# Product Demand Prediction

**Presented by:**

**Kashish Bai**
**Ehsan Hussain**
**Muhammad Usman**

# 2.1 Problem description

A product company plans to offer discounts on its product during the upcoming holiday season. The company wants to find the price at which its product can be a better deal compared to its competitors. For this task, the company provided a dataset of past changes in sales based on price changes. You need to train a model that can predict the demand for the product in the market with different price segments.

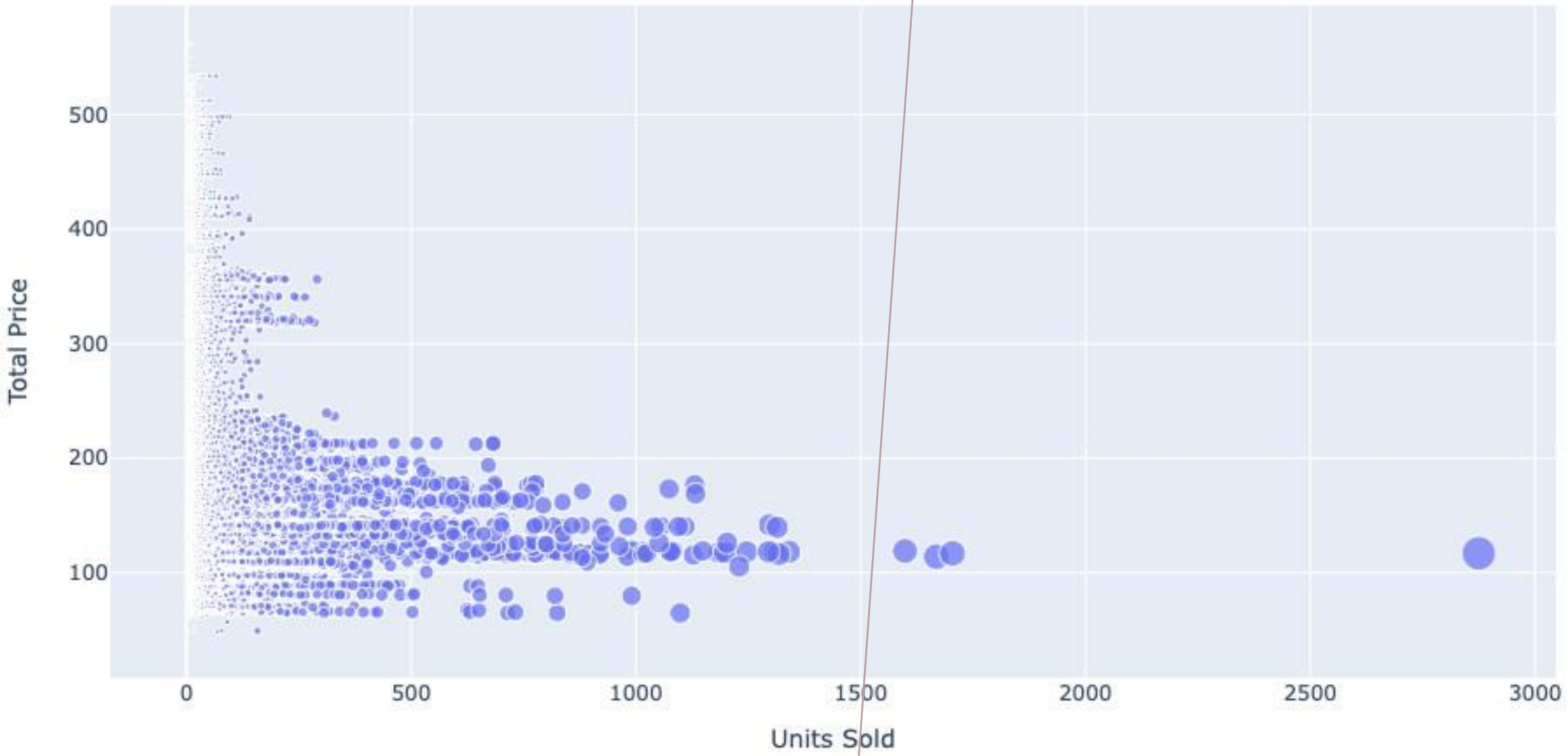
# 2.2 Dataset Information:

- **Data Collection**: Historical sales data
- **Dataset Link:** DATASET
- **Data Pre-processing**: Clean and pre-process the data, handle missing values.
- **Model Selection**: Choose suitable regression algorithms (e.g., Decision Tree, Random Forest, Gradient Boost) for demand forecasting.
- **Model Training**: Train the selected model using the pre-processed data.
- **Evaluation**: Evaluate the model's performance using appropriate regression metrics (e.g., Mean Absolute Error, Root Mean Squared Error,R- Squared).

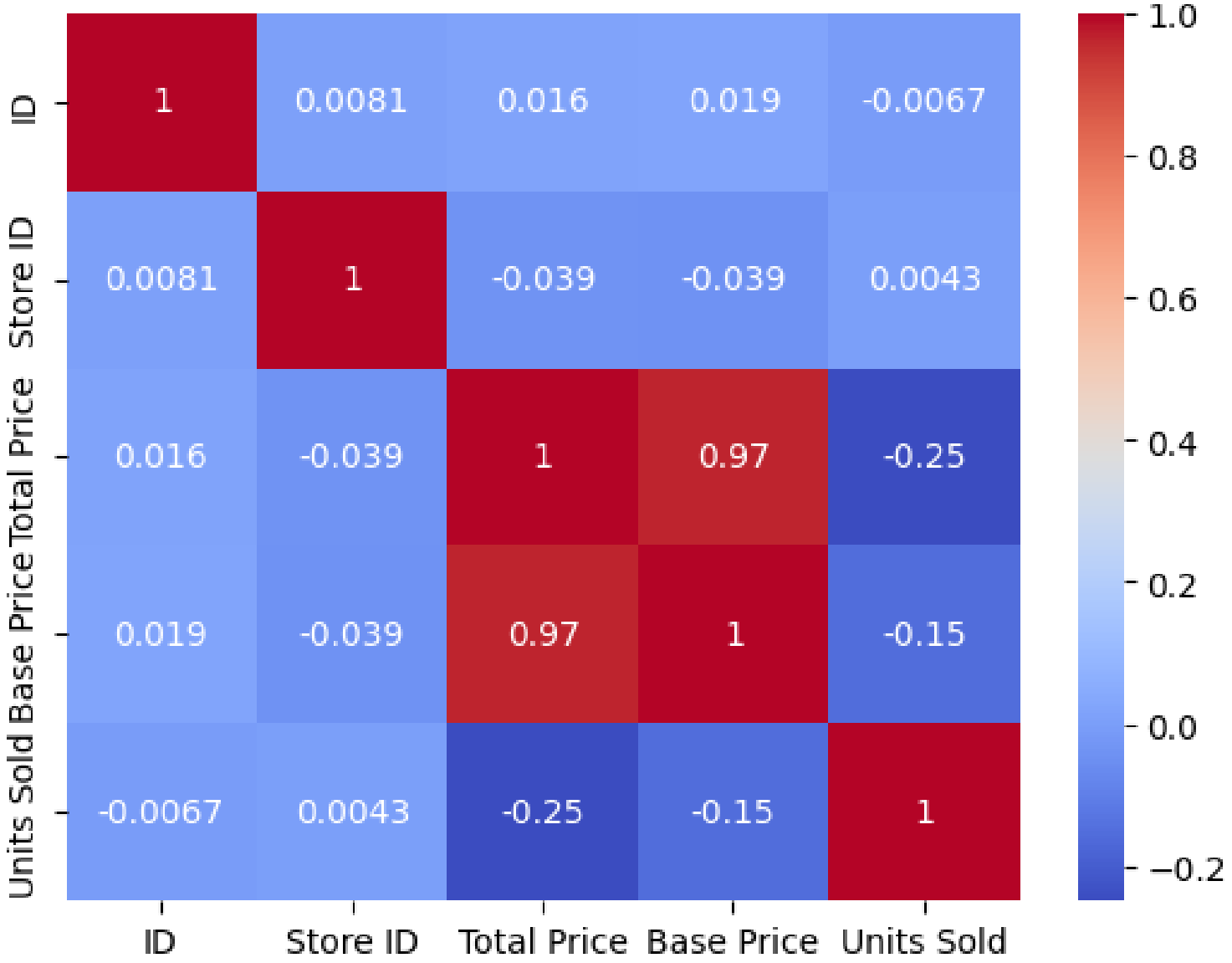
# 2.3 Dataset Columns

- Product ID
- Store ID
- Total Price (after discount)
- Base Price (original price)
- Units Sold (Quantity Demanded)
- **Training Data:**
  - **Independent Variables (Features):**
    - Total Price: The total price of a product.
    - Base Price: The base price of the product.
    - Store ID.
  - **Dependent Variable (Target):**
    - Units Sold: The number of units of the product sold.

# Dataset columns



# Data correlation



- **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.
- **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.
- **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.

## 2.4 Modules/ Libraries used:

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

# Models used

- **DecisionTreeRegressor**: A scikit-learn regressor implementing a decision tree for regression tasks, predicting target values based on input features.

- **GradientBoostingRegressor**: A scikit-learn regressor employing gradient boosting for ensemble learning, combining weak learners to improve predictive performance.

- **RandomForestRegressor**: A scikit-learn regressor utilizing a random forest, an ensemble of decision trees, to enhance accuracy and control overfitting in regression tasks.

# 2.5 Train and Test Splits

- **train_test_split function:**

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.

```
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)
```

# Gradient Boost Regressor Tree Model

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

```
# Create and train a linear regression model
# Create and train a Gradient Boosting Regressor model (change RandomForestRegressor to GradientBoostingRegressor)
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.

```
y_pred = model.predict(X_test)
# Calculate the Mean Squared Error (MSE) to evaluate the model
print('R2:',metrics.r2_score(y_test,y_pred))
print('MAE:', metrics.mean_absolute_error(y_test, y_pred))
print('MSE:', metrics.mean_squared_error(y_test, y_pred))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

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

# Decision Tree Regressor Model

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

```
# Create and train a linear regression model
model = DecisionTreeRegressor(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.

```
[8]
    y_pred = model.predict(X_test)
    # Calculate the Mean Squared Error (MSE) to evaluate the model
    print('R2:',metrics.r2_score(y_test,y_pred))
    print('MAE:', metrics.mean_absolute_error(y_test, y_pred))
    print('MSE:', metrics.mean_squared_error(y_test, y_pred))
    print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

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

# RandomForest Model

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

```
# Create and train a linear regression model
model = RandomForestRegressor(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.

```
y_pred = model.predict(X_test)
# Calculate the Mean Squared Error (MSE) to evaluate the model
print('R2:',metrics.r2_score(y_test,y_pred))
print('MAE:', metrics.mean_absolute_error(y_test, y_pred))
print('MSE:', metrics.mean_squared_error(y_test, y_pred))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

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

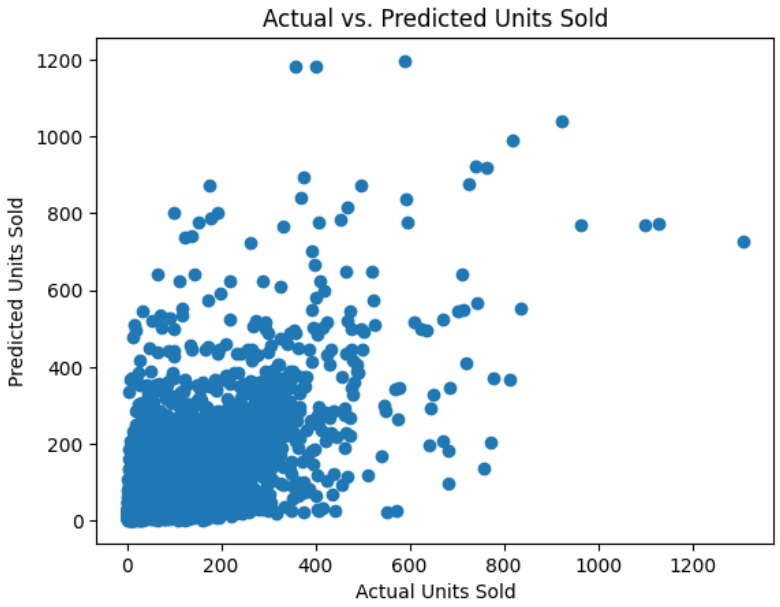
# Why does the Linear Regression model fail?

- 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:
  2. Weak Correlation with the Dependent Variable
  3. Overfitting

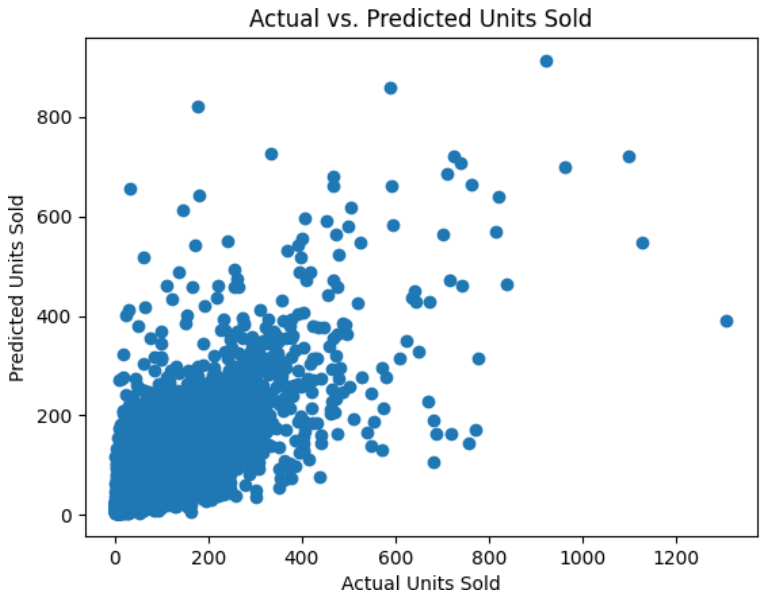
```
R2: 0.1490719889302594
MAE: 32.49815502614977
MSE: 2784.619420940248
RMSE: 52.76949327916886
```

Linear Regression Metrics

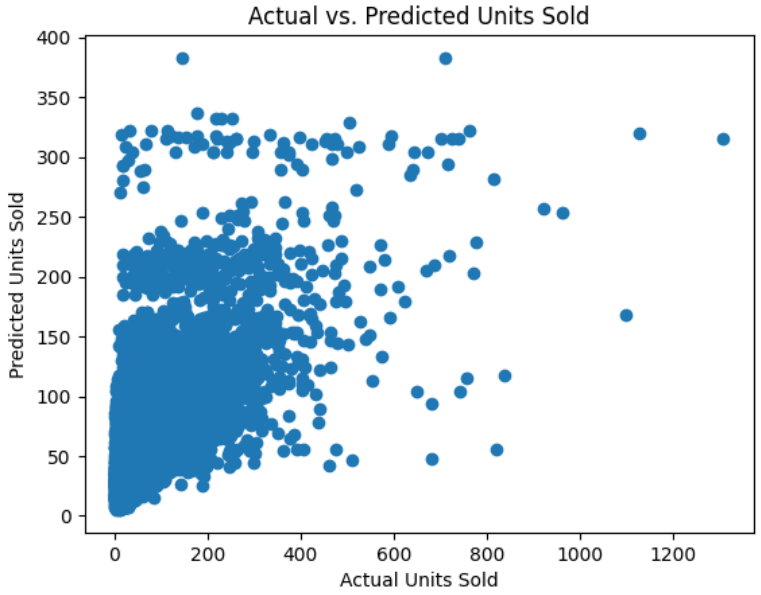
# Actual vs prediction




Decision Tree Regressor




Random Forest Regressor




Gradient Boosting Regressor

# 2.7 Metrics:

- R-Squared($R^2$)
- Mean Absolute Error (MAE)
- Mean Squared Error (MSE)
- Root Mean Squared Error (RMSE)

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

# 2.7 Metrics:

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

**Decision Tree Regressor**

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

**Gradient Boosting Regressor**

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

**Random Forest Regressor**