**Лабораторная работа № 1.**

**Часть 1. Линейная регрессия.**

Цель: закрепить теоретические сведения о линейной регрессии, получить практические навыки построения модели линейной регрессии.

Задачи:

* Вспомнить и усвоить основные понятия моделей линейной регрессии;
* Научиться строить модели линейной регрессии с использованием метода Наименьших квадратов;
* Научиться использовать библиотеку statsmodels;
* Научиться строить модели линейной регрессии с использованием метода градиентного спуска;
* Научиться использовать библиотеку scikit-learn.

**Линейная регрессия. Концепция**

Экономические явления, как правило, определяются большим числом одновременно и совокупно действующих факторов. В связи с этим часто возникает задача исследования зависимости одной зависимой переменной Y от нескольких объясняющих переменных *X1, X2, ,Xm* . Это задача решается с помощью ***множественного регрессионного анализа.***

Пусть имеется вектор наблюдаемых значений m объясняющих переменных . Тогда вектор наблюдаемых значений объясняемой переменной Y имеет вид: *y1,y2,…,yp*, а линейная регрессионная модель может быть представлена в стандартном виде так:

*yi =* *β1xi1 + β2xi2 +β3xi3 +…+ βmxim + ui , i = 1,…,p* (1)

Полагаем *xi1  = 1*, т.е. *β1* - свободный член,  - случайная ошибка, являющаяся разницей между реальным и прогнозным значением.

Метод наименьших квадратов

Оценкой наименьших квадратов является такой вектор параметров *(β1,β2,…,βm)*, который минимизирует сумму квадратов отклонений наблюдаемых значений от отчетных, т.е.

.

Уравнение (1) можно записать в матричном виде так:

*Y = XB + U* (3),

где

 (4),  (5),  (6),  (7)

Тогда уравнение (2) в матричной форме будет выглядеть следующим образом:

*S = UTU = (Y - XB)T(Y - XB)*

Решением данного уравнения будет вектор:



Вывод данного вектора подробно был рассмотрен на лекции.

Метод наименьших квадратов в python.

Библиотека NumPy

NumPy — это библиотека для языка программирования Python, которая предоставляет поддержку для работы с многомерными массивами и матрицами, а также включает в себя большое количество математических функций для выполнения операций над этими массивами. Она является одной из основных библиотек для научных вычислений в Python и широко используется в области анализа данных, машинного обучения и численных расчетов. np.array — это функция, предоставляемая библиотекой NumPy, которая используется для создания объектов типа ndarray. Она принимает в качестве аргумента последовательность (например, список или кортеж) и возвращает новый массив NumPy.

Пример:

import numpy as np

# Создание одномерного массива

array\_1d = np.array([1, 2, 3, 4, 5])

print("Одномерный массив:", array\_1d)

# Создание двумерного массива

array\_2d = np.array([[1, 2, 3], [4, 5, 6]])

print("Двумерный массив:\n", array\_2d)

Основные возможности NumPy:

* Массивы: NumPy предоставляет объект ndarray, который является многомерным массивом фиксированного размера, содержащим элементы одного типа. Это позволяет эффективно хранить и обрабатывать данные. Для создания массива array
* Операции над массивами: NumPy поддерживает векторизированные операции, что позволяет выполнять математические операции над массивами без необходимости использования циклов.
* Функции для линейной алгебры: NumPy включает функции для выполнения операций линейной алгебры, таких как умножение матриц, вычисление определителей и собственных значений.
* Статистические функции: Библиотека предоставляет функции для вычисления различных статистических показателей, таких как среднее, медиана, стандартное отклонение и т.д.
* Удобные функции для работы с данными: NumPy включает функции для генерации случайных чисел, работы с интервалами и преобразования данных.

Для установки библиотеки необходимо в консоли написать:

pip install numpy

После установки для использования данной библиотеки необходимо импортировать данную библиотеку:

import numpy as np

Рассмотрим использование библиотеки NumPy для построения модели линейной регрессии с использованием метода наименьших квадратов.

1. На первом этапе сгенерируем исходные данные:

np.random.seed(0)  # Для воспроизводимости

n\_samples = 100

n\_features = 3

# Генерируем случайные данные

X = np.random.rand(n\_samples, n\_features)

# Генерируем коэффициенты для линейной модели

true\_coefficients = np.array([1.5, -2.0, 3.0])

# Генерируем целевую переменную с добавлением шума

y = X @ true\_coefficients + np.random.normal(0, 0.1, n\_samples)

# Добавляем столбец единиц для свободного члена (intercept)

X\_b = np.c\_[np.ones((n\_samples, 1)), X]  # Добавляем столбец единиц

Функция np.random.seed() в библиотеке NumPy используется для инициализации генератора случайных чисел. Установка seed позволяет получить воспроизводимые результаты при генерации случайных чисел. Это особенно полезно в научных исследованиях и при разработке алгоритмов, когда необходимо повторить эксперименты с теми же случайными данными.

Когда вы устанавливаете seed, например, с помощью np.random.seed(0), вы фиксируете начальное состояние генератора случайных чисел. Это означает, что каждый раз, когда вы будете генерировать случайные числа после установки этого параметра, вы будете получать одни и те же числа.

np.random.rand(n\_samples, n\_features) используется для генерации двумерного массива (матрицы) случайных чисел, которые равномерно распределены в диапазоне от 0 до 1.

Параметры:

n\_samples: количество строк в массиве, что соответствует количеству объектов (или наблюдений).

n\_features: количество столбцов в массиве, что соответствует количеству признаков (или переменных) для каждого объекта.

Последними двумя строчками мы создаем данные для зависимой переменной y таким образом, чтобы они линейно зависели от X (коэффициента линейной зависимости задали в true\_coefficients, но дополнительно добавляем нормально распределенный шум с помощью np.random.normal(0, 0.1, n\_samples).

В NumPy оператор @ используется для выполнения матричного умножения. Он был введен в Python 3.5 и позволяет более удобно и читаемо выполнять операции умножения матриц, чем использование функции np.dot() или метода .dot(), которые также позволяют выполнять матричное умножение.

Для того, чтобы в нашем уравнении было учтено смещение (свободный член *β0*) добавляем в матрицу X вектор длины X, состоящий из

1. Реализуем метод наименьших квадратов с использованием библиотеки NumPy:

# Метод наименьших квадратов

# theta = (X\_b^T \* X\_b)^(-1) \* X\_b^T \* y

theta\_best = np.linalg.inv(X\_b.T @ X\_b) @ X\_b.T @ y

# Выводим результаты

print("Коэффициенты регрессии:")

print(theta\_best)

В результате получаем коэффициенты:

[-0.00497861, 1.45913248, -2.00200434, 3.0253231 ], которые очень близки к нашей исходной зависимости [1.5, -2.0, 3.0].

При построении модели линейной регрессии важно заранее знать характер зависимостей в данных. Для этого удобно использовать визуализацию с применением диаграммы рассеяния. Для построения диаграмму рассеяния в python удобно использовать библиотеку визуализации matplotlib. Для построения диаграмм рассеяния для нашего примера можно использовать следующий код:

import matplotlib.pyplot as plt

plt.scatter(X[:, 1], y)

plt.show()

plt.scatter(X[:, 2], y)

plt.show()

plt.scatter(X[:, 3], y)

plt.show()

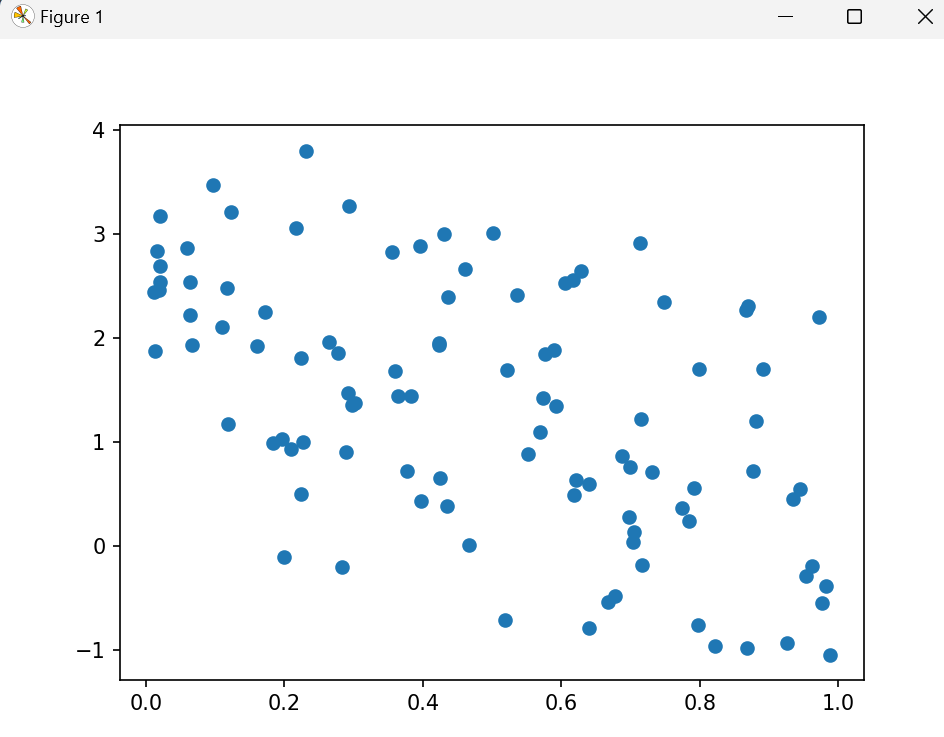
Обратите внимание, что обращение к двумерному массиву NumPy производится за счет использования индексов. Первый индекс означает номер строки, второй – номер столбца.

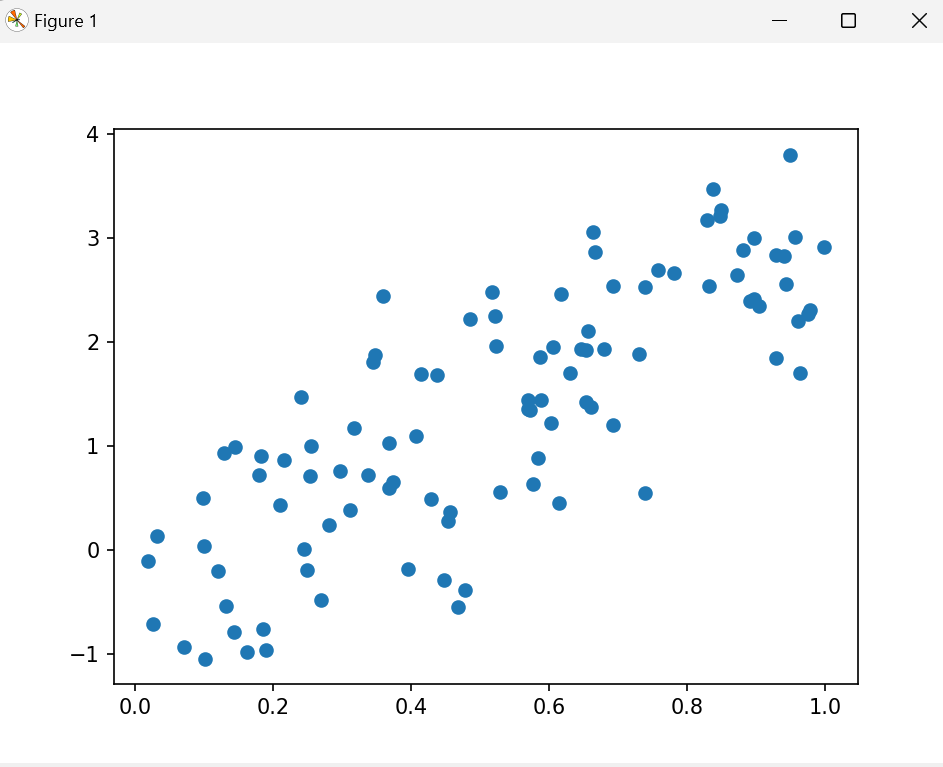
Если вы хотите сделать срез по столбцу или строке используется синтаксис:

[start:stop]. Если вы хотите сделать срез, который начинается с нулевого элемента (элемента с индексом 0), ноль можно не писать. Аналогично, если вы хотите сделать срез от какого-то элемента до последнего, индекс последнего элемента так же можно не писать.

Изображение выглядит как снимок экрана, диаграмма

Автоматически созданное описание





**Задание 1**

1. Импортируйте библиотеку NumPy. С помощью данной библиотеки загрузите файл linear.csv и выведите в консоль. Для загрузки файла в массив NumPy используйте функцию np.genfromtxt, а в скобках укажите название файла. Загружаемый файл должен лежать в той же папке, что и исполняемый скрипт.

# Загрузка данных из CSV-файла

data = np.genfromtxt('data.csv', delimiter=',', skip\_header=1)

Параметры функции np.genfromtxt():

fname: имя файла или путь к файлу. В примере 'data.csv'

delimiter: символ, используемый для разделения значений (по умолчанию пробел).

skip\_header: количество строк, которые нужно пропустить в начале файла (например, для пропуска заголовков).

dtype: тип данных для массива (по умолчанию float).

1. Разделите данные на зависимые и независимые переменные с использованием срезов, используя следующий синтаксис:

X = data[:, :-1]

y = data[:, -1]

отрицательный индекс означает последний элемент.

1. Для загруженных данных построить диаграмму рассеяния зависимостей y от каждого x. Сделайте выводы о характере зависимости.
2. Реализуйте алгоритм линейной регрессии, рассмотренный выше. Выведите результат в консоль.

**Использование библиотеки statsmodels**

statsmodels — это библиотека для Python, которая предоставляет классы и функции для оценки статистических моделей, выполнения статистических тестов и анализа данных. Она особенно полезна для эконометрики и статистического анализа, и часто используется в сочетании с библиотеками, такими как NumPy и Pandas.

Основные возможности statsmodels:

* Оценка моделей: statsmodels поддерживает множество статистических моделей, включая линейные и нелинейные регрессии, временные ряды, модели для анализа выживаемости и многие другие.
* Статистические тесты: Библиотека предоставляет функции для выполнения различных статистических тестов, таких как тесты на нормальность, тесты на равенство дисперсий и т.д.
* Анализ временных рядов: statsmodels включает инструменты для анализа временных рядов, включая ARIMA, SARIMA и другие модели.
* Интерпретация результатов: Библиотека предоставляет удобные методы для получения сводок и интерпретации результатов моделей.

Если у вас еще не установлена библиотека statsmodels, вы можете установить ее с помощью pip:

pip install statsmodels

Для импорта библиотеки пишем:

import statsmodels.api as sm

Для реализации модели линейной регрессии с использованием данной библиотеки рассмотрим следующий пример:

# Создаем модель

model = sm.OLS(y, X)  # OLS - метод наименьших квадратов

results = model.fit()  # Подгоняем модель

print(results.summary())

# Получаем прогнозы

predictions = results.predict(X)  # Прогнозируем на тех же данных

# Выводим первые 5 прогнозов

print("Первые 5 прогнозов:")

print(predictions[:5])

В строке model = sm.OLS(y, X) вы создаете объект модели линейной регрессии с использованием метода наименьших квадратов (OLS — Ordinary Least Squares) из библиотеки statsmodels.

Параметры:

y: это зависимая переменная (вектор значений, которые вы хотите предсказать).

X: это матрица независимых переменных (факторы, которые вы используете для предсказания y). Важно, чтобы X включал константу (свободный член), если вы хотите, чтобы модель учитывала его.

model.fit() - это метод обучения (непосредственное вычисление искомых коэффициентов с использованием метода наименьших квадратов).

results.summary() – выводит итоговые результаты по полученной модели.

Что включает в себя results.summary():

* **Общая информация о модели:**
  + Dep. Variable: Зависимая переменная (например, Y).
  + Model: Тип модели (например, OLS).
  + Method: Метод оценки (обычно Least Squares).
  + Date: Дата и время выполнения.
  + Time: Время выполнения.
* **Статистика модели:**
  + R-squared: Коэффициент детерминации, который показывает, какая доля вариации зависимой переменной объясняется независимыми переменными.
  + Adj. R-squared: Скорректированный коэффициент детерминации, который учитывает количество независимых переменных в модели.
  + F-statistic: Статистика F для проверки значимости модели в целом.
  + Prob (F-statistic): p-значение для F-статистики.
* **Коэффициенты модели:**

Для каждой независимой переменной и свободного члена (константы) отображаются:

* coef: Оцененные коэффициенты.
* std err: Стандартные ошибки коэффициентов.
* t: Значения t-статистики для проверки значимости коэффициентов.
* P>|t|: p-значения для проверки значимости коэффициентов.
* [0.025 0.975]: Доверительные интервалы для коэффициентов.
* **Статистика для остатков:**
  + Omnibus: Тест на нормальность остатков.
  + Prob(Omnibus): p-значение для теста Омнибус.
  + Skew: Коэффициент асимметрии остатков.
  + Kurtosis: (эксцесс) статистическая мера, которая измеряет форму распределения данных и указывает на степень, в которой его точки отличаются от точек нормального распределения.
  + Durbin-Watson: Тест на автокорреляцию остатков.
  + Jarque-Bera (JB): Тест на нормальность остатков.
  + Prob(JB): p-значение для теста Джарке-Бера.

Для получения прогнозов используется метод results.predict(X).

**Задание 2**

1. Для данных, используемых на предыдущем этапе, построить модель линейной регрессии с использованием библиотеки statsmodels.
2. Загрузите файл test.csv и постройте прогнозы для этих точек. Сравните результат с фактическими данными. Сделайте вывод о качестве и точности модели.

Модель линейной регрессии, построенная на основе предоставленных данных, демонстрирует низкую точность. Это подтверждается низким значением коэффициента детерминации (R²), который составляет всего 0.008. Это указывает на то, что модель объясняет лишь незначительную часть вариации зависимой переменной на основе независимых переменных. Кроме того, высокая среднеквадратичная ошибка (MSE) — 2454893.32 — свидетельствует о значительном расхождении между фактическими и прогнозируемыми значениями. Таким образом, модель имеет ограниченную предсказательную способность и нуждается в доработке, например, в улучшении качества данных, добавлении новых факторов или использовании более сложных методов машинного обучения.

**Линейная регрессия с использованием метода градиентного спуск**а

Кроме метода наименьших квадратов для нахождения коэффициентов линейной регрессии можно использовать метод градиентного спуска.

Градиентный спуск — это итеративный алгоритм оптимизации, который используется для минимизации функции потерь. В контексте линейной регрессии он применяется для нахождения оптимальных значений коэффициентов модели, которые минимизируют разницу между предсказанными и фактическими значениями зависимой переменной.

Основные шаги градиентного спуска в линейной регрессии:

1. Инициализация: Начальные значения коэффициентов (обычно случайные или нулевые).
2. Выбор функции потерь: В линейной регрессии обычно используется среднеквадратичная ошибка (MSE):

где — фактические значения, а — предсказанные значения

1. Вычисление градиента: Градиент функции потерь по каждому коэффициенту показывает направление, в котором нужно изменять коэффициенты, чтобы уменьшить ошибку.
2. Вычисление градиента: Градиент функции потерь по каждому коэффициенту показывает направление, в котором нужно изменять коэффициенты, чтобы уменьшить ошибку.
3. Обновление коэффициентов: Коэффициенты обновляются по следующей формуле:

где – текущее значение j-го параметра модели, – градиент функции потерь , – скорость обучения (размер шага).

1. Повторение: Шаги 3 и 4 повторяются до тех пор, пока не будет достигнута сходимость (например, когда изменения в функции потерь становятся очень маленькими).

Попробуем реализовать данную логику на python. Предположим, что есть некоторые данные:

X = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]

y = [52, 74, 79, 95, 115, 110, 129, 126, 147, 146, 156, 184]

Для них необходимо реализовать модель линейной регрессии с использованием градиентного спуска. Реализуем данную логику с применением функций:

1. Напишем функцию вычисления градиента для смещения:

def get\_gradient\_at\_b(x, y, b, a):

  N = len(x)

  diff = 0

  for i in range(N):

    x\_val = x[i]

    y\_val = y[i]

    diff += (y\_val - ((a \* x\_val) + b))

  b\_gradient = -(2/N) \* diff

  return b\_gradient

Объяснение функции

Параметры:

* x: массив значений независимой переменной.
* y: массив значений зависимой переменной.
* b: текущее значение свободного члена (константы) в модели.
* a: текущее значение коэффициента наклона (углового коэффициента) в модели.

Вычисление градиента:

* N = len(x): определяет количество наблюдений.
* diff: переменная для накопления разностей между фактическими значениями y и предсказанными значениями, вычисленными по формуле ax+b.
* Цикл for проходит по всем наблюдениям, вычисляя разность для каждого значения и накапливая её в diff. Далее вычисляется градиент по b. Возвращает значение градиента по b.

1. Напишем функцию вычисления градиента для коэффициента при x.

def get\_gradient\_at\_a(x, y, b, a):

  N = len(x)

  diff = 0

  for i in range(N):

      x\_val = x[i]

      y\_val = y[i]

      diff += x\_val \* (y\_val - ((a \* x\_val) + b))

  a\_gradient = -(2/N) \* diff

  return a\_gradient

Функция get\_gradient\_at\_m предназначена для вычисления градиента функции потерь по отношению к коэффициенту наклона a. Давайте разберем, как она работает, и затем я предложу полный пример использования этой функции в контексте градиентного спуска для линейной регрессии.

Объяснение функции

Параметры:

* x: массив значений независимой переменной.
* y: массив значений зависимой переменной.
* b: текущее значение свободного члена (константы) в модели.
* a: текущее значение коэффициента наклона (углового коэффициента) в модели.

Вычисление градиента:

* N = len(x): определяет количество наблюдений.
* diff: переменная для накопления разностей между фактическими значениями y и предсказанными значениями, вычисленными по формуле a⋅x+b.
* Цикл for проходит по всем наблюдениям, вычисляя разность для каждого значения и накапливая её в diff. Далее вычисляем градиент по a. Функция выводит значение градиента и возвращает его.

1. Определим функцию одного шага градиентного спуска.

def step\_gradient (x,y, b\_current, a\_current, learning\_rate):

    b\_gradient = get\_gradient\_at\_b(x, y, b\_current, a\_current)

    m\_gradient = get\_gradient\_at\_m(x, y, b\_current, a\_current)

    b = b\_current - (learning\_rate \* b\_gradient)

    a = m\_current - (learning\_rate \* a\_gradient)

    return [b, a]

Функция step\_gradient предназначена для выполнения одного шага градиентного спуска в линейной регрессии. Она обновляет значения свободного члена b и коэффициента наклона m на основе вычисленных градиентов и заданной скорости обучения. Давайте разберем, как она работает, и затем я предложу полный пример использования этой функции в контексте градиентного спуска.

Объяснение функции

Параметры:

* x: массив значений независимой переменной.
* y: массив значений зависимой переменной.
* b\_current: текущее значение свободного члена (константы) в модели.
* a\_current: текущее значение коэффициента наклона (углового коэффициента) в модели.
* learning\_rate: скорость обучения, которая определяет, насколько сильно обновляются коэффициенты на каждом шаге.

Вычисление градиентов:

* b\_gradient: вычисляется с помощью функции get\_gradient\_at\_b, которая возвращает градиент по b.
* a\_gradient: вычисляется с помощью функции get\_gradient\_at\_a, которая возвращает градиент по m.

Обновление коэффициентов:

* Новое значение b вычисляется как текущее значение b минус произведение скорости обучения и градиента по b.
* Новое значение a вычисляется аналогично.
* Функция возвращает список, содержащий обновленные значения b и a.

1. Реализуем непосредственно градиентный спуск.

def gradient\_descent(x, y, learning\_rate, num\_iteration):

  b=0

  a=0

  for i in range(num\_iteration):

    b,a= step\_gradient(x, y, b, a, learning\_rate)

  return b,a

1. Рассчитаем для заданных данных:

b, a = gradient\_descent(X, y, 0.01, 1000)

1. Построим диаграмму рассеяния для исходных данных и график полученной прямой.

y = [m\*x + b for x in month]

plt.plot(month, revenue, "o")

plt.plot(month, y)

plt.show()

Изображение выглядит как снимок экрана, текст, линия, График

Автоматически созданное описание

**Задание 3**

1. Для данных из файла выберите один из x выполните прогнозирование с использованием написанного кода. Визуализируйте полученный результат.

**Линейная регрессия с использованием Scikit-learn**

Scikit-learn — это популярная библиотека для машинного обучения на Python, которая предоставляет простые и эффективные инструменты для анализа данных и построения моделей. Она поддерживает множество алгоритмов машинного обучения, включая классификацию, регрессию, кластеризацию и уменьшение размерности, а также предоставляет инструменты для предобработки данных, оценки моделей и выбора параметров.

Если у вас еще не установлена библиотека, вы можете установить ее с помощью pip:

pip install scikit-learn

При импортировании библиотеки обычно импортируют класс конкретного модуля. В случае линейной регрессии – LinearRegression.

from sklearn.linear\_model import LinearRegression

Далее рассмотрим реализацию того же примера с использованием scikit-learn.

1. Импортируем библиотеки и приведем наши данные к виду, релевантному для использования в scikit-learn:

import numpy as np

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LinearRegression

# Данные в виде списков

X = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]  # Независимая переменная

y = [52, 74, 79, 95, 115, 110, 129, 126, 147, 146, 156, 184]  # Зависимая переменная

# Преобразуем списки в массивы NumPy и изменяем форму X

X = np.array(X).reshape(-1, 1)  # Преобразуем в двумерный массив

y = np.array(y)

Классы, включенные в scikit-learn, содержат методы, которые умеют обрабатывать только массивы NumPy и DataFrame от Pandas. Поэтому исходные списки необходимо привести к типу np.array.

.reshape(-1, 1) изменяет форму (или размерность) массива. В данном случае -1 означает, что NumPy автоматически определит размерность этого измерения на основе общего количества элементов в массиве, а 1 указывает, что мы хотим, чтобы массив имел одно измерение с размерностью 1. Это преобразование необходимо, потому что Scikit-learn ожидает, что входные данные (независимая переменная) будут иметь форму двумерного массива, где каждая строка представляет собой отдельный пример, а каждый столбец — отдельную переменную. В случае линейной регрессии, если у вас только одна независимая переменная, вам нужно представить ее как двумерный массив с одной колонкой. Если у вас несколько независимых переменных, то reshape делать не нужно.

1. Создаем модель линейной регрессии:

# Создание модели линейной регрессии

model = LinearRegression()

# Обучение модели

model.fit(X, y)

1. Для построения прогноза используется метод predict:

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

**Задание 4**

1. Постройте модель линейной регрессии для данных из файла, загруженного в предыдущих заданиях. Сверьте коэффициенты регрессии, полученные с использованием statsmodels и библиотеки Scikit-learn.

Данные были загружены, обработаны, а пропуски (NaN) заменены средними значениями соответствующих столбцов.

Получены коэффициенты наклона и свободного члена:

* Scikit-learn: коэффициент наклона – 27.6127.6127.61, свободный член – 344.09344.09344.09.
* Statsmodels: коэффициент наклона – 27.6127.6127.61, свободный член – 344.09344.09344.09. Различия в коэффициентах минимальны, что подтверждает корректность реализации.

Оценка модели на тестовом наборе показала:

* Среднеквадратичная ошибка (MSE): 298696.07298696.07298696.07.
* Коэффициент детерминации (R²): 0.6740.6740.674, что указывает на удовлетворительную степень объяснения данных моделью.

**Метрики линейной регрессии в Scikit-learn**

Коэффициент детерминации *R*2 (R-квадрат) рассчитывается как доля вариации зависимой переменной, которая объясняется независимыми переменными в модели. В Scikit-learn для его вычисления используется следующая формула:

где (сумма квадратов остатков) — это сумма квадратов разностей между фактическими значениями ( и предсказанными значениями модели ():

(сумма квадратов отклонения от среднего) — это сумма квадратов разностей между фактическими значениями () и средним значением зависимой переменной ():

**Интерпретация**

* Если *R*2=1, это означает, что модель объясняет всю вариацию в данных.
* Если *R*2=0, это означает, что модель не объясняет никакой вариации, и предсказания равны среднему значению зависимой переменной.
* Отрицательные значения *R*2 могут возникать, если модель хуже, чем простое среднее значение.

Для использования этой метрики в scikit-learn необходимо импортировать соответствующий метод из модуля metrics:

from sklearn.metrics import r2\_score

Далее применить к соответствующим данным. Рассчитаем для наших данных:

y\_pred = model.predict(X)

r2 = r2\_score(y, y\_pred)

**Задание 5.**

Рассчитайте для построенных в scikit-learn ранее моделей R2.

Выводы

В рамках данной работы было рассмотрено:

1. несколько альтернативных подходов к построению модели линейной регрессии;
2. применение различных приемов python для реализации линейной регрессии;
3. различия между библиотеками, позволяющими осуществлять построение моделей линейной регрессии

**Самостоятельная работа**

1. Возьмите и загрузите csv файл с данными о недвижимости (Nedvig.csv)
2. Определите независимые и зависимую переменную.
3. Постройте диаграмму рассеяния для полученных данных.
4. Используя библиотеку statsmodel постройте модель линейной регрессии. Выведите в консоль информацию о модели. Какие вы можете сделать выводы?
5. Используя метод predict и файл Nedvig\_test.csv рассчитайте прогноз для данных.
6. Используя библиотеку Scikit-learn построить модель линейной регрессии.
7. Рассчитайте прогноз, используя метод predict и файл Nedvig\_test.csv.
8. Расcчитайте R2

Контрольные вопросы:

1. Что такое задача регрессии?
2. Что такое линейная регрессия?
3. Что такое множественная линейная регрессия?
4. В чем суть метода наименьших квадратов?
5. Ограничения метода наименьших квадратов.