В данной работе я познакомился с описательной статистикой. Описательная статистика — это описание и интегральные параметры наборов данных. Если говорить о метриках, то в этой части была проработана центральная метрика (которая говорит нам о центрах концентрации данных, таких как среднее, медиана и мода) и метрика вариативности данных (которая говорит о разбросе значений, таких как дисперсия и стандартное отклонение). Так как я работал в PyCharm, то первым делом я зашел в интерпретатор, установил необходимые библиотеки и вызвал их:

import math

import statistics

import numpy as np

import scipy.stats

import pandas as pd

После этого, я создал некоторые исходные данные:

x = [10.0, 2, 2.5, 5, 26.0]  
x\_with\_nan = [10.0, 2, 2.5, math.nan, 5, 26.0]  
print(f'Вывод исходных данных, которые содержатся в x:{x}')  
print(f'Вывод исходных данных, которые содержатся в x\_with\_nan:{x\_with\_nan}')

Теперь у нас есть списки x и x\_with\_nan. Они почти одинаковы, с той разницей, что x\_with\_nan содержат nan значение. Важно понимать поведение процедур статистики Python, когда они сталкиваются с нечисловым значением (nan). В науке о данных пропущенные значения являются общими, и вы часто будете заменять их на nan. Теперь создаем объекты np.ndarray и pd.Series, соответствующие x и x\_with\_nan:

y, y\_with\_nan = np.array(x), np.array(x\_with\_nan)  
z, z\_with\_nan = pd.Series(x), pd.Series(x\_with\_nan)  
print(f'Вывод данных, которые содержатся в y и y\_with\_nan:{y}, {y\_with\_nan}')  
print(f'Вывод данных, которые содержатся в z и в z\_with\_nan: {z}, {z\_with\_nan}')

Теперь у нас есть два массива NumPy (y и y\_with\_nan) и два объекта Series Pandas (z и z\_with\_nan). Все это — 1D‑последовательности значений. После формирования исходных данных, приступаем к расчету центральной метрики, а именно среднего значения:

#Рассчет средних значений  
mean\_1 = sum(x) / len(x)  
print(f'Расчет среднего значения, используя sum и len: {mean\_1}')  
mean\_2 = statistics.mean(x)  
print(f'Расчет среднего значения, используя встроенные функции статистики Python (statistics.mean(x)): {mean\_2}')  
mean\_3 = statistics.fmean(x)  
print(f'Расчет среднего значения, используя встроенные функции статистики Python (statistics.fmean(x)): {mean\_3}')  
mean\_4 = statistics.mean(x\_with\_nan)  
print(f'Расчет среднего значения, который содержит значения nan, используя встроенные функции статистики Python (statistics.mean(x)): {mean\_4}')  
mean\_5 = np.mean(y)  
print(f'Расчет среднего значения, используя NumPy: {mean\_5}')  
np.nanmean(y\_with\_nan)  
print(f'Расчет среднего значения с помощью NumPy, игнорируя nan: {np.nanmean(y\_with\_nan)}')  
mean\_6 = z.mean()  
print(f'Расчет среднего значения объекта pd.Series: {mean\_6}')

Первое среднее значение было рассчитано на чистом Python, используя [sum()](https://docs.python.org/3/library/functions.html" \l "sum" \t "_blank) и [len()](https://docs.python.org/3/library/functions.html" \l "len" \t "_blank), без импорта библиотек. Хотя это чисто и элегантно, но для расчета второго, третьего и четвертого значения были применены встроенные функции библиотеки statistics Python. При расчете пятого среднего значения была использована библиотека NumPy и функция np.mean. А шестое среднее значение было рассчитано с помощью метода .mean() библиотеки Pandas.

Далее, было рассчитано средневзвешенное значение. Средневзвешенное или также называемое средневзвешенным арифметическим или средневзвешенным значением, является обобщением среднего арифметического, которое позволяет вам определить относительный вклад каждой точки данных в результат. Сам расчет:

#Рассчет средневзвешанных значений  
x = [6.0, 1, 2.5, 6, 25.0]  
w = [0.1, 0.2, 0.3, 0.25, 0.15]  
wmean = sum(w[i] \* x[i] for i in range(len(x))) / sum(w)  
print(f'Расчет средневзвешанного с помощью range: {wmean}')  
wmean2 = sum(x\_ \* w\_ for (x\_, w\_) in zip(x, w)) / sum(w)  
print(f'Расчет средневзвешанного с помощью zip: {wmean2}')  
y, z, w = np.array(x), pd.Series(x), np.array(w)  
wmean3= np.average(y, weights=w)  
print(f'Расчет средневзвешанного с помощью np.average для массивово NumPy или серии Pandas: {wmean3}')  
o = (w \* y).sum() / w.sum()  
print(f'Расчет средневзвешанного с помощью поэлементного умножения w \* y: {o}')  
w = np.array([0.1, 0.2, 0.3, 0.0, 0.2, 0.1])  
print(f'Расчет средневзвешанного для набора, который содержит nan : {(w \* y\_with\_nan).sum() / w.sum()}')

Первое и второе средневзвешенное было рассчитано В чистом Python используя комбинацию sum() с [range()](https://realpython.com/courses/python-range-function/" \t "_blank) и [zip()](https://realpython.com/python-zip-function/" \t "_blank). Опять же, это чистая и элегантная реализация, в которой вам не нужно импортировать какие-либо библиотеки. Однако, если у вас большие наборы данных, то NumPy, вероятно, будет лучшим решением. Можно использовать [np.average()](https://docs.scipy.org/doc/numpy/reference/generated/numpy.average.html), как это сделано при расчете третьего показателя, для массивов NumPy или серии Pandas. Для расчета четвертого и пятого показателя, было использовано поэлементное умножение с методом .sum().

После этого, было рассчитано гармоническое среднее, что есть обратная величина от среднего значения обратных величин всех элементов в наборе данных:

#Гармоническое среднее  
hmean = len(x) / sum(1 / item for item in x)  
print(f'Расчет гармонического среднего: {hmean}')  
hmean2 = statistics.harmonic\_mean(x)  
print(f'Расчет гармонического среднего с помощью statistics.harmonic\_mean(): {hmean2}')  
statistics.harmonic\_mean(x\_with\_nan)  
print(f'Расчет гармонического среднего, где есть nan: {statistics.harmonic\_mean(x\_with\_nan)}')  
statistics.harmonic\_mean([1, 0, 2])  
print(f'Расчет гармонического среднего, где есть 0: {statistics.harmonic\_mean([1, 0, 2])}')  
scipy.stats.hmean(y)  
print(f'Расчет гармонического среднего с помощью scipy.stats.hmean(): {scipy.stats.hmean(y)}')

Как обычно, первое значение было рассчитано на чистом Python. Второе, третье и четвертое значение было рассчитано с помощью функции statistics.harmonic\_mean(). И последнее было рассчитано используя scipy.stats.hmean.

Следом был рассчитан среднее геометрическое:

#Среднее геометрическое  
gmean = 1  
for item in x:  
 gmean \*= item  
 gmean \*\*= 1 / len(x)  
print(f'Вычисление геометрического среднего: {gmean}')  
gmean2 = statistics.geometric\_mean(x)  
print(f'Вычисление геометрического среднего с помощью statistics.geometric\_mean(): {gmean2}')  
gmean3 = statistics.geometric\_mean(x\_with\_nan)  
print(f'Вычисление геометрического среднего где есть nan: {gmean3}')  
scipy.stats.gmean(y)  
print(f'Вычисление геометрического среднего с помощью scipy.stats.gmean(): {scipy.stats.gmean(y)}')

Первое значение было рассчитано на чистом Python. Второе и третье значение было рассчитано с помощью функции statistics.geometric\_mean(). И последнее было рассчитано используя scipy.stats.gmean.

Медиана — это средний элемент отсортированного набора данных. Расчет медианы представлен внизу:

n = len(x)  
if n % 2:  
 median\_ = sorted(x)[round(0.5\*(n-1))]  
else:  
 x\_ord, index = sorted(x), round(0.5 \* n)  
 median\_ = 0.5 \* (x\_ord[index-1] + x\_ord[index])  
print(f'Расчет медианы: {median\_}')  
median\_2 = statistics.median(x)  
print(f'Расчет медианы с помощью statistics.median(): {median\_2}')  
statistics.median\_low(x[:-1])  
print(f'Расчет медианы с помощью statistics.median\_low: {statistics.median\_low(x[:-1])}')  
statistics.median\_high(x[:-1])  
print(f'Расчет медианы с помощью statistics.median\_high {statistics.median\_high(x[:-1])}')

median\_2 = np.median(y)  
print(f'Расчет медианы с помощью np.median: {median\_2}')

Первое значение было рассчитано на чистом Python. Следующие три были найдены используя statistics.median, при этом, median\_low() возвращает меньшее, а median\_high() — большее среднее значение. И последняя была найдена с помощью NumPy и функции np.median().

Мода — это значение в наборе данных, которое встречается чаще всего. Если такого значения не существует, набор является мультимодальным, поскольку он имеет несколько модальных значений.  Расчет моды представлен внизу:

u = [2, 3, 2, 8, 12]  
mode\_ = max((u.count(item), item) for item in set(u))[1]  
print(f'Вычисление моды: {mode\_}')  
mode\_2 = statistics.mode(u)  
print(f'Вычисление моды с помощью statistics.mode(): {mode\_2}')  
mode\_3 = statistics.multimode(u)  
print(f'Вычисление моды с помощью statistics.multimode(): {mode\_3}')  
mode\_4 = scipy.stats.mode(u)  
print(f'Вычисление моды с помощью scipy.stats.mode(): {mode\_4}')

Первое значение, как обычно, получено используя чистый Python. Вы используете u.count(), чтобы получить количество вхождений каждого элемента в u. Элемент с максимальным количеством вхождений — это мода. Обратите внимание, что вам не нужно использовать set(u). Вместо этого вы можете заменить его просто на u и повторить весь список. Второе и третье значение было вычислено с помощью [statistics.mode()](https://docs.python.org/3/library/statistics.html" \l "statistics.mode" \t "_blank) и [statistics.multimode()](https://docs.python.org/3/library/statistics.html" \l "statistics.multimode" \t "_blank). Обратите внимание, mode() вернула одно значение, а multimode() в результате вернула список. Однако, это не единственное различие между двумя функциями. Если существует более одного модального значения, то mode() вызывает StatisticsError, а multimode() возвращает список со всеми режимами. И последнее значение было найдено с помощью функции, которая возвращает объект с модальным значением и количество его повторений в наборе данных.

Центральных метрик недостаточно для описания данных. Практически всегда необходимы **метрики оценки вариативности данных**, которые количественно определяют разброс точек данных. И первым показателем метрики оценки вариативности данных была дисперсия. Дисперсия количественно определяет разброс данных. Численно показывает, как далеко точки данных от среднего значения.  Сам расчет:

n = len(x)  
mean = sum(x) / n  
var\_ = sum((item - mean)\*\*2 for item in x) / (n - 1)  
print(f'Оценка дисперсии на чистом Python: {var\_}')  
var\_1= statistics.variance(x)  
print(f'Оценка дисперсии с помощью statistics.variance(): {var\_1}')  
statistics.variance(x\_with\_nan)  
print(f'Оценка дисперсии с помощью statistics.variance(), где есть nan: {statistics.variance(x\_with\_nan)}')  
var\_2 = np.var(y, ddof=1)  
print(f'Оценка дисперсии, используя NumPy с помощью np.var(): {var\_2}')  
var\_3 = y.var(ddof=1)  
print(f'Оценка дисперсии, используя NumPy с помощью метода .var(): {var\_3}')

Первый метод расчета – используя чистый Python. В целом, этого достаточно и можно правильно дать оценку дисперсии. Однако, более короткое и элегантное решение — использовать функцию [statistics.variance()](https://docs.python.org/3/library/statistics.html#statistics.variance) (как сделано это при расчете второго показателя). В результате мы получили тот же результат для дисперсии, что и в первом. И оставшиеся последние два показателя были рассчитаны используя NumPy, а именно функции np.var() и метода .var()

Далее, было рассчитано среднеквадратичное отклонение. Стандартное отклонение выборки является еще одним показателем разброса данных. Он связан с оценкой дисперсией, поскольку стандартное отклонение есть положительным квадратный корень из оценки дисперсии. Стандартное отклонение часто более удобно, чем дисперсия, потому что имеет ту же размерность, что и данные. Сам расчет:

#Среднеквадратичное отклонение  
std\_ = var\_ \*\* 0.5  
print(f'Расчет среднеквадратичного отклонения на чистом Python: {std\_}')  
std\_2 = statistics.stdev(x)  
print(f'Расчет среднеквадратичного отклонения с помощью statistics.stdev(): {std\_2}')  
np.std(y, ddof=1)  
print(f'Расчет среднеквадратичного отклонения с помощью NumPy: {np.std(y, ddof=1)}')

После этого, было найдено смещение:

#Смещение  
x = [8.0, 1, 2.5, 4, 28.0]  
n = len(x)  
mean\_ = sum(x) / n  
var\_ = sum((item - mean\_)\*\*2 for item in x) / (n - 1)  
std\_ = var\_ \*\* 0.5  
skew\_ = (sum((item - mean\_)\*\*3 for item in x)  
 \* n / ((n - 1) \* (n - 2) \* std\_\*\*3))  
print(f'Расчет смещения на чистом Python: {skew\_}')  
z, z\_with\_nan = pd.Series(x), pd.Series(x\_with\_nan)  
print(f'Расчет смещения с помощью Pandas: {z.skew()}')

Первый показатель, был найден, соответственно с помощью чистого Python, а второй с помощью Pandas, используя метод .skew().

Процентиль — такой элемент в наборе данных, так что p элементов в наборе данных меньше или равно его значению. Кроме того, (100 - p) элементов больше или равно этому значению. Если в наборе данных есть два таких элемента, то процентиль является их средним арифметическим. Расчет процентиля представлен внизу:

#Процентили  
x = [-5.0, -1.1, 0.1, 2.0, 8.0, 12.8, 21.0, 25.8, 41.0]  
print(f'Расчет процентилей с помощью statistics.quantiles(): {statistics.quantiles(x, n=2)}')  
statistics.quantiles(x, n=4, method='inclusive')  
print(f"Расчет процентилей с помощью statistics.quantiles(): {statistics.quantiles(x, n=4, method='inclusive')}")  
y = np.array(x)  
np.percentile(y, 5)  
print(f'Нахождение 5 процентиля : {np.percentile(y, 5)}')  
np.percentile(y, 95)  
print(f'Нахождение 95 процентиля : {np.percentile(y, 95)}')  
z, z\_with\_nan = pd.Series(y), pd.Series(y\_with\_nan)  
z.quantile(0.05)  
print(f'Нахождение процентиля используя метод .quantile(): {z.quantile(0.05)}')

Первый показатель был найден с помощью statistics.quantiles.В этом примере 8,0 — медиана x, а 0,1 и 21,0 — это 25-й и 75-й процентили выборки соответственно. Параметр n определяет количество результирующих процентилей с равной вероятностью, а метод определяет, как их вычислять. Следующие показатели, а именно 5 и 95 процентили, были найдены с помощью библиотеки NumPy, функции np.percentile(). И последний показатель был найдет используя метод .quantile().

Диапазон данных — это разница между максимальным и минимальным элементом в наборе данных. Эти показатели я нашел используя функцию [np.ptp()](https://docs.scipy.org/doc/numpy/reference/generated/numpy.ptp.html" \t "_blank):

#Диапазон  
np.ptp(y)  
np.ptp(z)  
np.ptp(y\_with\_nan)  
np.ptp(z\_with\_nan)  
print(f'Нахождение диапазона с помощью функции np.ptp(): {np.ptp(y),np.ptp(z),np.ptp(y\_with\_nan),np.ptp(z\_with\_nan)}')

И последнее, что я искал – сводка описательной статистики:

#Сводка описательной статистики  
result = scipy.stats.describe(y, ddof=1, bias=False)  
print(f'Сводка описательной статистики с помощью scipy.stats.describe(): {result}')  
result2 = z.describe()  
print(f'Сводка описательной статистики с помощью метода .describe() в Pandas: {result2}')

Первый показатель был найден с помощью scipy.stats.describe(). В качестве первого аргумента необходимо передать набор данных, который может быть представлен массивом NumPy, списком, кортежем или любой другой подобной структурой данных. Можно опустить ddof = 1, так как это значение по умолчанию и имеет значение только при расчете дисперсии. Указано bias = False для принудительного исправления асимметрии и эксцесса статистического смещения.

description() возвращает объект, который содержит следующую описательную статистику:

* **nobs** — количество наблюдений или элементов в вашем наборе данных;
* **minmax** — кортеж с минимальными и максимальными значениями;
* **mean** — среднее значение;
* **variance** — дисперсия;
* **skewness** — асимметрия;
* **kurtosis** — эксцесс вашего набора данных.

Второй показатель был найден с помощью метода .describe() библиотеки Pandas.