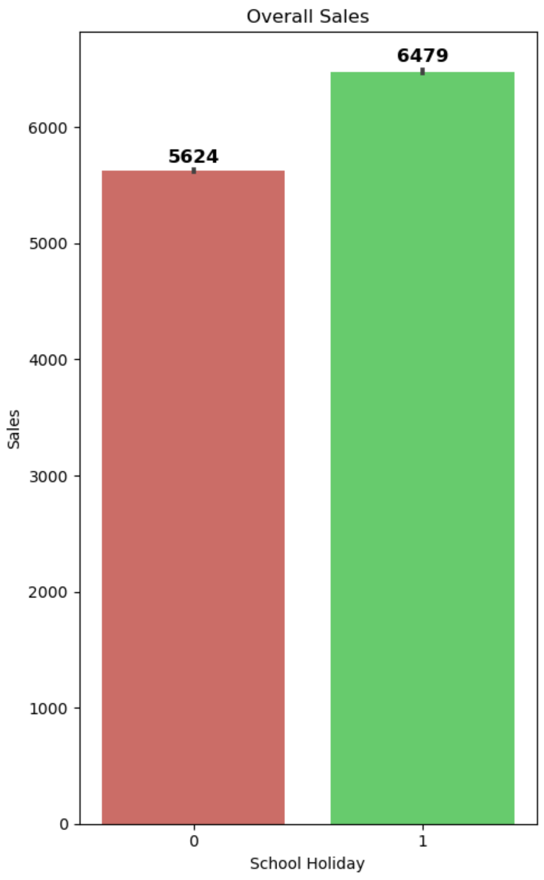


Figure 3.6 – School Holiday

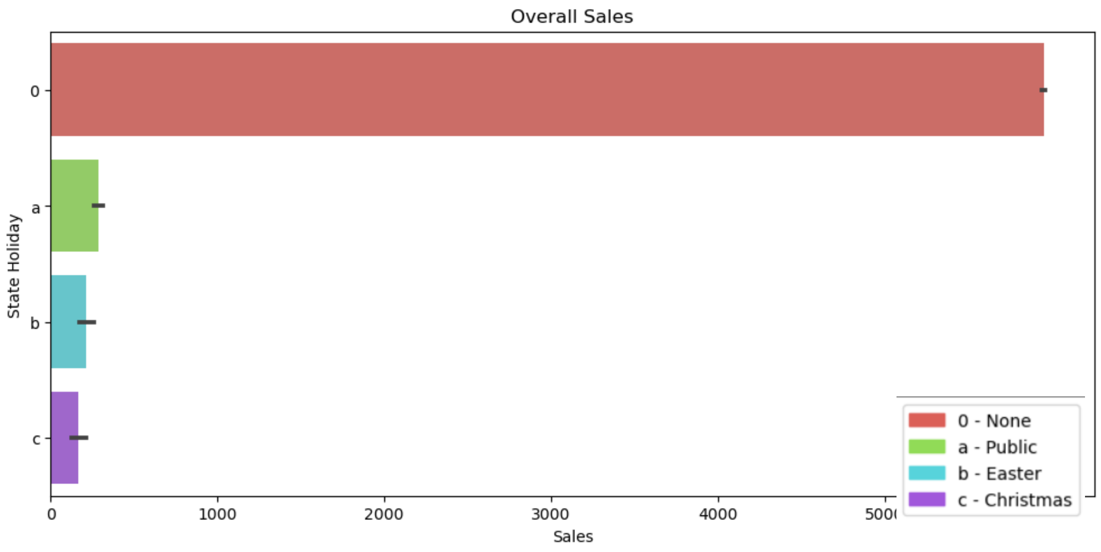


Figure 3.7 – State Holiday

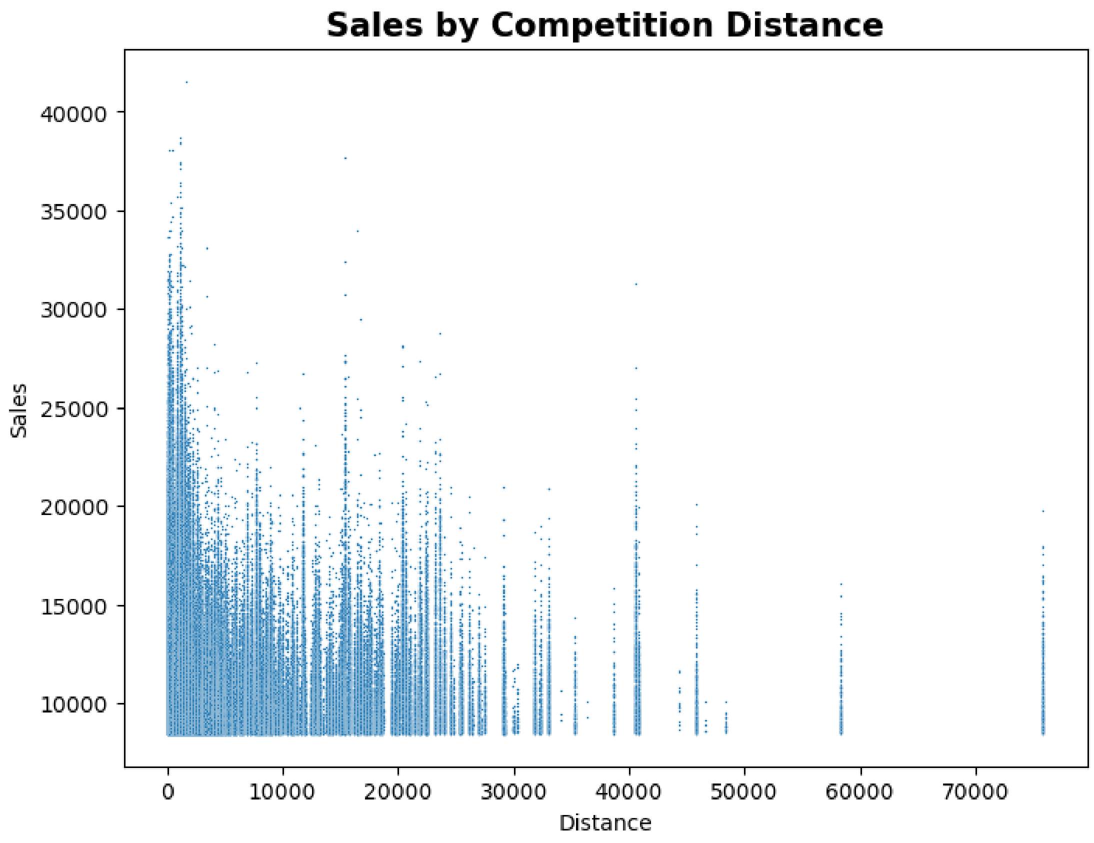


Figure 3.8 – Competition Distance

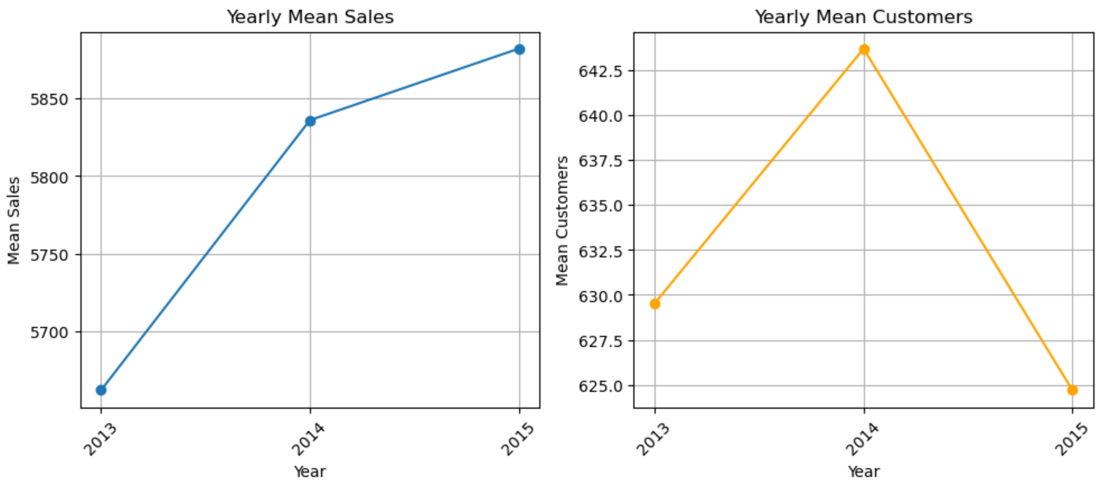


Figure 3.9 – Mean Sales and Mean Customers by year

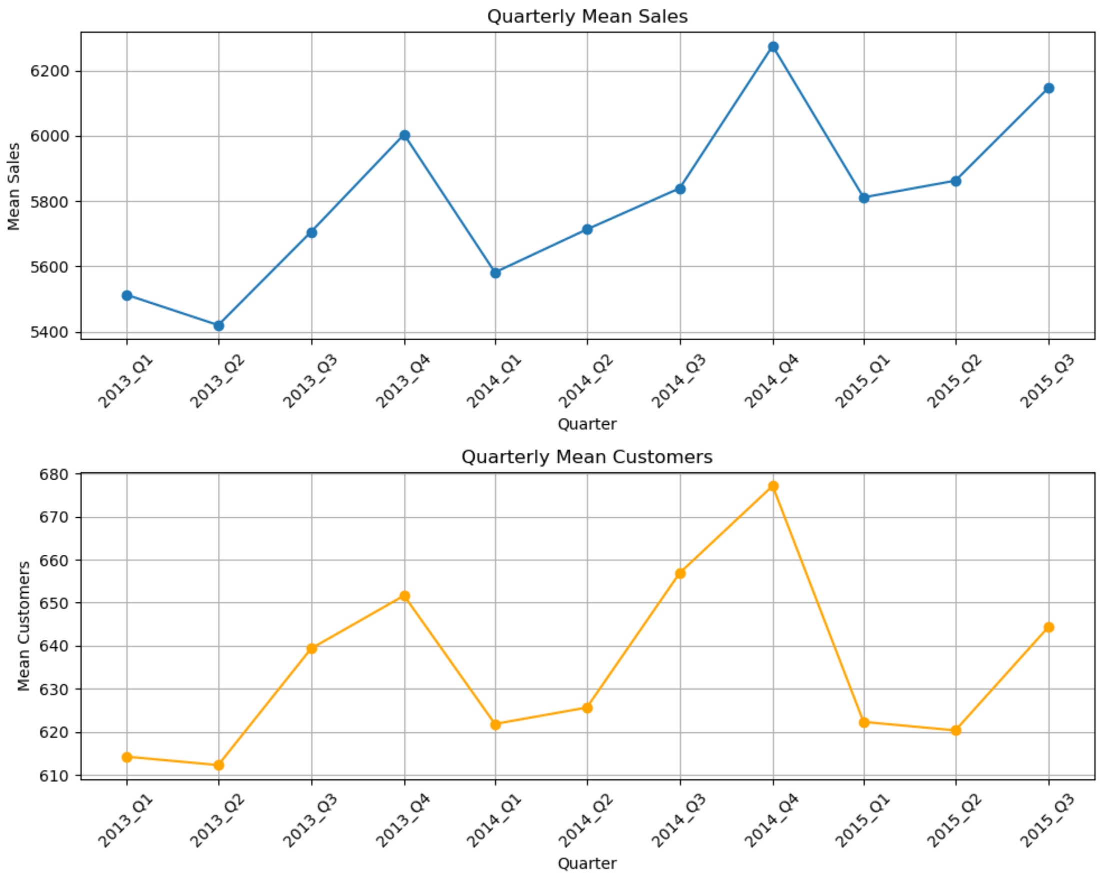


Figure 3.10 - Mean Sales and Mean Customers by quarter

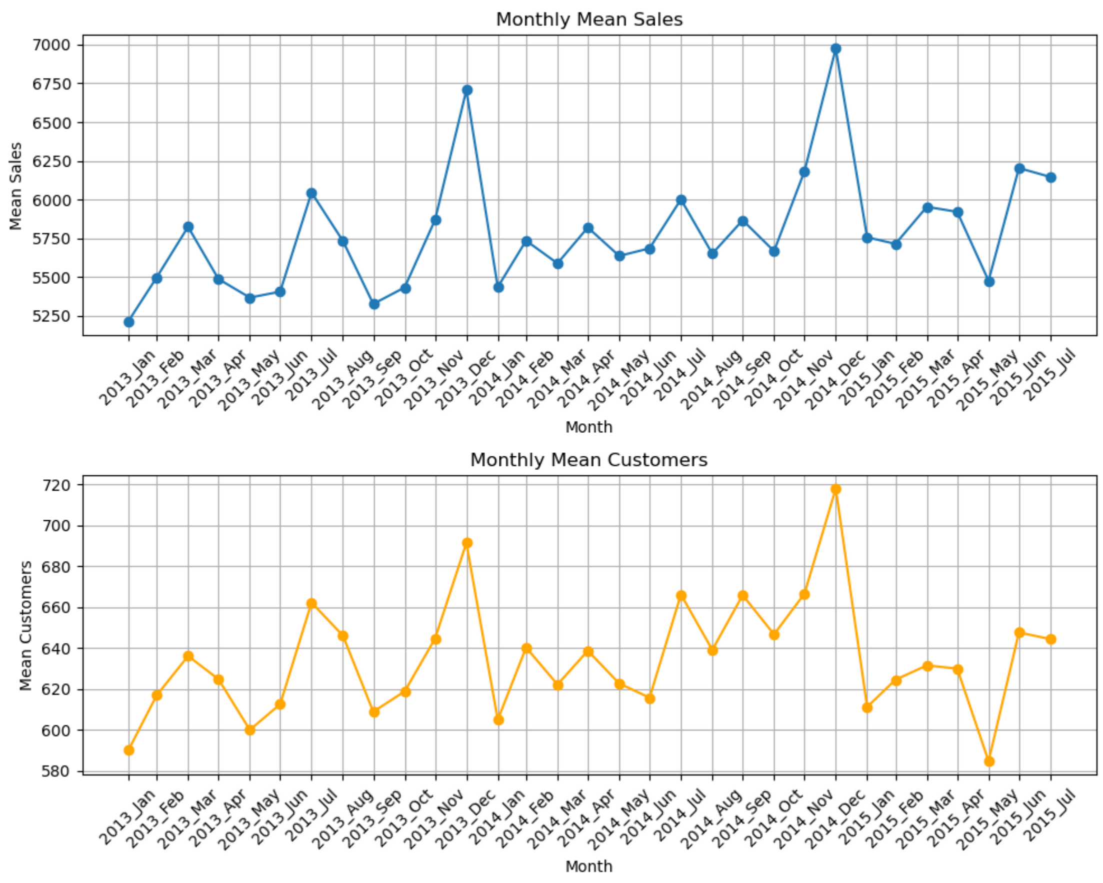


Figure 3.11 – Mean Sales and Mean Customers by month

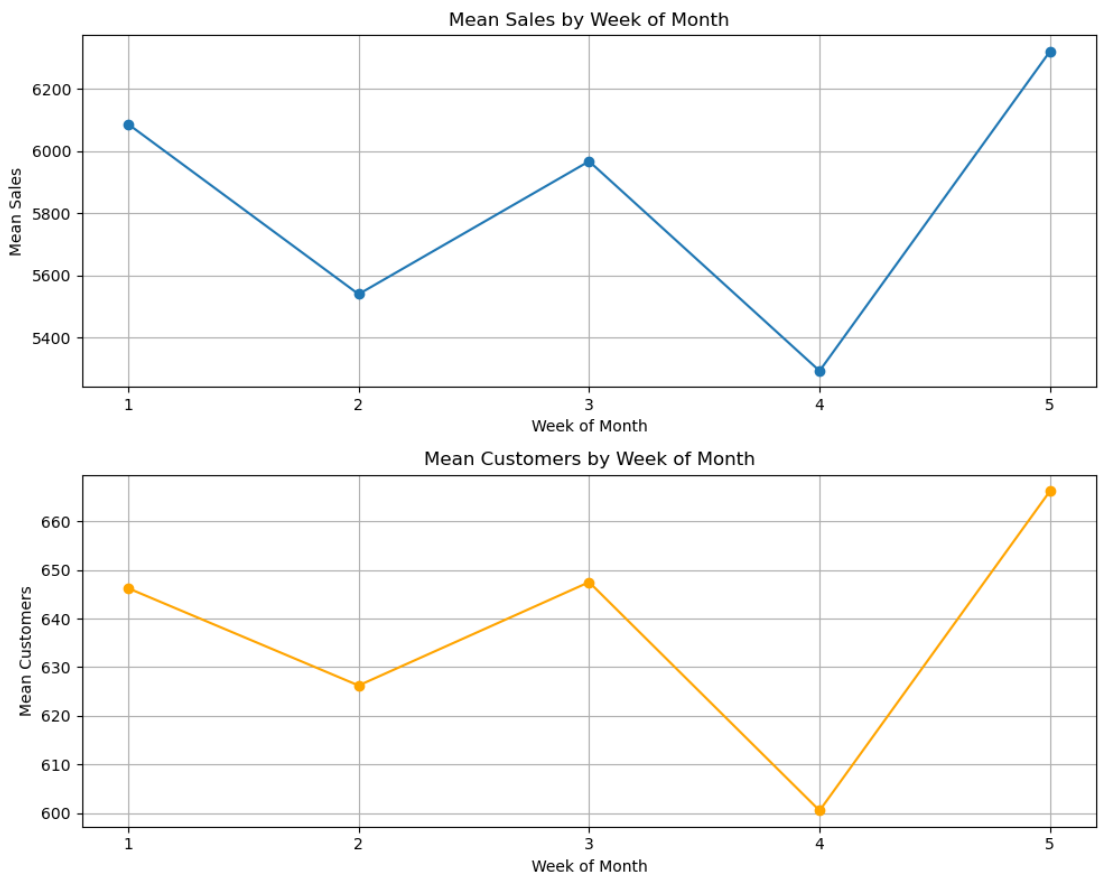


Figure 3.12 - Mean Sales and Mean Customers by week of month

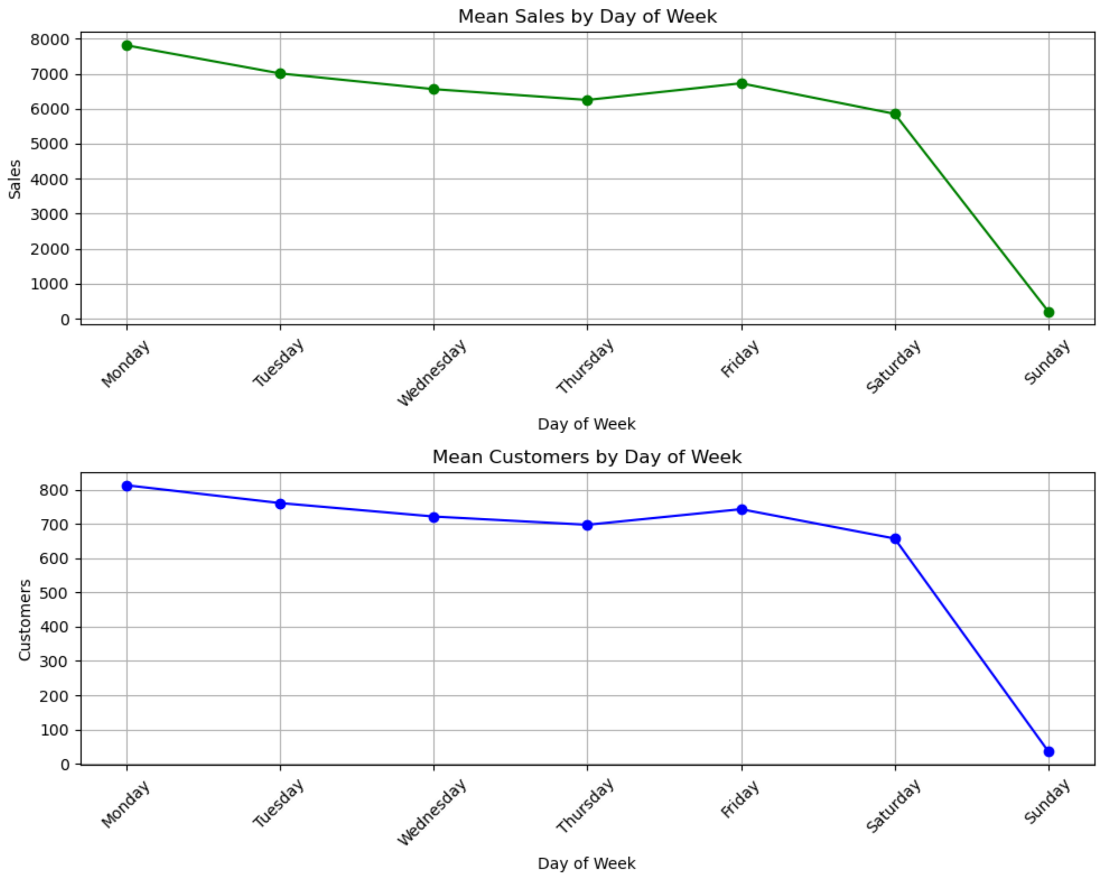


Figure 3.13 - Mean Sales and Mean Customers by day of week (Overall Sales)

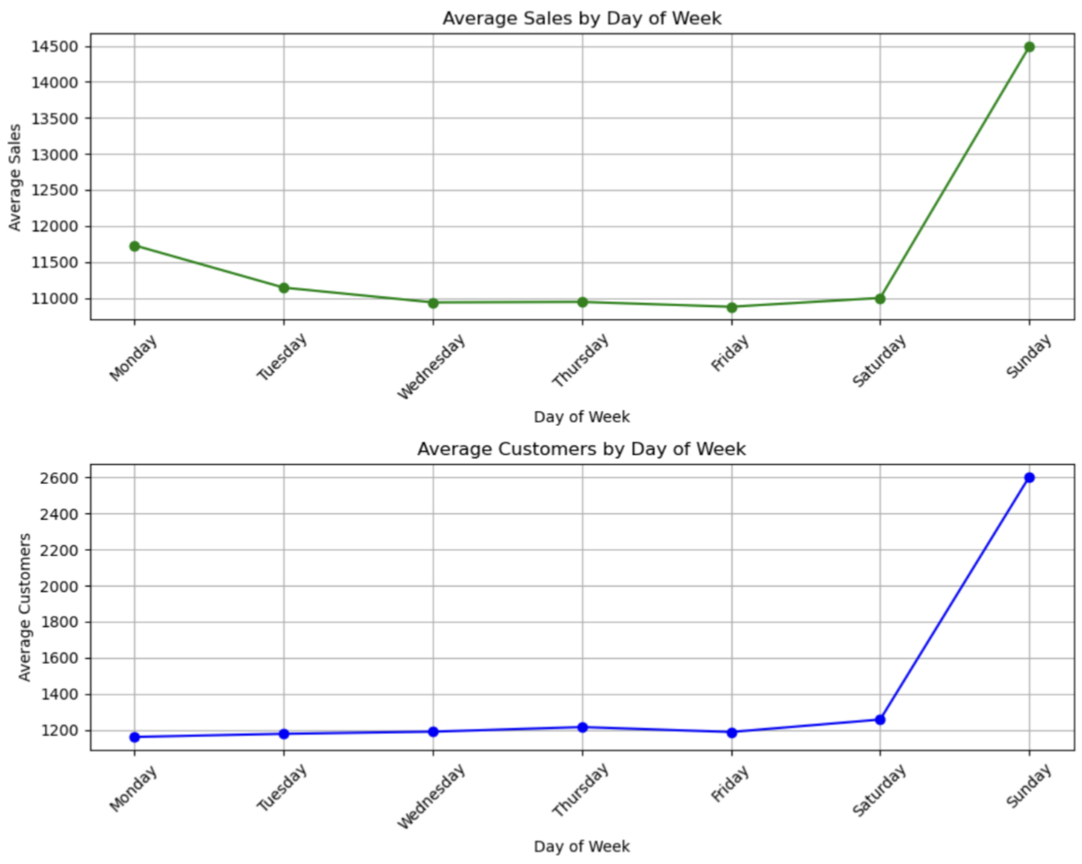


Figure 3.14 - Mean Sales and Mean Customers by day of week (Top 20% Sales)

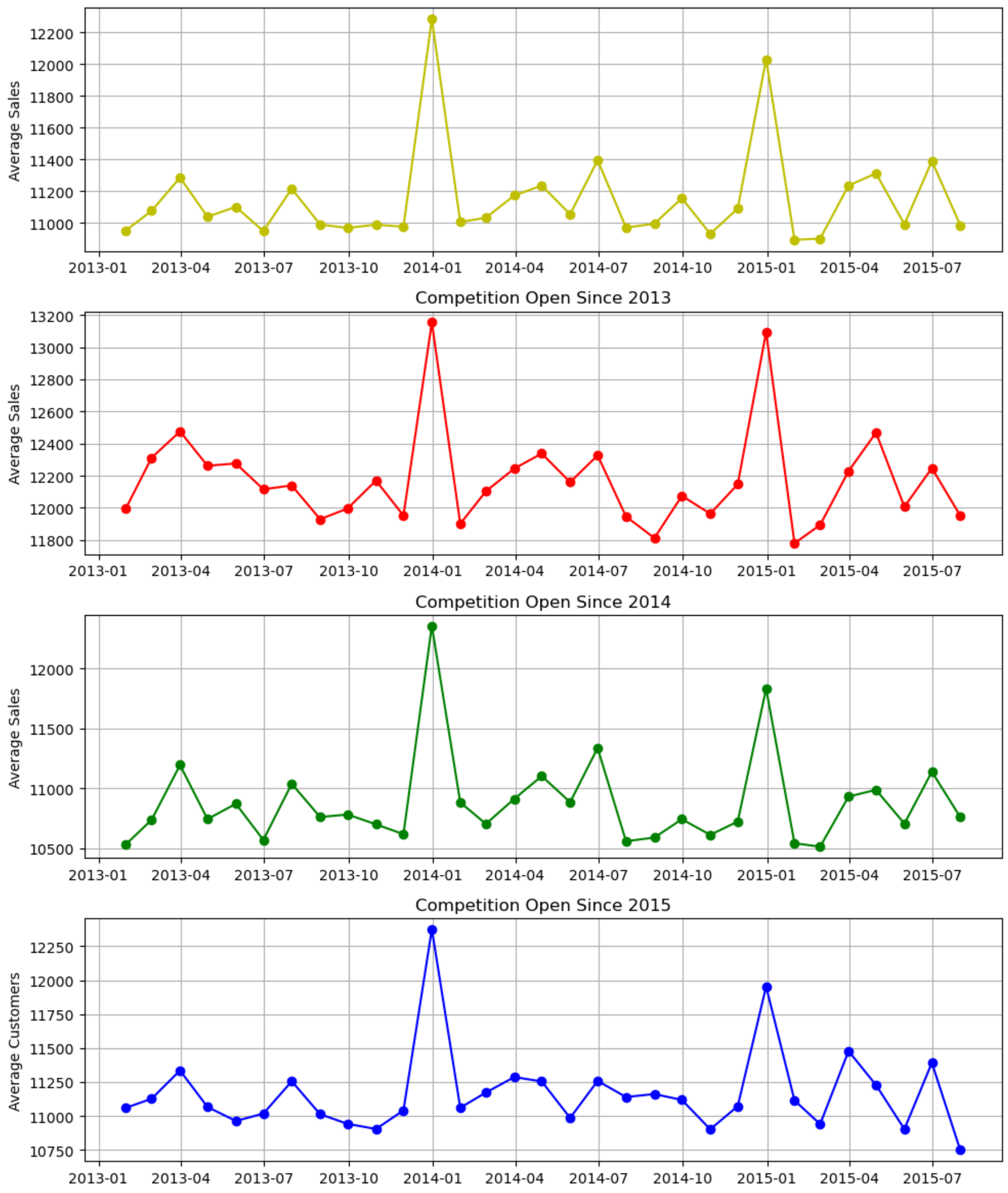


Figure 3.15 – Competition Open Since (by year)

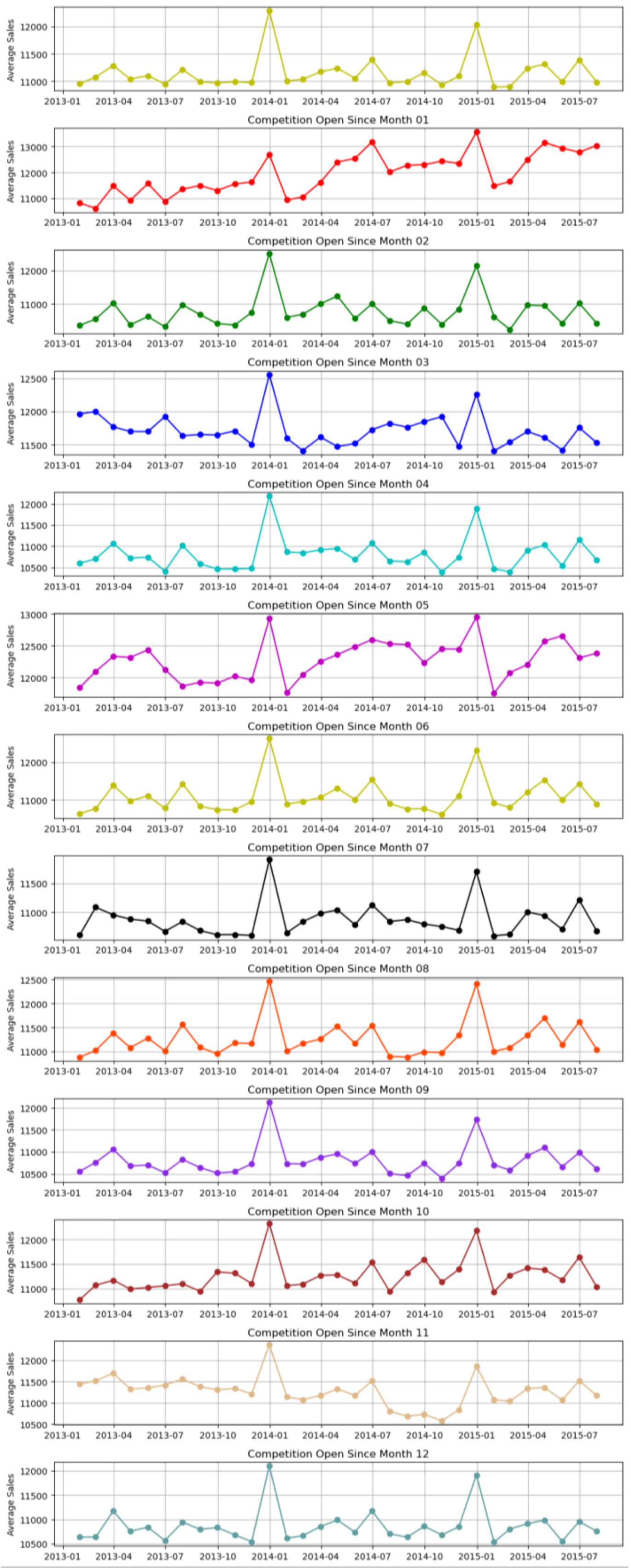
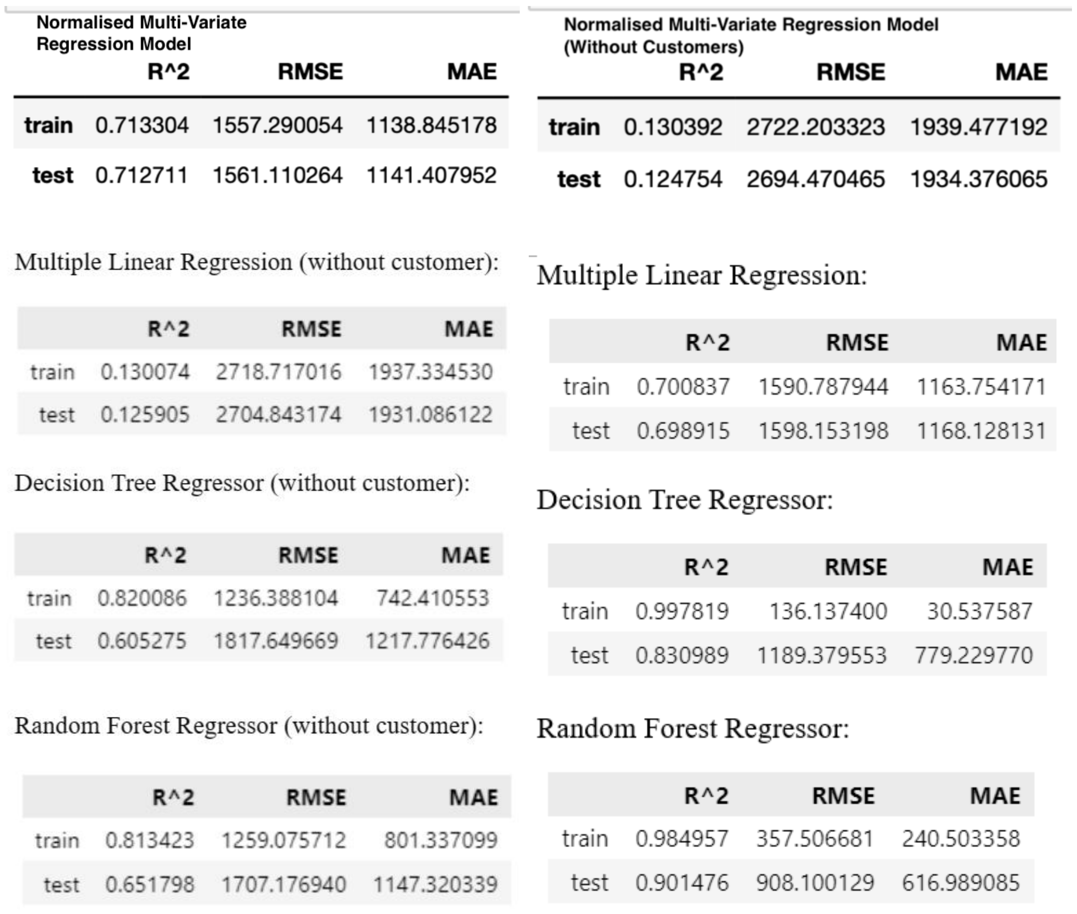


Figure 3.16 – Competition Open Since (by month)



Normalised Multi-Variate Regression Model

| | R^2 | RMSE | MAE |
|---|---|---|---|
| **train** | 0.713304 | 1557.290054 | 1138.845178 |
| **test** | 0.712711 | 1561.110264 | 1141.407952 |

Normalised Multi-Variate Regression Model (Without Customers)

| | R^2 | RMSE | MAE |
|---|---|---|---|
| **train** | 0.130392 | 2722.203323 | 1939.477192 |
| **test** | 0.124754 | 2694.470465 | 1934.376065 |

Multiple Linear Regression (without customer):

| | R^2 | RMSE | MAE |
|---|---|---|---|
| train | 0.130074 | 2718.717016 | 1937.334530 |
| test | 0.125905 | 2704.843174 | 1931.086122 |

Multiple Linear Regression:

| | R^2 | RMSE | MAE |
|---|---|---|---|
| train | 0.700837 | 1590.787944 | 1163.754171 |
| test | 0.698915 | 1598.153198 | 1168.128131 |

Decision Tree Regressor (without customer):

| | R^2 | RMSE | MAE |
|---|---|---|---|
| train | 0.820086 | 1236.388104 | 742.410553 |
| test | 0.605275 | 1817.649669 | 1217.776426 |

Decision Tree Regressor:

| | R^2 | RMSE | MAE |
|---|---|---|---|
| train | 0.997819 | 136.137400 | 30.537587 |
| test | 0.830989 | 1189.379553 | 779.229770 |

Random Forest Regressor (without customer):

| | R^2 | RMSE | MAE |
|---|---|---|---|
| train | 0.813423 | 1259.075712 | 801.337099 |
| test | 0.651798 | 1707.176940 | 1147.320339 |

Random Forest Regressor:

| | R^2 | RMSE | MAE |
|---|---|---|---|
| train | 0.984957 | 357.506681 | 240.503358 |
| test | 0.901476 | 908.100129 | 616.989085 |

Figure 4.0 – Several Models Performance

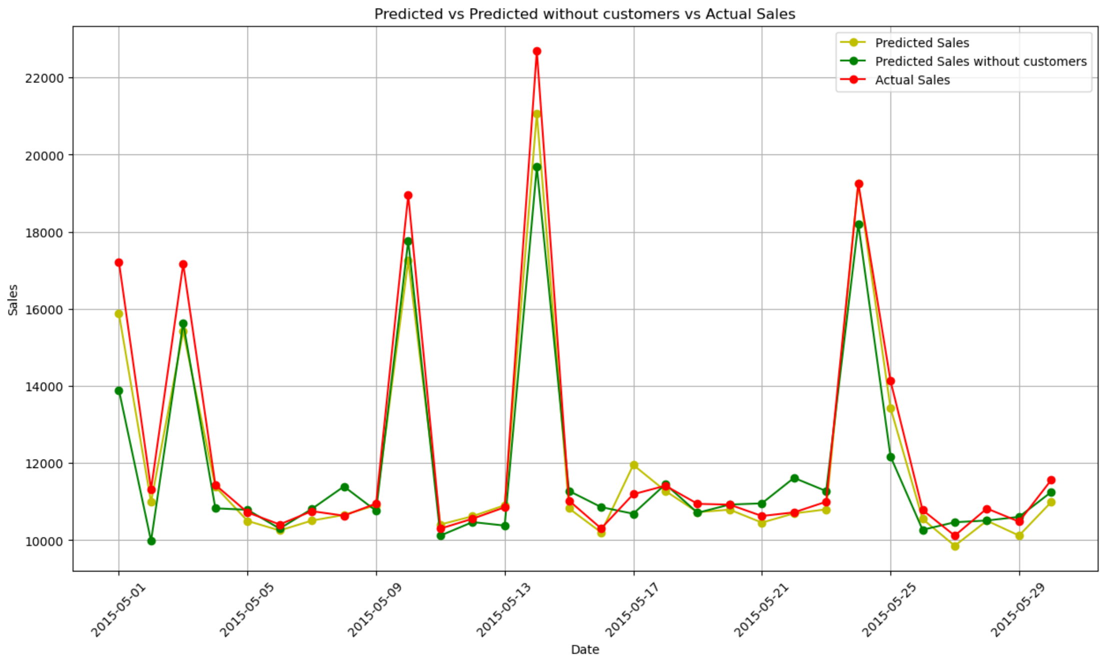


Figure 4.1 – Predicted Sales and Actual Sales

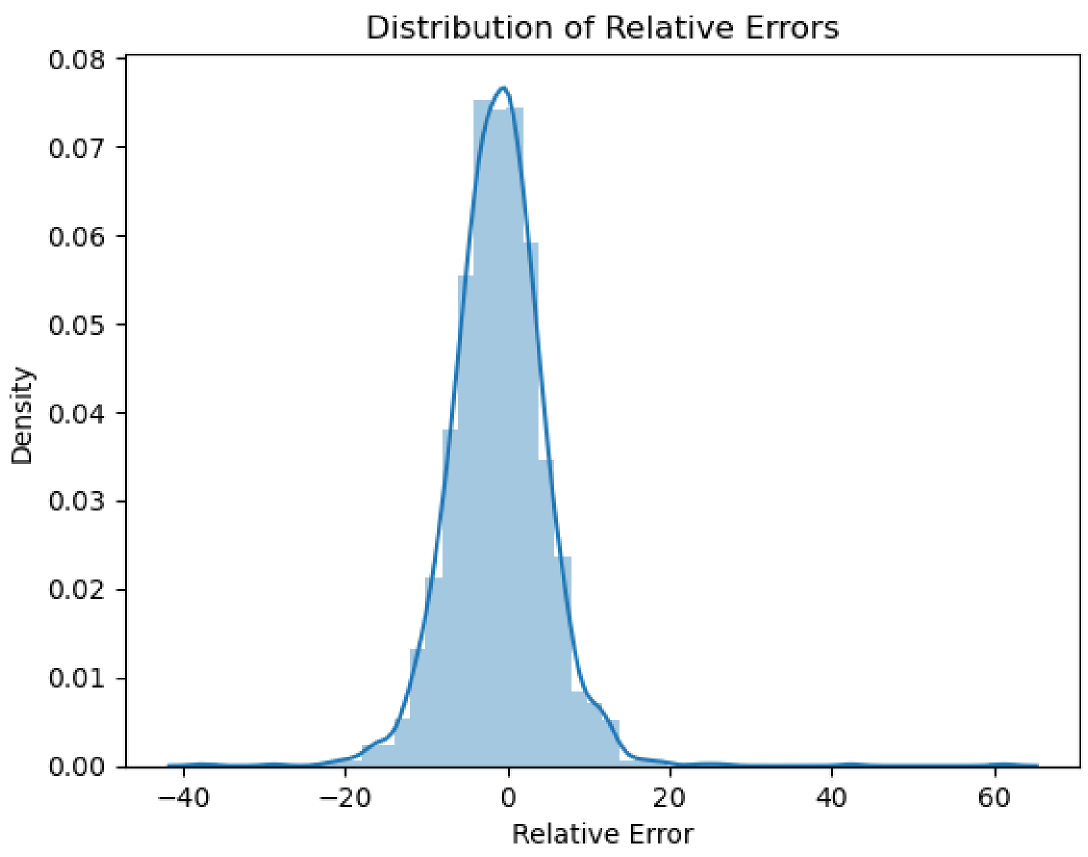


Figure 4.2 – Distribution of Relative Errors

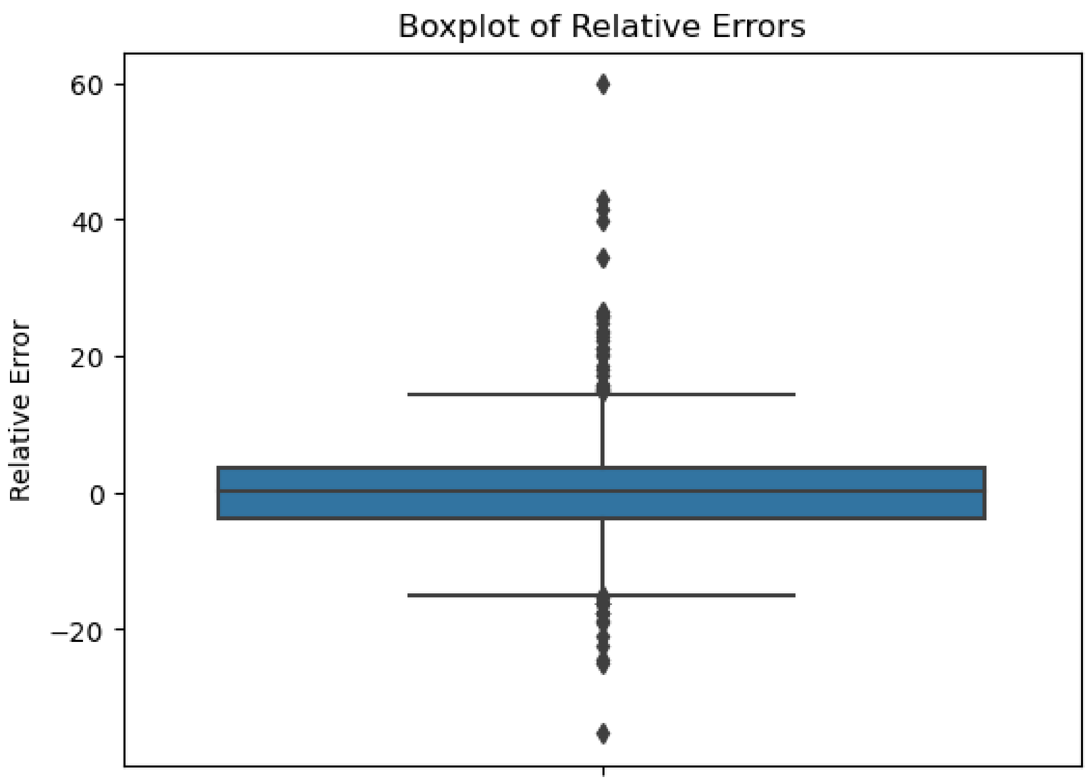
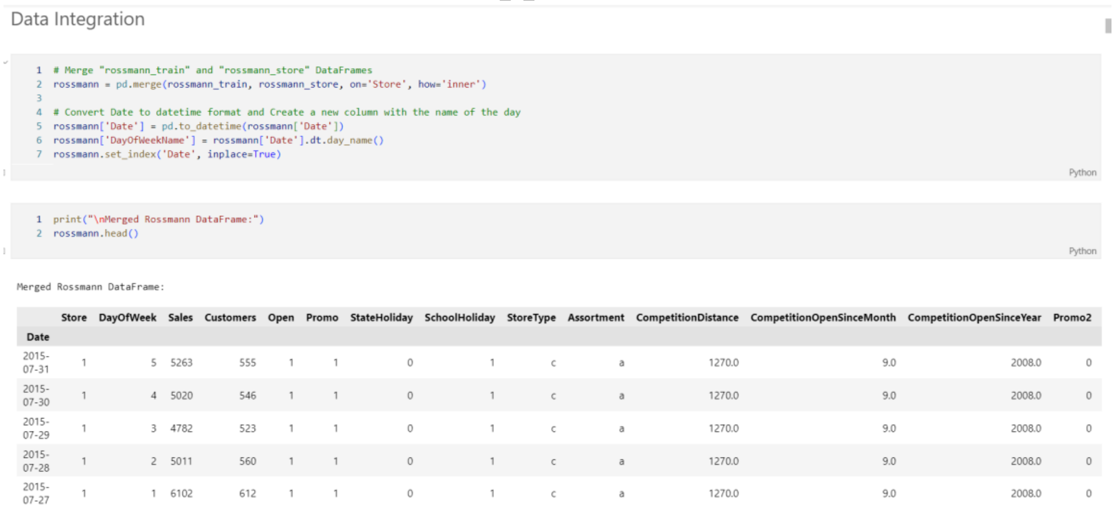


Figure 4.3 – Box and whisker plot of Relative Error

# Reference

Presh Talwalkar, (October 23, 2012), Why are McDonald’s and Burger King usually located near each other? Fast food location game theory https://mindyourdecisions.com/blog/2012/10/23/why-are-mcdonalds-and-burger-king-usually-located-near-each-other-fast-food-location-game-theory/

# Appendix



Data Integration

```python
# Merge "rossmann_train" and "rossmann_store" DataFrames
rossmann = pd.merge(rossmann_train, rossmann_store, on='Store', how='inner')

# Convert Date to datetime format and Create a new column with the name of the day
rossmann['Date'] = pd.to_datetime(rossmann['Date'])
rossmann['DayOfWeekName'] = rossmann['Date'].dt.day_name()
rossmann.set_index('Date', inplace=True)
```

Python

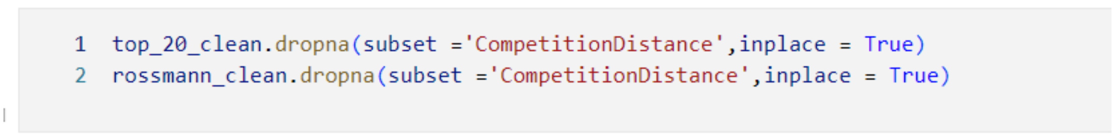
```python
print("\nMerged Rossmann DataFrame:")
rossmann.head()
```

Python

```
Merged Rossmann DataFrame:
```

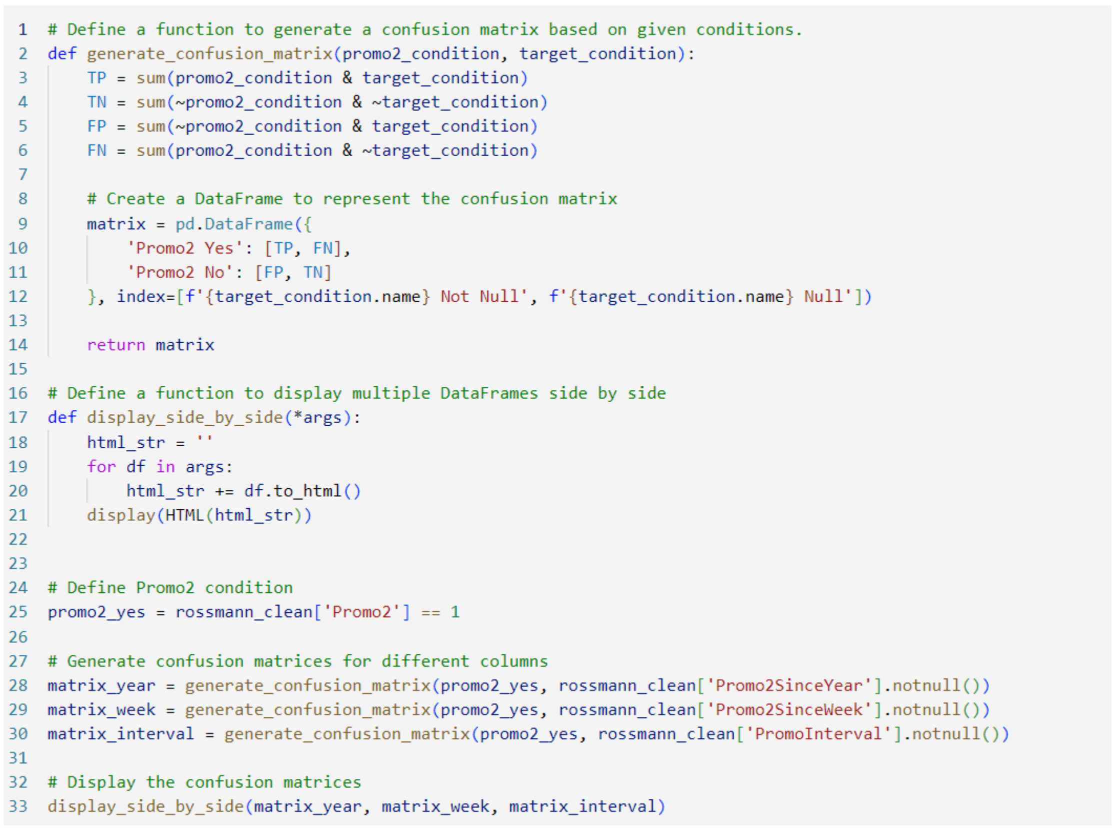
| | Store | DayOfWeek | Sales | Customers | Open | Promo | StateHoliday | SchoolHoliday | StoreType | Assortment | CompetitionDistance | CompetitionOpenSinceMonth | CompetitionOpenSinceYear | Promo2 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Date** | | | | | | | | | | | | | | |
| 2015-07-31 | 1 | 5 | 5263 | 555 | 1 | 1 | 0 | 1 | c | a | 1270.0 | 9.0 | 2008.0 | 0 |
| 2015-07-30 | 1 | 4 | 5020 | 546 | 1 | 1 | 0 | 1 | c | a | 1270.0 | 9.0 | 2008.0 | 0 |
| 2015-07-29 | 1 | 3 | 4782 | 523 | 1 | 1 | 0 | 1 | c | a | 1270.0 | 9.0 | 2008.0 | 0 |
| 2015-07-28 | 1 | 2 | 5011 | 560 | 1 | 1 | 0 | 1 | c | a | 1270.0 | 9.0 | 2008.0 | 0 |
| 2015-07-27 | 1 | 1 | 6102 | 612 | 1 | 1 | 0 | 1 | c | a | 1270.0 | 9.0 | 2008.0 | 0 |

Data Wrangling - Part 1(Data Integration)

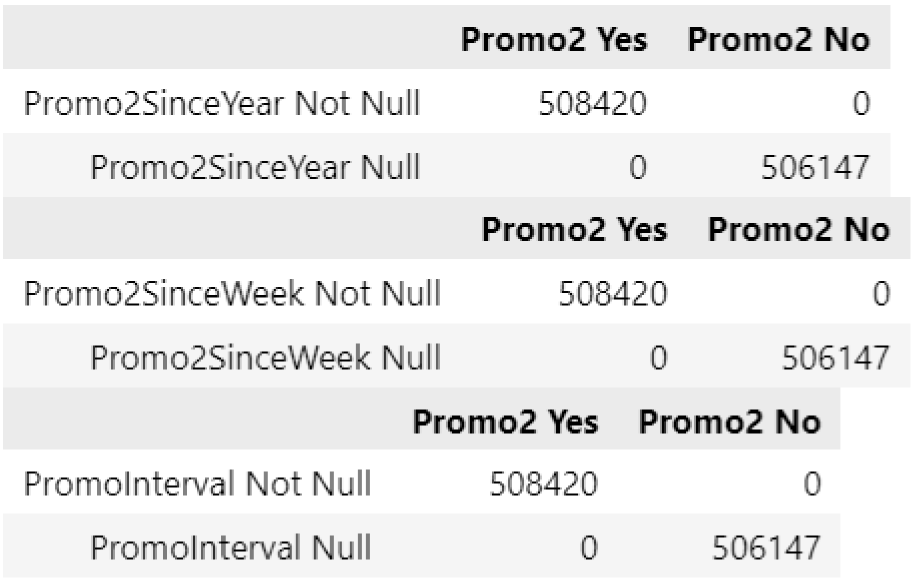


```python
top_20_clean.dropna(subset ='CompetitionDistance',inplace = True)
rossmann_clean.dropna(subset ='CompetitionDistance',inplace = True)
```

Data Wrangling - Part 2 (Data Cleaning)



```python
# Define a function to generate a confusion matrix based on given conditions.
def generate_confusion_matrix(promo2_condition, target_condition):
    TP = sum(promo2_condition & target_condition)
    TN = sum(~promo2_condition & ~target_condition)
    FP = sum(~promo2_condition & target_condition)
    FN = sum(promo2_condition & ~target_condition)

    # Create a DataFrame to represent the confusion matrix
    matrix = pd.DataFrame({
        'Promo2 Yes': [TP, FN],
        'Promo2 No': [FP, TN]
    }, index=[f'{target_condition.name} Not Null', f'{target_condition.name} Null'])

    return matrix

# Define a function to display multiple DataFrames side by side
def display_side_by_side(*args):
    html_str = ''
    for df in args:
        html_str += df.to_html()
    display(HTML(html_str))


# Define Promo2 condition
promo2_yes = rossmann_clean['Promo2'] == 1

# Generate confusion matrices for different columns
matrix_year = generate_confusion_matrix(promo2_yes, rossmann_clean['Promo2SinceYear'].notnull())
matrix_week = generate_confusion_matrix(promo2_yes, rossmann_clean['Promo2SinceWeek'].notnull())
matrix_interval = generate_confusion_matrix(promo2_yes, rossmann_clean['PromoInterval'].notnull())

# Display the confusion matrices
display_side_by_side(matrix_year, matrix_week, matrix_interval)
```



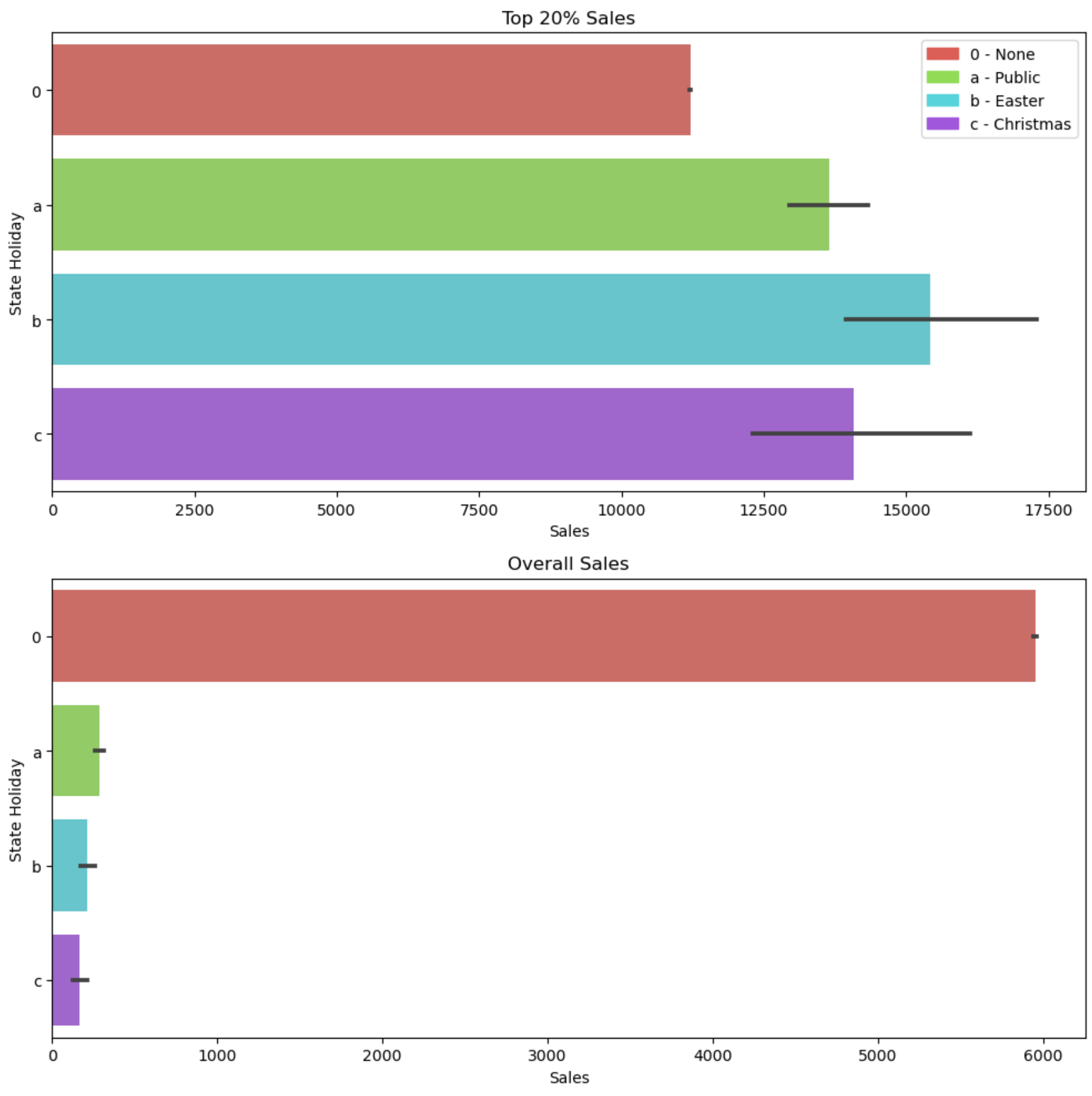
| | Promo2 Yes | Promo2 No |
|---|---|---|
| Promo2SinceYear Not Null | 508420 | 0 |
| Promo2SinceYear Null | 0 | 506147 |

| | Promo2 Yes | Promo2 No |
|---|---|---|
| Promo2SinceWeek Not Null | 508420 | 0 |
| Promo2SinceWeek Null | 0 | 506147 |

| | Promo2 Yes | Promo2 No |
|---|---|---|
| PromoInterval Not Null | 508420 | 0 |
| PromoInterval Null | 0 | 506147 |

Data Wrangling - Part 3 (Check the presence of missing values when there is no second promotion )

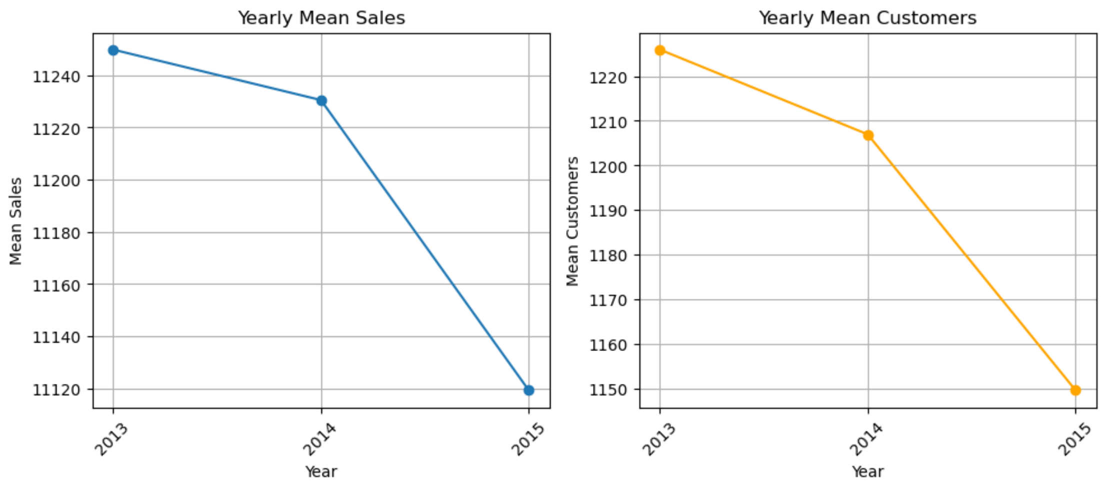
```python
# fig 3.7
colors = sns.color_palette('hls', 4)

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

plt.subplot(2, 1, 1)
sns.barplot(data=top_20_clean, x='Sales',y='StateHoliday', palette=colors)
plt.xlabel('Sales')
plt.ylabel('State Holiday')
plt.title('Top 20% Sales')

legend_patches = [mpatches.Patch(color=colors[i], label=label) for i, label in enumerate(['0 - None', 'a - Public','b - Easter','c - Christmas'])]
plt.legend(handles=legend_patches, loc='upper right')

plt.subplot(2, 1, 2)
sns.barplot(data=rossmann_clean, x='Sales', y='StateHoliday', palette=colors)
plt.xlabel('Sales')
plt.ylabel('State Holiday')
plt.title('Overall Sales')

plt.tight_layout()
```



Data Analysis - Part 1(Graph plotting)

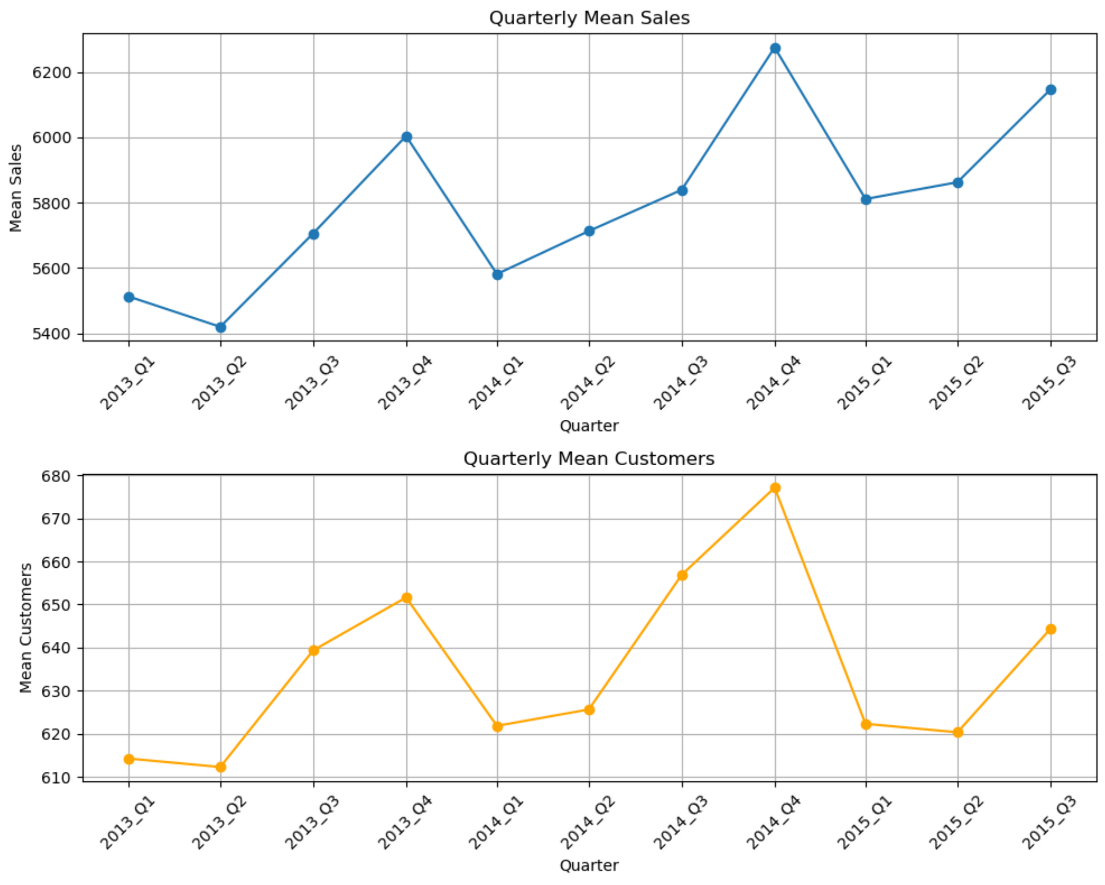
```python
top_20_clean.sort_index(inplace=True)
top_20_years = top_20_clean.resample("1Y").mean()

rossmann_clean.sort_index(inplace=True)
rossmann_years = rossmann_clean.resample("1Y").mean()
```

```python
years = [str(year) for year in top_20_years.index.year.unique()] # Extract years

# Create a line graph for the quarter means
plt.figure(figsize=(10, 8))
plt.subplot(2,2,1)
plt.plot(top_20_years['Sales'], marker='o')
plt.title('Yearly Mean Sales')
plt.xlabel('Year')
plt.ylabel('Mean Sales')
plt.xticks(ticks=top_20_years.index, labels=years, rotation=45) # Set x-ticks as years
plt.tight_layout()
plt.grid()

plt.subplot(2,2,2)
plt.plot(top_20_years['Customers'], marker='o', color='orange')
plt.title('Yearly Mean Customers')
plt.xlabel('Year')
plt.ylabel('Mean Customers')
plt.xticks(ticks=top_20_years.index, labels=years, rotation=45) # Set x-ticks as years
plt.tight_layout()
plt.grid()
```



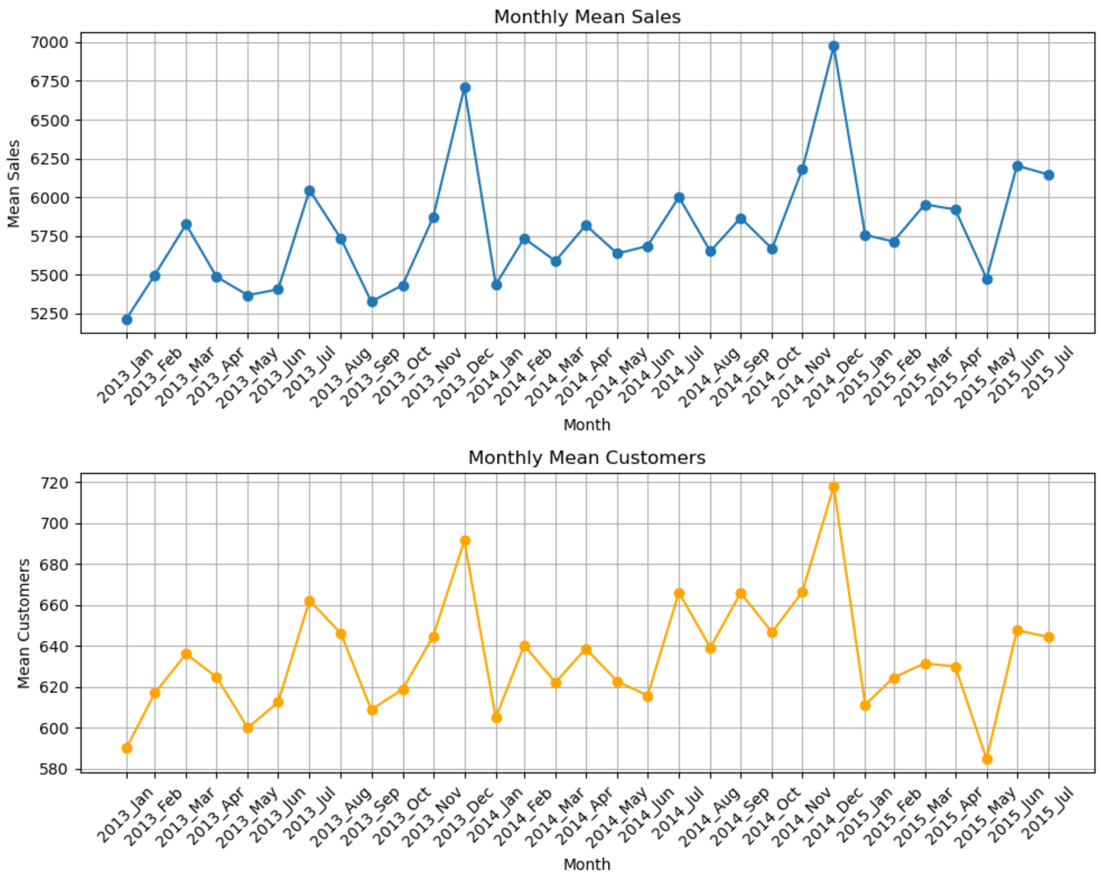
Data Analysis - Part 2 (Time Series Analysis – Yearly)

```python
# fig 3.10
# 1M
rossmann_clean.sort_index(inplace=True)
rossmann_quarters = rossmann_clean.resample("Q").mean()
quarter_labels = [f"{date.year}_Q{((date.month - 1) // 3) + 1}" for date in top_20_quarters.index]

# Create a line graph for the quarter means
plt.figure(figsize=(10, 8))
plt.subplot(2,1,1)
plt.plot(rossmann_quarters['Sales'], marker='o')
plt.title('Quarterly Mean Sales')
plt.xlabel('Quarter')
plt.ylabel('Mean Sales')
plt.xticks(ticks=rossmann_quarters.index, labels=quarter_labels, rotation=45)
plt.tight_layout()
plt.grid()

plt.subplot(2,1,2)
plt.plot(rossmann_quarters['Customers'], marker='o', color='orange')
plt.title('Quarterly Mean Customers')
plt.xlabel('Quarter')
plt.ylabel('Mean Customers')
plt.xticks(ticks=rossmann_quarters.index, labels=quarter_labels, rotation=45)
plt.tight_layout()
plt.grid()
```



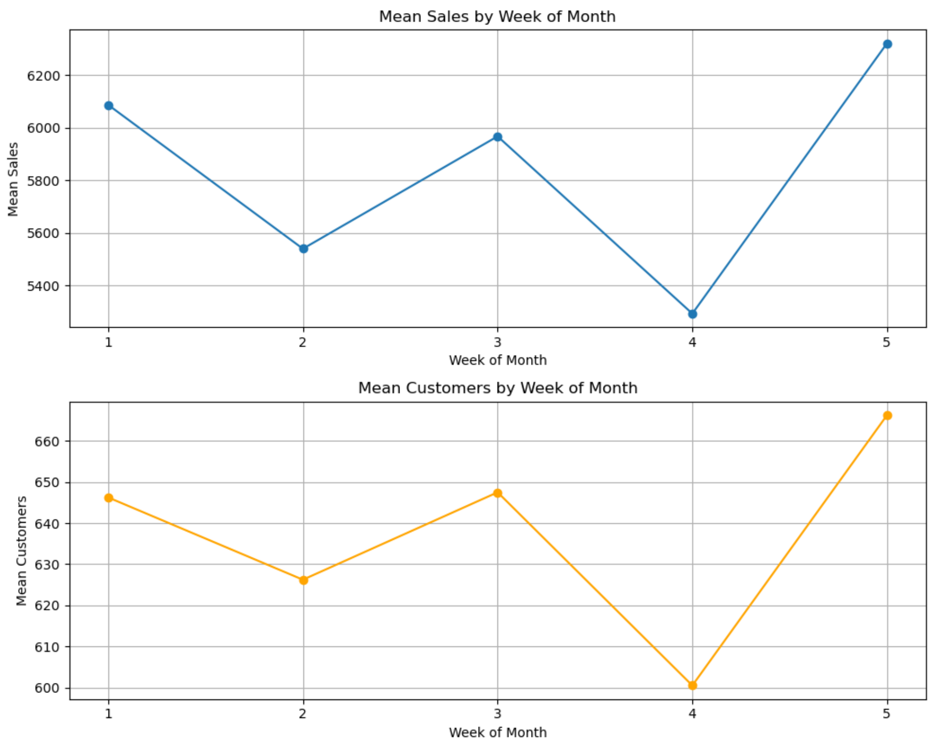
Data Analysis - Part 3 (Time Series Analysis - Quarterly)

```python
# fig 3.11
rossmann_clean.sort_index(inplace=True)
rossmann_months = rossmann_clean.resample("1M").mean()
month_labels = [f"{date.year}_{date.strftime('%b')}" for date in rossmann_months.index]

# Create a line graph for the quarter means
plt.figure(figsize=(10, 8))
plt.subplot(2,1,1)
plt.plot(rossmann_months['Sales'], marker='o')
plt.title('Monthly Mean Sales')
plt.xlabel('Month')
plt.ylabel('Mean Sales')
plt.xticks(ticks=rossmann_months.index,labels=month_labels,rotation=45)
plt.tight_layout()
plt.grid()

plt.subplot(2,1,2)
plt.plot(rossmann_months['Customers'], marker='o', color='orange')
plt.title('Monthly Mean Customers')
plt.xlabel('Month')
plt.ylabel('Mean Customers')
plt.xticks(ticks=rossmann_months.index,labels=month_labels,rotation=45)
plt.tight_layout()
plt.grid()
```



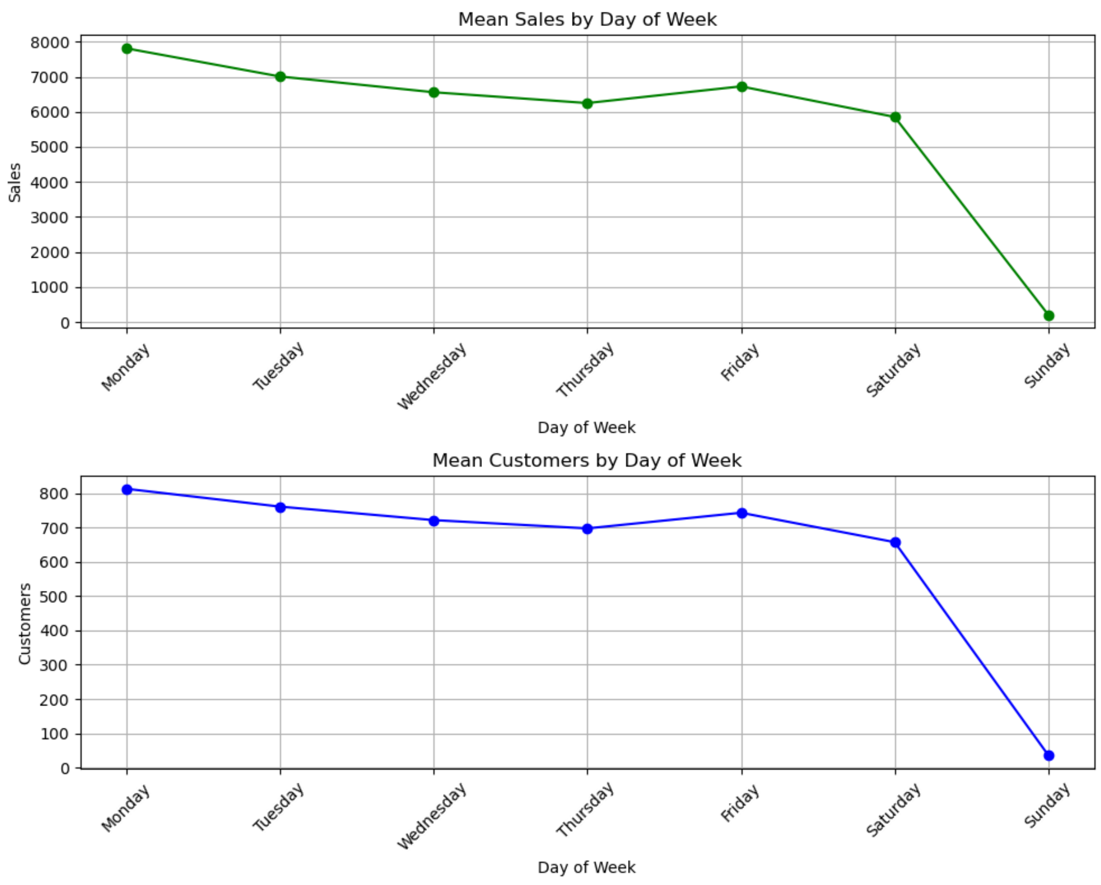
Data Analysis - Part 4 (Time Series Analysis - Monthly)

```python
# fig 3.12

rossmann_clean['week_of_month'] = rossmann_clean.index.day // 7 + 1
rossmann_weekly = rossmann_clean.groupby('week_of_month').mean()

# Plotting
fig, ax = plt.subplots(2, 1, figsize=(10, 8))


ax[0].plot(rossmann_weekly['Sales'], marker='o')
ax[0].set_title('Mean Sales by Week of Month')
ax[0].set_xlabel('Week of Month')
ax[0].set_ylabel('Mean Sales')
ax[0].set_xticks([1, 2, 3, 4, 5])
ax[0].grid(True)


ax[1].plot(rossmann_weekly['Customers'], marker='o', color='orange')
ax[1].set_title('Mean Customers by Week of Month')
ax[1].set_xlabel('Week of Month')
ax[1].set_ylabel('Mean Customers')
ax[1].set_xticks([1, 2, 3, 4, 5])
ax[1].grid(True)

plt.tight_layout()
plt.show()
```

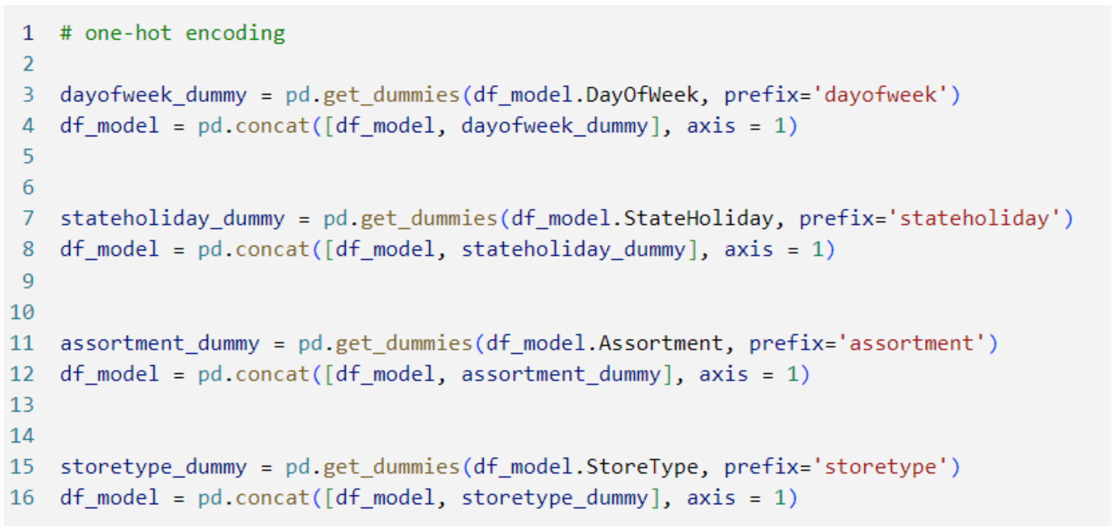


Data Analysis - Part 5 (Time Series Analysis - Week of Month)

```python
# fig 3.13
rossmann_daily = rossmann_clean.groupby('DayOfWeekName')[['Sales', 'Customers']].mean().reindex(
    ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday']
)

# Plotting
fig, ax = plt.subplots(2, 1, figsize=(10, 8))

# Sales subplot
ax[0].plot(rossmann_daily['Sales'], marker='o', color='g', label='Total')
ax[0].set_title('Mean Sales by Day of Week')
ax[0].set_xlabel('Day of Week')
ax[0].set_ylabel('Sales')
ax[0].grid(True)
ax[0].set_xticks(list(range(7)))
ax[0].set_xticklabels(['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday'], rotation=45)

# Customers subplot
ax[1].plot(rossmann_daily['Customers'], marker='o', color='b', label='Total')
ax[1].set_title('Mean Customers by Day of Week')
ax[1].set_xlabel('Day of Week')
ax[1].set_ylabel('Customers')
ax[1].grid(True)
ax[1].set_xticks(list(range(7)))
ax[1].set_xticklabels(['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday'], rotation=45)

plt.tight_layout()
plt.show()
```

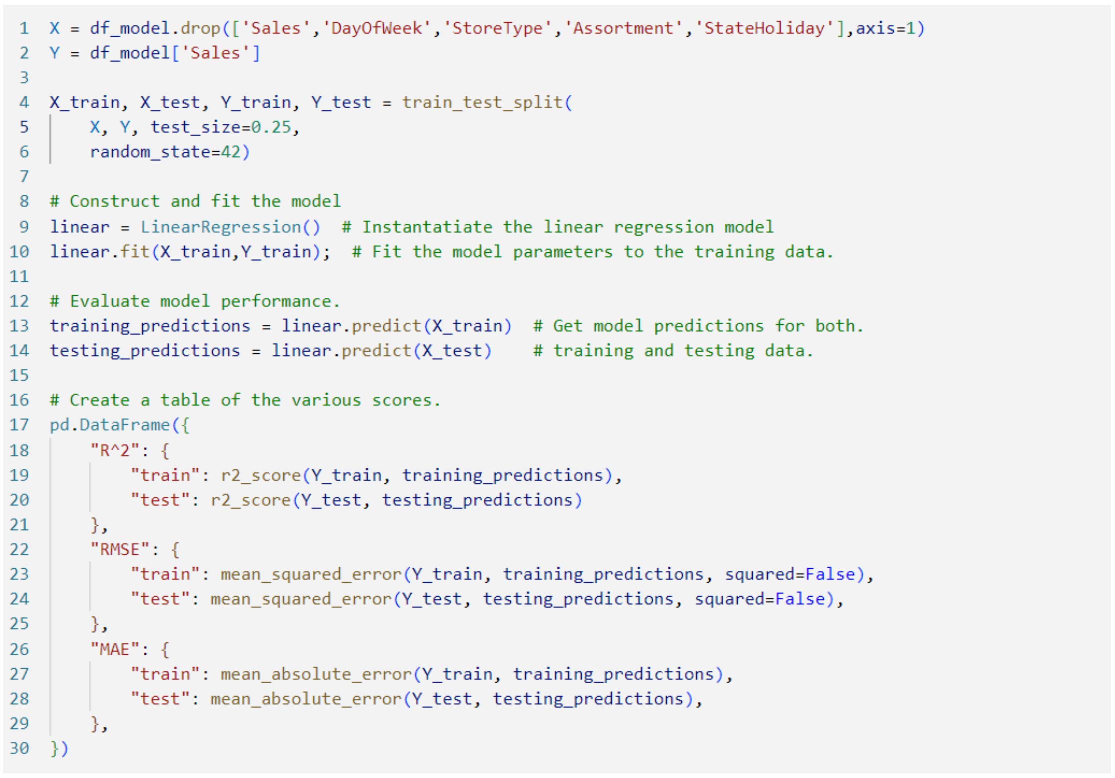


Data Analysis - Part 6 (Times Series Analysis - Days of Week)



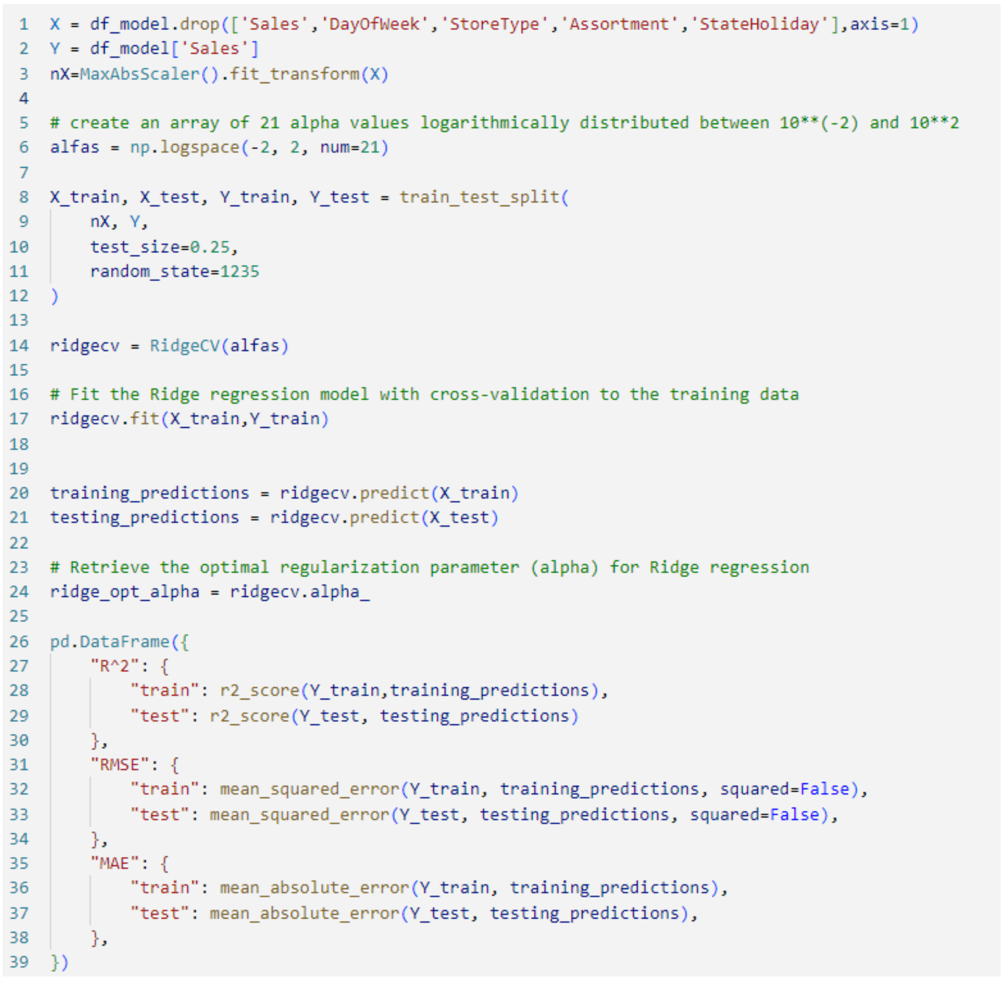
```python
# one-hot encoding

dayofweek_dummy = pd.get_dummies(df_model.DayOfWeek, prefix='dayofweek')
df_model = pd.concat([df_model, dayofweek_dummy], axis = 1)


stateholiday_dummy = pd.get_dummies(df_model.StateHoliday, prefix='stateholiday')
df_model = pd.concat([df_model, stateholiday_dummy], axis = 1)


assortment_dummy = pd.get_dummies(df_model.Assortment, prefix='assortment')
df_model = pd.concat([df_model, assortment_dummy], axis = 1)


storetype_dummy = pd.get_dummies(df_model.StoreType, prefix='storetype')
df_model = pd.concat([df_model, storetype_dummy], axis = 1)
```

Modelling - One-Hot Encoding



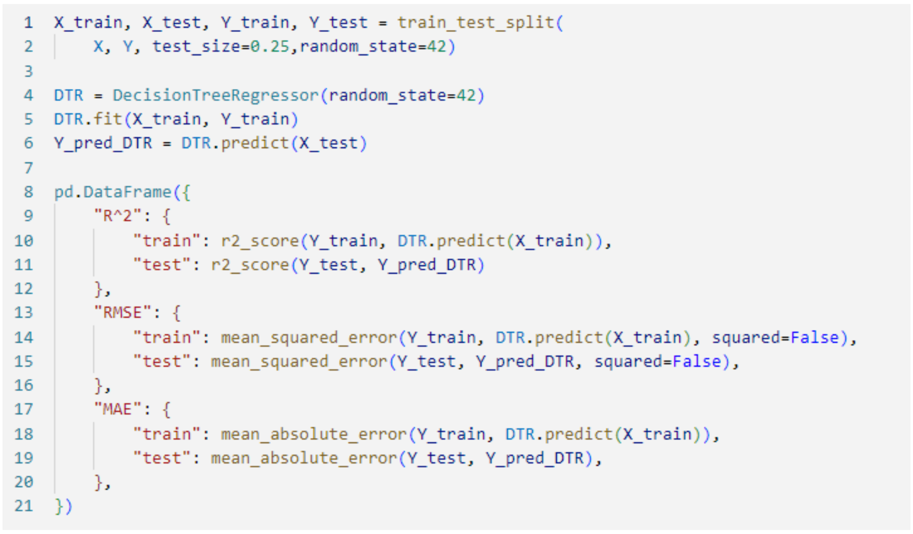
```python
X = df_model.drop(['Sales','DayOfWeek','StoreType','Assortment','StateHoliday'],axis=1)
Y = df_model['Sales']

X_train, X_test, Y_train, Y_test = train_test_split(
    X, Y, test_size=0.25,
    random_state=42)

# Construct and fit the model
linear = LinearRegression()  # Instantatiate the linear regression model
linear.fit(X_train,Y_train);  # Fit the model parameters to the training data.

# Evaluate model performance.
training_predictions = linear.predict(X_train)  # Get model predictions for both.
testing_predictions = linear.predict(X_test)    # training and testing data.

# Create a table of the various scores.
pd.DataFrame({
    "R^2": {
        "train": r2_score(Y_train, training_predictions),
        "test": r2_score(Y_test, testing_predictions)
    },
    "RMSE": {
        "train": mean_squared_error(Y_train, training_predictions, squared=False),
        "test": mean_squared_error(Y_test, testing_predictions, squared=False),
    },
    "MAE": {
        "train": mean_absolute_error(Y_train, training_predictions),
        "test": mean_absolute_error(Y_test, testing_predictions),
    },
})
```

Modelling - Multiple Linear Regression



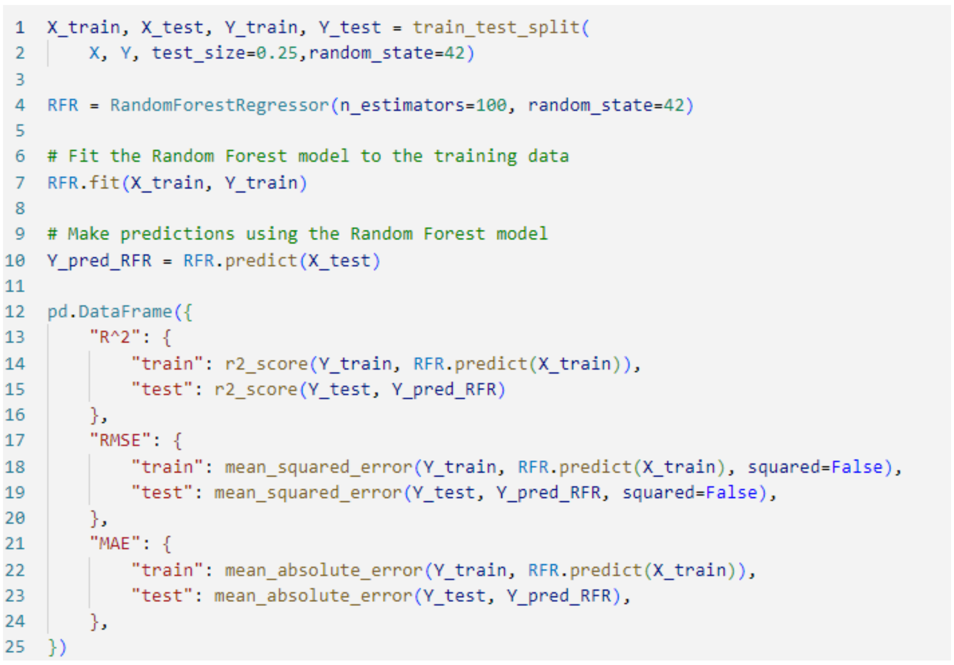
```python
X = df_model.drop(['Sales','DayOfWeek','StoreType','Assortment','StateHoliday'],axis=1)
Y = df_model['Sales']
nX=MaxAbsScaler().fit_transform(X)

# create an array of 21 alpha values logarithmically distributed between 10**(-2) and 10**2
alfas = np.logspace(-2, 2, num=21)

X_train, X_test, Y_train, Y_test = train_test_split(
    nX, Y,
    test_size=0.25,
    random_state=1235
)

ridgecv = RidgeCV(alfas)

# Fit the Ridge regression model with cross-validation to the training data
ridgecv.fit(X_train,Y_train)


training_predictions = ridgecv.predict(X_train)
testing_predictions = ridgecv.predict(X_test)

# Retrieve the optimal regularization parameter (alpha) for Ridge regression
ridge_opt_alpha = ridgecv.alpha_

pd.DataFrame({
    "R^2": {
        "train": r2_score(Y_train,training_predictions),
        "test": r2_score(Y_test, testing_predictions)
    },
    "RMSE": {
        "train": mean_squared_error(Y_train, training_predictions, squared=False),
        "test": mean_squared_error(Y_test, testing_predictions, squared=False),
    },
    "MAE": {
        "train": mean_absolute_error(Y_train, training_predictions),
        "test": mean_absolute_error(Y_test, testing_predictions),
    },
})
```

Modelling - Ridge Regression with Normalization



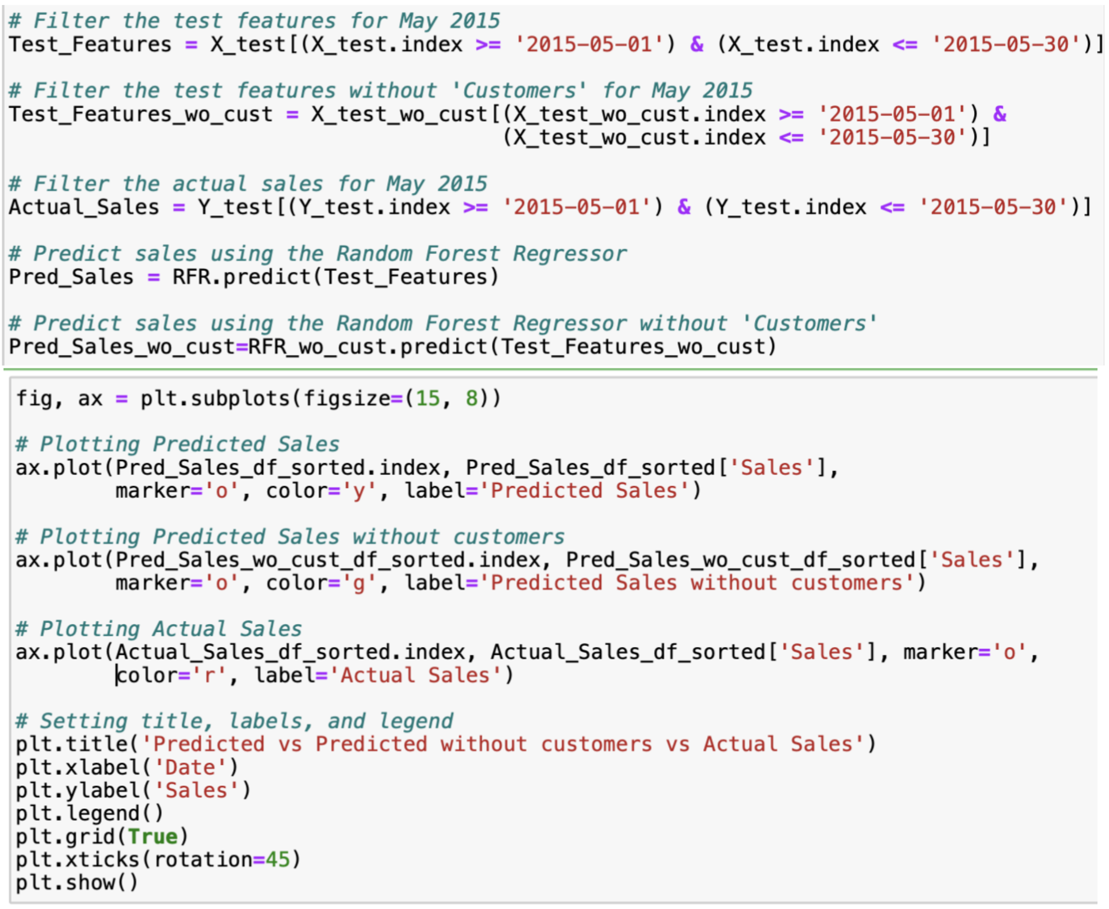
```python
X_train, X_test, Y_train, Y_test = train_test_split(
    X, Y, test_size=0.25,random_state=42)

DTR = DecisionTreeRegressor(random_state=42)
DTR.fit(X_train, Y_train)
Y_pred_DTR = DTR.predict(X_test)

pd.DataFrame({
    "R^2": {
        "train": r2_score(Y_train, DTR.predict(X_train)),
        "test": r2_score(Y_test, Y_pred_DTR)
    },
    "RMSE": {
        "train": mean_squared_error(Y_train, DTR.predict(X_train), squared=False),
        "test": mean_squared_error(Y_test, Y_pred_DTR, squared=False),
    },
    "MAE": {
        "train": mean_absolute_error(Y_train, DTR.predict(X_train)),
        "test": mean_absolute_error(Y_test, Y_pred_DTR),
    },
})
```

Modelling - Decision Tree Regression



```python
X_train, X_test, Y_train, Y_test = train_test_split(
    X, Y, test_size=0.25,random_state=42)

RFR = RandomForestRegressor(n_estimators=100, random_state=42)

# Fit the Random Forest model to the training data
RFR.fit(X_train, Y_train)

# Make predictions using the Random Forest model
Y_pred_RFR = RFR.predict(X_test)

pd.DataFrame({
    "R^2": {
        "train": r2_score(Y_train, RFR.predict(X_train)),
        "test": r2_score(Y_test, Y_pred_RFR)
    },
    "RMSE": {
        "train": mean_squared_error(Y_train, RFR.predict(X_train), squared=False),
        "test": mean_squared_error(Y_test, Y_pred_RFR, squared=False),
    },
    "MAE": {
        "train": mean_absolute_error(Y_train, RFR.predict(X_train)),
        "test": mean_absolute_error(Y_test, Y_pred_RFR),
    },
})
```

Modelling - Random Forest Regression



```python
# Filter the test features for May 2015
Test_Features = X_test[(X_test.index >= '2015-05-01') & (X_test.index <= '2015-05-30')]

# Filter the test features without 'Customers' for May 2015
Test_Features_wo_cust = X_test_wo_cust[(X_test_wo_cust.index >= '2015-05-01') &
                                       (X_test_wo_cust.index <= '2015-05-30')]

# Filter the actual sales for May 2015
Actual_Sales = Y_test[(Y_test.index >= '2015-05-01') & (Y_test.index <= '2015-05-30')]

# Predict sales using the Random Forest Regressor
Pred_Sales = RFR.predict(Test_Features)

# Predict sales using the Random Forest Regressor without 'Customers'
Pred_Sales_wo_cust=RFR_wo_cust.predict(Test_Features_wo_cust)
```

```python
fig, ax = plt.subplots(figsize=(15, 8))

# Plotting Predicted Sales
ax.plot(Pred_Sales_df_sorted.index, Pred_Sales_df_sorted['Sales'],
        marker='o', color='y', label='Predicted Sales')

# Plotting Predicted Sales without customers
ax.plot(Pred_Sales_wo_cust_df_sorted.index, Pred_Sales_wo_cust_df_sorted['Sales'],
        marker='o', color='g', label='Predicted Sales without customers')

# Plotting Actual Sales
ax.plot(Actual_Sales_df_sorted.index, Actual_Sales_df_sorted['Sales'], marker='o',
        color='r', label='Actual Sales')

# Setting title, labels, and legend
plt.title('Predicted vs Predicted without customers vs Actual Sales')
plt.xlabel('Date')
plt.ylabel('Sales')
plt.legend()
plt.grid(True)
plt.xticks(rotation=45)
plt.show()
```

Modelling - Prediction for May of 2015