Конспект по теме «Векторы и векторные операции»

Создание векторов

В Python наборы данных обычно представлены списками. В математике упорядоченный набор числовых данных — это **вектор**, или **арифметический вектор**. К векторам применимы те же операции, что и к числам: сложение, вычитание и умножение. И в отличие от действий со списками, операции с векторами в Python производятся в сотни раз быстрее.

Чтобы работать с векторами, обратимся к возможностям библиотеки **NumPy.** Создадим список из двух чисел и преобразуем список в вектор:

```
import numpy as np

numbers1 = [2, 3] # Список Python
vector1 = np.array(numbers1) # Массив NumPy
print(vector1)
```

Создать вектор можно без промежуточной переменной:

```
import numpy as np
vector2 = np.array([6, 2])
print(vector2)
```

Вектор можно преобразовать в список:

```
numbers2 = list(vector2) # Список из вектора
print(numbers2)
```

Столбец структуры *DataFrame* в *Pandas* преобразуется в вектор *NumPy* атрибутом **values**:

```
import pandas as pd

data = pd.DataFrame([1, 7, 3])
print(data[0].values)
```

Функцией *len()* определим размер вектора, то есть количество элементов в нём:

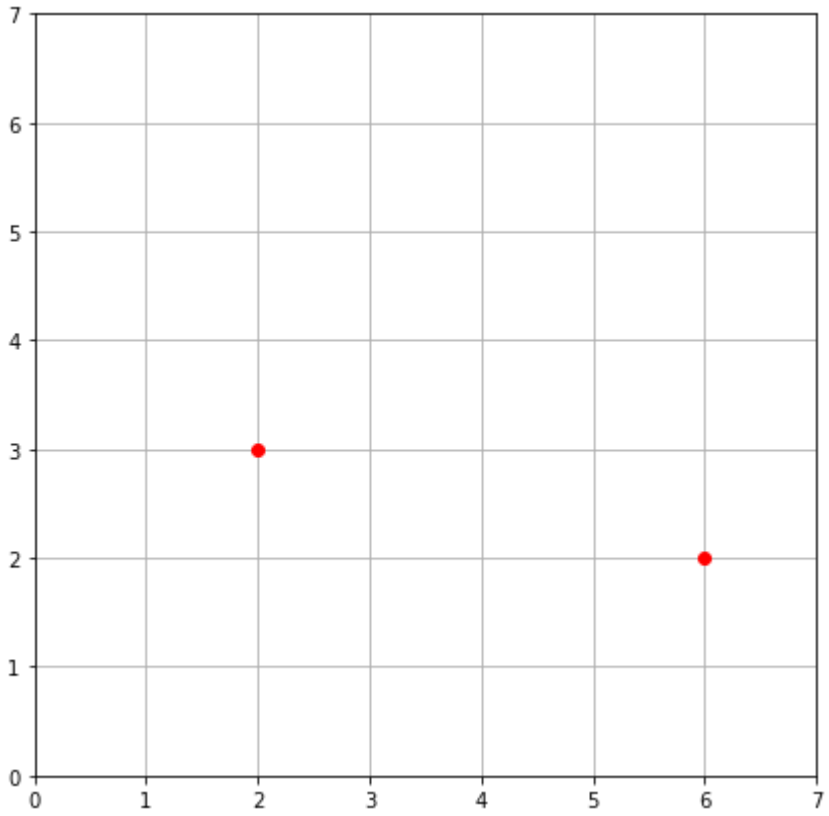
```
print(len(vector2))
```

Изображение векторов

Изобразим **двумерный вектор**. Он состоит из двух чисел: первое — это координата по горизонтальной оси *x*, а второе — по вертикальной *y*. Вектор изображается **точкой** либо **стрелкой**, которая соединяет начало координат и точку с координатами *(x, y)*. Стрелкой вектор отображается, когда нужно показать движение. Если мы имеем дело с коллекцией векторов, которые могут лежать на одной линии, вектор лучше отображать точкой.

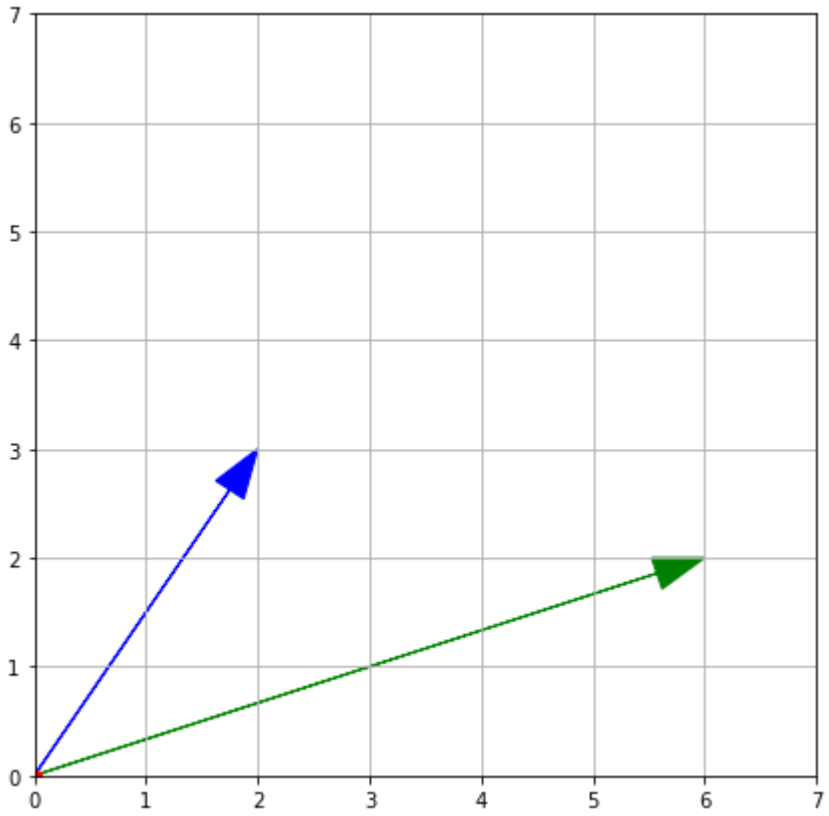
Элементы вектора также называют координатами.

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

vector1 = np.array([2, 3])
vector2 = np.array([6, 2])

plt.figure(figsize=(7, 7))
plt.axis([0, 7, 0, 7])
# аргумент 'ro' задаёт стиль графика
# 'r' - красный цвет (англ. red)
# 'o' - круг
plt.plot([vector1[0], vector2[0]], [vector1[1], vector2[1]], 'ro')
plt.grid(True)
plt.show()
```



Представим эти же векторы стрелками. Вместо метода *plt.plot()* вызовем **plt.arrow()** (англ. «стрела»).

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

vector1 = np.array([2, 3])
vector2 = np.array([6, 2])

plt.figure(figsize=(7, 7))
plt.axis([0, 7, 0, 7])
plt.arrow(0, 0, vector1[0], vector1[1], head_width=0.3,
          length_includes_head="True", color='b')
plt.arrow(0, 0, vector2[0], vector2[1], head_width=0.3,
          length_includes_head="True", color='g')
plt.plot(0, 0, 'ro')
plt.grid(True)
plt.show()
```



Сложение и вычитание векторов

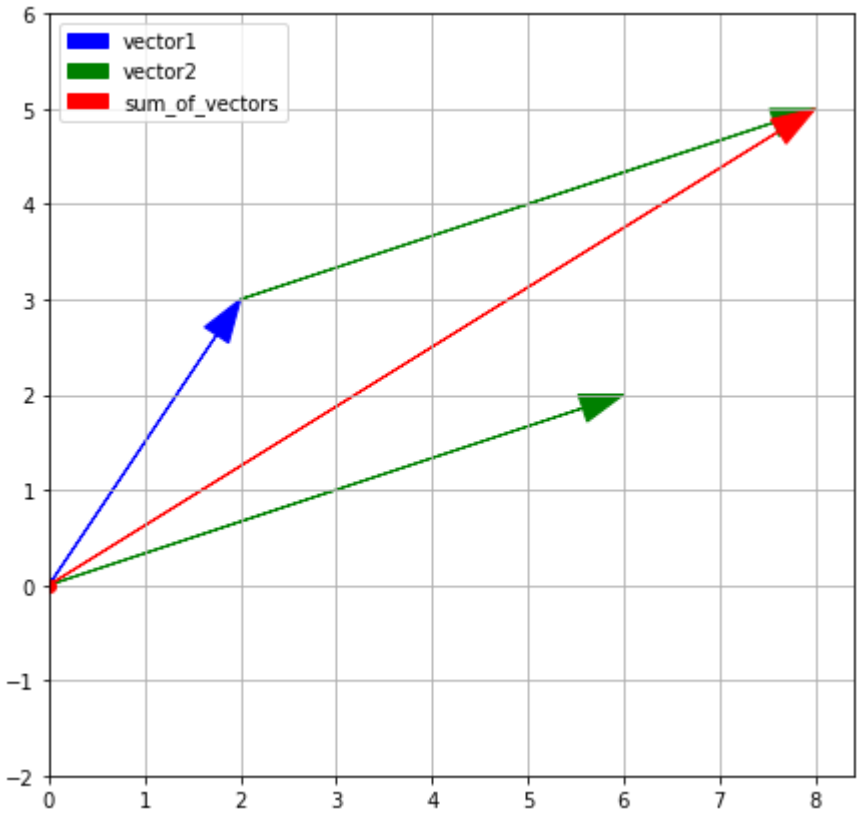
Векторы можно складывать и вычитать, если они одного размера. Векторы одинакового размера состоят из равного количества чисел. В результате их сложения получается вектор, каждая координата которого равна сумме координат векторов-слагаемых. То есть его первая координата равна сумме первых координат, а вторая — сумме вторых. При вычитании образуется вектор, у которого каждая координата равна разности координат заданных векторов.

Сложение и вычитание векторов производится по элементам:

Вектор	Координаты
$\bar{x}$	$(x_1, x_2, \dots, x_n)$
$\bar{y}$	$(y_1, y_2, \dots, y_n)$
$\bar{x} + \bar{y}$	$(x_1 + y_1, x_2 + y_2, \dots, x_n + y_n)$
$\bar{x} - \bar{y}$	$(x_1 - y_1, x_2 - y_2, \dots, x_n - y_n)$

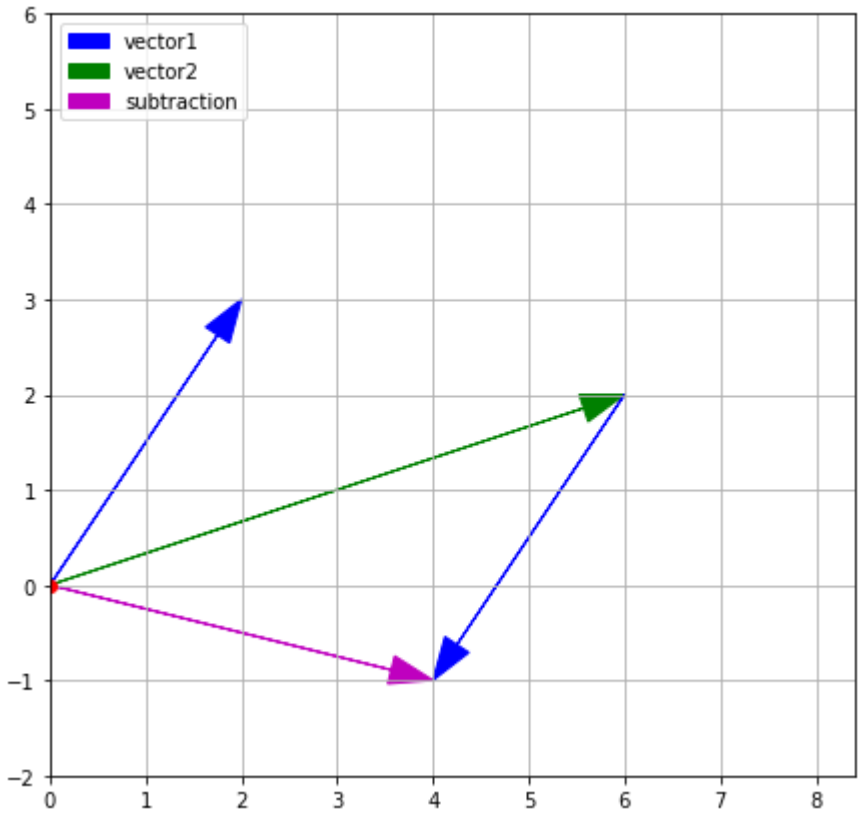
```
import numpy as np

vector1 = np.array([2, 3])
vector2 = np.array([6, 2])
sum_of_vectors = vector1 + vector2
subtraction_of_vectors = vector2 - vector1
```

Если от конца синего вектора *vector1* отложить вектор, равный по длине и направлению зелёному *vector2*, то получится красный вектор *sum_of_vectors*.



Такой треугольник придаёт сложению векторов геометрический смысл. Если каждый вектор — это движение в определённом направлении, то сумма двух — перемещение вдоль первого вектора, а затем — вдоль второго.

А разность — это шаг, к примеру, вдоль вектора *vector2*, а затем — в противоположном вектору *vector1* направлении.



Умножение вектора на число

Помимо сложения и вычитания, векторы можно умножать на числа. Каждая координата вектора умножается на одно и то же число:

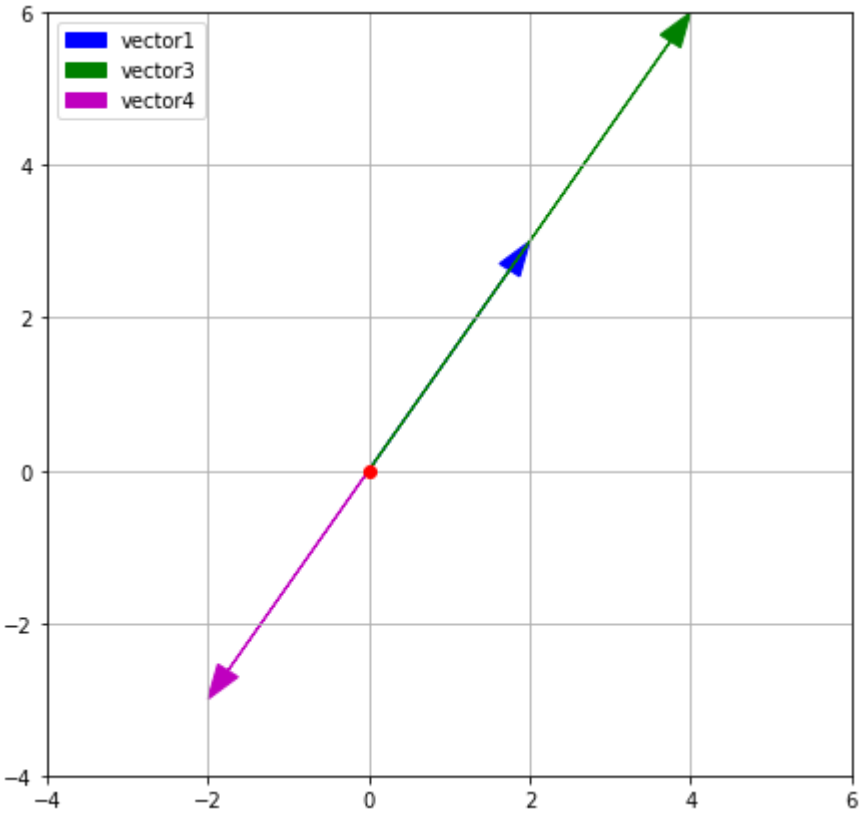
Вектор	Координаты
$\bar{x}$	$(x_1, x_2, \dots, x_n)$
$k\bar{x}$	$(kx_1, kx_2, \dots, kx_n)$

Если это число отрицательное, все координаты тоже меняют знаки на противоположные.

```
import numpy as np
```

```
vector1 = np.array([2, 3])
vector3 = 2 * vector1
vector4 = -1 * vector1
```

При умножении на положительное число векторы на плоскости сохраняют направление, но стрелки становятся длиннее или короче. При умножении на отрицательное — направление меняется на противоположное.



Среднее значение векторов

Если отдельные векторы в наборе описывают, к примеру, клиентов по признакам, то среднее значение часто описывает типичного, или *среднестатического* клиента. Для набора векторов $a1$, $a2$,..., an (где *n* — общее число векторов) их **среднее значение** — это произведение суммы всех векторов на число $1/n$, то есть новый вектор *a:*

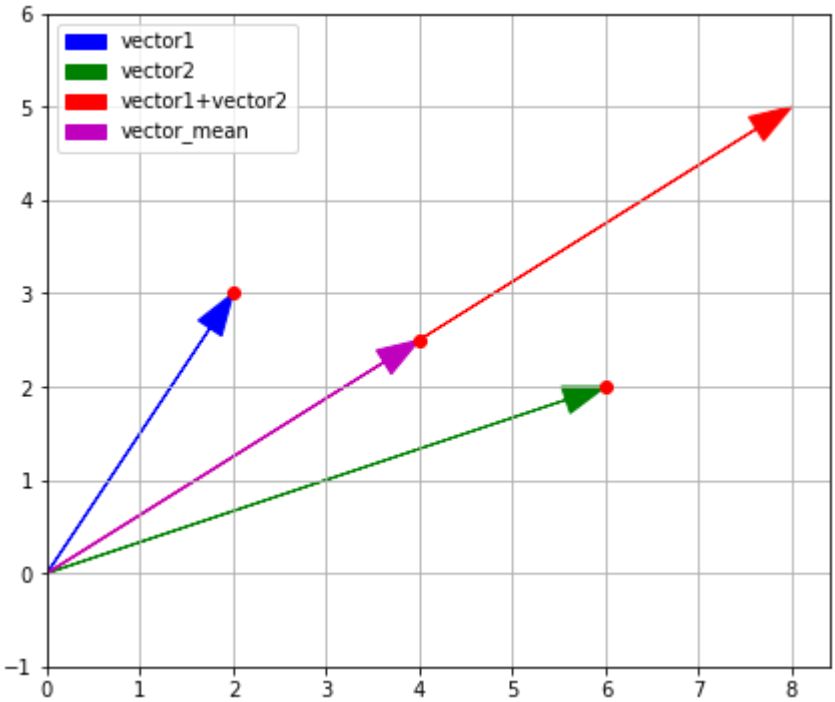
$$a = \frac{1}{n}(a_1 + a_2 + \ldots + a_n)$$

Если в наборе только один вектор (n=1), он совпадает со средним: a=a1. Для двух векторов среднее вычисляется так: a=0.5(a1+a2). А среднее для пары двумерных векторов — это середина отрезка, соединяющего a1 и a2.

```
import numpy as np

vector1 = np.array([2, 3])
vector2 = np.array([6, 2])
vector_mean = .5*(vector1+vector2)
print(vector_mean)
```

Первая координата вектора среднего значения — это среднее значение первых координат векторов *vector1* и *vector2*, а вторая — среднее вторых координат.

На плоскости эти векторы изображаются так: строится вектор *vector1+vector2*, затем он умножается на 0.5.



Векторизованные функции

Средства библиотеки *NumPy* позволяют производить и другие операции над векторами. Применив функцию *np.array*, после умножения и деления двух массивов одного размера получим новый вектор такого же размера:

```
import numpy as np

array1 = np.array([2, -4, 6, -8])
array2 = np.array([1, 2, 3, 4])
array_mult = array1 * array2
array_div = array1 / array2
print("Произведение двух массивов: ", array_mult)
print("Частное двух массивов: ", array_div)
```

Если арифметические операции производятся над массивом и отдельным числом, то действие применяется к каждому элементу массива. И снова образуется массив такого же размера.

Чтобы проверить, произведём над массивом и числом операции сложения, вычитания и деления:

```
import numpy as np

array2 = np.array([1, 2, 3, 4])
array2_plus_10 = array2 + 10
array2_minus_10 = array2 - 10
array2_div_10 = array2 / 10
print("Сумма: ", array2_plus_10)
print("Разность: ", array2_minus_10)
print("Частное массива и числа: ", array2_div_10)
```

К массиву также поэлементно применимы и стандартные математические функции, например, возведение в степень или логарифмы.

Возведём массив во вторую степень:

```
import numpy as np
```

```
numbers_from_0 =  np.array([0, 1, 2, 3, 4])
squares = numbers_from_0**2
print(squares)
```

Всё это можно сделать и циклами над списками. Но операции с векторами в библиотеке *NumPy* работают быстрее.

Среди элементов массива *values* максимальное и минимальное значения — это некоторые числа *MAX* и *MIN* при условии, что *MAX*>*MIN*. Для анализа данные нужно преобразовать. Каждый элемент x нашего массива должен перейти в число в интервале от 0 (*MIN*) до 1 (*MAX*). Формула функции **min_max_scale** выглядит так:

$$f(x) = \frac{x - MIN}{MAX - MIN}$$

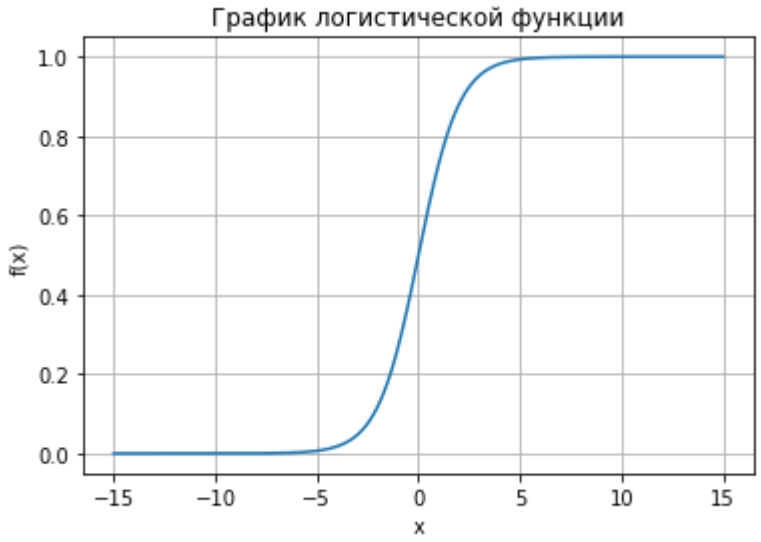
Чтобы применить эту функцию ко всем элементам массива *values*, вызовем методы *max()* и *min()*. Они найдут его максимальное и минимальное значения. В результате получим массив той же длины, но с преобразованными элементами:

```
import numpy as np
def min_max_scale(values):
    return (values - min(values)) / (max(values) - min(values))

print(min_max_scale(our_values))
```

Иногда значения могут быть сколь угодно большими. Их нужно преобразовать так, чтобы все значения попали в интервал от 0 до 1. Для этого применяют **логистическую функцию**, или **логистическое преобразование**:

$$f(x) = \frac{1}{1 + \exp(-x)}$$

где **exp()** — это **экспонента**, или **показательная функция**. Она возводит *e* — **число Эйлера** — в степень аргумента. Число названо в честь швейцарского математика Леонарда Эйлера, приблизительно равно 2.718281828.



Проведём логистическое преобразование:

```
import numpy as np

def logistic_transform(values):
    return 1 / (1 + np.exp(- values))

print(logistic_transform(our_values))
```

Векторизация метрик

В переменной *target* сохраним набор фактических значений, а в *predictions* — предсказанных. Оба набора типа *np.array*.

Для вычисления метрик качества применим стандартные функции *NumPy*:

- *sum()* (чтобы найти сумму элементов массива);
- *mean()* (для расчёта их среднего значения).

Вызовем их в формате: `<имя массива>.sum()` и `<имя массива>.mean()`.

Например, **средний квадрат отклонения** (*MSE*) вычисляется по формуле:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(target_i - predictions_i)^2$$

где n — длина каждого из массивов, ∑ — суммирование по всем объектам выборки (i меняется от 1 до n). Порядковые элементы векторов *target* и *predictions* обозначаются *target_i* и *predictions_i*.

Запишем эту формулу с применением *sum()*:

```
def mse1(target, predictions):
    n = target.size
    return((target - predictions)**2).sum()/n
```

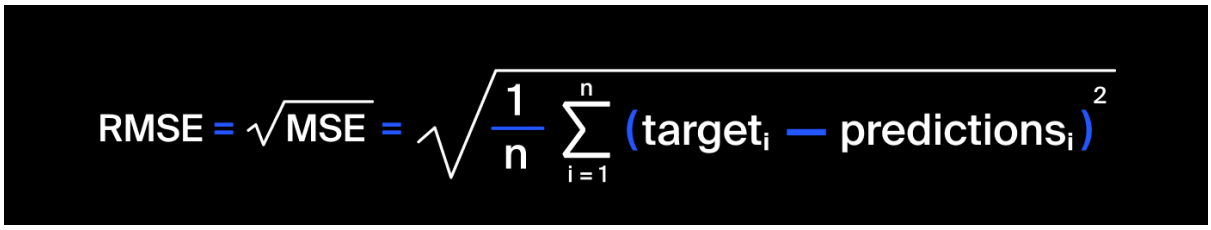
Заметили, что сумма нескольких чисел, поделённая на их количество, — это среднее значение? Запишем формулу *MSE* проще — с применением функции *mean():*

```
def mse2(target, predictions):
    return((target - predictions)**2).mean()
```

Аналогично можно написать функцию для вычисления *MAE* с применением метода *mean()*.


```
import numpy as np

def mae(target, predictions):
    return np.abs((target - predictions)).mean()

print(mae(target, predictions))
```

Также векторными функциями можно рассчитать RMSE по такой формуле:




```
import numpy as np

def rmse(target, predictions):
    return (((target-predictions)**2).mean())**0.5

print(rmse(target, predictions))
```