**Тема занятия №31: Работа с базой данных. Big data**

**1. Библиотека pandas**

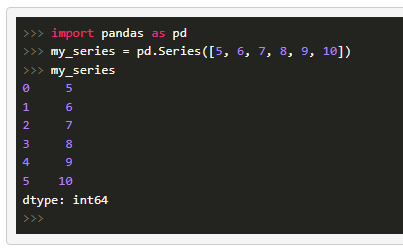
pandas — это высокоуровневая Python библиотека для анализа данных. Почему я её называю высокоуровневой, потому что построена она поверх более низкоуровневой библиотеки NumPy (написана на Си), что является большим плюсом в производительности. В экосистеме Python, pandas является наиболее продвинутой и быстроразвивающейся библиотекой для обработки и анализа данных. В своей работе мне приходится пользоваться ею практически каждый день, поэтому я пишу эту краткую заметку для того, чтобы в будущем ссылаться к ней, если вдруг что-то забуду. Также надеюсь, что читателям блога заметка поможет в решении их собственных задач с помощью pandas, и послужит небольшим введением в возможности этой библиотеки.

DataFrame и Series

Чтобы эффективно работать с pandas, необходимо освоить самые главные структуры данных библиотеки: DataFrame и Series. Без понимания что они из себя представляют, невозможно в дальнейшем проводить качественный анализ.

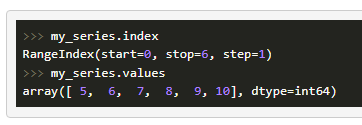
Series

Структура/объект Series представляет из себя объект, похожий на одномерный массив (питоновский список, например), но отличительной его чертой является наличие ассоциированных меток, т.н. индексов, вдоль каждого элемента из списка. Такая особенность превращает его в ассоциативный массив или словарь в Python.

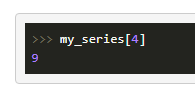


В строковом представлении объекта Series, индекс находится слева, а сам элемент справа. Если индекс явно не задан, то pandas автоматически создаёт RangeIndex от 0 до N-1, где N общее количество элементов. Также стоит обратить, что у Series есть тип хранимых элементов, в нашем случае это int64, т.к. мы передали целочисленные значения.

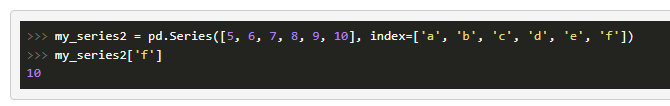
У объекта Series есть атрибуты, через которые можно получить список элементов и индексы, это values и index соответственно.



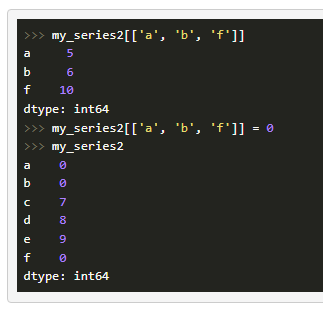
Доступ к элементам объекта Series возможны по их индексу (вспоминается аналогия со словарем и доступом по ключу).



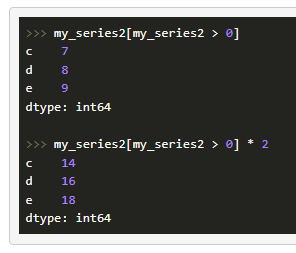
Индексы можно задавать явно:



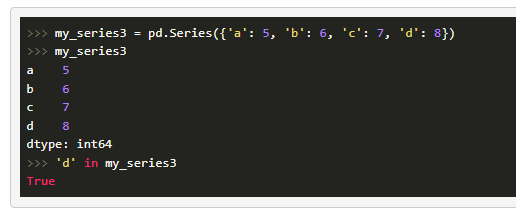
Делать выборку по нескольким индексам и осуществлять групповое присваивание:



Фильтровать Series как душе заблагорассудится, а также применять математические операции и многое другое:



Если Series напоминает нам словарь, где ключом является индекс, а значением сам элемент, то можно сделать так:



DataFrame

Объект DataFrame лучше всего представлять себе в виде обычной таблицы и это правильно, ведь DataFrame является табличной структурой данных. В любой таблице всегда присутствуют строки и столбцы. Столбцами в объекте DataFrame выступают объекты Series, строки которых являются их непосредственными элементами.

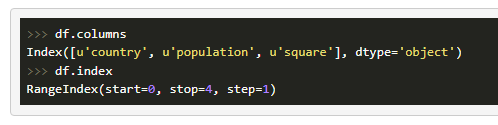
DataFrame проще всего сконструировать на примере питоновского словаря:



Чтобы убедиться, что столбец в DataFrame это Series, извлекаем любой:



Объект DataFrame имеет 2 индекса: по строкам и по столбцам. Если индекс по строкам явно не задан (например, колонка по которой нужно их строить), то pandas задаёт целочисленный индекс RangeIndex от 0 до N-1, где N это количество строк в таблице.



В таблице у нас 4 элемента от 0 до 3.

Доступ по индексу в DataFrame

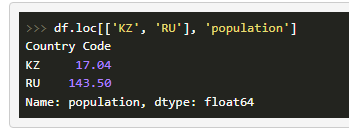
Индекс по строкам можно задать разными способами, например, при формировании самого объекта DataFrame или "на лету":



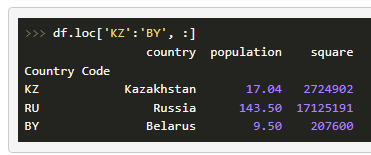
Как видно, индексу было задано имя - Country Code. Отмечу, что объекты Series из DataFrame будут иметь те же индексы, что и объект DataFrame:



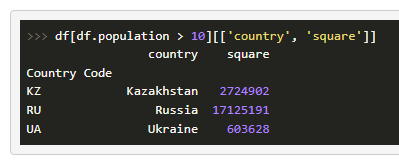
Можно делать выборку по индексу и интересующим колонкам:



Как можно заметить, .loc в квадратных скобках принимает 2 аргумента: интересующий индекс, в том числе поддерживается слайсинг и колонки.



Фильтровать DataFrame с помощью т.н. булевых массивов:



**2. Библиотека numpy**

NumPy — это библиотека Python, которую применяют для математических вычислений: начиная с базовых функций и заканчивая линейной алгеброй. Полное название библиотеки — Numerical Python extensions, или «Числовые расширения Python».

У этой библиотеки есть несколько важных особенностей, которые сделали ее популярным инструментом. Во-первых, исходный ее код в свободном доступе хранится на GitHub, поэтому NumPy называют open-source модулем для Python.

Во-вторых, библиотека написана на языках C и Fortran. Это компилируемые языки (языки программирования, текст которых преобразуется в машинный код — набор инструкций для конкретного типа процессора. Преобразование происходит с помощью специальной программы-компилятора, благодаря нему вычисления на компилируемых языках происходят быстрее), на которых вычисления производятся гораздо быстрее и эффективнее, чем на интерпретируемых языках (языки программирования, которые не заточены под конкретный тип процессора и могут быть запущены на разных типах устройств). К этим языкам относится и сам Python.

Где используется NumPy?

Научные вычисления. NumPy пользуются ученые для решения многомерных задач в математике и физике, биоинформатике, вычислительной химии и даже когнитивной психологии.

Создание новых массивных библиотек. На основе NumPy появляются новые типы массивов, возможности которых выходят за рамки того, что предлагает библиотека. Например, библиотеки Dask, CuPy или XND.

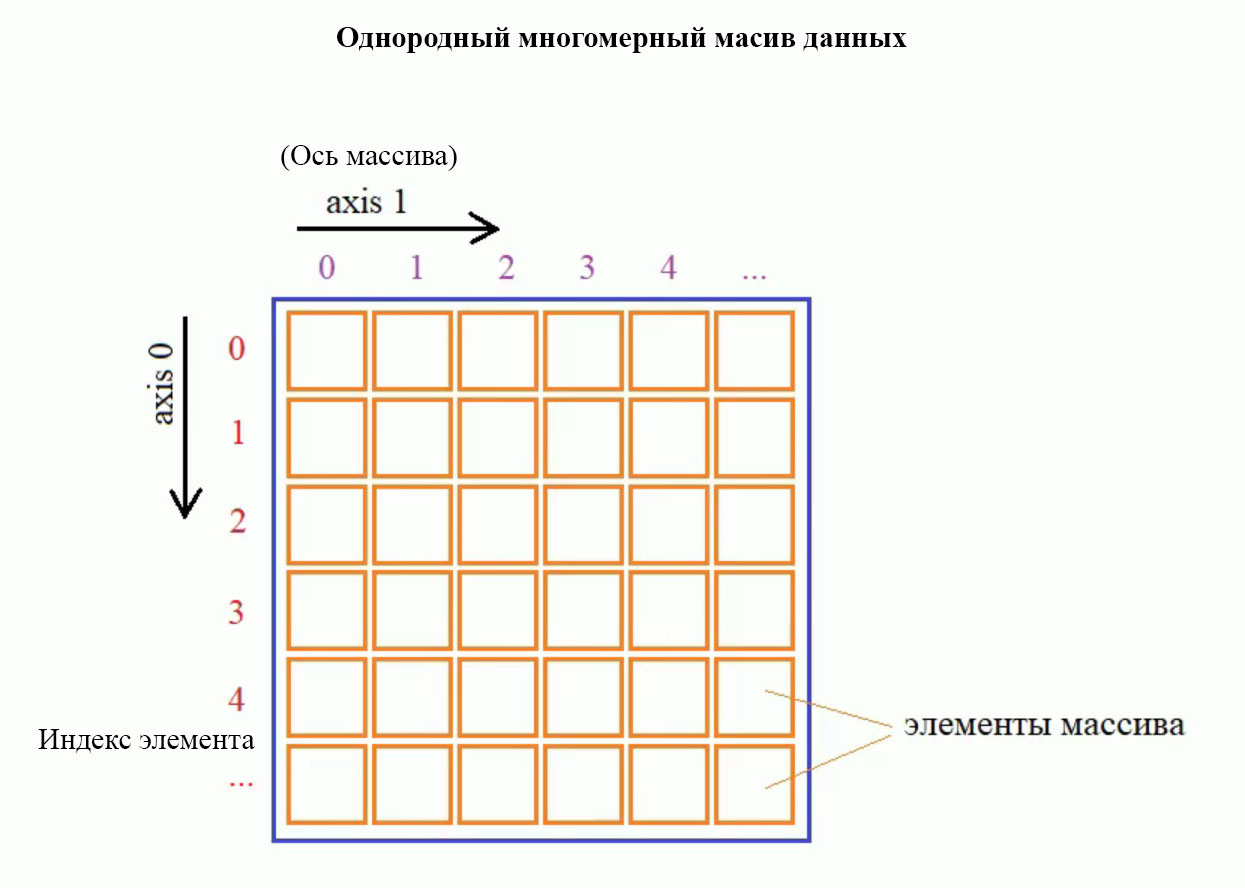
Data Science. В основе экосистемы для анализа данных лежит NumPy. Библиотека используется на всех этапах работы с данными: извлечение и преобразование, анализ, моделирование и оценка, репрезентация.

Machine Learning. Библиотеки для машинного обучения scikit-learn и SciPy тоже работают благодаря вычислительным мощностям NumPy.

Визуализация данных. По сравнению непосредственно с Python возможности NumPy позволяют исследователям визуализировать наборы данных, которые гораздо больше по размеру. Например, библиотека лежит в основе системы PyViz, которая включает в себя десятки программ для визуализации.

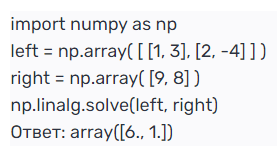
Как работает NumPy?

Для начала разберемся в устройстве массивов, которые обрабатывает NumPy. Рассмотрим однородный двумерный массив. Он выглядит как простая таблица — две оси значений и ячейки внутри (элементы массива). Если появится третья ось, то массив станет трехмерным. Важное условие — все элементы должны иметь единый тип данных, например только целые числа.



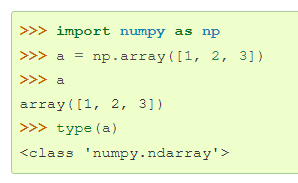
Конечно, кроме двумерных массивов, библиотека NumPy обрабатывает и другие, с различным количеством осей. Эту вариативность обозначают числом N, как любую переменную в математической задаче. Поэтому обычно говорят, что NumPy работает с N-мерными массивами данных.

С этими данными NumPy производит вычисления, используя математические функции, генераторы случайных чисел, линейные уравнения или преобразования Фурье. Например, можно решить систему уравнений методом linalg.solve:

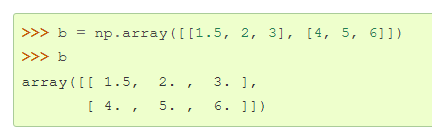


Создание массивов

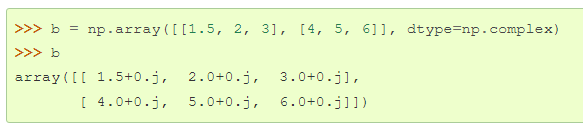
В NumPy существует много способов создать массив. Один из наиболее простых - создать массив из обычных списков или кортежей Python, используя функцию numpy.array() (запомните: array - функция, создающая объект типа ndarray):



Функция array() трансформирует вложенные последовательности в многомерные массивы. Тип элементов массива зависит от типа элементов исходной последовательности (но можно и переопределить его в момент создания).

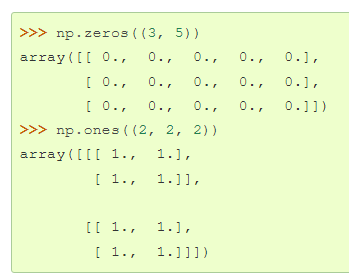


Можно также переопределить тип в момент создания:

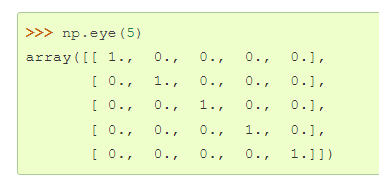


Функция array() не единственная функция для создания массивов. Обычно элементы массива вначале неизвестны, а массив, в котором они будут храниться, уже нужен. Поэтому имеется несколько функций для того, чтобы создавать массивы с каким-то исходным содержимым (по умолчанию тип создаваемого массива — float64).

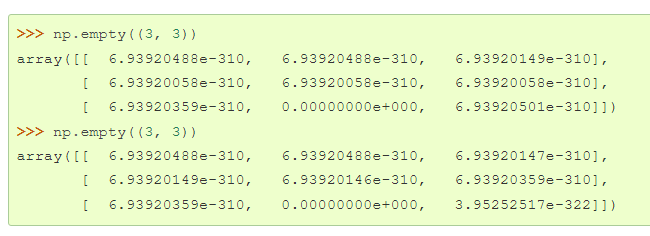
Функция zeros() создает массив из нулей, а функция ones() — массив из единиц. Обе функции принимают кортеж с размерами, и аргумент dtype:



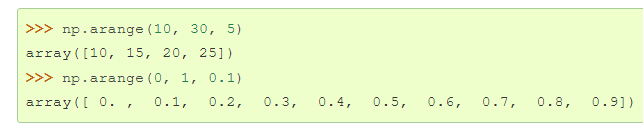
Функция eye() создаёт единичную матрицу (двумерный массив)



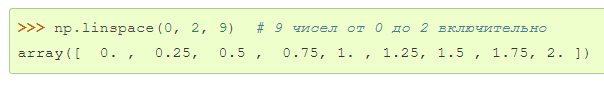
Функция empty() создает массив без его заполнения. Исходное содержимое случайно и зависит от состояния памяти на момент создания массива (то есть от того мусора, что в ней хранится):



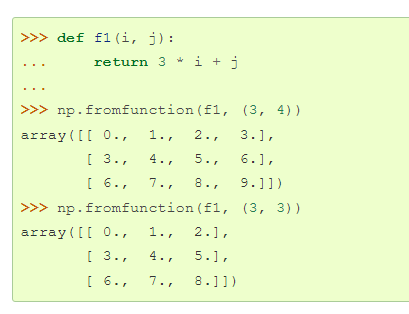
Для создания последовательностей чисел, в NumPy имеется функция arange(), аналогичная встроенной в Python range(), только вместо списков она возвращает массивы, и принимает не только целые значения:



Вообще, при использовании arange() с аргументами типа float, сложно быть уверенным в том, сколько элементов будет получено (из-за ограничения точности чисел с плавающей запятой). Поэтому, в таких случаях обычно лучше использовать функцию linspace(), которая вместо шага в качестве одного из аргументов принимает число, равное количеству нужных элементов:

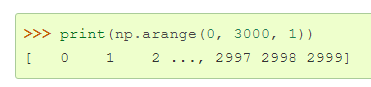


fromfunction(): применяет функцию ко всем комбинациям индексов



Печать массивов

Если массив слишком большой, чтобы его печатать, NumPy автоматически скрывает центральную часть массива и выводит только его уголки.



Если вам действительно нужно увидеть весь массив, используйте функцию numpy.set\_printoptions:



И вообще, с помощью этой функции можно настроить печать массивов "под себя". Функция numpy.set\_printoptions принимает несколько аргументов:

precision : количество отображаемых цифр после запятой (по умолчанию 8).

threshold : количество элементов в массиве, вызывающее обрезание элементов (по умолчанию 1000).

edgeitems : количество элементов в начале и в конце каждой размерности массива (по умолчанию 3).

linewidth : количество символов в строке, после которых осуществляется перенос (по умолчанию 75).

suppress : если True, не печатает маленькие значения в scientific notation (по умолчанию False).

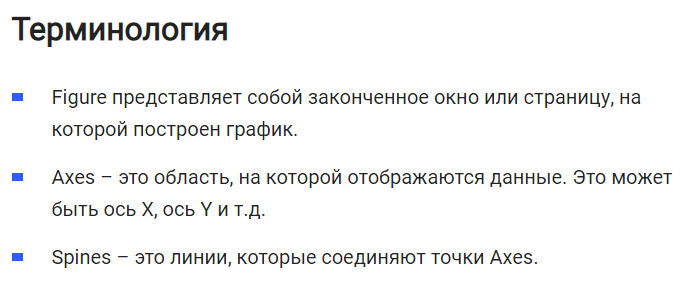
nanstr : строковое представление NaN (по умолчанию 'nan').

infstr : строковое представление inf (по умолчанию 'inf').

formatter : позволяет более тонко управлять печатью массивов. Здесь я его рассматривать не буду, можете почитать здесь (на английском).

**1. Библиотека matplotlib**

Библиотека matplotlib в Python помогает нам отображать данные на графиках в простейшем виде. Если вы знакомы с построением графиков в MATLAB, то Matplotlib будет легко использовать для базового построения графиков. Чтобы начать понимать, как Matplotlib помогает нам строить графики и фигуры визуализации для представления данных, нам нужно знать некоторые из основных терминов, которые мы будем часто использовать в этом посте. Давайте сначала изучим эти термины.



Установка Matplotlib



Теперь мы готовы создать несколько примеров, используя эту библиотеку визуализации данных. Начало работы В этом разделе мы начнем с построения графика и начнем передавать данные функциям matplotlib в python .

Линейный график

Мы начнем с очень простого примера построения графика. Мы просто будем использовать два списка Python в качестве источника данных для точек графика. Напишем для этого фрагмент кода:

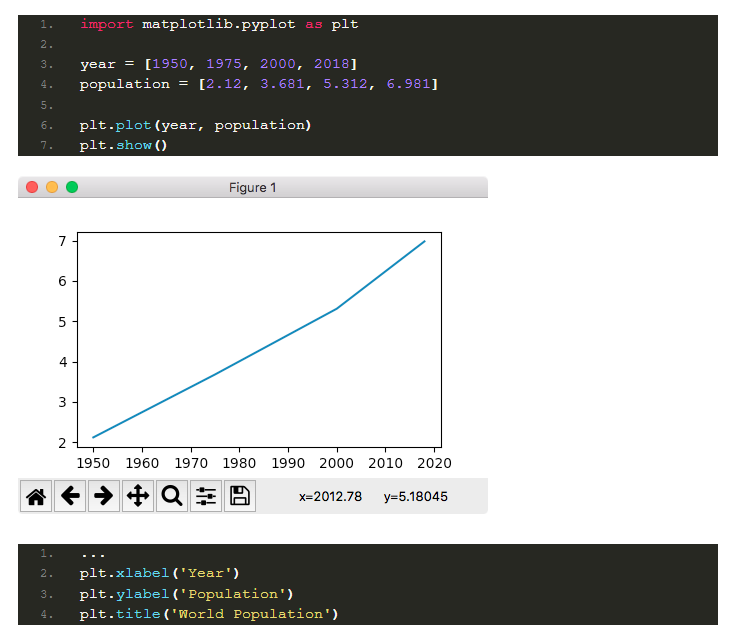
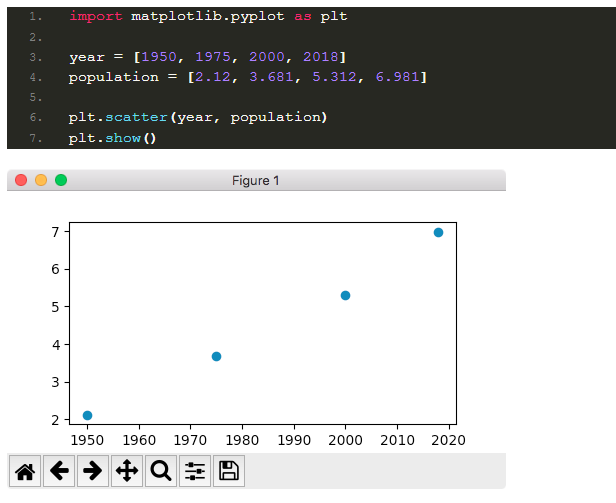
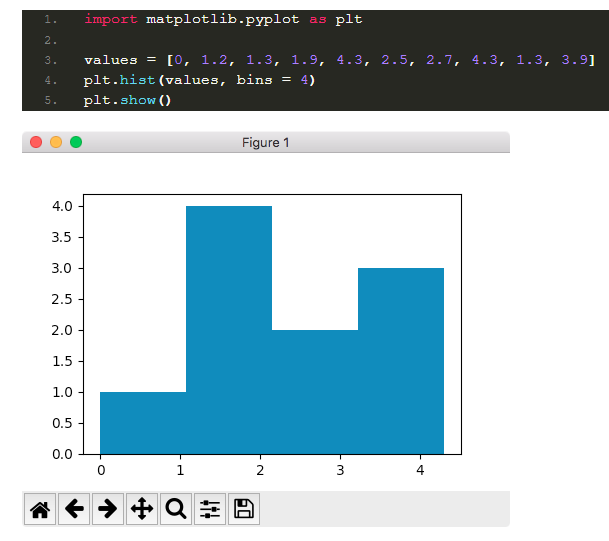


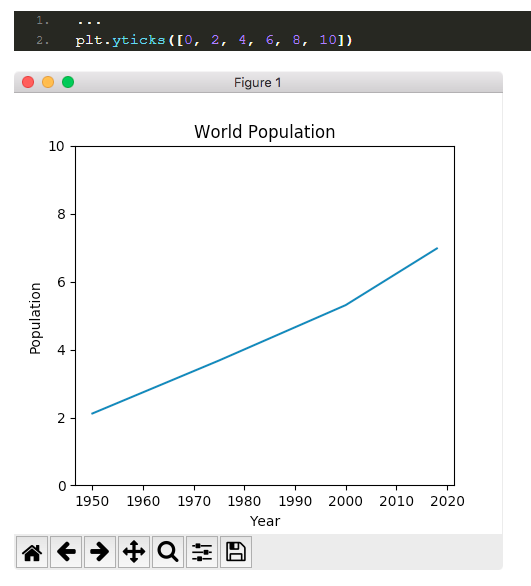
Диаграмма рассеяния Вышеупомянутый график показывал точки, которые фактически не были переданы в массиве, поскольку он показывает линию. Что, если мы хотим видеть только фактические точки на графике? Диаграмма разброса достигает этого:



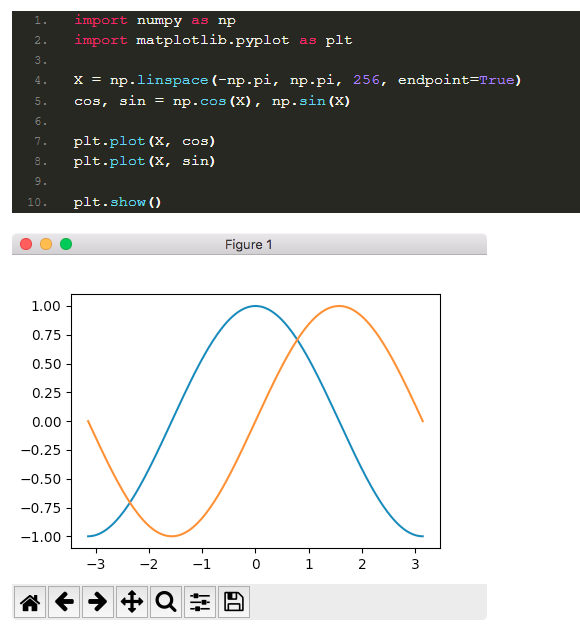
Гистограммы В этом разделе мы познакомим вас с гистограммами. В то время как графики информируют нас о том, как меняются наши данные, гистограмма описывает, как наши данные распределяются. Чем больше значений в диапазоне, тем выше полоса диапазона. Мы используем функцию hist() для построения гистограммы. У него есть 2 важных параметра: список значений для построения; количество диапазонов для распределения этих точек. Продемонстрируем это с помощью фрагмента кода:



Настройка Plot Если вы заметили первый график Line, мы увидим, что ось Y не начинается с 0. Мы можем изменить это:



Рисование нескольких кривых Совершенно распространено рисование нескольких кривых на одном графике для сравнения. Попробуем вот это:



Изменение цвета и добавление надписи на графике Как мы видели, кривые выглядят красиво, но разве все они не так похожи? Что, если мы хотим изменить их цвет и показать, что представляет каждый цвет? Попробуем вместе нарисовать синусоидальную и косинусную кривые:

