Визуализация данных и статистика

Оглавление

Визуализация данных

1.1 Введение

Наш курс посвящен анализу данных, и он не ограничивается только алгоритмами машинного обучения. Немаловажную роль занимает теория вероятности и статистика. Понимание статистики поможет вам в дальнейшем подготовить данные для обучения, например, понять какое распределение имеет целевая переменная, избежать несбалансированности классов данных, найти зависимости между признаками и целевой переменной. Вы узнаете, или вспомните основные определения теории вероятностей и математической статистики, нужные для курса, и научитесь генерировать распределение в *SciPy*, находить доверительный интервал и проводить AB-тестирование.

1.2 Визуализация с Matplotlib

Вы могли быть знакомы с библиотекой для визуализации данных *Matplotlib*, если работали с *pandas*, когда визуализировали распределение той или иной колонки, потому что она лежит в основе *pandas* и множества других библиотек. Поэтому вам может быть очень полезно уметь с ней работать, и мы познакомимся с тем, как строить различные графики, как их настраивать, и как их сохранять в изображение.

Начнем с того, что посмотрим, как визуализируется график чего бы то ни было в *pandas*. Сделаем *DataFrame*, в котором будет две колонки: x и y.

```
import pandas as pd
from numpy.random import exponential

df = pd.DataFrame({'x': range(20), 'y': exponential(10,20)})
```

Для того чтобы что-то изобразить в IPython Notebook, вам нужно сказать, чтобы все, что рисуется в *Matplotlib*, рисовалось внутри, а не на отдельном окошке, как это делается в некоторых случаях. Для этого мы используем magic команду

```
%matplotlib inline
```

Дальше мы берем наш *DataFrame*, колонку y и делаем какой-то вызов для визуализации. Например, вызов *hist*, который изображает гистограмму нашего распределения, где высота столбика показывает количество наблюдений с таким значением, которое показано по x.

```
df.y.hist()
```

Вы можете делать различные вызовы в *pandas*, они позволяют визуализировать данные с помощью гистограммы, или точек, или разделений. Но все эти способы заимствованы из библиотеки *Matplotlib*. Для того чтобы начать работать с библиотекой *Matplotlib*, нам нужно её импортировать, но мы импортируем не саму библиотеку, а ее модуль *pyplot*. Модуль *pyplot* позволяет работать с библиотекой *Matplotlib* в оперативном стиле. Мы говорим, что нужно изобразить, какие атрибуты изменить, и всё это применяется к текущему изображению.

```
import matplotlib.pyplot as plt
```

Самое простое что мы можем сделать - это взять датасет, допустим, мы будем изображать успешность игрока в зависимости от номера попытки. У нас есть экспонециальное распределение с параметром x, и пусть будет 20 попыток. И у нас будет второе экспонециальное распределение с параметром 6 и размером 20. Попробуем изобразить какое-то из них.

```
data1 = exponential(5, 20)
data2 = exponential(6, 20)

plt.plot(data1)
```

Второй метод изображения данных - это не непрерывная линия, а точка, где у нас есть кординаты x и y. Для этого есть метод *scatter* и для него мы должны передать два объекта.

```
plt.scatter(range(len(data2)), data2)
```

Мы увидели два графика на одном изображении. Во-первых, они одного цвета, поэтому непонятно о чем они говорят. Во-вторых, нет никаких подписей ни у осей, ни у всего графика. И также мы не видим никакой информации о том, что это номера попыток, потому что ось X изображена не целыми числами, а дробными. Мы можем исправить все эти недостатки с помощью **настройки нашего графика**.

Начнем с **цветов**. Мы можем указать у второго игрока цвет не синий, а какой-то конкретный, например, укажем красный цвет.

```
plt.scatter(range(len(data2)), data2, color='red')
```

Теперь давайте дадим **название всему графику**. Для этого есть метод *title*.

```
plt.scatter(range(len(data2)), data2, color='red')
plt.title('Results')
```

Практически любому текстовому объекту *Matplotlib* вы можете передать параметр *fontdict*, который представляет из себя словарь, с некоторыми атрибутами вашего текста. Например

```
plt.title('Results', fontdict={'fontsize': 20})
```

Теперь давайте **подпишем оси**

```
x.label('Attempt number')
y.label('Result')
```

Оба из них также настраиваются. Можно их развернуть, указать цвет шрифта, размер и многое

другое.

Далее было бы неплохо понять, какой из этих графиков отображает значение какого игрока. Для этого мы можем вызвать метод *legend*, и он отобразит легенду. **Легенда** - это описание того, что изображено на вашем графике и что чем кодируется.

```
plt.plot(data1, label='First player')
plt.scatter(range(len(data2)), data2, color='red', label='Second player')
plt.title('Results', fontdict={'fontsize': 20})
x.label('Attempt number')
y.label('Result')
plt.legend()
```

Легенда автоматически располагается там где меньше символов данных.

И давайте сделаем так, чтобы у нас наши **заметки на оси** X были только целочисленными. Для этого воспользуемся методом *xticks*

```
plt.xticks(range(0, 20, 4))
```

Для того чтобы **сохранить наше изображение**, не только в IPython Notebook, а передать его кому-то еще, мы можем сохранить его в файл. Для этого есть метод *savefig*. Важно указать здесь правильное расширение, потому что от расширения файла зависит его формат: если мы сохраним его как *results.png*, это будет *png*-картинка. Если это будет *.pdf* то это будет *pdf*-файл, в зависимости от нужд вы можете сохранять его в разных форматах.

```
plt.savefig('results.pdf')
```

1.3 Расширенная визуализация с Matplotlib

Давайте познакомимся с библиотекой *Matplotlib* еще глубже, поймем, из чего она состоит, и как мы можем изображать сразу несколько графиков на одном изображении.

```
from numpy.random import exponential
import matplotlib.pyplot as plt
%matplotlib inline
```

Для того чтобы понять, из чего состоит библиотека *Matplotlib*, начнем с самой базовой функции *subplots*. Она возвращает нам кортеж из двух объектов *fig* и *axes*. И если мы ее вызываем, мы получаем пустой график с двумя осями X и Y. Давайте разберемся, из чего он состоит.

```
fig, axes = plt.subplots()
```

У нас есть *figure* — это всё наше всё изображение. В нем может быть один или больше *axes*. *Axes* — это холст, по сути, на котором мы изображаем наши графики. И в одном *axes* мы можем строить сразу несколько графиков. У axes есть две оси - это ось X и ось Y. И мы можем каждому *axes* делать свой заголовок, свои подписи к осям, свои лимиты у осей и множество других вещей.

Что можно делать с объектом *axes*? Мы берем *axes* и вызываем какой-то метод, например, *plot*, так же, как мы делали прежде, только мы вызываем *plot* не самого модуля *pyplot*, а конкретного

объекта *axes*, тем самым мы говорим, что мы хотим **нарисовать не где-то, а вот конкретно на этом холсте**. И говорим, что мы хотим нарисовать.

```
axes.plot(exponential(5, 20))
```

Если мы хотим **изменить какие-то свойства у этого холста**, мы используем *set_ title.*

```
axes.set_tiltle('Chart')
```

Теперь **мы более явно понимаем, что мы не рисуем просто в некоем пространстве, а мы рисуем конкретно на этом холсте**, и мы можем менять его свойства. У этого *axes* мы можем делать почти все то же самое, что мы делали до этого. Мы можем менять цвета, можем сказать *set_ xlim*, например, и многое-многое другое. Пока нет особых плюсов, но плюсы появятся, когда сейчас решим рисовать не один график, или даже не один график на одном холсте, а несколько холстов, где каждый будет иметь свои координаты и свои настройки.

Для того чтобы **нарисовать несколько графиков**, нам нужно указать, сколько мы хотим строк у наших графиков и сколько колонок. Для этого мы вызовем всё так же метод *subplots* и дадим два параметра. Первый — это *nrows*, сколько строк мы хотим, а второй — это *ncols*.

```
fig, axes = plt.subplots(nrows=2, ncols=3)
```

Если вам кажется, что размер вашего изображения слишком маленький, вы можете делать *figsize* (размеры указываются в дюймах)

```
fig, axes = plt.subplots(nrows=2, ncols=3, figsize=(10, 5))
```

Давайте теперь начнем **заполнять каждый из 6 графиков**. Как это сделать? Для этого нам нужно использовать объект *axes*, в данном случае это уже не один объект, какой-то холст, а это список списков холстов. И его структура соответствует тому, как выглядит ваше изображение. В данном случае это два списка по три элемента в каждом списке. Давайте изобразим на каждом холсте свой график и дадим ему свое название.

Чтобы лучше разобраться в индексах, название будет содержать в себе название колонки и номер строки. Для этого мы будем идти по нашему списку списков *axes* и на каждой из осей что-то рисовать.

```
for row, row_axes in enumerate(axes):
    for column, ax in enumerate(row_axes):
        ax.plot(exponential(column, 20))
        ax.set_title('Canvas column {} row {}'.format(column+1, row+1))
```

Посмотрим, что получается. Мы сразу видим, что у нас не помещается всё удобно, и чтобы это исправить, мы можем объекту *fig*, который содержит все наши холсты, сказать ему, чтобы он их **распределил таким образом, чтобы они все помещались**.

```
fig.tight_layout()
```

Таким образом, мы можем идти по всем этим холстам и на каждом из них делать самые разные вещи. Мы можем рисовать на них разные графики, можем у каждого из них делать

свой набор осей, можем делать свои подписи осей и свои заголовки. Это позволяет нам детально настраивать то, как мы изображаем каждый из этих графиков. И в результате, когда мы все это нарисовали, если мы хотим что-то сохранить, мы все так же используем объект *fig* и говорим *savefig*, и указываем, куда его сохранить. И у нас появится картинка с шестью графиками.

1.4 Визуализация с pandas

Давайте познакомимся с тем, как мы можем использовать библиотеку *Matplotlib* для визуализации данных, хранящихся в наших *pandas* датафреймах. Мы будем использовать датасет *titanic*, который хранит информацию о пассажирах «Титаника», о их признаках и о факте того, выжили они в этой катастрофе или нет.

```
import pandas as pd
df = pd.read_csv('titanic.csv')
df.head()
```

Исследование любого датасета стоит начать с того, чтобы посмотреть какие-то свойства распределений тех или иных колонок. Начнем с того, что посмотрим, какие были стоимости у билетов и какое количество пассажиров владело какими билетами. Для этого берем наш *DataFrame*, берем поле *Fare*, в котором хранится информация о стоимости, и сейчас мы воспользуемся стандартным методом библиотеки *pandas* — атрибутом *plot*. Атрибут *plot* есть у колонок и у всего *DataFrame*. Он содержит в себе набор методов, которые позволяют вызвать ту или иную визуализацию из арсенала библиотеки *Matplotlib*. Например, мы хотим посмотреть гистограмму распределения. Просто вызываем *hist* и не забываем указать директиву *%matplotlib inline*, для того чтобы наша визуализация была отражена внутри нашего Jupyter Notebook.

```
%matplotlib inline
df.Fare.plot.hist()
```

Мы видим гистограмму. Гистограмма берет наши данные, распределяет их по какому-то количеству интервалов значений и считает количество наблюдений, которые попали в тот или иной интервал. Мы видим, что больше всего наблюдений, или пассажиров, купили самые дешевые билеты. Мы можем **детализировать наши представления, указав количество колонок**, например, 20, или уменьшить детализацию, указав меньше колонок.

```
df.Fare.plot.hist(bins=20)
```

Мы также можем воспользоваться другими методами визуализации, например, **построить плотностное распределение этого значения**. Для этого вызовем у атрибута *plot* метод *kde*.

```
df.Fare.plot.kde()
```

Мы можем строить **распределение не какой-то одной величины, а между разными величинами**. Самое простое — это *scatter plot*. Для этого нам нужно вызывать метод уже не у конкретной колонки, а у всего *DataFrame*. Естественно, нам нужно принять какие-то два значения

x и y. В виде x мы возьмем, например, все тот же *Fare*, а в виде y мы возьмем факт того, выжил пассажир или нет, атрибут *Survived*.

```
df.plot.scatter(x='Fare', y='Survived')
```

Мы видим два распределения, которые схожи между собой, но чуть отличаются. Мы видим, что как будто бы сверху в *Survived* чуть больше людей с большей ценой билета, а внизу - с чуть меньшей ценой. Давайте посмотрим, как еще мы можем визуализировать эти данные, чтобы подтвердить такое наше предположение или как-то его опровергнуть. Для этого давайте вспомним, что в *pandas* есть метод *groupby*, который позволяет группировать данные по какому-то признаку, и построим два графика: для тех, кто *Survived* и не *Survived*. Для этого мы вызываем метод *groupby* и говорим, что хотим сгруппировать по факту выживания, *Survived*. У всего этого мы получаем группы и говорим, что мы хотим взять *Fare*, и вызываем метод *plot.kde*, чтобы нам нарисовалось распределение наших данных.

```
df.groupby('Survived').Fare.plot.kde()
```

Видим два графика, которые чуть отличаются. Но

- непонятно, какой из них отвечает за *Survived*, какой — нет;
- слишком странный масштаб, есть минусовые значения.

Давайте как-то это исправим.

Для этого вспоминаем, что *Matplotlib* поддерживает два метода работы с ним. Один — это **процедурный**, более простой, и второй — **объектно-ориентированный**. Воспользуемся **процедурным**. Для этого нам нужен теперь сам модуль *Matplotlib*, импортируем его, как мы делали это прежде.

```
import matplotlib.pyplot as plt
```

И после того как мы вызвали отрисовку нашего kde, мы хотим видеть **легенду**.

```
df.groupby('Survived').Fare.plot.kde()
plt.legend()
```

Вот, на графике, который нарисовал *Matplotlib* из *DataFrame*, у нас появилась легенда, показывающая нуль — это синие, это невыжившие люди; 1 - оранжевые, выжившие.

Также давайте **изменим наш диапазон значений** по x. Скажем *xlim*, чтобы он начинался как минимум с 0, а заканчивался, не учитывая весь огромный хвост, 200.

```
plt.xlim(0, 200)
```

Теперь давайте посмотрим, как можно работать в **объектном** режиме. Во-первых, если мы вызовем какой-либо из методов атрибута *plot*, например, все тот же *hist*, видим, что он возвращает нам на самом деле уже объект *axes*, с которым мы уже работали. Мы можем его сохранить в переменную и работать с ней так, как мы уже работали с объектами *Matplotlib*. Например, мы вызовем *set_title*, и у нас появился **заголовок у нашего графика**. Это можно было делать и в процедурном стиле, но можно делать и таким образом.

```
ax = df.Fare.plot.hist()
ax.set_title('Visualization')
```

Также мы теперь можем **сохранить изображение**.

```
ax.figure.savefig('something.png')
```

Что еще нам позволяет делать знание о том, что у нас есть объект *axes*? Оно позволяет нам нарисовать визуализации из *pandas* в какие-то уже другие холсты, которые могли быть сделаны нами. Как мы это сделаем? Следующим образом.

Мы берем и сами создаем наши *figure* и *axes*, как мы это делали прежде, указав метод *subplots*

```
fig, ax = plt.subplots(figsize=(10,5))
```

И после этого, когда мы вызываем у атрибута *plot* метод отрисовки чего-то, например, *kde*, мы говорим ему, на каком из *axes* ему нужно рисовать.

```
df.Survived.plot.kde(label='A11', ax=ax)
for label, class_df in df.groupby('Pclass'):
    print(label)
    class_df.Survived.plot.kde(ax=ax, label=label)
plt.legend()
```

Давайте посмотрим, что получилось. Вот мы нарисовали множество графиков, а именно четыре, которые показывают, выжил человек или не выжил, и учитывают распределение по классам кабин.

1.5 Интерактивная визуализация с plotly

Мы научились делать визуализации, которые были статичными. Мы брали данные, рисовали график, получали картинку.

Теперь мы познакомимся с библиотекой визуализации данных *plotly*, которая позволяет делать интерактивные визуализации.

Для этого она использует браузер и использует *JS*-библиотеку - тоже *plotly*, которая позволяет делать визуализацию, с которой вы можете взаимодействовать. Со стороны *Python* вам понадобится тоже библиотека *plotly*, которую нужно установить, и её можно использовать в двух режимах. Один режим — это **онлайн-режим**, когда у вас используется сервер *plotly*, вам нужно получить логин, пароль и это позволяет вам сохранять ваше изображение в облаке и иметь к нему доступ в любой момент с любого устройства.

```
import plotly.plotly as plt
```

Второй способ — это **оффлайн-режим**, когда вы используете визуализацию и библиотеку визуализации, которая у вас есть локально на вашей машине, и вы можете все визуализировать у себя в браузере и, возможно, потом сохранять в облако дополнительной кнопкой.

```
import plotly.offline as offline
```

Мы будем работать с *plotly* оффлайн, и для создания визуализации импортируем все графы объекта

```
from plotly.graph_objs import *
```

Графы объекта — это те объекты, которые вы используете, чтобы создавать визуализации. Также для того чтобы визуализировать данные через *plotly* или IPython Notebook, нам нужно вызвать метод *init_notebook_mode*. Что он делает? Он берет весь *js* и все *css* стили, которые нужны библиотеке для визуализации, и встраивает их в ваш IPython Notebook.

```
offline.init_notebook_mode()
```

У него еще есть параметр — *connected*, который может быть *True* или *False*. По default он *False*, и это означает, что используется *JS*-библиотека из *plotly package Python*, и поэтому вам не нужен вообще Интернет, чтобы работать. Во втором случае, если *connected* будет *True*, то ваш IPython Notebook будет меньше весить; и будет использоваться в real-time и скачиваться с сайта *plotly* вся *JS*-обвязка.

Что нам делать дальше? Нам нужно скачать какой-то dataset. Мы будем использовать *pandas* для манипуляции с dataset. И возьмем dataset, который предоставляет *plotly*. Этот dataset, который рассказывает нам о данных GDP, то есть о валовом продукте, и продолжительности жизни в разных странах, на разных континентах: в Америке и в Европе. И мы его подготовим для дальнейшей визуализации, отсортировав dataset по GDP, по валовому продукту.

```
import pandas as pd
df = pd.read_csv('https://raw.githubusercontent.com/yankev/test/master/
                life-expectancy-per-GDP-2007.csv')
df.sort_values('gdp_percap', inplace=True)
df.head()
```

Для **создания визуализации** в *plotly* используется такое понятие, как *trace*. *Trace* — это визуализация какого-то одного распределения данных. В нашем случае мы будем делать *trace* в виде *scatter plot*, то есть точек или линий. И для этого мы используем объект *scatter*.

```
trace = Scatter(x=df.gdp_percap, y=df.lif_exp)
```

Итак, мы передали два параметра. Что нам нужно сделать дальше? Дальше нам нужно создать объект *data*, который инкапсулирует в себе все те графики, которые мы хотим нарисовать. В данном случае у нас всего один график, один *trace*. Поэтому мы делаем массив из одного *trace*. И с этим объектом мы можем идти к *plotly*, в данном случае к оффлайн его версии, и сказать *iplot* что-то, а именно *data*.

```
data = Data([trace])
offline.iplot(data)
```

Как мы видим, получился интерактивный график. Мы наводим на конкретное значение, и даже если у нас всего пять засечек на оси X, мы видим конкретные значения x и y, которые принимают наши данные. Мы также можем выделить часть графика, и увидеть более детальное распределение в каком-то диапазоне. И можем нажать на кнопку «сохранить» изображение или сохранить

его в облаке, где вам будет предложено создать аккаунт, чтобы вы могли делиться вашим изображением.

Сейчас мы изобразили один график. Давайте **сделаем несколько графиков** или несколько *trace*. Для этого чуть обработаем наш датасет. Сделаем следующие вещи: сделаем новую колонку *population*, куда мы извлечем из строки *country* её население. И сделаем колонку *name*, где мы запишем название страны. И также разобьем наш *DataFrame* на два: один, который на континенте Америка, второй, который на континенте Европа.

```
df['population'] = df.country.str.split(':').apply(lambda words: float(words[-1]))
df['name'] = df.country.str.split(':').apply(lambda words: words[1].split('<br>')[0])

americas = df[(df.continent=='Americas')]
europe = df[(df.continent=='Europe')]
```

Отлично, у нас есть два *DataFrame*. Мы создаем два *trace*. Первый — это тот же *scatter*, но не на всем *DataFrame*, а на *americas*. Второй — это то же самое, но на *europe*. Получившиеся два *trace* мы передаем в массив и передаем в объект *data*, чтобы показать две визуализации.

```
trace1 = Scatter(x=americas.gdp_percap, y=americas.life_exp)
trace2 = Scatter(x=europe.gdp_percap, y=europe.life_exp)

data = Data([trace1, trace2])

offline.iplot(data)
```

Запустим и видим два line chart разного цвета и с легендой. Но легенды ничего не говорят нам о том, какой из них чем является. Давайте исправим это, модифицировав наш *trace scatter*. Объект *scatter* и сам *trace* отвечают за то, как выглядит конкретная одна визуализация. Соответственно, нам можно как передать данные, так и можно настроить какие-то параметры, например, можно настроить параметр *name*.

```
trace1 = Scatter(x=americas.gdp_percap, y=americas.life_exp, name='Americas')
trace2 = Scatter(x=europe.gdp_percap, y=europe.life_exp, name='Europe')

data = Data([trace1, trace2])

offline.iplot(data)
```

Наша легенда видоизменилась, у нас теперь написаны нормальные названия двух наших графиков.

Кажется, что line chart не идеально подходит для данной визуализации. Куда проще было бы сделать просто точки, как в *scatter plot*. Для этого мы можем настроить наш *scatter trace*, указав ему **режим, в котором он рисуется**. В данном случае нас интересует режим *markers* и мы сделаем это у обоих графиков, у обоих *trace*.

```
trace1 = Scatter(x=americas.gdp_percap, y=americas.life_exp, name='Americas', mode='markers')
trace2 = Scatter(x=europe.gdp_percap, y=europe.life_exp, name='Europe', mode='markers')
```

```
data = Data([trace1, trace2])

offline.iplot(data)
```

Всего мы видим два набора точек. Но когда мы наводимся на одну из точек, мы не знаем, что это за страна, что не очень удобно, но мы можем это исправить, передав в параметр *text* список из названий, где названия будут названия наших стран, которые также хранятся в нашем *DataFrame*.

```
trace1 = Scatter(
    x=americas.gdp_percap,
    y=americas.life_exp,
    name='Americas',
    mode='markers',
    text=americas.name
)
trace2 = Scatter(
    x=europe.gdp_percap,
    y=europe.life_exp,
    name='Europe',
    mode='markers',
    text=europe.name
)

data = Data([trace1, trace2])

offline.iplot(data)
```

Для того чтобы предоставить еще больше контекста нашей визуализации, мы можем **изменять размер точки в зависимости от какого-то параметра**. Скажем, будем изменять его в зависимости от размера населения. Это поле хранится у нас в *DataFrame*, в ключе *population*. И для изменения этого размера нам нужно указать параметр *marker*.

Мы делаем *marker* следующим образом. Мы говорим: *marker* — это словарь, у которого есть ключ *size*, и этот *size* будет такой же, как в *population*. Но если мы скажем просто *population*, то размеры будут миллионными, что заполонит нам весь график, поэтому мы нормализуем это на максимальное значение — население и умножаем, скажем, на 20, чтобы получить размер наших точек от нуля до 20.

```
trace1 = Scatter(
    x=americas.gdp_percap,
    y=americas.life_exp,
    name='Americas',
    mode='markers',
    text=americas.name,
    marker={'size': americas.population/americas.population.max()*20}
)
trace2 = Scatter(
```

```
    x=europe.gdp_percap,
    y=europe.life_exp,
    name='Europe',
    mode='markers',
    text=europe.name
)

data = Data([trace1, trace2])

offline.iplot(data)
```

Теперь у нас разного размера наши точки. Мы сразу видим, что какая-то страна большая, у нее большой GDP и достаточно высокая продолжительность жизни ожидаемая. А есть какие-то маленькие страны, а какие-то страны совсем маленькие, мы их даже не видим. Если мы увеличим, мы сможем увидеть эти страны. **И это особенность *plotly*, что мы можем таким образом все-таки увидеть больший диапазон значений, чем мы могли бы, если бы у нас была статическая визуализация**. Но в то же время кажется, что указание размера страны как размер нашей точки, не очень хорошее решение, потому что многие точки пропали после этого. Поэтому давайте уберем этот параметр.

Статистический анализ

2.1 Определение вероятности

Вспомним основные понятия и свойства вероятностей и рассмотрим различные примеры.

Представим, что мы подбрасывает кубик шестью гранями. Мы заранее не знаем какое число выпадет, но можем сказать, что всего возможно шесть исходов выпадения от единицы до шести. Нас может интересовать некое событие, например, выпадение 6 или выпадения только нечётных чисел - это 1, 3 или 5. Мы **предполагаем, что кубик симметричный**, поэтому каждое число может появиться с равными шансами.

Разберем другой эксперимент, где будут подбрасываться два кубика. Будем считать кубики различимыми и тогда их можно пронумеровать. Определим пары (i, j), где i - это число выбывших очков на первом кубике и j - это число очков выпавших на втором кубике. В этом испытании всего возможно 36 исходов.

Давайте рассмотрим событие A: на **первом кубике выпадает число 3**. Тогда оно состоит из таких пар (возможных случаев): $A = \{(3,1); (3,2); (3,3); (3,4); (3,5); (3,6)\}$

Другое событие B - это **сумма выпавших очков равна пяти**: $B = \{(2,3); (4,1); (3,2); (1,4)\}$

Вероятность события можно задать как отношение числа благоприятствующих этому событию исходов m, к общему числу всех возможных исходов: $\mathbb{P}(A) = \dfrac{m}{n}$

Вернемся к предыдущим примерам и найдём вероятность этих событий. В первом случае благоприятствуюих исходов всего 6, поэтому $\mathbb{P}(A) = \dfrac{6}{36} = \dfrac{1}{6}$. Во втором: $\mathbb{P}(B) = \dfrac{4}{36} = \dfrac{1}{9}$

Свойства вероятности:

- $0 \leq \mathbb{P}(A) \leq 1$. Событие A, для которого $\mathbb{P}(A) = 0$, называется **невозможным**
- $\mathbb{P}(A) + \mathbb{P}(\bar{A}) = 1$, где $\bar{A}$ - **противоположное** событие

Прежде, чем рассматривать другие свойства вероятности разберем некоторые **операции с событиями**.

- **Пересечение** событий $A \cap B$ - это такое событие, которое состоит в том, что произошли оба события A и B. Иначе говоря, события A, B содержат исходы общие для событий A и B.

Вернемся к примеру с двумя кубиками и рассмотрим пересечение событий A и B. То есть на первом кубике выпала цифра 3 и сумма цифр на двух кубиках равна 5. В данном случае мы видим, что у нас есть один общий исход - пара (3,2). Тогда $A \cap B = \{(3, 2)\}$

- **Объединение** событий $A \cup B$ - это событие, состоящее в том, что произошло хотя бы одно из этих событий. То есть, событие $A \cup B$ включает в себя исходы как события A, так и события B.

Давайте теперь рассмотрим более сложные события. Возьмем две одинаковые монеты и пусть событие A - это выпадения орла на первой монете. Событие B - выпадение орла на второй монете. Тогда событие AB выпадение двух орлов на двух монетах. **На то, что событие B произойдет, событие A никак не влияет**. То есть вероятность события B не зависит от вероятности события A. Этот пример приводит нас к формальному определению **независимости событий**.

События A и B являются **независимыми**, если $\mathbb{P}(AB) = \mathbb{P}(A) \cdot \mathbb{P}(B)$. Также это определение независимости можно расширить и на N событий.

Рассмотрим другой пример событий. Кубик бросается один раз и известно, что выпало более двух очков. Какова вероятность того, что выпало нечётное число очков? Обозначим событие $B = \{3; 4; 5; 6\}$, событие $A = \{3; 5\}$. Нужно **найти вероятность A при условии, что событие B уже случилось**.

Это и есть **условная вероятность**, которая вычисляется так: $\mathbb{P}(A|B) = \dfrac{\mathbb{P}(AB)}{\mathbb{P}(B)} = \dfrac{2}{4} = \dfrac{1}{2}$

2.2 Случайная величина

Познакомимся с определением **случайной величины**. **Случайную величину** можно определить как функцию, которая каждому элементарному исходу ставит в соответствие вещественное число.

Давайте рассмотрим примеры случайных величин. Вернемся к примеру с кубиком и рассмотрим следующие случайные величины. Пусть ξ — это такая случайная величина, равная числу, выпавшему на кубике. $\xi(x) = x$.

Рассмотрим другую случайную величину — η, и она принимает два значения: η равняется единице, если выпало нечетное число, и η равняется нулю, если выпало четное число.

$$\eta(x) = \begin{cases} 1, & x \in \{1; 3; 5\} \\ 0, & x \in \{2; 4; 6\} \end{cases}$$

Далее мы рассмотрим дискретные и непрерывные случайные величины. **Дискретная** случайная величина принимает **конечное или счётное число значений**. Примеры дискрентных случайных величин — количество очков, выпавших при подбрасывании кубика. **Непрерывная** случайная величина принимает **несчётное число значений**, например, серию измерений температуры можно описать непрерывной случайной величиной.

Мы определили случайную величину, и для ее описания не всегда удобно использовать набор вероятностей, и можно использовать функцию распределения. **Закон распределения вероятностей можно представить в виде функции распределения вероятностей случайной**

величины, которая может использоваться как для дискретных, так и для непрерывных случайных величин. Давайте разберемся с тем, что это такое.

Функция распределения — это функция, характеризующая вероятность $F_\xi(x) = \mathbb{P}(\xi < x)$

Свойства функции распределения:

- $0 \leq F_\xi(x) \leq 1$
- $F_\xi(x)$ неубывающая функция

Разберем функцию распределения на примере бернуллиевской случайной величины. Рассмотрим случайное событие с двумя исходами, например, попадание мяча в ворота. Результат можно представить в виде двух значений: попадание — единица, и промах — нуль. Вероятность попадания в ворота, например, равна p, а вероятность промаха, соответственно, равняется $(1-p)$. Таким образом, мы пришли с вами к описанию распределения Бернулли. Случайная величина имеет **распределение Бернулли**, если она принимает только два значения: 0 и 1 с вероятностями p и $(1-p)$ соответственно. Функция распределения в данном случае задается следующим образом.

$$F_\xi(x) = \begin{cases} 0, & x \leq 0 \\ 1-p, & 0 < x \leq 1 \\ 1, & x > 1 \end{cases}$$

Данное распределение описывает случайные эксперименты с двумя исходами: успех и неудача.

Теперь давайте перейдем к **непрерывной** случайной величине. В силу того, что непрерывная случайная величина принимает несчетное число значений, **вероятность того, что случайная величина принимает какое-то конкретное значение, равна нулю**. Тут мы говорим о вероятности того, что значение лежит в некотором интервале. Как найти эту вероятность, мы покажем дальше.

Непрерывную случайную величину можно также задать через плотность вероятности. **Плотность вероятности** — это производная функции распределения. Чтобы посчитать вероятность того, что случайная величина лежит в промежутке $(a; b)$, нужно взять интеграл от плотности распределения случайной величины.

В качестве примера непрерывной случайной величины рассмотрим **равномерное распределение**. Случайная величина на отрезке $[a; b]$ принимает любое значение с равной вероятностью. Тогда плотность вероятности задается как

$$f_\xi(x) = \begin{cases} \dfrac{1}{b-a}, & x \in [a; b] \\ 0, & \text{otherwise} \end{cases}$$

Примером такой случайной величины может быть время ожидания прибытия лифта. Лифт может приехать сразу, тогда время ожидания лифта равно нуль секунд, может приехать через некоторое время, допустим, через 50 секунд. В этом случае $a = 0$ и $b = 50$.

2.3 Показатели центра распределения

Описательная статистика часто предполагает использование нескольких чисел для характеристики данных. В исполМатематическое ожидание — это мера среднего значения случайной величины, которое определяется по формуле. На слайде представлена формула для дискретной случайной величины.ьзовании описательной статистики есть два аспекта. Первое — это определение места расположения центра распределения, так называемое среднее значение. Их может быть несколько. И насколько показатели центра распределения отличаются друг от друга, например, смещенное распределение или нет. **Среднее значение, медиана и мода** — это показатели центра, и они отражают типичное значение у распределения. Напомним, что среднее значение — это то значение, вокруг которого группируются все остальные. Давайте перейдем к их рассмотрению.

Математическое ожидание — это мера среднего значения случайной величины, которое определяется по формуле.

Для **дискретной** случайной величины:

$$\mathbb{E}\xi = \sum_i a_i p_i = \sum_i a_i \mathbb{P}(\xi_i = a)$$

Для **непрерывной** случайной величины:

$$\mathbb{E}\xi = \int_{-\infty}^{\infty} x f_\xi(x) dx$$

На практике для **оценки матожидания выборки**, состоящей из n элементов используется **выборочное среднее**:

$$\bar{x} = \frac{\sum_{i=1}^{n} x_i}{n}$$

Недостатком данной статистики является то, что **при наличии выбросов среднее значение сильно смещается**, как, например, в этом случае: $\{1; 1.5; 2; 2.23; 3; 3.1; 3.8; 30; 31\}$. Выборочное среднее в этом случае равно 8.6, но заметим, что большая часть значений лежит от 1 до 4.

Теперь перейдем к следующей характеристике распределения — медиане. **Медиана** — это такое значение, которое делит пополам распределение. Грубо говоря, медианой случайной величины является такое число, что $F_\xi(mediana) = \mathbb{P}(\xi < mediana) = 0.5$.

Если мы рассматриваем некоторую выборку, то **медиану можно найти следующим способом**. Для начала нужно проверить, чётное или нечётное число элементов в выборке. Если оно нечётное, то необходимо отсортировать массив и взять элемент, находящийся посередине. Если же в выборке чётное число элементов, то медиану можно вычислить как среднее между двумя центральными элементами.

Медиана незначительно смещается при наличии аномальных значений. **Недостаток медианы** в том, что для размерности случайной величины больше, чем 1, она сложна в вычислении.

Следующая характеристика случайной величины, о которой мы поговорим, это мода. **Мода** — это наиболее часто встречаемое значение в выборке.

Например, для данного массива $\{21; 21; 21; 23; 24; 26; 26; 28\}$ мода будет равна 21. Добавим, что **мод может быть несколько**.

И наконец, мы переходим к последней по порядку, но не по значимости, характеристике распределения — дисперсии. **Дисперсия** характеризует разброс случайной величины вокруг её среднего значения, поэтому она вычисляется как $\mathbb{D}(X) = \mathbb{E}[(X - \mathbb{E}(X))^2]$, где $\mathbb{E}(X)$ - это математическое ожидание случайной величины X. Дисперсия имеет размерность квадрата случайной величины. **Иногда удобнее пользоваться значением, размерность которой совпадает с исходной случайной величиной**, поэтому используется корень из дисперсии, который называется **среднеквадратичным отклонением**. $\sigma(X) = \sqrt{\mathbb{D}(X)}$.

Если же дана выборка, то по ней можно вычислить **выборочную дисперсию**

$$S = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

Мы уже рассмотрели основные статистики распределения. Давайте теперь разберем, как они соотносятся друг с другом в зависимости от формы распределения. Распределение можно поделить на два типа — **симметричное и смещенное**.

В симметричном распределении все три показателя совпадают: среднее равно медиане, равно моде.

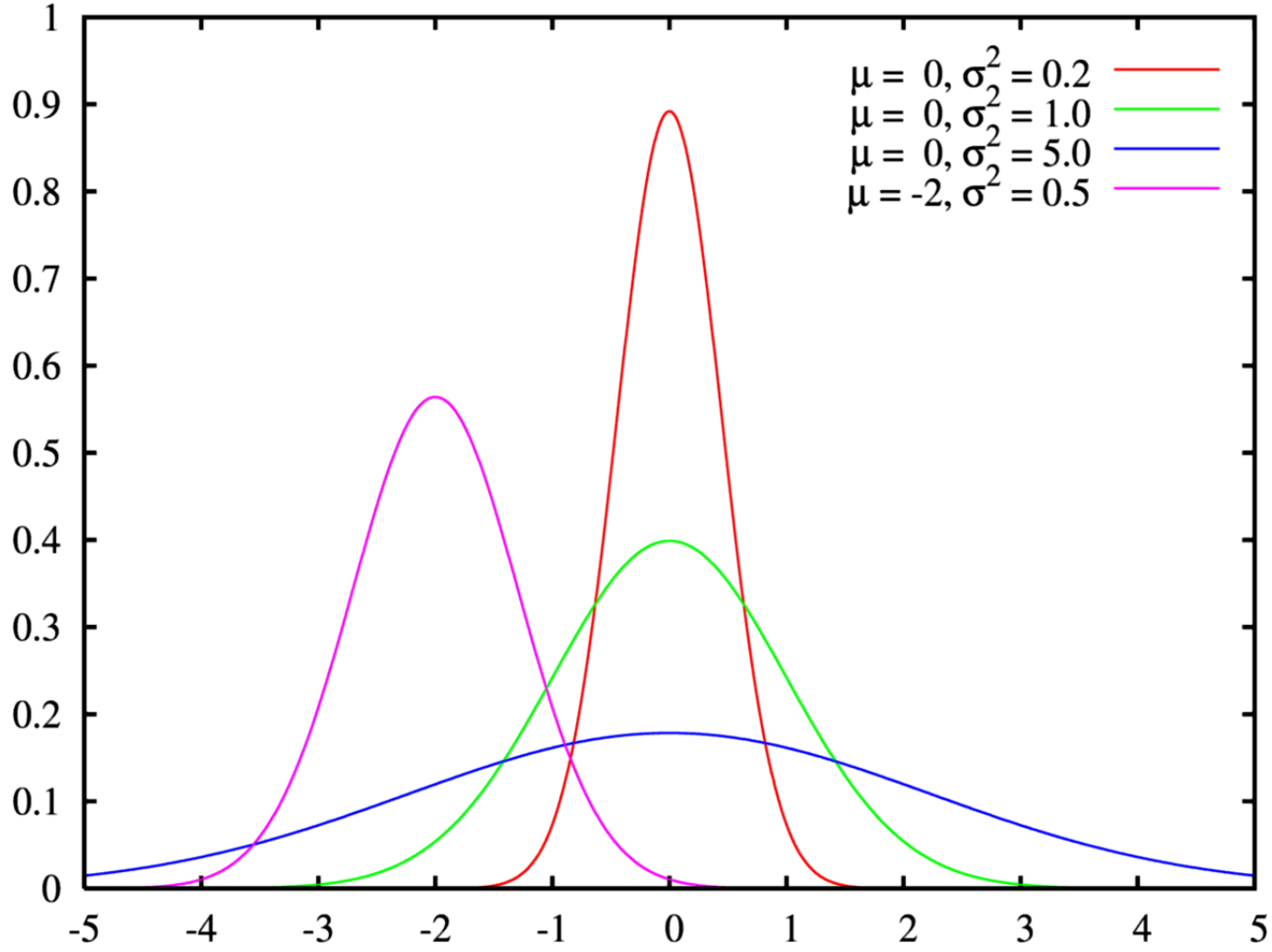
Асимметричное распределение можно разделить на два типа: со смещением влево и со смещением вправо.

2.4 Нормальное распределение

Рассмотрим нормально распределенную случайную величину. В качестве примера можно взять спортсмена, у которого регулярно собирается большое количество показателей, например, вес, пульс, уровень сахара в крови. Эти показатели могут изменяться, но в среднем колеблются около некоторого значения. Такие процессы можно описать **нормальной или гауссовой случайной величиной**. Функция плотности вероятности выглядит следующим образом

$f(x|\mu\sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}}\, e^{\frac{-(x-mu)^2}{2\sigma^2}}$, где μ - математическое ожидание, σ - среднеквадратичное отклонение.

Давайте рассмотрим, как выглядит график нормального распределения. Ось X отвечает за значение случайной величины, ось Y представляет значение плотности вероятности, **максимум функции приходится на среднее значение μ и распределение является симметричным относительно μ**. Наиболее вероятное значение случайной величины расположено близко к пику, и чем менее вероятно значение, тем дальше оно находится от пика. **Большее значение σ показывает больший разброс значений от среднего**. Меньшее значение σ, соответственно, показывает, что значение случайной величины сгруппировано вокруг среднего.



Возьмем интервал значений $(\mu - \sigma; \mu + \sigma)$. Мы знаем, что вероятность попадания в интервал равна площади под графиком. Мы опустим вычисления и перейдем к результату. В итоге мы получим вероятность попадания в этот интервал, равную $\mathbb{P}(-\sigma \leq x - \mu \leq \sigma) = 0.68$. Возьмем другой интервал: $(\mu - 2\sigma; \mu + 2\sigma)$. Для этого интервала мы получим вероятность, равную $\mathbb{P}(-2\sigma \leq x - \mu \leq 2\sigma) = 0.95$. **А для интервала** $(\mu - 3\sigma; \mu + 3\sigma)$ **вероятность равна почти 1!** $\mathbb{P}(-3\sigma \leq x - \mu \leq 3\sigma) = 0.997$ - то есть почти все значения лежат в этом интервале.

Давайте рассмотрим частный случай нормального распределения, когда $\mu = 0$ и $\sigma = 1$. Это распределение называется **стандартным нормальным распределением**. Нормальное распределение с ненулевым значением μ можно превратить в стандартное с помощью преобразования **z-score**.

Разберем формулу. Сначала мы центрируем случайную величину, вычитая μ, а затем нормируем на стандартное отклонение: $z = \dfrac{x - \mu}{\sigma}$

Зачем необходимо данное преобразование? **z-преобразования** часто используются, чтобы все наблюдения перевести в **z-шкалу** для упрощения работы с данными.

Например, для некоторой нормальной случайной величины необходимо вычислить вероятность попадания в некоторый интервал. Для этого нам необходимо вычислить интеграл от плотности вероятности, который просто так не вычисляется. Применяя преобразование, мы получаем новые значения интервала, вероятности для которых уже вычислены, и мы уже оперируем посчитанными значениями.

2.5 Центральная предельная теорема

Поговорим про центральную предельную теорему и рассмотрим, почему она является одной из ключевых теорем в статистике. Давайте рассмотрим некоторую выборку размера n $\{x_1, \ldots, x_n\}$ из распределения $F(x)$. По данной выборке мы можем вычислить выборочное среднее $\bar{x} = \frac{\sum_{i=1}^{n} x_i}{n}$. **Какое распределение может иметь выборочное среднее?**

Наберем выборку размера n из распределения $F(x)$, а затем вычислим выборочное среднее. Таких выборок нужно набрать достаточно много, чтобы можно было построить гистограмму. Заметим, что при увеличении размера выборки распределение все больше похоже на нормальное, становится более гладким и симметричным. Так мы приходим к центральной предельной теореме.

Центральная предельная теорема утверждает, что выборочное среднее суммы достаточно большого количества независимых одинаково распределенных случайных величин будет иметь нормальное распределение. При этом сами случайные величины могут изначально принадлежать другому распределению. Это распределение может быть как дискретным, так и непрерывным.

Почему центральная предельная теорема так важна? Зачастую очень сложно оценить среднее по всей совокупности, но мы можем это сделать, полагаясь на некоторую выборку, и чем больше выборок мы рассмотрим, тем точнее будет наша оценка.

2.6 Зависимость между случайными величинами

В общем виде для определения зависимости между случайными величинами необходимо найти их совместное распределение. На практике это не всегда возможно, но мы можем оценить зависимость, оперируя математическим ожиданием. **Если математическое ожидание произведения случайных величин не равняется произведению матожиданий, то можно сказать, что случайные величины зависимы**. Это позволяет обойти поиск совместного распределения.

Давайте теперь перейдем к ковариации. **Ковариация** является одним из простых способов проверить зависимость случайных величин. Ковариация показывает, насколько две случайные величины отличаются друг от друга.

$$cov(\eta, \xi) = \mathbb{E}[(\eta - \mathbb{E}\eta)(\xi - \mathbb{E}\xi)] = \mathbb{E}(\eta\xi) - \mathbb{E}(\eta)\mathbb{E}(\xi)$$

Если ковариация не равна 0, то можно предположить, что случайные величины зависимы.

Но у ковариации есть недостатки. Отметим, **ковариация не является безразмерной величиной**. Допустим, что x — это расстояние, проезжаемое автомобилем в километрах, y — общий запас топлива в баке в литрах, тогда размерность ковариации будет километры умножить на литры. Это не позволяет оценить силу зависимости между случайными величинами. Поэтому **нужно нормировать ковариацию, чтобы представить силу зависимости между ними**.

Итак, следующая величина есть нормированная ковариация - это **корреляция**.

$$\rho(\eta, \xi) = \frac{cov(\eta, \xi)}{\sqrt{\mathbb{D}\eta}\sqrt{\mathbb{D}\xi}}$$

- Корреляция **больше 0** — случайные величины положительно зависимы, то есть при увеличении одной другая также увеличивается.
- Корреляция **равна 0** — величины независимы.
- Корреляция **отрицательна** — при увеличении одной другая случайная величина уменьшается пропорционально.

Отметим важный момент. **Если две случайные величины независимы, то ковариация равна 0**. Обратное утверждение не всегда верно.

2.7 Распределение Стьюдента

Давайте представим ситуацию, в которой у нас уже есть некоторая выборка, и мы хотим смоделировать нормальное распределение, но по некоторым причинам мы **не можем вычислить среднеквадратичное отклонение**, например, выборка слишком маленькая. Пусть $\{x_1, \ldots, x_n\}$ — некоторая выборка из нормального распределения. Мы не знаем, чему равно квадратичное отклонение, но можем найти **выборочное среднее**

$$\bar{x} = \frac{\sum_{i=1}^{n} x_i}{n}$$

и **исправленную выборочную дисперсию** по выборке

$$S_0^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2$$

Тогда случайная величина $t = \dfrac{\bar{x} - \mu}{S_0/\sqrt{n}}$ имеет распределение Стьюдента с $(n-1)$ степенями свободы, где n — это размер выборки, то есть $t \in St(n-1)$. Можно заметить, что формула очень **похожа на z-преобразование**. Мы не знаем истинное значение σ, поэтому мы заменяем его на выборочное значение.

График плотности распределения Стьюдента, как и нормального распределения, является симметричным и колоколообразным, но с более широкими хвостами. Чем выше степень свободы, то есть размер выборки, тем ближе распределение Стьюдента к нормальному. Нормальное распределение и распределение Стьюдента являются одними из основных в

статистике и машинном обучении. Если размер выборки невелик, или квадратичное отклонение неизвестно, то можно заменить Гауссово распределение Стьюдентом.

2.8 Статистика в SciPy

Нам понадобятся библиотеки *NumPy* и *SciPy*, а точнее понадобится модуль *stats*.

```
import numpy as np
from scipy import stats
import matplotlib.pyplot as plt
%matplotlib inline
plt.style.use('ggplot')
```

Для начала мы научимся создавать дискретные и непрерывные распределения уже известного типа. Но прежде чем переходить конкретно к ним, давайте посмотрим как в самом модуле реализованы классы, отвечающие за случайные величины.

В *stats* есть два класса - *rv_continuous* и *rv_discrete*, в которых реализованы все методы, относящиеся к случайным величинам. Можно посчитать функцию распределения, подсчитать плотность вероятности в случае непрерывной случайной величины, различные статистики - в общем, все необходимые методы уже реализованы в этом классе. Для создания распределения известного типа необходимо создать класс который наследуется от этих двух классов. Если же необходимо создавать свои случайные величины, то можно наследоваться от этих классов и их использовать.

Давайте перейдем к дескретным распределениям и начнем с распределения Бернулли. Напомню, что случайная величина имеет **распределение Бернулли**, если она принимает только два значения: нуль или один с некоторой вероятностью. **Для создания случайной величины нам необходимо указать соответствующий класс, он называется так же как случайная величина, и задать этот единственный параметр**.

```
rv_bernoulli = stats.bernoulli(p=0.3)
```

Далее инициализируется объект и уже можно из этого объекта создавать выборки. Для этого мы можем только задать размер выборки и получим массив.

```
rv_bernoulli.rvs(14)
```

Давайте теперь перейдем к биномиальному распределению. **Биномиальное распределение** - это обобщение распределения Бернулли. Только мы рассматриваем не один эксперимент, а n экспериментов с вероятностью успеха p. И теперь для создания уже объекта, отвечающего за биноминальное распределение, необходимо указать два параметра: количество экспериментов и вероятность.

```
rv_binom = stats.binom(100, p=0.9)
```

Также, мы можем из этого распределения **сгенерировать некоторую выборку**.

```
rv_binom.rvs(8)
```

Давайте теперь попробуем нарисовать гистограмму по данной выборке.

```
plt.hist(rv_binom.rvs(80), bins=10)
```

По оси X у нас откладываются значения из нашей выборки, по оси Y откладывается то количество раз, сколько данное значение встретилась в выборке.

Давайте теперь рассмотрим непрерывное распределение. И начнем с равномерного. Напомню, что **равномерное распределение** - это когда случайная величина определена на некотором отрезке, и она встречается с одинаковой вероятностью. А за пределами этого отрезка вероятность встретить случайно величину равна нулю.

Для того чтобы создать случайную величину равномерного распределения, нам необходимо указать старт отрезка, а также на сколько единиц нам нужно сдвинуться от стартовой точки. То есть, **мы передаем не начало и конец, а именно старт и смещение**.

```
a = 5
b = 10

rv_uniform = stats.uniform(a, b - a)
```

Теперь, после того как мы создали объект случайной величины, мы можем **посчитать функцию распределения в некоторой точке**. Если мы берем значения из нашего интервала, то функция распределения не будет равна нулю. Если мы возьмем значение, которое не попадает в наш интервал, то значение функции распределения будет равно либо единице, либо нулю.

```
rv_uniform.cdf(5.5)
```

Аналогично с **плотностью вероятности**. Значение внутри интервала - плотность вероятности не равна нулю. Значение вне интервала - тогда плотность вероятность будет равна нулю.

```
rv_uniform.pdf(7)
```

Давайте теперь рассмотрим как выглядит **график функции распределения**. Для этого нам необходимо создать некоторые значения, которые мы передаем в нашу функцию распределения, и в итоге мы получаем массив значений функции распределения и далее его строим.

```
X = np.linspace(a - 2, b + 2, 100)\n
cdf = rv_uniform.cdf(X)
plt.plot(X, cdf)

plt.ylabel('F(x)')
plt.xlabel('x')
plt.ylim([0,1.5])
plt.title(u'Cumulative distribution function for uniform')
```

Здесь отчетливо видно, что наша случайная величина определена на отрезке $[5; 10]$. И за его пределами значение либо нуль, либо единица.

Давайте теперь посмотрим, как выглядит **график плотность вероятности**. Так как, плотность вероятности - это производная от функции распределения, то на определенном отрезке плотность вероятность постоянна.

```
X = np.linspace(a-2 , b + 2, 1000)
pdf = rv_uniform.pdf(X)
plt.plot(X, pdf)

plt.ylabel('f(x)')
plt.xlabel('x')
#plt.ylim([0,1])
plt.xlim([4, 12])
plt.title(u'PDF for uniform')
```

Давайте теперь перейдем к важному распределению статистики - **нормальному распределению**. Для его создания нам необходимо задавать два параметра: μ - среднее значение и σ - среднеквадратичное отклонение. Для создания объекта нормальной величины нам необходимо определить такие аргументы, как *loc*, отвечающий за среднее значение и *scale* - это будет наша сигма.

```
mu = 2
sigma = 0.5

rv_norm = stats.norm(loc=mu, scale=sigma)
```

А также мы можем из этой случайной величины **сделать sample данных**.

```
rv_norm.rvs(17)
```

И давайте теперь посмотрим, как уже для нормальной величины будет выглядеть **функция распределения**.

```
x = np.linspace(0, 4, 100)
cdf = rv_norm.cdf(x)
plt.plot(x, cdf)

plt.ylabel('F(x)')
plt.xlabel('x')
```

В данном случае, как мы видим, характер функции распределения не меняется, она также определена от нуля до единицы и не убывает, но график уже более сглаженный.

И давайте теперь посмотрим, как выглядит **плотность вероятности**.

```
x = np.linspace(0,4,100)
pdf = rv_norm.pdf(x)
plt.plot(x, pdf)

plt.ylabel('f(x)')
plt.xlabel('x')
```

Так как, нормальное распределение симметрично, то и график соответственно симметричный. Так же мы можем поиграться с параметрами нормального распределения, например, зафиксируем μ и будем менять среднеквадратичное отклонение. И посмотрим, как плотность вероятности будет

меняться. Если значение σ маленькое, то график более узкий, и чем больше значение сигма, тем график выглядит более широким.

Помимо того, что можно вычислять функцию распределения плотность вероятности, можно использовать библиотеку *stats* для **вычисления различных статистик**. Для этого мы у нашей величины можем вызвать метод *stats* и выбрать соответствующие моменты. То есть, мы можем посчитать среднее, дисперсию и параметр смещённости (то есть, если распределение асимметричное, например смещение влево или вправо, то этот параметр не будет равен нулю).

```
mean, var, skew = rv_norm.stats(moments='mvs')
```

2.9 Доверительный интервал

Зачем нужен доверительный интервал? **Доверительный интервал** — это показатель точности измерений. Его применяют как для оценки среднего, так и для оценки дисперсии. Он также отражает, насколько величина, подсчитанная по выборке, отражает истинное значение по некоторой генеральной совокупности. Мы рассмотрим, как можно построить доверительный интервал для среднего, для нормального распределения. Также посмотрим, что делать, если неизвестна дисперсия. И самое главное — что делать, если дана выборка, но непонятно из какого распределения. Давайте перейдем к определению доверительного интервала.

Вообще говоря, **доверительный интервал** определяется через вероятность того, что оцениваемый параметр (это может быть среднее или дисперсия) не выходит за границы. Эта вероятность — некоторый уровень доверия. То есть это вероятность того, что доверительный интервал содержит точные значения. Обычно берут уровень доверия, равный 0.95, либо 0.99.

$\mathbb{P}(LB \leq \theta \leq RB) = p$, где p - уровень доверия, левую и правую границы LB и RB нам нужно найти.

Давайте перейдем к **нормальному распределению в случае известной дисперсии**. В этом случае формула доверительного интервала выглядит следующим образом. Нам необходимо найти границы, в которых находится среднее значение.

$\mathbb{P}(\bar{X} - z_{1-\frac{\alpha}{2}}\frac{\sigma}{\sqrt{n}} \leq \mu \leq \bar{X} + z_{1-\frac{\alpha}{2}}\frac{\sigma}{\sqrt{n}}) = 1 - \alpha$, где $\bar{X}$ - выборочное среднее, σ — известная дисперсия по генеральной совокупности, то есть по всему распределению, n — размер выборки.

И здесь непонятным остается параметр z. Сейчас мы его разберем. Давайте взглянем на график нормального распределения. Как мы помним, существует **правило двух и трех сигм**, что 95% значений содержатся в пределах $\pm 2\sigma$, и 99% значений содержатся в интервале $\pm 3\sigma$. И давайте посмотрим, чему же соответствуют эти интервалы. Мы можем также найти функцию распределения, которая по сути накапливает значения. То есть если нам необходимо выбрать уровень доверия, равный 0.95, то соответствующее значение функции распределения будет близко к 97%. Далее мы вычисляем некоторый **z-score**, основываясь на функции распределения. И уже эти значения будут задавать границы интервала.

Давайте теперь посмотрим, как мы это сможем вычислить, используя библиотеку *SciPy*. Итак, мы зафиксировали некоторое исходное распределение с некоторыми параметрами μ и σ.

```
population = stats.norm.rvs(loc=2, scale=5, size=100000)
```

Далее мы сделаем sample из этого распределения. Для этого можно воспользоваться методом *np.random.choice*, где мы передаем исходное распределение и указываем, какой размер выборки мы хотим получить.

```
sample_size = 100
sample = np.random.choice(a = population, size = sample_size)
```

Давайте дальше найдем **выборочное среднее и дисперсию по генеральной совокупности**.

```
sample_mean = sample.mean()
st_dev = population.std()
```

И затем мы можем **подсчитать соответствующее z-value**.

```
z_value = stats.norm.ppf(q = 0.975)
print("z-value:", z_value)
```

Заметим, что **нормальное распределение симметрично, и поэтому нам достаточно найти z-value с одного конца, потому что в силу симметричности левая граница будет равна правой, но с противоположным знаком**.

```
z_value = stats.norm.ppf(q = 0.025)
print("z-value:", z_value)
```

Здесь мы используем функцию, которая называется *probability percentile function (ppf)*. Это функция, обратная функции распределения. Здесь мы просто задаем процент и вычисляем сразу же z-value.

Мы разобрали каждый параметр по отдельности. Давайте соберем все вместе и найдем **доверительный интервал**.

```
interval = z_value * (st_dev/np.sqrt(sample_size))
conf_inv = (sample_mean - interval, sample_mean + interval)

print("Confidence interval:", conf_inv)
```

Давайте теперь все полученные нами выше вычисления **соберем в одну функцию** и попробуем применить подсчет доверительного интервала для другой выборки.

```
def compute_ci(sample, st_dev):

    z_value = stats.norm.ppf(q = 0.975)
    sample_size = len(sample)
    interval = z_value * (st_dev/np.sqrt(sample_size))
    conf_inv = (sample_mean - interval, sample_mean + interval)

    return conf_inv
```

Например, мы возьмем выборку большего размера, чем исходная.

```
np.random.seed(5)
sample_size = 2000
sample = np.random.choice(a = population, size = sample_size)

ci = compute_ci(sample, st_dev)

print("conf interval for 2000 sample size:", ci)
```

Мы заметим, что в этом случае доверительный интервал стал уже. **То есть чем больше выборка, тем точнее мы можем оценить параметры распределения.**

Мы рассмотрели случаи с известной дисперсией. А теперь представим, что у нас дана очень маленькая выборка и мы **не можем оценить по ней дисперсию**. И тут на помощь нам приходит распределение Стьюдента. Давайте посмотрим, как в этом случае будет выглядеть формула доверительного интервала. Нам также необходимо оценить среднее

$\mathbb{P}(\bar{X} - t_{1-\frac{\alpha}{2},n-1}\frac{S_0}{\sqrt{n}} \leq \mu \leq \bar{X} + t_{1-\frac{\alpha}{2},n-1}\frac{S_0}{\sqrt{n}}) = 1 - \alpha$, где $\bar{X}$ - выборочное среднее, S_0 — исправленная выборочная дисперсия, n — размер выборки.

Давайте тоже **соберем некоторую функцию и подсчитаем в этом случае доверительный интервал**. Для того чтобы нам вычислить распределение Стьюдента, нам необходимо использовать уже соответствующий класс t и из него вызвать метод, который мы вызывали в случае нормального распределения. И передать параметры, которые будут соответствовать нашему интервалу.

```
def compute_ci_t(sample, alpha=0.95):

    n = sample.shape[0]
    mu, se = np.mean(sample), stats.sem(sample)
    bound = se * stats.t.ppf((1 + alpha) / 2., n-1)

    return mu - bound, mu + bound
```

Давайте теперь посмотрим, как в этом случае будет выглядеть **доверительный интервал**.

```
sample = np.random.choice(a = population, size = 30)
ci_t = compute_ci_t(sample, alpha=0.95)
print("conf interval with t test for 2000 sample size:", ci_t)
```

Мы видим, что в этом случае доверительный интервал стал сильно больше, потому что дисперсия неизвестна. Поэтому имеет смысл его расширять.

Мы рассмотрели, как можно вычислить доверительный интервал для нормального распределения. Но зачастую на практике нам **дана некоторая выборка, и неизвестно, какого она происхождения**. Как в этом случае можно оценить какие-то параметры по этой выборке и как понять, что действительно выборка репрезентативная и статистически значимая? В этом случае нам поможет Центральная Предельная Теорема (ЦПТ). Давайте вспомним, что **распределение средних — это есть нормальное распределение**. А для нормального распределения мы уже знаем, как вычислять доверительный интервал.

Давайте попробуем вычислить доверительный интервал на примере поездок такси в Мехико.

```
taxi_mex = pd.read_csv('taxi-routes/mex_clean.csv')

def generate_distribution_sample(data, sample_size, dist_size):

    sample_means = []
    for i in tqdm(range(dist_size)):
        sample = np.random.choice(a = data, size = sample_size)
        sample_means.append(np.mean(sample))

    return sample_means
```

Для начала нам необходимо сгенерировать распределение средних. Для этого нам необходимо задать размер выборки и сгенерировать таких выборок очень много. Ведь как вы помните, одним из условий теоремы было то, что **чем больше размер выборки, тем ближе распределение средних к нормальному распределению**.

```
sample_size = 10000
dist_size = 50000

sample_means = generate_distribution_sample(taxi_mex['dist_meters']/1000,
                                            sample_size, dist_size)
```

Давайте теперь посмотрим, как выглядит **гистограмма для распределения средних**.

```
plt.hist(sample_means, bins=100)
plt.xlabel('distance in km')
```

Мы видим, что оно уже более симметричное и можно понять, какой разброс значений у этого параметра. Далее, так как это уже нормальное распределение, то мы можем вычислить соответствующие персентили, и после этого мы получаем наш доверительный интервал. Давайте для теста возьмем расстояние, которое проезжает такси в километрах, и **построим** для этого значения **доверительный интервал**.

```
np.sort(sample_means)
lb = np.percentile(sample_means, 2.5)
ub = np.percentile(sample_means, 97.5)
print("conf interval for bootstrap:", (lb, ub))
```

2.10 Проверка гипотез и распределение Стьюдента

Давайте познакомимся с тем, как можно проверять гипотезы, используя t-критерий Стьюдента. Вообще говоря, в статистике **t-критерий Стьюдента** — это набор методов для проверки гипотез. И зачастую его используют для проверки равенства средних значений по выборке. Для этого нам необходимо вычислить сначала t-статистику и затем ее сравнить с некоторым пороговым значением. И после этого мы можем сказать, что гипотеза принимается или отклоняется.

T-статистика обычно строится по следующему принципу. В числителе мы указываем величину с нулевым матожиданием, а в знаменателе — стандартное отклонение от этой случайной величины.

Как вообще построен **алгоритм проверки гипотез**? Для начала нам необходимо определить некоторую нулевую гипотезу. В случае сравнения средних мы предполагаем, что средние значения по двум выборкам равны. И также определяем альтернативную ей, то есть эти средние значения неравны. После этого мы вычисляем t-статистику и сравниваем так называемое p-value. **P-value** — это, формально говоря, вероятность принять нулевую гипотезу при условии, что верна альтернативная. То есть, если p-value очень маленькое значение, то, скорее всего, не происходит каких-то случайных процессов. А если же p-value довольно большое, то вероятность того, что выборки, полученные случайным образом, не имеют никаких связей, очень большая.

Давайте попробуем рассмотреть **три случая применения критерия Стьюдента**.

- Первый случай — это когда мы **сравниваем среднее значение по выборке с каким-то известным средним**.
- Второй случай — когда мы **сравниваем два значения по двум независимым выборкам**.
- Третий случай — когда у нас дана серия измерений, и мы хотим **проверить, насколько статистически значимы все измерения**.

Давайте перейдем к **одновыборочному критерию**. Для этого мы возьмем dataset поездок по городам Мехико и Богота и попробуем проверить, отличается ли как-то продолжительность поездок по этим городам и время ожидания такси.

```
import numpy as np
from scipy import stats
import matplotlib.pyplot as plt
import pandas as pd
%matplotlib inline

taxi_bog = pd.read_csv('taxi-routes/bog_clean.csv')
taxi_mex = pd.read_csv('taxi-routes/mex_clean.csv')
```

Предположим, что мы хотим проверить время ожидания такси. Мы предполагаем, что среднее время ожидания — это десять минут. Для этого в *stats* уже реализованы все методы, и для **проверки одновыборочного критерия** мы используем метод *ttest_ 1samp*. Для этого нам необходимо передать нашу выборку и также передать то значение среднего, которое мы ожидаем увидеть. Результатом работы этого метода будет некоторый класс, который возвращает два параметра: *statistic* — это, собственно говоря, t-статистика, и рассчитанный *pvalue*.

```
sample = taxi_mex['wait_sec'].sample(n=3000)/60
stats.ttest_1samp(sample, 10)
```

Для того чтобы понять, можем ли мы **отвергнуть или принять нулевую гипотезу**, нам надо посмотреть на значение p-value. Обычно p-value берут равным 0.05 или 0.01. Но мы здесь зафиксируем значение, равное 0.05. Мы видим, что для этой гипотезы p-value меньше, чем 0.05, то есть у нас получается, что не равны средние, то есть мы отвергаем нулевую гипотезу.

Давайте теперь **рассмотрим две выборки**. Возьмем поездки по одному городу, по Мехико, и сравним с поездками по Боготе.

```
taxi_mex['pickup_datetime'] = pd.to_datetime(taxi_mex.pickup_datetime)
taxi_mex['month'] = taxi_mex['pickup_datetime'].dt.month

taxi_bog.shape
taxi_mex.shape
```

Мы предполагаем, что средняя продолжительность поездок по двум городам одинаковая. Мы можем подсчитать также t-статистику. Для этого нам уже нужен другой метод, который называется *ttest_ind*, и мы передаем уже сюда наши две выборки. Далее возвращается так же, как в предыдущем случае, вычисленная статистика и p-value.

```
stats.ttest_ind(taxi_mex['trip_duration'].sample(n=3000),
                taxi_bog['trip_duration'].sample(n=3000))
```

Мы видим, что в данном случае p-value очень близко к 0, то есть мы опять отвергаем нулевую гипотезу. То есть все-таки продолжительность поездок отличается.

А вот в случае ожидания времени такси мы уже не можем отвергнуть нулевую гипотезу, что в принципе логично. Примерно в среднем люди ожидают такси одинаковое время.

```
stats.ttest_ind(taxi_mex['wait_sec'].sample(n=3000),
                taxi_bog['wait_sec'].sample(n=3000))
```

А теперь давайте **посмотрим поездки в одном городе, но в разные месяцы**. Например, возьмем поездки по городу Мехико за ноябрь и декабрь и посмотрим, отличается ли как-то продолжительность поездок. В данном случае мы используем тоже метод из *SciPy*, он называется *ttest_rel*. И мы передаем уже наши две выборки.

```
control = taxi_mex[taxi_mex.month == 11]['trip_duration'].sample(n=1000)
treatment = taxi_mex[taxi_mex.month == 12]['trip_duration'].sample(n=1000)

stats.ttest_rel(control, treatment)
```

И как мы видим, в данном случае p-value уже сильно больше, чем наш уровень значимости. То есть мы уже не можем отвергнуть нулевую гипотезу, что в принципе логично. Мы предполагаем, что ноябрь и декабрь так близко находятся в календаре, что, скорее всего, погодные условия никак не влияют на продолжительность поездок.