

# Machine Learning

Presented by



## Daftar Isi

A. Modelling	03
B. Introduction to Unsupervised Learning	06
C. Introduction to Supervised Learning	22

# Prologue

Sebelumnya kita sudah membahas penerapan data preprocessing yang kita tahu aktivitas tersebut perlu diperhatikan dalam Data Science terutama sebelum membangun model machine learning. Kali ini kita akan mencoba mengulik aktivitas selanjutnya yang bisa dikatakan “inti” dalam Data Science , yaitu membangun model machine learning.

Apabila di modul selanjutnya kita membahas bagaimana menampilkan informasi berdasarkan data di masa lalu melalui Exploratory Data Analysis (EDA), kini kita mencoba membahas bagaimana membangun model machine learning untuk menganalisis karakteristik data. Modul kali ini kita akan mempelajari secara lengkap proses penerapan data preprocessing seperti:

- I. Modelling
- II. Introduction to Unsupervised Learning
- III. Introduction to Supervised Learning

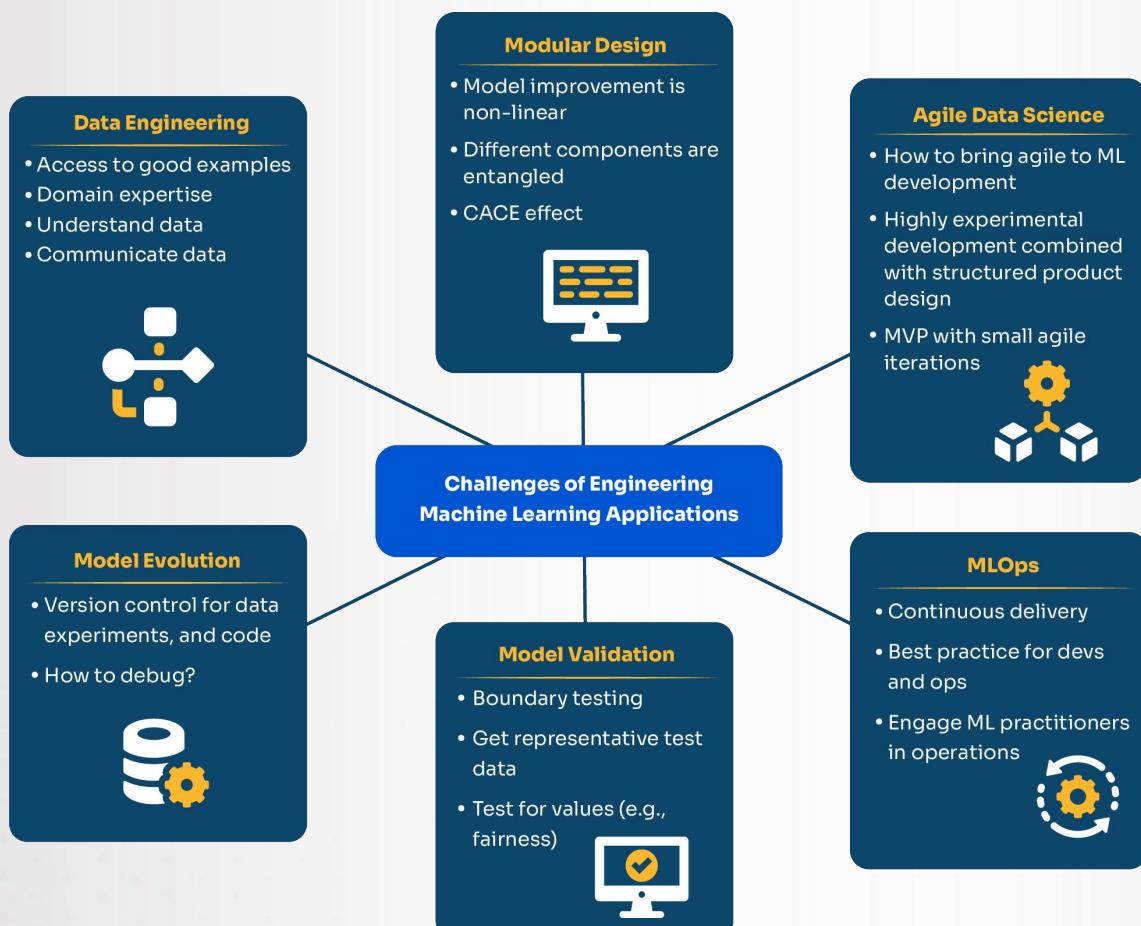
Harapannya, kita bisa memperoleh gambaran bagaimana membangun suatu model machine learning berdasarkan berbagai pendekatan beserta tools yang bisa digunakan melalui modul ini.

# Modelling

Dalam konteks data, modelling adalah **representasi visual dari suatu sistem informasi** yang bertujuan mengilustrasikan tipe data yang digunakan dan disimpan dalam sistem, menemukan hubungan antara tipe data, mengelompokkan data, serta mengatur data dari segi format dan atributnya.

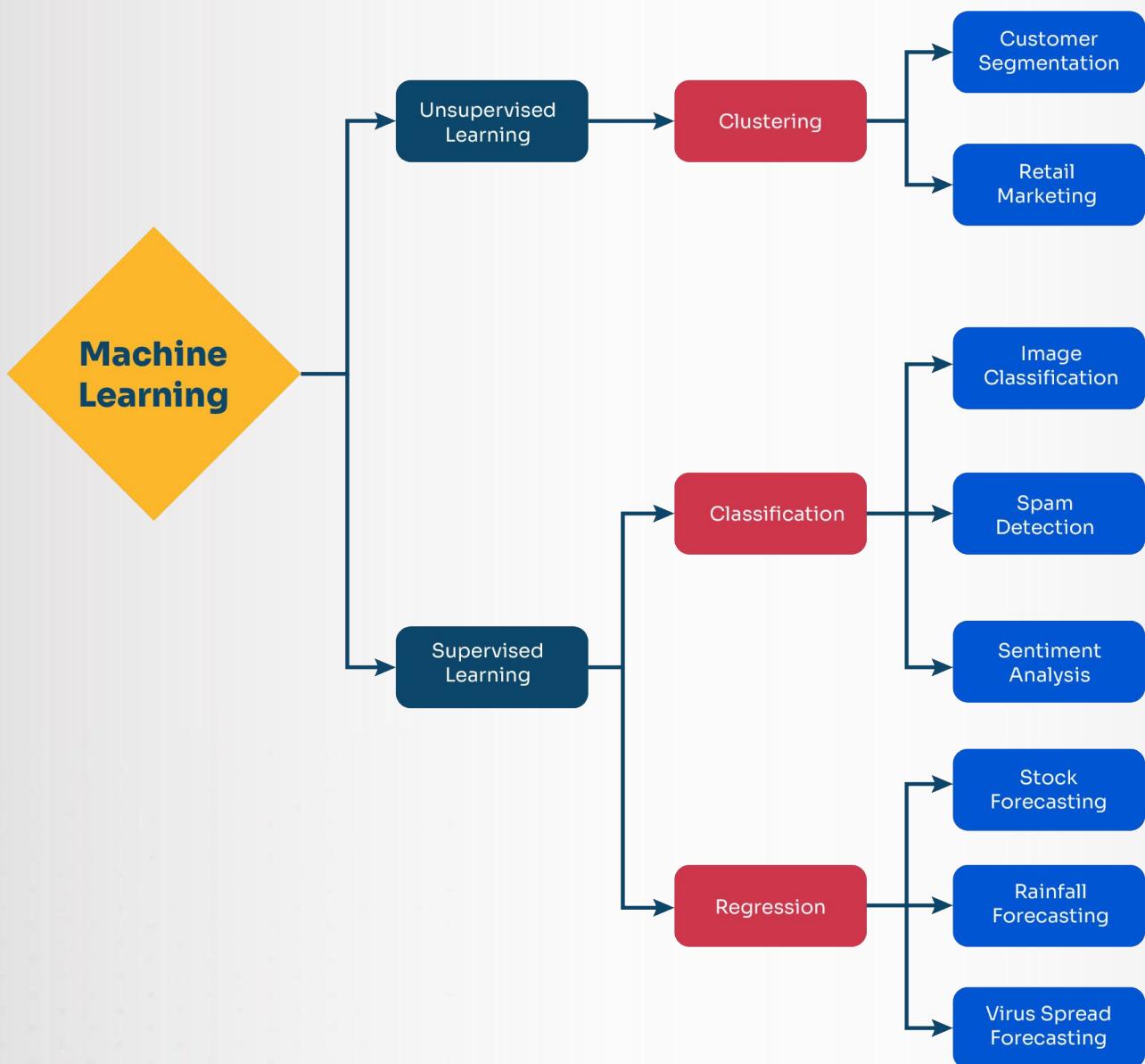
Sebagai contoh, data modelling dapat diterapkan pada sistem reservasi hotel. Hotel dapat didaftarkan dalam sistem dan pelanggan dapat memesan kamar di hotel pilihan mereka. Aktivitas berikutnya yang meliputi tipe kamar, reservasi, layanan, dan tagihan dapat direpresentasikan melalui diagram sederhana.

**Model adalah sarana membantu pemahaman terhadap masalah** yang ada dan komponen dari suatu metode yang digunakan dalam aktivitas pengembangan. Model juga dapat berfungsi sebagai alat bantu komunikasi yang dapat digunakan oleh peneliti untuk menjelaskan sesuatu.



Dalam pembuatan model machine learning, terdapat dua pendekatan populer yang sering digunakan, yaitu unsupervised learning dan supervised learning seperti terlihat pada gambar di bawah. Perbedaan mendasar di antara keduanya adalah cara kerja mereka dalam mengelompokkan atau mengklasifikasikan data.

**Unsupervised learning mampu bekerja tanpa label kelas** atau desired output dalam implementasinya. Sementara itu, **supervised learning membutuhkan label kelas** dalam implementasinya. Selanjutnya, kita akan membahas satu persatu dua pendekatan itu.

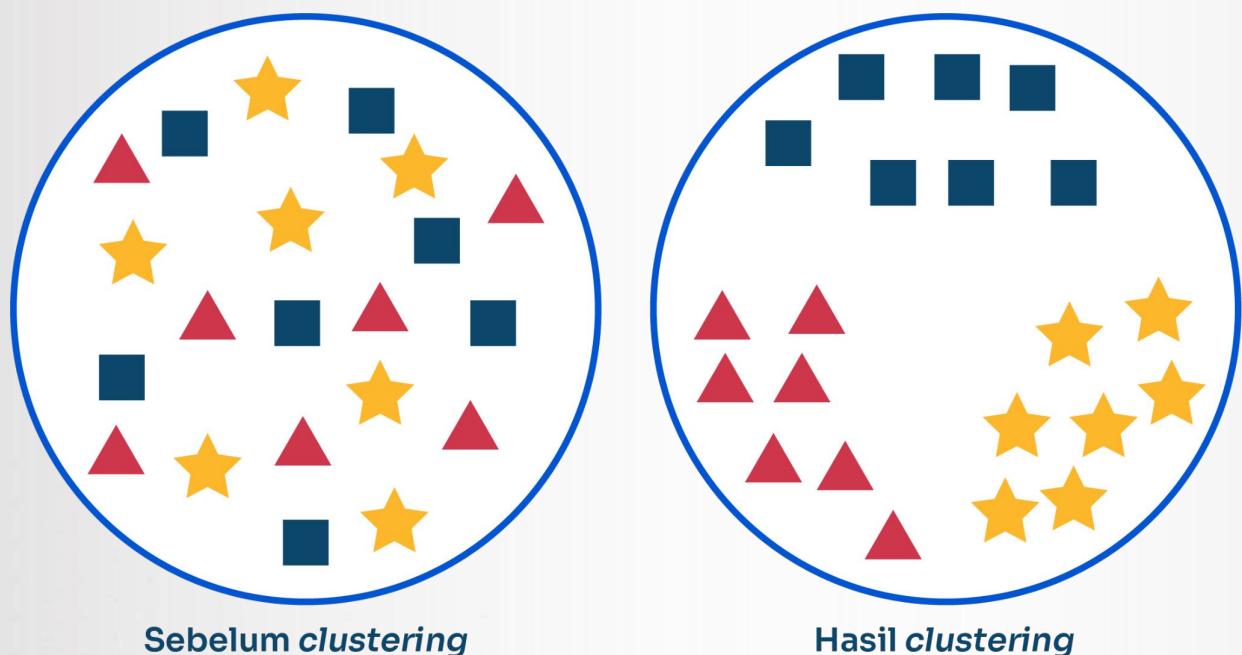


# Introduction to Unsupervised Learning

Proses pengelompokan data tanpa desired output biasa disebut clustering (klasterisasi). Dalam clustering, setiap fitur tidak berkorespondensi dengan kelas tertentu.

Jika kita mencoba analogikan desired output adalah ilmu yang diajarkan oleh guru, maka unsupervised learning bekerja tanpa “bimbingan” guru tersebut. Model unsupervised learning memang dibangun untuk mengenali pola dataset tanpa adanya desired output yang diberikan. Pendekatan unsupervised learning **tidak perlu membagi data menjadi training dataset dan testing dataset**.

Salah satunya metode penerapan unsupervised learning adalah K-Means dan DBSCAN dimana keduanya memiliki karakteristik masing-masing.



## K-Means

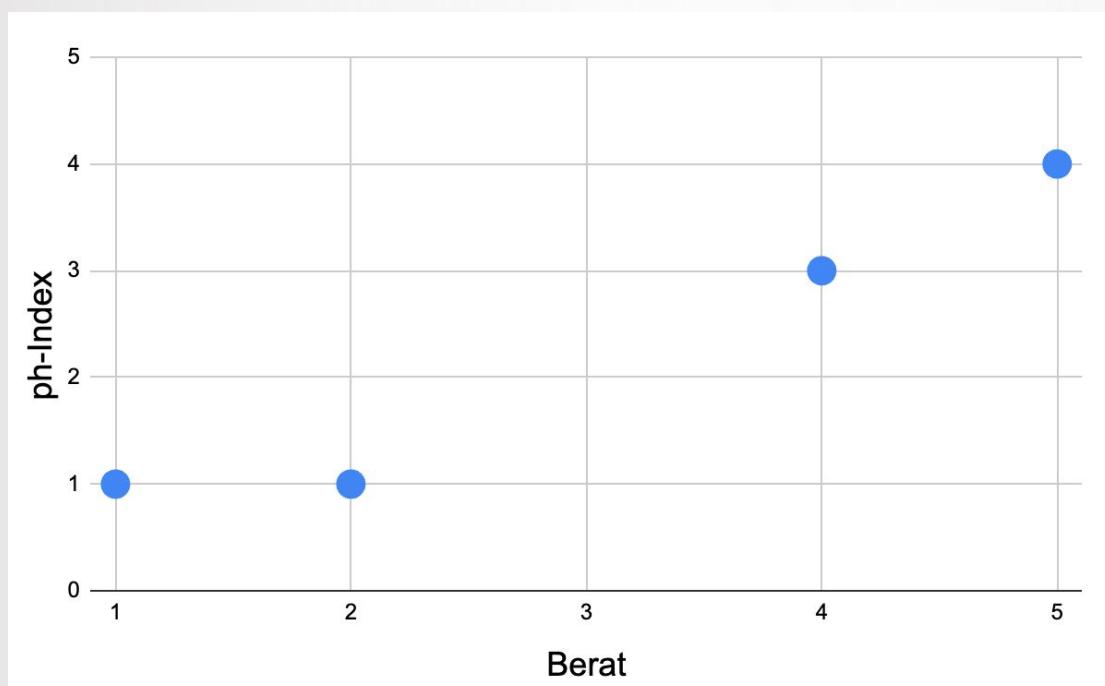
K-Means **mengelompokkan data ke dalam cluster yang memiliki kesamaan** dan ketidakmiripan dengan objek-objek milik cluster lain. Makna ‘K’ pada K-Means merupakan jumlah kelompok data (cluster) yang terbentuk. Misalnya, K = 3 mengacu pada tiga cluster. Oleh karena itu, kita harus bereksperimen pada berapa banyak jumlah cluster yang ideal. Biasanya algoritma ini mengelompokkan dengan meminimalisir jarak antara titik dengan pusat clusternya. Model ini memiliki kelemahan dalam mengelompokkan dataset yang memiliki banyak outlier. Secara garis besar, K-Means dapat dilakukan dalam 5 langkah, yaitu:

1. Menentukan k
2. Menentukan titik awal centroid sebanyak k
3. Memasukkan setiap data ke dalam cluster berdasarkan centroid terdekat
4. Mengubah posisi centroid dari masing-masing cluster
5. Ulangi langkah pertama hingga sudah tidak ada perubahan centroid

Untuk lebih memahami bagaimana K-Means bekerja, kita coba pada data di tabel di bawah yuk! Data berikut adalah data tipe obat yang memiliki atribut berat dan kandungan pH. Kita akan mencoba mengelompokkan data ini menjadi dua buah cluster.

Obat	Berat	ph-Index
A	1	1
B	2	1
C	3	3
D	4	4

Perhatikan gambar di bawah, gambar berikut adalah visualisasi persebaran titik data berdasarkan atribut ‘Berat’ dan ‘ph-Index’

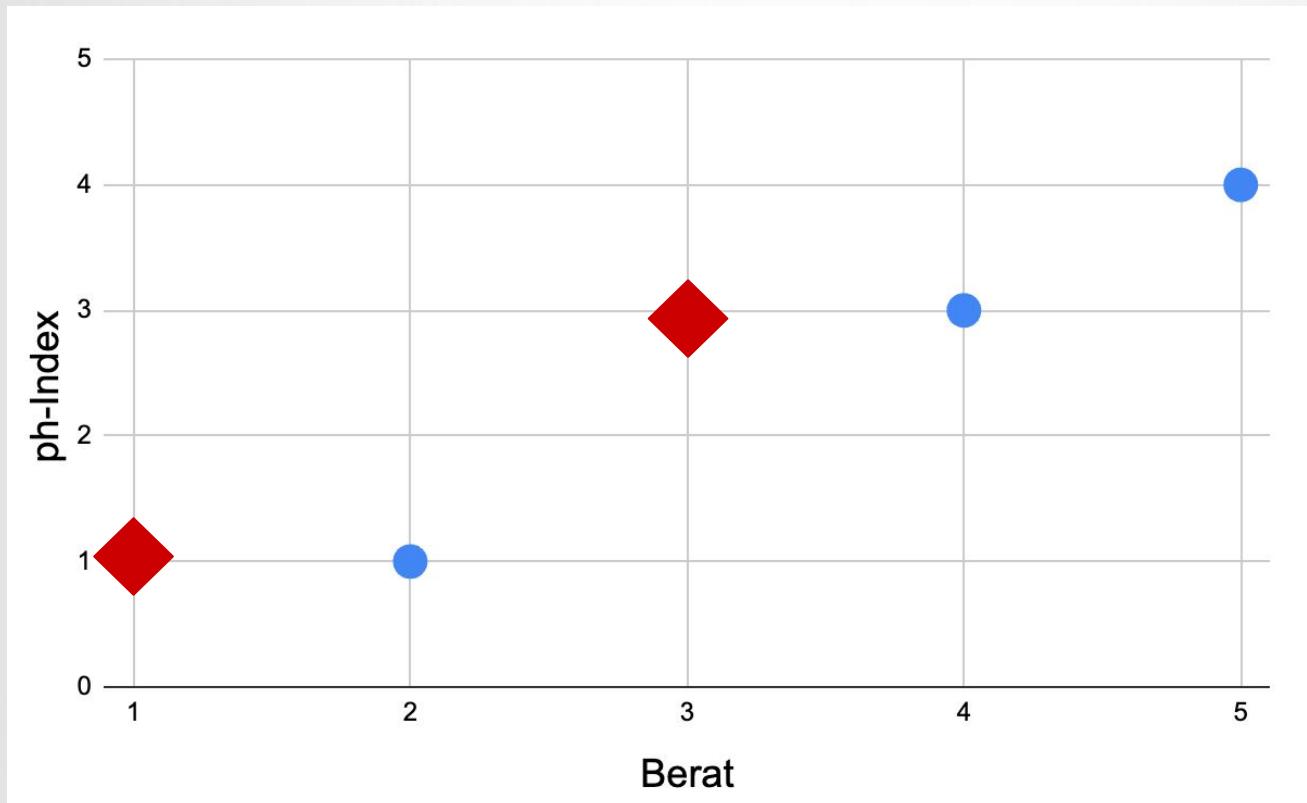


## 1. Menentukan k

Kita akan mengelompokkan data ke dalam dua kelompok. Maka, nilai k pada model K-Means kali ini adalah 2.

## 2. Menentukan titik awal centroid sebanyak k

Kita menentukan titik awal centroid masing-masing k. Apabila terdapat k sebanyak 2, maka kita akan menentukan titik awal sebanyak 2 kali. Centroid adalah pusat atau titik data pada suatu cluster. Pada percobaan pertama penerapan K-Means, penentuan posisi centroid dilakukan secara acak atau tidak ada perhitungan khusus. Misalkan titik centroid 1 kita posisikan pada koordinat (1,1) dan centroid 2 pada (3,3) seperti pada gambar di bawah.



### 3. Memasukkan setiap data ke dalam cluster berdasarkan centroid terdekat

Selanjutnya, kita memasukkan tiap data yang kita miliki ke dalam cluster berdasarkan centroid terdekat. Misalkan, data A memiliki jarak paling dekat dengan centroid 1, maka data A masuk ke dalam cluster 1. Sebaliknya, apabila data A memiliki jarak paling dekat dengan centroid 2, maka akan masuk ke cluster 2. Perhitungan jarak dilakukan menggunakan persamaan Euclidean Distance sebagai berikut:

$$d = \sqrt{((X_i - X_0)^2 + (Y_i - Y_0)^2)}$$

Catatan:

- $X_i \rightarrow$  Posisi sumbu X titik data pengamatan
- $Y_i \rightarrow$  Posisi sumbu Y titik data pengamatan
- $X_0 \rightarrow$  Posisi sumbu X titik centroid
- $Y_0 \rightarrow$  Posisi sumbu Y titik centroid

Diketahui bahwa posisi centroid kita adalah (1,1) untuk centroid 1 dan (3,3) untuk centroid 2.

Jarak tiap titik data dengan centroid adalah sebagai berikut:

- Obat A → (1,1)
  - Jarak dengan Centroid 1 →  $\sqrt{(1-1)^2 + (1-1)^2} = 0$
  - Jarak dengan Centroid 2 →  $\sqrt{(1-3)^2 + (1-3)^2} = 2,83$

**Obat A masuk cluster 1 karena lebih dekat dengan centroid 1** dengan jarak hanya 0

- Obat B → (2,1)
  - Jarak dengan Centroid 1 →  $\sqrt{(2-1)^2 + (1-1)^2} = 1$
  - Jarak dengan Centroid 2 →  $\sqrt{(2-3)^2 + (1-3)^2} = 2,24$

**Obat B masuk cluster 1 karena lebih dekat dengan centroid 1** dengan jarak hanya 1

- Obat C → (4,3)

Jarak dengan masing-masing centroid:

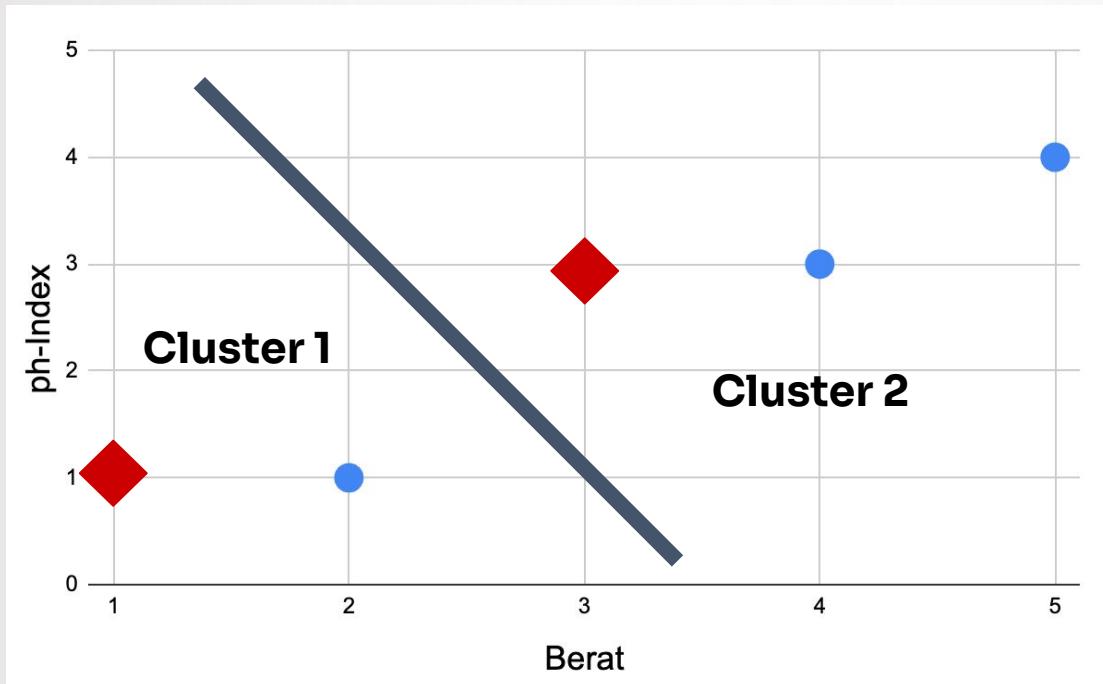
- Jarak dengan Centroid 1 →  $\sqrt{(4-1)^2 + (3-1)^2} = 2,24$
- Jarak dengan Centroid 2 →  $\sqrt{(4-3)^2 + (3-3)^2} = 1$

**Obat C masuk cluster 2 karena lebih dekat dengan centroid 2** dengan jarak hanya 1

- Obat D → (5,4)
  - Jarak dengan Centroid 1 →  $\sqrt{(5-1)^2 + (4-1)^2} = 5$
  - Jarak dengan Centroid 2 →  $\sqrt{(5-3)^2 + (5-3)^2} = 2,83$

**Obat D masuk cluster 2 karena lebih dekat dengan centroid 2** dengan jarak hanya 2,83

Setelah mengetahui jarak terdekat tiap data dengan centroid, kita dapat memasukkan data ke dalam cluster. Cluster yang terbentuk dapat dilihat pada gambar di bawah



#### 4. Mengubah posisi centroid

Mengubah posisi centroid dilakukan melalui menghitung rata-rata nilai anggota cluster. Apabila kita perhatikan gambar di atas, anggota tiap centroid adalah sebagai berikut:

- Anggota centroid 1
  - Pada sumbu x, anggota pertama berada koordinat 1 sementara anggota kedua berada pada koordinat 2
  - Pada sumbu y, masing-masing anggota berada pada koordinat 1
- Anggota centroid 2
  - Pada sumbu x, anggota pertama berada koordinat 4 sementara anggota kedua berada pada koordinat 5
  - Pada sumbu y, anggota pertama berada koordinat 3 sementara anggota kedua berada pada koordinat 4

Kita menentukan posisi centroid terbaru dengan menghitung rata-rata nilai anggota centroid seperti berikut:

- **Posisi centroid 1 terbaru**

Berhubung centroid 2 memiliki anggota yang berada pada koordinat:

- 1 dan 2 pada sumbu x
- 1 dan 1 pada sumbu y

maka, posisi centroid terbaru adalah  $(1+2)/2 = 1,5$  pada sumbu x dan  $(1+1)/2 = 1$  pada sumbu y

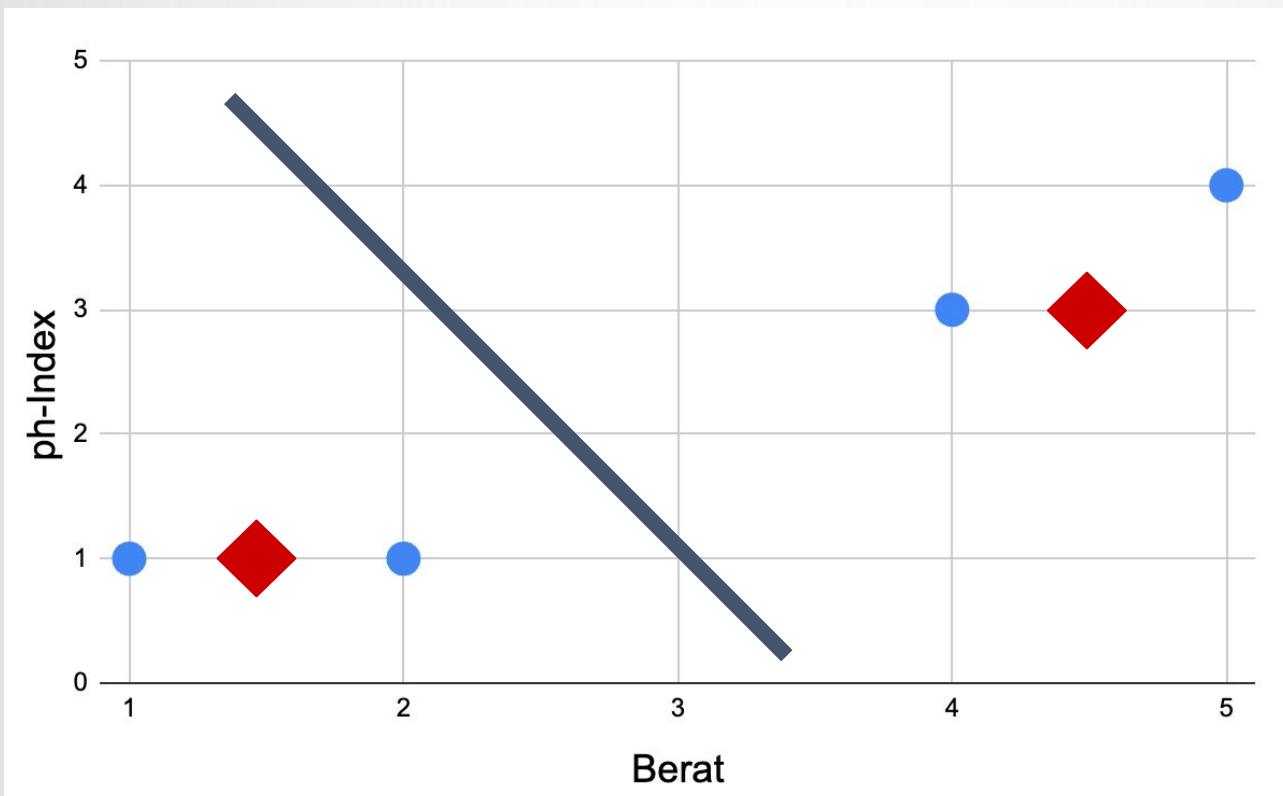
- **Posisi centroid 2 terbaru**

Berhubung centroid 2 memiliki anggota yang berada pada koordinat:

- 4 dan 5 pada sumbu x
- 3 dan 4 pada sumbu y

maka, posisi centroid terbaru adalah  $(4+5)/2 = 4,5$  pada sumbu x dan  $(3+4)/2 = 3,5$  pada sumbu y

Posisi centroid terbaru dapat dilihat pada gambar di bawah:



Berhubung posisi centroid telah berubah, kita memperbarui anggota cluster menggunakan langkah ketiga seperti yang telah kita lakukan sebelumnya.

- Obat A → (1,1)
  - Jarak dengan Centroid 1 →  $\sqrt{(1-1,5)^2+(1-1)^2} = 0,5$
  - Jarak dengan Centroid 2 →  $\sqrt{(1-4,5)^2+(1-3,5)^2} = 4,30$

**Obat A masuk cluster 1 karena lebih dekat dengan centroid 1** dengan jarak hanya 0,5

- Obat B → (2,1)
  - Jarak dengan Centroid 1 →  $\sqrt{(2-1,5)^2+(1-1)^2} = 0,5$
  - Jarak dengan Centroid 2 →  $\sqrt{(2-4,5)^2+(1-3,5)^2} = 3,54$

**Obat B masuk cluster 1 karena lebih dekat dengan centroid 1** dengan jarak hanya 0,5

- Obat C → (4,3)

Jarak dengan masing-masing centroid:

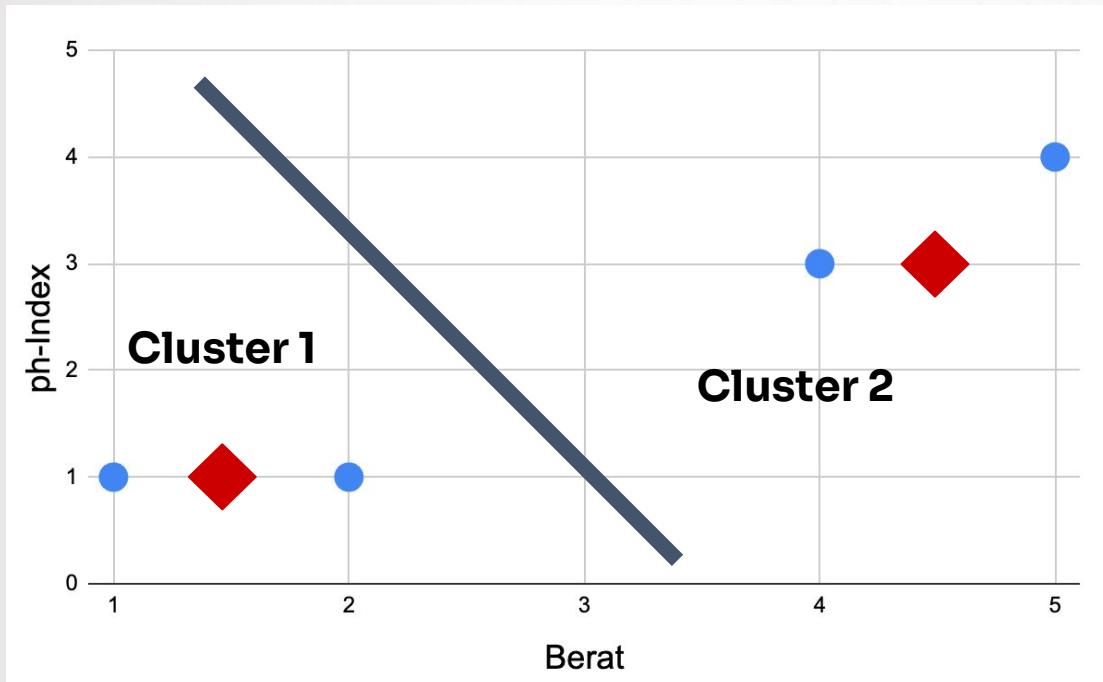
- Jarak dengan Centroid 1 →  $\sqrt{(4-1,5)^2+(3-1)^2} = 3,20$
- Jarak dengan Centroid 2 →  $\sqrt{(4-4,5)^2+(3-3,5)^2} = 0,71$

**Obat C masuk cluster 2 karena lebih dekat dengan centroid 2** dengan jarak hanya 0,71

- Obat D → (5,4)
  - Jarak dengan Centroid 1 →  $\sqrt{(5-1,5)^2+(4-1)^2} = 4,61$
  - Jarak dengan Centroid 2 →  $\sqrt{(5-4,5)^2+(5-3,5)^2} = 1,58$

**Obat D masuk cluster 2 karena lebih dekat dengan centroid 2** dengan jarak hanya 1,58

Setelah mengetahui jarak terdekat tiap data dengan posisi centroid terbaru, kita mengetahui anggota tiap cluster. Perhatikan gambar di bawah, terlihat tidak ada perubahan anggota yang signifikan dari sebelumnya.



Proses mengubah posisi centroid dilakukan secara berulang dan berhenti ketika:

- Konvergen

Dikatakan konvergen apabila tidak ada perubahan signifikan pada anggota cluster

- Sudah memenuhi jumlah iterasi yang diinginkan

Kita bisa mengulangi proses sesuai iterasi yang diinginkan. Apabila kita menginginkan sebanyak 5 iterasi, maka proses mengubah centroid diulangi sebanyak 5 kali

Kali ini kita akan mencoba menerapkan K-Means untuk mengelompokkan dataset iris menggunakan python:

## 1. Memasukkan Package dan Dataset

Seperti yang sudah kita bahas pada modul sebelumnya, package adalah sekumpulan modul atau fungsi atau kode bahasa pemrograman python yang memudahkan kita menulis perintah. Kali ini kita membutuhkan package pandas, numpy, seaborn, dan matplotlib.

Terlihat juga kita memasukkan package StandardScaler() untuk standarisasi dataset. Sementara itu, kita akan menggunakan fungsi KMeans yang disediakan oleh sklearn. Kita juga membutuhkan fungsi silhouette\_score untuk evaluasi model clustering. Tuliskan script berikut untuk memasukkan package dan dataset.

```
● ● ●

# Importing the Library
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler

# Importing the KMeans Library
from sklearn.cluster import KMeans
# Importing the Model Evaluation Library
from sklearn.metrics import silhouette_score

# Importing the Dataset
df = sns.load_dataset("iris")
```

Data yang akan kita coba kelompokkan dapat dilihat pada gambar berikut:

sepal_length	sepal_width	petal_length	petal_width	species
5.1	3.5	1.4	0.2	setosa
4.9	3.0	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
4.6	3.1	1.5	0.2	setosa
5.0	3.6	1.4	0.2	setosa

## 2. Standarisasi Dataset

Selanjutnya, kita bisa melakukan standarisasi dataset menggunakan fungsi **.StandardScaler()**. Secara garis besarnya, kita akan mengubah data kita menjadi dalam rentang nilai yang sama. Namun yang menjadi catatan, langkah ini lebih bersifat rekomendasi saja karena biasanya tergantung dari bagaimana proses Exploratory Data Analysis (EDA) yang dilakukan



```
# Drop the 'Species' column
X = df.drop(columns= 'species')

X_scaled = StandardScaler().fit_transform(X)
```

## 3. Membentuk model K-Means

Kita akan mencoba konfigurasi jumlah cluster pada K-Means sebesar 4. Maka, kita memasukkan nilai parameter(n\_clusters pada K-Means sebanyak 4.



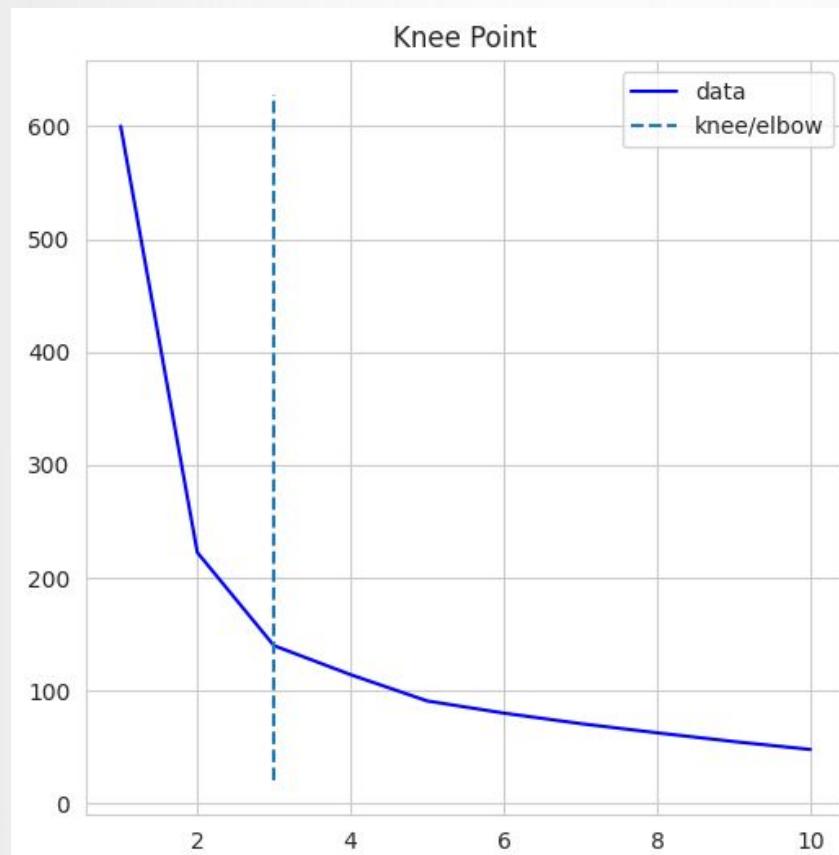
```
n_cluster = 3
kmean = KMeans(n_clusters= n_cluster)
# fitting model
kmean.fit(X_scaled)
```

## 4. Mencari jumlah cluster optimal

Mencari jumlah optimal dari cluster ( $k$ ) dengan menggunakan metode **Elbow Curve**. Elbow Curve merupakan salah satu metode yang bisa digunakan untuk menemukan jumlah optimal dari cluster ( $k$ ), yang langkah-langkah penggerjaan adalah sebagai berikut.

```
# Mengambil nilai SSE dari masing-masing jumlah cluster
sse = []
for k in range(1, 11):
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(X_scaled)
    sse.append(kmeans.inertia_)

# Mencari elbow/knee (jumlah cluster paling optimal)
kl = KneeLocator(range(1, 11), sse, curve="convex",
                 direction="decreasing")
print("The ideal number of the cluster: {}"
      "clusters".format(kl.knee))
kl.plot_knee()
```



Pada metode elbow curve, lokasi ‘tikungan’ yang terbentuk di plot, pada umumnya dianggap sebagai indikator jumlah cluster yang tepat. Akan tetapi nilai k optimal yang diperoleh dari metode elbow curve, sering kali bersifat “ambigu” atau belum pasti akan menghasilkan jumlah cluster (k) yang optimal. Oleh karena itu, alternatif lainnya adalah menggunakan Silhouette Analysis.

**Silhouette analysis** adalah pengukuran seberapa dekat setiap titik pada sebuah cluster dengan titik lain di cluster-nya. Semakin tinggi nilai rata - rata silhouette mengindikasikan pengelompokan cluster baik. Nilai dari Silhouette ada diantara -1 sampai dengan 1. Jika nilainya mendekati angka 1, maka titik data akan sangat mirip dengan titik data lainnya di cluster yang sama.

```
● ● ●

n_cluster_list = np.arange(start=2, stop=11, step=1)
sillhoute_score_list = []
for cluster in n_cluster_list:
    kmeans_model = KMeans(n_clusters = cluster)
    kmeans_model.fit_transform(X_scaled)
    sillhoute_index = silhouette_score(X, kmeans_model.labels_)
    print('Silhouette Score(n= {0}): {1}'.format(cluster,
    sillhoute_index))
    sillhoute_score_list.append(sillhoute_index)

sillhoute_score_df = pd.DataFrame({'Cluster' :
n_cluster_list, 'Sillhoute Score' :
sillhoute_score_list}).sort_values('Sillhoute Score', ascending =
False)
sillhoute_score_df
```

Berdasarkan gambar di samping, dapat disimpulkan bahwa untuk n\_clusters = 2 menghasilkan nilai silhouette yang tinggi.

Cluster	Sillhoute Score
2	0.686735
3	0.506153
4	0.348100
5	0.308861
9	0.279911
7	0.274930
6	0.269884
11	0.266398
10	0.261006
8	0.237991

## 5. Menetapkan label cluster

Menetapkan label cluster untuk setiap titik data sebagai berikut:



```
# assign the cluster result
df['Cluster'] = kmean.labels_
df
```

Kini kita memperoleh tabel beserta cluster-nya

sepal_length	sepal_width	petal_length	petal_width	species	Cluster
5.1	3.5	1.4	0.2	setosa	0
4.9	3.0	1.4	0.2	setosa	0
4.7	3.2	1.3	0.2	setosa	0
4.6	3.1	1.5	0.2	setosa	0
5.0	3.6	1.4	0.2	setosa	0

Link source code yang lebih lengkap dapat diakses [disini](#) ya!

Walaupun mudah dari segi implementasi, K-Means memiliki kelemahan dimana kita harus bereksperimen pada penentuan jumlah cluster yang bisa menghabiskan banyak waktu pada penentuan cluster. Terdapat algoritma alternatif untuk mengatasi kelemahan tersebut seperti DBSCAN dan Hierarchical Clustering.

## DBSCAN

Merupakan singkatan dari Density-Based Spatial Clustering of Application yang unggul dalam mengelompokkan data yang memiliki banyak outlier. Mengapa unggul? hal ini dikarenakan algoritma ini mengelompokkan data berdasarkan kepadatan titik data sehingga data yang terlampaui jauh atau diindikasikan sebagai outlier pasti akan diperhatikan oleh algoritma ini. Kita harus bereksperimen pada radius cluster dan berapa banyak minimum data pada suatu cluster apabila menerapkan algoritma ini.

## Hierarchical Clustering

Mengelompokkan data menggunakan pendekatan membagi data secara sekuensial atau berurutan sehingga membentuk sebuah hierarki struktur data. Hierarki akan berbentuk seperti pohon dimana struktur ini biasa dikenal sebagai dendrogram. Algoritma ini menggunakan dua pendekatan yang bisa kita pilih untuk mengelompokkan data, yaitu:

- **Divisive**

Menggunakan strategi top-down dimana mengelompokkan data dari suatu cluster besar terlebih dahulu yang kemudian membaginya menjadi beberapa cluster kecil

- **Agglomerative**

Kebalikan dari divisive, pendekatan ini menggunakan strategi bottom-up dimana langsung mengelompokkan data menjadi beberapa cluster kecil yang kemudian dikelompokkan menjadi sebuah cluster besar

# Introduction to Supervised Learning

Bayangkan Supervised Learning adalah proses belajar di sekolah seperti gambar berikut

- Ilmu atau angka yang ditulis di papan tulis adalah input yang akan diberikan ke siswa
- Siswa adalah model yang akan memproses ilmu yang diterima
- Siswa menjadi paham terhadap ilmu adalah desired output



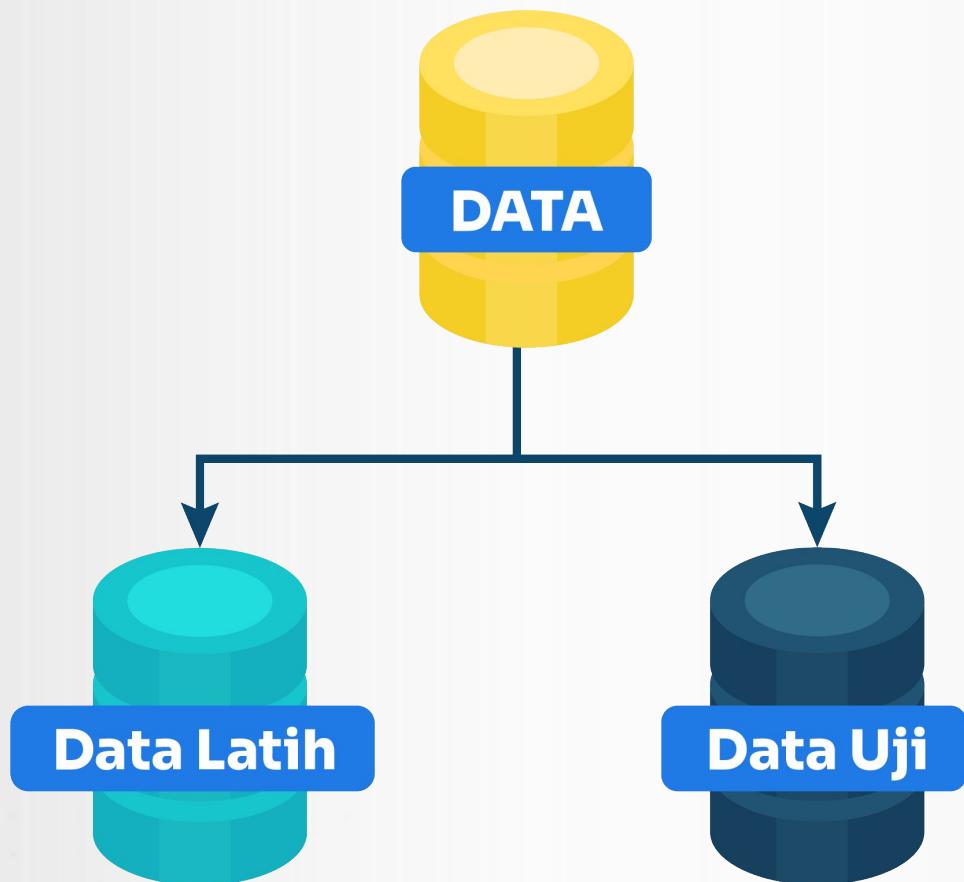
Kita harus memahami dulu kalau data harus dibagi menjadi dua bagian ketika membicarakan terkait supervised learning, yaitu:

- **Training dataset**

Himpunan data yang **digunakan untuk melatih atau membangun model** machine learning

- **Testing dataset**

Himpunan data yang **digunakan untuk menguji model** setelah proses training selesai. Testing dataset bersifat unseen, artinya model maupun manusia tidak boleh melihat sampel data ini, karena dapat mengakibatkan bias atau data tidak benar-benar teruji pada kumpulan data yang berbeda



Suatu data memiliki beberapa komponen seperti fitur, atribut, dan kelas. Fitur adalah variabel independen yang mempengaruhi output. Setiap fitur memiliki atribut nilai dengan tipe data dan range tertentu. Sedangkan kelas atau label adalah variabel dependen atau output.

Perhatikan contoh data berikut. Data ini berisi informasi penumpang Titanic yang mengalami kecelakaan pada 1912 silam. Informasi tentang seorang penumpang selamat atau tidak adalah kolom ‘survived’ dimana 0 mengindikasikan penumpang tersebut tidak selamat sementara 1 adalah selamat. Kolom ini lah yang menjadi kelas atau label pada data.

Fitur pada data ini adalah selain ‘survived’, yaitu ‘pclass’, ‘sex’, ‘age’, ‘sibsp’, ‘parch’, dan ‘fare’. Setiap fitur tersebut memiliki nilai yang beragam mulai dari range maupun tipe data.

<b>survived</b>	<b>pclass</b>	<b>sex</b>	<b>age</b>	<b>sibsp</b>	<b>parch</b>	<b>fare</b>
0	3	male	22.0	1	0	7.2500
1	1	female	38.0	1	0	71.2833
1	3	female	26.0	0	0	7.9250
1	1	female	35.0	1	0	53.1000
0	3	male	35.0	0	0	8.0500

Atribut

Fitur

Salah satu jenis supervised learning adalah regresi yang dapat digunakan untuk melihat hubungan sebab-akibat antar variabel.

Beberapa kegunaan dari regresi adalah untuk mengetahui variabel-variabel yang memiliki pengaruh terhadap variabel dependen.

Selain itu, regresi juga bisa digunakan untuk memprediksi output berupa bilangan kontinu. Terdapat beberapa jenis regresi seperti regresi linear, regresi linear berganda, regresi polinomial, dll.



Pembuatan model regresi dapat dilakukan menggunakan bahasa pemrograman Python. Secara garis besar, berikut adalah tahapan pembuatan model regresi menggunakan Python menggunakan dataset nilai dan IQ:

## 1. Memasukkan Package dan Dataset

Package adalah sekumpulan modul atau fungsi atau kode bahasa pemrograman python yang dapat memudahkan kita menulis perintah. Berikut cara kita memasukkan package dan dataset dalam bahasa pemrograman python.

Package yang biasa digunakan untuk memasukkan data adalah pandas. Seperti yang sudah kita singgung di pembahasan modul Data Preprocessing, package ini dapat dipakai untuk memasukkan berbagai file yang kita butuhkan baik dalam bentuk csv, excel, maupun json.

```
● ○ ●

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# Importing Data
general_data = pd.read_csv('content/drive/MyDrive/Startup
Campus/tes.csv')
general_data.head()
```

Gambaran dataset kita adalah seperti ini :

	A		B	
1	IQ		Nilai	
2	100		92	
3	110		86	
4	90		78	
5	105		82	
6	100		81	
7	100		81	

	A		B	
8	IQ		Nilai	
9	102		81	
10	100		81	
11	101		81	
12	121		88	
13	97		52	
14	98		55	

Kita menggunakan fungsi `.plot()` untuk membuat suatu tabel ditampilkan menjadi sebuah grafik yang di dalamnya terdapat parameter atau kode *syntax* yang bisa dimodifikasi sendiri sesuai kebutuhan, berikut rinciannya:

- ‘x=...’ mengatur data apa yang ditampilkan sebagai sumbu horizontal
- ‘y=...’ mengatur data apa yang ditampilkan sebagai sumbu vertikal
- ‘style=...’ mengatur bagaimana tampilan data yang ingin ditampilkan. Apabila kita menulis ‘o’ maka akan ditampilkan sebagai lingkaran (seperti di gambar sebelumnya. Kalau kita mengubahnya dengan ‘\*’, maka akan ditampilkan dalam bentuk bintang)

Selain itu, kita dapat memodifikasi tampilan grafik supaya lebih intuitif dengan mengatur judul grafik, judul sumbu x, dan judul sumbu y dengan rincian:

- Kita dapat mengatur judul grafik dengan memasukkan teks ke dalam fungsi `plt.title()`
- Kita dapat mengatur judul pada sumbu x grafik dengan memasukkan teks ke dalam fungsi `plt.xlabel()`
- Kita dapat mengatur judul pada sumbu y grafik dengan memasukkan teks ke dalam fungsi `plt.ylabel()`



```
general_data.plot(x= 'IQ', y = 'Nilai', style = 'o')
plt.title('IQ vs Percentage')
plt.xlabel('IQ')
plt.ylabel('Percentage Nilai')
plt.show()
```

## 2. Membagi data menjadi dua bagian

Selanjutnya kita membagi data menjadi 2 bagian dengan proporsi 80% untuk data training dan 20% sisanya untuk testing

- Pertama, kita mendefinisikan variabel “x” sebagai variabel independen dan “y” adalah variabel dependen
- Selanjutnya, kita mengimplementasikan pembagian data menggunakan syntax `train_test_split()` dengan pengaturan seperti berikut:
  - Memasukkan daftar array yang akan kita bagi. Dalam hal ini kita akan membagi array yang sudah didefinisikan sebagai “x” dan “y”
  - Kita akan menentukan proporsi data testing dengan mengatur nilai parameter “`test_size`” . Apabila kita menginginkan proporsi sebesar 20% dari total data, maka kita tulis `0.2`. Namun, kalau kita menginginkan proporsi sebesar 30% dari total data, maka kita tulis `0.3`
  - Isilah nilai “`random_state`” dengan suatu nilai agar hasil running program tidak berubah tiap kita menjalankannya
- Berhubung kita memasukkan dua buah array(x dan y), maka tiap array akan menghasilkan dua array baru. Tiap array baru tersebut akan merepresentasikan array untuk fase training dan testing model. Berikut lebih detailnya:
  - Array x akan terbagi menjadi `x_train` untuk training dan `x_test` untuk testing
  - Array y akan terbagi menjadi `y_train` untuk training dan `y_test` untuk



```
x = general_data.iloc[:: -1].values
y = general_data.iloc[:1].values

from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y,
test_size = 0.2, random_state = 0)
```

Dokumentasi syntax `train_test_split` lebih lengkap bisa dicek [disini](#) ya!

### 3. Analisis Model Regresi

Ketikkan script seperti dibawah. Pertama, kita membuat variabel baru bernama “regressor” yang kemudian kita menjalankan syntax `.fit()`. Fungsi tersebut berfungsi membangun model pada data training.

`.fit()` membutuhkan parameter dua buah array, yaitu array yang berisi kumpulan nilai variabel independen (`x_train`) dan array yang berisi kumpulan nilai variabel dependen (“`y_train`”).

Hasil running program tersebut adalah sebuah variabel bernama “`regressor`” yang akan menyimpan model matematis berdasarkan proses membangun model.

```
● ● ●

from sklearn.linear_model import LinearRegression

regressor = LinearRegression()
regressor.fit(x_train, y_train)

print(regressor.intercept_)
print(regressor.coef)

# Output: 23.18155410312282
# Output: [0.55610022]
```

Selanjutnya, kita bisa menampilkan nilai intercept dan koefisien model. Perhatikan script di bawah ini. Kita menampilkan nilai intercept dengan menuliskan ‘print(regressor.intercept\_)’ dan ‘print(regressor.coef)’ untuk menampilkan nilai koefisien model. 2 nilai ini digunakan untuk menentukan model matematika regresi yang baru.

```
● ● ●  
print(regressor.intercept_)  
print(regressor.coef)  
  
# Output: 23.18155410312282  
# Output: [0.55610022]
```

Berdasarkan hasil keluaran program diatas, persamaan model regresi kita akan seperti ini. Tujuan umum analisis regresi linear adalah mengetahui pengaruh variabel bebas terhadap variabel terikat. Seperti yang kita tahu, persamaan umum regresi linear adalah  $Y = \beta_0 + \beta_1 X$  dimana Y adalah variabel terikat, X adalah variabel bebas, serta  $\beta_0$  dan  $\beta_1$  adalah koefisien regresi. Setiap peningkatan satu variabel ‘Nilai’ akan membuat nilai ‘IQ’ meningkat sebesar 0,556.

**IQ = 23,018 + 0,556 \* Nilai**

Setelah membangun model berdasarkan data training, model tersebut dapat digunakan untuk memprediksi nilai IQ. Kita akan menggunakan syntax `.predict()` yang berisi array variabel independen (`x_test`).

Variabel `y_pred` adalah array yang berisi hasil prediksi model.

```
● ● ●  
y_pred = regressor.predict(x_test)
```

Silahkan tulis script berikut untuk menyandingkan nilai data aktual dan nilai hasil prediksi model.

```
● ● ●  
df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})  
df
```

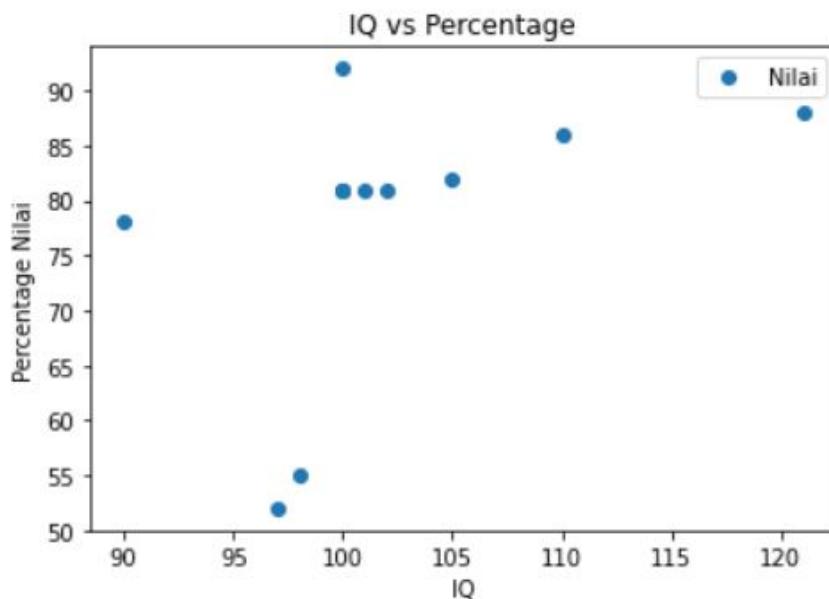
Output program di atas adalah sebagai berikut. Kolom ‘Actual’ merupakan data aktual yang kita miliki, sementara kolom ‘Predicted’ adalah data hasil prediksi model regresi.

	Actual	Predicted	
0	81	79.740378	
1	55	77.515977	
2	81	78.628177	

Apabila kita mengulas kembali modul 2, kita dapat menganalisa dan menyelidiki sebuah dataset melalui proses Exploratory Data Analysis (EDA) dimana terdapat beberapa proses yang bisa dilakukan seperti distinguish attributes, univariate analysis, bi-multivariate analysis, dan data outlier. Kali ini, kita coba mengulas salah satunya, yaitu analisis bi-/multivariate untuk melihat hubungan antar dua variabel seperti melalui diagram scatter di bawah ini. Hal ini tentu bertujuan meningkatkan pemahaman kita terkait model regresi yang akan dibangun.

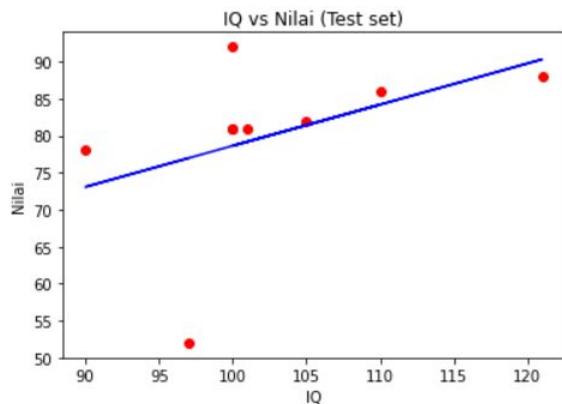
Silahkan coba script berikut untuk menampilkan diagram *scatter* yang berfungsi menampilkan hubungan antara IQ dan nilai dalam persebaran datanya. Diagram ini dapat menampilkan bagaimana persebaran data antar dua variabel tersebut sehingga kita memperoleh suatu informasi. Berdasarkan grafik ini, kita dapat mengetahui gambaran secara umum bahwa semakin tinggi IQ akan diikuti nilai yang lebih tinggi. Artinya, semakin tinggi seseorang biasanya ada kecenderungan nilai yang diperoleh lebih tinggi.

```
[20] general_data.plot(x='IQ',y='Nilai',style='o')
    plt.title('IQ vs Percentage')
    plt.xlabel('IQ')
    plt.ylabel('Percentage Nilai')
    plt.show()
```



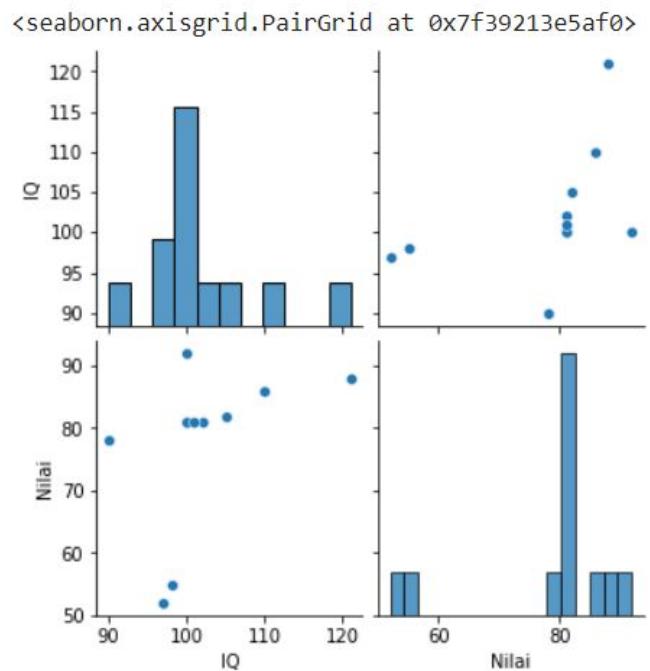
Silahkan tulis script berikut untuk me-visualisasikan bagaimana hubungan variabel Nilai dan IQ serta garis regresinya.

```
[28] plt.scatter(x_train, y_train, color = 'red')
    plt.plot(x_train, regressor.predict(x_train),color = 'blue')
    plt.title('IQ vs Nilai (Test set)')
    plt.xlabel('IQ')
    plt.ylabel('Nilai')
    plt.show()
```



Selanjutnya, kita bisa menampilkan plot dari IQ dan Nilai menggunakan script seperti dibawah ini. Kita menggunakan fungsi **.pairplot()** yang disediakan oleh package Seaborn. Kita bisa melihat bagaimana tabel distribusi dan persebaran data masing-masing variabel IQ dan Nilai.

```
[27] sns.pairplot(general_data)
```



Syntax implementasi pembuatan model regresi bisa dilihat [disini](#) ya!

Apabila membahas regresi, maka kita tidak akan bisa lepas dari aktivitas peramalan (forecasting). Forecasting adalah bagian luas dari berbagai industri dimana sudah sangat sering digunakan dalam membantu analisis data.

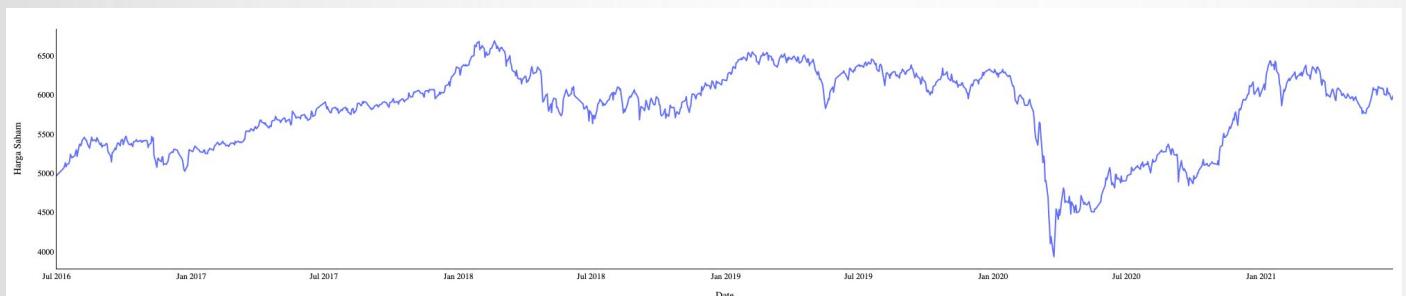
Kali ini kita akan membahas beberapa model dasar dan sederhana yang bisa dipakai untuk forecasting. Terlepas dari kesederhanaannya, model ini dapat menawarkan hasil yang baik dan memberikan informasi yang berharga. Sesuai namanya, yaitu aktivitas peramalan, aktivitas ini berfungsi memberikan informasi terkait nilai di masa mendatang. Oleh karena itu, data yang digunakan dalam analisis peramalan biasanya bersifat time series karena urut berdasarkan waktu. Apa itu time series? Apa saja contoh penerapannya? Yuk kita bahas satu per satu!



# Time Series

Waktu merupakan salah satu faktor penting yang tidak dapat diabaikan. Apa yang membedakan data deret waktu dari data lain adalah bahwa data yang dimiliki berubah seiring berjalannya waktu. Dengan kata lain, waktu adalah variabel penting karena menunjukkan bagaimana data menyesuaikan sepanjang titik data serta hasil akhir. Tentu hal ini dapat menjadi tambahan sumber informasi serta dapat menyatakan bagaimana urutan antar data.

Time series adalah **data yang memiliki urutan** berbagai titik data **yang terjadi secara berurutan dalam jangka waktu**. Analisis data ini biasanya memerlukan data yang cukup besar serta konsisten berdasarkan waktu. Salah satu contoh analisis data time series adalah analisis pergerakan harga saham. Coba perhatikan grafik pergerakan harga saham di bawah ini.



Data harga saham termasuk data time series karena selalu urut berdasarkan waktu. Misalkan harga di tahun 2020 mengalami penurunan yang signifikan sekitar 3000, namun di tahun 2022 kini harganya menjadi 4500. **Cara analisis data time series tidak bisa disamakan dengan data non time series.** Mengapa?

Apabila kita mengubah susunan datanya, maka bisa mengubah makna data. Oleh karena itu, perlu sebuah analisis khusus terhadap data tipe ini. Salah satu contoh lain analisis data time series adalah data cuaca, penggunaan listrik, atau peramalan demam berdarah.

Terdapat beberapa komponen yang harus diperhatikan dalam menganalisis data time series, yaitu:

- **Tren**

Memiliki karakteristik **tidak ada interval tetap** dan perbedaan apa pun dalam kumpulan data (tren positif atau negatif)

- **Musiman**

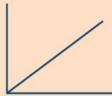
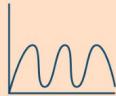
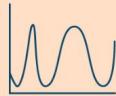
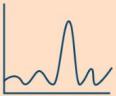
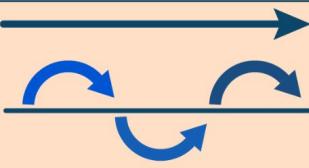
Memiliki **karakteristik interval reguler** atau tetap bergeser dalam kumpulan data dalam garis waktu yang berkelanjutan

- **Siklus**

Memiliki **karakteristik tidak ada interval reguler** serta ketidakpastian dalam polanya

- **Irregularity**

Memiliki karakteristik **pola data selalu tak terduga** dalam suatu rentang waktu

	<b>Trend</b>	<b>Seasonality</b>	<b>Cyclical</b>	<b>Irregularity</b>
Time	Fixed Time Interval	Fixed Time Interval	Not Fixed Time Interval	Not Fixed Time Interval
Duration	Long and Short Term	Short Term	Long and Short Term	Regular/Irregular
Visualization				
Nature - I	Gradual	Swings between Up or Down	Repeating Up and Down	Errored or High Fluctuation
Nature - II	Upward/Down Trend	Pattern repeatable	No fixed period	Short and Not repeatable
Prediction Capability	Predictable	Predictable	Challenging	Challenging
Market Model				Highly random/Unforeseen Events - along with white noise.

Setelah memahami bagaimana karakteristik forecasting dan data time series, sekarang kita coba membahas model apa saja yang bisa digunakan untuk forecasting yuk!

## Moving Average Forecast

Moving Average merupakan salah satu model sederhana untuk diimplementasikan dimana mengasumsikan **semua nilai masa depan sama dengan rata-rata dari semua pengamatan sebelumnya**. Perhatikan persamaan berikut

$$y_{T+h} = \bar{y} = \frac{\sum_{t=1}^T y_t}{T}$$

dimana:

- 'h' adalah waktu yang akan datang yang akan dilakukan forecasting
- 'T' adalah panjang deret waktu
- 'y\_t' adalah nilai yang diamati pada waktu t
- 'y\_bar' adalah rata-rata dari nilai yang diamati

Oleh karena itu, kita harus memiliki data di masa lalu apabila ingin menerapkan model ini supaya bisa memperoleh nilai di masa mendatang.

Perhatikan tabel di bawah, kita mempunyai data penjualan selama tiga bulan terakhir. Apabila kita menerapkan forecasting menggunakan model average forecasting, maka prediksi di bulan keempat sama dengan nilai rata-rata selama tiga bulan terakhir.

Bulan	Penjualan (dalam juta rupiah)
Januari	23
Februari	28
Maret	22
April	?

Berhubung data yang dimiliki adalah bulan Januari hingga Maret, maka perhitungannya adalah sebagai berikut:

$$\frac{\text{Nilai Rata-Rata Januari} + \text{Nilai Rata-Rata Februari} + \text{Nilai Rata-Rata Maret}}{3}$$

Berdasarkan perhitungan di atas, penjualan di April diprediksi sekitar 24,3 juta rupiah dengan rincian sebagai berikut:

$$(23+28+22):3 = 24,3$$

Bulan	Penjualan (dalam juta rupiah)
Januari	23
Februari	28
Maret	22
April	24,3

## Naive Forecast

Model Naive Forecast mengasumsikan **nilai masa depan sama dengan nilai pengamatan saat ini**. Model ini dianggap sebagai tolok ukur untuk perkiraan apa pun dan sering digunakan untuk memodelkan pasar saham dan data keuangan karena sifatnya yang tidak menentu.

$$y_{T+h} = y_T$$

Perhatikan tabel di bawah, kita mempunyai data penjualan selama tiga bulan terakhir. Apabila kita menerapkan peramalan menggunakan naive forecasting, maka prediksi di bulan keempat sama dengan nilai di bulan ketiga.

Bulan	Penjualan (dalam juta rupiah)
Januari	23
Februari	28
Maret	22
April	?

Jumlah penjualan di bulan April diprediksi sama dengan bulan Maret, yaitu sekitar 22 juta rupiah

Bulan	Penjualan (dalam juta rupiah)
Januari	23
Februari	28
Maret	22
April	22

## Seasonal Naive Forecast

Metode ini merupakan pengembangan naive forecasting, namun yang membedakan adalah **nilai prediksi sama dengan nilai pengamatan terbaru di musim yang sama**. Oleh karena itu, ini dikenal sebagai Seasonal Naive Forecasting. Misalnya, ramalan untuk kuartal satu berikutnya sama dengan nilai kuartal satu tahun sebelumnya. Model ini berguna ketika kita memiliki variasi musiman yang jelas dan besar dalam deret waktu kita. Secara matematis model ditulis sebagai berikut:

$$y_{T+h} = y_{T+h-m}$$

Di mana m adalah musiman dari data. Misalkan:

- Data bulanan dengan musiman tahunan akan memiliki m=12
- Data triwulan akan memiliki m=4
- Data mingguan akan memiliki m=52

Misalkan kita mempunyai data seperti tabel berikut dimana kita mempunyai data penjualan selama satu tahun terakhir. Apabila kita menerapkan peramalan menggunakan seasonal naive forecasting dengan periode enam bulan ke belakang, maka prediksi di bulan Desember sama dengan nilai penjualan di bulan Mei.

Bulan	Penjualan (dalam jutaan rupiah)
Januari	23
Februari	25
Maret	26
April	28
Mei	30
Juni	31
Juli	23
Agustus	25
Oktober	26
November	28
Desember	?



Jumlah penjualan di bulan Desember diprediksi sama dengan penjualan di enam bulan sebelumnya

Bulan	Penjualan (dalam jutaan rupiah)
Januari	23
Februari	25
Maret	26
April	28
Mei	30
Juni	31
Juli	23
Agustus	25
Oktober	26
November	28
Desember	30

Selain itu, kita juga dapat menggunakan model pengembangan Moving Average, yaitu Autoregressive-Moving Average (ARMA) untuk menganalisis data time series. Model ini cukup sering digunakan khususnya pada studi kasus peramalan (forecasting) dimana datanya berkorelasi dalam suatu deret waktu.

ARMA merupakan kombinasi model Moving Average seperti yang sudah kita bahas sebelumnya dengan model Auto-Regressive (AR). Model ini merupakan proses memprediksi nilai di masa mendatang berdasarkan kumulasi data di masa lalu sehingga cocok digunakan dalam memprediksi data time series. Model ini mengasumsikan nilai di masa mendatang sama dengan nilai di masa kini. Selain itu, model ini sangat cocok digunakan pada data yang stasioner. Namun yang harus diingat, data bisa dikatakan stasioner apabila:

- Nilai rata-rata harus benar-benar konstan dalam data selama analisis
- Variansi harus konstan sehubungan dengan kerangka waktu
- Covariance mengukur hubungan antara dua variabel

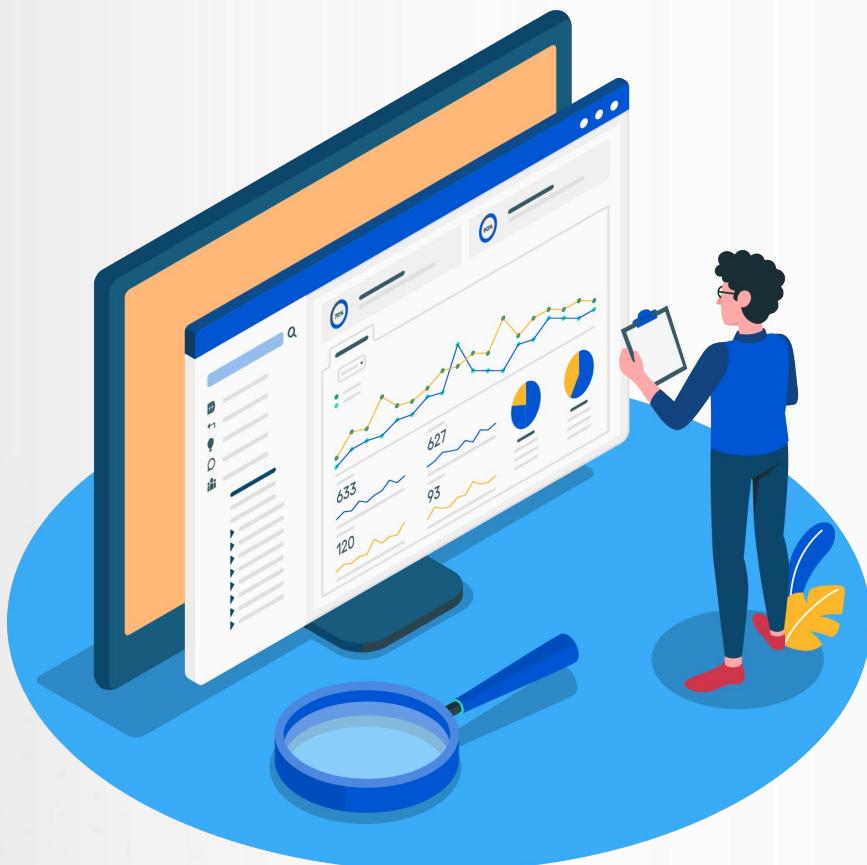


Image [ref](#)

Ada beberapa notasi yang harus kita ketahui dalam mengimplementasikan model ARMA untuk peramalan, yaitu:

- **p**

Parameter yang terkait dengan aspek AR dari model yang menggabungkan nilai-nilai sebelumnya yaitu kelambatan variabel dependen. Misalnya jika p adalah 5, prediktor untuk  $x(t)$  akan menjadi  $x(t-1) \dots x(t-5)$

- **q**

Ukuran jendela bagian rata-rata bergerak dari model. Misalnya jika q adalah 5, prediktor untuk  $x(t)$  akan menjadi  $e(t-1) \dots e(t-5)$  di mana  $e(i)$  adalah nilai error antara hasil prediksi moving average pada saat ke-i dengan nilai aktual

Apabila data kita memang tidak stasioner, kita dapat menggunakan pengembangan dari ARMA, yaitu Autoregressive Integrated Moving Average (ARIMA). Model ini bisa dikatakan lebih baik daripada ARMA karena mampu mengatasi data non stasioner. Sebelum kita mengenal ARIMA, sebaiknya pahami terlebih dahulu istilah-istilah di bawah ini:

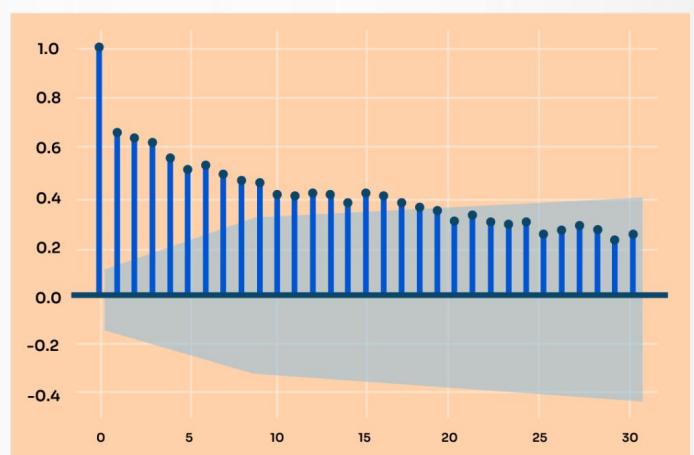
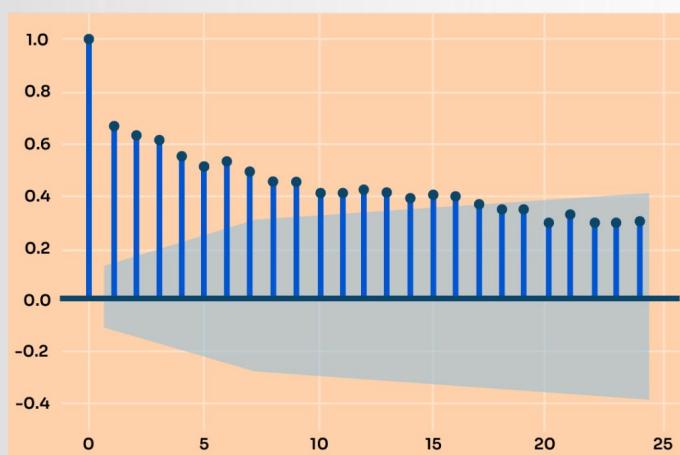
- **Autocorrelation Function (ACF)**

ACF digunakan untuk menunjukkan dan seberapa mirip suatu nilai dalam time series tertentu dengan nilai sebelumnya

- **Partial Autocorrelation (PACF)**

PACF mirip dengan ACF dan sedikit sukar untuk dipahami. PACF dapat menunjukkan korelasi urutan suatu data dengan dirinya sendiri dengan sejumlah unit waktu

## Auto-correlation and Partial Auto-Correlation



Sementara itu, model ARIMA memiliki 3 parameter yang digunakan untuk menganalisis data time series yang dinotasikan sebagai p, d, dan q.

- **p**

Parameter yang terkait dengan aspek AR dari model yang menggabungkan nilai-nilai sebelumnya yaitu kelambatan variabel dependen. Misalnya jika p adalah 5, prediktor untuk  $x(t)$  akan menjadi  $x(t-1) \dots x(t-5)$

- **d**

Parameter yang mempengaruhi berapa banyak differencing yang harus diaplikasikan dalam model

- **q**

Ukuran jendela bagian rata-rata bergerak dari model. Misalnya jika q adalah 5, prediktor untuk  $x(t)$  akan menjadi  $e(t-1) \dots e(t-5)$  di mana  $e(i)$  adalah nilai error antara hasil prediksi moving average pada saat ke-i dengan nilai aktual

Di luar konteks regresi, terdapat aktivitas lain yang harus kita pahami dalam pembelajaran supervised learning. Aktivitas yang dimaksud adalah klasifikasi. Apabila regresi biasanya digunakan untuk memprediksi nilai kontinyu, maka klasifikasi digunakan untuk memprediksi dalam beberapa bentuk kelompok atau kelas seperti mengklasifikasikan suatu pesan apakah termasuk spam atau tidak.

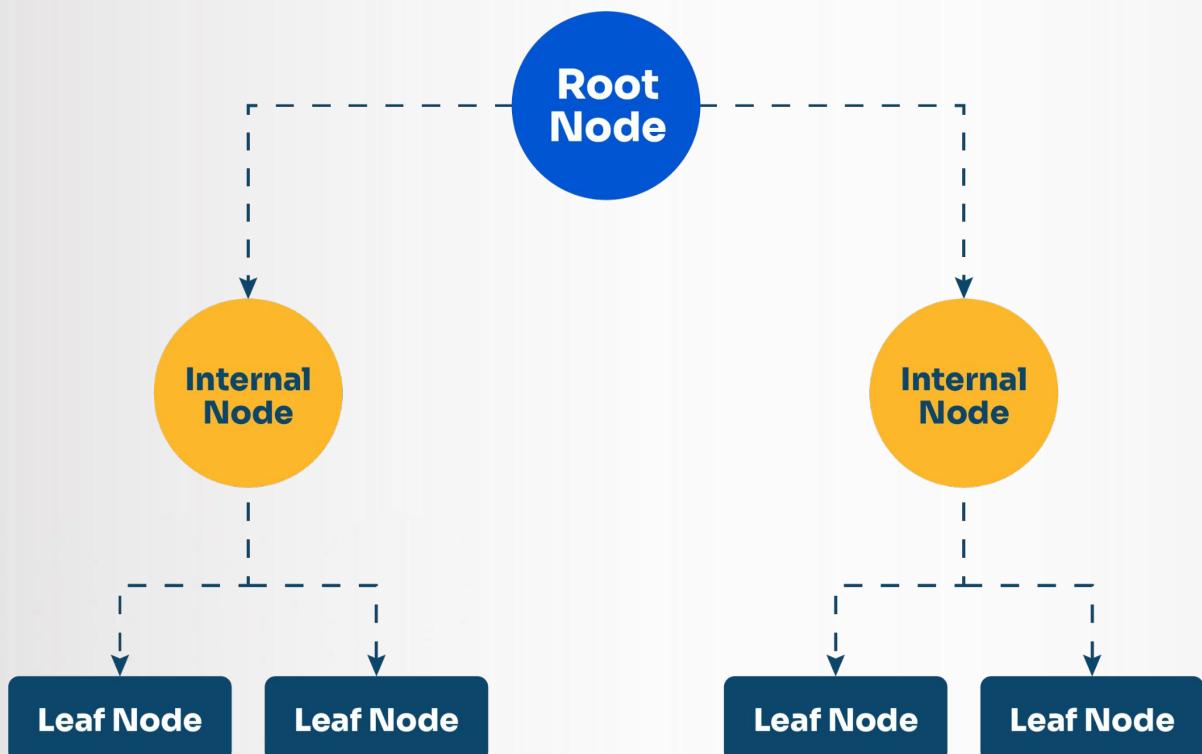
Ada beberapa metode atau algoritma yang bisa dipakai seperti decision tree, random forest, dan gradient boosting. Setiap algoritma punya karakteristik masing-masing, yuk kita ulas satu per satu.



## Decision Tree

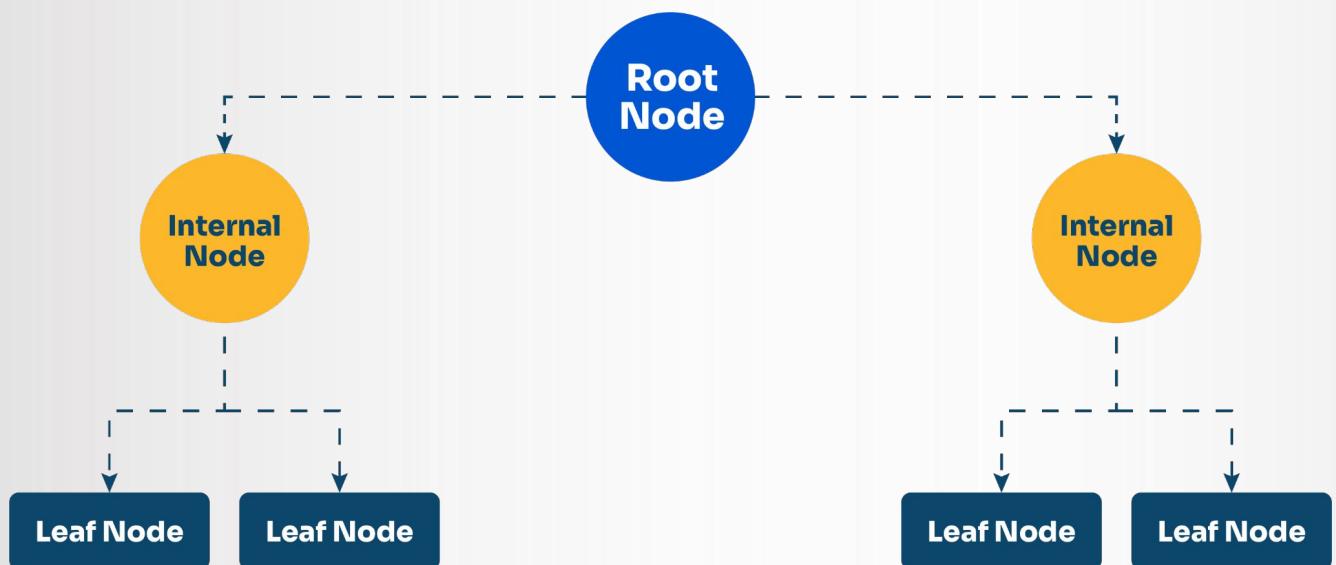
Decision Trees adalah algoritma pembelajaran yang dapat digunakan untuk tugas klasifikasi dan regresi. Algoritma ini memiliki hierarki, struktur pohon, yang terdiri dari simpul akar, cabang, simpul internal dan simpul daun dimana sama seperti sebuah pohon.

Algoritma ini dapat digunakan untuk merepresentasikan keputusan dan pengambilan keputusan secara visual dan eksplisit dalam analisis keputusan. Pembelajaran decision trees menggunakan strategi membagi dan untuk mengidentifikasi titik perpecahan yang optimal dalam sebuah pohon. Proses pemisahan ini kemudian diulangi secara top-down, rekursif hingga semua, atau sebagian besar telah diklasifikasikan di bawah label kelas tertentu. Apakah semua poin data diklasifikasikan sebagai kumpulan homogen atau tidak sangat bergantung pada kompleksitas decision trees.

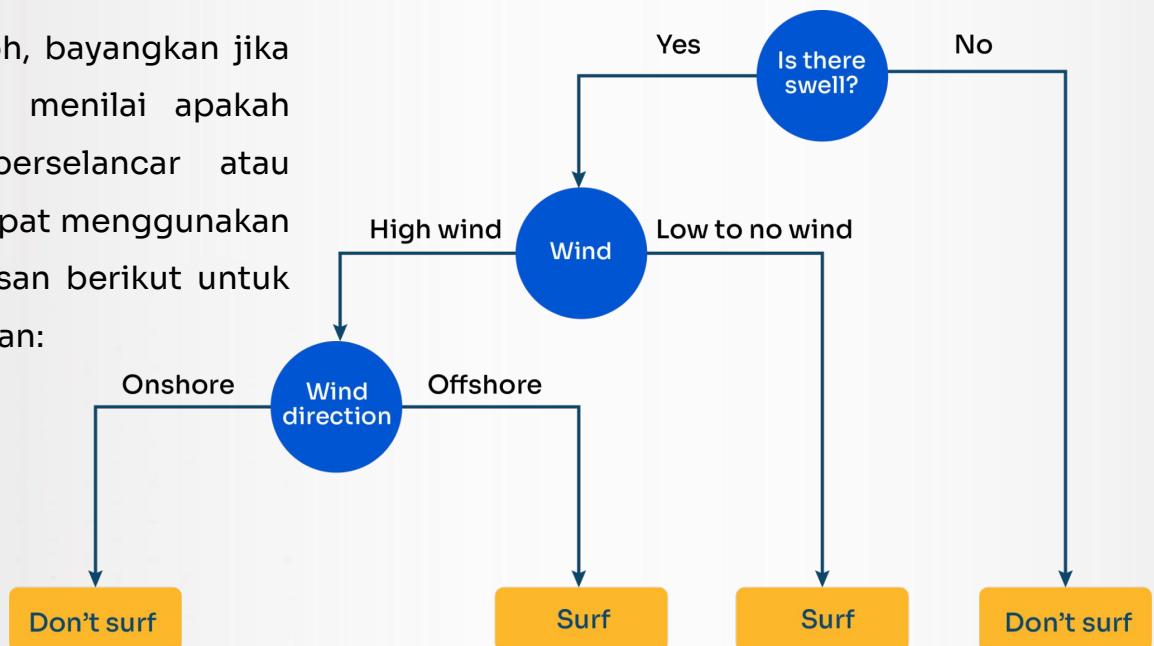


Kalau kita lihat gambar berikut, decision trees dimulai dengan simpul akar (root node) yang berwarna biru. Cabang keluar dari simpul akar kemudian masuk ke simpul internal (internal node) juga dikenal sebagai simpul keputusan.

Berdasarkan fitur yang tersedia, kedua tipe simpul melakukan evaluasi untuk membentuk himpunan bagian yang homogen, yang dilambangkan dengan simpul daun (leaf node), atau simpul terminal. Node daun (leaf node) mewakili semua kemungkinan hasil dalam kumpulan data.



Sebagai contoh, bayangkan jika kita mencoba menilai apakah kita harus berselancar atau tidak. Anda dapat menggunakan aturan keputusan berikut untuk membuat pilihan:



Selain itu, ada yang harus diperhatikan terkait kelebihan dan kekurangan decision tree, yaitu:

## Kelebihan

- **Mudah diinterpretasikan** dimana sifat hierarki *decision trees* juga memudahkan untuk melihat atribut mana yang paling penting
- **Sedikit atau tidak diperlukan persiapan data** karena dapat menangani berbagai tipe data — misalnya nilai diskrit atau kontinu, dan nilai kontinu dapat diubah menjadi nilai kategoris melalui penggunaan ambang batas
- Selain itu, hal tersebut juga dapat menangani data *missing value*, yang dapat menjadi masalah bagi pengklasifikasi lainnya seperti Naïve Bayes
- Lebih fleksibel baik untuk tugas klasifikasi maupun regresi, membuatnya lebih fleksibel daripada beberapa algoritma lainnya

## Kekurangan

- **Rentan mengalami overfitting**
- **Lebih mahal** berhubung decision trees mengambil pendekatan pencarian yang masif selama konstruksi sehingga berdampak pada sumber daya dan waktu komputasi
- Estimator varian tinggi

Variasi kecil dalam data dapat menghasilkan pohon keputusan yang sangat berbeda. Bagging, atau rata-rata estimasi, dapat menjadi metode untuk mengurangi varian dari decision trees. Namun, pendekatan ini terbatas karena dapat menyebabkan prediktor yang sangat berkorelasi

Salah satu upaya untuk meningkatkan akurasi atau performa model adalah melalui penggabungan model atau biasa disebut ensemble learning.

Secara umum, ensemble learning bekerja dengan menggabungkan hasil prediksi dari model dasar atau model yang lebih lemah melalui sebuah model. Model yang menggabungkan hasil prediksi tersebut biasa disebut meta model, sementara model yang digabungkan bisa disebut base model.

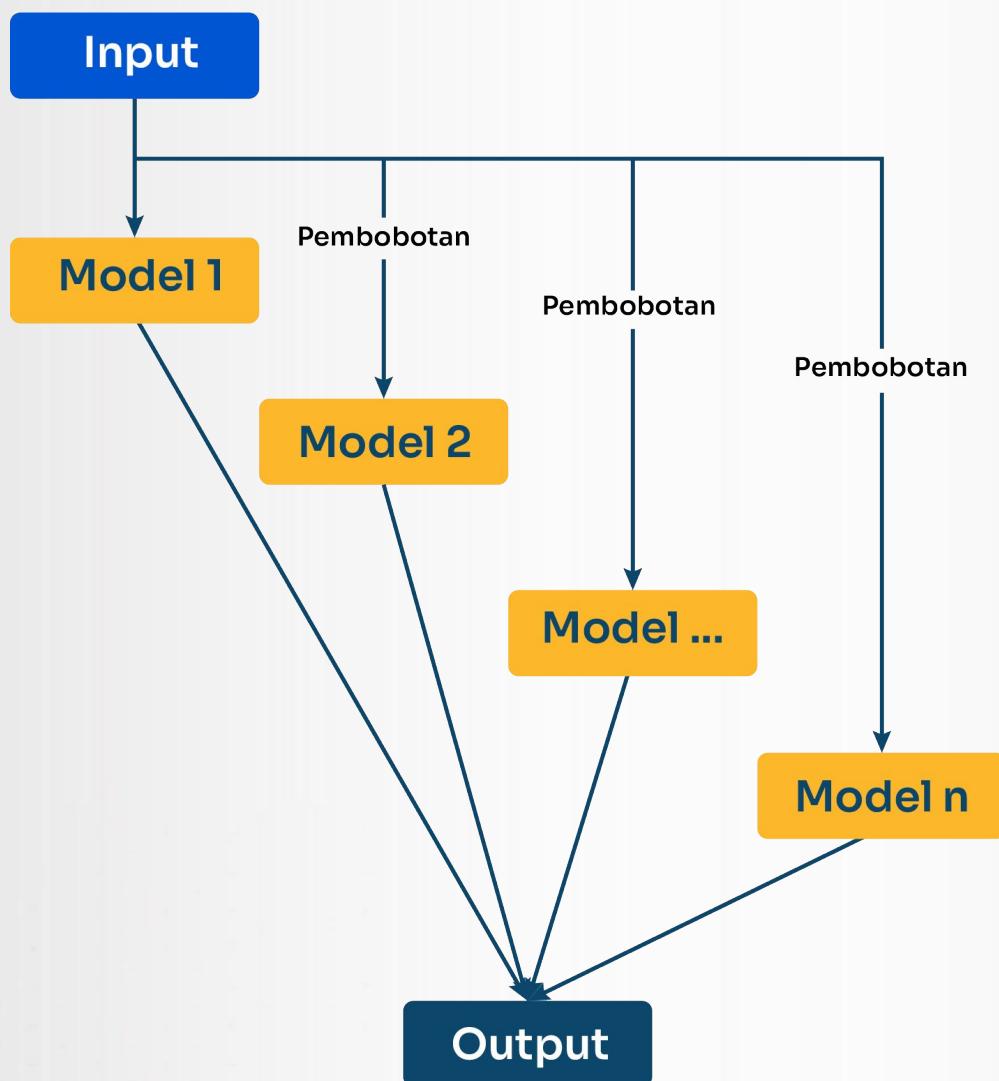
Penerapan ensemble learning dapat dilakukan dengan beberapa teknik, diantaranya secara berurutan(sequential), pararel, dan menumpuk (stacking).

Beberapa teknik ensemble learning, yaitu:

### Sequential Ensemble Learning

Teknik yang dilakukan dengan **mengasosiasikan output dari model yang lemah secara berurutan. Performa model ditingkatkan dengan cara memberikan bobot** yang lebih tinggi pada sampel yang salah diklasifikasikan.

Salah satu contoh model yang menerapkan model ini adalah Gradient Boosting dan XGBoost.



## Gradient Boosting

Konsep utama boosting adalah bagaimana **meningkatkan kinerja suatu model dasar atau lemah**. Dalam machine learning, kita dapat menggunakan gradient boosting untuk menyelesaikan masalah klasifikasi dan regresi. Metode ini menciptakan model secara bertahap. Hal tersebut menyimpulkan bahwa model tersebut mengaktifkan optimalisasi fungsi kerugian yang dapat dibedakan secara absolut. Saat menambahkan setiap yang lemah, model baru dibuat yang memberikan estimasi variabel respons yang lebih tepat.

Algoritma gradient boosting memerlukan komponen di bawah ini untuk berfungsi:

### 1. Loss Function

Kita **mengoptimalkan loss function untuk mengurangi kesalahan** dalam prediksi. Gradient boosting bekerja dengan tidak memberikan bobot yang lebih tinggi pada hasil prediksi yang dianggap rendah

### 1. Weak learner

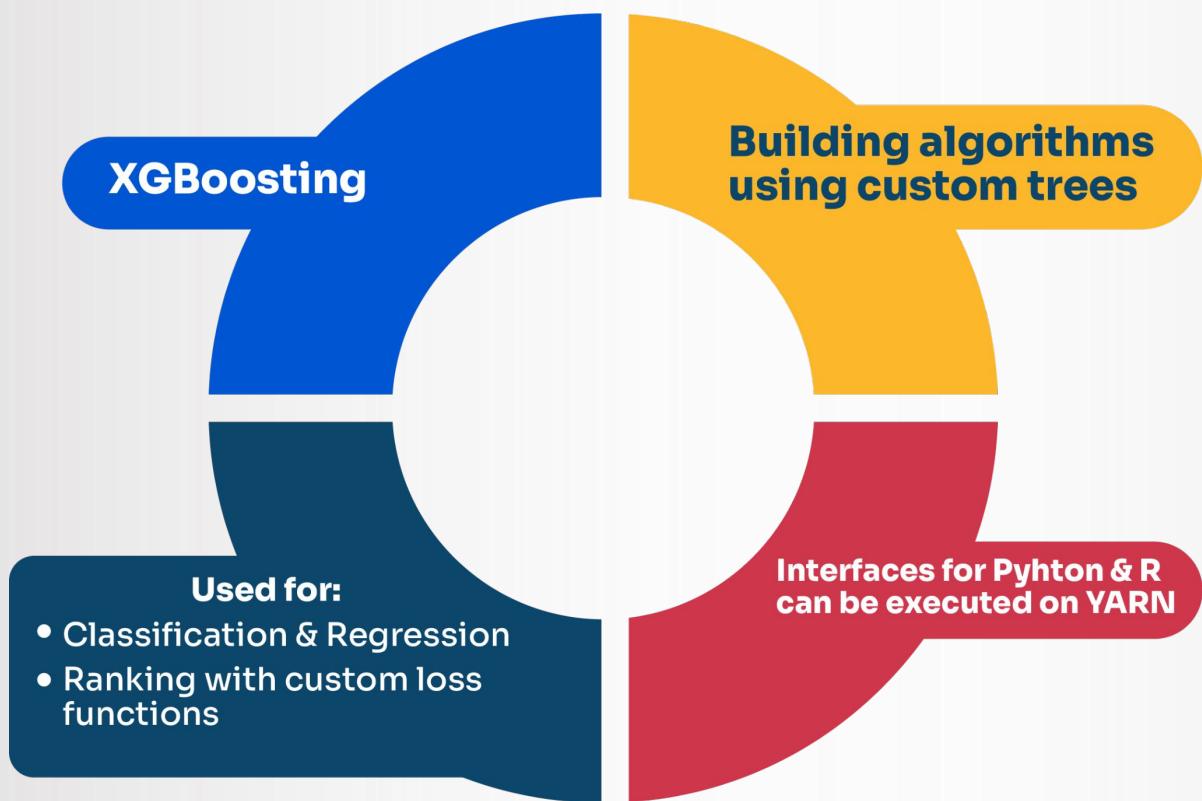
Weak learner atau **model pembelajar lemah yang digunakan** gradient boosting adalah **decision tree**. Gradient boosting memang membutuhkan komputasi yang besar karena biasanya bekerja dengan memodifikasi decision tree seperti menambah jumlah pohon, menambah kedalaman pohon, atau jumlah node dalam pohon

### 1. Model aditif

Umumnya gradient boosting bekerja dengan menambahkan pohon baru dimana **keluaran pohon yang baru digunakan untuk mengkoreksi hasil prediksi** yang ada

- **XGBoost**

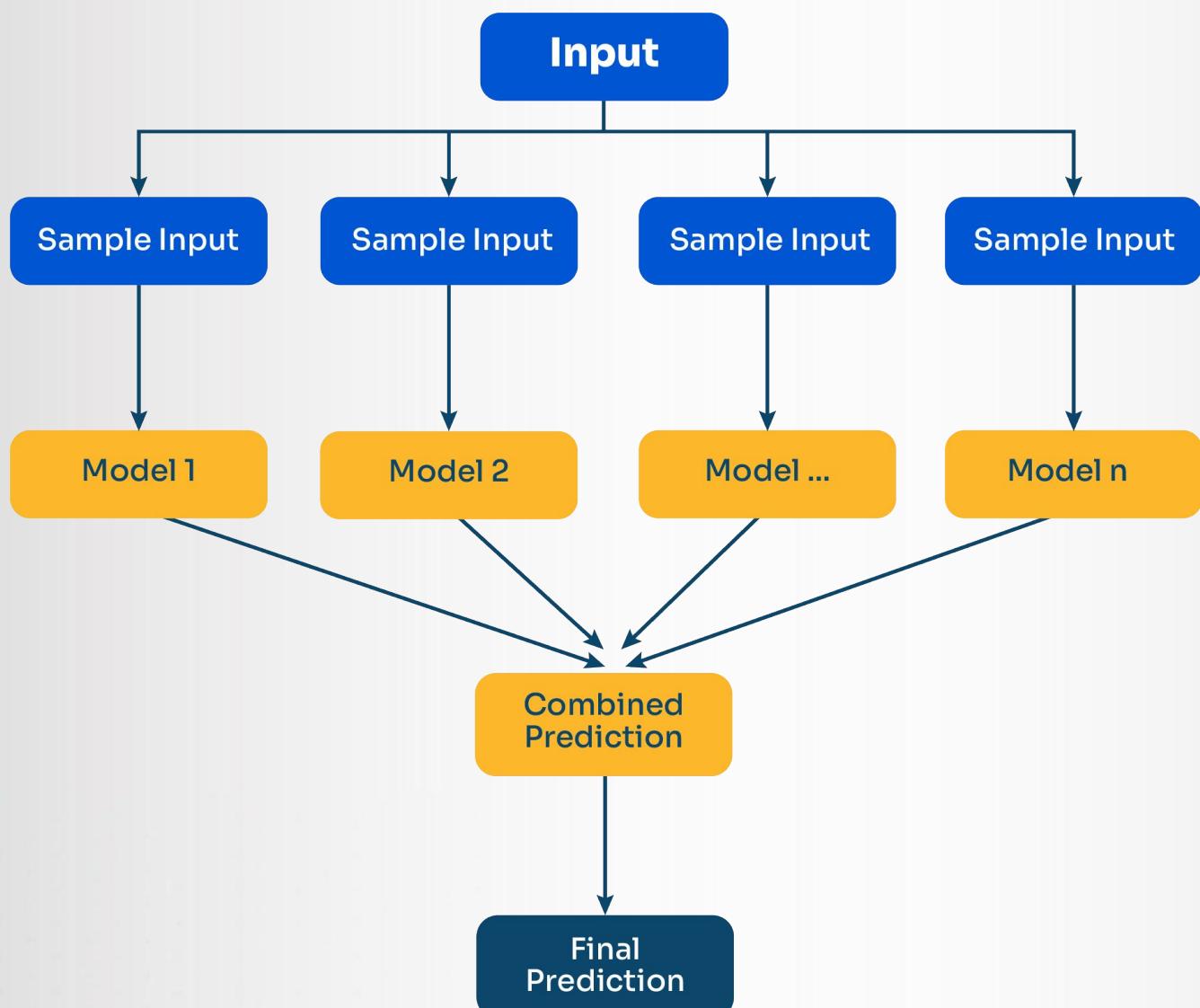
Extreme Gradient Boosting (XGBoost) merupakan pengembangan dari gradient boosting. Pengembangan dilakukan dengan tujuan adanya suatu model boosting yang mampu menghasilkan kinerja lebih baik disertai dengan waktu komputasi yang lebih cepat. Oleh karena itu, peneliti atau data scientist menggunakan ini daripada gradient boosting seperti yang kita bahas sebelumnya.



Beberapa teknik ensemble learning, yaitu:

### Parallel Ensemble Learning

Teknik yang dilakukan dengan **mengasosiasikan output dari model yang lemah secara paralel** dengan rata-rata atau pemilihan (voting). Hal tersebut bertujuan mengurangi kesalahan hasil prediksi. Salah satu contoh model yang menerapkan model ini adalah Random Forest seperti yang sudah kita bahas sebelumnya.



## Random Forest

**Algoritma ini membangun berdasarkan kelompok decision trees** pada sampel yang berbeda dan menggunakan metode voting untuk menemukan model terbaik.

Dalam analogi kehidupan nyata untuk memahami konsep ini lebih jauh. Contohnya seorang siswa bernama X ingin memilih mata pelajaran setelah 10+2, dan dia bingung tentang pilihan mata pelajaran berdasarkan keahliannya. Jadi dia memutuskan untuk berkonsultasi dengan berbagai orang seperti sepupunya, guru, orang tua, mahasiswa sarjana, dan pekerja. Dia mengajukan berbagai pertanyaan kepada mereka seperti mengapa dia harus memilih, peluang kerja, biaya kursus, dll. Akhirnya, setelah berkonsultasi dengan berbagai orang tentang kursus, dia memutuskan untuk mengambil kursus yang disarankan oleh kebanyakan orang.

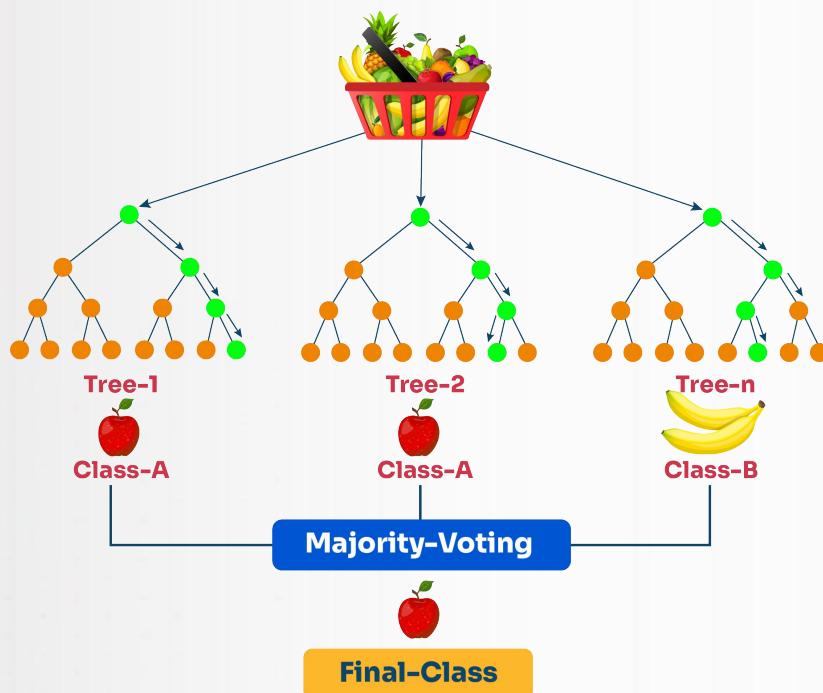


Salah satu fitur terpenting random forest adalah dapat menangani kumpulan data yang berisi variabel kontinu seperti dalam kasus regresi dan variabel kategori seperti dalam kasus klasifikasi. Hal tersebut menghasilkan hasil yang lebih baik untuk masalah klasifikasi. Namun, sebenarnya bagaimana algoritma ini berjalan?

- Langkah 1: Di random forest n sejumlah catatan acak diambil dari kumpulan data yang memiliki sejumlah k catatan.
- Langkah 2: Individual random forest dibangun untuk setiap sampel.
- Langkah 3: Setiap random forest akan menghasilkan output.
- Langkah 4: Hasil akhir dipertimbangkan berdasarkan Pemungutan Suara Mayoritas atau Rata-Rata untuk Klasifikasi dan regresi masing-masing.

Sebagai contoh: pertimbangkan keranjang buah sebagai data seperti yang ditunjukkan pada gambar di bawah ini. Sekarang sejumlah n sampel diambil dari keranjang buah dan *random forest* individual dibuat untuk setiap sampel.

Setiap *random forest* akan menghasilkan *output* seperti yang ditunjukkan pada gambar. Hasil akhir dipertimbangkan berdasarkan suara terbanyak. Pada gambar di bawah ini kita dapat melihat bahwa *random forest* mayoritas menghasilkan apel jika dibandingkan dengan pisang, sehingga hasil akhir diambil sebagai apel.



Selain itu, ada yang harus diperhatikan terkait kelebihan dan kekurangan Random Forest, yaitu:

## Kelebihan

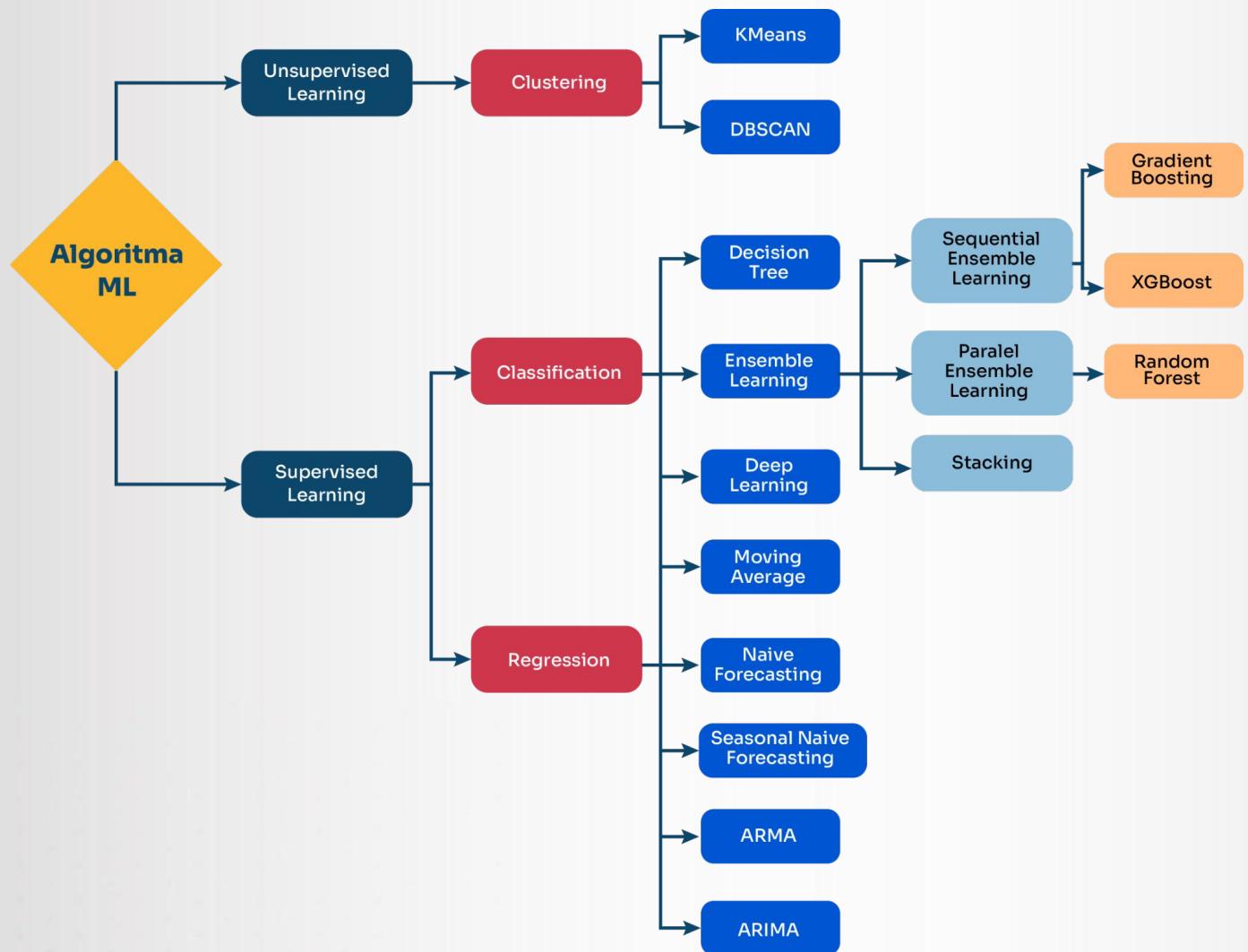
- Mampu meminimalisir overfitting karena output didasarkan pada voting mayoritas atau rata-rata
- Sangat stabil karena diambil rata-rata jawaban yang diberikan oleh banyak pohon.
- Mempertahankan keragaman karena semua atribut tidak dipertimbangkan saat membuat setiap pohon keputusan meskipun tidak benar dalam semua kasus.
- Kebal terhadap dimensi. Karena setiap pohon tidak mempertimbangkan semua atribut, ruang fitur berkurang.
- Tidak perlu memisahkan data menjadi pelatihan dan pengujian karena akan selalu ada 30% data yang tidak terlihat oleh pohon keputusan yang dibuat dari bootstrap

## Kekurangan

- Random forest sangat kompleks jika dibandingkan dengan decision trees dimana keputusan dapat dibuat dengan mengikuti jalur pohon tersebut
- Waktu pelatihan lebih banyak dibandingkan dengan model lain karena kerumitannya. Setiap kali harus membuat prediksi, setiap decision trees harus menghasilkan keluaran untuk data masukan yang diberikan

Sejauh ini kita telah membahas berbagai macam model atau algoritma machine learning mulai dari unsupervised learning hingga supervised learning seperti terlihat pada gambar di bawah. Secara umum, hampir semua model kecuali ARMA dan ARIMA dapat digunakan baik untuk tugas klasifikasi maupun regresi. Hal ini mengindikasikan fleksibilitas penggunaan model machine learning pada berbagai tugas.

Setiap model pasti memiliki kelebihan dan kekurangan masing-masing sehingga tidak ada model yang dikatakan sempurna. Oleh karena itu, kita harus cermat dalam memilih suatu model dalam memecahkan suatu permasalahan.



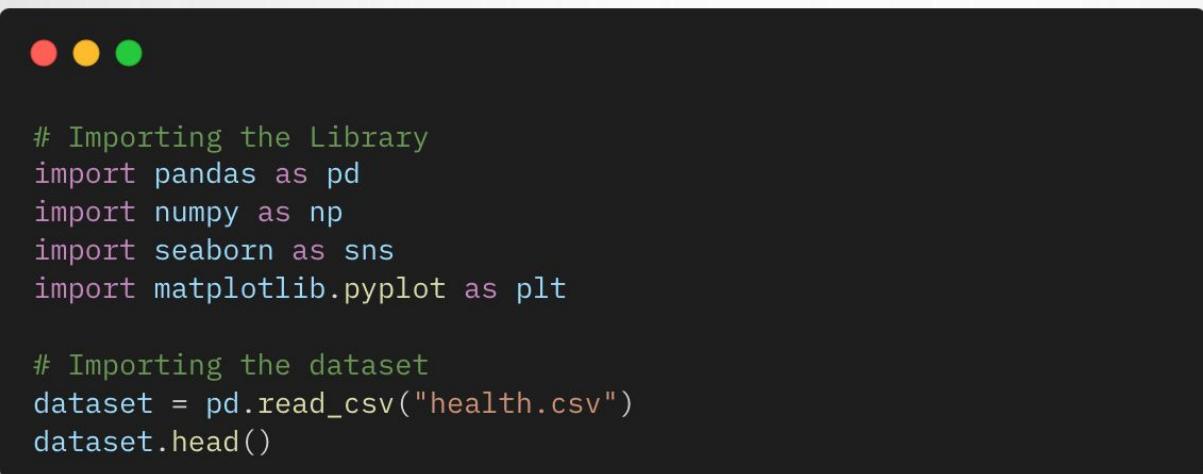
Setelah membahas berbagai macam model atau algoritma machine learning, kita coba belajar bagaimana mengimplementasikannya di python yuk!

Kali ini kita akan mencoba membangun model pada dataset kesehatan. Kita bisa menggunakan salah satu model yang sudah kita bahas, yaitu Random Forest.

Berikut merupakan langkah-langkah dalam mengimplementasikan model machine learning menggunakan python:

## 1. Memasukkan Package dan Dataset

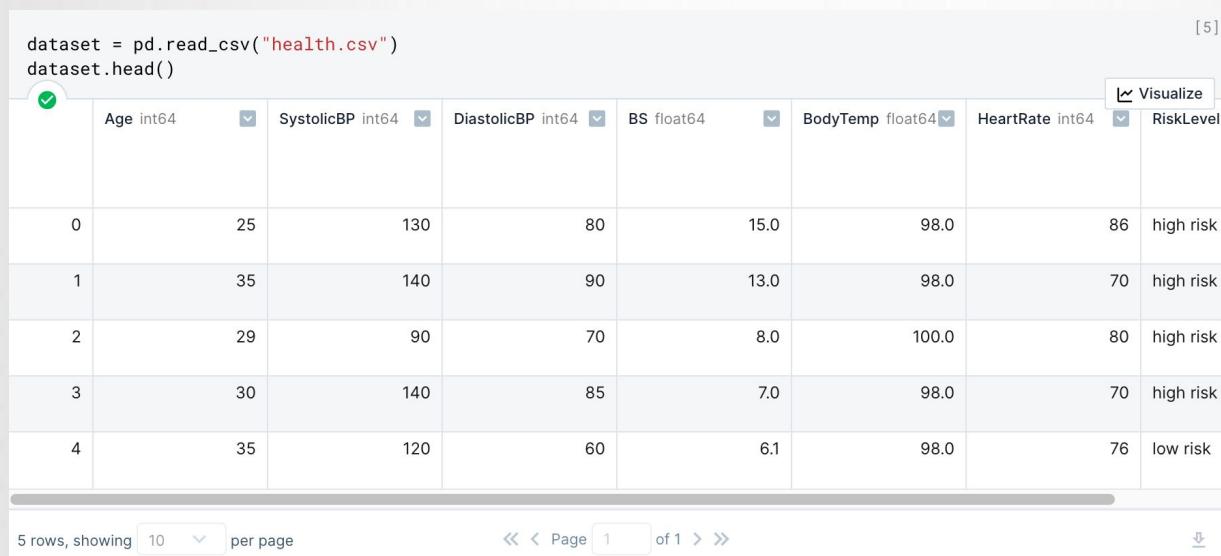
Seperti yang sudah kita bahas sebelumnya, package adalah sekumpulan modul atau fungsi atau kode bahasa pemrograman python yang dapat memudahkan kita menulis perintah. Kali ini kita membutuhkan package pandas, numpy, seaborn, dan matplotlib. Tuliskan script berikut untuk memasukkan package dan dataset.



```
# Importing the Library
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

# Importing the dataset
dataset = pd.read_csv("health.csv")
dataset.head()
```

Kita akan mencoba memprediksi apakah suatu pasien memiliki resiko penyakit yang tinggi atau tidak. Resiko ini direpresentasikan oleh fitur ‘RiskLevel’.

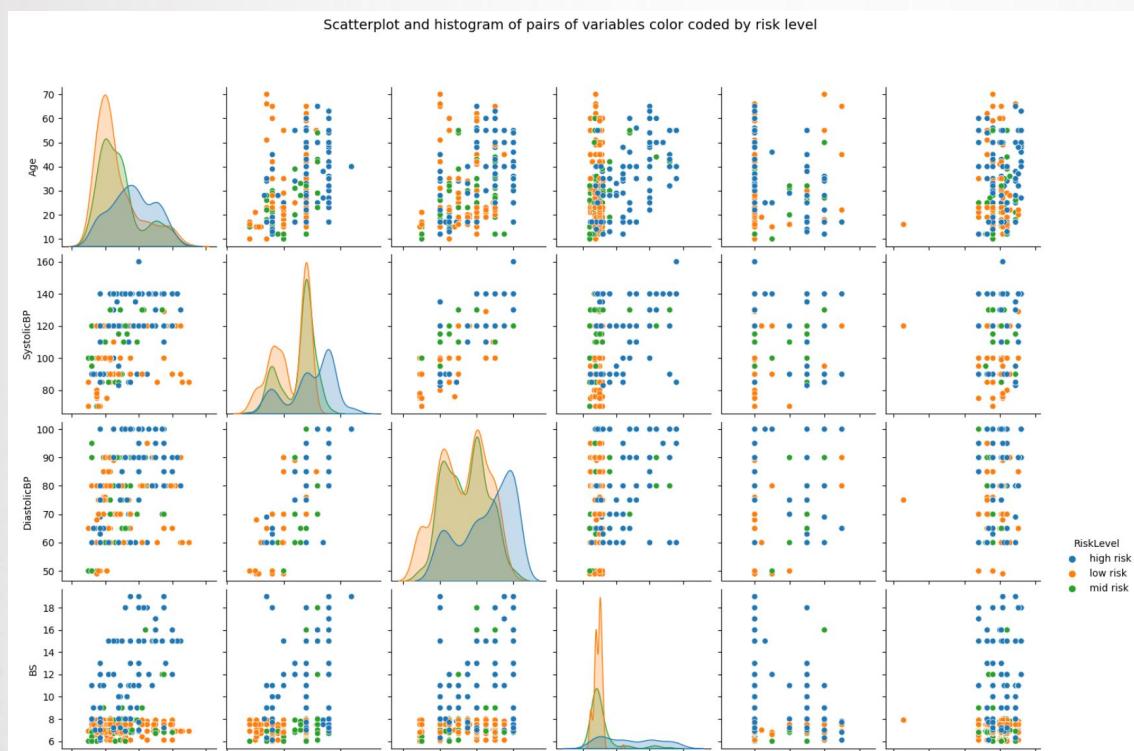


	Age int64	SystolicBP int64	DiastolicBP int64	BS float64	BodyTemp float64	HeartRate int64	RiskLevel
0	25	130	80	15.0	98.0	86	high risk
1	35	140	90	13.0	98.0	70	high risk
2	29	90	70	8.0	100.0	80	high risk
3	30	140	85	7.0	98.0	70	high risk
4	35	120	60	6.1	98.0	76	low risk

Seperti pembahasan di modul sebelumnya, kita **menganalisa dan menyelidiki sebuah dataset** melalui Exploratory Data Analysis (EDA) terlebih dahulu seperti menganalisis bi-/multivariate untuk melihat bagaimana korelasi antar fitur. Seperti yang sudah kita bahas di modul Data Preprocessing, kita bisa menggunakan pair plot untuk efisiensi waktu dalam menganalisis bi-/multivariate. Silahkan tuliskan script berikut untuk menampilkan pair plot:

```
● ● ●
g = sns.pairplot(dataset, hue='RiskLevel')
g.fig.suptitle("Scatterplot and histogram of pairs of variables color coded by risk level",
               fontsize = 14, # defining the size of the title
               y=1.05); # y = definig title y position (height)
```

Gambar berikut adalah pair plot yang dapat membantu kita menganalisis bi-/multivariate tiap fitur. Namun yang harus diingat, proses EDA tidak sebatas hanya analisis ini saja berhubung banyak proses yang bisa dilakukan. Oleh karena itu, coba diulas kembali di modul 2 ya!



Apabila kita telusuri berapa banyak level resiko yang berada pada ‘RiskLevel’ menggunakan script berikut, terdapat tiga jenis resiko.



```
dataset['RiskLevel'].unique()
```

```
array(['high risk', 'low risk', 'mid risk'], dtype=object)
```

## 2. Feature Engineering

Langkah selanjutnya kita bisa melakukan feature engineering sehingga dataset kita bisa dilakukan permodelan. Kita akan mengubah nilai tiap ‘RiskLevel’ menjadi numerik melalui proses label encoding.

Label encoding merupakan teknik yang mengubah nilai non-integer menjadi integer. Sebagai contoh pada dataset kita:

- Nilai ‘low risk’ diubah menjadi 0
- Nilai ‘mid risk’ diubah menjadi 1
- Nilai ‘high risk’ diubah menjadi 2

Tuliskan script berikut untuk mengimplementasikannya di python:



```
dataset['RiskLevel'] = dataset['RiskLevel'].replace('low risk',  
0).replace('mid risk', 1).replace('high risk', 2)
```

Perhatikan dua gambar di bawah, kini tiap nilai ‘RiskLevel’ sudah direpresentasikan menjadi angka setelah kita menerapkan label encoding.



RiskLevel object	RiskLevel int64
high risk	2
low risk	0

Sebelum Label Encoding      Sesudah Label Encoding

### 3. Membagi data menjadi dua bagian

Selanjutnya kita membagi data menjadi 2 bagian dengan proporsi 80% untuk data training dan 20% sisanya untuk testing

- Pertama, kita mendefinisikan variabel “x” sebagai variabel independen dan “y” adalah variabel dependen
- Selanjutnya, kita mengimplementasikan pembagian data menggunakan syntax `train_test_split()` dengan pengaturan seperti berikut:
  - Memasukkan daftar array yang akan kita bagi. Dalam hal ini kita akan membagi array yang sudah didefinisikan sebagai “x” dan “y”
  - Kita akan menentukan proporsi data testing dengan mengatur nilai parameter “`test_size`”
  - Isilah nilai “`random_state`” dengan suatu nilai agar hasil running program tidak berubah tiap kita menjalankannya
- Berhubung kita memasukkan dua buah array, yaitu x dan y, maka tiap array akan menghasilkan dua array baru. Tiap array baru merepresentasikan array untuk fase training dan testing model. Berikut lebih detailnya:
  - Array x akan terbagi menjadi `x_train` untuk training dan `x_test` untuk testing
  - Array y akan terbagi menjadi `y_train` untuk training dan `y_test` untuk testing

```
● ● ●

# Define the Independent and Dependent Variable
y = dataset['RiskLevel']
X = dataset.drop(['RiskLevel'], axis=1)

# Split the dataset into training and testing
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state= 42)
```

## 4. Membangun Model Random Forest

Ketikkan script seperti dibawah. Perhatikan script di bawah “Importing the Library”. Pertama, kita memasukkan library RandomForestClassifier yang disediakan oleh package Sklearn.

Selanjutnya, kita mendefinisikan variabel ‘rfc’ yang menyimpan informasi model Random Forest. Kali ini kita mencoba menentukan beberapa parameter Random Forest seperti:

- `n_estimator = 3`  
Kita menggunakan tiga buah pohon pada model kita
- `max_depth = 5`  
Kita mendefinisikan setiap pohon memiliki kedalaman hingga 5 node

Kita juga perlu mengisi nilai ‘random\_state’ supaya hasil keluaran model tidak berubah tiap kali kita menjalankannya.

```
# Importing the Library
from sklearn.ensemble import RandomForestClassifier

# Define the model
SEED = 42
rfc = RandomForestClassifier(n_estimators=3,
                            max_depth=5,
                            random_state=SEED)
```

Selanjutnya, kita menjalankan syntax `.fit()`. Fungsi tersebut berfungsi membangun model pada data training.

Sama halnya dengan model regresi, `.fit()` membutuhkan parameter dua buah array dimana array yang pertama berisi kumpulan nilai variabel independen (`x_train`) dan array yang kedua berisi kumpulan nilai variabel dependen (“`y_train`”).

```
# Fit the Random Forest model
rfc.fit(X_train, y_train)

# Predict
y_pred = rfc.predict(X_test)
```

## 5. Analisis Model

Analisis model dapat dilakukan melalui dua aktivitas, yaitu kinerja model dan feature importance. Pengukuran kinerja model dapat kita tampilkan dalam confusion matrix. Matriks tersebut biasa digunakan untuk melihat kinerja model klasifikasi secara keseluruhan. Melalui matriks tersebut kita bisa melihat pengukuran kesalahan model seperti akurasi, precision, recall, dan f1-score.

Feature importance adalah teknik mengetahui seberapa besar pengaruh suatu fitur terhadap model machine learning yang direpresentasikan dalam sebuah nilai koefisien.

Pertama, kita harus memasukkan package yang dibutuhkan seperti `classification_report` dan `confusion_matrix`.

Selanjutnya, kita memanggil fungsi `confusion_matrix()` dengan memasukkan `y_test` yang berisi variabel dependen aktual dan `y_pred` yang berisi hasil prediksi model. Hampir sama seperti penerapan fungsi `predict()` bukan?



```
# Importing the Library
from sklearn.metrics import classification_report,
confusion_matrix

# Define the Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt='d').set_title('Maternal risks
confusion matrix (0 = low risk, 1 = medium risk, 2 = high risk)')

print(classification_report(y_test,y_pred))
```

Apabila kita memasukkan fungsi `confusion_matrix()` ke dalam `print()`, maka akan tertampil gambar berikut. Terlihat bahwa nilai akurasi model adalah 0,66 atau sekitar 66%. Sebenarnya pengukuran kinerja model merupakan faktor penting dalam membangun model. Namun, kali ini kita hanya berfokus pada membangun model saja. Pembahasan lebih detail bisa dicek di modul selanjutnya, ya!

	precision	recall	f1-score	support
0	0.60	0.89	0.71	80
1	0.65	0.34	0.45	76
2	0.82	0.77	0.79	47
accuracy			0.66	203
macro avg	0.69	0.67	0.65	203
weighted avg	0.67	0.66	0.63	203

Kamu bisa mengakses source code penerapan Random Forest [disini](#) ya!

Pembahasan yang sudah kita bahas merupakan proses menerapkan suatu model machine learning pada suatu dataset. Sebenarnya, kita bisa menggunakan model lain seperti Decision Tree, Gradient Boosting, dan XGBoost. Bagaimana caranya kalau kita ingin mencoba model lain?

Kalau melihat script berikut, sebenarnya kita cukup berfokus pada bagaimana memanggil fungsi RandomForestClassifier() berikut:

```
● ● ●

# Importing the Library
from sklearn.ensemble import RandomForestClassifier

# Define the model
SEED = 42
rfc = RandomForestClassifier(n_estimators=3,
                             max_depth=5,
                             random_state=SEED)
```

Kita bisa mengganti RandomForestClassifier() tersebut dengan fungsi lain sesuai model yang ingin kita coba. Misalkan kita ingin mencoba Decision Tree, kita bisa memanggil fungsi DecisionTreeClassifier yang disediakan oleh Sklearn yang kemudian kita memanggil fungsi **.fit()**.

```
● ● ●

# Importing the Library
from sklearn.ensemble import DecisionTreeClassifier

SEED = 42
rfc = DecisionTreeClassifier(max_depth=5, random_state=SEED)
rfc.fit(X_train, y_train)
```

# Feature Importance

Selanjutnya, kita **melihat seberapa penting sebuah fitur dalam memprediksi target variabel melalui analisis feature importance**. Analisis ini dapat digunakan untuk melihat ranking tingkat pengaruh suatu fitur terhadap variabel target seperti gambar di bawah.

Feature importance dapat memberikan informasi berharga dari beberapa sisi seperti:

- Dataset

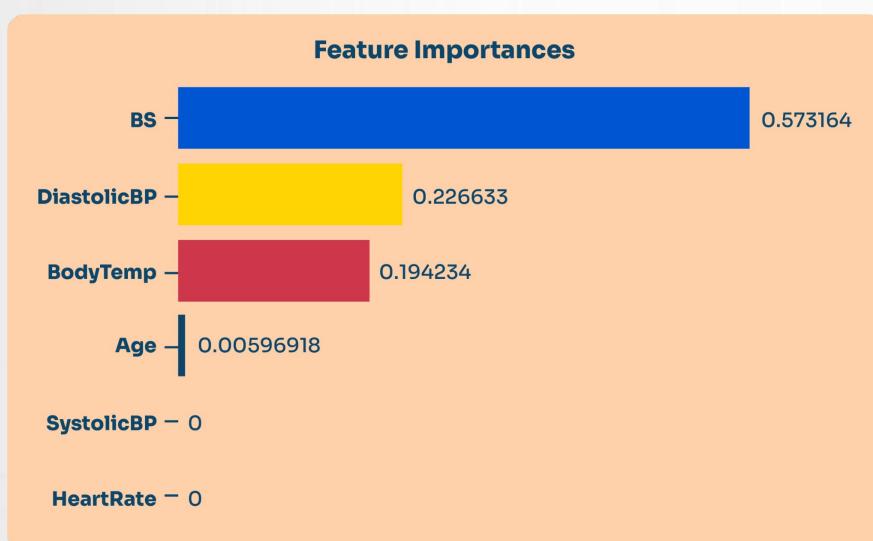
Nilai koefisien dapat menekankan fitur apa saja yang paling relevan dengan target kelas. Semakin tinggi nilai koefisien suatu fitur mengindikasikan fitur tersebut berpengaruh besar pada suatu dataset

- Model machine learning

Pengukuran nilai koefisien biasa dilakukan setelah kita membangun model. Setelah model mempelajari karakteristik dataset kita, biasanya model tersebut bisa memberi informasi fitur mana yang paling relevan dengan target kelas

- Sarana meningkatkan kinerja model

Setelah mengetahui fitur mana yang paling relevan, kita bisa menggunakan fitur tersebut untuk membangun model yang lebih baik. Hal ini dikarenakan, kita hanya menggunakan fitur yang benar-benar berdampak untuk mengklasifikasikan data. Fitur yang dianggap tidak relevan dapat kita abaikan dalam membangun model



Ketika memilih rumah, pasti ada beberapa pertimbangan seperti:

1. Jarak dengan tempat kerja?
2. Lingkungannya bagaimana? Sering banjir? Sering terjadi kasus pencurian? Akses jalannya sepi?
3. Dekat tempat umum seperti mall/pasar/stasiun?
4. Fasilitas perumahan serta sertifikat kepemilikannya nanti bagaimana

Dari beberapa pertimbangan tersebut, pasti ada salah satu yang menjadi prioritas utama kita dalam memilih rumah seperti mengutamakan lingkungan yang bebas banjir. Selain itu, rumah kita bisa dekat dengan tempat kerja sehingga tidak mengeluarkan biaya lebih besar untuk transportasi. Beberapa pertanyaan atau faktor itu salah satu contoh penerapan feature importance dalam kehidupan sehari-hari.



Apabila kita mengulas kembali ke pembahasan sebelumnya, kita telah membahas bagaimana membangun model Random Forest untuk memprediksi level resiko pada dataset kesehatan. Pembahasannya bisa kalian akses [disini](#) untuk mengingat kembali.

Sekarang kita coba implementasi bagaimana menerapkan feature importance yang dihasilkan oleh model Random Forest di python yuk! Silahkan coba tuliskan script di bawah ini.



```
# Organizing feature names and importances in a DataFrame
features_df = pd.DataFrame({'features': rfc.feature_names_in_,
                            'importances': rfc.feature_importances_})

# Sorting data from highest to lowest
features_df_sorted = features_df.sort_values(by='importances',
                                              ascending=False)

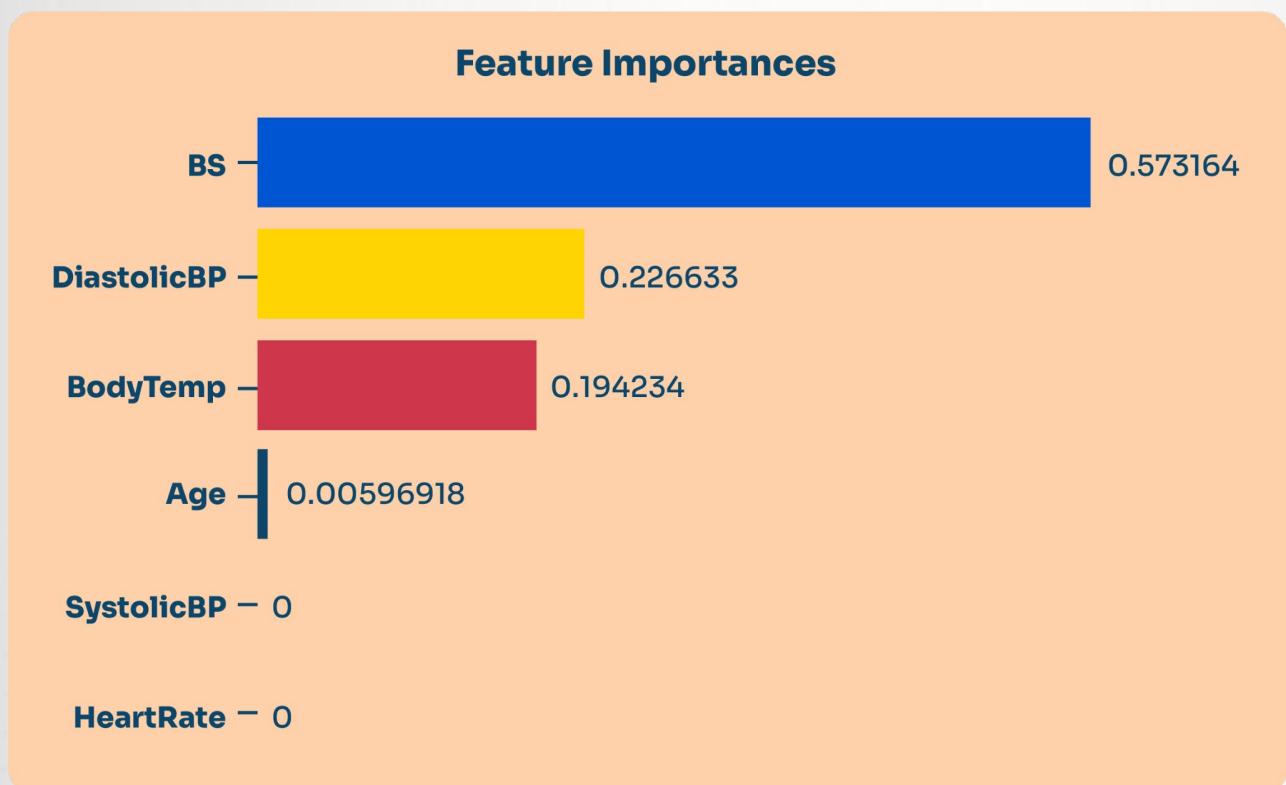
# Barplot of the result without borders and axis lines
g = sns.barplot(data=features_df_sorted, x='importances', y='features', palette="rocket")
sns.despine(bottom = True, left = True)
g.set_title('Feature importances')
g.set(xlabel=None)
g.set(ylabel=None)
g.set(xticks[])
for value in g.containers:
    g.bar_label(value, padding=2)
```

Pertama, kita akan membuat dataframe atau tabel yang menampilkan informasi nama fitur dan nilai koefisien yang merepresentasikan “kepentingan” tiap fitur terhadap model. Semakin besar nilai koefisien, maka semakin besar pengaruh fitur tersebut terhadap model.

Selanjutnya, kita mengurutkan berdasarkan nilai koefisien terbesar untuk mendapat informasi fitur mana yang memiliki nilai koefisien terbesar. Kita dapat menampilkan nilai koefisien fitur yang telah urut dalam sebuah grafik berikut supaya lebih intuitif. Hasilnya, tiga fitur dengan nilai koefisien terbesar:

- ‘BS’ yang merepresentasikan kandungan gula dalam darah
- ‘DiastolicBP’ yang merepresentasikan tekanan darah diastolic atau tekanan darah yang berada di arteri
- ‘BodyTemp’ yang merepresentasikan suhu tubuh

Hasil analisis ini dapat memberikan informasi terkait faktor mana saja yang mempengaruhi level resiko kesehatan, yaitu mulai dari kandungan gula dalam darah, tekanan darah diastolic, dan suhu tubuh. Selain itu, diketahui bahwa tekanan darah di jantung yang direpresentasikan oleh ‘SystolicBP’ dan detak jantung relatif tidak berpengaruh pada level resiko kesehatan karena memiliki nilai koefisien nol.



Kamu bisa mengakses source code penerapan feature importance [disini](#) ya!

Kita bisa mengganti `RandomForestClassifier()` tersebut dengan `GradientBoostingClassifier()` yang kemudian kita memanggil fungsi `.fit()` untuk membangun model menggunakan Gradient Boosting seperti terlihat di script berikut:

```
# Importing the Library
from sklearn.ensemble import GradientBoostingClassifier

SEED = 42
rfc = GradientBoostingClassifier(n_estimators=100,
learning_rate=1.0,max_depth=1, random_state=SEED)

# Fit the model to training set
rfc.fit(X_train, y_train)
```

Berbeda halnya dengan XGBoost, kita harus me-install package `xgboost` pada komputer kita. Hal ini dikarenakan package tersebut biasanya masih belum tersedia di komputer kita. Kita harus menuliskan script berikut untuk me-install package:

**!pip install xgboost**

Kemudian kita bisa menggunakan XGBoost seperti model lain pada umumnya

```
# Installing the Library
!pip install xgboost

# Importing the Library
from xgboost import XGBClassifier

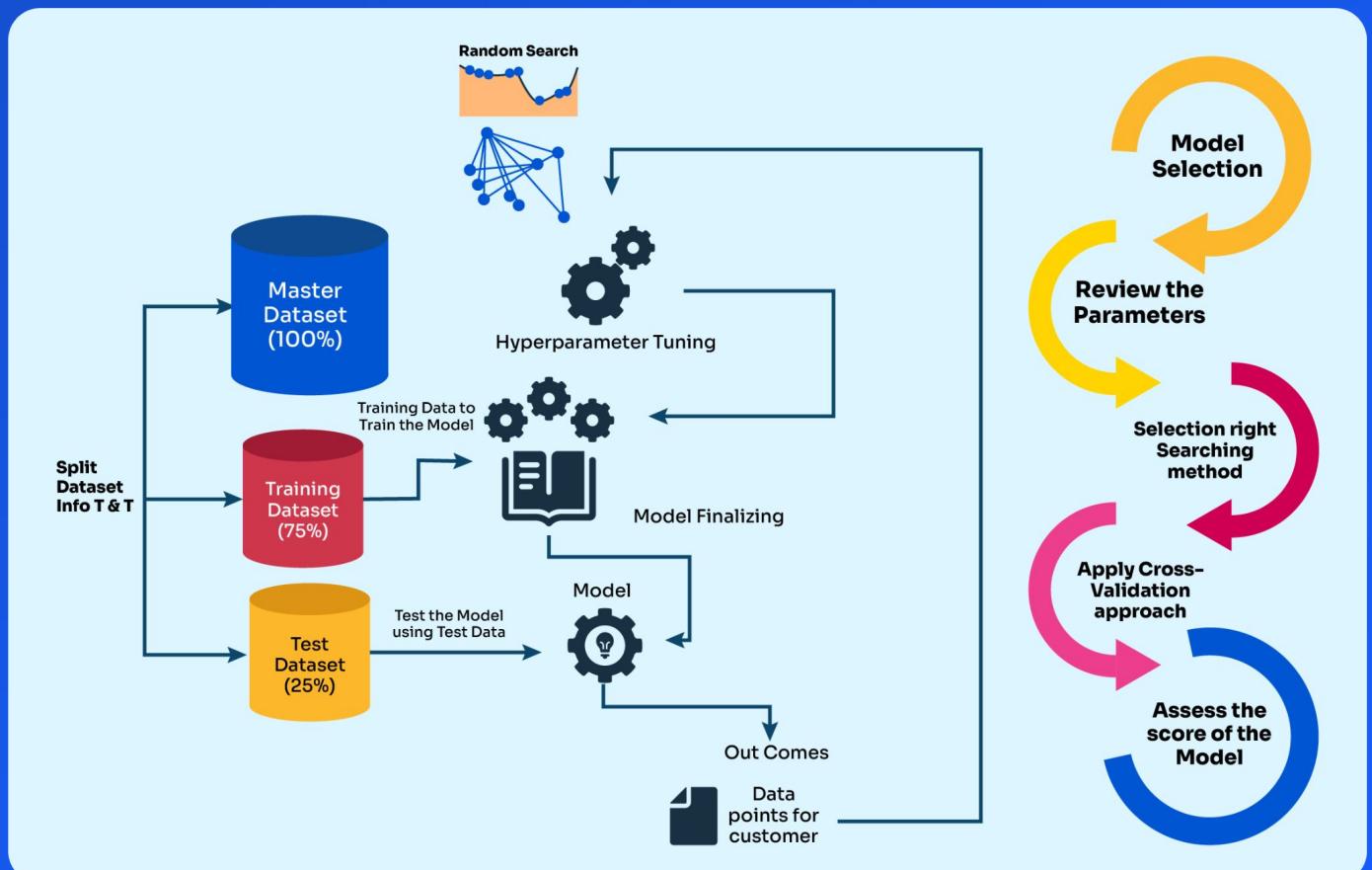
SEED = 42
rfc = XGBClassifier()

# Fit the model to training set
rfc.fit(X_train, y_train)
```

Sebenarnya, setiap model machine learning memiliki beberapa nilai parameter yang dapat dikonfigurasi. Sebagai contoh, kita bisa melakukan eksperimen pada jumlah pohon pada decision tree atau random forest. Mengapa harus kita harus eksperimen nilai ini?

Hal ini dikarenakan konfigurasi parameter yang berbeda, bisa saja hasilnya juga berbeda seperti peningkatan kinerja model. Proses konfigurasi parameter ini biasa disebut hyperparameter tuning.

Hyperparameter tuning adalah sebuah proses dalam menentukan kombinasi yang tepat bagi setiap hyperparameter yang ada di setiap model, yang dapat memaksimalkan hasil prediksi model tersebut.



## EXERCISE

Berikut adalah langkah - langkah untuk mengerjakan *exercise*:

1. Buka *link* Google Colab menggunakan Google Chrome.
2. Klik ‘File’.
3. Klik ‘Save a Copy in Drive’.
4. Kerjakan *exercise* dengan mengisi coding pada baris yang tertera dan menjawab Insight yang ditanyakan.
5. Untuk baris coding yang sudah terisi, anda disarankan untuk langsung melakukan run baris tersebut tanpa mengubah isinya.
6. Perhatikan instruksi lain yang tertera pada Google Colab.

Google Colab dapat diakses pada:

[Exercise Modul 4 DS](#)

Nb:

Pada exercise modul 4 ini, terdapat kunci jawaban exercise modul 2 dan 3.



**THANK YOU**