Машинное обучение

Практика EDA. Часть 1

Материалы > Анализ и обработка данных

На прошлом занятии мы изучали различные классификации данных, задачи EDA, а также познакомились с основными библиотеками для создания визуализаций. Сегодня мы свяжем эти концепции в практической работе по анализу датасета «Титаник» и датасета Tips.

В первую очередь подготовим датасеты.

Откроем блокнот к этому занятию

Подготовка данных

Датасет «Титаник»

Скачаем обучающий датасет «Титаник», подгрузим его в сессионное хранилище Google Colab и импортируем в блокнот.

train.csv **Скачать**

```
# для импорта используем функцию read_csv()
titanic = pd.read_csv('/content/train.csv')
```

Как мы уже знаем, посмотреть на первые или последние несколько (по умолчанию, пять) строк можно с помощью методов **.head()** и **.tail()** соответственно.

```
# посмотрим на первые три записи
titanic.head(3)
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN

Иногда для получения более объективного представления о данных удобно использовать **метод .sample()**, который по умолчанию выдает одно случайное наблюдение.

```
# выведем пять случайных строк
titanic.sample(5)
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare
207	208	1	3	Albimona, Mr. Nassef Cassem	male	26.00	0	0	2699	18.7875
68	69	1	3	Andersson, Miss. Erna Alexandra	female	17.00	4	2	3101281	7.9250
305	306	1	1	Allison, Master. Hudson Trevor	male	0.92	1	2	113781	151.5500
362	363	0	3	Barbara, Mrs. (Catherine David)	female	45.00	0	1	2691	14.4542
863	864	0	3	Sage, Miss. Dorothy Edith "Dolly"	female	NaN	8	2	CA. 2343	69.5500

Метод .info() для каждого столбца выводит количество непустных (not-null) значений и тип данных. Кроме того, этот метод считает количество столбцов каждого типа и общий объем памяти, занимаемый датасетом.

```
titanic.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  891 non-null    int64
 1   Survived     891 non-null    int64
 2   Pclass       891 non-null    int64
 3   Name         891 non-null    object
 4   Sex          891 non-null    object
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    int64
 7   Parch        891 non-null    int64
 8   Ticket       891 non-null    object
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object
 11  Embarked     889 non-null    object
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

Конечно, посмотреть количество пропусков удобнее, например, с помощью последовательного применения методов **.isnull()** и **.sum()**.

```
# метод .isnull() выдает логический массив, где пропуски обозначены как True
# метод .sum() по умолчанию суммирует эти True или единицы по столбцам (axis = 0)
titanic.isnull().sum()
```

```
PassengerId      0
Survived         0
```

```
Pclass          0
Name            0
Sex             0
Age           177
SibSp           0
Parch           0
Ticket          0
Fare            0
Cabin         687
Embarked        2
dtype: int64
```

Теперь выполним несложную предобработку данных.

```
# в частности, избавимся от столбца Cabin
titanic.drop(labels = 'Cabin', axis = 1, inplace = True)
# заполним пропуски в столбце Age медианным значением
titanic.Age.fillna(titanic.Age.median(), inplace = True)
# два пропущенных значения в столбце Embarked заполним портом Southhampton
titanic.Embarked.fillna('S', inplace = True)
# проверим результат (найдем общее количество пропусков сначала по столбцам, затем по с
titanic.isnull().sum().sum()
```

```
'0'
```

Более сложные методы обработки данных мы рассмотрим в третьем и четвертом разделах курса.

Датасет Tips

Кроме того, импортируем хранящийся в библиотеке Seaborn датасет Tips. В нем содержатся 244 записи о чаевых, которые официант ресторана получал на протяжении нескольких месяцев.

```
# для импорта воспользуемся функцией load_dataset() с параметром 'tips'
tips = sns.load_dataset('tips')
tips.head(3)
```

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3

Вновь воспользуемся методом **.info()**.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 244 entries, 0 to 243
Data columns (total 7 columns):
 #   Column      Non-Null Count  Dtype   
---  ------      --------------  -----   
 0   total_bill  244 non-null    float64 
 1   tip         244 non-null    float64 
 2   sex         244 non-null    category
 3   smoker      244 non-null    category
 4   day         244 non-null    category
 5   time        244 non-null    category
 6   size        244 non-null    int64   
dtypes: category(4), float64(2), int64(1)
memory usage: 7.4 KB
```

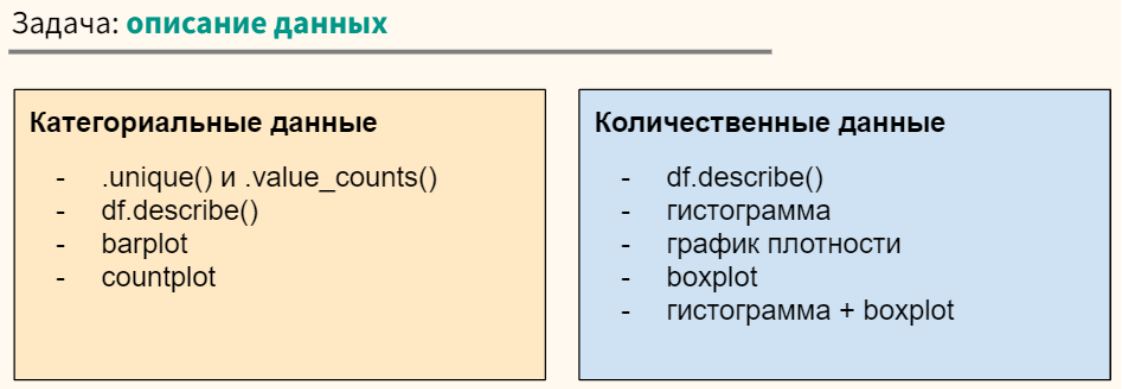
Пропущенных значений в этом датасете нет.

```
tips.isnull().sum()
```

```
total_bill    0
tip           0
sex           0
smoker        0
day           0
time          0
size          0
dtype: int64
```

Теперь, когда данные подгружены, перейдем к их описанию, нахождению различий и выявлению взаимосвязей.

Описание данных



Категориальные переменные

Методы .unique() и .value_counts()

Применение этих методов аналогично использованию метода библиотеки Numpy **np.unique()** с параметром `return_counts = True`. Применим его.

```
np.unique(titanic.Survived, return_counts = True)
```

```
(array([0, 1]), array([549, 342]))
```

Теперь воспользуемся методами библиотеки Pandas.

```
# первый метод возвращает только уникальные значения
titanic.Survived.unique()
```

```
array([0, 1])
```

```
# второй - уникальные значения и их частоту
titanic.Survived.value_counts()
```

```
0    549
1    342
Name: Survived, dtype: int64
```

При этом для нахождения относительной частоты делить на общее количество строк не нужно. Достаточно указать параметр `normalize = True`.

```
titanic.Survived.value_counts(normalize = True)
```

```
0    0.616162
1    0.383838
Name: Survived, dtype: float64
```

Долю «единичек» при наличии двух классов, обозначенных как 0 и 1, можно посчитать и так.

```
titanic.Survived.mean().round(2)
```

```
0.38
```

df.describe()

Исследование качественных переменных удобно начать с **метода .describe()**. Его применение к категориальным столбцам выдаст:

- общее количество значений (count);
- количество уникальных значений (unique);
- наиболее часто встречающееся значение (top);
- и количество таких значений (freq).

Применим метод **.describe()** к столбцам Sex и Embarked.

```
titanic[['Sex', 'Embarked']].describe()
```

	Sex	Embarked
count	891	891
unique	2	3
top	male	S
freq	577	646

Перейдем к графическим методам.

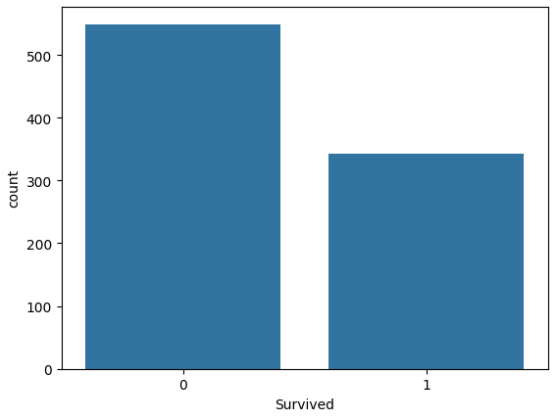
countplot и barplot

Рассмотрим два по сути однаковых графика **countplot** и **barplot**. И тот, и другой считают количество значений в каждой из категорий. С точки зрения Питона, различие заключается в том, что

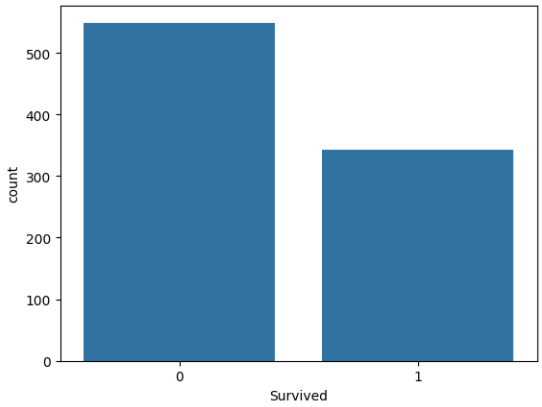
- в countplot мы считаем количество наблюдений в каждой из категорий в процессе построения графика; а
- в случае barplot эти метрики уже должны быть посчитаны.

Рассмотрим на примерах. Проще всего countplot и barplot построить с помощью библиотеки **Seaborn**.

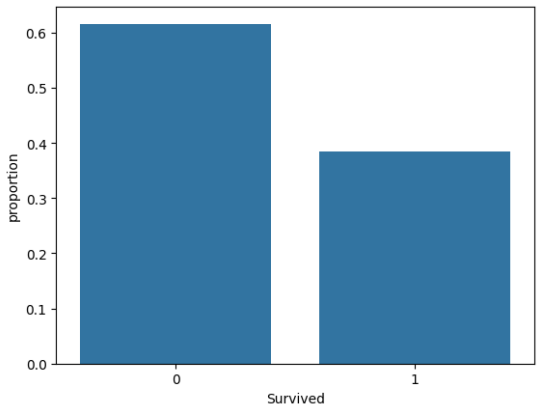
```
# функция countplot() сама посчитает количество наблюдений в каждой из категорий
sns.countplot(x = 'Survived', data = titanic);
```




```
# для функции barplot() количество наблюдений можно посчитать
# с помощью метода .value_counts()
sns.barplot(x = titanic.Survived, y = titanic.Survived.value_counts());
```



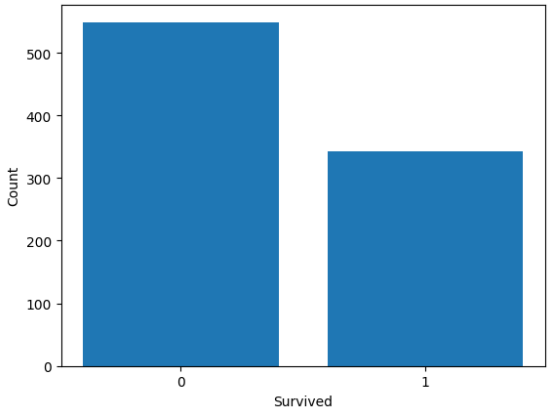

```
# относительное количество наблюдений удобно вывести с параметром normalize = True
sns.barplot(x = titanic.Survived,
            y = titanic.Survived.value_counts(normalize = True));
```



В библиотеке **Matplotlib** мы можем построить только barplot (**функция bar()**). Отдельного инструмента для построения countplot в ней нет. Количество наблюдений мы можем найти с помощью метода **.value_counts()**.

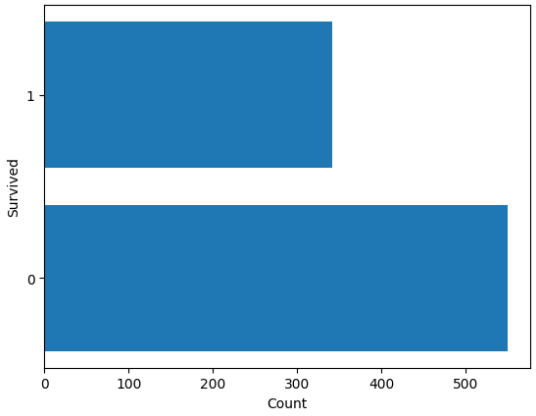
```
# первым параметром (по оси x) передадим уникальные значения,
# вторым параметром - количество наблюдений
plt.bar(titanic.Survived.unique(),
        titanic.Survived.value_counts(),
        # кроме того, явно пропишем значения оси x
        # (в противном случае будет указана просто числовая шкала)
        tick_label = ['0', '1'])
```

```
8
9  plt.xlabel('Survived')
10 plt.ylabel('Count');
```



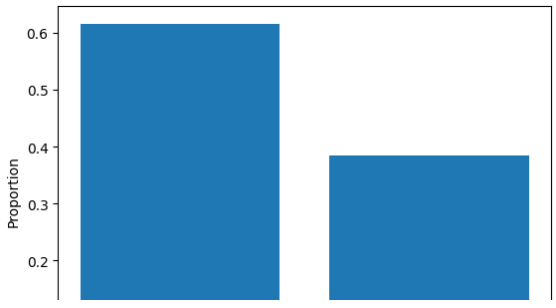
Горизонтальную столбчатую диаграмму (horizontal barplot) можно построить с помощью **функции barh()**.

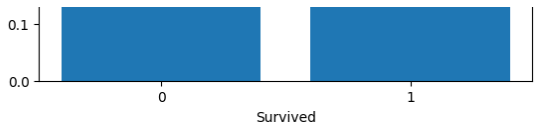
```
1 plt.barh(titanic.Survived.unique(),
2          titanic.Survived.value_counts(),
3          tick_label = ['0', '1'])
4
5 plt.xlabel('Count')
6 plt.ylabel('Survived');
```



Снова воспользуемся параметром `normalize = True` метода **.value_counts()** для нахождения относительной частоты каждой категории признака.

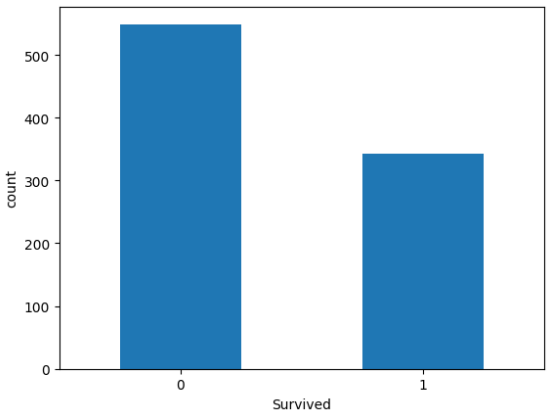
```
1 plt.bar(titanic.Survived.unique(),
2         titanic.Survived.value_counts(normalize = True),
3         tick_label = ['0', '1'])
4
5 plt.xlabel('Survived')
6 plt.ylabel('Proportion');
```





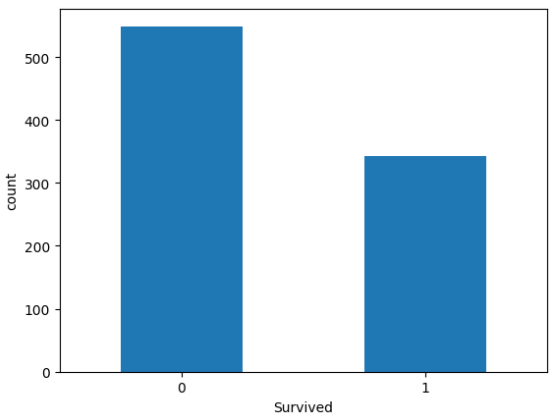
Для того чтобы построить такой график в библиотеке Pandas, вначале необходимо сгруппировать данные по столбцу Survived, затем выбрать один столбец (например, PassengerId), посчитать количество наблюдений в каждой группе через метод **.count()** и наконец построить столбчатую диаграмму с помощью **метода .plot.bar()**.

```
# параметр rot = 0 ставит деления шкалы по оси x вертикально
titanic.groupby('Survived')['PassengerId'].count().plot.bar(rot = 0)
plt.ylabel('count');
```



Код можно упростить, если сначала выбрать желаемый признак (столбец), затем воспользоваться методом **.value_counts()** и наконец применить **метод .plot.bar()**.

```
titanic.Survived.value_counts().plot.bar(rot = 0)
plt.xlabel('Survived')
plt.ylabel('count');
```



Мы продолжим изучать столбчатые диаграммы, когда перейдем к нахождению различий между двумя категориальными признаками.

Количественные данные

df.describe()

Если применить **метод .describe()** к количественным данным, то результат будет отличаться от рассмотренных выше категориальных признаков.

```
# применим метод .describe() к количественным признакам
tips[['total_bill', 'tip']].describe().round(2)
```

	total_bill	tip
count	244.00	244.00
mean	19.79	3.00
std	8.90	1.38
min	3.07	1.00
25%	13.35	2.00
50%	17.80	2.90
75%	24.13	3.56
max	50.81	10.00

Как мы видим метод выдает:

- **count** — количество наблюдений;
- **mean** — среднее арифметическое;
- **std** или standard deviation — среднее квадратическое отклонение;
- **min** и **max** — минимальное и максимальное значения; а также
- **25%**, **50%** и **75%** — первый, второй (он же медиана) и третий квартили.

Здесь будет полезно сделать небольшое отступление и ближе познакомиться с новыми для нас способами оценки данных.

Среднее арифметическое и СКО

К настоящему моменту мы уже знаем, как находить среднее арифметическое и среднее квадратическое отклонение. Проблема же этих метрик, как мы уже говорили, заключается в том, что они сильно подвержены выбросам.

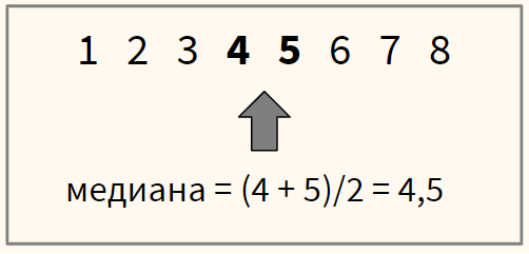
Квантили и робастная статистика

В этом смысле медиана дает более надежную оценку среднего при наличии выбросов.

> Статистические методы и алгоритмы, устойчивые к выбросам и менее зависимые от *предположений* (assumptions) модели, еще называют **робастными** (robust statistics).

Напомню, что **медианой** называется число, которое находится в середине упорядоченного от меньшего к большему набора чисел. В случае нечетного количества чисел, мы просто берем то значение, которое находится посередине.

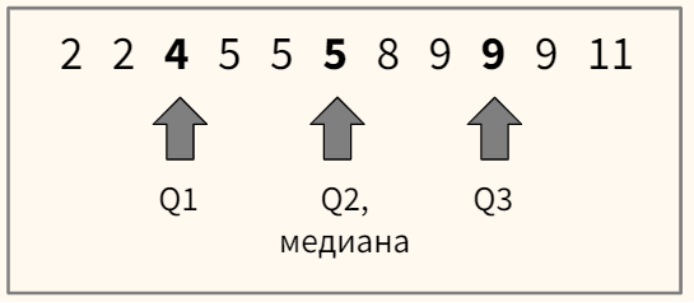
Если количество чисел четное — два срединных значения складываются и делятся на два.



> Медиану можно также определить как значение, которое наши данные (или случайная величина) не превышают с вероятностью 50 процентов (отсюда знак % в выводе метода **.describe()**).

Аналогично можно найти, например, значение, которое величина не будет превышать с вероятностью 25 или 75 процентов. Такие значения будут называться первым и третьим **квартилями** (quartile, от латинского — quarta, «четверть»), потому что они делят распределение на четыре части.

По-английски первый, второй и третий квартили принято обозначать как Q1, Q2 и Q3.



Кроме этого, можно найти **децили** (deciles), они делят распределение на десять частей. Наконец, если мы хотим найти конкретное значение, то будем искать **квантиль** (quantile). Если вероятность выражена в процентах, то квантиль принято называть **процентилем** или **перцентилем** (percentile).

Вывести конкретный процентиль в методе **.describe()** можно с помощью параметра *percentiles*.

```
1 # выведем второй и четвертый дециль, а также 99-й процентиль
2 tips[['total_bill', 'tip']].describe(percentiles = [0.2, 0.4, 0.99]).round(2)
```

	total_bill	tip
count	244.00	244.00
mean	19.79	3.00

mean	16.79	3.00
std	8.90	1.38
min	3.07	1.00
20%	12.64	2.00
40%	16.22	2.48
50%	17.80	2.90
99%	48.23	7.21
max	50.81	10.00

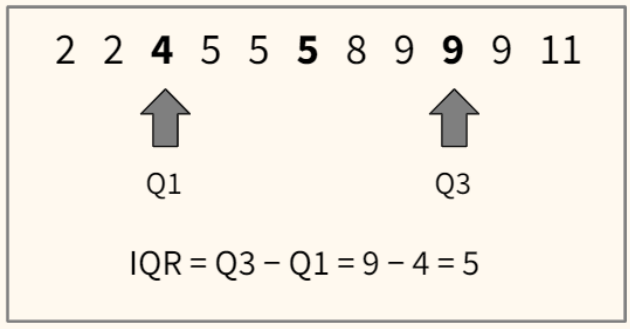
Закрепим полученные знания, проанализировав приведенные выше метрики.

- Медианное значение обоих признаков чуть ниже среднего арифметического;
- 40 процентов чаевых были ниже 2,48 доллара; наконец
- 99 процентов чеков были ниже 48,23 доллара.

Рассмотрим еще одну очень полезную меру разброса.

Межквартильный размах

Межквартильный размах (interquartile range) — робастная (устойчивая к выбросам) альтернатива среднему квадратическому отклонению. Рассчитывается как разница между третьим (Q3) и первым (Q1) квартилями.



Зачастую количественные данные удобнее анализировать с помощью графиков. Для этого есть три основных инструмента: гистограмма, график плотности и boxplot.

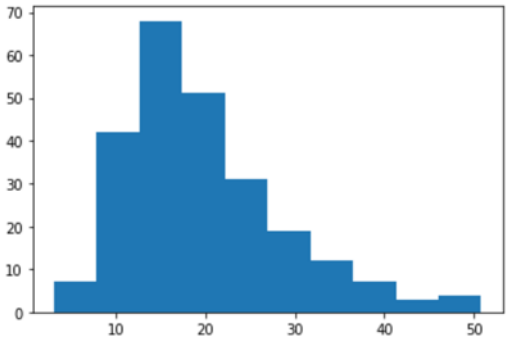
Гистограмма

С **гистограммой** мы уже знакомы.

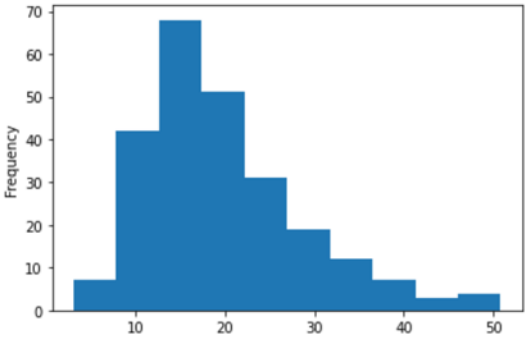
> Напомню, что для построения гистограммы мы делим наши данные на интервалы (bins) и считаем, сколько наблюдений попало в каждый из них.

Построим несколько графиков с использованием рассматриваемых нами библиотек.

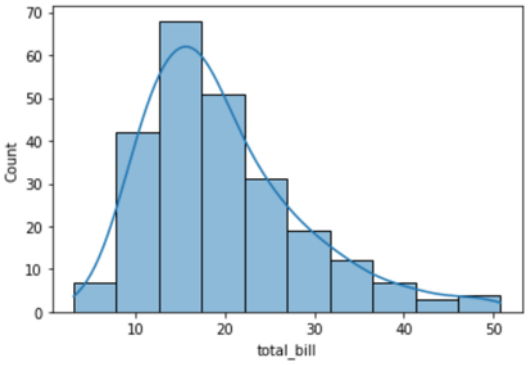
```
# гистограмма распределения размера чека с помощью библиотеки Matplotlib
plt.hist(tips.total_bill, bins = 10);
```



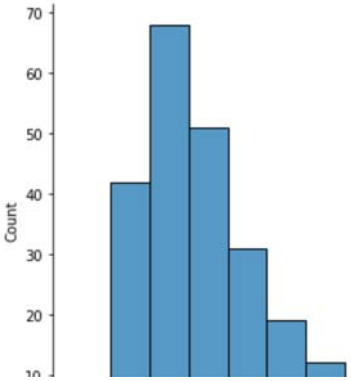

```
# такую же гистограмму можно построить с помощью Pandas
tips.total_bill.plot.hist(bins = 10);
```

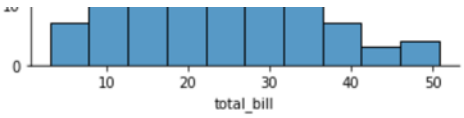



```
# в библиотеке Seaborn мы указываем источник данных, что будет на оси x и количество ин
# параметр kde = True добавляет кривую плотности распределения
sns.histplot(data = tips, x = 'total_bill', bins = 10, kde = True);
```



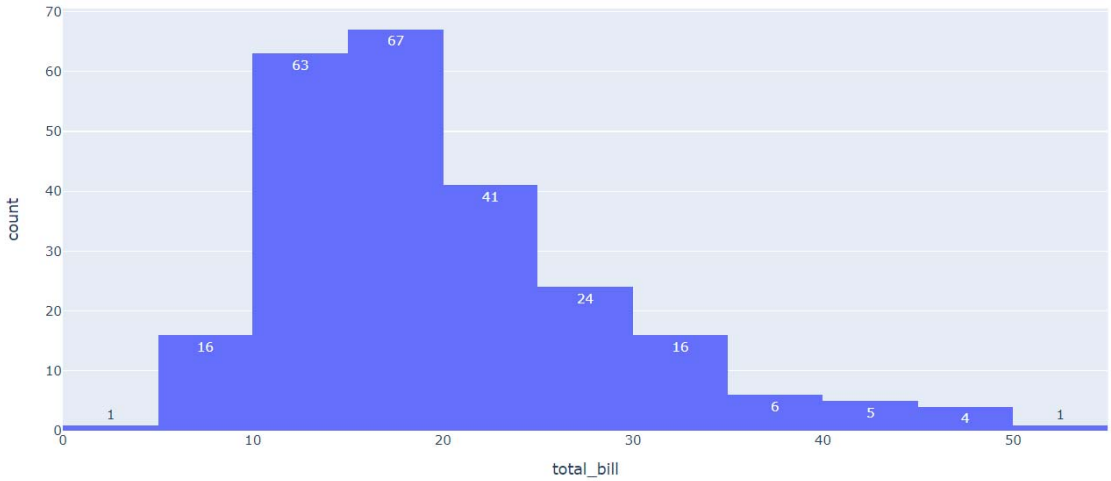

```
# функция displot() - еще один способ построить гистограмму в Seaborn
# для этого используется параметр по умолчанию kind = 'hist'
sns.displot(data = tips, x = 'total_bill', kind = 'hist', bins = 10);
```





Обратите внимание, что функция называется именно **displot()**, а не **distplot()**, которая объявлена устаревшей и не рекомендуемой к использованию (deprecated).

```
# Plotly, как уже было сказано, позволяет построить интерактивную гистограмму
# параметр text_auto = True выводит количество наблюдений в каждом интервале
px.histogram(tips, x = 'total_bill', nbins = 10, text_auto = True)
```



Что можно сказать после изучения этих графиков? Распределение **скошено вправо** (skewed right или positively skewed), т.е. в нем есть несколько чеков на достаточно большую сумму, которые и создают правый «хвост».

Хотя на графике эта особенность распределения более очевидна, к такому же выводу мы могли прийти проанализировав разницу между средним арифметическим и медианой.

> Когда медиана меньше среднего арифметического, мы наблюдаем *скошенное вправо* распределение.

В целом соотношение скошенности распределения со средним арифметическом, медианой и модой приведено на графике ниже.

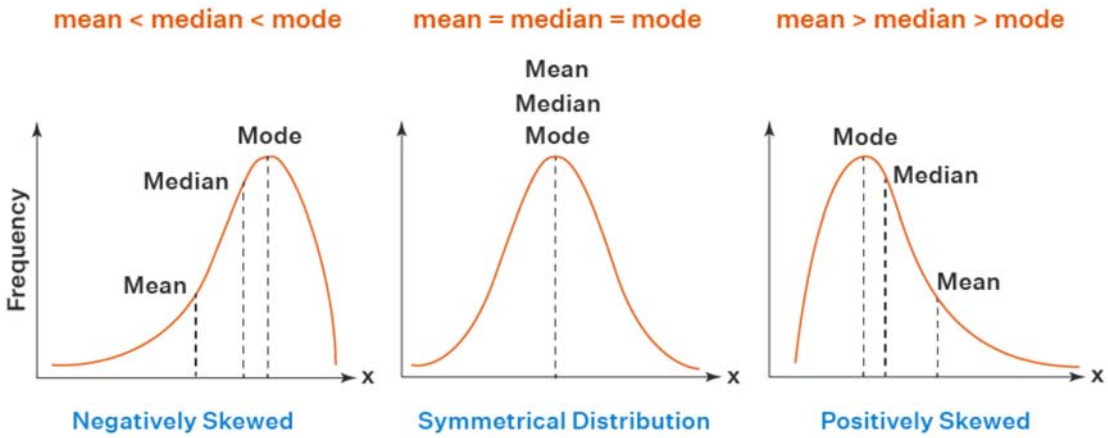


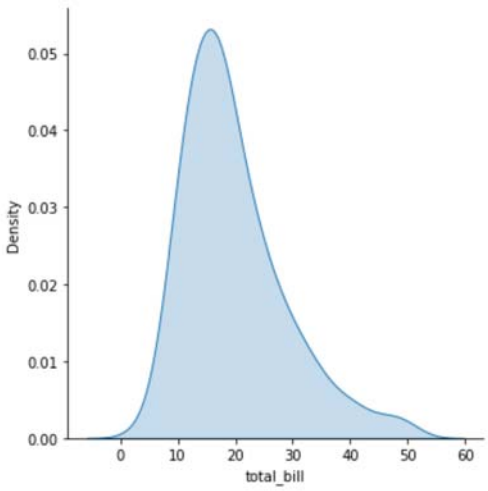
График плотности

С **графиком плотности** (density plot) мы столкнулись при изучении нормального распределения и модуля random.

> Напомню, график плотности позволяет визуализировать непрерывное случайное распределение.

Построим такой график с помощью библиотеки Seaborn.

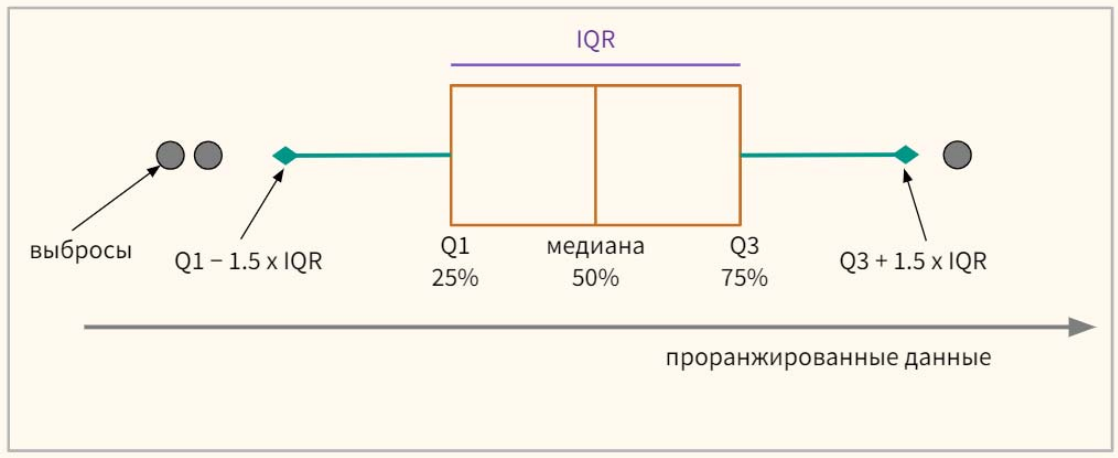
```
# для этого используем функцию displot(), которой передадим датафрейм tips,
# какой признак вывести по оси x, тип графика kind = 'kde',
# а также заполним график цветом через fill = True
sns.displot(tips, x = 'total_bill', kind = 'kde', fill = True);
```



Добавлю, для справки, что в Seaborn значение параметра **kde** расшифровывается как **kernel density estimation** (ядерная оценка плотности), непараметрический способ оценки плотности случайной величины.

boxplot

После знакомства с квантилями и робастной статистикой понимание графика **boxplot** или как его еще называют **box-and-wisker plot** (ящик с усами) не вызовет сложностей.



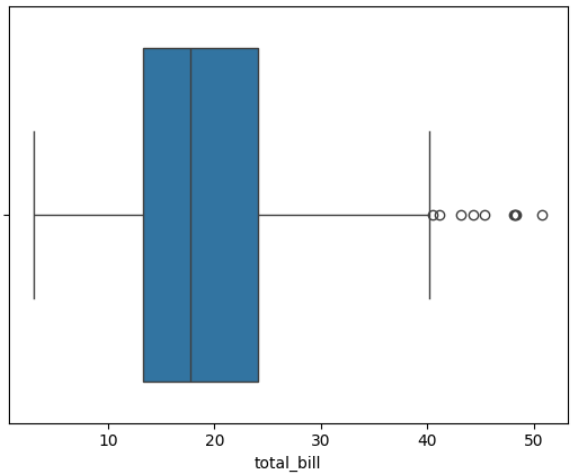
В первую очередь замечу, что boxplot строится на *проранжированных по возрастанию данных*. Теперь обратим внимание на сам «*ящик*» (box):

- его левый край отражает первый квартиль (Q1) или 25-тый процентиль (25%);
- вертикальная полоса посередине — медиана, второй квартиль (Q2) или 50-тый процентиль (50%);
- правый край, соответственно, третий квартиль (Q3) или 75-тый процентиль (75%);
- ширина ящика равна межквартильному размаху (IQR).

Усы (whiskers), то есть линии с ромбами на концах, отражают разброс данных за пределами IQR и рассчитываются как функция от этого значения. Данные, которые находятся за пределами этого диапазона, считаются *выбросами* (outliers).

Давайте построим этот график с помощью Seaborn.

```
# для этого функции boxplot() достаточно передать параметр x
# с данными необходимого столбца
sns.boxplot(x = tips.total_bill);
```



Проверим расчет межквартильного размаха и усов. Вновь посмотрим на результат метода **.describe()**.

	total_bill	tip
count	244.00	244.00
mean	19.79	3.00
std	8.90	1.38
min	3.07	1.00
25%	13.35	2.00
50%	17.80	2.90
75%	24.13	3.56
max	50.81	10.00

Первый и третий квартили $(13{,}35, 24{,}13)$, а также медиана $(17{,}80)$ соответствуют графику. Рассчитаем IQR.

$$\text{IQR} = \text{Q3} - \text{Q1} = 24{,}13 - 13{,}35 = 10{,}78$$

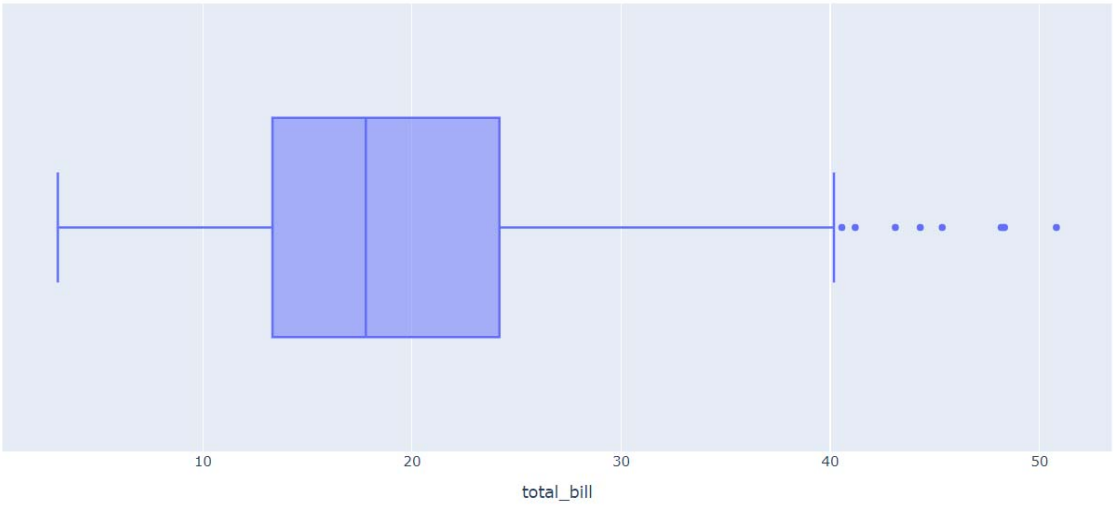
Теперь определим длину «усов».

$$\text{left} = \text{Q1} - 1{,}5 \times \text{IQR} = 13{,}35 - 1{,}5 \times 10{,}78 = -2{,}82$$

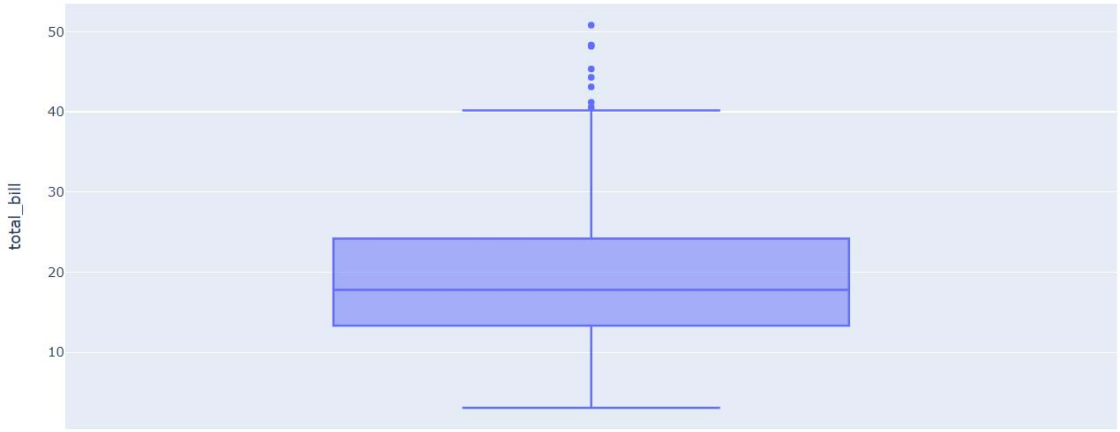
$$\text{right} = \text{Q3} + 1{,}5 \times \text{IQR} = 24{,}13 + 1{,}5 \times 10{,}78 = 40{,}3$$

Как мы видим, значения на графике совпадают с нашими расчетами. Аналогично мы можем воспользовать библиотекой Plotly.

```
# если передать нужный нам столбец в параметр x,
# то мы получим горизонтальный boxplot
px.box(tips, x = 'total_bill')
```




```
# если в y, то вертикальный
px.box(tips, y = 'total_bill')
```



Также приведу код для библиотек Matplotlib и Pandas.

```
# boxplot в Matplotlib
plt.boxplot(tips.total_bill);
```

```
# boxplot в Pandas
tips.total_bill.plot.box();
```

Гистограмма и boxplot

Гистограмма, с одной стороны, и boxplot, с другой, имеют свои достоинства и недостатки. В

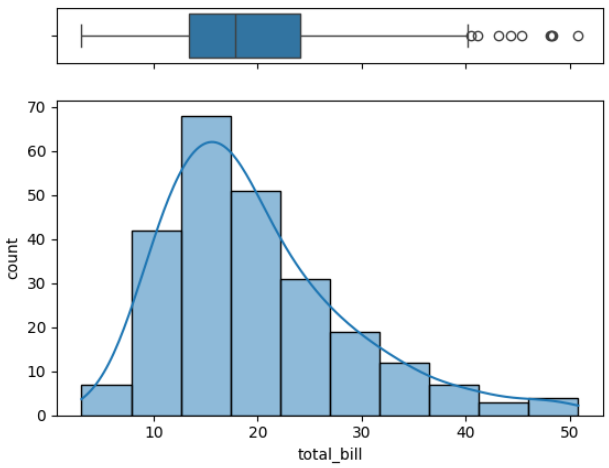
частности,

- **гистограмма** хорошо выявляет полимодальность (то есть несколько мод, «горбиков» в данных), при этом она сильно зависит от выбранного количества интервалов и не показывает выбросы;
- **boxplot** наоборот, показывает выбросы, но не справляется с полимодальностью.

Поэтому часто бывает удобно сразу построить оба графика распределения данных. Первый подобный график мы построили при изучении нормального распределения. Теперь у нас больше знаний, и мы лучше поймем суть каждого из них.

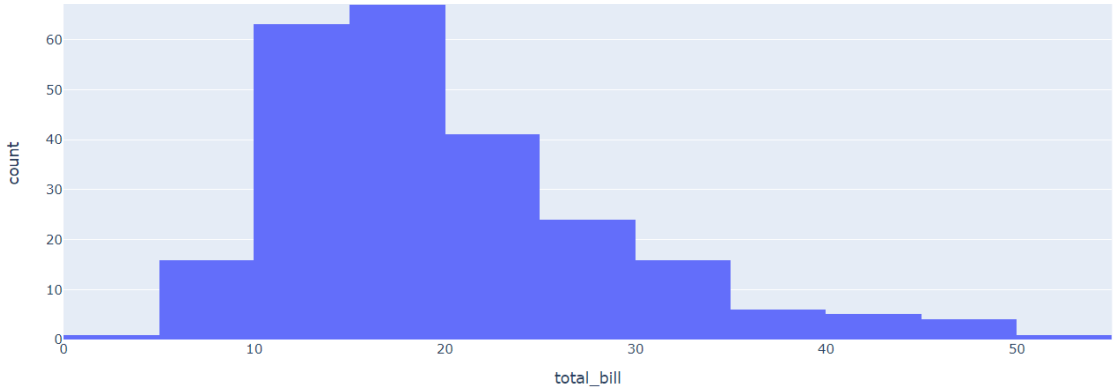
Вначале построим совмещенный график с помощью двух библиотек: **Matplotlib** и **Seaborn**. Первую мы будем использовать для создания подграфиков (рассмотрены в третьей части) и подписей, вторую — для самих визуализаций.

```
# создадим два подграфика ax_box и ax_hist
# кроме того, укажем, что нам нужны:
fig, (ax_box, ax_hist) = plt.subplots(2, # две строки в сетке подграфиков,
                                      sharex = True, # единая шкала по оси x и
                                      gridspec_kw = {'height_ratios': (.15, .85)}) # про

# затем создадим графики, указав через параметр ax, в какой подграфик поместить каждый
sns.boxplot(x = tips['total_bill'], ax = ax_box)
sns.histplot(x = tips['total_bill'], ax = ax_hist, bins = 10, kde = True)

# добавим подписи к каждому из графиков через метод .set()
ax_box.set(xlabel = '') # пустые кавычки удаляют подпись (!)
ax_hist.set(xlabel = 'total_bill')
ax_hist.set(ylabel = 'count')

# выведем результат
plt.show()
```

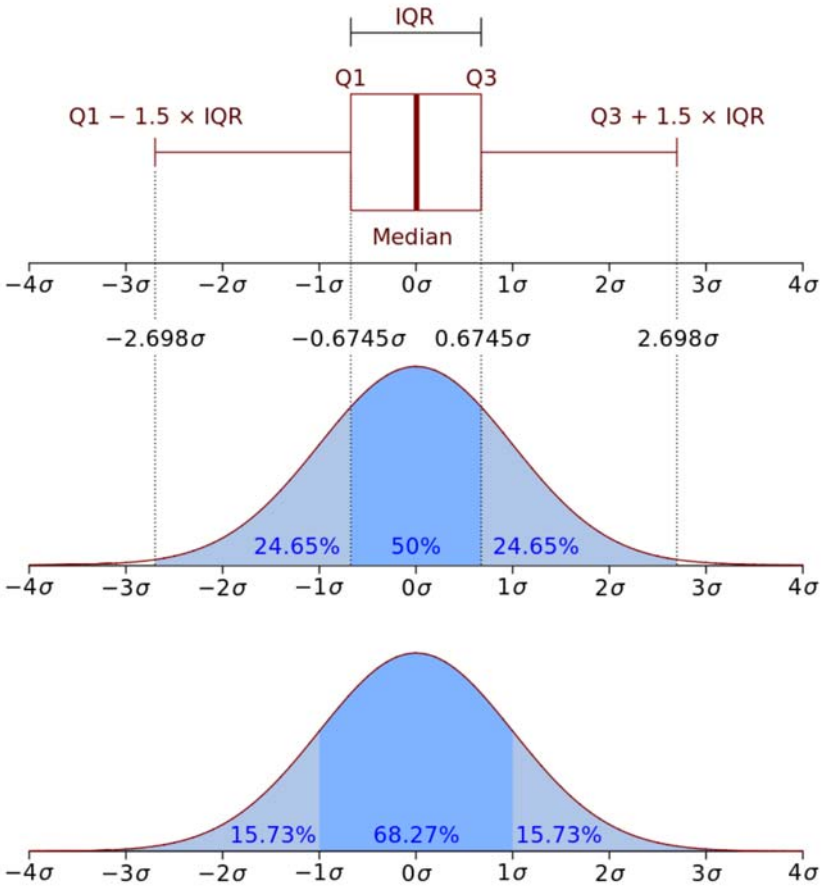


В **Plotly** такой график можно построить с меньшим количеством кода.

```
# воспользуемся функцией histogram(),
px.histogram(tips, # передав ей датафрейм,
             x = 'total_bill', # конкретный столбец для построения данных,
             nbins = 10, # количество интервалов в гистограмме
             marginal = 'box') # и тип дополнительного графика
```



Прежде чем перейти к нахождению различий между признаками, еще раз приведу график плотности и boxplot нормального распределения с показателями среднего арифметического, СКО (σ, сигма), медианы и остальных квартилей, межквартильного размаха (IQR) и других метрик.



Источник: Википедия

После изучения материалов этого занятия, думаю, он стал более понятен.