MOBILE PRICE PREDICTION USING PYTHON

STUDENT NAME: Aditya Vitthal Papal

PROJECT NAME: Mobile Price Prediction using Machine Learning

(Minor Project)

DATE: 01/03/2023

INDEX

CONTENT	PAGE NO

ABSTRACT:

Mobile phones are an integral part of our lives today, and with the ever-increasing number of models and brands available in the market, it can be challenging to decide which one to buy. The mobile phone market is one of the most dynamic and competitive industries in the world, with an ever-increasing number of new models and features being released every year. In order to stay competitive, it is essential for mobile phone manufacturers to accurately predict the prices of their products. To aid in this decision-making process, this study proposes a mobile price prediction classification model using Python.

This project aims to develop a mobile price prediction classification model using Python. The proposed model will use a dataset of mobile phone features, such as RAM, battery capacity, and camera quality, to predict the price range of the mobile phone.

The dataset will be pre-processed, cleaned, and transformed to remove any missing or inconsistent data. The model will be trained and tested using a random split of the dataset, with evaluation metrics such as accuracy, precision, and recall used to measure the performance of the model.

INTRODUCTION:

In recent years, the mobile phone industry has become highly competitive, with a large number of models being released every year. This has led to a wide range of prices, from budget models to high-end flagship phones. It has become crucial for manufacturers and retailers to predict the prices of mobile phones accurately to stay competitive in the market.

Mobile price prediction is a classification problem that can be tackled using machine learning techniques in Python. In this project, we aim to build a predictive model that can classify mobile phones into different price ranges based on their features.

To accomplish this, we will use a dataset that contains information about various mobile phone models and their respective prices. This dataset may include features such as battery life, camera quality, screen size, and other technical specifications.

Our goal is to pre-process the data and extract meaningful features that are relevant for the classification task. We will then train a machine learning model using various classification algorithms, such as Decision Trees, Random Forests, or Support Vector Machines, to predict the price range of mobile phones.

Finally, we will evaluate the performance of our model by testing it on a holdout dataset, using metrics such as accuracy, precision, and recall. This model can be useful for consumers, manufacturers, and retailers in predicting the prices of mobile phones and making informed decisions about pricing strategies, product development, and marketing campaigns.

OBJECTIVES:

The main objective of building a mobile price prediction classification model using Python is to predict the price range of mobile phones based on their features. This can help manufacturers, retailers, and consumers in various ways, such as:

- Manufacturers can use this model to predict the prices of their upcoming mobile phone models based on their features. This can help them set competitive prices and make informed decisions about pricing strategies.
- Retailers can use this model to predict the prices of mobile phones from different brands and models. This can help them decide which products to stock and at what prices.
- Consumers can use this model to compare the prices of different mobile phone models and choose the one that best fits their budget and needs.

Overall, the main objective of this model is to help stakeholders in the mobile phone industry make informed decisions about pricing, product development, and marketing strategies.

SCOPE:

The scope of a mobile price prediction classification model using Python is vast, and it can be applied in various industries and use cases. Here are some of the scopes of this model:

1. **E-commerce:** E-commerce websites can use this model to predict the prices of mobile phones and display them to customers. This can help customers compare prices across different websites and make informed purchasing decisions.
2. **Manufacturers:** Mobile phone manufacturers can use this model to predict the prices of their upcoming models based on their features and specifications. This can help them set competitive prices and make informed decisions about pricing strategies.
3. **Retailers:** Retailers can use this model to predict the prices of mobile phones from different brands and models. This can help them decide which products to stock and at what prices.
4. **Consumers:** Consumers can use this model to compare the prices of different mobile phone models and choose the one that best fits their budget and needs.
5. **Market research:** Market research companies can use this model to analyse the prices of mobile phones across different brands and models. This can help them identify trends in the market and make predictions about future prices.

Overall, the scope of a mobile price prediction classification model using Python is wide and can be applied in various industries and use cases. This model can help stakeholders in the mobile phone industry make informed decisions about pricing, product development, and marketing strategies.

METHODOLOGY:

The methodology for building a mobile price prediction classification model using Python involves the following steps:

Data collection: The first step is to collect the data that contains information about various mobile phone models and their prices. This data should include features such as brand, model, screen size, battery life, camera quality, storage capacity, RAM, operating system, and other technical specifications.

Data pre-processing: The next step is to preprocess the data by handling missing values, encoding categorical features, and scaling numerical features. This is an essential step as it helps in preparing the data for the machine learning algorithms.

Feature engineering: Feature engineering involves selecting the most relevant features and transforming them into a format that can be used by machine learning algorithms. This step helps in improving the accuracy of the model.

Model selection: The next step is to select a suitable machine learning algorithm for the classification task. Some of the popular algorithms include Decision Trees, Random Forests, Support Vector Machines, or Gradient Boosting.

Model training: Once the algorithm is selected, the next step is to train the model on the pre-processed data. This involves splitting the data into training and testing datasets and using the training dataset to teach the model to predict the price range of mobile phones based on their features.

Model evaluation: After training the model, we need to evaluate its performance using various metrics such as accuracy, precision, and recall. This step helps in determining the effectiveness of the model in predicting the price range of mobile phones.

Model deployment: The final step is to deploy the model in production. This involves integrating the model with an application or website that can be used by stakeholders in the mobile phone industry to make informed decisions about pricing, product development, and marketing strategies.

Overall, the methodology for building a mobile price prediction classification model using Python involves data collection, pre-processing, feature engineering, model selection, model training, model evaluation, and model deployment.

TYPES OF MODELS:

Logistic Regression:

Logistic Regression is a popular machine learning algorithm used for binary classification tasks, where the output variable takes only two possible values (e.g., 0 or 1, true or false, yes or no). It models the probability of the output variable taking a particular value, given the input features.

In Logistic Regression, the input features are combined linearly using weights, and the resulting value is passed through a logistic function that converts the output to a value between 0 and 1. This value represents the probability of the output variable taking a particular value.

KNN:

K-Nearest Neighbours (KNN) is a popular machine learning algorithm used for classification tasks. It is a non-parametric algorithm, meaning it does not make any assumptions about the underlying distribution of the data. Instead, it simply uses the input features to find the K nearest neighbours of a new data point and assigns it to the most common class among those neighbours.

The value of K is a hyperparameter that needs to be tuned based on the problem and the dataset. A smaller value of K may result in overfitting, while a larger value of K may result in underfitting. In general, it is a good practice to try different values of K and choose the one that gives the best performance on a validation set.

SVM Classifier with linear and rbf kernel:

Support Vector Machine (SVM) is a popular machine learning algorithm used for classification tasks. It is a powerful algorithm that

can handle both linearly separable and non-linearly separable datasets by using different types of kernel functions.

There are several types of kernel functions available in SVM, including linear, polynomial, and radial basis function (RBF) kernels. The linear kernel is used for linearly separable datasets and is computationally efficient. The RBF kernel is the most commonly used kernel in SVM and can handle non-linearly separable datasets by mapping the input features into an infinite-dimensional space.

Decision Tree Classifier:

Decision Tree Classifier is a popular machine learning algorithm used for classification tasks. It is a non-parametric algorithm that builds a tree-like model of decisions and their possible consequences based on the input features.

The basic idea behind decision tree classification is to partition the input space into smaller regions based on the values of the input features. At each node of the tree, the algorithm selects the input feature that best splits the data into the purest possible subsets, where each subset contains examples of the same class. This splitting process is repeated recursively until all the data points in a leaf node belong to the same class, or until a stopping criterion is met.

Random Forest Classifier:

Random Forest Classifier is an ensemble machine learning algorithm that combines multiple decision trees to improve the accuracy and robustness of the model. It is a popular algorithm for classification tasks that can handle both binary and multi-class classification problems.

The basic idea behind random forest classification is to create a set of decision trees that are trained on different subsets of the data and different subsets of the input features. This randomness helps to reduce the variance of the model and avoid overfitting. During the training process, each decision tree is built by randomly selecting a subset of the data and a subset of the input features. The tree is then constructed using the same algorithm as the decision tree classifier.

The final prediction of the random forest classifier is obtained by combining the predictions of all the decision trees. For classification tasks, the majority vote is used to determine the final class label.

OUTPUT:

After using various machine learning algorithms and then training the dataset with these different algorithms, a final result was observed on which model was performing the best, i.e. which model gave the most accurate predictions for the mobile price prediction model.

The following table shows the observations in ascending order.

Different Models	Accuracy (in Percentage)
Decision tree Classification	85
Random Forest Regression	93
KNN Classification	93
SVM Classification with linear and RBL Kernel	95
Logistic Regression	95.5

CONCLUSION:

After testing many models and training and testing regressively, it was observed that logistic regression provided the most accurate results i.e. 95.5%, which was followed by SVM classification and its type with 95% accuracy. Others also gave predictions which were above 80% accurate.

Mobile Price Prediction

Data Cleaning

```
import pandas as pd
import matplotlib.pyplot as pit
import seaoorn as sns
import numpy as np
```

```
traln = pd.read_csv("I rain.csv")
```

```
test = pd.nead_csv("test.csv")
```

```
train.head()
```

	battery_power	blue	clock speed	dual sim	Ie	four_g	int_memory	m_dep	mobile wt	n cores
0	842	0	2.2	0	1	0	7	0.6	188	2
1	1021	1	0.S	1	0	1	53	0.7	136	3
2	563	1	OS	1	2	1	41	0.9	145	5
3	615	1	2.5	0	0	0	10	0.8	131	6
4	1821	1	1.2	0	13	1	44	0.6	141	2

5 rows x 21 columns

```
test.head()
```

	id	battery power	blue	clock speed	dual_sim	fc	four g	int memory	m dep	mobile wt ...	pc
0	1	1043	1	1.8	1	14	0	5	0.1	193	16
1	2	841	1	0.5	1	4	1	61	0.8	191	12
Z	3	1807	1	2.8	0	1	0	27	0.9	186	4
3	4	1546	0	0.5	1	18	1	25	0.5	96	20
4	5	1434	0	1.4	0	11	1	49	0.5	108	18

5 rows 21 columns

```
train.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
nam              0
sc_h             0
sc_w             0
talk_time        0
three_g          0
touch screen     0
wifi             0
price_range      0
dtype: int64
```

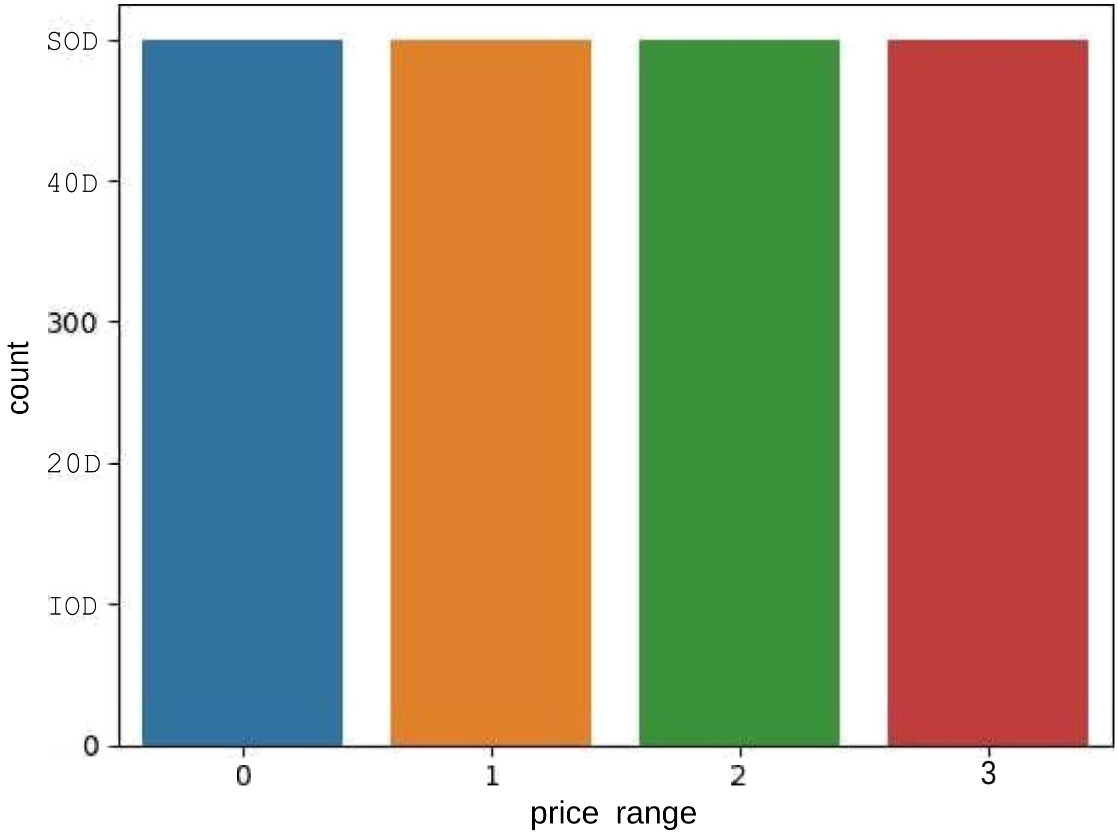
```
test.isnull().sum()
```

```
id               0
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
nam              0
sc_h             0
sc_w             0
talk_time        0
three g          0
touch_screen     0
wiki             0
dtype: int64
```

```
te st .drop('1d  , ax1s=1, 1nplace=True)
```

```
sns.countplot(train["price range"])
```

```
D:\Software_Installation\Anaconda\lib\site-packages\seaborn\_decorators.py:36:      FutureWar
ning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid
positional argument will be data , and passing other arguments without an explicit keys
ord will result in an error or misinterpretation.
  warnings.warn(
<AxesSubplot:xlabe1='price_range', yIabeI='count'>
```




```
trair.shape
```

```
(2000, 21)
```

```
te st . shape
```

```
(1000, 20)
```

```
train.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
 Z   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 17]

```
test.1nfo()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 20 columns):
 #   Column         Non-Null Count  Dtype

 0   battery_power  1000 non-null   int64
 1   blue           1000 non-null   int64
 2   clock_speed    1000 non-null   float64
 3   dual_sim       1000 non-null   int64
 4   fc             1000 non-null   int64
 5   four_g         1000 non-null   int64
 6   int_memory     1000 non-null   int64
 7   m_dep          1000 non-null   float64
 8   mobile_wt      1000 non-null   int64
 9   n_cores        1000 non-null   int64
 10  pc             1000 non-null   int64
 11  px_height      1000 non-null   int64
 12  px_width       1000 non-null   int64
 13  ram            1000 non-null   int64
 14  sc_h           1000 non-null   int64
 15  sc_w           1000 non-null   int64
 16  talk_time      1000 non-null   int64
 17  three_g        1000 non-null   int64
 18  touch_screen  1000  non-null int64
                         n  null  int64
d  9pe  *oat64(Z  ,  10
memony usage: 156.4 KB
```

In [19]

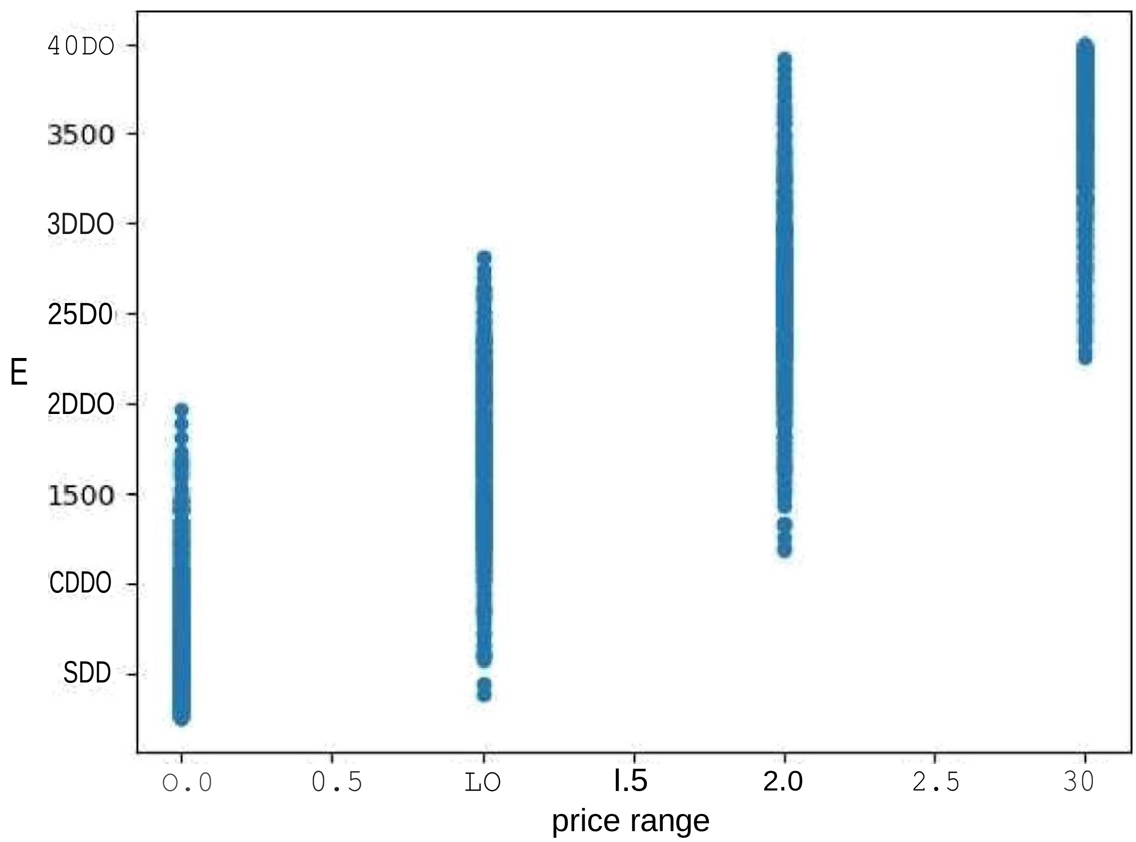
```
train.describe()
```

Out [19] :

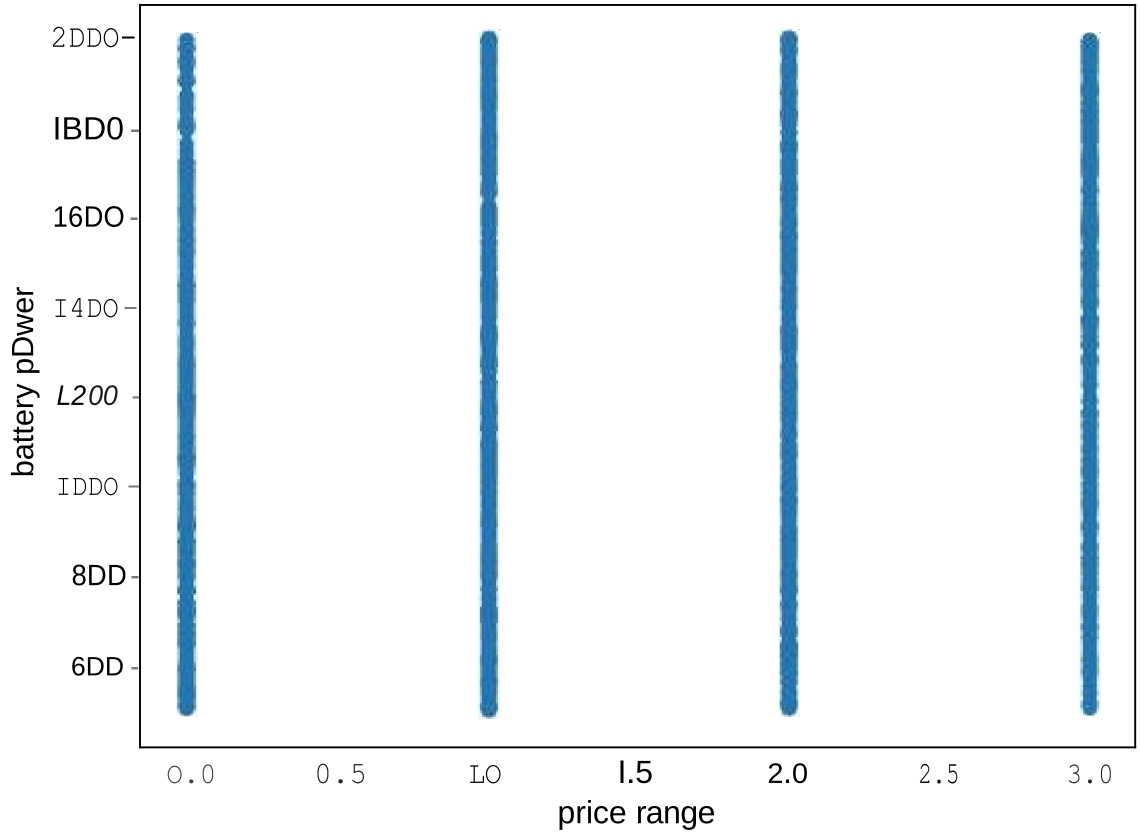
	battery power	blue	clock speed	dual sim	fc	four g	int memory	
count	2000.000000	2000.0000	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000	2000
mean	1238.518500	0.4950	1.522250	0.509500	4.309500	0.521500	32046500	0
std	439.418206	0.5001	0.816004	0.500035	4.341444	0.499662	18.145715	0
min	501.000000	0.0000	0.500000	0.000000	0.000000	0.000000	2.000000	0
25%	851.750000	0.0000	0.700000	0.000000	1.000000	0.000000	16.000000	0
50°#	1226.000000	0.0000	1.500000	70O00OO	3.000000	1.000000	32.000000	0
75%	1615.250000	1.0000	2.200000	1.000000	7.000000	1.000000	48.000000	0
max	1998.000000	1.0000	3.000000	1.000000	19.000000	1.000000	64.000000	1

8 rows 21 columns

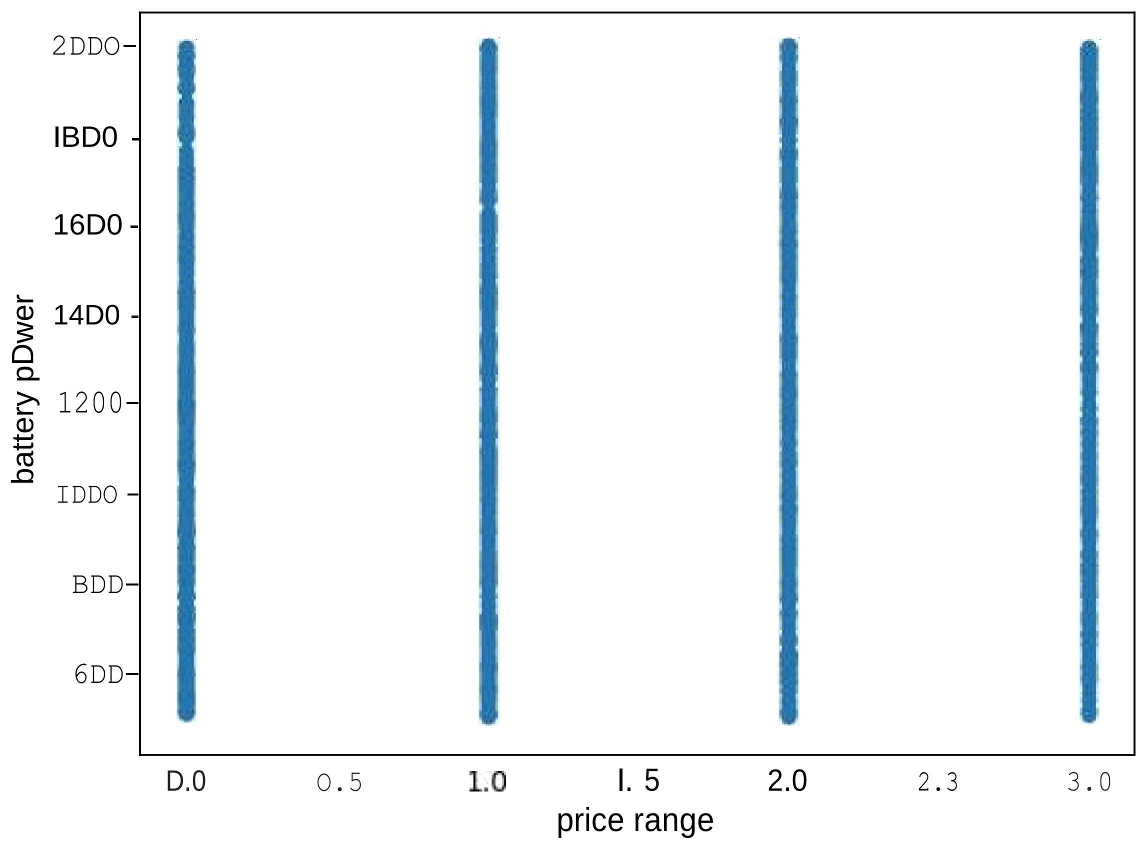
```
train.plot(x='price_range',y='ram',kind='scatter')
plt.show()
```



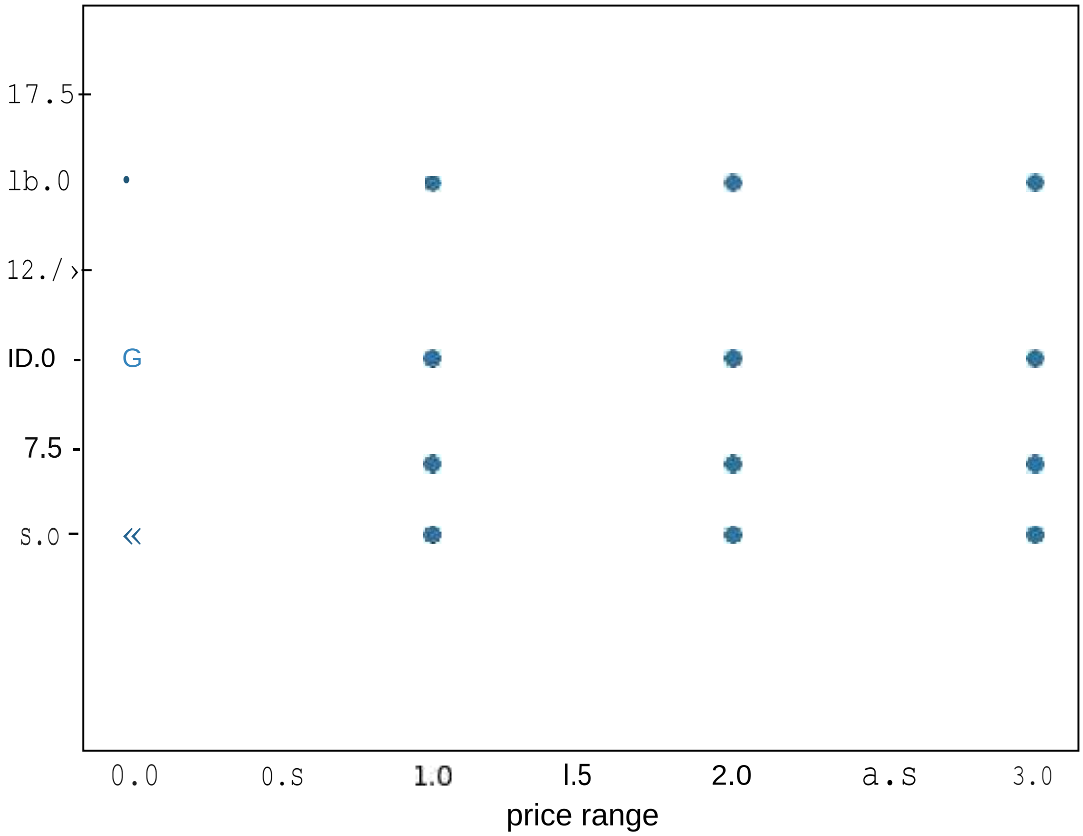

```
train.plot(x='price_range',y='battery_power',kind='scatter')
plt.show()
```



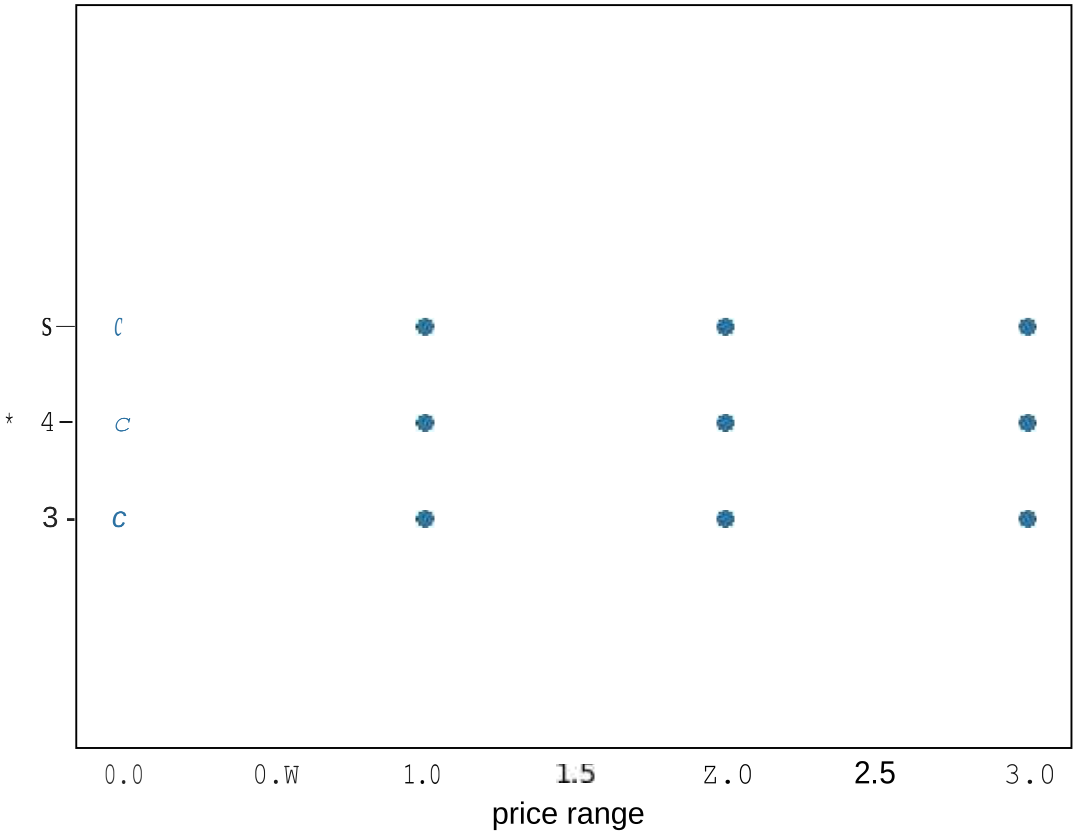

```
train.plot(x-'price_range',y='battery_power',kind='scatter')
plt,show()
```



```
train.plot(x='price_range',y:'fc',kind='scatte ')
plt.show()
```




```
train.plot(x='price_range',y='n_cores',kind-'scatter')
plt.show()
```



In [25]:
```
train.corr()
```

Out [25] :

	battery_power	blue	clock speed	dual_sim	le	four g	int memory	n
battery_power	1.000000	0.011252	0.011482	-0.041847	0.033334	0.015665	-0.004004	0.0.
blue	0.011252	1.000000	0.021419	0.035198	0.003593	0.013443	0.041177	0.0T
clock speed	0.011482	0.021419	1.000000	-0.001315	-0.000434	-0.043073	0.006545	-0.0'
dual_sim	-0.041847	0.035198	-0.001315	1.000000	-0.029123	0.003187	-0.015679	-0.0:
le	0.033334	0.003593	-0.000434	-0.029123	LOOOOOO	-0046560	-0.029133	-0.01
four_g	0.015665	0.013443	-0.043073	0.003187	-0.016560	1.000000	0.008690	-0.0T
int_memory	-0.004004	0.041177	0.006545	-0.015679	-0.029133	0.008690	1.000000	0.0T
m dep	0.034085	0.004049	-0.014364	-0.022142	-0.001791	-0.001823	0.006886	1.0
mobile_wt	0.001844	-0.008605	0.012350	-0.008979	0.023618	-0.016537	-0.034214	0.0:
n_cores	-0.029727	0.036161	-0.005724	-0.024658	-0.013356	-0.029706	-0.028310	-0.0T
pc	0.03 1441	-0.009952	-0.005245	-0.017143	0.644595	-0.005598	-0.033273	0.0:
px_height	0.014901	-0.006872	-0.014523	-0.020875	-0.009990	-0.019236	0.010441	0.0:
px_width	-0.008402	-0.041533	-0.009476	0.014291	-0.005176	0.007448	-0.008335	0.0:
ram	-0.000653	0.026351	0.003443	0.041072	0.015099	0.007313	0.032813	-0.0T
sc h	-0.029959	-0.002952	-0.029078	-0.011949	-0.011014	0.027166	0.037771	-0.0:
sc_w	-0.021421	0.000613	-0.007378	-0.016666	-0.012373	0.037005	0.011731	-0.0'

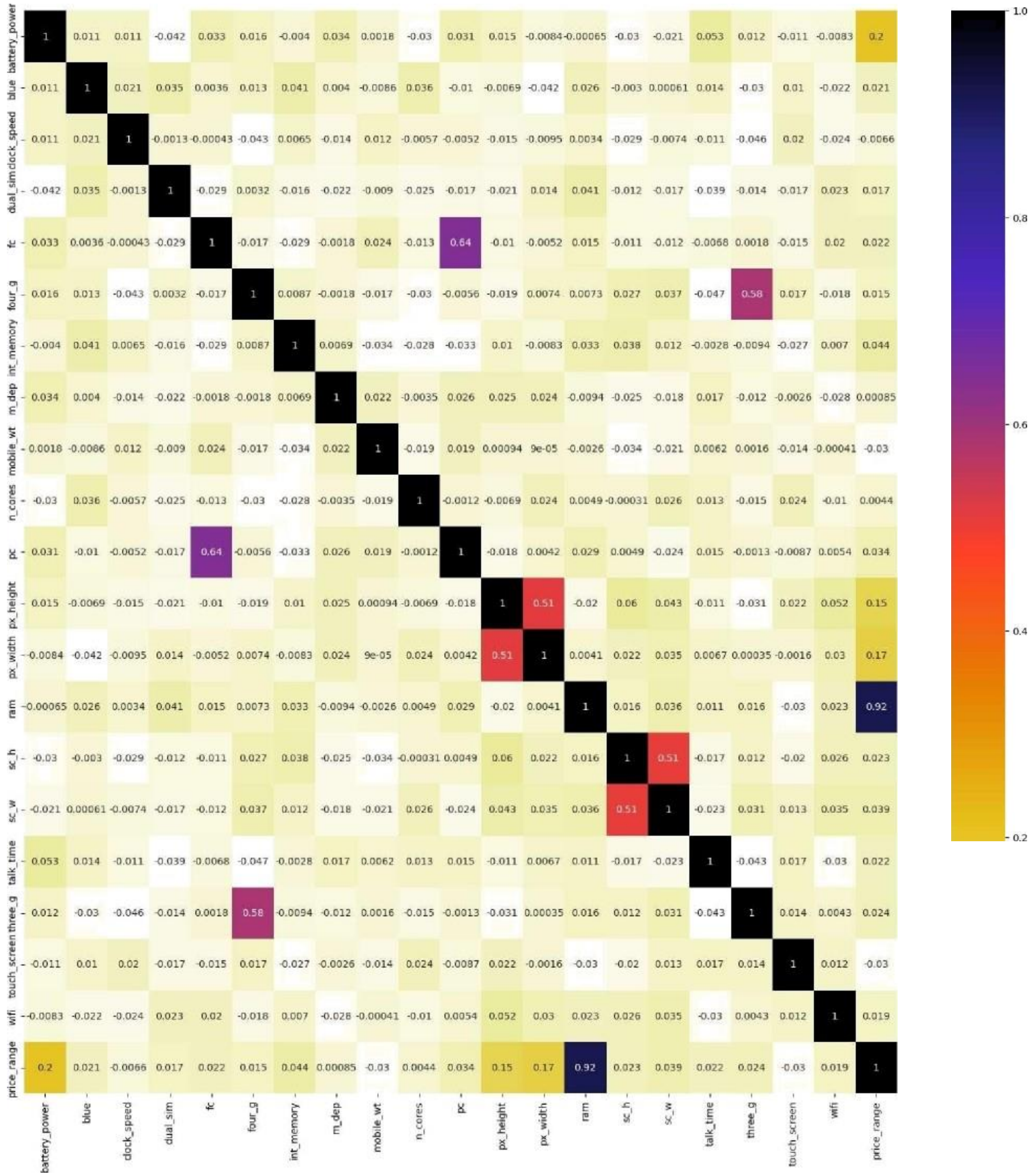
	battery_power	blue	clock speed	dual_sim	Ie	four g	int_memory	n
talk_time	0.052510	0.013934	-0.011432	-0.039404	-0.006829	-0.046628	-0.002790	0.0'
three_g	0.011522	-0.030236	-0.046433	-0.014008	0.001793	0.584246	-0.009366	-0.0'
touch screen	-0.010516	0.0l 0061	0.019756	-0.017117	-0.014828	0.016758	-0.026999	-0.0l
wifi	-0.008343	-0.021863	-0.024471	0.022740	0.020085	-0.017620	0.006993	-0.0.
price_range	0.200723	0.020573	-0.006606	0.017444	0.021998	0.014772	0.044435	0.0l

21 rows 21 columns

Freatures Engineering

```
import seaborn as sns
from matplotlib.pyplot import figure
#Using Peorson Corrcfotion
fig :    plt.figure(figsize:(20, 2g))
cor =  train.corr()
sns.heatmap(cor, annot=True, cmap=plt.cm.CMRmap_r)
```

‹AxesSubp 1ot : ›




```
de* correlation(dataset, threshold):
    col corr - set()  # Set of old the nomes o/ conrelotea columns
    corr matrix = dataset.corr()
    for i in range(len(corr matrix.columns)):
        for j in range(i):
            if abs(corr matrix.iloc[i, j]) › threshold: # we ore interested in odsolure
                colname = corr_matrix.columns[i]  A getring tne nome o/ column
                col_cor*.add(colname)
    return col corr
```

```
corr_+eatu es = correlation(train, 0.8)
len(set(corr_features))
```

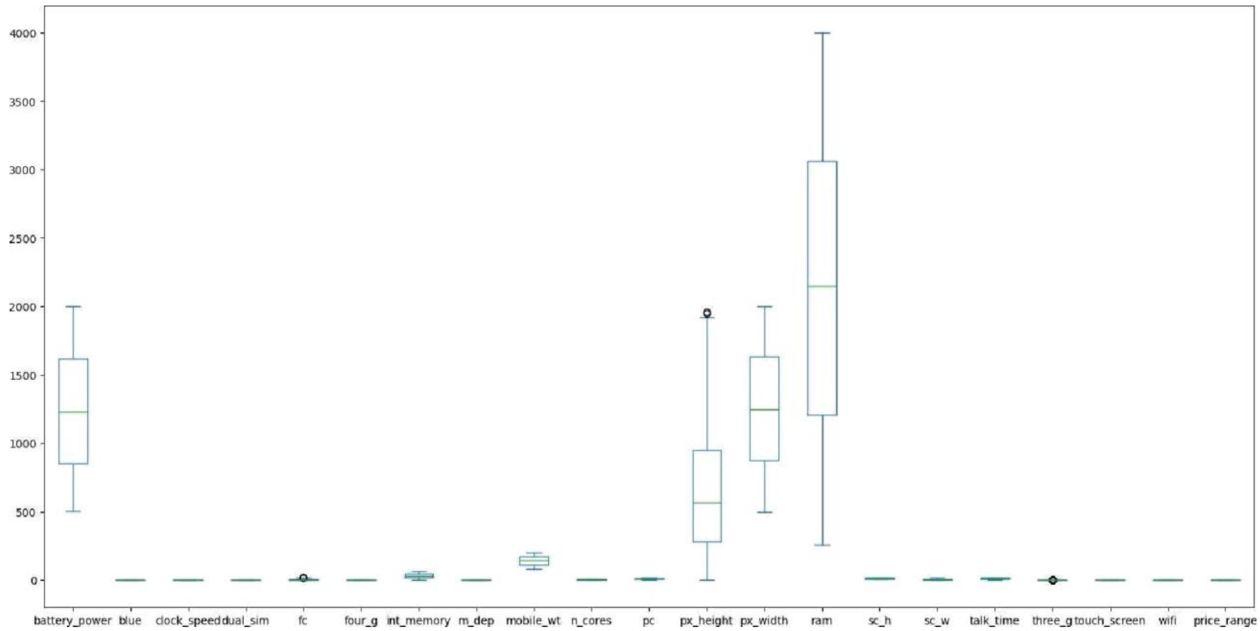
```
1
```

```
corr_features
```

```
{'price_range'}
```

```
train.plot(kind='box',figsize=(20,10))
```

```
<AxesSubplot:>
```




```
X = train.drop('price_rangw',axis=1)
y = train[
```

Trainning DataSet

```
*rom sklearn.model_selection import train_test_split
X_train, X_test, Y train, Y test = train test split(X,y,test_size=0.1,random state=101)
```

```
from sklearn.prep ocessing import StandardScale
sc = StandardScaler()
X train = sc.fit transform(X train)
X_test = sc.transform(X test)
test = sc.transform(test)
```

```
X_train
```

```
amay([[-1. 62737257,    -0. 98675438, -1. 01271559, . . . , -1.78222729,
        - 1.00892875,  - 0. 99888951] ,
       [ -0.75199354,  1.01342342,  0.58093235, ..., -1.78222729,
         0.99115027,  -0.9988 gsi],
```

```
       [-0.20630271,  1.01342342,  0.70352065, ...,  0.56109566,
        -1.00892875,  1.00111173],

       [ 0.69636086,  1.01342342, -0.03200917, ...,  0.56109566,
        -1.00892875, -0.99888951],
       [ 0.83733099, -0.98675438, -1.2578922 , ...,  0.561B9566,
        -1.00892875,  1.00111173],
       [ 0.4144206 , -0.98675438, -0.39977408, ...,  0.56109566,
         0.99115027,  1.00111173]])
```

In [40]:

```
X_test
```

Out[4];

```
array([[ 0.28481903, -0.98675438, -1.2578922 , ...,  0.56109566,
        -1.00892875, -0.99888951],
       [-1.44092821, -0.98675438, -1.2578922 , ...,  0.56109566,
         0.99115027,  1.00111173],
       [-1.49322358, -0.98675438, -0.15459747, ...,  0.561B9566,
        -1.00892875,  1.00111173],

       [-0.55418061,  1.01342342,  0.33575574, ...,  0.56109566,
        -1.00892875, -0.99888951],
       [ 0.09610095, -0.98675438, -0.89012729, ...,  0.56109566,
         0.99115027,  1.00111173],
       [-1.60690917, -0.sg67s438,  1.07128556, ...,  0.56109566,
         0.99115027, -0.99888951]])
```

In [4/]'

```
test
```

Out [4z]

```
array([[-0. 4541373 ,  1.01342342,  0.33575574, ..., -1. 78222729,
         0.99115027, -0.99888951],
       [-0.91342707,  1.01342342, -1.2578922, ...,  0.56109566,
        -1.00892875, -0.99888951],
       [ 1.2829785 ,  1.01342342,  1.56163877, ..., -1.78222729,
         0.99115027,  1.00111173],

       [-0.13127022, -0.sg67s438, -0.15459747, ...,  0.56109566,
        -1.00892875, -0.99888951],
       [ 0.65998148,  1.01342342, -1.2578922 , ..., -1.78222729,
         0.99115027, -0.99888951],
       [ 0.0619952g,  1.01342342, -1.2578922  ...,  0.56109566,
        -1.00892875,  1.00111173]])
```

Decision Tree Classification

In [43]'

```
from sklearn.tree import DecisionTreeClassifier
dtc = DecisionTreeClassifier()
dtc.fit(X_train , Y_train)
```

Out[43]:

```
DecisionTreeClassifier()
```

In [44]'

```
pred = dtc . p redict ( X_test)
pred
```

out[4s].

```
array([1, 1, 2, 1, 0, l, 2, 1, 1, 1, 0, 1, 1, 1, I, 0, 1, l, 1, 0, 3, 1,
       2, 3, 2, 2, 2, 1, 0, 0, 2, 3, 0, 0, 3, 0, 0, 0, 1, I, 1, 2, 3, 2,
       3, 1, 1, 3, 3, 1, 0, 0, 2, 3, 3, 2, 0, 3, 3, 3, 2, 2, 3, 1, 3, 2,
       0, 1, 0, 2, 1, 2, 3, 2, 2, 3, 3, 2, 0, 2, 0, 0, 2, 1, 2, 2, 2, 1,
       0, B, 3, 3, B, 2, 0, 3, 2, 0, 2, 3, 0, 2, 2, 3, 0, 2, 0, 0, 2, 0,
```

```
       I, 0, 3, 2, 2, 2, 1, 3, 2, 0, 3, 3, 1, 3, 1, 3, 3, 2, 1,         1, 1, 0,
       1, 1, 0, 2, 3, 0, 2, 3, 1, 3, 0, 1, 0, 0, 1, 3, 2, 0, 2, 1, 3, 2,
       3, 2, 2, 0, 3, 1  2, 2, 2, 2, 1, 2, 1  1, 3, 3, 1, 2, 0,   3, I, 3,
       1, 2, 3, 1 , 2, 1, 8, 1, 3, 2, 1, 2, 1  3, 1, 8, 2, 2, 0,  3, B, 0,
       3, 0], dtype=int64)
```

In [45].

```python
from sklearn.metrics import accuracy_score, confusion_matrix
dtc_acc = accuracy_score(pred,Y_test)
print(dtc_acc)
print(confusion_matrix(pred,Y_test))
```

```
0.855
[[43  3  0  0]
 [ 7 39  7  0]
 [ 0  4 49  2]
 [ 0  0  6 40]]
```

SVC Classification

In [46]:

```python
from sklearn.svm import SVC
knn=SVC()
knn.fit(X_train,Y_train)
```

Out [46] : wC ( )

In [47]

```python
pred1 = knn. predict (X_t est )
pred1
```

Out[47]:

```
array([1, 1, 2, 1, 1, 1, 2, 1, 1, 1, 0, 1, 1, 1, }, 0, 1, 1, 0, 0, 3, 1,
       2, 3, 2, 2, 2, 2, 0, 0, 2, 3, 0, 0, 3, 0, 0, 0, 1, 1, 1, 1, 3, 2,
       3, 0, 2, 3, 3, 1, 0, 1, 2, 3, 2, 2, 0, 3, 2, 3, 2, 2, 3, 1, 3, 1,
       0, I, 0, 2, 2, 2, 3, 2, 1, 3, 3, 2, 1, 2, 0, 0, 2, 2, 2, 2, 2, 1,
       0, B, 3, 2, B, 2, 0, 3, 2, 0, 2, 3, 0, 1, 3, 3, 0, 3, 0, 0, 2, 0,
       1, 0, 3, 2, 1, 1, 1, 3, 1, 0, 3, 2, 2, 3, 1, 2, 3, 2, 1, 1 , 1, 8,
       0, 1, 0, 1, 3, 0, 2, 3, 1, 3, 0, 0, 0, 1, 1, 3, 2, 0, 2, 0, 2, 2,
       3, 2, 2, 0, 3, 2, 2, 2, 1, 2, 1, 2, 1, B, 3, 3, 1, 2, B, 3, 1, 3,
       2, 2, 3, 2, 1, 1, 0, 1, 2, 2, 2, 2, 0, 3, 1, 0, 2, 2, B, 2, B, 6,
       3, 0], dtype=int64)
```

In [48]:

```python
from sklearn.metrics import accuracy_score
svc_acc = accuracy_score(pred1,Y_test)
print(svc_acc)
print(confusion_matrix(pred1,Y_test))
```

```
0.88
[[46  3  0  0]
 [ 4 40  8  0]
 [ 0  3 52  4]
 [ 0  0  2 38]]
```

Logistic Regression

In [50]:

```python
from sklearn.linear model import LogisticRegression  # tts g  lgsst/icorion
lr=LogisticRegression()
lr.fit(X_train,Y_train)
```

```
LogisticRegression()
```

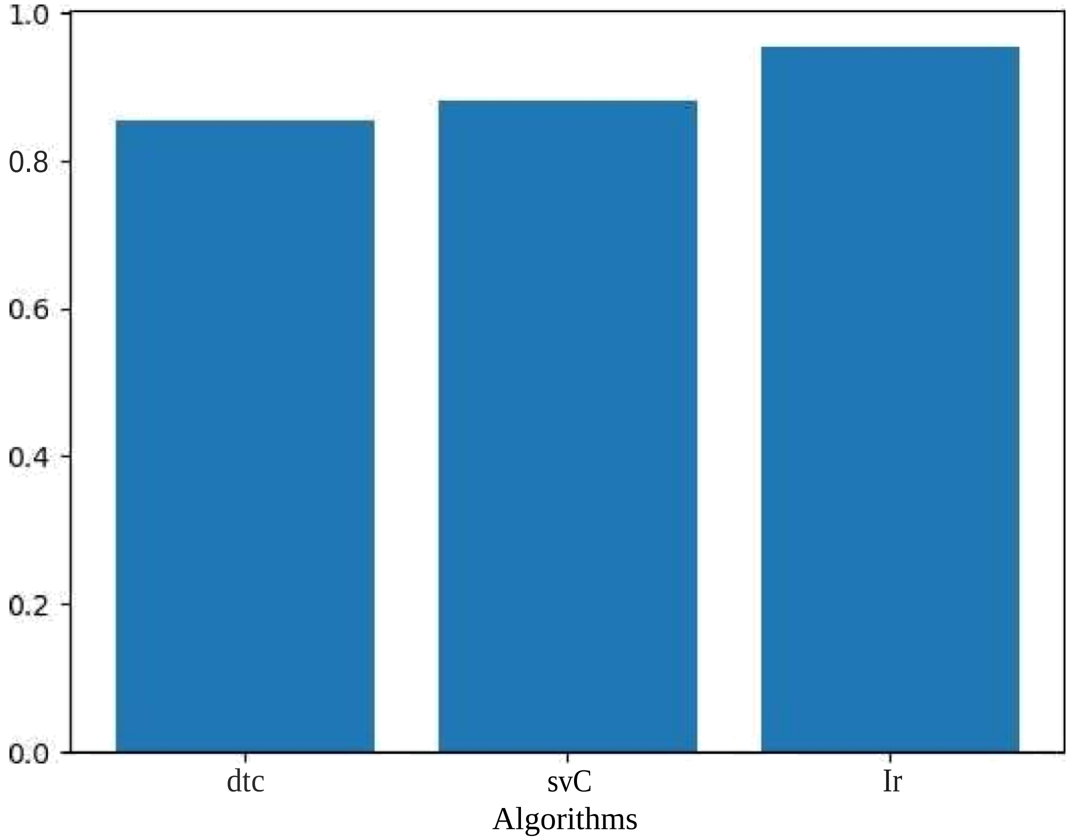
```
pred2 = 1 n. p red1ct X_t e st )
pred2
```

```
array([1, 1, 2, 1, 1, 1, 2, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 3, 1,
       2, 3, 2, 2, 2, 2, 8, 0, 2, 3, 0, 0, 3, 0, 0, 0, 1, 1, 1, 2, 3, 2,
       3, 0, 1, 3, 3, 1, 0, 0, 3, 3, 3, 3, 1, 3, 2, 3, 2, 2, 3, 1, 3, 1,
       0, 0, 0, 2, 1, 2, 3, 2, 1, 3, 3, 2, 0, 2, 0, 0, 2, 1, 2, 2, 2, 1,
       0, 0, 3, 2, 0, 2, 0, 3, 2, 0, 2, 3, 0, 1, 3, 3, 0, 3, 0, 0, 2, 0,
       1, 0, 3, 2, 2, 1, 1, 3, 1, 0, 3, 2, 2, 3, 1, 2, 3, 2, 1, 1, 1, 0,
       0, 1, 0, 2, 3, 0, 2, 3, 1, 3, 0, 0, 0, 1, 1, 2, 2, 0, 3, 1, 2, 2,
       3, 2, 2, 0, 3, 2, 2, 2, 2, 2, 1, 2, 1, 1, 3, 3, 1, 2, 0, 3, 1, 3,
       2, 2, 3, 2, 2, 1, 0, 1, 3, 2, 1, 2, 0, 3, 1, 0, 2, 2, 0, 2, 0, 0,
       3, 0], dtype=int64)
```

```
from sklearn.metrics import accuracy score
lr_acc: accuracy_score(pred2,Y_test)
print(lr_acc)
print(confusion_matrix(pred2,Y_test))
```

```
0.955
[[49  1  0  0]
 [ 1 45  3  0]
 [ 0  0 56  1]
 [ 0  0  3 41]]
```

```
plt.bar(x=[ 'dtc','svc','lr'],height=[dtc_acc,svc_acc,lr_acc])
plt.xlabel(”Algorithms“)
plt.ylabel(”Accuracy Score")
plt.show()
```




```
lr.predict(test)
```

```
array([3, 3, 2, 3, 1, 3, 3, 1, 3, 0, 3, 3, 0, 0, 2, 0, 2, 1, 3, 2, 1, 3,
       1, 1, 3, 0, 2, 0, 3, 0, 2, 0, 3, 0, 1, 1, 3, 1, 2, 1, 1, 2, 0, 0,
       0, 1, 0, 3, 1, 2, 1, 0, 3, 0, 3, 0, 3, 1, 1, 3, 3, 3, 0, 1, 1, 1,
       2, 3, I, 2, 1, 2, 2, 3, 3, 0, 2, 0, 1, 3, 0, 3, 3, 0, 3, 0, 3, 1,
       3, 0, 1, 2, 2, 1, 2, 2, 0, 2, 1, 2, 1, 0, 0, 3, 0, 2, 1, 1, 2, 3,
       3, 3, 1, 3, 3, 3, 3, 2, 3, 0, 0, 3, 2, 1, 2, 0, 3, 2, 2, 2, 0, 2,
       2, 1, 3, 1, 1, 0, 3, 2, 1, 2, 1, 3, 2, 3, 3, 3, 2, 3, 2, 3, 1, 0,
       3, 2, 3, 3, 3, 3, 3, 2, 3, 3, 3, 3, 1, 0, 3, 0, 0, 0, 2, 1, 0, 1,
       0, 0, 1, 2, 1, 0, 0, 1, 1, 2, 2, 1, 0, 0, 0, 1, 0, 3, 1, 0, 2, 2,
       3, 3, 1, 2, 3, 2, 3, 2, 2, 1, 0, 0, 1, 3, 0, 2, 3, 3, 0, 2, 0, 3,
       2, 3, 3, 1, 0, I, 0, 3, 0, 1, 0, 2, 2, 1, 3, 1, 3, 0, 3, 1, 2, 0,
       0, 2, 1, 3, 3, 3, 1, 1, 3, 0, 0, 2, 3, 3, 1, 3, 1, 1, 3, 2, 1, 2,
       3, 3, 3, 1, 0, 0, 2, 3, 1, 1, 3, 2, 1, 3, 0, 0, 3, 0, 0, 3, 2, 3,
       3, 2, 1, 3, 3, 2, 3, 1, 2, 1, 2, 0, 2, 3, 1, 0, 0, 3, 0, 3, 0, 1,
       2, 0, 2, 3, 1, 3, 2, 2, 1, 2, 0, 0, 0, 1, 3, 2, 0, 0, 0, 3, 2, 0,
       2, 3, I, 2, 2, 2, 3, 1, 3, 3, 2, 2, 2, 3, 3, 0, 3, 0, 3, 1, 3, 1,
       2, 3, 0, 1, 0, 3, 1, 3, 2, 3, 0, 0, 0, 0, 2, 0, 0, 2, 2, 1, 2, 2,
       2, 0, 1, 0, 0, 3, 2, 0, 3, 1, 2, 2, 1, 2, 3, 1, 1, 2, 2, 1, 2, 0,
       1, 1, 0, 3, 2, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 2, 2, 3, 2, 3, 0, 3,
       0, 3, 0, 1, 1, 0, 2, 0, 3, 2, 3, 3, 1, 3, 1, 3, 1, 2, 2, 0, 1, 2,
       1, 1, 0, 0, 0, 1, 2, 1, 0, 3, 2, 0, 2, 2, 0, 0, 3, 1, 2, 0, 2, 3,
       3, 0, 3, 0, 2, 3, 2, 3, 0, 2, 0, 2, 3, 0, 1, 1, 0, 0, 1, 1, 1, 3,
       3, 3, 2, 3, 1, 2, 2, 3, 3, 3, 2, 0, 2, 1, 2, 2, 1, 0, 2, 2, 0, 0,
       0, 3, 1, 0, 2, 2, 2, 0, 3, 1, 2, 2, 1, 3, 0, 2, 3, 0, 1, 1, 3, 3,
       2, 1, 1, 3, 2, 0, 3, 0, 2, 0, 3, 3, 1, 3, 2, 2, 3, 0, 1, 2, 3, 1,
       3, 2, 3, 1, 1, 0, 0, 3, 1, 0, 3, 2, 3, 3, 0, 3, 3, 3, 2, 3, 3, 1,
       2, 0, 2, 2, 3, I, 0, 1, 1, 2, 2, 2, 0, 0, 2, 2, 3, 2, 0, 2, 1, 3,
       3, 0, I, 3, 0, 2, 1, 1, 0, 0, 2, 1, 0, 1, 1, 2, 2, 0, 2, 2, 1, 0,
       3, 0, 0, 3, 2, 0, 0, 0, 0, 0, 3, 0, 3, 1, 3, 2, 1, 3, 3, 0, 1, 0,
       3, 2, 2, 2, 0, 3, 0, 2, 0, 2, 0, 0, 1, 1, 1, 2, 1, 3, 1, 3, 2, 2,
       1, 3, 2, 0, 2, 2, 0, 3, 3, 0, 2, 1, 1, 2, 0, 3, 2, 0, 3, 2, 3, 0,
```

```
0, 3, 0, 2, 2, 3, 2, 2, 2, 2, 1, 2, 3, 0, 1, 0, 1, 2, 1, 0, 0, 1,
0, 0, 3, 0, 1, 2, 0, 1, 0, 1, 3, 0, 3, 2, 3, 0, 0, 1, 2, 2, 1, 0,
1, 1, 0, 1, 1, 0, 0, 3, 3, 0, 3, 1, 1, 3, 0, 1, 0, 2, 2, 0, 3, 1,
0, 3, 0, 1, 0, 3, 3, 3, 2, 3, 0, 3, 2, 0, 0, 0, 3, 3, 2, 0, 2, 1,
3, 0, 0, 2, 2, 0, 3, 1, 2, 1, 1, 1, 3, 1, 1, 1, 2, 1, 0, 2, 2, 0,
2, 0, 0, 0, 0, 2, 3, 3, 3, 0, 1, 2, 1, 1, 0, 0, 2, 1, 0, 2, 0, 3,
2, 2, 1, 2, 0, 2, 1, 3, 0, 0, 3, 2, 3, 0, 0, 2, 3, 3, 1, 3, 2, 1,
0, B, 3, 3, l, 3, 0, 0, B, 2, 2, 1, 2, B, 3, 2, 1, 2, 3, 3, B, 1,
1, 2, l, 2, 2, 0, 1, 3, 1, 1, 3, 0, 2, 3, 2, 1, 1, I, 3, 3, B, 2,
3, 0, 2, 3, 2, 2, 2, 3, 2, 0, 1, 2, 1, 2, 1, 1, 2, 2, 2, 1, 2, 1,
0, I, 3, 1, 0, 1, 2, 3, 1, 0, 0, 3, 2, 2, 3, 0, 3, 2, 2, 1, 3, 0,
1, 3, I, 1, 1, 2, 3, 2, 0, 3, 0, 2, 3, 0, 3, 1, 3, 3, 1, 0, 2, 3,
1, 0, 2, 1, 2, I, 2, 0, 2, 2, 0, 2, 3, 2, 3, 0, 2, I, l 2, 2, 3,
3, 0, 2, 1, 2, 1, 3, 1, 1, 3, 0, 1, 0, B, 3, 3, 2, B, 0, 0, B, 3,
2, 3, 3, 0, 0, 2, 1, 0, 2, 2], dtype=int64)
```

Random Forest Regression

In [56]:

```
from sklearn.ensemble import RandomForestRegressor
```

In [59]

```
regressor  RardomForestRegressOr(n_estimators=l00,criteriOn='squared_error')
regressor.fit(X_train,Y_train)
```

out[s]; RandomForestRegressor()

In [61]

```
regressor.score(X_test, Y_test)
```

Out [61] : 0. 935 7660738714B91

In [62]

```
regressor  RandomForestRegressor(n_estimators=750,criterion='squared_error')
regressor.fit(X_train, Y_train)
regcessor.score(X_tes I,  Y_tesI)
```

Out[62]; 0.9361161650706794

I’ t68]'

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

Out[68]:

```
array([9.76000000e-01, 1.00533333e+00, 2.00133333e+00, 9.62666667e-01,
       1.97333333e-01, 1.18133333e+00, 2.00666667e+00, 8.50666667e-01,
       9.57333333e-01, B.93933333e-B1, 0.0000B0B0e+00, 7.65333333e-01,
       1.09866667e+00, 9.53333333e-01, 1.06400000e+00, 7.73333333e-02,
       8.14666667e-01, 1.63733333e+00, 7.70666667e-01, 3.32000000e-01,
       2.99866667e+00, 5.20000000e-01, 2.20266667e+00, 2.96266667e+00,
       2.02800000e+00, 1.98666667e+00, 2.08666667e+00, 1.16666667e+00,
       0.00000000e+00, 2.B0000000e-02, 1.91466667e+00, 2.96666667e+00,
       8.00000000e-03, g.&000000Be-03, 2.98400000e+00, 9.33333333e-03,
       8.00000000e-03, 1.33333333e-03, 1.02800000e+00, 1.29333333e+00,
       8.05333333e-01, 2.18533333e+00, 2.73600000e+00, 2.17600000e+00,
       2.82000000e+00, 7.20000000e-01, 1.20000000e+00, 2.95600000e+00,
       2.97200000e+0B, 1.13200B00e+00, 5.33333333e-03, 5.40000000e-01,
       2.13200000e+0B, 2.72266667e+00, 2.95066667e+00, 2.16666667e+00,
       3.B4B00000e-01, 2.892B0000e+00, 2.33g66667e+00, 3.000000B0e+00,
       I.9B133333e+0B, 1.98666667e+00, 2.99200B0Be+00, 9.960000B0e-Bl,
       2.87200000e+00, 1.19066667e+00, 7.46666667e-02, 2.98666667e-01,
```

```
       5.33333333e-03, 1.91200000e+00, 1.02266667e+00, 1.64800000e+00,
       2.93600000e+00, 1.85866667e+00, 1.66533333e+00, 2.99866667e+00,
       2.90533333e+00, 1.97866667e+00, 2.40000000e-02, 2.09333333e+00,
       2.76000000e-01, 2.53333333e-02, 2.16533333e+00, 7.54666667e-01,
    1.98933333e+00, 2.29733333e+00, 1.87066667e+00, 1.41466667e+00,
       1.06666667e-02, 1.33333333e-03, 2.90666667e+00, 2.52000000e+00,
       0.00000000e+00, 2.02000000e+00, 0.00000000e+00, 2.96666667e+00,
    1.85200000e+00, 0.00000000e+00, 1.79333333e+00, 2.98933333e+00,
       2.66666667e-03, 1.28800000e+00, 2.13333333e+00, 2.74400000e+00,
       2.53333333e-02, 2.43733333e+00, 1.93333333e-01, 5.33333333e-03,
       2.02666667e+00, 4.00000000e-03, 9.97333333e-01, 0.00000000e+00,
       2.99200000e+00, 1.98666667e+00, 1.97866667e+00, 1.76400000e+00,
       1.51200000e+00, 2.80533333e+00, 1.07066667e+00, 6.13333333e-02,
       2.99733333e+00, 2.94400000e+00, 1.80933333e+00, 2.98400000e+00,
       1.07200000e+00, 2.72266667e+00, 2.99066667e+00, 1.69200000e+00,
       9.62666667e-01, 9.90666667e-01, 4.12000000e-01, 3.20000000e-02,
       6.69333333e-01, 1.02400000e+00, 6.80000000e-02, 1.88000000e+00,
       3.00000000e+00, 7.20000000e-02, 1.90533333e+00, 2.73733333e+00,
       1.41333333e+00, 2.99600000e+00, 1.33333333e-03, 6.44000000e-01,
       5.33333333e-02, 4.32000000e-01, 8.21333333e-01, 2.61866667e+00,
       2.22400000e+00, 2.66666667e-03, 2.68666667e+00, 8.89333333e-01,
       2.60666667e+00, 1.88266667e+00, 3.00000000e+00, 2.30533333e+00,
       2.00533333e+00, 2.80000000e-02, 2.99866667e+00, 1.29466667e+00,
       1.93600000e+00, 1.93600000e+00, 1.99866667e+00, 1.76000000e+00,
       9.68000000e-01, 1.97333333e+00, 1.16400000e+00, 9.26666667e-01,
       2.50133333e+00, 2.98533333e+00, 1.06400000e+00, 1.98133333e+00,
       5.33333333e-02, 2.99866667e+00, 1.45066667e+00, 2.85600000e+00,
       1.49466667e+00, 1.94g 0000e+00, 2.66133333e+00, 1.67733333e+00,
       1.52000000e+00, 9.48000000e-01, 7.20000000e-02, 5.97333333e-01,
       2.84133333e+00, 2.34400000e+00, 1.06000000e+00, 1.97200000e+00,
       1.08000000e-01, 2.96000000e+00, 9.09333333e-01, 0.00000000e+00,
       2.03733333e+00, 2.35066667e+00, 7.06666667e-02, 2.35866667e+00,
       0.00000000e+00, 1.06666667e-02, 2.96000000e+00, 3.86666667e-02])
```

```
Y_test
```

```
1458    1
198     1
1276    2
1243    1
1267    1

773     2
756     0
1166    8
1734    3
138     0
Name: price_range, Length: 200, dtype: int64
```

```
Y_tes.ilo[-1]
```

```
0
```

Confusion Matrix

```
#train test split of data
from sklearn.model_selection import train_test_spl1t
X_train, X_valid, y_train, y_valid= train_test_split(X, y, test_size=0.2, random_state=
```

In [75]:

```
#conf-us hon mom›x
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
def my_confusion_matrix(y_test, y_pred, plt_title):
    cm=confusion matrix(y test, y pred)
    print(classification_report(y_test, y_pred))
    sns.heatmap(cm,annot=True, fmt='g', cbar=False, cmap='BuPu')
    plt.xlabel('Predicted Values')
    plt.ylabel('Actual Values')
    plt.title(plt_title)
    plt.show()
    return cm
```

KNN Classification

In [76]:

```
from sklearn.neighbors import KNeighborsClassifier
knn   KNeighborsClassifier(n_neighbors=3,leaf_size=25)
```

In [77]:

```
knr.fit(X_train, y_train)
y_pred_knn=knn.predict(X_valid)
```

```
D:\Software_Installation\Anaconda\lib\site-packages\sklearn\neighbors\_classification.p
y:228: FutureWarning: Unlike other reduction functions (e.g. 'skew', 'kurtosis'), the de
fault behavior of 'mode' typically preserves the axis it acts along. In SciPy 1.11.0, th
is behavior will change: the default value of  keepdims' will become False, the 'axis' o
ver which the statistic is taken will be eliminated, and the value None will no longer b
e accepted. Set 'keepdims' to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=l)
```

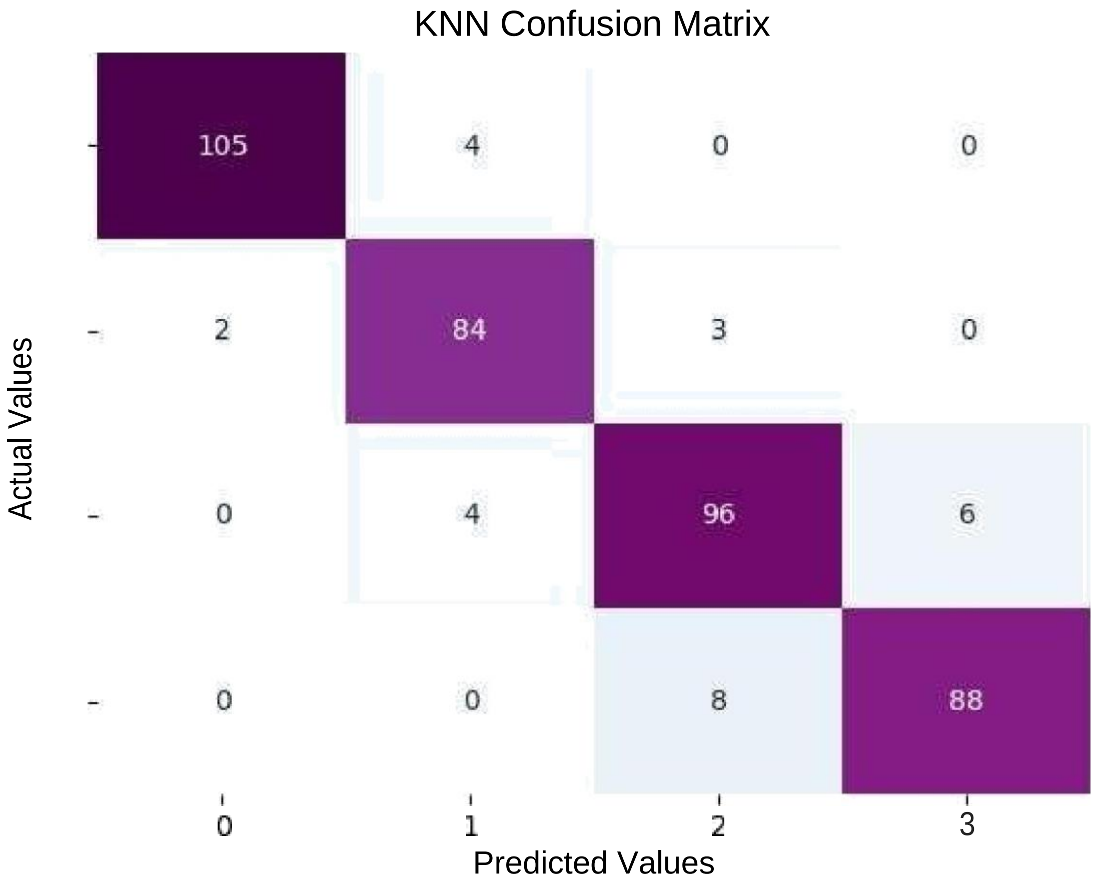
In [78]:

```
print('KNN Classifier Accuracy Score:',accuracy_score(y_valid,y_pred_knn))
cm_rfc=my_confusion_matrix(y_valid,y_pred_knn, 'KNN Confusion Matrix')
```

```
KNN Classifier Accuracy Score:  0.9325
              precision    recall  fl-score   support

           0       0.98      0.96      0.97       109
           I       0.91      0.94      0.99        89
           2       B.9B      B.91      0.90       106
           3       B.94      0.92      B.93        96

    accuracy                           B.93       400
   macro avg       0.93      0.93      0.93       400
weighted avg       6.93      6.93      8.93       400
```



SVM Classifier

```
from sklearn import sum
svm_clf = svm,SVC(decision_function_shdpe='ovo')
```

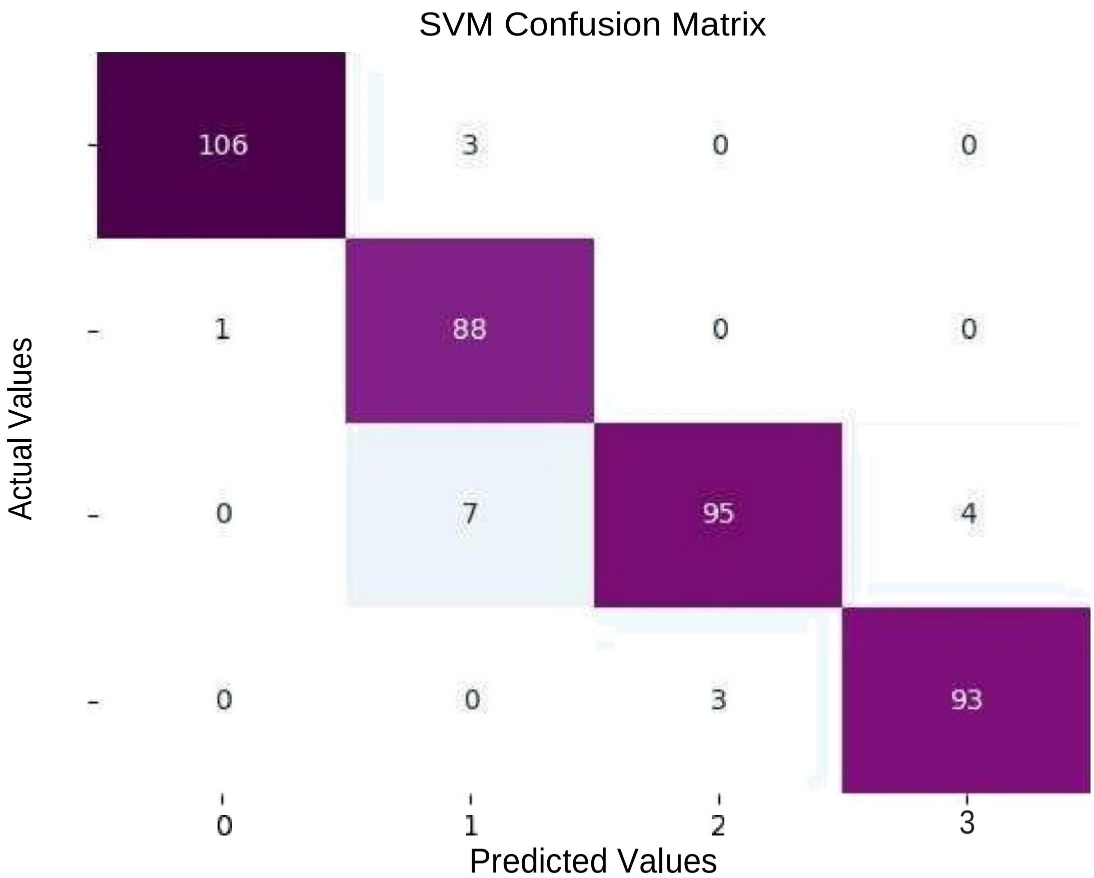
```
svm_clf.fit(X_train, y_train)
y_pred_svm=svm_clf.predict(X_valid)
```

```
print('SVM Classi*ie  Accuracy Score: ',accuracy_score(y_valid,y_pred_svm))
cm_rfc=my_confusion_matrix(y_valid, y_pred_svm, 'SUM Confusion Matrix')
```

```
SVM Classifier Accuracy Score:  0.955
              precision    recall  fl-score   support

           0       0.99      0.97      0.98       109
           1       0.90      0.99      0.94        89
           2       0.97      0.90      0.93       106
           3       0.96      0.97      0.96        96

    accunac y                          0.95       400
   macro avg       0.95      0.96      0.95       400
weighted avg       0.96      0.95      0.95       400
```



Logistic Regression Score : 95.5%

Random Forest Regression Score : 93%

KNN Classification Score : 93%

SVM Classifier Score : 95%

Decision Tree Classification Score : 85%