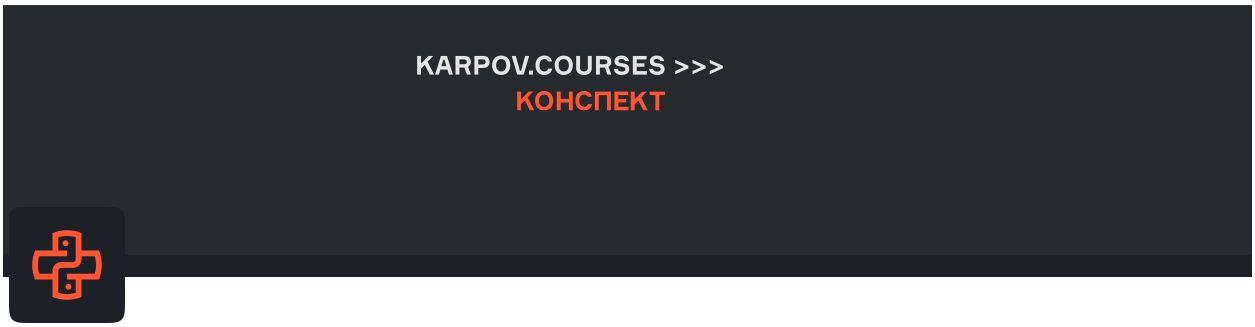


> Конспект > 5 урок > Решение продуктовых задач

Оглавление

> Чтение сжатых файлов

У функции `pd.read_csv` есть аргумент `compression`, который принимает строчку типа компрессии и открывает заархивированный файл:

```
# здесь тип компрессии - zip
ads_data = pd.read_csv('ads_data.zip', compression='zip')
```

Такой способ сработает только в том случае, если в архиве один файл csv.

Больше информации

> quantile

Метод для поиска определённых перцентилей (см. также квантиль).
Принимает число от 0 до 1, обозначающее перцентиль в виде доли:

- 0 — нулевой перцентиль
- 0.1 — десятый перцентиль
- 0.75 — 75-й перцентиль (также третий квартиль)

```
df
```

	values
0	1
1	2
2	3
3	4
4	5

```
df.quantile(q=0.75)
```

```
values    4.0
Name: 0.75, dtype: float64
```

Также в `q` можно передать список всех желаемых перцентилей:

```
df.quantile(q=[0.5, 0.7])
```

	values
0.5	3.0
0.7	3.8

Что делать со значениями, не попадающими в перцентиль?

Если ровно по заданному перцентилю в датафрейме нет значения, то по умолчанию метод линейно выведет его. Поменять это поведение можно с помощью параметра *interpolation*. Вариант 'higher' берёт большую точку из смежных:

```
df.quantile(q=[0.5, 0.7], interpolation='higher')
```

	values
0.5	3
0.7	4

P.S. Краткость — сестра таланта, но не в создании названий параметров в одну букву. Здесь сложно перепутать, так как название метода намекает, но когда будете создавать свои функции и методы, называйте всё осмысленно.

Документация

> Альтернативный способ создания колонок

Колонки в датафрейме можно также создать с помощью метода `assign`. Он возвращает исходный датафрейм с добавленными колонками — нужно перезадать переменную, чтобы изменить датафрейм.

В метод передаются аргументы формата `название колонки = её содержимое` — как название параметра и его значение при вызове функции. Здесь название колонок нужно писать без кавычек.

Так, мы задаём новую колонку `loyality`, значения в которой являются отношением значений в колонке `max_orders` к значениям в колонке `orders`:

```
users_data = users_data.assign(loyality = users_data.max_orders / users_data.orders)
```

Больше информации

> Замена пропущенных значений

Замена пропущенных значений — часто возникающая задача в некоторых сферах. Одно из простых решений — заменить все на одно значение (например, 0). Это можно сделать с помощью метода `fillna`, принимающего значение, на которое будут заменены пропущенные значения:

```
retention_pivot.head(3)
```

date	2019-04-01	2019-04-02	2019-04-03	2019-04-04	2019-04-05	2019-04-06
start_date						
2019-04-01	1.0	0.6	0.4	0.3	0.2	0.1
2019-04-02	NaN	1.0	0.6	0.4	0.3	0.2
2019-04-03	NaN	NaN	1.0	0.6	0.4	0.3

```
retention_pivot.head(3).fillna(0)
```

date start_date	2019-04-01	2019-04-02	2019-04-03	2019-04-04	2019-04-05	2019-04-06
2019-04-01	1.0	0.6	0.4	0.3	0.2	0.1
2019-04-02	0.0	1.0	0.6	0.4	0.3	0.2
2019-04-03	0.0	0.0	1.0	0.6	0.4	0.3

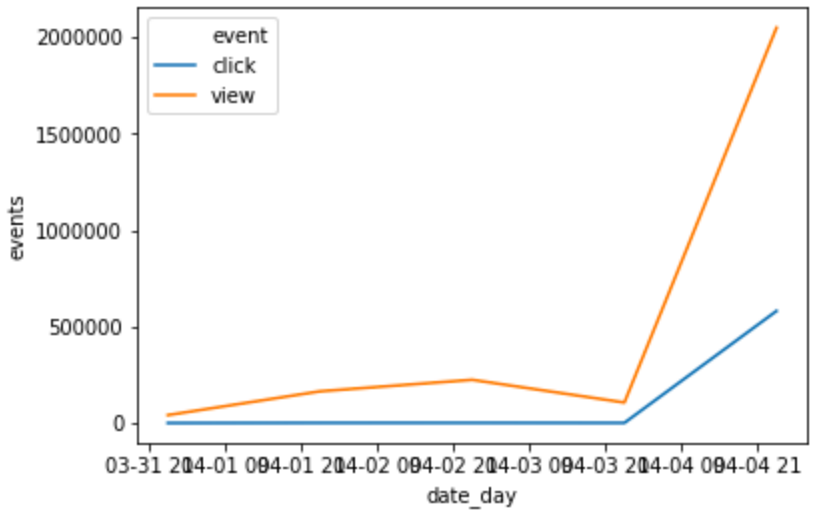
Документация

> Lineplot

Line chart — линейная диаграмма. По оси x и y откладываются значения точек, эти точки соединяются. Аргумент `hue` принимает имя колонки, по значениям которой идёт разделение на цвета:

```
sns.lineplot(x="date_day", y="events", hue="event", data=events_by_day)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fedf47bae80>
```



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()  # после этой операции df будет сод
ержать <= строк, чем до неё
```

Дополнительные аргументы

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

```
# удалит дубликаты, только если повторяющиеся значения будут
в колонках 'Date' и 'Last'
df.drop_duplicates(subset=['Date', 'Last'])
```

Документация

Вывести все дубликаты:

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

Документация