In [1]:

```
# Importing libraries

import numpy as np
import zipfile
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt


import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.layers import Dropout
from tensorflow.keras import callbacks
from keras.callbacks import ModelCheckpoint, TensorBoard
from keras.callbacks import EarlyStopping

# Clustering
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
from sklearn import mixture

# 3D Visualization
import plotly.express as px
import plotly as py
import plotly.graph_objs as go
```

In [2]:

```


from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import ShuffleSplit, train_test_split
from sklearn.metrics import confusion_matrix, precision_recall_curve
from sklearn.metrics import recall_score, classification_report, auc, roc_curve
from sklearn.metrics import precision_recall_fscore_support, f1_score

from numpy.random import seed
seed(7)
from tensorflow.random import set_seed
set_seed(11)
from sklearn.model_selection import train_test_split

SEED = 123 #used to help randomly select the data points
DATA_SPLIT_PCT = 0.2
```

In [3]:

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

In [4]:

```
# consider that sensor was having an issue and its giving the same reading for these tim
# I used forward fill propagation as a method of dealing with missing values

df.fillna(method = 'ffill' , inplace = True)
```

In [5]:

```
df.head()
```

Out[5]:

| | Unnamed: 0 | timestamp | sensor_00 | sensor_01 | sensor_02 | sensor_03 | sensor_04 | sensor_0 |
|---|---|---|---|---|---|---|---|---|
| **0** | 0 | 2018-04-01 00:00:00 | 2.465394 | 47.09201 | 53.2118 | 46.310760 | 634.3750 | 76.4597 |
| **1** | 1 | 2018-04-01 00:01:00 | 2.465394 | 47.09201 | 53.2118 | 46.310760 | 634.3750 | 76.4597 |
| **2** | 2 | 2018-04-01 00:02:00 | 2.444734 | 47.35243 | 53.2118 | 46.397570 | 638.8889 | 73.5459 |
| **3** | 3 | 2018-04-01 00:03:00 | 2.460474 | 47.09201 | 53.1684 | 46.397568 | 628.1250 | 76.9889 |
| **4** | 4 | 2018-04-01 00:04:00 | 2.445718 | 47.13541 | 53.2118 | 46.397568 | 636.4583 | 76.5889 |

5 rows × 55 columns

In [6]:

```
#Locate indices of failure events and recovering and normal state
df_n = df.loc[df['machine_status'] == 'NORMAL'].copy()
df_b = df.loc[df['machine_status'] == 'BROKEN'].copy()
df_r = df.loc[df['machine_status'] == 'RECOVERING'].copy()

df_n.index = df_n.timestamp
df_b.index = df_b.timestamp
df_r.index = df_r.timestamp
```

In [7]:

```
df_n.drop(columns=['Unnamed: 0', 'timestamp', 'machine_status','sensor_51','sensor_15'],
df_b.drop(columns=['Unnamed: 0', 'timestamp', 'machine_status','sensor_51','sensor_15'],
df_r.drop(columns=['Unnamed: 0', 'timestamp', 'machine_status','sensor_51','sensor_15'],
df_n.head()
```

Out[7]:

| timestamp | sensor_00 | sensor_01 | sensor_02 | sensor_03 | sensor_04 | sensor_05 | sensor_06 | sensor_07 | s |
|---|---|---|---|---|---|---|---|---|---|
| 2018-04-01 00:00:00 | 2.465394 | 47.09201 | 53.2118 | 46.310760 | 634.3750 | 76.45975 | 13.41146 | 16.13136 | |
| 2018-04-01 00:01:00 | 2.465394 | 47.09201 | 53.2118 | 46.310760 | 634.3750 | 76.45975 | 13.41146 | 16.13136 | |
| 2018-04-01 00:02:00 | 2.444734 | 47.35243 | 53.2118 | 46.397570 | 638.8889 | 73.54598 | 13.32465 | 16.03733 | |
| 2018-04-01 00:03:00 | 2.460474 | 47.09201 | 53.1684 | 46.397568 | 628.1250 | 76.98898 | 13.31742 | 16.24711 | |

In [8]:

```
df_n.describe()
```

Out[8]:

| | sensor_00 | sensor_01 | sensor_02 | sensor_03 | sensor_04 | sens |
|---|---|---|---|---|---|---|
| count | 205836.000000 | 205836.000000 | 205836.000000 | 205836.000000 | 205836.000000 | 205836.0 |
| mean | 2.420479 | 48.186489 | 51.636921 | 44.167010 | 625.673663 | 75.3 |
| std | 0.242464 | 2.247538 | 1.878258 | 1.668233 | 50.820977 | 9.3 |
| min | 0.000000 | 0.000000 | 37.413190 | 33.289930 | 2.798032 | 0.0 |
| 25% | 2.444734 | 46.701390 | 50.651040 | 43.142361 | 628.240700 | 70.5 |
| 50% | 2.456539 | 48.263890 | 51.736111 | 44.357635 | 633.333313 | 75.6 |
| 75% | 2.499826 | 49.565970 | 52.907986 | 45.355900 | 637.963000 | 80.6 |
| max | 2.549016 | 56.727430 | 56.032990 | 48.220490 | 800.000000 | 99.9 |

8 rows × 50 columns

# Prepare data for LSTM

In [9]:

```
# Firts, obtain the array

input_X = df_n.values
n_features = input_X.shape[1]
```

In [10]:

```
def temporalize(X, lookback):

    '''
    Inputs
    X         A 2D numpy array ordered by time of shape:
              (n_observations x n_features)
    y         A 1D numpy array with indexes aligned with
              X, i.e. y[i] should correspond to X[i].
              Shape: n_observations.
    lookback  The window size to look back in the past
              records. Shape: a scalar.

    Output
    output_X  A 3D numpy array of shape:
              ((n_observations-lookback-1) x lookback x
              n_features)
    output_y  A 1D array of shape:
              (n_observations-lookback-1), aligned with X.
    '''

    output_X = []

    for i in range(len(X) - lookback - 1):
        t = []
        for j in range(1, lookback + 1):
            # Gather the past records upto the lookback period
            t.append(X[[(i + j + 1)], :])
        output_X.append(t)

    return np.squeeze(np.array(output_X))
```

In [11]:

```
lookback= 5
len(input_X)-lookback-1, range(1, lookback+1)
```

Out[11]:

```
(205830, range(1, 6))
```

In [12]:

```
X = temporalize(input_X, lookback)
```

In [13]:

```
X.shape, input_X.shape
# Samples * lookback * features ,  Samples * features
```

Out[13]:

```
((205830, 5, 50), (205836, 50))
```

# Train-Test Split

In [14]:

```
X_train, X_test = train_test_split(np.array(X), test_size=DATA_SPLIT_PCT, random_state=S
```

In [15]:

```
X_train.shape
```

Out[15]:

```
(164664, 5, 50)
```

# Data standardization

In [16]:

```
def flatten(X):
    '''
    Flatten a 3D array.

    Input
    X            A 3D array for lstm, where the array is sample x timesteps x features.

    Output
    flattened_X  A 2D array, sample x features.
    '''
    flattened_X = np.empty((X.shape[0], X.shape[2]))  # sample x features array.
    for i in range(X.shape[0]):
        flattened_X[i] = X[i, (X.shape[1]-1), :]
    return(flattened_X)
```

In [17]:

```
def scale(X, scaler):
    '''
    Scale 3D array.

    Inputs
    X            A 3D array for lstm, where the array is sample x timesteps x features.
    scaler       A scaler object, e.g., sklearn.preprocessing.StandardScaler, sklearn.pr

    Output
    X            Scaled 3D array.
    '''
    for i in range(X.shape[0]):
        X[i, :, :] = scaler.transform(X[i, :, :])

    return X
```

In [18]:

```
# Initialize a scaler using the training data.
scaler = StandardScaler().fit(flatten(X_train))
```

In [19]:

```
X_train_scaled = scale(X_train, scaler)
```

In [20]:

```
a = flatten(X_train_scaled)
print('colwise mean', np.mean(a, axis=0).round(6))
print('colwise variance', np.var(a, axis=0))
```

```
colwise mean [ 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.]
colwise variance [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.]
```

# Building architecture

In [21]:

```
X_train_scaled.shape
```

Out[21]:

```
(164664, 5, 50)
```

In [22]:

```
timesteps =  X_train_scaled.shape[1] # equal to the lookback
n_features =  X_train_scaled.shape[2] # equal to the columns

epochs = 30
batch = 64
lr = 0.0001
```

In [23]:

```
# Recurrent autoencoder

#Encoder

input_layer = keras.Input(shape=(timesteps,n_features))
encoding_layer = layers.LSTM(32, activation='tanh', return_sequences=True)(input_layer)
hidden_layer = layers.LSTM(16,activation='tanh', return_sequences=False)(encoding_layer)

compact_features = layers.RepeatVector(timesteps)(hidden_layer)

#encoder_2d = keras.Model(inputs=input_layer, outputs = compact_features)

# Decoder

decoding_1layer = layers.LSTM(32, activation='tanh', return_sequences=True)(compact_feat
decoding_2layer = layers.LSTM(40, activation='tanh', return_sequences=True)(decoding_1la

decoding_3layer = layers.Dense(n_features)
output_layer = layers.TimeDistributed(decoding_3layer)(decoding_2layer)

autoencoder = keras.Model(inputs=input_layer, outputs=output_layer)
autoencoder.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 5, 50)]           0

 lstm (LSTM)                 (None, 5, 32)             10624

 lstm_1 (LSTM)               (None, 16)                3136

 repeat_vector (RepeatVecto  (None, 5, 16)             0
 r)

 lstm_2 (LSTM)               (None, 5, 32)             6272

 lstm_3 (LSTM)               (None, 5, 40)             11680

 time_distributed (TimeDist  (None, 5, 50)             2050
 ributed)

=================================================================
Total params: 33762 (131.88 KB)
Trainable params: 33762 (131.88 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

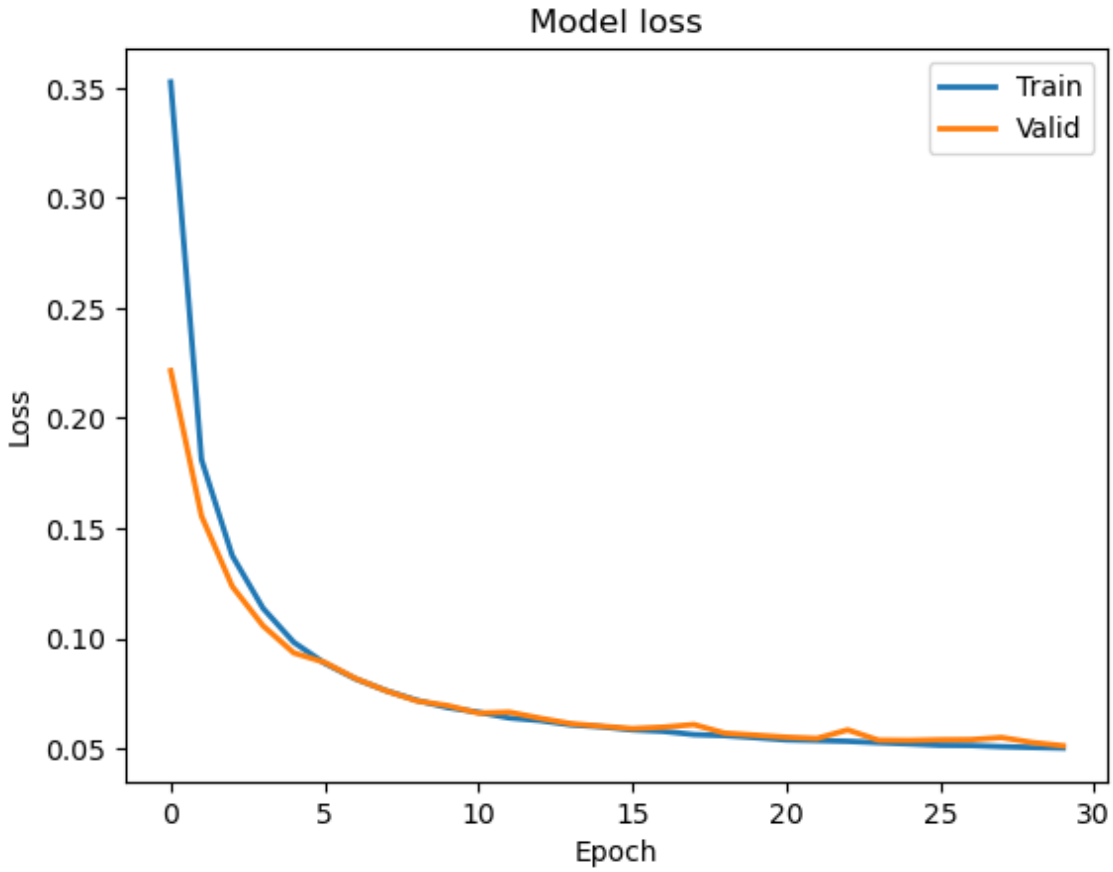
In [24]:

```
autoencoder.compile(loss='mse', optimizer='adam')

cp = ModelCheckpoint(filepath="lstm_autoencoder_classifier.h5",
                               save_best_only=True,
                               verbose=0)

tb = TensorBoard(log_dir='./logs',
                 histogram_freq=0,
                 write_graph=True,
                 write_images=True)



lstm_autoencoder = autoencoder.fit(X_train_scaled, X_train_scaled,
                               validation_split = 0.33,
                                epochs = epochs,
                                #callbacks = [E_Stop],
                                batch_size = batch,
                                shuffle = True,
                                verbose=2
                                )
```

```
Epoch 1/30
1724/1724 - 48s - loss: 0.3526 - val_loss: 0.2215 - 48s/epoch - 28ms/st
ep
Epoch 2/30
1724/1724 - 37s - loss: 0.1812 - val_loss: 0.1556 - 37s/epoch - 21ms/st
ep
Epoch 3/30
1724/1724 - 36s - loss: 0.1375 - val_loss: 0.1236 - 36s/epoch - 21ms/st
ep
Epoch 4/30
1724/1724 - 34s - loss: 0.1137 - val_loss: 0.1057 - 34s/epoch - 19ms/st
ep
Epoch 5/30
1724/1724 - 34s - loss: 0.0982 - val_loss: 0.0934 - 34s/epoch - 19ms/st
ep
Epoch 6/30
1724/1724 - 35s - loss: 0.0887 - val_loss: 0.0891 - 35s/epoch - 20ms/st
ep
Epoch 7/30
```

In [25]:

```
plt.plot(lstm_autoencoder.history['loss'], linewidth=2, label='Train')
plt.plot(lstm_autoencoder.history['val_loss'], linewidth=2, label='Valid')
plt.legend(loc='upper right')
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.show()
```




# Running just the Encoder

In [26]:

```
LSTM_encoder = keras.Model(inputs=input_layer, outputs = compact_features)
LSTM_encoder.summary()
```

```
Model: "model_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 5, 50)]           0

 lstm (LSTM)                 (None, 5, 32)             10624

 lstm_1 (LSTM)               (None, 16)                3136

 repeat_vector (RepeatVecto  (None, 5, 16)             0
 r)

=================================================================
Total params: 13760 (53.75 KB)
Trainable params: 13760 (53.75 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

In [27]:

```
X_test_scaled = scale(X_test, scaler)
```

In [28]:

```
Result_LSTM = LSTM_encoder.predict(X_test_scaled)
```

```
1287/1287 [==============================] - 7s 4ms/step
```

In [29]:

```
Result_LSTM.shape
```

Out[29]:

```
(41166, 5, 16)
```

In [30]:

```
pca = PCA(n_components=2)
```

In [31]:

```
Result_1_LSTM = Result_LSTM.reshape(Result_LSTM.shape[0], -1)
```

In [32]:

```
Result_1_LSTM.shape
```

Out[32]:

```
(41166, 80)
```

In [33]:

```
F_Result_LSTM = pca.fit_transform(Result_1_LSTM)
```

In [34]:

```
pca.explained_variance_ratio_
```

Out[34]:

```
array([0.41728452, 0.32852784], dtype=float32)
```

In [35]:

```
F_Result_LSTM.shape
```

Out[35]:

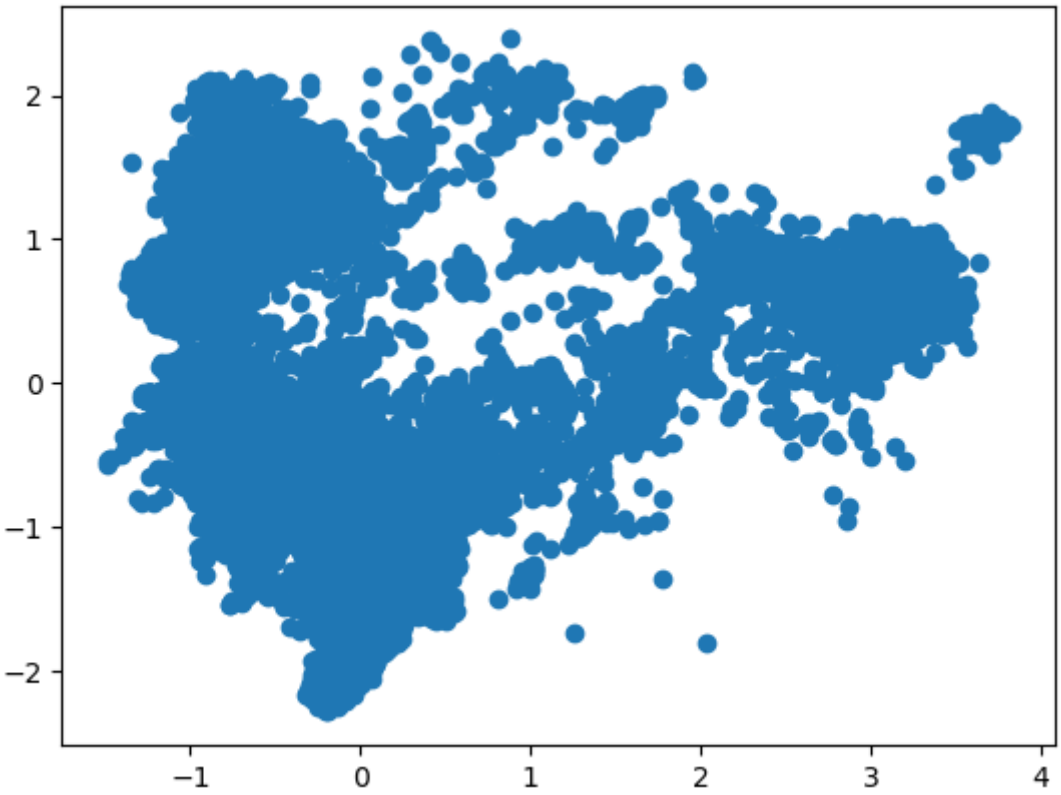
```
(41166, 2)
```

In [36]:

```
plt.scatter(x=F_Result_LSTM[:,0], y= F_Result_LSTM[:,1])
```

Out[36]:

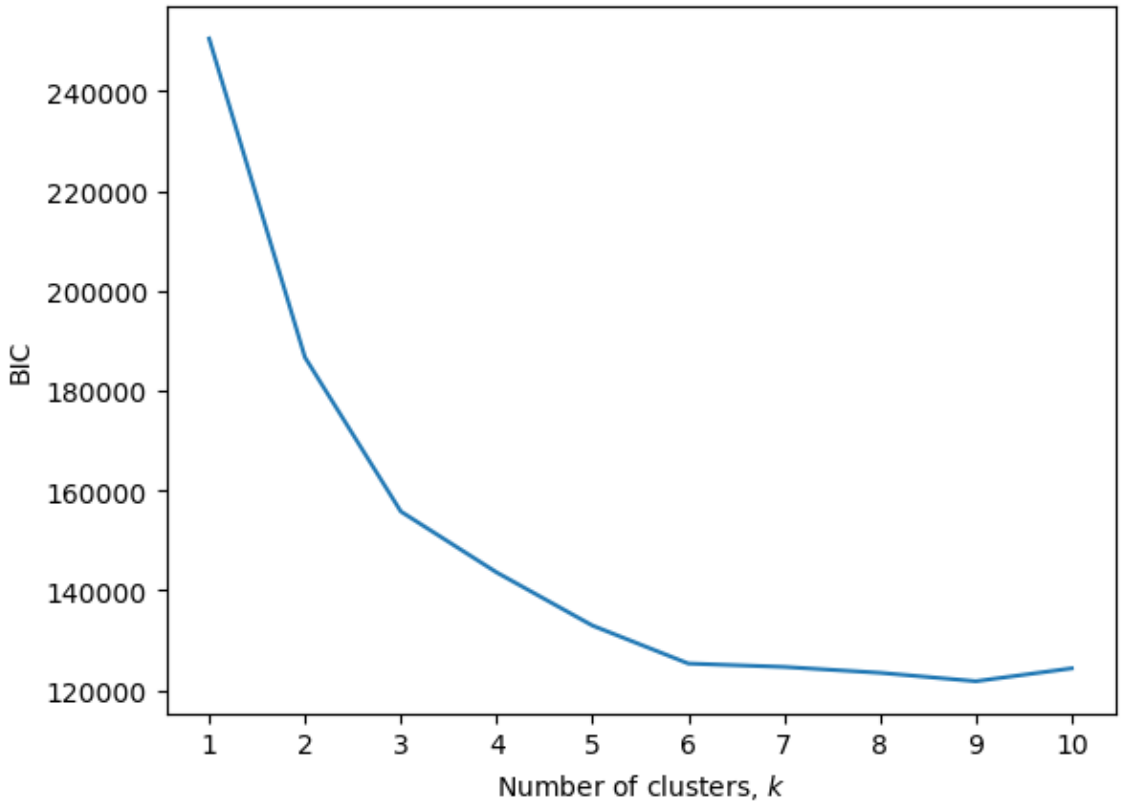
```
<matplotlib.collections.PathCollection at 0x1b489b1cb80>
```



# Using BIC method to find the proper number of clusters

In [37]:

```
ks = np.arange(1,11)
bics = []
for k in ks:
    gmm = mixture.GaussianMixture(n_components=k, covariance_type='full')
    gmm.fit(F_Result_LSTM)
    bics.append(gmm.bic(F_Result_LSTM))

# Plot the data
fig, ax = plt.subplots()
ax.plot(ks, bics)
ax.set_xlabel(r'Number of clusters, $k$')
ax.set_ylabel('BIC')
ax.set_xticks(ks);
```




# 2D Plot

In [38]:

```
F_Result_LSTM.shape
```

Out[38]:

```
(41166, 2)
```

In [39]:

```
df_FR_2d_LSTM = pd.DataFrame(F_Result_LSTM)
df_FR_2d_LSTM.columns = ['x','y']
df_FR_2d_LSTM.head()
```

Out[39]:

| | x | y |
|---|---|---|
| **0** | -0.195539 | -1.592148 |
| **1** | 2.816493 | 0.385995 |
| **2** | -0.633428 | 1.402577 |
| **3** | -0.755334 | 1.242345 |
| **4** | 2.695727 | 0.276364 |

In [40]:

```
Model = KMeans(n_clusters=5, init='k-means++', max_iter=1000)
Model.fit(df_FR_2d_LSTM)
```

```
C:\Users\user\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:870:
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 warning
  warnings.warn(
```

Out[40]:

```
KMeans(max_iter=1000, n_clusters=5)
```

**In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.**
**On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.**

In [41]:

```
df_FR_2d_LSTM['Cluster'] = Model.fit_predict(df_FR_2d_LSTM)
df_FR_2d_LSTM['labels'] = Model.labels_
centroids = Model.cluster_centers_
df_FR_2d_LSTM.head()
```

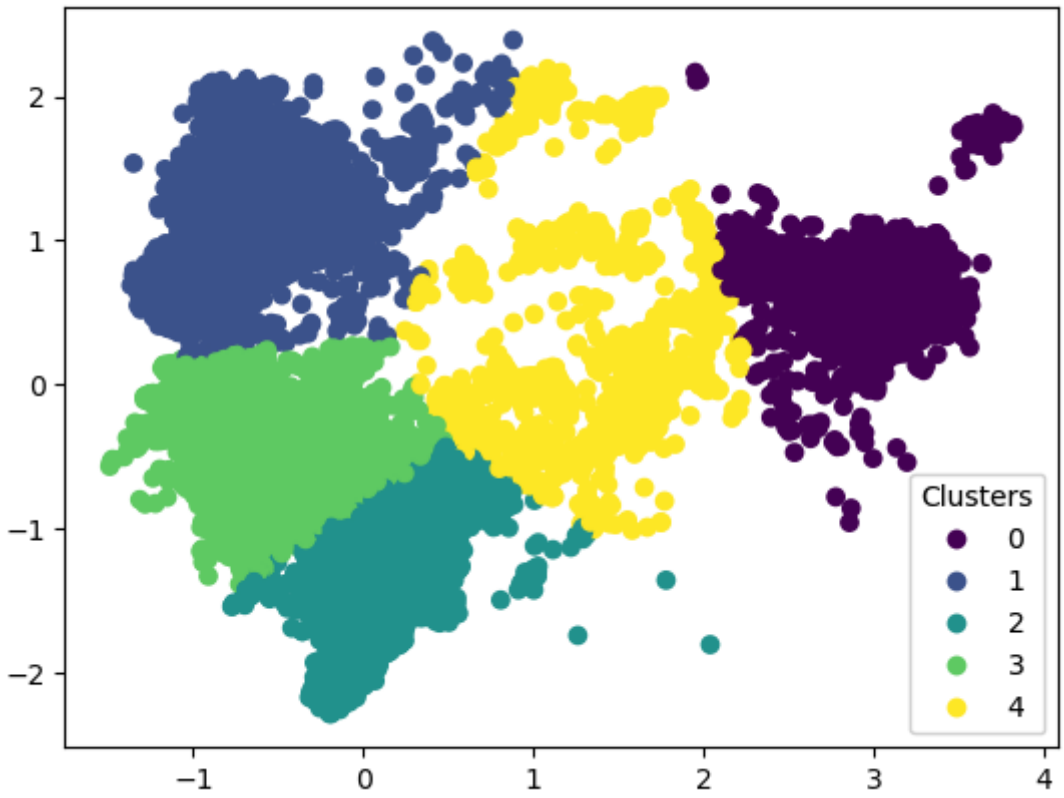
```
C:\Users\user\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:870:
FutureWarning: The default value of `n_init` will change from 10 to 'auto'
in 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
```

Out[41]:

| | x | y | Cluster | labels |
|---|---|---|---|---|
| **0** | -0.195539 | -1.592148 | 2 | 2 |
| **1** | 2.816493 | 0.385995 | 0 | 0 |
| **2** | -0.633428 | 1.402577 | 1 | 1 |
| **3** | -0.755334 | 1.242345 | 1 | 1 |
| **4** | 2.695727 | 0.276364 | 0 | 0 |

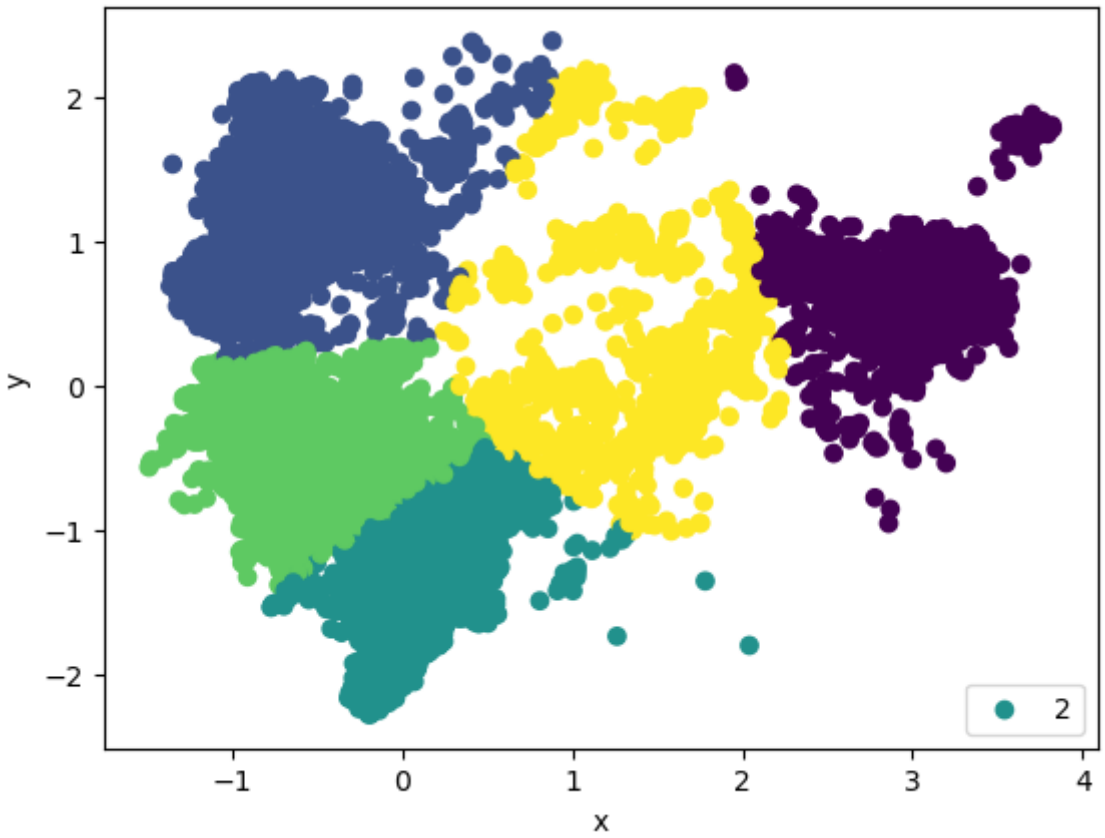
In [73]:

```
fig, ax = plt.subplots()

scatter = ax.scatter(df_FR_2d_LSTM['x']
                     ,df_FR_2d_LSTM['y'],
                     c=df_FR_2d_LSTM['Cluster']
                    )

legend1 = ax.legend(*scatter.legend_elements(num='auto'),
                    loc="lower right",
                    title="Clusters")

ax.add_artist(legend1)
plt.show()
```



In [69]:

```
plt.scatter(df_FR_2d_LSTM['x'], df_FR_2d_LSTM['y'], c=df_FR_2d_LSTM['Cluster'])
plt.ylabel('y')
plt.xlabel('x')
plt.legend(df_FR_2d_LSTM['Cluster']
           ,loc='lower right')

#plt.legend(['Cludster_1',
 #            'Cluster_2',
  #           'Cluster_3',
   #          'Cluster_4',
    #         'Cluster_5']
     #       ,loc='lower right'
      #      )
plt.show()
```




# 3D Plot

In [43]:

```
pca_3d = PCA(n_components=3)
```

In [44]:

```
Result_LSTM.shape
```

Out[44]:

```
(41166, 5, 16)
```

In [45]:

```
Result_1_LSTM.shape
```

Out[45]:

```
(41166, 80)
```

In [46]:

```
F_Result_3d_LSTM = pca_3d.fit_transform(Result_1_LSTM)
```

In [47]:

```
pca_3d.explained_variance_ratio_
```

Out[47]:

```
array([0.4172846 , 0.3285278 , 0.05525877], dtype=float32)
```

In [48]:

```
F_Result_3d_LSTM.shape
```

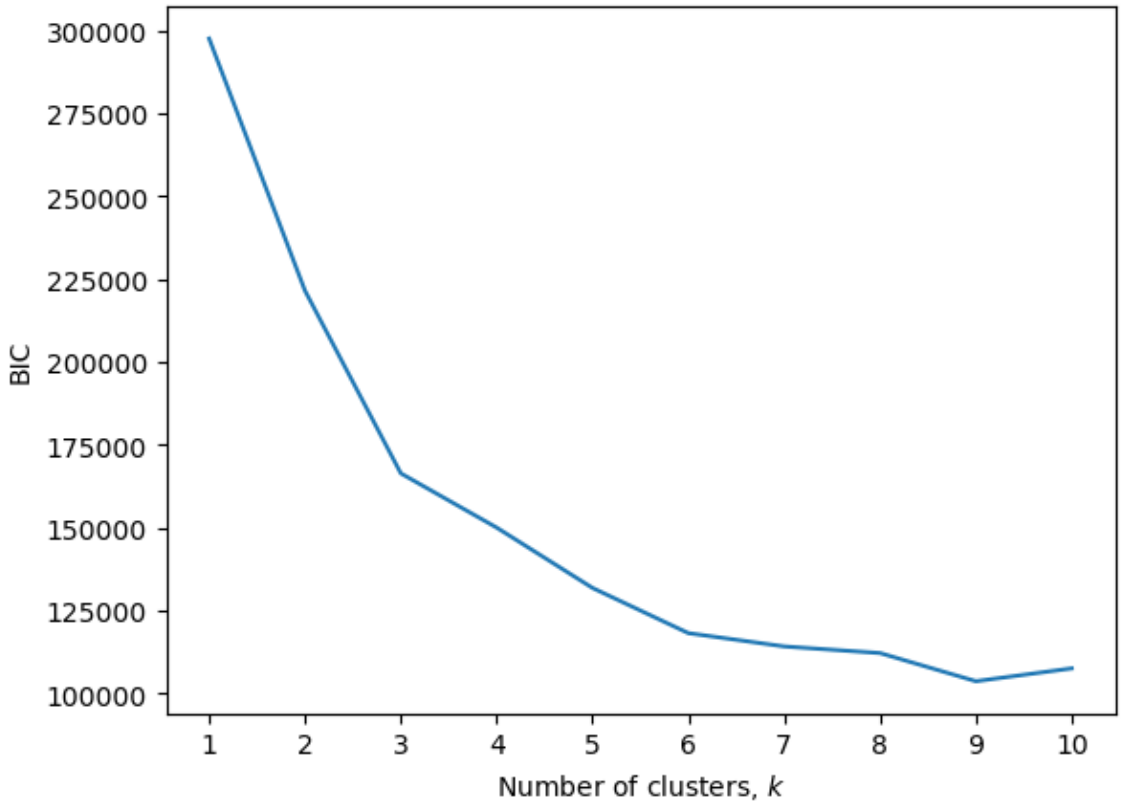
Out[48]:

```
(41166, 3)
```

# Number of clusters

In [49]:

```
ks = np.arange(1,11)
bics = []
for k in ks:
    gmm = mixture.GaussianMixture(n_components=k, covariance_type='full')
    gmm.fit(F_Result_3d_LSTM)
    bics.append(gmm.bic(F_Result_3d_LSTM))

# Plot the data
fig, ax = plt.subplots()
ax.plot(ks, bics)
ax.set_xlabel(r'Number of clusters, $k$')
ax.set_ylabel('BIC')
ax.set_xticks(ks);
```



In [50]:

```
# Creating DataFrame
df_FR_3d_LSTM = pd.DataFrame(F_Result_3d_LSTM)
df_FR_3d_LSTM.columns = ['x','y','z']
df_FR_3d_LSTM.head()
```

Out[50]:

| | x | y | z |
|---|---|---|---|
| **0** | -0.195546 | -1.592167 | 0.355916 |
| **1** | 2.816493 | 0.385994 | -0.305002 |
| **2** | -0.633427 | 1.402577 | 0.029464 |
| **3** | -0.755333 | 1.242345 | 0.058242 |
| **4** | 2.695728 | 0.276362 | -0.273730 |

In [51]:

```
# Working with 8 clusters
Model2 = KMeans(n_clusters=6, init='k-means++', max_iter=1000)
Model2.fit(df_FR_3d_LSTM)
```

```
C:\Users\user\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:870:
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 warning
  warnings.warn(
```

Out[51]:

```
KMeans(max_iter=1000, n_clusters=6)
```

**In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.**
**On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.**

In [52]:

```
df_FR_3d_LSTM['Cluster'] = Model2.fit_predict(df_FR_3d_LSTM)
df_FR_3d_LSTM['labels'] = Model2.labels_
centroids = Model2.cluster_centers_
df_FR_3d_LSTM.head()
```

```
C:\Users\user\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:870:
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 warning
  warnings.warn(
```

Out[52]:

| | x | y | z | Cluster | labels |
|---|---|---|---|---|---|
| **0** | -0.195546 | -1.592167 | 0.355916 | 1 | 1 |
| **1** | 2.816493 | 0.385994 | -0.305002 | 2 | 2 |
| **2** | -0.633427 | 1.402577 | 0.029464 | 0 | 0 |
| **3** | -0.755333 | 1.242345 | 0.058242 | 0 | 0 |
| **4** | 2.695728 | 0.276362 | -0.273730 | 2 | 2 |

# Graph

In [53]:

```
#df_FR_3d['labels'] = labels

trace = go.Scatter3d(
                    x = df_FR_3d_LSTM['x'],
                    y = df_FR_3d_LSTM['y'],
                    z = df_FR_3d_LSTM['z'],
                    mode = 'markers',
                    marker=dict(color=df_FR_3d_LSTM['labels'], size = 5, line=dict(color
                    )

data = [trace]
layout = go.Layout(
                    title='Clusters',
                    scene= dict(
                                xaxis = dict(title = 'x'),
                                yaxis = dict(title = 'y'),
                                zaxis = dict(title = 'z'))
                                )

fig = go.Figure(data=data, layout=layout)
py.offline.iplot(fig)
plt.show()
```

In [ ]:

```
```