Создание объекта `Series` из словаря

Допустим у нас имеется словарь с ценами акций трех металлургических компаний:

```
shares = {'ММК': 52.50, 'Северсталь': 1386.20, 'НЛМК': 214.20}
```

Создадим объект `Series`, передав параметру `data` словарь `shares`:

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

print(srs)
```

Результат:

```
ММК             52.5
Северсталь    1386.2
НЛМК           214.2
dtype: float64
```

По итогу выполнения кода сформировался объект `Series`, в котором метками индекса являются ключи словаря, а значениями серии стали значения словаря.

А теперь давайте попробуем создать серию из того же словаря, но передать свои индексы:

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

print(srs)
```

Результат:

```
A   NaN
B   NaN
C   NaN
dtype: float64
```

В результате колонка индексов сформировалась согласно переданным значениям параметру `index`, а вот в столбце с данными мы видим записи `NaN`, означающие отсутствие какого-либо значения (`NaN` еще называют *пропусками*).

Логика произошедшей операции заключается в следующем: сначала сформировалась колонка с индексами, согласно значениям переданным параметру `index`. Затем значения индексов соотнеслись с ключами словаря, и если произошло совпадение ключа и метки индекса, то к данной метке индекса "привязывается" значение из словаря. Если среди ключей словаря не нашлось совпадения с меткой индекса, то данной метке индекса присваивается значение `NaN`.

Например, если передадим параметру `index` не все ключи словаря, то результат будет следующим:

```
shares = {'ММК': 52.50, 'Северсталь': 1386.20, 'НЛМК': 214.20}

srs = pd.Series(shares, index=['ММК', 'Северсталь'])

print(srs)
```

Результат:

```
ММК             52.5
Северсталь    1386.2
dtype: float64
```

Атрибут dtype

Еще один способ, помимо метода `info()`, узнать информацию о типе данных - применить к объекту `Series` атрибут **`dtype`**.

Например:

```
shares = {'ММК': 52.50, 'Северсталь': 1386.20, 'НЛМК': 214.20}
srs = pd.Series(shares)
print(srs)
```

```
ММК             52.5
Северсталь    1386.2
НЛМК           214.2
dtype: float64
```

```
print(srs.dtype)
```

Результат:

```
float64
```

Мы получили информацию, что серия `srs` содержит данные типа `'float64'`.

Теперь давайте разберемся, какие типы данных может содержать объект `Series` (или `DataFrame`).

Типы данных

Проверка типа данных - это одно из первых действий, которое аналитик должен произвести после загрузки новых данных в *pandas* для дальнейшего анализа.

Чтобы в процессе анализа данных не получать ошибки или неожиданные результаты, нужно использовать правильные типы данных. Напомню, что язык Python содержит такие типы данных, как: `str`, `int`, `float`, `bool`, `datetime`. Библиотека же *pandas* содержит свой набор типов данных для хранения и манипулирования ими.

Основные типы данных, используемые в *pandas*:

- `object` - текстовые или смешанные числовые и нечисловые значения;
- `int64` - целые числа;
- `float64` - числа с плавающей точкой;
- `bool` - булевое значение: `True`/`False`;
- `datetime64[ns]` - дата и время;
- `timedelta64[ns]` - разность двух datetime элементов;
- `category` - ограниченное множество текстовых элементов.

Как правило, при создании объектов `Series` и `DataFrame` *pandas* автоматически подбирает типы данных, но бывают случаи, когда пользователю необходимо самостоятельно указать тип данных или сделать преобразование одного типа в другой.

Стоит еще отметить, что все значения в объекте `Series` всегда одного типа данных. Не может быть такого, что одни данные в серии, например, имеют тип `'object'`, а другие - тип `'float64'`.

Параметр `dtype` конструктора класса `Series`

Вернемся к параметрам конструктора класса `Series` и освежим в памяти его синтаксис:

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

Параметр **`dtype`** конструктора класса `Series` позволяет пользователю задавать тип данных при создании серии.

Например, из словаря `shares = {'ММК': 52, 'Северсталь': 1386.20, 'НЛМК': 214.20}`, если не указывать тип данных при создании серии, сформируется объект `Series`, тип данных в котором будет `'float64'`.

Давайте из этого словаря создадим `Series` с типом данных `'object'`:

```
shares = {'ММК': 52, 'Северсталь': 1386.20, 'НЛМК': 214.20}

srs = pd.Series(shares, dtype='object')

print(srs)
```

Результат:

```
ММК              52
Северсталь    1386.2
НЛМК           214.2
dtype: object
```

Как видим, создался объект с типом данных `'object'`.

Функция type()

Функция **`type()`** позволяет получить информацию не о типе данных внутри объекта, а узнать к какому классу принадлежит

сам объект.

```
print(type(srs))
```

Результат:

```
<class 'pandas.core.series.Series'>
```

Объект `srs` принадлежит к классу `Series`.

Метод astype()

В объектах `Series` возможно производить изменения типов данных, что позволяет сделать метод **`astype()`**. Этому методу передается тот тип данных, в который нужно преобразовать значения структуры `Series`.

В предыдущем примере мы серии `srs` присвоили тип данных `'object'`. Давайте поменяем его на тип данных `'float64'`:

```
srs.astype('float64')
```

```
ММК             52.0
Северсталь    1386.2
НЛМК           214.2
dtype: float64
```

Видим, что вроде как тип данных поменялся на вещественный. Но в данном случае произошло создание нового объекта `Series`, а в самой серии `srs` тип данных остался прежним:

```
print(srs)
```

```
ММК               52
Северсталь    1386.2
НЛМК           214.2
dtype: object
```

Но если нужно, чтобы тип данных поменялся и в исходной серии, используйте оператор присваивания `'='`:

```
srs = srs.astype('float64')
print(srs)
```

Результат:

```
ММК             52.0
Северсталь    1386.2
НЛМК           214.2
dtype: float64
```