KARPOV.COURSES >>>
КОНСПЕКТ

> Конспект > 4 урок > Исследование данных и создание метрик

> Оглавление

> Уникальные значения

`unique` — метод, возвращающий уникальные значения в колонке.

```
data.fam_sp.unique()
```

```
array(['PASTA ALIMENTICIA SE'], dtype=object)
```

Уникальные значения возвращаются массивом array, для простоты воспринимайте их как списки.

Документация

Число уникальных значений

`nunique` — метод, который считает число уникальных значений в колонке.

```
data.fam_sp.nunique()
```

```
1
```

Документация

> Разделение строк

`split` — метод, разбивающий строку на куски и помещающий фрагменты в список. По умолчанию делит по пустым символам (пробел, табы, перенос строки).

```
brand_info = 'MARAVILLA 500 G Store_Brand'
brand_info.split()
```

```
['MARAVILLA', '500', 'G', 'Store_Brand']
```

Больше информации

> Анонимные функции

Обычно используются, когда нужно куда-то быстро поместить нечасто используемый функционал. Если вы планируете использовать анонимную функцию больше одного раза, напишите обычную функцию :)

```
lambda x: do something
```

- `lambda` — ключевое слово, задающее анонимную функцию (не имеющую имени)
- `x` — то, как мы назвали аргумент, принимаемый функцией
- `:` — разделяет заголовок и тело безымянной функции
- `do something` — тело функции, должно помещаться в одну строчку, будет автоматически возвращаться без `return`

```
# Take 1 argument and add 3 to itlambda x: x + 3
```

Один из примеров использования лямбда-функции — переименование колонок в датафрейме. Здесь мы делаем их заглавными и заменяем дефисы на нижние подчёркивания:

```
# df is a dataframe as usual
df = df.rename(columns=lambda c: c.upper().replace('-', '_'))
```

Больше информации

> Применение функций к датафрейму

`apply` — применяет переданную в него функцию ко всем колонкам вызванного датафрейма. Чтобы применить функцию к одной колонке датафрейма, можно выбрать её перед применением `apply`, например:

```
data.art_sp.apply(lambda c: c.split()[-1])

0        Store_Brand
1        Store_Brand
2            Brand_1
```

Документация

> Сброс индекса

Иногда вам может захотеться перевести индекс датафрейма в колонку. Для этого существует метод `reset_index`. Индексом становится дефолтная последовательность чисел от 0 до числа строк - 1.

```
df
```

event	click	view
date_day		
2019-04-01	881	41857
2019-04-02	1612	165174
2019-04-03	1733	224843
2019-04-04	1447	107098
2019-04-05	581790	2050279

```
df.reset_index()
```

	event	date_day	click	view
0	2019-04-01	881	41857	
1	2019-04-02	1612	165174	
2	2019-04-03	1733	224843	
3	2019-04-04	1447	107098	
4	2019-04-05	581790	2050279	

Удаление индекса

Аргумент `drop` отвечает за то, нужно ли переводить индекс в колонку или убрать его из таблицы:

```
df.reset_index(drop=True)
```

	event	click	view
0		881	41857
1		1612	165174
2		1733	224843
3		1447	107098
4		581790	2050279

Документация

Обратная сбросу индекса операция - установка индекса, когда существующая колонка (или колонки) переводят в индекс. Для этого существует метод `set_index()`, в качестве аргумента в него нужно передать строку с названием колонки, которая должна стать индексом (или список со строками - названиями колонок, если их несколько).

Документация

> Поиск пропущенных значений

`isna` — это чудо-метод, с помощью которого можно быстро найти пропущенные значения в датафрейме:

```
df.head()
```

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

Применив его, на выходе мы получаем датафрейм той же размерности, где в каждой ячейке `True` или `False` — в зависимости от того, было ли значение пропущено:

```
df.isna()
```

	Id	sum
0	False	False
1	False	False
2	False	True
3	False	False
4	False	True

В связке с ним можно использовать, например, `sum`, чтобы посмотреть на число `NA` в разных колонках:

```
df.isna().sum()
```

```
id      0
sum     4
dtype: int64
```

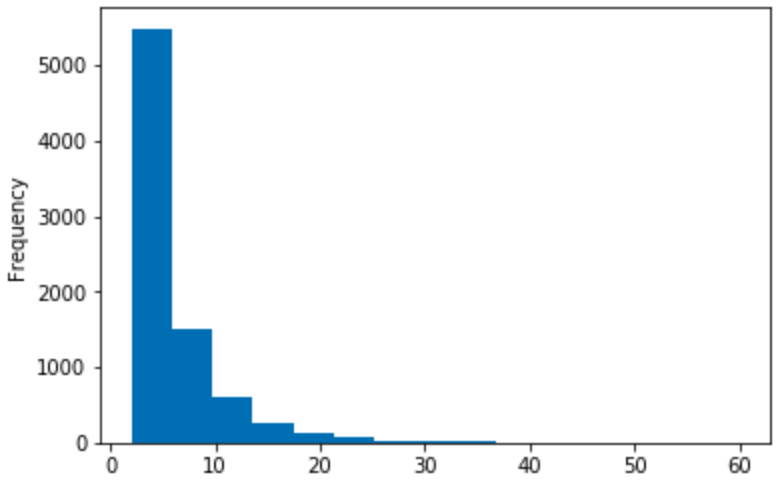
Документация

> Гистограмма в pandas

Самый простой способ визуализировать данные — вызвать метод `plot` у датафрейма (или его колонки). Например, гистограмма значений в колонке `orders`:

```
users_data.orders.plot('hist', bins=15)
```

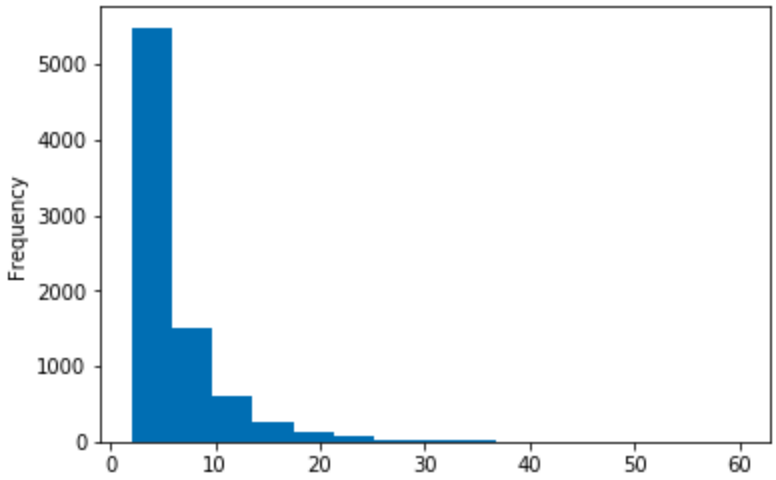
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe0d35869e8>
```



Другой вариант записи:

```
users_data.orders.plot.hist(bins=15)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe0f0dd1ef0>
```



Функции рисования имеют весьма большое количество параметров, используйте их при необходимости. `bins` здесь — число диапазонов (корзин/бакетов), на которые мы разделяем значения.

Документация

> Использование matplotlib и seaborn вместе

Через `matplotlib` можно нарисовать что угодно, но часто на это уходит слишком много строк кода, поэтому её в основном используют для тонкой настройки графиков, построенных с помощью `seaborn`, и их сохранения.

Настройка графиков

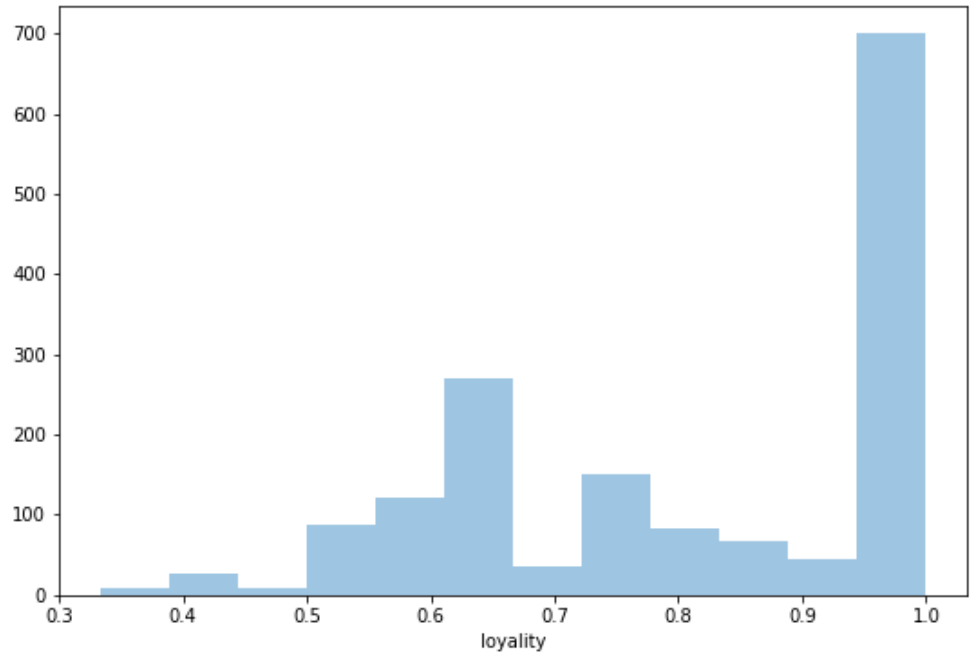
Важный момент: большинство настроек должны быть написаны к каждому графику отдельно. Иными словами, настройки, написанные в ячейке с одним графиком, не будут применены к другому.

Изменить размер

В `figure` в `figsize` подаётся кортеж (как список, только в круглых скобках) с масштабом графика формата `(ширина, высота)`.

```
plt.figure(figsize=(9, 6))
sns.distplot(users_data.query('lovely_brand == "Store_Brand"').loyality, kde=False)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe0d3514240>
```



Больше информации

Сохранение картинки

Сохранить график можно с помощью `savefig` , где аргумент — путь к сохраняемой картинке (желаемое название и формат):

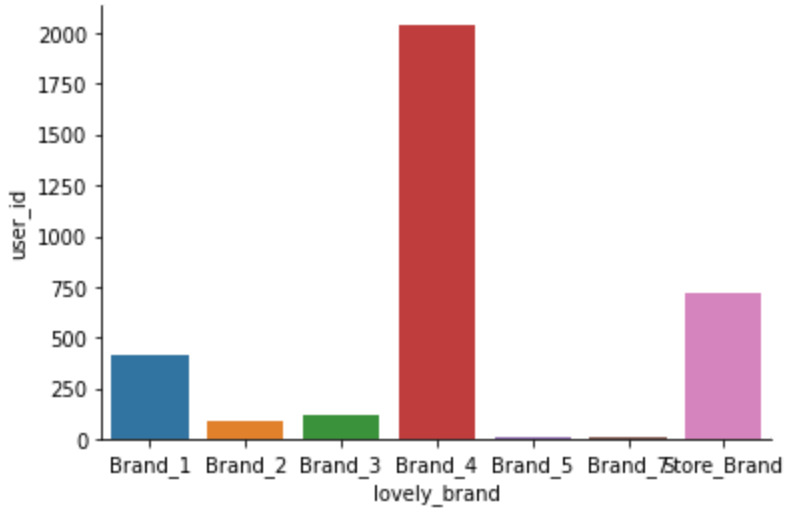
```
sns.distplot(users_data.query('lovely_brand == "Store_Brand"').loyality, kde=False)
plt.savefig('1.jpg')
```

Документация

Убрать рамку

Есть в `seaborn` и свои функции для настройки внешнего вида графиков. Например, `sns.despine` уберет часть рамки, по умолчанию — сверху и справа.

```
sns.barplot(x="lovely_brand", y="user_id", data=brands_loyalty)
sns.despine()
```



Документация

> Атрибуты времени

Временные серии обладают атрибутом dt, в котором находится множество атрибутов и методов для доступа ко времени. Давайте посмотрим на часть из них:

```
df.start_at
```

```
0        2010-11-16 16:44:00
1        2010-06-01 00:34:00
2        2010-05-31 05:01:00
3        2010-06-01 00:29:00
4        2010-09-11 23:55:00
                 ...
23106    2010-07-31 13:15:00
23107    2010-10-02 05:26:00
23108    2010-09-21 09:56:00
23109    2010-04-29 04:30:00
23110    2010-03-16 19:58:00
Name: start_at, Length: 23111, dtype: datetime64[ns]
```

Микросекунды

`dt.microsecond` — сколько микросекунд в указанном времени (если время 5 минут, 0 секунд и 3 микросекунды, то он вернёт 3, а не 5 * 60 * 106).

```
df.start_at.dt.microsecond
```

```
0        0
1        0
2        0
3        0
4        0
        ..
23106    0
23107    0
23108    0
23109    0
23110    0
Name: start_at, Length: 23111, dtype: int64
```

Секунды

```
df.start_at.dt.second
```

```
0        0
1        0
2        0
3        0
4        0
        ..
23106    0
23107    0
23108    0
23109    0
23110    0
Name: start_at, Length: 23111, dtype: int64
```

Минуты

```
df.start_at.dt.minute
```

```
0        44
1        34
2         1
3        29
4        55
        ..
23106    15
23107    26
23108    56
23109    30
23110    58
Name: start_at, Length: 23111, dtype: int64
```

Час

```
df.start_at.dt.hour
```

```
0        16
1         0
2         5
3         0
4        23
         ..
23106    13
23107     5
23108     9
23109     4
23110    19
Name: start_at, Length: 23111, dtype: int64
```

День месяца

```
df.start_at.dt.day
```

```
0        16
1         1
2        31
3         1
4        11
         ..
23106    31
23107     2
23108    21
23109    29
23110    16
Name: start_at, Length: 23111, dtype: int64
```

Номер дня недели

```
df.start_at.dt.weekday
```

```
0        1
1        1
2        0
3        1
4        5
        ..
23106    5
23107    5
23108    1
23109    3
23110    1
Name: start_at, Length: 23111, dtype: int64
```

Начинается с 0, то есть понедельник будет днем 0, а воскресенье - днем 6.

Имя дня недели

```
df.start_at.dt.day_name()
```

```
0         Tuesday
1         Tuesday
2          Monday
3         Tuesday
4        Saturday
           ...
23106    Saturday
23107    Saturday
23108     Tuesday
23109    Thursday
23110     Tuesday
Name: start_at, Length: 23111, dtype: object
```

Номер недели в году

```
df.start_at.dt.week
```

```
0         46
1         22
2         22
3         22
4         36
          ..
23106     30
23107     39
23108     38
23109     17
23110     11
Name: start_at, Length: 23111, dtype: int64
```

Номер месяца

```
df.start_at.dt.month
```

```
0         11
1          6
2          5
3          6
4          9
          ..
23106      7
23107     10
23108      9
23109      4
23110      3
Name: start_at, Length: 23111, dtype: int64
```

Название месяца

```
df.start_at.dt.month_name()
```

```
0          November
1              June
2               May
3              June
4         September
            ...
23106          July
23107       October
23108     September
23109         April
23110         March
Name: start_at, Length: 23111, dtype: object
```

Год

```
df.start_at.dt.year
```

```
0        2010
1        2010
2        2010
3        2010
4        2010
        ...
23106    2010
23107    2010
23108    2010
23109    2010
23110    2010
Name: start_at, Length: 23111, dtype: int64
```

Число дней в текущем месяце

```
df.start_at.dt.daysinmonth
```

```
0         30
1         30
2         31
3         30
4         30
         ..
23106     31
23107     31
23108     30
23109     30
23110     31
Name: start_at, Length: 23111, dtype: int64
```

Разность времени

Timedelta — это тип данных, соответствующий разнице двух времён, то есть какая-то продолжительность времени.

```
df.wait_time
```

```
0       -1 days +23:42:00
1                     NaT
2                     NaT
3                     NaT
4                00:05:00
                ...
23106            00:00:00
23107   -1 days +23:47:00
23108   -1 days +23:51:00
23109            00:07:00
23110                 NaT
Name: wait_time, Length: 23111, dtype: timedelta64[ns]
```

Компоненты

Все единицы измерения времени можно извлечь сразу с помощью атрибута `components`

```
df.wait_time.dt.components
```

	days	hours	minutes	seconds	milliseconds	microseconds	nanoseconds
0	-1.0	23.0	42.0	0.0	0.0	0.0	0.0
1	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	0.0	0.0	5.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...
23106	0.0	0.0	0.0	0.0	0.0	0.0	0.0
23107	-1.0	23.0	47.0	0.0	0.0	0.0	0.0
23108	-1.0	23.0	51.0	0.0	0.0	0.0	0.0
23109	0.0	0.0	7.0	0.0	0.0	0.0	0.0
23110	NaN	NaN	NaN	NaN	NaN	NaN	NaN

23111 rows × 7 columns

Больше информации

> Дополнительные материалы

- статья про джойны, о которой Анатолий сказал в видео
- Пример использования lambda-функции
- 50 графиков на python с кодом