Course:CSCE 5215 Machine Learning

Professor: Zeenat Tariq

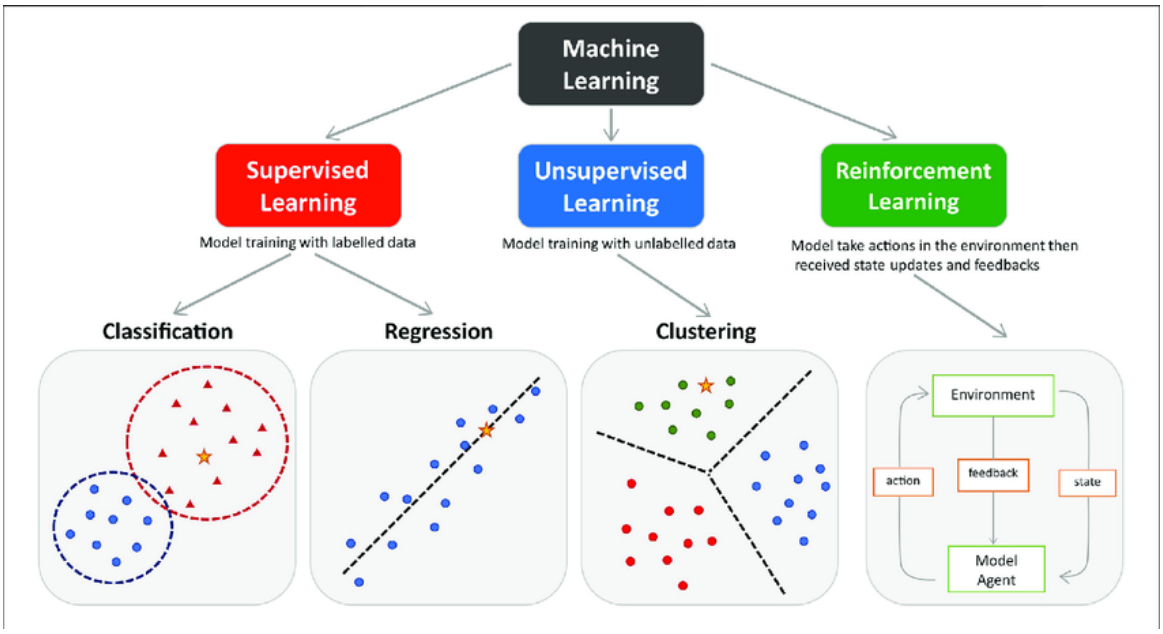
Week2 Day-2

You dont need to go over KNN

The goal is preprocessing and evaluation and learning about sklearn

Lets have a quick review

```
from IPython.display import Image
Image(url="https://www.researchgate.net/publication/354960266/figure/fig1/AS:1075175843983363@1633353305883/The-main-
```



Numpy:

1) A fundamental library within scientific computing.

2) Central package of libraries, including scikit-learn, matplotlib, and pandas.

3) Designed for multi-dimensional arrays, optimizing performance for fast array operations.

```
import numpy as np
```

```
arr_1d = np.array([1,2,3])
print (f"Create an array: {arr_1d}")
print(f"Shape: {arr_1d.shape}")
print(f"Dimention: {arr_1d.ndim}")
print(f"Data type: {arr_1d.dtype}")
print(f"Total number of elements: {arr_1d.size}")
print(f"Size(in byte): {arr_1d.itemsize}")
print(f"Access different elements: {arr_1d[0]}")
```

```
Create an array: [1 2 3]
Shape: (3,)
Dimention: 1
Data type: int64
Total number of elements: 3
Size(in byte): 8
Access different elements: 1
```

1) array/ matrix

2) Mathematic operation

3) Random numbers

Pandas:

```
import pandas as pd
df = pd.read_csv("https://raw.githubusercontent.com/omairaasim/machine_learning/master/project_11_k_nearest_neighbor/
df.head()
```

	Gender	Age	Salary	Purchase Iphone
0	Male	19	19000	0
1	Male	35	20000	0
2	Female	26	43000	0
3	Female	27	57000	0
4	Male	19	76000	0

```
df.iloc[:,:].head()
```

	Gender	Age	Salary	Purchase Iphone
0	Male	19	19000	0
1	Male	35	20000	0
2	Female	26	43000	0
3	Female	27	57000	0
4	Male	19	76000	0

```
df.iloc[:5,:-1]
```

	Gender	Age	Salary
0	Male	19	19000
1	Male	35	20000
2	Female	26	43000
3	Female	27	57000
4	Male	19	76000

First Activity

```
# import pandas as pd
# import numpy as np
# def age_gender_analysis(inp:str) -> None:
#     """
#     Analyze age and gender to identify potential buyers.

#     Parameter
#     ---------
#     inp:
#       The gender for which age analysis will be performed.

#     """
#     data = df.loc[df["Gender"] == inp]
#     min_age = np.min(data["Age"])
#     max_age = np.max(data["Age"])
#     print (f"Young {inp}")
#     print (data.loc[data["Age"]== min_age])
#     print ("-"*5)
#     print (f"Old {inp}")
#     print (data.loc[data["Age"]== max_age])

# # df = pd.read_csv("https://raw.githubusercontent.com/omairaasim/machine_learning/master/project_11_k_nearest_neighb
# df.rename(columns={"Salary":"Income"}, inplace=True)
# stat = df["Gender"].value_counts()
# print (stat)
# print ("-"*5)
# for inp in ["Female", "Male"]:
#     age_gender_analysis(inp)
```

Let us learn about the package -Scikit-Learn

Scikit-learn, also known as sklearn, is a popular open-source machine learning library in Python. It provides a wide range of tools and algorithms for various machine learning tasks, including classification, regression, clustering, dimensionality reduction, model selection, and preprocessing of data.

Sklearn is built on top of other scientific computing libraries in Python, such as NumPy, SciPy, and matplotlib, and provides a consistent and user-friendly interface for working with machine learning algorithms. It aims to provide accessible and efficient implementations of state-of-the-art machine learning techniques and tools.

```
# import the package
import sklearn
```

```
# check the version
sklearn.__version__
```

```
'1.2.2'
```

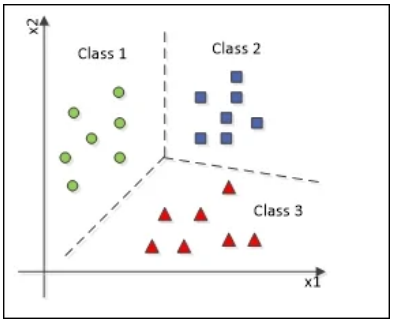
```
# check the sklearn propertes and what it offers
```

```
# check datasets
from sklearn import datasets
dir(datasets)
```

```
['__all__',
 '__builtins__',
 '__cached__',
 '__doc__',
 '__file__',
 '__getattr__',
 '__loader__',
 '__name__',
 '__package__',
 '__path__',
 '__spec__',
 '_arff_parser',
 '_base',
 '_california_housing',
 '_covtype',
 '_kddcup99',
 '_lfw',
 '_olivetti_faces',
 '_openml',
 '_rcv1',
 '_samples_generator',
 '_species_distributions',
 '_svmlight_format_fast',
 '_svmlight_format_io',
 '_twenty_newsgroups',
 'clear_data_home',
 'dump_svmlight_file',
 'fetch_20newsgroups',
 'fetch_20newsgroups_vectorized',
 'fetch_california_housing',
 'fetch_covtype',
 'fetch_kddcup99',
 'fetch_lfw_pairs',
 'fetch_lfw_people',
 'fetch_olivetti_faces',
 'fetch_openml',
 'fetch_rcv1',
 'fetch_species_distributions',
 'get_data_home',
 'load_breast_cancer',
 'load_diabetes',
 'load_digits',
 'load_files',
 'load_iris',
 'load_linnerud',
 'load_sample_image',
 'load_sample_images',
 'load_svmlight_file',
 'load_svmlight_files',
 'load_wine',
 'make_biclusters',
 'make_blobs',
 'make_checkerboard',
 'make_circles',
 'make_classification',
 'make_friedman1',
 'make_friedman2',
```

```
'make friedman3',
```

```
from IPython.display import Image
Image(url="https://static.packt-cdn.com/products/9781785880902/graphics/B04971_06_01.jpg")
```



We will be using the iris dataset which is loaded into our datafame in the link below.The Iris dataset is a popular dataset in machine learning and is often used for classification tasks. It contains measurements of four features (sepal length, sepal width, petal length, and petal width) of three different species of Iris flowers (Setosa, Versicolor, and Virginica).

```
# import only what you need
from sklearn.datasets import load_iris
```

To use the Iris dataset in scikit-learn, you need to import the dataset from the sklearn.datasets module. the load_iris() function is called from the sklearn.datasets module, and the returned dataset is assigned to the variable iris. The iris variable now holds the Iris dataset, which consists of the input features in the iris.data attribute and the corresponding target labels in the iris.target attribute.

```
# Load the Iris dataset
iris = load_iris()
iris
```

```
{'data': array([[5.1, 3.5, 1.4, 0.2],
       [4.9, 3. , 1.4, 0.2],
       [4.7, 3.2, 1.3, 0.2],
       [4.6, 3.1, 1.5, 0.2],
       [5. , 3.6, 1.4, 0.2],
       [5.4, 3.9, 1.7, 0.4],
       [4.6, 3.4, 1.4, 0.3],
       [5. , 3.4, 1.5, 0.2],
       [4.4, 2.9, 1.4, 0.2],
       [4.9, 3.1, 1.5, 0.1],
       [5.4, 3.7, 1.5, 0.2],
       [4.8, 3.4, 1.6, 0.2],
       [4.8, 3. , 1.4, 0.1],
       [4.3, 3. , 1.1, 0.1],
       [5.8, 4. , 1.2, 0.2],
       [5.7, 4.4, 1.5, 0.4],
       [5.4, 3.9, 1.3, 0.4],
       [5.1, 3.5, 1.4, 0.3],
       [5.7, 3.8, 1.7, 0.3],
       [5.1, 3.8, 1.5, 0.3],
       [5.4, 3.4, 1.7, 0.2],
       [5.1, 3.7, 1.5, 0.4],
       [4.6, 3.6, 1. , 0.2],
       [5.1, 3.3, 1.7, 0.5],
       [4.8, 3.4, 1.9, 0.2],
       [5. , 3. , 1.6, 0.2],
       [5. , 3.4, 1.6, 0.4],
       [5.2, 3.5, 1.5, 0.2],
       [5.2, 3.4, 1.4, 0.2],
       [4.7, 3.2, 1.6, 0.2],
       [4.8, 3.1, 1.6, 0.2],
       [5.4, 3.4, 1.5, 0.4],
       [5.2, 4.1, 1.5, 0.1],
       [5.5, 4.2, 1.4, 0.2],
       [4.9, 3.1, 1.5, 0.2],
       [5. , 3.2, 1.2, 0.2],
       [5.5, 3.5, 1.3, 0.2],
       [4.9, 3.6, 1.4, 0.1],
       [4.4, 3. , 1.3, 0.2],
       [5.1, 3.4, 1.5, 0.2],
       [5. , 3.5, 1.3, 0.3],
       [4.5, 2.3, 1.3, 0.3],
       [4.4, 3.2, 1.3, 0.2],
       [5. , 3.5, 1.6, 0.6],
       [5.1, 3.8, 1.9, 0.4],
       [4.8, 3. , 1.4, 0.3],
```

```
            [5.1, 3.8, 1.6, 0.2],
            [4.6, 3.2, 1.4, 0.2],
            [5.3, 3.7, 1.5, 0.2],
            [5. , 3.3, 1.4, 0.2],
            [7. , 3.2, 4.7, 1.4],
            [6.4, 3.2, 4.5, 1.5],
            [6.9, 3.1, 4.9, 1.5],
            [5.5, 2.3, 4. , 1.3],
            [6.5, 2.8, 4.6, 1.5],
            [5.7, 2.8, 4.5, 1.3],
            [6.3, 3.3, 4.7, 1.6],
            [4.9, 2.4, 3.3, 1. ],
```

```
# See features names
iris.feature_names
```

```
['sepal length (cm)',
 'sepal width (cm)',
 'petal length (cm)',
 'petal width (cm)']
```

```
# Extract the features and target variable
X = iris.data
y = iris.target
```

In the Iris dataset, the target variable is the "species" of the iris flowers. It has three classes or labels: "setosa," "versicolor," and "virginica." Each sample in the dataset is associated with one of these three species.

```
y
```

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

2a.Find the mean, minimum, and maximum values of the sepal length feature:

```
import numpy as np

sepal_length = X[:, 0]  # Extract sepal length feature
mean_sepal_length = np.mean(sepal_length)
min_sepal_length = np.min(sepal_length)
max_sepal_length = np.max(sepal_length)

print("Mean sepal length:", mean_sepal_length)
print("Minimum sepal length:", min_sepal_length)
print("Maximum sepal length:", max_sepal_length)
```

```
Mean sepal length: 5.843333333333334
Minimum sepal length: 4.3
Maximum sepal length: 7.9
```

2b.Calculate the covariance matrix of the features:

A covariance matrix is a square matrix that summarizes the covariance between multiple variables. It provides information about the relationships and the extent to which variables change together. In the context of statistics and data analysis, covariance measures how two variables vary or move together.

```
X
```

```
array([[5.1, 3.5, 1.4, 0.2],
       [4.9, 3. , 1.4, 0.2],
       [4.7, 3.2, 1.3, 0.2],
       [4.6, 3.1, 1.5, 0.2],
       [5. , 3.6, 1.4, 0.2],
       [5.4, 3.9, 1.7, 0.4],
       [4.6, 3.4, 1.4, 0.3],
       [5. , 3.4, 1.5, 0.2],
       [4.4, 2.9, 1.4, 0.2],
       [4.9, 3.1, 1.5, 0.1],
       [5.4, 3.7, 1.5, 0.2],
```

```
            [4.8, 3.4, 1.6, 0.2],
            [4.8, 3. , 1.4, 0.1],
            [4.3, 3. , 1.1, 0.1],
            [5.8, 4. , 1.2, 0.2],
            [5.7, 4.4, 1.5, 0.4],
            [5.4, 3.9, 1.3, 0.4],
            [5.1, 3.5, 1.4, 0.3],
            [5.7, 3.8, 1.7, 0.3],
            [5.1, 3.8, 1.5, 0.3],
            [5.4, 3.4, 1.7, 0.2],
            [5.1, 3.7, 1.5, 0.4],
            [4.6, 3.6, 1. , 0.2],
            [5.1, 3.3, 1.7, 0.5],
            [4.8, 3.4, 1.9, 0.2],
            [5. , 3. , 1.6, 0.2],
            [5. , 3.4, 1.6, 0.4],
            [5.2, 3.5, 1.5, 0.2],
            [5.2, 3.4, 1.4, 0.2],
            [4.7, 3.2, 1.6, 0.2],
            [4.8, 3.1, 1.6, 0.2],
            [5.4, 3.4, 1.5, 0.4],
            [5.2, 4.1, 1.5, 0.1],
            [5.5, 4.2, 1.4, 0.2],
            [4.9, 3.1, 1.5, 0.2],
            [5. , 3.2, 1.2, 0.2],
            [5.5, 3.5, 1.3, 0.2],
            [4.9, 3.6, 1.4, 0.1],
            [4.4, 3. , 1.3, 0.2],
            [5.1, 3.4, 1.5, 0.2],
            [5. , 3.5, 1.3, 0.3],
            [4.5, 2.3, 1.3, 0.3],
            [4.4, 3.2, 1.3, 0.2],
            [5. , 3.5, 1.6, 0.6],
            [5.1, 3.8, 1.9, 0.4],
            [4.8, 3. , 1.4, 0.3],
            [5.1, 3.8, 1.6, 0.2],
            [4.6, 3.2, 1.4, 0.2],
            [5.3, 3.7, 1.5, 0.2],
            [5. , 3.3, 1.4, 0.2],
            [7. , 3.2, 4.7, 1.4],
            [6.4, 3.2, 4.5, 1.5],
            [6.9, 3.1, 4.9, 1.5],
            [5.5, 2.3, 4. , 1.3],
            [6.5, 2.8, 4.6, 1.5],
            [5.7, 2.8, 4.5, 1.3],
            [6.3, 3.3, 4.7, 1.6],
            [4.9, 2.4, 3.3, 1. ],
```

```
# positive number: two variables tend to increase or decrease in tandem.
# negative number: as one variable increases, a second variable tends to decrease.
```

```
cov_matrix = np.cov(X.T)

print("Covariance matrix:")
print(cov_matrix)
```

```
Covariance matrix:
[[ 0.68569351 -0.042434    1.27431544  0.51627069]
 [-0.042434    0.18997942 -0.32965638 -0.12163937]
 [ 1.27431544 -0.32965638  3.11627785  1.2956094 ]
 [ 0.51627069 -0.12163937  1.2956094   0.58100626]]
```

```
cov_matrix.shape
```

```
(4, 4)
```

```
# Why do we need to transpose?
# 1) Simplifies interpretation
```

```
cov_matrix = np.cov(X)

print("Covariance matrix:")
print(cov_matrix)
```

```
Covariance matrix:
[[4.75       4.42166667 4.35333333 ... 2.915      2.475      2.6       ]
 [4.42166667 4.14916667 4.055      ... 2.95583333 2.50416667 2.62833333]
 [4.35333333 4.055      3.99       ... 2.68833333 2.28166667 2.39666667]
 ...
 [2.915      2.95583333 2.68833333 ... 4.18916667 3.65083333 3.835     ]
```

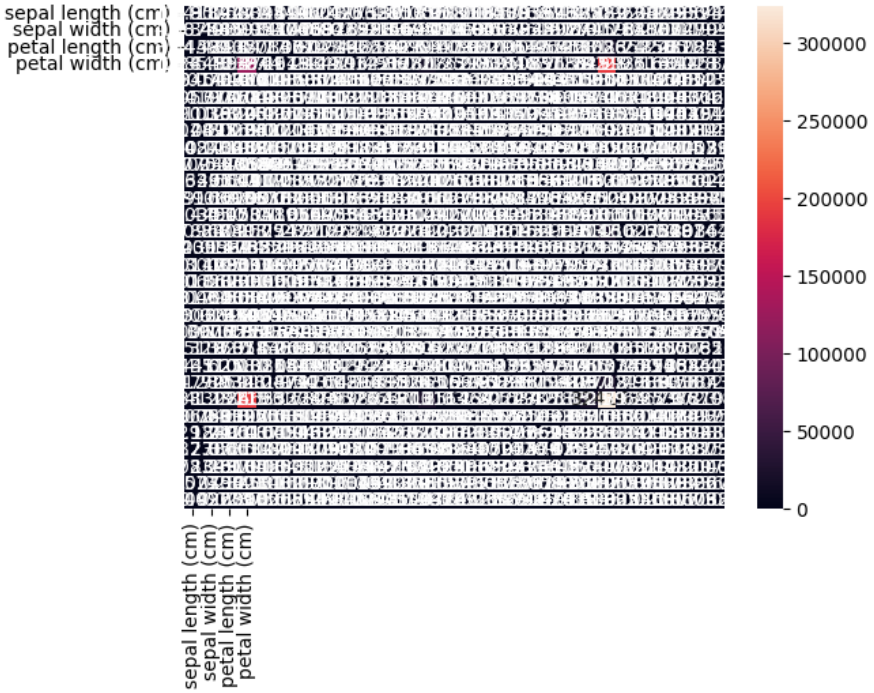
```
       [2.475      2.50416667 2.28166667 ... 3.65083333 3.20916667 3.375     ]
       [2.6        2.62833333 2.39666667 ... 3.835      3.375      3.55      ]]
```

```
# What are these values?
```

```
cov_matrix.shape
```

```
    (150, 150)
```

```
# Seaborn is a Python data visualization library
# Matplotlib is a plotting library
import seaborn as sns
import matplotlib.pyplot as plt
sns.heatmap(cov_matrix, annot=True ,fmt='g', xticklabels=iris.feature_names, yticklabels=iris.feature_names)
plt.show()
```



2c.Calculate the correlation coefficient between sepal length and petal length:

```
sepal_length = X[:, 0]  # Extract sepal length feature
petal_length = X[:, 2]  # Extract petal length feature

correlation_coefficient = np.corrcoef(sepal_length, petal_length)[0, 1]

print("Correlation coefficient between sepal length and petal length:", correlation_coefficient)
```

```
    Correlation coefficient between sepal length and petal length: 0.8717537758865831
```

The np.corrcoef() function in NumPy is used to calculate the correlation coefficients between multiple variables. It computes the correlation coefficient matrix, which measures the linear relationship between pairs of variables.

2d.Count the number of occurrences of each unique value in the target variable (species):

```
unique_species, species_counts = np.unique(y, return_counts=True)

print("Species counts:")
for species, count in zip(unique_species, species_counts):
    print(f"Species {species}: {count} occurrences")
```

```
    Species counts:
    Species 0: 50 occurrences
    Species 1: 50 occurrences
    Species 2: 50 occurrences
```

The np.unique() function in NumPy is used to find the unique elements in an array. It returns a sorted array of unique values without any repetitions. The function takes an array-like object as input and returns a new array containing only the unique elements.

2e.Normalize feature to have zero mean and unit variance:

For some algorithem, it's essential to normalize the features.

```
import seaborn as sns
sns.load_dataset("iris")
```

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	virginica
146	6.3	2.5	5.0	1.9	virginica
147	6.5	3.0	5.2	2.0	virginica
148	6.2	3.4	5.4	2.3	virginica
149	5.9	3.0	5.1	1.8	virginica

150 rows × 5 columns

StandardScaler: Removing the mean and scaling to unit variance.

This operation is performed feature-wise in an independent way.

```
petal_width = X[:, 3]  # Extract petal width feature ????
scale_petal_width = (petal_width - np.mean(petal_width)) / np.std(petal_width)

print("Scale petal width:")
print(scale_petal_width)
```

```
Scale petal width:
[-1.31544430e+00 -1.31544430e+00 -1.31544430e+00 -1.31544430e+00
 -1.31544430e+00 -1.05217993e+00 -1.18381211e+00 -1.31544430e+00
 -1.31544430e+00 -1.44707648e+00 -1.31544430e+00 -1.31544430e+00
 -1.44707648e+00 -1.44707648e+00 -1.31544430e+00 -1.05217993e+00
 -1.05217993e+00 -1.18381211e+00 -1.18381211e+00 -1.18381211e+00
 -1.31544430e+00 -1.05217993e+00 -1.31544430e+00 -9.20547742e-01
 -1.31544430e+00 -1.31544430e+00 -1.05217993e+00 -1.31544430e+00
 -1.31544430e+00 -1.31544430e+00 -1.31544430e+00 -1.05217993e+00
 -1.44707648e+00 -1.31544430e+00 -1.31544430e+00 -1.31544430e+00
 -1.31544430e+00 -1.44707648e+00 -1.31544430e+00 -1.31544430e+00
 -1.18381211e+00 -1.18381211e+00 -1.31544430e+00 -7.88915558e-01
 -1.05217993e+00 -1.18381211e+00 -1.31544430e+00 -1.31544430e+00
 -1.31544430e+00 -1.31544430e+00  2.64141916e-01  3.95774101e-01
  3.95774101e-01  1.32509732e-01  3.95774101e-01  1.32509732e-01
  5.27406285e-01 -2.62386821e-01  1.32509732e-01  2.64141916e-01
 -2.62386821e-01  3.95774101e-01 -2.62386821e-01  2.64141916e-01
  1.32509732e-01  2.64141916e-01  3.95774101e-01 -2.62386821e-01
  3.95774101e-01 -1.30754636e-01  7.90670654e-01  1.32509732e-01
  3.95774101e-01  8.77547895e-04  1.32509732e-01  2.64141916e-01
  2.64141916e-01  6.59038469e-01  3.95774101e-01 -2.62386821e-01
 -1.30754636e-01 -2.62386821e-01  8.77547895e-04  5.27406285e-01
  3.95774101e-01  5.27406285e-01  3.95774101e-01  1.32509732e-01
  1.32509732e-01  1.32509732e-01  8.77547895e-04  2.64141916e-01
  8.77547895e-04 -2.62386821e-01  1.32509732e-01  8.77547895e-04
  1.32509732e-01  1.32509732e-01 -1.30754636e-01  1.32509732e-01
  1.71209594e+00  9.22302838e-01  1.18556721e+00  7.90670654e-01
  1.31719939e+00  1.18556721e+00  6.59038469e-01  7.90670654e-01
  7.90670654e-01  1.71209594e+00  1.05393502e+00  9.22302838e-01
  1.18556721e+00  1.05393502e+00  1.58046376e+00  1.44883158e+00
  7.90670654e-01  1.31719939e+00  1.44883158e+00  3.95774101e-01
  1.44883158e+00  1.05393502e+00  1.05393502e+00  7.90670654e-01
  1.18556721e+00  7.90670654e-01  7.90670654e-01  7.90670654e-01
  1.18556721e+00  5.27406285e-01  9.22302838e-01  1.05393502e+00
```

```
       1.31719939e+00  3.95774101e-01  2.64141916e-01  1.44883158e+00
       1.58046376e+00  7.90670654e-01  7.90670654e-01  1.18556721e+00
       1.58046376e+00  1.44883158e+00  9.22302838e-01  1.44883158e+00
       1.71209594e+00  1.44883158e+00  9.22302838e-01  1.05393502e+00
       1.44883158e+00  7.90670654e-01]
```

```
from sklearn.preprocessing import StandardScaler
# StandardScaler
scalar = StandardScaler()
scalar.fit_transform(X)
```

```
array([[-9.00681170e-01,  1.01900435e+00, -1.34022653e+00,
        -1.31544430e+00],
       [-1.14301691e+00, -1.31979479e-01, -1.34022653e+00,
        -1.31544430e+00],
       [-1.38535265e+00,  3.28414053e-01, -1.39706395e+00,
        -1.31544430e+00],
       [-1.50652052e+00,  9.82172869e-02, -1.28338910e+00,
        -1.31544430e+00],
       [-1.02184904e+00,  1.24920112e+00, -1.34022653e+00,
        -1.31544430e+00],
       [-5.37177559e-01,  1.93979142e+00, -1.16971425e+00,
        -1.05217993e+00],
       [-1.50652052e+00,  7.88807586e-01, -1.34022653e+00,
        -1.18381211e+00],
       [-1.02184904e+00,  7.88807586e-01, -1.28338910e+00,
        -1.31544430e+00],
       [-1.74885626e+00, -3.62176246e-01, -1.34022653e+00,
        -1.31544430e+00],
       [-1.14301691e+00,  9.82172869e-02, -1.28338910e+00,
        -1.44707648e+00],
       [-5.37177559e-01,  1.47939788e+00, -1.28338910e+00,
        -1.31544430e+00],
       [-1.26418478e+00,  7.88807586e-01, -1.22655167e+00,
        -1.31544430e+00],
       [-1.26418478e+00, -1.31979479e-01, -1.34022653e+00,
        -1.44707648e+00],
       [-1.87002413e+00, -1.31979479e-01, -1.51073881e+00,
        -1.44707648e+00],
       [-5.25060772e-02,  2.16998818e+00, -1.45390138e+00,
        -1.31544430e+00],
       [-1.73673948e-01,  3.09077525e+00, -1.28338910e+00,
        -1.05217993e+00],
       [-5.37177559e-01,  1.93979142e+00, -1.39706395e+00,
        -1.05217993e+00],
       [-9.00681170e-01,  1.01900435e+00, -1.34022653e+00,
        -1.18381211e+00],
       [-1.73673948e-01,  1.70959465e+00, -1.16971425e+00,
        -1.18381211e+00],
       [-9.00681170e-01,  1.70959465e+00, -1.28338910e+00,
        -1.18381211e+00],
       [-5.37177559e-01,  7.88807586e-01, -1.16971425e+00,
        -1.31544430e+00],
       [-9.00681170e-01,  1.47939788e+00, -1.28338910e+00,
        -1.05217993e+00],
       [-1.50652052e+00,  1.24920112e+00, -1.56757623e+00,
        -1.31544430e+00],
       [-9.00681170e-01,  5.58610819e-01, -1.16971425e+00,
        -9.20547742e-01],
       [-1.26418478e+00,  7.88807586e-01, -1.05603939e+00,
        -1.31544430e+00],
       [-1.02184904e+00, -1.31979479e-01, -1.22655167e+00,
        -1.31544430e+00],
       [-1.02184904e+00,  7.88807586e-01, -1.22655167e+00,
        -1.05217993e+00],
       [-7.79513300e-01,  1.01900435e+00, -1.28338910e+00,
        -1.31544430e+00],
       [-7.79513300e-01,  7.88807586e-01, -1.34022653e+00,
        -1.31544430e+00],
```

StandardScaler can be influenced by outliers

```
X[0]
```

```
array([5.1, 3.5, 1.4, 0.2])
```

```
# Normalization
from sklearn.preprocessing import Normalizer
```

This normalization technique is often used when the direction of the data points is more important than their absolute values. It's common in various machine learning algorithms, especially those that rely on similarity measures or distances between data points, such as cosine

similarity.

By normalizing samples to unit norm, you ensure that the differences in scale among samples do not dominate the analysis or algorithms, and you focus more on the relationships between directions of vectors rather than their magnitudes.

columns that have large values will dominate the distance measure.

normalization is crucial for the following reasons: 1. Distance-based Metrics: KNN relies on distance-based metrics

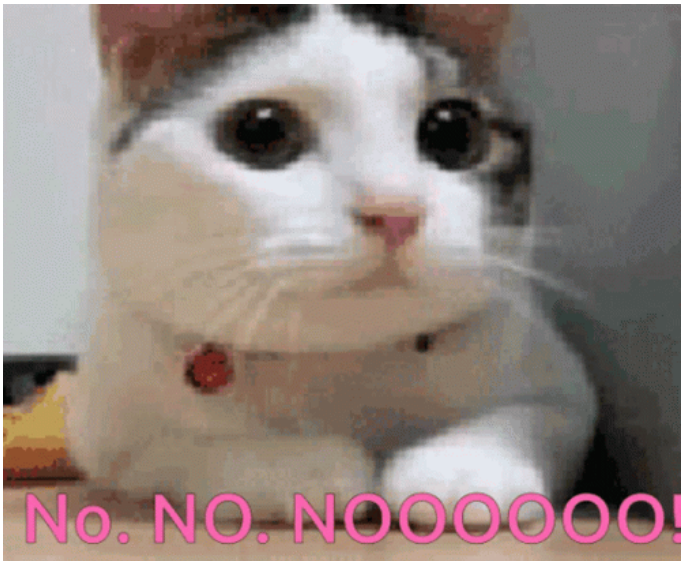
```
# Normalizer
norm = Normalizer()
X = norm.fit_transform(X)
```

```
X
```

```
array([[0.80377277, 0.55160877, 0.22064351, 0.0315205 ],
       [0.82813287, 0.50702013, 0.23660939, 0.03380134],
       [0.80533308, 0.54831188, 0.2227517 , 0.03426949],
       [0.80003025, 0.53915082, 0.26087943, 0.03478392],
       [0.790965  , 0.5694948 , 0.2214702 , 0.0316386 ],
       [0.78417499, 0.5663486 , 0.2468699 , 0.05808704],
       [0.78010936, 0.57660257, 0.23742459, 0.0508767 ],
       [0.80218492, 0.54548574, 0.24065548, 0.0320874 ],
       [0.80642366, 0.5315065 , 0.25658935, 0.03665562],
       [0.81803119, 0.51752994, 0.25041771, 0.01669451],
       [0.80373519, 0.55070744, 0.22325977, 0.02976797],
       [0.786991  , 0.55745196, 0.26233033, 0.03279129],
       [0.82307218, 0.51442011, 0.24006272, 0.01714734],
       [0.8025126 , 0.55989251, 0.20529392, 0.01866308],
       [0.81120865, 0.55945424, 0.16783627, 0.02797271],
       [0.77381111, 0.59732787, 0.2036345 , 0.05430253],
       [0.79428944, 0.57365349, 0.19121783, 0.05883625],
       [0.80327412, 0.55126656, 0.22050662, 0.04725142],
       [0.8068282 , 0.53788547, 0.24063297, 0.04246464],
       [0.77964883, 0.58091482, 0.22930848, 0.0458617 ],
       [0.8173379 , 0.51462016, 0.25731008, 0.03027177],
       [0.78591858, 0.57017622, 0.23115252, 0.06164067],
       [0.77577075, 0.60712493, 0.16864581, 0.03372916],
       [0.80597792, 0.52151512, 0.26865931, 0.07901744],
       [0.776114  , 0.54974742, 0.30721179, 0.03233808],
       [0.82647451, 0.4958847 , 0.26447184, 0.03305898],
       [0.79778206, 0.5424918 , 0.25529026, 0.06382256],
       [0.80641965, 0.54278246, 0.23262105, 0.03101614],
       [0.81609427, 0.5336001 , 0.21971769, 0.03138824],
       [0.79524064, 0.54144043, 0.27072022, 0.03384003],
       [0.80846584, 0.52213419, 0.26948861, 0.03368608],
       [0.82225028, 0.51771314, 0.22840286, 0.06090743],
       [0.76578311, 0.60379053, 0.22089897, 0.0147266 ],
       [0.77867447, 0.59462414, 0.19820805, 0.02831544],
       [0.81768942, 0.51731371, 0.25031309, 0.03337508],
       [0.82512295, 0.52807869, 0.19802951, 0.03300492],
       [0.82699754, 0.52627116, 0.19547215, 0.03007264],
       [0.78523221, 0.5769053 , 0.22435206, 0.01602515],
       [0.80212413, 0.54690282, 0.23699122, 0.03646019],
       [0.80779568, 0.53853046, 0.23758697, 0.03167826],
       [0.80033301, 0.56023311, 0.20808658, 0.04801998],
       [0.86093857, 0.44003527, 0.24871559, 0.0573959 ],
       [0.78609038, 0.57170209, 0.23225397, 0.03573138],
       [0.78889479, 0.55222635, 0.25244633, 0.09466737],
       [0.76693897, 0.57144472, 0.28572236, 0.06015208],
       [0.82210585, 0.51381615, 0.23978087, 0.05138162],
       [0.77729093, 0.57915795, 0.24385598, 0.030482  ],
       [0.79594782, 0.55370283, 0.24224499, 0.03460643],
       [0.79837025, 0.55735281, 0.22595384, 0.03012718],
       [0.81228363, 0.5361072 , 0.22743942, 0.03249135],
       [0.76701103, 0.35063361, 0.51499312, 0.15340221],
       [0.74549757, 0.37274878, 0.52417798, 0.17472599],
       [0.75519285, 0.33928954, 0.53629637, 0.16417236],
       [0.75384916, 0.31524601, 0.54825394, 0.17818253],
       [0.7581754 , 0.32659863, 0.5365549 , 0.17496355],
       [0.72232962, 0.35482858, 0.57026022, 0.16474184],
       [0.72634846, 0.38046824, 0.54187901, 0.18446945],
       [0.75916547, 0.37183615, 0.51127471, 0.15493173],
```

Do we need to standardize/ Normalize variables before doing correlation analysis?

```
from IPython.display import Image
Image(url='https://media.tenor.com/P4nq8scOcbYAAAAC/kitty-no.gif')
```



The correlation coefficient is independent of change of origin and scale

THIS IS JUST AN INTRODUCTION, YOU CAN SKIP IT

Popular algorithms for multi-class classification:

k-Nearest Neighbors

Decision Trees

Naive Bayes

Random Forest

Gradient Boosting

K-Nearest Neighbour (KNN)

Simple supervised algorithm used for the regression but especially more for the classification.

KNN is a lazy learner who does not learn a discriminative function from the training dataset. Instead, it stores the dataset, and at the classification time, it performs actions on the dataset.

The core idea behind KNN is to find a similarity between the new data and the available data.

A data point is classified based on the plurality vote of its neighbors, namely the most common label among the nearest neighbors.

Symbol K in KNN refers to the number of nearest neighbors included in the classification process.

A low value causes skewed classification, while a large value raises difficulties in searching the nearest neighbors for samples and resource issues.

It's popular in many fields, including:

1) Computer Vision

2) Content Recommendation

KNN can be used for both classification and regression predictive problems. However, it is more widely used in classification problems in the industry.

How Does the KNN Algorithm Work?

distances and similarities

```
from IPython.display import Image
Image(url="https://www.gstatic.com/education/formulas2/553212783/en/euclidean_distance.svg")
```

$\sqrt{n}$

```
import numpy as np

def e_dist(point1, point2):
  sum_sq = np.sqrt(np.sum(np.square(point1 - point2)))
  return sum_sq

point1 = np.array((1, 1, 1))
point2 = np.array((1, 1, 1))
e_dist(point1, point2)
```

```
0.0
```

```
1 / (1 + e_dist(point1, point2))
```

```
1.0
```

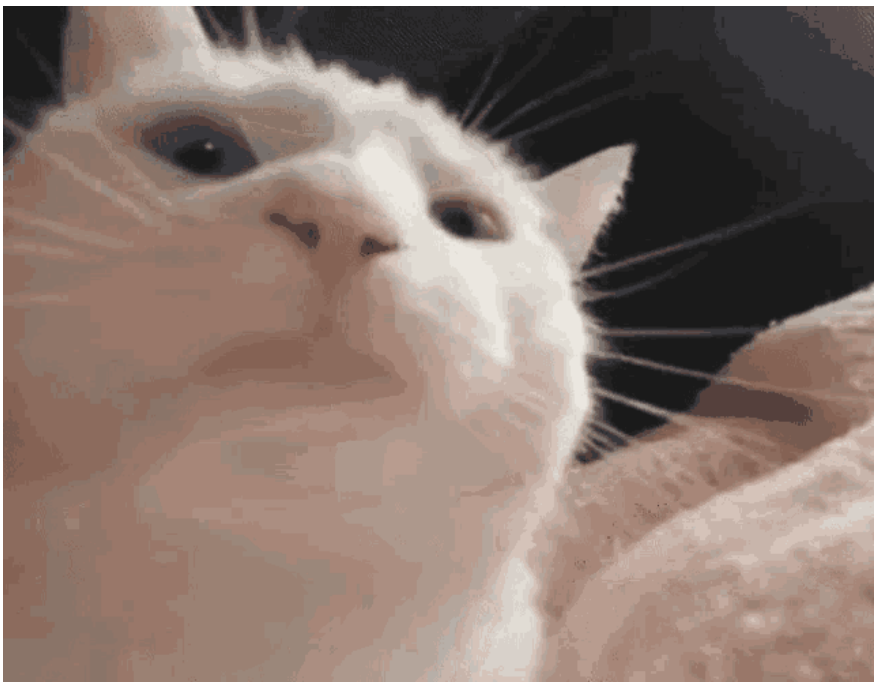
```
from sklearn.metrics.pairwise import euclidean_distances
euclidean_distances(X)
```

```
array([[0.        , 0.05330734, 0.00503045, ..., 0.46521736, 0.48185206,
        0.47601942],
       [0.05330734, 0.        , 0.04916396, ..., 0.43783455, 0.46086858,
        0.45194148],
       [0.00503045, 0.04916396, 0.        , ..., 0.46124034, 0.47818608,
        0.47231469],
       ...,
       [0.46521736, 0.43783455, 0.46124034, ..., 0.        , 0.06626116,
        0.04181293],
       [0.48185206, 0.46086858, 0.47818608, ..., 0.06626116, 0.        ,
        0.04755527],
       [0.47601942, 0.45194148, 0.47231469, ..., 0.04181293, 0.04755527,
        0.        ]])
```

Overfitting: the model gives accurate predictions for training but not for testing.

Underfitting:the model has not learned the patterns in the training data well

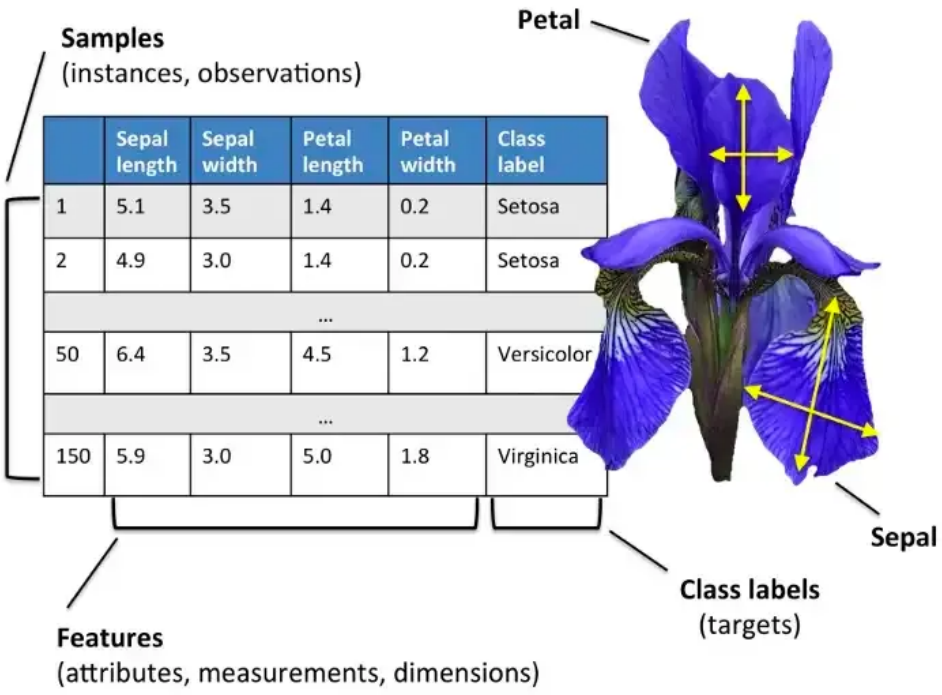
```
# Lets train a simple model
from IPython.display import Image
Image(url='https://media.tenor.com/e-LsbnNHQ5cAAAAd/catjam-cat-dancing.gif')
```



```
# Create train and test dataset
from sklearn.model_selection import train_test_split
```

Lets make it clear

```
from IPython.display import Image
Image(url="https://miro.medium.com/v2/resize:fit:1400/format:webp/1*H2UmG5L1I5bzFCW006N5Ag.png")
```



	Sepal length	Sepal width	Petal length	Petal width	Class label
1	5.1	3.5	1.4	0.2	Setosa
2	4.9	3.0	1.4	0.2	Setosa
			...		
50	6.4	3.5	4.5	1.2	Versicolor
			...		
150	5.9	3.0	5.0	1.8	Virginica

```
t_size = 0.30

# random_state: controls the shuffling proces
x_train, x_test, y_train, y_test = train_test_split(
X, y, test_size=t_size, random_state=0)

# check the shapes
x_train.shape, x_test.shape, y_train.shape, y_test.shape
```

```
((105, 4), (45, 4), (105,), (45,))
```

```
# See if datasets have same population
np.unique(y_train, return_counts=True)
```

```
(array([0, 1, 2]), array([34, 32, 39]))
```

```
x_train, x_test, y_train, y_test = train_test_split(
X, y, test_size=t_size, random_state=0, stratify=y)

np.unique(y_train, return_counts=True)
```

```
(array([0, 1, 2]), array([35, 35, 35]))
```

```
x_train.shape, x_test.shape, y_train.shape, y_test.shape
```

```
((105, 4), (45, 4), (105,), (45,))
```

```
np.unique(y_test, return_counts=True)
```

```
(array([0, 1, 2]), array([15, 15, 15]))
```

```
# this is just the introdution
from sklearn.neighbors import KNeighborsClassifier
knn_classifier = KNeighborsClassifier(n_neighbors=5)
knn_classifier.fit(x_train, y_train)
knn_classifier.score(x_test, y_test) # to find score
pred_test_knn = knn_classifier.predict(x_test)
pred_test_knn
```

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

```
x_test[:,0]
```

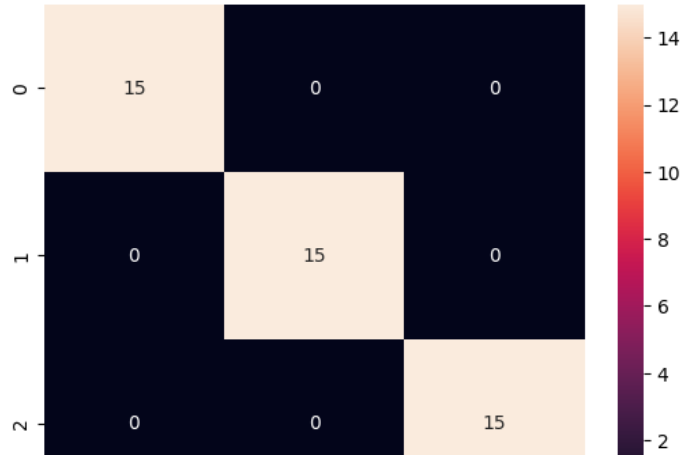
```
array([0.67017484, 0.69052512, 0.80779568, 0.76693897, 0.76701103,
       0.80597792, 0.72337118, 0.69025916, 0.77729093, 0.72232962,
       0.78591858, 0.71653899, 0.79594782, 0.73122464, 0.75676497,
       0.69385414, 0.76444238, 0.76945444, 0.74549757, 0.80377277,
       0.7581754 , 0.69276796, 0.776114  , 0.77242925, 0.73089855,
       0.71491405, 0.6925518 , 0.65387747, 0.74714194, 0.71562645,
       0.7431482 , 0.77964883, 0.78010936, 0.72992443, 0.75457341,
       0.68307923, 0.71524936, 0.78523221, 0.81803119, 0.73923462,
       0.82699754, 0.69198788, 0.80003025, 0.82307218, 0.72415258])
```

```
x_test
```

```
array([[0.67017484, 0.36168166, 0.59571097, 0.2553047 ],
       [0.69052512, 0.32145135, 0.60718588, 0.22620651],
       [0.80779568, 0.53853046, 0.23758697, 0.03167826],
       [0.76693897, 0.57144472, 0.28572236, 0.06015208],
       [0.76701103, 0.35063361, 0.51499312, 0.15340221],
       [0.80597792, 0.52151512, 0.26865931, 0.07901744],
       [0.72337118, 0.34195729, 0.57869695, 0.15782644],
       [0.69025916, 0.35097923, 0.5966647 , 0.21058754],
       [0.77729093, 0.57915795, 0.24385598, 0.030482  ],
       [0.72232962, 0.35482858, 0.57026022, 0.16474184],
       [0.78591858, 0.57017622, 0.23115252, 0.06164067],
       [0.71653899, 0.3307103 , 0.57323119, 0.22047353],
       [0.79594782, 0.55370283, 0.24224499, 0.03460643],
       [0.73122464, 0.31338199, 0.56873028, 0.20892133],
       [0.75676497, 0.35228714, 0.53495455, 0.13047672],
       [0.69385414, 0.29574111, 0.63698085, 0.15924521],
       [0.76444238, 0.27125375, 0.55483721, 0.18494574],
       [0.76945444, 0.35601624, 0.50531337, 0.16078153],
       [0.74549757, 0.37274878, 0.52417798, 0.17472599],
       [0.80377277, 0.55160877, 0.22064351, 0.0315205 ],
       [0.7581754 , 0.32659863, 0.5365549 , 0.17496355],
       [0.69276796, 0.31889319, 0.61579374, 0.1979337 ],
       [0.776114  , 0.54974742, 0.30721179, 0.03233808],
       [0.77242925, 0.33706004, 0.51963422, 0.14044168],
       [0.73089855, 0.30454106, 0.58877939, 0.1624219 ],
       [0.71491405, 0.30207636, 0.59408351, 0.21145345],
       [0.6925518 , 0.30375079, 0.60750157, 0.24300063],
       [0.65387747, 0.34250725, 0.62274045, 0.25947519],
       [0.74714194, 0.33960997, 0.54337595, 0.17659719],
       [0.71562645, 0.3523084 , 0.56149152, 0.22019275],
       [0.7431482 , 0.36505526, 0.5345452 , 0.16948994],
       [0.77964883, 0.58091482, 0.22930848, 0.0458617 ],
       [0.78010936, 0.57660257, 0.23742459, 0.0508767 ],
       [0.72992443, 0.39103094, 0.53440896, 0.16944674],
       [0.75457341, 0.34913098, 0.52932761, 0.16893434],
       [0.68307923, 0.34153961, 0.59769433, 0.24395687],
       [0.71524936, 0.40530797, 0.53643702, 0.19073316],
       [0.78523221, 0.5769053 , 0.22435206, 0.01602515],
       [0.81803119, 0.51752994, 0.25041771, 0.01669451],
       [0.73923462, 0.37588201, 0.52623481, 0.187941  ],
       [0.82699754, 0.52627116, 0.19547215, 0.03007264],
       [0.69198788, 0.34599394, 0.58626751, 0.24027357],
       [0.80003025, 0.53915082, 0.26087943, 0.03478392],
       [0.82307218, 0.51442011, 0.24006272, 0.01714734],
       [0.72415258, 0.32534391, 0.56672811, 0.22039426]])
```

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test,pred_test_knn)
sns.heatmap(cm,annot=True)
```

```
<Axes: >
```




```
pred_test_knn = knn_classifier.predict_proba(x_test)
pred_test_knn
```

```
array([[0. , 0. , 1. ],
       [0. , 0. , 1. ],
       [1. , 0. , 0. ],
       [1. , 0. , 0. ],
       [0. , 1. , 0. ],
       [1. , 0. , 0. ],
       [0. , 0.6, 0.4],
       [0. , 0. , 1. ],
       [1. , 0. , 0. ],
       [0. , 0.8, 0.2],
       [1. , 0. , 0. ],
       [0. , 0. , 1. ],
       [1. , 0. , 0. ],
       [0. , 0. , 1. ],
       [0. , 1. , 0. ],
       [0. , 0.2, 0.8],
       [0. , 1. , 0. ],
       [0. , 1. , 0. ],
       [0. , 1. , 0. ],
       [1. , 0. , 0. ],
       [0. , 1. , 0. ],
       [0. , 0.2, 0.8],
       [1. , 0. , 0. ],
       [0. , 1. , 0. ],
       [0. , 0.4, 0.6],
       [0. , 0. , 1. ],
       [0. , 0. , 1. ],
       [0. , 0. , 1. ],
       [0. , 1. , 0. ],
       [0. , 0.2, 0.8],
       [0. , 1. , 0. ],
       [1. , 0. , 0. ],
       [1. , 0. , 0. ],
       [0. , 1. , 0. ],
       [0. , 1. , 0. ],
       [0. , 0. , 1. ],
       [0. , 1. , 0. ],
       [1. , 0. , 0. ],
       [1. , 0. , 0. ],
       [0. , 1. , 0. ],
       [1. , 0. , 0. ],
       [0. , 0. , 1. ],
       [1. , 0. , 0. ],
       [1. , 0. , 0. ],
       [0. , 0. , 1. ]])
```

Cross validation (CV)

```
from IPython.display import Image
Image(url="https://www.section.io/engineering-education/how-to-implement-k-fold-cross-validation/5-fold-cv.jpeg")
```

The main purpose of cross validation is to prevent overfitting

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import GridSearchCV


def KNN():
    knn_grid = {
        "n_neighbors": list(range(1, 10, 1)),
        "weights": ["distance"],
        "metric": ["euclidean"],
    }

    knn = KNeighborsClassifier()
    # grid = GridSearchCV(knn, param_grid, cv=10, scoring='accuracy', return_train_score=False,verbose=1)

    return GridSearchCV(
        estimator = knn,
        param_grid = knn_grid,
        cv=10,
        n_jobs=1,
        scoring="accuracy",
    )
model = KNN()
model.fit(x_train, y_train)
```

```
GridSearchCV
estimator: KNeighborsClassifier
KNeighborsClassifier
```

```
model.best_estimator_
```

```
KNeighborsClassifier
KNeighborsClassifier(metric='euclidean', n_neighbors=4, weights='distance')
```

```
model.best_score_
```

```
0.9809090909090911
```

```
model.best_params_
```

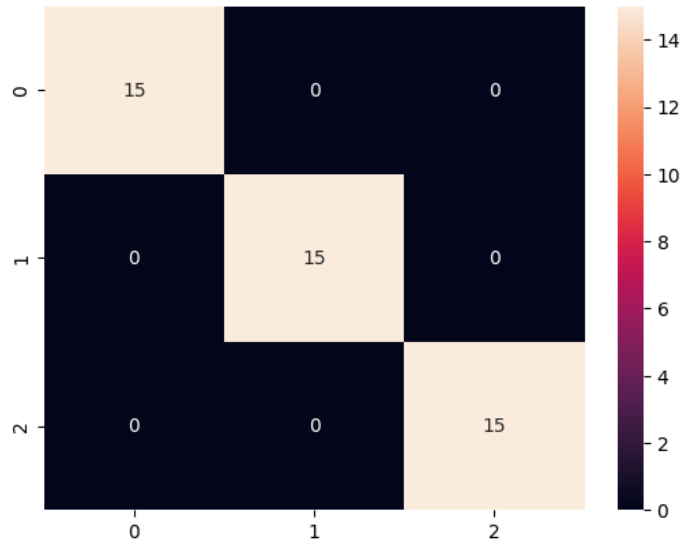
```
{'metric': 'euclidean', 'n_neighbors': 4, 'weights': 'distance'}
```

```
pred_test_knn = model.best_estimator_.predict(x_test)
pred_test_knn
```

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

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test,pred_test_knn)
sns.heatmap(cm,annot=True)
```

```
<Axes: >
```




```
# finding the whole report
from sklearn.metrics import classification_report
print(classification_report(y_test, pred_test_knn))
```

```
                  precision    recall  f1-score   support

               0       1.00      1.00      1.00        15
               1       1.00      1.00      1.00        15
               2       1.00      1.00      1.00        15

        accuracy                           1.00        45
       macro avg       1.00      1.00      1.00        45
    weighted avg       1.00      1.00      1.00        45
```

```
x_test.shape
```

```
(45, 4)
```

```
x_test[0].shape
```

```
(4,)
```

```
id = 0
model.best_estimator_.predict([x_test[id]]), y_test[id]
```

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

Advantages

1) Easy to implement

2) Few hyperparameters

Disadvantages

1) Does not scale well: Since KNN is a lazy algorithm

2) Prone to overfitting

```
# Practice
# 1) Select breast_cancer datasets in sklearn
# 2) create features and labels
# 3) divide the dataset in this way: 75% for training and 25% for testing. make sure the number of labels are identica
```

```
# 4) train simple KNN
# 5) print test accuracy
# 6) plot confusion matrix
# 7) print classification report


# 1
from sklearn.datasets import load_breast_cancer
lin = load_breast_cancer()


# 2
X = lin.data
y = lin.target


cov_matrix = np.cov(X.T)

print("Covariance matrix:")
print(cov_matrix)
```

```
Covariance matrix:
[[ 1.24189201e+01  4.90758156e+00  8.54471417e+01  1.22448341e+03
   8.45445983e-03  9.41970568e-02  1.90127582e-01  1.12475116e-01
   1.42731729e-02 -7.75370629e-03  6.63650325e-01 -1.89188600e-01
   4.80354973e+00  1.17968162e+02 -2.35533633e-03  1.30005142e-02
   2.06588280e-02  8.17956306e-03 -3.03898170e-03 -3.97624858e-04
   1.65137495e+01  6.43310002e+00  1.14288570e+02  1.88822722e+03
   9.62462515e-03  2.29249204e-01  3.87386440e-01  1.72392739e-01
   3.57457580e-02  4.49735060e-04]
 [ 4.90758156e+00  1.84989087e+01  3.44397592e+01  4.85993787e+02
  -1.41477877e-03  5.37668058e-02  1.03692344e-01  4.89769268e-02
   8.41887566e-03 -2.32115824e-03  3.29037393e-01  9.16695072e-01
   2.44944877e+00  5.08408652e+01  8.54099045e-05  1.47865987e-02
   1.86039300e-02  4.34837964e-03  3.24506954e-04  6.19772637e-04
   7.32926700e+00  2.41101485e+01  5.17459332e+01  8.41283832e+02
   7.61106975e-03  1.88010026e-01  2.70110082e-01  8.34908538e-02
   2.79419949e-02  9.26012881e-03]
 [ 8.54471417e+01  3.44397592e+01  5.90440480e+02  8.43577235e+03
   7.08360652e-02  7.14714125e-01  1.38723362e+00  8.02360375e-01
   1.21921583e-01 -4.48588794e-02  4.66140102e+00 -1.16298834e+00
   3.40530278e+01  8.23492755e+02 -1.47881764e-02  1.09111241e-01
   1.67296235e-01  6.10547010e-02 -1.63964262e-02 -3.55136526e-04
   1.13858063e+02  4.52581133e+01  7.92328208e+02  1.30261484e+04
   8.35255275e-02  1.74247791e+00  2.85850554e+00  1.23184809e+00
   2.84299746e-01  2.23905222e-02]
 [ 1.22448341e+03  4.85993787e+02  8.43577235e+03  1.23843554e+05
   8.76178126e-01  9.26493079e+00  1.92449238e+01  1.12419582e+01
   1.45959589e+00 -7.03426366e-01  7.14909447e+01 -1.28671677e+01
   5.17009995e+02  1.28085176e+04 -1.76220977e-01  1.33972528e+00
   2.20595229e+00  8.08460183e-01 -2.10896413e-01 -1.85185365e-02
   1.63752134e+03  6.21824934e+02  1.13417898e+04  1.92192558e+05
   9.92513546e-01  2.16166024e+01  3.76344154e+01  1.67017894e+01
   3.12580933e+00  2.37562255e-02]
 [ 8.45445983e-03 -1.41477877e-03  7.08360652e-02  8.76178126e-01
   1.97799700e-04  4.89573915e-04  5.85242774e-04  3.02167060e-04
   2.15054513e-04  5.80685868e-05  1.17577046e-03  5.30728260e-04
   8.41955814e-03  1.57742108e-01  1.40354828e-05  8.03299928e-05
   1.05454056e-04  3.30349152e-05  2.33418901e-05  1.05543329e-05
   1.44869117e-02  3.11810959e-03  1.12879473e-01  1.65529930e+00
   2.58604266e-04  1.04547843e-03  1.27612364e-03  4.65056762e-04
   3.43093408e-04  1.26834336e-04]
 [ 9.41970568e-02  5.37668058e-02  7.14714125e-01  9.26493079e+00
   4.89573915e-04  2.78918740e-03  3.71813492e-03  1.70323279e-03
   8.72518055e-04  2.10813102e-04  7.28582209e-03  1.34613491e-03
   5.86119478e-02  1.09470847e+00  2.14545745e-05  6.98668545e-04
   9.09523186e-04  2.09293726e-04  1.00400852e-04  7.08958339e-05
   1.36642905e-01  8.05441202e-02  1.04741276e+00  1.53234364e+01
   6.81954177e-04  7.19433104e-03  8.99373999e-03  2.83126752e-03
   1.66709974e-03  6.55670173e-04]
 [ 1.90127582e-01  1.03692344e-01  1.38723362e+00  1.92449238e+01
   5.85242774e-04  3.71813492e-03  6.35524790e-03  2.85018988e-03
   1.09418750e-03  1.89558774e-04  1.39701630e-02  3.35187749e-03
   1.06443014e-01  2.23911946e+00  2.35922336e-05  9.56914602e-04
   1.66349126e-03  3.36091422e-04  1.17306921e-04  9.47774191e-05
   2.65180994e-01  1.46934203e-01  1.95434992e+00  3.06824051e+01
   8.16944485e-04  9.46943910e-03  1.47039422e-02  4.51347883e-03
   2.01950274e-03  7.41417054e-04]
 [ 1.12475116e-01  4.89769268e-02  8.02360375e-01  1.12419582e+01
```

```
lin.feature_names
```

```
array(['mean radius', 'mean texture', 'mean perimeter', 'mean area',
       'mean smoothness', 'mean compactness', 'mean concavity',
       'mean concave points', 'mean symmetry', 'mean fractal dimension',
```

```
            'radius error', 'texture error', 'perimeter error', 'area error',
            'smoothness error', 'compactness error', 'concavity error',
            'concave points error', 'symmetry error',
            'fractal dimension error', 'worst radius', 'worst texture',
            'worst perimeter', 'worst area', 'worst smoothness',
            'worst compactness', 'worst concavity', 'worst concave points',
            'worst symmetry', 'worst fractal dimension'], dtype='<U23')
```

```
import numpy as np

radius_error = X[:, 0]
mean_radius_error = np.mean(radius_error)
min_radius_error = np.min(radius_error)
max_radius_error = np.max(radius_error)

print("Mean sepal length:", mean_radius_error)
print("Minimum sepal length:", min_radius_error)
print("Maximum sepal length:", max_radius_error)
```

```
Mean sepal length: 14.127291739894552
Minimum sepal length: 6.981
Maximum sepal length: 28.11
```

```
from sklearn.preprocessing import StandardScaler
# StandardScaler
scalar = StandardScaler()
scalar.fit_transform(X)
```

```
array([[ 1.09706398, -2.07333501,  1.26993369, ...,  2.29607613,
         2.75062224,  1.93701461],
       [ 1.82982061, -0.35363241,  1.68595471, ...,  1.0870843 ,
        -0.24388967,  0.28118999],
       [ 1.57988811,  0.45618695,  1.56650313, ...,  1.95500035,
         1.152255  ,  0.20139121],
       ...,
       [ 0.70228425,  2.0455738 ,  0.67267578, ...,  0.41406869,
        -1.10454895, -0.31840916],
       [ 1.83834103,  2.33645719,  1.98252415, ...,  2.28998549,
         1.91908301,  2.21963528],
       [-1.80840125,  1.22179204, -1.81438851, ..., -1.74506282,
        -0.04813821, -0.75120669]])
```

```
from sklearn.preprocessing import Normalizer
# Normalizer
norm = Normalizer()
X = norm.fit_transform(X)
```

```
# 3
x_train, x_test, y_train, y_test = train_test_split(
X, y, test_size=0.25, random_state=0, stratify=y)
```

```
# 4
from sklearn.neighbors import KNeighborsClassifier
model = KNeighborsClassifier()
model.fit(x_train, y_train)
predict_test = model.predict(x_test)
```

```
# 5
model.score(x_test, y_test)
```

```
0.9090909090909091
```

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test,predict_test)
sns.heatmap(cm,annot=True)
```

```
<Axes: >
```


```
from sklearn.metrics import classification_report
print(classification_report(y_test, predict_test))
```

```
                precision    recall  f1-score   support

             0       0.87      0.89      0.88        53
             1       0.93      0.92      0.93        90

      accuracy                           0.91       143
     macro avg       0.90      0.90      0.90       143
  weighted avg       0.91      0.91      0.91       143
```

Please explain in detail your understanding of the entire activity in **atleast 200 words.**

There are three types of learning in ML which are Supervised (Classification, Regression), Unsupervised (Clustering), and reinforcement Learning.

Scikit-Learn

- 'sklearn' library is used to implement classification regression clustering, Dimensionality Reduction, Preprocessing, and many other types of learning.
- sklearn provides different types of datasets that can be used by importing it.
- An example taught in class was the iris dataset.
- The covariance matrix has a big size of (150,150), so it is important to transpose it.

Seaborn

- Seaborn is used to easily see what we have loaded using data visualization.

Preprocessing

- One of the normalization techniques is the Standard scale used for preprocessing.
- It puts in a single scale.
- scalar.fit_transform(X) In this fit is used to do 'scaling' and 'transform' applies to the result.
- Normalization is another technique for preprocessing, which is used when the direction of the data point is more important than values.
- Normalization must be done for the training data.
- We don't need normalization or standardization variables before doing correlation analysis.

Training and testing data

- To split training and testing datasets we use the train_test_split function which is a part of 'sklearn'.

Confusion Matrix

- 'confusion_matrix' is used for the visualization of the performance of the algorithm.
- It shows how accurately the algorithm is performing.

- Cross-validation (Every iteration has a test dataset) is used to prevent overfitting.

K-Nearest Neighbores (KNN) is classification algoritm where as K-Mean is clusturing algoritm. If K is so low or so high then the result will not be accurate. So choosing proper K value is essential for accurate output.