Business Problem :

a mobile's factory owner needs to estimate the price ranges of mobile based on deferent mobile Specification

data dictionary :

- id:ID
- 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 Megabytes
- 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

In [1]:

```
#pip install cufflinks
```

In [2]:

```
#pip install chart_studio
```

In [3]:

```
#pip install plotly
```

In [4]:

```
#pip install lightgbm
```

In [5]:

```
#pip install pandas_profiling
```

In [6]:

```
import numpy as np
import pandas as pd
from pandas_profiling import ProfileReport
import seaborn as sns
import matplotlib.pyplot as plt
sns.set_style("darkgrid")
```

In [7]:

```
from collections import Counter
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import RobustScaler
from sklearn.preprocessing import label_binarize
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import VotingClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.svm import SVC
from sklearn.svm import LinearSVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import SGDClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
from sklearn.model_selection import train_test_split
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import confusion_matrix
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.metrics import roc_auc_score
from sklearn.metrics import auc
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import accuracy_score
from sklearn.metrics import mean_squared_error
from sklearn.metrics import f1_score
from sklearn.metrics import roc_curve
from sklearn.multiclass import OneVsRestClassifier
from lightgbm import LGBMClassifier
```

In [8]:

```
import warnings
warnings.filterwarnings("ignore")
import chart_studio.plotly as py
import cufflinks as cf
import plotly.express as px
%matplotlib inline
from plotly.offline import download_plotlyjs, init_notebook_mode, plot, iplot
init_notebook_mode(connected = True)
cf.go_offline();
import plotly.graph_objs as go
import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

Load and Check Data

In [9]:

```
df = pd.read_csv('train.csv')
df.head(10)
```

Out[9]:

	battery_power	blue	clock_speed	dual_sim	fc	four_g	int_memory	m_dep	mobile_wt	n_
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	
5	1859	0	0.5	1	3	0	22	0.7	164	
6	1821	0	1.7	0	4	1	10	0.8	139	
7	1954	0	0.5	1	0	0	24	0.8	187	
8	1445	1	0.5	0	0	0	53	0.7	174	
9	509	1	0.6	1	2	1	9	0.1	93	

10 rows × 21 columns

In [10]:

```
1 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 [11]:

```
1 df.profile_report()
```

Summarize dataset: 34/34 [00:45<00:00, 1.33s/it, 100% Completed]

Generate report structure: 1/1 [00:27<00:00, 27.49s/it] 100%

Render HTML: 100% 1/1 [00:17<00:00, 17.97s/it]

Overview

Dataset statistics

Number of variables	21
Number of observations	2000
Missing cells	0
Missing cells (%)	0.0%
Duplicate rows	0
Duplicate rows (%)	0.0%
Total size in memory	328.2 KiB
Average record size in memory	168.1 B

Variable types

Numeric	14
Categorical	7

Warnings

`fc` is highly correlated with `pc`	**High correlation**
`four_g` is highly correlated with `three_g`	**High correlation**
`pc` is highly correlated with `fc`	**High correlation**
`px_height` is highly correlated with `px_width`	**High correlation**

Out[11]:

In [ ]:

```
1
```

In [ ]:

```
1
```

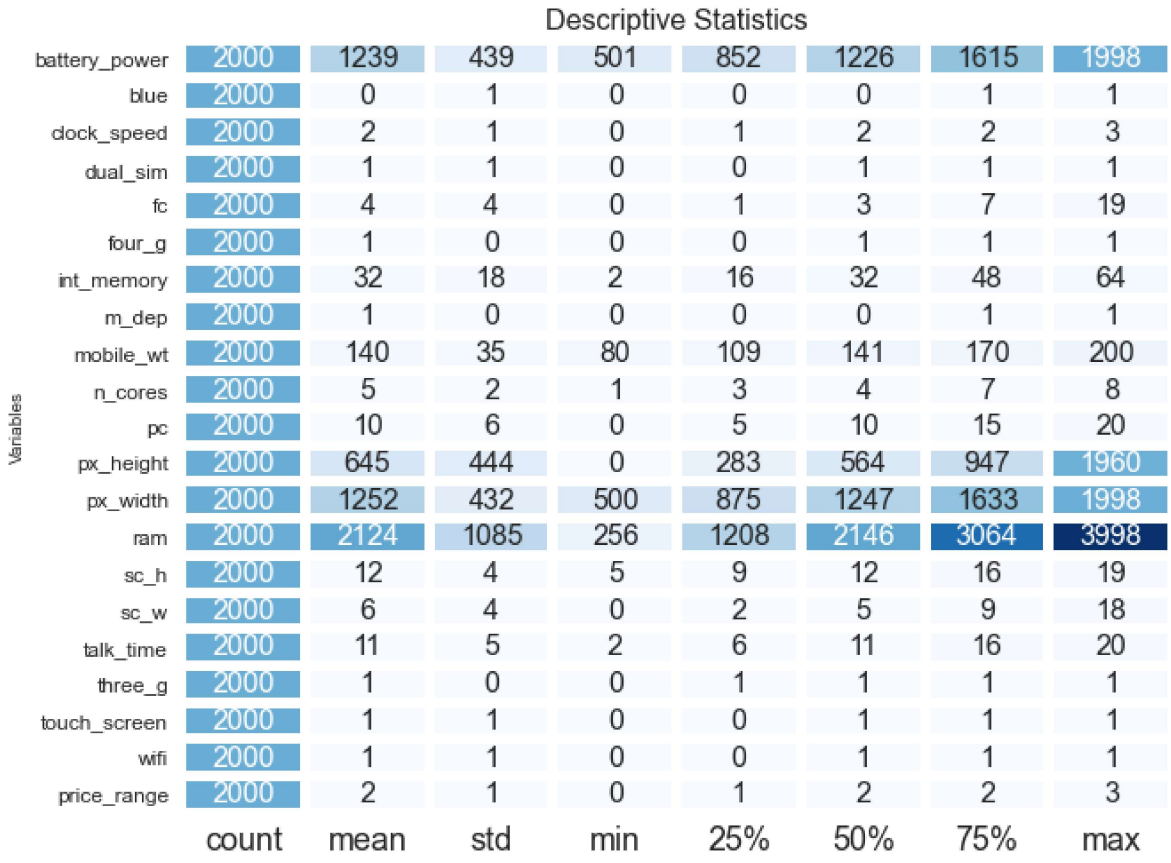
In [ ]:

```
1
```

Descriptive Statistics

In [12]:

```
1  desc = df.describe().T
2
3  df1 = pd.DataFrame(index=['battery_power','blue','clock_speed','dual_sim','fc','four_g'
4  'px_width','ram','sc_h','sc_w','talk_time','three_g','touch_screen','wifi','price_range
5                     columns= ["count","mean","std","min",
6                               "25%","50%","75%","max"], data= desc )
7
8  f,ax = plt.subplots(figsize=(10,8))
9
10 sns.heatmap(df1, annot=True,cmap = "Blues", fmt= '.0f',
11             ax=ax,linewidths = 5, cbar = False,
12             annot_kws={"size": 16})
13
14 plt.xticks(size = 18)
15 plt.yticks(size = 12, rotation = 0)
16 plt.ylabel("Variables")
17 plt.title("Descriptive Statistics", size = 16)
18 plt.show()
```



In [ ]:

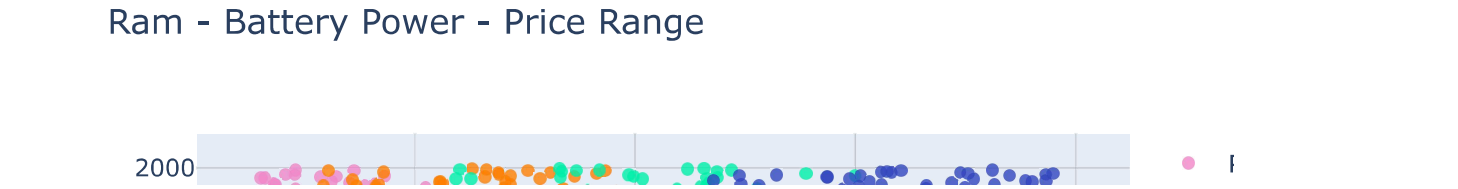
```
1
```

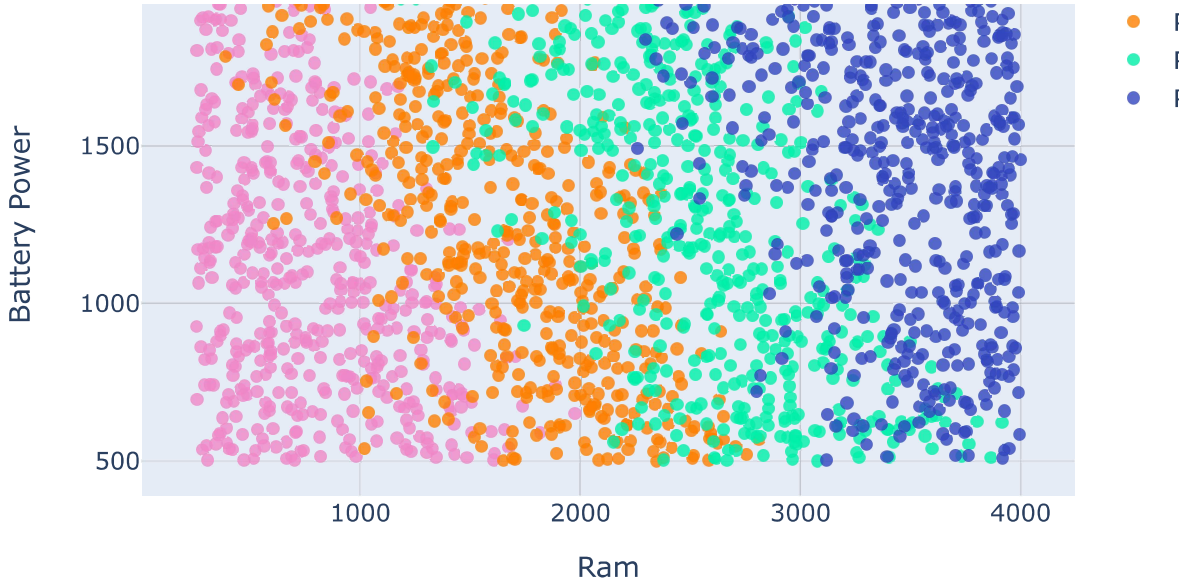
Data Analysis 📊📈

Ram - Battery Power - Price Range

In [13]:

```
df_pr_0 = df[df.price_range == 0]
df_pr_1 = df[df.price_range == 1]
df_pr_2 = df[df.price_range == 2]
df_pr_3 = df[df.price_range == 3]

trace1 =go.Scatter(
                    y = df_pr_0.battery_power,
                    x = df_pr_0.ram,
                    mode = "markers",
                    name = "Price Range: 0",
                    marker = dict(color = 'rgba(240, 136, 200, 0.8)'),
                    text= df_pr_0.price_range)

trace2 =go.Scatter(
                    y = df_pr_1.battery_power,
                    x = df_pr_1.ram,
                    mode = "markers",
                    name = "Price Range: 1",
                    marker = dict(color = 'rgba(255, 128, 2, 0.8)'),
                    text= df_pr_1.price_range)

trace3 =go.Scatter(
                    y = df_pr_2.battery_power,
                    x = df_pr_2.ram,
                    mode = "markers",
                    name = "Price Range: 2",
                    marker = dict(color = 'rgba(0, 240, 170, 0.8)'),
                    text= df_pr_2.price_range)

trace4 =go.Scatter(
                    y = df_pr_3.battery_power,
                    x = df_pr_3.ram,
                    mode = "markers",
                    name = "Price Range: 3",
                    marker = dict(color = 'rgba(50, 70, 190, 0.8)'),
                    text= df_pr_3.price_range)


data = [trace1, trace2, trace3, trace4]

layout = dict(title = 'Ram - Battery Power - Price Range',
              xaxis= dict(title= 'Ram',
                          ticklen= 5,zeroline= False),
              yaxis= dict(title= 'Battery Power',
                          ticklen= 5,zeroline= False),
             autosize=False,
             width=700,
             height=450,)
fig = dict(data = data, layout = layout)

iplot(fig)
```

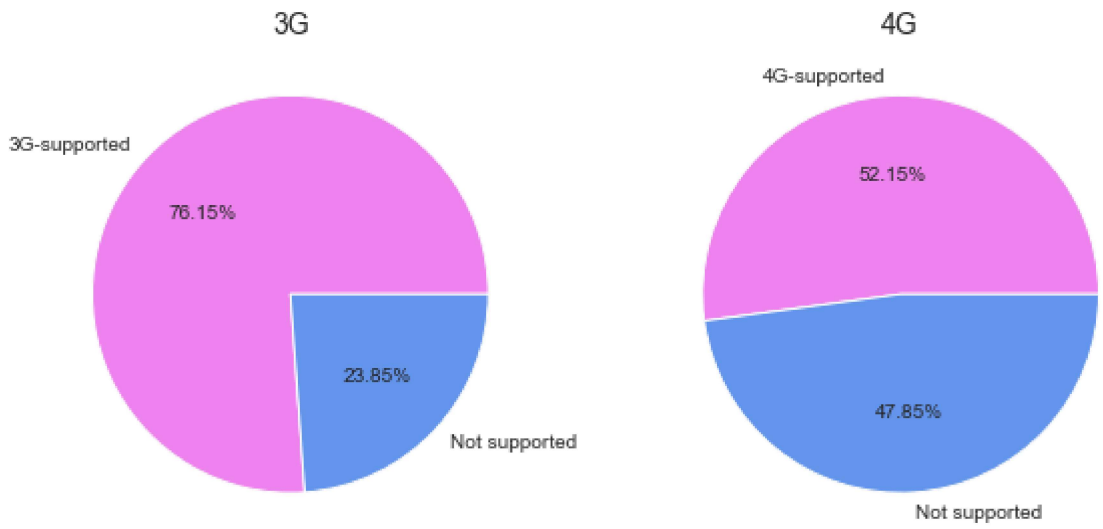




3G - 4G

In [14]:

```
values_3g = df["three_g"].value_counts()
values_4g = df["four_g"].value_counts()

labels4g = ["4G-supported",'Not supported']
labels3g = ["3G-supported",'Not supported']

colors = ["violet", "cornflowerblue"]

fig = plt.figure(figsize=(10,10))
ax1 = plt.subplot2grid((2,2),(0,0))
plt.pie(x= values_3g, autopct="%.2f%%",
        labels=labels3g, pctdistance=0.6,
        colors = colors)
plt.title('3G', size = 14)

ax1 = plt.subplot2grid((2,2), (0, 1))
plt.pie(x =values_4g,autopct="%.2f%%",
        labels=labels4g, pctdistance=0.6,
        colors = colors)
plt.title('4G', size = 14)
plt.show()
```



Price Range

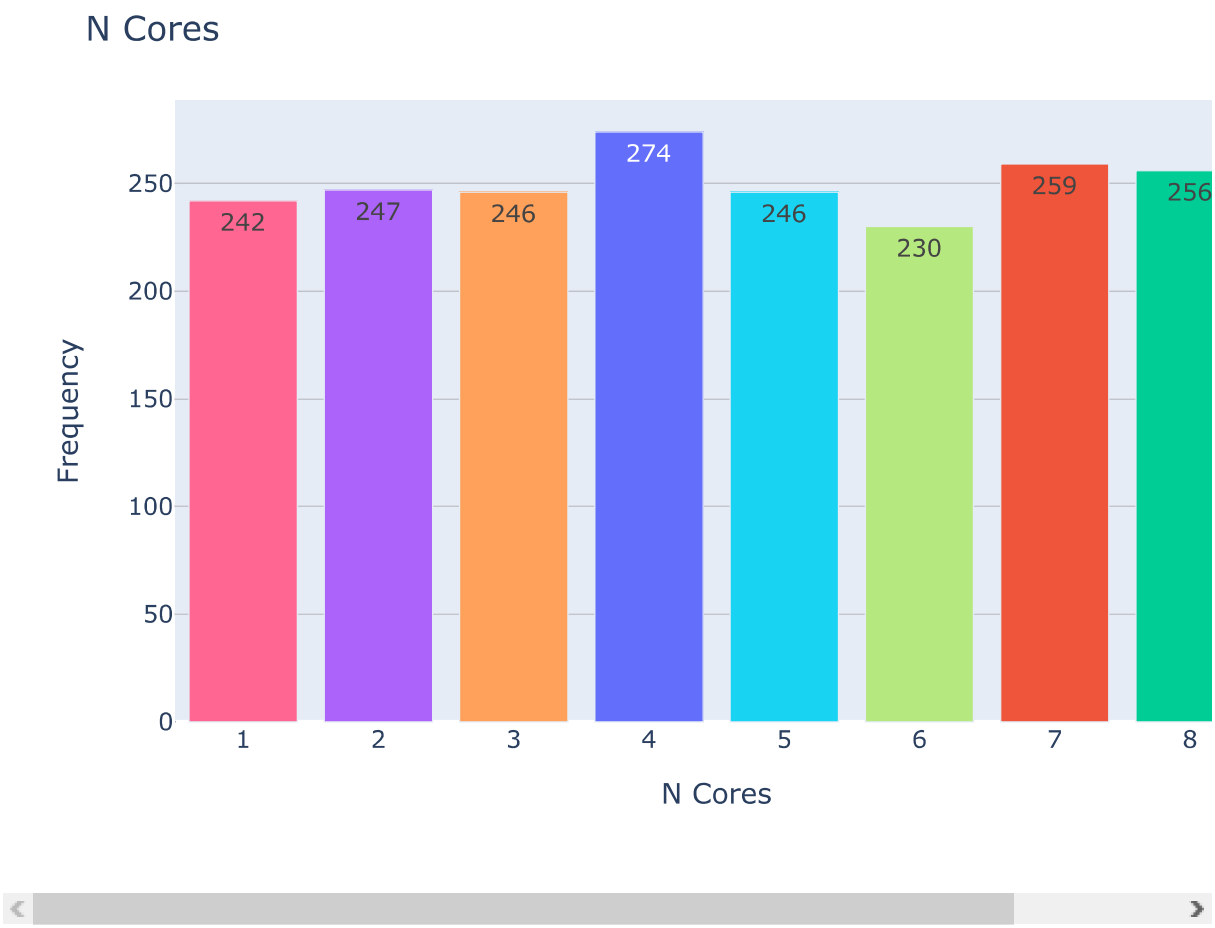
In [15]:

```
price_range = df["price_range"].value_counts()

fig = px.bar(price_range, y = 'price_range', text = 'price_range',

                    color = ["Price Range: 0","Price Range: 1",
                             "Price Range: 2","Price Range: 3"])


fig.update_layout(title = "Price Range", showlegend = False,
                  xaxis = dict(zeroline = False, tickmode = 'linear', tick0 = 0, dtick =
                  autosize=False,
                  width=700,
                  height=450,
                  xaxis_title_text = 'Price Range',
                  yaxis_title_text = 'Frequency')

fig
```

N Cores

In [16]:

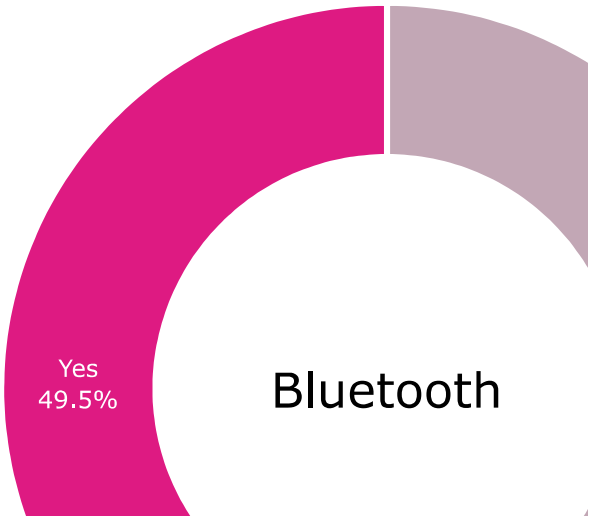
```
n_cores = df["n_cores"].value_counts()

fig = px.bar(n_cores, y = 'n_cores', text = 'n_cores',

                    color = ["4","7","8","2","3","5","1","6"])

fig.update_layout(title = "N Cores", showlegend = False,
                  xaxis = dict(zeroline = False, tickmode = 'linear', tick0 = 0, dtick =
                  autosize=False,
                  width=700,
                  height=450,
                  xaxis_title_text = 'N Cores',
                  yaxis_title_text = 'Frequency')

fig
```



Bluetooth

In [17]:

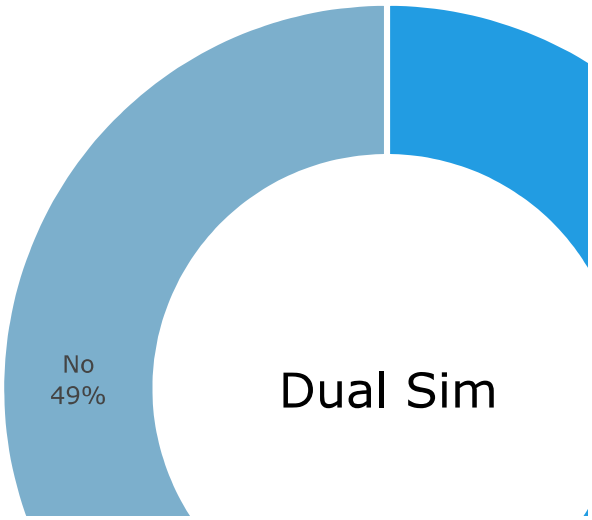
```
fig = px.pie(df['blue'].value_counts().reset_index(), values = 'blue',
             names = ['No', 'Yes'])
fig.update_traces(textposition = 'inside',
                  textinfo = 'percent + label',
                  hole = 0.6,
                  marker = dict(colors = ['#C2A7B5','#DE1A82'],
                                line = dict(color = 'white', width = 3)))

fig.update_layout(annotations = [dict(text = 'Bluetooth',
                                      x = 0.5, y = 0.5,
                                      font_size = 24, showarrow = False,
                                      font_family = 'Verdana',
                                      font_color = 'black')],
                  showlegend = False)

fig.show()
```



Dual Sim

In [18]:

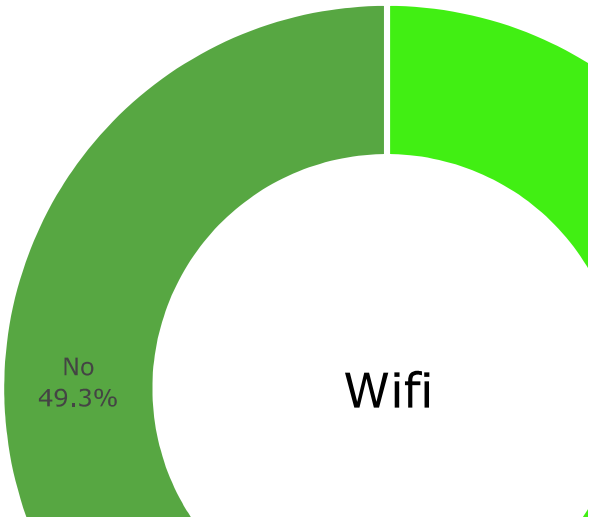
```
fig = px.pie(df['dual_sim'].value_counts().reset_index(), values = 'dual_sim',
             names = ['Yes', 'No'])
fig.update_traces(textposition = 'inside',
                  textinfo = 'percent + label',
                  hole = 0.6,
                  marker = dict(colors = ['#229CE2','#7CAFCC'],
                                line = dict(color = 'white', width = 3)))

fig.update_layout(annotations = [dict(text = 'Dual Sim',
                                      x = 0.5, y = 0.5,
                                      font_size = 24, showarrow = False,
                                      font_family = 'Verdana',
                                      font_color = 'black')],
                  showlegend = False)

fig.show()
```



Wifi

In [19]:

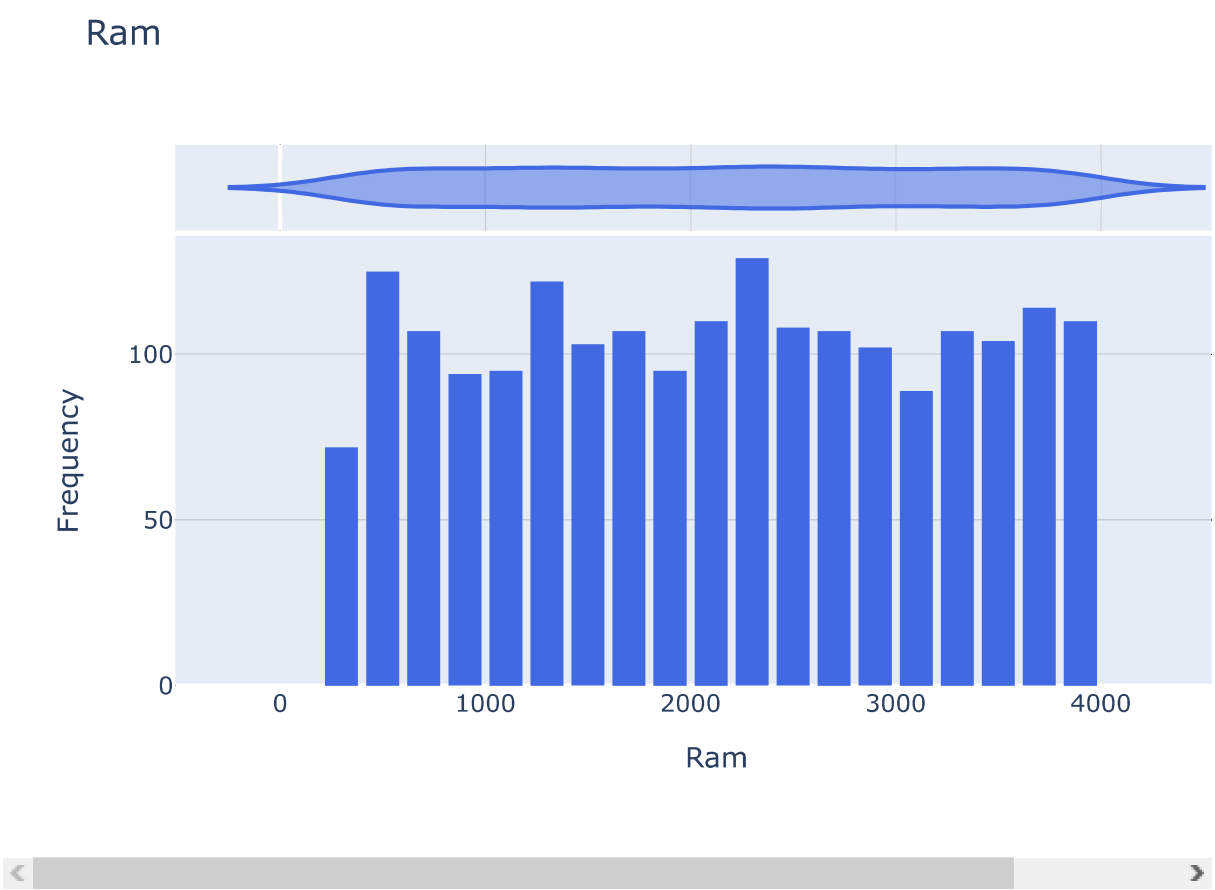
```
fig = px.pie(df['wifi'].value_counts().reset_index(), values = 'wifi',
             names = ['Yes', 'No'])
fig.update_traces(textposition = 'inside',
                  textinfo = 'percent + label',
                  hole = 0.6,
                  marker = dict(colors = ['#41EF13','#57A742'],
                                line = dict(color = 'white', width = 3)))

fig.update_layout(annotations = [dict(text = 'Wifi',
                                      x = 0.5, y = 0.5,
                                      font_size = 24, showarrow = False,
                                      font_family = 'Verdana',
                                      font_color = 'black')],
                  showlegend = False)

fig.show()

```



RAM

In [20]:

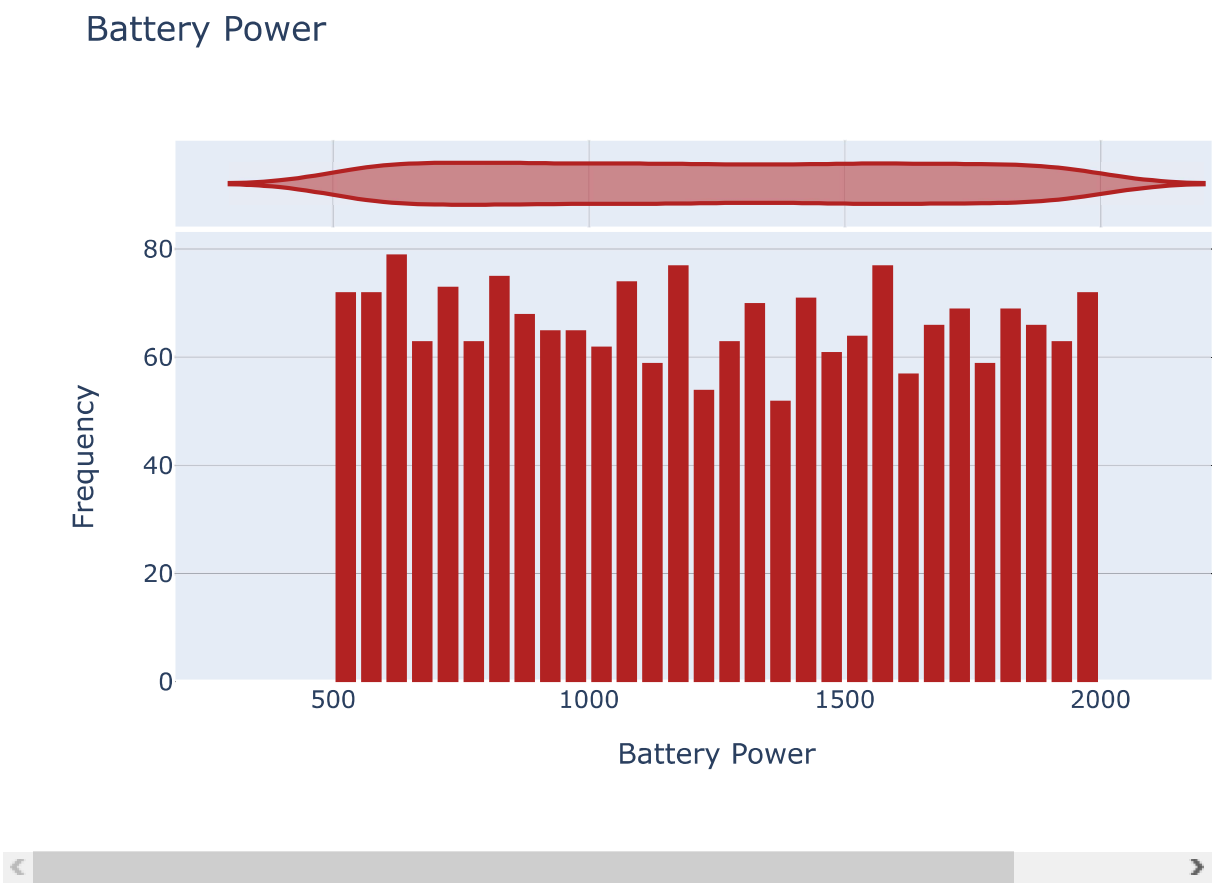
```
fig = px.histogram(df,x = 'ram',
                   title = 'Ram',
                    marginal = 'violin',
                   color_discrete_sequence = ['royalblue'])
fig.update_layout(
xaxis_title_text = 'Ram',
yaxis_title_text = 'Frequency',
bargap = 0.2, showlegend = False,
              autosize=False,
              width=700,
              height=450)

iplot(fig)
```



Battery Power

In [21]:

```
fig = px.histogram(df,x = 'battery_power',
                   title = 'Battery Power',
                    marginal = 'violin',
                   color_discrete_sequence = ['firebrick'])
fig.update_layout(
xaxis_title_text = 'Battery Power',
yaxis_title_text = 'Frequency',
bargap = 0.2, showlegend = False,
              autosize=False,
              width=700,
              height=450)

iplot(fig)

```



Ram - Price Range

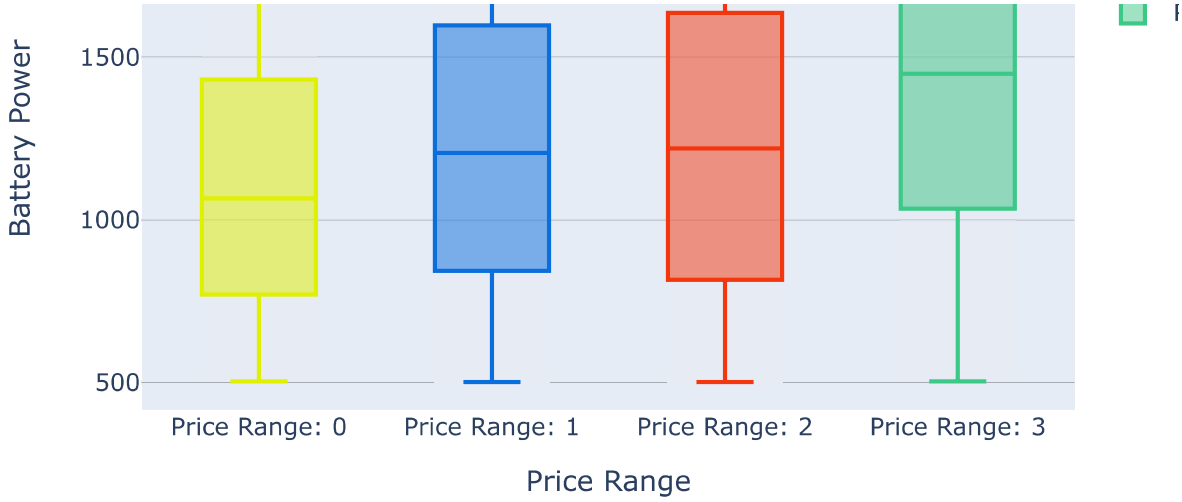
In [22]:

```
df_pr_0 = df[df.price_range == 0]
df_pr_1 = df[df.price_range == 1]
df_pr_2 = df[df.price_range == 2]
df_pr_3 = df[df.price_range == 3]

trace0 = go.Box(
    y=df_pr_0.ram,
    name = 'Price Range: 0',
    marker = dict(
        color = 'rgb(223, 240, 0)',
    )
)
trace1 = go.Box(
    y=df_pr_1.ram,
    name = 'Price Range: 1',
    marker = dict(
        color = 'rgb(10, 110, 220)',
    )
)
trace2 = go.Box(
    y=df_pr_2.ram,
    name = 'Price Range: 2',
    marker = dict(
        color = 'rgb(242, 54, 14)',
    )
)
trace3 = go.Box(
    y=df_pr_3.ram,
    name = 'Price Range: 3',
    marker = dict(
        color = 'rgb(60, 200, 135)',
    )
)

data = [trace0, trace1, trace2, trace3]

layout = dict(title = 'Ram - Price Range',
              xaxis= dict(title= 'Price Range',
                          ticklen= 5,zeroline= False),
              yaxis= dict(title= 'Ram',
                          ticklen= 5,zeroline= False),
             autosize=False,
             width=700,
             height=450)

fig = dict(data = data, layout = layout)

iplot(fig)
```

Battery Power - Price Range

In [23]:

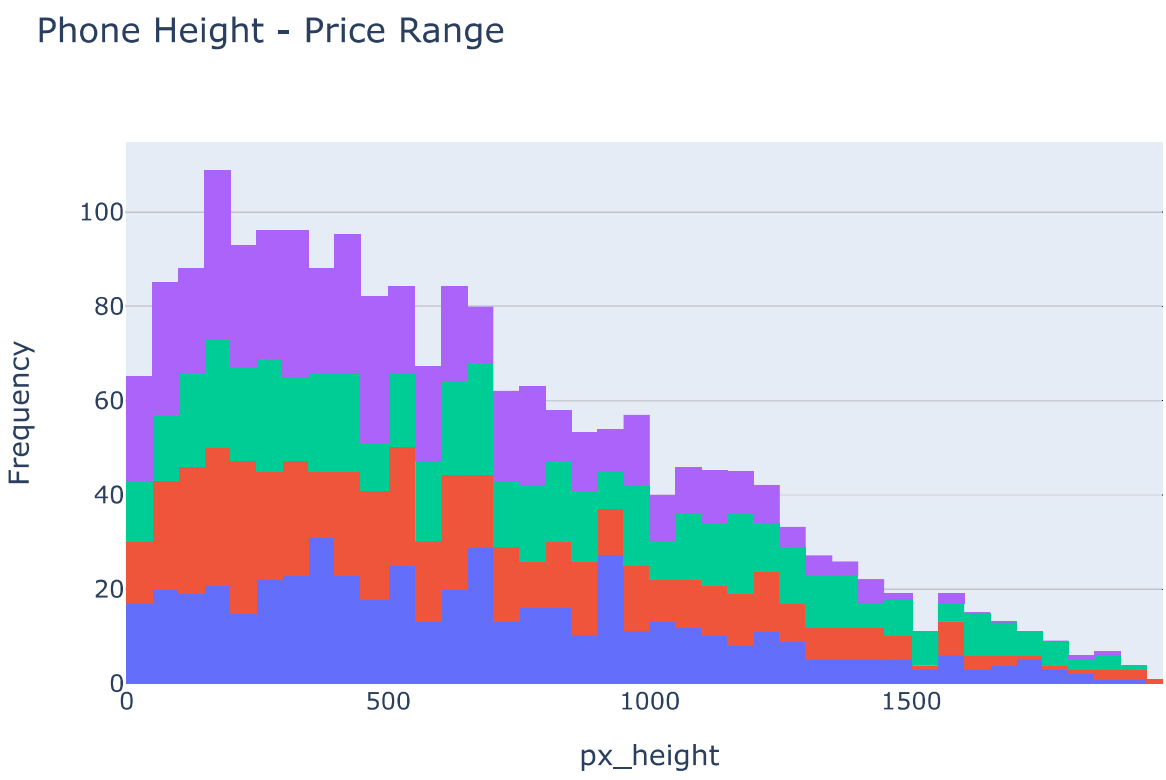
```
df_pr_0 = df[df.price_range == 0]
df_pr_1 = df[df.price_range == 1]
df_pr_2 = df[df.price_range == 2]
df_pr_3 = df[df.price_range == 3]

trace0 = go.Box(
    y=df_pr_0.battery_power,
    name = 'Price Range: 0',
    marker = dict(
        color = 'rgb(223, 240, 0)',
    )
)
trace1 = go.Box(
    y=df_pr_1.battery_power,
    name = 'Price Range: 1',
    marker = dict(
        color = 'rgb(10, 110, 220)',
    )
)
trace2 = go.Box(
    y=df_pr_2.battery_power,
    name = 'Price Range: 2',
    marker = dict(
        color = 'rgb(242, 54, 14)',
    )
)
trace3 = go.Box(
    y=df_pr_3.battery_power,
    name = 'Price Range: 3',
    marker = dict(
        color = 'rgb(60, 200, 135)',
    )
)

data = [trace0, trace1, trace2, trace3]

layout = dict(title = 'Battery Power - Price Range',
              xaxis= dict(title= 'Price Range',
                          ticklen= 5,zeroline= False),
              yaxis= dict(title= 'Battery Power',
                          ticklen= 5,zeroline= False),
             autosize=False,
             width=700,
             height=450)

fig = dict(data = data, layout = layout)

iplot(fig)
```



Phone Height - Price Range

In [24]:

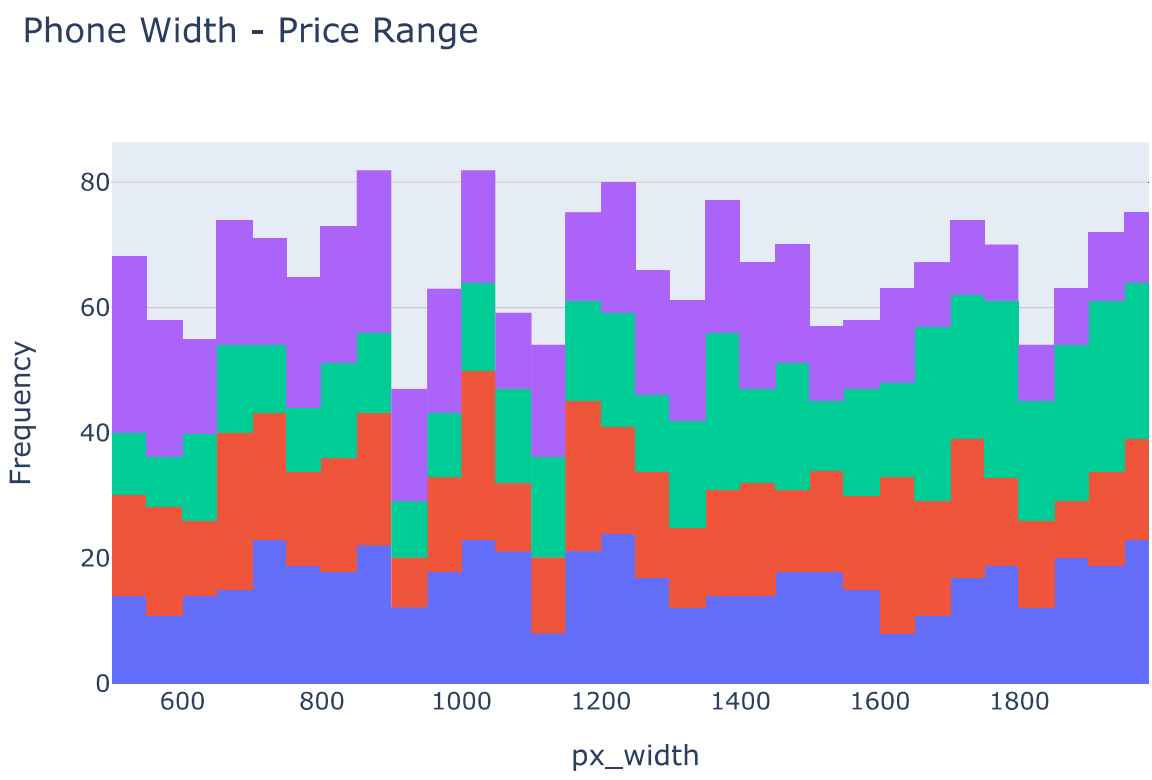
```
fig = px.histogram(df, x = 'px_height',
                   color = 'price_range',
                   title = "Phone Height - Price Range")
fig.update_layout(
xaxis_title_text = 'px_height',
yaxis_title_text = 'Frequency',autosize=False,
    width=700,
    height=450)
fig
```



Phone Width - Price Range

In [25]:

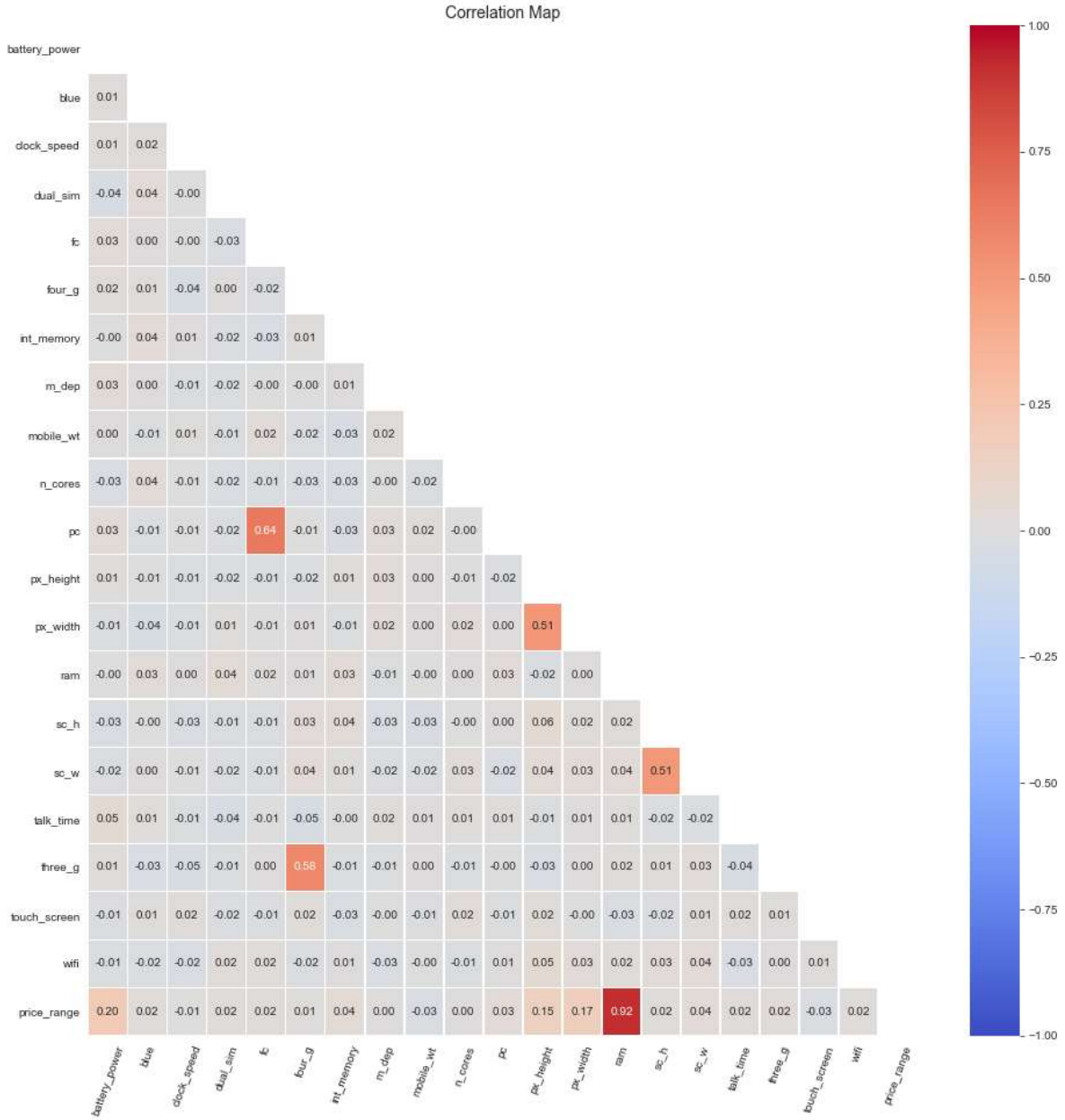
```
fig = px.histogram(df, x = 'px_width',
                   color = 'price_range',
                   title = "Phone Width - Price Range")
fig.update_layout(
xaxis_title_text = 'px_width',
yaxis_title_text = 'Frequency',autosize=False,
    width=700,
    height=450)
fig
```



Correlation Map

In [26]:

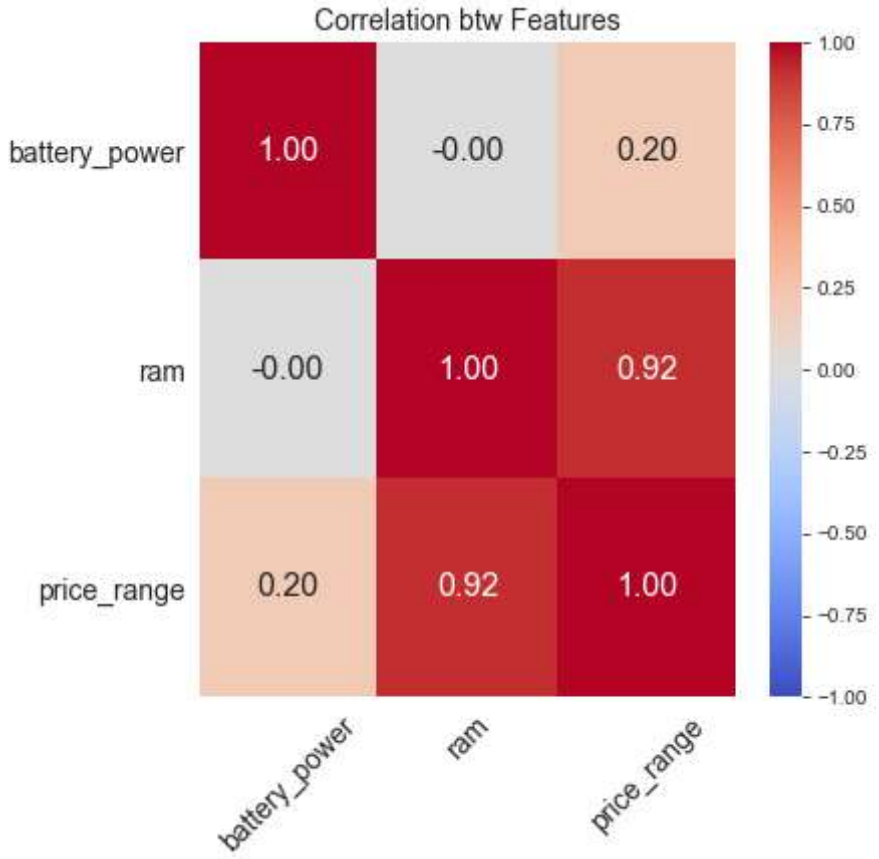
```
matrix = np.triu(df.corr())
sns.set_style("white")
f,ax=plt.subplots(figsize = (16,16))
sns.heatmap(df.corr(),annot= True,fmt = ".2f",ax=ax,
            vmin = -1,
            vmax = 1, mask = matrix,cmap = "coolwarm",
            linewidth = 0.2,linecolor = "white")
plt.xticks(rotation=70)
plt.yticks(rotation=0)
plt.title('Correlation Map', size = 14)
plt.show()
```



Thresholded Correlation Map

In [27]:

```
corr_matrix = df.corr()

threshold = 0.20
filter = np.abs(corr_matrix["price_range"])>threshold
corr_features = corr_matrix.columns[filter].tolist()

f,ax=plt.subplots(figsize = (6,6))
sns.heatmap(df[corr_features].corr(),annot= True,fmt = ".2f",
            vmin = -1,vmax = 1,ax=ax,annot_kws={"size": 16},cmap = "coolwarm")
plt.xticks(rotation=45, size = 14)
plt.yticks(rotation=0, size = 14)
plt.title('Correlation btw Features', size = 14)
plt.show()
```



Preprocessing

Train Test Split

In [28]:

```
random_state = 42

X = df.iloc[:,0:20].values
y = df.iloc[:, 20].values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state
```

Standardization

In [29]:

```
sc = StandardScaler()

X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

Models

Logistic Regression

In [30]:

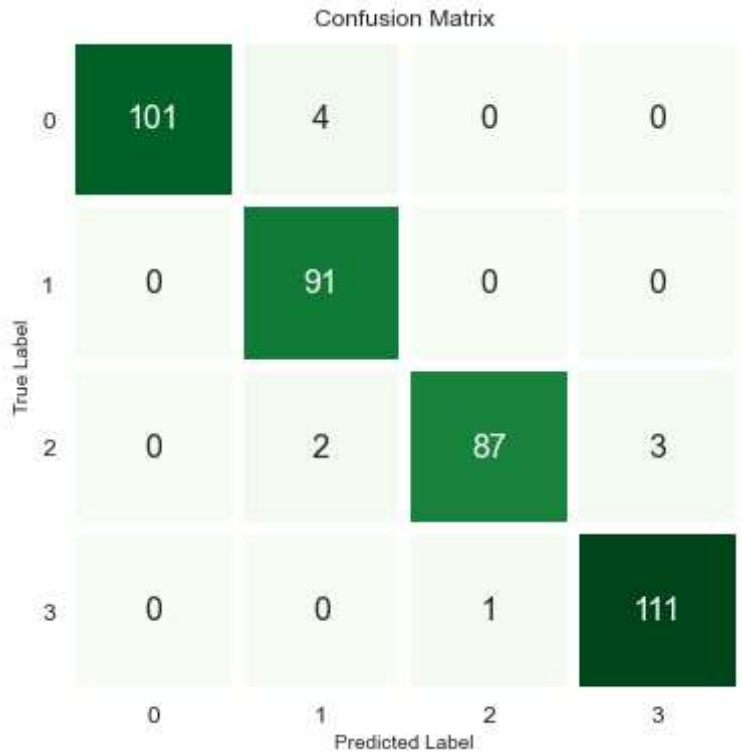
```
classifier = LogisticRegression(random_state = random_state)

classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)

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

```
              precision    recall  f1-score   support

           0       1.00      0.96      0.98       105
           1       0.94      1.00      0.97        91
           2       0.99      0.95      0.97        92
           3       0.97      0.99      0.98       112

    accuracy                           0.97       400
   macro avg       0.98      0.97      0.97       400
weighted avg       0.98      0.97      0.98       400
```

Confusion Matrix

In [31]:

```
cm = confusion_matrix(y_test, y_pred)

df1 = pd.DataFrame(columns=["0","1","2","3"], index= ["0","1","2","3"], data= cm )

f,ax = plt.subplots(figsize=(6,6))

sns.heatmap(df1, annot=True,cmap="Greens", fmt= '.0f',ax=ax,linewidths = 5, cbar = Fals
plt.xlabel("Predicted Label")
plt.xticks(size = 12)
plt.yticks(size = 12, rotation = 0)
plt.ylabel("True Label")
plt.title("Confusion Matrix", size = 12)
plt.show()
```



Linear Discriminant Analysis

In [32]:

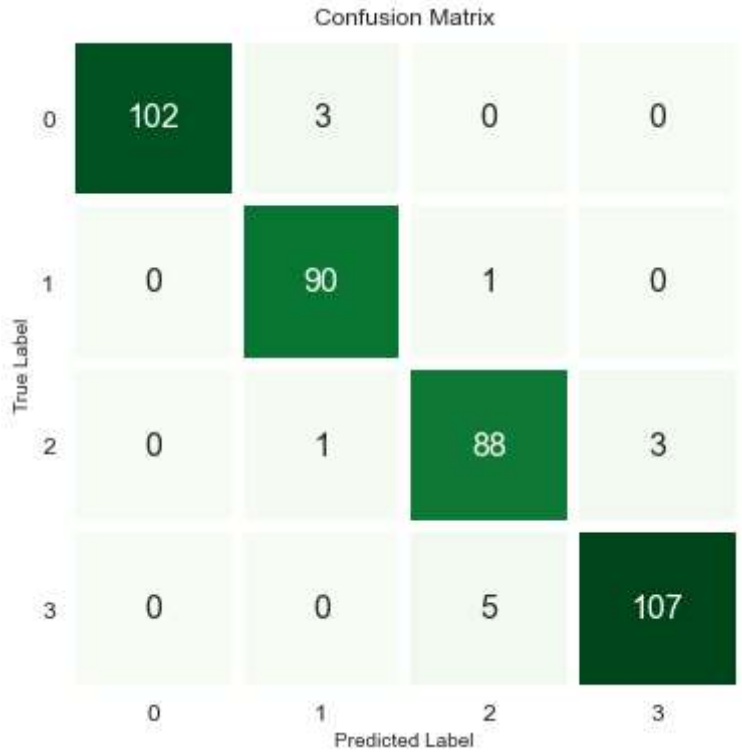
```
lda = LDA(n_components = 2)

X_train_lda = lda.fit_transform(X_train,y_train)
X_test_lda = lda.transform(X_test)

classifier_lda = LogisticRegression(random_state = random_state)
classifier_lda.fit(X_train_lda, y_train)

y_pred_lda = classifier_lda.predict(X_test_lda)

print(classification_report(y_test, y_pred_lda))
```

```
              precision    recall  f1-score   support

           0       1.00      0.97      0.99       105
           1       0.96      0.99      0.97        91
           2       0.94      0.96      0.95        92
           3       0.97      0.96      0.96       112

    accuracy                           0.97       400
   macro avg       0.97      0.97      0.97       400
weighted avg       0.97      0.97      0.97       400
```

Confusion Matrix

In [33]:

```
cm = confusion_matrix(y_test, y_pred_lda)

df1 = pd.DataFrame(columns=["0","1","2","3"], index= ["0","1","2","3"], data= cm )

f,ax = plt.subplots(figsize=(6,6))

sns.heatmap(df1, annot=True,cmap="Greens", fmt= '.0f',ax=ax,linewidths = 5, cbar = Fals
plt.xlabel("Predicted Label")
plt.xticks(size = 12)
plt.yticks(size = 12, rotation = 0)
plt.ylabel("True Label")
plt.title("Confusion Matrix", size = 12)
plt.show()
```



Hyperparameter Tuning - Grid Search - Cross Validation

```
# We will compare 8 classifier and evaluate mean accuracy of each of them by
stratified cross validation.

Decision Tree Classifier
SVC
Random Forest Classifier
Logistic Regression
KNN Classifier
Stochastic Gradient Descent Classifier
Gradient Boosting Classifier
LightGBM Classifier
```

In [34]:

```
classifier = [DecisionTreeClassifier(random_state = random_state),
              SVC(random_state = random_state, probability = True),
              RandomForestClassifier(random_state = random_state),
              LogisticRegression(random_state = random_state),
              KNeighborsClassifier(),
              SGDClassifier(random_state = random_state),
              GradientBoostingClassifier(random_state = random_state),
              LGBMClassifier(random_state = random_state)]

dt_param_grid = {"min_samples_split" : range(10,500,20),
                 "max_depth": range(1,20,2)}

svc_param_grid = {"kernel" : ["rbf"],
                  "gamma": [0.001, 0.01, 0.1, 1],
                  "C": [1,10,50,100,200,300,1000]}

rf_param_grid = {"max_features": [1,3,10],
                 "min_samples_split":[2,3,10],
                 "min_samples_leaf":[1,3,10],
                 "bootstrap":[False],
                 "n_estimators":[100,300],
                 "criterion":["gini"]}

logreg_param_grid = {"C":np.logspace(-4, 4, 20),
                     "penalty": ["l1","l2","none"]}

knn_param_grid = {"n_neighbors": np.linspace(2,20,12, dtype = int).tolist(),
                  "weights": ["uniform","distance"],
                  "metric":["euclidean","manhattan","minkowski"],
                  "leaf_size": [30]}

sgdc_param_grid = {
    "loss" : ["hinge", "log", "squared_hinge", "modified_huber"],
    "alpha" : [0.0001, 0.001, 0.01, 0.1],
    "penalty" : ["l2", "l1", "none"]}

gbc_param_grid = {
    "learning_rate": [0.05, 0.1, 0.2],
    "min_samples_split": [2,3,10],
    "min_samples_leaf": [1,3,10]
    }


lgbmc_param_grid = {
    'num_leaves': [31, 127],
    'reg_alpha': [0.1, 0.5]}


classifier_param = [dt_param_grid,
                    svc_param_grid,
                    rf_param_grid,
                    logreg_param_grid,
                    knn_param_grid,
                    sgdc_param_grid,
                    gbc_param_grid,
                    lgbmc_param_grid]

cv_result = []
best_estimators = []
```

```
mean_squared_errors = []
roc_auc_scores = []
recall_scores = []
precision_scores = []
f1_scores = []


for i in range(len(classifier)):
    print("------------------------------------------------------------------------------
    clf = GridSearchCV(classifier[i],
                       param_grid=classifier_param[i],
                       cv = StratifiedKFold(n_splits = 10),
                       scoring = "accuracy",
                       n_jobs = -1,verbose = 2)

    clf.fit(X_train,y_train)

    cv_result.append(clf.best_score_)

    mean_squared_errors.append(mean_squared_error(y_test,clf.predict(X_test)))

    roc_auc_scores.append(roc_auc_score(y_test, clf.predict_proba(X_test), multi_class

    recall_scores.append(recall_score(y_test, clf.predict(X_test), average='weighted')

    precision_scores.append(precision_score(y_test, clf.predict(X_test), average='weig

    f1_scores.append(f1_score(y_test, clf.predict(X_test), average='weighted'))

    best_estimators.append(clf.best_estimator_)

    print("Model: {}".format(classifier[i]))
    print("Accuracy: %{}".format(round(cv_result[i]*100,2)))
    print("MSE: {}".format(mean_squared_errors[i]))
    print("ROC AUC: {}".format(roc_auc_scores[i]))
    print("Recall: {}".format(recall_scores[i]))
    print("Precision: {}".format(precision_scores[i]))
    print("F1-Score: {}".format(f1_scores[i]))
    print("Best Estimator: {}".format(clf.best_estimator_))

print("------------------------------------------------------------------------------")

sns.set_style("darkgrid")
cv_results = pd.DataFrame({"Accuracy":cv_result,
                           "MSE":mean_squared_errors,
                           "ROC AUC":roc_auc_scores,
                           "Recall": recall_scores,
                           "Precision": precision_scores,
                           "F1-Score":f1_scores,
                           "Models":["DecisionTreeClassifier",
                                     "SVC",
                                     "RandomForestClassifier",
                                     "LogisticRegression",
                                     "KNeighborsClassifier",
                                     "SGDClassifier",
                                     "GBClassifier",
                                     "LGBMClassifier"]})

cv_results.index = cv_results["Models"]

cv_results  = cv_results.drop(["Models"], axis = 1)
```

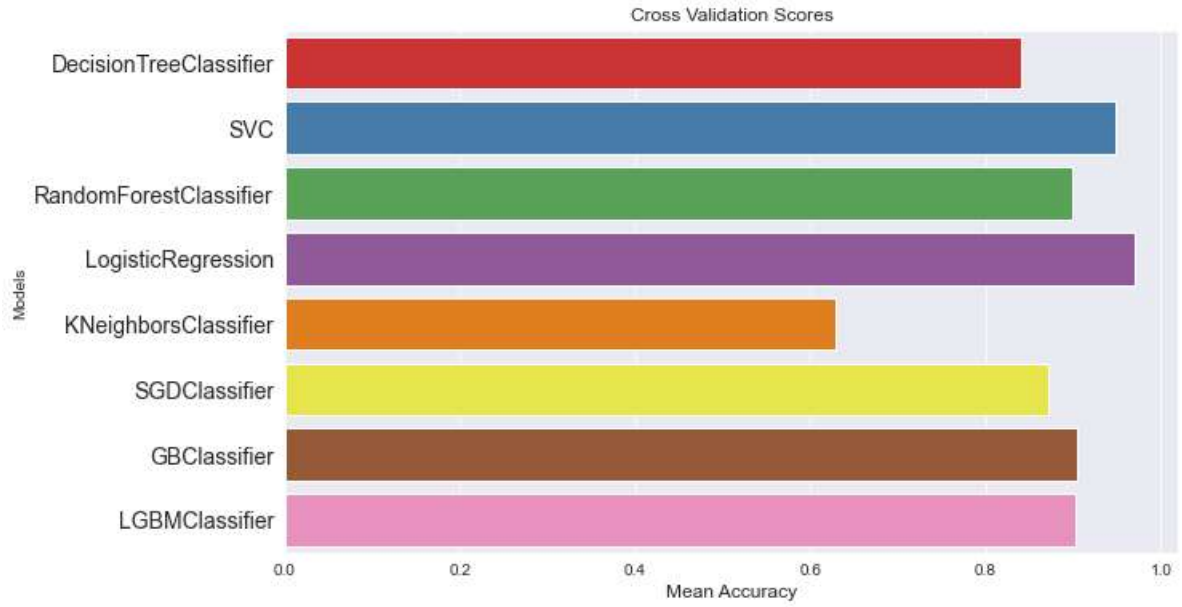
```
121 
122 f,ax = plt.subplots(figsize=(14,10))
123 
124 sns.heatmap(cv_results, annot=True,cmap = "Blues",fmt= '.3f',
125             ax=ax,linewidths = 5, cbar = False,
126             annot_kws={"size": 18})
127 
128 plt.xticks(size = 18)
129 plt.yticks(size = 18, rotation = 0)
130 plt.ylabel("Models")
131 plt.title("Grid Search Results", size = 16)
132 plt.show()
```

```
--------------------------------------------------------------------------
-
Fitting 10 folds for each of 250 candidates, totalling 2500 fits


[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent worker
s.
[Parallel(n_jobs=-1)]: Done  72 tasks      | elapsed:    0.2s
[Parallel(n_jobs=-1)]: Done 2408 tasks      | elapsed:    5.4s
[Parallel(n_jobs=-1)]: Done 2500 out of 2500 | elapsed:    5.4s finished
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent worker
s.

Model: DecisionTreeClassifier(random_state=42)
Accuracy: %84.06
MSE: 0.175
ROC AUC: 0.911858935204842
Recall: 0.825
Precision: 0.8276565656565658
F1-Score: 0.8257884190113292
```

Cross Validation Scores

In [35]:

```
sns.set_style("darkgrid")
cv_results = pd.DataFrame({"Cross Validation Means":cv_result,
                           "Models":["DecisionTreeClassifier", "SVC",
                                     "RandomForestClassifier",
                                     "LogisticRegression",
                                     "KNeighborsClassifier",
                                     "SGDClassifier",
                                     "GBClassifier",
                                     "LGBMClassifier"]})

plt.figure(figsize = (10,6))
sns.barplot("Cross Validation Means", "Models",
            data = cv_results, palette = "Set1")
plt.xlabel("Mean Accuracy",
           size = 12)
plt.yticks(size = 14)
plt.title("Cross Validation Scores",
          size = 12)
plt.show()
```



Ensemble Learning

In [36]:

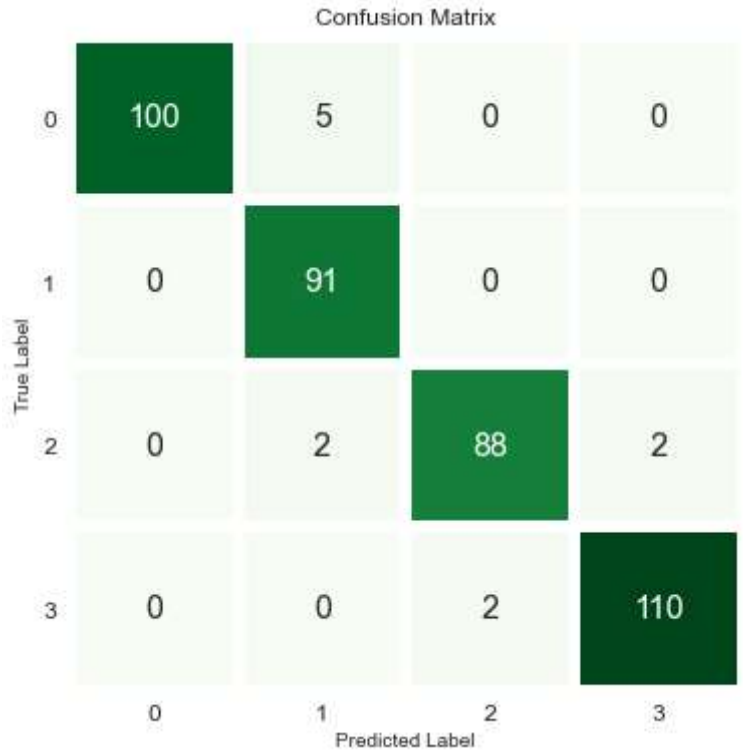
```
votingC = VotingClassifier(estimators = [("svc",best_estimators[1]),
                                         ("lr",best_estimators[3]),
                                         ("gbc",best_estimators[6])])

votingC = votingC.fit(X_train, y_train)

voting_pred = votingC.predict(X_test)

print(classification_report(y_test, voting_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      0.95      0.98       105
           1       0.93      1.00      0.96        91
           2       0.98      0.96      0.97        92
           3       0.98      0.98      0.98       112

    accuracy                           0.97       400
   macro avg       0.97      0.97      0.97       400
weighted avg       0.97      0.97      0.97       400
```

Confusion Matrix

In [37]:

```
cm = confusion_matrix(y_test, voting_pred)

df1 = pd.DataFrame(columns=["0","1","2","3"], index= ["0","1","2","3"], data= cm )

f,ax = plt.subplots(figsize=(6,6))

sns.heatmap(df1, annot=True,cmap="Greens", fmt= '.0f',ax=ax,linewidths = 5, cbar = Fals
plt.xlabel("Predicted Label")
plt.xticks(size = 12)
plt.yticks(size = 12, rotation = 0)
plt.ylabel("True Label")
plt.title("Confusion Matrix", size = 12)
plt.show()
```



Best Estimator

In [38]:

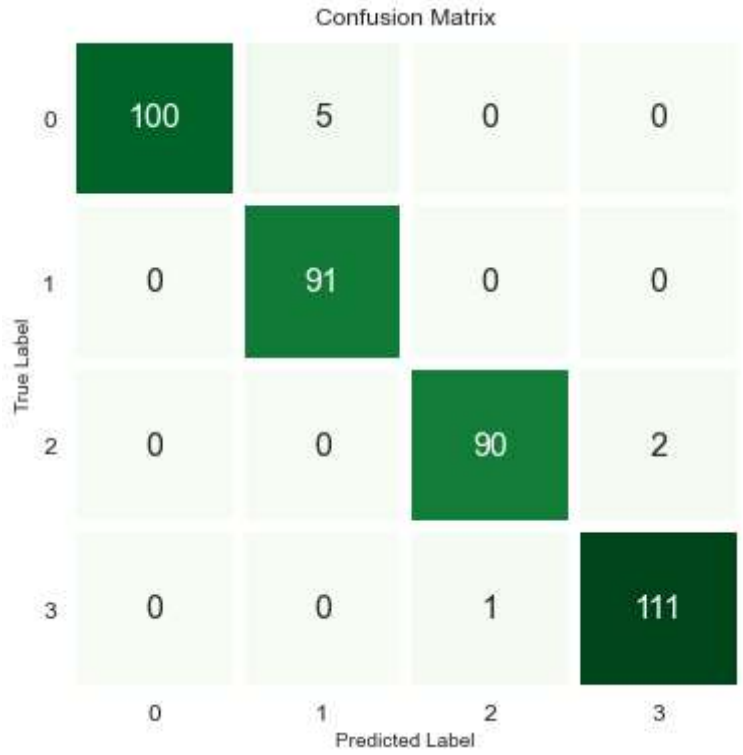
```
classifier = LogisticRegression(C=545.5594781168514, random_state=42)

classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)

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

```
              precision    recall  f1-score   support

           0       1.00      0.95      0.98       105
           1       0.95      1.00      0.97        91
           2       0.99      0.98      0.98        92
           3       0.98      0.99      0.99       112

    accuracy                           0.98       400
   macro avg       0.98      0.98      0.98       400
weighted avg       0.98      0.98      0.98       400
```

Confusion Matrix

In [39]:

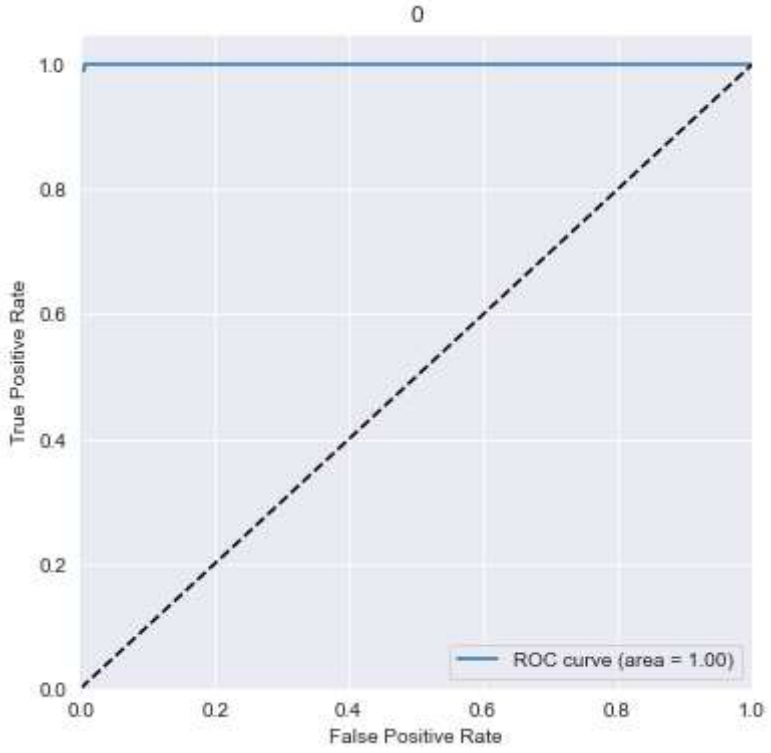
```
cm = confusion_matrix(y_test, y_pred)

df1 = pd.DataFrame(columns=["0","1","2","3"], index= ["0","1","2","3"], data= cm )

f,ax = plt.subplots(figsize=(6,6))

sns.heatmap(df1, annot=True,cmap="Greens", fmt= '.0f',ax=ax,linewidths = 5, cbar = Fals
plt.xlabel("Predicted Label")
plt.xticks(size = 12)
plt.yticks(size = 12, rotation = 0)
plt.ylabel("True Label")
plt.title("Confusion Matrix", size = 12)
plt.show()
```

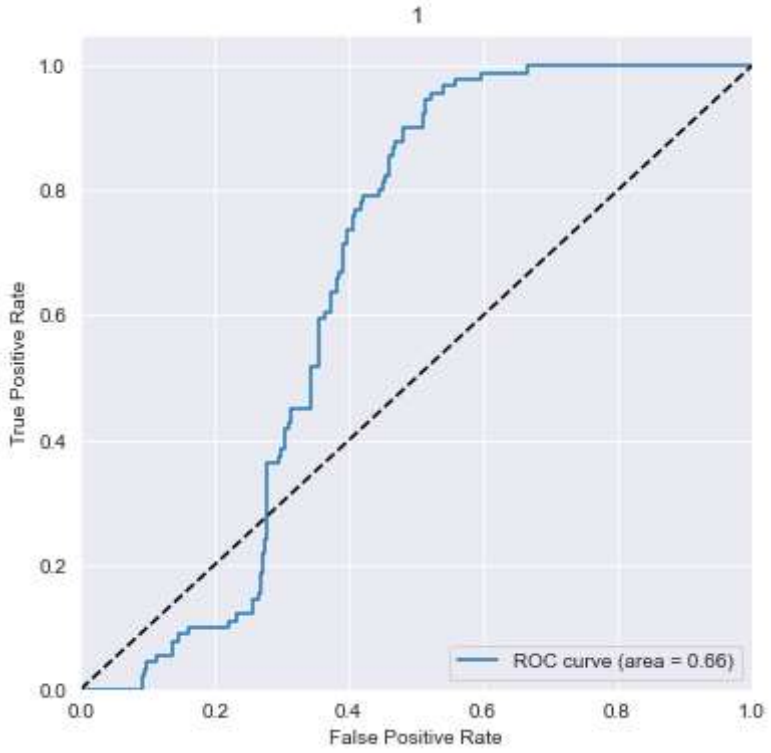


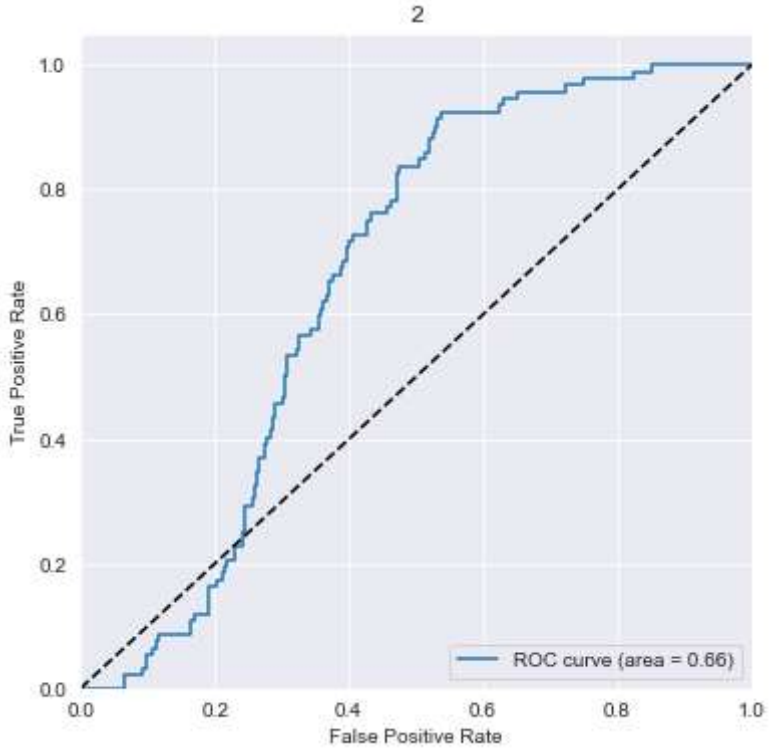
Positive Rates

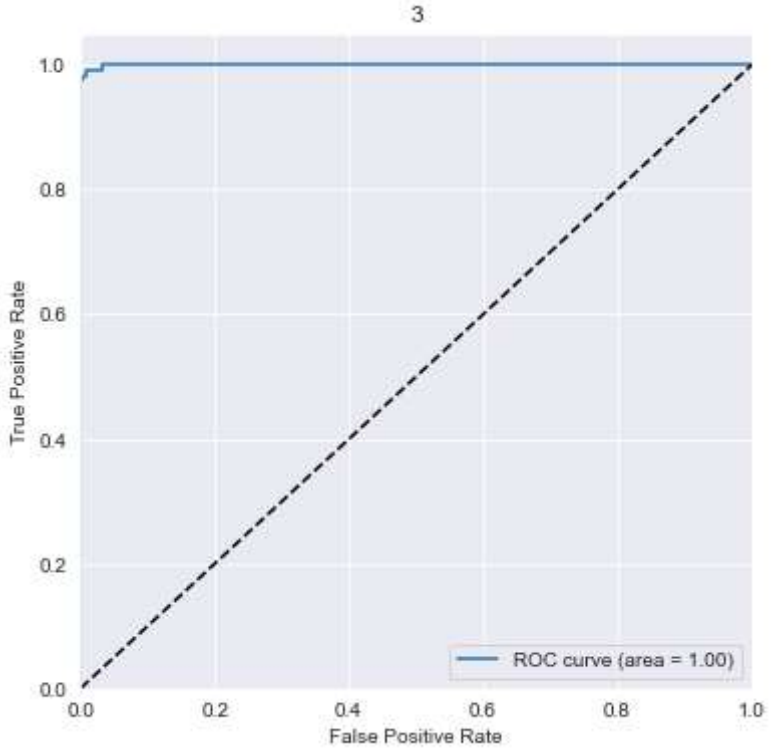
In [40]:

```
X = df.iloc[:,0:20].values
y = df.iloc[:, 20].values

y = label_binarize(y, classes=[0,1,2,3])
n_classes = 4

X_train, X_test, y_train, y_test =\
    train_test_split(X, y, test_size=0.2, random_state=42)

sc = StandardScaler()

X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

clf = OneVsRestClassifier(best_estimators[3])
y_score = clf.fit(X_train, y_train).decision_function(X_test)

fpr = dict()
tpr = dict()
roc_auc = dict()

for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test[:, i],
                                  y_score[:, i])
    roc_auc[i] = auc(fpr[i],
                     tpr[i])

for i in range(n_classes):
    plt.figure(figsize = (6,6))
    plt.plot(fpr[i], tpr[i], label='ROC curve (area = %0.2f)' % roc_auc[i])
    plt.plot([0, 1], [0, 1], 'k--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title(i)
    plt.legend(loc="lower right")
    plt.show()
```


0
True Positive Rate
False Positive Rate
ROC curve (area = 1.00)
0.0
0.2
0.4
0.6
0.8
1.0


1
True Positive Rate
False Positive Rate
ROC curve (area = 0.66)
0.0
0.2
0.4
0.6
0.8
1.0


2
True Positive Rate
False Positive Rate
ROC curve (area = 0.66)
0.0
0.2
0.4
0.6
0.8
1.0



Test.csv

In [41]:

```
1 dft= pd.read_csv('test.csv')
2 dft.head()
```

Out[41]:

	id	battery_power	blue	clock_speed	dual_sim	fc	four_g	int_memory	m_dep	mobile_wt
0	1	1043	1	1.8	1	14	0	5	0.1	193
1	2	841	1	0.5	1	4	1	61	0.8	191
2	3	1807	1	2.8	0	1	0	27	0.9	186
3	4	1546	0	0.5	1	18	1	25	0.5	96
4	5	1434	0	1.4	0	11	1	49	0.5	108

5 rows × 21 columns

In [42]:

```
1 dft.sample(5)
```

Out[42]:

	id	battery_power	blue	clock_speed	dual_sim	fc	four_g	int_memory	m_dep	mobile
414	415	658	1	2.4	0	0	1	38	1.0	
973	974	1796	1	0.5	0	0	0	44	0.2	
860	861	1951	0	2.1	0	5	0	22	0.5	
855	856	1270	0	0.5	0	1	1	62	0.5	
938	939	940	1	2.0	1	7	1	24	0.6	

5 rows × 21 columns

In [43]:

```
1 print(df.shape, '\n', dft.shape)
```

```
(2000, 21)
 (1000, 21)
```

focusing on train data

In [44]:

```
1 colt=dft.columns
2
3 colt
```

Out[44]:

```
Index(['id', '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'],
      dtype='object')
```

data preparation

In [45]:

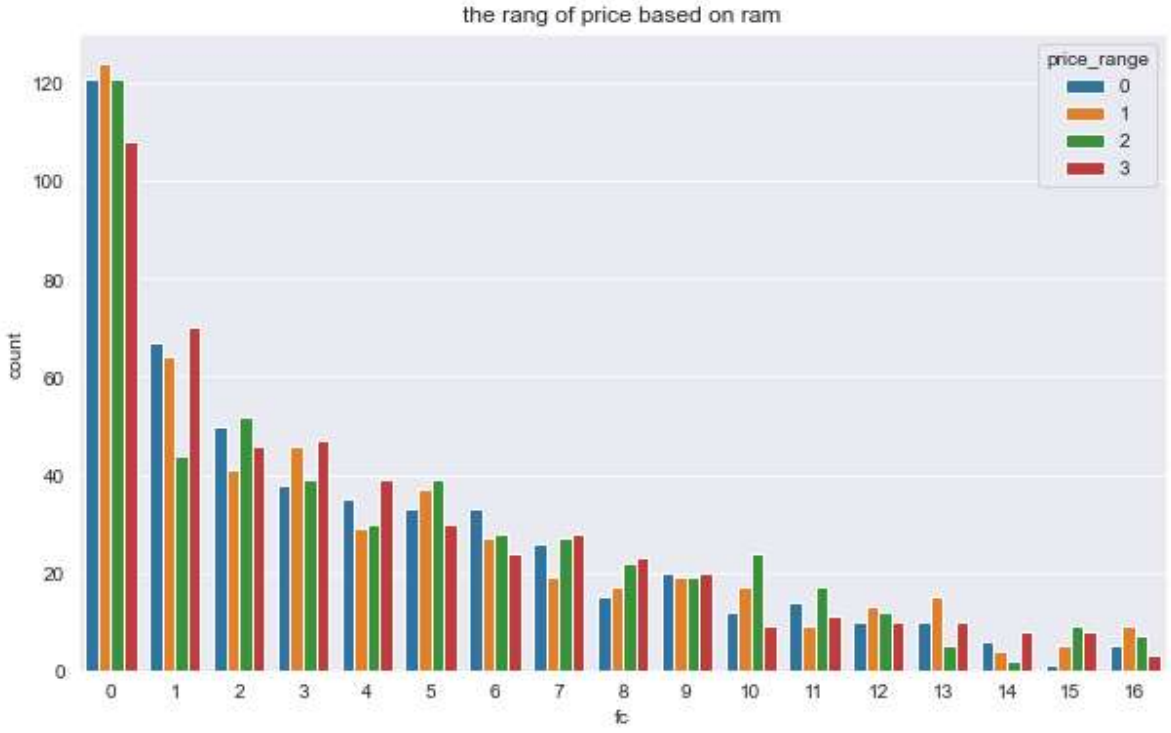
```
1 X=df.drop('price_range' , axis=1)
2 y=df.price_range
3
4 X_train , X_test , y_train ,y_test=train_test_split(X,y , test_size=0.2 , random_state=
```

In [46]:

```
# #   remove the outlier


df=df[(df['fc']<17)]
```

In [47]:

```
plt.figure(figsize=(10,6))
sns.countplot(x='fc', data=df, hue= 'price_range').set_title('the rang of price based c
```



Baseline Model

In [48]:

```
def baseline_model(n_preds, pred):
    return pd.Series([pred for n in range(n_preds)])

# make baseline preds
baseline_preds = baseline_model(len(y_test), np.mean(y_train))
```

In [49]:

```
mse_bl=mean_squared_error(y_true=y_test,
                  y_pred=baseline_preds,
                  squared=False)
mse_bl
```

Out[49]:

```
1.1228495071134867
```

Linear Regression model

In [50]:

```
from sklearn.linear_model import LinearRegression
lr=LinearRegression()
lr.fit(X_train, y_train)
```

Out[50]:

```
LinearRegression()
```

In [51]:

```
1 preds=lr.predict(X_test )
2 preds
```

Out[51]:

```
array([ 0.58818248,  1.82415214,  1.75035299,  2.88288036,  1.62318216,
        2.60254899,  1.41618596,  1.91877932, -0.09573825, -0.21421173,
        1.81123556,  1.83286404,  2.48446502, -0.18665127,  0.98348599,
        2.47199176,  1.4741242 ,  1.84136057,  2.16961979,  2.85331983,
        2.24841953,  2.6472683 ,  0.26153926, -0.12090612,  0.87318424,
        0.47074503,  0.27548006, -0.00505636,  0.19532002,  1.13155179,
        3.06910724,  1.84229677, -0.14651814,  1.85193464,  0.73946167,
        1.93708206,  0.33387756,  1.17224826,  0.4487441 ,  0.25745995,
        0.78613947,  1.35515181,  1.10027979,  1.77408844,  3.07384329,
        1.4629077 ,  2.3186361 ,  1.89913094,  0.78290014,  0.23431017,
        1.97132828,  1.99319675,  1.73581261,  1.5889805 ,  1.33147707,
        0.84196367,  2.99509459,  2.83930378,  2.67138641,  1.43009796,
        0.12863823,  2.96472594,  2.18449173, -0.18334847,  1.52345437,
        2.9949579 ,  0.98310054,  1.2421977 , -0.3185788 ,  1.37166214,
        0.56997886, -0.33709424,  2.18646234, -0.05290714,  1.81908137,
        2.51839965,  0.39875125,  2.9025417 ,  1.25808118,  2.3427563 ,
        2.91819707,  2.56265715,  0.13448348,  2.5194856 ,  1.64798659,
       -0.35826994,  2.90986848,  1.94181551,  0.34403366,  1.91137799,
        2.78155602,  0.73371552,  2.93807978, -0.09755419,  1.47809471,
        0.52127051,  2.98956802,  2.69487403, -0.26774681,  2.23994687,
        2.33145562,  2.62838395,  0.14073856,  0.89072521,  0.1512592 ,
        1.64810354,  1.18878215,  0.7551649 ,  0.68487469,  2.31815903,
        1.95626133, -0.19971651,  1.74665749,  3.01251452,  3.09448484,
        0.25039154,  0.07477078,  0.94316343,  3.50712695,  2.92099727,
        1.58811775,  2.75805223,  0.53372121,  1.46487494,  1.76524721,
        2.21890747,  0.64039425,  1.43302267, -0.01788198,  1.21339256,
        2.24051475,  2.72020639,  1.67191512,  0.46234226,  3.34539516,
        1.36802116,  2.8147585 , -0.32147324,  1.48088207,  2.38342767,
        1.28557317,  0.55886439,  1.49422594, -0.07422244,  0.53682561,
        2.32572614,  0.52479859,  1.15454862,  0.91262485,  1.16510497,
        1.79357264,  2.9556915 ,  0.81950929,  0.32870385,  1.98262264,
        3.44938256,  0.72132671,  2.50640157,  1.97473271,  1.31868647,
        0.08224244, -0.339832  ,  0.30094391,  1.92643682,  3.60791655,
        1.06284036, -0.32889956,  0.4414403 ,  3.43841579,  3.33863805,
        1.22611111,  1.93479192,  1.87078187,  3.12169482,  2.10043228,
        3.3405779 ,  2.17453007,  0.05783323,  1.87140819,  3.24637125,
        2.54717083,  0.8038563 ,  3.38203242,  3.54213258,  3.0525713 ,
        2.405895  ,  0.28829743,  2.37283278,  0.46733958, -0.12261864,
        0.43979961,  0.50832441,  2.25572135,  3.10757402,  1.52310824,
        1.78434849,  2.75029982,  2.90534732,  0.97262091,  0.9204592 ,
        1.03258085,  2.44871877,  0.56840611,  1.75041366,  1.16666125,
        1.29287338,  1.16226044,  1.30358869, -0.31518604,  0.38174651,
        0.8939994 ,  1.07394755,  1.94631206, -0.1509068 ,  2.87110185,
        1.56816545,  2.92902008,  3.57099597,  0.4969231 ,  0.98208201,
       -0.1778838 ,  0.66810743,  0.03455674,  2.71741168,  2.57336136,
        2.8495846 ,  1.49648716, -0.14756242,  1.84101551,  0.76745736,
       -0.30937569,  2.35103027,  1.22371913,  0.65138889,  2.59028243,
        0.73727566,  1.10688246,  1.62104359,  3.13226199,  1.39755334,
       -0.38297723,  1.7756361 ,  2.81081855,  2.24418829,  2.47065202,
       -0.24632513,  0.07087221,  2.80268597,  1.57172541,  1.38394471,
        1.49407766,  1.23701192,  1.43240494,  1.26849896,  1.34513722,
        0.94225043, -0.51219358,  2.40921073,  1.5288871 ,  2.4014176 ,
        2.68540243, -0.11284311,  0.49415146,  1.10280022,  0.20700862,
        4.02298131, -0.04565787,  1.89076013,  1.45309397,  2.81987104,
```

```
       1.16619639,  2.09406264,  1.5057153 ,  0.49555463,  0.54870068,
       2.66097153, -0.30227106,  1.40628554,  1.40990504,  2.2131194 ,
       3.04143867,  0.04578148, -0.031494  ,  2.80533311,  2.95827539,
       1.44161967,  2.46463159,  3.27185865,  3.32370593,  0.94160705,
       1.01807456,  0.24825746,  1.01029645,  2.7439815 ,  2.13508742,
       0.32910353,  2.14461664,  0.90587812,  2.10897809,  1.00752219,
       1.94103924, -0.29321118,  3.57548806,  2.45724372,  3.34856238,
       2.63010161,  1.42034565,  2.20672192,  0.97643002,  2.39441476,
       2.31764776,  1.77553206,  0.18570609,  1.97981121,  0.52475262,
       2.29831507,  0.23209878,  2.88046328,  3.0090253 ,  0.15613683,
       1.37708145,  0.44194436,  1.08502215,  2.27837762,  0.14716532,
       3.02336626, -0.17171903,  2.67979645,  1.98385077,  2.65358167,
       2.12505321,  1.23067929,  1.07472903,  2.39800348,  2.5414606 ,
       1.38826146,  1.4468988 ,  0.2255011 ,  1.75631085,  3.05018447,
       1.87296232,  1.21133164,  2.62680637,  0.5064031 ,  2.9247972 ,
       1.08046652,  0.17138448,  2.98783861,  0.82935613, -0.3416708 ,
       2.2118631 ,  0.02784413,  2.20953634,  1.65944213,  2.73612969,
       2.96460154,  0.26579658,  1.88108869,  2.35256472,  1.88020128,
       1.84129803,  0.98335158,  1.96841515,  0.85844671,  2.53131569,
       0.86848572,  2.3671069 ,  0.44740326,  1.74792096,  2.91096826,
       3.00508911,  0.96209661,  0.34952577,  0.43933193,  0.36294028,
      -0.29542185,  2.40492535,  2.85519678,  1.12855852,  2.7812262 ,
       2.67861637,  2.94504536,  0.45951518,  1.91074393,  0.93909732,
       2.34130043,  0.8571081 ,  2.00704973,  3.15718549,  2.23392014,
       3.38509353,  0.56705451, -0.65462441,  1.71693104,  2.19319046,
       2.45383792,  0.53179431,  1.5784931 ,  1.87377518,  0.24579903])
```

In [52]:

```
1 # retour au test.csv
2 dft.head()
```

Out[52]:

	id	battery_power	blue	clock_speed	dual_sim	fc	four_g	int_memory	m_dep	mobile_wt
0	1	1043	1	1.8	1	14	0	5	0.1	193
1	2	841	1	0.5	1	4	1	61	0.8	191
2	3	1807	1	2.8	0	1	0	27	0.9	186
3	4	1546	0	0.5	1	18	1	25	0.5	96
4	5	1434	0	1.4	0	11	1	49	0.5	108

5 rows × 21 columns

In [53]:

```
# sans compter la 1ère colonne
predt=lr.predict(dft.drop('id', axis=1))
predt
```

Out[53]:

```
array([ 2.50883263e+00,  2.87989950e+00,  2.14844694e+00,  3.43977805e+00,
        1.24696405e+00,  2.74099010e+00,  3.57846240e+00,  1.01709872e+00,
        2.61425050e+00,  1.31181916e-03,  3.34678317e+00,  3.21180892e+00,
       -1.18076932e-02,  4.31602512e-01,  1.99770214e+00,  4.36139198e-01,
        1.88343584e+00,  1.16755315e+00,  3.11185049e+00,  1.59825340e+00,
        7.97397033e-01,  2.48614093e+00,  1.32109854e+00,  1.32081164e+00,
        3.19431650e+00,  2.71927083e-02,  2.04695701e+00, -3.44412163e-01,
        2.62805038e+00,  4.06415649e-02,  2.16280246e+00,  2.13950004e-01,
        2.84582156e+00,  1.93636932e-01,  5.48663486e-01,  9.72005643e-01,
        2.77195881e+00,  1.21900904e+00,  2.16387545e+00,  9.53917132e-01,
        1.09759815e+00,  1.73748505e+00, -3.99622402e-01, -3.56458507e-01,
        4.05275273e-04,  8.74976993e-01,  3.75447859e-01,  2.60299414e+00,
        1.05382235e+00,  2.15539201e+00,  1.38975774e+00, -5.71754498e-03,
        2.33163862e+00,  4.07984847e-01,  3.12809132e+00,  6.67434961e-01,
        3.05549475e+00,  1.08203021e+00,  7.10295027e-01,  3.14308419e+00,
        3.18164183e+00,  2.52382677e+00,  2.44608828e-01,  1.44291148e+00,
        6.37488050e-01,  1.10412698e+00,  1.41791822e+00,  2.44023139e+00,
        9.45163761e-01,  1.69363777e+00,  1.06808811e+00,  1.96410683e+00,
```

In [54]:

```
#accuracy_score(y_true=y_test, y_pred=preds)
#Classification metrics can't handle a mix of multiclass and continuous targets
```

In [55]:

```
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import r2_score
lr_mae = mean_absolute_error(y_test , preds)
lr_mse = mean_squared_error(y_test, preds)
lr_rse = mean_squared_error(y_test, preds , squared=False)
lr_r2 = r2_score(y_test , preds)
```

In [56]:

```
print('Linear Regression model Mean Absolute Error:', lr_mae)
print('Linear Regression model Mean squared Error:', lr_mse)
print('Linear Regression model  Squared Error:', lr_rse)
print('Linear Regression model R2 Score:',lr_r2)
```

```
Linear Regression model Mean Absolute Error: 0.26278042712698946
Linear Regression model Mean squared Error: 0.0996739255532883
Linear Regression model  Squared Error: 0.3157117760763578
Linear Regression model R2 Score: 0.9208053229620056
```

Logistic Regression

In [57]:

```
lreg = LogisticRegression()
lreg.fit(pd.DataFrame(X_train), y_train)
```

Out[57]:

```
LogisticRegression()
```

In [58]:

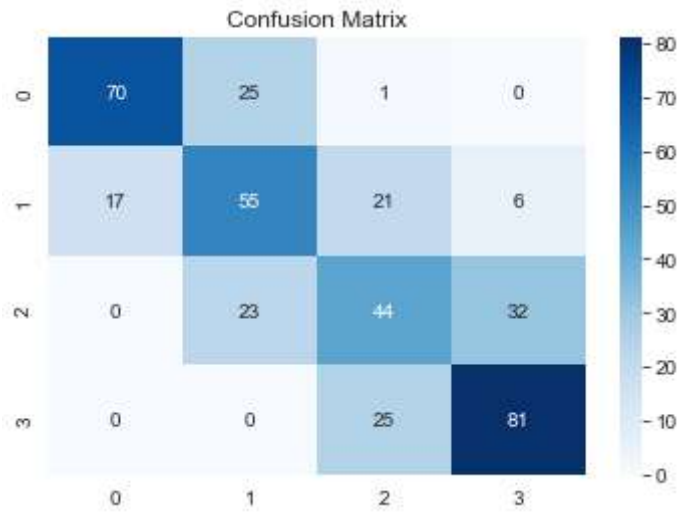
```
lreg_preds = lreg.predict(X_test)
```

In [59]:

```
from sklearn.metrics import accuracy_score , confusion_matrix
# what do they look like? (show me some of the predictions)
# create confusion matrix plot
cf_matrix = confusion_matrix(y_test, lreg_preds)
sns.heatmap(cf_matrix, annot=True, fmt='', cmap='Blues').set_title("Confusion Matrix")
```

Out[59]:

```
Text(0.5, 1.0, 'Confusion Matrix')
```



In [60]:

```
accuracy_score(y_true=y_test, y_pred=lreg_preds)
```

Out[60]:

```
0.625
```

In [61]:

```
lreg_mae = mean_absolute_error(y_test , lreg_preds)
lreg_mse = mean_squared_error(y_test, lreg_preds)
lreg_rse = mean_squared_error(y_test, lreg_preds , squared=False)
lreg_r2 = r2_score(y_test , lreg_preds)
```

In [75]:

```
print('Logistic Regression Mean Absolute Error:', lreg_mae)
print('Logistic Regression Mean squared Error:', lreg_mse)
print('Logistic Regression  Squared Error:', lreg_rse)
print('Logistic Regression R2 Score:',lreg_r2)
```

```
Logistic Regression Mean Absolute Error: 0.3925
Logistic Regression Mean squared Error: 0.4275
Logistic Regression  Squared Error: 0.653834841531101
Logistic Regression R2 Score: 0.6603351955307262
```

Creating & Training KNN Model

In [63]:

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=15)
knn.fit(X_train,y_train)
```

Out[63]:

```
KNeighborsClassifier(n_neighbors=15)
```

In [64]:

```
knn.score(X_test,y_test)
```

Out[64]:

```
0.935
```

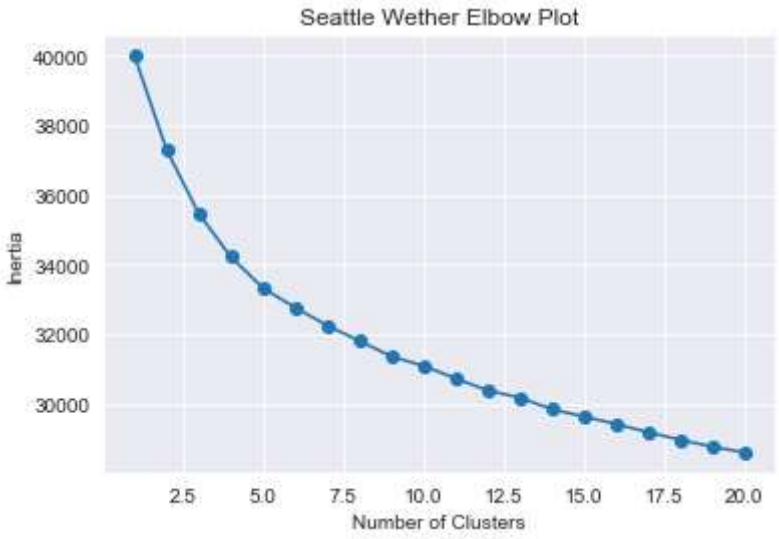
Elbow Method For optimum value of K

In [65]:

```
# Scale Data
ss2=StandardScaler()
sc2=ss2.fit_transform(X)
```

In [66]:

```
# Create Elbow Plot
from sklearn.cluster import KMeans
inertia_ls = [] # save inertia values
for k in range(1, 21): # test different n_clusters between 1-10
    km = KMeans(n_clusters=k)
    km.fit(sc2) # apply KMeans
    inertia_ls.append(km.inertia_) # Append inertia to list

# Plot Elbow
plt.plot([i for i in range(1, 21)], inertia_ls, marker="o")
plt.title("Seattle Wether Elbow Plot")
plt.xlabel("Number of Clusters")
plt.ylabel("Inertia");
```



In [67]:

```
# Apply KMeans and Plot KMeans Results and Actual Results
km=KMeans(n_clusters=4)
preds=km.fit_predict(sc2) # make labe and prediction in single step
preds
```

Out[67]:

```
array([0, 1, 1, ..., 3, 3, 2])
```

submition

In [68]:

```
# Submit 'price_range into 'test.csv''
dft['price_range']=predt
```

In [69]:

```
1 dft
```

Out[69]:

	id	battery_power	blue	clock_speed	dual_sim	fc	four_g	int_memory	m_dep	mobi
0	1	1043	1	1.8	1	14	0	5	0.1	
1	2	841	1	0.5	1	4	1	61	0.8	
2	3	1807	1	2.8	0	1	0	27	0.9	
3	4	1546	0	0.5	1	18	1	25	0.5	
4	5	1434	0	1.4	0	11	1	49	0.5	
...	...	...	...	...	...	...	...	...	...	
995	996	1700	1	1.9	0	0	1	54	0.5	
996	997	609	0	1.8	1	0	0	13	0.9	
997	998	1185	0	1.4	0	1	1	8	0.5	
998	999	1533	1	0.5	1	0	0	50	0.4	
999	1000	1270	1	0.5	0	4	1	35	0.1	

1000 rows × 22 columns

In [70]:

```
1 X = dft
2 y=dft['price_range']
3 lr=LinearRegression()
4 lr.fit(X, y)
5 preds=lr.predict(X)
```

In [71]:

```
1 preds
```

Out[71]:

```
array([ 2.50883263e+00,  2.87989950e+00,  2.14844694e+00,  3.43977805e+00,
        1.24696405e+00,  2.74099010e+00,  3.57846240e+00,  1.01709872e+00,
        2.61425050e+00,  1.31181916e-03,  3.34678317e+00,  3.21180892e+00,
       -1.18076932e-02,  4.31602512e-01,  1.99770214e+00,  4.36139198e-01,
        1.88343584e+00,  1.16755315e+00,  3.11185049e+00,  1.59825340e+00,
        7.97397033e-01,  2.48614093e+00,  1.32109854e+00,  1.32081164e+00,
        3.19431650e+00,  2.71927083e-02,  2.04695701e+00, -3.44412163e-01,
        2.62805038e+00,  4.06415649e-02,  2.16280246e+00,  2.13950004e-01,
        2.84582156e+00,  1.93636932e-01,  5.48663486e-01,  9.72005643e-01,
        2.77195881e+00,  1.21900904e+00,  2.16387545e+00,  9.53917132e-01,
        1.09759815e+00,  1.73748505e+00, -3.99622402e-01, -3.56458507e-01,
        4.05275273e-04,  8.74976993e-01,  3.75447859e-01,  2.60299414e+00,
        1.05382235e+00,  2.15539201e+00,  1.38975774e+00, -5.71754498e-03,
        2.33163862e+00,  4.07984847e-01,  3.12809132e+00,  6.67434961e-01,
        3.05549475e+00,  1.08203021e+00,  7.10295027e-01,  3.14308419e+00,
        3.18164183e+00,  2.52382677e+00,  2.44608828e-01,  1.44291148e+00,
        6.37488050e-01,  1.10412698e+00,  1.41791822e+00,  2.44023139e+00,
        9.45163761e-01,  1.69363777e+00,  1.06808811e+00,  1.96410683e+00,
```

In [72]:

```
1 len(preds)
```

Out[72]:

```
1000
```

In [73]:

```
1 # Compute Accuracy Score of KMean Labels with True Labels
2 dft["price_rangep"] =preds
```

In [74]:

```
1 dft
```

Out[74]:

	id	battery_power	blue	clock_speed	dual_sim	fc	four_g	int_memory	m_dep	mobi
0	1	1043	1	1.8	1	14	0	5	0.1	
1	2	841	1	0.5	1	4	1	61	0.8	
2	3	1807	1	2.8	0	1	0	27	0.9	
3	4	1546	0	0.5	1	18	1	25	0.5	
4	5	1434	0	1.4	0	11	1	49	0.5	
...	...	...	...	...	...	...	...	...	...	
995	996	1700	1	1.9	0	0	1	54	0.5	
996	997	609	0	1.8	1	0	0	13	0.9	
997	998	1185	0	1.4	0	1	1	8	0.5	
998	999	1533	1	0.5	1	0	0	50	0.4	
999	1000	1270	1	0.5	0	4	1	35	0.1	

1000 rows × 23 columns

In [ ]:

```
1
```