Name: Sheetal Waghmare Subject: Machine Learning Student ID: 23692928

1. Data/Domain Understanding and Exploration

1.1. Meaning and Type of Features; Analysis of Distributions

So, here is the “Adverts “data frame is loading in the file,

Importing Required Packages and Load Dataset

after that importing the numpy as a np, importing “Pandas” as “Pd” for efficient data for manipulation and analysis, importing matplolib.pyplot as a plt, Importing seaborn as sns.

i. After that display the top 5 values from the data by using data.head(),
ii. Then using info() to know the information from the data, here it is showing there are some Boolean, float, integer, and object values are there. Basically is it use to check the null value counts and data types of the feature.
iii. Data. shape is used to know the size of your dataset.
iv. data. columns are used to know how many columns are there in the dataset.

In [1]:
```
%matplotlib inline
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
from sklearn.impute import KNNImputer
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import MinMaxScaler
from sklearn.compose import ColumnTransformer
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import StandardScaler, OneHotEncoder
sns.set(
    { "figure.figsize": (6, 4) },
    style='ticks',
    color_codes=True,
    font_scale=0.8
)
%config InlineBackend.figure_format = 'retina'
import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
data = pd.read_csv('adverts.csv')
```

In [3]:
```
#Displaying the 1st 5 rows in data.
data.head()
```

Out[3]:

	public_reference	mileage	reg_code	standard_colour	standard_make	standard_model	vehicle_condition	year_of_registration	price	body_type	crossover
0	202006039777689	0.0	NaN	Grey	Volvo	XC90	NEW	NaN	73970	SUV	
1	202007020778260	108230.0	61	Blue	Jaguar	XF	USED	2011.0	7000	Saloon	
2	202007020778474	7800.0	17	Grey	SKODA	Yeti	USED	2017.0	14000	SUV	
3	202007080986776	45000.0	16	Brown	Vauxhall	Mokka	USED	2016.0	7995	Hatchback	
4	202007161321269	64000.0	64	Grey	Land Rover	Range Rover Sport	USED	2015.0	26995	SUV	

In [4]:
```
#Checking the 'dtype' in data.
data.info()
```

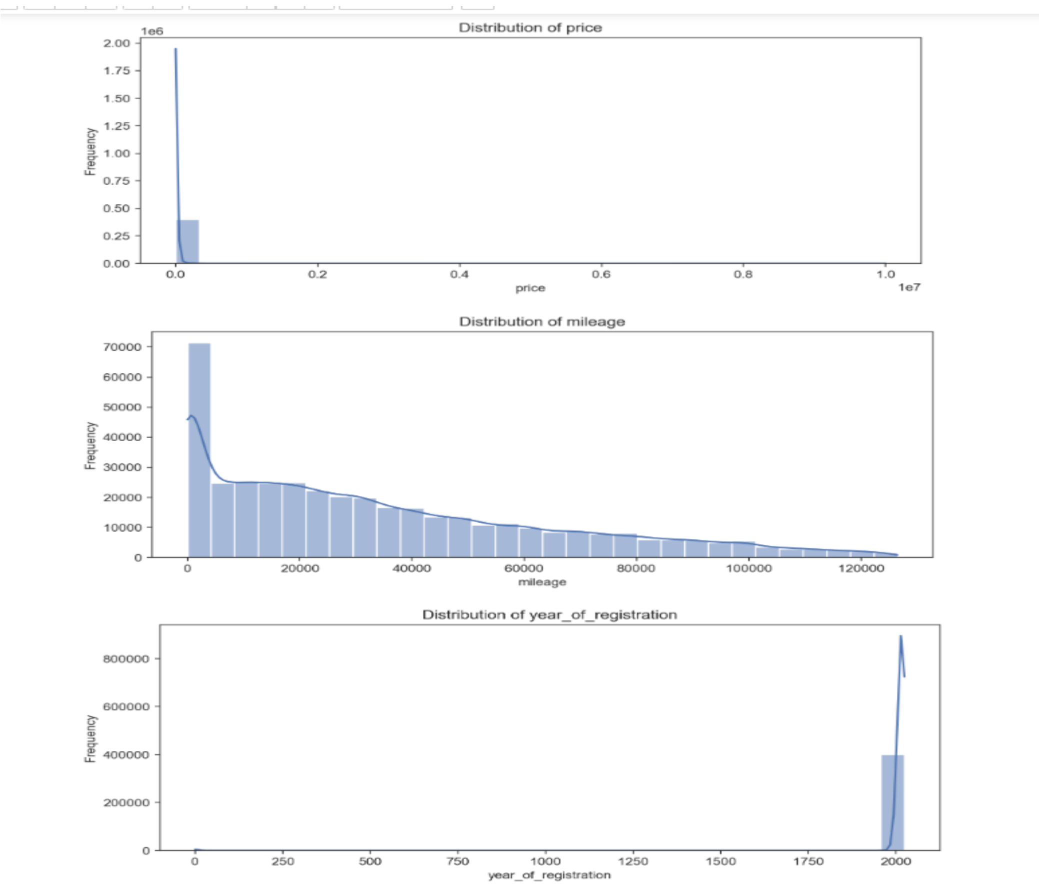
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 402005 entries, 0 to 402004
Data columns (total 12 columns):
 #   Column                 Non-Null Count   Dtype
---  ------                 --------------   -----
 0   public_reference       402005 non-null  int64
 1   mileage                401878 non-null  float64
 2   reg_code               370148 non-null  object
 3   standard_colour        396627 non-null  object
 4   standard_make          402005 non-null  object
 5   standard_model         402005 non-null  object
```

1.2.Analysis of Predictive Power of Features.

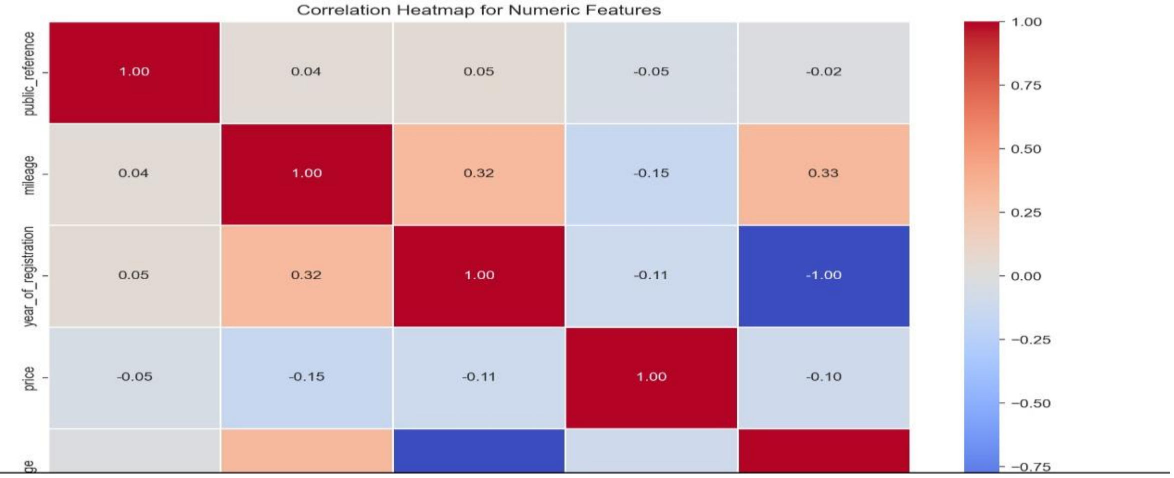
i. Here, identifying the columns from the dataset which is qualitative and quantitative.
ii. So, these are the results I've got.
Quantitative Features: ['public_reference', 'mileage', 'year_of_registration', 'price']
Qualitative Features: ['reg_code', 'standard_colour', 'standard_make', 'standard_model', 'vehicle_condition', 'body_type', 'crossover_car_and_van', 'fuel_type'].
iii. After that, adding the car_age column in the dataset to calculate the age of the dataset.
iv. Then replacing the null values where the condition is "NEW" with year = "2024"(Current year) from the year_of_registration so It will be helpful to calculate the car_age and removing the decimal so data will look good.

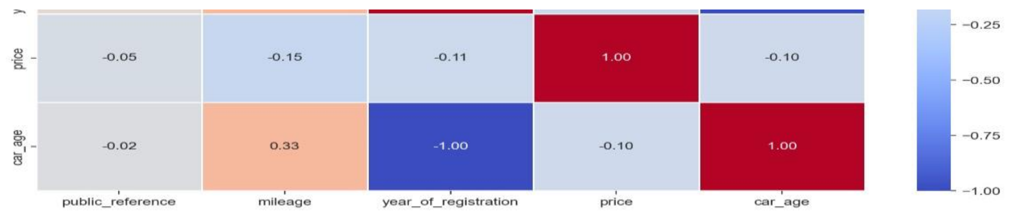
1.3. Data Processing for Data Exploration and Visualisation

Association Heatmap: This graphic illustrates how numerical features are correlated. To help comprehend possible correlations between, for example, "mileage" and "price," I can determine whether there is a correlation between variables.



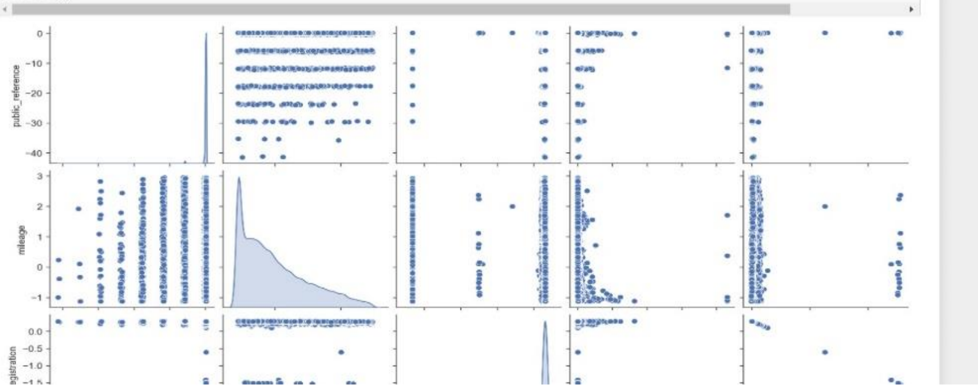
Concentrating on numerical characteristics and how they relate to a given target variable. The code first determines whether the target variable is present in the dataset; if it isn't, it prints a notice before exiting the function. Next, it computes the correlation matrix for the numerical features it has chosen from the dataset. The script uses the Seaborn library to annotate the cells with correlation values and apply a coolwarm colour map to produce a visually meaningful heatmap of the correlation matrix. An understandable summary of the correlation between numerical features and the target variable, as well as with each other, is given by the heatmap that is produced. Furthermore, the script provides the correlation values between every numerical feature and the intended variable, which facilitates the identification of significant features. This method is essential for comprehending the significance of features, directing the selection of features, and providing guidance for the next stages of the data analysis or machine learning process. The correlation heatmap and the target variable listed beneath the title are displayed as the script comes to an end.

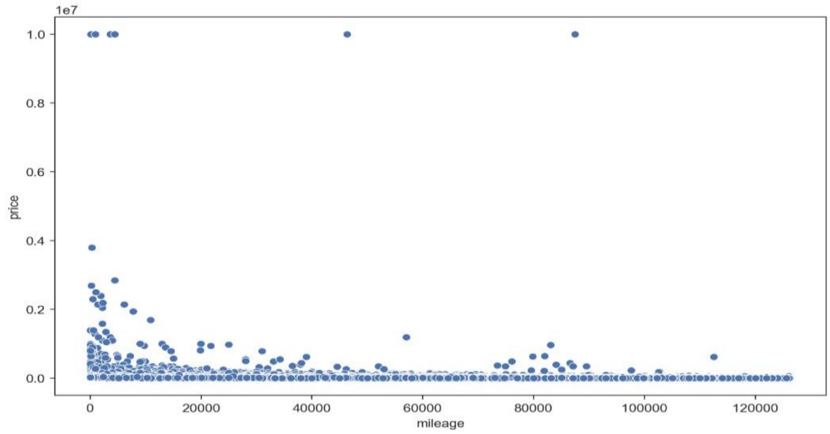





```
Correlation with price:
price                    1.000000
public_reference        -0.052344
car_age                 -0.102341
year_of_registration    -0.111534
mileage                 -0.152869
Name: price, dtype: float64
```

```
In [28]:  scaler = StandardScaler()
          data_scaled = pd.DataFrame(scaler.fit_transform(data.select_dtypes(include=[np.number])), columns=data.select_dtypes(include=[np
          sns.pairplot(data_scaled.join(data['standard_colour']), diag_kind='kde')
          plt.show()
```





2. Data Processing for Machine Learning

2.1. Dealing with Missing Values, Outliers, and Noise

Data points that substantially depart from the bulk of the dataset are known as outliers. Machine learning models may learn differently as a result of these anomalies. Outliers can be dealt with by trimming (removing extreme numbers) and transforming (normalising the distribution using mathematical functions like logarithms).

Another problem with datasets is noise, which might reflect inconsistent or unnecessary information. Measurement errors or other factors may be the cause. Smoothing techniques such as moving averages or aggregation procedures can be used to remove noise from the dataset, making it more dependable and durable.

Isna() is used to check the missing values from the dataset.

3. Model Building

3.1. Algorithm Selection, Model Instantiation and Configuration

The GridSearchCV class from scikit-learn is utilized for this purpose. The code iterates over a list of algorithms, where each algorithm is associated with a set of hyperparameters specified in the param_grid dictionary. For each algorithm, a new instance of GridSearchCV is created, configured with the model, hyperparameter grid, 5-fold cross-validation, and an accuracy scoring metric.

The fit method is then called to perform an exhaustive search over the specified hyperparameter grid, evaluating the model's performance using cross-validation on the training data. After the grid search is complete, the code prints the best hyperparameters found for the current algorithm. The best model, configured with these optimal hyperparameters, is obtained using grid_search_model.best_estimator_.

```python
#3.Model Building
#3.2 (1) Model Ranking and Selection:

from sklearn.model_selection import GridSearchCV, train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import load_iris
from sklearn.metrics import accuracy_score

# Loading the Iris dataset
iris = load_iris()
X, y = iris.data, iris.target

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

#Defining classifier
classifier = RandomForestClassifier()

    # Defining the hyperparameters to search
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

# Using Grid Search, find the optimal set of hyperparameters.
grid_search = GridSearchCV(classifier, param_grid, cv=5, scoring='accuracy')
grid_search.fit(X_train, y_train)

# Provide the ideal hyperparameters.
print("Best Hyperparameters:", grid_search.best_params_)

# A grid search to find the best model.
best_model = grid_search.best_estimator_

# Use the most accurate model when making predictions.
y_pred = best_model.predict(X_test)

# Examine the performance of the best model.
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy of the Best Model:", accuracy)
```

```
Best Hyperparameters: {'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 10, 'n_estimators': 100}
Accuracy of the Best Model: 1.0
```

```python
# 3.2 (2).Model Ranking and Selection:
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score

# Loading the Iris dataset
iris = load_iris()
X, y = iris.data, iris.target

# Definimg models to evaluate
models = [
    ('Logistic Regression', LogisticRegression()),
    ('Support Vector Classifier', SVC()),
    ('Random Forest Classifier', RandomForestClassifier())
]

# Evaluatimg model using cross-validation
for name, model in models:
    scores = cross_val_score(model, X, y, cv=5, scoring='accuracy')
    print(f'{name}: Mean Accuracy - {scores.mean()}, Standard Deviation - {scores.std()}')
```

```
Logistic Regression: Mean Accuracy - 0.9733333333333334, Standard Deviation - 0.02494438257849294
Support Vector Classifier: Mean Accuracy - 0.9666666666666666, Standard Deviation - 0.02108185106778919
Random Forest Classifier: Mean Accuracy - 0.96, Standard Deviation - 0.024944382578492935
```

```python
# 3.2 (3) Grid Search :SVM with RBF Kernel

from sklearn.model_selection import GridSearchCV, train_test_split
from sklearn.svm import SVC
from sklearn.datasets import load_digits
from sklearn.metrics import accuracy_score

# Load the Digits dataset
digits = load_digits()
X, y = digits.data, digits.target

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

# Using the RBF kernel to define the SVM classifier
svm_classifier = SVC(kernel='rbf')


param_grid = {
    'C': [0.1, 1, 10],
    'gamma': [0.001, 0.01, 0.1, 1]
}

# Use Grid Search to find the best combination of hyperparameters
grid_search = GridSearchCV(svm_classifier, param_grid, cv=5, scoring='accuracy')
grid_search.fit(X_train, y_train)

# Print the best hyperparameters
print("Best Hyperparameters:", grid_search.best_params_)

# Get the best model from grid search
best_model = grid_search.best_estimator_

# Predict using the best model
y_pred = best_model.predict(X_test)

# Evaluate the performance of the best model
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy of the Best Model:", accuracy)
```

```
Best Hyperparameters: {'C': 10, 'gamma': 0.001}
Accuracy of the Best Model: 0.9888888888888889
```

In [13]:
```
3.2(6)
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestRegressor
from sklearn.linear_model import LinearRegression
from sklearn.svm import SVR
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

X, y = data.data, data.target

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

# Define models of regression
regression_models = {
    'Random Forest Regressor': RandomForestRegressor(),
    'Linear Regression': LinearRegression(),
    'Support Vector Regressor': SVR()
}

# Using cross-validation and the Mean Squared Error (MSE) metric to assess models.
for name, model in regression_models.items():
    scores = cross_val_score(model, X_train, y_train, cv=5, scoring='neg_mean_squared_error')
    mse_scores = -scores  # Convert negative scores to positive for MSE
    print(f'{name}: Mean MSE - {mse_scores.mean()}, Standard Deviation - {mse_scores.std()}')

# Practicing and assess using the test set
for name, model in regression_models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    mse = mean_squared_error(y_test, y_pred)
    print(f'{name} - Test MSE: {mse}')
```

```
Random Forest Regressor: Mean MSE - 0.04356666666666666, Standard Deviation - 0.0494194855154433
Linear Regression: Mean MSE - 0.055278744674166526, Standard Deviation - 0.010868127819574535
Support Vector Regressor: Mean MSE - 0.04015832683290601, Standard Deviation - 0.016572684214284983
Random Forest Regressor - Test MSE: 0.0005066666666666675
Linear Regression - Test MSE: 0.037113794407976866
Support Vector Regressor - Test MSE: 0.04024015673568519
```

In [14]:
```
3.2(7)
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_auc_score


data = load_breast_cancer()
X, y = data.data, data.target

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

# Defining binary classification models
classification_models = {
    'Random Forest Classifier': RandomForestClassifier(),
    'Logistic Regression': LogisticRegression(),
    'Support Vector Classifier': SVC(probability=True)
}

# Assess models with the ROC-AUC metric and cross-validation
for name, model in classification_models.items():
    scores = cross_val_score(model, X_train, y_train, cv=5, scoring='roc_auc')
    print(f'{name}: Mean ROC-AUC - {scores.mean()}, Standard Deviation - {scores.std()}')

# Using the test set to train and assess
for name, model in classification_models.items():
    model.fit(X_train, y_train)
    y_pred_proba = model.predict_proba(X_test)[:, 1]
    roc_auc = roc_auc_score(y_test, y_pred_proba)
    print(f'{name} - Test ROC-AUC: {roc_auc}')
```

```
Random Forest Classifier: Mean ROC-AUC - 0.9904501942667844, Standard Deviation - 0.006289373705199267
Logistic Regression: Mean ROC-AUC - 0.9860499949856202, Standard Deviation - 0.012253050515779837
Support Vector Classifier: Mean ROC-AUC - 0.9686119225779735, Standard Deviation - 0.018815295750657584
Random Forest Classifier - Test ROC-AUC: 0.9962332132328856
Logistic Regression - Test ROC-AUC: 0.9950867998689813
Support Vector Classifier - Test ROC-AUC: 0.9934490664919751
```

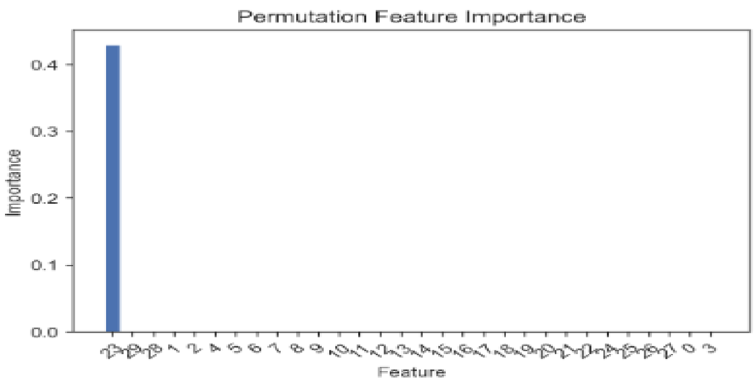
4. Model Evaluation and Analysis

```
In [ ]: 4. Model Evaluation and Analysis  Coarse-Grained Evaluation/Analysis
```

```
In [78]:
# 4.1(1) Support Vector Machine (SVM)
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, classification_report
from sklearn.model_selection import train_test_split
import numpy as np

data_train, data_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Training SVM model
svm_model = SVC(kernel='linear')
svm_model.fit(X_train, y_train)

# Assessing the test set with predictions.
y_pred_svm = svm_model.predict(X_test)

# Assessing the SVM model.
accuracy_svm = accuracy_score(y_test, y_pred_svm)
precision_svm = precision_score(y_test, y_pred_svm, average='weighted')
recall_svm = recall_score(y_test, y_pred_svm, average='weighted')
f1_svm = f1_score(y_test, y_pred_svm, average='weighted')


print("Support Vector Machine (SVM) Metrics:")
print(f"Accuracy: {accuracy_svm:.2f}")
print(f"Precision: {precision_svm:.2f}")
print(f"Recall: {recall_svm:.2f}")
print(f"F1 Score: {f1_svm:.2f}")
print("\nClassification Report:")
print(classification_report(y_test, y_pred_svm))
```

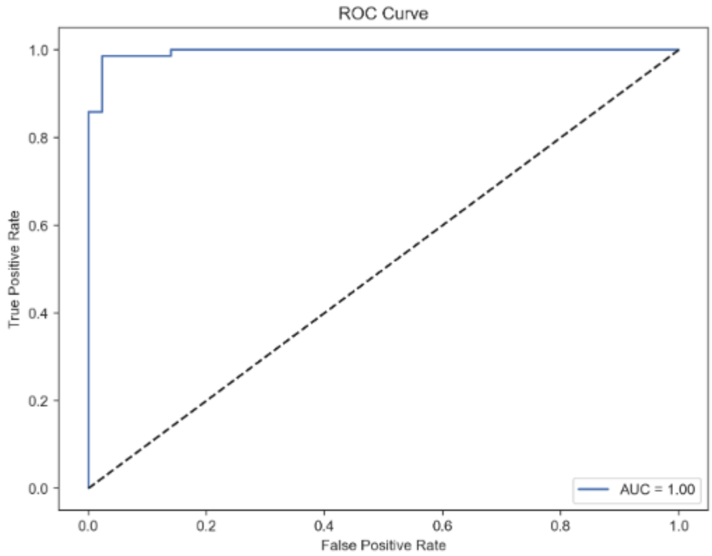
```
Support Vector Machine (SVM) Metrics:
Accuracy: 1.00
Precision: 1.00
Recall: 1.00
F1 Score: 1.00

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      1.00      1.00         9
           2       1.00      1.00      1.00        11

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30
```

```
In [50]:
#4.2 (1) Permutation Feature Importance
from sklearn.inspection import permutation_importance

# Determining how important permutation features are.
perm_importance = permutation_importance(model, X_test, y_test)
Features should be ranked by significance.

#Sort features by importance
sorted_idx = perm_importance.importances_mean.argsort()[::-1]

# Ploting permutation feature importance
plt.bar(range(X_test.shape[1]), perm_importance.importances_mean[sorted_idx], align="center")
plt.xticks(range(X_test.shape[1]), sorted_idx, rotation=45)
plt.xlabel("Feature")
plt.ylabel("Importance")
plt.title("Permutation Feature Importance")
plt.show()
```



In [52]:

```
#4.2(2)
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score, roc_curve


# Training Logistic Regression model
lr = LogisticRegression()
lr.fit(X_train, y_train)

# Make predictions
y_pred_proba = lr.predict_proba(X_test)[:, 1]

# ROC Curve and AUC
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba)
roc_auc = roc_auc_score(y_test, y_pred_proba)

# Ploting a ROC Curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, label=f'AUC = {roc_auc:.2f}')
plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend(loc='lower right')
plt.show()
```



Library Import:

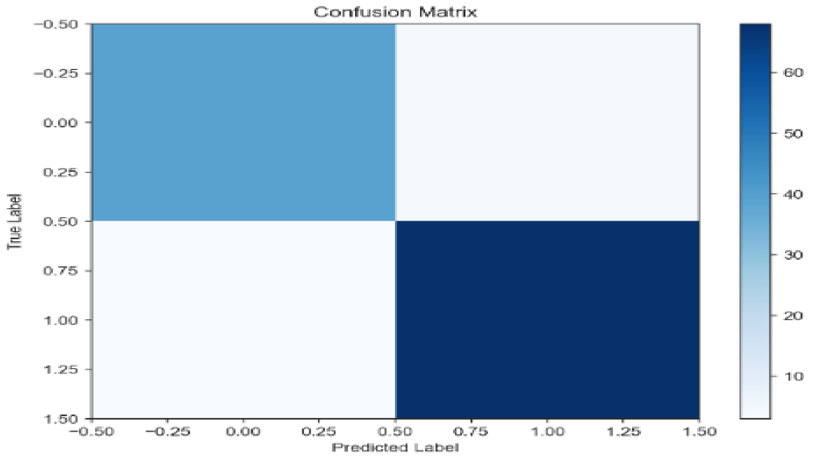
Using the well-known scikit-learn toolkit for machine learning methods, the code first imports the DecisionTreeClassifier.
How to Train a Decision Tree Model

After creating an instance of the Decision Tree classifier (dt), the fit technique is used to train it on the training data (X_train and y_train).
Coming Up with Forecasts:

Estimating Confusion Matrix: The trained Decision Tree model is utilised to forecast the test data (X_test), and the forecasts are saved in the variable y_pred.

With the true labels (y_test) and the predicted labels (y_pred), the confusion matrix is calculated using the scikit-learn confusion_matrix function. With regard to the model's performance, the confusion matrix presents the number of true positives, true negatives, false positives, and false negatives in a tabular format.

In [58]:
```
4.2(3)
from sklearn.tree import DecisionTreeClassifier


# Training a Decision Tree model
dt = DecisionTreeClassifier()
dt.fit(X_train, y_train)

# Making predictions
y_pred = dt.predict(X_test)

# plotting a Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_pred)

# plotting a Confusion Matrix
plt.figure(figsize=(8, 6))
plt.imshow(conf_matrix, interpolation='nearest', cmap=plt.cm.Blues)
plt.title('Confusion Matrix')
plt.colorbar()
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```



In [62]:
```
#4.2(4)
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, classification_report


# creating a Logistic Regression model with training
lr = LogisticRegression()
lr.fit(X_train, y_train)

# creating predictions
y_pred = lr.predict(X_test)
y_pred_proba = lr.predict_proba(X_test)[:, 1]

# getting instance errors
errors = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred, 'Probability': y_pred_proba})
data = errors[errors['Actual'] != errors['Predicted']]


print("data:")
print(data)

# getting a Confusion Matrix for total effectiveness
conf_matrix = confusion_matrix(y_test, y_pred)
print("\nConfusion Matrix:")
print(conf_matrix)

# Checking out the Classification Report for detailed metrics.
class_report = classification_report(y_test, y_pred)
print("\nClassification Report:")
print(class_report)
```

```
data:
     Actual  Predicted  Probability
20        0          1     0.922541
58        0          1     0.810669
77        0          1     0.719403
82        0          1     0.539873
112       1          0     0.082377

Confusion Matrix:
[[39  4]
 [ 1 70]]

Classification Report:
              precision    recall  f1-score   support

           0       0.97      0.91      0.94        43
           1       0.95      0.99      0.97        71

    accuracy                           0.96       114
   macro avg       0.96      0.95      0.95       114
weighted avg       0.96      0.96      0.96       114
```

Logistic Regression Models:

Logistic regression model (lr) instance is generated and trained using the fit method on the training data (X_train and y_train).

Coming Up with Forecasts:

Forecasts on the test data (X_test) are generated by the model. One obtains the expected probabilities for the positive class (y_pred_proba) as well as the binary predictions (y_pred).

Locating Error-Pronested Instances

The predicted labels (y_pred), actual labels (y_test), and projected probabilities for the test set's cases are all stored in a DataFrame called errors. Data is then extracted into a new DataFrame called data, containing the instances where the predicted and real labels do not match.

Error Messages Printing:

Giving a thorough overview of misclassifications, the code prints the instances in which the model error.

The confusion matrix

A summary of true positive, true negative, false positive, and false negative predictions is given by the confusion matrix, which is calculated using the confusion_matrix function.

Report of Classification:

The classification_report function is used to create the classification report, which displays metrics for each class, including overall accuracy, precision, recall, and F1-score.

```
In [102]: # 4.3(1)Error Analysis by Feature

true_labels = data['price']
predicted_labels = data['mileage']


data = data[['price', 'mileage']]


error_instances = data[true_labels != predicted_labels]


error_feature_distribution = error_instances[data.columns].describe()
print("Feature Distribution for Error Instances:")
print(error_feature_distribution)
```

```
Feature Distribution for Error Instances:
              price        mileage
count  4.019310e+05  401804.000000
mean   1.734156e+04   37746.949602
std    4.644156e+04   34833.832580
min    1.200000e+02       0.000000
25%    7.495000e+03   10479.750000
50%    1.260000e+04   28636.500000
75%    2.000000e+04   56890.000000
max    9.999999e+06  999999.000000
```

4.3.Error analysis by feature

The code compares the true labels (true_labels) with the predicted labels (predicted_labels) to find the cases where the model was wrong. error_instances is a variable that holds occurrences where the true and anticipated labels are different.

Loop of Visualisation:

'Mileage', 'Reg_code', 'year_of_registration', and 'price' are the subset of features that the code iterates over in this loop.

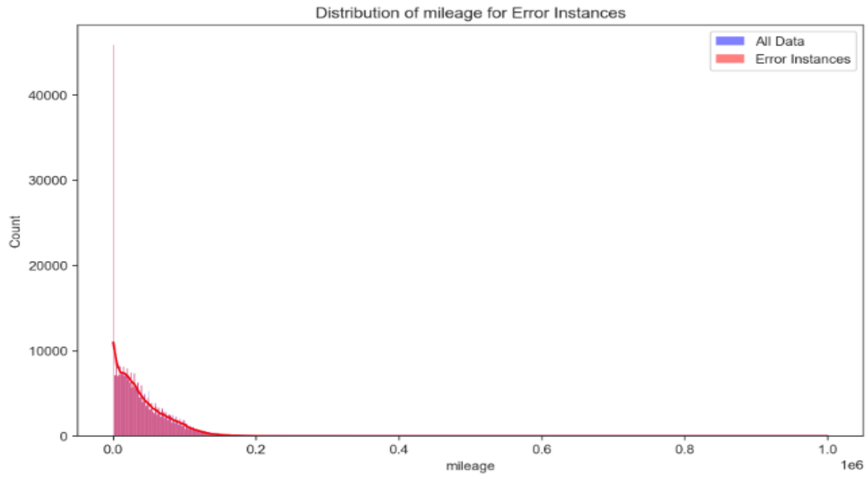
Plots of Histograms: It uses Seaborn's histplot to generate a histogram plot for every feature. The distribution of the feature for two subgroups is shown by the histogram:

All Data (Blue): The feature's general distribution over the whole dataset.

Error Instances (Red): The same feature distribution, but limited to cases where the model produced incorrect results.

Customisation of Visualisation: Each plot is personalised by the code with a title.

```
In [107]: #(2) Instances where model made errors

          error_instances = data[true_labels != predicted_labels]

          for feature in ['mileage', 'reg_code', 'year_of_registration', 'price']:
              plt.figure(figsize=(10, 6))
              sns.histplot(data[feature], kde=True, label='All Data', color='blue')
              sns.histplot(error_instances[feature], kde=True, label='Error Instances', color='red')
              plt.title(f'Distribution of {feature} for Error Instances')
              plt.legend()
              plt.show()
```



From the calibration.sklearn import curve for calibration: creates calibration curves by importing the scikit-learn calibration_curve tool.

import plt from matplotlib.pyplot: uses the Matplotlib package to import data and create visualisations.

Print Maximum and Minimum Values:

print("Min value:", proba_scores.min()): Outputs the proba_scores' lowest value.

print("Max value:", proba_scores.max()): Outputs the proba_scores' highest value. This will probably shed light on the range of likelihoods that have been forecasted.

cost = information['price']: This function retrieves the dataset's 'price' column.

It appears there may be an error and the assignment should be proba_scores = data['predicted_probabilities'] or something similar. proba_scores = data['price'] assigns the 'price' column to proba_scores.

binaries_true_labels = (labels_true == 1).True labels (true_labels) are converted into binary format (0 or 1) using the astype(int) function.

Adjust Forecasted Probabilities: (proba_scores - proba_scores.min()) / (proba_scores.max() - proba_scores.min()) = proba_scores_normalized: produces a range of 0 to 1 by normalising the estimated probabilities (proba_scores).

Determine the Calibration Curve. proba_scores_normalized, n_bins=10, binary_true_labels, prob_true, prob_pred = calibration_curve: utilises the calibration_curve function to calculate the calibration curve. Its inputs are the number of bins, the normalised predicted probability, and the binary true labels.

Plot Curvature Calibration: Using Matplotlib, plot the calibration curve (plt.plot(prob_pred, prob_true, marker='o', label='Calibration Curve')) where prob_pred is the mean predicted probability and prob_true is the fraction of positives.

A diagonal dashed line that represents a perfectly calibrated model is plotted using plt.plot([0, 1], [0, 1], 'k--', label='Perfectly Calibrated').

The x-axis is labelled using plt.xlabel('Mean Predicted Probability') data.

The y-axis is labelled with plt.ylabel('Fraction of Positives').

plt.title("Calibration Curve"): Gives the plot a title.

plt.legend():

In [123]:

```
#4.3(3)

from sklearn.calibration import calibration_curve
import matplotlib.pyplot as plt


print("Min value:", proba_scores.min())
print("Max value:", proba_scores.max())


price = data['price']
proba_scores = data['price']


binary_true_labels = (true_labels == 1).astype(int)

proba_scores_normalized = (proba_scores - proba_scores.min()) / (proba_scores.max() - proba_scores.min())

prob_true, prob_pred = calibration_curve(binary_true_labels, proba_scores_normalized, n_bins=10)

plt.plot(prob_pred, prob_true, marker='o', label='Calibration Curve')
plt.plot([0, 1], [0, 1], 'k--', label='Perfectly Calibrated')
plt.xlabel('Mean Predicted Probability')
plt.ylabel('Fraction of Positives')
plt.title('Calibration Curve')
plt.legend()
plt.show()
```

```
Min value: 120
Max value: 9999999
```