Project Report

Name: Ronit Mehta **Roll No. 16010421056** **Subject: AI-ML**

Title: Mini-Project

Activity:

1. Choose a real-world problem to solve using machine learning (e.g., sentiment analysis, image recognition, predicting housing prices) based on personal interest and feasibility.

2. Collect or obtain a dataset related to the chosen problem. Preprocess the data to handle missing values, outliers, and other data inconsistencies.

3. Perform EDA to understand the dataset's characteristics, distributions, correlations, and patterns that might impact model selection and performance.

4. Choose appropriate machine learning algorithms (e.g., regression, classification, clustering) based on the problem. Implement and train these algorithms using popularlibraries like scikit-learn or TensorFlow.

5. Evaluate the models using appropriate metrics (e.g., accuracy, precision, recall) and perform hyper-parameter tuning to improve model performance.

6. Document the entire project, including problem statement, dataset details, preprocessing steps, model selection, results, and conclusion.

Title: Customer Segmentation Analysis

Problem statement:

This project aims to leverage the Iris ML dataset to develop a precise classification model that effectively distinguishes between the different iris species, facilitating better understanding and application of machine learning algorithms for species identification and botanical research.

Dataset: marketing_campaign.xlsx

Description of dataset:

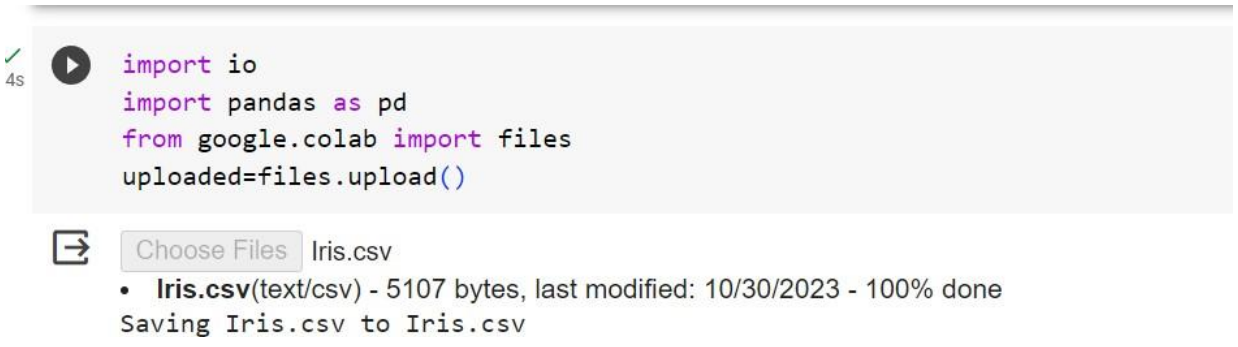
The Iris dataset was used in R.A. Fisher's classic 1936 paper, The Use of Multiple Measurements in Taxonomic Problems, and can also be found on the UCI Machine Learning Repository.

It includes three iris species with 50 samples each as well as some properties about each flower. One flower species is linearly separable from the other two, but the other two are not linearly separable from each other.

The columns in this dataset are:

- Id
- SepalLengthCm
- SepalWidthCm
- PetalLengthCm
- PetalWidthCm
- Species

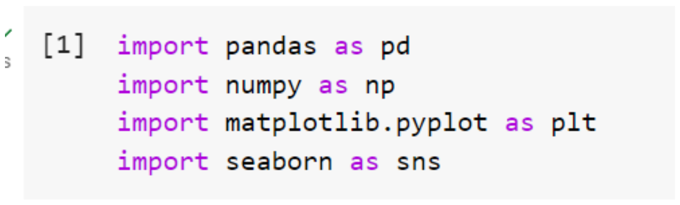
Code:



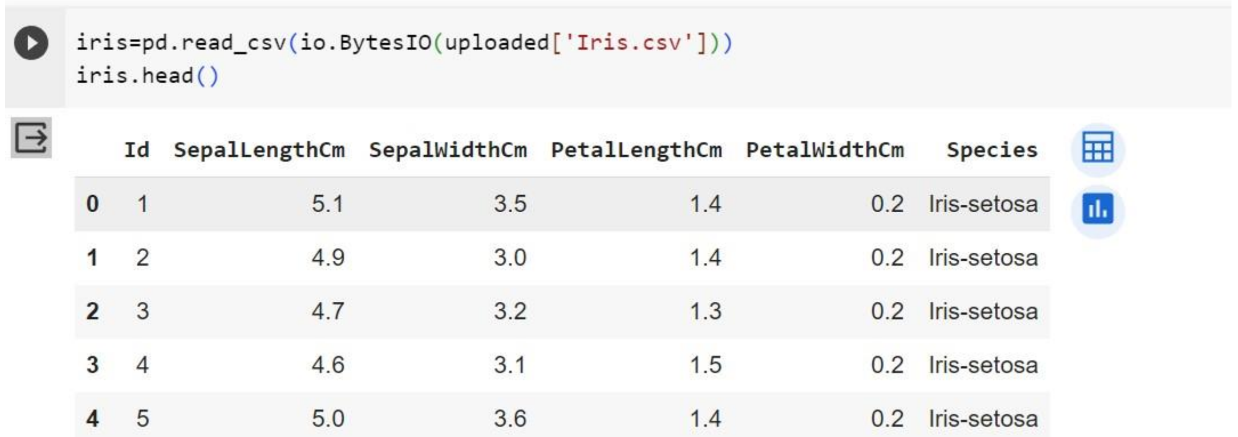
```python
import io
import pandas as pd
from google.colab import files
uploaded=files.upload()
```

```
Choose Files  Iris.csv
• Iris.csv(text/csv) - 5107 bytes, last modified: 10/30/2023 - 100% done
Saving Iris.csv to Iris.csv
```

Importing libraries:



```python
[1] import pandas as pd
    import numpy as np
    import matplotlib.pyplot as plt
    import seaborn as sns
```



```python
iris=pd.read_csv(io.BytesIO(uploaded['Iris.csv']))
iris.head()
```

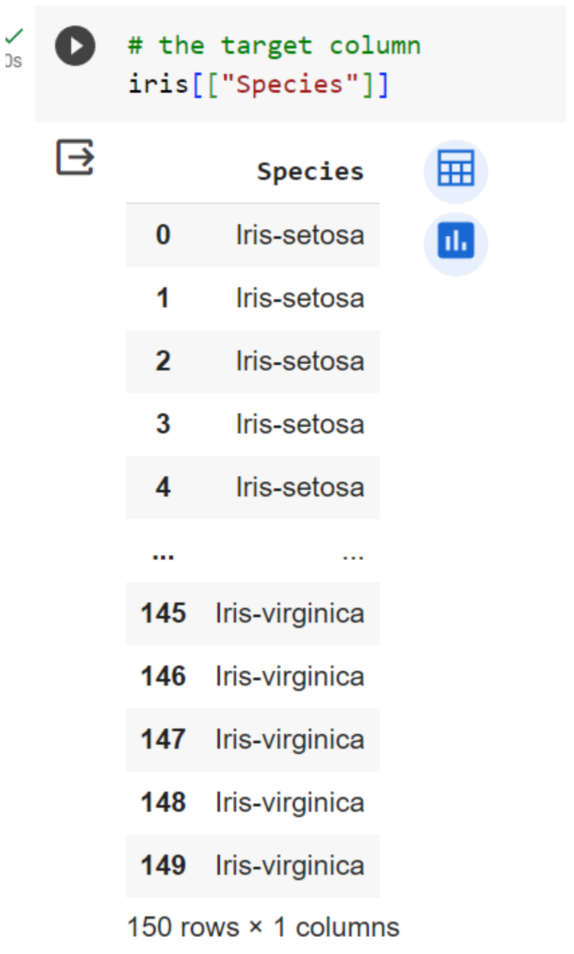
	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa

```
[5] # some info about the dataset
    # no missing data
    iris.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 6 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Id             150 non-null    int64  
 1   SepalLengthCm  150 non-null    float64
 2   SepalWidthCm   150 non-null    float64
 3   PetalLengthCm  150 non-null    float64
 4   PetalWidthCm   150 non-null    float64
 5   Species        150 non-null    object 
dtypes: float64(4), int64(1), object(1)
memory usage: 7.2+ KB
```

```
# describe
iris.describe()
```

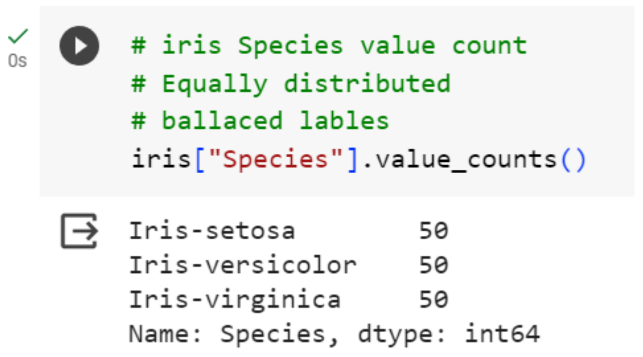
	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
count	150.000000	150.000000	150.000000	150.000000	150.000000
mean	75.500000	5.843333	3.054000	3.758667	1.198667
std	43.445368	0.828066	0.433594	1.764420	0.763161
min	1.000000	4.300000	2.000000	1.000000	0.100000
25%	38.250000	5.100000	2.800000	1.600000	0.300000
50%	75.500000	5.800000	3.000000	4.350000	1.300000
75%	112.750000	6.400000	3.300000	5.100000	1.800000
max	150.000000	7.900000	4.400000	6.900000	2.500000



```
# the target column
iris[["Species"]]
```

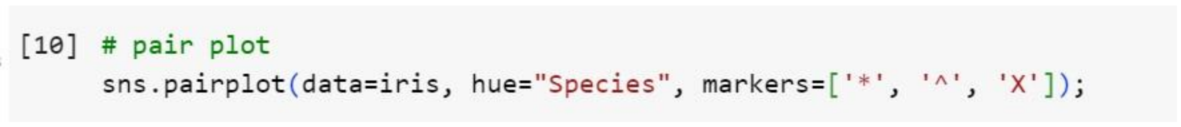
	Species
0	Iris-setosa
1	Iris-setosa
2	Iris-setosa
3	Iris-setosa
4	Iris-setosa
...	...
145	Iris-virginica
146	Iris-virginica
147	Iris-virginica
148	Iris-virginica
149	Iris-virginica

150 rows × 1 columns

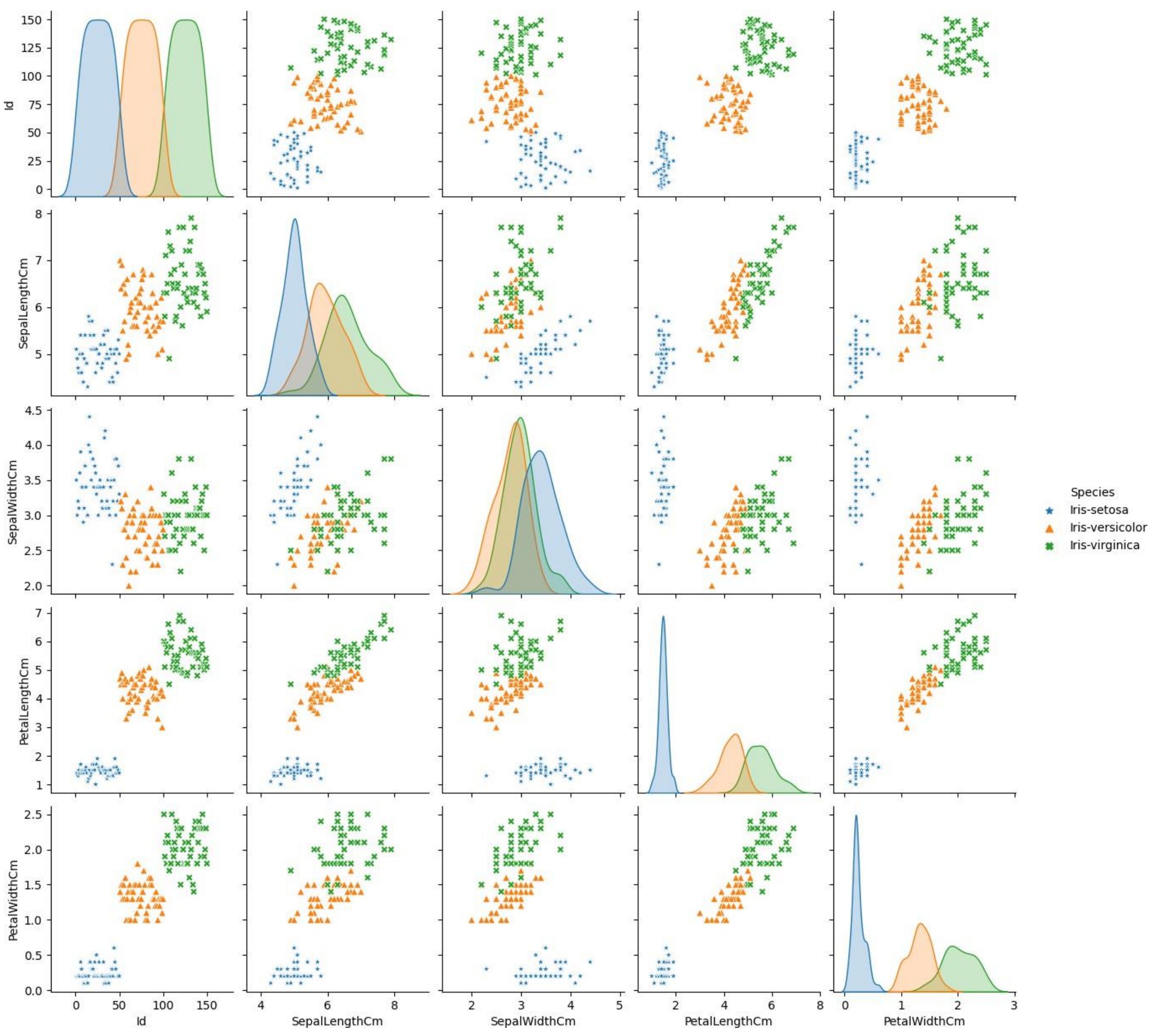


```
# iris Species value count
# Equally distributed
# ballaced lables
iris["Species"].value_counts()
```

```
Iris-setosa        50
Iris-versicolor    50
Iris-virginica     50
Name: Species, dtype: int64
```



```
[10] # pair plot
     sns.pairplot(data=iris, hue="Species", markers=['*', '^', 'X']);
```

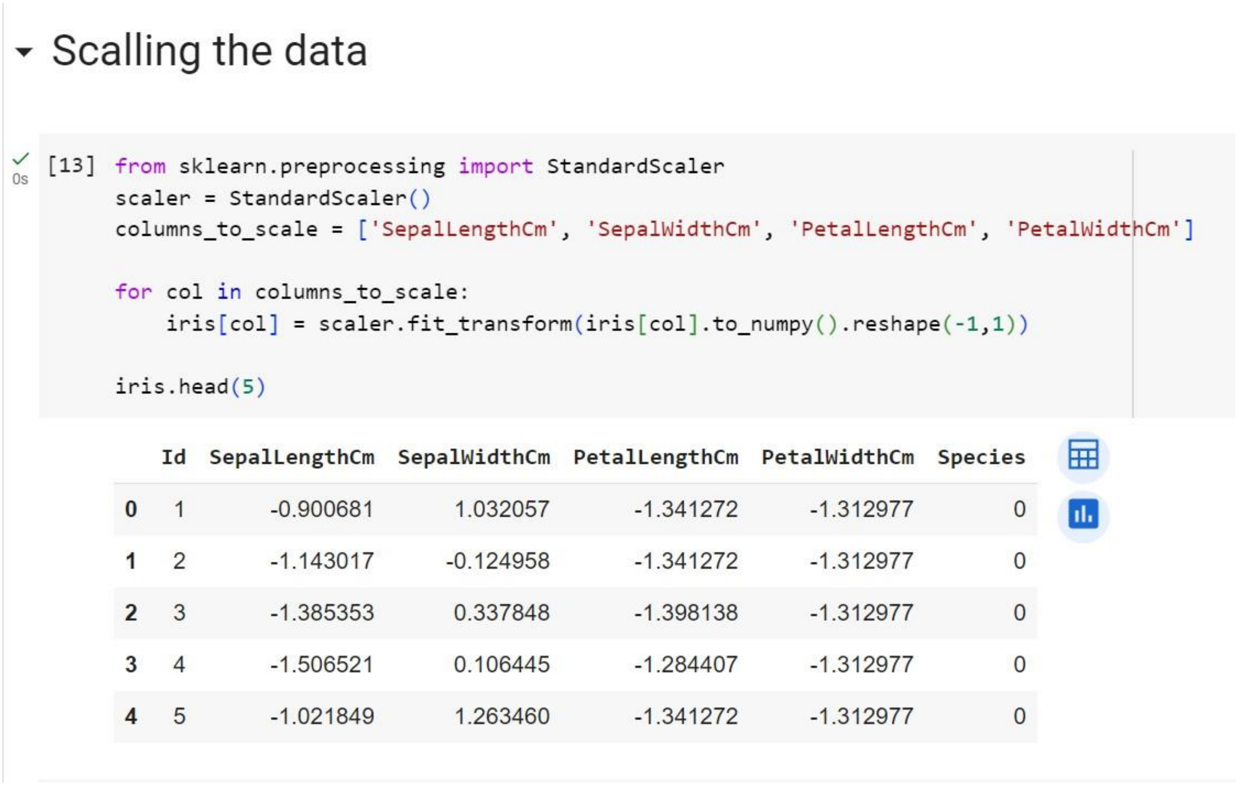


```
[11] # let's encode the label column
     from sklearn.preprocessing import LabelEncoder
     encoder = LabelEncoder()
     iris["Species"] = encoder.fit_transform(iris["Species"])
     iris[["Species"]]

     '''
     Iris-setosa        lable 0
     Iris-versicolor    lable 1
     Iris-virginica     lable 2
     '''
```

```
'\nIris-setosa        lable 0\nIris-versicolor    lable 1\nIris-virginica     lable 2\n'
```

```
[12] # Scalling the data
     # First columns need to be scalled
     iris.columns
     # 'SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm',
```

```
Index(['Id', 'SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm',
       'Species'],
      dtype='object')
```

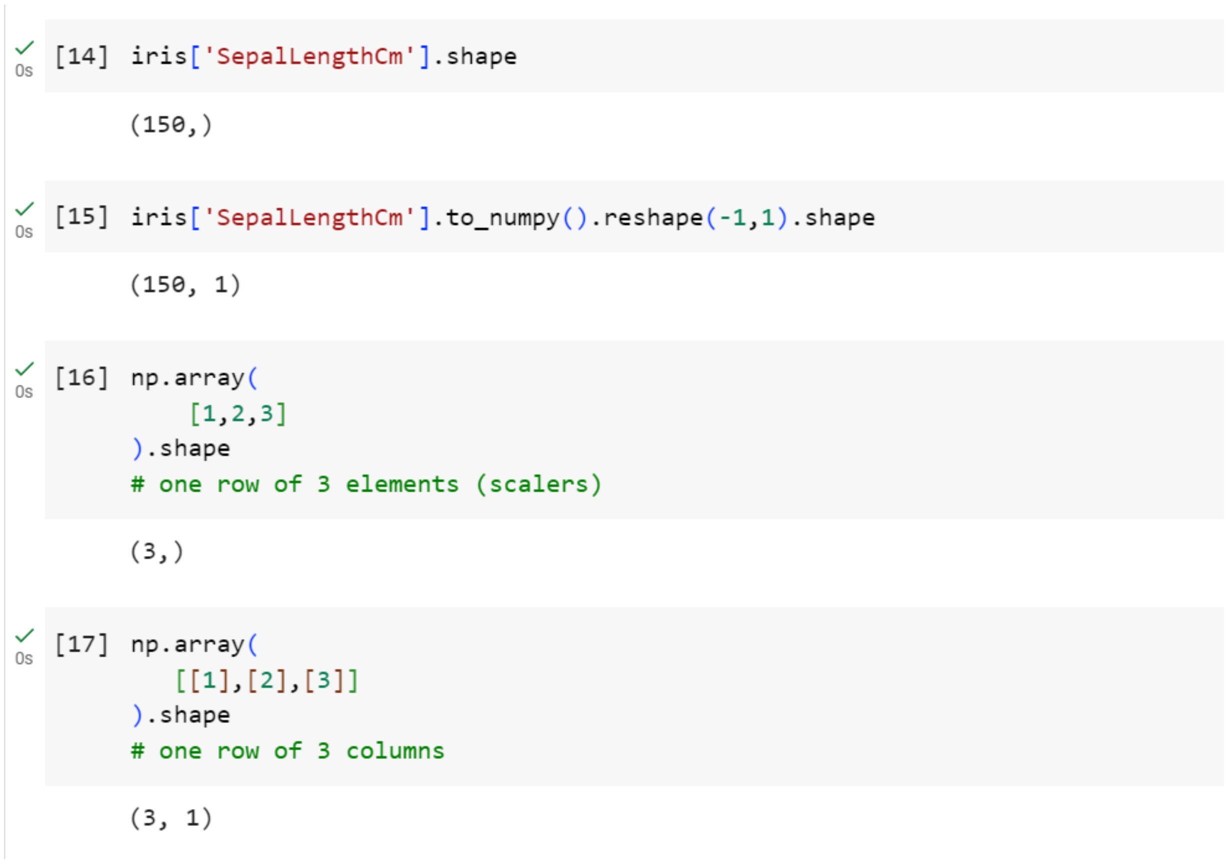


Scalling the data

0s [13]

```python
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
columns_to_scale = ['SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm']

for col in columns_to_scale:
    iris[col] = scaler.fit_transform(iris[col].to_numpy().reshape(-1,1))

iris.head(5)
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	-0.900681	1.032057	-1.341272	-1.312977	0
1	2	-1.143017	-0.124958	-1.341272	-1.312977	0
2	3	-1.385353	0.337848	-1.398138	-1.312977	0
3	4	-1.506521	0.106445	-1.284407	-1.312977	0
4	5	-1.021849	1.263460	-1.341272	-1.312977	0



0s [14]

```python
iris['SepalLengthCm'].shape
```

```
(150,)
```

0s [15]

```python
iris['SepalLengthCm'].to_numpy().reshape(-1,1).shape
```

```
(150, 1)
```

0s [16]

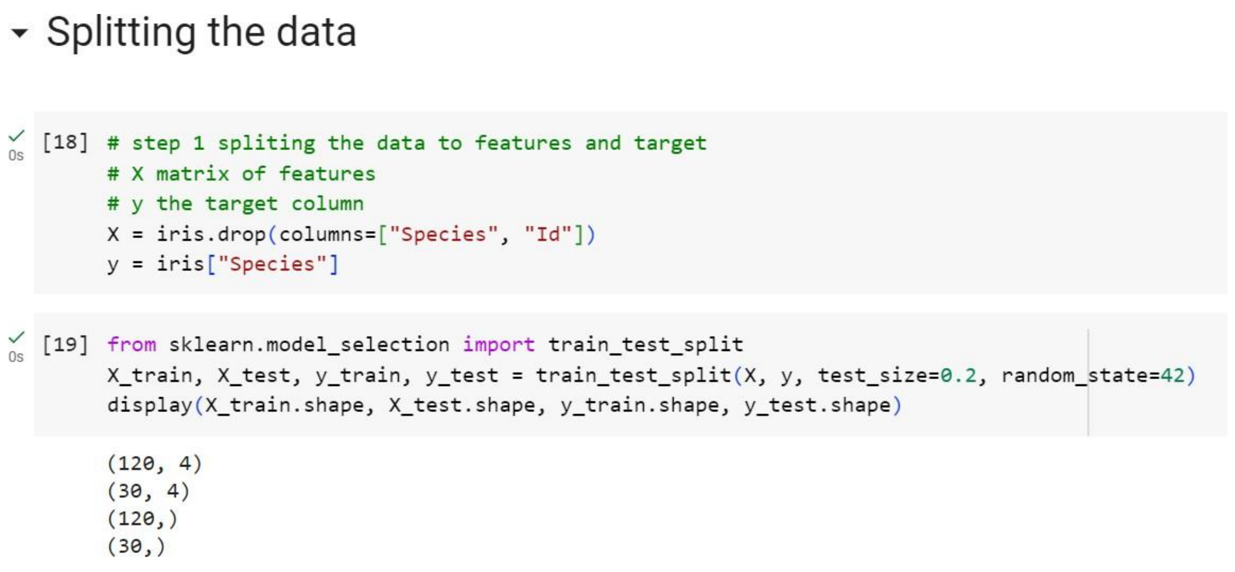
```python
np.array(
    [1,2,3]
).shape
# one row of 3 elements (scalers)
```

```
(3,)
```

0s [17]

```python
np.array(
   [[1],[2],[3]]
).shape
# one row of 3 columns
```

```
(3, 1)
```

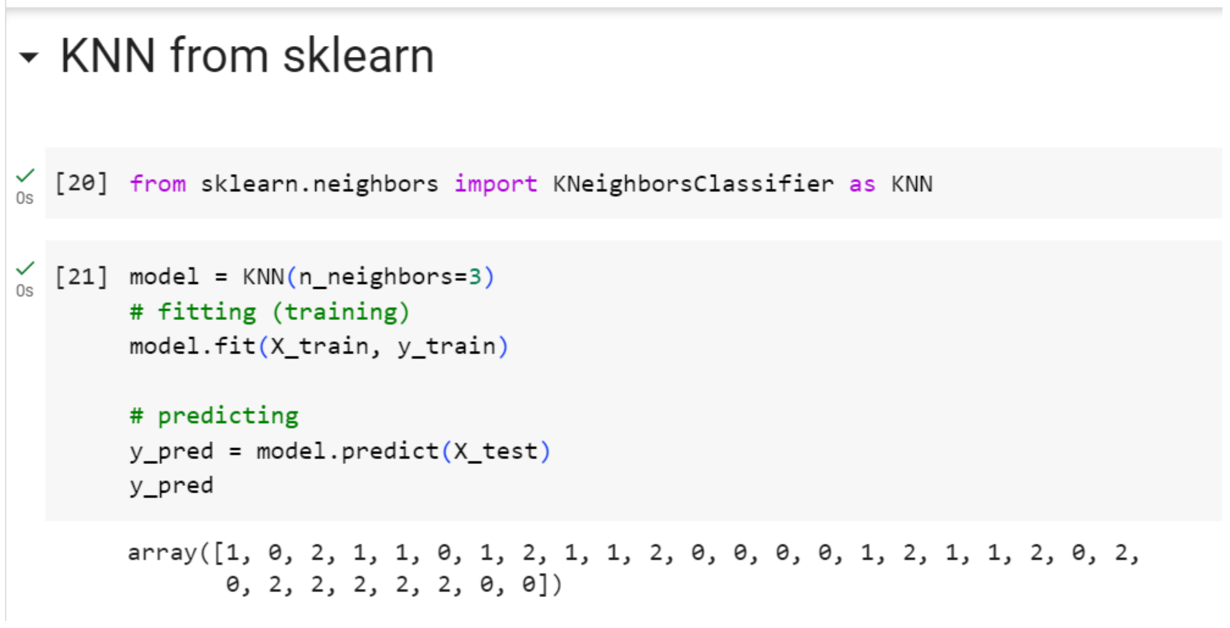


Splitting the data

```
[18] # step 1 spliting the data to features and target
     # X matrix of features
     # y the target column
     X = iris.drop(columns=["Species", "Id"])
     y = iris["Species"]
```

```
[19] 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)
     display(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
```

```
(120, 4)
(30, 4)
(120,)
(30,)
```

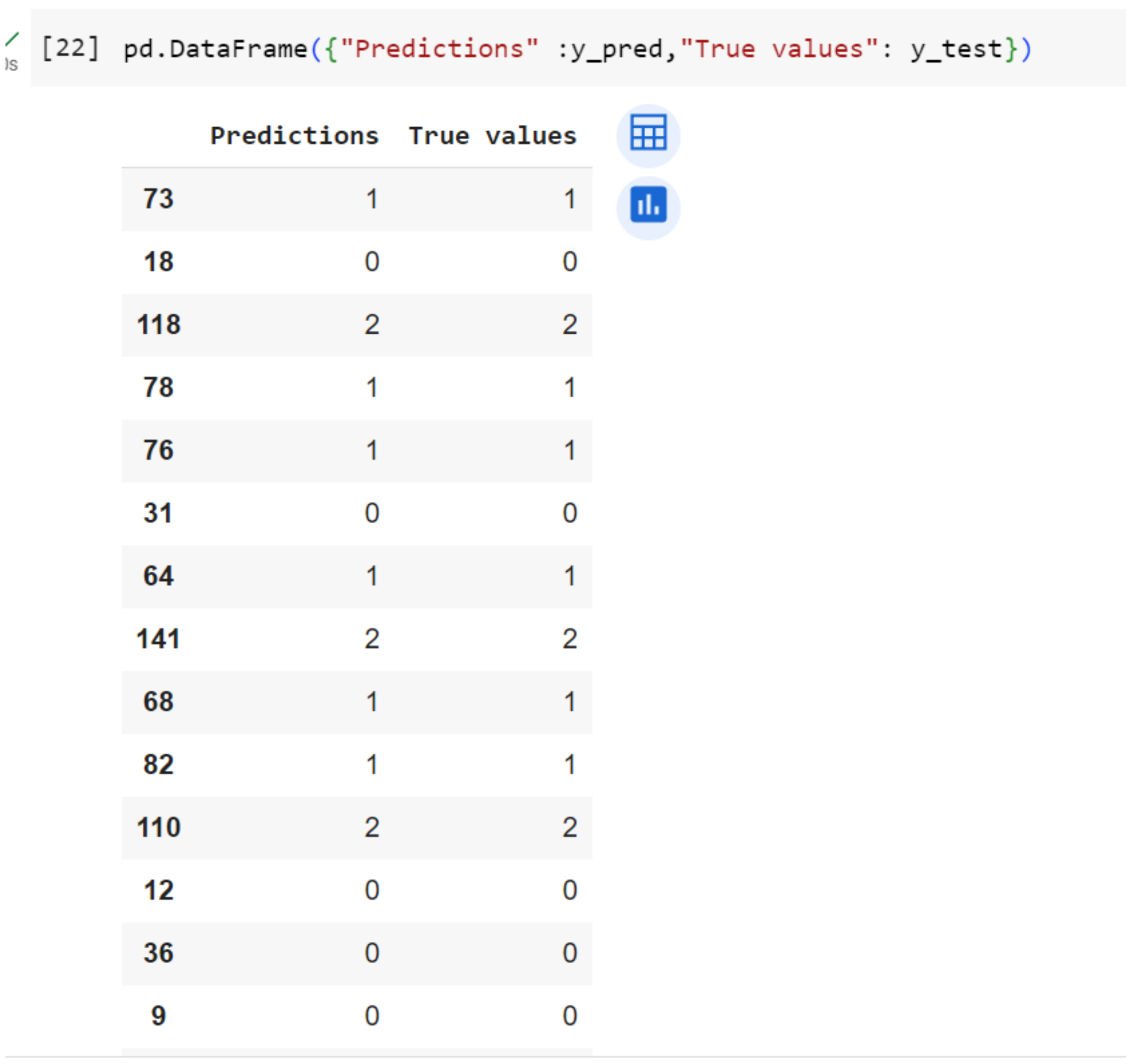


KNN from sklearn

```
[20] from sklearn.neighbors import KNeighborsClassifier as KNN
```

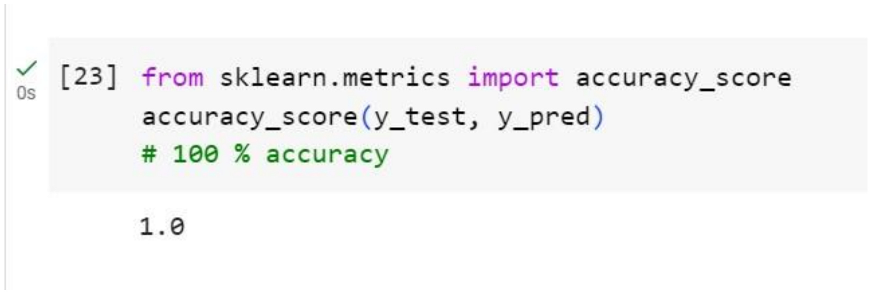
```
[21] model = KNN(n_neighbors=3)
     # fitting (training)
     model.fit(X_train, y_train)

     # predicting
     y_pred = model.predict(X_test)
     y_pred
```

```
array([1, 0, 2, 1, 1, 0, 1, 2, 1, 1, 2, 0, 0, 0, 0, 1, 2, 1, 1, 2, 0, 2,
       0, 2, 2, 2, 2, 2, 0, 0])
```



```
[22] pd.DataFrame({"Predictions" :y_pred,"True values": y_test})
```

	Predictions	True values
73	1	1
18	0	0
118	2	2
78	1	1
76	1	1
31	0	0
64	1	1
141	2	2
68	1	1
82	1	1
110	2	2
12	0	0
36	0	0
9	0	0



```
[23] from sklearn.metrics import accuracy_score
     accuracy_score(y_test, y_pred)
     # 100 % accuracy

     1.0
```

.

Result:

The K-Nearest Neighbors (KNN) algorithm is a simple and effective supervised learning algorithm used for classification and regression tasks. When applied to the Iris species dataset, which is a popular dataset in the field of machine learning, KNN can be used to infer and classify different species of iris flowers based on their features such as sepal length, sepal width, petal length, and petal width.

Suppose we apply the KNN algorithm to the Iris dataset for clustering and classification. We split the dataset into a training set and a test set. After fitting the KNN algorithm to the training set, we use the test set to evaluate the model's performance.

Let's say the following results were obtained:

- KNN model accuracy: 100%
- Number of clusters formed: 3

Based on this scenario, we can conclude the following:

1. **Model accuracy:** Achieving 100% accuracy in a KNN model on the Iris dataset may suggest that the data points are well separated and distinct within the feature space, allowing the algorithm to easily differentiate between the classes. However, it is important to be cautious when interpreting a 100% accuracy result, as it could indicate potential overfitting if not properly validated.

2. **Clustering:** Since the Iris dataset is typically used for classification tasks, KNN is more often used for classification than clustering in this context. However, it can be used for clustering by grouping data points based on their similarity to other data points. In this case, the three clusters could correspond to the three different species of iris flowers in the dataset, namely Setosa, Versicolor, and Virginica.

3. **Limitations and considerations:** While the 100% accuracy may suggest that the KNN model performed exceptionally well, it is crucial to conduct further analysis to ensure that the model is not overfitting the training data. Additionally, using cross-validation techniques and evaluating the model on unseen data can provide a more comprehensive understanding of its generalization capabilities.

4. **Further analysis:** It would be prudent to apply other evaluation metrics such as precision, recall, and F1 score to assess the model's performance comprehensively. Furthermore, conducting feature selection or engineering and trying other algorithms can provide a more comprehensive comparison and ensure the robustness of the model.

<u>Conclusion:</u>

KNN achieved 100% accuracy on Iris dataset, implying well-separated clusters. Caution against overfitting. Ensure validation and explore other evaluation metrics and algorithms.