Mohamed Kharma

CSC 447

Final Project Report

Sales Forecasting

Introduction:

Sales forecasting is the process of predicting the future sales of a product or service. This process holds significance for businesses of all sizes and industries, as it aids in effective resource planning and allocation. With accurate sales forecasting, companies can make well-informed decisions regarding production, inventory management, marketing, and staffing. For instance, if a company anticipates a surge in sales for the upcoming quarter, it can proactively increase production and hire additional staff to meet the heightened demand. On the other hand, if sales are expected to slow down, the company can adapt its operations to minimize costs and reduce waste. Also, sales forecasting enables businesses to identify market trends and seize opportunities. Through careful analysis of past sales data and external factors such as the economy and competitors, companies can make informed choices about which products or services to prioritize in the future. Overall, sales forecasting is crucial for businesses because it allows them to plan and allocate resources effectively, make informed decisions, and identify trends and opportunities in the market.

Dataset:

The goal of this project is to forecast and predict the sales accurately for Walmart using various machine learning models such as Linear Regression Model, Random Forest Model and K-Neighbors Model.

The dataset used for this project is the **Walmart Store Sales**, which is a public dataset available on Kaggle. Walmart is one of the largest retailers globally, operating in various countries. The dataset is a historical data that comprises weekly sales data for 45 Walmart stores and 99 departments over a three-year period from 2010-02-05 to 2012-11-01. The dataset consists of three major files, which are stores data, the train data and the features data.

stores.csv - This file contains anonymized information about 45 stores, indicating the type and size of store. The stores data file contains store number, store size and store type as there are three types of stores A, B and C. The file has 45 Data rows in total.

```
# Importing stores data
stores_df=pd.read_csv('Data/stores.csv')
stores_df.head()
```

]:

	Store	Type	Size
0	1	A	151315
1	2	A	202307
2	3	B	37392
3	4	A	205863
4	5	B	34875

train.csv - The train data file contains store number, department number, date, walmart weekly_sales and whether or not that day is a holiday. The file has 115064 Data rows in total.

```
# Importing train data
train_df=pd.read_csv('Data/train.csv',parse_dates=['Date'])
train_df.head()
```

	Store	Dept	Date	Weekly_Sales	IsHoliday
0	1	1	2010-02-05	24924.50	False
1	1	1	2010-02-12	46039.49	True
2	1	1	2010-02-19	41595.55	False
3	1	1	2010-02-26	19403.54	False
4	1	1	2010-03-05	21827.90	False

features.csv - The features data file contains additional data related to the store, and regional activities for the given dates. It contains the attributes like the store number, date, temperature, fuel price, promotional markdowns 1-5, inflation(CPI), unemployment rate and whether or not that day is a holiday. The file has 8190 Data rows in total.

```
# Importing features data
features_df=pd.read_csv('Data/features.csv',parse_dates=['Date'])
features_df.head()
```

]:

	Store	Date	Temperature	Fuel_Price	MarkDown1	MarkDown2	MarkDown3	MarkDown4	MarkDown5	CPI	Unemployment	IsHoliday
0	1	2010-02-05	42.31	2.572	NaN	NaN	NaN	NaN	NaN	211.096358	8.106	False
1	1	2010-02-12	38.51	2.548	NaN	NaN	NaN	NaN	NaN	211.242170	8.106	True
2	1	2010-02-19	39.93	2.514	NaN	NaN	NaN	NaN	NaN	211.289143	8.106	False
3	1	2010-02-26	46.63	2.561	NaN	NaN	NaN	NaN	NaN	211.319643	8.106	False
4	1	2010-03-05	46.50	2.625	NaN	NaN	NaN	NaN	NaN	211.350143	8.106	False

Handling missing values in features dataset for the promotional markdown 1-5 NaN values

```
# Checking missing values in features data
features_df.isnull().sum()
```

```
Store             0
Date              0
Temperature       0
Fuel_Price        0
MarkDown1      4158
MarkDown2      5269
MarkDown3      4577
MarkDown4      4726
MarkDown5      4140
CPI             585
Unemployment    585
IsHoliday         0
dtype: int64
```

```
# Adding the missing values of 0 when there is not MarkDown
features_df["CPI"].fillna(features_df["CPI"].median(),inplace=True)
features_df["Unemployment"].fillna(features_df["Unemployment"].median(),inplace=True)

for i in range(1,6):
  features_df["MarkDown"+str(i)] = features_df["MarkDown"+str(i)].apply(lambda x: 0 if x < 0 else x)
  features_df["MarkDown"+str(i)].fillna(value=0,inplace=True)
```

```
features_df.head()
```

:

	Store	Date	Temperature	Fuel_Price	MarkDown1	MarkDown2	MarkDown3	MarkDown4	MarkDown5	CPI	Unemployment	IsHoliday
0	1	2010-02-05	42.31	2.572	0.0	0.0	0.0	0.0	0.0	211.096358	8.106	False
1	1	2010-02-12	38.51	2.548	0.0	0.0	0.0	0.0	0.0	211.242170	8.106	True
2	1	2010-02-19	39.93	2.514	0.0	0.0	0.0	0.0	0.0	211.289143	8.106	False
3	1	2010-02-26	46.63	2.561	0.0	0.0	0.0	0.0	0.0	211.319643	8.106	False
4	1	2010-03-05	46.50	2.625	0.0	0.0	0.0	0.0	0.0	211.350143	8.106	False

Merging the three data files (stores, train and features) into one main dataset.

```
# Creating the main dataset that merges three different datasets (train, stores, and features) on common columns.

# merges the train and stores datasets on the 'Store' column using a left join (how='left').
data_df = train_df.merge(stores_df, on='Store', how='left')
# merges the resulting dataset with the features dataset on the 'Store' and 'Date' columns using a left join (how='left').
data_df = data_df.merge(features_df, on=['Store', 'Date'], how='left')

# getting rid of repeated column with different name
data_df.drop(columns='IsHoliday_x',inplace=True)
data_df.rename(columns={"IsHoliday_y" : "IsHoliday"}, inplace=True)

#The resulting master dataset should contain information about the stores, the dates, and the sales data.
data_df.head()
```

3]:

	Store	Dept	Date	Weekly_Sales	Type	Size	Temperature	Fuel_Price	MarkDown1	MarkDown2	MarkDown3	MarkDown4	MarkDown5	CPI	
0	1	1	2010-02-05	24924.50	A	151315	42.31	2.572	0.0	0.0	0.0	0.0	0.0	211.096358	
1	1	1	2010-02-12	46039.49	A	151315	38.51	2.548	0.0	0.0	0.0	0.0	0.0	211.242170	
2	1	1	2010-02-19	41595.55	A	151315	39.93	2.514	0.0	0.0	0.0	0.0	0.0	211.289143	
3	1	1	2010-02-26	19403.54	A	151315	46.63	2.561	0.0	0.0	0.0	0.0	0.0	211.319643	
4	1	1	2010-03-05	21827.90	A	151315	46.50	2.625	0.0	0.0	0.0	0.0	0.0	211.350143	

The 'Date' attribute was set as the new index of the main dataset and the dataset was sorted based on date.

```
# Sorting the data based on Date
data_df['Date'] = pd.to_datetime(data_df['Date'])
data_df.sort_values(by=['Date'],inplace=True)
data_df.set_index(data_df.Date, inplace=True)
data_df.head()
```

1]:

	Store	Dept	Date	Weekly_Sales	Type	Size	Temperature	Fuel_Price	MarkDown1	MarkDown2	MarkDown3	MarkDown4	MarkDown5	C
Date														
2010-02-05	1	1	2010-02-05	24924.50	A	151315	42.31	2.572	0.0	0.0	0.0	0.0	0.0	211.0963
2010-02-05	29	5	2010-02-05	15552.08	B	93638	24.36	2.788	0.0	0.0	0.0	0.0	0.0	131.5279
2010-02-05	29	6	2010-02-05	3200.22	B	93638	24.36	2.788	0.0	0.0	0.0	0.0	0.0	131.5279
2010-02-05	29	7	2010-02-05	10820.05	B	93638	24.36	2.788	0.0	0.0	0.0	0.0	0.0	131.5279
2010-02-05	29	8	2010-02-05	20055.64	B	93638	24.36	2.788	0.0	0.0	0.0	0.0	0.0	131.5279

The 'Date' column was split into 'Year', 'Month', and 'Week' to add these columns to the main dataset and use them for data visualization later on.

```python
# Adding year, month and week columns by spliting the Date column
data_df['Year'] = data_df['Date'].dt.year
data_df['Month'] = data_df['Date'].dt.month
data_df['Week'] = data_df['Date'].dt.isocalendar().week
```

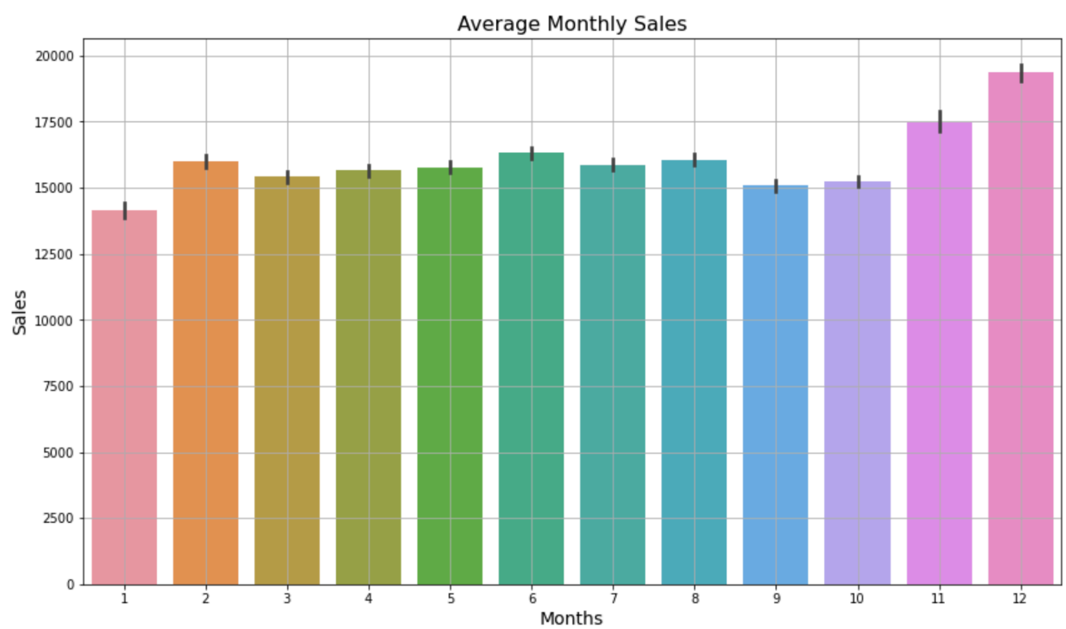
```python
data_df.head()
```

Size	Temperature	Fuel_Price	MarkDown1	MarkDown2	MarkDown3	MarkDown4	MarkDown5	CPI	Unemployment	IsHoliday	Year	Month	Week
151315	42.31	2.572	0.0	0.0	0.0	0.0	0.0	211.096358	8.106	False	2010	2	5
93638	24.36	2.788	0.0	0.0	0.0	0.0	0.0	131.527903	10.064	False	2010	2	5
93638	24.36	2.788	0.0	0.0	0.0	0.0	0.0	131.527903	10.064	False	2010	2	5
93638	24.36	2.788	0.0	0.0	0.0	0.0	0.0	131.527903	10.064	False	2010	2	5
93638	24.36	2.788	0.0	0.0	0.0	0.0	0.0	131.527903	10.064	False	2010	2	5

Exploratory Data Analysis (EDA)

Let's start by viewing how sales changes starting from January all the way to December

```python
# Average Monthly Sales
plt.figure(figsize=(14,8))
sns.barplot(x='Month',y='Weekly_Sales',data=data_df)
plt.ylabel('Sales',fontsize=14)
plt.xlabel('Months',fontsize=14)
plt.title('Average Monthly Sales',fontsize=16)
plt.savefig('plots/avg_monthly_sales.png') # save copy of image
plt.grid()
```



Based on the provided data, Walmart store sales tend to spike the most in November and

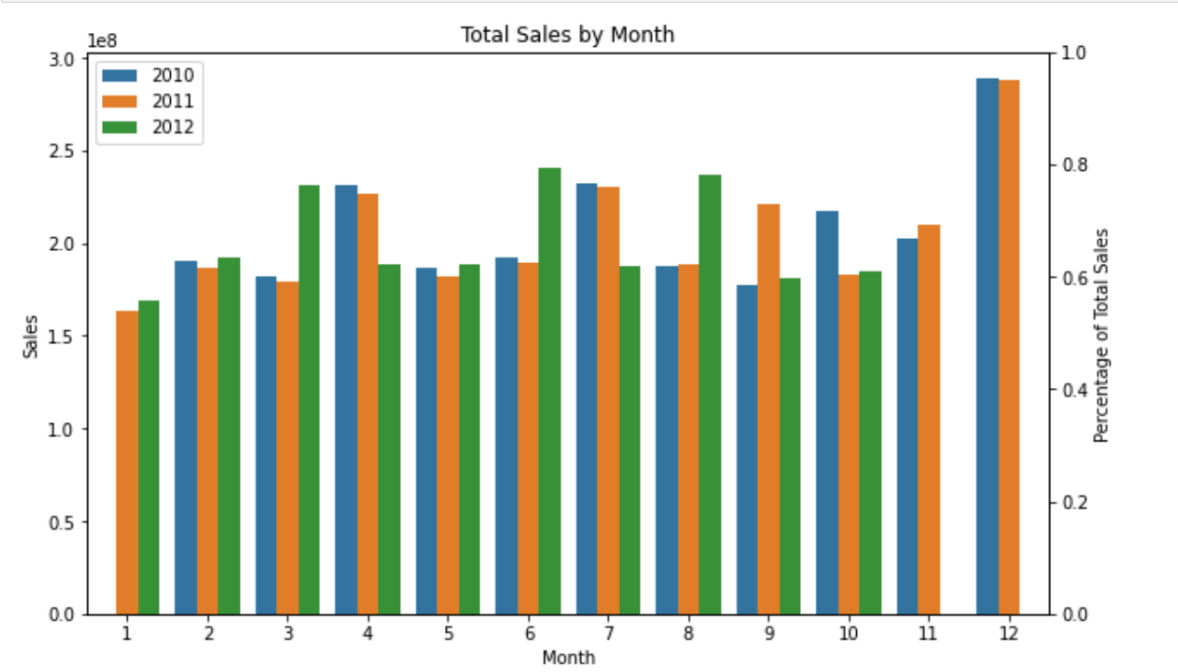
December, which is expected considering it is the end of the year and there are multiple holidays during that time of the year.

Getting the monthly sales for each month of the 3 years, we notice that the data for January of 2010 and November, December of 2012 are missing from the dataset as it was not provided.

```
# getting the monthly sales for each month of the years
monthly_sales = pd.crosstab(data_df["Year"], data_df["Month"], values=data_df["Weekly_Sales"],aggfunc='sum')
monthly_sales
```

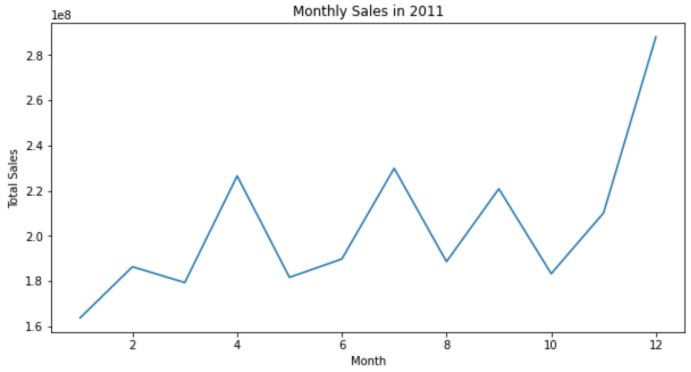
Month	1	2	3	4	5	6	7	8	9	10
Year										
2010	NaN	1.903330e+08	1.819198e+08	2.314124e+08	1.867109e+08	1.922462e+08	2.325801e+08	1.876401e+08	1.772679e+08	2.171618e+08	2.0
2011	1.637040e+08	1.863313e+08	1.793564e+08	2.265265e+08	1.816482e+08	1.897734e+08	2.299114e+08	1.885993e+08	2.208477e+08	1.832613e+08	2.1
2012	1.688945e+08	1.920636e+08	2.315097e+08	1.889209e+08	1.887665e+08	2.406103e+08	1.875095e+08	2.368508e+08	1.806455e+08	1.843617e+08	

The following graph shows the sales increase/decrease over the years for the same month, where the blue, orange and green bars represent the years 2010, 2011 and 2012 respectively.



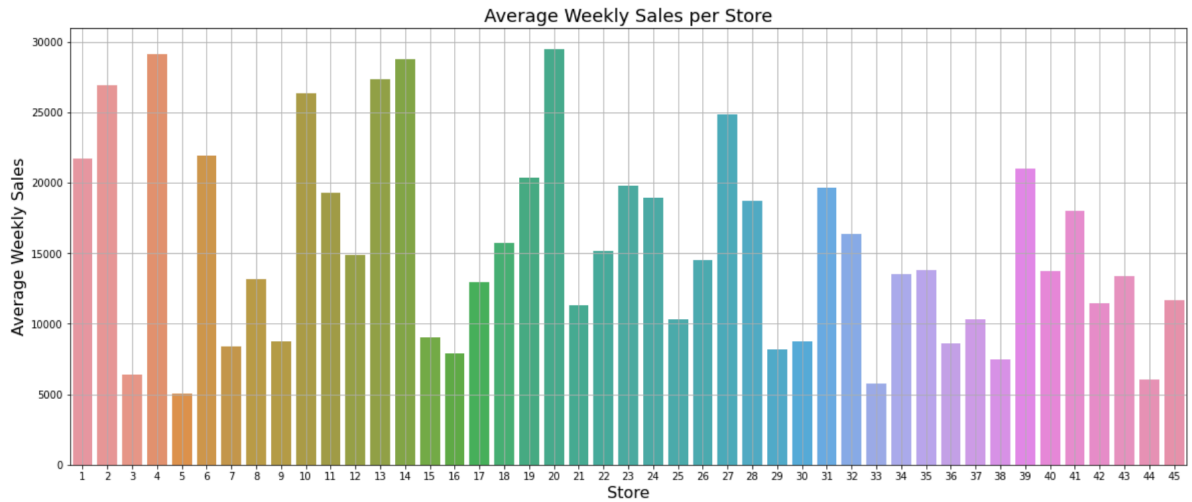
Since the year 2011 is the only year that we were provided with the sales for all 12 months, we can use it to see how sales spikes the most in December compared to other months.

```
# filter the data for the year 2011
df_2011 = data_df[data_df['Year'] == 2011]

# group the data by month and calculate the total sales for each month
m_sales = df_2011.groupby('Month')['Weekly_Sales'].sum()

# plot the bar graph for the monthly sales
plt.figure(figsize=(10,5))
plt.plot(m_sales.index, m_sales.values)
plt.xlabel('Month')
plt.ylabel('Total Sales')
plt.title('Monthly Sales in 2011')
plt.show()
```



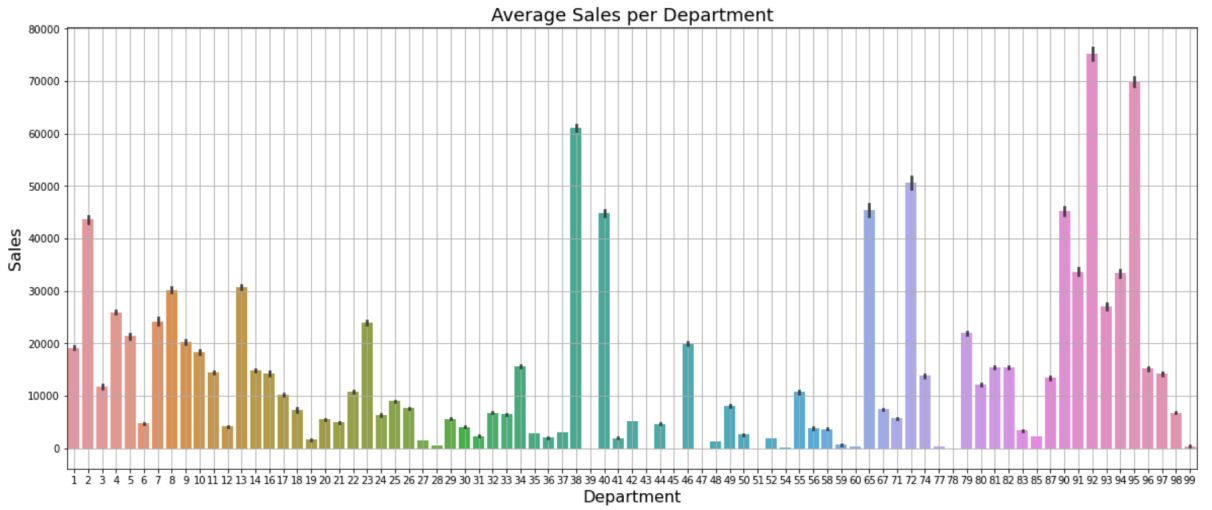
Using the provided data, we are also able to see which stores out of the 45 had the most sales on average each week

```
# Calculate average weekly sales for each store
w_avg_sales = data_df.groupby(['Store'])['Weekly_Sales'].mean().reset_index()

# Plot the average sales per store
plt.figure(figsize=(20,8))
sns.barplot(x='Store', y='Weekly_Sales', data=w_avg_sales)
plt.grid()
plt.title('Average Weekly Sales per Store', fontsize=18)
plt.ylabel('Average Weekly Sales', fontsize=16)
plt.xlabel('Store', fontsize=16)
plt.savefig('plots/avg_weekly_sales_store.png')
plt.show()
```



Additionally, using the dataset, we are able to see which department out of the 99 departments had the most sales on average each week

```
# Plot the average sales per department
plt.figure(figsize=(20,8))
sns.barplot(x='Dept',y='Weekly_Sales',data=data_df)
plt.grid()
plt.title('Average Sales per Department', fontsize=18)
plt.ylabel('Sales', fontsize=16)
plt.xlabel('Department', fontsize=16)
plt.savefig('plots/avg_sales_dept.png')
plt.show()
```



Based on the plot results, we notice that departments 92,95, and 38 have the most weekly sales on average.

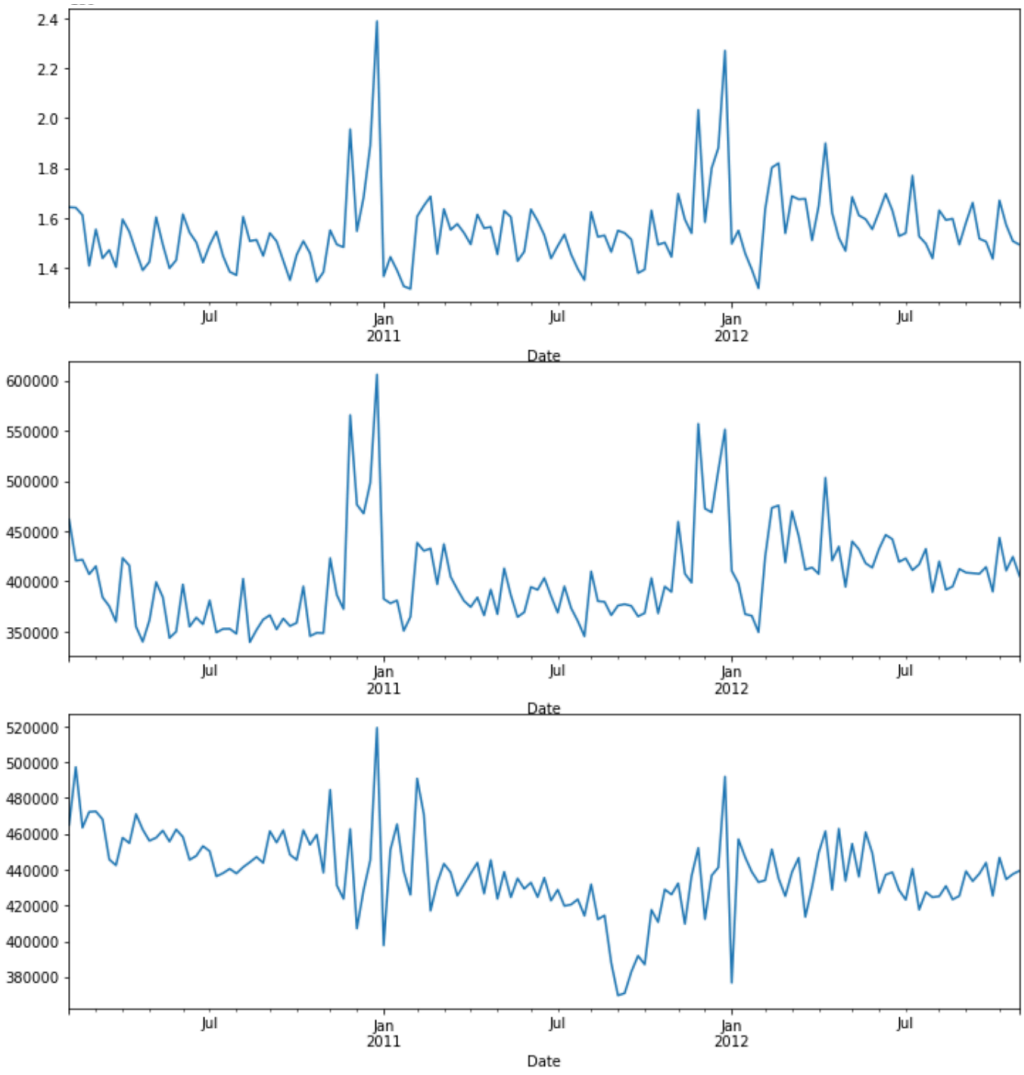
Time Series Analysis:

Time series analysis is a statistical method used to analyze and interpret data collected over a period of time. It focuses on understanding the patterns, trends, and underlying factors that influence the data's behavior. In the context of this report, time series analysis was conducted on the weekly sales data of representative stores from different store types (a, b, and c). The goal was to gain insights into the sales trends exhibited by these stores over time.

By converting the date column to a datetime type and preparing the data by ensuring the appropriate data types, the dataset was prepared for analysis. One store was selected from each store type to represent their respective groups, and their weekly sales data was extracted. To

visualize the trends in the weekly sales, the data was downsampled from a daily frequency to a weekly frequency using the resample method. This allowed for a clearer understanding of the present trends and patterns. Line plots were then created to represent the weekly sales over time for each store.

```
# Convert the date column to a datetime type
train_df['Date'] = pd.to_datetime(train_df['Date'])

# Data Preparation: input should be float type
train_df['Weekly_Sales'] = train_df['Weekly_Sales'] * 1.0

# Assigning one store from each category
sales_a = train_df[train_df.Store == 1].set_index('Date')['Weekly_Sales']
sales_b = train_df[train_df.Store == 3].set_index('Date')['Weekly_Sales']
sales_c = train_df[train_df.Store == 30].set_index('Date')['Weekly_Sales']

f, (ax1, ax2, ax3) = plt.subplots(3, figsize=(12, 13))

# Trend
sales_a.resample('W').sum().plot(ax=ax1)
sales_b.resample('W').sum().plot(ax=ax2)
sales_c.resample('W').sum().plot(ax=ax3)
```



Each of these graphs represent a storetype and the plots shown above show that sales for StoreType A, B and C typically peak during the holiday season (Christmas) and then start to drop by the start of the new year.

Data normalization:

Also known as feature scaling, is a preprocessing technique used to rescale numeric features in a dataset to a specific range. The purpose of normalization is to bring all the features to a similar scale, preventing certain features from dominating the others due to differences in their magnitudes.

```
# chenge the type of holiday to be integer or 1 or 0
data_df['IsHoliday'] = data_df['IsHoliday'].astype('int')

num_col = ['Weekly_Sales','Size','Temperature','Fuel_Price','CPI','Unemployment','MarkDown1', 'MarkDown2', 'MarkDown3', 'Mark

# nomralizating function for the the dataset
minmax_scale = MinMaxScaler(feature_range=(0, 1))
def normalization(df,col):
  for i in col:
    arr = df[i]
    arr = np.array(arr)
    df[i] = minmax_scale.fit_transform(arr.reshape(len(arr),1))
  return df

# Before
data_df.head()
```

]4]:

Date	Store	Dept	Date	Weekly_Sales	Type	Size	Temperature	Fuel_Price	MarkDown1	MarkDown2	...	Unemployment	IsHoliday	Year	Month	We
2010-02-05	1	1	2010-02-05	24924.50	A	151315	42.31	2.572	0.0	0.0	...	8.106	0	2010	2	
2010-02-05	9	97	2010-02-05	668.48	B	125833	38.01	2.572	0.0	0.0	...	6.415	0	2010	2	
2010-02-05	9	85	2010-02-05	693.87	B	125833	38.01	2.572	0.0	0.0	...	6.415	0	2010	2	
2010-02-05	8	80	2010-02-05	8654.60	A	155078	34.14	2.572	0.0	0.0	...	6.299	0	2010	2	
2010-02-05	9	55	2010-02-05	11123.56	B	125833	38.01	2.572	0.0	0.0	...	6.415	0	2010	2	

5 rows × 24 columns

```
# Normalize the dataset and store it in nordata_df
nordata_df = normalization(data_df.copy(),num_col)
# After
nordata_df.head()
```

']:

Date	Store	Dept	Date	Weekly_Sales	Type	Size	Temperature	Fuel_Price	MarkDown1	MarkDown2	...	Unemployment	IsHoliday	Year	Month	W
2010-02-05	1	1	2010-02-05	0.042851	A	0.630267	0.434149	0.0501	0.0	0.0	...	0.405118	0	2010	2	
2010-02-05	9	97	2010-02-05	0.008104	B	0.492338	0.392074	0.0501	0.0	0.0	...	0.243052	0	2010	2	
2010-02-05	9	85	2010-02-05	0.008141	B	0.492338	0.392074	0.0501	0.0	0.0	...	0.243052	0	2010	2	
2010-02-05	8	80	2010-02-05	0.019544	A	0.650636	0.354207	0.0501	0.0	0.0	...	0.231934	0	2010	2	
2010-02-05	9	55	2010-02-05	0.023081	B	0.492338	0.392074	0.0501	0.0	0.0	...	0.243052	0	2010	2	

5 rows × 24 columns

Data Preparing:

In the following provided code snippet, the normalized dataset is used to create the variables X and Y that represent the independent variables (features) and the dependent variable (target) respectively for the model.

```
# Create X and Y variables for the model
X = nordata_df.drop(['Weekly_Sales'],axis=1)
Y = nordata_df.Weekly_Sales

# split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.3, random_state=42)
```

The test_size parameter is set to 0.3, which means that 30% of the data will be used for testing, while the remaining 70% will be used for training the model. The random_state parameter is set to 42 to ensure reproducibility, meaning that the same random split will be obtained each time the code is executed.

Machine Learning Models:

Machine learning models such as Linear Regression, Random Forest Regression and K Neighbors model were used to predict the weekly sales of Walmart, then, the performance of these models were compared using evaluation metrics, which helped determine which model performs best for predicting the weekly sales of Walmart.

Linear Regression Model:

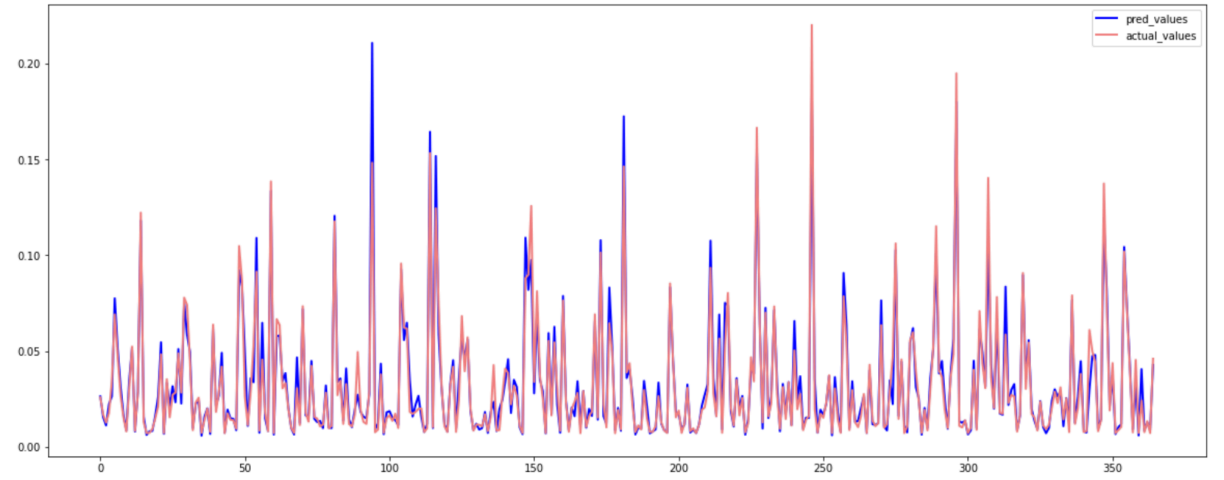
Linear regression is a simple and commonly used model that assumes a linear relationship between the independent variables and the target variable. It works well when there is a linear relationship between the features and the target variable.

```
# Create a linear regression object and fit the model to the training data
lr = LinearRegression(normalize=False)
lr.fit(X_train, y_train)

lr_acc = lr.score(X_test,y_test)*100
print("Linear Regression Accuracy - ",lr_acc)

y_pred = lr.predict(X_test)
print("Mean Absolute Error: " , metrics.mean_absolute_error(y_test, y_pred))
print("Root Mean Square Error: " , np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
print("R-squared (R2): " , metrics.explained_variance_score(y_test, y_pred))
```

```
Linear Regression Accuracy -  90.91817190867872
Mean Absolute Error:  0.003977225692650983
Root Mean Square Error:  0.009808431241745066
R-squared (R2):  0.909181836452132
```

The model achieves an accuracy of 90.92% (R-squared score) on the test data. The mean absolute error is approximately 0.00398, and the root mean square error is approximately 0.00981. These metrics provide insights into the accuracy and performance of the linear regression model in predicting the target variable (Weekly_Sales).

```
#Graph
plt.figure(figsize=(20,8))
plt.plot(lr.predict(X_test[:365]), label="pred_values", linewidth=2.0,color='blue')
plt.plot(y_test[:365].values, label="actual_values", linewidth=2.0,color='lightcoral')
plt.legend(loc="best")
plt.show()
```



The graph above shows the predicted sales (blue line) and the actual sales (red line) over the span of 365 days for Linear regression.

Random Forest Model:

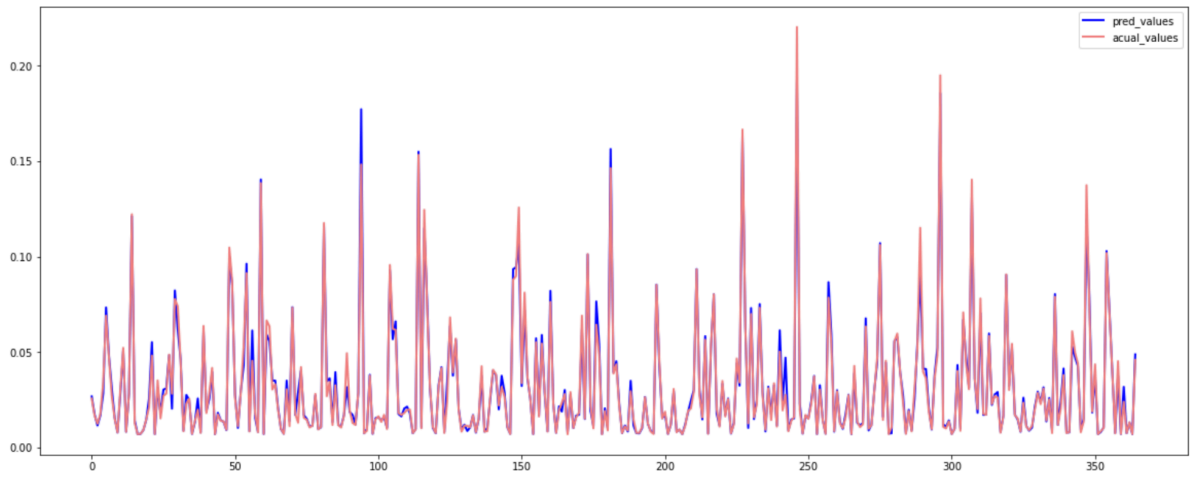
Random forest regression is an ensemble model that combines multiple decision trees to make predictions. It is effective for capturing non-linear relationships and handling complex datasets.

```
# Create a Random forest regression object and fit the model to the training data
rf = RandomForestRegressor()
rf.fit(X_train, y_train)

rf_acc = rf.score(X_test,y_test)*100
print("Random Forest Accuracy - ",rf_acc)

y_pred = rf.predict(X_test)
print("Mean Absolute Error: " , metrics.mean_absolute_error(y_test, y_pred))
print("Root Mean Square Error: " , np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
print("R-squared (R2): " , metrics.explained_variance_score(y_test, y_pred))
```

```
Random Forest Accuracy -  96.85627417477062
Mean Absolute Error:  0.0020908007714652873
Root Mean Square Error:  0.005770789658927128
R-squared (R2):  0.9685638642905959
```

The Random Forest model achieved a high accuracy of 96.85% (R-squared score) on the test data, indicating that it can effectively predict the weekly sales of Walmart. The mean absolute error and root mean squared error values are relatively low, indicating that the model's predictions are close to the actual values.

```
# Graph
plt.figure(figsize=(20,8))
plt.plot(rf.predict(X_test[:365]), label="pred_values", linewidth=2.0,color='blue')
plt.plot(y_test[:365].values, label="acual_values", linewidth=2.0,color='lightcoral')
plt.legend(loc="best")
plt.show()
```



The graph above shows the predicted sales (blue line) and the actual sales (red line) over the span of 365 days for Random Forest Regression.

K Neighbors Model:

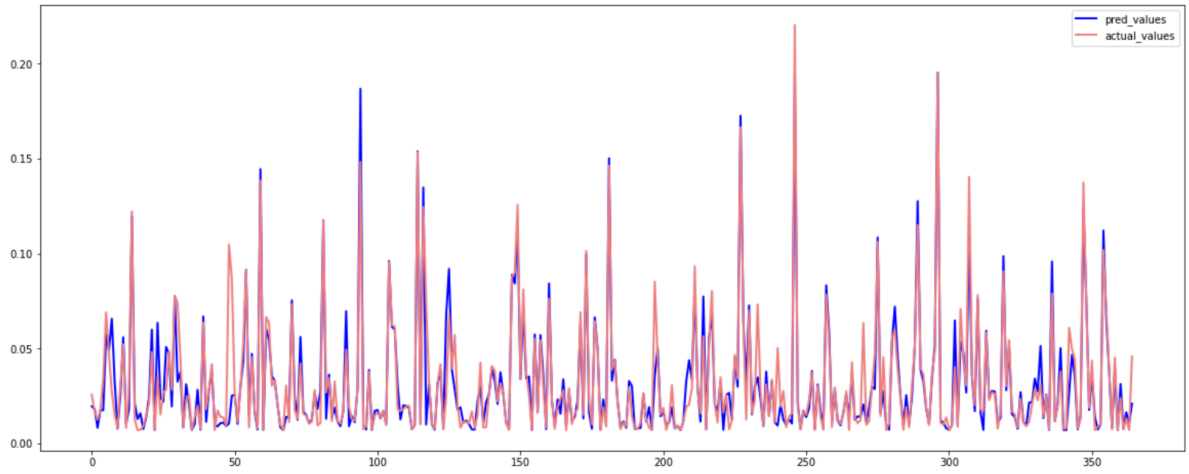
K Neighbors, or K-Nearest Neighbors (KNN), is a simple yet effective supervised learning algorithm that can be used for both classification and regression tasks. In the case of regression, it predicts the value of a target variable by finding the average or weighted average of the K nearest neighbors in the feature space.

```
# Create a KNeighborsRegressor object and fit the model to the training data
knn = KNeighborsRegressor(n_neighbors = 1,weights = 'uniform')
knn.fit(X_train,y_train)

knn_acc = knn.score(X_test, y_test)*100
print("KNeighbhbors Accuracy - ",knn_acc)

y_pred = knn.predict(X_test)
print("Mean Absolute Error: " , metrics.mean_absolute_error(y_test, y_pred))
print("Root Mean Square Error: " , np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
print("R-squared (R2): " , metrics.explained_variance_score(y_test, y_pred))
```

```
Root Mean Square Error:  0.011938365120982283
R-squared (R2):  0.8654641620100811
```

The K Neighbors model achieved an accuracy of 86.55% (R-squared score) on the test data, indicating that it can reasonably predict the weekly sales of Walmart. The mean absolute error and root mean square error values are relatively low, indicating that the model's predictions are close to the actual values. The R-squared score of 0.86546416 indicates that approximately 87% of the variance in the target variable can be explained by the model.

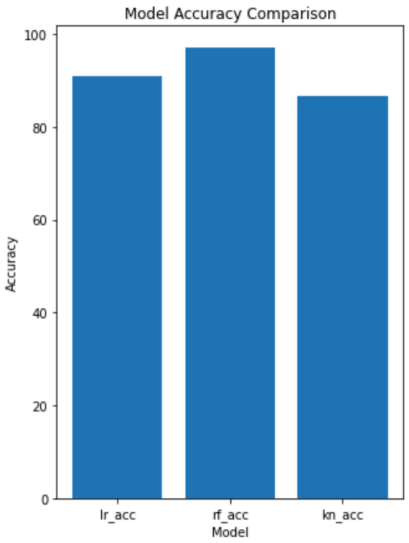
```
plt.figure(figsize=(20,8))
plt.plot(knn.predict(X_test[:365]), label="pred_values", linewidth=2.0,color='blue')
plt.plot(y_test[:365].values, label="actual_values", linewidth=2.0,color='lightcoral')
plt.legend(loc="best")
plt.show()
```



Comparing the models:

While the Linear Regression and K Neighbors model performed well, they had a lower accuracy compared to the Random Forest model. However, they still provide reasonable predictions and can be considered as an alternative model for predicting the weekly sales of Walmart.

```
# create bar plot
plt.figure(figsize=(5,7))
plt.bar(acc_df['model'], acc_df['accuracy'])
plt.xlabel('Model')
plt.ylabel('Accuracy')
plt.title('Model Accuracy Comparison')
plt.show()
```



It's important to note that the choice of parameters, such as the number of neighbors (K) and the weighting scheme, can affect the performance of the K Neighbors model. Experimenting with different parameter values may lead to improved accuracy and better predictions. Overall, the Random Forest Regression model demonstrates strong performance in predicting the weekly sales of Walmart, providing accurate and reliable predictions.

Conclusion:

In this project, we analyzed the Walmart Store Sales dataset to forecast and predict the weekly sales accurately. After conducting exploratory data analysis, we observed sales patterns and identified top-performing stores and departments. Time series analysis revealed the seasonal

nature of sales, with peaks during the holiday season. We then trained and evaluated three machine learning models: Linear Regression, Random Forest Regression, and K Neighbors Regression. The Random Forest model demonstrated the highest accuracy of 96.85%, outperforming the other models. However, the Linear Regression and K Neighbors models also provided reasonable predictions. Leveraging these models allows businesses to make informed decisions, optimize resource allocation, and maximize profitability based on accurate sales forecasts.