In [6]:
```
from sklearn.datasets import make_blobs
```

In [7]:
```
#create random dataset
data=make_blobs(n_samples=200,n_features=2,centers=3,cluster_std=1.6,random_state=101)
#n_samples = Total number of points equally divided among cluster.
#n_features = It indicated the number of features(columns)
#center = It determine number of cluster to be generated
# cluster_std=it sets the stsndard deviation of clusters high value makes the clusters to spread out
```

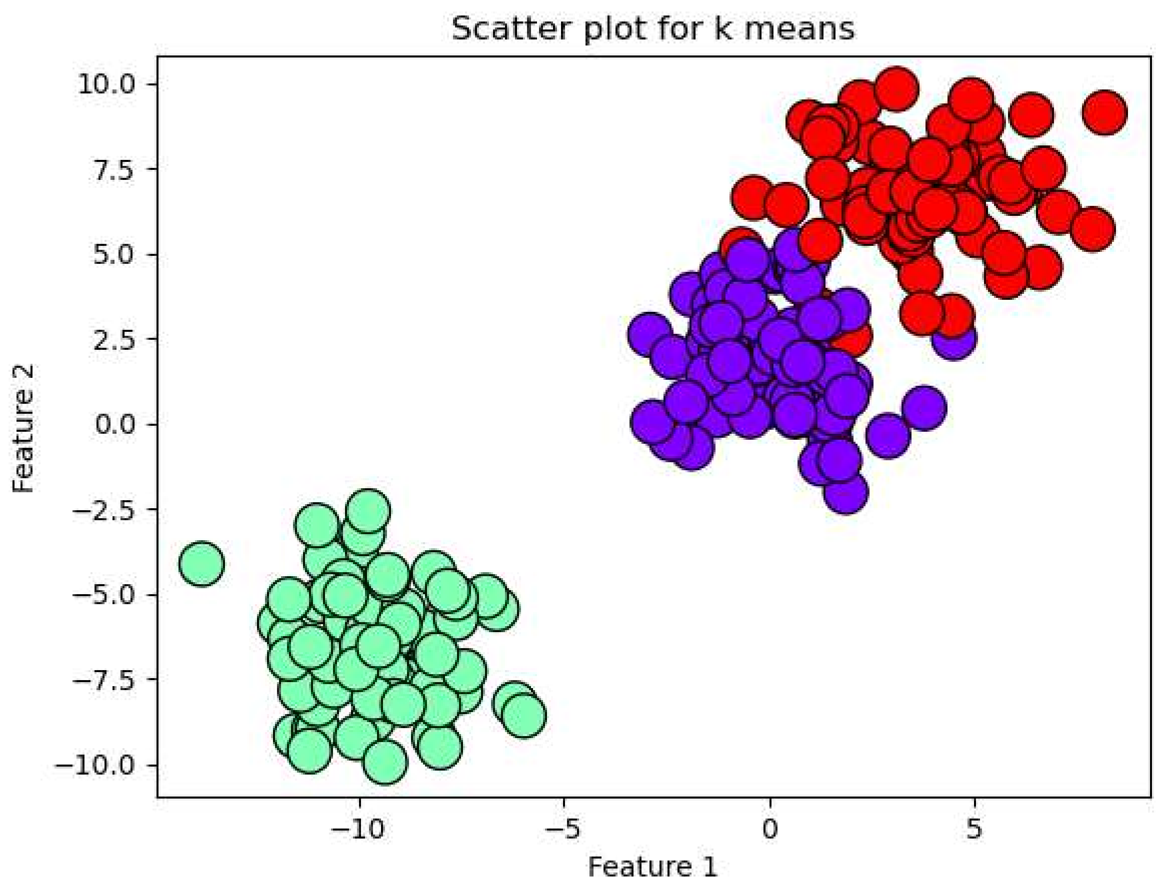
In [8]:
```
data
```

Out[8]:
```
(array([[  1.14686658,   3.36779908],
        [-10.05625782,  -3.78000376],
        [ -0.07506257,   0.48835932],
        [ -1.8846996 ,   3.78534453],
        [-11.52088716,  -9.18224527],
        [ -0.90615321,   1.5901156 ],
        [  5.31725825,   7.2055911 ],
        [ -1.26124905,   1.72823095],
        [-11.07783892,  -9.00933573],
        [  0.62988505,   0.19915645],
        [  1.79402071,   6.44556718],
        [  1.36976127,   0.90244287],
        [  1.35486137,  -0.03480811],
        [  5.20973207,   7.8718912 ],
        [  4.57619442,   7.79521043],
        [  1.35003178,   1.94078582],
        [ -9.57339298,  -5.45186063],
        [  1.8663056 ,  -2.01258793],
        [-13.80970682,  -4.1334675 ],
```

In [9]:
```
import matplotlib.pyplot as plt
x,y =data

plt.scatter(x[:,0],x[:,1],c=y,cmap='rainbow',edgecolor='black',s=250)
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.title("Scatter plot for k means")
```

Out[9]: Text(0.5, 1.0, 'Scatter plot for k means')



In [10]:
```
data[0].shape
```

Out[10]: (200, 2)

In [11]:
```
from sklearn.cluster import KMeans
```

In [12]:
```
KMeans= KMeans(n_clusters=4)
```

In [13]:
```
KMeans.fit(data[0])
```

```
C:\ProgramData\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1412: FutureWarning: The default value
of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warni
ng
  super()._check_params_vs_input(X, default_n_init=10)
C:\ProgramData\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1436: UserWarning: KMeans is known to
have a memory leak on Windows with MKL, when there are less chunks than available threads. You can avoid it
by setting the environment variable OMP_NUM_THREADS=1.
  warnings.warn(
```

Out[13]:
```
▾ KMeans
KMeans(n_clusters=4)
```

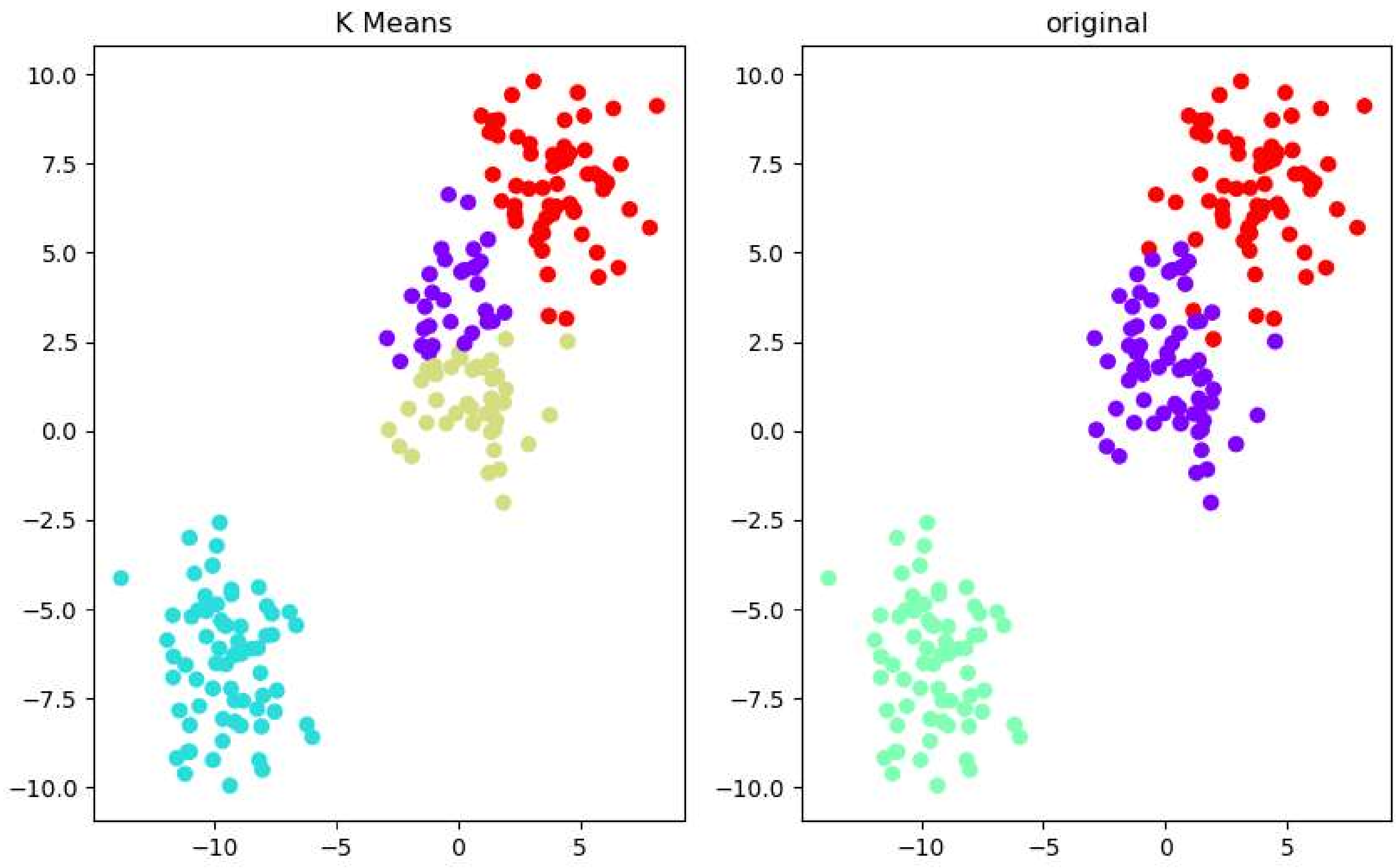
In [14]:
```
KMeans.cluster_centers_
```

Out[14]:
```
array([[-0.19751051,  3.83102454],
       [-9.47259134, -6.51081416],
       [ 0.60637724,  0.77020273],
       [ 4.05652885,  6.91272057]])
```

In [15]:
```
fig, (ax1,ax2) = plt.subplots(1,2, figsize=(10,6))
ax1.set_title(" K Means") #Predicted data
ax1.scatter(data[0][:,0],data[0][:,1],c=KMeans.labels_, cmap="rainbow")

ax2.set_title("original") # Original data
ax2.scatter(data[0][:,0],data[0][:,1],c=data[1], cmap="rainbow")
```

Out[15]: <matplotlib.collections.PathCollection at 0x1fb0effe890>



In [16]:
```
#project :4
import pandas as pd
import numpy as np
```

In [17]:
```
import matplotlib.pyplot as plt
import seaborn as sns
```

In [18]:
```
df=pd.read_csv(r"C:\Users\vippa\Downloads\College_Data.unknown")
```

In [19]:
```
df.head()
```

Out[19]:

	Unnamed: 0	Private	Apps	Accept	Enroll	Top10perc	Top25perc	F.Undergrad	P.Undergrad	Outstate	Room.Board	Books	Personal
0	Abilene Christian University	Yes	1660	1232	721	23	52	2885	537	7440	3300	450	2200
1	Adelphi University	Yes	2186	1924	512	16	29	2683	1227	12280	6450	750	1500
2	Adrian College	Yes	1428	1097	336	22	50	1036	99	11250	3750	400	1165
3	Agnes Scott College	Yes	417	349	137	60	89	510	63	12960	5450	450	875
4	Alaska Pacific University	Yes	193	146	55	16	44	249	869	7560	4120	800	1500

In [20]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 777 entries, 0 to 776
Data columns (total 19 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   Unnamed: 0   777 non-null    object 
 1   Private      777 non-null    object 
 2   Apps         777 non-null    int64  
 3   Accept       777 non-null    int64  
 4   Enroll       777 non-null    int64  
 5   Top10perc    777 non-null    int64  
 6   Top25perc    777 non-null    int64  
 7   F.Undergrad  777 non-null    int64  
 8   P.Undergrad  777 non-null    int64  
 9   Outstate     777 non-null    int64  
 10  Room.Board   777 non-null    int64  
 11  Books        777 non-null    int64  
 12  Personal     777 non-null    int64  
 13  PhD          777 non-null    int64  
 14  Terminal     777 non-null    int64  
 15  S.F.Ratio    777 non-null    float64
 16  perc.alumni  777 non-null    int64  
 17  Expend       777 non-null    int64  
 18  Grad.Rate    777 non-null    int64  
dtypes: float64(1), int64(16), object(2)
memory usage: 115.5+ KB
```

In [21]:
```
#to total number of null or NAN field
df.isna().sum()
```

Out[21]:
```
Unnamed: 0     0
Private        0
Apps           0
Accept         0
Enroll         0
Top10perc      0
Top25perc      0
F.Undergrad    0
P.Undergrad    0
Outstate       0
Room.Board     0
Books          0
Personal       0
PhD            0
Terminal       0
S.F.Ratio      0
perc.alumni    0
Expend         0
Grad.Rate      0
dtype: int64
```

In [22]:
```
df.duplicated()
```

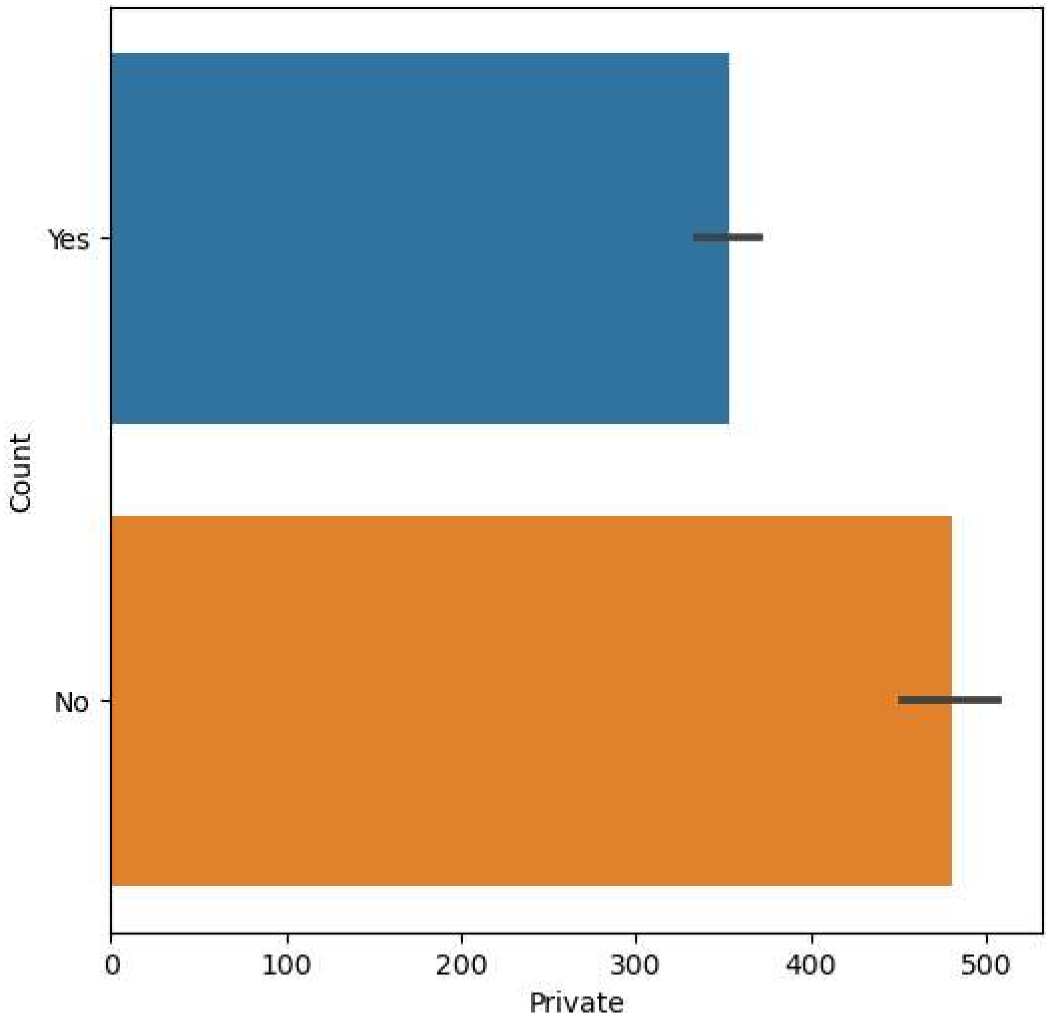
Out[22]:
```
0      False
1      False
2      False
3      False
4      False
       ...
772    False
773    False
774    False
775    False
776    False
Length: 777, dtype: bool
```

In [23]:
```
if not df.duplicated().empty:
    print(df[df.duplicated()])
else:
    print("No duplicate datas")
```

```
Empty DataFrame
Columns: [Unnamed: 0, Private, Apps, Accept, Enroll, Top10perc, Top25perc, F.Undergrad, P.Undergrad, Outstat
e, Room.Board, Books, Personal, PhD, Terminal, S.F.Ratio, perc.alumni, Expend, Grad.Rate]
Index: []
```

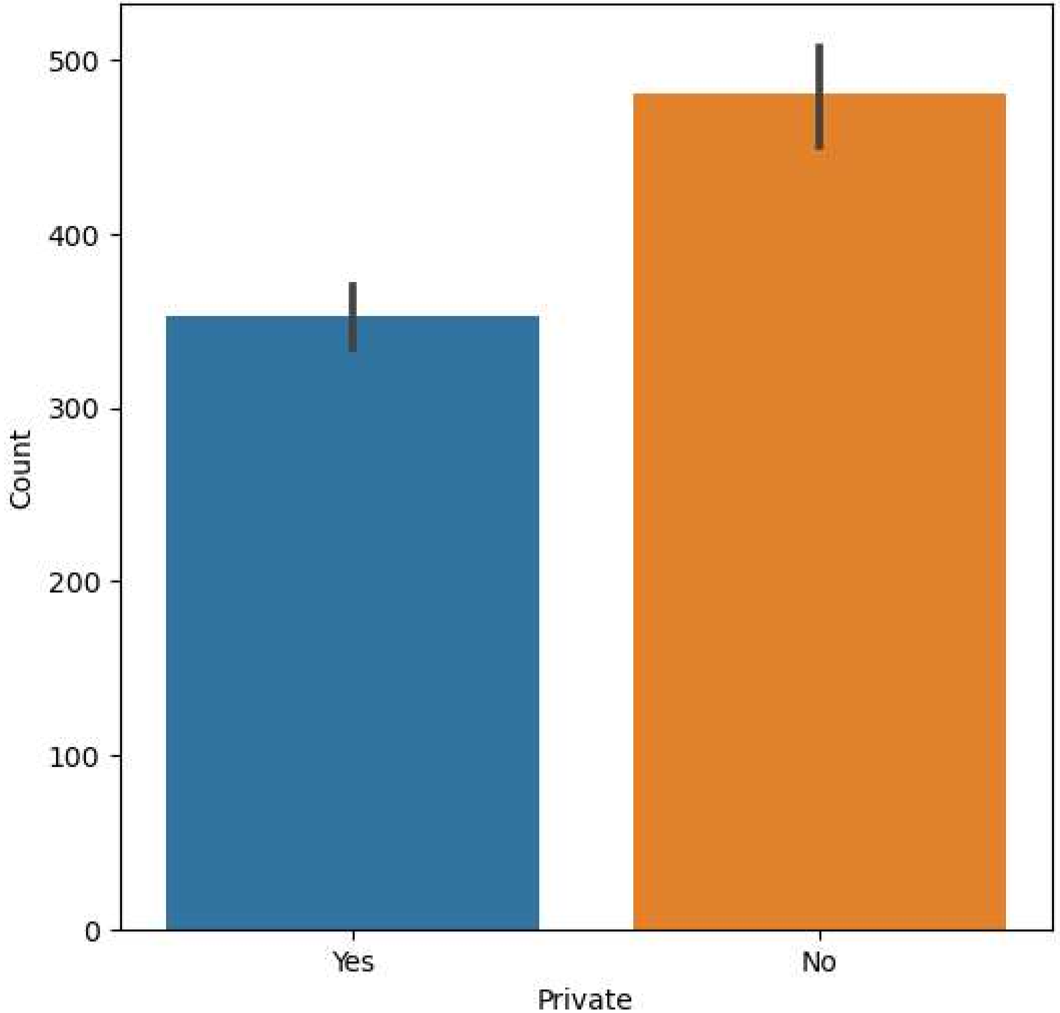
In [24]:
```
plt.figure(figsize=(6,6))
sns.barplot(x=df.index,y=df['Private'])
plt.xlabel("Private")
plt.ylabel("Count")

plt.savefig("comprison.png")
```



In [25]:
```
plt.figure(figsize=(6,6))
sns.barplot(x=df['Private'],y=df.index)
plt.xlabel("Private")
plt.ylabel("Count")

plt.savefig("comprison.png")
```

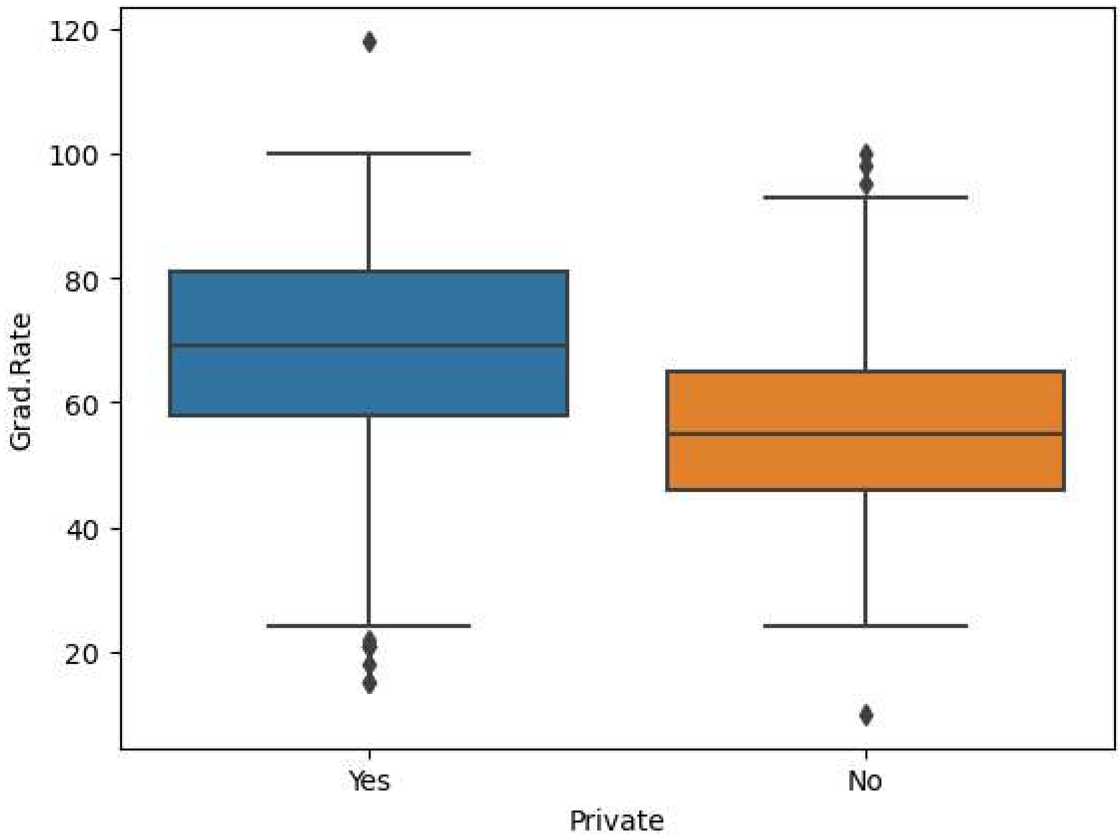


In [26]:
```
df.columns
```

Out[26]:
```
Index(['Unnamed: 0', 'Private', 'Apps', 'Accept', 'Enroll', 'Top10perc',
       'Top25perc', 'F.Undergrad', 'P.Undergrad', 'Outstate', 'Room.Board',
       'Books', 'Personal', 'PhD', 'Terminal', 'S.F.Ratio', 'perc.alumni',
       'Expend', 'Grad.Rate'],
      dtype='object')
```

In [27]:
```
sns.boxplot(x="Private",y="Grad.Rate",data=df)
```

Out[27]:
```
<Axes: xlabel='Private', ylabel='Grad.Rate'>
```



In [28]:
```
df[df['Grad.Rate']>100]['Grad.Rate']
```

Out[28]:
```
95    118
Name: Grad.Rate, dtype: int64
```

In [29]:
```
d1 = {"Grade_Rate":{"collage1":118,"collage2":100},"b":200,"c":300}
```

In [30]:
```
d1
```

Out[30]:
```
{'Grade_Rate': {'collage1': 118, 'collage2': 100}, 'b': 200, 'c': 300}
```

In [31]:
```
# Change value of a to 400
d1["Grade_Rate"]["collage1"]=100
```

In [32]:
```
d1
```

Out[32]:
```
{'Grade_Rate': {'collage1': 100, 'collage2': 100}, 'b': 200, 'c': 300}
```

In [33]:
```
df['Grad.Rate']["Cazenovia Collage"]=100
```

```
C:\Users\vippa\AppData\Local\Temp\ipykernel_16608\1066289043.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#return
ing-a-view-versus-a-copy)
  df['Grad.Rate']["Cazenovia Collage"]=100
```

In [34]:
```
df[df['Grad.Rate']>100]
```

Out[34]:

	Unnamed: 0	Private	Apps	Accept	Enroll	Top10perc	Top25perc	F.Undergrad	P.Undergrad	Outstate	Room.Board	Books	Persona
95	Cazenovia College	Yes	3847	3433	527	9	35	1010	12	9384	4840	600	50

In [35]:
```
df[95] =100
```

In [36]:
```
df[df['Grad.Rate']>100]
```

Out[36]:

	Unnamed: 0	Private	Apps	Accept	Enroll	Top10perc	Top25perc	F.Undergrad	P.Undergrad	Outstate	Room.Board	Books	Persona
95	Cazenovia College	Yes	3847	3433	527	9	35	1010	12	9384	4840	600	50

In [48]:
```
from sklearn.cluster import KMeans
```

In [49]:
```
KMeans =KMeans(n_clusters=2)
```

In [57]:
```
KMeans
```

Out[57]:

```
KMeans
KMeans(n_clusters=2)
```

In [50]:
```
features=df.iloc[:,2:]
features
```

Out[50]:

	Apps	Accept	Enroll	Top10perc	Top25perc	F.Undergrad	P.Undergrad	Outstate	Room.Board	Books	Personal	PhD	Terminal
0	1660	1232	721	23	52	2885	537	7440	3300	450	2200	70	78
1	2186	1924	512	16	29	2683	1227	12280	6450	750	1500	29	30
2	1428	1097	336	22	50	1036	99	11250	3750	400	1165	53	66
3	417	349	137	60	89	510	63	12960	5450	450	875	92	97
4	193	146	55	16	44	249	869	7560	4120	800	1500	76	72
...	...	...	...	...	...	...	...	...	...	...	...	...	...
772	2197	1515	543	4	26	3089	2029	6797	3900	500	1200	60	60
773	1959	1805	695	24	47	2849	1107	11520	4960	600	1250	73	75
774	2097	1915	695	34	61	2793	166	6900	4200	617	781	67	75
775	10705	2453	1317	95	99	5217	83	19840	6510	630	2115	96	96
776	2989	1855	691	28	63	2988	1726	4990	3560	500	1250	75	75

777 rows × 18 columns

In [51]:
```
features.columns = features.columns.astype(str)
```

In [52]:
```
from sklearn.preprocessing import StandardScaler
```

In [53]:
```
scaler = StandardScaler()
scaled_features = scaler.fit_transform(features)
```

In [54]:
```
scaled_features
```

Out[54]:
```
array([[-3.46881819e-01, -3.21205453e-01, -6.35089011e-02, ...,
        -5.01910084e-01, -3.18251941e-01,  0.00000000e+00],
       [-2.10884040e-01, -3.87029908e-02, -2.88584214e-01, ...,
         1.66109850e-01, -5.51261842e-01,  0.00000000e+00],
       [-4.06865631e-01, -3.76317928e-01, -4.78121319e-01, ...,
        -1.77289956e-01, -6.67766793e-01,  0.00000000e+00],
       ...,
       [-2.33895071e-01, -4.23771558e-02, -9.15087008e-02, ...,
        -2.56241250e-01, -9.59029170e-01,  0.00000000e+00],
       [ 1.99171118e+00,  1.77256262e-01,  5.78332661e-01, ...,
         5.88797079e+00,  1.95359460e+00,  0.00000000e+00],
       [-3.26765760e-03, -6.68715889e-02, -9.58163623e-02, ...,
        -9.87115613e-01,  1.95359460e+00,  0.00000000e+00]])
```

In [55]:
```
scaled_features.shape
```

Out[55]:
```
(777, 18)
```

In [59]:
```
df['cluster']=KMeans.fit_predict(scaled_features)
```

```
C:\ProgramData\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1412: FutureWarning: The default value
of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warni
ng
  super()._check_params_vs_input(X, default_n_init=10)
```

In [60]:
```
df
```

4	Pacific University	Yes	193	146	55	16	44	249	869	7560	...	800	1500
...	...	...	...	...	...	...	...	...	...	...	...	...	...
772	Worcester State College	No	2197	1515	543	4	26	3089	2029	6797	...	500	1200
773	Xavier University	Yes	1959	1805	695	24	47	2849	1107	11520	...	600	1250
774	Xavier University of Louisiana	Yes	2097	1915	695	34	61	2793	166	6900	...	617	781
775	Yale University	Yes	10705	2453	1317	95	99	5217	83	19840	...	630	2115
776	York College of Pennsylvania	Yes	2989	1855	691	28	63	2988	1726	4990	...	500	1250

In [61]:
```
KMeans.labels_
```

Out[61]:
```
array([1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1,
       1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1,
       0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1,
       1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0,
       1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1,
       1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1,
       1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0,
       1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1,
       1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0,
       1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1,
       0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1,
       1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1,
       1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0,
       0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0,
       0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1,
       1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1,
       1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1,
       0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1,
       0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0,
       1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0,
       1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1,
       1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0,
       0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0,
       0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1,
       1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0,
       0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0,
       1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0,
       1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1,
       1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1,
       0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0,
       0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1,
       1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0,
       0, 0, 1, 1, 1, 0, 1])
```

In [62]:
```
from sklearn.metrics import confusion_matrix,accuracy_score
```

In [63]:
```
print(confusion_matrix(df['cluster'],KMeans.labels_))
```

```
[[291   0]
 [  0 486]]
```

In [64]:
```
print(accuracy_score(KMeans.labels_,df['cluster']))
```

```
1.0
```

In [65]:
```
features.columns
```

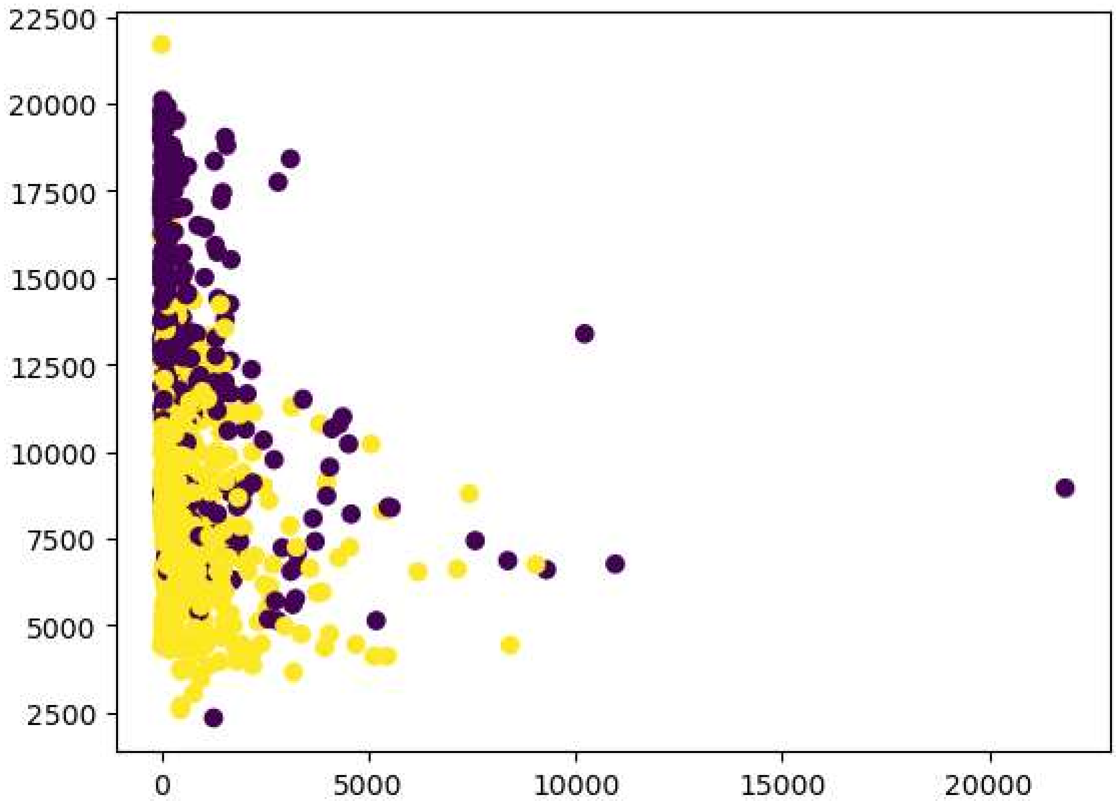
Out[65]:
```
Index(['Apps', 'Accept', 'Enroll', 'Top10perc', 'Top25perc', 'F.Undergrad',
       'P.Undergrad', 'Outstate', 'Room.Board', 'Books', 'Personal', 'PhD',
       'Terminal', 'S.F.Ratio', 'perc.alumni', 'Expend', 'Grad.Rate', '95'],
      dtype='object')
```

In [66]:
```
features['P.Undergrad']
```

Out[66]:
```
0       537
1      1227
2        99
3        63
4       869
       ...
772    2029
773    1107
774     166
775      83
776    1726
Name: P.Undergrad, Length: 777, dtype: int64
```

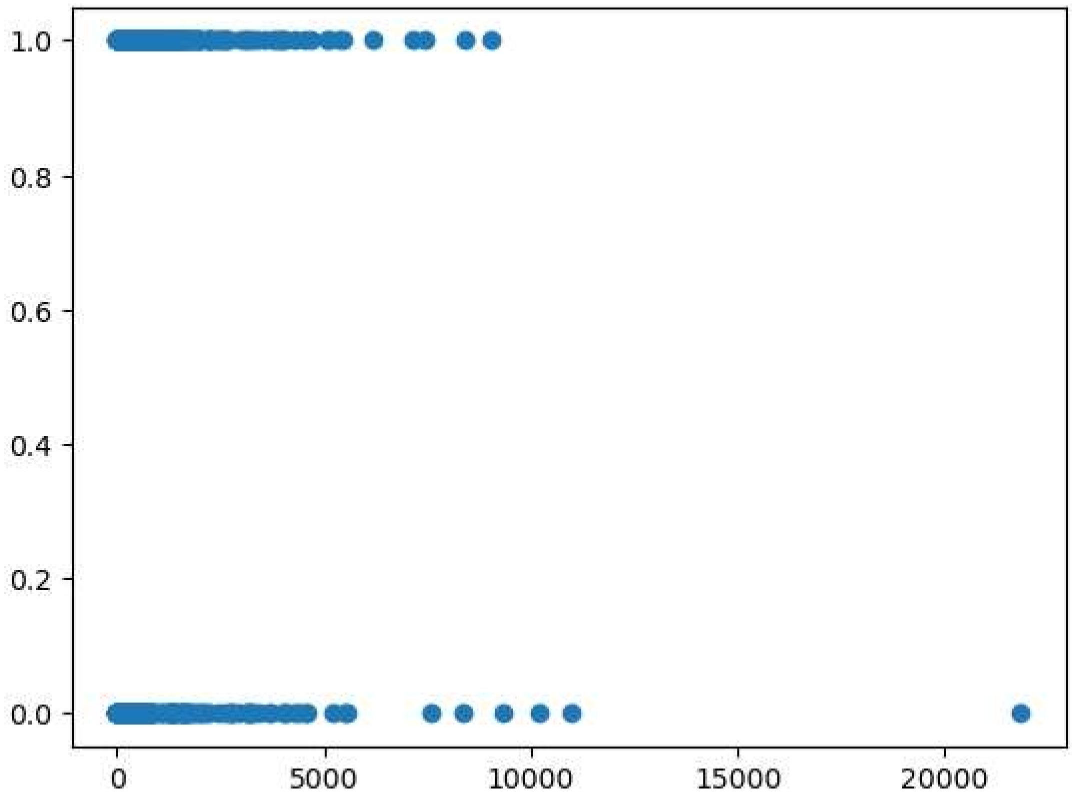
In [68]:
```
plt.scatter(features["P.Undergrad"],features['Outstate'],c=KMeans.labels_)
```

Out[68]: <matplotlib.collections.PathCollection at 0x1fb113da010>



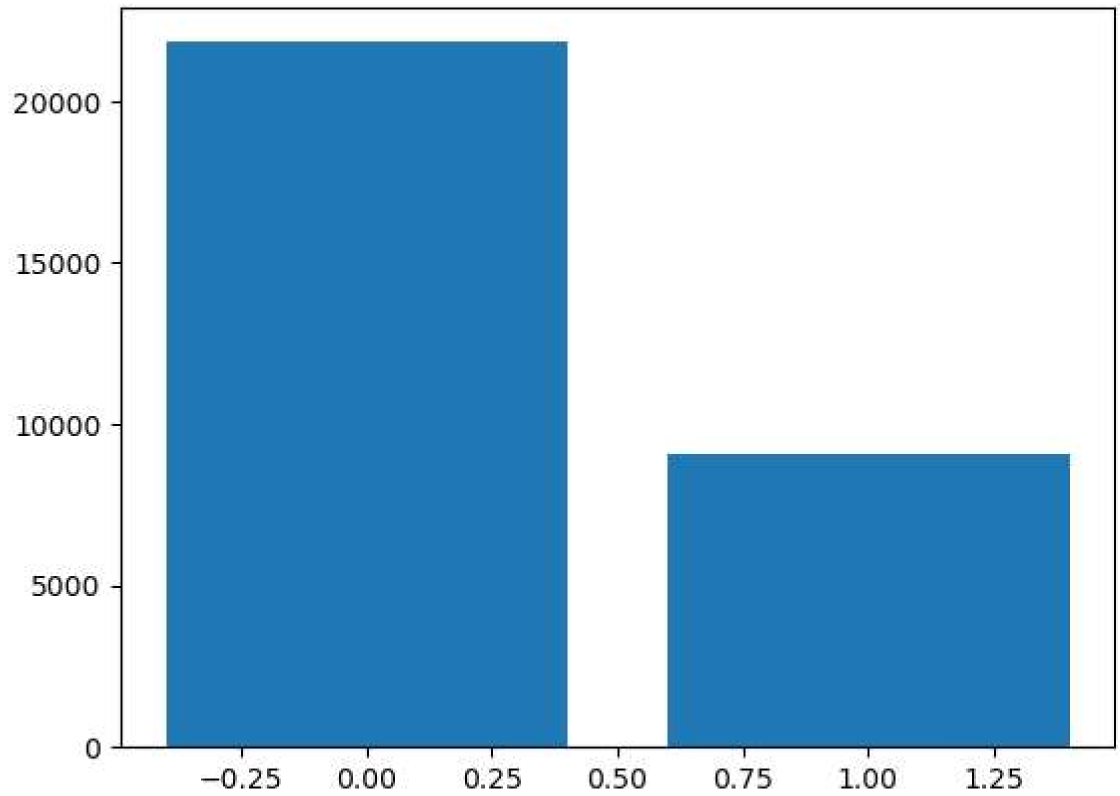
In [69]:
```
plt.scatter(features["P.Undergrad"],KMeans.labels_)
```

Out[69]: <matplotlib.collections.PathCollection at 0x1fb12572610>



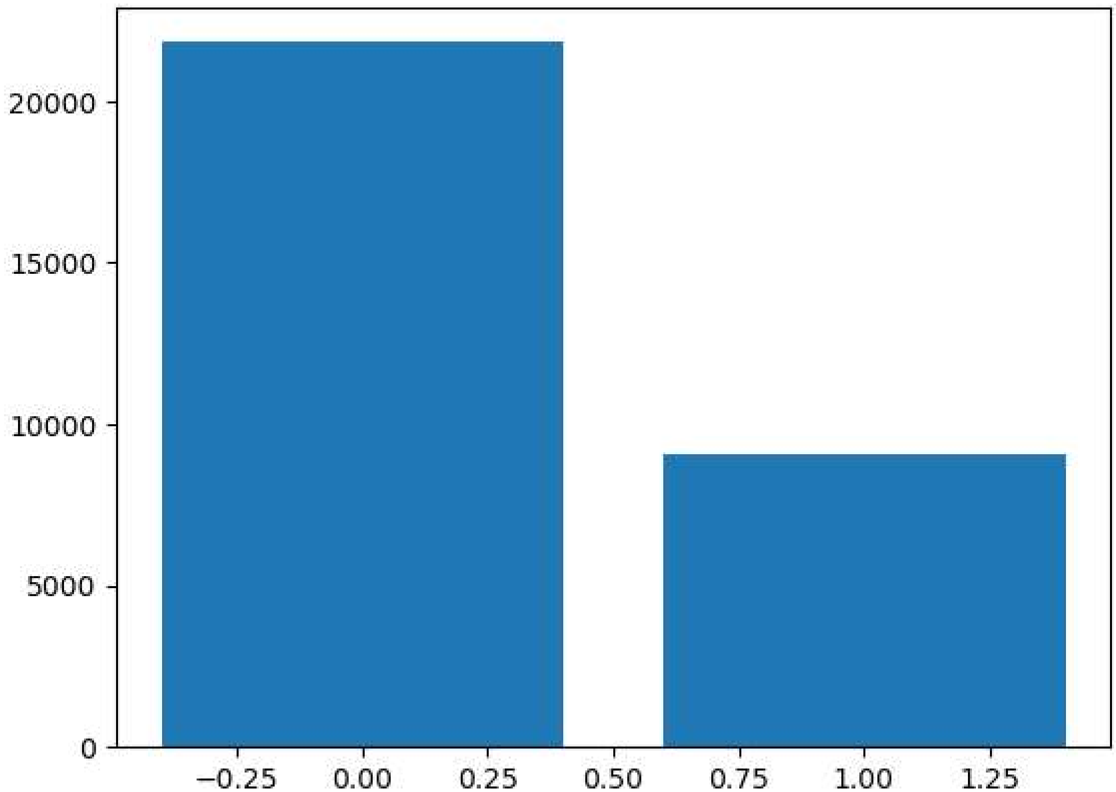
In [70]:
```
plt.bar(KMeans.labels_,features["P.Undergrad"])
```

Out[70]: <BarContainer object of 777 artists>



In [71]:
```
plt.bar(df['cluster'],features["P.Undergrad"])
```

Out[71]: <BarContainer object of 777 artists>



In [72]:
```
# Diff between KNN and K means Clustering
# 1.) KNN is used for classification and regression
#     K means is for Clustering problems

# 2.) KNN is supervised algorithm
#     K means is unsupervised algorithm

# 3.) To training KNN, we need a dataset with all the
#data points having class labels
# For training K means, we no need any such information

# 4.) We use KNN to predict the class label or new points
# we use K means to find patterns in a given dataset by grouping datapoints
# into clusters
```

In [74]:
```
cd=pd.read_csv(r"C:\Users\vippa\Downloads\Classified Data.unknown")
```

In [75]:
```
cd
```

Out[75]:

	Unnamed: 0	WTT	PTI	EQW	SBI	LQE	QWG	FDJ	PJF	HQE	NXJ	TARGET CLASS
0	0	0.913917	1.162073	0.567946	0.755464	0.780862	0.352608	0.759697	0.643798	0.879422	1.231409	1
1	1	0.635632	1.003722	0.535342	0.825645	0.924109	0.648450	0.675334	1.013546	0.621552	1.492702	0
2	2	0.721360	1.201493	0.921990	0.855595	1.526629	0.720781	1.626351	1.154483	0.957877	1.285597	0
3	3	1.234204	1.386726	0.653046	0.825624	1.142504	0.875128	1.409708	1.380003	1.522692	1.153093	1
4	4	1.279491	0.949750	0.627280	0.668976	1.232537	0.703727	1.115596	0.646691	1.463812	1.419167	1
...	...	...	...	...	...	...	...	...	...	...	...	...
995	995	1.010953	1.034006	0.853116	0.622460	1.036610	0.586240	0.746811	0.319752	1.117340	1.348517	1
996	996	0.575529	0.955786	0.941835	0.792882	1.414277	1.269540	1.055928	0.713193	0.958684	1.663489	0
997	997	1.135470	0.982462	0.781905	0.916738	0.901031	0.884738	0.386802	0.389584	0.919191	1.385504	1
998	998	1.084894	0.861769	0.407158	0.665696	1.608612	0.943859	0.855806	1.061338	1.277456	1.188063	1
999	999	0.837460	0.961184	0.417006	0.799784	0.934399	0.424762	0.778234	0.907962	1.257190	1.364837	1

1000 rows × 12 columns

In [76]:
```
cd.head()
```

Out[76]:

	Unnamed: 0	WTT	PTI	EQW	SBI	LQE	QWG	FDJ	PJF	HQE	NXJ	TARGET CLASS
0	0	0.913917	1.162073	0.567946	0.755464	0.780862	0.352608	0.759697	0.643798	0.879422	1.231409	1
1	1	0.635632	1.003722	0.535342	0.825645	0.924109	0.648450	0.675334	1.013546	0.621552	1.492702	0
2	2	0.721360	1.201493	0.921990	0.855595	1.526629	0.720781	1.626351	1.154483	0.957877	1.285597	0
3	3	1.234204	1.386726	0.653046	0.825624	1.142504	0.875128	1.409708	1.380003	1.522692	1.153093	1
4	4	1.279491	0.949750	0.627280	0.668976	1.232537	0.703727	1.115596	0.646691	1.463812	1.419167	1

In [78]:
```
#project :5
cd=pd.read_csv(r"C:\Users\vippa\Downloads\Classified Data.unknown",index_col=0)
cd
```

Out[78]:

	WTT	PTI	EQW	SBI	LQE	QWG	FDJ	PJF	HQE	NXJ	TARGET CLASS
0	0.913917	1.162073	0.567946	0.755464	0.780862	0.352608	0.759697	0.643798	0.879422	1.231409	1
1	0.635632	1.003722	0.535342	0.825645	0.924109	0.648450	0.675334	1.013546	0.621552	1.492702	0
2	0.721360	1.201493	0.921990	0.855595	1.526629	0.720781	1.626351	1.154483	0.957877	1.285597	0
3	1.234204	1.386726	0.653046	0.825624	1.142504	0.875128	1.409708	1.380003	1.522692	1.153093	1
4	1.279491	0.949750	0.627280	0.668976	1.232537	0.703727	1.115596	0.646691	1.463812	1.419167	1
...	...	...	...	...	...	...	...	...	...	...	...
995	1.010953	1.034006	0.853116	0.622460	1.036610	0.586240	0.746811	0.319752	1.117340	1.348517	1
996	0.575529	0.955786	0.941835	0.792882	1.414277	1.269540	1.055928	0.713193	0.958684	1.663489	0
997	1.135470	0.982462	0.781905	0.916738	0.901031	0.884738	0.386802	0.389584	0.919191	1.385504	1
998	1.084894	0.861769	0.407158	0.665696	1.608612	0.943859	0.855806	1.061338	1.277456	1.188063	1
999	0.837460	0.961184	0.417006	0.799784	0.934399	0.424762	0.778234	0.907962	1.257190	1.364837	1

1000 rows × 11 columns

In [80]:
```
from sklearn.preprocessing import StandardScaler
scaler= StandardScaler()
scaler.fit(cd.drop('TARGET CLASS',axis=1))
```

Out[80]:
```
▾ StandardScaler

StandardScaler()
```

In [81]:
```
scaled_features = scaler.transform(cd.drop('TARGET CLASS',axis=1))
scaled_features
```

Out[81]:
```
array([[-0.12354188,  0.18590747, -0.91343069, ..., -1.48236813,
        -0.9497194 , -0.64331425],
       [-1.08483602, -0.43034845, -1.02531333, ..., -0.20224031,
        -1.82805088,  0.63675862],
       [-0.78870217,  0.33931821,  0.30151137, ...,  0.28570652,
        -0.68249379, -0.37784986],
       ...,
       [ 0.64177714, -0.51308341, -0.17920486, ..., -2.36249443,
        -0.81426092,  0.11159651],
       [ 0.46707241, -0.98278576, -1.46519359, ..., -0.03677699,
         0.40602453, -0.85567   ],
       [-0.38765353, -0.59589427, -1.4313981 , ..., -0.56778932,
         0.3369971 ,  0.01034996]])
```

In [82]:
```
cd_feat=pd.DataFrame(scaled_features)
cd_feat
```

Out[82]:

	0	1	2	3	4	5	6	7	8	9
0	-0.123542	0.185907	-0.913431	0.319629	-1.033637	-2.308375	-0.798951	-1.482368	-0.949719	-0.643314
1	-1.084836	-0.430348	-1.025313	0.625388	-0.444847	-1.152706	-1.129797	-0.202240	-1.828051	0.636759
2	-0.788702	0.339318	0.301511	0.755873	2.031693	-0.870156	2.599818	0.285707	-0.682494	-0.377850
3	0.982841	1.060193	-0.621399	0.625299	0.452820	-0.267220	1.750208	1.066491	1.241325	-1.026987
4	1.139275	-0.640392	-0.709819	-0.057175	0.822886	-0.936773	0.596782	-1.472352	1.040772	0.276510
...	...	...	...	...	...	...	...	...	...	...
995	0.211653	-0.312490	0.065163	-0.259834	0.017567	-1.395721	-0.849486	-2.604264	-0.139347	-0.069602
996	-1.292453	-0.616901	0.369613	0.482648	1.569891	1.273495	0.362784	-1.242110	-0.679746	1.473448
997	0.641777	-0.513083	-0.179205	1.022255	-0.539703	-0.229680	-2.261339	-2.362494	-0.814261	0.111597
998	0.467072	-0.982786	-1.465194	-0.071465	2.368666	0.001269	-0.422041	-0.036777	0.406025	-0.855670
999	-0.387654	-0.595894	-1.431398	0.512722	-0.402552	-2.026512	-0.726253	-0.567789	0.336997	0.010350

1000 rows × 10 columns

In [83]:
```
cd_feat.head()
```

Out[83]:

	0	1	2	3	4	5	6	7	8	9
0	-0.123542	0.185907	-0.913431	0.319629	-1.033637	-2.308375	-0.798951	-1.482368	-0.949719	-0.643314
1	-1.084836	-0.430348	-1.025313	0.625388	-0.444847	-1.152706	-1.129797	-0.202240	-1.828051	0.636759
2	-0.788702	0.339318	0.301511	0.755873	2.031693	-0.870156	2.599818	0.285707	-0.682494	-0.377850
3	0.982841	1.060193	-0.621399	0.625299	0.452820	-0.267220	1.750208	1.066491	1.241325	-1.026987
4	1.139275	-0.640392	-0.709819	-0.057175	0.822886	-0.936773	0.596782	-1.472352	1.040772	0.276510

In [84]:
```
#Example of standard scalar
data=np.array([[0,0],[0,1],[1,0],[1,1]])
data
```

Out[84]:
```
array([[0, 0],
       [0, 1],
       [1, 0],
       [1, 1]])
```

In [85]:
```
scl=StandardScaler()
scl
```

Out[85]:
```
▾ StandardScaler
StandardScaler()
```

In [88]:
```
scl_data=scl.fit_transform(data)
scl_data
```

Out[88]:
```
array([[-1., -1.],
       [-1.,  1.],
       [ 1., -1.],
       [ 1.,  1.]])
```

In [89]:
```
scl_data.mean()
```

Out[89]:
```
0.0
```

In [90]:
```
scl_data.std()
```

Out[90]:
```
1.0
```

In [94]:
```
cd.head()#original data
```

Out[94]:

	WTT	PTI	EQW	SBI	LQE	QWG	FDJ	PJF	HQE	NXJ	TARGET CLASS
0	0.913917	1.162073	0.567946	0.755464	0.780862	0.352608	0.759697	0.643798	0.879422	1.231409	1
1	0.635632	1.003722	0.535342	0.825645	0.924109	0.648450	0.675334	1.013546	0.621552	1.492702	0
2	0.721360	1.201493	0.921990	0.855595	1.526629	0.720781	1.626351	1.154483	0.957877	1.285597	0
3	1.234204	1.386726	0.653046	0.825624	1.142504	0.875128	1.409708	1.380003	1.522692	1.153093	1
4	1.279491	0.949750	0.627280	0.668976	1.232537	0.703727	1.115596	0.646691	1.463812	1.419167	1

In [95]:
```
cd_feat.head()#scaled data
```

Out[95]:

	0	1	2	3	4	5	6	7	8	9
0	-0.123542	0.185907	-0.913431	0.319629	-1.033637	-2.308375	-0.798951	-1.482368	-0.949719	-0.643314
1	-1.084836	-0.430348	-1.025313	0.625388	-0.444847	-1.152706	-1.129797	-0.202240	-1.828051	0.636759
2	-0.788702	0.339318	0.301511	0.755873	2.031693	-0.870156	2.599818	0.285707	-0.682494	-0.377850
3	0.982841	1.060193	-0.621399	0.625299	0.452820	-0.267220	1.750208	1.066491	1.241325	-1.026987
4	1.139275	-0.640392	-0.709819	-0.057175	0.822886	-0.936773	0.596782	-1.472352	1.040772	0.276510

In [98]:
```
# to name this columns
cd_feat=pd.DataFrame(scaled_features,columns=cd.columns[:-1])
cd_feat
```

Out[98]:

	WTT	PTI	EQW	SBI	LQE	QWG	FDJ	PJF	HQE	NXJ
0	-0.123542	0.185907	-0.913431	0.319629	-1.033637	-2.308375	-0.798951	-1.482368	-0.949719	-0.643314
1	-1.084836	-0.430348	-1.025313	0.625388	-0.444847	-1.152706	-1.129797	-0.202240	-1.828051	0.636759
2	-0.788702	0.339318	0.301511	0.755873	2.031693	-0.870156	2.599818	0.285707	-0.682494	-0.377850
3	0.982841	1.060193	-0.621399	0.625299	0.452820	-0.267220	1.750208	1.066491	1.241325	-1.026987
4	1.139275	-0.640392	-0.709819	-0.057175	0.822886	-0.936773	0.596782	-1.472352	1.040772	0.276510
...	...	...	...	...	...	...	...	...	...	...
995	0.211653	-0.312490	0.065163	-0.259834	0.017567	-1.395721	-0.849486	-2.604264	-0.139347	-0.069602
996	-1.292453	-0.616901	0.369613	0.482648	1.569891	1.273495	0.362784	-1.242110	-0.679746	1.473448
997	0.641777	-0.513083	-0.179205	1.022255	-0.539703	-0.229680	-2.261339	-2.362494	-0.814261	0.111597
998	0.467072	-0.982786	-1.465194	-0.071465	2.368666	0.001269	-0.422041	-0.036777	0.406025	-0.855670
999	-0.387654	-0.595894	-1.431398	0.512722	-0.402552	-2.026512	-0.726253	-0.567789	0.336997	0.010350

1000 rows × 10 columns

In [99]:
```
cd_feat.head()
```

Out[99]:

	WTT	PTI	EQW	SBI	LQE	QWG	FDJ	PJF	HQE	NXJ
0	-0.123542	0.185907	-0.913431	0.319629	-1.033637	-2.308375	-0.798951	-1.482368	-0.949719	-0.643314
1	-1.084836	-0.430348	-1.025313	0.625388	-0.444847	-1.152706	-1.129797	-0.202240	-1.828051	0.636759
2	-0.788702	0.339318	0.301511	0.755873	2.031693	-0.870156	2.599818	0.285707	-0.682494	-0.377850
3	0.982841	1.060193	-0.621399	0.625299	0.452820	-0.267220	1.750208	1.066491	1.241325	-1.026987
4	1.139275	-0.640392	-0.709819	-0.057175	0.822886	-0.936773	0.596782	-1.472352	1.040772	0.276510

In [100]:
```
cd_feat.isna().sum()
```

Out[100]:
```
WTT    0
PTI    0
EQW    0
SBI    0
LQE    0
QWG    0
FDJ    0
PJF    0
HQE    0
NXJ    0
dtype: int64
```

In [101]:
```
from sklearn.model_selection import train_test_split
x=cd_feat
y=cd['TARGET CLASS']
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3)
```

In [102]:
```
x.shape
```

Out[102]:
```
(1000, 10)
```

In [103]:
```
x_train.shape
```

Out[103]:
```
(700, 10)
```

In [104]:
```
x_test.shape
```

Out[104]:
```
(300, 10)
```

In [92]:
```
from sklearn.neighbors import KNeighborsClassifier
```

In [93]:
```
KNN=KNeighborsClassifier(n_neighbors=3)
```

In [105]:
```
# To train model
KNN.fit(x_train,y_train)
```

Out[105]:
```
▾ KNeighborsClassifier
KNeighborsClassifier(n_neighbors=3)
```

In [106]:
```
pred=KNN.predict(x_test)# to predict
pred
```

Out[106]:
```
array([1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1,
       0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0,
       0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1,
       0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1,
       0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0,
       0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0,
       1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1,
       1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0,
       1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1,
       1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1,
       1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1,
       0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0,
       0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1,
       0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0], dtype=int64)
```

In [107]:
```
from sklearn.metrics import accuracy_score
```

In [108]:
```
acc=accuracy_score(pred,y_test)
acc
```

Out[108]:
```
0.9266666666666666
```

In [109]:

```
# To find the error rate
error_rate = []
for val in range(1,30):
    knn=KNeighborsClassifier(n_neighbors=val)
    knn.fit(x_train, y_train)
    pred_i = knn.predict(x_test)
    error_rate.append(np.mean(pred_i != y_test))
error_rate
```

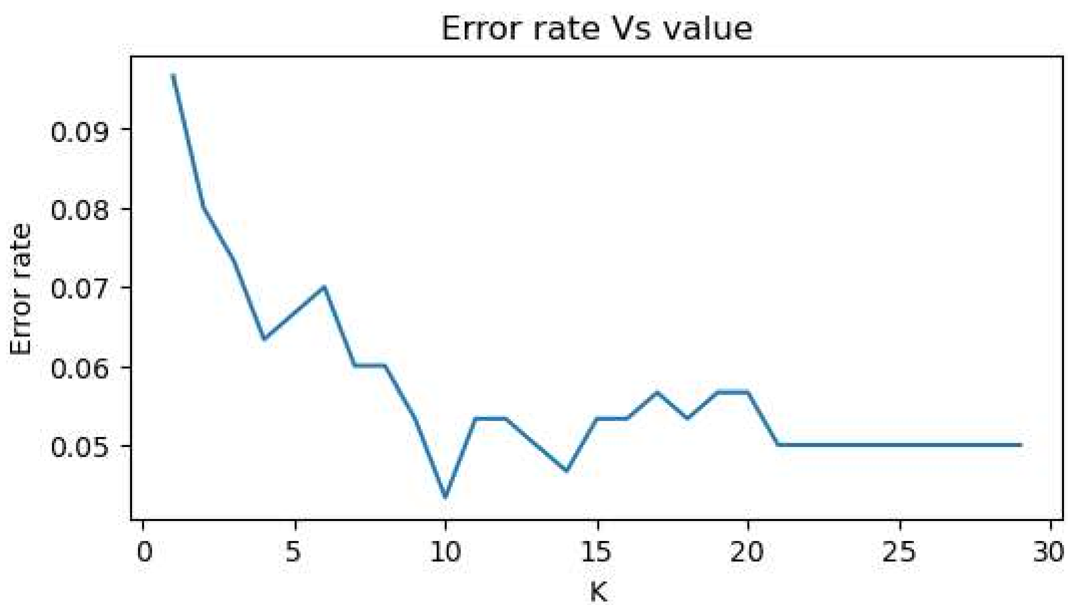
Out[109]:

```
[0.09666666666666666,
 0.08,
 0.07333333333333333,
 0.06333333333333334,
 0.06666666666666667,
 0.07,
 0.06,
 0.06,
 0.05333333333333334,
 0.043333333333333335,
 0.05333333333333334,
 0.05333333333333334,
 0.05,
 0.04666666666666667,
 0.05333333333333334,
 0.05333333333333334,
 0.056666666666666664,
 0.05333333333333334,
 0.056666666666666664,
 0.056666666666666664,
 0.05,
 0.05,
 0.05,
 0.05,
 0.05,
 0.05,
 0.05,
 0.05,
 0.05]
```

In [110]:
```
import matplotlib.pyplot as plt
```

In [111]:
```
plt.figure(figsize=(6,3))
plt.plot(range(1,30),error_rate)
plt.title("Error rate Vs value")
plt.xlabel("K")
plt.ylabel("Error rate")
```

Out[111]: Text(0, 0.5, 'Error rate')



In [112]:
```
#project :6
df=pd.read_csv(r"C:\Users\vippa\Downloads\cancerKNNAlgorithmDataset.csv")
```

In [113]:
```
df.head()
```

Out[113]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_me
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.30
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.24
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19

5 rows × 33 columns

In [ ]: