2203A51551

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
from plotly.offline import iplot , plot
from plotly.subplots import make_subplots
from sklearn.model_selection import train_test_split , GridSearchCV
from sklearn.metrics import ConfusionMatrixDisplay
from sklearn.preprocessing import MinMaxScaler
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
import warnings
warnings.filterwarnings('ignore')
```

```
df = pd.read_csv("/content/train.csv")
```

```
df.sample(5)
```

	battery_power	blue	clock_speed	dual_sim	fc	four_g	int_memory	m_dep	mobile_wt	n_cores	...	px_height	px_width	ram
670	638	0	1.7	1	11	0	27	0.5	102	4	...	1171	1383	2735
1279	1602	0	0.6	0	12	0	58	0.4	170	1	...	1259	1746	3622
1587	1093	0	0.5	0	1	1	31	0.6	171	2	...	1310	1420	1646
396	788	0	2.5	1	4	1	57	0.9	91	8	...	42	1161	3969
1711	916	1	0.5	0	0	1	47	0.2	82	4	...	1147	1393	1659

5 rows × 21 columns

```
print(f"Number of Row : {df.shape[0]}\nNumber of Columns : {df.shape[1]}")
```

```
Number of Row : 2000
Number of Columns : 21
```

```
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
```

```
df.isna().sum()
```

```
battery_power    0
blue             0
clock_speed      0
dual_sim         0
fc               0
four_g           0
```

```
int_memory       0
m_dep            0
mobile_wt        0
n_cores          0
pc               0
px_height        0
px_width         0
ram              0
sc_h             0
sc_w             0
talk_time        0
three_g          0
touch_screen     0
wifi             0
price_range      0
dtype: int64
```

```
df.describe()
```

	battery_power	blue	clock_speed	dual_sim	fc	four_g	i
count	2000.000000	2000.0000	2000.000000	2000.000000	2000.000000	2000.000000	20
mean	1238.518500	0.4950	1.522250	0.509500	4.309500	0.521500	
std	439.418206	0.5001	0.816004	0.500035	4.341444	0.499662	
min	501.000000	0.0000	0.500000	0.000000	0.000000	0.000000	
25%	851.750000	0.0000	0.700000	0.000000	1.000000	0.000000	
50%	1226.000000	0.0000	1.500000	1.000000	3.000000	1.000000	
75%	1615.250000	1.0000	2.200000	1.000000	7.000000	1.000000	
max	1998.000000	1.0000	3.000000	1.000000	19.000000	1.000000	

8 rows × 21 columns

```
pd.DataFrame({'Count':df.shape[0],
              'Null':df.isnull().sum(),
              'Null %':df.isnull().mean() * 100,
              'Cardinality':df.nunique()
})
```

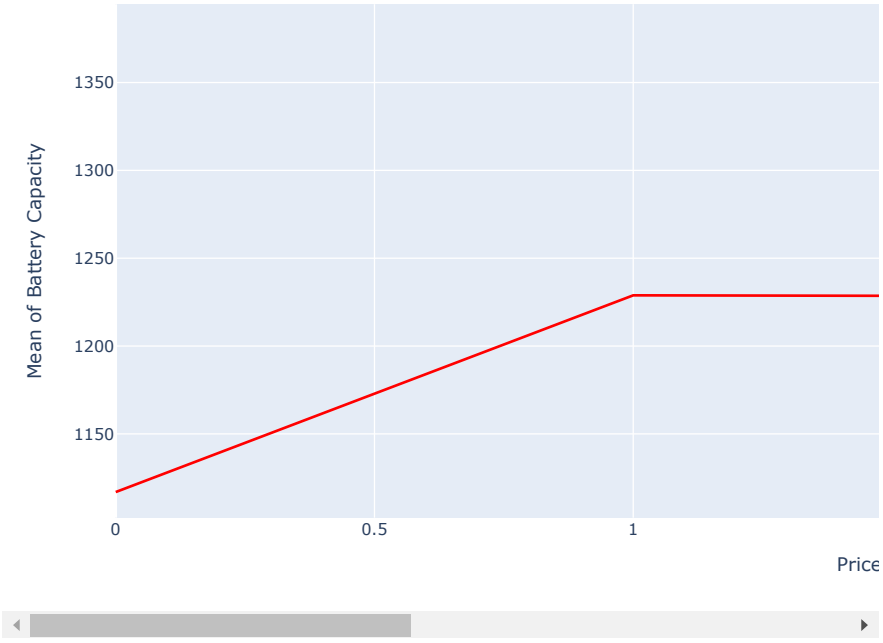
	Count	Null	Null %	Cardinality
battery_power	2000	0	0.0	1094
blue	2000	0	0.0	2
clock_speed	2000	0	0.0	26
dual_sim	2000	0	0.0	2
fc	2000	0	0.0	20
four_g	2000	0	0.0	2
int_memory	2000	0	0.0	63
m_dep	2000	0	0.0	10
mobile_wt	2000	0	0.0	121
n_cores	2000	0	0.0	8
pc	2000	0	0.0	21
px_height	2000	0	0.0	1137
px_width	2000	0	0.0	1109
ram	2000	0	0.0	1562
sc_h	2000	0	0.0	15
sc_w	2000	0	0.0	19
talk_time	2000	0	0.0	19
three_g	2000	0	0.0	2
touch_screen	2000	0	0.0	2
wifi	2000	0	0.0	2
price_range	2000	0	0.0	4

```
df.duplicated().any()
```

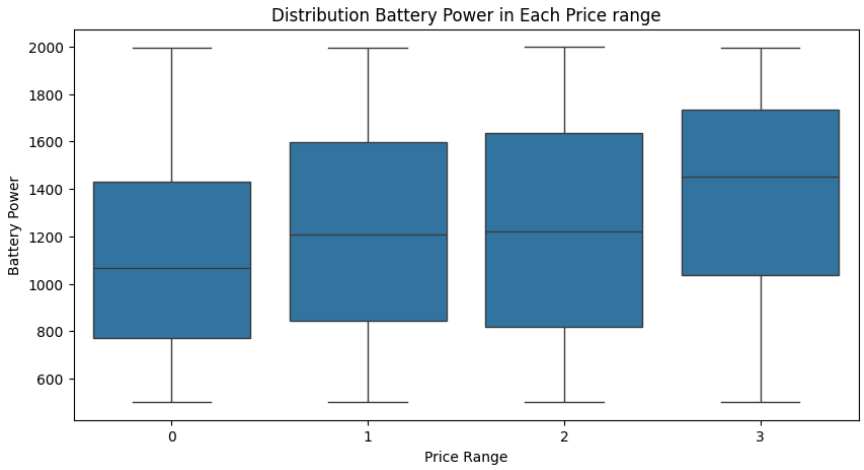
```
False
```

```
df_battery_price = df.groupby('price_range')['battery_power'].mean()
```

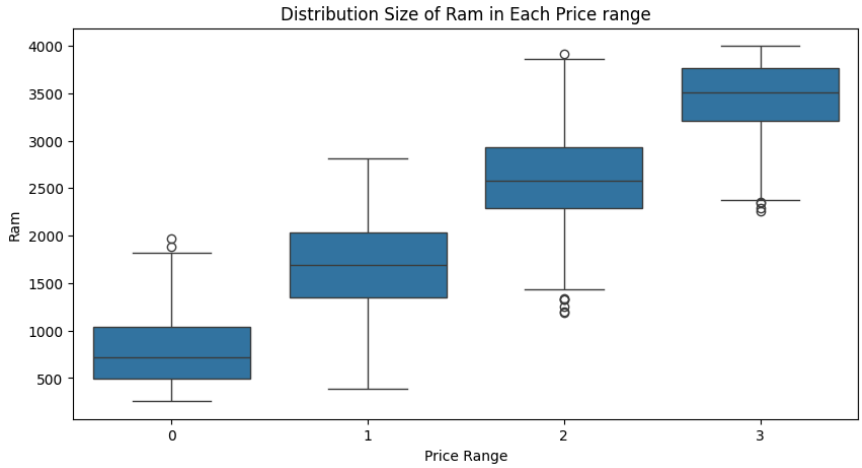
```
iplot(px.line(df_battery_price,
              labels={'value':'Mean of Battery Capacity','price_range':'Price Range'},
              color_discrete_sequence=['red']
              ))
```



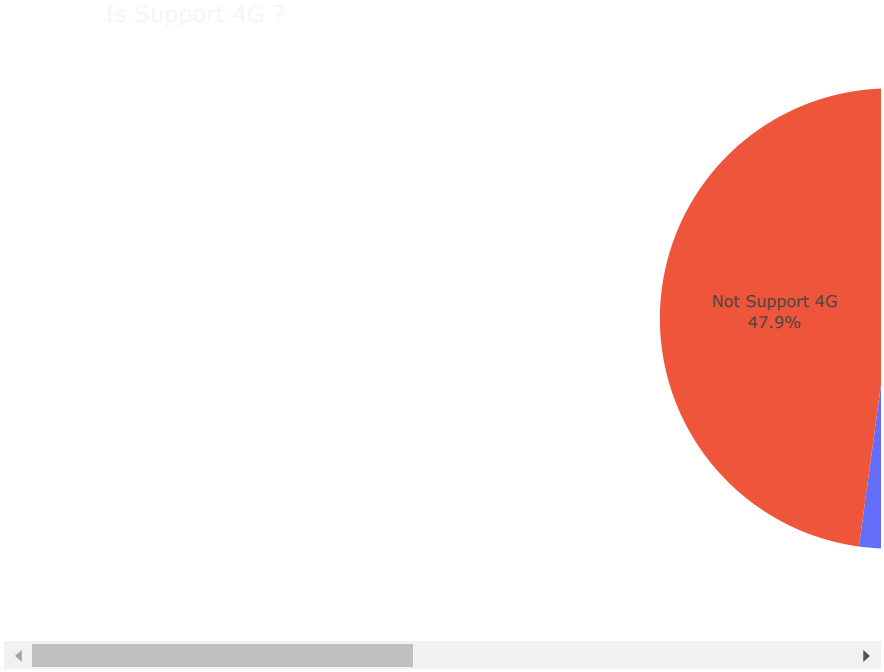

```
plt.figure(figsize=(10,5))
plt.title('Distribution Battery Power in Each Price range')
sns.boxplot(x=df['price_range'],y=df['battery_power'])
plt.xlabel('Price Range')
plt.ylabel('Battery Power')
plt.show()
```



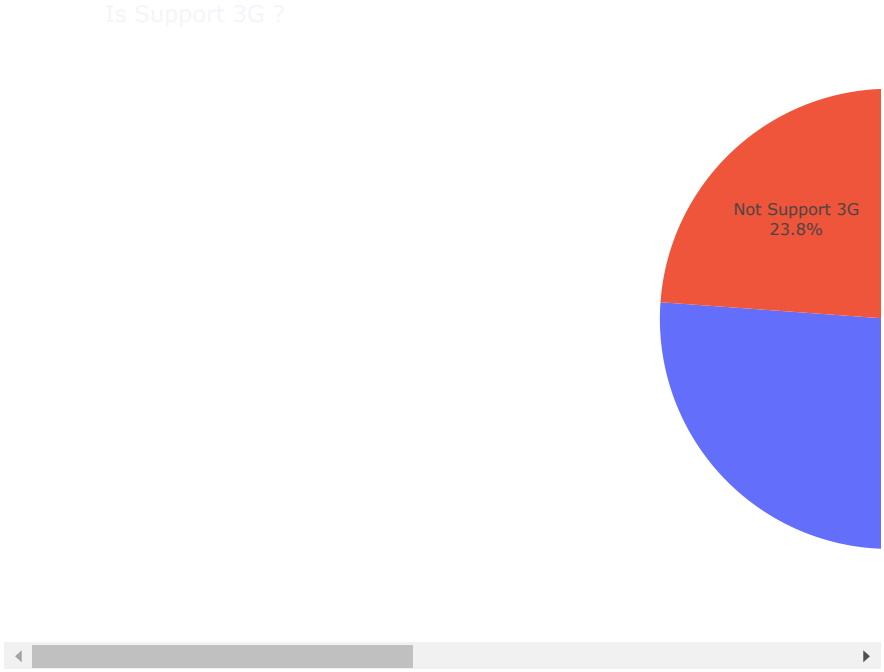

```
plt.figure(figsize=(10,5))
plt.title('Distribution Size of Ram in Each Price range')
sns.boxplot(x=df['price_range'],y=df['ram'])
plt.xlabel('Price Range')
plt.ylabel('Ram')
plt.show()
```




```
df_4g = df['four_g'].value_counts()
iplot(px.pie(values=df_4g,
             names=['Support 4G','Not Support 4G'],
             template='plotly_dark',
             title='Is Support 4G ?'
             ).update_traces(textinfo='label+percent'))
```




```
df_3g = df['three_g'].value_counts()
iplot(px.pie(values=df_3g,
             names=['Support 3G','Not Support 3G'],
             template='plotly_dark',
             title='Is Support 3G ?'
             ).update_traces(textinfo='label+percent'))
```




```
x = df.drop(columns='price_range')
y = df.price_range


scaler = MinMaxScaler()
x = scaler.fit_transform(x)


x_train , x_test , y_train , y_test = train_test_split(x,y,test_size=0.2)


print(f'Shape of X_Train {x_train.shape}')
print(f'Shape of X_Test {x_test.shape}')
print(f'Shape of Y_Train {y_train.shape}')
print(f'Shape of Y_Test {y_test.shape}')
```

```
Shape of X_Train (1600, 20)
Shape of X_Test (400, 20)
Shape of Y_Train (1600,)
Shape of Y_Test (400,)
```

```
model_params = {
    'svm':{
        'model' : SVC(gamma='auto'),
        'params':{
            'C':[1,10,20],
            'kernel':['rbf','linear']
        }
    },
    'random_forest':{
        'model':RandomForestClassifier(),
        'params':{
            'n_estimators':[1,5,10]
        }
    },
    'logistic_regression':{
        'model':LogisticRegression(solver='liblinear',multi_class='auto'),
        'params':{
            'C':[1,5,10]
        }
    }
}
```

```
scores = []

for model_name , mp in model_params.items():
    clf = GridSearchCV(mp['model'],mp['params'],cv=5,return_train_score=False)
    clf.fit(x,y)
    scores.append({
            'model':model_name,
            'best_scores':clf.best_score_,
            'best_params':clf.best_params_
        }
    )
pd.DataFrame(scores,columns=['model','best_scores','best_params'])
```

	model	best_scores	best_params
0	svm	0.9675	{'C': 20, 'kernel': 'linear'}
1	random_forest	0.7905	{'n_estimators': 10}
2	logistic_regression	0.8375	{'C': 10}

```
model_svm = SVC(kernel='linear',C=20)
model_svm.fit(x_train,y_train)
```

```
SVC(C=20, kernel='linear')
```

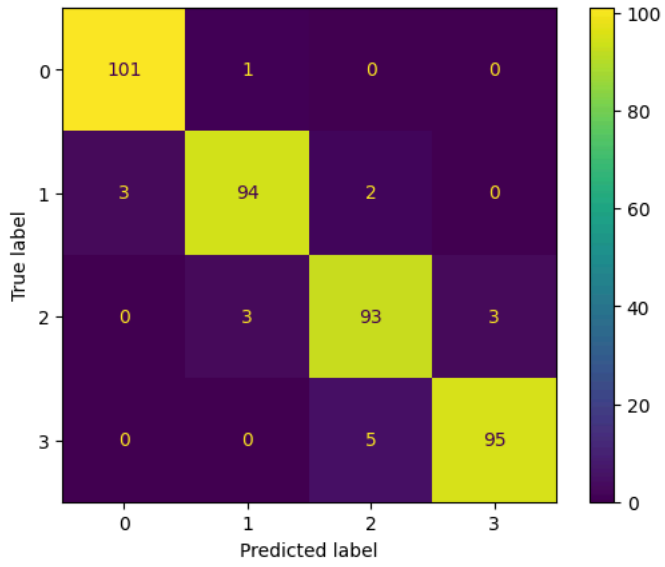
```
score_svm_train = model_svm.score(x_train,y_train)
print(f"Train accuracy: {score_svm_train}")
```

```
Train accuracy: 0.979375
```

```
score_svm_test = model_svm.score(x_test,y_test)
print(f"Test accuracy: {score_svm_test}")
```

```
Test accuracy: 0.9575
```

```
ConfusionMatrixDisplay.from_estimator(model_svm,
                                      x_test,
                                      y_test);
```




```
model_LR = LogisticRegression(C=10)
model_LR.fit(x_train,y_train)
```

```
LogisticRegression(C=10)
```

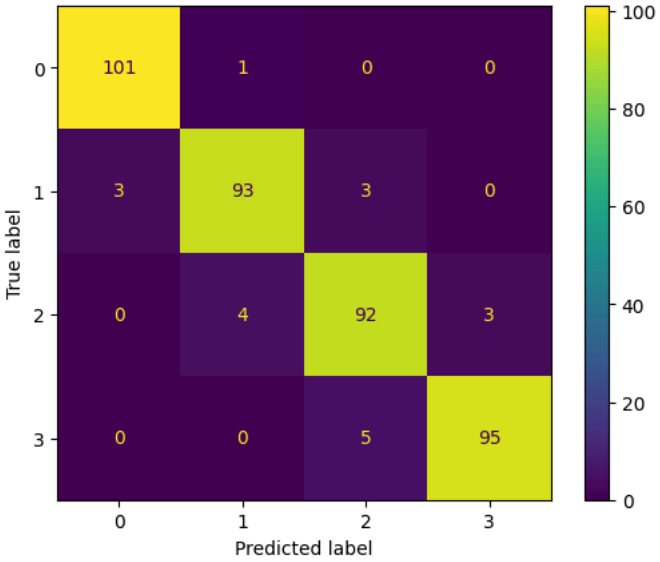
```
score_LR_train = model_LR.score(x_train,y_train)
print(f"Train accuracy: {score_LR_train}")
```

```
Train accuracy: 0.9775
```

```
score_LR_test = model_LR.score(x_test,y_test)
print(f"Test accuracy: {score_LR_test}")
```

```
Test accuracy: 0.9525
```

```
ConfusionMatrixDisplay.from_estimator(model_LR,
                                      x_test,
                                      y_test);
```




```
model_RFC = RandomForestClassifier(n_estimators=10,random_state=42)
model_RFC.fit(x_train,y_train)
```

```
RandomForestClassifier
RandomForestClassifier(n_estimators=10, random_state=42)
```

```
score_RFC_train = model_RFC.score(x_train,y_train)
print(f"Train accuracy: {score_RFC_train}")
```

```
Train accuracy: 0.9975
```

```
score_RFC_test = model_RFC.score(x_test,y_test)
print(f"Test accuracy: {score_RFC_test}")
```

```
Test accuracy: 0.795
```

```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import GradientBoostingRegressor
import xgboost as xgb
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score



# Training Gradient Boosting model
gb_model = GradientBoostingRegressor(n_estimators=100, learning_rate=0.1, max_depth=3, random_state=42)
gb_model.fit(x_train, y_train)

# Training XGBoost model
xgb_model = xgb.XGBRegressor(n_estimators=100, learning_rate=0.1, max_depth=3, random_state=42)
xgb_model.fit(x_train, y_train)

# Making predictions on the testing set
gb_predictions = gb_model.predict(x_test)
xgb_predictions = xgb_model.predict(x_test)
```

```
mae = mean_absolute_error(y_test, gb_predictions)
print("Mean Absolute Error (MAE):", mae)

# Calculate Mean Squared Error (MSE)
mse = mean_squared_error(y_test, gb_predictions)
print("Mean Squared Error (MSE):", mse)

# Calculate Root Mean Squared Error (RMSE)
rmse = mean_squared_error(y_test, gb_predictions, squared=False)
print("Root Mean Squared Error (RMSE):", rmse)

# Calculate R-squared (R2) score
r2 = r2_score(y_test, gb_predictions)
print("R-squared (R2) Score:", r2)
```

```
Mean Absolute Error (MAE): 0.22669212491592525
Mean Squared Error (MSE): 0.08024038069086553
Root Mean Squared Error (RMSE): 0.2832673307864243
R-squared (R2) Score: 0.9363143150709184
```

```
# Calculate Mean Absolute Error (MAE)
mae = mean_absolute_error(y_test, xgb_predictions)
print("Mean Absolute Error (MAE):", mae)

# Calculate Mean Squared Error (MSE)
mse = mean_squared_error(y_test, xgb_predictions)
```