```
#first load all the important library is needed for prediction
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
import plotly.express as px
import plotly.io as pio
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report
```

```
#collect the data from the file as csv data
df=pd.read_csv("/content/mobile_data (1).csv")
```

```
#load the mobile dataset by using read_csv function which load the csv file and it make the data tabular form
df
```

| | battery_power | blue | clock_speed | dual_sim | fc | four_g | int_memory | m_dep | mobile_wt | n_cores | ... | px_height | px_widt |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 842 | 0 | 2.2 | 0 | 1 | 0 | 7 | 0.6 | 188 | 2 | ... | 20 | 75 |
| **1** | 1021 | 1 | 0.5 | 1 | 0 | 1 | 53 | 0.7 | 136 | 3 | ... | 905 | 198 |
| **2** | 563 | 1 | 0.5 | 1 | 2 | 1 | 41 | 0.9 | 145 | 5 | ... | 1263 | 171 |
| **3** | 615 | 1 | 2.5 | 0 | 0 | 0 | 10 | 0.8 | 131 | 6 | ... | 1216 | 178 |
| **4** | 1821 | 1 | 1.2 | 0 | 13 | 1 | 44 | 0.6 | 141 | 2 | ... | 1208 | 121 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | . |
| **1995** | 794 | 1 | 0.5 | 1 | 0 | 1 | 2 | 0.8 | 106 | 6 | ... | 1222 | 189 |
| **1996** | 1965 | 1 | 2.6 | 1 | 0 | 0 | 39 | 0.2 | 187 | 4 | ... | 915 | 196 |
| **1997** | 1911 | 0 | 0.9 | 1 | 1 | 1 | 36 | 0.7 | 108 | 8 | ... | 868 | 163 |
| **1998** | 1512 | 0 | 0.9 | 0 | 4 | 1 | 46 | 0.1 | 145 | 5 | ... | 336 | 67 |
| **1999** | 510 | 1 | 2.0 | 1 | 5 | 1 | 45 | 0.9 | 168 | 6 | ... | 483 | 75 |

2000 rows × 21 columns

```
#perform EDA analysis
```

```
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.describe()
```

| | battery_power | blue | clock_speed | dual_sim | fc | four_g | int_memory | m_dep | mobile_wt | |
|---|---|---|---|---|---|---|---|---|---|---|
| **count** | 2000.000000 | 2000.0000 | 2000.000000 | 2000.000000 | 2000.000000 | 2000.000000 | 2000.000000 | 2000.000000 | 2000.000000 | 20( |
| **mean** | 1238.518500 | 0.4950 | 1.522250 | 0.509500 | 4.309500 | 0.521500 | 32.046500 | 0.501750 | 140.249000 | |
| **std** | 439.418206 | 0.5001 | 0.816004 | 0.500035 | 4.341444 | 0.499662 | 18.145715 | 0.288416 | 35.399655 | |
| **min** | 501.000000 | 0.0000 | 0.500000 | 0.000000 | 0.000000 | 0.000000 | 2.000000 | 0.100000 | 80.000000 | |
| **25%** | 851.750000 | 0.0000 | 0.700000 | 0.000000 | 1.000000 | 0.000000 | 16.000000 | 0.200000 | 109.000000 | |
| **50%** | 1226.000000 | 0.0000 | 1.500000 | 1.000000 | 3.000000 | 1.000000 | 32.000000 | 0.500000 | 141.000000 | |
| **75%** | 1615.250000 | 1.0000 | 2.200000 | 1.000000 | 7.000000 | 1.000000 | 48.000000 | 0.800000 | 170.000000 | |
| **max** | 1998.000000 | 1.0000 | 3.000000 | 1.000000 | 19.000000 | 1.000000 | 64.000000 | 1.000000 | 200.000000 | |

8 rows × 21 columns

```
df.corr()
```

| | battery_power | blue | clock_speed | dual_sim | fc | four_g | int_memory | m_dep | mobile_wt | n_core |
|---|---|---|---|---|---|---|---|---|---|---|
| **battery_power** | 1.000000 | 0.011252 | 0.011482 | -0.041847 | 0.033334 | 0.015665 | -0.004004 | 0.034085 | 0.001844 | -0.02972 |
| **blue** | 0.011252 | 1.000000 | 0.021419 | 0.035198 | 0.003593 | 0.013443 | 0.041177 | 0.004049 | -0.008605 | 0.03616 |
| **clock_speed** | 0.011482 | 0.021419 | 1.000000 | -0.001315 | -0.000434 | -0.043073 | 0.006545 | -0.014364 | 0.012350 | -0.00572 |
| **dual_sim** | -0.041847 | 0.035198 | -0.001315 | 1.000000 | -0.029123 | 0.003187 | -0.015679 | -0.022142 | -0.008979 | -0.02465 |
| **fc** | 0.033334 | 0.003593 | -0.000434 | -0.029123 | 1.000000 | -0.016560 | -0.029133 | -0.001791 | 0.023618 | -0.01335 |
| **four_g** | 0.015665 | 0.013443 | -0.043073 | 0.003187 | -0.016560 | 1.000000 | 0.008690 | -0.001823 | -0.016537 | -0.02970 |
| **int_memory** | -0.004004 | 0.041177 | 0.006545 | -0.015679 | -0.029133 | 0.008690 | 1.000000 | 0.006886 | -0.034214 | -0.02831 |
| **m_dep** | 0.034085 | 0.004049 | -0.014364 | -0.022142 | -0.001791 | -0.001823 | 0.006886 | 1.000000 | 0.021756 | -0.00350 |
| **mobile_wt** | 0.001844 | -0.008605 | 0.012350 | -0.008979 | 0.023618 | -0.016537 | -0.034214 | 0.021756 | 1.000000 | -0.01898 |
| **n_cores** | -0.029727 | 0.036161 | -0.005724 | -0.024658 | -0.013356 | -0.029706 | -0.028310 | -0.003504 | -0.018989 | 1.00000 |
| **pc** | 0.031441 | -0.009952 | -0.005245 | -0.017143 | 0.644595 | -0.005598 | -0.033273 | 0.026282 | 0.018844 | -0.00119 |
| **px_height** | 0.014901 | -0.006872 | -0.014523 | -0.020875 | -0.009990 | -0.019236 | 0.010441 | 0.025263 | 0.000939 | -0.00687 |
| **px_width** | -0.008402 | -0.041533 | -0.009476 | 0.014291 | -0.005176 | 0.007448 | -0.008335 | 0.023566 | 0.000090 | 0.02448 |
| **ram** | -0.000653 | 0.026351 | 0.003443 | 0.041072 | 0.015099 | 0.007313 | 0.032813 | -0.009434 | -0.002581 | 0.00486 |
| **sc_h** | -0.029959 | -0.002952 | -0.029078 | -0.011949 | -0.011014 | 0.027166 | 0.037771 | -0.025348 | -0.033855 | -0.00031 |
| **sc_w** | -0.021421 | 0.000613 | -0.007378 | -0.016666 | -0.012373 | 0.037005 | 0.011731 | -0.018388 | -0.020761 | 0.02582 |
| **talk_time** | 0.052510 | 0.013934 | -0.011432 | -0.039404 | -0.006829 | -0.046628 | -0.002790 | 0.017003 | 0.006209 | 0.01314 |
| **three_g** | 0.011522 | -0.030236 | -0.046433 | -0.014008 | 0.001793 | 0.584246 | -0.009366 | -0.012065 | 0.001551 | -0.01473 |
| **touch_screen** | -0.010516 | 0.010061 | 0.019756 | -0.017117 | -0.014828 | 0.016758 | -0.026999 | -0.002638 | -0.014368 | 0.02377 |
| **wifi** | -0.008343 | -0.021863 | -0.024471 | 0.022740 | 0.020085 | -0.017620 | 0.006993 | -0.028353 | -0.000409 | -0.00996 |
| **price_range** | 0.200723 | 0.020573 | -0.006606 | 0.017444 | 0.021998 | 0.014772 | 0.044435 | 0.000853 | -0.030302 | 0.00439 |

21 rows × 21 columns

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

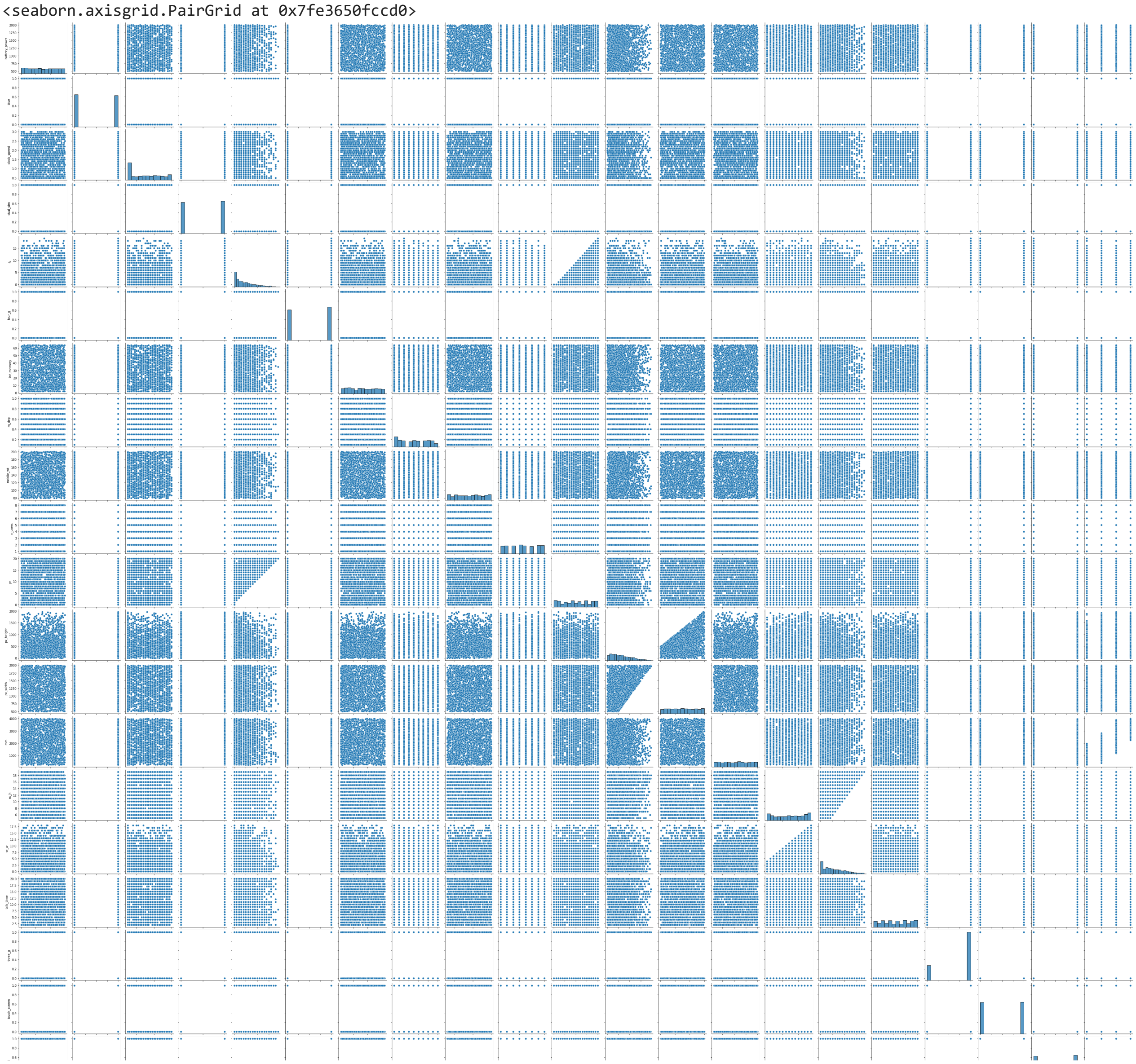
```
battery_power    0
blue             0
clock_speed      0
dual_sim         0
fc               0
four_g           0
int_memory       0
m_dep            0
mobile_wt        0
n_cores          0
pc               0
px_height        0
px_width         0
ram              0
sc_h             0
sc_w             0
talk_time        0
three_g          0
touch_screen     0
wifi             0
price_range      0
dtype: int64
```

```
#visualization of different features by using different graphs representation
```

```
#  we use the seaborn.pair plot for visualization weather there is linear or non linear model or the data having more correration
```

```
sns.pairplot(df)
```

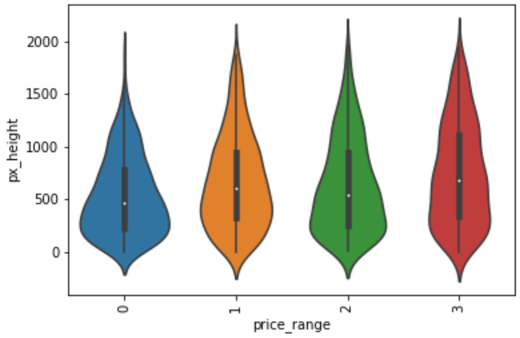
```
<seaborn.axisgrid.PairGrid at 0x7fe3650fccd0>
```



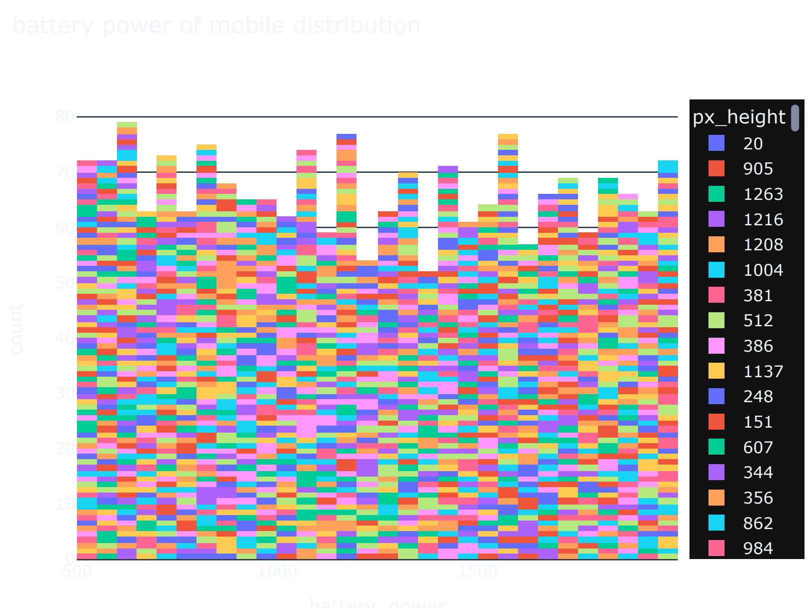
```
# we will see the visualization of battery vs price_range
```

```
# Violin plot
sns.violinplot(x='price_range', y='px_height', data=df)
plt.xticks(rotation=90)
```

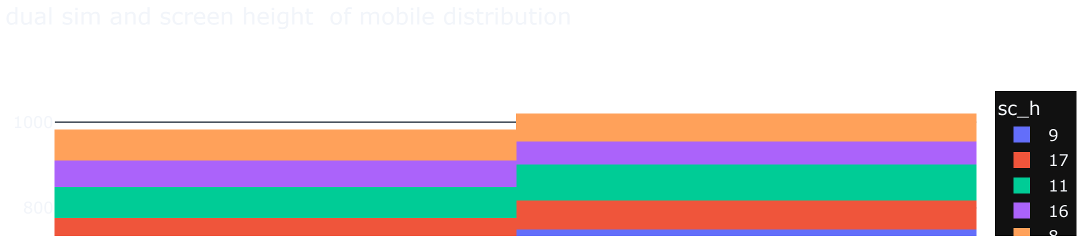
```
(array([0, 1, 2, 3]), <a list of 4 Text major ticklabel objects>)
```



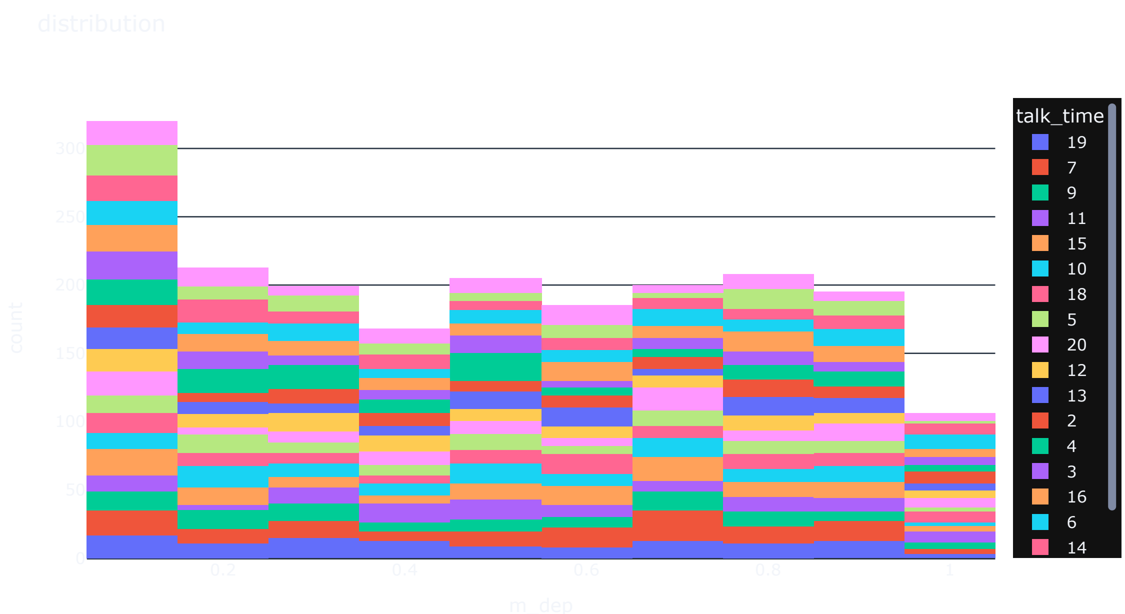

```
#we use the ploty library for interactive visualization
ax= px.histogram(df,x= "battery_power", template= "plotly_dark",color= "px_height",title='battery power of mobile distribution')
ax.show()
```



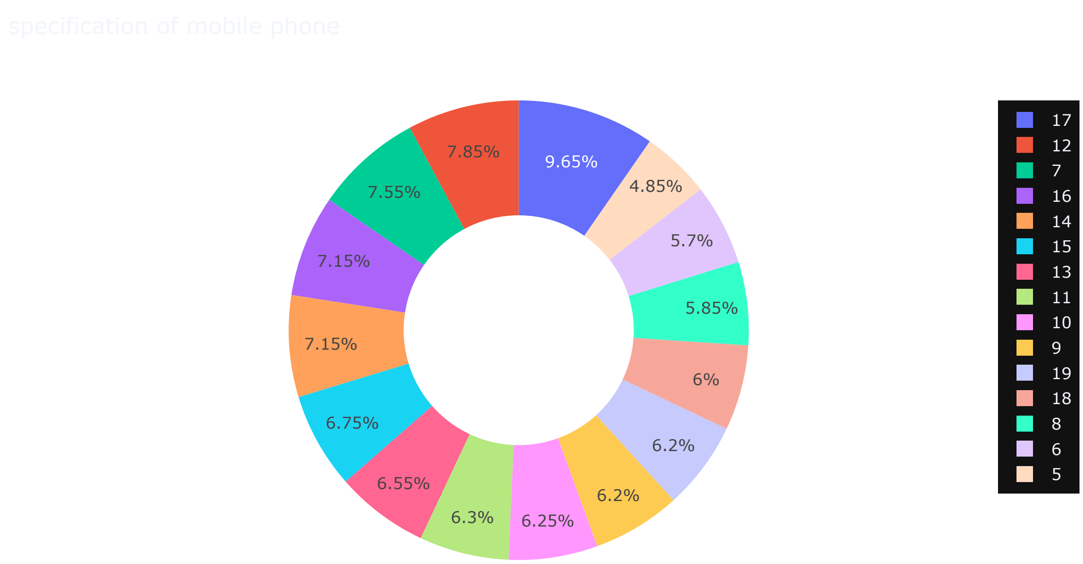

```
ax= px.histogram(df,x= "dual_sim", template= "plotly_dark",color= "sc_h",title='dual sim and screen height  of mobile distribution')
ax.show()
```




```
ax= px.histogram(df,x= "m_dep", template= "plotly_dark",color= "talk_time",title=' distribution')
ax.show()
```

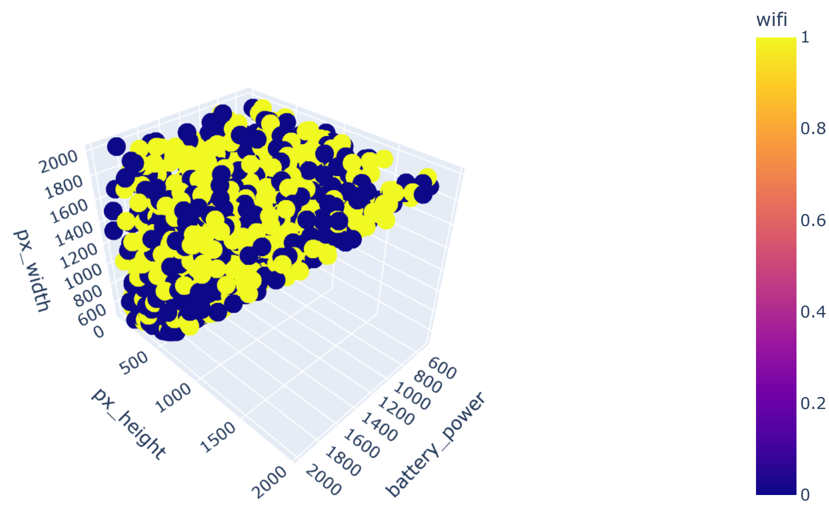



```
ax= px.pie(df, names= "sc_h",template= "plotly_dark",title= "specification of mobile phone",hole= 0.5)
ax.show()
```



```
#scatter plot
```

```
fig = px.scatter_3d(df, x='battery_power', y='px_height', z='px_width',
              color='wifi')
fig.show()
```




```
#we import plotly for better presentation
```

```
#sepaerating x and y for taining and testing
```

```
x = df.iloc[:,: -1]
y = df["price_range"]
```

```
x
```

| | battery_power | blue | clock_speed | dual_sim | fc | four_g | int_memory | m_dep | mobile_wt | n_cores | pc | px_height | px_width |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 842 | 0 | 2.2 | 0 | 1 | 0 | 7 | 0.6 | 188 | 2 | 2 | 20 | 756 |
| **1** | 1021 | 1 | 0.5 | 1 | 0 | 1 | 53 | 0.7 | 136 | 3 | 6 | 905 | 1988 |
| **2** | 563 | 1 | 0.5 | 1 | 2 | 1 | 41 | 0.9 | 145 | 5 | 6 | 1263 | 1716 |
| **3** | 615 | 1 | 2.5 | 0 | 0 | 0 | 10 | 0.8 | 131 | 6 | 9 | 1216 | 1786 |
| **4** | 1821 | 1 | 1.2 | 0 | 13 | 1 | 44 | 0.6 | 141 | 2 | 14 | 1208 | 1212 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **1995** | 794 | 1 | 0.5 | 1 | 0 | 1 | 2 | 0.8 | 106 | 6 | 14 | 1222 | 1890 |
| **1996** | 1965 | 1 | 2.6 | 1 | 0 | 0 | 39 | 0.2 | 187 | 4 | 3 | 915 | 1965 |
| **1997** | 1911 | 0 | 0.9 | 1 | 1 | 1 | 36 | 0.7 | 108 | 8 | 3 | 868 | 1632 |
| **1998** | 1512 | 0 | 0.9 | 0 | 4 | 1 | 46 | 0.1 | 145 | 5 | 5 | 336 | 670 |
| **1999** | 510 | 1 | 2.0 | 1 | 5 | 1 | 45 | 0.9 | 168 | 6 | 16 | 483 | 754 |

2000 rows × 20 columns

```
y
```

```
0       1
1       2
```

```
2       2
3       2
4       1
       ..
1995    0
1996    2
1997    3
1998    0
1999    3
Name: price_range, Length: 2000, dtype: int64
```

```
#split the dats x and y for training and testing
```

```
xtrain, xtest, ytrain, ytest = train_test_split(x,y, test_size=0.3, random_state=1)
```

```
#import library of linear regression
```

```
# Fitting Multiple Linear Regression to the Training set
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(xtrain, ytrain)
```

```
LinearRegression()
```

```
# Predicting the Test set results
ypred = regressor.predict(xtest)
```

```
#evaluate the model
from sklearn.metrics import r2_score
```

```
r2_score(ytest,ypred)
```

```
0.909367881466235
```

```
#loss function
from sklearn.metrics import mean_absolute_error
```

```
mean_absolute_error(ytest,ypred)   #MAE
```

```
0.2751015081723019
```

```
from sklearn.metrics import mean_squared_error  #MSE
```

```
mean_squared_error(ytest,ypred)
```

```
0.10794260141782923
```

```
np.sqrt(mean_squared_error(ytest,ypred))
```

```
0.32854619373511124
```

```
#building different models by using classification ml algorithm
logreg = LogisticRegression()
knn = KNeighborsClassifier()
svm = SVC()
dt=DecisionTreeClassifier()
```

```
#using define function to create the fifferent model
```

```
def mymodel(model):
  model.fit(xtrain, ytrain)
  ypred = model.predict(xtest)
  print(classification_report(ytest, ypred))
  return model
```

```
mymodel(logreg)
```

```
              precision    recall  f1-score   support

           0       0.78      0.80      0.79       135
           1       0.53      0.58      0.55       149
           2       0.53      0.40      0.45       168
           3       0.67      0.78      0.72       148

    accuracy                           0.63       600
   macro avg       0.63      0.64      0.63       600
weighted avg       0.62      0.63      0.62       600

LogisticRegression()
```

```
mymodel(knn)
```

```
              precision    recall  f1-score   support

           0       0.96      0.98      0.97       135
           1       0.91      0.91      0.91       149
           2       0.88      0.89      0.88       168
           3       0.94      0.91      0.93       148

    accuracy                           0.92       600
   macro avg       0.92      0.92      0.92       600
weighted avg       0.92      0.92      0.92       600

KNeighborsClassifier()
```

```
mymodel(svm)
```

```
              precision    recall  f1-score   support

           0       0.95      0.99      0.97       135
           1       0.93      0.93      0.93       149
           2       0.94      0.90      0.92       168
           3       0.95      0.97      0.96       148

    accuracy                           0.94       600
   macro avg       0.94      0.95      0.94       600
weighted avg       0.94      0.94      0.94       600

SVC()
```

```
mymodel(dt)
```

```
              precision    recall  f1-score   support

           0       0.91      0.86      0.89       135
           1       0.77      0.85      0.81       149
           2       0.84      0.80      0.82       168
           3       0.91      0.91      0.91       148

    accuracy                           0.85       600
   macro avg       0.86      0.86      0.86       600
weighted avg       0.86      0.85      0.85       600

DecisionTreeClassifier()
```

```
#feature scaling
```

```
from sklearn.preprocessing import StandardScaler
```

```
sc=StandardScaler()
xtrain=sc.fit_transform(xtrain)
```

```
xtest=sc.transform(xtest)
```

```
#hypertuning using solver parameter
```

```
logreg=LogisticRegression(solver="saga")
logreg.fit(xtrain,ytrain)
ypred=logreg.predict(xtest)
```

```
print(classification_report(ytest,ypred))
```

```
              precision    recall  f1-score   support

           0       0.96      0.99      0.97       135
           1       0.93      0.92      0.93       149
           2       0.92      0.89      0.91       168
           3       0.93      0.95      0.94       148

    accuracy                           0.94       600
   macro avg       0.94      0.94      0.94       600
weighted avg       0.93      0.94      0.93       600
```

```
#checking overfitted on dt
dt.score(xtrain,ytrain)
```

```
0.26071428571428573
```

```
dt1=DecisionTreeClassifier(max_depth=5)
mymodel(dt1)
```

```
              precision    recall  f1-score   support

           0       0.95      0.85      0.90       135
           1       0.72      0.91      0.81       149
           2       0.81      0.71      0.76       168
           3       0.88      0.86      0.87       148

    accuracy                           0.83       600
   macro avg       0.84      0.83      0.83       600
weighted avg       0.84      0.83      0.83       600

DecisionTreeClassifier(max_depth=5)
```

```
dt2=DecisionTreeClassifier(max_depth=10)
mymodel(dt2)
```

```
              precision    recall  f1-score   support

           0       0.91      0.88      0.89       135
           1       0.77      0.85      0.81       149
           2       0.86      0.78      0.82       168
           3       0.89      0.92      0.91       148

    accuracy                           0.85       600
   macro avg       0.86      0.86      0.86       600
weighted avg       0.86      0.85      0.86       600

DecisionTreeClassifier(max_depth=10)
```

```
for i in range(1,50):
 dt1=DecisionTreeClassifier(max_depth=i)
 dt2.fit(xtrain,ytrain)
 ypred=dt2.predict(xtest)
 print(f"{i}= {accuracy_score(ytest,ypred)}")
```

```
1= 0.8566666666666667
2= 0.8583333333333333
3= 0.8466666666666667
4= 0.8516666666666667
5= 0.85
6= 0.8483333333333334
7= 0.8666666666666667
8= 0.8466666666666667
9= 0.8516666666666667
10= 0.85
11= 0.8566666666666667
12= 0.865
13= 0.8583333333333333
14= 0.8583333333333333
15= 0.865
16= 0.86
17= 0.8516666666666667
18= 0.8616666666666667
19= 0.8433333333333334
20= 0.8583333333333333
21= 0.86
22= 0.86
23= 0.8533333333333334
24= 0.8533333333333334
```

```
25= 0.86
26= 0.86
27= 0.8483333333333334
28= 0.855
29= 0.8516666666666667
30= 0.8533333333333334
31= 0.8616666666666667
32= 0.85
33= 0.8566666666666667
34= 0.8616666666666667
35= 0.8533333333333334
36= 0.8583333333333333
37= 0.855
38= 0.8516666666666667
39= 0.855
40= 0.8583333333333333
41= 0.865
42= 0.8533333333333334
43= 0.855
44= 0.855
45= 0.8566666666666667
46= 0.8566666666666667
47= 0.8583333333333333
48= 0.86
49= 0.8566666666666667
```

```
#best value of max_depth=10
dt3=DecisionTreeClassifier(max_depth=11)
mymodel(dt3)
```

```
              precision    recall  f1-score   support

           0       0.91      0.85      0.88       135
           1       0.75      0.84      0.79       149
           2       0.86      0.77      0.81       168
           3       0.89      0.95      0.92       148

    accuracy                           0.85       600
   macro avg       0.85      0.85      0.85       600
weighted avg       0.85      0.85      0.85       600

DecisionTreeClassifier(max_depth=11)
```

```
#gini
dt8=DecisionTreeClassifier(criterion="gini",min_samples_leaf=15)
mymodel(dt8)
```

```
              precision    recall  f1-score   support

           0       0.86      0.85      0.86       135
           1       0.75      0.81      0.78       149
           2       0.81      0.80      0.80       168
           3       0.90      0.86      0.88       148

    accuracy                           0.83       600
   macro avg       0.83      0.83      0.83       600
weighted avg       0.83      0.83      0.83       600

DecisionTreeClassifier(min_samples_leaf=15)
```

```
dt9=DecisionTreeClassifier(criterion="gini",max_depth=35)
mymodel(dt9)
```

```
              precision    recall  f1-score   support

           0       0.93      0.87      0.90       135
           1       0.77      0.86      0.81       149
           2       0.85      0.77      0.81       168
           3       0.88      0.93      0.91       148

    accuracy                           0.85       600
   macro avg       0.86      0.86      0.86       600
weighted avg       0.86      0.85      0.85       600

DecisionTreeClassifier(max_depth=35)
```

```
for i in range(1,70):
 dt11=DecisionTreeClassifier(criterion="entropy",max_depth=i)
 dt11.fit(xtrain,ytrain)
```

```
ypred=dt11.predict(xtest)
print(f"{i}= {accuracy_score(ytest,ypred)}")
```

```
12= 0.8533333333333334
13= 0.8416666666666667
14= 0.8466666666666667
15= 0.835
16= 0.855
17= 0.8416666666666667
18= 0.8466666666666667
19= 0.8466666666666667
20= 0.8433333333333334
21= 0.8466666666666667
22= 0.8316666666666667
23= 0.8433333333333334
24= 0.8566666666666667
25= 0.84
26= 0.835
27= 0.8433333333333334
28= 0.8383333333333334
29= 0.8383333333333334
30= 0.84
31= 0.8483333333333334
32= 0.8366666666666667
33= 0.845
34= 0.8433333333333334
35= 0.8466666666666667
36= 0.8533333333333334
37= 0.8516666666666667
38= 0.8316666666666667
39= 0.8533333333333334
40= 0.8316666666666667
41= 0.8466666666666667
42= 0.84
43= 0.84
44= 0.8433333333333334
45= 0.835
46= 0.83
47= 0.8433333333333334
48= 0.8416666666666667
49= 0.8533333333333334
50= 0.8483333333333334
51= 0.8366666666666667
52= 0.8416666666666667
53= 0.84
54= 0.8533333333333334
55= 0.8516666666666667
56= 0.8516666666666667
57= 0.8466666666666667
58= 0.8416666666666667
59= 0.84
60= 0.8433333333333334
61= 0.845
62= 0.845
63= 0.845
64= 0.8483333333333334
65= 0.8483333333333334
66= 0.85
67= 0.8416666666666667
68= 0.8566666666666667
69= 0.85
```

```
#best value of max_depth when criterion=entropy
dt12=DecisionTreeClassifier(criterion="entropy",max_depth=50)
mymodel(dt12)
```

```
              precision    recall  f1-score   support

           0       0.90      0.84      0.87       135
           1       0.75      0.87      0.80       149
           2       0.83      0.79      0.81       168
           3       0.90      0.87      0.89       148

    accuracy                           0.84       600
   macro avg       0.85      0.84      0.84       600
weighted avg       0.84      0.84      0.84       600

DecisionTreeClassifier(criterion='entropy', max_depth=50)
```

```
# import Random Forest classifier

from sklearn.ensemble import RandomForestClassifier
```

```
# instantiate the classifier

rfc = RandomForestClassifier(random_state=0)



# fit the model

rfc.fit(xtrain, ytrain)



# Predict the Test set results

ypred = rfc.predict(xtest)



# Check accuracy score

from sklearn.metrics import accuracy_score

print('Model accuracy score with 10 decision-trees : {0:0.4f}'. format(accuracy_score(ytest, ypred)))
```

```
     Model accuracy score with 10 decision-trees : 0.8650
```

```
#we import baggingclassifier model
from sklearn.ensemble import BaggingClassifier


bg=BaggingClassifier(LogisticRegression())
bg.fit(xtrain,ytrain)
ypred=bg.predict(xtest)
print(classification_report(ytest,ypred))
```

```
              precision    recall  f1-score   support

           0       0.96      0.98      0.97       135
           1       0.93      0.92      0.92       149
           2       0.91      0.89      0.90       168
           3       0.93      0.94      0.94       148

    accuracy                           0.93       600
   macro avg       0.93      0.93      0.93       600
weighted avg       0.93      0.93      0.93       600
```

```
bg=BaggingClassifier(DecisionTreeClassifier())
bg.fit(xtrain,ytrain)
ypred=bg.predict(xtest)
print(classification_report(ytest,ypred))
```

```
              precision    recall  f1-score   support

           0       0.91      0.93      0.92       135
           1       0.81      0.83      0.82       149
           2       0.81      0.84      0.82       168
           3       0.95      0.85      0.90       148

    accuracy                           0.86       600
   macro avg       0.87      0.86      0.86       600
weighted avg       0.86      0.86      0.86       600
```

```
models=[]
accuracy=[]
models.append(("logistic regression",LogisticRegression()))
models.append(("Decision Tree",DecisionTreeClassifier()))
```

```
models
```

```
[('logistic regression', LogisticRegression()),
 ('Decision Tree', DecisionTreeClassifier())]
```

```
vc=VotingClassifier(estimators=models)
vc.fit(xtrain,ytrain)
ypred=vc.predict(xtest)
print(classification_report(ytest,ypred))
```

```
              precision    recall  f1-score   support

           0       0.89      1.00      0.94       135
           1       0.86      0.88      0.87       149
           2       0.88      0.84      0.86       168
           3       0.96      0.89      0.92       148

    accuracy                           0.90       600
   macro avg       0.90      0.90      0.90       600
weighted avg       0.90      0.90      0.90       600
```

```
# Boosting
```

```
from sklearn.ensemble import GradientBoostingClassifier
```

```
gd=GradientBoostingClassifier()
gd.fit(xtrain,ytrain)
ypred=gd.predict(xtest)
print(classification_report(ytest,ypred))
```

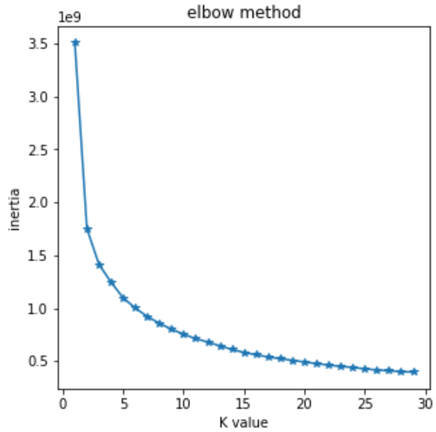
```
              precision    recall  f1-score   support

           0       0.97      0.95      0.96       135
           1       0.84      0.89      0.87       149
           2       0.86      0.83      0.84       168
           3       0.93      0.93      0.93       148

    accuracy                           0.90       600
   macro avg       0.90      0.90      0.90       600
weighted avg       0.90      0.90      0.90       600
```

```
# Here we don't have domain knowledge hence we can not decide the value of K first.
#so we are using elbow method to decide the value of K.
```

Elbow method

```
wcss=[]
for k in range(1,30):
    km=KMeans(n_clusters=k,init="k-means++",n_init=10,max_iter=300,random_state=1)
    km.fit(x)
    wcss.append(km.inertia_)
```

```
plt.figure(figsize=(5,5))
plt.title("elbow method")
plt.plot(range(1,30),wcss,marker="*")
plt.xlabel("K value")
plt.ylabel("inertia")
plt.show()
```




0s completed at 13:30