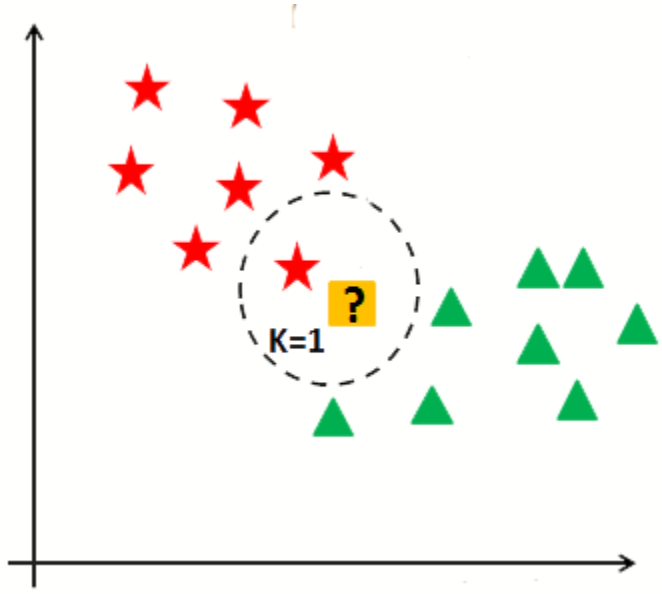
Klasifikasi Bunga Iris menggunakan KNN Python

Klasifikasi merupakan salah satu kegiatan yang paling sering dilakukan menggunakan *machine learning*. Pada artikel ini kamu akan mencoba untuk membuat sebuah model untuk melakukan klasifikasi pada spesies bunga iris (Fisher, 1936). Bunga iris memiliki tiga spesies, yaitu: *versicolor, virginica*, dan *setosa*. Dataset yang digunakan berisi informasi panjang dan lebar untuk masing-masing kelopak pada bunga (sepal dan petal). Klasifikasi yang akan kita buat ini termasuk ke dalam jenis *multiclass classification*, karena terdapat lebih dari dua kelas output.

K-Nearest Neighbor

K-Nearest Neighbor merupakan algoritma untuk melakukan klasifikasi dengan cara membandingkan jarak ketetanggaan tiap-tiap titik data dengan data yang akan diprediksi. Pada algoritma ini metode jarak yang digunakan biasanya adalah jarak Euclidean. Data baru akan dibandingkan jarak terdekat dengan tetangganya, semakin banyak tetangga dengan jarak yang dekat, maka data tersebut akan masuk ke dalam kelas tersebut.



Visualisasi KNN.

Untuk membuat sebuah model klasifikasi ini, kita akan melakukan beberapa proses, yaitu:

1. *Data ingestion*
    i. Membaca data dari file CSV.
2. *Exploratory data analysis*
    i. Statistik deskriptif data.
    ii. Sampel data teratas.
    iii. *Pair plot* distribusi data.
3. *Preprocessing*
    i. Memisahkan *features* dan *label*.
    ii. Membagi data latih dan uji.
    iii. *Label encoding*.
    iv. Standarisasi data.
    v. Menentukan nilai k berdasarkan *mean absolute error*.
4. *Training*, menggunakan algoritma KNN.
5. *Evaluation*, menggunakan *confusion matrix*.

Data Ingestion

Sebelum dapat membaca data, kita perlu mengimpor *library* yang akan digunakan. Pada artikel ini kita akan menggunakan *library* NumPy, Pandas, Matplotlib, Seaborn, dan Scikit-Learn. Kelima *library* ini digunakan untuk membaca, mengolah, dan melakukan *training* model

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import ConfusionMatrixDisplay
```

Setelah mengimpor *library* yang akan digunakan, selanjutnya kamu akan membaca file CSV dengan memanggil fungsi **read_csv**. Data yang dibaca akan disimpan sebagai objek DataFrame. Objek DataFrame ini akan menampung data seperti sebuah tabel. Kamu dapat melakukan berbagai pengolahan data pada objek DataFrame ini.

```
columns = ["sepal-length", "sepal-width", "petal-length", "petal-width",
"class"]
df = pd.read-csv(r'https://archive.ics.uci.edu/ml/machine-learning-
databases/iris/iris.data', names=columns)
```

Exploratory Data Analysis

Setelah kamu memuat data yang akan diolah sebagai DataFrame, kammu dapat mulai melakukan eksplorasi data. Proses eksplorasi data yang akan dilakukan yaitu menghitung statistika deskripsi data, melihat sampel data teratas, dan melakukan visualisasi sebaran data dan distribusinya.

Statistik Deskriptif Data

```
df.describe()
```

	sepal-length	sepal-width	petal-length	petal-width
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.054000	3.758667	1.198667
std	0.828066	0.433594	1.764420	0.763161
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

Dapat dilihat pada output di atas masing-masing kelas terdapat 150 data. Semua atribut memiliki *mean* dan standar deviasi yang berbeda-beda pula. Jika kita melihat pada *min* dan *max*, terlihat bahwa *range* antar atribut juga bervariasi.

Karena kamu akan menggunakan model KNN, maka data perlu di standarisasi sebelum digunakan untuk melakukan *training*. Hal ini karena KNN menggunakan prinsip metrik jarak dengan Euclidean. Apabila terdapat data dengan *range* yang jauh, maka dengan melakukan standarisasi akan mempercepat proses *training* dan meningkatkan akurasi model.

Sampel Data

```
df.head()
```

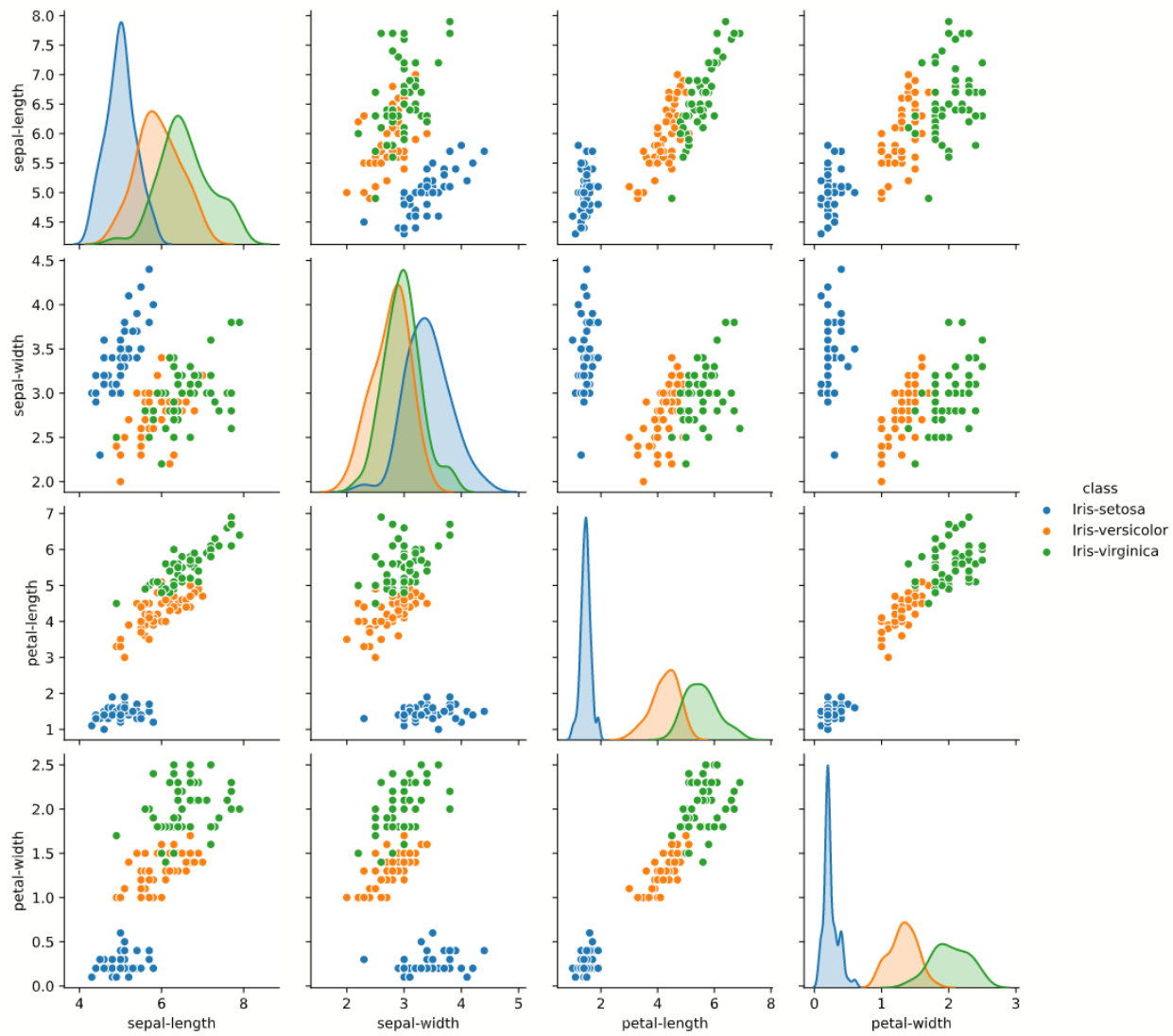
	sepal-length	sepal-width	petal-length	petal-width	class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

Dari output di atas dapat dilihat ternyata data di dalam file CSV disusun berurutan berdasarkan kelas. Artinya sebelum melakukan *training* data harus diacak dan dibagi seimbang.

Pair Plot

Pada tahap ini kamu akan membuat pair plot untuk membantu memvisualisasikan hubungan antar variabel pada dataset.

```
sns.pairplot(df, hue='class')
```



Pair Plot antar Variabel.

Berdasarkan plot yang dibuat, dapat dilihat bahwa kelas *iris-setosa* selalu terpisah dari kelas yang lain. Artinya saat melakukan klasifikasi terdapat kemungkinan besar bahwa model akan selalu dapat membedakan spesies *setosa* dengan baik. Dapat dilihat juga distribusi data untuk *petal-length*, spesies *setosa* terpisah dari kelas yang lain.

Selain itu, jika dilihat persebaran datanya pada diagram pencar, sebagian besar kombinasi atribut miliki korelasi Pearson yang positif. Artinya *features* yang terdapat pada dataset ini baik untuk digunakan untuk membuat sebuah model. Selain itu, dengan korelasi yang tinggi, maka model klasifikasi dapat diubah menjadi model regresi untuk melakukan peramalan.

Preprocessing

Memisahkan Features dan Label

Pada tahap ini kamu melakukan operasi *slicing* untuk membagi data dari dalam objek DataFrame. Pada tahap ini juga kamu akan mengambil data sebagai *array*, bukan objek DataFrame. Hal ini dilakukan karena *library* scikit-learn menggunakan tipe data *NumPy array* untuk digunakan dalam algoritmanya.

```
X = df.iloc[:, :-1].values
y = df.iloc[:, 4].values
```

Membagi Data Latih dan Uji

Pada tahap ini kamu akan membagi 150 data untuk latih dan uji. Kamu akan menggunakan fungsi **train_test_split** untuk memudahkan proses pembagian data agar data yang dibagi seimbang.

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

Label Encoding

Karena kelas dari dataset ini berupa teks, maka perlu dilakukan *encoding*. Pada kasus ini kamu akan menggunakan *label encoding*, karena internal algoritma ini menggunakan ketetanggan, maka nilai output tidak akan terbias dari penggunaan teknik *encoding* ini.

```
lb = LabelEncoder()
lb.fit(y_train)

y_train = lb.transform(y_train)
y_test = lb.transform(y_test)
```

Standarisasi

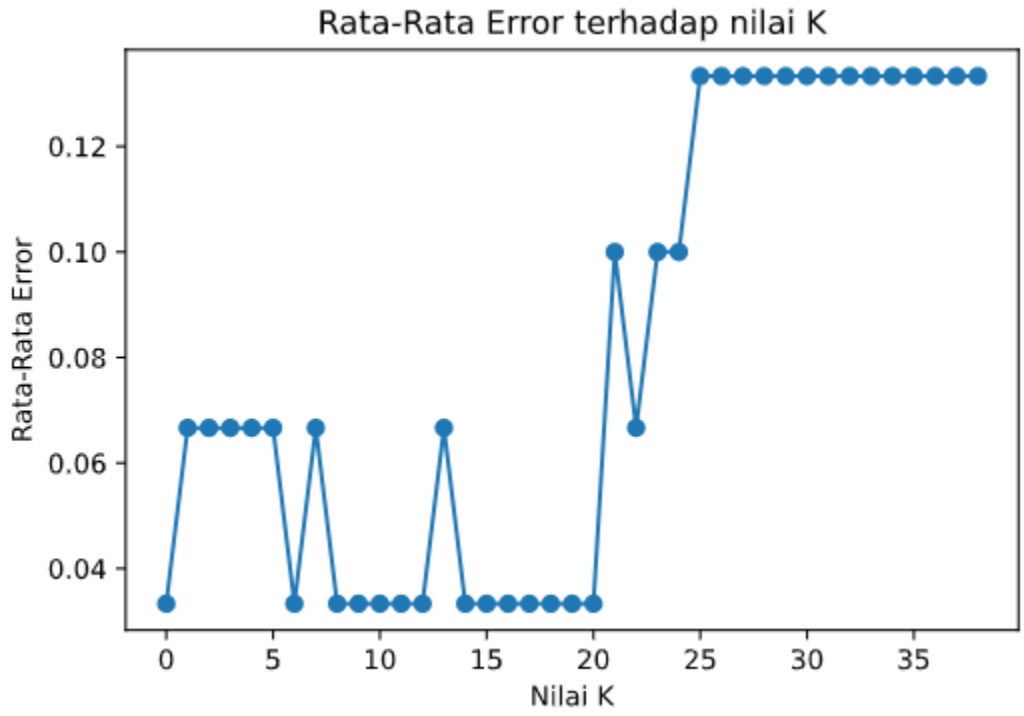
Algoritma KNN menggunakan metrik jarak (Euclidean), sehingga memiliki data yang terdistribusi merata khususnya pada distribusi normal akan meningkatkan performa model KNN ini. Maka dari itu diperlukan proses standarisasi data.

```
scaler = StandardScaler()
scaler.fit(X_train)

X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

Menentukan Nilai Konstanta k

Seperti yang sudah di jelaskan sebelumnya, bahwa KNN membutuhkan suatu nilai konstanta k untuk menentukan berapa banyak tetangga yang akan digunakan oleh model. Kode di bawah ini akan melakukan *training* sebanyak 4o kali dengan nilai k dari 1 hingga 40. Angka 40 ini bersifat bebas dan dapat diganti dengan angka lain.

```
error = []

for i in range(1, 40):
    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(X_train, y_train)

    pred_i = knn.predict(X_test)
    error.append(np.mean(pred_i != y_test))

plt.figure()
plt.plot(range(1, 40), error, color='red', marker='o',
         markerfacecolor='blue', markersize=10)
plt.title('Rata-Rata Error terhadap nilai K')
plt.xlabel('Nilai K')
plt.ylabel('Rata-Rata Error')
plt.show()
```



Dapat dilihat pada grafik yang dihasilkan, bahwa terdapat beberapa nilai k yang dapat meminimalisasi nilai *error*. Contohnya nilai k awal yang digunakan adalah 40, dapat

dilihat pada grafik bahwa nilai $k=40$ memiliki tingkat *error* sebesar 0,17. Ternyata nilai k yang kecil seperti 1, 2, 4, dan 5 memiliki tingkat *error* yang kecil.

Setelah mengetahui nilai k dengan *error* yang rendah, kamu dapat melatih kembali model kamu dengan mengubah nilai k sesuai dengan grafik yang ada. Teknik ini merupakan salah satu teknik sederhana untuk menghasilkan model yang lebih baik. Terdapat banyak cara lain untuk meningkatkan akurasi dan performa suatu model yang akan di bahas pada bab selanjutnya.

Training

Pada proses *training* ini kamu akan menggunakan nilai konstanta $k = 4$. Kamu bisa mengubah angka ini sesuai dengan keinginanmu sesuai pada hasil observasi pada tahap *preprocessing* sebelumnya.

```
classifier = KNeighborsClassifier(n_neighbors=4)
classifier.fit(X_train, y_train)
```

Evaluation

Setelah proses *training*, kamu akan mengevaluasi apakah model yang kamu buat memiliki akurasi yang baik atau belum.

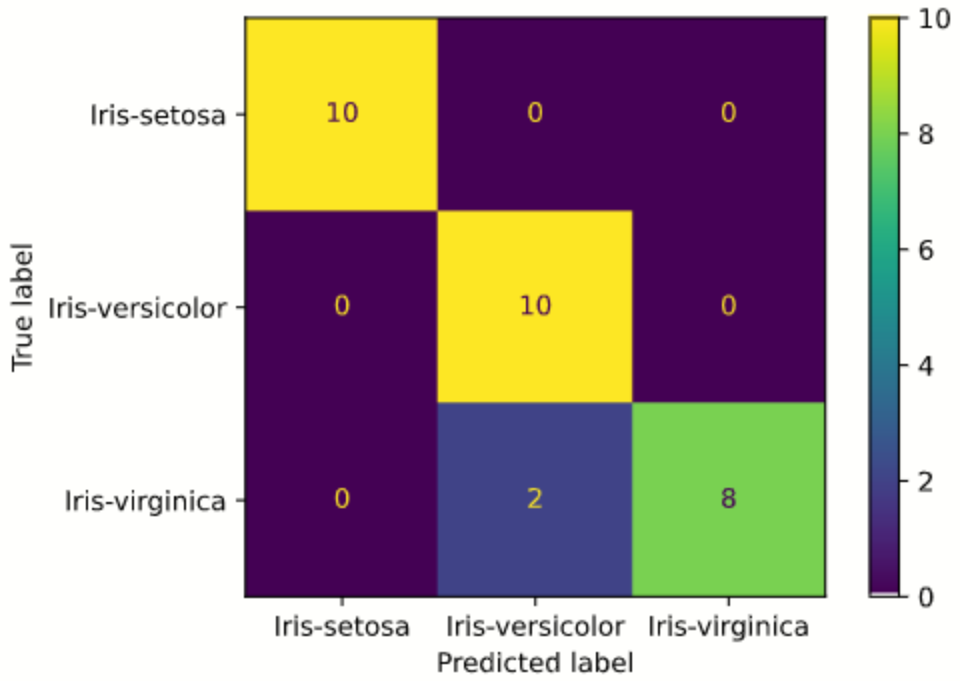
```
y_pred = classifier.predict(X_test)
print(classification_report(y_test, y_pred, target_names=lb.classes_))
```

```
precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        10
Iris-versicolor       0.83      1.00      0.91        10
 Iris-virginica       1.00      0.80      0.89        10

       accuracy                           0.93        30
      macro avg       0.94      0.93      0.93        30
   weighted avg       0.94      0.93      0.93        30
```

Berdasarkan hasil output dari **classification_report**, diketahui bahwa model dapat membedakan semua data dengan spesies *iris-setosa* dengan sempurna. Tetapi pada kelas lain seperti *iris-versicolor* dan *iris-virginica* masih terdapat kesalahan klasifikasi yang ditandai dengan *precision* dan *recall* yang tidak mencapai angka 1.

Secara keseluruhan model ini berhasil mendapatkan akurasi sebesar 93%. Hasil ini sudah sangat baik melihat juga pada nilai *F1 score* yang tinggi menandakan bahwa sebaran hasil klasifikasi yang seimbang untuk kelas-kelas lainnya.

```
cm = confusion_matrix(y_test, y_pred)
ConfusionMatrixDisplay(cm, display_labels=lb.classes_).plot()
```


Iris-setosa
Iris-versicolor
Iris-virginica
True label
10
0
0
0
10
0
0
2
8
Iris-setosa
Iris-versicolor
Iris-virginica
Predicted label
10
8
6
4
2
0