Установка pandas

Разработка пакета начата в 2008 году сотрудником AQR Capital Management Уэсом МакКинни (Wes McKinney). Перед уходом из AQR ему удалось убедить руководство компании опубликовать исходный код библиотеки под свободной лицензией.

Pandas – программная библиотека на языке Python для обработки и анализа данных. Работа *pandas* с данными строится поверх библиотеки NumPy, которая является инструментом более низкого уровня. Предоставляет специальные структуры данных и операции для манипулирования числовыми таблицами и временн***ы***ми рядами.

Один из вариантов установки *pandas* – воспользоваться пакетным менеджером `pip`:

```
!pip install pandas
```

или

```
pip install pandas
```

Затем для работы с библиотекой pandas нужно ее импортировать командой:

```
import pandas as pd
```

А также, часто при анализе данных приходится пользоваться функциями или методами библиотеки NumPy, которая импортируется командой:

```
import numpy as np
```

Объект `Series`

В **pandas** существуют две основные структуры данных: ***Series*** и ***DataFrame***.

Сначала познакомимся с объектом `Series`.

`Series` – это одномерный объект, содержащий ***массив данных*** любого типа (числовые, строковые, булевы, даты и т.д.) и ассоциированный с ним ***массив меток***, называемый ***индексом***. Индекс генерируется автоматически (целочисленные значения: `0, 1, 2, 3` и т.д.), или же индивидуально задается пользователем (например: `'a', 'b', 'c', 'd'`).

Series

index	value
0	15
1	12
2	7
3	29

Слева отображаются значения индекса, а справа - сами значения (данные).

Пример объекта `Series`, в котором индекс сгенерирован автоматически:

```
0    84.1375
1    91.6525
2    11.7650
dtype: float64
```

Пример объекта `Series`, в котором индекс задан пользователем:

```
usd/rub    84.1375
eur/rub    91.6525
cny/rub    11.7650
dtype: float64
```

Во втором примере значениями индекса являются наименования валютных пар.

Создание объекта `Series`

Создаются объекты с помощью конструктора класса `Series` , который имеет следующий синтаксис:

```
pandas.Series(data=None, index=None, dtype=None, name=None, copy=False)
```

- `data` - структура, на базе которой будет создан объект `Series`;
- `index` - список меток, который будет использоваться для доступа к элементам объекта `Series`.
- `dtype` - тип данных;
- `name` - имя объекта `Series`;
- `copy` - если параметр принимает значение `True`, то будет создана копия массива данных и на ее базе сформирован объект `Series`. По умолчанию `False`.

Создать объект `Series` можно разными способами:

- из списка Python;
- из словаря;
- используя NumPy-массив;
- используя константу.

Для начала нужно импортировать библиотеку *pandas*. Если её импортировать таким образом:

```
import pandas
```

то при написании кода, в тех местах, где требуется указывать наименование `'pandas'`, - его придется указывать целиком. Т.е. при создании объекта `Series` нужно будет написать следующий код:

```
srs = pandas.Series()
```

В данном случае мы создали пустой объект `Series`.

Так как каждый раз прописывание в коде целиком слова `'pandas'` является лишней и ненужной тратой времени, то придумали использовать его сокращение - `'pd'`. И общепринятый импорт библиотеки *pandas* осуществляется следующим образом:

```
import pandas as pd
```

В этом случае создание пустого объекта `Series` будет выглядеть так:

```
srs = pd.Series()
```

Итак, с импортом разобрались. Теперь давайте приступим к рассмотрению вариантов создания объектов `Series`, и начнем с питоновских списков.

Создание объекта `Series` из списка

Создадим список для примера:

```
lst = [20, 30, 40]
print(lst)
```

Результат:

```
[20, 30, 40]
```

А теперь передадим его в конструктор `Series`:

```
srs = pd.Series(data=lst)
print(srs)
```

Результат:

```
0    20
1    30
2    40
dtype: int64
```

В итоге мы получили ожидаемый объект `Series`, где слева расположены индексы, справа идут значения, а подпись снизу: `dtype: int64` означает, что все значения в нашей серии - целочисленные.

Мы можем не создавать предварительно список, а передать его сразу в конструктор, и можем также не прописывать наименование `data`:

```
srs = pd.Series([20, 30, 40])
print(srs)
```

и получим аналогичный результат:

```
0    20
1    30
2    40
dtype: int64
```

Метод `info()`

Одним из частоиспользуемых и полезных методов, применительно к объектам `Series` и `DataFrame`, является метод **`info()`**, который позволяет получить краткую информацию об этих структурах данных, такую как: тип используемой индексации, количество и наименования столбцов (для `DataFrame`), наличие пропусков в данных, общее количество строк, типы используемых данных, объем занимаемой памяти объектом.

Давайте применим метод `info()` к нашей серии `srs`:

```
srs.info()
```

Результат:

```
<class 'pandas.core.series.Series'>
RangeIndex: 3 entries, 0 to 2
Series name: None
Non-Null Count  Dtype
--------------  -----
3 non-null      int64
dtypes: int64(1)
memory usage: 152.0 bytes
```

Здесь мы видим, что наш объект является структурой класса `Series`, содержит три записи, проиндексированные от 0 до 2. Объект не имеет названия (`Series name: None`). Из трех записей все три заполнены целочисленными значениями, и объем памяти, занимаемый этим объектом, составляет 152 байта.

Конечно в данном случае информация о количестве байтов нам не особо интересна. Но когда структуры данных содержат десятки тысяч, сотни тысяч или миллионы строк, то информация, о занимаемом объектом объеме памяти, начинает играть более существенную роль.

Параметр `index` конструктора класса `Series`

Используя параметр `index`, мы можем создать свою индексацию объекта `Series`. Давайте, воспользовавшись вышеприведенным примером, создадим буквенные индексы в серии, передав параметру `index` список с соответствующими значениями:

```
srs = pd.Series(lst, index=['A', 'B', 'C'])

print(srs)
```

Результат:

```
A    20
B    30
C    40
dtype: int64
```

Теперь посмотрим информацию о полученном объекте:

```
srs.info()
```

Результат:

```
<class 'pandas.core.series.Series'>
Index: 3 entries, A to C
Series name: None
Non-Null Count  Dtype
--------------  -----
3 non-null      int64
dtypes: int64(1)
memory usage: 48.0+ bytes
```

Как видим, объект `Series` создался с заданными нами значениями индекса.

Атрибут `index` объекта `Series`

Когда объект `Series` уже создан, он в своем арсенале имеет так называемые "атрибуты", которые прописываются следующим образом: сначала пишется имя объекта `Series`, далее ставится "точка", а затем прописывается наименование атрибута без каких-либо скобок.

Атрибуты используются для извлечения данных или любой информации, относящейся к конкретному объекту `Series`.

Атрибут `index` возвращает объект `pandas.Index`, содержащий метки индекса объекта `Series`.

Применим атрибут `index` к серии из предыдущего примера:

```
print(srs.index)
```

Результат:

```
Index(['A', 'B', 'C'], dtype='object')
```

Также колонке индекса можно присвоить наименование через другой атрибут, который называется `name`. Например,

назовем колонку с индексами - `'Индекс'`:

```
srs.index.name = 'Индекс'

print(srs)
```

Результат:

```
Индекс
A    20
B    30
C    40
dtype: int64
```

Атрибут `values` объекта `Series`

Атрибут `values` возвращает массив значений объекта `Series`.

Выведем значения серии `srs`:

```
display(srs.values)
```

Результат:

```
array([20, 30, 40])
```

Если, например, нужно вывести массив значений меток индекса, а не объект `pandas.Index`, то используется комбинация атрибутов `index` и `values`:

```
display(srs.index.values)
```

Результат:

```
array(['A', 'B', 'C'], dtype=object)
```