Mobile Price Classification

Bob has started his own mobile company. He wants to give tough fight to big companies like Apple,Samsung etc.

He does not know how to estimate price of mobiles his company creates. In this competitive mobile phone market you cannot simply assume things. To solve this problem he collects sales data of mobile phones of various companies.

Bob wants to find out some relation between features of a mobile phone(eg:- RAM,Internal Memory etc) and its selling price. But he is not so good at Machine Learning. So he needs your help to solve this problem.

In this problem you do not have to predict actual price but a price range indicating how high the price is

Let's Dive into it

Import necessary libraries

In [5]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

Read 'mobile_price.csv' dataset and store it in a DataFrame

In [6]:
```
df = pd.read_csv('mobile_price.csv')
```

View top 5 rows

In [7]:
```
pd.set_option('display.max_columns', None) # Used to view non-truncated columns
```

In [8]:
```
df.head()
```

Out[8]:

	battery_power	blue	clock_speed	dual_sim	fc	four_g	int_memory	m_dep	mobile_wt	n_co
0	842	0	2.2	0	1	0	7	0.6	188	
1	1021	1	0.5	1	0	1	53	0.7	136	
2	563	1	0.5	1	2	1	41	0.9	145	
3	615	1	2.5	0	0	0	10	0.8	131	
4	1821	1	1.2	0	13	1	44	0.6	141	

View info of the dataset

In [9]:
```
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
```

View basic statistical information about the dataset

In [10]:
```
df.describe()
```

Out[10]:

	battery_power	blue	clock_speed	dual_sim	fc	four_g	int_memory
count	2000.000000	2000.0000	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000
mean	1238.518500	0.4950	1.522250	0.509500	4.309500	0.521500	32.046500
std	439.418206	0.5001	0.816004	0.500035	4.341444	0.499662	18.145715
min	501.000000	0.0000	0.500000	0.000000	0.000000	0.000000	2.000000
25%	851.750000	0.0000	0.700000	0.000000	1.000000	0.000000	16.000000
50%	1226.000000	0.0000	1.500000	1.000000	3.000000	1.000000	32.000000
75%	1615.250000	1.0000	2.200000	1.000000	7.000000	1.000000	48.000000
max	1998.000000	1.0000	3.000000	1.000000	19.000000	1.000000	64.000000

Check for null values

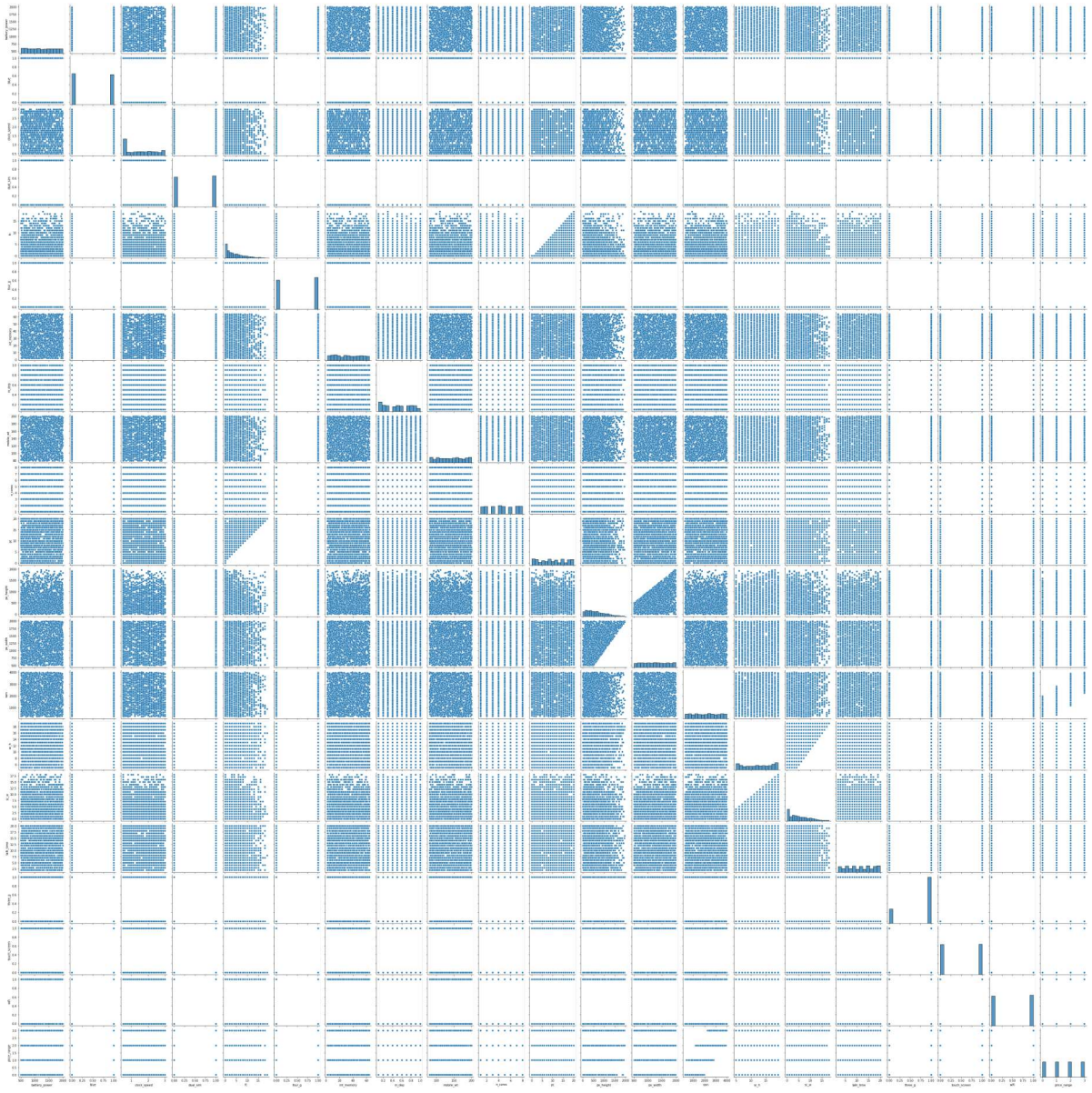
In [7]:
```
df.isna().sum()
```

Out[7]:
```
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
```

Visualization

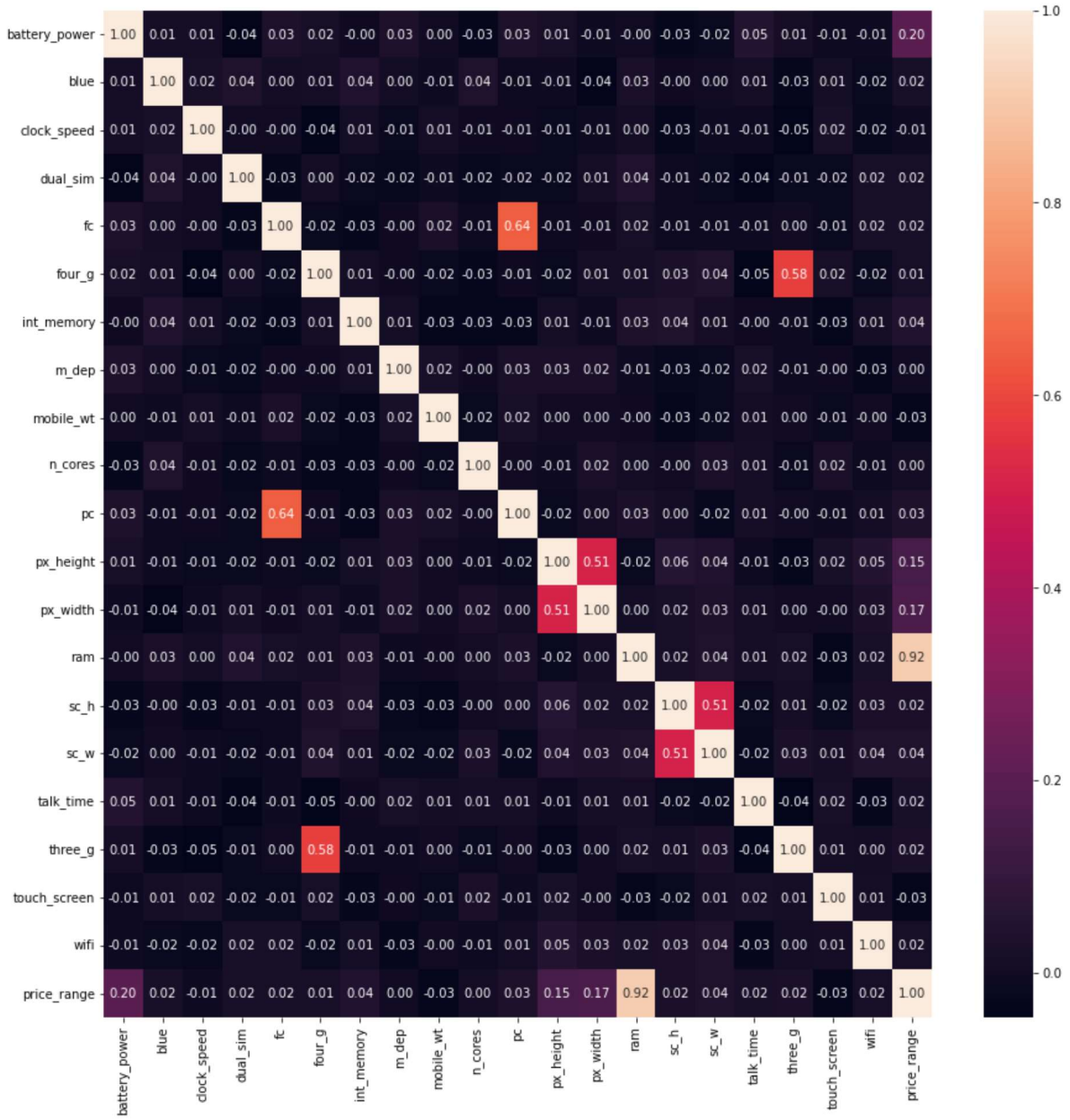
Plot a pairplot of the dataset

In [14]:
```
sns.pairplot(df)
plt.show()
```



Plot a heatmap to show the correlation between features

In [17]:
```
plt.figure(figsize = (15,15))
sns.heatmap(df.corr(),annot=True,fmt='.2f')
plt.show()
```



Split the data into input and target variables

In [11]:
```
X =  df.drop(columns = ['price_range'])
y = df['price_range']
```

Standardise the data with StandardScaler

In [12]:
```
from sklearn.preprocessing import StandardScaler
```

In [13]:
```
scaler = StandardScaler()
```

In [14]:
```
xcolumns = X.columns
```

In [15]:
```
X = scaler.fit_transform(X)
```

In [13]:
```
X = pd.DataFrame(X, columns = xcolumns)
```

In [14]:
```
X.head()
```

Out[14]:

	battery_power	blue	clock_speed	dual_sim	fc	four_g	int_memory	m_dep
0	-0.902597	-0.990050	0.830779	-1.019184	-0.762495	-1.043966	-1.380644	0.340740
1	-0.495139	1.010051	-1.253064	0.981177	-0.992890	0.957886	1.155024	0.687548
2	-1.537686	1.010051	-1.253064	0.981177	-0.532099	0.957886	0.493546	1.381165
3	-1.419319	1.010051	1.198517	-1.019184	-0.992890	-1.043966	-1.215274	1.034357
4	1.325906	1.010051	-0.395011	-1.019184	2.002254	0.957886	0.658915	0.340740

Split the dataset into training and testing set

In [21]:
```
from sklearn.model_selection import train_test_split
```

In [23]:
```
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size = 0.25, random_
```

In [24]:
```

```

In [24]:
```
X_train.shape
```

Out[24]:
```
(1500, 20)
```

In [25]:
```
X_test.shape
```

Out[25]:
```
(500, 20)
```

Create random forest model.

Let n_estimator = 100

In [26]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [34]:
```
model = RandomForestClassifier(n_estimators= 100)
```

Train the model

In [35]:
```
model.fit(X_train, y_train)
```

Out[35]:
```
RandomForestClassifier()
```

Check the score of our model

In [36]:
```
model.score(X_train, y_train)
```

Out[36]:
```
1.0
```

Predict using X_test

In [37]:
```
y_pred = model.predict(X_test)
y_pred
```

Out[37]:
```
array([0, 1, 1, 3, 2, 3, 3, 2, 3, 1, 0, 1, 3, 3, 0, 2, 2, 3, 2, 0, 0, 3,
       1, 0, 0, 1, 3, 0, 1, 2, 0, 3, 0, 1, 0, 3, 3, 1, 1, 3, 3, 3, 0, 0,
       3, 3, 2, 0, 1, 2, 3, 2, 3, 2, 1, 3, 1, 3, 1, 0, 3, 2, 3, 0, 2, 0,
       0, 3, 3, 2, 0, 1, 2, 0, 2, 3, 0, 3, 2, 3, 1, 0, 3, 0, 2, 2, 1, 2,
       3, 3, 2, 3, 3, 2, 0, 2, 3, 1, 1, 1, 0, 0, 3, 3, 2, 0, 0, 1, 1, 1,
       0, 1, 3, 2, 0, 3, 1, 3, 0, 1, 3, 1, 0, 1, 3, 2, 3, 2, 0, 3, 1, 0,
       1, 2, 3, 1, 3, 2, 0, 0, 3, 1, 3, 3, 1, 2, 2, 2, 0, 2, 0, 3, 1, 2,
       1, 0, 2, 3, 0, 1, 3, 2, 3, 0, 1, 3, 0, 3, 3, 0, 3, 1, 2, 1, 1, 1,
       2, 1, 3, 1, 0, 2, 1, 0, 0, 3, 1, 2, 3, 2, 2, 1, 2, 1, 2, 2, 3, 1,
       2, 1, 3, 1, 2, 1, 3, 0, 1, 3, 3, 2, 0, 1, 2, 0, 0, 2, 3, 0, 1, 3,
       0, 3, 0, 0, 1, 1, 2, 3, 2, 0, 1, 2, 0, 1, 1, 3, 3, 3, 2, 0, 1, 3,
       2, 1, 0, 3, 2, 1, 2, 1, 1, 2, 3, 1, 3, 2, 1, 1, 2, 3, 1, 3, 2, 2,
       2, 2, 3, 1, 2, 1, 0, 3, 2, 0, 2, 1, 1, 3, 2, 2, 2, 1, 3, 1, 0, 2,
       3, 0, 0, 1, 2, 2, 1, 2, 2, 0, 2, 3, 0, 3, 3, 3, 0, 0, 3, 0, 3, 0,
       0, 0, 3, 0, 1, 2, 0, 1, 1, 3, 1, 2, 3, 3, 0, 0, 0, 0, 0, 2, 1, 1,
       3, 1, 0, 0, 1, 2, 3, 1, 2, 1, 0, 2, 0, 0, 3, 2, 1, 1, 0, 2, 0, 3,
       3, 1, 2, 3, 0, 0, 1, 2, 3, 3, 1, 3, 0, 3, 1, 2, 0, 0, 2, 0, 0, 2,
       2, 3, 0, 1, 1, 3, 0, 3, 0, 1, 0, 1, 0, 2, 2, 0, 1, 3, 0, 1, 0, 3,
       1, 3, 0, 0, 0, 0, 3, 0, 0, 3, 3, 1, 1, 3, 0, 0, 2, 1, 0, 0, 1, 2,
       0, 0, 3, 0, 0, 0, 2, 1, 1, 1, 0, 3, 0, 0, 2, 3, 1, 0, 2, 1, 3, 1,
       0, 0, 0, 0, 2, 0, 1, 1, 0, 3, 3, 2, 2, 3, 0, 3, 2, 3, 3, 0, 0, 2,
       0, 1, 1, 0, 0, 2, 0, 2, 3, 0, 1, 1, 1, 1, 2, 1, 2, 1, 0, 2, 3, 2,
       2, 1, 0, 3, 1, 1, 0, 2, 2, 2, 0, 1, 1, 2, 2, 2], dtype=int64)
```

Check the accuracy score of our prediction

In [38]:
```
from sklearn import metrics
```

In [39]:
```
metrics.accuracy_score(y_test, y_pred)
```

Out[39]:
```
0.892
```

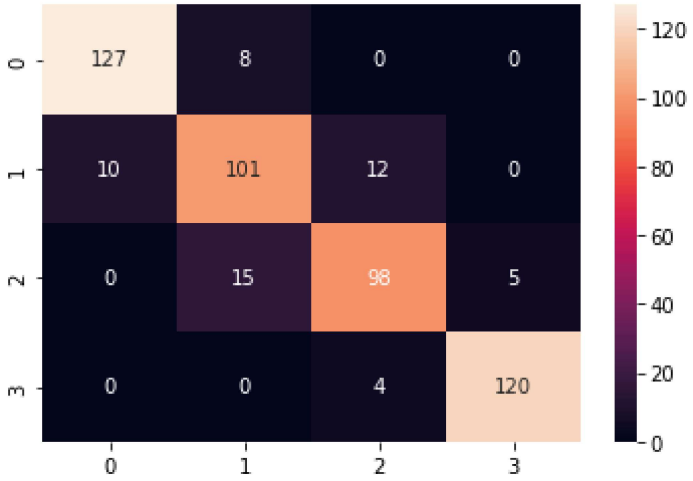
Create a confusion matrix

In [40]:
```
metrics.confusion_matrix(y_test, y_pred)
```

Out[40]:
```
array([[127,   8,   0,   0],
       [ 10, 101,  12,   0],
       [  0,  15,  98,   5],
       [  0,   0,   4, 120]], dtype=int64)
```

Plot confusion matrix on heatmap

In [41]:
```
sns.heatmap(metrics.confusion_matrix(y_test,y_pred), annot = True, fmt = 'd')
plt.show()
```



Create classification report

In [42]:
```
print(metrics.classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.93      0.94      0.93       135
           1       0.81      0.82      0.82       123
           2       0.86      0.83      0.84       118
           3       0.96      0.97      0.96       124

    accuracy                           0.89       500
   macro avg       0.89      0.89      0.89       500
weighted avg       0.89      0.89      0.89       500
```