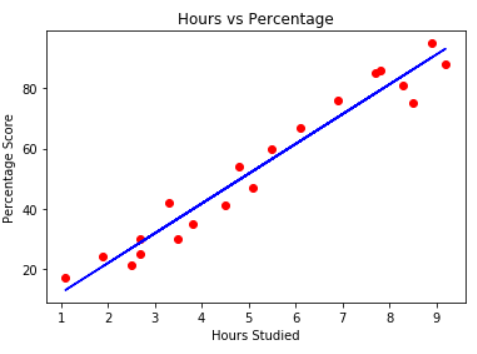
**ЛИНЕЙНАЯ РЕГРЕССИЯ С ПОМОЩЬЮ SCIKIT-LEARN В PYTHON.**

Существует два типа алгоритмов машинного обучения с учителем: регрессия и классификация. Первый прогнозирует непрерывные выходы значений, а второй – дискретные. Например, прогнозирование стоимости дома в долларах является проблемой регрессии, тогда как прогнозирование злокачественной или доброкачественной опухоли является проблемой классификации. В практической работе изучим, что такое линейная регрессия и как ее можно реализовать с помощью библиотеки Python Scikit-Learn, которая является одной из самых популярных библиотек машинного обучения для Python.

**Теория линейной регрессии**. Термин «линейность» в алгебре относится к линейной зависимости между двумя или более переменными. Если мы нарисуем это соотношение в двухмерном пространстве (в данном случае между двумя переменными), мы получим прямую линию. Давайте рассмотрим скрипт, в котором мы хотим определить линейную зависимость между количеством часов, которые студент учится, и процентом оценок, которые студент набирает на экзамене. Мы хотим выяснить, сколько часов ученик готовится к тесту, насколько высокий балл он может набрать? Если мы нанесем независимую переменную (часы) на ось x, а зависимую переменную (процент) на ось y, линейная регрессия даст нам прямую линию, которая наилучшим образом соответствует точкам данных, как показано на рисунке ниже.



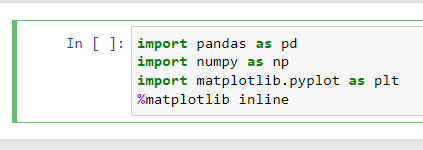
Мы знаем, что уравнение прямой в основном: y = mx + b

Где, b – точка пересечения, а m – наклон линии. Таким образом, алгоритм линейной регрессии дает нам наиболее оптимальное значение для точки пересечения и наклона (в двух измерениях). Переменные y и x остаются неизменными, поскольку они являются функциями данных и не могут быть изменены. Значения, которые мы можем контролировать, – это точка пересечения и наклон. В зависимости от значений точки пересечения и наклона может быть несколько прямых линий. По сути, алгоритм линейной регрессии помещает несколько строк в точки данных и возвращает строку, которая дает наименьшую ошибку. Эту же концепцию можно распространить на случаи, когда существует более двух переменных. Это называется множественной линейной регрессией. Например, рассмотрим скрипт, в котором вы должны спрогнозировать цену дома на основе его площади, количества спален, среднего дохода людей в этом районе, возраста дома и т.д. В этом случае зависимая переменная зависит от нескольких независимых переменных. Модель регрессии, включающая несколько переменных, может быть представлена как: y = b0 + m1b1 + m2b2 + m3b3 + … … mnbn Это уравнение гиперплоскости. Помните, что модель линейной регрессии в двух измерениях – это прямая линия, в трех измерениях – это плоскость, а в более чем трех измерениях – гиперплоскость.

**Простая линейная регрессия.**

В этой задаче регрессии мы спрогнозируем процент оценок, которые, как ожидается, получит студент, на основе количества часов, которые он изучил. Это простая задача линейной регрессии, поскольку она включает всего две переменные.

**Импорт библиотек.** Чтобы импортировать необходимые библиотеки для этой задачи, выполните следующие операторы импорта:



Код был выполнен с использованием Jupyter iPython Notebook.

**Набор данных.** Следующая команда импортирует набор данных CSV с помощью Pandas:

dataset = pd.read\_csv('D:\student\_scores.csv')

Теперь давайте немного исследуем наш набор данных. Для этого выполните следующий скрипт:

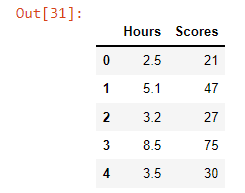
dataset.shape

После этого вы должны увидеть следующее: (25, 2) Это означает, что в нашем наборе данных 25 строк и 2 столбца.

Давайте посмотрим, как на самом деле выглядит наш набор данных. Для этого используйте метод head():

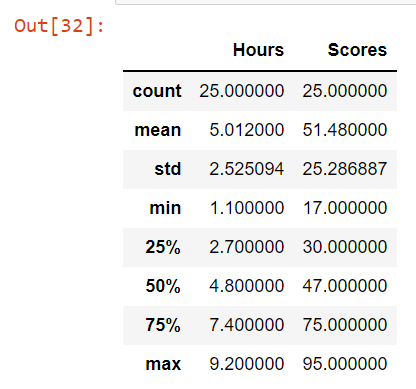
dataset.head()

Вышеупомянутый метод извлекает первые 5 записей из нашего набора данных, которые будут выглядеть следующим образом:

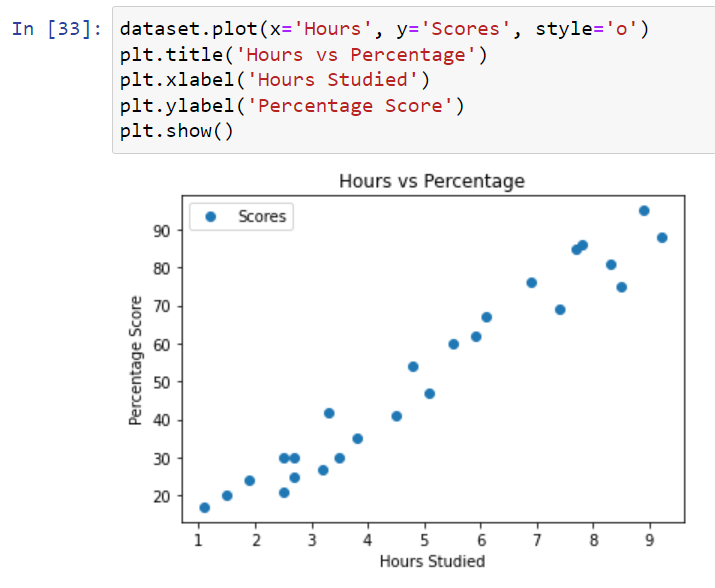


Чтобы увидеть статистические детали набора данных, мы можем использовать description():

dataset.describe()



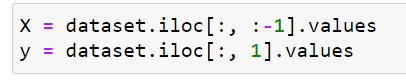
И, наконец, давайте нарисуем наши точки данных на двухмерном графике, чтобы взглянуть на наш набор данных и посмотреть, сможем ли мы вручную найти какую-либо связь между данными. Мы можем создать сюжет с помощью следующего скрипта:



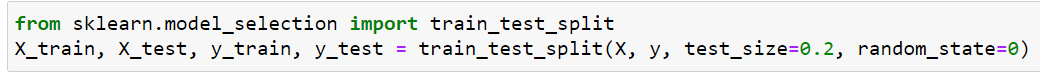
В приведенном выше скрипте мы используем функцию plot() dataframe pandas и передаем ей имена столбцов для координаты x и координаты y, которые являются «часами» и «счетами» соответственно.

Из приведенного выше графика мы можем ясно видеть, что существует положительная линейная зависимость между количеством изученных часов и процентом набранных баллов.

**Подготовка данных.** Теперь у нас есть представление о статистических деталях наших данных. Следующим шагом является разделение данных на «атрибуты» и «метки». Атрибуты – это независимые переменные, а метки – это зависимые переменные, значения которых должны быть предсказаны. В нашем наборе данных всего два столбца. Мы хотим предсказать процентную оценку в зависимости от изученных часов. Поэтому наш набор атрибутов будет состоять из столбца «Часы», а меткой будет столбец «Оценка». Чтобы извлечь атрибуты и метки, выполните следующий сценарий:

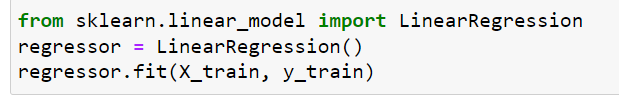


Атрибуты хранятся в переменной X. Мы указали «-1» в качестве диапазона для столбцов, так как мы хотели, чтобы наш атрибут содержал все столбцы, кроме последнего, которым является «Результаты». Точно так же переменная **y** содержит метки. Мы указали 1 для столбца метки, так как индекс столбца «Результаты» равен 1. Помните, что индексы столбца начинаются с 0, причем 1 является вторым столбцом. В следующем разделе мы увидим лучший способ указать столбцы для атрибутов и меток. Теперь, когда у нас есть атрибуты и метки, следующим шагом будет разделение этих данных на обучающий и тестовый наборы. Мы сделаем это с помощью встроенного в Scikit-Learn метода train\_test\_split():



Приведенный выше скрипт разделяет 80% данных на обучающий набор, а 20% данных – на набор тестов. Переменная test\_size -–это то место, где мы фактически указываем пропорцию набора тестов.

**Обучение алгоритму.** Мы разделили наши данные на наборы для обучения и тестирования, и теперь, наконец, пришло время обучить наш алгоритм. Выполните следующую команду:



С Scikit-Learn чрезвычайно просто реализовать модели линейной регрессии, поскольку все, что вам действительно нужно сделать, это импортировать класс LinearRegression, создать его экземпляр и вызвать метод fit() вместе с нашими обучающими данными. В разделе теории мы сказали, что модель линейной регрессии в основном находит наилучшее значение для точки пересечения и наклона, в результате чего получается линия, которая наилучшим образом соответствует данным. Чтобы увидеть значение точки пересечения и наклона, вычисленное алгоритмом линейной регрессии для нашего набора данных, выполните следующий код. Чтобы получить перехват:

print(regressor.intercept\_)

Полученное значение должно быть 2.018160041434683.

Для получения наклона (коэффициента x):

print(regressor.coef\_)

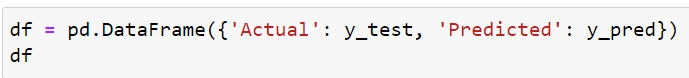
Результат должен быть 9.91065648.

Это означает, что на каждую единицу изменения в изученных часах изменение оценки составляет около 9,91%. Или, проще говоря, если студент учится на один час больше, чем готовился к экзамену ранее, он может рассчитывать на повышение на 9,91% баллов, полученных студентом ранее.

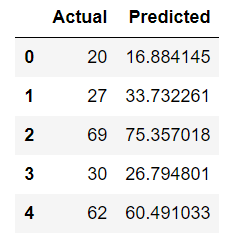
**Прогнозы.** Теперь, когда мы обучили наш алгоритм, пришло время сделать некоторые прогнозы. Для этого мы воспользуемся нашими тестовыми данными и посмотрим, насколько точно наш алгоритм предсказывает процентную оценку. Чтобы сделать прогнозы на тестовых данных, выполните следующий скрипт:

y\_pred = regressor.predict(X\_test)

Y\_pred – это массив numpy, который содержит все предсказанные значения для входных значений в серии X\_test. Чтобы сравнить фактические выходные значения для X\_test с прогнозируемыми значениями, выполните следующий скрипт:



Результат выглядит так:



Хотя наша модель не очень точна, прогнозируемые проценты близки к фактическим.

Примечание: Значения в вышеприведенных столбцах могут отличаться в вашем случае, потому что функция train\_test\_split случайным образом разбивает данные на обучающие и тестовые наборы, и ваши разбиения, вероятно, будут отличаться от показанного на рисунке.

**Оценка алгоритма.** Последний шаг – оценить производительность алгоритма. Этот шаг особенно важен для сравнения того, насколько хорошо разные алгоритмы работают с конкретным набором данных. Для алгоритмов регрессии обычно используются три показателя оценки:

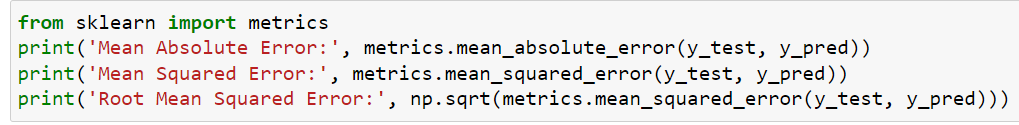
1) Средняя абсолютная ошибка (MAE) – это среднее абсолютное значение ошибок. Он рассчитывается как: Mean Absolute Error

2) Среднеквадратичная ошибка (MSE) – это среднее значение квадратов ошибок, которое рассчитывается как: Mean Squared Error

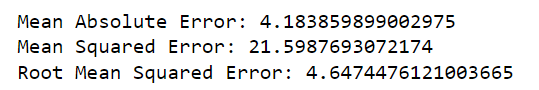
3) Среднеквадратичная ошибка (RMSE) – это квадратный корень из среднего квадрата ошибок: Root Mean Squared Error

К счастью, нам не нужно выполнять эти вычисления вручную. Библиотека Scikit-Learn поставляется со встроенными функциями, которые можно использовать, чтобы узнать эти значения для нас.

Давайте найдем значения этих показателей, используя наши тестовые данные. Выполните следующий код:

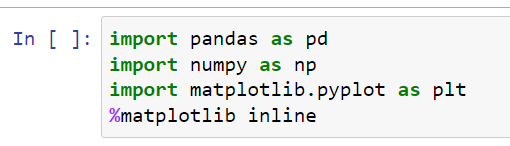


Результат будет выглядеть:



Вы можете видеть, что значение среднеквадратичной ошибки составляет 4,64, что составляет менее 10% от среднего значения процентов всех студентов, т.е. 51,48. Это означает, что наш алгоритм проделал достойную работу. **Множественная линейная регрессия**. Линейная регрессия с участием нескольких переменных называется «множественной линейной регрессией». Шаги по выполнению множественной линейной регрессии почти аналогичны шагам простой линейной регрессии. Разница заключается в оценке. Вы можете использовать его, чтобы узнать, какой фактор имеет наибольшее влияние на прогнозируемый результат и как разные переменные связаны друг с другом. В этом разделе мы будем использовать множественную линейную регрессию для прогнозирования потребления газа (в миллионах галлонов) в 48 штатах США на основе налогов на газ (в центах), дохода на душу населения (в долларах), дорог с твердым покрытием (в милях) и доли население, имеющее водительские права. Первые два столбца в приведенном выше наборе данных не предоставляют никакой полезной информации, поэтому они были удалены из файла набора данных. Теперь давайте разработаем регрессионную модель для этой задачи. Импорт библиотек Следующий скрипт импортирует необходимые библиотеки:

**Импорт библиотек.** Следующий скрипт импортирует необходимые библиотеки:



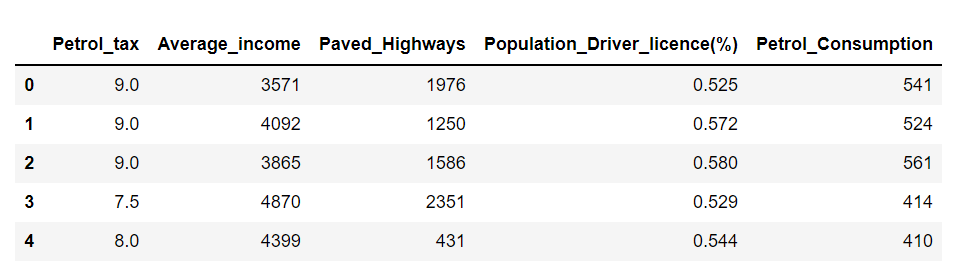
**Набор данных.** Следующая команда импортирует набор данных из файла:

dataset = pd.read\_csv('D:\petrol\_consumption.csv')

Как и в прошлый раз, давайте посмотрим, как на самом деле выглядит наш набор данных. Выполните команду head():

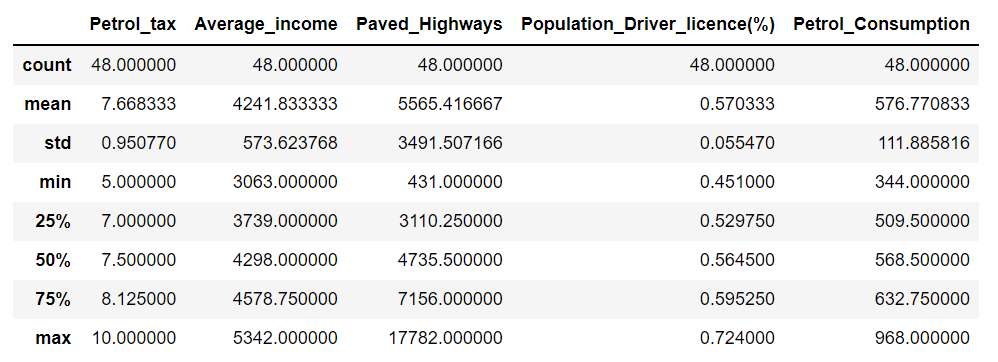
dataset.head()

Первые несколько строк нашего набора данных выглядят так:



Чтобы увидеть статистические детали набора данных, мы снова воспользуемся командой describe():

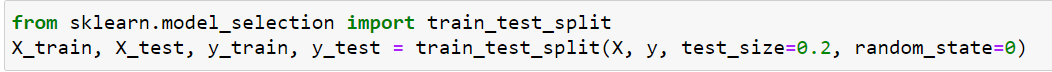
dataset.describe()



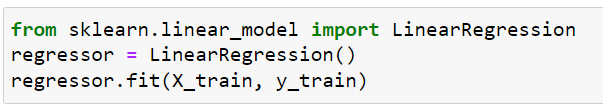
**Подготовка данных**. Следующим шагом является разделение данных на атрибуты и метки, как мы делали ранее. Однако, в отличие от прошлого раза, на этот раз мы собираемся использовать имена столбцов для создания набора атрибутов и метки. Выполните следующий скрипт:



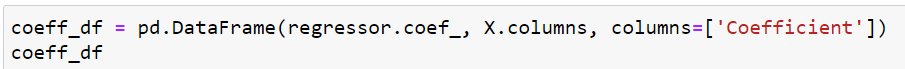
Выполните следующий код, чтобы разделить наши данные на обучающий и тестовый наборы:



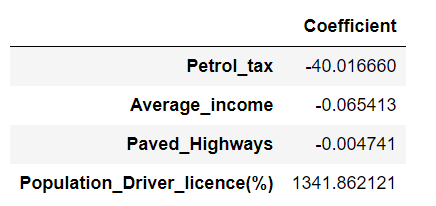
**Обучение алгоритму.** И, наконец, для обучения алгоритма мы выполняем тот же код, что и раньше, используя метод fit() класса LinearRegression:



Как было сказано ранее, в случае многомерной линейной регрессии регрессионная модель должна найти наиболее оптимальные коэффициенты для всех атрибутов. Чтобы увидеть, какие коэффициенты выбрала наша регрессионная модель, выполните следующий скрипт:



Результат должен выглядеть примерно так:

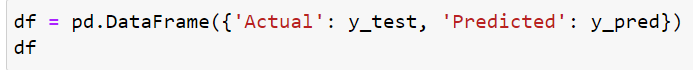


Это означает, что увеличение «petrol\_tax» на единицу приводит к снижению потребления газа на 40 миллиона галлонов. Аналогичным образом, увеличение доли населения, имеющего водительские права, приводит к увеличению потребления газа на 1,341 миллиарда галлонов. Мы видим, что «Average\_income» и «Paved\_Highways» очень мало влияют на потребление газа.

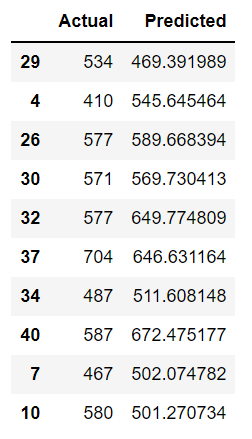
**Прогнозы.** Чтобы сделать прогнозы на тестовых данных, выполните следующий скрипт:

y\_pred = regressor.predict(X\_test)

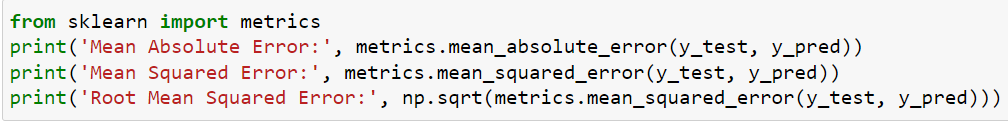
Чтобы сравнить фактические выходные значения для X\_test с прогнозируемыми значениями, выполните следующий скрипт:



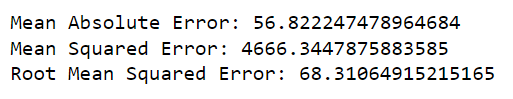
Результат выглядит так:



**Оценка алгоритма.** Последний шаг – оценить производительность алгоритма. Мы сделаем это, найдя значения для MAE, MSE и RMSE. Выполните следующий скрипт:



Результат будет выглядеть примерно так:



Вы можете видеть, что значение среднеквадратичной ошибки составляет 68,3, что немного больше 10% от среднего значения потребления газа во всех штатах. Это означает, что наш алгоритм не был очень точным, но все же может делать достаточно хорошие прогнозы.

Есть много факторов, которые могли повлиять на эту неточность, некоторые из которых перечислены здесь:

1) Требуется больше данных: данных за один год – это не так уж и много, тогда как накопление данных за несколько лет могло бы помочь нам немного повысить точность.

2) Ошибочные предположения: мы предположили, что эти данные имеют линейную зависимость, но это может быть не так. Визуализация данных может помочь вам определить это.

3) Плохие функции: используемые нами функции могли не иметь достаточно высокой корреляции со значениями, которые мы пытались предсказать.

В практической работе мы изучили один из самых фундаментальных алгоритмов машинного обучения, то есть линейную регрессию. Мы реализовали как простую линейную регрессию, так и множественную линейную регрессию с помощью библиотеки машинного обучения Scikit-Learn.

Отчет сохранить в виде HTML файла.

