Phase 2 : Product Demand Analysis

Abstract:

Supply and demand are two fundamental concepts of sellers and customers. Predicting demand accurately is critical for organisations in order to be able to make plans. The main aim of our project is to develop a machine learning model(Decision Tree Regression, Random Forest Regression) that forecasts product demand based on historical sales data.

2.1 Problem Description:

The problem is to create a machine learning model that forecasts product demand based on historical sales data and external factors. The goal is to help businesses optimise inventory management and production planning to efficiently meet customer needs. This project involves data collection, data pre-processing, feature engineering, model selection, training, and evaluation.

2.2 Dataset Information:

The primary goal of this project is to create a predictive model that can estimate the demand for the product in the market under different pricing strategies. By analysing historical sales data, the project aims to identify the price point at which the product is perceived as a better deal compared to competitors, leading to increased sales.

1. **Data Collection**: Historical sales data

   Data Link: https://www.kaggle.com/datasets/chakradharmattapalli/product-demand-prediction-with-machine-learning

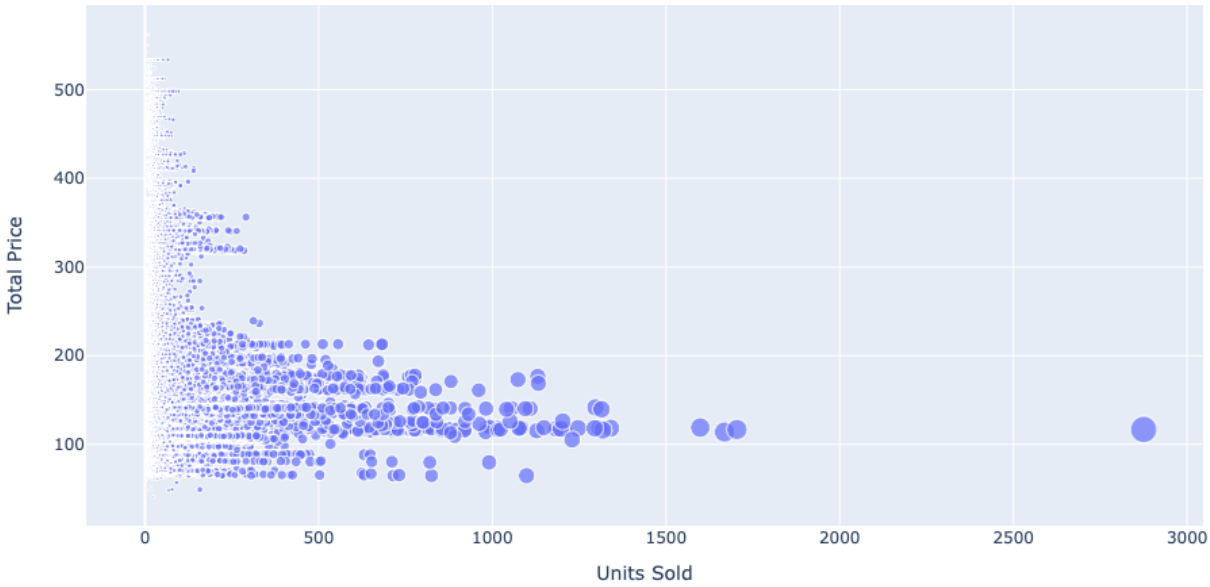
2. **Data Pre-processing**: Clean and pre-process the data, handle missing values, and convert categorical features into numerical representations
3. **Feature Engineering**: Create additional features that capture seasonal patterns, trends, and external influences on product demand.
4. **Model Selection**: Choose suitable regression algorithms (e.g., Decision Tree, Random Forest, Gradient Boost) for demand forecasting.
5. **Model Training**: Train the selected model using the pre-processed data.
6. **Evaluation**: Evaluate the model's performance using appropriate regression metrics (e.g., Mean Absolute Error, Root Mean Squared Error ,R Squared).

2.3 Dataset Columns:

The dataset provided for this task includes the following key information:

- **Product ID**: Each product in the company's inventory is assigned a unique identifier, allowing us to distinguish between different items.

- **Store ID**: This represents the specific store or location where the product was sold. It helps in understanding regional variations in demand.
- **Total Price**: This is the actual price at which the product was sold to customers during the sales transaction. It includes any discounts or promotions applied.
- **Base Price**: The base price is the original or standard price at which the product is typically sold before any discounts or promotions are applied.
- **Units Sold (Quantity Demanded)**: This is the quantity of the product that was purchased by customers at a given price point. It reflects the demand for the product at that particular price.



Plot of Total Price vs Units Sold

Training Data:

1. **Independent Variables (Features):**
    a. Total Price: The total price of a product.
    b. Base Price: The base price of the product.
2. **Dependent Variable (Target):**
    a. Units Sold: The number of units of the product sold.

During the training phase, machine learning model uses "Total Price" and "Base Price" as independent variables to predict the "Units Sold."

Testing Data:

For model testing or prediction:

1. **Input Features:**
    a. Total Price: The total price of a product.
    b. Base Price: The base price of the product.
2. **Output Feature (Prediction):**
    a. Units Sold: The model will predict the number of units of the product that are expected to be sold based on the provided "Total Price" and "Base Price."

In summary, the trained model takes "Total Price" and "Base Price" as input and predicts the "Units Sold" as the output. This setup allows to estimate the number of units sold for a given total price and base price combination.

2.4.1 Modules/Libraries used:

- Pandas
- Numpy
- Seaborn
- Matplotlib
- sklearn.model_selection
- sklearn.ensemble
- sklearn.metrics

Pandas :

Pandas is a powerful Python library for data manipulation and analysis. It provides data structures like DataFrame for tabular data and Series for one-dimensional data. With built-in functions, it facilitates tasks such as filtering, grouping, and merging, making data handling in Python concise and efficient.

Numpy :

Numpy is a fundamental Python library for numerical operations, offering a powerful array object. It enables efficient mathematical operations on large datasets. With a wide range of functions, it's essential for tasks like linear algebra, statistical analysis, and random number generation.

Seaborn :

Seaborn is a data visualisation library for Python based on Matplotlib. It simplifies creating informative and attractive statistical graphics. With high-level abstractions, it streamlines the process of producing aesthetically pleasing charts like heatmaps and violin plots.With built-in themes and colour palettes, seaborn enhances the aesthetics of your plots effortlessly.

Matplotlib :

Matplotlib is a powerful Python library for creating static, animated, and interactive visualisations. It offers a flexible and comprehensive set of plotting tools for a wide range of data visualisation needs. With a simple interface, it allows fine-tuning of every aspect of a plot. Matplotlib is the go-to choice for generating publication-quality charts and graphs.

Sklearn.model_selection :

sklearn.model_selection in scikit-learn is crucial for model evaluation and parameter tuning in machine learning. It provides functions for splitting datasets into train and test sets, cross-validation strategies, and hyperparameter tuning with grid search. The “train_test_split” method easily divides data, while “cross_val_score” facilitates cross-validation. GridSearchCV automates hyperparameter tuning.

Sklearn.ensemble :

sklearn.ensemble in scikit-learn is a module for ensemble learning techniques, combining multiple models for improved performance. Random Forests and Gradient Boosting are popular algorithms provided. These ensembles enhance model robustness and predictive accuracy. The module offers easy-to-use interfaces for building, training, and evaluating ensemble models.

Sklearn.metrics :

"sklearn.metrics" in scikit-learn provides a variety of evaluation metrics for assessing model performance. It includes functions for classification, regression, and clustering metrics. Regression metrics like R-Squared(R^2), Mean Absolute Error (MAE), Mean Squared Error (MSE), Root Mean Squared Error (RMSE).

2.4.2 Models used:

- Decision Tree Regressor
- Gradient Boosting Regressor
- Random Forest Regressor

I. Decision Tree Regressor

Decision Tree Regressor used in machine learning for predicting continuous values. Unlike decision trees for classification, which predict categorical labels, decision tree regression predicts a continuous target variable.

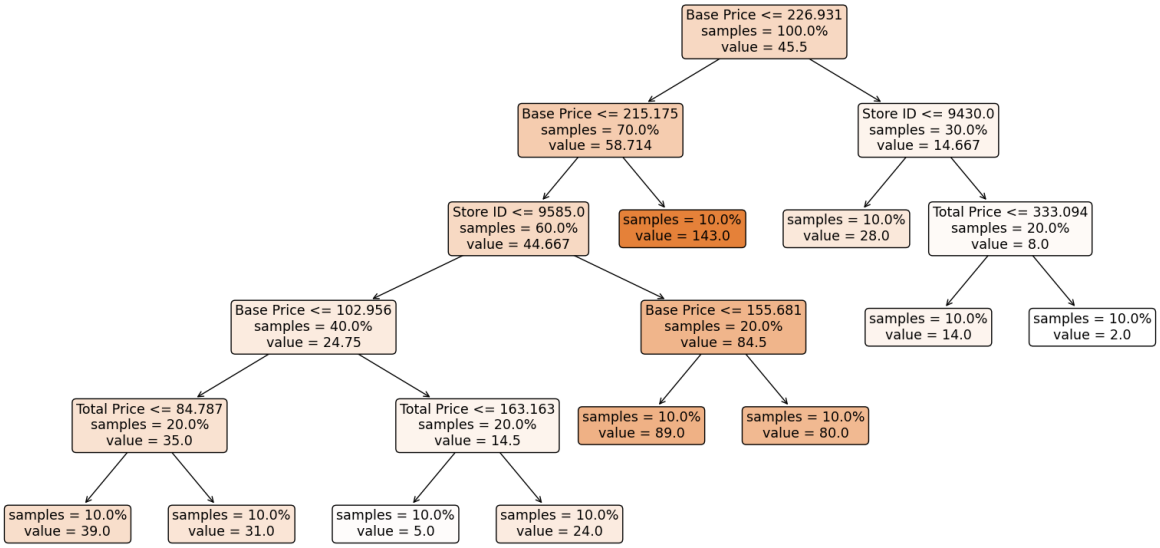
1.**Tree Structure**: Like its classification counterpart, a decision tree for regression is a tree-like model composed of nodes. Each internal node represents a decision based on a feature, and each leaf node (terminal node) represents the predicted continuous value.

2.**Splitting Criteria**:The algorithm selects the feature and the split point that minimises the sum of squared differences between the predicted values and the actual values in each partition. The goal is to create homogeneous subsets in terms of the target variable.

3.**Prediction**: To predict the target value for a new instance, we traverse the tree from the root to a leaf, following the decision rules at each node.
4.**Disadvantages : Overfitting** - Decision trees can be prone to overfitting, capturing noise in the training data. Techniques like pruning or setting a minimum number of samples required to make a split can help alleviate this.

5.**Advantages**: Decision trees are interpretable, easy to understand, and can handle both numerical and categorical data.

Decision Tree Regressor often used in scenarios where the relationship between features and the target variable is nonlinear.



Decision Tree Model of the Dataset

II. Gradient Boosting Regressor :

Gradient Boosting Regressor is a machine learning model that belongs to the family of ensemble methods, specifically gradient boosting. It's an ensemble of decision trees, and it builds trees sequentially, with each tree correcting the errors of the previous one.

1.**Decision Trees as Base Learners:** Gradient Boosting Regressor builds decision trees as its base learners. These are often shallow trees to avoid overfitting.

2.**Sequential Learning:** Trees are added to the model sequentially. Each tree corrects the errors (residuals) of the combined ensemble of the previous trees.

3.**Gradient Descent Optimization:** The term "gradient" in Gradient Boosting comes from the optimization method used, which is gradient descent. It minimises the loss function by moving in the direction of steepest descent (negative gradient).

4.**Learning Rate:** The learning rate is a hyperparameter that determines the contribution of each tree to the final prediction. A lower learning rate requires more trees but can lead to better generalisation.

5.**Regularization:** Gradient Boosting Regressor includes regularisation techniques to prevent overfitting.

6.**Loss Function:** The choice of the loss function depends on the type of problem (regression or classification). For regression, the mean squared error is often used.

The purpose of Gradient Boosting Regressor is similar to that of a Decision Tree Regressor—to predict a continuous target variable. However, it often performs better in terms of accuracy by combining the strengths of multiple weak learners (simple decision trees) to form a strong predictive model.

III. Random Forest Regressor :

Random Forest Regressor is a machine learning algorithm used for regression tasks. It's an ensemble learning method, meaning it combines the predictions of multiple individual models to create a stronger overall model. In the case of Random Forest, these individual models are decision trees.

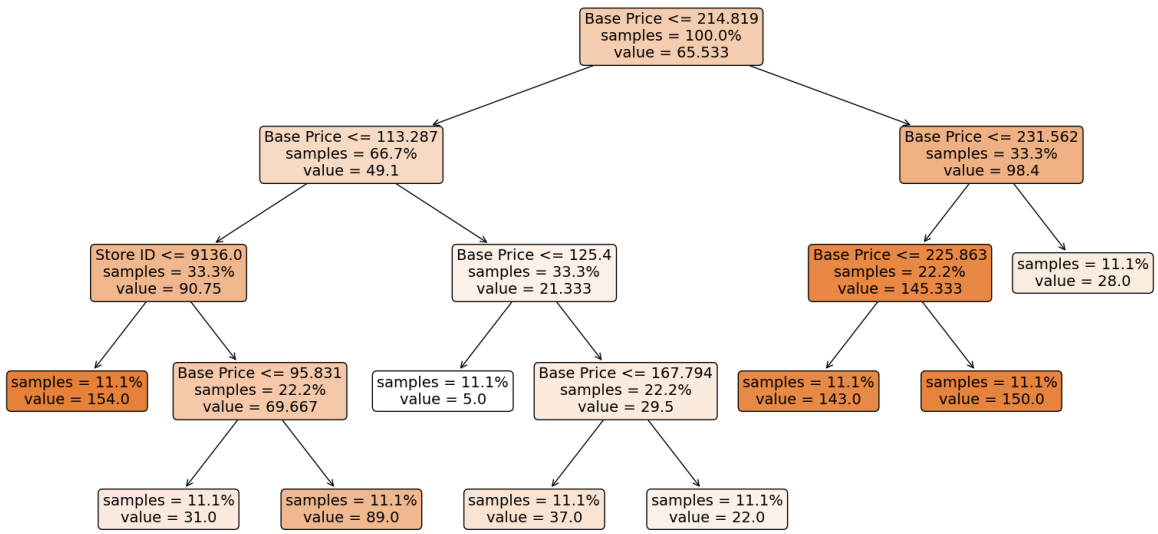
1.**Decision Trees**: Each tree in the forest is a decision tree. Decision trees recursively split the dataset into subsets based on the most significant attribute at each node. This process continues until a stopping criterion is met (e.g., a certain depth is reached).

2.**Randomness**: The "random" part comes in two ways:

- **Random Subsets**: When building each tree, a random subset of the features is selected for making decisions at each node. This helps in decorrelating the trees and prevents overfitting.
- **Bootstrapping**: Each tree is trained on a random subset of the training data. This is known as bootstrapping, where samples are drawn with replacement. This further introduces diversity among the trees.

3.**Voting/Averaging**: Once all the trees are built, for regression tasks, the predictions from each tree are averaged to get the final prediction. This ensemble approach tends to be more robust and less prone to overfitting compared to individual trees.

4.**Robustness**: Random Forests are robust to outliers in the data and handle non-linear relationships well. They also provide a way to assess feature importance.



Random Forest Regression Model for the Dataset

2.5 Train and Test:

Training and testing a machine learning model involves several steps.

- **Data Splitting:** The train_test_split function is used to split the dataset into training and testing sets. This allows us to train the model on one subset of the data and test its performance on another, unseen subset.

train_test_split is a function from the sklearn.model_selection module in scikit-learn. It's a commonly used function for splitting a dataset into two subsets: one for training our machine learning model and the other for testing its performance.

The train_test_split function splits arrays or matrices into random train and test subsets. It takes several parameters, including your features (X) and labels (y), and splits them into four subsets: X_train, X_test, y_train, and y_test.

SOURCE CODE:

```
from sklearn.model_selection import train_test_split

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

- **X** is a feature matrix.
- **y** is the target variable (labels).
- **test_size** is the proportion of the dataset to include in the test split. In this case, 20% of the data will be used for testing (test_size=0.2).
- **random_state** is a seed for the random number generator. Providing a specific seed ensures reproducibility. Different seeds will result in different random splits.

After running the code, we will have four datasets:

- **X_train**: The features for training your model.
- **X_test**: The features for testing your model.
- **y_train**: The corresponding labels for training.
- **y_test**: The corresponding labels for testing.

- **Model Initialization:** The machine learning model (Random Forest Regressor in this case) is initialised with specified hyperparameters

- **Training the Model:** The fit function is used to train the model using the training data (X_train and y_train).

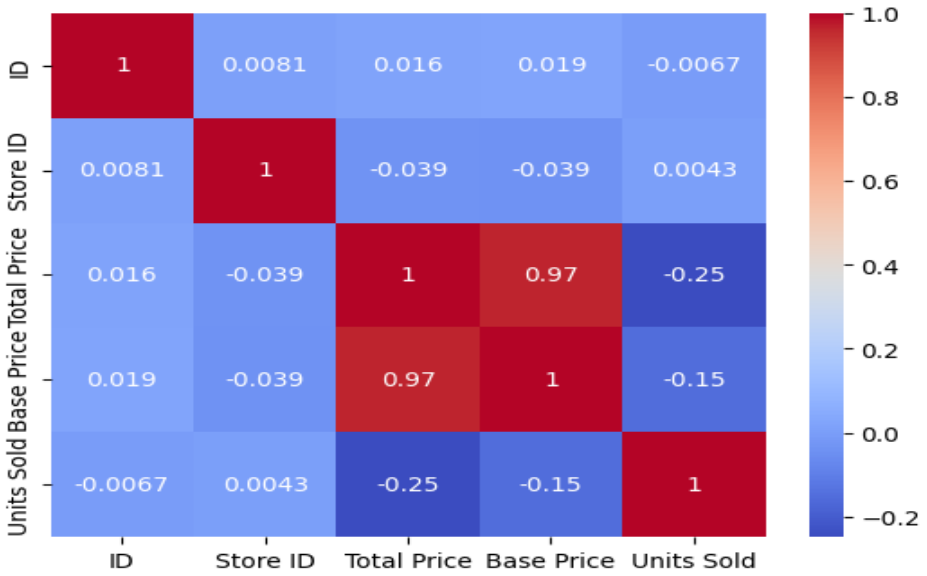
```
#Model Creation
# Create and train a Gradient Boosting Regressor model
model = GradientBoostingRegressor(n_estimators=100, random_state=42)
model.fit(X_train, y_train)
```

- **Making Predictions:** The trained model is used to make predictions on the test data (X_test). The resulting predictions are stored in the predictions variable.

```
# Predict a test data with a trained model
y_pred = model.predict(X_test)
```

2.6 Data Correlation:

A correlation matrix is a table showing correlation coefficients between variables. Each cell in the table shows the correlation between two variables. A correlation matrix is used to summarise data, as an input into a more advanced analysis, and as a diagnostic for advanced analyses.



Heat Map for the Correlation Matrix

	ID	Store ID	Total Price	Base Price	Units Sold
ID	1.000000	0.007461	0.008473	0.018911	-0.010608
Store ID	0.007461	1.000000	-0.038315	-0.038855	-0.004369
Total Price	0.008473	-0.038315	1.000000	0.958885	-0.235625
Base Price	0.018911	-0.038855	0.958885	1.000000	-0.140022
Units Sold	-0.010608	-0.004369	-0.235625	-0.140022	1.000000

Correlation Matrix

2.6.1 Insights:

1. **Total Price and Base Price (Correlation: 0.958885)** : Total Price and Base Price have a very strong positive correlation. This means that there is a strong linear relationship between the total price and the base price of the products. As the base price increases, the total price tends to increase accordingly.

2. **Total Price and Units Sold (Correlation: -0.235625)** : Total Price and Units Sold have a moderate negative correlation. This suggests that as the total price of the products increases, the number of units sold tends to decrease. Customers may buy fewer units when the price is higher.

3. **Base Price and Units Sold (Correlation: -0.140022)** : Base Price and Units Sold also have a moderate negative correlation. This indicates that as the base price of the products increases, the number of units sold tends to decrease. This relationship is similar to the one observed with total price.

2.6.2 Why does the **Linear Regression** model **fail**?

```
R2: 0.1490719889302594
MAE: 32.49815502614977
MSE: 2784.619420940248
RMSE: 52.76949327916886
```

Accuracy Metrics Linear Regression Model

By observing the correlation matrix, high correlation (0.96) among the independent variables and a relatively weak correlation (-0.26) between the dependent and independent variables, linear regression might not be the best choice for modelling this relationship effectively. Because:

1. Multicollinearity:

In linear regression, highly correlated independent variables can lead to multicollinearity. Multicollinearity occurs when two or more independent variables in a regression model are highly correlated, making it difficult for the model to estimate the individual effect of each variable on the dependent variable accurately. This situation can lead to unstable coefficient estimates, making interpretations and predictions unreliable.

2. Weak Correlation with the Dependent Variable:

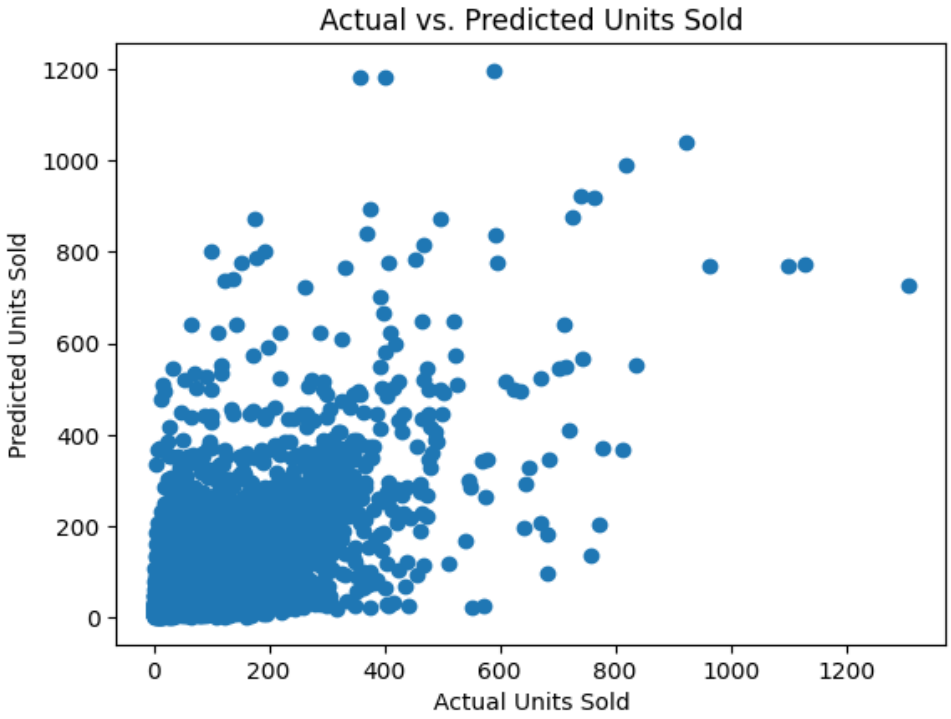
The weak correlation between the independent variables and the dependent variable (-0.26) suggests a weak linear relationship. Linear regression assumes a linear relationship between independent and dependent variables. When the correlation is weak, linear regression might not capture the underlying patterns effectively, leading to poor predictions.

3. Overfitting:

In the case of high multicollinearity, linear regression might fit the training data very well (overfitting), but it may fail to generalise to new, unseen data. Overfitting occurs when the model learns the noise in the training data instead of the actual underlying patterns, leading to poor performance on new data.

Accuracy Comparison:

Decision Tree

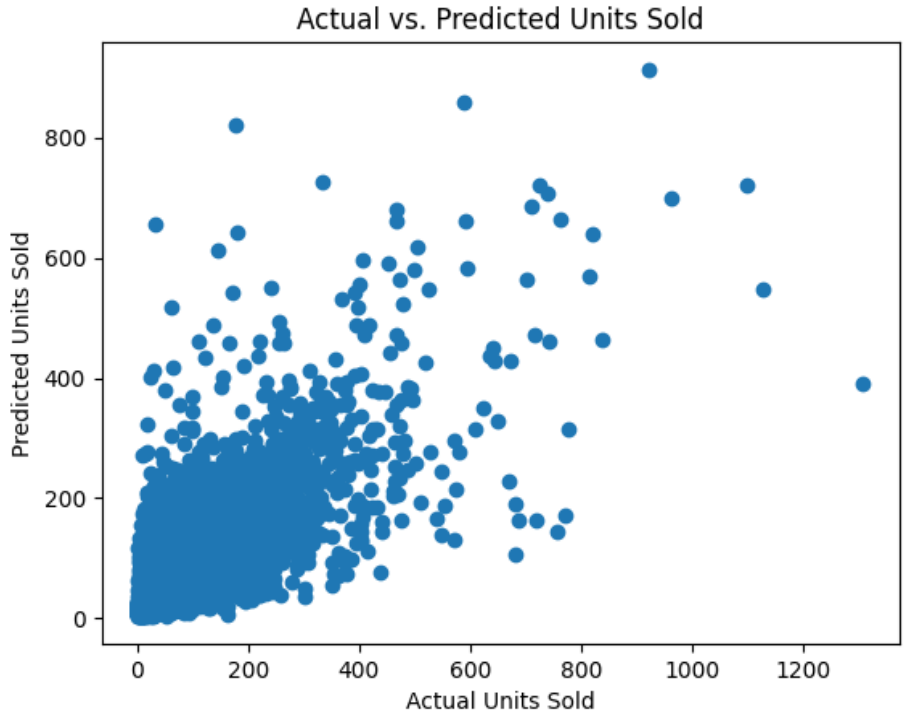


Decision Tree Regression Model :
Plot of Actual Units Sold vs Predicted Units Sold

```
R2: 0.46182404986623227
MAE: 20.54729114753958
MSE: 1761.15392034337
RMSE: 41.96610442182321
```

Accuracy Metrics for Decision Tree

Random Forest

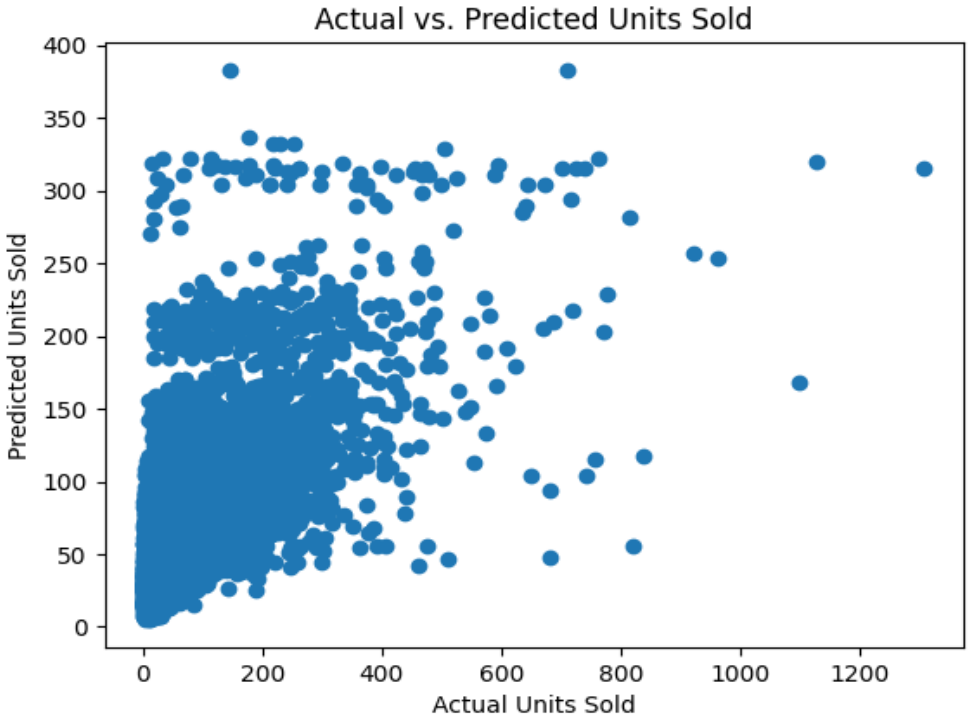


Random Forest Regression Model :
Plot of Actual Units Sold vs Predicted Units Sold

```
R2: 0.6136579174937777
MAE: 18.93038515316013
MSE: 1264.285171104979
RMSE: 35.55678797508261
```

Accuracy Metrics for Random Forest Model

Gradient Boosting



Gradient Boosting Regression Model :
Plot of Actual Units Sold vs Predicted Units Sold

```
R2: 0.4273579061461198
MAE: 25.559876376161668
MSE: 1873.9426544306266
RMSE: 43.28905929251208
```

Accuracy Metrics for Gradient Boosting Model

2.7 Metrics:

R-Squared(R^2)

R^2, or the coefficient of determination, measures how well a regression model predicts the variance in the dependent variable. It ranges from 0 to 1, where 1 indicates perfect predictions. It's a handy tool to evaluate the goodness of fit in regression analysis.

Mean Absolute Error (MAE) :

Mean Absolute Error (MAE) is a metric that calculates the average absolute differences between predicted and actual values in a dataset. It provides a straightforward measure of prediction accuracy. With MAE, the absolute errors are summed up and divided by the number of observations, giving a clear average error value. Unlike some other metrics, MAE treats all errors equally, making it robust and easy to interpret.

Mean Squared Error (MSE):

Mean Squared Error (MSE) measures the average of the squared differences between predicted and actual values. It's a common metric in regression analysis. Squaring the errors emphasises larger discrepancies, giving a comprehensive view of overall model performance. However, because it squares the errors, it's sensitive to outliers.

Root Mean Squared Error (RMSE) :

Root Mean Squared Error (RMSE) is the square root of the Mean Squared Error (MSE). It shares the same intuitive interpretation as the original data's unit, making it more easily understandable. RMSE penalises large errors, offering a balanced view of prediction accuracy. Like MSE, it's widely used in regression analysis. Lower RMSE values indicate better model performance, and it provides a sense of the average magnitude of prediction errors in the original units of the data.

Comparison of Models

Metrics/Model	Decision Tree	Random Forest	Gradient Boosting
R-squared	0.46	0.61	0.43
Mean Absolute Error(MAE)	20.55	18.93	25.56
Mean Squared Error (MSE)	1761.15	1264.28	1873.94
Root Mean Squared Error (RMSE)	41.97	35.56	43.29

Conclusion:

In conclusion, after thorough evaluation of Random Forest, Gradient Boosting, and Decision Tree models for product demand prediction, Random Forest emerged as the superior choice with an R-squared score of 0.61. Its exceptional ability to capture complex patterns and provide accurate forecasts positions it as the most reliable model for predicting future product demand, making it the recommended solution for our problem statement.