KARPOV.COURSES >>>
КОНСПЕКТ

> Конспект > 4 урок > PYTHON

> Оглавление 4 урока

14. Соединение датафреймов

> Проверка на начало строки

`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

> Альтернативный способ создания списка

List comprehension — способ, который часто используется как лаконичная (и убыстренная) замена циклов, где заполняется список:

```
xs = [i + 3 for i in range(10)]
```

Аналогично:

```
xs = []
for i in range(10):
    xs.append(i + 3)
```

В таком способе можно прописать условия и даже вложенные циклы. Однако не стоит сильно их нагружать: читаемость кода — одно из его самых важных качеств, а вложенные конструкции делают код сложнее для понимания.

```
# Get all even numbers
evens = [i for i in range(10) if i % 2 == 0]
```

```
# Analogous to
even = []
for i in range(10):
    if i % 2 == 0:
        even.append(i)
```

Больше информации

> Конвертация типов в датафрэйме

Нередка ситуация, когда тип данных в колонке не соответствует желаемому. Почему это вообще важно? Для разных типов определены разные операции — `'0.15' * 100` не переведёт дробь в проценты.

Чтобы это исправить, есть метод `astype`, в который можно передать словарь, где ключи — названия колонок, а значения — новые типы для них. Метод возвращает новый датафрэйм с изменёнными типами:

```
df = df.astype({'money': 'float'})  # df.money will be rational number after this line
```

Для конвертации типов колонок есть более простой вариант — передайте желаемый тип при вызове `astype` от колонки:

```
df.height = df.height.astype('float')  # df.height will be rational number
```

Документация

> Удаление колонок

Чтобы убрать часть колонок из датафрэйма, воспользуйтесь методом `drop`, куда можно передать список из названий, которые нужно убрать. Метод также позволяет убирать строки по индексу, для указания измерения, в котором мы работаем, используется
аргумент `axis` (`0` — строки, `1` — колонки). Лучше использовать более понятные `columns` / `index`. Возвращается новый датафрэйм:

```
df = df.drop(columns='Date')  # drop Date column
```

```
df = df.drop(index=350)  # drop row with 350 index
```

Документация

> Фильтрация дубликатов

Дубликаты — повторяющиеся наблюдения, которых не должно быть. Быстро их убрать позволяет метод `drop_duplicates`, возвращающий таблицу без них:

```
df = df.drop_duplicates()  # df will contain <= rows than before after this operation
```

Вывести все дубликаты:

```
df.loc[df.duplicated()]
```

Дополнительные аргументы

`subset` — принимает список колонок, по которым нужно смотреть дупликацию.

```
# Drop only if duplicates are in 'Date' or 'Last' columns
df.drop_duplicates(subset=['Date', 'Last'])
```

Документация

> Сравнение строк

Строки сравниваются в лексикографическом порядке (по алфавиту, как в языковых словарях)

```
'1' < '2'
True
```

```
'abc' < 'b'
True
```

Больше информации

> Соединение сравнений (comparison chaining)

Эти 2 записи тождественны:

```
1 < 2 and 2 < 3
True
```

```
1 < 2 < 3

True
```

Документация

> Конвертация во время

`pd.to_datetime()` — метод, позволяющий превратить строки во время. Это позволяет удобно с ним работать.

```
meal_data['Date'] = pd.to_datetime(meal_data['Date'])
```

Атрибут `dt` позволяет извлекать временные характеристики из колонки с датой:

```
meal_data['Date'].dt.year.head()

0    2019
1    2019
2    2019
```

```
df['Date'].dt.month.head()
```

```
0     9
1     9
2     9
3     9
5    10
```

Документация

Парсинг дат

Также вы можете заранее распарсить дату при загрузке датасета, передав в `parse_dates` список колонок, в которых содержится дата:

```
pd.read_csv('some.csv', parse_dates=[1])  # order of columns starting from 0
```

Документация

> Открывание файлов

Рассмотренный способ открывания файлов с помощью `pd.read_csv` не единственный, и не первый в питоне. Традиционно любой файл открывается с помощью функции `open`, принимающий путь к файлу. Это менее удобно, так как это базовый способ, который далее усложняется в том же `pd.read_csv`

```
file = open('path_to_file')
```

У `file` есть различные методы на чтение содержимого, например `readlines:`

```
lines = file.readlines()  # lines is a list with lines from the file
```

В конце работы с файлом — то есть, когда он вам больше не понадобится — его нужно закрыть, для чего используется метод `close`

```
file.close()
```

Существует более удобный и предпочтительный способ с контекстным менеджером.

Больше информации

> Просмотр содержимого папок

Просмотр папок и многие другие операции, связанные с файлами и папками, выполняются с помощью библиотеки `os`. Для получения списка файлов используется `listdir`. Метод `os.listdir` принимает путь к папке и возвращает её содержимое в виде списка:

```
import os
```

```
os.listdir('/etc')
```

```
['console-setup',
 'sound',
 'hosts.allow',
 'gimp',
```

Названия файлов в папке вместе с путём к ней, позволяют реконструировать полный путь и работать с этими файлами:

```
path = '/etc'
path_to_file = path + '/' + os.listdir(path)[0]  # os.path.join is better for construc
ting path
```

При вложенности папок и необходимости добраться до дна можно использовать `os.walk`

```
for path, dirs, files in os.walk('Res_Tree'):
    print(path, dirs, files)
```

```
Res_Tree ['M000547', 'F000570', 'F000545'] []
Res_Tree/M000547 ['res_2019.09.13_0.0_6655DA_Container-dat_3057_144-149-294_M000547'] []
```

На каждой итерации (первый этап цикла) метод возвращает тройку из пути к нынешней папке, списков папок и файлов, хранящихся в этой папке.

Также есть более предпочтительный вариант — модуль pathlib.

Больше информации

> Прерывание цикла

При необходимости можно выйти из цикла с помощью слова `break`. Это бывает нужно сделать при выполнении определённых условий.

Искусственный пример — на итерации, где в `i` попадёт число **больше 5**, цикл будет прерван, и будет выполняться код ниже него. Таким образом будут напечатаны все числа до того, которое больше 5 (в данном случае **7**).

```
numbers = [1, 3, 2, 4, 5, 7, 10]

for i in numbers
    if i > 5:
        break
    print(i)
```

```
13245
```

Пропуск итераций цикла

Другой частый случай — пропуск каких-то итераций, для этого используется слово `continue`

Здесь будут напечатаны только чётные числа:

```
numbers = [1, 3, 2, 4, 5, 7, 10]

for i in numbers:
    if i % 2 != 0:
        continue
    print(i)
```

```
2410
```

Справедливости ради, простой код может быть написан без применения `continue`

```
for i in :
    if i % 2 == 0:
        print(i)
```

А он понадобится для более сложных случаев.

Больше информации

> Удаление пропущенных значений

`dropna` — метод, позволяющий выкинуть из датафрэйма все строки, содержащие пропущенные значения.

```
df.head()
```

	Id	sum
0	1	150.0
1	2	230.0
2	3	NaN
3	4	143.0
4	5	NaN

Так мы выкинем все строки, где было хотя бы одно пропущенное значение:

```
df.dropna()
```

	Id	sum
0	1	150.0
1	2	230.0
3	4	143.0
5	6	223.0
8	9	143.0

У `dropna` есть набор интересных параметров, с которыми можно ознакомиться в документации:

Документация

> Проверка на вхождение

Чтобы узнать, есть ли элемент в списке, используется оператор `in`

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

```
True
```

Больше информации

В *pandas* есть более эффективный метод `isin`, принимающий коллекцию, в которой содержатся искомые значения.

Документация

Логическое индексирование

```
wanted_clients = [5681, 111793]
ads_data.loc[ads_data.client_union_id.isin(wanted_clients)]
```

	ad_id	time	event	date	ad_cost_type	has_video	client_union_id	campaign_union_id	platform	ad_cost
0	23456	1554076848	view	2019-04-01	CPM	0	5681	23456	android	190.0
1	111941	1554135866	view	2019-04-01	CPM	0	111793	111941	ios	215.2
2	111941	1554135866	view	2019-04-01	CPM	0	111793	111941	ios	215.2
4	23456	1554135866	view	2019-04-01	CPM	0	5681	23456	web	190.0

Запросы

Согласно документации, работает быстрее

```
wanted_clients = [5681, 111793]
ads_data.query('client_union_id == @wanted_clients')
```

	ad_id	time	event	date	ad_cost_type	has_video	client_union_id	campaign_union_id	platform	ad_cost
0	23456	1554076848	view	2019-04-01	CPM	0	5681	23456	android	190.0
1	111941	1554135866	view	2019-04-01	CPM	0	111793	111941	ios	215.2
2	111941	1554135866	view	2019-04-01	CPM	0	111793	111941	ios	215.2
4	23456	1554135866	view	2019-04-01	CPM	0	5681	23456	web	190.0

Использование `==` как аналог `isin` является старым и не работает со списками. Вместо него лучше использовать `in`:

```
wanted_clients = [5681, 111793]
ads_data.query('client_union_id in @wanted_clients')
```

	ad_id	time	event	date	ad_cost_type	has_video	client_union_id	campaign_union_id	platform	ad_cost	target_audience_count	us
0	23456	1554076848	view	2019-04-01	CPM	0	5681	23456	android	190.0	125560	
1	111941	1554135866	view	2019-04-01	CPM	0	111793	111941	ios	215.2	32277	
2	111941	1554135866	view	2019-04-01	CPM	0	111793	111941	ios	215.2	32277	
4	23456	1554135866	view	2019-04-01	CPM	0	5681	23456	web	190.0	125560	

Документация

> Соединение датафрэймов

Для объединения двух или более датафрэймов существует метод pd.concat(). Он принимает кортеж из нескольких датафрэймов и возвращает один большой из них. Соединять можно вертикально (увеличиваем число строк) или горизонтально (увеличиваем число столбцов).

Документация

Добавление строк

Есть менее общий вариант — метод append() принимает другой датафрэйм и прибавляет его вертикально.

Для записывания в датафрэйм новых строк из другого датафрэйма можно использовать метод append, возвращающий новый датафрэйм, где прибавлены строки из второго.

```
both_df = first_df.append(second_df)
```

Документация