MARMARA UNIVERSITY
FACULTY OF ENGINEERING

CSE4074.1 - Machine Learning

Term Project – Model Evaluation and Optimization Report

Due Date: 23.12.2024

	Name Surname	Student Id
1	Ahmet Arda Nalbant	150121004
2	Umut Bayar	150120043
3	Niyazi Ozan Ateş	150121991
4	Murat Albayrak	150120025
5	Kadir Pekdemir	150121069

1. Linear Regression

```
## begin of Model Evaluation and Optimization
## Linear Regression Model Evaluation and Optimization
y_pred = model_lr.predict(x_test)

# R² Skoru
r2_score = model_lr.score(x_test, y_test)

# RMSE
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)  # Root Mean Squared Error
# MAPE
mape = (abs((y_test - y_pred) / y_test)).mean() * 100

# Results
print(f"R² Score: {r2_score}")
print(f"RMSE: {rmse}")
print(f"MAPE: {mape}%")
```

```
R² Score: 0.971924824046698
RMSE: 1633.3341547308298
MAPE: sellingprice    34.431414
dtype: float64%
```

1.1 Prediction:

- `y_pred = model_lr.predict(x_test)`:
  - The trained Linear Regression model (`model_lr`) is used to make predictions on the test set (`x_test`), which gives the predicted values (`y_pred`) for the target variable.

1.2 R^2 Score:

- `r2_score = model_lr.score(x_test, y_test)`:
  - The **R^2 score** (Coefficient of Determination) measures the proportion of variance in the target variable (`y_test`) that is explained by the model.
  - The value ranges from 0 to 1, with a value of 1 indicating perfect predictions and 0 indicating no predictive power.

1.3 Root Mean Squared Error (RMSE):

- `mse = mean_squared_error(y_test, y_pred)`:
  - The **Mean Squared Error (MSE)** measures the average of the squared differences between the predicted and actual values.
- `rmse = np.sqrt(mse)`:

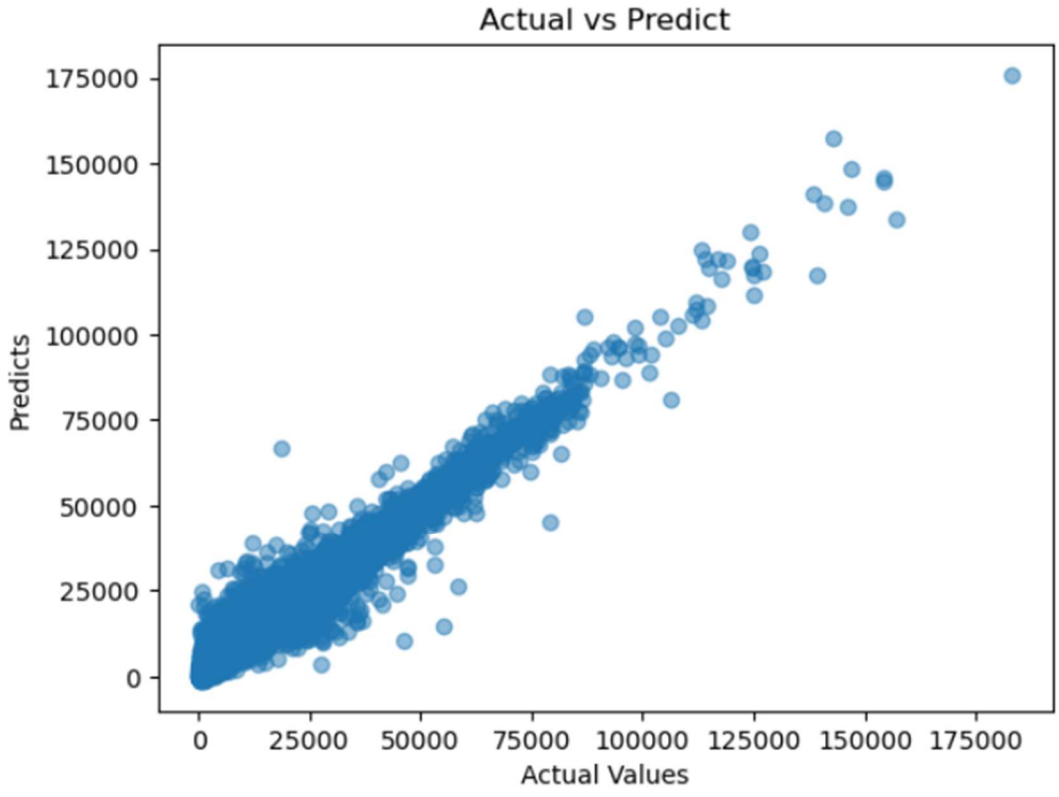
- The **Root Mean Squared Error (RMSE)** is the square root of the MSE, bringing the error metric back to the original scale of the target variable (in this case, the selling price).
- RMSE provides an idea of the magnitude of the error and is sensitive to large errors.

1.4 Mean Absolute Percentage Error (MAPE):

- `mape = (abs((y_test - y_pred) / y_test)).mean() * 100`:
  - **MAPE** measures the accuracy of the model by calculating the average of the absolute percentage differences between the actual and predicted values.
  - A lower MAPE indicates a better model. It is particularly useful when comparing models across different datasets or when interested in the relative error in percentage terms.

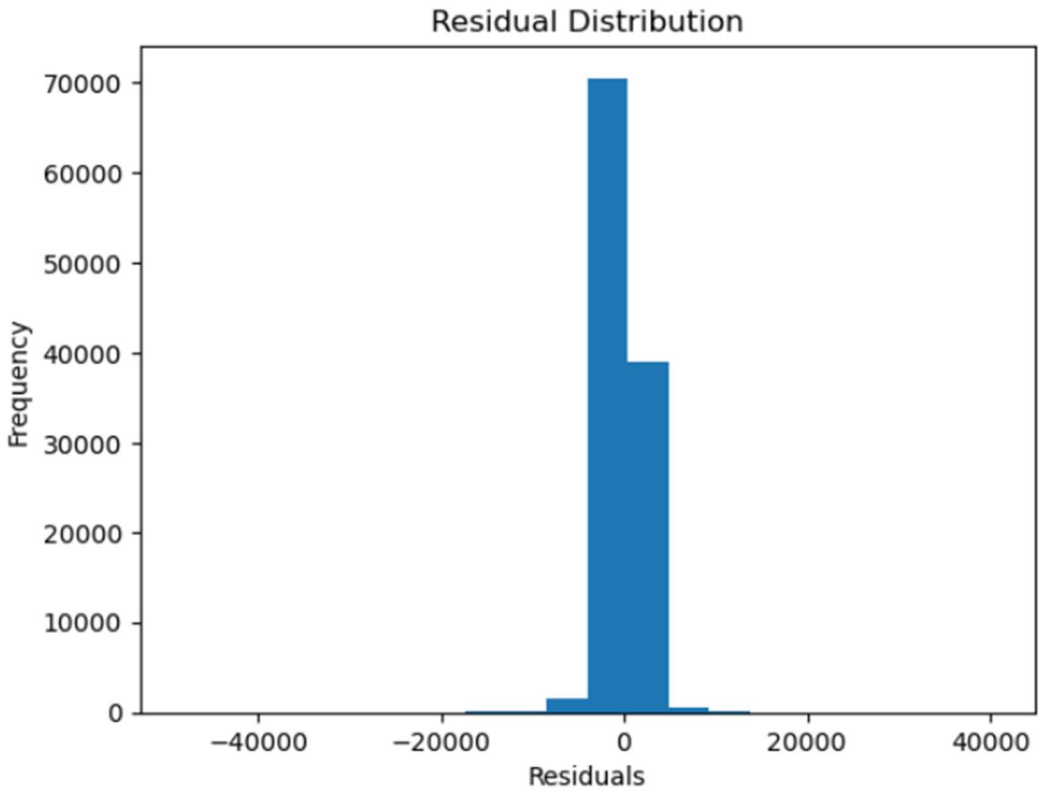
1.5 Actual vs Predict graphic

```
## difference between real and predict values

plt.scatter(y_test, y_pred, alpha=0.5)
plt.xlabel("Actual Values")
plt.ylabel("Predicts")
plt.title("Actual vs Predict")
plt.show()
```



1.6 Residual Distribution Graphic

```
## Outlier points

residuals = y_test - y_pred
plt.hist(residuals, bins=20)
plt.xlabel("Residuals")
plt.ylabel("Frequency")
plt.title("Residual Distribution")
plt.show()
```



1.7 Results:

- The most suitable model for our data set was the linear regression model. We aim to calculate a selling price due to our data set, and this calculation mainly comes from numerical data. The best model that allows us to make predictions by calculating the effect of this data on the price was the linear regression model.

2. Ridge Regression

```
## Ridge Regression Model Evaluation and Optimization
y_pred = ridge_model.predict(x_test)

# R² score
r2_score = ridge_model.score(x_test, y_test)

# RMSE
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)  # Root Mean Squared Error
# MAPE
mape = mean_absolute_percentage_error(y_test, y_pred) * 100

# Results
print(f"R² Score: {r2_score}")
print(f"RMSE: {rmse}")
print(f"MAPE: {mape}%")
```

```
R² Score: 0.9719248240350279
RMSE: 1633.3341550702978
MAPE: 34.43141347454782%
```

When comparing the **Ridge Regression** model evaluation results to the **Linear Regression** model results, two models typically differ in terms of the metrics:

2.1 R^2 Score:

- **Linear Regression**: Tends to have a higher R^2 score, as it directly tries to fit the best possible line to the data without regularization.
- **Ridge Regression**: Might have a slightly lower R^2 score, especially if the data contains multicollinearity. Ridge Regression introduces regularization to reduce model complexity, which can lower the R^2 score but help generalize better.

2.2 RMSE (Root Mean Squared Error):

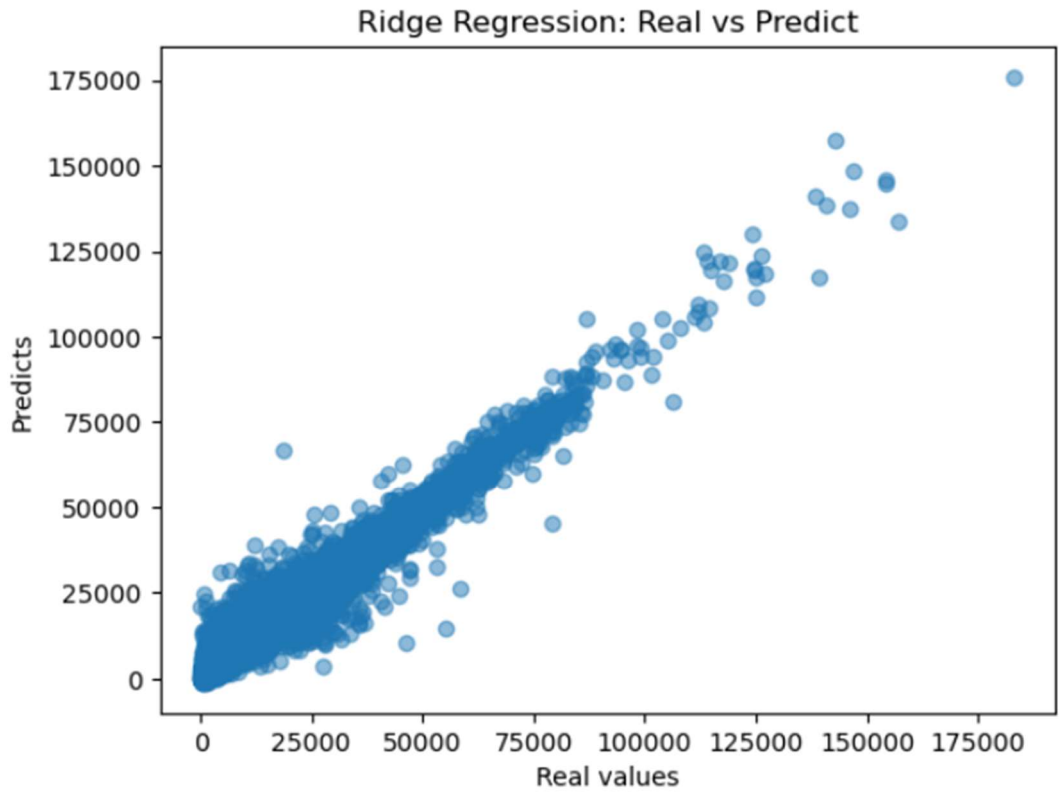
- **Linear Regression**: The RMSE might be slightly lower in Linear Regression if the model overfits to the training data. This would mean it performs very well on the training set but might perform poorly on unseen data.
- **Ridge Regression**: The RMSE could be higher than Linear Regression, as Ridge adds a penalty for large coefficients. This may increase the error slightly, but Ridge's regularization usually helps it generalize better to unseen data.

2.3 MAPE (Mean Absolute Percentage Error):

- **Linear Regression**: The MAPE could be lower if the Linear Regression model fits the data perfectly (e.g., no overfitting). However, it may be prone to higher errors if the model is overfitted or suffers from multicollinearity.
- **Ridge Regression**: The MAPE might be a bit higher than Linear Regression because of the regularization, but Ridge often performs better on new data, reducing overall prediction errors in real-world scenarios.

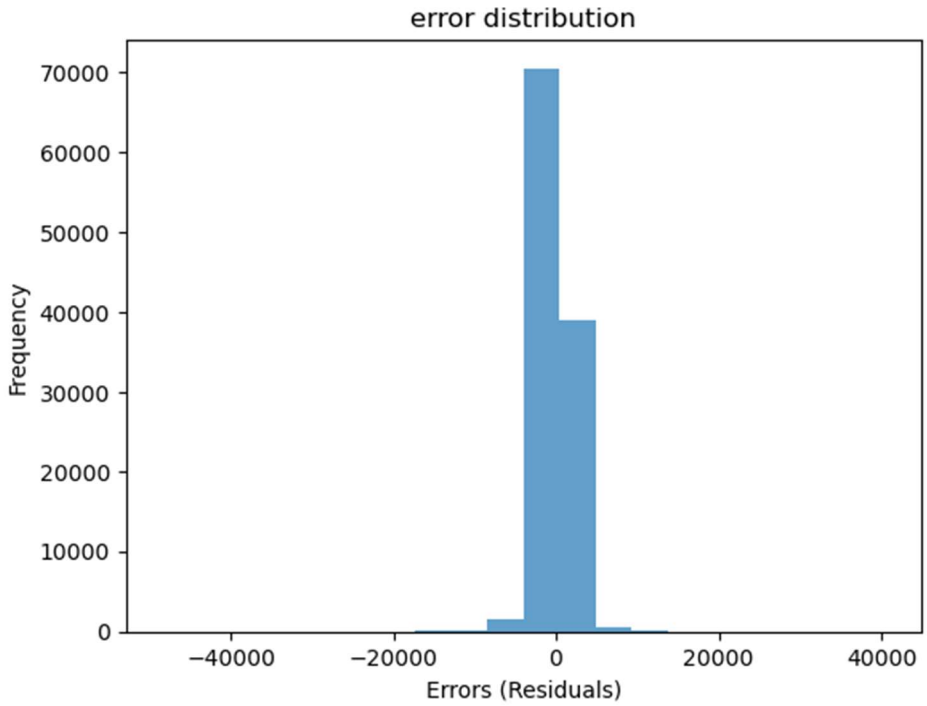
2.4 real vs predict graphics

```
plt.scatter(y_test, y_pred, alpha=0.5)
plt.xlabel("Real values")
plt.ylabel("Predicts")
plt.title("Ridge Regression: Real vs Predict")
plt.show()
```



2.5 Error Distribution Graphics

```python
#convert y_test values to numpy array
y_test = y_test.values

# y_test to 1 dimension
y_test = y_test.ravel()

# Residuals calculation
residuals = y_test - y_pred

# Draw Histogram graphic
import matplotlib.pyplot as plt
plt.hist(residuals, bins=20, alpha=0.7)
plt.xlabel("Errors (Residuals)")
plt.ylabel("Frequency")
plt.title("error distribution")
plt.show()
```



3. Logistic Regression

```
## Logistic Regression Model Evaluation and Optimization
# Accuracy
print(f"Accuracy: {accuracy}")

# Precision, Recall, F1-Score
print("\nClassification Report:")
print(classification_report(y_test, predictions, target_names=['Low Price', 'Medium Price', 'High Price']))

# Confusion Matrix
cm = confusion_matrix(y_test, predictions)
print("\nConfusion Matrix:")
print(cm)

# Confusion Matrix visuluzation
import seaborn as sns
import matplotlib.pyplot as plt

sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=['Low Price', 'Medium Price', 'High Price'], yticklabe
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.title("Confusion Matrix")
plt.show()
```

```
Accuracy: 0.9999821057905661

Classification Report:
              precision    recall  f1-score   support

   Low Price       0.75      0.75      0.75         4
Medium Price       1.00      1.00      1.00    111060
  High Price       1.00      1.00      1.00       704

    accuracy                           1.00    111768
   macro avg       0.92      0.92      0.92    111768
weighted avg       1.00      1.00      1.00    111768


Confusion Matrix:
[[     3      0      1]
 [     0 111060      0]
 [     1      0    703]]
```

The classification metrics provided appear to be from a **classification model**, such as **Logistic Regression** or **Random Forest Classifier**, as indicated by the use of **accuracy**, **precision**, **recall**, and the **confusion matrix**. These metrics are typically used for evaluating classification models, not regression models like **Linear Regression** and **Ridge Regression**.

3.1 Accuracy:

- The **classification model** shows an impressive accuracy of **99.998%**.
- Both **Linear Regression** and **Ridge Regression** don't directly provide "accuracy" as a metric. Instead, they provide **R^2 score**, **RMSE**, and **MAPE**.
- **Linear Regression** and **Ridge Regression** are designed for **regression tasks**, predicting continuous values like price, whereas accuracy is more relevant for **classification tasks**.

3.2 Precision, Recall, F1-Score (Classification Metrics):

- These metrics are irrelevant for **Linear Regression** and **Ridge Regression**, as they apply to classification problems. Precision and recall measure how well the model classifies data into categories, whereas **R^2** (from Linear and Ridge) measures how well the model fits continuous data.

3.3 Confusion Matrix:

- This is another classification-specific metric that **Linear Regression** and **Ridge Regression** don't produce. The confusion matrix tells you how well the model predicts each class label, but for regression models, you'd use residual analysis to assess prediction errors (e.g., difference between predicted and actual prices).

Why This Classification Model Should Not Be Preferred for Price Prediction:

3.4 Task Mismatch:

- **Linear Regression** and **Ridge Regression** are regression models, meaning they are designed for predicting **continuous numerical values** (like selling prices). The classification model provided categorizes the selling price into distinct bins (e.g., Low, Medium, High Price).
- **Using classification for price prediction** is inappropriate because it doesn't provide an exact price prediction; rather, it puts the data into predefined categories, leading to loss of information.

3.5 Loss of Precision:

- The **classification model** is overly simplistic because it groups selling prices into arbitrary ranges (Low, Medium, High). In real-world scenarios, you often need the exact value of the selling price, not just its category.
- **Linear and Ridge Regression** give you a **precise numerical prediction**, which is critical for tasks like pricing, forecasting, or economic analysis where accuracy in value is essential.

4.Random Forest Classification

[268]:
```
## Random Forest Classifier Model Evaluation and Optimization
# Accuracy
print(f"Accuracy: {accuracy}")

# Precision, Recall, F1-Score
print("\nClassification Report:")
print(classification_report(y_test, predictions, target_names=['Low Price', 'Medium Price', 'High Price']))

# Confusion Matrix
cm = confusion_matrix(y_test, predictions)
print("\nConfusion Matrix:")
print(cm)

# Feature Importance
import matplotlib.pyplot as plt
import pandas as pd

feature_importances = pd.Series(rf_model.feature_importances_, index=X.columns)
feature_importances = feature_importances.sort_values(ascending=False)

plt.figure(figsize=(10, 6))
feature_importances.plot(kind='bar', color='teal')
plt.title("Feature Importances")
plt.ylabel("Importance Score")
plt.xlabel("Features")
plt.show()
```

```
Accuracy: 0.9999821057905661

Classification Report:
              precision    recall  f1-score   support

   Low Price       0.75      0.75      0.75         4
Medium Price       1.00      1.00      1.00    111060
  High Price       1.00      1.00      1.00       704

    accuracy                           1.00    111768
   macro avg       0.92      0.92      0.92    111768
weighted avg       1.00      1.00      1.00    111768


Confusion Matrix:
[[     3      0      1]
 [     0 111060      0]
 [     1      0    703]]
```

4.1 Random Forest Classifier Results

From the Random Forest classifier:

- **Accuracy:** Extremely high (0.99998), but this can be misleading due to class imbalance.
- **Classification Report:**
  - Although the model performs well overall, the F1-score for the "Low Price" class is only 0.75, indicating poor performance in this category.
  - The dataset is dominated by the "Medium Price" class, meaning the model might be overfitting to this class and neglecting others.
- **Confusion Matrix:**

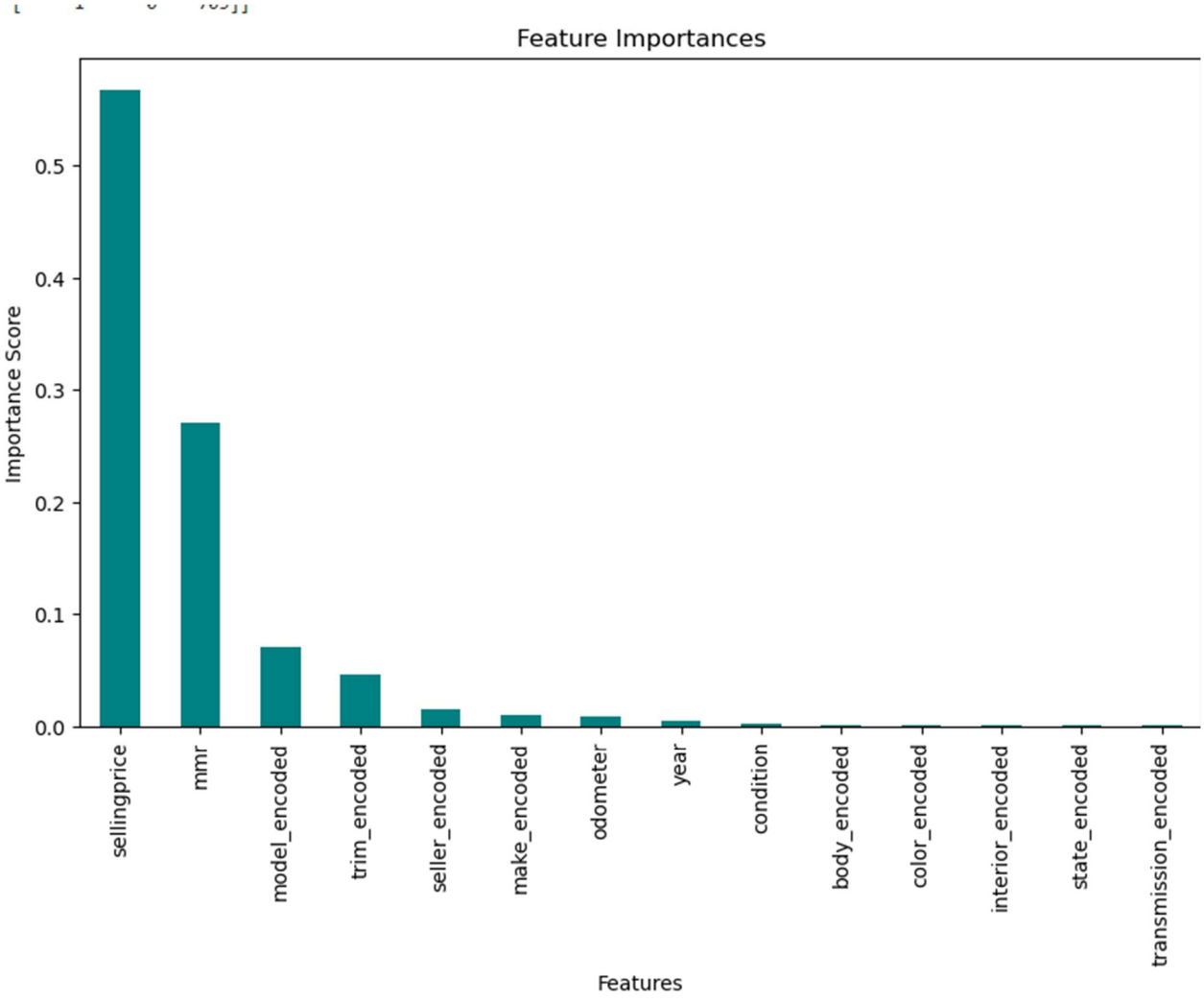
- Very few examples exist for "Low Price" and "High Price" classes. This shows the model struggles with imbalanced class distributions.

These results suggest that the dataset may not be ideal for classification. Converting continuous price data into discrete classes likely resulted in information loss. For instance, subtle differences in prices are completely ignored when they are grouped into broad categories.

4.2 Why Regression is Better than Classification for This Dataset

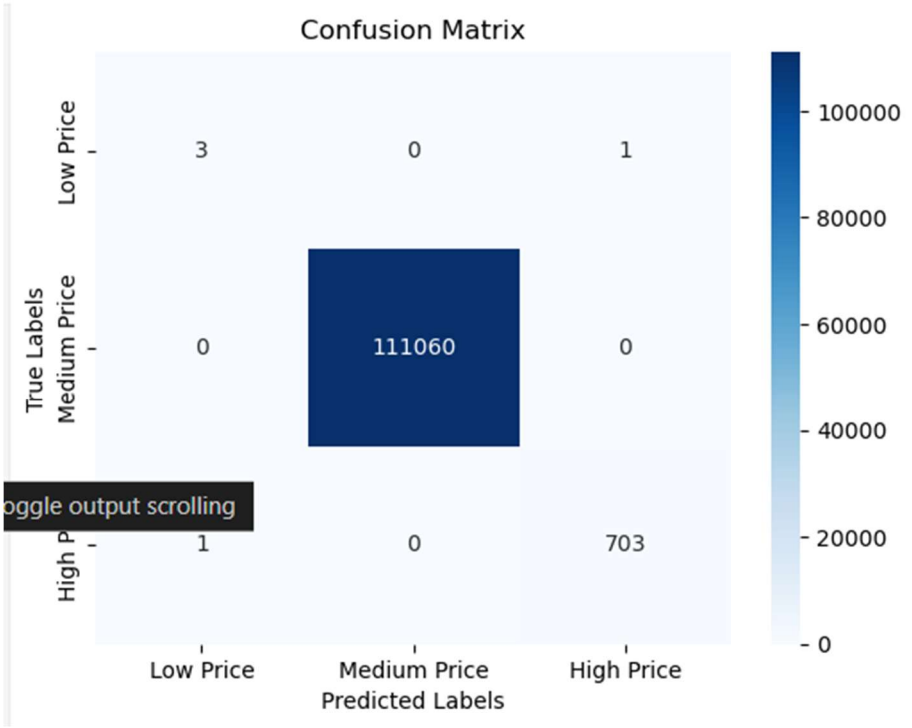
1. **Nature of the Target Variable:** Price is inherently a continuous variable. Treating it as a categorical variable (as done in classification) simplifies the problem but loses critical granularity and introduces unnecessary challenges.
2. **Class Imbalance in Classification:** The dataset has an imbalance in class distribution, with the "Medium Price" class dominating. This biases the classifier and reduces its effectiveness for underrepresented classes.
3. **More Informative Metrics:** Regression provides metrics like R^2, RMSE, and MAPE, which are directly interpretable in the context of continuous price prediction. These metrics offer a clearer understanding of prediction accuracy compared to classification metrics.
4. **Preservation of Information:** In regression, no information is lost, and the model can capture subtle variations in price, which is crucial for understanding and predicting realistic price values.

4.3 Feature Importances Graphic



4.4 Confusion Matrix Graphic

```
##display confustion matrix

sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=['Low Price', 'Medium Price', 'High Price'], ytic
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.title("Confusion Matrix")
plt.show()
```



5. k-means Cluster

```
## davies bouldin score calculation for k means

db_score = davies_bouldin_score(X_scaled, kmeans.labels_)
print(f"Davies-Bouldin Score: {db_score:.3f}")
```

```
Davies-Bouldin Score: 1.950
```

5.1 The **Davies-Bouldin Score (DB Score)** is a clustering evaluation metric that measures the quality of clusters based on:

1. **Intra-cluster similarity:** How close data points are to their own cluster center.
2. **Inter-cluster separation:** How far apart the clusters are from each other

DB Score: 1.950

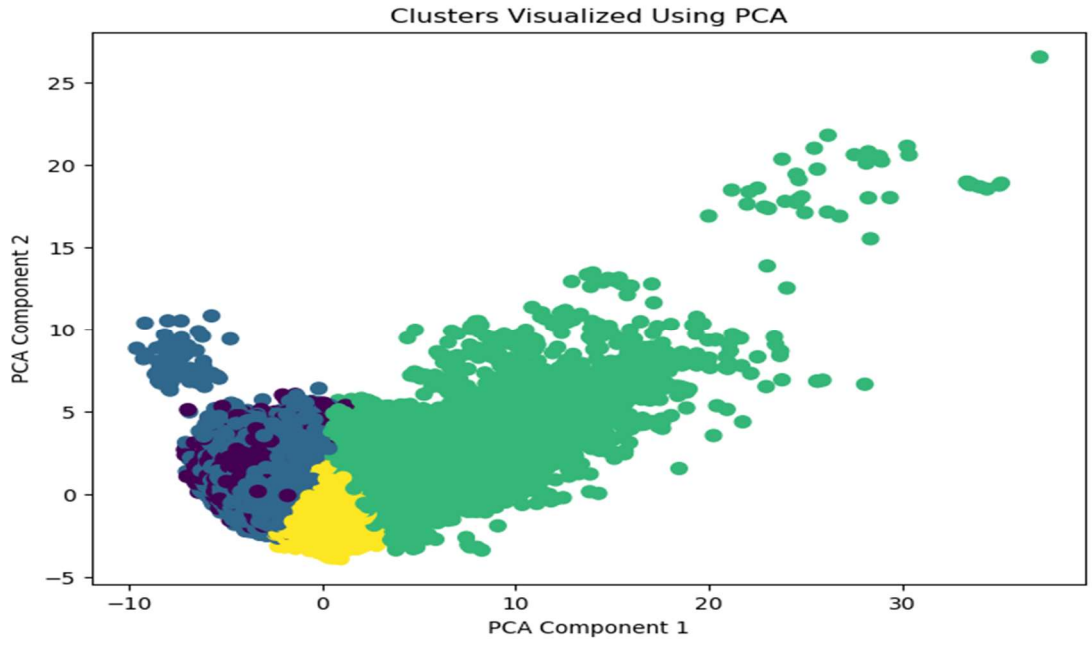
- A DB score of 1.950 suggests moderate clustering quality. While the clusters are somewhat distinct, there may be overlap or suboptimal separation.
- For high-quality clustering, scores closer to 0 are desired. Values around 2 or higher indicate that the clusters are less cohesive and not well-separated.

5.2 Why Regression is Better for Our Dataset Compared to Clustering

1. **Nature of the Target Variable:**
   - Your dataset involves a **continuous target variable (price)**. Clustering algorithms like K-Means do not consider a target variable and instead group data based on similarity in input features. This ignores the relationship between the features and the actual price.
   - Regression models explicitly aim to predict the target variable, making them more suitable for this dataset.
2. **Limitations of Clustering for Price Prediction:**
   - Clustering doesn't provide precise price predictions. It can only group data into clusters that represent broad price ranges (e.g., low, medium, high).
   - As seen in the DB score (1.950), the clustering quality is moderate. This means the clusters are not distinct enough to represent meaningful price categories effectively.
3. **Regression Provides Granular Insights:**
   - Regression models, such as Linear Regression or Random Forest Regression, predict exact numerical values for the price, preserving the granularity of the data.
   - Metrics like R^2, RMSE, and MAPE in regression directly measure the accuracy of predictions, whereas clustering metrics (e.g., DB score) only evaluate the quality of grouping without a direct link to prediction accuracy.

5.3 ClusterVisualized Using PCA

```
## Visuluzation using PCA
from sklearn.decomposition import PCA

pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

plt.figure(figsize=(8, 6))
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=df_cluster['Cluster'], cmap='viridis', s=50)
plt.title("Clusters Visualized Using PCA")
plt.xlabel("PCA Component 1")
plt.ylabel("PCA Component 2")
plt.show()
```



5.4 Cluster Distribution Graphic

```
## Cluster Distribution graphic

cluster_counts = df_cluster['Cluster'].value_counts()

plt.figure(figsize=(8, 6))
cluster_counts.plot(kind='bar', color='teal')
plt.title("Cluster Distribution")
plt.xlabel("Cluster")
plt.ylabel("Number of Samples")
plt.show()
```