```
# coding: utf-8

# # Analyse de données financières : Détection d'anomalies dans un porteuille
d'actifs

# In[2]:

import quandl
import pandas as pd
import numpy as np
import datetime
import matplotlib.pyplot as plt
import seaborn
from sklearn.ensemble import IsolationForest
from sklearn.neighbors.kde import KernelDensity

# instruction pour plotter dans notebook
get_ipython().magic('matplotlib inline')

# connection à quandl via l'api key
# quandl.ApiConfig.api_key = " 3CipYF1Y3fzjvDgg7E2n"


# In[3]:

# Lecture des données
# df = pd.read_csv("all_stocks_5yr.csv", index_col= [0,2],usecols=
[0,4,6],parse_dates=[0])
df = pd.read_csv("all_stocks_5yr.csv",usecols=[0,4,6], parse_dates=[0])
df.head()
df.Date = pd.to_datetime(df.Date)
df = df.loc[(df.Date<datetime.datetime(2017, 8, 11,0,0,0)),:]
df = df.loc[df.Date>datetime.datetime(2015, 8, 11,0,0,0),:]

# Rendements journaliers obtenue à partit d'un tableau dynamique
df_rendements = df.pivot('Date', 'Name', 'Close').pct_change()
# fontion de calcul de moyenne mobile sur le rendement
def rends_moov(dff):
    return dff.rolling(window =40).mean()



# Calculate the moving average sur les rendements
df_moving_rendements = df_rendements.apply(rends_moov)
df_moving_rendements.plot(figsize=(15,9), legend=False, title="Moyenne mobile des
rendements (40)")
plt.ylabel("Rendements en %", size=20)
plt.title('Moyenne mobile des rendements (40)', size=25)
plt.xlabel("Dates",size=20)


# ## Volatilités



#fonction de calcul des volatilités sur une période
def volts(dff):
    return dff.rolling(40).std() * np.sqrt(40)




# Define the minumum of periods to consider
# min_periods = 75
df_moving_volts = df_rendements.apply(volts)
df_moving_volts.plot(figsize=(15,9), legend=False, title="Volatilités mobiles
(40)")
plt.ylabel("Volatilités en %", size=20)
```

```
plt.title('Volatilités mobiles (40)', size=25)
plt.xlabel("Dates",size=20)


# # Detection d'anomalies

# # Détection sur les rendements

# df_2 est de dataframe qui contient les données dont on doit détecter les
anomalies
# Applications de l'isolation forest sur les données de rendements ou de
volatilités
clf = IsolationForest(n_estimators=100, max_samples='auto')

# fit de l'estimateur
clf.fit(df_moving_rendements.transpose().dropna(axis=0, how='all').dropna(axis=1,
how='any'))

# the anomaly score of the input samples. the lower the more abnormal.
scores_pred = pd.DataFrame(clf.decision_function(df_moving_rendements.transpose
().dropna(axis=0, how='all').dropna(axis=1,
how='any')),index=df_moving_rendements.transpose().dropna(axis=0, how='all').dropna
(axis=1, how='any').index.values)

predictions = pd.DataFrame(clf.predict(df_moving_rendements.transpose().dropna
(axis=0, how='all').dropna(axis=1, how='any')),
index=df_moving_rendements.transpose().dropna(axis=0, how='all').dropna(axis=1,
how='any').index.values)

# Les scores de tous les individus
scores_pred.plot(figsize=(24,8))

# Les anomalies prédits ou détectées par le modèle
predictions[predictions==-1].dropna().transpose().columns

# le score des anomalies
scores_pred.loc[predictions[predictions==-1].dropna().transpose
().columns,:].transpose()

# le score des anomalies
scores_pred.loc[predictions[predictions==-1].dropna().transpose().columns,:].plot
(figsize=(24,8), title='Scores des anomalies de rendements')


# # Détection sur les volatilités

# df_2 est de dataframe qui contient les données dont on doit détecter les
anomalies
# Applications de l'isolation forest sur les données de rendements ou de
volatilités
clf = IsolationForest(n_estimators=100, max_samples='auto')

# fit de l'estimateur
clf.fit(df_moving_volts.transpose().dropna(axis=0, how='all').dropna(axis=1,
how='any'))

# the anomaly score of the input samples. the lower the more abnormal.
scores_pred2 = pd.DataFrame(clf.decision_function(df_moving_volts.transpose
().dropna(axis=0, how='all').dropna(axis=1,
how='any')),index=df_moving_volts.transpose().dropna(axis=0, how='all').dropna
(axis=1, how='any').index.values)

#
predictions2 = pd.DataFrame(clf.predict(df_moving_volts.transpose().dropna(axis=0,
how='all').dropna(axis=1, how='any')), index=df_moving_volts.transpose().dropna
(axis=0, how='all').dropna(axis=1, how='any').index.values)

# Les scores de tous les individus
scores_pred2.plot(figsize=(24,8))
```

```
# Les anomalies prédits ou détectées par le modèle
predictions2[predictions2==-1].dropna().transpose().columns

# le score des anomalies
scores_pred2.loc[predictions2[predictions2==-1].dropna().transpose
().columns,:].transpose()


# Graphique des scores des anomalies
scores_pred2.loc[predictions2[predictions2==-1].dropna().transpose
().columns,:].plot(figsize=(24,8))

# comparaison anomalies obetues selon le rendements vs selon la volatilité afin de
comparer quelle sont détectées dans les deux cas
predictions2[predictions2==-1].dropna().transpose().columns.sort_values
()==predictions[predictions==-1].dropna().transpose().columns.sort_values()


# # Détection sur la fonction de densité

# fonction d'estimation des fonction de densités des rendements des actifs
def fdensite(X):
    X_plot = np.linspace(-5, 10, 504)[:, np.newaxis]
    return np.exp(KernelDensity(kernel='gaussian', bandwidth=0.75).fit(X.dropna
().values.reshape(-1, 1)).score_samples(X_plot))


df_densites = df_rendements.apply(fdensite)
df_densites.to_csv('df_densite.csv')
df_densites.A.plot()


import seaborn as sns

# Set up the matplotlib figure
f, axes = plt.subplots(2, 2, figsize=(12, 12), sharex=True)
sns.despine(left=True)

sns.distplot(df_rendements.A, hist=False, rug=True,kde=True, color="b", kde_kws=
{"shade": True}, ax=axes[0, 0])

sns.distplot(df_rendements.AAP, hist=False, rug=True,kde=True,color="r", kde_kws=
{"shade": True}, ax=axes[0, 1])

sns.distplot(df_rendements.ADSK, hist=False, rug=True,kde=True, color="y", kde_kws=
{"shade": True},  ax=axes[1, 0])

sns.distplot(df_rendements.AKAM, hist=False, rug=True,kde=True,color="g", kde_kws=
{"shade": True},  ax=axes[1, 1])


# df_2 est de dataframe qui contient les données dont on doit détecter les
anomalies
# Applications de l'isolation forest sur les données de rendements ou de
volatilités
clf = IsolationForest(n_estimators=100, max_samples='auto')

# fit de l'estimateur
clf.fit(df_densites.transpose())

# the anomaly score of the input samples. the lower the more abnormal.
scores_pred3 = pd.DataFrame(clf.decision_function(df_densites.transpose
()),index=df_densites.transpose().index.values)

#
predictions3 = pd.DataFrame(clf.predict(df_densites.transpose()),
index=df_densites.transpose().index.values)
```

```
# Les scores de tous les individus
scores_pred3.plot(figsize=(24,8))
plt.ylabel("Scores", size=20)
plt.title('Scores des densités des actifs', size=25)
plt.xlabel("Actifs",size=20)

# Les anomalies prédites ou détectées par le modèle
predictions3[predictions3==-1].dropna().transpose().columns

# le score des anomalies
scores_pred3.loc[predictions3[predictions3==-1].dropna().transpose
().columns,:].transpose()

# le score des anomalies
scores_pred3.loc[predictions3[predictions3==-1].dropna().transpose
().columns,:].plot(figsize=(24,8))
plt.ylabel("Scores", size=20)
plt.title('Scores des anomalies de rendements', size=25)
plt.xlabel("Actifs",size=20)


# # Nielson siegel svensson détection sur densité

# In[42]:
#a = df_rendements.A.reset_index().A.sort_values().reset_index().A.to_csv
('nss_data.csv')
a= pd.read_csv("sols.csv", index_col=0)

# df_2 est de dataframe qui contient les données dont on doit détecter les
anomalies
# Applications de l'isolation forest sur les données de rendements ou de
volatilités
clf = IsolationForest(n_estimators=100, max_samples='auto')

# fit de l'estimateur
clf.fit(a)

# the anomaly score of the input samples. the lower the more abnormal.
scores_pred4 = pd.DataFrame(clf.decision_function(a),index=a.index.values)

#
predictions4 = pd.DataFrame(clf.predict(a), index=a.index.values)

# Les scores de tous les individus
scores_pred3.plot(figsize=(24,10), title='Scores des courbes de densités par NSS
des actifs ', fontsize=15)
plt.ylabel("Scores des anomalies",size=20)
plt.title('Scores des courbes de densités par NSS des actifs ', size=25)
plt.xlabel("actifs",size=20)

# le score des anomalies
scores_pred4.loc[predictions4[predictions4==-1].dropna().transpose
().columns,:].plot(figsize=(24,8))
plt.ylabel("Scores", size=20)
plt.title('Scores des anomalies de rendements (NSS)', size=25)
plt.xlabel("Actifs",size=20)

# Les anomalies prédites ou détectées par le modèle
predictions4[predictions4==-1].dropna().transpose().columns

predictions3[predictions3==-1].dropna().transpose().columns

# Exemple de graphique des anomalies densite
# Set up the matplotlib figure
f, axes = plt.subplots(2, 1, figsize=(12, 12), sharex=True)
sns.despine(left=True)

sns.distplot(df_rendements.A, hist=False, rug=True,kde=True, color="g", kde_kws=
```

```
{"shade": True}, ax=axes[0])

sns.distplot(df_rendements.ADM, hist=False, rug=True,kde=True,color="r", kde_kws=
{"shade": True}, ax=axes[0])

sns.distplot(df_rendements.ABBV, hist=False, rug=True,kde=True, color="g", kde_kws=
{"shade": True},  ax=axes[1])

sns.distplot(df_rendements.ZION, hist=False, rug=True,kde=True,color="r", kde_kws=
{"shade": True},  ax=axes[1])
```