

МИРЭК | 4 модуль

Авторы: Алла Тамбовцева, Татьяна Рогович, НИУ ВШЭ*

Основы программирования в Python

Семинар 3

Введение в NumPy

Базовые операции с массивами

Сегодня мы познакомимся с библиотекой NumPy (сокращение от *Numeric Python*), которая часто используется в задачах, связанных с машинным обучением.

Чтобы мы смогли на конкретных примерах увидеть, зачем эта библиотека используется, давайте её импортируем. Если вы уже устанавливали Anaconda, то библиотека NumPy также была установлена на ваш компьютер. Проверим: импортируем библиотеку с сокращённым названием, так часто делают, чтобы не «таскать» за собой в коде длинное название. Сокращение np для библиотеки NumPy – распространённое, можно даже сказать, общепринятое, его часто можно увидеть в документации или официальных тьюториалах.

```
In [2]: import numpy as np
```

Возможности:

- поддержка многомерных массивов (включая матрицы);
- поддержка высокоуровневых математических функций, предназначенных для работы с многомерными массивами.

Так же NumPy лежит в основе датафреймов pandas, и многие вещи (фильтрация, индексирование, математические операции), которые обсудим в этом уроке, потом транслируются в pandas.

В Python к числовым типам относятся:

- int
- float
- bool
- complex

В numpy имеются эти типы, а также обёртки над этими типами, которые **используют реализацию типов на C**, например, int8 , int16 , int32 , int64 (подробнее о типах данных numpy можно прочитать [здесь \(https://www.numpy.org/devdocs/user/basics.types.html\)](https://www.numpy.org/devdocs/user/basics.types.html)). За счёт того, что используются типы данных из C, numpy получает ускорение операций.

```
In [5]: type(np.int(32)), type(np.int32(32)), type(np.int64(32))
```

```
Out[5]: (int, numpy.int32, numpy.int64)
```

Переполнение

В связи с особенностями типов numpy важно помнить о переполнении.

Например, 64-битный int в C хранит числа от -9223372036854775808 до 9223372036854775807

```
In [6]: print(np.int64(1e10)) # все поместились
```

```
10000000000
```

```
In [7]: print(np.int64(10e18)) # не поместились
```

```
-----
-----
OverflowError                                Traceback (most recent call last)
all last)
<ipython-input-7-111fb4090569> in <module>()
----> 1 print(np.int64(10e18)) # не поместились
```

```
OverflowError: int too big to convert
```

```
In [8]: print(10e18) # а в стандартный питоновский integer - помещается
```

```
1e+19
```

Массивы Ndarray

Основным объектом `numpy` является `Ndarray` – это n -мерный массив (сокращение от *n -dimensional array*), структура данных, которая позволяет хранить набор элементов одного типа: либо только целые числа, либо числа с плавающей точкой, либо строки, либо булевы (логические) значения.

Производить вычисления с массивами гораздо быстрее и эффективнее чем со списками.

Массивы могут быть одномерными, то есть визуально ничем не отличаться от простого списка значений.

```
In [9]: np.array([0, 2, 3, 4])
```

```
Out[9]: array([0, 2, 3, 4])
```

А могут быть многомерными (n -мерными), то есть представлять собой вложенный список («список списков»):

```
In [10]: np.array([[1, 2],  
                 [1, 0]]) # двумерный
```

```
Out[10]: array([[1, 2],  
                 [1, 0]])
```

Или даже «список таблиц»:

```
In [17]: x = np.array([[[6, 3],  
                      [6, 8]],  
                      [[1, 100],  
                       [0, 1]]]) # трехмерный
```

```
In [18]: x[1][0][1] # вторая колонка первой строки второго измерения
```

```
Out[18]: 100
```

Мы чаще всего будем работать с двумерными массивами. Про двумерный массив можно думать как про матрицу или про таблицу. Так, массив во втором примере выше можно рассматривать как таблицу, состоящую из двух строк и трёх столбцов, как таблицу 2×3 (сначала указывается число строк, затем – число столбцов). Отсюда следует важный факт: число элементов в списках внутри массива должно совпадать. Проверим на примере – возьмём списки разной длины, то есть списки, состоящие из разного числа элементов, и объединим их в массив:

```
In [19]: np.array([[0, 0, 1],
                 [0, 1]])
```

```
Out[19]: array([list([0, 0, 1]), list([0, 1])], dtype=object)
```

Получилось что-то немного странное. Никакой ошибки Python не выдал, но воспринимать этот объект как полноценный массив он уже не будет: он будет считать, что в такой таблице у нас есть две строки и ноль столбцов!

Теперь давайте посмотрим, что будет, если мы попробуем объединить в массив объекты разных типов, например, целые числа и числа с плавающей точкой:

```
In [20]: np.array([[5, 8.2],
                 [1.2, 1, ]])
```

```
Out[20]: array([[5., 8.2],
                 [1.2, 1.]])
```

```
In [3]: np.array([[5, 8.2],
                 [1.2, 'ba', ]])
```

```
Out[3]: array([('5', '8.2'),
                 ('1.2', 'ba')], dtype='<U32')
```

Все элементы были автоматически приведены к одному типу (можно считать, что тип *float* «сильнее» типа *integer*). Можете самостоятельно проверить, что будет, если мы «смешаем» в списке строковые и числовые значения.

Чем же удобны массивы? Во-первых, они занимают меньше места и памяти. Во-вторых, с ними очень удобно работать: все операции над массивами будут производиться поэлементно: то есть, для выполнения действий над каждым элементом массива, нам не придется использовать какие-то специальные конструкции вроде циклов, мы сможем обращаться сразу ко всему массиву. Например, давайте представим, что у нас есть массив со значениями явки на выборы в долях, а мы хотим получить результаты в процентах (домноженные на 100).

В отличие от питоновских листов - ndarray - это полноценный вектор. И он поддерживает операции над векторами. Так пример выше - это умножение вектора на скаляр.

```
In [21]: turnout = np.array([0.62, 0.43, 0.79, 0.56])
turnout
```

```
Out[21]: array([0.62, 0.43, 0.79, 0.56])
```

Чтобы домножить каждое число в массиве на 100, нам достаточно домножить на 100 `turnout`:

```
In [22]: turnout * 100 # ГОТОВО!
```

```
Out[22]: array([62., 43., 79., 56.])
```

Точно так же можем производить операции с несколькими массивами — действия будут выполняться поэлементно (сложение векторов)

```
In [9]: # поэлементное сложение векторов
A = np.array([2, 3, 5])
B = np.array([0, 8, 6])

A + B
```

```
Out[9]: array([ 2, 11, 11])
```

```
In [10]: # полеэлементное умножение векторов
A * B
```

```
Out[10]: array([ 0, 24, 30])
```

Выполним сразу несколько действий — посчитаем явку на основе массивов с числом действительных и недействительных бюллетеней.

```
In [24]: valid = np.array([32, 45, 50, 44])# действительные бюллетени
invalid = np.array([3, 11, 2, 6]) # недействительные бюллетени
total = np.array([65, 72, 80, 100]) # всего зарегистрированных избирателей

(valid + invalid) / total * 100
```

```
Out[24]: array([53.84615385, 77.77777778, 65.          , 50.        ])
```

Массивы `ndarray` и операции с ними

Наиболее важные атрибуты объектов ndarray:

1. **ndarray.ndim** - число измерений (чаще их называют "оси") массива.
2. **ndarray.shape** - размеры массива, его форма. Это кортеж натуральных чисел, показывающий длину массива по каждой оси. Для матрицы из n строк и m столбов, shape будет (n,m) . Число элементов кортежа shape равно ndim.
3. **ndarray.size** - количество элементов массива. Очевидно, равно произведению всех элементов атрибута shape.
4. **ndarray.dtype** - объект, описывающий тип элементов массива. Можно определить dtype, используя стандартные типы данных Python. Можно хранить и пустые типы, например: bool, int16, int32, int64, float16, float32, float64, complex64

Одномерный массив

```
In [29]: M = np.array([[2, 5],  
                   [6, 8],  
                   [1, 3]])  
M
```

```
Out[29]: array([[2, 5],  
                 [6, 8],  
                 [1, 3]])
```

Массивы бывают многомерными, значит, у массива есть число измерений. Давайте его найдём:

```
In [13]: M.ndim # dimensions  
Out[13]: 2
```

Действительно, всего два измерения: чтобы указать на число 5 из этого массива, нам понадобятся всего две координаты – номер строки и номер столбца. Теперь посмотрим на форму или вид массива (**shape**):

```
In [14]: M.shape # 3 строки и 2 столбца, т.е. 3 списка по 2 элемента  
Out[14]: (3, 2)
```

Кроме того, можем найти общее число элементов в массиве, его длину, размер (**size**):

```
In [15]: M.size # всего 6 элементов  
Out[15]: 6
```

Работа с элементами массива

Если нам нужно обратиться к элементам массива, то эта операция будет похожа на работу со вложенными списками:

```
In [16]: M
```

```
Out[16]: array([[2, 5],  
                 [6, 8],  
                 [1, 3]])
```

```
In [17]: M[0] # весь первый список в M
```

```
Out[17]: array([2, 5])
```

```
In [18]: M[0][1] # второй элемент первого списка в M
```

```
Out[18]: 5
```

Или не совсем как со списками, без двойных скобок:

```
In [19]: M[0, 1]
```

```
Out[19]: 5
```

Ещё можно выбирать сразу несколько элементов массива. Для этого воспользуемся срезами (*slices*):

```
In [20]: M[0:2] # с элемента с индексом 0 до элемента с индексом 1 включительно
```

```
Out[20]: array([[2, 5],  
                 [6, 8]])
```

Обратите внимание: правый конец среза не включается.

Концы среза можно опускать, если нас интересуют все элементы, начиная с некоторого элемента и до конца массива или начиная с первого элемента массива и до некоторого элемента (правый конец точно так же включаться не будет):

```
In [21]: M[1:] # с элемента с индексом 1 до конца
```

```
Out[21]: array([[6, 8],  
                 [1, 3]])
```

```
In [22]: M[:2] # с начала массива до элемента с индексом 1 включительно
```

```
Out[22]: array([[2, 5],  
                 [6, 8]])
```

Еще можно взять полный срез – выбрать все элементы массива:

```
In [23]: M[:]
```

```
Out[23]: array([[2, 5],  
                 [6, 8],  
                 [1, 3]])
```

Кроме того, при выборе элементов можно выставлять шаг. По умолчанию мы выбираем все элементы подряд, шаг равен 1, но это можно изменить:

```
In [24]: M[0:3:2] # с нулевого по третий через 2
```

```
Out[24]: array([[2, 5],  
                 [1, 3]])
```

Концы среза по-прежнему можно опускать:

```
In [25]: M[0::2]
```

```
Out[25]: array([[2, 5],  
                 [1, 3]])
```

Или сделать более интересную вещь, взять отрицательный шаг и выбрать все элементы в обратном порядке, с конца:

```
In [26]: M[::-1]
```

```
Out[26]: array([[1, 3],  
                 [6, 8],  
                 [2, 5]])
```

Ещё про операции с массивами

Теперь посмотрим на другие операции с массивами. Создадим простой одномерный массив, содержащий оценки группы школьников:

```
In [27]: marks = np.array([5, 4, 3, 5, 5, 4, 3, 4])  
marks
```

```
Out[27]: array([5, 4, 3, 5, 5, 4, 3, 4])
```

Найдем самую плохую, минимальную оценку:

```
In [28]: marks.min()
```

```
Out[28]: 3
```

А теперь самую высокую, максимальную:

```
In [29]: marks.max()
```

```
Out[29]: 5
```

И средний балл:

```
In [30]: marks.mean()
```

```
Out[30]: 4.125
```

Медиану мы так не найдём — нет метода `median()`, но зато есть такая функция:

```
In [31]: np.median(marks)
```

```
Out[31]: 4.0
```

А теперь найдем номер ученика с самой высокой оценкой:

```
In [32]: marks.argmax()
```

```
Out[32]: 0
```

И номер ученика с самой низкой оценкой:

```
In [33]: marks.argmax()
```

```
Out[33]: 2
```

Внимание: если таких несколько, будет выведено первое совпадение, как для `argmin()`, так и для `argmax()`.

Конечно, мы не сможем сейчас рассмотреть все доступные методы, относящиеся к массивам (некоторые часто используемые методы мы еще будем обсуждать в следующем модуле), но при желании на перечень доступных методов можно посмотреть, набрав название массива, поставив точку и нажав на *Tab* (показать).

Теперь посмотрим на многомерный массив, для удобства возьмём двумерный:

```
In [52]: grades = np.array([[3, 5, 5, 4, 3],  
                         [3, 3, 4, 3, 3],  
                         [5, 5, 5, 4, 5]])
```

Пусть это будут оценки трёх студентов 5 контрольных работ. Попробуем теперь найти средний балл за контрольные по каждому группе. Для этого необходимо указать, по какому измерению мы будем двигаться (грубо говоря, по строкам или столбцам):

```
In [53]: grades.mean(axis = 1) # по строкам, три оценки - одна для каждого студента
```

```
Out[53]: array([4., 3.2, 4.8])
```

А теперь найдем средний балл по каждой контрольной работе:

```
In [54]: grades.mean(axis = 0) # по столбцам, пять оценки - одна для каждой работы
```

```
Out[54]: array([3.66666667, 4.33333333, 4.66666667, 3.66666667, 3.66666667])
```

Таким же образом можно было посмотреть на минимальное и максимальное значение (можете потренироваться самостоятельно).

Создание массивов

Как создать массив?

Способ 1

С первым способом мы уже отчасти познакомились: можно получить массив из готового списка, воспользовавшись функцией `array()`:

```
In [37]: np.array([10.5, 45, 2.4])
```

```
Out[37]: array([10.5, 45., 2.4])
```

Способ 2

Можно создать массив на основе промежутка, созданного с помощью `arange()` – функции `numpy`, похожей на стандартный `range()`, только более гибкую. Посмотрим, как работает эта функция.

```
In [38]: np.arange(2, 9) # по умолчанию шаг равен 1, как обычный range()
```

```
Out[38]: array([2, 3, 4, 5, 6, 7, 8])
```

По умолчанию эта функция создает массив, элементы которого начинаются со значения 2 и заканчиваются на значении 8 (правый конец промежутка не включается), следя друг за другом с шагом 1. Но этот шаг можно менять:

```
In [39]: np.arange(2, 9, 3) # с шагом 3
```

```
Out[39]: array([2, 5, 8])
```

И даже делать дробным!

```
In [40]: np.arange(2, 9, 0.5)
```

```
Out[40]: array([2., 2.5, 3., 3.5, 4., 4.5, 5., 5.5, 6., 6.5, 7., 7.5, 8., 8.5])
```

Или создать массив из диапазона значений `[start, stop]` с заданием количества точек.

```
In [32]: m = np.linspace(0, 5, 5)
print(m)

[0. 1.25 2.5 3.75 5.]
```

Способ 3

Еще массив можно создать совсем с нуля. Единственное, что нужно четко представлять – это его размерность, его форму, то есть опять же, число строк и столбцов. Библиотека `numpy` позволяет создать массивы, состоящие из нулей или единиц, а также «пустые» массивы (на практике используются редко). Удобство заключается в том, что сначала можно создать массив, инициализировать его (например, заполнить нулями), а затем заменить нули на другие значения в соответствии с требуемыми условиями.

Так выглядит массив из нулей:

```
In [41]: Z = np.zeros((3, 3)) # размеры в виде кортежа – не теряйте еще одни
круглые скобки
Z

Out[41]: array([[0., 0., 0.],
 [0., 0., 0.],
 [0., 0., 0.]])
```

```
In [30]: # создание вектора из нулей
v = np.zeros(4)
print(v)

[0. 0. 0. 0.]
```

А так – массив из единиц:

```
In [42]: O = np.ones((4, 2))
O

Out[42]: array([[1., 1.],
 [1., 1.],
 [1., 1.],
 [1., 1.]])
```

А так выглядит единичная матрица – таблица из 0 и 1, в которой число строк и столбцов одинаково, и где на главной диагонали стоят 1:

```
In [43]: E = np.eye(5)
E
```

```
Out[43]: array([[1., 0., 0., 0., 0.],
   [0., 1., 0., 0., 0.],
   [0., 0., 1., 0., 0.],
   [0., 0., 0., 1., 0.],
   [0., 0., 0., 0., 1.]])
```

Создайте матрицу 5x5 со значениями строк в диапазоне от 0 до 4

```
In [31]: m = np.zeros((5, 5))
m += np.arange(5)
print(m)
```

```
[[0. 1. 2. 3. 4.]
 [0. 1. 2. 3. 4.]
 [0. 1. 2. 3. 4.]
 [0. 1. 2. 3. 4.]
 [0. 1. 2. 3. 4.]]
```

Способ 4

Создание массива случайных чисел.

В numpy есть аналог модуля random - numpy.random. Используя типизацию из C, он как и свой аналог генерирует случайные данные.

```
In [39]: np.random.seed(42)
```

```
In [35]: # массив чисел из равномерного (uniform) распределения в диапазоне
# [0, 1)
# np.random.rand(d0, d1, d2, ...) d0, d1,... - размеры возвращаемого массива
print(np.random.rand(2, 2))
print(np.random.rand(2, 2).shape)
```

```
[[0.37454012 0.95071431]
 [0.73199394 0.59865848]]
(2, 2)
```

```
In [38]: # массив чисел из стандартного нормального (norm) распределения
np.random.randn(2, 3, 2)
print(np.random.randn(2, 3, 2).shape)
```

```
(2, 3, 2)
```

```
In [40]: # массив из случайно выбранных чисел  
# size - размер возвращаемого массива, replace=False без замещения  
np.random.choice(a=np.arange(20), size=5, replace=False)
```

```
Out[40]: array([ 0, 17, 15, 1, 8])
```

```
In [41]: np.random.choice(a=np.arange(20), size=(2, 3), replace=True)
```

```
Out[41]: array([[ 1, 0, 11],  
[11, 16, 9]])
```

Изменение размерности списков

Вспомним, что у нас есть массив оценок студентов `grades` :

```
In [55]: grades
```

```
Out[55]: array([[3, 5, 5, 4, 3],  
[3, 3, 4, 3, 3],  
[5, 5, 5, 4, 5]])
```

Как поменять структуру массива так, чтобы, например, оценки были записаны группами по три оценки? Воспользоваться методом `.reshape()`, который позволяет поменять форму массива.

```
In [56]: grades.reshape(5, 3)
```

```
Out[56]: array([[3, 5, 5],  
[4, 3, 3],  
[3, 4, 3],  
[3, 5, 5],  
[5, 4, 5]])
```

Теперь массив двумерный, и чтобы обратиться к элементу массива, нам нужно указывать две вещи: индекс списка и индекс элемента в этом списке. Метод `.reshape()` удобен, но при его использовании стоит помнить, что не любой массив можно превратить в массив другой формы – общее число элементов в массиве должно позволять получить новое число списков и элементов в них. Так, массив `grades`, в котором всего 15 элементов, нельзя превратить в массив вида $(2, 8)$ (таблица 2×8), потому что для такой формы понадобится 16 элементов! И Python явно об этом сообщит:

```
In [57]: grades.reshape(2, 8)
```

```
-----
-----
ValueError                                Traceback (most recent call last)
all last)
<ipython-input-57-52df4f35f5b0> in <module>()
----> 1 grades.reshape(2, 8)

ValueError: cannot reshape array of size 15 into shape (2,8)
```

Если нам нужно просто поменять местами строки и столбцы в таблице, то есть списки в массиве, можно воспользоваться транспонированием, которое осуществляется в NumPy с помощью метода `.transpose()`:

```
In [58]: grades.transpose()
```

```
Out[58]: array([[3, 3, 5],
                 [5, 3, 5],
                 [5, 4, 5],
                 [4, 3, 4],
                 [3, 3, 5]])
```

Кроме того, в противоположность `.reshape()`, который часто используется для разбиения одномерного массива на многомерный из нескольких маленьких списков, в NumPy существует «обратный» метод `.ravel()`, который позволяет любой многомерный массив превратить в одномерный, состоящий из одного списка, другими словами, сделать массив «плоским»:

```
In [59]: grades.ravel()
```

```
Out[59]: array([3, 5, 5, 4, 3, 3, 3, 4, 3, 3, 5, 5, 5, 4, 5])
```

Примечание: в NumPy есть ещё другой метод для создания «плоских» массивов – `flatten()`.

Проверка условий на массивах

Давайте посмотрим, каким образом можно проверять условия на массивах и отбирать элементы по условию. Создадим массив со значениями возраста:

```
In [60]: ages = np.array([[15, 23, 32, 45, 52],
                         [68, 34, 55, 78, 20],
                         [25, 67, 33, 45, 14]])
```

Давайте попробуем узнать, какие значения массива соответствуют людям трудоспособного возраста: от 16 лет и старше:

```
In [61]: ages >= 16 # больше или равно
```

```
Out[61]: array([[False,  True,  True,  True,  True],
                 [ True,  True,  True,  True,  True],
                 [ True,  True,  True,  True, False]])
```

Все элементы, кроме первого в первом списке и кроме последнего в последнем списке: на всех позициях, кроме указанных, стоят значения `True`, что означает, что условие выполняется. То, что мы получили сейчас – это булев массив, массив, состоящий из булевых (логических) значений, значений `True` и `False`.

Теперь попробуем сформулировать более сложное условие: проверим, какие элементы соответствуют людям старше 18, но младше 60 лет:

```
In [62]: (ages > 18) & (ages < 60) # & – одновременное условие
```

```
Out[62]: array([[False,  True,  True,  True,  True],
                 [False,  True,  True, False,  True],
                 [ True, False,  True,  True, False]])
```

Как посчитать, сколько элементов массива удовлетворяют некоторым условиям?

Суммируем значения по всему массиву: Python понимает, что значение `True` – это 1, а `False` – это 0, поэтому нет необходимости превращать все значения в числовые, мы можем просто сложить все «единички»:

```
In [63]: ((ages > 18) & (ages < 60)).sum()
```

```
Out[63]: 10
```

А теперь проверим, какие значения соответствуют людям либо младше 18, либо старше 60:

```
In [64]: (ages < 18) | (ages > 60) # | – или – хотя бы одно условие верно
```

```
Out[64]: array([[ True, False, False, False, False],
                 [ True, False, False,  True, False],
                 [False,  True, False, False,  True]])
```

А как увидеть сами значения, которые удовлетворяют определенным условиям? Заключить условие в квадратные скобочки:

```
In [65]: ages[ages >= 16]
```

```
Out[65]: array([23, 32, 45, 52, 68, 34, 55, 78, 20, 25, 67, 33, 45])
```

```
In [66]: ages[(ages >= 16) & (ages < 60)]
```

```
Out[66]: array([23, 32, 45, 52, 34, 55, 20, 25, 33, 45])
```

Внимание: не забудьте круглые скобки для каждого условия, иначе Python поймёт всё неправильно и вернёт ошибку:

```
In [67]: ages[ages >= 16 & ages < 60]
```

```
-----
ValueError                                Traceback (most recent call last)
 in <module>()
      1 ages[ages >= 16 & ages < 60]
-----
```

ValueError: The truth value of an array with more than one element is ambiguous. Use a.any() or a.all()

Операции с векторами и матрицами в нампай (дополнительный материал)

Скалярное произведение векторов

```
In [74]: a = np.array([3, 1, 5, 2])
b = np.array([2, 5, 2, 4])
# <a, b> = 3*1 + 1*5 + 5*2 + 2*4
print(a @ b)    # python 3 style
print(a.dot(b))
print(np.dot(a, b))
```

```
29
29
29
```

Умножение матриц

Операция умножения определена для двух матриц, таких что число столбцов первой равно числу строк второй.

Пусть матрицы A и B таковы, что $A \in \mathbb{R}^{n \times k}$ и $B \in \mathbb{R}^{k \times m}$.

Произведением матриц A и B называется матрица C , такая что

$$c_{ij} = \sum_{r=1}^k a_{ir} b_{rj}$$

, где c_{ij} — элемент матрицы C , стоящий на пересечении строки с номером i и столбца с номером j .

```
In [11]: a = np.array([[1, 2], [2, 0]])
b = np.array([[2,5], [1, 3]])
print(a)
print(b)
print(a @ b)      # python 3 style
print(a.dot(b))
print(np.dot(a, b))
```

```
[[1 2]
 [2 0]]
[[2 5]
 [1 3]]
[[ 4 11]
 [ 4 10]]
[[ 4 11]
 [ 4 10]]
[[ 4 11]
 [ 4 10]]
```

!!!Не путайте поокординатное умножение с матричным!!!

```
In [13]: print(a * b)
```

```
[[ 2 10]
 [ 2  0]]
```

Умножение матриц и векторов

```
In [82]: m = np.array([[1, 2], [0, 1], [2, 4]])
print(m)
v = np.array([2, 5])
print("v = ", v)
m @ v
```

```
[[1 2]
 [0 1]
 [2 4]]
v =  [2 5]
```

```
Out[82]: array([12, 5, 24])
```

Полезные функции и методы для работы с массивами

```
In [84]: a = np.random.choice(a=np.linspace(1, 50, 50), size=10, replace=False)
print(a)
```

```
[ 5. 24. 19. 1. 29. 27. 11. 35. 13. 22.]
```

1. Замена элементов по индексу

```
In [86]: np.put(a, ind=[0, 2], v=[-44, -55])
a
```

```
Out[86]: array([-44., 24., -55., 1., 29., 27., 11., 35., 13., 22.])
```

2. Выделение массива по условию

```
In [89]: # замена элементов массива по условию: a if a < 0 else 0
np.where(a < 0, a, 0)
```

```
Out[89]: array([-44., 0., -55., 0., 0., 0., 0., 0., 0., 0.])
```

```
In [90]: # выбор элементов по условию
a[np.where(a < 0)]
```

```
Out[90]: array([-44., -55.])
```

3. Сортировка

```
In [91]: # сортировка  
np.sort(a)
```

```
Out[91]: array([-55., -44., 1., 11., 13., 22., 24., 27., 29., 35.])
```

```
In [92]: # индексы отсортированного списка  
np.argsort(a)
```

```
Out[92]: array([2, 0, 3, 6, 8, 9, 1, 5, 4, 7], dtype=int64)
```

4. Any и All для сложных логических условий

Any возвращает True, если хотя бы один элемент True

All возвращает True, если все элементы True

```
In [93]: any([True, True, False, True, False, False, False])
```

```
Out[93]: True
```

```
In [94]: all([True, True, False, True, False, False, False])
```

```
Out[94]: False
```

```
In [98]: # сравнение векторов  
np.array([1, 1, 0, 0]) == np.array([1, 1, 0, 2])
```

```
Out[98]: array([ True,  True,  True, False])
```

```
In [96]: all(np.array([1, 1, 0, 0]) == np.array([1, 1, 0, 2]))
```

```
Out[96]: False
```

```
In [97]: any(np.array([1, 1, 0, 0]) == np.array([1, 1, 0, 2]))
```

```
Out[97]: True
```

МИРЭК | 4 модуль

Автор: Татьяна Рогович

Основы программирования в Python

Семинар 3

Введение в Pandas

Pandas - библиотека для работы с табличными данными в питоне.

- Документация: <https://pandas.pydata.org/>
- 10 minutes intro: https://pandas.pydata.org/pandas-docs/stable/getting_started/10min.html
- Pandas Cheat-Sheet: https://pandas.pydata.org/Pandas_Cheat_Sheet.pdf

```
In [1]: import pandas
```

Чтобы не писать название библиотеки целиком каждый раз, когда понадобится ее использовать, принято сокращать название библиотеки и импортировать ее как "pd":

```
In [2]: import pandas as pd # импортировали библиотеку pandas и назвали ее pd
```

В Pandas есть тип данных датафрейм (DataFrame), в котором удобно хранить таблицы с данными. Создадим небольшой датафрейм своими руками:

```
In [3]: df = pd.DataFrame() # создали пустой датафрейм с помощью метода DataFrame() библиотеки pandas (pd)
df['a'] = [10,20,30] # создаем колонку "a" и помещаем в нее столбец с данными - [10, 20, 30]
df
```

Out[3]:

	a
0	10
1	20
2	30

В датафрейме автоматически создалась нумерация строк - по умолчанию она с 0.

```
In [4]: df['b'] = ['one', 'two', 'three']
df
```

Out[4]:

	a	b
0	10	one
1	20	two
2	30	three

Конечно, чаще всего приходится работать с уже готовыми наборами данных. Такие данные обычно хранятся в формате xls(x) - для работы в Excel, или (чаще) в формате csv - comma-separated value. Попробуем импортировать csv файл с данными о пассажирах Титаника: они лежат в файле 'titanic.csv' (попробуйте открыть его в текстовом редакторе и посмотрите, как он устроен внутри).

```
In [5]: data = pd.read_csv('https://raw.githubusercontent.com/rogovich/2019_HSE_DPO_Python_for_data_analysis/master/lectures-seminars/10-23-2019_Pandas/titanic.csv')
```

Функция read_csv читает данные из файла формата csv и преобразует в pandas.DataFrame. Аналогичная функция read_excel может читать данные в формате xls(x).

Посмотрим на наши данные:

In [6]: `data.head() # функция head() показывает первые строки датафрейма, по умолчанию 5`

Out[6]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.250
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...)	female	38.0	1	0	PC 17599	71.283
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.925
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.100
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.050

In [8]: `data.head(10) # можно передать аргументом количество строк, которые хотите увидеть`

Out[8]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.250
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...)	female	38.0	1	0	PC 17599	71.280
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.925
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.100
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.050
5	6	0	3	Moran, Mr. James	male	NaN	0	0	330877	8.450
6	7	0	1	McCarthy, Mr. Timothy J	male	54.0	0	0	17463	51.862
7	8	0	3	Palsson, Master. Gosta Leonard	male	2.0	3	1	349909	21.075
8	9	1	3	Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)	female	27.0	0	2	347742	11.130
9	10	1	2	Nasser, Mrs. Nicholas (Adele Achem)	female	14.0	1	0	237736	30.070

In [7]: `data.tail(10) # можно посмотреть последние записи`

Out[7]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket
881	882	0	3	Markun, Mr. Johann	male	33.0	0	0	349257 7
882	883	0	3	Dahlberg, Miss. Gerda Ulrika	female	22.0	0	0	7552 10
883	884	0	2	Banfield, Mr. Frederick James	male	28.0	0	0	C.A./SOTON 34068 10
884	885	0	3	Sutefall, Mr. Henry Jr	male	25.0	0	0	SOTON/OQ 392076 7
885	886	0	3	Rice, Mrs. William (Margaret Norton)	female	39.0	0	5	382652 28
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	211536 18
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	112053 30
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	Nan	1	2	W.C. 6607 28
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369 30
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	370376 7

По столбцам идут признаки, по строкам - объекты (пассажиры).

In [8]: `data.shape # функция shape показывает размерность датафрейма (строк, столбцов)`

Out[8]: (891, 12)

```
In [9]: data.columns # СПИСОК СТОЛБЦОВ
```

```
Out[9]: Index(['PassengerId', 'Survived', 'Pclass', 'Name', 'Sex', 'Age',
   'SibSp',
   'Parch', 'Ticket', 'Fare', 'Cabin', 'Embarked'],
  dtype='object')
```

Описание признаков:

PassengerId - id пассажира

Survived бинарная переменная: выжил пассажирил (1) или нет (0)

Pclass - класс пассажира

Name - имя пассажира

Sex - пол пассажира

Age - возраст пассажира

SibSp - количество родственников (братьев, сестер, супругов) пассажира на борту

Parch - количество родственников (родителей / детей) пассажира на борту

Ticket - номер билета

Fare - тариф (стоимость билета)

Cabin - номер кабинки

Embarked - порт, в котором пассажир сел на борт (C - Cherbourg, S - Southampton, Q = Queenstown)

Так можно обратиться к столбцу:

```
In [10]: data['Age'].head()
```

```
Out[10]: 0    22.0
1    38.0
2    26.0
3    35.0
4    35.0
Name: Age, dtype: float64
```

```
In [11]: data.Age.head()
```

```
Out[11]: 0    22.0
1    38.0
2    26.0
3    35.0
4    35.0
Name: Age, dtype: float64
```

Или к нескольким столбцам сразу:

```
In [12]: data[['Age', 'Sex']].head()
```

```
Out[12]:
```

	Age	Sex
0	22.0	male
1	38.0	female
2	26.0	female
3	35.0	female
4	35.0	male

А так - к строке по индексу:

```
In [15]: data.loc[0]
```

```
Out[15]: PassengerId      1
Survived          0
Pclass            3
Name      Braund, Mr. Owen Harris
Sex                male
Age              22
SibSp             1
Parch             0
Ticket        A/5 21171
Fare            7.25
Cabin           NaN
Embarked         S
Name: 0, dtype: object
```

In [13]: `data.iloc[0]`

```
Out[13]: PassengerId          1
Survived           0
Pclass            3
Name      Braund, Mr. Owen Harris
Sex                  male
Age                 22
SibSp             1
Parch             0
Ticket          A/5 21171
Fare                7.25
Cabin              NaN
Embarked          S
Name: 0, dtype: object
```

In [14]: `data.loc[1:3] # строки с 1 по 3`

Out[14]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fai
1		2	1	Cumings, Mrs. John Bradley (Florence Briggs Th... Heikkinen, Miss. Laina Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	38.0	1	0	PC 17599	71.283
2		3	1	3	female	26.0	0	0	STON/O2. 3101282	7.925
3		4	1	1	female	35.0	1	0	113803	53.100

In [16]: `data.loc[1:3, 'Survived':'Sex'] # строки с 1 по 3, колонки от Survived до Sex`

Out[16]:

	Survived	Pclass	Name	Sex
1	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th... Heikkinen, Miss. Laina Futrelle, Mrs. Jacques Heath (Lily May Peel)	female
2	1	3		female
3	1	1		female

loc возвращает данные на основе индекса, а **iloc** - основываясь исключительно на позиции индекса, начиная с 0.

Пример:

```
In [6]: df = pd.DataFrame()
df['Name'] = ['Tom', 'Jack', 'Nick', 'Juli']
df['Mark'] = [99, 98, 95, 90]
df.index = [1,3,2,0]
df
```

Out[6]:

	Name	Mark
1	Tom	99
3	Jack	98
2	Nick	95
0	Juli	90

```
In [7]: df.loc[1]
```

Out[7]: Name Tom
Mark 99
Name: 1, dtype: object

```
In [19]: df.iloc[1]
```

Out[19]: Name Jack
Mark 98
Name: 3, dtype: object

```
In [20]: df.loc[1:2]
```

Out[20]:

	Name	Mark
1	Tom	99
3	Jack	98
2	Nick	95

```
In [21]: df.iloc[1:2]
```

Out[21]:

	Name	Mark
3	Jack	98

Можем сделать индексом даже колонку с именем. И тоже сможем обращаться к объектам через .loc

```
In [10]: df.index = df.Name  
df.loc['Nick']['Mark']
```

```
Out[10]: 95
```

```
In [11]: df.loc['Jack':'Nick']
```

```
Out[11]:
```

Name	Mark
Name	
Jack	Jack 98
Nick	Nick 95

Кроме того, можно выбирать объекты, удовлетворяющие каким-то свойствам, например, все пассажиры-женщины:

```
In [22]: data[data.Sex == 'female'].head()
```

Out[22]:

PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare
1	2	1	Cumings, Mrs. John Bradley (Florence Briggs Th...)	female	38.0	1	0	PC 17599	71.283
2	3	1	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.925
3	4	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.100
8	9	1	Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)	female	27.0	0	2	347742	11.133
9	10	1	Nasser, Mrs. Nicholas (Adele Achem)	female	14.0	1	0	237736	30.070

Пассажиры первого класса:

```
In [12]: data[data.Pclass == 1].head()
```

Out[12]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare
1	1	2	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833
3	3	4	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000
6	6	7	0	McCarthy, Mr. Timothy J	male	54.0	0	0	17463	51.8625
11	11	12	1	Bonnell, Miss. Elizabeth	female	58.0	0	0	113783	26.5500
23	23	24	1	Sloper, Mr. William Thompson	male	28.0	0	0	113788	35.5000

Пассажиры первого или второго классов:

```
In [13]: data[data.Pclass.isin([1,2]).head()]
```

Out[13]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare
1	1	2	1	Cumings, Mrs. John Bradley (Florence Briggs Th... Futrelle, Mrs. Jacques Heath (Lily May Peel) McCarthy, Mr. Timothy J Nasser, Mrs. Nicholas (Adele Achem) Bonnell, Miss. Elizabeth	female	38.0	1	0	PC 17599	71.2833 113803 53.1000 17463 237736 30.0708 113783 26.5500
3	3	4	1	1	female	35.0	1	0		
6	6	7	0	1	male	54.0	0	0	17463	51.8625
9	9	10	1	2	female	14.0	1	0	237736	30.0708
11	11	12	1	1	female	58.0	0	0	113783	26.5500

Пассажиры младше 18:

In [14]: `data[data.Age < 18].head()`

Out[14]:

PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare
7	8	0	Palsson, Master. Gosta Leonard	male	2.0	3	1	349909	21.075
9	10	1	Nasser, Mrs. Nicholas (Adele Achem)	female	14.0	1	0	237736	30.070
10	11	1	Sandstrom, Miss. Marguerite Rut	female	4.0	1	1	PP 9549	16.700
14	15	0	Vestrom, Miss. Hulda Amanda Adolfina	female	14.0	0	0	350406	7.854
16	17	0	Rice, Master. Eugene	male	2.0	4	1	382652	29.125

Девушки в возрасте от 18 до 25, путешествующие в одиночку (без каких-либо родственников):

In [15]: `data[(data.Sex == 'female') & (data.Age > 18) & (data.Age < 25) & (data.SibSp == 0) & (data.Parch == 0)]`

Out[15]:

PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket
44	45	1	Devaney, Miss. Margaret Delia	female	19.0	0	0	330958
56	57	1	Rugg, Miss. Emily	female	21.0	0	0	C.A. 31026 1
106	107	1	Salkjelsvik, Miss. Anna Kristine	female	21.0	0	0	343120
141	142	1	Nysten, Miss. Anna Sofia	female	22.0	0	0	347081
199	200	0	Yrois, Miss. Henriette ("Mrs Harbeck")	female	24.0	0	0	248747 1
			Connolly,					

289	290	1	3	Miss. Kate	female	22.0	0	0	370373
293	294	0	3	Haas, Miss. Aloisia	female	24.0	0	0	349236
310	311	1	1	Hays, Miss. Margaret Bechstein	female	24.0	0	0	11767 8
345	346	1	2	Brown, Miss. Amelia "Mildred"	female	24.0	0	0	248733 1
369	370	1	1	Aubart, Mme. Leontine Pauline	female	24.0	0	0	PC 17477 6
376	377	1	3	Landergren, Miss. Aurora Adelia	female	22.0	0	0	C 7077
404	405	0	3	Oreskovic, Miss. Marija	female	20.0	0	0	315096
427	428	1	2	Phillips, Miss. Kate Florence ("Mrs Kate Louis...")	female	19.0	0	0	250655 2
473	474	1	2	Jerwan, Mrs. Amin S (Marie Marthe Thuillard)	female	23.0	0	0	SC/AH Basle 541 1
474	475	0	3	Strandberg, Miss. Ida Sofia	female	22.0	0	0	7553
501	502	0	3	Canavan, Miss. Mary	female	21.0	0	0	364846
554	555	1	3	Ohman, Miss. Velin	female	22.0	0	0	347085
627	628	1	1	Longley, Miss. Gretchen Fiske	female	21.0	0	0	13502 7
641	642	1	1	Sagesser, Mlle. Emma	female	24.0	0	0	PC 17477 6
649	650	1	3	Stanley, Miss. Amy Zillah Elsie	female	23.0	0	0	CA. 2314
708	709	1	1	Cleaver, Miss. Alice	female	22.0	0	0	113781 15
				Mayne,					

710	711	1	1	Mlle. Berthe Antonine ("Mrs de Villiers")	female	24.0	0	0	PC	17482	4
816	817	0	3	Heininen, Miss. Wendla Maria	female	23.0	0	0	STON/O2.	3101290	
882	883	0	3	Dahlberg, Miss. Gerda Ulrika	female	22.0	0	0	7552	1	
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	112053	3	

Сколько таких путешественниц?

```
In [16]: data[(data.Sex == 'female') & (data.Age > 18) & (data.Age < 25) & (data.SibSp == 0) &(data.Parch == 0)].shape
```

```
Out[16]: (25, 12)
```

Задание:

- 1) Посчитайте количество пассажиров первого класса, которые сели на борт в Саутгемптоне.
- 2) Солько пассажиров третьего класса, которые путешествовали в компании 2 или более родственников (братьев / сестер / супругов)?
- 3) Сколько в среднем стоил билет первого класса?

```
In [30]: # (ᵔ⌇ᵔ)ゞ — ☆・*
```

Иногда нужно создать новый признак из уже существующих, например, нам интересно, сколько всего родственников путешествовало с каждым пассажиром - просто сложим столбцы SibSp и Parch и поместим сумму в новый столбец FamilySize.

```
In [18]: data['FamilySize'] = data['SibSp'] + data['Parch']
data.head(10)
```

Out[18]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.250
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...)	female	38.0	1	0	PC 17599	71.283
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.925
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.100
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.050
5	6	0	3	Moran, Mr. James	male	NaN	0	0	330877	8.450
6	7	0	1	McCarthy, Mr. Timothy J	male	54.0	0	0	17463	51.862
7	8	0	3	Palsson, Master. Gosta Leonard	male	2.0	3	1	349909	21.075
8	9	1	3	Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)	female	27.0	0	2	347742	11.130
9	10	1	2	Nasser, Mrs. Nicholas (Adele Achem)	female	14.0	1	0	237736	30.070

А теперь давайте создадим переменную, которая бы нам показывала, что пассажир ехал в одиночку. Такой пассажир путешествовал без родственников. Мы напишем условие с помощью анонимной функции (1, если FamilySize равно 0 и 0 во всех остальных случаях) и применим ее к столбцу FamilySize с помощью метода .apply().

```
In [20]: data['Alone'] = data['FamilySize'].apply(lambda x: 1 if x == 0 else 0)
data.head(10)
```

Out[20]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.250
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...)	female	38.0	1	0	PC 17599	71.280
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.925
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.100
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.050
5	6	0	3	Moran, Mr. James	male	NaN	0	0	330877	8.450
6	7	0	1	McCarthy, Mr. Timothy J	male	54.0	0	0	17463	51.862
7	8	0	3	Palsson, Master. Gosta Leonard	male	2.0	3	1	349909	21.075
8	9	1	3	Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)	female	27.0	0	2	347742	11.130
9	10	1	2	Nasser, Mrs. Nicholas (Adele Achem)	female	14.0	1	0	237736	30.070

Lambda-функции

Это особый вид функций, которые объявляются с помощью ключевого слова **lambda** вместо **def**:
Лямбда-функции принимают любое количество аргументов, но не могут содержать несколько выражений и всегда возвращают только одно значение. В программировании на **Python** можно обойтись без анонимных функций, которые по сути являются обычными, но без имени и с ограничением в одно выражение. Однако их использование в нужных местах упрощает написание и восприятие кода. Пишется так:

- lambda arguments: expression

arguments - аргументы, expression - выражение, возвращающее значение.

Пример (lambda функция, которая добавляет к переданному аргументу 1 и возвращает результат):

```
In [21]: add_1 = lambda x: x + 1  
add_1(8)
```

```
Out[21]: 9
```

```
In [22]: add_2 = lambda x,y: x + y  
add_2(3,5)
```

```
Out[22]: 8
```

Также мы можем сочетать lambda функцию с оператором if (но только для двух условий)

```
In [32]: even = lambda x: 'YES' if x % 2 == 0 else 'NO'  
even(2)
```

```
Out[32]: 'YES'
```

Давайте вернемся к "Титанику", функция, которую применяем к столбцу, может быть и посложнее. Давайте из каждого имени пассажира достанем его титул. Сначала потренируемся на одном пассажире.

```
In [23]: data.loc[0]['Name']
```

```
Out[23]: 'Braund, Mr. Owen Harris'
```

Ок, выбрали имя. Это строка. Давайте подумаем, как достать из нее титул. Вроде бы титул всегда после фамилии. Разобьем строку по запятой и достанем второй (первый по индексу) элемент. Потом разобьем по точке и возьмем первый (нулевой) элемент. И избавимся от пробела.

```
In [25]: data.loc[0]['Name'].split(',')[1].split('.')[0].strip()
```

```
Out[25]: 'Mr'
```

Ура! Теперь напишем функцию, которая будет забирать титул из имени, а потом применим ее к колонке Name.

```
In [26]: def return_title(full_name):
    return(full_name.split(',')[1].split('.')[0].strip())
```

Теперь сформируем новый столбец family_name из столбца Name с помощью написанной нами функции:

```
In [27]: data['Title'] = data.Name.apply(return_title)
data.head()
```

```
Out[27]:
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.250
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...)	female	38.0	1	0	PC 17599	71.283
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.925
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.100
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.050

Кстати, удалить колонку можно так. В нашем анализе мы не будем использовать колонку Ticket, давайте удалим ее.

```
In [28]: del data['Ticket']
```

In [29]: `data.head()`

Out[29]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Fare	Cabin	
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	7.2500	NaN	
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...)	female	38.0	1	0	71.2833	C85	
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	7.9250	NaN	
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	53.1000	C123	
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	8.0500	NaN	

Полезно посмотреть, какие значения содержатся в столбце. Для категориальных данных можно посчитать частоту встречаемости каждого значения с помощью функции `value_counts`:

In [30]: `data['Title'].value_counts()`

Out[30]:

Mr	517
Miss	182
Mrs	125
Master	40
Dr	7
Rev	6
Major	2
Mlle	2
Col	2
Ms	1
the Countess	1
Lady	1
Capt	1
Sir	1
Mme	1
Don	1
Jonkheer	1
Name: Title, dtype: int64	

Очень много уникальных значений! Обычно это не очень хорошо для статистического анализа, давайте все такие титулы переименуем в Misc (другие).

```
In [35]: data['Title'] = data['Title'].apply(lambda x: 'Misc' if x not in ['Mr', 'Miss', 'Mrs', 'Master'] else x)
```

```
In [36]: data['Title'].value_counts()
```

```
Out[36]: Mr      517  
Miss     182  
Mrs      125  
Master    40  
Misc      27  
Name: Title, dtype: int64
```

```
In [37]: data['Pclass'].value_counts()
```

```
Out[37]: 3      491  
1      216  
2      184  
Name: Pclass, dtype: int64
```

```
In [38]: data['Embarked'].value_counts()
```

```
Out[38]: S      644  
C      168  
Q      77  
Name: Embarked, dtype: int64
```

Для количественных данных удобнее смотреть минимальные/максимальные/средние значения:

```
In [39]: print(data['Age'].min())  
print(data['Age'].max())  
print(data['Age'].mean())
```

```
0.42  
80.0  
29.69911764705882
```

В Pandas есть функция `describe()`, которая делает удобную сводную таблицу по всем количественным столбцам сразу (обратите внимание, что для Pandas количественные данные = все, что представлено числами, что, разумеется, неверно в общем случае):

In [40]: `data.describe()`

Out[40]:

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fa
count	891.000000	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	446.000000	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	257.353842	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	223.500000	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	446.000000	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	668.500000	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

In [41]: `data[['Age', 'Fare']].describe() # также можно применять только к отдельным колонкам`

Out[41]:

	Age	Fare
count	714.000000	891.000000
mean	29.699118	32.204208
std	14.526497	49.693429
min	0.420000	0.000000
25%	20.125000	7.910400
50%	28.000000	14.454200
75%	38.000000	31.000000
max	80.000000	512.329200

Данные можно сортировать:

In [42]: `data.sort_values(by=['Age']).head() # сортируем по возрасту, по умолчанию сортировка по возрастанию`

Out[42]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Fare	Cab
803	804	1	3	Thomas, Master. Assad Alexander	male	0.42	0	1	8.5167	Nan
755	756	1	2	Hamalainen, Master. Viljo	male	0.67	1	1	14.5000	Nan
644	645	1	3	Baclini, Miss. Eugenie	female	0.75	2	1	19.2583	Nan
469	470	1	3	Baclini, Miss. Helene Barbara	female	0.75	2	1	19.2583	Nan
78	79	1	2	Caldwell, Master. Alden Gates	male	0.83	0	2	29.0000	Nan

In [43]: `data.sort_values(by=['Age'], ascending=False).head() # сортируем по возрасту, теперь по убыванию`

Out[43]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Fare	Cabin
630	631	1	1	Barkworth, Mr. Algernon Henry Wilson	male	80.0	0	0	30.0000	A2
851	852	0	3	Svensson, Mr. Johan	male	74.0	0	0	7.7750	Nan
493	494	0	1	Artagaveytia, Mr. Ramon	male	71.0	0	0	49.5042	Nan
96	97	0	1	Goldschmidt, Mr. George B	male	71.0	0	0	34.6542	A
116	117	0	3	Connors, Mr. Patrick	male	70.5	0	0	7.7500	Nan

In [44]: `data.sort_values(by=['Age', 'Fare'], ascending=False).head() # сортируем сперва по возрасту (по убыванию),
потом стоимости билета (по убыванию)`

Out[44]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Fare	Cabin
630	631	1	1	Barkworth, Mr. Algernon Henry Wilson	male	80.0	0	0	30.0000	A2
851	852	0	3	Svensson, Mr. Johan	male	74.0	0	0	7.7750	Nan
493	494	0	1	Artagaveytia, Mr. Ramon	male	71.0	0	0	49.5042	Nan
96	97	0	1	Goldschmidt, Mr. George B	male	71.0	0	0	34.6542	A
116	117	0	3	Connors, Mr. Patrick	male	70.5	0	0	7.7500	Nan

In [45]: `data.sort_values(by=['Age', 'Fare'], ascending=[False, True]).head()
сортируем сперва по возрасту (по убыванию),
потом стоимости билета (по возрастанию)`

Out[45]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Fare	Cabin
630	631	1	1	Barkworth, Mr. Algernon Henry Wilson	male	80.0	0	0	30.0000	A2
851	852	0	3	Svensson, Mr. Johan	male	74.0	0	0	7.7750	Nan
96	97	0	1	Goldschmidt, Mr. George B	male	71.0	0	0	34.6542	A
493	494	0	1	Artagaveytia, Mr. Ramon	male	71.0	0	0	49.5042	Nan
116	117	0	3	Connors, Mr. Patrick	male	70.5	0	0	7.7500	Nan

И группировать:

In [46]: `data.groupby('Sex') # разбиение всех объектов на 2 группы по полу – возвращает просто сгруппированный датафрейм`

Out[46]: `<pandas.core.groupby.groupby.DataFrameGroupBy object at 0x00000211 9C95FBA8>`

```
In [47]: data.groupby('Sex')[ 'Pclass'].value_counts() # группируем по полу и  
считаем для каждого пассажира разных классов
```

```
Out[47]: Sex      Pclass  
female    3          144  
           1          94  
           2          76  
male     3          347  
         1          122  
         2          108  
Name: Pclass, dtype: int64
```

```
In [49]: data.groupby('Sex')[ 'Age'].mean() # средний возраст для пассажиров  
каждого из полов
```

```
Out[49]: Sex  
female    27.915709  
male     30.726645  
Name: Age, dtype: float64
```

Задание: отличается ли распределение выживших и не выживших среди мужчин и женщин?

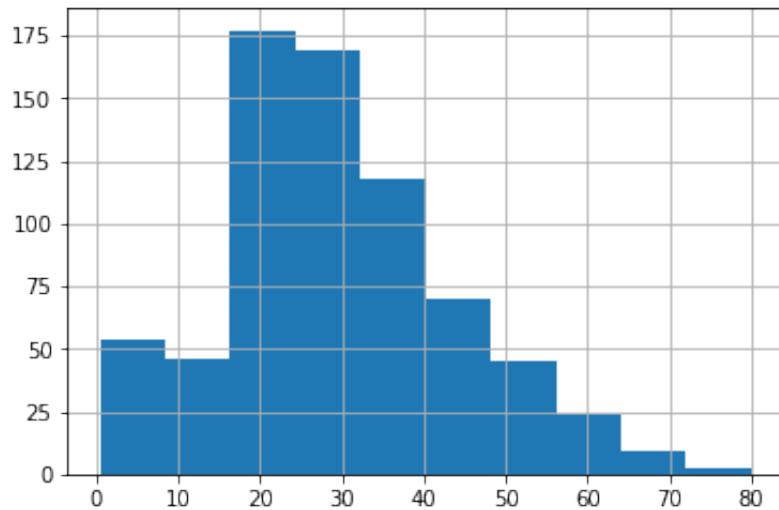
```
In [63]: # (^_^)ゞ — ☆・*
```

Смотреть на числа и таблицы не очень удобно - построим графики!

```
In [50]: # это библиотека matplotlib для отрисовки графиков, мы поговорим о  
ней подробнее на семинаре 4 :)  
# со знака процента начинаются магические функции – эта позволяет нам  
строить графики прямо в блокноте (inline)  
%matplotlib inline
```

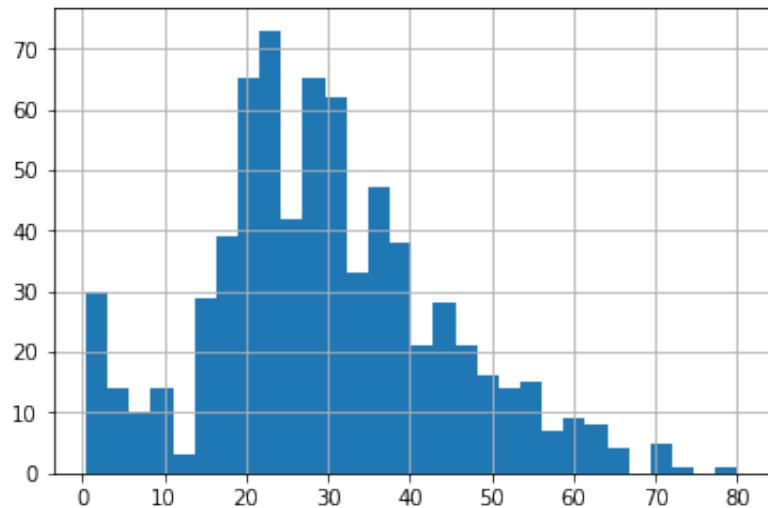
```
In [51]: data.Age.hist() # гистограмма распределения возраста среди пассажиров
```

```
Out[51]: <matplotlib.axes._subplots.AxesSubplot at 0x2119d672320>
```



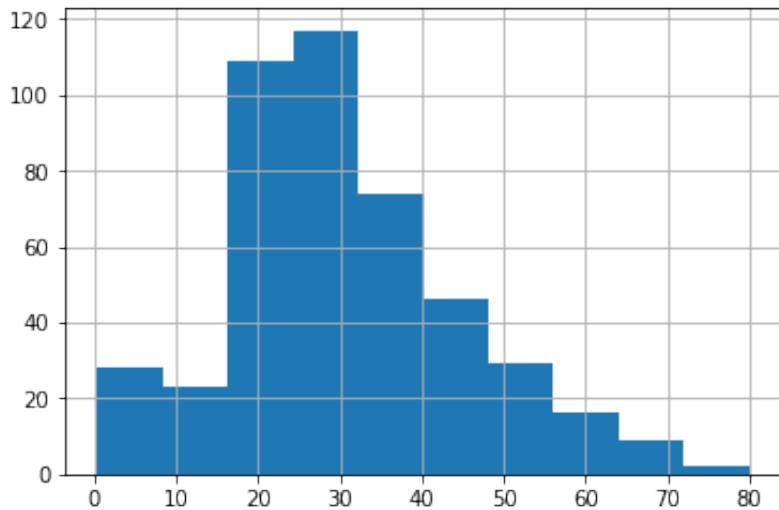
```
In [52]: data.Age.hist(bins = 30) # увеличили кол-во столбцов (бинов)
```

```
Out[52]: <matplotlib.axes._subplots.AxesSubplot at 0x2119d93e048>
```



In [53]: `data[data.Sex == 'male'].Age.hist() #гистограмма распределения возраста среди мужчин`

Out[53]: <matplotlib.axes._subplots.AxesSubplot at 0x2119c968c50>



Задание: постройте гистограмму для распределения стоимости билетов (Fare).

In [50]: `# (ᵔ⌇ᵔ)ゞ — ☆・*`

Задание: сравнивте гистограммы распределения стоимости билетов для разных классов (Pclass).

In [51]: `# (ᵔ⌇ᵔ)ゞ — ☆・*`

Построим столбчатую диаграмму для признака "класс пассажира":

1) Сгруппируем все данные по признаку Pclass - `data.groupby('Pclass')` и посчитаем количество в каждой из групп - `size()`

In [68]: `data.groupby('Pclass').size()`

Out[68]:

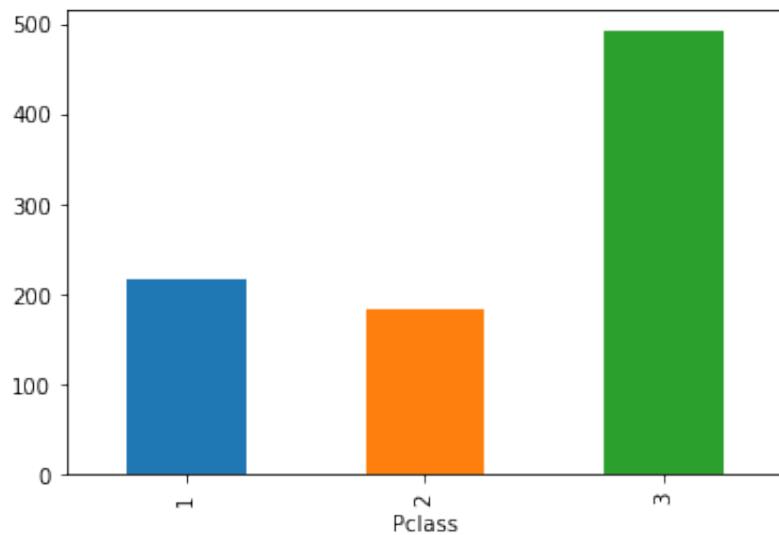
Pclass	size
1	216
2	184
3	491

dtype: int64

2) На полученных данных построим диаграмму:

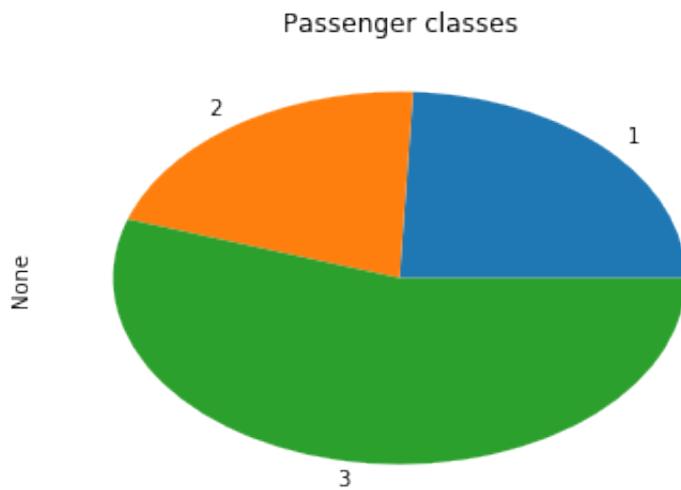
```
In [72]: data.groupby('Pclass').size().plot(kind = 'bar')
```

```
Out[72]: <matplotlib.axes._subplots.AxesSubplot at 0x2500b911ba8>
```



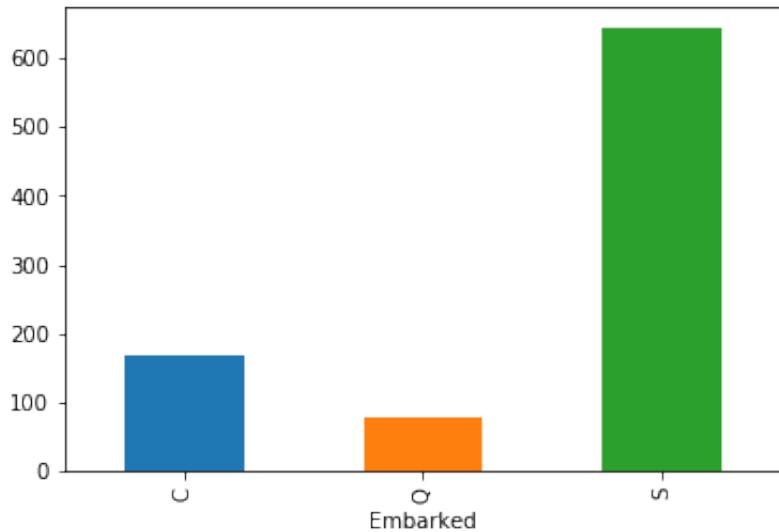
```
In [73]: data.groupby('Pclass').size().plot(kind = 'pie', title = 'Passenger classes')
```

```
Out[73]: <matplotlib.axes._subplots.AxesSubplot at 0x2500b9ce9b0>
```



```
In [74]: data.groupby('Embarked').size().plot(kind = 'bar')
```

```
Out[74]: <matplotlib.axes._subplots.AxesSubplot at 0x2500ba26160>
```



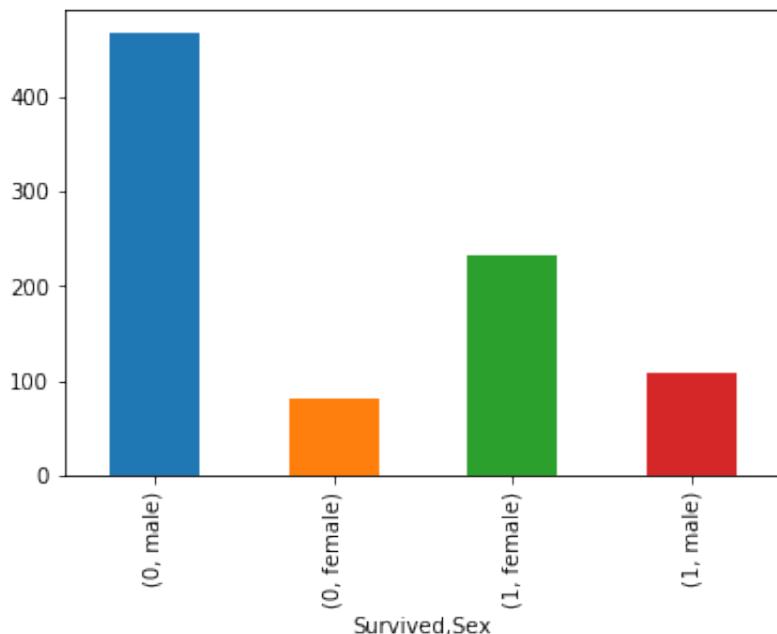
Задание: постройте по гистограмме и круговой диаграмме для еще двух категориальных признаков.

```
In [56]: # (^_^)ゞ — ☆・*
```

Можно делать и чуть более сложные графики. Давайте посмотрим на распределение выживших среди мужчин и женщин.

```
In [75]: data.groupby(['Survived'])['Sex'].value_counts().plot(kind = 'bar')
```

```
Out[75]: <matplotlib.axes._subplots.AxesSubplot at 0x2500ba78198>
```



Так мы видим разбивку, но хотелось бы получить составные столбики. Для этого выполним метод `unstack()`, который преобразует датафрейм.

```
In [76]: data.groupby(['Survived'])['Sex'].value_counts() # без unstack
```

```
Out[76]: Survived   Sex
0           male    468
             female   81
1           female  233
             male    109
Name: Sex, dtype: int64
```

```
In [77]: data.groupby(['Survived'])['Sex'].value_counts().unstack() # с unstack. Обратите внимание, как изменилась структура датафрема.
```

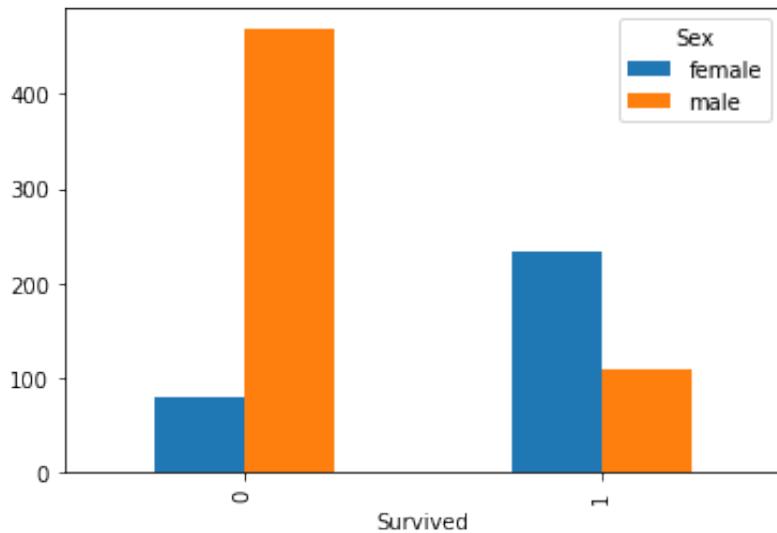
```
Out[77]:
```

	Sex	female	male
Survived			
0		81	468
1		233	109

А вот его мы уже можем визуализировать.

```
In [78]: data.groupby(['Survived'])['Sex'].value_counts().unstack().plot(kind = 'bar')
```

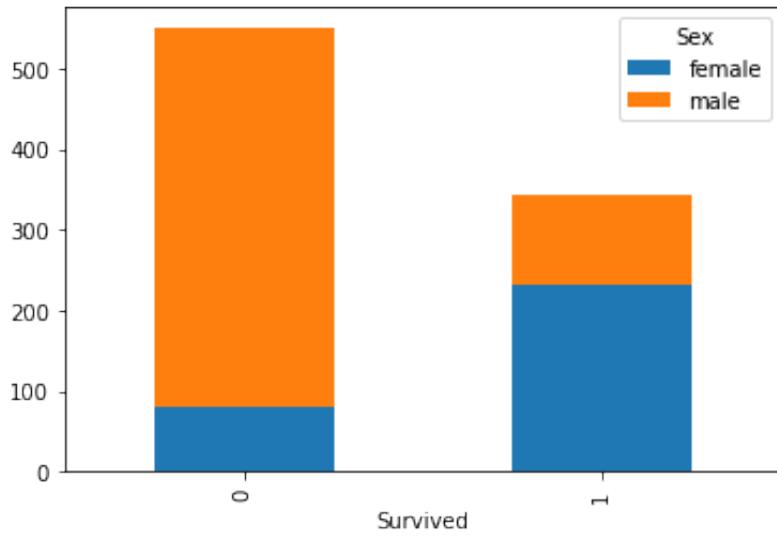
```
Out[78]: <matplotlib.axes._subplots.AxesSubplot at 0x2500caa28d0>
```



Уже лучше. А еще можно добавить дополнительный параметр в метод plot() и станет совсем хорошо.

```
In [79]: data.groupby(['Survived'])['Sex'].value_counts().unstack().plot(kind = 'bar', stacked = True)
```

```
Out[79]: <matplotlib.axes._subplots.AxesSubplot at 0x2500cb0b080>
```



Сохранение датафрейма:

```
In [80]: data.to_csv('new_titanic.csv')
```

Описательные статистики в Python

Описательная статистика на Python: количественные и качественные данные

Описательная статистика или дескриптивная статистика (англ. descriptive statistics) занимается обработкой эмпирических данных, их систематизацией, наглядным представлением в форме графиков и таблиц, а также их количественным описанием посредством статистических показателей.

Для начала разберемся с несколькими важными определениями:

Распределение

Случайная величина — это переменная, значения которой представляют собой исходы какого-нибудь случайного феномена или эксперимента. Простыми словами: это численное выражение результата случайного события. Случайная величина является одним из основных понятий теории вероятностей.

Распределение вероятностей — это закон, описывающий область значений случайной величины и вероятности их исхода (появления).

Чтобы чуть лучше понять, что же такое распределение, давайте посмотрим на гистограмму признака "Age".

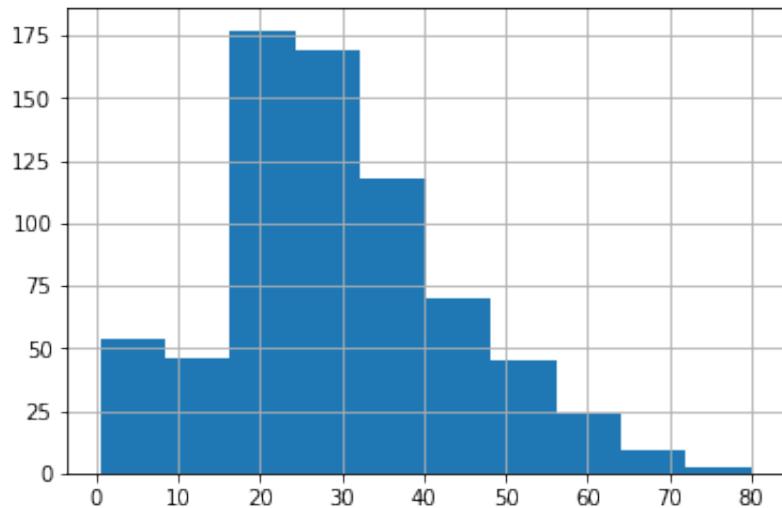
Гистограмма

Гистограмма представляет собой столбчатую диаграмму.

При построении гистограммы множество значений признака разбивается на k интервалов, эти интервалы становятся основаниями столбцов. Высоты столбцов пропорциональны количеству (частоте) значений признака, попадающих в соответствующий интервал.

```
In [54]: data.Age.hist()
```

```
Out[54]: <matplotlib.axes._subplots.AxesSubplot at 0x2119d98e898>
```

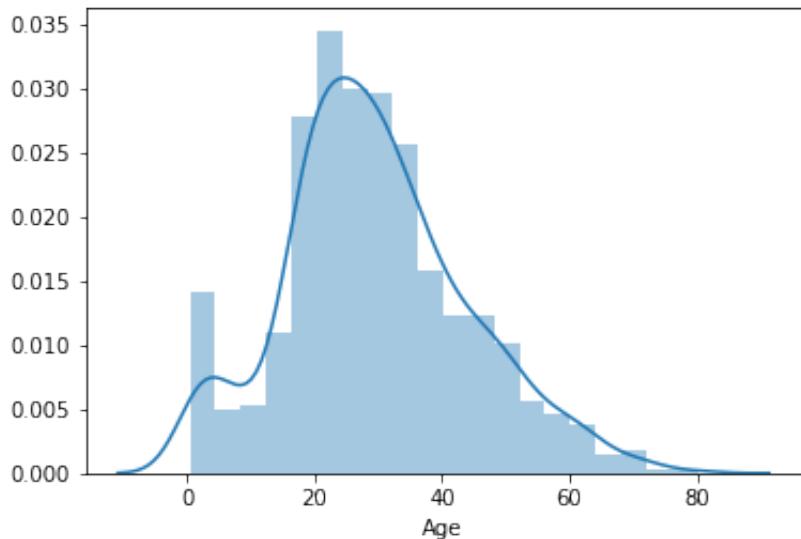


Обычно, когда говорят о распределении, представляют сглаженную линию, под которой могли бы находиться все наши данные. Такой график уже показывает нам не сколько раз встречается каждое значение, а какую долю распределения такие значения составляют. Давайте импортируем библиотеку для визуализаций seaborn (о ней поговорим подробнее в следующий раз), которая поможет нам построить такой график.

```
In [55]: import seaborn as sns # импортируем библиотеку под именем sns  
  
sns.distplot(data.Age.dropna()) # применяем функцию, которая строит график распределения из библиотеки seaborn к нашим данным.  
# методом .dropna() опускаем ячейк и с пропущенными значениями - иначе seaborn сломается.
```

C:\Users\rogov\Anaconda3\lib\site-packages\matplotlib\axes_axes.py:6462: UserWarning: The 'normed' kwarg is deprecated, and has been replaced by the 'density' kwarg.
warnings.warn("The 'normed' kwarg is deprecated, and has been "

Out[55]: <matplotlib.axes._subplots.AxesSubplot at 0x2119f8a88d0>



В идеальном мире многие вещи распределены нормально (например, вес и рост людей). Это значит, что такое распределение имеет определенные параметры (не будем их обсуждать в этом курсе) и выглядит как колокол (а вот это можно и запомнить).

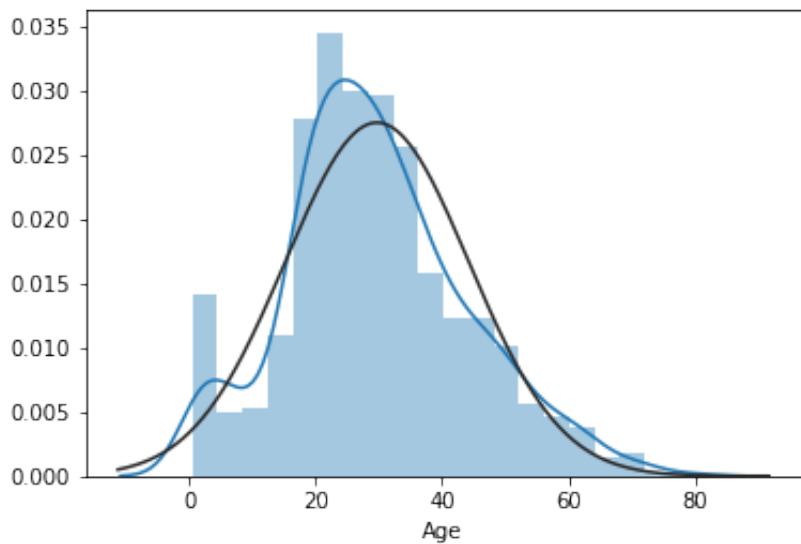
Для некоторых моделей в статистике мы можем применять только переменные, которые распределены нормально. Давайте посмотрим, как распределен возраст пассажиров относительно нормального распределения. Для этого импортируем функцию, которая задает нормальное распределение.

```
In [56]: # из пакета для работы со статистическими данными и моделями импорт
иrуем команду, которая задает нормальное распределение
from scipy.stats import norm

sns.distplot(data.Age.dropna(), fit=norm) # параметр fit строит на
м еще один график поверх существующего,
# сейчас мы просим его п
остроить нормальное распределение
```

C:\Users\rogov\Anaconda3\lib\site-packages\matplotlib\axes_axes.py:6462: UserWarning: The 'normed' kwarg is deprecated, and has been replaced by the 'density' kwarg.
warnings.warn("The 'normed' kwarg is deprecated, and has been "

Out[56]: <matplotlib.axes._subplots.AxesSubplot at 0x2119f9db5c0>



Как распределение возраста выглядит относительно нормального?

Задание: постройте распределение цены билетов, как он выглядит относительно нормального распределения?

Максимальное и минимальное значения

In [57]: data.Age.min()

Out[57]: 0.42

In [58]: data.Age.max()

Out[58]: 80.0

```
In [86]: min_fare = #?
max_fare = #?

print("Минимальная стоимость билета на Титанике составляла %s, а максимальная - %s" % (min_fare, max_fare))
```

Меры центральной тенденции

Меры среднего уровня дают усредненную характеристику совокупности объектов по определенному признаку.

В зависимости от типа шкалы измерения применяются следующие меры среднего уровня:

- для количественных данных - среднее значение (арифметическое среднее), медиана, мода
- для категориальных:

для порядковых данных - медиана, мода

для номинальных данных - мода

На самом деле таких мер довольно много, например: взвешенное среднее — среднее значение, учитывающее весовые коэффициенты для каждого значения, гармоническое среднее — количество наблюдений, деленное на сумму инвертированных значений наблюдений, и так далее.

Среднее значение (математическое ожидание)

Математическое ожидание вычисляется по формуле:

$$\bar{x} = \frac{1}{n} \sum_{i=1}^n x_i = \frac{1}{n}(x_1 + \dots + x_n)$$

```
In [59]: data.Age.mean()
```

```
Out[59]: 29.69911764705882
```

```
In [86]: mean_fare = #?
print("Средняя стоимость билета на Титанике составляла %s" % (mean_fare))

File "<ipython-input-86-235d5c23aaad>", line 1
    mean_fare = #?
    ^
SyntaxError: invalid syntax
```

Немного усложним задачу

```
In [60]: data[['Pclass', 'Age']].groupby('Pclass').mean()
```

Out[60]:

Pclass	Age
1	38.233441
2	29.877630
3	25.140620

Медиана

Если x_1, x_2, \dots, x_n – упорядоченные по возрастанию или убыванию числовые значения рассматриваемого признака, n – объем выборки, то **медиана** – это средний элемент для нечетного n и полусумма средних элементов для четного n .

Для порядковой шкалы медиана является такой точкой на шкале, которая делит всю совокупность опрошенных на две равных части — тех, кто отметил градации меньше этой точки (либо равные ей), и тех, кто отметил градации больше этой точки.

Вопрос знатокам: можно ли посчитать медиану для категориальных не порядковых данных? 🤔

```
In [61]: data.Age.median()
```

Out[61]: 28.0

```
In [23]: median_fare = #?
print("Медиана стоимости билета на Титанике составляла %s" % (median_fare))
```

Мода

Мода – значение во множестве наблюдений, которое встречается наиболее часто.

```
In [62]: data.Pclass.mode()
```

Out[62]: 0 3
dtype: int64

```
In [63]: data.Pclass.value_counts()
```

Out[63]: 3 491
1 216
2 184
Name: Pclass, dtype: int64

Меры разброса

Меры разброса показывают, насколько хорошо данные значения представляют совокупность.
Как меры разброса применяются:

- дисперсия случайной величины и среднеквадратическое отклонение
- коэффициент вариации (это отношение среднеквадратического отклонения к среднему значению, выраженное в процентах, показывает однородность выборки)
- и так далее

Дисперсия и среднеквадратическое отклонение

Дисперсия значений признака является суммой квадратов отклонений этих значений от их среднего, деленной на число наблюдений:

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

Среднеквадратическое отклонение, стандартное отклонение или стандартный разброс - квадратный корень из дисперсии, равный σ

Стандартное отклонение измеряется в тех же единицах, что и сама случайная величина, а дисперсия измеряется в квадратах этой единицы измерения.

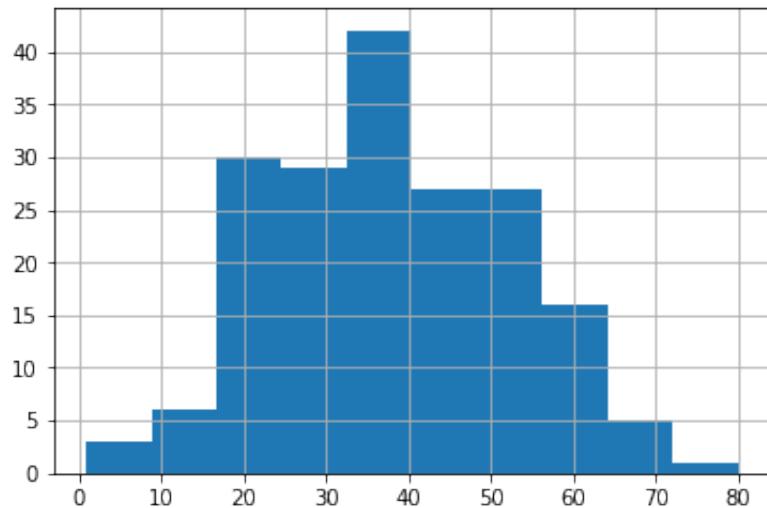
In [64]: `data[['Pclass' , 'Age']].groupby('Pclass').std()`

Out[64]:

Pclass	Age
1	14.802856
2	14.001077
3	12.495398

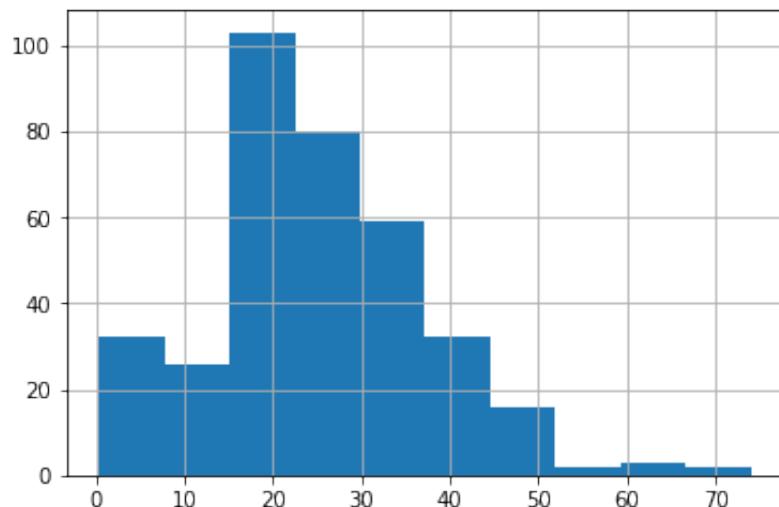
```
In [65]: data[data.Pclass == 1].Age.hist()
```

```
Out[65]: <matplotlib.axes._subplots.AxesSubplot at 0x2119fa585c0>
```



```
In [66]: data[data.Pclass == 3].Age.hist()
```

```
Out[66]: <matplotlib.axes._subplots.AxesSubplot at 0x2119facedd8>
```



Квантили

Квантиль - значение, которое заданная случайная величина не превышает с фиксированной вероятностью. Если вероятность задана в процентах, то квантиль называется процентилем или перцентилем.

Например, фраза «для развитых стран 99-процентиль продолжительности жизни составляет 100 лет» означает, что ожидается, что 99 % людей проживут не более, чем 100 лет.

Относительно нашего датасета фраза "75%-перцентиль возраста пассажиров Титаника равна 38 лет" означает, что 75% пассажиров были не старше 38 лет.

Вопросы знатокам:

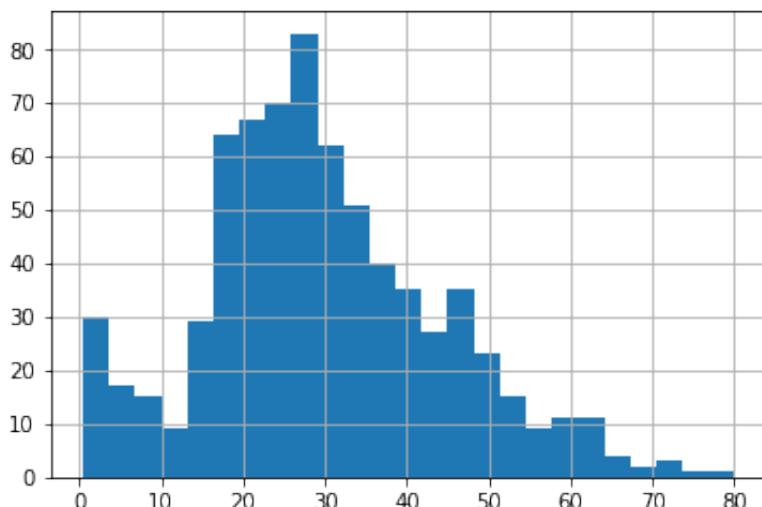
- Как еще можно назвать 50%-перцентиль? 🤔
- Что значит фраза "75%-перцентиль стоимости билетов на Титаник была равна 31"? 🤔

```
In [67]: data.Age.describe()
```

```
Out[67]: count    714.000000
          mean     29.699118
          std      14.526497
          min      0.420000
          25%     20.125000
          50%     28.000000
          75%     38.000000
          max     80.000000
          Name: Age, dtype: float64
```

```
In [68]: data.Age.hist(bins=25)
```

```
Out[68]: <matplotlib.axes._subplots.AxesSubplot at 0x2119fb61c50>
```



Описание массива нечисловых данных

В качестве базовой описательной статистики для категориальных признаков можно использовать следующий набор характеристик:

- количество уникальных представителей массива
- частоты встречаемости этих представителей
- наиболее часто встречающиеся представители (мода распределения)
- наиболее редко встречающиеся представители

```
In [69]: data.Pclass.unique()
```

```
Out[69]: array([3, 1, 2], dtype=int64)
```

```
In [70]: data.Pclass.value_counts()
```

```
Out[70]: 3    491  
1    216  
2    184  
Name: Pclass, dtype: int64
```

```
In [71]: data.Pclass.mode()
```

```
Out[71]: 0    3  
dtype: int64
```

И еще чуть-чуть! Работа с пропущенными значениями.

Если вы помните, то переменная Age содержит пропущенные значения. Давайте посмотрим информацию об этой колонке.

```
In [72]: data[['Age']].info() # обратите внимание - двойные скобки. Так pandas нам вернет датафрейм из одной колонки, а не список.  
# А метод info() работает только с датафреймом
```

```
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 891 entries, 0 to 890  
Data columns (total 1 columns):  
Age    714 non-null float64  
dtypes: float64(1)  
memory usage: 7.0 KB
```

```
In [73]: type(data['Age']) # вот так - объект подобный списку (Series)
```

```
Out[73]: pandas.core.series.Series
```

```
In [74]: type(data[['Age']]) # а вот так - датафрейм
```

```
Out[74]: pandas.core.frame.DataFrame
```

Вернемся к info(). Мы видим, что из 891 наблюдения у нас только 714 ненулевых значений. Значит, у этих пассажиров возраст неизвестен. Ваши идеи, что можно с этим сделать?

Есть разные варианты того, что делать с пропущенными значениями - от "не делать ничего и выкинуть их" до "давайте предскажем их значения с помощью нейронки". Почитать можно здесь:

<https://towardsdatascience.com/6-different-ways-to-compensate-for-missing-values-data-imputation-with-examples-6022d9ca0779> (<https://towardsdatascience.com/6-different-ways-to-compensate-for-missing-values-data-imputation-with-examples-6022d9ca0779>)

Мы с вами попробуем сделать второй по сложности вариант (после не делать ничего) и заменить их средним значением (средним или медианой). Для категориальных данных можно заполнять пропущенные значения модой.

Пропущенные значения могут быть закодированы по-разному - 0, 'No response', '999'. В итоге мы их всегда хотим привести к объекту NaN (not a number), с которым могут работать методы pandas. В нашем датасете они уже нужного формата. В других случаях, нужно будет отфильтрь значения и привести их нужному виду.

```
In [75]: print(data.loc[5, 'Age'])
print(type(data.loc[5, 'Age']))
```

```
nan
<class 'numpy.float64'>
```

In [76]: `data[data['Age'].isnull()].head() # выводим значения датафрейма, в которых отсутствует возраст`
`# Они возвращают True методу .isnull()`

Out[76]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Fare	Cabin
5	6	0	3	Moran, Mr. James	male	NaN	0	0	8.4583	NaT
17	18	1	2	Williams, Mr. Charles Eugene	male	NaN	0	0	13.0000	NaT
19	20	1	3	Masselmani, Mrs. Fatima	female	NaN	0	0	7.2250	NaT
26	27	0	3	Emir, Mr. Farred Chehab	male	NaN	0	0	7.2250	NaT
28	29	1	3	O'Dwyer, Miss. Ellen "Nellie"	female	NaN	0	0	7.8792	NaT

In [77]: `data['Age'].median() # вспомним какая у нас медиана`

Out[77]: 28.0

In [78]: `data['Age_Median'] = data['Age'].fillna(data['Age'].median()) # сохранию результат заполнения в новую колонку`

In [79]: `data[data['Age'].isnull()].head() # смотрим, что произошло с возрастом в новой колонке у тех, у кого он отсутсвовал`

Out[79]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Fare	Cabin
5	6	0	3	Moran, Mr. James	male	NaN	0	0	8.4583	NaT
17	18	1	2	Williams, Mr. Charles Eugene	male	NaN	0	0	13.0000	NaT
19	20	1	3	Masselmani, Mrs. Fatima	female	NaN	0	0	7.2250	NaT
26	27	0	3	Emir, Mr. Farred Chehab	male	NaN	0	0	7.2250	NaT
28	29	1	3	O'Dwyer, Miss. Ellen "Nellie"	female	NaN	0	0	7.8792	NaT

In [80]: `data.head() # А у всех остальных – их нормальный возраст.`

Out[80]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Fare	Cabin	
0		1	0	Braund, Mr. Owen Harris	male	22.0	1	0	7.2500	NaN	
1		2	1	Cumings, Mrs. John Bradley (Florence Briggs Th...)	female	38.0	1	0	71.2833	C85	
2		3	1	Heikkinen, Miss. Laina	female	26.0	0	0	7.9250	NaN	
3		4	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	53.1000	C123	
4		5	0	Allen, Mr. William Henry	male	35.0	0	0	8.0500	NaN	

Такой метод один из самых тупорынх, но сойдет для первого знакомства или черновика анализа в серьезном исследовании. Давайте попробуем теперь немного улучшить его. Посмотрим, как отличается медианный возраст для мужчин и женщин.

In [81]: `data.groupby('Sex')[['Age']].median()`

Out[81]:

Sex	Age
female	27.0
male	29.0

Name: Age, dtype: float64

Разница два года! Было бы логично в наших данных заполнять недостающие значения по полу.

In [82]: `data["Age_Median_Sex"] = data["Age"].fillna(data.groupby('Sex')[["Age"]].transform("median"))`

In [83]: `data[data['Age'].isnull()].head()`

Out[83]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Fare	Cabin
5	6	0	3	Moran, Mr. James	male	NaN	0	0	8.4583	NaN
17	18	1	2	Williams, Mr. Charles Eugene	male	NaN	0	0	13.0000	NaN
19	20	1	3	Masselmani, Mrs. Fatima	female	NaN	0	0	7.2250	NaN
26	27	0	3	Emir, Mr. Farred Chehab	male	NaN	0	0	7.2250	NaN
28	29	1	3	O'Dwyer, Miss. Ellen "Nellie"	female	NaN	0	0	7.8792	NaN

In [84]: `data.head() # Опять проверяем, что это все применилось только к нужным людям`

Out[84]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Fare	Cabin
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	7.2500	NaN
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...)	female	38.0	1	0	71.2833	C85
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	7.9250	NaN
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	53.1000	C123
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	8.0500	NaN

Разберем как работает предыдущий кусок кода

```
In [85]: # эта функция возвращает нам колонку возраст, где все значения заменены медианой по условию пола  
# data.groupby('Sex') - группирует наши значения по полу  
# ['Age'] - колонка, с которой работаем  
# transform('median') - высчитывает медианный возраст для каждого пола и подставляет ее вместо значения  
data.groupby('Sex')[['Age']].transform('median').head()
```

```
Out[85]: 0    29.0  
1    27.0  
2    27.0  
3    27.0  
4    29.0  
Name: Age, dtype: float64
```

```
In [86]: # когда передаем это все как аргумент методу .fillna - заполнение по медиане работает только для отсутствующих значений.  
data["Age"].fillna(data.groupby('Sex')[["Age"]].transform('median')).head(10)
```

```
Out[86]: 0    22.0  
1    38.0  
2    26.0  
3    35.0  
4    35.0  
5    29.0  
6    54.0  
7     2.0  
8    27.0  
9    14.0  
Name: Age, dtype: float64
```

Задание Заполните отсутствующие значения переменной возраст на основании титула.

```
In [ ]:
```

Заполнение по моде для категориальных переменных

Тоже самое (почти!) работает и для категориальных переменных.

In [87]: `data[data["Embarked"].isnull()]`

Out[87]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Fare	Cabin	Em
61	62	1	1	Icard, Miss. Amelie	female	38.0	0	0	80.0	B28	
829	830	1	1	Stone, Mrs. George Nelson (Martha Evelyn)	female	62.0	0	0	80.0	B28	

Давайте посмотрим, что возвращает нам функция мода - не число, как например median или mean, а список.

In [88]: `data['Embarked'].mode()`

Out[88]: 0 S
dtype: object

Чтобы передать ее результат методу fillna, нам нужно "вытащить" значение из него (а это мы умеем делать - оно лежит под нулевым индексом).

In [89]: `data['Embarked'].mode()[0]`

Out[89]: 'S'

In [90]: `# Применяем
data["Embarked_Mode"] = data["Embarked"].fillna(data['Embarked'].mode()[0])`

```
In [91]: # проверяем  
data.loc[61]
```

```
Out[91]: PassengerId      62  
Survived          1  
Pclass            1  
Name      Icard, Miss. Amelie  
Sex                female  
Age              38  
SibSp             0  
Parch             0  
Fare              80  
Cabin            B28  
Embarked         NaN  
FamilySize        0  
Alone             1  
Title            Miss  
Age_Median       38  
Age_Median_Sex   38  
Embarked_Mode    S  
Name: 61, dtype: object
```

МИРЭК | 4 модуль

Автор: Татьяна Рогович

Основы программирования в Python

Семинары 5-6

Библиотека для визуализации данных: matplotlib

Во-первых, кроме привычных вам графиков существует еще огромное множество всего (и некоторые вещи работают гораздо лучше привычных нам). Python умеет строить и сложные штуки. Есть несколько классных сайтов, которые помогают выбрать подходящий график для ваших данных:

<https://www.data-to-viz.com/> (<https://www.data-to-viz.com/>)

<https://datavizproject.com/> (<https://datavizproject.com/>)

<https://datavizcatalogue.com/RU/> (<https://datavizcatalogue.com/RU/>)

Сегодня мы будем работать с несколькими датасетами, чтобы посмотреть разные виды графиков.

```
In [2]: # Наша основная библиотека для визуализаций
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

# магическая функция, чтобы графики отображались в блокноте под ячейками
%matplotlib inline
```

Линейный график (trend line, line plot): кто живет в лесу?

Сначала будем работать с синтетическим набором данных "Кто живет в лесу". У нас есть переменная год и переменные, которые обозначают, сколько в этом году было в лесу зайцев, рысей и морковок.

```
In [2]: # если файл не лежит в рабочей директории jupyter, то не забудьте прописать полный путь к нему
# также обратите внимание, что данные в файле разделены не запятыми, а табуляцией - добавим параметр, который сообщает пандас нужный знак разделителя
forest = pd.read_csv('https://raw.githubusercontent.com/rogovich/2020_MIREC_PfDA/master/Seminars/S5_S6_Matplotlib/populations.txt', sep='\t')
forest.head()
```

Out[2]:

	year	hare	lynx	carrot
0	1900	30000.0	4000.0	48300
1	1901	47200.0	6100.0	48200
2	1902	70200.0	9800.0	41500
3	1903	77400.0	35200.0	38200
4	1904	36300.0	59400.0	40600

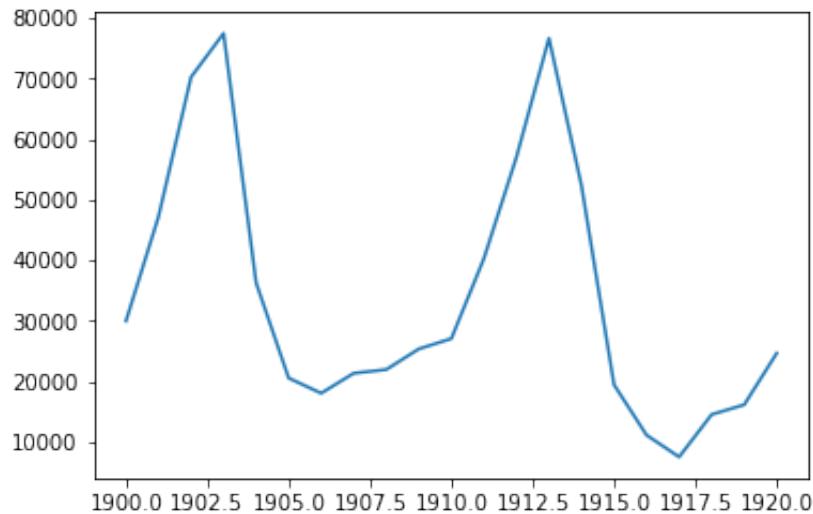
```
In [3]: forest.shape
```

Out[3]: (21, 4)

Давайте для начала посмотрим как выглядит объект, который создает matplotlib И разберемся, как строить в нем графики. Первый вариант создания графика - plt.plot(), которому мы просто можем передать два аргумента - что положить на ось x, а что на y. Если у вас переменные в числовом формате, то без проблем получите линейный график (line plot)

```
In [4]: plt.plot(forest.year, forest.hare) # первый аргумент - ось x, второй аргумент - ось y.
```

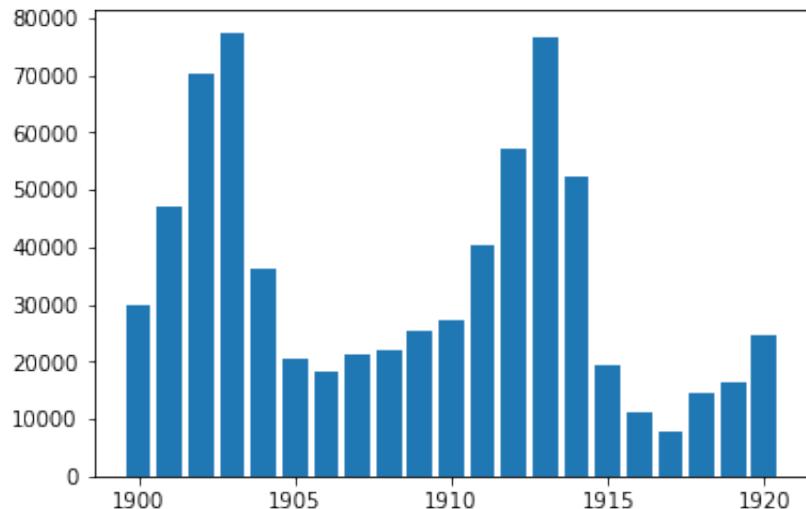
```
Out[4]: [
```



Чтобы построить столбчатую диаграмму - меняем функцию `plot` на `bar`. Все виды графиков и функций можно найти в документации [matplotlib](https://matplotlib.org/api/pyplot_summary.html) (https://matplotlib.org/api/pyplot_summary.html)

```
In [5]: plt.bar(forest.year, forest.hare)
```

```
Out[5]: <BarContainer object of 21 artists>
```



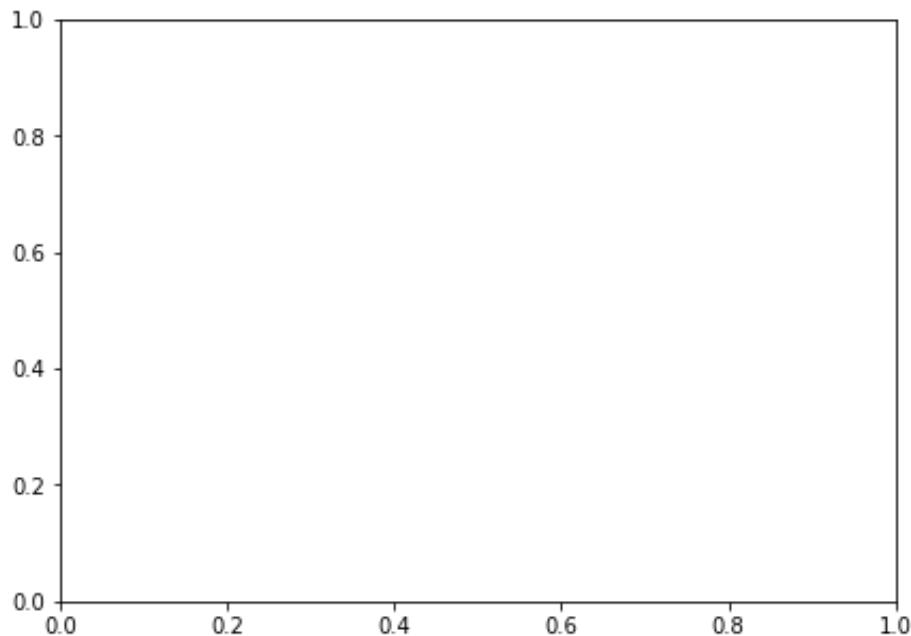
Давайте вернемся к нашему линейному графику, сохраним его в переменной и попробуем сделать его посимватичней.

В идеальной вселенной мы создаем графики функцией `subplots` (которая может, например, создать систему из нескольких графиков) и сложить то, что она генерирует в две переменные. Обычно их называют `fig` и `ax`.

Переменная `fig` (или как вы ее назвали. Тут главное, что это первая переменная, которую вы определили) отвечает за график в целом. Воспринимайте ее как файл, который хранит график как картинку.

Переменная `ax` (так, которую мы определяем второй) - это ось координат, на которой мы собственно строим график. Мы можем настраивать внешний вид этого объекта (потому что все элементы графика хранятся как раз в ней).

```
In [6]: fig, ax = plt.subplots(1,1, figsize=(7,5)) # создали объект из 1 ряда 1 колонки графика (что сюрприз - один график!)
# Размер - по сути это размер нашего прямоугольника в неочевидных единицах.
# Какой размер удачный?
экспериментируйте!
```



Выше мы создали пустую шкалу координат. Давайте теперь попробуем наложить на нее график.

```
In [7]: forest.head() # вспомнили, какие переменные у нас есть.
```

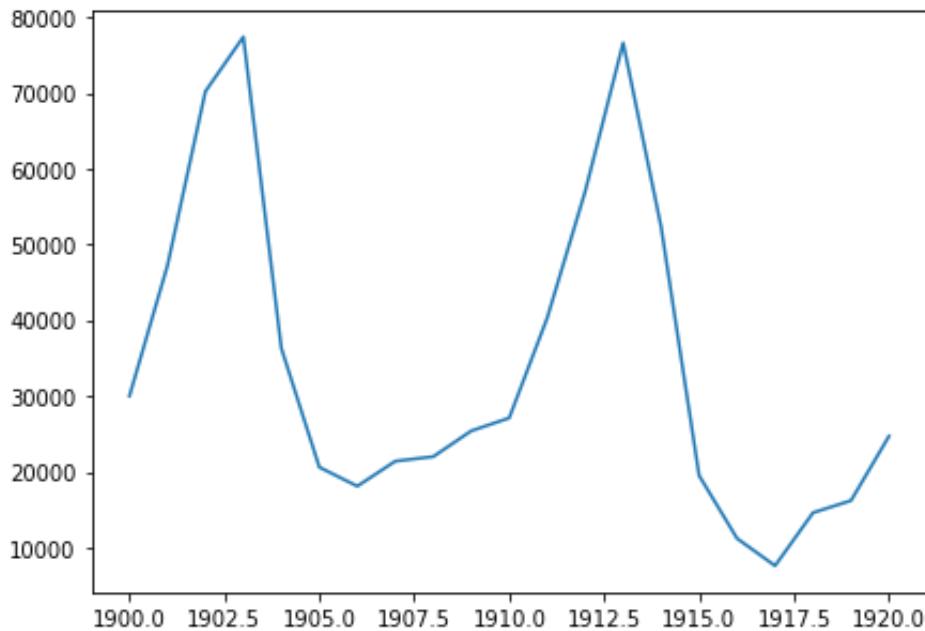
Out[7]:

	year	hare	lynx	carrot
0	1900	30000.0	4000.0	48300
1	1901	47200.0	6100.0	48200
2	1902	70200.0	9800.0	41500
3	1903	77400.0	35200.0	38200
4	1904	36300.0	59400.0	40600

Посмотрим, как выглядит наш график по умолчанию.

```
In [8]: fig, ax = plt.subplots(1,1, figsize=(7,5)) # создаем переменную заново, чтобы вы точно стирали то, что в ней лежит на каждом шаге,  
# если вы захотите внести изменения в код  
ax.plot(forest.year, forest.hare) # обратите внимание - график мы теперь строим как метод объекта ax.  
# Ниже все модификации графика тоже применяем к объекту.
```

Out[8]: [`<matplotlib.lines.Line2D at 0x1b925906898>`]



```
In [9]: fig, ax = plt.subplots(1,1, figsize=(7,5))

# параметры самого графика (цвет линии, стиль и т.д.) определяем как
# параметры в методе plot()
# меняем цвет и стиль линии на пунктир. Matplotlib знает некоторые
# стандартные цвета, и их можно задать прямо словом
# так же можно передать hex цвет. Например, #8c92ac
ax.plot(forest.year, forest.hare, color = 'grey', ls = ':')

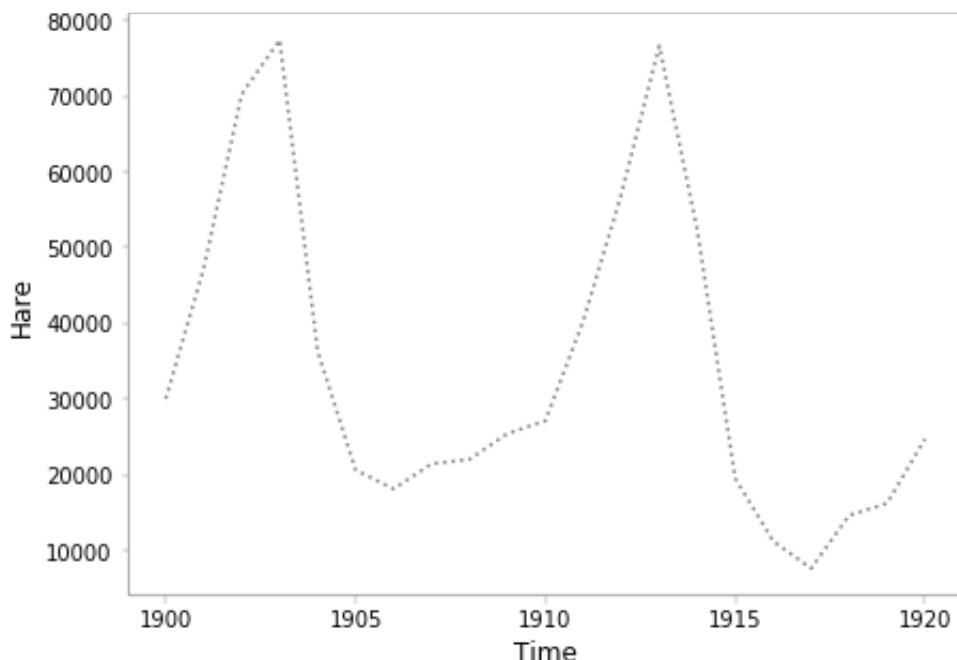
# если вы обратили внимание, то в нашем самом первом графике шкала
# с годами сломалась и стала float. Matplotlib принудительно
# делает x непрерывной переменной для линейного графика. Мы хотим о
#ставить шкалу год в целых числах.
ax.locator_params(integer=True)

# называем шкалы x и y, выбираем размер шрифта.
ax.set_xlabel('Time', fontsize=12)
ax.set_ylabel('Hare', fontsize=12)

# делаем правую и верхнюю границу графика невидимыми
ax.spines['right'].set_visible(False)
ax.spines['top'].set_visible(False)

# делаем засечки на шкалах x и у потоньше
ax.xaxis.set_tick_params(width=0.2)
ax.yaxis.set_tick_params(width=0.2)

# уменьшаем толщину оставших границ графика с помощью цикла (можно
# и без цикла отдельной строкой для каждой границы, как делали выше)
for spine in ['bottom','left']:
    ax.spines[spine].set_linewidth(0.2)
```

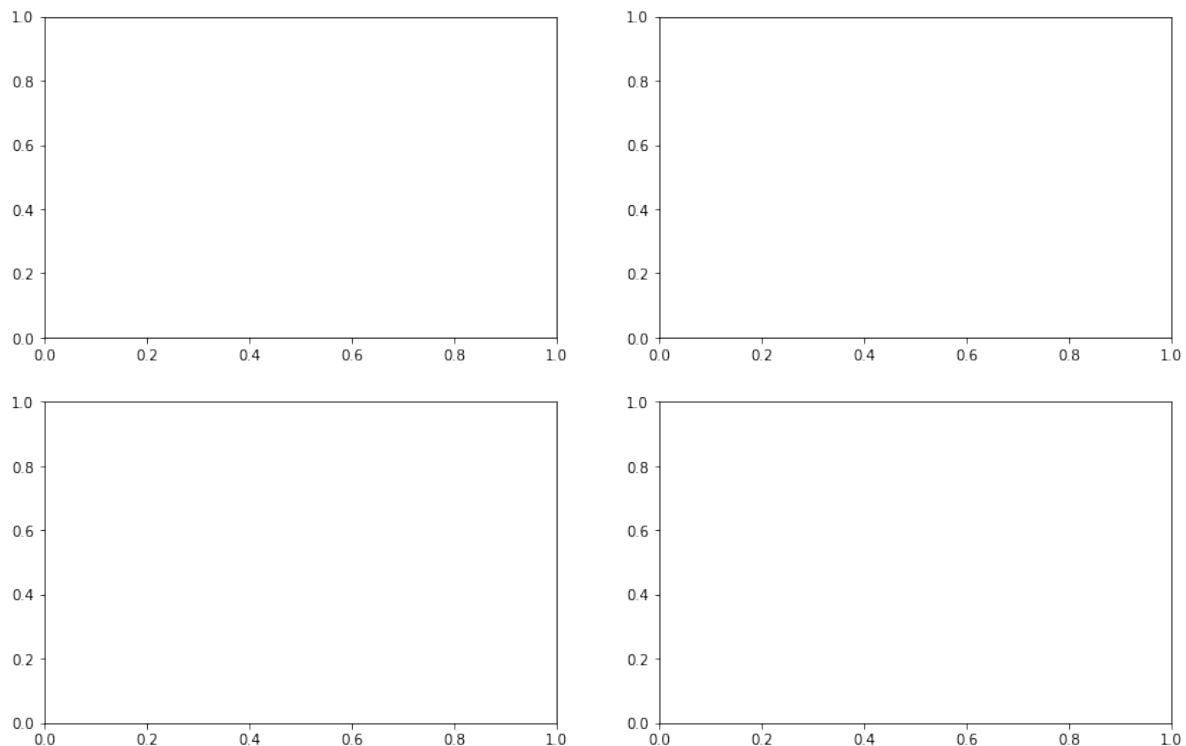


Больше видов графиков и параметров, которые мы можем изменять - в документации Matplotlib.

Упражнение

Сейчас мы посмотрим, как создавать систему графиков и попробуем построить тренды для всех обитателей нашего леса.

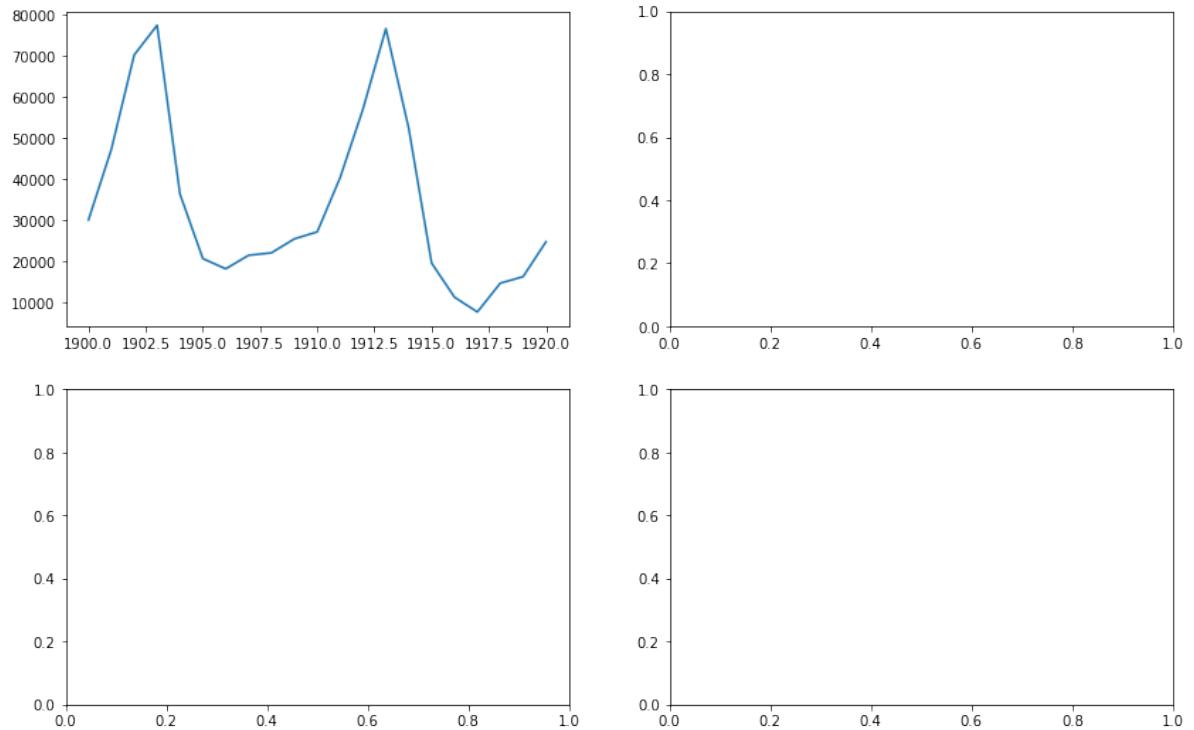
In [10]: `fig, ax = plt.subplots(2,2, figsize=(14,9)) # увеличиваем количество объектов до двух рядов и двух колонок. Также увеличиваем размер изображения.`



Класс, теперь у нас есть четыре графика. Теперь объект `ax` - это некоторая матрица (или вложенный список - как вам удобнее воспринимать). И чтобы обратиться к каждому графику, нужно обратиться к нему по индексу.

```
In [11]: fig, ax = plt.subplots(2,2, figsize=(14,9))
ax[0][0].plot(forest.year, forest.hare) # Обращаемся к первому графику в первой колонке.
```

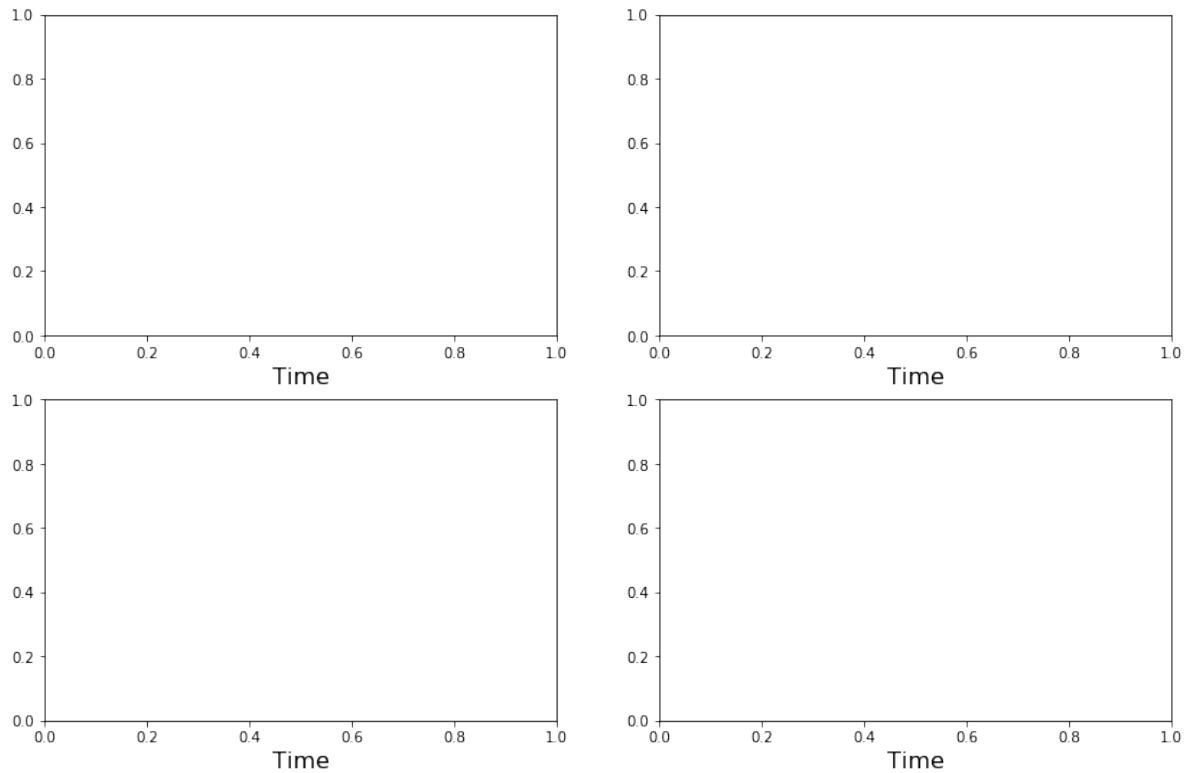
```
Out[11]: [
```



Ниже посмотрим как с помощью цикла `for` можно применить какую-то модификацию ко всем графикам системы.

```
In [12]: fig, ax = plt.subplots(2, 2, figsize=(14,9))

for x in range(2):
    for y in range(2):
        ax[x][y].set_xlabel('Time', fontsize = 16)
```



Задание (10 минут):

- Нарисуйте во втором графике изменения в популяции рысей.
- В третьем - морковки.
- В четвертом - все три графика вместе (подумайте как).
- Каждый график должен быть разного цвета. В четвертом графике - каждый график должен быть такого же цвета, как и в своей ячейке.
- Подпишите шкалы у и х для каждого графика.

Если вы знаете цикл for - попробуйте написать цикл, который бы массово применял изменения во внешнем виде графика (например, удалял бы верхние и правые границы у всех графиков)

Пример решения:

```
In [12]: fig, ax= plt.subplots(2, 2, figsize=(14,9))

for x in range(2):
    for y in range(2):
        ax[x][y].set_xlabel('Time', fontsize=14)
        ax[x][y].locator_params(integer=True)
        ax[x][y].spines['right'].set_visible(False)
        ax[x][y].spines['top'].set_visible(False)
        ax[x][y].xaxis.set_tick_params(width=0.2)
        ax[x][y].yaxis.set_tick_params(width=0.2)
        for axis in ['top','bottom','left','right']:
            ax[x][y].spines[axis].set_linewidth(0.2)

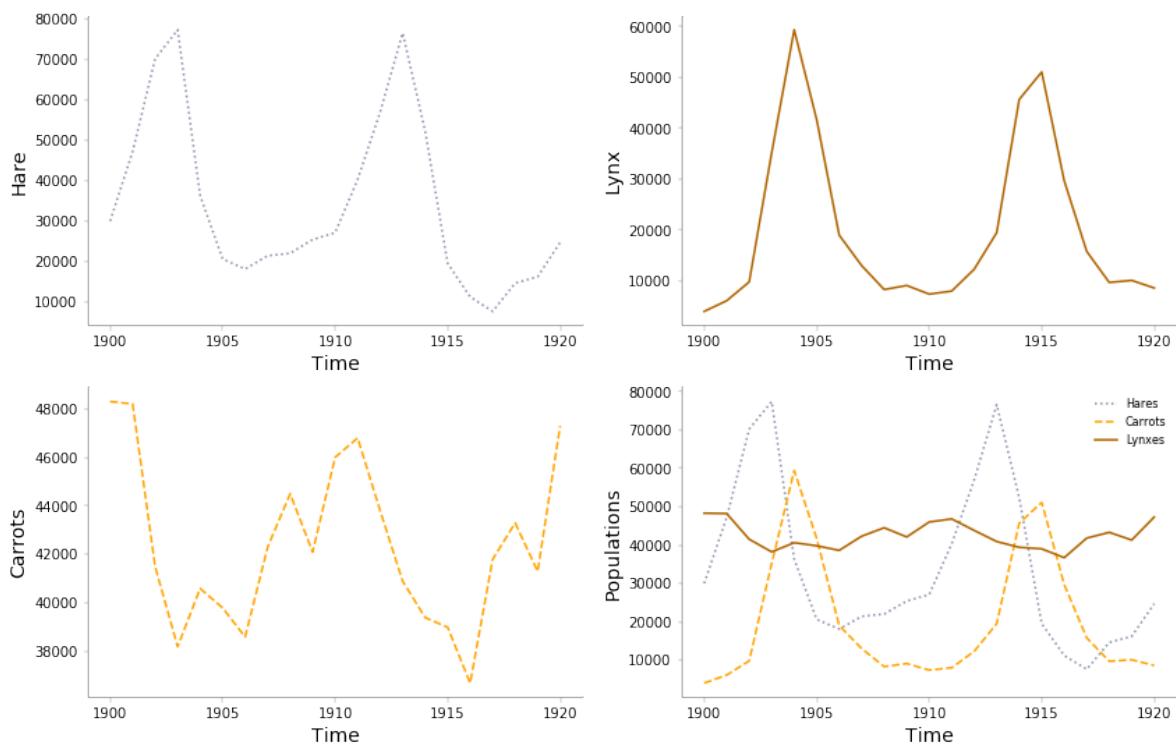
ax[0][0].plot(forest.year, forest.hare, color="#8c92ac", ls = ':')
ax[0][0].set_ylabel('Hare', fontsize=14)

ax[1][0].plot(forest.year, forest.carrot, color="#ffa500", ls = '--')
ax[1][0].set_ylabel('Carrots', fontsize = 14)

ax[0][1].plot(forest.year, forest.lynx, color="#b06500", ls = '-');
ax[0][1].set_ylabel('Lynx', fontsize = 14)

ax[1][1].plot(forest.year, forest.hare, label = 'Hares', color="#8c92ac", ls = ':')
ax[1][1].plot(forest.year, forest.lynx, label = 'Carrots', color="#ffa500", ls = '--')
ax[1][1].plot(forest.year, forest.carrot, label = 'Lynxes', color="#b06500", ls = '-')
ax[1][1].set_ylabel('Populations', fontsize = 14)
ax[1][1].legend(loc=1, fontsize=8, frameon=False) # задаем легенду.
loc отвечает за ее местоположение (экспериментируйте!), frameon - наличие рамки вокруг легенды.
```

Out[12]: <matplotlib.legend.Legend at 0x1e3314a09b0>



In [13]:

```
# сохраним график. Если не зададите путь - сохранится в ту папку,
которую сейчас jupyter считает рабочей
# (то, что вы видете на первом экране в браузере, когда запускаете
jupyter)
fig.savefig("my_new_plot.png")
```

Мультивариативный график рассеяния (multivariate scatter plot): преступления в США

Сейчас будем работать с набором данных, который содержит информацию о количестве преступлений в штатах США в 2005 году.

```
In [3]: crimes = pd.read_csv('https://raw.githubusercontent.com/rogovich/2020_MIREC_PfDA/master/Seminars/S5_S6_Matplotlib/crimeRatesByState2005.tsv', sep='\t') # тут разделитель - тоже табуляция
crimes.head()
```

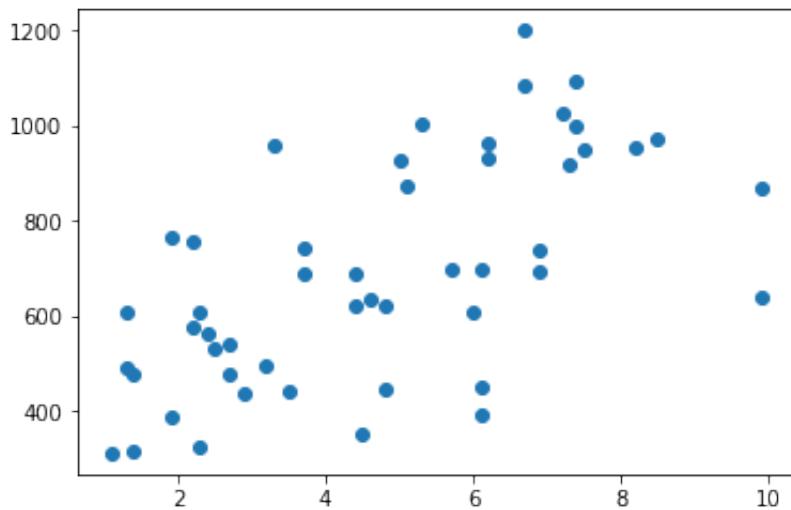
Out[3]:

	state	murder	Forcible_rate	Robbery	aggravated_assult	burglary	larceny_theft	...
0	Alabama	8.2	34.3	141.4		247.8	953.8	2650.0
1	Alaska	4.8	81.1	80.9		465.1	622.5	2599.1
2	Arizona	7.5	33.8	144.4		327.4	948.4	2965.2
3	Arkansas	6.7	42.9	91.1		386.8	1084.6	2711.2
4	California	6.9	26.0	176.1		317.3	693.3	1916.5

Давайте начнем с графика с двумя переменными. Тут все просто - непрерывная переменная по x, непрерывная переменная по y: а точка на пересечении значений x и y - место нашего штата в этой системе координат. Давайте для начала построим график для убийств и ограблений. График рассеяния можно использовать и для категориальных переменных - посмотрим на другом примере.

```
In [15]: fig, ax = plt.subplots() # если нужно создать один график, то можем не указывать количество в аргументах
ax.scatter(crimes['murder'], crimes['burglary']) # вместо метода plot используем scatter
```

Out[15]: <matplotlib.collections.PathCollection at 0x1b925cce240>

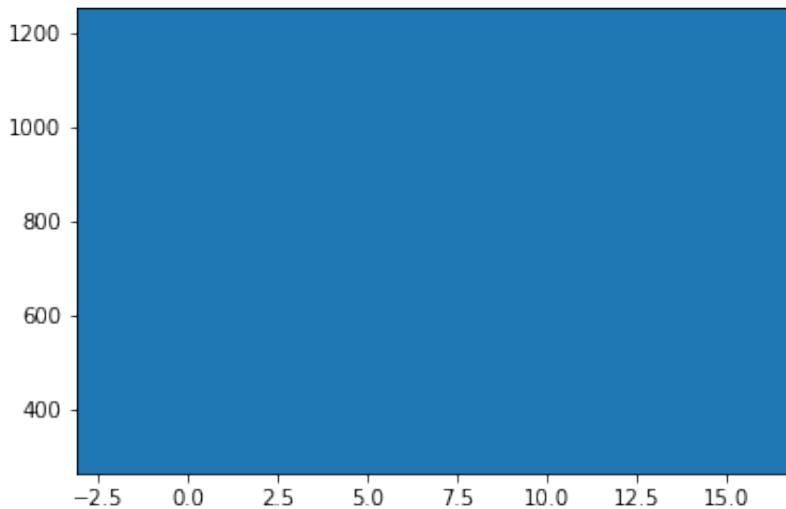


Попробуйте проинтерпретировать график выше. Дальше мы попробуем закодировать в нем еще две переменные. Есть идеи как?

Мультивариативный график - это такой график, из которого мы можем считать значения для более чем двух переменных. Самый простой способ закодировать дополнительную информацию - цвет и размер точки. Давайте добавим значение переменной `population` (размер населения штата) на график как размер точки.

```
In [16]: fig, ax = plt.subplots()  
  
# добавляем параметр s (size) и говорим, какая переменная будет за  
# него отвечать  
ax.scatter(crimes['murder'], crimes['burglary'], s = crimes['popula  
tion'])
```

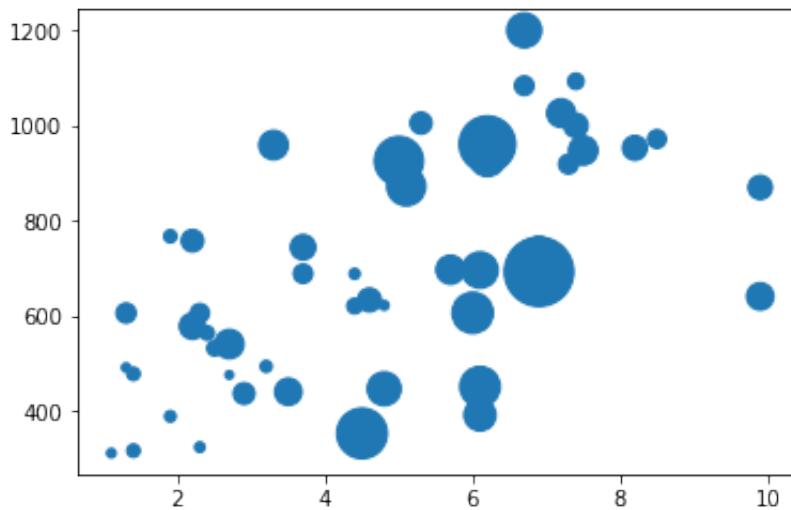
Out[16]: <matplotlib.collections.PathCollection at 0x1b925f65828>



Во! Размер населения такой большой, что точка захватила всю область координат. Давайте попробуем нашу переменную масштабировать - нам же важны относительные размеры штатов относительно друг друга, а не абсолютные значения. Значения масштабирования тоже выбираем экспериментально: то, что лучше выглядит и более информативно.

```
In [17]: fig, ax = plt.subplots()
ax.scatter(crimes['murder'], crimes['burglary'], s = crimes['population']/35000)
```

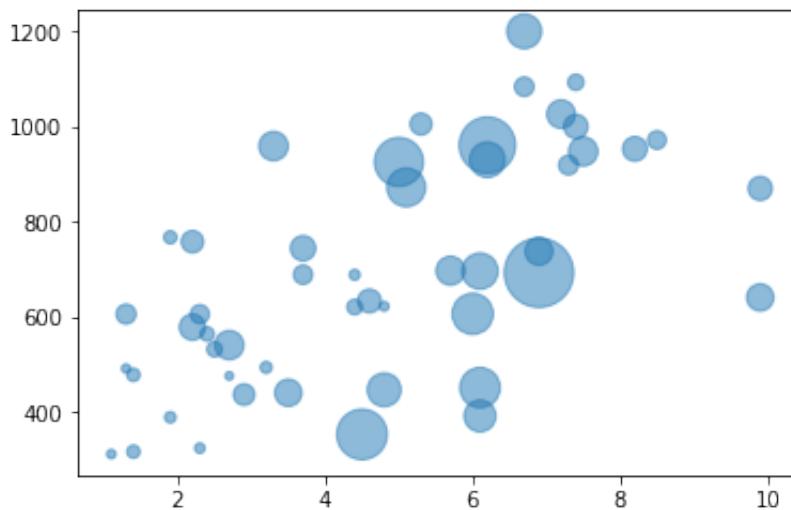
```
Out[17]: <matplotlib.collections.PathCollection at 0x1b925fbae80>
```



Класс, только некоторые точки слились. Давайте добавим параметр прозрачности, чтобы было видно, где они накладываются друг на друга.

```
In [18]: fig, ax = plt.subplots()
ax.scatter(crimes['murder'], crimes['burglary'], s = crimes['population']/35000, alpha = 0.5) # параметр alpha задает прозрачность точки от 0 до 1
```

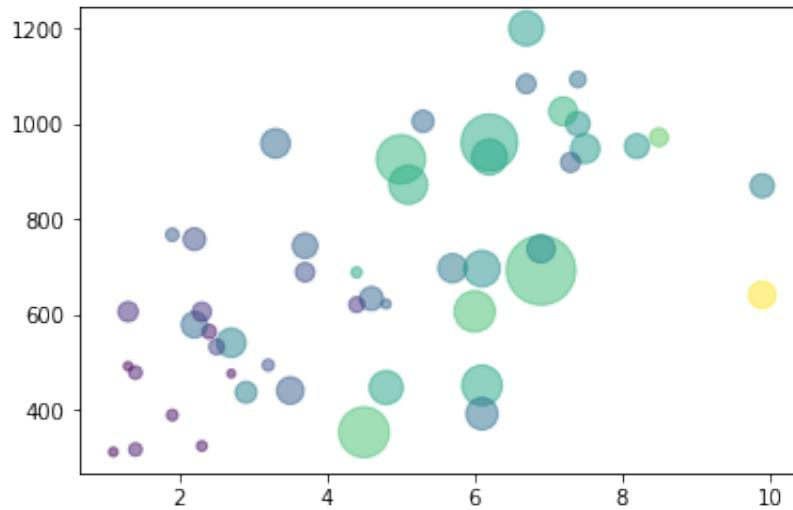
```
Out[18]: <matplotlib.collections.PathCollection at 0x1b9260700b8>
```



Давайте попробуем проинтерпретировать этот график. А потом добавим еще какую-нибудь переменную в виде цвета.

```
In [19]: fig, ax = plt.subplots()
ax.scatter(crimes['murder'], crimes['burglary'], s = crimes['population']/35000, alpha = 0.5,
           c = crimes['Robbery']) # задаем новый аргумент с (color)
и присваиваем ему значение переменной
```

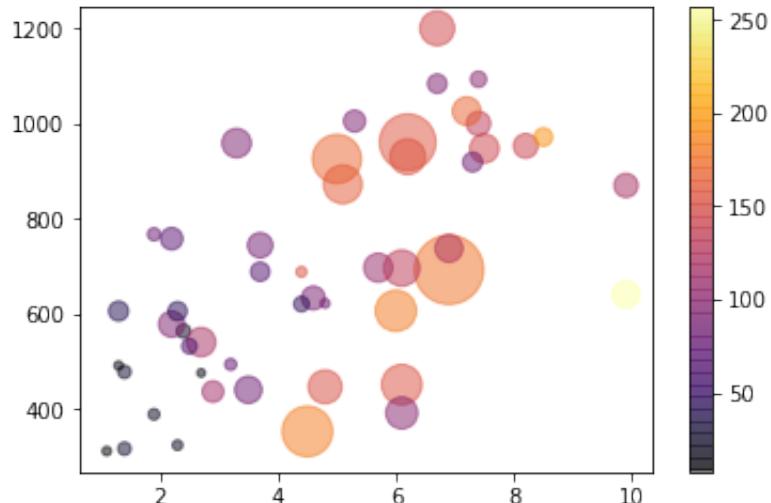
Out[19]: <matplotlib.collections.PathCollection at 0x1b9260bd748>



Осталось узнать, что значит какой цвет. Для этого нужно сохранить график в переменную и передать ее как аргумент функции colorbar(). Также можем поменять цветовую шкалу с помощью аргумента cmap (какие цветовые карты есть в matplotlib? Гуглим документацию!).

```
In [20]: fig, ax = plt.subplots()
color_graph = ax.scatter(crimes['murder'], crimes['burglary'], s = crimes['population']/35000, alpha = 0.5, cmap = 'inferno',
                        c = crimes['Robbery'])
plt.colorbar(color_graph)
```

Out[20]: <matplotlib.colorbar.Colorbar at 0x1b925c4b320>



Проинтерпретируем?

Сделаем симпатичней и подпишем штаты

```
In [21]: fig, ax = plt.subplots(figsize = (22,10))

color_graph = ax.scatter(crimes['murder'], crimes['burglary'], s =
crimes['population']/35000, c = crimes['Robbery'], cmap = 'inferno'
, alpha = 0.5, linewidth = 0)

ax.spines['right'].set_visible(False)
ax.spines['top'].set_visible(False)

ax.yaxis.set_ticks_position('left')
ax.xaxis.set_ticks_position('bottom')

ax.spines['left'].set_linewidth(0.5)
ax.spines['bottom'].set_linewidth(0.5)

ax.set_xlabel('Murder', fontsize = 10)
ax.set_ylabel('Bulglarly', fontsize = 10)

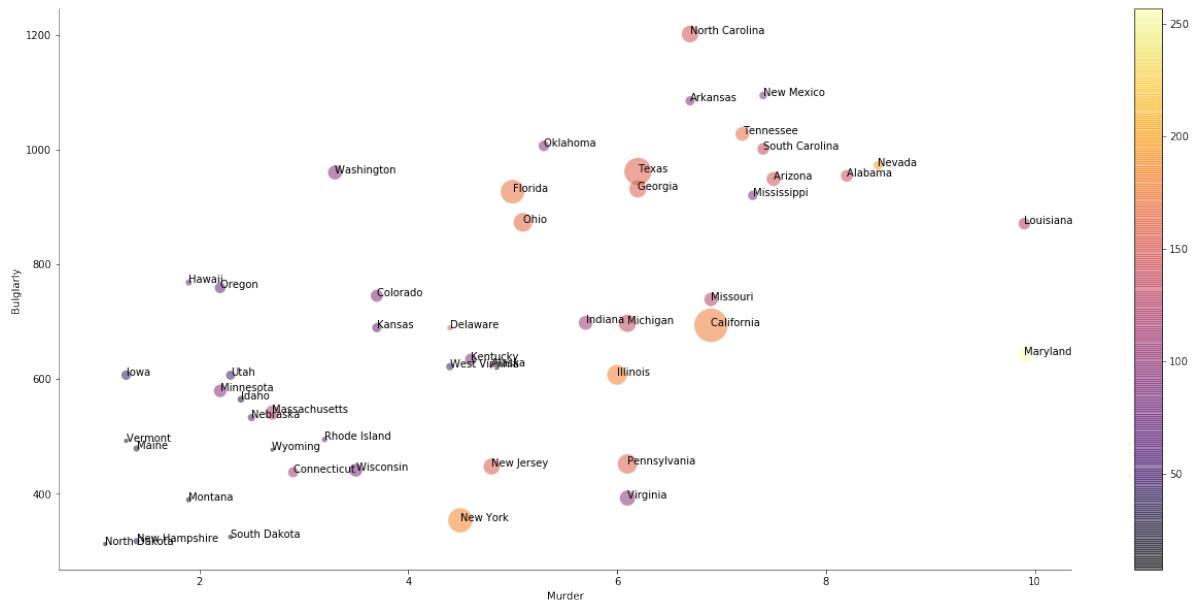
for i, state in enumerate(crimes['state']): # с помощью enumerate
    создаем из колонок с названиями штатов объект кортежей вида индекс
    - название штата.
    ax.annotate(state, (crimes['murder'][i], crimes['burglary'][i])
, fontsize = 10) # используем метод annotate, которому говорим, что
    нужно брать имя штата

# из кортежа, который создали с помощью enumerate, а координаты под
    писи задаем через

# значения наших переменных x и у для нужного индекса из объекта en
    umerate (обращается к нужному

# ряду в датафрейме)

plt.colorbar(color_graph)
plt.savefig('crimes.png') # сохраняем
```



```
In [22]: # СМОТРИМ ЧТО ВНУТРИ объекта enumerate()
list(enumerate(crimes['state']))
```

```
Out[22]: [(0, 'Alabama '),
(1, 'Alaska '),
(2, 'Arizona '),
(3, 'Arkansas'),
(4, 'California '),
(5, 'Colorado '),
(6, 'Connecticut '),
(7, 'Delaware '),
(8, 'Florida '),
(9, 'Georgia '),
(10, 'Hawaii '),
(11, 'Idaho '),
(12, 'Illinois '),
(13, 'Indiana '),
(14, 'Iowa '),
(15, 'Kansas '),
(16, 'Kentucky '),
(17, 'Louisiana '),
(18, 'Maine '),
(19, 'Maryland '),
(20, 'Massachusetts'),
(21, 'Michigan'),
(22, 'Minnesota '),
(23, 'Mississippi '),
(24, 'Missouri '),
(25, 'Montana '),
(26, 'Nebraska '),
(27, 'Nevada '),
(28, 'New Hampshire '),
(29, 'New Jersey'),
(30, 'New Mexico '),
(31, 'New York'),
(32, 'North Carolina '),
(33, 'North Dakota '),
(34, 'Ohio '),
(35, 'Oklahoma '),
(36, 'Oregon '),
(37, 'Pennsylvania'),
(38, 'Rhode Island '),
(39, 'South Carolina '),
(40, 'South Dakota '),
(41, 'Tennessee '),
(42, 'Texas '),
(43, 'Utah'),
(44, 'Vermont '),
(45, 'Virginia '),
(46, 'Washington '),
(47, 'West Virginia '),
(48, 'Wisconsin '),
(49, 'Wyoming ')]
```

График рассеяния на агрегированном датасете (строим графики в цикле)

Следующий небольшой пример сделаем на датасете проекта gapminder - это набор социо-экономических индикаторов для ряда стран за 200 лет. Мы возьмем небольшой срез набора данных с 1957 года с шагом пять лет и две переменные - ВВП на душу населения (gdp per capita) и продолжительность жизни.

www.gapminder.org

```
In [3]: gapminder = pd.read_csv('https://raw.githubusercontent.com/rogovich/2019_HSE_DPO_Python_for_data_analysis/master/lectures-seminars/10-30-2019_Viz/gapminderData.csv')
gapminder.head()
```

Out[3]:

	country	year	pop	continent	lifeExp	gdpPercap
0	Afghanistan	1952	8425333.0	Asia	28.801	779.445314
1	Afghanistan	1957	9240934.0	Asia	30.332	820.853030
2	Afghanistan	1962	10267083.0	Asia	31.997	853.100710
3	Afghanistan	1967	11537966.0	Asia	34.020	836.197138
4	Afghanistan	1972	13079460.0	Asia	36.088	739.981106

```
In [24]: gapminder.shape
```

Out[24]: (1704, 6)

```
In [25]: gapminder.year.values # смотрим, какие значения есть в колонке год
```

Out[25]: array([1952, 1957, 1962, ..., 1997, 2002, 2007], dtype=int64)

Сделаем колонку continent категориальной, потому что категории закодировать по строкам не получится

```
In [26]: gapminder['continent'] = pd.Categorical(gapminder['continent'])
```

In [27]: `gapminder['continent'].cat.codes.head() # теперь у нас есть такая числовая разметка категорий, которую мы сможем скормить переменной графика`

Out[27]:

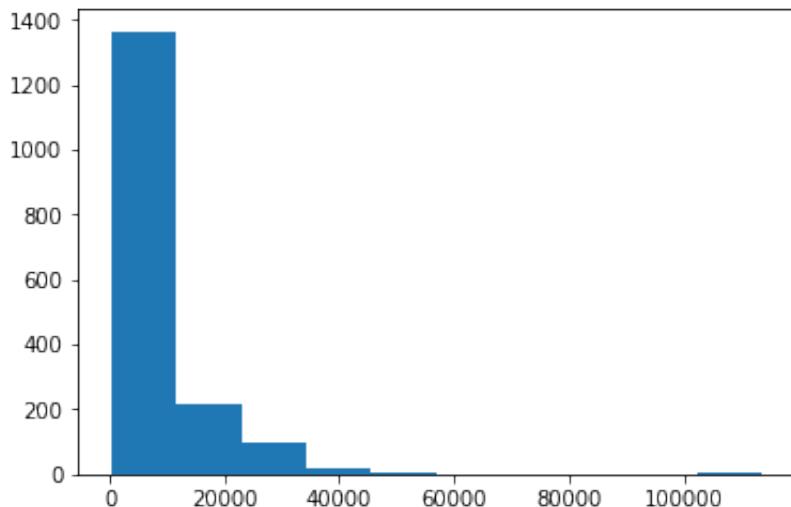
0	2
1	2
2	2
3	2
4	2
dtype: int8	

Еще мы хотим логарифмировать переменную gdp. Многие переменные, связанные, например, с доходом имеют логарифмическое распределение (что логично – у нас очень много людей или стран с каким-то небольшим достатком и есть очень длинный хвост наблюдений с большими значениями по шкале).

In [28]: `plt.hist(gapminder['gdpPercap'])`

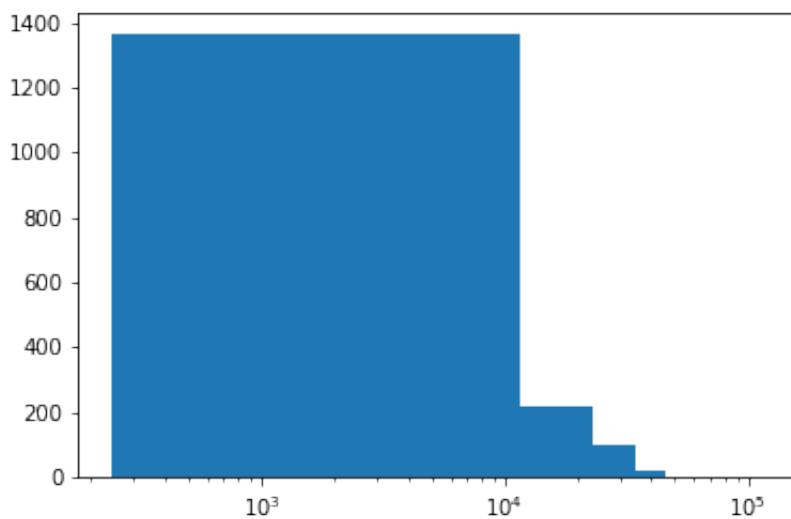
Out[28]:

```
(array([1.365e+03, 2.150e+02, 9.600e+01, 1.900e+01, 3.000e+00, 1.000e+00,
       0.000e+00, 1.000e+00, 1.000e+00, 3.000e+00]),
 array([ 241.1658765 , 11569.36257885, 22897.5592812 , 34225.75598355,
        45553.9526859 , 56882.14938825, 68210.3460906 , 79538.54279295,
        90866.7394953 , 102194.93619765, 113523.1329      ]),
 <a list of 10 Patch objects>)
```



С логарифмированием шкалы мы избавимся от длинного хвоста (ниже посмотрим, как это влияет на график.) Насчет логарифмированных шкал нужно волноваться в статистических моделях, потому что от этого у вас поменяется интерпретация коэффициентов. Сейчас нам достаточно понимать, что у стран, находящихся правее по шкале - ВВП на душу населения выше.

```
In [29]: plt.hist(gapminder[ 'gdpPercap' ])
plt.xscale('log')
```



Упражнение

Сейчас в наборе данных у нас есть каждая страна в каждый год с 1952 по 2007. Давайте сделаем срез для 1952 года и попробуем построить график рассеяния (почти то же самое, что делали выше):

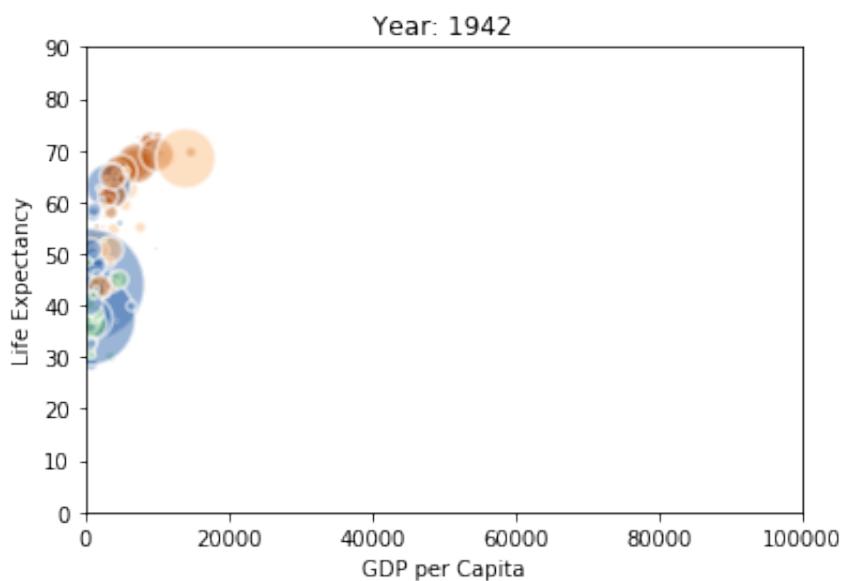
- x - доход
- y - продолжительность жизни
- размер - население
- цвет - континент

Подпишем шкалы x и y.

Пример без лог-преобразования шкалы x

```
In [30]: tmp = gapminder[gapminder.year == 1952 ]
plt.scatter(tmp['gdpPercap'], tmp['lifeExp'], s = tmp['pop']/200000
,
           c=tmp['continent'].cat.codes, cmap= "Accent", alpha=0.5
,
           edgecolors="white", linewidth=2) # добавили два новых аргумента - цвет и обводка границ точек.
plt.xlabel("GDP per Capita")
plt.ylabel("Life Expectancy")
plt.title("Year: "+str(1942)) # заголовок графика
plt.ylim(0, 90) # давайте еще добавим лимиты шкал, чтобы шкала не менялась, когда мы будем строить графики для других годов
plt.xlim(0,100000)
```

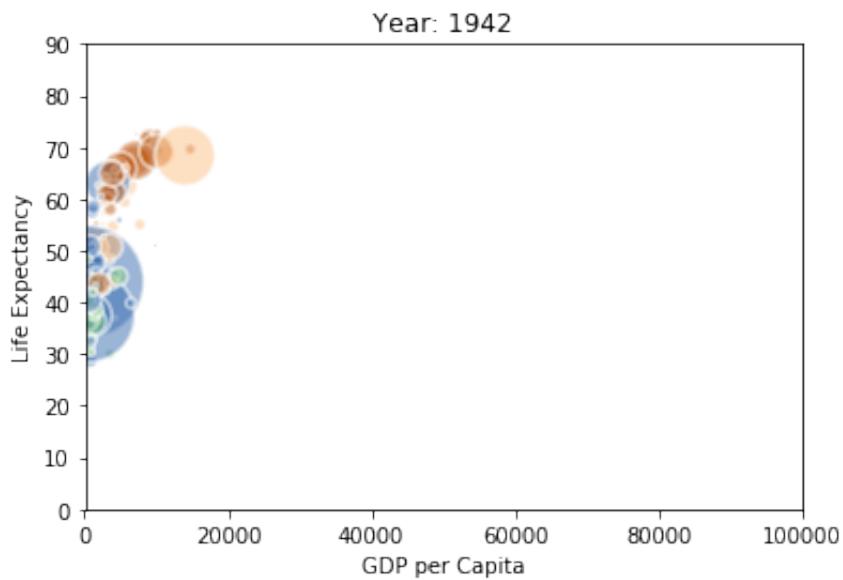
Out[30]: (0, 100000)



То же самое, но с лог-преобразованием

```
In [31]: tmp = gapminder[gapminder.year == 1952 ]
plt.scatter(tmp['gdpPerCap'], tmp['lifeExp'], s = tmp['pop']/200000
,
           c=tmp['continent'].cat.codes, cmap= "Accent", alpha=0.5
, edgecolors="white", linewidth=2)
plt.xlabel("GDP per Capita")
plt.ylabel("Life Expectancy")
plt.title("Year: "+str(1942) )
plt.ylim(0, 90)
plt.xlim(100,100000) # меняем лимиты для лог-шкалы
```

Out[31]: (100, 100000)



А теперь давайте автоматизируем эту красоту, чтобы получить график для каждой пятилетки с 1952 до 2007. Потом при желании мы можем их собрать в gif в какой-нибудь программе и получить анимированный график.

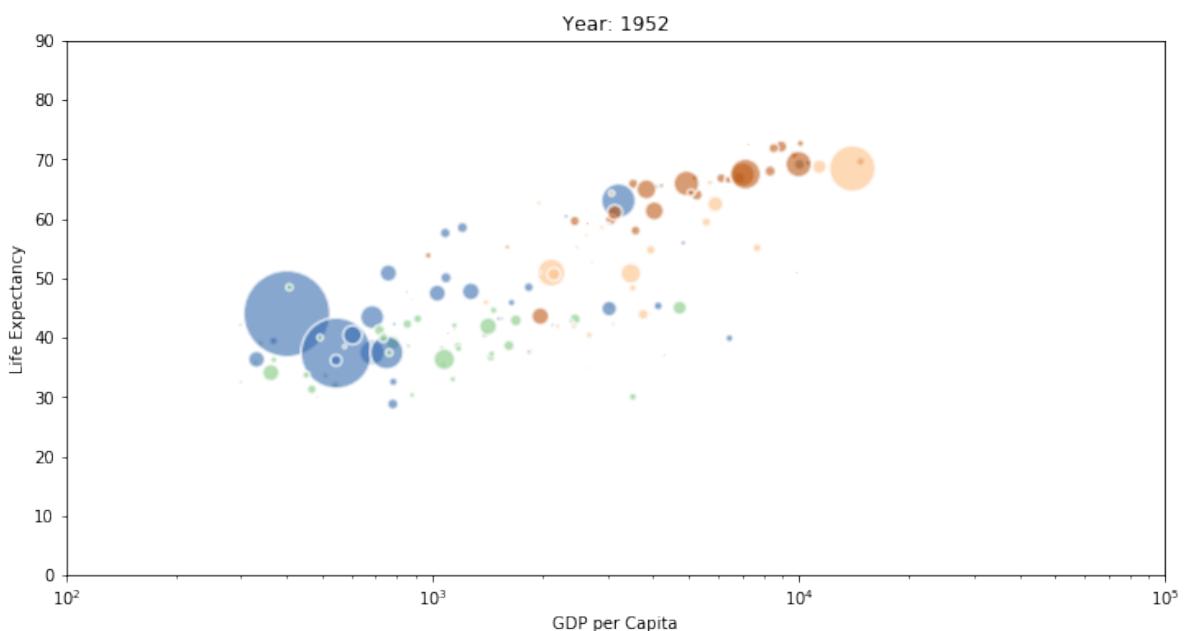
```
In [32]: for i in gapminder.year.unique(): # пишем цикл, который проходит по всем уникальным значениям в колонке год

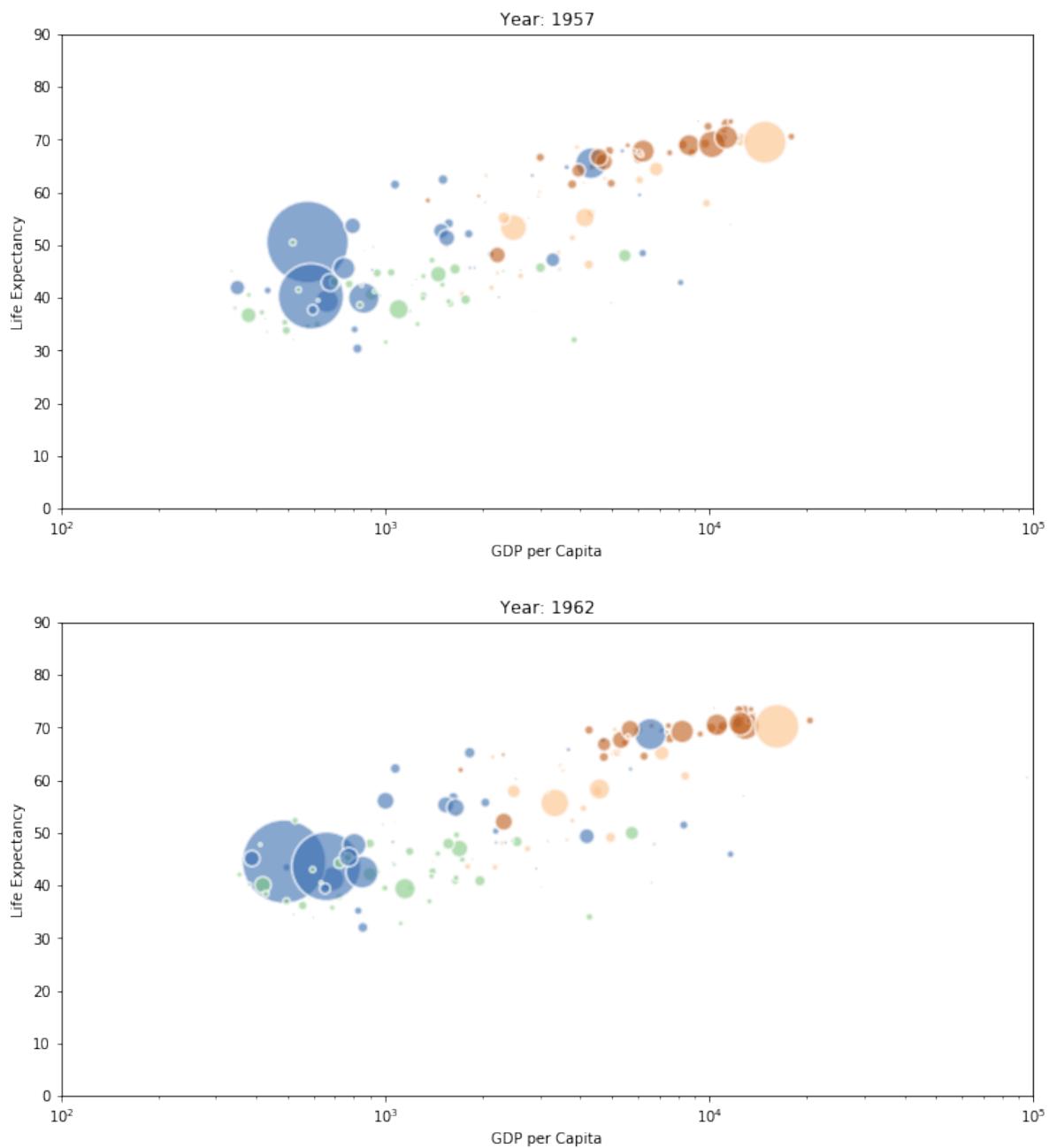
    # создаем фигуру
    fig = plt.figure(figsize=(12, 6))

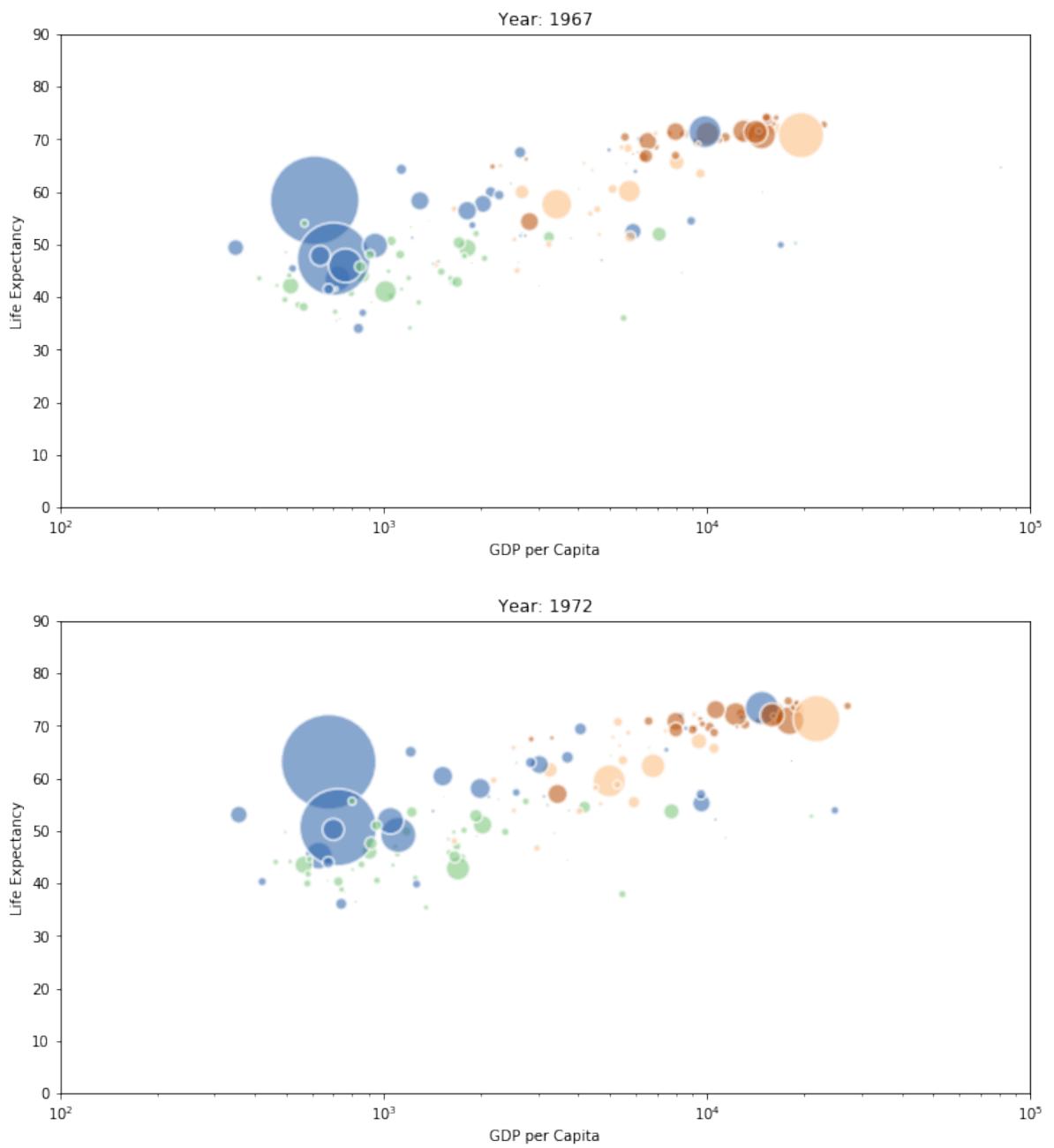
    tmp = gapminder[gapminder.year == i] # создаем срез датафрейма для года на данной итерации
    plt.scatter(tmp['gdpPercap'], tmp['lifeExp'], s=tmp['pop']/200000,
                c=tmp['continent'].cat.codes, cmap="Accent", alpha=0.6, edgecolors="white", linewidth=2)

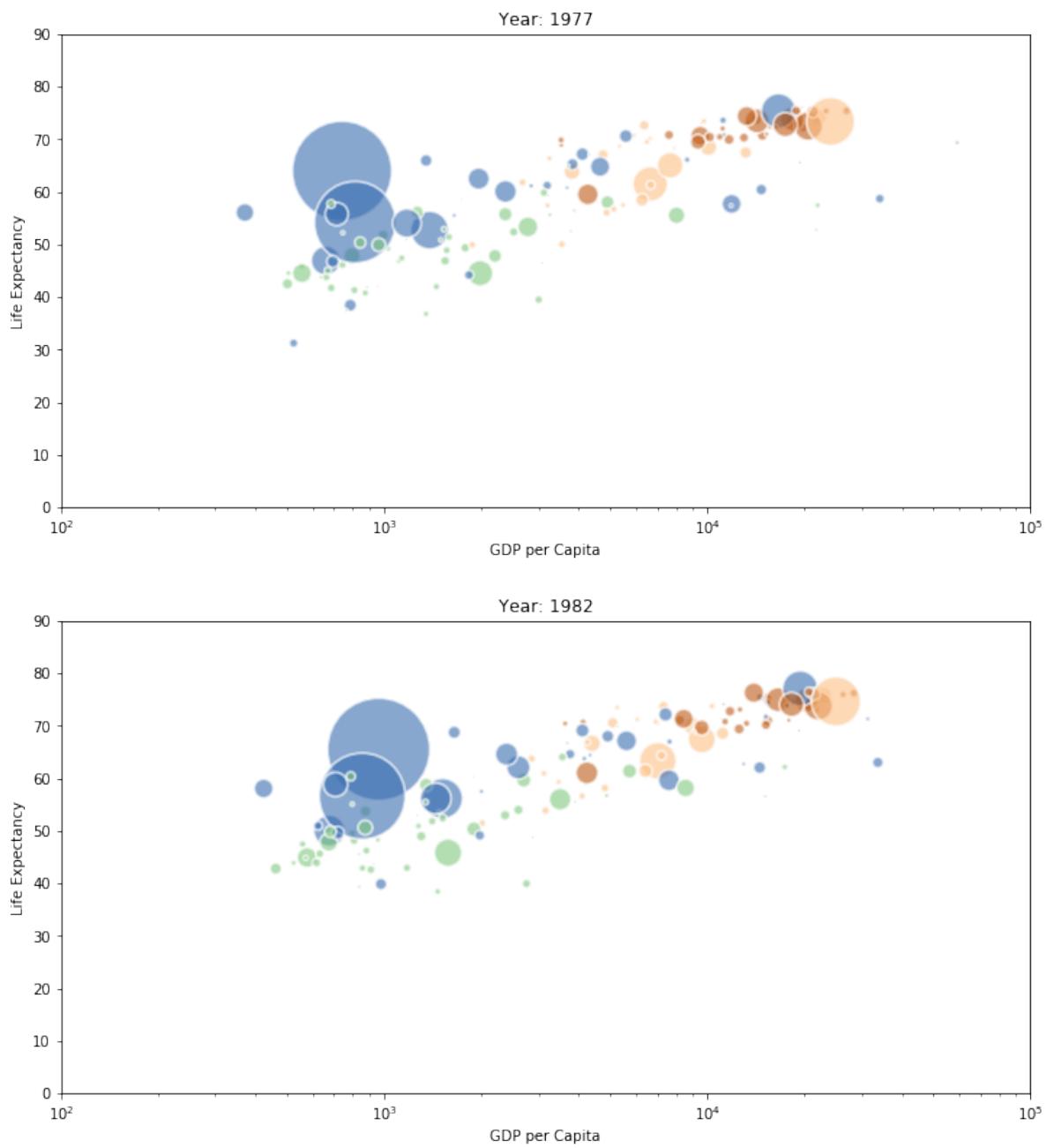
    plt.xscale('log')
    plt.xlabel("GDP per Capita")
    plt.ylabel("Life Expectancy")
    plt.title("Year: "+str(i))
    plt.ylim(0, 90)
    plt.xlim(100,100000)

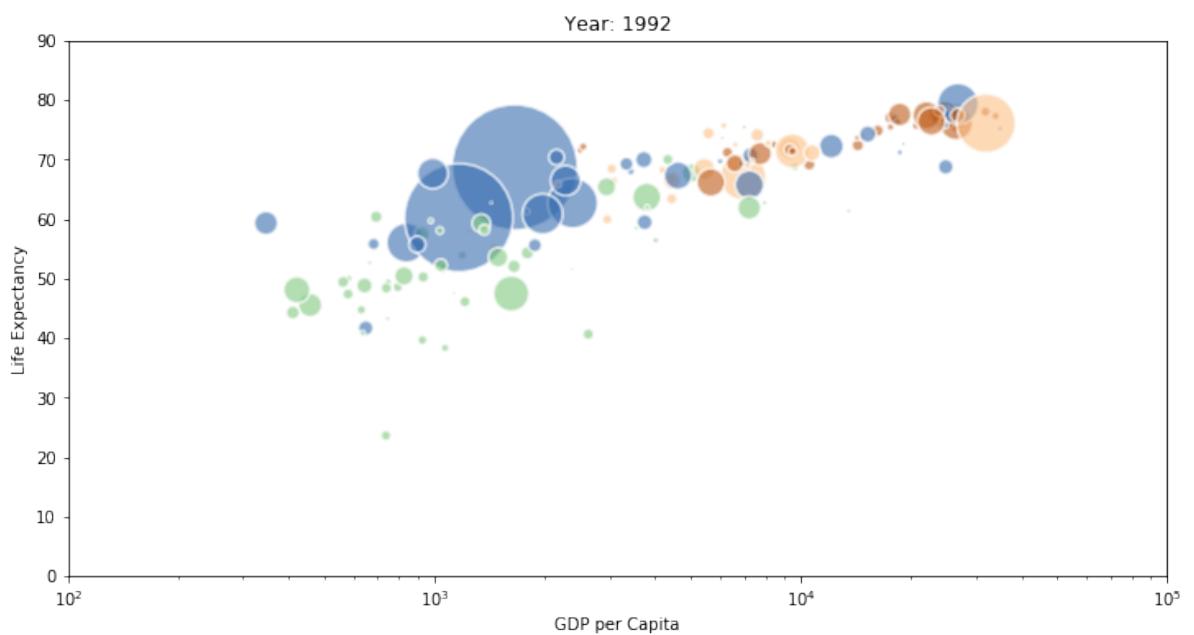
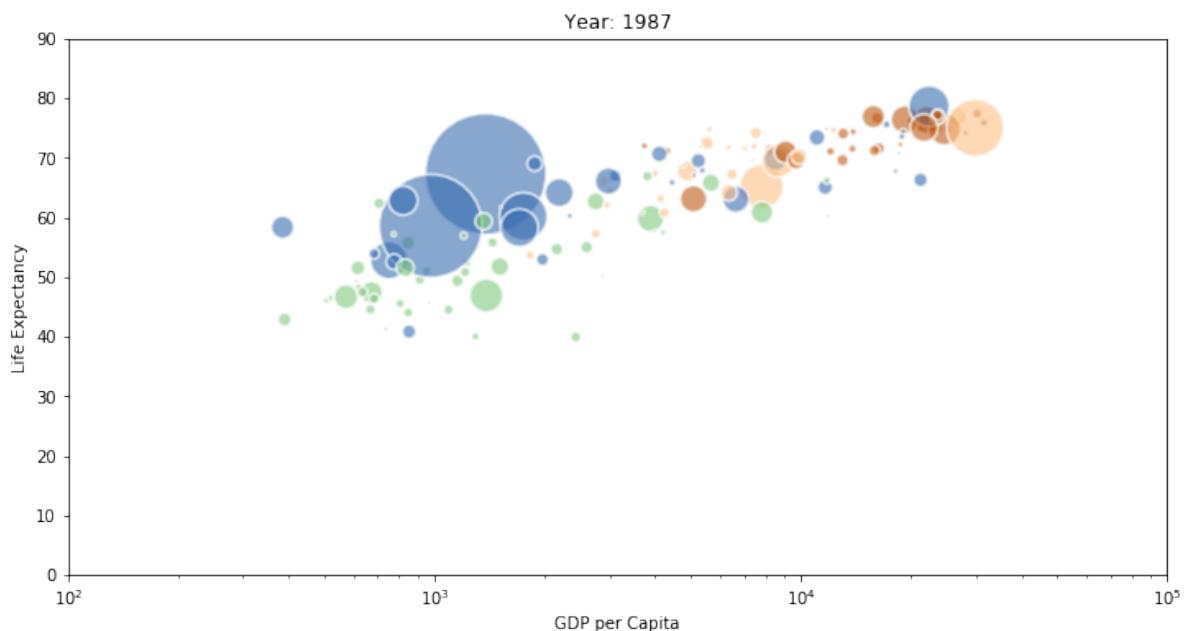
    # Сохраняем каждый график под уникальным именем.
    filename='Gapminder_step'+str(i)+'.png'
    plt.savefig(filename, dpi=96)
```

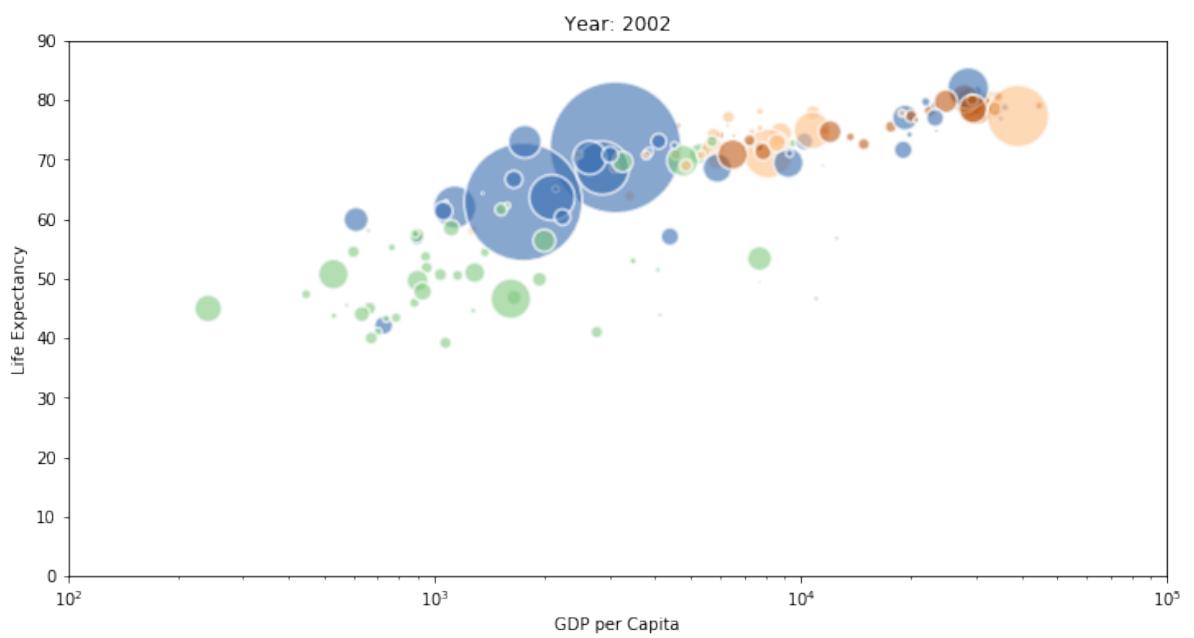
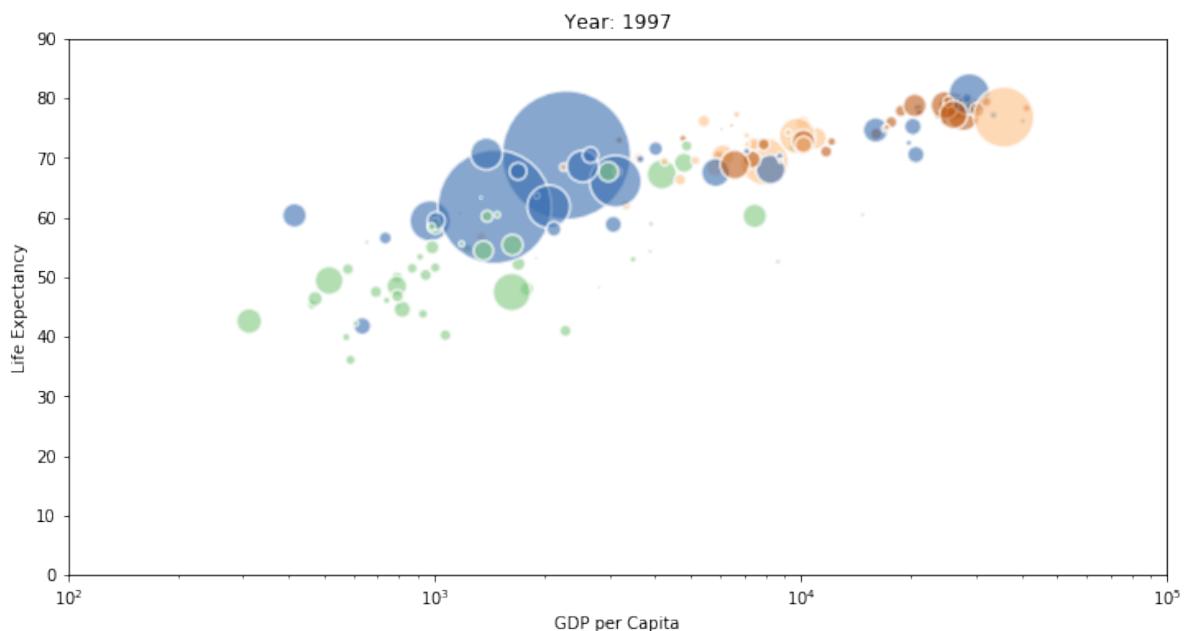


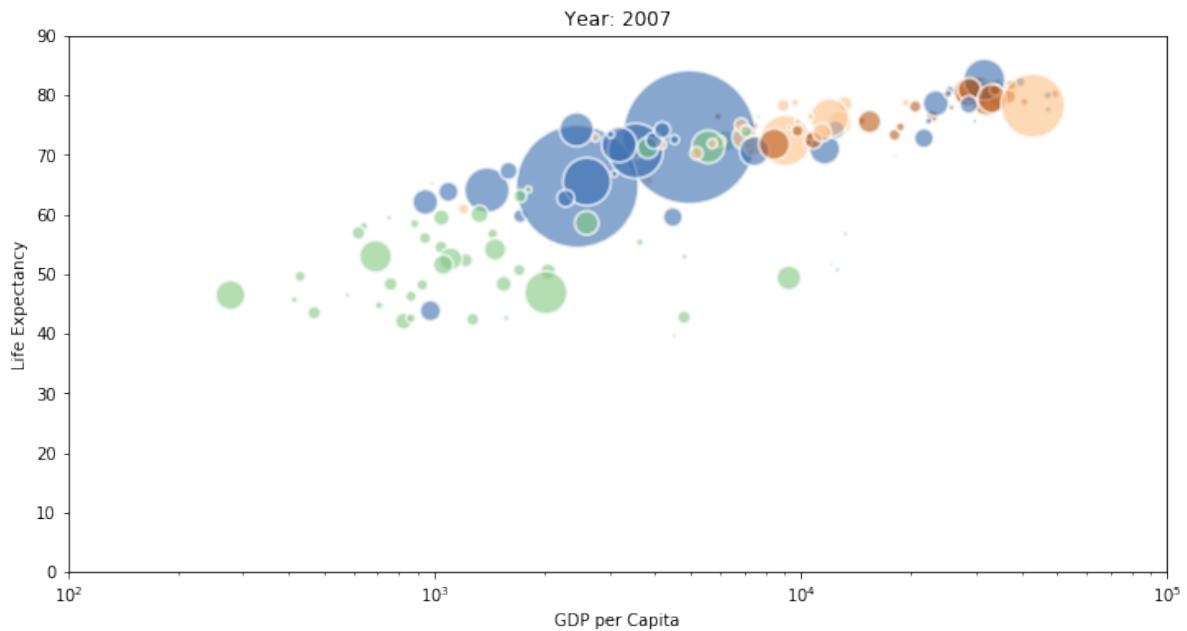












Интерактивный Gapminder с Plotly Express

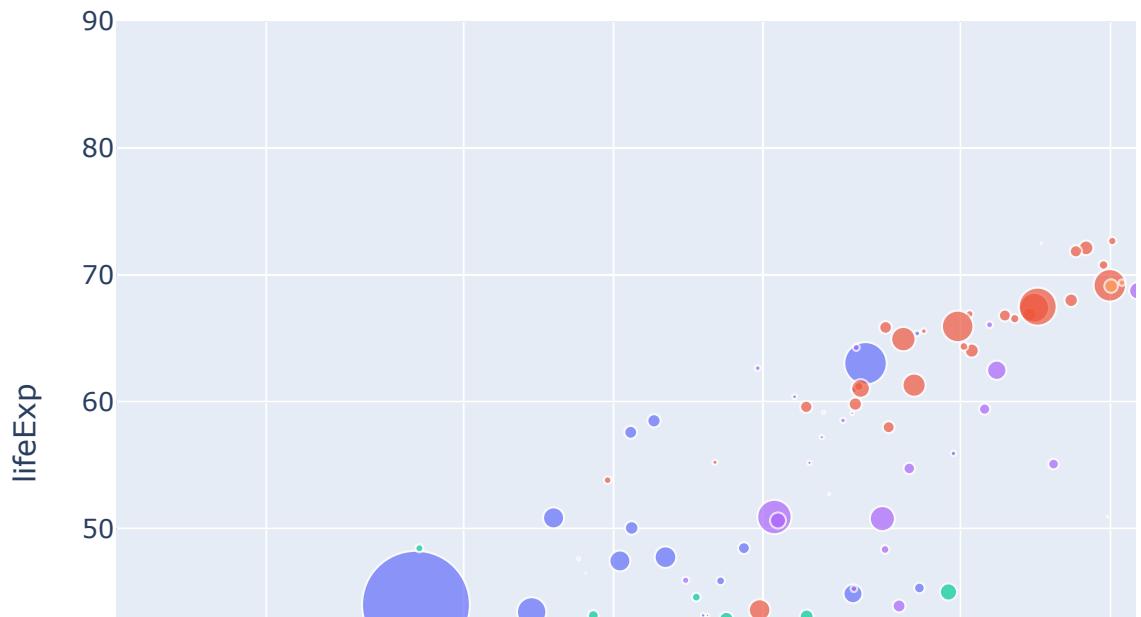
Мы, к сожалению, не успеваем поработать с библиотекой Plotly на наших занятиях, но можно самостоятельно разобраться с помощью блокнота [здесь](#)

(https://github.com/rogovich/2020_MIREC_PfDA/blob/master/Seminars/S5_S6_Matplotlib/Bonus/2020_MI)

Plotly - библиотека для построения интерактивных графиков с возможностью очень точечной настройки параметров. Давайте для ознакомления построим интерактивный график для gapminder с помощью модуля plotly.express.

```
In [4]: import plotly.express as px

# в начале указываем название датасета, с которым работаем
# animation_frame - название переменной, отвечающей за анимацию
# size_max - шкалирование населения (в matplotlib мы эту проблему решали через деление)
# hover_name - что отображается по наведению мышки.
px.scatter(gapminder, x="gdpPercap", y="lifeExp", animation_frame="year",
           size="pop", color="continent", hover_name="country",
           log_x=True, size_max=55, range_x=[100,100000], range_y=[25,90])
```



**Графики для категориальных переменных:
столбчатые диаграммы и график рассеяния для
категориальных переменных**

Тут будем работать по мотивам вот этого блокнота.

<https://nbviewer.jupyter.org/github/yaph/ipython-notebooks/blob/master/movie-body-counts.ipynb>
[\(https://nbviewer.jupyter.org/github/yaph/ipython-notebooks/blob/master/movie-body-counts.ipynb\)](https://nbviewer.jupyter.org/github/yaph/ipython-notebooks/blob/master/movie-body-counts.ipynb)

Будем работать с датасетом, который подсчитывает количество смертей в фильмах (методологию сбора данных можно посмотреть по ссылке выше). Несколько графиков мы рассмотрим на занятии, остальные можно изучить самостоятельно.

Очень часто для того, чтобы получить тот график, который хотим - нужно сделать правильную группировку. С этим тоже тут поработаем.

```
In [37]: movies = pd.read_csv('https://raw.githubusercontent.com/rogovich/2020_MIREC_PfDA/master/Seminars/S5_Matplotlib/bodycount_Directors_Genra.csv', index_col=0)
movies.head()
```

Out[37]:

	Film	Year	Body_Count	MPAA_Rating	Genre	Director	Length_Minutes	IMD
0	24 Hour Party People	2002	7.0	R	Biography	Michael Winterbottom	117	
1	24 Hour Party People	2002	7.0	R	Comedy	Michael Winterbottom	117	
2	24 Hour Party People	2002	7.0	R	Drama	Michael Winterbottom	117	
3	24 Hour Party People	2002	7.0	R	Music	Michael Winterbottom	117	
4	28 Days Later	2002	53.0	R	Horror	Danny Boyle	113	

Обратите внимание, что мы работаем с таким датасетом, где один и тот же фильм может встречаться несколько раз (это связано с тем, что он может относиться к нескольким жанрам или имеет несколько режиссеров - эта информация добавляет дополнительные ряды). Поэтому при некоторых агрегациях будем выкидывать дубликаты из колонки Film, чтобы не было задвоения.

```
In [41]: movies.shape # размер оригинального датафрейма
```

```
Out[41]: (1682, 10)
```

```
In [42]: movies.drop_duplicates('Film').shape # размер датафрейма для уникальных фильмов
```

```
Out[42]: (537, 10)
```

Давайте сгруппируем фильмы по году и суммируем значения для того, чтобы построить столбчатый график.

Агрегация по годам. Столбчатый график

Давайте построим два столбчатых графика, которые показывают количество фильмов за каждый год и общее количество смертей на экране.

```
In [43]: movies_year = movies.drop_duplicates('Film').groupby('Year').sum()
```

```
In [44]: movies_year.tail()
```

```
Out[44]:
```

	Body_Count	Length_Minutes	IMDB_Rating	Film_Count	Body_Count_Min
--	------------	----------------	-------------	------------	----------------

Year	Body_Count	Length_Minutes	IMDB_Rating	Film_Count	Body_Count_Min
2007	4095.0	5475	327.8	48	35.992220
2008	1785.0	2850	170.9	26	16.522174
2009	605.0	1235	75.3	11	5.708305
2010	519.0	463	29.0	4	4.528441
2013	156.0	119	6.5	1	1.310924

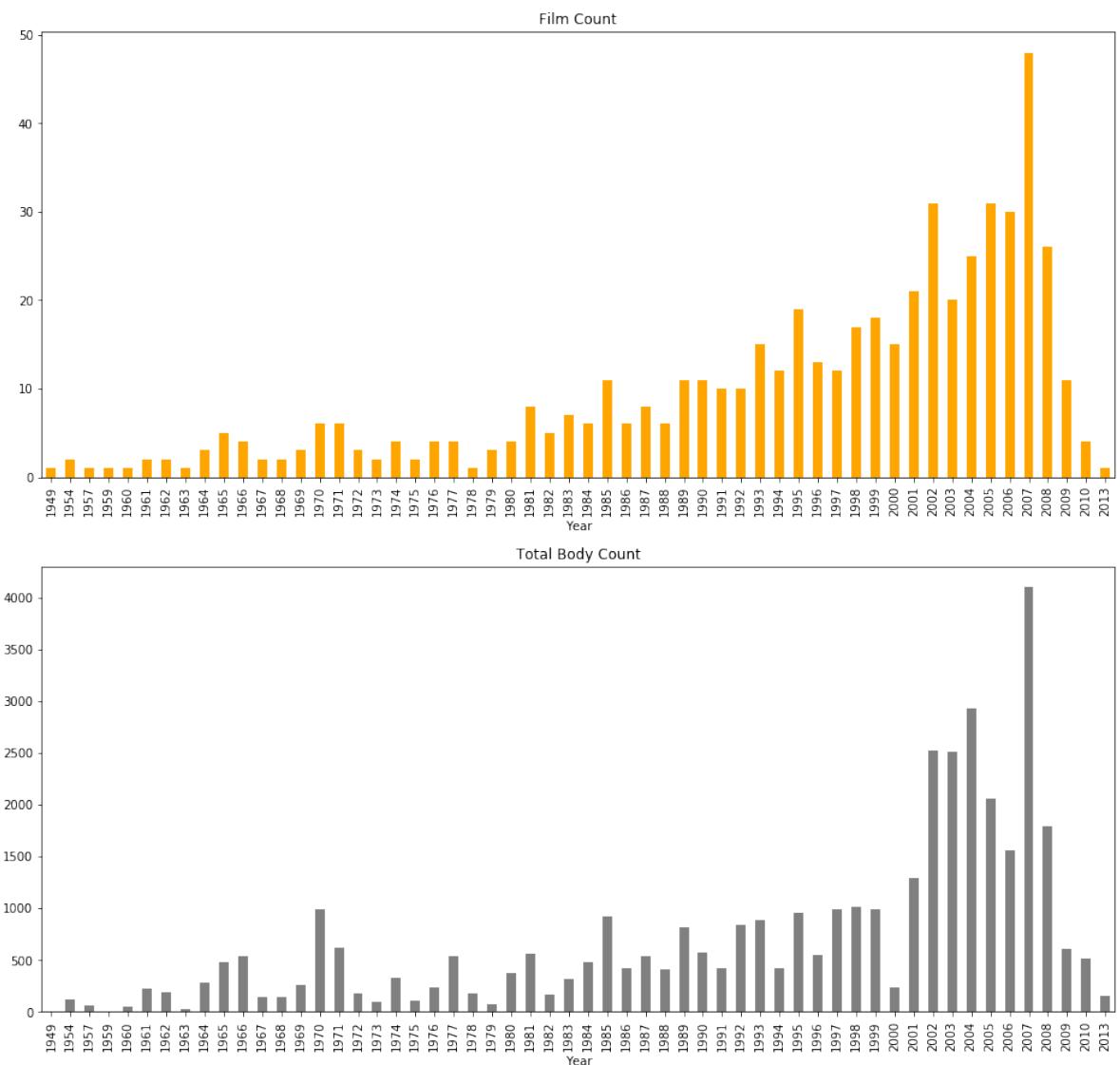
```
In [45]: fig, ax = plt.subplots(2, 1, figsize=(16, 15))

# обратите внимание, что здесь меняем синтаксис - делаем график, пр
именяя метод plot из пандас к колонке.
# И уже в параметрах этого метода прописываем тип графика и ось коо
рдинат, на которую хотим его положить.
# При таком подходе у нас каждый год отобразится на оси координат

movies_year['Film_Count'].plot(kind='bar', ax=ax[0], color = '#ffa5
00')
ax[0].set_title('Film Count')

movies_year['Body_Count'].plot(kind='bar', ax=ax[1], color = 'grey'
)
ax[1].set_title('Total Body Count')
```

Out[45]: Text(0.5,1,'Total Body Count')



Агрегация по фильмам. Сортировка по значениям. Горизонтальный столбчатый график

Сейчас мы сгруппируем датасет по фильмам, найдем 10 самых жестоких и визуализируем их.

```
In [46]: movies_film = movies.drop_duplicates('Film').set_index('Film') # агрегируем и делаем индексом название фильма, так как индекс станет шкалой
movies_film.head()
```

Out[46]:

Film	Year	Body_Count	MPAA_Rating	Genre	Director	Length_Minutes	IMDB_I
24							
Hour	2002	7.0		R	Biography	Michael Winterbottom	117
Party							
People							
28							
Days	2002	53.0		R	Horror	Danny Boyle	113
Later							
28							
Weeks	2007	212.0		R	Horror	Juan Carlos Fresnadillo	100
Later							
30							
Days	2007	67.0		R	Horror	David Slade	113
of							
Night							
300	2007	600.0		R	Action	Zack Snyder	117

```
In [47]: movies_film.sort_values(by=['Body_Count'])['Body_Count'].tail(10) # сортируем и берем 10 последних значений
```

Out[47]:

Film	
King Arthur	378.0
Windtalkers	389.0
Lord of the Rings: Two Towers	468.0
A Fistful of Dynamite	471.0
The Last Samurai	558.0
Troy	572.0
Tae Guk Gi: The Brotherhood of War	590.0
300	600.0
Kingdom of Heaven	610.0
Lord of the Rings: Return of the King	836.0
Name: Body_Count, dtype: float64	

```
In [48]: fig, ax = plt.subplots(nrows=1, ncols=2, figsize=(10, 8))

# график для количества убитых людей в фильме. Горизонтальный график
# делаем параметром barh (bar horizontal)
movies_film.sort_values(by=['Body_Count'])['Body_Count'].tail(10).plot(kind='barh', ax=ax[0])
ax[0].set_title('Total Body Count')

movies_film.sort_values(by=['Body_Count_Min'])['Body_Count_Min'].tail(10).plot(kind='barh', ax=ax[1]) # для количества убитых людей на
# минуту фильма

ax[1].set_title('Body Count per Minute')
ax[1].yaxis.set_ticks_position('right')

for i in range(2):
    ax[i].set_ylabel('', visible=False) # убираем подпись к шкале,
    # которая генерируется автоматически
```



Рейтинги: график рассеяния для категориальной переменной

Графики рассеяния для категориальных переменных тоже имеют место быть. В данном случае у нас будет категориальная переменная по x (буквенный рейтинг фильма). Такой график помогает нам увидеть разброс значений в рамках категорий, а также некоторые корреляции и зависимости. Не забывайте только добавлять прозрачность, потому что в случае категориальных переменных большое количество точек может накладываться друг на друга.

```
In [49]: movies['MPAA_Rating'].value_counts()
```

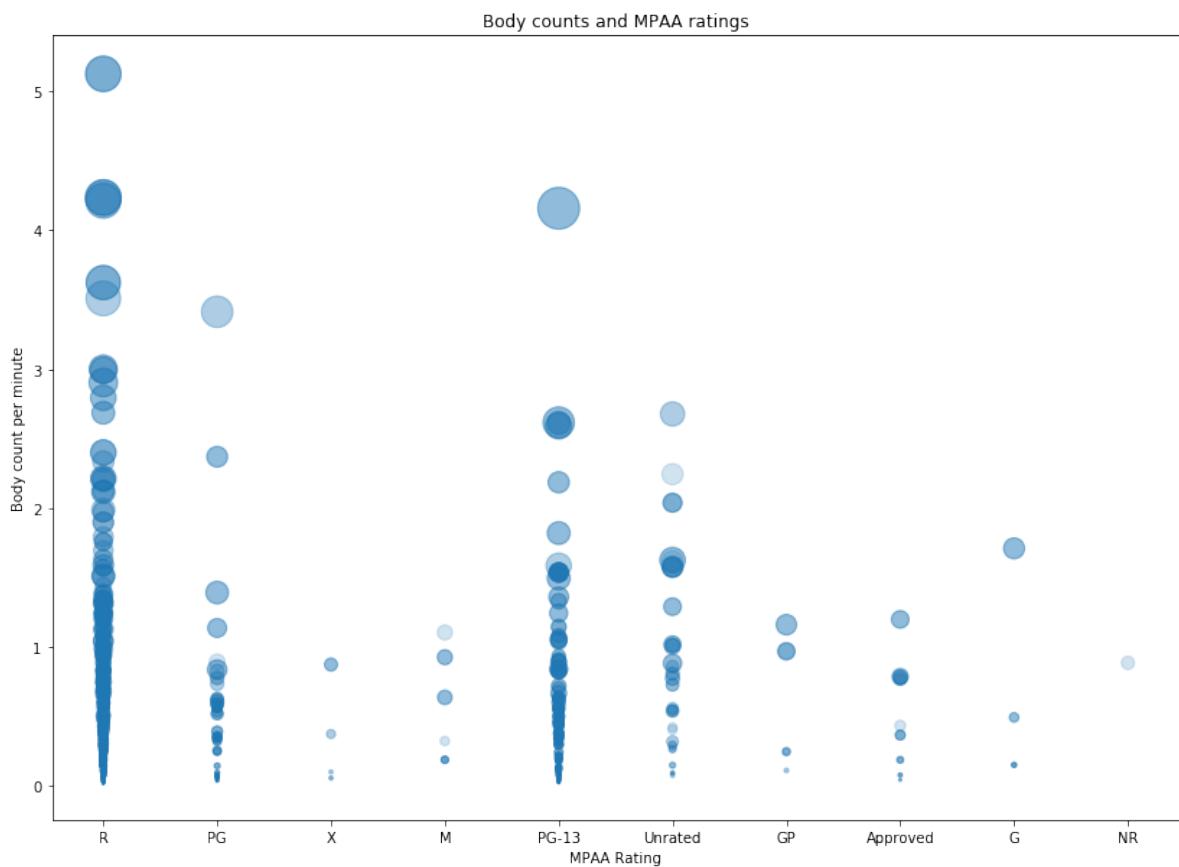
```
Out[49]: R          1039  
PG-13       383  
PG          114  
Unrated      70  
Approved     28  
M           13  
GP          13  
G            11  
X            10  
NR           1  
Name: MPAA_Rating, dtype: int64
```

Упражнение:

Постройте график рассеяния для MPAA_Rating по шкале x, Body_Count_Min по y и в качестве размера точки возьмите абсолютное значение Body Count.

```
In [50]: fig, ax = plt.subplots(figsize=(14, 10))  
  
ax.scatter(movies['MPAA_Rating'], movies['Body_Count_Min'], s=movies['Body_Count'], alpha=.2)  
ax.set_title('Body counts and MPAA ratings')  
ax.set_xlabel('MPAA Rating')  
ax.set_ylabel('Body count per minute')
```

```
Out[50]: Text(0,0.5,'Body count per minute')
```



Проинтерпретируйте график.

Так же для работы с такими данными могут понадобиться такие графики, как `swarmplot` (например, есть в библиотеке `seaborn`). В таком графике столбец переменной немного расширен, что позволяет точкам немного распределиться по горизонтали.

<https://seaborn.pydata.org/generated/seaborn.swarmplot.html>

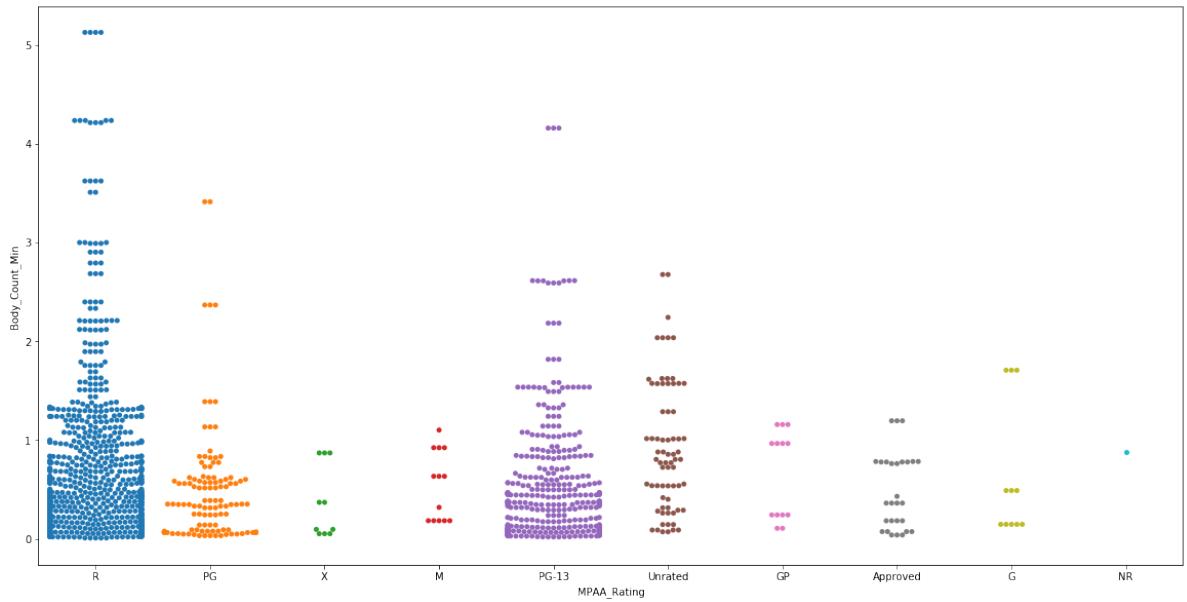
(<https://seaborn.pydata.org/generated/seaborn.swarmplot.html>)

```
In [51]: import seaborn as sns

fig, ax = plt.subplots(1,1, figsize = (20,10))

sns.swarmplot(movies[ 'MPAA_Rating' ], movies[ 'Body_Count_Min' ])
```

Out[51]: <matplotlib.axes._subplots.AxesSubplot at 0x1b928275128>



МИРЭК | 4 модуль

Автор: Татьяна Рогович

Основы программирования в Python

Семинар 7

Введение в ML. Разведывательный анализ данных.

```
In [2]: import pandas as pd  
import numpy as np  
import matplotlib.pyplot as plt  
import seaborn as sns  
import scipy.stats as st  
%matplotlib inline
```

Что такое разведывательный анализ данных или Exploratory Data Analysis (EDA)?

Разведывательный анализ данных помогает нам изучить данные, обработать пропущенные значения, убедиться, что данные корректно выглядят и в них нет ошибок, выбрать методы для обработки данных (например, стандартизация или OneHotEncoder), сделать первичный отбор признаков для модели.

По сути это работа с дескриптивными статистиками и визуализацией переменных

Главные методы:

1. Одномерный анализ

Распределения и дескриптивные статистики для каждой переменной.

2. Двумерный анализ

Исследуем переменные попарно (зависимые и независимые переменные друг с другом, независимые переменные при подозрении на мультиколлинеарность).

3. Снижение размерности

Поиск и выделение признаков, которые отвечают за наибольшую вариативность в данных.

4 цели EDA

- Обнаружить паттерны
- Заметить аномалии
- Сформулировать гипотезы
- Проверить предположения

Что исследуем во время EDA?

- Тренды
- Распределения
- Центральные тенденции и разброс
- Выбросы
- Корреляции
- Проверяем гипотезы
- Визуально исследуем данные

Сегодня мы работаем с достаточно известным набором данных House Prices Competition.

Подробное описание данных и переменных по ссылке: <https://www.kaggle.com/c/house-prices-advanced-regression-techniques/data> (<https://www.kaggle.com/c/house-prices-advanced-regression-techniques/data>)

```
In [3]: data = pd.read_csv('https://raw.githubusercontent.com/rogovich/2019-2020_PolSci_Data_Analysis_in_Python/master/12week_ML_Intro/house_data.csv')
```

```
In [4]: data.shape
```

```
Out[4]: (1460, 81)
```

Мы работает с частью данных, которые в соревновании применяются для тренировки модели. Всего 1460 наблюдений и 81 признак, из которых 36 количественных, 43 категориальных + id + целевая переменная SalePrice

Количественные признаки: 1stFlrSF, 2ndFlrSF, 3SsnPorch, BedroomAbvGr, BsmtFinSF1, BsmtFinSF2, BsmtFullBath, BsmtHalfBath, BsmtUnfSF, EnclosedPorch, Fireplaces, FullBath, GarageArea, GarageCars, GarageYrBlt, GrLivArea, HalfBath, KitchenAbvGr, LotArea, LotFrontage, LowQualFinSF, MSSubClass, MasVnrArea, MiscVal, MoSold, OpenPorchSF, OverallCond, OverallQual, PoolArea, ScreenPorch, TotRmsAbvGrd, TotalBsmtSF, WoodDeckSF, YearBuilt, YearRemodAdd, YrSold

Категориальные признаки: Alley, BldgType, BsmtCond, BsmtExposure, BsmtFinType1, BsmtFinType2, BsmtQual, CentralAir, Condition1, Condition2, Electrical, ExterCond, ExterQual, Exterior1st, Exterior2nd, Fence, FireplaceQu, Foundation, Functional, GarageCond, GarageFinish, GarageQual, GarageType, Heating, HeatingQC, HouseStyle, KitchenQual, LandContour, LandSlope, LotConfig, LotShape, MSZoning, MasVnrType, MiscFeature, Neighborhood, PavedDrive, PoolQC, RoofMatl, RoofStyle, SaleCondition, SaleType, Street, Utilitif

```
In [4]: data.head()
```

```
Out[4]:
```

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContou
0	1	60	RL	65.0	8450	Pave	NaN	Reg	Li
1	2	20	RL	80.0	9600	Pave	NaN	Reg	Li
2	3	60	RL	68.0	11250	Pave	NaN	IR1	Li
3	4	70	RL	60.0	9550	Pave	NaN	IR1	Li
4	5	60	RL	84.0	14260	Pave	NaN	IR1	Li

5 rows × 81 columns

In [5]: `data.tail()`

Out[5]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandC
1455	1456	60	RL	62.0	7917	Pave	NaN	Reg	
1456	1457	20	RL	85.0	13175	Pave	NaN	Reg	
1457	1458	70	RL	66.0	9042	Pave	NaN	Reg	
1458	1459	20	RL	68.0	9717	Pave	NaN	Reg	
1459	1460	20	RL	75.0	9937	Pave	NaN	Reg	

5 rows × 81 columns

In [7]: `data.info() # изучаем данные – пропущенные значения и типы`

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1460 entries, 0 to 1459
Data columns (total 81 columns):
Id                 1460 non-null int64
MSSubClass        1460 non-null int64
MSZoning          1460 non-null object
LotFrontage       1201 non-null float64
LotArea            1460 non-null int64
Street             1460 non-null object
Alley              91 non-null object
LotShape            1460 non-null object
LandContour        1460 non-null object
Utilities           1460 non-null object
LotConfig           1460 non-null object
LandSlope           1460 non-null object
Neighborhood        1460 non-null object
Condition1          1460 non-null object
Condition2          1460 non-null object
BldgType            1460 non-null object
HouseStyle           1460 non-null object
OverallQual         1460 non-null int64
OverallCond         1460 non-null int64
YearBuilt            1460 non-null int64
YearRemodAdd        1460 non-null int64
RoofStyle            1460 non-null object
RoofMatl             1460 non-null object
Exterior1st          1460 non-null object
Exterior2nd          1460 non-null object
MasVnrType           1452 non-null object
MasVnrArea           1452 non-null float64
ExterQual            1460 non-null object
ExterCond            1460 non-null object
Foundation           1460 non-null object
BsmtQual             1423 non-null object
BsmtCond             1423 non-null object
```

```

BsmtExposure      1422 non-null object
BsmtFinType1      1423 non-null object
BsmtFinSF1         1460 non-null int64
BsmtFinType2      1422 non-null object
BsmtFinSF2         1460 non-null int64
BsmtUnfSF          1460 non-null int64
TotalBsmtSF        1460 non-null int64
Heating            1460 non-null object
HeatingQC           1460 non-null object
CentralAir          1460 non-null object
Electrical          1459 non-null object
1stFlrSF            1460 non-null int64
2ndFlrSF            1460 non-null int64
LowQualFinSF        1460 non-null int64
GrLivArea           1460 non-null int64
BsmtFullBath        1460 non-null int64
BsmtHalfBath        1460 non-null int64
FullBath             1460 non-null int64
HalfBath             1460 non-null int64
BedroomAbvGr        1460 non-null int64
KitchenAbvGr        1460 non-null int64
KitchenQual          1460 non-null object
TotRmsAbvGrd        1460 non-null int64
Functional           1460 non-null object
Fireplaces           1460 non-null int64
FireplaceQu          770 non-null object
GarageType            1379 non-null object
GarageYrBlt           1379 non-null float64
GarageFinish          1379 non-null object
GarageCars             1460 non-null int64
GarageArea             1460 non-null int64
GarageQual            1379 non-null object
GarageCond             1379 non-null object
PavedDrive            1460 non-null object
WoodDeckSF            1460 non-null int64
OpenPorchSF           1460 non-null int64
EnclosedPorch          1460 non-null int64
3SsnPorch             1460 non-null int64
ScreenPorch            1460 non-null int64
PoolArea              1460 non-null int64
PoolQC                 7 non-null object
Fence                  281 non-null object
MiscFeature            54 non-null object
MiscVal                 1460 non-null int64
MoSold                  1460 non-null int64
YrSold                  1460 non-null int64
SaleType                  1460 non-null object
SaleCondition            1460 non-null object
SalePrice                  1460 non-null int64
dtypes: float64(3), int64(35), object(43)
memory usage: 924.0+ KB

```

```
In [5]: # выделяем количественные и категориальные переменные
numeric_features = data.select_dtypes(include=[np.number])
categorical_features = data.select_dtypes(include=[np.object])
```

```
In [6]: numeric_features.iloc[:, :18].describe() # смотрим описательные статистики первых 18 количественных переменных
```

Out[6]:

	Id	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	
count	1460.000000	1460.000000	1201.000000	1460.000000	1460.000000	1460.000000	1
mean	730.500000	56.897260	70.049958	10516.828082	6.099315	5.575342	1
std	421.610009	42.300571	24.284752	9981.264932	1.382997	1.112799	
min	1.000000	20.000000	21.000000	1300.000000	1.000000	1.000000	1
25%	365.750000	20.000000	59.000000	7553.500000	5.000000	5.000000	1
50%	730.500000	50.000000	69.000000	9478.500000	6.000000	5.000000	1
75%	1095.250000	70.000000	80.000000	11601.500000	7.000000	6.000000	2
max	1460.000000	190.000000	313.000000	215245.000000	10.000000	9.000000	2

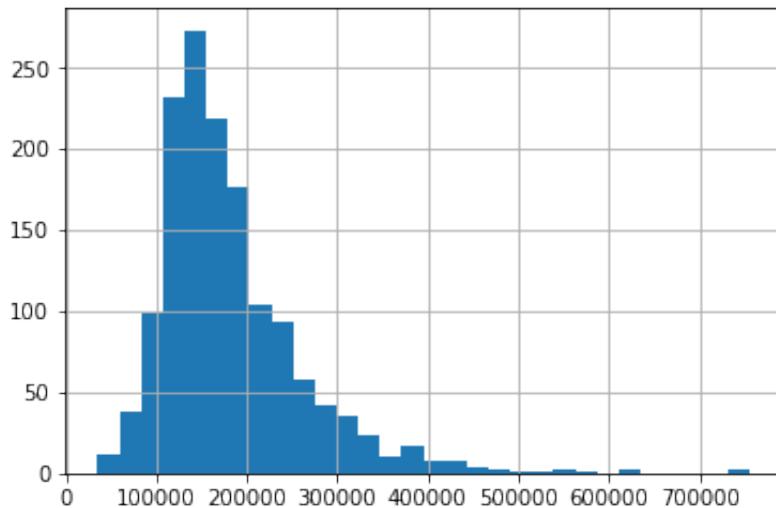
Количественные переменные: распределения

Отдельно всегда стоит изучить целевую переменную, которую собираемся предсказывать. В этом случае это цена дома.

```
In [8]: y = data['SalePrice'] # выделяем целевую переменную в вектор y
```

```
In [10]: y.hist(bins = 30) # строим распределение
```

```
Out[10]: <matplotlib.axes._subplots.AxesSubplot at 0x26988ee9e10>
```

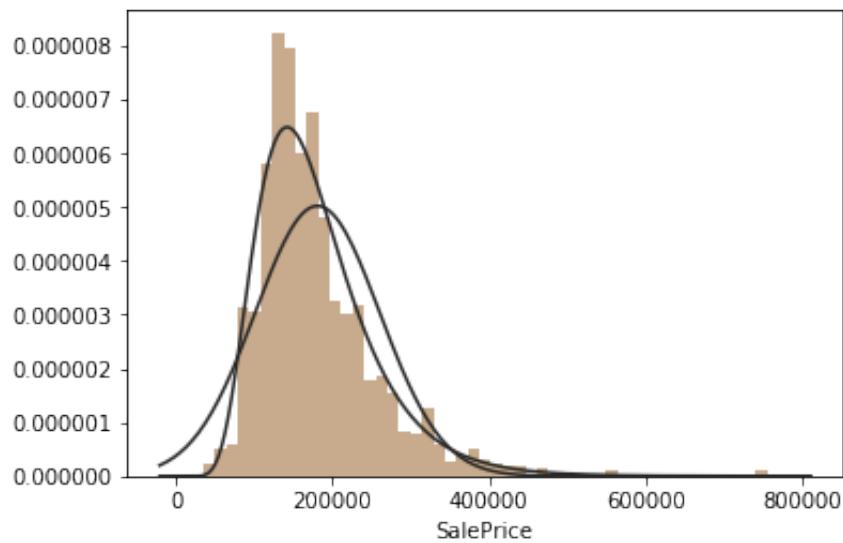


EDA можно проводить только методами pandas и matplotlib, но библиотека seaborn умеет немного больше. Например, мы можем посмотреть, как выглядит наше распределение относительно стандартных форм (например, нормального и лог-нормального).

```
In [11]: sns.distplot(y, kde=False, fit=st.norm)
sns.distplot(y, kde=False, fit=st.lognorm)
```

```
C:\Users\rogov\Anaconda3\lib\site-packages\matplotlib\axes\_axes.py:6462: UserWarning: The 'normed' kwarg is deprecated, and has been replaced by the 'density' kwarg.  
    warnings.warn("The 'normed' kwarg is deprecated, and has been "  
C:\Users\rogov\Anaconda3\lib\site-packages\matplotlib\axes\_axes.py:6462: UserWarning: The 'normed' kwarg is deprecated, and has been replaced by the 'density' kwarg.  
    warnings.warn("The 'normed' kwarg is deprecated, and has been "  
    warnings.warn("The 'normed' kwarg is deprecated, and has been "
```

Out[11]: <matplotlib.axes._subplots.AxesSubplot at 0x2698de2a1d0>

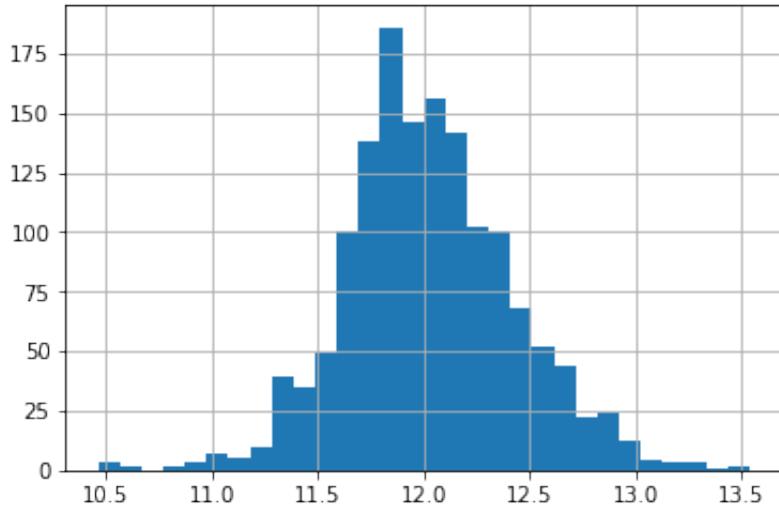


Мы видим, что наше распределение больше соответствует логарифмическому. Очень часто мы хотим нормировать такое распределение (могут быстрее сходиться алгоритмы, линейные модели в принципе предполагают, что переменные должны быть распределены нормально, корреляция легче интерпретируется и т.д.)

Давайте посмотрим, как будет выглядеть график цены, после применения лог-функции к нашему вектору.

```
In [13]: np.log(y).hist(bins = 30) # строим распределение
```

```
Out[13]: <matplotlib.axes._subplots.AxesSubplot at 0x2698de01cc0>
```



Матрицы корреляции

Окей, вроде разобрались с одномерными распределениями. Кстати, изучая распределения можно как раз увидеть аномальные значения, решить, что с ними делать. Следующий шаг - это изучение зависимостей в данных. Для этого обычно используют тепловую карту корреляций, попарные графики и графики рассеяния.

Для начала найдем коэффициенты попарных корреляций между количественными переменными.

[Подробнее про корреляцию \(<http://statistica.ru/theory/koeffitsient-korrelyatsii/>\)](http://statistica.ru/theory/koeffitsient-korrelyatsii/)

```
In [14]: correlation = numeric_features.corr()
correlation.head() # получаем датафрейм, где на пересечении рядов и
# колонок коэффициенты корреляции Пирсона для этих пар
```

```
Out[14]:
```

	Id	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	Y
Id	1.000000	0.011156	-0.010601	-0.033226	-0.028365	0.012609	-0
MSSubClass	0.011156	1.000000	-0.386347	-0.139781	0.032628	-0.059316	0
LotFrontage	-0.010601	-0.386347	1.000000	0.426095	0.251646	-0.059213	0
LotArea	-0.033226	-0.139781	0.426095	1.000000	0.105806	-0.005636	0
OverallQual	-0.028365	0.032628	0.251646	0.105806	1.000000	-0.091932	0

5 rows × 38 columns

```
In [15]: print(correlation['SalePrice'].sort_values(ascending = False), '\n')  
# ВЫВОДИМ КОЭФФИЦЕНТЫ ПОПАРНОЙ КОРРЕЛЯЦИИ С SalePrice
```

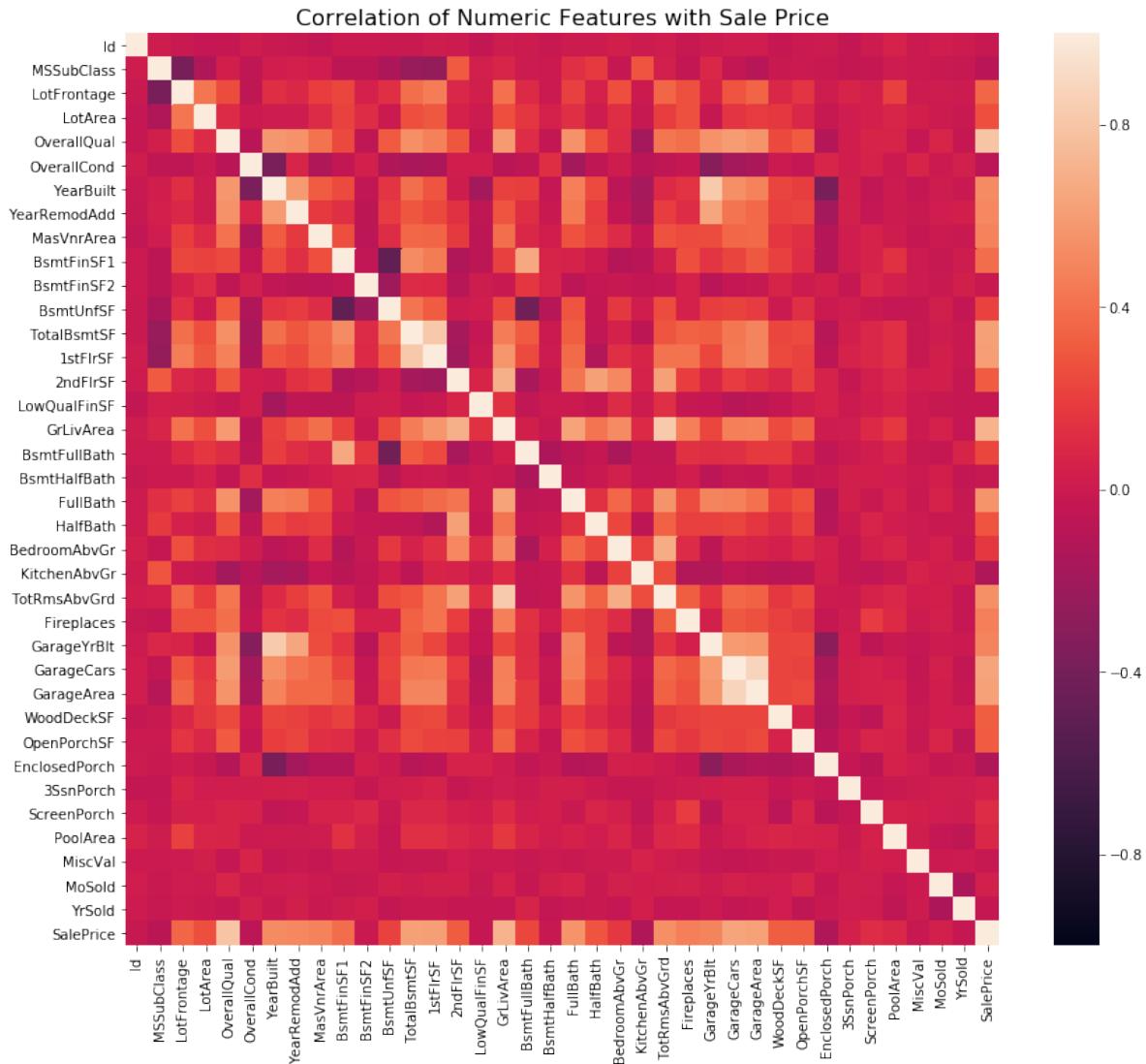
SalePrice	1.000000
OverallQual	0.790982
GrLivArea	0.708624
GarageCars	0.640409
GarageArea	0.623431
TotalBsmtSF	0.613581
1stFlrSF	0.605852
FullBath	0.560664
TotRmsAbvGrd	0.533723
YearBuilt	0.522897
YearRemodAdd	0.507101
GarageYrBlt	0.486362
MasVnrArea	0.477493
Fireplaces	0.466929
BsmtFinSF1	0.386420
LotFrontage	0.351799
WoodDeckSF	0.324413
2ndFlrSF	0.319334
OpenPorchSF	0.315856
HalfBath	0.284108
LotArea	0.263843
BsmtFullBath	0.227122
BsmtUnfSF	0.214479
BedroomAbvGr	0.168213
ScreenPorch	0.111447
PoolArea	0.092404
MoSold	0.046432
3SsnPorch	0.044584
BsmtFinSF2	-0.011378
BsmtHalfBath	-0.016844
MiscVal	-0.021190
Id	-0.021917
LowQualFinSF	-0.025606
YrSold	-0.028923
OverallCond	-0.077856
MSSubClass	-0.084284
EnclosedPorch	-0.128578
KitchenAbvGr	-0.135907
Name: SalePrice, dtype: float64	

```
In [16]: fig, ax = plt.subplots(figsize = (14,12))

plt.title('Correlation of Numeric Features with Sale Price', size=16)

sns.heatmap(correlation, vmax=1.0, vmin = -1)
```

Out[16]: <matplotlib.axes._subplots.AxesSubplot at 0x2698e17b748>



На таком графике можно увидеть признаки, которые сильно коррелируют с целевой переменной, а значит обладают большим предсказательным потенциалом. А также найти переменные, которые сильно коррелируют между собой, и избавиться от части из них в финальной модели. Здесь нас интересуют значения ближе к -1 и 1.

Мы сразу видим, что есть два кластера с потенциальной мультиколлинеарностью (наличие линейной зависимости между объясняющими переменными (факторами) регрессионной модели) - 'TotalBsmtSF' и '1stFlrSF', и переменные 'GarageX'.

Касаемо 'SalePrice', мы видим, что 'GrLivArea' и 'OverallQual' сильнее всего коррелируют с ценой. Давайте попробуем рассмотреть наши данные подробнее.

Для начала отфильтруем по индексу 10 топ-корреляций с ценой (мы берем 11, потому что тут еще включается корреляция переменной самой с собой.)

```
In [17]: correlation['SalePrice'].sort_values(ascending = False).head(11).index # так мы уже умеем это делать
```

```
Out[17]: Index(['SalePrice', 'OverallQual', 'GrLivArea', 'GarageCars', 'GarageArea',
       'TotalBsmtSF', '1stFlrSF', 'FullBath', 'TotRmsAbvGrd', 'YearBuilt',
       'YearRemodAdd'],
      dtype='object')
```

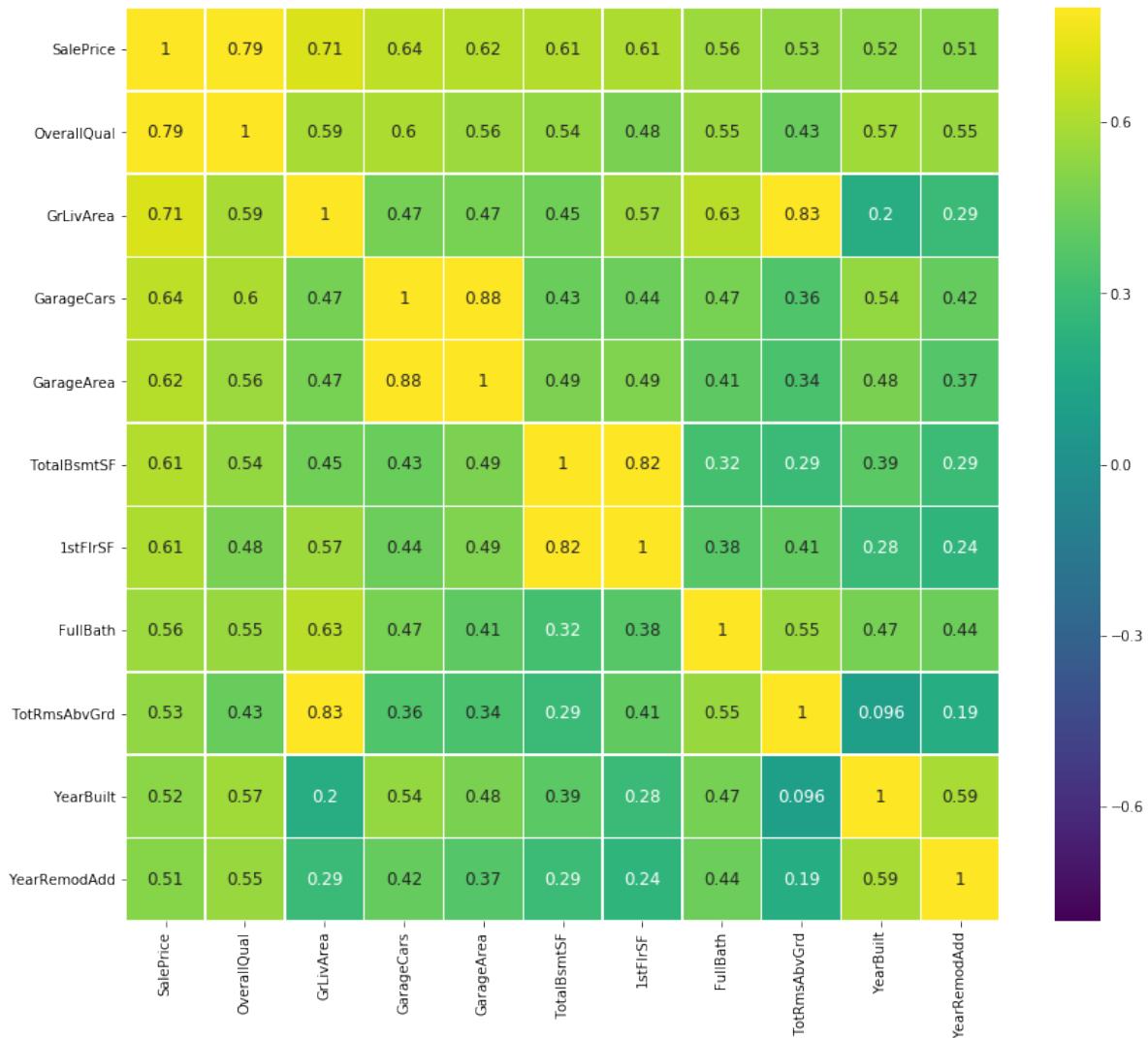
```
In [18]: correlation.nlargest(11,'SalePrice')['SalePrice'].index # альтернативный вариант
```

```
Out[18]: Index(['SalePrice', 'OverallQual', 'GrLivArea', 'GarageCars', 'GarageArea',
       'TotalBsmtSF', '1stFlrSF', 'FullBath', 'TotRmsAbvGrd', 'YearBuilt',
       'YearRemodAdd'],
      dtype='object')
```

```
In [19]: cols = correlation.nlargest(11,'SalePrice')['SalePrice'].index # храним интересующие нас колонки для фильтрации
```

```
In [20]: fig, ax = plt.subplots(figsize = (14,12))
sns.heatmap(correlation.loc[cols, cols], vmax=0.8, vmin = -0.8, linewidths=0.5, annot=True,cmap='viridis',
            linecolor="white",xticklabels = cols.values ,annot_kws = {'size':12},yticklabels = cols.values)
```

Out[20]: <matplotlib.axes._subplots.AxesSubplot at 0x2698e68eda0>



Из такой приближенной тепловой карты, мы видим, что переменные, которые мы подозревали, действительно коррелируют между собой. И что цена действительно сильнее всего связана с 'GrLivArea' и 'OverallQual'

Промежуточные выводы

- 'OverallQual' и 'GrLivArea' сильно коррелируют с 'SalePrice'.
- 'GarageCars' и 'GarageArea' сильно коррелируют между собой. Что логично, потому что количество машин, помещающихся в гараж, по сути функция площади гаража. Для будущей модели мы можем использовать одну переменную из двух (например, ту, которая сильнее коррелирует с ценой)
- 'TotalBsmtSF' и '1stFloor' тоже выглядят близнецами.
- Как и 'TotRmsAbvGrd' и 'GrLivArea'.

Попарные визуализации

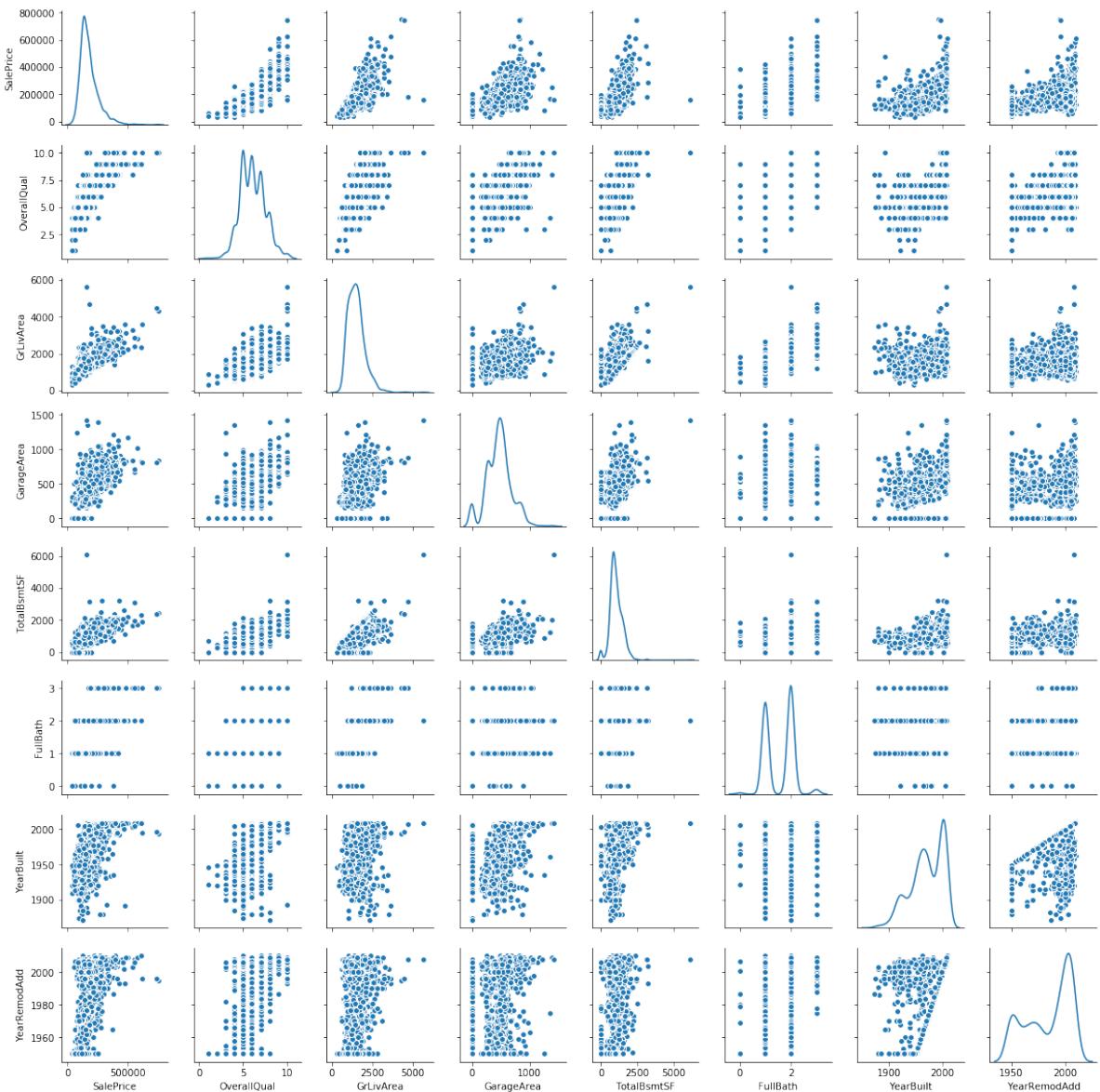
С помощью матрицы корреляций мы нашли переменные, которые сильнее всего коррелируют с ценой дома. Теперь с помощью попарных графиков, мы можем лучше изучить эти взаимосвязи. Для начала удалим по переменной из пар, где мы обнаружили мультиколлинеарность.

```
In [21]: cols
```

```
Out[21]: Index(['SalePrice', 'OverallQual', 'GrLivArea', 'GarageCars', 'GarageArea',
       'TotalBsmtSF', '1stFlrSF', 'FullBath', 'TotRmsAbvGrd', 'YearBuilt',
       'YearRemodAdd'],
      dtype='object')
```

```
In [22]: new_cols = ['SalePrice', 'OverallQual', 'GrLivArea', 'GarageArea',
       'TotalBsmtSF', 'FullBath', 'YearBuilt', 'YearRemodAdd']
```

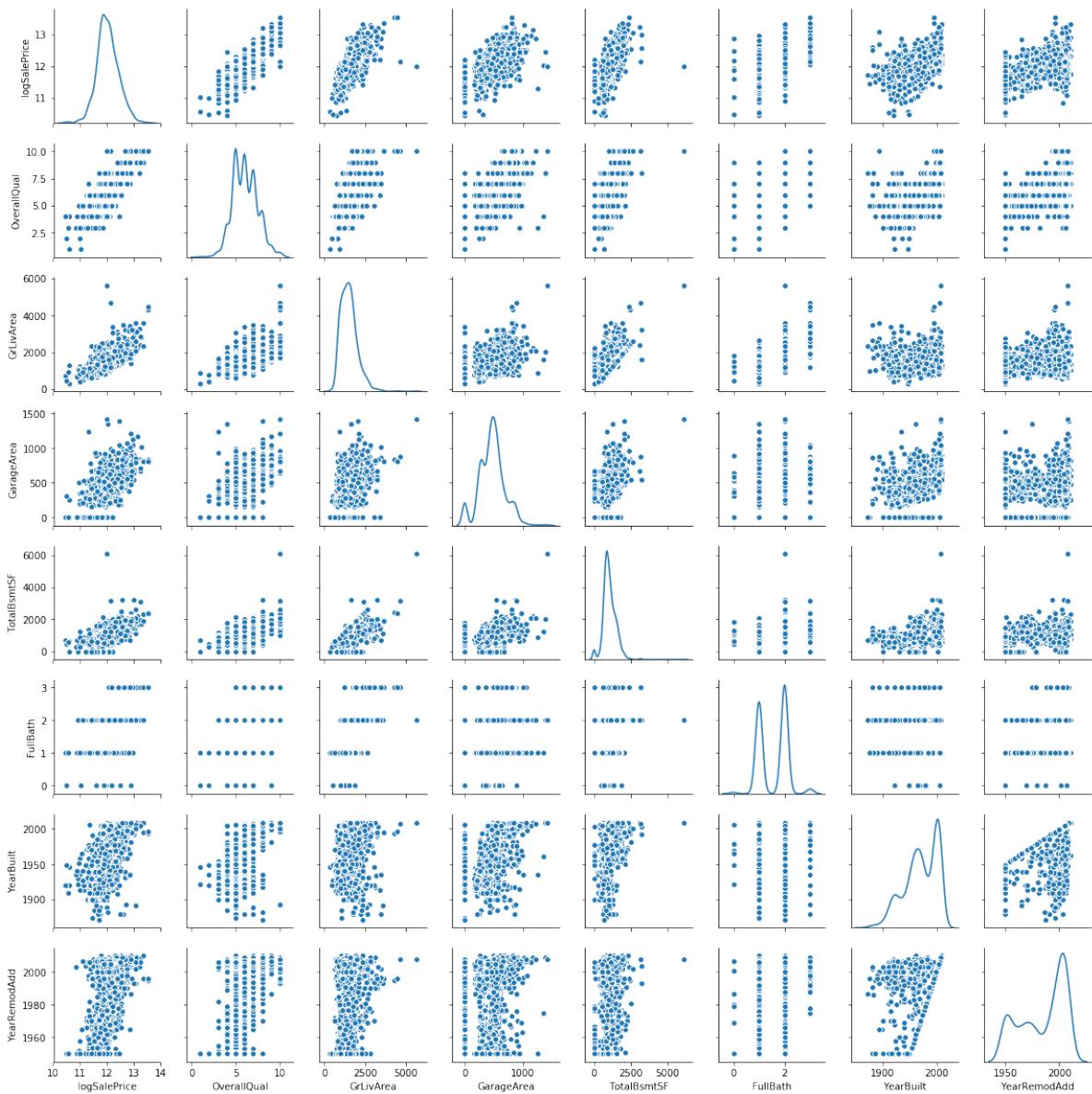
```
In [23]: sns.pairplot(data[new_cols], size = 2, kind = 'scatter', diag_kind='kde')
plt.show()
```



Кстати, выше я уже упоминала, что один из кейсов для лог-нормирования распределение - оценка взаимосвязи в графиках рассеяния. Давайте посмотрим, как это влияет.

```
In [43]: data['logSalePrice'] = np.log(data['SalePrice']) # создаем новую переменную с логарифмом цены
new_cols_log = ['logSalePrice', 'OverallQual', 'GrLivArea', 'GarageArea', 'TotalBsmtSF', 'FullBath', 'YearBuilt', 'YearRemodAdd']
```

```
In [44]: sns.pairplot(data[new_cols_log], size = 2 , kind = 'scatter', diag_kind = 'kde')
plt.show()
```



Видим, что многие графики приняли более линейную форму (например цена и OverallQual).

Промежуточные выводы

- Например, мы видим как связаны переменные 'TotalBsmtSF' и 'GrLiveArea'. Мы можем увидеть диагональ, которая как бы делить область координат на секторы, и все наблюдения сконцентрированы ниже нее. Это логично, потому что мы ожидаем, что площадь подвального помещения будет меньше или равна площади дома.
- Если посмотрим на взаимосвязь 'SalePrice' и 'YearBuilt', то увидим, что отношения между ними экспоненциальные - в последние годы цены растут быстрее по отношению к прошедшим годам. Кстати, это одна из причин, почему коэффициент корреляции Пирсона был невысокий - эта связь не линейная.

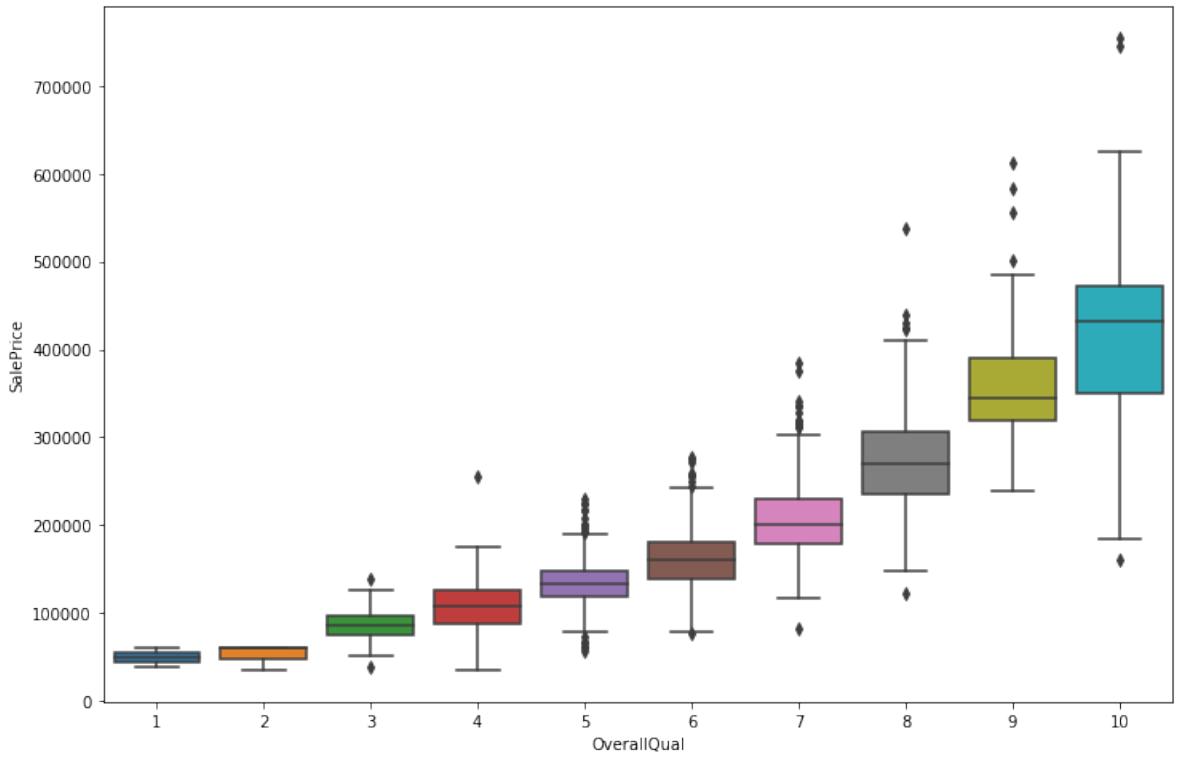
Попарные ящики с усами для категориальных переменных

В том случае, если одна из наших переменных категориальная, а вторая непрерывная, лучше использовать ящики с усами или скрипичные графики, чтобы сравнить распределения непрерывной переменной внутри категорий.

[Подробнее про ящик с усами \(\[https://datavizcatalogue.com/RU/metody/diagramma_razmaha.html\]\(https://datavizcatalogue.com/RU/metody/diagramma_razmaha.html\)\)](https://datavizcatalogue.com/RU/metody/diagramma_razmaha.html)

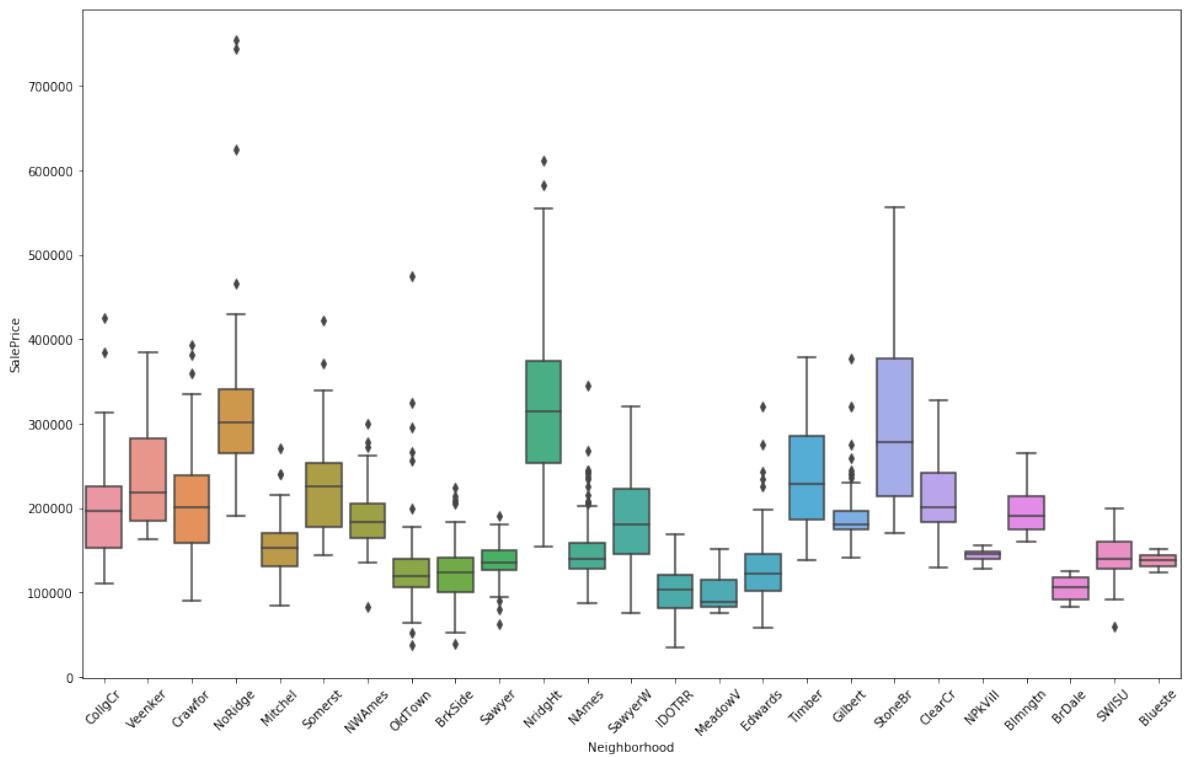
```
In [26]: fig, ax = plt.subplots(figsize=(12, 8))
sns.boxplot(data[ 'OverallQual' ], data[ "SalePrice" ])
```

```
Out[26]: <matplotlib.axes._subplots.AxesSubplot at 0x269916ee358>
```



```
In [27]: f, ax = plt.subplots(figsize=(16, 10))
fig = sns.boxplot(x='Neighborhood', y="SalePrice", data=data)
plt.xticks(rotation=45)
```

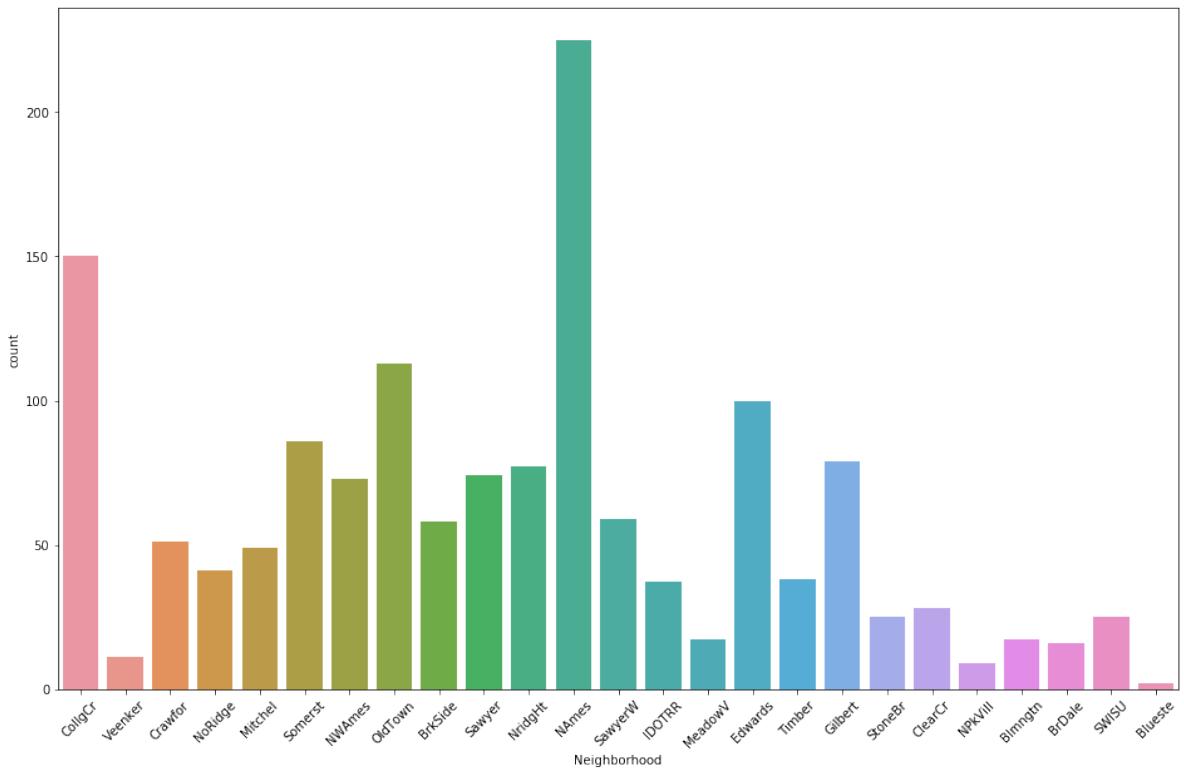
```
Out[27]: (array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14,
   15, 16,
   17, 18, 19, 20, 21, 22, 23, 24]),  
<a list of 25 Text xticklabel objects>)
```



Кстати, чтобы оценить распределение категориальной переменной, мы обычно используем бар-чарт, который показывает, сколько наблюдений попадает в каждую категорию.

```
In [30]: f, ax = plt.subplots(figsize=(16, 10))
fig = sns.countplot(x='Neighborhood', data=data)
plt.xticks(rotation=45)
```

```
Out[30]: (array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14,
   15, 16,
   17, 18, 19, 20, 21, 22, 23, 24]),  
<a list of 25 Text xticklabel objects>)
```



Мы видим, что у нас есть несколько районов с похожими ценами. Это, например, можно использовать для снижения размерности - объединять переменную Neighborhood в более крупные категории.

Таким образом мы теперь можем изучать зависимость цены от категориальных переменных. Здесь в основном идет речь про создание новых признаков за счет выделения в отдельные переменные категорий, которые значительно отличаются от других (например, деревянные крыши). Или уменьшение размерности внутри категорий, а также работа с выбросами.

Дополнительные материалы

Также существуют специальные библиотеки для EDA. Например, `pandas_profiling`. Ее нет в стандартном наборе Анаконды, но ее легко установить через pip. [Статья на хабре](https://habr.com/ru/company/ruvds/blog/451478/) (<https://habr.com/ru/company/ruvds/blog/451478/>)

```
In [ ]:
```

МИРЭК | 4 модуль

Автор: Татьяна Рогович

Основы программирования в Python

Семинар 7

Введение в ML. Задача регрессии. Решающее дерево и случайный лес

Этот блокнот составлен на основе большого блокнота, который доступен по этой [ссылке](#) (<https://www.kaggle.com/dansbecker/your-first-machine-learning-model>). В нем присутствуют дополнительные комментарии и другие примеры работы алгоритма. Если хотите как следует разобраться с деревьями на базовом уровне - обязательно читайте.

Загружаем наши данные

Нашим датасетом будет информация о домах, которая включает в себя множество параметров, например, количество и размер этажей дома, год постройки, общая площадь, цена и многое другое (остальные столбцы, можете изучить самостоятельно).

```
In [2]: import pandas as pd

#Загружаем данные
home_data = pd.read_csv('https://raw.githubusercontent.com/rogovich/2019-2020_PolSci_Data_Analysis_in_Python/master/12week_ML_Intro/home_data.csv')
```

Выбор целевой переменной

В нашем случае, все более-менее очевидно. Это цена на дом. Обычно, цель предсказания помечают у.

```
In [3]: y_train = home_data.SalePrice
```

Выбор признаков (Features)

Столбцы, которые есть в нашей модели и которые в последствии будут использованы для предсказания, называются признаками (features). В нашем случае, эти колонки будут определять стоимость дома. Иногда используются все колонки, кроме той на которую делается предсказание. А вдругих случаях, лучше выбрать только часть из них. Как определиться? Для этого проводят разведывательный анализ или сравнивают качество работы моделей на разных выборках признаков.

Мы будем использовать не все столбцы, чтобы алгоритм все считал быстрее. Добавим в модель только те столбцы, которые сильнее всего коррелировали с ценой, когда мы исследовали данные.

```
In [4]: feature_columns = ['OverallQual', 'GrLivArea', 'GarageArea', 'TotalBsmtSF', 'FullBath', 'YearBuilt', 'YearRemodAdd']
```

Обычно, такие данные обозначаются X:

```
In [5]: X_train = home_data[feature_columns]
```

Строим модель

Мы будем использовать модуль scikit-learn для создания модели. Scikit-learn одна из самых популярных библиотек для моделирования данных, хранящихся в датафреймах.

Для построения модели нужно выполнить следующие шаги:

Define: Какого типа будет модель? Дерево решений? Какой-то другой тип?

Fit: запускаем модель на тренировочных данных (обучаем ее), чтобы алгоритм нашел в них некие закономерности и зависимости.

Predict: Предсказать результат.

Evaluate: Определить насколько точным оказалось предсказывание, оценить качество модели

В нашем примере, мы будем использовать дерево решений из scikit-learn и тренировать модель с признаками, выделенными ранее. Импортируем функцию DecisionTreeRegressor

```
In [6]: from sklearn.tree import DecisionTreeRegressor
```

Инициализируем модель и обучим ее на наших данных.

```
In [7]: # Инициализируем модель
home_model = DecisionTreeRegressor()

# Обучаем модель
home_model.fit(X_train, y_train)

Out[7]: DecisionTreeRegressor()
```

Нам выше вывелась информация о нашем регрессоре и с какими параметрами он обучался.

Мы обучили модель. Но как хорошо она обучилась? Давайте предскажем значения и сравним их с теми ценами, которые на самом деле соответствуют домам.

```
In [8]: print("Наше предсказание:", home_model.predict(X_train.tail(10)))
print("Реальная цена домов:", y_train.tail(10).tolist())

Наше предсказание: [136000. 287090. 145000. 84500. 185000. 175000
. 210000. 266500. 142125.
147500.]
Реальная цена домов: [136000, 287090, 145000, 84500, 185000, 175000,
0, 210000, 266500, 142125, 147500]
```

Результаты похожи, но кое-где есть отличия. И мы отображаем всего 10 элементов из всей выборки. Посмотрим насколько правильно нам предсказала модель для всей выборки. Мы можем, конечно, пройтись в цикле по всем элементам и посчитать ошибку у каждого. Но у нас могут быть как хорошие, так и плохие предсказания. И в идеальном случае нам нужна одна метрика, которая скажет как хорошо мы обучили данные.

Воспользуемся метрикой МАЕ (Mean Absolute Error) - Средняя Абсолютная Ошибка. Ее можно представить в таком виде: ошибка = реальная цена – предсказанная цена. Например, если цена дома 150 000, мы предсказали цены в 100 000, то ошибка будет 50 000. Для ошибок в отрицательную сторону берем модуль. Это одна из простейших метрик.

Импортируем нужную функцию опять же из sklearn

```
In [9]: from sklearn.metrics import mean_absolute_error
```

Делаем предсказание и передаем в функцию предсказанное и реальное значение

```
In [10]: pred = home_model.predict(X_train)
mae = mean_absolute_error(pred, y_train)
print(mae)
```

111.4013698630137

Получили ошибку достаточно маленькую ошибку цены в долларах, что было бы очень хорошим результатом, если не одно но.

Валидация данных

Мы только что проверяли обученную модель на тех же данных, на которых мы ее и обучали. И это неверно. Т.к. в нашем случае нам нужно предсказывать цены новых домов, которых не было в обучающей выборке. Поэтому не удивительно, что MAE = 111.

Опишем эту проблему таким примером. У нас есть большой рынок жилья. Цена жилья не связана с цветом входной двери. Но в нашей обучающей выборке у всех домов с высокой ценой была зеленая дверь. Модель увидит эту зависимость и будет предсказывать для новых домов с зеленой дверью высокую цену, что не будет корректным. Но при валидации модели на обучающих данных мы получили маленькую ошибку и посчитали, что модель натренирована хорошо. Если бы мы это сделали на тестовых данных, то увидели бы что ошибка велика и нужно переделать модель.

Обучающие и тестовые выборки

Давайте разделим наши данные на 2 части: обучающую выборку и тестовую. Воспользуемся опять модулем sklearn и функцией train_test_split.

```
In [11]: from sklearn.model_selection import train_test_split
```

Разобъем данные как для X, так и для y. Каждый раз разбитие происходит случайно, мы можем только указать в какой пропорции мы хотим разбить наши данные, например 30% всей выборки на обучение, остальное на тест.

```
In [12]: # заполнили массивы с нашими данными
X = home_data[feature_columns]
y = home_data.SalePrice

# Делим
train_X, test_X, train_y, test_y = train_test_split(X, y, random_state = 42, test_size = 0.25)
```

Посмотрим сколько данных попала в обучающую и тестовые выборки

```
In [13]: print(len(train_X))
```

1095

```
In [14]: print(len(test_X))
```

365

В обучающей 1095, а в тестовой 365 домов.

Теперь как и в прошлый раз построим модель и посчитаем MAE

```
In [15]: # Инициализируем модель
home_model = DecisionTreeRegressor()
# Обучаем модель
home_model.fit(train_X, train_y)

# Предсказываем цены и считаем MAE
val_predictions = home_model.predict(test_X)
print(mean_absolute_error(test_y, val_predictions))
```

27109.372146118723

Воу. Это большая ошибка по сравнению с нашей изначальной ошибкой на обучающей выборке. И выходит, что для новых домов, наша модель цены предсказывала бы плохо.

Наша модель обучилась не очень оптимально, но к счастью есть несколько способов как можно улучшить модель, например, экспериментируя с параметрами модели.

Экспериментируем с моделями

Можно сказать что мы столкнулись с проблемой переобучения (overfitting), когда модель предсказывает обучающие данные практически идеально, а тестовые данные и другие данные предсказывает плохо.

Существует еще и недообучение (underfitting), когда модель не может найти параметры по которым можно хорошо разделить данные, и она плохо предсказывает даже на обучающих данных.

Так как мы используем DecisionTreeRegressor, т.е. дерево, в случае overfitting дерево обычно очень глубокое, а при underfitting получаем деревья небольшие. Поэтому мы можем попробовать регулировать глубину дерева. В DecisionTreeRegressor есть параметр max_leaf_nodes, который мы можем контролировать. Чем больше листьев мы разрешиим модели делать, тем дальше мы уйдем от underfitting, но будем приближаться к overfitting, т.е. в идеальном случае нам нужно подобрать примерно среднюю глубину дерева для модели.

Создадим функцию, которая будет строить для нас деревья с различным параметром max_leaf_nodes. А замерять модели будем снова через MAE.

```
In [16]: def get_mae(max_leaf_nodes, train_X, test_X, train_y, test_y):
    # инициализируем модель с параметром max_leaf_node, который мы
    передали в функцию
    model = DecisionTreeRegressor(max_leaf_nodes=max_leaf_nodes, ra
ndom_state=0)
    # обучаем модель на обучающих данных
    model.fit(train_X, train_y)
    # предсказываем на тестовых данных
    preds_val = model.predict(test_X)
    # считаем MAE
    mae = mean_absolute_error(test_y, preds_val)
    # возвращем значение ошибки
    return(mae)
```

В функцию мы соответственно передаем количество листов, которые мы хотим и обучающие и тестовые выборки. Сделаем небольшой цикл по различным значениям глубины от 5 до 5000 и выведем результаты.

```
In [17]: # сравниваем различные значения MAE для различной глубины деревьев
for max_leaf_nodes in [5, 10, 50, 100, 500, 1000, 5000]:
    my_mae = get_mae(max_leaf_nodes, train_X, test_X, train_y, test
_y)
    print("Максимальная глубина дерева: %d \t\t MEA: %d" %(max_le
af_nodes, my_mae))
```

Максимальная глубина дерева: 5	MEA: 31239
Максимальная глубина дерева: 10	MEA: 28860
Максимальная глубина дерева: 50	MEA: 23791
Максимальная глубина дерева: 100	MEA: 23124
Максимальная глубина дерева: 500	MEA: 24609
Максимальная глубина дерева: 1000	MEA: 25065
Максимальная глубина дерева: 5000	MEA: 25069

Как мы видим, самый лучший результат мы получили для глубины 100, но при этом ошибка достаточно большая и искажает предсказания для новых данных. С другой стороны, если мы посмотрим описательные статистики для нашей цены, мы увидим, что по сравнению со медианой и интерквартильным размахом, наша ошибка не настолько ужасна. Поэтому величина ошибки предсказания достаточно хитрая штука для интерпретации.

```
In [17]: train_y.describe()
```

```
Out[17]: count      1095.000000
          mean      181712.286758
          std       77955.082565
          min       34900.000000
          25%      130000.000000
          50%      165000.000000
          75%      215000.000000
          max       745000.000000
          Name: SalePrice, dtype: float64
```

Random Forest

Дерево решений оставляет с серьезным выбором. Глубокое дерево с overfitting с несколькими домами в каждом листе или мелкое дерево с несколькими листами, которое не может определить параметры и разделить данные? И все это с приличной ошибкой.

Некоторые другие модели деревьев смогли решить эту проблему и уменьшить при этом итоговую ошибку. Попробуем взять Random Forest. Он использует множество деревьев и делает предсказание путем усреднения прогнозов каждого дерева.

[Подробнее про случайный лес тут.](#)

(<https://dyakonov.org/2016/11/14/%D1%81%D0%BB%D1%83%D1%87%D0%B0%D0%B9%D0%BD%D0%BB%D0%B5%D1%81-random-forest/>)

Попробуем использовать Random Forest на наших данных. Импортируем его из модуля sklearn

```
In [18]: from sklearn.ensemble import RandomForestRegressor
```

Строим теперь модель с тем же параметрами и данными как в DecissionTreeRegressor

```
In [20]: # Инициализируем модель
forest_model = RandomForestRegressor()
# Обучаем модель
forest_model.fit(train_X, train_y)
# Делаем предсказание
randomf_preds = forest_model.predict(test_X)
# Посчитаем и выведем MAE
print(mean_absolute_error(test_y, randomf_preds))
```

19223.97926888454

Ошибка уменьшилась, но все равно составляет больше 20 тысяч долларов. Можно продолжить дальше экспериментировать с моделью, изменяя параметры или увеличивая обучающую выборку.

Давайте изменим параметр n_estimators, который отвечает за количество деревьев в модели (не путать с количеством листьев в DecisionTree)

```
In [23]: # Инициализируем модель
forest_model = RandomForestRegressor(n_estimators = 100)
# Обучаем модель
forest_model.fit(train_X, train_y)
# Делаем предсказание
randomf_preds = forest_model.predict(test_X)
# Посчитаем и выведем MAE
print(mean_absolute_error(test_y, randomf_preds))
```

19295.26145844749

Уже 19 тысяч! Много это или мало? Можно ли остановиться?

Конечно, мы всегда стараемся минимизировать нашу ошибку, но это и не всегда возможно. Возможно, здесь лучше справится другая модель. Так же мы взяли очень мало признаков, при чем не основываясь на какой-то особой теории - возможно, преобразование наших признаков или же их замена, улучшили бы качество.

С остальными параметрами можно ознакомиться в [документации \(<https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html>\)](https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html) функции

Дополнительным приятным бонусом использования моделей дерева и случайного леса является то, что мы можем использовать их определения важности признаков. Внутри себя дерево ранжирует признаки на основании того, насколько разбиение по ним снижает неоднородность данных. Так мы можем использовать лес для отбора наиболее важных признаков.

```
In [26]: list(zip(feature_columns, forest_model.feature_importances_))
```

```
Out[26]: [('OverallQual', 0.5888254349468932),
           ('GrLivArea', 0.18480296033254287),
           ('GarageArea', 0.04702267495102462),
           ('TotalBsmtSF', 0.09534714277278056),
           ('FullBath', 0.01472342776903842),
           ('YearBuilt', 0.038914154395293415),
           ('YearRemodAdd', 0.030364204832426855)]
```

```
In [27]: print("Features sorted by their score:")
print(sorted(zip(map(lambda x: round(x, 4), forest_model.feature_importances_), feature_columns),
              reverse=True))
```

```
Features sorted by their score:
[(0.5819, 'OverallQual'), (0.1839, 'GrLivArea'), (0.0998, 'TotalBsmtSF'), (0.0494, 'GarageArea'), (0.0381, 'YearBuilt'), (0.0293, 'YearRemodAdd'), (0.0176, 'FullBath')]
```

Давайте сравним с матрицей корреляции. Если бы мы ориентировались только на корреляции, мы считали два первых признака почти одинаково важным. По важностям, определенным решющим деревом видно, что 'OverallQual' с большим отрывом опережает площадь.

```
In [33]: home_data[feature_columns + ['SalePrice']].corr()['SalePrice']
```

```
Out[33]: OverallQual      0.790982
          GrLivArea       0.708624
          GarageArea       0.623431
          TotalBsmtSF     0.613581
          FullBath         0.560664
          YearBuilt        0.522897
          YearRemodAdd    0.507101
          SalePrice        1.000000
          Name: SalePrice, dtype: float64
```

На следующем семинаре мы вернемся к "Титанику" и попробуем, наконец, предсказать, кто же выжил во время крушения.

МИРЭК | 4 модуль

Автор: Татьяна Рогович

Основы программирования в Python

Семинар 8

Предсказываем выживших на Титаник: выбор лучшей модели

Мы уже немного познакомились с Титаником. Сегодня мы попробуем поучаствовать в тренировочном соревновании на kaggle: посмотрим, как это все в принципе работает, сделаем пару предсказаний (пока без статистических моделей, только на основе здравого смысла).

Чтобы участвовать в соревнованиях и скачивать данные, нужно зарегистрировать профиль на kaggle и залогиниться. Мы сегодня работаем с соревнованием по "Титанику".

<https://www.kaggle.com/c/titanic/overview> (<https://www.kaggle.com/c/titanic/overview>)

Скачаем и откроем (или загрузим из ссылок) все три набора данных, которые лежат во вкладке Data.

```
In [1]: import pandas as pd  
import numpy as np  
%matplotlib inline
```

```
In [2]: test = pd.read_csv('https://raw.githubusercontent.com/rogovich/2020_MIREC_PfDA/master/Seminars/S8_ML_Titanic/test.csv')  
train = pd.read_csv('https://raw.githubusercontent.com/rogovich/2020_MIREC_PfDA/master/Seminars/S8_ML_Titanic/train.csv')  
submission = pd.read_csv('https://raw.githubusercontent.com/rogovich/2020_MIREC_PfDA/master/Seminars/S8_ML_Titanic/gender.csv')
```

Инспектируем. Это датасет, с которым мы уже работали, здесь 891 пассажир и для них мы знаем значение признака Survived.

```
In [3]: display(train.head())
display(train.shape)
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.250
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...)	female	38.0	1	0	PC 17599	71.283
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.925
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.100
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.050

(891, 12)

А это так называемая тестовая выборка. В ML мы строим модель на тренировочных данных, для которых знаем значение зависимой переменной, и проверяем качество нашей модели на тестовых данных, для которых мы тоже знаем эту метку, но прячем ее от компьютера. Чтобы потом сравнить, что предсказал алгоритм с реальным значением. В kaggle тестовую метку Survived мы не видим, поэтому, чтобы проверить наше предсказание нам надо загрузить файл на сервер Kaggle, чтобы он его проверил (сравнил ваш ответ с правильным) и выдал вам оценку.

```
In [23]: display(test.head())
display(test.shape)
```

	PassengerId	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	E
0	892	3	Kelly, Mr. James	male	34.5	0	0	330911	7.8292	NaN	
1	893	3	Wilkes, Mrs. James (Ellen Needs)	female	47.0	1	0	363272	7.0000	NaN	
2	894	2	Myles, Mr. Thomas Francis	male	62.0	0	0	240276	9.6875	NaN	
3	895	3	Wirz, Mr. Albert	male	27.0	0	0	315154	8.6625	NaN	
4	896	3	Hirvonen, Mrs. Alexander (Helga E Lindqvist)	female	22.0	1	1	3101298	12.2875	NaN	

(418, 11)

В нашем соревновании лежит файл submission example, который показывает вам, как должен выглядеть файл для загрузки. В некоторых соревнованиях все устроено также, а в других вам надо загружать код, который будет обрабатывать скрытую тестовую выборку прямо на сервере.

```
In [24]: display(submission.head())
display(submission.shape)
```

	PassengerId	Survived
0	892	0
1	893	1
2	894	0
3	895	0
4	896	1

(418, 2)

Видим, что здесь только две колонки - id пассажира и метка Survived. Обратите внимание, что это пассажиры из тестовой выборки.

Помните, мы заполняли значение возраста? Одна из наших моделей сегодня будет учитывать возраст. Давайте посмотрим, есть ли пропущенные значения в тестовой выборке.

In [4]: test.info()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 418 entries, 0 to 417
Data columns (total 11 columns):
PassengerId    418 non-null int64
Pclass          418 non-null int64
Name            418 non-null object
Sex             418 non-null object
Age             332 non-null float64
SibSp           418 non-null int64
Parch           418 non-null int64
Ticket          418 non-null object
Fare            417 non-null float64
Cabin           91 non-null object
Embarked        418 non-null object
dtypes: float64(2), int64(4), object(5)
memory usage: 36.0+ KB
```

Гипотеза 1. Все утонули.

Когда мы начинаем анализ, мы впервые очередь смотрим на распределение нашей целевой (зависимой) переменной. Здесь мы хотим предсказывать Survived.

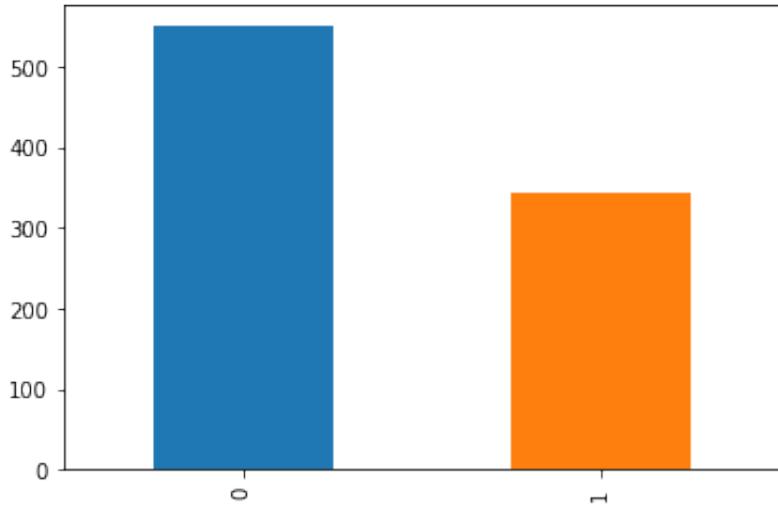
In [6]: train['Survived'].value_counts(normalize=True) # параметр normalize сразу выдает нам пропорции, а не абсолютные значения.

Out[6]: 0 0.616162
1 0.383838
Name: Survived, dtype: float64

Мы видим, что большинство пассажиров утонуло. Это будет наша baseline model - давайте попробуем сделать предсказание, что все утонули. Baseline нужен для того, чтобы понять, ниже какого предсказания не должны падать ваши модели. Наверное, если ваша модель работает хуже, чем просто "предсказать самый частый класс", то это не лучшая модель. Для количественной целевой переменной самым простым бэйзлайном может быть предсказание медианы и среднего этой переменной для каждого наблюдения.

```
In [7]: train['Survived'].value_counts().plot(kind='bar')
```

```
Out[7]: <matplotlib.axes._subplots.AxesSubplot at 0x2b46a93bb38>
```



Чтобы протестировать наше предположение - запишем его в файл и отправим на сервер.

```
In [9]: submission['Survived'] = 0 # ставим метку 0 во все ряды
```

```
In [13]: submission.to_csv('alldied.csv', index = False) # если не поставим  
index = False, то сохраним индекс в отдельную колонку и такой форма  
т файла не пройдет валидатор
```

Получили оценку! Наш score 0.62679. Для бинарной переменной этой значит, что мы всего в 62% случаев оказались правы. Давайте попробуем улучшить наше предсказание.

Гипотеза 2. Спасаем женщин.

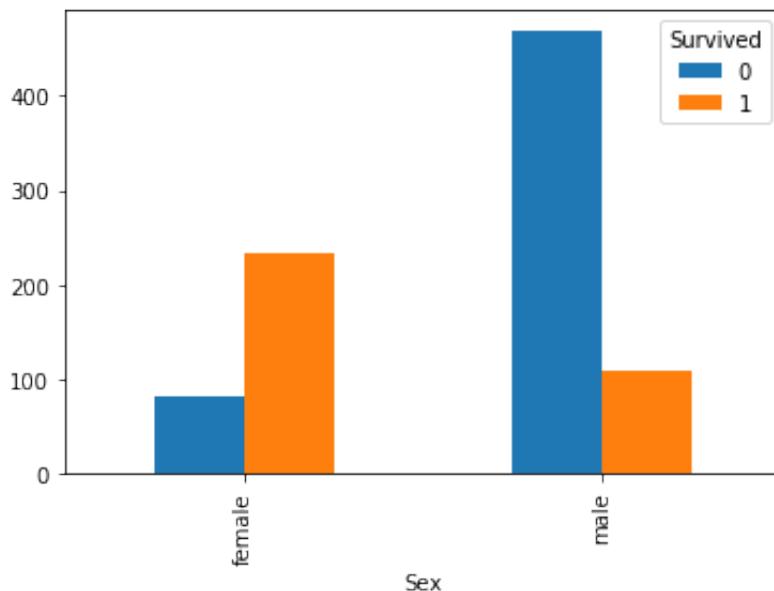
Если хоть что-то знаете про "Титаник", то наверняка помните историю про "первыми спасаем женщин и детей". Давайте попробуем улучшить наше предсказание, основывая на половом признаке.

```
In [14]: train.groupby('Sex')['Survived'].value_counts(normalize=True) #
```

```
Out[14]: Sex      Survived
female    1        0.742038
          0        0.257962
male     0        0.811092
         1        0.188908
Name: Survived, dtype: float64
```

```
In [31]: train.groupby('Sex')['Survived'].value_counts().unstack().plot(kind='bar')
```

```
Out[31]: <matplotlib.axes._subplots.AxesSubplot at 0x2182b2035c0>
```



Возможно, наше предположение верно. Пропорция выживших женщин больше пропорции выживших мужчин. Перезапишем наш файл.

```
In [15]: submission['Survived'] = test['Sex'].apply(lambda x: 1 if x == 'female' else 0)
submission.head()
```

```
Out[15]:
```

	PassengerId	Survived
0	892	0
1	893	1
2	894	0
3	895	0
4	896	1

```
In [16]: submission.to_csv('gender.csv', index = False)
```

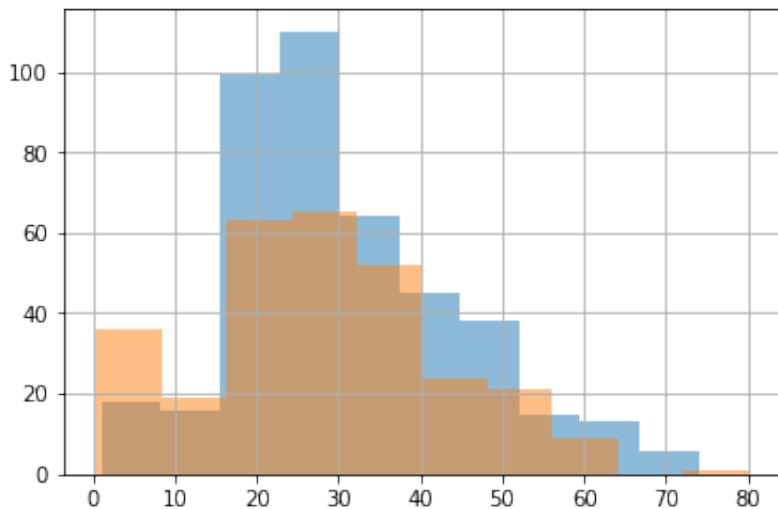
0.77! Неплохо, учитывая, что это была совсем простая эвристика. Попробуем еще лучше?

Гипотеза 3. Женщины и дети

А теперь давайте посмотрим, что там с возрастом.

```
In [17]: train.groupby('Survived')[['Age']].hist(alpha = 0.5, bins = 10) # параметр alpha задает прозрачность графика, bins - размер групп
```

```
Out[17]: Survived
0      AxesSubplot(0.125,0.125;0.775x0.755)
1      AxesSubplot(0.125,0.125;0.775x0.755)
Name: Age, dtype: object
```



Тут можно идти по возрастным группам и предположить, что дети до 10 лет имели больше шансов выжить, а вот молодые люди от 15 до 30 лет, скорее погибли. Но такое сложное условие уже оставим для решающих деревьев, а только скорректируем предсказание для детей младше 10 лет. Заодно посмотрим, как задавать новую переменную через метод библиотеки numpy.where

```
In [20]: submission['Survived'] = np.where((test['Sex'] == 'female') | (test['Age'] < 11), 1, 0) # в качестве аргументов передаем условие, затем , что делать, если True,
# что делать, если False
```

```
In [21]: submission.head()
```

Out[21]:

	PassengerId	Survived
0	892	0
1	893	1
2	894	0
3	895	0
4	896	1

```
In [22]: submission.to_csv('gender_child.csv', index = False)
```

0.77! Еще немного лучше. Давайте теперь попробуем применить к нашим данным алгоритмы машинного обучения и посмотрим, справятся ли они лучше. У нас достаточно серьезный baseline - 77% правильных ответов. Будем стараться преодолеть это значение.

Предсказываем выживших с помощью ML

Объединяя train и test, для того, чтобы обрабатывать пропущенные значения и создавать признаки. Но разведывательный анализ будем делать только на тренировочной выборке.

```
In [23]: import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.tree import DecisionTreeClassifier

%matplotlib inline
```

```
In [29]: def plot_categories( df , cat , target , **kwargs ):
    row = kwargs.get( 'row' , None )
    col = kwargs.get( 'col' , None )
    facet = sns.FacetGrid( df , row = row , col = col )
    facet.map( sns.barplot , cat , target )
    facet.add_legend()

def plot_correlation_map( df ):
    corr = df.corr()
    _ , ax = plt.subplots( figsize =( 12 , 10 ) )
    cmap = sns.diverging_palette( 220 , 10 , as_cmap = True )
    _ = sns.heatmap(
        corr,
        cmap = cmap,
        square=True,
        cbar_kws={ 'shrink' : .9 },
        ax=ax,
        annot = True,
        annot_kws = { 'fontsize' : 12 }
    )

def plot_variable_importance( X , y ):
    tree = DecisionTreeClassifier( random_state = 99 )
    tree.fit( X , y )
    plot_model_var_imp( tree , X , y )

def plot_model_var_imp( model , X , y ):
    imp = pd.DataFrame(
        model.feature_importances_ ,
        columns = [ 'Importance' ] ,
        index = X.columns
    )
    imp = imp.sort_values( [ 'Importance' ] , ascending = True )
    imp[ : 10 ].plot( kind = 'barh' )
    print (model.score( X , y ))
```

```
In [25]: full = train.append(test, ignore_index = True, sort=False)

del train, test

print ('Datasets:' , 'full:' , full.shape)
```

Datasets: full: (1309, 12)

```
In [30]: plot_correlation_map(full[:891])
```



Нет особо сильных корреляций с зависимой переменной, но видим умеренную корреляцию между Fare и Pclass. Пожалуй, выберем из них одну.

Создание новых переменных: Title

Создадим переменную титул, посмотрим, как она связана с выживанием + используем ее для заполнения пропусков в возрасте.

```
In [31]: full['Title'] = full['Name'].apply(lambda name: name.split(',')[1].split('.')[0].strip())
full['Title'] = full['Title'].apply(lambda x: x if x in
                                    ['Mr', 'Miss', 'Mrs', 'Master',
                                     'Rev', 'Dr'] else 'Misc')
full['Title'].value_counts()
```

```
Out[31]: Mr      757
Miss    260
Mrs     197
Master   61
Misc     18
Dr       8
Rev      8
Name: Title, dtype: int64
```

Заполняем пропуски в Age и Embarked

Теперь используем титул для заполнения пропущенных значений. Опять работаем на всей выборке. Сначала посмотрим, где у нас есть пропущенные значения.

```
In [32]: full.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1309 entries, 0 to 1308
Data columns (total 13 columns):
PassengerId    1309 non-null int64
Survived        891 non-null float64
Pclass          1309 non-null int64
Name            1309 non-null object
Sex             1309 non-null object
Age             1046 non-null float64
SibSp           1309 non-null int64
Parch           1309 non-null int64
Ticket          1309 non-null object
Fare            1308 non-null float64
Cabin           295 non-null object
Embarked        1307 non-null object
Title           1309 non-null object
dtypes: float64(3), int64(4), object(6)
memory usage: 133.0+ KB
```

Обработаем пропущенные значения в переменных Age (по титулу), Fare (средним по классу), Embarked (модой).

```
In [33]: full['Age'] = full.Age.fillna(full.groupby('Title')['Age'].transform('median'))

In [34]: full['Fare'] = full.Fare.fillna(full.groupby('Pclass')['Fare'].transform('median'))

In [35]: full['Embarked'] = full.Embarked.fillna(full['Embarked'].mode()[0])

In [36]: full.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1309 entries, 0 to 1308
Data columns (total 13 columns):
PassengerId    1309 non-null int64
Survived        891 non-null float64
Pclass          1309 non-null int64
Name            1309 non-null object
Sex             1309 non-null object
Age             1309 non-null float64
SibSp           1309 non-null int64
Parch           1309 non-null int64
Ticket          1309 non-null object
Fare            1309 non-null float64
Cabin           295 non-null object
Embarked        1309 non-null object
Title           1309 non-null object
dtypes: float64(3), int64(4), object(6)
memory usage: 133.0+ KB
```

Так же пропуски можно заполнять с помощью алгоритмов машинного обучения - например, попробуйте заполнить пропуски в возрасте с помощью алгоритма kNN, используя все остальные переменные, кроме Survived, чтобы рассчитать расстояние между точками (такой вопрос будет в контрольной).

Создание новых переменных: размер семьи

Мы тоже уже это делали - складываем горизонтальных и вертикальных родственников, а также учитываем самого пассажира.

```
In [37]: full['FamilySize'] = full['Parch'] + full['SibSp'] + 1
```

А теперь давайте попробуем понять, имеет ли смысл объединить категории.

```
In [38]: full[:891].groupby('FamilySize')['Survived'].count()
```

Out[38]: FamilySize

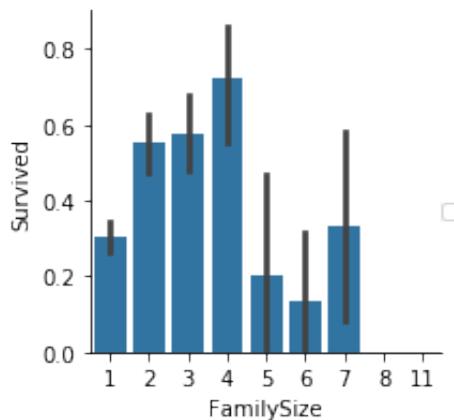
1	537
2	161
3	102
4	29
5	15
6	22
7	12
8	6
11	7

Name: Survived, dtype: int64

```
In [39]: plot_categories(full[:891], cat = 'FamilySize', target = 'Survived')
```

C:\Users\rogov\Anaconda3\lib\site-packages\seaborn\axisgrid.py:703
: UserWarning: Using the barplot function without specifying `order` is likely to produce an incorrect plot.

```
warnings.warn(warning)
```



Мы видим, что в семьях размеров 2-4 процент выживших похож. И нет значительной разницы между семьями больше 5 человек. Давайте используем эту информацию для перекодирования переменных.

```
In [40]: full['Family_Single'] = full['FamilySize'].apply(lambda s: 1 if s == 1 else 0)
full['Family_Small'] = full['FamilySize'].apply(lambda s: 1 if 2 <= s <= 4 else 0)
full['Family_Large'] = full['FamilySize'].apply(lambda s: 1 if 5 <= s else 0)

full.head()
```

Out[40]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fai
0	1	0.0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.250
1	2	1.0	1	Cumings, Mrs. John Bradley (Florence Briggs Th...)	female	38.0	1	0	PC 17599	71.283
2	3	1.0	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.925
3	4	1.0	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.100
4	5	0.0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.050

Подготовка данных

Часть алгоритмов гораздо эффективней работает с переменными, которые закодированы как бинарные. Для этого можно применять метод OneHotEncoding, а можно сделать это методами pandas.

```
In [44]: sex = pd.get_dummies(full.Sex , prefix='Sex')
sex.head()
```

Out[44]:

	Sex_female	Sex_male
0	0	1
1	1	0
2	1	0
3	1	0
4	0	1

```
In [46]: embarked = pd.get_dummies(full.Embarked , prefix='Embarked' )
embarked.head()
```

Out[46]:

	Embarked_C	Embarked_Q	Embarked_S
0	0	0	1
1	1	0	0
2	0	0	1
3	0	0	1
4	0	0	1

```
In [47]: pclass = pd.get_dummies(full.Pclass , prefix='Pclass' )
pclass.head()
```

Out[47]:

	Pclass_1	Pclass_2	Pclass_3
0	0	0	1
1	1	0	0
2	0	0	1
3	1	0	0
4	0	0	1

```
In [48]: title = pd.get_dummies(full.Title , prefix='Title' )
title.head()
```

Out[48]:

	Title_Dr	Title_Master	Title_Misc	Title_Miss	Title_Mr	Title_Mrs	Title_Rev
0	0	0	0	0	1	0	0
1	0	0	0	0	0	1	0
2	0	0	0	1	0	0	0
3	0	0	0	0	0	1	0
4	0	0	0	0	1	0	0

Объединяем датасет

Теперь все наши обработанные переменные объединим в один датасет, а потом разделим его обратно на нашу тренировочную и тестовую выборки.

```
In [49]: full_X = pd.concat([full['Age'] , embarked, title, pclass, full['Family_Single'] , full['Family_Small'] , full['Family_Large']] , axis=1)
full_X.head()
```

Out[49]:

	Age	Embarked_C	Embarked_Q	Embarked_S	Title_Dr	Title_Master	Title_Misc	Title_N
0	22.0	0	0	1	0	0	0	0
1	38.0	1	0	0	0	0	0	0
2	26.0	0	0	1	0	0	0	0
3	35.0	0	0	1	0	0	0	0
4	35.0	0	0	1	0	0	0	0

Я не беру оплату за проезд, потому что она коррелирует с классом и полом, потому что он "зашит" в титул.

```
In [50]: from sklearn.model_selection import train_test_split

train_valid_X = full_X[:891]
train_valid_y = full[:891].Survived

train_X , test_X , train_y , test_y = train_test_split(train_valid_X , train_valid_y , test_size = 0.3)

print(full_X.shape , train_X.shape, train_y.shape, test_X.shape, test_y.shape)
```

(1309, 17) (623, 17) (623,) (268, 17) (268,)

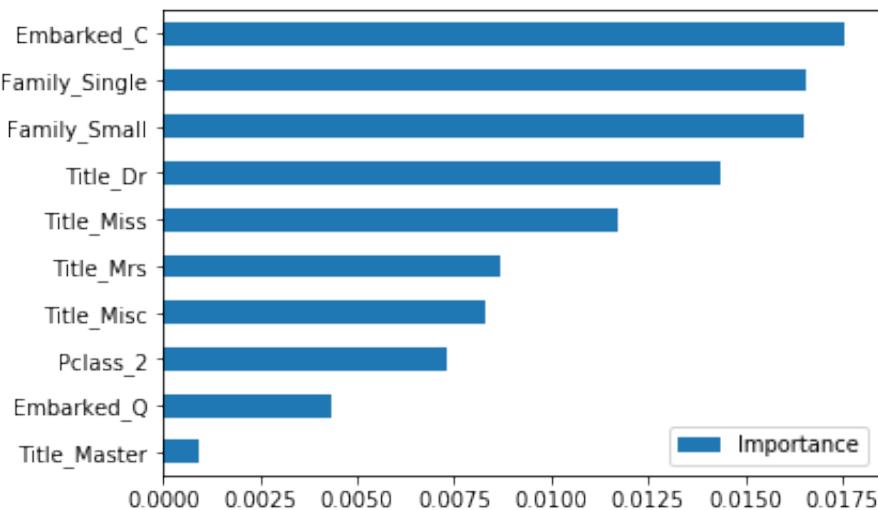
Пробуем разные модели

```
In [52]: from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier , GradientBoostingClassifier
```

У нас есть функция, которая строит график для важности признаков на основе разбиения DesicionTreeClassifier

```
In [53]: plot_variable_importance(train_X, train_y)
```

0.9293739967897271



Если вы дальше хотите экспериментировать с добавлением или удалением моделей, то этот график как раз вам поможет.

А теперь как раз перейдем к построению разных моделей. Метрикой ошибки возьмем accuracy, как и в соревновании.

Кросс-валидация, поиск параметров по сетке, случайный лес

```
In [54]: from sklearn.model_selection import cross_validate
```

```
In [55]: model = RandomForestClassifier(n_estimators=100)
```

Про кросс-валидацию можно подробнее прочитать здесь: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.cross_validate.html (https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.cross_validate.html)

Мы передаем функции cross_validate модель, тренировные x и y, указываем на сколько частей будем разбивать тренировочную выборку, метрику ошибки, и возвращать ли ошибку на тренировочной выборке.

```
In [56]: scores = cross_validate(model, train_X, train_y, cv=3, scoring='accuracy')
```

```
In [57]: scores
```

```
Out[57]: {'fit_time': array([0.30818653, 0.15059543, 0.28922462]),  
          'score_time': array([0.01795197, 0.01595712, 0.01695466]),  
          'test_score': array([0.80769231, 0.76923077, 0.7826087 ])}
```

Нас здесь интересует именно test_score - мы смотрим, что она не сильно скачет в зависимости от того, какую часть выборки мы отложили для теста.

```
In [88]: np.mean(scores['test_score'])
```

```
Out[88]: 0.7592205499814195
```

А вот так в среднем вела себя модель. Давайте попробуем написать функцию которая еще и будет делать подбор параметров по сетке внутри кросс-валидации.

```
In [58]: from sklearn.model_selection import GridSearchCV
```

```
In [59]: parameters = {'n_estimators':[10, 50, 100, 200]}
model_cv = GridSearchCV(RandomForestClassifier(), parameters,
cv=3, scoring=('accuracy'))
```

```
In [60]: model_cv.fit(train_X, train_y)
```

```
Out[60]: GridSearchCV(cv=3, estimator=RandomForestClassifier(),
param_grid={'n_estimators': [10, 50, 100, 200]},
scoring='accuracy')
```

```
In [118]: model_cv.cv_results_
```

```
Out[118]: {'mean_fit_time': array([0.0269293 , 0.10604914, 0.28557007, 0.363
69483]),
'std_fit_time': array([0.00293653, 0.01316522, 0.16577883, 0.0590
9422]),
'mean_score_time': array([0.00132974, 0.00797892, 0.01196758, 0.0
1928147]),
'std_score_time': array([0.00047013, 0.00293655, 0.00162898, 0.00
329161]),
'param_n_estimators': masked_array(data=[10, 50, 100, 200],
mask=[False, False, False, False],
fill_value='?',
dtype=object),
'params': [{'n_estimators': 10},
{'n_estimators': 50},
{'n_estimators': 100},
{'n_estimators': 200}],
'split0_test_score': array([0.74038462, 0.73557692, 0.73076923, 0
.73076923]),
'split1_test_score': array([0.76923077, 0.77884615, 0.75480769, 0
.75480769]),
'split2_test_score': array([0.76811594, 0.77294686, 0.77294686, 0
.76811594]),
'mean_test_score': array([0.75922953, 0.76243981, 0.75280899, 0.7
5120385]),
'std_test_score': array([0.01334917, 0.01916957, 0.01727005, 0.01
545265]),
'rank_test_score': array([2, 1, 3, 4]),
'split0_train_score': array([0.9060241 , 0.91566265, 0.91807229,
0.91807229]),
'split1_train_score': array([0.90120482, 0.91566265, 0.91566265,
0.91566265]),
'split2_train_score': array([0.92307692, 0.92548077, 0.92548077,
0.92548077]),
'mean_train_score': array([0.91010195, 0.91893536, 0.91973857, 0.
91973857]),
'std_train_score': array([0.00938328, 0.00462831, 0.00417782, 0.0
0417782])}
```

Здесь теперь анализируем иначе. Так 'split0_test_score': array([0.73076923, 0.75, 0.74038462, 0.74038462]) это значения каждого из параметров n_estimators по первой разбивке. Но не пугайтесь, лучшую модель выбрать несложно - нужно посмотреть на средний score или на ранги (rank_test_score). По рангу понимаем, что лучше всего справился лес с 50 деревьями.

```
In [92]: model_cv.cv_results_['rank_test_score']
```

```
Out[92]: array([3, 1, 2, 4])
```

Давайте посмотрим теперь внимательно на среднюю скорость по тестам. То же самое.

```
In [93]: model_cv.cv_results_['mean_test_score']
```

```
Out[93]: array([0.759267, 0.76565403, 0.76564629, 0.75921281])
```

Давайте теперь эту модель проверим на нашей отложенной тестовой выборке.

```
In [94]: best_model = RandomForestClassifier(n_estimators=50)
```

```
In [95]: best_model.fit(train_X, train_y)
```

```
Out[95]: RandomForestClassifier(n_estimators=50)
```

```
In [96]: from sklearn.metrics import accuracy_score  
accuracy_score(best_model.predict(test_X), test_y)
```

```
Out[96]: 0.746268656716418
```

Запомнили значение. Чуть позже выберем нашего победителя и отправим его на kaggle.

Напоминаю, что лучшее наше предсказание по baseline модели было 0.77 (все женщины и дети выжили).

KNN

Давайте теперь попробуем KNN. Тоже с поиском параметров по сетке.

```
In [97]: parameters = {'n_neighbors':[3, 5, 10, 15]}  
model_cv = GridSearchCV(KNeighborsClassifier(), parameters,  
cv=3, scoring='accuracy')
```

```
In [98]: model_cv.fit(train_X, train_y)
```

```
Out[98]: GridSearchCV(cv=3, estimator=KNeighborsClassifier(),
                      param_grid={'n_neighbors': [3, 5, 10, 15]},
                      return_train_score=True, scoring='accuracy')
```

```
In [99]: model_cv.cv_results_
```

```
Out[99]: {'mean_fit_time': array([0.00629656, 0.00465886, 0.00401306, 0.006
6483]),
 'std_fit_time': array([0.0033048 , 0.00124711, 0.00083435, 0.0016
9604]),
 'mean_score_time': array([0.00432412, 0.00400337, 0.00398358, 0.0
0432054]),
 'std_score_time': array([4.67925130e-04, 1.34438390e-05, 8.189741
32e-04, 4.71430981e-04]),
 'param_n_neighbors': masked_array(data=[3, 5, 10, 15],
                                    mask=[False, False, False, False],
                                    fill_value='?',
                                    dtype=object),
 'params': [{'n_neighbors': 3},
 {'n_neighbors': 5},
 {'n_neighbors': 10},
 {'n_neighbors': 15}],
 'split0_test_score': array([0.70192308, 0.73076923, 0.63461538, 0
.64903846]),
 'split1_test_score': array([0.73557692, 0.74519231, 0.71153846, 0
.68269231]),
 'split2_test_score': array([0.70048309, 0.69082126, 0.74879227, 0
.75845411]),
 'mean_test_score': array([0.71266103, 0.72226093, 0.69831537, 0.6
9672829]),
 'std_test_score': array([0.01621464, 0.02299777, 0.04754106, 0.04
575808]),
 'rank_test_score': array([2, 1, 3, 4]),
 'split0_train_score': array([0.84337349, 0.8313253 , 0.73975904,
0.74939759]),
 'split1_train_score': array([0.84578313, 0.81445783, 0.7253012 ,
0.70361446]),
 'split2_train_score': array([0.81009615, 0.79807692, 0.73076923,
0.69230769]),
 'mean_train_score': array([0.83308426, 0.81462002, 0.73194316, 0.
71510658]),
 'std_train_score': array([0.01628479, 0.01357408, 0.00596047, 0.0
2468287])}
```

Тут уже на кросс-валидации видим, что качество ниже RandomForest, но можем протестировать модель с 3 соседями на тестовой выборке.

```
In [100]: best_model = KNeighborsClassifier(n_neighbors=3)
best_model.fit(train_X, train_y)
accuracy_score(best_model.predict(test_X), test_y)
```

```
Out[100]: 0.7238805970149254
```

Модель справилась хуже. Идем экспериментировать дальше.

Логистическая регрессия

Попробуем, наверняка, знакомую вам логистическую регрессию. Она как раз должна здорово работать с бинарным таргетом. Тут будем подбирать параметр регуляризации - алгоритм, который зануляет значения некоторых переменных для более эффективной работы модели.

https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html
[\(https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html\)](https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html)

```
In [101]: parameters = {'penalty':['l1', 'l2']}
model_cv = GridSearchCV(LogisticRegression(), parameters,
                        cv=3, scoring=('accuracy'))
```

```
In [102]: model_cv.fit(train_X, train_y)
```

```
C:\Users\rogov\Anaconda3\lib\site-packages\sklearn\model_selection\_validation.py:552: FitFailedWarning: Estimator fit failed. The score on this train-test partition for these parameters will be set to nan. Details:
Traceback (most recent call last):
  File "C:\Users\rogov\Anaconda3\lib\site-packages\sklearn\model_selection\_validation.py", line 531, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "C:\Users\rogov\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py", line 1302, in fit
    solver = _check_solver(self.solver, self.penalty, self.dual)
  File "C:\Users\rogov\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py", line 443, in _check_solver
    "got %s penalty." % (solver, penalty))
ValueError: Solver lbfgs supports only 'l2' or 'none' penalties, got l1 penalty.
```

```
FitFailedWarning)
C:\Users\rogov\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:762: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. OF ITERATIONS REACHED LIMIT.
```

Increase the number of iterations (max_iter) or scale the data as shown in:

```
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
    extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
C:\Users\rogov\Anaconda3\lib\site-packages\sklearn\linear_model\_\_
logistic.py:762: ConvergenceWarning: lbfgs failed to converge (stat
us=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

Increase the number of iterations (`max_iter`) or scale the data as shown in:

```
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
    extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
C:\Users\rogov\Anaconda3\lib\site-packages\sklearn\linear_model\_\_
logistic.py:762: ConvergenceWarning: lbfgs failed to converge (stat
us=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

Increase the number of iterations (`max_iter`) or scale the data as shown in:

```
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
    extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
```

```
Out[102]: GridSearchCV(cv=3, estimator=LogisticRegression(),
                      param_grid={'penalty': ['l1', 'l2']}, return_train_sc
ore=True,
                      scoring='accuracy')
```

In [103]: model_cv.cv_results_

```
Out[103]: {'mean_fit_time': array([0.00232553, 0.04737655]),
 'std_fit_time': array([0.00047137, 0.01142082]),
 'mean_score_time': array([0.          , 0.00283066]),
 'std_score_time': array([0.          , 0.00022872]),
 'param_penalty': masked_array(data=['l1', 'l2'],
                                mask=[False, False],
                                fill_value='?',
                                dtype=object),
 'params': [{('penalty': 'l1'), ('penalty': 'l2')},
 'split0_test_score': array([      nan, 0.79807692]),
 'split1_test_score': array([      nan, 0.82211538]),
 'split2_test_score': array([      nan, 0.83091787]),
 'mean_test_score': array([      nan, 0.81703673]),
 'std_test_score': array([      nan, 0.01387988]),
 'rank_test_score': array([2, 1]),
 'split0_train_score': array([      nan, 0.82409639]),
 'split1_train_score': array([      nan, 0.81445783]),
 'split2_train_score': array([      nan, 0.81490385]),
 'mean_train_score': array([      nan, 0.81781935]),
 'std_train_score': array([      nan, 0.00444226])}
```

Наконец-то пробили 80%. Видим, что логистическая регрессия со штрафом l2 справилась лучше, давайте ее и проверим на отложенной выборке.

In [108]: best_model = LogisticRegression(penalty = 'l2')
best_model.fit(train_X, train_y)
accuracy_score(best_model.predict(test_X), test_y)

```
C:\Users\rogov\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:762: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. OF ITERATIONS REACHED LIMIT.
```

Increase the number of iterations (max_iter) or scale the data as shown in:

```
https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
https://scikit-learn.org/stable/modules/linear\_model.html#logistic-regression
extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
```

Out[108]: 0.8432835820895522

Градиентный бустинг

Последний классификатор, на который сегодня посмотрим - градиентный бустинг. Как и случайный лес - это ансамбль (делает предсказание на основании многих алгоритмов). Как случайный лес, использует решающие деревья. Главное отличие в том, что случайный лес строит много-много деревьев, а потом выбирает предсказание из них за счет голосования или усреднения непрерывное предсказание. Градиентный бустинг строит по одному дереву, для каждого из них считает ошибку и каждым последующим деревом пытается минимизировать ошибку предыдущего. Давайте посмотрим, как он справится здесь.

<https://www.datasciencecentral.com/profiles/blogs/decision-tree-vs-random-forest-vs-boosted-trees-explained> (<https://www.datasciencecentral.com/profiles/blogs/decision-tree-vs-random-forest-vs-boosted-trees-explained>)

```
In [109]: scores = cross_validate(GradientBoostingClassifier(), train_X, train_y, cv=3, scoring='accuracy')  
scores
```

```
Out[109]: {'fit_time': array([0.11771512, 0.11326528, 0.11671805]),  
           'score_time': array([0.00299263, 0.00199533, 0.0019958]),  
           'test_score': array([0.79326923, 0.79807692, 0.81642512]),  
           'train_score': array([0.8626506, 0.83855422, 0.83653846])}
```

```
In [110]: best_model = GradientBoostingClassifier()  
best_model.fit(train_X, train_y)  
accuracy_score(best_model.predict(test_X), test_y)
```

```
Out[110]: 0.8283582089552238
```

Kaggle Submission

Все равно хуже, чем логистическая регрессия. Давайте сгенерируем для нее предсказание и проверим его на kaggle.

```
In [61]: model = LogisticRegression(penalty = 'l2')
test_X = full_X[891:]
model.fit(train_valid_X, train_valid_y)
pred = model.predict(test_X)
test = pd.DataFrame({ 'PassengerId': full[891:].PassengerId , 'Survived': pred})
test[ 'Survived' ] = test[ 'Survived' ].apply(lambda x: int(x))
print(test.shape)
test.head()
```

(418, 2)

```
C:\Users\rogov\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:762: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. OF ITERATIONS REACHED LIMIT.
```

Increase the number of iterations (max_iter) or scale the data as shown in:

```
https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
```

Out[61]:

	PassengerId	Survived
891	892	0
892	893	1
893	894	0
894	895	0
895	896	1

```
In [62]: test.to_csv('titanic_pred_lr.csv' , index = False )
```

На тестовой выборке kaggle получили 78% - хуже чем для нашего эксперимента, но лучше baseline (небольшая, но победа). Если вы хотите дальше развиваться в машинном обучении - kaggle это отличное место, чтобы обрабатывать полученные навыки.

In []: