HOUNTON_JeanPhilippes_Learning_11

January 18, 2021

1 Introduction et motivations

Our study aims to explore the wealth of a number of indicators in order to find out a model to predict the price range of mobile phone. To do this, a first step will be data processing. Indeed we will proceed to univariate and bivariate studies of our variables, in order to better understand the data we have. In a second step we will propose different models to fit the price range and finally, we will be able to predict the price range based on the available explanatory features.

To do this, the 3 main models we will use are: * *Logistic Regression* * *kNN* * CART

2 Initial data processing

Before starting our project, it is necessary to understand the type of each features we have and the information it provides us. Our database contains 2.000 observations and 21 variables. Among these 21 variables, we have the price range that we are trying to explain, which is indeed a numerical type variable. The 20 remaining variables are therefore the potential explanatory variables of our models.

Our first work consists in differentiating the quantitative discrete variables from the quantitative continuous variables because they are not treated in the same way. Features blue, fc, dual_sim, four_g, n_cores, three_g, touch screen and wifi are quantitative discrete. The others are all continuous.

Load Package

```
[145]: import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn import tree
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split
#from sklearn.model_selection import cross_validate
from pandas.plotting import scatter_matrix
from sklearn.metrics import classification_report, confusion_matrix,␣
 ↪accuracy_score
#plt.rcParams['figure.figsize'] = (15., 15. / 16 * 9)
```

```
%matplotlib inline
```

Load Data

[146]:
```
df = pd.read_csv('train.csv')
X = df.drop('price_range',axis=1)
y = df['price_range']
data_to_predict = pd.read_csv('test.csv')
```

[147]:
```
print(df.shape)
df.head()
```

```
(2000, 21)
```

[147]:
```
   battery_power  blue  clock_speed  dual_sim  fc  four_g  int_memory  m_dep  \
0            842     0          2.2         0   1       0           7    0.6
1           1021     1          0.5         1   0       1          53    0.7
2            563     1          0.5         1   2       1          41    0.9
3            615     1          2.5         0   0       0          10    0.8
4           1821     1          1.2         0  13       1          44    0.6

   mobile_wt  n_cores  ...  px_height  px_width   ram  sc_h  sc_w  talk_time  \
0        188        2  ...         20       756  2549     9     7         19
1        136        3  ...        905      1988  2631    17     3          7
2        145        5  ...       1263      1716  2603    11     2          9
3        131        6  ...       1216      1786  2769    16     8         11
4        141        2  ...       1208      1212  1411     8     2         15

   three_g  touch_screen  wifi  price_range
0        0             0     1            1
1        1             1     0            2
2        1             1     0            2
3        1             0     0            2
4        1             1     0            1

[5 rows x 21 columns]
```

[148]:
```
df.columns
```

[148]:
```
Index(['battery_power', 'blue', 'clock_speed', 'dual_sim', 'fc', 'four_g',
       'int_memory', 'm_dep', 'mobile_wt', 'n_cores', 'pc', 'px_height',
       'px_width', 'ram', 'sc_h', 'sc_w', 'talk_time', 'three_g',
       'touch_screen', 'wifi', 'price_range'],
      dtype='object')
```

Here is the attributes of our dataset: * **id**: ID * **battery_power**: Total energy a battery can store in one time * **blue**: Has bluetooth or not * **clock_speed**: Speed at which microprocessor executes instructions * **dual_sim**: Has dual sim support or not * **fc**: Front Camera mega pixels * **four_g**: Has

4G or not * **int_memory**: Internal Memory * **m_dep**: Mobile Depth * **mobile_wt**: Weight of mobile phone * **n_cores**: Number of cores of processor * **pc**: Primary Camera mega pixels * **px_height**: Pixel Resolution Height * **px_width**: Pixel Resolution Width * **ram**: Random Access Memory * **sc_h**: Screen Height of mobile * **sc_w**: Screen Width of mobile * **talk_time**: Longest time that a single battery charge will last * **three_g**: Has 3G or not * **touch_screen**: Has touch screen or not * **wifi**: Has wifi or not * **price_range**: This is the target variable with value of 0 (low), 1 (medium), 2 (high) and 3 (very high cost)

[149]:
```
df.describe()
```

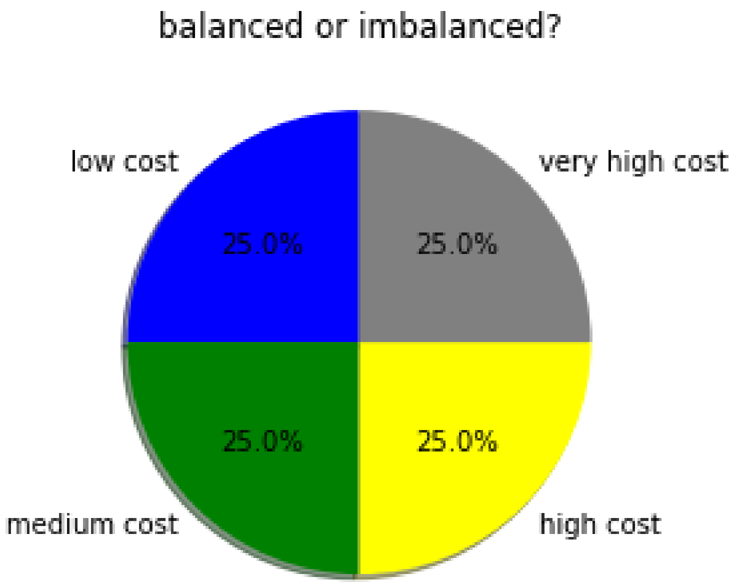
[149]:
```
       battery_power       blue  clock_speed     dual_sim           fc  \
count    2000.000000  2000.0000  2000.000000  2000.000000  2000.000000
mean     1238.518500     0.4950     1.522250     0.509500     4.309500
std       439.418206     0.5001     0.816004     0.500035     4.341444
min       501.000000     0.0000     0.500000     0.000000     0.000000
25%       851.750000     0.0000     0.700000     0.000000     1.000000
50%      1226.000000     0.0000     1.500000     1.000000     3.000000
75%      1615.250000     1.0000     2.200000     1.000000     7.000000
max      1998.000000     1.0000     3.000000     1.000000    19.000000

            four_g   int_memory        m_dep    mobile_wt      n_cores  ...  \
count  2000.000000  2000.000000  2000.000000  2000.000000  2000.000000  ...
mean      0.521500    32.046500     0.501750   140.249000     4.520500  ...
std       0.499662    18.145715     0.288416    35.399655     2.287837  ...
min       0.000000     2.000000     0.100000    80.000000     1.000000  ...
25%       0.000000    16.000000     0.200000   109.000000     3.000000  ...
50%       1.000000    32.000000     0.500000   141.000000     4.000000  ...
75%       1.000000    48.000000     0.800000   170.000000     7.000000  ...
max       1.000000    64.000000     1.000000   200.000000     8.000000  ...

         px_height     px_width          ram         sc_h         sc_w  \
count  2000.000000  2000.000000  2000.000000  2000.000000  2000.000000
mean    645.108000  1251.515500  2124.213000    12.306500     5.767000
std     443.780811   432.199447  1084.732044     4.213245     4.356398
min       0.000000   500.000000   256.000000     5.000000     0.000000
25%     282.750000   874.750000  1207.500000     9.000000     2.000000
50%     564.000000  1247.000000  2146.500000    12.000000     5.000000
75%     947.250000  1633.000000  3064.500000    16.000000     9.000000
max    1960.000000  1998.000000  3998.000000    19.000000    18.000000

         talk_time      three_g  touch_screen         wifi  price_range
count  2000.000000  2000.000000   2000.000000  2000.000000  2000.000000
mean     11.011000     0.761500      0.503000     0.507000     1.500000
std       5.463955     0.426273      0.500116     0.500076     1.118314
min       2.000000     0.000000      0.000000     0.000000     0.000000
25%       6.000000     1.000000      0.000000     0.000000     0.750000
50%      11.000000     1.000000      1.000000     1.000000     1.500000
```

```
75%       16.000000     1.000000     1.000000     1.000000     2.250000
max       20.000000     1.000000     1.000000     1.000000     3.000000

[8 rows x 21 columns]
```

3 Features study

In this section, we will study the link between explanatory variables and the price range to know their impact on our models. We know that a strong correlation (positive or negative) between two variables can explain a link between them but this is not necessary true.

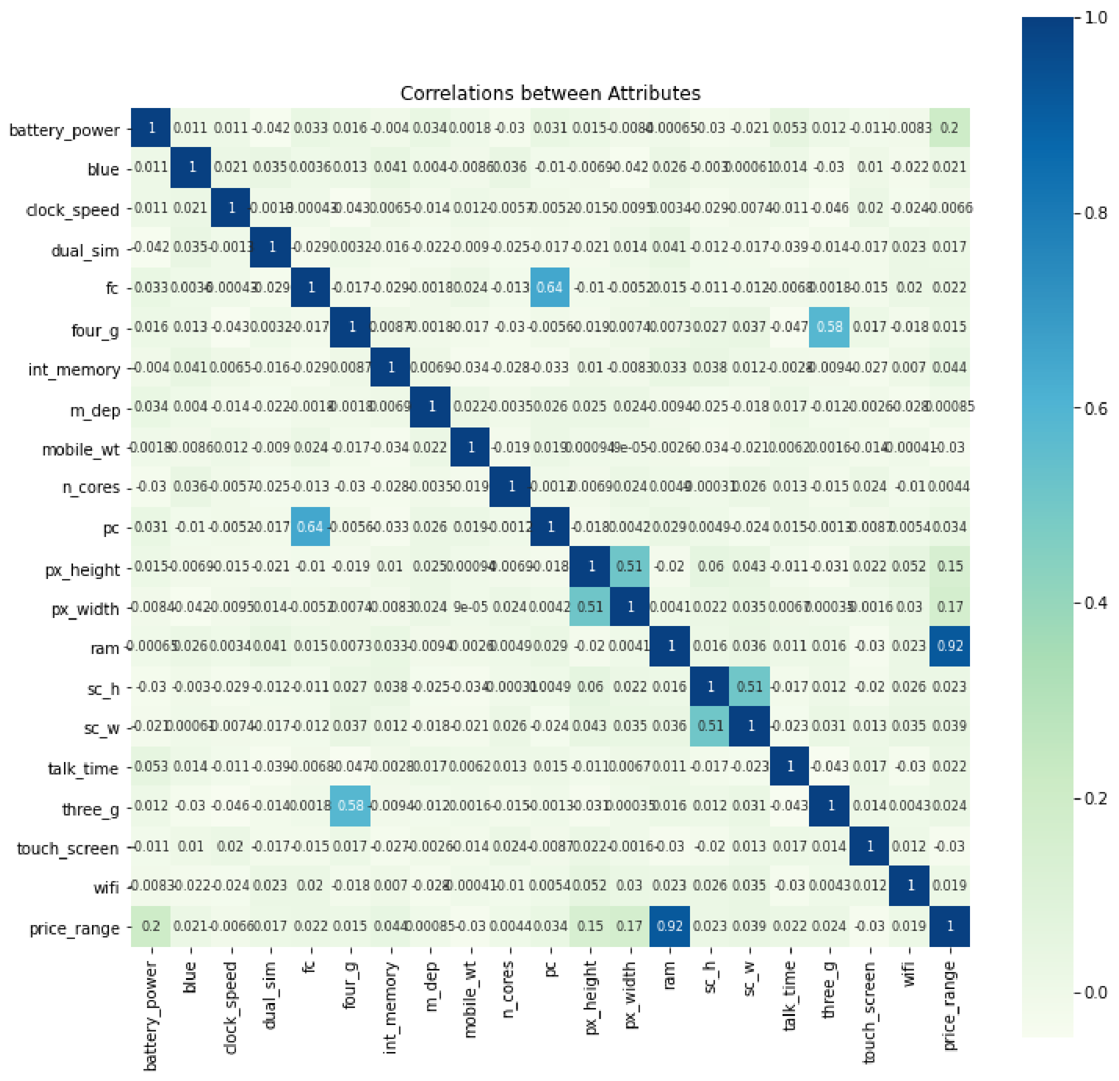
[150]:
```
labels = ["low cost", "medium cost", "high cost", "very high cost"]
values = df['price_range'].value_counts().values
colors = ['blue','green','yellow', 'grey']
fig1, ax1 = plt.subplots()
ax1.pie(values, labels=labels, colors=colors, autopct='%1.1f%%', shadow=True,␣
 ↪startangle=90)
ax1.set_title('balanced or imbalanced?')
plt.show()
```



Since our dataset is balanced our work may be more precise.

To verify the existence of a link between our variables and the price range, we realise a plot of them in function of the price range.

```
[151]: fig = plt.subplots (figsize = (12, 12))
       sns.heatmap(df.corr (), square = True, cbar = True, annot = True, cmap="GnBu",␣
        ↪annot_kws = {'size': 8})
       plt.title('Correlations between Attributes')
       plt.show()
```



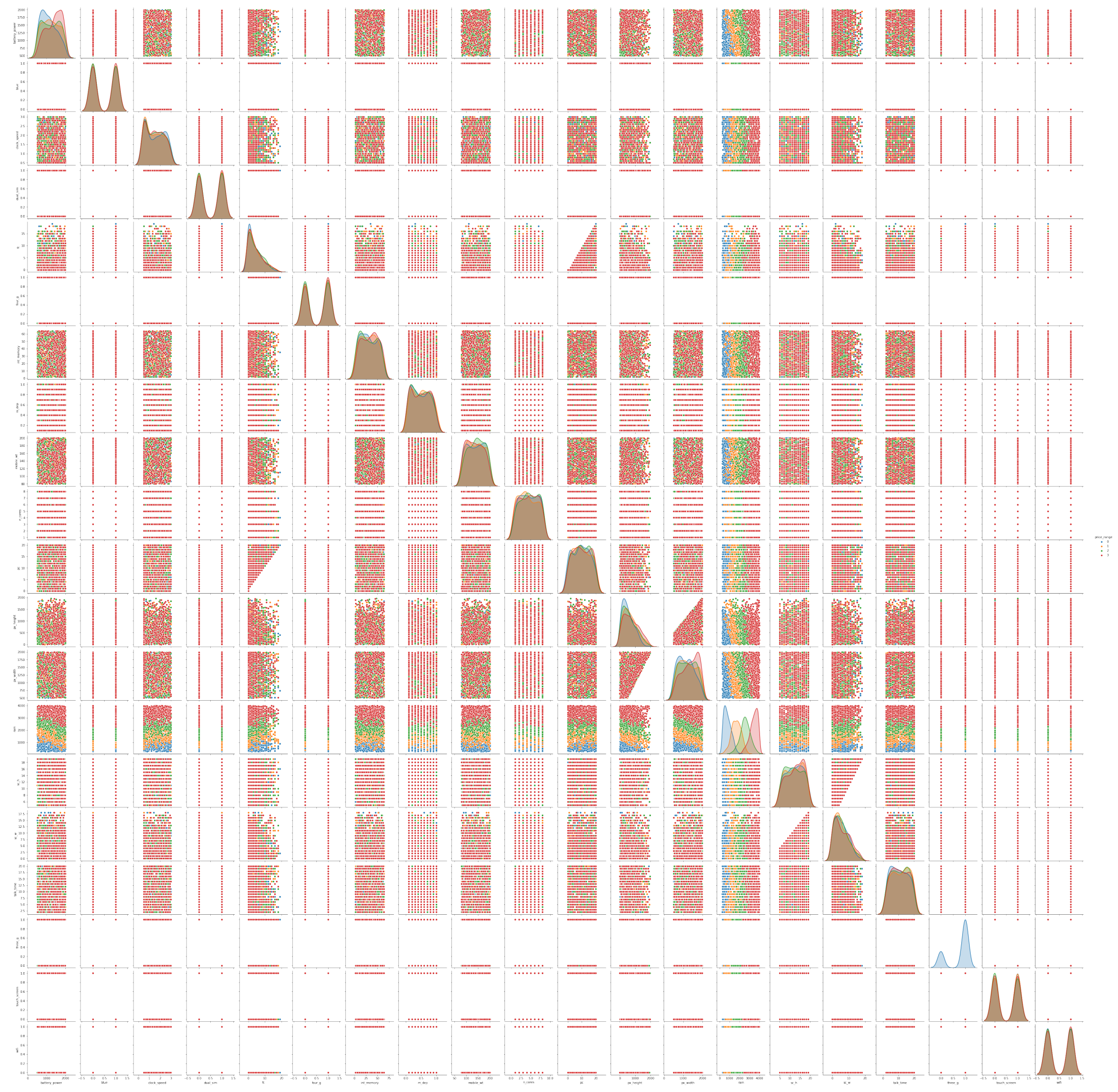
We see from the heatmap : - ram is the most influential variable - most of the variables have very little correlation to price range - primary camera and front Camera mega pixels have correlation (in fact it make sense because both of them reflect technology level of resolution of the related phone model) but they do not effect prige range. - having 3G and 4G is correlated - there is no highly correlated inputs in our dataset, so there is no multicollinearity problem.

In order to havve a global view of our data, we decide to have a look at some pairwise plot of our dataset.

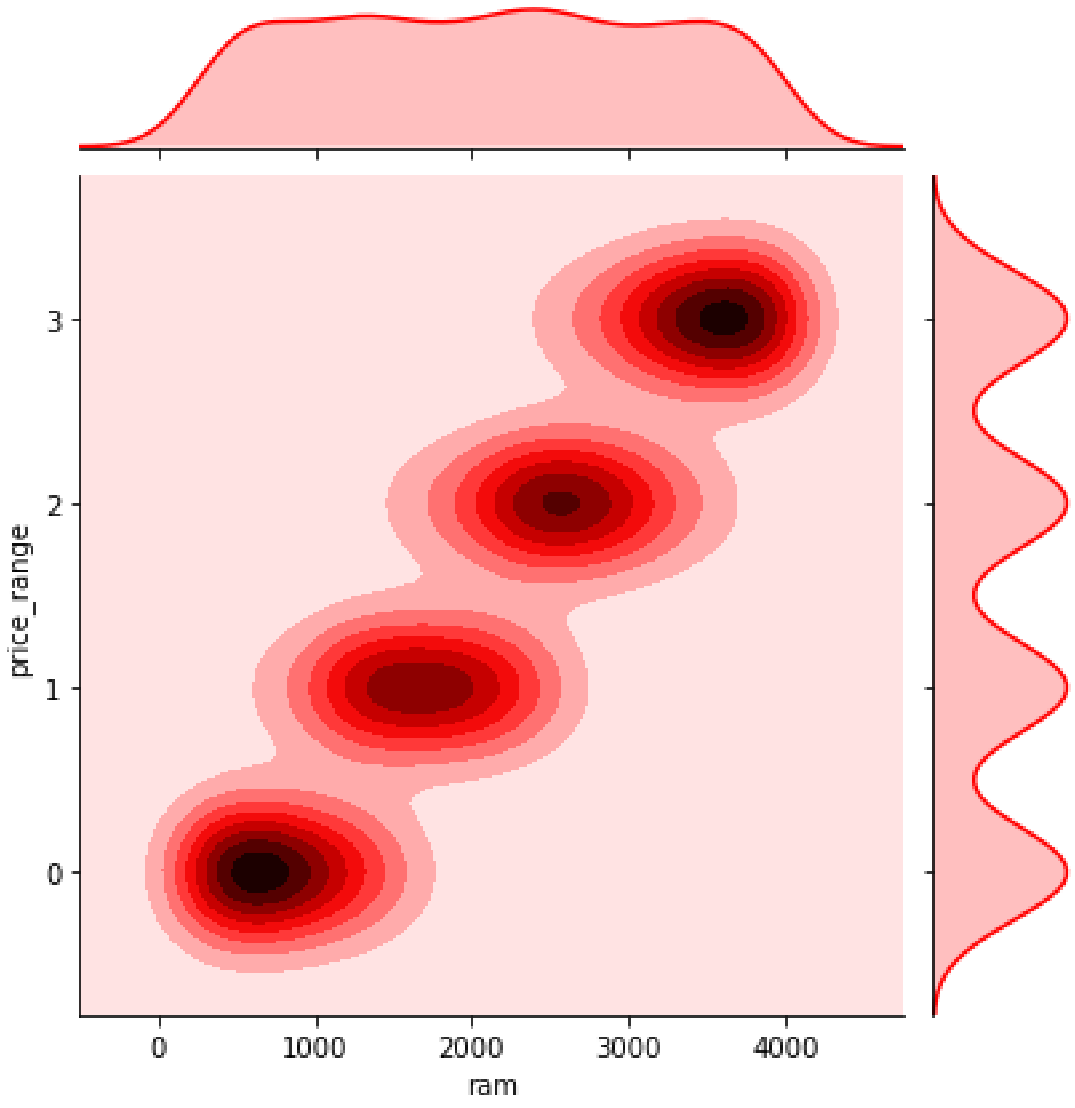
```
[152]: sns.pairplot(df,hue='price_range')
```

```
/Users/Jean-Philippes/opt/anaconda3/lib/python3.8/site-
packages/seaborn/distributions.py:369: UserWarning: Default bandwidth for data
is 0; skipping density estimation.
  warnings.warn(msg, UserWarning)
/Users/Jean-Philippes/opt/anaconda3/lib/python3.8/site-
packages/seaborn/distributions.py:369: UserWarning: Default bandwidth for data
is 0; skipping density estimation.
  warnings.warn(msg, UserWarning)
/Users/Jean-Philippes/opt/anaconda3/lib/python3.8/site-
packages/seaborn/distributions.py:369: UserWarning: Default bandwidth for data
is 0; skipping density estimation.
  warnings.warn(msg, UserWarning)
```

```
[152]: <seaborn.axisgrid.PairGrid at 0x7fab31b59790>
```



```
[153]: sns.jointplot(x='ram',y='price_range',data=df,color='red',kind='kde')

[153]: <seaborn.axisgrid.JointGrid at 0x7fab23022a00>
```



Since most of the variables are not correlated to price range, we decide to keep all of them.

4 Models

From this database, we create a training set of size n_{train} corresponding to 70% of the initial set and a test set of size n_{test} that contains the remaining 30%.

We will use a confusion matrix, which is a summary of the results of predictions on a classification problem. Correct and incorrect predictions will be highlighted and broken down by class (true positives, true negatives, false positives or false negatives). The results are thus compared with the actual values.

Our accuracy function R (proportion of variables of interest good predicted by the estimator) will be :

$$R(h) = \frac{1}{n_{test}} \sum_{i=1}^{n_{test}} 1_{Y_i = h(\hat{X}_i)}$$

and our error funtion will simply be the Mean Square Error :

$$MSE(h) = \frac{1}{n_{test}} \sum_{i=1}^{n_{test}} (Y_i - h(\hat{X}_i))^2$$

where h is our prediction function depending on X.

```
[154]: #We split out dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3,␣
 ↪random_state = 101, stratify = y)
```

4.1 Logistic Regression

Our first method, we will study the case of logistic regression. This method makes use of what's called the logistic function to calculate the odds that a given data point belongs to a given class.

Indeed, our objective is to model $f(\hat{X}) = P(Y = k/X)$ as a function linearly dependent on the observations. That is, there exists $(a_0, a_1, \ldots, a_d)$ such that :

$$f(x) = g(a_0 + a_1 x_1 + ... + a_d x_d)$$

where

$$g(t) = \frac{e^t}{1 + e^t}$$

We fit the Logistic model on our training set.

```
[155]: lr = LogisticRegression(multi_class = 'multinomial', solver = 'sag',  max_iter =␣
 ↪10000)
lr.fit(X_train,y_train)
```

```
[155]: LogisticRegression(max_iter=10000, multi_class='multinomial', solver='sag')
```

```
[156]: y_pred_lr = lr.predict(X_test)
```

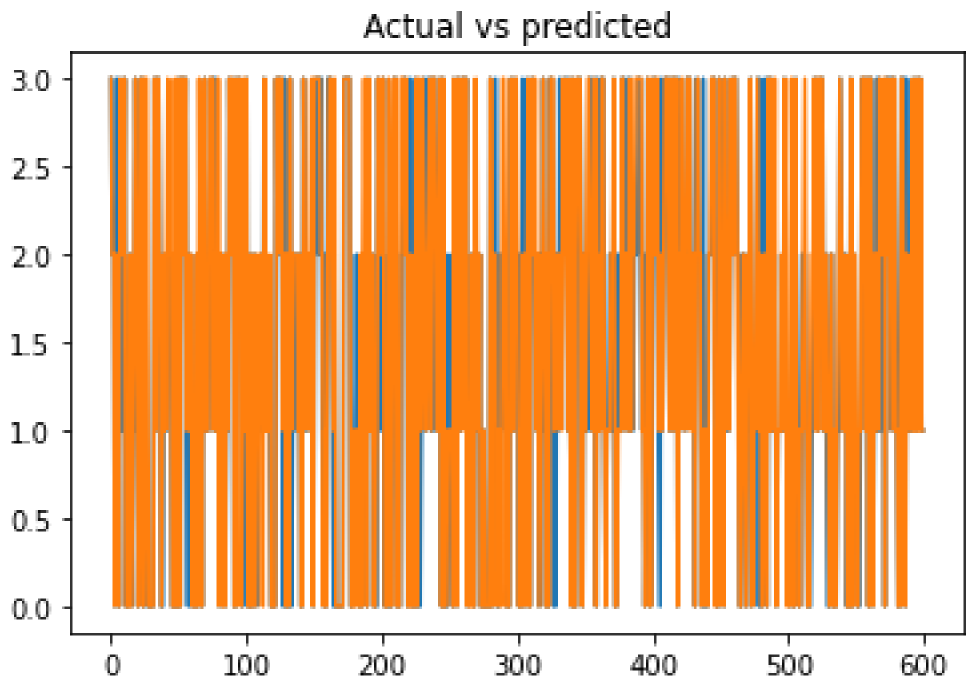
After running the algorithm we compare our predicted values to the actual values and have this confusion matrix :

```
[157]: confusion_matrix_lr = confusion_matrix(y_test, y_pred_lr)
       confusion_matrix_lr
```

```
[157]: array([[129,  21,   0,   0],
              [ 12,  95,  39,   4],
              [  1,  24,  85,  40],
              [  0,   1,  25, 124]])
```

```
[158]: t=range(len(y_test))
       plt.plot(t, y_test) #<- Actual
       plt.plot(t,y_pred_lr)# <- predicted
       plt.title("Actual vs predicted")
```

```
[158]: Text(0.5, 1.0, 'Actual vs predicted')
```



This graph show us the **Actual vs predicted** values. In fact the have a lot of difference between blue and orange values.

```
[159]: acc_lr = accuracy_score(y_test, y_pred_lr)
       MSE_lr = mean_squared_error(y_test,y_pred_lr)
       print("Accuracy of Logistic Regression is : ", acc_lr)
       print("Mean Squared Error of Logistic Regression is : ", MSE_lr)
```

```
Accuracy of Logistic Regression is :  0.7216666666666667
Mean Squared Error of Logistic Regression is :  0.30833333333333335
```

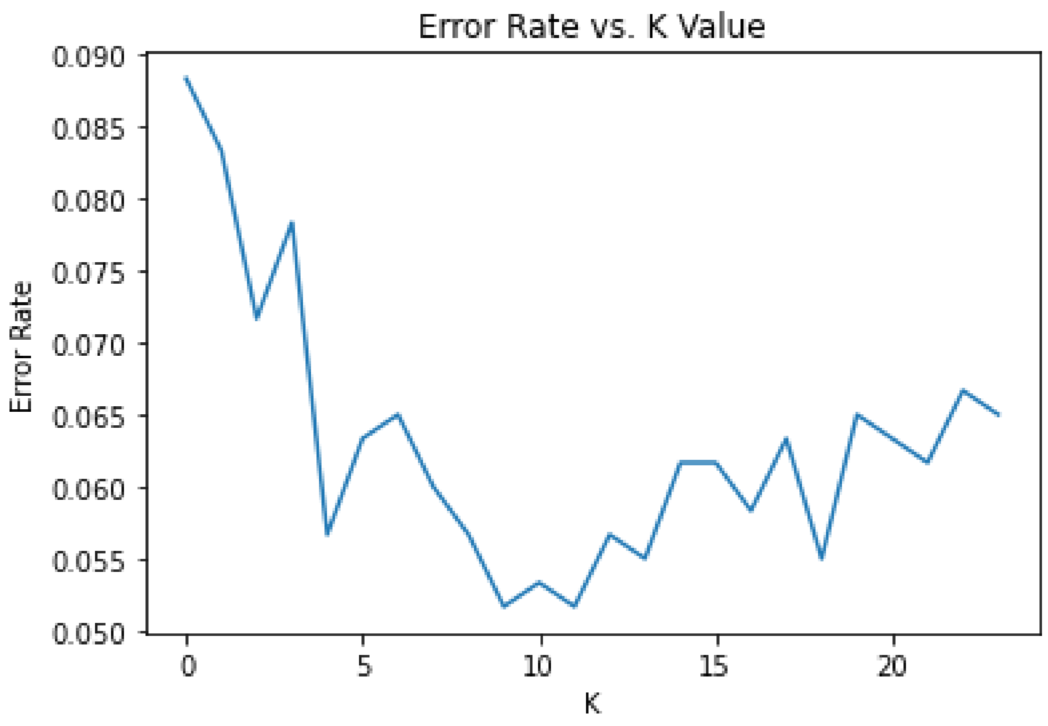
The **Logistic regression** accuracy rate is about **72%** and the mean square error is about **31%**. Since we want to maximise the accuracy and minimize the MSE, our results means that our model is globally not bad.

4.2 kNN

The second method tested is k-nearest neighbors. First, we will determine the number of neighbors *k* which will minimize the MSE. Of course, the maximum number of neighbors must not exceed the cardinality of our test sample. For each value of k, we use our training set to create the model and then calculate the error on the test set.

[160]:
```
res = []
for i in range(1,25) :
    neigh = KNeighborsClassifier(n_neighbors=i)
    neigh.fit(X_train,y_train)
    y_pred_knn = neigh.predict(X_test)
    res.append(mean_squared_error(y_test,y_pred_knn))
plt.plot(res)
plt.title('Error Rate vs. K Value')
plt.xlabel('K')
plt.ylabel('Error Rate')
```

[160]:
```
Text(0, 0.5, 'Error Rate')
```



[161]:
```
print(res.index(min(res)))
print(res[res.index(min(res))])
```

```
9
0.051666666666666666
```

From the graph we see that the error is minimal for **kopt = 9**. We get a very low error rate of around **5.17%**.

Now we can fit our knn model knoming the optimal value of k.

[162]:
```
neigh = KNeighborsClassifier(n_neighbors=9)
neigh.fit(X_train,y_train)
```

[162]:
```
KNeighborsClassifier(n_neighbors=9)
```

[163]:
```
y_pred_knn = neigh.predict(X_test)
```

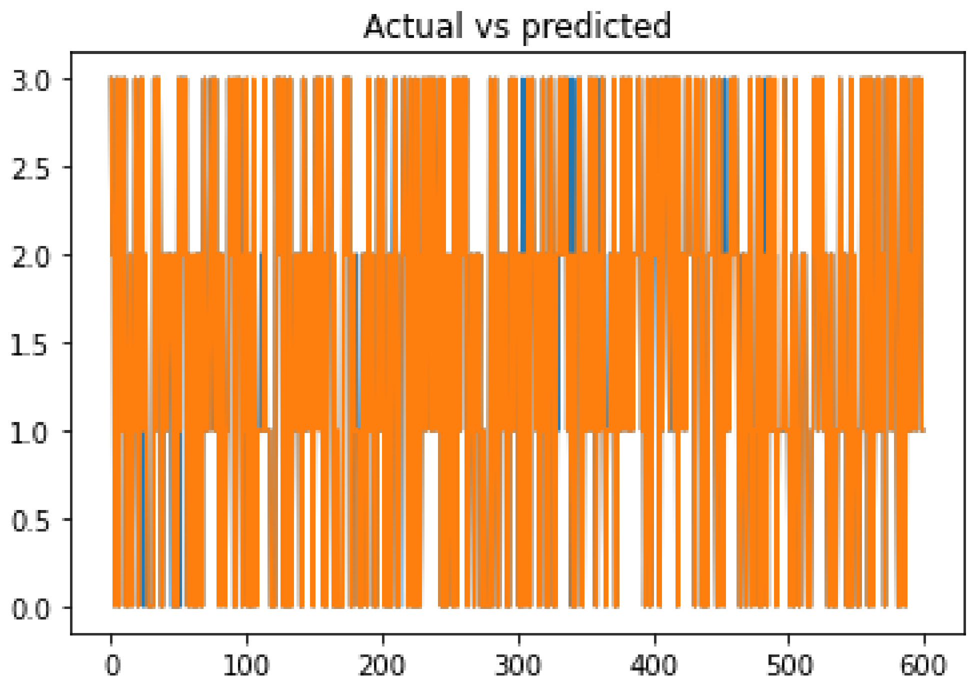
After running the algorithm we compare our predicted values to the actual values and have this confusion matrix :

[164]:
```
print(confusion_matrix(y_test, y_pred_knn))
```

```
[[145   5   0   0]
 [  1 143   6   0]
 [  0   9 138   3]
 [  0   0  10 140]]
```

[165]:
```
t=range(len(y_test))
plt.plot(t, y_test) #<- Actual
plt.plot(t,y_pred_knn)# <- predicted
plt.title("Actual vs predicted")
```

[165]:
```
Text(0.5, 1.0, 'Actual vs predicted')
```



This graph show us the **Actual vs predicted** values. We have much less blue in this graph compared to the one of the Logistic regression.

```
[166]: acc_knn = accuracy_score(y_test, y_pred_knn)
MSE_knn = mean_squared_error(y_test,y_pred_knn)
print("Accuracy of Logistic Regression is : ", acc_knn)
print("Mean Squared Error of Logistic Regression is : ", MSE_knn)
```

```
Accuracy of Logistic Regression is :  0.9433333333333334
Mean Squared Error of Logistic Regression is :  0.056666666666666664
```

The **kNN** accuracy rate is about **94.33%** and the mean square error is about **5.67%%**. Since we want to maximise the accuracy and minimize the MSE, our results means that our model is very good.

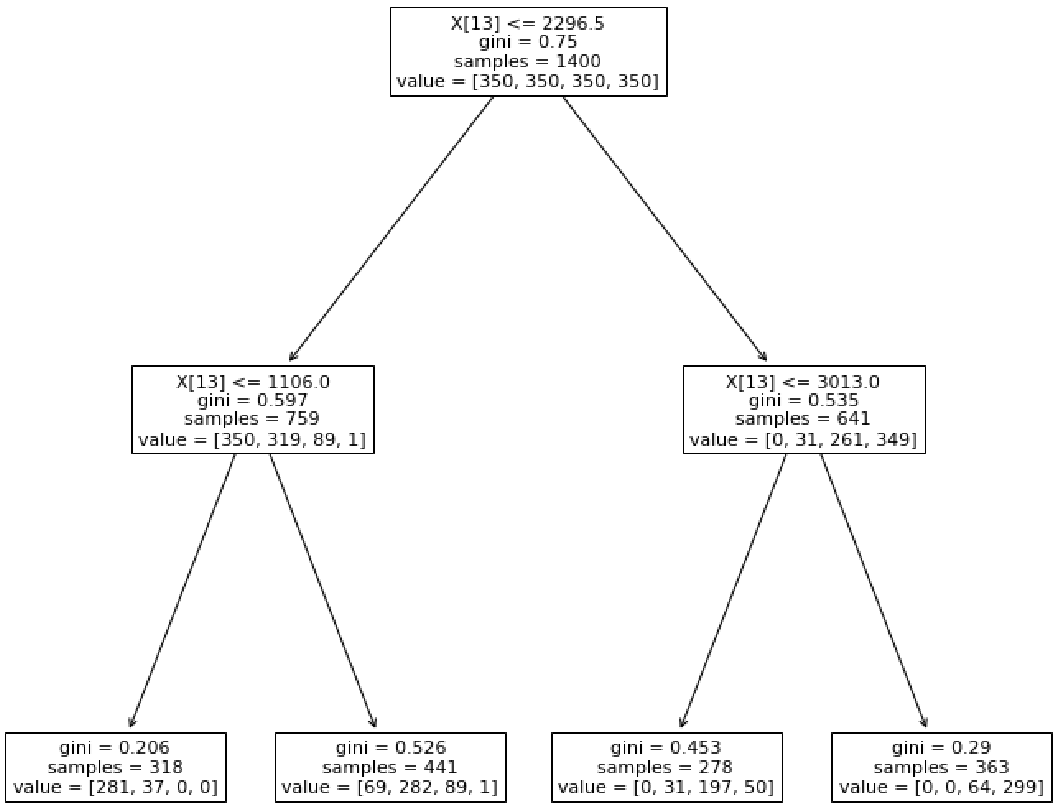
4.3 CART

Here, we will be using a simple algorithm known as a decision tree. Decision trees are rule-based classifiers that take in features and follow a 'tree structure' of binary decisions to ultimately classify a data point into one of two or more categories. In addition to being easy to both use and interpret, decision trees allow us to visualize the 'logic flowchart' that the model generates from the training data.

```
[167]: cart = tree.DecisionTreeClassifier(max_leaf_nodes=4)
arbre = cart.fit(X_train,y_train)
```

Here is the decision tree of our model :

[168]:
```
fig = plt.subplots (figsize = (12, 12))
tree.plot_tree(arbre)
plt.show()
```



[169]:
```
y_pred_cart = cart.predict(X_test)
```

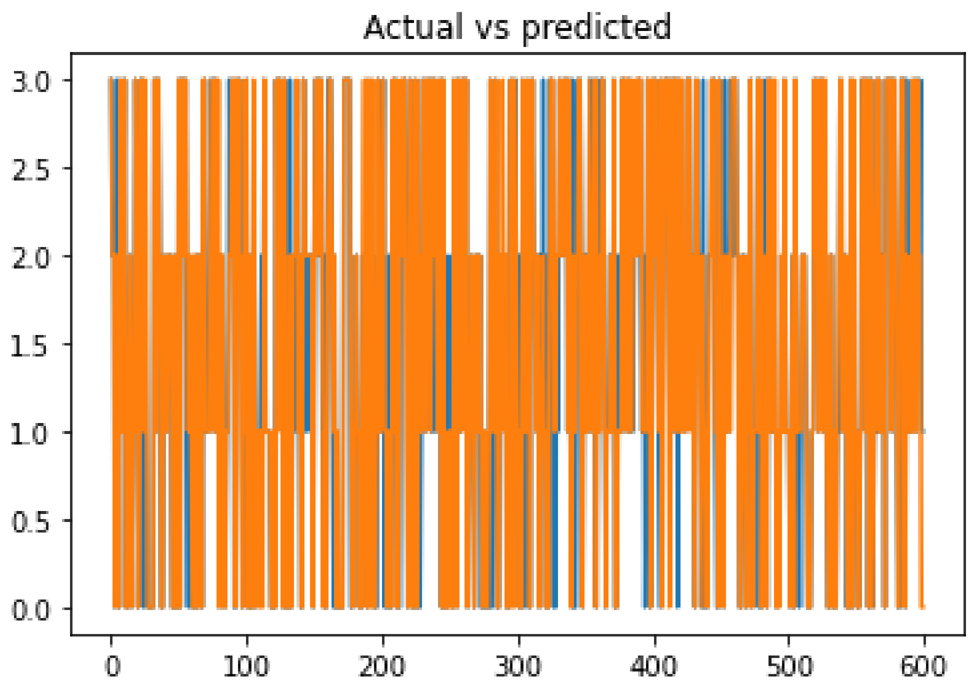
After running the algorithm we compare our predicted values to the actual values and have this confusion matrix :

[170]:
```
print(confusion_matrix(y_test,y_pred_cart))
```

```
[[121  29   0   0]
 [ 12 121  17   0]
 [  0  37  89  24]
 [  0   1  16 133]]
```

[171]:
```
t=range(len(y_test))
plt.plot(t, y_test) #<- Actual
plt.plot(t,y_pred_cart)# <- predicted
plt.title("Actual vs predicted")
```

[171]:
```
Text(0.5, 1.0, 'Actual vs predicted')
```



This graph show us the **Actual vs predicted** values. We have much less blue in this graph compared to the one of the Logistic regression but much more than the one of kNN.

[172]:
```
acc_cart = accuracy_score(y_test, y_pred_cart)
MSE_cart = mean_squared_error(y_test,y_pred_cart)
print("Accuracy of Logistic Regression is : ", acc_cart)
print("Mean Squared Error of Logistic Regression is : ", MSE_cart)
```

```
Accuracy of Logistic Regression is :  0.7733333333333333
Mean Squared Error of Logistic Regression is :  0.23166666666666666
```

The **CART** accuracy rate is about **77.33%** and the mean square error is about **23.17%%**. Since we want to maximise the accuracy and minimize the MSE, our results means that our model is globally good. The results are almost the same for the logistic regression.

5 Models comparision

Although our knn performance is good, it's a bad idea to immediately assume that it's therefore the perfect tool for this job – there's always the possibility of other models that will perform even better! It's always a worthwhile idea to at least test a few other algorithms and find the one that's best for our data.

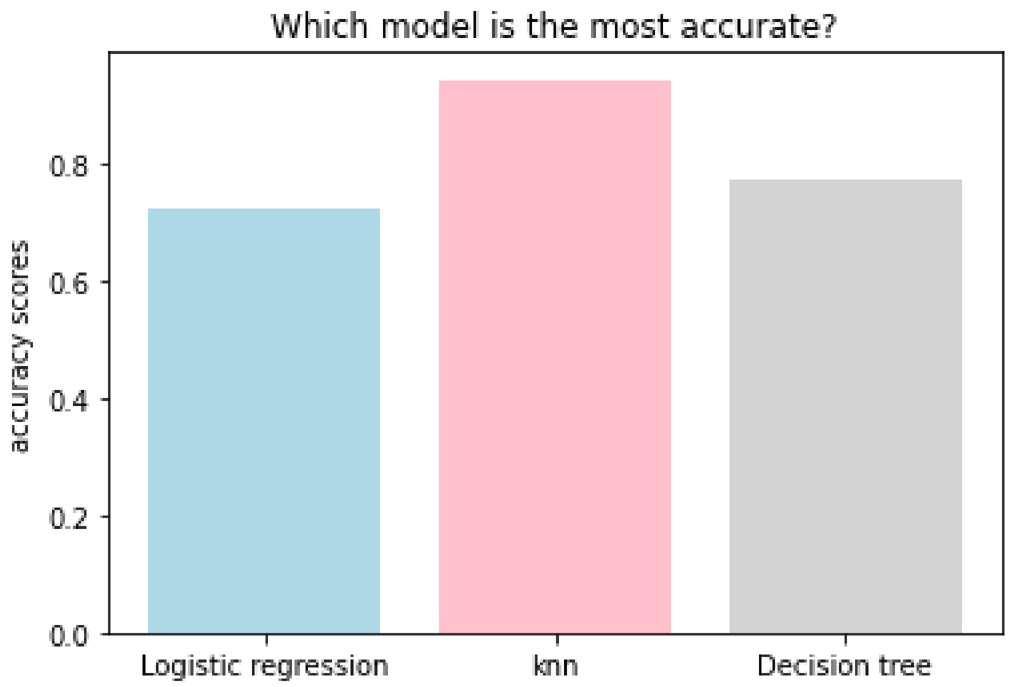
Once we have 3 differents models, we can compare them on a few performance metrics, such as false positive and false negative rate (or how many points are inaccurately classified).

[173]:
```
# Create the classification report for our models

class_rep_lr  = classification_report(y_test,y_pred_lr)
class_rep_knn = classification_report(y_test,y_pred_knn)
class_rep_cart = classification_report(y_test,y_pred_cart)


print("Logistic Regression : \n", class_rep_lr)
print("kNN Classification : \n", class_rep_knn)
print("Decision Tree: \n", class_rep_cart)
```

```
Logistic Regression :
               precision    recall  f1-score   support

           0       0.91      0.86      0.88       150
           1       0.67      0.63      0.65       150
           2       0.57      0.57      0.57       150
           3       0.74      0.83      0.78       150

    accuracy                           0.72       600
   macro avg       0.72      0.72      0.72       600
weighted avg       0.72      0.72      0.72       600

kNN Classification :
               precision    recall  f1-score   support

           0       0.99      0.97      0.98       150
           1       0.91      0.95      0.93       150
           2       0.90      0.92      0.91       150
           3       0.98      0.93      0.96       150

    accuracy                           0.94       600
   macro avg       0.94      0.94      0.94       600
weighted avg       0.94      0.94      0.94       600

Decision Tree:
               precision    recall  f1-score   support
           0       0.91      0.81      0.86       150
           1       0.64      0.81      0.72       150
           2       0.73      0.59      0.65       150
           3       0.85      0.89      0.87       150

    accuracy                           0.77       600
   macro avg       0.78      0.77      0.77       600
weighted avg       0.78      0.77      0.77       600
```

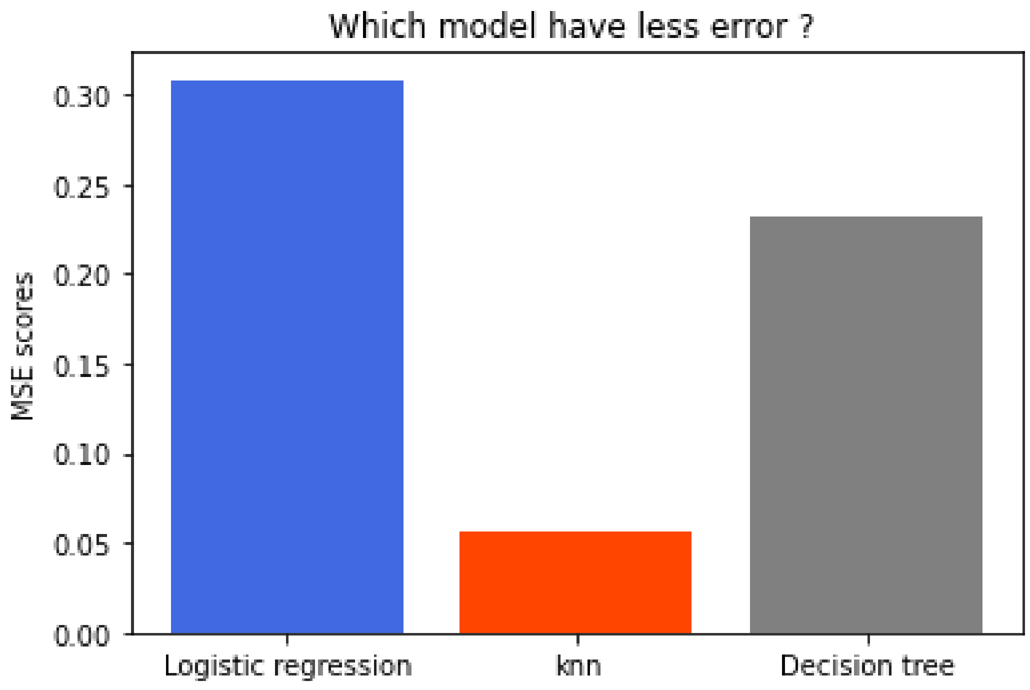
[174]:
```
models = ['Logistic regression', 'knn', 'Decision tree']
acc_scores = [acc_lr, acc_knn, acc_cart]
MSE_scores = [MSE_lr, MSE_knn, MSE_cart]
```

[175]:
```
plt.bar(models, acc_scores, color=['lightblue', 'pink', 'lightgrey'])
plt.ylabel("accuracy scores")
plt.title("Which model is the most accurate?")
plt.show()
```



We cleary see that the **kNN** model is more accurate than the **decision tree** and **logistic regression**.

[176]:
```
plt.bar(models, MSE_scores, color=['royalblue', 'orangered', 'grey'])
plt.ylabel("MSE scores")
plt.title("Which model have less error ? ")
plt.show()
```



Here, the MSE of both **Decision tree** and **Logistic Regression** are more than the one of the **kNN** model.

The best prediction model with no doubt is the **kNN**. In fact, the **kNN model have the biggest accurary score and the smallest mean square error**.

6 Price prediction of Test.csv Using KNN for Prediction

```
[177]: data_to_predict = data_to_predict.drop('id',axis=1)
       data_to_predict.head()
```

```
[177]:    battery_power  blue  clock_speed  dual_sim  fc  four_g  int_memory  m_dep  \
       0           1043     1          1.8         1  14       0           5    0.1
       1            841     1          0.5         1   4       1          61    0.8
       2           1807     1          2.8         0   1       0          27    0.9
       3           1546     0          0.5         1  18       1          25    0.5
       4           1434     0          1.4         0  11       1          49    0.5

          mobile_wt  n_cores  pc  px_height  px_width   ram  sc_h  sc_w  talk_time  \
       0        193        3  16        226      1412  3476    12     7          2
       1        191        5  12        746       857  3895     6     0          7
       2        186        3   4       1270      1366  2396    17    10         10
       3         96        8  20        295      1752  3893    10     0          7
       4        108        6  18        749       810  1773    15     8          7
```

```
   three_g  touch_screen  wifi
0        0             1     0
1        1             0     0
2        0             1     1
3        1             1     0
4        1             0     1
```

Now we can use our kNN model to predict our values.

```
[178]: predicted_price=neigh.predict(data_to_predict)
       predicted_price
```

```
[178]: array([3, 3, 2, 3, 1, 3, 3, 1, 3, 0, 3, 3, 0, 0, 2, 0, 2, 1, 3, 2, 1, 3,
              1, 1, 3, 0, 2, 0, 3, 0, 2, 0, 3, 0, 0, 1, 3, 1, 2, 1, 1, 2, 0, 0,
              0, 1, 0, 3, 1, 2, 1, 0, 3, 0, 3, 0, 3, 1, 1, 3, 3, 2, 0, 2, 1, 1,
              2, 3, 1, 2, 1, 2, 2, 3, 3, 0, 2, 0, 1, 3, 0, 3, 3, 0, 3, 0, 3, 1,
              3, 0, 1, 2, 2, 1, 2, 2, 1, 2, 1, 2, 1, 0, 0, 3, 0, 2, 0, 1, 2, 3,
              3, 3, 1, 3, 3, 3, 3, 1, 3, 0, 0, 3, 2, 1, 2, 0, 3, 2, 3, 1, 0, 2,
              2, 1, 3, 1, 1, 0, 3, 2, 1, 3, 1, 2, 2, 3, 3, 2, 2, 3, 2, 3, 1, 0,
              3, 2, 3, 3, 3, 3, 2, 2, 3, 3, 3, 3, 1, 0, 3, 0, 0, 0, 2, 1, 0, 1,
              0, 0, 1, 2, 1, 0, 0, 1, 1, 2, 2, 1, 0, 0, 0, 1, 0, 3, 1, 0, 2, 2,
              3, 3, 1, 2, 3, 2, 3, 2, 2, 1, 1, 0, 1, 2, 0, 2, 3, 3, 0, 2, 0, 3,
              2, 3, 3, 1, 0, 1, 0, 3, 0, 1, 0, 2, 2, 1, 2, 0, 2, 0, 3, 1, 2, 0,
              0, 2, 1, 3, 2, 3, 1, 1, 3, 0, 0, 2, 3, 3, 1, 3, 1, 1, 3, 2, 1, 2,
              3, 3, 3, 1, 0, 0, 2, 3, 1, 1, 3, 2, 1, 3, 0, 0, 3, 0, 0, 3, 2, 3,
              3, 2, 1, 3, 3, 2, 3, 1, 2, 1, 2, 0, 2, 3, 1, 0, 0, 3, 0, 3, 0, 1,
              2, 0, 2, 3, 1, 3, 2, 2, 1, 2, 0, 0, 0, 1, 3, 2, 0, 0, 0, 3, 2, 0,
              2, 3, 1, 2, 2, 2, 3, 1, 3, 3, 2, 2, 2, 3, 3, 1, 3, 0, 3, 1, 3, 1,
              3, 3, 0, 1, 0, 3, 1, 3, 2, 3, 0, 0, 0, 0, 2, 0, 0, 2, 2, 1, 2, 2,
              2, 0, 1, 0, 0, 3, 2, 0, 3, 1, 2, 2, 1, 2, 3, 1, 1, 2, 2, 1, 2, 0,
              1, 1, 0, 3, 2, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 2, 2, 3, 2, 3, 0, 3,
              0, 3, 0, 1, 1, 1, 1, 0, 3, 2, 3, 3, 1, 3, 1, 3, 1, 3, 2, 0, 1, 2,
              1, 2, 0, 0, 0, 1, 2, 1, 0, 3, 2, 0, 2, 2, 0, 0, 3, 1, 1, 0, 3, 2,
              3, 0, 3, 0, 2, 3, 3, 3, 0, 2, 0, 2, 3, 0, 1, 1, 0, 0, 1, 1, 1, 3,
              3, 3, 2, 3, 1, 2, 2, 3, 3, 3, 2, 0, 2, 1, 2, 2, 1, 0, 2, 2, 0, 0,
              0, 3, 1, 0, 2, 2, 2, 0, 3, 1, 2, 2, 0, 3, 0, 2, 3, 0, 1, 1, 3, 3,
              1, 1, 2, 3, 2, 0, 3, 0, 2, 0, 3, 3, 1, 3, 2, 2, 3, 0, 1, 2, 3, 1,
              3, 3, 3, 1, 1, 0, 0, 3, 1, 0, 3, 2, 3, 2, 0, 3, 3, 3, 2, 3, 3, 1,
              2, 0, 2, 2, 3, 1, 0, 1, 1, 2, 2, 2, 0, 0, 2, 2, 3, 2, 0, 2, 1, 3,
              3, 0, 1, 3, 0, 2, 1, 1, 0, 0, 2, 1, 0, 1, 1, 2, 2, 0, 2, 2, 1, 0,
              3, 0, 0, 3, 2, 0, 0, 0, 0, 0, 3, 0, 3, 1, 3, 2, 1, 3, 3, 0, 1, 0,
              3, 2, 2, 2, 1, 3, 0, 2, 0, 2, 0, 0, 1, 1, 1, 2, 1, 3, 1, 3, 2, 2,
              1, 3, 2, 0, 1, 2, 0, 3, 3, 0, 2, 1, 1, 2, 0, 3, 2, 0, 3, 2, 3, 0,
              0, 3, 0, 2, 2, 3, 2, 2, 2, 2, 1, 2, 3, 0, 1, 1, 1, 2, 1, 0, 0, 1,
              0, 0, 3, 1, 1, 2, 0, 0, 1, 1, 3, 0, 3, 2, 3, 0, 0, 1, 2, 2, 1, 0,
              1, 2, 0, 1, 1, 0, 0, 3, 3, 0, 3, 1, 2, 3, 0, 1, 0, 2, 2, 0, 3, 1,
              0, 3, 1, 1, 0, 3, 3, 3, 2, 3, 0, 3, 2, 0, 0, 0, 3, 3, 2, 0, 2, 1,
              2, 0, 0, 3, 2, 1, 3, 1, 2, 1, 1, 1, 3, 1, 1, 1, 2, 1, 0, 1, 2, 0,
              2, 0, 0, 1, 0, 2, 3, 3, 3, 0, 1, 2, 2, 1, 0, 0, 2, 1, 0, 2, 0, 2,
              2, 2, 1, 2, 0, 2, 1, 3, 0, 0, 3, 2, 3, 0, 0, 2, 3, 3, 1, 2, 2, 1,
              0, 0, 2, 3, 0, 3, 0, 0, 0, 2, 2, 1, 2, 0, 3, 2, 1, 2, 3, 3, 0, 2,
              1, 2, 1, 2, 2, 0, 1, 3, 1, 1, 3, 0, 2, 3, 2, 1, 1, 1, 3, 2, 0, 2,
              3, 0, 2, 3, 2, 2, 2, 3, 2, 0, 1, 2, 1, 2, 1, 1, 2, 2, 2, 1, 2, 1,
              1, 1, 3, 1, 0, 1, 2, 3, 1, 0, 0, 3, 2, 2, 3, 0, 3, 3, 2, 1, 3, 0,
              1, 3, 1, 1, 1, 1, 3, 2, 0, 3, 0, 2, 3, 0, 2, 2, 3, 3, 1, 0, 2, 3,
              1, 0, 2, 1, 2, 1, 2, 0, 2, 3, 0, 2, 3, 2, 3, 0, 2, 1, 1, 2, 2, 3,
              3, 0, 2, 1, 2, 1, 3, 0, 0, 3, 0, 1, 0, 0, 3, 3, 2, 0, 0, 0, 0, 3,
              2, 3, 3, 0, 0, 2, 1, 0, 2, 2])
```

[179]:
```
data_to_predict['price_range']=predicted_price
```

6.1 Conclusion

However, we have seen that some models are more likely to be credible than others. In particular, the kNN model is of good quality - better accuracy value, which means that for any threshold value the kNN estimator will be generally more reliable than logistic regression or decision tree. In addition, thanks to the correlation matrix, we therefore know that the main things that determine a mobile price are **ram** and **battery power** in a small way.