```
 # Script:
#    L'apprentissage profond sous Keras

# Description:
# Ce script est dédié au développement d'un réseau de neurons DL sous Keras
# avec Tensorflow comme backend,
# le développement de cet algorithme suit étapes suivant:
# 1- Importation du bibliothèques et les données;
# 2- Spécification des variables indépendants et dépendant;
# 3- Normalisation des données;
# 4- Création du modèle séquentiel;
# 5- Calcul des indice de pérformances.

# Version:
#     Mohammed AMEKSA:       Juin 2019       Script Original
```

3.1 Importer les bibliothèques et les données

```
In [21]: import numpy as np
 import pandas as pd
 # pour l'évaluation
 from sklearn import metrics
 # importer le package de standardisation
 from sklearn.preprocessing import StandardScaler
 # Importer les packages et les bibliothèques de Keras
 import keras
 from keras.models import Sequential
 from keras.layers import Dense
```

3.1.1 Importation des données

```
In [4]: # Importer les deux fichiers de données
dataset_train = pd.read_csv('Train-Equil-Lon-Lat-Hour-Month-RedVisi.csv')
dataset_test = pd.read_csv('Test-Equil-Lon-Lat-Hour-Month-RedVisi.csv')
#Détérminer pour chaque fichie les variables independants et le target
#fichier d'entrainement
X_train = dataset_train.iloc[:, 0:-1].values
y_train = dataset_train.iloc[:, -1].values
#fichier de test
X_test = dataset_test.iloc[:, 0:-1].values
y_test = dataset_test.iloc[:, -1].values
```

3.1.2 Normalisation des données

```
In [5]: # standardisé les données en faisant appel à un instance de StandardScaler
# cette instance suit la régle "x-men/std"
scaler = StandardScaler()
scaler.fit(X_train)
```

```
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

3.1.3 Création du modèle

```
 # Initialiser le modèle séquentiel
keras_model = Sequential()
# Ajout du couche d'entrée et du premier couche caché
keras_model.add(Dense(output_dim = 68 , init = 'uniform',
activation = 'relu', input_dim = 34))
# Ajout du deuxième couche cachée
keras_model.add(Dense(output_dim = 68, init = 'uniform',
activation = 'relu'))
# ajouter la couche de sortie (output layer)
keras_model.add(Dense(output_dim = 1, init = 'uniform',
activation = 'relu'))
# Compilation de l'ANN
keras.optimizers.Adam(lr=0.001, beta_1=0.9, beta_2=0.999,
      epsilon=1e-08, decay=0.0, amsgrad=False)
keras_model.compile(optimizer = 'adam', loss = 'mean_squared_error',
    metrics = ['mse'])
# Ajuster l'ANN sur les données d'entrainement
keras_model.fit(X_train, y_train, batch_size = 10, nb_epoch =100)

# on fait des prédiction et on évalue le modèle
y_pred = keras_model.predict(X_test)
```

3.1.4 Calculer les indice de performance

```
 biais_ann=np.mean(y_pred)-np.mean(y_test)
var_ann = metrics.explained_variance_score(y_test,y_pred)
mae_ann = metrics.mean_absolute_error(y_test,y_pred)
mse_ann =metrics.mean_squared_error(y_test,y_pred)
rmse_ann = np.sqrt(mse_ann)

## ce morceau de code pour calculer le coefficient de correlation
df=pd.DataFrame(columns=['y_pred','y_test'])
yp_=pd.DataFrame(y_pred,columns=['y_p'])
df['y_pred'] = yp_['y_p']
df['y_test'] = pd.Series(y_test)

cc1=np.sum((df['y_pred']-np.mean(df['y_pred']))*(df['y_test']-np.mean(df['y_test'])))
cc2= (np.sqrt(np.sum((df['y_pred']-np.mean(df['y_pred']))**2)))
*(np.sqrt(np.sum((df['y_test']-np.mean(df['y_test']))**2)))
cc=cc1/cc2

# affichage des résultats
```

```
print("CC_ANN: %.5f" %cc)
print("BIAIS_ANN: %.5f" % biais_ann)
print("MSE_ANN: %.5f" % mse_ann)
print("MAE_ANN: %.5f" %mae_ann)
print("RMSE_ANN: %.5f" %rmse_ann)
```