DATA71011
Coursework Project

Student #: 11356590

I. Introduction

Sales forecasting is one of the most challenging issues that the retail industry faces. This is because various factors influence sales, such as promotional activities, competition, holidays, seasonality, and geographical location. To tackle this challenge, I employed statistical and machine-learning methods to analyze years of historical data. The objective of the essay is to predict the daily sales of a chain of drug stores in Germany, which has more than 1000 stores, from August 1st to September 17th, 2015. The performance of different analytical methods will be compared using Root Mean Square Percentage Error (RMSPE) to assess the proportion of predicted values to actual values. A reliable sales forecast can assist store managers in devising tailored marketing strategies for each store, thereby amplifying overall productivity, profitability, and ultimately, customer satisfaction.

II. Methodology

1. Data Selection and Preparation

During the data preprocessing phase, I recognized that sales patterns are likely to be more similar within the same date range. Hence, I initially selected a date range from the training data that closely matched the dates in the test data. After conducting several tests and comparisons, I ultimately settled on the period from August 1st to September 15th, which comprised a total of 98,400 records.

2. Data Merging and Filtering

Next, I merged this new training dataset (new_train) with the store information table to enrich the dataset with additional store-related information. Subsequently, I filtered out the data for stores that were closed during this period. Stores that were closed during this period would have had zero customers and sales, making them unnecessary for prediction.

3. Feature Processing and Adjustment

For the merged dataset, I removed some unnecessary features originally present in the store information table, such as "Promo2", "Promo2SinceWeek", "Promo2SinceYear", and "PromoInterval". These features were redundant as the "Promo" field in the new dataset provided similar information. Additionally, I made adjustments to the information regarding competitors. I set up a new column called "hasCompetition" to indicate whether there were competitors present on a given date. Moreover, for missing values in the "CompetitionDistance" column, I adjusted them to the maximum value in that column, as competitors located very far away would likely have minimal impact on sales.

4. Holiday Labeling

During the data processing, I observed that the "StateHoliday" column contained representations of various holidays. Considering that the types of holidays encountered during the prediction period are likely to be similar each year, I converted the data in this column to distinguish between holidays (1) and non-holidays (0).

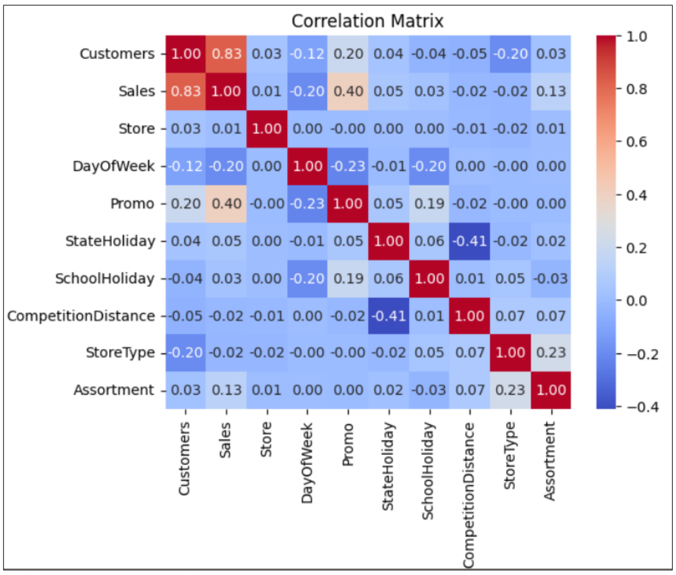
5. Data Encoding

Finally, I utilized both One-Hot Encoding and Label Encoding techniques to transform the ['StoreType', 'Assortment'] columns. This encoding was implemented to facilitate exploratory data analysis (EDA) and machine learning model training, ensuring that the data format adheres to the requirements of the model.

III. Exploratory Data Analysis (EDA)

Before incorporating machine learning models, I conducted exploratory data analysis (EDA) to understand which features are related to the predicted sales and customer count, and to identify any features that exhibit high similarity and could potentially be removed. Therefore, I plotted scatter plots for each feature against "Customers" or "Sales" but did not find any feature values that were significantly correlated with "Customers" or "Sales". Scatter plots are included in the appendix to illustrate the encoded data's distribution and relationships.

I also created a Correlation Matrix heatmap to examine the relationships between features. Except for the "Promo" feature, which showed a slightly higher correlation, other features exhibited a low correlation with sales and customer count, with correlation coefficients close to zero.



Plot 1: Correlation Matrix

In conclusion, apart from the "Promo" feature showing a relatively higher correlation, there were no clear linear relationships observed between other features and the target variables. This suggests that there may not be direct linear relationships between the features and the target variables, or the correlations are weak.

Based on this analysis, it appears that most features in the dataset may not have direct linear relationships with the target variables. Further exploration of nonlinear relationships or alternative feature selection methods may be necessary to enhance model performance and interpretability. Techniques such as feature combinations and dimensionality reduction may need to be considered to fully exploit the potential of the data and improve prediction accuracy.

IV. Machine Learning Model Performance

In selecting machine learning models, I considered four classical regression models: Linear Regression, Random Forest, Decision Tree, and Gradient Boosting. For the decision tree-based models (Random Forest and Decision Tree), I employed two different feature encoding methods: Label Encoding and One-Hot Encoding. The final selection of features for the models included 'hasCompetition', 'DayOfWeek', 'Open', 'Promo', 'StateHoliday', 'SchoolHoliday', and 'CompetitionDistance'. In the One-Hot Encoding process for 'StoreType' and 'Assortment', these features were expanded into 'StoreType_a', 'StoreType_b', 'StoreType_c', 'StoreType_d', 'Assortment_a', 'Assortment_b', and 'Assortment_c', allowing the models to better understand the categorical data by treating each category as a separate feature.

The reason for this approach lies in the fact that decision tree-based models split features based on different categories rather than their magnitude. Label Encoding converts categorical features into continuous numbers, allowing the model to better understand the relative relationships between features. On the other hand, One-Hot Encoding expresses the independence between categories more effectively, preventing the model from forming incorrect biases when handling categorical features. By employing both encoding methods simultaneously, I aimed to compare their effects on the performance of decision tree-based models and select the most suitable encoding method for the dataset and models. This approach enhances model accuracy, robustness, and adaptability to different types of feature data.

The RMSPE values for each model under both encoding methods are summarized in the following table:

Model	**Encoding Method**	**RMSPE for Customers Model**	**RMSPE for Sales Model**
Linear Regression	One-Hot Encoding	51.37	50.19
Random Forest Regressor	One-Hot Encoding	17.47	23.91
Decision Tree Regressor	One-Hot Encoding	19.11	24.61
Gradient Boosting Regressor	One-Hot Encoding	45.58	47.58
Random Forest Regressor	Label Encoding	17.43	24.14
Decision Tree Regressor	Label Encoding	18.25	24.51

In this table, a comparative view of the RMSPE values for each model under different encoding methods is provided, aiding in model performance assessment and decision-making. It can be observed that the Random Forest Regressor performs the best across both Label Encoding and One-Hot Encoding methods. Therefore, the Random Forest Regressor was selected as the final model. While the results between Label Encoding and One-Hot Encoding are very close, a slightly better performance is noticed with the One-Hot Encoding method for tree-based models. Hence, One-Hot En
coding was chosen as the preferred encoding method.

This outcome aligns with the initial findings from our exploratory data analysis, specifically the Correlation Matrix, which indicated that the selected features did not exhibit strong linear correlations with the 'Customers' and 'Sales' columns. This lack of linear relationship underscores the superior performance of tree-based models, such as the Random Forest Regressor, which are better equipped to handle complex, nonlinear interactions between features. Therefore, the choice of tree-based models, supported by the most effective encoding method, One-Hot Encoding, is substantiated by both the EDA and the subsequent model performance evaluation, highlighting the importance of understanding the underlying data characteristics in guiding the model selection process.

V. Conclusion

Using Root Mean Square Percentage Error (RMSPE) to evaluate machine learning models provides insights into their performance. Among the different encoding methods, namely Label Encoding and One-Hot Encoding, the Random Forest regression model exhibited the most outstanding performance. Specifically, under One-Hot Encoding, the Random Forest regression model achieved an RMSPE of 17.47 for the Customers model and 23.91 for the Sales model, outperforming other models such as Linear Regression, Decision Tree regression, and Gradient Boosting regression models.

Although the results between Label Encoding and One-Hot Encoding are similar, slightly better performance was observed under One-Hot Encoding, especially for tree-based models like Random Forest and Decision Tree. This suggests that One-Hot Encoding may provide a more suitable representation of categorical features for these models.

Additionally, it's worth noting that the data used for this analysis was derived from the merged dataset obtained by combining the test.csv and store.csv files using the methodology described earlier. The resulting dataset, referred to as “new_train”, served as the basis for model training and evaluation. Details regarding the data preprocessing steps and model training process can be found in the respective sections of this report.

VI. Appendix

In [31]:
```python
import pandas as pd

directory = "./"
stores = pd.read_csv(directory + "store.csv")
trains = pd.read_csv(directory + "train.csv")
tests = pd.read_csv(directory  + "test.csv")

stores
```

```
C:\Users\user\AppData\Local\Temp\ipykernel_20348\3014701360.py:5: DtypeWarning: Columns (7) have mixed type
s. Specify dtype option on import or set low_memory=False.
  trains = pd.read_csv(directory + "train.csv")
```

Out[31]:

	Store	StoreType	Assortment	CompetitionDistance	CompetitionOpenSinceMonth	CompetitionOpenSinceYear	Promo2
0	1	c	a	1270.0	9.0	2008.0	0
1	2	a	a	570.0	11.0	2007.0	1
2	3	a	a	14130.0	12.0	2006.0	1
3	4	c	c	620.0	9.0	2009.0	0
4	5	a	a	29910.0	4.0	2015.0	0
...	...	...	...	...	...	...	...
1110	1111	a	a	1900.0	6.0	2014.0	1
1111	1112	c	c	1880.0	4.0	2006.0	0
1112	1113	a	c	9260.0	NaN	NaN	0
1113	1114	a	c	870.0	NaN	NaN	0
1114	1115	d	c	5350.0	NaN	NaN	1

1115 rows × 10 columns

In [32]:
```python
trains
```

Out[32]:

	Store	DayOfWeek	Date	Sales	Customers	Open	Promo	StateHoliday	SchoolHoliday
0	1	5	31/07/2015	5263	555	1	1	0	1
1	2	5	31/07/2015	6064	625	1	1	0	1
2	3	5	31/07/2015	8314	821	1	1	0	1
3	4	5	31/07/2015	13995	1498	1	1	0	1
4	5	5	31/07/2015	4822	559	1	1	0	1
...	...	...	...	...	...	...	...	...	...
1017204	1111	2	01/01/2013	0	0	0	0	a	1
1017205	1112	2	01/01/2013	0	0	0	0	a	1
1017206	1113	2	01/01/2013	0	0	0	0	a	1
1017207	1114	2	01/01/2013	0	0	0	0	a	1
1017208	1115	2	01/01/2013	0	0	0	0	a	1

1017209 rows × 9 columns

In [33]:
```python
import pandas as pd

# Assuming the "Date" column is not already in datetime format
# Convert the "Date" column to datetime format
trains["Date"] = pd.to_datetime(trains["Date"], format="%d/%m/%Y", errors='coerce')

# Set the filtering conditions for all years
start_date = pd.to_datetime("08/01", format="%m/%d")
```

```
end_date = pd.to_datetime("09/17", format="%m/%d")

# Filter the data
new_trains = trains[(trains["Date"].dt.month == start_date.month) |
                    ((trains["Date"].dt.month == end_date.month) & (trains["Date"].dt.day <= end_date.c
                    
# Display the filtered data
new_trains
```

Out[33]:

	Store	DayOfWeek	Date	Sales	Customers	Open	Promo	StateHoliday	SchoolHoliday
334555	1	3	2014-09-17	4383	490	1	1	0	0
334556	2	3	2014-09-17	6469	762	1	1	0	0
334557	3	3	2014-09-17	8034	795	1	1	0	0
334558	4	3	2014-09-17	8594	1173	1	1	0	0
334559	5	3	2014-09-17	5685	651	1	1	0	0
...	...	...	...	...	...	...	...	...	...
780825	1111	4	2013-08-01	5733	487	1	1	0	1
780826	1112	4	2013-08-01	12394	866	1	1	0	1
780827	1113	4	2013-08-01	8504	864	1	1	0	1
780828	1114	4	2013-08-01	24140	4022	1	1	0	0
780829	1115	4	2013-08-01	5745	378	1	1	0	1

98400 rows × 9 columns

In [34]:

```
import pandas as pd


# merge two DataFrame，use "Store" to connect
merged_df = pd.merge(new_trains, stores, on="Store", how="left")

# remove close stores
merged_df = merged_df[merged_df["Open"]==1]

merged_df
```

Out[34]:

	Store	DayOfWeek	Date	Sales	Customers	Open	Promo	StateHoliday	SchoolHoliday	StoreType	Assortment	Con
0	1	3	2014-09-17	4383	490	1	1	0	0	c	a	
1	2	3	2014-09-17	6469	762	1	1	0	0	a	a	
2	3	3	2014-09-17	8034	795	1	1	0	0	a	a	
3	4	3	2014-09-17	8594	1173	1	1	0	0	c	c	
4	5	3	2014-09-17	5685	651	1	1	0	0	a	a	
...	...	...	...	...	...	...	...	...	...	...	...	
98395	1111	4	2013-08-01	5733	487	1	1	0	1	a	a	
98396	1112	4	2013-08-01	12394	866	1	1	0	1	c	c	
98397	1113	4	2013-08-01	8504	864	1	1	0	1	a	c	
98398	1114	4	2013-08-01	24140	4022	1	1	0	0	a	c	
98399	1115	4	2013-08-01	5745	378	1	1	0	1	d	c	

84089 rows × 18 columns

In [35]:
```
# merged_df.to_excel(directory+"merged_data.xlsx", index=False)
```

Cleaning data

In [36]:
```
# Delete the column related to Promo2
### because we can determine whether a promotion is happening from the 'Promo' column.
merged_df.drop(["Promo2", "Promo2SinceWeek", "Promo2SinceYear", "PromoInterval"], axis=1, inplace=True)
```

In [37]:
```
import pandas as pd
# Fill missing values with '2013-12-24'
merged_df['CompetitionOpenSinceYear'].fillna(2013, inplace=True)
merged_df['CompetitionOpenSinceMonth'].fillna(12, inplace=True)

# Convert 'CompetitionOpenSinceYear' and 'CompetitionOpenSinceMonth' to integers
merged_df['CompetitionOpenSinceYear'] = merged_df['CompetitionOpenSinceYear'].astype(int)
merged_df['CompetitionOpenSinceMonth'] = merged_df['CompetitionOpenSinceMonth'].astype(int)

# Combine 'CompetitionOpenSinceYear' and 'CompetitionOpenSinceMonth' into a new column 'CompetitionOpenDate
merged_df['CompetitionOpenDate'] = pd.to_datetime(merged_df['CompetitionOpenSinceYear'].astype(str) +
                                                  '-' +
                                                  merged_df['CompetitionOpenSinceMonth'].astype(str) +
                                                  '-1',
                                                  format='%Y-%m-%d')

# Delete the original 'CompetitionOpenSinceYear' and 'CompetitionOpenSinceMonth' columns
merged_df.drop(['CompetitionOpenSinceYear', 'CompetitionOpenSinceMonth'], axis=1, inplace=True)

# Create a new column 'hasCompetition' based on the comparison
merged_df['hasCompetition'] = 1  # Initially set to 1

# Compare 'CompetitionOpenDate' with 'Date' and update 'hasCompetition' accordingly
merged_df.loc[merged_df['CompetitionOpenDate'] > merged_df['Date'], 'hasCompetition'] = 0


# Find the maximum value in the 'CompetitionDistance' column
```

```
max_distance = merged_df['CompetitionDistance'].max()

# Fill missing values in the 'CompetitionDistance' column with the maximum value
merged_df['CompetitionDistance'].fillna(max_distance, inplace=True)

# Replace 'CompetitionDistance' values with the maximum value where 'hasCompetition' is 0
merged_df.loc[merged_df['hasCompetition'] == 0, 'CompetitionDistance'] = max_distance
```

- Add a 'hasCompetition' column to aid in determining whether a store had competitors during a specific period. => for linear model
- Modify the 'CompetitionDistance' column, setting the competition distance to the maximum value for stores without competitors. => for tree model

In [38]:
```
# Set non-zero values in the 'StateHoliday' column to 1
merged_df["StateHoliday"] = merged_df["StateHoliday"].apply(lambda x: 1 if x != "0" else 0)

# Convert the column's data type to integer (int)
merged_df["StateHoliday"] = merged_df["StateHoliday"].astype(int)
```

In [39]:
```
# Creating one-hot encoded DataFrame
encoded_df = pd.get_dummies(merged_df[['StoreType', 'Assortment']], drop_first=False)

# Concatenating the encoded DataFrame with the original DataFrame
merged_df= pd.concat([merged_df, encoded_df], axis=1)
```

In [40]:
```
from sklearn.preprocessing import LabelEncoder

# Initialize the LabelEncoder
label_encoder = LabelEncoder()

# Apply Label Encoding to 'StoreType' and 'Assortment' columns
merged_df['StoreType'] = label_encoder.fit_transform(merged_df['StoreType'])
merged_df['Assortment'] = label_encoder.fit_transform(merged_df['Assortment'])
```

- hasCompetition,StoreType_a, StoreType_b, StoreType_c, StoreType_d, Assortment_a, Assortment_b, Assortment_c for linear model, such as linear regression, logistic regression
- CompetitionDistance, StoreType, Assortment for tree model, such as Decision Tree, Random Forest ,Gradient Boosting

EDA

In [41]:
```
# check no missing data
merged_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 84089 entries, 0 to 98399
Data columns (total 21 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   Store                84089 non-null  int64
 1   DayOfWeek            84089 non-null  int64
 2   Date                 84089 non-null  datetime64[ns]
 3   Sales                84089 non-null  int64
 4   Customers            84089 non-null  int64
 5   Open                 84089 non-null  int64
 6   Promo                84089 non-null  int64
 7   StateHoliday         84089 non-null  int32
 8   SchoolHoliday        84089 non-null  int64
 9   StoreType            84089 non-null  int32
 10  Assortment           84089 non-null  int32
 11  CompetitionDistance  84089 non-null  float64
 12  CompetitionOpenDate  84089 non-null  datetime64[ns]
 13  hasCompetition       84089 non-null  int64
 14  StoreType_a          84089 non-null  uint8
 15  StoreType_b          84089 non-null  uint8
 16  StoreType_c          84089 non-null  uint8
 17  StoreType_d          84089 non-null  uint8
 18  Assortment_a         84089 non-null  uint8
 19  Assortment_b         84089 non-null  uint8
 20  Assortment_c         84089 non-null  uint8
dtypes: datetime64[ns](2), float64(1), int32(3), int64(8), uint8(7)
memory usage: 9.2 MB
```

In [42]:

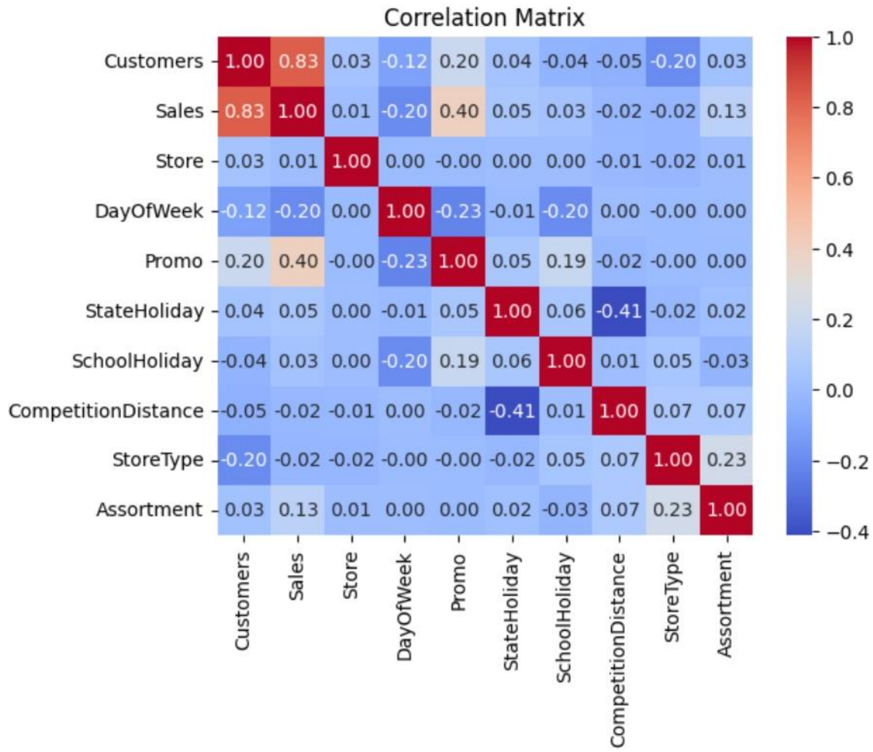
```
merged_df.describe()
```

Out[42]:

	Store	DayOfWeek	Sales	Customers	Open	Promo	StateHoliday	SchoolHoliday	StoreTy
count	84089.000000	84089.000000	84089.000000	84089.000000	84089.0	84089.000000	84089.000000	84089.000000	84089.000
mean	559.226427	3.518546	6664.188265	753.118184	1.0	0.433660	0.457040	0.454328	1.1892
std	321.975621	1.742222	2923.732708	394.143151	0.0	0.495582	0.498154	0.497913	1.3574
min	1.000000	1.000000	0.000000	0.000000	1.0	0.000000	0.000000	0.000000	0.000
25%	282.000000	2.000000	4675.000000	512.000000	1.0	0.000000	0.000000	0.000000	0.000
50%	558.000000	4.000000	6124.000000	669.000000	1.0	0.000000	0.000000	0.000000	0.000
75%	839.000000	5.000000	8002.000000	881.000000	1.0	1.000000	1.000000	1.000000	3.000
max	1115.000000	7.000000	33913.000000	5145.000000	1.0	1.000000	1.000000	1.000000	3.000

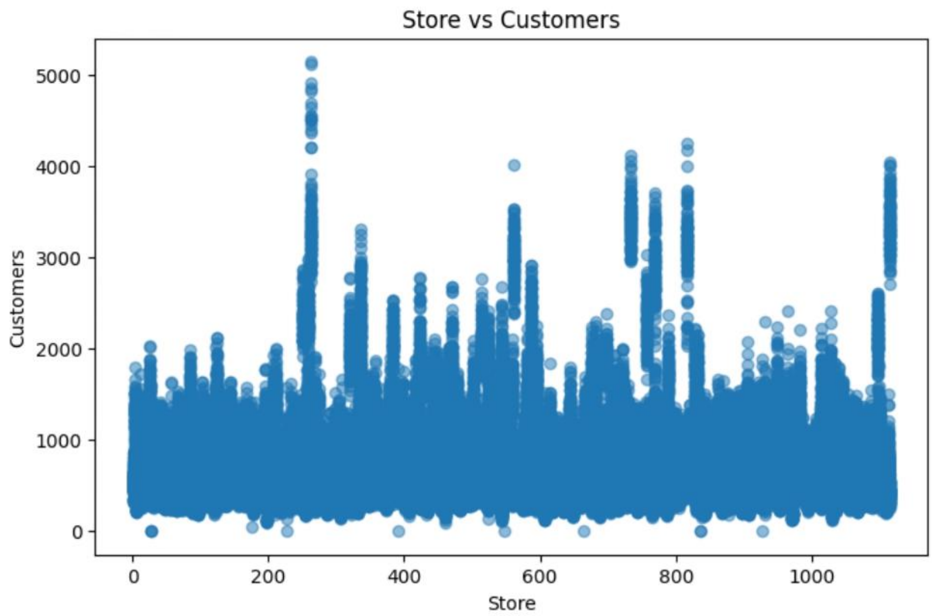
In [43]:

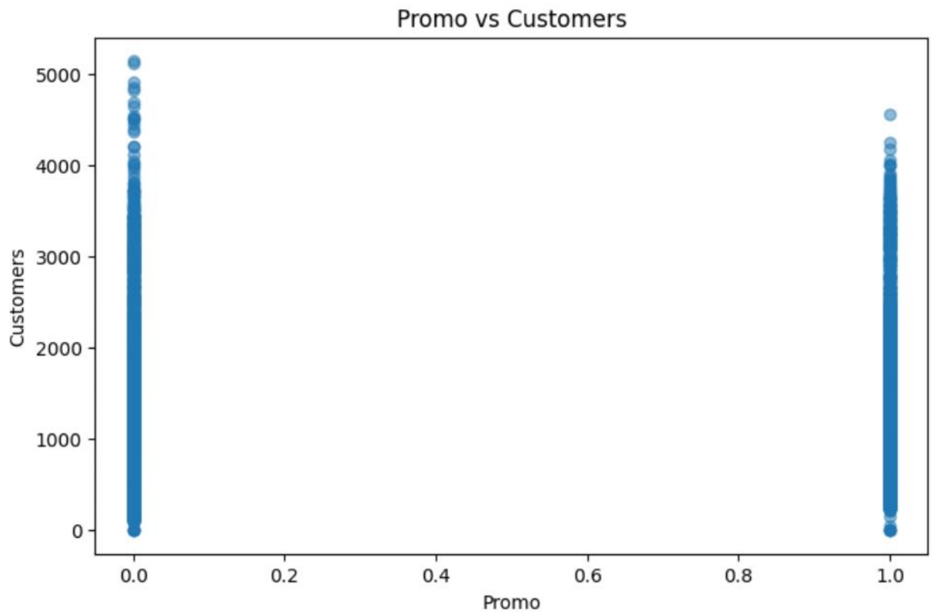
```
import seaborn as sns
import matplotlib.pyplot as plt

# Calculate the correlation matrix for selected columns
correlation_matrix = merged_df[['Customers', 'Sales', 'Store', 'DayOfWeek', 'Promo', 'StateHoliday',
                                'SchoolHoliday', 'CompetitionDistance',
                                'StoreType', 'Assortment']].corr()

# Visualize the correlation matrix using a heatmap

# Create a heatmap using seaborn with annotations
sns.heatmap(correlation_matrix, annot=True, fmt=".2f", cmap='coolwarm')

# Set the title for the heatmap
plt.title('Correlation Matrix')

# Display the heatmap
plt.show()
```

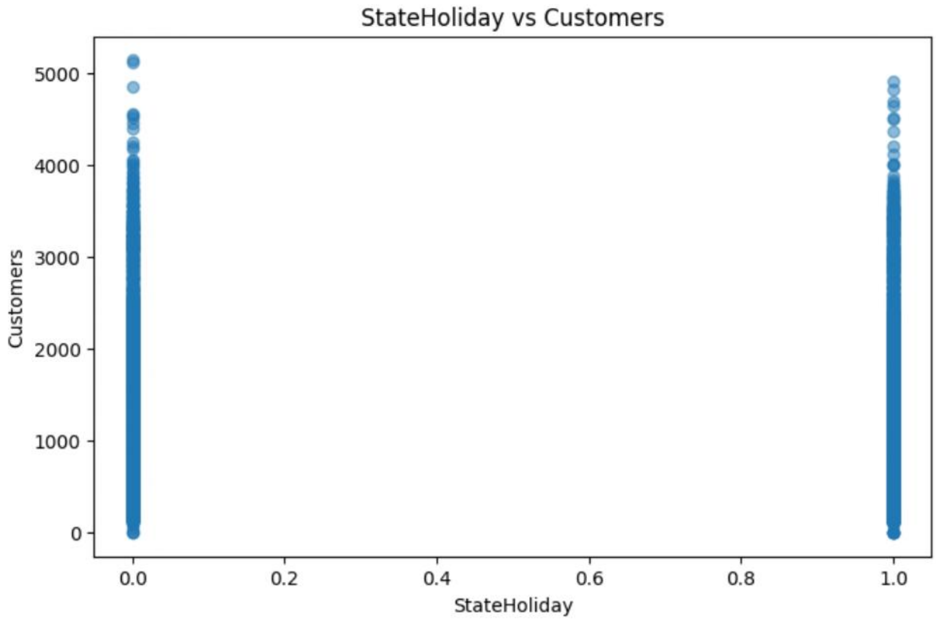


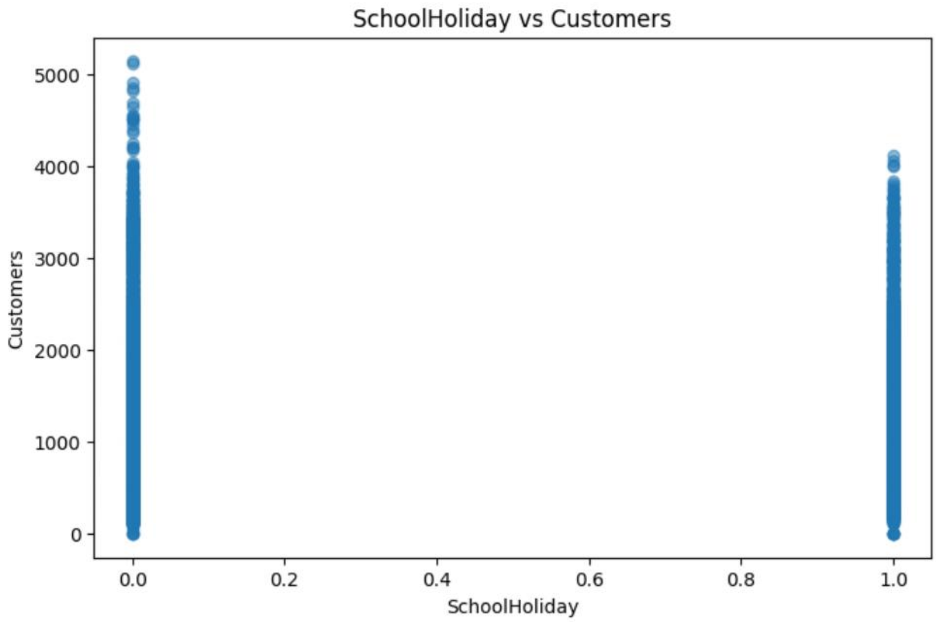
In [44]:

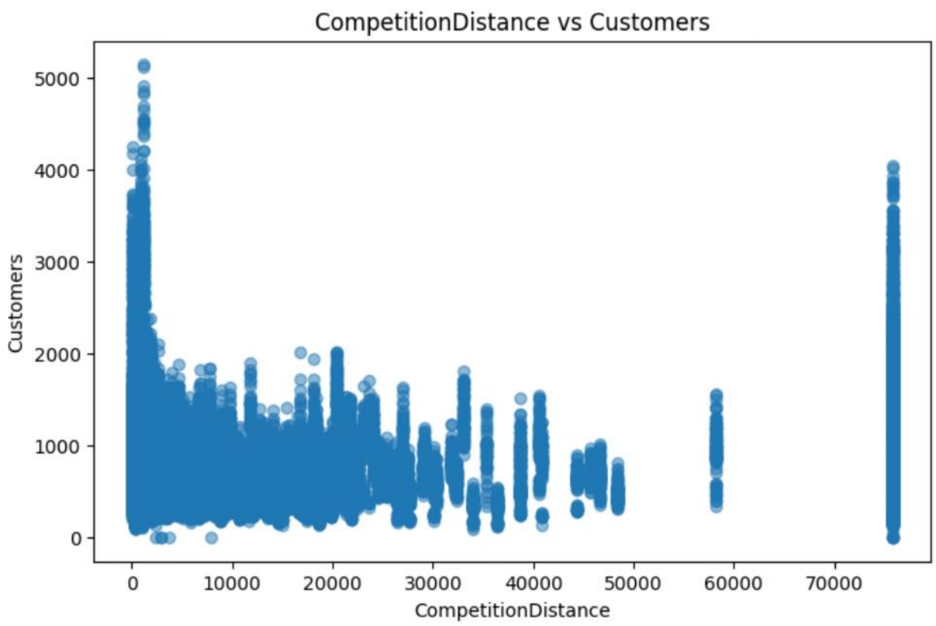
```python
# Scatter plots of features vs. Customers and features vs. Sales

# List of features to visualize
features = ['Store', 'Promo', 'StateHoliday', 'SchoolHoliday', 'CompetitionDistance', 'StoreType', 'Assortm

# Iterate through each feature and create scatter plots
for feature in features:
    # Create a new figure with a specific size
    plt.figure(figsize=(8, 5))

    # Scatter plot of the feature against 'Customers'
    plt.scatter(merged_df[feature], merged_df['Customers'], alpha=0.5)

    # Set the title for the plot
    plt.title(f'{feature} vs Customers')

    # Label the x-axis and y-axis
    plt.xlabel(feature)
    plt.ylabel('Customers')

    # Display the scatter plot
    plt.show()

# Repeat the same process for 'Sales'
for feature in features:
    plt.figure(figsize=(8, 5))
    plt.scatter(merged_df[feature], merged_df['Sales'], alpha=0.5)
    plt.title(f'{feature} vs Sales')
    plt.xlabel(feature)
    plt.ylabel('Sales')
    plt.show()
```


Store vs Customers
Customers
Store
5000
4000
3000
2000
1000
0
0
200
400
600
800
1000

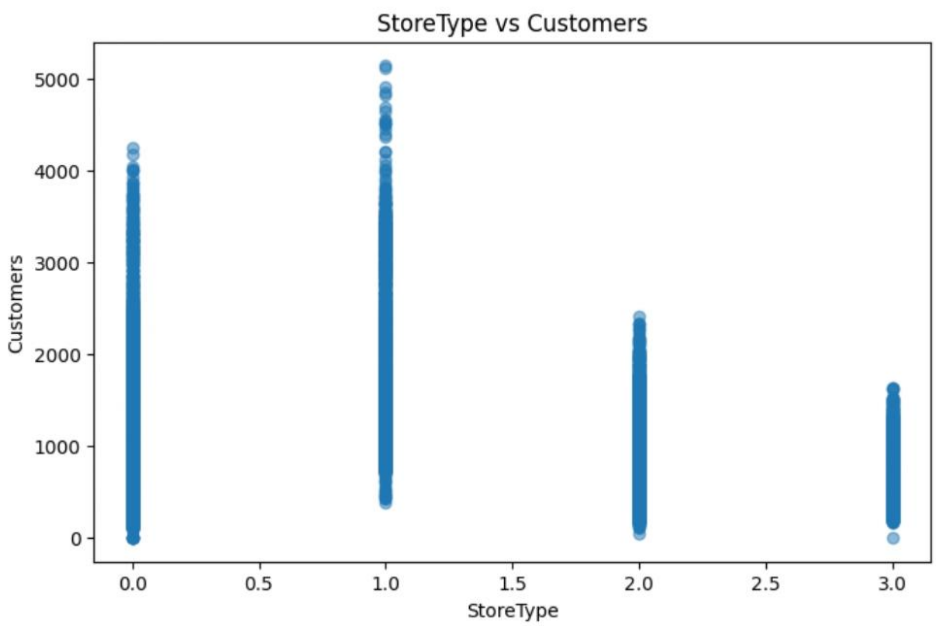

Promo vs Customers
Customers
Promo
5000
4000
3000
2000
1000
0
0.0
0.2
0.4
0.6
0.8
1.0


StateHoliday vs Customers
Customers
StateHoliday
0
1000
2000
3000
4000
5000
0.0
0.2
0.4
0.6
0.8
1.0

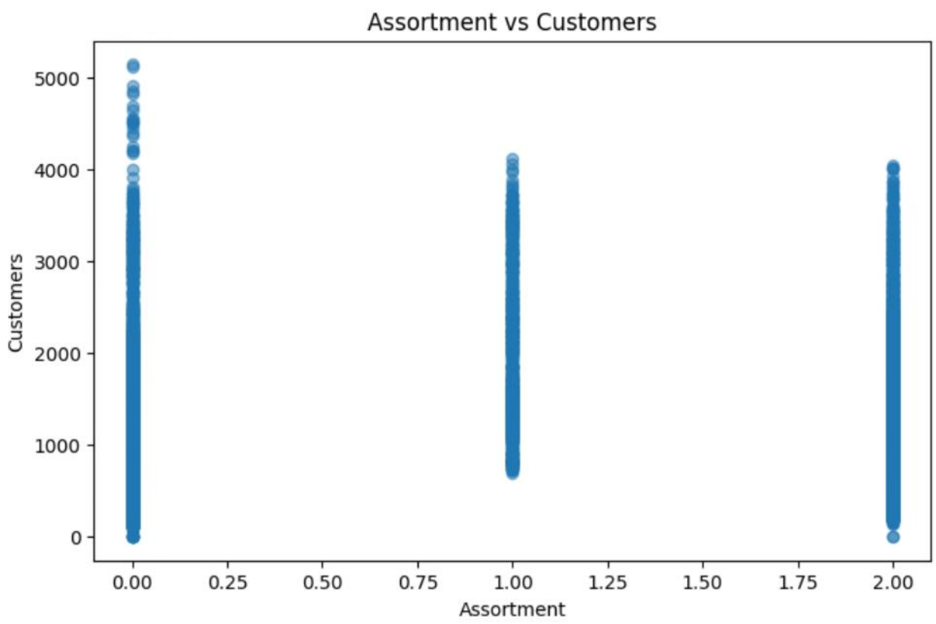

SchoolHoliday vs Customers
Customers
SchoolHoliday
0
1000
2000
3000
4000
5000
0.0
0.2
0.4
0.6
0.8
1.0

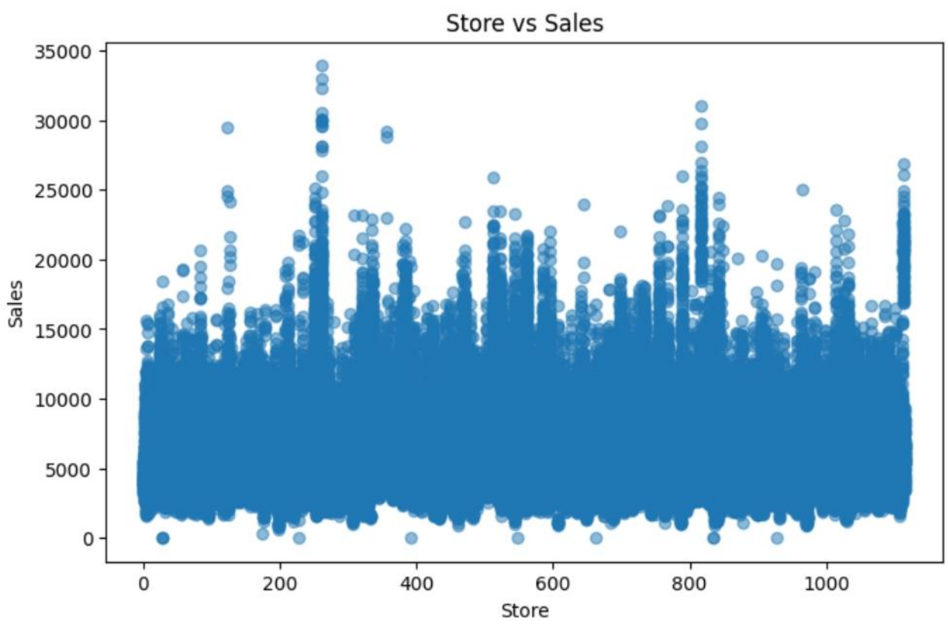


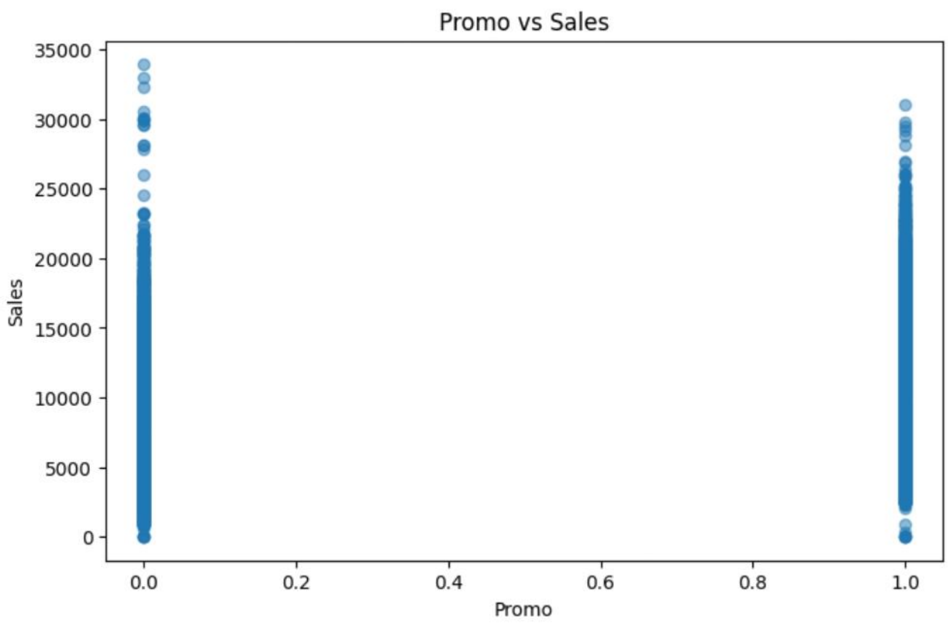
CompetitionDistance vs Customers
Customers
CompetitionDistance
0
1000
2000
3000
4000
5000
10000
20000
30000
40000
50000
60000
70000

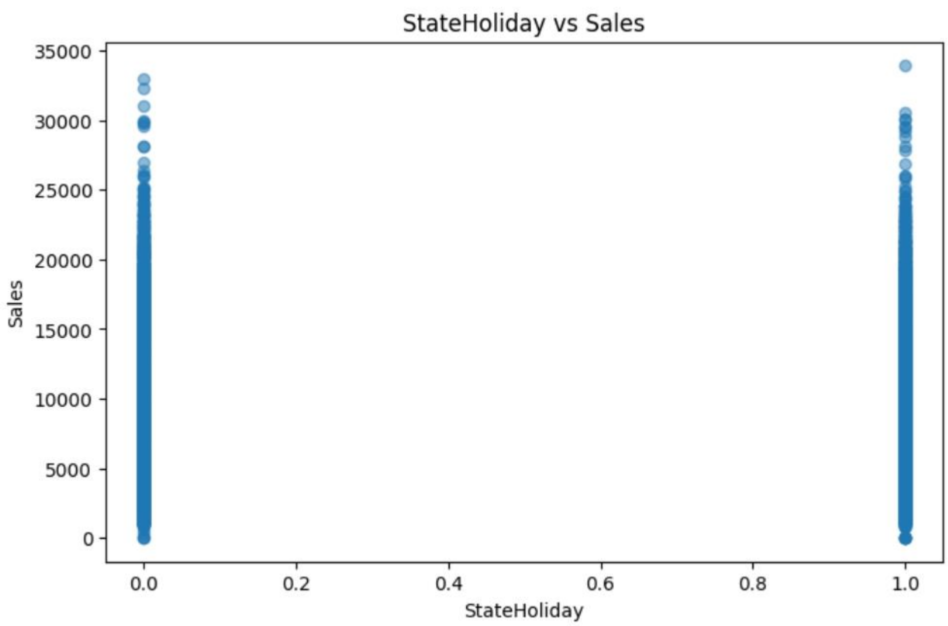


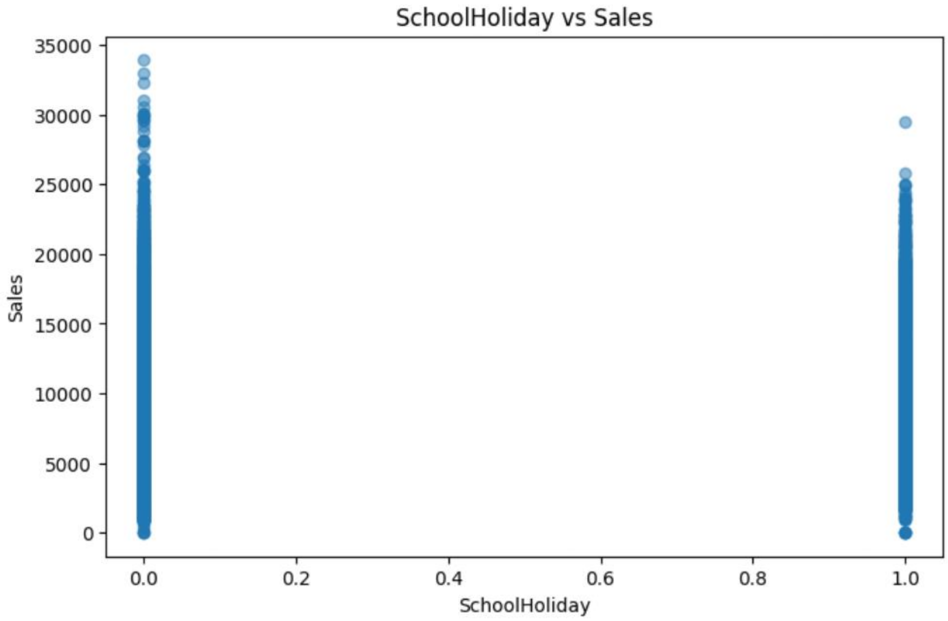
StoreType vs Customers
Customers
StoreType
0
1000
2000
3000
4000
5000
0.0
0.5
1.0
1.5
2.0
2.5
3.0


Assortment vs Customers
Customers
Assortment
5000
4000
3000
2000
1000
0
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00

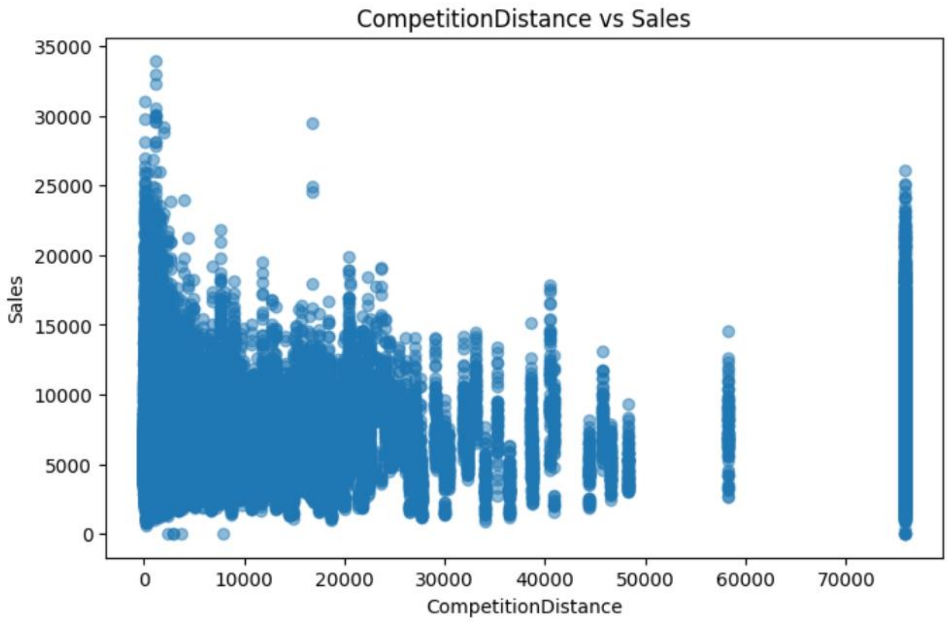

Store vs Sales
Sales
Store
35000
30000
25000
20000
15000
10000
5000
0
0
200
400
600
800
1000


Promo vs Sales
35000
30000
25000
20000
15000
10000
5000
0
Sales
0.0
0.2
0.4
0.6
0.8
1.0
Promo

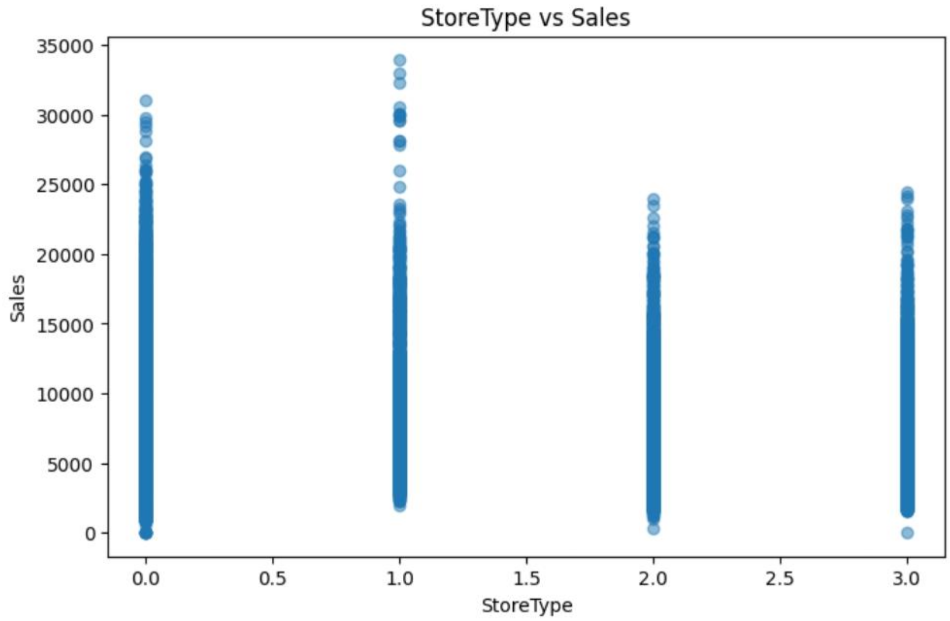

StateHoliday vs Sales
35000
30000
25000
20000
15000
10000
5000
0
Sales
0.0
0.2
0.4
0.6
0.8
1.0
StateHoliday

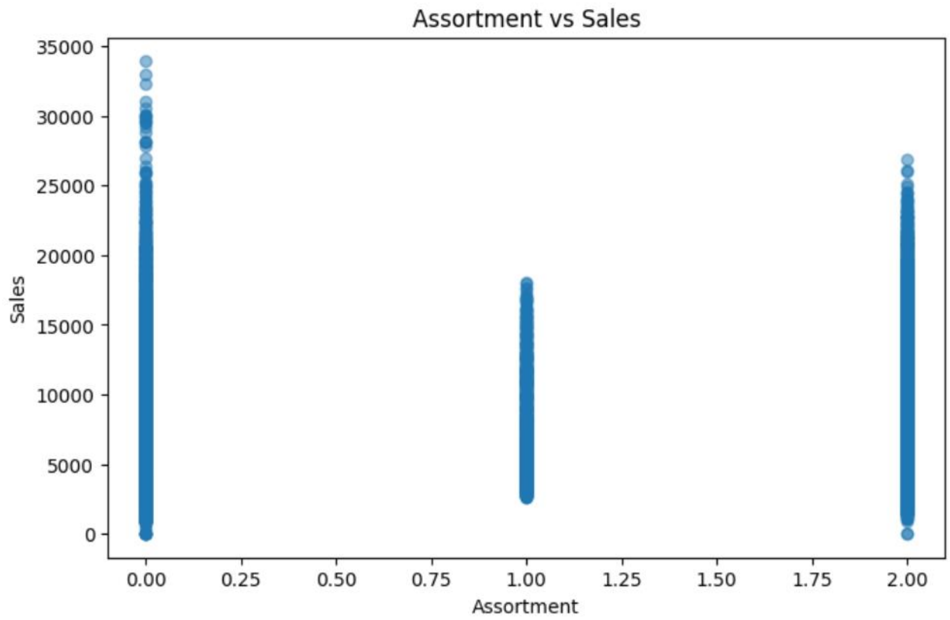


SchoolHoliday vs Sales
Sales
SchoolHoliday
35000
30000
25000
20000
15000
10000
5000
0
0.0
0.2
0.4
0.6
0.8
1.0



CompetitionDistance vs Sales
Sales
CompetitionDistance
35000
30000
25000
20000
15000
10000
5000
0
0
10000
20000
30000
40000
50000
60000
70000





Model training

In [45]:
```
# calculate_rmspe
import numpy as np

def calculate_rmspe(y_true, y_pred):
    epsilon = 1e-8
    rmspe = np.sqrt(np.mean(np.square((y_true - y_pred) / (y_true + epsilon)))) * 100
    return rmspe
```

One hot encoding features

In [46]:
```
from sklearn.model_selection import train_test_split

# Define the features (X) and target variables (y1 for Customers, y2 for Sales)
X = merged_df[['hasCompetition', 'StoreType_a', 'StoreType_b', 'StoreType_c', 'StoreType_d','Store',
               'Assortment_a', 'Assortment_b', 'Assortment_c', 'DayOfWeek',
               'Open', 'Promo', 'StateHoliday', 'SchoolHoliday', "CompetitionDistance"]]
y1 = merged_df['Customers']
y2 = merged_df['Sales']

# Split the data into training and testing sets (80% training, 20% testing)
X_train, X_test, y1_train, y1_test, y2_train, y2_test = train_test_split(X, y1, y2, test_size=0.2, random_s
```

In [47]:
```
#linear regression
from sklearn.linear_model import LinearRegression
import numpy as np

# Create two separate linear regression models
model_customers = LinearRegression()
model_sales = LinearRegression()

# Train the models on the training data
model_customers.fit(X_train, y1_train)
model_sales.fit(X_train, y2_train)

# Make predictions on the test data
y1_pred = model_customers.predict(X_test)
y2_pred = model_sales.predict(X_test)

# Calculate Root Mean Square Percentage Error (RMSPE) for both models
rmspe_customers = calculate_rmspe(y1_test, y1_pred)
rmspe_sales = calculate_rmspe(y2_test, y2_pred)

# Print the RMSPE for both models rounded to two decimal places
print("Linear regression")
print("RMSPE for Customers Model: {:.2f}".format(rmspe_customers))
print("RMSPE for Sales Model: {:.2f}".format(rmspe_sales))
```

```
Linear regression
RMSPE for Customers Model: 51.37
RMSPE for Sales Model: 50.19
```

In [48]:
```
# RandomForestRegressor
import numpy as np
from sklearn.ensemble import RandomForestRegressor

# Create two separate Random Forest models
model_customers = RandomForestRegressor(random_state=42)
model_sales = RandomForestRegressor(random_state=42)

# Train the models on the training data
model_customers.fit(X_train, y1_train)
model_sales.fit(X_train, y2_train)

# Make predictions on the test data
y1_pred = model_customers.predict(X_test)
y2_pred = model_sales.predict(X_test)

# Calculate Root Mean Square Percentage Error (RMSPE) for both models
rmspe_customers = calculate_rmspe(y1_test, y1_pred)
rmspe_sales = calculate_rmspe(y2_test, y2_pred)

# Print the RMSPE for both models rounded to two decimal places
print("Random Forest Regressor")
print("RMSPE for Customers Model: {:.2f}".format(rmspe_customers))
print("RMSPE for Sales Model: {:.2f}".format(rmspe_sales))
```

```
Random Forest Regressor
RMSPE for Customers Model: 17.47
RMSPE for Sales Model: 23.91
```

In [49]:
```
# DecisionTreeRegressor
import numpy as np
from sklearn.tree import DecisionTreeRegressor
```

```
# Create two separate Decision Tree models
model_customers = DecisionTreeRegressor(random_state=42)
model_sales = DecisionTreeRegressor(random_state=42)

# Train the models on the training data
model_customers.fit(X_train, y1_train)
model_sales.fit(X_train, y2_train)

# Make predictions on the test data
y1_pred = model_customers.predict(X_test)
y2_pred = model_sales.predict(X_test)

# Calculate Root Mean Square Percentage Error (RMSPE) for both models
rmspe_customers = calculate_rmspe(y1_test, y1_pred)
rmspe_sales = calculate_rmspe(y2_test, y2_pred)

# Print the RMSPE for both models rounded to two decimal places
print("Decision Tree Regressor")
print("RMSPE for Customers Model: {:.2f}".format(rmspe_customers))
print("RMSPE for Sales Model: {:.2f}".format(rmspe_sales))
```

```
Decision Tree Regressor
RMSPE for Customers Model: 19.11
RMSPE for Sales Model: 24.61
```

In [50]:

```
# GradientBoostingRegressor
import numpy as np
from sklearn.ensemble import GradientBoostingRegressor


# Create two separate Gradient Boosting models
model_customers = GradientBoostingRegressor(random_state=42)
model_sales = GradientBoostingRegressor(random_state=42)

# Train the models on the training data
model_customers.fit(X_train, y1_train)
model_sales.fit(X_train, y2_train)

# Make predictions on the test data
y1_pred = model_customers.predict(X_test)
y2_pred = model_sales.predict(X_test)

# Calculate Root Mean Square Percentage Error (RMSPE) for both models
rmspe_customers = calculate_rmspe(y1_test, y1_pred)
rmspe_sales = calculate_rmspe(y2_test, y2_pred)

# Print the RMSPE for both models rounded to two decimal places
print("GradientBoostingRegressor")
print("RMSPE for Customers Model: {:.2f}".format(rmspe_customers))
print("RMSPE for Sales Model: {:.2f}".format(rmspe_sales))
```

```
GradientBoostingRegressor
RMSPE for Customers Model: 45.58
RMSPE for Sales Model: 47.58
```

Label encoding features

In [51]:

```
from sklearn.model_selection import train_test_split

# Define the features (X) and target variables (y1 for Customers, y2 for Sales)
X = merged_df[['hasCompetition', 'StoreType', 'Assortment', 'Store', 'DayOfWeek',
               'Open', 'Promo', 'StateHoliday', 'SchoolHoliday', "CompetitionDistance"]]
y1 = merged_df['Customers']
y2 = merged_df['Sales']

# Split the data into training and testing sets (80% training, 20% testing)
X_train, X_test, y1_train, y1_test, y2_train, y2_test = train_test_split(X, y1, y2, test_size=0.2, random_s
```

In [52]:

```
# RandomForestRegressor

import numpy as np
from sklearn.ensemble import RandomForestRegressor

# Create two separate Random Forest models
```

```python
model_customers = RandomForestRegressor(random_state=42)
model_sales = RandomForestRegressor(random_state=42)

# Train the models on the training data
model_customers.fit(X_train, y1_train)
model_sales.fit(X_train, y2_train)

# Make predictions on the test data
y1_pred = model_customers.predict(X_test)
y2_pred = model_sales.predict(X_test)

# Calculate Root Mean Square Percentage Error (RMSPE) for both models
rmspe_customers = calculate_rmspe(y1_test, y1_pred)
rmspe_sales = calculate_rmspe(y2_test, y2_pred)

# Print the RMSPE for both models rounded to two decimal places
print("Random Forest Regressor")
print("RMSPE for Customers Model: {:.2f}".format(rmspe_customers))
print("RMSPE for Sales Model: {:.2f}".format(rmspe_sales))
```

```
Random Forest Regressor
RMSPE for Customers Model: 17.43
RMSPE for Sales Model: 24.14
```

In [53]:
```python
# DecisionTreeRegressor

import numpy as np
from sklearn.tree import DecisionTreeRegressor

# Create two separate Decision Tree models
model_customers = DecisionTreeRegressor(random_state=42)
model_sales = DecisionTreeRegressor(random_state=42)

# Train the models on the training data
model_customers.fit(X_train, y1_train)
model_sales.fit(X_train, y2_train)

# Make predictions on the test data
y1_pred = model_customers.predict(X_test)
y2_pred = model_sales.predict(X_test)

# Calculate Root Mean Square Percentage Error (RMSPE) for both models
rmspe_customers = calculate_rmspe(y1_test, y1_pred)
rmspe_sales = calculate_rmspe(y2_test, y2_pred)

# Print the RMSPE for both models rounded to two decimal places
print("Decision Tree Regressor")
print("RMSPE for Customers Model: {:.2f}".format(rmspe_customers))
print("RMSPE for Sales Model: {:.2f}".format(rmspe_sales))
```

```
Decision Tree Regressor
RMSPE for Customers Model: 18.25
RMSPE for Sales Model: 24.51
```

test datasets pre-processing

In [54]:
```python
import pandas as pd

# Merge two DataFrames, using "Store" as the key to connect them
merged_df1 = pd.merge(tests, stores, on="Store", how="left")

# Remove columns related to Promo2
# We can determine whether a promotion is happening from the 'Promo' column.
merged_df1.drop(["Promo2", "Promo2SinceWeek", "Promo2SinceYear", "PromoInterval"], axis=1, inplace=True)

import pandas as pd
# Fill missing values in 'CompetitionOpenSinceYear' and 'CompetitionOpenSinceMonth' with '2013-12-24'
merged_df1['CompetitionOpenSinceYear'].fillna(2013, inplace=True)
merged_df1['CompetitionOpenSinceMonth'].fillna(12, inplace=True)

# Convert 'CompetitionOpenSinceYear' and 'CompetitionOpenSinceMonth' to integers
merged_df1['CompetitionOpenSinceYear'] = merged_df1['CompetitionOpenSinceYear'].astype(int)
merged_df1['CompetitionOpenSinceMonth'] = merged_df1['CompetitionOpenSinceMonth'].astype(int)
```

```python
# Combine 'CompetitionOpenSinceYear' and 'CompetitionOpenSinceMonth' into a new column 'CompetitionOpenDate
merged_df1['CompetitionOpenDate'] = pd.to_datetime(merged_df1['CompetitionOpenSinceYear'].astype(str) +
                                                   '-' +
                                                   merged_df1['CompetitionOpenSinceMonth'].astype(str) +
                                                   '-1',
                                                   format='%Y-%m-%d')

# Delete the original 'CompetitionOpenSinceYear' and 'CompetitionOpenSinceMonth' columns
merged_df1.drop(['CompetitionOpenSinceYear', 'CompetitionOpenSinceMonth'], axis=1, inplace=True)

# Create a new column 'hasCompetition' based on the comparison
merged_df1['hasCompetition'] = 1  # Initially set to 1

# Compare 'CompetitionOpenDate' with 'Date' and update 'hasCompetition' accordingly
merged_df1.loc[merged_df1['CompetitionOpenDate'] > merged_df1['Date'], 'hasCompetition'] = 0

# Find the maximum value in the 'CompetitionDistance' column
max_distance = merged_df1['CompetitionDistance'].max()

# Fill missing values in the 'CompetitionDistance' column with the maximum value
merged_df1['CompetitionDistance'].fillna(max_distance, inplace=True)

# Replace 'CompetitionDistance' values with the maximum value where 'hasCompetition' is 0
merged_df1.loc[merged_df1['hasCompetition'] == 0, 'CompetitionDistance'] = max_distance

# Set non-zero values in the 'StateHoliday' column to 1
merged_df1["StateHoliday"] = merged_df1["StateHoliday"].apply(lambda x: 1 if x != "0" else 0)

# Convert the column's data type to integer (int)
merged_df1["StateHoliday"] = merged_df1["StateHoliday"].astype(int)

# Create one-hot encoded DataFrame
encoded_df = pd.get_dummies(merged_df1[['StoreType', 'Assortment']], drop_first=False)

# Concatenate the encoded DataFrame with the original DataFrame
merged_df1 = pd.concat([merged_df1, encoded_df], axis=1)

from sklearn.preprocessing import LabelEncoder

# Initialize the LabelEncoder
label_encoder = LabelEncoder()

# Apply Label Encoding to 'StoreType' and 'Assortment' columns
merged_df1['StoreType'] = label_encoder.fit_transform(merged_df1['StoreType'])
merged_df1['Assortment'] = label_encoder.fit_transform(merged_df1['Assortment'])
```

```
C:\Users\user\AppData\Local\Temp\ipykernel_20348\3504885884.py:33: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to inconsistently parsed dates! Speci
fy a format to ensure consistent parsing.
  merged_df1.loc[merged_df1['CompetitionOpenDate'] > merged_df1['Date'], 'hasCompetition'] = 0
```

In [55]:
```python
merged_df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 41088 entries, 0 to 41087
Data columns (total 21 columns):
 #   Column               Non-Null Count  Dtype         
---  ------               --------------  -----         
 0   Store                41088 non-null  int64         
 1   DayOfWeek            41088 non-null  int64         
 2   Date                 41088 non-null  object        
 3   Sales                0 non-null      float64       
 4   Customers            0 non-null      float64       
 5   Open                 41077 non-null  float64       
 6   Promo                41088 non-null  int64         
 7   StateHoliday         41088 non-null  int32         
 8   SchoolHoliday        41088 non-null  int64         
 9   StoreType            41088 non-null  int32         
 10  Assortment           41088 non-null  int32         
 11  CompetitionDistance  41088 non-null  float64       
 12  CompetitionOpenDate  41088 non-null  datetime64[ns]
 13  hasCompetition       41088 non-null  int64         
 14  StoreType_a          41088 non-null  uint8         
 15  StoreType_b          41088 non-null  uint8         
 16  StoreType_c          41088 non-null  uint8         
 17  StoreType_d          41088 non-null  uint8         
 18  Assortment_a         41088 non-null  uint8         
 19  Assortment_b         41088 non-null  uint8         
 20  Assortment_c         41088 non-null  uint8         
dtypes: datetime64[ns](1), float64(4), int32(3), int64(5), object(1), uint8(7)
memory usage: 4.5+ MB
```

In [56]:

```
merged_df1["Open"].fillna(1, inplace=True)
```

In [57]:

```
merged_df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 41088 entries, 0 to 41087
Data columns (total 21 columns):
 #   Column               Non-Null Count  Dtype         
---  ------               --------------  -----         
 0   Store                41088 non-null  int64         
 1   DayOfWeek            41088 non-null  int64         
 2   Date                 41088 non-null  object        
 3   Sales                0 non-null      float64       
 4   Customers            0 non-null      float64       
 5   Open                 41088 non-null  float64       
 6   Promo                41088 non-null  int64         
 7   StateHoliday         41088 non-null  int32         
 8   SchoolHoliday        41088 non-null  int64         
 9   StoreType            41088 non-null  int32         
 10  Assortment           41088 non-null  int32         
 11  CompetitionDistance  41088 non-null  float64       
 12  CompetitionOpenDate  41088 non-null  datetime64[ns]
 13  hasCompetition       41088 non-null  int64         
 14  StoreType_a          41088 non-null  uint8         
 15  StoreType_b          41088 non-null  uint8         
 16  StoreType_c          41088 non-null  uint8         
 17  StoreType_d          41088 non-null  uint8         
 18  Assortment_a         41088 non-null  uint8         
 19  Assortment_b         41088 non-null  uint8         
 20  Assortment_c         41088 non-null  uint8         
dtypes: datetime64[ns](1), float64(4), int32(3), int64(5), object(1), uint8(7)
memory usage: 4.5+ MB
```

Predict

In [58]:

```
# Define the features (X) and target variables (y1 for Customers, y2 for Sales)
X_train = merged_df[['hasCompetition', 'StoreType_a', 'StoreType_b', 'StoreType_c', 'StoreType_d',
               'Assortment_a', 'Assortment_b', 'Assortment_c', 'Store', 'DayOfWeek',
               'Open', 'Promo', 'StateHoliday', 'SchoolHoliday', "CompetitionDistance"]]
y1_train = merged_df['Customers']
y2_train = merged_df['Sales']
```

```
X_test = merged_df1[['hasCompetition', 'StoreType_a', 'StoreType_b', 'StoreType_c', 'StoreType_d',
                'Assortment_a', 'Assortment_b', 'Assortment_c', 'Store', 'DayOfWeek',
                'Open', 'Promo', 'StateHoliday', 'SchoolHoliday', "CompetitionDistance"]]
```

In [59]:
```
# RandomForestRegressor
import numpy as np
from sklearn.ensemble import RandomForestRegressor

# Create two separate Random Forest models
model_customers = RandomForestRegressor(random_state=42)
model_sales = RandomForestRegressor(random_state=42)

# Train the models on the training data
model_customers.fit(X_train, y1_train)
model_sales.fit(X_train, y2_train)

# Make predictions on the test data
tests['Customers'] = np.round(model_customers.predict(X_test))
tests['Sales']     = np.round(model_sales.predict(X_test))
```

In [60]:
```
## If stores close, then Sales and Customers are 0

# Check if the "Open" column in the 'tests' DataFrame has a value of 0
closed_rows = tests[tests["Open"] == 0]

# For the rows where "Open" is 0, set the corresponding values in the "Sales" and "Customers" columns to 0
tests.loc[closed_rows.index, ["Sales", "Customers"]] = 0

tests[tests["Open"] == 0]
```

Out[60]:

	Store	DayOfWeek	Date	Sales	Customers	Open	Promo	StateHoliday	SchoolHoliday
543	703	4	17/09/2015	0.0	0.0	0.0	1	0	0
676	879	4	17/09/2015	0.0	0.0	0.0	1	0	0
840	1097	4	17/09/2015	0.0	0.0	0.0	1	0	0
1399	703	3	16/09/2015	0.0	0.0	0.0	1	0	0
1532	879	3	16/09/2015	0.0	0.0	0.0	1	0	0
...	...	...	...	...	...	...	...	...	...
40227	1111	7	02/08/2015	0.0	0.0	0.0	0	0	0
40228	1112	7	02/08/2015	0.0	0.0	0.0	0	0	0
40229	1113	7	02/08/2015	0.0	0.0	0.0	0	0	0
40230	1114	7	02/08/2015	0.0	0.0	0.0	0	0	0
40231	1115	7	02/08/2015	0.0	0.0	0.0	0	0	1

5984 rows × 9 columns

In [61]:
```
tests.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 41088 entries, 0 to 41087
Data columns (total 9 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Store          41088 non-null  int64  
 1   DayOfWeek      41088 non-null  int64  
 2   Date           41088 non-null  object 
 3   Sales          41088 non-null  float64
 4   Customers      41088 non-null  float64
 5   Open           41077 non-null  float64
 6   Promo          41088 non-null  int64  
 7   StateHoliday   41088 non-null  object 
 8   SchoolHoliday  41088 non-null  int64  
dtypes: float64(3), int64(4), object(2)
memory usage: 2.8+ MB
```

In [62]:
```
tests
```

Out[62]:

	Store	DayOfWeek	Date	Sales	Customers	Open	Promo	StateHoliday	SchoolHoliday
0	1	4	17/09/2015	4916.0	554.0	1.0	1	0	0
1	3	4	17/09/2015	8641.0	857.0	1.0	1	0	0
2	7	4	17/09/2015	9480.0	1035.0	1.0	1	0	0
3	8	4	17/09/2015	6471.0	804.0	1.0	1	0	0
4	9	4	17/09/2015	6108.0	550.0	1.0	1	0	0
...	...	...	...	...	...	...	...	...	...
41083	1111	6	01/08/2015	2772.0	237.0	1.0	0	0	0
41084	1112	6	01/08/2015	8514.0	725.0	1.0	0	0	0
41085	1113	6	01/08/2015	5616.0	584.0	1.0	0	0	0
41086	1114	6	01/08/2015	21346.0	3677.0	1.0	0	0	0
41087	1115	6	01/08/2015	6263.0	445.0	1.0	0	0	1

41088 rows × 9 columns

In [63]:
```
tests.to_csv('test.csv')
```

In [ ]:
```
```