Машинное обучение в финансах

Лекция 5: Введение в программирование на Python часть 2. Pandas.

Роман В. Литвинов*

*CRO
Финансовая Группа БКС

Высшая школа экономики, Март 2021

Содержание

Содержание

Что такое Pandas. История создания пакета

Pandas представляет собой мощную билиотеку для анализа данных (загрузка, манипулирование, вычисления и анализ), в тч big data. Это стандарт де-факто в отрасли. Название Pandas происходит от словосочетания panel data.

Представьте себе вычислительную мощность NumPy, которую можно применить к табличным данным (близким по фомату к Excel/ реляционным базам данных/датафреймам R) и временным рядам.

Изначально пакет создавался Уэсом Маккини во время его работы в передовом количественном фонде AQR Capital. Поскольку изначально библиотека решала ежедневные задачи фонда, связанные с обработкой большого потока финансовых данных, в ней много полезных функций, связанных с такого рода вычислениями/ анализом.

NB: один из текущих спонсоров пакета - крупный количественный фонд Two Sigma.

Содержание

Dataframes, series

Двумя основными структурами данных Pandas являются DataFrame и Series.

Series представляет собой формат, заточенный под хранение временных рядов.

DataFrame ориентирован на двумерные табличные массивы данных, похожие таблицы Excel.

Для того, чтобы начать работать с этим пакетом, традиционно, нам необходимо его загрузить. Сделать это можно с использованием следующей команды:

```
import pandas as pd
```

NB: Дополнительная установка не требуется, поскольку пакет идет предустановленным в составе дистрибутива Anaconda.

Dataframes, series

Series представляет собой одномерный массив данных и ассоциированный с ним массив индексов.

При условии, что пакет pandas загружен создать такой массив можно с помощью команды:

```
A = pd.Series ([100,105,108,119,123,140])
```

Атрибуты values и index возвращают массив и информацию об индексах, соотвественно:

```
A.values
A.index
```

Доступ к соответствующему элементу можно получить указав его индекс (индексация по-прежнему начинается с нуля):

```
A[2]
A[2] = 10
```

Dataframes, series

Удобно, что к Series можно применять функции NumPy, фильтрацию с помощью логических условий и скалярные операции (умножение, деление на скаляр и тп)

```
A[A>120]
B=A*2
C=B/10
np.exp(C/10)
```

При создании объекта Series можно задать как массив данных, так и массив индексов (в этом смысле Series похож на словарь - связка ключ-значение):

```
D=pd.Series([1,2,4], index = ['Mar', 'Apr','May'])
D['May']
```

Dataframes, series

DataFrame - табличная структура данных. Как и таблица Excel состоит из набора столбцов. При этом в отличие от NumPy в разных столбцах могут содержаться разные типы данных (численные переменные, логические, строки).

Доступ к соотвествующей ячейке осуществляется с помощью индекса строки и столбца (элемент матрицы с i-ым/j-ым порядковым номером).

```
data=pd.DataFrame({'year':[2016,2017,2018,2019,
                           2020,2021],
                   'price':[75,90,100,90,85,120]})
```

Полезный (для больших фреймов) метод head() отображает первые пять строк таблицы. Метод info() возвращает описание вашего массива (количество столбцов, ненулевых значений , типы данных и тп):

```
data.head()
data.info()
```

Dataframes, series

Столбец фрейма можно извлечь по имени:

```
data['price']
```

Строки можно извлекать с помощью атрибута loc():

```
data.loc[0]
```

Можно добавлять новые столбцы. Длина должна совпадать с длиной фрейма (для Series недостающие данные будут автоматически заполнены NA - сокр. not available):

```
data['D/E'] =[4,3.5,3,3.5,4,2]
```

Удалять столбцы можно с помощью команды del:

```
data['dummy'] =1
del data['dummy']
```

Содержание

Импорт данных, обзор

Представить себе ситуацию, когда нам требуется в ручную создать массив с ежедневными курсами валют за несколько десятков лет можно только в страшном сне (хотя подобное иногда и случается).

В нормальном режиме ежедневно вы будете работать с данными импортированными из того или иного источника (файл, внутренняя база данных, дата-фид внешнего провайдера данных и тп).

Есть смысл выделить четыре основные канала получения данных:

- загрузка данных из файла (Excel, csv, json и тп). Pandas имеет встроенные функции для этого.
- загрузка из базы данных (sql)
- загрузка данных стороннего провайдера через API (Bloomberg, EOD, Quandl, СПАРК и тп).
- web scraping (twitter, html таблицы с сайтов и тп).

`https://pandas.pydata.org/pandas-docs/stable/reference/index.html`

Импорт данных, обзор

Сегодня мы подробно рассмотрим механику загрузки данных из Excel файла.

`https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.read_excel.html#pandas.read_excel`

В целом остальные опции импорта можно освоить самостоятельно, в процессе возникновения соответствующей потребности (вся необходимая информация добывается с помощью ссылки приведенной на предыдущем слайде, либо чтения документации API конкретного поставщика данных)

Для работы в классе я выгрузил с сайта ЦБ РФ информацию с динамикой курса руб/долл за последние 20 лет и разослал вам соотвествующий файл. Его надо разместить в рабочей директории (путь к ней выясняется с помощью полезной команды os.getcwd())

`https://www.cbr.ru/currency_base/dynamics/`

Импорт данных, обзор

Загрузка данных осуществляется с помощью функции Pandas read excel(). В качестве аргументов указываются адрес размещения файла на жестком диске (либо его название, если он находится в рабочей директории) и другие (опциональные) параметры:

```
file_name = 'RC_F27_03_2000_T25_03_2021.xlsx'
fx_data = pd.read_excel(file_name)
```

Полезными опциональными аргументами функции также являются:

```
fx_data = pd.read_excel(file_name,
                        sheet_name = "Sheet1",
                        index_col=0)
```

С полным списком аргументов вы можете ознакомиться по приведенной выше ссылке.

Импорт данных, обзор

После загрузки данных можно сделать краткий обзор характеристик массива с помощью команды:

```
fx_data.info()
```

На выходе мы получим сообщение следующего содержания:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5207 entries, 0 to 5206
Data columns (total 4 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   nominal  5207 non-null   int64
 1   data     5207 non-null   datetime64[ns]
 2   curs     5207 non-null   float64
 3   cdx      5207 non-null   object
dtypes: datetime64[ns](1), float64(1), int64(1), object(1)
memory usage: 162.8+ KB
```

Импорт данных, обзор

Следующее, что вы обычно делаете, это смотрите, как выглядит "шапка"массива. Для этого с помощью метода head() выводите на экран первые пять строк фрейма:

```
fx_data.head()
```

Выглядит это следующим образом:

	nominal	data	curs	cdx
0	1	2000-03-28	28.31	Доллар США
1	1	2000-03-29	28.29	Доллар США
2	1	2000-03-30	28.27	Доллар США
3	1	2000-03-31	28.46	Доллар США
4	1	2000-04-01	28.60	Доллар США

Импорт данных, обзор

Также иногда бывает полезным проверить размерность массива с помощью встроенного метода shape:

```
fx_data.shape
```

Чтобы сформировать окончательное представление о данных, с которым вам придется работать, можно расчитать набор статистических показателей (среднее, стандратное отклонение и тп):

```
fx_data.describe()
```

	nominal	curs
count	5207.0	5207.000000
mean	1.0	40.072187
std	0.0	16.538909
min	1.0	23.125500
25%	1.0	28.584800
50%	1.0	31.083400
75%	1.0	57.776150
max	1.0	83.591300

Содержание

Очистка данных и манипуляции с данными (склейка, комбинирование, изменение размерности)

После загрузки и знакомства с данными нам может потребоваться их изменить. Отдельная большая наука - это тестирование выбросов, заполнение пропусков и прочие вещи. Сегодня мы не будем подробно обсуждать эти аспекты. Сделаем это на лекции о feature engineering. Но, тем не менее, вещи первой необходимости мы затронем.

Первое, что может потребоваться, это переименование столбцов или их удаление. Это делается с помощью следующих команд:

```
fx_data.rename(columns={'curs': 'RUR/USD'})
del fx_data['cdx']
```

Еще одна полезная вещь - это удаление дублирующихся строк:

```
fx_data.drop_duplicates()
```

С помощью метода dropna() можно отбросить пустые/частично пустые строки:

```
fx_data.dropna()
```

Очистка данных и манипуляции с данными (склейка, комбинирование, изменение размерности)

Теперь представим ситуацию, когда у нас имеется два массива (например, с курсом евро и курсом доллара) и мы хотим слить их в один. Это можно сделать с помощью функции concat. В случае, если параметр axis равен 0 объединение идет по строкам, а если он равен единице, то по столбцам.

```
FX=pd.concat([fx_data,fx_data2],axis=1)
```

Другая полезная команда это возможность слияния нескольких массивов по определенному столбцу (содержащему общие для этих массивов данные). Пустые строки при несовпадении размерностей данных здесь отбрасываются автоматически.

```
FX_data=pd.merge(fx_data,fx_data2, on='data',
                 how='left')
```

Содержание

Вычисления и аналитика

В Pandas имеется большое количество встроеных функций для анализа данных содержащихся в фрейме. С помощью методов min() и max() можно вычислить минимум и максимум:

```
FX_data['RUR/USD'].min()
FX_data['RUR/USD'].max()
```

Есть встроенные методы для вычисления среднего, медианы, моды, асимметрии и эксцесса:

```
FX_data['RUR/USD'].mean()
FX_data['RUR/USD'].median()
FX_data['RUR/USD'].mode()
FX_data['RUR/USD'].skew()
FX_data['RUR/USD'].kurtosis()
```

Также возможно рассчитать дисперсию и стандартное отклонение:

```
FX_data['RUR/USD'].std()
FX_data['RUR/USD'].var()
```

Вычисления и аналитика

Прочие полезные функции это сумма значений, произведение, куммулятивная сумма и произведение (возвращает ряд, такой же размерности как исходный, содержащий куммулятивную сумму/произведение)

```
FX_data['RUR/USD'].sum()
FX_data['RUR/USD'].prod()
FX_data['RUR/USD'].cumsum()
FX_data['RUR/USD'].cumprod()
```

Еще, пожалуй, стоит отметить две следующие функции для расчета модуля значений и количества наблюдений:

```
FX_data['RUR/USD'].abs()
FX_data['RUR/USD'].count()
```

Вычисления и аналитика

Но как вам такое? Вы можете применить к данным содержащимся в массиве свою произвольную функцию или, например, функцию NumPy!

```
def func(x):
    y = x**2+10
    return y

FX_data['EUR_transform'] = func(FX_data['RUR/EUR'])

FX_data['EUR_transform_2'] =
                        np.exp(FX_data['RUR/EUR']/100)
```

Вычисления и аналитика

Отдельного восторга вызывают два следующих встроенных метода (помните я говорил вам о том, что Макини писал этот пакет, когда работал в AQR).

Первый это shift, позволяющий сдвигать данные для расчета относительных показателей (однодневные, месячные и прочие доходности, например)

```
FX_data['EUR_1d_rets'] =
np.log(FX_data['RUR/EUR']/FX_data['RUR/EUR'].shift(1))
```

Второй это rolling для вычисления скользящих показателей с заданным окном (скользящие средние, например). Ниже пример расчета 3-х месячной скользящей средней цены:

```
FX_data['EUR_3m_av'] =
FX_data['RUR/EUR'].rolling(window=64).mean()
```

Содержание

Визуализация

Все, что мы обсуждали на прошлой лекции в части пакета matplotlib, работает также хорошо и с данными Pandas. Но в Pandas есть свои встроенные эффективные средства визуализации датафреймов.

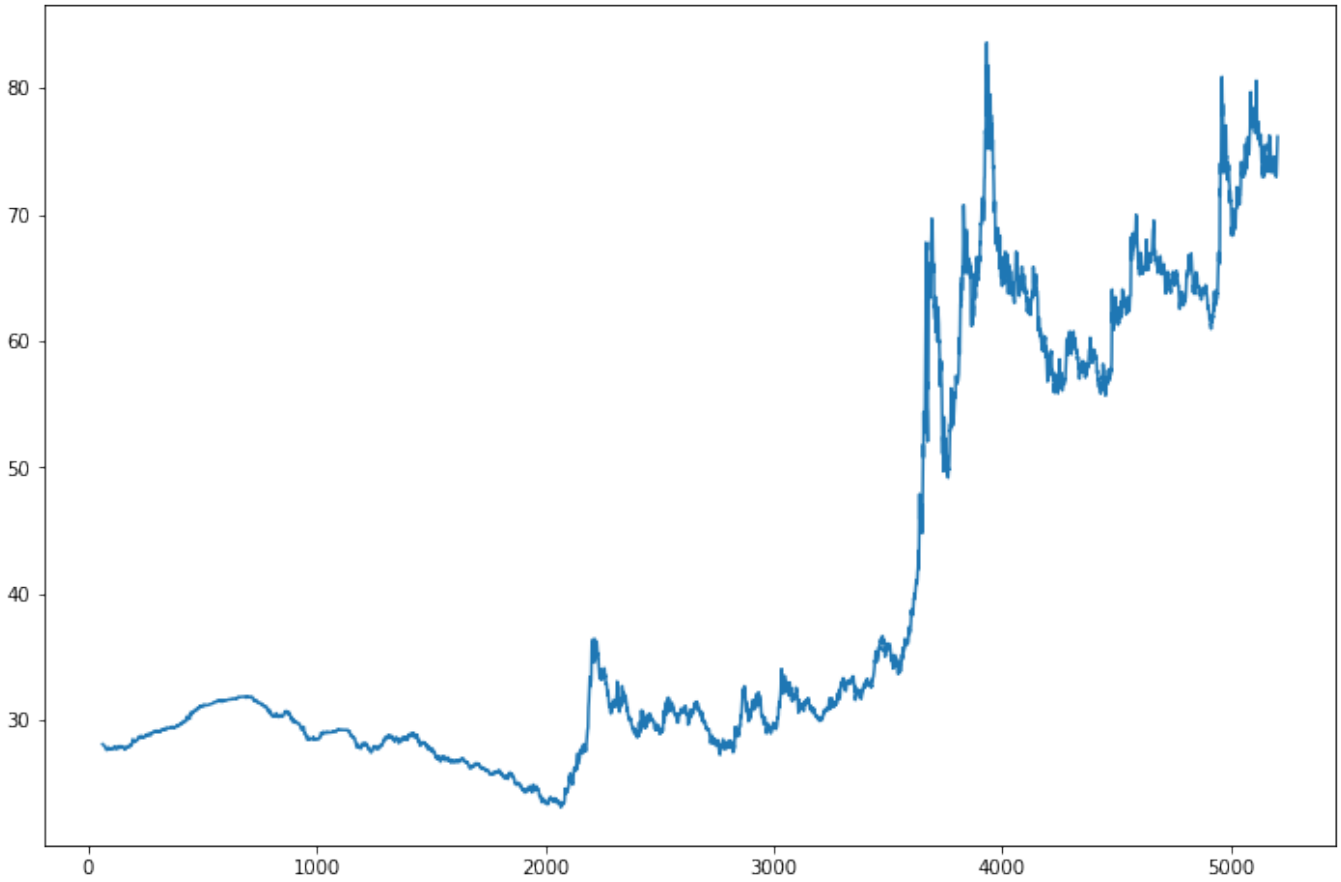
Эти методы можно использовать для быстрой визуальной инспекции временного ряда/рядов, с которыми вы работаете. Чтобы почувствовать динамику, отметить наличие экстремальных выбросов, волатильность и тп.

Обычно это делается на ранней стадии после зарузки данных и перед началом их очистки, устраниения выбросов и трансформации. Чем раньше вы увидите, то с чем собираетесь оперировать, тем быстрее сформируете своего рода интуитивное понимание данных. Что очень вам поможет.

Визуализация

Для быстрой визуализации временного ряда используется метод plot():

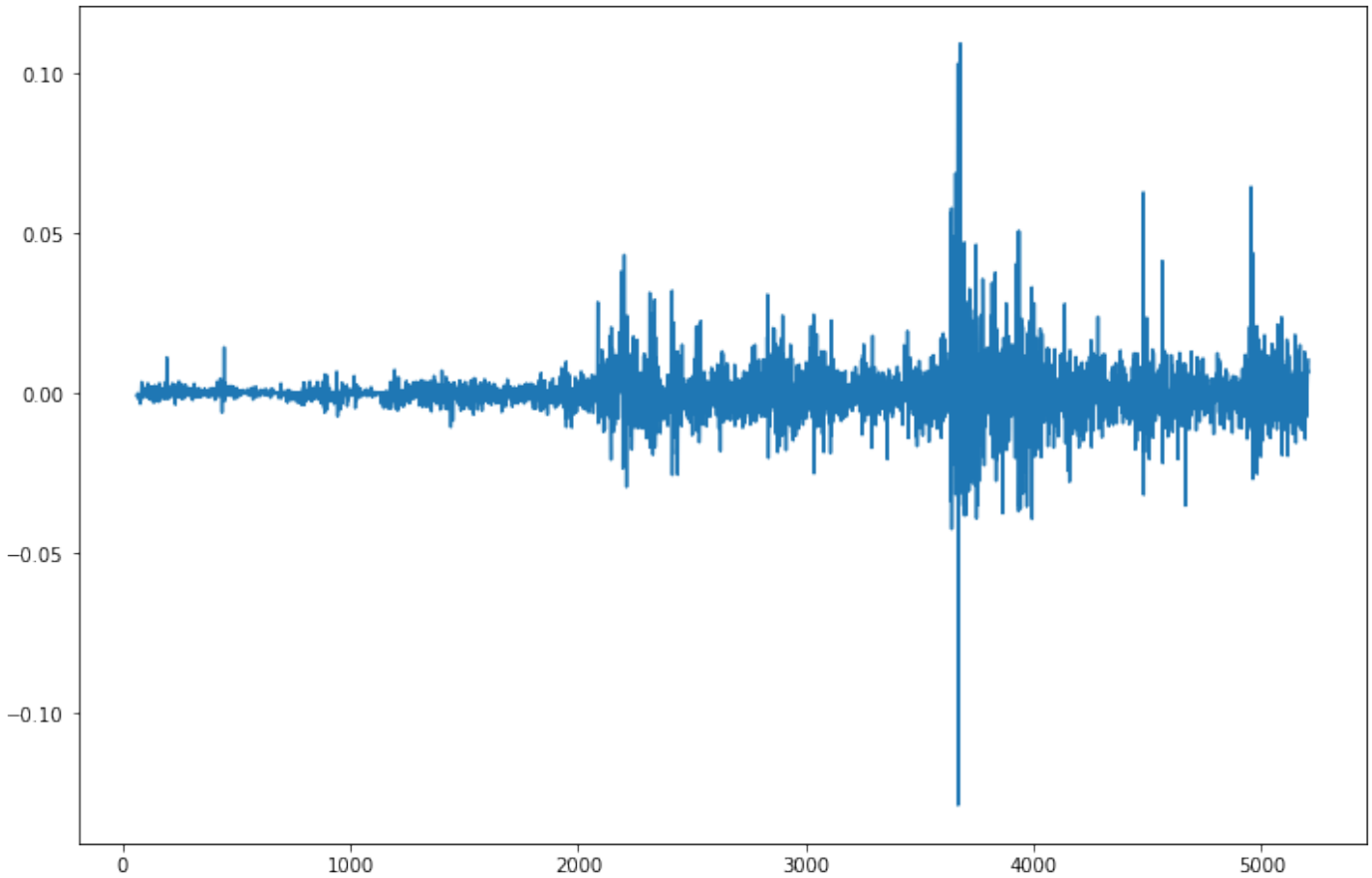
```
FX_data['RUR/USD'].plot(figsize=(12,8))
```



Визуализация

Следующий пример - построение графика однодневной доходности RUR/USD:

```
FX_data['USD_1d_rets'].plot(figsize=(12,8))
```



Визуализация

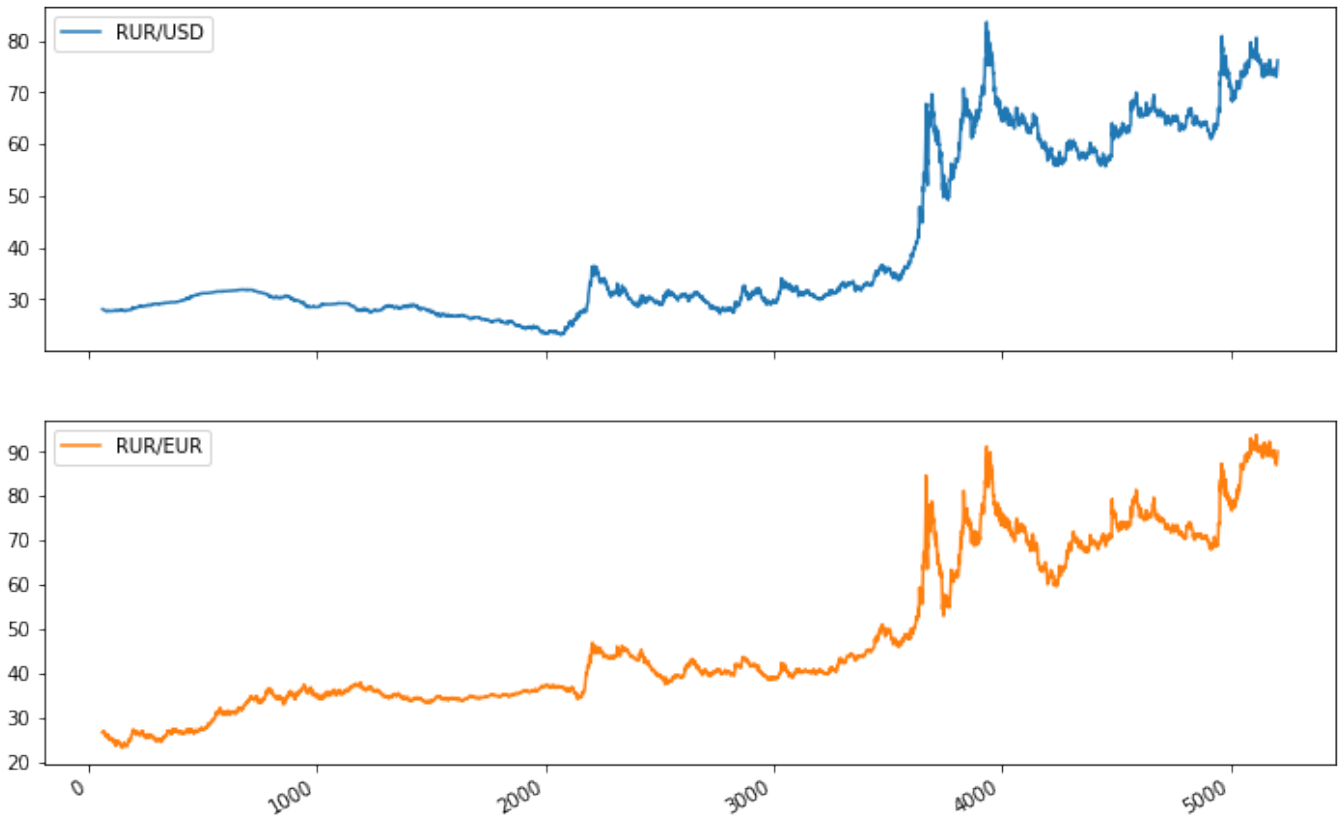
Вы также можете строить график для нескольких временных рядов:

```
courses=['RUR/USD','RUR/EUR']
FX_data[courses].plot(figsize=(12,8))
```

Визуализация

Иногда полезно использовать параметр subplots для отображения каждого ряда на отдельном графике:

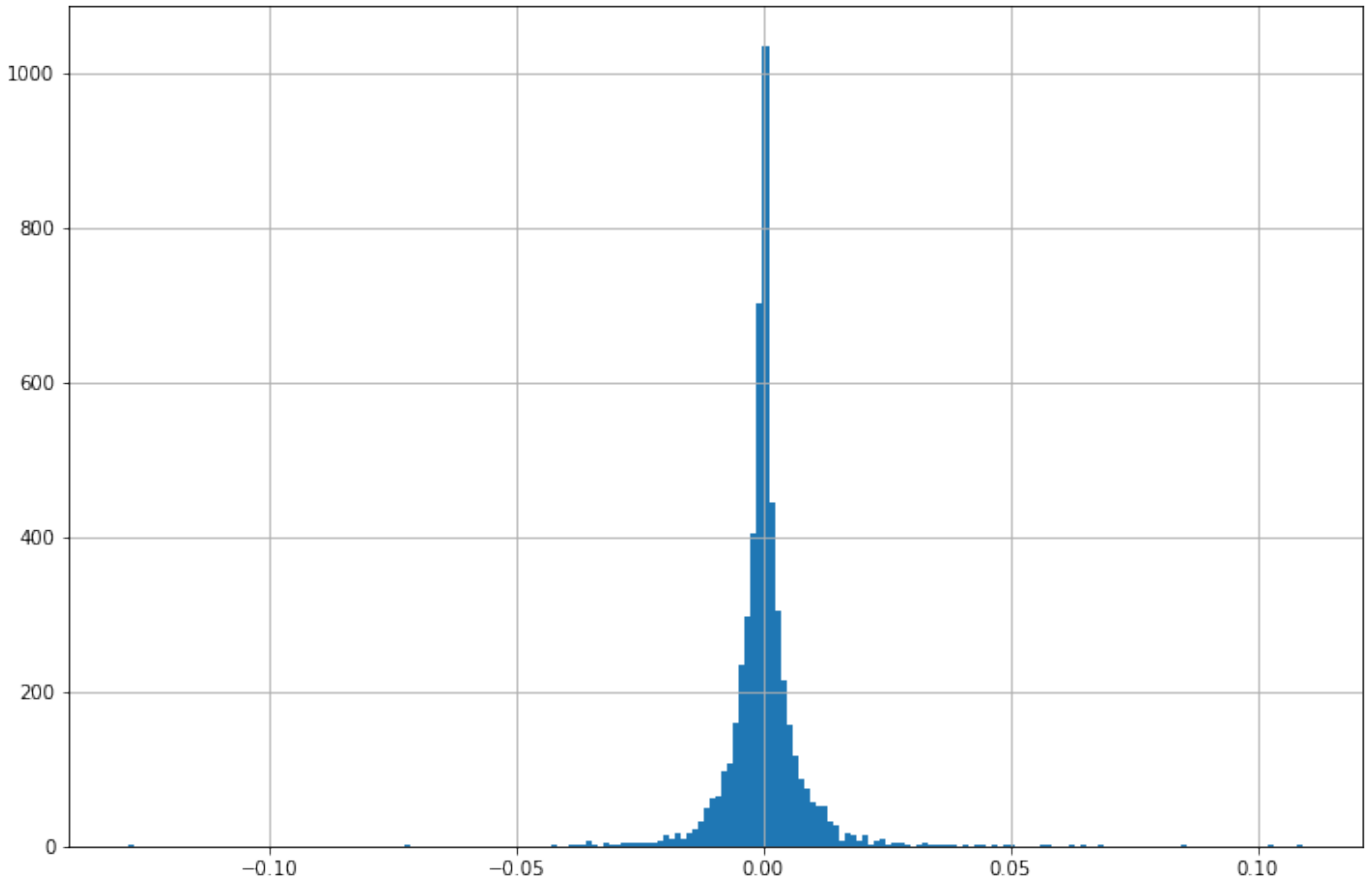
```
FX_data[courses].plot(figsize=(12,8), subplots=True)
```



Визуализация

Последний на сегодня часто используемый в работе базовый пример - построение гистограммы:

```
FX_data['USD_1d_rets'].hist(bins=200, figsize=(12,8))
```



Литература

Дополнительная литература к сегодняшней лекции:

- McKinney, W. (2018). Python for data science. O'Reily. (есть на русском)
- Pine, D. (2019) Introduction to Python for science and engineering. CRC Press.
- Albon C. (2018) Machine learning with Python cookbook. O'Reily.