Pandas_analyse_bitcoin

August 10, 2020

Document rédigé par BOUNGOTO BIBAYI Yoanne

1 Pandas pour les Timeseries (Analyse du Bitcoin)

Idéale pour des phénomèmes qui evoluent au court du temps comme l'analyse du climat ou l'études de la bourse, pour la Maintenance prédictive avec des capteurs IoT etc.

2 Petite parenthèse sur la Maintenance Prédictive

La maintenance prédictive commence par la collecte de données des actifs. Il existe un moyen simple de faciliter cela et de rendre les actifs intelligents: en y ajoutant des capteurs connectés à l'Internet des objets (IoT).La maintenance prédictive devient optimisée pour gérer et stocker en toute sécurité de grandes quantités de données à partir de tout type de capteur IoT.

A partir de là, le champ de tout les possibles s'ouvrent en terme d'analyse En utilisant des algorithmes d'apprentissage automatique / IA de pointe, les données sont analysées en temps réel et fournissent les résultats souhaitez. Une des solution possible: prédire des choses comme la prochaine période de maintenance pour un actif et quels actifs sont les plus utilisés les uns par rapport aux autres en temps réel. Les données peuvent être disponibles via des tableaux de bord conviviaux ou via des services Web qui se connectent à n'importe quelle application.

Peu import le budget maintenance, il sera toujours possible de faire au mieux avec les moyens disponible.

3 Analyse du Bitcoin

NB: Je tiens à preciser tout de même que je n'ai aucune connaissance en trading, à la bourse, l'exercice est simplement effectué dans le but de mettre en évidence les bases que j'ai sur pandas pour l'analyse des phénomèmes qui evoluent au cours du temps. Après l'utilisation des outils s'arrêt ou peut s'arreter notre imagination (ce que l'on veut réellement observer). A force de pratique, on apprend encore plus.

3.1 Importation des librairies numpy, pandas pour la manipluation des données et matplotlib pour la visualisation des graphes

[1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
```

3.2 Importation du dataset (format csv)

[2]:
```
bitcoin=pd.read_csv('BTC-EUR.CSV')
```

3.3 Verification de l'importation du dataset en ayant un aperçu des 3 premières lignes du tableau

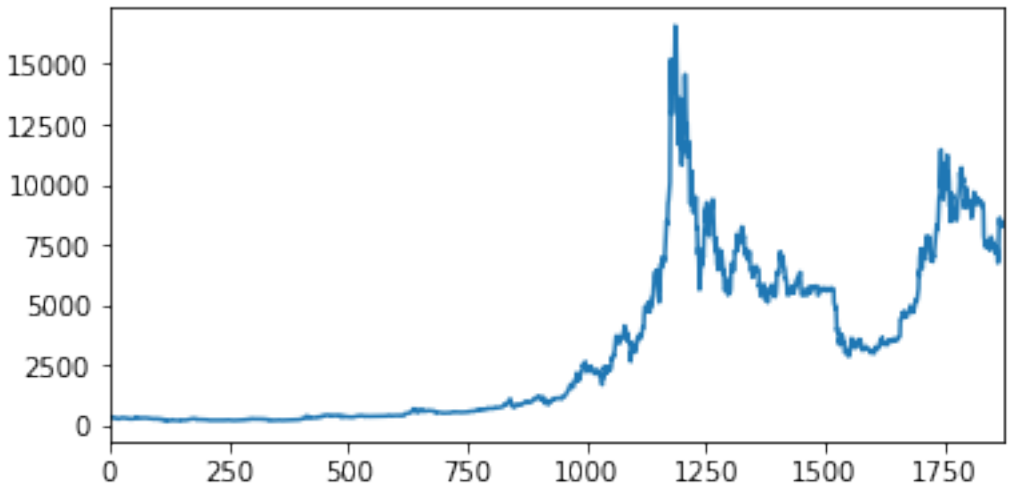
[3]:
```
bitcoin.head(3)
```

[3]:

	Date	Open	High	Low	Close	Adj Close	Volume
0	2014-09-17	359.546204	361.468506	351.586884	355.957367	355.957367	16389165
1	2014-09-18	355.588409	355.505402	319.789459	328.539368	328.539368	26691849
2	2014-09-19	328.278503	330.936707	298.921021	307.761139	307.761139	29560102

3.4 Observation de l'évolution du Bitcoin

[4]:
```
bitcoin['Close'].plot(figsize=(6, 3))
plt.show()
```



3.4.1 Observations:

On peut constater que nous n'avons pas en abcisse l'evolution suivant des dates comme 2010, 2020 ou sept-2019 etc. C'est normal, pour le moment nous n'avons pas encore indiqué à pandas que nous voulons travailler sur une base temporelle. Donc dans notre dataset on n'a un index par défaut que l'on peu voir dans notre tableau au point 3 et sur l'axe des abcisse au point 4. Voir ci-dessous (la fonction pour l'afficher l'index).

[5]:
```
print(bitcoin.index)
```

```
RangeIndex(start=0, stop=1874, step=1)
```

3.5 Date Time Index

Indique que la colonne index sera égale à nos dates avec index_col et parse_dates

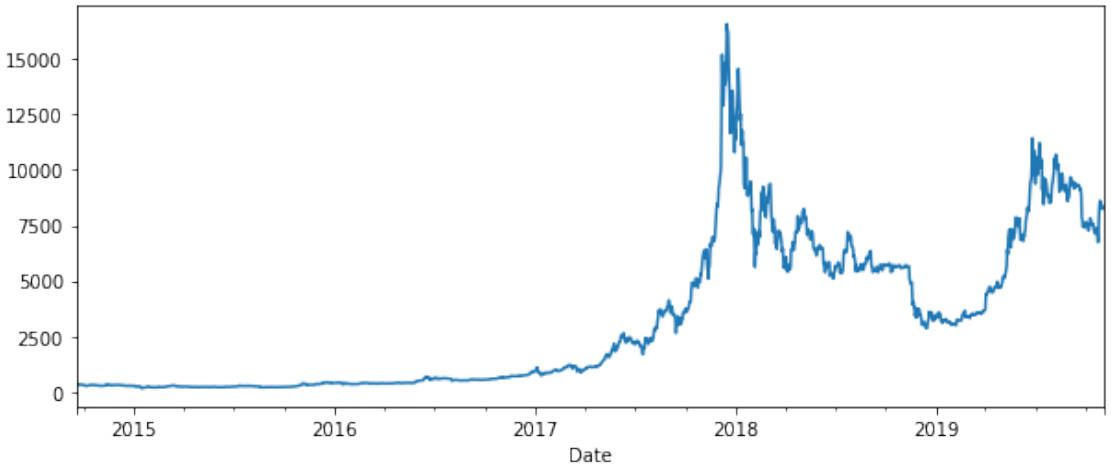
[6]:
```
bitcoin=pd.read_csv('BTC-EUR.CSV', index_col='Date', parse_dates=True)
```

[7]:
```
bitcoin.head(3)
```

[7]:

Date	Open	High	Low	Close	Adj Close	Volume
2014-09-17	359.546204	361.468506	351.586884	355.957367	355.957367	16389165
2014-09-18	355.588409	355.505402	319.789459	328.539368	328.539368	26691849
2014-09-19	328.278503	330.936707	298.921021	307.761139	307.761139	29560102

[8]:
```
bitcoin['Close'].plot(figsize=(10,4))
plt.show()
```



3.5.1 Observations:

On peut constater le changement d'index, d'abord sur notre tableau au dessus avec sa première colonne en Date et sur l'abcisse de notre graphe. Desormais on peut faire toutes les manipulations possibles avec les dates, parce que pandas comprend les notions de jours, mois, année et heure,min,sec.

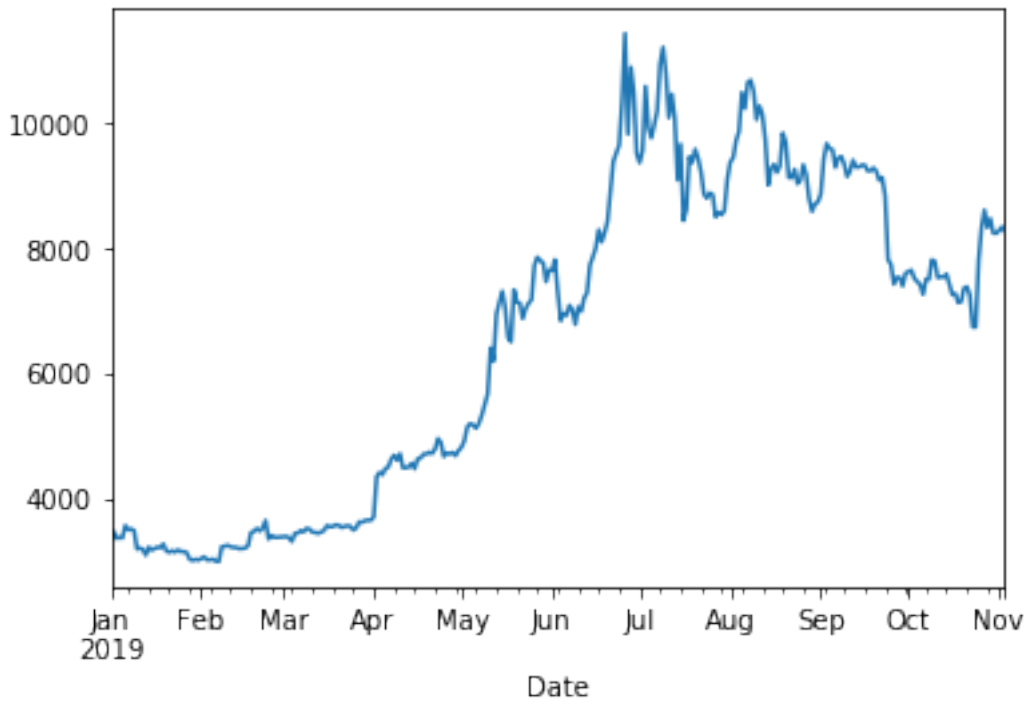
[9]:
```
#verification
bitcoin.index
```

[9]:
```
DatetimeIndex(['2014-09-17', '2014-09-18', '2014-09-19', '2014-09-20',
               '2014-09-21', '2014-09-22', '2014-09-23', '2014-09-24',
               '2014-09-25', '2014-09-26',
               ...
               '2019-10-25', '2019-10-26', '2019-10-27', '2019-10-28',
               '2019-10-29', '2019-10-30', '2019-10-31', '2019-11-01',
               '2019-11-02', '2019-11-03'],
              dtype='datetime64[ns]', name='Date', length=1874, freq=None)
```

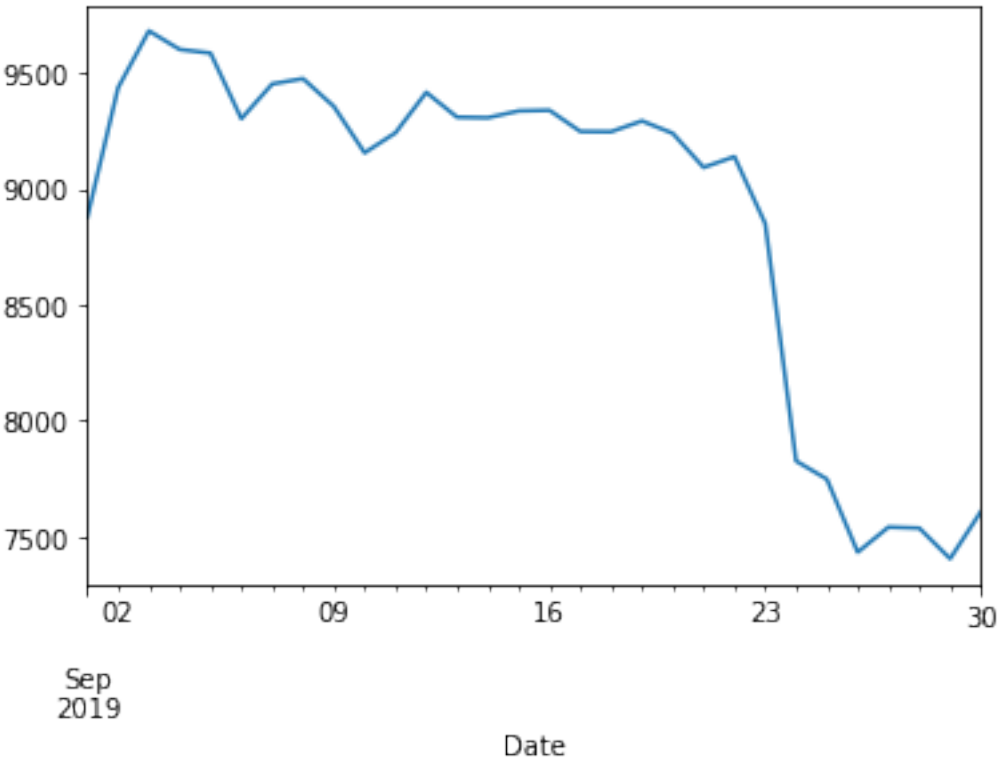
3.6 Analyse TimesSéries (Indexing et Slicing sur les dates)

Si on veut voir l'evoluion du bitcoin seulement en 2019.

[10]:
```
bitcoin['2019']['Close'].plot()
plt.show()
```



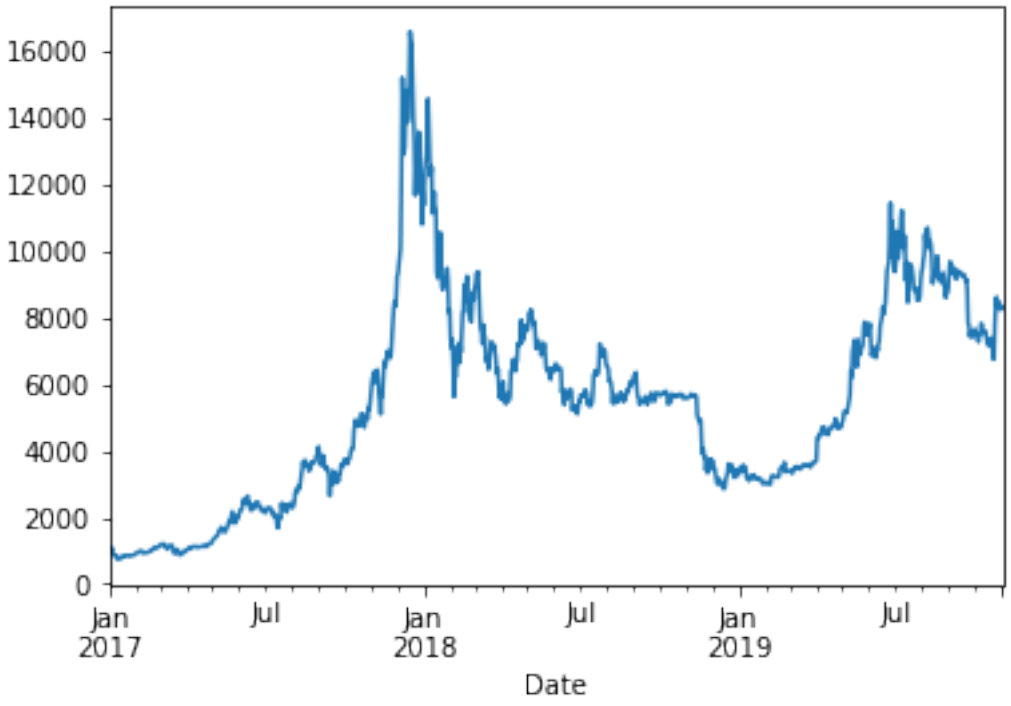
[11]:
```
#En septembre 2019
bitcoin['2019-09']['Close'].plot()
plt.show()
```



3.6.1 Observation:

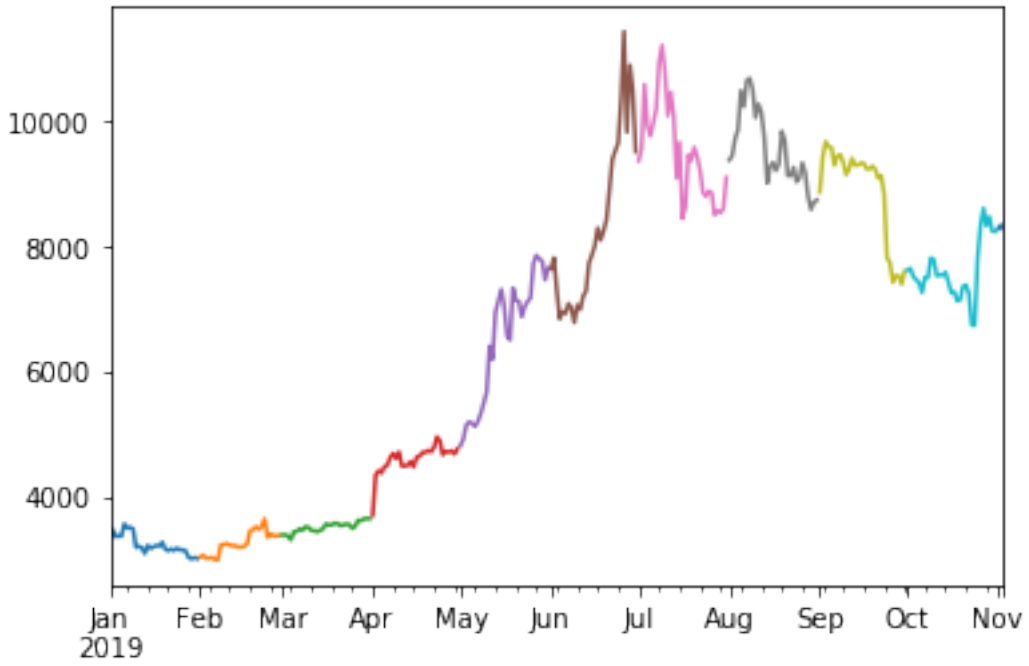
On peut voir que ça sait bien cassé la figure entre le 23 et le 30 du mois de septembre.

[12]:
```
# Entre 2017 et 2019
bitcoin.loc['2017':'2019', 'Close'].plot()
plt.show()
```



3.7 Fonction resample: Permet de regrouper nos données selon une frequence temporelle, super utile

[13]:
```
bitcoin.loc['2019', 'Close'].resample('M').plot() #Resample M=par Mois
plt.show()
```

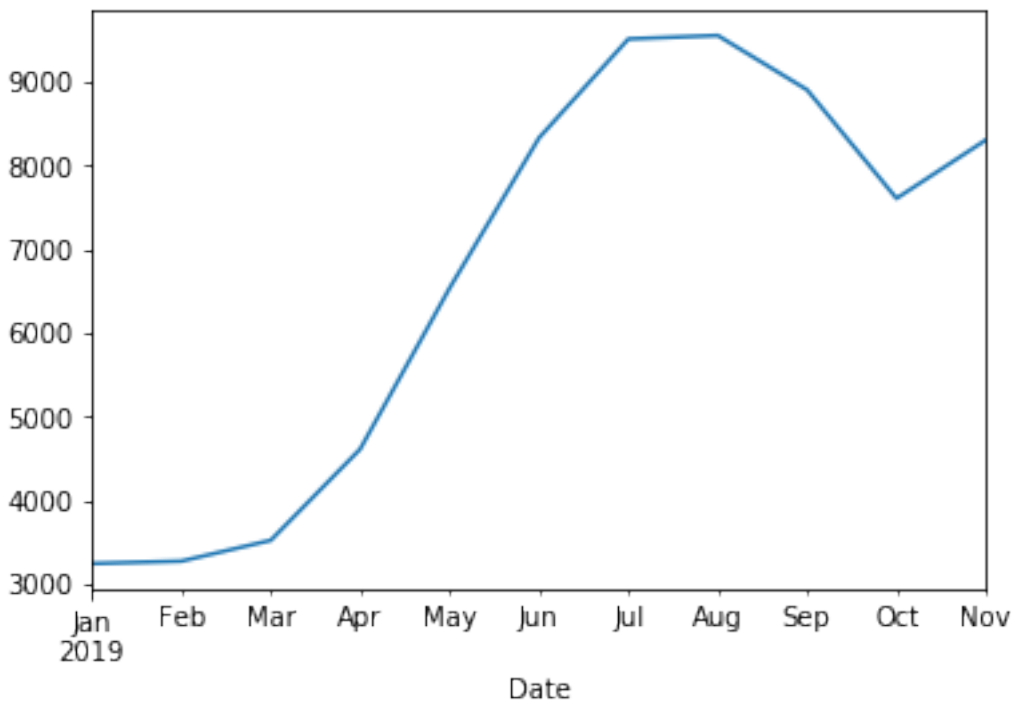


3.7.1 Observations:

On peut voir le changement de couleur après chaque interval de mois, exemple jan-feb ou may-jun. Qui dit groupe de données dit statistiques.

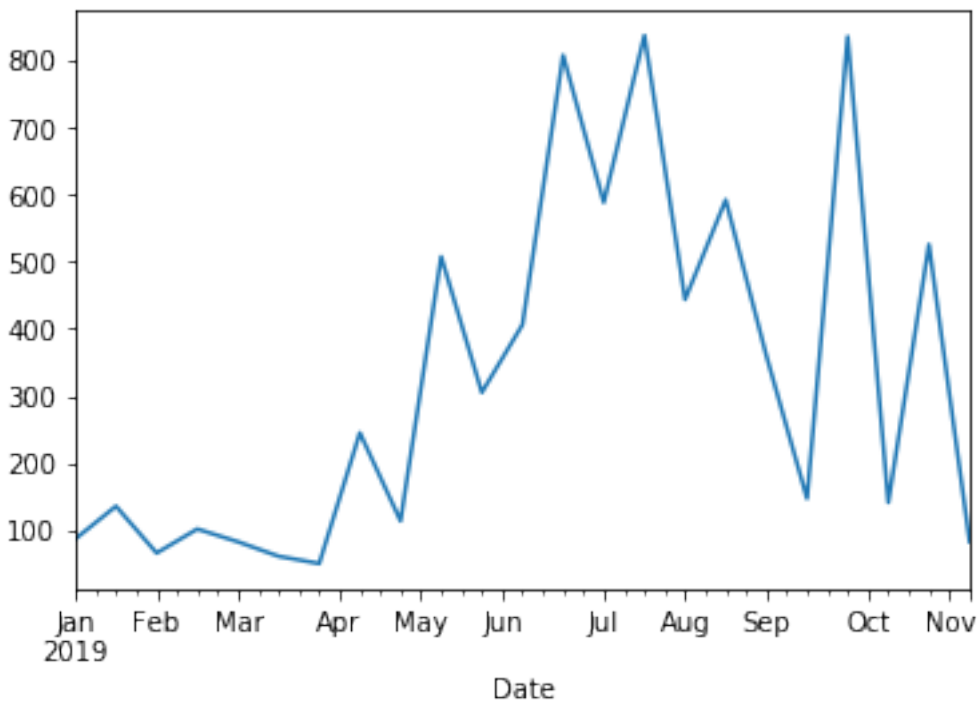
3.7.2 Resample, l'évolution du bitcoin 2019 en faisant la moyenne chaque M, possible d'écrire w=week ou 2w etc.

[14]:
```
bitcoin.loc['2019', 'Close'].resample('M').mean().plot()
plt.show()
```



3.8 L'écart-type avec std

[15]:
```
bitcoin.loc['2019', 'Close'].resample('2W').std().plot()
plt.show()
```

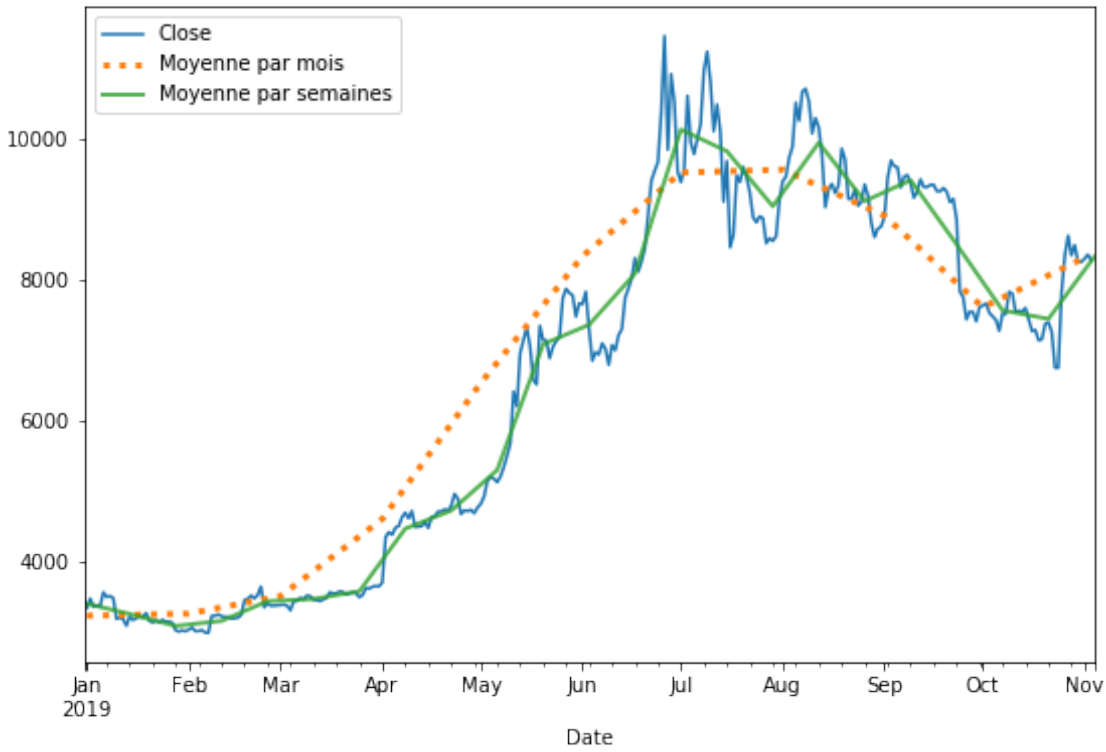


3.8.1 Observations:

On peut voir à quel point le bitcoin était volatile. De Jan-Apr il était stable, de May à nov il était très volatile.

3.9 Superposition de plusieurs courbes

[16]:
```
plt.figure(figsize=(9, 6))
bitcoin.loc['2019', 'Close'].plot()
bitcoin['2019']['Close'].resample('M').mean().plot(label='Moyenne par mois',␣
 ↪lw=3, ls=':')
bitcoin['2019']['Close'].resample('2W').mean().plot(label='Moyenne par␣
 ↪semaines', lw=2, alpha=0.8)
plt.legend()
plt.show()
```



3.9.1 Observations:

Nous avons superposé nos trois graphes (ça peut sembler impressionnant mais y'a rien de compliqué). le premier c'est l'evolution du bitcoin sur 2019. le second c'est l'evolution de la moyenne par mois. le troisième c'est l'evolution de la moyenne par semaine. ls, lw, alpha –> paramètre de style de nos courbe, pour les differencier.

3.10 Fonction aggregate

La fonction aggregate est souvent utilisée par dessus resample pour ressortir quelques statistiques comme la moyonne(mean(), l'écart-type (std), les valeurs max et min.

[17]:
```
bitcoin['Close'].resample('W').agg(['mean','std','min','max']).head(5)
```

[17]:

Date	mean	std	min	max
2014-09-21	324.329858	19.437050	307.761139	355.957367
2014-09-28	319.654266	13.412409	297.578705	339.189758
2014-10-05	286.932312	19.846387	256.162079	306.417480
2014-10-12	280.593305	13.560469	261.076508	299.508667
2014-10-19	306.296256	5.875245	298.801849	317.118896

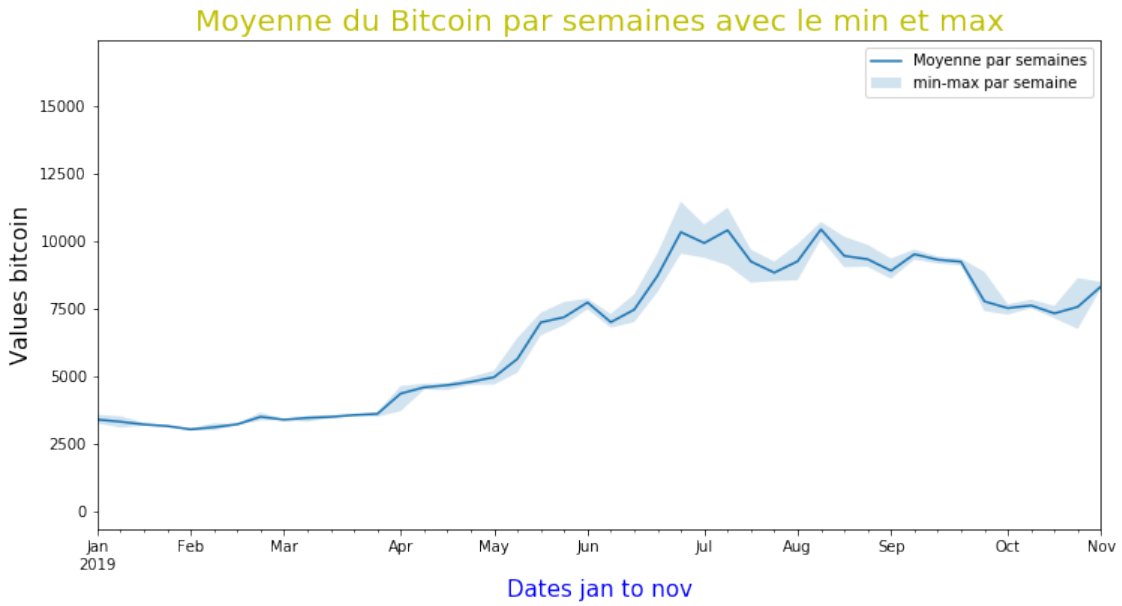
3.10.1 Courbe moyenne du bitcoin par semaine

[18]:
```
m=bitcoin['Close'].resample('W').agg(['mean','std','min','max'])

plt.figure(figsize=(12, 6))

m['mean']['2019'].plot(label='Moyenne par semaines')
plt.fill_between(m.index, m['max'], m['min'], alpha=0.2, label='min-max par␣
 ↪semaine')

plt.title('Moyenne du Bitcoin par semaines avec le min et max', color='y',␣
 ↪fontsize=20)
plt.ylabel('Values bitcoin', fontsize=15)
plt.xlabel('Dates jan to nov', fontsize=15, color='blue')
plt.legend()
plt.show()
```



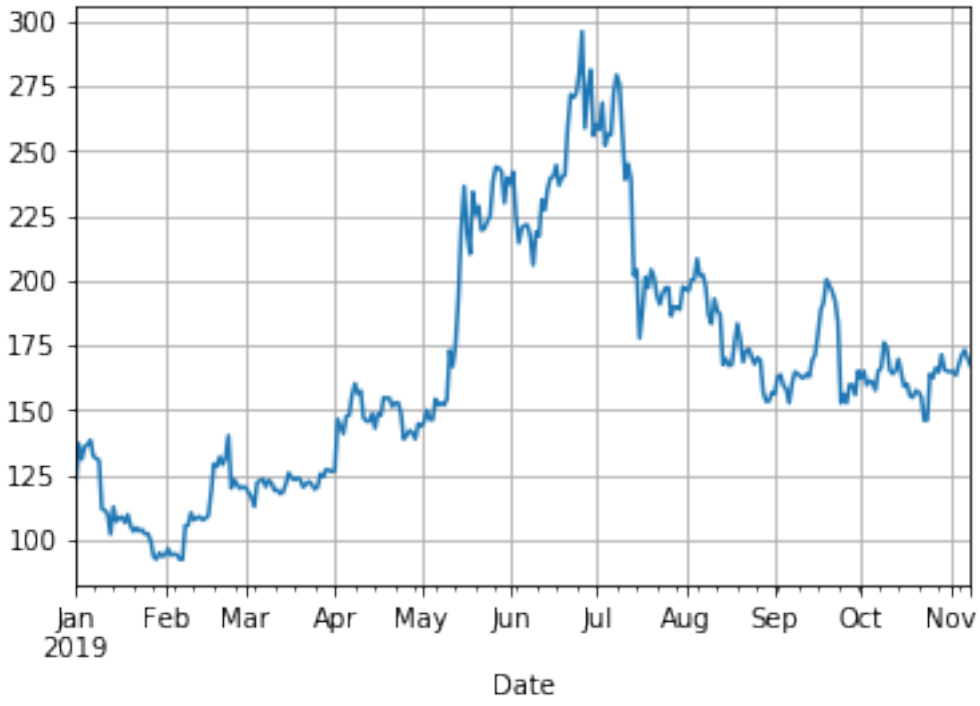
4 Assembler les datasets

Ici nous allons dans un premier temps importer le fichier sur les données ethereum (monnaie Cryptographique comme le Bitcoin) et verifier son importation. Dans la partie 10 nous allons assembler les deux fichiers ethereum$_{B}itcoinpourtenterdelescomparer.Importationdudatasetethereum$

[20]:
```
ethereum.head()
```

[20]:
```
                Open      High       Low     Close  Adj Close   Volume
Date
2015-08-07  2.592821  3.231847  2.299453  2.526771   2.526771   149784
2015-08-08  2.546495  2.551098  0.651468  0.686651   0.686651   614518
2015-08-09  0.643639  0.801941  0.573504  0.640651   0.640651   485733
2015-08-10  0.651688  0.662660  0.579996  0.643283   0.643283   368004
2015-08-11  0.642956  1.024735  0.604369  0.967177   0.967177  1325151
```

[21]:
```
ethereum.loc['2019','Close'].plot()
plt.grid(True)
plt.show()
```



4.1 Fonction merge pour l'assemblage de deux datasets

Definir une colonne sur laquelle l'assemblage doit s'effectuer, on l'occurence on prendra Date (on=Date) puisqu'il la partage en commun et ils ont les même valeurs. On choisira dans un second temps la base sur laquelle les assembler. inner, outer, left, right. Avec inner on les assemblera seulement sur l'index qu'ils ont en commun. Donc toutes les valeurs du bitcoin datant d'avant 2015-08-07 ne seront pas prises en compte.

[22]:
```
pd.merge(bitcoin, ethereum, on='Date', how='inner' ).head()
```

[22]:
```
                Open_x      High_x       Low_x     Close_x  Adj Close_x  \
Date
2015-08-07  255.233948  255.762100  253.128769  254.840027   254.840027
```

```
2015-08-08  254.983139  255.152679  237.635574  237.897186   237.897186
2015-08-09  238.005646  243.371613  237.414993  241.952362   241.952362
2015-08-10  242.312881  243.509033  239.726135  240.143463   240.143463
2015-08-11  240.027252  244.892654  240.676193  244.892654   244.892654

            Volume_x    Open_y    High_y     Low_y   Close_y  Adj Close_y  \
Date
2015-08-07  38724637  2.592821  3.231847  2.299453  2.526771     2.526771
2015-08-08  53352474  2.546495  2.551098  0.651468  0.686651     0.686651
2015-08-09  21713764  0.643639  0.801941  0.573504  0.640651     0.640651
2015-08-10  19049668  0.651688  0.662660  0.579996  0.643283     0.643283
2015-08-11  23035866  0.642956  1.024735  0.604369  0.967177     0.967177

            Volume_y
Date
2015-08-07    149784
2015-08-08    614518
2015-08-09    485733
2015-08-10    368004
2015-08-11   1325151
```

Changer les suffixes x et y par leur noms respectives _btc pour bitcoin et _eth pour ethereum

[23]:
```
pd.merge(bitcoin, ethereum, on='Date', how='inner',suffixes=('_btc', '_eth')).
↪head()
```

[23]:
```
              Open_btc    High_btc     Low_btc   Close_btc  Adj Close_btc  \
Date
2015-08-07  255.233948  255.762100  253.128769  254.840027     254.840027
2015-08-08  254.983139  255.152679  237.635574  237.897186     237.897186
2015-08-09  238.005646  243.371613  237.414993  241.952362     241.952362
2015-08-10  242.312881  243.509033  239.726135  240.143463     240.143463
2015-08-11  240.027252  244.892654  240.676193  244.892654     244.892654

            Volume_btc  Open_eth  High_eth   Low_eth  Close_eth  \
Date
2015-08-07    38724637  2.592821  3.231847  2.299453   2.526771
2015-08-08    53352474  2.546495  2.551098  0.651468   0.686651
2015-08-09    21713764  0.643639  0.801941  0.573504   0.640651
2015-08-10    19049668  0.651688  0.662660  0.579996   0.643283
2015-08-11    23035866  0.642956  1.024735  0.604369   0.967177

            Adj Close_eth  Volume_eth
Date
2015-08-07       2.526771      149784
2015-08-08       0.686651      614518
2015-08-09       0.640651      485733
2015-08-10       0.643283      368004
```

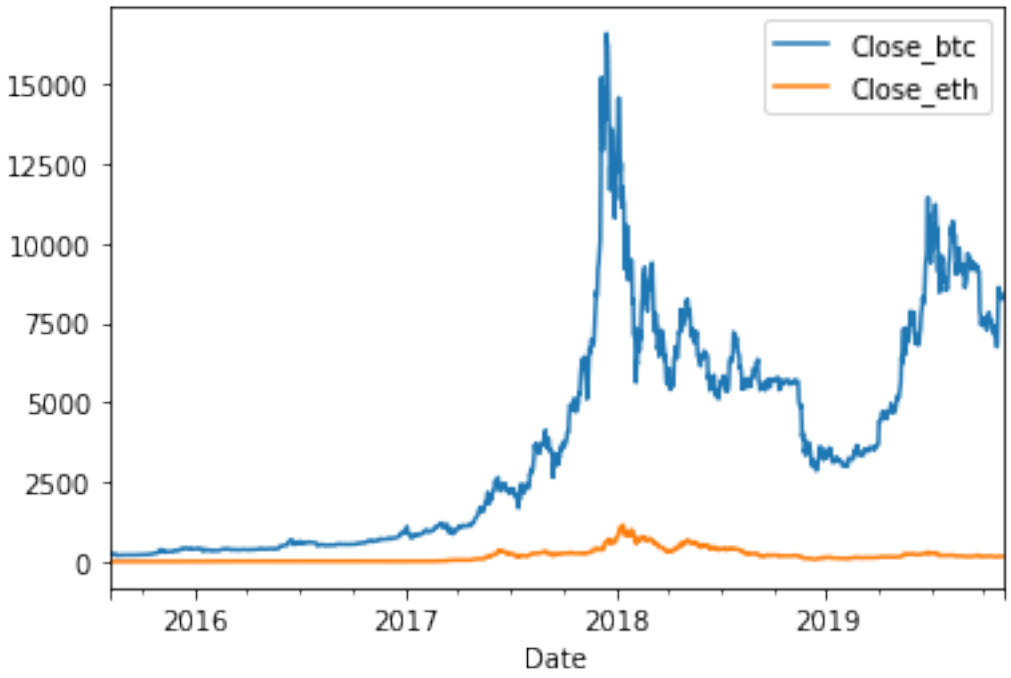
```
2015-08-11        0.967177      1325151
```

4.2 Courbe de tendance du Bitcoin et de l'ethereum

[24]:
```
btc_eth=pd.merge(bitcoin, ethereum, on='Date', how='inner',suffixes=('_btc',␣
↪'_eth'))
```

[25]:
```
btc_eth[['Close_btc', 'Close_eth']].plot()
```

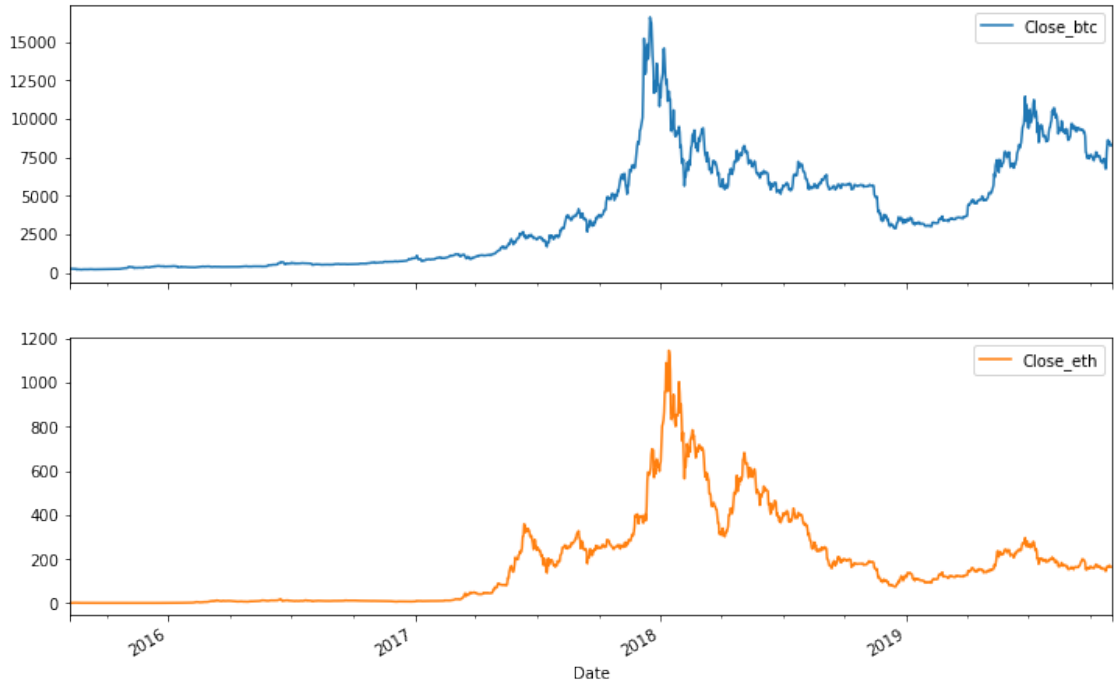
[25]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x2d4d59b37f0>
```



4.2.1 Observations:

Les deux crytographes ne partagent pas la même echelle, ceux qui rend l'observation de eth un peu plus compliqué. Nous allons utiliser la fonction subplot de matplotlib pour avoir les deux tendances sur un même plan.

[26]:
```
btc_eth[['Close_btc', 'Close_eth']].plot(subplots=True, figsize=(12,8))
plt.show()
```



4.2.2 Observations:

On obtient sur un même plan deux graphiques différents. Il est possible de dire que le bitcoin et ethereum sont bien corréler.

4.3 Verifions à combien il sont corrélés avec une matrice de corrélation

[27]:
```
btc_eth[['Close_btc', 'Close_eth']].corr()
```

[27]:
```
            Close_btc  Close_eth
Close_btc    1.000000   0.779899
Close_eth    0.779899   1.000000
```

4.3.1 Observations:

On constate une corrélation de 77 pourcent. On pourra par la suite l'observer sur un graphe avec la fonction sns.heatmap de seaborn.

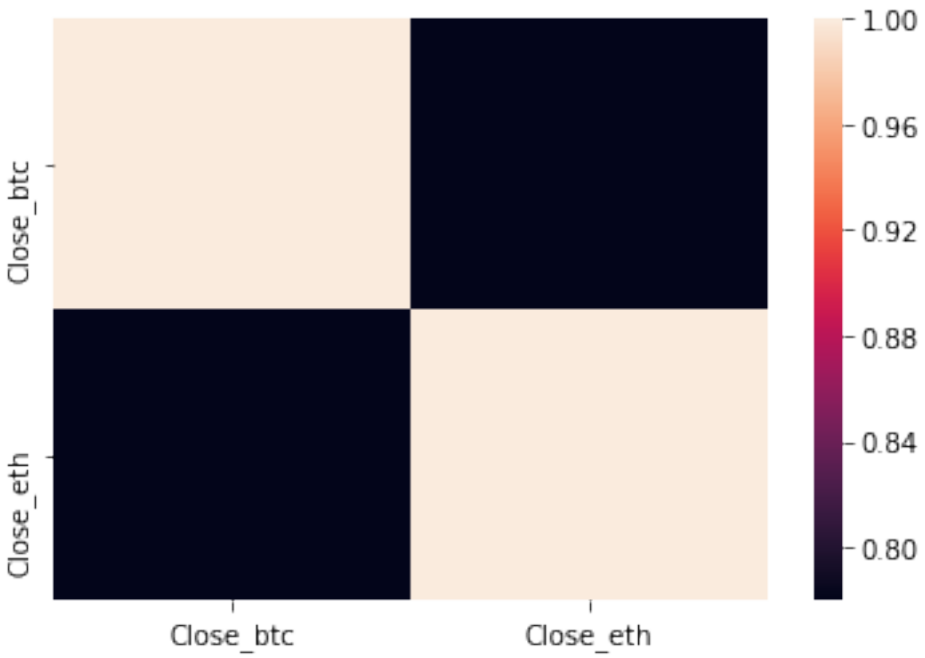
[ ]:
```

```

[28]:
```
import seaborn as sns
```

[29]:
```
corrélation=btc_eth[['Close_btc', 'Close_eth']].corr()
sns.heatmap(corrélation)
```

[29]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x2d4d6592780>
```



5 Conclusion

Voila, quelques que fonctions de la librairie pandas, il est possible d'aller encore plus loin. Ce travail est effectué dans Jupyter nootebook et le rendu des importations des tableaux sont beaucoup plus agréable à voir que la version convertie en pdf? qui ne laisse apparaître que les valeurs.