MOBILE PRICE CLASSIFICATION

Since we are to predict the price range indicating how high the price is based on that we classify our price range from 0-3 where,

0 - Low cost

1 - Medium cost

2 - High cost

3 - Very High cost

In [2]:

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

In [3]:

```
df = pd.read_csv('mobile_price_range_data.csv')
df.head()
```

Out[3]:

	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 rows × 21 columns

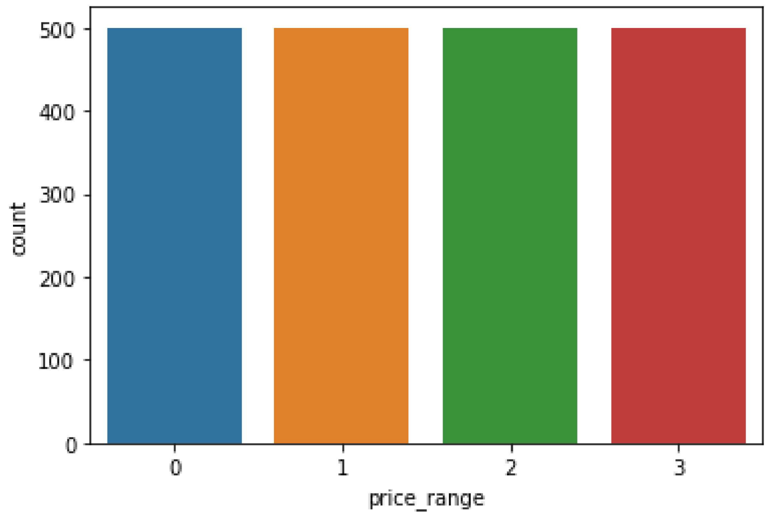
In [4]:

```
sns.countplot(df['price_range']) #shows me the count of observations
```

```
C:\Users\fozan\anaconda3\lib\site-packages\seaborn\_decorators.py:36: Future
Warning: Pass the following variable as a keyword arg: x. From version 0.12,
the only valid positional argument will be `data`, and passing other argumen
ts without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

Out[4]:

```
<AxesSubplot:xlabel='price_range', ylabel='count'>
```



In [5]:

```
df.shape
```

Out[5]:

```
(2000, 21)
```

In [6]:

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

Out[6]:

```
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
```

In [7]:

```
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 [8]:

```
df.describe()
```

Out[8]:

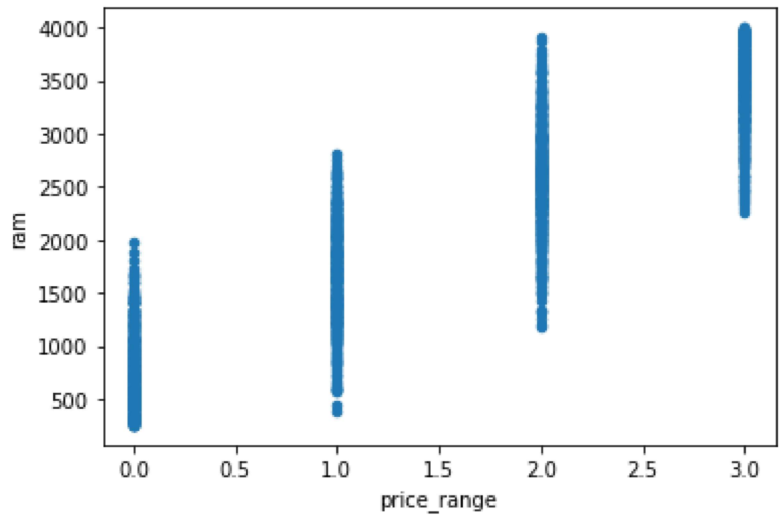
	battery_power	blue	clock_speed	dual_sim	fc	four_g	int_men
count	2000.000000	2000.0000	2000.000000	2000.000000	2000.000000	2000.000000	2000.00C
mean	1238.518500	0.4950	1.522250	0.509500	4.309500	0.521500	32.046
std	439.418206	0.5001	0.816004	0.500035	4.341444	0.499662	18.145
min	501.000000	0.0000	0.500000	0.000000	0.000000	0.000000	2.00C
25%	851.750000	0.0000	0.700000	0.000000	1.000000	0.000000	16.00C
50%	1226.000000	0.0000	1.500000	1.000000	3.000000	1.000000	32.00C
75%	1615.250000	1.0000	2.200000	1.000000	7.000000	1.000000	48.00C
max	1998.000000	1.0000	3.000000	1.000000	19.000000	1.000000	64.00C

8 rows × 21 columns

Generating visualizations with respect to a few features such as - ram, battery_power, fc, wifi etc.

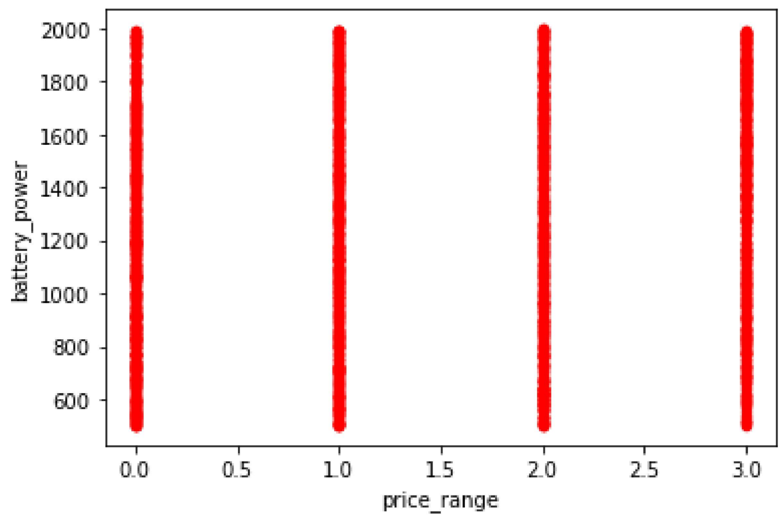
In [9]:

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



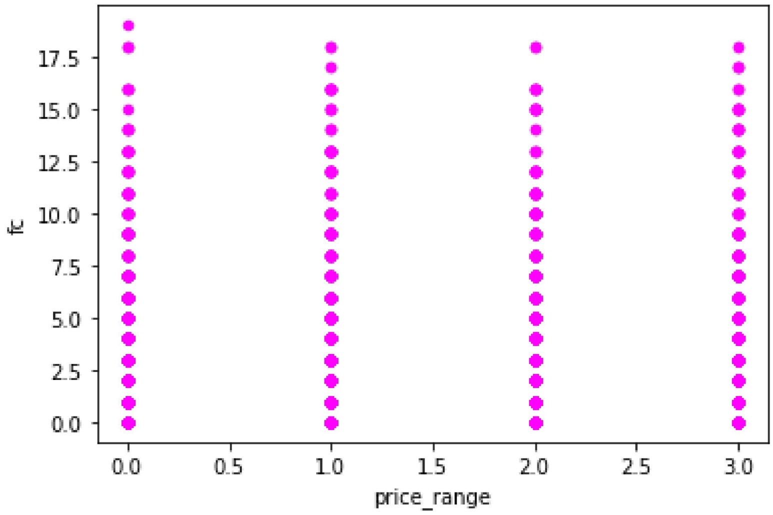
In [10]:

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



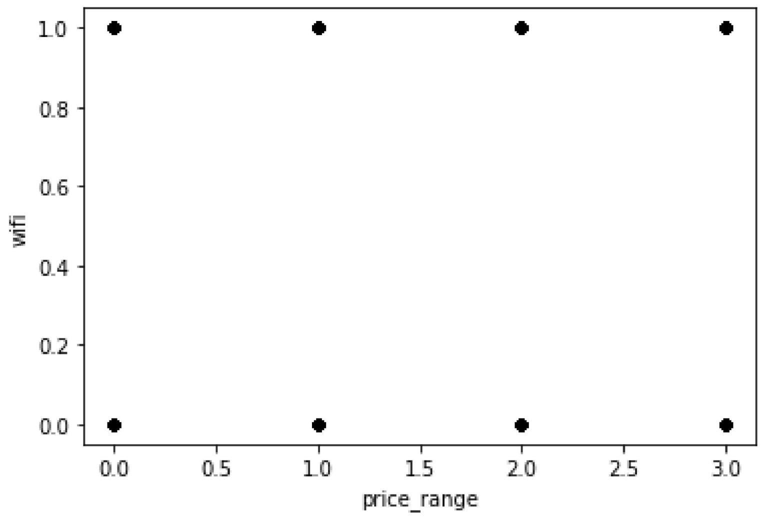
In [11]:

```
df.plot(kind = 'scatter',x = 'price_range', y = 'fc',color = 'magenta')
plt.show()
```



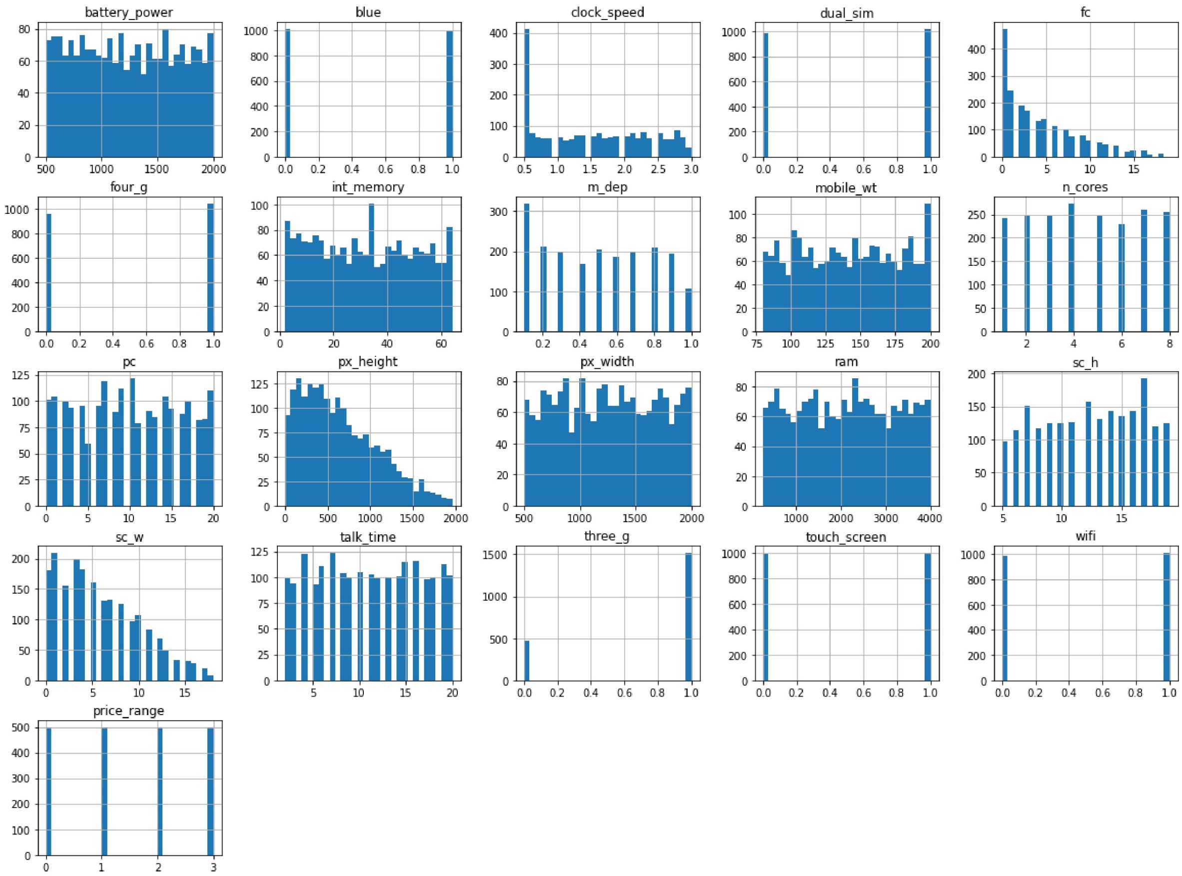
In [12]:

```
df.plot(kind = 'scatter',x = 'price_range', y = 'wifi',color = 'black')
plt.show()
```



In [13]:

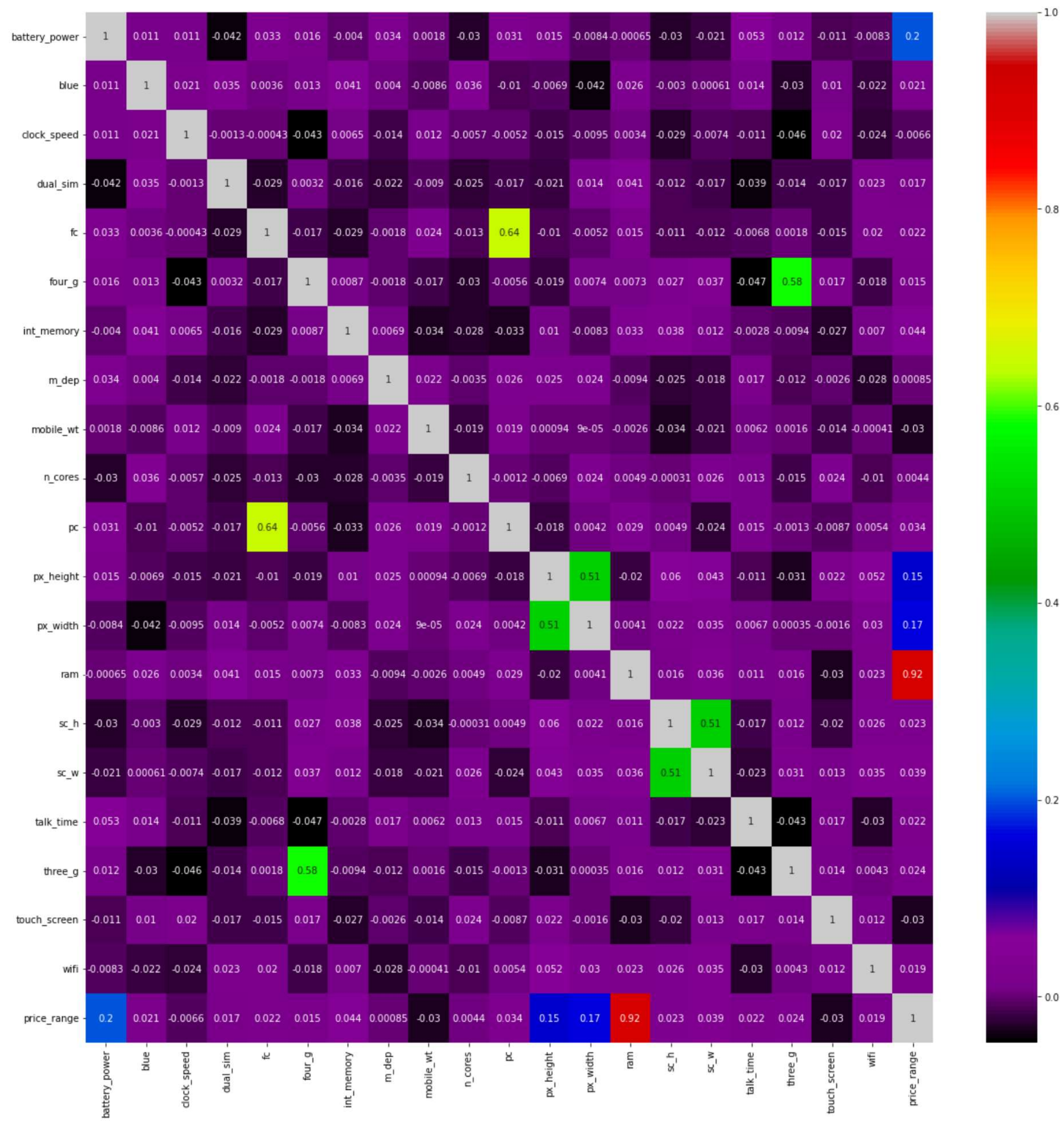
```
df.hist(figsize=(20,15), bins=30)
plt.show()
```



We can generate a 'CORRELATION HEATMAP' which is a graphical representation of a correlation matrix to understand the relation between different varaibles

In [14]:

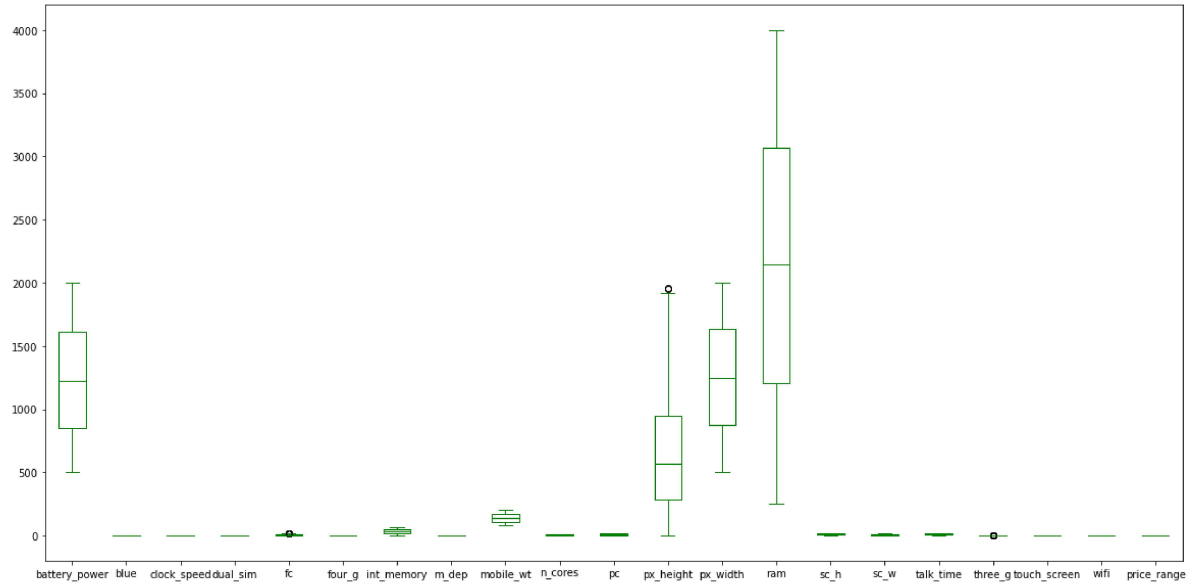
```
plt.figure(figsize=(20,20))
sns.heatmap(df.corr(),annot=True,cmap=plt.cm.nipy_spectral)
plt.show()
```



We can also generate a 'BOXPLOT' in order to determine potential outliers

In [15]:

```
df.plot(kind= 'box',figsize=(20,10),color = 'Green')
plt.show()
```



In [16]:

```
x = df.drop('price_range',axis=1)
y = df['price_range']
print(type(x))
print(type(y))
```

```
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.series.Series'>
```

In [17]:

```
print(x.shape)
print(y.shape)
```

```
(2000, 20)
(2000,)
```

In [18]:

```
from 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)
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(1800, 20)
(200, 20)
(1800,)
(200,)
```

In [19]:

```
# 200 rows are dumped into the x_test and the remaining 1800 rows are dumped into the x_tra
# Similarly 200 rows are dumped into the y_test and the remaining 1800 rows are dumped into
# y_train.

0.1*2000
```

Out[19]:

```
200.0
```

We perform Standardization

- Standardization comes into picture when features of input data set have large differences between their ranges, or simply when they are measured in different measurement units
- Standardization makes all the features value in range 0 to 1

In [20]:

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

In [21]:

```
sc = StandardScaler()
x_train = sc.fit_transform(x_train)
x_test = sc.fit_transform(x_test)
```

In [22]:

```
x_train
```

Out[22]:

```
array([[-1.62737257, -0.98675438, -1.01271559, ..., -1.78222729,
        -1.00892875, -0.99888951],
       [-0.75199354,  1.01342342,  0.58093235, ..., -1.78222729,
         0.99115027, -0.99888951],
       [-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 , ...,  0.56109566,
        -1.00892875,  1.00111173],
       [ 0.4144206 , -0.98675438, -0.39977408, ...,  0.56109566,
         0.99115027,  1.00111173]])
```

In [23]:

```
x_test
```

Out[23]:

```
array([[ 0.38671473, -1.02020406, -1.21086485, ...,  0.54653573,
        -0.98019606, -1.16316   ],
       [-1.36645979, -1.02020406, -1.21086485, ...,  0.54653573,
         1.02020406,  0.85972695],
       [-1.41958629, -1.02020406, -0.10729182, ...,  0.54653573,
        -0.98019606,  0.85972695],
       ...,
       [-0.46561913,  0.98019606,  0.38318508, ...,  0.54653573,
        -0.98019606, -1.16316   ],
       [ 0.19499736, -1.02020406, -0.84300717, ...,  0.54653573,
         1.02020406,  0.85972695],
       [-1.53507869, -1.02020406,  1.11890043, ...,  0.54653573,
         1.02020406, -1.16316   ]])
```

1) Logistic Regression

In [24]:

```
from sklearn.linear_model import LogisticRegression
```

In [25]:

```
m1 = LogisticRegression()
m1.fit(x_train,y_train)
```

Out[25]:

```
LogisticRegression()
```

In [26]:

```
ypred_m1 = m1.predict(x_test)
print(ypred_m1)
```

```
[1 1 2 1 1 1 2 1 1 1 0 1 1 1 1 0 0 2 0 1 3 1 2 3 2 2 2 2 0 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
 2 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 3 2 3 1 3 3 2 1 1 1 0 0 1 0 2 3 0 2 3 1 3 0 0 0 1 1 3
 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
 3 1 2 0 3 1 0 2 2 0 2 0 0 3 0]
```

In [27]:

```
from sklearn.metrics import confusion_matrix,classification_report
```

In [28]:

```
cm = confusion_matrix(y_test,ypred_m1)
print(cm)
print(classification_report(y_test,ypred_m1))
```

```
[[49  1  0  0]
 [ 0 46  0  0]
 [ 0  1 55  6]
 [ 0  0  0 42]]
              precision    recall  f1-score   support

           0       1.00      0.98      0.99        50
           1       0.96      1.00      0.98        46
           2       1.00      0.89      0.94        62
           3       0.88      1.00      0.93        42

    accuracy                           0.96       200
   macro avg       0.96      0.97      0.96       200
weighted avg       0.96      0.96      0.96       200
```

In [29]:

```
from sklearn.metrics import accuracy_score
LR_acc = accuracy_score(ypred_m1,y_test)
print(LR_acc)
```

```
0.96
```

Hence Logistic Regression shows a 96% Accuracy

2) KNN Classification

In [30]:

```
from sklearn.neighbors import KNeighborsClassifier
```

In [47]:

```
m2 = KNeighborsClassifier(n_neighbors = 19)
m2.fit(x_train,y_train)
```

Out[47]:

```
KNeighborsClassifier(n_neighbors=19)
```

In [48]:

```
ypred_m2 = m2.predict(x_test)
print(ypred_m2)
```

```
[2 1 2 1 1 1 3 1 0 0 0 1 1 0 0 0 1 1 1 0 2 1 3 3 2 2 0 0 0 0 0 3 0 1 3 0 0
 0 1 1 1 3 2 3 3 1 2 3 3 0 0 0 2 3 3 2 0 3 3 3 2 1 3 1 3 1 0 1 0 2 0 1 3 1
 1 3 2 2 1 2 0 0 3 2 3 2 2 2 0 1 3 1 0 2 0 2 1 0 2 2 0 2 3 2 0 3 0 0 2 0 0
 0 3 1 1 1 1 3 2 0 3 2 2 3 1 2 3 2 0 1 1 2 0 1 0 0 3 0 3 3 1 3 0 2 0 1 1 3
 2 0 2 0 2 2 3 2 2 0 3 0 2 0 2 2 1 2 2 0 3 3 2 1 1 3 1 3 2 2 2 2 2 1 0 1 3
 3 2 1 0 3 1 0 2 2 0 1 0 0 3 0]
```

In [49]:

```
from sklearn.metrics import confusion_matrix,classification_report
```

In [50]:

```


cm = confusion_matrix(y_test,ypred_m2)
print(cm)
print(classification_report(y_test,ypred_m2))
```

```
[[40  8  2  0]
 [11 27  8  0]
 [ 6 12 33 11]
 [ 0  0  9 33]]
              precision    recall  f1-score   support

           0       0.70      0.80      0.75        50
           1       0.57      0.59      0.58        46
           2       0.63      0.53      0.58        62
           3       0.75      0.79      0.77        42

    accuracy                           0.67       200
   macro avg       0.67      0.68      0.67       200
weighted avg       0.66      0.67      0.66       200
```

In [51]:

```
from sklearn.metrics import accuracy_score
```

In [52]:

```
KNN_acc = accuracy_score(ypred_m2,y_test)
print(KNN_acc)
```

```
0.665
```

Hence KNN shows a 66.5% accuracy

3) SVM Classifier

- Linear kernel

In [53]:

```
from sklearn.svm import SVC
```

In [54]:

```
m3 = SVC(kernel ='linear',C = 1)
m3.fit(x_train,y_train)
```

Out[54]:

```
SVC(C=1, kernel='linear')
```

In [55]:

```
ypred_m3 = m3.predict(x_test)
print(ypred_m3)
```

```
[1 1 2 1 1 1 2 1 1 1 0 1 1 1 1 0 0 2 0 1 3 1 2 3 2 2 2 2 0 0 2 3 0 0 3 0 0
 0 2 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 2 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 3 0 2 0 3 2 0 2 3 0 1 3 3 0 3 0 0 2 0 1
 0 3 2 2 2 1 3 1 0 3 3 2 3 1 3 3 2 1 1 1 0 0 1 0 1 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
 3 1 2 0 3 1 0 2 2 0 2 0 0 3 0]
```

In [56]:

```
from sklearn.metrics import confusion_matrix,classification_report
cm = confusion_matrix(y_test,ypred_m3)
print(cm)
print(classification_report(y_test,ypred_m3))
```

```
[[49  1  0  0]
 [ 0 43  3  0]
 [ 0  3 53  6]
 [ 0  0  0 42]]
              precision    recall  f1-score   support

           0       1.00      0.98      0.99        50
           1       0.91      0.93      0.92        46
           2       0.95      0.85      0.90        62
           3       0.88      1.00      0.93        42

    accuracy                           0.94       200
   macro avg       0.93      0.94      0.94       200
weighted avg       0.94      0.94      0.93       200
```

In [57]:

```
from sklearn.metrics import accuracy_score
SVML_acc = accuracy_score(ypred_m3,y_test) #SVML - SVM Linear
print(SVML_acc)
```

```
0.935
```

SVM Classifier with the Linear model generates an accuracy of 93.5%

- rbf kernel

In [58]:

```
m4 = SVC(kernel = 'rbf',gamma = 0.1,C = 1)
m4.fit(x_train,y_train)
```

Out[58]:

```
SVC(C=1, gamma=0.1)
```

In [59]:

```
ypred_m4 = m4.predict(x_test)
print(ypred_m4)
```

```
[1 1 2 1 1 1 2 1 1 1 0 1 1 1 1 1 1 1 0 0 3 0 2 3 2 2 2 2 0 0 2 3 0 0 3 0 0
 0 1 1 1 3 3 2 3 1 2 3 3 1 0 1 2 3 2 2 1 3 2 3 2 2 3 1 3 1 0 1 0 2 1 1 3 1
 1 3 3 2 1 2 0 0 2 2 2 2 2 1 0 0 3 2 0 2 0 3 1 0 2 3 0 2 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 0 0 1 0 1 3 0 2 3 1 3 0 0 0 1 1 3
 2 0 2 0 2 1 3 2 2 0 3 2 2 2 1 2 1 2 2 1 3 3 1 2 1 3 1 3 2 2 3 2 1 1 0 1 2
 3 2 1 0 2 1 0 2 2 0 2 0 0 3 0]
```

In [60]:

```
cm = confusion_matrix(y_test,ypred_m4)
print(cm)
print(classification_report(y_test,ypred_m4))
```

```
[[42  8  0  0]
 [ 3 39  4  0]
 [ 0 11 47  4]
 [ 0  0  5 37]]
              precision    recall  f1-score   support

           0       0.93      0.84      0.88        50
           1       0.67      0.85      0.75        46
           2       0.84      0.76      0.80        62
           3       0.90      0.88      0.89        42

    accuracy                           0.82       200
   macro avg       0.84      0.83      0.83       200
weighted avg       0.84      0.82      0.83       200
```

In [61]:

```
from sklearn.metrics import accuracy_score
SVMrbf_acc = accuracy_score(ypred_m4,y_test) #SVMrbf - SVM rbf
print(SVMrbf_acc)
```

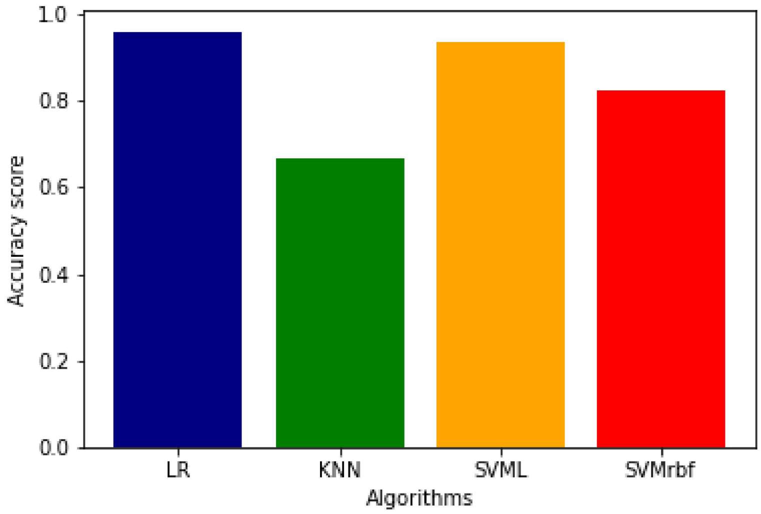
```
0.825
```

SVM Classifier with the rbf model generates an accuracy of 82.5%

Report on the Model with the best accuracy

In [62]:

```
plt.bar(x = ['LR','KNN','SVML','SVMrbf'],height = [LR_acc,KNN_acc,SVML_acc,SVMrbf_acc],
        color = ['navy','green','orange','red'])
plt.xlabel('Algorithms')
plt.ylabel('Accuracy score')
plt.show()
```



Therefore, Logistic Regression with a 96% Accuracy exhibits the highest accuracy among the 3 Models.