Practical 1 - Intro

Question 1
What is the duration of our 5-day workshop in minutes?
Answer:

```
# What is the duration of our 5-day workshop in minutes?
workshop__total_hour = 7 #9am - 5pm, minus 1-hour Lunch break
workshop_in_minutes = workshop__total_hour * 60
duration = workshop_in_minutes * 5
print("The duration of 5-day workshop", duration, "minutes")
```

```
The duration of 5-day workshop 2100 minutes
```

Question 2
Find x in -x + -19.4 = 1844.12 + 2x.
Answer:

```
from sympy import symbols, solve

x = symbols('x')
expr = 1844.12 + 2*x + x + 19.4
solution = solve(expr)

print(solution)
```

```
[-621.173333333333]
```

Question 3
How far does light travel in the time that your computer does one cycle?
Assuming that the computer is processing at a frequency of 4GHz.
Answer:

```python
speed_of_light = 299792458  # meters per second
frequency = 4 * 10**9  # 4 GHz
time = 1 / frequency

distance = speed_of_light * time

print("Distance light travels in one cycle of the computer:", distance, "meters")

# ans approx 0.07500000000000001
```

```
Distance light travels in one cycle of the computer: 0.07494811450000001 meters
```

Question 4
Write a function to find the square root of a number.
Answer:

```python
def squareroot(num):
    # your code
    # cant use math.sqrt(num)
    guess = num / 2  # Initial guess
    while True:
        new_guess = (guess + num / guess) / 2
        if abs(new_guess - guess) < 1e-6:  # Check for convergence
            return new_guess
        guess = new_guess

a = 100
print(squareroot(a))
```

```
10.0
```

Question 5
What is the purpose of using List Comprehensions?
Answer:

List comprehension is to simplify and shorten the code, to make it more readable. It is similar to for loop, and it will return an output in every iteration.

Syntax:

```
new_list = [expression for item in iterable if condition]
```

Example 1:

```python
numbers = [1, 2, 3, 4, 5, 6, 7, 8]
squares = [n*n for n in numbers]

print(squares)
```

```
[1, 4, 9, 16, 25, 36, 49, 64]
```

Example 2:

```python
numbers = [1, 2, 3, 4, 5]
# Create a new list with squares of even numbers from the original list
squared_even = [x**2 for x in numbers if x % 2 == 0]


print(squared_even)  # Output: [4, 16]
```

```
[4, 16]
```

Practical 2 - NumPy

In [1]:
```python
import numpy as np
```

Question 1

1. Create a 3x3 matrix with values ranging from 0 to 8

In [2]:
```python
# Q1
q1 = np.arange(9).reshape(3, 3)
print("q1 is")
print(q1)

# Sample answer:
# a1 is [[0 1 2]
# [3 4 5]
# [6 7 8]]
```

```
q1 is
[[0 1 2]
 [3 4 5]
 [6 7 8]]
```

Question 2

1. Create a 10x10 array with random values and find the minimum and maximum values

In [3]:
```python
# Q2
q2 = np.random.random((10, 10))
# print(q2)
print("max value of q2 is", np.max(q2))
print("min value of q2 is", np.min(q2))

# Sample answer:
# max value of a2 is 0.9992937628379465
# min value of a2 is 0.012441575894276191
```

```
max value of q2 is 0.9902116553624828
min value of q2 is 0.03288893521020175
```

Question 3

1. Create a 8x8 matrix and fill it with a checkerboard pattern

In [4]:
```python
# Q3
q3 = np.zeros((8, 8))
q3[1::2, ::2] = 1
q3[::2, 1::2] = 1
print("q3 is")
print(q3)

# Sample answer (!!wrong!!):
```

```
# a3 is
# [[1. 1. 1. 1. 1. 1. 1. 1.]
# [0. 0. 0. 0. 0. 0. 0. 0.]
# [1. 1. 1. 1. 1. 1. 1. 1.]
# [0. 0. 0. 0. 0. 0. 0. 0.]
# [1. 1. 1. 1. 1. 1. 1. 1.]
# [0. 0. 0. 0. 0. 0. 0. 0.]
# [1. 1. 1. 1. 1. 1. 1. 1.]
# [0. 0. 0. 0. 0. 0. 0. 0.]]
```

```
q3 is
[[0. 1. 0. 1. 0. 1. 0. 1.]
 [1. 0. 1. 0. 1. 0. 1. 0.]
 [0. 1. 0. 1. 0. 1. 0. 1.]
 [1. 0. 1. 0. 1. 0. 1. 0.]
 [0. 1. 0. 1. 0. 1. 0. 1.]
 [1. 0. 1. 0. 1. 0. 1. 0.]
 [0. 1. 0. 1. 0. 1. 0. 1.]
 [1. 0. 1. 0. 1. 0. 1. 0.]]
```

Question 4

1. Create random vector of size 10 and replace the maximum value by 0

In [5]:
```
# Q4
q4 = np.random.rand(10)
print("original q4 is")
print(q4)
max_index = np.argmax(q4)
q4[max_index] = 0
print("new q4 is")
print(q4)

# Sample answer:
# original a4 is
# [0.32642686 0.842169 0.3270934 0.76815538 0.72293528
0.72772311 # 0.31843191 0.41676299 0.0269001 0.44165806]
# new a4 is
# [0.32642686 0. 0.3270934 0.76815538 0.72293528 0.72772311 #
0.31843191 0.41676299 0.0269001 0.44165806]
```

```
original q4 is
[0.95499748 0.82597215 0.77841279 0.71767224 0.95954109
 0.79457117 0.24178283 0.64083669 0.53328774 0.45344192]
new q4 is
[0.95499748 0.82597215 0.77841279 0.71767224 0. 0.79457117
 0.24178283 0.64083669 0.53328774 0.45344192]
```

Question 5

1. Create a $4 * 4$ identity matrix.

In [6]:
```
# Q5
q5 = np.eye(4)
```

```
print("matrix identity q5 is")
print(q5)

# Sample answer:
# matrix identity a5 is
# [[1. 0. 0. 0.]
# [0. 1. 0. 0.]
# [0. 0. 1. 0.]
# [0. 0. 0. 1.]]
```

```
matrix identity q5 is
[[1. 0. 0. 0.]
 [0. 1. 0. 0.]
 [0. 0. 1. 0.]
 [0. 0. 0. 1.]]
```

Question 6

1. Generate the 2D array

In [7]:
```
# Q6
q6 = np.random.rand(3, 3)
print("2-D array q6 is")
print(q6)

# Sample answer:
# 2-D array a6 is
# [[0.51586491 0.66398948 0.97664544]
# [0.04570482 0.89286948 0.96746235]
# [0.58957188 0.62527391 0.15207168]]
```

```
2-D array q6 is
[[0.29299351 0.6801015 0.64070941]
 [0.22896768 0.96508168 0.04578627]
 [0.70248292 0.07067561 0.12894765]]
```

Question 7

$$4 \times 4 \times 4$$

1. Generate a random array of Gaussianly (Normal) distributed numbers.

In [8]:
```
# Q7
q7 = np.random.randn(4, 4, 4)
print("3-D array q7 is")
print(q7)

# Sample answer:
# 3-D array a7 is
# [[[ 1.24532934 -0.21856698 -0.73979455 0.97734318]
# [-0.39611546 2.08889299 1.5800086 -1.49495715]
# [-0.4902839 -0.32530276 0.36206673 -1.81680732]
# [-0.78322585 0.77552733 0.10999834 -1.99478335]]

# [[ 0.42654637 -2.00054199 -0.73081536 1.5650001 ]
# [ 0.52327309 0.44793719 -0.69319204 -0.50252064]
# [ 0.75281488 -0.98192208 -1.09342367 -0.16395248]
```

```
# [-0.39731477 0.8379485 0.28013044 0.46306382]]

# [[-0.37884152 -0.76614664 -0.70402131 -0.20758354]
# [-1.08024839 -0.23268198 -1.57187888 -0.06078278]
# [-1.20464148 -0.08984497 -0.47294482 -0.68721336]
# [ 1.32338635 -1.04715013 -0.619475 -0.83355595]]

# [[ 0.40107191 0.43863124 1.61187035 -1.19535028]
# [ 0.95272223 -1.06955353 -0.78299704 1.51630909]
# [-0.74651708 0.84885255 1.47740253 0.06694602]
# [ 0.4545381 -1.94715794 -1.0914511 0.5649201 ]]]
```

```
3-D array q7 is
[[[-1.81531699e+00 -1.04457366e+00 7.54664969e-01 -3.33505862e-01]
  [-5.67523176e-01 -5.97250656e-01 1.93564977e+00 8.02507882e-01]
  [ 4.48818835e-01 1.26089059e+00 7.62945664e-02 2.09293006e+00]
  [ 6.33947907e-02 -1.01832785e-01 7.44216440e-01 1.81028246e+00]]

 [[ 3.22518455e-01 7.72158524e-01 -8.65445944e-01 -1.19624417e+00]
  [-3.77440508e-01 1.92030315e+00 -9.50797763e-01 -2.14669049e-01]
  [-5.92813264e-02 -1.00317572e-01 -5.70142442e-02 3.39354075e-01]
  [ 1.37640543e+00 7.32334690e-01 -1.18050263e+00 -3.84516477e-01]]

 [[ 5.45419323e-01 7.91991217e-01 8.37163501e-01 1.12767059e+00]
  [ 2.14693785e-01 1.69217186e-01 1.96609625e-03 7.58260193e-01]
  [ 6.99284174e-02 -5.37951235e-01 1.02706257e+00 -6.30223690e-01]
  [ 4.29453040e-01 -1.85204504e+00 1.55170499e+00 -2.74745038e-02]]

 [[ 2.10151459e-01 -1.24571114e+00 2.49989333e-01 1.06552764e+00]
  [-6.15017001e-02 2.17628515e+00 -4.44976476e-01 -1.89893089e-01]
  [-5.97328246e-01 -9.44290973e-01 -2.32800075e+00 6.85795506e-01]
  [ 1.01248907e+00 -7.02785314e-01 1.21018612e+00 -1.40230240e+00]]]
```

Question 8

1. Generate `n` evenly spaced intervals between 0. and 1.

In [9]:
```
# Q8
# q8 = np.arange(0,10,1) # this is for integer
q8 = np.linspace(0, 1, 11) # this is for non-integer step, such as 0.1
# np.linspace(start, stop, num),
# num: Number of samples to generate. Default is 50. Must be non-negative. #
reference:
https://numpy.org/doc/stable/reference/generated/numpy.linspace.html#numpy.

print("q8 is")
print(q8)

# Sample answer:
# a8 is
# [0. 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1. ]
```

```
q8 is
[0. 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1. ]
```

Question 9

1. Create a vector and then reverse the vector (first element becomes last)

```
In [10]: # Q9
q9 = np.arange(10)
print("q9 is")
print(q9)
print("the reverse of q9 is")
print(q9[::-1])

# Sample answer:
# a9 is
# [0 1 2 3 4 5 6 7 8 9]
# the reverse of a9 is
# [9 8 7 6 5 4 3 2 1 0]
```

```
q9 is
[0 1 2 3 4 5 6 7 8 9]
the reverse of q9 is
[9 8 7 6 5 4 3 2 1 0]
```

Practical 3 - Pandas

Answer **ALL** questions.

Q1: show the top 5 rows of data

Code:

```
ufo.head(5)
```

Output:

	City	Colors Reported	Shape Reported	State	Time
0	Ithaca	NaN	TRIANGLE	NY	6/1/1930 22:00
1	Willingboro	NaN	OTHER	NJ	6/30/1930 20:00
2	Holyoke	NaN	OVAL	CO	2/15/1931 14:00
3	Abilene	NaN	DISK	KS	6/1/1931 13:00
4	New York Worlds Fair	NaN	LIGHT	NY	4/18/1933 19:00

Q2: show the last 10 rows of data

Code:

```
ufo.tail(10)
```

Output:

	City	Colors Reported	Shape Reported	State	Time
18231	Pismo Beach	NaN	OVAL	CA	12/31/2000 20:00
18232	Lodi	NaN	NaN	WI	12/31/2000 20:30
18233	Anchorage	RED	VARIOUS	AK	12/31/2000 21:00
18234	Capitola	NaN	TRIANGLE	CA	12/31/2000 22:00
18235	Fountain Hills	NaN	NaN	AZ	12/31/2000 23:00
18236	Grant Park	NaN	TRIANGLE	IL	12/31/2000 23:00
18237	Spirit Lake	NaN	DISK	IA	12/31/2000 23:00
18238	Eagle River	NaN	NaN	WI	12/31/2000 23:45
18239	Eagle River	RED	LIGHT	WI	12/31/2000 23:45
18240	Ybor	NaN	OVAL	FL	12/31/2000 23:59

Q3: check the data type

Code:

```
type(ufo)
```

Output:

```
pandas.core.frame.DataFrame
```

Q4: #show all rows for the column 'City' , and the unique cities

Code:

All rows:	Unique cities:
`ufo['City']`	`pd.Series(ufo['City'].unique(), name='City')`

Output:

All rows:

```
0                  Ithaca
1             Willingboro
2                 Holyoke
3                 Abilene
4        New York Worlds Fair
                 ...
18236          Grant Park
18237         Spirit Lake
18238         Eagle River
18239         Eagle River
18240                Ybor
Name: City, Length: 18241, dtype: object
```

Unique cities:

```
0                  Ithaca
1             Willingboro
2                 Holyoke
3                 Abilene
4        New York Worlds Fair
                 ...
6472       Albrightsville
6473              Eufaula
6474             Capitola
6475           Grant Park
6476                 Ybor
Name: City, Length: 6477, dtype: object
```

Q5: determine the shape (dimension) of the data

Code:

```
ufo.shape
```

Output:

```
(18241, 5)
```

Q6: show all data for 'City' that starts with 'E'

Code:

```
ufo[ufo['City'].str.startswith('E', na=False)]
```

Output:

	City	Colors Reported	Shape Reported	State	Time
8	Eklutna	NaN	CIGAR	AK	10/15/1936 17:00
55	Espanola	NaN	CIRCLE	NM	6/1/1947 17:00
109	Excelsior	NaN	CIRCLE	MN	8/15/1949 0:00
140	East Palestine	NaN	LIGHT	OH	7/10/1950 20:30
179	Evergreen	NaN	DISK	CO	6/6/1952 13:00
...	...	...	...	...	...
18182	Evansville	NaN	FIREBALL	IN	12/24/2000 20:00
18215	El Campo	NaN	OTHER	TX	12/29/2000 9:00
18224	Eufaula	NaN	DISK	OK	12/29/2000 23:30
18238	Eagle River	NaN	NaN	WI	12/31/2000 23:45
18239	Eagle River	RED	LIGHT	WI	12/31/2000 23:45

557 rows × 5 columns

Q7: count the number of reported cases for 'LIGHT'

Code:

```
ufo[ufo['Shape Reported'] == 'LIGHT'].shape[0]
```

Output:

```
2803
```

Q8: count the number of shape reported, group by state and city

Code:

```
ufo.groupby(['State', 'City'])['Shape Reported'].count()
```

Output:

```
State  City
AK     Adak                          1
       Alaska                        2
       Anchorage                    12
       Arctic                        1
       Auke Bay                      2
                                    ..
WY     Ten Sleep                     1
       Wheeling                      0
       Wyoming                       2
       Yellowstone National Park     1
       Yellowstone Park              1
Name: Shape Reported, Length: 8029, dtype: int64
```

Practical 4 - sklearn

Read the data

Code Snippet:

```
#read the dataset
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split

test_data_original = pd.read_csv('leaf_test.csv')
test_ids = test_data_original['id']
train_data = pd.read_csv('leaf_train.csv').drop('id',axis=1)
test_data = pd.read_csv('leaf_test.csv').drop('id',axis=1)
# always drop id if the dataset has it.
```

Code Snippet:

```
train_data
```

Output:

	species	margin1	margin2	margin3	margin4	margin5	margin6	margin7	margin8	margin9	...	texture55	texture56	texture57	texture!
0	Acer_Opalus	0.007812	0.023438	0.023438	0.003906	0.011719	0.009766	0.027344	0.0	0.001953	...	0.007812	0.000000	0.002930	0.0029:
1	Pterocarya_Stenoptera	0.005859	0.000000	0.031250	0.015625	0.025391	0.001953	0.019531	0.0	0.000000	...	0.000977	0.000000	0.000000	0.0009
2	Quercus_Hartwissiana	0.005859	0.009766	0.019531	0.007812	0.003906	0.005859	0.068359	0.0	0.000000	...	0.154300	0.000000	0.005859	0.0009
3	Tilia_Tomentosa	0.000000	0.003906	0.023438	0.005859	0.021484	0.019531	0.023438	0.0	0.013672	...	0.000000	0.000977	0.000000	0.0000
4	Quercus_Variabilis	0.005859	0.003906	0.048828	0.009766	0.013672	0.015625	0.005859	0.0	0.000000	...	0.096680	0.000000	0.021484	0.0000
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
985	Magnolia_Salicifolia	0.060547	0.119140	0.007812	0.003906	0.000000	0.148440	0.017578	0.0	0.001953	...	0.242190	0.000000	0.034180	0.0000
986	Acer_Pictum	0.001953	0.003906	0.021484	0.107420	0.001953	0.000000	0.000000	0.0	0.029297	...	0.170900	0.000000	0.018555	0.0000
987	Alnus_Maximowiczii	0.001953	0.003906	0.000000	0.021484	0.078125	0.003906	0.007812	0.0	0.003906	...	0.004883	0.000977	0.004883	0.0273
988	Quercus_Rubra	0.000000	0.000000	0.046875	0.056641	0.009766	0.000000	0.000000	0.0	0.037109	...	0.083008	0.030273	0.000977	0.0029:
989	Quercus_Afares	0.023438	0.019531	0.031250	0.015625	0.005859	0.019531	0.035156	0.0	0.003906	...	0.000000	0.000000	0.002930	0.0000

990 rows × 193 columns

Code Snippet:

```
test_data
```

Output:

	margin1	margin2	margin3	margin4	margin5	margin6	margin7	margin8	margin9	margin10	...	texture55	texture56	texture57	texture58	texture5
0	0.019531	0.009766	0.078125	0.011719	0.003906	0.015625	0.005859	0.000000	0.005859	0.023438	...	0.006836	0.000000	0.015625	0.000977	0.01562
1	0.007812	0.005859	0.064453	0.009766	0.003906	0.013672	0.007812	0.000000	0.033203	0.023438	...	0.000000	0.000000	0.006836	0.001953	0.01367
2	0.000000	0.000000	0.001953	0.021484	0.041016	0.000000	0.023438	0.000000	0.011719	0.005859	...	0.128910	0.000000	0.000977	0.000000	0.0000
3	0.000000	0.000000	0.009766	0.011719	0.017578	0.000000	0.003906	0.000000	0.003906	0.001953	...	0.012695	0.015625	0.002930	0.036133	0.01367
4	0.001953	0.000000	0.015625	0.009766	0.039062	0.000000	0.009766	0.000000	0.005859	0.000000	...	0.000000	0.042969	0.016602	0.010742	0.04101
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
589	0.000000	0.000000	0.003906	0.015625	0.041016	0.000000	0.017578	0.000000	0.005859	0.013672	...	0.098633	0.000000	0.004883	0.000000	0.0039
590	0.000000	0.003906	0.003906	0.005859	0.017578	0.000000	0.017578	0.005859	0.000000	0.005859	...	0.012695	0.004883	0.004883	0.002930	0.0097
591	0.017578	0.029297	0.015625	0.013672	0.003906	0.015625	0.025391	0.000000	0.000000	0.009766	...	0.073242	0.000000	0.028320	0.000000	0.0019
592	0.013672	0.009766	0.060547	0.025391	0.035156	0.025391	0.039062	0.000000	0.003906	0.023438	...	0.003906	0.000000	0.000977	0.000000	0.01171
593	0.000000	0.117190	0.000000	0.019531	0.000000	0.136720	0.001953	0.005859	0.000000	0.007812	...	0.107420	0.012695	0.016602	0.000977	0.0048

594 rows × 192 columns

missing value?

Code Snippet:

```
print(train_data.isnull().any().any())
print(test_data.isnull().any().any())
```

Output:

```
False
False
```

Data aggregation

Code Snippet:

```
train_data.describe()
```

Output:

	margin1	margin2	margin3	margin4	margin5	margin6	margin7	margin8	margin9	margin10	...	texture55	texture56	
count	990.000000	990.000000	990.000000	990.000000	990.000000	990.000000	990.000000	990.000000	990.000000	990.000000	...	990.000000	990.000000	99
mean	0.017412	0.028539	0.031988	0.023280	0.014264	0.038579	0.019202	0.001083	0.007167	0.018639	...	0.036501	0.005024	
std	0.019739	0.038855	0.025847	0.028411	0.018390	0.052030	0.017511	0.002743	0.008933	0.016071	...	0.063403	0.019321	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.000000	
25%	0.001953	0.001953	0.013672	0.005859	0.001953	0.000000	0.005859	0.000000	0.001953	0.005859	...	0.000000	0.000000	
50%	0.009766	0.011719	0.025391	0.013672	0.007812	0.015625	0.015625	0.000000	0.005859	0.015625	...	0.004883	0.000000	
75%	0.025391	0.041016	0.044922	0.029297	0.017578	0.056153	0.029297	0.000000	0.007812	0.027344	...	0.043701	0.000000	
max	0.087891	0.205080	0.156250	0.169920	0.111330	0.310550	0.091797	0.031250	0.076172	0.097656	...	0.429690	0.202150	

8 rows × 192 columns

Code Snippet:

```
test_data.describe()
```

Output:

	margin1	margin2	margin3	margin4	margin5	margin6	margin7	margin8	margin9	margin10	...	texture55	texture56	t
count	594.000000	594.000000	594.000000	594.000000	594.000000	594.000000	594.000000	594.000000	594.000000	594.000000	...	594.000000	594.000000	59
mean	0.017562	0.028425	0.031858	0.022556	0.014527	0.037497	0.019222	0.001085	0.007092	0.018798	...	0.035291	0.005923	
std	0.019585	0.038351	0.025719	0.028797	0.018029	0.051372	0.017122	0.002697	0.009515	0.016229	...	0.064482	0.026934	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.000000	
25%	0.001953	0.001953	0.013672	0.005859	0.001953	0.000000	0.005859	0.000000	0.001953	0.005859	...	0.000000	0.000000	
50%	0.009766	0.010743	0.023438	0.013672	0.007812	0.013672	0.015625	0.000000	0.005859	0.015625	...	0.003906	0.000000	
75%	0.028809	0.041016	0.042969	0.027344	0.019531	0.056641	0.029297	0.000000	0.007812	0.027344	...	0.038086	0.000000	
max	0.085938	0.189450	0.167970	0.164060	0.093750	0.271480	0.087891	0.021484	0.083984	0.083984	...	0.353520	0.441410	

8 rows × 192 columns

Logistic Regression model with Grid Search CV

Code Snippet:

```
import warnings
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression

model = LogisticRegression()
parameters = [{'C': list(np.arange(1000, 2000, 200)),
               'fit_intercept': [True, False],
               'tol' : [1e-5,1e-4],
               'solver' : ['newton-cg','lbfgs']}]

# Filter out the warning
with warnings.catch_warnings():
    warnings.filterwarnings("ignore", category=UserWarning)
    grid = GridSearchCV(model, parameters)
    grid.fit(X_train, y_train)
```

```
print(grid)
print()
print(grid.get_params())
```

Output:

```
GridSearchCV(estimator=LogisticRegression(),
             param_grid=[{'C': [1000, 1200, 1400, 1600, 1800],
                          'fit_intercept': [True, False],
                          'solver': ['newton-cg', 'lbfgs'],
                          'tol': [1e-05, 0.0001]}])

{'cv': None, 'error_score': nan, 'estimator__C': 1.0, 'estimator__class_weight': None, 'estimator__dual': False, 'estimator__fi
t_intercept': True, 'estimator__intercept_scaling': 1, 'estimator__l1_ratio': None, 'estimator__max_iter': 100, 'estimator__mul
ti_class': 'auto', 'estimator__n_jobs': None, 'estimator__penalty': 'l2', 'estimator__random_state': None, 'estimator__solver':
'lbfgs', 'estimator__tol': 0.0001, 'estimator__verbose': 0, 'estimator__warm_start': False, 'estimator': LogisticRegression(),
'n_jobs': None, 'param_grid': [{'C': [1000, 1200, 1400, 1600, 1800], 'fit_intercept': [True, False], 'tol': [1e-05, 0.0001], 's
olver': ['newton-cg', 'lbfgs']}], 'pre_dispatch': '2*n_jobs', 'refit': True, 'return_train_score': False, 'scoring': None, 'ver
bose': 0}
```

Code Snippet:

```
### Performance evaluation & parameter tuning

print(grid.best_score_)
print(grid.best_estimator_.solver )
print(grid.best_estimator_.C )
print(grid.best_estimator_.fit_intercept )
print(grid.best_estimator_.tol )

print('Accuracy (training set): {}'.format(grid.score(X_train, y_train)))
print('Accuracy (testing set): {}'.format(grid.score(X_test, y_test)))
```

Output:

```
0.9326500997641938
newton-cg
1800
False
1e-05
Accuracy (training set): 1.0
Accuracy (testing set): 0.9233870967741935
```

Practical 4 Note:

- features/attributes
- actual output/y-label/class/label
- instance/records/rows

Practical 5 - Supervised Learning

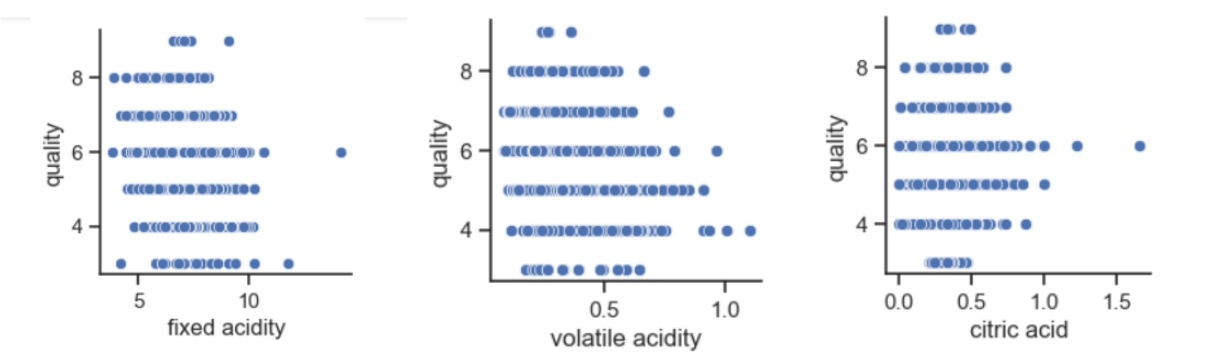
Read the Wine Quality White dataset (`winequality.csv`) from your data folder.

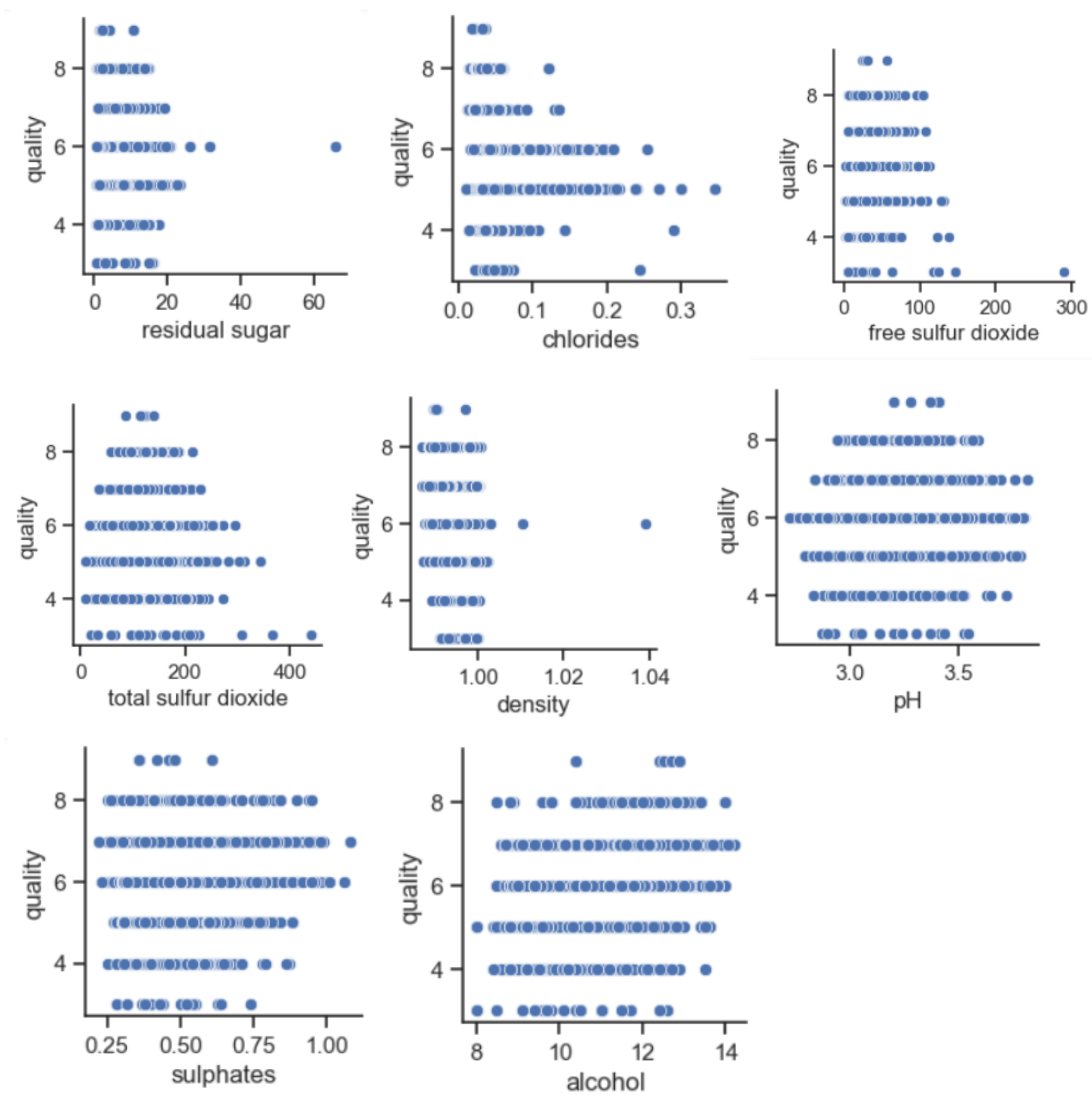
Perform the following tasks:

- select features to be the first 11 columns while the last column to be the target
- plot the pair plot for all features
- separate the data to 80:20 training and testing datasets
- create an instance of Neighbours Classifier and fit the data
- Measure the accuracy and Root Mean Square Error (RMSE) for the fitted model
- Perform grid search to find the best parameters for KNN, for `weights: ['uniform', 'distance']`, `n_neighbors: [40, 60, 80, 100, 120, 140]`

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, accuracy_score
from sklearn.model_selection import GridSearchCV

import pandas as pd

wineQuality = pd.read_csv('winequality.csv')

# select features to be the first 11 columns while the last column to be the target
X_wineQuality = wineQuality.drop('quality', axis=1) #input features
Y_wineQuality = wineQuality['quality'] #class label/actual output
```

```
# plot the pair plot for all features
import matplotlib.pyplot as plt

# Define the features and target
features = ['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar', 'chlorides',
            'free sulfur dioxide', 'total sulfur dioxide', 'density', 'pH', 'sulphates', 'alcohol']

# Create a pair plot for each feature against quality
for feature in features:
    sns.set(style="ticks")
    sns.pairplot(wineQuality, x_vars=[feature], y_vars=['quality'], diag_kind='kde')
    plt.show()
```






```python
# separate the data to 80:20 training and testing datasets
Xtrain_wine, Xtest_wine, Ytrain_wine, Ytest_wine = train_test_split(X_wineQuality,
                                                                    Y_wineQuality,
                                                                    train_size = 0.8,
                                                                    random_state=0)
```

```python
# create an instance of Neighbours Classifier and fit the data
# 1. choose model class (KNN)
# 2. instantiate model
wine_knn = KNeighborsClassifier(n_neighbors = 5) #n_neighbors = k-value
# 3. fit model to data (used on training dataset)
wine_knn.fit(Xtrain_wine, Ytrain_wine)
```

```
import warnings
warnings.filterwarnings("ignore", category=FutureWarning)

# Measure the accurary and Root Mean Square Error (RMSE) for the fitted model

# 4. predict on new data (testing)
y_wine_knn = wine_knn.predict(Xtest_wine)

# 5. evaluate performance
# Calculate accuracy
accuracy = accuracy_score(Ytest_wine, y_wine_knn)
print("Accuracy:", accuracy)
# Calculate Root Mean Square Error (RMSE)
rmse = mean_squared_error(Ytest_wine, y_wine_knn, squared=False)
print("RMSE:", rmse)
```

```
Accuracy: 0.45510204081632655
RMSE: 0.9752027523357721
```

```python
# Perform grid search to find the best parameters for KNN,
# for weights: ['uniform', 'distance'], n_neighbors: [40, 60, 80, 100, 120, 140]

wine_knn = KNeighborsClassifier(n_neighbors = 5)

# Define the parameter grid
k_range = list(range(1, 140))
parameters = [
    {'n_neighbors': [40, 60, 80, 100, 120, 140],
     'weights': ['uniform', 'distance']}
]

# Filter out the warning
with warnings.catch_warnings():
    warnings.filterwarnings("ignore", category=UserWarning)

    # Create GridSearchCV instance
    grid = GridSearchCV(wine_knn, parameters, cv=10,
                        scoring='accuracy', return_train_score=False,verbose=1)

    # Fit the grid search to your data
    grid_search = grid.fit(Xtrain_wine, Ytrain_wine)

# Print the best parameters and the corresponding accuracy score
print(grid)
print("Best parameters:", grid_search.best_params_)
print("Best mean cross-validated accuracy:", grid_search.best_score_)
```

```
Fitting 10 folds for each of 12 candidates, totalling 120 fits
GridSearchCV(cv=10, estimator=KNeighborsClassifier(),
             param_grid=[{'n_neighbors': [40, 60, 80, 100, 120, 140],
                          'weights': ['uniform', 'distance']}],
             scoring='accuracy', verbose=1)
Best parameters: {'n_neighbors': 60, 'weights': 'distance'}
Best mean cross-validated accuracy: 0.6245465577535362
```

Exercise 5.1

Data Preprocessing

Code:

```
# Change non-numerical data to numerical
from sklearn.preprocessing import LabelEncoder

# Define the columns with non-numerical data
categorical_columns = [' phnum', ' intplan', ' voice', ' label']

# Initialize the LabelEncoder
label_encoder = LabelEncoder()

# Apply label encoding to the categorical columns
for col in categorical_columns:
    train_data[col] = label_encoder.fit_transform(train_data[col])
    test_data[col] = label_encoder.fit_transform(test_data[col])
```

```
# split data into input features and label

X_train_data = train_data.drop(' label', axis=1)
y_train_data = train_data[' label']
X_test_data = test_data.drop(' label', axis=1)
y_test_data = test_data[' label']
```

Fit and Test Models

KNN

Code:

```
# 1. choose model class
from sklearn.neighbors import KNeighborsClassifier
```

```
import warnings
warnings.filterwarnings("ignore", category=FutureWarning)

# find the best n_neighbors value
from sklearn.model_selection import GridSearchCV
knn = KNeighborsClassifier()
param_grid = {'n_neighbors': range(1, 20)}

# Instantiate GridSearchCV
grid_search = GridSearchCV(knn, param_grid, cv=5, scoring='accuracy')

# Fit the model
grid_search.fit(X_train_data, y_train_data)

# Get the best k-value
best_k = grid_search.best_params_['n_neighbors']
print("Best k-value:", best_k)
```

```
Best k-value: 7
```

```
# 2. instantiate model
customer_knn = KNeighborsClassifier(n_neighbors = 7)
# 3. fit model to data
customer_knn.fit(X_train_data, y_train_data)
# 4. predict on new data
y_test_data_predicted_knn = customer_knn.predict(X_test_data)
```

Logistic Regression
Code:

```
# 1. choose model class
from sklearn.linear_model import LogisticRegression
```

```
warnings.filterwarnings("ignore", category=UserWarning)

# find the best C value
param_grid = {
    'penalty': ['l1', 'l2'],
    'C': [0.001, 0.01, 0.1, 1, 10, 100]
}

logreg = LogisticRegression(solver='liblinear')
grid_search = GridSearchCV(logreg, param_grid, cv=5, scoring='accuracy')

grid = grid_search.fit(X_train_data, y_train_data)
grid.best_estimator_
```

```
LogisticRegression(C=1, solver='liblinear')
```

```
# 2. instantiate model
customer_logistic = grid.best_estimator_
# 3. fit model to data
customer_logistic.fit(X_train_data, y_train_data)
# 4. predict on new data
y_test_data_predicted_logreg = customer_logistic.predict(X_test_data)
```

SVM
Code:

```
# 1. choose model class
from sklearn.svm import SVC
```

```
# 1. Choose model class and use GridSearchCV to find best parameters
param_grid = {
    'C': [0.000000001, 0.000001, 0.0001, 0.001, 0.01, 0.1, 1, 10, 100]
}

svm = SVC()
grid_search = GridSearchCV(svm, param_grid, cv=5, scoring='accuracy', n_jobs=-1)

grid = grid_search.fit(X_train_data, y_train_data)
grid.best_estimator_
```

```
SVC(C=1e-09)
```

```
# 2. Instantiate model
customer_svm = grid.best_estimator_

# 3. Fit model to data
customer_svm.fit(X_train_data, y_train_data)

# 4. Predict on new data
y_test_data_predicted_svm = customer_svm.predict(X_test_data)
```

Gaussian Naive Bayes
Code:

```
# 1. choose model class
from sklearn.naive_bayes import GaussianNB
# 2. instantiate model
gnb = GaussianNB()
# 3. fit model to data
gnb.fit(X_train_data, y_train_data)
# 4. predict on new data
y_test_data_predicted_gnb = gnb.predict(X_test_data)
```

Model Comparison/ Performance Evaluation

Code:

```
from sklearn.metrics import mean_squared_error, mean_squared_log_error, r2_score
```

```
#KNN

# Mean square error (MSE)
mse_knn = mean_squared_error(y_test_data, y_test_data_predicted_knn)
print("Mean Square Error (MSE):", mse_knn)

# Root mean square error (RMSE)
rmse_knn = mean_squared_error(y_test_data, y_test_data_predicted_knn, squared=False)
print("Root Mean Square Error (RMSE):", rmse_knn)

# R2 (variance) score
r2_knn = r2_score(y_test_data, y_test_data_predicted_knn)
print("R2 (Variance) Score:", r2_knn)

# Signal-to-Noise Ratio (SNR)
mean_squared_predicted_knn = np.mean(y_test_data_predicted_knn ** 2)
mean_squared_error_knn = np.mean((y_test_data - y_test_data_predicted_knn) ** 2)
snr_knn = 10 * np.log10(mean_squared_predicted_knn / mean_squared_error_knn)
print("Signal-to-Noise Ratio (SNR):", snr_knn)
```

```
#Logistic Regression

# Mean square error (MSE)
mse_lr = mean_squared_error(y_test_data, y_test_data_predicted_logreg)
print("Mean Square Error (MSE):", mse_lr)

# Root mean square error (RMSE)
rmse_lr = mean_squared_error(y_test_data, y_test_data_predicted_logreg, squared=False)
print("Root Mean Square Error (RMSE):", rmse_lr)

# R2 (variance) score
r2_lr = r2_score(y_test_data, y_test_data_predicted_logreg)
print("R2 (Variance) Score:", r2_lr)

# Signal-to-Noise Ratio (SNR)
mean_squared_predicted_lr = np.mean(y_test_data_predicted_logreg ** 2)
mean_squared_error_lr = np.mean((y_test_data - y_test_data_predicted_logreg) ** 2)
snr_lr = 10 * np.log10(mean_squared_predicted_lr / mean_squared_error_lr)
print("Signal-to-Noise Ratio (SNR):", snr_lr)
```

```python
#SVM

# Mean square error (MSE)
mse_svm = mean_squared_error(y_test_data, y_test_data_predicted_svm)
print("Mean Square Error (MSE):", mse_svm)

# Root mean square error (RMSE)
rmse_svm = mean_squared_error(y_test_data, y_test_data_predicted_svm, squared=False)
print("Root Mean Square Error (RMSE):", rmse_svm)

# R2 (variance) score
r2_svm = r2_score(y_test_data, y_test_data_predicted_svm)
print("R2 (Variance) Score:", r2_svm)

# Signal-to-Noise Ratio (SNR)
mean_squared_predicted_svm = np.mean(y_test_data_predicted_svm ** 2)
mean_squared_error_svm = np.mean((y_test_data - y_test_data_predicted_svm) ** 2)
snr_svm = 10 * np.log10(mean_squared_predicted_svm / mean_squared_error_svm)
print("Signal-to-Noise Ratio (SNR):", snr_svm)
```

```python
#Gaussian Naive Bayes

# Mean square error (MSE)
mse_gnb = mean_squared_error(y_test_data, y_test_data_predicted_gnb)
print("Mean Square Error (MSE):", mse_gnb)

# Root mean square error (RMSE)
rmse_gnb = mean_squared_error(y_test_data, y_test_data_predicted_gnb, squared=False)
print("Root Mean Square Error (RMSE):", rmse_gnb)

# R2 (variance) score
r2_gnb = r2_score(y_test_data, y_test_data_predicted_gnb)
print("R2 (Variance) Score:", r2_gnb)

# Signal-to-Noise Ratio (SNR)
mean_squared_predicted_gnb = np.mean(y_test_data_predicted_gnb ** 2)
mean_squared_error_gnb = np.mean((y_test_data - y_test_data_predicted_gnb) ** 2)
snr_gnb = 10 * np.log10(mean_squared_predicted_gnb / mean_squared_error_gnb)
print("Signal-to-Noise Ratio (SNR):", snr_gnb)
```

Output:

KNN	Mean Square Error (MSE): 0.12117576484703059 Root Mean Square Error (RMSE): 0.3481030951414115 R2 (Variance) Score: -0.041771854271853925 Signal-to-Noise Ratio (SNR): -7.738724524043686
Logistic Regression	Mean Square Error (MSE): 0.12837432513497302 Root Mean Square Error (RMSE): 0.3582936297716902 R2 (Variance) Score: -0.10365929115929085 Signal-to-Noise Ratio (SNR): -4.6118205361821465
SVM	Mean Square Error (MSE): 0.13437312537492502 Root Mean Square Error (RMSE): 0.36656940048908204 R2 (Variance) Score: -0.15523215523215494 Signal-to-Noise Ratio (SNR): -inf
Gaussian Naive Bayes	Mean Square Error (MSE): 0.1271745650869826 Root Mean Square Error (RMSE): 0.35661543024241477 R2 (Variance) Score: -0.09334471834471803 Signal-to-Noise Ratio (SNR): 0.12120632675853438

Conclusion

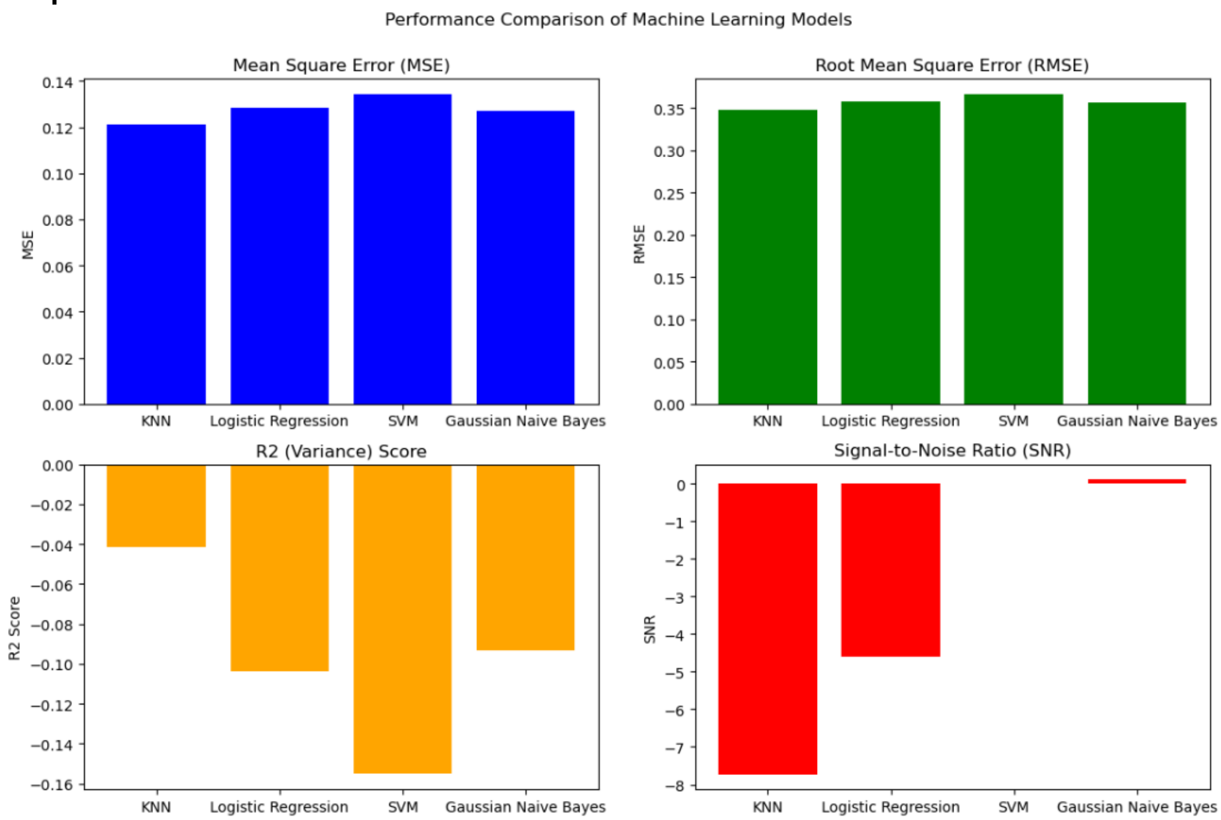
Present your findings or summary from your work

#provide visualisations

Code:

```
import matplotlib.pyplot as plt
import numpy as np

# Model names
models = ['KNN', 'Logistic Regression', 'SVM', 'Gaussian Naive Bayes']
# Performance metrics for each model
mse = [mse_knn, mse_lr, mse_svm, mse_gnb]
rmse = [rmse_knn, rmse_lr, rmse_svm, rmse_gnb]
r2 = [r2_knn, r2_lr, r2_svm, r2_gnb]
snr = [snr_knn, snr_lr, float(snr_svm), snr_gnb]  # Note: Handling negative infinity
# Create subplots
fig, axs = plt.subplots(2, 2, figsize=(12, 8))
fig.suptitle('Performance Comparison of Machine Learning Models')
# Bar plot for MSE
axs[0, 0].bar(models, mse, color='blue')
axs[0, 0].set_title('Mean Square Error (MSE)')
axs[0, 0].set_ylabel('MSE')
# Bar plot for RMSE
axs[0, 1].bar(models, rmse, color='green')
axs[0, 1].set_title('Root Mean Square Error (RMSE)')
axs[0, 1].set_ylabel('RMSE')
# Bar plot for R2 Score
axs[1, 0].bar(models, r2, color='orange')
axs[1, 0].set_title('R2 (Variance) Score')
axs[1, 0].set_ylabel('R2 Score')
# Bar plot for SNR
axs[1, 1].bar(models, snr, color='red')
axs[1, 1].set_title('Signal-to-Noise Ratio (SNR)')
axs[1, 1].set_ylabel('SNR')
# Adjust layout
plt.tight_layout()
plt.subplots_adjust(top=0.9)
# Show the plot
plt.show()
```

Output:



Discussion:

- KNN has the lowest MSE and RMSE among all models, indicating better prediction accuracy and smaller prediction errors. However, its R2 score is slightly negative, suggesting a poor fit to the data. The negative SNR indicates the noise is much higher than the signal.
- Logistic Regression shows slightly higher MSE and RMSE compared to KNN, with a negative R2 score and negative SNR. It seems to have a similar performance trend to KNN but with slightly worse results.
- SVM has the highest MSE, RMSE, and negative R2 score, indicating relatively poor performance. The undefined SNR suggests that the model's predictions and actual values do not match.
- Gaussian Naive Bayes has similar MSE and RMSE to Logistic Regression, but its R2 score is better. The positive SNR suggests that the signal is more pronounced compared to the noise.

From this evaluation, it appears that **Gaussian Naive Bayes has the relatively best performance** among the models considered, with the lowest negative R2 score and a positive SNR. However, it's essential to interpret these results with caution and consider the specific context and requirements of your problem before making a final decision about model selection.

Supervised Learning	**Unsupervised Learning**
Data is **labelled** with predefined classes.	Class labels of the data are unknown.
Develop predictive models based on both input and output data. • **Classification** when the output is categorical / nominal (discrete labels such as classifying spam email vs non-spam email) • **Regression** when the output is real values (continuous data such as predicting house prices).	Group and interpret data based only on input data. **Clustering** • Finding association (in features) • Dimension reduction • Sometimes called knowledge discovery **Dimensionality reduction** **Association rule**
Goal: Learn a mapping from inputs x to outputs y, given a labelled set of input-output pairs.	Goal: Given a set of data, the task is to establish the existence of clusters in the data.

Classification	**Regression**
no rain, rain	Amount of rain 1,2mm, 1.3mm…
Logistic regression, SVM (support vector machine), KNN, NN, Gaussian Naive Bayes	Linear regression, SVM, NN

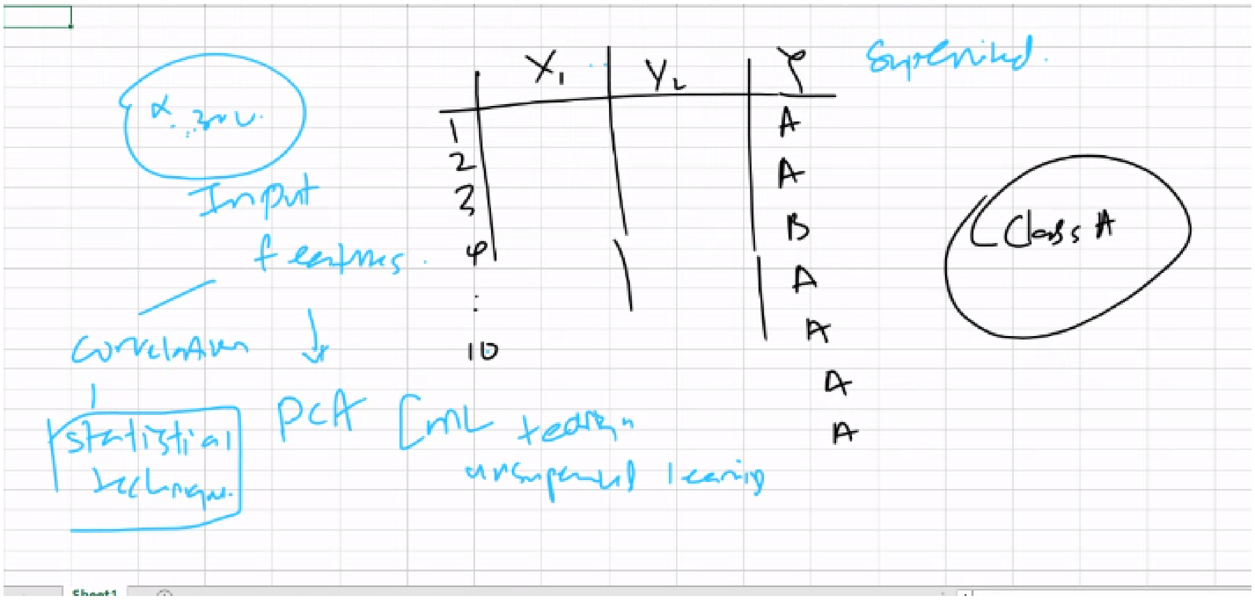
Performance metrics
• Accuracy Rate
• F-measure
• Recall
	Performance metrics
	• Mean square error (MSE)
	  ○ lower better
	• Normalised mean square error (NMSE)
	  ○ 0 - 1, lower better
	• Root mean square error (RMSE)
	  ○ ower better
	• R^2 (variance) score
	  ○ 0 - 1, higher better
	• Signal-to-noise ratio

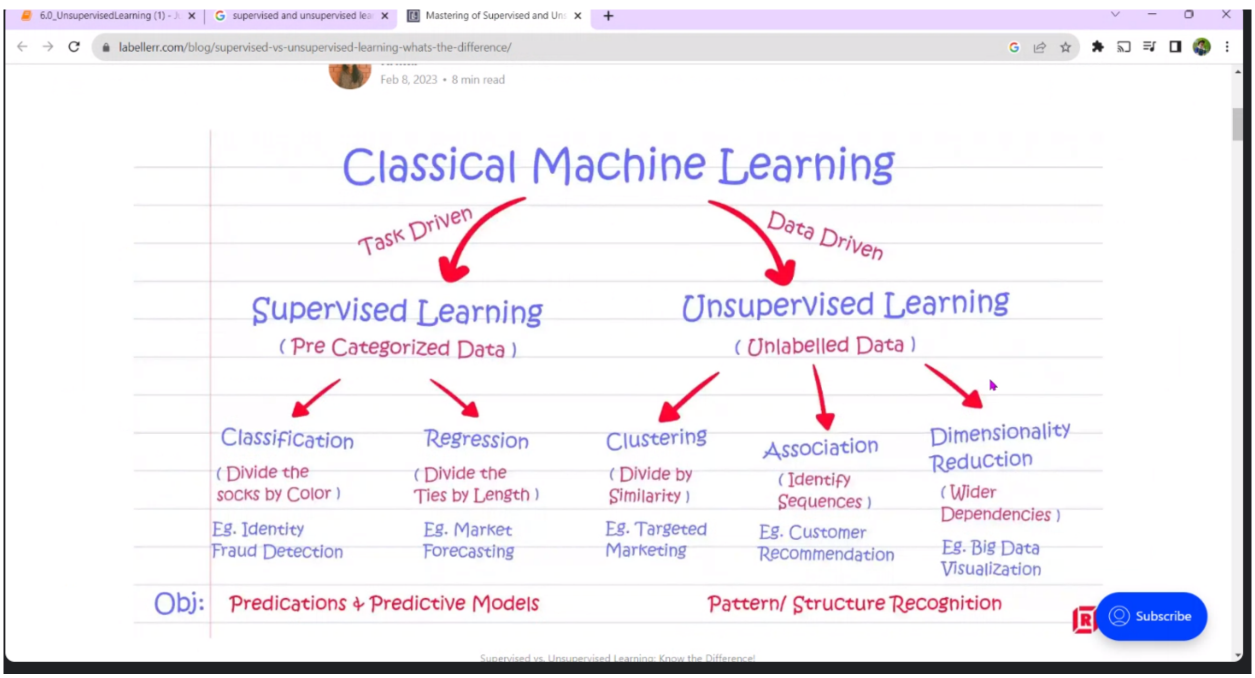
Practical 6 - Unsupervised Learning

<table>
<tr><td>Clustering</td><td>Group and interpret data based only on input data.
Techniques:
K-means (based on similarity of colours, size)
Hierarchical clustering</td></tr>
<tr><td>Association rules</td><td>Example: determine how often milk is bought together with bread</td></tr>
<tr><td>Dimensionality reduction</td><td>Technique: Principal Component Analysis
Example:
a dataset with shape (50, 101) for supervised learning
100 columns are input features and 1 column is class label.
We want to extract just 10 principal components as the input features.
Parameters:
Minimum (E.g. 2)
2-max
Maximum (E.g. 10)
Constraint: min (n,m) where n is features and m is records.
In this case, it’s min(100,50) = 50.
So maximum can only be max 50.
Steps:
Identify the principal component
Explained variance</td></tr>
</table>

Accuracy on testing set 100% is weird, might be caused by 3 reasons
1. wrong propotion of testing and training
2. inbalanced dataset
3. 300 input features to 50 records = cursed of dimensionality

Semi-supervised learning if (50 labelled, 50 unlabelled)


X1
Y2
Y
Input
features.
correlation
statistical
technique.
PCA
A
A
B
A
A
A
A
Class A


6.0_UnsupervisedLearning (1)
supervised and unsupervised lea
Mastering of Supervised and Un
labellerr.com/blog/supervised-vs-unsupervised-learning-whats-the-difference/
Feb 8, 2023 • 8 min read
Classical Machine Learning
Task Driven
Data Driven
Supervised Learning
(Pre Categorized Data)
Unsupervised Learning
(Unlabelled Data)
Classification
(Divide the socks by Color)
Eg. Identity Fraud Detection
Regression
(Divide the Ties by Length)
Eg. Market Forecasting
Clustering
(Divide by Similarity)
Eg. Targeted Marketing
Association
(Identify Sequences)
Eg. Customer Recommendation
Dimensionality Reduction
(Wider Dependencies)
Eg. Big Data Visualization
Obj: Predications + Predictive Models
Pattern/ Structure Recognition
Subscribe
Supervised vs. Unsupervised Learning: Know the Difference!