In [1]:
```
# moduls for data
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from pyecharts.charts import Pie
from pyecharts import options as opts
from plotly.subplots import make_subplots
import plotly.graph_objects as go

# moduls for preprocessing ans model selection
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV


# moduls for model implementation
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier

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

from sklearn.ensemble import BaggingClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import VotingClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.ensemble import AdaBoostClassifier
from xgboost  import XGBClassifier



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

In [2]:
```
df = pd.read_csv("mobile_data.csv")
```

In [3]:
```
df.shape
```

Out[3]: (2000, 21)

In [4]:
```
df.head().T
```

Out[4]:

	0	1	2	3	4
battery_power	842.0	1021.0	563.0	615.0	1821.0
blue	0.0	1.0	1.0	1.0	1.0
clock_speed	2.2	0.5	0.5	2.5	1.2
dual_sim	0.0	1.0	1.0	0.0	0.0
fc	1.0	0.0	2.0	0.0	13.0
four_g	0.0	1.0	1.0	0.0	1.0
int_memory	7.0	53.0	41.0	10.0	44.0
m_dep	0.6	0.7	0.9	0.8	0.6
mobile_wt	188.0	136.0	145.0	131.0	141.0
n_cores	2.0	3.0	5.0	6.0	2.0
pc	2.0	6.0	6.0	9.0	14.0
px_height	20.0	905.0	1263.0	1216.0	1208.0
px_width	756.0	1988.0	1716.0	1786.0	1212.0
ram	2549.0	2631.0	2603.0	2769.0	1411.0
sc_h	9.0	17.0	11.0	16.0	8.0
sc_w	7.0	3.0	2.0	8.0	2.0
talk_time	19.0	7.0	9.0	11.0	15.0
three_g	0.0	1.0	1.0	1.0	1.0
touch_screen	0.0	1.0	1.0	0.0	1.0
wifi	1.0	0.0	0.0	0.0	0.0
price_range	1.0	2.0	2.0	2.0	1.0

In [5]:
```
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
```

INFORMATION ABOUT DATA

=battery_power: Total energy a battery can store in one time measured in mAh.

=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 in Gigabytes.

m_dep: Mobile Depth in cm.

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 in Mega Byte.

sc_h: Screen Height of mobile in cm.

sc_w: Screen Width of mobile in cm.

talk_time: longest time that a single battery charge will last when you are.

=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 cost), 1(medium cost), 2(high cost) and 3(very high cost)

In [6]:
```
df.duplicated().any()
```

Out[6]:
```
False
```

In [7]:
```
df.isnull().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
```

In [8]:
```
df_columns = df.columns
df_columns
```

Out[8]:
```
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')
```

In [9]:
```
values  = {}

for i in df_columns:
    values[i] = df[i].nunique()
    print(f"Number of unique values in {values.popitem()}\n")
```

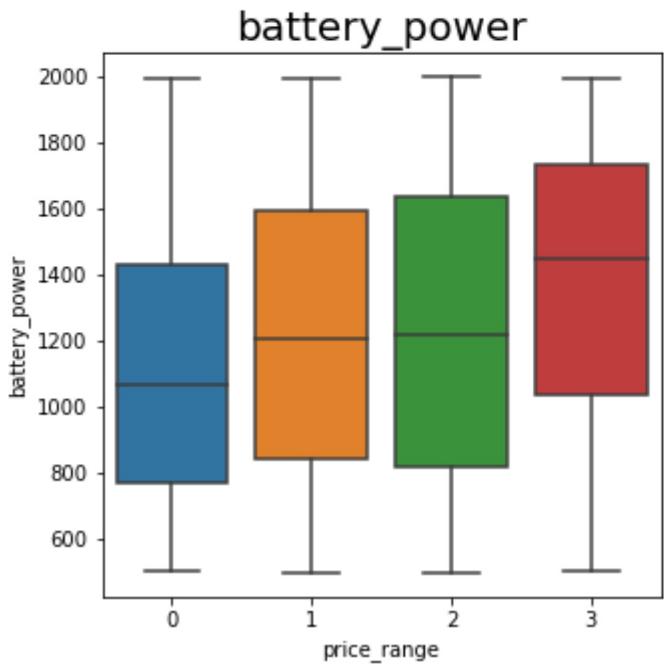
```
Number of unique values in ('battery_power', 1094)

Number of unique values in ('blue', 2)

Number of unique values in ('clock_speed', 26)

Number of unique values in ('dual_sim', 2)

Number of unique values in ('fc', 20)

Number of unique values in ('four_g', 2)

Number of unique values in ('int_memory', 63)

Number of unique values in ('m_dep', 10)

Number of unique values in ('mobile_wt', 121)

Number of unique values in ('n_cores', 8)

Number of unique values in ('pc', 21)

Number of unique values in ('px_height', 1137)

Number of unique values in ('px_width', 1109)

Number of unique values in ('ram', 1562)

Number of unique values in ('sc_h', 15)

Number of unique values in ('sc_w', 19)

Number of unique values in ('talk_time', 19)

Number of unique values in ('three_g', 2)

Number of unique values in ('touch_screen', 2)

Number of unique values in ('wifi', 2)

Number of unique values in ('price_range', 4)
```

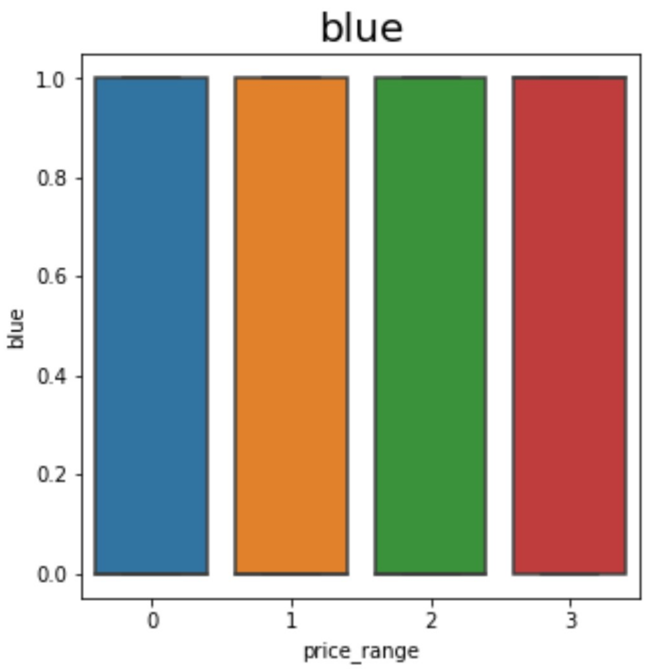
In [10]:
```
df.describe().T
```

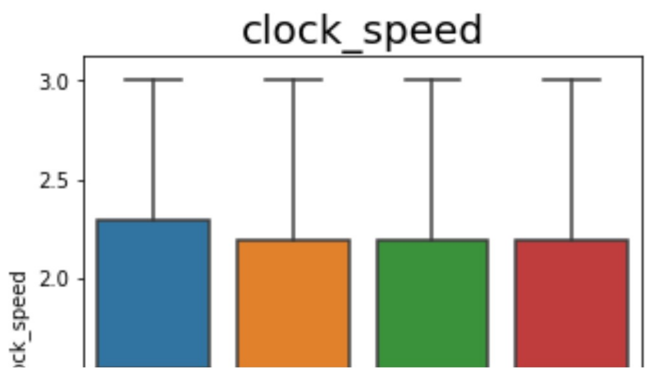
Out[10]:

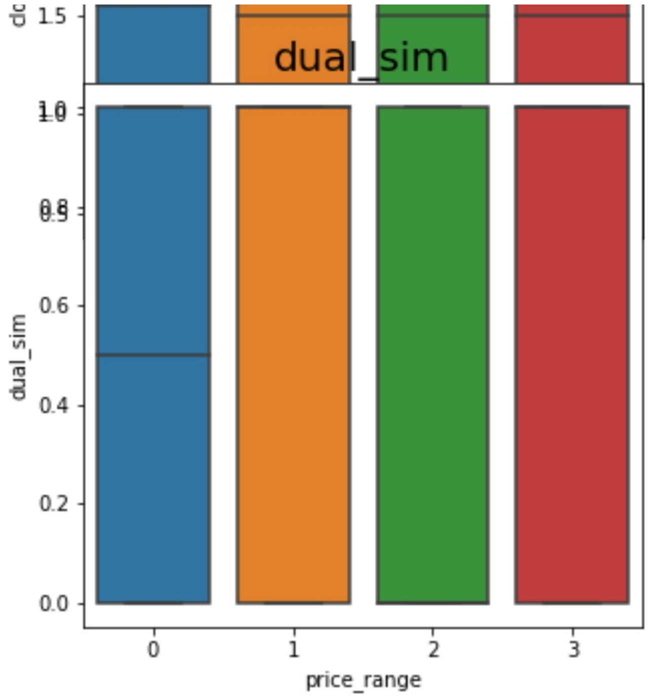
	count	mean	std	min	25%	50%	75%	max
battery_power	2000.0	1238.51850	439.418206	501.0	851.75	1226.0	1615.25	1998.0
blue	2000.0	0.49500	0.500100	0.0	0.00	0.0	1.00	1.0
clock_speed	2000.0	1.52225	0.816004	0.5	0.70	1.5	2.20	3.0
dual_sim	2000.0	0.50950	0.500035	0.0	0.00	1.0	1.00	1.0
fc	2000.0	4.30950	4.341444	0.0	1.00	3.0	7.00	19.0
four_g	2000.0	0.52150	0.499662	0.0	0.00	1.0	1.00	1.0
int_memory	2000.0	32.04650	18.145715	2.0	16.00	32.0	48.00	64.0
m_dep	2000.0	0.50175	0.288416	0.1	0.20	0.5	0.80	1.0
mobile_wt	2000.0	140.24900	35.399655	80.0	109.00	141.0	170.00	200.0
n_cores	2000.0	4.52050	2.287837	1.0	3.00	4.0	7.00	8.0
pc	2000.0	9.91650	6.064315	0.0	5.00	10.0	15.00	20.0
px_height	2000.0	645.10800	443.780811	0.0	282.75	564.0	947.25	1960.0
px_width	2000.0	1251.51550	432.199447	500.0	874.75	1247.0	1633.00	1998.0
ram	2000.0	2124.21300	1084.732044	256.0	1207.50	2146.5	3064.50	3998.0
sc_h	2000.0	12.30650	4.213245	5.0	9.00	12.0	16.00	19.0
sc_w	2000.0	5.76700	4.356398	0.0	2.00	5.0	9.00	18.0
talk_time	2000.0	11.01100	5.463955	2.0	6.00	11.0	16.00	20.0
three_g	2000.0	0.76150	0.426273	0.0	1.00	1.0	1.00	1.0
touch_screen	2000.0	0.50300	0.500116	0.0	0.00	1.0	1.00	1.0
wifi	2000.0	0.50700	0.500076	0.0	0.00	1.0	1.00	1.0
price_range	2000.0	1.50000	1.118314	0.0	0.75	1.5	2.25	3.0

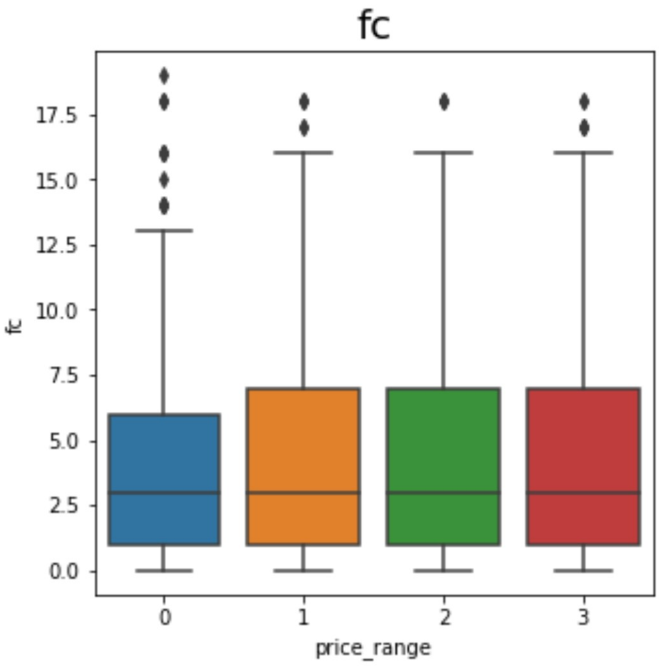
In [11]:
```
columns = df.iloc[:,:-1].columns
columns
for index,col in enumerate(columns):
    plt.figure(figsize = (5,5))
    sns.boxplot(x=df["price_range"],y = df[col])
    plt.title(col, size = 20)
    plt.show()
```

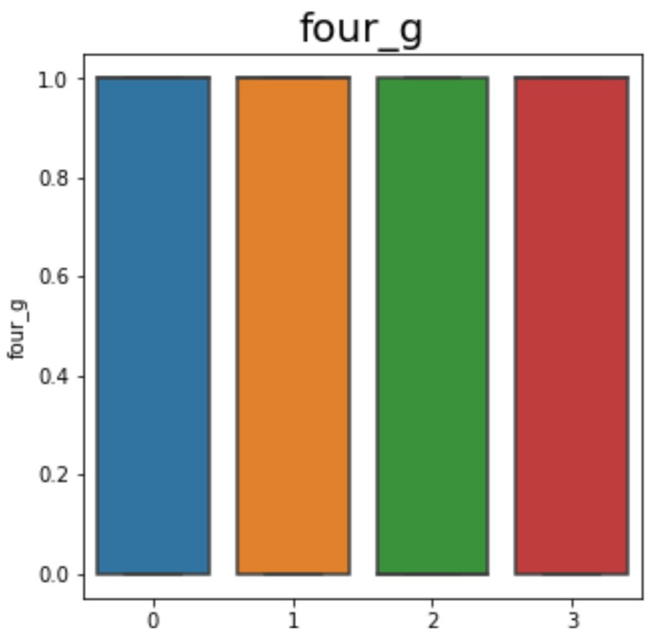




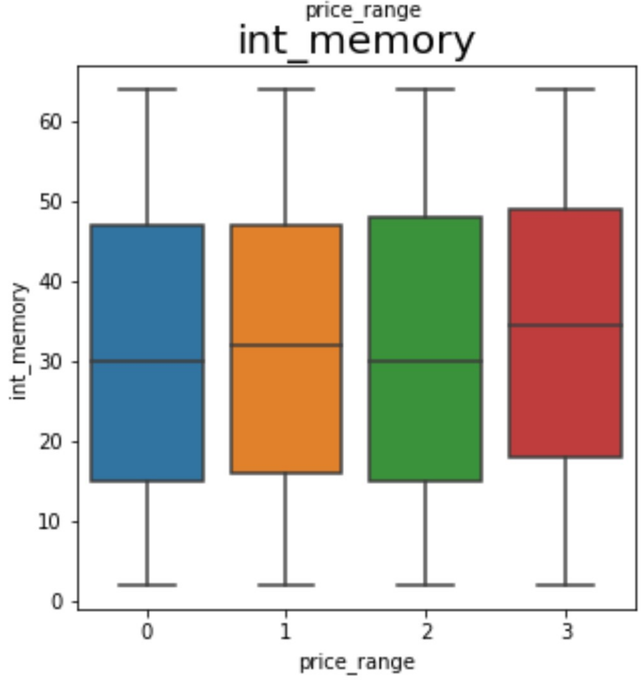


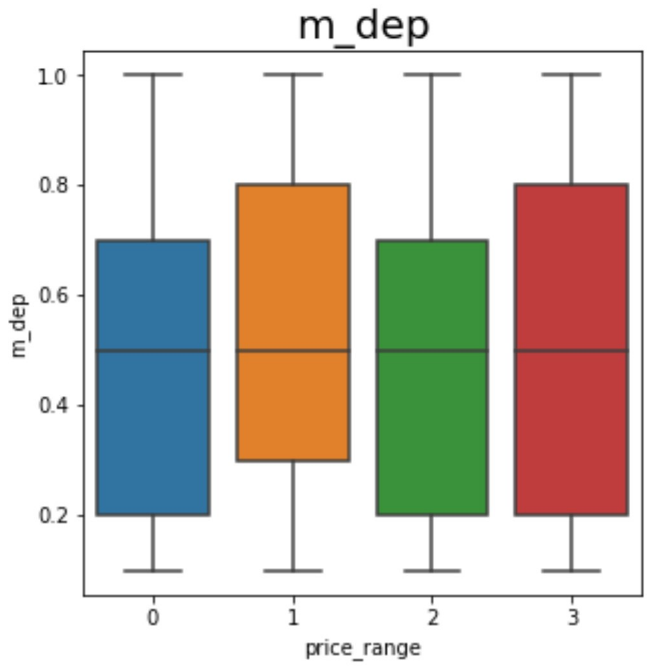

dual_sim
dual_sim
price_range
0
1
2
3
0.0
0.2
0.4
0.6
1.0

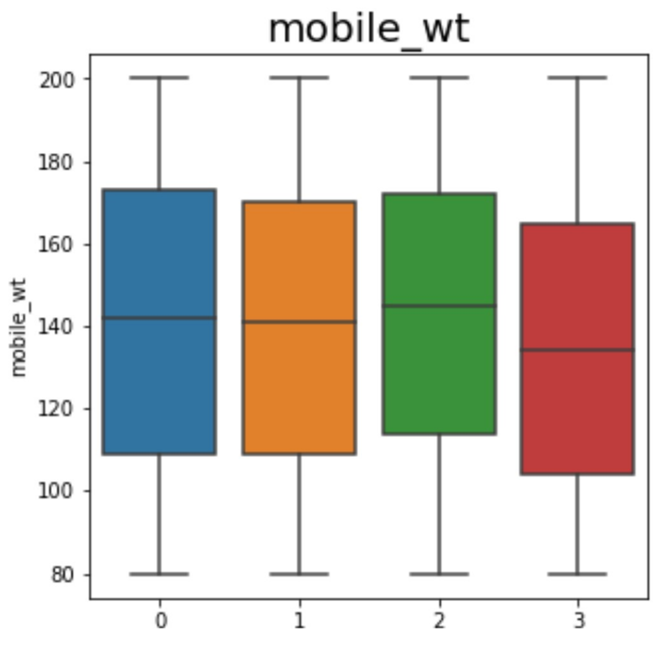

fc
fc
price_range
0
1
2
3
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5


four_g
four_g
0
1
2
3
0.0
0.2
0.4
0.6
0.8
1.0

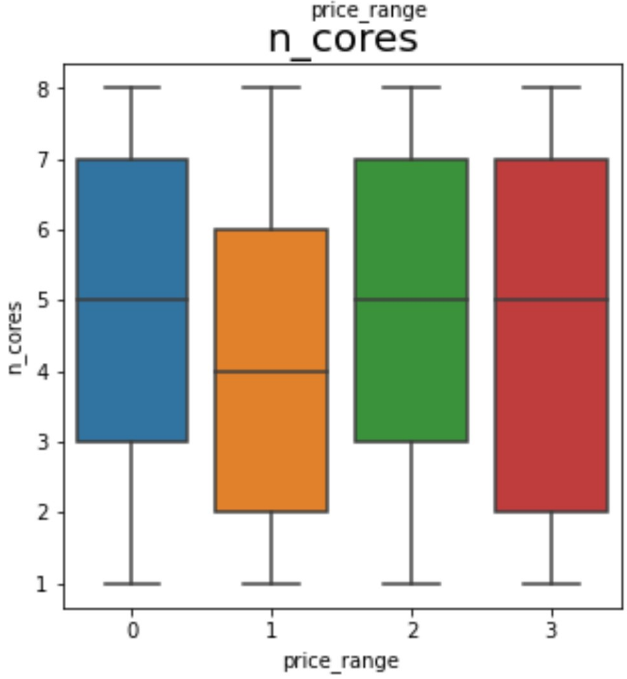
price_range

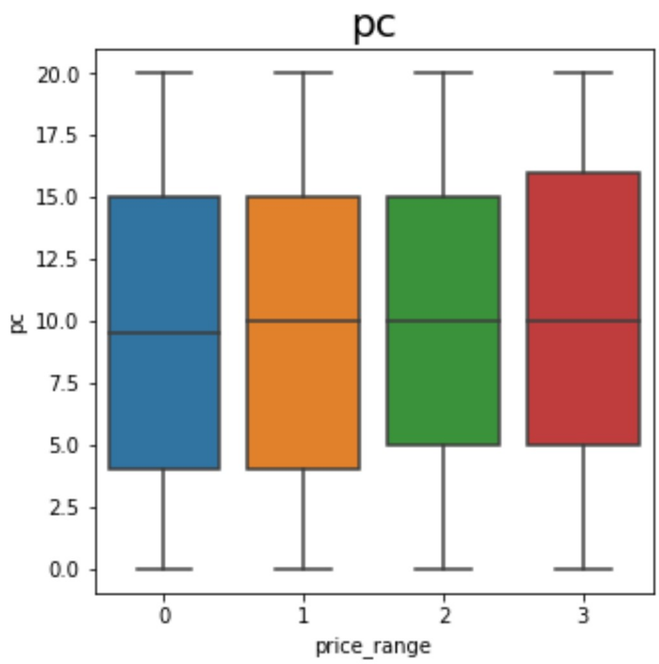

int_memory
60
50
40
30
20
10
0
int_memory
0
1
2
3
price_range

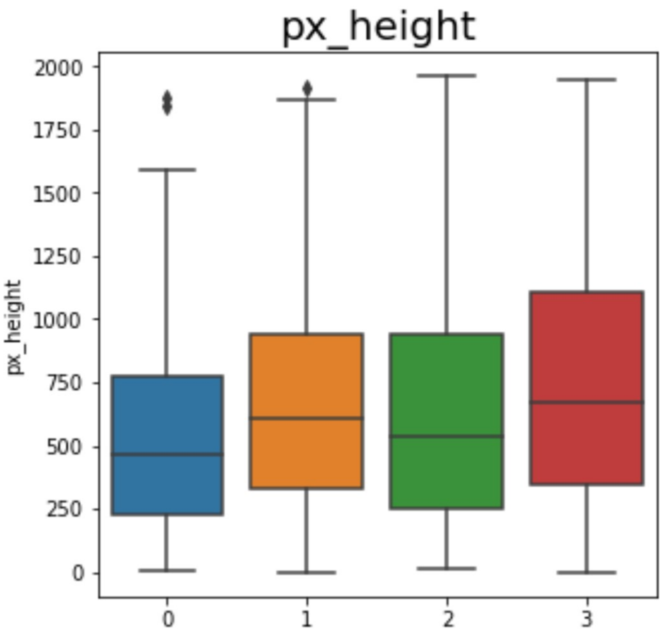

m_dep
1.0
0.8
0.6
0.4
0.2
m_dep
0
1
2
3
price_range


mobile_wt
200
180
160
140
120
100
80
mobile_wt
0
1
2
3

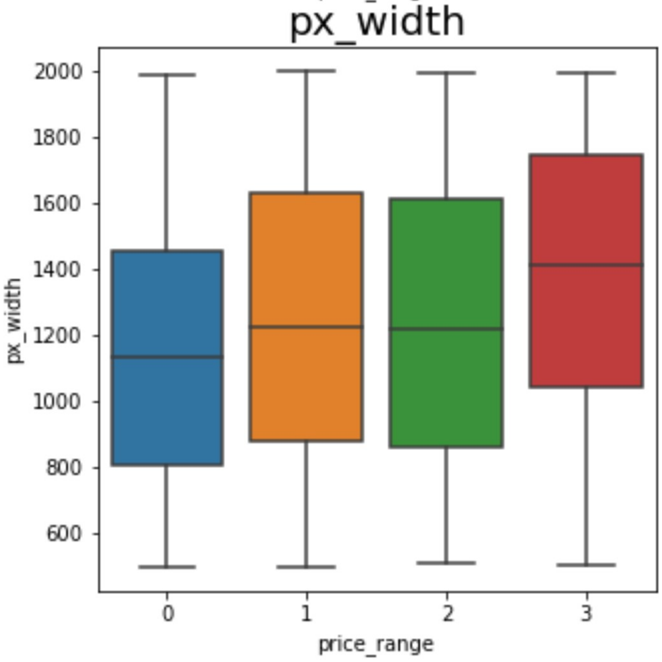
price_range

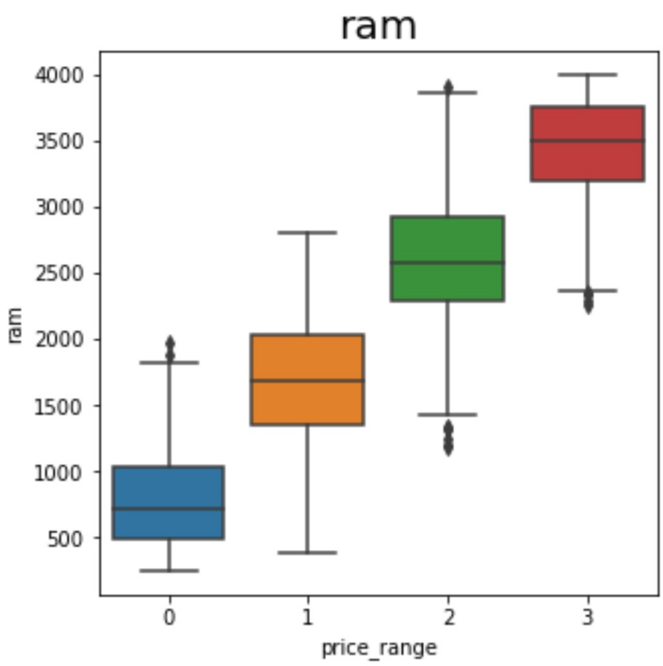

n_cores
n_cores
price_range
0
1
2
3
1
2
3
4
5
6
7
8

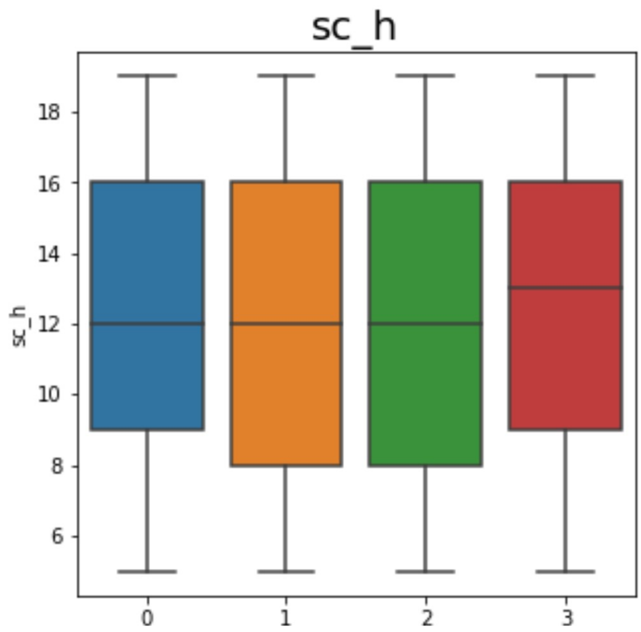

pc
pc
price_range
0
1
2
3
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0


px_height
px_height
0
1
2
3
0
250
500
750
1000
1250
1500
1750
2000

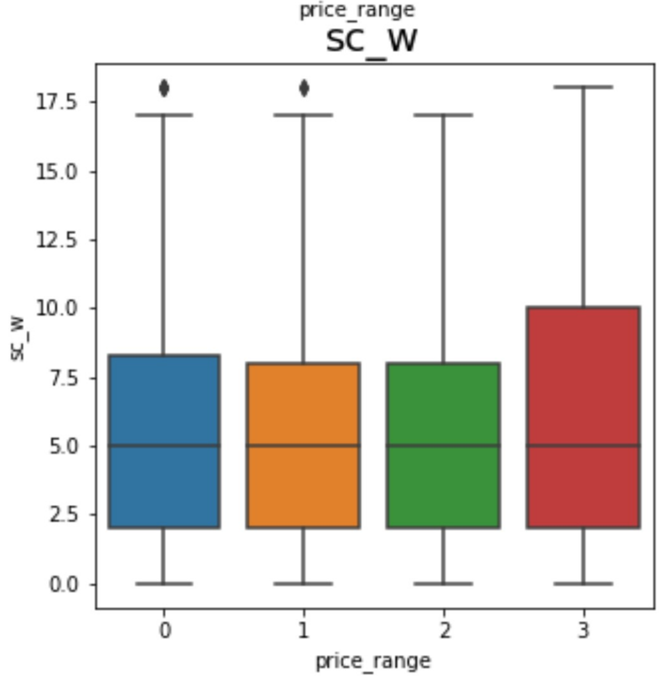
price_range

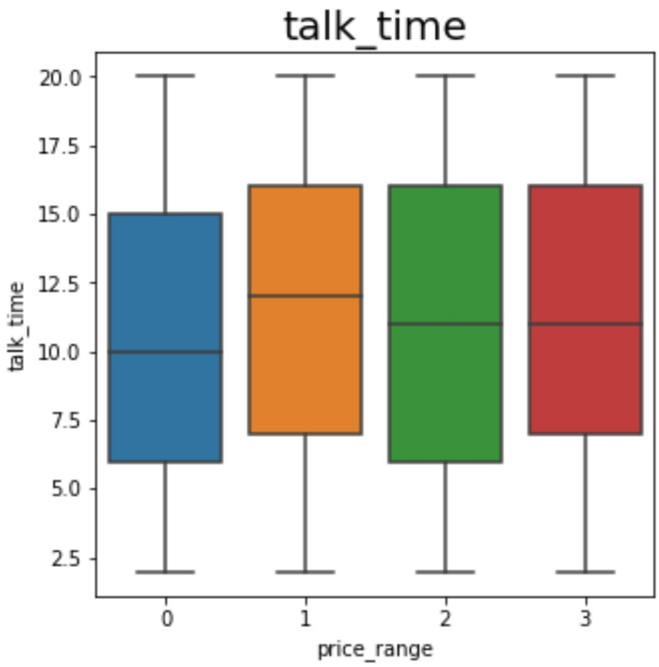

px_width
2000
1800
1600
1400
1200
1000
800
600
px_width
0
1
2
3
price_range

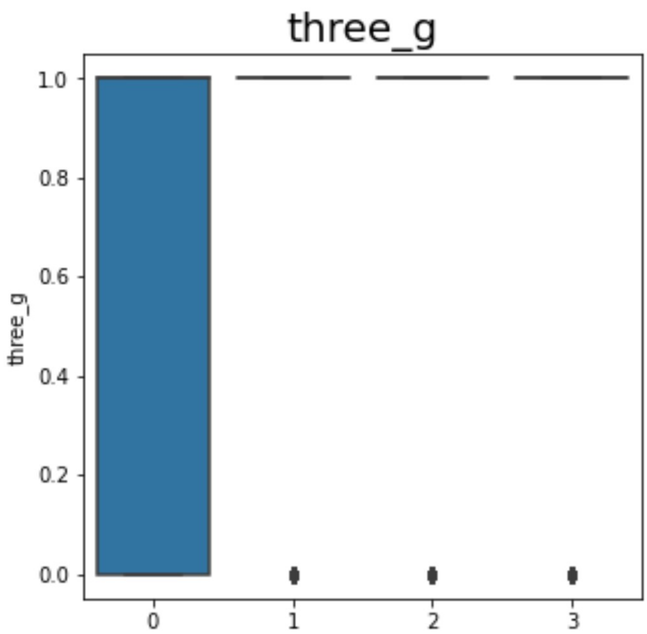

ram
4000
3500
3000
2500
2000
1500
1000
500
ram
0
1
2
3
price_range


sc_h
18
16
14
12
10
8
6
sc_h
0
1
2
3

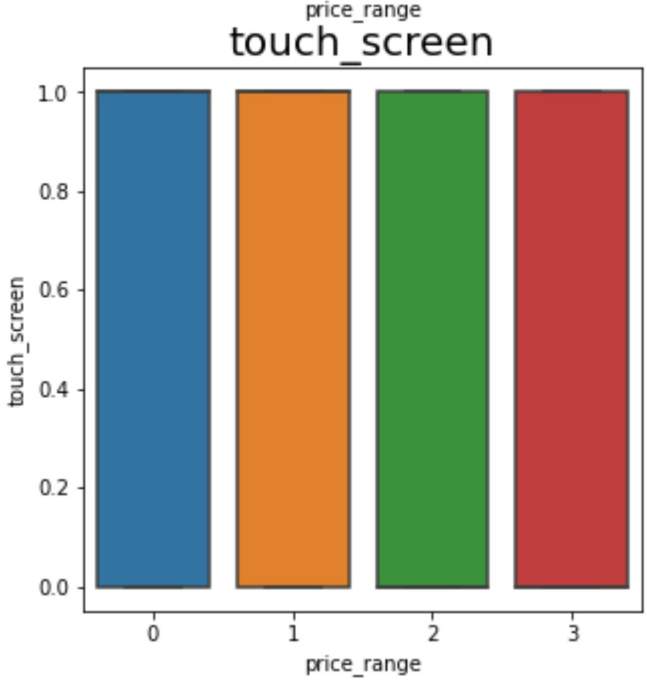
price_range

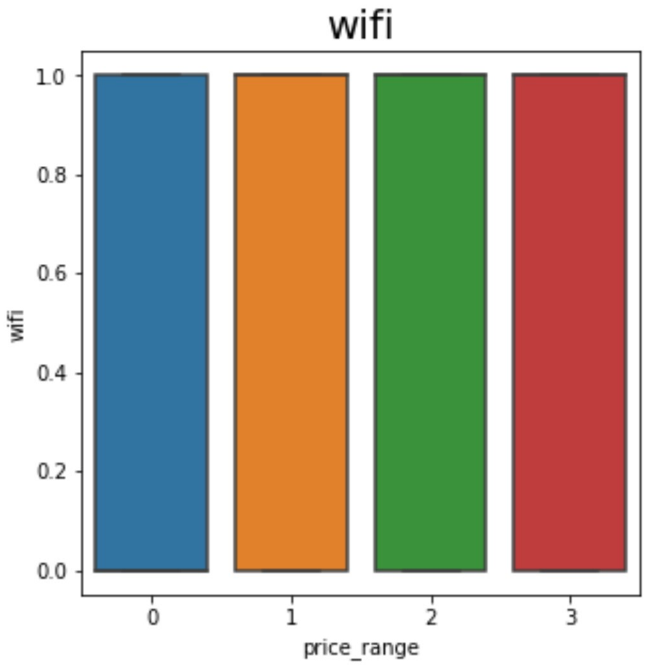

sc_w
sc_w
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
0
1
2
3
price_range


talk_time
talk_time
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0
1
2
3
price_range

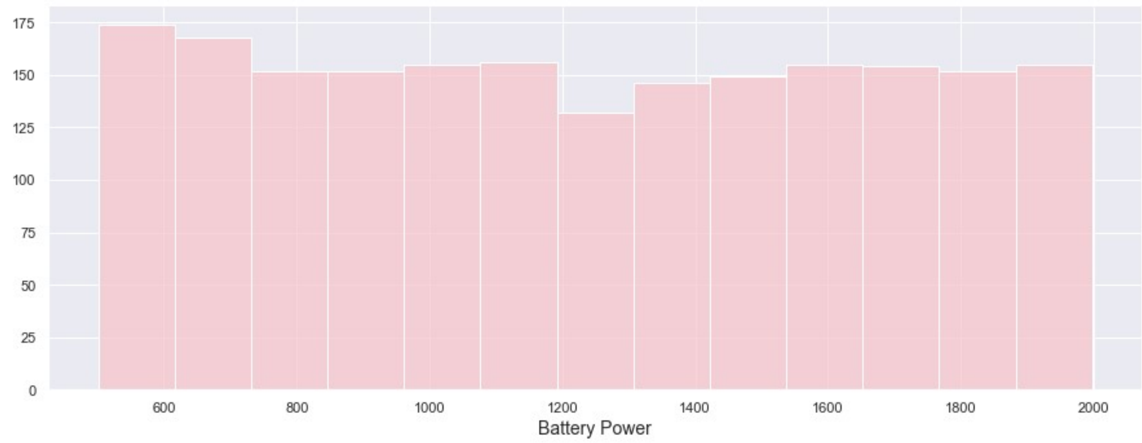

three_g
three_g
1.0
0.8
0.6
0.4
0.2
0.0
0
1
2
3

price_range


touch_screen
1.0
0.8
0.6
0.4
0.2
0.0
touch_screen
0
1
2
3
price_range

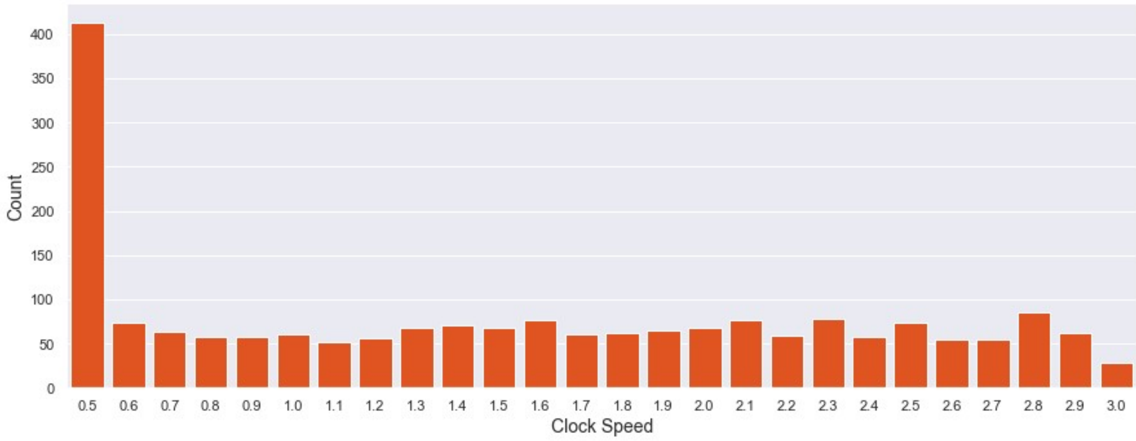

wifi
1.0
0.8
0.6
0.4
0.2
0.0
wifi
0
1
2
3
price_range

In [12]:
```
sns.set(rc={'figure.figsize':(5,5)})
sns.displot(x='battery_power',
            kind="hist",
            data=df,
            height=5,
            aspect=2.5,
            color = "#F7C5CC")
plt.xlabel("Battery Power", size=14)
plt.ylabel("", size=14)

plt.tight_layout()
plt.show()
```



In [ ]:

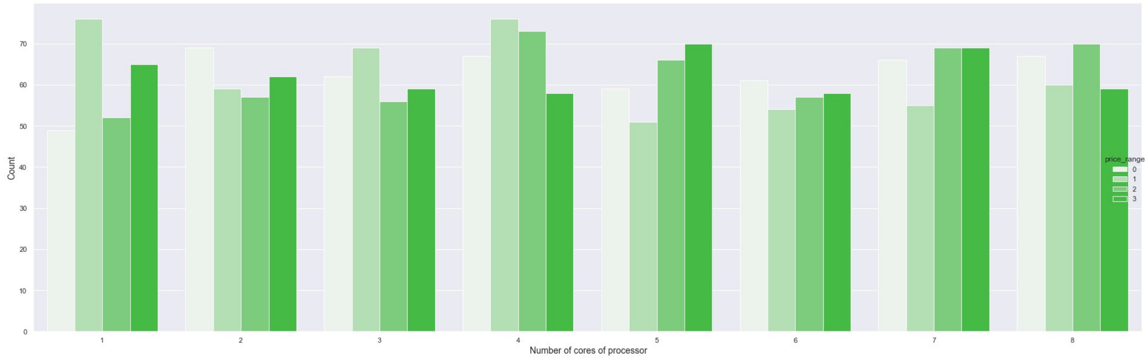
In [13]:
```
colors='#FF4500'
sns.catplot(x="clock_speed",
            kind="count",
            data=df,
            height=5,
            aspect=2.5,
            color=colors)
plt.xlabel("Clock Speed", size=14)
plt.ylabel("Count", size=14)

plt.tight_layout()
plt.show()
df["clock_speed"].value_counts(sort=True)
```



Out[13]:
```
0.5    413
2.8     85
2.3     78
2.1     76
1.6     76
2.5     74
0.6     74
1.4     70
1.3     68
1.5     67
2.0     67
1.9     65
0.7     64
2.9     62
1.8     62
1.0     61
1.7     60
2.2     59
0.9     58
2.4     58
0.8     58
1.2     56
2.6     55
2.7     55
1.1     51
3.0     28
Name: clock_speed, dtype: int64
```

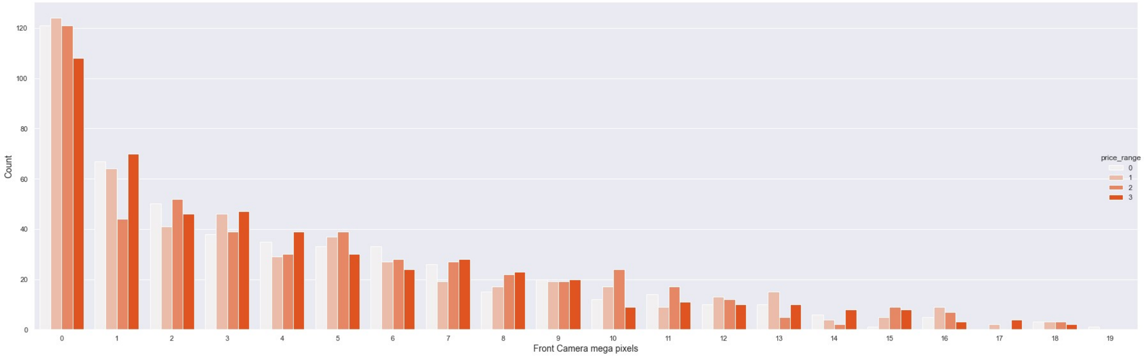
The clock speed determines how many instruction the processor can execute per second. A processor with a 1 GHz clock speed can process 1 billion instructions per second The general rule is that higher clock speed make for faster phones.

In [14]:
```
colors='#32CD32'
sns.catplot(x="n_cores",
            kind="count",
            data=df,
            legend=True,
            hue="price_range",
            height=8,
            aspect=3.0,
            color=colors)
plt.xlabel(" Number of cores of processor", size=14)
plt.ylabel("Count", size=14)

plt.tight_layout()
plt.show()
df["n_cores"].value_counts(sort=True)
```



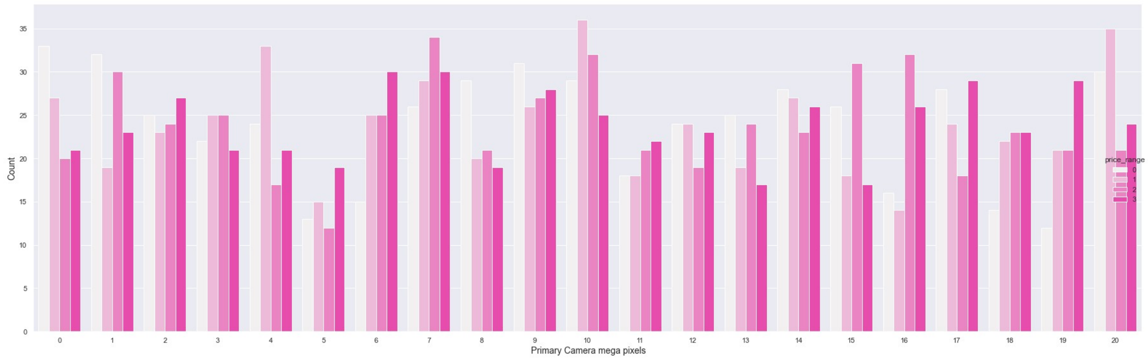
Out[14]:
```
4    274
7    259
8    256
2    247
3    246
5    246
1    242
6    230
Name: n_cores, dtype: int64
```

In [15]:
```
colors='#FF4500'
sns.catplot(x="fc",
            kind="count",
            data=df,
            legend=True,
            hue="price_range",
            height=8,
            aspect=3.0,
            color=colors)
plt.xlabel("Front Camera mega pixels", size=14)
plt.ylabel("Count", size=14)

plt.tight_layout()
plt.show()
df["fc"].value_counts(sort=True)
```



Out[15]:
```
0     474
1     245
2     189
3     170
5     139
4     133
6     112
7     100
9      78
8      77
10     62
11     51
12     45
13     40
16     24
15     23
14     20
18     11
17      6
19      1
Name: fc, dtype: int64
```

In [16]:
```
colors='#FF34B3'
sns.catplot(x="pc",
            kind="count",
            data=df,
            legend=True,
            hue="price_range",
            height=8,
            aspect=3.0,
            color=colors)
plt.xlabel(" Primary Camera mega pixels", size=14)
plt.ylabel("Count", size=14)

plt.tight_layout()
plt.show()
df["pc"].value_counts(sort=True)
```

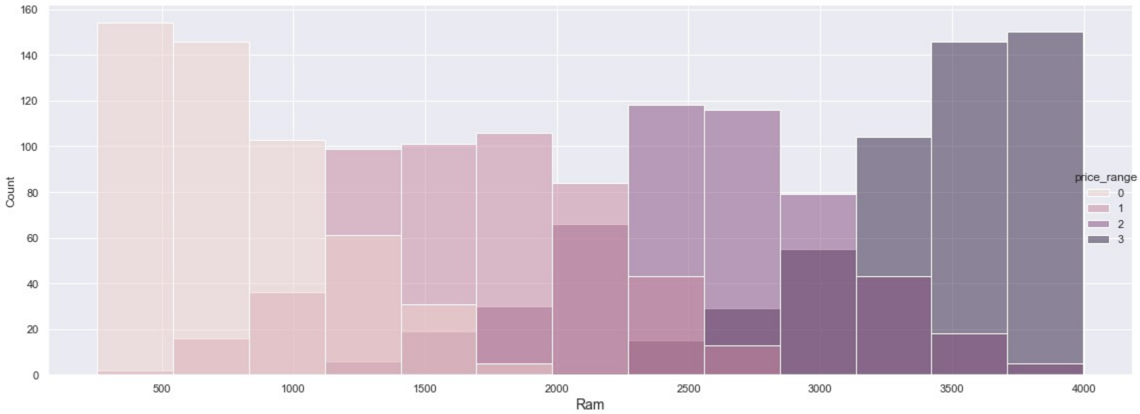


Out[16]:
```
10    122
7     119
9     112
20    110
1     104
14    104
0     101
2      99
17     99
6      95
4      95
3      93
15     92
12     90
8      89
16     88
13     85
19     83
18     82
11     79
5      59
Name: pc, dtype: int64
```

In [17]:
```
df["price_range"][(df["pc"]==0) & (df["fc"]==0)].value_counts()
```

Out[17]:
```
0    33
1    27
3    21
2    20
Name: price_range, dtype: int64
```

In [18]:
```
sns.set(rc={'figure.figsize':(5,5)})
sns.displot(x='ram',
            kind="hist",
            hue = "price_range",
            data=df,
            legend=True,
            height=6,
            aspect=2.5,
            color = "#DC143C")
plt.xlabel("Ram", size=14)

plt.tight_layout()
plt.show()
```



Pyecharts: Nightingale Rose Pie Chart

In [19]:
```
d = df['blue'].value_counts()
c= ["bluetooth supported","bluetooth Not supported"]
color_series = ["2C6BA0","#FF4500"]
rosechart = Pie(init_opts=opts.InitOpts(width = "1050px", height = "250px")
rosechart.set_colors(color_series)
rosechart.add("Bluetooth", [list(z) for z in zip(c,d)], radius = ["35%","95

rosechart.set_series_opts(label_opts=opts.LabelOpts(is_show=True, position=

rosechart.render_notebook()
```

Out[19]:

In [20]:
```
d = df['dual_sim'].value_counts()
c= ["Dual sim supported","Dual sim Not supported"]
color_series = ["#DE1A82","#C2A7B5"]
rosechart = Pie(init_opts=opts.InitOpts(width = "1050px", height = "250px")
rosechart.set_colors(color_series)
rosechart.add("Dual Sim", [list(z) for z in zip(c,d)], radius = ["45%","95%

rosechart.set_series_opts(label_opts=opts.LabelOpts(is_show=True, position=

rosechart.render_notebook()
```

Out[20]:

In [21]:
```
d = df['touch_screen'].value_counts()
c= ["Touch screen supported","Touch_screen Not supported"]
color_series = ["#BC8F8F","#FFC1C1 "]
rosechart = Pie(init_opts=opts.InitOpts(width = "1050px", height = "250px")
rosechart.set_colors(color_series)
rosechart.add("Touch Screen", [list(z) for z in zip(c,d)], radius = ["45%",

rosechart.set_series_opts(label_opts=opts.LabelOpts(is_show=True, position=

rosechart.render_notebook()
```

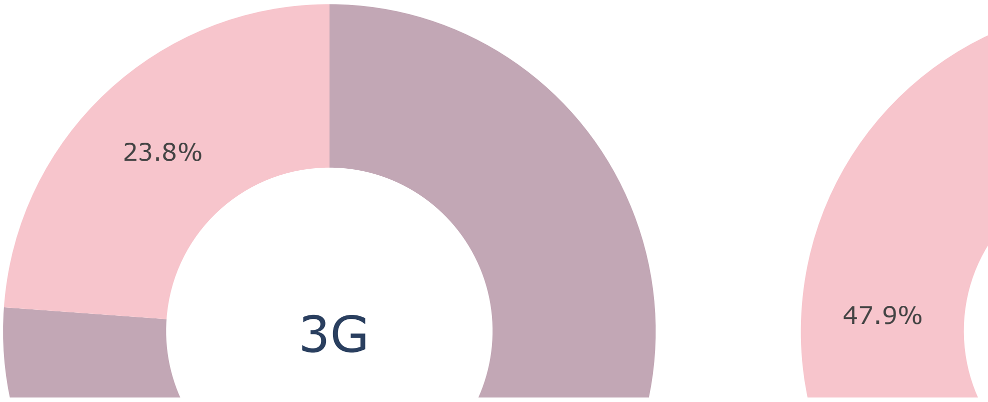
Out[21]:

In [22]:
```
d = df['wifi'].value_counts()
c= ["wifi supported","wifi Not supported"]
color_series = ["#CC313D","#F7C5CC "]
rosechart = Pie(init_opts=opts.InitOpts(width = "1050px", height = "250px")
rosechart.set_colors(color_series)
rosechart.add("", [list(z) for z in zip(c,d)], radius = ["45%","95%"],

rosechart.set_series_opts(label_opts=opts.LabelOpts(is_show=True, position=

rosechart.render_notebook()
```

Out[22]:

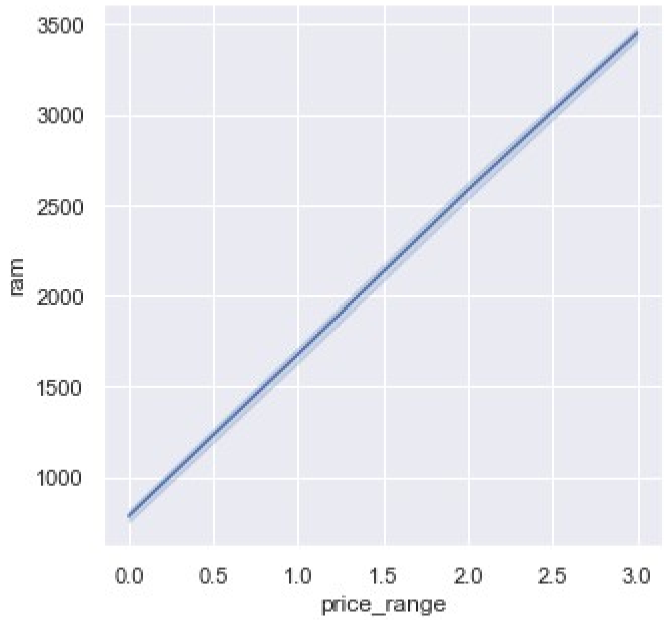
Subplot Donut Pie Chart

In [23]:
```
fig = make_subplots(rows=1, cols=2, specs=[[{'type' : 'domain'},{'type' : '

fig.add_trace(go.Pie(labels = ["3G supported","3G Not supported"], values= 
fig.add_trace(go.Pie(labels = ["4G supported","4G Not supported"], values= 

fig.update_traces(hole = .5, hoverinfo = "label+percent+name",marker = dict
fig.update_layout(annotations =[dict(text = "3G", x = 0.20, y = 0.5, font_s

fig.show()
```



In [24]:
```
sns.lineplot(data = df, x = "price_range", y = "ram" )
```

Out[24]:
```
<AxesSubplot:xlabel='price_range', ylabel='ram'>
```

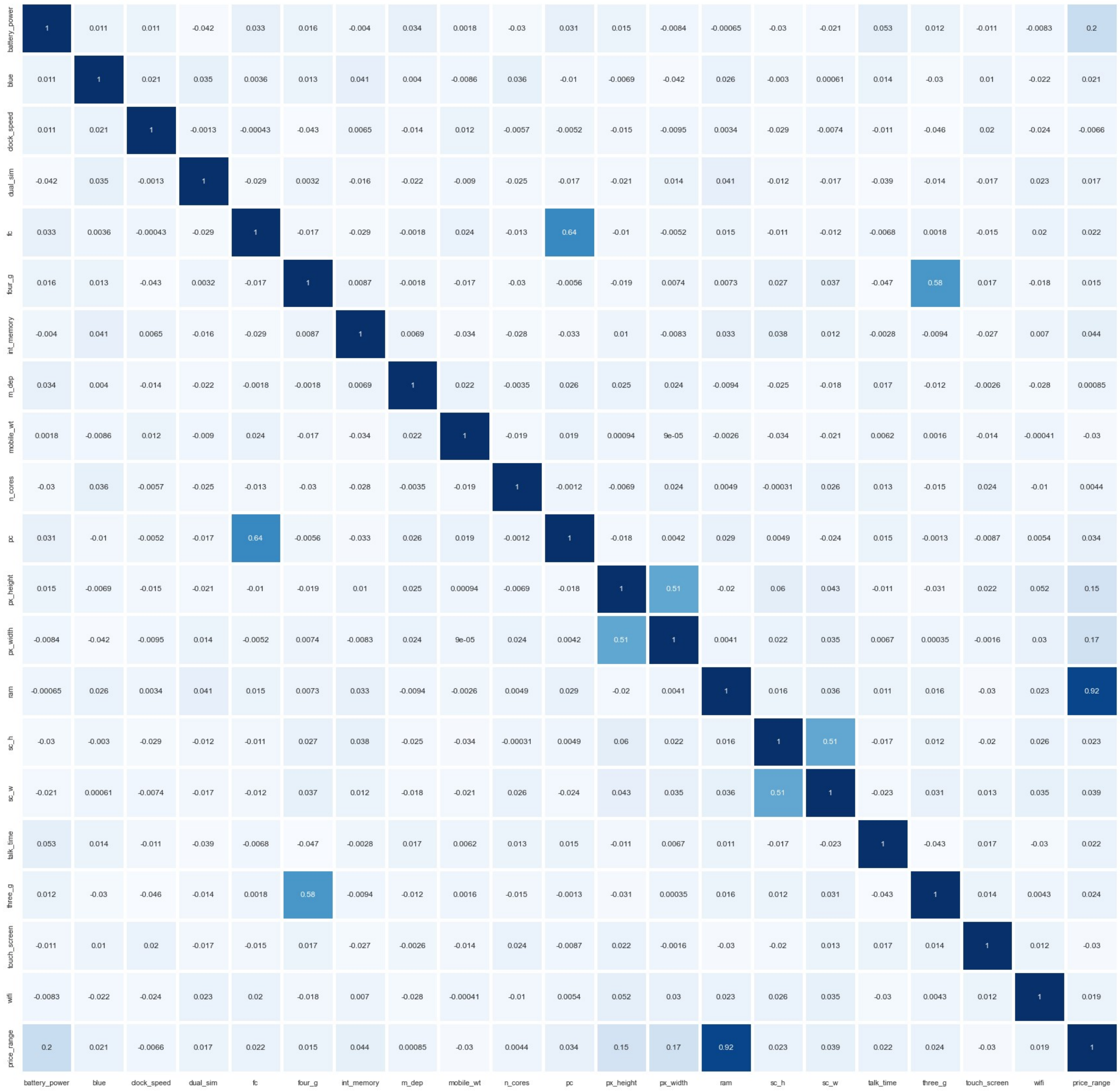


In [25]:
```
df.groupby(['blue','three_g','four_g','wifi'])['price_range'].count()
```

Out[25]:
```
blue  three_g  four_g  wifi
0     0        0       0       101
                       1       127
      1        0       0       123
                       1       139
               1       0       263
                       1       257
1     0        0       0       136
                       1       113
      1        0       0       103
                       1       115
               1       0       260
                       1       263
Name: price_range, dtype: int64
```

In [26]:
```python
cor = df.corr()
plt.figure(figsize = (30,30))
sns.heatmap(cor,annot=True,cmap="Blues",cbar = False,linewidths = 5)
```

Out[26]: <AxesSubplot:>



In [27]:
```
x = df.iloc[:, :-1]
x.head().T
```

Out[27]:

	0	1	2	3	4
battery_power	842.0	1021.0	563.0	615.0	1821.0
blue	0.0	1.0	1.0	1.0	1.0
clock_speed	2.2	0.5	0.5	2.5	1.2
dual_sim	0.0	1.0	1.0	0.0	0.0
fc	1.0	0.0	2.0	0.0	13.0
four_g	0.0	1.0	1.0	0.0	1.0
int_memory	7.0	53.0	41.0	10.0	44.0
m_dep	0.6	0.7	0.9	0.8	0.6
mobile_wt	188.0	136.0	145.0	131.0	141.0
n_cores	2.0	3.0	5.0	6.0	2.0
pc	2.0	6.0	6.0	9.0	14.0
px_height	20.0	905.0	1263.0	1216.0	1208.0
px_width	756.0	1988.0	1716.0	1786.0	1212.0
ram	2549.0	2631.0	2603.0	2769.0	1411.0
sc_h	9.0	17.0	11.0	16.0	8.0
sc_w	7.0	3.0	2.0	8.0	2.0
talk_time	19.0	7.0	9.0	11.0	15.0
three_g	0.0	1.0	1.0	1.0	1.0
touch_screen	0.0	1.0	1.0	0.0	1.0
wifi	1.0	0.0	0.0	0.0	0.0

In [28]:
```
y = df.iloc[:, -1]
y.head().T
```

Out[28]:
```
0    1
1    2
2    2
3    2
4    1
Name: price_range, dtype: int64
```

In [29]:
```
ss = StandardScaler()

x = ss.fit_transform(x)
```

In [30]:
```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.30, r
```

In [31]:
```
x_test[0:5]
```

Out[31]:
```
array([[-0.46554659, -0.9900495 ,  1.32109556,  0.98117712, -0.53209893,
         0.95788598,  0.71403853, -0.69968647,  1.40576067, -0.66476784,
        -0.97586945, -1.04381219,  0.09600863, -1.0652418 ,  0.63945335,
        -1.0945264 , -0.73426721,  0.55964063, -1.00601811, -1.01409939],
       [-0.53838837,  1.0100505 ,  0.34046327, -1.01918398, -0.76249466,
        -1.04396559,  0.71403853,  0.68754816,  1.12320139,  1.08404594,
        -0.48104847,  0.68269683, -0.5658884 , -0.82088073,  1.58907778,
         1.660732  ,  1.27942995,  0.55964063,  0.99401789, -1.01409939],
       [-1.43297644,  1.0100505 , -1.2530642 , -1.01918398, -0.07130748,
         0.95788598, -1.2152739 , -1.39330378, -1.67413547,  0.6468425 ,
        -0.64598879, -1.19933324,  0.63061776, -0.20214008,  0.87685946,
        -0.63531667,  1.27942995,  0.55964063,  0.99401789,  0.98609664],
       [ 0.61797484, -0.9900495 , -1.13048516,  0.98117712, -0.76249466,
        -1.04396559, -0.00256323,  0.68754816, -0.14831537, -0.66476784,
        -0.31610814,  0.71650575,  0.62136046, -1.17128528, -1.0223894 ,
        -0.86492153, -0.18507707,  0.55964063, -1.00601811,  0.98609664],
       [-0.98682056,  1.0100505 ,  0.21788424,  0.98117712, -0.99289039,
         0.95788598,  1.43064028,  1.38116548,  0.78413025, -0.66476784,
         0.01377252, -1.27145894,  0.67690426,  1.17365881, -1.25979551,
         0.05349793, -1.6495841 ,  0.55964063, -1.00601811, -1.01409939]])
```

In [32]:
```
y_test[0:5]
```

Out[32]:
```
674     0
1699    0
1282    1
1315    1
1210    2
Name: price_range, dtype: int64
```

Target column is

KNN Model

In [33]:
```
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(x_train, y_train)
y_pred = knn.predict(x_test)
```

```
---------------------------------------------------------------------------
-
AttributeError                            Traceback (most recent call las
t)
Input In [33], in <cell line: 3>()
      1 knn = KNeighborsClassifier(n_neighbors=3)
      2 knn.fit(x_train, y_train)
----> 3 y_pred = knn.predict(x_test)

File ~\anaconda3\lib\site-packages\sklearn\neighbors\_classification.py:23
4, in KNeighborsClassifier.predict(self, X)
    218 """Predict the class labels for the provided data.
    219
    220 Parameters
   (...)
    229     Class labels for each data sample.
    230 """
    231 if self.weights == "uniform":
    232     # In that case, we do not need the distances to perform
    233     # the weighting so we do not compute them.
--> 234     neigh_ind = self.kneighbors(X, return_distance=False)
    235     neigh_dist = None
    236 else:

File ~\anaconda3\lib\site-packages\sklearn\neighbors\_base.py:824, in KNei
ghborsMixin.kneighbors(self, X, n_neighbors, return_distance)
    817 use_pairwise_distances_reductions = (
    818     self._fit_method == "brute"
    819     and ArgKmin.is_usable_for(
    820         X if X is not None else self._fit_X, self._fit_X, self.eff
ective_metric_
    821     )
    822 )
    823 if use_pairwise_distances_reductions:
--> 824     results = ArgKmin.compute(
    825         X=X,
    826         Y=self._fit_X,
    827         k=n_neighbors,
    828         metric=self.effective_metric_,
    829         metric_kwargs=self.effective_metric_params_,
    830         strategy="auto",
    831         return_distance=return_distance,
    832     )
    834 elif (
    835     self._fit_method == "brute" and self.metric == "precomputed" a
nd issparse(X)
    836 ):
    837     results = _kneighbors_from_graph(
    838         X, n_neighbors=n_neighbors, return_distance=return_distanc
e
    839     )

File ~\anaconda3\lib\site-packages\sklearn\metrics\_pairwise_distances_red
uction\_dispatcher.py:277, in ArgKmin.compute(cls, X, Y, k, metric, chunk_
size, metric_kwargs, strategy, return_distance)
```

```
    196 """Compute the argkmin reduction.
    197 
    198 Parameters
   (...)
    274 returns.
    275 """
    276 if X.dtype == Y.dtype == np.float64:
--> 277     return ArgKmin64.compute(
    278         X=X,
    279         Y=Y,
    280         k=k,
    281         metric=metric,
    282         chunk_size=chunk_size,
    283         metric_kwargs=metric_kwargs,
    284         strategy=strategy,
    285         return_distance=return_distance,
    286     )
    288 if X.dtype == Y.dtype == np.float32:
    289     return ArgKmin32.compute(
    290         X=X,
    291         Y=Y,
   (...)
    297         return_distance=return_distance,
    298     )

File sklearn\metrics\_pairwise_distances_reduction\_argkmin.pyx:95, in skl
earn.metrics._pairwise_distances_reduction._argkmin.ArgKmin64.compute()

File ~\anaconda3\lib\site-packages\sklearn\utils\fixes.py:139, in threadpo
ol_limits(limits, user_api)
    137     return controller.limit(limits=limits, user_api=user_api)
    138 else:
--> 139     return threadpoolctl.threadpool_limits(limits=limits, user_api
=user_api)

File ~\anaconda3\lib\site-packages\threadpoolctl.py:171, in threadpool_lim
its.__init__(self, limits, user_api)
    167 def __init__(self, limits=None, user_api=None):
    168     self._limits, self._user_api, self._prefixes = \
    169         self._check_params(limits, user_api)
--> 171     self._original_info = self._set_threadpool_limits()

File ~\anaconda3\lib\site-packages\threadpoolctl.py:268, in threadpool_lim
its._set_threadpool_limits(self)
    265 if self._limits is None:
    266     return None
--> 268 modules = _ThreadpoolInfo(prefixes=self._prefixes,
    269                           user_api=self._user_api)
    270 for module in modules:
    271     # self._limits is a dict {key: num_threads} where key is eithe
r
    272     # a prefix or a user_api. If a module matches both, the limit
    273     # corresponding to the prefix is chosed.
    274     if module.prefix in self._limits:

File ~\anaconda3\lib\site-packages\threadpoolctl.py:340, in _ThreadpoolInf
o.__init__(self, user_api, prefixes, modules)
    337     self.user_api = [] if user_api is None else user_api
    339     self.modules = []
--> 340     self._load_modules()
```

```
    341     self._warn_if_incompatible_openmp()
    342 else:

File ~\anaconda3\lib\site-packages\threadpoolctl.py:373, in _ThreadpoolInf
o._load_modules(self)
    371     self._find_modules_with_dyld()
    372 elif sys.platform == "win32":
--> 373     self._find_modules_with_enum_process_module_ex()
    374 else:
    375     self._find_modules_with_dl_iterate_phdr()

File ~\anaconda3\lib\site-packages\threadpoolctl.py:485, in _ThreadpoolInf
o._find_modules_with_enum_process_module_ex(self)
    482         filepath = buf.value
    484         # Store the module if it is supported and selected
--> 485         self._make_module_from_path(filepath)
    486 finally:
    487     kernel_32.CloseHandle(h_process)

File ~\anaconda3\lib\site-packages\threadpoolctl.py:515, in _ThreadpoolInf
o._make_module_from_path(self, filepath)
    513 if prefix in self.prefixes or user_api in self.user_api:
    514     module_class = globals()[module_class]
--> 515     module = module_class(filepath, prefix, user_api, internal_ap
i)
    516     self.modules.append(module)

File ~\anaconda3\lib\site-packages\threadpoolctl.py:606, in _Module.__init
__(self, filepath, prefix, user_api, internal_api)
    604 self.internal_api = internal_api
    605 self._dynlib = ctypes.CDLL(filepath, mode=_RTLD_NOLOAD)
--> 606 self.version = self.get_version()
    607 self.num_threads = self.get_num_threads()
    608 self._get_extra_info()

File ~\anaconda3\lib\site-packages\threadpoolctl.py:646, in _OpenBLASModul
e.get_version(self)
    643 get_config = getattr(self._dynlib, "openblas_get_config",
    644                      lambda: None)
    645 get_config.restype = ctypes.c_char_p
--> 646 config = get_config().split()
    647 if config[0] == b"OpenBLAS":
    648     return config[1].decode("utf-8")

AttributeError: 'NoneType' object has no attribute 'split'
```

In [ ]:
```
accuracy_score(y_test, y_pred)
```

In [ ]:
```
m = confusion_matrix(y_test, y_pred)
plt.figure(figsize = (5,5))
sns.heatmap(m,annot=True,cmap="binary",cbar = False,linewidths = 5)
```

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

In [ ]:
```
ac_list = []
for i in range(1,30):
    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(x_train, y_train)
    y_pred = knn.predict(x_test)

    ac = accuracy_score(y_test, y_pred)
    ac_list.append(ac)

ac_list
```

In [ ]:
```
plt.plot(range(1,30), ac_list,':ok' )
plt.show()
```

In [ ]:
```
knn = KNeighborsClassifier(n_neighbors=25)
knn.fit(x_train, y_train)
y_pred = knn.predict(x_test)
```

In [ ]:
```
accuracy_score(y_test, y_pred)
```

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

In [ ]:
```
m = confusion_matrix(y_test, y_pred)
plt.figure(figsize = (5,5))
sns.heatmap(m,annot=True,cmap="binary",cbar = False,linewidths = 5)
```

Logistic Regression Model

In [ ]:
```
logreg = LogisticRegression()
logreg.fit(x_train,y_train)
y_pred = logreg.predict(x_test)
print(accuracy_score(y_test, y_pred))
```

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

In [ ]:
```
from tabulate import tabulate

params = [ ['lbfgs','l2'],['lbfgs','none'],
           ['liblinear','l1'],['liblinear','l2'],
           ['newton-cg','l2'],['newton-cg','none'],
           ['sag','l2'],['sag','none'],
           ['saga','l1'],['saga','l2'],['saga','none'] ]


all_combinations = []

for i in params:

    from sklearn.linear_model import LogisticRegression

    lr = LogisticRegression(solver=i[0] , penalty=i[1])

    lr.fit(x_train,y_train)

    y_pred = lr.predict(x_test)

    from sklearn.metrics import accuracy_score
    acc = accuracy_score(y_test,y_pred)


    all_combinations.append([i[0],i[1],acc])
head = ['Solver','Penalty','Accuracy' ]
print(tabulate(all_combinations,headers=head,tablefmt="grid"))
```

In [ ]:
```
logreg = LogisticRegression(solver="lbfgs", penalty ="none")
logreg.fit(x_train,y_train)
y_pred = logreg.predict(x_test)
print(accuracy_score(y_test, y_pred))
```

In [ ]:
```
m = confusion_matrix(y_test, y_pred)
plt.figure(figsize = (5,5))
sns.heatmap(m,annot=True,cmap="binary",cbar = False,linewidths = 5)
```

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

Decision Tree Model

In [ ]:
```
dt = DecisionTreeClassifier()
dt.fit(x_train, y_train)
y_pred = dt.predict(x_test)
print(accuracy_score(y_test, y_pred))

print(classification_report(y_test,y_pred))
```

In [ ]:
```
dt_grid = {"min_samples_split" : range(2,50),
                "max_depth": range(1,20),
               "criterion" : ["gini","entropy"]}
```

In [ ]:
```
grid = GridSearchCV(dt, dt_grid, verbose=3)
grid.fit(x_train, y_train)
```

In [ ]:
```
grid.best_params_
```

In [ ]:
```
dt = DecisionTreeClassifier(max_depth = 6, min_samples_split = 30, criterio
dt.fit(x_train, y_train)
y_pred = dt.predict(x_test)
print(accuracy_score(y_test, y_pred))

print(classification_report(y_test,y_pred))
```

In [ ]:
```
m = confusion_matrix(y_test, y_pred)
plt.figure(figsize = (5,5))
sns.heatmap(m,annot=True,cmap="binary",cbar = False,linewidths = 5)
```

SVM model

In [ ]:
```
svm = SVC()
svm.fit(x_train, y_train)
y_pred = svm.predict(x_test)
print(classification_report(y_test, y_pred))
```

In [ ]:
```
svm_param = {"kernel" : ['linear', 'rbf', 'sigmoid'],
                 "gamma": [0.001, 0.01, 0.1, 1],
                 "C": [0.1, 1,10,100]}
```

In [ ]:
```
grid = GridSearchCV(svm, svm_param, verbose=3)
grid.fit(x_train, y_train)
```

In [ ]:
```
grid.best_params_
```

In [ ]:
```
svm = SVC( C=100, kernel='linear',  gamma=0.001)
svm.fit(x_train, y_train)
y_pred = svm.predict(x_test)
print(accuracy_score(y_test, y_pred))
```

In [ ]:
```
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

In [ ]:
```
m = confusion_matrix(y_test, y_pred)
plt.figure(figsize = (5,5))
sns.heatmap(m,annot=True,cmap="binary",cbar = False,linewidths = 5)
```

Ensemble Learning

1. Bgging

1. Boosting

1. Voting

Bagging

knn

In [ ]:
```
bg = BaggingClassifier(KNeighborsClassifier(n_neighbors=25))
bg.fit(x_train,y_train)
y_pred = bg.predict(x_test)
print(accuracy_score(y_test, y_pred))

print(classification_report(y_test, y_pred))
```

In [ ]:
```
m = confusion_matrix(y_test, y_pred)
plt.figure(figsize = (5,5))
sns.heatmap(m,annot=True,cmap="binary",cbar = False,linewidths = 5)
```

Logistic Regression

In [ ]:
```
bg = BaggingClassifier(LogisticRegression(solver="lbfgs", penalty ="none"))
bg.fit(x_train,y_train)
y_pred = bg.predict(x_test)
print(accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

In [ ]:
```
m = confusion_matrix(y_test, y_pred)
plt.figure(figsize = (5,5))
sns.heatmap(m,annot=True,cmap="binary",cbar = False,linewidths = 5)
```

Decision Tree

In [ ]:
```
bg=BaggingClassifier(DecisionTreeClassifier(max_depth = 6, min_samples_spli
bg.fit(x_train,y_train)
y_pred=bg.predict(x_test)
print(accuracy_score(y_test, y_pred))

print(classification_report(y_test,y_pred))
```

In [ ]:
```
m = confusion_matrix(y_test, y_pred)
plt.figure(figsize = (5,5))
sns.heatmap(m,annot=True,cmap="binary",cbar = False,linewidths = 5)
```

SVM

In [ ]:
```
bg=BaggingClassifier(SVC( C=100, kernel='linear',  gamma=0.001))
bg.fit(x_train,y_train)
y_pred=bg.predict(x_test)
print(accuracy_score(y_test, y_pred))

print(classification_report(y_test,y_pred))
```

In [ ]:
```
m = confusion_matrix(y_test, y_pred)
plt.figure(figsize = (5,5))
sns.heatmap(m,annot=True,cmap="binary",cbar = False,linewidths = 5)
```

Ramdom Forest

In [ ]:
```
rf = RandomForestClassifier()
rf.fit(x_train, y_train)
y_pred = rf.predict(x_test)
print(accuracy_score(y_test, y_pred))

print(classification_report(y_test, y_pred))
```

In [ ]:
```
m = confusion_matrix(y_test, y_pred)
plt.figure(figsize = (5,5))
sns.heatmap(m,annot=True,cmap="binary",cbar = False,linewidths = 5)
```

Voting Classifier

In [ ]:
```
models = [('logistic regression', LogisticRegression(solver="lbfgs", penalt
 ('Decision Tree', DecisionTreeClassifier(max_depth = 6, min_samples_split
```

In [ ]:
```
vc = VotingClassifier(estimators=models)
vc.fit(x_train, y_train)
y_pred = vc.predict(x_test)
print(accuracy_score(y_test, y_pred))

print(classification_report(y_test, y_pred))
```

In [ ]:
```
m = confusion_matrix(y_test, y_pred)
plt.figure(figsize = (5,5))
sns.heatmap(m,annot=True,cmap="binary",cbar = False,linewidths = 5)
```

Ada Boost

In [ ]:
```
adb = AdaBoostClassifier()
adb.fit(x_train, y_train)
y_pred = adb.predict(x_test)
print(accuracy_score(y_test, y_pred))

print(classification_report(y_test, y_pred))
```

In [ ]:
```
m = confusion_matrix(y_test, y_pred)
plt.figure(figsize = (5,5))
sns.heatmap(m,annot=True,cmap="binary",cbar = False,linewidths = 5)
```

Gradient Boosting

In [ ]:
```
gbc = GradientBoostingClassifier()
gbc.fit(x_train, y_train)
y_pred = gbc.predict(x_test)
print(accuracy_score(y_test, y_pred))

print(classification_report(y_test, y_pred))
```

In [ ]:
```
m = confusion_matrix(y_test, y_pred)
plt.figure(figsize = (5,5))
sns.heatmap(m,annot=True,cmap="binary",cbar = False,linewidths = 5)
```

xgboost

In [ ]:
```
xgb = XGBClassifier()
xgb.fit(x_train, y_train)
y_pred = xgb.predict(x_test)
print(accuracy_score(y_test, y_pred))

print(classification_report(y_test, y_pred))
```

In [ ]:
```
m = confusion_matrix(y_test, y_pred)
plt.figure(figsize = (5,5))
sns.heatmap(m,annot=True,cmap="binary",cbar = False,linewidths = 5)
```

Best Model for prediction : Logistic Regression

In [ ]:
```
logreg = LogisticRegression(solver="lbfgs", penalty ="none",multi_class="mu
logreg.fit(x_train,y_train)
y_pred = logreg.predict(x_test)
print(accuracy_score(y_test, y_pred))

print(classification_report(y_test,y_pred))
```

In [ ]:
```
m = confusion_matrix(y_test, y_pred)
plt.figure(figsize = (5,5))
sns.heatmap(m,annot=True,cmap="binary",cbar = False,linewidths = 5)
```

In [ ]:
```
ypredprob = logreg.predict_proba(x_test)
ypredprob.shape
```

In [ ]:
```
from sklearn.multiclass import OneVsRestClassifier
from sklearn.metrics import roc_curve, auc
```

In [ ]:
```
from sklearn.preprocessing import label_binarize
classes=df['price_range'].unique()
y_tesr_bin = label_binarize(y_test, classes=classes)

fpr = {}
tpr ={}
thresh = {}
roc_auc = dict()

n_class = classes.shape[0]
for i in range(n_class):
    fpr[i],tpr[i],thresh[i] = roc_curve(y_tesr_bin[:,1],ypredprob[:,i])
    roc_auc[i] = auc(fpr[i], tpr[i])

    plt.plot(fpr[i], tpr[i], linestyle = "--", label = "%s vs Rest (AUC=%0.


plt.plot([0,1], [0,1], 'b--', color='#FFFF00')
plt.xlim([0,1])
plt.ylim([0,1.05])
plt.title("Multiclass Roc Curve")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.legend(loc = "lower right")
```

In [ ]:

In [ ]:

In [ ]: