KARPOV.COURSES >>>
КОНСПЕКТ

> Конспект > 2 урок > Первые аналитические задачи

> Оглавление

> Нюансы считывания csv

```
pd.read_csv('path_to_your.csv')  # read_excel for reading exc
el files
```

Считывает csv файл, который лежит по указанному в скобках пути. На Windows пути к файлам содержат символ `\`, который является специальным символом в строках во многих языках программирования, включая Питон. Поэтому необходимо сделать следующее — либо удвоить все символы `\` в строке, содержащей путь, либо поставить `r` перед строкой:

- путь на windows — `C:\user\docs\Letter.txt`
- строка с путём — `'C:\user\docs\Letter.txt'`
- валидная строка с путём путь на windows — `'C:\\user\\docs\\Letter.txt'` или `r'C:\user\docs\Letter.txt'`

На сервере мы работаем с Unix путями, например, `/home/user/letter.txt`. С ними меньше таких проблем — достаточно поместить путь в кавычки, чтобы всё было хорошо.

Дополнительные аргументы функции read_csv

Аргументы (или параметры) — это настройки, которые мы можем задать для функции.

- **encoding** — параметр в `read_csv`, отвечает за кодировку текста, которая может быть различной. Самая распространённая — utf-8. Пример указания кодировки:

```
pd.read_csv('path_to_your.csv', encoding='Windows-1251')  # n
ow you are reading file encoded with Windows-1251
```

- **sep** — разделитель между ячейками в строке (по умолчанию `,`)

```
pd.read_csv('path_to_your.csv', encoding='Windows-1251', sep
=';')  # now you additionally specified that fields are separ
ated with ;
```

- **parse_dates** — указывает, стоит ли воспринимать даты как даты (по умолчанию они воспринимаются пандасом как строки). Параметр с датами может принимать несколько значений:
  - `True` — пытается перевести в дату первую колонку
  - список колонок — пытается перевести в дату указанные в списке колонки

```
# And create_data, payment_data columns will be treated as da
ta
```

```
pd.read_csv('path_to_your.csv', encoding='Windows-1251', sep
=';', parse_dates=['create_data', 'payment_data'])
```

Документация

> Цепочка методов (method chaining)

Приём для объединения нескольких действий в одно. Большинство методов датафрэймов возвращают вам результат, который довольно часто тоже является датафрэймом. Следовательно, от него тоже можно вызвать метод.

Ванильная запись:

```
df = df.query('income >=  1000')
df = df.groupby(['title', 'status'], as_index=False).agg({'in
come': 'sum', 'id': 'count'})  # groupby is usually immediate
ly followed by agg
df = df.sort_values(['title', 'status'])
```

Сокращённая запись:

```
df  = df.query('income >=  1000').groupby(['title', 'statu
s'], as_index=False).agg({'income': 'sum', 'id': 'count'}).so
rt_values(['title', 'status'])
```

Как можно заметить, эта запись довольно длинная и не очень удобная для чтения. Обычно такая цепочка оформляется в блок, где каждый метод идёт на своей строке. Есть 2 варианта оформления, какой выбрать — вопрос предпочтения и конвенций в вашей организации:

```
# \ after each nonfinal line to demarcate line continuation f
or python
df  = df.query('income >=  1000') \
  .groupby(['title', 'status'], as_index=False) \
```

```
   .agg({'income': 'sum', 'id': 'count'}) \
   .sort_values(['title', 'status'])
```

```
# Parentheses around the whole expression for the same purpos
e as backslash in previous example

df  = (df.query('income >=  1000')
         .groupby(['title', 'status'], as_index=False)
         .agg({'income': 'sum', 'id': 'count'})
         .sort_values(['title', 'status']))
```

Больше информации

> Несколько способов сделать агрегацию

1. Для агрегации часто применяется метод `agg`, обычно его применяют после группировки методом `groupby`.

Существуют разные способы передать в `agg` информацию о том, что и как вы хотите агрегировать. Самый простой и полный — использовать словарь, в котором ключами являются названия колонок, а значениями — применяемые к ним функции:

```
product_data.groupby('title').agg({'money': 'sum'})
```

title	money
Курс обучения «Консультант»	208163.49
Курс обучения «Специалист»	160862.64
Курс обучения «Эксперт»	148992.80
Курс от Школы Диетологов. Бизнес	18752.54
Курс от Школы Диетологов. Повышение квалификации.	88384.92
Подписка «ОНЛАЙН ДИЕТОЛОГ» с ежемесячным автосписанием	366947.20

Чтобы применить несколько функций, используйте список функций. Можно передать как сами функции (`sum`), так и обозначающие их строки (`'sum'`).

В результате агрегации из массива значений (колонка) получается одно значение на каждую агрегирующую функцию.

Полное название метода `agg` — `aggregate`, можно использовать любое из них, они работают одинаково. Сами разработчики pandas рекомендуют использовать короткое название.

Документация

2. Также можно применить метод `agg` к колонке, а в метод подать не словарь, а агрегирующую функцию — встроенную в python (например, sum), созданную в pandas (например, pandas.Series.mode) или написанную самостоятельно::

```
def my_func(x):
    return x.sum()
```

```
df.groupby('tittle').money.agg(my_func)
```

```
tittle
Курс обучения «Консультант»                                208163.49
Курс обучения «Специалист»                                 160862.64
Курс обучения «Эксперт»                                    148992.80
Курс от Школы Диетологов. Бизнес                            18752.54
Курс от Школы Диетологов. Повышение квалификации.           88384.92
Подписка «ОНЛАЙН ДИЕТОЛОГ» с ежемесячным автосписанием     366947.20
Name: money, dtype: float64
```

Но в этом случае результат будет являться Series, а в примере выше с `agg` - датафреймом.

3. Аналогичный результат можно получить и без метода `agg`, обращаясь к колонке после группировки и применяя к этой колонке агрегирующую функцию:

```
df.groupby('tittle').money.sum()
```

```
tittle
Курс обучения «Консультант»                                  208163.49
Курс обучения «Специалист»                                   160862.64
Курс обучения «Эксперт»                                      148992.80
Курс от Школы Диетологов. Бизнес                              18752.54
Курс от Школы Диетологов. Повышение квалификации.             88384.92
Подписка «ОНЛАЙН ДИЕТОЛОГ» с ежемесячным автосписанием       366947.20
Name: money, dtype: float64
```

Здесь снова результат будет являться Series

> Фильтрация с помощью запросов

В пандасе есть возможность фильтровать данные, используя метод `query`, принимающий строку с запросом. Внутри него можно использовать названия колонок (если они без пробелов). При использовании строк внутри запроса экранируйте кавычки `\` или используйте другую пару.

```
product_data.query("status == 'Завершен'")
```

	id	create_data	payment_date	title	status
0	1062823	01.12.2019 10:50	01.12.2019 10:52	Курс обучения «Эксперт»	Завершен
1	1062855	01.12.2019 20:53	01.12.2019 21:27	Курс обучения «Эксперт»	Завершен
12	1062938	05.12.2019 12:07	22.12.2019 12:29	Курс обучения «Консультант»	Завершен

В `query` также можно передать сразу несколько условий. Условия, которые должны выполняться одновременно, соединяются с помощью `and` или `&`:

```
product_data.query("title == 'Курс обучения «Эксперт»' and st
atus == 'Завершен'")
```

Когда должно удовлетворяться одно из условий – `or` или `|`:

```
product_data.query("title == 'Курс обучения «Эксперт»' or sta
tus == 'Завершен'")
```

Оба условия нужно писать в единых кавычках, а каждое условие и названия колонок - без кавычек. Целые числа и булевы значения (True, False) также пишутся без кавычек, а строки - в кавычках (двойных или одинарных - главное, не тех, в которые взяты оба условия).

Один знак равно означает присвоение, два - проверку на равенство, а восклицательный знак и знак равно - проверку на неравенство.

Чтобы сравнить значения в колонке со значением в какой-либо переменной, поставьте перед названием переменной `@`

```
course = 'Курс обучения «Эксперт»'
product_data.query("title == @course & status != 'Завершен'")
```

Документация

Метод `query` работает по аналогии с `loc`, их синтаксис отличается, но результаты будут одинаковыми:

```
df.loc[df.status == 'Завершен']
```

	number	create_date	payment_date	tittle	status
0	1062823	01.12.2019 10:50	01.12.2019 10:52	Курс обучения «Эксперт»	Завершен
1	1062855	01.12.2019 20:53	01.12.2019 21:27	Курс обучения «Эксперт»	Завершен
12	1062938	05.12.2019 12:07	22.12.2019 12:29	Курс обучения «Консультант»	Завершен

Видео: Анатолий Карпов демонстрирует подходы к фильтрации данных в pandas

> Новые функции и строковые методы

- `replace` – применяется к строкам, принимает 2 строки: что заменить и на что

Больше информации

```
my_string = 'Крайне важное название'
my_string.replace(' ', '_')
```

```
'Крайне_важное_название'
```

Аналогичный метод в pandas

- `strip` – применяется к строкам, по умолчанию убирает пробелы слева и справа

Больше информации

```
user_name = '     Vasya Vedrov '
user_name.strip()
```

```
'Vasya Vedrov'
```

Аналогичный метод в pandas

- `lower` – применяется к строкам, переводит все символы в нижний регистр

```
'Hello, World!'.lower()
```

```
'hello, world!'
```

Аналогичный метод в pandas

- `startswith` — строковый метод, принимающий другую строку и возвращающий `True` или `False` в зависимости от того, начинается ли

исходная строка с переданной

```
'Abyss'.startswith('Ab')
True
```

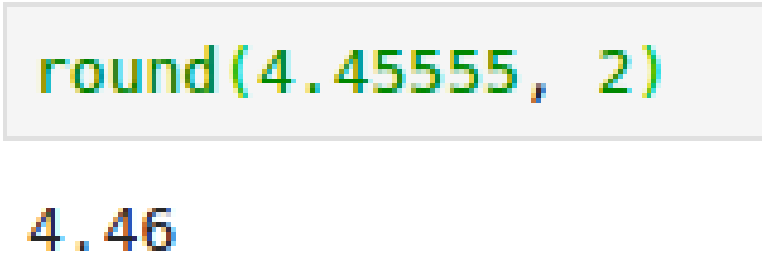
```
'Abyss'.startswith('ab')
False
```

Больше информации

Аналогичный метод в pandas

- `round` – применяется к дробным числам, округляет их. Можно передать дополнительный аргумент, который означает число знаков после запятой. Если этот аргумент не передан, округление идет до целого числа

Больше информации



Встроенная в python функция round работает схожим образом с пандасовским методом round, но они имеют разный синтаксис: в функцию дробное значение подают, а метод применяется к дробному значению в Series

Функция python:

```
round(12.345, 1)
```

```
12.3
```

Метод pandas:

```
s = pd.Series(12.345) #создадим серию из одного значения
s.round(1)
```

```
0    12.3
dtype: float64
```

> Векторизация

Векторизация — это специальная техника, позволяющая в пандасе быстро выполнять в одну строчку операции, которые в чистом питоне требуют как минимум одного цикла. Быстрота связана с тем, что код пандаса реализован на более быстром чем Питон языке, а в Питоне просто представлены функции. Благодаря векторизации мы можем делать различные операции со всеми колонками целиком, не отвлекаясь на итерирование по элементам.

```
df.driver_score
```

```
0       5.0
1       4.0
2       5.0
3       5.0
4       0.0
       ...
7426    0.0
7427    5.0
7428    5.0
7429    5.0
7430    0.0
Name: driver_score, Length: 7431, dtype: float64
```

Умножим каждый элемент на 3

```
df.driver_score * 3
```

```
0       15.0
1       12.0
2       15.0
3       15.0
4        0.0
        ...
7426     0.0
7427    15.0
7428    15.0
7429    15.0
7430     0.0
Name: driver_score, Length: 7431, dtype: float64
```

Или выясним для каждого значения, больше ли оно, чем 3

```
df.driver_score > 3
```

```
0        True
1        True
2        True
3        True
4       False
        ...
7426    False
7427     True
7428     True
7429     True
7430    False
Name: driver_score, Length: 7431, dtype: bool
```

Такой синтаксис не удобен для использования цепочки методов, поэтому в pandas есть метод `mul`, с помощью которого можно умножить каждое значение на определенное число - и продолжить цепочку методов дальше

```
money_title.money.mul(10).round()
```

```
5    3669472.0
0    2081635.0
1    1608626.0
2    1489928.0
4     883849.0
3     187525.0
Name: money, dtype: float64
```

Документация

None

`None` – это специальный тип данных в Питоне, который имеет только одно одноимённое значение – `None`. Используется в тех случаях, когда нужно обозначить ничто. Обычно (но совсем необязательно) его возвращают функции, которые как-то изменяют данные.

```
xs = [1, 2, 3]
a = xs.append(4)

print(xs)
[1, 2, 3, 4]

print(a)
None
```

Толку от присвоения выше (`a = xs.append(4)`) нет, просто постепенно запоминайте такие функции, и не перезаписывайте ваши данные вызовами типа

```
xs = xs.append(4)
```

Документация

> Время

Для работы с датой и временем можно использовать модуль `datetime`. Для получения данных о времени в момент вызова функции используйте функцию `today` в одноимённом подмодуле:

```
import datetime

date = datetime.datetime.today()
```

Само по себе это даст вам специальный тип даты. Чтобы перевести его в строку, сделайте следующее:

```
datetime.datetime.today().strftime('%Y-%m-%d-%H:%M:%S')
'2020-01-30-00:07:12'
```

`strftime` форматирует дату по переданному ему формату:

- % – обозначает что дальше будет часть даты
- Y – год 4-мя знаками
- m – месяц 2-мя знаками
- d – день
- H – час
- M – минуты
- S – секунды

Можно использовать только часть фрагментов даты, разделители между ними – на ваше усмотрение (в примере это `-` и `:`). Немного примеров:

```
from datetime import datetime


# current date and time
now = datetime.now()
```

```
print(f'Full time format of now is {now}')
Full time format of now is 2020-06-01 17:54:40.010540
```

```
# Year
year = now.strftime("%Y")
print("year:", year)
year: 2020
```

```
# Month
month = now.strftime("%m")
print("month:", month)
month: 06
```

```
# Day
day = now.strftime("%d")
print("day:", day)
day: 01
```

```
# Time
time = now.strftime("%H:%M:%S")
print("time:", time)
time: 17:54:40
```

```
# Date and time
date_time = now.strftime("%m/%d/%Y, %H:%M:%S")
print("date and time:",date_time)
date and time: 06/04/2020, 17:54:40
```

Документация

Погрешность арифметики

В компьютере используется двоичная система счисления, в которой не выразить точно любое десятичное число. Из-за этого при выполнении действий может накапливаться ошибка. Обычно это не страшно (она в порядках меньше -5-ого), но бывает нужна точность. Для этого есть специальные библиотеки.

Документация