APPLIED DATA SCIENCE

IBM NAAN MUDHALVAN Phase-2

TEAM MEMBERS

- DHANULESH.S
- SURIYA.V
- NARENTHIRA KUMAR.G
- MAHESWAREN.R
- ARYAN REDDY.R

PROJECT TITTLE : PRODUCT DEMAND ANALYSIS

INTRODUCTION

Product demand prediction using machine learning is a data-driven approach that leverages advanced algorithms and historical data to forecast future demand for specific products. In essence, it's about using

technology to make more accurate predictions about what customers will want in the future. This short note provides an overview of this vital concept:

- Accurate demand prediction is crucial for businesses to optimize their operations, reduce costs, and enhance customer satisfaction. It enables them to meet market demands effectively and efficiently.

- Machine learning models rely on historical sales data, customer behavior patterns, and various external factors to make predictions. The more data available, the more accurate the predictions can be .

- Machine learning algorithms like regression, time series analysis, and neural networks are used to analyze the data and identify trends, seasonality, and other factors influencing demand.

In essence, product demand prediction using machine learning empowers businesses to make proactive decisions based on data-driven insights, ultimately leading to better resource allocation, cost reduction, and improved customer satisfaction.

Content for Project Phase 2:

Consider incorporating time series forecasting techniques like ARIMA or Prophet to capture temporal patterns in demand data.

DATA SOURCE

A good data source for Product Demand Prediction using machine learning should be accurate,complete but it doesn’t have time related column.

Dataset link:

https://www.kaggle.com/datasets/chakradharmattapalli/product-demand-prediction-with-machine-learning Product demand prediction is a crucial process for businesses to optimize inventory management, production planning, and overall supply chain operations. Here is a general procedure for product demand prediction:

1. Data Collection:

- **Gather historical sales** data for the product(s) we want to predict demand for. This data may include sales records, invoices, and customer orders.
- **Collect relevant external data**, such as market trends, economic indicators, seasonality factors, and customer demographics.

2. Data Preprocessing:

- **Clean** and **preprocess the data** to handle missing values, outliers, and inconsistencies.
- Convert data into a structured format suitable for analysis, such as a time series dataset.

3. Data Exploration and Visualization:

- Perform **exploratory data analysis** (EDA) to understand the distribution of data, identify patterns, and detect trends.
- Create visualizations like time series plots, histograms, and scatter plots to gain insights into the data.

4. Feature Engineering:

- Identify relevant features (variables) that may influence product demand. These could include factors like price, promotions, holidays, and seasonality.
- Create new features or transform existing ones to better represent relationships with demand.

5. Model Selection:

- Choose an appropriate demand forecasting model based on the nature of your data. Common models include:
  - Time Series Models (e.g., ARIMA, Exponential Smoothing)
  - Machine Learning Models (e.g., Linear Regression, Random Forest, Neural Networks)

- Deep Learning Models (e.g., Recurrent Neural Networks, LSTM, GRU)

6. Data Splitting:

- Split dataset into training, validation, and testing sets. The training set is used to train the model, the validation set is used to tune hyperparameters, and the testing set is used to evaluate model performance.

7. Model Training:

- Train the selected model(s) using the training data, optimizing the chosen objective function **(e.g.,** Mean Squared Error for regression models**).**
- Experiment with different model hyperparameters to improve performance.

8. Model Evaluation:

- Evaluate the model's performance on the validation and testing sets using appropriate evaluation metrics (e.g., Mean Absolute Error, Root Mean Squared Error, Mean Absolute Percentage Error).
- Compare the performance of different models to select the best one.

9. Hyperparameter Tuning:

- Fine-tune model hyperparameters using techniques like grid search or random search to optimize predictive accuracy.

10. Model Deployment:

- Once you are satisfied with the model's performance, deploy it in a production environment. This may involve integrating it with your inventory management or supply chain system.

11. Monitoring and Maintenance:

- Continuously monitor the model's performance in a real-world setting and retrain it periodically to account for changing demand patterns.
- Update the model with new data and reevaluate its performance over time.

12. Scenario Analysis:

- Use the model to perform scenario analysis, such as "what-if" simulations, to understand how different factors (e.g., price changes, marketing campaigns) may impact future demand.

Remember that demand prediction is an ongoing process, and the accuracy of your predictions can improve as you collect more data and refine your models. Additionally, incorporating domain knowledge and feedback from stakeholders can enhance the effectiveness of your demand prediction system.

Advanced Regression Techniques

Certainly, here's a single-line summary for each of the mentioned models:

1. **Linear Regression:** Linear regression models predict continuous outcomes by assuming a linear relationship between variables.

2. **Random Forest:** Random Forest is an ensemble technique that combines multiple decision trees to improve prediction accuracy and handle complex data.

3. **Gradient Boosting:** Gradient Boosting is an ensemble method that builds decision trees sequentially, optimizing for errors and offering high predictive accuracy.

Program for above regression:

Linear Regression

Import Libraries

```
In [22]: import pandas as pd
         import numpy as np
         import matplotlib.pyplot as plt
         from sklearn.model_selection import train_test_split
         from sklearn.linear_model import LinearRegression
         from sklearn.metrics import mean_squared_error,r2_score
```

Laod CSV File

```
In [3]: data=pd.read_csv('poductDemand.csv')
        data
```

Out[3]:

	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

Remove NaN value

```
In [9]: data=data.dropna()
```

```
In [23]: x = data.drop(columns=['Total Price'])
         y = data['Total Price']
```

Split the data into Training and Testing set

```
In [24]: x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=0)
         model=LinearRegression()
         model.fit(x_train,y_train)

Out[24]: LinearRegression()
```

Make Prediction

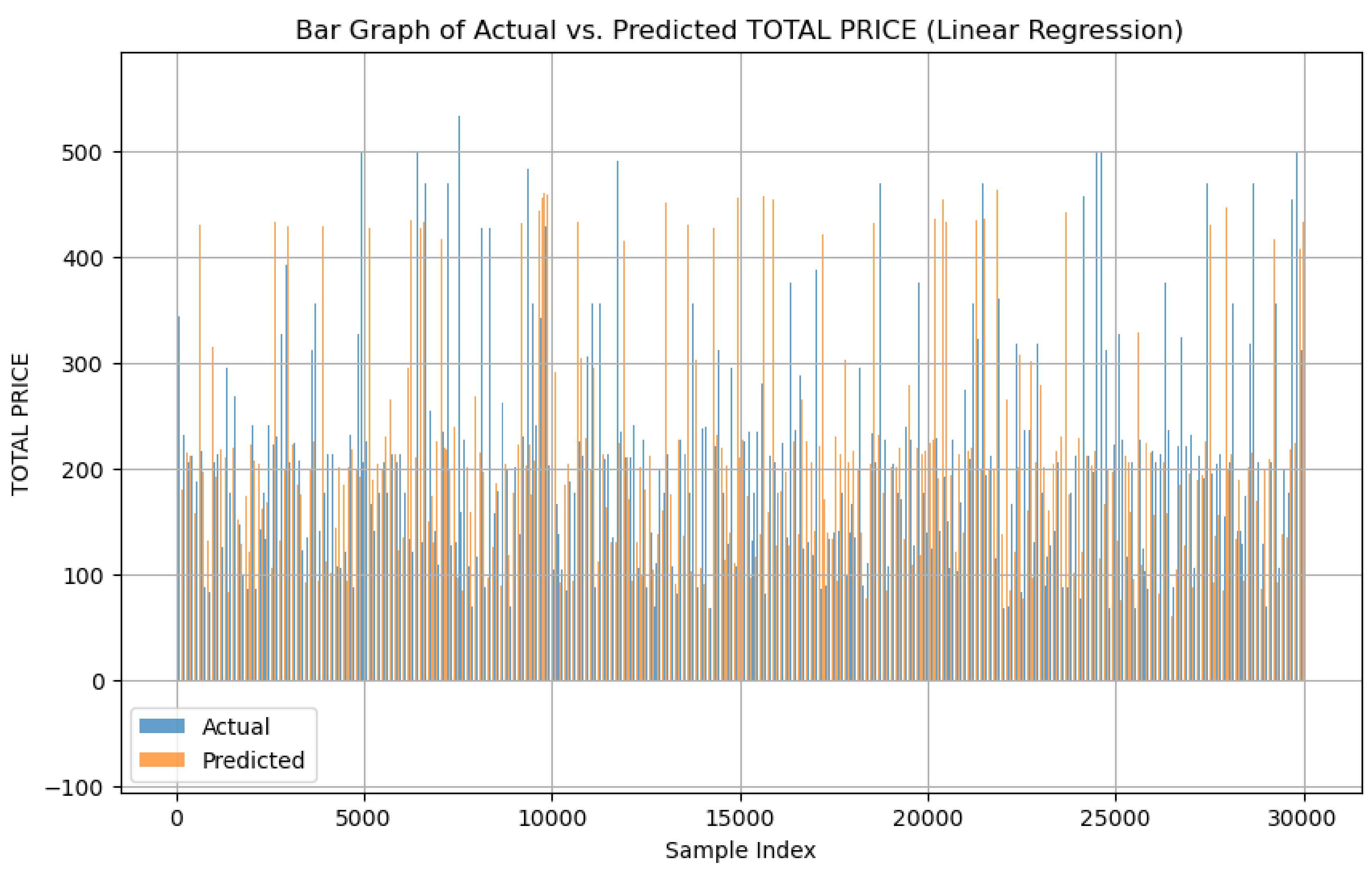
```
In [25]: y_pred=model.predict(x_test)
```

```
In [26]: mse=mean_squared_error(y_test,y_pred)
         r2=r2_score(y_test,y_pred)
         print(f'Mean Squared error(MSE):{mse}')
         print(f'R-Squared (R2):{r2}')

         Mean Squared error(MSE):749.3469180475779
         R-Squared (R2):0.929259349883286
```

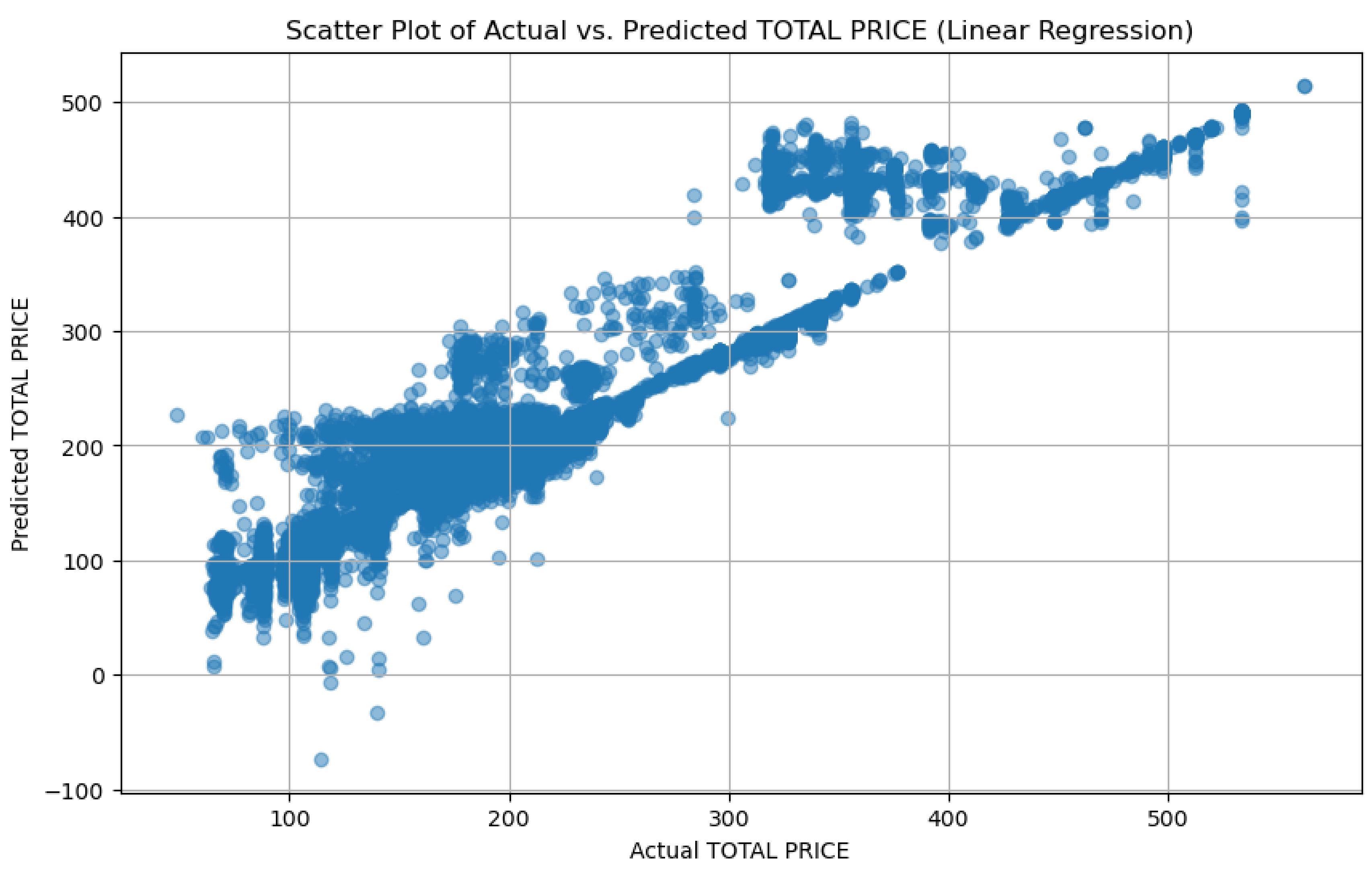
Create a Bar Graph to Visualize Model Performance

```
In [27]: sample_indices = np.arange(len(y_test))
         plt.figure(figsize=(10, 6))
         plt.bar(sample_indices, y_test, width=0.4, label='Actual', align='center', alpha=0.7)
         plt.bar(sample_indices + 0.4, y_pred, width=0.4, label='Predicted', align='center', alpha=0.7)
         plt.xlabel('Sample Index')
         plt.ylabel('TOTAL PRICE')
         plt.title('Bar Graph of Actual vs. Predicted TOTAL PRICE (Linear Regression)')
         plt.legend()
         plt.grid(True)
         plt.show()
```



Create a Scatter Plot to Visualize Model Performance

```
In [28]: plt.figure(figsize=(10, 6))
         plt.scatter(y_test, y_pred, alpha=0.5)
         plt.xlabel('Actual TOTAL PRICE')
         plt.ylabel('Predicted TOTAL PRICE')
         plt.title('Scatter Plot of Actual vs. Predicted TOTAL PRICE (Linear Regression)')
         plt.grid(True)
         plt.show()
```



Random Forest Regressor

Import Libraries

In [2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import plotly.express as px
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error,r2_score
```

Laod CSV File

In [3]:
```
data=pd.read_csv('poductDemand.csv')
data
```

Out[3]:

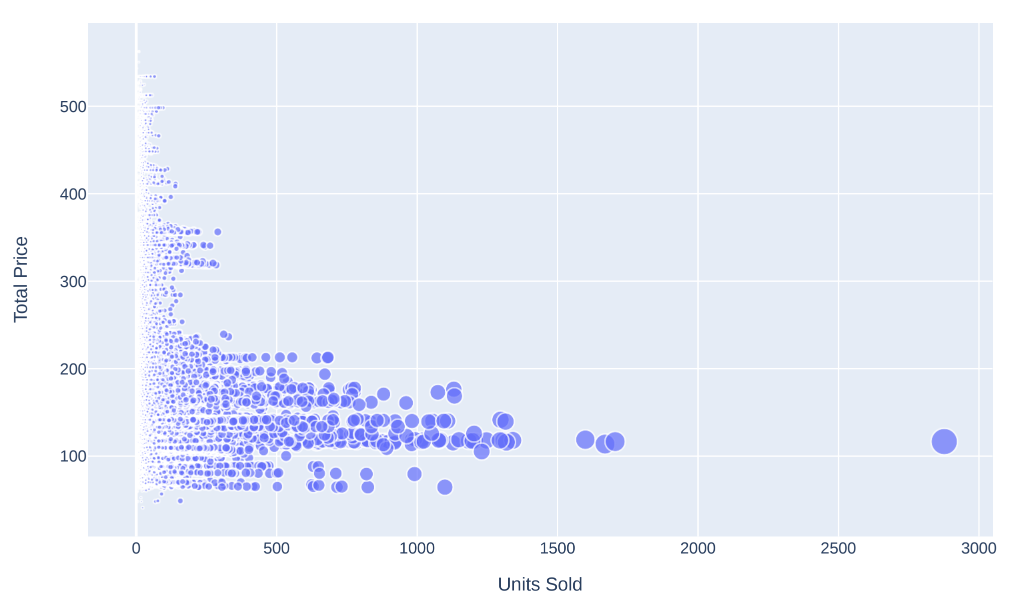
	ID	Store ID	Total Price	Base Price	Units Sold
0	1	8091	99.0375	111.8625	20
1	2	8091	99.0375	99.0375	28
2	3	8091	133.9500	133.9500	19
3	4	8091	133.9500	133.9500	44
4	5	8091	141.0750	141.0750	52
...	...	...	...	...	...
150145	212638	9984	235.8375	235.8375	38
150146	212639	9984	235.8375	235.8375	30
150147	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

Remove NaN value

In [9]:
```
data=data.dropna()
```

In [5]:
```
fig=px.scatter(data,x="Units Sold",y="Total Price",size="Units Sold")
fig.show()
```



Split the data into features(x) and target(y)

In [15]:
```
x=data[['ID','Store ID','Base Price','Units Sold']]
y=data['Total Price']
```

Split the data into Training and Testing set

In [16]:
```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=0)
model=RandomForestRegressor(n_estimators=100,random_state=0)
model.fit(x_train,y_train)
```

Out[16]:
```
RandomForestRegressor(random_state=0)
```

Make Prediction

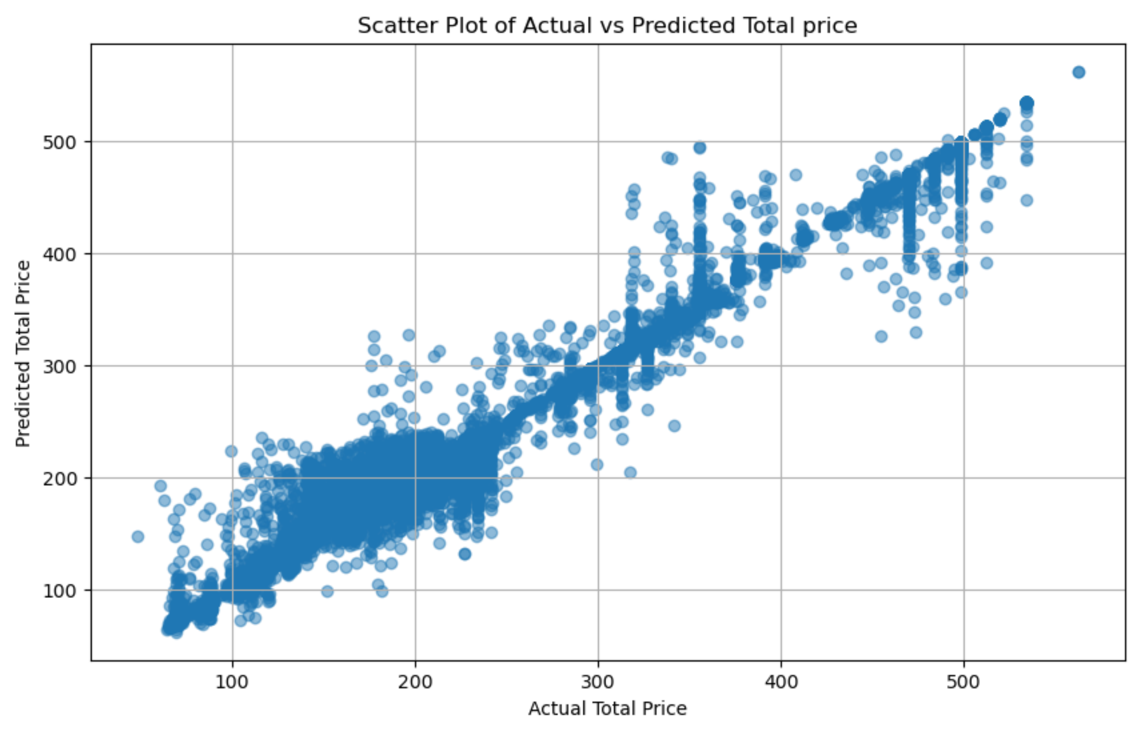
In [17]:
```
y_pred=model.predict(x_test)
```

In [18]:
```
mse=mean_squared_error(y_test,y_pred)
r2=r2_score(y_test,y_pred)
print(f'Mean Squared error(MSE):{mse}')
print(f'R-Squared (R2):{r2}')
```

```
Mean Squared error(MSE):150.45419255618455
R-Squared (R2):0.9857966622162927
```

Create a Scatter Plot to Visualize Model Performance

In [19]:
```
plt.figure(figsize=(10,6))
plt.scatter(y_test,y_pred,alpha=0.5)
plt.xlabel('Actual Total Price')
plt.ylabel('Predicted Total Price')
plt.title('Scatter Plot of Actual vs Predicted Total price')
plt.grid(True)
plt.show()
```



Gradient Boosting Regression

Import Libraries

In [2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.metrics import mean_squared_error, r2_score
```

Laod CSV File

In [3]:
```
data=pd.read_csv('poductDemand.csv')
data
```

Out[3]:

	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

Remove NaN value

In [4]:
```
data=data.dropna()
```

In [5]:
```
x = data.drop(columns=['Total Price'])
y = data['Total Price']
```

Split the data into Training and Testing set

In [6]:
```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=0)
model = GradientBoostingRegressor(n_estimators=100, random_state=0)
model.fit(x_train,y_train)
```

Out[6]:
```
GradientBoostingRegressor(random_state=0)
```

Make Prediction

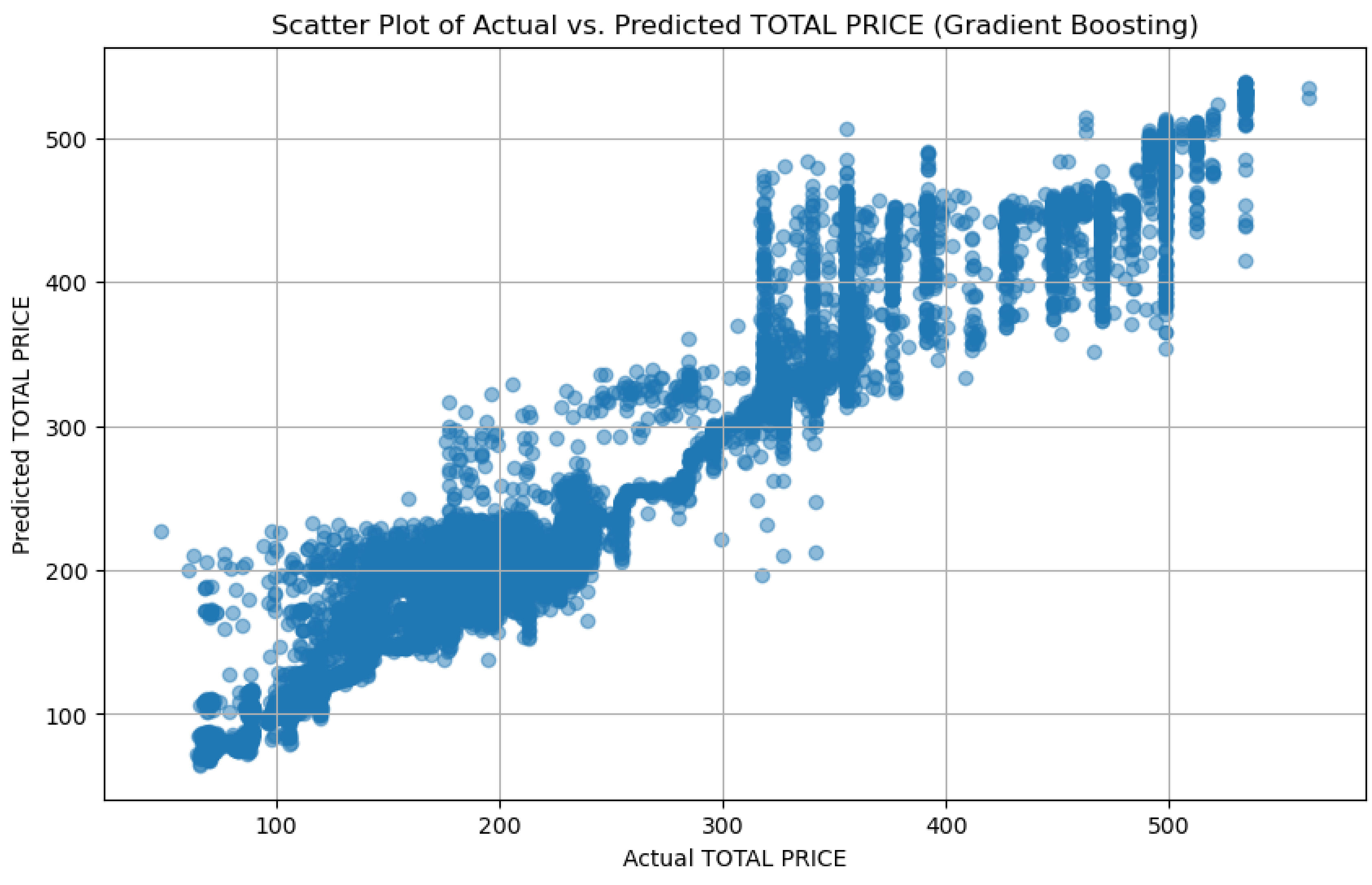
In [7]:
```
y_pred=model.predict(x_test)
```

In [8]:
```
mse=mean_squared_error(y_test,y_pred)
r2=r2_score(y_test,y_pred)
print(f'Mean Squared error(MSE):{mse}')
print(f'R-Squared (R2):{r2}')
```

```
Mean Squared error(MSE):396.4389864759951
R-Squared (R2):0.9625749423137783
```

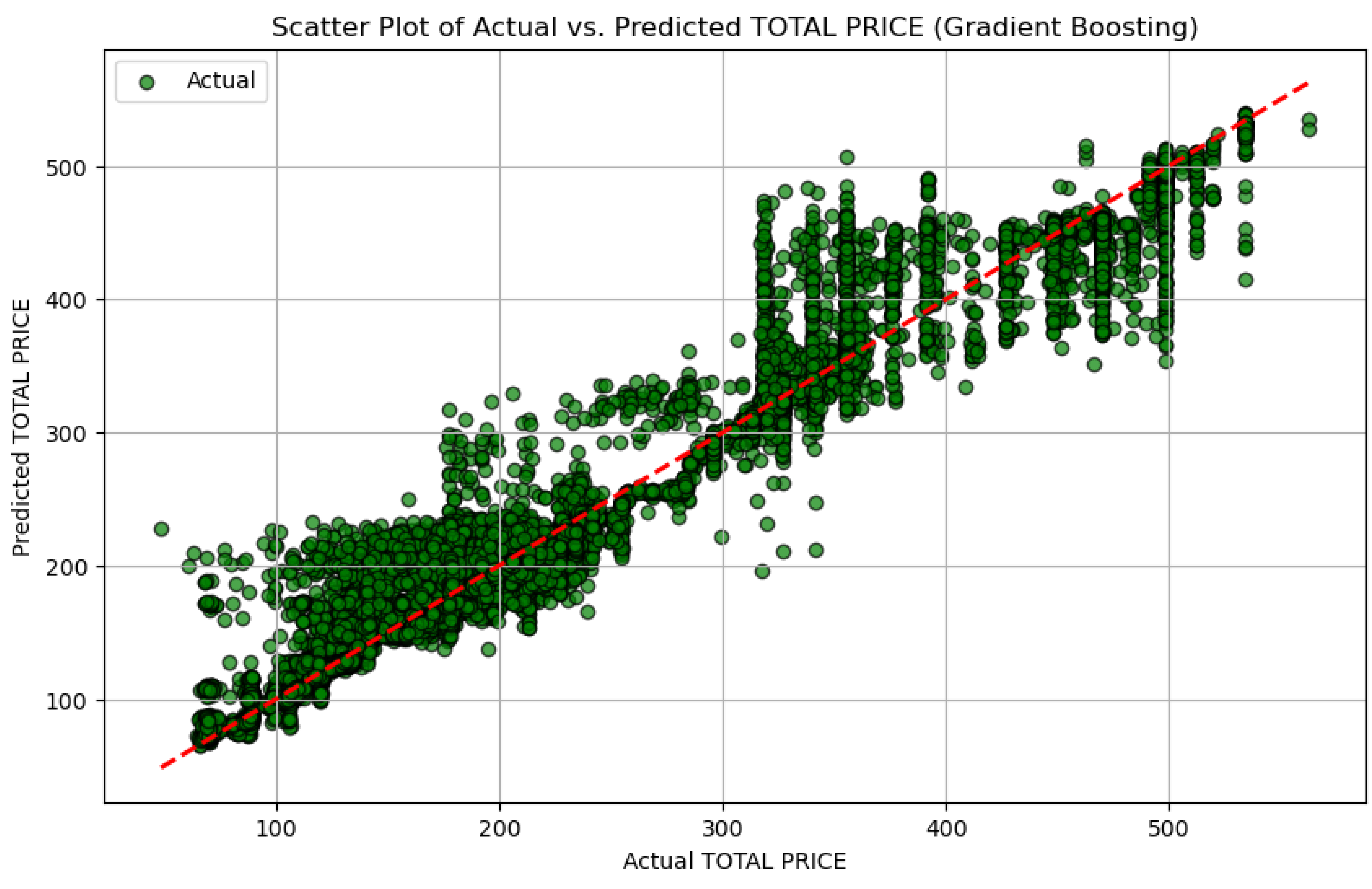
Create a Scatter Plot to Visualize Model Performance

In [9]:
```
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred, alpha=0.5)
plt.xlabel('Actual TOTAL PRICE')
plt.ylabel('Predicted TOTAL PRICE')
plt.title('Scatter Plot of Actual vs. Predicted TOTAL PRICE (Gradient Boosting)')
plt.grid(True)
plt.show()
```



Create a scatter plot to visualize model performance with different colors

In [10]:
```
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred, alpha=0.7, c='g', label='Actual', edgecolors='k')
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], linestyle='--', color='r', linewidth=2)
plt.xlabel('Actual TOTAL PRICE')
plt.ylabel('Predicted TOTAL PRICE')
plt.title('Scatter Plot of Actual vs. Predicted TOTAL PRICE (Gradient Boosting)')
plt.legend()
plt.grid(True)
plt.show()
```



CONCLUSION FOR PRODUCT DEMAND PREDICTION

In conclusion, product demand prediction using machine learning and applied regressions, including linear regression, Random Forest, and Gradient Boosting, offers a powerful toolkit for businesses to make data-driven decisions.

Linear regression provides simplicity and interpretability, Random Forest excels in handling complex relationships and feature importance, while Gradient Boosting offers superior predictive accuracy.

The choice of the specific model depends on the complexity of the data and the desired level of prediction accuracy, ensuring that businesses have the flexibility to tailor their demand forecasting strategies to meet their unique needs.