Практическое занятие 1
Тема «Элементы описательной статистики»

Описательная статистика – это описание и интегральные параметры наборов данных, к которым относятся центральные метрики (которые характеризуют центры концентрации данных – среднее арифметическое, медиана и мода) и метрики вариативности данных (которые характеризуют разброс значений – дисперсия и стандартное отклонение).

1) Установим необходимые библиотеки и вызовем их:

```
import math
import statistics
import numpy as np
import scipy.stats
import pandas as pd
```

2) Создадим списки x и x_with_nan:

```
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}')
```

```
Вывод исходных данных, которые содержатся в x:[10.0, 2, 2.5, 5, 26.0]
Вывод исходных данных, которые содержатся в x_with_nan:[10.0, 2, 2.5, nan, 5
, 26.0]
```

Списки x и x_with_nan 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.

3) Создадим 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}')
```

```
Вывод данных, которые содержатся в y и y_with_nan:[10.   2.   2.5  5.  26. ]
, [10.   2.   2.5  nan  5.  26. ]
Вывод данных, которые содержатся в z и в z_with_nan: 0    10.0
```

```
1     2.0
2     2.5
3     5.0
4    26.0
dtype: float64, 0    10.0
1     2.0
2     2.5
3     NaN
4     5.0
5    26.0
dtype: float64
```

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.

4) После формирования исходных данных, приступаем к расчету центральной метрики, а именно **среднего значения**:

```
#Расчет 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.mean(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}')
```

```
Расчет среднего значения, используя sum и len: 9.1
Расчет среднего значения, используя встроенные функции статистики Python (st
atistics.mean(x)): 9.1
Расчет среднего значения, используя встроенные функции статистики Python (st
atistics.fmean(x)): 9.1
Расчет среднего значения, который содержит значения nan, используя встроенны
е функции статистики Python (statistics.mean(x)): nan
Расчет среднего значения, используя NumPy: 9.1
Расчет среднего значения с помощью NumPy, игнорируя nan: 9.1
Расчет среднего значения объекта pd.Series: 9.1
```

Первое среднее значение было рассчитано на чистом Python, используя `sum()` и `len()`, 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.

5) Рассчитаем **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:

```
#Расчет 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()}')
```

```
Расчет средневзвешанного с помощью range: 6.8
Расчет средневзвешанного с помощью zip: 6.8
Расчет средневзвешанного с помощью np.average для массивово NumPy или серии
Pandas: 6.8
Расчет средневзвешанного с помощью поэлементного умножения w * y: 6.8
Расчет средневзвешанного для набора, который содержит nan : nan
```

Первое и второе средневзвешенное было рассчитано в чистом Python, используя комбинацию `sum()` с `range()` и `zip()`. 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()`, 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()`.

6) Рассчитаем **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)}')
```

```
Расчет гармонического среднего: 2.8195488721804507
Расчет гармонического среднего с помощью statistics.harmonic_mean(): 2.81954
8872180451
Расчет гармонического среднего, где есть nan: nan
Расчет гармонического среднего, где есть 0: 0
Расчет гармонического среднего с помощью scipy.stats.hmean(): 2.819548872180
4507
```

Как обычно, первое значение было рассчитано на чистом Python. Второе, третье и четвертое значения были рассчитаны с помощью функции `statistics.harmonic_mean()`. Последнее значение было рассчитано с использованием `scipy.stats.hmean`.

7) Рассчитаем **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)}')
```

```
Вычисление геометрического среднего: (1.5798787739851539-1.0248989283402022e
-07j)
Вычисление геометрического среднего с помощью scipy.stats.gmean(): nan
```

Первое значение было рассчитано на чистом Python. Второе и третье значения мы можем рассчитать с помощью функции

`statistics.geometric_mean()`,

но только, если у нас установлен Python 3.8 (поэтому эти строки закомментарены). И последнее значение было рассчитано с использованием `scipy.stats.gmean`.

8) **Медиана** – это средний элемент отсортированного набора данных. Расчет медианы представлен в следующем программном коде:

```
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}')
```

```
Расчет медианы: 6.0
Расчет медианы с помощью statistics.median(): 6.0
Расчет медианы с помощью statistics.median_low: 2.5
Расчет медианы с помощью statistics.median_high 6.0
Расчет медианы с помощью np.median: 6.0
```

Первое значение было рассчитано на чистом Python. Следующие три были найдены с использованием `statistics.median`. При этом, `median_low()` возвращает меньшее, а `median_high()` – большее среднее значение. И последнее значение было найдено с помощью NumPy и функции `np.median()`.

9) **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 в программном коде:

```
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}')
```

```
Вычисление моды: 2
Вычисление моды с помощью statistics.mode(): 2
Вычисление моды с помощью scipy.stats.mode(): ModeResult(mode=array([2]), co
unt=array([2]))
```

Первое значение, как обычно, получено, используя чистый Python. Вы использовали `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 значение было вычислено с помощью `statistics.mode()` (`statistics.multimode()` – закомментарено: поддерживается Python 3.8). Обратите внимание, `mode()` вернула одно значение (`multimode()` должна вернуть 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.

10) 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 является дисперсия. **Дисперсия** количественно определяет разброс данных и численно показывает, как далеко расположены точки данных от среднего значения.

Расчет дисперсии:

```
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}')
```

```
Оценка дисперсии на чистом Python: 94.04999999999998
Оценка дисперсии с помощью statistics.variance(): 94.04999999999998
Оценка дисперсии с помощью statistics.variance(), где есть nan: nan
Оценка дисперсии, используя NumPy с помощью np.var(): 94.04999999999998
Оценка дисперсии, используя NumPy с помощью метода .var(): 94.04999999999998
```

Первый метод расчета – использование Python. В целом, этого достаточно и можно правильно дать оценку дисперсии. Однако, более короткое и элегантное решение – использовать функцию statistics.variance() (как сделано это при расчете второго показателя). В результате мы получили тот же результат для дисперсии, что и в первом методе. Последние два показателя были рассчитаны с использованием NumPy, а именно функции np.var() и метода .var().

11) Рассчитаем среднеквадратичное отклонение. Стандартное отклонение выборки является еще одним показателем разброса данных. Он связан с оценкой дисперсии, 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. Расчет среднеквадратичного отклонения:

```
#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)}')
```

```
Расчет среднеквадратичного отклонения на чистом Python: 9.697937925146768
Расчет среднеквадратичного отклонения с помощью statistics.stdev(): 9.697937
925146768
Расчет среднеквадратичного отклонения с помощью NumPy: 9.697937925146768
```

12) Найдем 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()}')
```

```
Расчет смещения на чистом Python: 1.9470432273905929
Расчет смещения с помощью Pandas: 1.9470432273905924
```

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

13) **Процентиль** – такой элемент в наборе данных, что p элементов в этом 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.
Расчитаем процентиль:

```
#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)}')
```

```
Нахождение 5 процентиля : -3.44
Нахождение 95 процентиля : 34.919999999999995
Нахождение процентиля используя метод .quantile(): -3.44
```

Первый показатель может быть найден с помощью `statistics.quantiles` (`statistics.quantiles` – закомментарено: поддерживается Python 3.8). 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()`. Последний показатель был найдет используя метод `.quantile()`.

14) **Диапазон данных** – это разница между максимальным и минимальным элементом в наборе данных. Эти показатели найдем с использованием функции `np.ptp()`:

```
#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)}')
```

```
Нахождение диапазона с помощью функции np.ptp(): (46.0, 46.0, nan, 24.0)
```

15) 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}')
```

```
Сводка описательной статистики с помощью scipy.stats.describe(): DescribeRes
ult(nobs=9, minmax=(-5.0, 41.0), mean=11.622222222222222, variance=228.75194
444444446, skewness=0.9249043136685094, kurtosis=0.14770623629658886)
Сводка описательной статистики с помощью метода .describe() в Pandas: count
9.000000
mean     11.622222
std      15.124548
min      -5.000000
25%       0.100000
50%       8.000000
75%      21.000000
```

```
max      41.000000
dtype: float64
```

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.