DBSCAN

Source:

https://www.kaggle.com/lava18/google-play-store-apps (https://www.kaggle.com/lava18/google-play-store-apps)

Defining the Problem Statement

This dataset records the attributes of Android mobile applications in the Google Play Store. From this dataset, we would like to be able to find the best clustering results/optimum number of clusters.

Collecting the Data

In [0]:

```
#importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn import datasets
from sklearn.cluster import KMeans
from sklearn.cluster import DBSCAN
from sklearn.preprocessing import StandardScaler
```

In [0]:

```
dataset = pd.read_csv('googleps_cleaned.csv')
dataset.head()
```

Out[0]:

	Category	Rating	Reviews	Size	Installs	Type	Price	Content Rating	Genres
0	0	High Rating	159.0	19.0	9	0	0.0	1	9
1	0	Average Rating	967.0	14.0	12	0	0.0	1	11
2	0	High Rating	87510.0	8.7	14	0	0.0	1	9
3	0	High Rating	215644.0	25.0	16	0	0.0	4	9
4	0	High Rating	967.0	2.8	11	0	0.0	1	10

In [0]:

```
# Util Functions
```

In [0]:

```
import seaborn as sns
from sklearn.cluster import DBSCAN
from sklearn.preprocessing import StandardScaler


def get_features(cols): return dataset.loc[:, cols]

def get_unique_class_and_count(series):
  res = pd.Series(series).value_counts()
  return res.keys(), res

def scaler_util(x):
  scaler = StandardScaler()
  X_scaled = scaler.fit_transform(x)
  return scaler, X_scaled
```

All columns DBSCAN

In [0]:

```
def perform_dbscan_all(x_scaled, eps=0.5, min_samples = 2 , left_column = 0, right_colu
mn=2, one_vs_all=False):
  dbscan_obj = DBSCAN(eps=eps, min_samples = min_samples)
  clusters = dbscan_obj.fit_predict(x_scaled)

  print("Ploting cluster distribution...")
  tmp , dist = get_unique_class_and_count(clusters)
  dist.plot(kind="bar")
  plt.show()

  # plot the cluster assignments

  if one_vs_all:
    fig, axs = plt.subplots(nrows=8, ncols=1, figsize=(15,15))

    for index, ax in enumerate(axs):
      ax.scatter(x_scaled[:, left_column], x_scaled[:, index], c=clusters)
      ax.set_xlabel("Feature 0")
      ax.set_ylabel("Feature 1")
    plt.show()
  else:
    plt.scatter(x_scaled[:, left_column], x_scaled[:, right_column], c=clusters)
    plt.xlabel("Feature 0")
    plt.ylabel("Feature 1")
    plt.show()

  return dbscan_obj , clusters


def dbscan_all(df, showplots=True, scale=False, eps=0.5, min_samples = 2, left_column =
0, right_column=2):

  x = df.values

  if scale:
    scaler_obj, X_scaled = scaler_util(x)
    x = X_scaled

  if showplots:
    sns.pairplot(pd.DataFrame(x))
    plt.show()

  # cluster the data
  return perform_dbscan_all(x, eps=eps, min_samples = min_samples ,left_column = left_c
olumn, right_column=right_column)
```

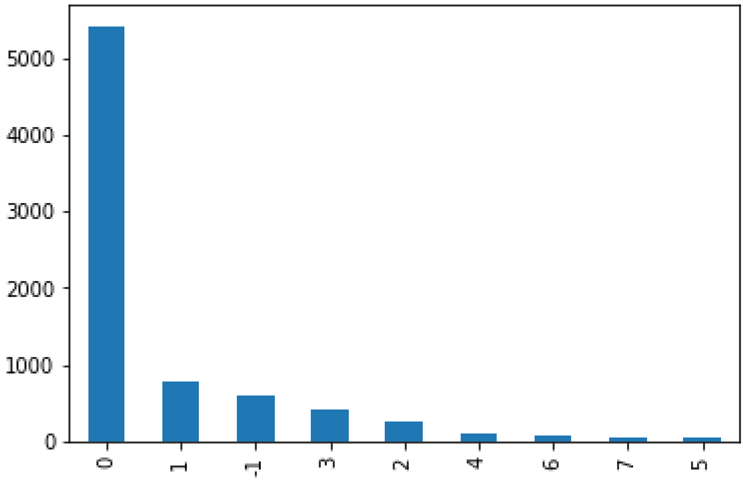
In [0]:

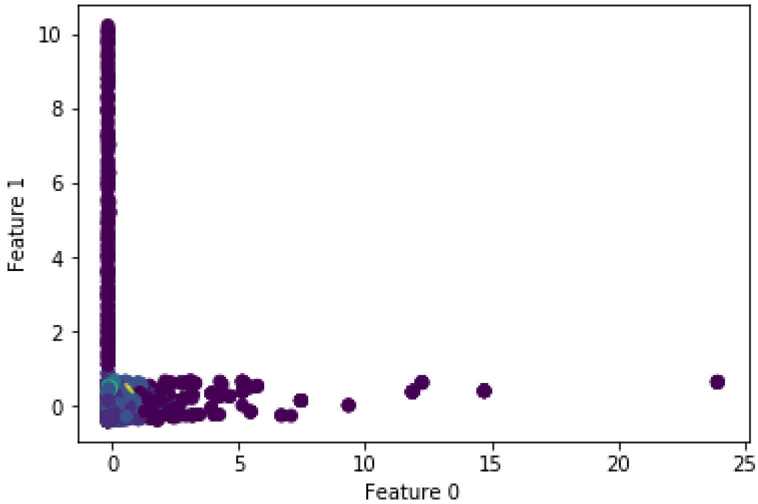
```
X_columns = ['Category', 'Reviews', 'Size', 'Installs', 'Type', 'Price','Content Ratin
g', 'Genres']
X = dataset[X_columns]
```

In [0]:

```
dbscan_obj, clust = dbscan_all(X, scale=True, showplots=False, min_samples=20, eps=0.8,
left_column = 1, right_column=2)
```

Ploting cluster distribution...

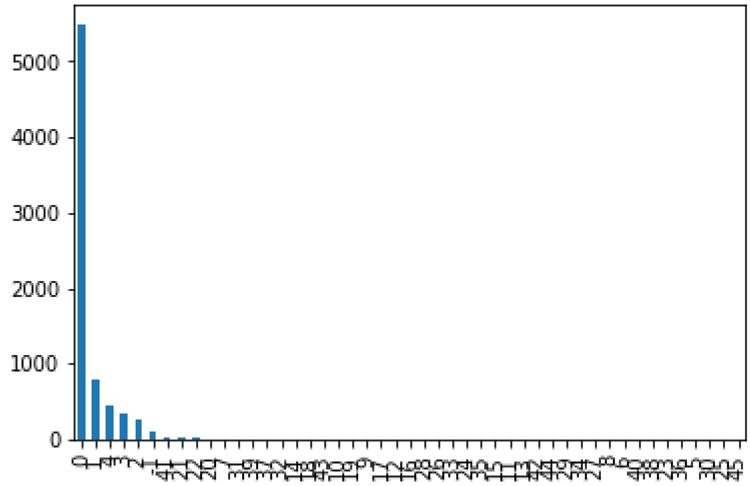


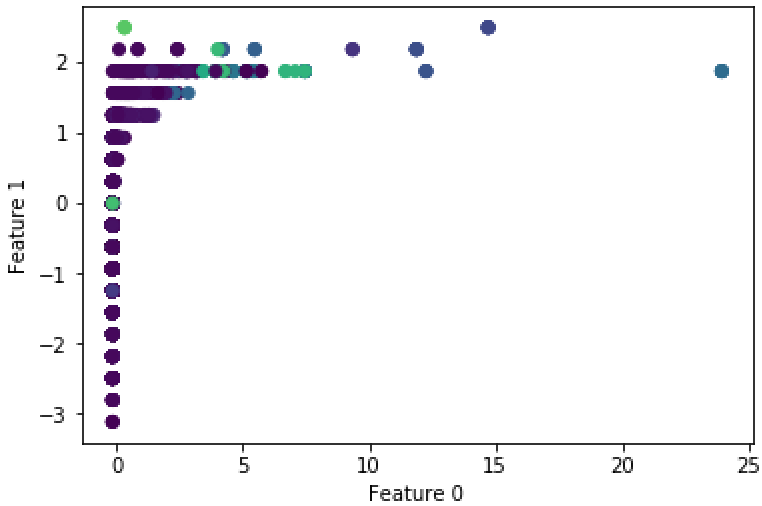


In [0]:

```
dbscan_obj, clusters = dbscan_all(X, scale=True, showplots=False, min_samples=3, eps=0.
8, left_column = 1, right_column=3)
```

```
Ploting cluster distribution...
```





In [0]:

```
# Number of clusters in labels, ignoring noise if present.
n_clusters_ = len(set(clusters)) - (1 if -1 in clusters else 0)
n_noise_ = list(clusters).count(-1)

print(f'Number of clusters = {n_clusters_}')
print(f'Number of noise sample = {n_noise_}')
```

```
Number of clusters = 46
Number of noise sample = 117
```

Using all columns, we observe that DBSCAN did not perform as well compared to KMeans. As you can see, it is sensitve to noise samples, and after tuning with different hyper parameters (epsilon and number of samples), we got different drastic results.

We will consider reducing the dimentionality from 8 Dim to 2 Dimension using the the next PCA Analysis. And we belive PCA will help to resolve this noise sentive variation issue.

Applying PCA

In [0]:

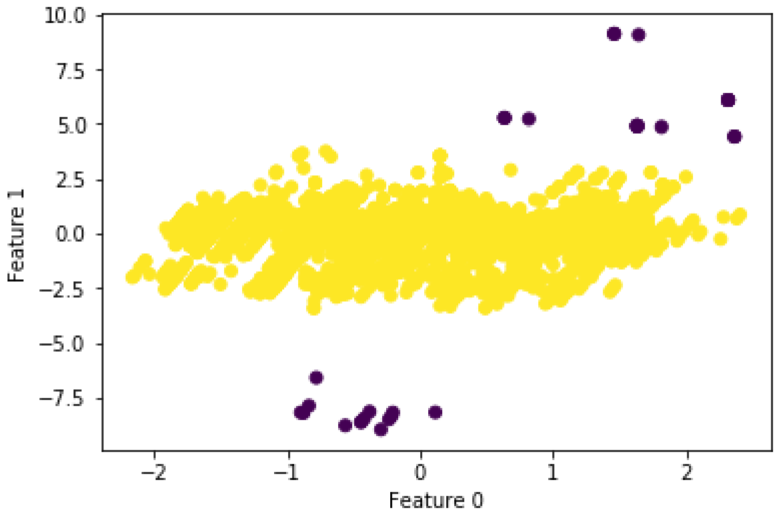
```
df = pd.read_csv('googleps_cleaned.csv')
```

In [0]:

```
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.cluster import DBSCAN
from sklearn.preprocessing import StandardScaler

def dbscan_with_pca(df, eps=0.8, min_samples = 15):
  features = ['Category', 'Reviews', 'Size', 'Installs', 'Type', 'Price', 'Content Rati
ng', 'Genres']

  x = df.loc[:, features].values
  y = df.loc[:,['Rating']].values

  x = StandardScaler().fit_transform(x)

  pca = PCA(n_components=2)
  principalComponents = pca.fit_transform(x)
  principalDf = pd.DataFrame(data = principalComponents
               , columns = ['principal component 1', 'principal component 2'])

  x = principalDf.values

  scaler = StandardScaler()
  X_scaled = scaler.fit_transform(x)

  # cluster the data
  dbscan = DBSCAN(eps=eps, min_samples = min_samples)
  clusters = dbscan.fit_predict(X_scaled)

  # plot the cluster assignments
  plt.scatter(X_scaled[:, 0], X_scaled[:, 1], c=clusters,)
  plt.xlabel("Feature 0")
  plt.ylabel("Feature 1")

  plt.show()
  return clusters
```

In [0]:

```
dbscan_with_pca(df)
```



In [0]:

```
pd.Series(y).value_counts()
```

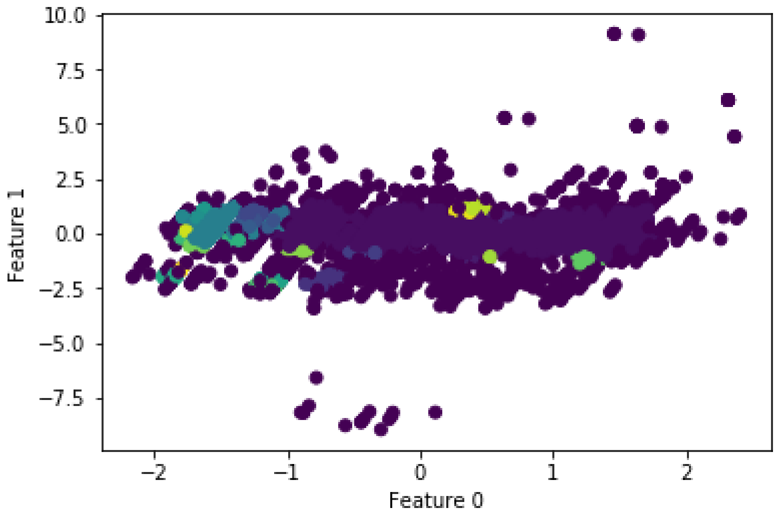
Out[0]:

```
High Rating       5923
Average Rating    1734
Low Rating          66
dtype: int64
```

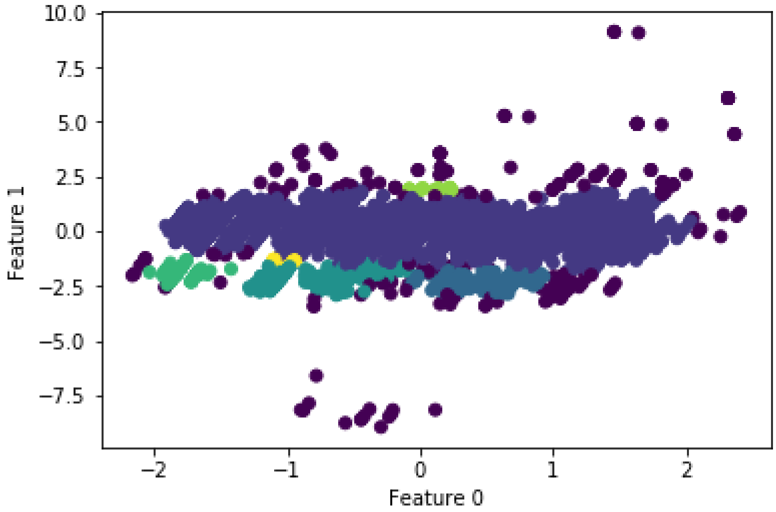
In [0]:

```
for eps in np.arange(0.1, .9 , 0.10):
  print("eps = ", eps)
  dbscan_with_pca(df, eps=eps)
```

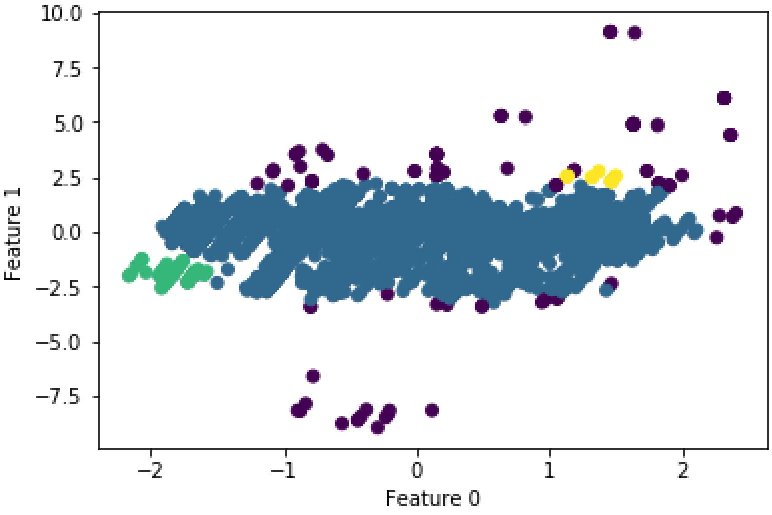
```
eps =  0.1
```



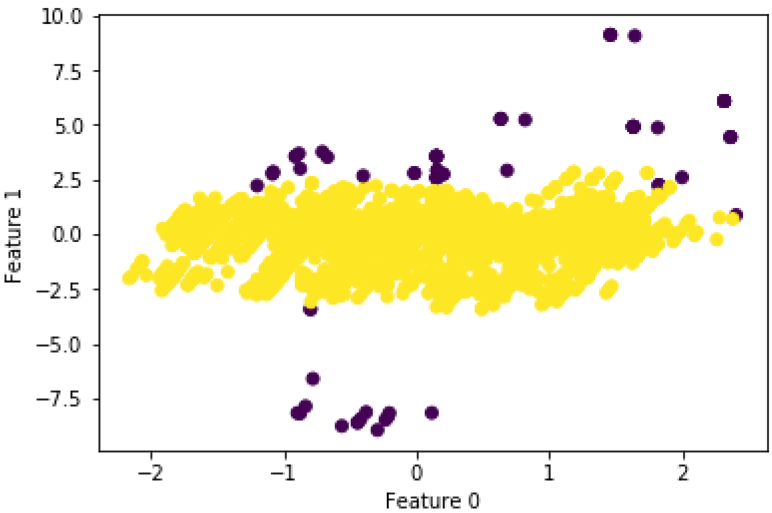

```
eps =  0.2
```



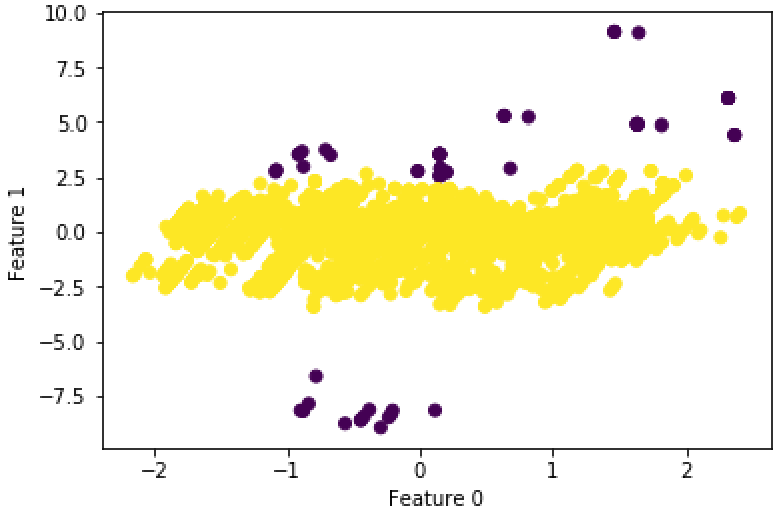

```
eps =  0.30000000000000004
```


10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
Feature 1
−2
−1
0
1
2
Feature 0

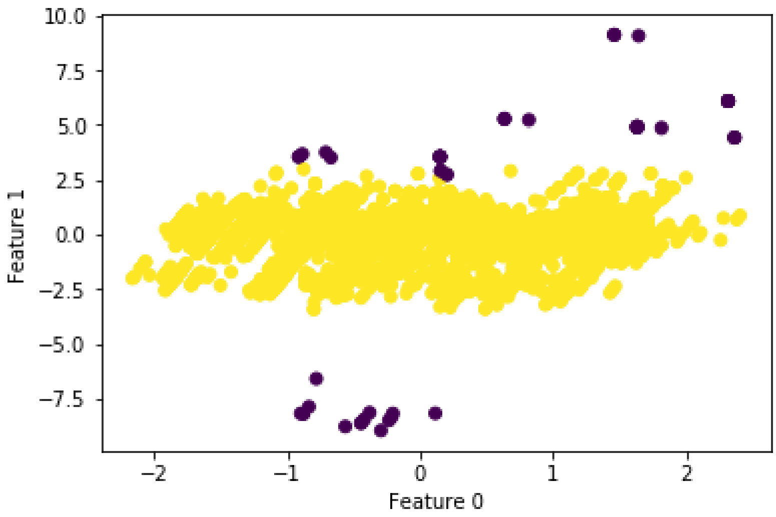
eps = 0.4


10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
Feature 1
−2
−1
0
1
2
Feature 0

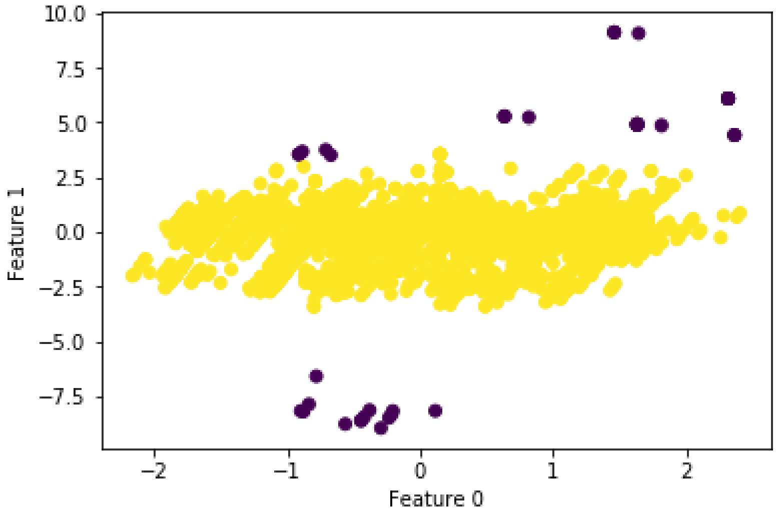
eps = 0.5



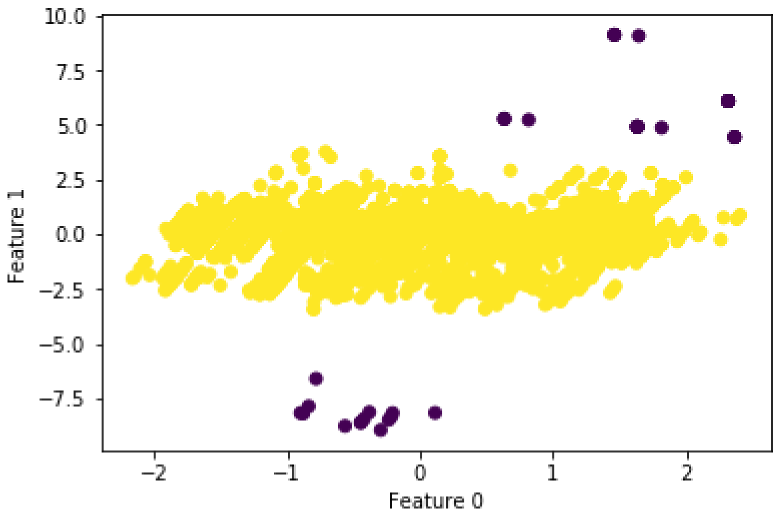

```
eps =  0.6
```




```
eps =  0.7000000000000001
```


10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
Feature 1
−2
−1
0
1
2
Feature 0

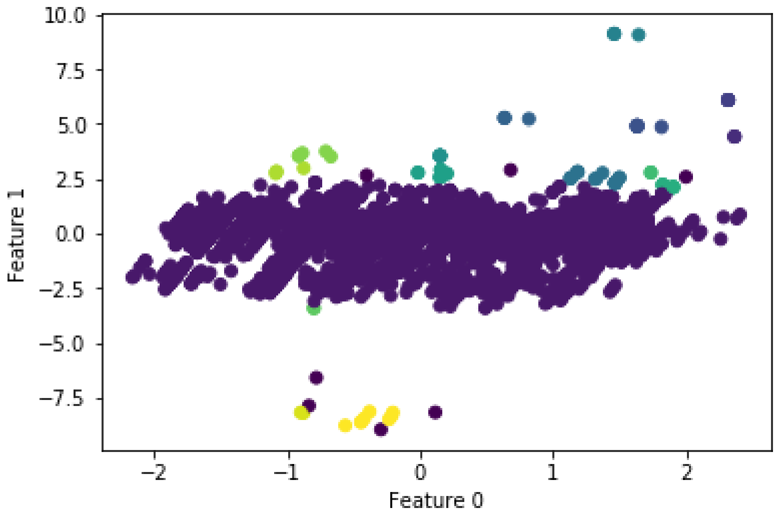
```
eps =  0.8
```


10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
Feature 1
−2
−1
0
1
2
Feature 0

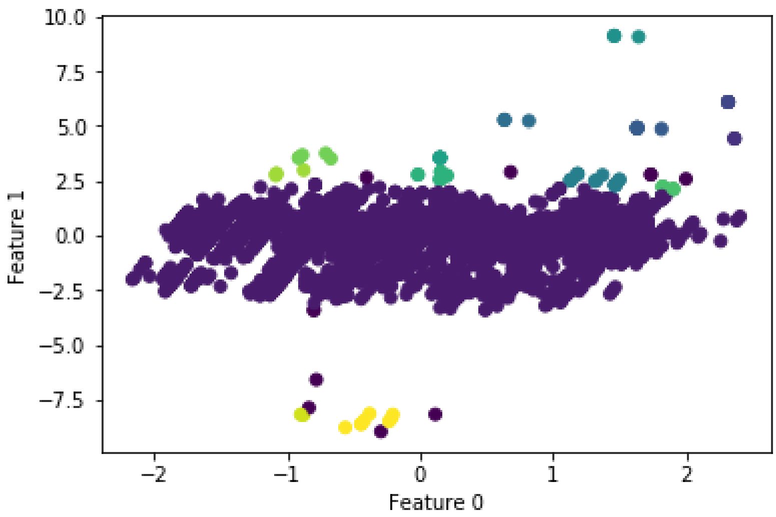
In [0]:

```
for min_s in np.arange(2,20):
  print("min_samples", min_s)
  dbscan_with_pca(df, eps=0.3, min_samples=min_s)
```

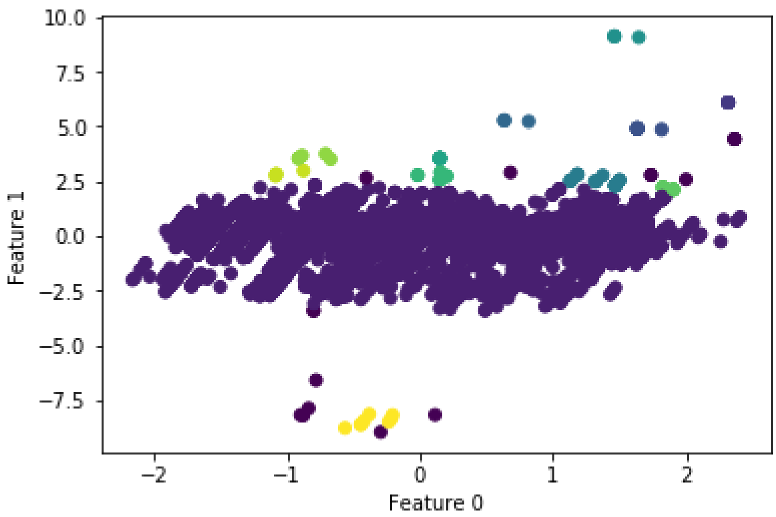
min_samples 2



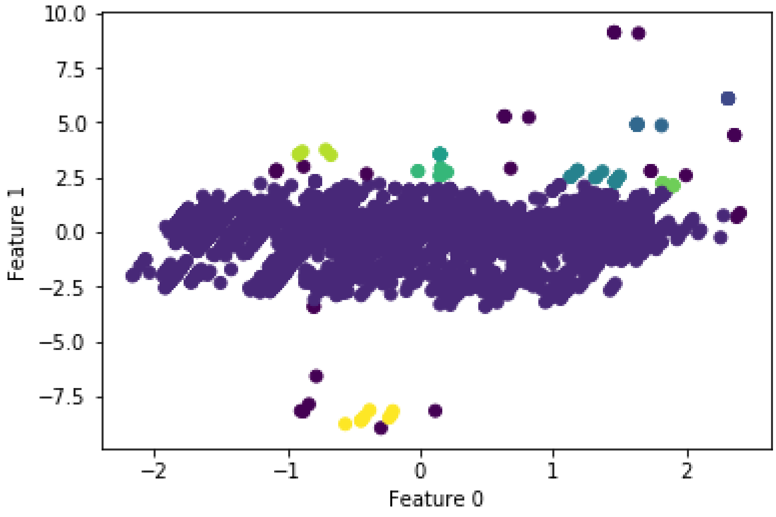
min_samples 3



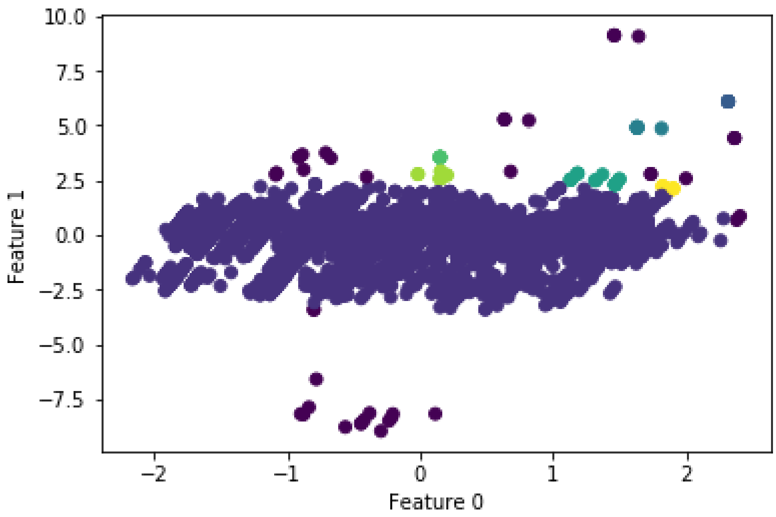
min_samples 4



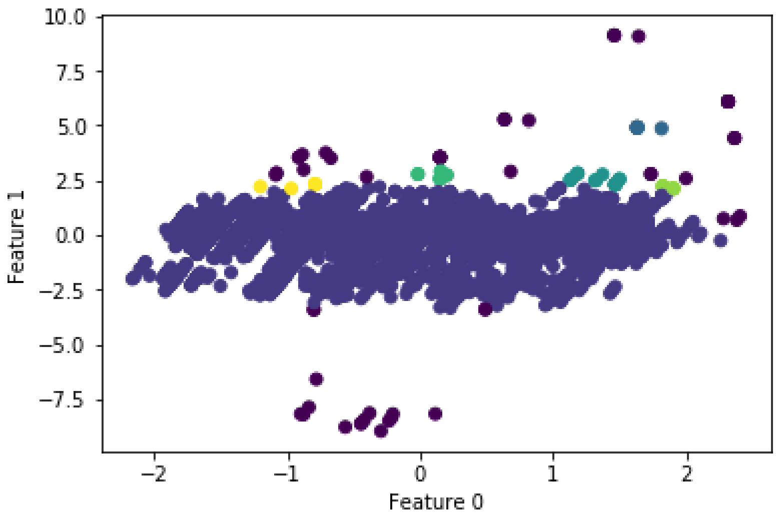
min_samples 5



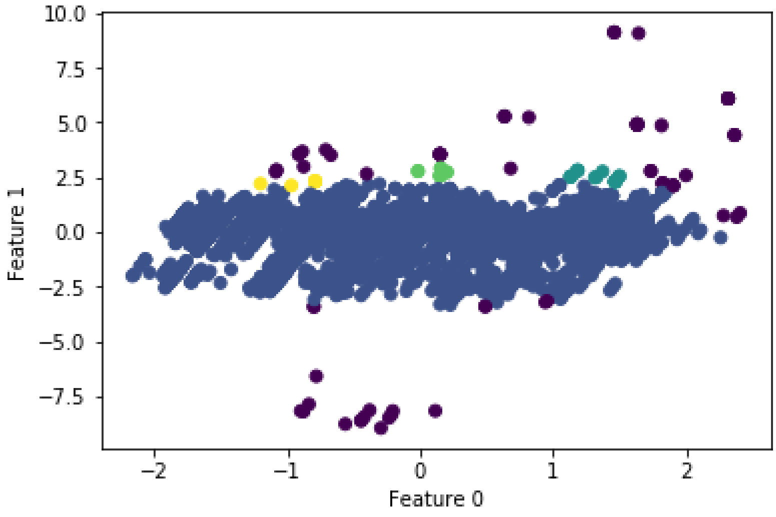
min_samples 6



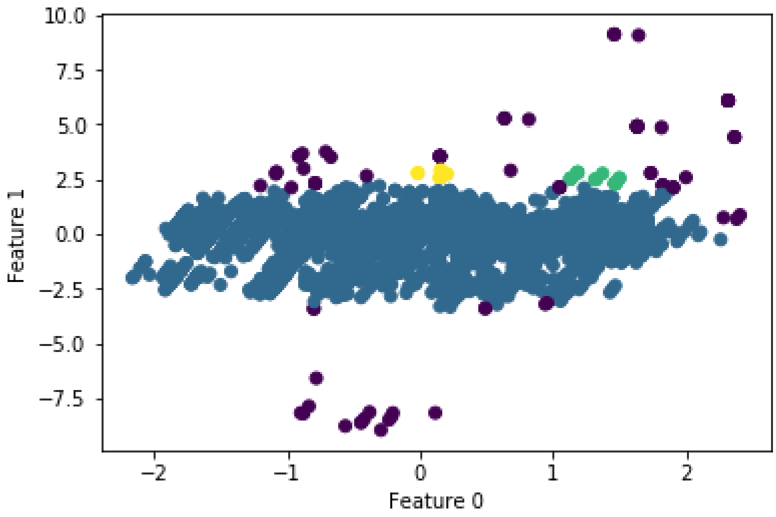
min_samples 7



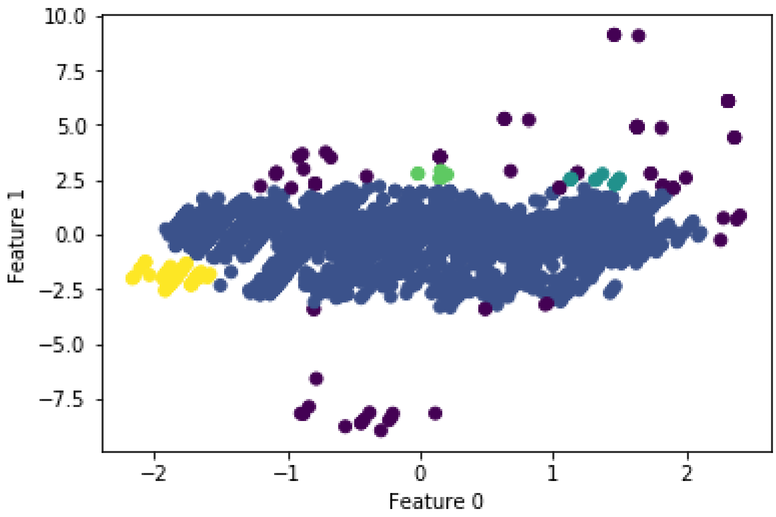
min_samples 8



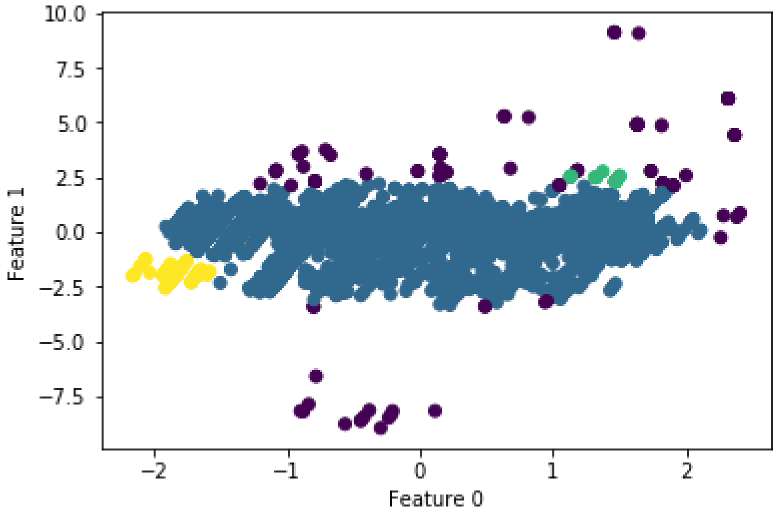
min_samples 9



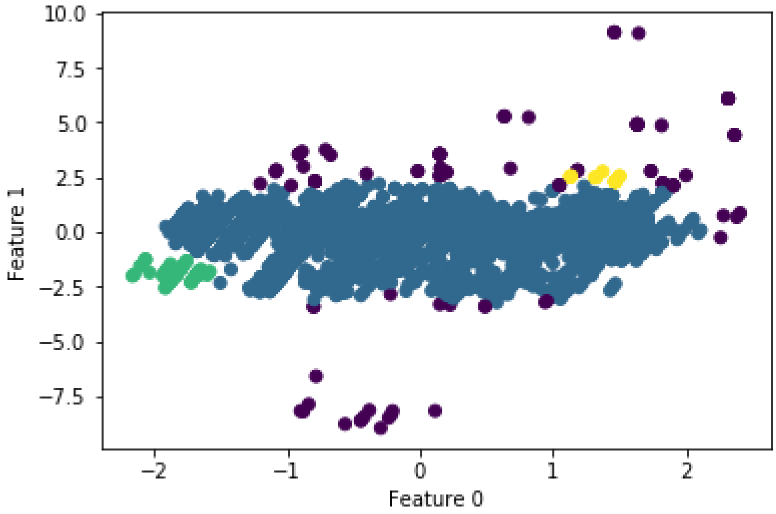
min_samples 10



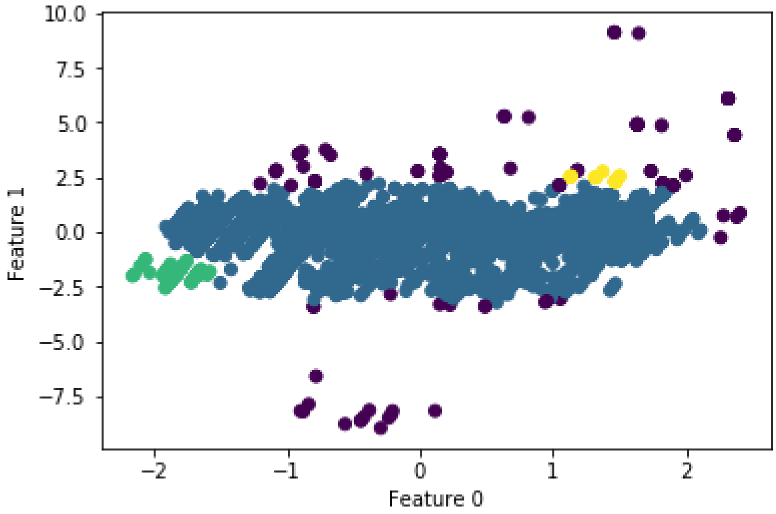
min_samples 11



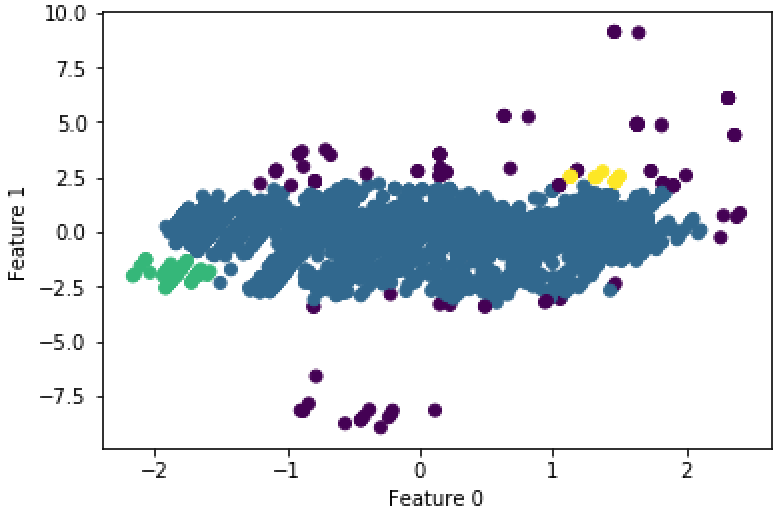
min_samples 12



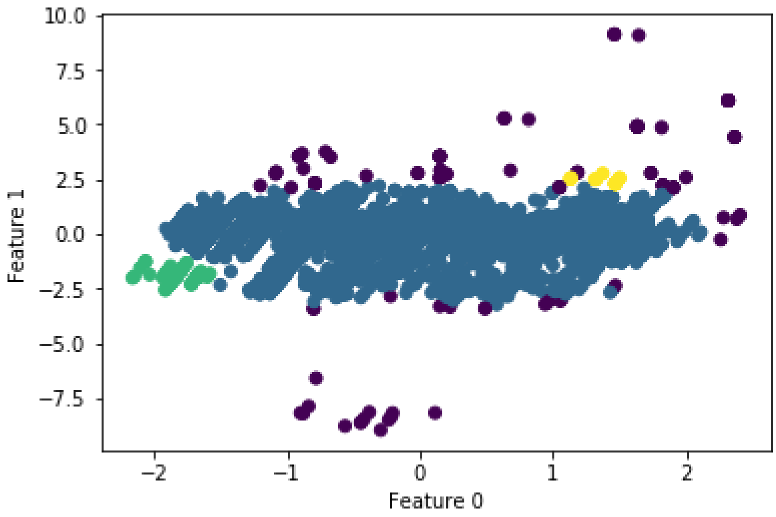
min_samples 13



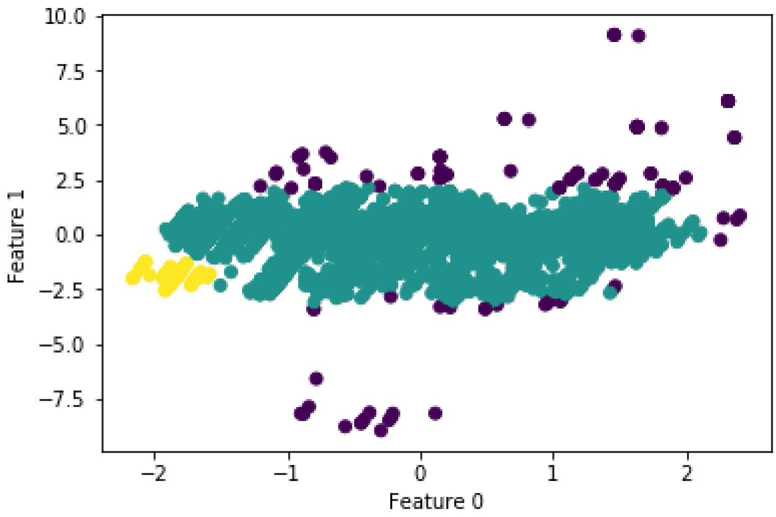
min_samples 14



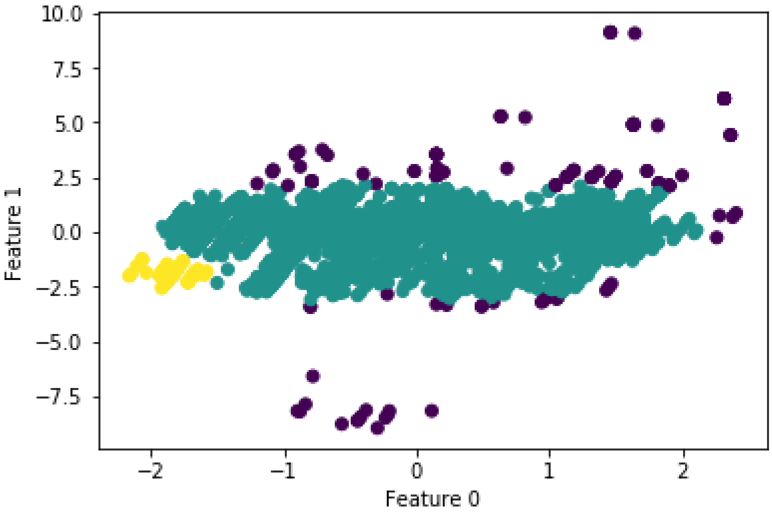

```
min_samples 15
```



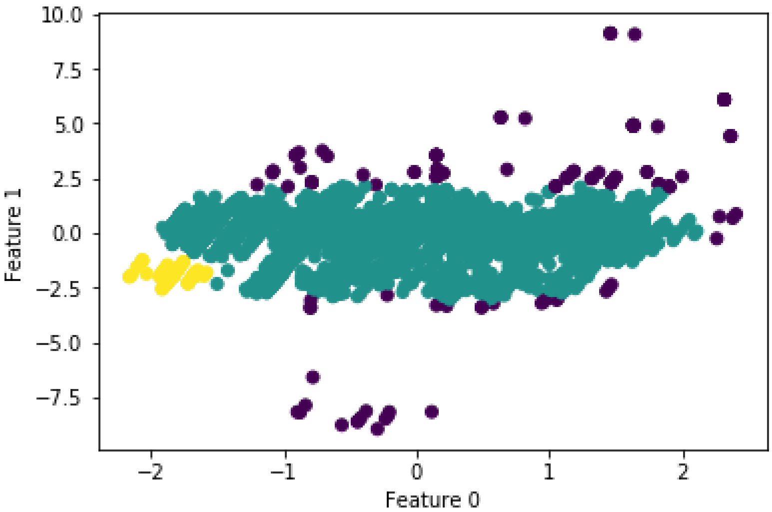

```
min_samples 16
```



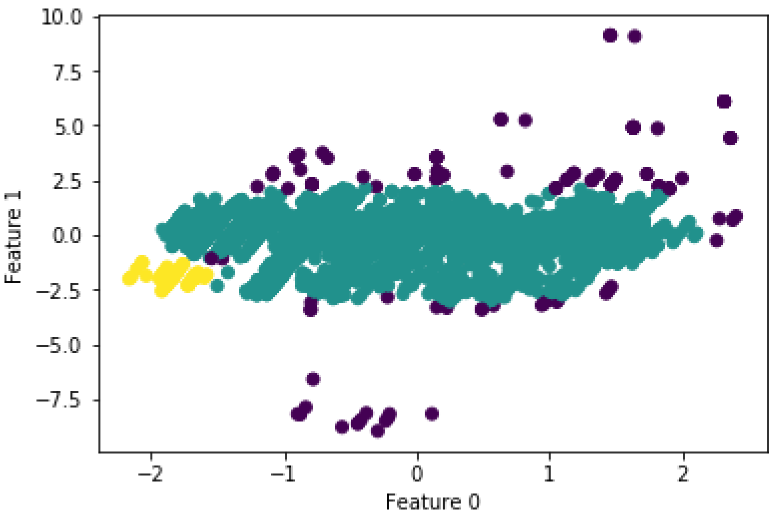

```
min_samples 17
```




```
min_samples 18
```

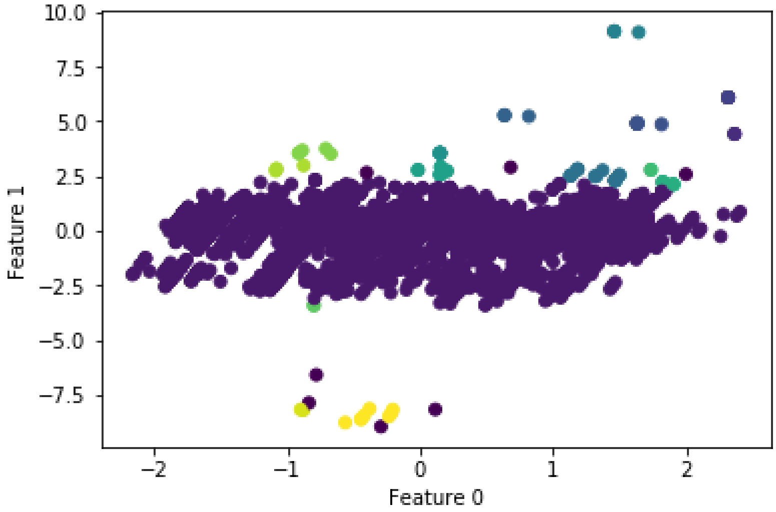



```
min_samples 19
```



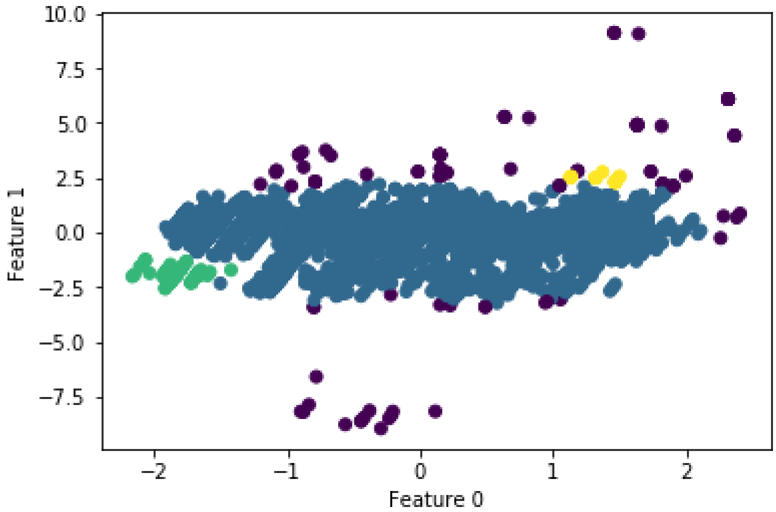
In [0]:

```
dbscan_with_pca(df, eps=0.3, min_samples=2)
```



In [0]:

```
clusters = dbscan_with_pca(df, eps=0.29, min_samples=13)
```



In [0]:

```
# Number of clusters in labels, ignoring noise if present.
n_clusters_ = len(set(clusters)) - (1 if -1 in clusters else 0)
n_noise_ = list(clusters).count(-1)

print(f'Number of clusters = {n_clusters_}')
print(f'Number of noise sample = {n_noise_}')
```

```
Number of clusters = 3
Number of noise sample = 104
```

Conclusion

After applying PCA we can see three cluster & and about 104 noise sample.

At the same time, there is one big cluster which is High Rating.