Practical Workbook

CT-356

Data Mining

Name: Iqra Nawaz

Roll No: DT-22005

Batch: 2022

Department: Data Science

Practical Workbook

Data Mining

(CT – 356)

Prepared by
Mr. Rohail Qamar (CS & IT)

Approved by

Chairman

Department of Computer Science & Information Technology NED University of Engineering & Technology

Table of Contents

Lab 01 - Familiarization & Setting Up the Environment

Objective:

The objective of this lab is to familiarize students with the process of installing and setting up Jupyter Notebook through Anaconda, as well as accessing and using Google Colab for Python programming. By the end of this lab, students will be able to install Jupyter Notebook on their local system using Anaconda, launch and navigate Jupyter Notebooks, and access Google Colab for cloud-based development.

Tools Required

Anaconda Distribution/ Google Colab/ Pycharm

What is Jupyter Notebook?

Jupyter Notebook is an open-source web application that allows users to create and share documents that contain live code, equations, visualizations, and narrative text. It supports various programming languages, but it is most commonly used with Python. Jupyter provides an interactive environment for data cleaning, transformation, numerical simulation, machine learning,and much more. Users can write and execute code in cells, visualize data, and document their thought processes all within a single interface. Jupyter Notebooks are highly popular in data science and academic research for their versatility and ease of use.

INSTALLING ANACONDA ON WINDOWS

Step 1:

Download Anaconda3 Latest version. https://repo.anaconda.com/archive/

Step 2:

Go to your Downloads folder and double-click the installer to launch. To prevent permission errors, do not launch the installer from the Favorites folder.

Step 3:

Click Next.

Step 4:

Read the licensing terms and click I Agree.

Step 5:

It is recommended that you install for Just Me, which will install Anaconda Distribution to just the current user account. Only select an install for All Users if you need to install for all users' accounts on the computer (which requires Windows Administrator privileges).

Step 6:

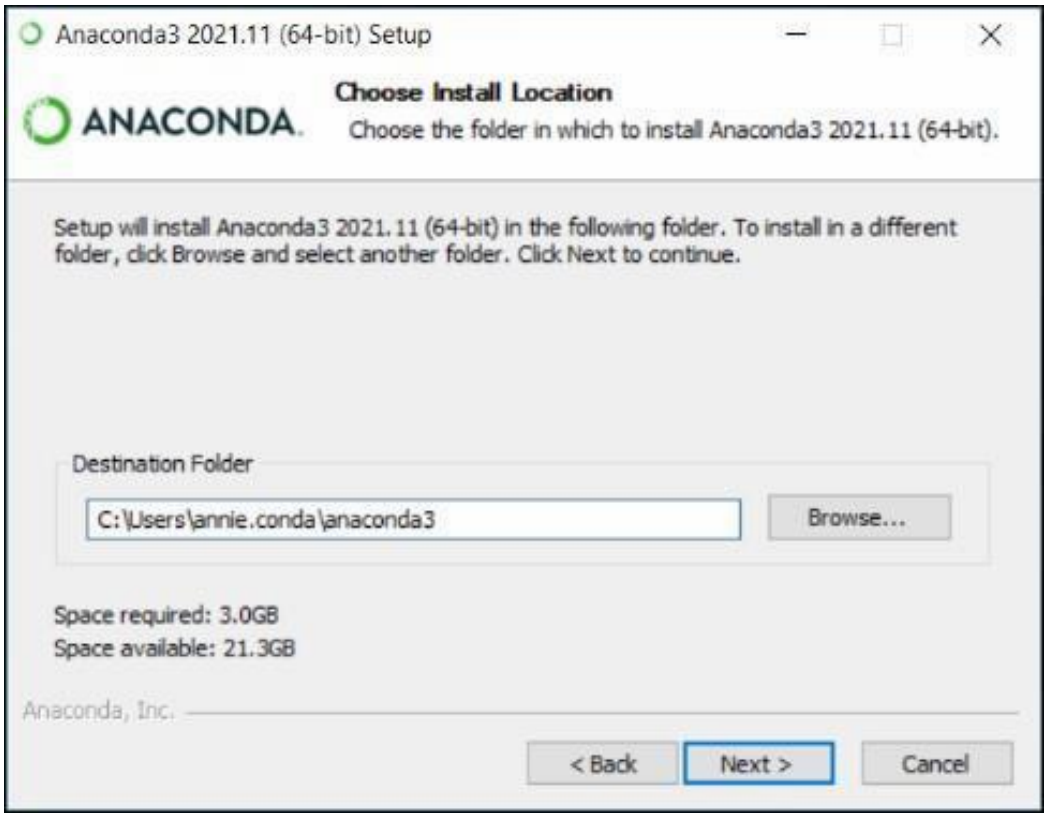
Click Next.

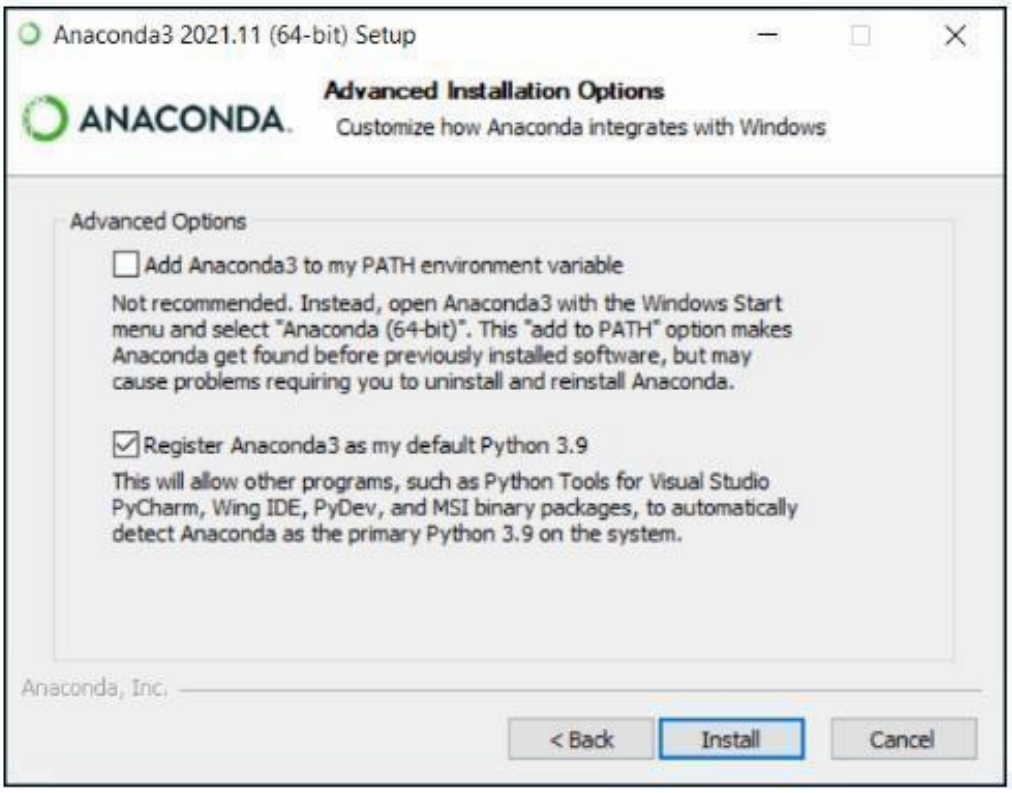
Step 7:

Select a destination folder to install Anaconda and click Next.

Step 8:

Choose whether to add Anaconda to your PATH environment variable or register Anaconda as your default Python.





Step 9:

Click Install. If you want to watch the packages Anaconda is installing, click Show Details.

Step 10:

Click Next.

Step 11:

After a successful installation you will see the “Thanks for installing Anaconda” dialog box:

What is Google Colab:

Google Colab (short for Collaboratory) is a cloud-based platform that allows users to write and execute Python code in an interactive notebook environment. It provides free

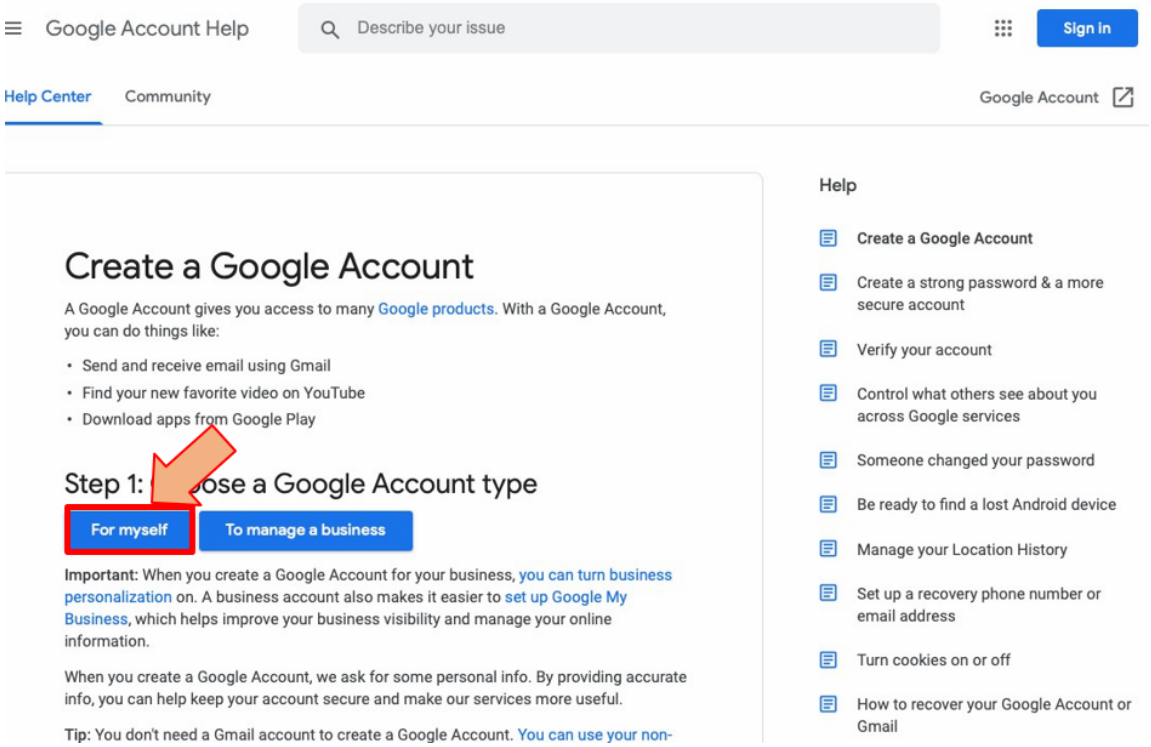
access to powerful computing resources, including GPUs, making it ideal for data analysis, machine learning, and deep learning tasks. Google Colab supports real-time collaboration, enabling multiple users to work on the same notebook simultaneously. It integrates seamlessly with Google Drive, allowing easy sharing and saving of notebooks. It is widely used for quick prototyping, learning, and experimentation with Python code.

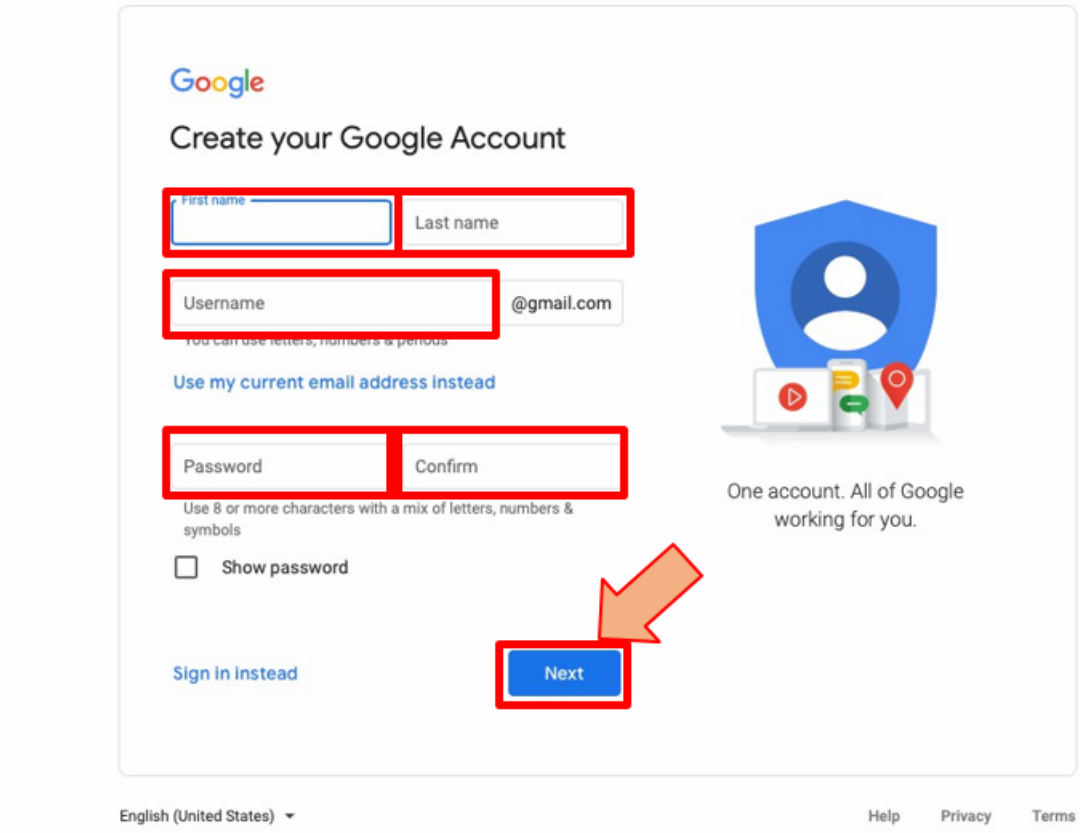
How to use Google Colab:

1. Create you Google account (you can skip this step if you already have a Google account.)
2. Accessing Google Colab

Creating Google Account

Visit: https://support.google.com/accounts/answer/27441?hl=en&ref_topic=3382296




Google
Create your Google Account
First name
Last name
Username
@gmail.com
Use my current email address instead
Password
Confirm
Use 8 or more characters with a mix of letters, numbers & symbols
Show password
Sign in instead
Next
One account. All of Google working for you.
English (United States)
Help
Privacy
Terms

Google

Hacka, welcome to Google

hthon780@gmail.com

Phone number (optional)

We'll use your number for account security. It won't be visible to others.

Recovery email address (optional)

We'll use it to keep your account secure

Month | Day | Year

Your birthday

Gender

Why we ask for this information

Back

Next

Your personal info is private & safe

Google

Privacy and Terms

To create a Google Account, you'll need to agree to the Terms of Service below.
In addition, when you create an account, we process your information as described in our Privacy Policy, including these key points:

Data we process when you use Google

- When you set up a Google Account, we store information you give us like your name, email address, and telephone number.
- When you use Google services to do things like write

data from trillions of search queries to build spell-correction models that we use across all of our services.

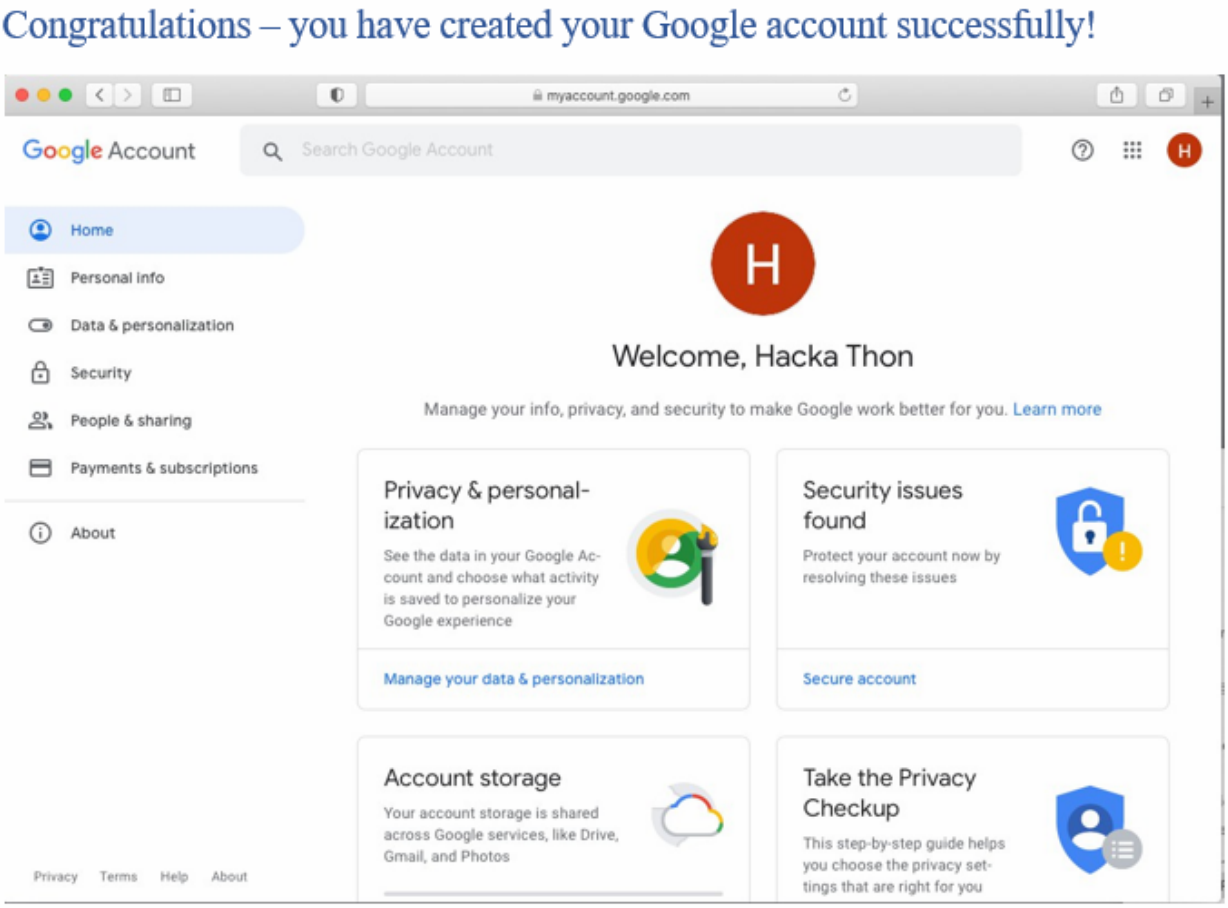
You're in control

Depending on your account settings, some of this data may be associated with your Google Account and we treat this data as personal information. You can control how we collect and use this data now by clicking "More Options" below. You can always adjust your controls later or withdraw your consent for the future by visiting My Account (myaccount.google.com).

MORE OPTIONS

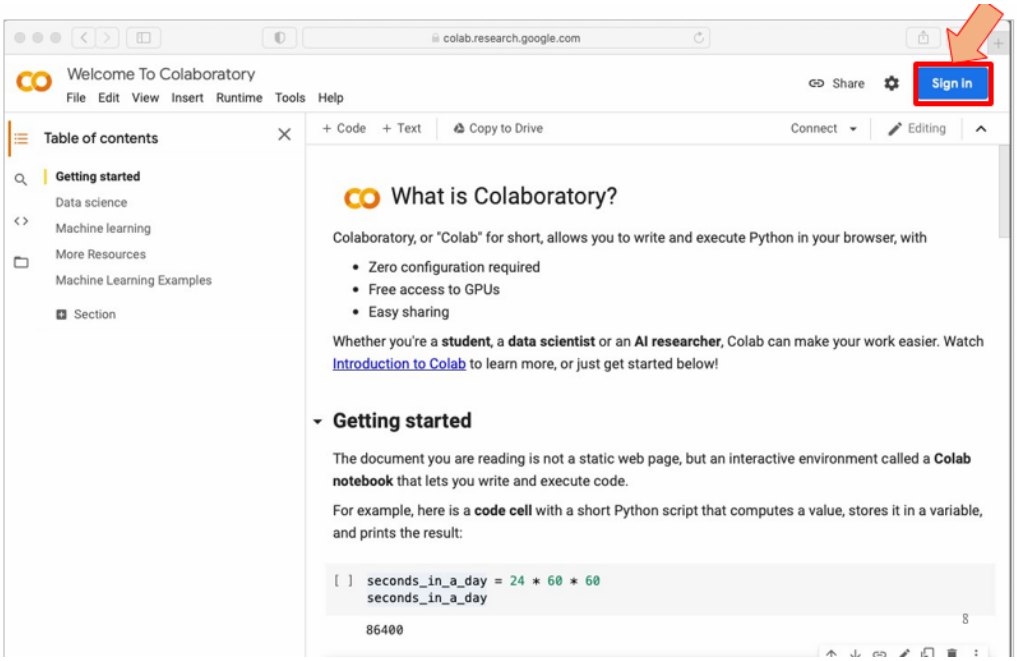
Cancel

I agree

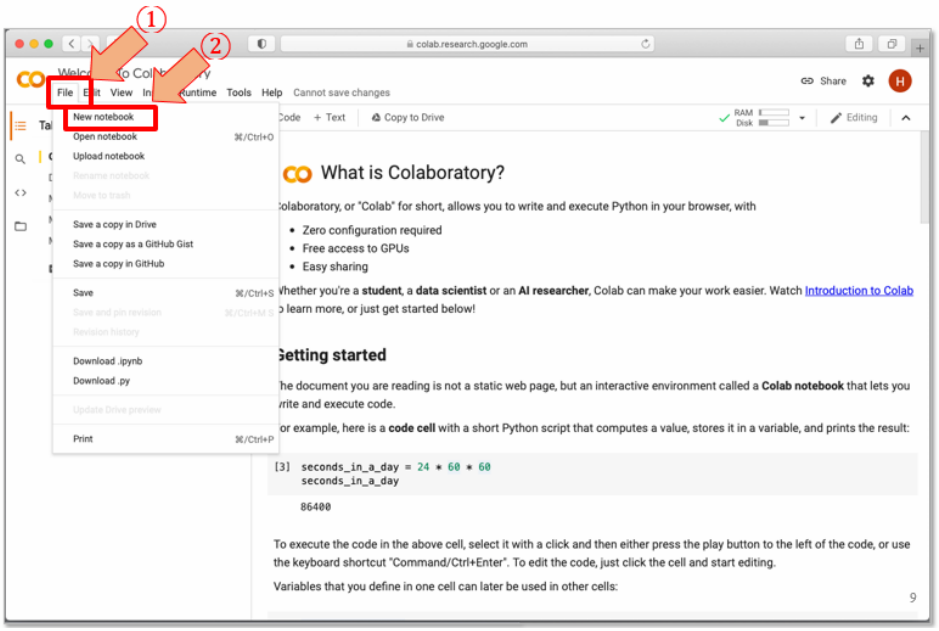


Accessing Colab

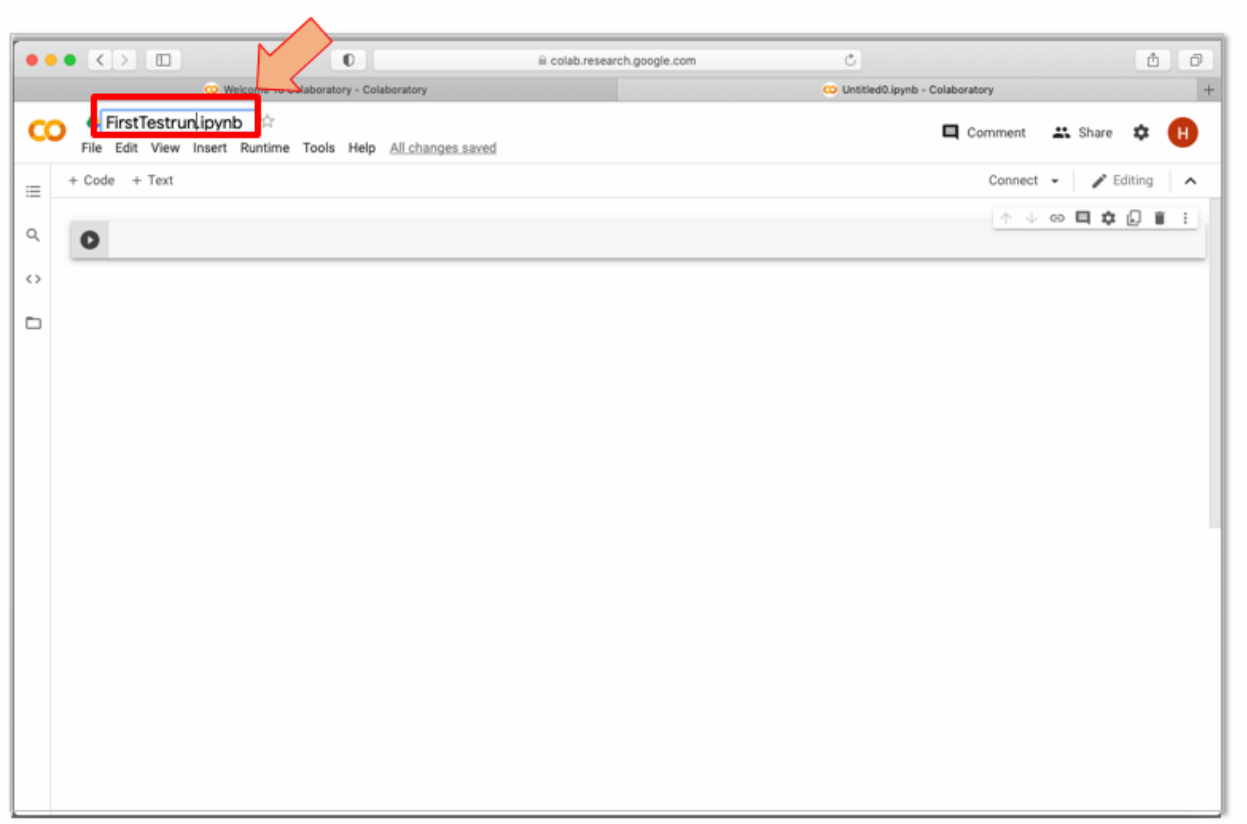
Visit: https://colab.research.google.com/



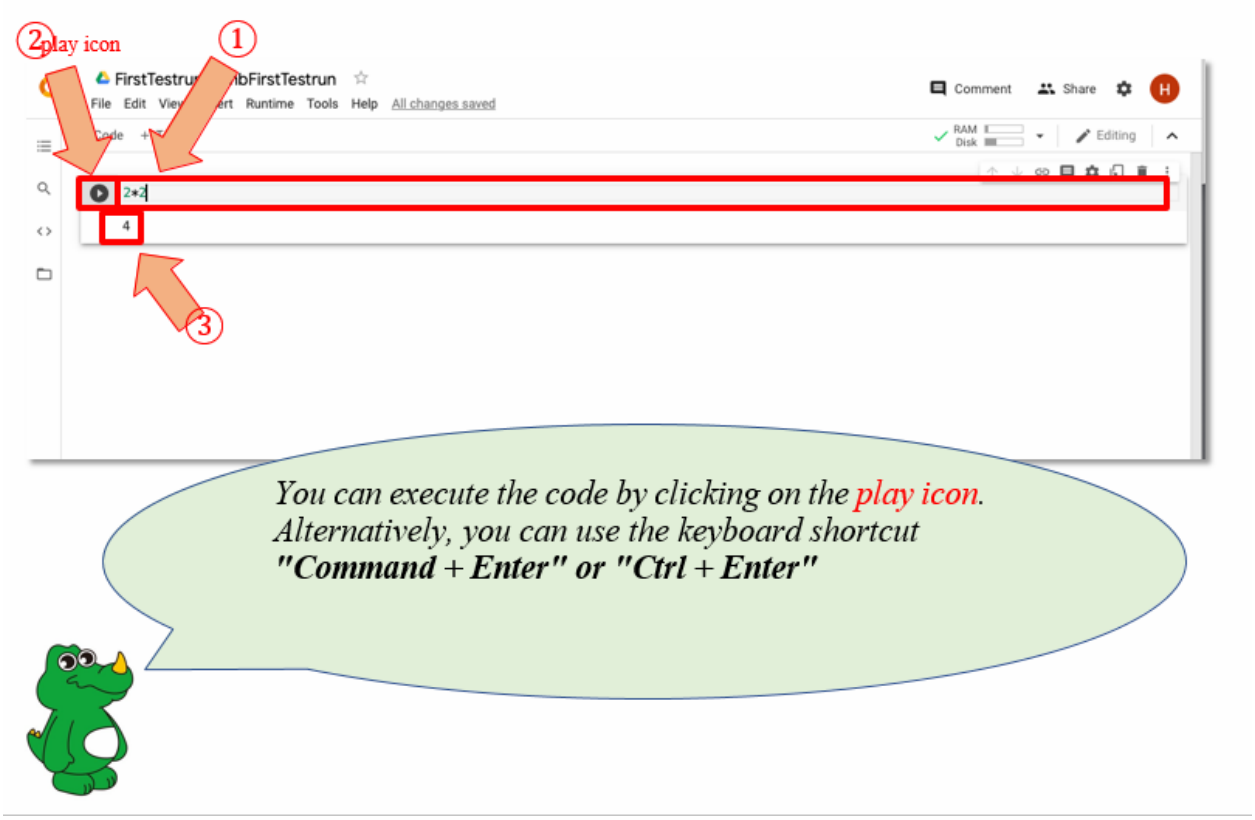
Starting a new Colab notebook



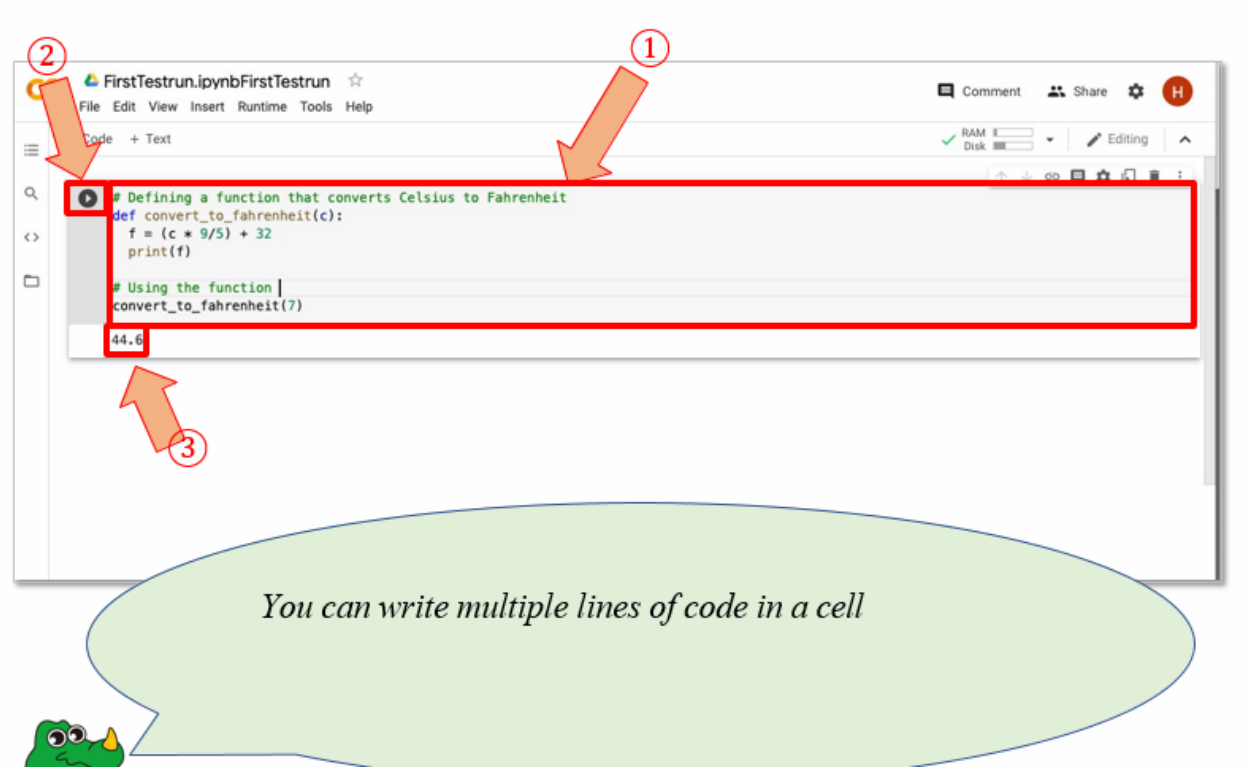
Changing the name of the notebook



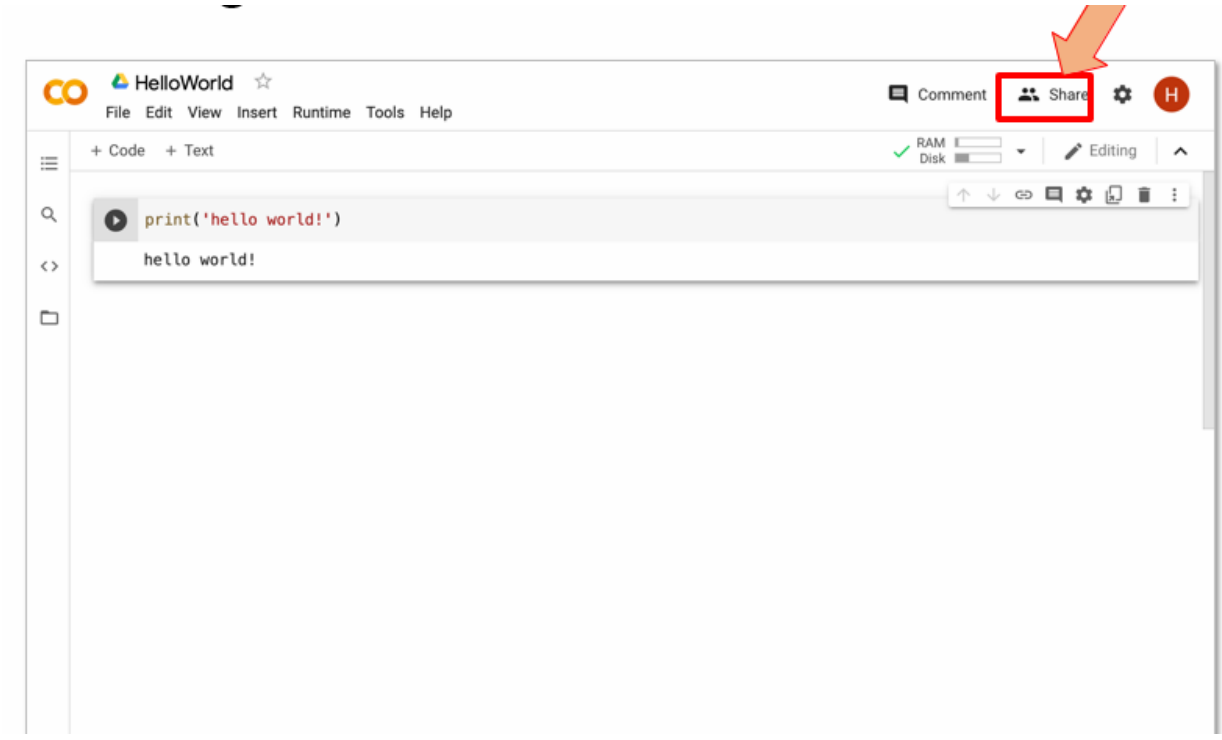
Write code in a cell, and execute it: Example 1

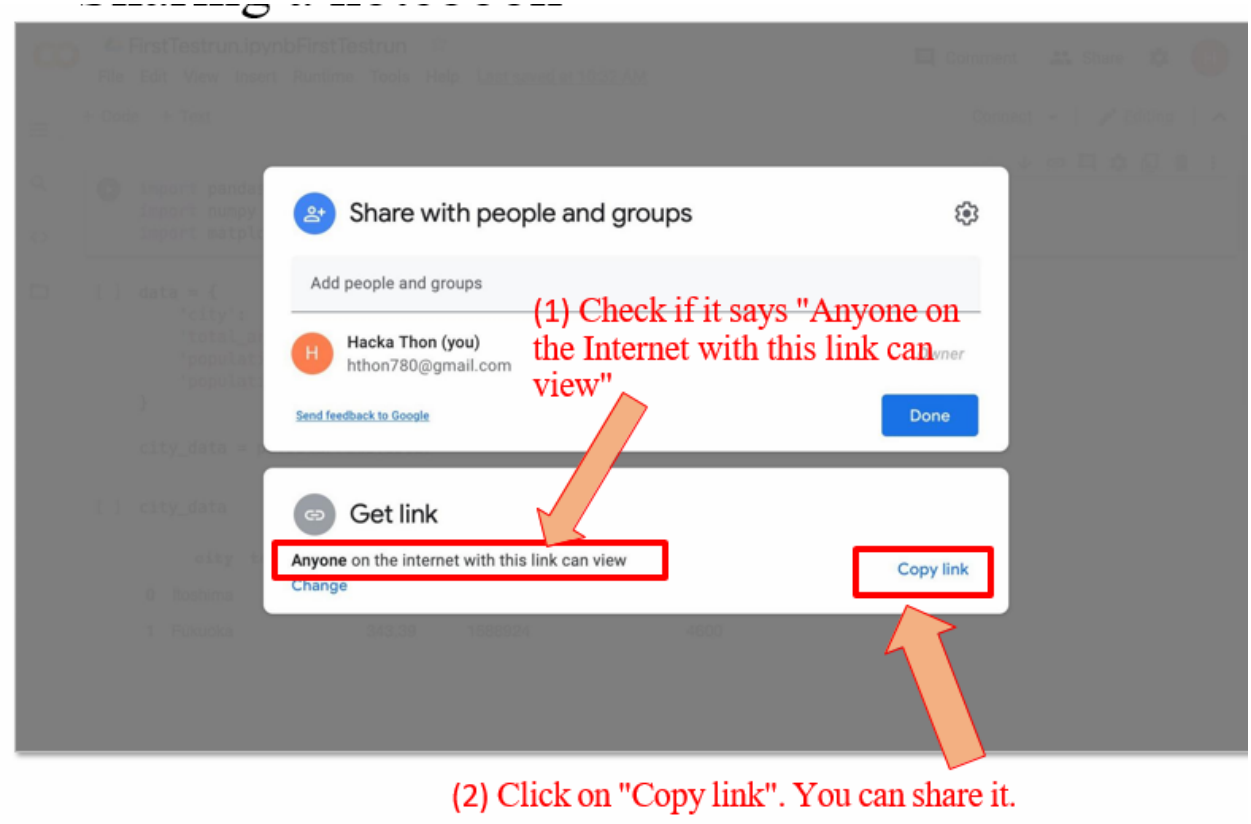


Write code in a cell, and execute it: Example 2

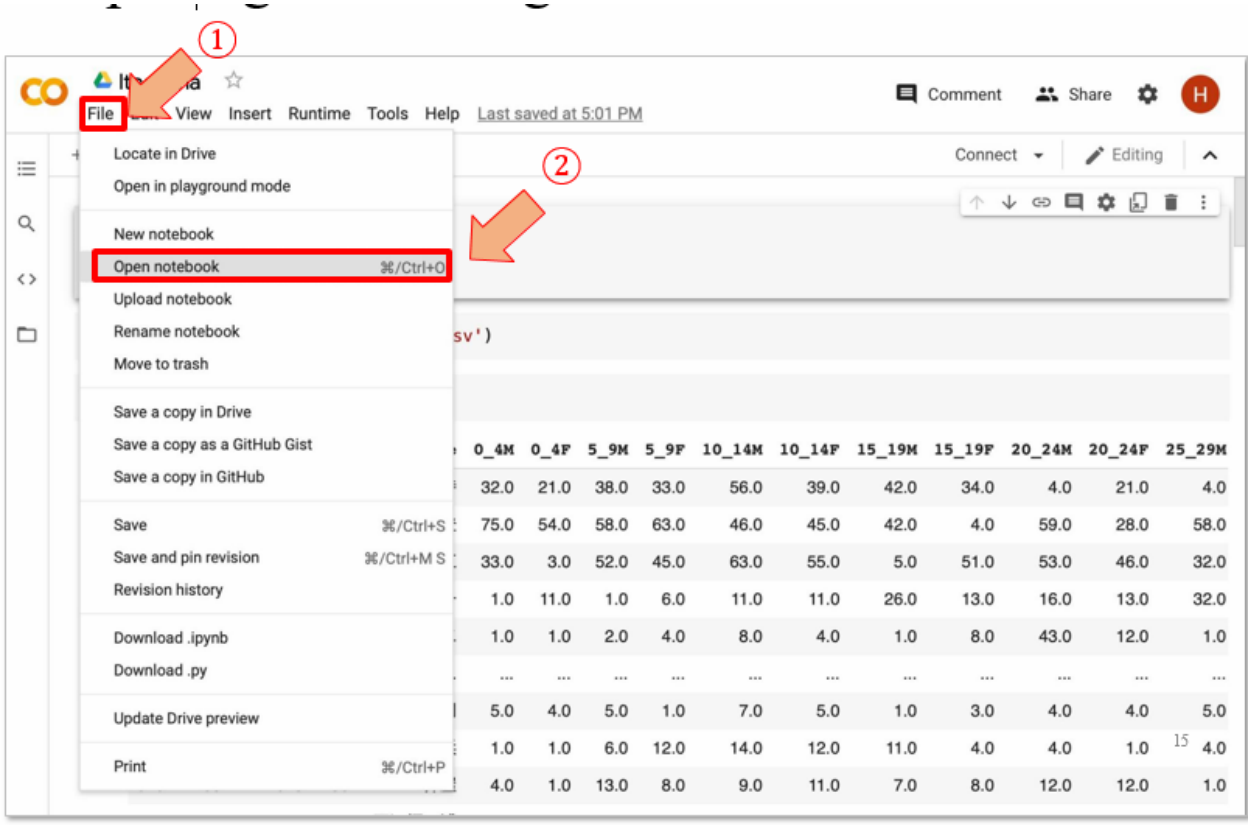


Sharing a Notebook

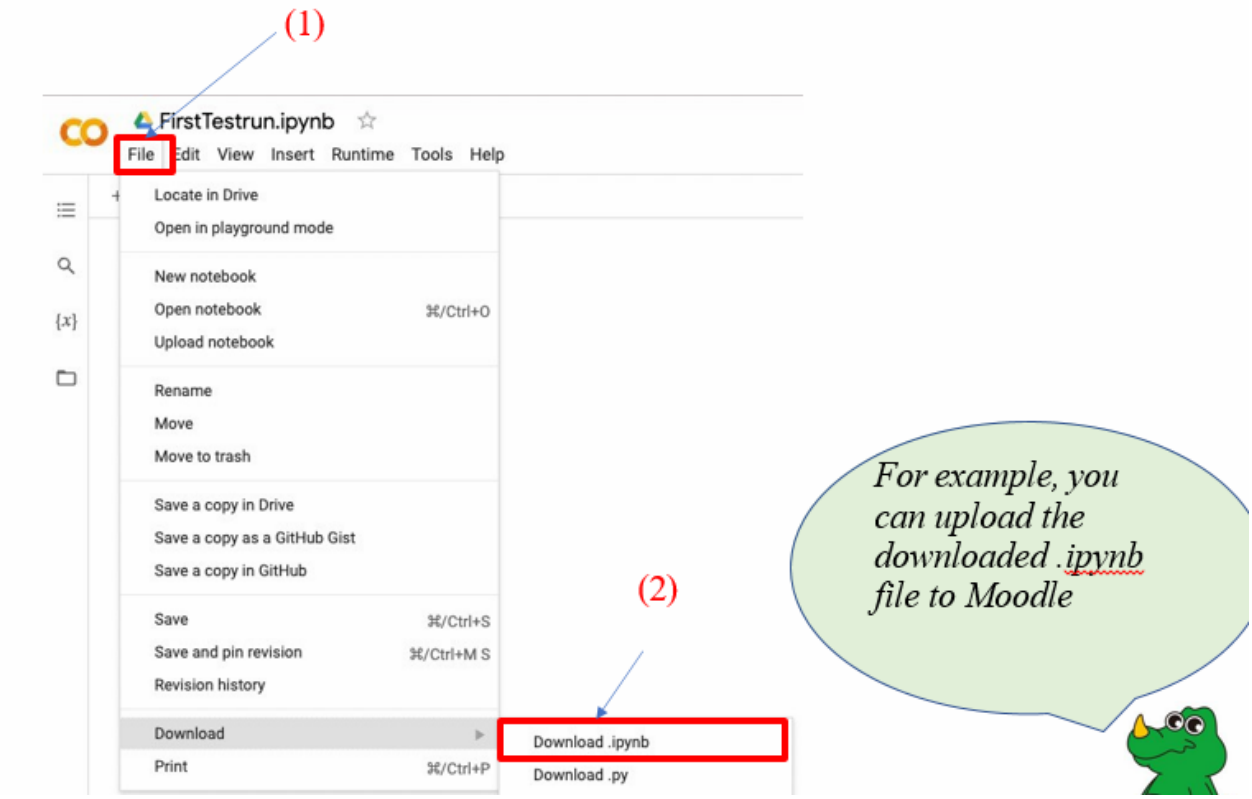




Opening an existing notebook



How to download your Colab notebook (.ipynb file)



Exercise:

1. Write a program to generate a random password of minimum length 8 characters and maximum length no more than 12 characters. The password should satisfy the following criteria:
a) Uppercase Alphabets
b) Lowercase Alphabets
c) Digits
d) Special Characters (like: _, @, etc.)

```
[1] import random
    import string
```

```
[2] def generate_password():
        length = random.randint(8, 12)  # Password length between 8 and 12
        characters = (
            string.ascii_uppercase +  # Uppercase letters
            string.ascii_lowercase +  # Lowercase letters
            string.digits +           # Digits
            "_@#!"                    # Special characters
        )
        password = ''.join(random.choice(characters) for _ in range(length))
        return password

    print("Generated Password:", generate_password())
```

```
Generated Password: dgQs37CMCIhW
```

2. Make a guessing game. Take a user input (any number between 1-5). Then the program will compare with a randomly generated number. Display "Correct Guess" if they match or "Try Again" if not.

```
def guessing_game():
    user_guess = int(input("Guess a number between 1 and 5: "))
    random_number = random.randint(1, 5)  # Generate a random number
    if user_guess == random_number:
        print("Correct Guess!")
    else:
        print("Try Again! The correct number was:", random_number)

guessing_game()
```

```
Guess a number between 1 and 5: 2
Try Again! The correct number was: 4
```

3. Find the 2nd largest element in the given array: arr = [1, 7, 5, 11, 2, 55].

```python
arr = [1, 7, 5, 11, 2, 55]

unique_sorted_array = sorted(set(arr), reverse=True)
second_largest = unique_sorted_array[1]
print("The 2nd largest element is:", second_largest)
```

```
The 2nd largest element is: 11
```

4. Write down the advantages of writing the code in Google Colab over Jupyter Notebook.

1. Accessibility: Google Colab can be accessed from anywhere without installation; it only needs a browser.
2. Free GPU/TPU Support: Colab provides free access to GPUs/TPUs for machine learning tasks.
3. Collaboration: Allows multiple users to edit the same notebook in real time.
4. Cloud Storage: Automatically saves notebooks in Google Drive.

5. Write down the disadvantages of writing the code in Jupyter Notebook over Google Colab.

1. Local Installation Needed: Jupyter requires installation on your computer.
2. No Free GPU Support: Jupyter lacks the free GPU/TPU options that Colab provides.
3. Limited Sharing Options: Sharing Jupyter notebooks requires manual uploads.
4. Storage: Notebooks are stored locally, increasing the risk of loss without backups.

Lab 02 - Exploratory Data Analysis using Pandas

Objective:

- Introduction to Pandas
- Understand the difference between series and Data frames
- Implementation of Series and Data frames
- Applying techniques to deal with missing data
- Summarizing the data
- To learn about concatenation, joining and merging of multiple data frames

Tools Required

Anaconda Distribution/ Google Colab/ Pycharm

Pandas Basics

In this section of the course we will learn how to use pandas for data analysis. You can think of pandas as an extremely powerful version of Excel, with a lot more features. In this lab of the course, you should go through the notebooks in this order. Pandas is a high-level data manipulation tool developed by Wes McKinney. It is built on the Numpy package and its key data structure is called the DataFrame. DataFrames allow you to store and manipulate tabular data in rows of observations and columns of variables There are several ways to create a DataFrame. One way is to use a dictionary. For example:

script.py

```
1  dict = {"country": ["Brazil", "Russia", "India", "China",
   "South Africa"],
2         "capital": ["Brasilia", "Moscow", "New Dehli",
   "Beijing", "Pretoria"],
3         "area": [8.516, 17.10, 3.286, 9.597, 1.221],
4         "population": [200.4, 143.5, 1252, 1357, 52.98] }
5
6  import pandas as pd
7  brics = pd.DataFrame(dict)
8  print(brics)
```

IPython Shell

```
    area    capital       country  population
0   8.516   Brasilia        Brazil      200.40
1  17.100     Moscow        Russia      143.50
2   3.286  New Dehli         India     1252.00
3   9.597    Beijing         China     1357.00
4   1.221   Pretoria  South Africa       52.98

In [1]:
```

Another way to create a DataFrame is by importing a csv file using Pandas. Now, the csv cars.csv is stored and can be imported using pd.read_csv:

script.py

```
# Import pandas as pd
import pandas as pd

# Import the cars.csv data: cars
cars = pd.read_csv('cars.csv')

# Print out cars
print(cars)
```

Indexing DataFrames

There are several ways to index a Pandas DataFrame. One of the easiest ways to do this is by using square bracket notation

In the example below, you can use square brackets to select one column of the cars DataFrame. You can either use a single bracket or a double bracket. The single bracket with output a Pandas Series, while a double bracket will output a Pandas DataFrame.

script.py solution.py

```
# Import pandas and cars.csv
import pandas as pd
cars = pd.read_csv('cars.csv', index_col = 0)

# Print out country column as Pandas Series
print(cars['cars_per_cap'])

# Print out country column as Pandas DataFrame
print(cars[['cars_per_cap']])

# Print out DataFrame with country and drives_right
columns
print(cars[['cars_per_cap', 'country']])
```

IPython Shell

```
IN             18
RU            200
MOR            70
EG             45
     cars_per_cap        country
US            809  United States
AUS           731      Australia
JAP           588          Japan
IN             18          India
RU            200         Russia
MOR            70        Morocco
EG             45          Egypt
```

Common Pandas Functions

1. **read_csv:**

To read a csv format file. It is stand for comma separated view

2. **Slicing data:**

One of the main works in using a pandas dataframe is to be able to slice. In data-science, slicing means creating smaller chunks of dataframe based on some specific

conditions. I will demonstrate how to use one condition slicing and multiple condition slicing.

3. **drop_duplicates:**

drop_duplicates is for deleting duplicate rows in a dataframe. In a general dataframe, there will be cases where you will want to have one copy for each record of the data, and no duplicates. In those situations, you have to use drop_duplicates. The use for this function is:

```
dataframe_deduplicated = dataframe[specific_columns].drop_duplicates()
```

4. **to_csv:**

to_csv is one of the most handy function while you are processing your data and want to save a dataframe for later use. The general use of to_csv is

```
dataframe.to_csv(filepath_to_save_dataframe,index =False)
```

5. **pd.DataFrame:**

Many of the pandas operations are best done using a dataframe. This is the reason, you may want to turn lists of list, lists of similar keyed dictionaries and matrices, multidimensional arrays into a dataframe for sake of operation and ease.

The general go to function for this is pd.dataframe. Let’s say we have two columns, which are student ids and student names. Now, you want to form a table out of it. The way it will work is:

```
student_table = pd.DataFrame()
```

6. **dataframe.fillna:**

This is a general pre-processing function. For dataframe, a common problem is to fill the na values. na values can occur due to different problems and can therefore be solved using different tricks. The cases where the na value percentages are within workable limits, the fillna is used to fill the na positions with some suitable values. In these cases, this fillna function is used. The general use will be:

```
dataframe_changed = dataframe.fillna(value_to_fill,inplace = False)
```

Series

The first main data type we will learn about for pandas is the Series data type. Let's import Pandas and explore the Series object. A Series is very similar to a NumPy array (in fact it is built on top of the NumPy array object). What differentiates the NumPy array from a Series, is that a Series can have axis labels, meaning it can be indexed by a label, instead of just a number location. It also doesn't need to hold numeric data, it can hold any arbitrary Python Object. Let's explore this concept through some examples:

```
: import numpy as np
  import pandas as pd
```

Creating a Series

You can convert a list,numpy array, or dictionary to a Series:

```
labels = ['a','b','c']
my_list = [10,20,30]
arr = np.array(my_list)
d = {'a':10,'b':20,'c':30}
```

```
pd.Series(data = arr)

0    10
1    20
2    30
dtype: int32
```

```
pd.Series(my_list,labels)

a    10
b    20
c    30
dtype: int64
```

```
pd.Series(arr,labels)

a    10
b    20
c    30
dtype: int32
```

** NumPy Arrays **

```
pd.Series(d)

a    10
b    20
c    30
dtype: int64
```

Using an Index

The key to using a Series is understanding its index. Pandas makes use of these index names or numbers by allowing for fast look ups of information (works like a hash table or dictionary).

Let's see some examples of how to grab information from a Series. Let us create two sereis, ser1 and ser2:

```
ser1 = pd.Series([1,2,3,4], index =['USA','Germany','USSR','Japan'])
```

```
ser1
```

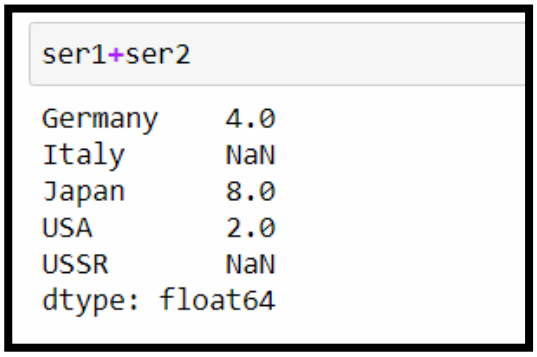
```
USA        1
Germany    2
USSR       3
Japan      4
dtype: int64
```

```
ser2 = pd.Series([1,2,5,4],['USA','Germany','Italy','Japan'])
```

```
ser2
```

```
USA        1
Germany    2
Italy      5
Japan      4
dtype: int64
```

```
ser2 = pd.Series([1,2,5,4],['USA','Germany','Italy','Japan'])
ser2
USA        1
Germany    2
Italy      5
Japan      4
dtype: int64
ser2['Italy']
5
```



```
ser1+ser2
Germany    4.0
Italy      NaN
Japan      8.0
USA        2.0
USSR       NaN
dtype: float64
```

DataFrames

DataFrames are the workhorse of pandas and are directly inspired by the R programming language. We can think of a DataFrame as a bunch of Series objects put together to share the same index. Let's use pandas to explore this topic.

```
import pandas as pd
import numpy as np
```

```
from numpy.random import randn
np.random.seed(101)
```

```
df = pd.DataFrame(randn(5,4),index='A B C D E'.split(),columns='W X Y Z'.split())
```

```
df
```

	W	X	Y	Z
A	2.706850	0.628133	0.907969	0.503826
B	0.651118	-0.319318	-0.848077	0.605965
C	-2.018168	0.740122	0.528813	-0.589001
D	0.188695	-0.758872	-0.933237	0.955057
E	0.190794	1.978757	2.605967	0.683509

Selection and Indexing

Let's learn the various methods to grab data from a DataFrame

```
df['W']
```

```
A    2.706850
B    0.651118
C   -2.018168
D    0.188695
E    0.190794
Name: W, dtype: float64
```

```
# Pass a list of column names
df[['W','Z']]
```

	W	Z
A	2.706850	0.503826
B	0.651118	0.605965
C	-2.018168	-0.589001
D	0.188695	0.955057
E	0.190794	0.683509

```
# SQL Syntax (NOT RECOMMENDED!)
df.W
```

```
A    2.706850
B    0.651118
C   -2.018168
D    0.188695
E    0.190794
Name: W, dtype: float64
```

DataFrame Columns are just Series

DataFrame Columns are just Series

```
type(df['W'])
```

```
pandas.core.series.Series
```

Creating a new column:

```
df['new'] = df['W'] + df['Y']
```

Missing Data

Let's show a few convenient methods to deal with Missing Data in pandas:

```
import numpy as np
import pandas as pd
```

```
df = pd.DataFrame({'A':[1,2,np.nan],
                   'B':[5,np.nan,np.nan],
                   'C':[1,2,3]})
```

```
df
```

	A	B	C
0	1.0	5.0	1
1	2.0	NaN	2
2	NaN	NaN	3

```
df.drop('new',axis=1)
```

	W	X	Y	Z
A	2.706850	0.628133	0.907969	0.503826
B	0.651118	-0.319318	-0.848077	0.605965
C	-2.018168	0.740122	0.528813	-0.589001
D	0.188695	-0.758872	-0.933237	0.955057
E	0.190794	1.978757	2.605967	0.683509

```
# Not inplace unless specified!
df
```

	W	X	Y	Z	new
A	2.706850	0.628133	0.907969	0.503826	3.614819
B	0.651118	-0.319318	-0.848077	0.605965	-0.196959
C	-2.018168	0.740122	0.528813	-0.589001	-1.489355
D	0.188695	-0.758872	-0.933237	0.955057	-0.744542
E	0.190794	1.978757	2.605967	0.683509	2.796762

```
df.dropna()
```

	A	B	C
0	1.0	5.0	1

```
df.dropna(axis=1)
```

	C
0	1
1	2
2	3

```
df.dropna(thresh=2)
```

	A	B	C
0	1.0	5.0	1
1	2.0	NaN	2

```
df.fillna(value='FILL VALUE')
```

	A	B	C
0	1	5	1
1	2	FILL VALUE	2
2	FILL VALUE	FILL VALUE	3

```
df['A'].fillna(value=df['A'].mean())
```

```
0    1.0
1    2.0
2    1.5
Name: A, dtype: float64
```

Groupby

The groupby method allows you to group rows of data together and call aggregate

```
df.groupby('Company')
```

```
<pandas.core.groupby.DataFrameGroupBy object at 0x0000012DD001D198>
```

You can save this object as a new variable:

```
by_comp = df.groupby("Company")
```

And then call aggregate methods off the object:

```
by_comp.mean()
```

Company	Sales
FB	296.5
GOOG	160.0
MSFT	232.0

functions

```
import pandas as pd
# Create dataframe
data = {'Company':['GOOG','GOOG','MSFT','MSFT','FB','FB'],
        'Person':['Sam','Charlie','Amy','Vanessa','Carl','Sarah'],
        'Sales':[200,120,340,124,243,350]}
```

```
df = pd.DataFrame(data)
```

```
df
```

	Company	Person	Sales
0	GOOG	Sam	200
1	GOOG	Charlie	120
2	MSFT	Amy	340
3	MSFT	Vanessa	124
4	FB	Carl	243
5	FB	Sarah	350

```
df.groupby('Company').mea
```

	Sales
Company	
FB	296.5
GOOG	160.0
MSFT	232.0

More examples of aggregate m

```
by_comp.std()
```

	Sales
Company	
FB	75.660426
GOOG	56.568542
MSFT	152.735065

```
by_comp.describe()
```

	Sales							
	count	mean	std	min	25%	50%	75%	max
Company								
FB	2.0	296.5	75.660426	243.0	269.75	296.5	323.25	350.0
GOOG	2.0	160.0	56.568542	120.0	140.00	160.0	180.00	200.0
MSFT	2.0	232.0	152.735065	124.0	178.00	232.0	286.00	340.0

```
by_comp.min()
```

	Person	Sales
Company		
FB	Carl	243
GOOG	Charlie	120
MSFT	Amy	124

```
by_comp.describe().transpose()
```

	Company	FB	GOOG	MSFT
Sales	count	2.000000	2.000000	2.000000
	mean	296.500000	160.000000	232.000000
	std	75.660426	56.568542	152.735065
	min	243.000000	120.000000	124.000000
	25%	269.750000	140.000000	178.000000
	50%	296.500000	160.000000	232.000000
	75%	323.250000	180.000000	286.000000
	max	350.000000	200.000000	340.000000

```
by_comp.describe().transpose()['GOOG']
```

	count	mean	std	min	25%	50%	75%	max
Sales	2.0	160.0	56.568542	120.0	140.0	160.0	180.0	200.0

Exercise:

1. Import pandas and read in the Salaries csv file and set it to a DataFrame called df_sal.

```
import pandas as pd
```

```
[8] df_sal = pd.read_csv("/content/lab2_3_Salaries.csv")
    print(df_sal.head())
```

```
   Id       EmployeeName                                      JobTitle  \
0   1     NATHANIEL FORD  GENERAL MANAGER-METROPOLITAN TRANSIT AUTHORITY
1   2       GARY JIMENEZ                 CAPTAIN III (POLICE DEPARTMENT)
2   3     ALBERT PARDINI                 CAPTAIN III (POLICE DEPARTMENT)
3   4  CHRISTOPHER CHONG            WIRE ROPE CABLE MAINTENANCE MECHANIC
4   5    PATRICK GARDNER    DEPUTY CHIEF OF DEPARTMENT,(FIRE DEPARTMENT)

     BasePay  OvertimePay   OtherPay  Benefits   TotalPay  TotalPayBenefits  \
0  167411.18         0.00  400184.25       NaN  567595.43         567595.43
1  155966.02    245131.88  137811.38       NaN  538909.28         538909.28
2  212739.13    106088.18   16452.60       NaN  335279.91         335279.91
3   77916.00     56120.71  198306.90       NaN  332343.61         332343.61
4  134401.60      9737.00  182234.59       NaN  326373.19         326373.19

   Year  Notes         Agency  Status
0  2011    NaN  San Francisco     NaN
1  2011    NaN  San Francisco     NaN
2  2011    NaN  San Francisco     NaN
3  2011    NaN  San Francisco     NaN
4  2011    NaN  San Francisco     NaN
```

2. How many rows and columns are there?

```
rows, columns = df_sal.shape
print(f"Number of Rows: {rows}, Number of Columns: {columns}")
```

```
Number of Rows: 148654, Number of Columns: 13
```

3. What is the average BasePay?

```
[10] average_basepay = df_sal['BasePay'].mean()
     print(f"Average BasePay: {average_basepay}")
```

```
Average BasePay: 66325.4488404877
```

4. What is the highest amount of OvertimePay in the dataset?

```
[11] highest_overtimepay = df_sal['OvertimePay'].max()
     print(f"Highest OvertimePay: {highest_overtimepay}")
```

```
Highest OvertimePay: 245131.88
```

5. What is the job title of JOSEPH DRISCOLL? Note: Use all caps, otherwise you may get an answer that doesn't match up (there is also a lowercase Joseph Driscoll). ?

```
[12] job_title = df_sal[df_sal['EmployeeName'] == 'JOSEPH DRISCOLL']['JobTitle'].values[0]
     print(f"Job Title of JOSEPH DRISCOLL: {job_title}")
```

```
Job Title of JOSEPH DRISCOLL: CAPTAIN, FIRE SUPPRESSION
```

6. How much does JOSEPH DRISCOLL make (including benefits)?

```
[13] total_pay = df_sal[df_sal['EmployeeName'] == 'JOSEPH DRISCOLL']['TotalPayBenefits'].values[0]
     print(f"JOSEPH DRISCOLL makes (including benefits): {total_pay}")
```

```
JOSEPH DRISCOLL makes (including benefits): 270324.91
```

7. What is the name of highest paid person (including benefits)?

```
[14] highest_paid_person = df_sal[df_sal['TotalPayBenefits'] == df_sal['TotalPayBenefits'].max()]['EmployeeName'].values[0]
     print(f"Highest Paid Person: {highest_paid_person}")
```

```
Highest Paid Person: NATHANIEL FORD
```

8. What is the name of lowest paid person (including benefits)? Do you notice something strange about how much he or she is paid?

```
[15] lowest_paid_person = df_sal[df_sal['TotalPayBenefits'] == df_sal['TotalPayBenefits'].min()]['EmployeeName'].values[0]
     lowest_pay = df_sal['TotalPayBenefits'].min()
     print(f"Lowest Paid Person: {lowest_paid_person}, Paid: {lowest_pay}")
```

```
Lowest Paid Person: Joe Lopez, Paid: -618.13
```

9. What was the average (mean) BasePay of all employees per year? (2011-2014)?

```
average_basepay_per_year = df_sal.groupby('Year')['BasePay'].mean()
print("Average BasePay per year (2011-2014):")
print(average_basepay_per_year)
```

```
Average BasePay per year (2011-2014):
Year
2011    63595.956517
2012    65436.406857
2013    69630.030216
2014    66564.421924
Name: BasePay, dtype: float64
```

10. What are the top 5 most common jobs?

```
[17] top_jobs = df_sal['JobTitle'].value_counts().head(5)
     print("Top 5 Most Common Jobs:")
     print(top_jobs)
```

```
Top 5 Most Common Jobs:
JobTitle
Transit Operator                7036
Special Nurse                   4389
Registered Nurse                3736
Public Svc Aide-Public Works    2518
Police Officer 3                2421
Name: count, dtype: int64
```

11. How many Job Titles were represented by only one person in 2013? (e.g. Job Titles with only one occurrence in 2013?)

```
unique_jobs_2013 = df_sal[df_sal['Year'] == 2013]['JobTitle'].value_counts()
one_person_jobs = (unique_jobs_2013 == 1).sum()
print(f"Number of Job Titles represented by only one person in 2013: {one_person_jobs}")
```

```
Number of Job Titles represented by only one person in 2013: 202
```

Lab 03 - Data Visualization

Objective:

- Understand the importance of data visualization
- Applying visualization techniques using matplotlib library
- Analyzing the data using visualization to get insights and useful information

Tools Required

Anaconda Distribution/ Google Colab/ Pycharm

THEORY

Data visualization is the discipline of trying to understand data by placing it in a visual context so that patterns, trends and correlations that might not otherwise be detected can be exposed. Python offers multiple great graphing libraries that come packed with lots of different features. No matter if you want to create interactive, live or highly customized plots python has an excellent library for you.

To get a little overview here are a few popular plotting libraries:

- **Matplotlib:** low level, provides lots of freedom
- **Pandas Visualization:** easy to use interface, built on Matplotlib
- **Seaborn:** high-level interface, great default styles
- **ggplot:** based on R's ggplot2, uses Grammar of Graphics
- **Plotly:** can create interactive plots

We will learn how to create basic plots using Matplotlib, Pandas visualization and Seaborn as well as how to use some specific features of each library.

Importing Datasets

In this article, we will use two datasets which are freely available. The Iris and Wine Reviews dataset, which we can both load in using pandas read_csv method.

```
import pandas as pd
iris = pd.read_csv('iris.csv', names=['sepal_length', 'sepal_width', 'petal_length',
print(iris.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

Matplotlib

Matplotlib is the most popular python plotting library. It is a low-level library with a Matlab like interface which offers lots of freedom at the cost of having to write more code.

To install Matplotlib pip and conda can be used.

```
pip install matplotlib
or
conda install matplotlib
```

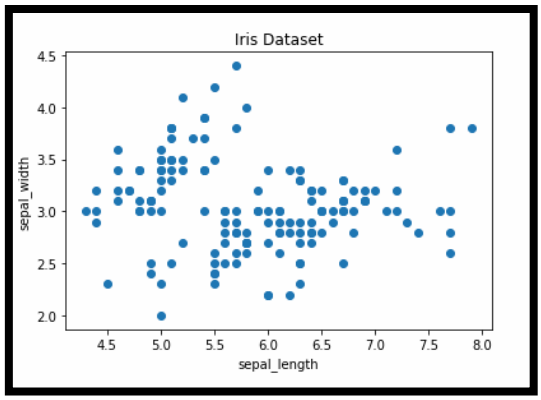
Matplotlib is specifically good for creating basic graphs like line charts, bar charts, histograms and many more. It can be imported by typing:

```
import matplotlib.pyplot as plt
```

Scatter Plot

To create a scatter plot in Matplotlib we can use the scatter method. We will also create a figure and an axis using plt.subplots so we can give our plot a title and labels.

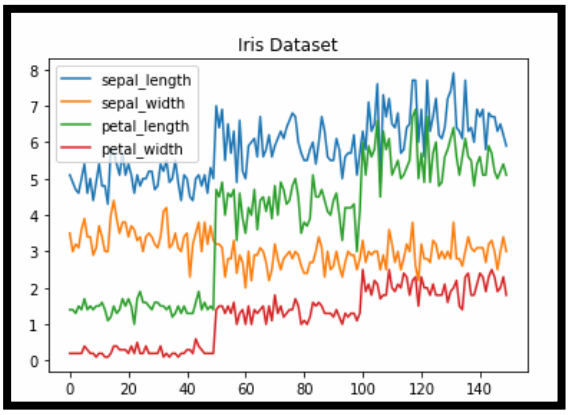
```
# create a figure and axis
fig, ax = plt.subplots()

# scatter the sepal_length against the sepal_width
ax.scatter(iris['sepal_length'], iris['sepal_width'])
# set a title and labels
ax.set_title('Iris Dataset')
ax.set_xlabel('sepal_length')
ax.set_ylabel('sepal_width')
```



Line Chart

In Matplotlib we can create a line chart by calling the plot method. We can also plot multiple columns in one graph, by looping through the columns we want and plotting each column on the same axis.

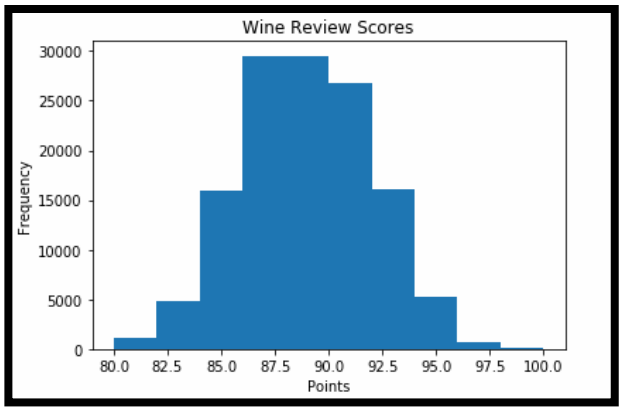
```
# get columns to plot
columns = iris.columns.drop(['class'])
# create x data
x_data = range(0, iris.shape[0])
# create figure and axis
fig, ax = plt.subplots()
# plot each column
for column in columns:
    ax.plot(x_data, iris[column], label=column)
# set title and legend
ax.set_title('Iris Dataset')
ax.legend()
```



Histogram

In Matplotlib we can create a Histogram using the hist method. If we pass it categorical data like the points column from the wine-review dataset it will automatically calculate how often each class occurs.

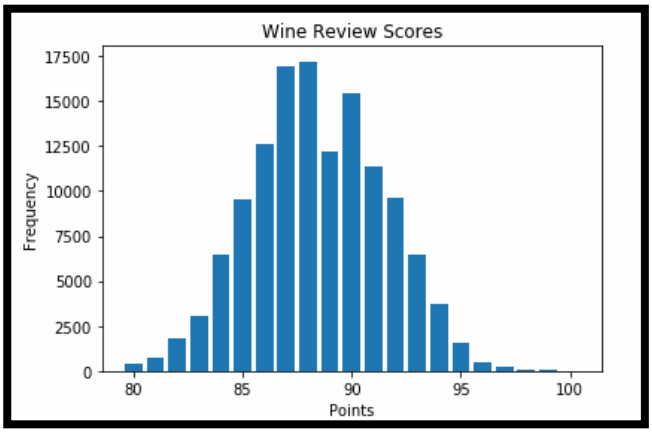
```
# create figure and axis
fig, ax = plt.subplots()
# plot histogram
ax.hist(wine_reviews['points'])
# set title and labels
ax.set_title('Wine Review Scores')
ax.set_xlabel('Points')
ax.set_ylabel('Frequency')
```



Bar Chart

A bar chart can be created using the bar method. The bar-chart isn't automatically calculating the frequency of a category so we are going to use pandas value_counts function to do this. The bar-chart is useful for categorical data that doesn't have a lot of different categories (less than 30) because else it can get quite messy.

```
# create a figure and axis
fig, ax = plt.subplots()
# count the occurrence of each class
data = wine_reviews['points'].value_counts()
# get x and y data
points = data.index
frequency = data.values
# create bar chart
ax.bar(points, frequency)
# set title and labels
ax.set_title('Wine Review Scores')
ax.set_xlabel('Points')
ax.set_ylabel('Frequency')
```



Seaborn

Seaborn is a Python data visualization library based on Matplotlib. It provides a high-level interface for creating attractive graphs. Seaborn has a lot to offer. You can create graphs in one line that would take you multiple tens of lines in Matplotlib. Its standard designs are awesome and it also has a nice interface for working with pandas dataframes.

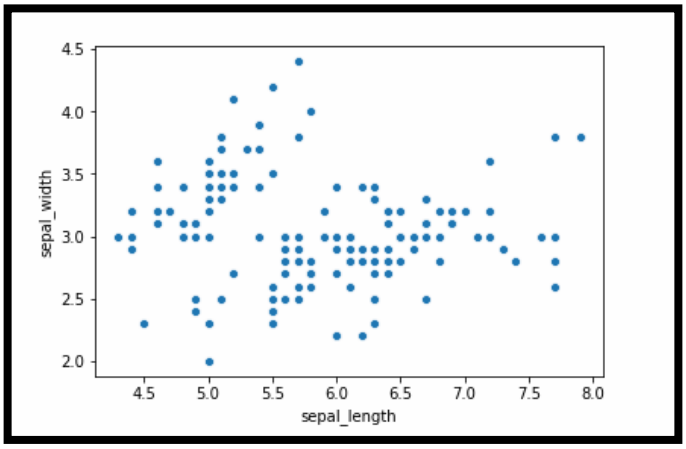
It can be imported by typing:

```
import seaborn as sns
```

Scatter plot

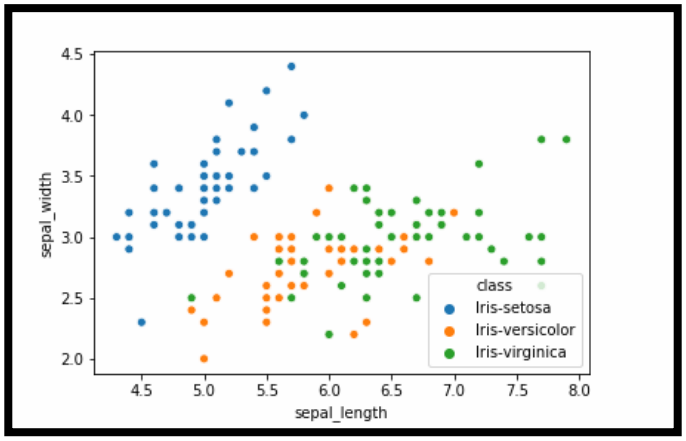
We can use the .scatterplot method for creating a scatterplot, and just as in Pandas we need to pass it the column names of the x and y data, but now we also need to pass the data as an additional argument because we aren't calling the function on the data directly as we did in Pandas.

```
sns.scatterplot(x='sepal_length', y='sepal_width', data=iris)
```



We can also highlight the points by class using the hue argument, which is a lot easier than in Matplotlib.

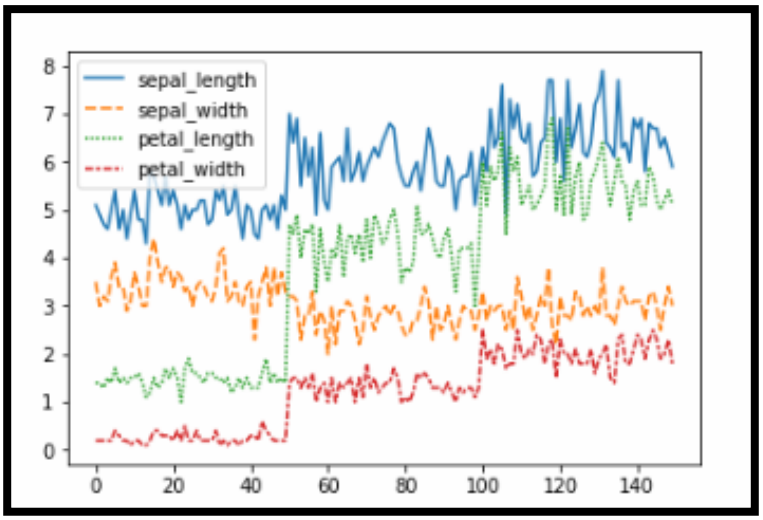
```
sns.scatterplot(x='sepal_length', y='sepal_width', hue='class', data=iris)
```



Line chart

To create a line-chart the sns.lineplot method can be used. The only required argument is the data, which in our case are the four numeric columns from the Iris dataset. We could also use the sns.kdeplot method which rounds of the edges of the curves and therefore is cleaner if you have a lot of outliers in your dataset.

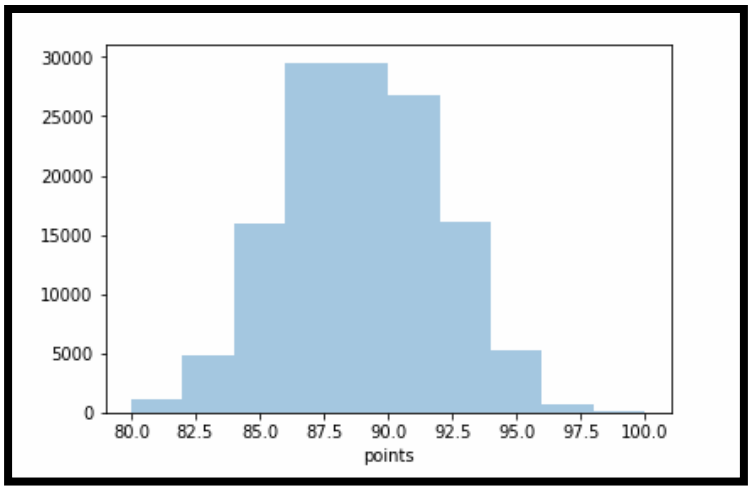
```
sns.lineplot(data=iris.drop(['class'], axis=1))
```



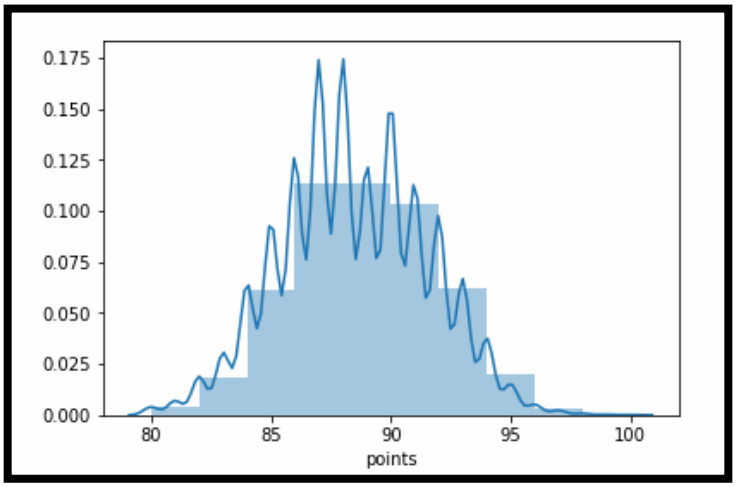
Histogram

To create a histogram in Seaborn we use the sns.distplot method. We need to pass it the column we want to plot and it will calculate the occurrences itself. We can also pass it the number of bins, and if we want to plot a gaussian kernel density estimate inside the graph

```
sns.distplot(wine_reviews['points'], bins=10, kde=False)
```



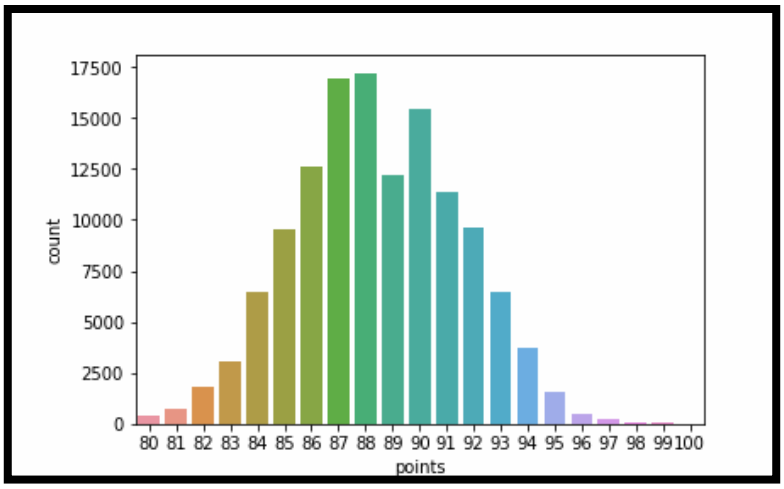

```
sns.distplot(wine_reviews['points'], bins=10, kde=True)
```



Bar chart

In Seaborn a bar-chart can be created using the sns.countplot method and passing it the data

```
sns.countplot(wine_reviews['points'])
```



Exercise:

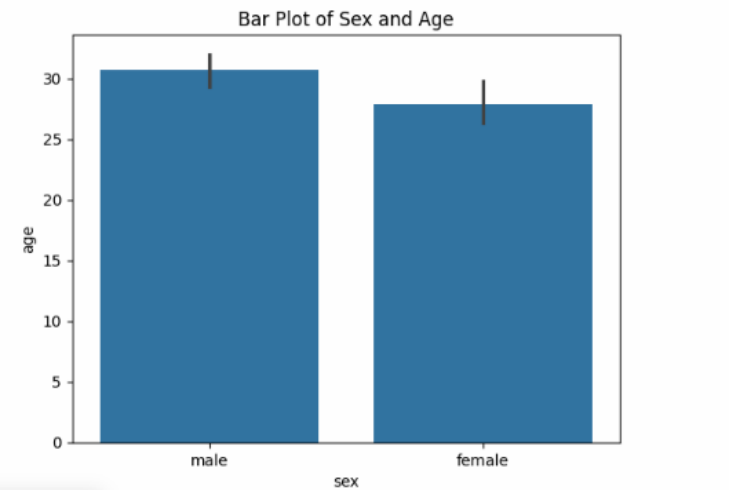
1. Import Seaborn and Matplotlib libraries and load the Titanic dataset.

```
[28] import seaborn as sns
     import matplotlib.pyplot as plt
     titanic = sns.load_dataset("titanic")
     print(titanic.head())
```

```
   survived  pclass     sex   age  sibsp  parch     fare embarked  class  \
0         0       3    male  22.0      1      0   7.2500        S  Third
1         1       1  female  38.0      1      0  71.2833        C  First
2         1       3  female  26.0      0      0   7.9250        S  Third
3         1       1  female  35.0      1      0  53.1000        S  First
4         0       3    male  35.0      0      0   8.0500        S  Third

     who  adult_male deck  embark_town alive  alone
0    man        True  NaN  Southampton    no  False
1  woman       False    C    Cherbourg   yes  False
2  woman       False  NaN  Southampton   yes   True
3  woman       False    C  Southampton   yes  False
4    man        True  NaN  Southampton    no   True
```

2. Make a bar plot with sex and age.

```
sns.barplot(x="sex", y="age", data=titanic)
plt.title("Bar Plot of Sex and Age")
plt.show()
```



3. Find how many passengers are alive.

```
[22] alive_count = titanic[titanic["alive"] == "yes"].shape[0]
     print(f"Number of passengers alive: {alive_count}")
```

```
Number of passengers alive: 342
```

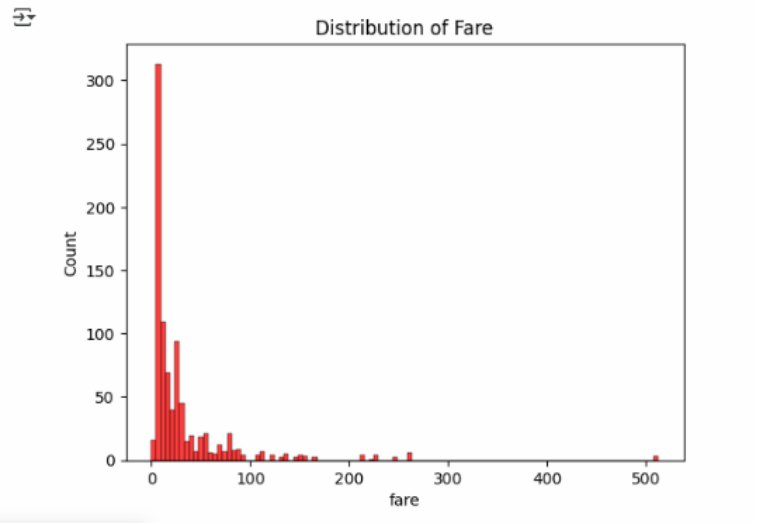
4. How many male and female passengers survived w.r.t passenger class?

```
survived_by_class_and_sex = titanic[titanic["survived"] == 1].groupby(["class", "sex"]).size()
print(survived_by_class_and_sex)
```

```
class   sex
First   female    91
        male      45
Second  female    70
        male      17
Third   female    72
        male      47
dtype: int64
```

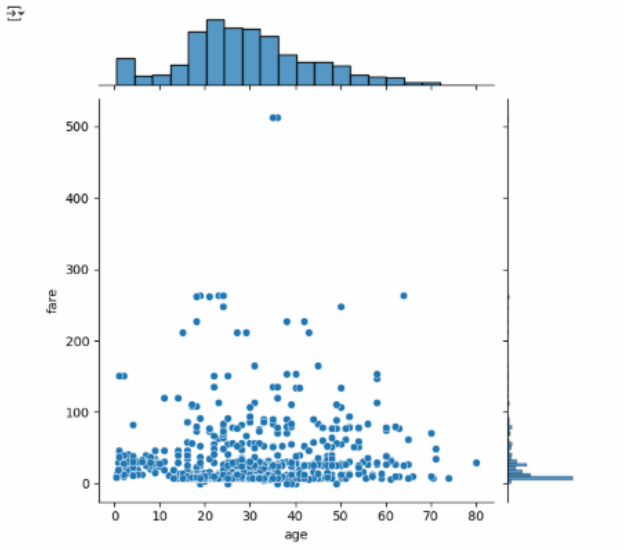
5. Make a distribution plot of fare, making the color of the bars red.

```
[23] sns.histplot(titanic["fare"], color="red", kde=False)
     plt.title("Distribution of Fare")
     plt.show()
```



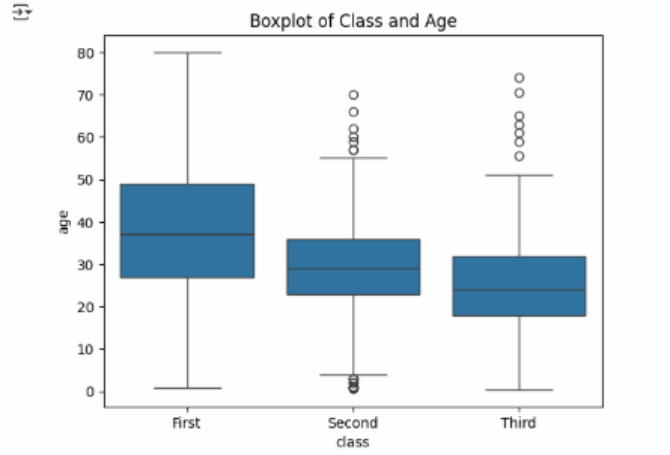
6. Make a joint plot of fare and age. Write down which age group is the highest in the graph.

```
sns.jointplot(x="age", y="fare", data=titanic, kind="scatter")
plt.show()
```



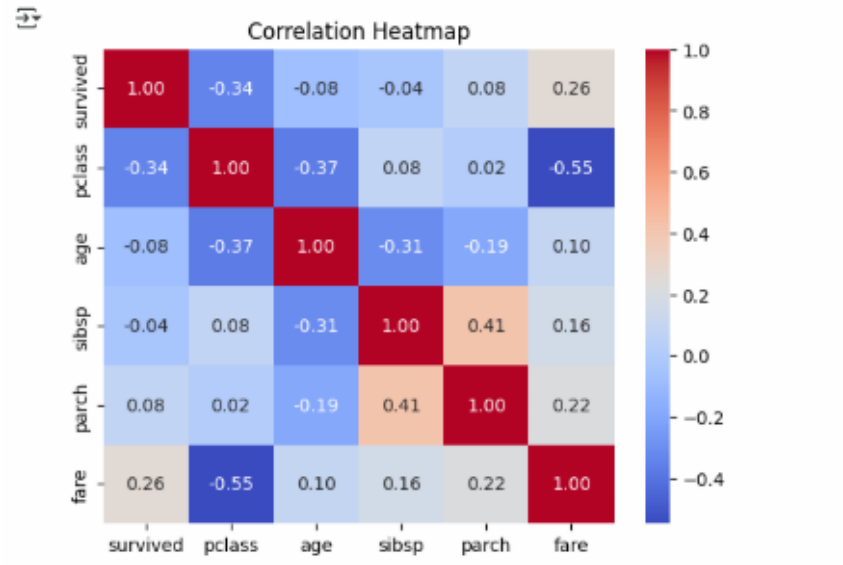
7. Make a box plot of class and age.

```
[26] # Boxplot of class and age
     sns.boxplot(x="class", y="age", data=titanic)
     plt.title("Boxplot of Class and Age")
     plt.show()
```



8. Show the graph of correlation between attributes of the dataset.

```
[28] # Select only numeric columns for correlation
     numeric_data = titanic.select_dtypes(include=['float64', 'int64'])
     correlation = numeric_data.corr()

     # Plot the heatmap
     sns.heatmap(correlation, annot=True, cmap="coolwarm", fmt=".2f")
     plt.title("Correlation Heatmap")
     plt.show()
```



Lab 04 - K-Means Clustering

Objective:

The objective of this lab is to provide students with hands-on experience in implementing the K-Means clustering algorithm for unsupervised learning. By the end of this lab, students will be able to preprocess datasets to make them suitable for clustering. Apply the K-Means algorithm using Python libraries such as scikit-learn. Evaluate clustering performance using metrics like inertia and silhouette scores. Visualize clustering results to interpret the grouping of data points effectively.

Tools Required

Anaconda Distribution/ Google Colab/ Pycharm

What is K-Means Clustering?

K-Means Clustering is an Unsupervised Learning algorithm, which groups the unlabeled dataset into different clusters. Here K defines the number of pre-defined clusters that need to be created in the process, as if K=2, there will be two clusters, and for K=3, there will be three clusters, and so on. It is an iterative algorithm that divides the unlabeled dataset into k different clusters in such a way that each dataset belongs only one group that has similar properties

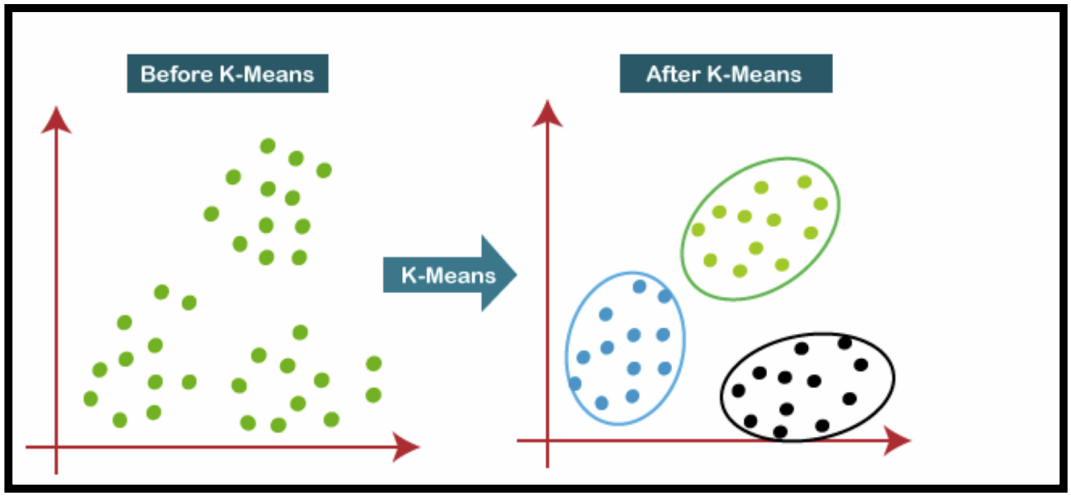
It is a centroid-based algorithm, where each cluster is associated with a centroid. The main aim of this algorithm is to minimize the sum of distances between the data point and their corresponding clusters.

The algorithm takes the unlabeled dataset as input, divides the dataset into k-number of clusters, and repeats the process until it does not find the best clusters. The value of k should be predetermined in this algorithm.

The k-means clustering algorithm mainly performs two tasks:

- Determines the best value for K center points or centroids by an iterative process.
- Assigns each data point to its closest k-center. Those data points which are near to the particular k-center, create a cluster

The below diagram explains the working of the K-means Clustering Algorithm:



How does the K-Means Algorithm Work?

The working of the K-Means algorithm is explained in the below steps:

Step-1: Select the number K to decide the number of clusters.

Step-2: Select random K points or centroids. (It can be other from the input dataset).

Step-3: Assign each data point to their closest centroid, which will form the predefined K clusters.

Step-4: Calculate the variance and place a new centroid of each cluster.

Step-5: Repeat the third steps, which means reassign each datapoint to the new closest centroid of each cluster.

Step-6: If any reassignment occurs, then go to step-4 else go to FINISH.

Step-7: The model is ready.

How to choose the value of "K number of clusters" in K-means Clustering?

The performance of the K-means clustering algorithm depends upon highly efficient clusters that it forms. However choosing the optimal number of clusters is a big task. There are some different ways to find the optimal number of clusters, but here we are discussing the most appropriate method to find the number of clusters or value of K. The method is given below:

Elbow Method: The Elbow method is one of the most popular ways to find the optimal number of clusters. This method uses the concept of WCSS value. **WCSS** stands for **Within Cluster Sum of Squares**, which defines the total variations within a cluster. The formula to calculate the value of WCSS (for 3 clusters) is given below:

$$WCSS= \sum_{Pi \text{ in Cluster1}} distance(P_i\ C_1)^2 + \sum_{Pi \text{ in Cluster2}} distance(P_i\ C_2)^2 + \sum_{Pi \text{ in CLuster3}} distance(P_i\ C_3)^2$$

In the above formula of WCSS,

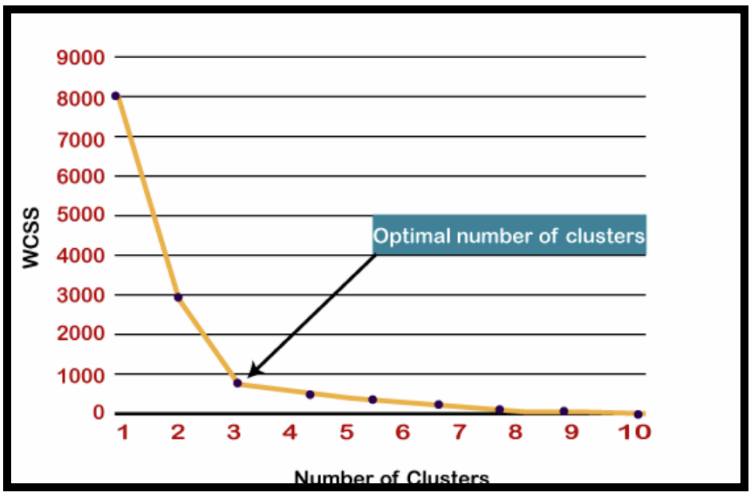
$\sum_{Pi \text{ in Cluster1}} distance(P_i\ C_1)^2$: It is the sum of the square of the distances between each data point and its centroid within a cluster1 and the same for the other two terms.

To measure the distance between data points and centroid, we can use any method such as Euclidean distance or Manhattan distance.

To find the optimal value of clusters, the elbow method follows the below steps:

- It executes the K-means clustering on a given dataset for different K values (ranges from 1-10).
- For each value of K, calculates the WCSS value.
- Plots a curve between calculated WCSS values and the number of clusters K.
- The sharp point of bend or a point of the plot looks like an arm, then that point is considered as the best value of K.

Since the graph shows the sharp bend, which looks like an elbow, hence it is known as the elbow method. The graph for the elbow method looks like the below image:



Code Example of K-means Clustering:

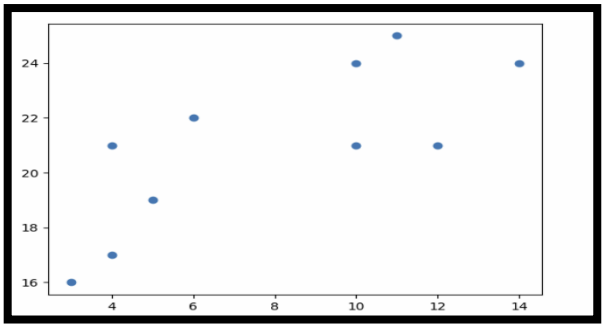
1) Import the modules you need.

```
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
```

2) Visualize the data

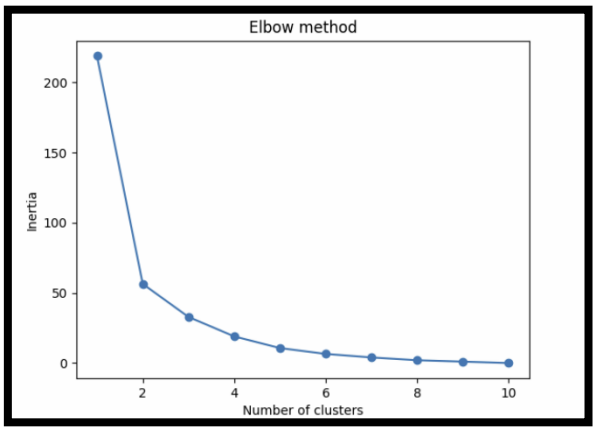
Start by visualizing some data points:

```
import matplotlib.pyplot as plt

x = [4, 5, 10, 4, 3, 11, 14 , 6, 10, 12]
y = [21, 19, 24, 17, 16, 25, 24, 22, 21, 21]

plt.scatter(x, y)
plt.show()
```



3) Determine the value of k using elbow method:

```python
from sklearn.cluster import KMeans

data = list(zip(x, y))
inertias = []

for i in range(1,11):
    kmeans = KMeans(n_clusters=i)
    kmeans.fit(data)
    inertias.append(kmeans.inertia_)

plt.plot(range(1,11), inertias, marker='o')
plt.title('Elbow method')
plt.xlabel('Number of clusters')
plt.ylabel('Inertia')
plt.show()
```

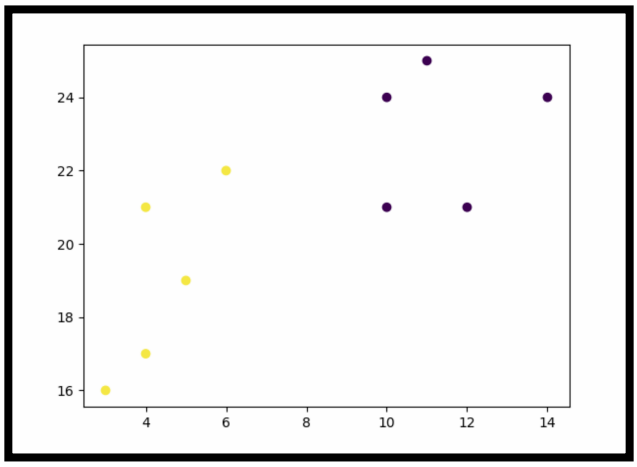


The elbow method shows that **2** is a good value for K, so we retrain and visualize the result:

4) Fit the model

```python
kmeans = KMeans(n_clusters=2)
kmeans.fit(data)

plt.scatter(x, y, c=kmeans.labels_)
plt.show()
```

5) Visualize the result:



Advantages of K-Means Algorithm:

1) Simplicity: It is easy to implement and understand, making it a popular choice for beginners and practical applications.

2) Efficiency: The algorithm is computationally efficient, with linear time complexity relative to the number of data points, making it suitable for large datasets.

3) Speed: It converges quickly compared to other clustering algorithms, particularly when clusters are well-separated.

Limitation of K-Means Algorithm:

1) Sensitivity to Initial Centroids: The algorithm's outcome depends on the initial placement of centroids, which can lead to suboptimal clustering results.

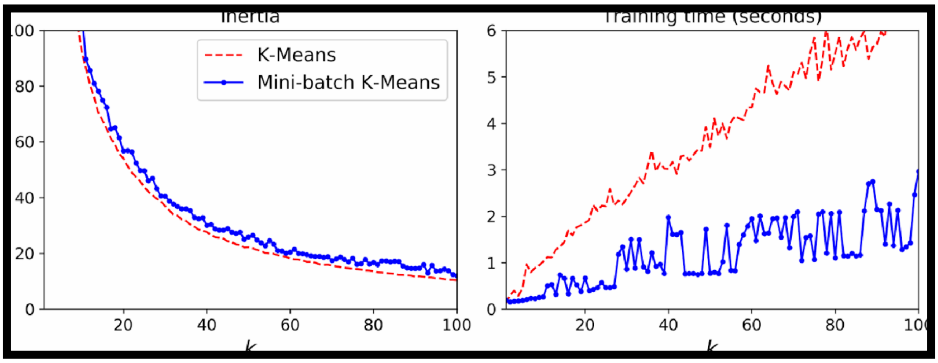
2) Predefined Number of Clusters: It requires the number of clusters (K) to be specified beforehand, which may not always be known or straightforward to determine.

3) Poor Performance with Irregular Clusters: K-Means struggles with non-spherical or overlapping clusters and is unsuitable for datasets with such characteristics.

Important variant of the K-Means algorithm was proposed in a 2010 paper by David Sculley. Instead of using the full dataset at each iteration, the algorithm is capable of using mini-batches, moving the centroids just slightly at each iteration. This speeds up the algorithm typically by a factor of three or four and makes it possible to cluster huge datasets that do not fit in memory. Scikit-Learn implements this algorithm in the Mini-Batch K-Means class. You can just use this class like the K-Means class:

Although the Mini-batch K-Means algorithm is much faster than the regular K-Means algorithm, its inertia is generally slightly worse, especially as the number of clusters increases. You can see this in Figure the plot on the left compares the inertias of Mini-batch K-Means and regular K-Means models trained on the previous dataset using various numbers of clusters k. The difference between the two curves remains fairly constant, but this difference becomes more and more significant as k increases, since

the inertia becomes smaller and smaller. In the plot on the right, you can see that Mini-batch K-Means is much faster than regular K-Means, and this difference increases with k.



Exercise:

1. Develop a K-Means algorithm program to cluster the dataset provided on the google classroom (housing.csv). Suppose 3 randomly chosen clusters. Apply the elbow method and silhouette score to evaluate the number of clusters.

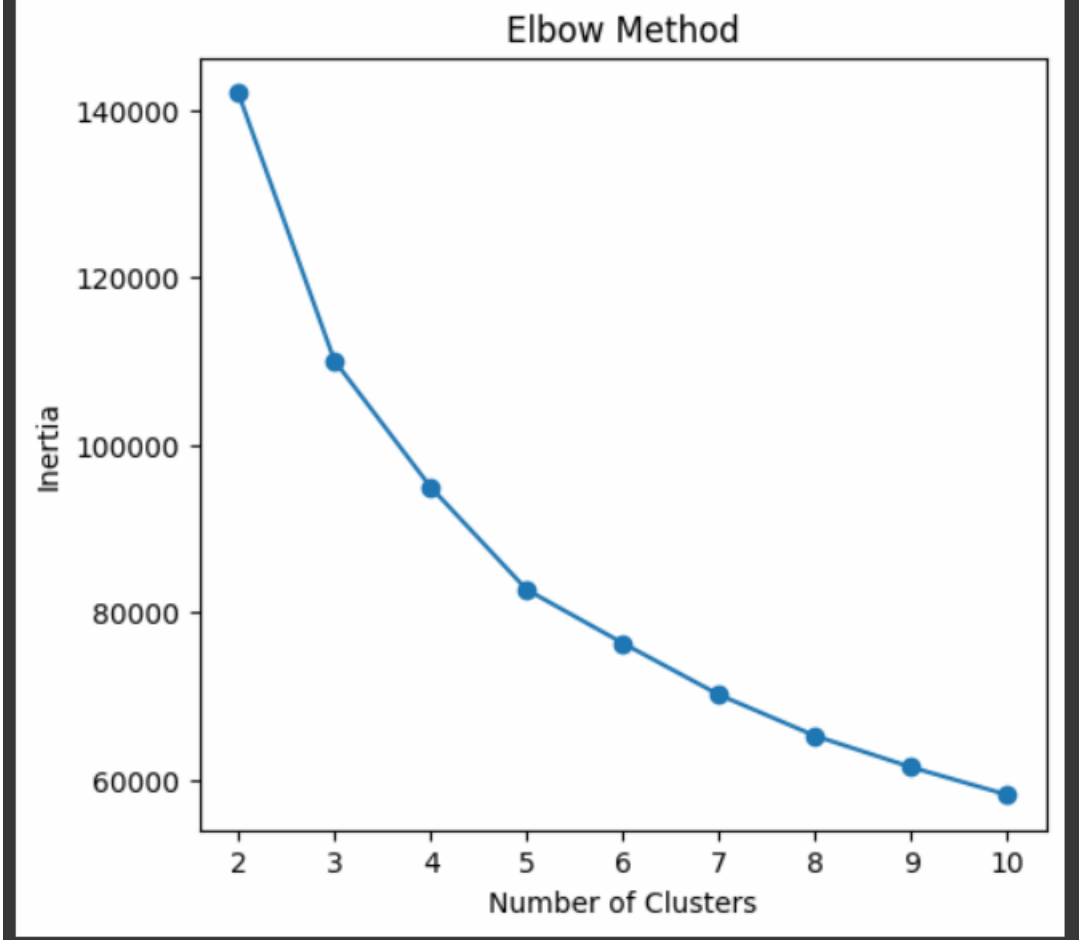
```
[2] import pandas as pd
    import numpy as np
    import matplotlib.pyplot as plt
    from sklearn.impute import SimpleImputer
    from sklearn.preprocessing import StandardScaler
    from sklearn.cluster import KMeans
    from sklearn.metrics import silhouette_score
```

```
df = pd.read_csv("/content/housing.csv")
df.head(5)
```

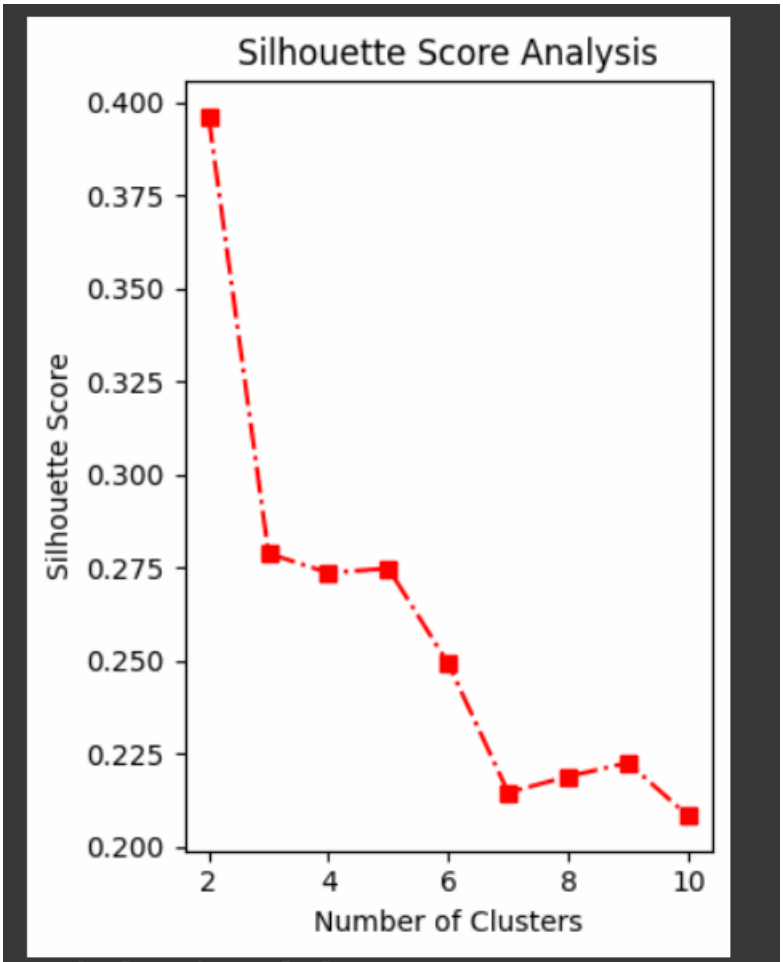
	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	ocean_proximi
0	-122.23	37.88	41	880	129.0	322	126	8.3252	452600	NEAR BA
1	-122.22	37.86	21	7099	1106.0	2401	1138	8.3014	358500	NEAR BA
2	-122.24	37.85	52	1467	190.0	496	177	7.2574	352100	NEAR BA
3	-122.25	37.85	52	1274	235.0	558	219	5.6431	341300	NEAR BA
4	-122.25	37.85	52	1627	280.0	565	259	3.8462	342200	NEAR BA

```
df_numeric = df.drop(columns=['ocean_proximity'], errors='ignore')

imputer = SimpleImputer(strategy="mean")
df_imputed = pd.DataFrame(imputer.fit_transform(df_numeric), columns=df_numeric.columns)

scaler = StandardScaler()
df_scaled = scaler.fit_transform(df_imputed)

kmeans = KMeans(n_clusters=3, random_state=42, n_init=10)
df_imputed['Cluster'] = kmeans.fit_predict(df_scaled)

inertia = []
silhouette_scores = []
k_values = range(2, 11)

for k in k_values:
    kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
    labels = kmeans.fit_predict(df_scaled)
    inertia.append(kmeans.inertia_)
    silhouette_scores.append(silhouette_score(df_scaled, labels))
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(k_values, inertia, marker='o', linestyle='-')
plt.xlabel('Number of Clusters')
plt.ylabel('Inertia')
plt.title('Elbow Method')
```

```
Text(0.5, 1.0, 'Elbow Method')
```



```
plt.subplot(1, 2, 2)
plt.plot(k_values, silhouette_scores, marker='s', linestyle='-.', color='red')
plt.xlabel('Number of Clusters')
plt.ylabel('Silhouette Score')
plt.title('Silhouette Score Analysis')

plt.tight_layout()
plt.show()

optimal_k = k_values[np.argmax(silhouette_scores)]
print(f"Optimal number of clusters: {optimal_k}")
```

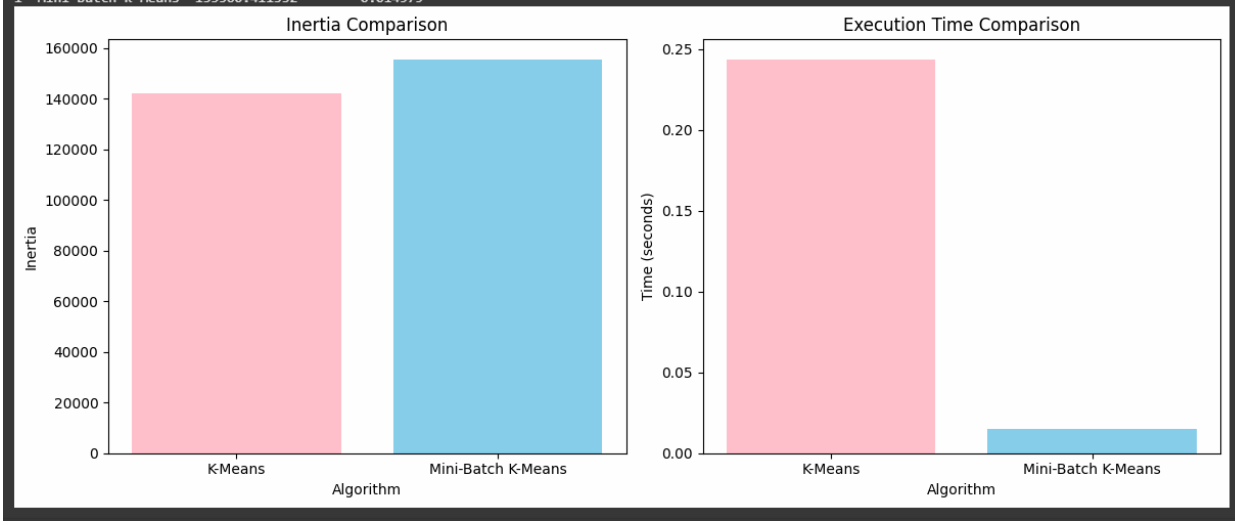



```
Optimal number of clusters: 2
```

2. Improve the results of K-Means algorithm by applying mini-batch K-Means to the cluster the similar dataset as mentioned above. Plot two graphs of performance and inertia for both techniques

```python
from sklearn.cluster import MiniBatchKMeans
import time

start_time = time.time()
kmeans = KMeans(n_clusters=optimal_k, random_state=42, n_init=10)
kmeans.fit(df_scaled)
kmeans_time = time.time() - start_time

start_time = time.time()
mini_kmeans = MiniBatchKMeans(n_clusters=optimal_k, random_state=42, batch_size=100)
mini_kmeans.fit(df_scaled)
mini_kmeans_time = time.time() - start_time

performance_data = {
    "Algorithm": ["K-Means", "Mini-Batch K-Means"],
    "Inertia": [kmeans.inertia_, mini_kmeans.inertia_],
    "Time (seconds)": [kmeans_time, mini_kmeans_time],
}

performance_df = pd.DataFrame(performance_data)
print(performance_df)

plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
plt.bar(["K-Means", "Mini-Batch K-Means"], [kmeans.inertia_, mini_kmeans.inertia_], color=['pink', 'skyblue'])
plt.xlabel("Algorithm")
plt.ylabel("Inertia")
plt.title("Inertia Comparison")

plt.subplot(1, 2, 2)
plt.bar(["K-Means", "Mini-Batch K-Means"], [kmeans_time, mini_kmeans_time], color=['pink', 'skyblue'])
plt.xlabel("Algorithm")
plt.ylabel("Time (seconds)")
plt.title("Execution Time Comparison")

plt.tight_layout()
plt.show()
```

```
           Algorithm        Inertia  Time (seconds)
0            K-Means  141997.960420        0.243805
1  Mini-Batch K-Means  155560.411332        0.014979
```



3. Discuss the pseudo code of the mini-batch K-Means algorithm

1. Start by randomly initializing k cluster centroids.
2. Iterate until the algorithm converges or reaches the maximum number of iterations:
   a. Select a small random subset of the dataset (mini-batch).
   b. For each point in the mini-batch, assign it to the closest cluster centroid.
   c. Recalculate the centroids for each cluster using the selected mini-batch points.
   d. Update the centroids by applying a weighted moving average based on the new calculations.
3. Output the final cluster centroids and the corresponding labels.

Lab 05 - K-Medoid Clustering

Objective:

The objective of this lab is to provide students with hands-on experience in implementing the K-

Medoids clustering algorithm for unsupervised learning. By the end of this lab, students will be able to preprocess datasets to make them suitable for clustering, apply the K-Medoids algorithm using Python libraries such as scikit-learn or Pyclustering, evaluate clustering performance using metrics like total cost (sum of dissimilarities) and silhouette scores, and visualize clustering results to interpret the grouping of data points effectively.

Tools Required

Anaconda Distribution/ Google Colab/ Pycharm

What is K-Medoid Clustering?

Medoid: A Medoid is a point in the cluster from which the sum of distances to other data points is minimal.

(or)

A Medoid is a point in the cluster from which dissimilarities with all the other points in the clusters are minimal.

K-Medoid is a clustering algorithm used to group data points into clusters based on their similarities. It's similar to K-Means but works differently when choosing the “center” of each cluster. Instead of using the average (mean) as K-Means does, K-Medoid selects actual data points (medoids) as the cluster centers. K-Medoids (also called Partitioning around Medoid) the algorithm was proposed in 1987 by Kaufman and Rousseeuw. A medoid can be defined as a point in the cluster, whose dissimilarities with all the other points in the cluster are minimal.

The dissimilarity of the medoid(Ci) and object(Pi) is calculated by using E = |Pi – Ci|

The cost in K-Medoid algorithm is given as:

$$c = \sum_{Ci} \sum_{Pi \in Ci} |Pi - Ci|$$

Algorithm:

Given the value of k and unlabelled data:

1. Choose k number of random points from the data and assign these k points to k number of clusters. These are the initial medoids.

2. For all the remaining data points, calculate the distance from each medoid and assign it to the cluster with the nearest medoid.

3. Calculate the total cost (Sum of all the distances from all the data points to the medoids)

4. Select a random point as the new medoid and swap it with the previous medoid. Repeat 2 and 3 steps.

5. If the total cost of the new medoid is less than that of the previous medoid, make the new medoid permanent and repeat step 4.

6. If the total cost of the new medoid is greater than the cost of the previous medoid, undo the swap and repeat step 4.

7. The Repetitions have to continue until no change is encountered with new medoids to classify data point.

Code Implementation:

Advantages of K-Medoid Clustering:

Robust to Outliers: K-Medoid uses actual data points as cluster centers, making it is less sensitive to extreme values compared to K-Means.

Interpretable Results: The medoids are real data points, which makes the cluster representatives easier to understand and interpret.

Works with Different Distance Metrics: K-Medoid can use various distance measures (e.g., Manhattan, Euclidean, etc.), making it flexible for different types of data.

Limitations of K-Medoid Clustering:

Computationally Expensive: K-Medoid is slower than K-Means, especially for large datasets, because it computes the distance between all points multiple times.

Not Scalable for Large Data: It doesn't scale well for big datasets due to its high computational cost.

Sensitivity to Initialization: The choice of initial medoids can affect the final clustering outcome, potentially leading to suboptimal clusters.

Exercise:

1. Develop a K-Medoid algorithm program to cluster the dataset provided on the google classroom (housing.csv). Suppose 3 randomly chosen clusters. Apply the elbow method and silhouette score to evaluate the number of clusters. What are the optimal K value using the above two mentioned methods.

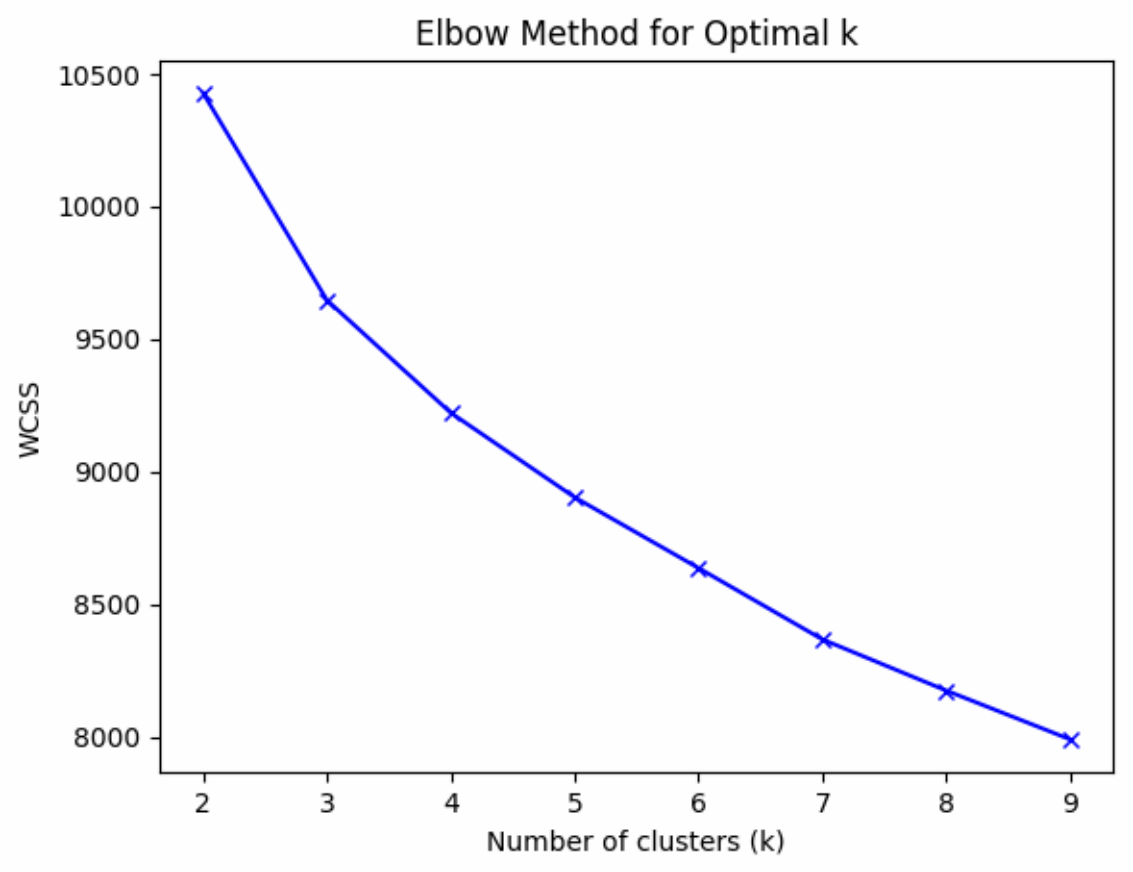
```
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import silhouette_score
from kmedoids import KMedoids
import matplotlib.pyplot as plt
```

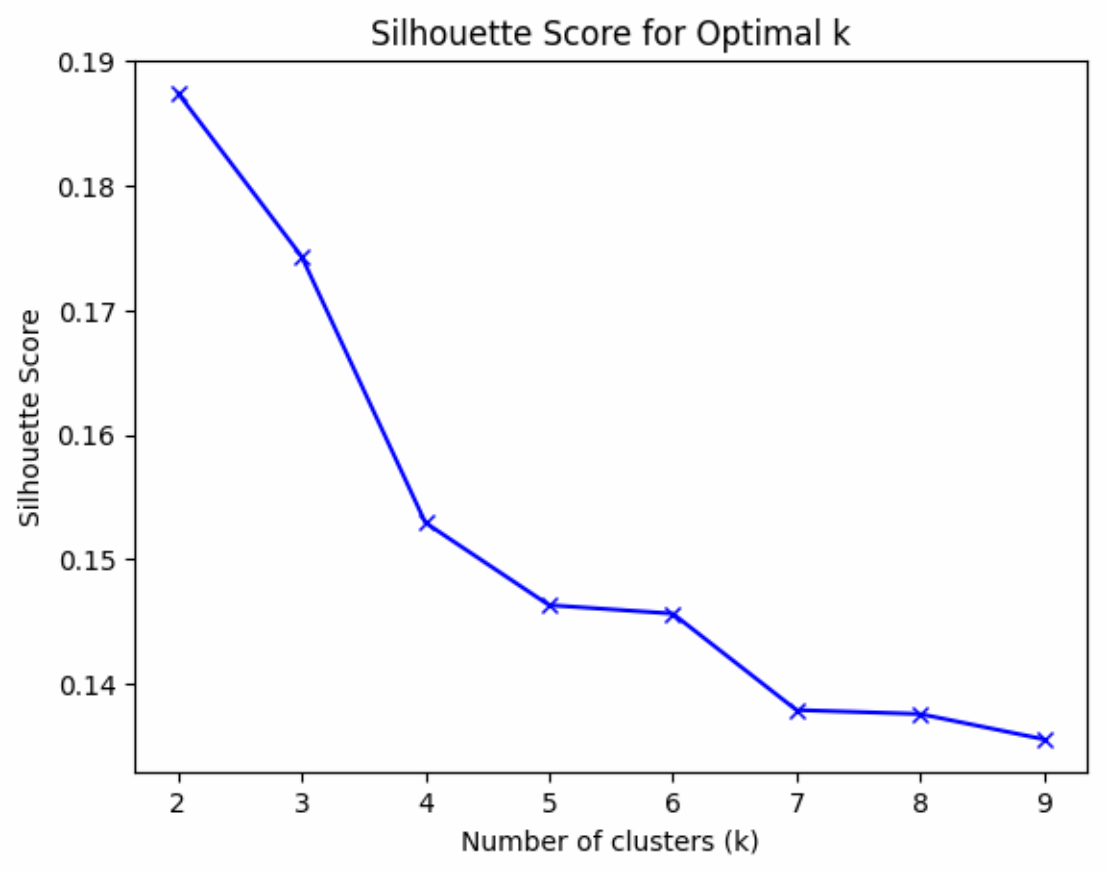
```
# Load the dataset
data = pd.read_csv('/content/housing.csv')
```

```
print(data.dtypes)
```

```
Avg. Area Income              float64
Avg. Area House Age           float64
Avg. Area Number of Rooms     float64
Avg. Area Number of Bedrooms  float64
Area Population               float64
Price                         float64
Address                        object
dtype: object
```

```python
# Ensure k is not greater than number of samples
num_samples = data_scaled.shape[0]
max_k = min(10, num_samples - 1)  # Avoid k > num_samples

# Compute Pairwise Distance Matrix (important for KMedoids)
distance_matrix = pairwise_distances(data_scaled)

def compute_wcss(distance_matrix, k):
    kmedoids = KMedoids(n_clusters=k, metric='precomputed', random_state=42).fit(distance_matrix)
    return kmedoids.inertia_

# Elbow Method
wcss = []
K = range(2, max_k)  # Ensure k is valid
for k in K:
    try:
        wcss.append(compute_wcss(distance_matrix, k))
    except Exception as e:
        print(f"Error at k={k}: {e}")
        break  # Stop if error occurs

 # Plot Elbow Method
 plt.plot(K[:len(wcss)], wcss, 'bx-')
 plt.xlabel('Number of clusters (k)')
 plt.ylabel('WCSS')
 plt.title('Elbow Method for Optimal k')
 plt.show()

 # Silhouette Score
 silhouette_scores = []
 for k in range(2, max_k):
     try:
         kmedoids = KMedoids(n_clusters=k, metric='precomputed', random_state=42).fit(distance_matrix)
         labels = kmedoids.labels_
         silhouette_scores.append(silhouette_score(distance_matrix, labels, metric='precomputed'))
     except Exception as e:
         print(f"Error at k={k}: {e}")
         break  # Stop if error occurs

 # Plot Silhouette Score
 plt.plot(range(2, 2 + len(silhouette_scores)), silhouette_scores, 'bx-')
 plt.xlabel('Number of clusters (k)')
 plt.ylabel('Silhouette Score')
 plt.title('Silhouette Score for Optimal k')
 plt.show()
```


Elbow Method for Optimal k
WCSS
Number of clusters (k)
10500
10000
9500
9000
8500
8000
2
3
4
5
6
7
8
9



Evaluating the Optimal Number of Clusters

To determine the optimal kk, we use:

- **Elbow Method**: Plots Within-Cluster Sum of Squares (WCSS) vs. kk. The "elbow point" indicates the optimal kk.
- **Silhouette Score**: Measures cluster cohesion and separation, ranging from -1 to 1. The optimal kk maximizes this score.

Combining both methods helps in selecting the best kk.

2. Discuss the pseudo code of the K-Medoid algorithm:

K-Medoids Algorithm:

Input: Dataset DD with nn points, number of clusters kk.
Initialize: Randomly select kk medoids.

Repeat:

1. **Assign** each point to the nearest medoid.
2. **Update**: Swap medoids with non-medoids if it reduces clustering cost. 3. **Terminate** when no further swaps improve cost or max iterations reached.

This ensures well-defined, representative clusters.

Lab 6 – DBSCAN Clustering

Objective:

The objective of this lab is to understand and implement the DBSCAN (Density-Based

Spatial Clustering of Applications with Noise) clustering algorithm, which groups data pointsbased on density and identifies noise or outliers effectively. By the end of the lab, students will learn how to apply DBSCAN to real-world datasets, configure its parameters (epsilon and minimum points), and interpret the clustering results. Additionally, students will compare

DBSCAN performance with other clustering algorithms, such as K-Means, to understand its advantages in handling clusters of arbitrary shapes and datasets with noise.

Tools Required:

Anaconda Distribution/ Google Colab/ Pycharm

What is Density-based clustering?

Density-Based Clustering refers to one of the most popular unsupervised learning methodologies used in model building and machine learning algorithms. The data points in

The region separated by two clusters of low point density are considered as noise. The surroundings with a radius ε of a given object are known as the ε neighborhood of the object.

If the ε neighborhood of the object comprises at least a minimum number, MinPts of objects, then it is called a core object

DBSCAN Clustering Algorithm

Clusters are dense regions in the data space, separated by regions of the lower density of points. The DBSCAN algorithm is based on this intuitive notion of “clusters” and “noise”.

The key idea is that for each point of a cluster, the neighborhood of a given radius has to contain at least a minimum number of points.

Why DBSCAN?

Partitioning methods (K-means, PAM clustering) and hierarchical clustering work for finding spherical-shaped clusters or convex clusters. In other words, they are suitable only for compact and well-separated clusters. Moreover, they are also severely affected by the presence of noise and outliers in the data.

Real-life data may contain irregularities, like:

1. Clusters can be of arbitrary shape such as those shown in the figure below.

2. Data may contain noise.

The figure above shows a data set containing non-convex shape clusters and outliers. Given such data, the k-means algorithm has difficulties in identifying these clusters with arbitrary shapes.

Parameters Required For DBSCAN Algorithm

1. eps: It defines the neighborhood around a data point i.e. if the distance between two points are lower or equal to ‘eps’ then they are considered neighbors. If the eps value is chosen too small then a large part of the data will be considered as an outlier. If it is chosen very large then the clusters will merge and the majority of the data points will be in the same clusters. One way to find the eps value is based on the k-distance graph.

2. MinPts: Minimum number of neighbors (data points) within eps radius. The larger the dataset, the larger value of MinPts must be chosen. As a general rule, the minimum MinPts can be derived from the number of dimensions D in the dataset as,

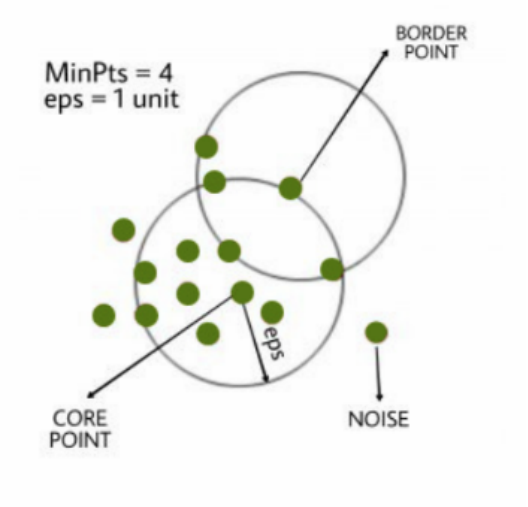
MinPts >= D+1. The minimum value of MinPts must be chosen at least 3.

In this algorithm, we have 3 types of data points.

Core Point: A point is a core point if it has more than MinPts points within eps.

Border Point: A point which has fewer than MinPts within eps but it is in the neighborhood of a core point.

Noise or outlier: A point which is not a core point or border point.



Advantages of DB-SCAN Clustering Algorithm:

The advantages of DBSCAN algorithm is given below:

Detects Arbitrary-Shaped Clusters: DBSCAN can identify clusters of any shape, including non-linear clusters, unlike algorithms like k-means that assume spherical clusters.

Handles Noise and Outliers: Points that do not belong to any cluster are classified as noise, making DBSCAN robust to outliers.

No Predefined Cluster Count Needed: Unlike k-means, DBSCAN does not require specifying the number of clusters beforehand.

Limitations of DB-SCAN Clustering Algorithm:

The limitations of dbscan algorithm is given below:

Sensitive to Parameters (eps and minPts): The results depend heavily on correctly choosing the eps (neighborhood radius) and minPts (minimum points to form a cluster).

Struggles with Varying Densities: DBSCAN has difficulty identifying clusters with different densities, as a single eps value may not fit all.

Exercise:

1. Apply the DBSCAN on the following dataset

a. 1.1 cluster_blob.csv

b. 1.2 cluster_moon.csv

c. 1.3 cluster_circles.csv

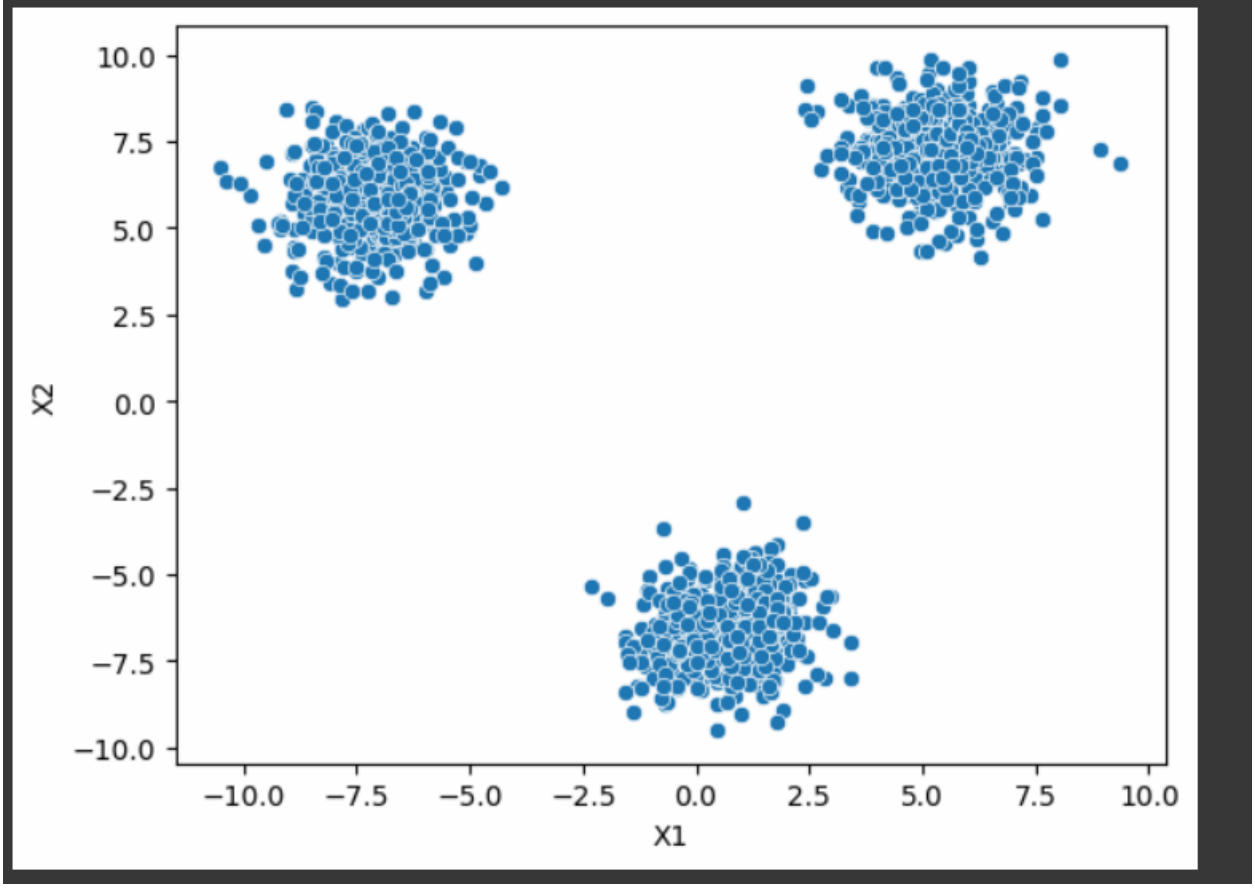
d. 1.4 cluster_two_blob_outliers.csv

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from sklearn.cluster import KMeans
from sklearn.cluster import DBSCAN
from sklearn.neighbors import NearestNeighbors
from sklearn.preprocessing import StandardScaler
```

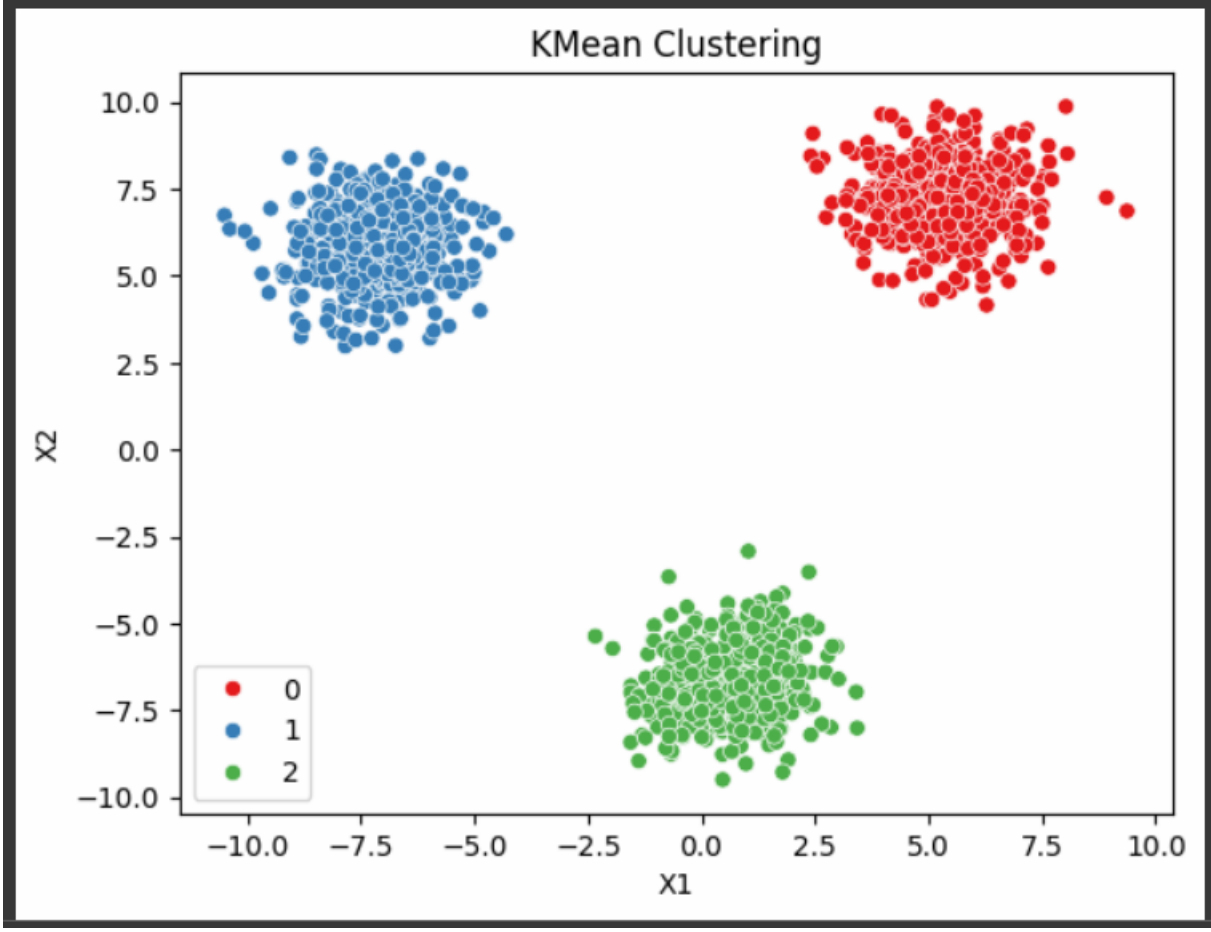
```
blob = pd.read_csv('/content/1.1 cluster_blob.csv')
blob.head()
```

	X1	X2
0	4.645333	6.822294
1	4.784032	6.422883
2	-5.851786	5.774331
3	-7.459592	6.456415
4	4.918911	6.961479

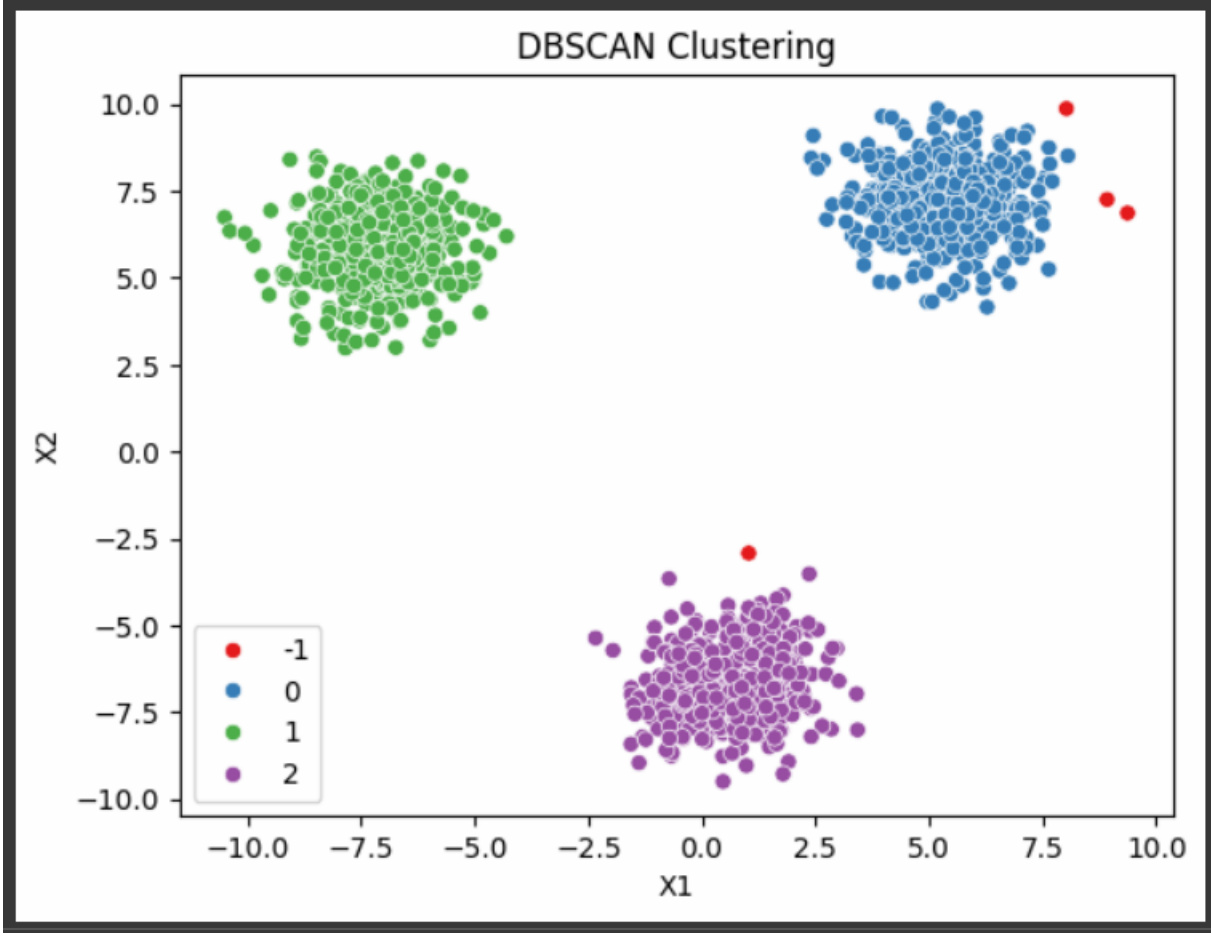
```
sns.scatterplot(data = blob, x = 'X1', y = 'X2')
plt.show()
```



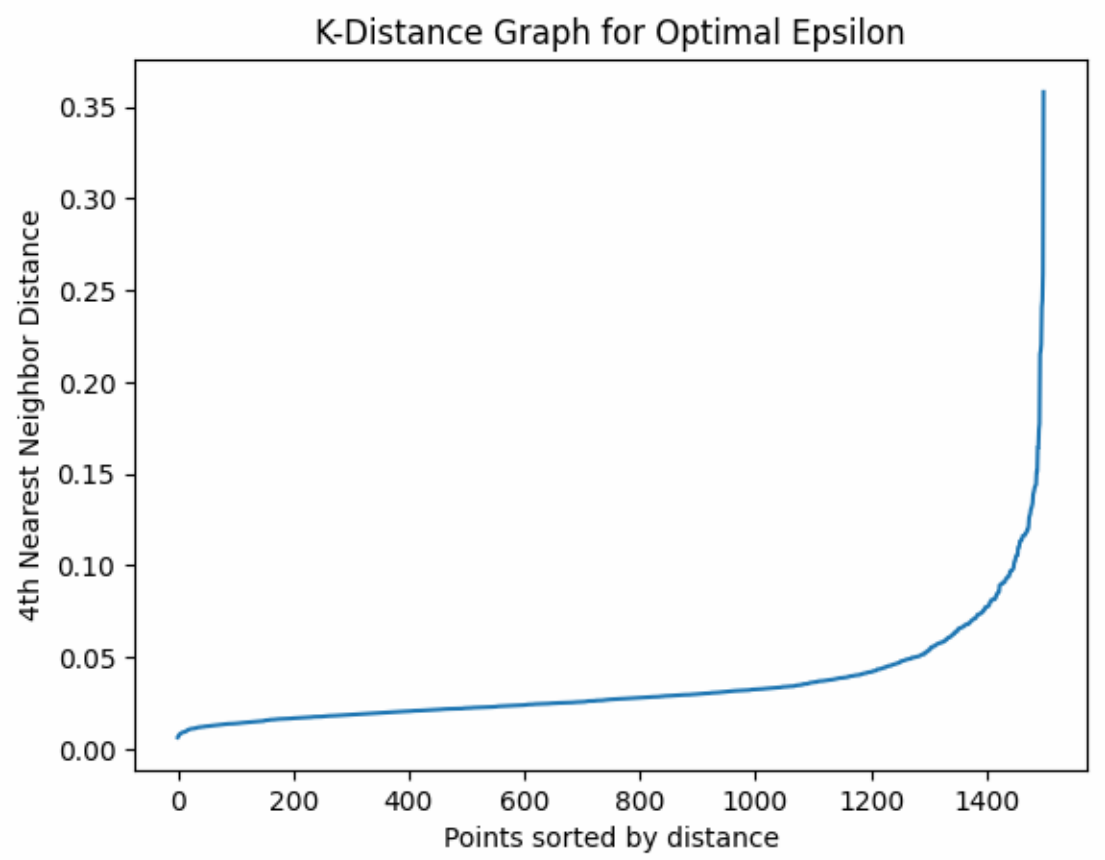
```
model = KMeans(n_clusters=3, random_state=42)
labels = model.fit_predict(blob)

sns.scatterplot(data=blob, x='X1', y='X2', hue=labels, palette="Set1")
plt.title(f"KMean Clustering")
plt.show()
```



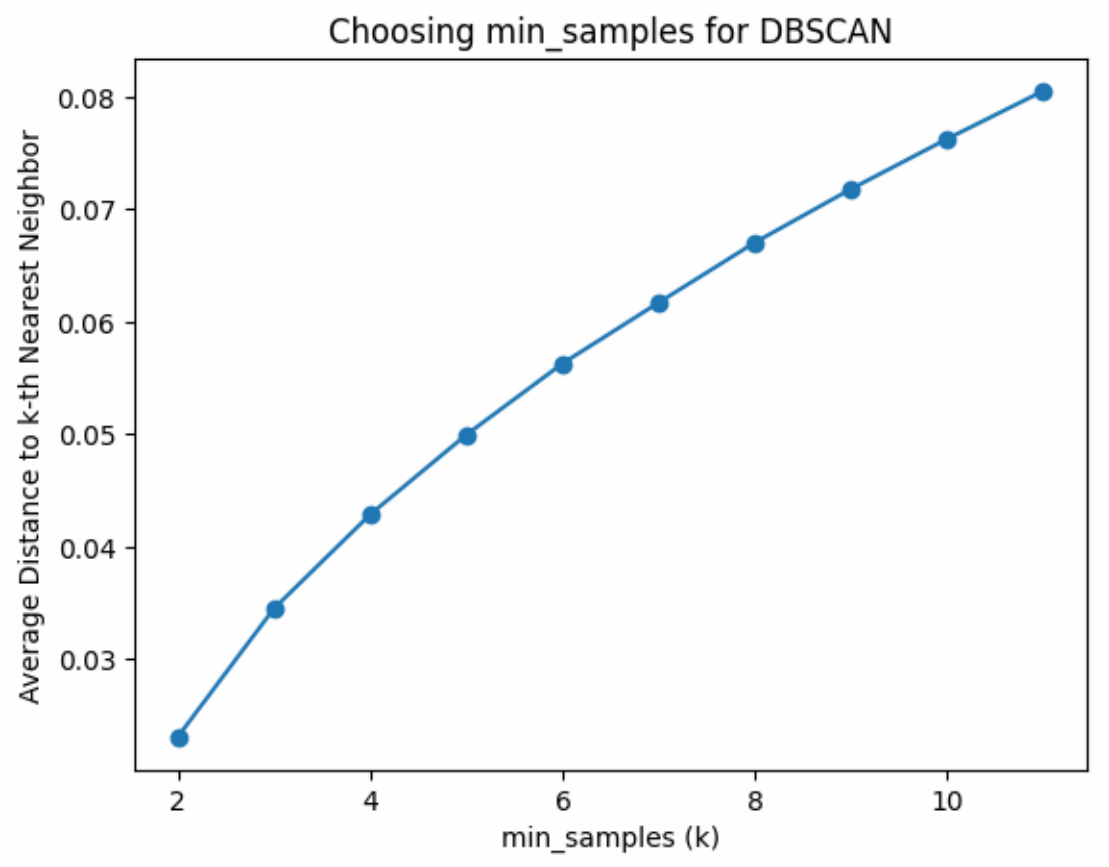
```
model = DBSCAN(eps = 1, min_samples=3)
labels = model.fit_predict(blob)

sns.scatterplot(data=blob, x='X1', y='X2', hue=labels, palette="Set1")
plt.title(f"DBSCAN Clustering")
plt.show()
```



```
scaler = StandardScaler()
data_scaled = scaler.fit_transform(blob)

neighbors = NearestNeighbors(n_neighbors=4)
neighbors_fit = neighbors.fit(data_scaled)
distances, indices = neighbors_fit.kneighbors(data_scaled)

distances = np.sort(distances[:, 3])
plt.plot(distances)
plt.xlabel("Points sorted by distance")
plt.ylabel("4th Nearest Neighbor Distance")
plt.title("K-Distance Graph for Optimal Epsilon")
plt.show()
```



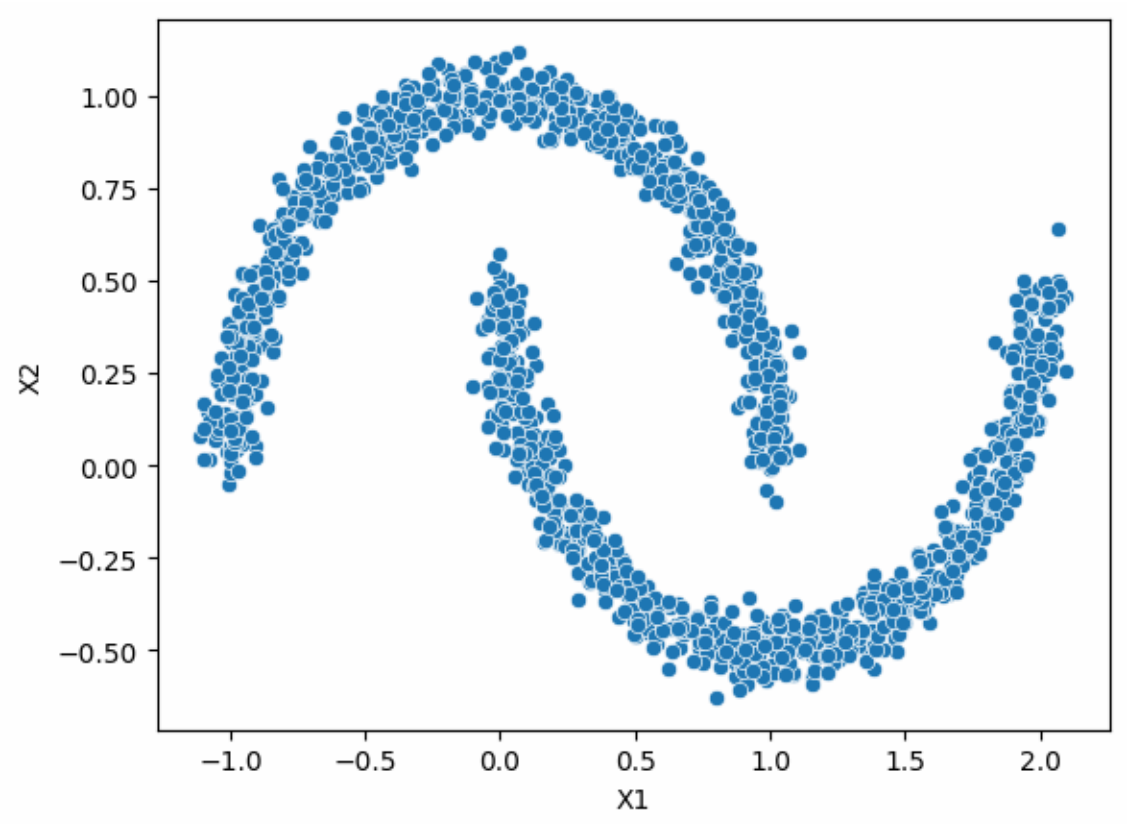
```python
k_values = range(2, 12)
avg_distances = []

for k in k_values:
    neighbors = NearestNeighbors(n_neighbors=k)
    neighbors_fit = neighbors.fit(data_scaled)
    distances, indices = neighbors_fit.kneighbors(data_scaled)
    avg_distances.append(np.mean(distances[:, k-1]))

plt.plot(k_values, avg_distances, marker='o')
plt.xlabel("min_samples (k)")
plt.ylabel("Average Distance to k-th Nearest Neighbor")
plt.title("Choosing min_samples for DBSCAN")
plt.show()
```



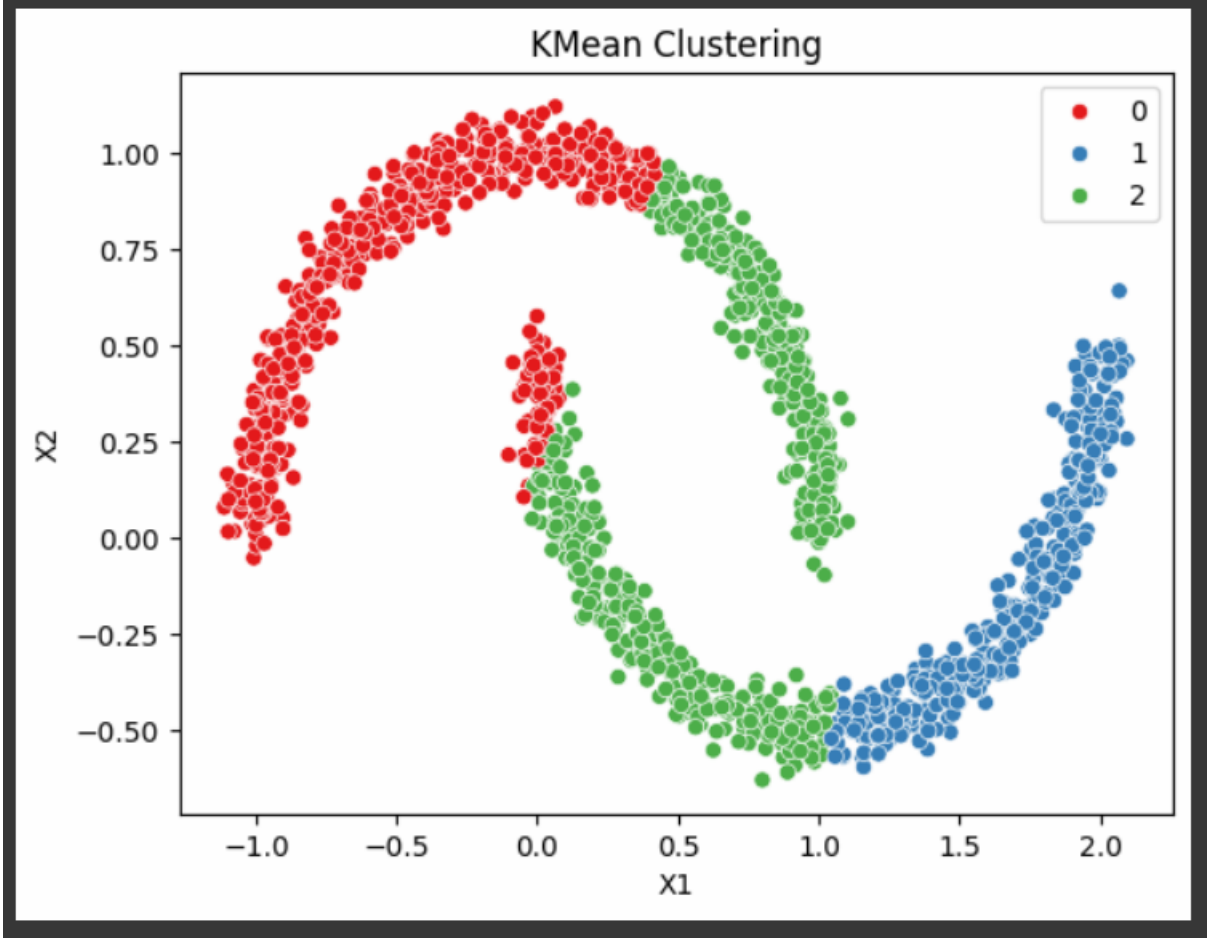
```
moon= pd.read_csv('/content/1.2 cluster_moon.csv')
moon.head()
```

	X1	X2
0	0.674362	-0.444625
1	1.547129	-0.239796
2	1.601930	-0.230792
3	0.014563	0.449752
4	1.503476	-0.389164

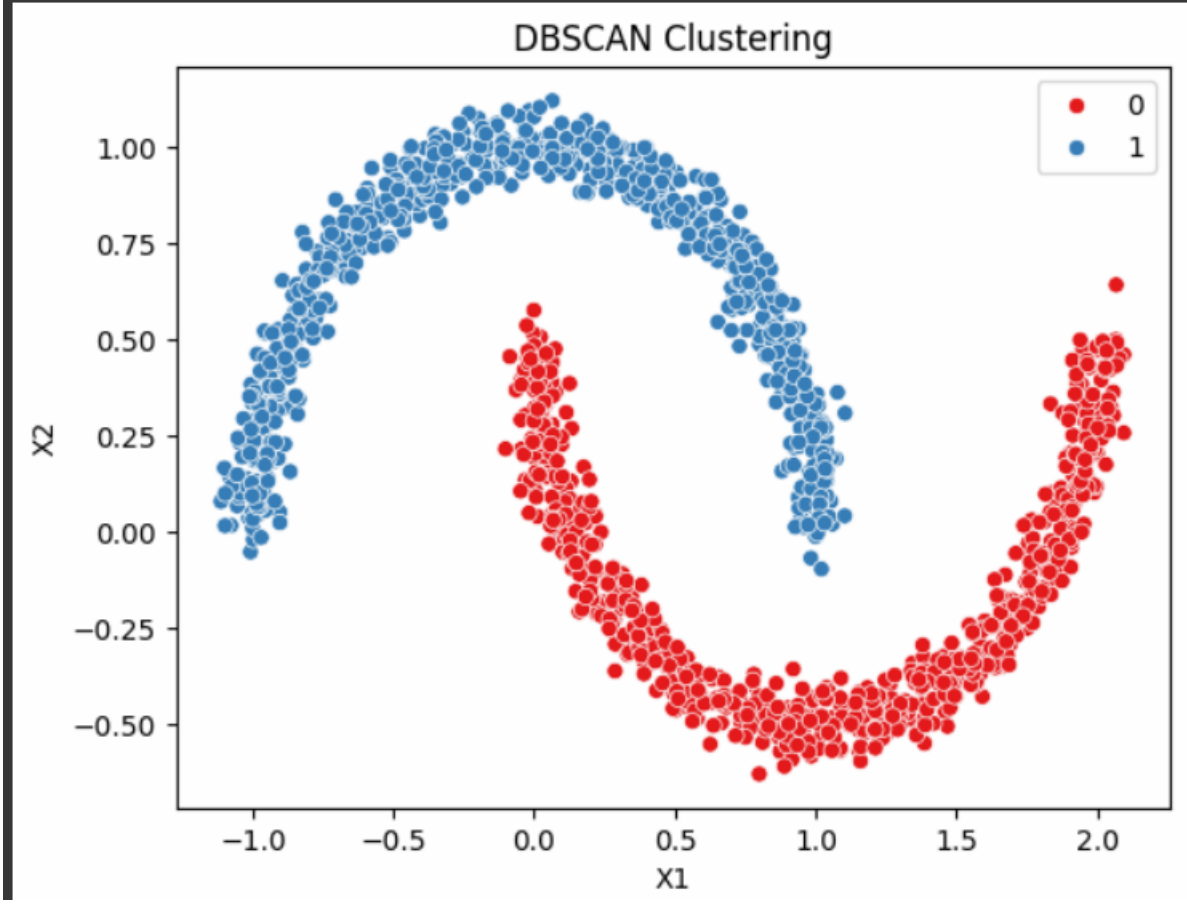
```
sns.scatterplot(data = moon, x = 'X1', y = 'X2')
plt.show()
```


1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
X2
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
X1

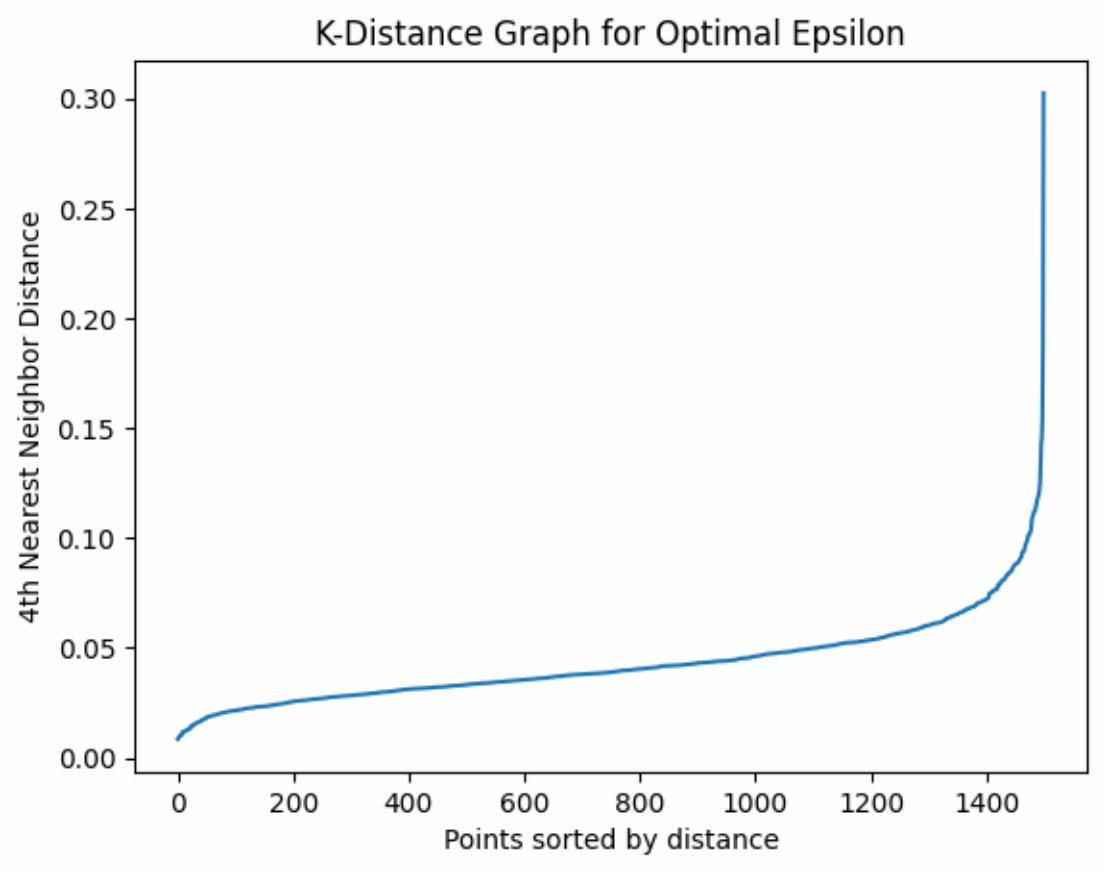
```python
model = KMeans(n_clusters=3, random_state=42)
labels = model.fit_predict(moon)

sns.scatterplot(data=moon, x='X1', y='X2', hue=labels, palette="Set1")
plt.title(f"KMean Clustering")
plt.show()
```



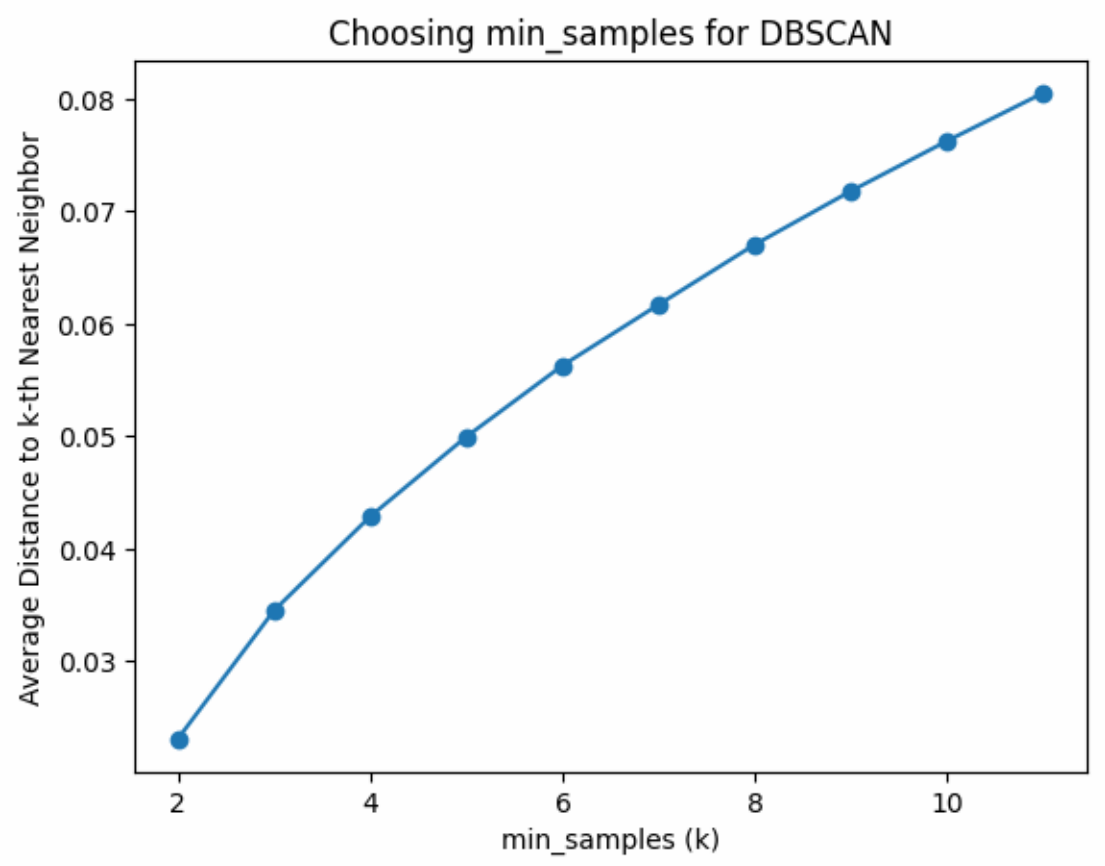
```
model = DBSCAN(eps = 0.15, min_samples= 3)
labels = model.fit_predict(moon)

sns.scatterplot(data=moon, x='X1', y='X2', hue=labels, palette="Set1")
plt.title(f"DBSCAN Clustering")
plt.show()
```



```
scaler = StandardScaler()
data_scaled = scaler.fit_transform(moon)

neighbors = NearestNeighbors(n_neighbors=4)
neighbors_fit = neighbors.fit(data_scaled)
distances, indices = neighbors_fit.kneighbors(data_scaled)

distances = np.sort(distances[:, 3])
plt.plot(distances)
plt.xlabel("Points sorted by distance")
plt.ylabel("4th Nearest Neighbor Distance")
plt.title("K-Distance Graph for Optimal Epsilon")
plt.show()
```


K-Distance Graph for Optimal Epsilon
4th Nearest Neighbor Distance
0.30
0.25
0.20
0.15
0.10
0.05
0.00
0
200
400
600
800
1000
1200
1400
Points sorted by distance

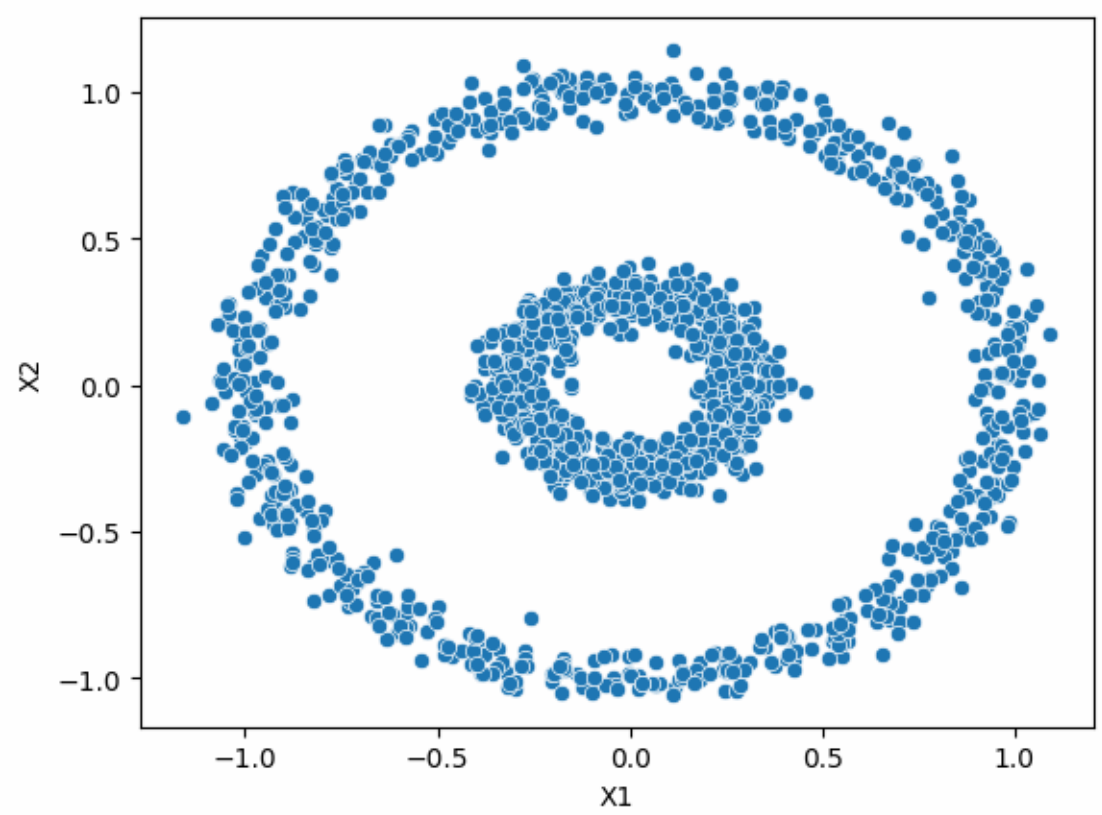
```
k_values = range(2, 12)
avg_distances = []

for k in k_values:
    neighbors = NearestNeighbors(n_neighbors=k)
    neighbors_fit = neighbors.fit(data_scaled)
    distances, indices = neighbors_fit.kneighbors(data_scaled)
    avg_distances.append(np.mean(distances[:, k-1]))

plt.plot(k_values, avg_distances, marker='o')
plt.xlabel("min_samples (k)")
plt.ylabel("Average Distance to k-th Nearest Neighbor")
plt.title("Choosing min_samples for DBSCAN")
plt.show()
```



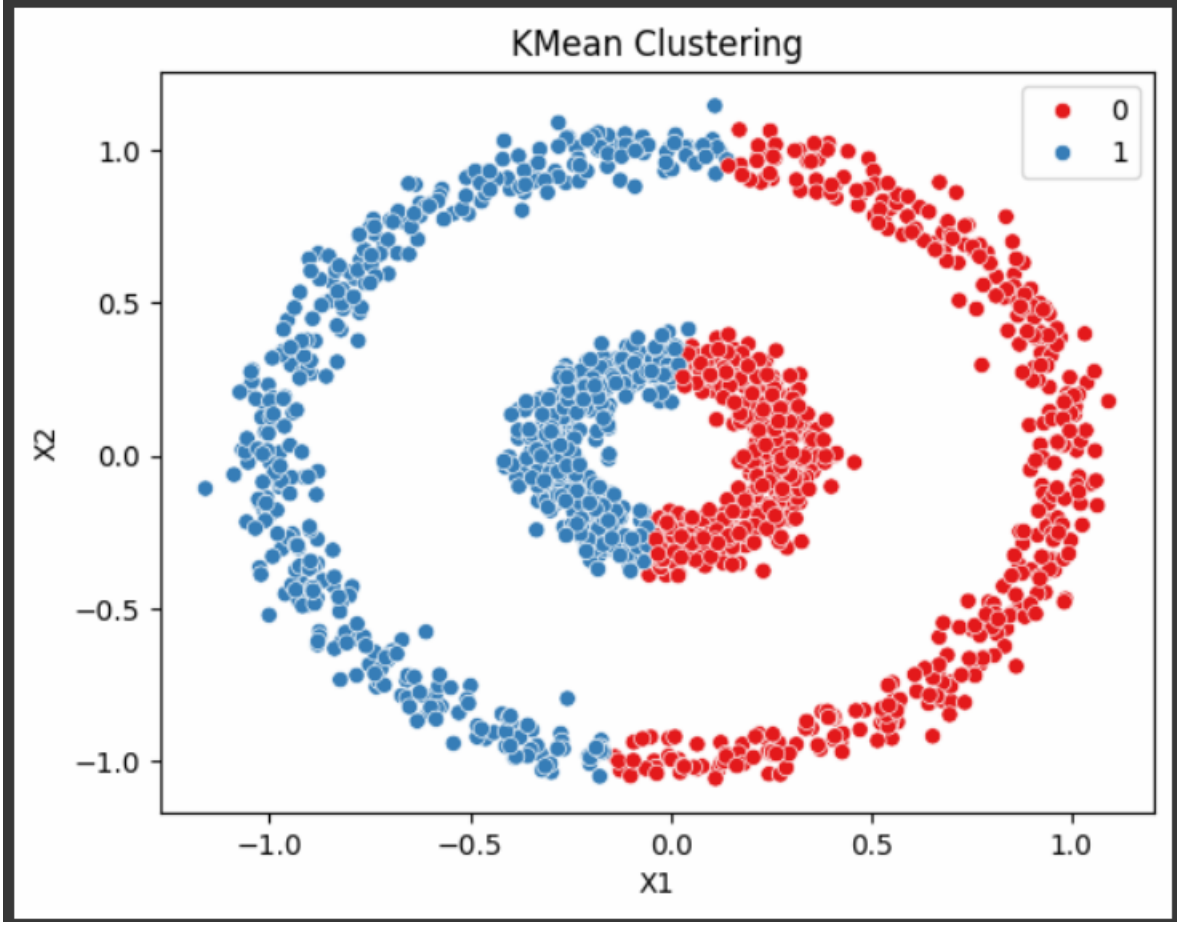

```
circle = pd.read_csv('/content/1.3 cluster_circles.csv')
circle.head()
```

	X1	X2
0	-0.348677	0.010157
1	-0.176587	-0.954283
2	0.301703	-0.113045
3	-0.782889	-0.719468
4	-0.733280	-0.757354

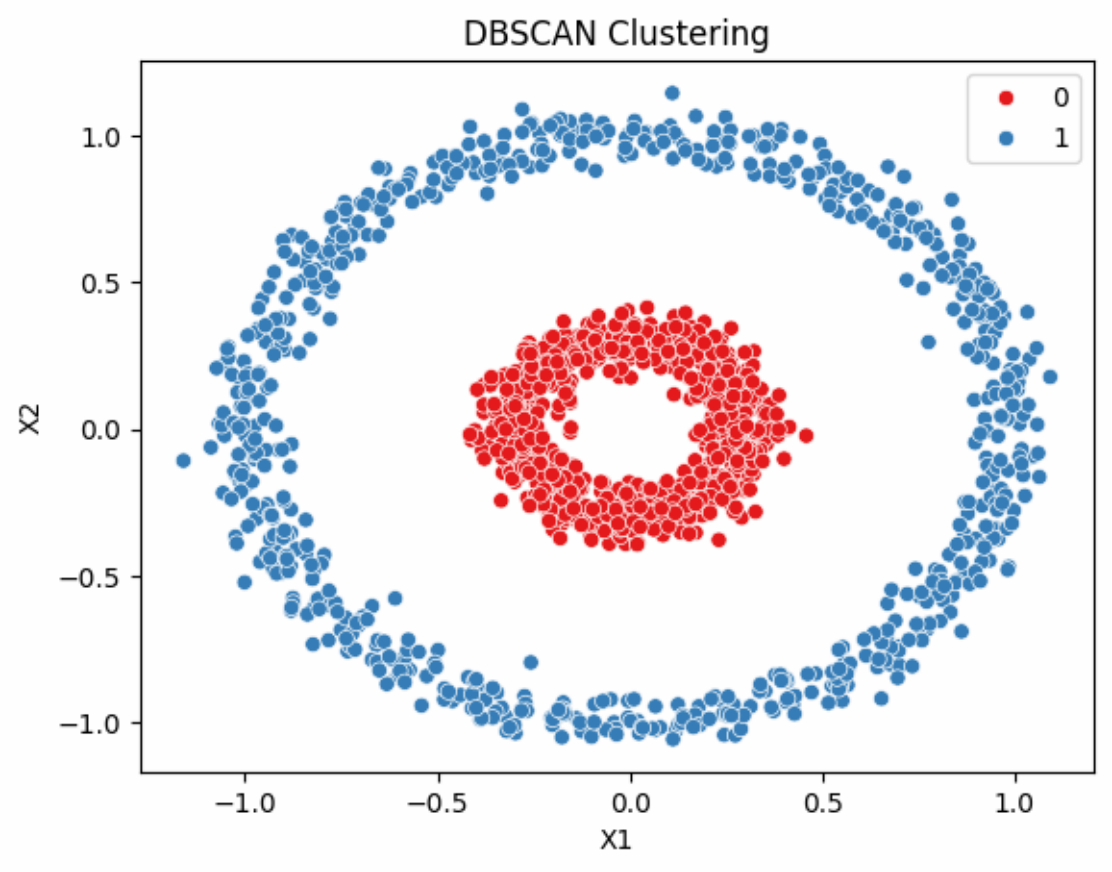
```
sns.scatterplot(data = circle, x = 'X1', y = 'X2')
plt.show()
```


1.0
0.5
0.0
−0.5
−1.0
X2
−1.0
−0.5
0.0
0.5
1.0
X1

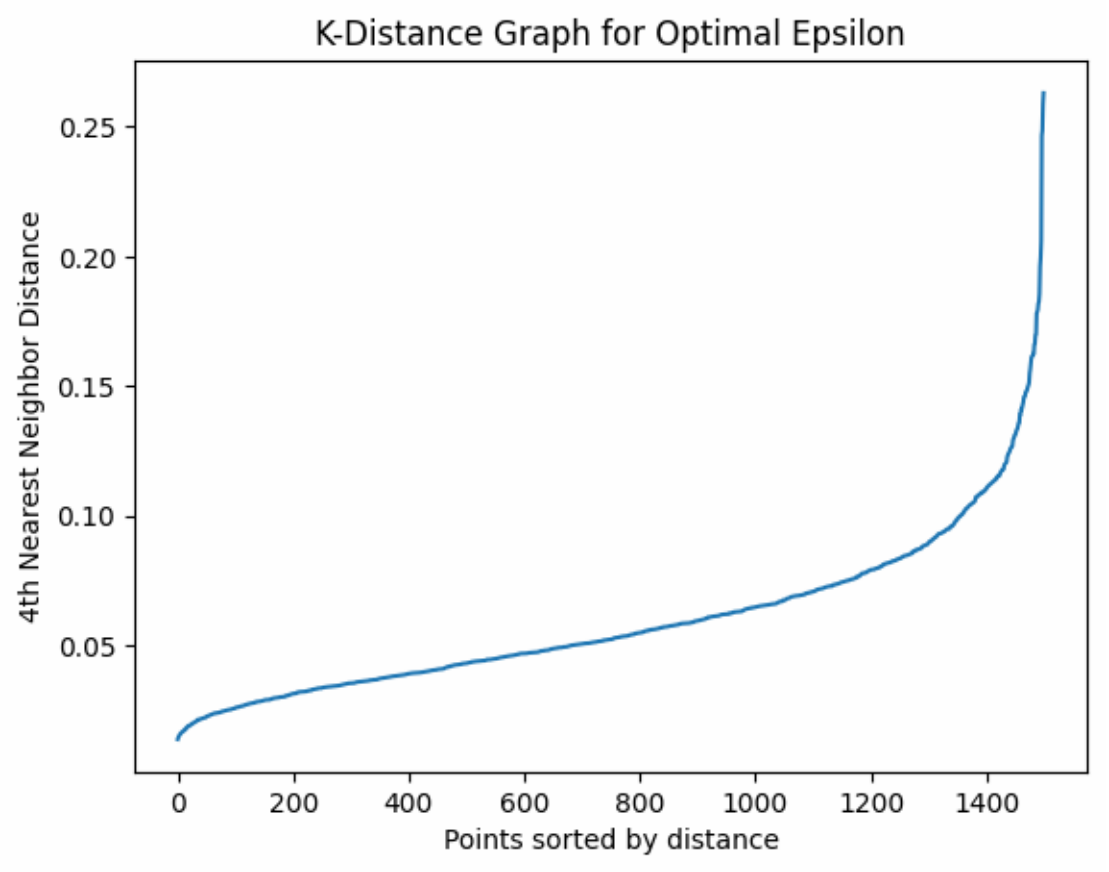
```
model = KMeans(n_clusters=2, random_state=42)
labels = model.fit_predict(circle)

sns.scatterplot(data=circle, x='X1', y='X2', hue=labels, palette="Set1")
plt.title(f"KMean Clustering")
plt.show()
```



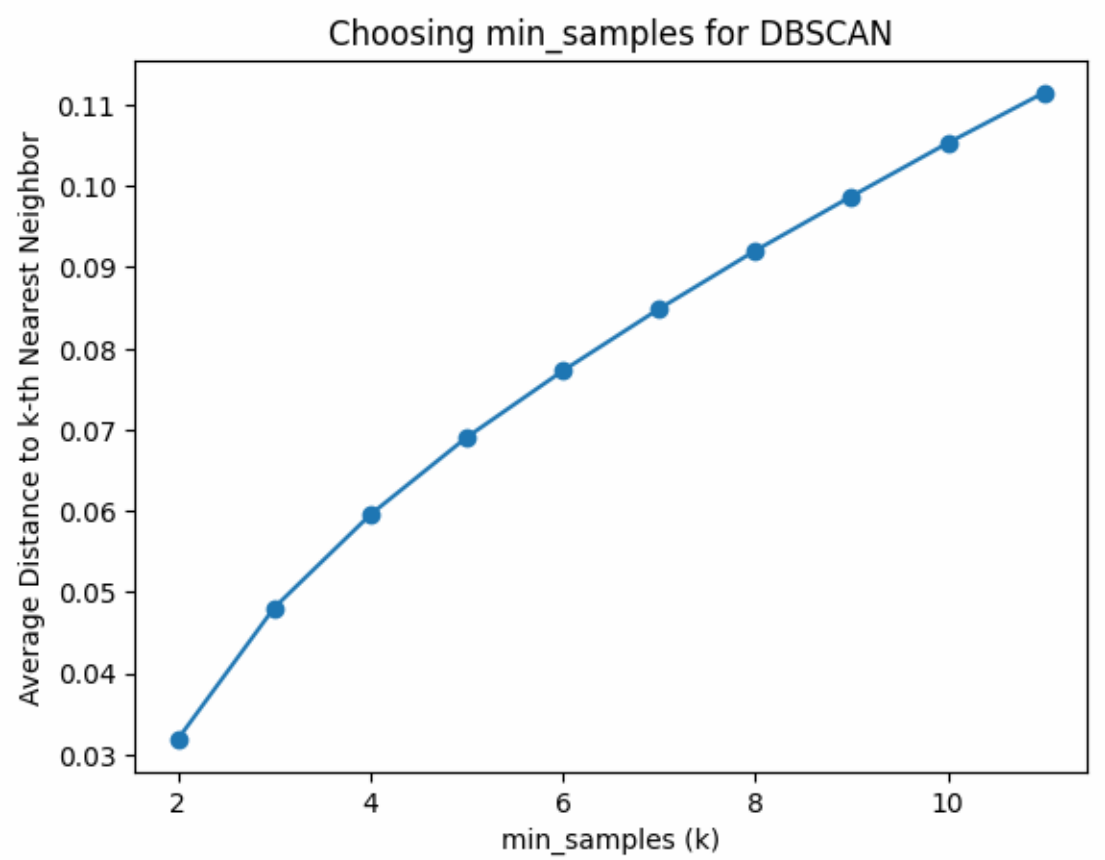
```
model = DBSCAN(eps = 0.15, min_samples= 3)
labels = model.fit_predict(circle)

sns.scatterplot(data=circle, x='X1', y='X2', hue=labels, palette="Set1")
plt.title(f"DBSCAN Clustering")
plt.show()
```


DBSCAN Clustering
0
1
1.0
0.5
0.0
−0.5
−1.0
X2
−1.0
−0.5
0.0
0.5
1.0
X1

```
scaler = StandardScaler()
data_scaled = scaler.fit_transform(circle)

neighbors = NearestNeighbors(n_neighbors=4)
neighbors_fit = neighbors.fit(data_scaled)
distances, indices = neighbors_fit.kneighbors(data_scaled)

distances = np.sort(distances[:, 3])
plt.plot(distances)
plt.xlabel("Points sorted by distance")
plt.ylabel("4th Nearest Neighbor Distance")
plt.title("K-Distance Graph for Optimal Epsilon")
plt.show()
```


K-Distance Graph for Optimal Epsilon
4th Nearest Neighbor Distance
0.25
0.20
0.15
0.10
0.05
0
200
400
600
800
1000
1200
1400
Points sorted by distance

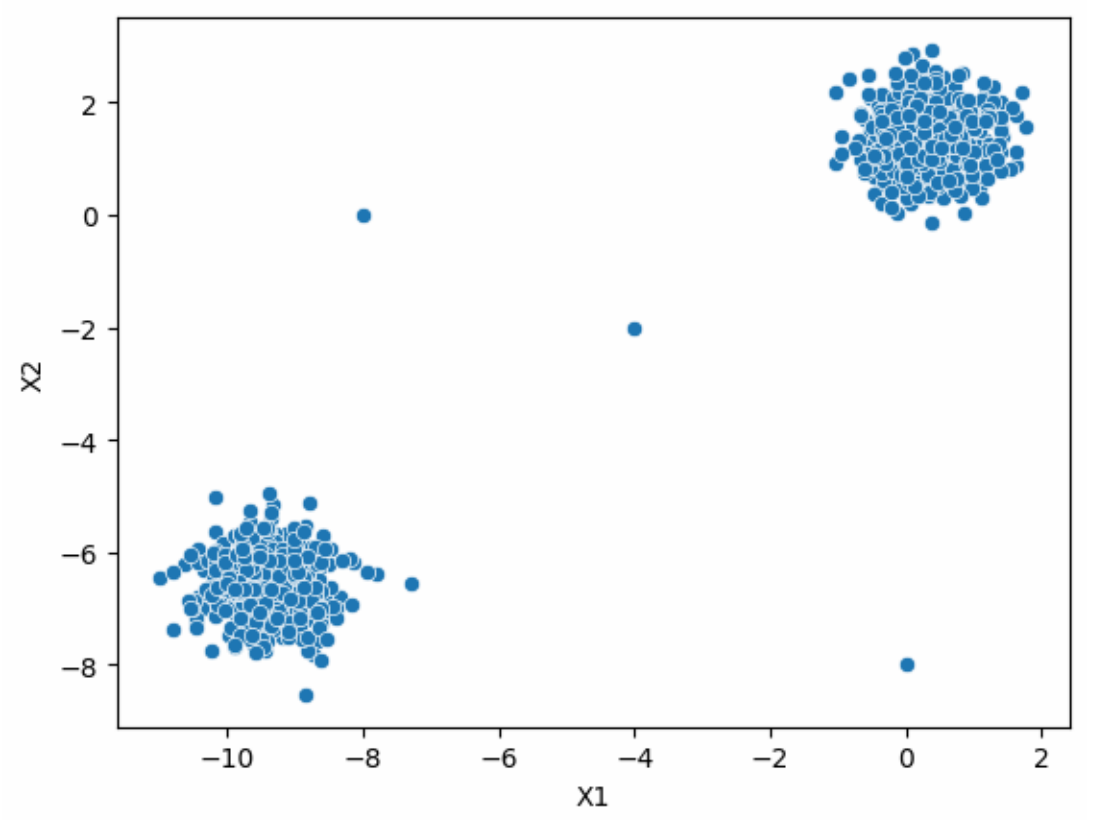
```
k_values = range(2, 12)
avg_distances = []

for k in k_values:
    neighbors = NearestNeighbors(n_neighbors=k)
    neighbors_fit = neighbors.fit(data_scaled)
    distances, indices = neighbors_fit.kneighbors(data_scaled)
    avg_distances.append(np.mean(distances[:, k-1]))

plt.plot(k_values, avg_distances, marker='o')
plt.xlabel("min_samples (k)")
plt.ylabel("Average Distance to k-th Nearest Neighbor")
plt.title("Choosing min_samples for DBSCAN")
plt.show()
```



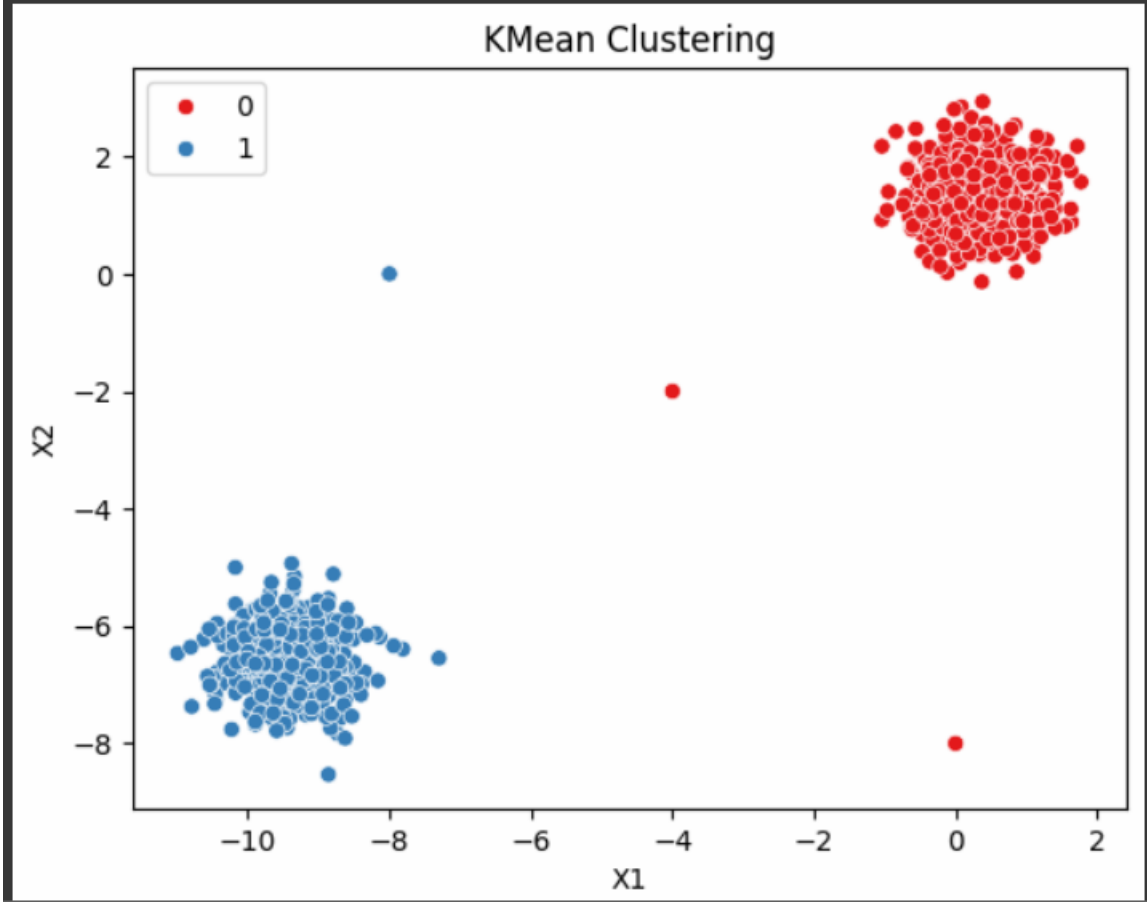

```
two_blob = pd.read_csv('/content/1.4 cluster_two_blob_outliers.csv')
two_blob.head()
```

	X1	X2
0	0.046733	1.765120
1	-8.994134	-6.508186
2	0.650539	1.264533
3	-9.501554	-6.736493
4	0.057050	0.188215

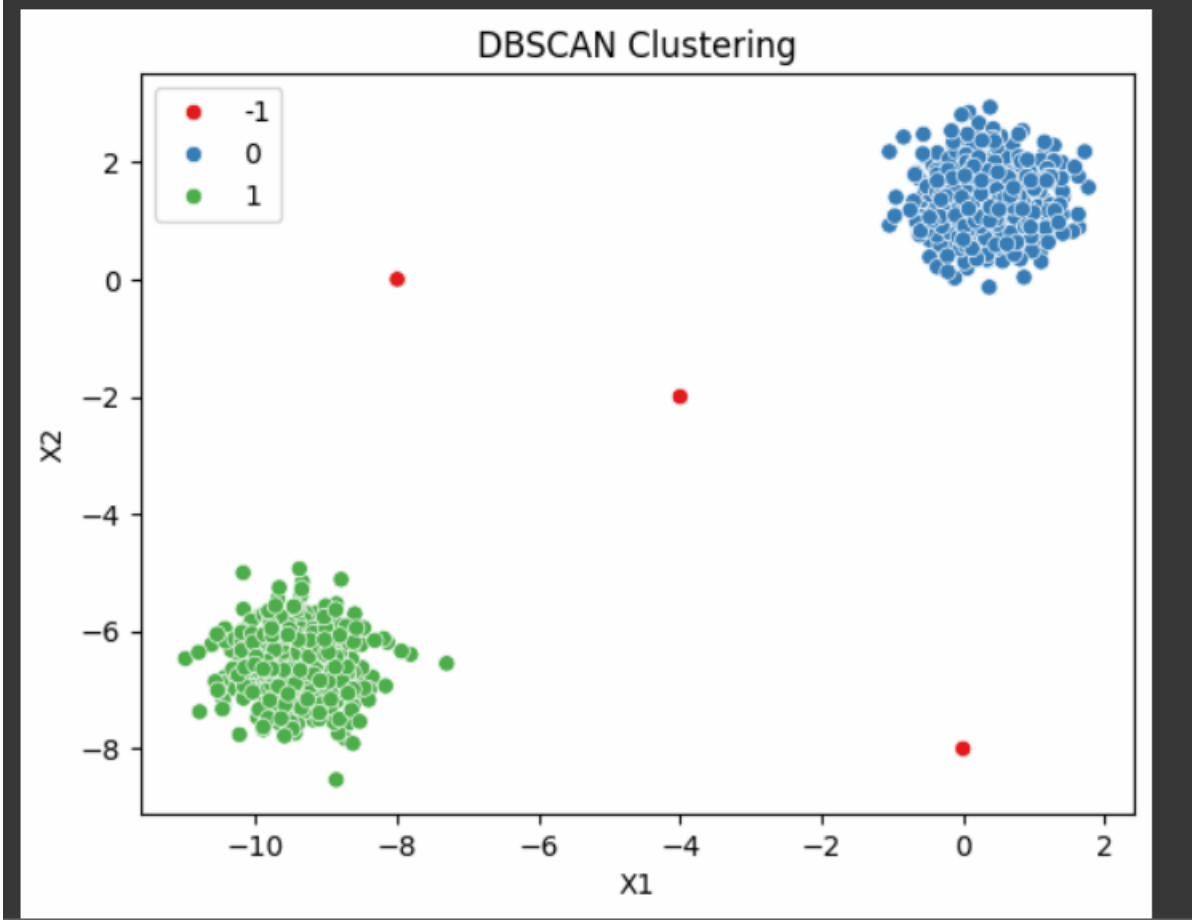
```
sns.scatterplot(data = two_blob, x = 'X1', y = 'X2')
plt.show()
```


2
0
−2
−4
−6
−8
X2
−10
−8
−6
−4
−2
0
2
X1

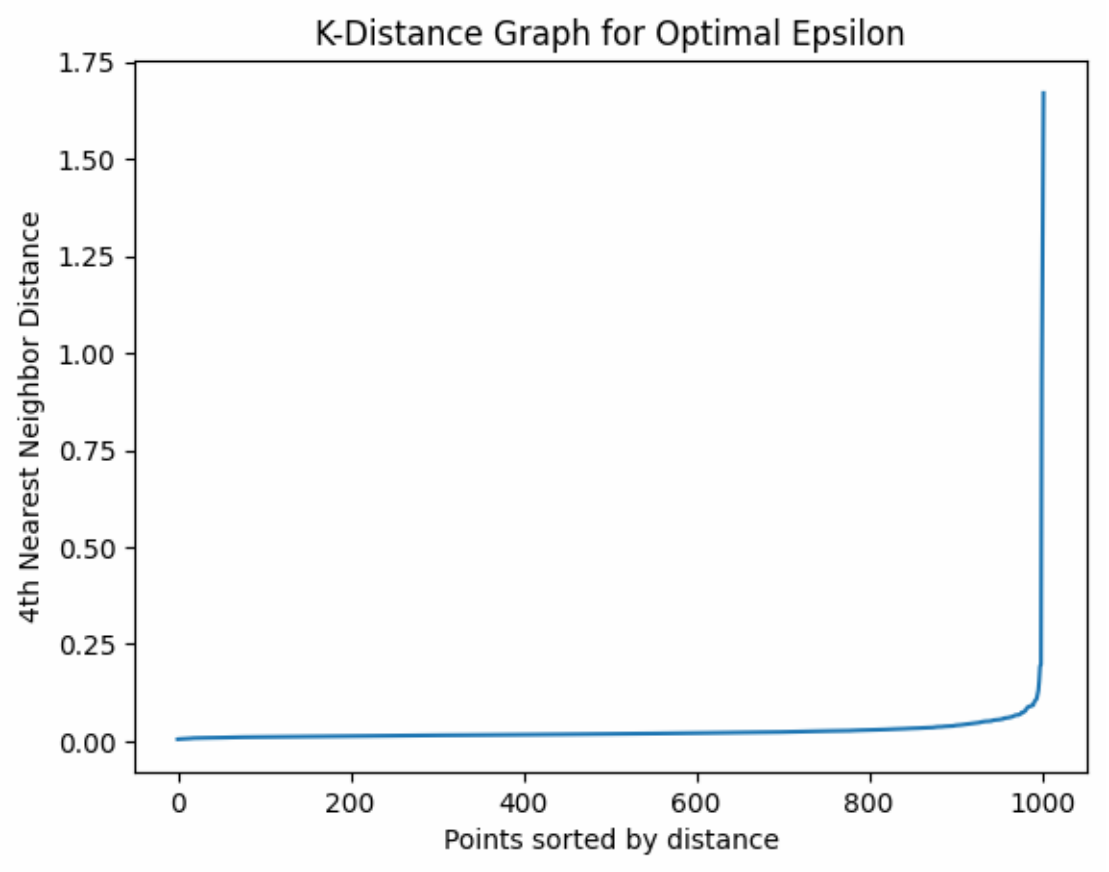
```python
model = KMeans(n_clusters=2, random_state=42)
labels = model.fit_predict(two_blob)

sns.scatterplot(data=two_blob, x='X1', y='X2', hue=labels, palette="Set1")
plt.title(f"KMean Clustering")
plt.show()
```



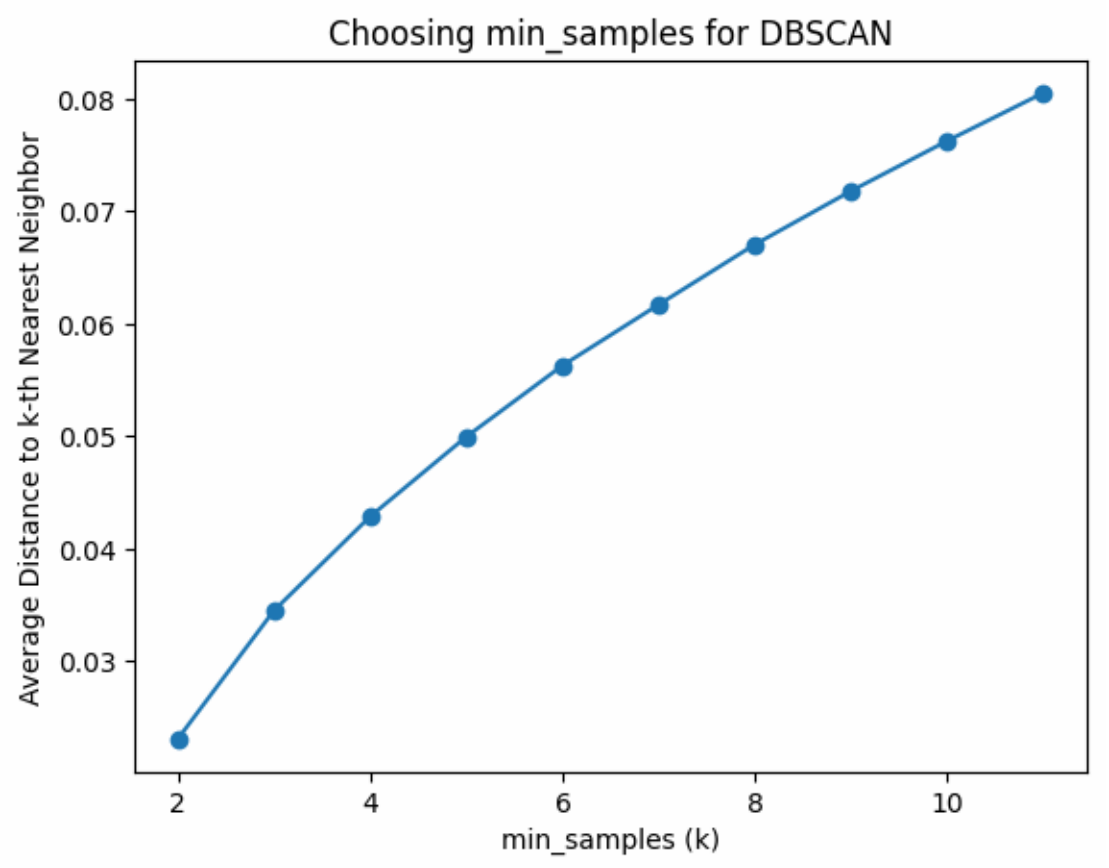
```
model = DBSCAN(eps = 1, min_samples=3)
labels = model.fit_predict(two_blob)

sns.scatterplot(data=two_blob, x='X1', y='X2', hue=labels, palette="Set1")
plt.title(f"DBSCAN Clustering")
plt.show()
```



```
scaler = StandardScaler()
data_scaled = scaler.fit_transform(two_blob)

neighbors = NearestNeighbors(n_neighbors=4)
neighbors_fit = neighbors.fit(data_scaled)
distances, indices = neighbors_fit.kneighbors(data_scaled)

distances = np.sort(distances[:, 3])
plt.plot(distances)
plt.xlabel("Points sorted by distance")
plt.ylabel("4th Nearest Neighbor Distance")
plt.title("K-Distance Graph for Optimal Epsilon")
plt.show()
```


K-Distance Graph for Optimal Epsilon
4th Nearest Neighbor Distance
Points sorted by distance
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0
200
400
600
800
1000

```
k_values = range(2, 12)
avg_distances = []

for k in k_values:
    neighbors = NearestNeighbors(n_neighbors=k)
    neighbors_fit = neighbors.fit(data_scaled)
    distances, indices = neighbors_fit.kneighbors(data_scaled)
    avg_distances.append(np.mean(distances[:, k-1]))

plt.plot(k_values, avg_distances, marker='o')
plt.xlabel("min_samples (k)")
plt.ylabel("Average Distance to k-th Nearest Neighbor")
plt.title("Choosing min_samples for DBSCAN")
plt.show()
```



2. Using the above datasets apply the hyperparameter optimization technique to get the optimal values of epsilon and min_sample parameters.

Data Set	Epsilon	Min_sample
1.1 cluster_blob.csv	1	3
1.2 cluster_moon.csv	0.15	3
1.3 cluster_circles.csv	0.15	3
1.4 cluster_two_blob_outliers.csv	1	3

3. Write down the at least three use cases of DBSCAN with real-life scenario

Use Case	Real-Life Scenario
Anomaly Detection	Detecting fraudulent transactions in banking systems where outliers (frauds) don't fit normal customer behavior.
Geospatial Clustering	Grouping locations of earthquakes based on proximity to identify seismic zones.
Customer Segmentation	Clustering customers based on shopping behavior for targeted marketing.

Lab 7 – Decision Trees

Objective:

The objective of this lab is to understand the working principles of Decision Trees for classification tasks and implement them. It aims to explore different attribute selection measures, including Information Gain (Entropy), Gain Ratio, and Gini Index, to determine the best feature for splitting at each node. Furthermore, the lab focuses on comparing the performance of different splitting criteria to analyze their impact on classification accuracy. Another key objective is to examine feature importance in decision tree classification and understand how different features contribute to decision-making.

Tools Required:

Anaconda Distribution/ Google Colab/ Pycharm

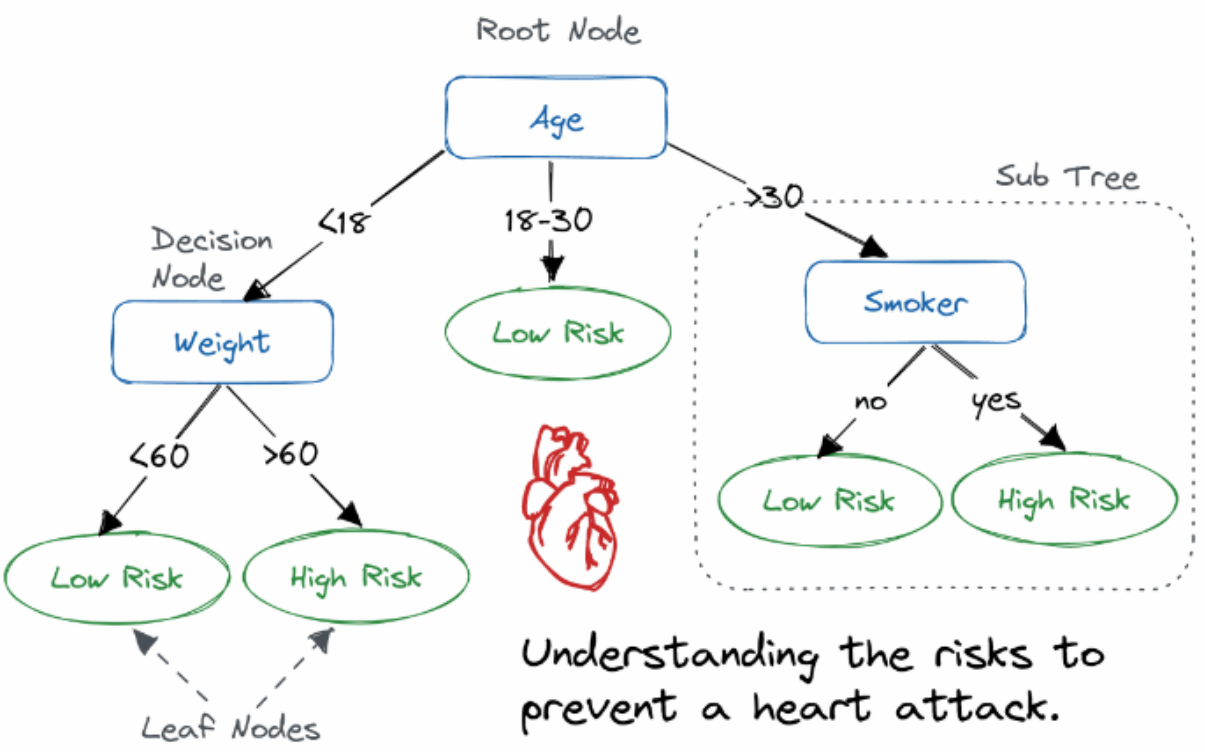
Classification

Classification is a two-step process; a learning step and a prediction step. In the learning step, the model is developed based on given training data. In the prediction step, the model is used to predict the response to given data. A Decision tree is one of the easiest and most popular classification algorithms used to understand and interpret data. It can be utilized for both classification and regression problems.

The Decision Tree Algorithm

A decision tree is a flowchart-like tree structure where an internal node represents a feature(or attribute), the branch represents a decision rule, and each leaf node represents the outcome.

The topmost node in a decision tree is known as the root node. It learns to partition on the basis of the attribute value. It partitions the tree in a recursive manner called recursive partitioning. This flowchart-like structure helps you in decision-making. It's visualized like a flowchart diagram which easily mimics human level thinking. That is why decision trees are easy to understand and interpret.



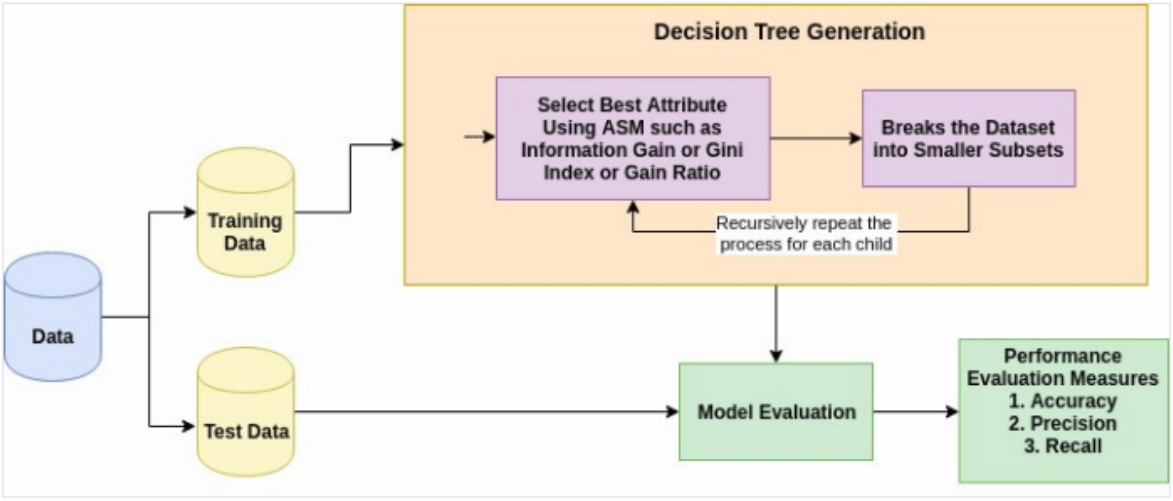
A decision tree is a white box type of ML algorithm. It shares internal decision-making logic, which is not available in the black box type of algorithms such as with a neural network. Its training time is faster compared to the neural network algorithm.

The time complexity of decision trees is a function of the number of records and attributes in the given data. The decision tree is a distribution-free or non-parametric method which does not depend upon probability distribution assumptions. Decision trees can handle high-dimensional data with good accuracy.

How Does the Decision Tree Algorithm Work?

The basic idea behind any decision tree algorithm is as follows:

1. Select the best attribute using Attribute Selection Measures (ASM) to split the records.
2. Make that attribute a decision node and breaks the dataset into smaller subsets.
3. Start tree building by repeating this process recursively for each child until one of the conditions will match:
    - All the tuples belong to the same attribute value.
    - There are no more remaining attributes.
    - There are no more instances.



Attribute Selection Measures

Attribute selection measure is a heuristic for selecting the splitting criterion that partitions data in the best possible manner. It is also known as splitting rules because it helps us to determine breakpoints for tuples on a given node. ASM provides a rank to each feature (or attribute) by explaining the given dataset. The best score attribute will be selected as a splitting attribute. In the case of a continuous-valued attribute, split points for branches also need to define. The most popular selection measures are Information Gain, Gain Ratio, and Gini Index.

Information Gain

Claude Shannon invented the concept of entropy, which measures the impurity of the input set. In physics and mathematics, entropy is referred to as the randomness or the impurity in a system. In information theory, it refers to the impurity in a group of examples. Information gain is the decrease in entropy. Information gain computes the difference between entropy before the split and average entropy after the split of the dataset based on given attribute values. ID3 (Iterative Dichotomiser) decision tree algorithm uses information gain.

$$\text{Info}(D) = -\sum_{i=1}^{m} pi \log_2 pi$$

Where Pi is the probability that an arbitrary tuple in D belongs to class Ci.

$$\text{Info}_A(D)=\sum_{j=1}^{V} \frac{|Dj|}{|D|} \quad \text{X Info}(D_j)$$

$$\text{Gain}(A)=\text{Info}(D)-\text{Info}_A(D)$$

Gain Ratio

Information gain is biased for the attribute with many outcomes. It means it prefers the attribute with a large number of distinct values. For instance, consider an attribute with a unique identifier, such as customer_ID, that has zero info(D) because of pure partition. This maximizes the information gain and creates useless partitioning.

C4.5, an improvement of ID3, uses an extension to information gain known as the gain ratio. Gain ratio handles the issue of bias by normalizing the information gain using Split Info. Java implementation of the C4.5 algorithm is known as J48, which is available in WEKA data mining tool.

$$SplitInfo_A(D) = -\sum_{j=1}^{v} \frac{|D_j|}{|D|} \times \log_2\left(\frac{|D_j|}{|D|}\right)$$

Gini Index

Another decision tree algorithm CART (Classification and Regression Tree) uses the Gini method to create split points.

$$\text{Gini}(D)=1-\sum_{i=1}^{m} \text{Pi}^2$$

Where pi is the probability that a tuple in D belongs to class Ci.

The Gini Index considers a binary split for each attribute. You can compute a weighted sum of the impurity of each partition. If a binary split on attribute A partitions data D into D1 and D2, the Gini index of D is:

$$\text{Gini}_A(D) = \frac{|D1|}{|D|}\text{Gini}(D_1) + \frac{|D2|}{|D|}\text{Gini}(D_2)$$

In the case of a discrete-valued attribute, the subset that gives the minimum gini index for that chosen is selected as a splitting attribute. In the case of continuous-valued attributes, the strategy is to select each pair of adjacent values as a possible split point, and a point with a smaller gini index is chosen as the splitting point.

$$\Delta Gini(A) = Gini(D) - Gini_A(D).$$

The attribute with the minimum Gini index is chosen as the splitting attribute.

Coding Example

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Load the Iris dataset
iris = datasets.load_iris()
X, y = iris.data, iris.target

# Split into training (80%) and testing (20%) sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize and train the Decision Tree classifier
clf = DecisionTreeClassifier(criterion="gini", max_depth=3, random_state=42)
```

```
clf.fit(X_train, y_train)


# Make predictions
y_pred = clf.predict(X_test)


# Compute accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Test Accuracy: {accuracy:.4f}")


# Visualize the decision tree
plt.figure(figsize=(10, 6))
plot_tree(clf, filled=True, feature_names=iris.feature_names,
class_names=iris.target_names)
plt.show()
```

Exercise

1. Apply the Decision Tree classifier on “penguins_size.csv” dataset. Set the random_state = 42 and compare the model accuracy with **gini index and entropy** criteria. Which criteria performs best?

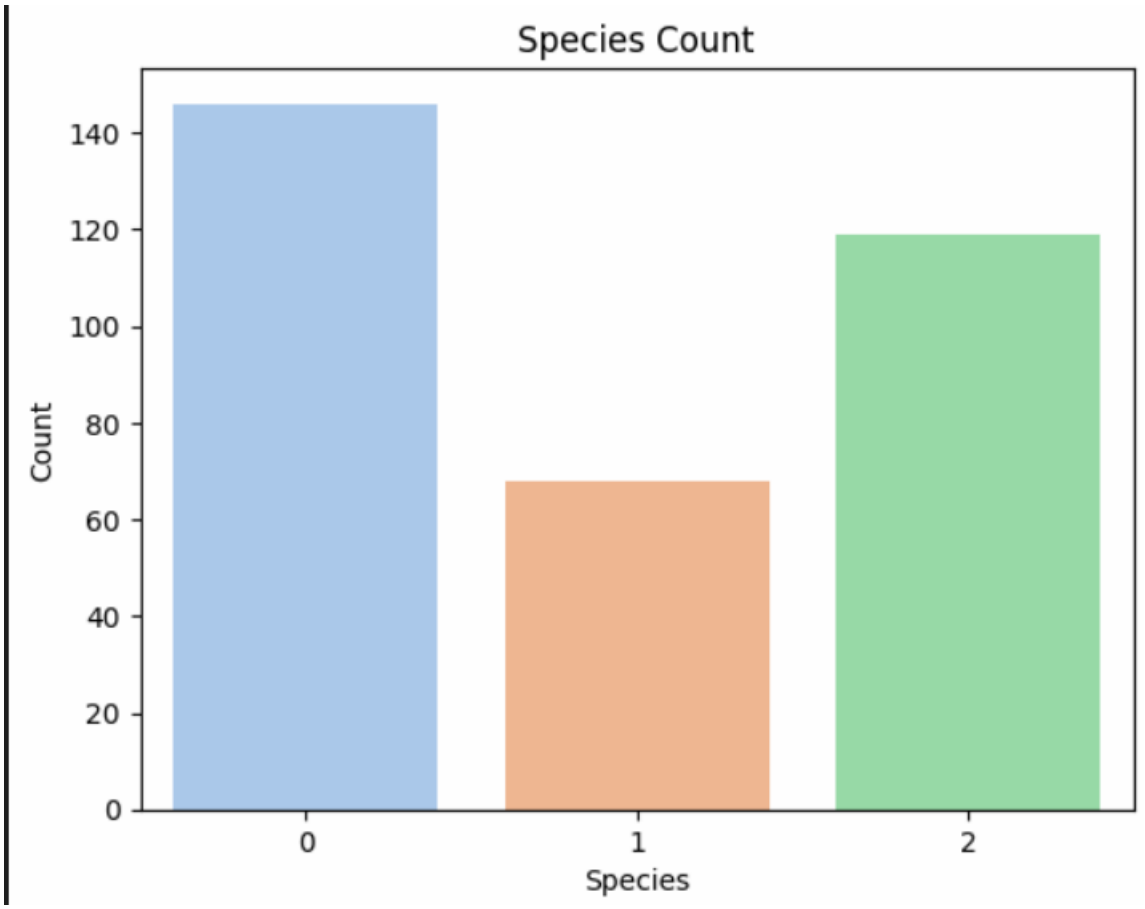
```python
df = pd.read_csv('/content/penguins.csv')
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 344 entries, 0 to 343
Data columns (total 9 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Unnamed: 0         344 non-null    int64  
 1   species            344 non-null    object 
 2   island             344 non-null    object 
 3   bill_length_mm     342 non-null    float64
 4   bill_depth_mm      342 non-null    float64
 5   flipper_length_mm  342 non-null    float64
 6   body_mass_g        342 non-null    float64
 7   sex                333 non-null    object 
 8   year               344 non-null    int64  
dtypes: float64(4), int64(2), object(3)
memory usage: 24.3+ KB
```

```python
print(df.isnull().sum())
```

```
Unnamed: 0           0
species              0
island               0
bill_length_mm       2
bill_depth_mm        2
flipper_length_mm    2
body_mass_g          2
sex                 11
year                 0
dtype: int64
```

```python
sns.countplot(data=df, x='species', palette='pastel')
plt.title('Species Count')
plt.xlabel('Species')
plt.ylabel('Count')
plt.show()
```


Species Count
Count
140
120
100
80
60
40
20
0
0
1
2
Species

```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

df.dropna(inplace=True)
if 'Unnamed: 0' in df.columns:
    df.drop('Unnamed: 0', axis=1, inplace=True)

df['species'] = df['species'].astype('category').cat.codes
df['island'] = df['island'].astype('category').cat.codes
df['sex'] = df['sex'].astype('category').cat.codes
X = df.drop('species', axis=1)
y = df['species']

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

clf_gini = DecisionTreeClassifier(criterion='gini', random_state=42)
clf_gini.fit(X_train, y_train)
accuracy_gini = accuracy_score(y_test, clf_gini.predict(X_test))

clf_entropy = DecisionTreeClassifier(criterion='entropy', random_state=42)
clf_entropy.fit(X_train, y_train)
accuracy_entropy = accuracy_score(y_test, clf_entropy.predict(X_test))

print("Gini Accuracy:", accuracy_gini)
print("Entropy Accuracy:", accuracy_entropy)

if accuracy_gini > accuracy_entropy:
    print("Gini performed better.")
elif accuracy_gini < accuracy_entropy:
    print("Entropy performed better.")
else:
    print("Both performed equally.")
```

```
Gini Accuracy: 0.9850746268656716
Entropy Accuracy: 1.0
Entropy performed better.
```

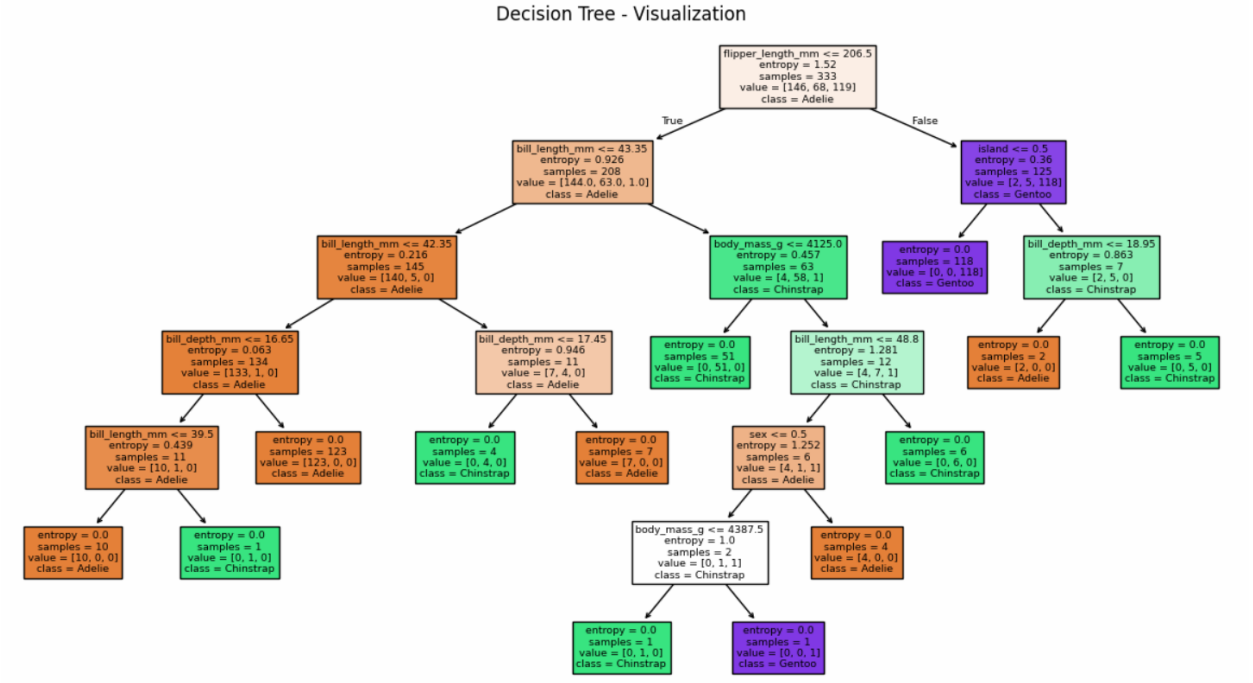
2. Which feature is most important? Sort it in highest to lowest order.

```python
import pandas as pd
from sklearn.tree import DecisionTreeClassifier

df = pd.read_csv('/content/penguins.csv')
df.dropna(inplace=True)
if 'Unnamed: 0' in df.columns:
    df.drop('Unnamed: 0', axis=1, inplace=True)

df['species'] = df['species'].astype('category').cat.codes
df['island'] = df['island'].astype('category').cat.codes
df['sex'] = df['sex'].astype('category').cat.codes

X = df.drop('species', axis=1)
y = df['species']

model = DecisionTreeClassifier(criterion='gini', random_state=42)
model.fit(X, y)

# Get and sort feature importances
importances = model.feature_importances_
features = X.columns
importance_df = pd.DataFrame({'Feature': features, 'Importance': importances})
importance_df = importance_df.sort_values(by='Importance', ascending=False)

print("Feature Importance (High to Low):\n")
print(importance_df)
```

```
Feature Importance (High to Low):

             Feature  Importance
3  flipper_length_mm    0.517364
1     bill_length_mm    0.369059
2      bill_depth_mm    0.078921
4        body_mass_g    0.017897
0             island    0.016760
5                sex    0.000000
6               year    0.000000
```

3. Visualize the decision tree classifier. Which is the best split when applying the above mentioned criteria?

```python
import pandas as pd
from sklearn.tree import DecisionTreeClassifier, plot_tree
import matplotlib.pyplot as plt

df = pd.read_csv('/content/penguins.csv')
df.dropna(inplace=True)
if 'Unnamed: 0' in df.columns:
    df.drop('Unnamed: 0', axis=1, inplace=True)

df['species'] = df['species'].astype('category').cat.codes
df['island'] = df['island'].astype('category').cat.codes
df['sex'] = df['sex'].astype('category').cat.codes

X = df.drop('species', axis=1)
y = df['species']

clf = DecisionTreeClassifier(criterion='entropy', random_state=42)
clf.fit(X, y)

# Visualize the tree
plt.figure(figsize=(15, 8))
plot_tree(clf, feature_names=X.columns, class_names=['Adelie', 'Chinstrap', 'Gentoo'], filled=True)
plt.title("Decision Tree - Visualization")
plt.show()
```



Lab 8 – Random Forest

Objective:

The objective of this lab is to understand the working principles of Random Forest for classification tasks and implement them. It aims to explore different attribute selection measures, including Information Gain (Entropy), Gain Ratio, and Gini Index, to determine the best feature for splitting at each node. Furthermore, the lab focuses on comparing the performance of different splitting criteria to analyze their impact on classification accuracy. Another key objective is to examine feature importance in random forest classification and understand how different features contribute to decision-making.

Tools Required:

Anaconda Distribution/ Google Colab/ Pycharm

An Overview of Random Forests

Random forests are a popular supervised machine learning algorithm that can handle both regression and classification tasks. Below are some of the main characteristics of random forests:

- Random forests are for supervised machine learning, where there is a labeled target variable.
- Random forests can be used for solving regression (numeric target variable) and classification (categorical target variable) problems.
- Random forests are an ensemble method, meaning they combine predictions from other models.
- Each of the smaller models in the random forest ensemble is a decision tree.

How Random Forest Classification Works

Imagine you have a complex problem to solve, and you gather a group of experts from different fields to provide their input. Each expert provides their opinion based on their expertise and experience. Then, the experts would vote to arrive at a final decision.

In a random forest classification, multiple decision trees are created using different random subsets of the data and features. Each decision tree is like an expert, providing its opinion on how to classify the data. Predictions are made by calculating the prediction for each decision tree and then taking the most popular result. (For regression, predictions use an averaging technique instead.)

In the diagram below, we have a random forest with n decision trees, and we've shown the first 5, along with their predictions (either "Dog" or "Cat"). Each tree is exposed to a different number of features and a different sample of the original dataset, and as such, every tree can be different. Each tree makes a prediction.

Looking at the first 5 trees, we can see that 4/5 predicted the sample was a Cat. The green circles indicate a hypothetical path the tree took to reach its decision. The random forest would count the number of predictions from decision trees for Cat and for Dog, and choose the most popular prediction.

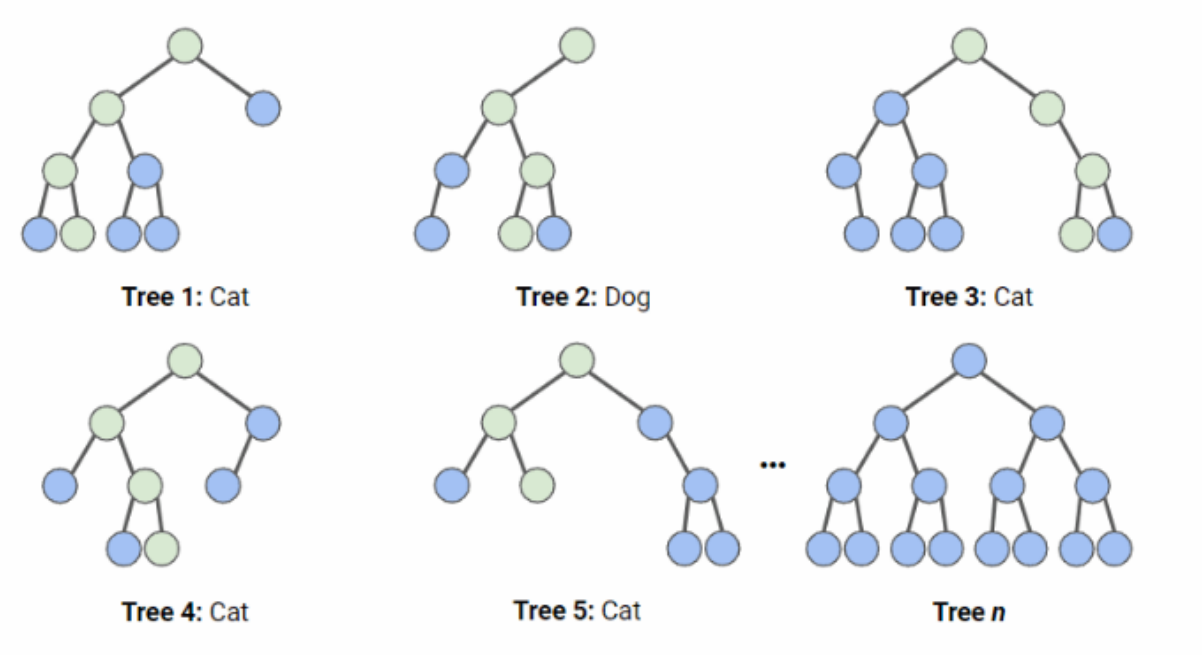


Illustration of how random forest classification works

Coding Example

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report
import seaborn as sns
from sklearn.metrics import confusion_matrix


# Load the Iris dataset
iris = datasets.load_iris()
X, y = iris.data, iris.target


# Split into training (80%) and testing (20%) sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
# Initialize and train the Random Forest classifier
clf = RandomForestClassifier(n_estimators=100, criterion="gini", random_state=42)
clf.fit(X_train, y_train)

# Make predictions
y_pred = clf.predict(X_test)

# Compute accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Test Accuracy: {accuracy:.4f}")

# Classification report
print("\nClassification Report:\n", classification_report(y_test, y_pred,
target_names=iris.target_names))

# Confusion matrix visualization
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(6, 5))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=iris.target_names,
yticklabels=iris.target_names)
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Confusion Matrix")
plt.show()
```

Exercise:

1. Apply the Random Forest classifier to the “penguins_size.csv”dataset with random_state = 42. Determine the highest accuracy achieved and identify the hyperparameter values that contribute to this optimal performance.

2. Which feature is most important? Sort it in highest to lowest order.

3. Visualize the decision tree classifier.

```
[9]  import pandas as pd
     import seaborn as sns
     import matplotlib.pyplot as plt
     from sklearn.model_selection import train_test_split
     from sklearn.ensemble import RandomForestClassifier
     from sklearn.preprocessing import LabelEncoder
     from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
     import numpy as np

     # Load the dataset
     penguins = pd.read_csv("/content/penguins_size.csv")

     # Drop rows with missing values
     penguins.dropna(inplace=True)

     # Encode categorical features
     le = LabelEncoder()
     penguins['species'] = le.fit_transform(penguins['species'])  # Target
     penguins['sex'] = le.fit_transform(penguins['sex'])
     penguins['island'] = le.fit_transform(penguins['island'])

     # Features and target
     X = penguins.drop('species', axis=1)
     y = penguins['species']
```

```python
# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize and train
clf = RandomForestClassifier(n_estimators=100, random_state=42)
clf.fit(X_train, y_train)

# Predict and evaluate
y_pred = clf.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)

print(f"Accuracy: {accuracy:.4f}")
print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

```
Accuracy: 1.0000

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        31
           1       1.00      1.00      1.00        13
           2       1.00      1.00      1.00        23

    accuracy                           1.00        67
   macro avg       1.00      1.00      1.00        67
weighted avg       1.00      1.00      1.00        67
```

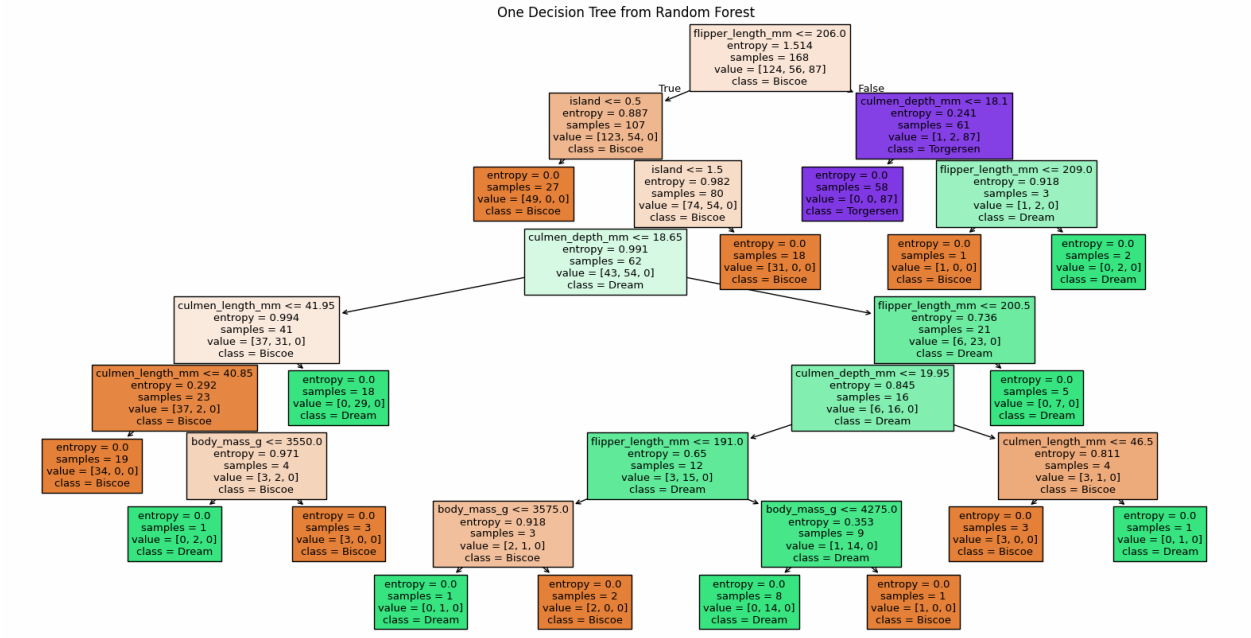
```python
# Feature importance
importances = clf.feature_importances_
feature_names = X.columns

# Sort
sorted_indices = np.argsort(importances)[::-1]
sorted_features = [(feature_names[i], importances[i]) for i in sorted_indices]

print("\nFeature Importances (High to Low):")
for feature, score in sorted_features:
    print(f"{feature}: {score:.4f}")
```

```
Feature Importances (High to Low):
culmen_length_mm: 0.3531
flipper_length_mm: 0.2344
culmen_depth_mm: 0.1796
body_mass_g: 0.1150
island: 0.1092
sex: 0.0088
```

```python
best_acc = 0
best_params = {}

for criterion in ['gini', 'entropy']:
    for depth in [3, 5, 10, None]:
        for n in [50, 100, 200]:
            clf = RandomForestClassifier(n_estimators=n, max_depth=depth, criterion=criterion, random_state=42)
            clf.fit(X_train, y_train)
            acc = accuracy_score(y_test, clf.predict(X_test))
            if acc > best_acc:
                best_acc = acc
                best_params = {"criterion": criterion, "max_depth": depth, "n_estimators": n}

print("\nBest Accuracy:", best_acc)
print("Best Parameters:", best_params)
```

```
Best Accuracy: 1.0
Best Parameters: {'criterion': 'gini', 'max_depth': 5, 'n_estimators': 50}
```

```python
from sklearn.tree import plot_tree

plt.figure(figsize=(20,10))
plot_tree(clf.estimators_[0], feature_names=X.columns, class_names=le.classes_, filled=True)
plt.title("One Decision Tree from Random Forest")
plt.show()
```



Lab 9 – Naïve Bayes Classifier

Objective:

The objective of the Naïve Bayes Classifier is to understand and implement a probabilistic classification algorithm based on Bayes' Theorem with the assumption of feature independence. This lab explores different Naïve Bayes models (Gaussian, Multinomial, Bernoulli) and their applications in classification tasks. It focuses on calculating probabilities, handling text and numerical data, and evaluating model performance using accuracy, precision, recall, and F1-score. By applying Naïve Bayes to a real dataset, students will gain practical experience in training, testing, and analyzing its strengths and limitations.

Tools Required:

Anaconda Distribution/ Google Colab/ Pycharm

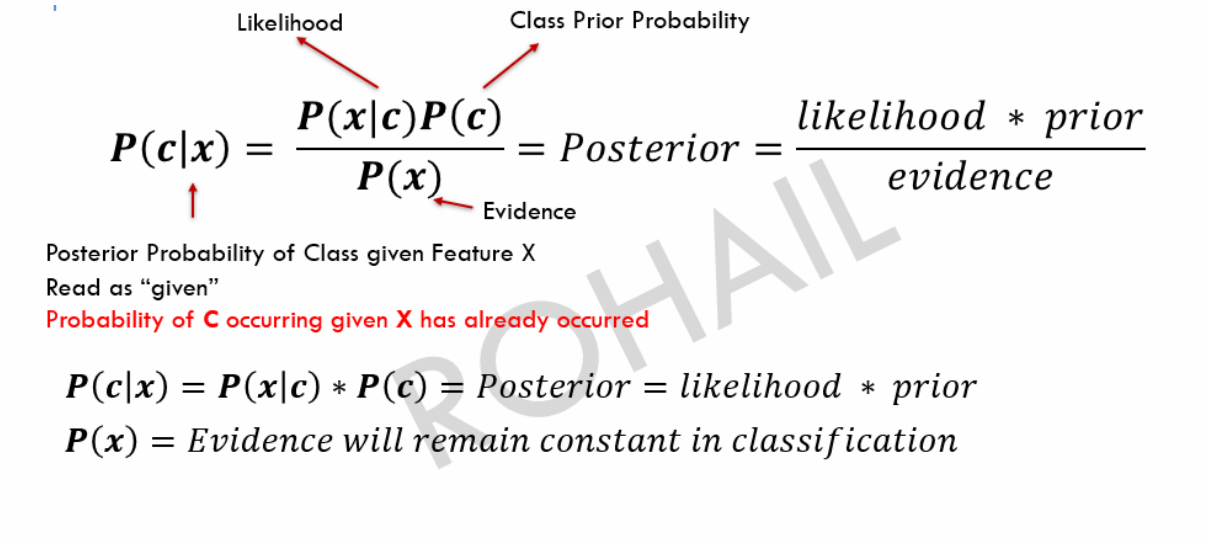
Classification

Classification is a two-step process; a learning step and a prediction step. In the learning step, the model is developed based on given training data. In the prediction step, the model is used to predict the response to given data. A Decision tree is one of the easiest and most popular classification algorithms used to understand and interpret data. It can be utilized for both classification and regression problems.

Naïve Bayes Classifier

Naive Bayes is a statistical classification technique based on Bayes Theorem. It is one of the simplest supervised learning algorithms. Naive Bayes classifier is the fast, accurate and reliable algorithm. Naive Bayes classifiers have high accuracy and speed on large datasets.

Naive Bayes classifier assumes that the effect of a particular feature in a class is independent of other features. For example, a loan applicant is desirable or not depending on his/her income, previous loan and transaction history, age, and location. Even if these features are interdependent, these features are still considered independently. This assumption simplifies computation, and that's why it is considered as naive. This assumption is called class conditional independence.



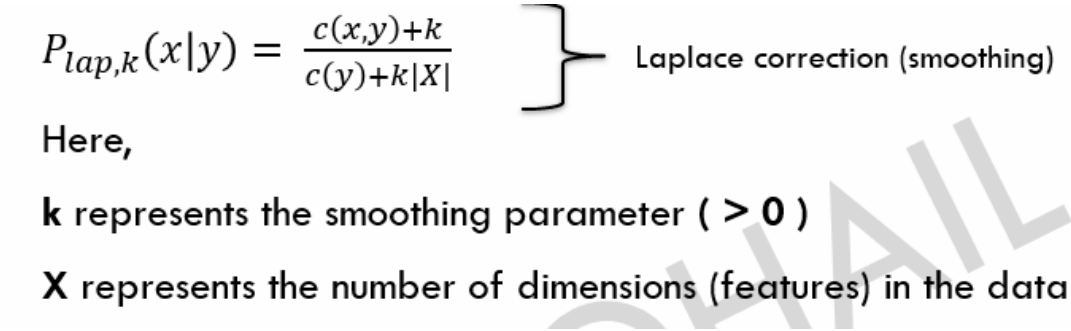
Zero Probability Problem

Suppose there is no tuple for a risky loan in the dataset; in this scenario, the posterior probability will be zero, and the model is unable to make a prediction. This problem is known as Zero Probability because the occurrence of the particular class is zero.

The solution for such an issue is the Laplace correction or Laplace Transformation. Laplace correction is one of the smoothing techniques. Here, you can assume that the dataset is large enough that adding one row of each class will not make a difference in the estimated probability. This will overcome the issue of probability values to zero.

For Example: Suppose that for the class loan risky, there are 1000 training tuples in the database. In this database, the income column has 0 tuples for low income, 990 tuples for medium income, and 10 tuples for high income. The probabilities of these events, without the Laplace correction, are 0, 0.990 (from 990/1000), and 0.010 (from 10/1000)

Now, apply Laplace correction on the given dataset. Let's add 1 more tuple for each income-value pair. The probabilities of these events:



Different Variants of Naïve Bayes

1. Gaussian Naive Bayes: Assumes that continuous features follow a Gaussian (normal) distribution. It is suitable for data with continuous features. It is used in binary and multi-classification problems

2. Multinomial Naive Bayes: Designed for discrete data, where features represent counts (e.g., word frequencies in a document). Commonly used in text classification problems.

3. Bernoulli Naive Bayes: Suitable for binary data, where features are binary variables It's often used in **text classification** tasks, considering the presence or absence of specific words It might perform better on some datasets, especially those with shorter documents. It is advisable to evaluate both models

4. Complement Naive Bayes: A variation of the multinomial Naive Bayes that is particularly effective for imbalanced datasets It adjusts the class probabilities inversely proportional to the number of documents in each class

Application of Naïve Bayes

1. Text classification
2. Face recognition
3. Weather prediction
4. Medical diagnosis
5. Spam filtering
6. Recommendation system
7. Sentiment analysis

Coding Example

```
import numpy as np

import pandas as pd

from sklearn.feature_extraction.text import CountVectorizer, TfidfTransformer

from sklearn.naive_bayes import MultinomialNB

from sklearn.pipeline import Pipeline

from sklearn.model_selection import train_test_split

from sklearn.metrics import accuracy_score, classification_report


# Sample dataset (Text and Labels)
```

```
data = {
    "text": [
        "I love programming in Python",
        "Python is great for data science",
        "I enjoy watching movies",
        "Data science and machine learning are amazing",
        "Movies are fun to watch",
        "I dislike boring movies",
        "Machine learning is powerful",
        "Python is my favorite language",
        "I hate bad movies"
    ],
    "label": ["tech", "tech", "entertainment", "tech", "entertainment", "entertainment", "tech", "tech", "entertainment"]
}

# Convert data to DataFrame
df = pd.DataFrame(data)

# Split dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(df["text"], df["label"], test_size=0.2, random_state=42)

# Create a pipeline with text preprocessing and Naïve Bayes classifier
text_clf = Pipeline([
    ('vect', CountVectorizer()),           # Convert text to token counts
    ('tfidf', TfidfTransformer()),         # Transform counts to TF-IDF scores
    ('clf', MultinomialNB())               # Apply Naïve Bayes classifier
])
```

```
# Train the model
text_clf.fit(X_train, y_train)

# Make predictions
y_pred = text_clf.predict(X_test)
```

```
# Evaluate model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")
print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

Exercise:

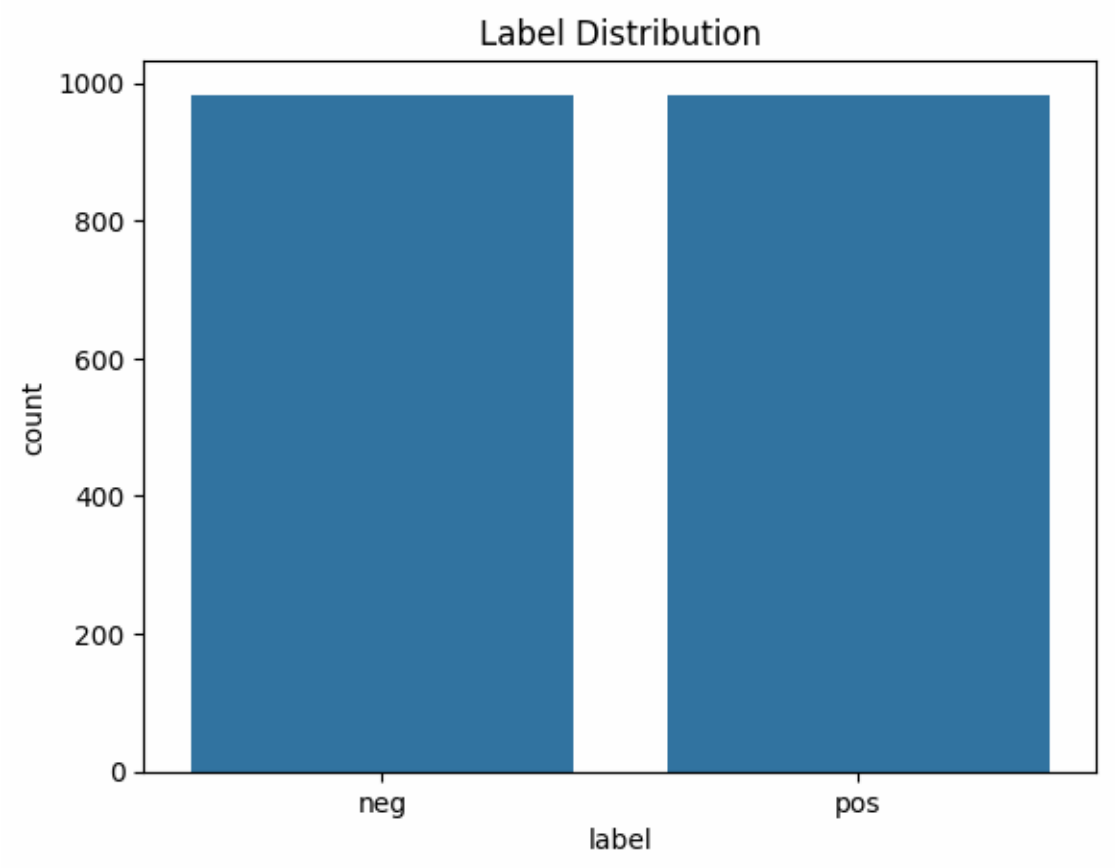
```
import pandas as pd

df = pd.read_csv("/content/movie_reviews.csv")
print(df.head())
```

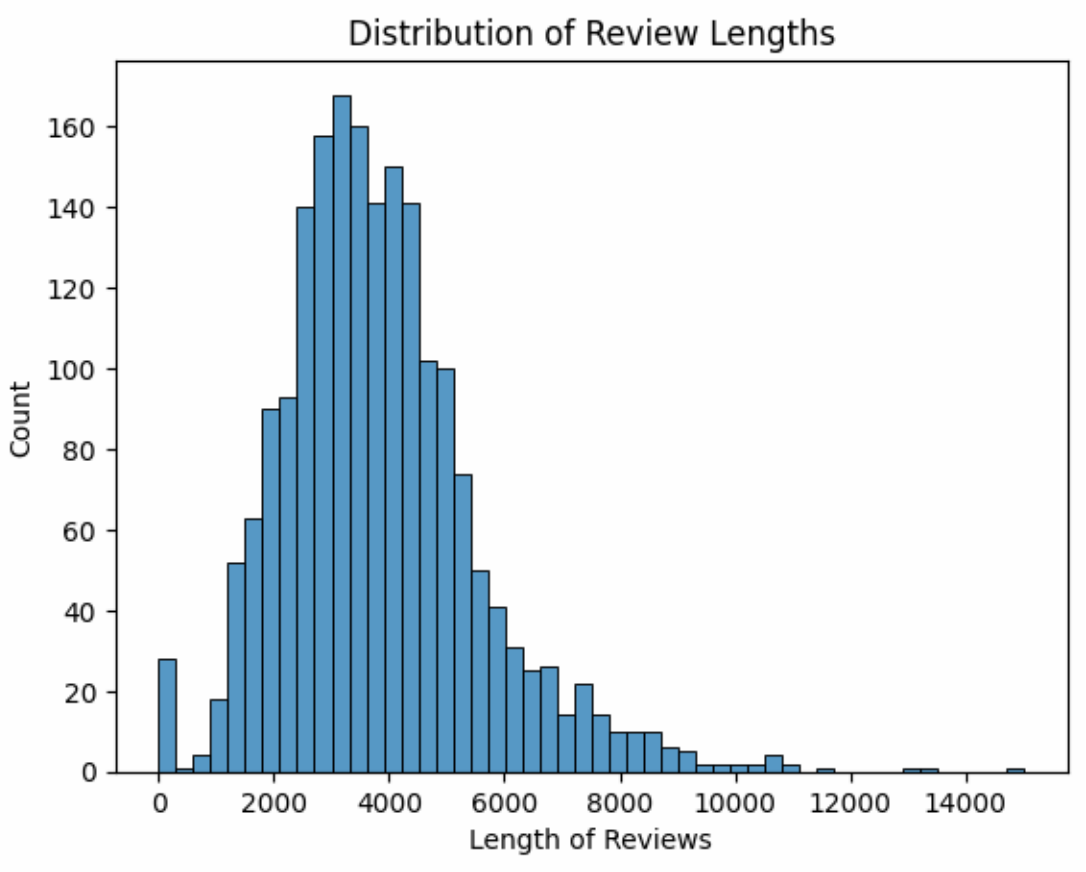
```
  label                                             review
0   neg  how do films like mouse hunt get into theatres...
1   neg  some talented actresses are blessed with a dem...
2   pos  this has been an extraordinary year for austra...
3   pos  according to hollywood movies made in last few...
4   neg  my first press screening of 1998 and already i...
```

1. Perform data preprocessing before feeding to classifier. Also do Exploratory Data Analysis,

```
print(df.isnull().sum())

df.dropna(inplace=True)

import seaborn as sns
import matplotlib.pyplot as plt

sns.countplot(x='label', data=df)
plt.title("Label Distribution")
plt.show()

df['text_length'] = df['review'].apply(len)

sns.histplot(df['text_length'], bins=50)
plt.title("Distribution of Review Lengths")
plt.xlabel("Length of Reviews")
plt.show()
```

```
label      0
review    35
dtype: int64
```


Label Distribution
1000
800
600
400
200
0
count
neg
pos
label


Distribution of Review Lengths
Count
Length of Reviews
0
20
40
60
80
100
120
140
160
0
2000
4000
6000
8000
10000
12000
14000

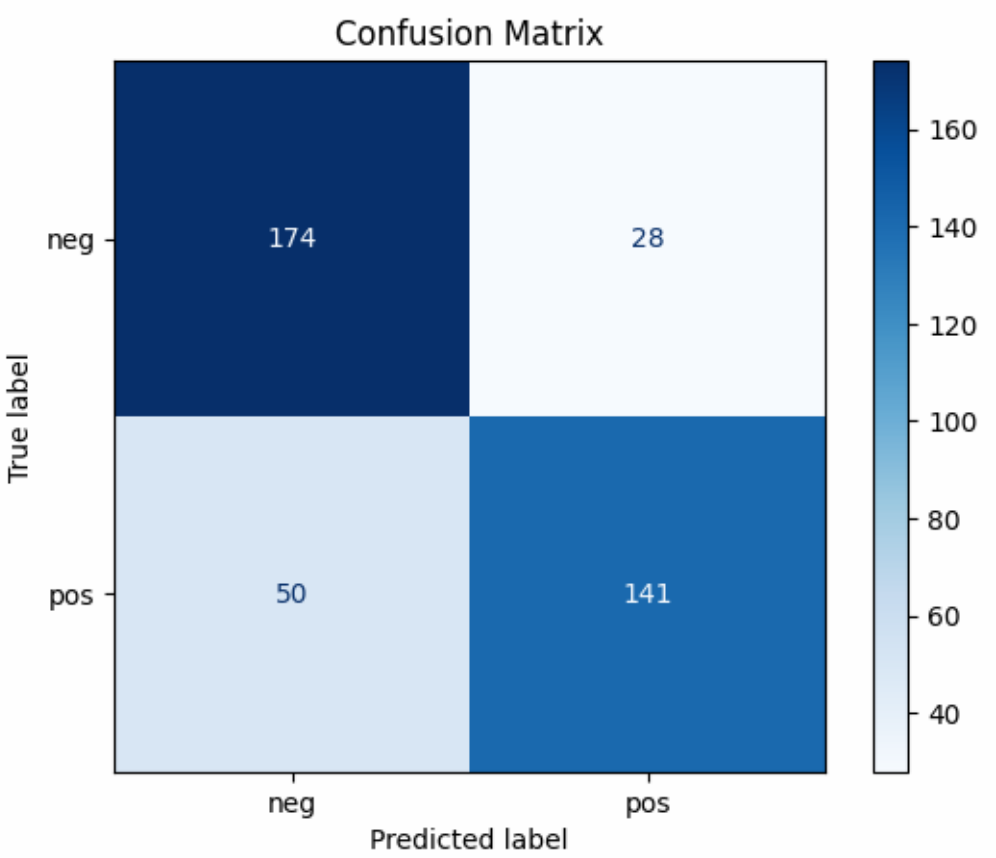
2. Apply the Naïve Bayes classifier on “movie_reviews.csv” dataset. Determine the highest accuracy achieved and identify the hyperparameter values that contribute to this optimal performance.

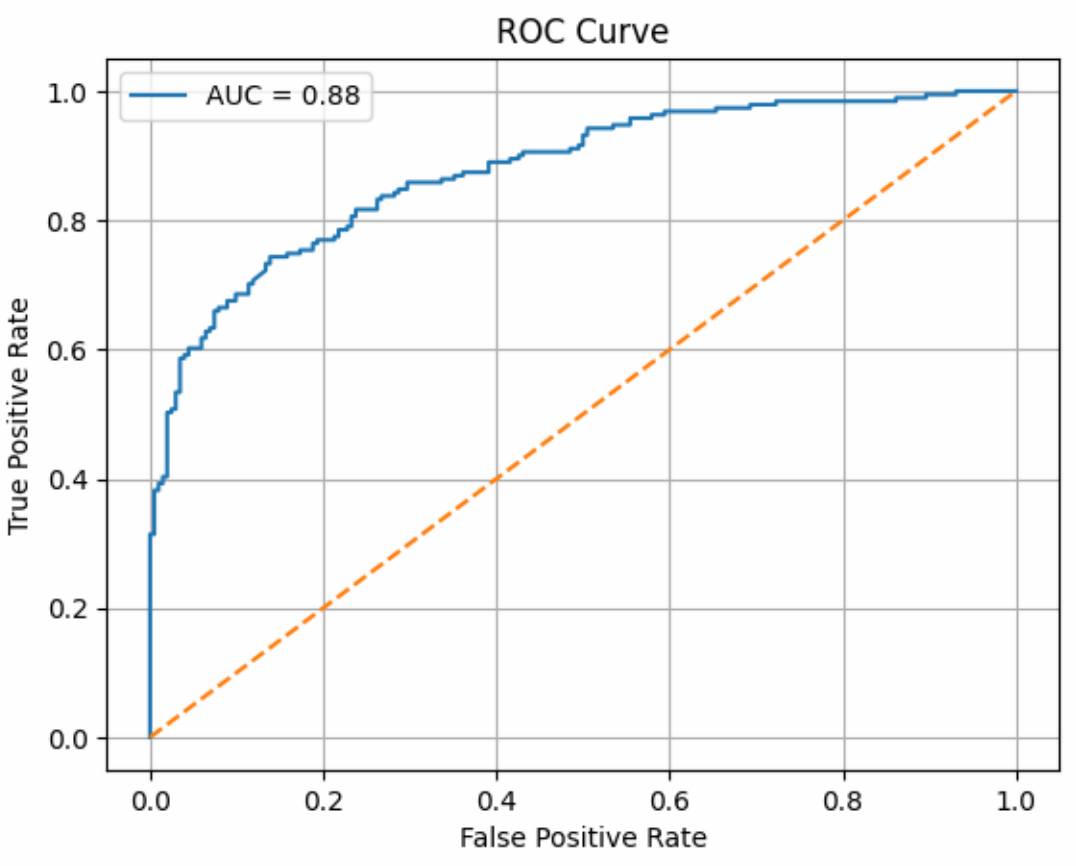
```python
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer, TfidfTransformer
from sklearn.pipeline import Pipeline
from sklearn.naive_bayes import MultinomialNB, BernoulliNB, GaussianNB
from sklearn.metrics import accuracy_score, classification_report

X_train, X_test, y_train, y_test = train_test_split(df['review'], df['label'], test_size=0.2, random_state=42)

pipelines = {
    "Multinomial": Pipeline([
        ('vect', CountVectorizer()),
        ('tfidf', TfidfTransformer()),
        ('clf', MultinomialNB())
    ]),
    "Bernoulli": Pipeline([
        ('vect', CountVectorizer(binary=True)),
        ('tfidf', TfidfTransformer(use_idf=False)),
        ('clf', BernoulliNB())
    ]),
}

for name, pipe in pipelines.items():
    pipe.fit(X_train, y_train)
    preds = pipe.predict(X_test)
    acc = accuracy_score(y_test, preds)
    print(f"\n{name} Naive Bayes Accuracy: {acc:.4f}")
    print(classification_report(y_test, preds))
```

```
Multinomial Naive Bayes Accuracy: 0.8015
              precision    recall  f1-score   support

         neg       0.78      0.86      0.82       202
         pos       0.83      0.74      0.78       191

    accuracy                           0.80       393
   macro avg       0.81      0.80      0.80       393
weighted avg       0.80      0.80      0.80       393


Bernoulli Naive Bayes Accuracy: 0.7812
              precision    recall  f1-score   support

         neg       0.73      0.92      0.81       202
         pos       0.88      0.63      0.74       191

    accuracy                           0.78       393
   macro avg       0.80      0.78      0.78       393
weighted avg       0.80      0.78      0.78       393
```

3. Visualize the performance of classifier using confusion matrix, classification report, and ROC curve

```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay, roc_curve, auc
from sklearn.preprocessing import LabelBinarizer

best_model = pipelines['Multinomial']
y_pred = best_model.predict(X_test)

cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=best_model.classes_)
disp.plot(cmap=plt.cm.Blues)
plt.title("Confusion Matrix")
plt.show()

lb = LabelBinarizer()
y_test_bin = lb.fit_transform(y_test)
y_prob = best_model.predict_proba(X_test)[:, 1]

fpr, tpr, _ = roc_curve(y_test_bin, y_prob)
roc_auc = auc(fpr, tpr)

plt.plot(fpr, tpr, label=f'AUC = {roc_auc:.2f}')
plt.plot([0, 1], [0, 1], linestyle='--')
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve")
plt.legend()
plt.grid()
plt.show()
```


Confusion Matrix
neg
pos
True label
174
28
50
141
neg
pos
Predicted label
160
140
120
100
80
60
40


ROC Curve
AUC = 0.88
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0

Lab 10 – Linear Regression

Objective:

The objective of the Naïve Bayes Classifier is to understand and implement a probabilistic classification algorithm based on Bayes' Theorem with the assumption of feature independence. This lab explores different Naïve Bayes models (Gaussian, Multinomial, Bernoulli) and their applications in classification tasks. It focuses on calculating probabilities, handling text and numerical data, and evaluating model performance using accuracy, precision, recall, and F1-score. By applying Naïve Bayes to a real dataset, students will gain practical experience in training, testing, and analyzing its strengths and limitations.

Tools Required:

Anaconda Distribution/ Google Colab/ Pycharm

Regression

Simple linear regression is a linear regression with one independent variable, also called the explanatory variable, and one dependent variable, also called the response variable. In simple linear regression, the dependent variable is continuous.

Ordinary least squares estimator

The most common way to do simple linear regression is through ordinary least squares (OLS) estimation. Because OLS is by far the most common method, the “ordinary least squares” part is often implied when we talk about simple linear regression.

Ordinary least squares works by minimizing the sum of the squared differences between the observed values (the actual data points) and the predicted values from the regression line. These differences are called residuals, and squaring them ensures that both positive and negative residuals are treated equally.

How Simple Linear Regression is Used

Simple linear regression helps make predictions and understand relationships between one independent variable and one dependent variable. For example, you might want to know how a tree’s height (independent variable) affects the number of leaves it has (dependent variable). By collecting data and fitting a simple linear regression model, you could predict the number of leaves based on the tree's height. This is the ‘making predictions’ part. But this approach also reveals how much the number of leaves changes, on average, as the tree grows taller, which is how simple linear regression is also used to understand relationships.

Simple linear regression equation

Let's take a look at the simple linear regression equation. We can start by first looking at the slope-intercept form of a straight line using notation that is common in geometry or algebra textbooks. That is, we will start at the beginning.

$$y = mx + b$$

Here
- *m* is the slope of the line
- *b* is the intercept

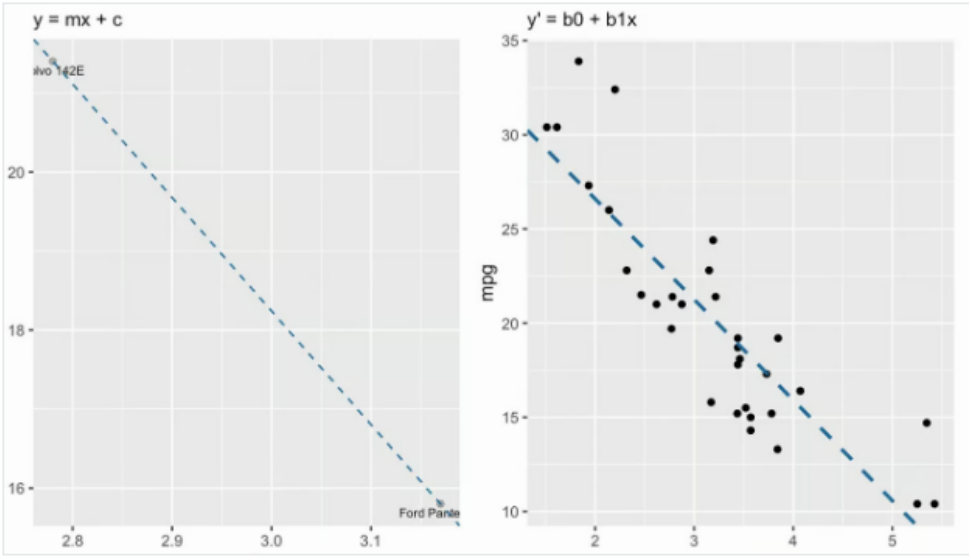
In the context of data science, you are more likely to see this equation instead:

$$y = b_0 + b_1 * x_1$$

Where
- *b0* is the y-intercept
- *b1* is the slope

The notation involving b0 and b1 helps us understand that we are looking at a situation where we are making a prediction on y, which is why we call it *ŷ*, or *y-hat*, since we don't expect our regression line will actually go through all the points.

The following visualization shows the conceptual difference between the slope-intercept form of the line, on the left, and the regression equation, on the right. In the language of linear algebra, we would say that the system of linear equations is overdetermined, meaning there are the more equations (thirty or so) than there are unknowns (two), so we don't expect to find a solution.



Simple linear regression model assumptions

Let's take a look now at the main simple linear regression model assumptions. If these assumptions are violated, we might want to consider a different approach. The first three, in particular, are strong assumptions and shouldn't be ignored.

1. Linearity: The relationship between the independent and dependent variables must be linear. If the relationship is non-linear, the model won't capture it well.

2. Independence of Errors: Residuals should be independent of each other. This means there should be no patterns or correlations between the residuals. This is something to watch for closely in time-ordered data.
3. Homoscedasticity: The residuals should have constant variance across all values of the independent variable. If the variance changes (heteroscedasticity), predictions in certain ranges of x may become less accurate.
4. Normality of Residuals: Residuals should ideally follow a normal, or Gaussian distribution. This is important for statistical testing, and asserting levels of confidence in our estimate. It's less critical for making predictions.

Application of Naïve Bayes

1. Sales forecasting
2. Risk assessment
3. Real estate price prediction
4. Healthcare

Coding Example

```
import numpy as np

# Sample data
X = np.array([1, 2, 3, 4, 5])
y = np.array([2, 4, 5, 4, 5])

# Calculate means
mean_X = np.mean(X)
mean_y = np.mean(y)

# Calculate standard deviations
sd_X = np.std(X, ddof=1)
sd_y = np.std(y, ddof=1)

# Calculate correlation
correlation = np.corrcoef(X, y)[0, 1]

# Calculate slope (b1) using the formula: b1 = (correlation * sd_y) / sd_X
slope = (correlation * sd_y) / sd_X

# Calculate intercept (b0) using the formula: b0 = mean_y - slope * mean_X
intercept = mean_y - slope * mean_X

# Print the slope and intercept
print(f"Slope (b1): {slope}")
print(f"Intercept (b0): {intercept}")

# Use the manually calculated coefficients to predict y values
y_pred = intercept + slope * X
```

```
print(f"Predicted values: {y_pred}")
```

Exercise

1. Load the “social_anxiety_dataset.csv” data.

```
import pandas as pd

df = pd.read_csv("/content/social_anxiety_dataset.csv")
print(df.head())
```

```
   Age  Gender Occupation  Sleep Hours  Physical Activity (hrs/week)  \
0   29  Female     Artist          6.0                           2.7
1   46   Other      Nurse          6.2                           5.7
2   64    Male      Other          5.0                           3.7
3   20  Female  Scientist          5.8                           2.8
4   49  Female      Other          8.2                           2.3

   Caffeine Intake (mg/day)  Alcohol Consumption (drinks/week) Smoking  \
0                       181                                10     Yes
1                       200                                 8     Yes
2                       117                                 4      No
3                       360                                 6     Yes
4                       247                                 4     Yes

  Family History of Anxiety  Stress Level (1-10)  Heart Rate (bpm)  \
0                        No                   10               114
1                       Yes                    1                62
2                       Yes                    1                91
3                        No                    4                86
4                        No                    1                98

   Breathing Rate (breaths/min)  Sweating Level (1-5) Dizziness Medication  \
0                            14                     4        No        Yes
1                            23                     2       Yes         No
2                            28                     3        No         No
3                            17                     3        No         No
4                            19                     4       Yes        Yes

   Therapy Sessions (per month) Recent Major Life Event  Diet Quality (1-10)  \
0                             3                     Yes                    7
1                             2                      No                    8
2                             1                     Yes                    1
3                             0                      No                    1
4                             1                      No                    3

   Anxiety Level (1-10)
0                   5.0
1                   3.0
2                   1.0
3                   2.0
4                   1.0
```

2. Perform Exploratory Data Analysis

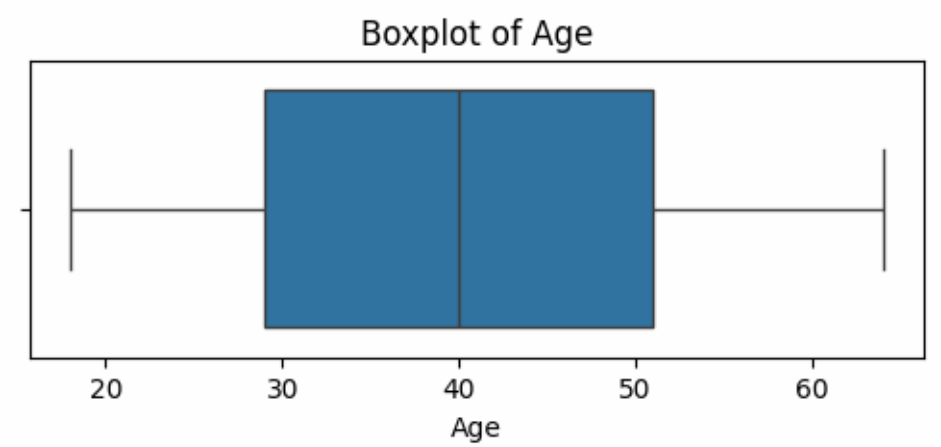
```
df.info()

df.describe(include='all')

for col in df.select_dtypes(include='object').columns:
    print(f"{col}: {df[col].unique()}")
```

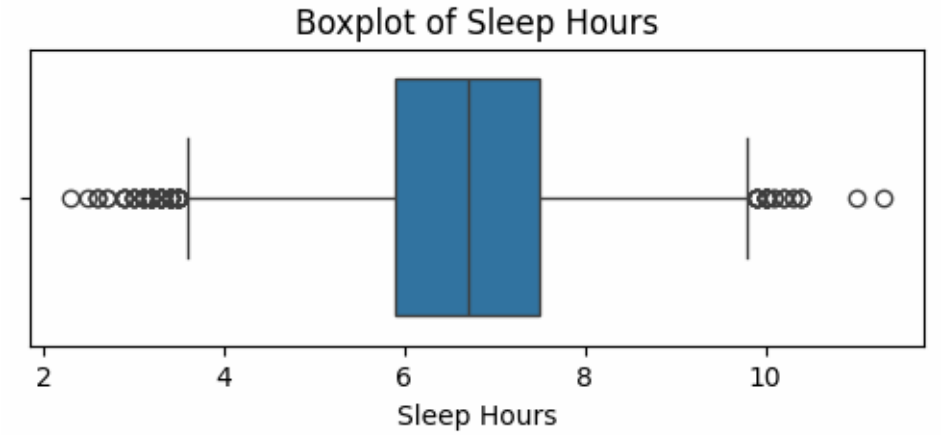
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 11000 entries, 0 to 10999
Data columns (total 19 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   Age                             11000 non-null  int64
 1   Gender                          11000 non-null  object
 2   Occupation                      11000 non-null  object
 3   Sleep Hours                     11000 non-null  float64
 4   Physical Activity (hrs/week)    11000 non-null  float64
 5   Caffeine Intake (mg/day)        11000 non-null  int64
 6   Alcohol Consumption (drinks/week)  11000 non-null  int64
 7   Smoking                         11000 non-null  object
 8   Family History of Anxiety       11000 non-null  object
 9   Stress Level (1-10)             11000 non-null  int64
 10  Heart Rate (bpm)                11000 non-null  int64
 11  Breathing Rate (breaths/min)    11000 non-null  int64
 12  Sweating Level (1-5)            11000 non-null  int64
 13  Dizziness                       11000 non-null  object
 14  Medication                      11000 non-null  object
 15  Therapy Sessions (per month)    11000 non-null  int64
 16  Recent Major Life Event         11000 non-null  object
 17  Diet Quality (1-10)             11000 non-null  int64
 18  Anxiety Level (1-10)            11000 non-null  float64
dtypes: float64(3), int64(9), object(7)
memory usage: 1.6+ MB
Gender: ['Female' 'Other' 'Male']
Occupation: ['Artist' 'Nurse' 'Other' 'Scientist' 'Lawyer' 'Teacher' 'Doctor'
 'Musician' 'Student' 'Engineer' 'Freelancer' 'Chef' 'Athlete']
Smoking: ['Yes' 'No']
Family History of Anxiety: ['No' 'Yes']
Dizziness: ['No' 'Yes']
Medication: ['Yes' 'No']
Recent Major Life Event: ['Yes' 'No']
```

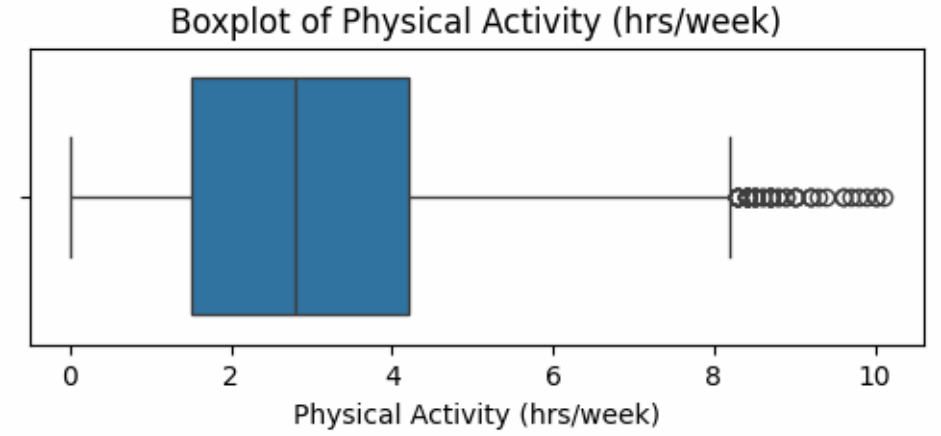
3. Perform the data preprocessing techniques such as identify outliers, missing values

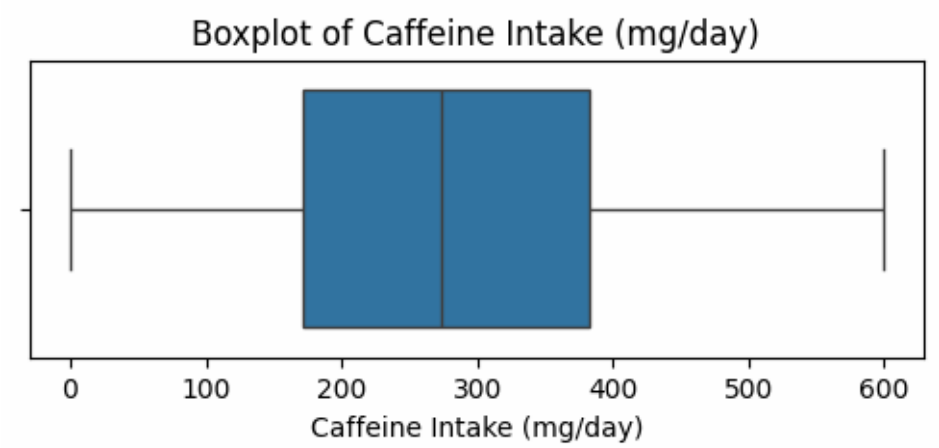
```
[21] print(df.isnull().sum())

     df = df.dropna()

     import seaborn as sns
     import matplotlib.pyplot as plt

     numeric_cols = df.select_dtypes(include='number').columns
     for col in numeric_cols:
         plt.figure(figsize=(6, 2))
         sns.boxplot(x=df[col])
         plt.title(f'Boxplot of {col}')
         plt.show()
```

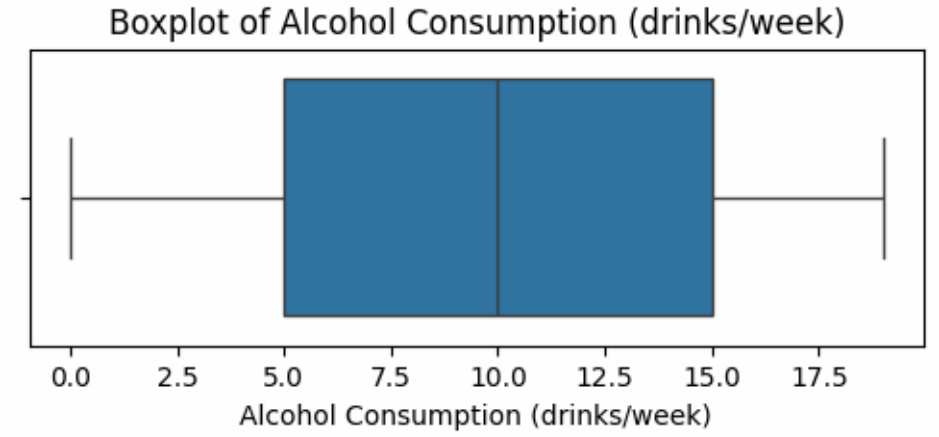
```
Age                                0
Gender                             0
Occupation                         0
Sleep Hours                        0
Physical Activity (hrs/week)       0
Caffeine Intake (mg/day)           0
Alcohol Consumption (drinks/week)  0
Smoking                            0
Family History of Anxiety          0
Stress Level (1-10)                0
Heart Rate (bpm)                   0
Breathing Rate (breaths/min)       0
Sweating Level (1-5)               0
Dizziness                          0
Medication                         0
Therapy Sessions (per month)       0
Recent Major Life Event            0
Diet Quality (1-10)                0
Anxiety Level (1-10)               0
dtype: int64
```

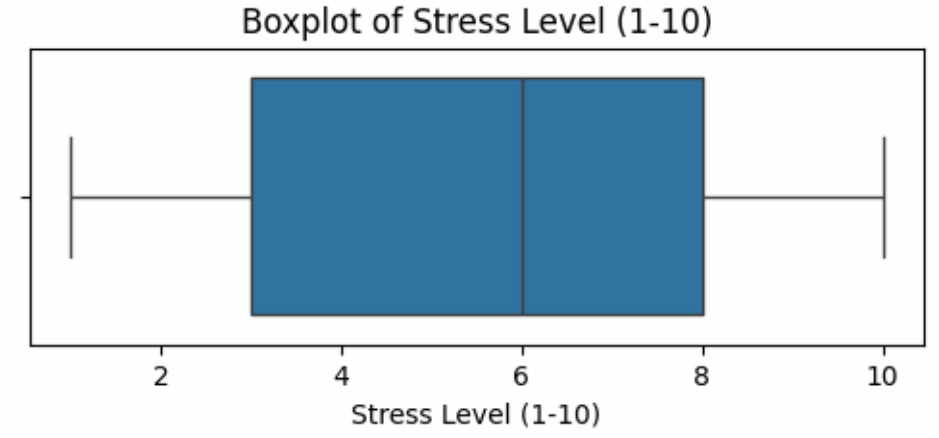

Boxplot of Age
20
30
40
50
60
Age

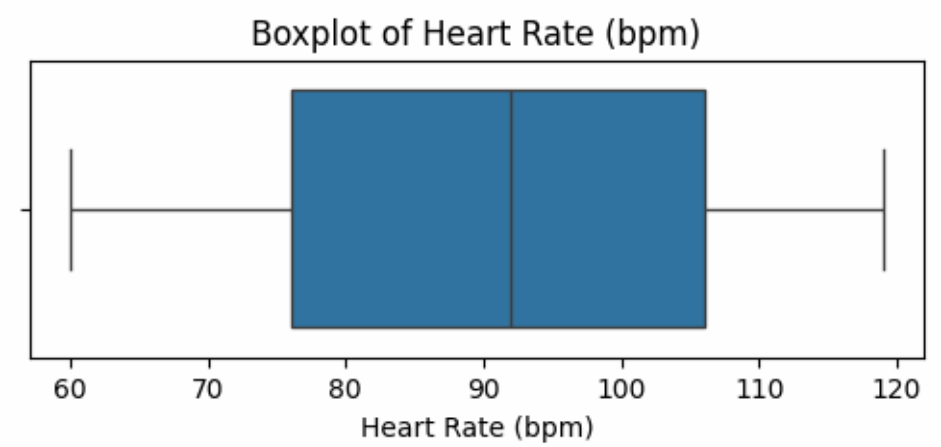

Boxplot of Sleep Hours
2
4
6
8
10
Sleep Hours

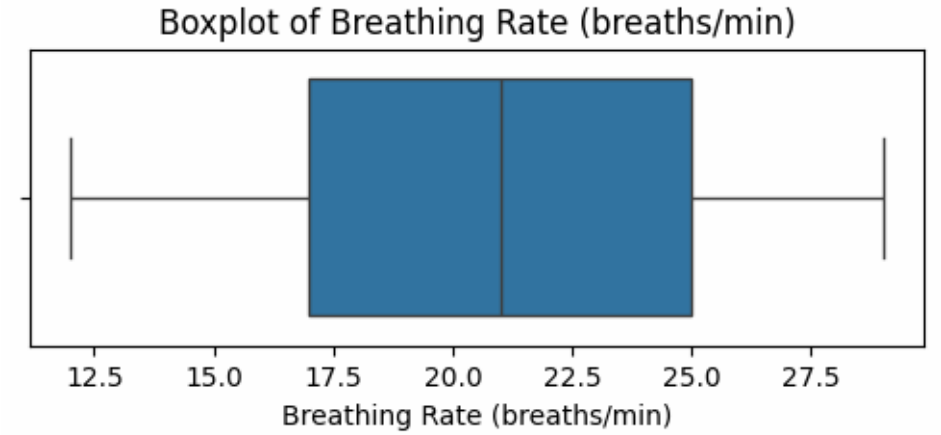

Boxplot of Physical Activity (hrs/week)
0
2
4
6
8
10
Physical Activity (hrs/week)

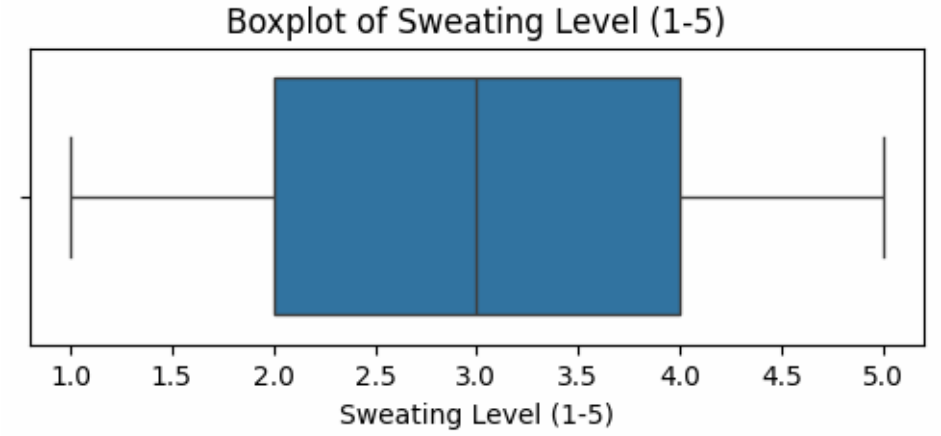

Boxplot of Caffeine Intake (mg/day)
0
100
200
300
400
500
600
Caffeine Intake (mg/day)

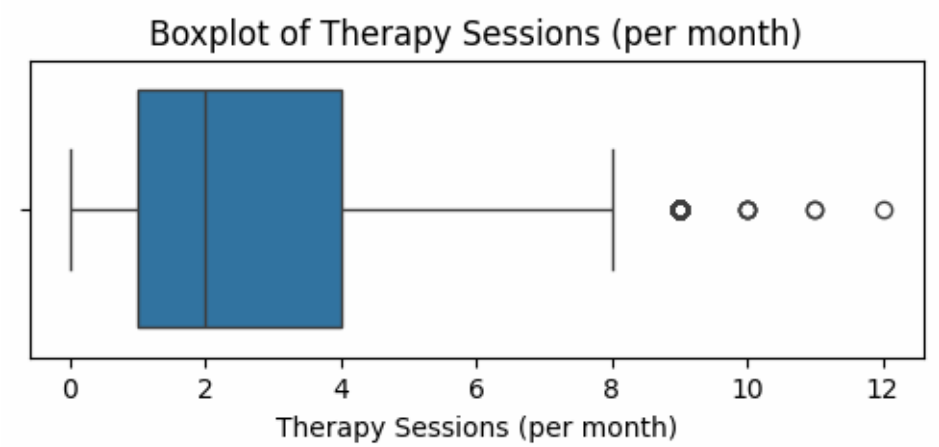

Boxplot of Alcohol Consumption (drinks/week)
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Alcohol Consumption (drinks/week)

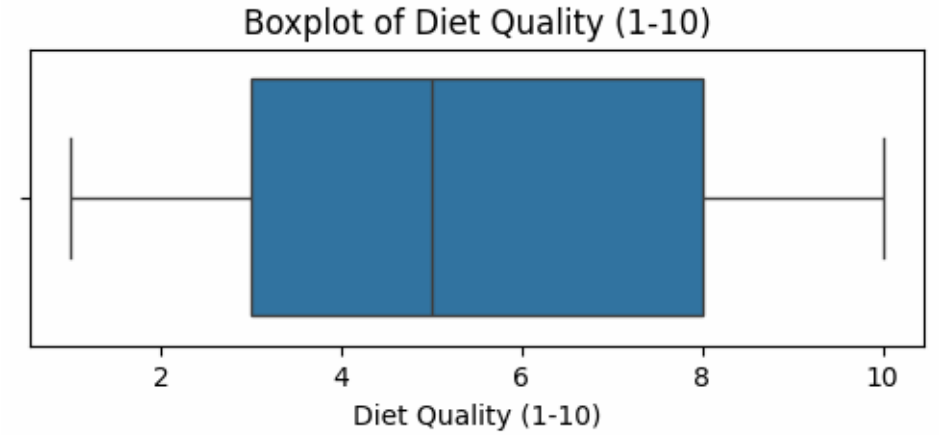

Boxplot of Stress Level (1-10)
2
4
6
8
10
Stress Level (1-10)

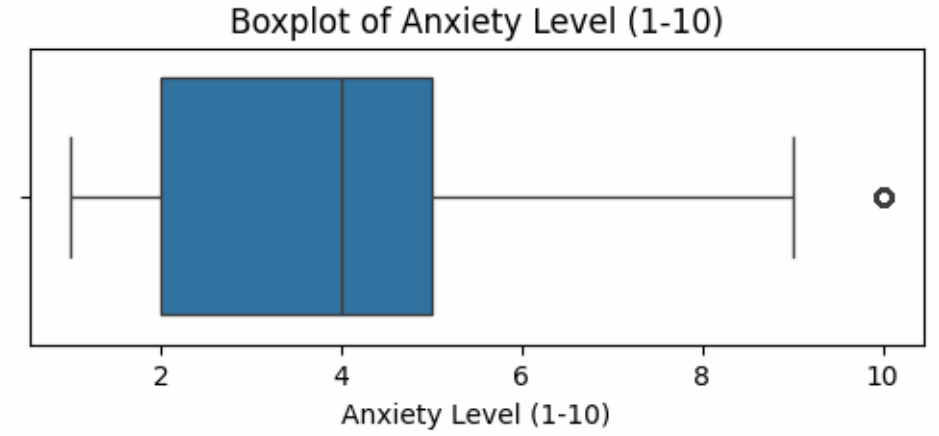

Boxplot of Heart Rate (bpm)
60
70
80
90
100
110
120
Heart Rate (bpm)


Boxplot of Breathing Rate (breaths/min)
12.5
15.0
17.5
20.0
22.5
25.0
27.5
Breathing Rate (breaths/min)


Boxplot of Sweating Level (1-5)
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
Sweating Level (1-5)


Boxplot of Therapy Sessions (per month)
0
2
4
6
8
10
12
Therapy Sessions (per month)


Boxplot of Diet Quality (1-10)
2
4
6
8
10
Diet Quality (1-10)


Boxplot of Anxiety Level (1-10)
2
4
6
8
10
Anxiety Level (1-10)

4. Apply feature scaling using Standard Scaler
5. Split the dataset into 70:30 ratio
6. Apply Linear Regression model

```
[23] from sklearn.preprocessing import StandardScaler

     target_column = 'Anxiety Level (1-10)'

     X = df.drop(target_column, axis=1)
     y = df[target_column]

     X_encoded = pd.get_dummies(X)

     scaler = StandardScaler()
     X_scaled = scaler.fit_transform(X_encoded)
```

```
[24] from sklearn.model_selection import train_test_split

     X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.3, random_state=42)
```

```
[26] from sklearn.linear_model import LinearRegression

     model = LinearRegression()
     model.fit(X_train, y_train)

     y_pred = model.predict(X_test)
```

7. Evaluate the model using MAE, RMSE and R-squared

```
[27] from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
     import numpy as np

     mae = mean_absolute_error(y_test, y_pred)
     rmse = np.sqrt(mean_squared_error(y_test, y_pred))
     r2 = r2_score(y_test, y_pred)

     print(f"Mean Absolute Error (MAE): {mae}")
     print(f"Root Mean Squared Error (RMSE): {rmse}")
     print(f"R-squared (R²): {r2}")

     Mean Absolute Error (MAE): 0.8941090700895843
     Root Mean Squared Error (RMSE): 1.122920257845281
     R-squared (R²): 0.7260476846963699
```

Lab 11 – Class Imbalance

Objective:

The objective of this lab is to equip students with practical skills to identify and address class imbalance in classification problems, a common issue in real-world data mining tasks. Students will implement and compare various resampling techniques such as random over-sampling, under-sampling, and SMOTE to balance datasets, and evaluate model performance using metrics suited for imbalanced data, including precision, recall, F1-score, and ROC-AUC. The lab aims to enhance students' understanding of the impact of imbalance on predictive models and develop competency in using tools like scikit-learn and imbalanced-learn to build more accurate classifiers.

Tools Required:

Anaconda Distribution/ Google Colab/ Pycharm

Intuition

Consider the following situation:

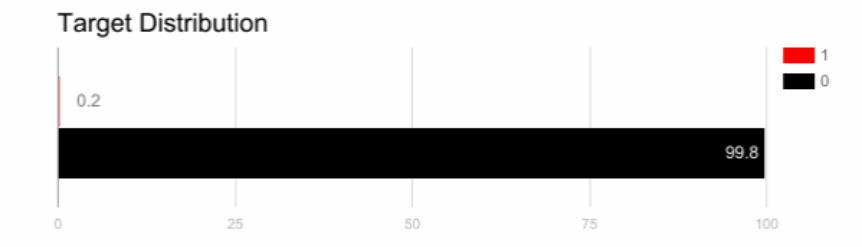
You are working on your dataset. You create a classification model and get 90% accuracy immediately. The results seem fantastic to you. You dive a little deeper and discover that almost entirety of the data belongs to one class. Damn! Imbalanced data can cause you a lot of frustration.

You feel very frustrated when you discovered that your data has imbalanced classes and that all of the great results you thought you were getting turn out to be a lie. What is even more frustrating is the good books don't even holistically cover this topic.

This is an example of a situation cases by an **imbalanced** dataset and the frustrating results it can cause.

In this tutorial, you will discover the techniques that you can use to deliver excellent results on datasets with imbalanced data. Specifically, you will cover:

- What is meant by imbalanced data?
- Why are imbalanced datasets a serious problem to tackle?
- Accuracy Paradox
- Different metrics for classifier evaluation
- Various approaches to handling imbalanced data
- Further reading on the topic



What is Imbalanced Data?

Imbalanced data typically refers to classification tasks where the classes are not represented equally.

For example, you may have a binary classification problem with 100 instances out of which 80 instances are labeled with Class-1, and the remaining 20 instances are marked with Class-2.

This is essentially an example of an imbalanced dataset, and the ratio of Class-1 to Class-2 instances is 4:1.

Be it a Kaggle competition or real test dataset, the class imbalance problem is one of the most common ones.

Most of the real-world classification problems display some level of class imbalance, which happens when there are not sufficient instances of the data that correspond to either of the class labels. Therefore, it is imperative to choose the evaluation metric of your model correctly. If it is not done, then you might end up adjusting/optimizing a useless parameter. In a real business-first scenario, this may lead to a complete waste.

There are problems where a class imbalance is not just common; it is bound to happen. For example, the datasets that deal with fraudulent and non-fraudulent transactions, it is very likely that the number of fraudulent transactions as compares to the number of non-fraudulent transactions will be very much less. And this is where the problem arises. You will study why.

Why are Imbalanced Datasets a Serious Problem to Tackle?

Although all many machine learning algorithms (both deep and statistical) have shown great success in many real-world applications, the problem of learning from **imbalanced data** is still yet to be *state-of-the-art*. And often, this learning from imbalanced data is referred to as **Imbalanced learning**.

Following are the significant problems of imbalanced learning:

- When the dataset has underrepresented data, the class distribution starts skew.
- Due to the inherent complex characteristics of the dataset, learning from such data requires new understandings, new approaches, new principles, and new tools to transform data. And moreover, this cannot anyway guarantee an efficient

solution to your business problem. In worst cases, it might turn to complete wastes with zero residues to reuse.

At this point of time one obvious question that must have come to your mind is - why in an age of GPUs, TPUs machine learning algorithms are failing to tackle imbalanced data efficiently? Quite an obvious question and you will find its answer now.

Evaluation of machine learning algorithms has a lot to do as to the reason why a particular machine learning algorithm does not perform when supplied with imbalanced data.

"*It is the case where your accuracy measures tell the story that you have excellent accuracy (such as 90%), but the accuracy is only reflecting the underlying class distribution.*" - **Machine Learning Mastery**

Suppose, you have a dataset (associated with a classification task) with two classes with a distribution ratio of 9:1. The total number of instances present in the dataset is 1000, and the class labels are Class-1 and Class-2. Therefore, w.r.t the distribution ratio, the number of instances that correspond to Class-1 is 900 while Class-2 instances are 100. Now, you applied a standard classifier (say Logistic Regression) and measured its performance concerning **classification accuracy** which gives the number of instances correctly classified by the classifier. Now, take a closer look and think very deeply.

You Logistic Regression model does not have to be very complicated to classify all the 1000 instances as Class-1. In that case, you would get a classification accuracy of 90% which is really not enough to test the actual quality of the classifier. Clearly, you need some other metric to evaluate the performance of the system. You will see that in a minute. The phenomenon you just studied is called **Accuracy Paradox**.

Approaches for Handling Imbalanced Data:

You will start this section by studying some metrics other than **classification accuracy** in order to truly judge a classifier when it is dealing with imbalanced data.

First, let's define four fundamental terms here:

- True Positive (TP) – An instance that is positive and is classified correctly as positive
- True Negative (TN) – An instance that is negative and is classified correctly as negative
- False Positive (FP) – An instance that is negative but is classified wrongly as positive
- False Negative (FN) – An instance that is positive but is classified incorrectly as negative

The following image will justify the above terms for themselves:

Name	Formula	Explanation
True Positive Rate (TP rate)	TP / (TP + FP)	The closer to 1, the better. TP rate = 1 when FP = 0. (No false positives)
True Negative Rate (TN rate)	TN / (TN + FN)	The closer to 1, the better. TN rate = 1 when FN = 0. (No false negatives)
False Positive Rate (FP rate)	FP / (FP + TN)	The closer to 0, the better. FP rate = 0 when FP = 0. (No false positives)
False Negative Rate (FN rate)	FN / (FN + TP)	The closer to 0, the better. FN rate = 0 when FN = 0. (No false negatives)

Now, assume that you trained another classifier on the toy dataset you just saw and this time you applied a Random Forest. And you got a classification accuracy of 70%. Now that, you know about True Positive and Negative Rates and False Positive and Negative Rates, you will investigate the performance of the earlier Logistic Regression and Random Forest in a bit more detailed manner.

Suppose you got the following True Positive and Negative Rates and False Positive and Negative Rates for Logistic Regression:

		Predicted Class	
		Class-1	Class-2
Actual Class	Class-1	810 (TP)	100 (FN)
	Class-2	90 (FP)	0 (TN)

Now, assume that the True Positive and Negative Rates and False Positive and Negative Rates for Random Forest are following:

		Predicted Class	
		Class-1	Class-2
Actual Class	Class-1	325	25
	Class-2	575	75

Just look at the number of negative classes correctly predicted (True Negatives) by both of the classifiers. As you are dealing with an imbalanced dataset, you need to give this number the most priority (because Class-1 dominant in the dataset). So, considering that, Random Forest trades away Logistic Regression easily.

Now, you are in an excellent place to study the approaches for combating imbalanced dataset problem.

(Remember, the above representation is popularly known as **Confusion Matrix**.)

Following are the two terms that are derived from the confusion matrix and very much used when you are evaluating a classifier.

Precision: Precision is the number of True Positives divided by the number of True Positives and False Positives. Put another way; it is the number of positive predictions divided by the total number of positive class values predicted. It is also called the Positive Predictive Value (PPV).

Precision can be thought of as a measure of a classifier's *exactness*. A low precision can also indicate a large number of False Positives.

Recall: Recall is the number of True Positives divided by the number of True Positives and the number of False Negatives. Put another way it is the number of positive predictions divided by the number of positive class values in the test data. It is also called Sensitivity or the True Positive Rate.

Recall can be thought of as a measure of a classifier's *completeness*. A low recall indicates many False Negatives.

Some other metrics that can be useful in this context:

- AUC
- ROC Curve
- f1-Score
- Matthews correlation coefficient (MCC)

The **breast cancer dataset** is a standard machine learning dataset. It contains 9 attributes describing 286 women that have suffered and survived breast cancer and whether or not breast cancer recurred within 5 years. Let's investigate this dataset to get you a real feel of the problem.

The dataset concerns a binary classification problem. Of the 286 women, 201 did not suffer a recurrence of breast cancer, leaving the remaining 85 that did.

Let's explore about the dataset more visually.

```
import numpy as np
import pandas as pd

# Load the dataset into a pandas dataframe
data = pd.read_csv("breast-cancer.data",header=None)

# See the data
print(data.head(10))
```

	0	1	2	3	4	5	6	7	8
0	no-recurrence-events	30-39	premeno	30-34	0-2	no	3	left	left_low
1	no-recurrence-events	40-49	premeno	20-24	0-2	no	2	right	right_up
2	no-recurrence-events	40-49	premeno	20-24	0-2	no	2	left	left_low
3	no-recurrence-events	60-69	ge40	15-19	0-2	no	2	right	left_up
4	no-recurrence-events	40-49	premeno	0-4	0-2	no	2	right	right_low
5	no-recurrence-events	60-69	ge40	15-19	0-2	no	2	left	left_low
6	no-recurrence-events	50-59	premeno	25-29	0-2	no	2	left	left_low
7	no-recurrence-events	60-69	ge40	20-24	0-2	no	1	left	left_low
8	no-recurrence-events	40-49	premeno	50-54	0-2	no	2	left	left_low
9	no-recurrence-events	40-49	premeno	20-24	0-2	no	2	right	left_up

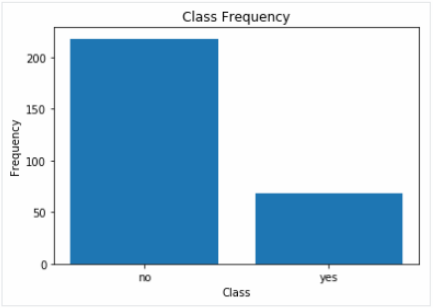
	9
0	no
1	no
2	no
3	no
4	no
5	no
6	no
7	no
8	no
9	no

The column names are numeric because you are using a partially preprocessed dataset. But if you are interested then you may refer to the following image:

1. Class: no-recurrence-events, recurrence-events
2. age: 10-19, 20-29, 30-39, 40-49, 50-59, 60-69, 70-79, 80-89, 90-99.
3. menopause: lt40, ge40, premeno.
4. tumor-size: 0-4, 5-9, 10-14, 15-19, 20-24, 25-29, 30-34, 35-39, 40-44, 45-49, 50-54, 55-59.
5. inv-nodes: 0-2, 3-5, 6-8, 9-11, 12-14, 15-17, 18-20, 21-23, 24-26, 27-29, 30-32, 33-35, 36-39.
6. node-caps: yes, no.
7. deg-malig: 1, 2, 3.
8. breast: left, right.
9. breast-quad: left-up, left-low, right-up, right-low, central.
10. irradiat: yes, no.

Let's see a bar graph of the class distributions;

```
import matplotlib.pyplot as plt

classes = data[9].values
unique, counts = np.unique(classes, return_counts=True)

plt.bar(unique,counts)
plt.title('Class Frequency')
plt.xlabel('Class')
plt.ylabel('Frequency')
plt.show()
```



You can clearly see the class imbalance here. yes denotes the instances which have cancer and is obvious, the number of these instances is minimal as compared to the instances corresponding to the other class.

Let's define "No Recurrences" and "Recurrences" event that is there in the dataset which will make things even more evident.

This concept should have sparked an ignition inside you by now. Let's move forward with it.

Well! Now, you have now enough reasons to wonder why considering only classification accuracy to evaluate your classification model is not a good choice.

Let's study some approaches now.

Re-sampling the dataset:

Dealing with imbalanced datasets includes various strategies such as improving classification algorithms or balancing classes in the training data (essentially a data preprocessing step) before providing the data as input to the machine learning algorithm. The latter technique is preferred as it has broader application and adaptation. Moreover, the time taken to enhance an algorithm is often higher than to generate the required samples. But for research purposes, both are preferred.

The main idea of sampling classes is to either increasing the samples of the minority class or decreasing the samples of the majority class. This is done in order to obtain a fair balance in the number of instances for both the classes.

There can be two main types of sampling:

- You can add copies of instances from the minority class which is called **over-sampling** (or more formally sampling with replacement), or
- You can delete instances from the majority class, which is called **under-sampling**.

This sounds even easier from the implementation perspective as well. Isn't it? Later in this post, you will get to know about a library dedicated to performing sampling.

Random under-sampling:

When you randomly eliminate instances from the majority class of a dataset and assign it to the minority class (without filling out the void created in majority class), it is known as **random under-sampling**. The void that gets created in the majority dataset for this makes the process *random*.

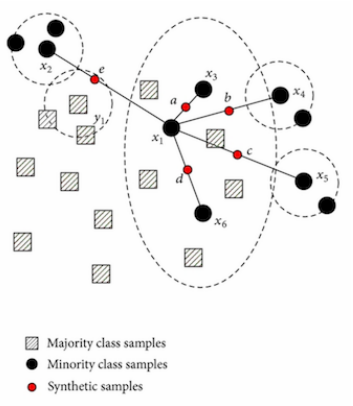
Random over-sampling:

Just like random under-sampling, you can perform random oversampling as well. But in this case, taking any help from the majority class, you increase the instances corresponding to the minority class by replicating them up to a constant degree. In this case, you do not decrease the number of instances assigned to the majority class. Say, you have a dataset with 1000 instances where 980 instances correspond to the majority class, and the reaming 20 instances correspond to the minority class. Now you over-sample the dataset by replicating the 20 instances up to 20 times. As a result, after performing over-sampling the total number of instances in the minority class will be 400.

Try generating synthetic samples:

A simple way to create synthetic samples is to sample the attributes from instances in the minority class randomly.

There are systematic algorithms that you can use to generate synthetic samples. The most popular of such algorithms is called SMOTE or the Synthetic Minority Over-sampling Technique. It was proposed in 2002, and you can take a look at the following info-graphic will give you a fair idea about the synthetic samples:



SMOTE is an oversampling method which creates “synthetic” example rather than oversampling by replacements. The minority class is over-sampled by taking each minority class sample and introducing synthetic examples along the line segments joining any/all of the k minority class nearest neighbors. Depending upon the amount of over-sampling required, neighbors from the k nearest neighbors are randomly chosen.

The heart of SMOTE is the construction of the minority classes. The intuition behind the construction algorithm is simple. You have already studied that oversampling causes overfitting, and because of repeated instances, the decision boundary gets tightened. What if you could generate **similar** samples instead of repeating them? In the original SMOTE paper (linked above) it has been shown that to a machine learning algorithm, these newly constructed instances are not exact copies, and thus it softens the decision boundary and thereby helping the algorithm to approximate the hypothesis more accurately.

Exercise

1. Load the “creditcard_fraud.csv” data

```
import pandas as pd

data = pd.read_csv("/content/creditcard_fraud.csv")
data.head()
```

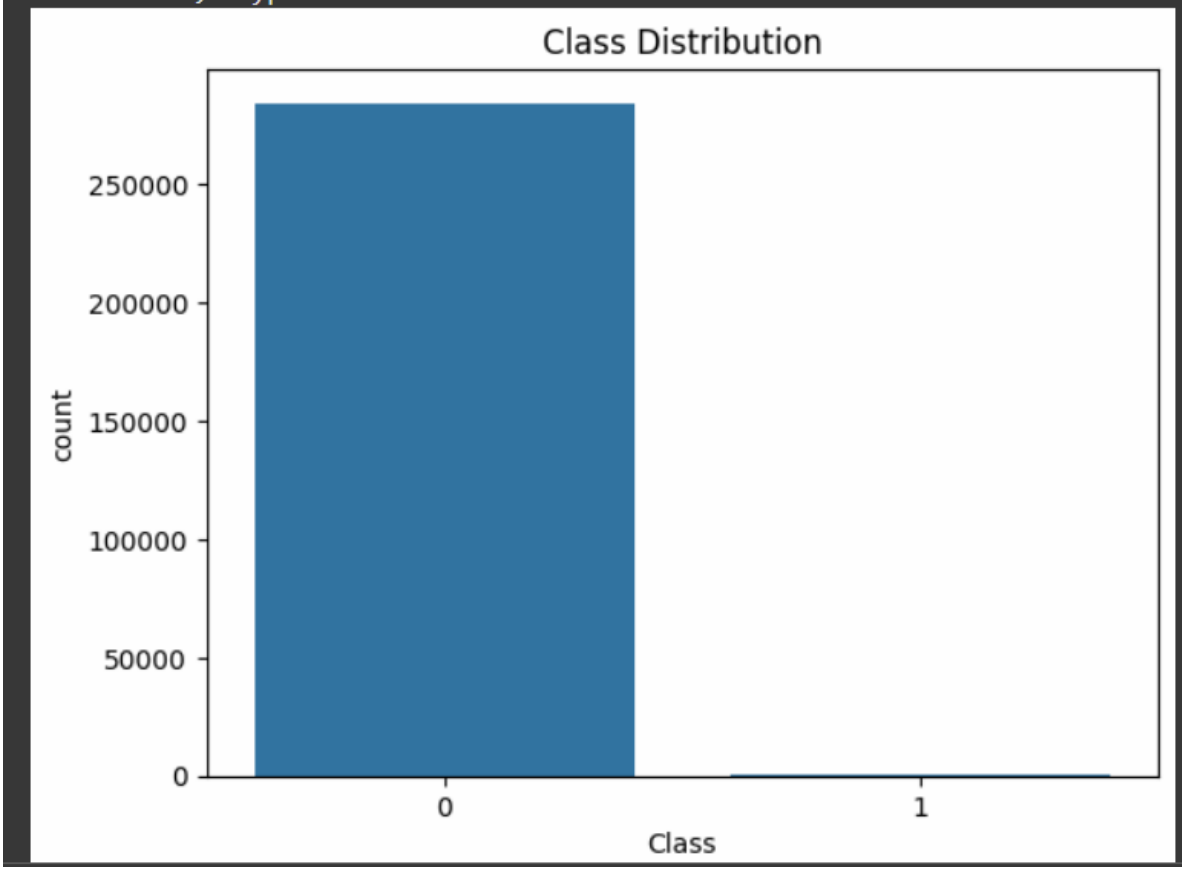
	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.110474	0.066928
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.638672	0.101288	-0.339846
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.771679	0.909412	-0.689281
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.005274	-0.190321	-1.175575
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.798278	-0.137458	0.141267

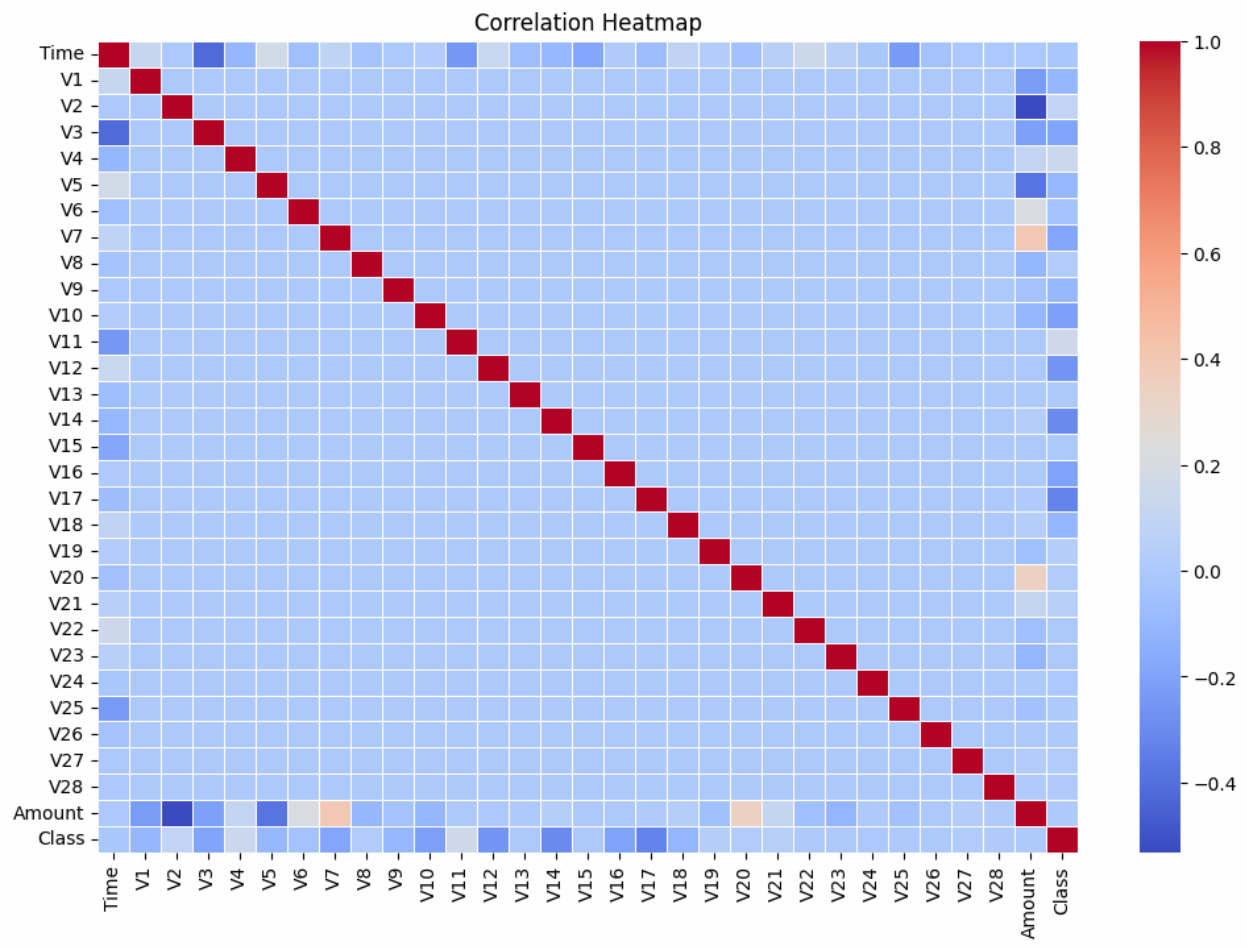
5 rows × 31 columns

2. Perform Exploratory Data Analysis

```
# Summary statistics
data.describe()

# Class distribution
print(data['Class'].value_counts())

# Visualizing the class distribution
import seaborn as sns
import matplotlib.pyplot as plt

sns.countplot(x='Class', data=data)
plt.title('Class Distribution')
plt.show()

# Correlation matrix
plt.figure(figsize=(12,8))
sns.heatmap(data.corr(), cmap="coolwarm", linewidths=0.5)
plt.title("Correlation Heatmap")
plt.show()
```

```
Class
0    284315
1       492
Name: count, dtype: int64
```

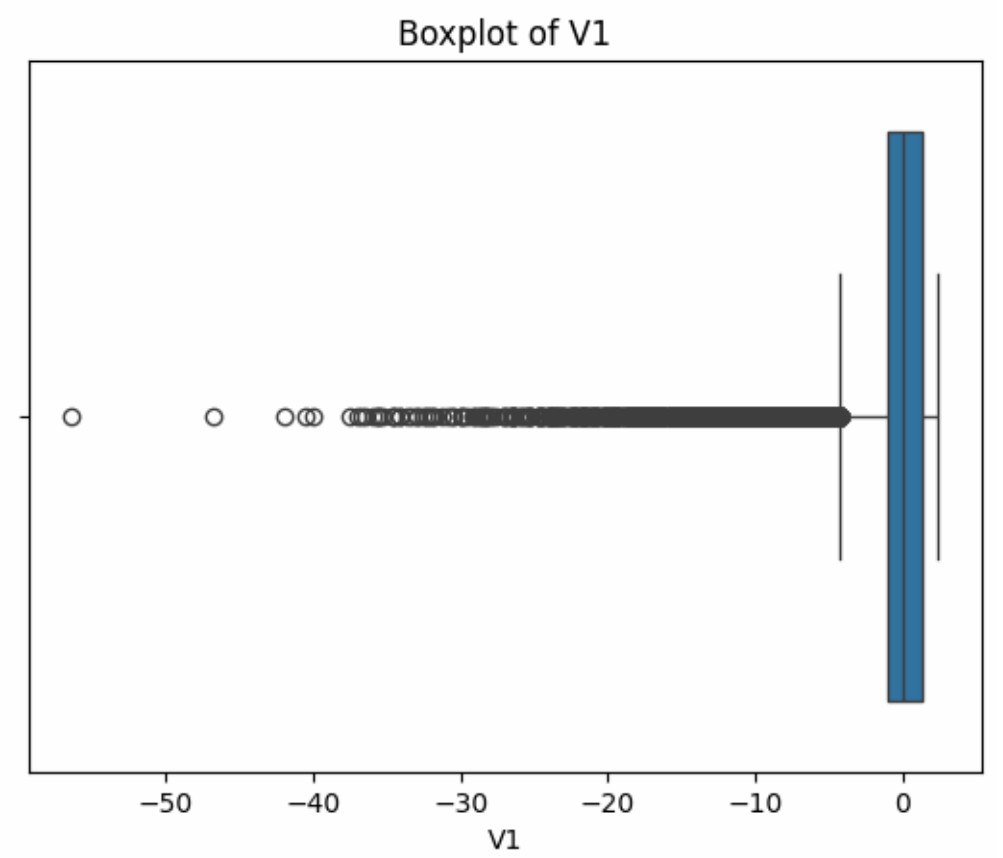


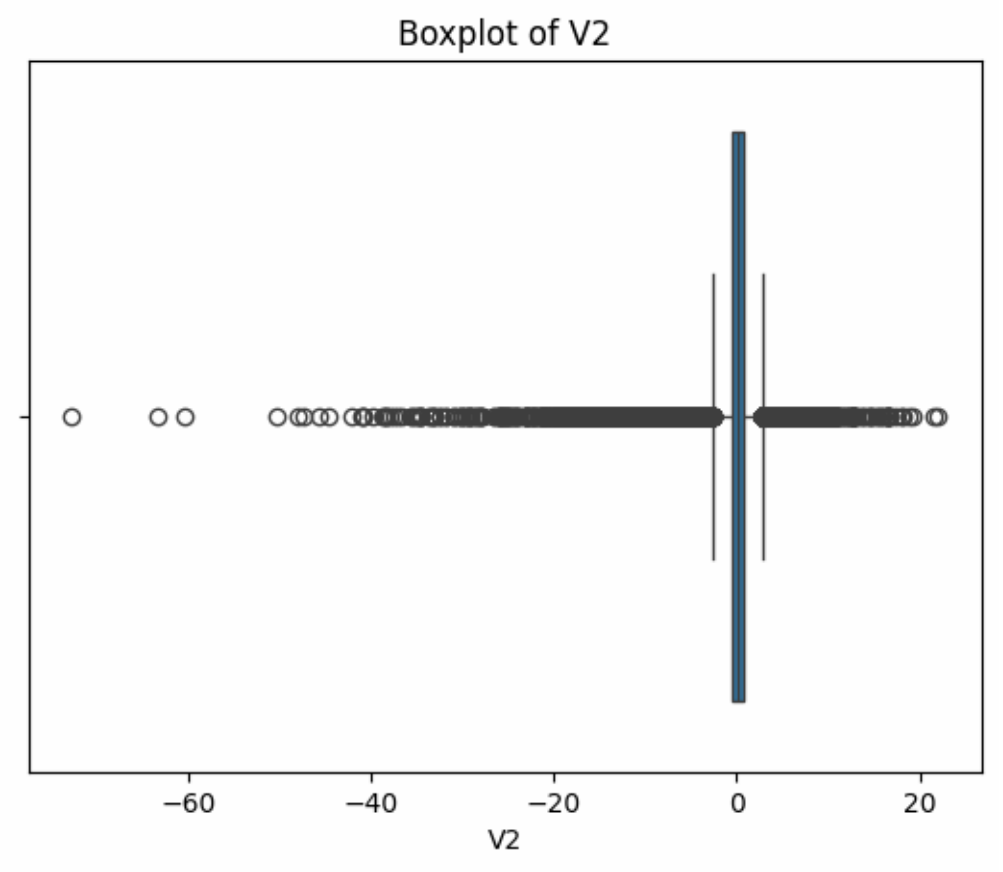


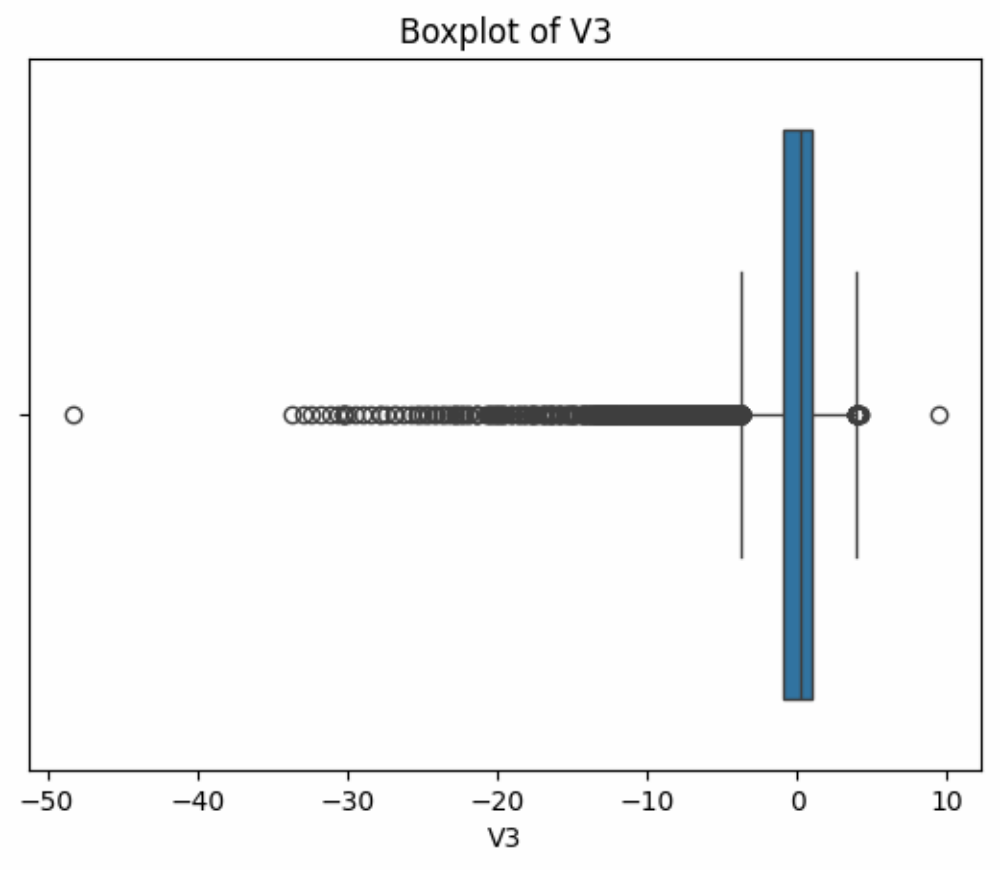
3. Perform the data preprocessing techniques such as identify outliers, missing values

```python
# Check for missing values
print(data.isnull().sum())

# Outlier detection using boxplots
for column in ['V1', 'V2', 'V3']:  # example features
    sns.boxplot(x=data[column])
    plt.title(f"Boxplot of {column}")
    plt.show()
```

```
Time      0
V1        0
V2        0
V3        0
V4        0
V5        0
V6        0
V7        0
V8        0
V9        0
V10       0
V11       0
V12       0
V13       0
V14       0
V15       0
V16       0
V17       0
V18       0
V19       0
V20       0
V21       0
V22       0
V23       0
V24       0
V25       0
V26       0
V27       0
V28       0
Amount    0
Class     0
dtype: int64
```


Boxplot of V1
−50
−40
−30
−20
−10
0
V1


Boxplot of V2
−60
−40
−20
0
20
V2



4. What are the distribution of classes?

```
class_dist = data['Class'].value_counts(normalize=True)
print("Class 0 (Not Fraud):", class_dist[0])
print("Class 1 (Fraud):", class_dist[1])
```

```
Class 0 (Not Fraud): 0.9982725143693799
Class 1 (Fraud): 0.0017274856306200 34
```

5. Apply feature scaling using Standard Scaler

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
scaled_features = scaler.fit_transform(data.drop(['Class'], axis=1))

X = pd.DataFrame(scaled_features, columns=data.columns[:-1])
y = data['Class']
```

6. Split the dataset into 70:30 ratio

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42, stratify=y
)
```

7. Apply Decision tree, Random forest, K-Nearest Neighbours, Support Vector Classifier, and Gaussian Naïve Bayes Classifier
8. Evaluate the model using precision score?

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import precision_score

models = {
    "Decision Tree": DecisionTreeClassifier(),
    "Random Forest": RandomForestClassifier(),
    "KNN": KNeighborsClassifier(),
    "SVM": SVC(),
    "Gaussian NB": GaussianNB()
}

precision_scores = {}

for name, model in models.items():
    model.fit(X_train, y_train)
    preds = model.predict(X_test)
    precision = precision_score(y_test, preds)
    precision_scores[name] = precision
    print(f"{name} Precision Score: {precision}")
```

```
Decision Tree Precision Score: 0.7669172932330827
Random Forest Precision Score: 0.957983193277311
KNN Precision Score: 0.9159663865546218
SVM Precision Score: 0.967741935483871
Gaussian NB Precision Score: 0.06043676993397664
```

9. Which model is getting the highest precision score?

```
best_model = max(precision_scores, key=precision_scores.get)
print(f"Best Model (Before SMOTE): {best_model} with Precision Score: {precision_scores[best_model]}")
```

```
Best Model (Before SMOTE): SVM with Precision Score: 0.967741935483871
```

10. Apply the SMOTE method to deal with the class imbalance problem.

```
from imblearn.over_sampling import SMOTE

smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(X, y)

# Split the resampled data
X_train_res, X_test_res, y_train_res, y_test_res = train_test_split(
    X_resampled, y_resampled, test_size=0.3, random_state=42
)
```

11. After SMOTE technique which model is getting the highest precision score?

```
precision_scores_smote = {}

for name, model in models.items():
    model.fit(X_train_res, y_train_res)
    preds = model.predict(X_test_res)
    precision = precision_score(y_test_res, preds)
    precision_scores_smote[name] = precision
    print(f"{name} Precision Score (After SMOTE): {precision}")

best_model_smote = max(precision_scores_smote, key=precision_scores_smote.get)
print(f"Best Model (After SMOTE): {best_model_smote} with Precision Score: {precision_scores_smote[best_model_smote]}")
```

```
Decision Tree Precision Score (After SMOTE): 0.9968694880211193
Random Forest Precision Score (After SMOTE): 0.9998010695437471
KNN Precision Score (After SMOTE): 0.9976995924658734
```