A large technology firm needs your help, they've been hacked! Luckily their forensic engineers have grabbed valuable data about the hacks, including information like session time,locations, wpm typing speed, etc. The forensic engineer relates to you what she has been able to figure out so far, she has been able to grab meta data of each session that the hackers used to connect to their servers. These are the features of the data:

'Session_Connection_Time': How long the session lasted in minutes 'Bytes Transferred': Number of MB transferred during session 'Kali_Trace_Used': Indicates if the hacker was using Kali Linux 'Servers_Corrupted': Number of server corrupted during the attack 'Pages_Corrupted': Number of pages illegally accessed 'Location': Location attack came from (Probably useless because the hackers used VPNs) 'WPM_Typing_Speed': Their estimated typing speed based on session logs. The technology firm has 3 potential hackers that perpetrated the attack. Their certain of the first two hackers but they aren't very sure if the third hacker was involved or not. They have requested your help! Can you help figure out whether or not the third suspect had anything to do with the attacks, or was it just two hackers? It's probably not possible to know for sure, but maybe what you've just learned about Clustering can help!

One last key fact, the forensic engineer knows that the hackers trade off attacks. Meaning they should each have roughly the same amount of attacks. For example if there were 100 total attacks, then in a 2 hacker situation each should have about 50 hacks, in a three hacker situation each would have about 33 hacks. The engineer believes this is the key element to solving this, but doesn't know how to distinguish this unlabeled data into groups of hackers.

In [2]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

In [4]:

```
df=pd.read_csv(r'C:\Users\chumj\Downloads\hack_data.csv')
```

In [6]:

```
df.head(2)
```

Out[6]:

	Session_Connection_Time	Bytes Transferred	Kali_Trace_Used	Servers_Corrupted	Pages_Corrupted	Location	WPM_Typing_Speed
0	8.0	391.09	1	2.96	7.0	Slovenia	72.37
1	20.0	720.99	0	3.04	9.0	British Virgin Islands	69.08

In [7]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 334 entries, 0 to 333
Data columns (total 7 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   Session_Connection_Time  334 non-null    float64
 1   Bytes Transferred        334 non-null    float64
 2   Kali_Trace_Used          334 non-null    int64
 3   Servers_Corrupted        334 non-null    float64
 4   Pages_Corrupted          334 non-null    float64
 5   Location                 334 non-null    object
 6   WPM_Typing_Speed         334 non-null    float64
dtypes: float64(5), int64(1), object(1)
memory usage: 18.4+ KB
```

In [8]:

```
df.describe().transpose()
```

```
df.describe().transpose()
```

Out[8]:

	count	mean	std	min	25%	50%	75%	max
Session_Connection_Time	334.0	30.008982	14.088201	1.0	18.0000	31.000	42.0000	60.0
Bytes Transferred	334.0	607.245269	286.335932	10.0	372.2000	601.650	843.7025	1330.5
Kali_Trace_Used	334.0	0.511976	0.500607	0.0	0.0000	1.000	1.0000	1.0
Servers_Corrupted	334.0	5.258503	2.301907	1.0	3.1225	5.285	7.4000	10.0
Pages_Corrupted	334.0	10.838323	3.063526	6.0	8.0000	10.500	14.0000	15.0
WPM_Typing_Speed	334.0	57.342395	13.411063	40.0	44.1275	57.840	70.5775	75.0

In [9]:

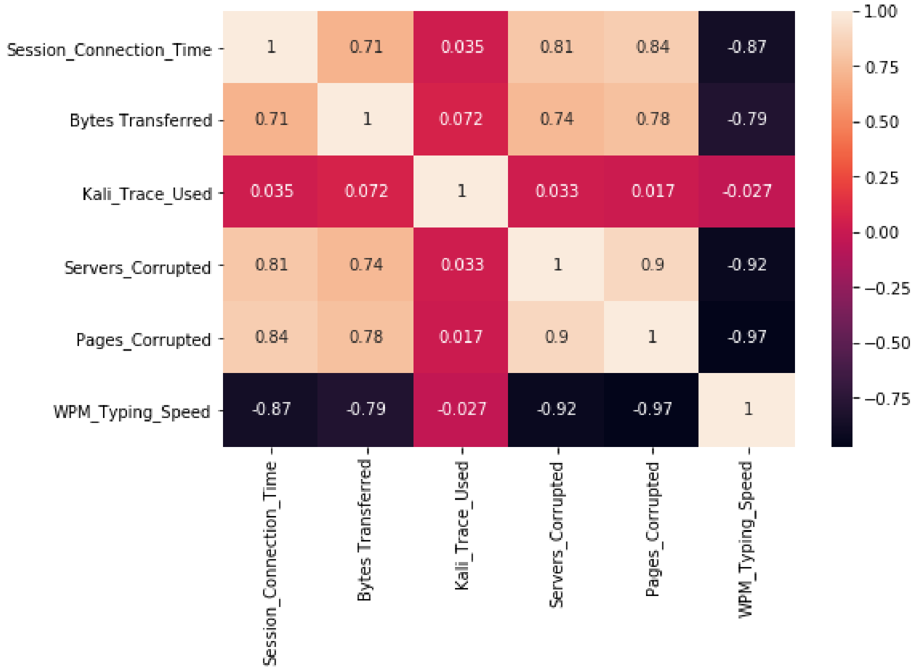
```
df.corr()
```

Out[9]:

	Session_Connection_Time	Bytes Transferred	Kali_Trace_Used	Servers_Corrupted	Pages_Corrupted	WPM_Typing
Session_Connection_Time	1.000000	0.713861	0.034687	0.808394	0.844167	-0.
Bytes Transferred	0.713861	1.000000	0.072436	0.739822	0.784081	-0.
Kali_Trace_Used	0.034687	0.072436	1.000000	0.033242	0.016931	-0.
Servers_Corrupted	0.808394	0.739822	0.033242	1.000000	0.897210	-0.
Pages_Corrupted	0.844167	0.784081	0.016931	0.897210	1.000000	-0.
WPM_Typing_Speed	-0.866077	-0.793344	-0.026560	-0.915629	-0.968662	1.

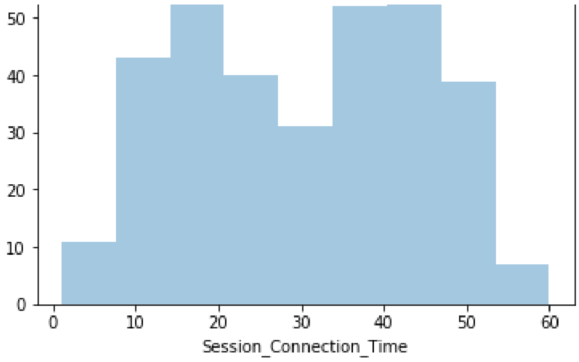
In [15]:

```
plt.figure(figsize=(8,5))
sns.heatmap(df.corr(),annot=True);
```



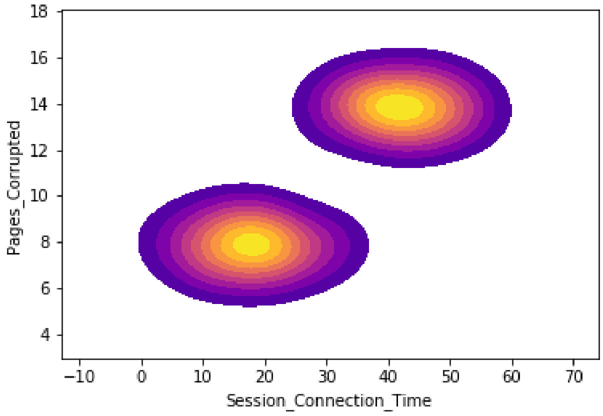
In [20]:

```
sns.distplot(df['Session_Connection_Time'],kde=False);
```



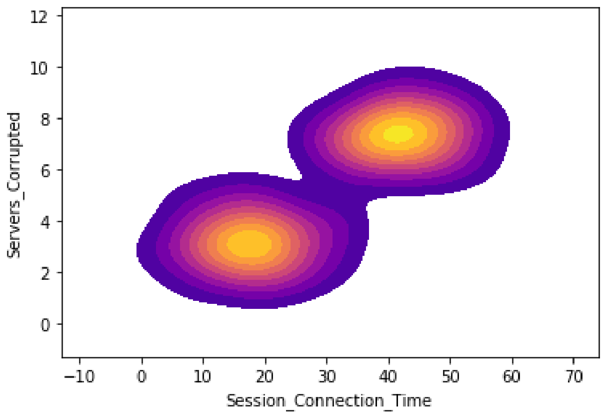
In [21]:

```
sns.kdeplot(df['Session_Connection_Time'],df['Pages_Corrupted'],cmap="plasma", shade=True,
shade_lowest=False);
```



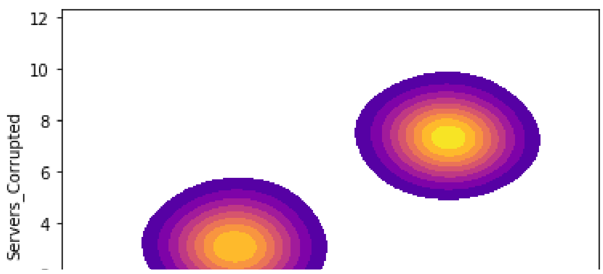
In [22]:

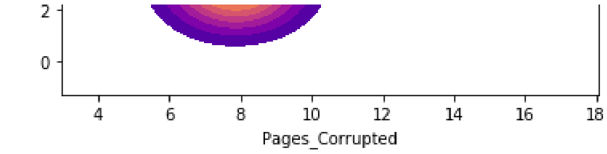
```
sns.kdeplot(df['Session_Connection_Time'],df['Servers_Corrupted'],cmap="plasma", shade=True,
shade_lowest=False);
```



In [23]:

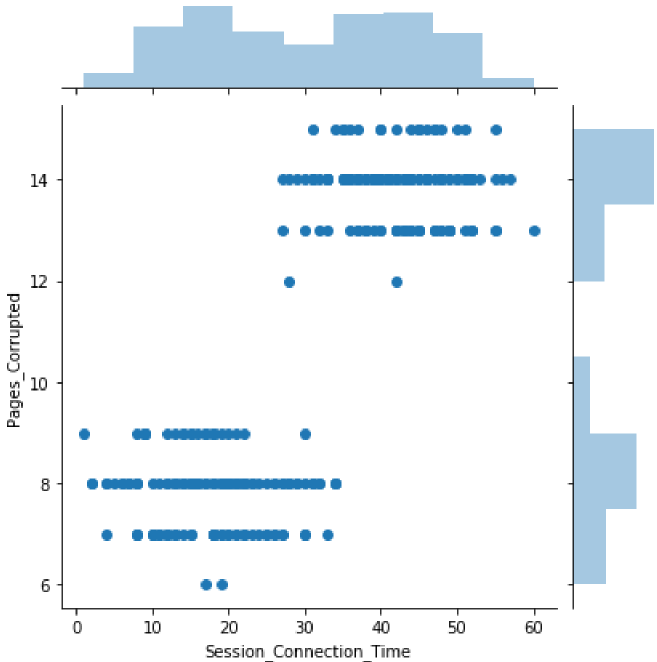
```
sns.kdeplot(df['Pages_Corrupted'],df['Servers_Corrupted'],cmap="plasma", shade=True, shade_lowest=
False);
```





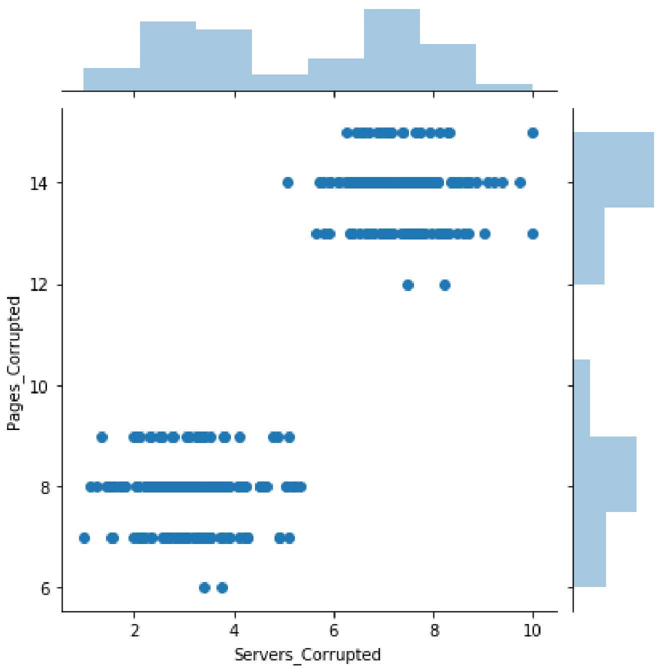
In [24]:

```
sns.jointplot(x='Session_Connection_Time',y='Pages_Corrupted',data=df);
```



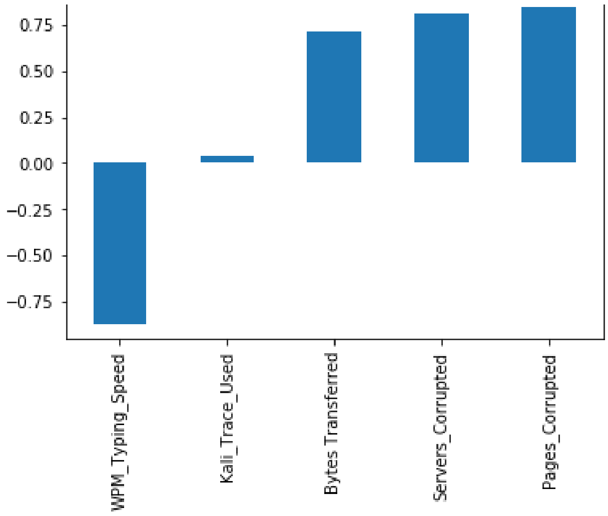
In [25]:

```
sns.jointplot(x='Servers_Corrupted',y='Pages_Corrupted',data=df);
```



In [28]:

```
df.corr()['Session_Connection_Time'].sort_values().drop('Session_Connection_Time').plot(kind='bar'
);
```



In [29]:

```
from sklearn.cluster import KMeans
```

In [41]:

```
DF1=df.drop('Location',axis=1)
```

In [42]:

```
DF1
```

Out[42]:

	Session_Connection_Time	Bytes Transferred	Kali_Trace_Used	Servers_Corrupted	Pages_Corrupted	WPM_Typing_Speed
0	8.0	391.09	1	2.96	7.0	72.37
1	20.0	720.99	0	3.04	9.0	69.08
2	31.0	356.32	1	3.71	8.0	70.58
3	2.0	228.08	1	2.48	8.0	70.80
4	20.0	408.50	0	3.57	8.0	71.28
...	...	...	...	...	...	...
329	39.0	761.91	1	6.99	14.0	43.23
330	43.0	983.48	0	8.60	13.0	43.21
331	39.0	690.22	1	6.80	13.0	42.75
332	36.0	1060.69	1	6.26	14.0	43.86
333	42.0	729.47	0	7.95	14.0	45.27

334 rows × 6 columns

In [43]:

```
# standardizing the data
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
data_scaled = scaler.fit_transform(DF1)
```

In [44]:

```
data_scaled
```

Out[44]:

```
array([[-1.56457197, -0.75603359,  0.97632801, -1.00001941, -1.25479003,
         1.12221913],
       [-0.71151736,  0.39783827, -1.02424594, -0.96521347, -0.60096808,
```

```
       [-0.71151756,  0.39783827, -1.02424594, -0.96521347, -0.60096808,
         0.87653121],
       [ 0.07044937, -0.87764658,  0.97632801, -0.67371374, -0.92787905,
         0.98854698],
       ...,
       [ 0.63915245,  0.29021584,  0.97632801,  0.67066562,  0.70667582,
        -1.0897189 ],
       [ 0.42588879,  1.58598701,  0.97632801,  0.43572553,  1.03358679,
        -1.00682724],
       [ 0.8524161 ,  0.42749826, -1.02424594,  1.17100097,  1.03358679,
        -0.90153241]])
```

In [45]:

```
# statistics of scaled data
pd.DataFrame(data_scaled)
```

Out[45]:

	0	1	2	3	4	5
0	-1.564572	-0.756034	0.976328	-1.000019	-1.254790	1.122219
1	-0.711517	0.397838	-1.024246	-0.965213	-0.600968	0.876531
2	0.070449	-0.877647	0.976328	-0.673714	-0.927879	0.988547
3	-1.991099	-1.326184	0.976328	-1.208855	-0.927879	1.004976
4	-0.711517	-0.695140	-1.024246	-0.734624	-0.927879	1.040821
...	...	...	...	...	...	...
329	0.639152	0.540962	0.976328	0.753330	1.033587	-1.053874
330	0.923504	1.315934	-1.024246	1.453799	0.706676	-1.055367
331	0.639152	0.290216	0.976328	0.670666	0.706676	-1.089719
332	0.425889	1.585987	0.976328	0.435726	1.033587	-1.006827
333	0.852416	0.427498	-1.024246	1.171001	1.033587	-0.901532

334 rows × 6 columns

In [48]:

```
pd.DataFrame(data_scaled).describe().transpose()
```

Out[48]:

	count	mean	std	min	25%	50%	75%	max
0	334.0	-2.695781e-16	1.0015	-2.062187	-0.853693	0.070449	0.852416	2.131998
1	334.0	-2.434845e-16	1.0015	-2.088950	-0.822104	-0.019570	0.827043	2.529686
2	334.0	-3.589943e-17	1.0015	-1.024246	-1.024246	0.976328	0.976328	0.976328
3	334.0	-4.653629e-18	1.0015	-1.852765	-0.929320	0.011528	0.931710	2.062903
4	334.0	-2.539552e-16	1.0015	-1.581701	-0.927879	-0.110602	1.033587	1.360498
5	334.0	-2.360055e-16	1.0015	-1.295081	-0.986851	0.037160	0.988360	1.318620

In [64]:

```
# defining the kmeans function with initialization as k-means++
km = KMeans(n_clusters=2, init='k-means++')

km
```

Out[64]:

```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
       n_clusters=2, n_init=10, n_jobs=None, precompute_distances='auto',
       random_state=None, tol=0.0001, verbose=0)
```

In [65]:

```
# fitting the k means algorithm on scaled data
km.fit(data_scaled)
```

Out[65]:

```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
       n_clusters=2, n_init=10, n_jobs=None, precompute_distances='auto',
       random_state=None, tol=0.0001, verbose=0)
```

In [66]:

```
# inertia on the fitted data
km.inertia_
```

Out[66]:

```
603.5778706408448
```

In [67]:

```
# fitting multiple k-means algorithms and storing the values in an empty list
SSE = []
for cluster in range(1,5):
    km = KMeans(n_jobs = -1, n_clusters = cluster, init='k-means++')
    km.fit(data_scaled)
    SSE.append(km.inertia_)
```

In [68]:

```
# converting the results into a dataframe and plotting them
frame = pd.DataFrame({'Cluster':range(1,5), 'SSE':SSE})
```

In [69]:

```
frame
```

Out[69]:

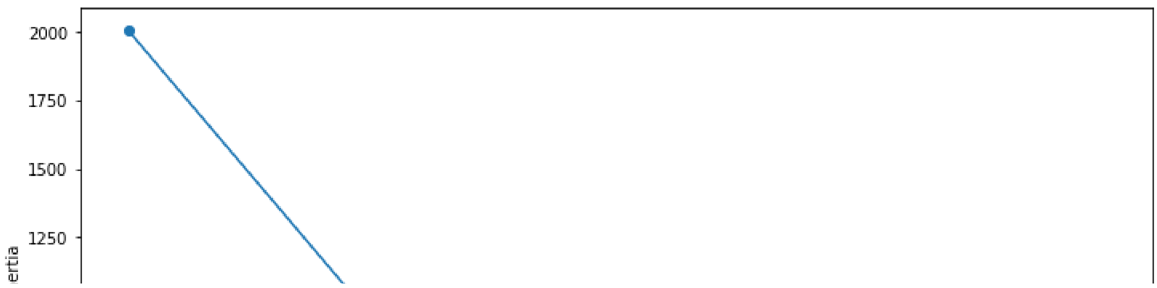
	Cluster	SSE
0	1	2004.000000
1	2	603.577871
2	3	435.453041
3	4	267.935815

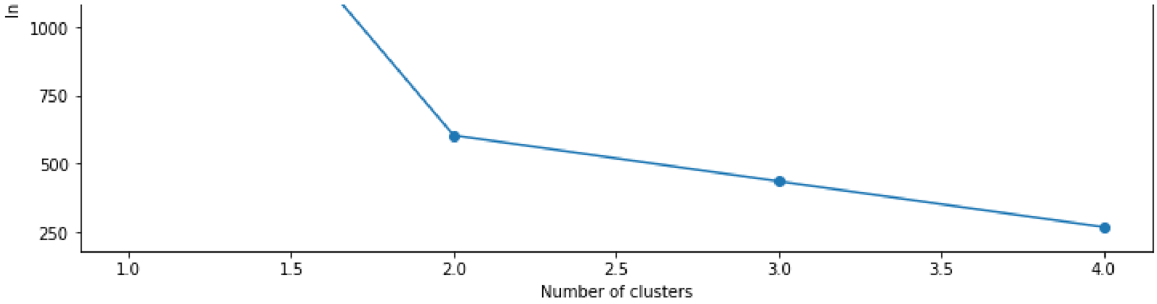
In [70]:

```
plt.figure(figsize=(12,6))
plt.plot(frame['Cluster'], frame['SSE'], marker='o')
plt.xlabel('Number of clusters')
plt.ylabel('Inertia')
```

Out[70]:

```
Text(0, 0.5, 'Inertia')
```





In [72]:

```
# k means using 2 clusters and k-means++ initialization
km = KMeans(n_jobs = -1, n_clusters = 2, init='k-means++')
km.fit(data_scaled)
pred = km.predict(data_scaled)
```

In [73]:

```
pred
```

Out[73]:

```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0])
```

let's look at the value count of points in each of the above-formed clusters:

In [74]:

```
frame = pd.DataFrame(data_scaled)
```

In [75]:

```
frame['cluster'] = pred
frame['cluster'].value_counts()
```

Out[75]:

```
1    167
0    167
Name: cluster, dtype: int64
```

In [76]:

```
# k means using 3 clusters and k-means++ initialization
km = KMeans(n_jobs = -1, n_clusters = 3, init='k-means++')
km.fit(data_scaled)
pred = km.predict(data_scaled)
```

In [77]:

```
frame = pd.DataFrame(data_scaled)
frame['cluster'] = pred
```

```
frame['cluster'] = pred
frame['cluster'].value_counts()
```

Out[77]:

```
1    167
0     84
2     83
Name: cluster, dtype: int64
```

In [78]:

```
# k means using 4 clusters and k-means++ initialization
km = KMeans(n_jobs = -1, n_clusters = 4, init='k-means++')
km.fit(data_scaled)
pred = km.predict(data_scaled)
```

In [79]:

```
frame = pd.DataFrame(data_scaled)
frame['cluster'] = pred
frame['cluster'].value_counts()
```

Out[79]:

```
0    88
2    84
1    83
3    79
Name: cluster, dtype: int64
```

2 clusters is the best ,base on the even counts of 167 and 167

In [ ]:

```

```

In [ ]:

```

```