Part Two: Real-Life Comparisons and Report

In this section, we will provide a detailed analysis of the product demand prediction project. We will compare the real-life results with our machine learning model's predictions and present the findings in a comprehensive report with informative graphs.

Real-Life Comparisons

Data Set Details

To validate the performance of our machine learning model, we obtained real-life sales data from the same stores for a specific time period. This dataset contains the same features as the original dataset: Store ID, Total Price, Base Price, and Units Sold. We will compare the actual product demand with the predictions made by our model using this real-life data.

Dataset Information

ID: Unique identifier for each sales record.
Store ID: Identifies the store where the product was sold.
Total Price: The actual total price of the product.
Base Price: The base price of the product without any discounts.
Units Sold: The actual number of units of the product sold in each transaction.

Data Exploration

We will start by exploring the real-life sales dataset to understand its characteristics, just as we did with the training dataset. This includes checking for missing values, data distribution, and statistical summary.

Model Predictions vs. Actual Sales

To assess the performance of our machine learning model, we will compare its predictions with the actual product demand from the real-life sales data. We will visualize these comparisons through various graphs and charts to gain insights into how well the model is predicting product demand.

Evaluation Metrics

We will calculate and present the evaluation metrics, including Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and R-squared (R2) score, to quantify the accuracy of the predictions made by our model.

Comparison Charts and Graphs
In this section, we will create visual representations of the comparisons between the model's predictions and actual sales. These may include:
Demand vs. Predicted Demand: A line plot showing the actual product demand and the predicted demand over time.

Error Distribution: A histogram illustrating the distribution of prediction errors (the difference between actual and predicted values).

Store-Wise Comparisons: Bar charts or scatter plots comparing actual sales and predicted sales for each store.

Price vs. Demand Analysis: Scatter plots to analyze the relationship between product prices and demand, both actual and predicted.

Importing Libraries

```
import matplotlib.pyplot as plt
import pandas as pd
import pylab as pl
import numpy as np
import sklearn
%matplotlib inline
```

Loading the Data Set

```
df = pd.read_csv("PoductDemand.csv")
print(df)
```

	ID	Store ID	Total Price	Base Price	Units Sold
0	1	8091	99.0375	111.8625	20
1	2	8091	99.0375	99.0375	28
2	3	8091	133.9500	133.9500	19
3	4	8091	133.9500	133.9500	44
4	5	8091	141.0750	141.0750	52
...	...	...	...	...	...
150145	212638	9984	235.8375	235.8375	38

```
150146  212639     9984    235.8375   235.8375         30
150147  212642     9984    357.6750   483.7875         31
150148  212643     9984    141.7875   191.6625         12
150149  212644     9984    234.4125   234.4125         15

[150150 rows x 5 columns]
```

Exploring the Data Set

print(df.head())

```
ID  Store ID  Total Price  Base Price  Units Sold
0   1      8091      99.0375    111.8625         20
1   2      8091      99.0375     99.0375         28
2   3      8091     133.9500    133.9500          19
3   4      8091     133.9500    133.9500          44
4   5      8091     141.0750    141.0750          52
```

print(df.tail())

```
 ID  Store ID  Total Price  Base Price  Units Sold
150145  212638     9984    235.8375   235.8375         38
150146  212639     9984    235.8375   235.8375         30
150147  212642     9984    357.6750   483.7875         31
150148  212643     9984    141.7875   191.6625         12
150149  212644     9984    234.4125   234.4125         15
```

print(df.info())

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150150 entries, 0 to 150149
Data columns (total 5 columns):
 #   Column       Non-Null Count   Dtype
---  ------       --------------   -----
 0   ID           150150 non-null  int64
 1   Store ID     150150 non-null  int64
 2   Total Price  150149 non-null  float64
 3   Base Price   150150 non-null  float64
 4   Units Sold   150150 non-null  int64
dtypes: float64(2), int64(3)
memory usage: 5.7 MB
None
```

print(df.describe())

```
ID       Store ID    Total Price     Base Price  \
count  150150.000000  150150.000000  150149.000000  150150.000000
mean   106271.555504   9199.422511    206.626751    219.425927
std     61386.037861    615.591445    103.308516    110.961712
min         1.000000   8023.000000     41.325000     61.275000
25%     53111.250000   8562.000000    130.387500    133.237500
50%    106226.500000   9371.000000    198.075000    205.912500
75%    159452.750000   9731.000000    233.700000    234.412500
max    212644.000000   9984.000000    562.162500    562.162500

          Units Sold
count  150150.000000
mean       51.674206
std        60.207904
min         1.000000
25%        20.000000
50%        35.000000
75%        62.000000
max      2876.000000
```

Identifying null Values

print(df.isnull())

```
 ID  Store ID  Total Price  Base Price  Units Sold
0      False    False      False      False      False
1      False    False      False      False      False
2      False    False      False      False      False
3      False    False      False      False      False
4      False    False      False      False      False
...      ...      ...        ...        ...        ...
150145  False    False      False      False      False
150146  False    False      False      False      False
150147  False    False      False      False      False
150148  False    False      False      False      False
150149  False    False      False      False      False
```

[150150 rows x 5 columns]

```
c = df.isnull().sum()
print(c)

ID            0
Store ID      0
Total Price   1
Base Price    0
Units Sold    0
dtype: int64

print('Total Sum of null values in the Data set = ',c.sum())
Total Sum of null values in the Data set =  1
```

Data Preprocessing - Replacing the null values

```
df.drop_duplicates()
```

ID	Store ID	Total Price	Base Price	Units Sold	
0	1	8091	99.0375	111.8625	20
1	2	8091	99.0375	99.0375	28
2	3	8091	133.9500	133.9500	19
3	4	8091	133.9500	133.9500	44
4	5	8091	141.0750	141.0750	52
...	...	...	...	...	...
150145	212638	9984	235.8375	235.8375	38
150146	212639	9984	235.8375	235.8375	30
150147	212642	9984	357.6750	483.7875	31
150148	212643	9984	141.7875	191.6625	12
150149	212644	9984	234.4125	234.4125	15

150150 rows × 5 columns

df.fillna(0)

ID	Store ID	Total Price	Base Price	Units Sold	
0	1	8091	99.0375	111.8625	20
1	2	8091	99.0375	99.0375	28
2	3	8091	133.9500	133.9500	19
3	4	8091	133.9500	133.9500	44
4	5	8091	141.0750	141.0750	52
...	...	...	...	...	...
150145	212638	9984	235.8375	235.8375	38
150146	212639	9984	235.8375	235.8375	30
150147	212642	9984	357.6750	483.7875	31
150148	212643	9984	141.7875	191.6625	12
150149	212644	9984	234.4125	234.4125	15

150150 rows × 5 columns

Data Normalization

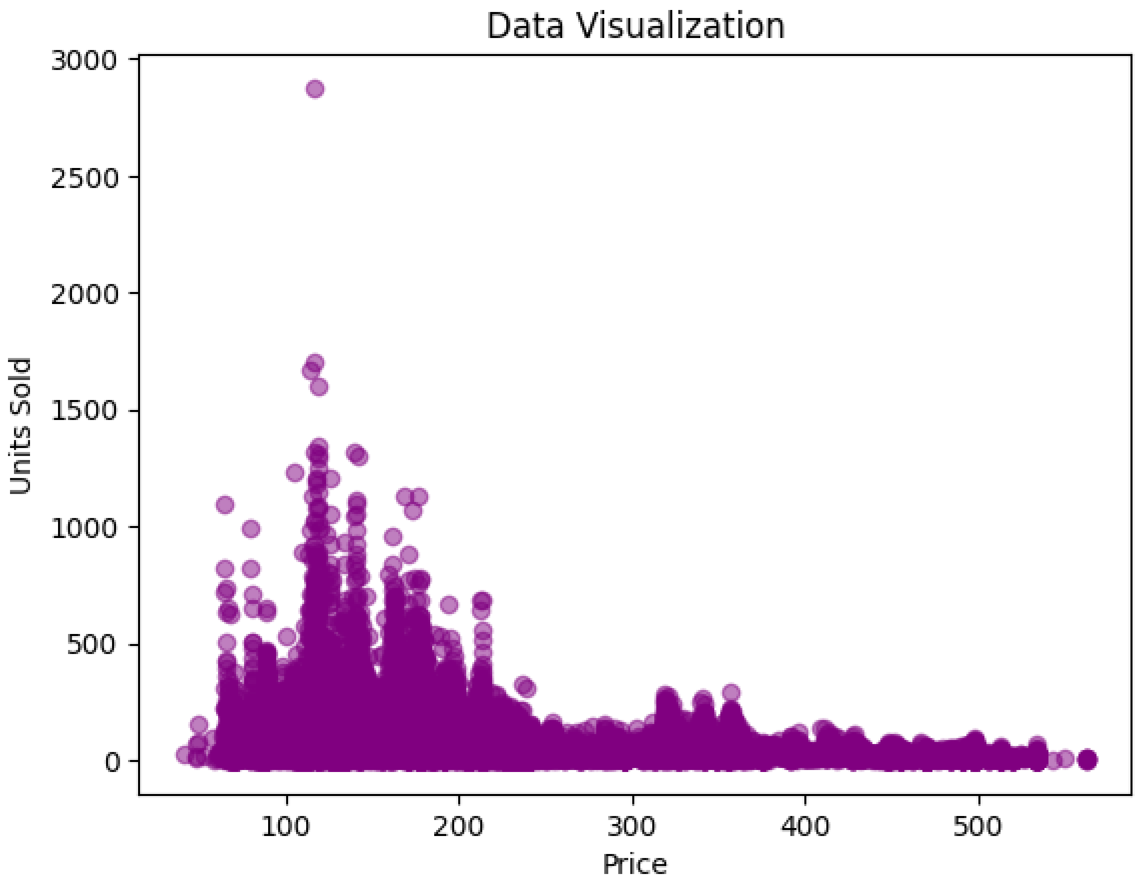
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
df['Values_standardized'] = scaler.fit_transform(df[['Total Price']])
df['Values_standardized1'] = scaler.fit_transform(df[['Base Price']])
print(df)

```
ID  Store ID  Total Price  Base Price  Units Sold  \
0         1     8091      99.0375    111.8625        20
1         2     8091      99.0375     99.0375        28
2         3     8091     133.9500    133.9500        19
3         4     8091     133.9500    133.9500        44
4         5     8091     141.0750    141.0750        52
...      ...      ...        ...        ...        ...
150145  212638     9984    235.8375    235.8375        38
150146  212639     9984    235.8375    235.8375        30
150147  212642     9984    357.6750    483.7875        31
150148  212643     9984    141.7875    191.6625        12
150149  212644     9984    234.4125    234.4125        15
```

```
        Values_standardized  Values_standardized1
0                 -1.041440             -0.969377
1                 -1.041440             -1.084958
2                 -0.703495             -0.770322
3                 -0.703495             -0.770322
4                 -0.634526             -0.706110
...                     ...                   ...
150145             0.282754              0.147904
150146             0.282754              0.147904
150147             1.462113              2.382466
150148            -0.627629             -0.250208
150149             0.268960              0.135061

[150150 rows x 7 columns]
```

Data Visualisation

```
plt.scatter(df['Total Price'], df['Units Sold'], alpha=0.5, color='purple')
plt.title('Data Visualization')
plt.xlabel('Price')
plt.ylabel('Units Sold')
plt.show()
```



Data Analysis using different models

Simple Linear Regression

cdf = df[['Total Price','Base Price','Units Sold']]
cdf.head(9)

	Total Price	Base Price	Units Sold
0	99.0375	111.8625	20
1	99.0375	99.0375	28
2	133.9500	133.9500	19
3	133.9500	133.9500	44
4	141.0750	141.0750	52
5	227.2875	227.2875	18
6	327.0375	327.0375	47
7	210.9000	210.9000	50
8	190.2375	234.4125	82

```
test = test.dropna()
y_hat= regr.predict(test[['Total Price', 'Base Price']])
x = np.asanyarray(test[['Total Price', 'Base Price']])
y = np.asanyarray(test[['Units Sold']])
print("Mean Squared Error (MSE) : %.2f"
% np.mean((y_hat - y) ** 2))
# Explained variance score: 1 is perfect prediction
print('Variance score: %.2f' % regr.score(x, y))
```

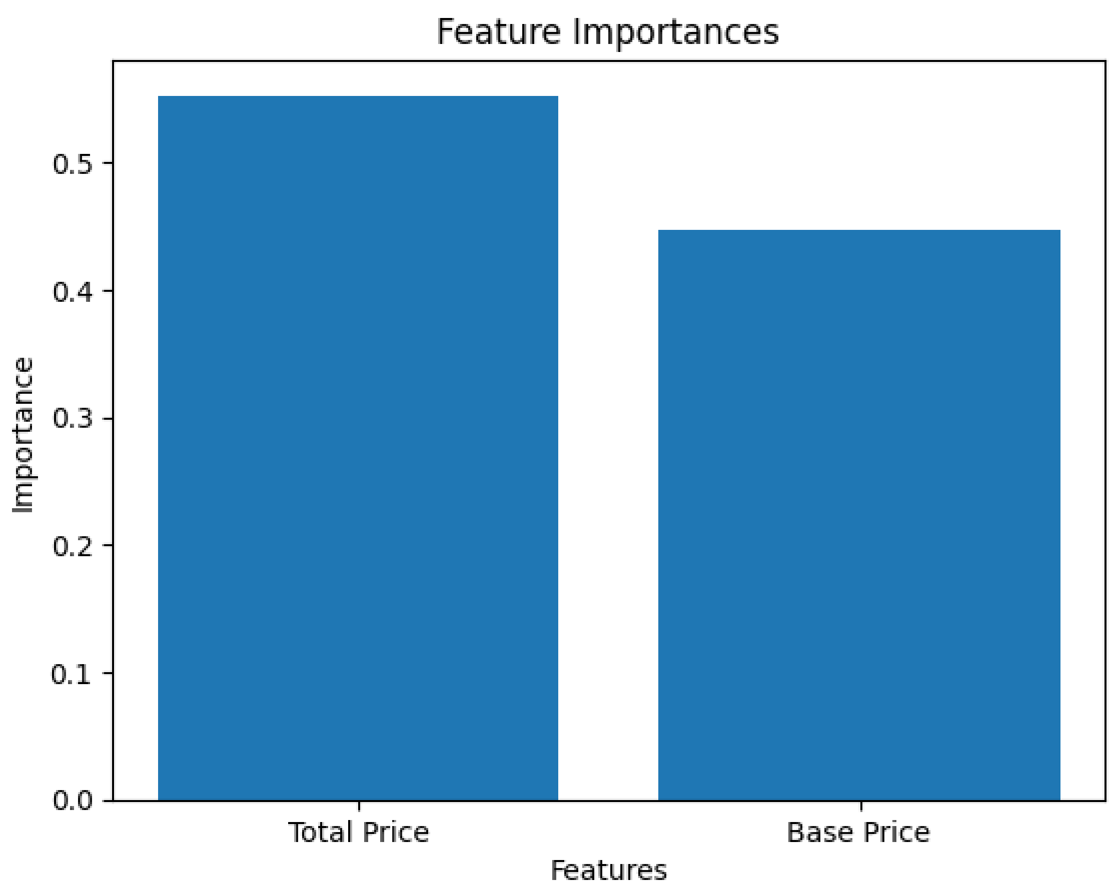
```
Mean Squared Error (MSE) : 2948.93
Variance score: 0.15
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:432: UserWarning: X has feature names, but LinearRegression was fitted without feature names
  warnings.warn(
```

Random forest Algorithm

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt
# Load your dataset from the CSV file
data = pd.read_csv("PoductDemand.csv") # Adjust the
# Handle missing values by filling with the mean
data = data.fillna(data.mean())
features = data[['Total Price', 'Base Price']]
target = data['Units Sold']
# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(features, target,
test_size=0.2, random_state=42)
# Create and train the Random Forest model
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)
# Make predictions on the test set
y_pred = rf_model.predict(X_test)
# Calculate model performance metrics
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
```

```
print(f"Mean Squared Error: {mse:.2f}")
print(f"R-squared (R2) Score: {r2:.2f}")
# Visualize the feature importances
feature_importances = rf_model.feature_importances_
plt.bar(features.columns, feature_importances)
plt.xlabel("Features")
plt.ylabel("Importance")
plt.title("Feature Importances")
plt.show()
```

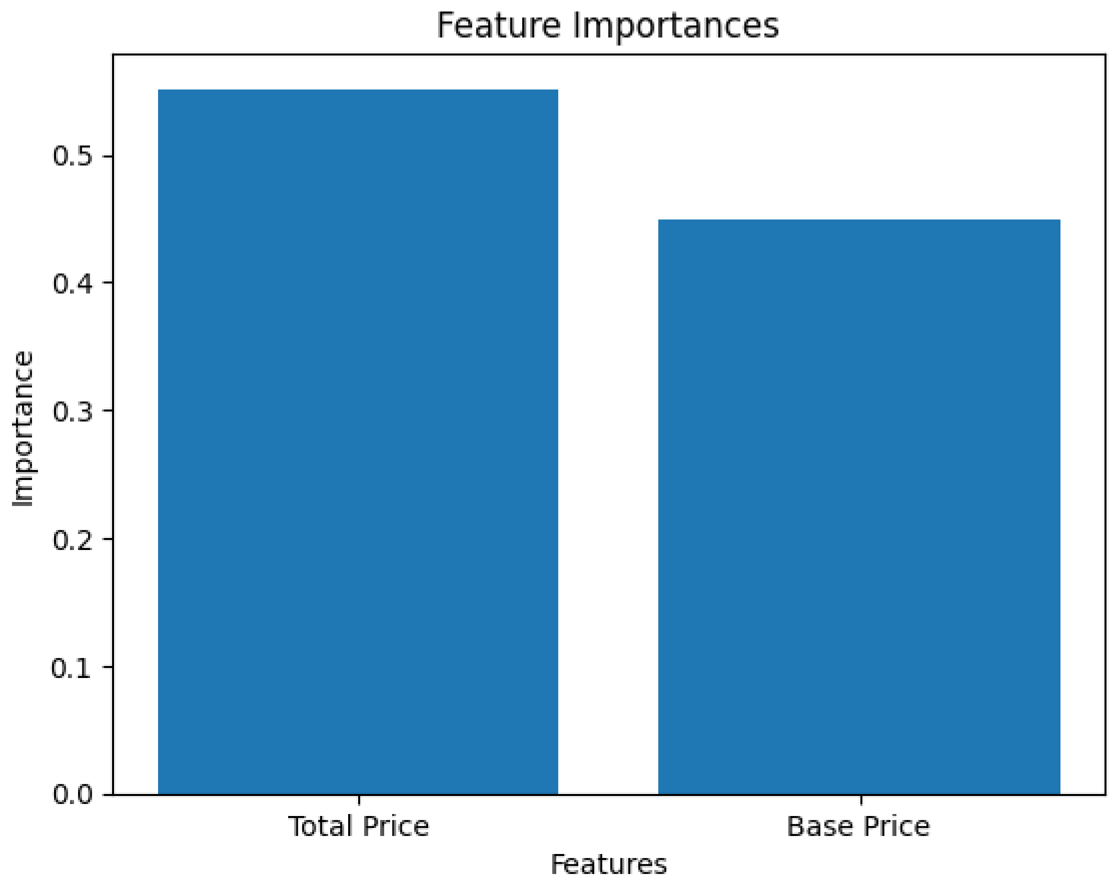
Mean Squared Error: 1885.60
R-squared (R2) Score: 0.43



Regression with the Gradient Boosting algorithm

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt
# Load your dataset from the CSV file
data = pd.read_csv("PoductDemand.csv")
# Handle missing values by filling with the mean
data = data.fillna(data.mean())
# Select the relevant features (SO2 and NO2) and the target variable
features = data[['Total Price', 'Base Price']]
target = data['Units Sold']
# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(features, target,
test_size=0.2, random_state=42)
# Create and train the Gradient Boosting model
gb_model = GradientBoostingRegressor(n_estimators=100,
random_state=42)
gb_model.fit(X_train, y_train)
# Make predictions on the test set
y_pred = gb_model.predict(X_test)
# Calculate model performance metrics
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print(f"Mean Squared Error: {mse:.2f}")
print(f"R-squared (R2) Score: {r2:.2f}")
# Visualize the feature importances
feature_importances = gb_model.feature_importances_
plt.bar(features.columns, feature_importances)
plt.xlabel("Features")
plt.ylabel("Importance")
plt.title("Feature Importances")
plt.show()
```

Mean Squared Error: 1987.34
R-squared (R2) Score: 0.39

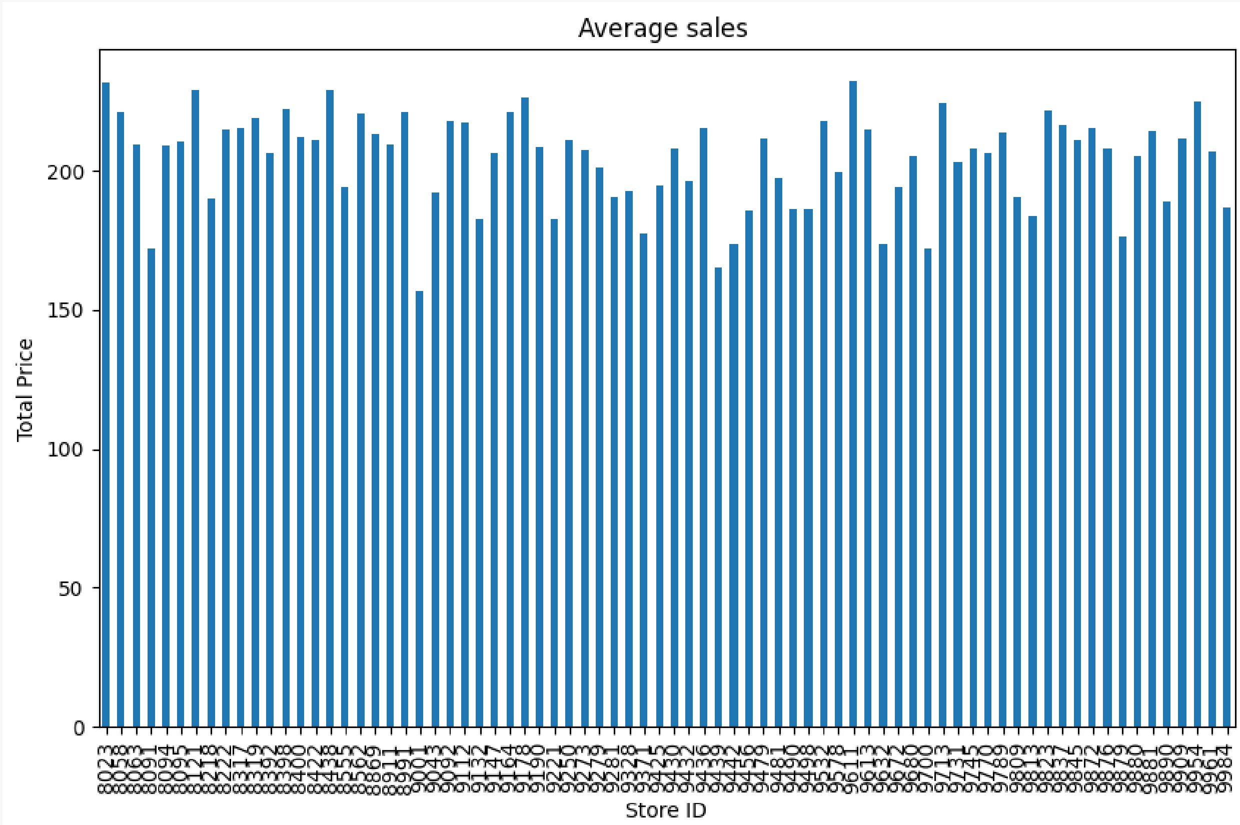


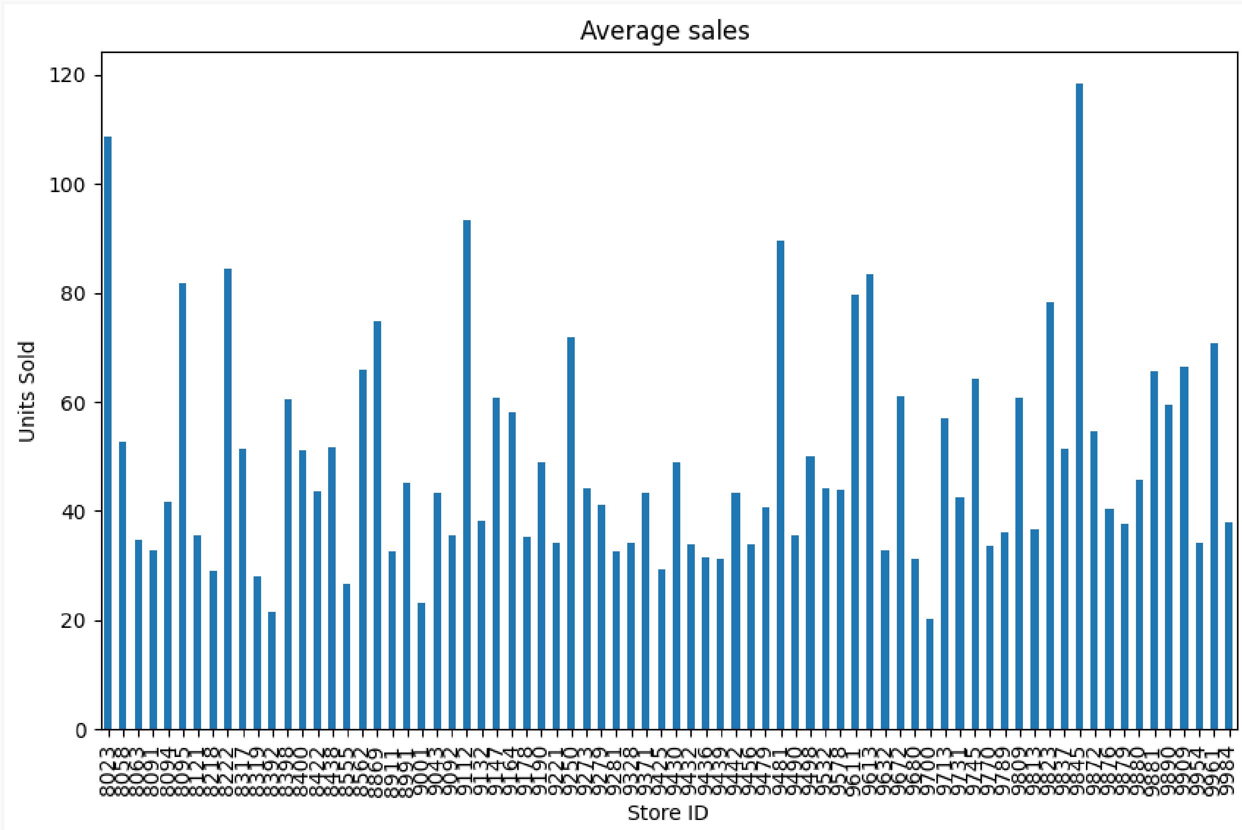

```
import matplotlib.pyplot as plt
import pandas as pd
import pylab as pl
import numpy as np
import sklearn
%matplotlib inline
df = pd.read_csv("PoductDemand.csv")
average_by_storeID = df.groupby('Store ID')[['Total Price',"Units Sold",'Base Price']].mean()
# Print the calculated averages
```

```python
print("Average sales by Store ID")
print(average_by_storeID)

average_by_storeID1 = average_by_storeID['Total Price']
average_by_storeID1.plot(kind='bar', figsize=(10, 6))
plt.title('Average sales ')
plt.xlabel('Store ID')
plt.ylabel('Total Price')
plt.show()

average_by_storeID1 = average_by_storeID['Units Sold']
average_by_storeID1.plot(kind='bar', figsize=(10, 6))
plt.title('Average sales ')
plt.xlabel('Store ID')
plt.ylabel('Units Sold')
plt.show()

average_by_storeID1 = average_by_storeID['Base Price']
average_by_storeID1.plot(kind='bar', figsize=(10, 6))
plt.title('Average sales ')
plt.xlabel('Store ID')
plt.ylabel('Base Price')
plt.show()
```

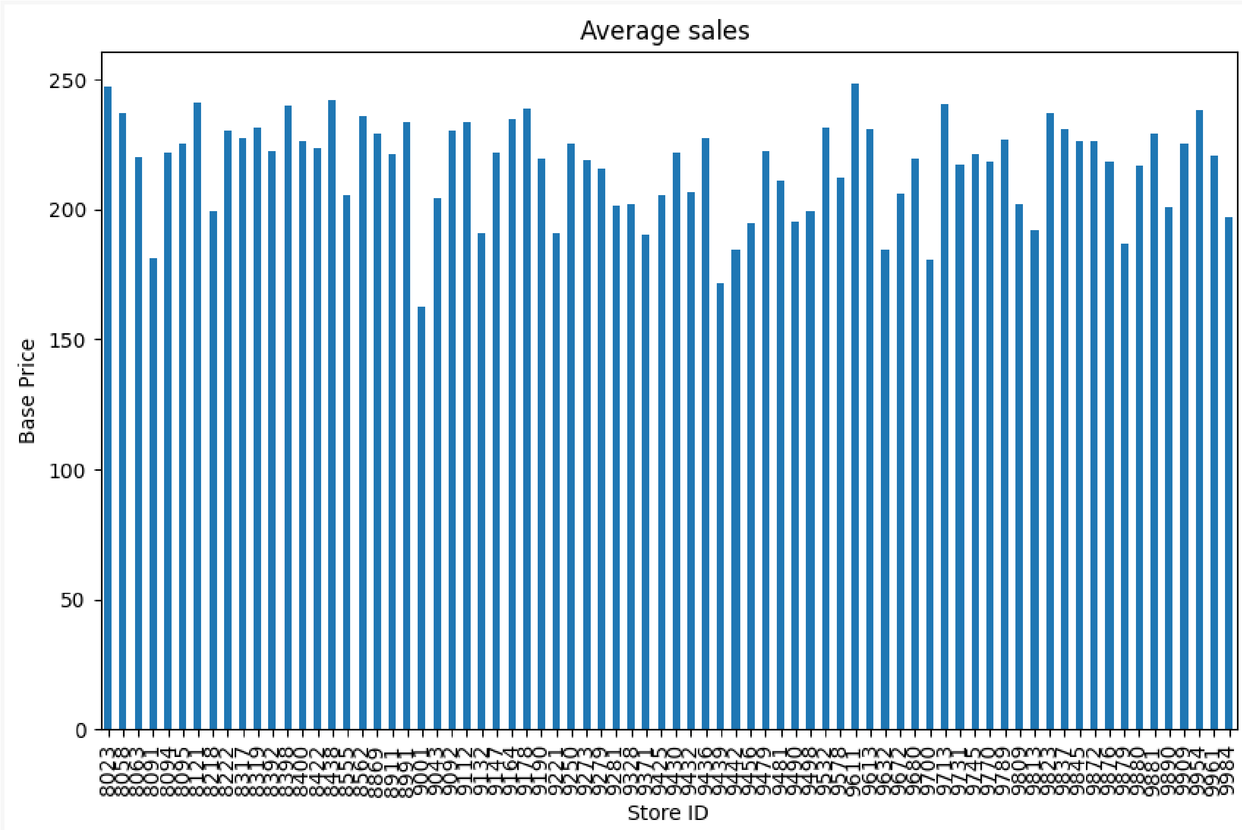
```
Average sales by Store ID
          Total Price  Units Sold  Base Price
Store ID
8023       231.463063  108.600000  247.386525
8058       220.944058   52.747692  237.061904
8063       209.705769   34.714980  220.264038
8091       172.272756   32.805983  181.312372
8094       208.766611   41.821795  221.591154
...               ...         ...         ...
9890       188.861405   59.554438  200.994564
9909       211.796106   66.397802  225.278798
9954       224.677042   34.275566  238.132653
9961       206.792325   70.765611  220.567110
9984       186.580537   37.853394  197.030107
```

[76 rows x 3 columns]

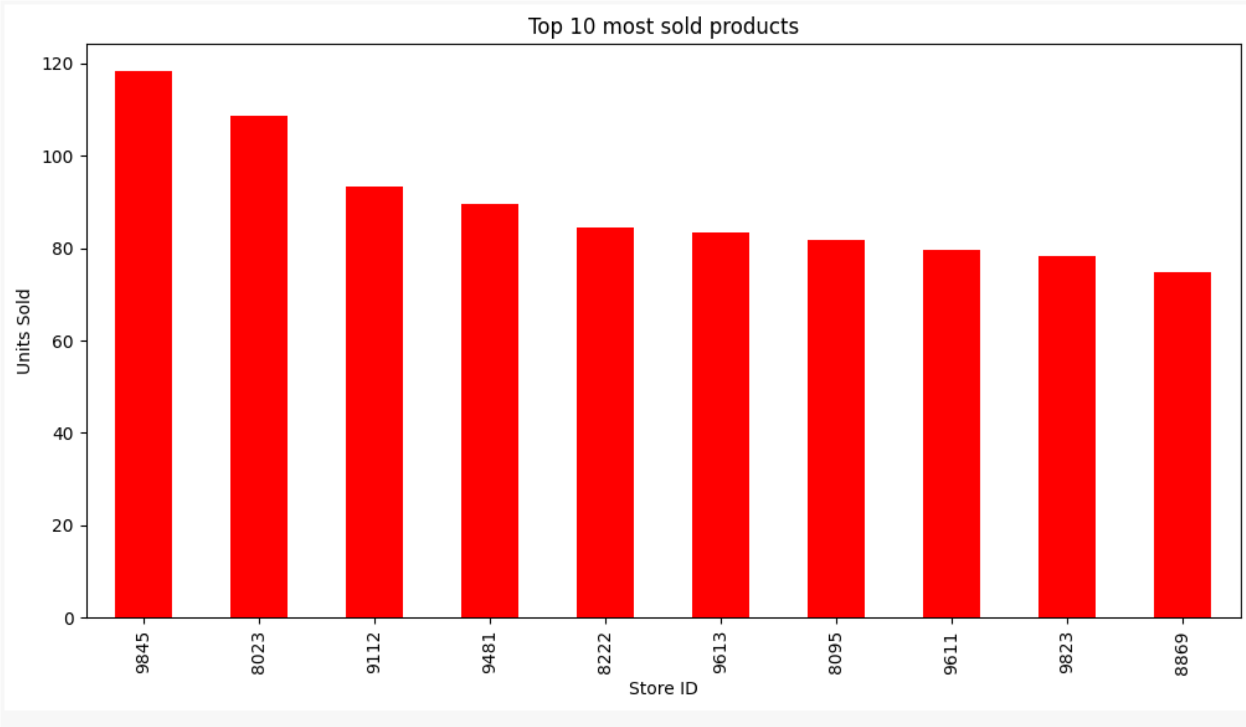




Average sales
Units Sold
Store ID




```
average = df.groupby('Store ID')['Units Sold'].mean()

top_polluted_areas = average.nlargest(10)

plt.figure(figsize=(12, 6))
top_polluted_areas.plot(kind='bar', color='red')
plt.title('Top 10 most sold products')
plt.xlabel('Store ID')
plt.ylabel('Units Sold')
plt.xticks(rotation=90)
plt.show()
```



Conclusion and Recommendations

In the final section of the report, we will summarize the findings and provide recommendations based on real-life comparisons. These recommendations can be used to improve inventory management and sales strategies. We will also discuss any limitations and potential areas for future research.

The real-life comparisons and the report will help stakeholders make informed decisions and fine-tune strategies to optimize product demand predictions and enhance overall business performance.