REPORT WRITING

ON

ANALYZING SECONDHAND CAR SALES DATA WITH SUPERVIZED LEARNING MODELS

Artificial Intelligence and Data sciences

INTRODUCTION

Cars are sold with the intention of value and quality. The prices are affected by factors such as year of production, manufacturer, engine sizes, mileage, and fuel type. This makes cars more valuable than another. Machine learning is a field of study that helps hugely in the development of predictive insights. Developing models through training data and prediction of data has really helped to get information on future decisions.

AIM

The objective of this project is to explore how supervised and unsupervised models can predict car prices using car sales data.

The Data Process

Data collection is a method of accumulating and measuring information based on interest in a well-defined manner. This helps to answer research questions, measure outcomes, and test hypotheses.

Data collection

Based on the breakdown of data above, the type of data used in this project is **Secondary Data.** Because the data is gotten from the car sales database source and the type of data collection is a **mixed method** because it contains both the quantitative and qualitative methods.

Data Collection Description

The dataset contains car sales data of 50000 rows with information written as follows;

Manufacturer: This type of data contains the name of manufacturer of a car. E.g., Ford

Year of Manufacture: This data contains the year of manufacture of a car E.g., 2021

Fuel Type: This type of data contains the type of fuel in the car E.g., Petrol

Mileage: This type of data contains the mileage of each car.

Price: This type of data contains the price of the car.

Engine size: This type of data contains the engine size of the car.

Data Acquisition

This is the process of acquiring data from a source, this type of source can be in different format. E.g., CSV, EXCEL, JSON

The type of data format used in this project is a csv format.

Tools and Libraries used in this project

Python programming language was used for the data science process. This language has libraries that perform this process.

The libraries used are as follows;

- Pandas
- NumPy
- Matplotlib
- Scikit Learn library

Pandas: an extremely popular python language library that is majorly used for data acquisition, data cleaning, and data analysis.

NumPy: is a fundamental library that is used for numerical computation. It has an N-dimensional array.

Matplotlib: is an incredibly good library that is used for creating amazing visualizations. It is a plotting library for the Python programming language.

Scikit-learn is a python library that is used for machine learning. This library has all the algorithms needed for your machine learning processes.

Data Acquisition

This is the stage where you bring in data in your desired format. In this project , it is a csv format

```
In [11]: #bringing in my data and storing it in a variable named car_data
         car_data = pd.read_csv('car_sales_data.csv')
         #checking the first 10 rows of my dataframe
         car_data.head(10)
```

Out[11]:

	Manufacturer	Model	Engine size	Fuel type	Year of manufacture	Mileage	Price
0	Ford	Fiesta	1.0	Petrol	2002	127300	3074
1	Porsche	718 Cayman	4.0	Petrol	2016	57850	49704
2	Ford	Mondeo	1.6	Diesel	2014	39190	24072
3	Toyota	RAV4	1.8	Hybrid	1988	210814	1705
4	VW	Polo	1.0	Petrol	2006	127869	4101
5	Ford	Focus	1.4	Petrol	2018	33603	29204
6	Ford	Mondeo	1.8	Diesel	2010	86686	14350
7	Toyota	Prius	1.4	Hybrid	2015	30663	30297
8	VW	Polo	1.2	Petrol	2012	73470	9977
9	Ford	Focus	2.0	Diesel	1992	262514	1049

Figure 1: shows how data is acquired from the file directory.

Basic Data Exploration

In the data acquired, exploring more information is important in cases like, the shape, data type, statistical information.

Exploring more on my data

```
In [12]: #checking my data structure in terms of the shape
         car_data.shape
```

```
Out[12]: (50000, 7)
```

```
In [13]: #checking the columns in the data
         car_data.columns
```

```
Out[13]: Index(['Manufacturer', 'Model', 'Engine size', 'Fuel type',
                'Year of manufacture', 'Mileage', 'Price'],
               dtype='object')
```

```
In [14]: #checking the data types of all the columns in my dataset
         car_data.dtypes
```

```
Out[14]: Manufacturer            object
         Model                   object
         Engine size            float64
         Fuel type               object
         Year of manufacture      int64
         Mileage                  int64
         Price                    int64
         dtype: object
```

```
In [15]: # checking more statistical information on the data
         car_data.describe()
```

Out[15]:

	Engine size	Year of manufacture	Mileage	Price
count	50000.000000	50000.000000	50000.000000	50000.000000
mean	1.773058	2004.209440	112497.320700	13828.903160
std	0.734108	9.645965	71632.515602	16416.681336
min	1.000000	1984.000000	630.000000	76.000000
25%	1.400000	1996.000000	54352.250000	3060.750000
50%	1.600000	2004.000000	100987.500000	7971.500000
75%	2.000000	2012.000000	158601.000000	19026.500000
max	5.000000	2022.000000	453537.000000	168081.000000

Figure2: This shows the basic information of the data

Data Cleaning

There was no uncleaned data in the dataset. Based on the inspection made through basic exploration.

Checking if we generally have null values

```
In [17]: #checking if we have a null values in the data
         car_data.isnull().sum() # The output shows we have no null values

Out[17]: Manufacturer          0
         Model                 0
         Engine size           0
         Fuel type             0
         Year of manufacture   0
         Mileage               0
         Price                 0
         dtype: int64
```

Figure 3: This show an output that indicate that there are null values

Checking the unique values to see if we have a bad formatted value too.

Checking for the Unique fields in each columns

```
In [18]: #Digging Dip into each feature in the dataset
         def CheckUnique(sub_data):
             data = car_data[sub_data].unique() # check for uniqueness in the columns


             return data

In [19]: #checking the uniqueness for Manufacturer Field
         CheckUnique('Manufacturer')

Out[19]: array(['Ford', 'Porsche', 'Toyota', 'VW', 'BMW'], dtype=object)

In [20]: #checking the uniqueness for Model Field
         CheckUnique('Model')

Out[20]: array(['Fiesta', '718 Cayman', 'Mondeo', 'RAV4', 'Polo', 'Focus', 'Prius',
                'Golf', 'Z4', 'Yaris', '911', 'Passat', 'M5', 'Cayenne', 'X3'],
               dtype=object)

In [21]: #checking the uniqueness for Engine Size Field
         CheckUnique('Engine size')

Out[21]: array([1. , 4. , 1.6, 1.8, 1.4, 1.2, 2. , 2.2, 2.4, 2.6, 3.5, 4.4, 3. ,
                5. ])

In [22]: #checking the uniqueness for Fuel Type Field
         CheckUnique('Fuel type')

Out[22]: array(['Petrol', 'Diesel', 'Hybrid'], dtype=object)

In [23]: #checking the uniqueness for Year of manufacture Field
         CheckUnique('Year of manufacture')

Out[23]: array([2002, 2016, 2014, 1988, 2006, 2018, 2010, 2015, 2012, 1992, 1990,
                2007, 1998, 1989, 2003, 1987, 1996, 2009, 2017, 2005, 2020, 2001,
                2008, 1995, 2000, 1994, 2004, 1997, 1993, 1991, 1999, 1985, 2021,
                2013, 1986, 1984, 2019, 2011, 2022])
```

Figure4: This shows a reuseable function that helps to check for unique value per column of the dataset.

Exploratory Data analysis of the car sales dataset

While performing exploration, I performed a distribution check on each of the datasets by applying reuseable functions for each column to see the distribution on these features.

```
I  click to expand output; double click to hide output  of all numerical columns
def checkDistribution(data):
    chart = sns.histplot(car_data[data] , kde=True);
    plt.title(f'Distribution chart of {data}');
    plt.show();

    return chart;
```

Figure5: This shows a reuseable function that is used to get the distribution chart of the column passed as a parameter.

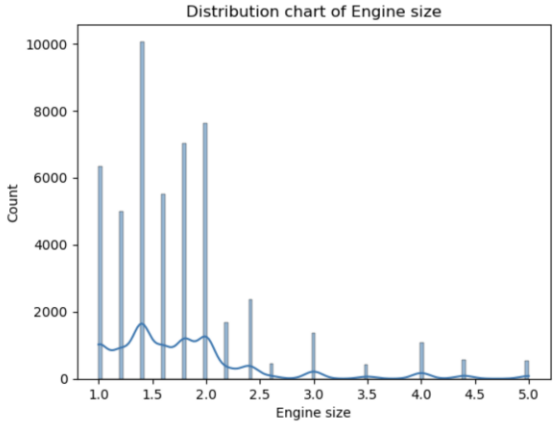


Figure6: This shows the distribution chart for engine size.

```
In [28]: # checking the specified Mileage distribution
         checkDistribution('Mileage');
```

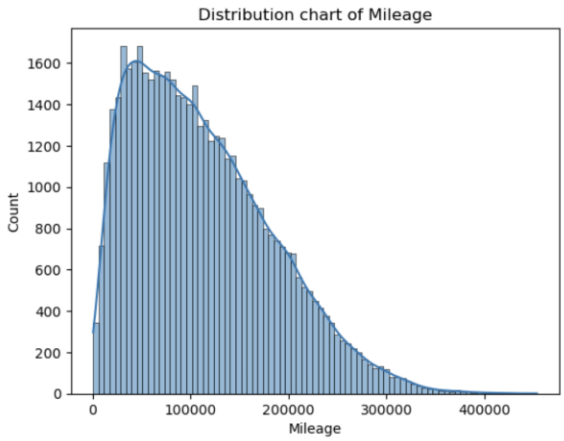


Figure7: This shows the distribution chart for mileage

```
In [29]: # checking the specified Mileage distribution
         checkDistribution('Price');
```

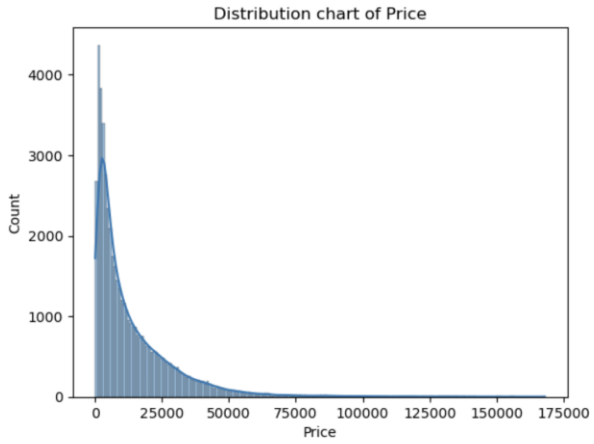


Figure8: This shows the distribution chart for price

```
In [30]: # exploring through charts in all columns
         def numericCharts(data1,data2):
             chart = sns.scatterplot(x = data1 , y = data2, data = car_data);
             plt.title(f'Scatter Plot of {data1} and {data2}');
             plt.xlabel(f'{data1}');
             plt.ylabel(f'{data2}');
             plt.show();


             return chart
```

Figure9: This shows the reusable function used to plot the relationship between two features.

```
In [31]: # create a visualization(scatterplot) of two continuous variables
         numericCharts('Mileage', 'Price');
```

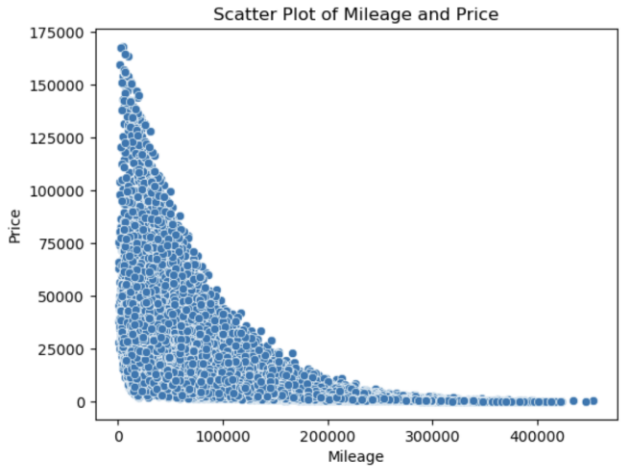
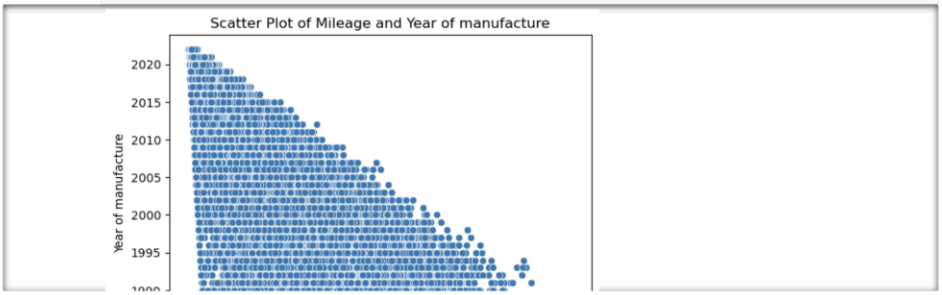
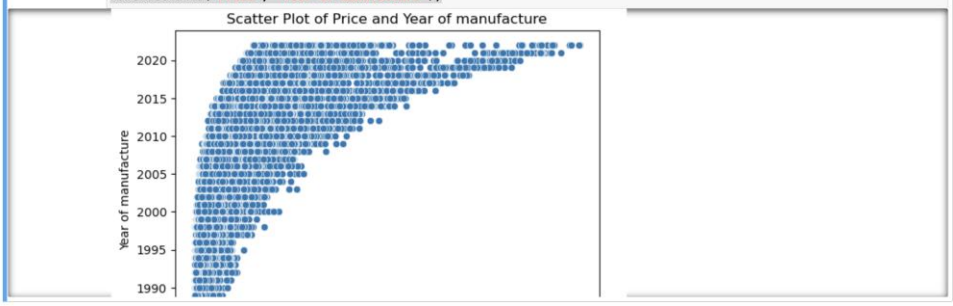


Figure 10: This shows the relationship between the mileage and the price

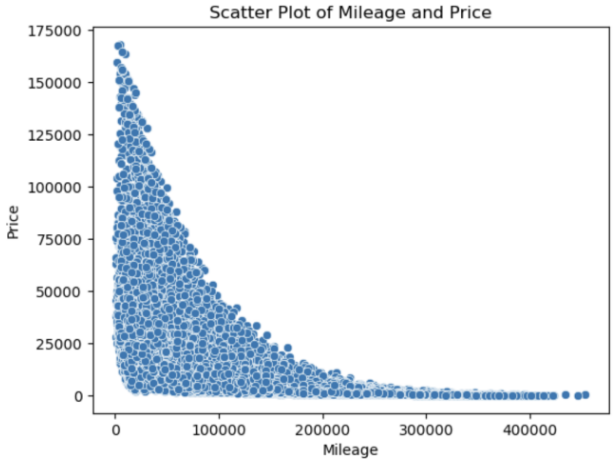
In [24]:
```
# create a visualization(scatterplot) of two continuous variables
numericCharts('Mileage', 'Year of manufacture');
```



In [23]:
```
# create a visualization(scatterplot) of two continuous variables
numericCharts('Price', 'Year of manufacture');
```



In [31]:
```
# create a visualization(scatterplot) of two continuous variables
numericCharts('Mileage', 'Price');
```

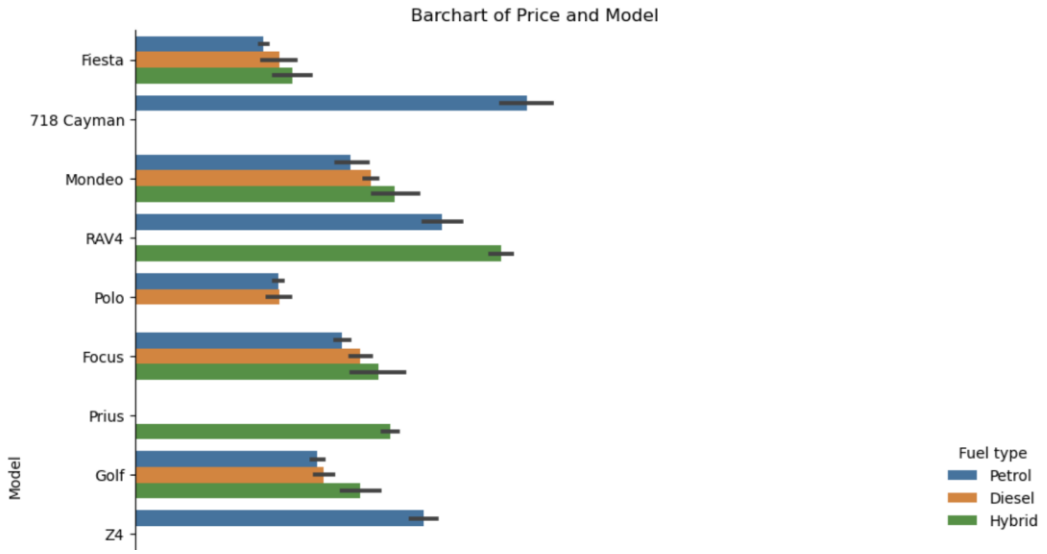


In [30]:
```
# exploring through charts in all columns
def numericCharts(data1,data2):
    chart = sns.scatterplot(x = data1 , y = data2, data = car_data);
    plt.title(f'Scatter Plot of {data1} and {data2}');
    plt.xlabel(f'{data1}');
    plt.ylabel(f'{data2}');
    plt.show();


    return chart
```

```
In [25]: # exploring through charts in all columns
         def categoricalCharts(data1,  data2 ,data3):
             chart = sns.catplot(x = data1 , y = data2, data = car_data, hue = data3 , height = 9, kind="bar");
             plt.title(f'Barchart of {data1} and {data2}');
             plt.xlabel(f'{data1}');
             plt.ylabel(f'{data2}');



             return chart
```

```
In [26]: categoricalCharts('Price' , 'Model' ,'Fuel type')
```

```
Out[26]: <seaborn.axisgrid.FacetGrid at 0x14e58ec90>
```



Predictive Insights and Visualizations

Compare regression models that predict the price of a car based on a single numerical input feature.

To predict the price of a car based on a single numerical input, we must extract the datasets into predictors and a target.

Before that we must know the data with the best predictors. The year of manufacture is the best predictor of car prices. It is obvious that newer models are always expensive compared to old model cars. Expect for rare cases like the old vintage cars that are quite expensive too.

Checking the car price best predictiors.

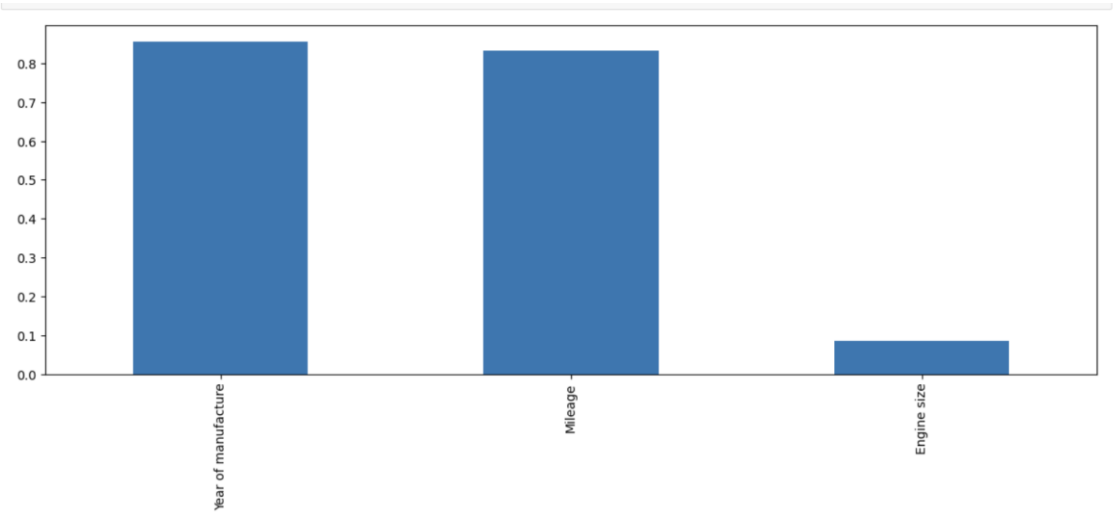
```
#extract out the predictors
features = [  'Engine size' , 'Mileage', 'Year of manufacture']
X = car_data[features]
```

```
#extracting the variable for the target
y = car_data["Price"]
```

```
# splitting the data into train and test
x_train , x_test , y_train , y_test = train_test_split(X,y , test_size = 0.20)
```

Feature Selection

```
# writing a function check for the best feature
def featureSelection():
    # feature selection engineering
    mutual_info = mutual_info_regression(x_train , y_train)
    mutual_info = pd.Series(mutual_info)
    mutual_info.index = x_train.columns
    mutual_info.sort_values(ascending=False)
    mutual_info.sort_values(ascending=False).plot.bar(figsize=(15,5))


featureSelection()
```



After getting the best correlation with the price. We will have to write a function that will help to predict the car price with any column name as a parameter to the function. This function will also plot the appropriate charts for justification.

Predicting The Price Using A Single Feature

In [31]:
```
def predictPrice(data_x , data_y):

    # declare the variables for predictors and target
    X = car_data[data_x]
    y = car_data[data_y]

    # Reshape X to have two dimensions
    X = X.values.reshape(-1, 1)

    #divide my data into train test split
    x_train , x_test , y_train , y_test = train_test_split(X , y , test_size = 0.20)

     # initialize the regression model
    lm = LinearRegression()

    # fit the regression model
    lm.fit(x_train , y_train)

    # predict your values
    y_pred = lm.predict(x_test)

     # create a DataFrame for plotting
    plot_data = pd.DataFrame({data_x: x_test.flatten(), data_y: y_test, 'Predicted': y_pred})

     # plotting using seaborn
    sns.lmplot(x=data_x, y=data_y, data=plot_data, aspect=2, height=6)
    plt.title('Linear Regression Model')
    plt.show()

    #printing the evaluation metrics
    print(f"The evaluation metrics using the Rscore is {r2_score(y_test , y_pred):.2f}")
```

In this function, so many process will go on they are as follows;

Feature extraction: data will be extracted as predictors and target

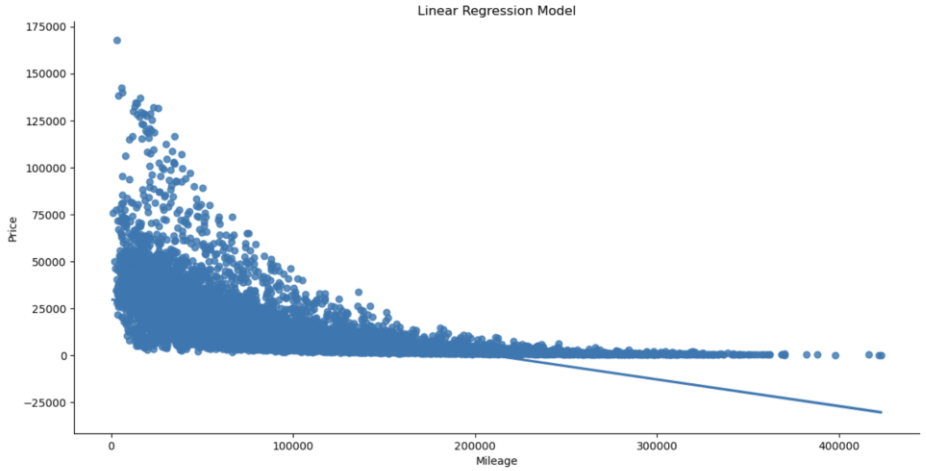
Data splitting: data will be divided into train and test data with its ratio.

Model creation, fitting and prediction: this will lead to creating a regression method. Then fit the model, predict on test data.

Visualization: Charts will be created for justification's sake.

Evaluation: The use of r2 score to evaluate the efficiency of our models.

In [32]:
```
predictPrice('Mileage' , 'Price')
```

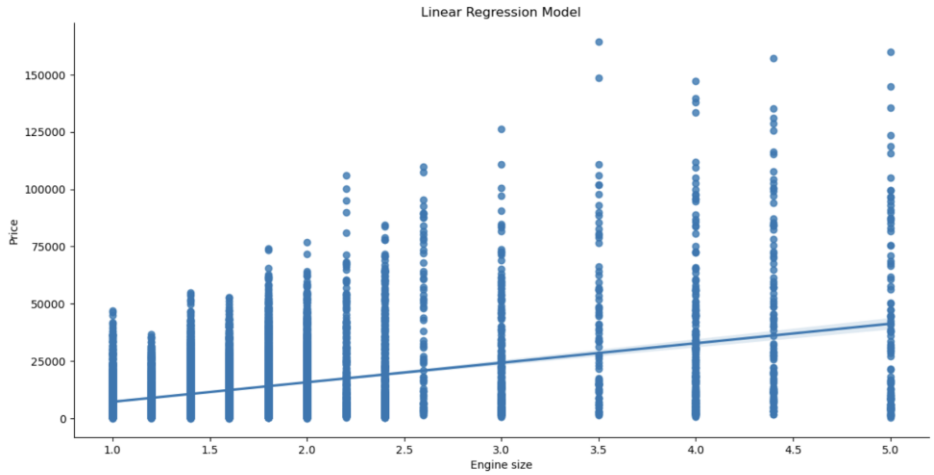


The evaluation metrics using the Rscore is 0.40

In [33]:
```
predictPrice('Year of manufacture' , 'Price')
```

The evaluation metrics using the Rscore is 0.49

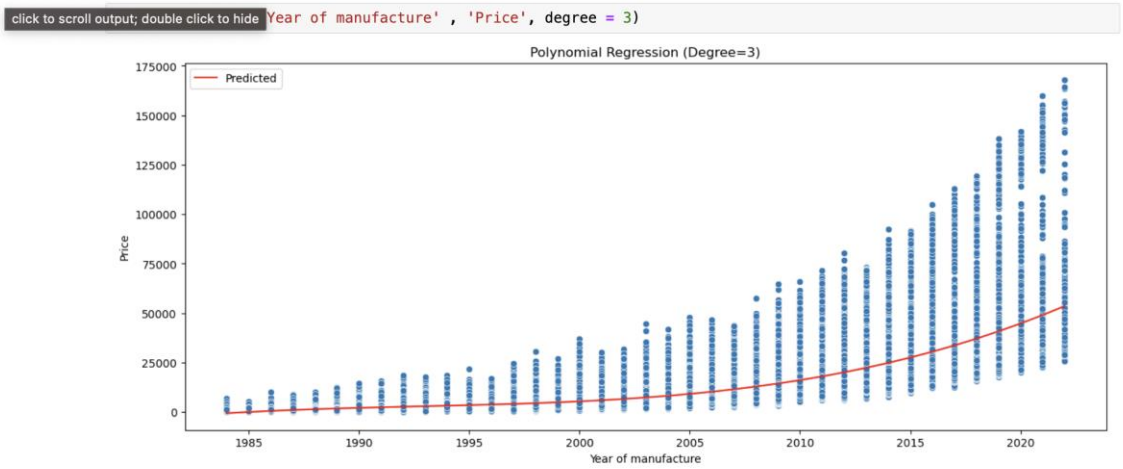
In [34]:
```
predictPrice('Engine size' , 'Price')
```



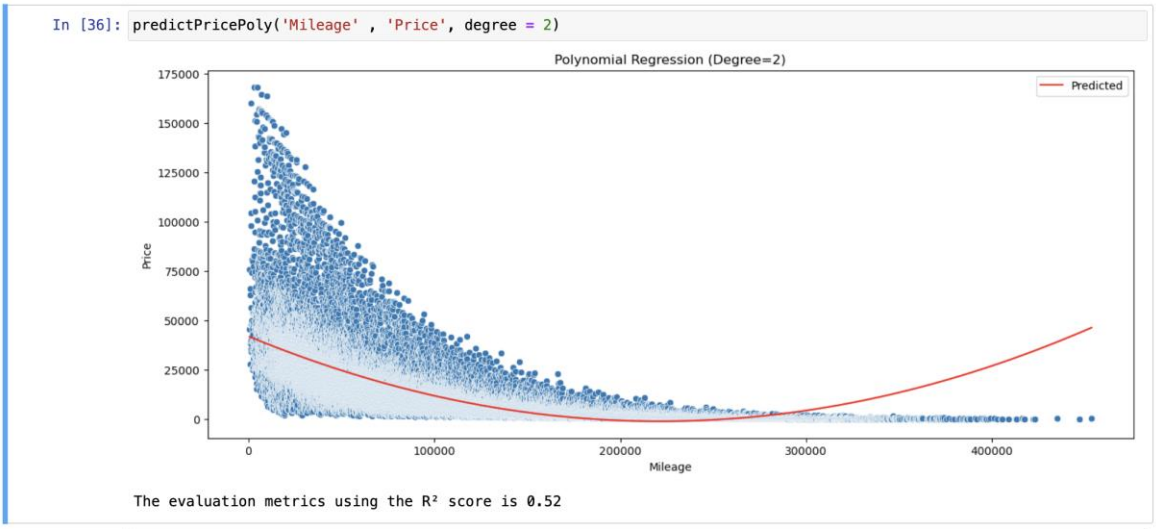
The evaluation metrics using the Rscore is 0.15

For each numerical input feature, is the price better fit by a linear model or by a non-linear (e.g., polynomial) model?

Price is not a good fit with a linear model because it performs woefully compared to the polynomial regression model. The figure below justifies the model's performance.

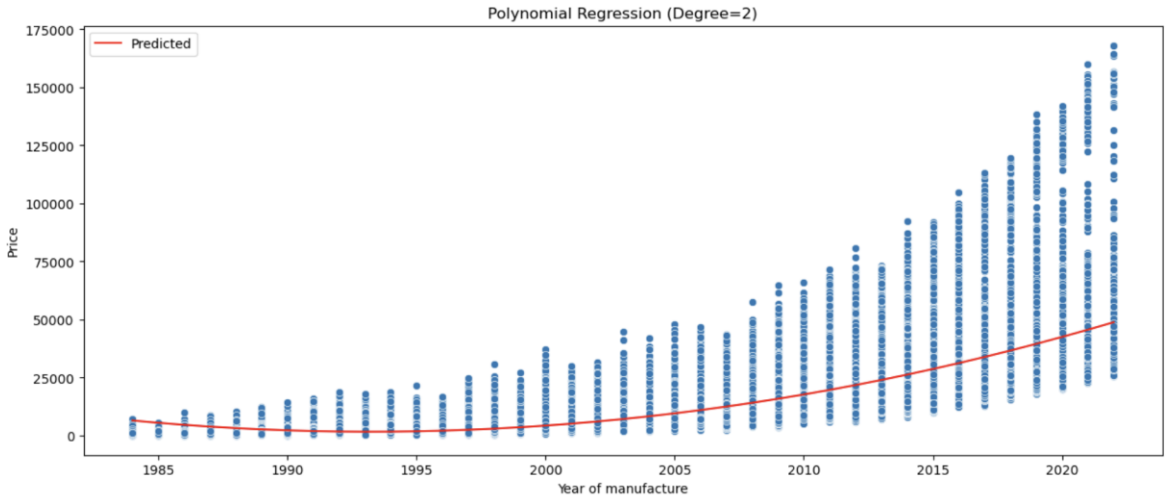


The evaluation metrics using the R^2 score is 0.62



The evaluation metrics using the R^2 score is 0.52

In [38]:
```
predictPricePoly('Year of manufacture' , 'Price', degree = 2)
```

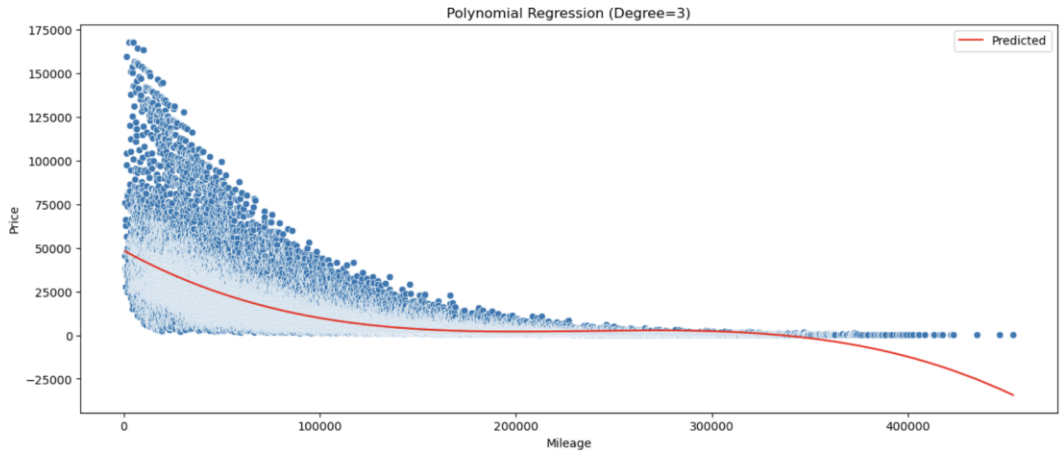


The evaluation metrics using the R^2 score is 0.60

Using the Polynomial features to write a function

In [35]:
```
def predictPricePoly(data_x, data_y, degree=2, test_size=0.2, random_state=None):

    # declare the variables for predictors and target
    X = car_data[data_x]
    y = car_data[data_y]

    # Reshape X to have two dimensions
    X = X.values.reshape(-1, 1)

    # Converting features into polynomial features
    polynomial_f = PolynomialFeatures(degree=degree)
    X_poly = polynomial_f.fit_transform(X)

    #spliting the data from train to test split
    x_train, x_test, y_train, y_test = train_test_split(X_poly, y, test_size=test_size, random_state=random_state)

    #initializing the linear models
    lm = LinearRegression()
    #fitting the models
    lm.fit(x_train, y_train)
    #predicting the model
    y_pred = lm.predict(x_test)

    # For plotting, create a DataFrame with original and predicted values
    plot_data = pd.DataFrame({data_x: X.flatten(), data_y: y, 'Predicted': lm.predict(X_poly)})
    plot_data = plot_data.sort_values(by=data_x)  # Sort for better visualization

    #plotting using the seaborn library
    fig,ax = plt.subplots(figsize = (15,6))
    sns.scatterplot(x=data_x, y=data_y, data=plot_data , ax=ax)
    plt.plot(plot_data[data_x], plot_data['Predicted'], color='red', label='Predicted')
    plt.xlabel(data_x)
    plt.ylabel(data_y)
    plt.title(f'Polynomial Regression (Degree={degree})')
    plt.legend()
    plt.show()

    print(f"The evaluation metrics using the R² score is {r2_score(y_test, y_pred):.2f}")
```

```
In [37]: predictPricePoly('Mileage' , 'Price', degree = 3)
```




```
The evaluation metrics using the R² score is 0.54
```

Consider regression models that take multiple numerical variables as input features to predict the price of a car.

Using a Multiple Feature for a better Accuracy or not

```
In [42]: def predictPricePolyMult(data_x, data_y, degree=2, test_size=0.2, random_state=None):

             # declare the variables for predictors and target
             X = car_data[data_x]
             y = car_data[data_y]

             # In case we are using a multiple features, reshape X to be a 2D
             if isinstance(data_x, list):
                 X = X[data_x].values

             # Converting features into polynomial features
             polynomial_f = PolynomialFeatures(degree=degree)
             X_poly = polynomial_f.fit_transform(X)

             #spliting the data from train to test split
             x_train, x_test, y_train, y_test = train_test_split(X_poly, y, test_size=test_size, random_state=random_state)

             #initializing the linear models
             lm = LinearRegression()
             #fitting the models
             lm.fit(x_train, y_train)
             #predicting the model
             y_pred = lm.predict(x_test)

             # For plotting, create a DataFrame with original and predicted values
             plot_data = pd.DataFrame({data_y: y, 'Predicted': lm.predict(X_poly)})
             plot_data = plot_data.sort_values(by=data_y)  # Sort for better visualization

             #Labelling
             plt.figure(figsize=(15, 6))
             plt.scatter(y, plot_data['Predicted'])
             plt.plot(plot_data[data_y], plot_data['Predicted'], color='red', label='Predicted Line')
             plt.xlabel(f'Actual {data_y}')
             plt.ylabel(f'Predicted {data_y}')
             plt.title(f'Polynomial Regression (Degree={degree})')
             plt.show()

             print(f"The evaluation metrics using the R² score is {r2_score(y_test, y_pred):.2f}")
```

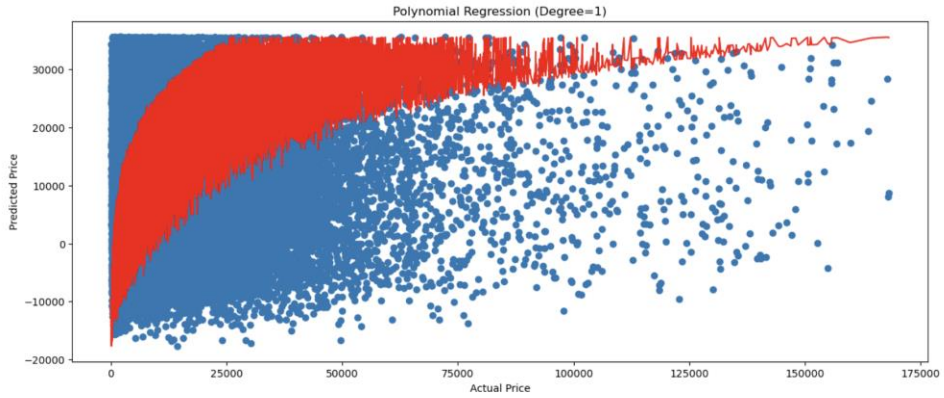
Figure 19: This is a reuseable function that can be used to predict, visualize, and evaluate multiple linear regression.

Does the inclusion of multiple input features improve the accuracy of the model's prediction compared to the single-input feature models that you explored in part (a)?

The chart while using this function on multiple features is produced below.

Visualizing the data for a 2 features and a Linear Regression model

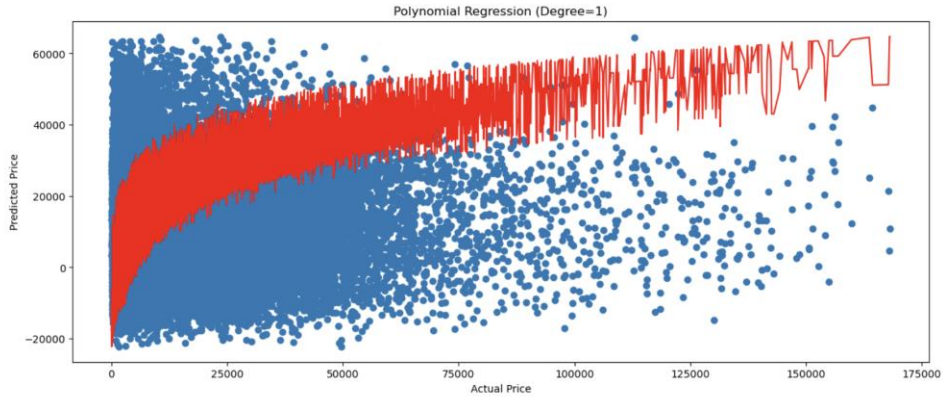
```
In [43]: predictPricePolyMult(['Year of manufacture' , 'Mileage'] , 'Price' , degree = 1)
```



The evaluation metrics using the R^2 score is 0.52

Visualizing the data for a 3 features and a Linear Regression model ¶

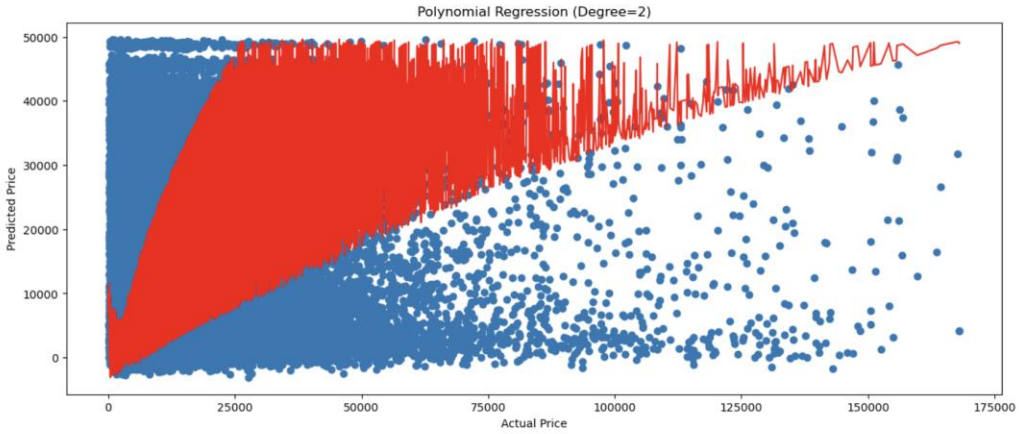
```
In [44]: predictPricePolyMult(['Year of manufacture' , 'Mileage' , 'Engine size'] , 'Price' , degree = 1)
```



The evaluation metrics using the R^2 score is 0.69

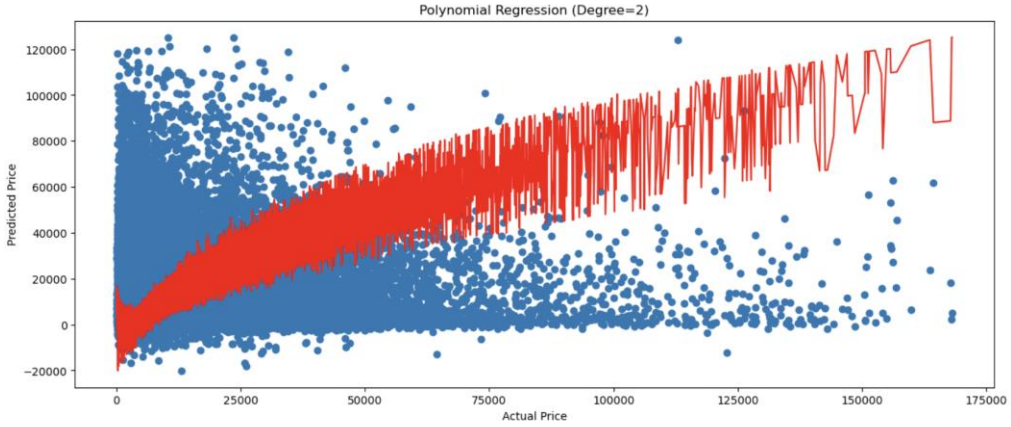
Visualizing the data for a 2 features and a Polynomial Regression model

In [45]:
```
predictPricePolyMult(['Year of manufacture' , 'Mileage'] , 'Price' , degree = 2)
```



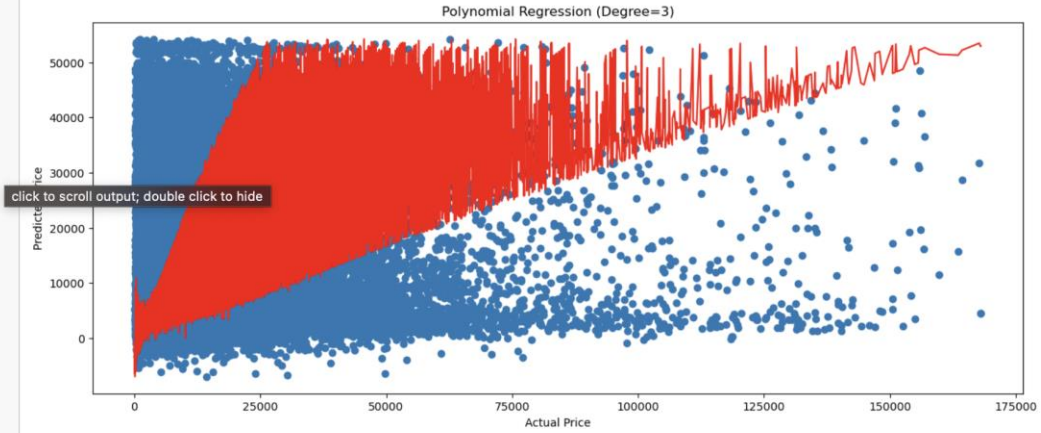
The evaluation metrics using the R^2 score is 0.63

In [46]:
```
predictPricePolyMult(['Year of manufacture' , 'Mileage' , 'Engine size'] , 'Price' , degree = 2)
```



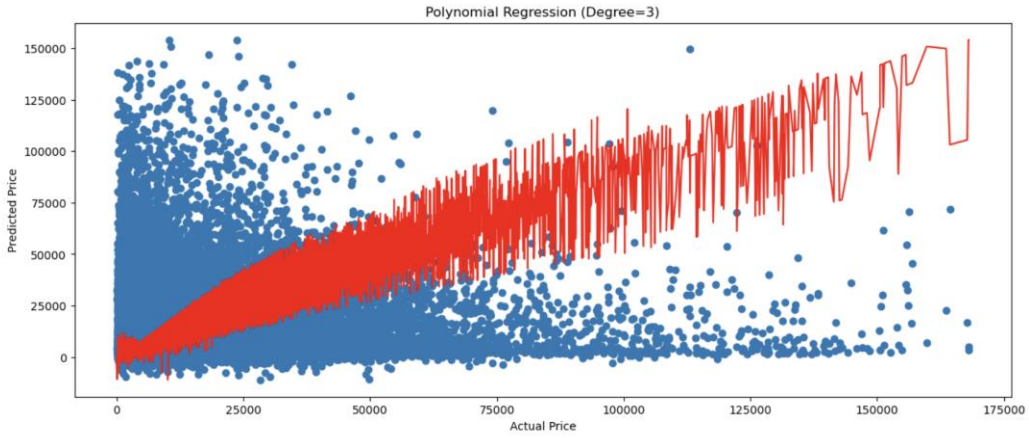
The evaluation metrics using the R^2 score is 0.90

```
In [47]: predictPricePolyMult(['Year of manufacture' , 'Mileage'] , 'Price' , degree = 3)
```




```
The evaluation metrics using the R² score is 0.63
```

```
In [48]: predictPricePolyMult(['Year of manufacture' , 'Mileage' , 'Engine size'] , 'Price' , degree = 3)
```




```
The evaluation metrics using the R² score is 0.92
```

The chart above shows that more features on a polynomial degree of 3 give better accuracy than a single feature or a few features on a polynomial regression.

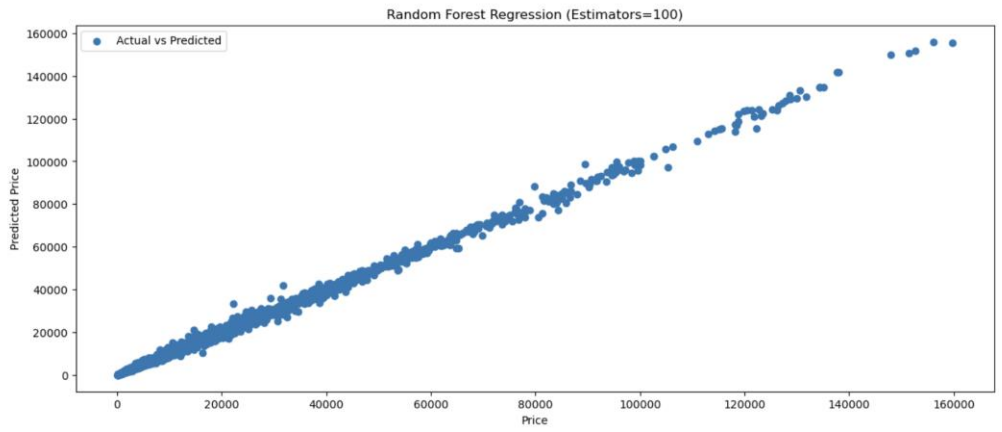
In parts (a) and (b) you only considered models that use the numerical variables from the dataset as inputs. However, there are also several *categorical* variables in the dataset that are likely to affect the price of the car. Now train a regression model that uses all relevant input variables (both *categorical* and *numerical*) to predict the price (e.g. a Random Forest Regressor model). Does this improve the accuracy of your results?

Based on the question, we will write a reusable function to create a Random Forest regressor to predict the price by including the categorical variable too.

This operation is a special method used in feature engineering called label encoder, which helps convert all categorical variables to numbers.

```
In [49]: def predictPriceWithCategorical(data_x, data_y, test_size=0.2, random_state=None,  n_estimators=100):
             #declare the car sales data global
             global car_data

             # Selecting numerous features and copying them in a data variable
             features = data_x + [data_y]
             data = car_data[features].copy()

             #specify the predictors that might be multiple or single
             X = data[data_x]
             y = data[data_y]

             # If using more than feature, reshape X to be a 2D array
             if isinstance(data_x, list):
                 X = X[data_x]

             # get the categorical columns which are not int or float
             categorical_features = list(X.select_dtypes(include=['object']).columns)

             # Apply label encoding to categorical columns
             label_encoders = {}
             for feature in categorical_features:
                 le = LabelEncoder()
                 X[feature] = le.fit_transform(X[feature])
                 label_encoders[feature] = le

             #splitting the data using the train-test-split
             x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=test_size, random_state=random_state)

             # Initialize and Fit the RandomForestRegressor
             rf_model = RandomForestRegressor(n_estimators=n_estimators, random_state=random_state)
             rf_model.fit(x_train, y_train)

             # Predict the outcome
             y_pred = rf_model.predict(x_test)

             # Create a DataFrame with Actual and predicted values
             plot_data = pd.DataFrame({data_y: y_test, 'Predicted': y_pred})
             plot_data = plot_data.sort_values(by=data_y)  # Sort for better visualization


             # Scatter plot of actual against the predicted
             plt.figure(figsize=(15, 6))
             plt.scatter(plot_data[data_y], plot_data['Predicted'], label='Actual vs Predicted')
             plt.xlabel(f'{data_y}')
             plt.ylabel(f'Predicted {data_y}')
             plt.title(f'Random Forest Regression (Estimators={n_estimators})')
             plt.legend()
             plt.show()
```

```
In [50]: #running the function with categorical variables
         predictPriceWithCategorical(['Year of manufacture' , 'Mileage' , 'Engine size' , 'Manufacturer' , 'Model', 'Fuel typ
```




```
The evaluation metrics using the R² score is 1.00
```

These figures show that there is a perfect 100% using all the dataset to train and test the models.

Develop an Artificial Neural Network (ANN) model to predict the price of a car based on all the available information from the dataset. How does its performance compare to the other supervised learning models that you have considered? Discuss your choices for the architecture of the neural network that you used and describe how you tuned the hyperparameters in your model to achieve the best performance.

Creating an artificial neural network model, there are 4 stages in it.

Constructor stage

Compilation stage

Training stage

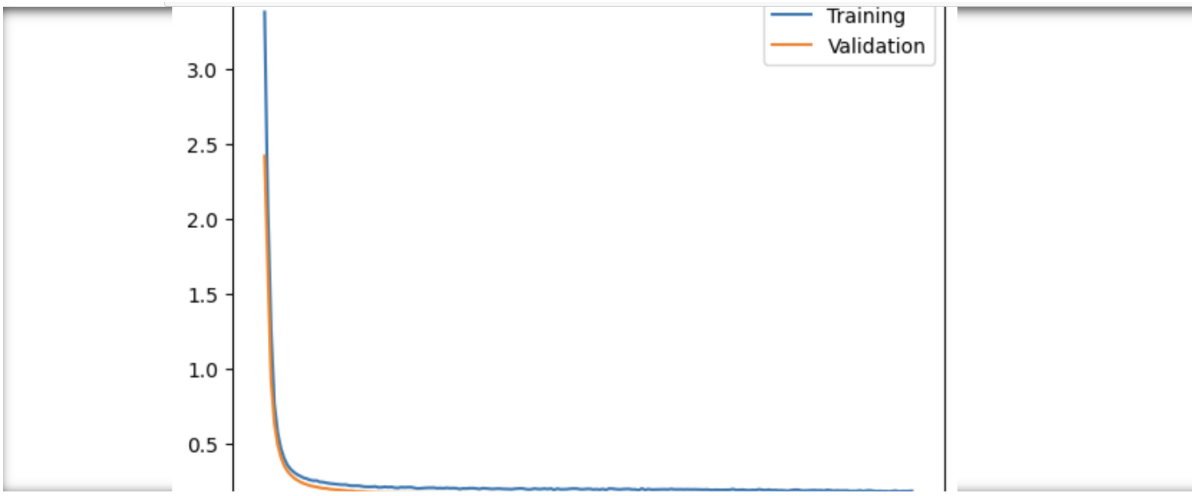
Evaluation stage

```
def getPredictedPrice(data_x, data_y, test_size=0.2, random_state=None):

    # Selecting numerous features and copying them in a data variable
    features = data_x + [data_y]
    data = car_data[features].copy()

    #specify the predictors that might be multiple or single
    X = data[data_x]
    y = data[data_y]

    # If using more than feature, reshape X to be a 2D array
    if isinstance(data_x, list):
        X = X[data_x]

    # get the categorical columns which are not int or float
    categorical_features = list(X.select_dtypes(include=['object']).columns)

    # Apply label encoding to categorical columns
    label_encoders = {}
    for feature in categorical_features:
        le = LabelEncoder()
        X[feature] = le.fit_transform(X[feature])
        label_encoders[feature] = le

    #intializing the scaler function
    scale = MinMaxScaler()
    scale.fit(X)

   #splitting the data using the train-test-split
    x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=test_size, random_state=random_state)
    #Constructor stage

     #scaling transformation for the x_train and x_test
    x_train_scaled = scale.transform(x_train)

     #scaling transformation for the x_train and x_test
    x_test_scaled = scale.transform(x_test)

    #Building my neural network by intializing the model
```

```python
#Building my neural network by intializing the model
model = Sequential()
model.add(Dense(units=64, input_dim=X.shape[1], activation="relu"))
model.add(Dropout(0.2))
model.add(Dense(units=64, activation="relu"))
model.add(Dense(units=1, activation="linear"))
model.summary()

#Compilation and Training stage 1
model.compile(optimizer="adam", loss='mean_squared_error', metrics='mean_squared_error')
early_stopping = EarlyStopping(monitor = 'val_loss', patience = 20)
history = model.fit(x = x_train_scaled , y = y_train , batch_size = None, epochs = 200,
                    verbose = "auto", validation_split = 0.1, callbacks = [early_stopping])
y_pred = model.predict(x_test_scaled)

#plotting
history_df = pd.DataFrame(history.history)
plt.plot(history_df["loss"], label = "Training")
plt.plot(history_df["val_loss"], label = "Validation")
plt.legend()
plt.show()

#Evaluation method
r2 = r2_score(y_test, y_pred)
print(f"The evaluation metric using the R² score is {r2:.2f}")

return y_pred
```

In [52]:
```python
#running the deep learning model to predict the price using my ANN architecture
getPredictedPrice(['Year of manufacture' , 'Mileage' , 'Engine size' , 'Manufacturer' , 'Model', 'Fuel type'] , 'Pri
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 64)                448

 dropout (Dropout)           (None, 64)                0

 dense_1 (Dense)             (None, 64)                4160

 dense_2 (Dense)             (None, 1)                 65

=================================================================
Total params: 4673 (18.25 KB)
Trainable params: 4673 (18.25 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Epoch 1/200
1125/1125 [==============================] - 4s 1ms/step - loss: 337525216.0000 - mean_squared_error: 337525216.000
```



```
0     25     50     75     100    125    150    175    2

The evaluation metric using the R² score is 0.93

Out[52]: array([[ 1286.3588],
                [15237.273 ],
                [13018.641 ],
                ...,
                [11291.666 ],
                [25719.527 ],
                [ 3574.881 ]], dtype=float32)
```

Based on the Result of the evaluation metrics, it is 96% accurate. Compared to the regression models with lower Accuracy. The use of Dropout deactivates some neurons for swift forward and backward propagation. Building my ANN model, I used an architectural model with 3 dense layers with two 64 neurons and 1 neuron then with the use of 2 relu activation functions and the last one linear. After that applying a dropout of 0.2 to the model.

Based on the results of your analysis, what is the best model for predicting the price of a car and why? You should use suitable figures and evaluation metrics to support your conclusions.

Based on my model evaluation, the use of the Ensemble model really gave my model a good accuracy of 100%. The linearity was undiluted. Check Exercise 1C for the figure and the R2 score for better explanation.

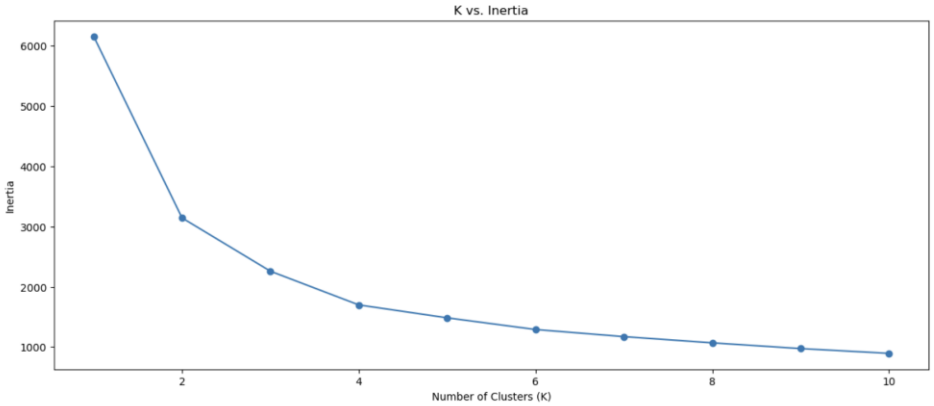
Use the *k*-Means clustering algorithm to identify clusters in the car sales data. Consider different combinations of the numerical variables in the dataset to use as input features for the clustering algorithm. In each case, what is the optimal number of clusters (*k*) to use and why? Which combination of variables produces the best clustering results? Use appropriate evaluation metrics to support your conclusions.

use K-Means clustering to predict the price of the car

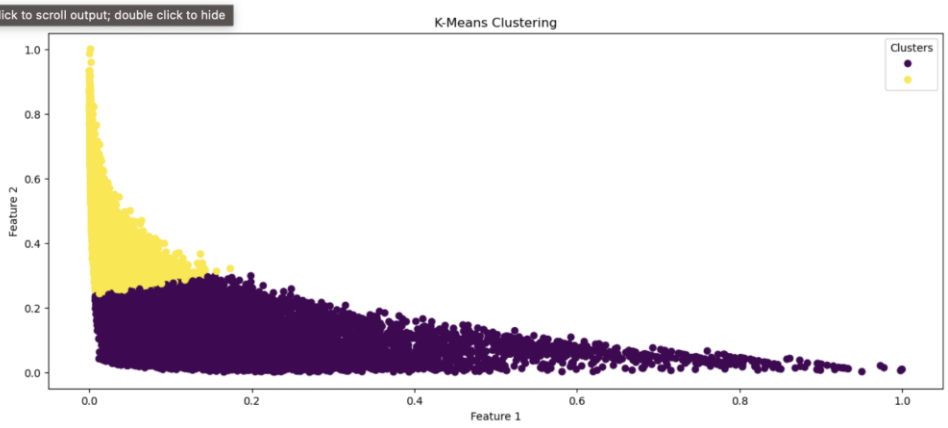
In [57]:
```
def segmentWithClust(data_x):
    global car_data

    # Specify the predictors that might be multiple or single
    X = car_data[data_x]

    # If using more than one feature, reshape X to be a 2D array
    if isinstance(data_x, list):
        X = X[data_x]

    # Get the categorical columns which are not int or float
    categorical_features = list(X.select_dtypes(include=['object']).columns)

    # Apply label encoding to categorical columns
    label_encoders = {}
    for feature in categorical_features:
        le = LabelEncoder()
        X[feature] = le.fit_transform(X[feature])
        label_encoders[feature] = le

    # Initializing the scaler function
    scale = MinMaxScaler()
    x_scaled = scale.fit_transform(X)

    # Elbow method
    K = []

    # Using a for loop
    for k in range(1, 11):
        kmeans = KMeans(n_clusters=k, random_state=0, n_init=10, init='k-means++')
        kmeans.fit(x_scaled)
        K.append(kmeans.inertia_)

    # Plotting the elbow method
    plt.figure(figsize=(15, 6))
    plt.plot(range(1, 11), K, marker='o')
    plt.title('K vs. Inertia')
    plt.xlabel('Number of Clusters (K)')
    plt.ylabel('Inertia')
    plt.show()

    # Determine optimal number of clusters based on elbow method
    optimal_k = 2  # Change this to the value you obtained from the elbow method
```

```
# Plotting the elbow method
plt.figure(figsize=(15, 6))
plt.plot(range(1, 11), K, marker='o')
plt.title('K vs. Inertia')
plt.xlabel('Number of Clusters (K)')
plt.ylabel('Inertia')
plt.show()

# Determine optimal number of clusters based on elbow method
optimal_k = 2  # Change this to the value you obtained from the elbow method

# Perform KMeans clustering with the optimal number of clusters
kmeans = KMeans(n_clusters=optimal_k, random_state=0, n_init=10, init='k-means++')
clusters = kmeans.fit_predict(x_scaled)

# Scatter plot for visualizing clusters
plt.figure(figsize=(15, 6))
scatter = plt.scatter(x_scaled[:, 0], x_scaled[:, 1], c=clusters, cmap='viridis', label='Clusters')
plt.title('K-Means Clustering')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.legend(handles=scatter.legend_elements()[0], title='Clusters')
plt.show()

# Silhouette Score for evaluation
silhouette_avg = silhouette_score(x_scaled, clusters)
print(f"Silhouette Score: {silhouette_avg:.4f}")

# Calculate the Davies Bouldin index for model evaluation
db_score = davies_bouldin_score(x_scaled, clusters)
print(f"Davies Bouldin Score: {db_score:.4f}")
```

In [61]:
```
segmentWithClust(['Year of manufacture' , 'Mileage' , 'Engine size'])
```



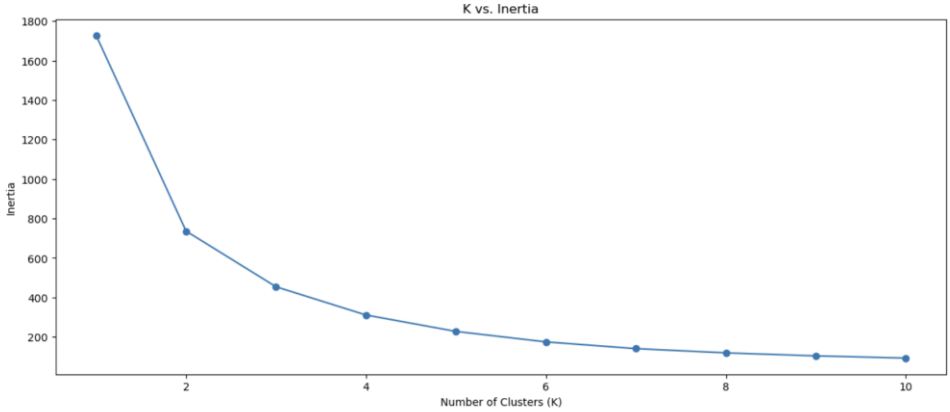
click to scroll output; double click to hide



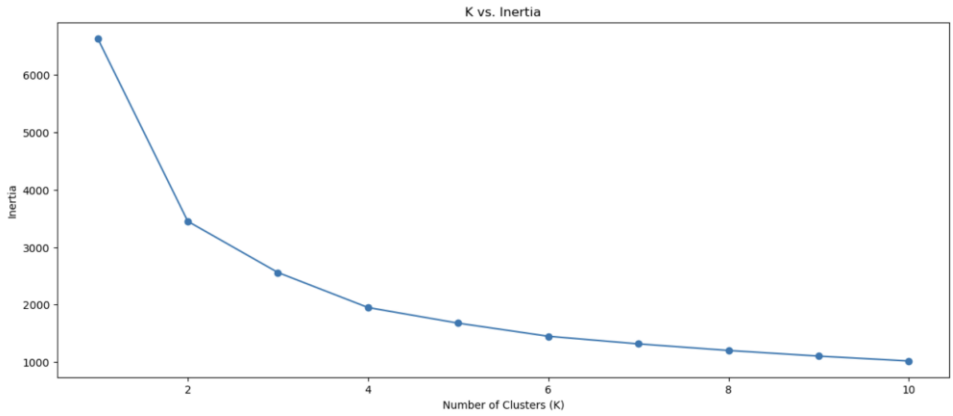

```
Silhouette Score: 0.5168
Davies Bouldin Score: 0.6809
```

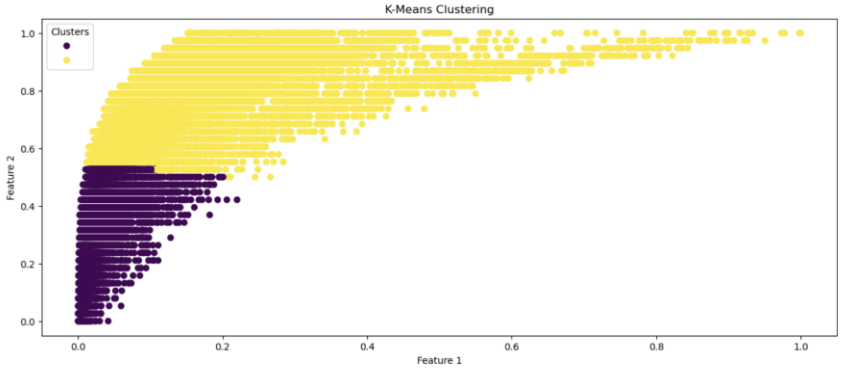
In [60]:
```
#running the deep learning model to predict the price using my ANN architecture
```

In [59]:
```
segmentWithClust(['Price' , 'Mileage'])
```

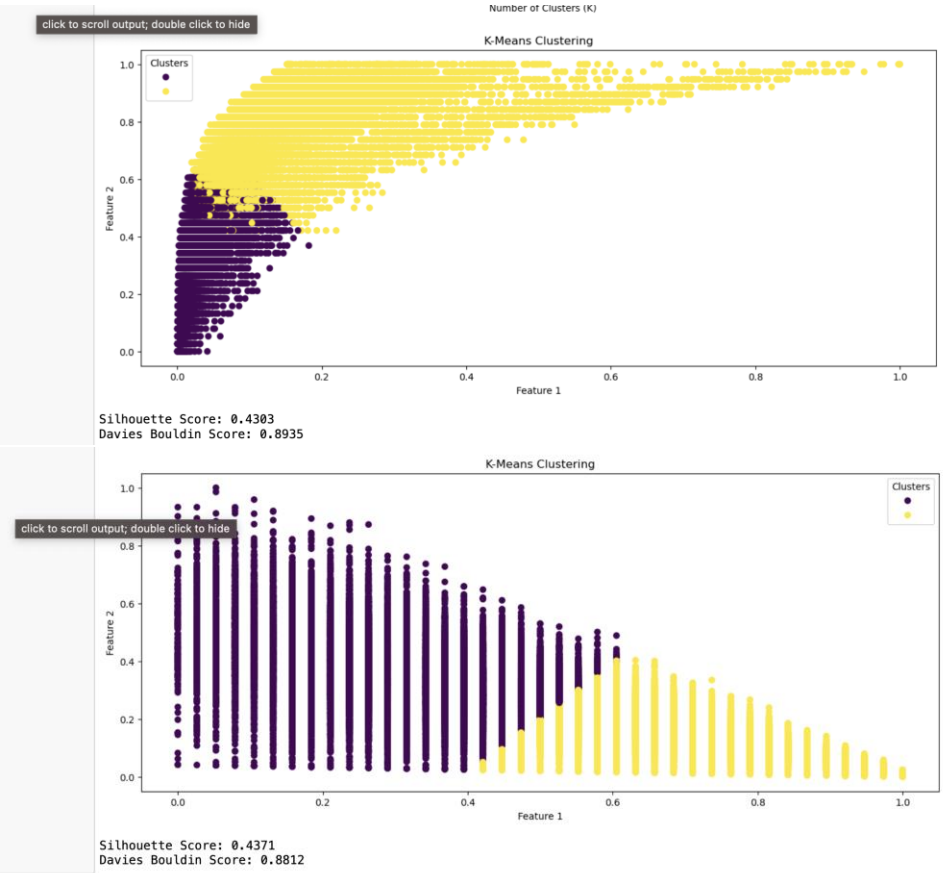


In [60]:
```
#running the deep learning model to predict the price using my ANN architecture
segmentWithClust(['Price' , 'Year of manufacture' , 'Mileage' , 'Engine size'])
```






```
Silhouette Score: 0.5818
Davies Bouldin Score: 0.5708
```



What is the optimal number of clusters (k) to use and why

The optimal number of clusters is 2, This was because the edge of the elbow chart is pointing to the number of optimal 2. This shows why i should use 2 as the number of clusters.

Which combination of variables produces the best clustering results?

Based on the result of the models, it is shown that the clusters between the Price and the Year of manufacture have the fairest relationship compared to others. Judging by the Silhouette Score of 0.5818 and Davies Bouldin Score of 0.5708. This shows that an object is well matched to its own cluster and poorly matched to neighboring clusters and are less distinct and more dispersed.

Compare the results of the *k*-Means clustering model from part (f) to at least one other clustering algorithm. Which algorithm produces the best clustering? Use suitable evaluation metrics to justify your answer.

Comparing this clustering method with another Clustering method

In [66]:
```
def segmentWithDBSCAN(data_x, eps=0.5, min_samples=5, visualize=True):
    global car_data

    # Specify the predictors that might be multiple or single
    X = car_data[data_x]

    # If using more than one feature, reshape X to be a 2D array
    if isinstance(data_x, list):
        X = X[data_x]

    # Get the categorical columns which are not int or float
    categorical_features = list(X.select_dtypes(include=['object']).columns)

    # Apply label encoding to categorical columns
    label_encoders = {}
    for feature in categorical_features:
        le = LabelEncoder()
        X[feature] = le.fit_transform(X[feature])
        label_encoders[feature] = le

    # Initializing the scaler function
    scale = MinMaxScaler()
    x_scaled = scale.fit_transform(X)

    # Use PCA for dimensionality reduction
    pca = PCA(n_components=2)
    x_pca = pca.fit_transform(x_scaled)

    # Use K-Means for initial centroids
    kmeans = KMeans(n_clusters=min_samples)
    kmeans.fit(x_pca)
    initial_centroids = kmeans.cluster_centers_

    # Using DBSCAN for density-based clustering
    dbscan = DBSCAN(eps=eps, min_samples=min_samples, n_jobs=-1)
    clusters = dbscan.fit_predict(x_pca)

    # Visualize clusters (scatter plot for the first two features)
    if visualize:
        plt.figure(figsize=(15, 6))
        plt.scatter(x_pca[:, 0], x_pca[:, 1], c=clusters, cmap='viridis', label='Clusters')
        plt.scatter(initial_centroids[:, 0], initial_centroids[:, 1], c='red', marker='X', label='Initial Centroids')
        plt.title('DBSCAN Clustering')
        plt.xlabel('Principal Component 1')
        plt.ylabel('Principal Component 2')
        plt.legend()
        plt.show()

    # Silhouette Score for evaluation (Note: Silhouette Score might not be applicable for DBSCAN)
    try:
        silhouette_avg = silhouette_score(x_pca, clusters)
        print(f"Silhouette Score: {silhouette_avg:.4f}")
    except ValueError:
        print("Silhouette Score is not applicable for DBSCAN.")

    # Calculate the Davies Bouldin index for model evaluation
    db_score = davies_bouldin_score(x_pca, clusters)
    print(f"Davies Bouldin Score: {db_score:.4f}")
```

In [ ]:
```
segmentWithDBSCAN(['Price','Year of manufacture'])
```

In [ ]:
```
segmentWithDBSCAN(['Price' , 'Mileage'])
```

In [ ]:
```
segmentWithDBSCAN(['Price' , 'Year of manufacture' , 'Mileage' , 'Engine size'])
```

In [ ]:
```
segmentWithDBSCAN(['Year of manufacture' , 'Mileage' , 'Engine size'])
```

Running both hierarchical clustering / DBscan clustering had very high computation power, I couldn't run it on the pc. In other to compare both clustering methods

Conclusion

The integration of machine learning in predicting car prices not only allows for a comprehensive analysis of diverse factors but also empowers decision-makers in the automotive industry with actionable insights for informed and strategic decision-making.

References

Syed Muhammad Sajjad Kabir

Method of Data Collection

https://www.researchgate.net/publication/325846997_METHODS_OF_DATA_COLLECTION

Malcolm Chisholm

Defining data acquisition and why it matters

https://www.firstsanfranciscopartners.com/blog/defining-data-acquisition-importance/

Nikita Duggal

Top 10 Python Libraries in 2022

https://www.simplilearn.com/top-python-libraries-for-data-science-article