Machine Learning thru Python

Машинне навчання - це процес, за яким комп'ютер навчається на основі вивчення даних та статистики а також програма (код), яка аналізує дані та навчається передбачати результат.

Набір даних

У світі комп'ютера набір даних - це будь-яка колекція даних. Це може бути все, починаючи від масиву до повної бази даних.

Приклад масиву:

```
[99,86,87,88,111,86,103,87,94,78,77,85,86]
```

Чи датафрейму

Carname	Color	Age	Speed	AutoPass
BMW	red	5	99	Y
Volvo	black	7	86	Y
VW	gray	8	87	N
VW	white	7	88	Y
Ford	white	2	111	Y
VW	white	17	86	Y
Tesla	red	2	103	Y
BMW	black	9	87	Y
Volvo	gray	4	94	N
Ford	white	11	78	N
Toyota	gray	12	77	N
VW	white	9	85	N
Toyota	blue	6	86	Y

Розглядаючи масив, ми можемо вважати, що середнє значення, ймовірно, становить близько 80 або 90, і ми також можемо визначити найвище і найнижче значення, але що ще ми можемо зробити?

А розглядаючи базу даних, ми бачимо, що найпопулярніший колір - білий, а найдавніша машина - 17 років, але що, якщо ми могли б передбачити, чи є у машини автоматичний пропуск, просто глядаючи на інші значення?

Саме для цього існує машинне навчання! Аналіз даних та передбачення результату!

Типи даних

Для аналізу даних важливо знати, з яким типом даних ми працюємо.

Ми можемо розділити типи даних на три основні категорії:

- Числові - це числа і можуть бути розділені на дві числові категорії:
    - Дискретні дані - це числа, обмежені цілими числами. Приклад: Кількість машин, що проїжджають.
    - Неперервні дані - це числа безкінечного значення. Приклад: Ціна товару або його розмір.
- Категоріальні - це значення, які не можна порівняти одне з одним. Приклад: Значення кольору або будь-які значення так/ні.
- Упорядковані - схожі на категоріальні дані, але їх можна порівнювати одне з одним. Приклад: Оцінки у школі, де A краще за B і так далі.

Знання типу даних джерела даних дозволить вам визначити, яку техніку використовувати при їх аналізі.

Середнє значення (Mean), медіана (Median) та мода (Mode)

Що ми можемо дізнатися, дивлячись на групу чисел?

У машинному навчанні (і в математиці) часто цікавлять нас три значення:

- Середнє значення (Mean) - середнє значення
- Медіана (Median) - значення по середині

- Мода (Mode) - найбільш поширене значення

> ❗ Приклад: Ми зареєстрували швидкість 13 автомобілів:
> speed = [99,86,87,88,111,86,103,87,94,78,77,85,86]
> Яке середнє, середнє значення або найпоширеніше значення швидкості?

Середнє значення (Mean)

Для обчислення середнього значення потрібно знайти суму всіх значень і поділити суму на кількість значень:

```
(99+86+87+88+111+86+103+87+94+78+77+85+86) / 13 = 89.77
```

Модуль NumPy має метод для цього. Дізнайтеся більше про модуль NumPy в нашому посібнику з NumPy.

```
import numpy
speed = [99,86,87,88,111,86,103,87,94,78,77,85,86]
x = numpy.mean(speed)
print(x)
```

Медіана (Median)

Медіана (Median) - це значення в середині після впорядкування всіх значень:

77, 78, 85, 86, 86, 86, 87, 87, 88, 94, 99, 103, 111

Важливо впорядкувати числа перед знаходженням медіани.

Модуль NumPy має метод для цього:

Приклад: Використайте метод median() з модуля NumPy для знаходження середнього значення:

```
import numpy
speed = [99,86,87,88,111,86,103,87,94,78,77,85,86]
x = numpy.median(speed)
print(x)
```

Якщо в середині є два числа, поділіть суму цих чисел на два.

77, 78, 85, 86, 86, 86, 87, 87,94, 98, 99, 103

(86 + 87) / 2 = 86.5

Мода (Mode)

Мода (Mode) - це значення, яке зустрічається найбільшу кількість разів:

99, 86, 87, 88, 111, 86, 103, 87, 94, 78, 77, 85, 86

Приклад: Використовуйте метод mode() з модуля SciPy для знаходження числа, яке зустрічається найбільшу кількість разів:

```
from scipy import stats
speed = [99,86,87,88,111,86,103,87,94,78,77,85,86]
x = stats.mode(speed)
print(x)
```

Середньоквадратичне відхилення (Standard Deviation)

Середньоквадратичне відхилення є числом, що описує, наскільки розподілені значення.

Низьке середньоквадратичне відхилення означає, що більшість чисел знаходяться близько до середнього значення.

Високе середньоквадратичне відхилення означає, що значення розподілені в ширшому діапазоні.

Приклад: На цей раз ми зареєстрували швидкість 7 автомобілів:

❗ speed = [86,87,88,86,87,85,86]

Середньоквадратичне відхилення дорівнює:

> **0.9**

Це означає, що більшість значень знаходяться в межах 0.9 від середнього значення, яке становить 86.4.

Давайте зробимо те саме з вибіркою чисел з більш широким діапазоном:

❗ speed = [32,111,138,28,59,77,97]

Середньоквадратичне відхилення дорівнює:

> **37.85**

Це означає, що більшість значень знаходяться в межах 37.85 від середнього значення, яке становить 77.4.

Як бачите, більше середньоквадратичне відхилення вказує на те, що значення розподілені в ширшому діапазоні.

Модуль NumPy має метод для обчислення середньоквадратичного відхилення:

❗ Приклад: Використайте метод std() з модуля NumPy для знаходження середньоквадратичного відхилення:

```
import numpy
speed = [86,87,88,86,87,85,86]
x = numpy.std(speed)
print(x)
```

```
import numpy
speed = [32,111,138,28,59,77,97]
```

```
x = numpy.std(speed)
print(x)
```

Варіантність (Variance)

Варіантність - це інше число, яке вказує, наскільки розподілені значення.

На самом ділі, якщо взяти квадратний корінь від варіантності, то отримає кожне варіантне значення.І навпаки, якщо помножити середньоквадратичне відхилення на себе, отримаєте варіантність.

Для обчислення варіантності потрібно виконати наступні кроки:

- Знайти середнє значення:

(32+111+138+28+59+77+97) / 7 = 77.4

- Для кожного значення знайдіть різницю від середнього:

32 - 77.4 = -45.4
111 - 77.4 = 33.6
138 - 77.4 = 60.6
28 - 77.4 = -49.4
59 - 77.4 = -18.4
77 - 77.4 = - 0.4
97 - 77.4 = 19.6

- Для кожної різниці знайдіть квадрат значення:

(-45.4)2 = 2061.16
(33.6)2 = 1128.96
(60.6)2 = 3672.36
(-49.4)2 = 2440.36
(-18.4)2 = 338.56
(- 0.4)2 = 0.16
(19.6)2 = 384.16

- Варіантність - це середнє значення цих квадратних різниць:

(2061.16+1128.96+3672.36+2440.36+338.56+0.16+384.16) / 7 = 1432.2

На щастя, у модулі NumPy є метод для обчислення варіантності:

```
import numpy
speed = [32,111,138,28,59,77,97]
x = numpy.var(speed)
print(x)
```

Варіантність є одним з показників, які допомагають нам зрозуміти, наскільки розподіл значень даних розсіяний. Вона використовується для вимірювання розбіжності або розсіювання значень навколо їх середнього значення. Основна ідея полягає в тому, що варіантність вказує, наскільки великі різниці між окремими значеннями та середнім значенням набору даних.

Варіантність використовується в багатьох галузях, включаючи статистику, економіку, науку про дані та машинне навчання. Вона дозволяє нам порівнювати розподіли даних, визначати, наскільки стійкими є результати вимірювань і використовувати її для прийняття рішень, проведення прогнозування та виявлення аномалій в даних.

Наприклад, варіантність може бути корисною для порівняння дисперсії результатів двох груп експериментів, визначення ступеня ризику в інвестиційному портфелі або виявлення змін у стабільності процесу виробництва.

Отже, розрахунок варіантності допомагає нам отримати кількісну міру розсіювання даних, що є важливим інструментом аналізу та розуміння набору даних.

Середньоквадратичне відхилення

Як ми вже з'ясували, формула для знаходження середньоквадратичного відхилення - це квадратний корінь від варіантності:

> √1432.25 = 37.85

Або, як у прикладі вище, використовуйте NumPy для обчислення середньоквадратичного відхилення:

Приклад: Використайте метод std() з модуля NumPy для знаходження середньоквадратичного відхилення:

```
import numpy
speed = [32,111,138,28,59,77 ,97]
x = numpy.std(speed)
print(x)
```

Середньоквадратичне відхилення часто позначається символом Сигма: σ

Варіантність часто позначається символом Сигма квадрат: σ^2

Перцентиль

Перцентилі використовуються в статистиці для надання вам числа, яке описує значення, які менші за вказаний відсоток значень.

Приклад: Давайте припустимо, що у нас є масив з віками всіх людей, які живуть на вулиці.

ages = [5,31,43,48,50,41,7,11,15,39,80,82,32,2,8,6,25,36,27,61,31]

Який є 75-й перцентиль? Відповідь - 43, що означає, що 75% людей мають вік 43 або менше.

Модуль NumPy має метод для знаходження вказаного перцентилю:

Використовуйте метод percentile() з модуля NumPy для знаходження перцентилей:

```
import numpy
ages = [5,31,43,48,50,41,7,11,15,39,80,82,32,2,8,6,25,36,2
7,61,31]
x = numpy.percentile(ages, 75)
print(x)
```

Розподіл даних

(Data Distribution) відноситься до способу розподілу або розповсюдження даних по різних значеннях або категоріях. Він надає уявлення про шаблони, частоти та характеристики даних. Розуміння розподілу даних є ключовим для аналізу даних та прийняття обґрунтованих рішень.

Існує кілька типів розподілів даних, і серед найпоширеніших можна виділити:

1. Нормальний розподіл: Іноді його називають гауссівським розподілом. Він має симетричну форму дзвону, де дані групуються навколо середнього значення з однаковою ймовірністю з обох сторін.
2. Розподіл з викривленням: Розподіл називається викривленим, коли дані не розподіляються симетрично. Він може бути або позитивно викривленим (викривлення вправо), або негативно викривленим (викривлення вліво), залежно від напрямку довгого хвоста.
3. Рівномірний розподіл: В рівномірному розподілі всі значення мають однакову ймовірність, і дані рівномірно розподілені по всьому діапазону.
4. Двовершинний(бімодальний) розподіл: Двовершинний розподіл виникає, коли в даних є дві відмінні вершини або моди.
5. Багатовершинний розподіл: Подібно до двовершинного розподілу, багатовершинний розподіл має більше двох відмінних вершин або мод.
6. Експоненційний розподіл: Експоненційний розподіл описує час між подіями в процесі Пуассона. Він характеризується швидким спадом після початкового піку.

Розуміння розподілу даних допомагає виявляти закономірності, викиди і робити прогнози або зробити висновки про популяцію, з якої зібрані дані. Статистичні показники, такі як середнє значення, медіана, мода, стандартне відхилення та перцентилі, використовуються для опису та аналізу розподілу даних.

Аналіз розподілу даних є основним кроком у різних галузях, включаючи статистику, науку про дані та машинне навчання, оскільки він є основою для моделювання та здійснення точних прогнозів на основі наявних даних.

Нормальний розподіл (також відомий як гауссівський розподіл) є одним з найважливіших та найпоширеніших типів розподілу даних. У нормальному розподілі дані симетрично розподілені навколо середнього значення, і є деякі математичні властивості, які визначають його форму.

У Python, модуль SciPy має функцію norm для моделювання нормального розподілу. Вона дозволяє створювати вибірки з нормально розподіленими даними та проводити аналіз.

Ось приклад використання модуля SciPy для створення вибірки з нормально розподіленими даними та побудови графіка:

```
pythonCopy code
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm

# Задаємо параметри нормального розподілу
mu = 0  # Середнє значення
sigma = 1  # Стандартне відхилення

# Генеруємо 1000 випадкових значень з нормального розподілу
data = np.random.normal(mu, sigma, 1000)

# Побудова гістограми вибірки
plt.hist(data, bins=30, density=True, alpha=0.5, color='sky
blue')

# Побудова графіка нормального розподілу
x = np.linspace(-4, 4, 100)
plt.plot(x, norm.pdf(x, mu, sigma), 'r-', lw=2)

# Налаштування вісей та заголовку
plt.xlabel('Значення')
plt.ylabel('Частота')
plt.title('Нормальний розподіл')

# Відображення графіка
plt.show()
```

У цьому прикладі ми генеруємо 1000 випадкових значень з нормального розподілу зі середнім значенням 0 і стандартним відхиленням 1. Потім ми

побудовуємо гістограму цих значень і порівнюємо з теоретичним графіком нормального розподілу за допомогою функції `norm.pdf()`.

Розподіл з викривленням (також відомий як асиметричний розподіл) відрізняється від нормального розподілу тим, що його форма несиметрична. Він може бути скошеним вліво або вправо.

У Python, модуль SciPy має кілька функцій для моделювання розподілів з викривленням, наприклад, бета-розподіл, гамма-розподіл, експоненційний розподіл та інші. Кожен з цих розподілів має свої особливості та параметри, що впливають на його форму.

Ось приклад використання модуля SciPy для моделювання розподілу з викривленням (бета-розподіл) та побудови графіка:

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import beta

# Задаємо параметри бета-розподілу
alpha = 2  # Параметр форми 1
beta = 3  # Параметр форми 2

# Генеруємо 1000 випадкових значень з бета-розподілу
data = np.random.beta(alpha, beta, 1000)

# Побудова гістограми вибірки
plt.hist(data, bins=30, density=True, alpha=0.5, color='sky
blue')

# Побудова графіка бета-розподілу
x = np.linspace(0, 1, 100)
plt.plot(x, beta.pdf(x, alpha, beta), 'r-', lw=2)

# Налаштування вісей та заголовку
plt.xlabel('Значення')
plt.ylabel('Частота')
plt.title('Розподіл з викривленням (бета-розподіл)')
```

```
# Відображення графіка
plt.show()
```

У цьому прикладі ми генеруємо 1000 випадкових значень з бета-розподілу з параметрами форми 1 і 2. Потім ми побудовуємо гістограму цих значень і порівнюємо з теоретичним графіком бета-розподілу за допомогою функції `beta.pdf()`.

Цей приклад демонструє, як можна моделювати та візуалізРозподіл з викривленням (асиметричний розподіл) відрізняється від нормального розподілу тим, що його форма несиметрична. Він може бути скошеним вліво або вправо.

Рівномірний розподіл - це тип ймовірнісного розподілу, де кожне значення в заданому інтервалі має однакову ймовірність випадання.

У Python, модуль NumPy має функцію `numpy.random.uniform()`, яка дозволяє генерувати випадкові значення з рівномірного розподілу.

Ось приклад використання функції `numpy.random.uniform()` для генерації випадкових значень з рівномірного розподілу та побудови графіка:

```
import numpy as np
import matplotlib.pyplot as plt

# Задаємо інтервал рівномірного розподілу
a = 0  # Нижня межа
b = 10  # Верхня межа

# Генеруємо 1000 випадкових значень з рівномірного розподіл
у
data = np.random.uniform(a, b, 1000)

# Побудова гістограми вибірки
plt.hist(data, bins=30, density=True, alpha=0.5, color='sky
blue')

# Налаштування вісей та заголовку
plt.xlabel('Значення')
plt.ylabel('Частота')
```

```
plt.title('Рівномірний розподіл')

# Відображення графіка
plt.show()
```

У цьому прикладі ми генеруємо 1000 випадкових значень з рівномірного розподілу в інтервалі від 0 до 10 за допомогою функції `numpy.random.uniform()`. Потім ми побудовуємо гістограму цих значень для візуалізації розподілу.

Цей приклад демонструє, як можна генерувати та візуалізувати випадкові значення з рівномірного розподілу в Python.

Двовершинний розподіл, також відомий як бімодальний розподіл, є типом ймовірнісного розподілу, у якому дані мають дві видимі вершини або піки. Це означає, що в розподілі є дві преобладаючі групи або підгрупи значень, які можуть мати різну форму, розташування або дисперсію.

В Python, ви можете створити приклад двовершинного розподілу за допомогою модуля NumPy і matplotlib. Одним із способів досягнення двовершинності є комбінування двох або більше розподілів, таких як нормальний розподіл.

Ось приклад використання модулів NumPy і matplotlib для створення графіка з двома вершинами:

```
import numpy as np
import matplotlib.pyplot as plt

# Створення двох нормальних розподілів
mu1, sigma1 = 0, 1
mu2, sigma2 = 3, 0.5
data1 = np.random.normal(mu1, sigma1, 1000)
data2 = np.random.normal(mu2, sigma2, 1000)

# Об'єднання двох розподілів
data = np.concatenate((data1, data2))

# Побудова гістограми
plt.hist(data, bins=30, density=True, alpha=0.5, color='sky
blue')
```

```
# Налаштування вісей та заголовку
plt.xlabel('Значення')
plt.ylabel('Частота')
plt.title('Двовершинний розподіл')

# Відображення графіка
plt.show()
```

У цьому прикладі ми створюємо два нормальних розподіли з різними параметрами `mu` і `sigma` за допомогою функції `numpy.random.normal()`. Потім ми об'єднуємо ці два розподіли за допомогою функції `numpy.concatenate()`. Нарешті, ми побудовуємо гістограму цих даних для візуалізації двовершинного розподілу.

Цей приклад показує, як створити і візуалізувати графік з двома вершинами, що ві

дповідає двовершинному або бімодальному розподілу в Python. Зверніть увагу, що у прикладі використовуються нормальні розподіли для створення двох вершин, але ви можете експериментувати з іншими розподілами та параметрами, щоб отримати бажаний двовершинний розподіл.

Експоненційний розподіл є одним з найпоширеніших розподілів у статистиці та ймовірнісних розрахунках. Він використовується для моделювання часу між подіями, які відбуваються незалежно одна від одної з певною середньою інтенсивністю.

Експоненційний розподіл має єдиний параметр, який називається параметром швидкості або інверсією середнього часу між подіями. Цей параметр позначається як λ (лямбда). Ймовірність того, що подія відбудеться протягом певного інтервалу часу, обчислюється за допомогою функції щільності ймовірності експоненційного розподілу.

У Python, модуль `numpy` містить функції для генерації вибірок з експоненційного розподілу та обчислення ймовірностей. Ось приклад використання `numpy` для генерації випадкових вибірок з експоненційного розподілу та побудови графіку функції щільності ймовірності:

```
import numpy as np
import matplotlib.pyplot as plt
```

```
# Параметр швидкості (лямбда)
lambda_param = 0.5

# Генерування 1000 випадкових значень з експоненційного роз
поділу
data = np.random.exponential(scale=1/lambda_param, size=100
0)

# Побудова графіку функції щільності ймовірності
x = np.linspace(0, 10, 100)
pdf = lambda_param * np.exp(-lambda_param * x)
plt.plot(x, pdf, 'r-', lw=2, label='PDF')

# Побудова гістограми випадкових значень
plt.hist(data, bins=30, density=True, alpha=0.5, color='sky
blue', label='Гістограма')

# Налаштування вісей та заголовку
plt.xlabel('Значення')
plt.ylabel('Щільність ймовірності')
plt.title('Експоненційий розподіл')
plt.legend()

# Відображення графіку
plt.show()
```

У цьому прикладі ми використовуємо `numpy.random.exponential` для генерації 1000 випадкових значень з експоненційного розподілу з заданим параметром швидкості. Потім ми побудовуємо графік функції щільності ймовірності (PDF) та гістограму випадкових значень.

Цей приклад можна модифікувати, змінюючи значення параметра швидкості `lambda_param` або кількість та діапазон випадкових значень, щоб відобразити різні форми та властивості експоненційного розподілу.

Багатовершинний розподіл, також відомий як бімодальний розподіл, відображається, коли набір даних має дві чітко виражені вершини або піки. Це означає, що дані розділені на дві групи або підпопуляції з різними значеннями.

Багатовершинний розподіл може виникати з різних причин, наприклад, коли ви досліджуєте показник, який має два режими чи дві різні умови, які впливають на дані. Наприклад, якщо ви досліджуєте розподіл зростання рослин, може виникнути багатовершинний розподіл, якщо у вас є дві різні сорти рослин з різними швидкостями зростання.

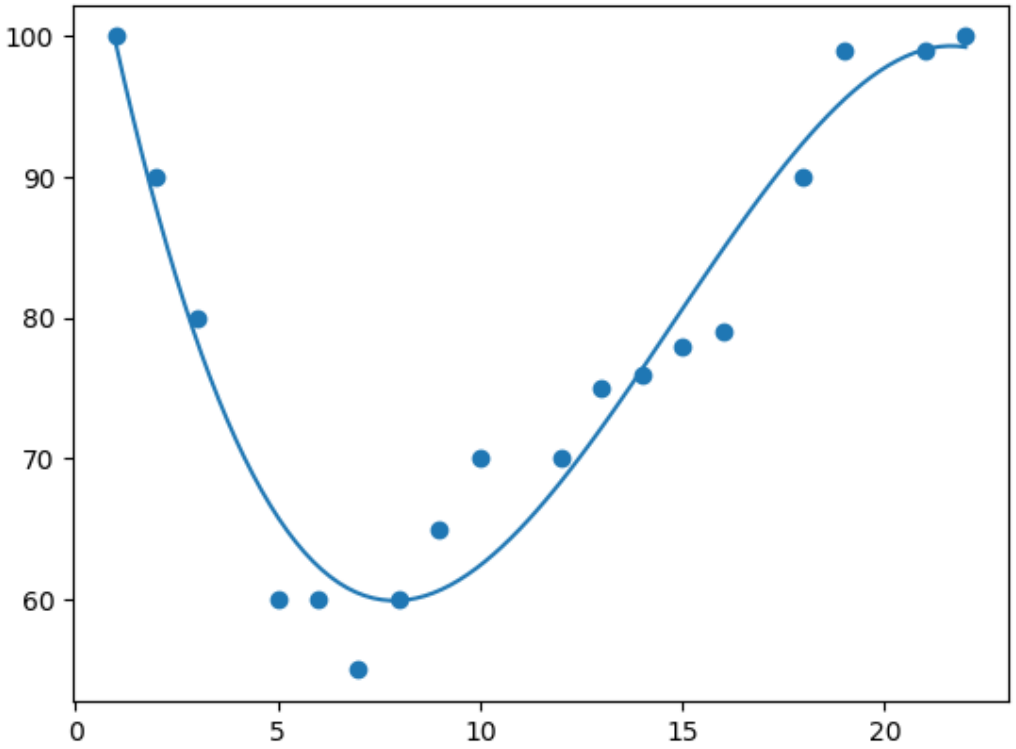
У Python ви можете моделювати багатовершинний розподіл, використовуючи різні статистичні методи та бібліотеки, такі як NumPy та SciPy. Вам можуть знадобитися додаткові дані або параметри для виконання аналізу та моделювання багатовершинного розподілу залежно від вашого конкретного сценарію.

Якщо у вас є конкретні дані або більш докладні вимоги щодо багатовершинного розподілу, будь ласка, надайте більше відомостей, і я з радістю надам вам більш конкретну допомогу.

Поліноміальна регресія

Якщо ваші точки даних очевидно не підходять для лінійної регресії (пряма лінія через всі точки даних), поліноміальна регресія може бути ідеальним варіантом.

Поліноміальна регресія, подібно до лінійної регресії, використовує взаємозв'язок між змінними x та y для знаходження найкращого способу провести лінію через точки даних.



Як це працює?

Python має методи для знаходження взаємозв'язку між точками даних та побудови лінії поліноміальної регресії. Ми покажемо вам, як використовувати ці методи, не занурюючись у математичні формули.

У наведеному нижче прикладі ми зареєстрували 18 автомобілів, як вони проходили певний пункт збору плати.

Ми зареєстрували швидкість автомобіля та час доби (годину), коли відбувся проїзд.

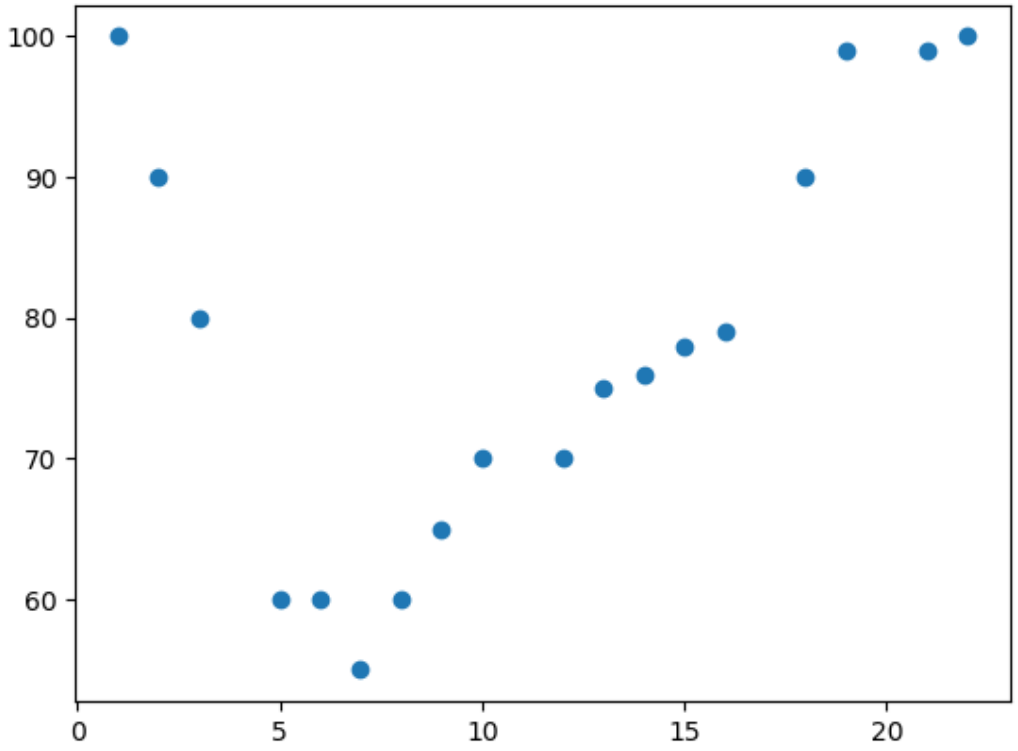
Ось графік, де вісь x представляє години дня, а вісь y представляє швидкість:

Почніть з побудови діаграми розсіювання:

```
import matplotlib.pyplot as plt
x = [1,2,3,5,6,7,8,9,10,12,13,14,15,16,18,19,21,22]
y = [100,90,80,60,60,55,60,65,70,70,75,76,78,79,90,99,99,10
0]
```

```
plt.scatter(x, y)
plt.show()
```

Результат:

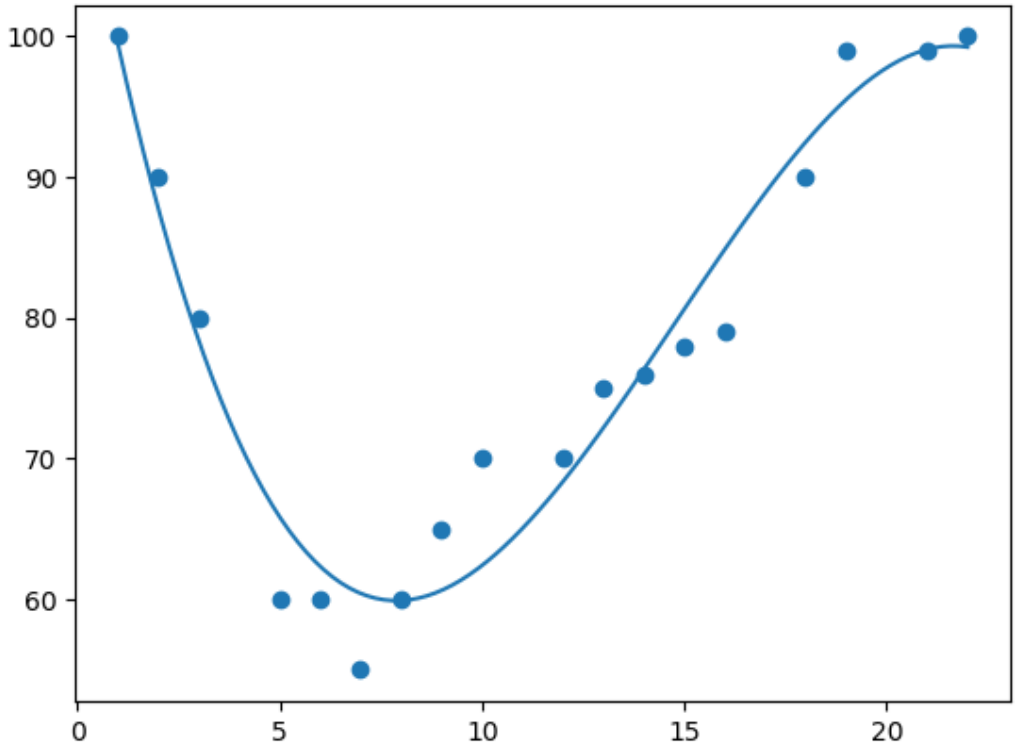


Імпортуйте numpy та matplotlib, а потім побудуйте лінію поліноміальної регресії:

```
import numpy
import matplotlib.pyplot as plt
x = [1,2,3,5,6,7,8,9,10,12,13,14,15,16,18,19,21,22]
y = [100,90,80,60,60,55,60,65,70,70,75,76,78,79,90,99,99,10
0]
mymodel = numpy.poly1d(numpy.polyfit(x, y, 3))
myline = numpy.linspace(1, 22, 100)
plt.scatter(x, y)
```

```
plt.plot(myline, mymodel(myline))
plt.show()
```

Результат:



Роз'яснення прикладу

```
import numpy
import matplotlib.pyplot as plt
```

Створіть масиви, які представляють значення вісей x та y:

```
x = [1,2,3,5,6,7,8,9,10,12,13,14,15,16,18,19,21,22]
y = [100,90,80,60,60,55,60,65,70,70,75,76,78,79,90,99,99,10
0]
```

NumPy має метод, який дозволяє нам побудувати поліноміальну модель:

```
mymodel = numpy.poly1d(numpy.polyfit(x, y, 3))
```

Потім вказуйте, як буде відображатись лінія, починаючи з позиції 1 і закінчуючи позицією 22:

```
myline = numpy.linspace(1, 22, 100)
```

Намалюйте вихідну діаграму розсіювання:

```
plt.scatter(x, y)
```

Намалюйте лінію поліноміальної регресії:

```
plt.plot(myline, mymodel(myline))
```

Відобразіть діаграму:

```
plt.show()
```

Коефіцієнт детермінації (R-Squared)

Важливо знати, наскільки добре відношення між значеннями вісей x та y. Якщо немає взаємозв'язку, поліноміальна регресія не може бути використана для прогнозу.

Відношення вимірюється за допомогою значення, яке називається коефіцієнтом детермінації (R-squared).

Значення коефіцієнта детермінації розташовується в діапазоні від 0 до 1, де 0 означає відсутність взаємозв'язку, а 1 означає повний взаємозв'язок.

Python та модуль Sklearn обчислять це значення для вас, все, що потрібно зробити, це надати йому масив x та y:

> ❗ Приклад
> Наскільки добре мої дані підходять для поліноміальної регресії?

```
import numpy
from sklearn.metrics import r2_score
x = [1,2,3,5,6,7,8,9,10,12,13,14,15,16,18,19,21,22]
y = [100,90,80,60,60,55,60,65,70,70,75,76,78,79,90,99,99,10
0]
mymodel = numpy.poly1d(numpy.polyfit(x, y, 3))
print(r2_score(y, mymodel(x)))
```

> Результат 0.94 показує, що є дуже хороше відношення, і ми можемо використовувати поліноміальну регресію для майбутніх прогнозів.

Прогнозування майбутніх значень

Тепер ми можемо використовувати отриману інформацію для прогнозування майбутніх значень.

> ! Давайте спробуємо прогнозувати швидкість автомобіля, який проїжджатиме пункт збору плат близько о 17:00 годині:

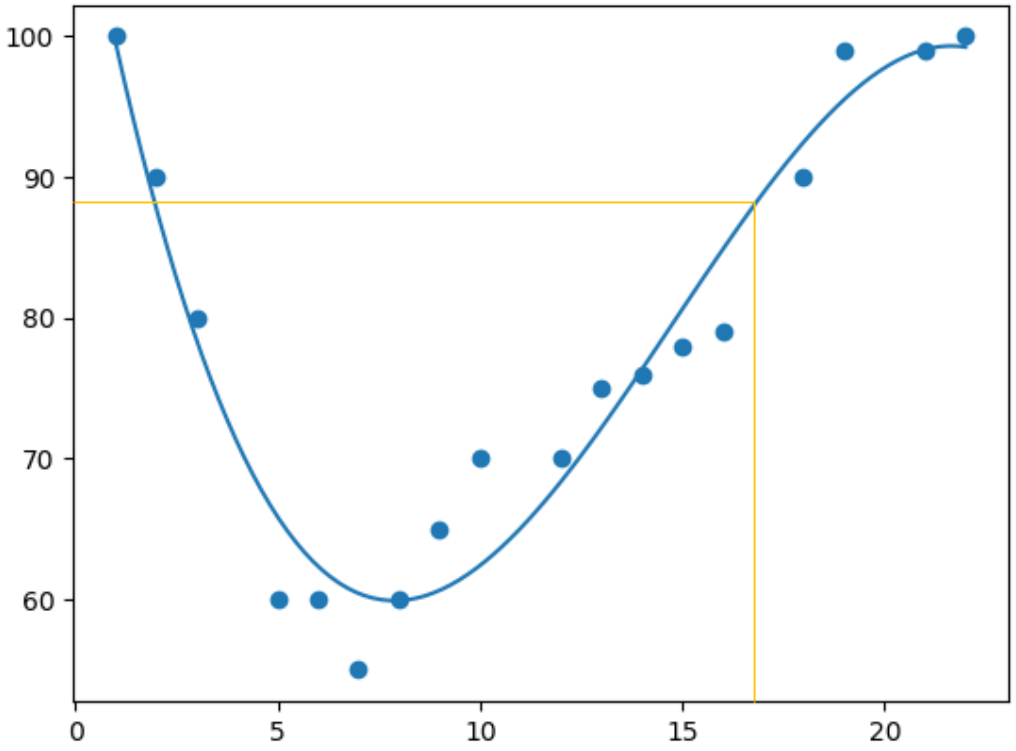
Для цього нам потрібний той самий масив mymodel з попереднього прикладу:

mymodel = numpy.poly1d(numpy.polyfit(x, y, 3))

Прогнозуйте швидкість автомобіля, який проїжджає о 17:00:

```
import numpy
from sklearn.metrics import r2_score
x = [1,2,3,5,6,7,8,9,10,12,13,14,15,16,18,19,21,22]
y = [100,90,80,60,60,55,60,65,70,70,75,76,78,79,90,99,99,10
0]
mymodel = numpy.poly1d(numpy.polyfit(x, y, 3))
speed = mymodel(17)
print(speed)
```

У цьому прикладі передбачено швидкість 88.87, що ми також можемо прочитати з діаграми:



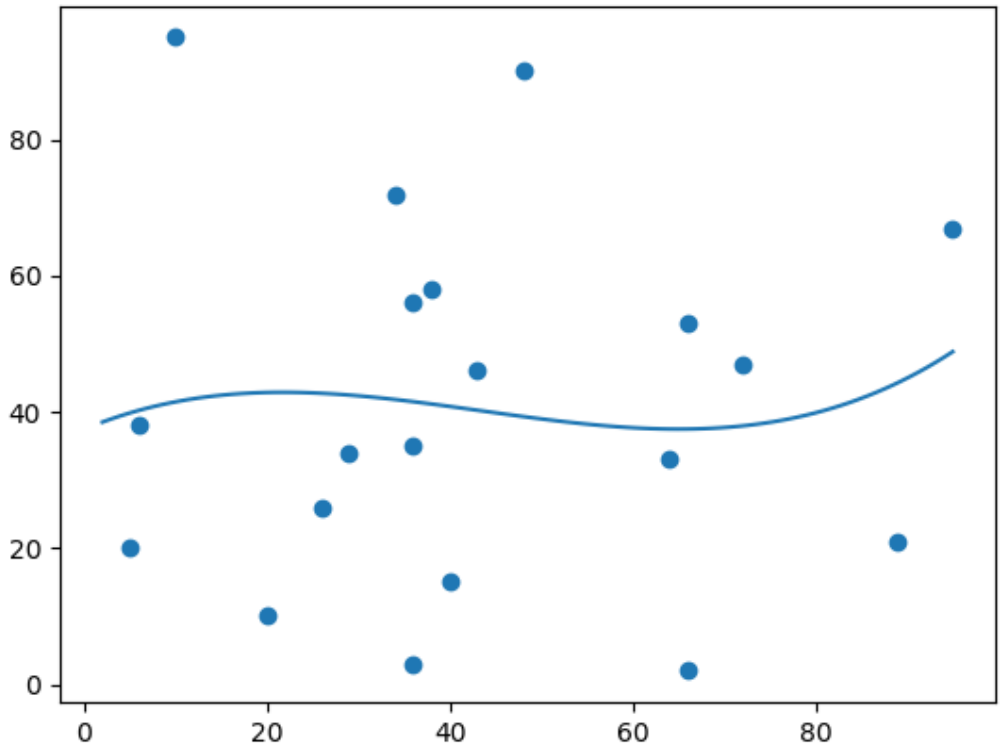
Погана відповідність?

Давайте створимо приклад, коли поліноміальна регресія не буде найкращим методом для прогнозування майбутніх значень.

> ! Ці значення для вісей x та y повинні призвести до дуже поганого відповідності для поліноміальної регресії:

```
import numpy
import matplotlib.pyplot as plt
x = [89,43,36,36,95,10,66,34,38,20,26,29,48,64,6,5,36,66,7
2,40]
y = [21,46,3,35,67,95,53,72,58,10,26,34,90,33,38,20,56,2,4
7,15]
```

```
mymodel = numpy.poly1d(numpy.polyfit(x, y, 3))
myline = numpy.linspace(2, 95, 100)
plt.scatter(x, y)
plt.plot(myline, mymodel(myline))
plt.show()
```



Щось взагалі не те. Якщо отримали такий вигляд, то у цьому випадку треба звернутись до коефіцієнта детермінації.

Ви повинні отримати дуже низьке значення коефіцієнта детермінації.

```
import numpy
from sklearn.metrics import r2_score
x = [89,43,36,36,95,10,66,34,38,20,26,29,48,64,6,5,36,66,7
2,40]
y = [21,46,3,35,67,95,53,72,58,10,26,34,90,33,38,20,56,2,4
7,15]
```

```
mymodel = numpy.poly1d(numpy.polyfit(x, y, 3))
print(r2_score(y, mymodel(x)))
```

Результат: 0.00995 вказує на дуже слабкий зв'язок і свідчить про те, що цей набір даних не підходить для поліноміальної регресії.

Множинна регресія

Множинна регресія схожа на лінійну регресію, але з більш ніж одним незалежним значенням, що означає, що ми намагаємося передбачити значення на основі двох або більше змінних.

Дивіться наступний набір даних, він містить деяку інформацію про автомобілі.

Car	Model	Volume	Weight	CO2
Toyota	Aygo	1000	790	99
Mitsubishi	Space Star	1200	1160	95
Skoda	Citigo	1000	929	95
Fiat	500	900	865	90
Mini	Cooper	1500	1140	105
VW	Up!	1000	929	105
Skoda	Fabia	1400	1109	90
Mercedes	A-Class	1500	1365	92
Ford	Fiesta	1500	1112	98
Audi	A1	1600	1150	99
Hyundai	I20	1100	980	99
Suzuki	Swift	1300	990	101
Ford	Fiesta	1000	1112	99
Honda	Civic	1600	1252	94
Hundai	I30	1600	1326	97
Opel	Astra	1600	1330	97
BMW	1	1600	1365	99

Mazda	3	2200	1280	104
Skoda	Rapid	1600	1119	104
Ford	Focus	2000	1328	105
Ford	Mondeo	1600	1584	94
Opel	Insignia	2000	1428	99
Mercedes	C-Class	2100	1365	99
Skoda	Octavia	1600	1415	99
Volvo	S60	2000	1415	99
Mercedes	CLA	1500	1465	102
Audi	A4	2000	1490	104
Audi	A6	2000	1725	114
Volvo	V70	1600	1523	109
BMW	5	2000	1705	114
Mercedes	E-Class	2100	1605	115
Volvo	XC70	2000	1746	117
Ford	B-Max	1600	1235	104
BMW	2	1600	1390	108
Opel	Zafira	1600	1405	109
Mercedes	SLK	2500	1395	120

Ми можем передбачити викиди CO2 автомобіля на основі об'єму двигуна, але за допомогою множинної регресії ми можемо додати більше змінних, наприклад, вагу автомобіля, щоб зробити прогноз більш точним.

Як це працює?

У Python ми маємо модулі, які виконають цю роботу за нас. Почнемо з імпорту модуля Pandas.

```
import pandas
```

Детальніше про модуль Pandas можна дізнатися в нашому посібнику Pandas Tutorial.

Модуль Pandas дозволяє нам зчитувати CSV-файли і повертати об'єкт DataFrame.

Файл призначений лише для тестування, його можна завантажити тут: data.csv

```
df = pandas.read_csv("data.csv")
```

Потім створіть список незалежних значень і назвіть його X.

Розмістіть залежні значення в змінній, названій y.

```
X = df[['Weight', 'Volume']]
y = df['CO2']
```

Порада: Зазвичай список незалежних значень називають з великої літери X, а список залежних значень - з малої літери y.

Ми будемо використовувати деякі методи з модуля sklearn, тому нам також потрібно імпортувати цей модуль:

```
from sklearn import linear_model
```

З модуля sklearn ми будемо використовувати метод LinearRegression() для створення об'єкта лінійної регресії.

У цього об'єкта є метод fit(), який приймає незалежні і залежні значення як параметри та заповнює об'єкт регресії даними, які описують залежність:

```
regr = linear_model.LinearRegression()
regr.fit(X, y)
```

Тепер у нас є об'єкт регресії, готовий передбачати значення CO2 на основі ваги та об'єму автомобіля:

```
# Передбачити викиди CO2 автомобіля, де вага становить 2300
кг, а об'єм - 1300 см3:
predictedCO2 = regr.predict([[2300, 1300]])
```

Подивіться на цілий приклад у дії:

```
import pandas
from sklearn import linear_model

df = pandas.read_csv("data.csv")

X = df[['Weight', 'Volume']]
y = df['CO2']

regr = linear_model.LinearRegression()
regr.fit(X, y)

# передбачити викиди CO2 автомобіля, де вага становить 2300
кг, а об'єм - 1300 см3:
predictedCO2 = regr.predict([[2300, 1300]])

print(predictedCO2)
```

107.2087328

Ми передбачили, що автомобіль з двигуном об'ємом 1,3 літра та вагою 2300 кг буде викидати приблизно 107 грамів CO2 на кожен пройдений кілометр.

Коефіцієнт

Коефіцієнт - це фактор, що описує залежність від невідомої змінної.

Наприклад: якщо x - це змінна, то 2x означає x, помножене на два. Тут x - це невідома змінна, а число 2 - це коефіцієнт.

У цьому випадку ми можемо запитати значення коефіцієнта для ваги щодо CO2, а також для об'єму щодо CO2. Відповідь або відповіді, які ми отримаємо, розкажуть нам, що станеться, якщо ми збільшимо або зменшимо одне з незалежних значень.

Вивести значення коефіцієнтів об'єкта регресії:

```
import pandas
from sklearn import linear_model

df = pandas.read_csv("data.csv")

X = df[['Weight', 'Volume']]
y = df['CO2']

regr = linear_model.LinearRegression()
regr.fit(X, y)

print(regr.coef_)
```

Результат:
[0.00755095 0.00780526]
Масив результатів представляє значення коефіцієнтів для ваги та об'єму.

Вага: 0.00755095
Об'єм: 0.00780526

Ці значення говорять нам, що якщо вага збільшиться на 1 кг, викиди CO2 збільшаться на 0.00755095 г.

А якщо об'єм двигуна (об'єм) збільшиться на 1 см3, викиди CO2 збільшаться на 0.00780526 г.

Мені здається, що це розумна припущення, але давайте перевіримо це!

Ми вже передбачили, що якщо автомобіль з двигуном об'ємом 1300 см3 має вагу 2300 кг, то викиди CO2 становитимуть при

близно 107 г.

А що, якщо ми збільшимо вагу на 1000 кг?

Приклад
Скопіюйте попередній приклад, але змініть вагу з 2300 на 3300:

```
import pandas
from sklearn import linear_model

df = pandas.read_csv("data.csv")
```

```
X = df[['Weight', 'Volume']]
y = df['CO2']

regr = linear_model.LinearRegression()
regr.fit(X, y)

predictedCO2 = regr.predict([[3300, 1300]])

print(predictedCO2)
```

Результат:
[114.75968007]

Ми передбачили, що автомобіль з двигуном об'ємом 1,3 літра та вагою 3300 кг буде викидати приблизно 115 грамів CO2 на кожен пройдений кілометр.

Це показує, що коефіцієнт 0.00755095 є правильним:

107.2087328 + (1000 * 0.00755095) = 114.75968

Шкалювання(Scale Features)

Шкалювання означає перетворення даних на нові значення, які легше порівнювати між собою, навіть якщо вони мають різні величини та одиниці вимірювання.

Візьмемо таблицю нижче, яка є тією ж самою набором даних, що й у розділі множинної регресії, але цього разу стовпець об'єму містить значення в літрах, а не в см3 (1.0 замість 1000).

Weight (kg)	Volume (liters)	CO2 (g/km)
2300	1.0	107
2500	1.3	115
1900	1.6	87
2100	1.7	99

Якщо ми порівняємо вагу та об'єм без шкалювання, може бути важко зрозуміти, який фактор має більший вплив на викиди CO2.

Для розв'язання цієї проблеми ми можемо використовувати шкалювання. Одним з поширених підходів є масштабування даних до діапазону від 0 до 1 за допомогою методу мінімаксного шкалювання (min-max scaling).

Мінімаксне шкалювання можна застосувати до кожного стовпця, розділивши кожне значення на максимальне значення у стовпці. Результатом буде новий набір значень, які лежать в діапазоні від 0 до 1.

Таким чином, шкалювання допомагає зрозуміти вплив різних факторів у наборі даних, навіть якщо вони мають різні величини та одиниці вимірювання.

Отже, коли ваші дані мають різні значення і навіть різні одиниці вимірювання, їх важко порівнювати. Що означає кілограми порівняно з метрами? Або висоту порівняно з часом?

Відповідь на цю проблему - шкалювання. Ми можемо перетворити дані на нові значення, які легше порівнювати.

Дивіться на таблицю нижче. Це той самий набір даних, який ми використовували в розділі про множинну регресію, але на цей раз стовпець об'єму містить значення в літрах замість см3 (1.0 замість 1000).

Марка	Модель	Об’єм	Вага	CO^2
Toyota	Aygo	1.0	790	99
Mitsubishi	Space Star	1.2	1160	95
Skoda	Citigo	1.0	929	95
Fiat	500	0.9	865	90
Mini	Cooper	1.5	1140	105
VW	Up!	1.0	929	105
Skoda	Fabia	1.4	1109	90
Mercedes	A-Class	1.5	1365	92
Ford	Fiesta	1.5	1112	98
Audi	A1	1.6	1150	99
Hyundai	I20	1.1	980	99
Suzuki	Swift	1.3	990	101
Ford	Fiesta	1.0	1112	99
Honda	Civic	1.6	1252	94
Hundai	I30	1.6	1326	97

Opel	Astra	1.6	1330	97
BMW	1	1.6	1365	99
Mazda	3	2.2	1280	104
Skoda	Rapid	1.6	1119	104
Ford	Focus	2.0	1328	105
Ford	Mondeo	1.6	1584	94
Opel	Insignia	2.0	1428	99
Mercedes	C-Class	2.1	1365	99
Skoda	Octavia	1.6	1415	99
Volvo	S60	2.0	1415	99
Mercedes	CLA	1.5	1465	102
Audi	A4	2.0	1490	104
Audi	A6	2.0	1725	114
Volvo	V70	1.6	1523	109
BMW	5	2.0	1705	114
Mercedes	E-Class	2.1	1605	115
Volvo	XC70	2.0	1746	117
Ford	B-Max	1.6	1235	104
BMW	2	1.6	1390	108
Opel	Zafira	1.6	1405	109
Mercedes	SLK	2.5	1395	120

Це може бути складно порівняти об'єм 1,0 з вагою 790, але якщо ми шкалуємо їх до порівняльних значень, ми легко можемо побачити, наскільки одне значення порівняно з іншим.

Існують різні методи для шкалювання даних, у цьому посібнику ми будемо використовувати метод, який називається стандартизацією.

Метод стандартизації використовує таку формулу:

$$z = (x - u)/s$$

Де z - нове значення, x - початкове значення, u - середнє значення і s - стандартне відхилення.

Якщо ви візьмете стовпець з вагою з наведеного вище набору даних, перше значення - 790, і шкальоване значення буде:

$$(790 - 1292.23)/238.74 = -2.1$$

Якщо ви візьмете стовпець об'єму з наведеного вище набору даних, перше значення - 1,0, і шкальоване значення буде:

$$(1,0 - 1.61)/0.38 = -1.59$$

Тепер ви можете порівняти -2.1 з -1.59, а не порівнювати 790 з 1.0.

Вам не потрібно робити це вручну, у модулі Python sklearn є метод під назвою StandardScaler(), який повертає об'єкт Scaler з методами для перетворення наборів даних.

Шкальування всіх значень у стовпцях "Weight" та "Volume":

```
import pandas
from sklearn.preprocessing import StandardScaler

df = pandas.read_csv("data.csv")

X = df[['Weight', 'Volume']]

scale = StandardScaler()
scaledX = scale.fit_transform(X)

print(scaledX)
```

Результат:
Зверніть увагу, що перші два значення -2.1 та -1.59 відповідають нашим розрахункам:

[[-2.10389253 -1.59336644]
[-0.55407235 -1.07190106]
[-1.52166278 -1.59336644]
[-1.78973979 -1.85409913]
[-0.63784641 -0.28970299]
[-1.52166278 -1.59336644]
[-0.76769621 -0.55043568]
[0.3046118 -0.28970299]
[-0.7551301 -0.28970299]
[-0.59595938 -0.0289703]

[-1.30803892 -1.33263375]
[-1.26615189 -0.81116837]
[-0.7551301 -1.59336644]
[-0.16871166 -0.0289703]
[0.14125238 -0.0289703]
[0.15800719 -0.0289703]
[0.3046118 -0.0289703]
[-0.05142797 1.53542584]
[-0.72580918 -0.0289703]
[0.14962979 1.01396046]
[1.2219378 -0.0289703]
[0.5685001   1.01396046]
[0.3046118   1.27469315]
[0.51404696 -0.0289703]
[0.51404696 1.01396046]
[0.72348212 -0.28970299]
[0.8281997   1.01396046]
[1.81254495 1.01396046]
[0.96642691 -0.0289703]
[1.72877089 1.01396046]
[1.30990057 1.27469315]
[1.90050772 1.01396046]
[-0.23991961 -0.0289703]
[0.40932938 -0.0289703]
[0.47215993 -0.0289703]
[0.4302729   2.31762392]]

Попередній приклад використовує шкальовані дані для передбачення значень CO2.

Приклад:
Передбачити викиди CO2 автомобіля з двигуном об'ємом 1,3 літра та вагою 2300 кілограмів:

```
import pandas
from sklearn import linear_model
from sklearn.preprocessing import StandardScaler

df = pandas.read_csv("data.csv")
```

```
X = df[['Weight', 'Volume']]
y = df['CO2']

scale = StandardScaler()
scaledX = scale.fit_transform(X)

regr = linear_model.LinearRegression()
regr.fit(scaledX, y)

scaled = scale.transform([[2300, 1.3]])

predictedCO2 = regr.predict([scaled[0]])
print(predictedCO2)
```

Результат:
[107.2087328]

Треновочні та тестові дані

Оцінка вашої моделі

У машинному навчанні ми створюємо моделі для передбачення результату певних подій, як у попередньому розділі, де ми передбачали викиди CO2 автомобіля, коли ми знали його вагу та об'єм двигуна.

Для вимірювання точності моделі можна використовувати метод, який називається Train/Test (навчання/тестування).

Що таке Train/Test

Train/Test - це метод для вимірювання точності вашої моделі.

Він називається Train/Test, тому що ви розбиваєте набір даних на дві частини: навчальний набір та тестовий набір.

80% для навчання і 20% для тестування.

Ви навчаєте модель, використовуючи навчальний набір.

Ви перевіряєте модель, використовуючи тестовий набір.

Навчання моделі означає створення моделі.

Тестування моделі означає перевірку точності моделі.

Почніть з набору даних

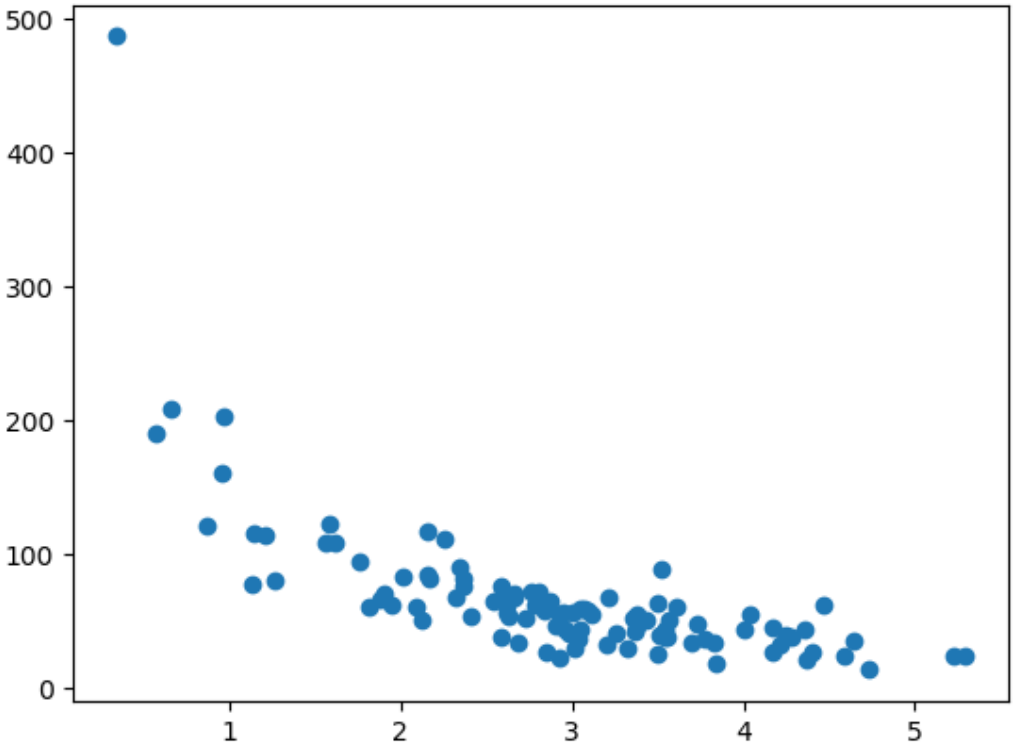
Почніть з набору даних, який ви хочете перевірити.

Наш набір даних відображає 100 покупців у магазині та їхні звички покупок.

```
import numpy
import matplotlib.pyplot as plt

numpy.random.seed(2)

x = numpy.random.normal(3, 1, 100)
y = numpy.random.normal(150, 40, 100) / x

plt.scatter(x, y)
plt.show()
```

Результат:
Вісь x представляє кількість хвилин до здійснення покупки.

Вісь y представляє суму грошей, витрачених на покупку.



Розбиття на навчальний/тестовий набір

Навчальний набір повинен бути випадковим вибором 80% від загальної кількості даних.

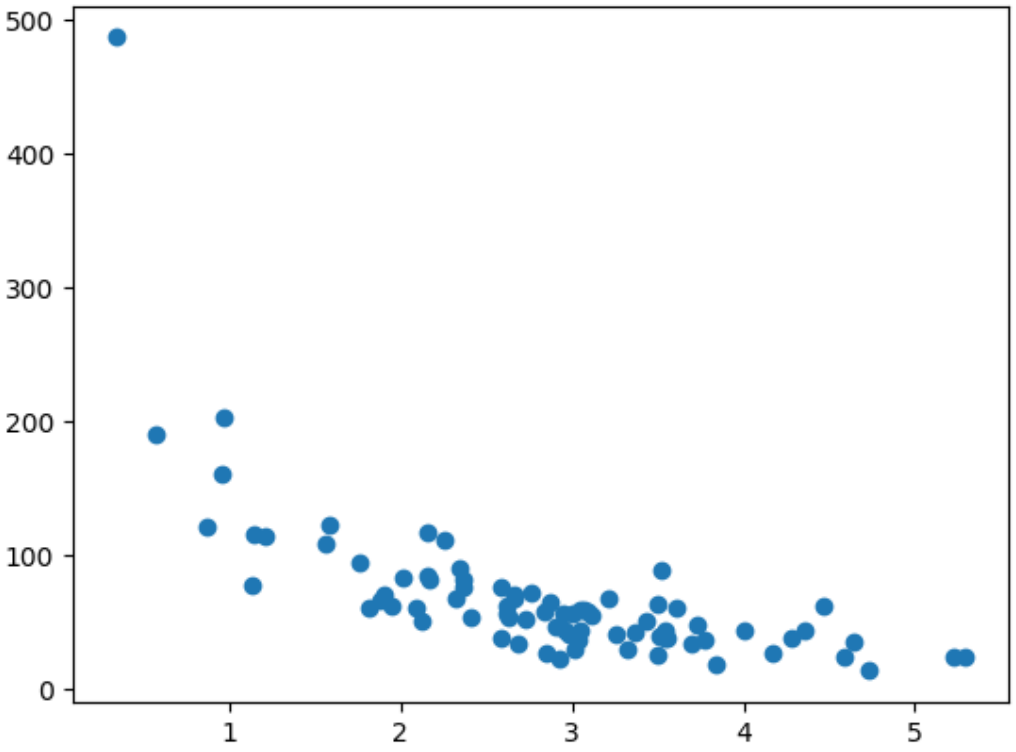
Тестовий набір повинен складатись з решти 20%.

```
train_x = x[:80]
train_y = y[:80]

test_x = x[80:]
test_y = y[80:]
```

Відображення навчального набору
Відобразіть ту саму діаграму розсіювання з навчальним набором:

```
plt.scatter(train_x, train_y)
plt.show()
```

Результат:
Вона виглядає так само, як і вихідний набір даних, тому воно, схоже, є справедливим вибором:



Відображення набору для тестування

Щоб переконатися, що набір для тестування не є повністю відмінним, ми також розглянемо набір для тестування.

```
plt.scatter(test_x, test_y)
plt.show()
```

Результат:
Набір для тестування також виглядає схожим на вихідний набір даних.

Приведення(fit) набору даних

Як виглядає набір даних? На мою думку, найкращим підходом буде поліноміальна регресія, тому давайте побудуємо лінію поліноміальної регресії.

Для побудови лінії через точки даних використовуємо метод plot() модуля matplotlib:

Приклад:
Побудова лінії поліноміальної регресії через точки даних:

```
import numpy
import matplotlib.pyplot as plt
numpy.random.seed(2)
x = numpy.random.normal(3, 1, 100)
y = numpy.random.normal(150, 40, 100) / x
train_x = x[:80]
train_y = y[:80]
test_x = x[80:]
test_y = y[80:]
mymodel = numpy.poly1d(numpy.polyfit(train_x, train_y, 4))
myline = numpy.linspace(0, 6, 100)
plt.scatter(train_x, train_y)
plt.plot(myline, mymodel(myline))
plt.show()
```