**Seaborn для визуализации данных в Python**

Seaborn — библиотека для создания статистических графиков на Python. Она построена на основе matplotlib и тесно интегрируется со структурами данных pandas. Seaborn помогает вам изучить и понять данные. Его функции построения графиков работают с датасетами и выполняют все необходимы преобразования для создания информативных графиков.

Синтаксис, ориентированный на набор данных, позволяет сосредоточиться на графиках, а не деталях их построения.

Официальная документация на английском: [**https://seaborn.pydata.org/index.html**](https://seaborn.pydata.org/index.html).

**Установка seaborn**

Официальные релизы seaborn можно установить из PyPI:

pip install seaborn

Библиотека также входит в состав дистрибутива **Anaconda**:

conda install seaborn

*Библиотека работает с Python версии 3.6+. Если их еще нет, эти библиотеки будут загружены при установке seaborn: numpy, scipy, pandas, matplotlib.*

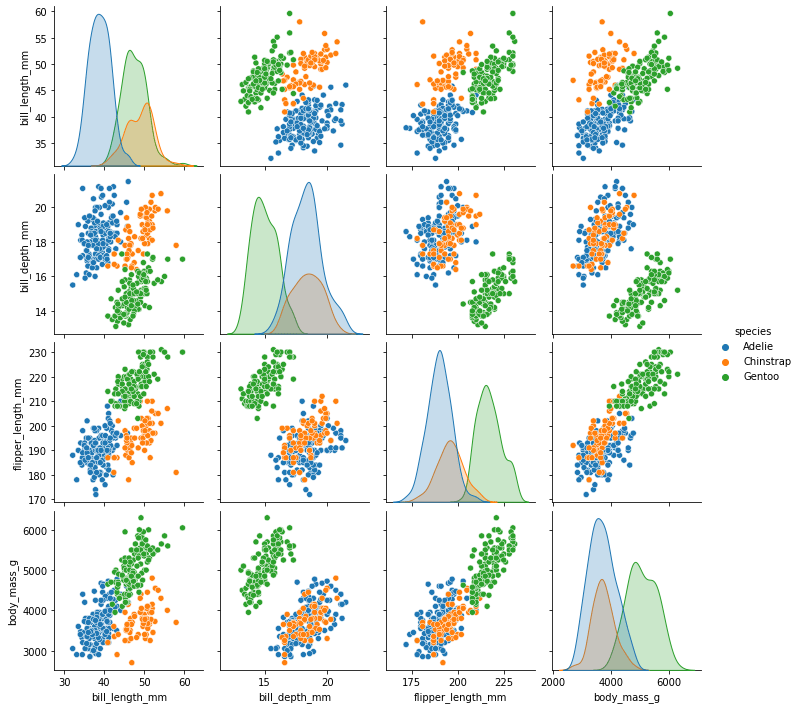
Как только вы установите Seaborn, можете скачать и построить тестовый график для одного из встроенных датасетов:

import seaborn as sns

df = sns.load\_dataset("penguins")

sns.pairplot(df, hue="species")

Выполнив этот код в Jupyter Notebook, увидите такой график.



Если вы не работаете с Jupyter, может потребоваться явный вызов matplotlib.pyplot.show():

import matplotlib.pyplot as plt

plt.show()

*Весь дальнейший код будет выполняться в Jupyter Notebook*

**Построение Bar Plot в Seaborn**

Гистограммы отображают числовые величины на одной оси и переменные категории на другой. Они позволяют вам увидеть, значения параметров для каждой категории.

Гистограммы можно использовать для визуализации временных рядов, а также только категориальных данных.

**Построение гистограммы**

Чтобы нарисовать гистограмму в Seaborn нужно вызвать функцию barplot(), и передать ей категориальные и числовые переменные, которые нужно визуализировать, как это сделано в примере:

import matplotlib.pyplot as plt

import seaborn as sns

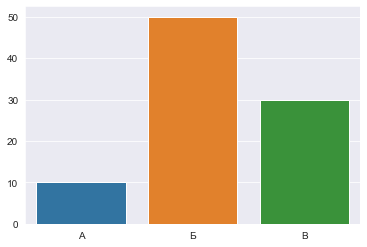
x = ['А', 'Б', 'В']

y = [10, 50, 30]

sns.barplot(x=x, y=y);

В данном случае, у нас есть несколько категориальных переменных в списке — А, Б и В. А также непрерывные переменные (числа) в другом списке — 10, 50 и 30. Зависимость между этими двумя элементами визуализируется на гистограмме, для чего эти два списка передаются в функцию sns.barplot().

В результате получается четкая и простая гистограмма:



Чаще всего вы будете работать с датасетами, которые содержат гораздо больше данных, чем тот что приведен в примере. Иногда к этим наборам данным требуется сортировка, или подсчитать, сколько раз повторяются то или другое значение.

Когда вы работаете с данными можете столкнуться с ошибками и пропусками, которые в них имеются. К счастью, Seaborn защищает нас и автоматически применяет фильтр, который основан на вычислении среднего значения предоставленных данных.

Давайте импортируем классический датасет Titanic и визуализируем Bar Plot с этими данными:

# Импорт данных

titanic\_dataset = sns.load\_dataset("titanic")

# Постройка графика

sns.barplot(x="sex", y="survived", data=titanic\_dataset);

В данном случае мы назначили осям Х и Y колонки "sex" и "survived", вместо жестко заданных.

Если мы выведем первые строки датасета (titanic\_dataset.head()), увидим такую таблицу:

survived pclass sex age sibsp parch fare ...

0 0 3 male 22.0 1 0 7.2500 ...

1 1 1 female 38.0 1 0 71.2833 ...

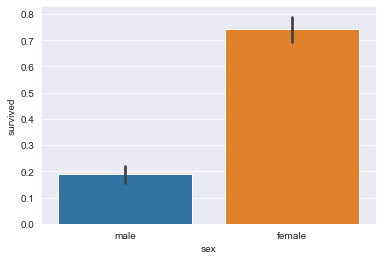
2 1 3 female 26.0 0 0 7.9250 ...

3 1 1 female 35.0 1 0 53.1000 ...

4 0 3 male 35.0 0 0 8.0500 ...

Убедитесь, что имена колонок совпадают с теми, которые вы назначили переменным x и y.

Наконец, мы используем эти данные и передаем их в качестве аргумента функции, с которой работаем. И получаем такой результат:



**Построение горизонтальной гистограммы**

Чтобы нарисовать горизонтальную, а не вертикальную гистограмму нужно просто поменять местами переменные передаваемые в x и y.

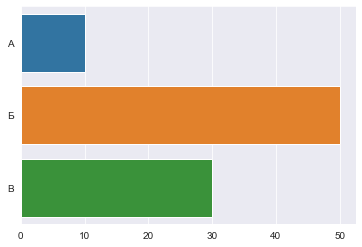
В этом случае категориальная переменная будет отображаться по оси Y, что приведет к постройке горизонтального графика:

x = ['А', 'Б', 'В']

y = [10, 50, 30]

sns.barplot(x=y, y=x);

График будет выглядеть так:



**Как изменить цвет в barplot()**

Изменить цвет столбцов довольно просто. Для этого нужно задать параметр color функции barplot и тогда цвет всех столбцов изменится на заданный.

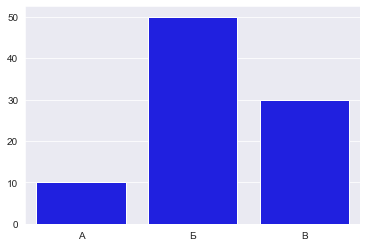
Изменим на голубой:

x = ['А', 'Б', 'В']

y = [10, 50, 30]

sns.barplot(x=x, y=y, color='blue');

Тогда график будет выглядеть так:



Можно установить аргумент pallete, который может принимать большое количество цветов. Довольно распространенное значение этого параметра hls:

sns.barplot(

x="embark\_town",

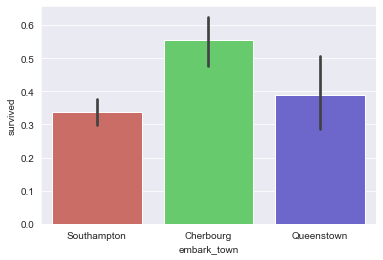
y="survived",

palette='hls',

data=titanic\_dataset

);

Что приведет к такому результату:



**Группировка Bar Plot в Seaborn**

Часто требуется сгруппировать столбцы на графиках по одному признаку. Допустим, вы хотите сравнить некоторые общие данные, выживаемость пассажиров, и сгруппировать их по заданным критериям.

Нам может потребоваться визуализировать количество выживших пассажиров, в зависимости от класса (первый, второй и третий), но также учесть, города из которого они прибыли.

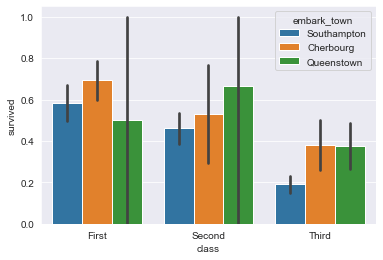
Всю эту информацию можно легко отобразить на гистограмме.

Чтобы сгруппировать столбцы вместе, мы используем аргумент hue. Этот аргумент группирует соответствующие данные и сообщает библиотеке Seaborn, как раскрашивать столбцы.

Давайте посмотрим на только что обсужденный пример:

sns.barplot(x="class", y="survived", hue="embark\_town", data=titanic\_dataset);

Получим такой график:



**Настройка порядка отображения групп столбцов на гистограмме**

Вы можете изменить порядок следования столбцов по умолчанию. Это делается с помощью аргумента order, который принимает список значений и порядок их размещения.

Например, до сих пор он упорядочивал классы с первого по третий. Что, если мы захотим сделать наоборот?

sns.barplot(

x="class",

y="survived",

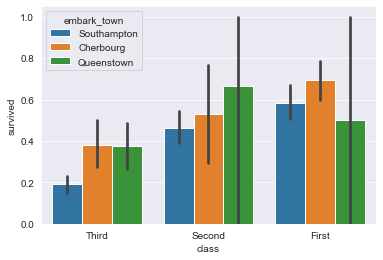
hue="embark\_town",

order=["Third", "Second", "First"],

data=titanic\_dataset

);

Получится такой график:



**Изменяем доверительный интервал в barplot()**

Вы также можете поэкспериментировать с доверительным интервалом, задав аргумент ci.

Например, вы можете отключить его, установив для него значение None, или использовать стандартное отклонение вместо среднего, установив sd, или даже установить верхний предел на шкале ошибок, установив capsize.

Давайте немного поэкспериментируем с атрибутом доверительного интервала:

sns.barplot(

x="class",

y="survived",

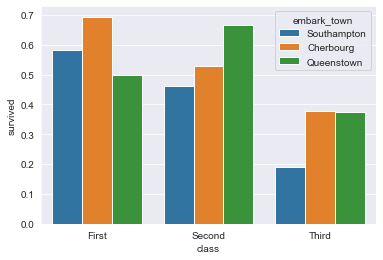
hue="embark\_town",

ci=None,

data=titanic\_dataset

);

Получим такой результат:



Или мы можем использовать стандартное отклонение:

sns.barplot(

x="class",

y="survived",

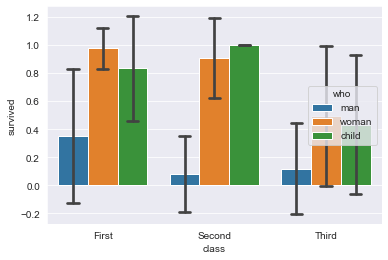
hue="who",

ci="sd",

capsize=0.1,

data=titanic\_dataset

);



Мы рассмотрели несколько способов построения гистограммы в Seaborn на примерах. Теперь перейдем к тепловым картам.

**Построение Heatmap в Seaborn**

Давайте посмотрим, как мы можем работать с библиотекой Seaborn на Python, чтобы создать базовую тепловую карту корреляции.  
Для наших целей мы будем использовать [**набор данных о жилье Ames**](https://www.kaggle.com/prevek18/ames-housing-dataset), доступный на Kaggle.com. Он содержит более 30 показателей, которые потенциально могут повлиять на стоимость недвижимости.

Поскольку Seaborn была написана на основе библиотеки визуализации данных Matplotlib, их довольно просто использовать вместе. Поэтому помимо стандартных модулей мы также собираемся импортировать Matplotlib.pyplot.

import pandas as pd

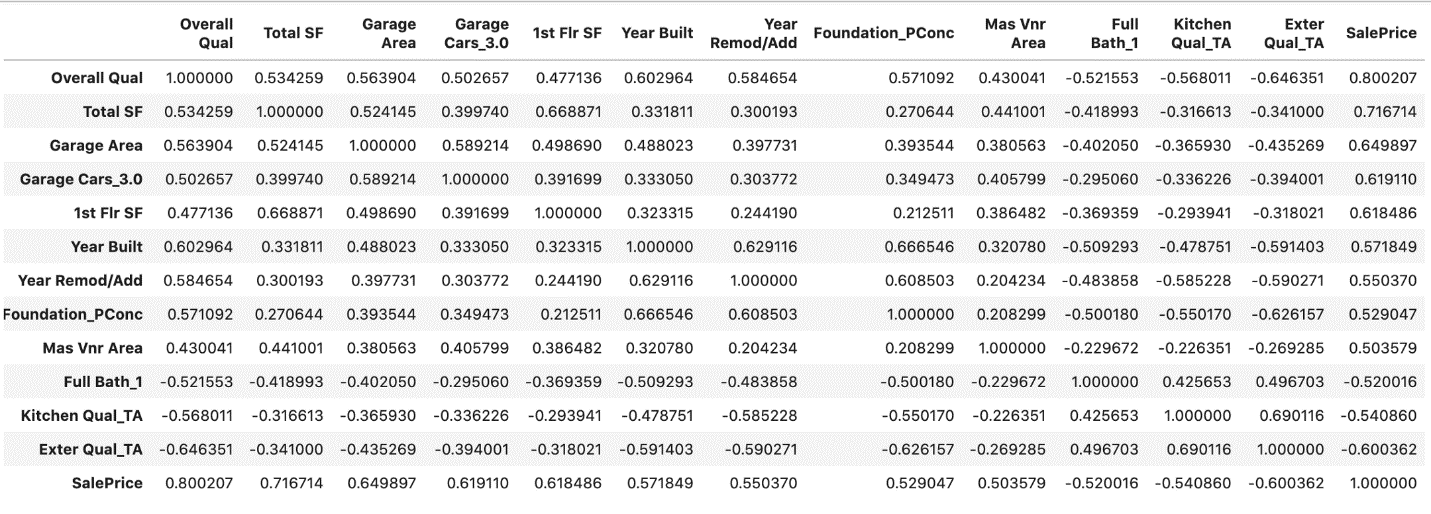
import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

Следующий код создает матрицу корреляции между всеми исследуемыми показателями и нашей переменной y (стоимость недвижимости).

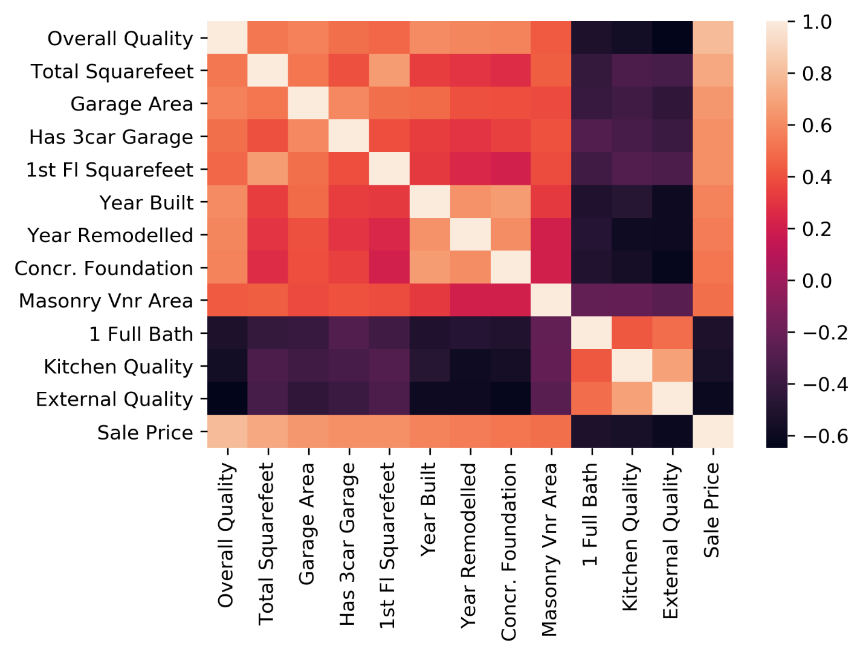
dataframe.corr()



Корреляционная матрица всего с 13 переменными. Нельзя сказать, что она совсем не читабельна. Однако почему бы не облегчить себе жизнь визуализацией?

**Простая тепловая карта в Seaborn**

sns.heatmap(dataframe.corr());



Сделаем базовую тепловую карту более полезной с минимальными усилиями.

Взгляните на список аргументов heatmap:

seaborn.heatmap(data, \*, vmin=None, vmax=None, cmap=None, center=None, robust=False, annot=None, fmt='.2g', annot\_kws=None, linewidths=0, linecolor='white', cbar=True, cbar\_kws=None, cbar\_ax=None, square=False, xticklabels='auto', yticklabels='auto', mask=None, ax=None, \*\*kwargs)

* vmin, vmax — устанавливают диапазон значений, которые служат основой для цветовой карты (colormap).
* cmap — определяет конкретную colormap, которую мы хотим использовать (ознакомьтесь с полным диапазоном цветовых палитр [**здесь**](https://python-graph-gallery.com/92-control-color-in-seaborn-heatmaps/)).
* center — принимает вещественное число для центрирования цветовой карты; если cmap не указан, используется colormap по умолчанию; если установлено значение True — все цвета заменяются на синий.
* annot — при значении True числовые значения корреляции отображаются внутри ячеек.
* cbar — если установлено значение False, цветовая шкала (служит легендой) исчезает.

# Увеличьте размер

heatmap plt.figure(figsize=(16, 6))

# Сохраните объект тепловой карты в переменной, чтобы легко получить к нему доступ,

# когда вы захотите включить дополнительные функции (например, отображение заголовка).

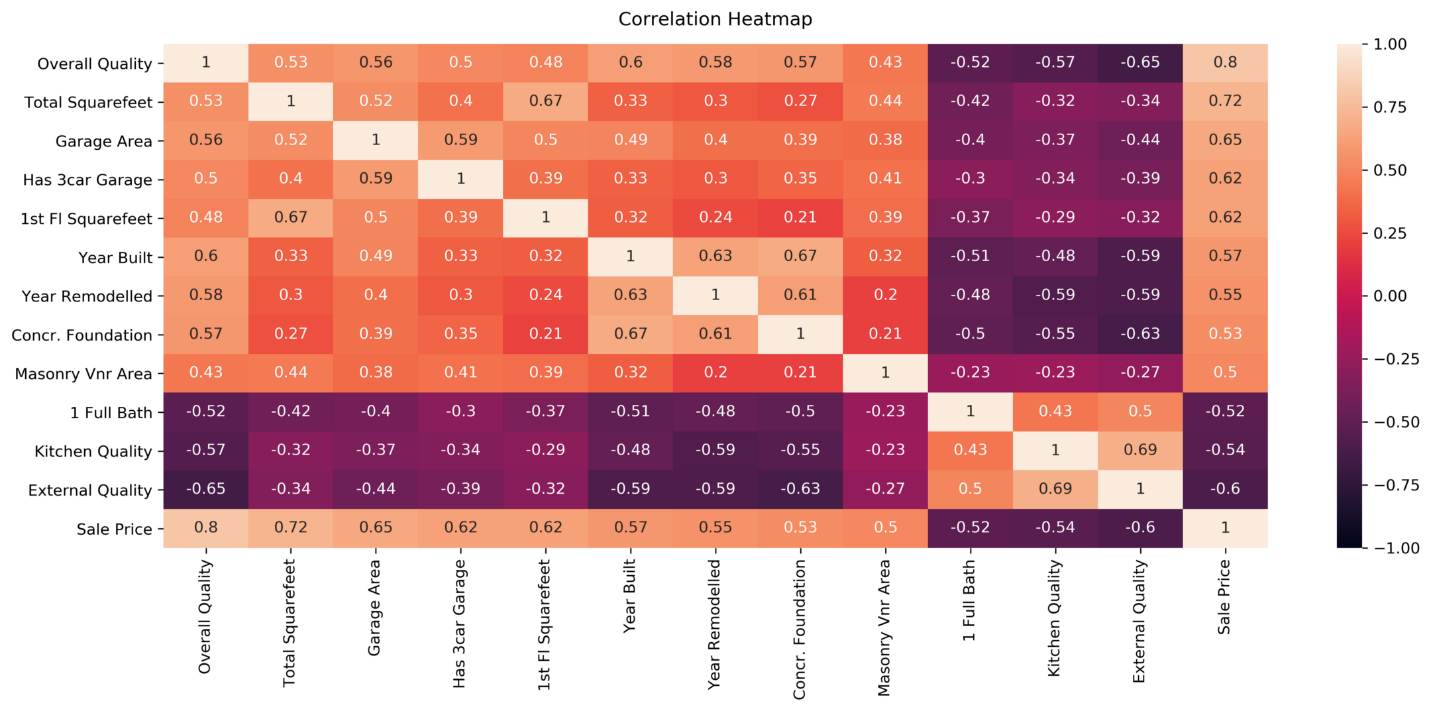
# Задайте диапазон значений для отображения на цветовой карте от -1 до 1 и установите для аннотации (annot) значение True,

# чтобы отобразить числовые значения корреляции на тепловой карте.

heatmap = sns.heatmap(dataframe.corr(), vmin=-1, vmax=1, annot=True)

# Дайте тепловой карте название. Параметр pad (padding) определяет расстояние заголовка от верхней части тепловой карты.

heatmap.set\_title('Correlation Heatmap', fontdict={'fontsize':12}, pad=12);



Для работы с heatmap лучше всего подходит расходящаяся цветовая палитра. Она имеет два очень разных темных (насыщенных) цвета на соответствующих концах диапазона интерполированных значений с бледной, почти бесцветной средней точкой. Проиллюстрируем это утверждение и разберемся с еще одной небольшой деталью: как сохранить созданную тепловую карту в файл png со всеми необходимыми x и y метками (xticklabels и yticklabels).

plt.figure(figsize=(16, 6))

heatmap = sns.heatmap(dataframe.corr(), vmin=-1, vmax=1, annot=True, cmap='BrBG')

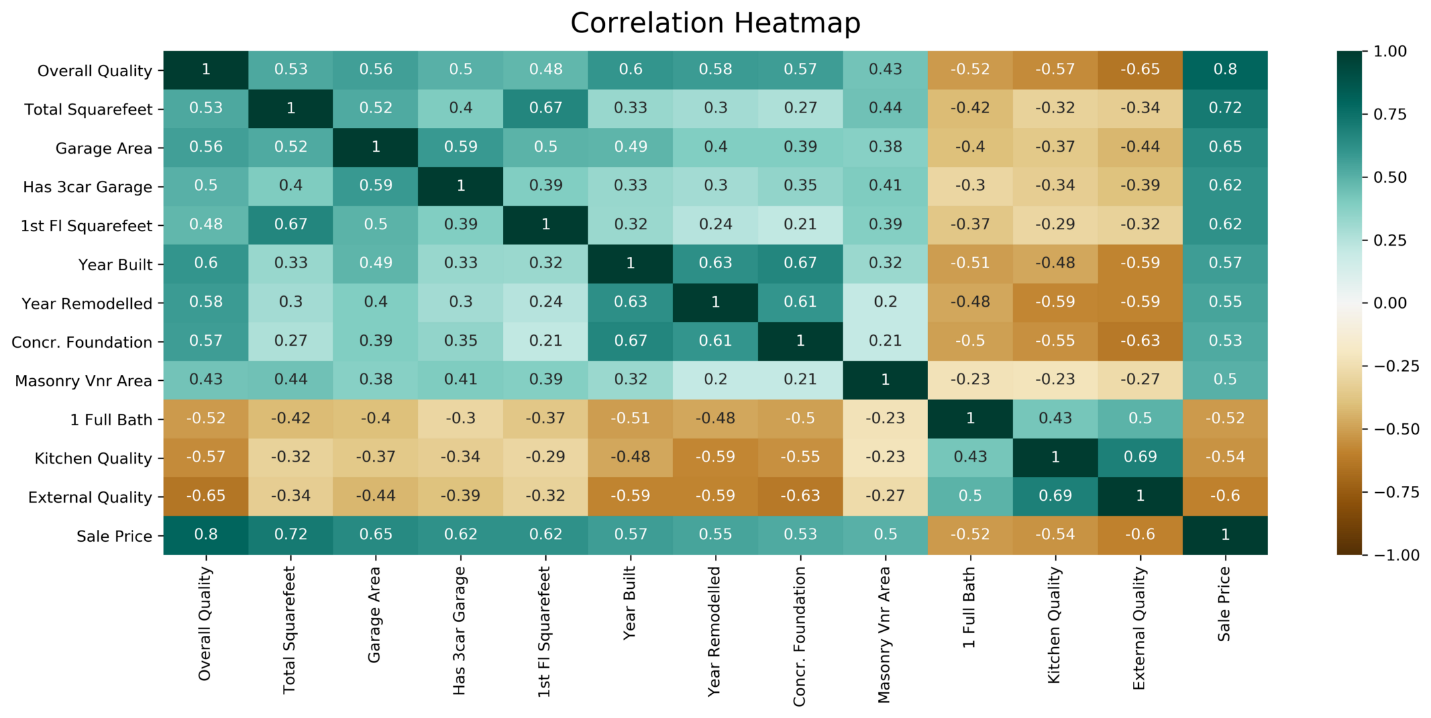
heatmap.set\_title('Correlation Heatmap', fontdict={'fontsize':18}, pad=12);

# Сохраните карту как png файл

# Параметр dpi устанавливает разрешение сохраняемого изображения в точках на дюйм

# bbox\_inches, когда установлен в значение 'tight', не позволяет обрезать лейблы

plt.savefig('heatmap.png', dpi=300, bbox\_inches='tight')

Более сильная корреляция на обоих концах спектра проявляется в виде темных (насыщенных) ячеек, слабая – в виде светлых ячеек.

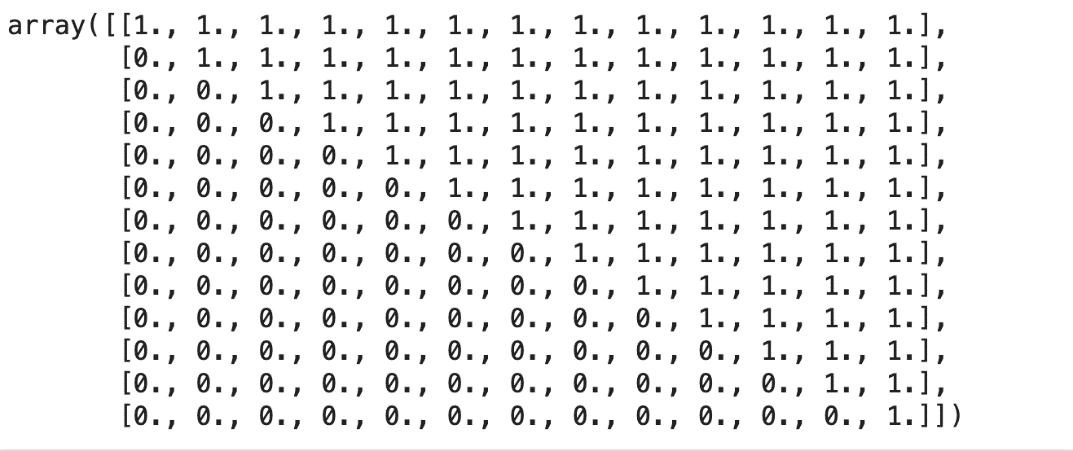
**Треугольная тепловая карта корреляции**

Взгляните на любую из приведенных выше тепловых карт. Если вы отбросите одну из ее половин по диагонали, обозначенной единицами, вы не потеряете никакой информации. Итак, давайте сократим тепловую карту, оставив только нижний треугольник.

Аргумент mask (маска) heatmap пригодится, чтобы скрыть часть тепловой карты. Маска — принимает в качестве аргумента массив логических значений или структуру табличных данных (dataframe). Если она предоставлена, ячейки тепловой карты, для которых значения маски является True, не отображаются.

Давайте воспользуемся функцией np.triu() библиотеки numpy, чтобы изолировать верхний треугольник матрицы, превращая все значения в нижнем треугольнике в 0. np.tril() будет делать то же самое, только для нижнего треугольника. В свою очередь функция np.ones\_like() изменит все изолированные значения на 1.

np.triu(np.ones\_like(dataframe.corr()))



Когда мы преобразуем тип данных к логическому, все 1 превращаются в True, а все 0 – в False.

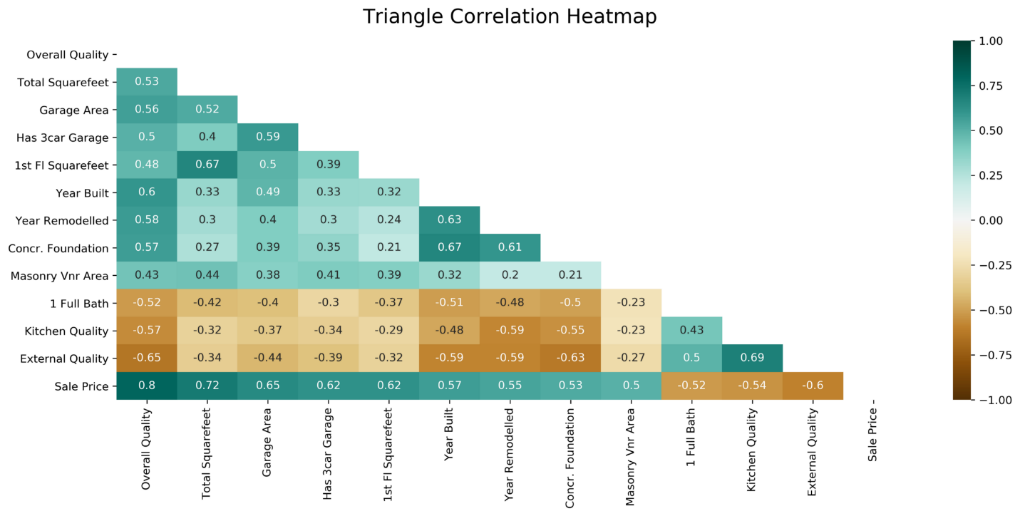
plt.figure(figsize=(16, 6))

# Определите маску, чтобы установить значения в верхнем треугольнике на True

mask = np.triu(np.ones\_like(dataframe.corr(), dtype=np.bool))

heatmap = sns.heatmap(dataframe.corr(), mask=mask, vmin=-1, vmax=1, annot=True, cmap='BrBG')

heatmap.set\_title('Triangle Correlation Heatmap', fontdict={'fontsize':18}, pad=16);



**Корреляция независимых переменных с зависимой**

Довольно часто мы хотим создать цветную карту, которая показывает выраженность связи между каждой независимой переменной, включенной в нашу модель, и зависимой переменной.

Следующий код возвращает корреляцию каждого параметра с «ценой продажи», единственной зависимой переменной в порядке убывания.

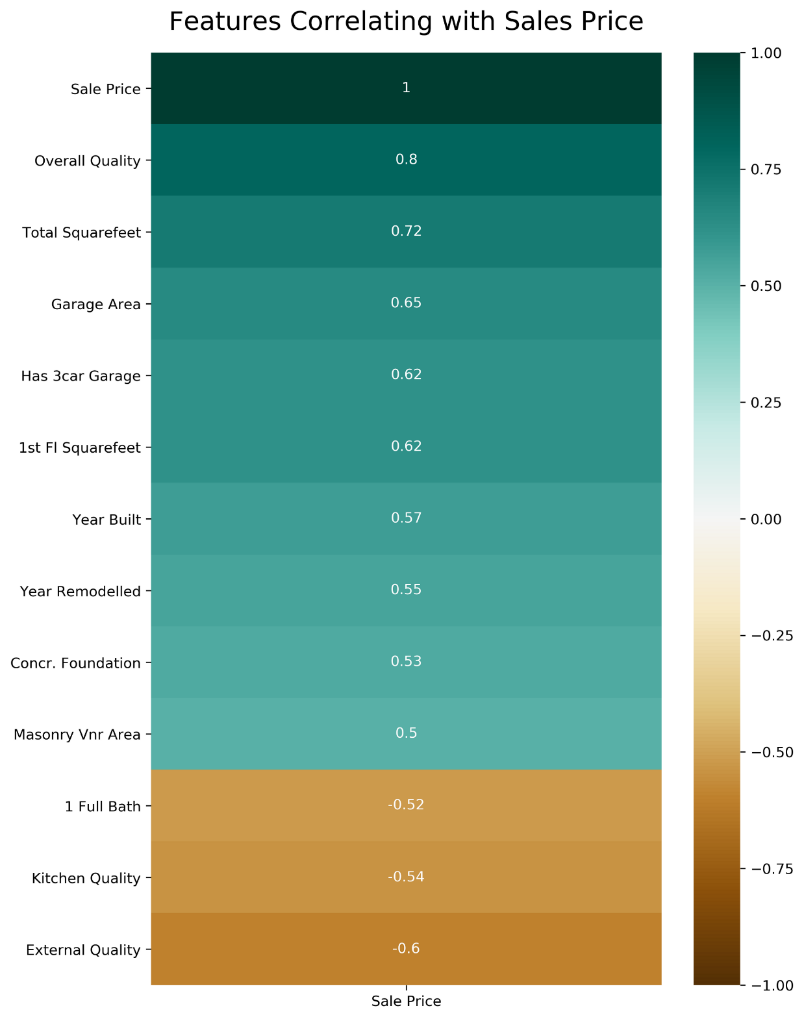
dataframe.corr()[['Sale Price']].sort\_values(by='Sale Price', ascending=False)

Давайте используем полученный список в качестве данных для отображения на тепловой карте.

plt.figure(figsize=(8, 12))

heatmap = sns.heatmap(dataframe.corr()[['Sale Price']].sort\_values(by='Sale Price', ascending=False), vmin=-1, vmax=1, annot=True, cmap='BrBG')

heatmap.set\_title('Features Correlating with Sales Price', fontdict={'fontsize':18}, pad=16);



Особенности Корреляция с ценой продажи

Эти примеры демонстрируют основную функциональность heatmap в Seaborn. Теперь перейдем к точечным диаграммам.

**Построение Scatter Plot в Seaborn**

Давайте рассмотрим процесс создания точечной диаграммы в Seaborn. Построим простые и трехмерные диаграммы рассеивания, а также групповые графики на базе FacetGrid.

**Импорт данных**

Мы будем использовать набор данных, [**основанный на мировом счастье**](https://www.kaggle.com/unsdsn/world-happiness). Сравнение его индекса с другими показателям отразит факторы, влияющие на уровень счастья в мире.

**Построение точечной диаграммы**

На графике отразим соотношение индекса счастья к экономике страны (ВВП на душу населения):

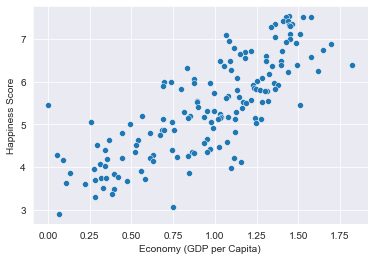
dataframe = pd.read\_csv('2016.csv')

sns.scatterplot(data=dataframe, x="Economy (GDP per Capita)", y="Happiness Score");

При помощи Seaborn очень легко составлять простые графики наподобие диаграмм рассеивания. Нам не обязательно использовать объект Figure и экземпляры Axes или что-нибудь настраивать. Здесь мы передали dataframe в качестве аргумента с данными, а признаки с информацией, которую нужно визуализировать, в x и y.

Оси диаграммы по умолчанию подписываются именами столбцов, которые соответствуют заголовкам из загружаемого файла. Ниже мы рассмотрим, как это изменить.

После выполнения кода мы получим следующее:



Результат показал прямую зависимость между ВВП на душу населения и предполагаемого уровня счастья жителей конкретной страны или региона.

**Построение группы графиков scatterplot при помощи FacetGrid**

Если требуется сравнить много переменных друг с другом, например, среднюю продолжительность жизни наряду с оценкой счастья и уровнем экономики, нет необходимости строить 3D-график.

Несмотря на существование двумерных диаграмм, позволяющих визуализировать соотношение между множествами переменных, не все из них просты в применении.

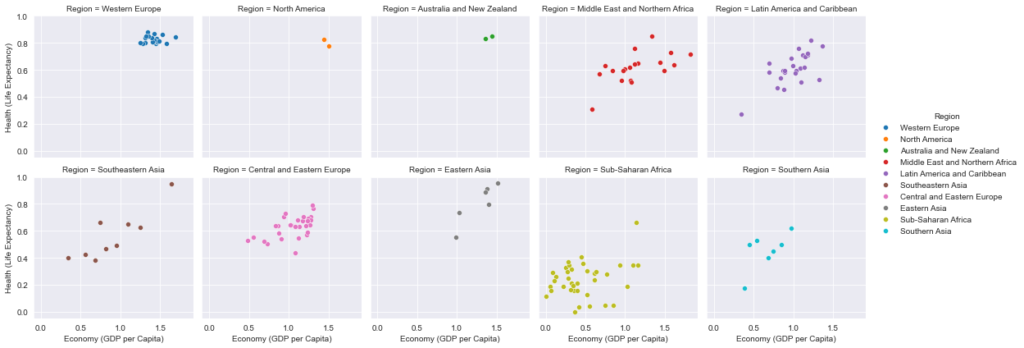
При помощи объекта FacetGrid, библиотека Seaborn позволяет обрабатывать данные и строить на их основе групповые взаимосвязанные графики.

Взглянем на следующий пример:

grid = sns.FacetGrid(dataframe, col="Region", hue="Region", col\_wrap=5)

grid.map(sns.scatterplot, "Economy (GDP per Capita)", "Health (Life Expectancy)")

grid.add\_legend();



В этом примере мы создали экземпляр объекта FacetGrid с параметром dataframe в качестве данных. При передаче значения "Region" аргументу col библиотека сгруппирует датасет по регионам и построит диаграмму рассеивания для каждого из них.

Параметр hue задает каждому региону собственный оттенок. Наконец, при помощи аргумента col\_wrap ширина области Figure ограничивается до 5-ти диаграмм. По достижении этого предела следующие графики будут построены на новой строке.

Для подготовки сетки перед выводом на экран мы используем метод map(). Тип диаграммы передается в первом аргументе со значением sns.scatterplot, а в качестве осей служат переменные x и y.

В результате будет сформировано 10 графиков по каждому региону с соответствующими им осями. Непосредственно перед печатью мы вызываем метод, добавляющий легенду с обозначением цветовой маркировки.

**Построение 3D-диаграммы рассеивания**

К сожалению, в Seaborn отсутствует собственный 3D-движок. Являясь лишь дополнением к Matplotlib, он опирается на графические возможности основной библиотеки. Тем не менее, мы все еще можем применить стиль Seaborn к трехмерной диаграмме.

Посмотрим, как она будет выглядеть с выборкой по уровням счастья, экономики и здоровья:

%matplotlib notebook

import matplotlib.pyplot as plt

import pandas as pd

import seaborn as sns

from mpl\_toolkits.mplot3d import Axes3D

df = pd.read\_csv('Downloads/2016.csv')

fig = plt.figure()

ax = fig.add\_subplot(111, projection = '3d')

x = df['Happiness Score']

y = df['Economy (GDP per Capita)']

z = df['Health (Life Expectancy)']

ax.set\_xlabel("Счастье")

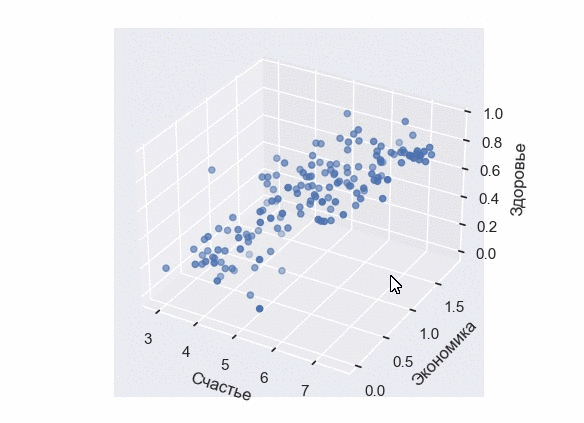
ax.set\_ylabel("Экономика")

ax.set\_zlabel("Здоровье")

ax.scatter(x, y, z)

plt.show()

В результате выполнения кода появится интерактивная 3D-визуализация, которую можно вращать и масштабировать в трехмерном пространстве:



**Настройка Scatter Plot**

При помощи Seaborn можно легко настраивать различные элементы создаваемых диаграмм. Например, присутствует возможность изменения цвета и размера каждой точки на графике.

Попробуем задать некоторые параметры и посмотреть, как изменится его внешний вид:

sns.scatterplot(

data=dataframe,

x="Economy (GDP per Capita)",

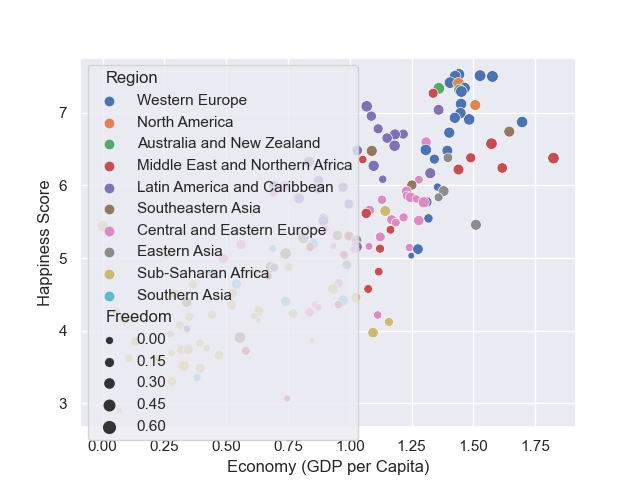
y="Happiness Score",

hue="Region",

size="Freedom"

);

Здесь мы применили оттенок к регионам — это означает, что данные по каждому из них будут раскрашены по-разному. Кроме того, при помощи аргумента size были заданы пропорции точек в зависимости от уровня свободы. Чем больше его значение, тем крупнее точка на диаграмме:



Или можно просто задать одинаковый цвет и размер для всех точек:

sns.scatterplot(

data=dataframe,

x="Economy (GDP per Capita)",

y="Happiness Score",

color="red",

sizes=5

);

Отлично, вы узнали несколько способов построения scatter plot в Seaborn. Перейдем к еще одному популярному графику.

**Построение Box Plot в Seaborn**

Box Plot, называемые также:

* графиками прямоугольников,
* коробчатыми графиками,
* графиками размаха
* или ящиками с усами за свой вид.

Они используются для визуализации сводной статистики датасета. Box Plot отображают атрибуты распределения, такие как диапазон и распределение данных в диапазоне (прямоугольника, «усы», медиана).

**Импорт данных**

Для создания box plot нужны непрерывные числовые данные, поскольку такая диаграмма отображает сводную статистику — медиану, диапазон и выбросы. Для примера воспользуемся набором данных [**forestfires.csv**](https://archive.ics.uci.edu/ml/machine-learning-databases/forest-fires/forestfires.csv) (сведения об индексе влажности лесной подстилки, осадках, температуре, ветре и т.д.).

Импортируем pandas для загрузки и анализа датасета, seaborn и модуль pyplot из matplotlib для визуализации:

import pandas as pd

from matplotlib import pyplot as plt

import seaborn as sns

Воспользуемся pandas для чтения CSV-файла в dataframe и выведем первые 5 строк. Кроме того, проверим, содержит ли набор данных пропущенные значения (Null, NaN):

# укажите свой путь к файлу forestfires

dataframe = pd.read\_csv("Downloads/forestfires.csv")

print(dataframe.isnull().values.any())

dataframe.head()

Код вернет False и верхнюю часть таблицы.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **X** | **Y** | **month** | **day** | **FFMC** | **DMC** | **DC** | **ISI** | **temp** | **RH** | **wind** | **rain** | **area** |
| **0** | 7 | 5 | mar | fri | 86.2 | 26.2 | 94.3 | 5.1 | 8.2 | 51 | 6.7 | 0.0 | 0.0 |
| **1** | 7 | 4 | oct | tue | 90.6 | 35.4 | 669.1 | 6.7 | 18.0 | 33 | 0.9 | 0.0 | 0.0 |
| **2** | 7 | 4 | oct | sat | 90.6 | 43.7 | 686.9 | 6.7 | 14.6 | 33 | 1.3 | 0.0 | 0.0 |
| **3** | 8 | 6 | mar | fri | 91.7 | 33.3 | 77.5 | 9.0 | 8.3 | 97 | 4.0 | 0.2 | 0.0 |
| **4** | 8 | 6 | mar | sun | 89.3 | 51.3 | 102.2 | 9.6 | 11.4 | 99 | 1.8 | 0.0 | 0.0 |

Print вывел False, значит – никаких пропущенных значений нет. Если бы они были, то пришлось бы дополнительно обрабатывать отсутствующие значения.

После проверки данных нужно выбрать признаки, которые будем визуализировать. Для удобства сохраним их в переменные с такими же названиями.

FFMC = dataframe["FFMC"]

DMC = dataframe["DMC"]

DC = dataframe["DC"]

RH = dataframe["RH"]

ISI = dataframe["ISI"]

temp = dataframe["temp"]

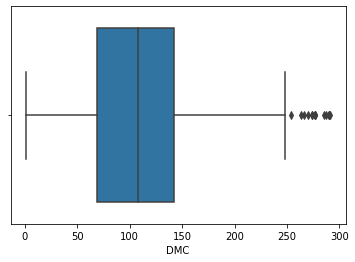
Это те колонки, которые содержат непрерывные числовые данные.

**Построение box plot**

Для создания диаграммы воспользуемся функцией boxplot в Seaborn, которой в качестве аргументов передадим переменные для визуализации:

sns.boxplot(x=DMC);

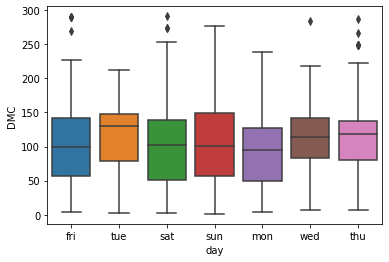
Для визуализации распределения только одного признака мы передаем его в переменную x. В этом случае Seaborn автоматически вычислит значения по оси y, что видно на следующем изображении.



Если требуется определенное распределение, сегментированное по типу, то можно для функции boxplot в качестве аргументов передать категориальную переменную в x и непрерывную переменную в y.

sns.boxplot(x=dataframe["day"], y=DMC);

Теперь получилась блочная диаграмма, созданная для каждого дня недели.

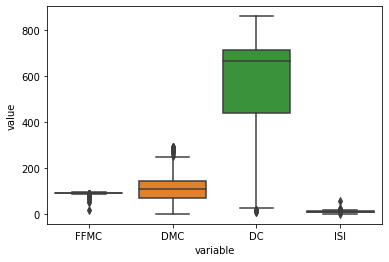


Если требуется визуализировать несколько столбцов одновременно, то аргументов x и y будет недостаточно. Для этих целей используется аргумент data, которому передается набор данных, содержащий требуемые переменные и их значения.

Создадим новый датасет, содержащий только те данные, которые мы хотим визуализировать. Затем к нему применим функцию melt(). Полученный в результате набор данных передается аргументу data. В аргументы x и y в этом случае передаются значения по умолчанию из melt (value и variable):

df = pd.DataFrame(data=dataframe, columns=["FFMC", "DMC", "DC", "ISI"])

sns.boxplot(x="variable", y="value", data=pd.melt(df));



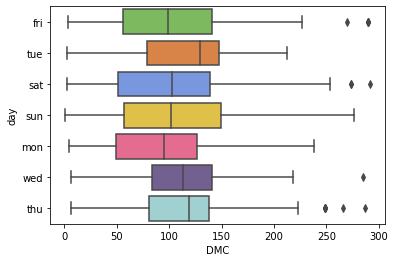
**Изменение цвета boxplot**

Seaborn автоматически назначает различные цвета различным переменным, чтобы можно было их легко визуально различить. Цвет диаграмм можно изменить, предоставив свой список цветов.

После определения списка цветов в виде HEX-значений или названий доcтупного цвета Matplotlib, можно передать их функции boxplot() в качестве аргумента palette:

colors = ['#78C850', '#F08030', '#6890F0','#F8D030', '#F85888', '#705898', '#98D8D8']

sns.boxplot(x=DMC, y=dataframe["day"], palette=colors);



**Настройка подписи осей**

С помощью Seaborn можно легко настроить подписи по осям X и Y. Например, изменить размер шрифта, подписи или повернуть их, чтобы сделать более удобными для чтения.

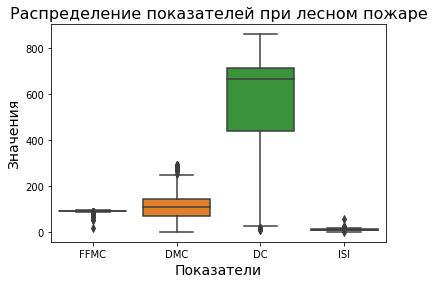
df = pd.DataFrame(data=dataframe, columns=["FFMC", "DMC", "DC", "ISI"])

boxplot = sns.boxplot(x="variable", y="value", data=pd.melt(df))

boxplot.axes.set\_title("Распределение показателей при лесном пожаре", fontsize=16)

boxplot.set\_xlabel("Показатели", fontsize=14)

boxplot.set\_ylabel("Значения", fontsize=14);



**Изменение порядка отображения блоков**

Для отображения блочных диаграмм в определенном порядке используется аргумент order, которому передается список имен столбцов в том порядке, в котором их нужно расположить:

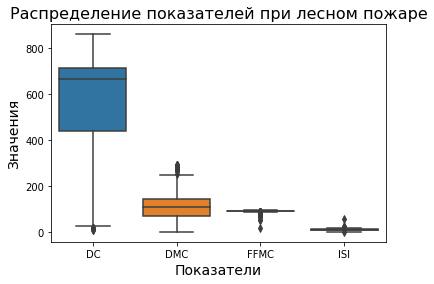
df = pd.DataFrame(data=dataframe, columns=["FFMC", "DMC", "DC", "ISI"])

boxplot = sns.boxplot(x="variable", y="value", data=pd.melt(df), order=["DC", "DMC", "FFMC", "ISI"])

boxplot.axes.set\_title("Распределение показателей при лесном пожаре", fontsize=16)

boxplot.set\_xlabel("Показатели", fontsize=14)

boxplot.set\_ylabel("Значения", fontsize=14);



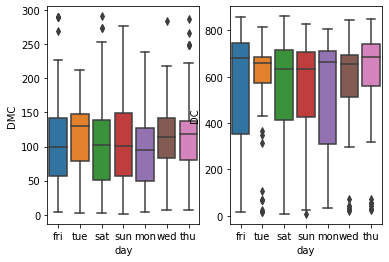
**Создание subplots с помощью Matplotlib**

Если необходимо разделить общий box plot на несколько для отдельных признаков, то это можно сделать. Определите область отрисовки (fig) и нужное количество координатных осей (axes) с помощью функции subplots из Matplotlib. Доступ к нужной области объекта axes можно получить через его индекс. Функция boxplot() принимает ax аргумент, который по индексу объекта axes получает область для построения диаграммы:

fig, axes = plt.subplots(1, 2)

sns.boxplot(x=day, y=DMC, orient='v', ax=axes[0])

sns.boxplot(x=day, y=DC, orient='v', ax=axes[1]);



**Box Plot с диаграммой рассеивания**

Для более наглядного восприятия распределения можно наложить точечную диаграмму рассеивания на блочную.

С этой целью последовательно создаем две диаграммы. Диаграмма, созданная функцией stripplot(), будет наложена поверх box plot, так как они выводятся в одной и той же области:

df = pd.DataFrame(data=dataframe, columns=["DC", "DMC"])

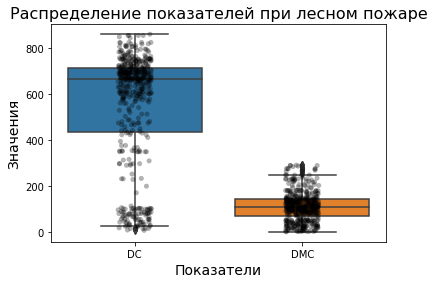
boxplot = sns.boxplot(x="variable", y="value", data=pd.melt(df), order=["DC", "DMC"])

boxplot = sns.stripplot(x="variable", y="value", data=pd.melt(df), marker="o", alpha=0.3, color="black", order=["DC", "DMC"])

boxplot.axes.set\_title("Распределение показателей при лесном пожаре", fontsize=16)

boxplot.set\_xlabel("Показатели", fontsize=14)

boxplot.set\_ylabel("Значения", fontsize=14);



Мы рассмотрели несколько способов построения Box Plot с помощью Seaborn и Python. Также узнали, как настроить цвета, подписи осей, порядок следования диаграмм, наложение точечных диаграмм и разделение диаграмм для отдельных величин.

Последний тип графика, о котором стоит упомянуть — Violin Plot.

**Построение Violin Plot в Seaborn**

Violin Plot или скрипичные диаграммы используются для визуализации распределения данных, отображая диапазон данных, медиану и область распределения данных.

Такие диаграммы, как и ящики с усами, показывают сводную статистику. Дополнительно они включают в себя графики плотности распределения, которые и определяют форму/распределение данных при визуализации.

**Импорт данных**

Для примера воспользуемся набором данных [**Gapminder**](https://www.kaggle.com/tklimonova/gapminder-datacamp-2007?select=gapminder_full.csv), содержащем информацию о численности населения, продолжительности жизни и другие данные по странам и годам, начиная с 1952 года.

Импортируем pandas, seaborn и модуль pyplot из matplotlib:

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

Далее загрузим датасет и посмотрим из чего он состоит.

dataframe = pd.read\_csv(

"Downloads/gapminder\_full.csv",

error\_bad\_lines=False,

encoding="ISO-8859-1"

)

dataframe.head()

В результате получим:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **country** | **year** | **population** | **continent** | **life\_exp** | **gdp\_cap** |
| **0** | Afghanistan | 1952 | 8425333 | Asia | 28.801 | 779.445314 |
| **1** | Afghanistan | 1957 | 9240934 | Asia | 30.332 | 820.853030 |
| **2** | Afghanistan | 1962 | 10267083 | Asia | 31.997 | 853.100710 |
| **3** | Afghanistan | 1967 | 11537966 | Asia | 34.020 | 836.197138 |
| **4** | Afghanistan | 1972 | 13079460 | Asia | 36.088 | 739.981106 |

Определим признаки, которые будем визуализировать. Для удобства сохраним их в переменные с такими же названиями.

country = dataframe.country

continent = dataframe.continent

population = dataframe.population

life\_exp = dataframe.life\_exp

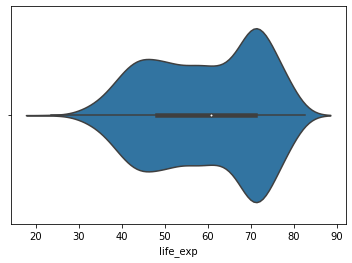
gdp\_cap = dataframe.gdp\_cap

**Построение простой скрипичной диаграммы**

Теперь, после того как мы загрузили данные и выбрали величины, которые хотим визуализировать, можно создать скрипичную диаграмму. Используем функцию violinplot(), которой в качестве аргумента x передадим переменную для визуализации.

Значения по оси Y будут высчитаны автоматически.

sns.violinplot(x=life\_exp);



При этом, можно было не выбирать предварительно данные по имени столбца и не сохранять в переменной life\_exp. Используя аргумент data, которому передан наш набор данных, и аргумент x, которому присвоено имя переменной «life\_exp», получим точно такой же результат.

sns.violinplot(x="life\_exp", data=dataframe);

На этом изображении Seaborn строит график распределения ожидаемой продолжительности жизни сразу по всем странам, так как использовалась только одна переменная life\_exp. В большинстве случаев такого типа переменная рассматривается на основе других переменных, таких как country или continent в нашем случае.

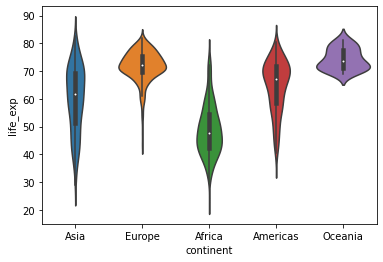
**Построение Violin Plot с осями X и Y**

Для того чтобы получить визуализацию распределения данных, сегментированное по типу, необходимо в качестве аргументов функции использовать категориальную переменную для x и непрерывную для y.

В этом наборе данных много стран. Если построить диаграммы для всех стран, то их будет слишком много, чтобы их можно было рассмотреть. Можно, конечно, выделить подмножество из набора данных и просто построить диаграммы, скажем, для 10 стран.

Вместо этого, построим violinplot для континентов.

sns.violinplot(x=continent, y=life\_exp, data=dataframe);



**Изменение подписи осей заголовка диаграммы**

Предположим, что необходимо изменить некоторые заголовки и подписи нашего графика, чтобы было проще его анализировать.

Несмотря на то, что Seaborn автоматически подписывает оси X и Y, можно изменить подписи с помощью функций set\_title() и set\_label() после создания объекта axes. Надо просто передать название, которое хотим дать нашему графику, функции set\_title().

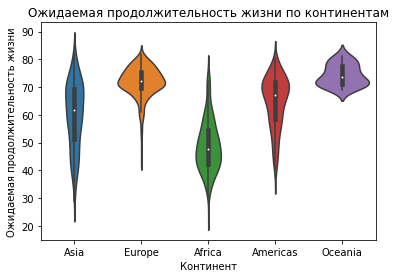
Для того чтобы подписать оси, используется функция set() с аргументами xlabel и ylabel или функции-обертки set\_xlabel()/set\_ylabel():

ax = sns.violinplot(x=continent, y=life\_exp)

ax.set\_title("Ожидаемая продолжительность жизни по континентам")

ax.set\_ylabel("Ожидаемая продолжительность жизни")

ax.set\_xlabel("Континент");



**Изменение цвета violinplot**

Для изменения цвета диаграмм можно создать список заранее выбранных цветов и передать этот список параметром pallete функции violinplot():

colors\_list = [

'#78C850', '#F08030', '#6890F0',

'#A8B820', '#F8D030', '#E0C068',

'#C03028', '#F85888', '#98D8D8'

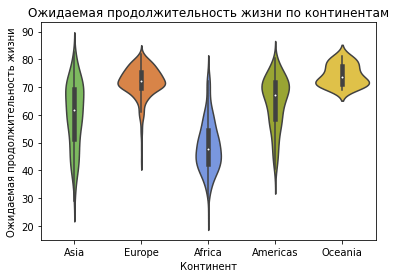
]

ax = sns.violinplot(x=continent, y=life\_exp, palette=colors\_list)

ax.set\_title("Ожидаемая продолжительность жизни по континентам")

ax.set\_ylabel("Ожидаемая продолжительность жизни")

ax.set\_xlabel("Континент");



**Violin Plot с диаграммой рассеивания**

Точечную диаграмму распределения можно наложить на скрипичную диаграмму, чтобы увидеть размещение точек, составляющих это распределение. Для этого просто создается одна область рисования, а затем последовательно в ней создаются две диаграммы.

colors\_list = [

'#78C850', '#F08030', '#6890F0',

'#A8B820', '#F8D030', '#E0C068',

'#C03028', '#F85888', '#98D8D8'

]

plt.figure(figsize=(16,8))

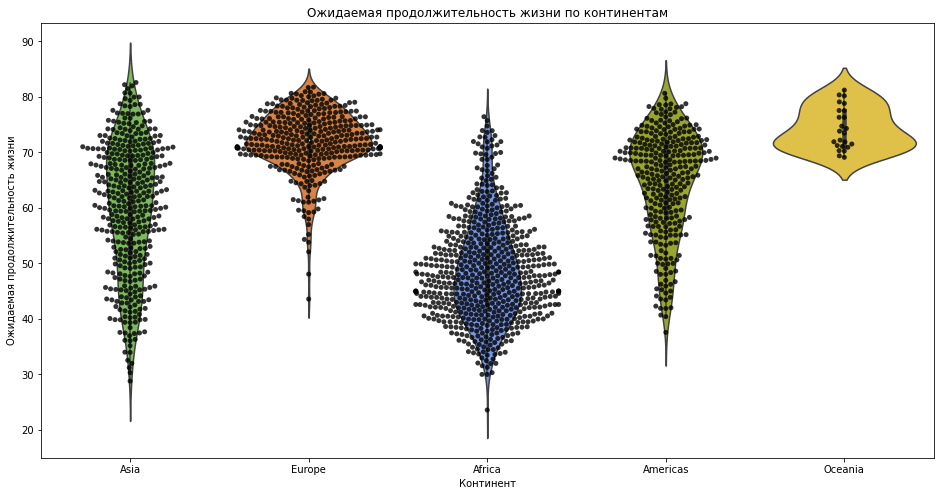
sns.violinplot(x=continent, y=life\_exp,palette=colors\_list)

sns.swarmplot(x=continent, y=life\_exp, color="k", alpha=0.8)

plt.title("Ожидаемая продолжительность жизни по континентам")

plt.ylabel("Ожидаемая продолжительность жизни")

plt.xlabel("Континент");



**Изменение стиля скрипичной диаграммы**

Можно легко изменить стиль и цвет нашей диаграммы, используя функции set\_style() и set\_palette() соответственно.

Seaborn поддерживает несколько различных вариантов изменения [**стиля**](https://seaborn.pydata.org/tutorial/aesthetics.html) и [**цветовой палитры**](https://seaborn.pydata.org/tutorial/color_palettes.html) графиков:

plt.figure(figsize=(16,8))

sns.set\_palette("RdBu")

sns.set\_style("darkgrid")

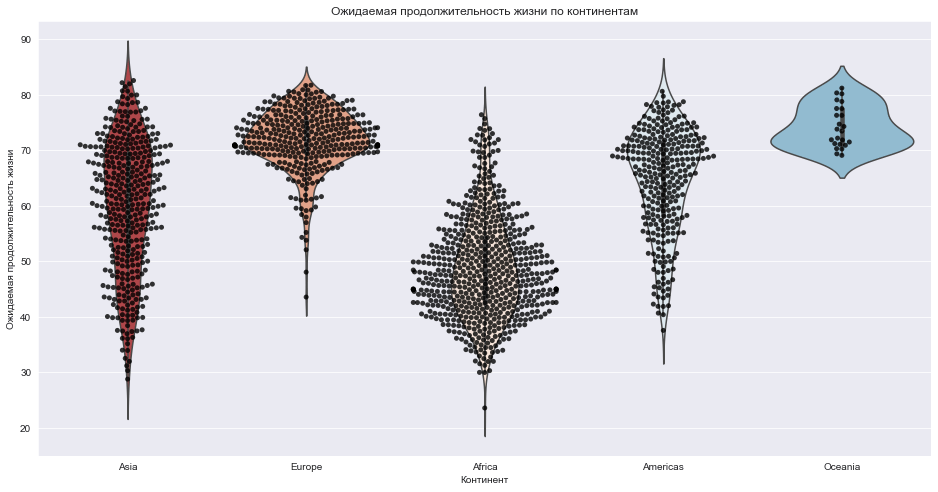
sns.violinplot(x=continent, y=life\_exp)

sns.swarmplot(x=continent, y=life\_exp, color="k", alpha=0.8)

plt.title("Ожидаемая продолжительность жизни по континентам")

plt.ylabel("Ожидаемая продолжительность жизни")

plt.xlabel("Континент");



**Построение Violin Plot для разных признаков**

Если нужно разделить визуализацию столбцов из набора данных на их собственные диаграммы, то это можно сделать. Создайте область рисования и сетку, в ячейках которой будут графики.

Для отображения диаграммы в соответствующей ячейке применяется функция add\_subplot(), которой передается адрес этой ячейки. Создание диаграммы делаем, как обычно, используя объект axes.

Можно использовать y=variable, либо data=variable.

fig = plt.figure(figsize=(6, 4))

gs = fig.add\_gridspec(1, 3)

ax = fig.add\_subplot(gs[0, 0])

sns.violinplot(data=population)

ax.set\_xlabel("Население")

ax = fig.add\_subplot(gs[0, 1])

sns.violinplot(data=life\_exp)

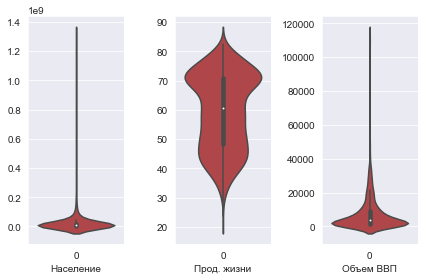
ax.set\_xlabel("Прод. жизни")

ax = fig.add\_subplot(gs[0, 2])

sns.violinplot(data=gdp\_cap)

ax.set\_xlabel("Объем ВВП")

fig.tight\_layout()



**Группировка скрипичных диаграмм по категориальному признаку**

По настоящему полезная вещь для violinplot — это группировка по значениям категориальной переменной. Например, если есть категориальная величина, имеющая два значения (обычно, True/False), то в этом случае можно группировать графики по этим значениям.

Допустим, есть набор данных по трудоустройству населения со столбцом employment и его значениями employed и unemployed. Тогда можно сгруппировать диаграммы по видам занятости.

Поскольку в наборе данных Gapminder нет столбца, подходящего для такой группировки, его можно сделать, рассчитав среднюю продолжительность жизни для определенного подмножества стран, например, европейских стран.

Назначим Yes/No значение новому столбцу above\_average\_life\_exp для каждой страны. Если средняя продолжительность жизни выше, чем в среднем по датасету, то это значение равно Yes, и наоборот:

# Отделяем европейские страны от исходного датасет

europe = dataframe.loc[dataframe["continent"] == "Europe"]

# Вычисляем среднее значение переменной "life\_exp"

avg\_life\_exp = dataframe["life\_exp"].mean()

# Добавим новую колонку

europe.loc[:, "above\_average\_life\_exp"] = europe["life\_exp"] > avg\_life\_exp

europe["above\_average\_life\_exp"].replace(

{True: "Yes", False: "No"},

inplace=True

)

Теперь, если вывести наш набор данных, то получим следующее:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | **country** | **year** | **population** | **continent** | **life\_exp** | **gdp\_cap** | **above\_average\_life\_exp** |
| **12** | Albania | 1952 | 1282697 | Europe | 55.23 | 1601.056136 | No |
| **13** | Albania | 1957 | 1476505 | Europe | 59.28 | 1942.284244 | No |
| **14** | Albania | 1962 | 1728137 | Europe | 64.82 | 2312.888958 | Yes |
| **15** | Albania | 1967 | 1984060 | Europe | 66.22 | 2760.196931 | Yes |
| **16** | Albania | 1972 | 2263554 | Europe | 67.69 | 3313.422188 | Yes |

Теперь можно построить скрипичные диаграммы, сгруппированные по новому столбцу, который мы вставили. Учитывая, что европейских стран много, для удобства визуализации выберем последние 50 строк, используя europe.tail():

europe = europe.tail(50)

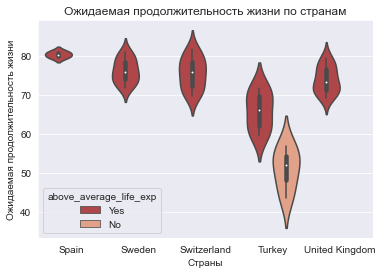
ax = sns.violinplot(x=europe.country, y=europe.life\_exp, hue=europe.above\_average\_life\_exp)

ax.set\_title("Ожидаемая продолжительность жизни по странам")

ax.set\_ylabel("Ожидаемая продолжительность жизни")

ax.set\_xlabel("Страны");

В результате получим:



Теперь страны с продолжительностью жизни меньше средней, ожидаемой отличаются по цвету.

**Разделение скрипичных диаграмм по категориальному признаку**

Если используется аргумент hue для категориальной переменной, имеющей два значения, то применив в функции violinplot() аргумент split и установив его в True, можно разделить скрипичные диаграммы пополам с учетом значения hue.

В нашем случае одна сторона скрипки (левая) будет представлять записи с ожидаемой продолжительностью жизни выше среднего, в то время как правая сторона будет использоваться для построения ожидаемой продолжительности жизни ниже среднего:

europe = europe.tail(50)

ax = sns.violinplot(

x=europe.country,

y=europe.life\_exp,

hue=europe.above\_average\_life\_exp,

split=True

)

ax.set\_title("Ожидаемая продолжительность жизни по странам")

ax.set\_ylabel("Ожидаемая продолжительность жизни")

ax.set\_xlabel("Страны");

