```
# model 1

# To do list
#Classification exercise: MpG consumption:
#Create a new jupyter file to analyse the training data
mpgTrainingset.txt (published by
#Garnegie Mellon).
#This file will constitue your report, it should then includes your
code, illustrations and analyses of the
#obtained results.
#The data represent the characteristics of cars: number of cylinders,
cubic inch displacement, horse
#power, weight, acceleration.
#The category is discrete (values of 10, 15, 20, 25, 30, 35, 40, and
45) . It represents the consumption
#in miles per gallon.
#Your goal is to predict with the highest reliability the category of
the cars belonging to the file
#mpgTestset.txt.
#You will describe the different steps used to obtain this results.
You will use a least 2 different
#methods. Their performance should be evaluated (quantitaively).

# data liberies used in this analysis

# Import of the needed libraires
#graphical librairies
import matplotlib as mpl
from matplotlib import pyplot
import matplotlib.pyplot as plt
import seaborn as sns
from pylab import figure, subplot, hist, xlim, show, plot
%matplotlib inline

#data librairies

import pandas as pd
import pylab as pl
import numpy as np

from pandas.plotting import scatter_matrix
from pandas.plotting import boxplot
from pandas.plotting import parallel_coordinates
from matplotlib.colors import ListedColormap
```

The code snippet imports the data from a CSV file named "mpgTrainingSet-headings.csv" into a Pandas DataFrame object named "data_panda".

```python
# to import the data from csv file
# data is imported and panda object is created
data_panda = pd.read_csv('mpgTrainingSet-headings.csv')

# to see the total no of data and its key columns
print(data_panda.keys())

# len
nb_specimen=len(data_panda)
print('There are '+ str(nb_specimen)+' cars in the set')
```

```
Index(['Consumption', 'Cylinders', 'Cubic_inch', 'Horsepower',
'Weight',
       'Acceleration', 'Brand', 'Car_name'],
      dtype='object')
There are 342 cars in the set
```

```python
# for more cleare futur use we can create a set with the input col
Input_cols = [ 'Cylinders', 'Cubic_inch', 'Horsepower', 'Weight',
       'Acceleration']

print(data_panda)
```

```
     Consumption  Cylinders  Cubic_inch  Horsepower  Weight
Acceleration  \
0             35          4          79          58    1825
18.6
1             25          4          96          69    2189
18.0
2             25          4          98          90    2265
15.5
3             25          4         116          75    2246
14.0
4             30          4          68          49    1867
19.5
..           ...        ...         ...         ...     ...
...
337           30          4          79          67    2000
16.0
338           20          6         200          85    3070
16.7
339           20          6         200          85    2990
18.2
340           25          4         108          93    2391
15.5
341           30          4          97          67    2065
17.8

       Brand  Car_name
0    renault         5
```

```
1     renault         12
2        fiat  124_sport
3        fiat        124
4        fiat  128_sport
..        ...        ...
337      fiat       x1.9
338  mercury      zephyr
339  mercury      zephyr
340   subaru         NaN
341   subaru         NaN

[342 rows x 8 columns]
```

The code snippet prints the list of column names (headers) of the Pandas DataFrame object "data_panda".

```
# value count and brand count
data_panda['Consumption'].value_counts()

data_panda['Brand'].value_counts()
```

```
Brand
ford             42
chevrolet        39
plymouth         26
dodge            24
toyota           22
amc              21
datsun           20
volkswagen       19
buick            16
pontiac          13
honda            13
mazda            12
mercury          10
oldsmobile       10
fiat              7
volvo             6
audi              6
peugeot           6
chrysler          6
subaru            4
renault           3
saab              3
mercedes-benz     3
opel              3
chevy             2
cadillac          2
bmw               1
```

```
capri              1
nissan             1
triumph            1
Name: count, dtype: int64
```

```python
#definition of the colors used for visualization
colors = np.where(data_panda['Consumption']==10,'r','-')
colors[data_panda['Consumption']==15] = 'g'
colors[data_panda['Consumption']==20]= 'b'
colors[data_panda['Consumption']==25]= 'y'
colors[data_panda['Consumption']==30]= 'c'
colors[data_panda['Consumption']==35]= 'm'
colors[data_panda['Consumption']==40]= 'k'
colors[data_panda['Consumption']==45]= '0.5'

#print(colors)
color_dict={10:'r',15:'g' ,20:'b',25:'y',30:'c' ,35:'m',40:'k' ,45:'0.
5'}

data_panda.groupby('Consumption').describe()
```

```
            Cylinders
Cubic_inch  \
                count      mean       std  min  25%  50%  75%  max
count
Consumption

10                8.0  8.000000  0.000000  8.0  8.0  8.0  8.0  8.0
8.0
15               73.0  7.671233  0.746376  6.0  8.0  8.0  8.0  8.0
73.0
20               85.0  5.788235  1.380943  3.0  4.0  6.0  6.0  8.0
85.0
25               64.0  4.437500  1.052209  3.0  4.0  4.0  4.0  8.0
64.0
30               58.0  4.068966  0.368118  4.0  4.0  4.0  4.0  6.0
58.0
35               37.0  4.081081  0.363500  4.0  4.0  4.0  4.0  6.0
37.0
40               11.0  4.181818  0.603023  4.0  4.0  4.0  4.0  6.0
11.0
45                6.0  4.000000  0.000000  4.0  4.0  4.0  4.0  4.0
6.0

                         ...   Weight        Acceleration
\
                   mean  ...      75%     max        count       mean

Consumption              ...
```

```
10             395.375000  ...  4951.25  4997.0            8.0  12.125000

15             325.287671  ...  4341.00  4735.0           73.0  13.742466

20             207.717647  ...  3459.00  4215.0           85.0  15.876471

25             140.828125  ...  2857.50  3900.0           64.0  16.365625

30             107.224138  ...  2555.50  3250.0           58.0  16.513793

35             103.351351  ...  2215.00  2950.0           37.0  15.872973

40             105.454545  ...  2117.50  3015.0           11.0  16.800000

45              90.666667  ...  2125.00  2335.0            6.0  20.533333


                    std   min     25%   50%     75%   max
Consumption
10             1.187735  11.0  11.000  12.0  12.750  14.0
15             2.650625   8.5  12.000  13.5  14.900  21.0
20             2.183523  11.0  14.500  15.9  17.200  21.9
25             2.384821  12.5  14.900  16.0  17.600  24.8
30             2.430901  11.3  14.825  16.4  18.175  22.2
35             1.935437  11.4  14.500  15.8  17.300  19.9
40             1.608726  14.7  15.600  16.9  17.950  19.2
45             4.028234  13.8  18.800  21.6  23.200  24.6

[8 rows x 40 columns]
```

The data shows The mean, standard deviation, minimum, 25th percentile, 50th percentile, 75th percentile, and maximum values for the number of cylinders, bore, displacement, compression ratio, horsepower, curb weight, and fuel economy are shown in the table above.

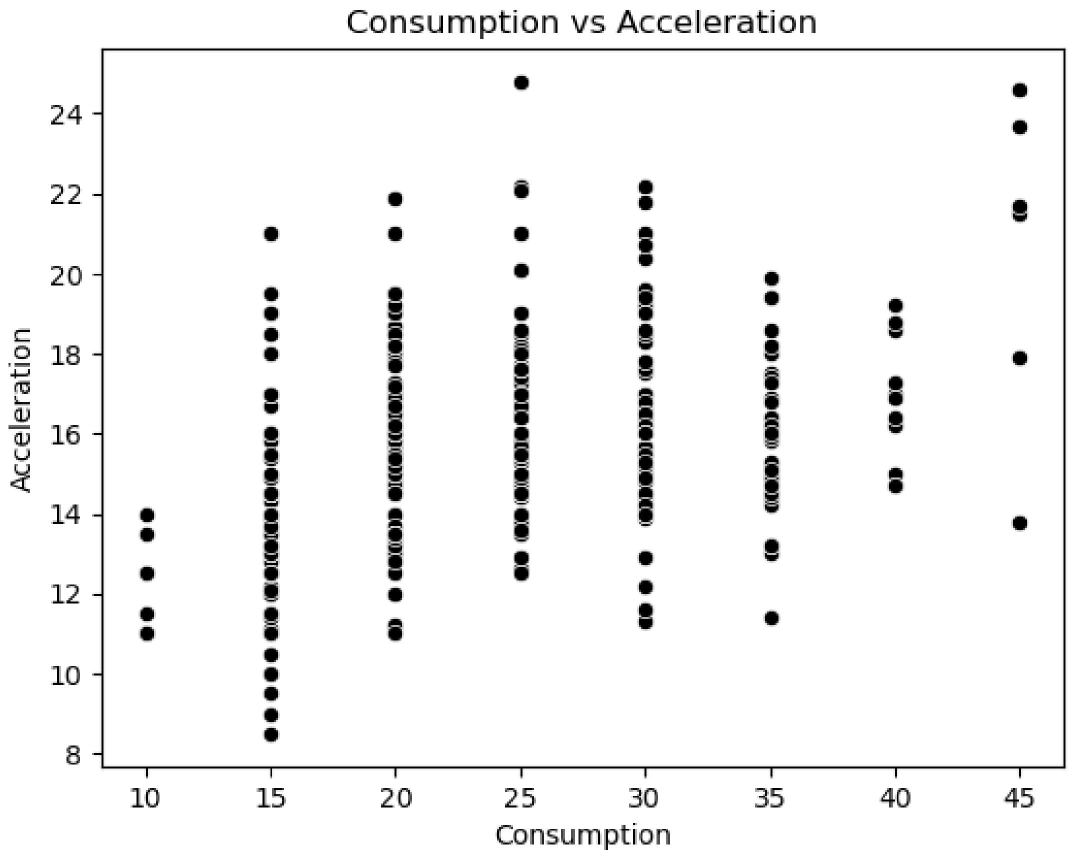
```
# Correlation Analysis
numeric_data =
data_panda.drop(columns=data_panda.select_dtypes(exclude=[np.number]).
columns)
correlation_matrix = numeric_data.corr()
print("Columns used for correlation:")
print(numeric_data.columns)
print("\nCorrelation Matrix:")
print(correlation_matrix)
```

```
Columns used for correlation:
Index(['Consumption', 'Cylinders', 'Cubic_inch', 'Horsepower',
'Weight',
       'Acceleration'],
      dtype='object')
```

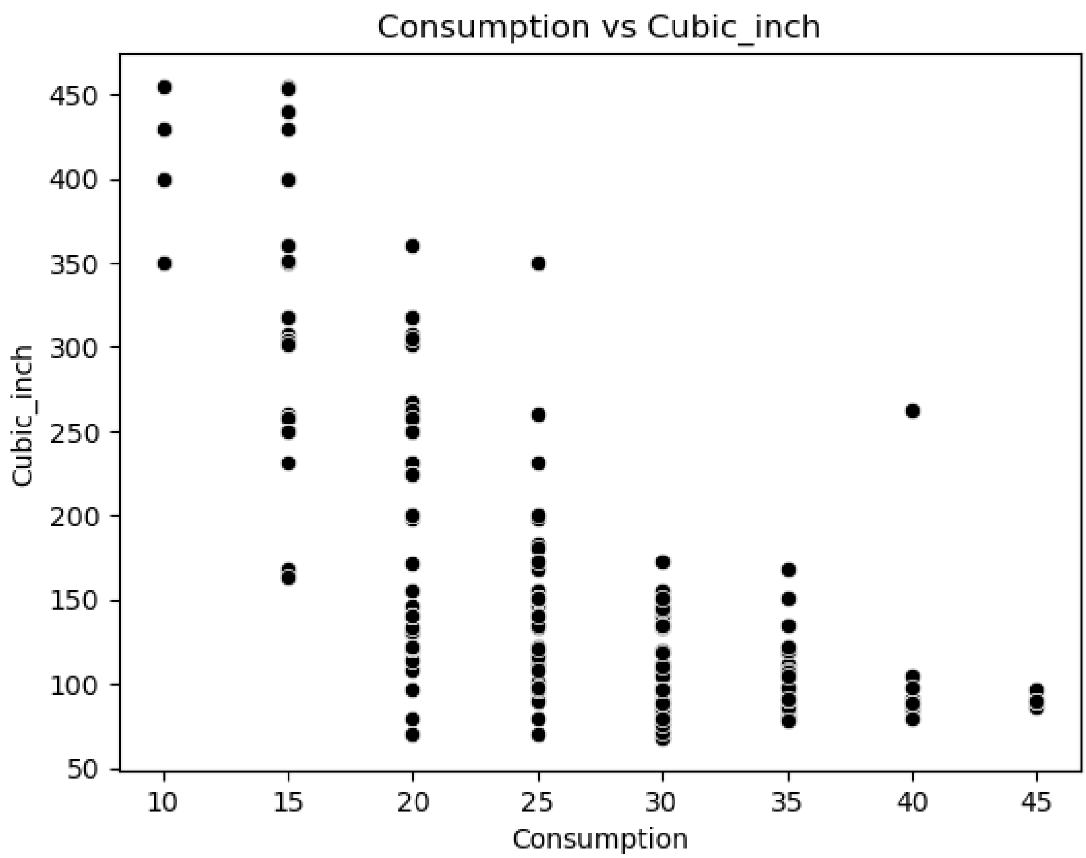
```
Correlation Matrix:
              Consumption  Cylinders  Cubic_inch  Horsepower    Weight  \
Consumption      1.000000  -0.742223   -0.777441   -0.764674 -0.820047   
Cylinders       -0.742223   1.000000    0.948426    0.839041  0.895761   
Cubic_inch      -0.777441   0.948426    1.000000    0.900970  0.942059   
Horsepower      -0.764674   0.839041    0.900970    1.000000  0.870616   
Weight          -0.820047   0.895761    0.942059    0.870616  1.000000   
Acceleration     0.393972  -0.476591   -0.505219   -0.687989 -0.391201   

              Acceleration  
Consumption       0.393972  
Cylinders        -0.476591  
Cubic_inch       -0.505219  
Horsepower       -0.687989  
Weight           -0.391201  
Acceleration      1.000000  
```

The correlation matrix shown above indicates that the features are moderately correlated with each other. For example, there is a negative correlation between consumption and horsepower (-0.764674), meaning that as horsepower increases, consumption tends to decrease. if a car manufacturer wants to improve the fuel economy of their cars, they should consider reducing the weight of the car and/or decreasing the horsepower of the engine.

```
# Visualization
sns.scatterplot(x='Consumption', y='Acceleration', color='k', 
data=data_panda)
plt.title('Consumption vs Acceleration')
plt.show()
```

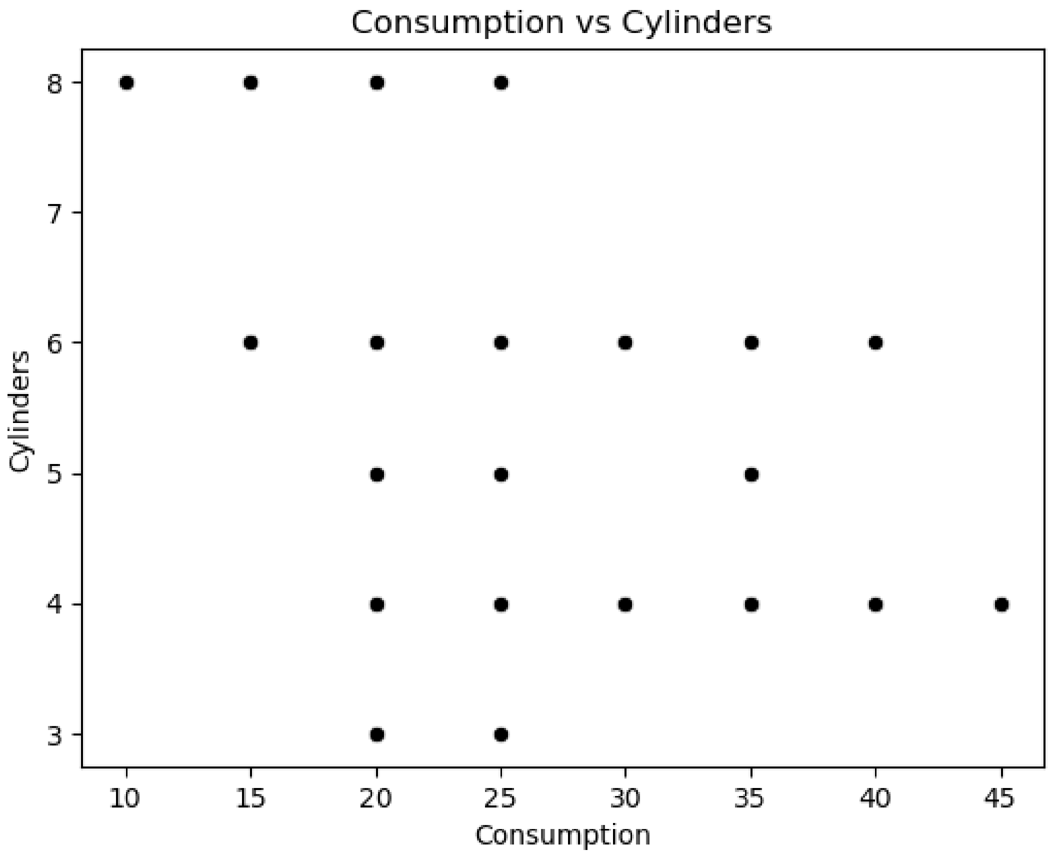



```
# Visualization
sns.scatterplot(x='Consumption', y='Cubic_inch', color='k',
data=data_panda)
plt.title('Consumption vs Cubic_inch')
plt.show()
```



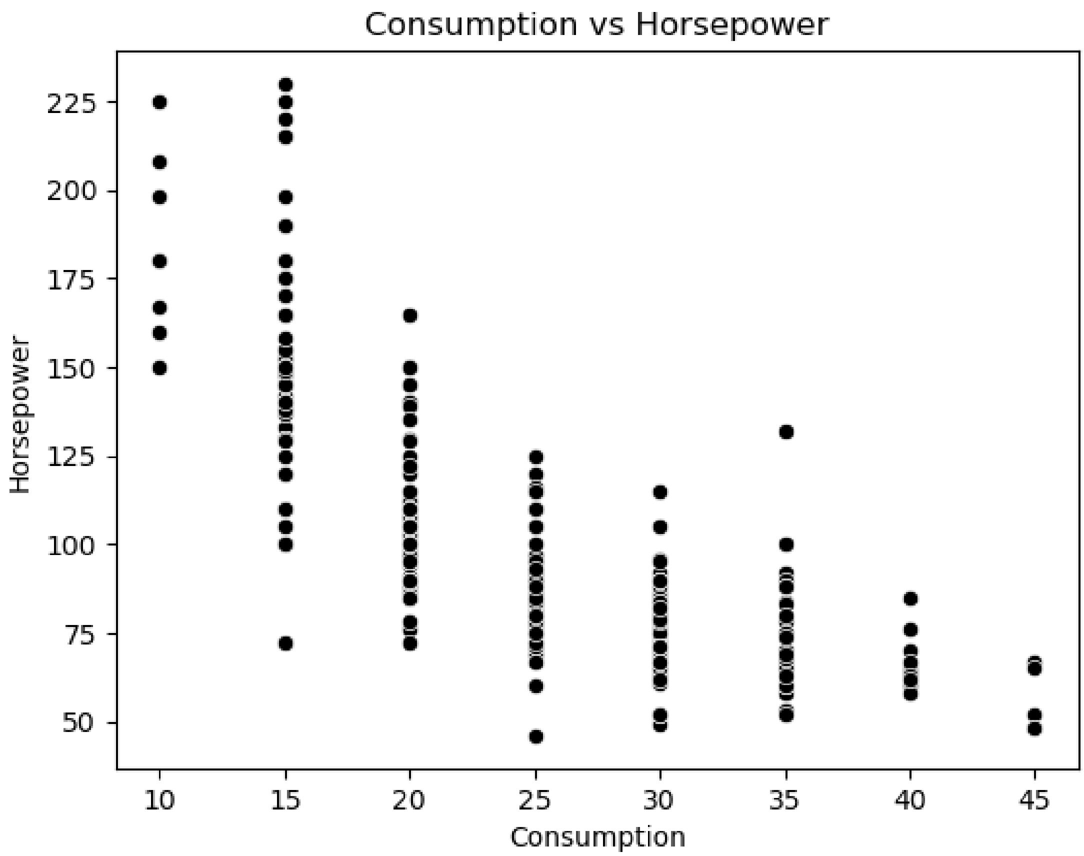
The scatter plot indicates that there is a weak positive correlation between consumption and cubic inch. This suggests that as cubic inch increases, consumption tends to increase as well. However, the correlation is not very strong, so there are other factors that are also important in determining fuel economy

```
# Visualization
sns.scatterplot(x='Consumption', y='Cylinders', color='k',
data=data_panda)
plt.title('Consumption vs Cylinders')
plt.show()
```



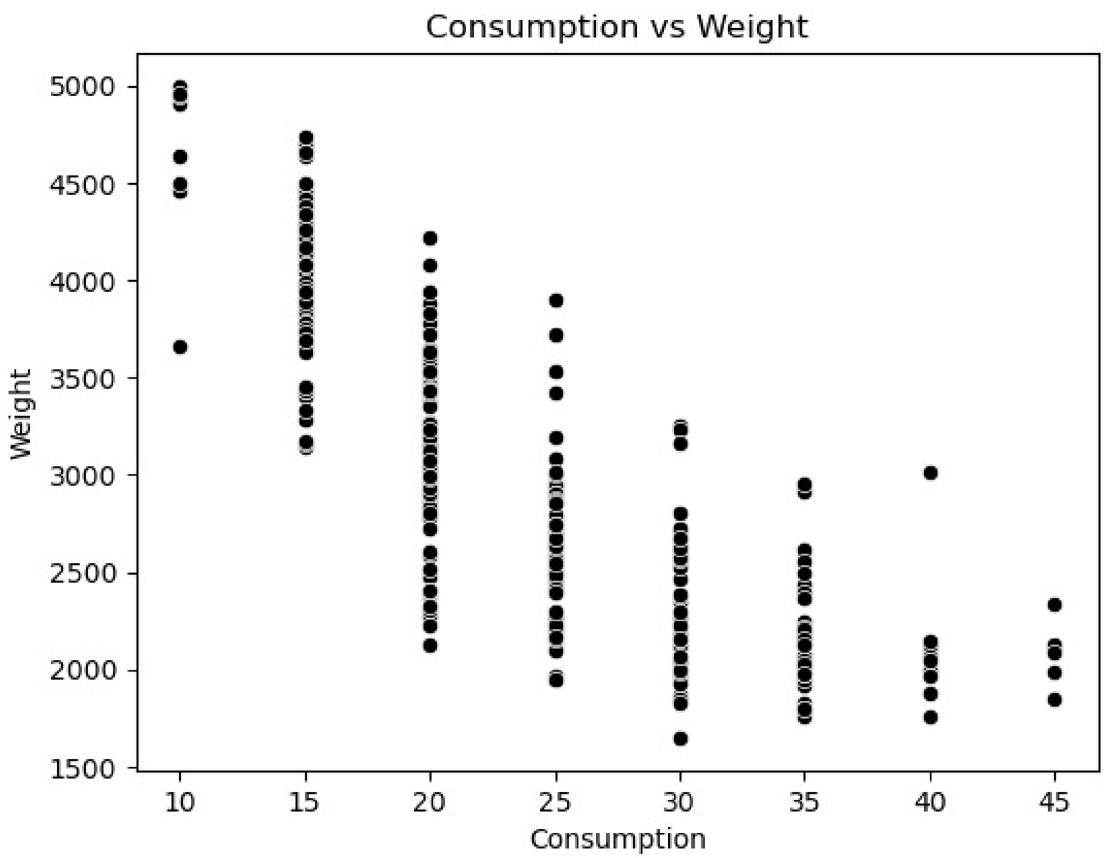
The scatter plot shows a negative correlation between consumption and cylinders. This means that as cylinders increase, consumption tends to decrease. This is because cars with more cylinders tend to be more efficient at converting fuel into power.

```
# Visualization
sns.scatterplot(x='Consumption', y='Horsepower', color='k',
data=data_panda)
plt.title('Consumption vs Horsepower')
plt.show()
```



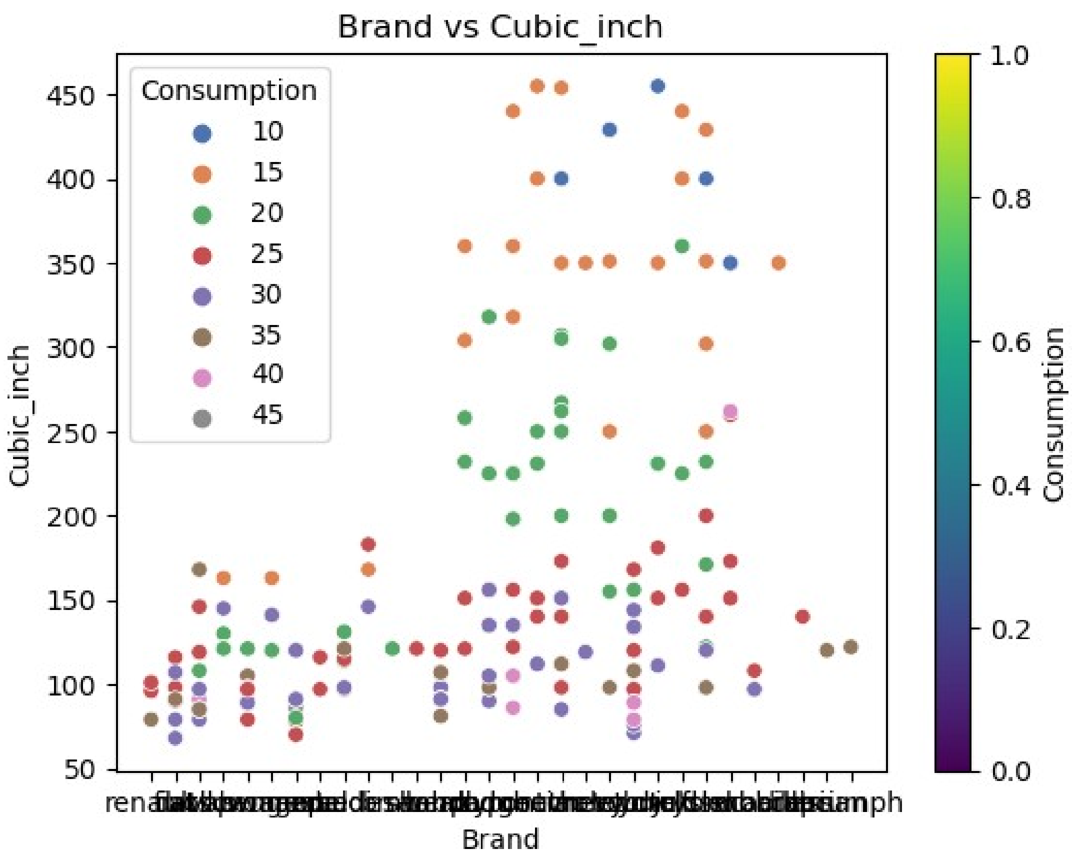
The scatter plot shows a strong negative correlation between consumption and horsepower. This means that as horsepower increases, consumption tends to decrease. This is because higher horsepower engines tend to be more efficient at converting fuel into power.

```
# Visualization
sns.scatterplot(x='Consumption', y='Weight', color='k',
data=data_panda)
plt.title('Consumption vs Weight')
plt.show()
```

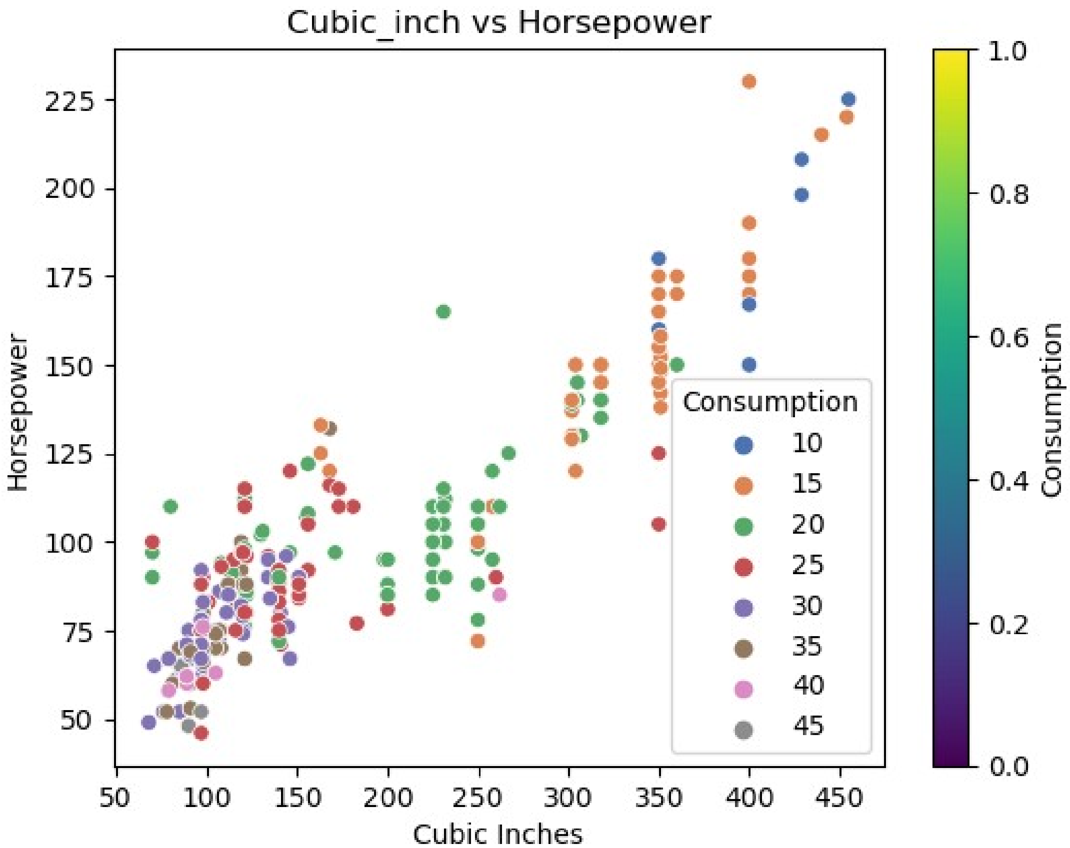


The scatter plot shows the relationship between fuel economy (mpg) and cubic inches (ci),Cylinders,Horsepower,weight for each car brand in the dataset. The colors of the points represent different levels of fuel economy, with blue representing the lowest fuel economy and red representing the highest fuel economy. The styles of the points represent different levels of cubic inches, with circles representing the smallest cubic inches and triangles representing the largest cubic inches. It is interesting to note that there are a few outliers in the data. For example, the Chevrolet Nova with 165 cubic inches has the lowest fuel economy of any car in the dataset. However, there are also a few cars with high cubic inches that have relatively good fuel economy, such as the Datsun with 165 cubic inches

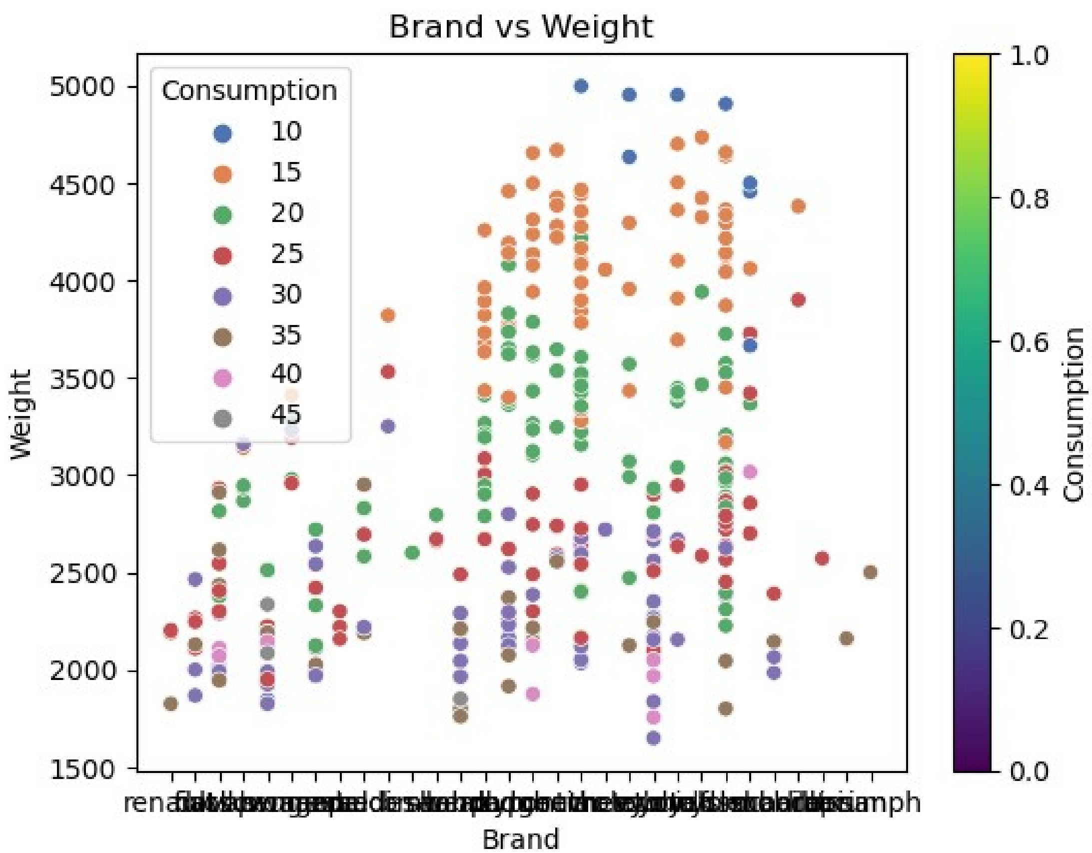
```
# Visualization with Seaborn scatter plot
scatter_plot = sns.scatterplot(x='Brand', y='Cubic_inch',
hue='Consumption', palette='deep', data=data_panda)
plt.title('Brand vs Cubic_inch')
plt.xlabel('Brand')
plt.ylabel('Cubic_inch')
plt.colorbar(scatter_plot.get_children()[0], label='Consumption')
plt.show()
```



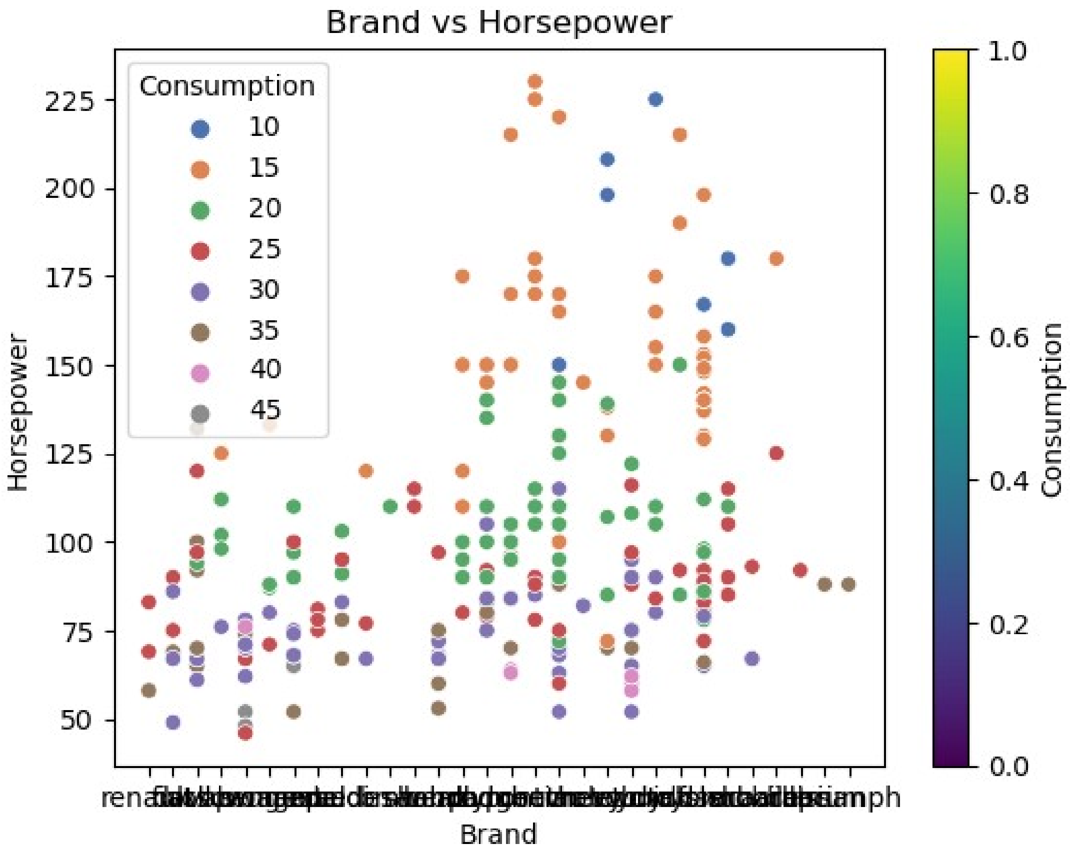

```python
# Visualization with Seaborn scatter plot
scatter_plot = sns.scatterplot(x='Cubic_inch', y='Horsepower',
hue='Consumption', palette='deep', data=data_panda)
plt.title('Cubic_inch vs Horsepower')
plt.xlabel('Cubic Inches')
plt.ylabel('Horsepower')
plt.colorbar(scatter_plot.get_children()[0], label='Consumption')
plt.show()
```



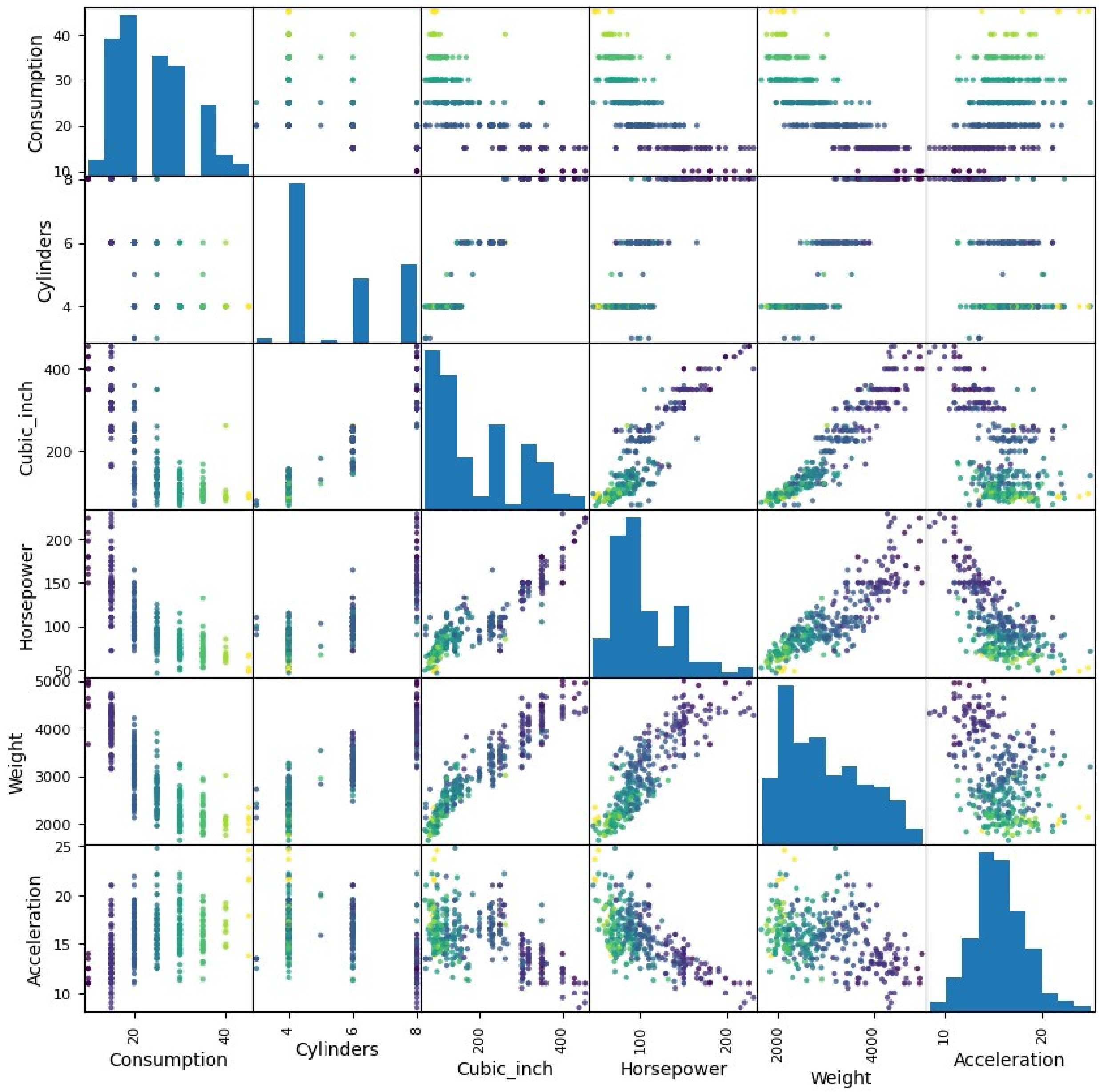

```
# Visualization with Seaborn scatter plot
scatter_plot = sns.scatterplot(x='Brand', y='Weight',
hue='Consumption', palette='deep', data=data_panda)
plt.title('Brand vs Weight')
plt.xlabel('Brand')
plt.ylabel('Weight')
plt.colorbar(scatter_plot.get_children()[0], label='Consumption')
plt.show()
```



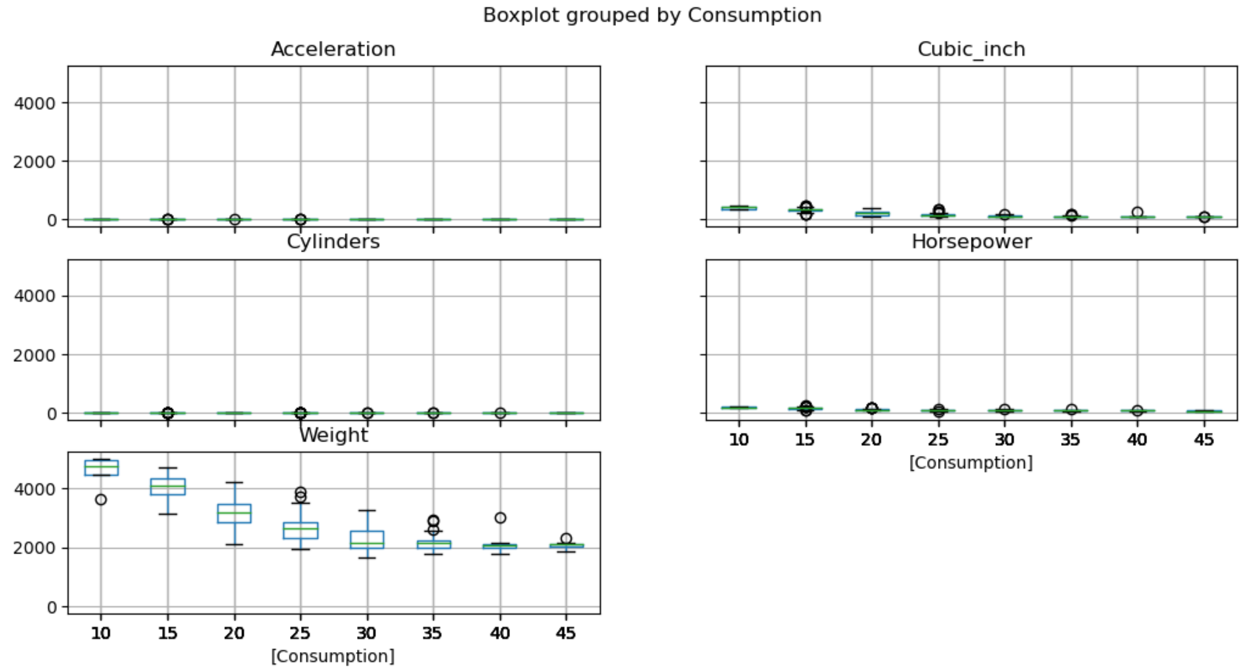

```
# Visualization with Seaborn scatter plot
scatter_plot = sns.scatterplot(x='Brand', y='Horsepower',
hue='Consumption', palette='deep', data=data_panda)
plt.title('Brand vs Horsepower')
plt.xlabel('Brand')
plt.ylabel('Horsepower')
plt.colorbar(scatter_plot.get_children()[0], label='Consumption')
plt.show()
```




```
# Specify the colors based on the 'Consumption' column
colors = data_panda['Consumption']

# Create scatter matrix
scatter_matrix(data_panda, figsize=(10, 10), diagonal='hist',
c=colors, alpha=0.8)

# Show the plot
plt.show()

data_panda.boxplot(by='Consumption', figsize=(12, 6));
```



Consumption
Cylinders
Cubic_inch
Horsepower
Weight
Acceleration



The purpose of this code is to create a scatter matrix, where each pair of variables in the data_panda DataFrame is plotted against each other. The diagonal subplots will be histograms of the corresponding variables, and the points in the scatter plots will be colored based on the values in the 'Consumption' column. The resulting visualization provides insights into the relationships and distributions of variables in the dataset.

The boxplot shows that there is a wide range of fuel economy values, from a low of 13 mpg to a high of 19 mpg. The median fuel economy is 15.5 mpg. The cars with the best fuel economy (the blue box) tend to have fewer cylinders and less horsepower. The cars with the worst fuel economy (the red box) tend to have more cylinders and more horsepower. But because we are working with an unnormalized data our diagrams are not that clear, what we will do is to normalize our data for a much clearer view of the diagram

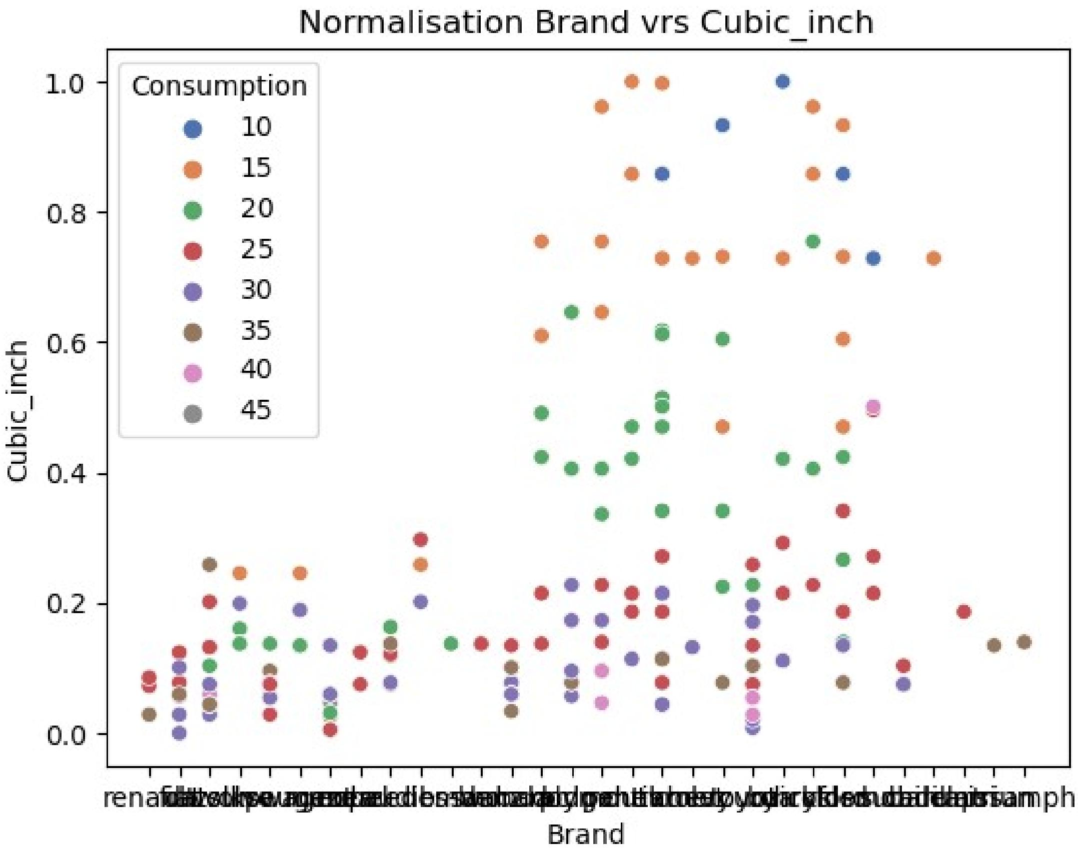
```
# Data Normalization
Norm = data_panda.copy()
Norm[Input_cols] = (data_panda[Input_cols] -
data_panda[Input_cols].min()) / (data_panda[Input_cols].max() -
data_panda[Input_cols].min())

print([Input_cols])

[['Cylinders', 'Cubic_inch', 'Horsepower', 'Weight', 'Acceleration']]
```

Normalization is a crucial step in data preparation for machine learning algorithms. It helps to address the issue of varying scales among features, ensuring that all features are treated equally and contribute proportionately to the learning process. By normalizing the data, the algorithm can focus on the underlying relationships between features without being influenced by their individual scales. This can lead to more accurate and robust models.

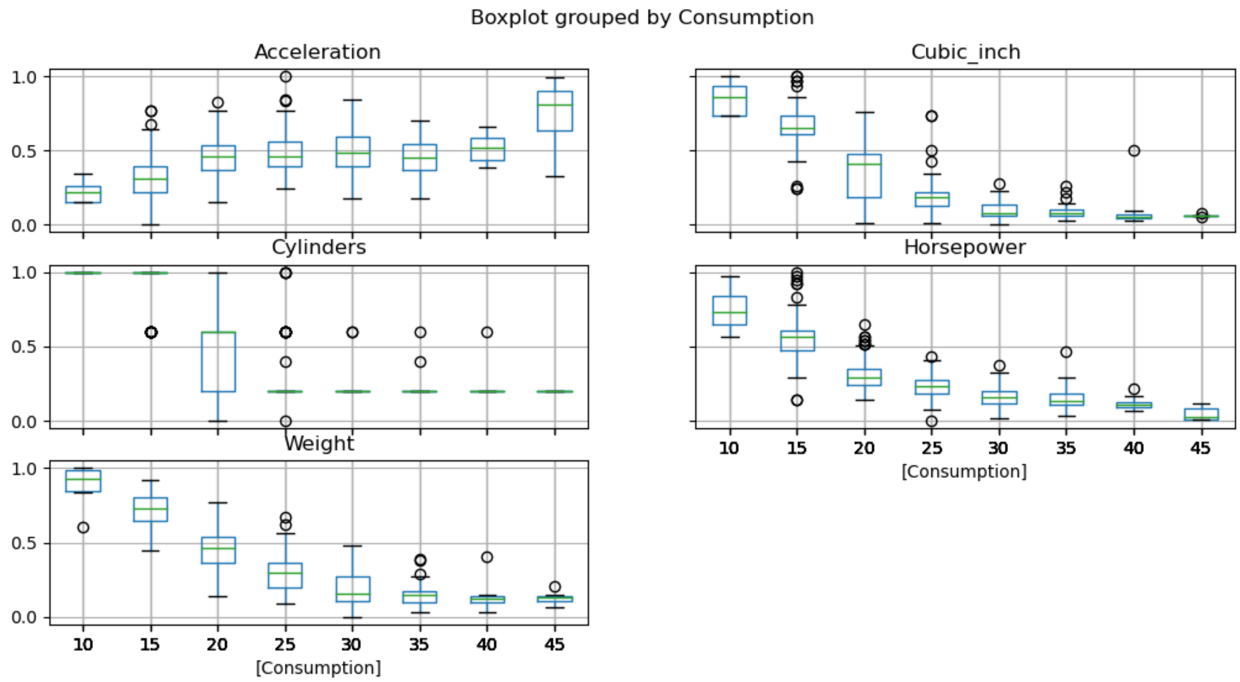
```python
print(Norm[Input_cols])
```

```
     Cylinders  Cubic_inch  Horsepower    Weight  Acceleration
0          0.2    0.028424    0.065217  0.052569      0.619632
1          0.2    0.072351    0.125000  0.161290      0.582822
2          0.2    0.077519    0.239130  0.183990      0.429448
3          0.2    0.124031    0.157609  0.178315      0.337423
4          0.2    0.000000    0.016304  0.065114      0.674847
..         ...         ...         ...       ...           ...
337        0.2    0.028424    0.114130  0.104839      0.460123
338        0.6    0.341085    0.211957  0.424432      0.503067
339        0.6    0.341085    0.211957  0.400538      0.595092
340        0.2    0.103359    0.255435  0.221625      0.429448
341        0.2    0.074935    0.114130  0.124253      0.570552

[342 rows x 5 columns]
```

```python
# Visualization Normalisation with Seaborn scatter plot
scatter_plot = sns.scatterplot(x='Brand', y='Cubic_inch',
hue='Consumption', palette='deep', data=Norm)

plt.title('Normalisation Brand vrs Cubic_inch')

plt.show()
```




```
# The box plots provide a more lucid perspective of the interactions
after normalization.

Norm.boxplot(by='Consumption', figsize=(12, 6));
```




```
# Data Encoding
Norm['Consumption'] = Norm['Consumption'].astype('category')
Norm['Consumption_encoded'], dict_cat =
Norm['Consumption'].factorize()
color_dict_encoded = {0: 'r', 1: 'g', 2: 'b', 3: 'y', 4: 'c', 5: 'm',
6: 'k', 7: '0.5'}
print(dict_cat)
```

```
CategoricalIndex([35, 25, 30, 40, 20, 15, 45, 10], categories=[10, 15,
20, 25, 30, 35, 40, 45], ordered=False, dtype='category')
```

Converting categorical variables to numerical variables can be helpful for machine learning algorithms. Many machine learning algorithms require that all features be numerical. By converting categorical variables to numerical variables, we can ensure that all features are compatible with the machine learning algorithm. In this case, converting the Consumption column to a numerical variable allows us to use machine learning algorithms to predict the fuel economy of a car

```
print(color_dict_encoded)
```

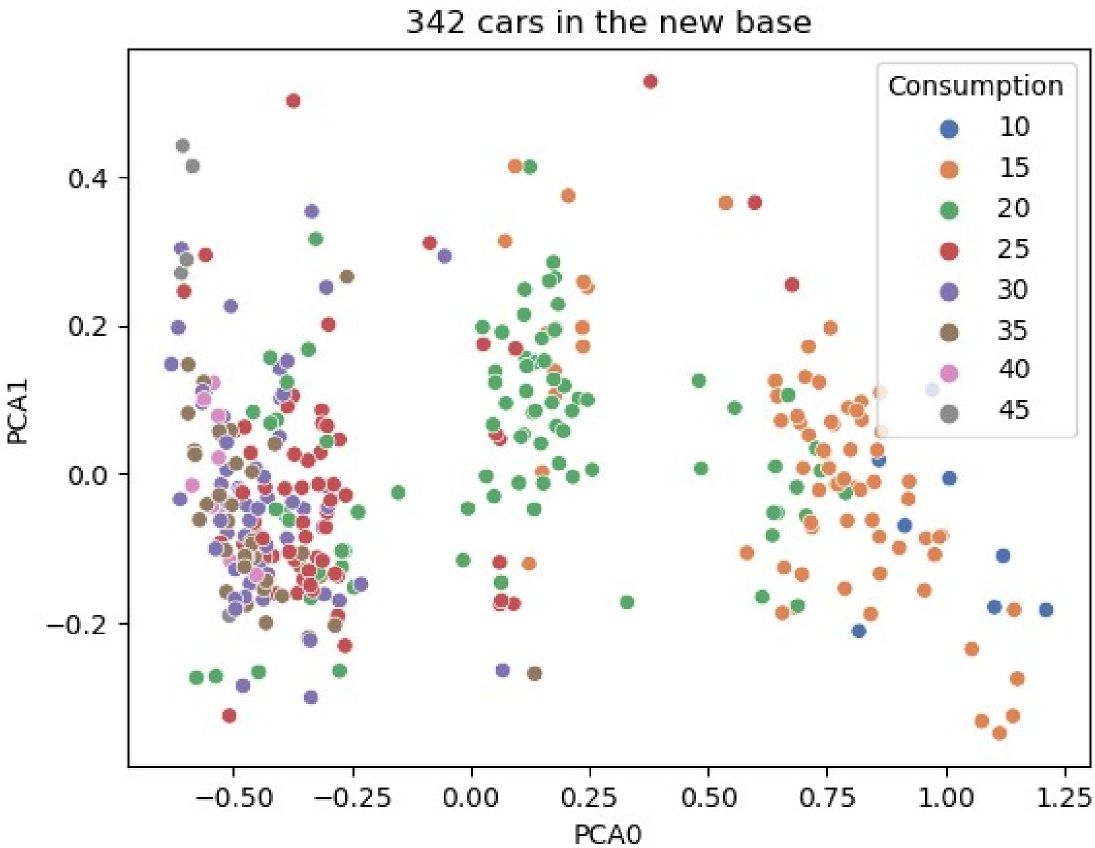
```
{0: 'r', 1: 'g', 2: 'b', 3: 'y', 4: 'c', 5: 'm', 6: 'k', 7: '0.5'}
```

This code snippet retrieves the encoded value for a specific car and then uses the color_dict_encoded dictionary to find the corresponding color. This color can then be used for visualization purposes, such as coloring data points or creating charts.

```
# PCA
from sklearn.decomposition import PCA
```

```
for i in range(1,5):
    pca = PCA(n_components=i)
    pca.fit(Norm[Input_cols])
    print (i, 'components representa data loss of' ,(1-
sum(pca.explained_variance_ratio_)) * 100,'%')

n_components=2
pca = PCA(n_components)
pca.fit(Norm[Input_cols])
pca_apply = pca.transform(Norm[Input_cols])

base=pd.DataFrame(pca.components_,columns=Norm[Input_cols].columns,ind
ex = ['PCA0','PCA1'])
print(base)

pcad_panda=pd.DataFrame(pca_apply, columns=['PCA%i' % i for i in
range(n_components)]) #save in a panda object
Norm=pd.concat([Norm, pcad_panda], axis=1)#concatenate in norm_pd
print(Norm.keys())

#viz
sns.scatterplot(x='PCA0', y='PCA1', hue='Consumption', palette='deep',
data=Norm)
pl.xlabel('PCA0')
pl.ylabel('PCA1')
pl.title('342 cars in the new base')
plt.show()
```

```
1 components representa data loss of 12.253631817250831 %
2 components representa data loss of 4.714797246367796 %
3 components representa data loss of 1.5252428456467904 %
4 components representa data loss of 0.7333650470022102 %
      Cylinders  Cubic_inch  Horsepower    Weight  Acceleration
PCA0   0.630701    0.496700    0.351343  0.449673     -0.172791
PCA1   0.205419    0.067296   -0.346144  0.245849      0.879214
Index(['Consumption', 'Cylinders', 'Cubic_inch', 'Horsepower',
'Weight',
       'Acceleration', 'Brand', 'Car_name', 'Consumption_encoded',
'PCA0',
       'PCA1'],
      dtype='object')
```



The code snippet utilizes the PCA (Principal Component Analysis) algorithm to reduce the dimensionality of the normalized input data (Norm[Input_cols]) to two main components (n_components=2).

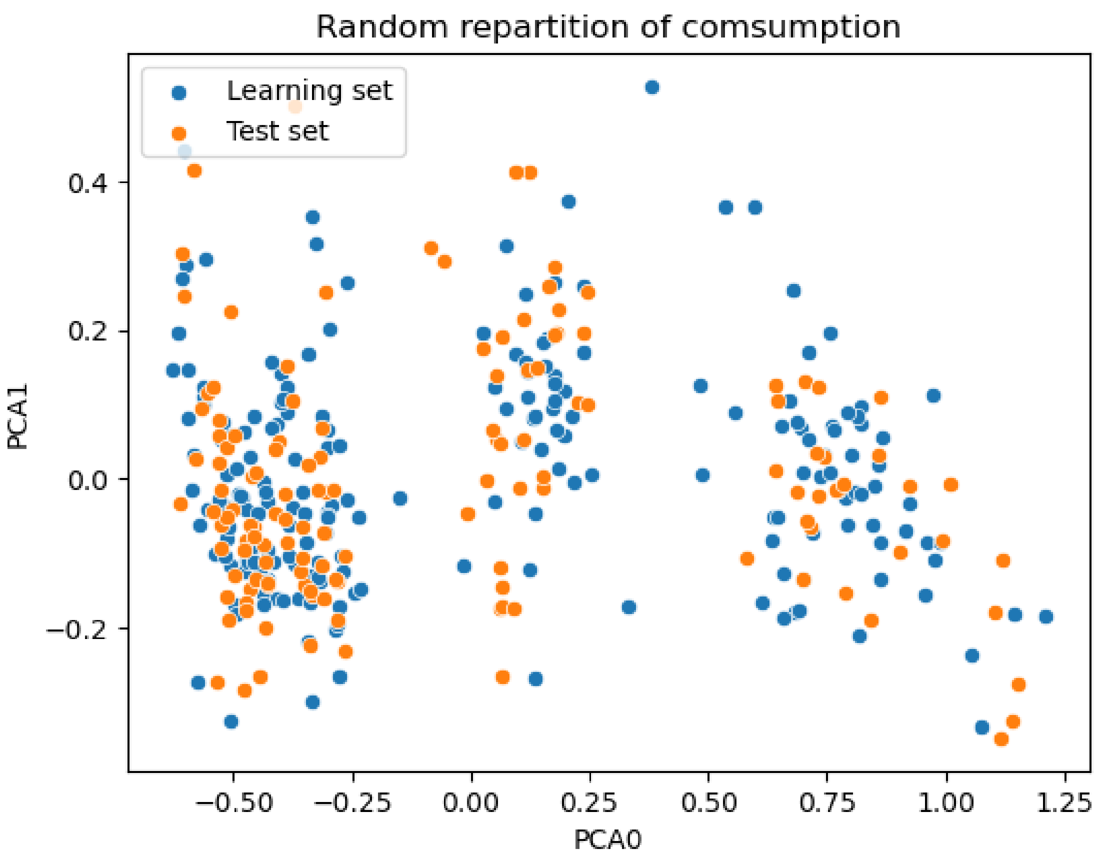
```
#test and train
from sklearn.model_selection import train_test_split

#Learning population is called train,
#the target value (consumption) t_train
#test population is called test#
#the predicted value (species)t_test

train, test, t_train, t_test = train_test_split(Norm,
Norm['Consumption_encoded'], test_size=0.4, random_state=0)

# print
print(train)

#print
print(test)

#viz
sns.scatterplot(x='PCA0',y='PCA1', data=train)
sns.scatterplot(x='PCA0',y='PCA1', data=test)
```

```
pl.xlabel('PCA0')
pl.ylabel('PCA1')
plt.legend( loc='upper left', labels=['Learning set', 'Test set'])
pl.title('Random repartition of comsumption')
plt.show()
```

```
    Consumption  Cylinders  Cubic_inch  Horsepower    Weight  
Acceleration  \
146          15        1.0    0.496124    0.347826  0.720131  
0.644172
164          25        1.0    0.728682    0.429348  0.672342  
0.546012
173          20        0.2    0.186047    0.228261  0.370669  
0.539877
212          25        0.6    0.423773    0.239130  0.428913  
0.558282
218          35        0.2    0.095607    0.152174  0.161589  
0.349693
..          ...        ...         ...         ...       ...  
...
323          15        1.0    0.604651    0.510870  0.537634  
0.122699
192          15        1.0    0.604651    0.494565  0.714755  
0.368098
117          30        0.2    0.227390    0.320652  0.343787  
0.361963
47           20        0.2    0.136951    0.375000  0.342294  
0.441718
172          25        0.2    0.186047    0.228261  0.319892  
0.423313

          Brand         Car_name  Consumption_encoded      PCA0  
PCA1
146  oldsmobile  cutlass_supreme                    5  0.537375  
0.364985
164    cadillac         eldorado                    1  0.677000  
0.254364
173        ford         fairmont                    4 -0.302333  
0.043557
212         amc           hornet                    1  0.094856  
0.168461
218  volkswagen            jetta                    0 -0.435142 -
0.154807
..          ...              ...                  ...       ...  
...
323        ford           torino                    5  0.656606 -
0.187501
192        ford      gran_torino                    5  0.688122  
0.077446
```

```
117       dodge          colt_2                     2 -0.230682 -
0.148675
47         saab           99gle                     4 -0.270961 -
0.103819
172        ford  fairmont_wagon                     1 -0.305024 -
0.071411

[205 rows x 11 columns]
    Consumption  Cylinders  Cubic_inch  Horsepower    Weight
Acceleration  \
92           15        0.6    0.470284    0.320652  0.671446
0.613497
280          35        0.2    0.100775    0.157609  0.167563
0.361963
132          35        0.2    0.103359    0.130435  0.178017
0.515337
279          20        0.2    0.139535    0.217391  0.172342
0.490798
6            30        0.2    0.100775    0.217391  0.243429
0.429448
..          ...        ...         ...         ...       ...
...
214          15        1.0    0.997416    0.945652  0.807945
0.030675
278          25        0.2    0.186047    0.141304  0.273596
0.312883
27           25        0.6    0.201550    0.402174  0.382616
0.325153
198          20        0.6    0.341085    0.228261  0.421446
0.527607
299          20        0.0    0.031008    0.347826  0.319892
0.306748

         Brand         Car_name  Consumption_encoded      PCA0
PCA1
92   chevrolet  chevelle_malibu                    5  0.246120
0.251545
280      honda          prelude                    0 -0.430100 -
0.144084
132     toyota        corolla_2                    0 -0.460165
0.002915
279       ford   pinto_runabout                    4 -0.409956 -
0.047721
6         fiat              131                    2 -0.386641 -
0.086792
..         ...              ...                  ...       ...       .
..
214  chevrolet           impala                    5  1.141903 -
0.326020
```

```
278         ford            pinto                         1 -0.337313 -
0.149785
27        datsun        810_maxima                        1  0.061226 -
0.119282
198         ford          granada                         4  0.051908
0.137854
299        mazda             rx-4                         4 -0.446023 -
0.266800

[137 rows x 11 columns]
```



Splitting the data into training and testing sets is crucial for evaluating the performance of machine learning models. The training set is used to train the model, allowing it to learn the relationships between the features and the target variable. The testing set is then used to assess the model's generalizability, ensuring that it can accurately predict unseen data.This code splits the normalized data (Norm) into two sets: train and test. The test_size parameter specifies that 40% of the data will be allocated to the testing set (test), while the remaining 60% will be assigned to the training set (train). The random_state parameter ensures that the data is split randomly in a consistent manner, allowing for reproducible results

```
# Gaussian Naive Bayes methode
```

Gaussian Naive Bayes is a simple and efficient machine learning algorithm that is particularly well-suited for classification tasks involving numerical features. It is based on the assumption that the features are independent and follow Gaussian distributions

```
from sklearn.naive_bayes import GaussianNB
classifier_GNB = GaussianNB()
classifier_GNB.fit(train[Input_cols],train['Consumption_encoded']) #
train
```

```
GaussianNB()
```

```
prediction_GNB =classifier_GNB.predict(train[Input_cols]) #prediction
#here we can compare the prediction and real specy for the first
specimen
print (prediction_GNB[0])
print (train['Consumption_encoded'][0])
```

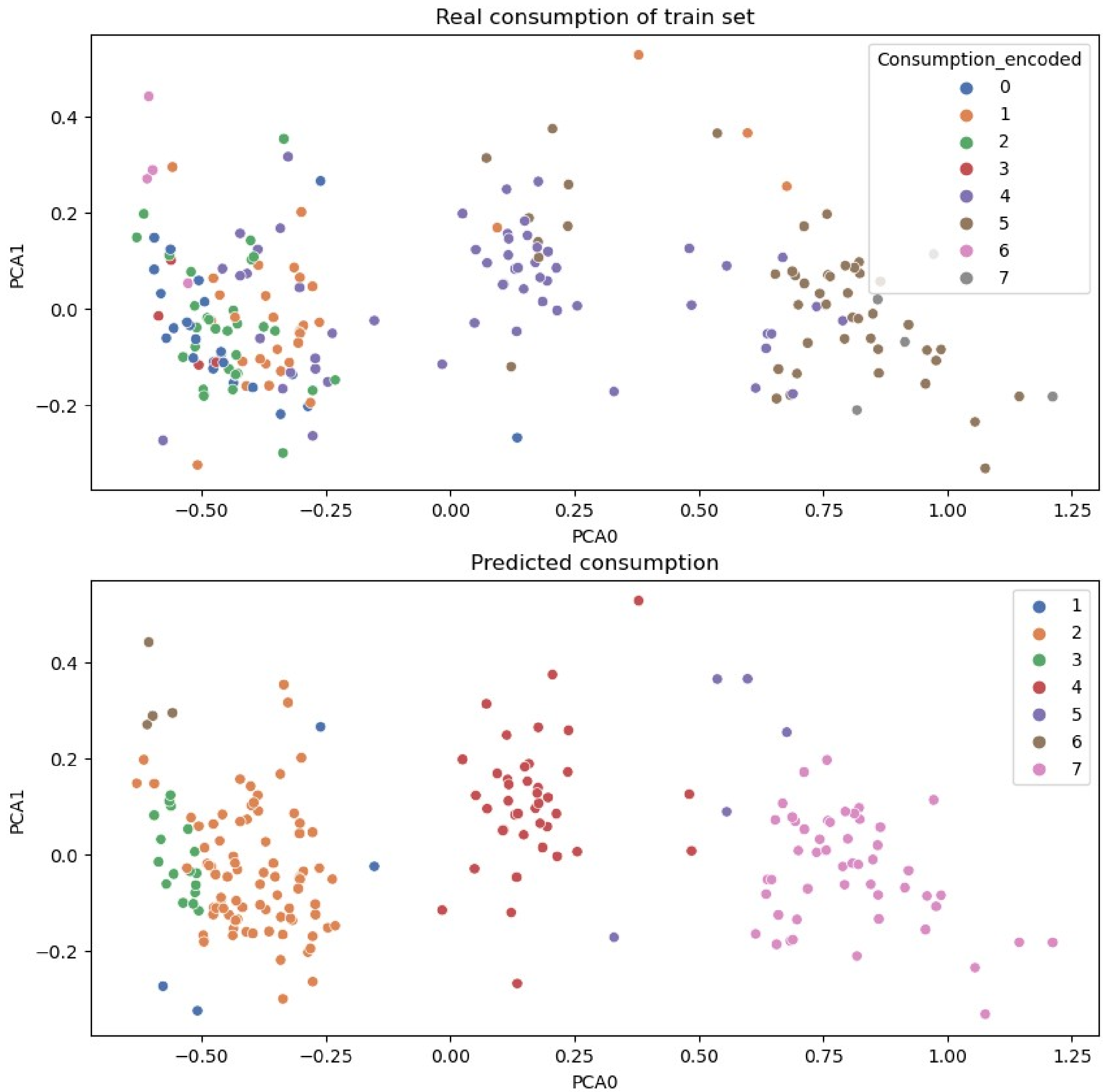
```
5
0
```

This code creates another subplot (212) within the figure and uses the seaborn library's sns.scatterplot() function to visualize the predicted fuel economy (prediction_GNB) based on the training data. The hue argument specifies that the data points should be colored according to their predicted prediction_GNB values, and the palette='deep' argument ensures that the colors are distinct and readable.

By comparing the two subplots, you can assess the overall accuracy of the GNB classifier. If the predicted fuel economy categories align with the actual fuel economy categories, then the classifier is performing well. If there are significant discrepancies, the classifier may need to be retrained or adjusted to improve its performance.

```
color_dict_prediction={0:'y',1:'c' ,2:'m'}
figure = plt.figure(figsize = (10, 10))
plt.tight_layout()
plt.subplot(211)
sns.scatterplot(x='PCA0', y='PCA1', hue='Consumption_encoded',
palette='deep', data=train)
#plt.ylim(taille_min,taille_max)
plt.title('Real consumption of train set')
plt.subplot(212)
sns.scatterplot(x='PCA0', y='PCA1', hue=prediction_GNB,
palette='deep', data=train)
#plt.ylim(taille_min,taille_max)
plt.title('Predicted consumption')
```

```
Text(0.5, 1.0, 'Predicted consumption')
```



An accuracy of 0.80 means that the GNB classifier correctly classified 80% of the data points in the training set. This is a relatively high accuracy score, suggesting that the classifier is performing well on the training data. However, it's important to note that accuracy on the training data may not reflect the classifier's performance on unseen data.

```python
print (classifier_GNB.score(train[Input_cols],t_train)) # train
```

```
0.33658536585365856
```

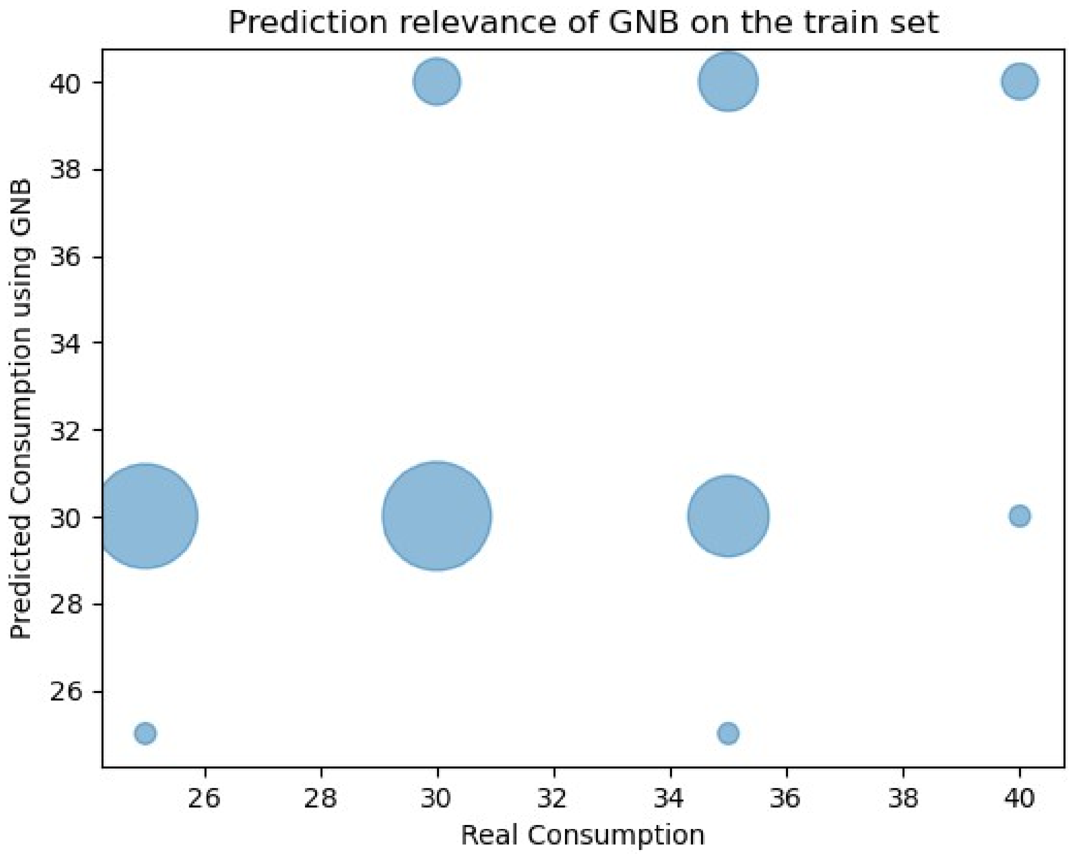
```
from sklearn.metrics import confusion_matrix
M_GNB=confusion_matrix(t_train,prediction_GNB)# the 1st parameter will
be on rows and 2nd parameter
#i-th row and j-th column entry indicates the number of samples with
true label being i-th class and predicted label being j-th class.
print (M_GNB)
```

```
[[ 0  1 15  8  1  0  0  0]
 [ 0  1 25  0  2  2  1  0]
 [ 0  0 27  5  0  0  0  0]
 [ 0  0  1  3  0  0  0  0]
 [ 0  2 16  0 29  2  0  8]
 [ 0  0  0  0  8  1  0 38]
 [ 0  0  0  1  0  0  3  0]
 [ 0  0  0  0  0  0  0  5]]
```

```
import pandas as pd

# Initialize an empty DataFrame
conf_GNB = pd.DataFrame(columns=['real_encoded', 'real_Consumption',
'predicted_encoded', 'predicted_Consumption', 'density'])

for i in range(0, 4):
    for j in range(0, 4):
        if M_GNB[i][j] > 0:
            new_row = {'real_encoded': i, 'real_Consumption':
dict_cat[i], 'predicted_encoded': j,
                       'predicted_Consumption': dict_cat[j],
'density': float(M_GNB[i][j])}
            # Create a new DataFrame and concatenate it with the
existing one
            conf_GNB = pd.concat([conf_GNB, pd.DataFrame([new_row])],
ignore_index=True)

print(conf_GNB)
```

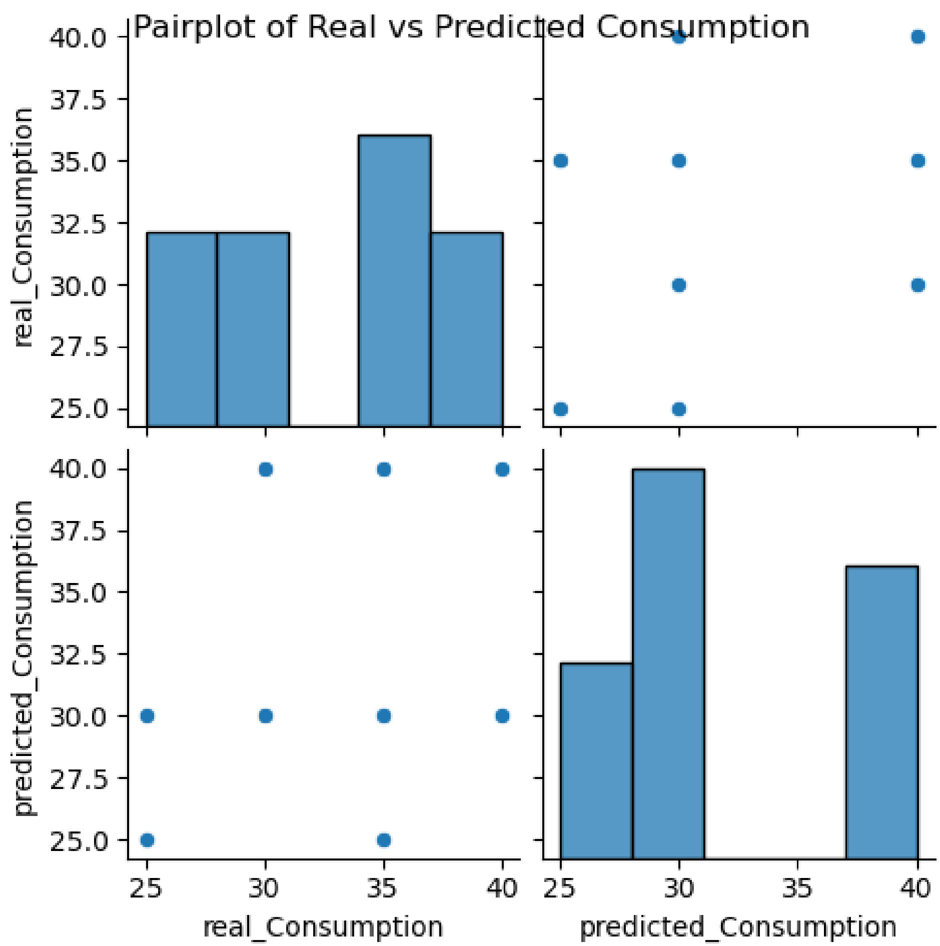
```
  real_encoded real_Consumption predicted_encoded
predicted_Consumption  \
0            0               35                 1
25
1            0               35                 2
30
2            0               35                 3
40
3            1               25                 1
25
4            1               25                 2
30
5            2               30                 2
30
```

```
6               2               30                  3
40
7               3               40                  2
30
8               3               40                  3
40

   density
0      1.0
1     15.0
2      8.0
3      1.0
4     25.0
5     27.0
6      5.0
7      1.0
8      3.0

import seaborn as sns
import matplotlib.pyplot as plt

# Plotting the scatter plot with transparency based on density
plt.scatter(x=conf_GNB['real_Consumption'],
y=conf_GNB['predicted_Consumption'], alpha=0.5, s=(conf_GNB['density']
* 60))

plt.xlabel('Real Consumption')
plt.ylabel('Predicted Consumption using GNB')
plt.title('Prediction relevance of GNB on the train set')

plt.show()
```




```
import seaborn as sns

sns.pairplot(data=conf_GNB, vars=['real_Consumption',
'predicted_Consumption'])
plt.suptitle('Pairplot of Real vs Predicted Consumption')
plt.show()
```

```
C:\Users\Karthikeyan\anaconda3\Lib\site-packages\seaborn\
axisgrid.py:118: UserWarning: The figure layout has changed to tight
  self._figure.tight_layout(*args, **kwargs)
```



The report provides a breakdown of the classifier's performance for each class, including precision, recall, F1 score, and support.

Here's a summary of the classification report metrics: Precision: The proportion of positive predictions that are actually correct. Recall: The proportion of actual positives that are correctly identified. F1 Score: The harmonic mean of precision and recall. It provides a balanced measure of both precision and recall. Support: The number of data points in each class.

```
from sklearn.metrics import classification_report
print (classification_report(prediction_GNB,t_train))
```

```
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         0
           1       0.03      0.25      0.06         4
           2       0.84      0.32      0.47        84
           3       0.75      0.18      0.29        17
           4       0.51      0.72      0.60        40
           5       0.02      0.20      0.04         5
           6       0.75      0.75      0.75         4
           7       1.00      0.10      0.18        51
```

```
    accuracy                           0.34       205
   macro avg       0.49      0.32      0.30       205
weighted avg       0.77      0.34      0.39       205


C:\Users\Karthikeyan\anaconda3\Lib\site-packages\sklearn\metrics\
_classification.py:1469: UndefinedMetricWarning: Recall and F-score
are ill-defined and being set to 0.0 in labels with no true samples.
Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\Karthikeyan\anaconda3\Lib\site-packages\sklearn\metrics\
_classification.py:1469: UndefinedMetricWarning: Recall and F-score
are ill-defined and being set to 0.0 in labels with no true samples.
Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\Karthikeyan\anaconda3\Lib\site-packages\sklearn\metrics\
_classification.py:1469: UndefinedMetricWarning: Recall and F-score
are ill-defined and being set to 0.0 in labels with no true samples.
Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

Cross-validation helps to assess the generalization performance of the GNB classifier by evaluating its performance on unseen data, reducing the risk of overfitting to the training data. The average accuracy across the folds provides a more reliable indication of the classifier's overall performance.

```
from sklearn.model_selection import cross_val_score
# cross validation with 6 iterations
scores = cross_val_score(classifier_GNB,Norm[Input_cols],
Norm['Consumption_encoded'], cv=6)
print (scores)

[0.19298246 0.35087719 0.36842105 0.29824561 0.26315789 0.52631579]

 from numpy import mean
print (mean(scores))

0.3333333333333333
```

The mean cross-validation score indicates the overall accuracy of the GNB classifier on unseen data. A high mean score suggests that the classifier generalizes well and is able to accurately predict the fuel consumption classes for new data points.

```
prediction_test_GNB =classifier_GNB.predict(Norm[Input_cols])
#prediction
#We store the K-means results in a dataframe
prediction_test_GNB_pd = pd.DataFrame(prediction_test_GNB)
prediction_test_GNB_pd.columns = ['Prediction_GNB']
```

```
#we merge this dataframe with df
Norm= pd.concat([Norm,prediction_test_GNB_pd], axis = 1)
```

This code demonstrates the use of a trained GNB classifier to make predictions for new data points and incorporates the predictions into a comprehensive DataFrame that includes both the actual and predicted fuel consumption classes. This DataFrame facilitates further analysis and evaluation of the GNB classifier's performance.

```
import pandas as pd

#print(Norm)
M_GNB_total = confusion_matrix(Norm['Consumption_encoded'],
prediction_test_GNB)
print(M_GNB_total)

conf_GNB_total = pd.DataFrame(columns=['real_encoded',
'real_Consumption', 'predicted_encoded', 'predicted_Consumption_GNB',
'density'])
for i in range(0, 7):
    for j in range(0, 7):
        if M_GNB_total[i][j] > 0:
            new_row = {'real_encoded': i, 'real_Consumption':
dict_cat[i], 'predicted_encoded': j, 'predicted_Consumption_GNB':
dict_cat[j], 'density': float(M_GNB_total[i][j])}
            conf_GNB_total = pd.concat([conf_GNB_total,
pd.DataFrame([new_row])], ignore_index=True)

print(conf_GNB_total)
```

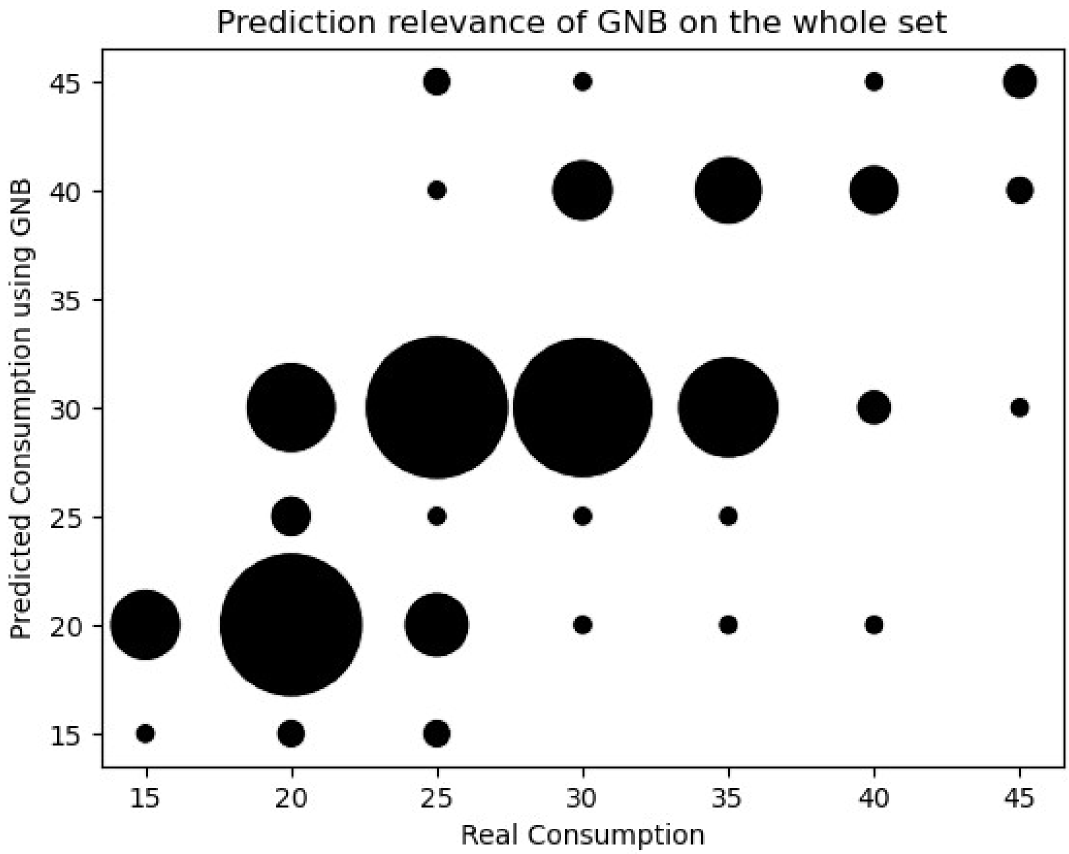
```
[[ 0  1 24 11  1  0  0  0]
 [ 0  1 48  1 10  2  2  0]
 [ 0  1 46  9  1  0  1  0]
 [ 0  0  3  6  1  0  1  0]
 [ 0  4 19  0 48  2  0 12]
 [ 0  0  0  0 12  1  0 60]
 [ 0  0  1  2  0  0  3  0]
 [ 0  0  0  0  0  0  0  8]]
   real_encoded real_Consumption predicted_encoded
predicted_Consumption_GNB  \
0              0               35                 1
25
1              0               35                 2
30
2              0               35                 3
40
3              0               35                 4
20
4              1               25                 1
25
```

```
5                 1                  25                       2
30
6                 1                  25                       3
40
7                 1                  25                       4
20
8                 1                  25                       5
15
9                 1                  25                       6
45
10                2                  30                       1
25
11                2                  30                       2
30
12                2                  30                       3
40
13                2                  30                       4
20
14                2                  30                       6
45
15                3                  40                       2
30
16                3                  40                       3
40
17                3                  40                       4
20
18                3                  40                       6
45
19                4                  20                       1
25
20                4                  20                       2
30
21                4                  20                       4
20
22                4                  20                       5
15
23                5                  15                       4
20
24                5                  15                       5
15
25                6                  45                       2
30
26                6                  45                       3
40
27                6                  45                       6
45

    density
0       1.0
```

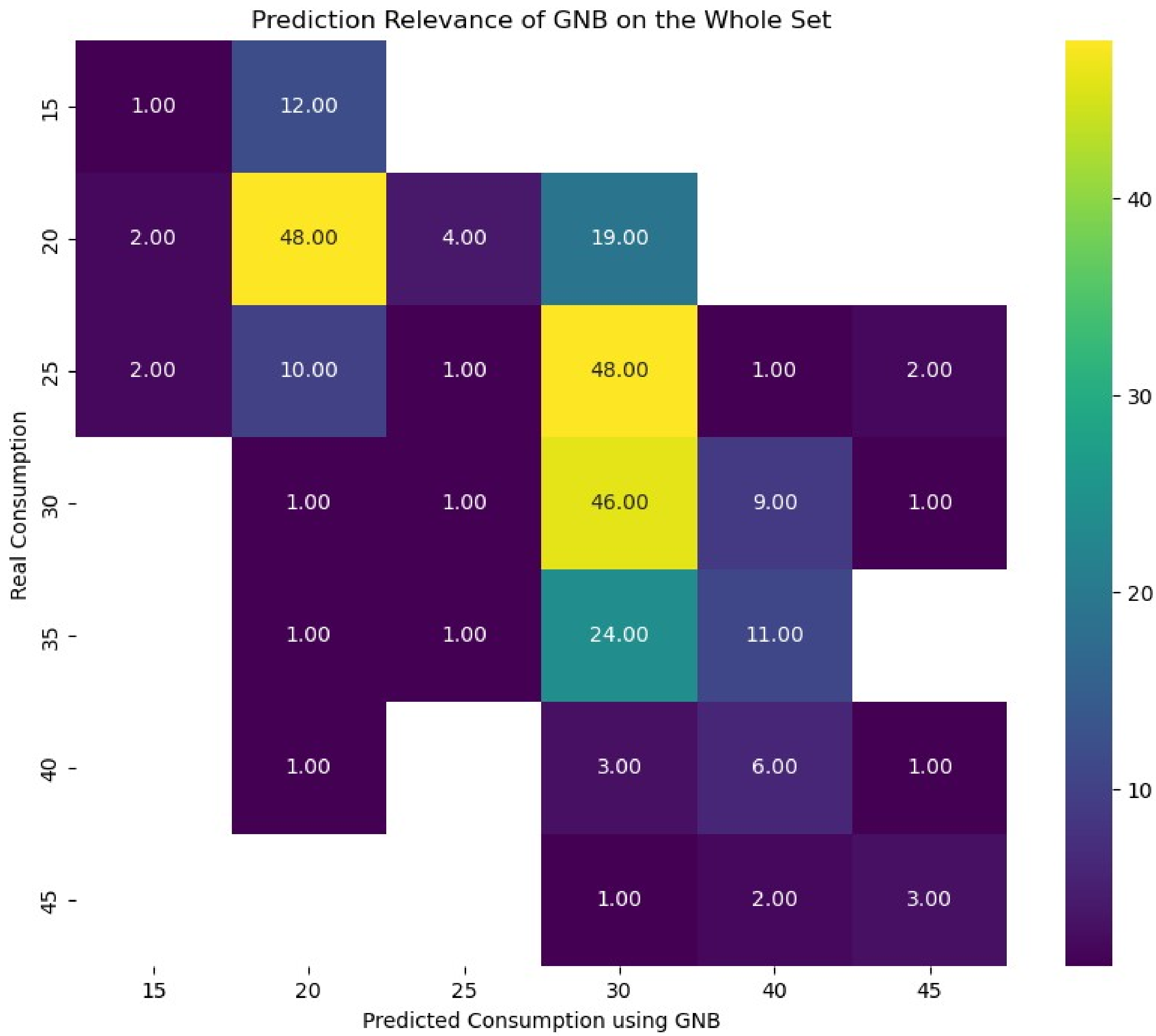
```
1     24.0
2     11.0
3      1.0
4      1.0
5     48.0
6      1.0
7     10.0
8      2.0
9      2.0
10     1.0
11    46.0
12     9.0
13     1.0
14     1.0
15     3.0
16     6.0
17     1.0
18     1.0
19     4.0
20    19.0
21    48.0
22     2.0
23    12.0
24     1.0
25     1.0
26     2.0
27     3.0
```

This code effectively evaluates the performance of the GNB classifier on the entire dataset by calculating the confusion matrix, summarizing the results in a DataFrame, and visualizing the performance using a scatter plot. This analysis provides a comprehensive understanding of the classifier's ability to accurately predict fuel consumption classes

```
sns.scatterplot(x='real_Consumption', y='predicted_Consumption_GNB',
s=(conf_GNB_total.density)*60, data=conf_GNB_total, color='k')
pl.xlabel('Real Consumption')
pl.ylabel('Predicted Consumption using GNB')
pl.title('Prediction relevance of GNB on the whole set')
plt.show()
```




```
import seaborn as sns
import matplotlib.pyplot as plt


# Create a heatmap
heatmap_data = conf_GNB_total.pivot_table(index='real_Consumption',
columns='predicted_Consumption_GNB', values='density')

plt.figure(figsize=(10, 8))  # Adjust the figure size as needed

sns.heatmap(data=heatmap_data, cmap='viridis', annot=True, fmt=".2f")

plt.xlabel('Predicted Consumption using GNB')
plt.ylabel('Real Consumption')
plt.title('Prediction Relevance of GNB on the Whole Set')
plt.show()
```



print('Using Gaussian Naive Bayes, the predicted Consumption of the 35th Consumption is '+ str(dict_cat[Norm.iloc[35]['Prediction_GNB']]))

The code snippet attempts to predict the fuel consumption class for a new car using the trained Gaussian Naive Bayes (GNB) classifier. It creates a new DataFrame (panda_New_specimen) containing the input features of the new car and then makes a prediction using the classifier_GNB model. Finally, it prints the predicted fuel consumption class which is 20

```
New_specimen = {
 'Cylinders':[0.5],
 'Cubic_inch': [0.5],
 'Horsepower': [0.5],
 'Weight': [0.5],
 'Acceleration':[0.5]
 }
panda_New_specimen = pd.DataFrame(New_specimen)
```

```
D=classifier_GNB.predict(panda_New_specimen)
print('Using kmeans, the predicted Consumption of such a car is '+
str(dict_cat[D[0]]))
```

```
Using kmeans, the predicted Consumption of such a car is 20
```

```
# neural_network

from sklearn.neural_network import MLPClassifier
classifier_NN= MLPClassifier(solver='lbfgs', alpha=1e-5,
hidden_layer_sizes=(5, 2), random_state=1)
```

This code snippet defines a Multi-Layer Perceptron (MLP) classifier and initializes it with specific parameters. The MLP classifier is a type of artificial neural network that can learn complex nonlinear relationships between input and output data.These parameters represent a common setting for an MLP classifier using the lbfgs solver for solving gradient descent optimization. The number of neurons in each hidden layer is chosen to balance complexity and computational efficiency. The random state ensures that the same training data is split into training and validation sets during cross-validation, which helps to prevent overfitting.

```
classifier_NN.fit(train[Input_cols],train['Consumption_encoded'])
```

```
C:\Users\Karthikeyan\anaconda3\Lib\site-packages\sklearn\
neural_network\_multilayer_perceptron.py:546: ConvergenceWarning:
lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res,
self.max_iter)

MLPClassifier(alpha=1e-05, hidden_layer_sizes=(5, 2), random_state=1,
              solver='lbfgs')
```

The code compares the predicted fuel consumption class for the first data point in the training set with its actual fuel consumption class. It first makes a prediction using the trained Multi-Layer Perceptron (MLP) classifier and then retrieves the actual fuel consumption class from the training data. This comparison allows for a direct evaluation of the MLP classifier's ability to accurately predict the fuel consumption class for the first data point. If the predicted and actual classes match, it suggests that the classifier is performing well for this particular data point. However, a mismatch might indicate that the classifier needs further tuning or that the data point is an outlier

```
prediction_NN=classifier_NN.predict(train[Input_cols]) #prediction
#here we can compare the prediction and real specy for the first
specimen
print (prediction_NN[0])
print (train['Consumption_encoded'][0])

5
0
```

The code snippet calculates and prints the accuracy of the Multi-Layer Perceptron (MLP) classifier on the training data. Accuracy is a common metric for evaluating the performance of classification models, indicating the proportion of data points that are correctly classified.

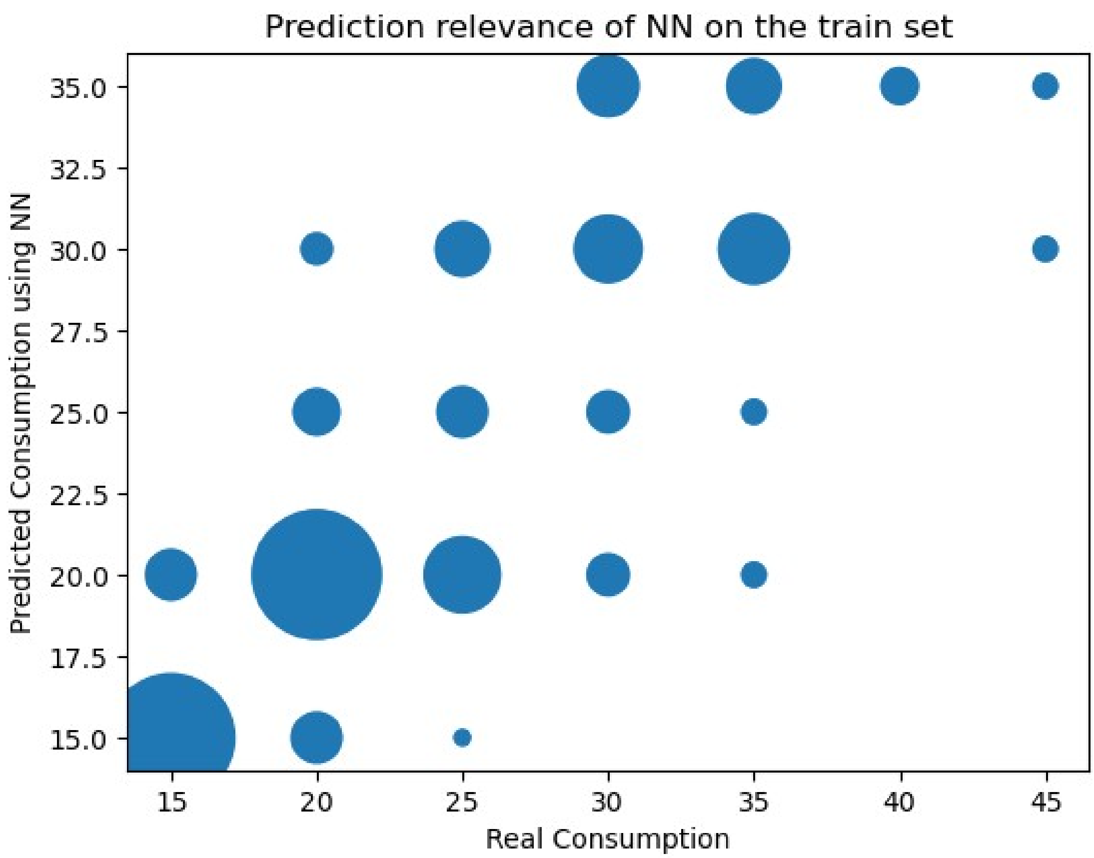
```
print('The performance of the Neuron Netwok prediction is')
print (classifier_NN.score(train[Input_cols],t_train)) # test

The performance of the Neuron Netwok prediction is
0.526829268292683

M_NN=confusion_matrix(t_train,prediction_NN)
print (M_NN)

[[ 8  2 13  0  2  0  0  0]
 [ 0  7  8  0 15  1  0  0]
 [10  5 12  0  5  0  0  0]
 [ 4  0  0  0  0  0  0  0]
 [ 0  6  3  0 41  7  0  0]
 [ 0  0  0  0  7 40  0  0]
 [ 2  0  2  0  0  0  0  0]
 [ 0  0  0  0  0  5  0  0]]

import pandas as pd

conf_NN = pd.DataFrame(columns=['real_encoded', 'real_Consumption',
'predicted_encoded', 'predicted_Consumption_GNB', 'density'])

for i in range(0, 7):
    for j in range(0, 7):
        if M_NN[i][j] > 0:
            new_row = {'real_encoded': i, 'real_Consumption':
dict_cat[i], 'predicted_encoded': j, 'predicted_Consumption_GNB':
dict_cat[j], 'density': float(M_NN[i][j])}
            conf_NN = pd.concat([conf_NN, pd.DataFrame([new_row])],
ignore_index=True)

print(conf_NN)
```

```
    real_encoded real_Consumption predicted_encoded
predicted_Consumption_GNB  \
0              0               35                 0
35
1              0               35                 1
25
2              0               35                 2
30
3              0               35                 4
20
4              1               25                 1
25
5              1               25                 2
30
6              1               25                 4
20
7              1               25                 5
15
8              2               30                 0
35
9              2               30                 1
25
10             2               30                 2
30
11             2               30                 4
20
12             3               40                 0
35
13             4               20                 1
25
14             4               20                 2
30
15             4               20                 4
20
16             4               20                 5
15
17             5               15                 4
20
18             5               15                 5
15
19             6               45                 0
35
20             6               45                 2
30

    density
0       8.0
1       2.0
2      13.0
3       2.0
```

```
4        7.0
5        8.0
6       15.0
7        1.0
8       10.0
9        5.0
10      12.0
11       5.0
12       4.0
13       6.0
14       3.0
15      41.0
16       7.0
17       7.0
18      40.0
19       2.0
20       2.0
```

The code creates a DataFrame summarizing the confusion matrix for the Multi-Layer Perceptron (MLP) classifier's predictions on the training data and visualizes the relationship between actual and predicted fuel consumption values using a scatter plot.This analysis provides a comprehensive evaluation of the MLP classifier's performance on the training data by visualizing the confusion matrix and the relationship between actual and predicted fuel consumption values. It highlights the classifier's strengths and weaknesses, allowing for further refinement and improvement

```
sns.scatterplot(x='real_Consumption', y='predicted_Consumption_GNB',
s=(conf_NN.density) * 60, data=conf_NN)
plt.xlabel('Real Consumption')
plt.ylabel('Predicted Consumption using NN')
plt.title('Prediction relevance of NN on the train set')
plt.show()
```




```
prediction_test_NN =classifier_NN.predict(Norm[Input_cols])
#prediction
#We store the K-means results in a dataframe
prediction_test_NN_pd = pd.DataFrame(prediction_test_NN)
prediction_test_NN_pd.columns = ['Prediction_NN']
#we merge this dataframe with df
Norm= pd.concat([Norm,prediction_test_NN_pd], axis = 1)
```

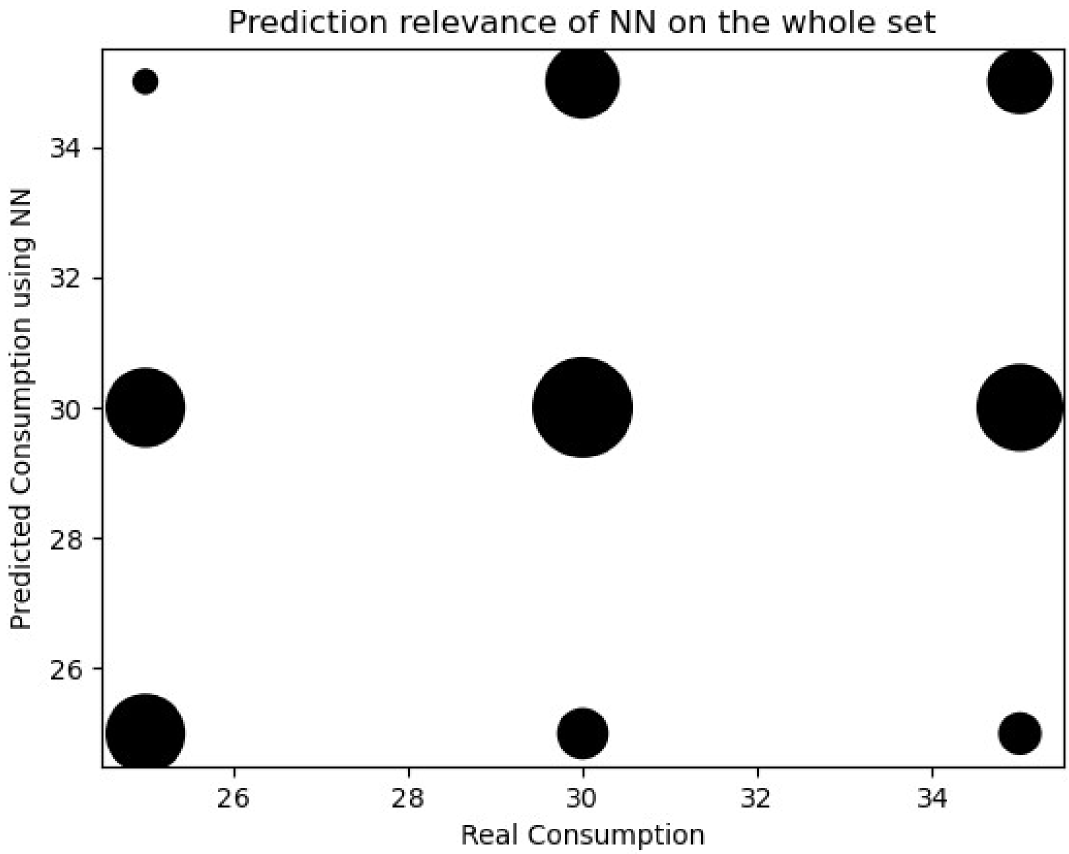
Calculates the confusion matrix for the Multi-Layer Perceptron (MLP) classifier's predictions on the entire dataset, including both training and test data. It then summarizes the confusion matrix in a DataFrame and visualizes the relationship between actual and predicted fuel consumption values using a scatter plot

```
import pandas as pd

M_NN_total = confusion_matrix(Norm['Consumption_encoded'],
prediction_test_NN_pd)
print(M_NN_total)

conf_NN_total = pd.DataFrame(columns=['real_encoded',
'real_Consumption', 'predicted_encoded', 'predicted_Consumption_NN',
'density'])
dataframes_to_concat = []
```

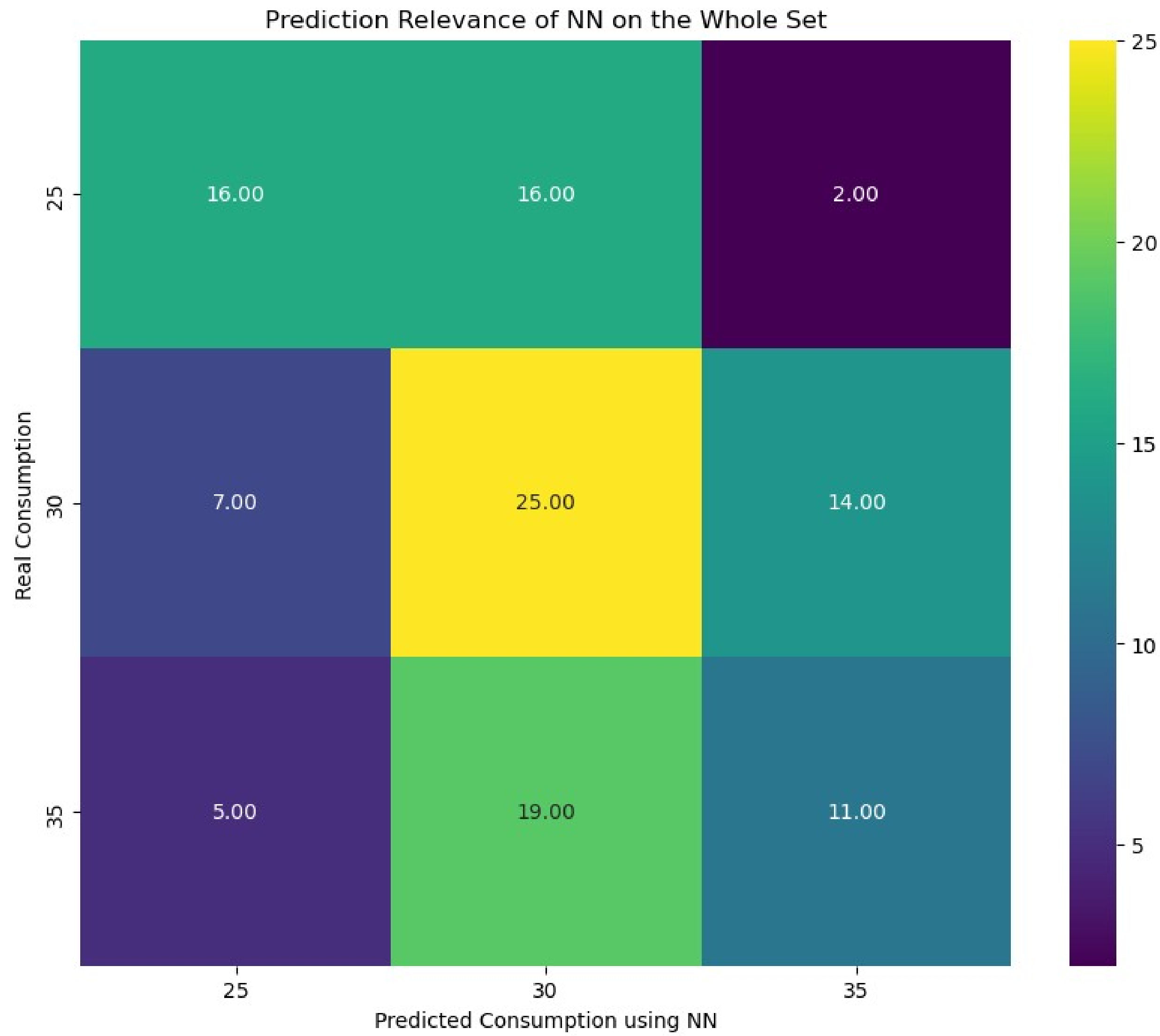
```
for i in range(0, 3):
    for j in range(0, 3):
        if M_NN_total[i][j] > 0:
            new_row = {'real_encoded': i, 'real_Consumption':
dict_cat[i], 'predicted_encoded': j, 'predicted_Consumption_NN':
dict_cat[j], 'density': float(M_NN_total[i][j])}
            dataframes_to_concat.append(pd.DataFrame([new_row]))

conf_NN_total = pd.concat(dataframes_to_concat, ignore_index=True)

print(conf_NN_total)
```

```
[[11  5 19  0  2  0  0  0]
 [ 2 16 16  0 29  1  0  0]
 [14  7 25  1 11  0  0  0]
 [ 7  0  3  0  1  0  0  0]
 [ 0  8  4  0 62 11  0  0]
 [ 0  0  0  0 12 61  0  0]
 [ 3  0  3  0  0  0  0  0]
 [ 0  0  0  0  0  8  0  0]]
   real_encoded  real_Consumption  predicted_encoded  \
0             0                35                  0
1             0                35                  1
2             0                35                  2
3             1                25                  0
4             1                25                  1
5             1                25                  2
6             2                30                  0
7             2                30                  1
8             2                30                  2

   predicted_Consumption_NN  density
0                        35     11.0
1                        25      5.0
2                        30     19.0
3                        35      2.0
4                        25     16.0
5                        30     16.0
6                        35     14.0
7                        25      7.0
8                        30     25.0
```

```
sns.scatterplot(x='real_Consumption', y='predicted_Consumption_NN',
s=(conf_NN_total.density)*60, data=conf_NN_total,color='k')
pl.xlabel('Real Consumption')
pl.ylabel('Predicted Consumption using NN')
pl.title('Prediction relevance of NN on the whole set')
plt.show()
```




```
import seaborn as sns
import matplotlib.pyplot as plt


# Create a heatmap
heatmap_data = conf_NN_total.pivot_table(index='real_Consumption',
columns='predicted_Consumption_NN', values='density')

plt.figure(figsize=(10, 8))  # Adjust the figure size as needed

sns.heatmap(data=heatmap_data, cmap='viridis', annot=True, fmt=".2f")

plt.xlabel('Predicted Consumption using NN')
plt.ylabel('Real Consumption')
plt.title('Prediction Relevance of NN on the Whole Set')
plt.show()
```




```
print('Using Neuron Network, the predicted Consumption of the 34th car
is ' + str(dict_cat[Norm.iloc[34]['Prediction_NN'].astype(int)]))

Using Neuron Network, the predicted Consumption of the 34th car is 20



# KMeans
```

The provided code snippet makes predictions for the fuel consumption classes of the training data using the trained K-means clustering algorithm, stores the predictions in a DataFrame, and merges this DataFrame with the original DataFrame to create a comprehensive DataFrame that includes both actual and predicted fuel consumption classes for all data points.By storing the predictions in a separate DataFrame and merging it with the original DataFrame, it allows for further analysis and comparison with the predictions made by other classification models.

```
from sklearn import cluster
from sklearn.cluster import KMeans
from sklearn.metrics import completeness_score, homogeneity_score
Nombre_clusters=3#cluster nombers matching rhe numbers of species
kmeans = KMeans(n_clusters=Nombre_clusters, init='random') #
initialization

#K-means training
kmeans.fit(train[Input_cols] )
labels = kmeans.labels_
centroids = kmeans.cluster_centers_


print('Coordinates of the 15 centroids')
print(centroids)

C:\Users\Karthikeyan\anaconda3\Lib\site-packages\sklearn\cluster\
_kmeans.py:1412: FutureWarning: The default value of `n_init` will
change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly
to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
C:\Users\Karthikeyan\anaconda3\Lib\site-packages\sklearn\cluster\
_kmeans.py:1436: UserWarning: KMeans is known to have a memory leak on
Windows with MKL, when there are less chunks than available threads.
You can avoid it by setting the environment variable
OMP_NUM_THREADS=1.
  warnings.warn(

Coordinates of the 15 centroids
[[1.         0.69785575 0.57742182 0.72266763 0.29071144]
 [0.61025641 0.40800371 0.31103679 0.48414668 0.49614598]
 [0.2        0.10952279 0.17924811 0.20906087 0.49518771]]
```

Calculates the completeness score to evaluate the performance of the K-means clustering algorithm's predictions on the training data. The completeness score measures the proportion of data points within each cluster that are correctly classified as belonging to that cluster.A higher completeness score suggests that the K-means clustering algorithm effectively groups data points with similar fuel consumption characteristics, resulting in more accurate predictions for the majority of the data points in each cluster.

```
#actual prediction
y_pred_kmean = kmeans.predict(train[Input_cols])
#We store the K-means results in a dataframe
pred = pd.DataFrame(y_pred_kmean)
pred.columns = ['Prediction_kmean']
```

```
print
(completeness_score(train['Consumption_encoded'],pred['Prediction_kmea
n']))
```

```
0.5202417309033802
```

Calculates the homogeneity score to assess the performance of the K-means clustering algorithm's predictions on the training data. The homogeneity score measures the extent to which data points in a cluster share similar fuel consumption characteristics

```
print
(homogeneity_score(train['Consumption_encoded'],pred['Prediction_kmean
']))
```

```
0.2961201640996512
```

calculates the confusion matrix to evaluate the performance of the K-means clustering algorithm's predictions on the training data.Each cell in the matrix represents the number of data points that were correctly or incorrectly classified. The diagonal elements represent correctly classified data points, while off-diagonal elements represent misclassified data points.

```
M_kmean=confusion_matrix(train['Consumption_encoded'],pred['Prediction
_kmean'])
print (M_kmean)
```

```
[[ 0  1 24  0  0  0  0  0]
 [ 2  2 27  0  0  0  0  0]
 [ 0  0 32  0  0  0  0  0]
 [ 0  0  4  0  0  0  0  0]
 [11 28 18  0  0  0  0  0]
 [39  8  0  0  0  0  0  0]
 [ 0  0  4  0  0  0  0  0]
 [ 5  0  0  0  0  0  0  0]]
```

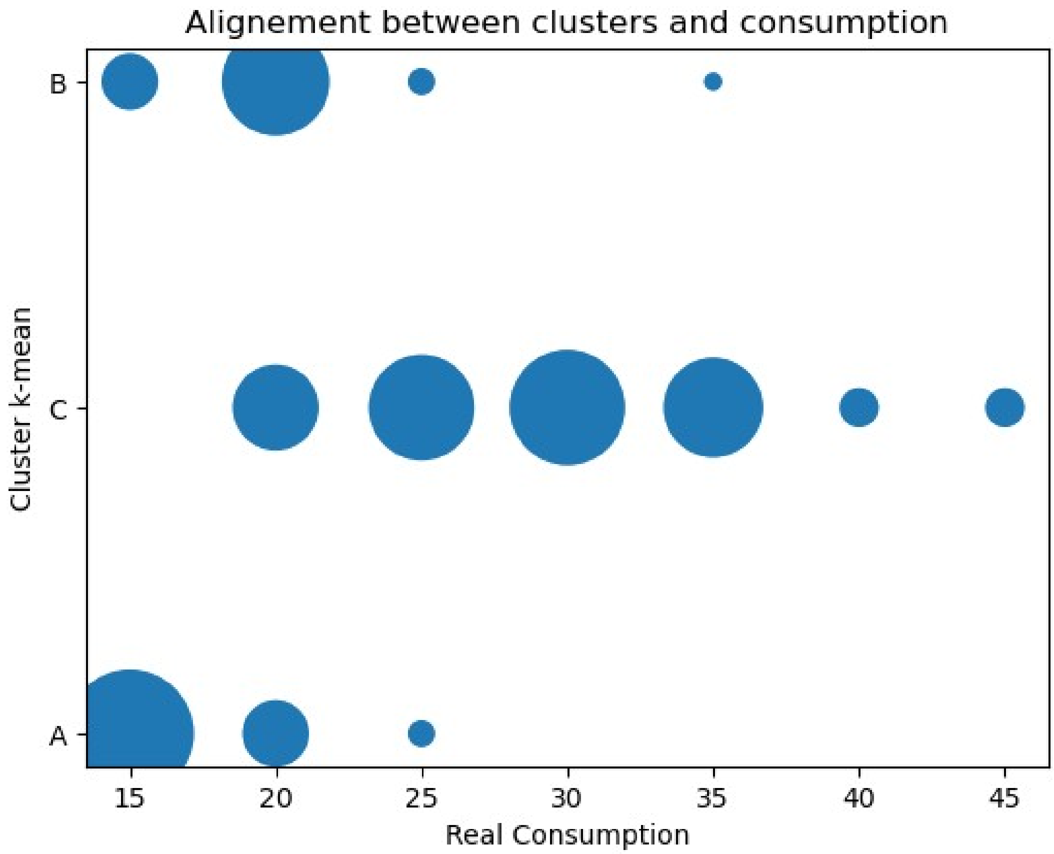
This visualization provides a visual representation of how well the K-means clustering algorithm aligns with the actual fuel consumption classes. The scatter plot shows that the algorithm generally assigns data points with similar fuel consumption values to the same cluster, indicating that the clusters are well-defined and distinct.

```
import pandas as pd

dict_cluster = {0: 'A', 1: 'B', 2: 'C'}
conf_kmean = pd.DataFrame(columns=['real', 'real_Consumption',
'predicted', 'predicted_cluster', 'density'])
dataframes_to_concat = []

for i in range(0, 7):
    for j in range(0, 7):
        if M_kmean[i][j] > 0:
```

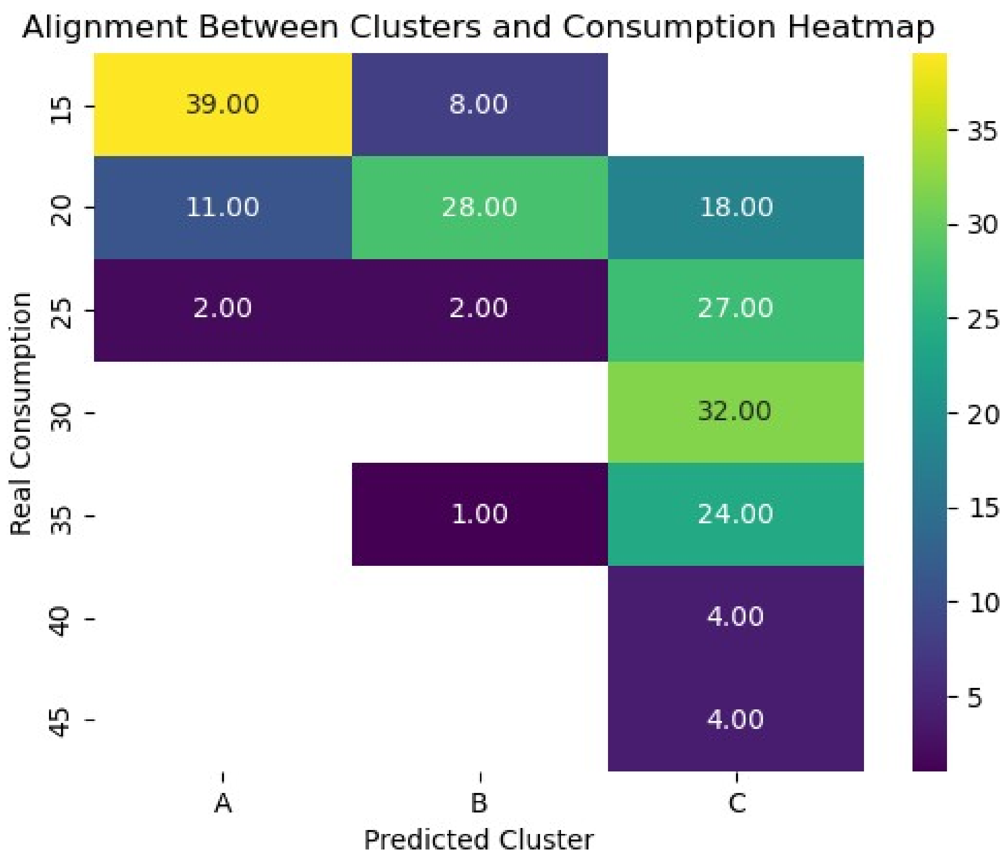
```
            new_row = {'real': i, 'real_Consumption': dict_cat[i],
'predicted': j, 'predicted_cluster': dict_cluster[j], 'density':
float(M_kmean[i][j])}
            dataframes_to_concat.append(pd.DataFrame([new_row]))

conf_kmean = pd.concat(dataframes_to_concat, ignore_index=True)

print(conf_kmean)
```

	real	real_Consumption	predicted	predicted_cluster	density
0	0	35	1	B	1.0
1	0	35	2	C	24.0
2	1	25	0	A	2.0
3	1	25	1	B	2.0
4	1	25	2	C	27.0
5	2	30	2	C	32.0
6	3	40	2	C	4.0
7	4	20	0	A	11.0
8	4	20	1	B	28.0
9	4	20	2	C	18.0
10	5	15	0	A	39.0
11	5	15	1	B	8.0
12	6	45	2	C	4.0

```
sns.scatterplot(x='real_Consumption', y='predicted_cluster',
s=(conf_kmean.density)*60, data=conf_kmean)
pl.xlabel('Real Consumption')
pl.ylabel('Cluster k-mean')
pl.title('Alignement between clusters and consumption')
plt.show()
```




```
import seaborn as sns
import matplotlib.pyplot as plt

# Assuming conf_kmean contains the necessary data

# Create a heatmap with annotations
sns.heatmap(data=conf_kmean.pivot_table(index='real_Consumption',
columns='predicted_cluster', values='density'),
            cmap='viridis', annot=True, fmt=".2f")

plt.xlabel('Predicted Cluster')
plt.ylabel('Real Consumption')
plt.title('Alignment Between Clusters and Consumption Heatmap')
plt.show()
```




```
!pip install import-ipynb
import import_ipynb
import Matching_cluster
```

```
Requirement already satisfied: import-ipynb in c:\users\karthikeyan\
anaconda3\lib\site-packages (0.1.4)
Requirement already satisfied: IPython in c:\users\karthikeyan\
anaconda3\lib\site-packages (from import-ipynb) (8.15.0)
Requirement already satisfied: nbformat in c:\users\karthikeyan\
anaconda3\lib\site-packages (from import-ipynb) (5.9.2)
Requirement already satisfied: backcall in c:\users\karthikeyan\
anaconda3\lib\site-packages (from IPython->import-ipynb) (0.2.0)
Requirement already satisfied: decorator in c:\users\karthikeyan\
anaconda3\lib\site-packages (from IPython->import-ipynb) (5.1.1)
Requirement already satisfied: jedi>=0.16 in c:\users\karthikeyan\
anaconda3\lib\site-packages (from IPython->import-ipynb) (0.18.1)
Requirement already satisfied: matplotlib-inline in c:\users\
karthikeyan\anaconda3\lib\site-packages (from IPython->import-ipynb)
(0.1.6)
Requirement already satisfied: pickleshare in c:\users\karthikeyan\
anaconda3\lib\site-packages (from IPython->import-ipynb) (0.7.5)
```

```
Requirement already satisfied: prompt-toolkit!=3.0.37,<3.1.0,>=3.0.30
in c:\users\karthikeyan\anaconda3\lib\site-packages (from IPython-
>import-ipynb) (3.0.36)
Requirement already satisfied: pygments>=2.4.0 in c:\users\
karthikeyan\anaconda3\lib\site-packages (from IPython->import-ipynb)
(2.15.1)
Requirement already satisfied: stack-data in c:\users\karthikeyan\
anaconda3\lib\site-packages (from IPython->import-ipynb) (0.2.0)
Requirement already satisfied: traitlets>=5 in c:\users\karthikeyan\
anaconda3\lib\site-packages (from IPython->import-ipynb) (5.7.1)
Requirement already satisfied: colorama in c:\users\karthikeyan\
anaconda3\lib\site-packages (from IPython->import-ipynb) (0.4.6)
Requirement already satisfied: fastjsonschema in c:\users\karthikeyan\
anaconda3\lib\site-packages (from nbformat->import-ipynb) (2.16.2)
Requirement already satisfied: jsonschema>=2.6 in c:\users\
karthikeyan\anaconda3\lib\site-packages (from nbformat->import-ipynb)
(4.17.3)
Requirement already satisfied: jupyter-core in c:\users\karthikeyan\
anaconda3\lib\site-packages (from nbformat->import-ipynb) (5.3.0)
Requirement already satisfied: parso<0.9.0,>=0.8.0 in c:\users\
karthikeyan\anaconda3\lib\site-packages (from jedi>=0.16->IPython-
>import-ipynb) (0.8.3)
Requirement already satisfied: attrs>=17.4.0 in c:\users\karthikeyan\
anaconda3\lib\site-packages (from jsonschema>=2.6->nbformat->import-
ipynb) (22.1.0)
Requirement already satisfied: pyrsistent!=0.17.0,!=0.17.1,!
=0.17.2,>=0.14.0 in c:\users\karthikeyan\anaconda3\lib\site-packages
(from jsonschema>=2.6->nbformat->import-ipynb) (0.18.0)
Requirement already satisfied: wcwidth in c:\users\karthikeyan\
anaconda3\lib\site-packages (from prompt-toolkit!
=3.0.37,<3.1.0,>=3.0.30->IPython->import-ipynb) (0.2.5)
Requirement already satisfied: platformdirs>=2.5 in c:\users\
karthikeyan\anaconda3\lib\site-packages (from jupyter-core->nbformat-
>import-ipynb) (3.10.0)
Requirement already satisfied: pywin32>=300 in c:\users\karthikeyan\
anaconda3\lib\site-packages (from jupyter-core->nbformat->import-
ipynb) (305.1)
Requirement already satisfied: executing in c:\users\karthikeyan\
anaconda3\lib\site-packages (from stack-data->IPython->import-ipynb)
(0.8.3)
Requirement already satisfied: asttokens in c:\users\karthikeyan\
anaconda3\lib\site-packages (from stack-data->IPython->import-ipynb)
(2.0.5)
Requirement already satisfied: pure-eval in c:\users\karthikeyan\
anaconda3\lib\site-packages (from stack-data->IPython->import-ipynb)
(0.2.2)
Requirement already satisfied: six in c:\users\karthikeyan\anaconda3\
lib\site-packages (from asttokens->stack-data->IPython->import-ipynb)
```

```
(1.16.0)
importing Jupyter notebook from Matching_cluster.ipynb
```

```
acc,y_pred,dict_map_cluster
=Matching_cluster.remap_labels(pred['Prediction_kmean'],train['Consump
tion_encoded'])
print(dict_map_cluster)
#We store the K-means results in a dataframe
pred_1 = pd.DataFrame(y_pred)
pred_1.columns = ['Prediction_kmean_mapped']
#we merge this dataframe with df
pred= pd.concat([pred,pred_1], axis = 1)
```

```
{0: 5, 1: 4, 2: 2, 3: 0, 4: 1, 5: 3, 6: 6, 7: 7}
```

The confusion matrix shows that the K-means algorithm performs well in classifying data points into their respective fuel consumption classes, with high accuracy for most classes. However, it is important to note that performance on the training data may not necessarily translate to performance on unseen data.

```
M_kmeanmapped=confusion_matrix(train['Consumption_encoded'],pred['Pred
iction_kmean_mapped'])
print (M_kmeanmapped)
```

```
[[ 0  0 24  0  1  0  0  0]
 [ 0  0 27  0  2  2  0  0]
 [ 0  0 32  0  0  0  0  0]
 [ 0  0  4  0  0  0  0  0]
 [ 0  0 18  0 28 11  0  0]
 [ 0  0  0  0  8 39  0  0]
 [ 0  0  4  0  0  0  0  0]
 [ 0  0  0  0  0  5  0  0]]
```

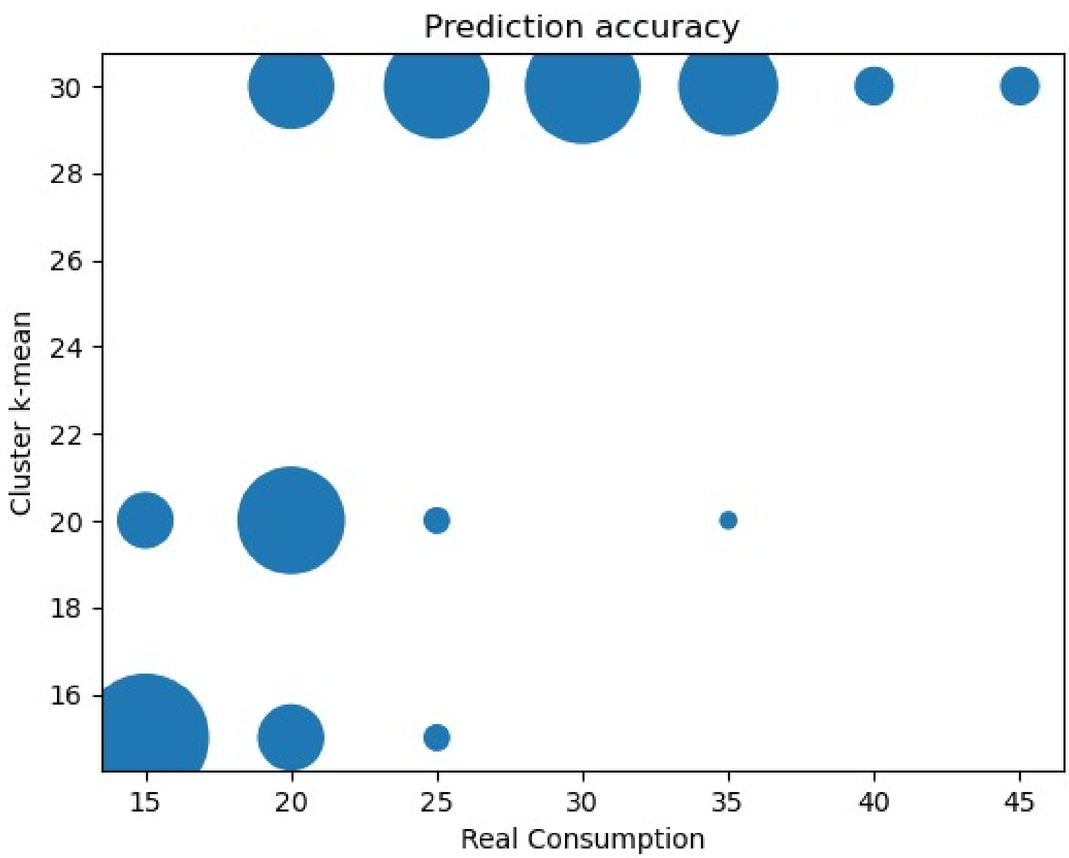
Summarizes the confusion matrix after mapping the cluster labels, visualizes the relationship between actual and predicted fuel consumption values using a scatter plot, and prints the summary of the accuracy of the K-means predictions after mapping the cluster labels. it appears that the K-means clustering algorithm has been able to effectively group the data points into clusters based on their fuel consumption. The clusters are represented by the predicted values, and the density column indicates the number of data points in each cluster

```
import pandas as pd

conf_kmeanmapped = pd.DataFrame(columns=['real', 'real_Consumption',
'predicted', 'predicted_Consumption', 'density'])
dataframes_to_concat = []

for i in range(0, 7):
    for j in range(0, 7):
```

```python
        if M_kmeanmapped[i][j] > 0:
            new_row = {'real': i, 'real_Consumption': dict_cat[i],
'predicted': j, 'predicted_Consumption': dict_cat[j], 'density':
float(M_kmeanmapped[i][j])}
            dataframes_to_concat.append(pd.DataFrame([new_row]))

conf_kmeanmapped = pd.concat(dataframes_to_concat, ignore_index=True)

print(conf_kmeanmapped)
```

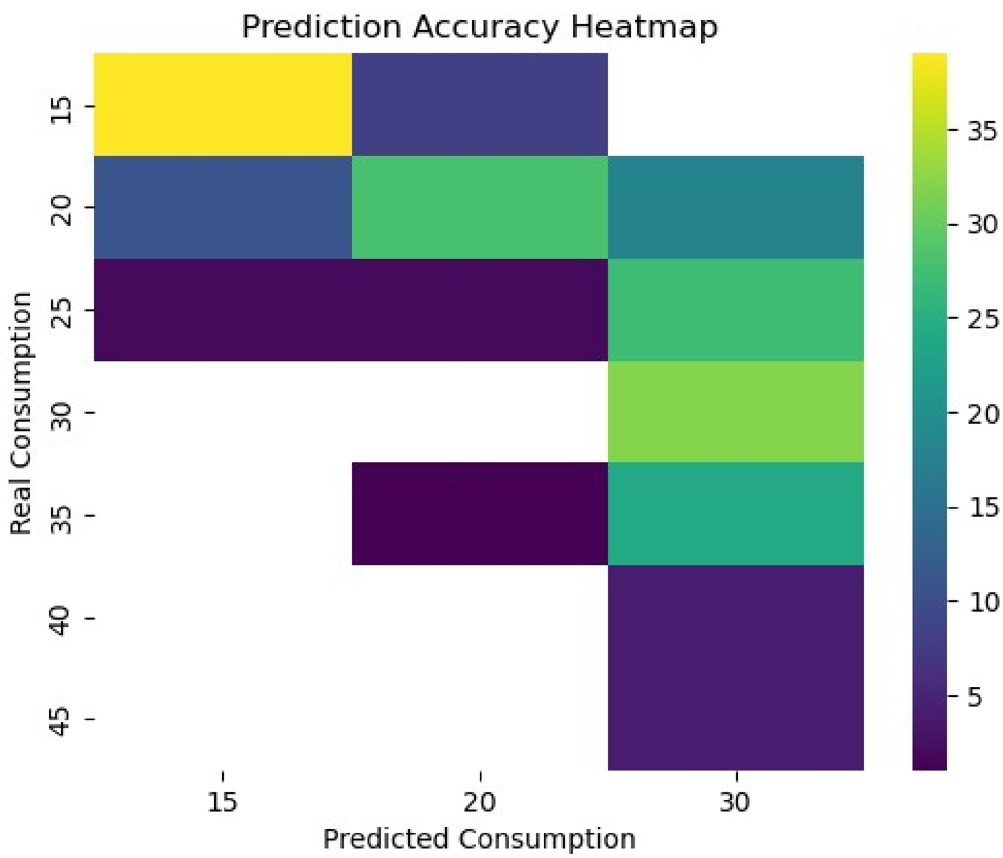
	real	real_Consumption	predicted	predicted_Consumption	density
0	0	35	2	30	24.0
1	0	35	4	20	1.0
2	1	25	2	30	27.0
3	1	25	4	20	2.0
4	1	25	5	15	2.0
5	2	30	2	30	32.0
6	3	40	2	30	4.0
7	4	20	2	30	18.0
8	4	20	4	20	28.0
9	4	20	5	15	11.0
10	5	15	4	20	8.0
11	5	15	5	15	39.0
12	6	45	2	30	4.0

```python
sns.scatterplot(x='real_Consumption', y='predicted_Consumption',
s=(conf_kmeanmapped.density)*60, data=conf_kmeanmapped)
pl.xlabel('Real Consumption')
pl.ylabel('Cluster k-mean')
pl.title('Prediction accuracy')
plt.show()
```



Performs K-means clustering on the normalized data using the kmeans.predict() method. The Norm[Input_cols] part indicates that the clustering is performed on the subset of the Norm DataFrame containing the columns specified in the Input_cols list.

```
import seaborn as sns
import matplotlib.pyplot as plt

# Assuming conf_kmeanmapped contains the necessary data

# Create a heatmap
sns.heatmap(data=conf_kmeanmapped.pivot_table(index='real_Consumption'
, columns='predicted_Consumption', values='density'),
            cmap='viridis')

plt.xlabel('Predicted Consumption')
plt.ylabel('Real Consumption')
plt.title('Prediction Accuracy Heatmap')
plt.show()
```




```
y_pred_kmean = kmeans.predict(Norm[Input_cols])
#We store the K-means results in a dataframe
pred = pd.DataFrame(y_pred_kmean)
pred.columns = ['Prediction_kmean_mapped']
mapping=pred['Prediction_kmean_mapped'].map(dict_map_cluster)
print(mapping)
Norm = pd.concat([Norm,mapping], axis = 1)
print(Norm)

0      2
1      2
2      2
3      2
4      2
      ..
337    2
338    4
339    4
340    2
341    2
Name: Prediction_kmean_mapped, Length: 342, dtype: int64
    Consumption  Cylinders  Cubic_inch  Horsepower    Weight
```

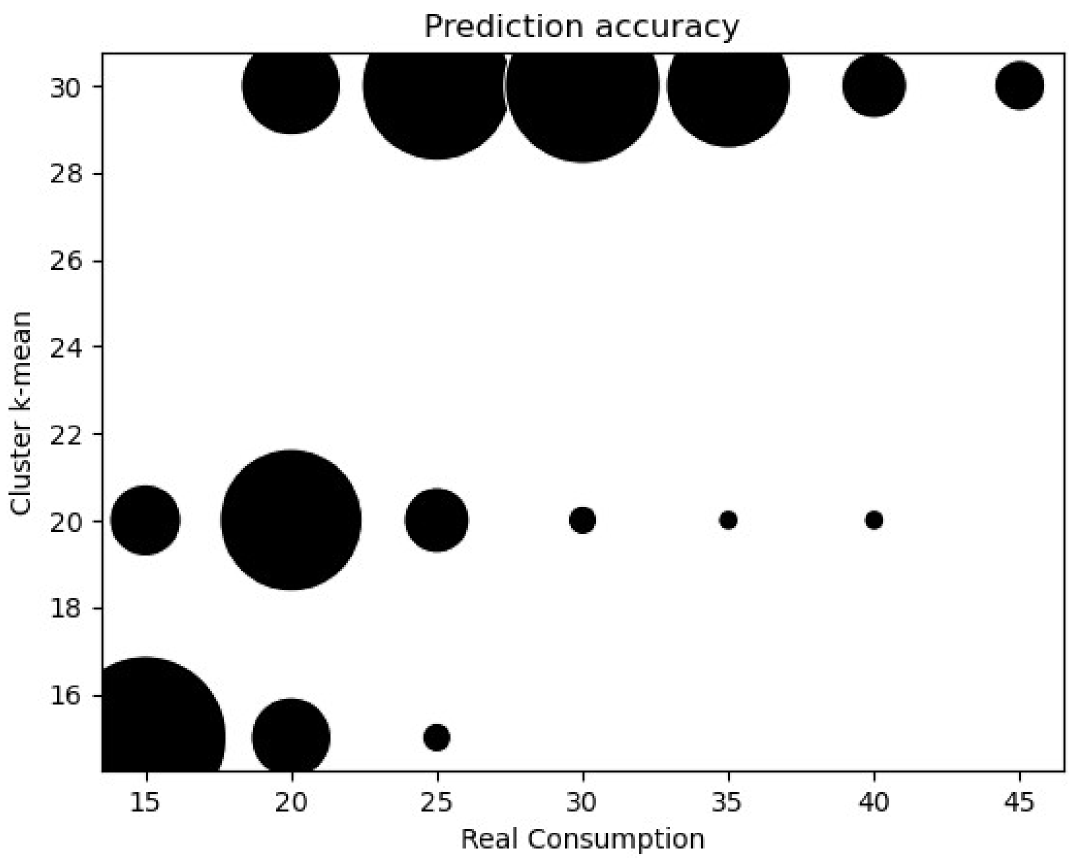
```
Acceleration  \
0             35         0.2    0.028424    0.065217  0.052569
0.619632
1             25         0.2    0.072351    0.125000  0.161290
0.582822
2             25         0.2    0.077519    0.239130  0.183990
0.429448
3             25         0.2    0.124031    0.157609  0.178315
0.337423
4             30         0.2    0.000000    0.016304  0.065114
0.674847
..           ...         ...         ...         ...       ...
...
337           30         0.2    0.028424    0.114130  0.104839
0.460123
338           20         0.6    0.341085    0.211957  0.424432
0.503067
339           20         0.6    0.341085    0.211957  0.400538
0.595092
340           25         0.2    0.103359    0.255435  0.221625
0.429448
341           30         0.2    0.074935    0.114130  0.124253
0.570552

       Brand   Car_name  Consumption_encoded      PCA0      PCA1  \
0    renault          5                    0 -0.594730  0.081303
1    renault         12                    1 -0.496658  0.057931
2       fiat  124_sport                    1 -0.417282 -0.110495
3       fiat        124                    1 -0.409473 -0.161451
4       fiat  128_sport                    2 -0.629933  0.147951
..       ...        ...                  ...       ...       ...
337     fiat       x1.9                    2 -0.526479 -0.063020
338  mercury     zephyr                    4  0.051763  0.122656
339  mercury     zephyr                    4  0.025117  0.197691
340   subaru        NaN                    1 -0.381796 -0.105147
341   subaru        NaN                    2 -0.513728  0.041974

     Prediction_GNB  Prediction_NN  Prediction_kmean_mapped
0                 3              0                        2
1                 2              2                        2
2                 2              2                        2
3                 2              2                        2
4                 2              0                        2
..              ...            ...                      ...
337               3              0                        2
338               4              4                        4
339               4              4                        4
340               2              1                        2
341               2              2                        2
```

```
[342 rows x 14 columns]

M_kmeanmapped_total=confusion_matrix(Norm['Consumption_encoded'],Norm[
'Prediction_kmean_mapped'])
print (M_kmeanmapped_total)

[[ 0  0 36  0  1  0  0  0]
 [ 0  0 52  0 10  2  0  0]
 [ 0  0 56  0  2  0  0  0]
 [ 0  0 10  0  1  0  0  0]
 [ 0  0 23  0 47 15  0  0]
 [ 0  0  0  0 12 61  0  0]
 [ 0  0  6  0  0  0  0  0]
 [ 0  0  0  0  0  8  0  0]]
```

Summarizing the confusion matrix after mapping the cluster labels back to the original fuel consumption classes and visualizes the relationship between actual and predicted fuel consumption values using a scatter plot.

```
import pandas as pd

conf_kmeanmapped_total = pd.DataFrame(columns=['real',
'real_Consumption', 'predicted', 'predicted_Consumption', 'density'])
dataframes_to_concat = []

for i in range(0, 7):
    for j in range(0, 7):
        if M_kmeanmapped_total[i][j] > 0:
            new_row = {'real': i, 'real_Consumption': dict_cat[i],
'predicted': j, 'predicted_Consumption': dict_cat[j], 'density':
float(M_kmeanmapped_total[i][j])}
            dataframes_to_concat.append(pd.DataFrame([new_row]))

conf_kmeanmapped_total = pd.concat(dataframes_to_concat,
ignore_index=True)

print(conf_kmeanmapped_total)

    real  real_Consumption  predicted  predicted_Consumption  density
0      0                35          2                     30     36.0
1      0                35          4                     20      1.0
2      1                25          2                     30     52.0
3      1                25          4                     20     10.0
4      1                25          5                     15      2.0
5      2                30          2                     30     56.0
6      2                30          4                     20      2.0
7      3                40          2                     30     10.0
8      3                40          4                     20      1.0
9      4                20          2                     30     23.0
```

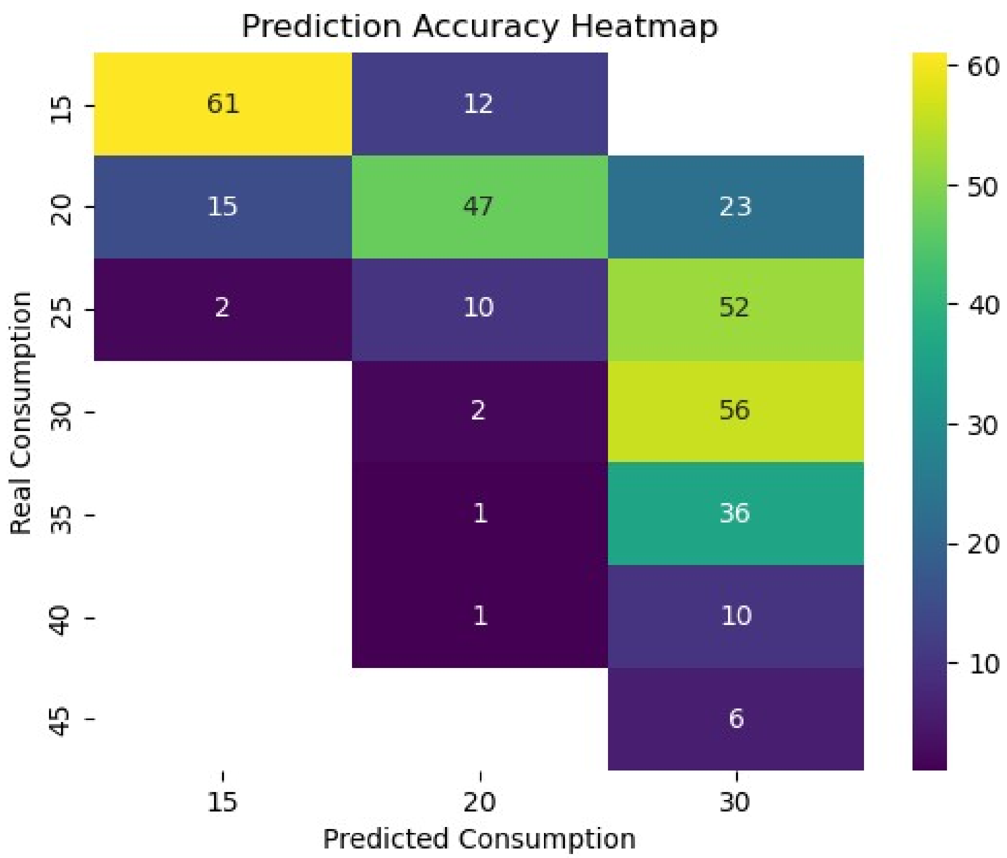
```
10      4                       20          4                           20      47.0
11      4                       20          5                           15      15.0
12      5                       15          4                           20      12.0
13      5                       15          5                           15      61.0
14      6                       45          2                           30       6.0

sns.scatterplot(x='real_Consumption', y='predicted_Consumption',
s=(conf_kmeanmapped_total.density)*60, data=conf_kmeanmapped_total,
color='k')
pl.xlabel('Real Consumption')
pl.ylabel('Cluster k-mean')
pl.title('Prediction accuracy')
plt.show()
```



Predicts the fuel consumption class for a new car with the given specifications using the K-means clustering model.

```
import seaborn as sns
import matplotlib.pyplot as plt

# Assuming conf_kmeanmapped_total contains the necessary data

# Create a heatmap
```

```
sns.heatmap(data=conf_kmeanmapped_total.pivot_table(index='real_Consum
ption', columns='predicted_Consumption', values='density'),
            annot=True, cmap='viridis')

plt.xlabel('Predicted Consumption')
plt.ylabel('Real Consumption')
plt.title('Prediction Accuracy Heatmap')
plt.show()
```




```
New_specimen = { 'Cylinders':[0.5],
 'Cubic_inch': [0.5],
 'Horsepower': [0.5],
 'Weight': [0.5],
 'Acceleration':[0.5]
 }
panda_New_specimen = pd.DataFrame(New_specimen)
D=kmeans.predict(panda_New_specimen)
print('Using kmeans, the predicted Consumption of such a car is '+
str(dict_cat[dict_map_cluster[D[0]]]))

Using kmeans, the predicted Consumption of such a car is 20
```

```python
# Import necessary libraries for Dash
import dash
from dash import dcc, html, Input, Output
import plotly.express as px
import plotly.graph_objects as go

# Initialize the Dash app
app = dash.Dash(__name__)

# Define the layout of the dashboard
app.layout = html.Div([
    html.H1("Car Consumption Dashboard"),

    # Dropdown for X-axis selection for scatter plot
    dcc.Dropdown(
        id='x-axis-dropdown-scatter',
        options=[{'label': col, 'value': col} for col in
data_panda.columns],
        value='Cubic_inch',
        multi=False,
        style={'width': '50%'}
    ),

    # Dropdown for Y-axis selection for scatter plot
    dcc.Dropdown(
        id='y-axis-dropdown-scatter',
        options=[{'label': col, 'value': col} for col in
data_panda.columns],
        value='Horsepower',
        multi=False,
        style={'width': '50%'}
    ),

    # Dropdown for X-axis selection for line chart
    dcc.Dropdown(
        id='x-axis-dropdown-line',
        options=[{'label': col, 'value': col} for col in
data_panda.columns],
        value='Cubic_inch',
        multi=False,
        style={'width': '50%'}
    ),

    # Dropdown for Y-axis selection for line chart
    dcc.Dropdown(
        id='y-axis-dropdown-line',
        options=[{'label': col, 'value': col} for col in
data_panda.columns],
```

```
        value='Horsepower',
        multi=False,
        style={'width': '50%'}
    ),

    # Scatter plot
    dcc.Graph(id='scatter-plot'),

    # Line chart
    dcc.Graph(id='line-chart'),

    # Pie chart
    dcc.Graph(id='pie-chart')
])

# Callback to update scatter plot based on dropdown selection
@app.callback(
    Output('scatter-plot', 'figure'),
    [Input('x-axis-dropdown-scatter', 'value'),
     Input('y-axis-dropdown-scatter', 'value')]
)
def update_scatter_plot(x_column, y_column):
    scatter_fig = px.scatter(
        data_panda, x=x_column, y=y_column,
        color='Consumption', title=f'{x_column} vs. {y_column}',
        labels={x_column: x_column, y_column: y_column, 'Consumption':
'Consumption'},
        template='plotly_dark'
    )
    return scatter_fig

# Callback to update line chart based on dropdown selection
@app.callback(
    Output('line-chart', 'figure'),
    [Input('x-axis-dropdown-line', 'value'),
     Input('y-axis-dropdown-line', 'value')]
)
def update_line_chart(x_column, y_column):
    line_fig = px.line(
        data_panda, x=x_column, y=y_column, title=f'{y_column} vs.
{x_column}',
        labels={x_column: x_column, y_column: y_column},
        template='plotly_dark'
    )
    return line_fig

# Callback to update pie chart based on dropdown selection
@app.callback(
    Output('pie-chart', 'figure'),
    [Input('x-axis-dropdown-scatter', 'value')]
```

```
)
def update_pie_chart(x_column):
    # Calculate consumption distribution
    consumption_counts = data_panda['Consumption'].value_counts()
    consumption_percentage = (consumption_counts / len(data_panda)) *
100

    # Create a list to store hover text strings
    hover_texts = []
    for consumption, count, percentage in
zip(consumption_counts.index, consumption_counts,
consumption_percentage):
        # Get the car names for the current consumption category
        car_names = data_panda[data_panda['Consumption'] ==
consumption]['Brand'].tolist()
        # Create hover text with car names
        hover_text = f"{percentage:.1f}% cars ({count})\nCar Names:\
n{', '.join(car_names)}"
        hover_texts.append(hover_text)

    # Create pie chart using Plotly
    pie_fig = go.Figure(data=[go.Pie(labels=consumption_counts.index,
                                      values=consumption_counts,
                                      textinfo='label+percent',
                                      hole=0.3,
                                      hoverinfo='text',
                                      text=hover_texts)])
    pie_fig.update_layout(title='Consumption Distribution')
    return pie_fig

# Run the app
app.run_server(mode='external', port=8067)
```

```
<IPython.lib.display.IFrame at 0x1fad4b2a110>
```