

金融財務研究会・午前の部

- はじめに -

Pythonの歴史？

Anaconda4.4.0(Jupyter)のインストール

各種ライブラリーのインストール

Python組み込みライブラリ

データモデル(リスト、タプル、辞書)

pandas-datareaderの使い方

データの可視化(matplotlib,seaborn)

-習得目標-

Jupyterの使い方

Pythonを使いこなす最低限の知識の習得

リスト、タプル、辞書の理解

Webからのデータのダウンロード

データの可視化

Pythonの歴史

Pythonは、1990年に登場したオープンソースのプログラミング言語であり、そのコードは、誰にとっても読みやすく、理解しやすいことが最大の特徴である。そして、同じ仕事をするプログラムはほぼ同じようなソースコードになる構造を持っている。従って、他人の書いたプログラムは読みやすく、また自分でも書きたいコードを素早く書くことができる。そして、そのようにして書かれたソースコードは変更することなしに、Window, Mac, Linuxなどさまざまなオペレーティングシステム上で実行可能である。開発者はオランダ人のグイド・ヴァンロッサムである。

Pythonはオープンソースライセンスで公開されていて、Python Japanなどの公式サイトからダウンロードできる。本書ではPythonをブラウザ上からインタラクティブに使うことのできるJupyter Notebookを用いる。

Anaconda のインストール

Anaconda Scientific Python Distributionが無料で提供するパッケージを導入すれば、(ほぼ必要なライブラリーが同時にインストールされる。Pythonの良さをまず知るために、AnacondaのホームページからPythonをインストールしてみよう。しかし、その前に忘れてはならないことがある。

ブラウザーとしてGoogle Chromeをインストール。

Anacondaのパッケージをインストール

パワーポイントの資料参照

または

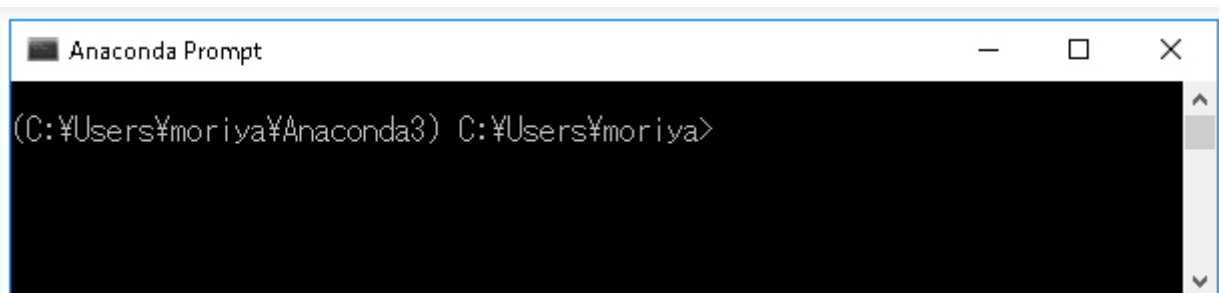
<https://qiita.com/innovation1005/items/2f433d6d859f075033a7>
(<https://qiita.com/innovation1005/items/2f433d6d859f075033a7>)

追加ライブラリのインストール

多くのデータ解析用のライブラリが標準インストールされているAnacondaでもすべてのライブラリが事前にインストールされているわけではありません。ExcelでもVBAを使うときにはエディターを起動できるように設定し、必要なライブラリを自分で選択しなければなりません。Anacondaでも同じような手順が必要です。そこで使うのがpipという便利なツールです。pipはAnacondaをインストールすると標準でインストールされていますので既に使っている読者の方もいるかもしれません。

Dos Promptの起動

windows menu> Anaconda3(64bit)> Anaconda Prompt>

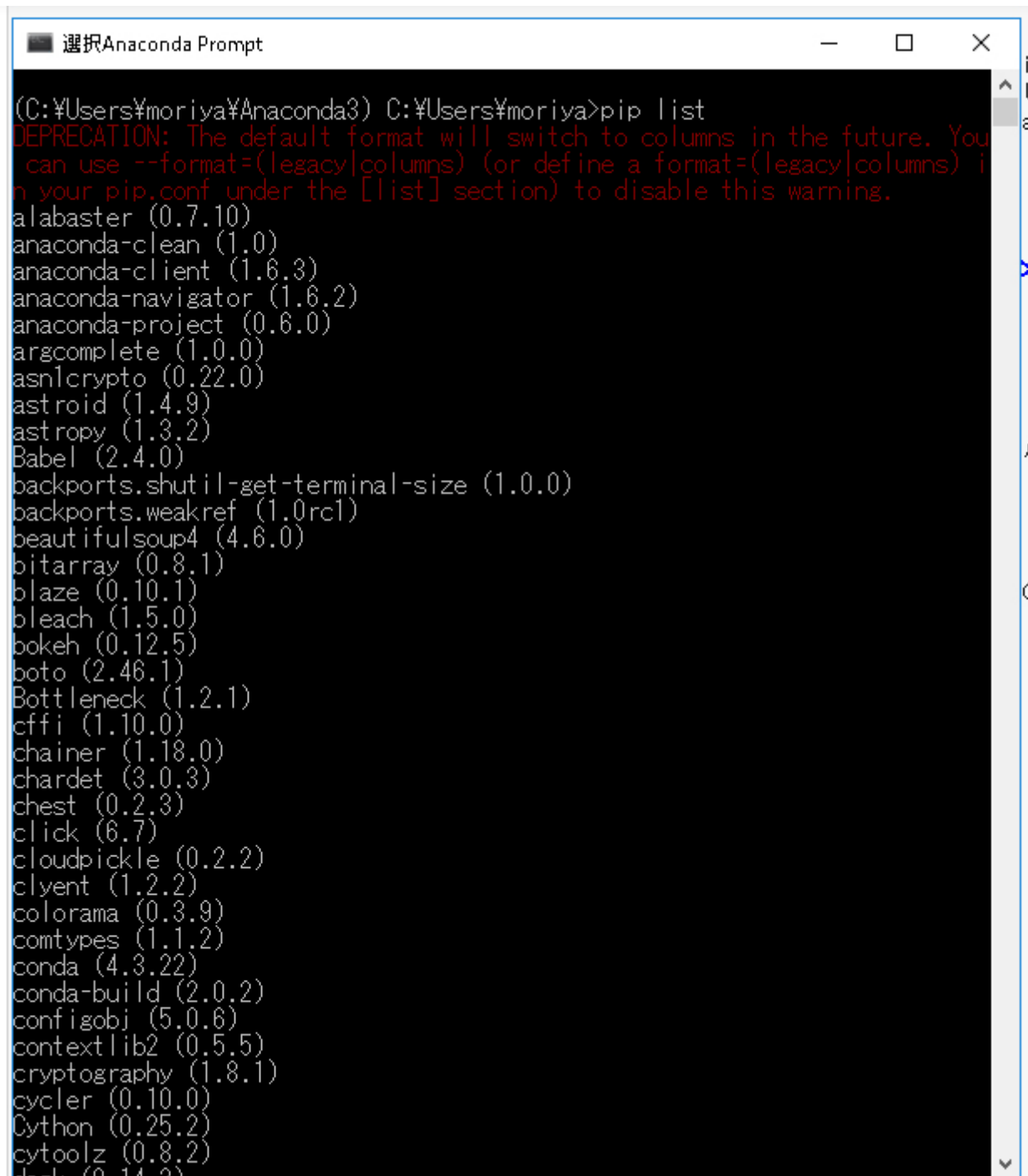


pipとconda

Anacondaに標準装備されていないライブラリを使う際に少し気を付けなければならないのはそのバージョンです。新しいライブラリを使うとほかのライブラリについても最近のバージョンが要求される場合があります。そのようなときに便利なのが、

pip list

です。このツールを実行すると、インストールされているすべてのライブラリとそのバージョンのリストを返してくれます。



```
(C:\Users\moriya\Anaconda3) C:\Users\moriya>pip list
DEPRECATION: The default format will switch to columns in the future. You
can use --format=(legacy|columns) (or define a format=(legacy|columns) i
n your pip.conf under the [list] section) to disable this warning.
alabaster (0.7.10)
anaconda-clean (1.0)
anaconda-client (1.6.3)
anaconda-navigator (1.6.2)
anaconda-project (0.6.0)
argcomplete (1.0.0)
asn1crypto (0.22.0)
astroid (1.4.9)
astropy (1.3.2)
Babel (2.4.0)
backports.shutil-get-terminal-size (1.0.0)
backports.weakref (1.0rc1)
beautifulsoup4 (4.6.0)
bitarray (0.8.1)
blaze (0.10.1)
bleach (1.5.0)
bokeh (0.12.5)
boto (2.46.1)
Bottleneck (1.2.1)
cffi (1.10.0)
chainer (1.18.0)
chardet (3.0.3)
chest (0.2.3)
click (6.7)
cloudpickle (0.2.2)
clyent (1.2.2)
colorama (0.3.9)
comtypes (1.1.2)
conda (4.3.22)
conda-build (2.0.2)
configobj (5.0.6)
contextlib2 (0.5.5)
cryptography (1.8.1)
cycler (0.10.0)
Cython (0.25.2)
cytoolz (0.8.2)
dask (0.14.2)
```

古くて必要なものは最新版に更新ができます。それは

`pip update ライブラリ名`

で行います。新しいライブラリは

`pip install ライブラリ名`

を用いてインストールします。もしバージョンを指定したいのであれば、

`pip install ライブラリ名==バージョン`

でインストールできます。では必要なライブラリをインストールしましょう。

pandas_datareaderのインストール¶

`pip install pandas-datareader`

seabornのインストール

pip install seaborn

Python プログラムをはじめよう

まずPythonで戸惑うのはべき乗演算と配列の作成ではないだろうか？一般的に、この2つを多用するので、この2つの方法が簡単に手に入らないと、それだけで投げ出してしまう。多くのユーザーはPythonのべき乗演算が

半角アスタリスク(*)

を2つ並べると知ったとたんに断念してしまう。ほとんどのプログラミング言語ではハットマーク(^)1つで済んでいた演算が2文字になってしまうのだから当然だ。

配列も問題だ！WEB上で「Python、配列」と検索すると「データ構造」、「リスト、タプル、辞書」に関するページが多数ヒットされる。しかし、なかなか配列に行き当たらない。そして、「配列」という言葉を発見するとそれは「Numpy配列」であったりする。それでもリスト、タプル、辞書が配列に似ているという記述から、仕方なくこれらを使い始めるが、ユーザーが多用する2次元配列で壁にぶつかってしまう。これに随分と手間取った挙句にここで断念してしまうユーザーもいるのではないだろうか？

しかし、Pythonを使う理由はこのデータ構造にあるといっても過言ではない。金融関連のユーザーであればこれらの演算スピードを実感してしまうと、もう手放せなくなってしまう。それにデータベースを構築する際の使い勝手も決して悪くはない。では心の準備ができたので前進！

プログラムの書き方の様式

Pythonのプログラムが、誰にとっても読みやすく、理解しやすいのは、プログラムのブロック構造を『字下げ』により決定するためである。そのために行構造を理解しておくと便利である。

1. 行(物理行)は改行によって終わる。
2. 複数行(物理行)にわたってプログラムを書くときには行の終わりにバックスラッシュ文字()を置く。そうするとそれは論理行になる。
3. 論理行の先頭は実行分のグループ化の判断に用いられる。『字下げ』の位置が重要な理由はここにある。
4. コメントが1行の場合には先頭をシャープ(#)で始める。複数行にわたる場合には

データモデル

Pythonではすべてのデータがオブジェクトで、プログラム自身もオブジェクトである。オブジェクトは型と値を持つ。

シーケンス型

シーケンス型とは有限な順序を持つ要素の集合である。

変更不能なシーケンス型：文字列、タプル型

1. 文字列: 順序を持った文字の集合
2. タプル: 数値、文字、文字列をカンマ(,)で区切って定義する。

変更可能なシーケンス型：リスト型

リストは数値、文字、文字列などを角括弧[]で囲って定義する。

数値型

整数型、長整数型、浮動小数点型、複素数型がある。

マップ型：インデックスを持つ有限のオブジェクトから成る集合

辞書(ディクショナリ)：数値、文字、文字列を用いてインデックスと要素をペアとして構成する。

呼び出し可能型

関数の呼び出しが可能な型である。

1. ユーザー定義関数
2. ユーザー定義メソッド
3. 組み込み関数
4. 組み込みメソッド

式

式の表現の方法について説明する。

算術変換

1. 1つの引数が複素数型であると、他も複素数型になる。
2. 引数に複素数型が無く、1つに浮動小数点があれば他も浮動小数点になる。
3. 引数に複素数、浮動小数点が無く、1つが長整数であれば他も長整数である。

算術演算

四則演算子は「+,-,*,/」で行う。

べき乗演算

べき乗演算はハットマーク(^)ではなくダブル半角アスタリスク(**)で行う。

比較演算

比較は"<", ">", "==", "<>", "!="で行う。また、比較はいくらでも連鎖することができる。a<b<c<dなど。比較演算の結果は真(True)、または偽(False)である。その他には"is", "not is", "in", "not in"がある。

1. "is", "not is"はデータの属性をチェックする。
2. "in", "not in"はコレクション型の要素のチェックに用いる。

単純文

文が1つの論理行に収まる場合を単純文という。ここでは直感的な記述にとどめる。

print文

得られたオブジェクトを書き出す。

break文

最も内側のループを終了させる。

import文

モジュールを検索し、必要に応じて初期化を行い、またローカルな名前を定義することもある。

複合文

複合分は複数行にまたがり、他の分の実行に何らかの制御を与える。ここではif文とfor文を説明する。本書ではこれ以外は使わない。

if 文 :条件分岐を実行する。

入力 [1]:

```
a=['x','y']
if 'x' in a:#もし文字xがリストaに含まれればOKと表示する。
    print("OK")
```

OK

for 文 : for 文ではループカウンタではなく、各要素に対して処理を最後まで繰り返す。

入力 [2]:

```
a=[1,2,3]
for i in a:
    print(i)#改行される。
```

1
2
3

入力 [3]:

```
a=[1,2,3]
for i in a:
    print(i,end=" ")#print文の最後にカンマ(, end=" ")を置くと改行されない。
```

123

入力 [4]:

```
a=[1, 2, 3]
for i in a:
    print(i, end=" ")#print文の最後にカンマ(, end=" ")を置くと改行されない。
    if i==2:
        break
```

12

いよいよPython 標準ライブラリーについて学ぼう

Python標準ライブラリーは、プログラム言語の核となるデータ型と組み込み関数が含まれる。

主な組み込み関数

Pythonの組み込み関数はどこでも利用可能である。使用頻度の高いものは

1. int(x) : 整数に変換する。
2. float(x) : 浮動小数点に変換する。
3. len(x) : 長さを返す。
4. max(x) : 最大値を返す。
5. min(x) : 最小値を返す。
6. print(x) : 出力する。
7. range(x) : 1からxまでの整数のリストを返す。
8. sum(x) : 合計を返す。

組み込み型

組み込み型は主に、数値型、シーケンス型、マッピング型に分けられる。

数値型

Pythonは、変数、引数、戻り値などの型をプログラムの実行前に決める必要のない動的型付言語である。

浮動小数点の演算が必要な場合には初期値を設定するときに0.0のように小数点以下を記述しておくことをお勧めする。

変更不可能なシーケンス型

文字列は単引用符、2重引用符によってくくられる。

例えば'innovation'。

本書では日本語を扱い場合にunicode文字列を使うことがある。その際には頭にuを付ける。例えば、u'革新'。

文字列連結

文字列の連結は「+」で行う。

要素の抽出

a[i]:aのゼロから数えてi番目の要素を参照。

入力 [5]:

```
a="abc"  
a[1]
```

出力[5]:

```
'b'
```

入力 [6]:

```
a[0]
```

出力[6]:

```
'a'
```

スライス

`a[i:j]`: aのiからj番目までの要素を参照。

`a[i:j:k]`: aのiからj番目までの要素をk毎にスライス。

入力 [7]:

```
a="abc"  
a[0:2]
```

出力[7]:

```
'ab'
```

入力 [8]:

```
a[0:3:2]
```

出力[8]:

```
'ac'
```

in 演算子

in演算子は文字列、リスト要素の検索にもin演算子を用いる。

タプル(Tuple)

タプルはカンマ演算子(,)を使って定義する。必ずしも丸括弧で囲む必要はない。項目の数があらかじめ決まっていれば、メモリの使用効率が高い。また辞書のキーとして使用することができ、値のコピーができる。リストと混同しやすいが値の置き換えはできない。

入力 [9]:

```
a=(1, 2, 3)  
a[0]
```

出力[9]:

```
1
```


変更可能なシーケンス型

リスト

Pythonは幾つかの変数をグループとして扱うリストというデータ構造を持っている。リストはカンマにより分離された値、または要素で角括弧[]で囲われている。

入力 [10]:

```
a=[1, 2, 3]
a
```

出力[10]:

```
[1, 2, 3]
```

マッピング型

辞書(ディクショナリ、dictionary)

辞書は2つの要素から成り立っていて、ひとつはインデックスであり、もう一つはそれに関連する要素である。Pythonの描画ソフトであるmatplotlibで用いる色の指定に用いる記号の辞書を作ってみよう。

入力 [11]:

```
a={"a":1, "b":2, "c":3}
a
```

出力[11]:

```
{'a': 1, 'b': 2, 'c': 3}
```

入力 [12]:

```
a={"a":1, "b":2, "c":3}
a["a"]
```

出力[12]:

```
1
```

やっと来た科学分析用、その他のモジュール、ライブラリ

IPython

IPythonは対話式処理とソフトウェアの開発の効率を上げるために開発された対話処理型インタプリタである。IPythonプロジェクトは2001年にフェルナンド・ベレスのプロジェクトとして発足した。現在ではそのプロジェクトは対話型ノートブックフォーマットの開発と高速の平行計算エンジンの開発を担っている。

Numpy

NumpyはNumerical Pythonを略して表記したものである。Pythonの科学計算の基本的な部分を構築する。高速で効率のよい科学計算を可能とするデータ構造を提供する。

pandas

金融時系列データのデータベースとしての機能を意識して設計されたモジュール。pandasの存在がPythonを使う1つの理由である。

matplotlib

matplotlibはPython用の2Dの描画ソフトである。

Scipy

ScipyはNumPy配列を使うことで科学、工学の分野で一般的に使われる積分、統計的検定、固有値の算出などを高速に行うことを可能にしている。

Statsmodels

統計モデル、時系列モデルのクラスと関数を提供する。

pandas_datareaderを用いたデータのダウンロード

pandas-datareaderはGoogle Financeをはじめとする民間のWEBサイトだけではなく、多くの公的なサイトからでもデータをダウンロードできます。特に、世界銀行とかOECDとかいうマルチナショナルといわれる機関の運営するサイトからデータをダウンロードできるのが特徴です。今回はGoogle FinanceとFREDからデータをダウンロードしてみましょう。Google Financeは株価とIPO、株式に関する情報、外国為替、金利、投資信託、企業情報などに関する米国の総合サイトです。

また、FREDはセントルイス連銀が運営する経済・金融関連の情報サイトです。Google Financeから米国のアップル社の株価と関連情報を、FREDからは日経株価指数を取得してみましょう。

pandas-datareaderの使い方はいたって簡単です。必要な情報は、データコード、情報源、ダウンロードの開始日と終了日です。

```
pandas_datareader (コード、情報源、開始日、終了日)
```

です。開始日、終了日は省略が可能です。指定しないと、直近までのデータがダウンロードされます。ではために米国アップル社の株価を取得してみましょう。

入力 [22]:

```
import pandas_datareader.data as web
tsd=web.DataReader("AAPL", "yahoo", "2020/1/1")#得られるデータはpandasのDataFrameである。
tsd.tail()#tailは最後の5つのデータを返してくれる。
```

出力[22]:

	High	Low	Open	Close	Volume	Adj Close
Date						
2021-03-26	121.480003	118.919998	120.349998	121.209999	93958900.0	121.209999
2021-03-29	122.580002	120.730003	121.650002	121.389999	80819200.0	121.389999
2021-03-30	120.400002	118.860001	120.110001	119.900002	85671900.0	119.900002
2021-03-31	123.519997	121.150002	121.650002	122.150002	118162900.0	122.150002
2021-04-01	124.180000	123.000000	123.660004	123.224998	17030475.0	123.224998

入力 [23]:

```
#データの最後に？をつけるとデータの型情報を返してくれる。
tsd?
```

入力 [24]:

```
#lenを使って中のデータの行の数を取得する。
print(len(tsd))
```

316

入力 [25]:

```
#平均の取引高を取得する
print(tsd.Volume.mean())
```

147799923.971519

入力 [26]:

```
#平均の取引高を取得する
print(tsd["Volume"].mean())
```

147799923.971519

データの可視化

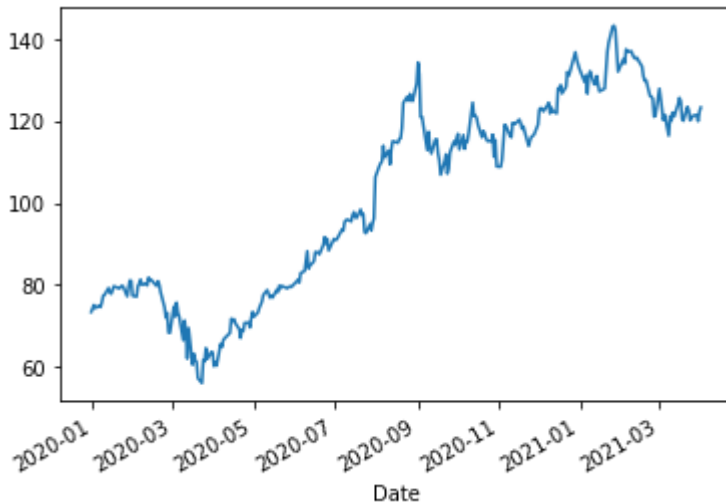
バーチャート

入力 [27]:

```
# %はグラフ表示のマジックコマンド、1ページに一回宣言すればOK
%matplotlib inline
tsd.Close.plot()
```

出力[27]:

<AxesSubplot: xlabel='Date'>



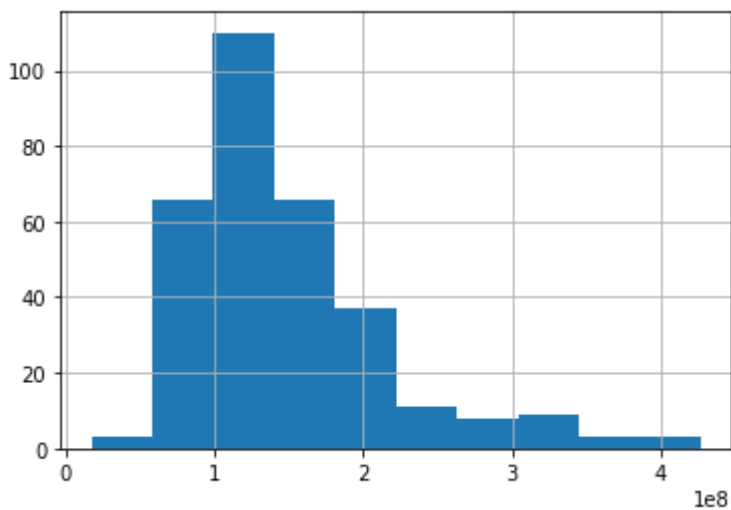
頻度図

入力 [28]:

```
tsd.Volume.hist()
```

出力[28]:

<AxesSubplot:>



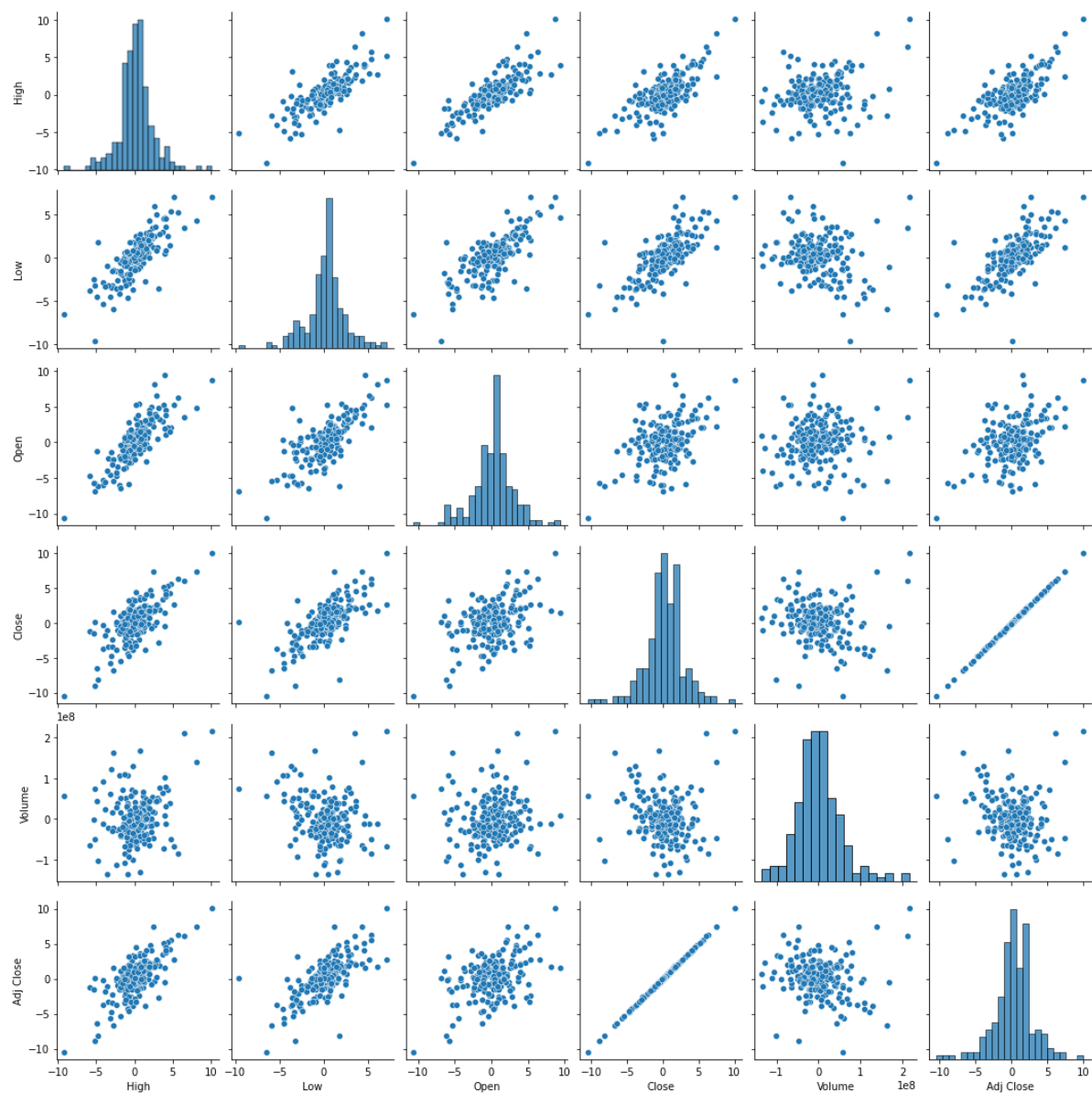
Seabornの活用

入力 [29]:

```
import seaborn as sns
#dtsd=tsd.drop("Volume",axis=1).diff().dropna()
dtsd=tsd.diff().dropna()
sns.pairplot(dtsd[:100])
```

出力[29]:

<seaborn.axisgrid.PairGrid at 0x2bd8708fb50>

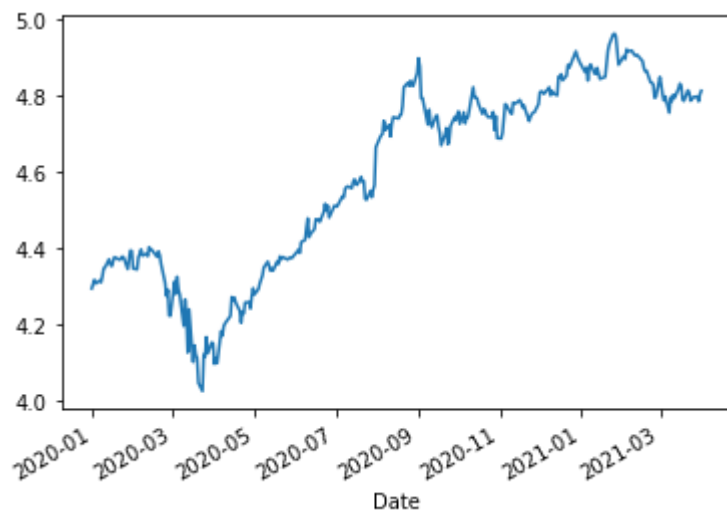


入力 [30]:

```
import numpy as np
Intsd=np.log(tsd).dropna()
Intsd.Close.plot()
```

出力[30]:

<AxesSubplot: xlabel='Date'>

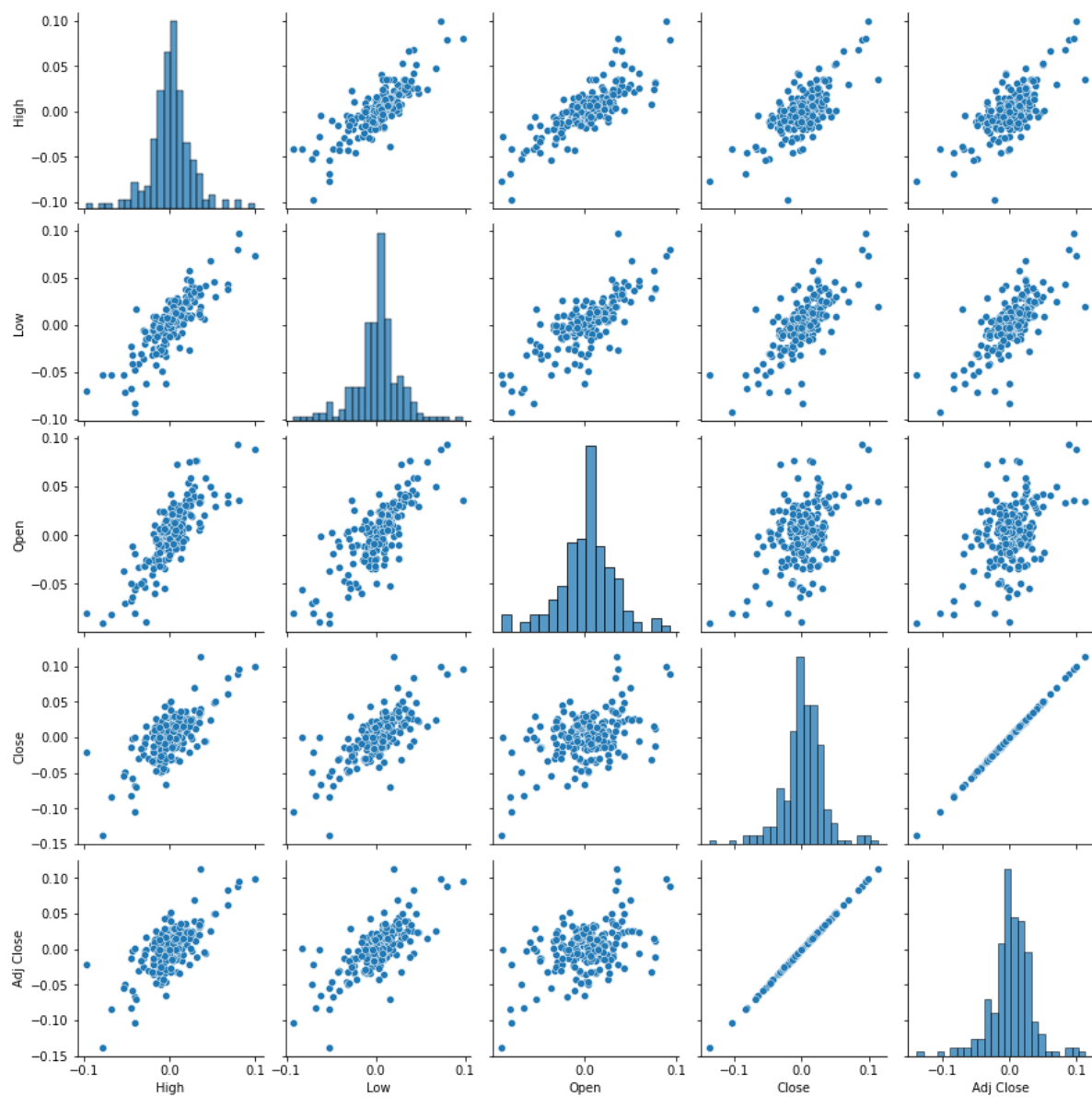


入力 [31]:

```
dIntsd=Intsd.drop("Volume",axis=1).diff().dropna()  
#dIntsd=Intsd.diff().dropna()  
sns.pairplot(dIntsd[:100])
```

出力[31]:

<seaborn.axisgrid.PairGrid at 0x2bd9d31ac10>



要約統計量

入力 [34]:

```
dIntsd.describe()
```

出力[34]:

	High	Low	Open	Close	Adj Close
count	315.000000	315.000000	315.000000	315.000000	315.000000
mean	0.001668	0.001683	0.001696	0.001644	0.001676
std	0.022188	0.023940	0.027399	0.027923	0.027918
min	-0.097218	-0.091858	-0.090584	-0.137708	-0.137708
25%	-0.009335	-0.009160	-0.011789	-0.011475	-0.011475
50%	0.001288	0.004260	0.003580	0.001096	0.001096
75%	0.011222	0.013071	0.015656	0.016232	0.016232
max	0.099904	0.097143	0.092812	0.113157	0.113157

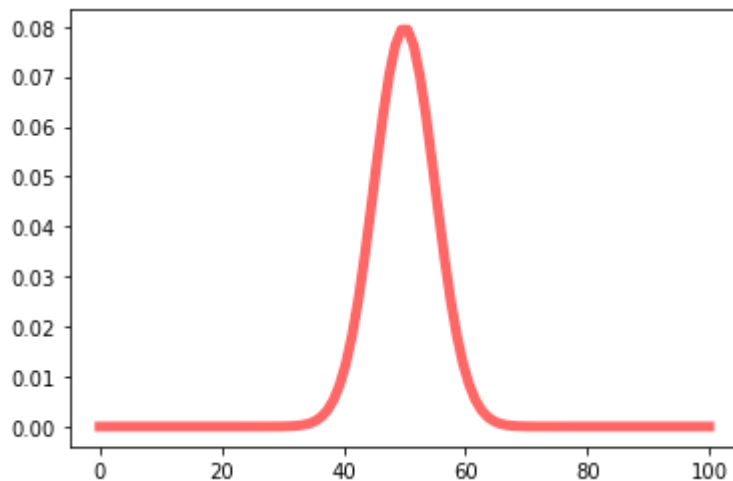
確率分布

入力 [35]:

```
from scipy.stats import norm
fig, ax = plt.subplots(1, 1)
x = np.linspace(0, 100, 100)
ax.plot(x, norm.pdf(x, loc=50, scale=5), 'r-', lw=5, alpha=0.6)
```

出力[35]:

[<matplotlib.lines.Line2D at 0x2bdf135d60>]

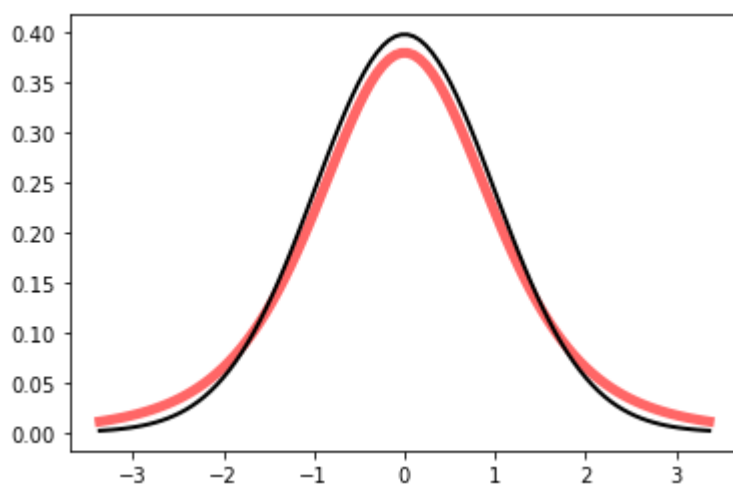


入力 [40]:

```
from scipy.stats import t
fig, ax = plt.subplots(1, 1)
df=5
x = np.linspace(t.ppf(0.01, df), t.ppf(0.99, df), 100)
ax.plot(x, t.pdf(x, df), 'r-', lw=5, alpha=0.6, label='t pdf')
df1=100
rv = t(df1)
ax.plot(x, rv.pdf(x), 'k-', lw=2, label='frozen pdf')
```

出力[40]:

[<matplotlib.lines.Line2D at 0x2bde7500190>]



入力 [41]:

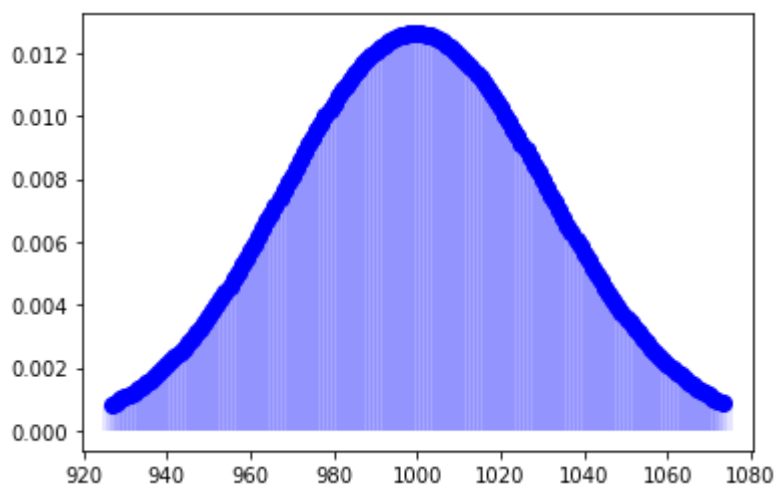
```
from scipy.stats import poisson

fig, ax = plt.subplots(1, 1)
mu = 1000
mean, var, skew, kurt = poisson.stats(mu, moments='mvsk')
print(mean, var, skew, kurt)
x = np.arange(poisson.ppf(0.01, mu),
              poisson.ppf(0.99, mu))
ax.plot(x, poisson.pmf(x, mu), 'bo', ms=8, label='poisson pmf')
ax.vlines(x, 0, poisson.pmf(x, mu), colors='b', lw=10, alpha=0.1)
```

```
1000.0 1000.0 0.03162277660168379 0.001
```

出力[41]:

<matplotlib.collections.LineCollection at 0x2bde4fcbe80>



入力 [42]:

```
from scipy.stats import chi2
fig, ax = plt.subplots(1, 1)
df=5
x = np.linspace(chi2.ppf(0.01, df), chi2.ppf(0.99, df), 100)
ax.plot(x, chi2.pdf(x, df), 'r-', lw=5, alpha=0.6, label='t pdf')
df1=100
rv = chi2(df1)
ax.plot(x, rv.pdf(x), 'k-', lw=2, label='frozen pdf')
```

出力[42]:

[<matplotlib.lines.Line2D at 0x2bde11e2190>]

