IS372 –Data Warehouse and Data Mining

Smart Phones Prices Prediction Project

Group Name

Emtenan Ibrahim Fallatah 4051593
Enas Asad Alghifari 3757072
Heba Bakur Alshanqiti 3750587
Reem Hameed Aljohani 4052012

Abstract:

Mobile phones are the best-selling electronic devices as people keep buying cell phones whenever they find new features in a new device. Recently, mobile phone companies become competing to develop the best features, which led to high prices, so we decided in this project develop a classification model for the data set containing the specifications of 2000 mobile phones trying to predict the best price ranges using python programming language.

Introduction:

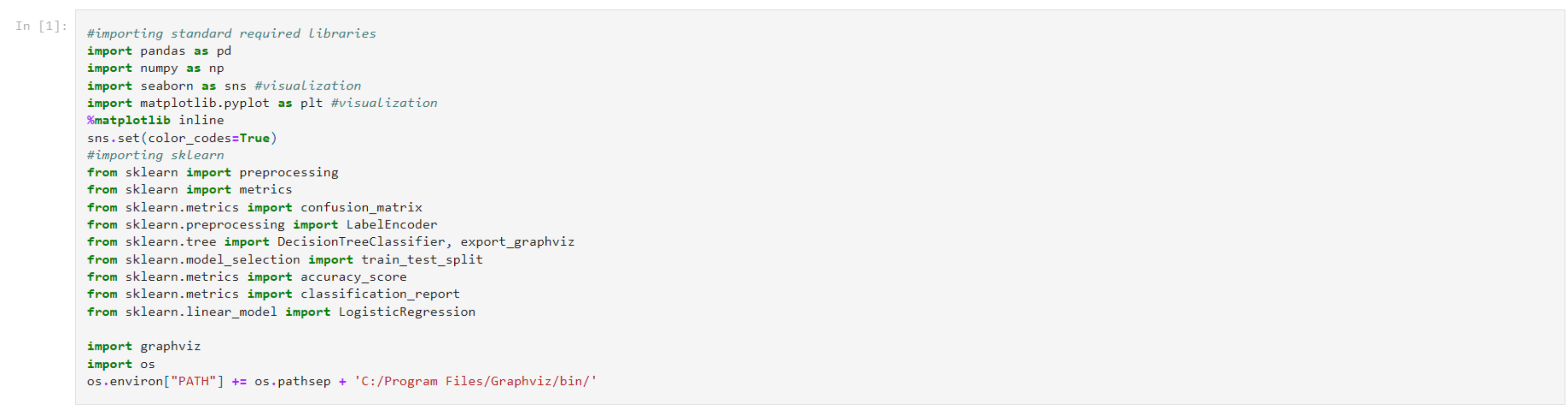
In this project, we obtain to explore and analyze a dataset that was found on Kaggle to predict the price range of cell phones based on the features for the phone and contain specifications of 2000 mobile phones using python programming language. we find out some relation between features of a mobile phone(eg:- RAM, Internal Memory , battery_power, touch_screen, price range etc...) and its selling price. We found that the RAM has the biggest impact on the price . Our task is to classify the price range of mobile phones we have four range [0, 1, 2, 3) The target variable indicates as below: 0 (low cost) 1 (medium cost) 2 (high cost) 3 (very high cost)
The problem can be solved problem of classification, by using the Logistic Regression algorithm.

Methods:

In this project, we apply a classification data mining technique using decision tree algorithm. generally, the feature with the highest accuracy among all others. and Logistic Regression used a standard scaler to scale are data variance and the accuracy score for training and validation in the model.

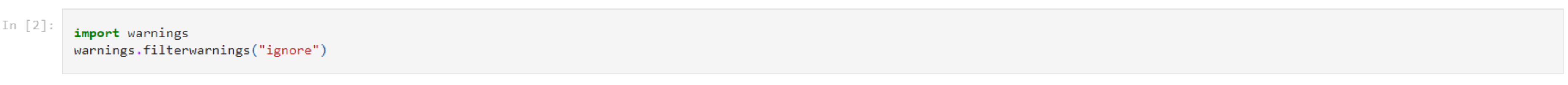
Codes:

1-load the most important packages



In [1]:

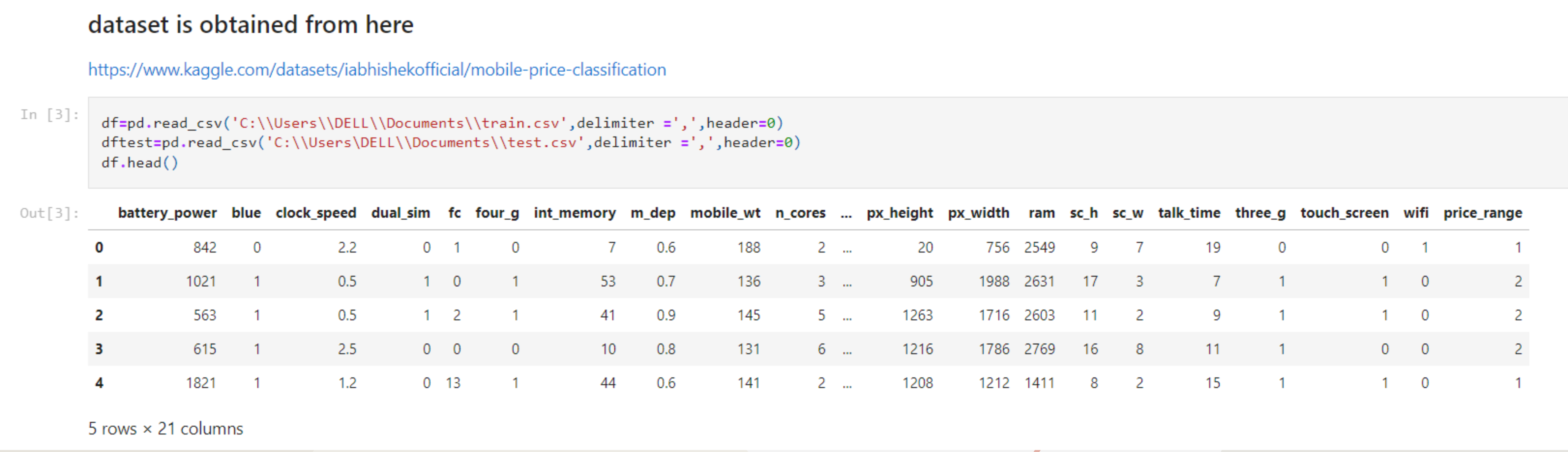
```
#importing standard required libraries
import pandas as pd
import numpy as np
import seaborn as sns #visualization
import matplotlib.pyplot as plt #visualization
%matplotlib inline
sns.set(color_codes=True)
#importing sklearn
from sklearn import preprocessing
from sklearn import metrics
from sklearn.metrics import confusion_matrix
from sklearn.preprocessing import LabelEncoder
from sklearn.tree import DecisionTreeClassifier, export_graphviz
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from sklearn.linear_model import LogisticRegression

import graphviz
import os
os.environ["PATH"] += os.pathsep + 'C:/Program Files/Graphviz/bin/'
```



In [2]:

```
import warnings
warnings.filterwarnings("ignore")
```

2-load the dataset from kaggle
https://www.kaggle.com/datasets/iabhishekofficial/mobile-price-classification



dataset is obtained from here

https://www.kaggle.com/datasets/iabhishekofficial/mobile-price-classification

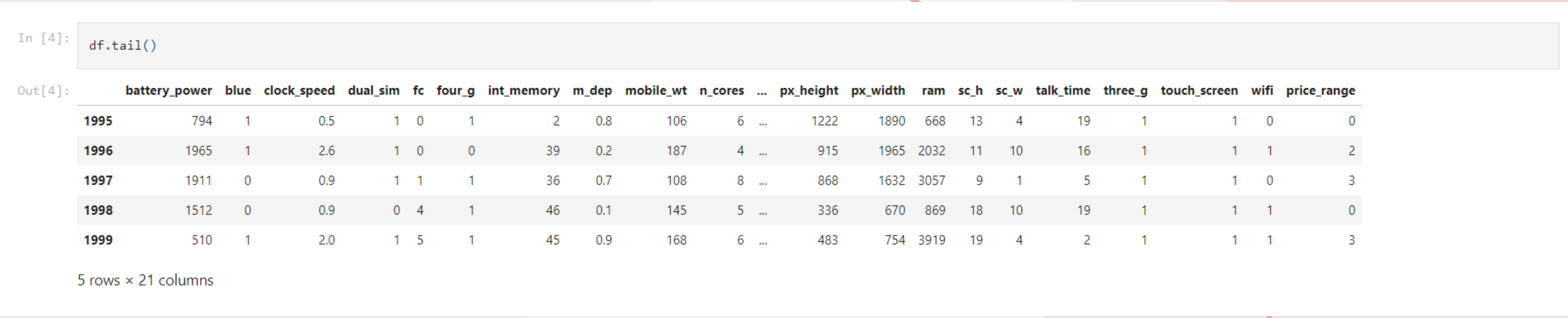
In [3]:

```
df=pd.read_csv('C:\\Users\\DELL\\Documents\\train.csv',delimiter =',',header=0)
dftest=pd.read_csv('C:\\Users\DELL\\Documents\\test.csv',delimiter =',',header=0)
df.head()
```

Out[3]:

	battery_power	blue	clock_speed	dual_sim	fc	four_g	int_memory	m_dep	mobile_wt	n_cores	...	px_height	px_width	ram	sc_h	sc_w	talk_time	three_g	touch_screen	wifi	price_range
0	842	0	2.2	0	1	0	7	0.6	188	2	...	20	756	2549	9	7	19	0	0	1	1
1	1021	1	0.5	1	0	1	53	0.7	136	3	...	905	1988	2631	17	3	7	1	1	0	2
2	563	1	0.5	1	2	1	41	0.9	145	5	...	1263	1716	2603	11	2	9	1	1	0	2
3	615	1	2.5	0	0	0	10	0.8	131	6	...	1216	1786	2769	16	8	11	1	0	0	2
4	1821	1	1.2	0	13	1	44	0.6	141	2	...	1208	1212	1411	8	2	15	1	1	0	1

5 rows × 21 columns



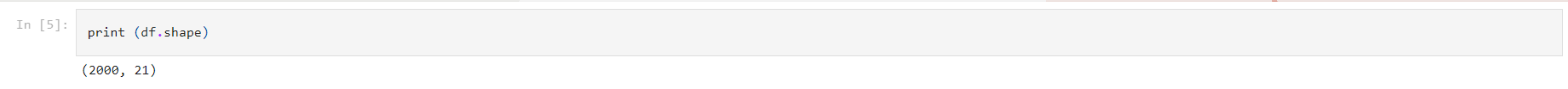
In [4]:

```
df.tail()
```

Out[4]:

	battery_power	blue	clock_speed	dual_sim	fc	four_g	int_memory	m_dep	mobile_wt	n_cores	...	px_height	px_width	ram	sc_h	sc_w	talk_time	three_g	touch_screen	wifi	price_range
1995	794	1	0.5	1	0	1	2	0.8	106	6	...	1222	1890	668	13	4	19	1	1	0	0
1996	1965	1	2.6	1	0	0	39	0.2	187	4	...	915	1965	2032	11	10	16	1	1	1	2
1997	1911	0	0.9	1	1	1	36	0.7	108	8	...	868	1632	3057	9	1	5	1	1	0	3
1998	1512	0	0.9	0	4	1	46	0.1	145	5	...	336	670	869	18	10	19	1	1	1	0
1999	510	1	2.0	1	5	1	45	0.9	168	6	...	483	754	3919	19	4	2	1	1	1	3

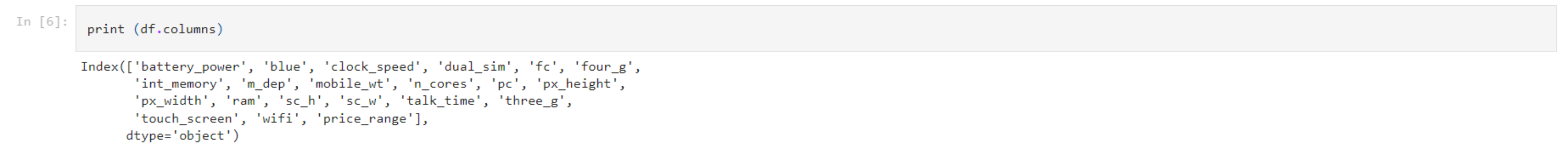
5 rows × 21 columns



In [5]:

```
print (df.shape)
```

```
(2000, 21)
```

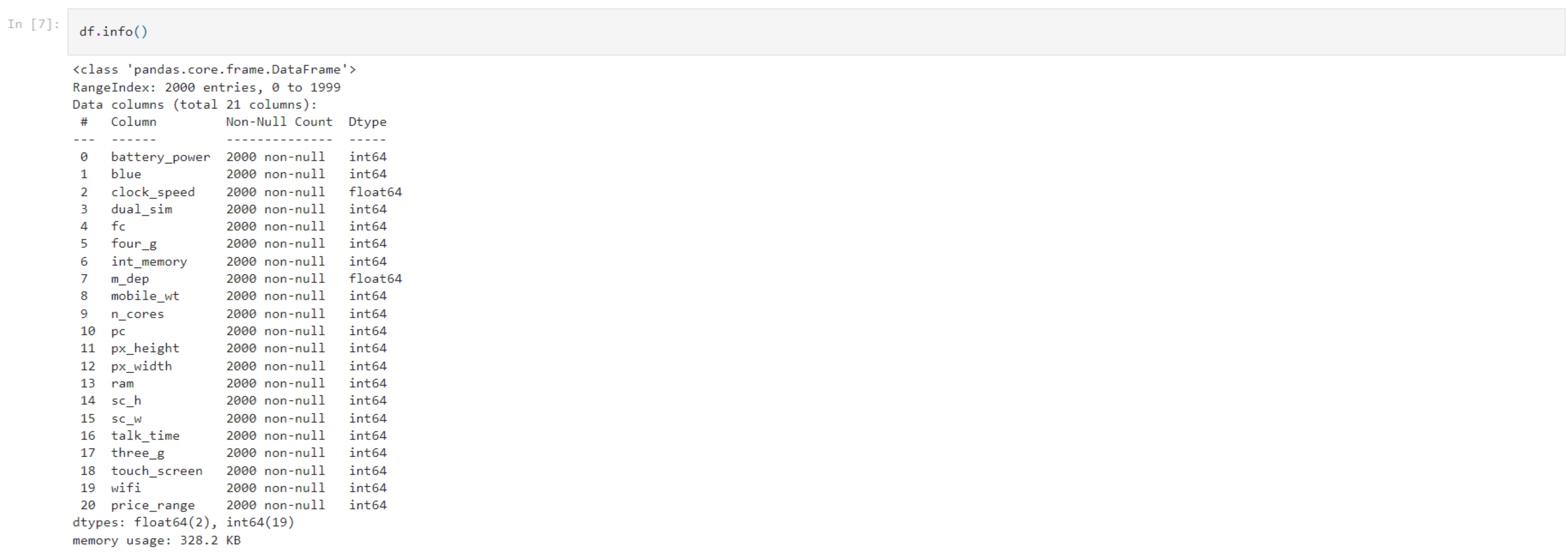


In [6]:

```
print (df.columns)
```

```
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')
```

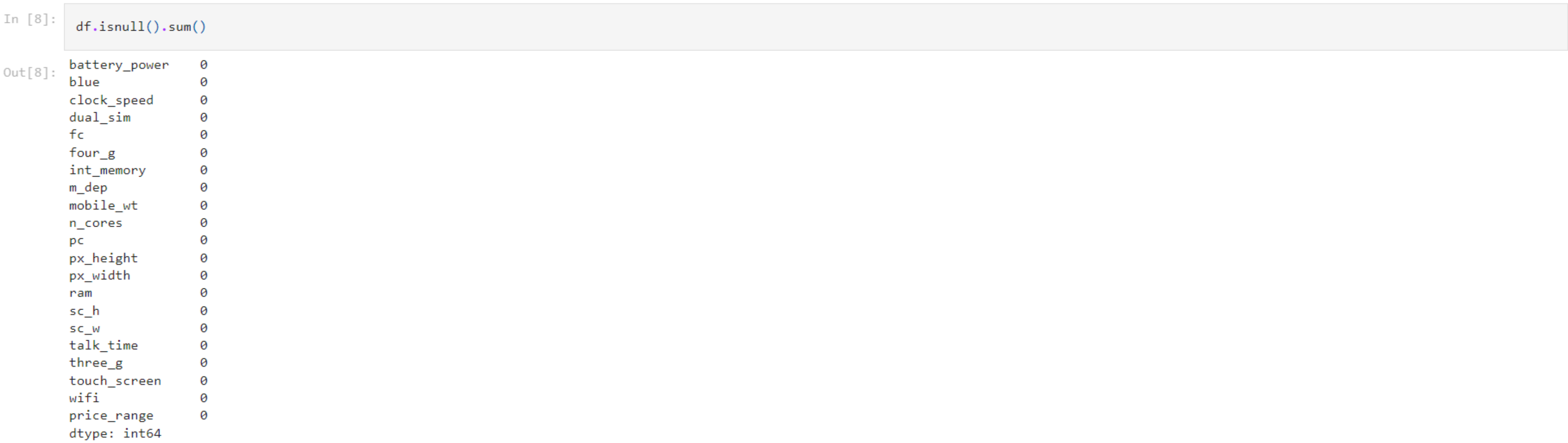
3-that Info(): It returns the names of the columns, type of data in each frame and all data in the dataset are numeric and the data type are int64 and float64 appears is null() Which means no has a missed value, so the dataset we have is already pre- processed.



In [7]:

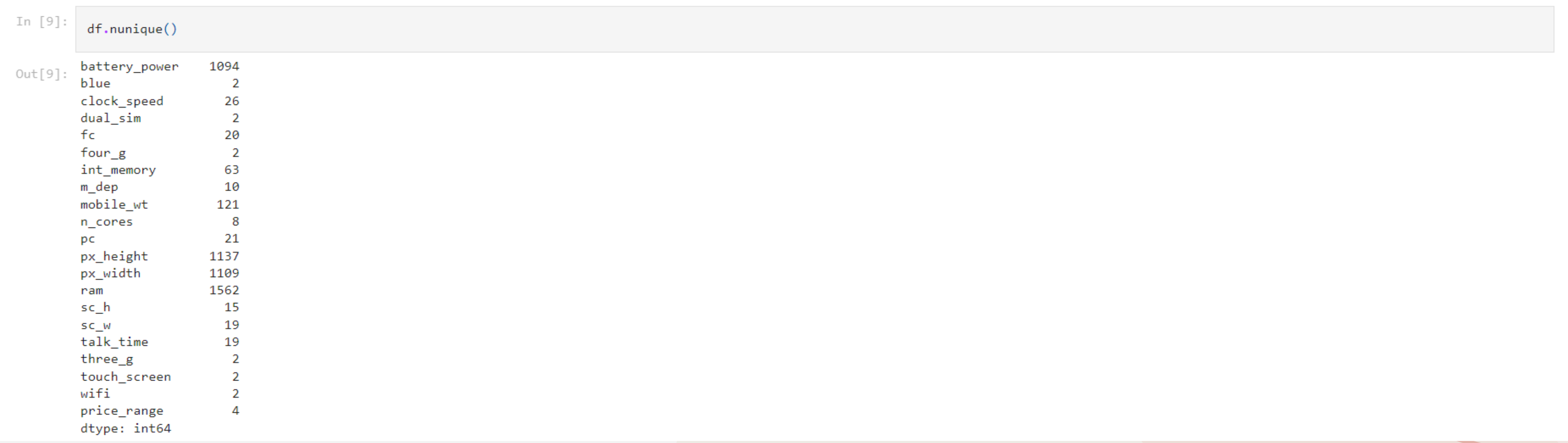
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2000 entries, 0 to 1999
Data columns (total 21 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   battery_power  2000 non-null   int64  
 1   blue           2000 non-null   int64  
 2   clock_speed    2000 non-null   float64
 3   dual_sim       2000 non-null   int64  
 4   fc             2000 non-null   int64  
 5   four_g         2000 non-null   int64  
 6   int_memory     2000 non-null   int64  
 7   m_dep          2000 non-null   float64
 8   mobile_wt      2000 non-null   int64  
 9   n_cores        2000 non-null   int64  
 10  pc             2000 non-null   int64  
 11  px_height      2000 non-null   int64  
 12  px_width       2000 non-null   int64  
 13  ram            2000 non-null   int64  
 14  sc_h           2000 non-null   int64  
 15  sc_w           2000 non-null   int64  
 16  talk_time      2000 non-null   int64  
 17  three_g        2000 non-null   int64  
 18  touch_screen   2000 non-null   int64  
 19  wifi           2000 non-null   int64  
 20  price_range    2000 non-null   int64  
dtypes: float64(2), int64(19)
memory usage: 328.2 KB
```



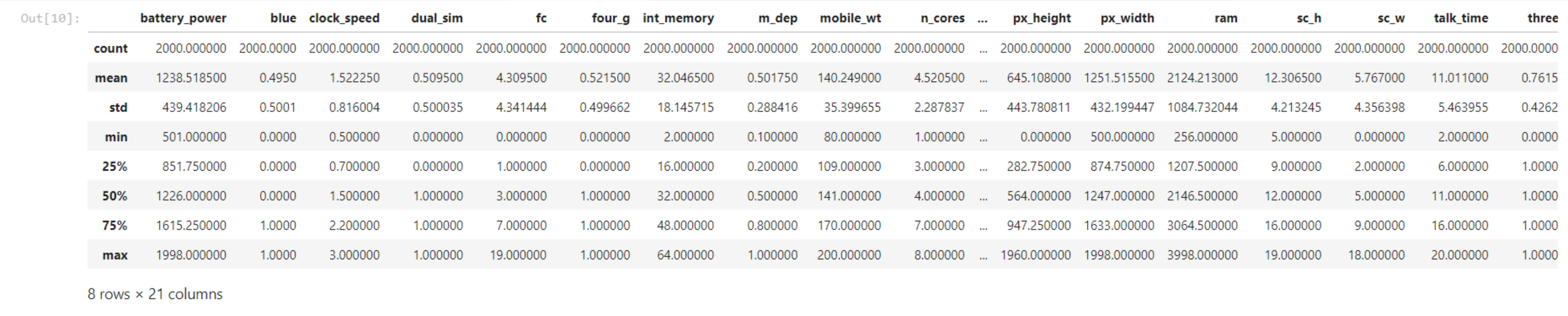
```
In [8]: df.isnull().sum()
```

```
Out[8]: battery_power    0
        blue             0
        clock_speed      0
        dual_sim         0
        fc               0
        four_g           0
        int_memory       0
        m_dep            0
        mobile_wt        0
        n_cores          0
        pc               0
        px_height        0
        px_width         0
        ram              0
        sc_h             0
        sc_w             0
        talk_time        0
        three_g          0
        touch_screen     0
        wifi             0
        price_range      0
        dtype: int64
```



```
In [9]:  df.nunique()
```

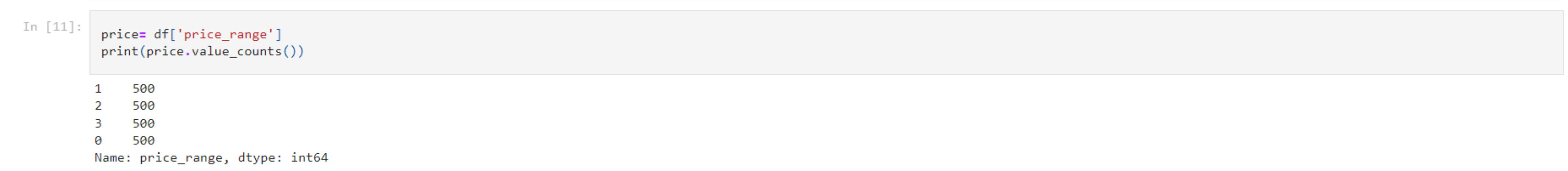
```
Out[9]:  battery_power    1094
         blue                2
         clock_speed        26
         dual_sim            2
         fc                 20
         four_g              2
         int_memory         63
         m_dep              10
         mobile_wt         121
         n_cores             8
         pc                 21
         px_height        1137
         px_width         1109
         ram              1562
         sc_h               15
         sc_w               19
         talk_time          19
         three_g             2
         touch_screen        2
         wifi                2
         price_range         4
         dtype: int64
```



Out[10]:

	battery_power	blue	clock_speed	dual_sim	fc	four_g	int_memory	m_dep	mobile_wt	n_cores	...	px_height	px_width	ram	sc_h	sc_w	talk_time	three
count	2000.000000	2000.0000	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000	...	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000	2000.0000
mean	1238.518500	0.4950	1.522250	0.509500	4.309500	0.521500	32.046500	0.501750	140.249000	4.520500	...	645.108000	1251.515500	2124.213000	12.306500	5.767000	11.011000	0.7615
std	439.418206	0.5001	0.816004	0.500035	4.341444	0.499662	18.145715	0.288416	35.399655	2.287837	...	443.780811	432.199447	1084.732044	4.213245	4.356398	5.463955	0.4262
min	501.000000	0.0000	0.500000	0.000000	0.000000	0.000000	2.000000	0.100000	80.000000	1.000000	...	0.000000	500.000000	256.000000	5.000000	0.000000	2.000000	0.0000
25%	851.750000	0.0000	0.700000	0.000000	1.000000	0.000000	16.000000	0.200000	109.000000	3.000000	...	282.750000	874.750000	1207.500000	9.000000	2.000000	6.000000	1.0000
50%	1226.000000	0.0000	1.500000	1.000000	3.000000	1.000000	32.000000	0.500000	141.000000	4.000000	...	564.000000	1247.000000	2146.500000	12.000000	5.000000	11.000000	1.0000
75%	1615.250000	1.0000	2.200000	1.000000	7.000000	1.000000	48.000000	0.800000	170.000000	7.000000	...	947.250000	1633.000000	3064.500000	16.000000	9.000000	16.000000	1.0000
max	1998.000000	1.0000	3.000000	1.000000	19.000000	1.000000	64.000000	1.000000	200.000000	8.000000	...	1960.000000	1998.000000	3998.000000	19.000000	18.000000	20.000000	1.0000

8 rows × 21 columns

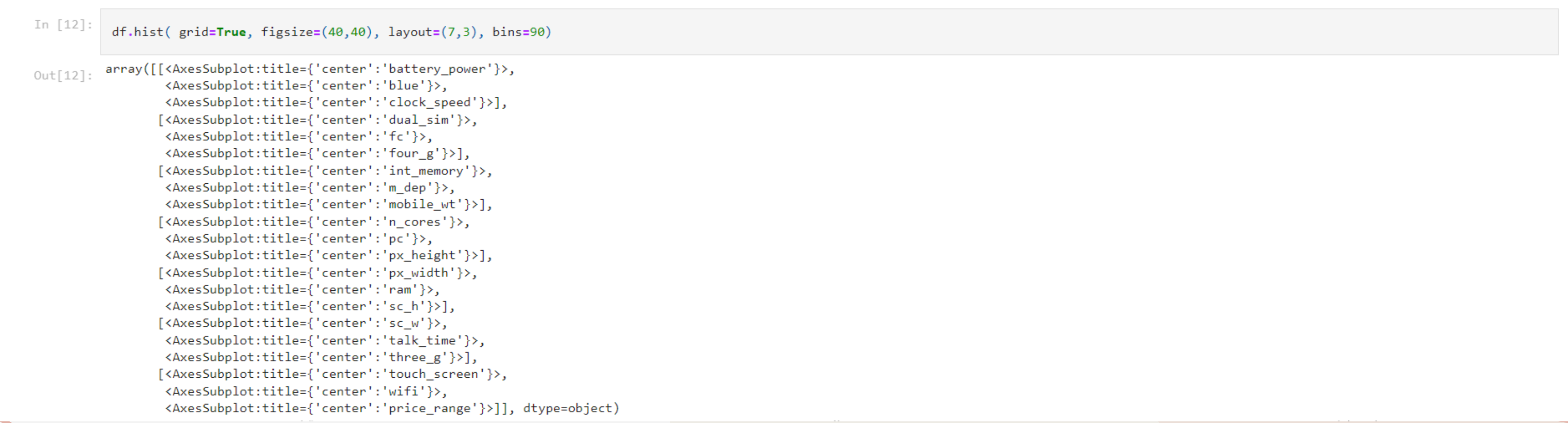


In [11]:

```
price= df['price_range']
print(price.value_counts())
```

```
1    500
2    500
3    500
0    500
Name: price_range, dtype: int64
```

2.Data visualization

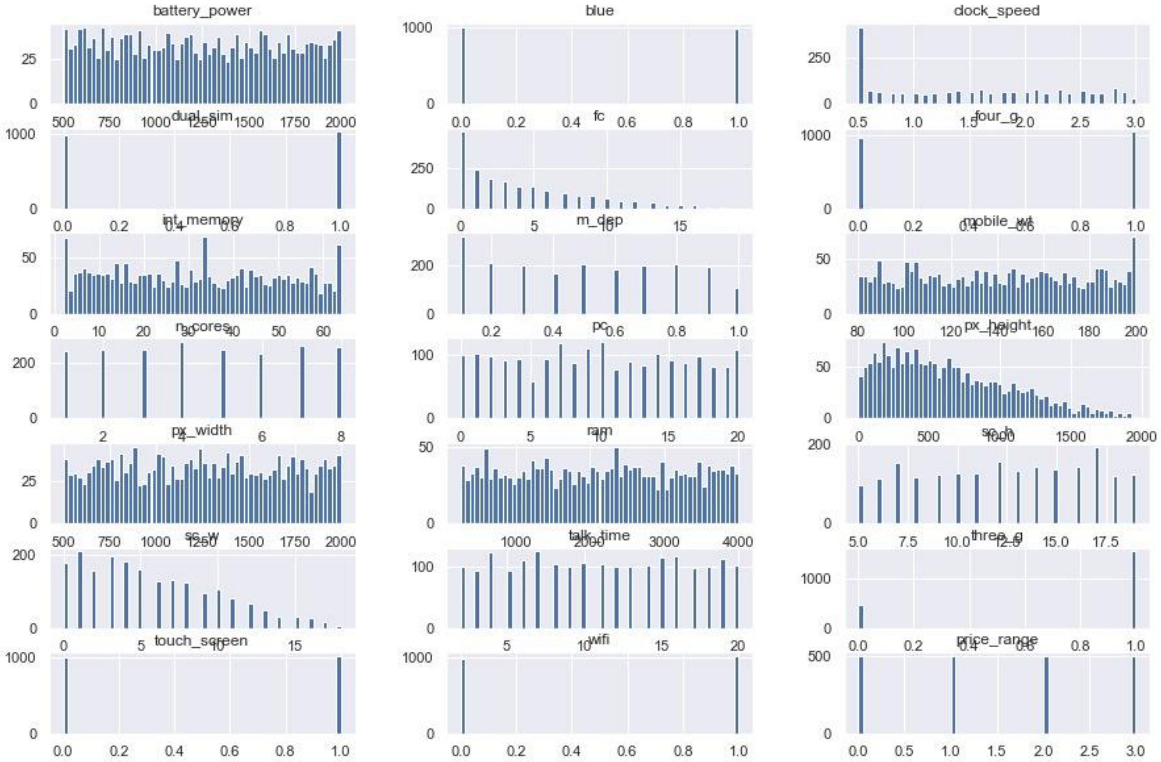


In [12]:

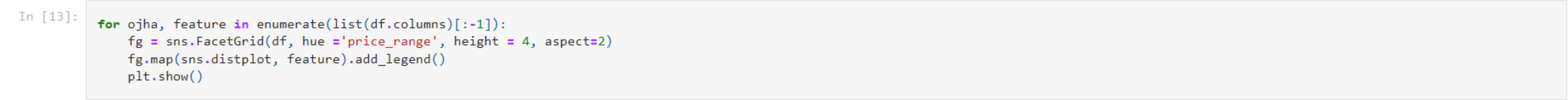
```
df.hist( grid=True, figsize=(40,40), layout=(7,3), bins=90)
```

Out[12]:

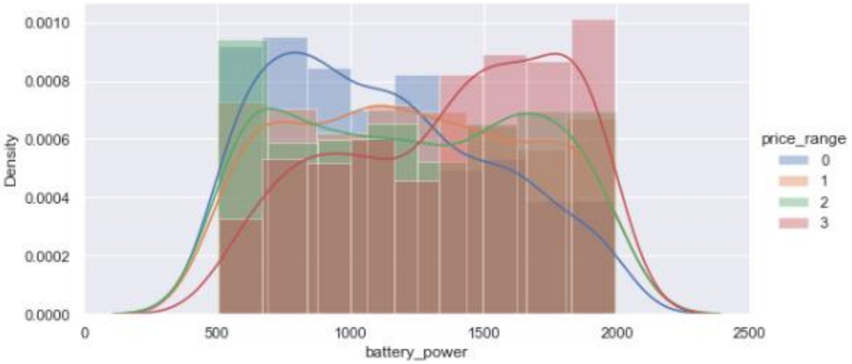
```
array([[<AxesSubplot:title={'center':'battery_power'}>,
        <AxesSubplot:title={'center':'blue'}>,
        <AxesSubplot:title={'center':'clock_speed'}>],
       [<AxesSubplot:title={'center':'dual_sim'}>,
        <AxesSubplot:title={'center':'fc'}>,
        <AxesSubplot:title={'center':'four_g'}>],
       [<AxesSubplot:title={'center':'int_memory'}>,
        <AxesSubplot:title={'center':'m_dep'}>,
        <AxesSubplot:title={'center':'mobile_wt'}>],
       [<AxesSubplot:title={'center':'n_cores'}>,
        <AxesSubplot:title={'center':'pc'}>,
        <AxesSubplot:title={'center':'px_height'}>],
       [<AxesSubplot:title={'center':'px_width'}>,
        <AxesSubplot:title={'center':'ram'}>,
        <AxesSubplot:title={'center':'sc_h'}>],
       [<AxesSubplot:title={'center':'sc_w'}>,
        <AxesSubplot:title={'center':'talk_time'}>,
        <AxesSubplot:title={'center':'three_g'}>],
       [<AxesSubplot:title={'center':'touch_screen'}>,
        <AxesSubplot:title={'center':'wifi'}>,
        <AxesSubplot:title={'center':'price_range'}>]], dtype=object)
```

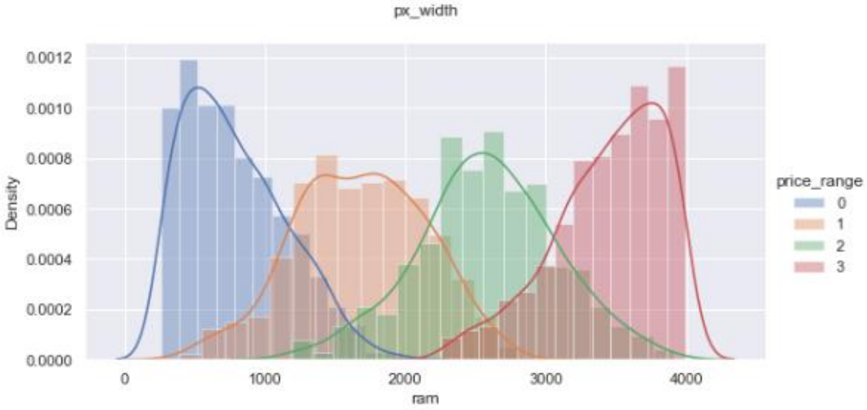

battery_power
0
25
500 750 1000 1250 1500 1750 2000
blue
0
1000
0.0 0.2 0.4 0.6 0.8 1.0
clock_speed
0
250
0.5 1.0 1.5 2.0 2.5 3.0
dual_sim
0
1000
0.0 0.2 0.4 0.6 0.8 1.0
fc
0
250
0 5 10 15
four_g
0
1000
0.0 0.2 0.4 0.6 0.8 1.0
int_memory
0
50
0 10 20 30 40 50 60
m_dep
0
200
0.2 0.4 0.6 0.8 1.0
mobile_wt
0
50
80 100 120 140 160 180 200
n_cores
0
200
2 4 6 8
pc
0
100
0 5 10 15 20
px_height
0
50
0 500 1000 1500 2000
px_width
0
25
500 750 1000 1250 1500 1750 2000
ram
0
50
1000 2000 3000 4000
sc_h
0
200
5.0 7.5 10.0 12.5 15.0 17.5
sc_w
0
200
0 5 10 15
talk_time
0
100
5 10 15 20
three_g
0
1000
0.0 0.2 0.4 0.6 0.8 1.0
touch_screen
0
1000
0.0 0.2 0.4 0.6 0.8 1.0
wifi
0
1000
0.0 0.2 0.4 0.6 0.8 1.0
price_range
0
500
0.0 0.5 1.0 1.5 2.0 2.5 3.0

A histogram represents the distribution of data by forming bins along the range of the data and then drawing bars to show the number of observations that fall in each bin

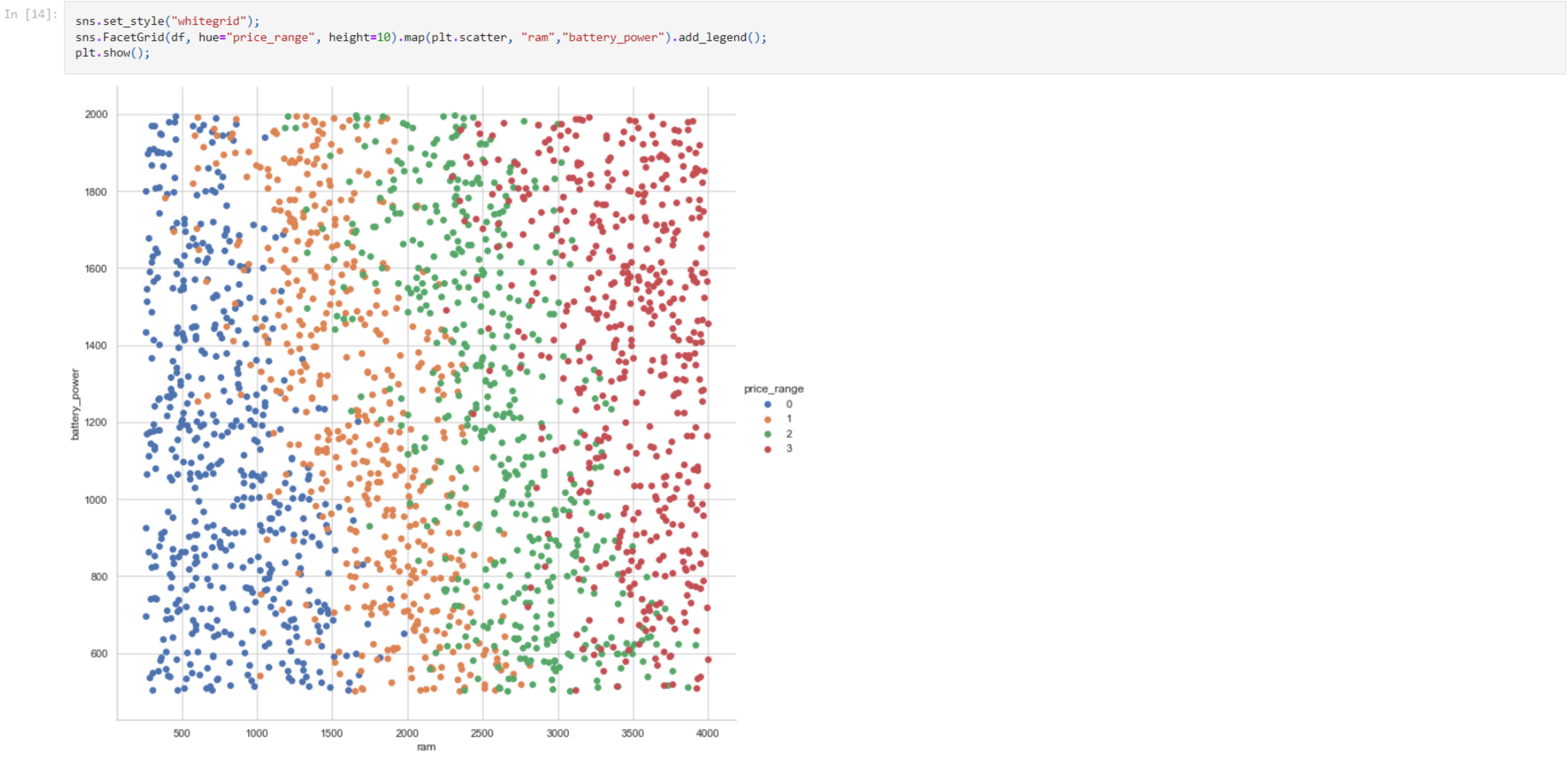


```
In [13]:  for ojha, feature in enumerate(list(df.columns)[:-1]):
              fg = sns.FacetGrid(df, hue ='price_range', height = 4, aspect=2)
              fg.map(sns.distplot, feature).add_legend()
              plt.show()
```





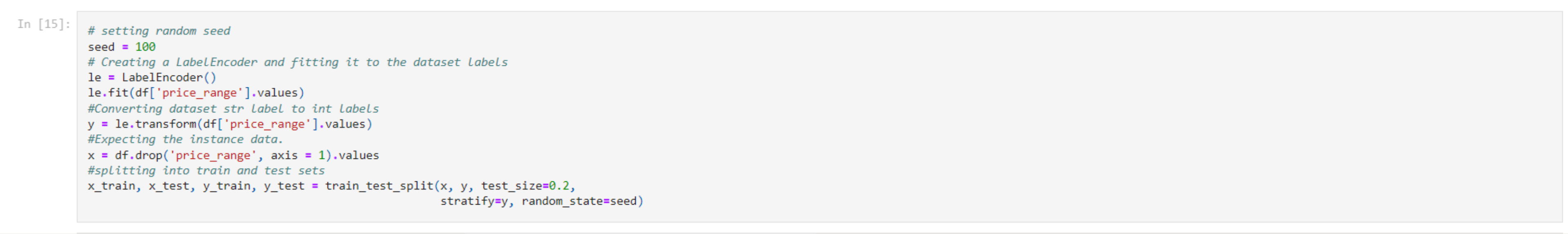
the scatter diagram for the ram & battery_power



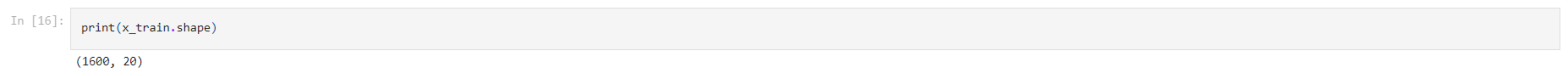
In [14]:

```
sns.set_style("whitegrid");
sns.FacetGrid(df, hue="price_range", height=10).map(plt.scatter, "ram","battery_power").add_legend();
plt.show();
```

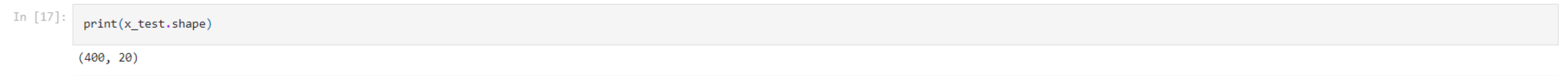
After making sure data frame correct start splitting the data up. Since there is a test & train CSV file,



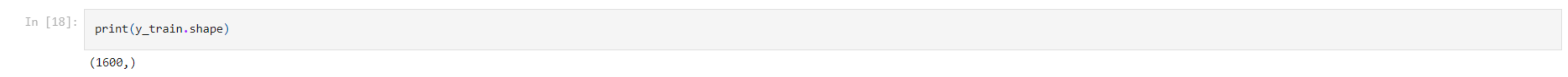
```
In [15]:
# setting random seed
seed = 100
# Creating a LabelEncoder and fitting it to the dataset labels
le = LabelEncoder()
le.fit(df['price_range'].values)
#Converting dataset str label to int labels
y = le.transform(df['price_range'].values)
#Expecting the instance data.
x = df.drop('price_range', axis = 1).values
#splitting into train and test sets
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2,
                                                    stratify=y, random_state=seed)
```



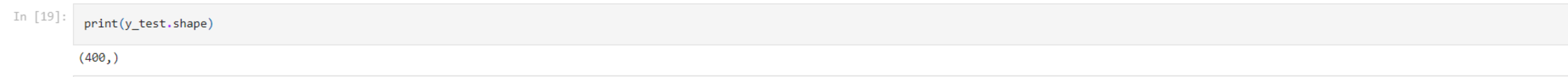
```
In [16]:
print(x_train.shape)

(1600, 20)
```



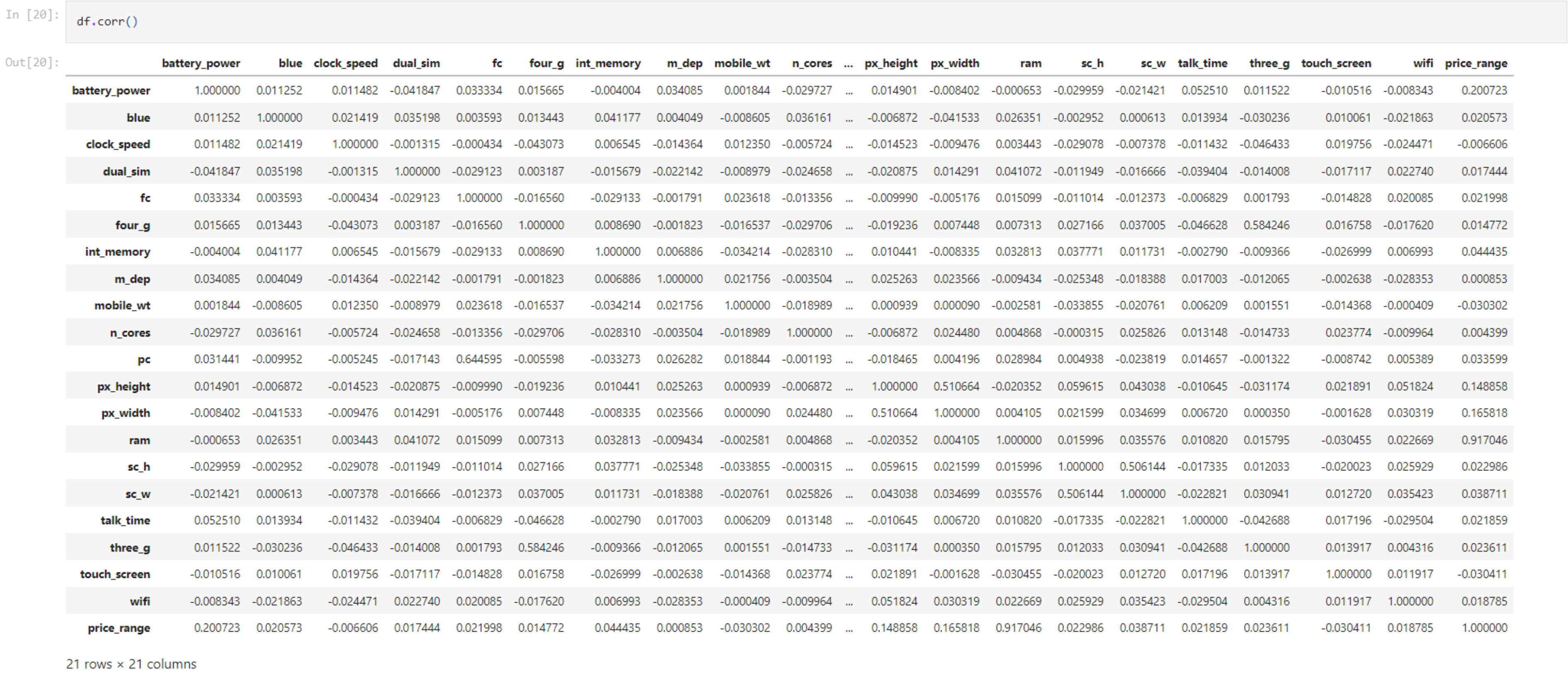
```
In [17]:
print(x_test.shape)

(400, 20)
```



```
In [18]:
print(y_train.shape)

(1600,)
```



```
In [19]:
print(y_test.shape)

(400,)
```



In [20]:

```
df.corr()
```

Out[20]:

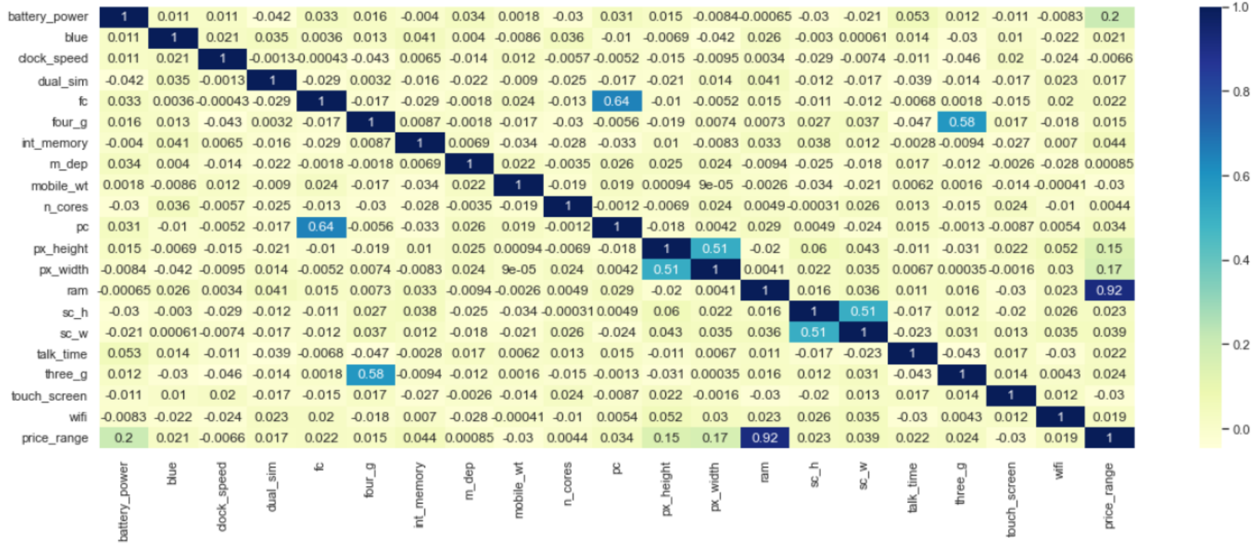
	battery_power	blue	clock_speed	dual_sim	fc	four_g	int_memory	m_dep	mobile_wt	n_cores	...	px_height	px_width	ram	sc_h	sc_w	talk_time	three_g	touch_screen	wifi	price_range
battery_power	1.000000	0.011252	0.011482	-0.041847	0.033334	0.015665	-0.004004	0.034085	0.001844	-0.029727	...	0.014901	-0.008402	-0.000653	-0.029959	-0.021421	0.052510	0.011522	-0.010516	-0.008343	0.200723
blue	0.011252	1.000000	0.021419	0.035198	0.003593	0.013443	0.041177	0.004049	-0.008605	0.036161	...	-0.006872	-0.041533	0.026351	-0.002952	0.000613	0.013934	-0.030236	0.010061	-0.021863	0.020573
clock_speed	0.011482	0.021419	1.000000	-0.001315	-0.000434	-0.043073	0.006545	-0.014364	0.012350	-0.005724	...	-0.014523	-0.009476	0.003443	-0.029078	-0.007378	-0.011432	-0.046433	0.019756	-0.024471	-0.006606
dual_sim	-0.041847	0.035198	-0.001315	1.000000	-0.029123	0.003187	-0.015679	-0.022142	-0.008979	-0.024658	...	-0.020875	0.014291	0.041072	-0.011949	-0.016666	-0.039404	-0.014008	-0.017117	0.022740	0.017444
fc	0.033334	0.003593	-0.000434	-0.029123	1.000000	-0.016560	-0.029133	-0.001791	0.023618	-0.013356	...	-0.009990	-0.005176	0.015099	-0.011014	-0.012373	-0.006829	0.001793	-0.014828	0.020085	0.021998
four_g	0.015665	0.013443	-0.043073	0.003187	-0.016560	1.000000	0.008690	-0.001823	-0.016537	-0.029706	...	-0.019236	0.007448	0.007313	0.027166	0.037005	-0.046628	0.584246	0.016758	-0.017620	0.014772
int_memory	-0.004004	0.041177	0.006545	-0.015679	-0.029133	0.008690	1.000000	0.006886	-0.034214	-0.028310	...	0.010441	-0.008335	0.032813	0.037771	0.011731	-0.002790	-0.009366	-0.026999	0.006993	0.044435
m_dep	0.034085	0.004049	-0.014364	-0.022142	-0.001791	-0.001823	0.006886	1.000000	0.021756	-0.003504	...	0.025263	0.023566	-0.009434	-0.025348	-0.018388	0.017003	-0.012065	-0.002638	-0.028353	0.000853
mobile_wt	0.001844	-0.008605	0.012350	-0.008979	0.023618	-0.016537	-0.034214	0.021756	1.000000	-0.018989	...	0.000939	0.000090	-0.002581	-0.033855	-0.020761	0.006209	0.001551	-0.014368	-0.000409	-0.030302
n_cores	-0.029727	0.036161	-0.005724	-0.024658	-0.013356	-0.029706	-0.028310	-0.003504	-0.018989	1.000000	...	-0.006872	0.024480	0.004868	-0.000315	0.025826	0.013148	-0.014733	0.023774	-0.009964	0.004399
pc	0.031441	-0.009952	-0.005245	-0.017143	0.644595	-0.005598	-0.033273	0.026282	0.018844	-0.001193	...	-0.018465	0.004196	0.028984	0.004938	-0.023819	0.014657	-0.001322	-0.008742	0.005389	0.033599
px_height	0.014901	-0.006872	-0.014523	-0.020875	-0.009990	-0.019236	0.010441	0.025263	0.000939	-0.006872	...	1.000000	0.510664	-0.020352	0.059615	0.043038	-0.010645	-0.031174	0.021891	0.051824	0.148858
px_width	-0.008402	-0.041533	-0.009476	0.014291	-0.005176	0.007448	-0.008335	0.023566	0.000090	0.024480	...	0.510664	1.000000	0.004105	0.021599	0.034699	0.006720	0.000350	-0.001628	0.030319	0.165818
ram	-0.000653	0.026351	0.003443	0.041072	0.015099	0.007313	0.032813	-0.009434	-0.002581	0.004868	...	-0.020352	0.004105	1.000000	0.015996	0.035576	0.010820	0.015795	-0.030455	0.022669	0.917046
sc_h	-0.029959	-0.002952	-0.029078	-0.011949	-0.011014	0.027166	0.037771	-0.025348	-0.033855	-0.000315	...	0.059615	0.021599	0.015996	1.000000	0.506144	-0.017335	0.012033	-0.020023	0.025929	0.022986
sc_w	-0.021421	0.000613	-0.007378	-0.016666	-0.012373	0.037005	0.011731	-0.018388	-0.020761	0.025826	...	0.043038	0.034699	0.035576	0.506144	1.000000	-0.022821	0.030941	0.012720	0.035423	0.038711
talk_time	0.052510	0.013934	-0.011432	-0.039404	-0.006829	-0.046628	-0.002790	0.017003	0.006209	0.013148	...	-0.010645	0.006720	0.010820	-0.017335	-0.022821	1.000000	-0.042688	0.017196	-0.029504	0.021859
three_g	0.011522	-0.030236	-0.046433	-0.014008	0.001793	0.584246	-0.009366	-0.012065	0.001551	-0.014733	...	-0.031174	0.000350	0.015795	0.012033	0.030941	-0.042688	1.000000	0.013917	0.004316	0.023611
touch_screen	-0.010516	0.010061	0.019756	-0.017117	-0.014828	0.016758	-0.026999	-0.002638	-0.014368	0.023774	...	0.021891	-0.001628	-0.030455	-0.020023	0.012720	0.017196	0.013917	1.000000	0.011917	-0.030411
wifi	-0.008343	-0.021863	-0.024471	0.022740	0.020085	-0.017620	0.006993	-0.028353	-0.000409	-0.009964	...	0.051824	0.030319	0.022669	0.025929	0.035423	-0.029504	0.004316	0.011917	1.000000	0.018785
price_range	0.200723	0.020573	-0.006606	0.017444	0.021998	0.014772	0.044435	0.000853	-0.030302	0.004399	...	0.148858	0.165818	0.917046	0.022986	0.038711	0.021859	0.023611	-0.030411	0.018785	1.000000

21 rows × 21 columns

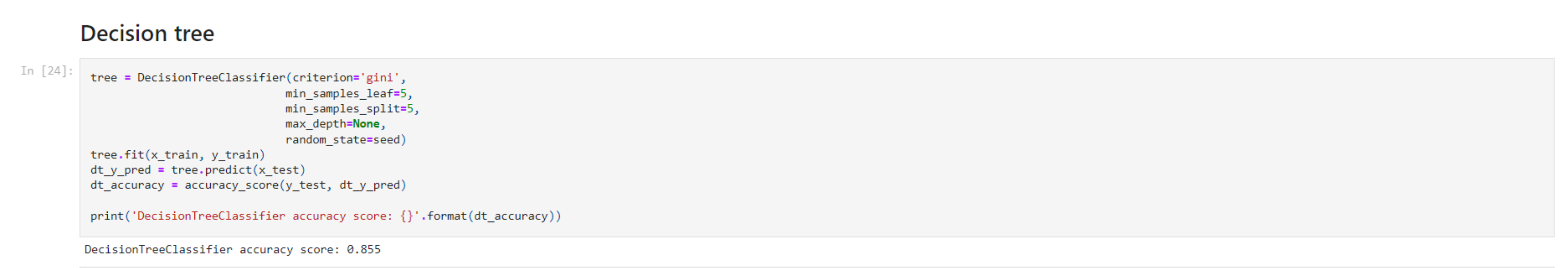
```
In [22]: np.corrcoef(df)
```

```
Out[22]: array([[1.        , 0.90128146, 0.87027602, ..., 0.93540453, 0.71878856,
          0.96990117],
         [0.90128146, 1.        , 0.98088888, ..., 0.96372714, 0.76652974,
          0.87040416],
         [0.87027602, 0.98088888, 1.        , ..., 0.92275458, 0.66808499,
          0.8859005 ],
         ...,
         [0.93540453, 0.96372714, 0.92275458, ..., 1.        , 0.87951045,
          0.87783515],
         [0.71878856, 0.76652974, 0.66808499, ..., 0.87951045, 1.        ,
          0.57737594],
         [0.96990117, 0.87040416, 0.8859005 , ..., 0.87783515, 0.57737594,
          1.        ]])
```

```
In [23]: plt.figure(figsize=(20, 7))
         sns.heatmap(df.corr(),cmap="YlGnBu",annot=True)
         plt.show()
```



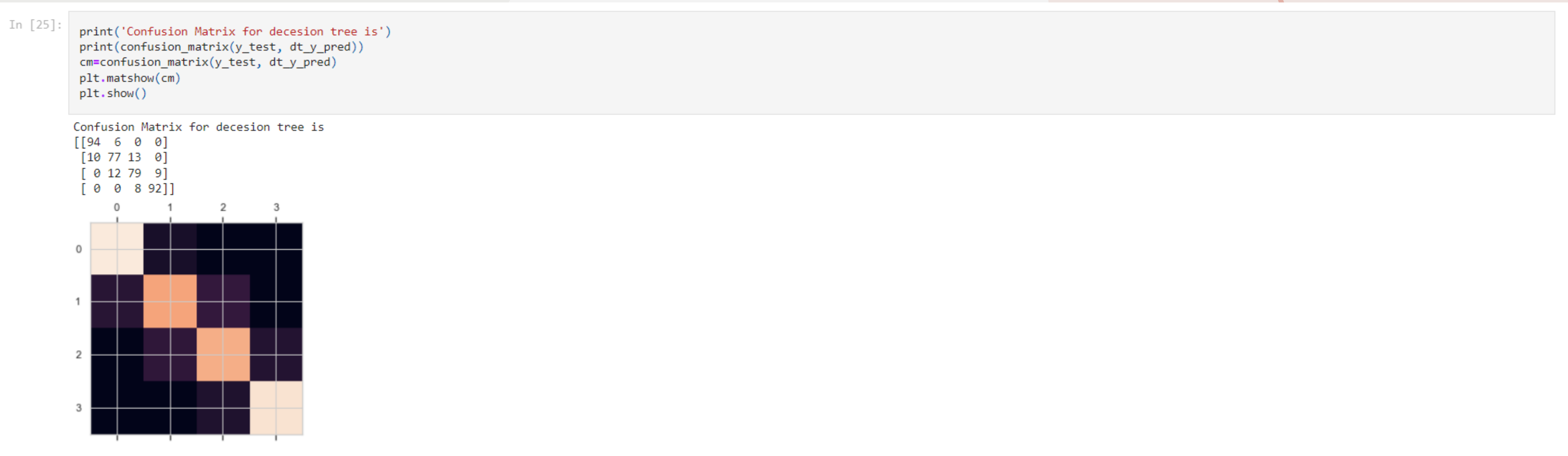
	battery_power	blue	clock_speed	dual_sim	fc	four_g	int_memory	m_dep	mobile_wt	n_cores	pc	px_height	px_width	ram	sc_h	sc_w	talk_time	three_g	touch_screen	wifi	price_range
battery_power	1	0.011	0.011	-0.042	0.033	0.016	-0.004	0.034	0.0018	-0.03	0.031	0.015	-0.0084	-0.00065	-0.03	-0.021	0.053	0.012	-0.011	-0.0083	0.2
blue	0.011	1	0.021	0.035	0.0036	0.013	0.041	0.004	-0.0086	0.036	-0.01	-0.0069	-0.042	0.026	-0.003	0.00061	0.014	-0.03	0.01	-0.022	0.021
clock_speed	0.011	0.021	1	-0.0013	-0.00043	-0.043	0.0065	-0.014	0.012	-0.0057	-0.0052	-0.015	-0.0095	0.0034	-0.029	-0.0074	-0.011	-0.046	0.02	-0.024	-0.0066
dual_sim	-0.042	0.035	-0.0013	1	-0.029	0.0032	-0.016	-0.022	-0.009	-0.025	-0.017	-0.021	0.014	0.041	-0.012	-0.017	-0.039	-0.014	-0.017	0.023	0.017
fc	0.033	0.0036	-0.00043	-0.029	1	-0.017	-0.029	-0.0018	0.024	-0.013	0.64	-0.01	-0.0052	0.015	-0.011	-0.012	-0.0068	0.0018	-0.015	0.02	0.022
four_g	0.016	0.013	-0.043	0.0032	-0.017	1	0.0087	-0.0018	-0.017	-0.03	-0.0056	-0.019	0.0074	0.0073	0.027	0.037	-0.047	0.58	0.017	-0.018	0.015
int_memory	-0.004	0.041	0.0065	-0.016	-0.029	0.0087	1	0.0069	-0.034	-0.028	-0.033	0.01	-0.0083	0.033	0.038	0.012	-0.0028	-0.0094	-0.027	0.007	0.044
m_dep	0.034	0.004	-0.014	-0.022	-0.0018	-0.0018	0.0069	1	0.022	-0.0035	0.026	0.025	0.024	-0.0094	-0.025	-0.018	0.017	-0.012	-0.0026	-0.028	0.00085
mobile_wt	0.0018	-0.0086	0.012	-0.009	0.024	-0.017	-0.034	0.022	1	-0.019	0.019	0.00094	9e-05	-0.0026	-0.034	-0.021	0.0062	0.0016	-0.014	-0.00041	-0.03
n_cores	-0.03	0.036	-0.0057	-0.025	-0.013	-0.03	-0.028	-0.0035	-0.019	1	-0.0012	-0.0069	0.024	0.0049	-0.00031	0.026	0.013	-0.015	0.024	-0.01	0.0044
pc	0.031	-0.01	-0.0052	-0.017	0.64	-0.0056	-0.033	0.026	0.019	-0.0012	1	-0.018	0.0042	0.029	0.0049	-0.024	0.015	-0.0013	-0.0087	0.0054	0.034
px_height	0.015	-0.0069	-0.015	-0.021	-0.01	-0.019	0.01	0.025	0.00094	-0.0069	-0.018	1	0.51	-0.02	0.06	0.043	-0.011	-0.031	0.022	0.052	0.15
px_width	-0.0084	-0.042	-0.0095	0.014	-0.0052	0.0074	-0.0083	0.024	9e-05	0.024	0.0042	0.51	1	0.0041	0.022	0.035	0.0067	0.00035	-0.0016	0.03	0.17
ram	-0.00065	0.026	0.0034	0.041	0.015	0.0073	0.033	-0.0094	-0.0026	0.0049	0.029	-0.02	0.0041	1	0.016	0.036	0.011	0.016	-0.03	0.023	0.92
sc_h	-0.03	-0.003	-0.029	-0.012	-0.011	0.027	0.038	-0.025	-0.034	-0.00031	0.0049	0.06	0.022	0.016	1	0.51	-0.017	0.012	-0.02	0.026	0.023
sc_w	-0.021	0.00061	-0.0074	-0.017	-0.012	0.037	0.012	-0.018	-0.021	0.026	-0.024	0.043	0.035	0.036	0.51	1	-0.023	0.031	0.013	0.035	0.039
talk_time	0.053	0.014	-0.011	-0.039	-0.0068	-0.047	-0.0028	0.017	0.0062	0.013	0.015	-0.011	0.0067	0.011	-0.017	-0.023	1	-0.043	0.017	-0.03	0.022
three_g	0.012	-0.03	-0.046	-0.014	0.0018	0.58	-0.0094	-0.012	0.0016	-0.015	-0.0013	-0.031	0.00035	0.016	0.012	0.031	-0.043	1	0.014	0.0043	0.024
touch_screen	-0.011	0.01	0.02	-0.017	-0.015	0.017	-0.027	-0.0026	-0.014	0.024	-0.0087	0.022	-0.0016	-0.03	-0.02	0.013	0.017	0.014	1	0.012	-0.03
wifi	-0.0083	-0.022	-0.024	0.023	0.02	-0.018	0.007	-0.028	-0.00041	-0.01	0.0054	0.052	0.03	0.023	0.026	0.035	-0.03	0.0043	0.012	1	0.019
price_range	0.2	0.021	-0.0066	0.017	0.022	0.015	0.044	0.00085	-0.03	0.0044	0.034	0.15	0.17	0.92	0.023	0.039	0.022	0.024	-0.03	0.019	1



Decision tree

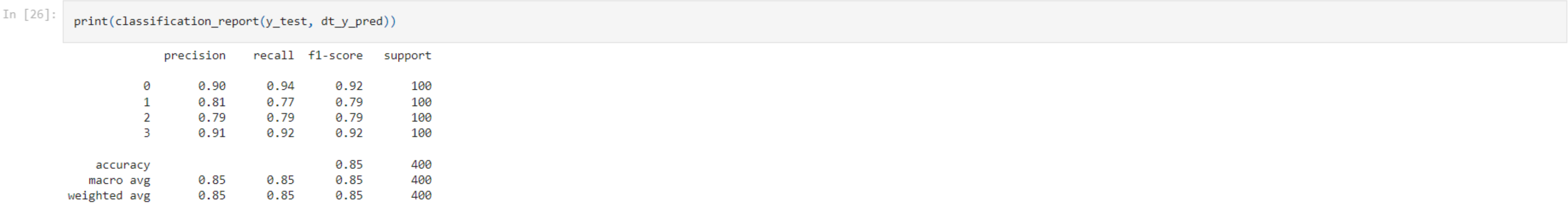
```
In [24]:
tree = DecisionTreeClassifier(criterion='gini',
                             min_samples_leaf=5,
                             min_samples_split=5,
                             max_depth=None,
                             random_state=seed)
tree.fit(x_train, y_train)
dt_y_pred = tree.predict(x_test)
dt_accuracy = accuracy_score(y_test, dt_y_pred)

print('DecisionTreeClassifier accuracy score: {}'.format(dt_accuracy))
```

```
DecisionTreeClassifier accuracy score: 0.855
```



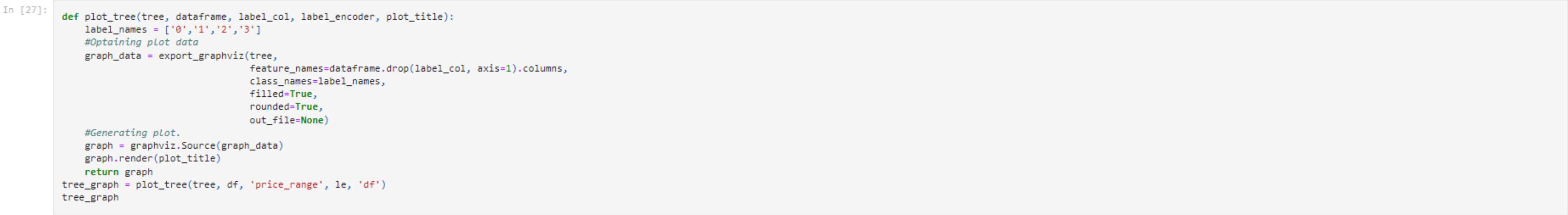
```
In [25]:
print('Confusion Matrix for decesion tree is')
print(confusion_matrix(y_test, dt_y_pred))
cm=confusion_matrix(y_test, dt_y_pred)
plt.matshow(cm)
plt.show()
```

```
Confusion Matrix for decesion tree is
[[94  6  0  0]
 [10 77 13  0]
 [ 0 12 79  9]
 [ 0  0  8 92]]
```

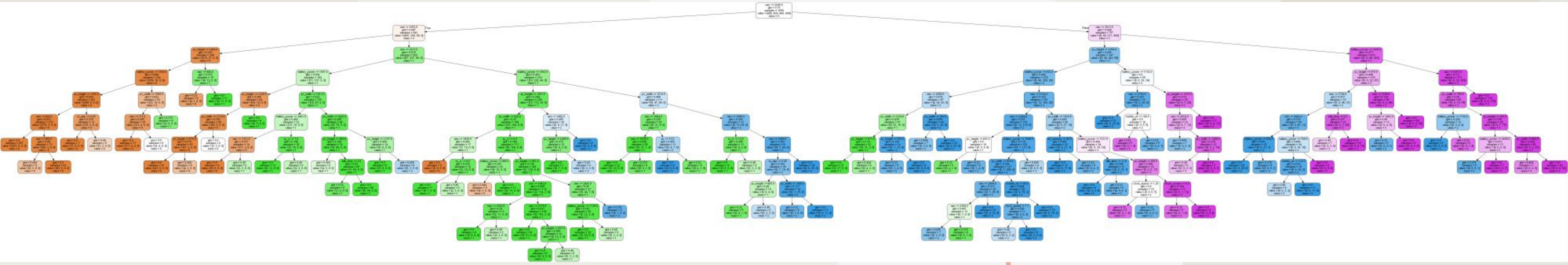


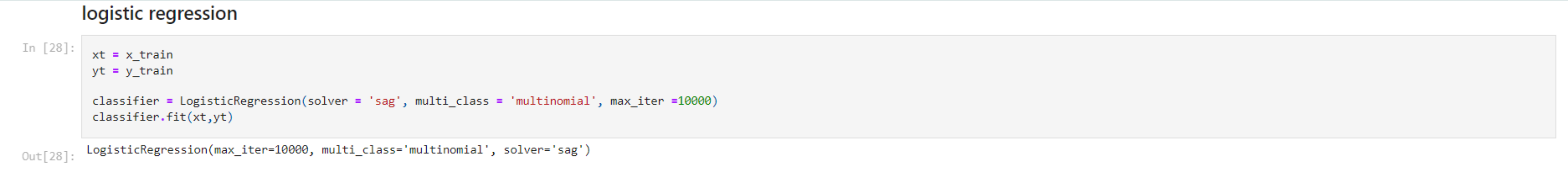
```
In [26]: print(classification_report(y_test, dt_y_pred))
```

```
              precision    recall  f1-score   support

           0       0.90      0.94      0.92       100
           1       0.81      0.77      0.79       100
           2       0.79      0.79      0.79       100
           3       0.91      0.92      0.92       100

    accuracy                           0.85       400
   macro avg       0.85      0.85      0.85       400
weighted avg       0.85      0.85      0.85       400
```



```
In [27]: def plot_tree(tree, dataframe, label_col, label_encoder, plot_title):
             label_names = ['0','1','2','3']
             #Optaining plot data
             graph_data = export_graphviz(tree,
                                          feature_names=dataframe.drop(label_col, axis=1).columns,
                                          class_names=label_names,
                                          filled=True,
                                          rounded=True,
                                          out_file=None)
             #Generating plot.
             graph = graphviz.Source(graph_data)
             graph.render(plot_title)
             return graph
         tree_graph = plot_tree(tree, df, 'price_range', le, 'df')
         tree_graph
```





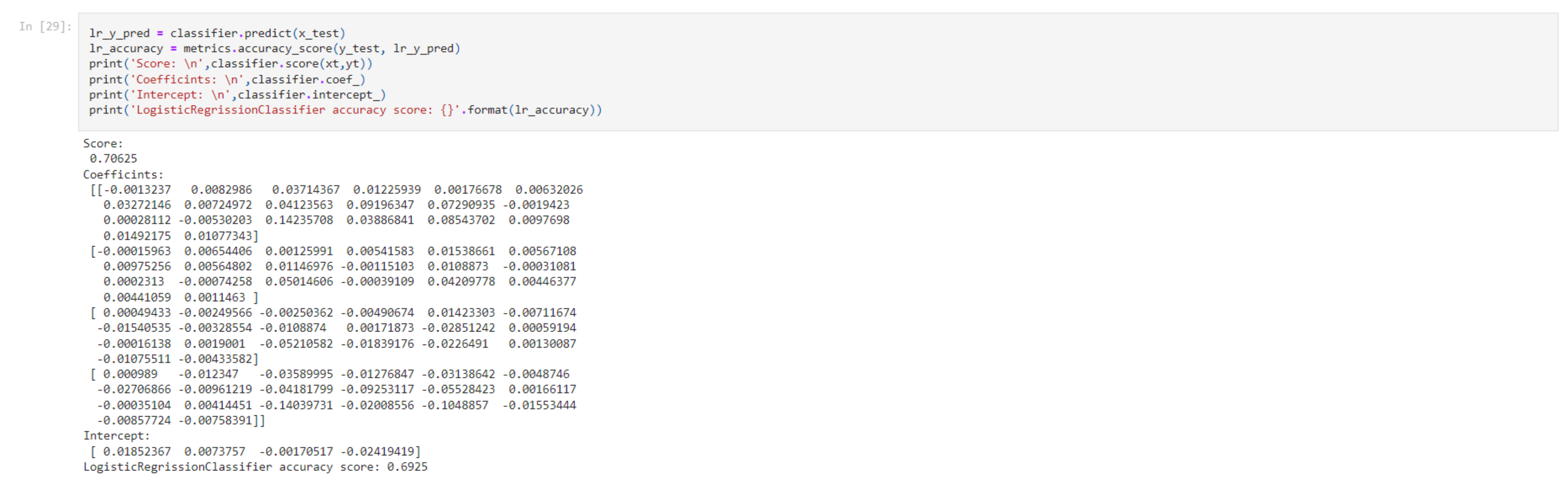
logistic regression

In [28]:

```
xt = x_train
yt = y_train

classifier = LogisticRegression(solver = 'sag', multi_class = 'multinomial', max_iter =10000)
classifier.fit(xt,yt)
```

Out[28]:

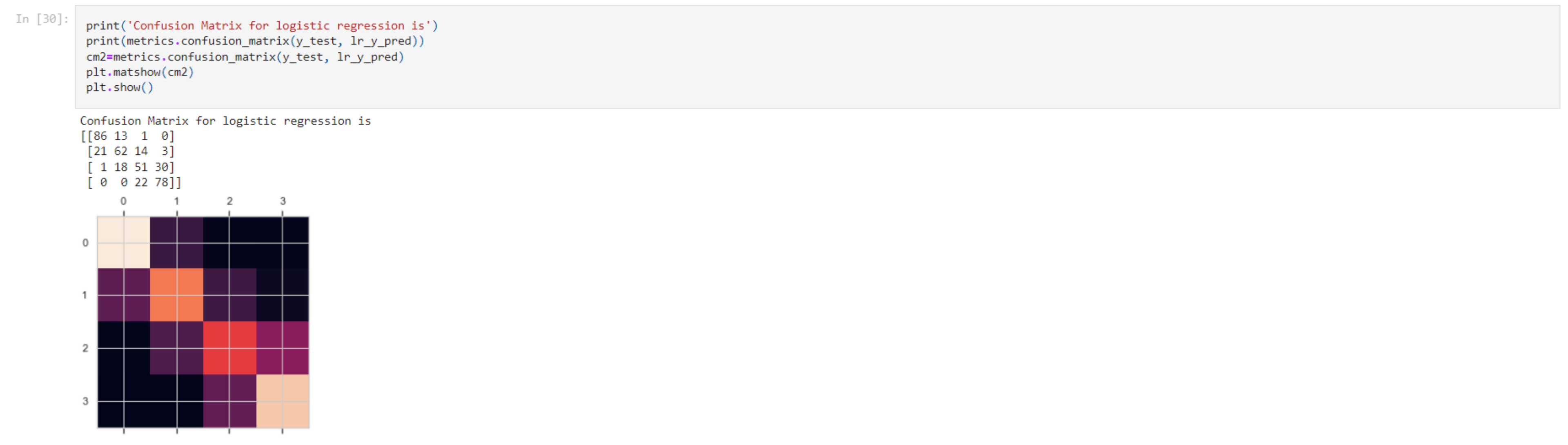
```
LogisticRegression(max_iter=10000, multi_class='multinomial', solver='sag')
```



In [29]:

```
lr_y_pred = classifier.predict(x_test)
lr_accuracy = metrics.accuracy_score(y_test, lr_y_pred)
print('Score: \n',classifier.score(xt,yt))
print('Coefficints: \n',classifier.coef_)
print('Intercept: \n',classifier.intercept_)
print('LogisticRegrissionClassifier accuracy score: {}'.format(lr_accuracy))
```

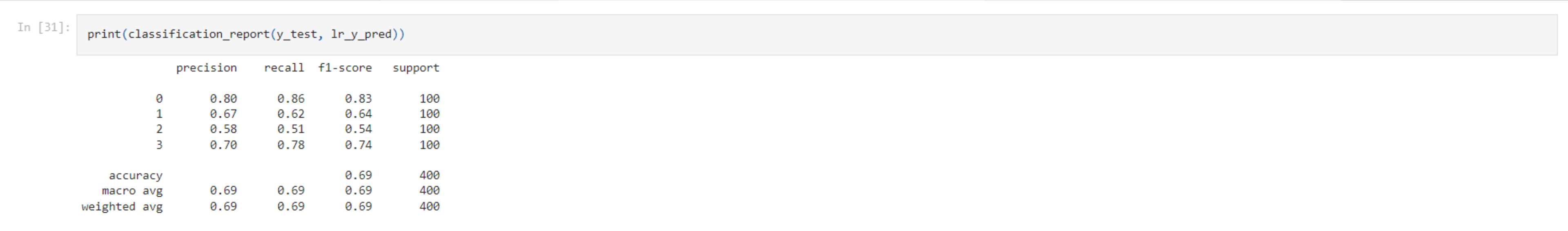
```
Score: 
 0.70625
Coefficints: 
 [[-0.0013237   0.0082986   0.03714367  0.01225939  0.00176678  0.00632026
   0.03272146  0.00724972  0.04123563  0.09196347  0.07290935 -0.0019423
   0.00028112 -0.00530203  0.14235708  0.03886841  0.08543702  0.0097698
   0.01492175  0.01077343]
 [-0.00015963  0.00654406  0.00125991  0.00541583  0.01538661  0.00567108
   0.00975256  0.00564802  0.01146976 -0.00115103  0.0108873  -0.00031081
   0.0002313  -0.00074258  0.05014606 -0.00039109  0.04209778  0.00446377
   0.00441059  0.0011463 ]
 [ 0.00049433 -0.00249566 -0.00250362 -0.00490674  0.01423303 -0.00711674
  -0.01540535 -0.00328554 -0.0108874   0.00171873 -0.02851242  0.00059194
  -0.00016138  0.0019001  -0.05210582 -0.01839176 -0.0226491   0.00130087
  -0.01075511 -0.00433582]
 [ 0.000989   -0.012347   -0.03589995 -0.01276847 -0.03138642 -0.0048746
  -0.02706866 -0.00961219 -0.04181799 -0.09253117 -0.05528423  0.00166117
  -0.00035104  0.00414451 -0.14039731 -0.02008556 -0.1048857  -0.01553444
  -0.00857724 -0.00758391]]
Intercept: 
 [ 0.01852367  0.0073757  -0.00170517 -0.02419419]
LogisticRegrissionClassifier accuracy score: 0.6925
```



In [30]:

```
print('Confusion Matrix for logistic regression is')
print(metrics.confusion_matrix(y_test, lr_y_pred))
cm2=metrics.confusion_matrix(y_test, lr_y_pred)
plt.matshow(cm2)
plt.show()
```

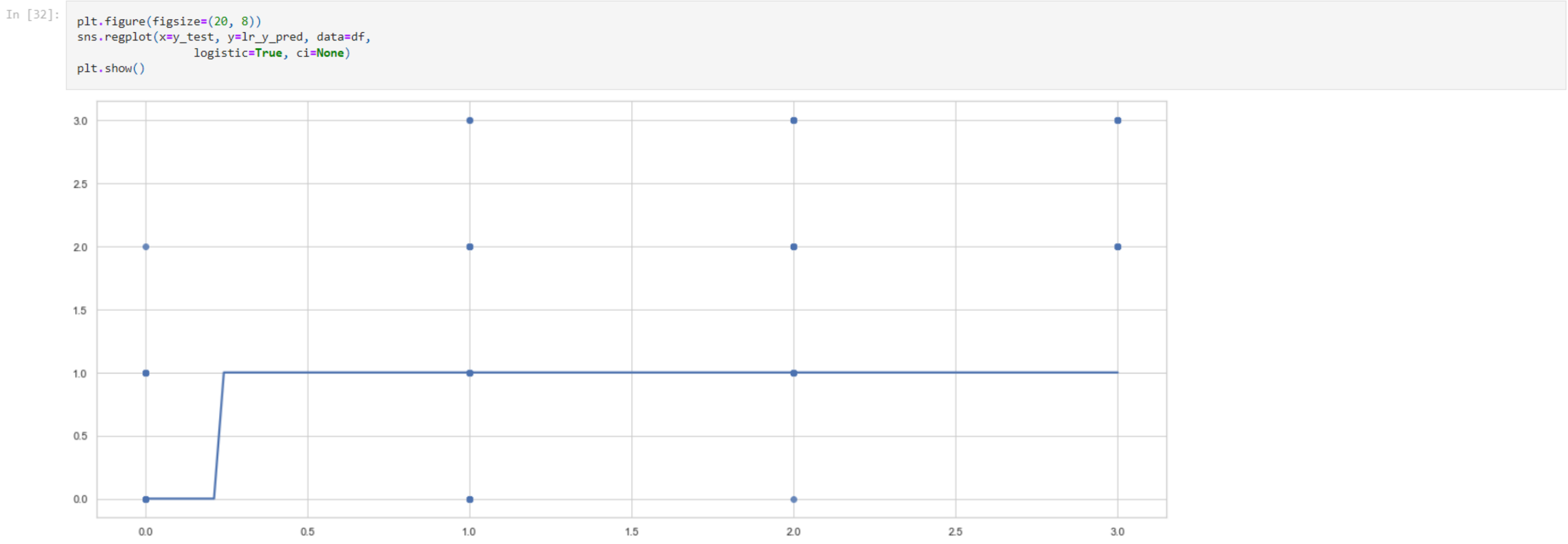
```
Confusion Matrix for logistic regression is
[[86 13  1  0]
 [21 62 14  3]
 [ 1 18 51 30]
 [ 0  0 22 78]]
```



In [31]:

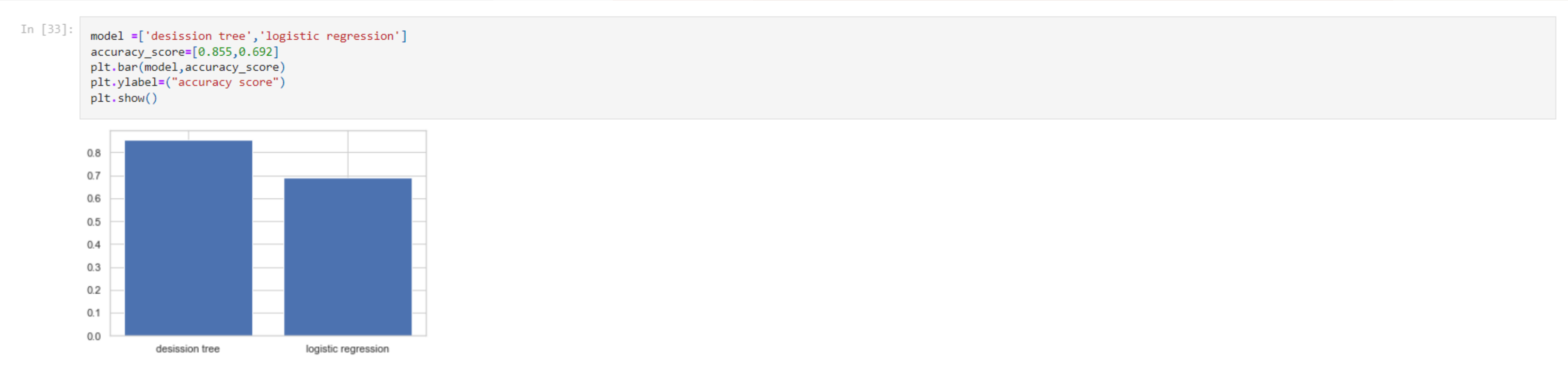
```
print(classification_report(y_test, lr_y_pred))
```

```
              precision    recall  f1-score   support

           0       0.80      0.86      0.83       100
           1       0.67      0.62      0.64       100
           2       0.58      0.51      0.54       100
           3       0.70      0.78      0.74       100

    accuracy                           0.69       400
   macro avg       0.69      0.69      0.69       400
weighted avg       0.69      0.69      0.69       400
```

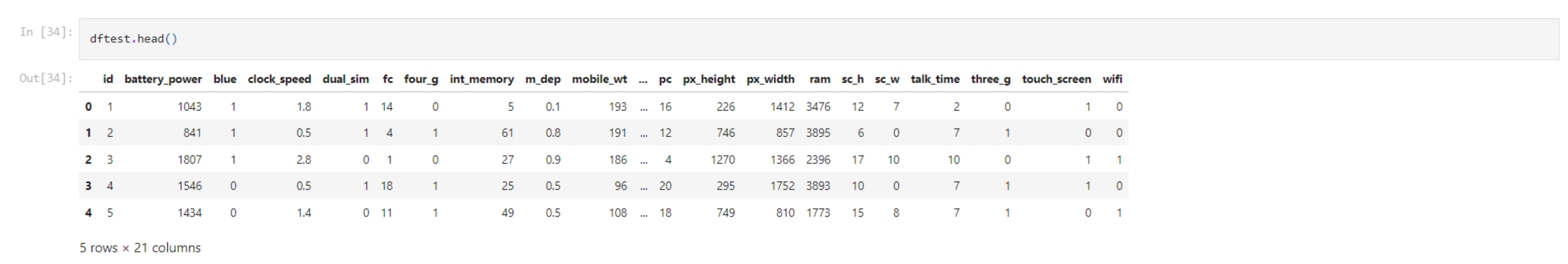


In [32]:

```
plt.figure(figsize=(20, 8))
sns.regplot(x=y_test, y=lr_y_pred, data=df,
                 logistic=True, ci=None)
plt.show()
```



```
In [33]: model =['desission tree','logistic regression']
         accuracy_score=[0.855,0.692]
         plt.bar(model,accuracy_score)
         plt.ylabel=("accuracy score")
         plt.show()
```

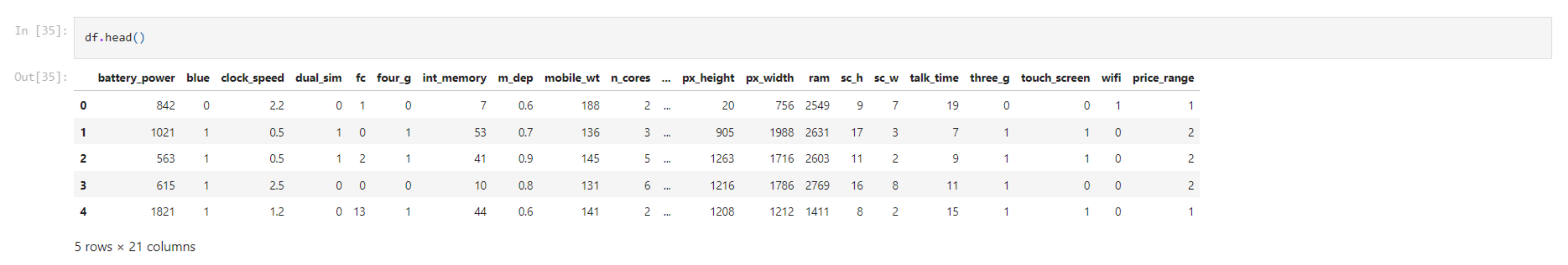


```
In [34]: dftest.head()
```

Out[34]:

	id	battery_power	blue	clock_speed	dual_sim	fc	four_g	int_memory	m_dep	mobile_wt	...	pc	px_height	px_width	ram	sc_h	sc_w	talk_time	three_g	touch_screen	wifi
0	1	1043	1	1.8	1	14	0	5	0.1	193	...	16	226	1412	3476	12	7	2	0	1	0
1	2	841	1	0.5	1	4	1	61	0.8	191	...	12	746	857	3895	6	0	7	1	0	0
2	3	1807	1	2.8	0	1	0	27	0.9	186	...	4	1270	1366	2396	17	10	10	0	1	1
3	4	1546	0	0.5	1	18	1	25	0.5	96	...	20	295	1752	3893	10	0	7	1	1	0
4	5	1434	0	1.4	0	11	1	49	0.5	108	...	18	749	810	1773	15	8	7	1	0	1

5 rows × 21 columns

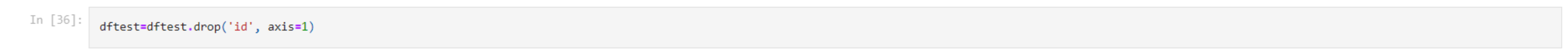


```
In [35]: df.head()
```

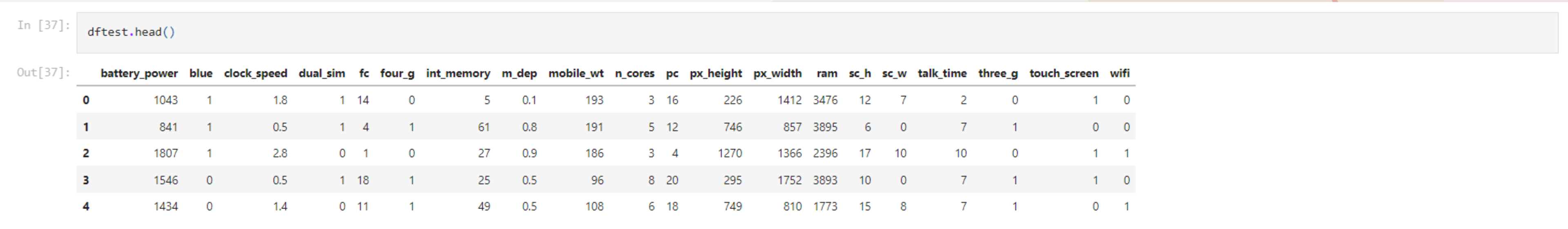
Out[35]:

	battery_power	blue	clock_speed	dual_sim	fc	four_g	int_memory	m_dep	mobile_wt	n_cores	...	px_height	px_width	ram	sc_h	sc_w	talk_time	three_g	touch_screen	wifi	price_range
0	842	0	2.2	0	1	0	7	0.6	188	2	...	20	756	2549	9	7	19	0	0	1	1
1	1021	1	0.5	1	0	1	53	0.7	136	3	...	905	1988	2631	17	3	7	1	1	0	2
2	563	1	0.5	1	2	1	41	0.9	145	5	...	1263	1716	2603	11	2	9	1	1	0	2
3	615	1	2.5	0	0	0	10	0.8	131	6	...	1216	1786	2769	16	8	11	1	0	0	2
4	1821	1	1.2	0	13	1	44	0.6	141	2	...	1208	1212	1411	8	2	15	1	1	0	1

5 rows × 21 columns



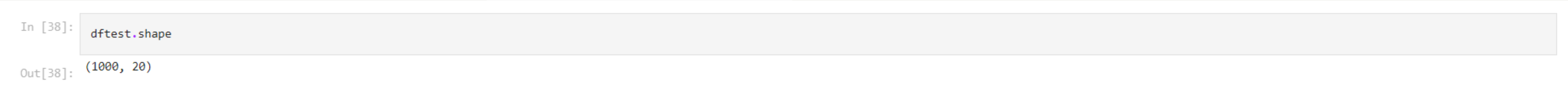
```
In [36]: dftest=dftest.drop('id', axis=1)
```



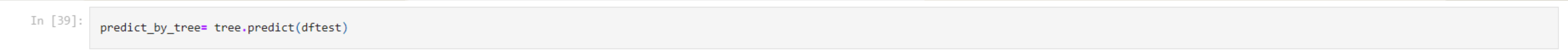
```
In [37]: dftest.head()
```

Out[37]:

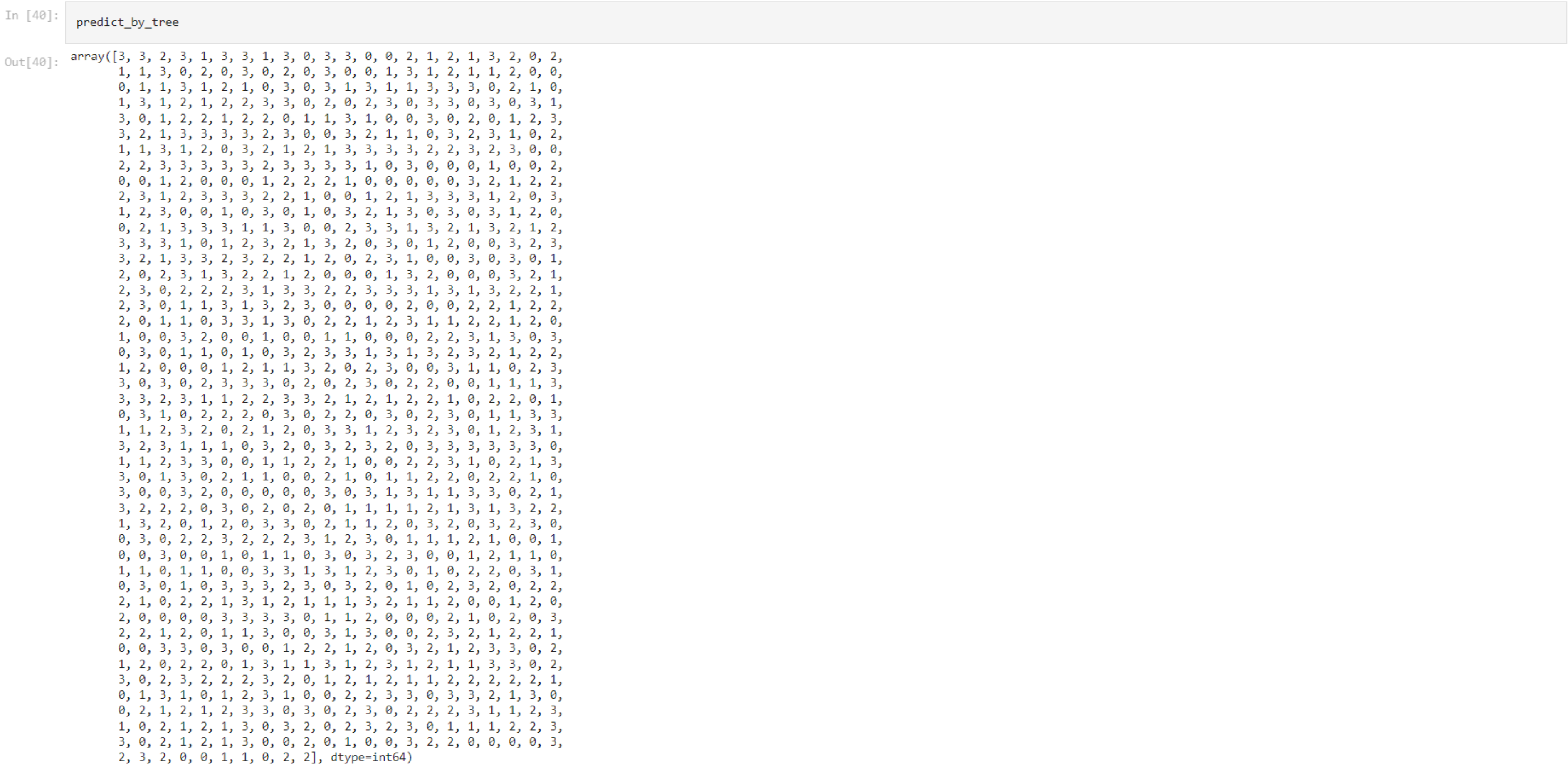
	battery_power	blue	clock_speed	dual_sim	fc	four_g	int_memory	m_dep	mobile_wt	n_cores	pc	px_height	px_width	ram	sc_h	sc_w	talk_time	three_g	touch_screen	wifi
0	1043	1	1.8	1	14	0	5	0.1	193	3	16	226	1412	3476	12	7	2	0	1	0
1	841	1	0.5	1	4	1	61	0.8	191	5	12	746	857	3895	6	0	7	1	0	0
2	1807	1	2.8	0	1	0	27	0.9	186	3	4	1270	1366	2396	17	10	10	0	1	1
3	1546	0	0.5	1	18	1	25	0.5	96	8	20	295	1752	3893	10	0	7	1	1	0
4	1434	0	1.4	0	11	1	49	0.5	108	6	18	749	810	1773	15	8	7	1	0	1



```
In [38]: dftest.shape

Out[38]: (1000, 20)
```

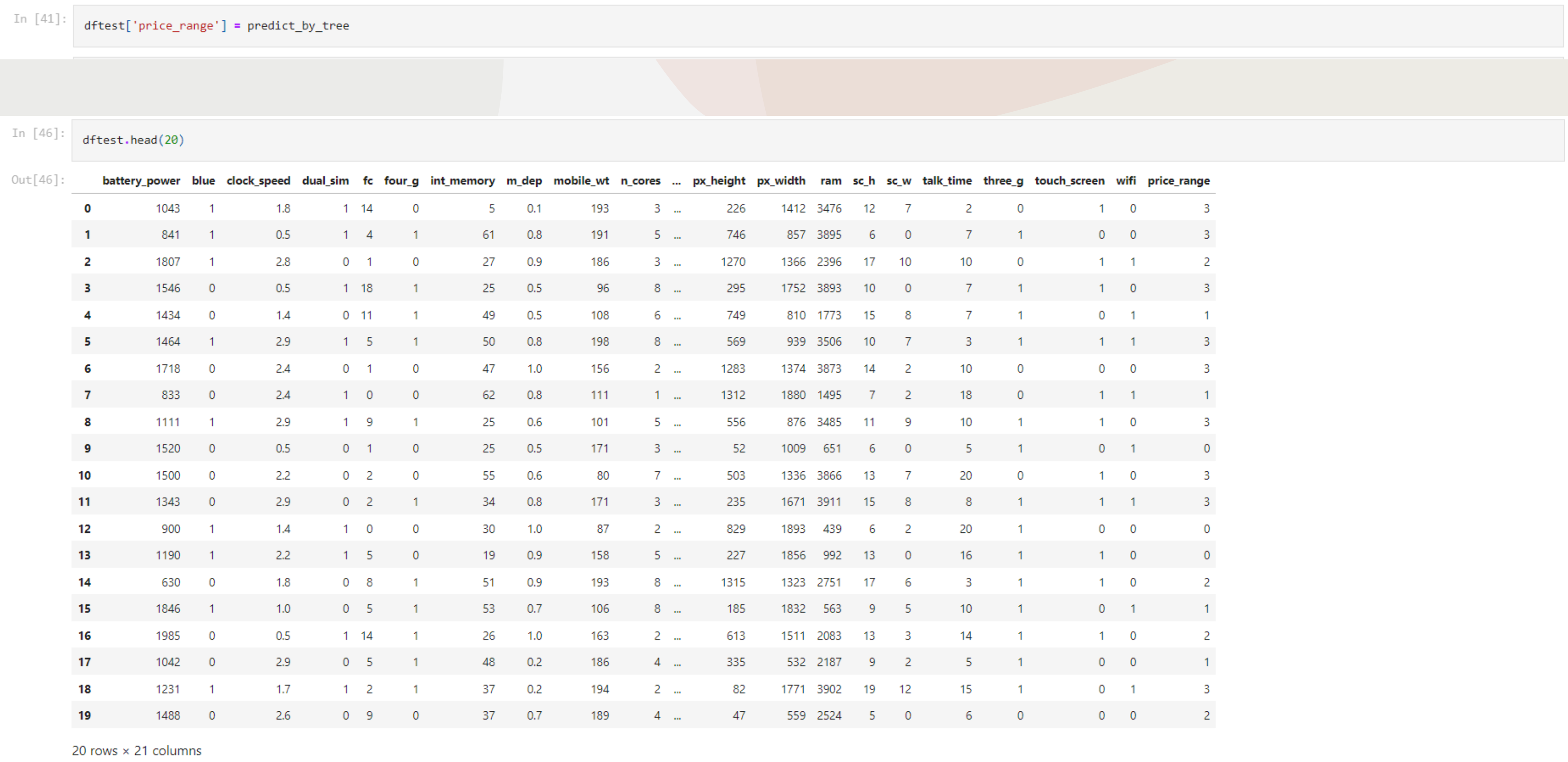


```
In [39]: predict_by_tree= tree.predict(dftest)
```



```
In [40]: predict_by_tree
```

```
Out[40]: array([3, 3, 2, 3, 1, 3, 3, 1, 3, 0, 3, 3, 0, 0, 2, 1, 2, 1, 3, 2, 0, 2,
                1, 1, 3, 0, 2, 0, 3, 0, 2, 0, 3, 0, 0, 1, 3, 1, 2, 1, 1, 2, 0, 0,
                0, 1, 1, 3, 1, 2, 1, 0, 3, 0, 3, 1, 3, 1, 1, 3, 3, 3, 0, 2, 1, 0,
                1, 3, 1, 2, 1, 2, 2, 3, 3, 0, 2, 0, 2, 3, 0, 3, 3, 0, 3, 0, 3, 1,
                3, 0, 1, 2, 2, 1, 2, 2, 0, 1, 1, 3, 1, 0, 0, 3, 0, 2, 0, 1, 2, 3,
                3, 2, 1, 3, 3, 3, 3, 2, 3, 0, 0, 3, 2, 1, 1, 0, 3, 2, 3, 1, 0, 2,
                1, 1, 3, 1, 2, 0, 3, 2, 1, 2, 1, 3, 3, 3, 3, 2, 2, 3, 2, 3, 0, 0,
                2, 2, 3, 3, 3, 3, 3, 2, 3, 3, 3, 3, 1, 0, 3, 0, 0, 0, 1, 0, 0, 2,
                0, 0, 1, 2, 0, 0, 0, 1, 2, 2, 2, 1, 0, 0, 0, 0, 0, 3, 2, 1, 2, 2,
                2, 3, 1, 2, 3, 3, 3, 2, 2, 1, 0, 0, 1, 2, 1, 3, 3, 3, 1, 2, 0, 3,
                1, 2, 3, 0, 0, 1, 0, 3, 0, 1, 0, 3, 2, 1, 3, 0, 3, 0, 3, 1, 2, 0,
                0, 2, 1, 3, 3, 3, 1, 1, 3, 0, 0, 2, 3, 3, 1, 3, 2, 1, 3, 2, 1, 2,
                3, 3, 3, 1, 0, 1, 2, 3, 2, 1, 3, 2, 0, 3, 0, 1, 2, 0, 0, 3, 2, 3,
                3, 2, 1, 3, 3, 2, 3, 2, 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, 1,
                2, 3, 0, 2, 2, 2, 3, 1, 3, 3, 2, 2, 3, 3, 3, 1, 3, 1, 3, 2, 2, 1,
                2, 3, 0, 1, 1, 3, 1, 3, 2, 3, 0, 0, 0, 0, 2, 0, 0, 2, 2, 1, 2, 2,
                2, 0, 1, 1, 0, 3, 3, 1, 3, 0, 2, 2, 1, 2, 3, 1, 1, 2, 2, 1, 2, 0,
                1, 0, 0, 3, 2, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 2, 2, 3, 1, 3, 0, 3,
                0, 3, 0, 1, 1, 0, 1, 0, 3, 2, 3, 3, 1, 3, 1, 3, 2, 3, 2, 1, 2, 2,
                1, 2, 0, 0, 0, 1, 2, 1, 1, 3, 2, 0, 2, 3, 0, 0, 3, 1, 1, 0, 2, 3,
                3, 0, 3, 0, 2, 3, 3, 3, 0, 2, 0, 2, 3, 0, 2, 2, 0, 0, 1, 1, 1, 3,
                3, 3, 2, 3, 1, 1, 2, 2, 3, 3, 2, 1, 2, 1, 2, 2, 1, 0, 2, 2, 0, 1,
                0, 3, 1, 0, 2, 2, 2, 0, 3, 0, 2, 2, 0, 3, 0, 2, 3, 0, 1, 1, 3, 3,
                1, 1, 2, 3, 2, 0, 2, 1, 2, 0, 3, 3, 1, 2, 3, 2, 3, 0, 1, 2, 3, 1,
                3, 2, 3, 1, 1, 1, 0, 3, 2, 0, 3, 2, 3, 2, 0, 3, 3, 3, 3, 3, 3, 0,
                1, 1, 2, 3, 3, 0, 0, 1, 1, 2, 2, 1, 0, 0, 2, 2, 3, 1, 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, 1, 1, 3, 3, 0, 2, 1,
                3, 2, 2, 2, 0, 3, 0, 2, 0, 2, 0, 1, 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, 3, 1, 2, 3, 0, 1, 1, 1, 2, 1, 0, 0, 1,
                0, 0, 3, 0, 0, 1, 0, 1, 1, 0, 3, 0, 3, 2, 3, 0, 0, 1, 2, 1, 1, 0,
                1, 1, 0, 1, 1, 0, 0, 3, 3, 1, 3, 1, 2, 3, 0, 1, 0, 2, 2, 0, 3, 1,
                0, 3, 0, 1, 0, 3, 3, 3, 2, 3, 0, 3, 2, 0, 1, 0, 2, 3, 2, 0, 2, 2,
                2, 1, 0, 2, 2, 1, 3, 1, 2, 1, 1, 1, 3, 2, 1, 1, 2, 0, 0, 1, 2, 0,
                2, 0, 0, 0, 0, 3, 3, 3, 3, 0, 1, 1, 2, 0, 0, 0, 2, 1, 0, 2, 0, 3,
                2, 2, 1, 2, 0, 1, 1, 3, 0, 0, 3, 1, 3, 0, 0, 2, 3, 2, 1, 2, 2, 1,
                0, 0, 3, 3, 0, 3, 0, 0, 1, 2, 2, 1, 2, 0, 3, 2, 1, 2, 3, 3, 0, 2,
                1, 2, 0, 2, 2, 0, 1, 3, 1, 1, 3, 1, 2, 3, 1, 2, 1, 1, 3, 3, 0, 2,
                3, 0, 2, 3, 2, 2, 2, 3, 2, 0, 1, 2, 1, 2, 1, 1, 2, 2, 2, 2, 2, 1,
                0, 1, 3, 1, 0, 1, 2, 3, 1, 0, 0, 2, 2, 3, 3, 0, 3, 3, 2, 1, 3, 0,
                0, 2, 1, 2, 1, 2, 3, 3, 0, 3, 0, 2, 3, 0, 2, 2, 2, 3, 1, 1, 2, 3,
                1, 0, 2, 1, 2, 1, 3, 0, 3, 2, 0, 2, 3, 2, 3, 0, 1, 1, 1, 2, 2, 3,
                3, 0, 2, 1, 2, 1, 3, 0, 0, 2, 0, 1, 0, 0, 3, 2, 2, 0, 0, 0, 0, 3,
                2, 3, 2, 0, 0, 1, 1, 0, 2, 2], dtype=int64)
```



In [41]:

```
dftest['price_range'] = predict_by_tree
```

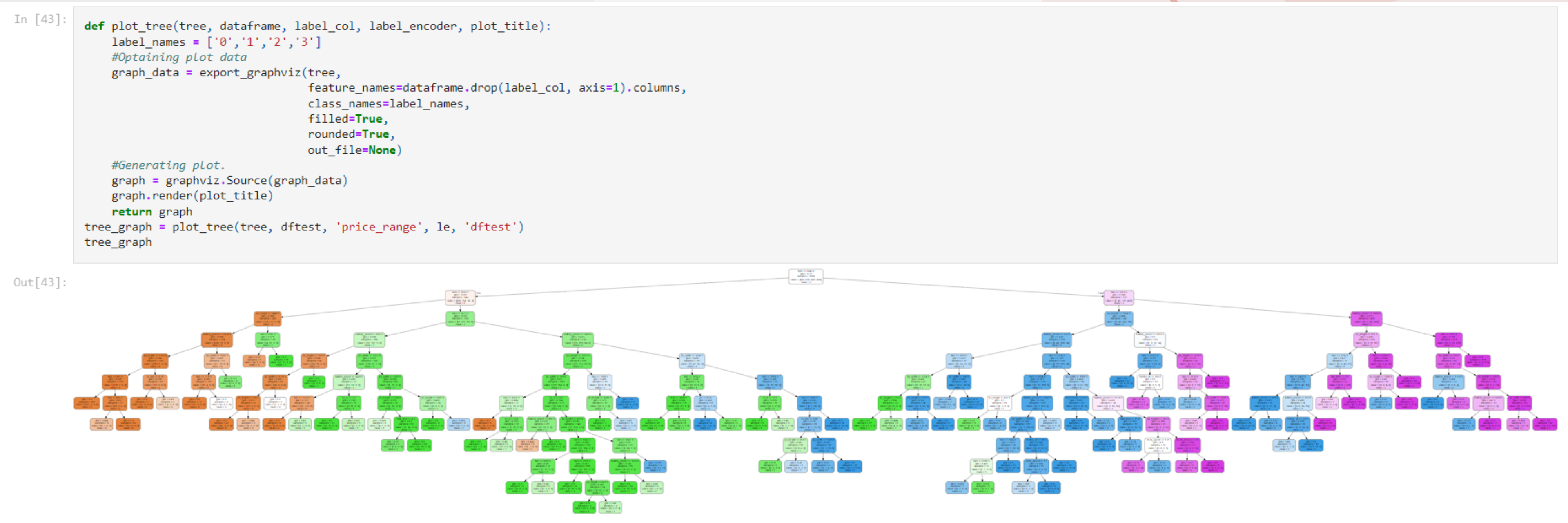
In [46]:

```
dftest.head(20)
```

Out[46]:

	battery_power	blue	clock_speed	dual_sim	fc	four_g	int_memory	m_dep	mobile_wt	n_cores	...	px_height	px_width	ram	sc_h	sc_w	talk_time	three_g	touch_screen	wifi	price_range
0	1043	1	1.8	1	14	0	5	0.1	193	3	...	226	1412	3476	12	7	2	0	1	0	3
1	841	1	0.5	1	4	1	61	0.8	191	5	...	746	857	3895	6	0	7	1	0	0	3
2	1807	1	2.8	0	1	0	27	0.9	186	3	...	1270	1366	2396	17	10	10	0	1	1	2
3	1546	0	0.5	1	18	1	25	0.5	96	8	...	295	1752	3893	10	0	7	1	1	0	3
4	1434	0	1.4	0	11	1	49	0.5	108	6	...	749	810	1773	15	8	7	1	0	1	1
5	1464	1	2.9	1	5	1	50	0.8	198	8	...	569	939	3506	10	7	3	1	1	1	3
6	1718	0	2.4	0	1	0	47	1.0	156	2	...	1283	1374	3873	14	2	10	0	0	0	3
7	833	0	2.4	1	0	0	62	0.8	111	1	...	1312	1880	1495	7	2	18	0	1	1	1
8	1111	1	2.9	1	9	1	25	0.6	101	5	...	556	876	3485	11	9	10	1	1	0	3
9	1520	0	0.5	0	1	0	25	0.5	171	3	...	52	1009	651	6	0	5	1	0	1	0
10	1500	0	2.2	0	2	0	55	0.6	80	7	...	503	1336	3866	13	7	20	0	1	0	3
11	1343	0	2.9	0	2	1	34	0.8	171	3	...	235	1671	3911	15	8	8	1	1	1	3
12	900	1	1.4	1	0	0	30	1.0	87	2	...	829	1893	439	6	2	20	1	0	0	0
13	1190	1	2.2	1	5	0	19	0.9	158	5	...	227	1856	992	13	0	16	1	1	0	0
14	630	0	1.8	0	8	1	51	0.9	193	8	...	1315	1323	2751	17	6	3	1	1	0	2
15	1846	1	1.0	0	5	1	53	0.7	106	8	...	185	1832	563	9	5	10	1	0	1	1
16	1985	0	0.5	1	14	1	26	1.0	163	2	...	613	1511	2083	13	3	14	1	1	0	2
17	1042	0	2.9	0	5	1	48	0.2	186	4	...	335	532	2187	9	2	5	1	0	0	1
18	1231	1	1.7	1	2	1	37	0.2	194	2	...	82	1771	3902	19	12	15	1	0	1	3
19	1488	0	2.6	0	9	0	37	0.7	189	4	...	47	559	2524	5	0	6	0	0	0	2

20 rows × 21 columns



In [43]:

```
def plot_tree(tree, dataframe, label_col, label_encoder, plot_title):
    label_names = ['0','1','2','3']
    #Optaining plot data
    graph_data = export_graphviz(tree,
                                 feature_names=dataframe.drop(label_col, axis=1).columns,
                                 class_names=label_names,
                                 filled=True,
                                 rounded=True,
                                 out_file=None)
    #Generating plot.
    graph = graphviz.Source(graph_data)
    graph.render(plot_title)
    return graph
tree_graph = plot_tree(tree, dftest, 'price_range', le, 'dftest')
tree_graph
```

Out[43]: