FIT 1043 Introduction to Data Science

Assignment 2

Lang Zolyn

30719704

Introduction

In this modern era, artificial intelligence is of paramount importance in developing the autonomous industry. Machine learning is very important as it will bring tremendous convenience to future generations. By using AI, many resources such as human capital, time consumption can be conserved. In this assignment, we will be focusing on the vehicle classification based on the data set provided containing all the necessary information about different vehicles. In order to make use of the data fully, many techniques need to be optimized throughout this assignment, namely describing data based on the basic statistics, data wrangling, implementing clustering algorithms, and data classification. The above techniques are mandatory as they can retrieve and extract data effectively.

Importing the necessary libraries

In [1]:
```
import numpy as np
import pandas as pd
import random
import matplotlib.pyplot as plt
import regressiondemo as rd
from sklearn.cluster import KMeans
from pandas import DataFrame, Series
from IPython.display import Image
from io import StringIO
import pydotplus
from sklearn import preprocessing
from sklearn import tree
%matplotlib inline
vehicle=pd.read_csv("FIT1043-vehicle-classifier.csv")
test=pd.read_csv("FIT1043-kaggle-test-data.csv")
train=pd.read_csv("FIT1043-kaggle-train-data.csv")
```

Description of data and data wrangling

In [2]:
```
vehicle.shape
```

Out[2]:
```
(156, 14)
```

This dataset contains 156 rows and 14 columns.

In [3]:
```
vehicle.isnull().sum()
```

Out[3]:
```
Manufacturer              0
Model                     0
Vehicle_class             0
Vehicle_alt_class       133
US_vehicle_type           0
Engine_size (litres)      0
Horsepower                0
Wheelbase                 0
Width                     0
Length                    0
Height                    0
Curb_weight               1
Fuel_capacity             0
Fuel_efficiency           2
dtype: int64
```

There is 133 missing values in the column Vehicle_alt_class, 1 missing value in the column Curb_weight and 2 missing values in the column Fuel_efficiency.

In [4]:
```
vehicle.head()
```

Out[4]:

	Manufacturer	Model	Vehicle_class	Vehicle_alt_class	US_vehicle_type	Engine_size (litres)	Horsepowe
0	Acura	Integra	Coupe	Sports	Passenger	1.8	14
1	Acura	TL	Sedan	NaN	Passenger	3.2	22
2	Acura	CL	Coupe	NaN	Passenger	3.2	22
3	Acura	RL	Sedan	NaN	Passenger	3.5	21
4	Audi	A4	Sedan	NaN	Passenger	1.8	15

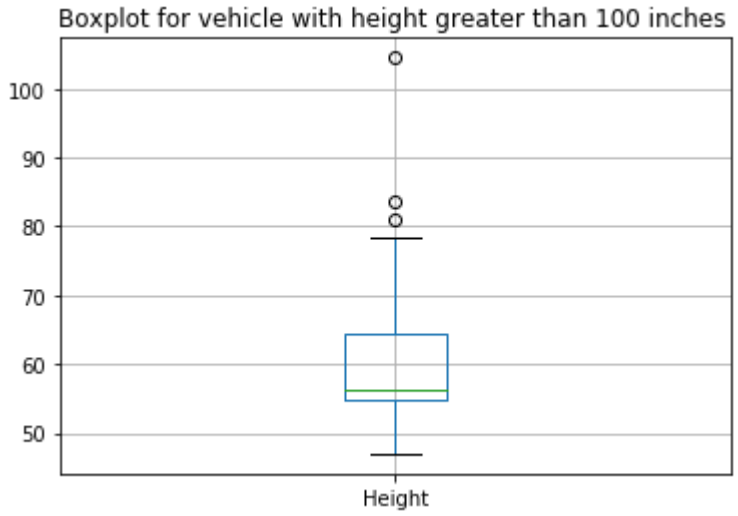
The first five rows are displayed to have an overview of the vehicle dataset.

In [5]:
```
vehicle.isnull().values.any()
```

Out[5]:
```
True
```

The purpose of plotting this boxplot is to detect if there is any outliers

In [6]:
```
vehicle.boxplot(column = 'Height')
plt.title('Boxplot for vehicle with height greater than 100 inches ')
plt.show()
```



In order to detect outliers, i have used a boxplot to visualize on the vehicle data set. From the boxplot, we can see that there is few outliers, the obvious one is vehicle with height greater than 100 inch.

In [7]:
```
vehicle =vehicle[vehicle['Height']<100]
```

In [8]:
```
vehicle.shape
```

Out[8]:
```
(155, 14)
```

After filtering the outlier, now we have 155 rows and 14 columns in the new vehicle data set.

In [9]:
```
vehicle.describe()
```

Out[9]:

	Engine_size (litres)	Horsepower	Wheelbase	Width	Length	Height	Curb_weight	Fue
count	155.000000	155.000000	155.000000	155.000000	155.000000	155.000000	154.000000	
mean	3.043871	184.922581	107.518710	71.118065	187.350968	58.894839	3.379117	
std	1.026094	55.411862	7.659921	3.420592	13.431884	7.053605	0.632412	
min	1.000000	55.000000	92.600000	62.600000	149.400000	47.000000	1.895000	
25%	2.300000	149.000000	103.000000	68.500000	177.550000	54.750000	2.969000	
50%	3.000000	175.000000	107.000000	70.500000	188.000000	56.200000	3.355000	
75%	3.500000	215.000000	112.200000	73.100000	196.150000	64.100000	3.810250	
max	8.000000	450.000000	138.700000	79.900000	224.500000	83.600000	5.572000	

In order to provide a more accurate description of data, the describe function is used.

In [10]:
```
vehicle.isnull().sum()
```

Out[10]:
```
Manufacturer              0
Model                     0
Vehicle_class             0
Vehicle_alt_class       133
US_vehicle_type           0
Engine_size (litres)      0
Horsepower                0
Wheelbase                 0
Width                     0
Length                    0
Height                    0
Curb_weight               1
Fuel_capacity             0
Fuel_efficiency           2
dtype: int64
```

Since there are null values in the respective columns, the missing values are filled by the mean of the column "Curb_weight" and "Fuel_efficiency". For the column "Vehicle_alt_class", it is more appropriate for us to impute the missing values instead of removing the rows that contain the missing value because there are 133 rows. Therefore, a new category named "ordinary" is used to fill in all the missing values. Pleas note that the values are replaced by the mean according to the respective vehicle class found below

Investigating relationship between columns

In [11]:
```
vehicle.groupby('Vehicle_class').size()
```

Out[11]:
```
Vehicle_class
Coupe     26
MPV       10
SUV       23
Sedan     88
Truck      8
dtype: int64
```

Based on the output, we can observe that most of the vehicle produced are Sedan, followed by Coupe, SUV, MPV and Truck. Most of the vehicles produced are Sedan, this is most likely due to the popular demand of Sedan cars as it is more family friendly compared to the others.

In [105]:
```
total={'Horsepower':{'Horsepower (sum)':'sum'}}
horsepower = vehicle.groupby('Vehicle_class').agg(total).reset_index()
horsepower.columns =horsepower .columns.droplevel(0)
horsepower.rename(columns = {'':'Vehicle class'},inplace = True)
```

In [13]:
```
total2={'Horsepower':{'Horsepower (count)':'count'}}
new_horsepower = vehicle.groupby('Vehicle_class').agg(total2).reset_index()
new_horsepower.columns =new_horsepower.columns.droplevel(0)
new_horsepower.rename(columns = {'':'Vehicle class'},inplace = True)
new_horsepower2=new_horsepower.drop("Vehicle class",axis=1)
```

In [14]:
```
averagefunc= pd.concat([horsepower,new_horsepower2], axis = 1)
```

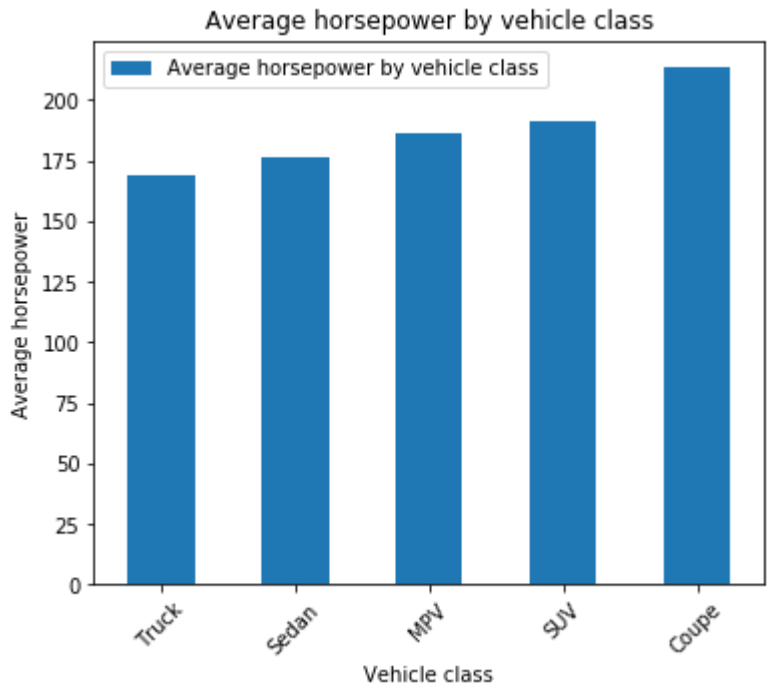
In [15]:
```
averagefunc['Average horsepower by vehicle class'] = averagefunc['Horsepower (su
averagefunc.sort_values(by=['Average horsepower by vehicle class'], inplace=True
remove_count =averagefunc.drop("Horsepower (sum)", axis=1)
remove_sum=remove_count.drop("Horsepower (count)", axis=1)
remove_sum
```

Out[15]:

	Vehicle class	Average horsepower by vehicle class
4	Truck	168.875000
3	Sedan	175.920455
1	MPV	186.400000
2	SUV	191.565217
0	Coupe	213.884615

In [16]:
```
ax=remove_sum.plot.bar(figsize=(6,5))
ax.set_xticklabels(remove_sum['Vehicle class'],rotation=45)
plt.xlabel('Vehicle class')
plt.ylabel('Average horsepower')
plt.title('Average horsepower by vehicle class')
```

Out[16]: Text(0.5, 1.0, 'Average horsepower by vehicle class')



From the bar graph shown, we can clearly see that Coupe has the highest average horsepower which is 218.74 as Coupe consumes a large amount of fuel. A coupe is famous for its big power and high acceleration, people tend to be attracted by its exterior styling.

In [17]:
```
total1={'Curb_weight':{'Curb weight (sum)':'sum'}}
weight =vehicle.groupby('Vehicle_class').agg(total1).reset_index()
weight.columns =weight .columns.droplevel(0)
weight.rename(columns = {'':'Vehicle class'},inplace = True)
```

In [18]:
```
total3={'Curb_weight':{'Curb weight (count)':'count'}}
new_weight= vehicle.groupby('Vehicle_class').agg(total3).reset_index()
new_weight.columns =new_weight.columns.droplevel(0)
new_weight.rename(columns = {'':'Vehicle class'},inplace = True)
new_weight2=new_weight.drop("Vehicle class",axis=1)
```

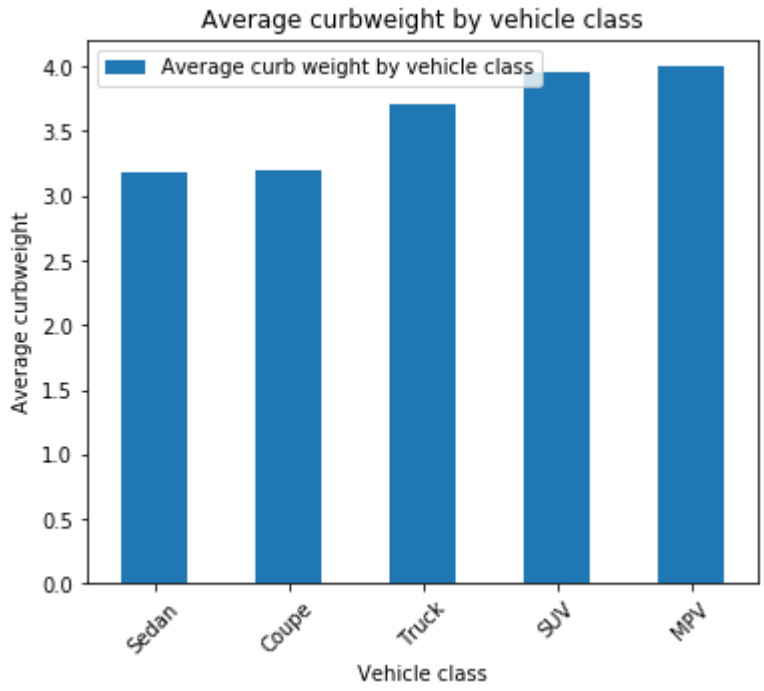
In [19]:
```
averagefunc= pd.concat([weight,new_weight2], axis = 1)
averagefunc['Average curb weight by vehicle class'] = averagefunc['Curb weight (
averagefunc.sort_values(by=['Average curb weight by vehicle class'], inplace=Tru
remove_count =averagefunc.drop("Curb weight (sum)", axis=1)
remove_sum2=remove_count.drop("Curb weight (count)", axis=1)
remove_sum2
```

Out[19]:

	Vehicle class	Average curb weight by vehicle class
3	Sedan	3.176345
0	Coupe	3.200885
4	Truck	3.711750
2	SUV	3.958174
1	MPV	4.008700

In [20]:
```
ax=remove_sum2.plot.bar(figsize=(6,5))
ax.set_xticklabels(remove_sum2['Vehicle class'],rotation=45)
plt.xlabel('Vehicle class')
plt.ylabel('Average curbweight')
plt.title('Average curbweight by vehicle class')
```

Out[20]:
```
Text(0.5, 1.0, 'Average curbweight by vehicle class')
```



According to the bar graph, MPV have the highest curb weight which is around 4 pounds and the vehicle with lowest curb weight is Sedan which is weighted around 3 pounds. Sedan is usually lighter compared to the other vehicles due to its design and size.

In [21]:
```
economy={'Fuel_efficiency':{'Fuel efficiency (sum)':'sum'}}
fuel_economy= vehicle.groupby('Vehicle_class').agg(economy).reset_index()
fuel_economy.columns =fuel_economy.columns.droplevel(0)
fuel_economy.rename(columns = {'':'Vehicle class'},inplace = True)
```

In [22]:
```
economy1={'Curb_weight':{'Fuel efficiency (count)':'count'}}
fuel_economy2= vehicle.groupby('Vehicle_class').agg(economy1).reset_index()
fuel_economy2.columns =fuel_economy2.columns.droplevel(0)
fuel_economy2.rename(columns = {'':'Vehicle class'},inplace = True)
fuel_economy2=fuel_economy2.drop("Vehicle class",axis=1)
```

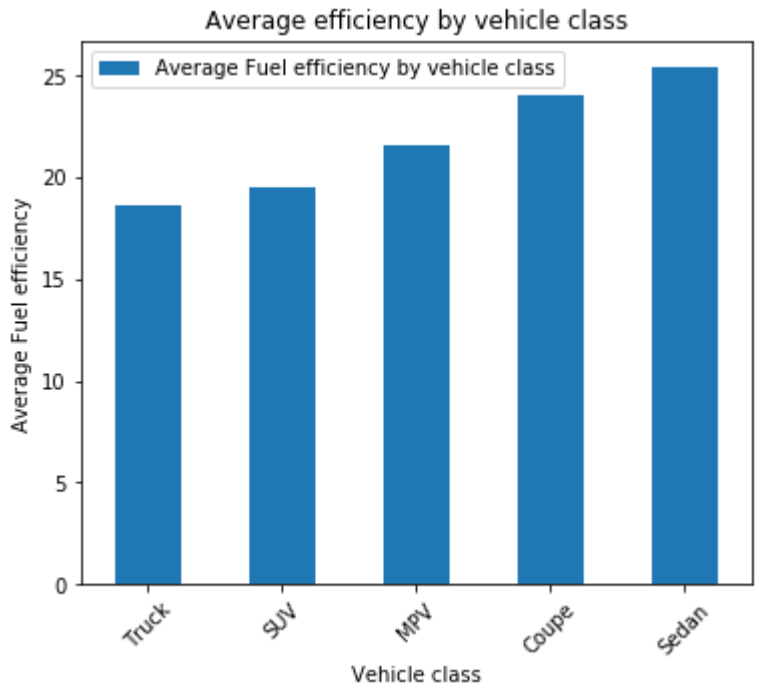
In [23]:
```
averagefunc= pd.concat([fuel_economy,fuel_economy2], axis = 1)
averagefunc['Average Fuel efficiency by vehicle class'] = averagefunc['Fuel effi
averagefunc.sort_values(by=['Average Fuel efficiency by vehicle class'], inplace
remove_count =averagefunc.drop("Fuel efficiency (sum)", axis=1)
ave_fuel=remove_count.drop("Fuel efficiency (count)", axis=1)
ave_fuel
```

Out[23]:

	Vehicle class	Average Fuel efficiency by vehicle class
4	Truck	18.625000
2	SUV	19.478261
1	MPV	21.600000
0	Coupe	24.000000
3	Sedan	25.436782

In [24]:
```
ax=ave_fuel.plot.bar(figsize=(6,5))
ax.set_xticklabels(ave_fuel['Vehicle class'],rotation=45)
plt.xlabel('Vehicle class')
plt.ylabel('Average Fuel efficiency')
plt.title('Average efficiency by vehicle class')
```

Out[24]:
```
Text(0.5, 1.0, 'Average efficiency by vehicle class')
```



The bar graph shows that Sedan has the highest average fuel efficiency, this means that the fuel consumption for Sedan vehicles is the lowest. In other words, Sedan vehicles are considered more eco-friendly, the sales for sedan vehicles are usually higher compared to the other vehicles because it is very convenient in terms of its cargo space and fuel consumption.

In [25]:
```
vehicle.isnull().values.any()
```

Out[25]:
```
True
```

Filling in the missing values using the average value found according to different vehicle classes.

In [26]:
```
vehicle['Vehicle_alt_class'] = vehicle['Vehicle_alt_class'].fillna('Ordinary')
vehicle['Curb_weight'] = vehicle['Curb_weight'].fillna(3.178)
vehicle['Fuel_efficiency'] = vehicle['Fuel_efficiency'].fillna(24.0)
```

Clustering

For clustering purposes, the columns "Vehicle_class", "Vehicle_alt_class" and "US_vehicle_type" are removed.

In [27]:
```
removing_columns =vehicle.drop("Vehicle_class", axis=1)
removing_columns2=removing_columns.drop("Vehicle_alt_class", axis=1)
vehicle_2=removing_columns2.drop("US_vehicle_type", axis=1)
vehicle_2.head()
```

Out[27]:

	Manufacturer	Model	Engine_size (litres)	Horsepower	Wheelbase	Width	Length	Height	Curb_weigh
0	Acura	Integra	1.8	140	101.2	67.3	172.4	52.6	2.63
1	Acura	TL	3.2	225	108.1	70.3	192.9	56.1	3.51
2	Acura	CL	3.2	225	106.9	70.6	192.0	54.7	3.47
3	Acura	RL	3.5	210	114.6	71.4	196.6	56.5	3.85
4	Audi	A4	1.8	150	103.0	68.2	178.0	55.7	2.99

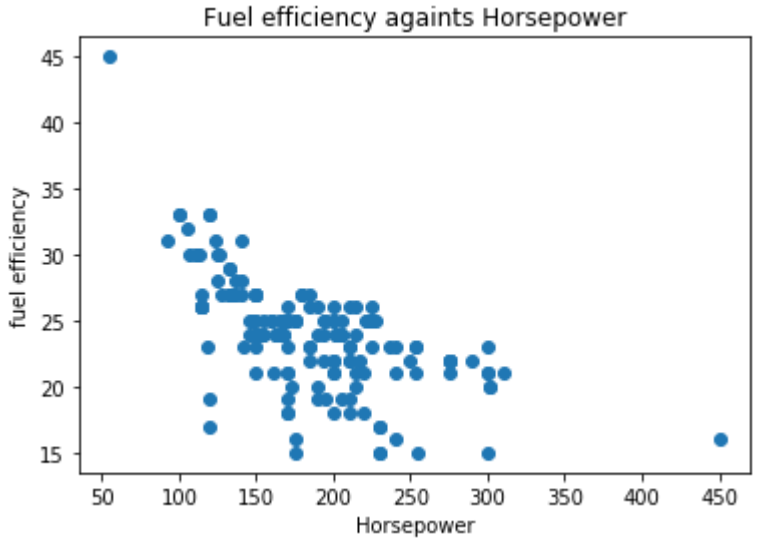
Un-supervised machine learning

Unsupervised machine learning happens when there is no specific function for the data, it learns from test data that has not been classified, categorized, or labeled. In other words, we do not have a corresponding output variable. One of the methods for unsupervised machine learning is data clustering, it is a common technique for statistical data analysis. It can be done by determining the K-means for each cluster and group the distinct values into different clusters. This can be achieved by grouping them according to their similarities and pattern. Unsupervised machine learning can be described in a way that when a newborn baby is born, he/she has not been told that a dog is a dog, but in years of growing up, he/she has observed that the animal with those particular physical characteristics is a dog. This means that the machine has not been given a specific method of function for it to produce the output but rather the machine observes and group the pattern of the data by itself.

Explanation for the chosen clustering inputs

The inputs for clustering that I have chosen is Horsepower and Fuel efficiency. The reason for choosing these two columns is because we would like to figure out what is the pattern and similarities of the data points, by applying k-means clustering, we will be able to find the subgroups among them. In order to investigate further, we can observe the result from the K-means clustering.

In [28]:
```
plt.scatter(x =vehicle_2['Horsepower'], y =vehicle_2['Fuel_efficiency'])
plt.title('Fuel efficiency againts Horsepower')
plt.xlabel('Horsepower')
plt.ylabel('fuel efficiency')
plt.show()
```

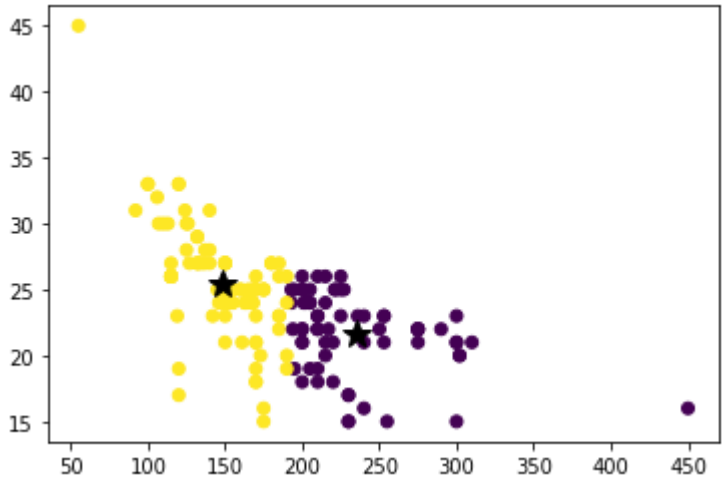


The diagram above displayed how the graph looks like before applying K-means.

Grouping them into 2 clusters

In [29]:
```
kmeans = KMeans(n_clusters=2, init='random').fit(vehicle_2[['Horsepower','Fuel_e
plt.scatter(x=vehicle_2['Horsepower'], y=vehicle_2['Fuel_efficiency'], c=kmeans.
plt.plot(kmeans.cluster_centers_[:,0], kmeans.cluster_centers_[:,1], 'k*', marke
```

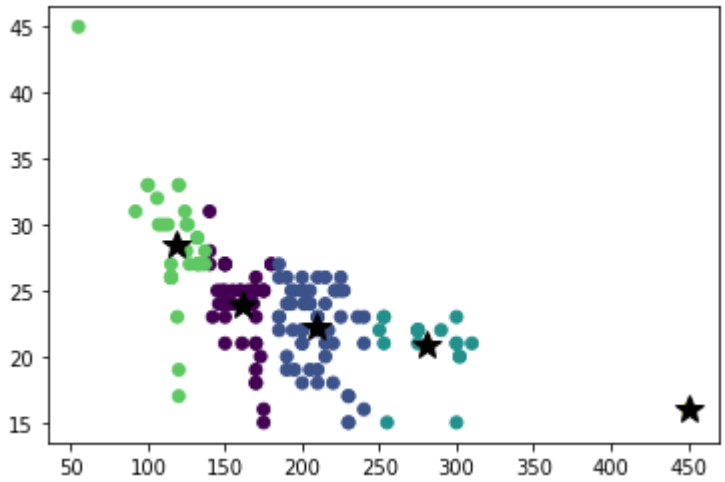
Out[29]:
```
[<matplotlib.lines.Line2D at 0x19023d5ed48>]
```



Grouping them into 5 clusters

In [30]:
```
kmeans = KMeans(n_clusters=5, init='random').fit(vehicle_2[['Horsepower','Fuel_e
plt.scatter(x=vehicle_2['Horsepower'], y=vehicle_2['Fuel_efficiency'], c=kmeans.
plt.plot(kmeans.cluster_centers_[:,0], kmeans.cluster_centers_[:,1], 'k*', marke
```

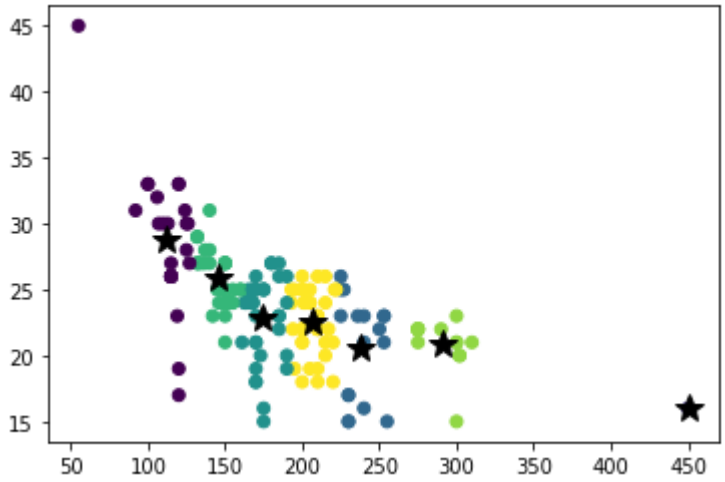
Out[30]:
```
[<matplotlib.lines.Line2D at 0x190244e2d08>]
```



Grouping them into 7 clusters

In [31]:
```
kmeans = KMeans(n_clusters=7, init='random').fit(vehicle_2[['Horsepower','Fuel_e
plt.scatter(x=vehicle_2['Horsepower'], y=vehicle_2['Fuel_efficiency'], c=kmeans.
plt.plot(kmeans.cluster_centers_[:,0], kmeans.cluster_centers_[:,1], 'k*', marke
```

Out[31]:
```
[<matplotlib.lines.Line2D at 0x19024555b88>]
```



Explanation of clustering output

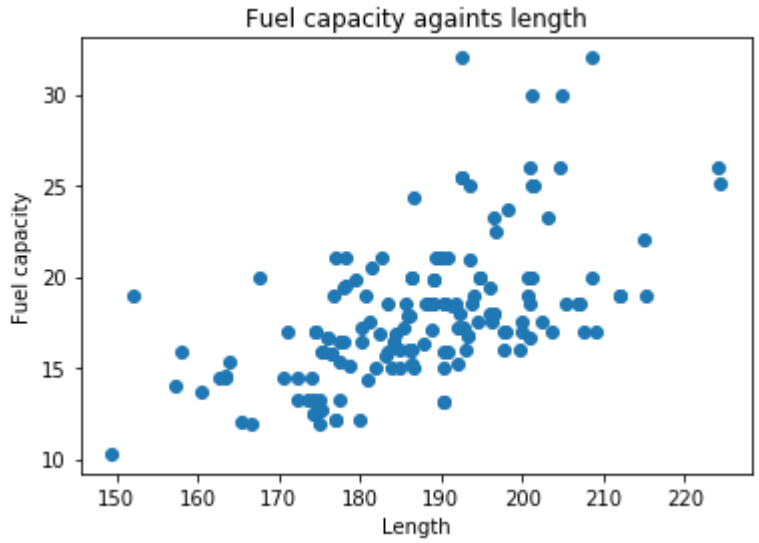
The data is grouped into a different number of clusters. From the clustering output, we can clearly see that there is a high correlation between column horsepower and fuel efficiency. It seems like the higher the horsepower, the more the fuel efficiency. This indicates that there is a strong relationship between the two variables since high correlation means that the neighboring points belong to the same cluster, this is not a good measure in terms of cluster validity. Also, my clusters are not separated clearly in the scatterplot, this means that the purity of my cluster is not very close to 1. My cluster is very close to the other cluster, this means that the chosen inputs for my clustering are not the most ideal one.

Clustering with improved sets of input

For the improved sets of input, I decided to go with two columns which are Length and Fuel capacity as they are less correlated. The Independent variable will work better in K-means clustering because the algorithm will partition them into groups by assigning labels to them, the ones with the same label will fall into the same cluster.

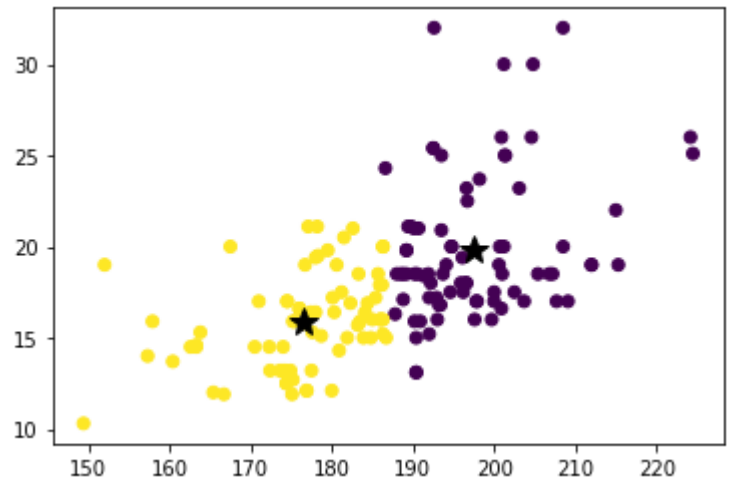
Repeating the same process with improved sets of input

In [32]:
```
plt.scatter(x =vehicle_2['Length'], y =vehicle_2['Fuel_capacity'])
plt.title('Fuel capacity againts length')
plt.xlabel('Length')
plt.ylabel('Fuel capacity')
plt.show()
```



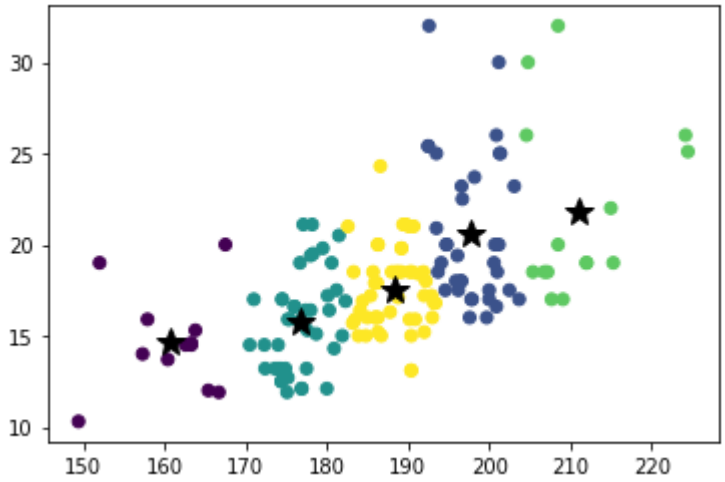
In [33]:
```
kmeans = KMeans(n_clusters=2, init='random').fit(vehicle_2[['Length','Fuel_capac
plt.scatter(x=vehicle_2['Length'], y=vehicle_2['Fuel_capacity'], c=kmeans.labels_
plt.plot(kmeans.cluster_centers_[:,0], kmeans.cluster_centers_[:,1], 'k*', marke
```

Out[33]:
```
[<matplotlib.lines.Line2D at 0x1902465bb88>]
```



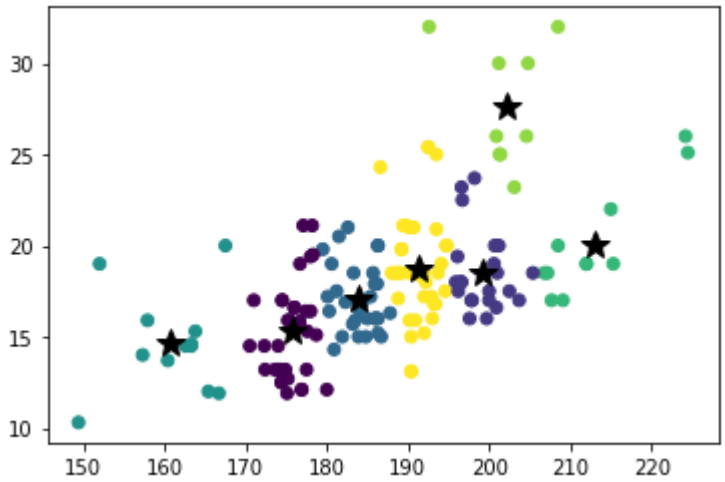
In [34]:
```
kmeans = KMeans(n_clusters=5, init='random').fit(vehicle_2[['Length','Fuel_capac
plt.scatter(x=vehicle_2['Length'], y=vehicle_2['Fuel_capacity'], c=kmeans.labels_
plt.plot(kmeans.cluster_centers_[:,0], kmeans.cluster_centers_[:,1], 'k*', marke
```

Out[34]:
```
[<matplotlib.lines.Line2D at 0x190246c08c8>]
```



In [35]:
```
kmeans = KMeans(n_clusters=7, init='random').fit(vehicle_2[['Length','Fuel_capac
plt.scatter(x=vehicle_2['Length'], y=vehicle_2['Fuel_capacity'], c=kmeans.labels_
plt.plot(kmeans.cluster_centers_[:,0], kmeans.cluster_centers_[:,1], 'k*', marke
```

Out[35]:
```
[<matplotlib.lines.Line2D at 0x1902471b908>]
```



From the improved sets of input, we can see that the clusters are now slightly further from the other clusters. I think that this result is better because the correlation is lower, the clusters now

have higher purity, that is the number of classes of the clusters is easier to distinguish now. Hence, the cluster validity has improved.

Classification

The difference between binary and multi-class classification.

Binary classification is when the algorithm only predicts whether the output belongs in one of two classes whereas multi-class classification involves predicting whether the output belongs in one of N-classes. We build a model using binary classification when our predicted result/output belongs only to 2 classes. In this case, our binary classification will be the column "US_vehicle_type" because the output has only two classes which are either passenger or car. The multi-class classification includes columns such as vehicle class where we have an output with more than two options.

Explanation of supervised machine learning, the notion of labelled data and the training and test datasets.

Supervised machine learning involves supervision during the process of evaluating by the model we constructed, we will have a set of inputs and output data (labeled data) from there, the model will learn the mapping function from the input and the corresponding output variables, this is to aid the model to predict the next output correctly when given another set of inputs variable. The model is trained so that the accuracy rate will increase, the larger the size of training data, the more accurate the prediction of the model will be, be aware that for supervised machine learning we already have the output we want in mind, that means that it is like a function where we make sure it produces what we want. In other words, the model has been "told" what to do rather than learning blindly (unsupervised machine learning). Using the same new born-baby example, the baby learned that the animal is a dog instead of observing over a long time. The notion of labeled data is for the model to know which output does it corresponds to, the training and test data sets are used to train and test the model, to form a good model, the usual ratio of train test is 70/30. We can say in a way that the more you train your model, the better the test results would be, but without testing the model, we wouldn't know how accurate the result of the model is. Training data sets can be said as a set of examples used to fit the parameter while the test data set will evaluate the model in an unbiased way.

Conduct a binary classification using the decision tree algorithm, where the labelled data is the “US_vehicle_type”.

Converting the "US_vehicle_type" into the number 0 and 1 for visualization purposes.

In [36]:
```
vehicle_new=vehicle.replace(['Passenger', 'Car'], [0, 1])
```

In [37]:
```
vehicle_new.head()
```

Out[37]:

	Manufacturer	Model	Vehicle_class	Vehicle_alt_class	US_vehicle_type	Engine_size (litres)	Horsepowe
0	Acura	Integra	Coupe	Sports	0	1.8	14
1	Acura	TL	Sedan	Ordinary	0	3.2	22
2	Acura	CL	Coupe	Ordinary	0	3.2	22
3	Acura	RL	Sedan	Ordinary	0	3.5	21
4	Audi	A4	Sedan	Ordinary	0	1.8	15

The sets of input chosen are Engine size and Horsepower. The target variable here is US vehicle type.

In [38]:
```
X = vehicle_new.iloc[:,[5,6]].values
y = vehicle_new.iloc[:, 4].values
```

In [39]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
X, y, test_size = 0.25, random_state = 0)
```

In [40]:
```
# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

In [41]:
```
# Fitting Decision Tree Classification to the Training set
from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier(
criterion = 'entropy', random_state = 0
)
classifier.fit(X_train, y_train)
```

Out[41]:
```
DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=None,
                       max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=0, splitter='best')
```

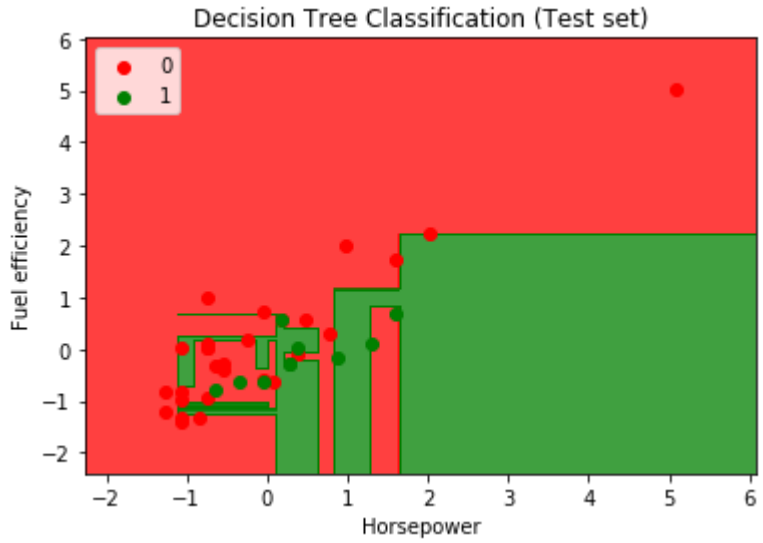
In [42]:
```
testXdata=test.iloc[:,[5,6]].values
testXdata=sc.fit_transform(testXdata)
y_pred=classifier.predict(testXdata)
y_pred
```

Out[42]:
```
array([0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0], dtype=int64)
```

Visualisation for the binary classification

In [43]:
```
# Visualising the Test set results
from matplotlib.colors import ListedColormap
X_set, y_set = X_test, y_test
X1, X2 = np.meshgrid(
    np.arange(
        start = X_set[:, 0].min() - 1,
        stop = X_set[:, 0].max() + 1,
        step = 0.01
    ),
    np.arange(
        start = X_set[:, 1].min() - 1,
        stop = X_set[:, 1].max() + 1,
        step = 0.01
    )
)
plt.contourf(
    X1,
    X2,
    classifier.predict(
        np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),
    alpha = 0.75,
    cmap = ListedColormap(('red', 'green'))
)
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(
        X_set[y_set == j, 0],
        X_set[y_set == j, 1],
        c = ListedColormap(('red', 'green'))(i),
        label = j
    )
plt.title('Decision Tree Classification (Test set)')
plt.xlabel('Horsepower')
plt.ylabel('Fuel efficiency')
plt.legend()
plt.show()
```

```
'c' argument looks like a single numeric RGB or RGBA sequence, which should be
avoided as value-mapping will have precedence in case its length matches with
'x' & 'y'.  Please use a 2-D array with a single row if you really want to spec
ify the same RGB or RGBA value for all points.
'c' argument looks like a single numeric RGB or RGBA sequence, which should be
avoided as value-mapping will have precedence in case its length matches with
'x' & 'y'.  Please use a 2-D array with a single row if you really want to spec
ify the same RGB or RGBA value for all points.
```



Making a dataframe to display the results, the ID starts from 1 (by default it will be 0) so i decided to type it out.

In [44]:
```
result_0 = {'Id': [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17],
        'Predicted': y_pred}

predicted_res= pd.DataFrame(result_0, columns = ['Id', 'Predicted'])
predicted_res
```

Out[44]:

	Id	Predicted
0	1	0
1	2	1
2	3	0
3	4	0
4	5	1
5	6	1
6	7	1
7	8	0
8	9	0
9	10	0
10	11	0
11	12	0
12	13	0
13	14	0
14	15	1
15	16	0
16	17	0

In [45]:
```
predicted_res.to_csv(r"C:\Users\USER\Desktop\FIT 1043 INTRO DATA SCI\30719704-ZOI
```

In [46]:
```
y_pred = classifier.predict(X_test)

from sklearn.metrics import confusion_matrix
matrix = confusion_matrix(y_test, y_pred)
matrix
```

Out[46]:
```
array([[25,  4],
       [ 4,  6]], dtype=int64)
```

The classifier predicted a total number of 39 predictions. From the confusion matrix, we have 6 True negative results, 25 true positive results, 4 false-positive, and 4 false negatives. The accuracy rate can be calculated by 36/39=0.79. This means that the model is not very accurate, but considering if it is a rather good model or not, we will consider other properties such as sensitivity, specificity, and precision. The error rate is also known as the "Misclassification rate" is 1-0.79=0.21. In this case, our sensitivity which is also known as "True positive rate" or "recall" is TP / (TP + FN) = 0.86. A model can have low precision and accuracy, so if the model has quite a number of false-positive, the precision should be low. The specificity is TN / (TN + FP)= 0.6 means that when the value is negative the prediction is often 60% correct. The precision of this model is TP / (TP + FP) = 0.86, this means that when a positive value is predicted, the prediction is often 86% correct. In short, if the model has high accuracy, sensitivity, precision, and specificity then the model is considered a good model.

In [47]:
```
from sklearn.metrics import accuracy_score
accuracy_score(y_test, y_pred)
```

Out[47]:
```
0.7948717948717948
```

Conduct a multi-class classification using the decision tree algorithm, where the labelled data is the “Vehicle_class”.

In [48]:
```
X = vehicle.iloc[:,5:12].values
y = vehicle.iloc[:, 2].values
```

In [49]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
X, y, test_size = 0.25, random_state = 0)
```

In [50]:
```
# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

In [51]:
```
# Fitting Decision Tree Classification to the Training set
from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier(
criterion = 'entropy', random_state = 0
)
classifier.fit(X_train, y_train)
```

Out[51]:
```
DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=None,
                       max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=0, splitter='best')
```

In [52]:
```
testXdata=test.iloc[:,1:8].values
```

In [53]:
```
testXdata=sc.transform(testXdata)
y_pred=classifier.predict(testXdata)
y_pred
```

Out[53]:
```
array(['Sedan', 'Coupe', 'SUV', 'Coupe', 'MPV', 'Coupe', 'Sedan', 'Sedan',
       'Coupe', 'Truck', 'SUV', 'SUV', 'Sedan', 'Sedan', 'Sedan', 'Sedan',
       'SUV'], dtype=object)
```

Making a dataframe to display the output

In [54]:
```
pd.DataFrame(y_pred).rename(columns={0:'Predicted'}).to_csv('30719704-ZOLYNLANG-
```

In [55]:
```
y_pred = classifier.predict(X_test)

from sklearn.metrics import confusion_matrix
matrix = confusion_matrix(y_test, y_pred)
matrix
```

Out[55]:
```
array([[ 8,  0,  0,  4,  0],
       [ 0,  2,  2,  0,  0],
       [ 0,  0,  3,  0,  0],
       [ 0,  0,  0, 17,  0],
       [ 0,  1,  1,  0,  1]], dtype=int64)
```

In [56]:
```
from sklearn.metrics import accuracy_score
accuracy_score(y_test, y_pred)
```

Out[56]:
```
0.7948717948717948
```

Visualisation for Vehicle class

In [57]:
```
vehicle2=vehicle
```

In [58]:
```
def convert_string(value):
    if 'SUV' in value:
        return 1
    elif "MPV" in value:
        return 2
    elif "Truck"in value:
        return 3
    elif "Sedan" in value:
        return 4
    elif "Coupe" in value:
        return 5
```

In [59]:
```
def plot_decision_tree(clf,feature_name,target_name):
    dot_data = StringIO()
    tree.export_graphviz(clf, out_file=dot_data,
                         feature_names=feature_name,
                         class_names=target_name,
                         filled=True, rounded=True,
                         special_characters=True)
    graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
    return Image(graph.create_png())
```

In [60]:
```
vehicle2.Vehicle_class = vehicle2.Vehicle_class.apply(convert_string)
```

In [61]:
```
vehicle2= pd.get_dummies(vehicle2)
```

In [62]:
```
X_train =vehicle2.loc[:,vehicle2.columns!='Vehicle_class']
```

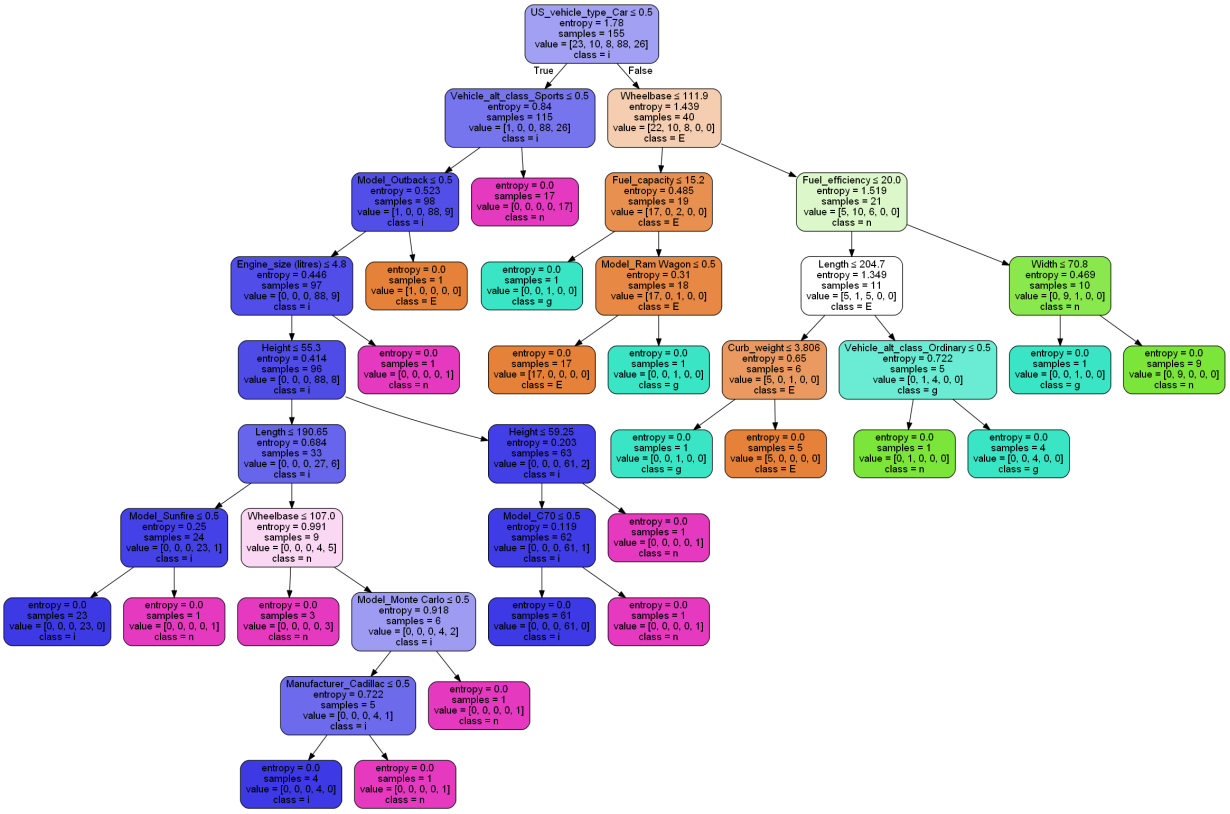
In [63]:
```
Y_train =vehicle2.Vehicle_class
```

In [64]:
```
clf = tree.DecisionTreeClassifier(criterion='entropy')
```

In [65]:
```
clf = clf.fit(X_train,Y_train)
```

In [66]:
```
plot_decision_tree(clf, X_train.columns,vehicle2.columns[1])
```

Out[66]:



Conduct another multi-class classification with the “Vehicle_alt_class” as the label.

In [67]:
```
X = vehicle.iloc[:,5:14].values
y = vehicle.iloc[:,3].values
```

In [68]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
X, y, test_size = 0.19, random_state = 0)
```

In [69]:
```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.fit_transform(X_test)
```

In [70]:
```
from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier(
criterion = 'entropy', random_state = 0
)
classifier.fit(X_train, y_train)
```

Out[70]:
```
DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=None,
                       max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=0, splitter='best')
```

In [71]:
```
# test=test.drop(columns=['Curb_weight','Fuel_efficiency'])
testXdata=test.iloc[:,1:10].values
```

In [72]:
```
testXdata=sc.fit_transform(testXdata)
y_pred=classifier.predict(testXdata)
y_pred
```

Out[72]:
```
array(['Ordinary', 'Sports', 'Ordinary', 'Sports', 'Ordinary', 'Ordinary',
       'Ordinary', 'Ordinary', 'Sports', 'Ordinary', 'Ordinary',
       'Ordinary', 'Ordinary', 'Ordinary', 'Ordinary', 'Ordinary',
       'Ordinary'], dtype=object)
```

In [73]:
```
result = {'Id': [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17],
        'Predicted':y_pred}

predicted_result= pd.DataFrame(result, columns = ['Id', 'Predicted'])
predicted_result
```

Out[73]:

	Id	Predicted
0	1	Ordinary
1	2	Sports
2	3	Ordinary
3	4	Sports
4	5	Ordinary
5	6	Ordinary
6	7	Ordinary
7	8	Ordinary
8	9	Sports
9	10	Ordinary
10	11	Ordinary
11	12	Ordinary
12	13	Ordinary
13	14	Ordinary
14	15	Ordinary
15	16	Ordinary
16	17	Ordinary

In [74]:
```
y_pred = classifier.predict(X_test)

from sklearn.metrics import confusion_matrix
matrix = confusion_matrix(y_test, y_pred)
matrix
```

Out[74]:
```
array([[ 0,  1,  0],
       [ 1, 23,  0],
       [ 0,  2,  3]], dtype=int64)
```

The accuracy score can be calculated by 26/30=0.86

In [75]:
```
from sklearn.metrics import accuracy_score
accuracy_score(y_test, y_pred)
```

Out[75]:
```
0.8666666666666667
```

In [76]:
```
ed).rename(columns={0:'Predicted'}).to_csv('30719704-ZOLYNLANG-vehicle_alt_class.
```

Visualisation for Vehicle_alt_class (multi-class visualisation)

In [77]:
```
def vehicle_alt_class(value):
    if 'Ordinary' in value:
        return 1
    elif "Sports" in value:
        return 2
    elif "Hatch"in value:
        return 3
    elif "SUV" in value:
        return 4
```

In [78]:
```
def plot_decision_tree(clf,feature_name,target_name):
    dot_data = StringIO()
    tree.export_graphviz(clf, out_file=dot_data,
                         feature_names=feature_name,
                         class_names=target_name,
                         filled=True, rounded=True,
                         special_characters=True)
    graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
    return Image(graph.create_png())
```

In [79]:
```
vehicle.Vehicle_alt_class = vehicle.Vehicle_alt_class.apply(vehicle_alt_class)
```

In [80]:
```
vehicle = pd.get_dummies(vehicle)
```

In [81]:
```
X_train = vehicle.loc[:,vehicle.columns!='Vehicle_alt_class']
```

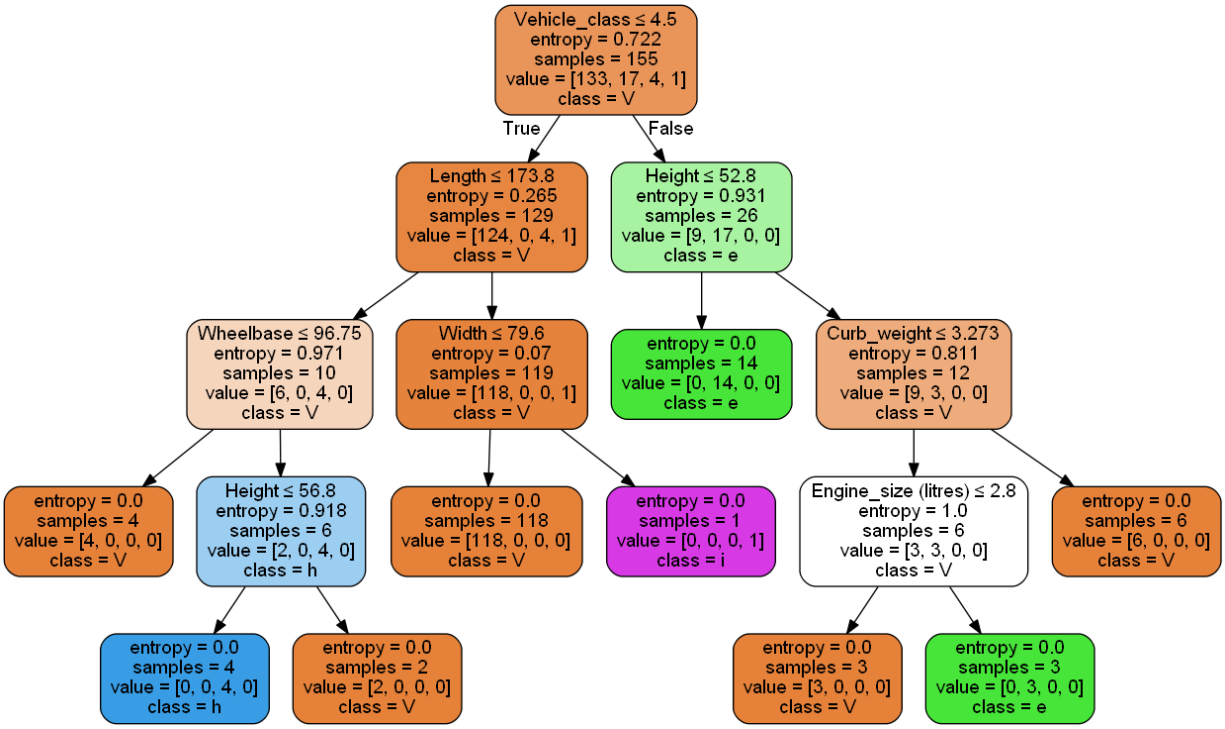
In [82]:
```
Y_train = vehicle.Vehicle_alt_class
```

In [83]:
```
clf = tree.DecisionTreeClassifier(criterion='entropy')
```

In [84]:
```
clf = clf.fit(X_train,Y_train)
```

In [85]:
```
plot_decision_tree(clf, X_train.columns,vehicle.columns[1])
```

Out[85]:



Decide on the evaluation metrics and explain how you evaluated your output.

In order to visualize my output, I used the listed-color map, confusion matrix, and tree diagram. For multi-class visualization, a tree diagram is the most appropriate way to output my results, this is because to classify which class it belongs to, we need to break it down using different columns, its build by a recursive partition (divide up) the feature space into regions while grouping similar instances, the prediction is the most common values in each region. Based on the vehicle_alt_class visualization, we can see it will be grouped by different attributes such as engine_size and wheelbase, from there it will be further partitioned into different classes, the same principle can be applied to the vehicle_class visualization.

Re-do your Part 3 (d) using the Random Forest algorithm

In [86]:
```
new_ver_vehicle=pd.read_csv("FIT1043-vehicle-classifier.csv")
```

In [87]:
```
new_ver_vehicle['Vehicle_alt_class'] = new_ver_vehicle['Vehicle_alt_class'].fill
new_ver_vehicle['Curb_weight'] = new_ver_vehicle['Curb_weight'].fillna(3.178)
new_ver_vehicle['Fuel_efficiency'] = new_ver_vehicle['Fuel_efficiency'].fillna(2
```

In [88]:
```
new_ver_vehicle =new_ver_vehicle[new_ver_vehicle['Height']<100]
```

In [89]:
```
testXdata=test.iloc[:,3:8].values
```

In [90]:
```
X = new_ver_vehicle.iloc[:,7:12].values
y = new_ver_vehicle.iloc[:, 2].values
```

In [91]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
X, y, test_size = 0.20, random_state = 0)
```

In [92]:
```
# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

In [93]:
```
# Fitting Random Forest Classification to the Training set
from sklearn.ensemble import RandomForestClassifier
classifier = RandomForestClassifier(
    n_estimators = 20,
    criterion = 'entropy',
    random_state = 0
)
classifier.fit(X_train, y_train)
# Predicting the Test set results

testXdata=sc.transform(testXdata)
y_pred=classifier.predict(testXdata)
y_pred
```

Out[93]:
```
array(['Sedan', 'Coupe', 'SUV', 'Coupe', 'MPV', 'Sedan', 'Sedan', 'Sedan',
       'Coupe', 'SUV', 'SUV', 'SUV', 'Sedan', 'Sedan', 'Sedan', 'Sedan',
       'SUV'], dtype=object)
```

In [94]:
```
pd.DataFrame(y_pred).rename(columns={0:'Predicted'}).to_csv('30719704-ZOLYNLANG-
```

Visualisation for Random Forest algorithm

In [95]:
```
y_pred = classifier.predict(X_test)

from sklearn.metrics import confusion_matrix
matrix = confusion_matrix(y_test, y_pred)
matrix
```

Out[95]:
```
array([[ 4,  0,  0,  3,  0],
       [ 0,  1,  2,  0,  0],
       [ 0,  0,  3,  0,  0],
       [ 0,  0,  0, 15,  0],
       [ 0,  0,  2,  0,  1]], dtype=int64)
```

In [96]:
```
from sklearn.metrics import accuracy_score
accuracy_score(y_test, y_pred)
```

Out[96]:
```
0.7741935483870968
```

The results are different. I decided to use this output as my final submission in Kaggle because the rate of correct prediction is higher. As mentioned earlier, when deciding on which is a better model, we also need to consider other factors such as the sensitivity, precision and specificity. The accuracy here can be calculated by (TP + TN) / (TP + TN + FP + FN)=24/31= 0.77.

In [97]:
```
new_ver_vehicle.Vehicle_alt_class =new_ver_vehicle.Vehicle_alt_class.apply(vehic
```

In [98]:
```
new_ver_vehicle = pd.get_dummies(new_ver_vehicle)
```

In [99]:
```
X_train = new_ver_vehicle.loc[:,new_ver_vehicle.columns!='Vehicle_alt_class']
```

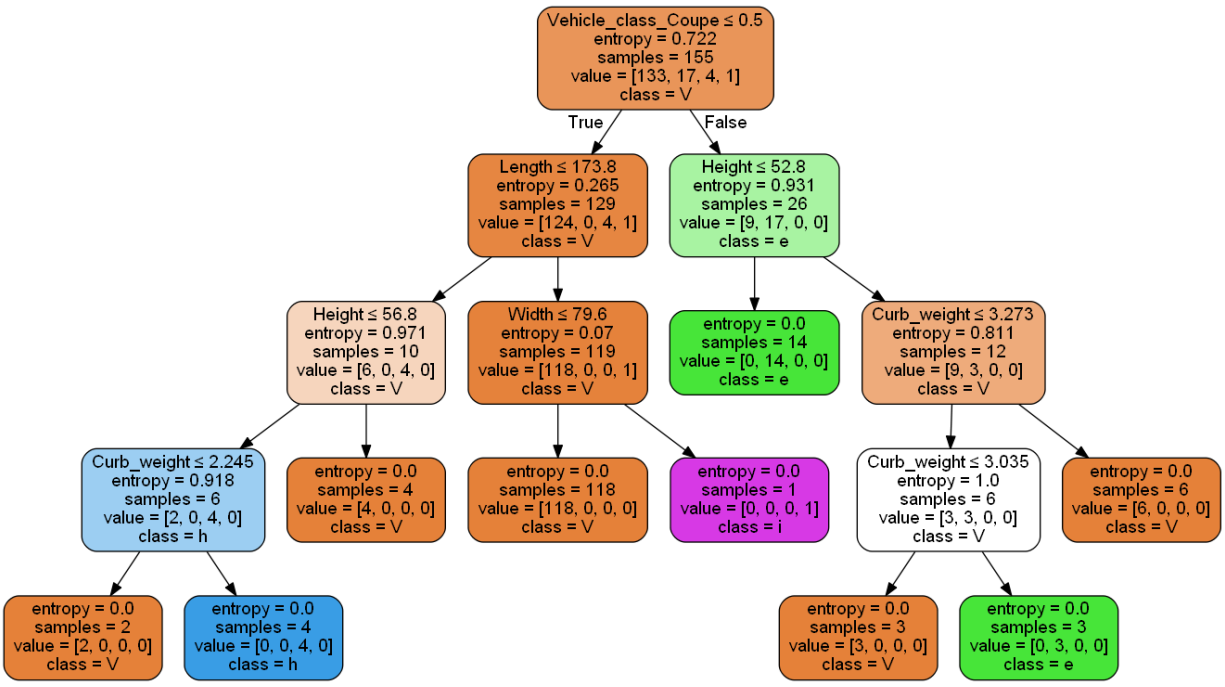
In [100]:
```
Y_train = new_ver_vehicle.Vehicle_alt_class
```

In [101]:
```
clf = tree.DecisionTreeClassifier(criterion='entropy')
```

In [102]:
```
clf = clf.fit(X_train,Y_train)
```

In [103]:
```
plot_decision_tree(clf, X_train.columns,vehicle.columns[1])
```

Out[103]:



Conclusion

In conclusion, supervised and unsupervised machine learning are very useful when it comes to extracting data as the model form can be used to predict results in various fields. Machine learning is mandatory as we are living in a world with data surrounding us. When models are exposed to new data, they are able to adapt independently. Besides, classification with confusion matrix is extremely helpful when it comes to medical, business, and research fields as it is able to predict the outcome with the given input data and it provides information for us to calculate the accuracy, sensitivity, precision, and specificity rate. Moreover, learning, reasoning, and decision making can be achieved through interpreting, analyzing patterns and structure in data without the need of humans. This algorithm can incorporate information to improve its decision making in the future.

References

Pandas (n.d.). Retrieved from https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.get_dummies.html (https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.get_dummies.html)

Bhatt, B (n.d.). Retrieved from https://bhattbhavesh91.github.io/ (https://bhattbhavesh91.github.io/)

Guru99: Supervised vs Unsupervised Learning: Key Differences. Retrieved from https://www.guru99.com/supervised-vs-unsupervised-learning.html (https://www.guru99.com/supervised-vs-unsupervised-learning.html)

Cluster Validation, (n.d.). Retrieved from http://www.cs.kent.edu/~jin/DM08/ClusterValidation.pdf (http://www.cs.kent.edu/~jin/DM08/ClusterValidation.pdf)

Haffner.P (2016) What is Machine Learning – and Why is it Important?. Retrieved from https://www.interactions.com/blog/technology/machine-learning-important/ (https://www.interactions.com/blog/technology/machine-learning-important/)