In [1]:
```
# Unsupervised learning algorithm
# K Means Clustering
# K Means Clustering is an unsupervised learning algorithm that will attempt to group si

# It is mainly used in:
# Clustering similar documents
# Clustering customers based on similar features
```

In [2]:
```
from sklearn.datasets import make_blobs
# The make_blobs function is used to generate Synthetic datasets for clustering and clas
# This function will create clusters of data points with Gaussian Distribution.
```

In [3]:
```
# Creating random dataset
data = make_blobs(n_samples=200, n_features=2, centers=3, cluster_std=5.6, random_state=
# n_samples = Total number of points equally divided among the clusters
# n_features = It indicates the number of features(columns)
# centers = It determine number of clusters to be generated
# clusted_std = It sets the standard deviation of clusters. High value makes the cluster
data
```

Out[3]:
```
(array([[ -5.24981605,  -4.90754635],
        [-11.62061369,   3.19390403],
        [ -1.08265038,  -1.82412171],
        [ -7.41638   ,   9.71532652],
        [-16.74681638, -15.71394126],
        [ -3.99146763,   2.03202524],
        [  9.34655477,   8.52472571],
        [ -5.23430305,   2.51542897],
        [-15.19614755, -15.10875788],
        [  1.38466629,  -2.83633178],
        [ -2.98477661,   5.864642  ],
        [  3.97423306,  -0.37482931],
        [  3.92208342,  -3.65520771],
        [  8.97021315,  10.85677608],
        [  6.75283137,  10.58839336],
        [  3.90517983,   3.25937101],
        [ -9.93058674,  -2.65759501],
        [  5.71213823, -10.57743709],
        [-24.75768521,   1.95678094],
        [ -7.50669603,  -5.47042511],
        [  6.80118781,   5.61107482],
        [ -4.938591  , -15.90777285],
        [ -5.10748325,  -4.92195377],
        [  2.83599543,   1.00386038],
        [  0.3807653 ,  -2.67450522],
        [  3.96319993,   3.42341868],
        [-10.97376961,   5.55803526],
        [-14.21773092,   2.40620013],
        [-11.02795336,   5.13745236],
        [  3.61608096,  -1.34759405],
        [ -5.53399286,   8.65283337],
        [ -5.30107418,  -2.74067061],
        [ -4.8569233 ,  11.84539091],
        [-10.99577267,  -0.61838855],
        [-10.58295234,   7.39677816],
        [ 13.77988294,  -0.67404326],
        [ -0.69598527,  12.17248613],
        [ -3.52719179,  12.28354958],
        [ -7.22713521, -10.10040402],
        [  7.40730314,   4.81358077],
        [-14.87031384, -15.09463097],
        [-18.10867365,  -4.13215051],
        [  6.07348381,   0.53640347],
        [-14.89247183, -12.47673022],
        [  1.97809804,   1.96236487],
        [ -1.78053203,   2.70323526],
        [  1.23394721,   6.07831595],
        [ 10.93082725,  -1.60065119],
        [  4.36438311,  -5.43590767],
        [  2.60960459,   3.02606826],
        [ -0.54996606,  -1.36348297],
        [ -7.43014736,  -6.03274459],
        [-14.93406105,   5.92714249],
        [ 12.1601229 ,   7.65616321],
        [ 14.92138932,   5.2410015 ],
        [ -0.58222196,   4.13388067],
        [  2.94498958,   2.69820205],
        [  4.41161975,   9.28778447],
        [ -2.38410135,  -7.70155751],
        [-12.68066018,   0.20098592],
        [ -8.43366456, -10.07967714],
        [-14.62846253,  -1.82690505],
        [-10.16918984, -14.04608059],
        [ -4.45194252, -16.86942771],
```

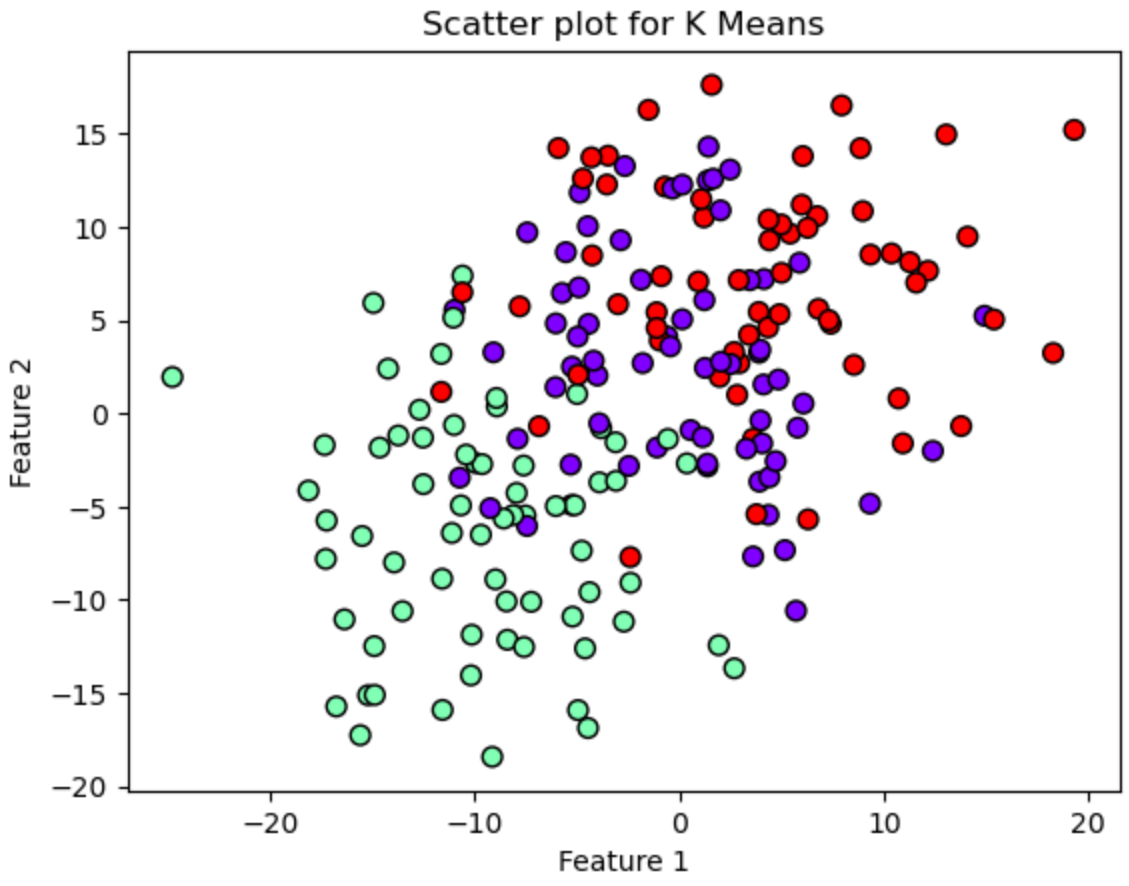
```
[  9.32593978,  -4.83462408],
[-16.34995533, -11.04076346],
[  2.68687897,   3.29981864],
[ -4.37469066,  -9.58185815],
[-17.22302221,  -5.74364264],
[-12.49517084,  -3.78789593],
[ -0.32595802,  12.06317859],
[ -8.10333994,  -5.44519625],
[-10.58935781,   6.4991462 ],
[  4.41050545,  -3.41995937],
[  3.89364577,   5.44477282],
[ -4.98339583,   1.0448202 ],
[ 10.37684638,   8.5854883 ],
[ -6.8437892 ,  -0.68125892],
[ -4.42125855,   4.80675769],
[  4.05936075,  -1.60076217],
[  1.93493197, -12.42917384],
[ -3.89338754,  -3.69210359],
[ -2.70074891, -11.16416059],
[  3.28932812,  -1.88505125],
[ 12.39511264,  -2.00409786],
[  0.15347386,  12.27697268],
[-13.51552696, -10.60218146],
[ -9.07041593,   3.2836011 ],
[  4.71454244,  -2.55678919],
[  3.61431741,  -7.66619609],
[ -6.02091464,  -4.96531814],
[  5.42499657,   9.64552807],
[  4.14274459,   7.21005216],
[ -3.08177872,  -3.61172116],
[ -5.89825096,  14.23819369],
[  8.8604806 ,  14.23310739],
[ 15.38218867,   5.04373595],
[  5.00255096,   7.54824385],
[  5.88638389,   8.08989931],
[  0.56014823,  -0.88855852],
[  1.15132247,  -1.27155728],
[ -1.86250456,   7.17179344],
[ -1.4946331 ,  16.28725818],
[  2.5259099 ,   2.65369957],
[ -5.20743544, -10.89299448],
[  4.99402708,  10.14242201],
[ -9.62333264,  -2.7144797 ],
[-17.26093767,  -7.79770689],
[  1.39642076,  12.49439275],
[  5.98802375,  11.20185662],
[ -0.9489662 ,   3.91186589],
[ -8.5700148 ,  -5.63079271],
[ -8.97398467,  -8.89463713],
[  1.20865205,  10.51883034],
[ -3.46985401,  13.82763337],
[  1.58951527,  17.62828863],
[ 11.58579716,   7.01937378],
[ -6.02140355,   4.83202802],
[ 19.30159377,  15.21093982],
[  1.66148352,  12.59460454],
[-10.66185214,  -4.92991414],
[  2.4897423 ,  13.08385551],
[-13.71972389,  -1.18069152],
[  6.30073365,  -5.67813819],
[  2.02289907,  10.89820012],
[  5.16729587,  -7.32786437],
[ -7.78881273,   5.744719  ],
[ 11.27565909,   8.11816138],
```

```
[  8.54195454,   2.60818957],
[ -0.86913535,   7.36274223],
[ -0.42713548,   3.60030766],
[ -3.10241438,  -1.54190753],
[-11.55909508, -15.89949828],
[ -4.28430809,  13.73659972],
[ -8.39183314, -12.13606931],
[  3.4188036 ,   4.21882032],
[-13.92643723,  -7.98905142],
[ -4.46112527,  10.04849802],
[-10.4052321 ,  -2.20078447],
[-11.10175564,  -6.40637461],
[ -7.59458492,  -2.79244983],
[ -1.10096395,   5.43355988],
[  6.27786082,   9.95680828],
[  2.69584278, -13.66611186],
[  3.45708123,   7.1522893 ],
[  1.43410687,  14.31008042],
[ 18.27248416,   3.24764627],
[ -1.09755957,   4.58636425],
[ -4.70029128,  12.60473604],
[  7.31828097,   5.01116956],
[ -9.22610866,  -5.07771699],
[-11.61866126,   1.16108127],
[-10.71861603,  -3.44120446],
[ 10.72202398,   0.80719218],
[-12.51542109,  -1.294115  ],
[ -4.5943801 , -12.5984737 ],
[-15.48477782,  -6.57155466],
[ -2.85608307,   9.29444593],
[ -2.37505481,  -9.07779765],
[  1.26263852,   2.44656215],
[  1.37586988,  -2.69058109],
[ -4.93185963,   2.07972363],
[  4.12216919,   1.55954243],
[ 13.05200063,  14.9667644 ],
[ -2.45580997,  -2.80943859],
[ -4.25177743,   8.47923093],
[  4.34719254,   4.61440727],
[  0.15112935,   5.0523764 ],
[  1.08050443,  11.49448065],
[  4.85112567,   1.82192592],
[  6.04304587,  13.80824563],
[  0.9340018 ,   7.07984909],
[ -4.9551033 ,   4.12796096],
[  2.91444877,   7.15689858],
[ -4.75813015,  -7.35598513],
[  4.90525112,   5.33108154],
[-10.13264796, -11.86486037],
[ -3.80780104,  -0.79083806],
[  7.93769593,  16.52341902],
[  2.03950902,   2.77514637],
[  5.79200276,  -0.76078498],
[  3.78124833,  -5.3983936 ],
[ -5.72266764,   6.46677918],
[-15.5775964 , -17.25441627],
[ -2.65386239,  13.27262508],
[ -7.577668  , -12.52264068],
[ -8.90619876,   0.37898376],
[ -3.90184935,  -0.52659188],
[ -9.13001131, -18.41420022],
[ -7.93975751,  -4.25911797],
[  4.37275142,  10.40593894],
[ 14.09335575,   9.49053914],
```

```
        [-11.57676101,  -8.86175039],
        [ -4.89815586,   6.76166819],
        [-17.32161928,  -1.69248758],
        [ -8.9226025 ,   0.83203907],
        [ -9.68630376,  -6.50000933],
        [ -6.03723717,   1.41540402],
        [ -4.17807591,   2.83322077],
        [ -7.89010596,  -1.35358227]]),
 array([2, 1, 0, 0, 1, 0, 2, 0, 1, 0, 2, 0, 0, 2, 2, 0, 1, 0, 1, 1, 2, 1,
        1, 2, 1, 0, 0, 1, 1, 2, 0, 0, 0, 1, 1, 2, 2, 2, 1, 2, 1, 1, 0, 1,
        2, 0, 0, 2, 0, 2, 1, 0, 1, 2, 0, 0, 2, 2, 2, 1, 1, 1, 1, 1, 0, 1,
        2, 1, 1, 1, 0, 1, 2, 0, 2, 1, 2, 2, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0,
        0, 0, 1, 2, 0, 1, 2, 2, 2, 2, 0, 0, 0, 0, 2, 0, 1, 2, 1, 1, 0, 2,
        2, 1, 1, 2, 2, 2, 2, 0, 2, 0, 1, 0, 1, 2, 0, 0, 2, 2, 2, 2, 0, 1,
        1, 2, 1, 2, 1, 0, 1, 1, 1, 2, 2, 1, 0, 0, 2, 2, 2, 2, 0, 2, 0, 2,
        1, 1, 1, 0, 1, 0, 0, 2, 0, 2, 0, 2, 2, 0, 2, 0, 2, 2, 0, 2, 1, 2,
        1, 1, 2, 0, 0, 2, 0, 1, 0, 1, 1, 0, 1, 1, 2, 2, 1, 0, 1, 1, 1, 0,
        0, 0]))
```

In [4]:
```
import matplotlib.pyplot as plt
x,y = data
plt.scatter(x[:, 0], x[:, 1], c=y, cmap="rainbow", edgecolor="black", s=50)
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.title("Scatter plot for K Means")
```

Out[4]:
```
Text(0.5, 1.0, 'Scatter plot for K Means')
```



In [5]:
```
data[0].shape
```

Out[5]:
```
(200, 2)
```

In [6]:
```
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=4)
kmeans.fit(data[0])
```

```
C:\ProgramData\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1412: FutureWarnin
g: 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 warning
  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 th
an available threads. You can avoid it by setting the environment variable OMP_NUM_THREA
DS=1.
  warnings.warn(
```

Out[6]:

▼ KMeans

KMeans(n_clusters=4)

In [7]:

```
kmeans.cluster_centers_
```

Out[7]:

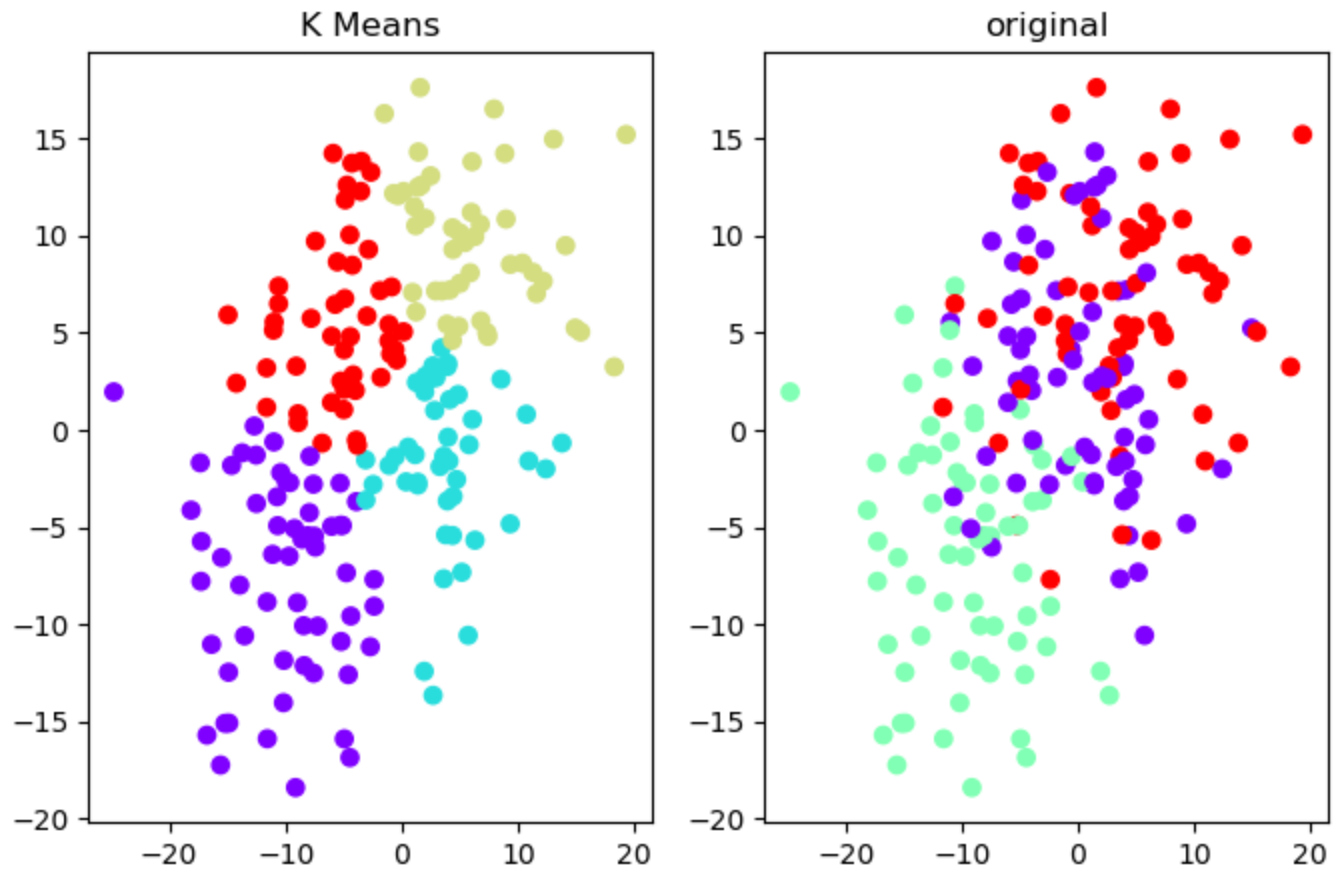
```
array([[-10.15568421,  -7.59115373],
       [  3.75744583,  -1.6588095 ],
       [  6.13969746,   9.71742729],
       [ -5.53321259,   5.67200087]])
```

In [8]:

```
fig, (ax1,ax2) = plt.subplots(1,2,figsize=(8,5))
ax1.set_title('K Means')
ax1.scatter(data[0][:,0], data[0][:,1], c=kmeans.labels_, cmap="rainbow")
ax2.set_title("original")
ax2.scatter(data[0][:,0], data[0][:,1], c=data[1], cmap="rainbow")
```

Out[8]:

```
<matplotlib.collections.PathCollection at 0x1bc1397a350>
```



In [9]:

```
# Project-4
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
df = pd.read_csv("College_Data")
```

Out[9]:

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

In [10]:
```
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 [11]:
```
df.isna().sum()
```

Out[11]:
```
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 [12]:
```
df.duplicated()
```

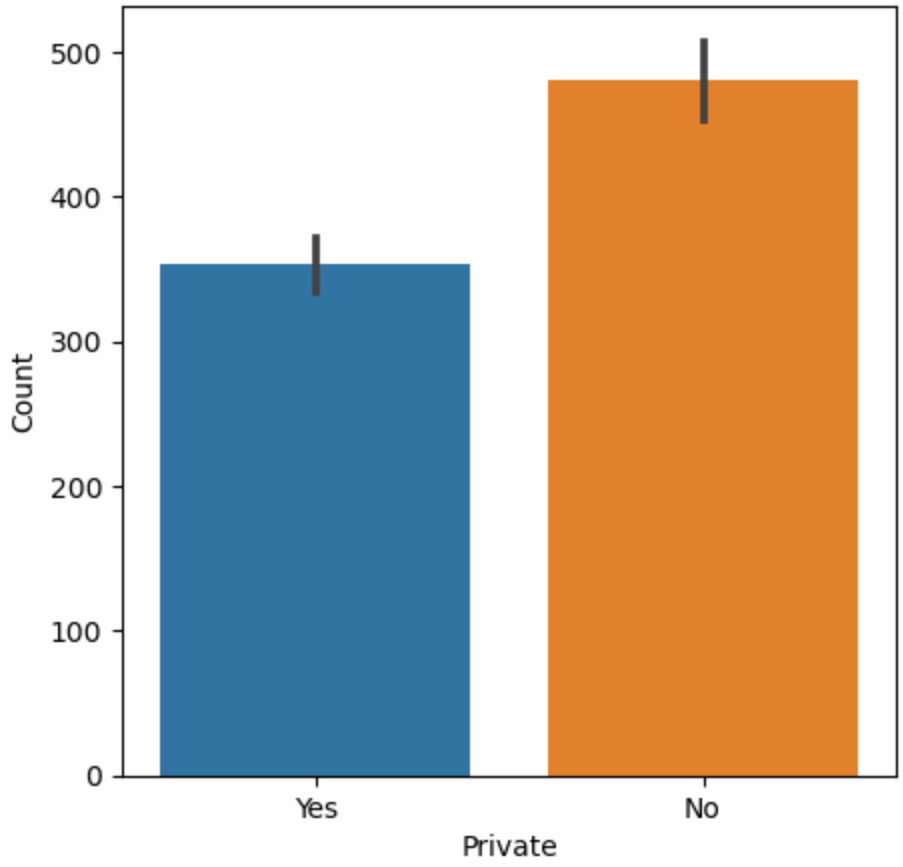
Out[12]:
```
0      False
1      False
2      False
3      False
4      False
       ...
772    False
773    False
774    False
775    False
776    False
Length: 777, dtype: bool
```

In [13]:
```
if not df[df.duplicated()].empty:
    print(df[df.duplicated()])
else:
    print("No duplicated datas")
```

```
No duplicated datas
```

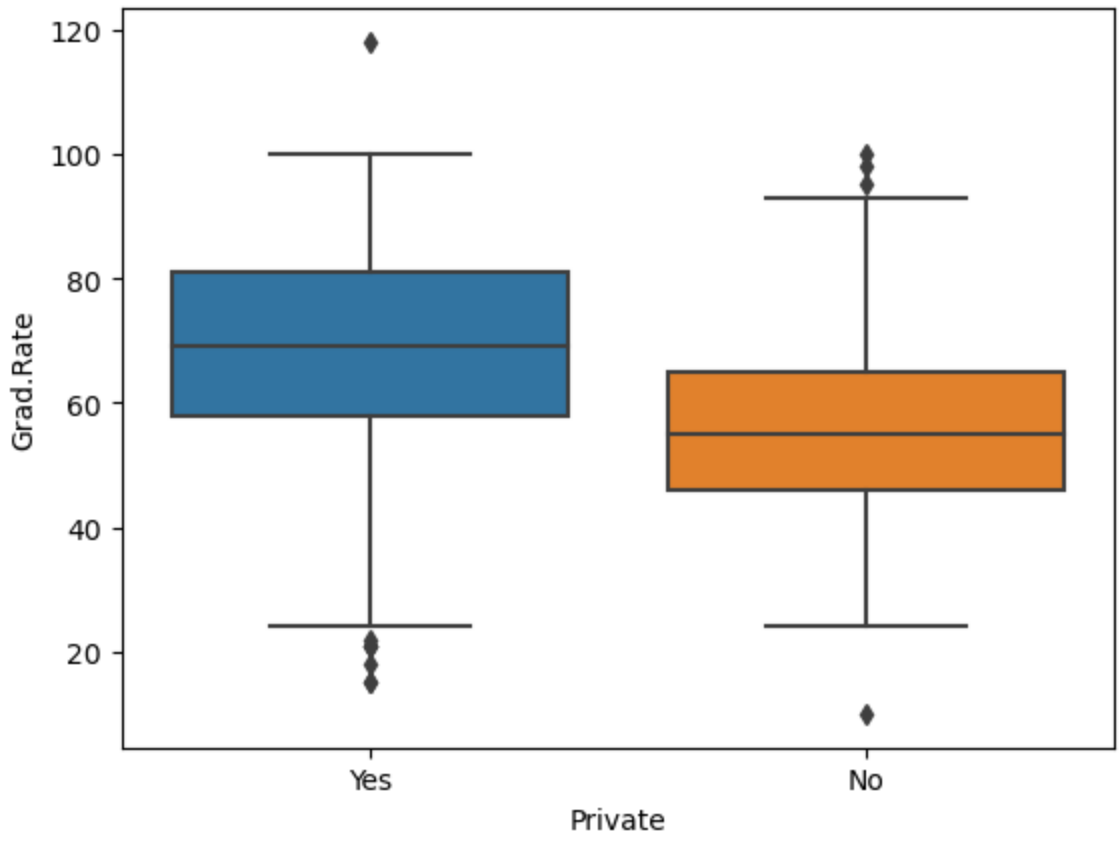
In [14]:
```
plt.figure(figsize=(5,5))
sns.barplot(x=df['Private'], y=df.index)
plt.xlabel("Private")
plt.ylabel("Count")

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



In [15]:
```
sns.boxplot(x="Private", y="Grad.Rate", data=df)
```

Out[15]:
```
<Axes: xlabel='Private', ylabel='Grad.Rate'>
```



In [16]:
```
df[(df["Grad.Rate"]>100)]["Grad.Rate"]
df["Grad.Rate"][95]=100
df[(df["Grad.Rate"]>100)]
```

```
C:\Users\tanut\AppData\Local\Temp\ipykernel_19360\377327122.py:2: SettingWithCopyWarnin
g:
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
  df["Grad.Rate"][95]=100
```

Out[16]:

	Unnamed: 0	Private	Apps	Accept	Enroll	Top10perc	Top25perc	F.Undergrad	P.Undergrad	Outstate	Room.B

In [17]:

```
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=2)
kmeans = df.drop("Private",axis=1)
kmeans
```

Out[17]:

	Unnamed: 0	Apps	Accept	Enroll	Top10perc	Top25perc	F.Undergrad	P.Undergrad	Outstate	Room.Boa
0	Abilene Christian University	1660	1232	721	23	52	2885	537	7440	33
1	Adelphi University	2186	1924	512	16	29	2683	1227	12280	64
2	Adrian College	1428	1097	336	22	50	1036	99	11250	37
3	Agnes Scott College	417	349	137	60	89	510	63	12960	54
4	Alaska Pacific University	193	146	55	16	44	249	869	7560	41
...	...	...	...	...	...	...	...	...	...	...
772	Worcester State College	2197	1515	543	4	26	3089	2029	6797	39
773	Xavier University	1959	1805	695	24	47	2849	1107	11520	49
774	Xavier University of Louisiana	2097	1915	695	34	61	2793	166	6900	42
775	Yale University	10705	2453	1317	95	99	5217	83	19840	65
776	York College of Pennsylvania	2989	1855	691	28	63	2988	1726	4990	35

777 rows × 18 columns

In [18]:

```
kmeans = KMeans(n_clusters=2)
features = df.iloc[:, 2:]
features
```

Out[18]:

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

777 rows × 17 columns

In [19]:
```
# Convert all columns datatypes to strings, to apply StandardScalar()
features.columns = features.columns.astype(str)
```

In [20]:
```
from sklearn.preprocessing import StandardScaler
# StandardScalar is a preprocessing class that is used to standardise or normalise the f
# It scales each feature in such a way that it has a mean of 0 and Std of 1.
```

In [21]:
```
scaler = StandardScaler()
scaled_features = scaler.fit_transform(features)
scaled_features.shape
```

Out[21]:
```
(777, 17)
```

In [22]:
```
kmeans = KMeans(n_clusters=2)
```

In [23]:
```
df['Cluster'] = kmeans.fit_predict(scaled_features)
df
```

```
C:\ProgramData\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1412: FutureWarnin
g: 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 warning
  super()._check_params_vs_input(X, default_n_init=10)
```

Out[23]:

	Unnamed: 0	Private	Apps	Accept	Enroll	Top10perc	Top25perc	F.Undergrad	P.Undergrad	Outstate	R
0	Abilene Christian University	Yes	1660	1232	721	23	52	2885	537	7440	
1	Adelphi University	Yes	2186	1924	512	16	29	2683	1227	12280	
2	Adrian College	Yes	1428	1097	336	22	50	1036	99	11250	
3	Agnes Scott College	Yes	417	349	137	60	89	510	63	12960	
4	Alaska Pacific University	Yes	193	146	55	16	44	249	869	7560	
...	...	...	...	...	...	...	...	...	...	...	...
772	Worcester State College	No	2197	1515	543	4	26	3089	2029	6797	
773	Xavier University	Yes	1959	1805	695	24	47	2849	1107	11520	
774	Xavier University of Louisiana	Yes	2097	1915	695	34	61	2793	166	6900	
775	Yale University	Yes	10705	2453	1317	95	99	5217	83	19840	
776	York College of Pennsylvania	Yes	2989	1855	691	28	63	2988	1726	4990	

777 rows × 20 columns

In [24]:
```
from sklearn.metrics import confusion_matrix, accuracy_score
```

In [25]:
```
print(confusion_matrix(df['Cluster'], kmeans.labels_))
```

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

In [26]:
```
print(accuracy_score(kmeans.labels_, df["Cluster"]))
```

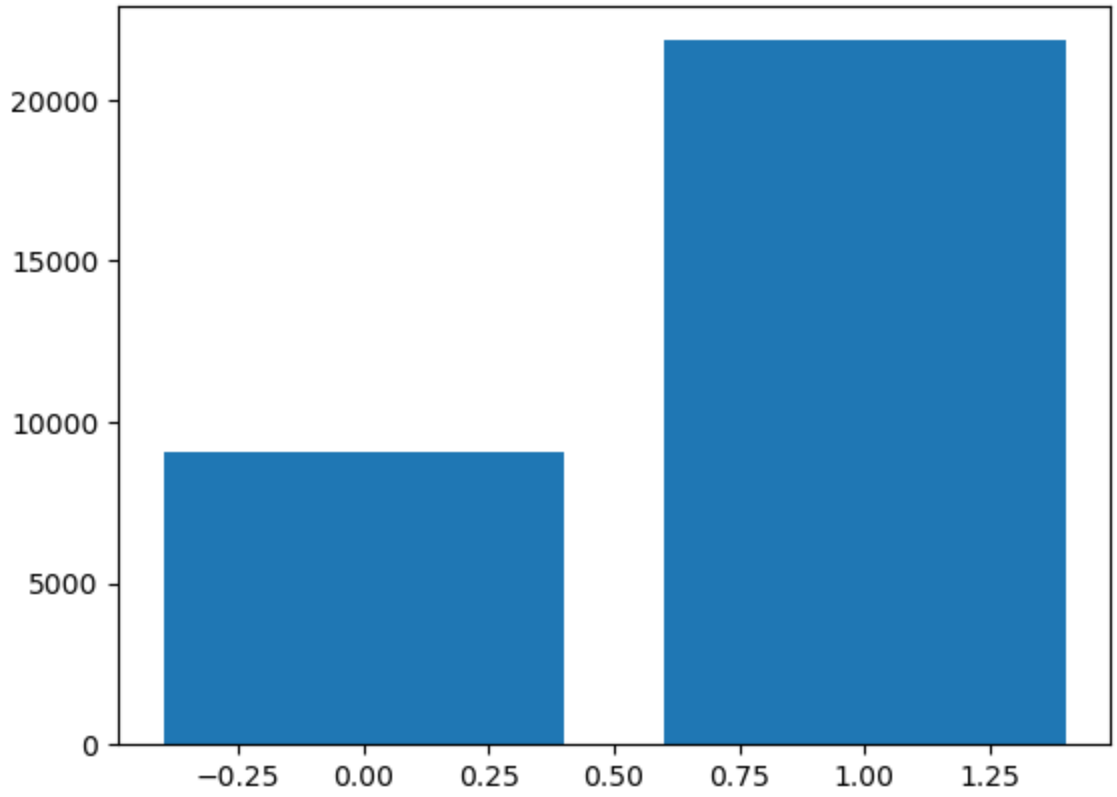
```
1.0
```

In [27]:
```
features.columns
```

Out[27]:
```
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'],
      dtype='object')
```

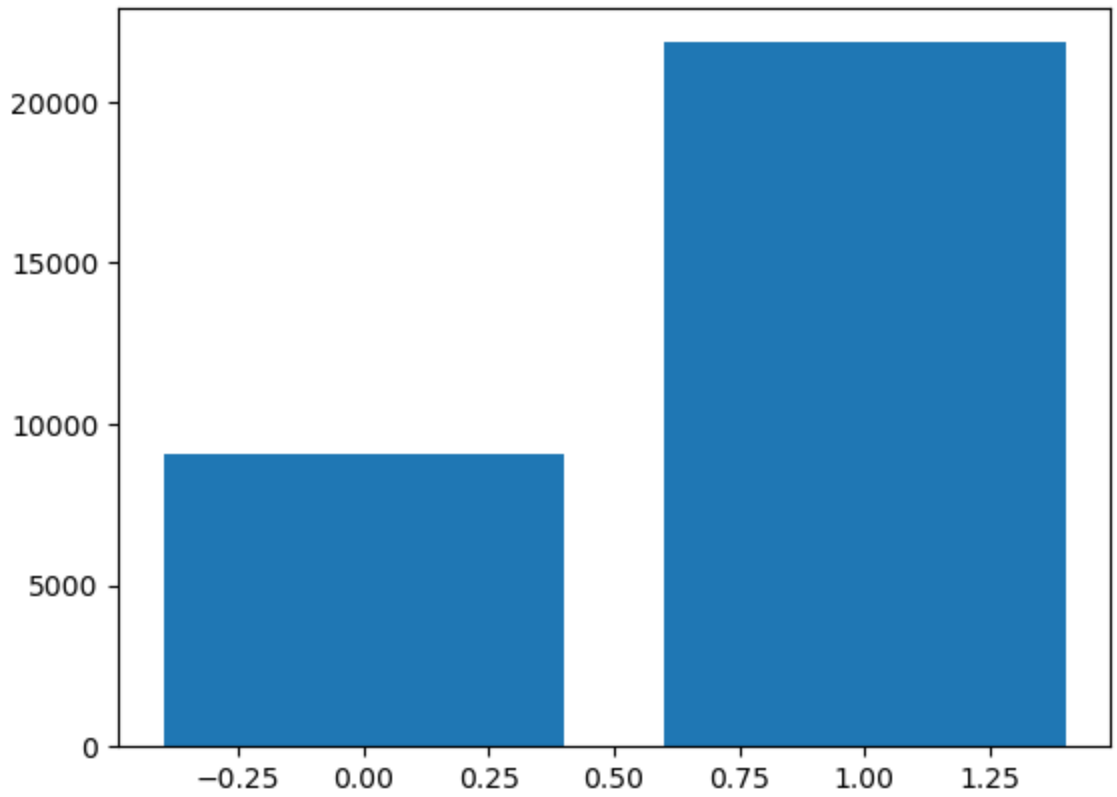
In [28]:
```
plt.bar(kmeans.labels_, features["P.Undergrad"])
```

Out[28]:
```
<BarContainer object of 777 artists>
```



In [29]:
```
plt.bar(df["Cluster"], features["P.Undergrad"])
```

Out[29]:
```
<BarContainer object of 777 artists>
```



In [30]:
```
# Diff between KNN and K means clustering

# 1) KNN is used for Classification and Regression
# K means is used for clustering problems

# 2) KNN is superviced algorithm
# K means is unsuperviced 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 into clusters
```

In [31]:
```
df = pd.read_csv("Classified Data", index_col=0)
df.head()
```

Out[31]:

	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 [32]:
```
from sklearn.preprocessing import StandardScaler
scalar = StandardScaler()
scalar.fit(df.drop("TARGET CLASS", axis=1))
```

Out[32]:
```
▼ StandardScaler
StandardScaler()
```

In [33]:
```
Scaled_features = scalar.transform(df.drop("TARGET CLASS", axis=1))
```

In [34]:
```
Scaled_features
```

Out[34]:
```
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 [35]:
```
df_feat = pd.DataFrame(Scaled_features)
df_feat.head()
```

Out[35]:

	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 [36]:
```
# Example of standard scalar
ay([[0,0],[0,1],[1,0],[1,1]])
```

```
data
```

Out[36]:
```
array([[0, 0],
       [0, 1],
       [1, 0],
       [1, 1]])
```

In [37]:
```
scl = StandardScaler()
scl
```

Out[37]:
```
▼ StandardScaler
StandardScaler()
```

In [38]:
```
scl_data = scl.fit_transform(data)
scl_data
```

Out[38]:
```
array([[-1., -1.],
       [-1.,  1.],
       [ 1., -1.],
       [ 1.,  1.]])
```

In [39]:
```
scl_data.mean()
```

Out[39]:
```
0.0
```

In [40]:
```
scl_data.std()
```

Out[40]:
```
1.0
```

In [41]:
```
df.head() #original data
```

Out[41]:

	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 [42]:
```
df_feat.head() #scaled data
```

Out[42]:

	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 [43]:
```
# To name the columns
df_feat = pd.DataFrame(Scaled_features, columns = df.columns[:-1])
df_feat.head()
```

Out[43]:

	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 [44]:
```
df_feat.isna().sum()
```

Out[44]:
```
WTT    0
PTI    0
EQW    0
SBI    0
LQE    0
QWG    0
FDJ    0
PJF    0
HQE    0
NXJ    0
dtype: int64
```

In [45]:
```
from sklearn.model_selection import train_test_split
x = df_feat
y = df["TARGET CLASS"]
x_train, x_test, y_train, y_test = train_test_split(x,y, test_size=0.3, random_state=101
x_train.shape
```

Out[45]:
```
(700, 10)
```

In [46]:
```
from sklearn.linear_model import LogisticRegression
log_model = LogisticRegression()
# To train the dataset
log_model.fit(x_train, y_train)
```

Out[46]:
```
▼ LogisticRegression
LogisticRegression()
```

In [47]:
```
pred = log_model.predict(x_test)
pred
```

Out[47]:
```
array([0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1,
       0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1,
       1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1,
       0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0,
       1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0,
       0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1,
       1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1,
       1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0,
       1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1,
       0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0,
       0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1,
       0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1,
       1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0,
       0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0], dtype=int64)
```

In [48]:
```
from sklearn.metrics import accuracy_score
accuracy_score(y_test, pred)
```

Out[48]:
```
0.9566666666666667
```

In [49]:
```
from sklearn.neighbors import KNeighborsClassifier
KNN = KNeighborsClassifier(n_neighbors=17)
KNN.fit(x_train, y_train)
```

Out[49]:
```
▼        KNeighborsClassifier
KNeighborsClassifier(n_neighbors=17)
```

In [50]:
```
pred = KNN.predict(x_test)
pred
```

Out[50]:
```
array([0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1,
       0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1,
       1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1,
       0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0,
       1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0,
       0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1,
       1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1,
       1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0,
       1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1,
       0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0,
       0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1,
       0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1,
       1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0,
       0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0], dtype=int64)
```

In [51]:
```
from sklearn.metrics import accuracy_score
accuracy_score(pred, y_test)
```

Out[51]:
```
0.95
```

In [60]:
```
# 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[60]:
```
[0.07666666666666666,
 0.09,
 0.05,
 0.07,
 0.056666666666666664,
 0.05,
 0.06666666666666667,
 0.06,
 0.05,
 0.056666666666666664,
 0.05333333333333334,
 0.04666666666666667,
 0.05,
 0.056666666666666664,
 0.056666666666666664,
 0.05,
 0.05,
 0.04666666666666667,
 0.05,
 0.05333333333333334,
 0.05,
 0.05,
 0.06333333333333334,
 0.056666666666666664,
 0.056666666666666664,
 0.05,
 0.05,
 0.04666666666666667,
 0.06]
```

In [62]:
```
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[62]:
```
Text(0, 0.5, 'Error Rate')
```

In [ ]: