Car Price Prediction: Analysing Classification Techniques and Feature Impact

GROUP MEMBERS:

Chan, Nathan
Choudhary, Ronakkumar Pitamber
Ebrahimi Aiden
Gautam, Biraj
Koirala, Samskar
Moradi, Amir
Mutanda, Franck Venance
Nwankwo, Louisian
Terchella, Mohamed Radhouan
Thapa Magar, Binaya

Problem Statement

- Geely Auto: Chinese company entering the US market.
- Compete with local and European brands.
- Dataset: Comprehensive data collected to analyze car pricing factors.

Objective

- Identify key factors influencing car prices.
- Analyze impact of variables on pricing.
- Build predictive model to estimate prices.
- Strategy development for US market entry.

Dataset Overview

Car Price Prediction Multiple Linear Regression

- **Number of Classes:** The dataset is primarily for regression, so it does
- not contain distinct classes.

- **Number of Features:** 25 features and 1 target variable.

- **Number of Samples:** 205.

Data Processing

Missing Values

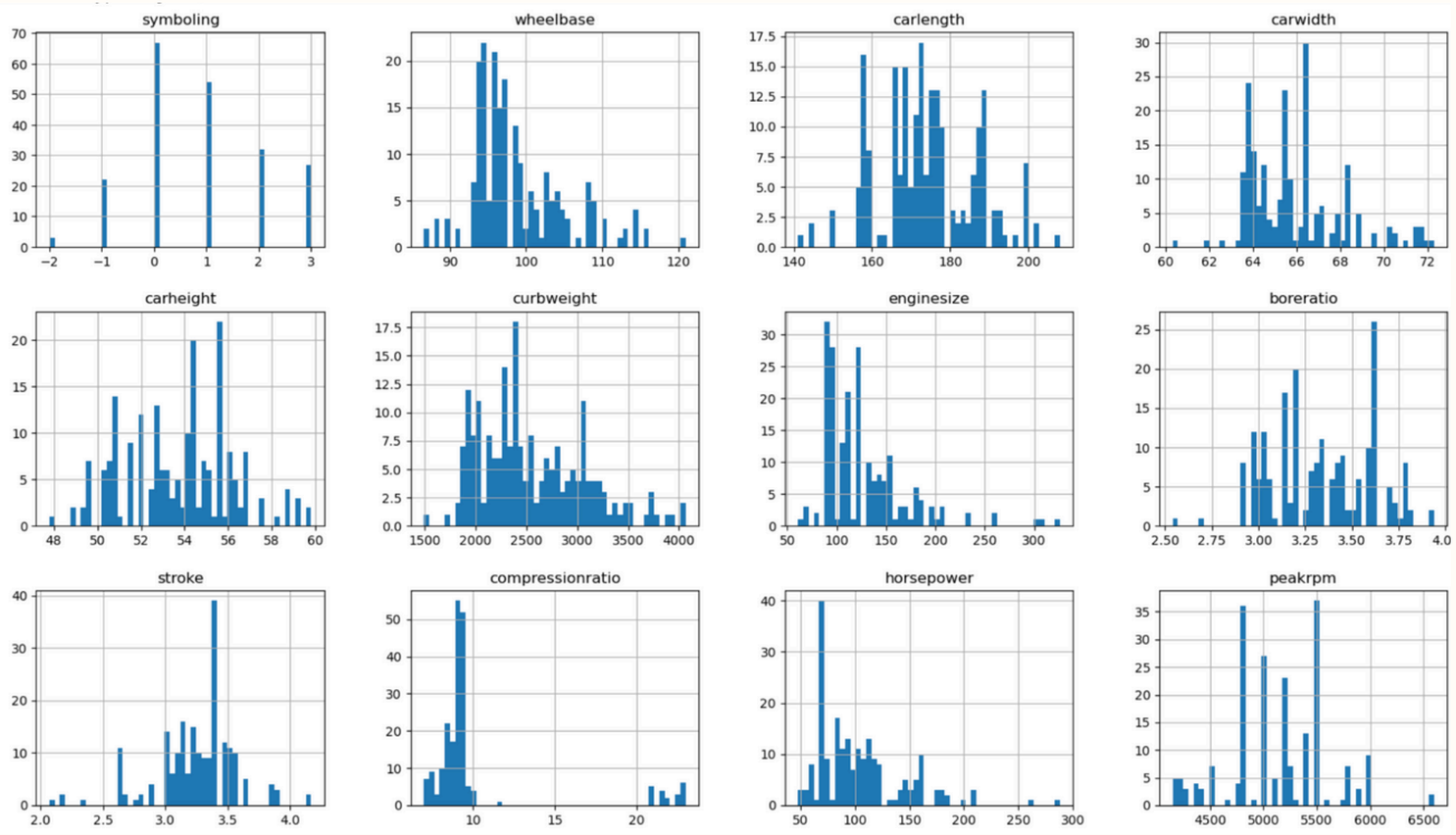
- Columns with 40% or more dropped.
- Missing values replaced with mode (Categorical) and mean (Numerical) values.

Irrelevant Columns Removed

- Columns with non-predictive information dropped.

```
# Step 2: Drop irrelevant columns and handle missing data

# Handling missing values
missing_summary = data.isnull().sum()
print("Missing values in each column:\n", missing_summary)

# Drop columns with too many missing values (if any)
threshold = 0.4  # Example: Drop columns with more than 40% missing val
data = data.loc[:, data.isnull().mean() < threshold]

# Impute remaining missing values with mean/median/mode as appropriate
for col in data.columns:
    if data[col].isnull().sum() > 0:
        if data[col].dtype == 'object':  # Categorical features
            data[col].fillna(data[col].mode()[0], inplace=True)
        else:  # Numerical features
            data[col].fillna(data[col].mean(), inplace=True)

print("Missing values after handling:\n", data.isnull().sum())

data = data.drop(['CarName', 'car_ID', 'price'], axis=1)
data.head()
```

Feature Distribution



Individual Histogram distribution for features in the dataset

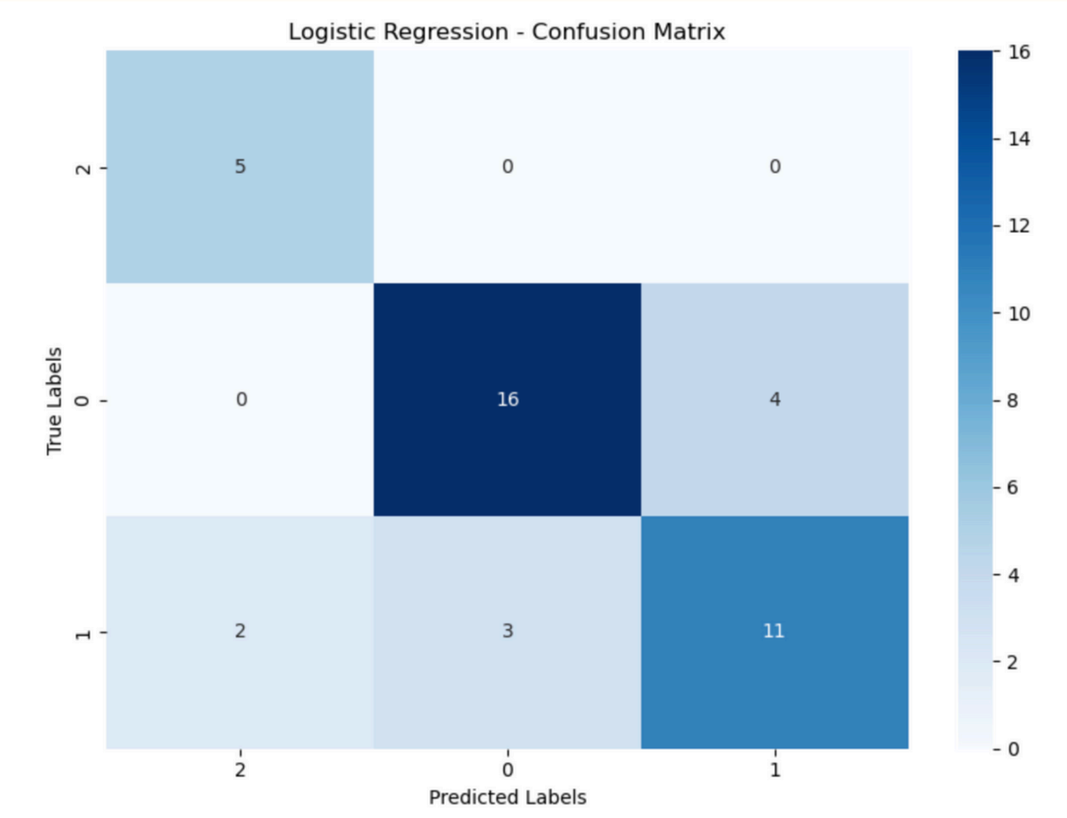
Classification Models Overview

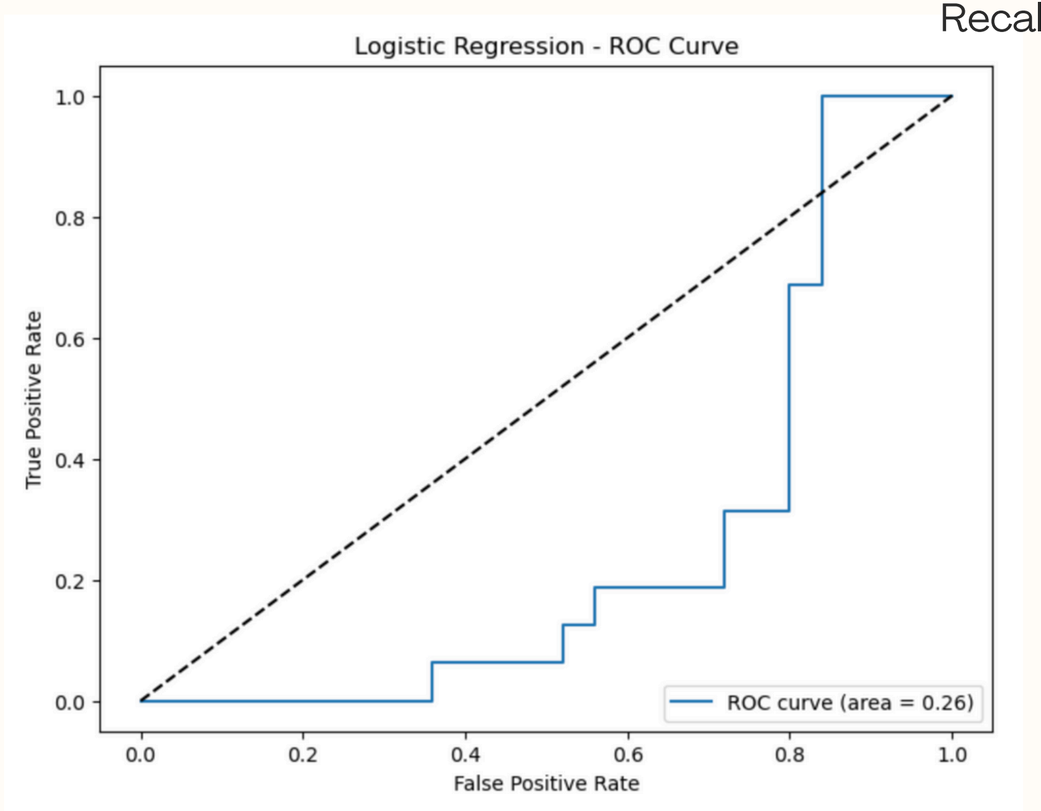
- Logistic Regression
- Decision Tree
- Random Forest
- SGD (Stochastic Gradient Descent)
- SVM (Support Vector Machine)

Logistic Regression

- The LogisticRegression model is defined with the max_iter=1000 parameter to ensure convergence for datasets that require more iterations. Logistic regression predicts probabilities using a linear combination of input features.
- Connection to Visualizations: The confusion matrix and ROC curve functions can be applied here to evaluate the model's classification performance and its ability to differentiate between classes.

Accuracy: 0.7805
Precision: 0.7841
Recall: 0.7805





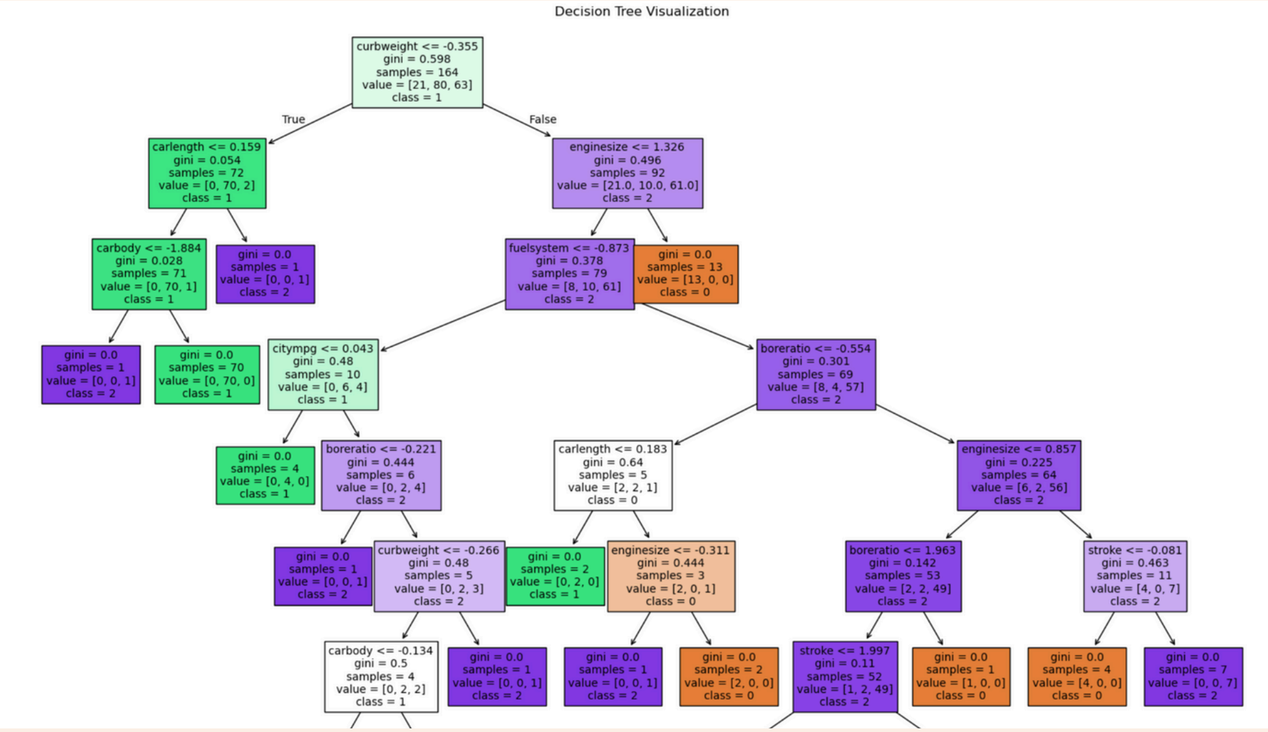
Decision Tree

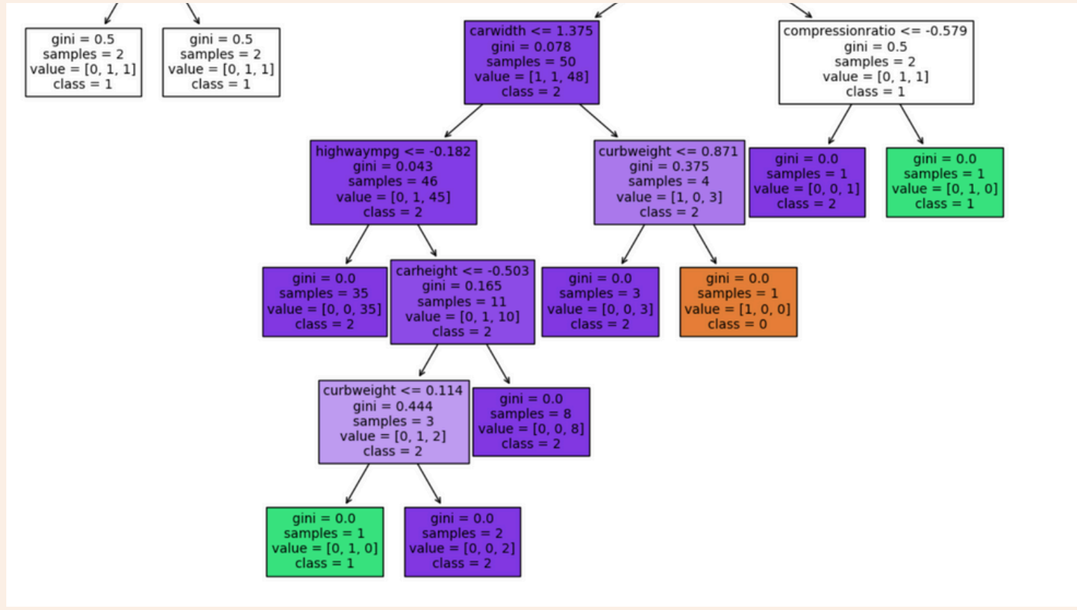
- The DecisionTreeClassifier is used for non-parametric classification. It builds a tree where decisions are based on feature thresholds. The random state ensures the same tree is produced every time the code is run.
- Connection to Visualizations: The plot_decision_tree function visualizes the structure of the decision tree, showing how features contribute to classification. This helps in explaining the model's behavior.

Accuracy: 0.8293

Precision: 0.8354

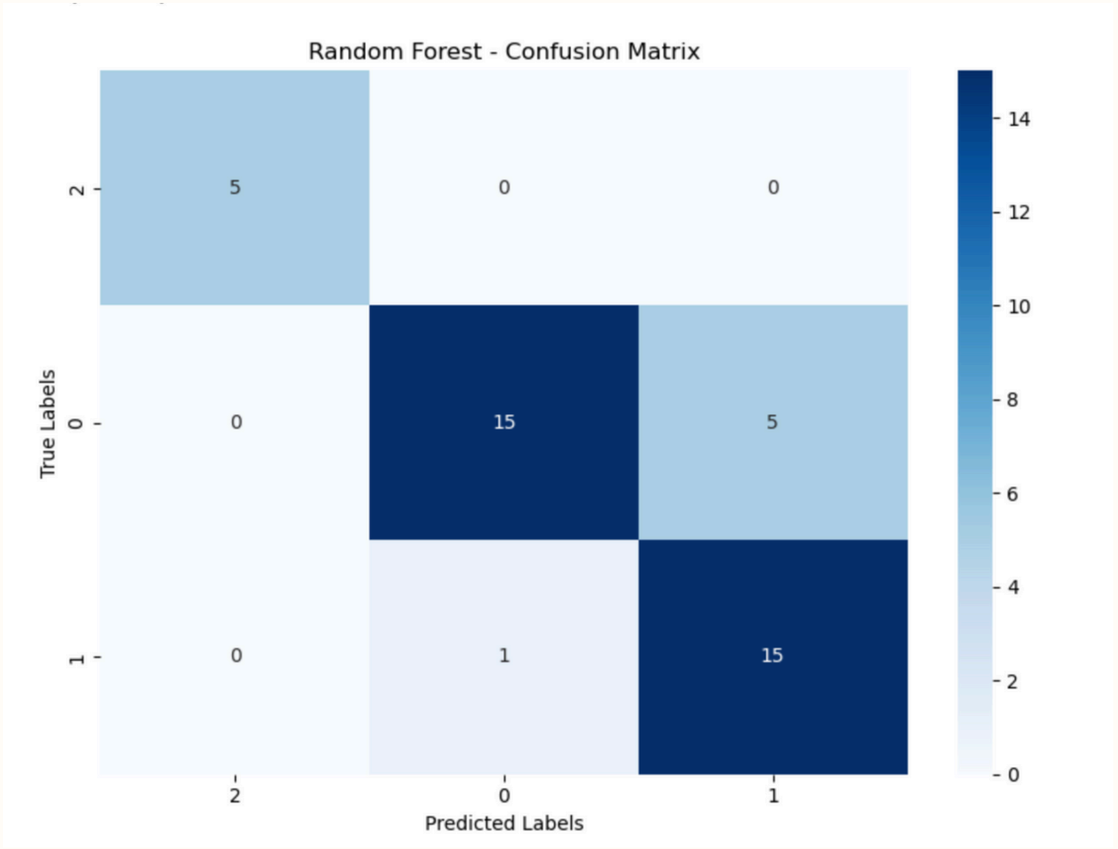
Recall: 0.8293





Random Forest

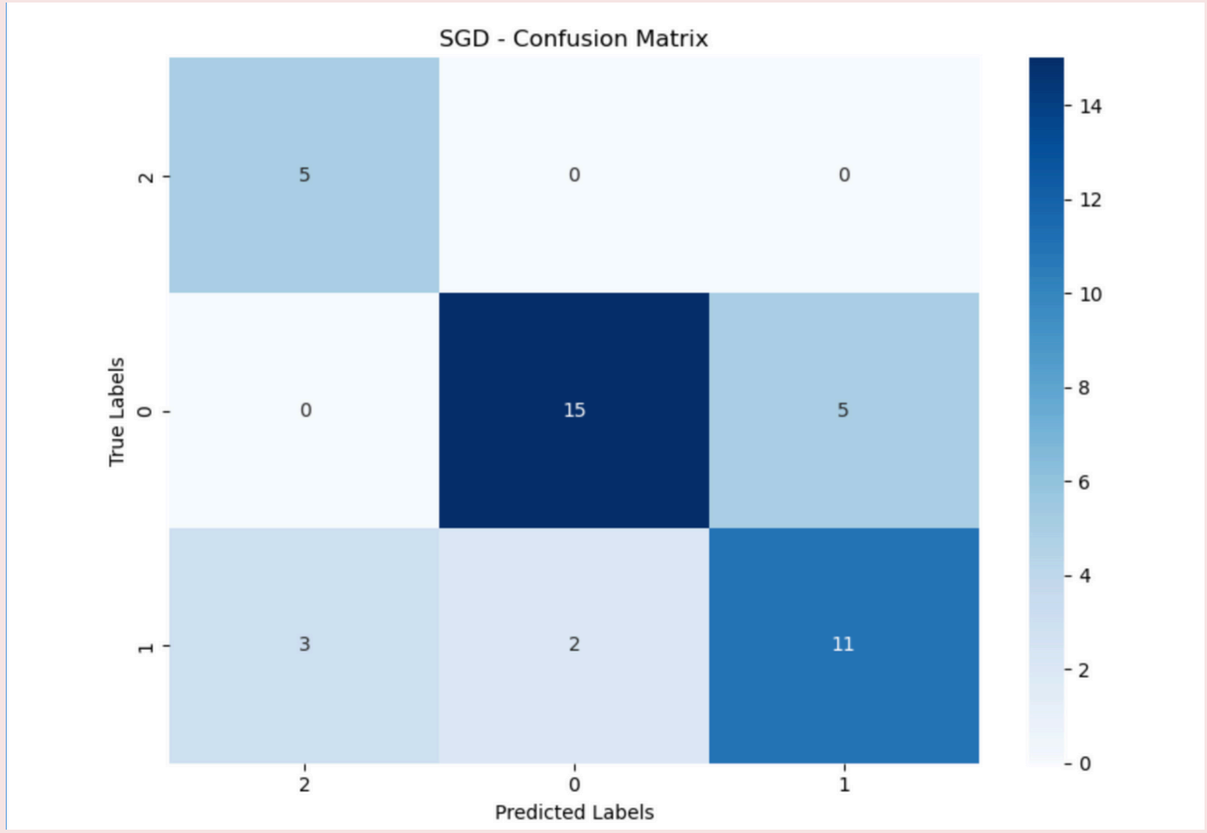
- The RandomForestClassifier aggregates the predictions of multiple decision trees to enhance accuracy and prevent overfitting.
- Connection to Visualizations: Use the plot_feature_importance function to display how much each feature contributes to the model's decision-making. This visualization helps highlight the most influential predictors.



Accuracy: 0.8537
Precision: 0.8720
Recall: 0.8537

Stochastic Gradient Descent (SGD)

- The SGDClassifier uses iterative optimization to minimize a loss function. This is particularly effective for large-scale datasets.
- Connection to Visualizations:
- The confusion matrix and ROC curve functions can be employed to assess the model's accuracy and performance across different thresholds.

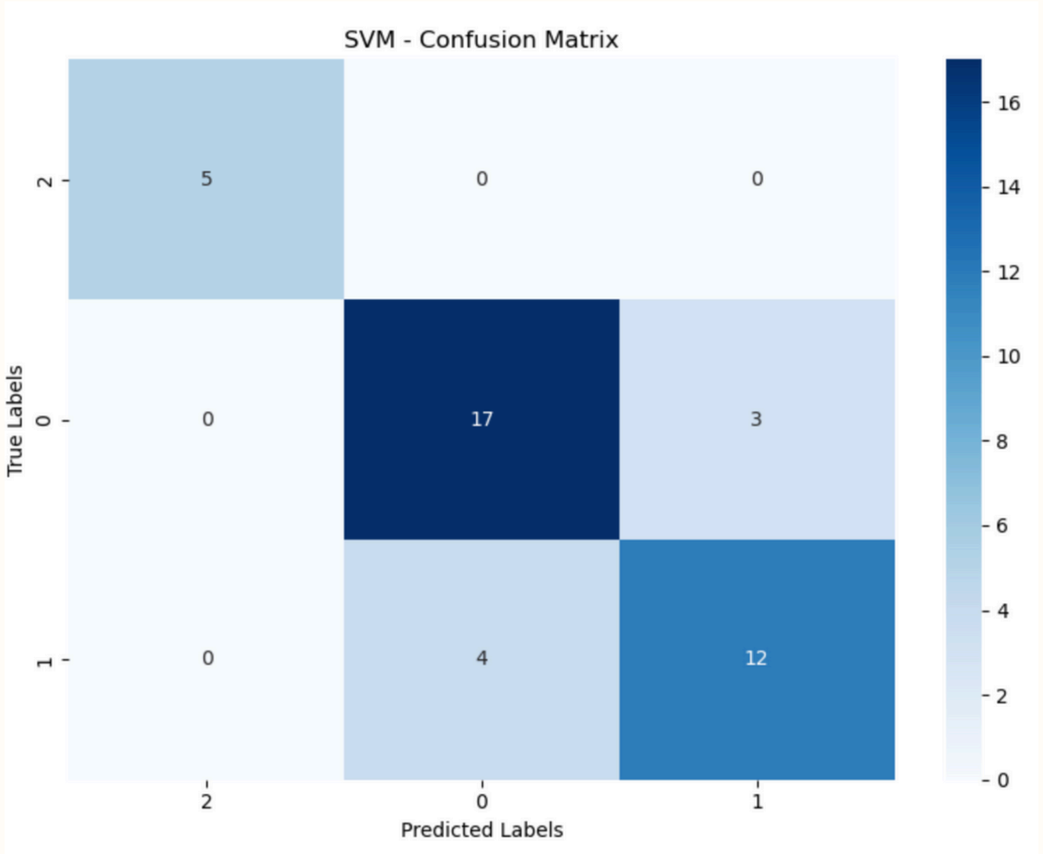


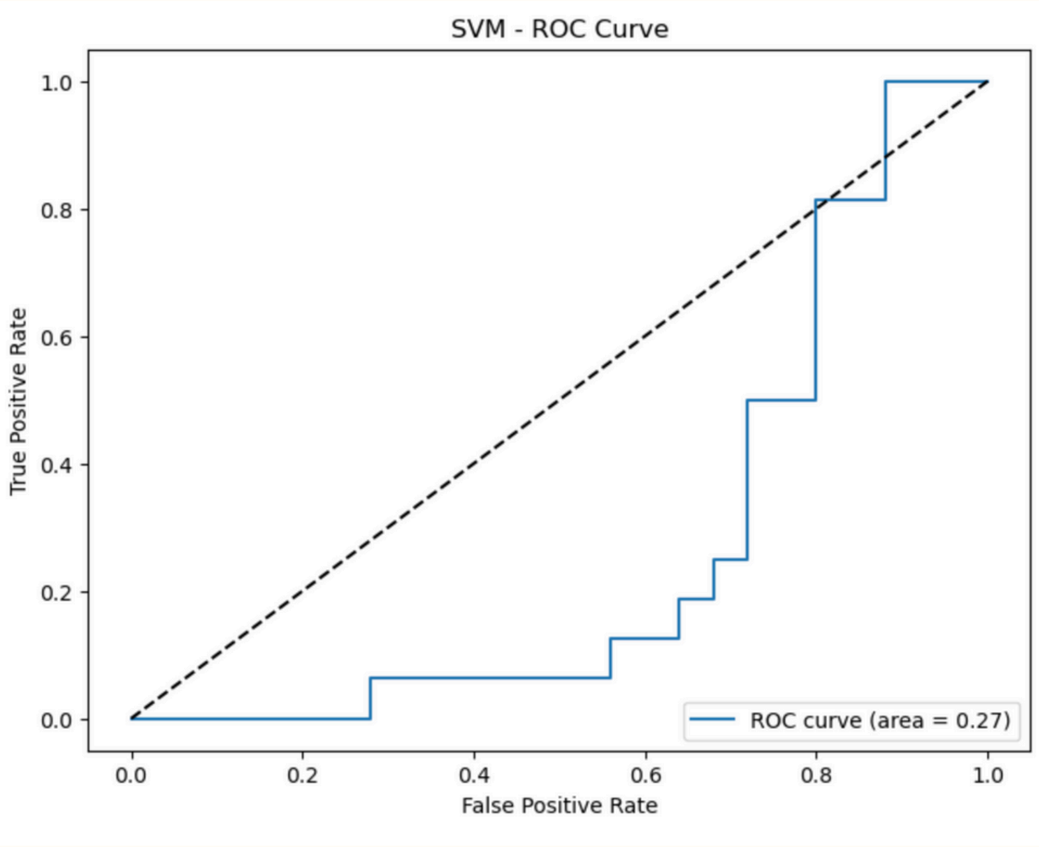
Accuracy: 0.7561
Precision: 0.7749
Recall: 0.7561

Support Vector Machine (SVM)

- The SVC classifier finds the optimal hyperplane to separate classes. The probability=True parameter enables probability predictions, useful for the ROC curve.
- Connection to Visualizations: Plotting the ROC curve demonstrates how well the model separates the classes across different thresholds. This is essential for understanding the trade-off between true positives and false positives.

Accuracy: 0.8293
Precision: 0.8290
Recall: 0.8293





Tuning with GridSearchCV

Objective: Optimize model performance by systematically exploring hyperparameter combinations.

Parameter grids for each model:

- Logistic Regression: {'C': [0.1, 1, 10], 'solver': ['liblinear', 'lbfgs']}
- Decision Tree: {'max_depth': [3, 5, 10], 'criterion': ['gini', 'entropy']}
- Random Forest: {'n_estimators': [50, 100, 200], 'max_depth': [5, 10, None]}
- SGD: {'alpha': [0.0001, 0.001, 0.01], 'loss': ['hinge', 'log'], 'penalty': ['l2', 'l1']}
- SVM: {'C': [0.1, 1, 10], 'kernel': ['linear', 'rbf'], 'gamma': ['scale', 'auto']}

- Use GridSearchCV for cross-validation to evaluate all parameter combinations and select the best configuration.
- Evaluate tuned models on the test dataset.

Outcome:

- Best Parameters for Logistic Regression: {'C': 0.1, 'solver': 'lbfgs'}
  Accuracy after tuning: 0.8049

- Best Parameters for Decision Tree: {'criterion': 'gini', 'max_depth': 10}
  Accuracy after tuning: 0.7805

- Best Parameters for Random Forest: {'max_depth': 5, 'n_estimators': 50}
  Accuracy after tuning: 0.8537

- Best Parameters for SGD: {'alpha': 0.001, 'loss': 'hinge', 'penalty': 'l1'}
  Accuracy after tuning: 0.7805

- Best Parameters for SVM: {'C': 10, 'gamma': 'scale', 'kernel': 'rbf'}
  Accuracy after tuning: 0.8293

Feature Removal Analysis

Hypothesis: Evaluate the impact of feature reduction on model performance.

Select a subset of features, retaining 80% of the total features:

```
reduced_features = X_train.columns[:int(len(X_train.columns) * 0.8)]
X_train_reduced = X_train[reduced_features]
X_test_reduced = X_test[reduced_features]
```

Retrain all models using the reduced feature set.

```
for name, clf in classifiers.items():
    clf.fit(X_train_reduced, y_train)
    y_pred = clf.predict(X_test_reduced)
    accuracy = accuracy_score(y_test, y_pred)
    print(f"Accuracy with reduced features for {name}: {accuracy:.4f}")
```

Findings:

- Some models showed minimal performance degradation, indicating the removed features had low importance.

Removed Features: [Compress, horsepower, peakrpm, citympg, highwaympg]

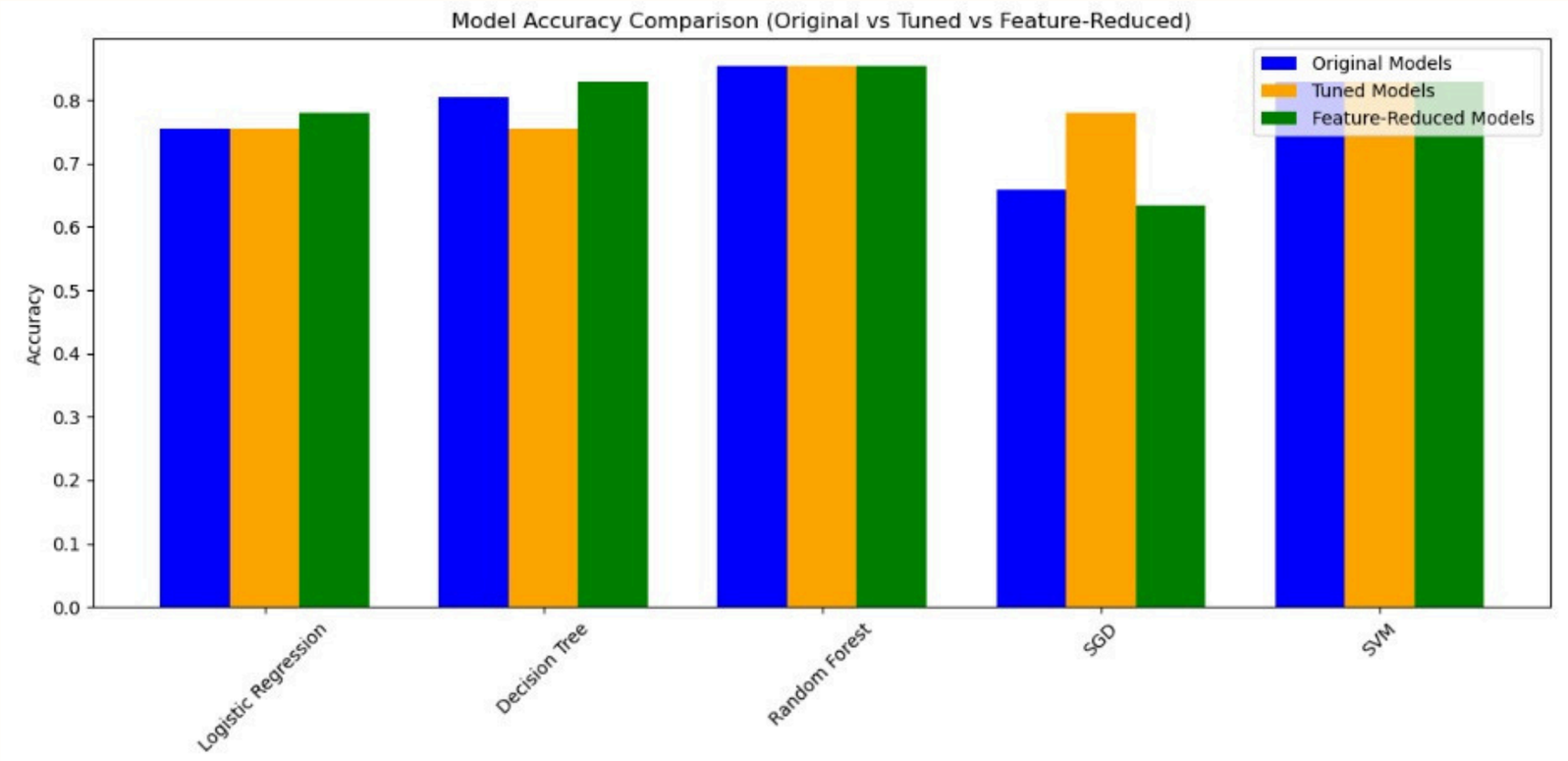
--- Retraining Logistic Regression with reduced features -
Accuracy with reduced features: 0.7561
--- Retraining Decision Tree with reduced features ---
Accuracy with reduced features: 0.8537
--- Retraining Random Forest with reduced features ---
Accuracy with reduced features: 0.8537
--- Retraining SGD with reduced features ---
Accuracy with reduced features: 0.7073
--- Retraining SVM with reduced features ---
Accuracy with reduced features: 0.8293

Comparative Analysis

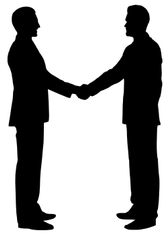
Objective: Visualize and compare the accuracy of models before and after tuning.

Key Observations:

- Feature reduction had mixed results – it helped Decision Tree but lowered SGD accuracy.
- Random Forest and SVM stayed strong, with little change from tuning or feature reduction.
- Tuning improved SGD but didn't help Decision Tree much, showing different model needs.

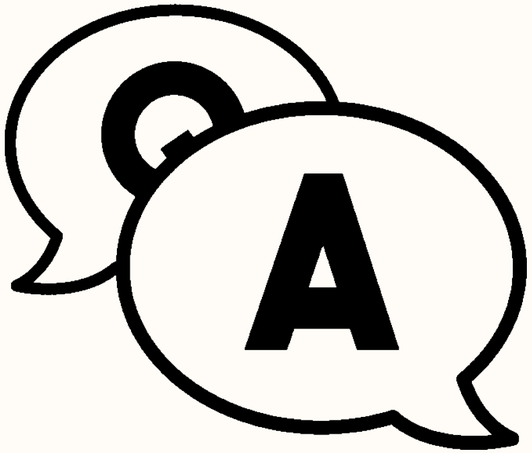


Conclusion



- **Summary of Findings:**
  - Hyperparameter tuning enhances model performance significantly.
  - Feature selection helps simplify models with minimal impact on performance.
  - Random Forest emerged as the best-performing model overall.
- **Recommendations:**
  - Leverage GridSearchCV for systematic model optimization.
  - Consider feature selection for reducing model complexity without compromising accuracy.
- **Challenges:**
  - Computational cost of GridSearchCV for large datasets.
  - Balancing model simplicity and performance.

Thank you!



References

- Kaggle. (n.d.). Car Price Prediction Dataset. Kaggle. Retrieved January 18, 2025, from https://www.kaggle.com/datasets/hellbuoy/car-price-prediction?resource=download