Loading Data

In [1]:

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
%matplotlib inline

from sklearn.metrics import classification_report,confusion_matrix
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
```

In [2]:

```
main_dataset = pd.read_csv("/Users/uttkarshraj/Documents/DATA ANALYTICS/train.csv")
```

In [3]:

```
main_dataset.shape
```

Out[3]:

```
(2000, 21)
```

##PRE-PROCESSING

In [4]:

```
#Return DataFrame with duplicate rows removed
main_dataset = main_dataset.drop_duplicates()

#Remove missing values.
main_dataset = main_dataset.dropna()
main_dataset.shape

#Return number of unique elements in the object
main_dataset.nunique()
main_dataset.shape
```

Out[4]:

```
(2000, 21)
```

In [5]:

```
main_dataset.describe()
```

Out[5]:

	Battery	Bluetooth	Clock Speed	Dual Sim	Front Camera	4G	Interna Memor
count	2000.000000	2000.0000	2000.000000	2000.000000	2000.000000	2000.000000	2000.00000(
mean	1238.518500	0.4950	1.522250	0.509500	4.309500	0.521500	32.04650(
std	439.418206	0.5001	0.816004	0.500035	4.341444	0.499662	18.14571!
min	501.000000	0.0000	0.500000	0.000000	0.000000	0.000000	2.00000(
25%	851.750000	0.0000	0.700000	0.000000	1.000000	0.000000	16.00000(
50%	1226.000000	0.0000	1.500000	1.000000	3.000000	1.000000	32.00000(
75%	1615.250000	1.0000	2.200000	1.000000	7.000000	1.000000	48.00000(
max	1998.000000	1.0000	3.000000	1.000000	19.000000	1.000000	64.00000(

8 rows × 21 columns

In [6]:

```
main_dataset.head()
```

Out[6]:

	Battery	Bluetooth	Clock Speed	Dual Sim	Front Camera	4G	Internal Memory	Mobile Depth	Mobile Weight	Processors	...	Pi Hei
0	842	0	2.2	0	1	0	7	0.6	188	2	...	
1	1021	1	0.5	1	0	1	53	0.7	136	3	...	9
2	563	1	0.5	1	2	1	41	0.9	145	5	...	12
3	615	1	2.5	0	0	0	10	0.8	131	6	...	12
4	1821	1	1.2	0	13	1	44	0.6	141	2	...	12

5 rows × 21 columns

Analysing Data

In [7]:

```
main_dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2000 entries, 0 to 1999
Data columns (total 21 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   Battery          2000 non-null   int64  
 1   Bluetooth        2000 non-null   int64  
 2   Clock Speed      2000 non-null   float64
 3   Dual Sim         2000 non-null   int64  
 4   Front Camera     2000 non-null   int64  
 5   4G               2000 non-null   int64  
 6   Internal Memory  2000 non-null   int64  
 7   Mobile Depth     2000 non-null   float64
 8   Mobile Weight    2000 non-null   int64  
 9   Processors       2000 non-null   int64  
 10  Primary Camera   2000 non-null   int64  
 11  Pixel Height     2000 non-null   int64  
 12  Pixel Width      2000 non-null   int64  
 13  RAM              2000 non-null   int64  
 14  Screen Height    2000 non-null   int64  
 15  Screen Width     2000 non-null   int64  
 16  Talk Time        2000 non-null   int64  
 17  3G               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: 343.8 KB
```

In [8]:

```
main_dataset.describe()
```

Out[8]:

	Battery	Bluetooth	Clock Speed	Dual Sim	Front Camera	4G	Interna Memor
count	2000.000000	2000.0000	2000.000000	2000.000000	2000.000000	2000.000000	2000.00000(
mean	1238.518500	0.4950	1.522250	0.509500	4.309500	0.521500	32.04650(
std	439.418206	0.5001	0.816004	0.500035	4.341444	0.499662	18.14571!
min	501.000000	0.0000	0.500000	0.000000	0.000000	0.000000	2.00000(
25%	851.750000	0.0000	0.700000	0.000000	1.000000	0.000000	16.00000(
50%	1226.000000	0.0000	1.500000	1.000000	3.000000	1.000000	32.00000(
75%	1615.250000	1.0000	2.200000	1.000000	7.000000	1.000000	48.00000(
max	1998.000000	1.0000	3.000000	1.000000	19.000000	1.000000	64.00000(

8 rows × 21 columns

Preprocessing the main_dataset

In [9]:

```
#Return DataFrame with duplicate rows removed
main_dataset = main_dataset.drop_duplicates()
```

In [10]:

```
#Remove missing values.
main_dataset = main_dataset.dropna()
main_dataset.shape
```

Out[10]:

```
(2000, 21)
```

In [11]:

```
#Return number of unique elements in the object
main_dataset.nunique()
main_dataset.shape
```

Out[11]:

```
(2000, 21)
```

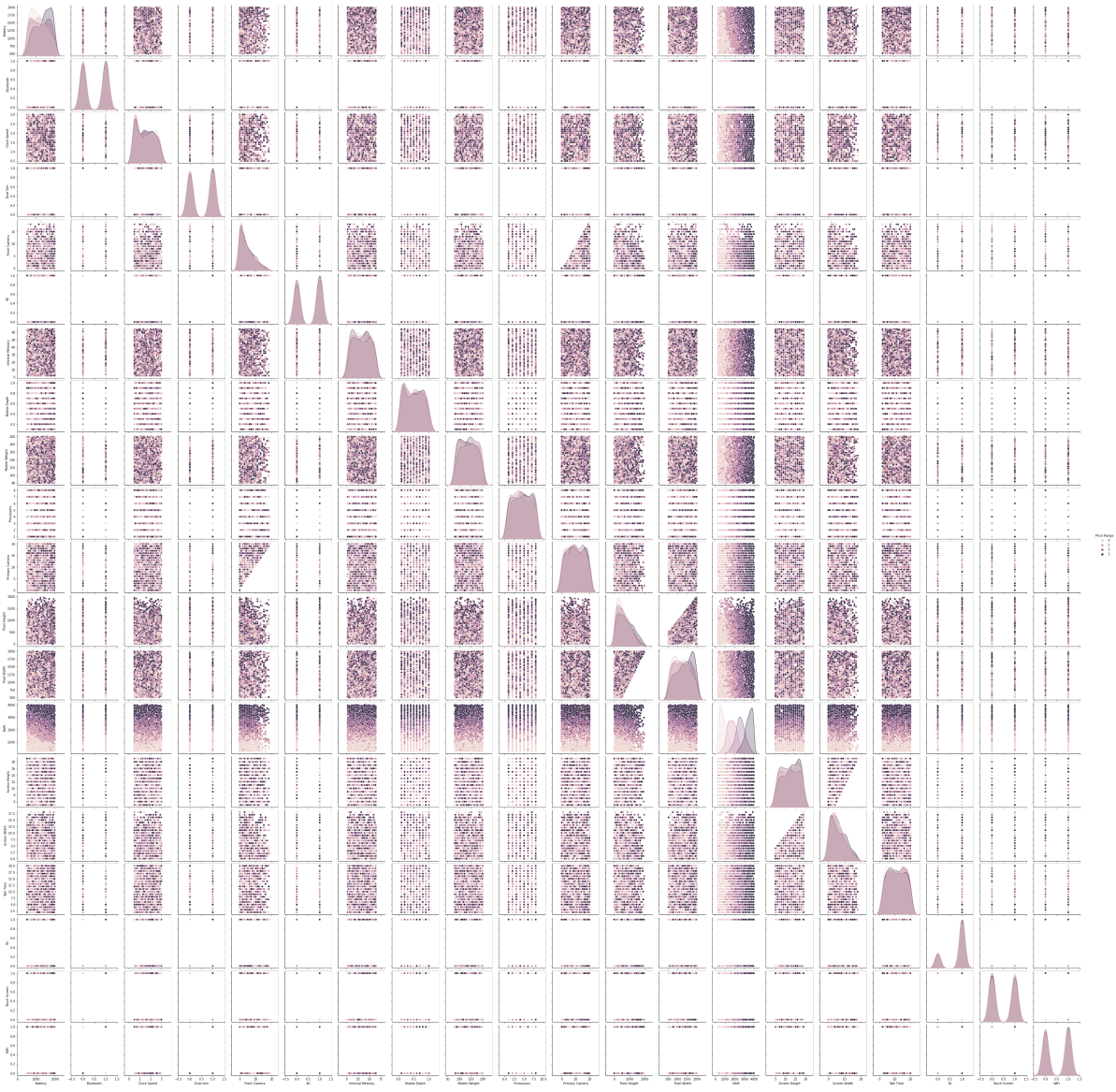
Data Visualisation & Analysis

In [12]:

```
sns.pairplot(main_dataset,hue='Price Range')
```

Out[12]:

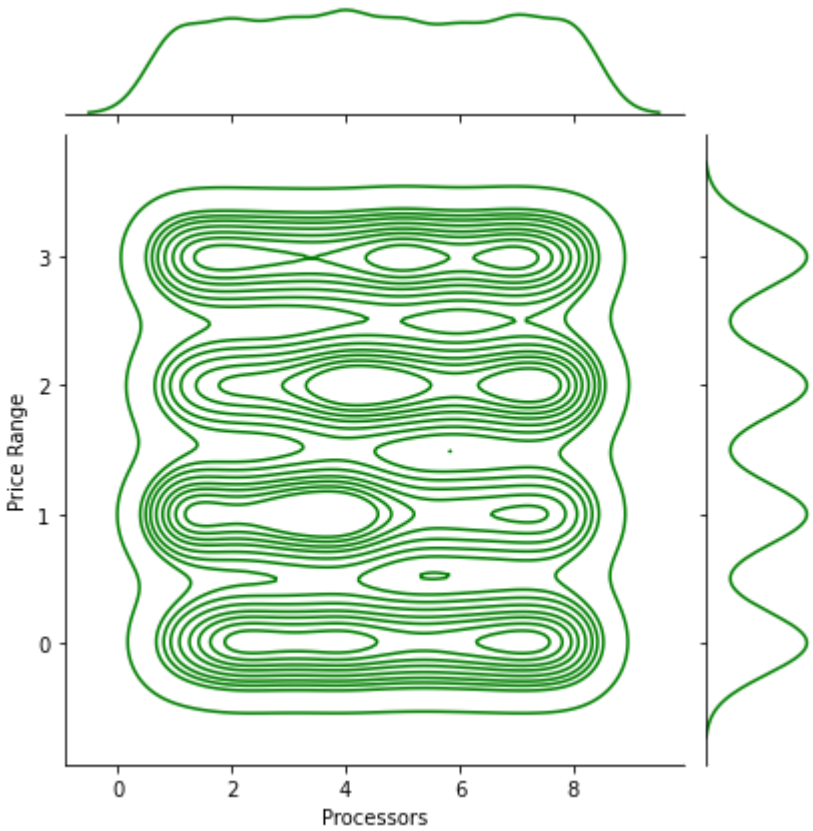
```
<seaborn.axisgrid.PairGrid at 0x7f94bac471f0>
```



How Processor got affected by price

In [13]:

```
sns.jointplot(x='Processors',y='Price Range',data=main_dataset,color='green',kind='k
```



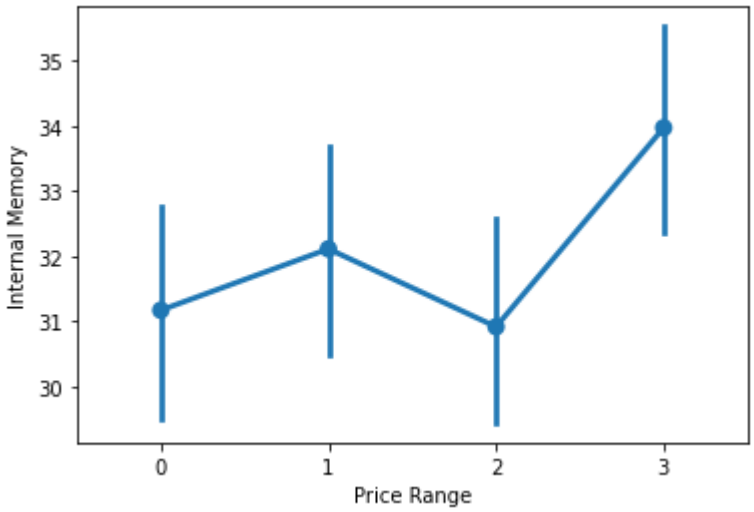
Internal Memory vs Price Range

In [14]:

```
sns.pointplot(y="Internal Memory", x="Price Range", data=main_dataset)
```

Out[14]:

```
<AxesSubplot:xlabel='Price Range', ylabel='Internal Memory'>
```



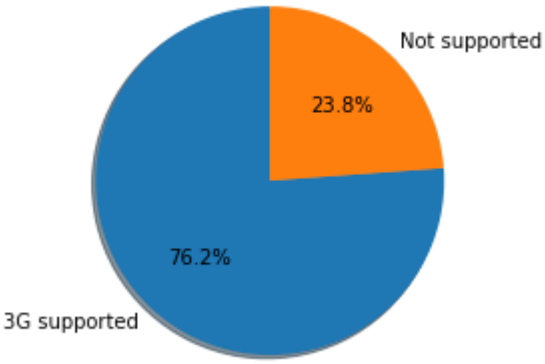
What % of Phones support 3G

In [15]:

```
labels = ["3G supported",'Not supported']
values=main_dataset['3G'].value_counts().values
```

In [16]:

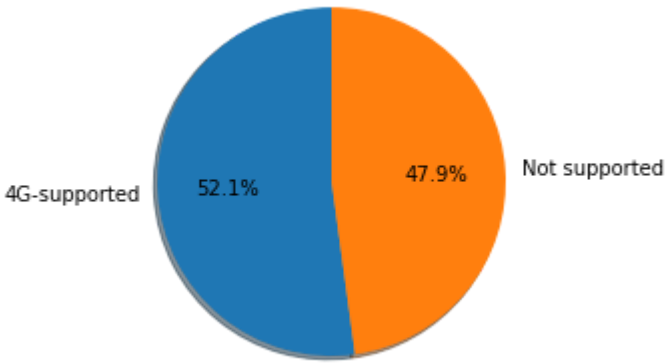
```
fig1, ax1 = plt.subplots()
ax1.pie(values, labels=labels, autopct='%1.1f%%',shadow=True,startangle=90)
plt.show()
```



What % of Phones support 4G

In [17]:

```
labels4g = ["4G-supported",'Not supported']
values4g = main_dataset['4G'].value_counts().values
fig1, ax1 = plt.subplots()
ax1.pie(values4g, labels=labels4g, autopct='%1.1f%%',shadow=True,startangle=90)
plt.show()
```



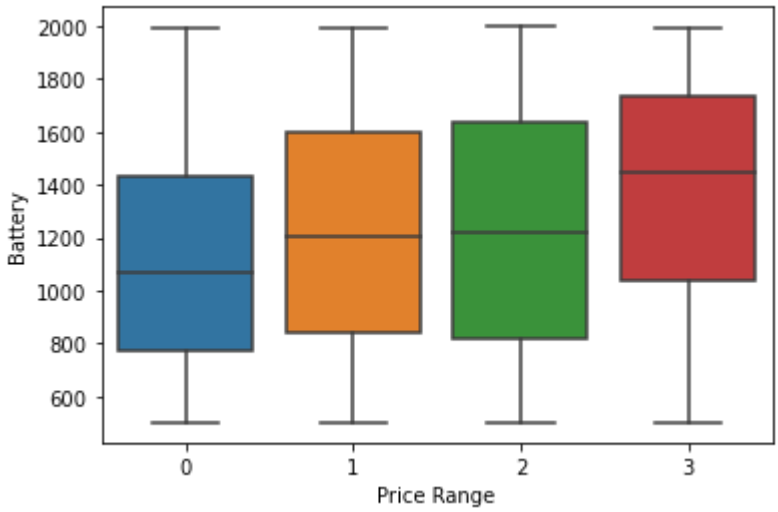
Battery Power vs Price Range

In [18]:

```
sns.boxplot(x="Price Range", y="Battery", data=main_dataset)
```

Out[18]:

```
<AxesSubplot:xlabel='Price Range', ylabel='Battery'>
```



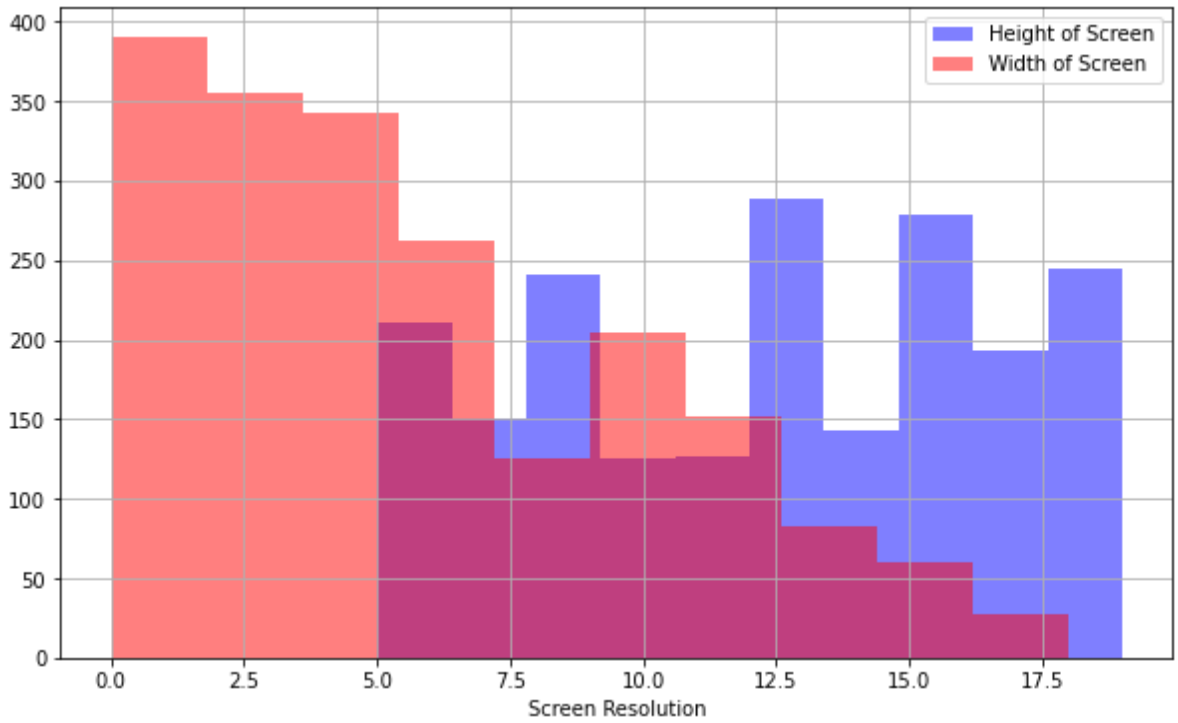
No. of Phones vs Screen Resolution (Width and Height)

In [19]:

```
plt.figure(figsize=(10,6))
main_dataset['Screen Height'].hist(alpha=0.5,color='blue',label='Height of Screen')
main_dataset['Screen Width'].hist(alpha=0.5,color='red',label='Width of Screen')
plt.legend()
plt.xlabel('Screen Resolution')
```

Out[19]:

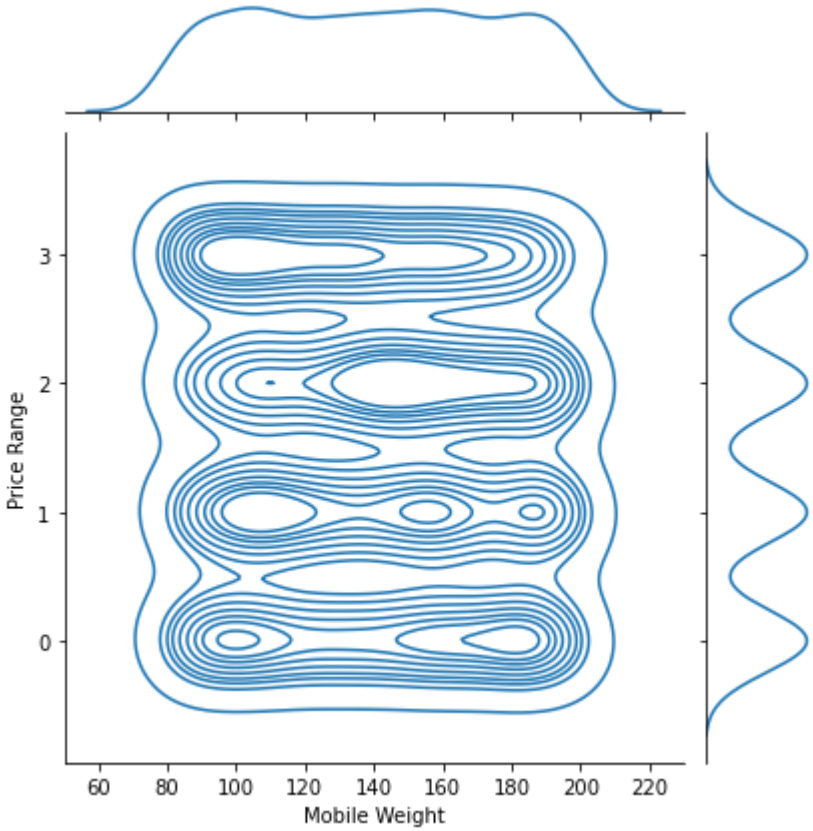
```
Text(0.5, 0, 'Screen Resolution')
```



Weight of mobile vs Price range

In [20]:

```
sns.jointplot(x='Mobile Weight',y='Price Range',data=main_dataset,kind='kde');
```



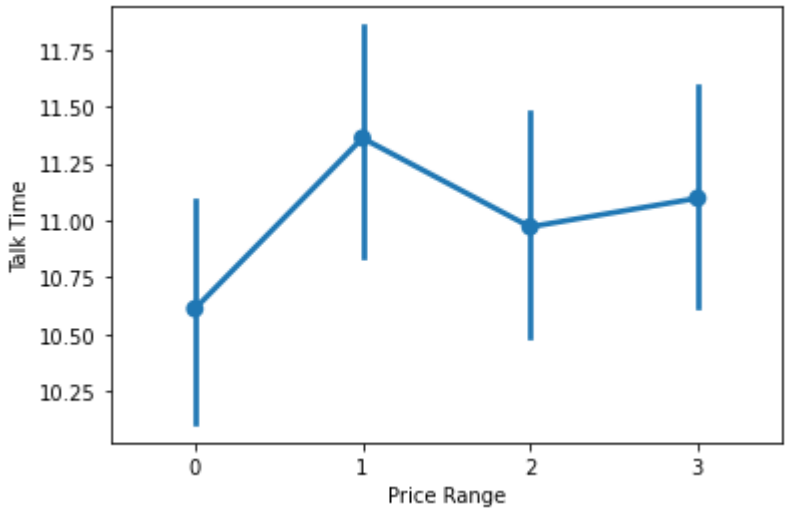
Talk time vs Price range

In [21]:

```
sns.pointplot(y="Talk Time", x="Price Range", data=main_dataset)
```

Out[21]:

```
<AxesSubplot:xlabel='Price Range', ylabel='Talk Time'>
```



X and y array

In [22]:

```
X = main_dataset.drop('Price Range',axis=1)
```

In [23]:

```
y = main_dataset['Price Range']
```

Splitting the dataset

In [24]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_sta
```

Training a Linear Regression Model

In [25]:

```
lm = LinearRegression()
```

In [26]:

```
lm.fit(X_train,y_train)
```

Out[26]:

```
LinearRegression()
```

In [27]:

```
lm.score(X_test,y_test)*100
```

Out[27]:

```
91.05323761864229
```

Training KNN Model

In [28]:

```
knn = KNeighborsClassifier(n_neighbors=10)
knn.fit(X_train,y_train)
```

Out[28]:

```
KNeighborsClassifier(n_neighbors=10)
```

In [29]:

```
knn.score(X_test,y_test)*100
```

Out[29]:

```
92.5
```

Elbow Method For optimum value of K

In [30]:

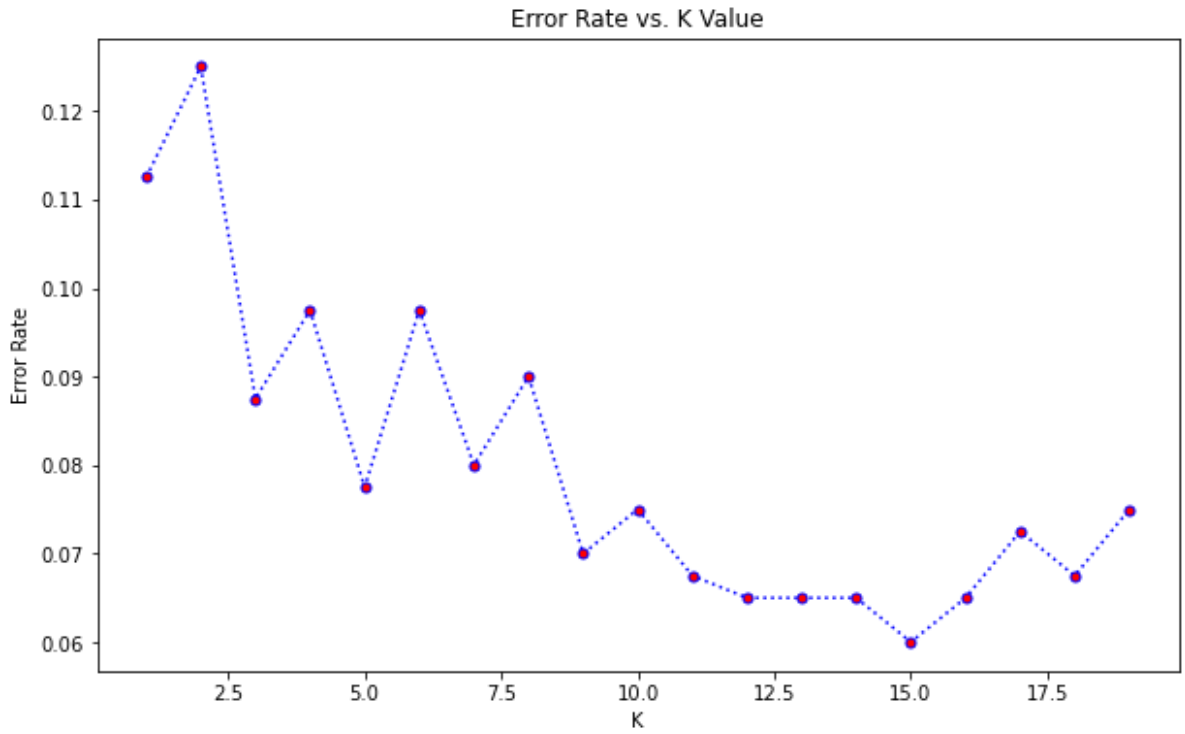
```
error_rate = []
for i in range(1,20):

    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(X_train,y_train)
    pred_i = knn.predict(X_test)
    error_rate.append(np.mean(pred_i != y_test))
```

In [31]:

```
plt.figure(figsize=(10,6))
plt.plot(range(1,20),error_rate,color='blue', linestyle='dotted', marker='o',
         markerfacecolor='red', markersize=5)
plt.title('Error Rate vs. K Value')
plt.xlabel('K')
plt.ylabel('Error Rate')
```

Out[31]:

```
Text(0, 0.5, 'Error Rate')
```



Training Logistic Regression Model

In [32]:

```
logmodel = LogisticRegression()
```

In [33]:

```
logmodel.fit(X_train,y_train)
```

```
/Users/uttkarshraj/opt/anaconda3/lib/python3.9/site-packages/sklearn/l
inear_model/_logistic.py:763: ConvergenceWarning: lbfgs failed to conv
erge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as show
n in:
    https://scikit-learn.org/stable/modules/preprocessing.html (http
s://scikit-learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic
-regression (https://scikit-learn.org/stable/modules/linear_model.html
#logistic-regression)
  n_iter_i = _check_optimize_result(
```

Out[33]:

```
LogisticRegression()
```

In [34]:

```
logmodel.score(X_test,y_test)*100
```

Out[34]:

```
68.0
```

Conclusion: Out of 3 techniques (KNN & Linear Regression) performed the best

Result : Linear Regression

In [35]:

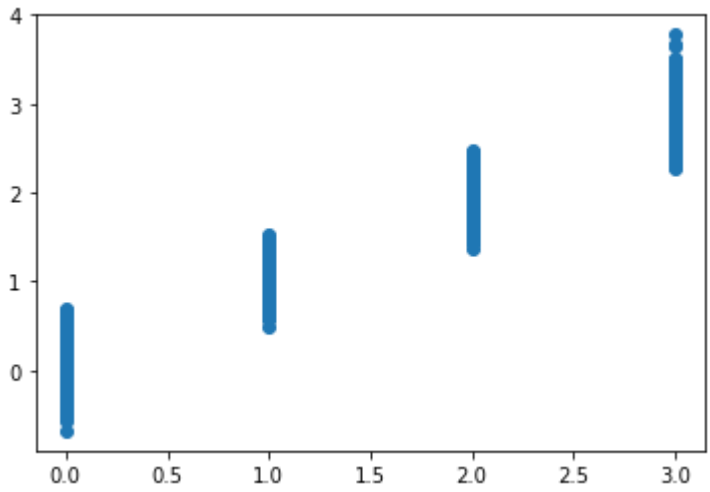
```
y_pred=lm.predict(X_test)
```

In [36]:

```
plt.scatter(y_test,y_pred)
```

Out[36]:

```
<matplotlib.collections.PathCollection at 0x7f94893ffac0>
```

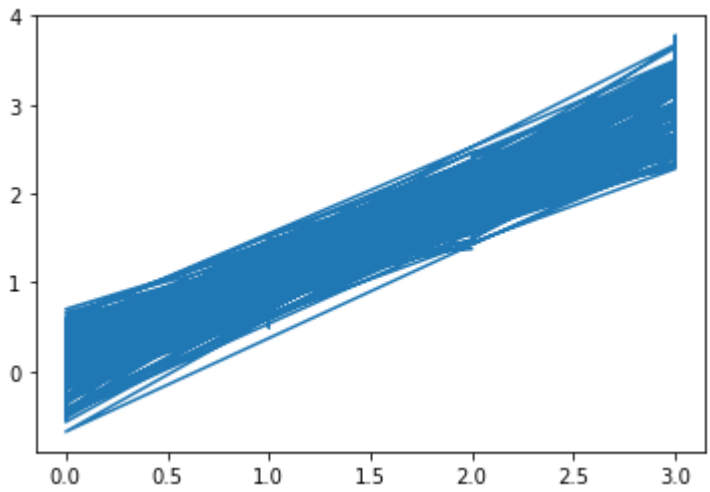


In [37]:

```
plt.plot(y_test,y_pred)
```

Out[37]:

```
[<matplotlib.lines.Line2D at 0x7f9499715310>]
```



Result : KNN

In [38]:

```
pred = knn.predict(X_test)
```

In [39]:

```
print(classification_report(y_test,pred))
```

```
              precision    recall  f1-score   support

           0       0.96      0.97      0.96        88
           1       0.88      0.93      0.91        98
           2       0.92      0.88      0.90       114
           3       0.95      0.94      0.94       100

    accuracy                           0.93       400
   macro avg       0.93      0.93      0.93       400
weighted avg       0.93      0.93      0.92       400
```

In [40]:

```
matrix=confusion_matrix(y_test,pred)
print(matrix)
```

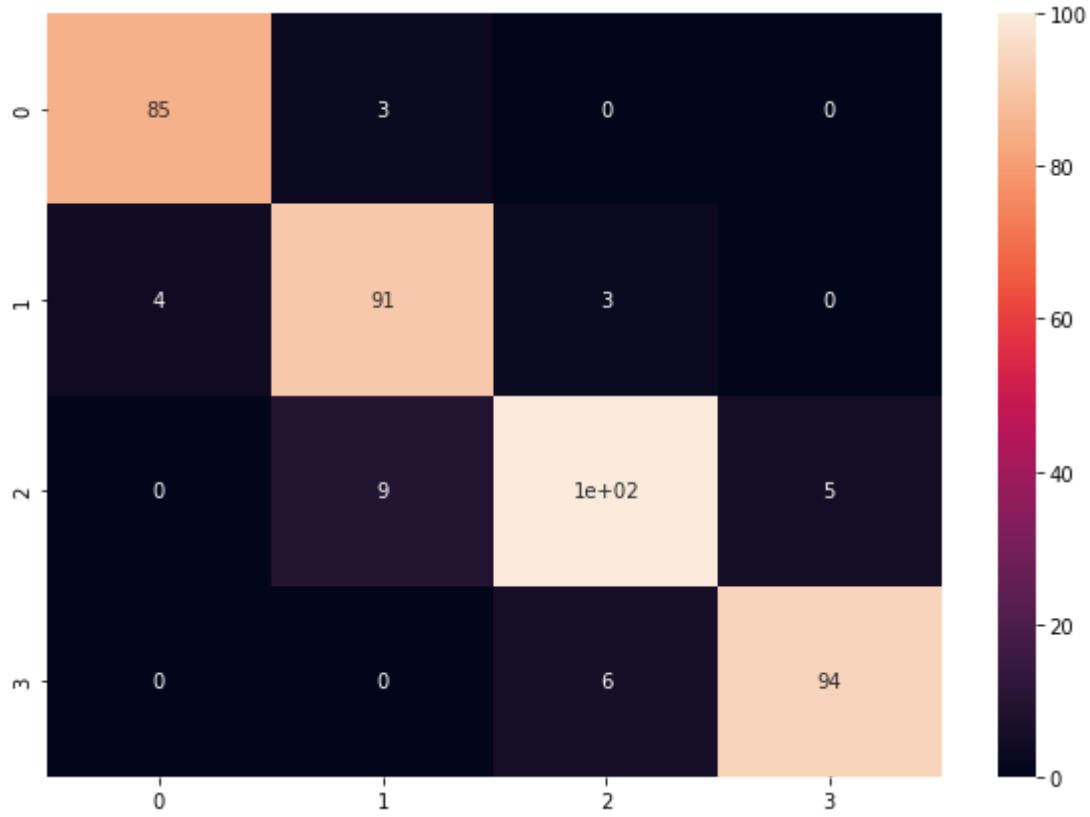
```
[[ 85   3   0   0]
 [  4  91   3   0]
 [  0   9 100   5]
 [  0   0   6  94]]
```

In [41]:

```
plt.figure(figsize = (10,7))
sns.heatmap(matrix,annot=True)
```

Out[41]:

```
<AxesSubplot:>
```



In [42]:

```
main_dataset.head()
```

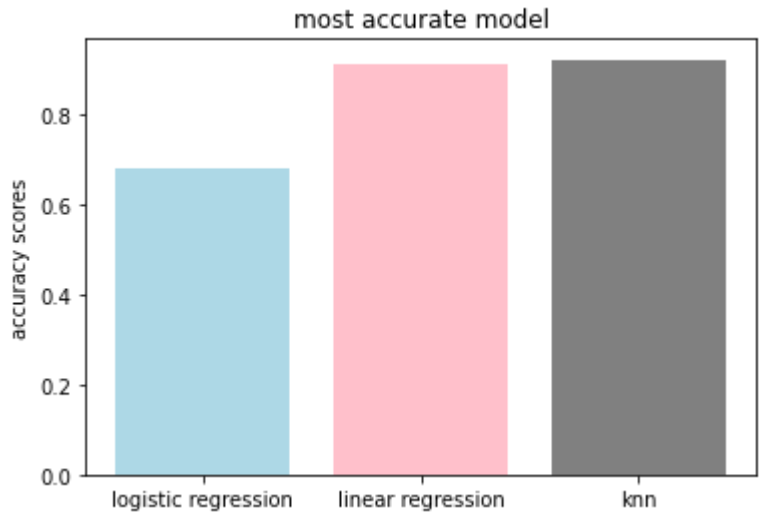
Out[42]:

	Battery	Bluetooth	Clock Speed	Dual Sim	Front Camera	4G	Internal Memory	Mobile Depth	Mobile Weight	Processors	...	Pi Hei
0	842	0	2.2	0	1	0	7	0.6	188	2	...	
1	1021	1	0.5	1	0	1	53	0.7	136	3	...	9
2	563	1	0.5	1	2	1	41	0.9	145	5	...	12
3	615	1	2.5	0	0	0	10	0.8	131	6	...	12
4	1821	1	1.2	0	13	1	44	0.6	141	2	...	12

5 rows × 21 columns

In [43]:

```
models = ['logistic regression', 'linear regression', 'knn']
acc_scores = [0.68, 0.91, 0.92]

plt.bar(models, acc_scores, color=['lightblue', 'pink', 'grey', ])
plt.ylabel("accuracy scores")
plt.title("most accurate model")
plt.show()
```



In [ ]:

In [ ]: