Etudiants :

GUICHENITI Hamoud

SIDI YAHYA Mohamed Ali

ZEMALACHE ilyes

Loading Data

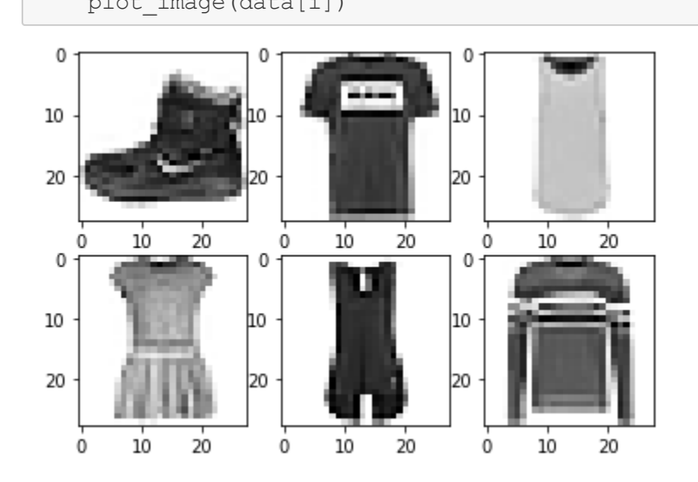
In [0]:
```python
import tensorflow as tf
from sklearn.decomposition import PCA
import numpy as np
from random import shuffle
import matplotlib.pyplot as plt
from sklearn.manifold import TSNE
from sklearn.cluster import KMeans
from sklearn.metrics.cluster import normalized_mutual_info_score as nmi
from sklearn.metrics.cluster import adjusted_rand_score as ari
import warnings
warnings.filterwarnings('ignore')
```

In [0]:
```python
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.fashion_mnist.load_data()
x_train = x_train.reshape(x_train.shape[0],x_train.shape[1]*x_train.shape[1])/255    # 60000 X 784
x_test = x_test.reshape(x_test.shape[0],x_test.shape[1]*x_test.shape[1])/255         # 10000 X 784
```

In [0]:
```python
data =np.concatenate((x_train,x_test))    # 70000 X 784
target=np.concatenate((y_train,y_test))   # 70000,
```

In [0]:
```python
# Simple fonction pour ploter une image
def plot_image(img):
  img=img*255
  img=img.reshape(28,28)
  plt.imshow(img,cmap="Greys")
```

In [6]:
```python
s=230
for i in range(6):
    s=s+1
    plt.subplot(s)
    plot_image(data[i])
```



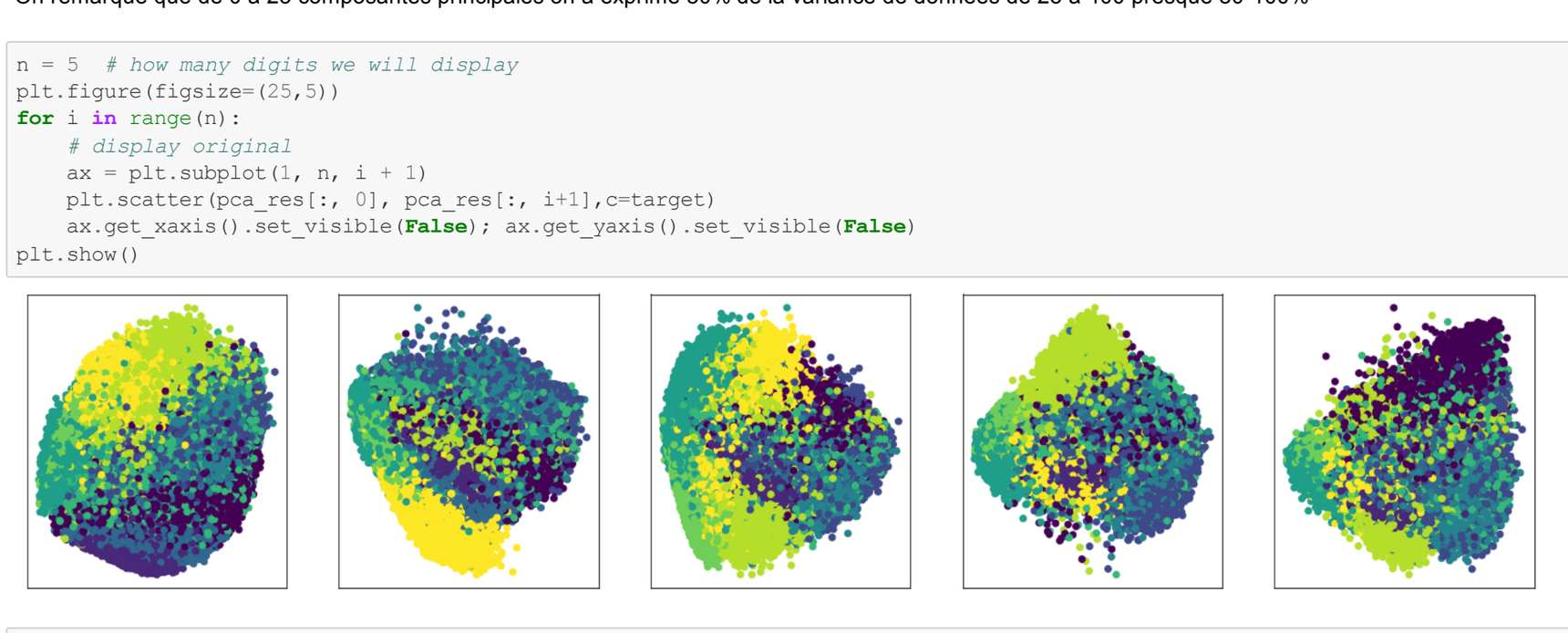
Reduction de dimension pour Fashion-MNIST avec PCA

In [0]:
```python
pca=PCA()
pca_res=pca.fit_transform(data)
```

In [8]:
```python
#Exprimer la variance en fonction de nombres de composantes principales
plt.plot(np.cumsum(pca.explained_variance_ratio_[:400]))
plt.xlabel('Nombre de composantes')
plt.ylabel('Variance Cumulé');
```

On remarque que de 0 à 25 composantes principales on a exprimé 80% de la variance de données de 25 a 400 presque 80-100%

In [9]:
```python
n = 5  # how many digits we will display
plt.figure(figsize=(25,5))
for i in range(n):
    # display original
    ax = plt.subplot(1, n, i + 1)
    plt.scatter(pca_res[:, 0], pca_res[:, i+1],c=target)
    ax.get_xaxis().set_visible(False); ax.get_yaxis().set_visible(False)
plt.show()
```

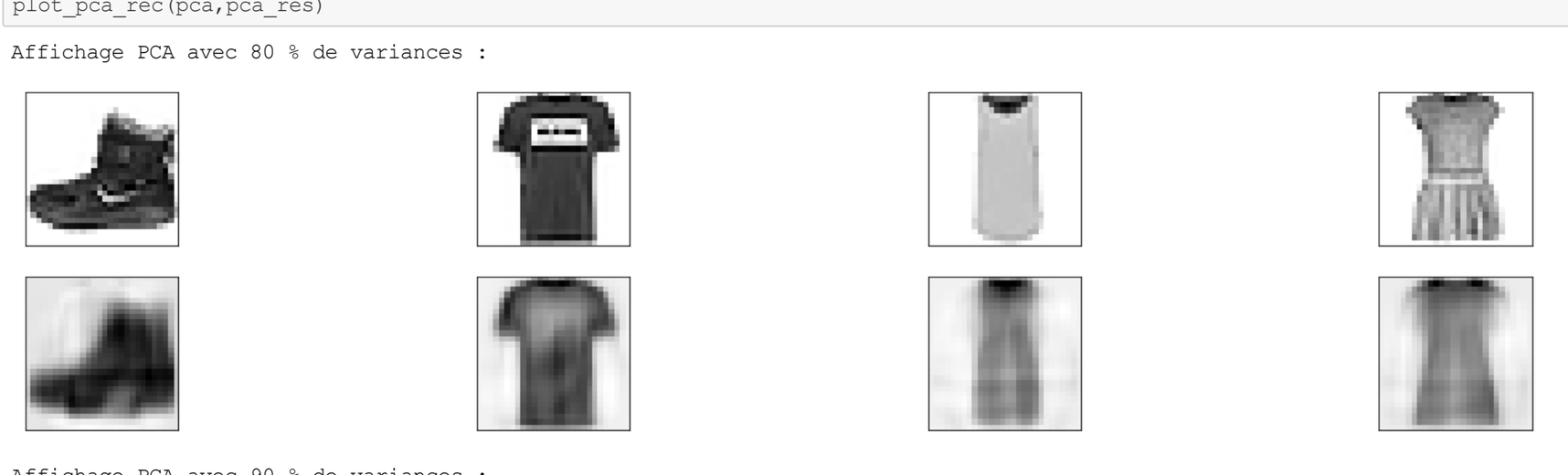


In [0]:
```python
pca25=PCA(n_components=25)
pca25_res =pca25.fit_transform(data)

pca100=PCA(n_components=100)
pca100_res =pca100.fit_transform(data)
```

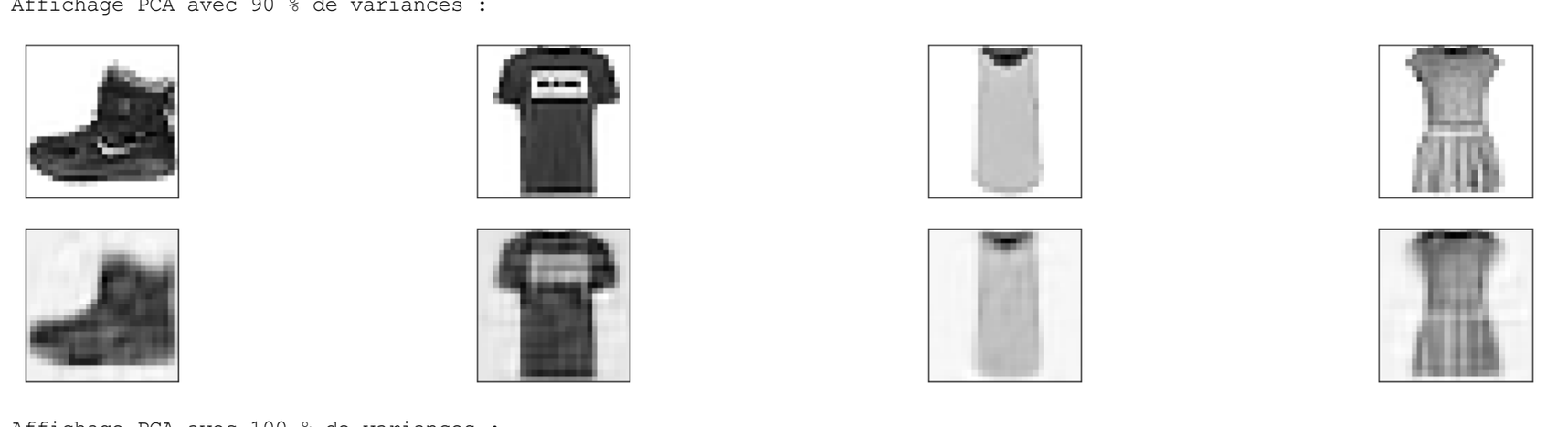
In [0]:
```python
def plot_pca_rec(pca,pca_res):
    n = 4  # how many digits we will display
    plt.figure(figsize=(25,5))
    for i in range(n):

        # display original
        ax = plt.subplot(2, n, i + 1)
        plt.imshow(data[i].reshape(28, 28), cmap='Greys')
        ax.get_xaxis().set_visible(False)
        ax.get_yaxis().set_visible(False)

        # display reconstruction
        ax = plt.subplot(2, n, i + 1 + n)
        plt.imshow(pca.inverse_transform(pca_res)[i].reshape(28, 28), cmap='Greys')
        ax.get_xaxis().set_visible(False)
        ax.get_yaxis().set_visible(False)
    plt.show()
```

In [12]:
```python
print("Affichage PCA avec 80 % de variances : ")
plot_pca_rec(pca25,pca25_res)
print("Affichage PCA avec 90 % de variances : ")
plot_pca_rec(pca100,pca100_res)
print("Affichage PCA avec 100 % de variances : ")
plot_pca_rec(pca,pca_res)
```

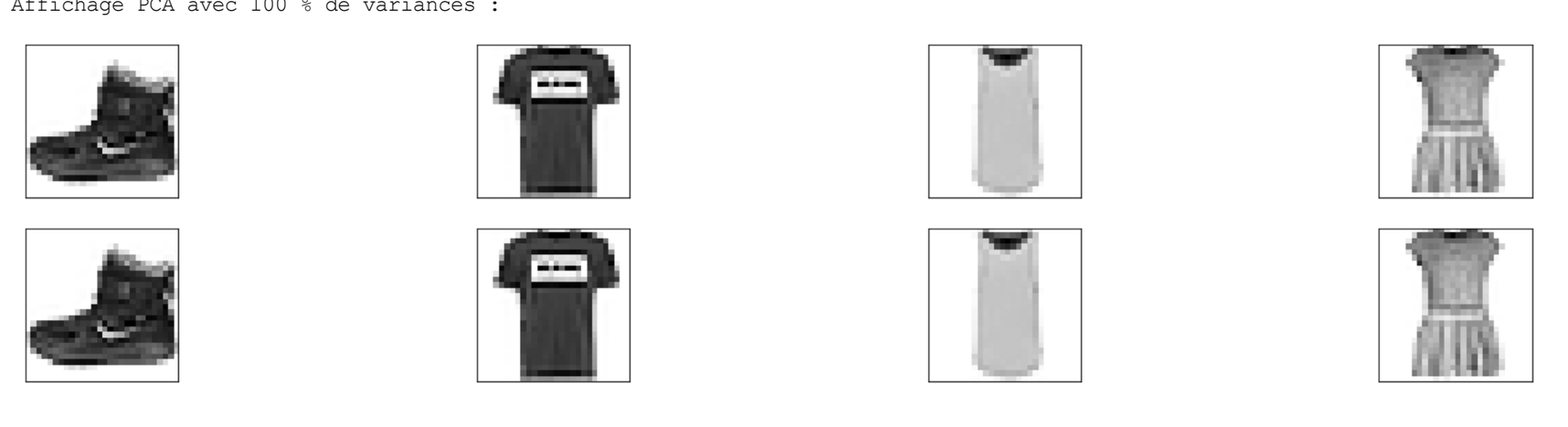
```
Affichage PCA avec 80 % de variances :
```



```
Affichage PCA avec 90 % de variances :
```



```
Affichage PCA avec 100 % de variances :
```



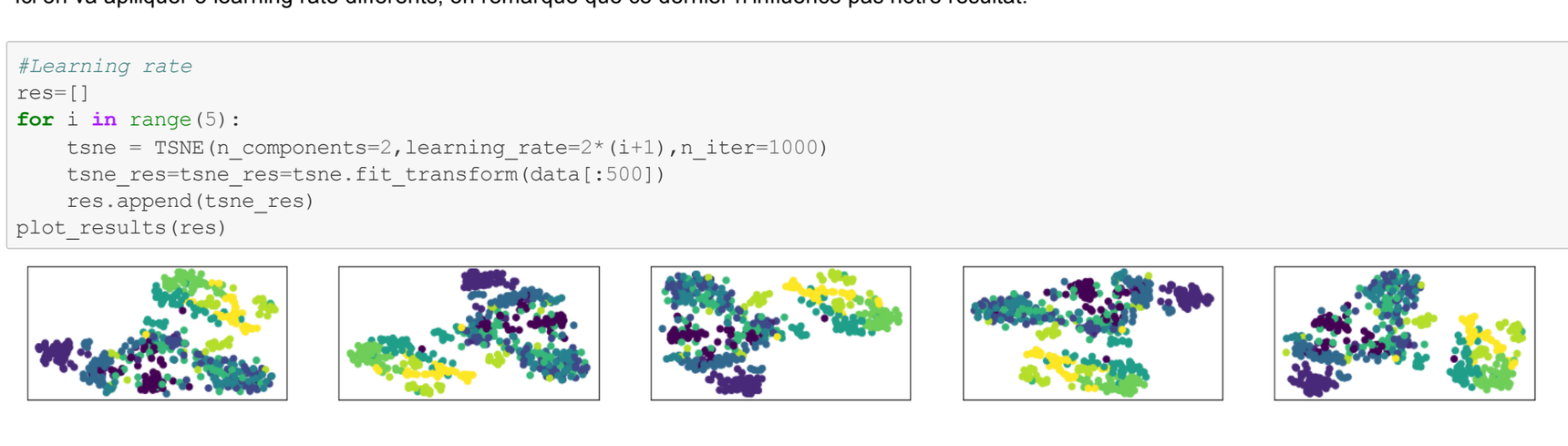
Reduction de dimension pour Fashion-MNIST avec t-SNE

T-SNE est très couteuse, pour cela on va appliquer notre étude sur une petite proportion de la data . La fonction t-SNE prends en compte un certain nombre de paramètres :Le taux d'apprentissage, la perplexité, l'exageration prématurée, le nombre d'itérations,le nombre de tentatives en cas de non-amélioration etc.

In [0]:
```python
def plot_results(res):
    n = 5 # how many digits we will display
    plt.figure(figsize=(25,5))
    for i in range(n):

        # display original
        ax = plt.subplot(2, n, i + 1)
        plt.scatter(res[i][:, 0], res[i][:, 1],c=target[:500])
        ax.get_xaxis().set_visible(False)
        ax.get_yaxis().set_visible(False)
    plt.show()
```

Ici on va appliquer 5 learning rate differents, on remarque que ce dernier n'influence pas notre résultat.

In [14]:
```python
#Learning rate
res=[]
for i in range(5):
    tsne = TSNE(n_components=2,learning_rate=2*(i+1),n_iter=1000)
    tsne_res=tsne_res=tsne.fit_transform(data[:500])
    res.append(tsne_res)
plot_results(res)
```

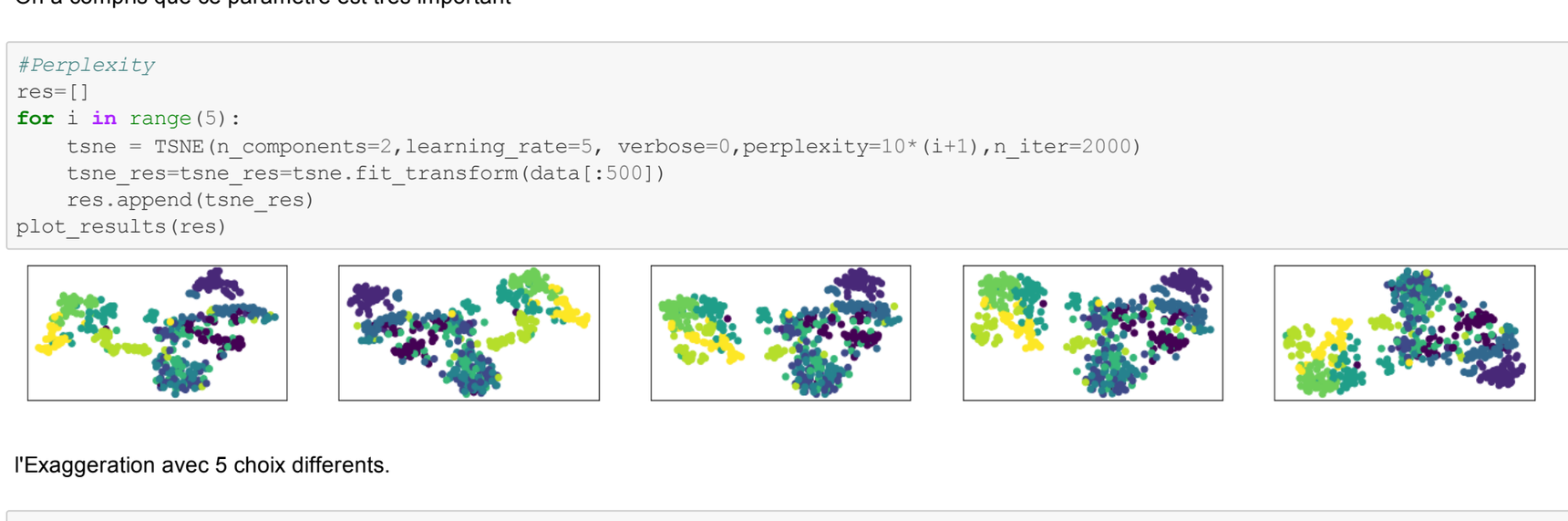


La même chose pour la perplexity, 5 choix differents. On remarque aussi une meilleur separation en changenant ce paramètre.

Après avoir consulté cet article : https://distill.pub/2016/misread-tsne/?fbclid=IwAR1KcfFl_xlGSyEaMcAJxfB-QoFO8dH1ojP5r4pJ-DDfbMTszL_1-IcKA74

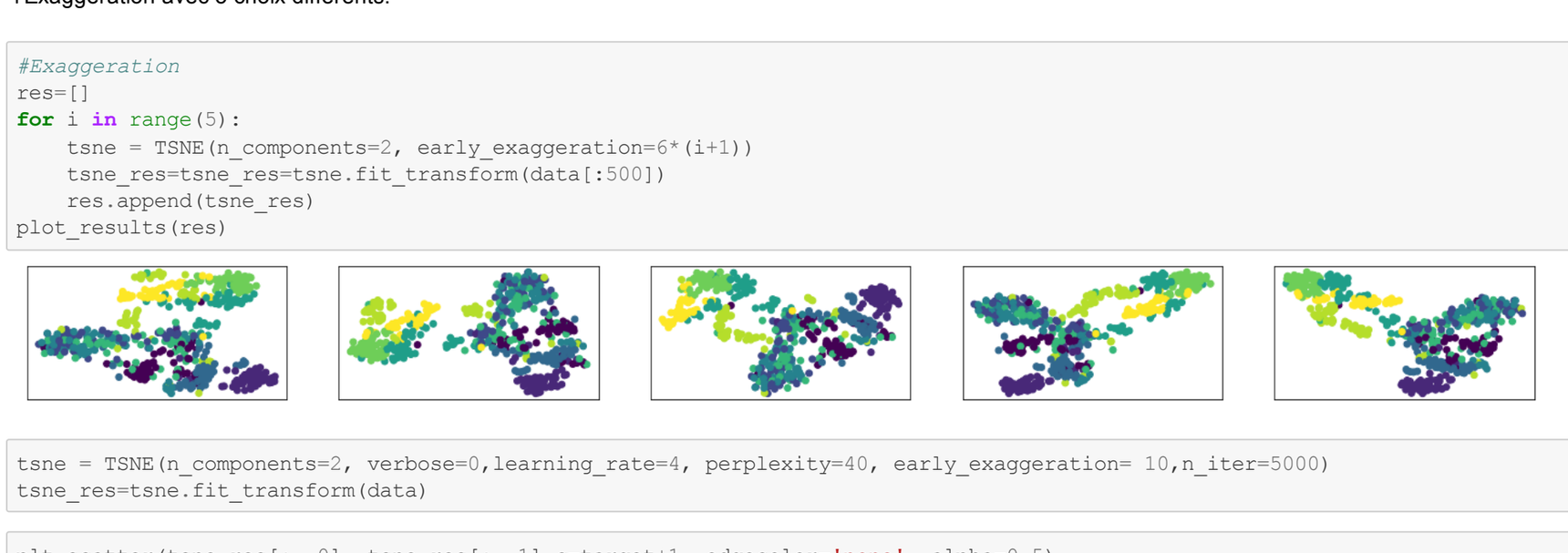
On a compris que ce paramètre est très important

In [15]:
```python
#Perplexity
res=[]
for i in range(5):
    tsne = TSNE(n_components=2,learning_rate=5, verbose=0,perplexity=10*(i+1),n_iter=2000)
    tsne_res=tsne_res=tsne.fit_transform(data[:500])
    res.append(tsne_res)
plot_results(res)
```



l'Exaggeration avec 5 choix differents.

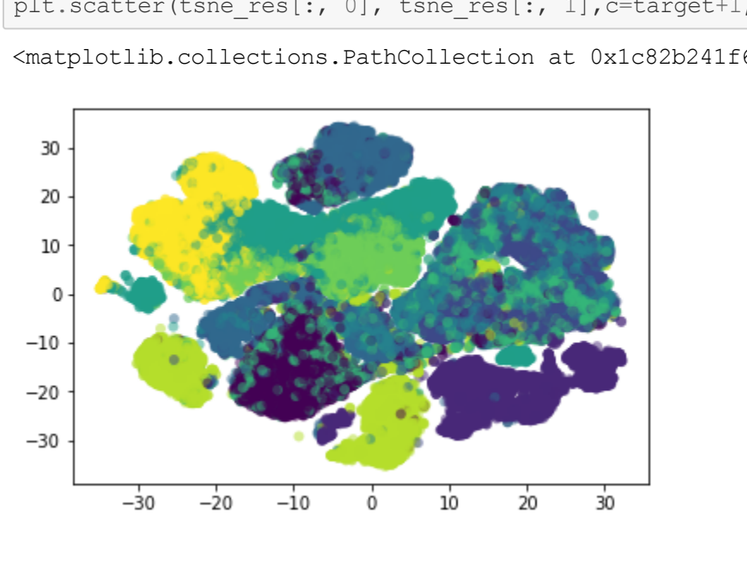
In [16]:
```python
#Exaggeration
res=[]
for i in range(5):
    tsne = TSNE(n_components=2, early_exaggeration=6*(i+1))
    tsne_res=tsne_res=tsne.fit_transform(data[:500])
    res.append(tsne_res)
plot_results(res)
```



In [0]:
```python
tsne = TSNE(n_components=2, verbose=0,learning_rate=4, perplexity=40, early_exaggeration= 10,n_iter=5000)
tsne_res=tsne.fit_transform(data)
```

In [0]:
```python
plt.scatter(tsne_res[:, 0], tsne_res[:, 1],c=target+1, edgecolor='none', alpha=0.5)
```

Out[0]:
```
<matplotlib.collections.PathCollection at 0x1c82b241f60>
```



AutoEncoder

In [0]:
```python
# Fonction pour evaluer les meilleurs paramètres à utiliser
def get_erreur(loss_func,optim,x_train,x_test):
        encoding_dim = 32
        input_img = tf.keras.layers.Input(shape=(784,))
        layer=tf.keras.layers.Dense(encoding_dim*16,activation="relu")(input_img)
        layer=tf.keras.layers.Dense(encoding_dim*8,activation="relu")(layer)
        layer=tf.keras.layers.Dense(encoding_dim*4,activation="relu")(layer)
        layer=tf.keras.layers.Dense(encoding_dim*2,activation="relu")(layer)
        layer=tf.keras.layers.Dense(encoding_dim,activation="relu")(layer)

        Encoded=tf.keras.layers.Dense(32,activation="relu")(layer)

        layer=tf.keras.layers.Dense(encoding_dim*2,activation="relu")(layer)
        layer=tf.keras.layers.Dense(encoding_dim*4,activation="relu")(layer)
        layer=tf.keras.layers.Dense(encoding_dim*8,activation="relu")(layer)
        layer=tf.keras.layers.Dense(encoding_dim*16,activation="relu")(layer)

        Decoded=tf.keras.layers.Dense(784,activation="sigmoid")(layer)
        AutoEncoder=tf.keras.models.Model(input_img,Decoded)
        Encoder=tf.keras.models.Model(input_img,Encoded)
        AutoEncoder.compile(optimizer =optim , loss = loss_func)
        history=AutoEncoder.fit(x_train,x_train,epochs=50,validation_data=(x_test,x_test),batch_size=256,shuffle=True)

        return min(history.history['loss']),history,Encoder,AutoEncoder

loss_function=['mean_squared_error','mean_absolute_error','mean_absolute_percentage_error','mean_squared_logarithmic_error']
Optimiser=['SGD','Adagrad','Adagrad','RMSprop','Adam']
```

In [0]:
```python
losses=[]
for loss in loss_function:
  e,_,_,_=get_erreur(loss,'Adam',x_train[:1000,:],x_test[:1000,:])
  losses.append(e)
print("La meilleure fonction pour le loss est : ",loss_function[losses.index(min(losses))])
```

In [0]:
```python
losses=[]
for opti in Optimiser:
  e,_,_,_=get_erreur('mean_squared_logarithmic_error',opti,x_train[:1000,:],x_test[:1000,:])
  losses.append(e)
print("Le meilleur optimizer est : ",Optimiser[losses.index(min(losses))])
```

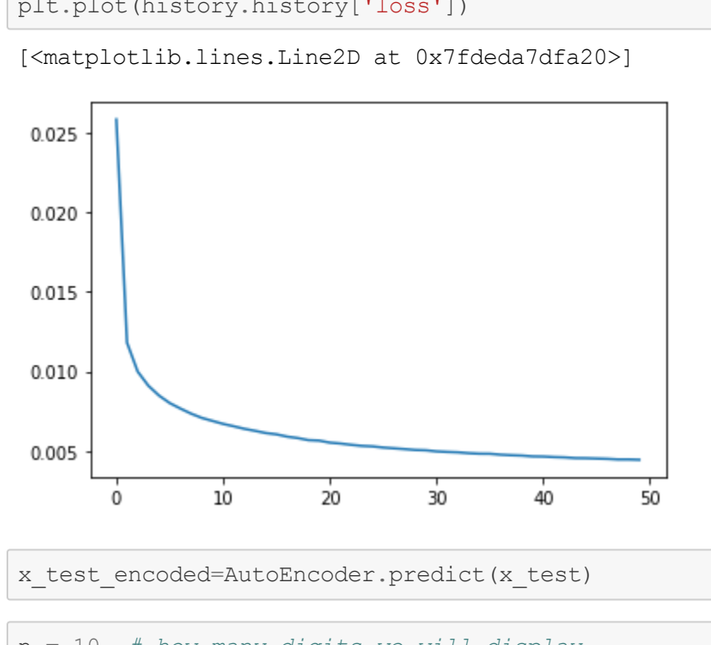
In [0]:
```python
encoding_dim = 32
```

In [19]:
```python
encoding_dim = 32
input_img = tf.keras.layers.Input(shape=(784,))
layer=tf.keras.layers.Dense(encoding_dim*16,activation="relu")(input_img)
layer=tf.keras.layers.Dense(encoding_dim*8,activation="relu")(layer)
layer=tf.keras.layers.Dense(encoding_dim*4,activation="relu")(layer)
layer=tf.keras.layers.Dense(encoding_dim*2,activation="relu")(layer)
layer=tf.keras.layers.Dense(encoding_dim,activation="relu")(layer)

Encoded=tf.keras.layers.Dense(32,activation="relu")(layer)

layer=tf.keras.layers.Dense(encoding_dim*2,activation="relu")(layer)
layer=tf.keras.layers.Dense(encoding_dim*4,activation="relu")(layer)
layer=tf.keras.layers.Dense(encoding_dim*8,activation="relu")(layer)
layer=tf.keras.layers.Dense(encoding_dim*16,activation="relu")(layer)
Decoded=tf.keras.layers.Dense(784,activation="sigmoid")(layer)

AutoEncoder=tf.keras.models.Model(input_img,Decoded)
Encoder=tf.keras.models.Model(input_img,Encoded)
AutoEncoder.compile(optimizer ='Adam' , loss = 'mean_squared_logarithmic_error')
history=AutoEncoder.fit(x_train,x_train,epochs=50,validation_data=(x_test,x_test),batch_size=256,shuffle=True)
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow_core/python/ops/resource_variable_ops.py:1630:
calling BaseResourceVariable.__init__ (from tensorflow.python.ops.resource_variable_ops) with constraint is deprecated an
d will be removed in a future version.
Instructions for updating:
If using Keras pass *_constraint arguments to layers.
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow_core/python/ops/math_grad.py:1424: where (from
tensorflow.python.ops.array_ops) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
Train on 60000 samples, validate on 10000 samples
Epoch 1/50
60000/60000 [==============================] - 2s 36us/sample - loss: 0.0259 - val_loss: 0.0138
Epoch 2/50
60000/60000 [==============================] - 1s 20us/sample - loss: 0.0118 - val_loss: 0.0105
Epoch 3/50
60000/60000 [==============================] - 1s 20us/sample - loss: 0.0100 - val_loss: 0.0096
Epoch 4/50
60000/60000 [==============================] - 1s 19us/sample - loss: 0.0091 - val_loss: 0.0087
Epoch 5/50
60000/60000 [==============================] - 1s 20us/sample - loss: 0.0085 - val_loss: 0.0083
Epoch 6/50
60000/60000 [==============================] - 1s 19us/sample - loss: 0.0080 - val_loss: 0.0079
Epoch 7/50
60000/60000 [==============================] - 1s 20us/sample - loss: 0.0077 - val_loss: 0.0076
Epoch 8/50
60000/60000 [==============================] - 1s 19us/sample - loss: 0.0073 - val_loss: 0.0076
Epoch 9/50
60000/60000 [==============================] - 1s 19us/sample - loss: 0.0071 - val_loss: 0.0072
Epoch 10/50
60000/60000 [==============================] - 1s 20us/sample - loss: 0.0069 - val_loss: 0.0068
Epoch 11/50
60000/60000 [==============================] - 1s 19us/sample - loss: 0.0067 - val_loss: 0.0068
Epoch 12/50
60000/60000 [==============================] - 1s 19us/sample - loss: 0.0065 - val_loss: 0.0068
Epoch 13/50
60000/60000 [==============================] - 1s 19us/sample - loss: 0.0064 - val_loss: 0.0064
Epoch 14/50
60000/60000 [==============================] - 1s 20us/sample - loss: 0.0063 - val_loss: 0.0063
Epoch 15/50
60000/60000 [==============================] - 1s 19us/sample - loss: 0.0061 - val_loss: 0.0062
Epoch 16/50
60000/60000 [==============================] - 1s 19us/sample - loss: 0.0060 - val_loss: 0.0061
Epoch 17/50
60000/60000 [==============================] - 1s 20us/sample - loss: 0.0059 - val_loss: 0.0060
Epoch 18/50
60000/60000 [==============================] - 1s 20us/sample - loss: 0.0058 - val_loss: 0.0059
Epoch 19/50
60000/60000 [==============================] - 1s 19us/sample - loss: 0.0057 - val_loss: 0.0058
Epoch 20/50
60000/60000 [==============================] - 1s 21us/sample - loss: 0.0056 - val_loss: 0.0058
Epoch 21/50
60000/60000 [==============================] - 1s 20us/sample - loss: 0.0055 - val_loss: 0.0057
Epoch 22/50
60000/60000 [==============================] - 1s 21us/sample - loss: 0.0055 - val_loss: 0.0056
Epoch 23/50
60000/60000 [==============================] - 1s 20us/sample - loss: 0.0054 - val_loss: 0.0055
Epoch 24/50
60000/60000 [==============================] - 1s 20us/sample - loss: 0.0053 - val_loss: 0.0055
Epoch 25/50
60000/60000 [==============================] - 1s 20us/sample - loss: 0.0053 - val_loss: 0.0055
Epoch 26/50
60000/60000 [==============================] - 1s 19us/sample - loss: 0.0052 - val_loss: 0.0055
Epoch 27/50
60000/60000 [==============================] - 1s 19us/sample - loss: 0.0052 - val_loss: 0.0053
Epoch 28/50
60000/60000 [==============================] - 1s 19us/sample - loss: 0.0051 - val_loss: 0.0053
Epoch 29/50
60000/60000 [==============================] - 1s 19us/sample - loss: 0.0051 - val_loss: 0.0052
Epoch 30/50
60000/60000 [==============================] - 1s 19us/sample - loss: 0.0050 - val_loss: 0.0053
Epoch 31/50
60000/60000 [==============================] - 1s 20us/sample - loss: 0.0050 - val_loss: 0.0052
Epoch 32/50
60000/60000 [==============================] - 1s 19us/sample - loss: 0.0049 - val_loss: 0.0051
Epoch 33/50
60000/60000 [==============================] - 1s 20us/sample - loss: 0.0049 - val_loss: 0.0052
Epoch 34/50
60000/60000 [==============================] - 1s 19us/sample - loss: 0.0049 - val_loss: 0.0051
Epoch 35/50
60000/60000 [==============================] - 1s 19us/sample - loss: 0.0048 - val_loss: 0.0052
Epoch 36/50
60000/60000 [==============================] - 1s 19us/sample - loss: 0.0048 - val_loss: 0.0051
Epoch 37/50
60000/60000 [==============================] - 1s 20us/sample - loss: 0.0048 - val_loss: 0.0050
Epoch 38/50
60000/60000 [==============================] - 1s 20us/sample - loss: 0.0047 - val_loss: 0.0050
Epoch 39/50
60000/60000 [==============================] - 1s 20us/sample - loss: 0.0047 - val_loss: 0.0050
Epoch 40/50
60000/60000 [==============================] - 1s 20us/sample - loss: 0.0047 - val_loss: 0.0049
Epoch 41/50
60000/60000 [==============================] - 1s 20us/sample - loss: 0.0046 - val_loss: 0.0049
Epoch 42/50
60000/60000 [==============================] - 1s 20us/sample - loss: 0.0046 - val_loss: 0.0049
Epoch 43/50
60000/60000 [==============================] - 1s 20us/sample - loss: 0.0046 - val_loss: 0.0049
Epoch 44/50
60000/60000 [==============================] - 1s 20us/sample - loss: 0.0045 - val_loss: 0.0049
Epoch 45/50
60000/60000 [==============================] - 1s 20us/sample - loss: 0.0045 - val_loss: 0.0048
Epoch 46/50
60000/60000 [==============================] - 1s 20us/sample - loss: 0.0045 - val_loss: 0.0048
Epoch 47/50
60000/60000 [==============================] - 1s 20us/sample - loss: 0.0045 - val_loss: 0.0049
Epoch 48/50
60000/60000 [==============================] - 1s 20us/sample - loss: 0.0045 - val_loss: 0.0047
Epoch 49/50
60000/60000 [==============================] - 1s 19us/sample - loss: 0.0045 - val_loss: 0.0048
Epoch 50/50
60000/60000 [==============================] - 1s 20us/sample - loss: 0.0044 - val_loss: 0.0048
```

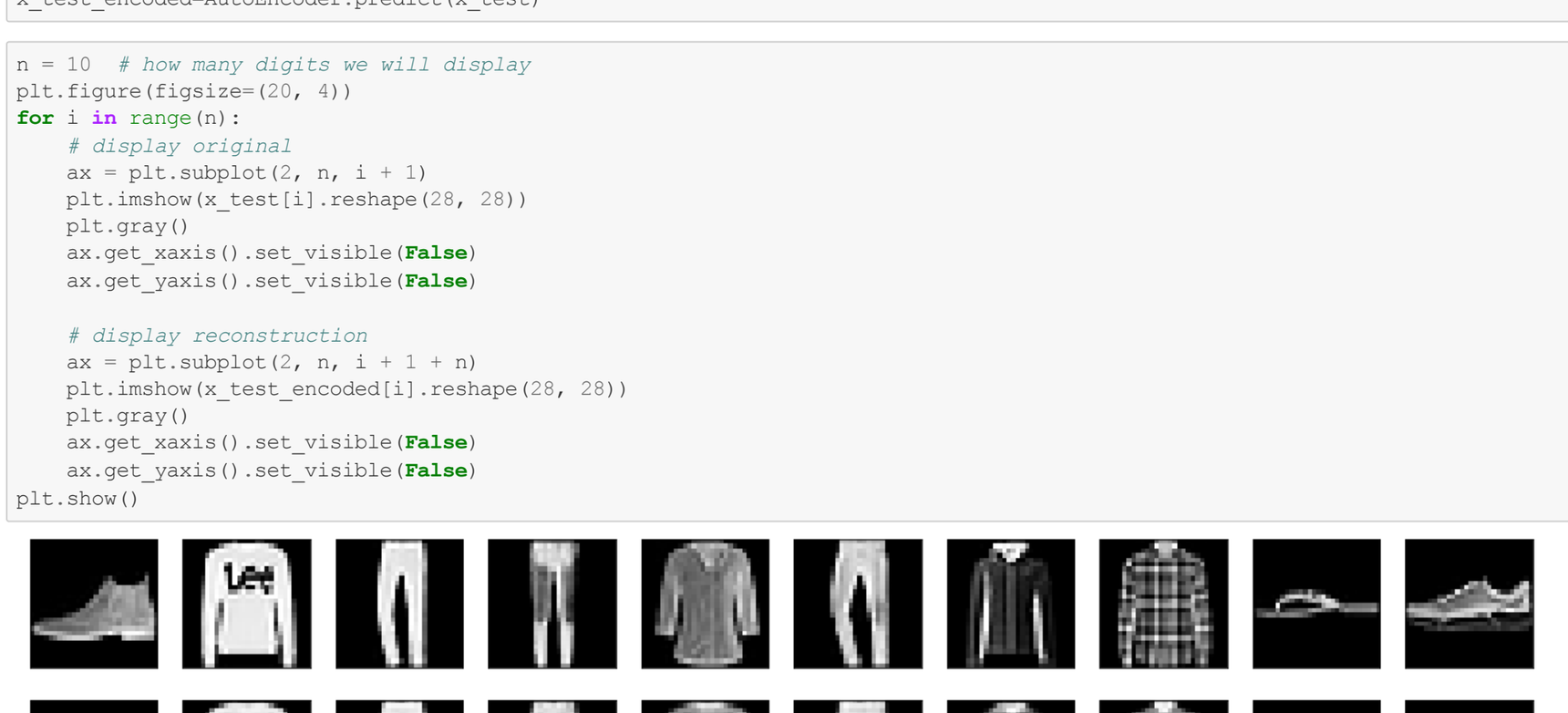
In [20]:
```python
plt.plot(history.history['loss'])
```

Out[20]:
```
[<matplotlib.lines.Line2D at 0x7fdeda7dfa20>]
```



In [0]:
```python
x_test_encoded=AutoEncoder.predict(x_test)
```

In [22]:
```python
n = 10  # how many digits we will display
plt.figure(figsize=(20, 4))
for i in range(n):
    # display original
    ax = plt.subplot(2, n, i + 1)
    plt.imshow(x_test[i].reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)

    # display reconstruction
    ax = plt.subplot(2, n, i + 1 + n)
    plt.imshow(x_test_encoded[i].reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)
plt.show()
```



In [0]:
```python
encoded_images = Encoder.predict(data)
```

Convontional Autoencoder (Bonus)

In [0]:
```python
from keras.layers import Input, Dense, Conv2D, MaxPooling2D, UpSampling2D
from keras.models import Model
from keras import backend as K

input_img = Input(shape=(28, 28, 1))

x = Conv2D(16, (3, 3), activation='relu', padding='same')(input_img)
x = MaxPooling2D((2, 2), padding='same')(x)
x = Conv2D(8, (3, 3), activation='relu', padding='same')(x)
x = MaxPooling2D((2, 2), padding='same')(x)
x = Conv2D(8, (3, 3), activation='relu', padding='same')(x)

encoded = MaxPooling2D((2, 2), padding='same')(x)

x = Conv2D(8, (3, 3), activation='relu', padding='same')(encoded)
x = UpSampling2D((2, 2))(x)
x = Conv2D(8, (3, 3), activation='relu', padding='same')(x)
x = UpSampling2D((2, 2))(x)
x = Conv2D(16, (3, 3), activation='relu')(x)
x = UpSampling2D((2, 2))(x)
decoded = Conv2D(1, (3, 3), activation='sigmoid', padding='same')(x)

Encoder2=Model(input_img, encoded)
autoencoder2 = Model(input_img, decoded)
autoencoder2.compile(optimizer='adam', loss='mean_squared_logarithmic_error')
```

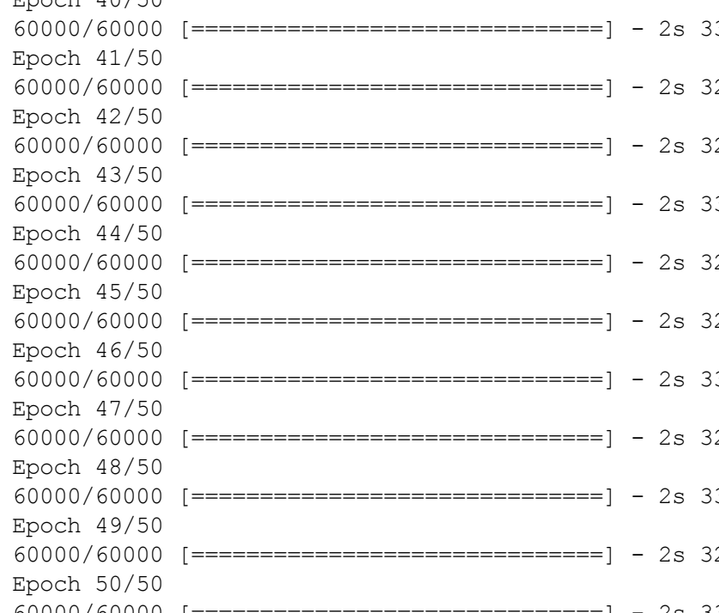
In [0]:
```python
x_train1 = np.reshape(x_train, (len(x_train), 28, 28, 1))
x_test1 = np.reshape(x_test, (len(x_test), 28, 28, 1))
```

In [53]:
```python
history2=autoencoder2.fit(x_train1,x_train1,epochs=50,
                validation_data=(x_test1,x_test1),
                batch_size=256,shuffle=True)
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.py:1033: The name tf.assi
gn_add is deprecated. Please use tf.compat.v1.assign_add instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.py:1020: The name tf.assi
gn is deprecated. Please use tf.compat.v1.assign instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.py:3005: The name tf.Sess
ion is deprecated. Please use tf.compat.v1.Session instead.

Train on 60000 samples, validate on 10000 samples
Epoch 1/50
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.py:190: The name tf.get_d
efault_session is deprecated. Please use tf.compat.v1.get_default_session instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.py:197: The name tf.Confi
gProto is deprecated. Please use tf.compat.v1.ConfigProto instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.py:207: The name tf.globa
l_variables is deprecated. Please use tf.compat.v1.global_variables instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.py:216: The name tf.is_va
riable_initialized is deprecated. Please use tf.compat.v1.is_variable_initialized instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.py:223: The name tf.varia
bles_initializer is deprecated. Please use tf.compat.v1.variables_initializer instead.

60000/60000 [==============================] - 7s 116us/step - loss: 0.0322 - val_loss: 0.0145
Epoch 2/50
60000/60000 [==============================] - 2s 32us/step - loss: 0.0125 - val_loss: 0.0114
Epoch 3/50
60000/60000 [==============================] - 2s 32us/step - loss: 0.0109 - val_loss: 0.0105
Epoch 4/50
60000/60000 [==============================] - 2s 32us/step - loss: 0.0103 - val_loss: 0.0101
Epoch 5/50
60000/60000 [==============================] - 2s 33us/step - loss: 0.0098 - val_loss: 0.0097
Epoch 6/50
60000/60000 [==============================] - 2s 33us/step - loss: 0.0095 - val_loss: 0.0093
Epoch 7/50
60000/60000 [==============================] - 2s 32us/step - loss: 0.0092 - val_loss: 0.0091
Epoch 8/50
60000/60000 [==============================] - 2s 32us/step - loss: 0.0090 - val_loss: 0.0090
Epoch 9/50
60000/60000 [==============================] - 2s 33us/step - loss: 0.0088 - val_loss: 0.0088
Epoch 10/50
60000/60000 [==============================] - 2s 32us/step - loss: 0.0087 - val_loss: 0.0086
Epoch 11/50
60000/60000 [==============================] - 2s 32us/step - loss: 0.0086 - val_loss: 0.0085
Epoch 12/50
60000/60000 [==============================] - 2s 32us/step - loss: 0.0084 - val_loss: 0.0084
Epoch 13/50
60000/60000 [==============================] - 2s 32us/step - loss: 0.0083 - val_loss: 0.0083
Epoch 14/50
60000/60000 [==============================] - 2s 32us/step - loss: 0.0082 - val_loss: 0.0082
Epoch 15/50
60000/60000 [==============================] - 2s 32us/step - loss: 0.0082 - val_loss: 0.0082
Epoch 16/50
60000/60000 [==============================] - 2s 32us/step - loss: 0.0081 - val_loss: 0.0081
Epoch 17/50
60000/60000 [==============================] - 2s 33us/step - loss: 0.0080 - val_loss: 0.0080
Epoch 18/50
60000/60000 [==============================] - 2s 32us/step - loss: 0.0080 - val_loss: 0.0080
Epoch 19/50
60000/60000 [==============================] - 2s 32us/step - loss: 0.0079 - val_loss: 0.0079
Epoch 20/50
60000/60000 [==============================] - 2s 32us/step - loss: 0.0078 - val_loss: 0.0078
Epoch 21/50
60000/60000 [==============================] - 2s 32us/step - loss: 0.0078 - val_loss: 0.0078
Epoch 22/50
60000/60000 [==============================] - 2s 32us/step - loss: 0.0077 - val_loss: 0.0077
Epoch 23/50
60000/60000 [==============================] - 2s 32us/step - loss: 0.0077 - val_loss: 0.0077
Epoch 24/50
60000/60000 [==============================] - 2s 32us/step - loss: 0.0076 - val_loss: 0.0076
Epoch 25/50
60000/60000 [==============================] - 2s 32us/step - loss: 0.0076 - val_loss: 0.0076
Epoch 26/50
60000/60000 [==============================] - 2s 32us/step - loss: 0.0076 - val_loss: 0.0076
Epoch 27/50
60000/60000 [==============================] - 2s 33us/step - loss: 0.0075 - val_loss: 0.0076
Epoch 28/50
60000/60000 [==============================] - 2s 32us/step - loss: 0.0075 - val_loss: 0.0075
Epoch 29/50
60000/60000 [==============================] - 2s 32us/step - loss: 0.0075 - val_loss: 0.0075
Epoch 30/50
60000/60000 [==============================] - 2s 33us/step - loss: 0.0074 - val_loss: 0.0074
Epoch 31/50
60000/60000 [==============================] - 2s 33us/step - loss: 0.0074 - val_loss: 0.0075
Epoch 32/50
60000/60000 [==============================] - 2s 32us/step - loss: 0.0074 - val_loss: 0.0074
Epoch 33/50
60000/60000 [==============================] - 2s 33us/step - loss: 0.0074 - val_loss: 0.0075
Epoch 34/50
60000/60000 [==============================] - 2s 32us/step - loss: 0.0074 - val_loss: 0.0074
Epoch 35/50
60000/60000 [==============================] - 2s 32us/step - loss: 0.0073 - val_loss: 0.0073
Epoch 36/50
60000/60000 [==============================] - 2s 32us/step - loss: 0.0073 - val_loss: 0.0073
Epoch 37/50
60000/60000 [==============================] - 2s 32us/step - loss: 0.0073 - val_loss: 0.0073
Epoch 38/50
60000/60000 [==============================] - 2s 32us/step - loss: 0.0072 - val_loss: 0.0072
Epoch 39/50
60000/60000 [==============================] - 2s 32us/step - loss: 0.0072 - val_loss: 0.0073
Epoch 40/50
60000/60000 [==============================] - 2s 33us/step - loss: 0.0072 - val_loss: 0.0072
Epoch 41/50
60000/60000 [==============================] - 2s 32us/step - loss: 0.0072 - val_loss: 0.0072
Epoch 42/50
60000/60000 [==============================] - 2s 32us/step - loss: 0.0072 - val_loss: 0.0072
Epoch 43/50
60000/60000 [==============================] - 2s 33us/step - loss: 0.0071 - val_loss: 0.0072
Epoch 44/50
60000/60000 [==============================] - 2s 32us/step - loss: 0.0071 - val_loss: 0.0071
Epoch 45/50
60000/60000 [==============================] - 2s 32us/step - loss: 0.0071 - val_loss: 0.0071
Epoch 46/50
60000/60000 [==============================] - 2s 33us/step - loss: 0.0071 - val_loss: 0.0071
Epoch 47/50
60000/60000 [==============================] - 2s 32us/step - loss: 0.0071 - val_loss: 0.0071
Epoch 48/50
60000/60000 [==============================] - 2s 33us/step - loss: 0.0070 - val_loss: 0.0070
Epoch 49/50
60000/60000 [==============================] - 2s 32us/step - loss: 0.0070 - val_loss: 0.0071
Epoch 50/50
60000/60000 [==============================] - 2s 33us/step - loss: 0.0070 - val_loss: 0.0070
```

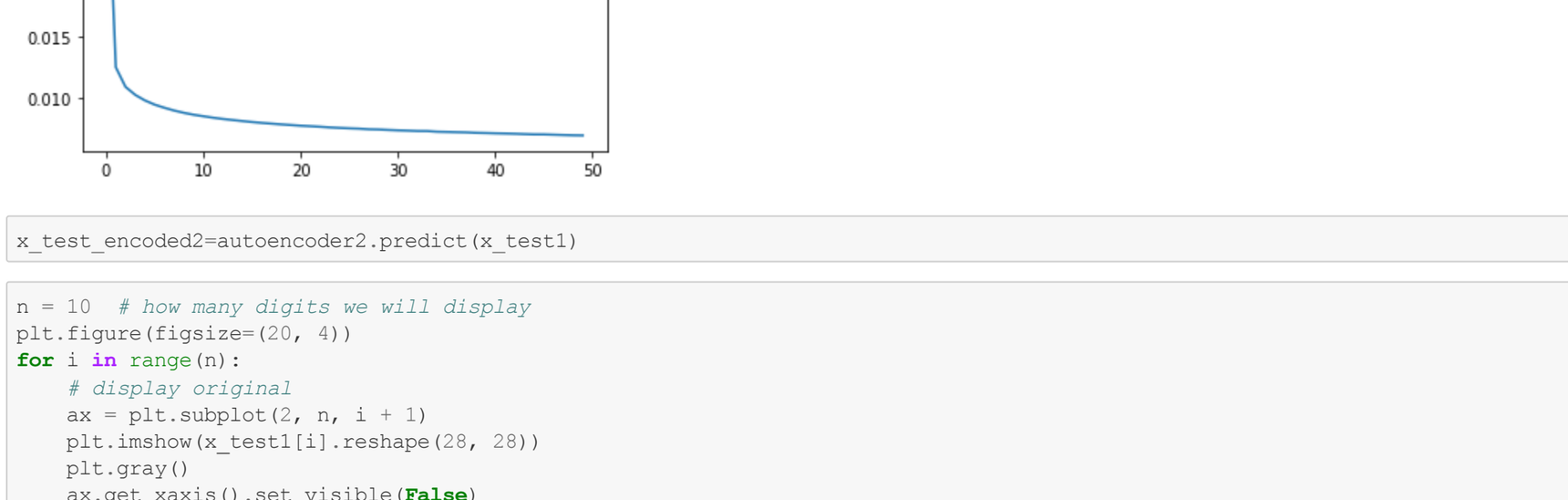
In [54]:
```python
plt.plot(history2.history['loss'])
```

Out[54]:
```
[<matplotlib.lines.Line2D at 0x7fcde9055470>]
```



In [0]:
```python
x_test_encoded2=autoencoder2.predict(x_test1)
```

In [56]:
```python
n = 10  # how many digits we will display
plt.figure(figsize=(20, 4))
for i in range(n):
    # display original
    ax = plt.subplot(2, n, i + 1)
    plt.imshow(x_test1[i].reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)

    # display reconstruction
    ax = plt.subplot(2, n, i + 1 + n)
    plt.imshow(x_test_encoded2[i].reshape(28, 28)*255)
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)
plt.show()
```



Kmeans et Evaluation NMI/ARI

In [0]:
```python
kmeans=KMeans(n_clusters=10,random_state=0).fit(data)    # DATA
kmeans1=KMeans(n_clusters=10,random_state=0).fit(encoded_images) ## AE
kmeans2=KMeans(n_clusters=10,random_state=0).fit(pca25_res)   # PCA 25 Composantes
kmeans3=KMeans(n_clusters=10,random_state=0).fit(pca100_res)  # PCA 100 Composantes
kmeans4=KMeans(n_clusters=10,random_state=0).fit(tsne_res)  # TSNE 2 composantes
```

On remarque selon les deux metrics NMI/ARI, TSNE est le meilleur pour l'instant, voyons voir le résultat de T-SNE sur les images encodés

In [69]:
```python
print("Evaluation pour la data complète : NMI ==> {0:.2f}, ARI ==> {1:.2f} ".format(nmi(target,kmeans.labels_),ari(target,kme
ans.labels_)))
print("Evaluation du résultat de l'AE   : NMI ==> {0:.2f}, ARI ==> {1:.2f} ".format(nmi(target,kmeans1.labels_),ari(target,km
eans1.labels_)))
print("Evaluation du résultat de PCA25  : NMI ==> {0:.2f}, ARI ==> {1:.2f} ".format(nmi(target,kmeans2.labels_),ari(target,km
eans2.labels_)))
print("Evaluation du résultat de PCA100 : NMI ==> {0:.2f}, ARI ==> {1:.2f} ".format(nmi(target,kmeans3.labels_),ari(target,km
eans3.labels_)))
print("Evaluation du résultat de TSNE   : NMI ==> {0:.2f},ARI ==> {1:.2f} ".format(nmi(target,kmeans4.labels_),ari(target,kme
ans4.labels_)))
```

```
Evaluation pour la data complète : NMI ==> 0.51, ARI ==> 0.35
Evaluation du résultat de l'AE   : NMI ==> 0.35, ARI ==> 0.16
Evaluation du résultat de PCA25  : NMI ==> 0.53, ARI ==> 0.38
Evaluation du résultat de PCA100 : NMI ==> 0.51, ARI ==> 0.35
Evaluation du résultat de TSNE :   NMI ==> 0.55, ARI ==> 0.45
```

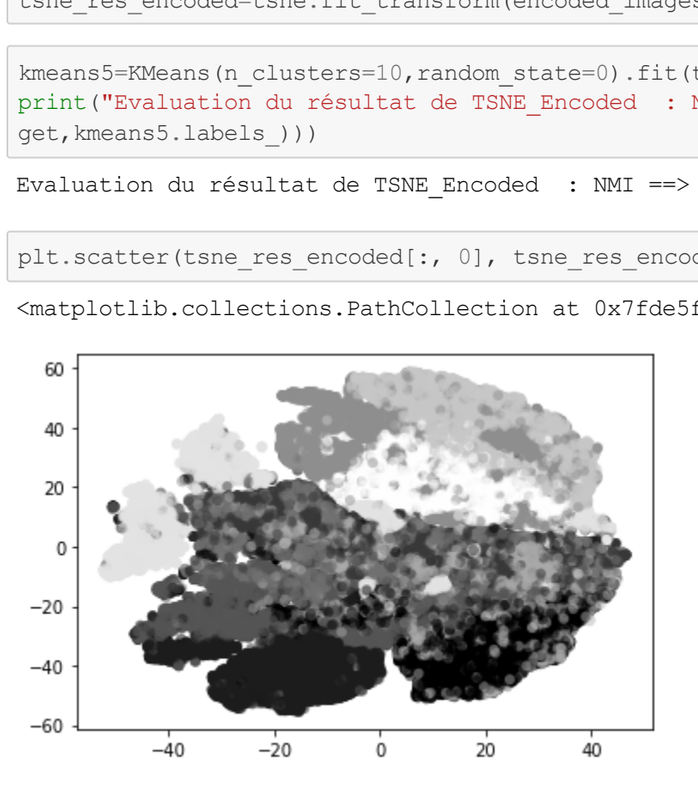
In [0]:
```python
tsne_res_encoded=tsne.fit_transform(encoded_images)
```

In [25]:
```python
kmeans5=KMeans(n_clusters=10,random_state=0).fit(tsne_res_encoded)  # TSNE sur encoded
print("Evaluation du résultat de TSNE_Encoded  : NMI ==> {0:.2f},ARI ==> {1:.2f} ".format(nmi(target,kmeans5.labels_),ari(tar
get,kmeans5.labels_)))
```

```
Evaluation du résultat de TSNE_Encoded  : NMI ==> 0.45,ARI ==> 0.30
```

In [26]:
```python
plt.scatter(tsne_res_encoded[:, 0], tsne_res_encoded[:, 1],c=target+1, edgecolor='none', alpha=0.5)
```

Out[26]:
```
<matplotlib.collections.PathCollection at 0x7fde5f79da58>
```



Conclusion :

Le meilleur résultat selon notre étude, est TSNE, mais sachant que cette derniere prends des heures pour donner résultat, on va préferer les résultats de PCA25 vu que son exécution est très rapide, en plus il garde autant d'informations que TSNE ou presque. Cependant pour la visualisation, TSNE remporte le challenge.