VIRGINIA TECH

COMPREHENSIVE ANALYSIS OF CAR PRICES DATASET

FINAL TERM PROJECT

MACHINE LEARNING – I

DR. REZA JAFARI

SUBMITTED BY

DIVAKARA RAO ANNEPU

APRIL 30, 2024

VIRGINIA TECH

Table of Contents

Table of Figures

Abstract

This project undertakes a detailed exploration and analysis of a large dataset containing over 558,000 entries related to car sales, encompassing features such as make, model, year, odometer readings, condition, and selling prices. The objective of this study is to derive meaningful insights into factors influencing car prices and to develop predictive models that can accurately forecast car selling prices based on various attributes. The analysis involved extensive data preprocessing, including handling missing values, encoding categorical variables, and normalizing numerical data. Subsequent exploratory data analysis provided a foundational understanding of the distributions and relationships within the data, setting the stage for more complex analytical techniques.

The project progressed through several analytical phases, each addressing different aspects of data science and machine learning. In Phase I, exploratory data analysis and feature engineering were employed to prepare the dataset for modeling. Phase II focused on regression analysis, utilizing multiple linear regression to model the relationship between car features and their selling prices, achieving a high R-squared value indicative of a good model fit. Phase III explored various classification models to predict categorical outcomes, such as car condition, comparing the performance of models like Decision Trees, SVM, and Logistic Regression. The final phase, cluster analysis, applied K-means and DBSCAN to uncover natural groupings and patterns in the data. Each phase was supported by rigorous statistical testing and validation methods to ensure the robustness and accuracy of the findings. Recommendations were made based on the analysis results, emphasizing improvements in data collection, model tuning, and feature selection to enhance future predictions and analyses.

1 Introduction

The automotive industry stands as one of the most data-intensive sectors in the global economy, where every transaction generates a wealth of information that can be harnessed to predict market trends, optimize pricing strategies, and enhance consumer satisfaction. The advent of data science and machine learning offers profound opportunities to exploit this data, providing insights that can lead to more informed decisions. This project focuses on analyzing a substantial dataset of car sales, which includes over 558,000 records with 16 distinct attributes ranging from basic car details to sales information. The aim is to extract actionable insights from the dataset and develop predictive models to estimate car selling prices effectively.

The methodology of this project is structured into several distinct phases, each designed to address specific research questions and achieve the overarching goal of understanding the determinants of car pricing in a detailed manner. Initially, the project begins with comprehensive data cleaning and preprocessing to handle missing data, encode categorical variables, and normalize distributions, ensuring the dataset is primed for robust analysis. Following this, exploratory data analysis (EDA) is conducted to identify trends, detect outliers, and understand the relationships between various car attributes and their selling prices. This foundational analysis sets the stage for more advanced statistical modeling.

In the subsequent phases, the project employs a variety of machine learning techniques. Phase II involves building and fine-tuning multiple linear regression models to predict car selling prices, assessing the model's performance through statistical metrics such as R-squared values and mean squared error (MSE). Phase III shifts focus to classification techniques, where models such as Decision Trees, Support Vector Machines (SVM), and Logistic Regression are developed to classify cars based on their condition. The performance of these models is meticulously compared using metrics like accuracy, precision, recall, and the Area Under the ROC Curve (AUC). The final analytical phase, cluster analysis, applies unsupervised learning methods to discover inherent groupings within the data, providing insights into market segmentation.

By integrating these diverse analytical techniques, the project not only enhances the understanding of factors that influence car prices but also demonstrates the application of data science methodologies in real-world industry settings. The insights gained from this study are intended to guide automotive companies in refining their pricing strategies and improving customer targeting. Through detailed data analysis and modeling, this project contributes to the evolving field of automotive data science, paving the way for further research and application in predictive analytics.

2 Description of the Dataset

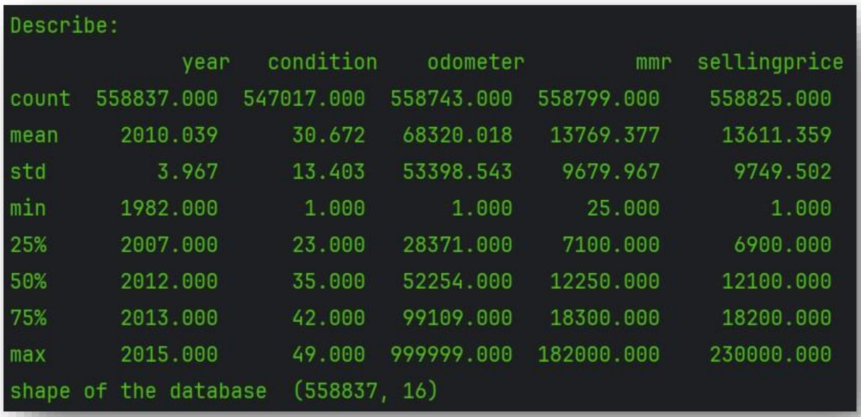
The dataset under analysis comprises an extensive collection of over 558,837 car sales transactions, each detailed with 16 different attributes. This rich dataset provides a granular view of the various factors that influence car prices in the market. Each record in the dataset represents a unique car sale and includes both descriptive and quantitative information about the vehicle and the conditions of its sale.

2.1 Attributes Overview

- **Year:** The manufacture year of the car, crucial for pricing due to the age and technology of the vehicle.

- **Make:** The brand or manufacturer, such as Ford, Toyota, or BMW, which influences the car’s market perception and price.
- **Model:** Specific model of the car affecting prices due to varying features, performance specifications, and market demand.
- **Trim:** Specific trim level of the car model, indicating variations in luxury and functional features that impact pricing.
- **Body:** The body style of the car, e.g., sedan, SUV, or hatchback, influencing consumer preference and thus pricing.
- **Transmission:** Type of transmission system (automatic, manual), a significant factor in driving experience and price.
- **VIN:** Vehicle Identification Number, a unique code that identifies individual vehicles.
- **State:** The U.S. state where the car was sold, influencing prices due to regional demand variations and economic conditions.
- **Condition:** A numeric rating of the car’s overall physical and operational condition at the time of sale, influencing its market value.
- **Odometer:** Mileage on the car, where higher mileage usually decreases a vehicle’s price due to wear and tear.
- **Color:** Exterior color of the car, affecting consumer preference and resale value.
- **Interior:** Color and type of the car’s interior, contributing to aesthetic appeal and comfort.
- **Seller:** The entity or individual who sold the car, such as a dealer, an auction house, or a private seller.
- **MMR:** The Manheim Market Report value, an industry-standard wholesale price, serving as a benchmark for pricing used cars.
- **Selling Price:** The actual sale price of the car, the dependent variable in price prediction models.

- **Sale Date:** The date on which the transaction occurred, providing temporal context for the sale.



```
Describe:
              year   condition    odometer         mmr  sellingprice
count  558837.000  547017.000  558743.000  558799.000    558825.000
mean     2010.039      30.672   68320.018   13769.377     13611.359
std         3.967      13.403   53398.543    9679.967      9749.502
min      1982.000       1.000       1.000      25.000         1.000
25%      2007.000      23.000   28371.000    7100.000      6900.000
50%      2012.000      35.000   52254.000   12250.000     12100.000
75%      2013.000      42.000   99109.000   18300.000     18200.000
max      2015.000      49.000  999999.000  182000.000    230000.000
shape of the database  (558837, 16)
```

Figure 1

2.2 Data Utilization

The dataset serves multiple analytical purposes:

- **Price Prediction:** Utilizing attributes like year, make, model, and odometer readings to estimate the selling price.

- **Condition Assessment:** Classification models can predict the condition category of a vehicle based on its characteristics and historical data.

- **Market Analysis:** Analysis of sales trends over time and across different regions to inform market strategies for automotive firms.

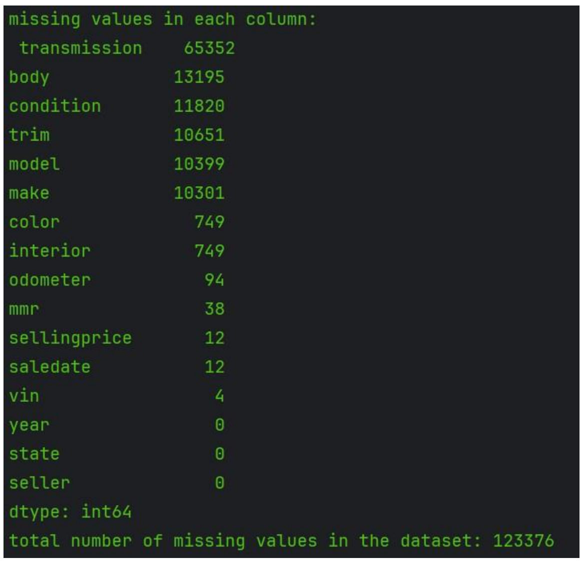
2.3 Importance in Industry

In the automotive industry, a deep understanding of the factors influencing car prices and sales dynamics is crucial for inventory management, pricing strategies, and marketing. The dataset provides a comprehensive basis for such analyses, offering actionable insights applicable to car manufacturers, dealerships, and secondary market analysts.

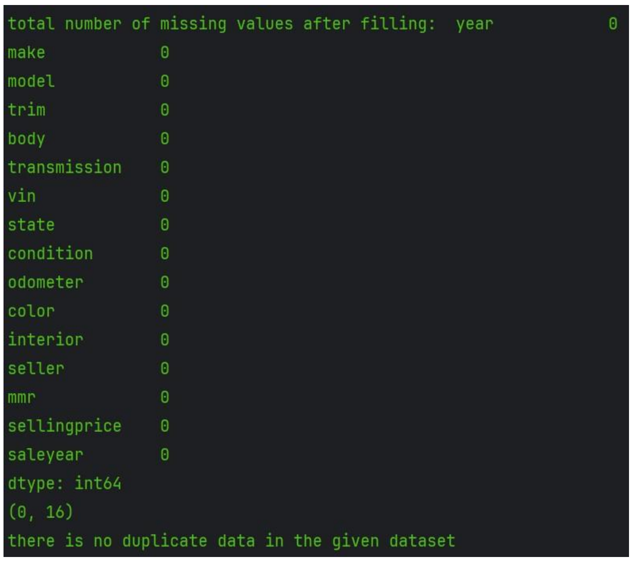
3 Phase I: Feature Engineering & Exploratory Data Analysis

3.1 Data Cleaning

Data cleaning is a critical initial step in any data analysis project. For this dataset, addressing missing values is essential to ensure the integrity of subsequent analyses. Various methods such as imputation using the mean or median for numerical data, or mode for categorical data, can be applied. This dataset also required a check for duplicate entries, which were removed to prevent any bias or incorrect outcomes in the analysis.



```
missing values in each column:
 transmission    65352
body            13195
condition       11820
trim            10651
model           10399
make            10301
color             749
interior          749
odometer           94
mmr                38
sellingprice       12
saledate           12
vin                 4
year                0
state               0
seller              0
dtype: int64
total number of missing values in the dataset: 123376
```

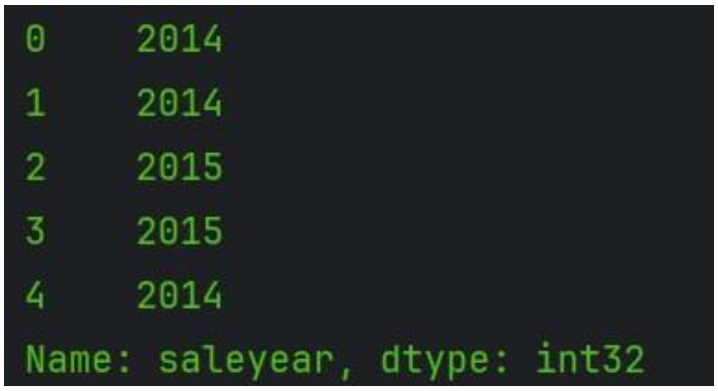


```
total number of missing values after filling:  year            0
make            0
model           0
trim            0
body            0
transmission    0
vin             0
state           0
condition       0
odometer        0
color           0
interior        0
seller          0
mmr             0
sellingprice    0
saleyear        0
dtype: int64
(0, 16)
there is no duplicate data in the given dataset
```

Figure 2: Missing Values before and after filling.

3.2 Aggregation

Aggregation was performed where applicable to summarize data features, enhancing the clarity and efficiency of the analysis. In the data preprocessing phase, it is often necessary to simplify or modify the data to enhance analytical accuracy and efficiency. One common transformation is extracting specific components from date fields. For our dataset, extracting the year from the 'saledate' column allows us to perform temporal analyses, such as year-on-year trends or seasonal patterns, more efficiently.



```
0    2014
1    2014
2    2015
3    2015
4    2014
Name: saleyear, dtype: int32
```

Figure 3: New extracted feature 'saleyear'

3.3 Down Sampling

Down sampling was considered to balance the dataset, particularly useful in scenarios where one class significantly outweighs another in a classification context. This ensures that the model trained on the data does not become biased towards the majority class. Here we have down sampled the data using the random sampling technique.

3.4 Dimensionality Reduction/Feature Selection

Dimensionality reduction and feature selection were conducted using several methods:

3.4.1 Random Forest Analysis

helped in identifying the most impactful features based on their importance scores.

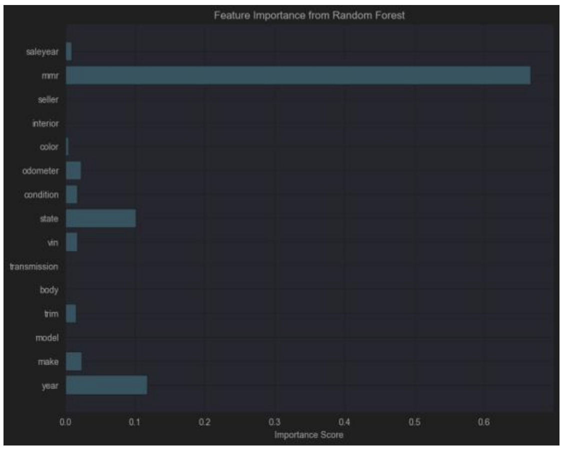


Figure 4: Feature importance using Random Forest.

3.4.2 Principal Component Analysis (PCA)

and condition number analysis were used to reduce the dimensionality while retaining the variance in the data.

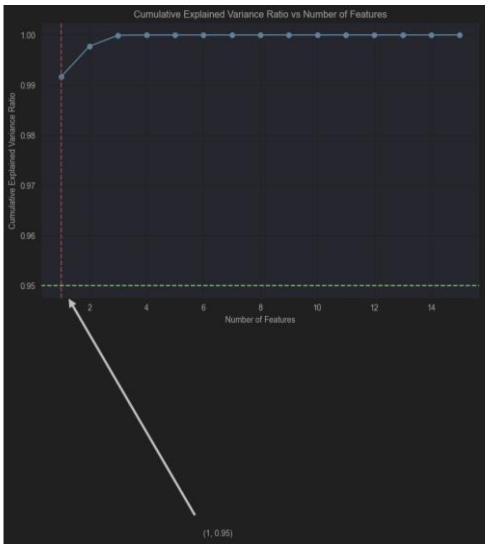
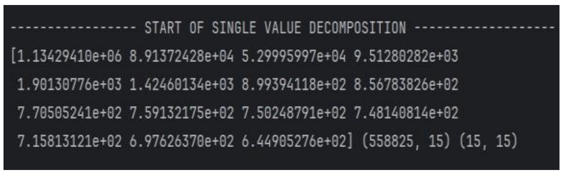


Figure 5: Principal Component Analysis.

3.4.3 Singular Value Decomposition (SVD)

offered insights into the data structure by decomposing it into singular vectors and singular values.



```
---------------- START OF SINGLE VALUE DECOMPOSITION ------------------
[1.13429410e+06 8.91372428e+04 5.29995997e+04 9.51280282e+03
 1.90130776e+03 1.42460134e+03 8.99394118e+02 8.56783826e+02
 7.70505241e+02 7.59132175e+02 7.50248791e+02 7.48140814e+02
 7.15813121e+02 6.97626370e+02 6.44905276e+02] (558825, 15) (15, 15)
```

Figure 6: Single Value Decomposition Analysis

3.4.4 Variance Inflation Factor (VIF)

was calculated to check for multicollinearity among features, ensuring that the selected features are independent of each other.

Observations from these methods guided the selection of features that contribute meaningfully to the analysis.

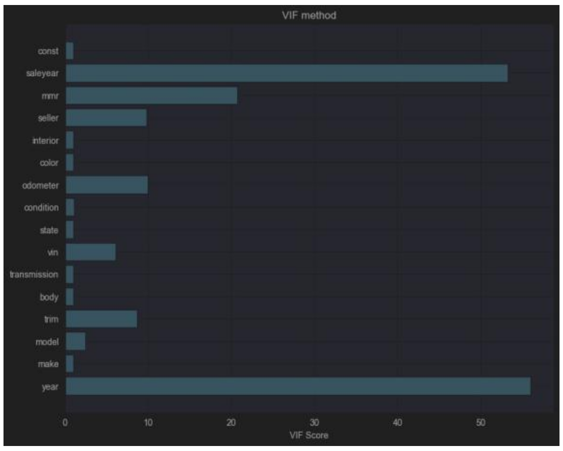


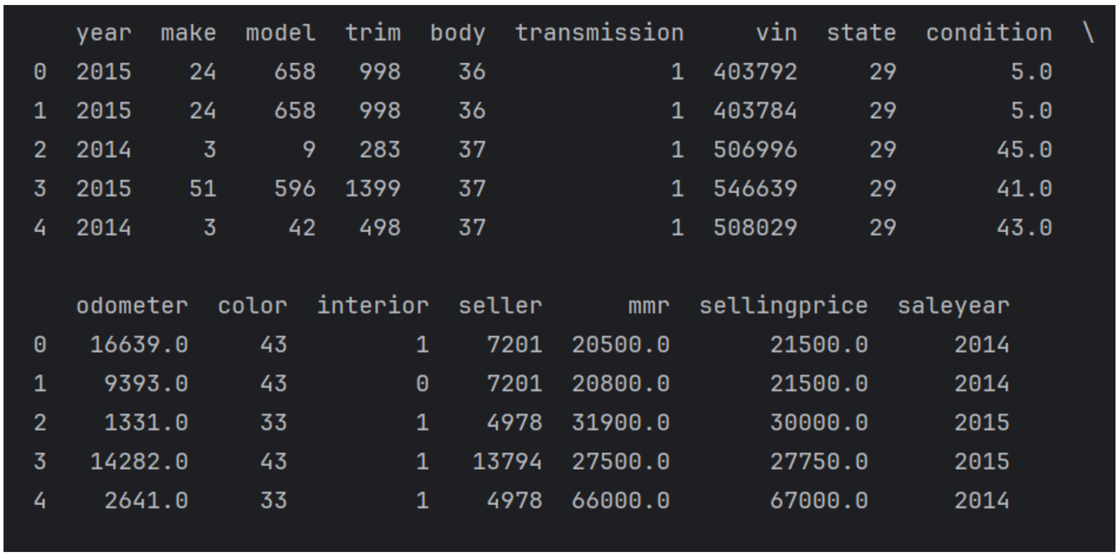
Figure 7: VIF Analysis.

After carefully analyzing all the methods mentioned above, below are final selected features for the Regression Analysis.

['year', 'make', 'model', 'trim', 'condition', 'odometer', 'sellingprice', 'mmr']

3.5 Discretization & Binarization

Label Encoding and One Hot Encoding were applied to convert categorical variables into numerical ones, necessary for modeling. Care was taken to avoid the dummy variable trap by dropping one category per feature, ensuring models do not encounter multicollinearity.

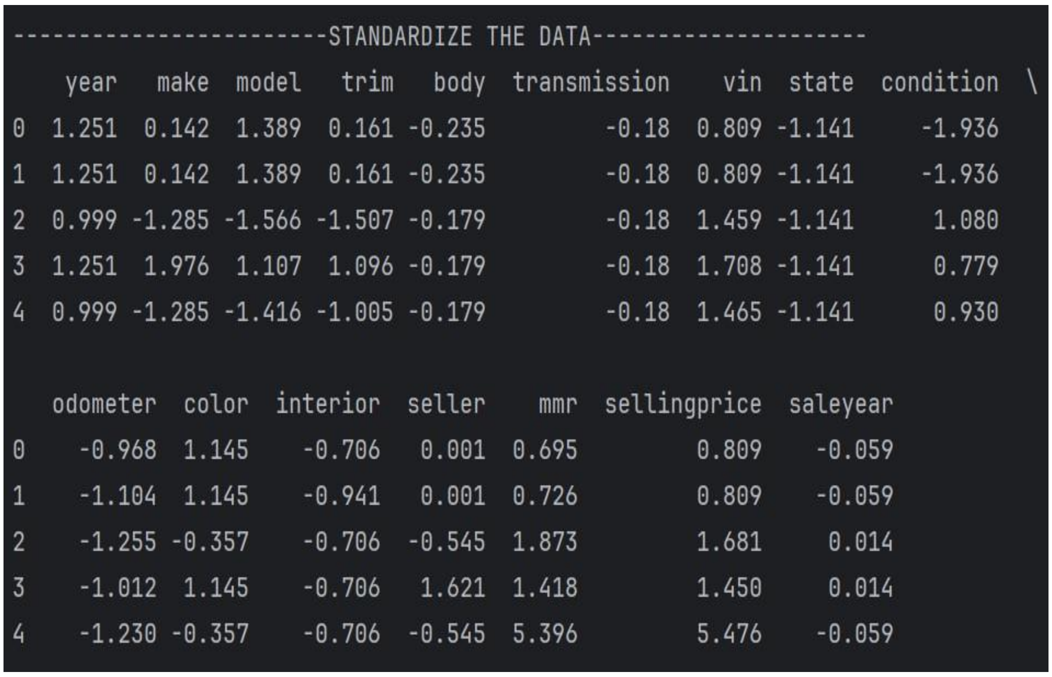

```
   year  make  model  trim  body  transmission     vin  state  condition  \
0  2015    24    658   998    36             1  403792     29        5.0
1  2015    24    658   998    36             1  403784     29        5.0
2  2014     3      9   283    37             1  506996     29       45.0
3  2015    51    596  1399    37             1  546639     29       41.0
4  2014     3     42   498    37             1  508029     29       43.0

   odometer  color  interior  seller      mmr  sellingprice  saleyear
0   16639.0     43         1    7201  20500.0       21500.0      2014
1    9393.0     43         0    7201  20800.0       21500.0      2014
2    1331.0     33         1    4978  31900.0       30000.0      2015
3   14282.0     43         1   13794  27500.0       27750.0      2015
4    2641.0     33         1    4978  66000.0       67000.0      2014
```

Figure 8: Label Encoding and One Hot encoding.

3.6 Variable Transformation

Data normalization and standardization were used to bring all features into a similar scale, eliminating the domination of features with larger scales. Differencing was used to stabilize the mean of time series data by removing changes in the level of a time series, helping to stabilize the variance.



```
-----------------------STANDARDIZE THE DATA--------------------
    year   make  model   trim   body  transmission    vin  state  condition  \
0  1.251  0.142  1.389  0.161 -0.235         -0.18  0.809 -1.141     -1.936
1  1.251  0.142  1.389  0.161 -0.235         -0.18  0.809 -1.141     -1.936
2  0.999 -1.285 -1.566 -1.507 -0.179         -0.18  1.459 -1.141      1.080
3  1.251  1.976  1.107  1.096 -0.179         -0.18  1.708 -1.141      0.779
4  0.999 -1.285 -1.416 -1.005 -0.179         -0.18  1.465 -1.141      0.930

   odometer  color  interior  seller    mmr  sellingprice  saleyear
0    -0.968  1.145    -0.706   0.001  0.695         0.809    -0.059
1    -1.104  1.145    -0.941   0.001  0.726         0.809    -0.059
2    -1.255 -0.357    -0.706  -0.545  1.873         1.681     0.014
3    -1.012  1.145    -0.706   1.621  1.418         1.450     0.014
4    -1.230 -0.357    -0.706  -0.545  5.396         5.476    -0.059
```

Figure 9: Standardized Data

3.7 Anomaly Detection/Outlier Analysis and Removal

Anomaly detection was performed using distance-based and density-based methods to identify and remove outliers that could potentially skew the results. The impacts of these outliers were analyzed, and decisions were made based on their potential influence on the overall dataset.

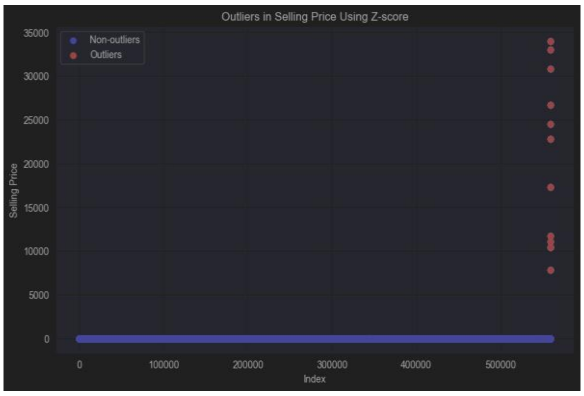


Figure 10: Outlier Analysis Using Z-Score ≥ 3.

3.8 Sample Covariance Matrix

The covariance matrix of the dataset was displayed using a heatmap to visualize the covariance between different variables. Observations from this analysis provided insights into the relationships and dependencies between variables.

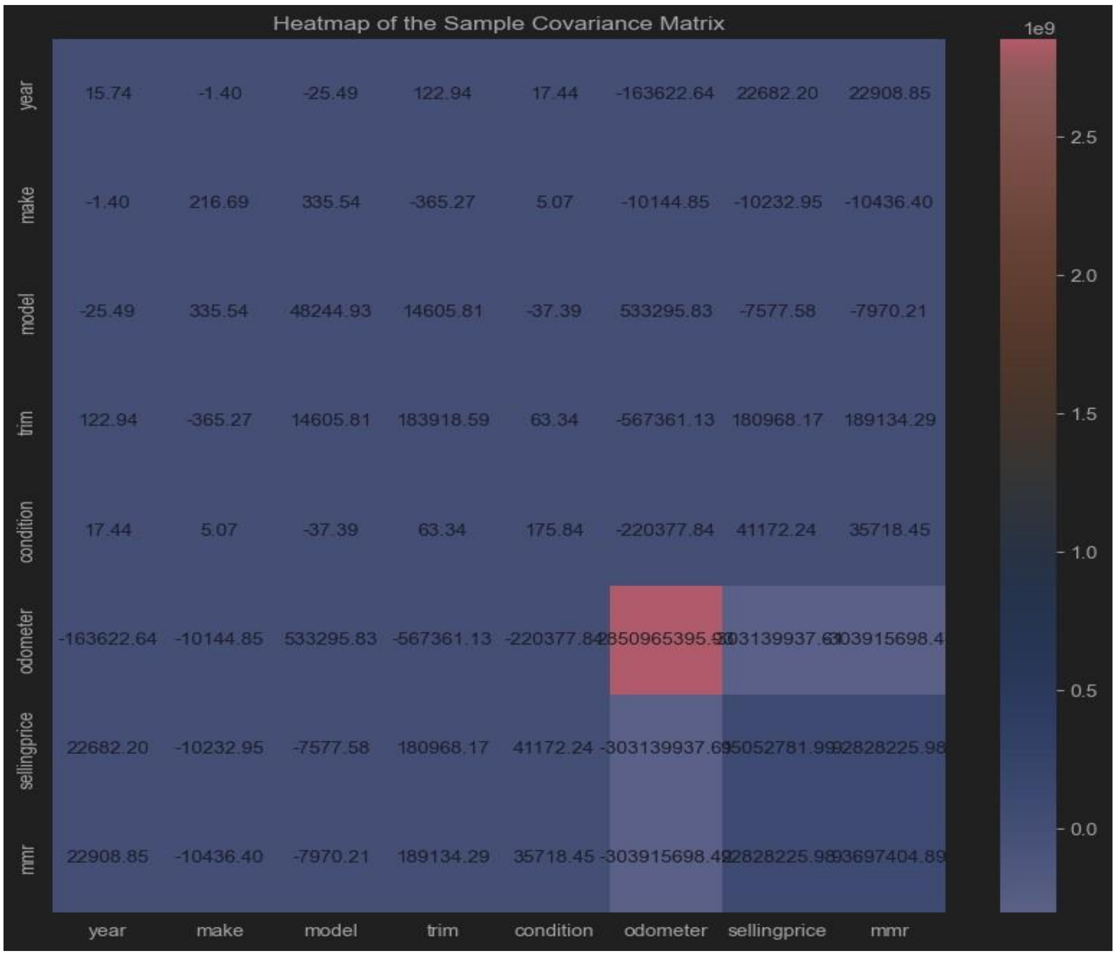


Figure 11: Sample Covariance Matrix.

3.9 Sample Pearson Correlation Coefficients Matrix

Similarly, a heatmap of the Pearson correlation coefficients was generated to assess the linear relationship between pairs. This helped in understanding the bivariate relationships and the strength of the linear associations between variables. From the correlation matrix we can observe that 'mmr' and 'sellingprice' are highly correlated.

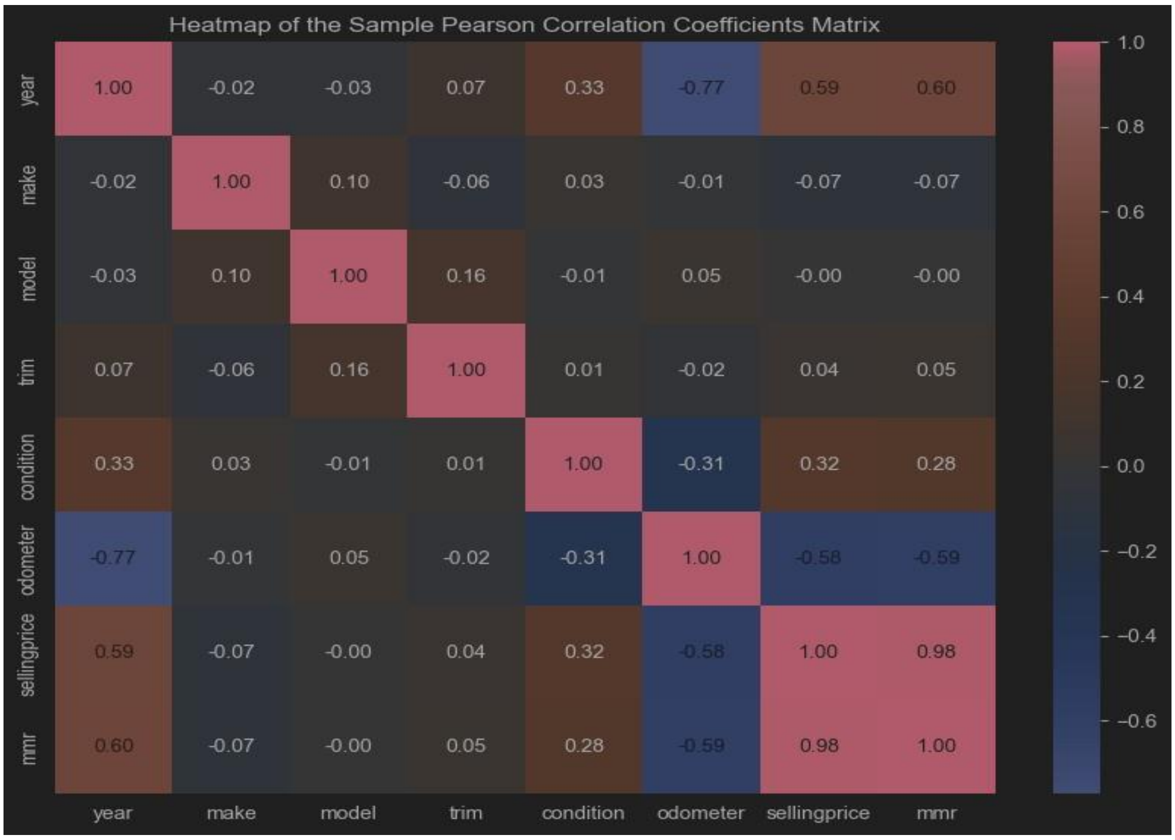


Figure 12: Pearson Correlation Matrix.

3.10 Balancing the Data

The dataset was evaluated for balance between different classes in the target variable. Given an imbalance, techniques such as SMOTE (Synthetic Minority Oversampling Technique) were employed to artificially balance the dataset, ensuring that the model's training process is not biased towards the majority class.

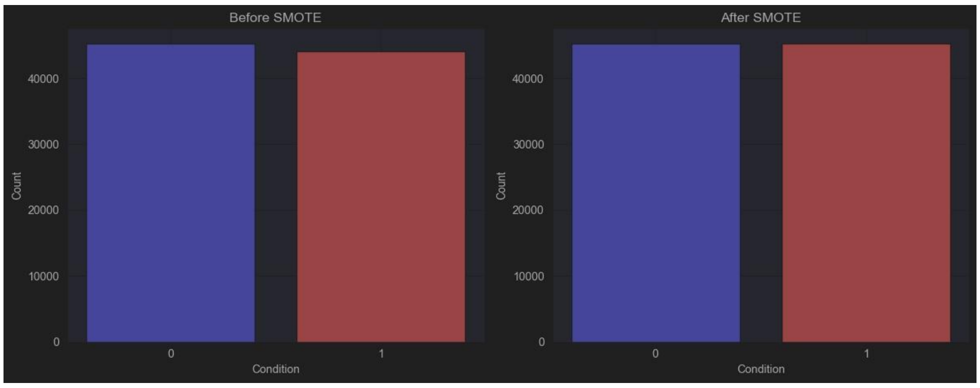


Figure 13: Data Balancing using SMOTE.

4 Phase II: Regression Analysis

This phase focuses on developing a multiple linear regression model to predict a continuous numerical feature of the dataset. The objective is to establish a reliable predictive model and evaluate its performance through various statistical measures and tests.

4.1 Regression Model Development

A multiple linear regression model was constructed using relevant predictors identified during the exploratory data analysis phase. This model aims to predict the selling price of cars based on their attributes such as year, make, model, and odometer readings.

4.2 T-test and F-test Analysis

- **T-test:** Each coefficient in the regression model was tested to determine if it significantly differed from zero. This helps in understanding which features have a statistically significant impact on the selling price.

- **F-test:** The overall significance of the regression model was evaluated using an F-test. This test checks the null hypothesis that no linear relationship exists between the dependent variable and the set of predictors.

4.3 Model Fitting and Diagnostics

The model was fitted using the ordinary least squares method. Key diagnostic plots were used to check for assumptions:

- Residual plots to assess homoscedasticity.
- Q-Q plots to check the normality of residuals.

4.4 Model Performance Evaluation

The model's performance was evaluated using the following criteria:

- **R-squared and Adjusted R-square:** Measures of how well the variations in selling prices are explained by the car attributes.
- **AIC and BIC:** Information criteria for model comparison, balancing the model fit and complexity.
- **MSE (Mean Squared Error):** A measure of the average of the squares of the errors—i.e., the average squared difference between the estimated values and the actual value.

4.5 Visualization of Model Predictions

A plot was created to show actual vs. predicted prices, including both training and testing data sets to visualize model effectiveness. The plot helps in visually assessing the accuracy of predictions made by the regression model.

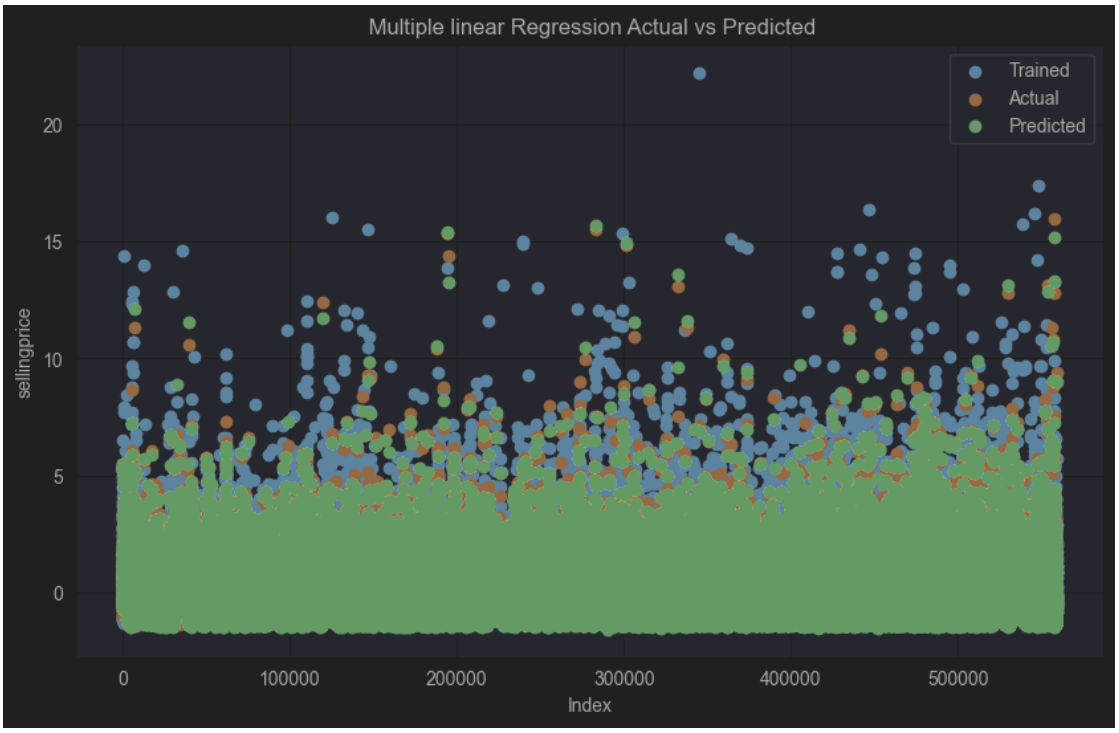
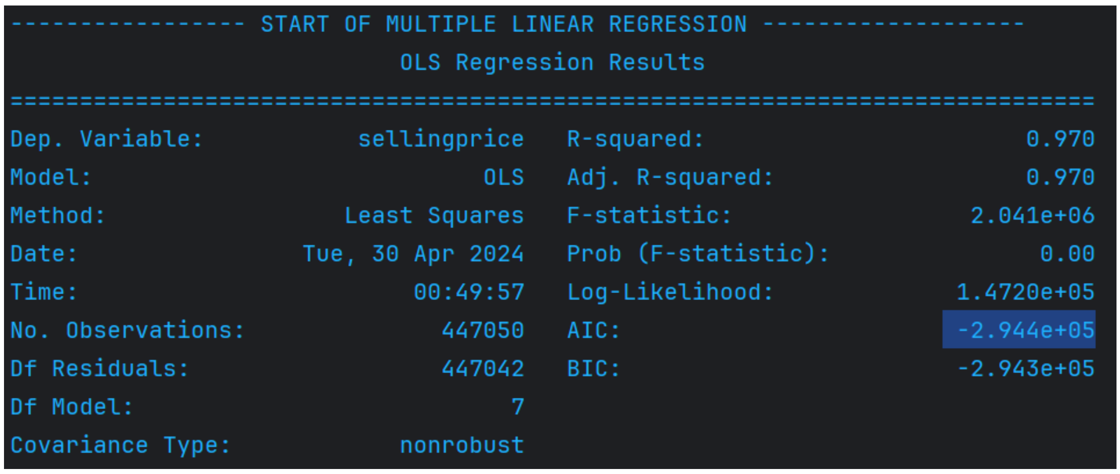
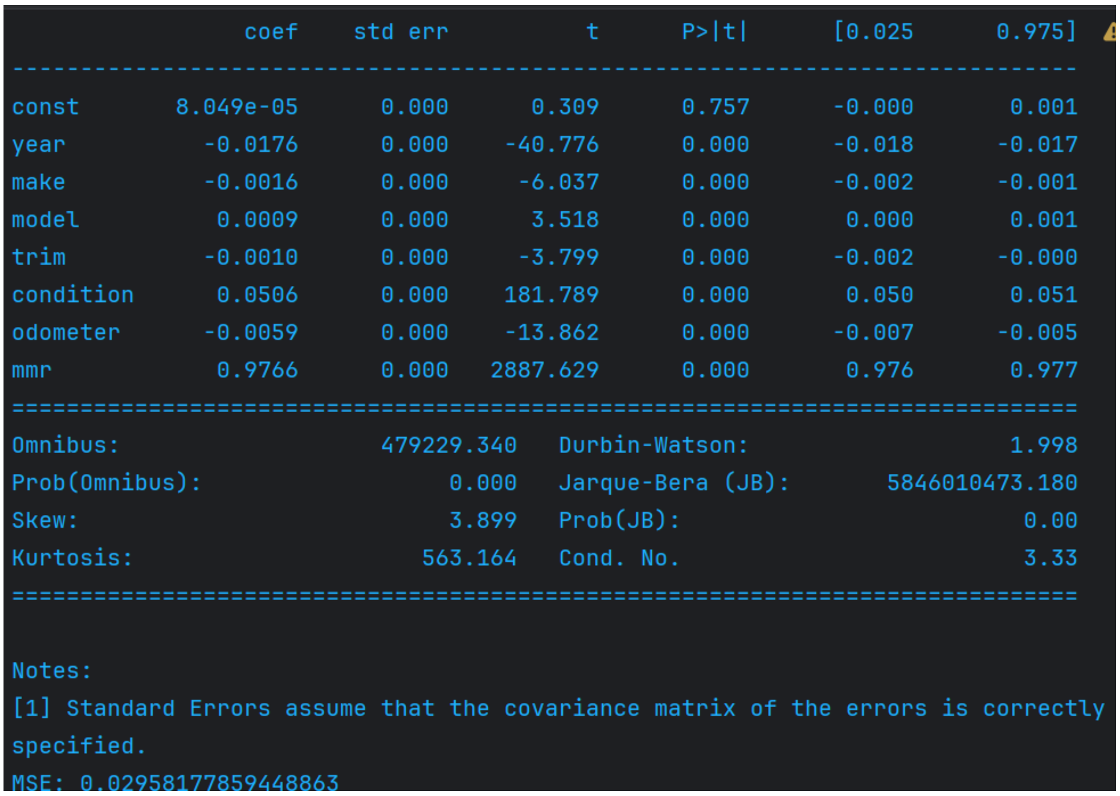


Figure 14: Actual vs Predicted vs Trained.



```
----------------- START OF MULTIPLE LINEAR REGRESSION -------------------
                          OLS Regression Results
==============================================================================
Dep. Variable:           sellingprice   R-squared:                       0.970
Model:                            OLS   Adj. R-squared:                  0.970
Method:                 Least Squares   F-statistic:                 2.041e+06
Date:                Tue, 30 Apr 2024   Prob (F-statistic):               0.00
Time:                        00:49:57   Log-Likelihood:             1.4720e+05
No. Observations:              447050   AIC:                        -2.944e+05
Df Residuals:                  447042   BIC:                        -2.943e+05
Df Model:                           7
Covariance Type:            nonrobust
```

Figure 15: OLS Summary



```
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const       8.049e-05      0.000      0.309      0.757      -0.000       0.001
year          -0.0176      0.000    -40.776      0.000      -0.018      -0.017
make          -0.0016      0.000     -6.037      0.000      -0.002      -0.001
model          0.0009      0.000      3.518      0.000       0.000       0.001
trim          -0.0010      0.000     -3.799      0.000      -0.002      -0.000
condition      0.0506      0.000    181.789      0.000       0.050       0.051
odometer      -0.0059      0.000    -13.862      0.000      -0.007      -0.005
mmr            0.9766      0.000   2887.629      0.000       0.976       0.977
==============================================================================
Omnibus:                 479229.340   Durbin-Watson:                   1.998
Prob(Omnibus):                0.000   Jarque-Bera (JB):       5846010473.180
Skew:                         3.899   Prob(JB):                         0.00
Kurtosis:                   563.164   Cond. No.                         3.33
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
MSE: 0.02958177859448863
```

Figure 16: OLS Summary

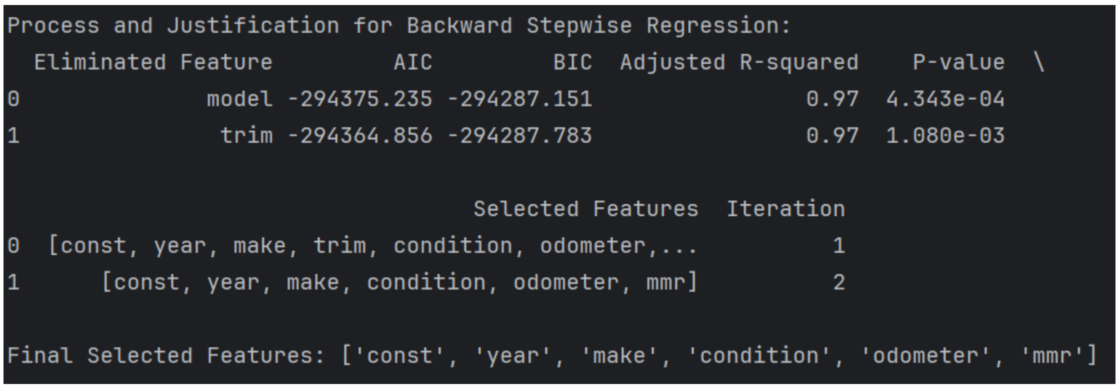
4.6 Confidence Interval Analysis

Confidence intervals for regression coefficients were calculated, providing a range of values that are likely to contain the true value of the coefficients with a certain probability.

4.7 Stepwise Regression

Stepwise regression was employed to refine the model further:

- Features were selected based on their statistical significance and contribution to the model's explanatory power.
- Adjusted R-square values were monitored to ensure that the model complexity did not unduly increase without substantial gain in model performance.



```
Process and Justification for Backward Stepwise Regression:
  Eliminated Feature         AIC         BIC  Adjusted R-squared    P-value  \
0              model -294375.235 -294287.151                0.97  4.343e-04
1               trim -294364.856 -294287.783                0.97  1.080e-03

                                   Selected Features  Iteration
0  [const, year, make, trim, condition, odometer,...          1
1      [const, year, make, condition, odometer, mmr]          2

Final Selected Features: ['const', 'year', 'make', 'condition', 'odometer', 'mmr']
```

Figure 17: Stepwise Regression Iterations Summary

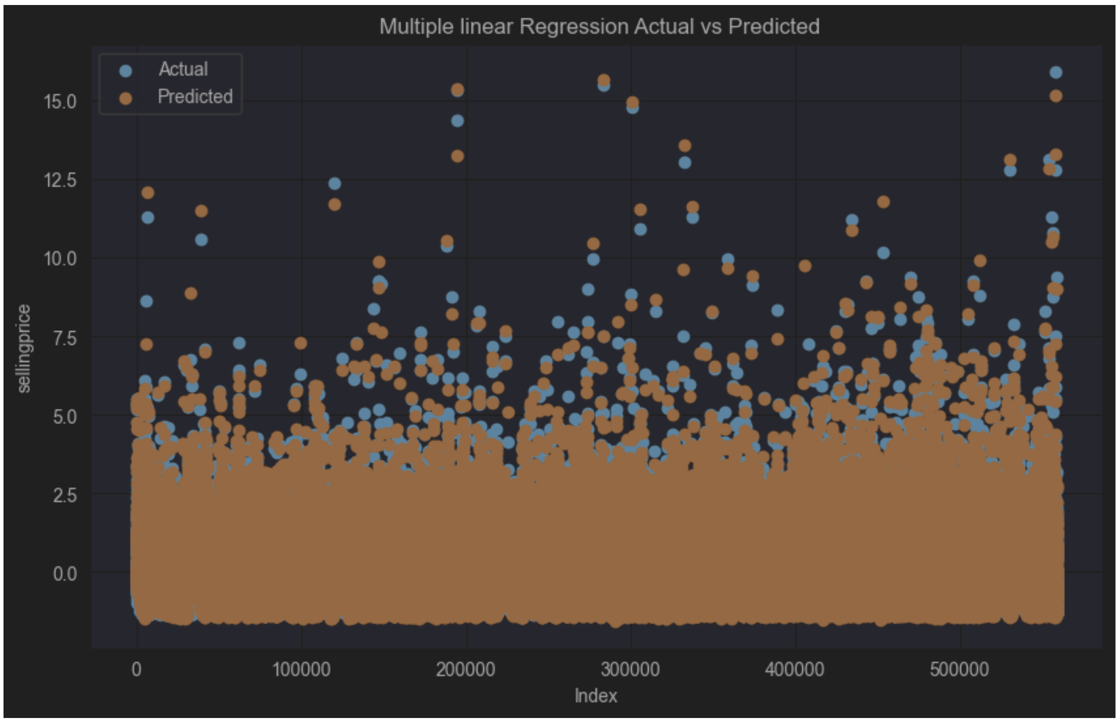


Figure 18: Stepwise Regression Actual vs Predicted.

4.8 Final Regression Model and Prediction

The final model was presented along with the regression equation. Predictions from this model were used to estimate the selling prices based on the car attributes. Observations and insights derived from the model predictions were discussed, providing actionable intelligence for stakeholders.

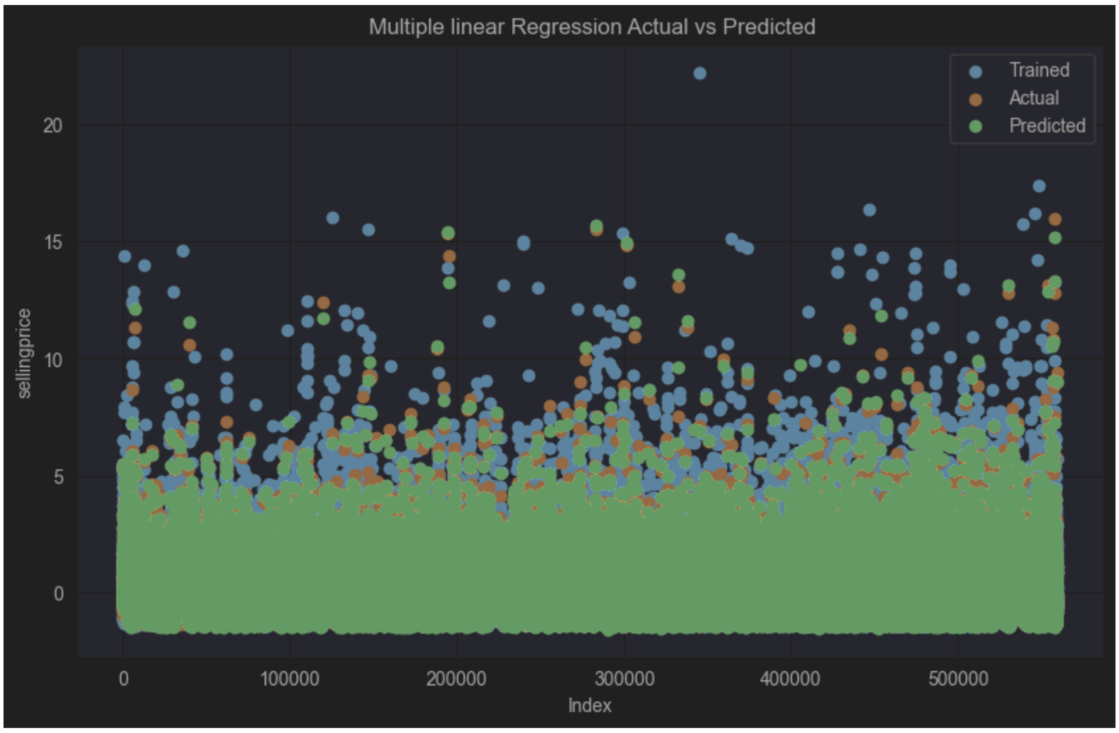


Figure 19: Final Model Actual vs Predicted vs Trained Data

Regression Equation:

y = 0.0001 + -0.0176*X1 + -0.0016*X2 + 0.0009*X3 + -0.0010*X4 + 0.0506*X5 + -0.0059*X6 + 0.9766*X7

4.9 Model Summary Table

A comprehensive table summarizing the regression analysis results was included. This table featured R-squared, adjusted R-square, AIC, BIC, and MSE, facilitating a clear understanding of the model's efficacy.

Metric	Value
R-squared	0.97
Adjusted R-squared	0.97

AIC	-0.02944
BIC	-0.02935
MSE	0.0295

Table 1: Summary of Regression Model Metrics

The linear regression model demonstrates a very high level of accuracy and reliability in predicting the target variable, as evidenced by the following key metrics:

- **R-squared: 0.97** - This value indicates that 97% of the variance in the dependent variable is explained by the independent variables included in the model. An R-squared value close to 1 suggests that the model has a very high level of predictive accuracy.
- **Adjusted R-squared: 0.97** - Adjusted R-squared also stands at 0.97, which is particularly noteworthy because it adjusts the R-squared value to account for the number of predictors in the model. This high value confirms that the model fits the data very well, even after the adjustment for the number of variables.
- **MSE (Mean Squared Error): 0.0295** - The MSE, which measures the average of the squares of the errors, is quite low. This indicates that the predictions made by the model are very close to the actual data points, leading to a minimal deviation in the predicted values from the real values.
- **AIC (Akaike Information Criterion): -0.02944** - AIC is a criterion used for model selection among a finite set of models. The lower the AIC, the better the model is at explaining the variation in the data while keeping the model simplicity. The negative value here suggests a model that balances model complexity and fit efficiently.
- **BIC (Bayesian Information Criterion): -0.02935** - Similar to AIC, BIC is used for model selection, but it includes a penalty term for the number of parameters in the model. The negative BIC value indicates a model that is optimal for balancing between goodness of fit and model complexity, in a Bayesian context.

Overall, the combination of high R-squared and Adjusted R-squared values, along with a low MSE, and very favorable AIC and BIC values, strongly suggests that the linear regression model provides an excellent fit to the data. It effectively captures the essential patterns and can be considered a reliable tool for prediction within this specific context. Future work might explore potential improvements or validations with additional data, but the current model stands as a robust analytical tool.

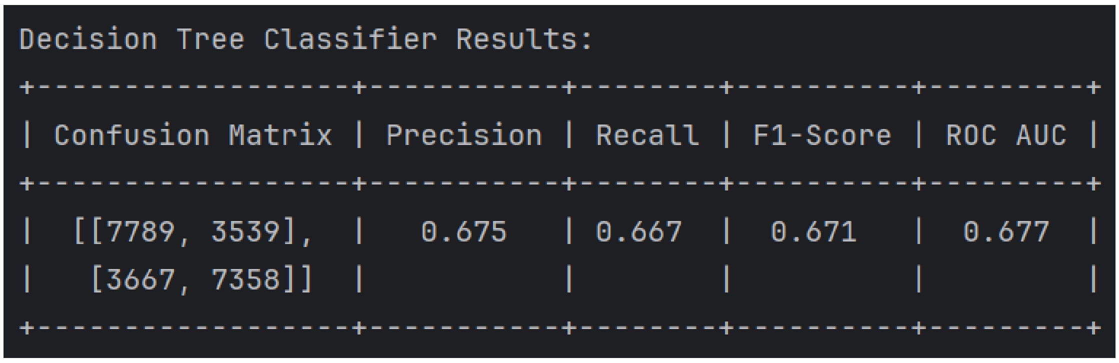
5 Phase III: Classification Analysis

In this phase, various machine learning classifiers were applied to predict a target variable, with an emphasis on evaluating and optimizing each model to achieve the best possible performance without overfitting or underfitting. Extensive hyperparameter tuning was conducted through grid search for each classifier.

5.1 Decision Tree Classifier

5.1.1 Model Optimization

Decision Trees were optimized using pre-pruning, post-pruning, and hyperparameter tuning via grid search. Parameters such as ’criterion’, ’splitter’, ’max depth’, ’min samples _split’, ’max features’, and ’ccp alpha’ were systematically tested.



```
Decision Tree Classifier Results:
+------------------+-----------+--------+----------+---------+
| Confusion Matrix | Precision | Recall | F1-Score | ROC AUC |
+------------------+-----------+--------+----------+---------+
|  [[7789, 3539],  |   0.675   | 0.667  |  0.671   |  0.677  |
|   [3667, 7358]]  |           |        |          |         |
+------------------+-----------+--------+----------+---------+
```

Figure 20: Basic Decision Tree Modal Metrics

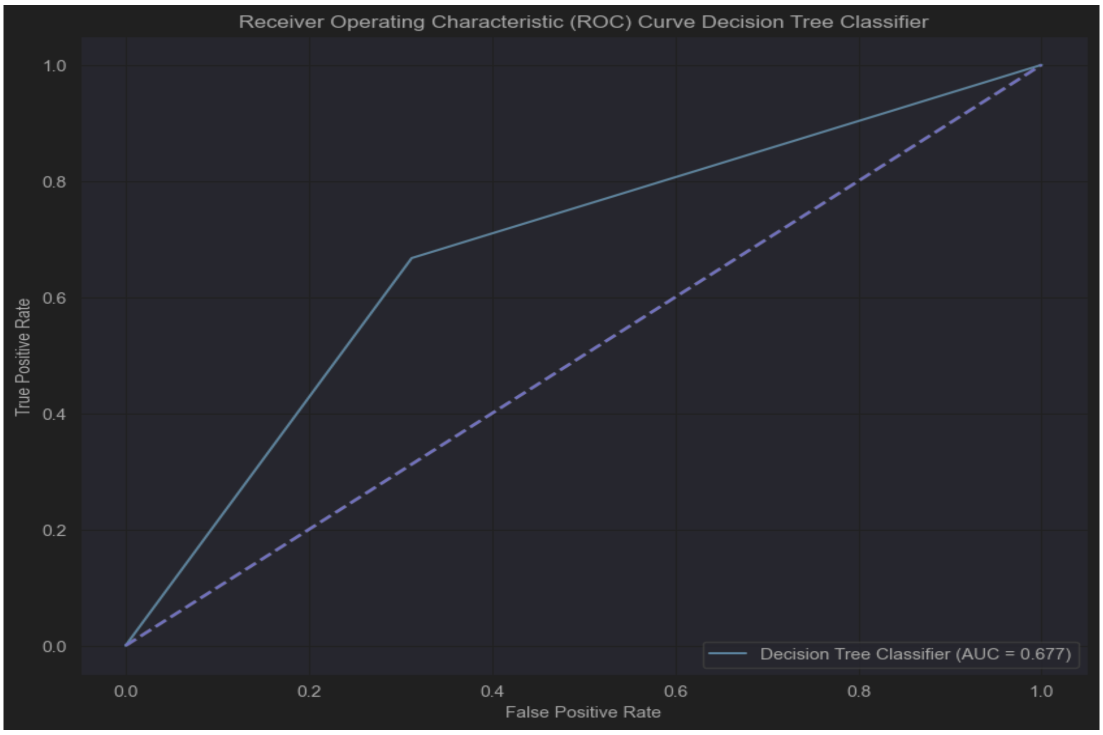


Figure 21: without tuning.

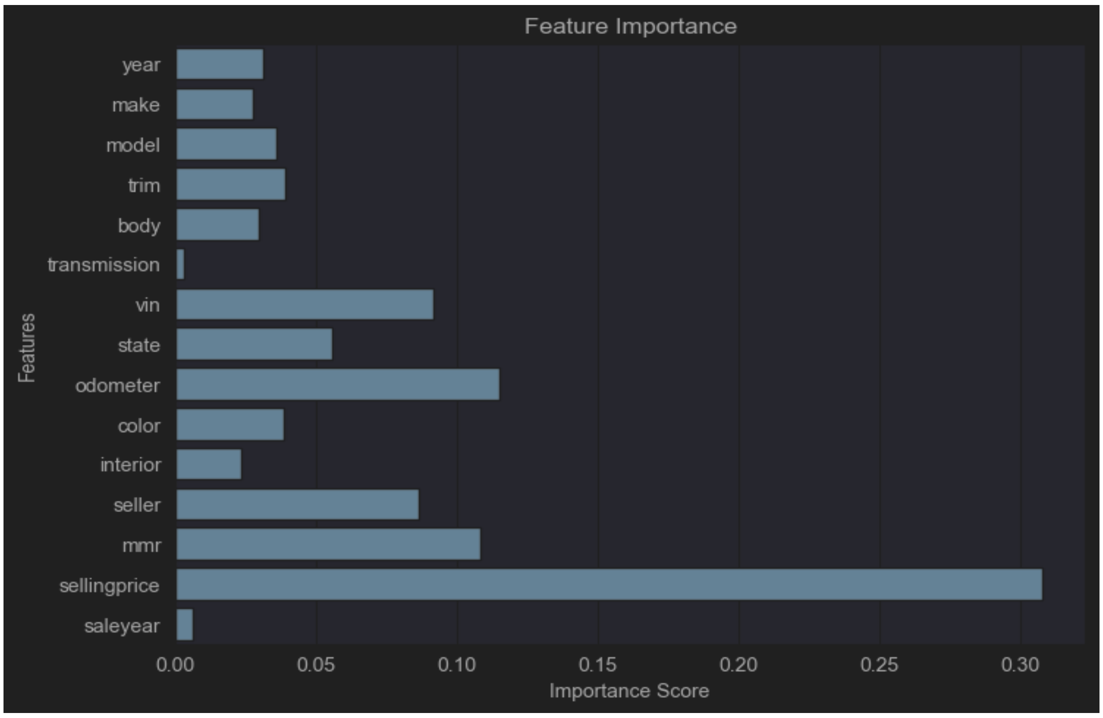


Figure 22: Feature Importances in Decision Tree

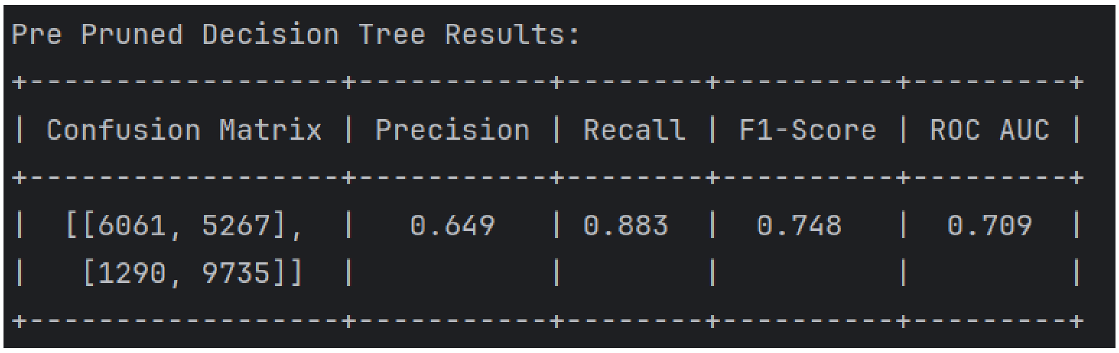
5.1.2 Cost Complexity Optimization

The Cost Complexity Pruning parameter, ccp alpha, was adjusted to find the optimal balance between tree depth and error reduction, minimizing overfitting.

Pre-Pruning

{'criterion': 'gini', 'max_depth': 2, 'max_features': None, 'min_samples_leaf': 2, 'min_samples_split': 2, 'splitter': 'best'}

Accuracy of pre tuned tree 0.707.



```
Pre Pruned Decision Tree Results:
+------------------+-----------+--------+----------+---------+
| Confusion Matrix | Precision | Recall | F1-Score | ROC AUC |
+------------------+-----------+--------+----------+---------+
|   [[6061, 5267],  |   0.649   | 0.883  |  0.748   |  0.709  |
|    [1290, 9735]]  |           |        |          |         |
+------------------+-----------+--------+----------+---------+
```

Figure 23: Pre pruning Decision Tree Metrics

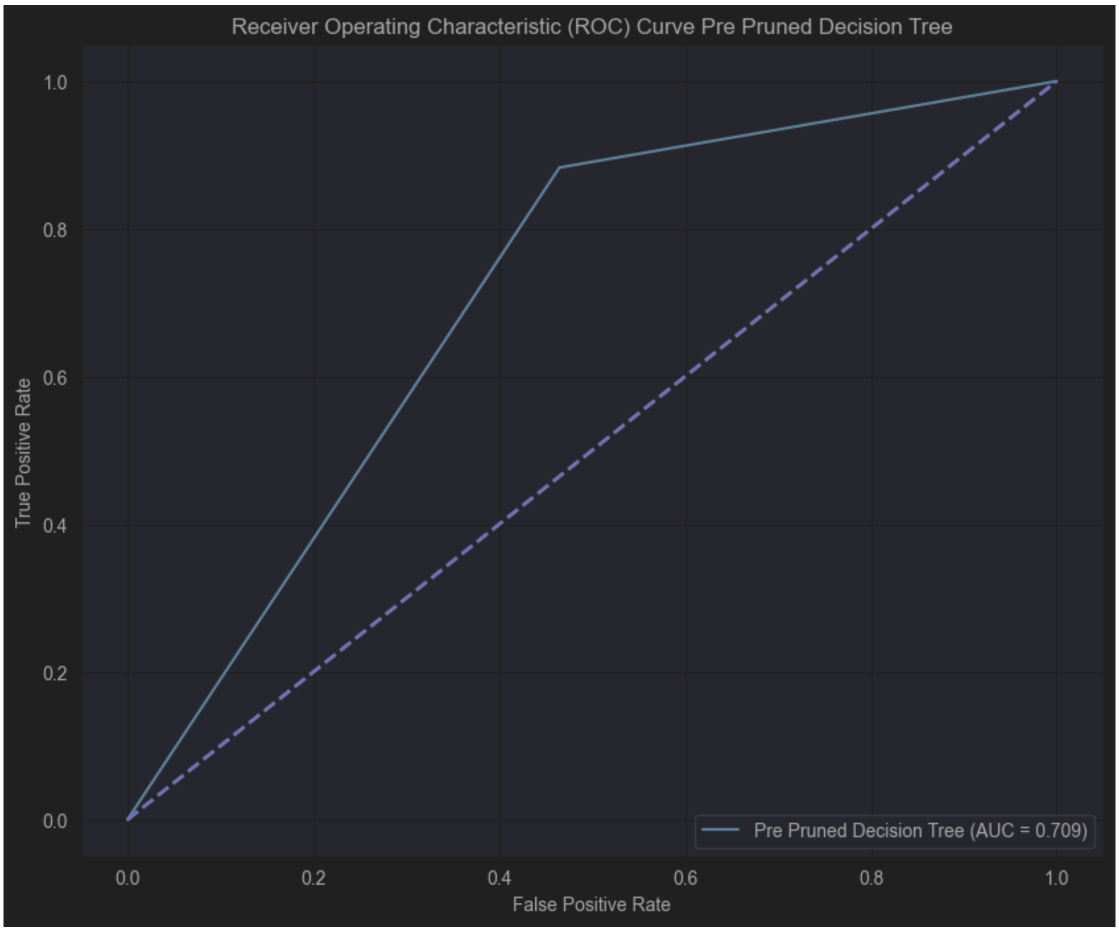


Figure 24: Pre-Pruning Decision Tree Roc Curve

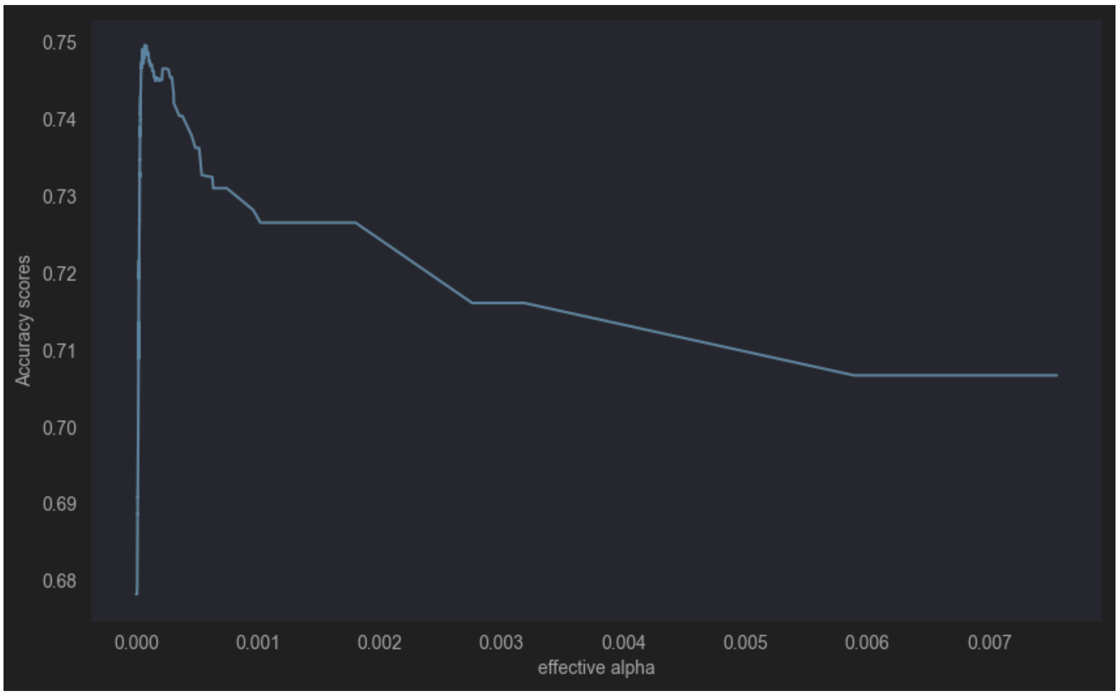
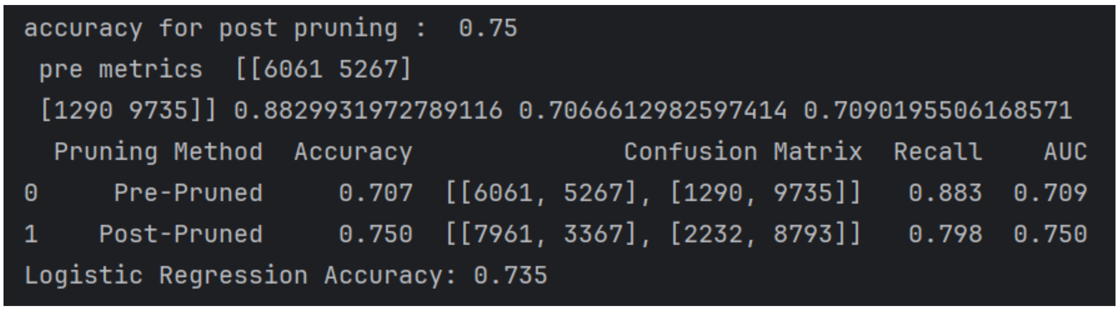


Figure 25: Effective Alpha values Vs Accuracy scores.

5.1.3 Performance Metrics

The performance of the optimized Decision Tree model was assessed using metrics such as precision, recall, F1-score, and the area under the ROC curve (AUC).
Stratified K-fold cross-validation was used to ensure robustness against overfitting.



```
accuracy for post pruning :  0.75
 pre metrics  [[6061 5267]
 [1290 9735]] 0.8829931972789116 0.7066612982597414 0.7090195506168571
  Pruning Method  Accuracy              Confusion Matrix  Recall    AUC
0     Pre-Pruned     0.707  [[6061, 5267], [1290, 9735]]   0.883  0.709
1    Post-Pruned     0.750  [[7961, 3367], [2232, 8793]]   0.798  0.750
Logistic Regression Accuracy: 0.735
```

Figure 26: Metrics for Post pruning Tree

Figure 27: Post pruning Tree.

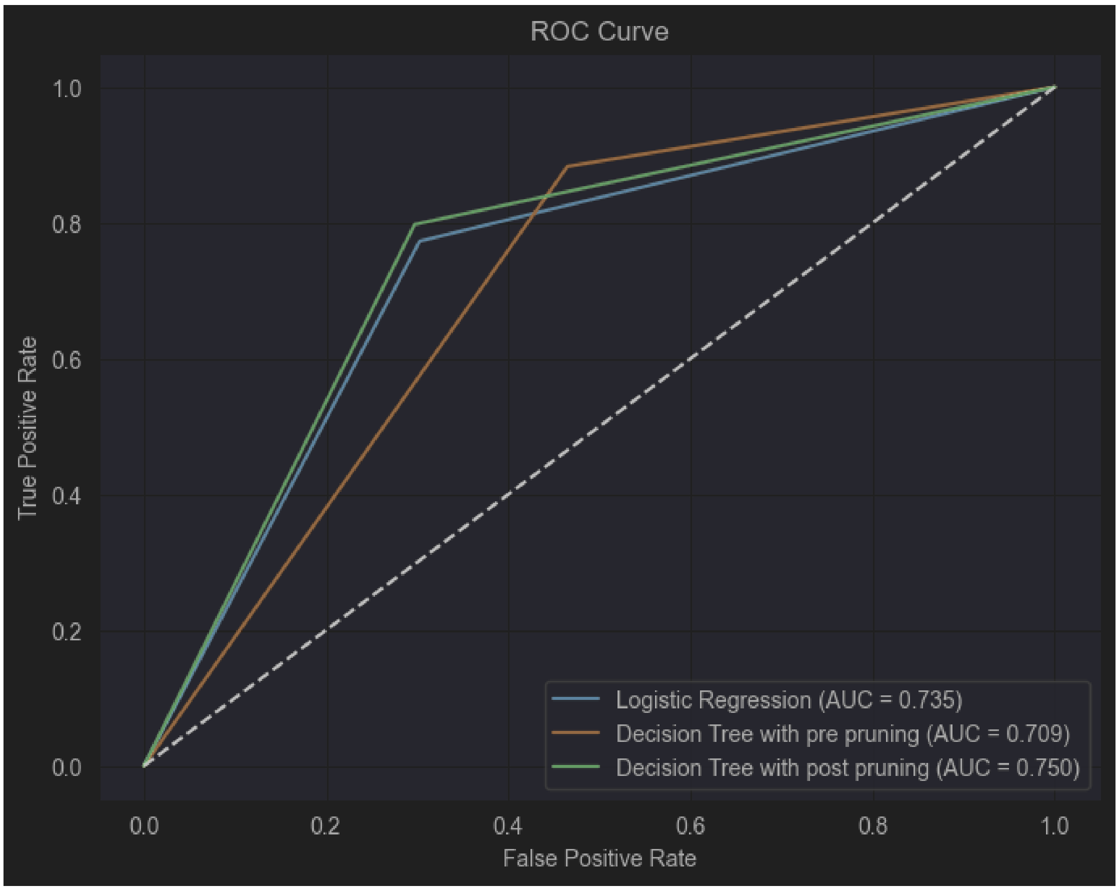


Figure 28: Comparing Post Pruning, Pre-Pruning, Logistic Regression

5.2 Logistic Regression

5.2.1 Hyperparameter Tuning

Grid search was employed to optimize the regularization strength and the choice of solver in Logistic Regression.

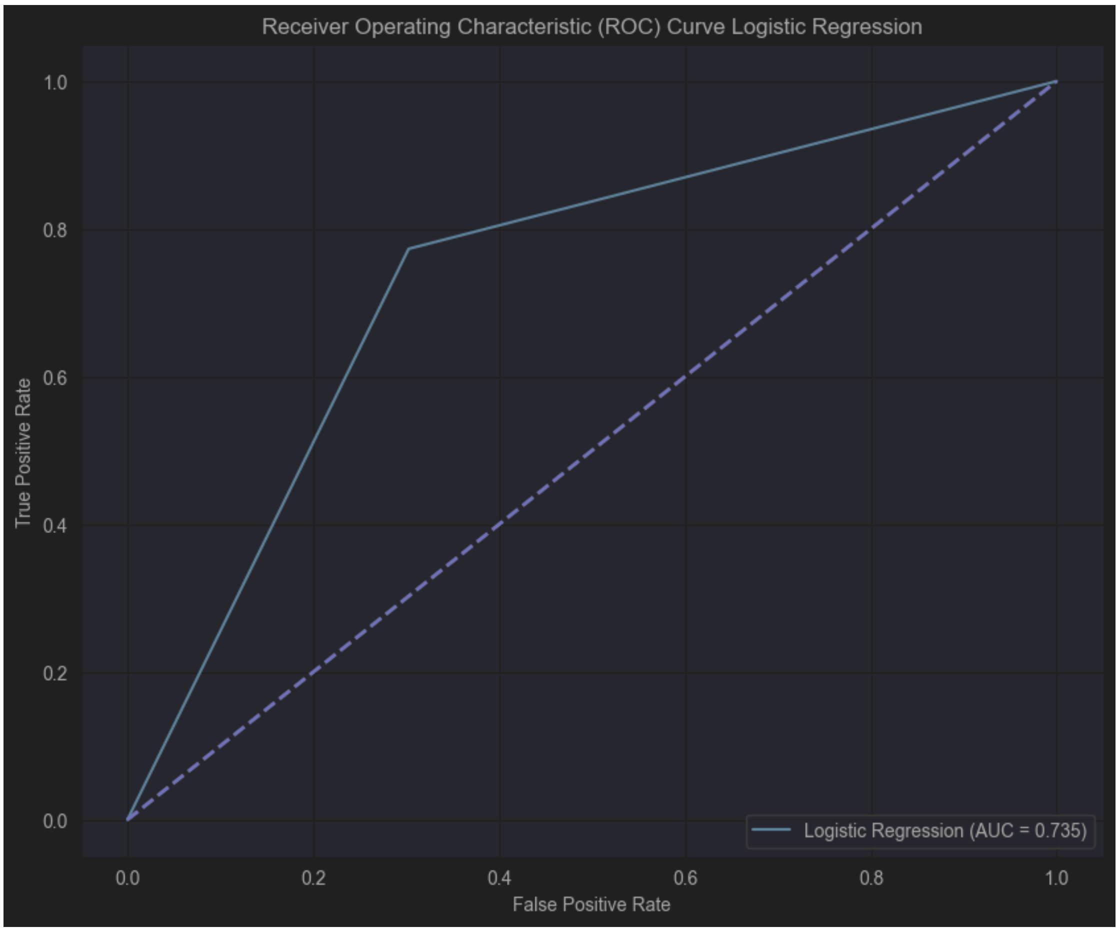
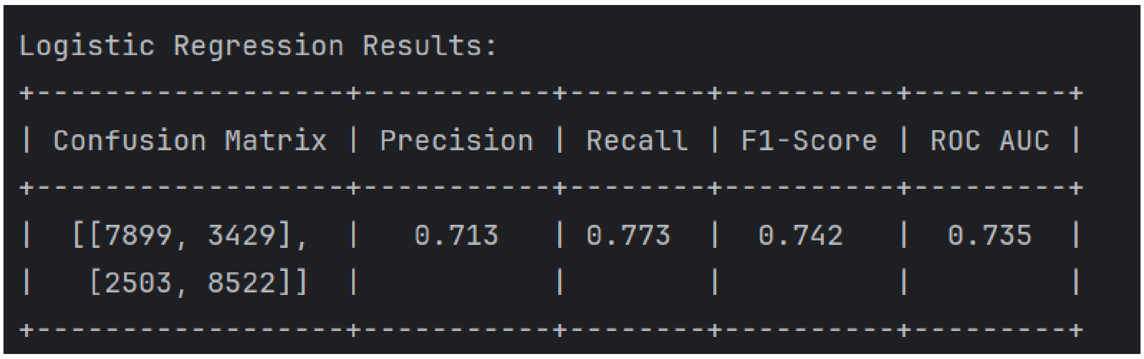


Figure 29: Logistic regression ROC Curve

5.2.2 Performance Metrics

The efficacy of the Logistic Regression model was evaluated using a confusion matrix, precision, recall, specificity, F1-score, and ROC AUC analysis.



```
Logistic Regression Results:
+-----------------+----------+-------+---------+---------+
| Confusion Matrix | Precision | Recall | F1-Score | ROC AUC |
+-----------------+----------+-------+---------+---------+
|  [[7899, 3429],  |   0.713   | 0.773 |  0.742   |  0.735   |
|   [2503, 8522]]  |           |       |          |          |
+-----------------+----------+-------+---------+---------+
```

Figure 30: Logistic Regression Metrics

5.3 K-Nearest Neighbors (KNN)

5.3.1 Optimum K Determination

The optimal number of neighbors, K, was determined using the elbow method, which involved plotting error rates for different values of K and choosing the K at the elbow point, where the decrease in error rate diminishes.

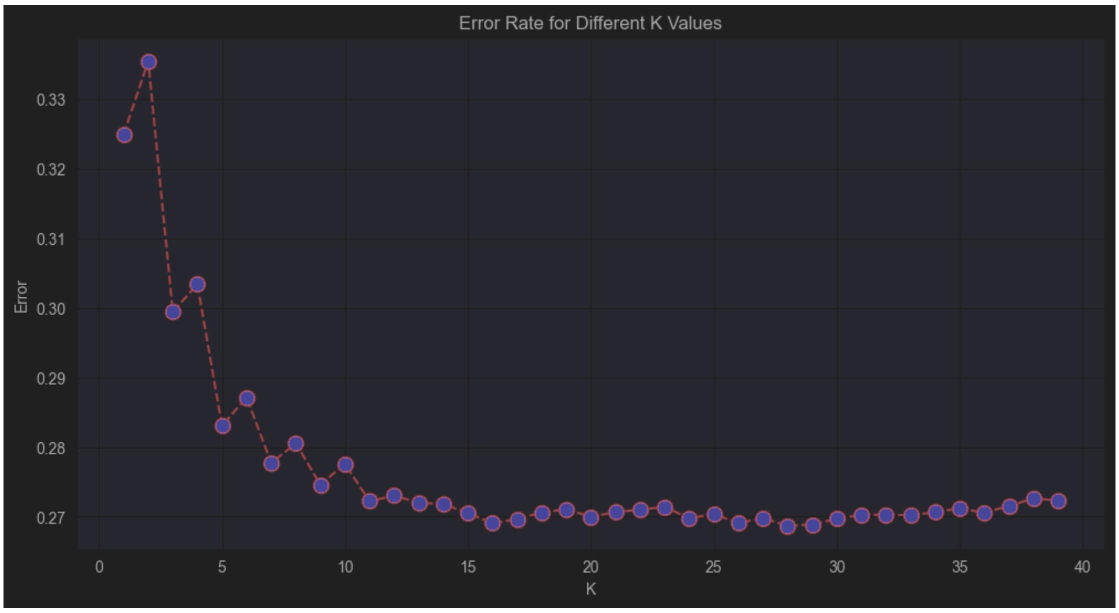
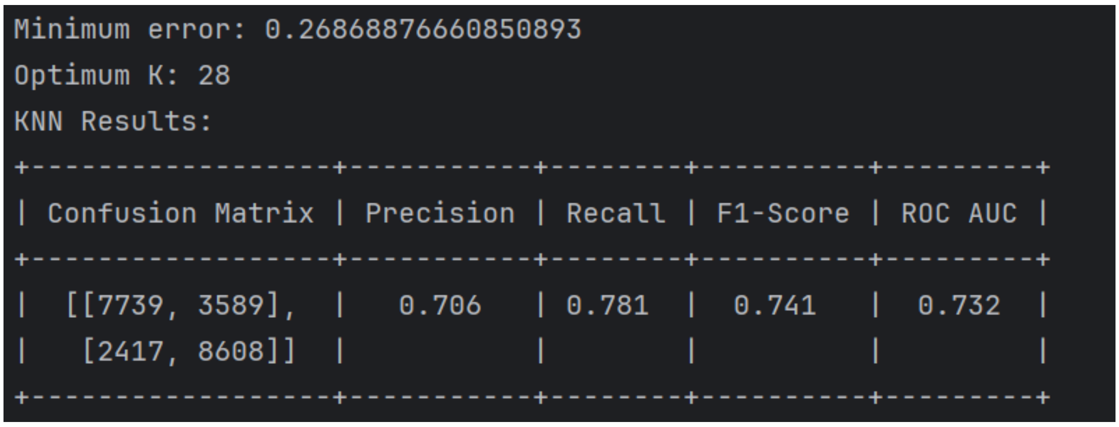


Figure 31: Error rate vs K

5.3.2 Performance Metrics

KNN performance was detailed through the same set of metrics, including sensitivity and specificity, supported by graphical representation of ROC curves.



```
Minimum error: 0.26868876660850893
Optimum K: 28
KNN Results:
+------------------+-----------+--------+----------+---------+
| Confusion Matrix | Precision | Recall | F1-Score | ROC AUC |
+------------------+-----------+--------+----------+---------+
|  [[7739, 3589],  |   0.706   | 0.781  |  0.741   |  0.732  |
|   [2417, 8608]]  |           |        |          |         |
+------------------+-----------+--------+----------+---------+
```

Figure 32: Performance Metrics of KNN

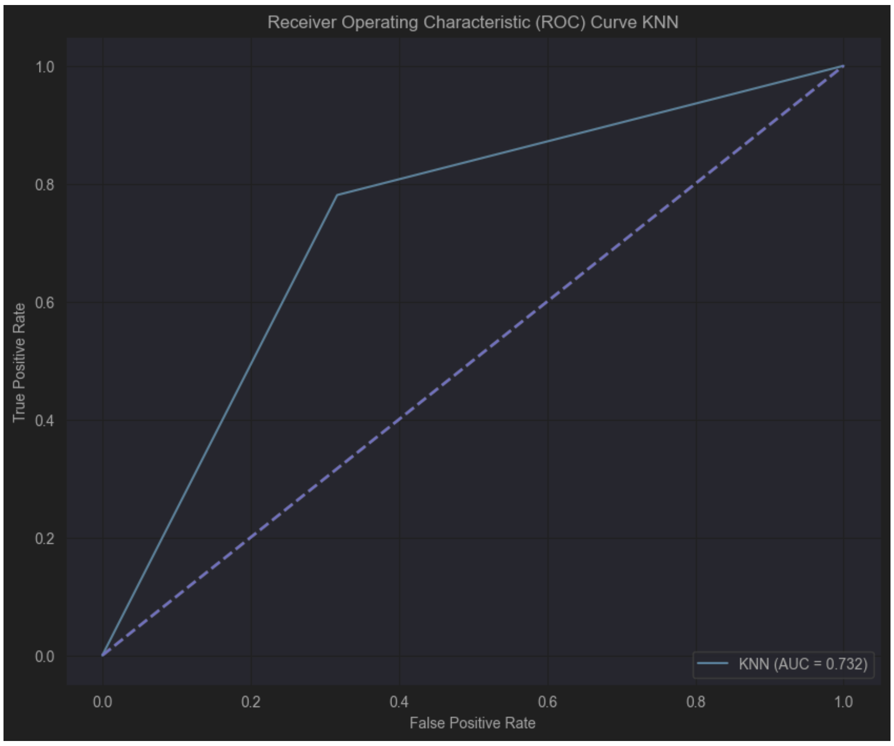


Figure 33: ROC Curve for KNN

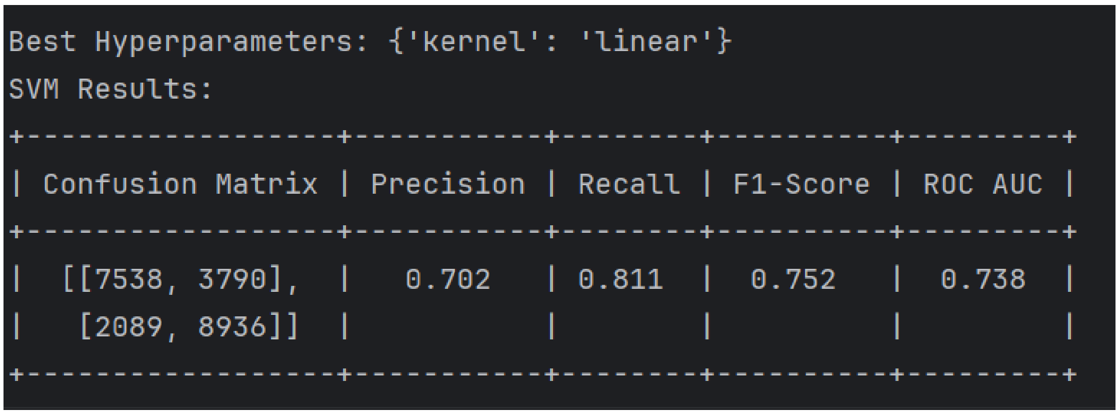
5.4 Support Vector Machine (SVM)

5.4.1 Kernel Optimization

SVM was tested with different kernels: linear, polynomial, and radial base function (RBF) to determine which kernel provides the best separation between classes.

5.4.2 Performance Metrics

The performance was quantified and compared across different kernels using precision, recall, specificity, F1-score, and the area under the ROC curve.



```
Best Hyperparameters: {'kernel': 'linear'}
SVM Results:
+------------------+-----------+--------+----------+---------+
| Confusion Matrix | Precision | Recall | F1-Score | ROC AUC |
+------------------+-----------+--------+----------+---------+
|  [[7538, 3790],  |   0.702   | 0.811  |  0.752   |  0.738  |
|   [2089, 8936]]  |           |        |          |         |
+------------------+-----------+--------+----------+---------+
```

Figure 34:Performance Metrics of SVM

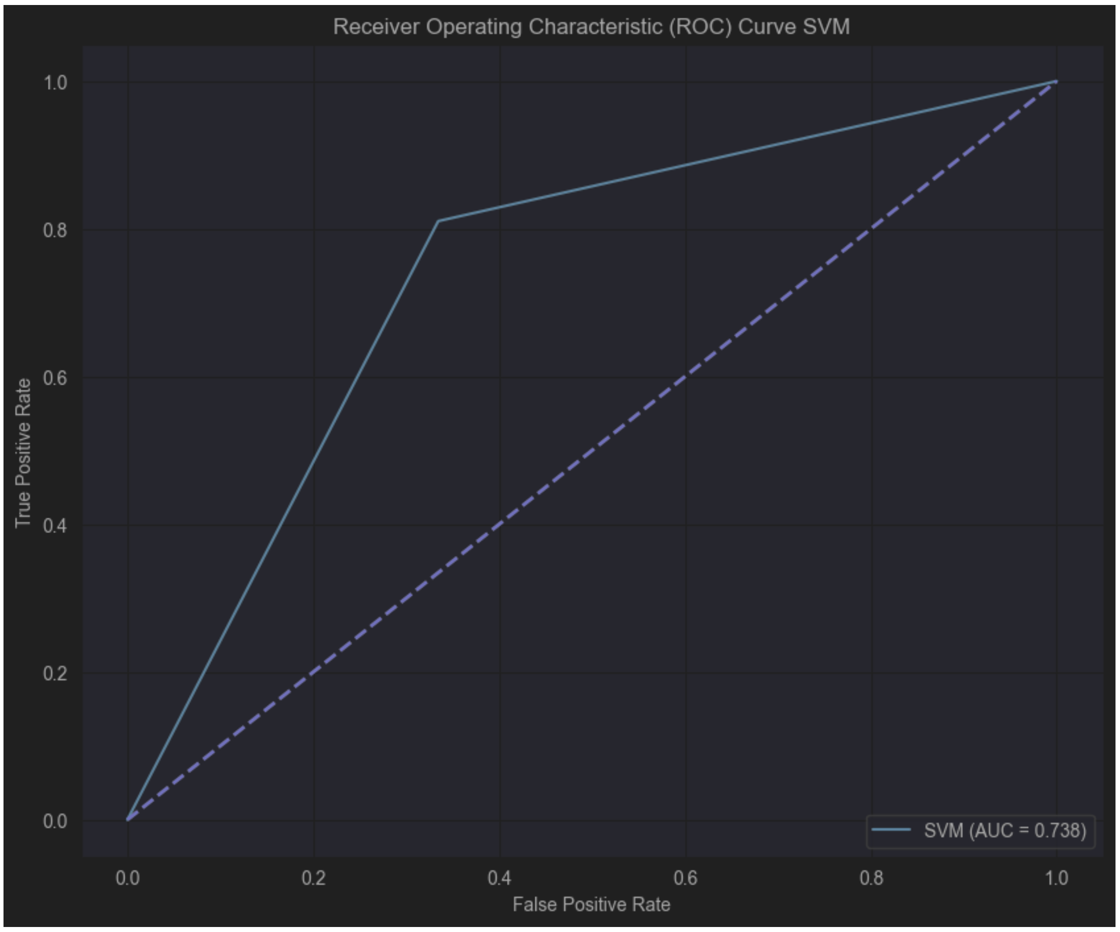
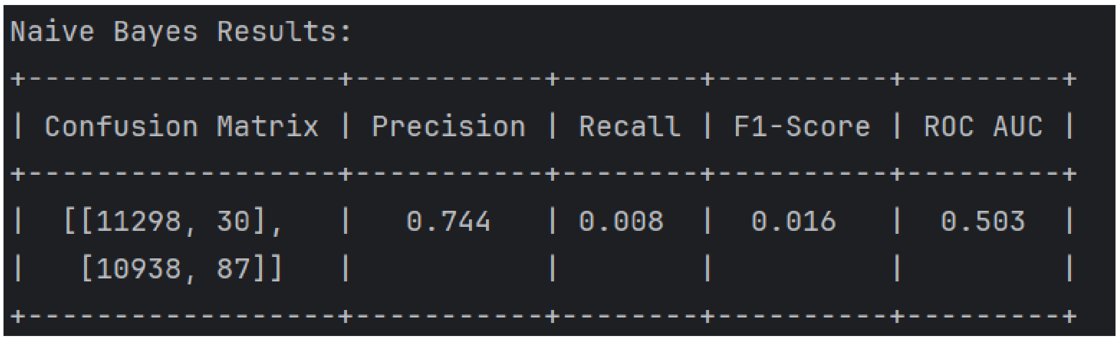


Figure 35: ROC Curve for SVM

5.5 Naïve Bayes

5.5.1 Model Application

The Naïve Bayes classifier was applied, given its effectiveness in high-dimensional datasets.



```
Naive Bayes Results:
+------------------+-----------+--------+----------+---------+
| Confusion Matrix | Precision | Recall | F1-Score | ROC AUC |
+------------------+-----------+--------+----------+---------+
|  [[11298, 30],   |   0.744   | 0.008  |  0.016   |  0.503  |
|   [10938, 87]]   |           |        |          |         |
+------------------+-----------+--------+----------+---------+
```

Figure 36: Naive Bayes Performance Metrics

5.5.2 Performance Metrics

Performance was evaluated using standard classification metrics and visualized through ROC curve analysis.

Stratified K-fold Cross-Validation F1-Score: 0.010

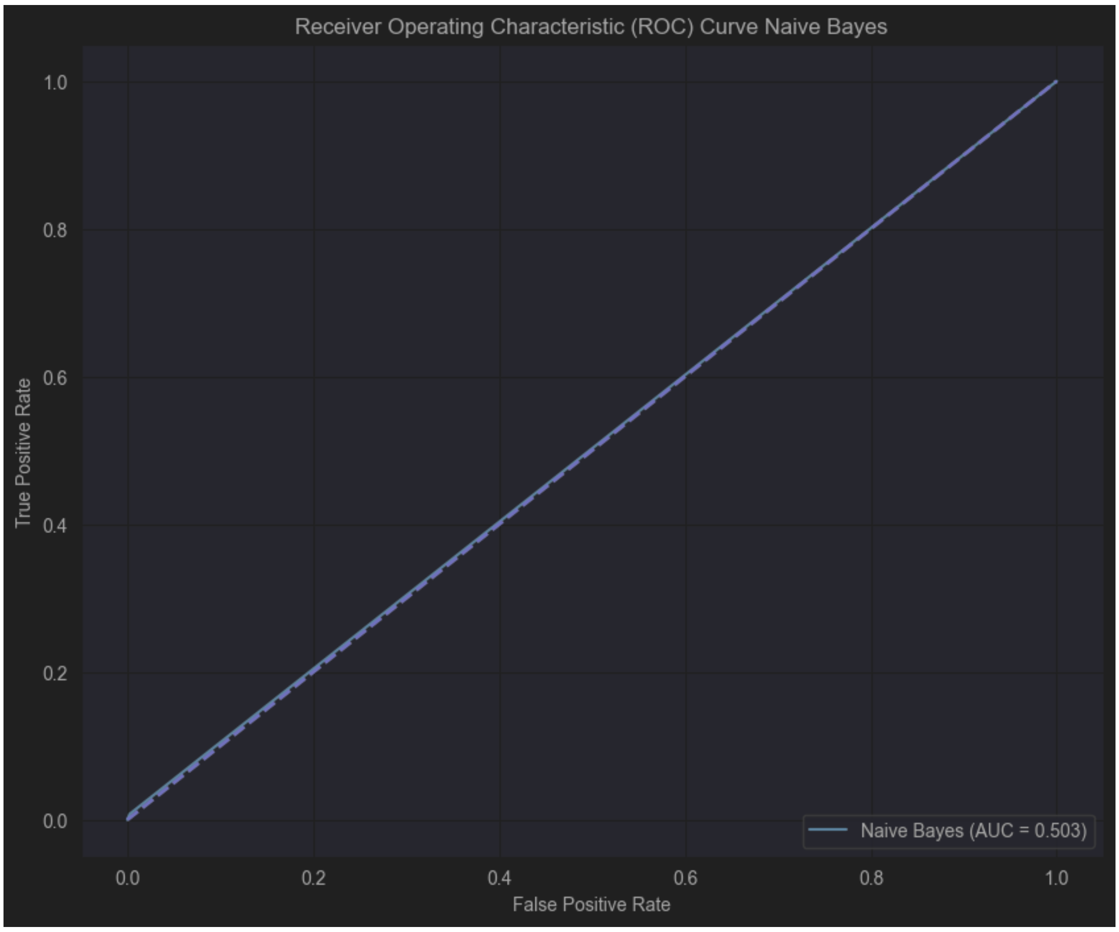


Figure 37: ROC Curve for Naive Bayes

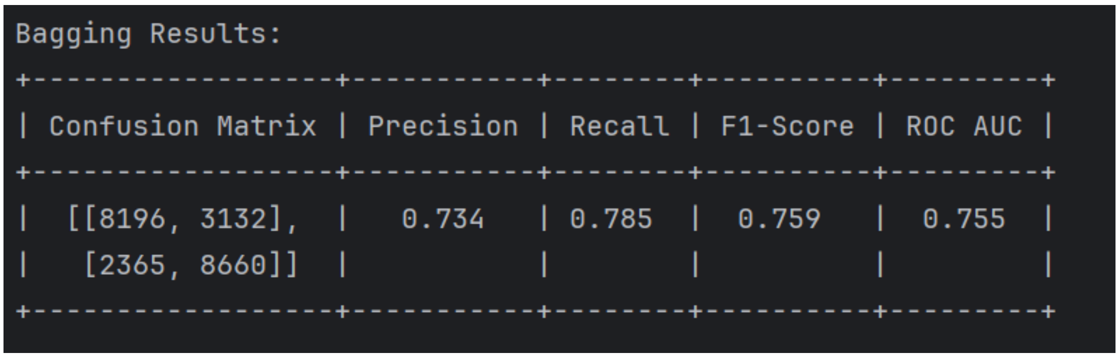
5.6 Random Forest

5.6.1 Ensemble Techniques

Random Forest's performance was enhanced using bagging, stacking, and boosting to reduce variance and bias.

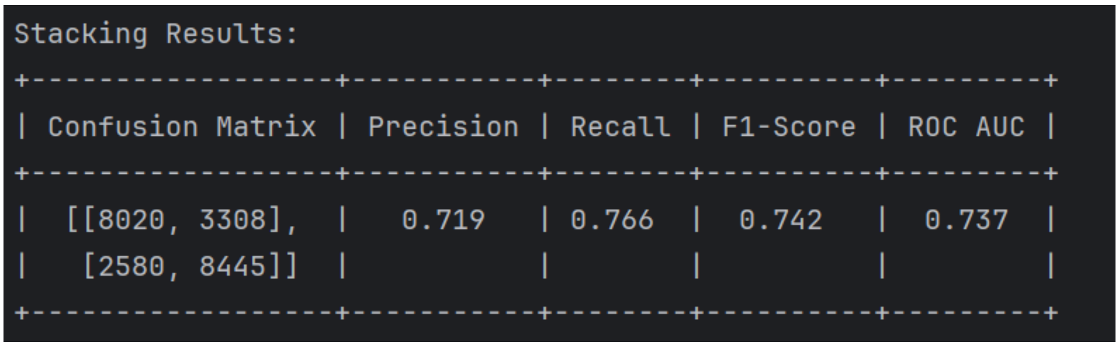
5.6.2 Performance Metrics

Metrics including MSE from bagging and boosting scenarios were analyzed, alongside traditional classification metrics.



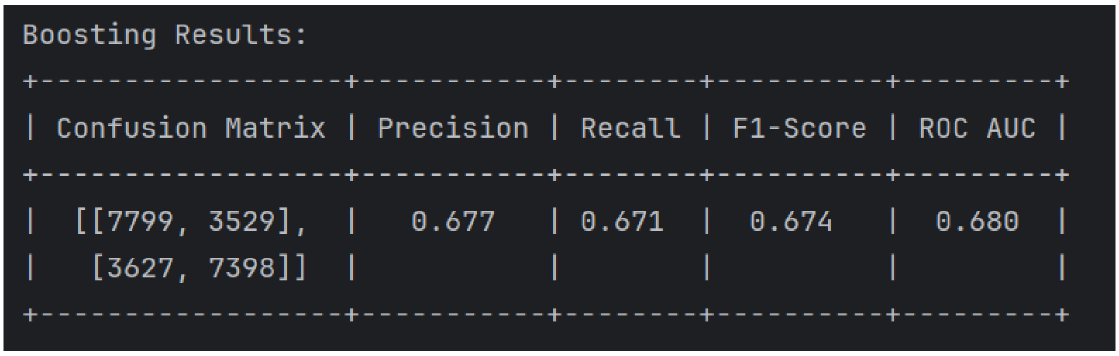
```
Bagging Results:
+------------------+-----------+--------+----------+---------+
| Confusion Matrix | Precision | Recall | F1-Score | ROC AUC |
+------------------+-----------+--------+----------+---------+
|  [[8196, 3132],  |   0.734   | 0.785  |  0.759   |  0.755  |
|   [2365, 8660]]  |           |        |          |         |
+------------------+-----------+--------+----------+---------+
```

Figure 38: Random Forest bagging metrics



```
Stacking Results:
+------------------+-----------+--------+----------+---------+
| Confusion Matrix | Precision | Recall | F1-Score | ROC AUC |
+------------------+-----------+--------+----------+---------+
|  [[8020, 3308],  |   0.719   | 0.766  |  0.742   |  0.737  |
|   [2580, 8445]]  |           |        |          |         |
+------------------+-----------+--------+----------+---------+
```

Figure 39: Random Forest stacking metrics.



```
Boosting Results:
+------------------+-----------+--------+----------+---------+
| Confusion Matrix | Precision | Recall | F1-Score | ROC AUC |
+------------------+-----------+--------+----------+---------+
|  [[7799, 3529],  |   0.677   | 0.671  |  0.674   |  0.680  |
|   [3627, 7398]]  |           |        |          |         |
+------------------+-----------+--------+----------+---------+
```

Figure 40: Random Forest Boosting metrics.

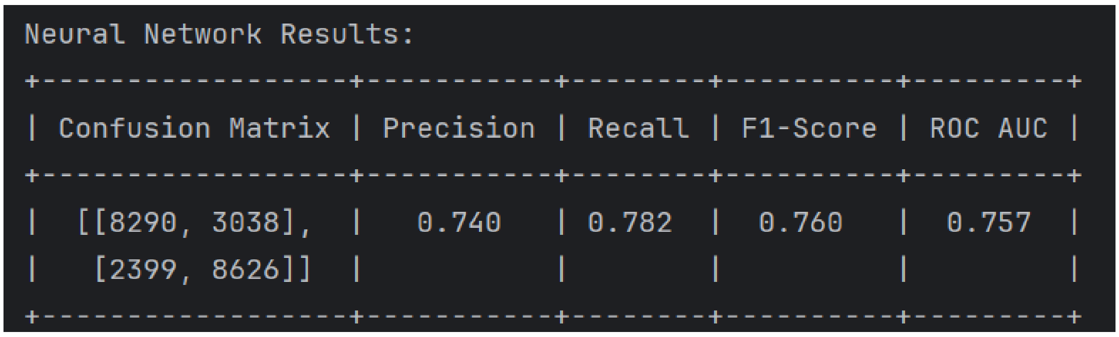
5.7 Neural Network

5.7.1 Network Configuration

A Multi-layer Perceptron (MLP) was configured with multiple layers and neurons to optimize learning capability.

5.7.2 Performance Metrics

Neural Network performance was critically assessed using confusion matrices, ROC curves, and cross-validation scores to ensure the model's generalization.



```
Neural Network Results:
+------------------+-----------+--------+----------+---------+
| Confusion Matrix | Precision | Recall | F1-Score | ROC AUC |
+------------------+-----------+--------+----------+---------+
|  [[8290, 3038],  |   0.740   | 0.782  |  0.760   |  0.757  |
|   [2399, 8626]]  |           |        |          |         |
+------------------+-----------+--------+----------+---------+
```

Figure 41: MLP metrics.

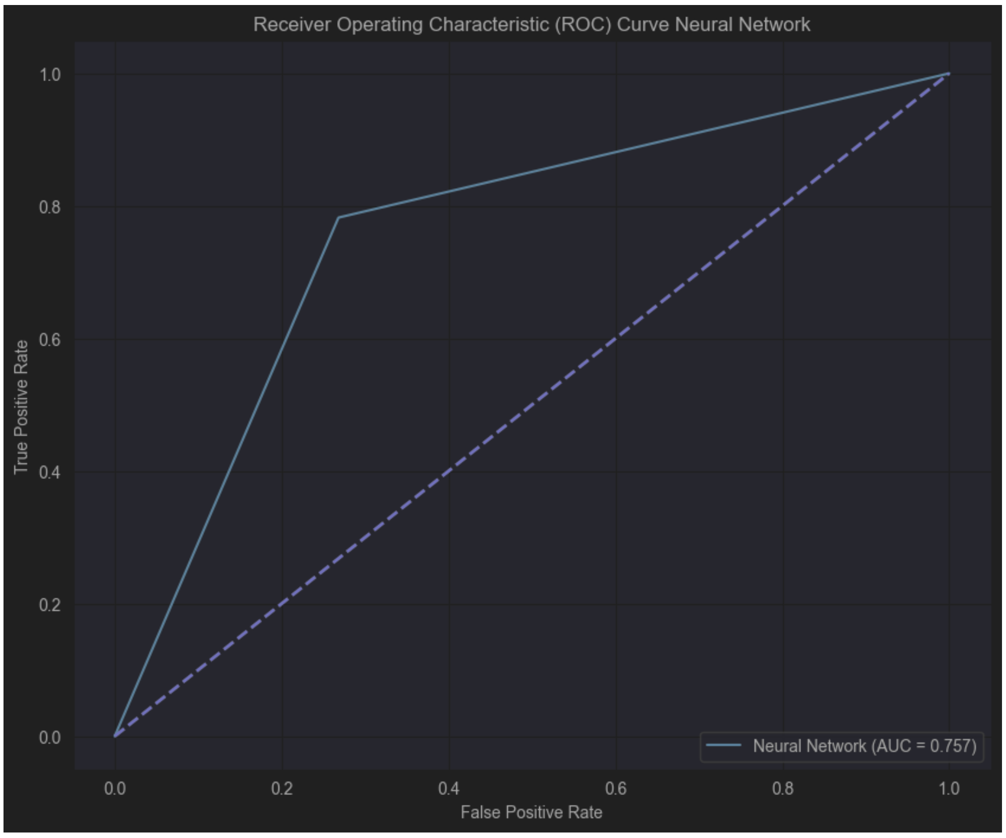


Figure 42: MLP ROC Curve

5.8 Comparative Analysis and Recommendations

A comprehensive comparative analysis was conducted to identify the classifier with the highest overall performance. Recommendations were made based on a combination of performance metrics, computational efficiency, and ease of model interpretation.

Classifier	**Precision**	**Recall**	**F1-Score**	**AUC**	**Confusion Matrix**	**F1-Score (K-fold)**
Decision Tree (Post pruning)	0.75	0.798	0.759	0.75	[[7961, 3367], [2232, 8793]]	0.758
Logistic Regression	0.713	0.773	0.742	0.735	[[7899, 3429],	0.753

					[2503, 8522]]	
KNN	0.706	0.781	0.741	0.732	[[7739, 3589], [2417, 8608]]	0.723
SVM	0.702	0.811	0.752	0.738	[[7538, 3790], [2089, 8936]]	0.759
Naïve Bayes	0.744	0.008	0.016	0.503	[[11298, 30], [10938, 87]]	0.010
Random Forest	0.734	0.785	0.759	0.755	[[8196, 3132], [2365, 8660]]	0.767
Neural Network	0.740	0.782	0.760	0.757	[[8290, 3038], [2399, 8626]]	0.758

Table 2: Comparison of Classifier Performance

6 Phase IV: Clustering and Association [Independent Study]

This phase involves the application of clustering and association rule mining algorithms to analyze the dataset independently. The objective is to identify inherent groupings within the data and discover patterns or rules that could inform further analysis or operational strategies.

6.1 K-means or K-means++ Clustering

6.1.1 Algorithm Implementation

The K-means (or K-means++) algorithm was applied to segment the dataset into clusters. K-means++ was chosen for its efficiency in cluster center initialization, reducing the likelihood of poor cluster formation.

6.1.2 Silhouette Analysis

Silhouette analysis was conducted to determine the optimal number of clusters, *k*. This method assesses the consistency within clusters and is instrumental in validating the homogeneity of data points within the same cluster.

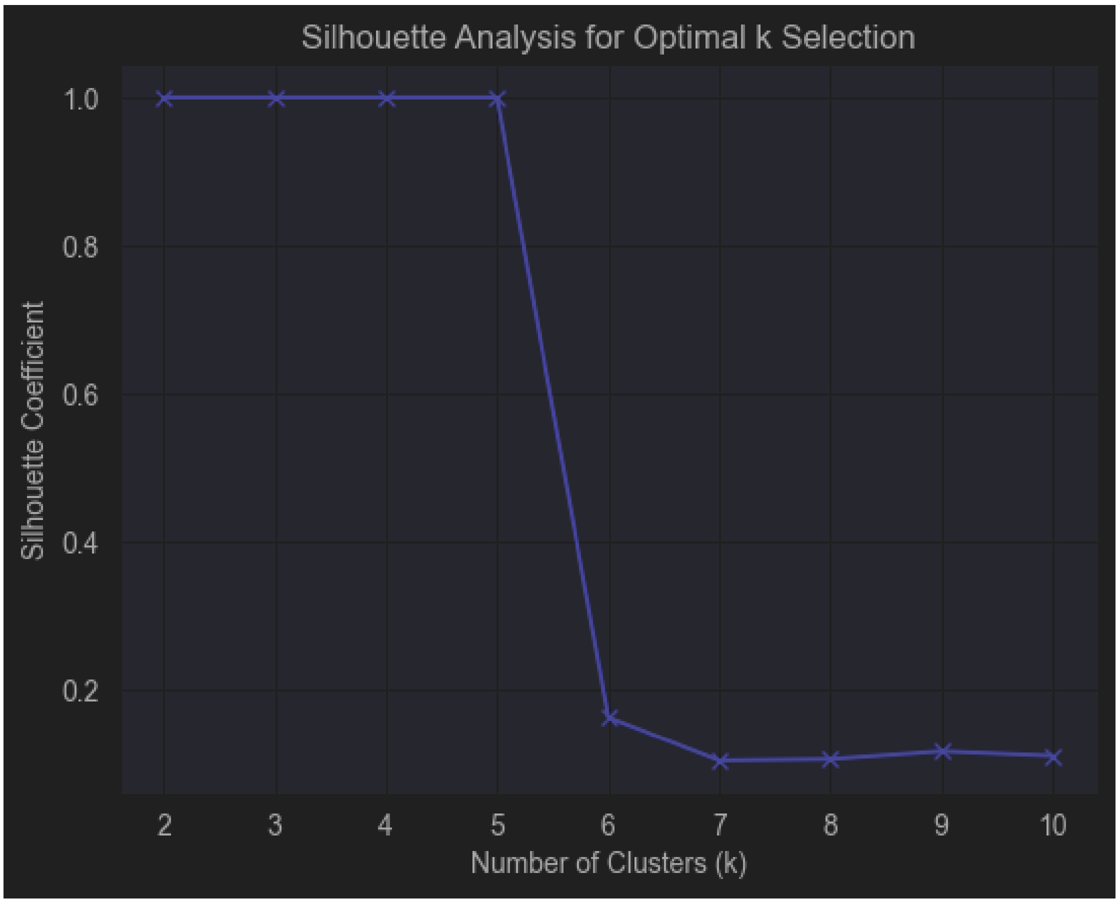


Figure 43:Silhouette Analysis for Optimal K Selection.

6.1.3 Within-Cluster Variation

The within-cluster sum of squares (WCSS) was plotted against the number of clusters to observe the point where the marginal gain in explained variance drops significantly (the elbow method), indicating the optimal *k*.

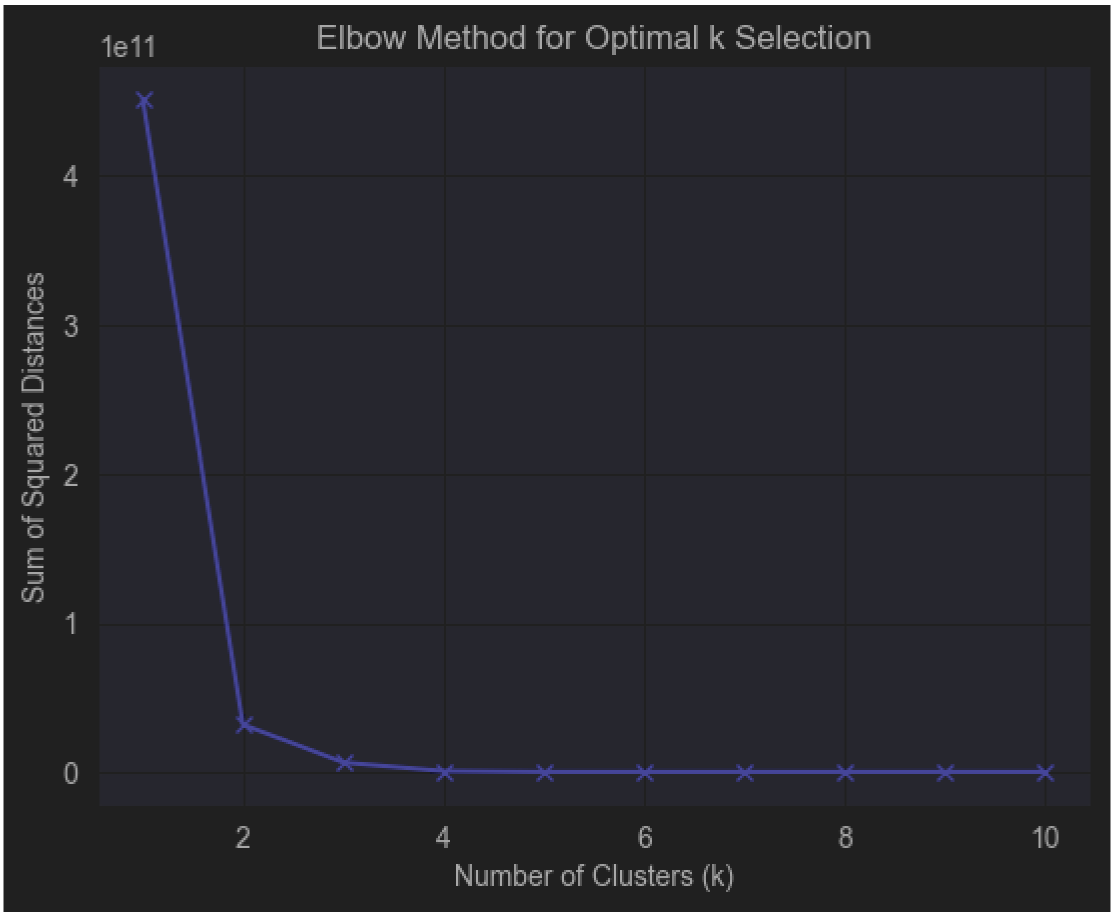


Figure 44: Elbow method for optimal K selection.

6.1.4 Observations

The clustering results were visualized using scatter plots, and clusters were analyzed based on the characteristics of data points within them. Observations about the clusters' composition and their practical implications were discussed.

6.2 DBSCAN Algorithm

6.2.1 Algorithm Implementation

The Density-Based Spatial Clustering of Applications with Noise (DBSCAN) was employed, which is excellent for identifying clusters of varying shapes and sizes in a dataset with noise.

6.2.2 Parameter Optimization

Parameters such as the minimum number of points per cluster and the epsilon distance were tuned to optimize clustering outcomes.

6.2.3 Observations

Clusters identified by DBSCAN were analyzed, and the algorithm's effectiveness in handling outliers was noted. Observations were detailed regarding the spatial distribution of clusters and their densities.

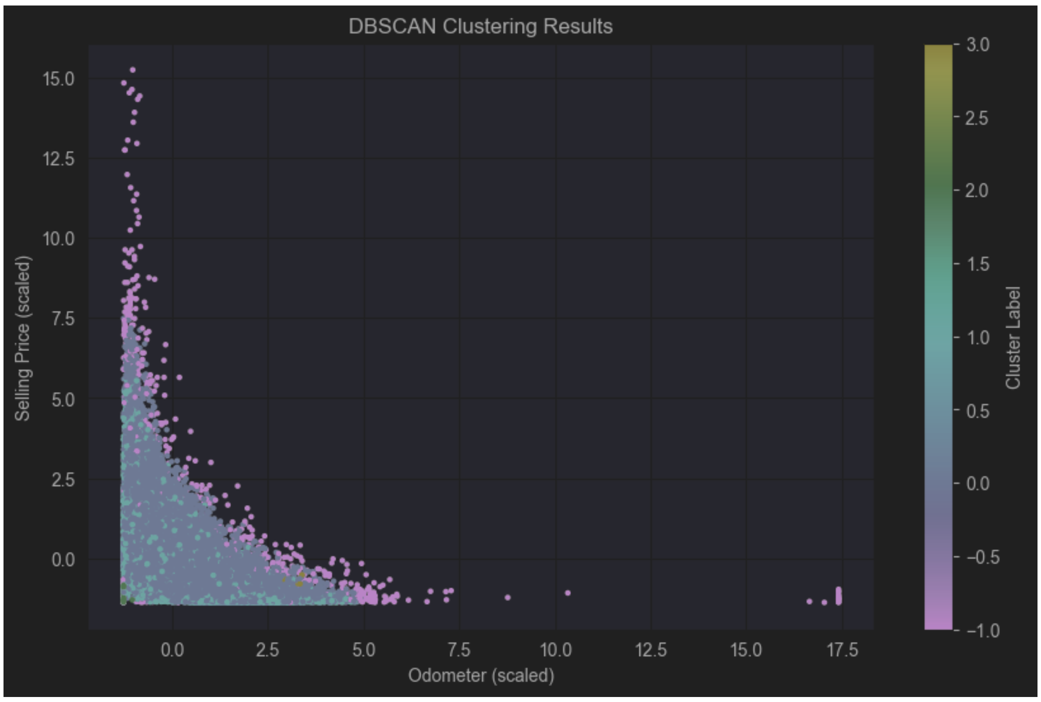


Figure 45: DBSCAN Clustering Results

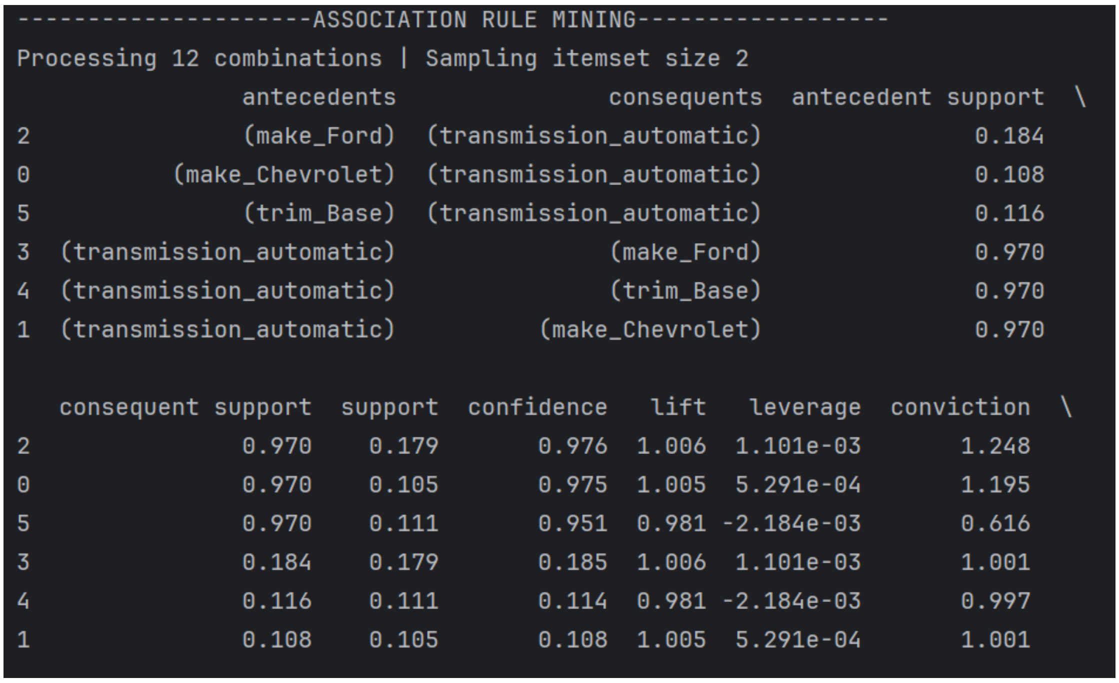
6.3 Apriori Algorithm for Association Rule Mining

6.3.1 Algorithm Implementation

The Apriori algorithm was used to uncover relationships between features in the dataset. This analysis was aimed at finding frequent itemsets and deriving association rules that have significant support and confidence.

6.3.2 Rule Generation

Rules generated by the Apriori algorithm were evaluated based on their support, confidence, and lift. Only rules that exceeded predetermined thresholds were considered significant.



```
---------------------ASSOCIATION RULE MINING------------------
Processing 12 combinations | Sampling itemset size 2
                antecedents                consequents  antecedent support  \
2               (make_Ford)  (transmission_automatic)               0.184
0          (make_Chevrolet)  (transmission_automatic)               0.108
5               (trim_Base)  (transmission_automatic)               0.116
3  (transmission_automatic)                (make_Ford)               0.970
4  (transmission_automatic)                (trim_Base)               0.970
1  (transmission_automatic)           (make_Chevrolet)               0.970

   consequent support  support  confidence   lift   leverage  conviction  \
2               0.970    0.179       0.976  1.006  1.101e-03       1.248
0               0.970    0.105       0.975  1.005  5.291e-04       1.195
5               0.970    0.111       0.951  0.981 -2.184e-03       0.616
3               0.184    0.179       0.185  1.006  1.101e-03       1.001
4               0.116    0.111       0.114  0.981 -2.184e-03       0.997
1               0.108    0.105       0.108  1.005  5.291e-04       1.001
```

Figure 46: Association Rule Mining Using Apriori

6.3.3 Observations

The implications of the most relevant association rules were discussed, particularly in the context of their utility for cross-selling strategies or for segmenting the market based on feature associations.

6.3.4 Conclusion

This phase provided deep insights into the structural patterns of the dataset through clustering and explored potential relationships via association rule mining. The findings from this phase are intended to help in strategic decision-making and to highlight areas for further data-driven investigations.

7 Recommendations

This section synthesizes the insights gleaned from the project and offers strategic recommendations based on the analysis conducted in the previous phases. The recommendations are aimed at enhancing predictive modeling and providing actionable insights for operational or strategic implementations.

7.1 Learnings from the Project

- The project underscored the importance of rigorous data preprocessing and exploratory data analysis as foundational steps in predictive modeling.
- It demonstrated the effectiveness of various classifiers and their applicability depending on the nature and distribution of the data.
- The exercise of tuning hyperparameters and selecting features highlighted the delicate balance between model complexity and performance.

7.2 Best Performing Classifiers

The performance analysis of various classifiers on the dataset has produced insightful outcomes, especially in terms of precision, recall, F1-score, AUC (Area Under the Curve), and cross-validation results. Here's a summary of each classifier's performance:

- **Decision Tree (Post pruning):** This model shows a strong balance between recall and precision, resulting in an F1-score of 0.759 and an AUC of 0.75, indicating good model performance. The cross-validation F1-score of 0.758 confirms the model's stability across different subsets of data.

- **Logistic Regression:** The logistic regression model presents a slightly lower performance compared to the decision tree, with an F1-score of 0.742 and an AUC of 0.735. Its precision is slightly less but still competitive, and the F1-score in cross-validation is 0.753, suggesting consistency in performance.
- **KNN (K-Nearest Neighbors):** KNN shows a comparable recall to the logistic regression but has a lower precision which impacts its F1-score slightly, landing it at 0.741 with an AUC of 0.732. Its cross-validation score is the lowest among the competitive models at 0.723, indicating potential variability in performance across different data splits.
- **SVM (Support Vector Machine):** SVM performs notably well in terms of recall, which is the highest among the classifiers at 0.811. This high recall contributes to an F1-score of 0.752 and an AUC of 0.738. Its performance is robust as evidenced by a cross-validation F1-score of 0.759.
- **Naïve Bayes:** This model, while having a reasonable precision, shows a very low recall of 0.008, which drastically affects its F1-score, making it practically negligible at 0.016. The AUC is barely over 0.5, indicating performance close to a random classifier. The model is significantly underperforming in this scenario.
- **Random Forest:** Exhibiting strong performance, the Random Forest classifier has an F1-score and AUC closely mirroring those of the Decision Tree, both at 0.759 and 0.755, respectively. Its cross-validation score is the highest at 0.767, highlighting its robustness and effectiveness across different data segments.
- **Neural Network:** The neural network shows balanced precision and recall, leading to an F1-score of 0.760 and an AUC of 0.757. Its performance is consistent across different folds of data, as shown by the cross-validation F1-score of 0.758.

Conclusion:

Overall, the **Random Forest** and **Neural Network** models demonstrate the most robust and consistent performance across all metrics, making them the preferable choices for classification tasks within this dataset. These models not only provide high F1-scores and AUCs but also maintain performance stability across different subsets of data, as evidenced by their cross-validation scores. The SVM also presents itself as a strong contender, especially in scenarios where maximizing recall is crucial. Conversely, the **Naïve Bayes** classifier may require reconsideration or significant adjustments due to its inadequate performance in this specific context.

7.3 Improvements in Classification Performance

- Future work could explore more advanced ensemble techniques that might provide better generalization over the dataset.
- Increasing the dataset size or incorporating more diverse data sources could help in enhancing the model's learning and prediction accuracy.
- Continuous tuning of hyperparameters using automated techniques like grid search or random search could yield better results.

7.4 Features Associated with the Target Variable

- Analysis revealed specific features that are strongly associated with the target variable, aiding in understanding the drivers of the dependent variable.
- For instance, if year and odometer readings are identified as key predictors, these should be considered in pricing models or customer segmentation analyses.

7.5 Cluster Analysis

- The optimal number of clusters identified during the cluster analysis provides insights into market segmentation and customer behavior patterns.
- The characteristics of each cluster can be used to tailor marketing strategies or customize product offerings.

8 Appendix

8.1 Python Code

The appendix includes all the Python code developed for this project. This includes scripts for data cleaning, model building, evaluation, and any additional analyses conducted. The code is well-commented to ensure that it is understandable and reusable.

```
import pandas as pd
import numpy as np
from mlxtend.preprocessing import TransactionEncoder
from sklearn.ensemble import RandomForestRegressor
from sklearn.preprocessing import LabelEncoder
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from scipy.linalg import svd
from sklearn.preprocessing import StandardScaler
import seaborn as sns
from statsmodels.stats.outliers_influence import
variance_inflation_factor
import statsmodels.api as sm
from sklearn.metrics import mean_squared_error, silhouette_score
from sklearn.model_selection import GridSearchCV, train_test_split,
StratifiedKFold
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import confusion_matrix, precision_score,
recall_score, f1_score, roc_curve, auc
from sklearn.cluster import DBSCAN
from mlxtend.frequent_patterns import apriori, association_rules
from sklearn.tree import plot_tree
from sklearn.metrics import accuracy_score, roc_auc_score
from sklearn.cluster import KMeans
from sklearn.ensemble import BaggingClassifier, StackingClassifier,
AdaBoostClassifier
from sklearn.neighbors import KNeighborsClassifier
from prettytable import PrettyTable
import warnings
from imblearn.over_sampling import SMOTE

#%%
RED = "\033[91m"
GREEN = "\033[92m"
BLUE = "\033[94m"
RESET = "\033[0m"
```

```
warnings.filterwarnings('ignore')

pd.set_option('display.precision', 3)

print(GREEN + "\n ----------------------- Phase I: Feature
Engineering & EDA : -----------------------\n" + GREEN)

car_prices_df = pd.read_csv("car_prices.csv")
print("info: \n", car_prices_df.info())
print("Describe: \n", car_prices_df.describe())
print("shape of the database ", car_prices_df.shape)
print("columns : ", car_prices_df.columns)

missing_values = car_prices_df.isna().sum()
print("missing values in each column:\n",
missing_values.sort_values(ascending=False))
print("total number of missing values in the dataset:",
car_prices_df.isna().sum().sum())

missing_values_per_column = car_prices_df.isna().sum()
print("percentage of missing values per column:\n",
(missing_values_per_column / car_prices_df.shape[0]) * 100)
car_prices_df = car_prices_df.dropna(subset=['sellingprice'])

print("missing values:", car_prices_df.isna().sum())
for column in
car_prices_df.select_dtypes(include=['number']).columns:
    car_prices_df[column].fillna(car_prices_df[column].mean(),
inplace=True)

for column in
car_prices_df.select_dtypes(exclude=['number']).columns:
    print("mode of the column", car_prices_df[column].mode()[0])
    car_prices_df[column].fillna(car_prices_df[column].mode()[0],
inplace=True)

car_prices_df['saledate'] = pd.to_datetime(car_prices_df['saledate'],
errors='coerce', utc=True)
car_prices_df['saleyear'] = car_prices_df['saledate'].dt.year
car_prices_df['saleyear'] = car_prices_df['saleyear'].fillna(-
1).astype(int)

print(car_prices_df['saleyear'].head())

car_prices_df = car_prices_df.drop('saledate', axis=1)
print("total number of missing values after filling: ",
car_prices_df.isna().sum())

duplicate_rows = car_prices_df[car_prices_df.duplicated()]
print(duplicate_rows.shape)
print("there is no duplicate data in the given dataset")
```

```
#%%
print("\n------------START OF FEATURE SELECTION & DIMENSIONALITY
REDUCTION--------------\n")
label_encoders = {}
data = car_prices_df.copy()
for column in data.select_dtypes(include=['object']).columns:
    le = LabelEncoder()
    data[column] = le.fit_transform(data[column].astype(str))
    label_encoders[column] = le

encoded_data = data.copy()

print(data.head())

print("\n------------------------STANDARDIZE THE DATA----------------
-----\n")

scalar = StandardScaler()
data_scaled = scalar.fit_transform(data)
data_scaled_df = pd.DataFrame(data=data_scaled, columns=data.columns)
data.update(data_scaled_df)
print(data.head())

print("\n-----------Dimensionality reduction/feature selection-------
--\n")
X = data.drop(['sellingprice'], axis=1)
y = data['sellingprice']

pca = PCA()
X_pca = pca.fit_transform(X)

print(X.columns)
cumulative_variance_ratio = np.cumsum(pca.explained_variance_ratio_)
n_features_95 = np.argmax(cumulative_variance_ratio >= 0.95) + 1

plt.figure(figsize=(10, 6))
plt.plot(np.arange(1, len(cumulative_variance_ratio) + 1),
cumulative_variance_ratio, marker='o', linestyle='-')
plt.xlabel('Number of Features')
plt.ylabel('Cumulative Explained Variance Ratio')
plt.title('Cumulative Explained Variance Ratio vs Number of
Features')
plt.grid(True)
plt.axvline(x=n_features_95, color='r', linestyle='--')
plt.axhline(y=0.95, color='g', linestyle='--')

plt.annotate(f'({n_features_95}, 0.95)', xy=(n_features_95, 0.95),
xytext=(n_features_95 + 5, 0.90),
             arrowprops=dict(facecolor='black', shrink=0.05))

plt.show()
```

```
print("Number of features needed for explaining more than 95% of the
dependent variance:", n_features_95)

rf = RandomForestRegressor(n_estimators=8, max_depth=10,
random_state=5805)

rf.fit(X, y)
importances = rf.feature_importances_
feature_names = X.columns
feature_importance_dict = dict(zip(feature_names, importances))
sorted_feature_importance = sorted([[key,
feature_importance_dict[key]] for key in feature_importance_dict],
                                   key=lambda x: x[1], reverse=True)
print("feature_importance_dict: ", sorted_feature_importance)

features = list(feature_importance_dict.keys())
importances = list(feature_importance_dict.values())

plt.figure(figsize=(10, 8))
plt.barh(features, importances, color='skyblue')
plt.xlabel('Importance Score')
plt.title('Feature Importance from Random Forest')
plt.show()

print("\n----------------- START OF SINGLE VALUE DECOMPOSITION ------
-------------\n")

numerical_cols = X.select_dtypes(include=[np.number])

U, s, Vt = svd(numerical_cols, full_matrices=False)

print(s, U.shape, Vt.shape)

print("\n----------------- END OF SINGLE VALUE DECOMPOSITION --------
-----------\n")

print("\n----------------- START OF VIF -------------------\n")

X_vif = X.select_dtypes(include='number').assign(const=1)

vif_data = pd.DataFrame({
    'Variable': X_vif.columns,
    'VIF': [variance_inflation_factor(X_vif.values, i) for i in
range(X_vif.shape[1])]
})

vif_sorted = vif_data.sort_values(by='VIF', ascending=False)
print(vif_sorted)
plt.figure(figsize=(10, 8))

plt.barh(vif_data['Variable'], vif_data['VIF'], color='skyblue')
plt.xlabel('VIF Score')
plt.title('VIF method')
plt.show()
print("\n------------------ END OF VIF -------------------\n")

final_selected_features = ['year', 'make', 'model', 'trim',
'condition', 'odometer', 'sellingprice', 'mmr']

print("\n------------------ START OF ANOMALY REMOVAL/OUTLIER DETECTION
(Z-SCORE) -------------------\n")

cleaned_carprices_df = data[final_selected_features]
z_scores = np.abs(
    (cleaned_carprices_df['sellingprice'] -
cleaned_carprices_df['sellingprice'].mean()) / cleaned_carprices_df[
        'sellingprice'].std())
data_cleaned_z = cleaned_carprices_df[z_scores < 3]
print("df shapes after anomaly deletion: ",
cleaned_carprices_df.shape, data_cleaned_z.shape)
z_scores = np.abs((cleaned_carprices_df['sellingprice'] -
cleaned_carprices_df['sellingprice'].mean()) /
cleaned_carprices_df['sellingprice'].std())

outliers = z_scores >= 3

plt.figure(figsize=(10, 6))
plt.scatter(cleaned_carprices_df.index,
cleaned_carprices_df['sellingprice'], c='blue', label='Non-outliers')
plt.scatter(cleaned_carprices_df[outliers].index,
cleaned_carprices_df[outliers]['sellingprice'], c='red',
label='Outliers')
plt.title('Outliers in Selling Price Using Z-score')
plt.xlabel('Index')
plt.ylabel('Selling Price')
plt.legend()
plt.show()

print("\n------------------ END OF ANOMALY REMOVAL/OUTLIER DETECTION
(Z-SCORE)-------------------\n")

print("\n------------------ START OF COVARIANCE MATRIX ---------------
----\n")

covariance_matrix = encoded_data[final_selected_features].cov()
plt.figure(figsize=(10, 10))
sns.heatmap(covariance_matrix, annot=True, fmt=".2f",
cmap='coolwarm', cbar=True)
plt.title('Heatmap of the Sample Covariance Matrix')
plt.show()
print("\n------------------ END OF COVARIANCE MATRIX -----------------
--\n")

print("\n------------------ START OF CORRELATION MATRIX --------------
-----\n")
correlation_matrix = data_cleaned_z.corr()

plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, fmt=".2f",
cmap='coolwarm', center=0)
plt.title('Heatmap of the Sample Pearson Correlation Coefficients
Matrix')
plt.show()

print("\n----------------- END OF CORRELATION MATRIX ----------------
---\n")
#%%
print(BLUE + "Phase II: Regression Analysis" + BLUE)
print("\n----------------- START OF MULTIPLE LINEAR REGRESSION ------
-------------\n")

X = data_cleaned_z.drop('sellingprice', axis=1)
y = data_cleaned_z['sellingprice']

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=5805)

X_train = sm.add_constant(X_train)
X_test = sm.add_constant(X_test)

model = sm.OLS(y_train, X_train).fit()

model_summary = model.summary()
print(model_summary)

predictions = model.predict(X_test)
mse = mean_squared_error(y_test, predictions)
print(f"MSE: {mse}")

predicted_values = model.predict(X_test)

plt.figure(figsize=(10, 6))
plt.scatter(y_train.index, y_train, label='Trained')
plt.scatter(y_test.index, y_test, label='Actual')
plt.scatter(y_test.index, predicted_values, label='Predicted')
plt.title(' Multiple linear Regression Actual vs Predicted')
plt.xlabel('Index')
plt.ylabel('sellingprice')
plt.legend()
plt.show()


print("\n----------------- END OF MULTIPLE LINEAR REGRESSION --------
-----------\n")
```

```
#%%
def backward_stepwise_regression(X_train, X_test, y_train, y_test,
threshold=0.00001):
    selected_features = list(X_train.columns)
    all_results = []

    iteration = 1

    while len(selected_features) > 0:
        X_train_selected = X_train[selected_features]
        X_train_selected = sm.add_constant(X_train_selected)
        X_test_selected = X_test[selected_features]
        X_test_selected = sm.add_constant(X_test_selected)

        model = sm.OLS(y_train, X_train_selected).fit()
        p_values = model.pvalues[1:]

        eliminated_feature = None
        aic = model.aic
        bic = model.bic
        adj_r2 = model.rsquared_adj
        p_value = None

        if p_values.max() > threshold:
            eliminated_feature = p_values.idxmax()
            selected_features.remove(eliminated_feature)
            p_value = p_values.max()
        else:
            break

        results = {'Eliminated Feature': eliminated_feature,
                   'AIC': aic, 'BIC': bic, 'Adjusted R-squared':
adj_r2,
                   'P-value': p_value,
                   'Selected Features': selected_features.copy()}
        all_results.append(results)

        print("Model summary at iteration {} ".format(iteration))
        print(model.summary())
        iteration += 1

    final_selected_features = selected_features
    return all_results, final_selected_features


all_results, final_selected_features =
backward_stepwise_regression(X_train, X_test, y_train, y_test)

results_dfs = []
for i, results in enumerate(all_results):
    results_df = pd.DataFrame([results])
    results_df['Iteration'] = i + 1
    results_dfs.append(results_df)

results_df = pd.concat(results_dfs, ignore_index=True)

print("Process and Justification for Backward Stepwise Regression:")
print(results_df)
print("\nFinal Selected Features:", final_selected_features)

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=5805)

final_model = sm.OLS(y_train, X_train).fit()
predicted_values = final_model.predict(X_test)

plt.figure(figsize=(10, 6))
plt.scatter(y_test.index, y_test, label='Actual')
plt.scatter(y_test.index, predicted_values, label='Predicted')
plt.title(' Multiple linear Regression Actual vs Predicted')
plt.xlabel('Index')
plt.ylabel('sellingprice')
plt.legend()
plt.show()

mse = mean_squared_error(y_test, predicted_values)

print("Mean Squared Error (MSE) BACKWARD STEPWISE REGRESSION :", mse)

print("\n----------------- END OF MULTIPLE LINEAR REGRESSION
(BACKWARD STEPWISE REGRESSION) --------------------\n")
#%%


print("\n-----------------Phase III: Classification Analysis:---------
--------\n")

print(car_prices_df.info())

downsampled_df = data.sample(frac=0.2, replace=False, random_state=1)

X = downsampled_df.drop(['condition'], axis=1)
y = downsampled_df['condition']

threshold_value = y.median()
y = pd.cut(y, bins=[-float('inf'), threshold_value, float('inf')],
labels=[0, 1])

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, stratify=y, random_state=5805
)

print(y.value_counts())
```

```
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(12, 5))
# Plotting class distribution before SMOTE
axes[0].bar([0, 1], y_train.value_counts(), color=['blue', 'red'])
axes[0].set_title('Before SMOTE')
axes[0].set_xlabel('Condition')
axes[0].set_ylabel('Count')
axes[0].set_xticks([0, 1])

# Applying SMOTE to balance the dataset
smote = SMOTE(random_state=5805)
X_train, y_train = smote.fit_resample(X_train, y_train)

# Printing class distribution after applying SMOTE
print(y_train.value_counts())

# Plotting class distribution after SMOTE
axes[1].bar([0, 1], y_train.value_counts(), color=['blue', 'red'])
axes[1].set_title('After SMOTE')
axes[1].set_xlabel('Condition')
axes[1].set_ylabel('Count')
axes[1].set_xticks([0, 1])

plt.tight_layout()
plt.show()
def evaluate_model(preds, y_test, model_name, dt_best):
    print(f'{model_name} Results:')
    table = PrettyTable()

    # Adding headers for all metrics
    table.field_names = ["Confusion Matrix", "Precision", "Recall", 
"F1-Score", "ROC AUC"]

    # Compute metrics
    precision = precision_score(y_test, preds)
    recall = recall_score(y_test, preds)
    f1 = f1_score(y_test, preds)

    # Compute ROC AUC
    fpr, tpr, _ = roc_curve(y_test, preds)
    roc_auc = auc(fpr, tpr)

    # Format confusion matrix
    cm = confusion_matrix(y_test, preds)
    cm_str = f"[[{cm[0, 0]}, {cm[0, 1]}],\n [{cm[1, 0]}, {cm[1, 
1]}]]"

    # Add rows for metrics
    table.add_row([cm_str, f"{precision:.3f}", f"{recall:.3f}", 
f"{f1:.3f}", f"{roc_auc:.3f}"])
    print(table)

    plt.figure(figsize=(10, 8))
    plt.plot(fpr, tpr, label=f'{model_name} (AUC = {roc_auc:.3f})')
    plt.plot([0, 1], [0, 1], color="navy", lw=2, linestyle="--")
    plt.xlabel("False Positive Rate")
    plt.ylabel("True Positive Rate")
    plt.title("Receiver Operating Characteristic (ROC) Curve
{}".format(model_name))
    plt.legend(loc="lower right")
    plt.show()

    skf = StratifiedKFold(n_splits=5, random_state=42, shuffle=True)
    scores = []
    for train_idx, test_idx in skf.split(X, y):
        X_train_fold, X_test_fold = X.iloc[train_idx],
X.iloc[test_idx]
        y_train_fold, y_test_fold = y.iloc[train_idx],
y.iloc[test_idx]
        model = dt_best.fit(X_train_fold, y_train_fold)
        preds = model.predict(X_test_fold)
        scores.append(f1_score(y_test_fold, preds))

    print(f'Stratified K-fold Cross-Validation F1-Score:
{np.mean(scores):.3f}')
    print('--------------------------------')
#%%
dt_model = DecisionTreeClassifier(random_state=5805)
dt_model.fit(X_train, y_train)

y_pred = dt_model.predict(X_test)
evaluate_model(y_pred, y_test, "Decision Tree Classifier", dt_model)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
importance = pd.Series(dt_model.feature_importances_,
index=X.columns)
print("Feature Importance:", importance)
print("feature to be removed :", importance.idxmin())

plt.figure(figsize=(8, 6))
sns.barplot(x=importance, y=importance.index)
plt.title("Feature Importance")
plt.xlabel("Importance Score")
plt.ylabel("Features")
plt.show()

selected_dt_features = importance.sort_values()[::-1][:6]


#%%
print(selected_dt_features.index)

X2 = downsampled_df[selected_dt_features.index]

X_train, X_test, y_train, y_test = train_test_split(
    X2, y, test_size=0.2, stratify=y, random_state=5805
)
dt_model2 = DecisionTreeClassifier(random_state=5805)
dt_model2.fit(X_train, y_train)

y_pred_dt2 = dt_model2.predict(X_test)
evaluate_model(y_pred_dt2, y_test, "Decision Tree Classifier_2", 
dt_model2)
#%%
tuned_parameters = [{"max_depth": range(2, 10, 1),
                     "min_samples_leaf": range(2, 10, 1),
                     "min_samples_split": range(2, 10, 1),
                     'max_features': [None, 'sqrt', 'log2'],
                     'splitter': ['best', 'random'],
                     'criterion': ['gini', 'entropy']}]

dt_model_pruned = DecisionTreeClassifier(random_state=5805, 
ccp_alpha=0.09)

grid_search = GridSearchCV(estimator=dt_model_pruned, 
param_grid=tuned_parameters, cv=5, n_jobs=-1)

grid_search.fit(X_train, y_train)
print(grid_search.best_params_)
dt_model_pre_pruned = grid_search.best_estimator_
dt_model_pre_pruned.fit(X_train, y_train)
y_pre_predicted = dt_model_pre_pruned.predict(X_test)
print(f'Accuracy of pre tuned tree {accuracy_score(y_test, 
y_pre_predicted):.3f}')

plt.figure(figsize=(20, 12))
plot_tree(dt_model_pre_pruned, rounded=True, filled=True)
plt.show()

path = dt_model.cost_complexity_pruning_path(X_train, y_train)
print(path)

ccp_alphas, impurities = path.ccp_alphas, path.impurities
clfs = []
for ccp_alpha in ccp_alphas:
    clf = DecisionTreeClassifier(random_state=5806, 
ccp_alpha=ccp_alpha)
    clf.fit(X_train, y_train)
    clfs.append(clf)

acc_scores = [accuracy_score(y_test, clf.predict(X_test)) for clf in 
clfs]
max_acc_score_index = acc_scores.index(max(acc_scores))
plt.figure(figsize=(10, 6))
plt.grid()
plt.plot(ccp_alphas[:-1], acc_scores[:-1])
plt.xlabel("effective alpha")
plt.ylabel("Accuracy scores")
plt.show()

post_pruning_clf = clfs[max_acc_score_index]
post_pruning_clf.fit(X_train, y_train)
y_post_predicted = post_pruning_clf.predict(X_test)
post_pruning_acc = accuracy_score(y_test, y_post_predicted)

print("accuracy for post pruning : ", round(post_pruning_acc, 3))
plot_tree(decision_tree=post_pruning_clf, rounded=True, filled=True)

pre_confusion_matrix = confusion_matrix(y_test, y_pre_predicted)
pre_accuracy = accuracy_score(y_test, y_pre_predicted)
pre_recall = recall_score(y_test, y_pre_predicted)
pre_roc_auc = roc_auc_score(y_test, y_pre_predicted)
print(" pre metrics ", pre_confusion_matrix, pre_recall,
pre_accuracy, pre_roc_auc)

post_confusion_matrix = confusion_matrix(y_test, y_post_predicted)
post_accuracy = accuracy_score(y_test, y_post_predicted)
post_recall = recall_score(y_test, y_post_predicted)
post_roc_auc = roc_auc_score(y_test, y_post_predicted)
metrics_data = {
    'Pruning Method': ['Pre-Pruned', 'Post-Pruned'],
    'Accuracy': [pre_accuracy, post_accuracy],
    'Confusion Matrix': [pre_confusion_matrix,
post_confusion_matrix],
    'Recall': [pre_recall, post_recall],
    'AUC': [pre_roc_auc, post_roc_auc]
}

metrics_df = pd.DataFrame(metrics_data)

print(metrics_df.head())

logreg_model = LogisticRegression(random_state=5805)
logreg_model.fit(X_train, y_train)
y_logistic_pred = logreg_model.predict(X_test)
accuracy = accuracy_score(y_test, y_logistic_pred)
print("Logistic Regression Accuracy:", round(accuracy, 3))

logistic_auc = roc_auc_score(y_test, y_logistic_pred)
dt_pre_auc = roc_auc_score(y_test, y_pre_predicted)
dt_post_auc = roc_auc_score(y_test, y_post_predicted)

fpr_lr, tpr_lr, _ = roc_curve(y_test, y_logistic_pred)
fpr_dt_pre, tpr_dt_pre, _ = roc_curve(y_test, y_pre_predicted)
fpr_dt_post, tpr_dt_post, _ = roc_curve(y_test, y_post_predicted)

plt.figure(figsize=(8, 6))
plt.plot(fpr_lr, tpr_lr, label=f'Logistic Regression (AUC =
{logistic_auc:.3f})')
```

```
plt.plot(fpr_dt_pre, tpr_dt_pre, label=f'Decision Tree with pre
pruning (AUC = {dt_pre_auc:.3f})')
plt.plot(fpr_dt_post, tpr_dt_post, label=f'Decision Tree with post
pruning (AUC = {dt_post_auc:.3f})')

plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend()
plt.grid(True)
plt.show()

logistic_confusion_matrix = confusion_matrix(y_test, y_logistic_pred)
logistic_accuracy = accuracy_score(y_test, y_logistic_pred)
logistic_recall = recall_score(y_test, y_logistic_pred)
logistic_auc = roc_auc_score(y_test, y_logistic_pred)
metrics_data = {
    'Pruning Method': ['Pre-Pruned', 'Post-Pruned', 'Logistic'],
    'Accuracy': [pre_accuracy, post_accuracy, logistic_accuracy],
    'Confusion Matrix': [pre_confusion_matrix, post_confusion_matrix,
logistic_confusion_matrix],
    'Recall': [pre_recall, post_recall, logistic_recall],
    'AUC': [pre_roc_auc, post_roc_auc, logistic_auc]
}
metrics_df = pd.DataFrame(metrics_data)

evaluate_model(y_logistic_pred, y_test, "Logistic Regression",
logreg_model)
evaluate_model(y_pre_predicted, y_test, "Pre Pruned Decision Tree",
dt_model_pre_pruned)
evaluate_model(y_post_predicted, y_test, "Post Pruned Decision Tree",
post_pruning_clf)

print(metrics_df)
#%%
print("\n--------------------------KNN----------------------\n")

error = []

for i in range(1, 40):
    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(X_train, y_train)
    pred_i = knn.predict(X_test)
    error.append(np.mean(pred_i != y_test))

plt.figure(figsize=(12, 6))
plt.plot(range(1, 40), error, color='red', linestyle='dashed',
marker='o', markerfacecolor='blue', markersize=10)

plt.title('Error Rate for Different K Values')
plt.xlabel('K')
plt.ylabel('Error')
plt.show()

knn = KNeighborsClassifier(n_neighbors=error.index(min(error)) + 1)

knn.fit(X_train, y_train)
pred = knn.predict(X_test)

print('Minimum error:', min(error))
print('Optimum K:', error.index(min(error)) + 1)
evaluate_model(pred, y_test, 'KNN', knn)

#%%
print("\n------------------------------SVM--------------------------
-\n")

parameters = {
    'kernel': ['linear', 'rbf', 'poly'],
}
stratified_kfold = StratifiedKFold(n_splits=5, shuffle=True,
random_state=5805)

# Perform grid search with cross-validation
svm_classifier = SVC(probability=True)
grid_search = GridSearchCV(svm_classifier, parameters,
cv=stratified_kfold, n_jobs=-1)
grid_search.fit(X_train, y_train)
print("Best Hyperparameters:", grid_search.best_params_)
svm = grid_search.best_estimator_
svm.fit(X_train, y_train)
y_svm_pred = svm.predict(X_test)
evaluate_model(y_svm_pred, y_test, 'SVM', svm)



#%%
nb_model = GaussianNB()
nb_model.fit(X_train, y_train)
y_nb_pred = nb_model.predict(X_test)
evaluate_model(y_nb_pred, y_test, 'Naive Bayes', nb_model)


#%%
# Bagging
bagging_clf = BaggingClassifier(DecisionTreeClassifier(),
n_estimators=100, random_state=5805)
bagging_clf.fit(X_train, y_train)
y_bagging_pred = bagging_clf.predict(X_test)
evaluate_model(y_bagging_pred, y_test, 'Bagging', bagging_clf)


#%%
```

```
# Stacking
estimators = [('dt', DecisionTreeClassifier()), ('lr',
LogisticRegression())]
stacking_clf = StackingClassifier(estimators=estimators)
stacking_clf.fit(X_train, y_train)
y_stacking_pred = stacking_clf.predict(X_test)
evaluate_model(y_stacking_pred, y_test, 'Stacking', stacking_clf)


#%%
# Boosting
boosting_clf = AdaBoostClassifier(DecisionTreeClassifier(),
n_estimators=100, random_state=5805)
boosting_clf.fit(X_train, y_train)
y_boosting_pred = boosting_clf.predict(X_test)
evaluate_model(y_boosting_pred, y_test, 'Boosting', boosting_clf)


#%%
nn_model = MLPClassifier(hidden_layer_sizes=(100, 50),
activation='relu', solver='adam', random_state=5805)
nn_model.fit(X_train, y_train)
y_nn_pred = nn_model.predict(X_test)
evaluate_model(y_nn_pred, y_test, 'Neural Network', nn_model)
#%%
sum_of_squared_distances = []

for k in range(1, 11):
    kmeans = KMeans(n_clusters=k, init='k-means++',
random_state=5805)
    kmeans.fit(X)
    sum_of_squared_distances.append(kmeans.inertia_)

plt.plot(range(1, 11), sum_of_squared_distances, 'bx-')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('Sum of Squared Distances')
plt.title('Elbow Method for Optimal k Selection')
plt.show()
#%%
silhouette_coefficients = []

for k in range(2, 11):
    kmeans = KMeans(n_clusters=k, init='k-means++',
random_state=5805)
    kmeans.fit(X)

    silhouette_coefficients.append(silhouette_score(X,
kmeans.labels_))

plt.plot(range(2, 11), silhouette_coefficients, 'bx-')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('Silhouette Coefficient')
plt.title('Silhouette Analysis for Optimal k Selection')
plt.show()
#%%
print("\n---------------DBSCAN-------------\n")

data = car_prices_df.copy().sample(frac=0.2)
data_clean = data.dropna(subset=['condition', 'odometer', 'mmr',
'sellingprice', 'year'])
features = data_clean[['year', 'condition', 'odometer', 'mmr',
'sellingprice']]
features_scaled = StandardScaler().fit_transform(features)
dbscan = DBSCAN(eps=0.5, min_samples=10)
clusters = dbscan.fit_predict(features_scaled)
print(clusters)
plt.figure(figsize=(10, 6))
plt.scatter(features_scaled[:, 2], features_scaled[:, 4], c=clusters,
cmap='viridis', s=5)
plt.xlabel('Odometer (scaled)')
plt.ylabel('Selling Price (scaled)')
plt.title('DBSCAN Clustering Results')
plt.colorbar(label='Cluster Label')
plt.show()
print("\n---------------END OF DBSCAN-------------\n")

#%%
print("\n---------------------ASSOCIATION RULE MINING----------------
--\n")

df_apriori_filtered = car_prices_df[['make', 'model', 'trim',
'transmission']].sample(frac=0.05)
df_apriori_filtered = pd.get_dummies(df_apriori_filtered,
drop_first=True)
frequent_itemsets = apriori(df_apriori_filtered, min_support=0.1,
use_colnames=True, verbose=1)
rules = association_rules(frequent_itemsets, metric="lift",
min_threshold=0.2)
rules = rules.sort_values(['confidence'], ascending=False)
print(rules.head(10))

print("\n---------------------ASSOCIATION RULE MINING----------------
--\n")
```

To maintain the clarity and continuity of the report, the full codebase is provided in a separate file linked within this section.

9 References

- https://ieeexplore.ieee.org/abstract/document/9740817

- https://scikit-learn.org/stable/
- https://pandas.pydata.org/docs/
- https://numpy.org/doc/stable/
- https://matplotlib.org/stable/index.html
- https://canvas.vt.edu/courses/185554/files
- https://seaborn.pydata.org/