Banknotes

September 13, 2019

[1]:
```
#import libraries
import numpy as np
import pandas as pd
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
```

[2]:
```
#load the data
data = pd.read_csv('Banknote-authentication-dataset-.csv')
data.head()
```

[2]:

	V1	V2
0	3.62160	8.6661
1	4.54590	8.1674
2	3.86600	-2.6383
3	3.45660	9.5228
4	0.32924	-4.4552

0.1 Exploratory analysis of the data

[3]:
```
max_v1 = np.max(data['V1'])
min_v1 = np.min(data['V1'])
print(max_v1, min_v1)
```

```
6.8248 -7.0421
```

[4]:
```
max_v2 = np.max(data['V2'])
min_v2 = np.min(data['V2'])
print(max_v2, min_v2)
```

```
12.9516 -13.7731
```

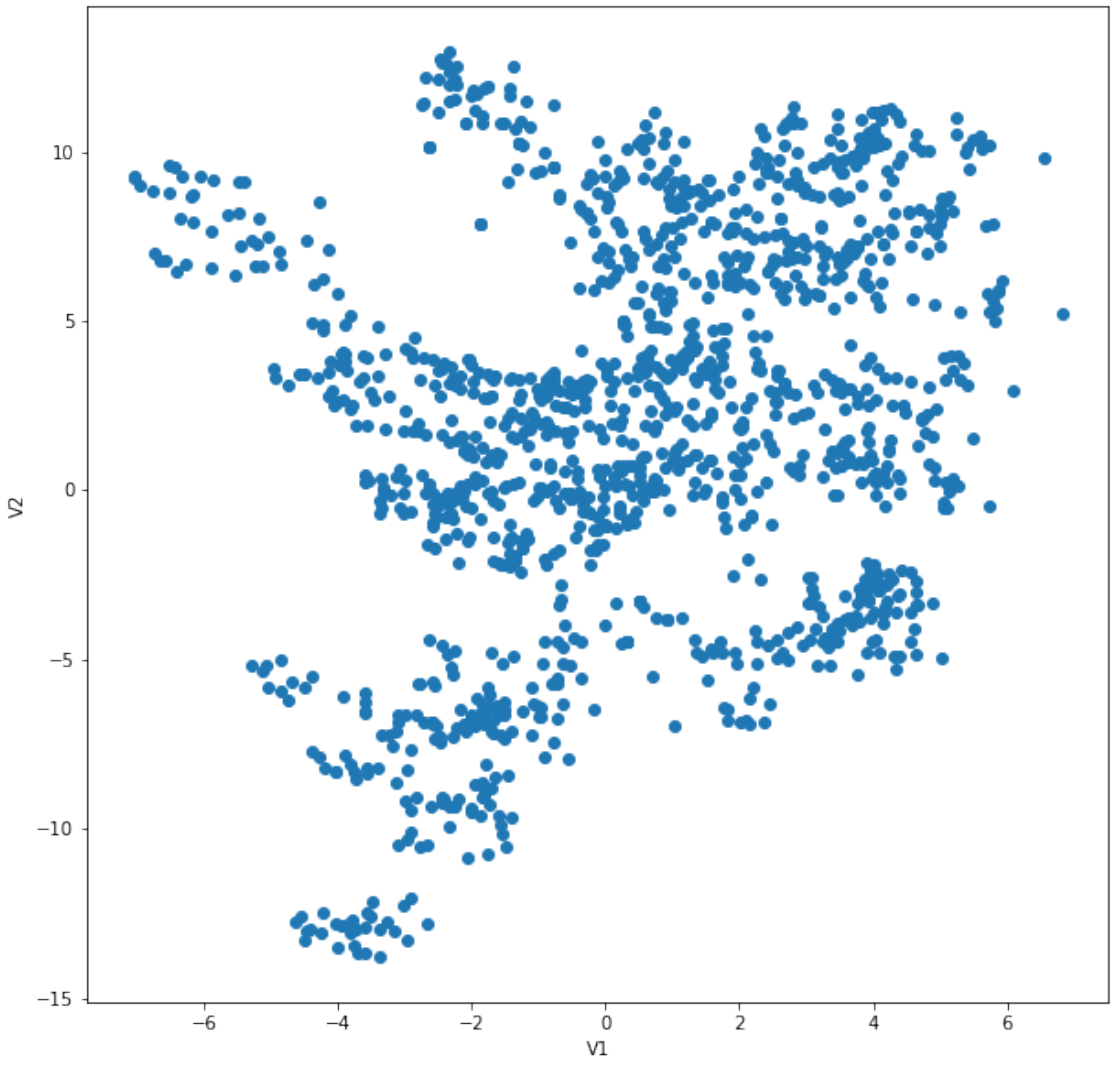
[5]:
```
mean_v1 = np.mean(data['V1'])
std_v1 = np.std(data['V1'])
print(mean_v1, std_v1)
```

```
0.43373525728862977 2.8417264052060993
```

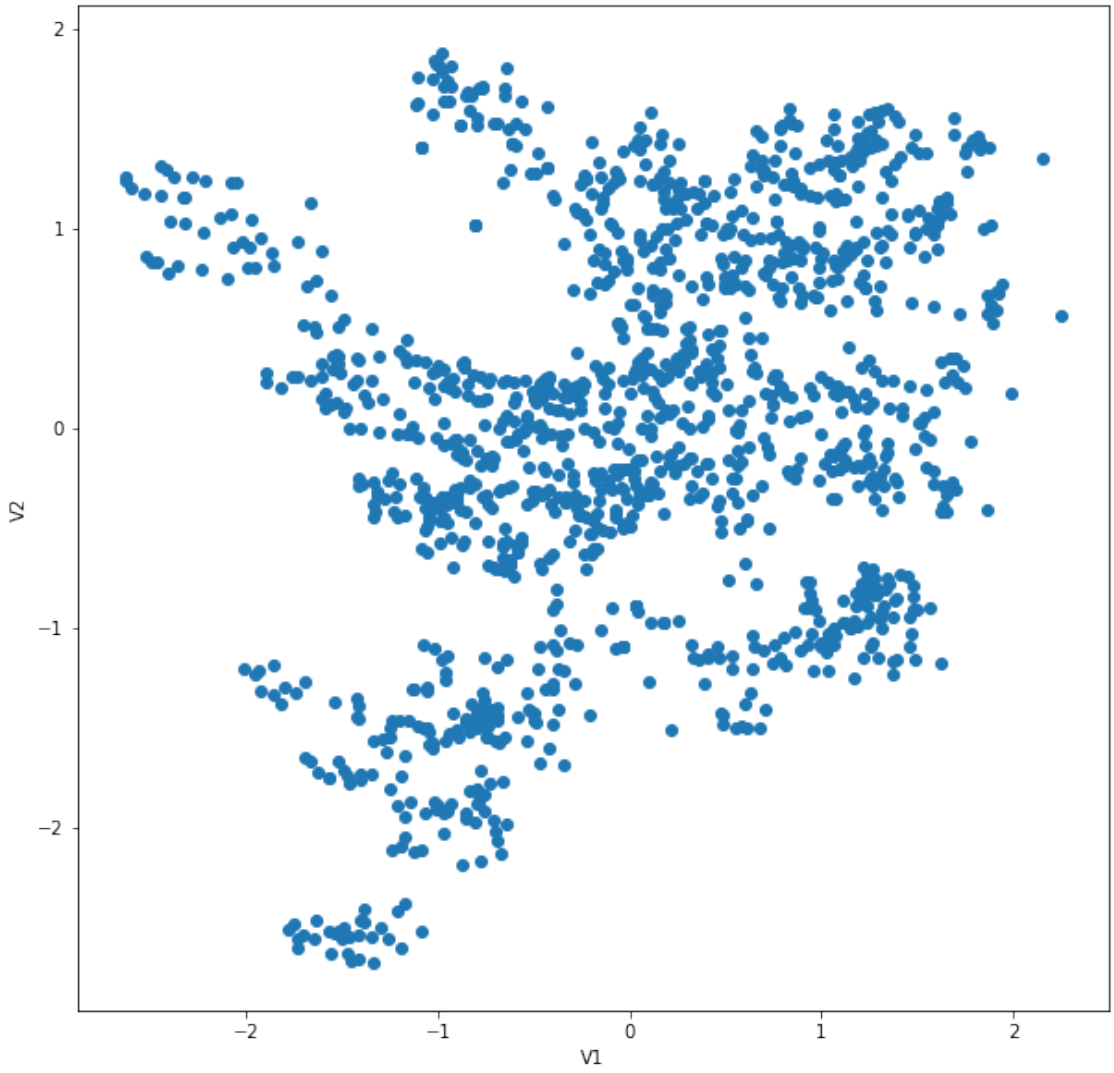
[6]:
```
mean_v2 = np.mean(data['V2'])
std_v2 = np.std(data['V2'])
print(mean_v2, std_v2)
```

```
1.9223531209912554 5.866907488271995
```

[7]:
```
#plot the standardised data
plt.figure(figsize = (10,10))
plt.xlabel('V1')
plt.ylabel('V2')
plt.scatter(data['V1'], data['V2'])
plt.show()
```



[8]:
```
# Standardize the data
X_std = StandardScaler().fit_transform(data)

#plot the standardised data
plt.figure(figsize = (10,10))
plt.xlabel('V1')
plt.ylabel('V2')
plt.scatter(X_std[:,0], X_std[:,1])
plt.show()
```

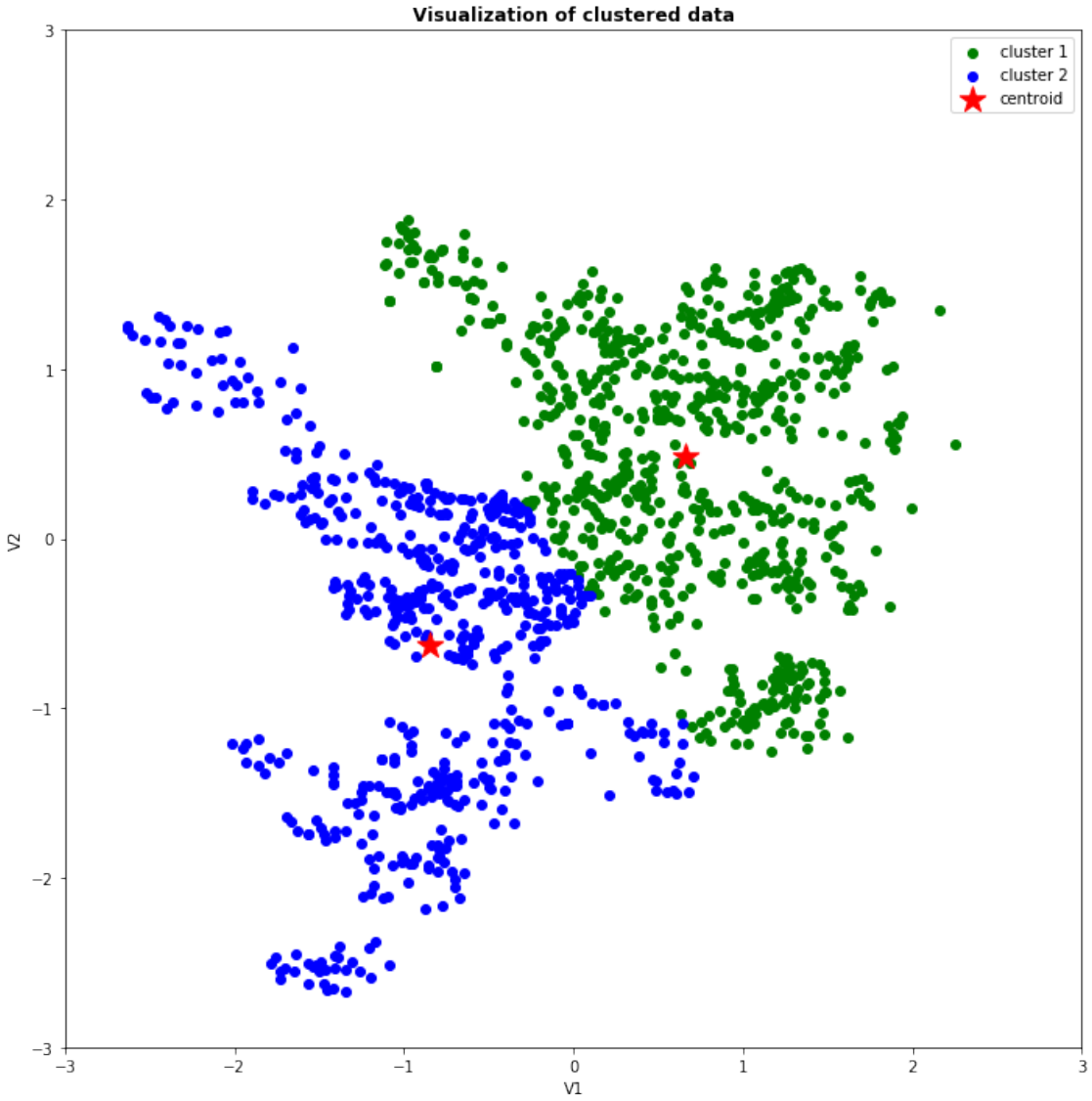


0.2 Step 1: run K-means on the given dataset

[9]:
```
# Standardize the data
X_std = StandardScaler().fit_transform(data)

# Run local implementation of kmeans
km = KMeans(n_clusters=2).fit(X_std)
centroids = km.cluster_centers_
```

0.3 Step 2: visualise the results.

[10]:
```
# Run local implementation of kmeans
km = KMeans(n_clusters=2).fit(X_std)
centroids = km.cluster_centers_

# Plot the clustered data
fig, ax = plt.subplots(figsize=(12, 12))
plt.scatter(X_std[km.labels_ == 0, 0], X_std[km.labels_ == 0, 1],
            c='green', label='cluster 1')
plt.scatter(X_std[km.labels_ == 1, 0], X_std[km.labels_ == 1, 1],
            c='blue', label='cluster 2')
plt.scatter(centroids[:, 0], centroids[:, 1], marker='*', s=300,
            c='r', label='centroid')
plt.legend()
plt.xlim([-3, 3])
plt.ylim([-3, 3])
plt.xlabel('V1')
plt.ylabel('V2')
plt.title('Visualization of clustered data', fontweight='bold')
ax.set_aspect('equal');
```



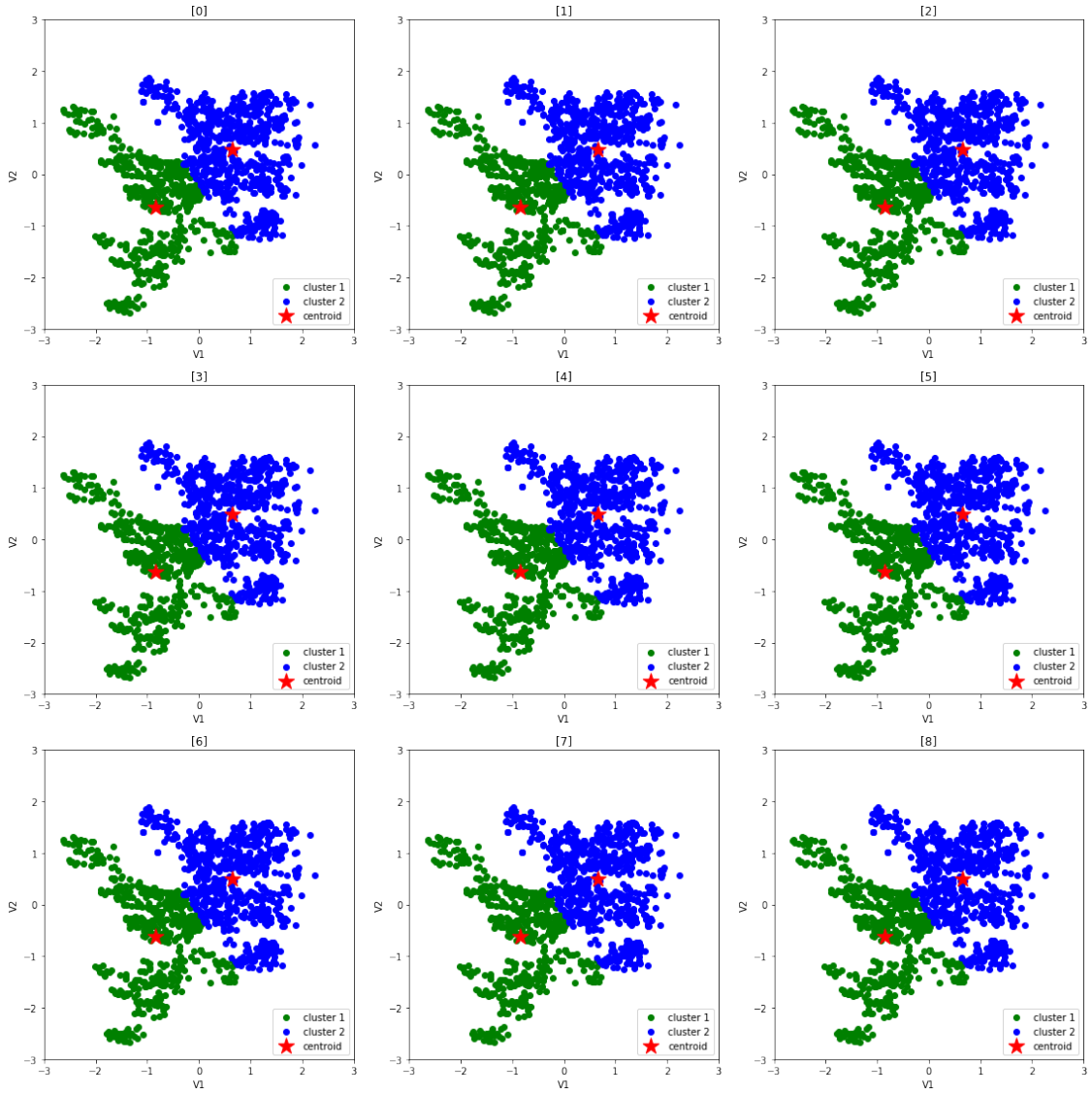
0.4 Step 3: re-run K-means several times and visualise the results.

[11]:
```
final_cents = []
final_inert = []
n_iter = 9

for sample in range(9):
    print('\nCentroid attempt: ', sample)
    km = KMeans(n_clusters=2, init='random', max_iter=1, n_init=1)#, verbose=1)
    km.fit(X_std)
    inertia_start = km.inertia_
    inertia_end = 0
```

```
    cents = km.cluster_centers_

    for iter in range(n_iter):
        km = KMeans(n_clusters=2, init=cents, max_iter=3, n_init=1)
        km.fit(X_std)
        intertia_end = km.inertia_
        cents = km.cluster_centers_
    final_cents.append(cents)
    final_inert.append(inertia_end)
    print('Difference between initial and final inertia: ',␣
 ↪inertia_start-inertia_end)
    print("Final Centroids:", cents)


fig, ax = plt.subplots(3, 3, figsize=(16, 16))
ax = np.ravel(ax)
centers = []
for i in range(n_iter):
    # Run local implementation of kmeans
    centroids = cents
    centers.append(centroids)
    ax[i].scatter(X_std[km.labels_ == 0, 0], X_std[km.labels_ == 0, 1],
                  c='green', label='cluster 1')
    ax[i].scatter(X_std[km.labels_ == 1, 0], X_std[km.labels_ == 1, 1],
                  c='blue', label='cluster 2')
    ax[i].scatter(centroids[:, 0], centroids[:, 1],
                  c='r', marker='*', s=300, label='centroid')
    ax[i].set_xlim([-3, 3])
    ax[i].set_ylim([-3, 3])
    ax[i].set_xlabel('V1')
    ax[i].set_ylabel('V2')
    ax[i].set_title([i])
    ax[i].legend(loc='lower right')
    ax[i].set_aspect('equal')
plt.tight_layout();
```

```
Centroid attempt:  0
Difference between initial and final inertia:  1734.52357449641
Final Centroids: [[-0.850656   -0.6310923 ]
 [ 0.65527953  0.48614465]]

Centroid attempt:  1
Difference between initial and final inertia:  1869.5197342341114
Final Centroids: [[ 0.65527953  0.48614465]
 [-0.850656   -0.6310923 ]]
```

```
Centroid attempt:  2
Difference between initial and final inertia:  2295.5182947496805
Final Centroids: [[-0.850656   -0.6310923 ]
 [ 0.65527953  0.48614465]]

Centroid attempt:  3
Difference between initial and final inertia:  1601.5076034675965
Final Centroids: [[ 0.65527953  0.48614465]
 [-0.850656   -0.6310923 ]]

Centroid attempt:  4
Difference between initial and final inertia:  1562.712456034259
Final Centroids: [[-0.850656   -0.6310923 ]
 [ 0.65527953  0.48614465]]

Centroid attempt:  5
Difference between initial and final inertia:  1954.959873218072
Final Centroids: [[ 0.65527953  0.48614465]
 [-0.850656   -0.6310923 ]]

Centroid attempt:  6
Difference between initial and final inertia:  2071.592219456241
Final Centroids: [[-0.850656   -0.6310923 ]
 [ 0.65527953  0.48614465]]

Centroid attempt:  7
Difference between initial and final inertia:  1569.2221272753648
Final Centroids: [[ 0.65527953  0.48614465]
 [-0.850656   -0.6310923 ]]

Centroid attempt:  8
Difference between initial and final inertia:  1609.034592303246
Final Centroids: [[-0.850656   -0.6310923 ]
 [ 0.65527953  0.48614465]]
```



From the repeated centroid attempts, the final centroids remain the same suggesting the centroids are stable, with the centroids for the clusters being: (0.65527953, 0.48614465) and (-0.850656, -0.6310923) for the standardised data.

0.5 Step 4: compare the results: is the K-means algorithm stable?

[12]:
```
openML = pd.read_csv('https://www.openml.org/data/get_csv/1586223/php50jXam')
openML.head()
```

[12]:

	V1	V2	V3	V4	Class
0	3.62160	8.6661	-2.8073	-0.44699	1
1	4.54590	8.1674	-2.4586	-1.46210	1
2	3.86600	-2.6383	1.9242	0.10645	1
3	3.45660	9.5228	-4.0112	-3.59440	1

```
4  0.32924 -4.4552  4.5718 -0.98880      1
```

[13]:
```
# Take random sample from Open ML data
random_sample = openML.sample(n=10)
random_sample
```

[13]:
```
          V1        V2        V3        V4  Class
319   2.9499   2.24930  1.345800 -0.037083      1
741   3.5862  -3.09570  2.809300  0.244810      1
582   4.0552   0.40143  1.456300  0.653430      1
598   1.1050   7.44320  0.410990 -3.033200      1
187   1.8314   6.36720 -0.036278  0.049554      1
98    3.8200  10.92790 -4.011200 -5.028400      1
514   2.0153   1.84790  3.137500  0.428430      1
1361 -1.5732   1.06360 -0.712320 -0.838800      2
1232 -4.1244   3.79090 -0.653200 -4.180200      2
1269 -2.1241  -6.89690  5.599200 -0.471560      2
```

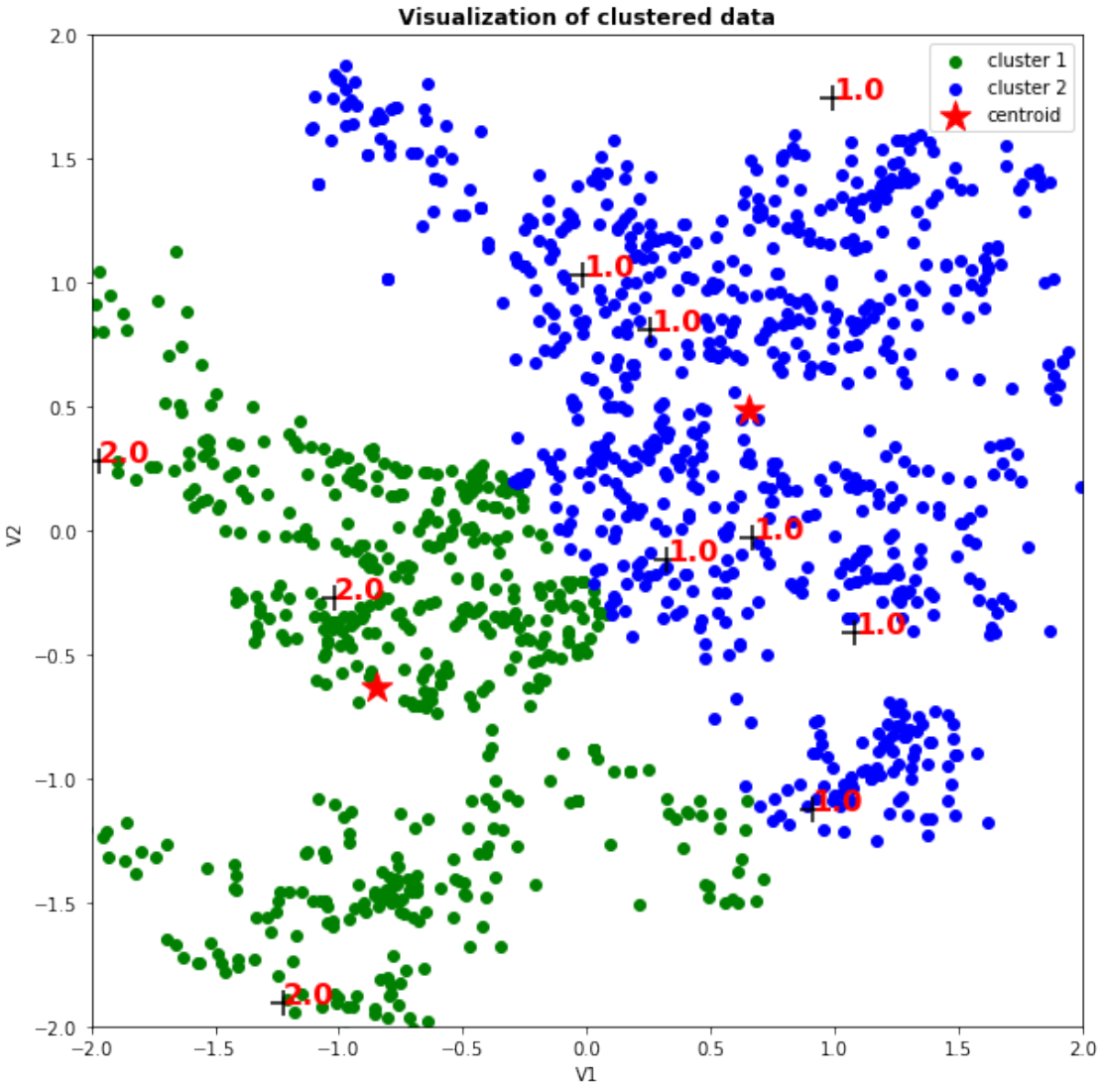
[14]:
```
# separate out V1, V2 and class columns
features = random_sample[['V1','V2', 'Class']]
```

[15]:
```
# standardise V1 and V2 values to match
col_names = ['V1', 'V2']
scaler = StandardScaler().fit_transform(features[col_names])
features[col_names] = scaler
```

```
/Users/charlottefettes/anaconda3/lib/python3.7/site-
packages/ipykernel_launcher.py:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-
docs/stable/indexing.html#indexing-view-versus-copy
  after removing the cwd from sys.path.
/Users/charlottefettes/anaconda3/lib/python3.7/site-
packages/pandas/core/indexing.py:543: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-
docs/stable/indexing.html#indexing-view-versus-copy
  self.obj[item] = s
```

[16]:
```
# Plot the clustered data with randomly selected clusters and labels
fig, ax = plt.subplots(figsize=(10, 10))
plt.scatter(X_std[km.labels_ == 0, 0], X_std[km.labels_ == 0, 1],
            c='green', label='cluster 1')
plt.scatter(X_std[km.labels_ == 1, 0], X_std[km.labels_ == 1, 1],
            c='blue', label='cluster 2')
```

```
plt.scatter(centroids[:, 0], centroids[:, 1], marker='*', s=300,
            c='r', label='centroid')

plt.scatter(features['V1'], features['V2'], color='black', marker='+', s=200)

for k,row in features.iterrows():
    plt.text(row['V1'], row['V2'], row['Class'], fontsize=16, fontweight='bold',␣
 ↪color='red')

plt.legend()
plt.xlim([-2, 2])
plt.ylim([-2, 2])
plt.xlabel('V1')
plt.ylabel('V2')
plt.title('Visualization of clustered data', fontweight='bold')
ax.set_aspect('equal');
```



0.6 Step 5: describe your results.

K-Means clustering of n=2 on the unlabelled standardised dataset returned two cluster centroids of (0.65527953, 0.48614465) and (-0.850656, -0.6310923), which are shown as stars in the scatterplot.

Re-running K-means 9 times with different, randomly selected starting conditions, continued to return these centroids, with the results illustrated by the subplots. This suggests the centroids are reliable.

The OpenML dataset, which included the labels for the data, was downloaded, and 10 points randomly selected and plotted with their class labels on the scatterplot with the unlabelled data and clusters colour-coded. From visualising this scatterplot, it is seen that all 10 observations selected at random are within different clusters depending on their class - 7 observations labelled 1 are within the blue cluster in the scatterplot and 3 observations labelled 2 are within the green scatterplot. This suggests that the algorithm is stable and reliably predicts which cluster an item with features V1 and V2 is in (based on this randomly selected data).