```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")
from google.colab import files
uploaded = files.upload()
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving test.csv to test.csv
Saving train.csv to train.csv
```

```
#reading train file
dftrain=pd.read_csv("train.csv")
#reading test file
dftest = pd.read_csv('test.csv')
```

```
dftrain.head(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	t
0	842	0	2.2	0	1	0	7	0.6	188	2	...	20	756	2549	9	7	
1	1021	1	0.5	1	0	1	53	0.7	136	3	...	905	1988	2631	17	3	
2	563	1	0.5	1	2	1	41	0.9	145	5	...	1263	1716	2603	11	2	
3	615	1	2.5	0	0	0	10	0.8	131	6	...	1216	1786	2769	16	8	
4	1821	1	1.2	0	13	1	44	0.6	141	2	...	1208	1212	1411	8	2	
5	1859	0	0.5	1	3	0	22	0.7	164	1	...	1004	1654	1067	17	1	
6	1821	0	1.7	0	4	1	10	0.8	139	8	...	381	1018	3220	13	8	
7	1954	0	0.5	1	0	0	24	0.8	187	4	...	512	1149	700	16	3	
8	1445	1	0.5	0	0	0	53	0.7	174	7	...	386	836	1099	17	1	
9	509	1	0.6	1	2	1	9	0.1	93	5	...	1137	1224	513	19	10	
10	769	1	2.9	1	0	0	9	0.1	182	5	...	248	874	3946	5	2	
11	1520	1	2.2	0	5	1	33	0.5	177	8	...	151	1005	3826	14	9	
12	1815	0	2.8	0	2	0	33	0.6	159	4	...	607	748	1482	18	0	
13	803	1	2.1	0	7	0	17	1.0	198	4	...	344	1440	2680	7	1	
14	1866	0	0.5	0	13	1	52	0.7	185	1	...	356	563	373	14	9	
15	775	0	1.0	0	3	0	46	0.7	159	2	...	862	1864	568	17	15	
16	838	0	0.5	0	1	1	13	0.1	196	8	...	984	1850	3554	10	9	
17	595	0	0.9	1	7	1	23	0.1	121	3	...	441	810	3752	10	2	
18	1131	1	0.5	1	11	0	49	0.6	101	5	...	658	878	1835	19	13	
19	682	1	0.5	0	4	0	19	1.0	121	4	...	902	1064	2337	11	1	

20 rows × 21 columns

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

	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	ta
0	1	1043	1	1.8	1	14	0	5	0.1	193	...	16	226	1412	3476	12	7	
1	2	841	1	0.5	1	4	1	61	0.8	191	...	12	746	857	3895	6	0	
2	3	1807	1	2.8	0	1	0	27	0.9	186	...	4	1270	1366	2396	17	10	
3	4	1546	0	0.5	1	18	1	25	0.5	96	...	20	295	1752	3893	10	0	
4	5	1434	0	1.4	0	11	1	49	0.5	108	...	18	749	810	1773	15	8	
5	6	1464	1	2.9	1	5	1	50	0.8	198	...	9	569	939	3506	10	7	
6	7	1718	0	2.4	0	1	0	47	1.0	156	...	3	1283	1374	3873	14	2	
7	8	833	0	2.4	1	0	0	62	0.8	111		2	1312	1880	1495	7	2	

```
dftrain.shape
```

```
(2000, 21)
```

```
dftest.shape
```

```
(1000, 21)
```

```
dftrain.nunique()
```

```
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
```

```
dftest.nunique()
```

```
id               1000
battery_power     721
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         694
px_width          743
ram               872
sc_h               15
sc_w               19
talk_time          19
three_g             2
touch_screen        2
wifi                2
dtype: int64
```

```
dftrain.isnull().sum()
```

```
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
```

```
dftest.isnull().sum()
```

```
id               0
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
dtype: int64
```

```
dftest.drop(columns='id', inplace=True)
```

```
dftest.shape
```

```
(1000, 20)
```

```
dftrain.shape
```

```
(2000, 21)
```

```
dftrain.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
```

```
dftrain.describe()
```

	battery_power	blue	clock_speed	dual_sim	fc	four_g	int_memory	m_dep	mobile_wt	n_cores	...	px_
count	2000.000000	2000.0000	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000	...	2000.

```
#column names in the train dataset
dftrain.columns
```

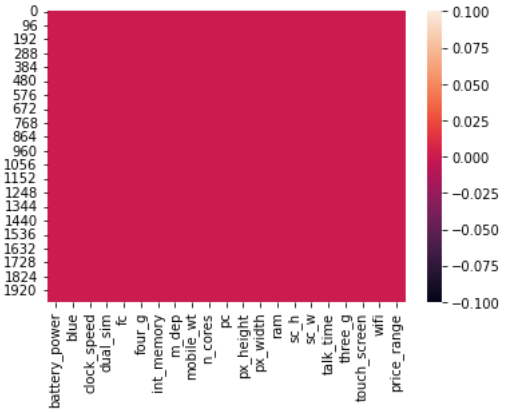
```
Index(['battery_power', 'blue', 'clock_speed', 'dual_sim', 'fc', 'four_g',
       'int_memory', 'm_dep', 'mobile_wt', 'n_cores', 'pc', 'px_height',
       'px_width', 'ram', 'sc_h', 'sc_w', 'talk_time', 'three_g',
       'touch_screen', 'wifi', 'price_range'],
      dtype='object')
```

75%	1615.250000	1.0000	2.200000	1.000000	7.000000	1.000000	48.000000	0.800000	170.000000	7.000000	...	947.

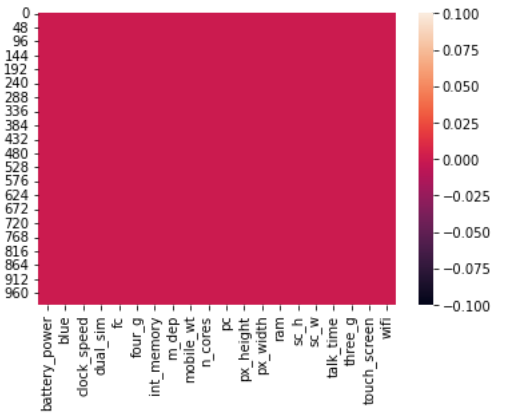
```
# Checking the skewness in the train dataset
dftrain.skew()
```

```
battery_power    0.031898
blue             0.020016
clock_speed      0.178084
dual_sim        -0.038035
fc               1.019811
four_g          -0.086144
int_memory       0.057889
m_dep            0.089082
mobile_wt        0.006558
n_cores          0.003628
pc               0.017306
px_height        0.666271
px_width         0.014787
ram              0.006628
sc_h            -0.098884
sc_w             0.633787
talk_time        0.009512
three_g         -1.228142
touch_screen    -0.012009
wifi            -0.028024
price_range      0.000000
dtype: float64
```

```
#  visualize null values using heat map
plt.figure()
sns.heatmap(dftrain.isnull())
plt.show()
```




```
plt.figure()
sns.heatmap(dftest.isnull())
plt.show()
```




```
dftrain.corr()
```

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

21 rows x 21 columns

```
plt.figure(figsize=(14,12))
plt.title('Correlation of Features')
sns.heatmap(dftrain.corr(),annot=True)
```

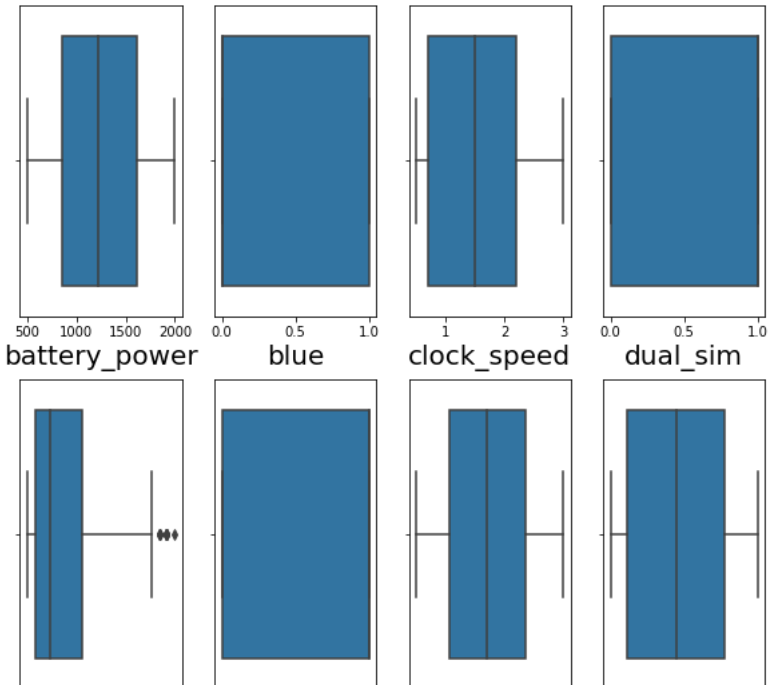
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fbfe7f1ffa0>
```


```
numcol=[]
for i in dftrain.dtypes.index:
  if dftrain.dtypes[i] != "object":
    numcol.append(i)
```

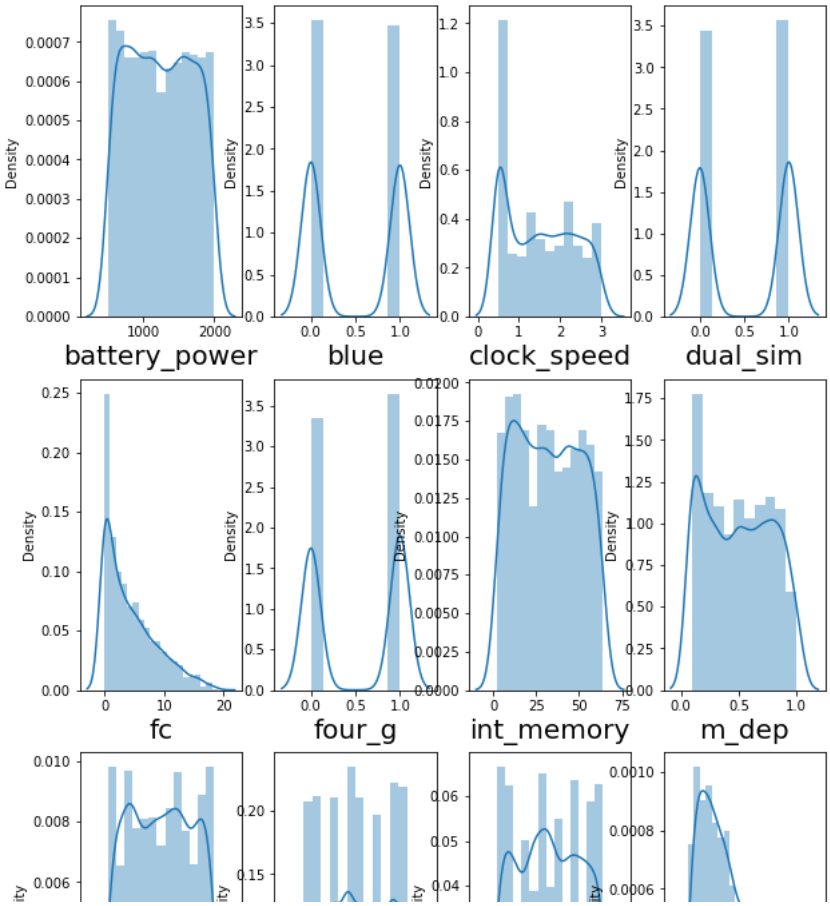
```
numcol
```

```
['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']


plt.figure(figsize=(10,30))
plotn=1
for i in numcol:
  if plotn<=21:
    ax=plt.subplot(6,4,plotn)
    sns.boxplot(dftrain[i])
    plt.xlabel(i,fontsize=20)
    plotn+=1
```

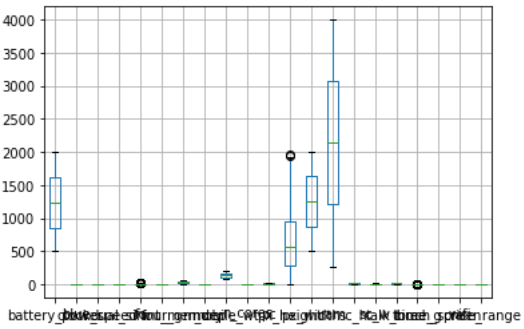



```
plt.figure(figsize=(10,30))
plotn=1
for i in numcol:
  if plotn<=21:
    ax=plt.subplot(6,4,plotn)
    sns.distplot(dftrain[i],)
    plt.xlabel(i,fontsize=20)
    plotn+=1
```

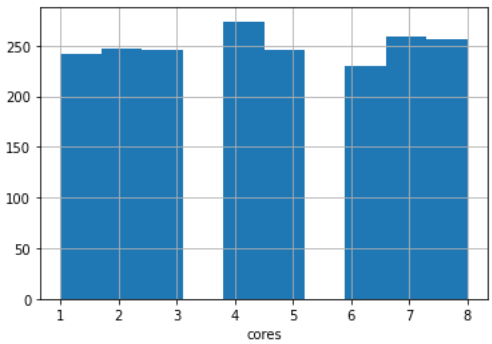



```
dftrain.boxplot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fbfe39544c0>
```

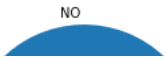


```
plt.hist(dftrain['n_cores'])
plt.xlabel('cores')
plt.grid(True)
```



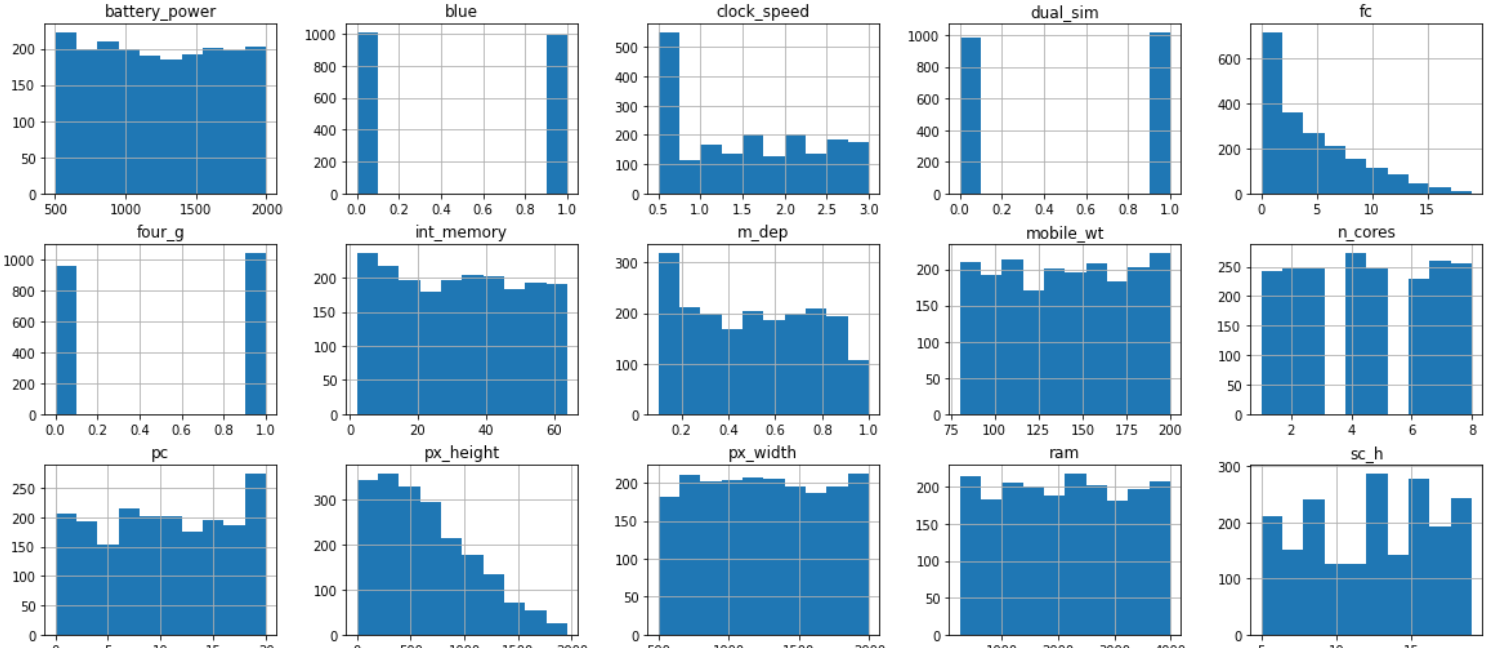

```
plt.pie(dftrain['dual_sim'].value_counts(),labels=['NO', 'YES'])
```

```
([<matplotlib.patches.Wedge at 0x7fbfe1aea190>,
  <matplotlib.patches.Wedge at 0x7fbfe1aea640>],
 [Text(-0.032824861949985885, 1.0995101311211117, 'NO'),
  Text(0.032824964893553486, -1.0995101280478217, 'YES')])
```



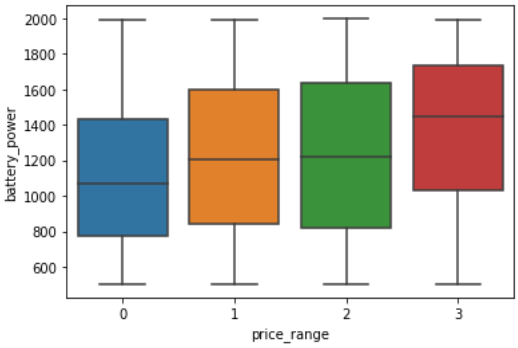

```
dftrain.hist(figsize=(20,15))
```

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7fbfe1afe610>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fbfe1bebd30>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fbfe1cd74c0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fbfe1cc4be0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fbfe1c3f340>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7fbfe1cef9a0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fbfe1cefa90>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fbfe297c760>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fbfe2209910>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fbfe1f1eb80>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7fbfe1d45880>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fbfe1b3ca60>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fbfe1aabeb0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fbfe1a672e0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fbfe1a946d0>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7fbfe1a41ac0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fbfe19f0eb0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fbfe19a92e0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fbfe19d56d0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fbfe1984ac0>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7fbfe1934eb0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fbfe18ed220>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fbfe190c7c0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fbfe18babb0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fbfe1869fa0>]],
      dtype=object)
```




```
sns.boxplot(x='price_range', y='battery_power', data=dftrain)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fbfe13ae730>
```

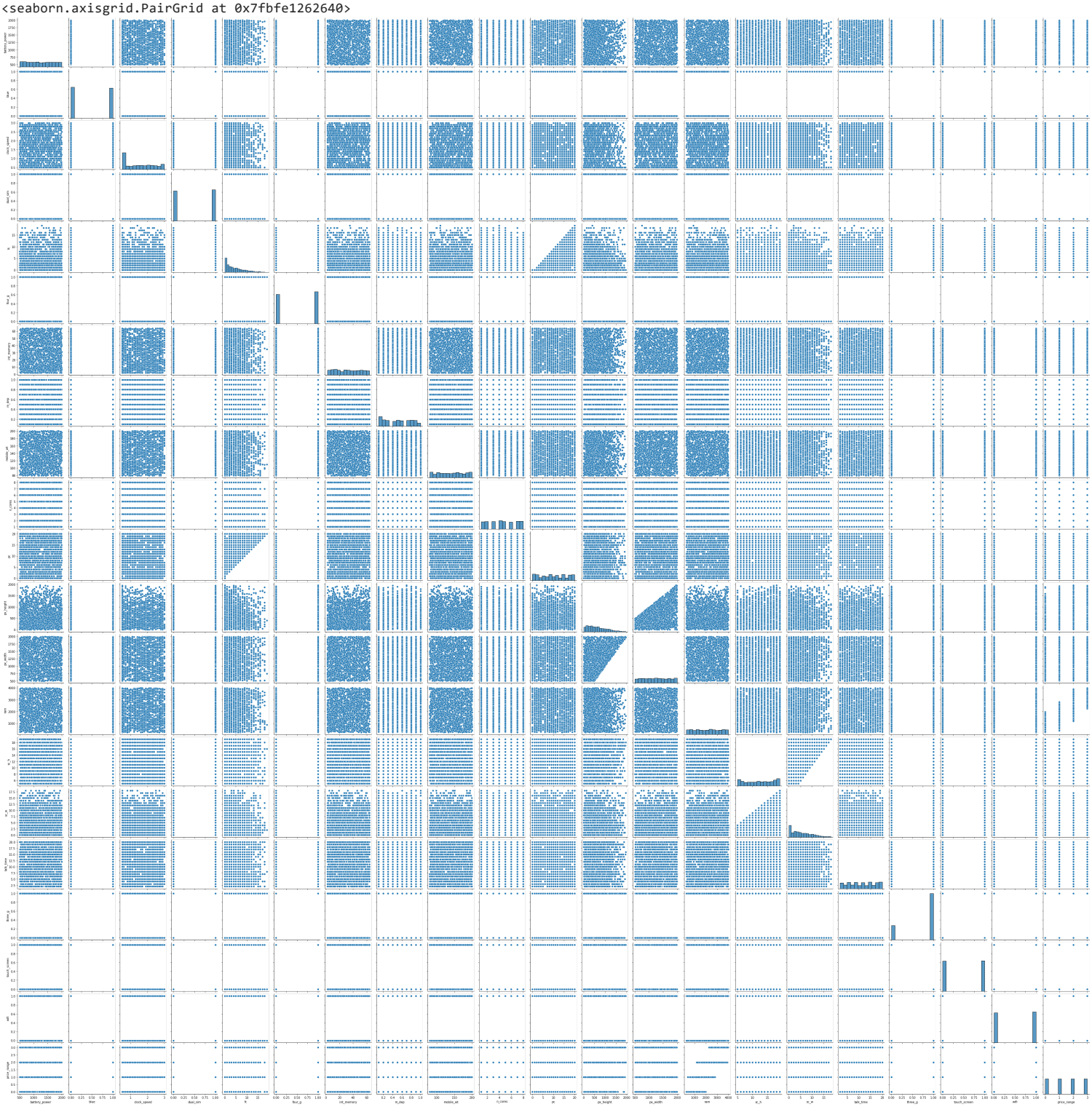



```
sns.boxplot(x='price_range', y='ram', data=dftrain)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fbfe12e23a0>
```


```
sns.pairplot(dftrain)
```

```
<seaborn.axisgrid.PairGrid at 0x7fbfe1262640>
```



```
df = pd.concat([dftrain, dftest])
```

```
df
```

	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
0	842	0	2.2	0	1	0	7	0.6	188	2	...	20	756	2549	9	7
1	1021	1	0.5	1	0	1	53	0.7	136	3	...	905	1988	2631	17	3
2	563	1	0.5	1	2	1	41	0.9	145	5	...	1263	1716	2603	11	2
3	615	1	2.5	0	0	0	10	0.8	131	6	...	1216	1786	2769	16	8
4	1821	1	1.2	0	13	1	44	0.6	141	2	...	1208	1212	1411	8	2
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
995	1700	1	1.9	0	0	1	54	0.5	170	7	...	644	913	2121	14	8
996	609	0	1.8	1	0	0	13	0.9	186	4	...	1152	1632	1933	8	1
997	1185	0	1.4	0	1	1	8	0.5	80	1	...	477	825	1223	5	0
998	1533	1	0.5	1	0	0	50	0.4	171	2	...	38	832	2509	15	11

```
df.isnull().sum()
```

```
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      1000
dtype: int64
```

```
df = df.dropna()
```

```
df.isnull().sum()
```

```
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
```

```
#separtion of target and feature
x = df.drop(columns='price_range')
y  = df.price_range
```

```
x[:200]
```

	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	t
0	842	0	2.2	0	1	0	7	0.6	188	2	2	20	756	2549	9	7	
1	1021	1	0.5	1	0	1	53	0.7	136	3	6	905	1988	2631	17	3	
2	563	1	0.5	1	2	1	41	0.9	145	5	6	1263	1716	2603	11	2	
3	615	1	2.5	0	0	0	10	0.8	131	6	9	1216	1786	2769	16	8	
4	1821	1	1.2	0	13	1	44	0.6	141	2	14	1208	1212	1411	8	2	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
195	1526	0	2.1	0	1	1	23	0.2	117	7	8	718	751	2227	18	10	
196	1989	0	2.5	1	0	1	41	0.8	94	3	13	1100	1497	1665	17	9	
197	1308	0	1.9	0	0	1	61	0.7	106	3	7	59	1215	3355	15	2	

```
y
```

```
0       1.0
1       2.0
2       2.0
3       2.0
4       1.0
       ...
1995    0.0
1996    2.0
1997    3.0
1998    0.0
1999    3.0
Name: price_range, Length: 2000, dtype: float64
```

```
from sklearn.model_selection import train_test_split
xtrain, xtest, ytrain, ytest = train_test_split(x, y, test_size=0.2)
```

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
```

```
clf = RandomForestClassifier()
clf.fit(xtrain, ytrain)
```

```
RandomForestClassifier()
```

```
#accuracy
clf.score(xtest,ytest)
```

```
0.8575
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
```

```
# Define a list of classifiers
classifiers = [LogisticRegression(),
               KNeighborsClassifier(n_neighbors=5),
               GaussianNB(),
               DecisionTreeClassifier(),
               RandomForestClassifier(n_estimators=100),
               SVC(kernel='linear')]
```

```
# Evaluate each classifier on the test data
for classifier in classifiers:
    model = classifier
    model.fit(xtrain, ytrain)
    ypred = model.predict(xtest)
    accuracy = accuracy_score(ytest, ypred)
    print(f'{type(classifier).__name__}: {accuracy:.3f}')
```

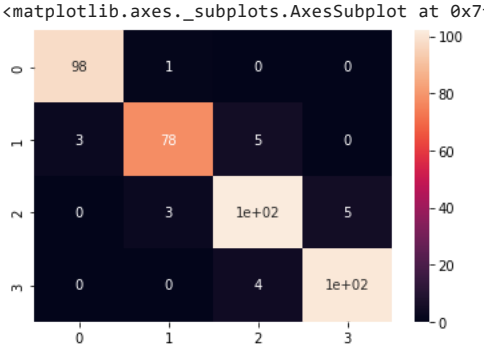
```
LogisticRegression: 0.627
KNeighborsClassifier: 0.910
GaussianNB: 0.800
DecisionTreeClassifier: 0.787
RandomForestClassifier: 0.868
SVC: 0.948
```

```
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report


# Evaluate each classifier on the test data
for classifier in classifiers:
    model = classifier
    model.fit(xtrain, ytrain)
    y_pred = model.predict(xtest)
    accuracy = accuracy_score(ytest, ypred)
    cm = confusion_matrix(ytest, ypred)
    cr = classification_report(ytest, ypred)
    print(f'{type(classifier).__name__}:')
    print(f'Accuracy: {accuracy:.3f}')
    print(f'Confusion Matrix: \n{cm}')
    print(f'Classification Report: \n{cr}')
    print('')
```

```
LogisticRegression:
Accuracy: 0.948
Confusion Matrix:
[[ 98   1   0   0]
 [  3  78   5   0]
 [  0   3 102   5]
 [  0   0   4 101]]
Classification Report:
              precision    recall  f1-score   support

         0.0       0.97      0.99      0.98        99
         1.0       0.95      0.91      0.93        86
         2.0       0.92      0.93      0.92       110
         3.0       0.95      0.96      0.96       105

    accuracy                           0.95       400
   macro avg       0.95      0.95      0.95       400
weighted avg       0.95      0.95      0.95       400


KNeighborsClassifier:
Accuracy: 0.948
Confusion Matrix:
[[ 98   1   0   0]
 [  3  78   5   0]
 [  0   3 102   5]
 [  0   0   4 101]]
Classification Report:
              precision    recall  f1-score   support

         0.0       0.97      0.99      0.98        99
         1.0       0.95      0.91      0.93        86
         2.0       0.92      0.93      0.92       110
         3.0       0.95      0.96      0.96       105

    accuracy                           0.95       400
   macro avg       0.95      0.95      0.95       400
weighted avg       0.95      0.95      0.95       400


GaussianNB:
Accuracy: 0.948
Confusion Matrix:
[[ 98   1   0   0]
 [  3  78   5   0]
 [  0   3 102   5]
 [  0   0   4 101]]
Classification Report:
              precision    recall  f1-score   support

         0.0       0.97      0.99      0.98        99
         1.0       0.95      0.91      0.93        86
         2.0       0.92      0.93      0.92       110
         3.0       0.95      0.96      0.96       105

    accuracy                           0.95       400
   macro avg       0.95      0.95      0.95       400
weighted avg       0.95      0.95      0.95       400
```

```
sns.heatmap(cm,annot=True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fbfd7929520>
```



```
from sklearn.model_selection import cross_val_score


# Evaluate each classifier using cross-validation
for classifier in classifiers:
    model = classifier
    scores = cross_val_score(model, x, y, cv=5)
    print('Classifier:', type(classifier).__name__)
    print('Cross-Validation Scores:', scores)
    print('Mean Score:', np.mean(scores))
    print('Standard Deviation:', np.std(scores))
    print()
```

```
Classifier: LogisticRegression
Cross-Validation Scores: [0.6325 0.6525 0.6425 0.63   0.625 ]
Mean Score: 0.6365
Standard Deviation: 0.009823441352194238

Classifier: KNeighborsClassifier
Cross-Validation Scores: [0.92   0.9175 0.925  0.925  0.91  ]
Mean Score: 0.9195
Standard Deviation: 0.005567764362830031

Classifier: GaussianNB
Cross-Validation Scores: [0.805  0.82   0.83   0.8075 0.7825]
Mean Score: 0.8089999999999999
Standard Deviation: 0.016015617378046958

Classifier: DecisionTreeClassifier
Cross-Validation Scores: [0.8325 0.8325 0.8    0.8325 0.8225]
Mean Score: 0.8240000000000001
Standard Deviation: 0.012609520212918482

Classifier: RandomForestClassifier
Cross-Validation Scores: [0.875  0.8825 0.9    0.8725 0.845 ]
Mean Score: 0.875
Standard Deviation: 0.01781852968120547

Classifier: SVC
Cross-Validation Scores: [0.9775 0.9675 0.9675 0.9725 0.98  ]
Mean Score: 0.9730000000000001
Standard Deviation: 0.005099019513592774
```

```
#As SVC shows the best results so we will be doing hyperparameter tuninng in svc
# Define the SVC classifier and its hyperparameters
classifier = SVC()
parameters = {
    'C': [0.1, 1, 10, 100, 1000],
    'kernel': ['linear', 'rbf']
}


# Perform hyperparameter tuning with 5-fold cross-validation
from sklearn.model_selection import GridSearchCV
model = GridSearchCV(classifier, parameters, cv=5)
model.fit(x, y)
```

```
GridSearchCV(cv=3, estimator=SVC(),
             param_grid={'C': [0.1, 1, 10, 100, 1000],
                         'kernel': ['linear', 'rbf']})
```

```
# Print the best hyperparameters and score
best_parameters = model.best_params_
best_score = model.best_score_
print('Best Parameters:', best_parameters)
print('Best Score:', best_score)
```

```
Best Parameters: {'C': 0.1, 'kernel': 'linear'}
Best Score: 0.9749967358663011
```