Statistika Deskriptif

by Thio Perdana

Statistika deskriptif berkaitan dengan bagaimana kita menjelaskan dan murumuskan suatu data. Umumnya digunakan dua metode pendekatan:

1. **Pendekatan kuantitatif** menjelaskan data secara numerik.
2. **Pendekatan visual** mengilustrasikan data menggunakan chart, plot, histogram, atau grafik lainnya.

Kita dapat menerapkan statistika deskriptif ke dalam satu atau banyak data. Ketika kita menjelaskan dan merumuskan suatu data tunggal, maka kita sedang melakukan **analisa univariate**. Ketika kita mencari hubungan statistika antara sepasang data, maka kita sedang melakukan **analisa bivariate**. Begitu juga dengan **analisa multivariate** yang berarti kita menerapkannya pada banyak data sekaligus.

Dalam statistika, **populasi** merupakan sekumpulan elemen atau sesuatu yang ingin di amati, namun cakupan populasi biasa sangat besar sehingga akan sangat sulit untuk megumpulkan dan menganalisa datanya. Itulah mengapa ahli statistik biasanya mencoba mengambil kesimpulan mengenai suatu populasi dengan menguji dan memilih perwakilan dari populasi tersebut. Perwakilan dari populasi ini yang biasa kita sebut **sampel**.

Central Tendency

Central tendency (ukuran pemusatan data), biasanya kita lebih mengenal central tendency dengan Mean, Median, dan Modus.

Mean

Mean atau rerata merupakan nilai rata-rata dari populasi. Mean dinyatakan dengan rumus berikut:

$$\mu = \frac{1}{N}\sum_{i=1}^{N} x_i$$

Sebagai contoh, jika disediakan data set:

X = {2,3,2,4,5,6,7,4,6,5}

Maka mean dari X adalah:

$$\mu = \frac{2+3+2+4+5+6+7+4+6+5}{10} = 4{,}4$$

Selain menggunakan cara mean sederhana seperti di atas, kita bisa menggunakan mean berbobot (weighted mean). Mean atau rerata berbobot adalah rata-rata yang dihitung dengan memperhitungkan timbangan / bobot untuk setiap datanya. Rumus dari perhitungan rerata berbobot seperti berikut:

$$\bar{x} = \frac{\sum_{i=1}^{n} x_i w_i}{\sum_{1=1}^{n} w_i}$$

Keterangan:

$\bar{x}$ = rata-rata tertimbang

x_i = nilai data ke-i

w_i = bobot data ke-i

n = jumlah data

Median

Median merupakan datum yang terletak di tengah dari sebuah data. Syarat perhitungan median adalah diurutkan dari nilai terkecil hingga terbesar. Perhitungan median ini akan berbeda jika data berjumlah ganjil atau genap. Jika berjumlah ganjil, maka perhitungan median:

$$Median = X_{\frac{n+1}{2}}$$

Jika data berjumlah genap, maka perhitungan median menjadi seperti berikut:

$$Median = \frac{X_{\frac{n}{2}} + X_{(\frac{n}{2}+1)}}{2}$$

Modus

Modus merupakan nilai yang sering muncul. biasanya dihitung dengan menghitung frekuensi dari datum

Measurement of Spread

Measurement of Spread (ukuran sebaran data), cara yang paling umum untuk mengetahui ukuran sebaran data adalah dengan mengukur nilai range, interquartile range(IQR), variansi, kovariansi, dan standar deviasi dari data tersebut.

Range

Range atau rentang populasi menunjukkan nilai maksimum suatu populasi dikurangi dengan nilai minimum populasi tersebut.

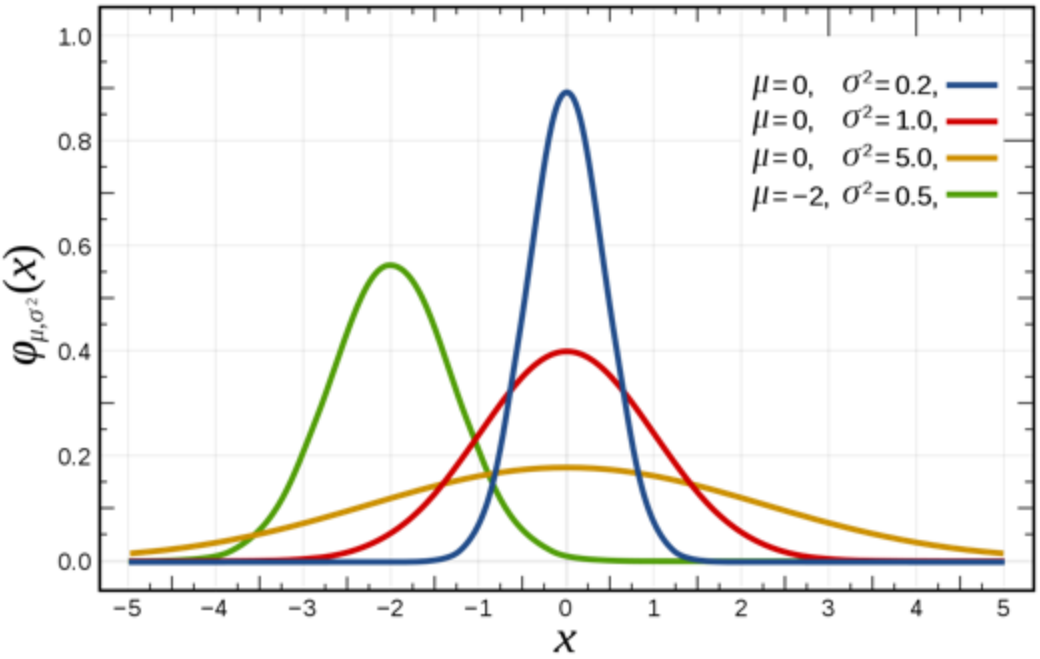
Variance

Variance atau varian adalah ukuran seberapa jauh kumpulan datum tersebar. Ketika nilai variance suatu populasi adalah nol artinya semua titik sampel bernilai sama. Variance yang bernilai kecil/rendah menandakan bahwa titik sampel cenderung dekat dengan mean. Sedangkan, jika nilai variance suatu populasi tinggi/besar menandakan bahwa titik sampel tersebar jauh dari mean (antara data poin tersebar jauh).

Variance dinyatakan dengan notasi σ^2 atau s^2 atau Var (X). Formula dari variance suatu populasi adalah sebagai berikut:

$$\sigma^2 = \frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2$$

Interpretasi dari rumusan di atas adalah jika nilai x cenderung terkonsentrasi di dekat mean, maka variansi kecil, sedangkan jika jauh dari mean maka variansinya besar. Dapat dilihat pada gambar berikut:



Pada kenyataannya, kita jarang sekali dapat mengukur seluruh populasi, yang bisa kita ukur hanya sampel, misalnya kita bisa saja mengukur tinggi badan peserta training data science, tapi kita tidak dapat mengukur tinggi badan seluruh manusia di bumi, hal ini akan mengakibatkan bias terhadap pengukuran yang dilakukan, oleh karena itu, agar tidak bias dalam mengukur nilai varian, maka N sebagai pembagi penjumlahan kuadrat (sum of squares) diganti dengan N-1 (degree of freedom). Rumusan varian sampel menjadi:

Standard Deviation

Standar deviasi atau simpangan baku adalah ukuran yang digunakan untuk mengukur jumlah variasi dari suatu data set. Standar deviasi diperoleh dari akar kuadrat dari variansi, dinyatakan sebagai berikut:

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2}$$

Standar deviasi dapat menjelaskan seberapa jauh simpangan data sampel dari mean.

Covariance

Covariance atau kovarian adalah ukuran dari hubungan atau relasi linear antara dua buah variabel. Jenis hubungan yang dapat terjadi antara dua buah variabel (berdasarkan kovariannya) yaitu:

- **Positif,** apabila nilai kovarian nya positif (>0), artinya ada relasi positif antara dua variabel. Sebagai contoh, jika nilai X meningkat maka nilai Y juga meningkat.
- **Negatif,** apabila nilai kovarian negatif (<0), artinya ada relasi negatif antara dua buah variabel. Sebagai contoh, jika jika nilai x meningkat maka nilai y menurun.
- **Zero,** apabila nilai kovarian nol, artinya tidak ada relasi antara dua variabel.

Formula kovarian antara X dan Y dirumuskan sebagai berikut:

$$Cov(X,Y) = \frac{1}{N}\sum_{i=1}^{N}(x_i - E(X))(y_i - E(Y))$$

Atau

$$Cov\ (X,Y) = \frac{1}{N}\sum_{i=1}^{N}(x_i - \mu_x)(y_i - \mu_y)$$

Dimana:

E(X), E(Y) = Nilai yang diharapkan atau probabilitas berbobot dari jumlah nilai

Jadi antara variansi dan kovarian sebenarnya tidak saling berhubungan. Namun untuk kasus dimana Y sama dengan X maka formula di atas menjadi:

$$Cov(X,X) = Var(X) \equiv \sigma^2(X) \equiv \sigma_X^2$$

Untuk kasus data 2 dimensi (x,y), apabila variabel yang sudah diketahui nilai kovariannya, lalu dicantumkan dlaam bentuk matriks, maka akan terbentuk matriks kovarian. Matriks kovarian dipresentasikan dalam bentuk seperti berikut:

$$S = \Sigma = \begin{bmatrix} \sigma(x,x) & \sigma(x,y) \\ \sigma(y,x) & \sigma(y,y) \end{bmatrix}$$

Atau

$$Cov(A) = \begin{bmatrix} \frac{\sum(x_i - \bar{X})(x_i - \bar{X})}{N} & \frac{\sum(x_i - \bar{X})(y - \bar{Y})}{N} \\ \frac{\sum(x_i - \bar{X})(y_i - \bar{Y})}{N} & \frac{\sum(y_i - \bar{Y})(y_i - \bar{Y})}{N} \end{bmatrix}$$

$$= \begin{bmatrix} Cov(X,X) & Cov(Y,Y) \\ Cov(X,Y) & Cov(Y,Y) \end{bmatrix}$$

Descriptive Statistics pada Python

Kita hanya akan menggunakan library statistics, NumPy, dan SciPy untuk menghitung nilai-nilai yang dihasilkan dari statistika deskriptif, library Pandas akan dipelajari pekan depan. Kita mulai dengan meng-import librarynya:

```
>>> import math
```

```
>>> import statistics
```

```
>>> import numpy as np
```

```
>>> import scipy.stats
```

Kita buat suatu data berupa python list yang berisi data numerik secara random:

```
>>> x = [8.0, 1, 2.5, 4, 28.0]
```

```
>>> x_dgn_nan = [8.0, 1, 2.5, math.nan, 4, 28.0]
```

```
>>> x
```

```
[8.0, 1, 2.5, 4, 28.0]
```

```
>>> x_dgn_nan
```

```
[8.0, 1, 2.5, nan, 4, 28.0]
```

Sekarang kita punya dua list, x dan x_dgn_nan, keduanya hampir sama, bedanya yaitu x_dgn_nan memiliki nilai nan (not a number). Sangat penting bagaimana python menghadapi nilai nan ini. Dalam data science, missing value sangat sering terjadi, dan biasanya nilai yang hilang ini diganti dengan nan. Sekarang kita coba buat np.ndarray objek dari variabel x dan x_dgn_nan:

```
>>> y, y_dgn_nan = np.array(x), np.array(x_dgn_nan)

>>> y

array([ 8. ,  1. ,  2.5, 4. , 28. ])

>>> y_dgn_nan

array([ 8. ,  1. ,  2.5,  nan,  4. , 28. ])
```

Menghitung Ukuran Pemusatan Data

Kita akan nenpelajari bagaimana menghitung ukuran pemusatan data ini menggunakna python.

Mean (rataan) Kita dapat menghitung mean hanya dengan python tanpa meng-import library apapun menggunakan sum() dan len():

```
>>> mean_ = sum(x) / len(x)

>>> mean_

8.7
```

Kita juga bisa menggunakan fungsi bawaan dari library statistics python:

```
>>> mean_ = statistics.mean(x)
>>> mean_
8.7
>>> mean_ = statistics.fmean(x)
>>> mean_
8.7
```

Fungsi fmean() mulai dikenalkan pada python 3.8 sebagai alternatif untuk perhitungan yang lebih cepat dan selalu menghasilkan nilai float. Namun, fungsi mean() dan fmean() akan menghasilkan nilai nan jika di dalam variabel yang dihitungnya berisi nilai nan:

```
>>> mean_ = statistics.mean(x_dgn_nan)
>>> mean_
nan
```

```
>>> mean_ = statistics.fmean(x_dgn_nan)

>>> mean_ nan
```

Jika kita ingin mengabaikan nilai nan yang ada pada variabel tersebut maka dapat menggunakan np.nanmean():

```
>>> np.nanmean(y_dgn_nan)

8.7
```

Weighted Mean atau rataan berbobot merupakan generalisasi dari rataan aritmatika sehingga kita dapat menentukan kontribusi relatif (bobot) dari setiap data point untuk penghitungan hasilnya. Kita dapat menghitung weighted mean menggunakan python dengan menggabungkan antara sum() dengan range() atau zip() seperti pada contoh berikut:

```
>>> x = [8.0, 1, 2.5, 4, 28.0]

>>> 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)

>>> wmean

6.95
```

```
>>> wmean = sum(x_ * w_ for (x_, w_) in zip(x, w)) / sum(w)

>>> wmean

6.95
```

Tetapi jika kita memiliki dataset yang cukup besar maka NumPy memberikan soolusi yang lebih baik untuk penghitungan weighted mean ini menggunakan np.average():

```
>>> y, w = np.array(x), np.array(w)

>>> wmean = np.average(y, weights=w)

>>> wmean

6.95
```

Hati-hati ketika melakukan perhitungan untuk data berisi nilai nan.

```
>>> w = np.array([0.1, 0.2, 0.3, 0.0, 0.2, 0.1])

>>> np.average(y_dgn_nan, weights=w)

nan
```

Harmonic Mean atau rataan harmonis dapat dirumuskan dengan $n/\Sigma_i(1/x_i)$ dimana i=1, 2, …, n dan n adalah jumlah data point pada dataset x. Kita bisa menghitungnya menggunakan python:

```
>>> hmean = len(x) / sum(1 / item for item in x)

>>> hmean

2.7613412228796843
```

Kita juga bisa menggunakan statistics.harmonic_mean():

```
>>> hmean = statistics.harmonic_mean(x)

>>> hmean

2.7613412228796843
```

Jika terdapat nilai nan, maka statistics.harmonic_mean() akan menghasilkan nilai nan, dan jika terdapat nilai negatif, maka akan terjadi error dalam perhitungannya:

```
>>> statistics.harmonic_mean(x_dgn_nan)

nan

>>> statistics.harmonic_mean([1, 0, 2])

0
```

```
>>> statistics.harmonic_mean([1, 2, -2])  #StatisticsError
```

Cara ketiga adalah dengan menggunakan scipy.stats.hmean():

```
>>> scipy.stats.hmean(y)
```

```
2.7613412228796843
```

Geometric Mean atau rataan geometris merupakan akar pangkat-n dari seluruh hasil perkalian elemen x_i sejumlah n pada dataset x, dapat dirumuskan dengan $\sqrt[n]{(\Pi_i x_i)}$, dimana dimana i=1, 2, …, n dan n adalah jumlah data point pada dataset x. Kita dapat menggunakan python untuk menghitung nilai rataan geometris ini sebagai berikut:

```
>>> gmean = 1
```

```
>>> for item in x:
```

```
...    gmean *= item
```

```
...
```

```
>>> gmean **= 1 / len(x)
```

```
>>> gmean
```

```
4.677885674856041
```

Pada python 3.8 kita dapat menggunakan statistics.geometric_mean(), yang akan mengubah semua nilai pada dataset menjadi float kemudian menghitung rataan geometrisnya:

```
>>> gmean = statistics.geometric_mean(x)

>>> gmean

4.67788567485604
```

Dengan menggunakan scipy.stats.gmean():

```
>>> scipy.stats.gmean(y)

4.67788567485604
```

Jika terdapat nilai nan pada dataset maka gmean() akan menghasilkan nan. Jika terdapat satu saja nilai 0, maka akan menghasilkan rataan geometris 0 disertai peringatan.

Median suatu sampel merupakan nilai tengah dari dataset yang telah diurutkan. Dataset tersebut dapat diurutkan secara naik atau turun. Jika jumlah elemen n dari dataset adalah ganjil maka median adalah elemen pada posisi 0.5(n+1). Jika n genap, maka mediannya adalah rataan aritmatika dari dua nilai tengah, yaitu yang berada pada posisi 0.5n dan 0.5n+1. Perbedaan utama antara sifat mean dan median dari suatu dataset adalah hubungannya dengan outlier pada dataset tersebut. Nilai mean sangat dipengaruhi oleh outlier sedangkan nilai median hampir tidak dipengaruhi atau bahkan tidak dipengaruhi sama sekali. Perhatikan gambar berikut:

Dataset bagian atas memiliki nilai 1, 2.5, 4, 8, dan 28. Nilai mean dari dataset tersebut adalah 8.7 dan nilai mediannya adalah 4. Dataset bagian bawah menunjukkan bagaimana perubahan nilai mean ketika nilai paling kanan pada dataset atas yang memiliki nilai 28 di pindahkan.

Berikut adalah salah satu cara mencari nilai median menggunakan python:

```
>>> 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])

...

>>> median_

4
```

Langkah penting pada proses di atas adalah:

1. Mengurutkan elemen-elemen pada dataset
2. Mencari elemen tengah pada dataset yang telah diurutkan

Kita juga bisa mendapatkan nilai median dengan statistics.median():

```
>>> median_ = statistics.median(x)

>>> median_

4
```

Cara yang lain untuk mendapatkan nilai median adalah menggunakan np.median():

```
>>> median_ = np.median(y)

>>> median_

4.0
```

Mode atau modus dari suatu sampel dataset adalah nilai yang paling banyak muncul dalam dataset tersebut. Dengan python kita dapat menghasilkan nilai modus suatu dataset sebagai berikut:

```
>>> u = [2, 3, 2, 8, 12]

>>> mode_ = max((u.count(item), item) for item in set(u))[1]

>>> mode_

2
```

Kita dapat mencari modus dari suatu daset menggunakan statistics.mode(), dan juga menggunakan statistics.multimode() (mulai dikenalkan pada python 3.8) jika nilai modusnya tidak hanya satu:

```
>>> mode_ = statistics.mode(u)

>>> mode_

2

>>> mode_ = statistics.multimode(u)

>>> mode_

[2]
```

Dapat dilihat bahwa mode() menghasilkan nilai tunggal, sedangkan multimode() menghasilkan list yang berisi nilai modus. Bahkan mode() akan menghasilkan error jika dalam dataset tersebut terdapat dua modus:

```
>>> v = [12, 15, 12, 15, 21, 15, 12]

>>> statistics.mode(v)  # StatisticsError

>>> statistics.multimode(v)

[12, 15]
```

Bagaimana jika dalam dataset terdapat nilai nan? Bisa dicoba sendiri hehe..

Untuk menghasilkan nilai modus jika menggunakan scipy.stats.mode():

```
>>> u, v = np.array(u), np.array(v)
```

```
>>> mode_ = scipy.stats.mode(u)
```

```
>>> mode_
```

```
ModeResult(mode=array([2]), count=array([2]))
```

```
>>> mode_ = scipy.stats.mode(v)
```

```
>>> mode_
```

```
ModeResult(mode=array([12]), count=array([3]))
```

Jika terdapat lebih dari satu nilai modus, scipy.stats.mode() akan menjadikan nilai terkecil sebagai modus.

Menghitung Ukuran Sebaran Data

Ukuran pemusatan saja tidak cukup untuk menjelaskan suatu data, kita juga perlu menghitung ukuran sebaran data. Beberapa ukuran sebaran data yang perlu diketahui yaitu:

Variance atau variansi. Menghitung variansi menggunakan python dapat dilakukan dengan cara berikut:

```
>>> n = len(x)
```

```
>>> mean_ = sum(x) / n
```

```
>>> var_ = sum((item - mean_)**2 for item in x) / (n - 1)
```

```
>>> var_
```

```
123.19999999999999
```

Cara yang lebih singkat dan elegan adalah menggunakan statistics.variance():

```
>>> var_ = statistics.variance(x)
```

```
>>> var_
```

```
123.2
```

Bagaimana jika terdapat nilai nan dalam dataset? Silahkan dicoba sendiri hehe..

Cara yang lain yaitu menggunakan fungsi np.var() atau metode .var():

```
>>> var_ = np.var(y, ddof=1)
```

```
>>> var_
```

```
123.19999999999999
```

```
>>> var_ = y.var(ddof=1)
```

```
>>> var_
```

```
123.19999999999999
```

Penting untuk mendefinisikan nilai ddof=1. Parameter ini digunakan agar perhitungan nilai s^2 sesuai yaitu menggunakan n-1 sebagai pembagi bukan n saja. Untuk dataset yang memiliki nilai nan, kita dapat mengabaikan nilai nan tersebut dengan np.nanvar():

```
>>> np.nanvar(y_dgn_nan, ddof=1)
```

```
123.19999999999999
```

Standar Deviasi atau simpangan baku. Menghitung standar deviasi menggunakan python dapat dilakukan dengan cara:

```
>>> std_ = var_ ** 0.5
```

```
>>> std_
```

```
11.099549540409285
```

Kita juga bisa menggunakan statistics.dev():

```
>>> std_ = statistics.stdev(x)

>>> std_

11.099549540409287
```

Jika kita menggunakan Numpy, perhatikan untuk menggunakan fungsi yang sesuai jika terdapat nilai nan:

```
>>> np.std(y, ddof=1)

11.099549540409285

>>> y.std(ddof=1)

11.099549540409285

>>> np.std(y_dgn_nan, ddof=1)

nan

>>> y_dgn_nan.std(ddof=1)

nan

>>> np.nanstd(y_dgn_nan, ddof=1)

11.099549540409285
```

Skewness, nilai dari skewness dapat dihasilkan menggunakan python dengan cara:

```
>>> 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))

>>> skew_

1.9470432273905929
```

Kita juga bisa menggunakan scipy.stats.skew():

```
>>> y, y_dgn_nan = np.array(x), np.array(x_dgn_nan)

>>> scipy.stats.skew(y, bias=False)

1.9470432273905927
```

```
>>> scipy.stats.skew(y_dgn_nan, bias=False)
```

```
nan
```

suatu dataset dapat dianggap simetris jika memiliki nilai skewness mendekati 0, yaitu antara -0.5 dan 0.5.

Percentiles ke-p dari sekumpulan data adalah nilai dimana p% dari data tersebut berada dibawahnya.

Setiap data memiliki tiga nilai kuartil, yang membagi data menjadi 4 bagian sama besar.

1. Kuartil pertama (Q1) adalah persentil ke-25 dari data
2. Kuartil kedua (Q2) adalah persentil ke-50 dari data yang juga merupakan median dari data tersebut.
3. Kuartil ketiga (Q3) adalah persentil ke-75 dari data

Nilai persentil dapat dicari menggunakan np.percentile():

```
>>> x = [-5.0, -1.1, 0.1, 2.0, 8.0, 12.8, 21.0, 25.8, 41.0]
```

```
>>> y = np.array(x)
```

```
>>> np.percentile(y, 5)
```

```
-3.44
```

```
>>> np.percentile(y, 95)
```

```
34.919999999999995
```

Jika kita ingin mengabaikan nilai nan pada data maka digunakan np.nanpercentile():

```
>>> y_dgn_nan = np.insert(y, 2, np.nan)

>>> y_dgn_nan

array([-5. , -1.1,  nan,  0.1,  2. ,  8. , 12.8, 21. , 25.8, 41. ])

>>> np.nanpercentile(y_dgn_nan, [25, 50, 75])

array([ 0.1,  8. , 21. ])
```

Ranges dari data adalah selisih antara elemen maksimum dan elemen minimum pada suatu dataset. Kita dapat menghitungnya dengan fungsi np.ptp():

```
>>> np.ptp(y)

46.0

>>> np.ptp(y_dgn_nan)

nan

Alternatifnya, kita bisa menggunakan fungsi bawaan python dan NumPy:

>>> np.amax(y) - np.amin(y)
```

```
46.0
```

```
>>> np.nanmax(y_dgn_nan) - np.nanmin(y_dgn_nan)
```

```
46.0
```

```
>>> y.max() - y.min()
```

```
46.0
```

Menghitung Korelasi Antara Sepasang Data

Korelasi atau hubungan antara sepasang data dapat dilihat dengan menghitung:

Covariance, dengan menggunakan fungsi dari python kovariansi dapat dihitung sebagai berikut:

```
>>> n = len(x)
```

```
>>> mean_x, mean_y = sum(x) / n, sum(y) / n
```

```
>>> cov_xy = (sum((x[k] - mean_x) * (y[k] - mean_y) for k in range(n))
```

```
...            / (n - 1))
```

```
>>> cov_xy
```

```
19.95
```

Dengan menggunakan np.cov() dari NumPy kita akan mendapatkan matriks kovariansi:

```
>>> cov_matrix = np.cov(x_, y_)

>>> cov_matrix

array([[38.5        , 19.95      ],

[19.95       , 13.91428571]])
```

Dimana nilai 38.5 atau posisi atas kiri merupakan nilai variansi dari x, dan nilai 13.91 merupakan nilai variansi dari y, dan dua nilai lainnya merupakan nilai kovariansi antara x dan y, yaitu 19.95.

Correlation coefficient, untuk menghitung koefisien korelasi hanya menggunakan fungsi bawaan python adalah sebagai berikut:

```
>>> var_x = sum((item - mean_x)**2 for item in x) / (n - 1)

>>> var_y = sum((item - mean_y)**2 for item in y) / (n - 1)

>>> std_x, std_y = var_x ** 0.5, var_y ** 0.5

>>> r = cov_xy / (std_x * std_y)

>>> r

0.861950005631606
```

Library scipy.stats memiliki fungsi pearsonr() yang menghitung nilai dari koefisien korelasi dan juga nilai p-value nya:

```
>>> r, p = scipy.stats.pearsonr(x_, y_)

>>> r

0.861950005631606

>>> p

5.122760847201171e-07
```

Jika ingin menggunakan NumPy, dapat memakai fungsi np.corrcoef() dengan argumen x_ dan y_, maka didapatkan matriks koefisien korelasinya:

```
>>> corr_matrix = np.corrcoef(x_, y_)

>>> corr_matrix

array([[1.          , 0.86195001],

[0.86195001, 1.        ]])
```

Nilai 1 pada matriks tersebut menunjukkan koefisien korelasi antara satu argumen dengan dirinya sendiri, sedangkan dua nilai yang lain menunjukkan koefisien korelasi antara kedua argumen yakni x_ dan y_.

```
>>> r = corr_matrix[0, 1]

>>> r

0.8619500056316061

>>> r = corr_matrix[1, 0]

>>> r

0.861950005631606
```

Kita juga bisa menggukanan fungsi scipy.stats.linregress() yang akan menghasilkan beberapa nilai, salah satunya adalah koefisien korelasinya:

```
>>> scipy.stats.linregress(x_, y_)

LinregressResult(slope=0.5181818181818181,                intercept=5.714285714285714,
rvalue=0.861950005631606, pvalue=5.122760847201164e-07, stderr=0.06992387660074979)

>>> result = scipy.stats.linregress(x_, y_)

>>> r = result.rvalue

>>> r

0.861950005631606
```

PERKENALAN NUMPY

NumPy (Numerical Python) adalah salah satu library utama dalam ekosistem pemrograman Python yang digunakan untuk komputasi numerik. NumPy menyediakan dukungan untuk array multidimensional dan operasi matematika pada array tersebut, yang sangat berguna dalam pemrosesan data, ilmu data, ilmu komputer, dan pemrograman ilmiah secara umum.

NumPy memiliki beberapa fitur utama:

Array NumPy: Struktur data inti yang digunakan untuk menyimpan data numerik dalam bentuk array multidimensional, seperti vektor, matriks, dan tensor.

Operasi Matematika: NumPy memungkinkan Anda melakukan berbagai operasi matematika pada array dengan mudah, seperti penambahan, pengurangan, perkalian, pembagian, serta fungsi matematika lainnya.

Broadcasting: Konsep yang memungkinkan NumPy untuk berinteraksi dengan array berbeda bentuk secara alami.

Fungsi Universal (ufuncs): Fungsi yang memungkinkan Anda melakukan operasi elemen-wise pada array NumPy.

Integrasi dengan data dan analisis statistik: NumPy sering digunakan sebagai komponen inti dalam ekosistem data science Python, digunakan bersama dengan library seperti Pandas, SciPy, dan Matplotlib.

NumPy memiliki peran yang sangat penting dalam pemrograman ilmiah dan data science, dan berikut adalah alasan-alasannya:

Kinerja: NumPy menyediakan implementasi array yang efisien, sehingga memungkinkan operasi matematika dan manipulasi data yang cepat, bahkan untuk data besar. Ini sangat penting dalam analisis data dan pemodelan matematika.

Konsistensi Data: Dalam data science, kita sering harus berurusan dengan data dalam bentuk matriks atau array multidimensional. NumPy memberikan struktur data yang konsisten dan kuat untuk menyimpan dan mengelola data ini.

Operasi Matematika: NumPy memungkinkan kita melakukan operasi matematika yang kompleks pada data numerik dengan mudah. Ini mencakup operasi matriks, statistik, transformasi, dan banyak lagi.

Integrasi dengan Libraries Lain: NumPy bekerja dengan baik dengan library data science lainnya seperti Pandas (untuk analisis data), SciPy (untuk ilmu pengetahuan), dan Matplotlib (untuk visualisasi data). Ini menciptakan ekosistem yang kuat untuk analisis data.

Kompatibilitas dengan Hardware: NumPy dapat diintegrasikan dengan library komputasi numerik berperforma tinggi lainnya seperti BLAS (Basic Linear Algebra Subprograms) dan LAPACK (Linear Algebra Package) untuk optimalisasi kinerja.

Untuk melakukan instalasi (dalam kasus numpy tidak tersedia) kita dapat menginstal NumPy dengan mudah menggunakan manajer paket Python, seperti pip. Berikut langkah-langkahnya:

1. Pastikan Anda memiliki Python terinstal. Anda bisa mendownload Python dari situs resmi (https://www.python.org/downloads/).
2. Buka terminal atau command prompt Anda.

3. Untuk menginstal NumPy, ketik perintah berikut dan tekan Enter:
   `pip install numpy`
4. pip akan mengunduh dan menginstal NumPy serta semua dependensinya. Setelah selesai, Anda akan memiliki NumPy terinstal di lingkungan Python Anda.

Setelah menginstal NumPy, Anda dapat mengimpor library ini di kode Python Anda dengan perintah `import numpy as np`, dan Anda siap untuk mulai menggunakan semua fitur dan kemampuan NumPy dalam pengembangan Anda.

ARRAY NUMPY

Array NumPy adalah struktur data inti yang digunakan dalam library NumPy (Numerical Python). Ini adalah objek yang digunakan untuk menyimpan dan mengelola data numerik dalam bentuk array multidimensional. Array NumPy memiliki beberapa karakteristik kunci:

Multidimensional: Array NumPy dapat memiliki dimensi yang lebih dari satu. Ini berarti Anda dapat memiliki array satu dimensi (seperti vektor), array dua dimensi (seperti matriks), atau bahkan array dengan lebih banyak dimensi (seperti tensor).

Homogen: Semua elemen dalam array NumPy harus memiliki tipe data yang sama. Ini memastikan efisiensi dalam operasi matematika dan penyimpanan data.

Ukuran yang Dinamis: Anda dapat mengubah ukuran array NumPy setelah dibuat, tetapi dimensi utama array ini tetap konsisten.

Operasi Matematika: Array NumPy mendukung berbagai operasi matematika elemen-wise, seperti penambahan, pengurangan, perkalian, pembagian, dan fungsi matematika lainnya.

Efisiensi Kinerja: Array NumPy dirancang untuk memberikan kinerja yang tinggi dalam pemrosesan data numerik. Implementasinya dioptimalkan untuk melakukan operasi pada data dalam skala besar dengan cepat.

Broadcasting: NumPy memiliki konsep yang disebut "broadcasting" yang memungkinkan operasi antara array dengan bentuk yang berbeda. Ini membuat kode lebih fleksibel dan lebih mudah dibaca.

Indeks dan Slicing: Anda dapat dengan mudah mengakses elemen-elemen dalam array NumPy menggunakan indeks dan melakukan slicing untuk mengambil bagian-bagian dari array.

Operasi Statistik dan Agregasi: NumPy menyediakan berbagai fungsi untuk melakukan statistik dan agregasi data, seperti menghitung mean, median, varian, dan banyak lagi.

Manipulasi Data: Anda dapat melakukan berbagai operasi pada array NumPy, termasuk reshaping (mengubah bentuk array), penggabungan array, dan filtering menggunakan mask (boolean indexing).

Integrasi dengan Library Lain: Array NumPy dapat dengan mudah diintegrasikan dengan library lain dalam ekosistem Python, seperti Pandas, SciPy, dan Matplotlib.

Array NumPy sangat berguna dalam berbagai bidang, termasuk ilmu data, ilmu komputer, pemodelan matematika, dan pemrosesan gambar, karena memungkinkan manipulasi data numerik secara efisien dan mudah.

Membuat array NumPy

Anda dapat membuat array NumPy dengan beberapa cara berikut:

1. **Menggunakan `np.array()`:** Cara paling sederhana adalah dengan menggunakan `np.array()` untuk mengubah sebuah list atau tuple menjadi array NumPy. Contohnya:

```
import numpy as np
my_list = [1, 2, 3, 4, 5]
my_array = np.array(my_list)
```

2. **Menggunakan Fungsi NumPy:**
    - `np.zeros()`: Membuat array dengan elemen nol.
    - `np.ones()`: Membuat array dengan elemen satu.
    - `np.arange()`: Membuat array dengan rentang nilai.
    - `np.linspace()`: Membuat array dengan sejumlah nilai terdistribusi linier.
    - `np.random.rand()`: Membuat array dengan elemen acak.
3. **Menggunakan Metode NumPy:**
    - `np.zeros_like()`: Membuat array nol dengan bentuk yang sama seperti array yang ada.
    - `np.ones_like()`: Membuat array satu dengan bentuk yang sama seperti array yang ada.
    - `np.copy()`: Membuat salinan array.

Menjelajahi array NumPy

Anda dapat menjelajahi array NumPy dengan menggunakan berbagai metode dan atribut, seperti:

- `shape`: Atribut yang mengembalikan tupel yang menyatakan bentuk (jumlah baris, jumlah kolom, dll.) dari array.
- `size`: Atribut yang mengembalikan jumlah elemen dalam array.
- `dtype`: Atribut yang menunjukkan tipe data elemen dalam array.
- `ndim`: Atribut yang menunjukkan jumlah dimensi dalam array.
- Indexing: Anda dapat mengakses elemen array dengan menggunakan indeks
- Slicing: Anda dapat menggunakan slicing untuk mengambil potongan dari array

Sifat-sifat array: bentuk, tipe data, dimensi

- **Bentuk (Shape)**: Merupakan atribut yang menyatakan dimensi array, seperti (baris, kolom) untuk array dua dimensi.
- **Tipe Data (Data Type)**: Menunjukkan tipe data elemen dalam array, seperti `int`, `float`, `string`, dll.
- **Dimensi (Number of Dimensions - ndim)**: Merupakan jumlah dimensi dalam array, seperti 1 untuk array satu dimensi, 2 untuk array dua dimensi, dan seterusnya.

Operasi dasar pada array NumPy

Array NumPy memungkinkan berbagai operasi matematika dan manipulasi data, seperti:

- Operasi elemen-wise: Anda dapat melakukan operasi matematika pada semua elemen array, seperti penambahan, pengurangan, perkalian, dan pembagian.
- Operasi matriks: NumPy mendukung operasi matriks seperti dot product dan perkalian matriks.
- Fungsi matematika: NumPy memiliki berbagai fungsi matematika bawaan seperti `np.sum()`, `np.mean()`, `np.min()`, `np.max()`, dan banyak lagi untuk analisis statistik.

Dengan pemahaman ini, Anda dapat mulai bekerja dengan array NumPy dan melakukan berbagai operasi pada data numerik dengan Python.

INDEXING DAN SLICING

Mengakses elemen array

Anda dapat mengakses elemen-elemen dalam array NumPy dengan menggunakan indeks. Indeks dimulai dari 0 untuk elemen pertama dalam array. Sebagai contoh:

```
import numpy as np


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


# Mengakses elemen pertama

elemen_pertama = my_array[0]  # Hasilnya 1


# Mengakses elemen ketiga
```

```
elemen_ketiga = my_array[2]  # Hasilnya 3
```

Anda juga dapat mengakses elemen dalam array multidimensional dengan menggunakan indeks untuk setiap dimensi. Misalnya:

```
import numpy as np

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


# Mengakses elemen di baris pertama dan kolom kedua

elemen = my_array[0, 1]  # Hasilnya 2
```

Slicing: mengambil bagian dari array

Slicing digunakan untuk mengambil potongan (subset) dari array. Anda dapat melakukan slicing pada array NumPy dengan menggunakan notasi `start:stop:step`, di mana:

- `start` adalah indeks elemen pertama yang ingin Anda ambil.
- `stop` adalah indeks elemen setelah elemen terakhir yang ingin Anda ambil (elemen ini tidak akan disertakan).
- `step` adalah langkah yang digunakan untuk mengambil elemen (opsional, jika tidak ditentukan, defaultnya adalah 1).

Contoh-contoh slicing:

```
import numpy as np


my_array = np.array([1, 2, 3, 4, 5])
```

```
# Mengambil elemen kedua hingga keempat
subset = my_array[1:4]  # Hasilnya [2, 3, 4]

# Mengambil elemen dengan indeks genap
subset = my_array[::2]  # Hasilnya [1, 3, 5]
```

Slicing juga dapat digunakan pada array multidimensional:

```
import numpy as np

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

# Mengambil baris kedua
baris_kedua = my_array[1, :]  # Hasilnya [4, 5, 6]

# Mengambil kolom kedua
kolom_kedua = my_array[:, 1]  # Hasilnya [2, 5]
```

Mengganti nilai elemen dalam array

Anda dapat mengganti nilai elemen dalam array dengan mengakses elemen tersebut dan menetapkan nilai baru kepadanya. Misalnya:

```
import numpy as np
```

```
my_array = np.array([1, 2, 3, 4, 5])


# Mengganti nilai elemen kedua menjadi 10

my_array[1] = 10
```

Anda juga dapat mengganti nilai dalam array multidimensional:

```
import numpy as np


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


# Mengganti nilai elemen di baris pertama dan kolom kedua

my_array[0, 1] = 20
```

Dengan pemahaman ini, Anda dapat dengan mudah mengakses elemen, melakukan slicing, dan mengganti nilai elemen dalam array NumPy sesuai dengan kebutuhan Anda.

OPERASI MATEMATIKA

Operasi Elemen-wise

Operasi elemen-wise adalah operasi yang diterapkan pada setiap elemen dalam array NumPy secara terpisah. Dalam konteks ini, elemen-wise berarti setiap elemen dalam array yang sesuai berinteraksi satu sama lain sesuai dengan aturan operasi yang Anda tentukan.

Contoh operasi elemen-wise:

```
import numpy as np


# Membuat dua array

array_a = np.array([1, 2, 3, 4])

array_b = np.array([5, 6, 7, 8])


# Penambahan elemen-wise

result = array_a + array_b  # Hasilnya [6, 8, 10, 12]


# Pengurangan elemen-wise

result = array_a - array_b  # Hasilnya [-4, -4, -4, -4]


# Perkalian elemen-wise

result = array_a * array_b  # Hasilnya [5, 12, 21, 32]


# Pembagian elemen-wise

result = array_a / array_b  # Hasilnya [0.2, 0.33333333, 0.42857143, 0.5]
```

Operasi Matriks (Dot Product)

Operasi matriks dalam NumPy, seperti dot product, digunakan untuk mengalikan dua array dengan aturan perkalian matriks. Untuk menghitung dot product, Anda dapat menggunakan `np.dot()` atau metode `.dot()` pada array NumPy.

Contoh operasi dot product:

```
import numpy as np


# Membuat dua matriks
matrix_a = np.array([[1, 2], [3, 4]])
matrix_b = np.array([[5, 6], [7, 8]])


# Menghitung dot product
result = np.dot(matrix_a, matrix_b)
# Hasilnya adalah:
# [[19, 22],
#  [43, 50]]
```

Fungsi Matematika Umum

NumPy menyediakan berbagai fungsi matematika umum yang dapat diterapkan pada elemen-elemen array atau matriks. Fungsi ini termasuk, tetapi tidak terbatas pada:

- `np.sum()`: Menghitung jumlah elemen dalam array.
- `np.mean()`: Menghitung rata-rata dari elemen-elemen array.
- `np.min()`: Mengambil nilai terkecil dalam array.

- `np.max()`: Mengambil nilai terbesar dalam array.
- `np.sqrt()`: Menghitung akar kuadrat dari setiap elemen dalam array.
- `np.exp()`: Menghitung eksponensial dari setiap elemen dalam array.

Contoh penggunaan fungsi matematika:

```
import numpy as np


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


# Menghitung sum

total = np.sum(my_array)  # Hasilnya 15


# Menghitung mean

average = np.mean(my_array)  # Hasilnya 3.0


# Menghitung akar kuadrat

square_root = np.sqrt(my_array)  # Hasilnya [1. 1.41421356 1.73205081 2.
2.23606798]
```

Dengan operasi elemen-wise, operasi matriks, dan berbagai fungsi matematika, NumPy memungkinkan Anda untuk melakukan berbagai operasi matematika pada data dalam array NumPy dengan mudah dan efisien.

STATISTIKA DASAR

Menghitung statistik dasar pada data adalah salah satu tugas umum dalam analisis data dan ilmu data. NumPy menyediakan berbagai fungsi yang memudahkan penghitungan statistik dasar pada array NumPy. Berikut adalah beberapa statistik dasar yang umum dihitung:

Rata-rata (Mean): Rata-rata adalah nilai tengah dari sejumlah data. Anda dapat menghitung rata-rata dari sebuah array NumPy menggunakan fungsi `np.mean()` atau `np.average()`.

```
import numpy as np


data = np.array([1, 2, 3, 4, 5])
mean = np.mean(data)  # Menghitung rata-rata
```

Median: Median adalah nilai tengah dalam data yang telah diurutkan. Anda dapat menghitung median dengan menggunakan fungsi `np.median()`.

```
import numpy as np


data = np.array([1, 2, 3, 4, 5])
median = np.median(data)  # Menghitung median
```

Modus (Mode): Modus adalah nilai yang muncul paling sering dalam data. NumPy tidak memiliki fungsi bawaan untuk menghitung modus, tetapi Anda dapat menggunakan library lain seperti `scipy.stats` untuk menghitung modus.

Nilai Terkecil (Minimum): Nilai terkecil dalam data dapat dihitung dengan fungsi `np.min()`.

```
import numpy as np


data = np.array([1, 2, 3, 4, 5])
minimum = np.min(data)  # Menghitung nilai terkecil
```

Nilai Terbesar (Maximum): Nilai terbesar dalam data dapat dihitung dengan fungsi `np.max()`.

```
import numpy as np


data = np.array([1, 2, 3, 4, 5])
maximum = np.max(data)  # Menghitung nilai terbesar
```

Jumlah (Sum): Jumlah adalah hasil dari penjumlahan semua nilai dalam data. Anda dapat menghitung jumlah dengan fungsi `np.sum()`.

```
import numpy as np


data = np.array([1, 2, 3, 4, 5])
total = np.sum(data)  # Menghitung jumlah
```

Variansi (Variance): Variansi mengukur sejauh mana data tersebar dari rata-rata. Anda dapat menghitung variansi dengan fungsi `np.var()`.

```
import numpy as np


data = np.array([1, 2, 3, 4, 5])
variance = np.var(data)  # Menghitung variansi
```

Deviasi Standar (Standard Deviation): Deviasi standar adalah akar kuadrat dari variansi dan mengukur sejauh mana data tersebar dari rata-rata. Anda dapat menghitung deviasi standar dengan fungsi `np.std()`.

```
import numpy as np


data = np.array([1, 2, 3, 4, 5])
std_deviation = np.std(data)  # Menghitung deviasi standar
```

Dengan menggunakan fungsi-fungsi NumPy ini, Anda dapat dengan mudah menghitung berbagai statistik dasar pada data dalam array NumPy. Hal ini sangat berguna dalam analisis data, pemodelan, dan pemahaman karakteristik data.