MACHINE LEARNING INTERNSHIP

AT CODEGNAN IT SOLUTIONS PRIVATE LIMITED

TEAM DETAILS:

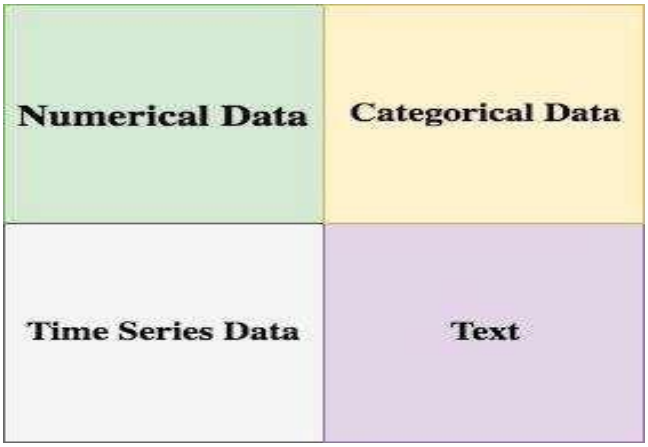
NAME	ROLL NUMBER
VENKAT CHOUDHARY ALA	20VV1A0503
KOPPISETTY VYSHNAVI	20VV1A0523
MAREEDU VENKATA MIDHUN	20VV1A0530

Week – 1 (15-05-2023 to 19-05-2023)

Day & Date	Learning Outcomes
Day – 1	Overview on AI, Machine Learning and different programming languages.
Day – 2	Introduction to Python, Python functions
Day – 3	Built-in Python Packages/Modules
Day - 4	Working on NumPy
Day – 5	Working on Pandas

Day – 1: Overview on Data Science, Machine Learning, AI and different programming languages.

Data: Data might be of any form. It is broadly categorized as Qualitative (Categorical) and Quantitative (Numerical).

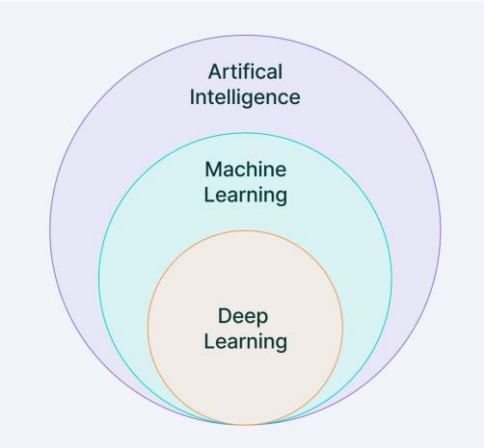


A brief history of AI:

Artificial Intelligence (AI) is a field of computer science and technology that focuses on creating intelligent machines capable of performing tasks that typically require human intelligence. The history of AI can be traced back to ancient times, with the concept of artificial beings and intelligent machines appearing in myths and folklore. However, the modern development of AI began in the mid-20th century.

- 1940s-1950s: The first mathematical models of neural networks are developed, and Alan Turing proposes the "Turing Test" to assess machine intelligence.
- 1956: The Dartmouth Conference marks the birth of AI as a formal field of study.
- 1950s-1960s: Early AI approaches focus on symbolic systems and rule-based problem-solving.
- 1970s-1980s: The "AI winter" period sees a decline in AI research due to limited progress and reduced funding.
- 1980s-1990s: Expert systems and machine learning gain prominence.
- 2000s-2010s: Neural networks and deep learning experience a resurgence, driven by advancements in computing power and big data availability.
- Recent years: AI applications expand across domains, with breakthroughs in reinforcement learning, generative models, and explainable AI.

It's important to note that this is a condensed overview of AI's history, and many significant milestones and contributions have been omitted for brevity. AI continues to evolve rapidly, and its impact on society and technology is likely to grow in the coming years.



Data Science Vs Artificial Intelligence:

Data Science and AI use data, the two fields differ in their focus. Data Science primarily focuses on understanding the data and extracting insights, while AI focuses on building intelligent systems that can make decisions or perform tasks autonomously.

- Data Science is a concept used to tackle big data and includes data cleansing, preparation, and analysis. A data scientist gathers data from multiple sources and applies machine learning, predictive analytics, and sentiment analysis to extract critical information from the collected data sets.
- Artificial Intelligence aims at enabling machines to execute reasoning by replicating human intelligence. Since the main objective of AI processes is to teach machines from experience, feeding the right information and self-correction is crucial.
- AI experts rely on deep learning and natural language processing to help machines identify patterns and inferences.

Introduction to Machine Learning:

“Machine Learning is the field of study that gives computers the ability to learn without explicitly programmed.”

Has been in the picture from the past 50 years. Initially coined by Arthur Samuel in the year 1959 as Field of Study that gives computers the ability to learn without being explicitly programmed. It was not that significant as of today 22 years ago. Best definition by Tom Mitchell -A Computer Program is said to learn from experience "E" with respect to some class of tasks "T" and performance measure "P", If its performance at tasks in "T“ as measured by "P", improves with experience "E".

- 22 years ago IBMs Deep Blue beat World Chess Champion Gary Kasparov, which marked the importance and gave people a serious attention.
- Then in the 21st century everything has seen exponential growth in the development and applications.
- 2006 marked the word Deep Learning by publications made by Geoffrey Hinton from the concept of Deep Belief Nets.
- 2011 IBM Watson beat Human champions in Jeopardy.
- Google started Brain Team and X Lab.
- 2014 FaceBook Face Tagging (DeepFace)
- 2016 Google’s Alpha Go beat Professional Players in Go game.
- 2017 - 2019 Self Driving Cars and Tensorflow influence.

6 Jars of Machine Learning:

The "6 Jars of Machine Learning" is a framework by Jeremy Howard that categorizes the challenges faced in building successful machine learning models.

1. Data
2. Tasks
3. Model
4. Loss
5. Learning
6. Evaluation

Each jar addresses a specific aspect of ML, from data quality to model optimization and evaluation, to achieve optimal results.

Machine Learning Life Cycle:

The machine learning life cycle typically involves the following four steps:

1. Data Preparation: This step involves collecting, cleaning, and preprocessing the data to make it suitable for machine learning algorithms. It includes tasks such as handling missing values, removing outliers, and transforming data into a usable format.

2. Model Training: In this step, the prepared data is used to train machine learning models. The models learn patterns and relationships in the data through various algorithms and techniques. The training process involves selecting the appropriate algorithm, tuning hyperparameters, and optimizing the model's performance.
3. Model Evaluation: Once the models are trained, they need to be evaluated to assess their performance and generalization capabilities. Evaluation metrics and techniques are used to measure how well the model performs on unseen data. This step helps in identifying any issues or areas for improvement.
4. Model Deployment and Maintenance: After a model is evaluated and deemed satisfactory, it can be deployed into a production environment to make predictions or decisions. Monitoring the model's performance, updating it with new data, and retraining it periodically are crucial for maintaining its accuracy and adapting to changing conditions.

The machine learning life cycle is an iterative process, and these steps are repeated as new data becomes available, new requirements arise, or improvements are needed to enhance the model's performance.

A brief overview on Machine Learning:

Machine Learning (ML) is a subset of artificial intelligence (AI) that focuses on developing algorithms and models that enable computers to learn and make predictions or decisions without explicit programming. It involves the study of statistical models and algorithms that allow systems to learn from and make predictions or take actions based on data. Here's an overview of Machine Learning:

1. Supervised Learning: Supervised learning is the most common type of ML. It involves training a model using labeled data, where the desired output or target variable is known. The model learns patterns and relationships in the data to make predictions or classifications on new, unseen data.
    - Examples of supervised learning algorithms include linear regression, logistic regression, decision trees, support vector machines, and neural networks.
2. Unsupervised Learning: Unsupervised learning involves training a model on unlabelled data, where the target variable is not provided. The goal is to discover patterns, structures, or clusters in the data without prior knowledge of the expected output.
    - Clustering algorithms, such as K-means clustering, hierarchical clustering, and dimensionality reduction techniques like Principal Component Analysis (PCA), are commonly used in unsupervised learning.
3. Reinforcement Learning: Reinforcement learning (RL) focuses on training an agent to interact with an environment and learn optimal actions through a reward-based system. The agent explores the environment, receives feedback (rewards or penalties), and adjusts its actions to maximize the cumulative reward over time.
    - RL has applications in robotics, gaming, autonomous systems, and optimization problems.

4. Feature Extraction and Selection: Feature extraction involves transforming raw data into a format that ML algorithms can understand and process effectively. Feature selection refers to identifying the most relevant and informative features for training models, eliminating redundant or irrelevant features.
5. Model Evaluation and Validation: To assess the performance of ML models, evaluation and validation techniques are employed.
    - Metrics such as accuracy, precision, recall, F1-score, and area under the ROC curve (AUC-ROC) are used to measure model performance. Techniques like cross-validation and holdout validation are used to estimate model generalization and prevent overfitting.
6. Deep Learning: Deep Learning is a subfield of ML that focuses on training deep neural networks with multiple hidden layers. It has revolutionized areas such as computer vision, natural language processing, and speech recognition.
    - Deep Learning architectures, like Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs), excel at learning complex patterns and structures in large datasets.
7. Model Deployment and Iteration: After training and evaluating ML models, they are deployed into real-world applications, where they make predictions or decisions. Models often require ongoing monitoring, maintenance, and periodic retraining to adapt to changing data distributions or evolving requirements.

Machine Learning has widespread applications, including image and speech recognition, natural language processing, recommendation systems, fraud detection, autonomous vehicles, and personalized medicine. It continues to advance rapidly, fuelled by the availability of big data, improved algorithms, and increased computational power.

Day – 2: Introduction to Python, Python functions.

Python is a popular high-level programming language known for its simplicity and readability. It was created by Guido van Rossum and first released in 1991. Python emphasizes clean and concise code, making it an excellent choice for beginners and experienced programmers alike. Some key aspects of Python include readability, versatility, easy-to-learn, cross-platform compatibility, extensibility, large community and ecosystem, etc. Python's versatility and ease of use have made it popular for a wide range of applications, from web development and scientific computing to artificial intelligence and data analysis. Its simplicity and powerful libraries make it an excellent choice for beginners and professionals alike.

Different Python IDE:

Different Python IDE includes: IDLE, SPYDER, PyCharm, eric, Atom, Jupyter, Anaconda, PyDev, Thonny etc.

Points to be followed in Python Programming:

3 steps in problem solving:

-->Understand the input type (numeric, collection, Boolean type)

-->Know your output format (pattern, number and so on...)

-->Map a logic to make input to output

Zen of Python:

The Zen of Python captures the core philosophy and values of the Python language, promoting code that is readable, concise, and maintainable. It serves as a guiding framework for Python developers to write elegant and effective code.

```
import this
```

```
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

Python loops and functions:

```
d=0
for i in range(1,15) :
    #print(i, end=' ')
    #print(i**2)
    if i%2 == 0:
        d+=1
        print(i, i**3)
print(f"even number count is: {d}")
```

```
2 8
4 64
6 216
8 512
10 1000
12 1728
14 2744
even number count is: 7
```

A sample function to calculate power of a number:

```
def sample(a, b):
    """ Sample function description """
    c=a**b
    print(c)

sample(2, 3)
```

```
8
```

Demonstrating Variable length arguments:

```
def check(*a) :
    """ Demonstration for variable length arguments"""
    print(a)
    print(type(a))
    print(len(a))

check()
check(1, 2,3,4,5)
check([1, 2,3,4,5])
check('code', 'gnan')
```

```
()
<class 'tuple'>
0
(1, 2, 3, 4, 5)
<class 'tuple'>
5
([1, 2, 3, 4, 5],)
<class 'tuple'>
1
('code', 'gnan')
<class 'tuple'>
2
```

Usage of *args:

```
def code(*a) :
    """ usage of *args"""
    print(a)
    s=0
    for i in a:
        if type(i) in (int, float) :
            s += i
    print (s)

code(1, 2,'code',3,[4,5],6,7,8+9j)
d=[10, 12, 13,15]
code (*d)
```

```
(1, 2, 'code', 3, [4, 5], 6, 7, (8+9j))
19
(10, 12, 13, 15)
50
```

```
def samp(**a) :
    print(a)
    print(type(a))

samp()
samp(name="codeGnan", internship= "MachineLearning")
samp(idnum=[501, 502, 503], name = [ " Manasa", " Gayatri"," Venkat"])
```

```
{}
<class 'dict'>
{'name': 'codeGnan', 'internship': 'MachineLearning'}
<class 'dict'>
{'idnum': [501, 502, 503], 'name': [' Manasa', ' Gayatri', ' Venkat']}
<class 'dict'>
```

Usage of dundders:

```
def name() :
    """ Usage of dundders"""
    if __name__ == "__main__":
        print("this program runs as a script")
    else:
        print("this program runs as a module")

name()
print(__name__)
```

```
this program runs as a script
__main__
```

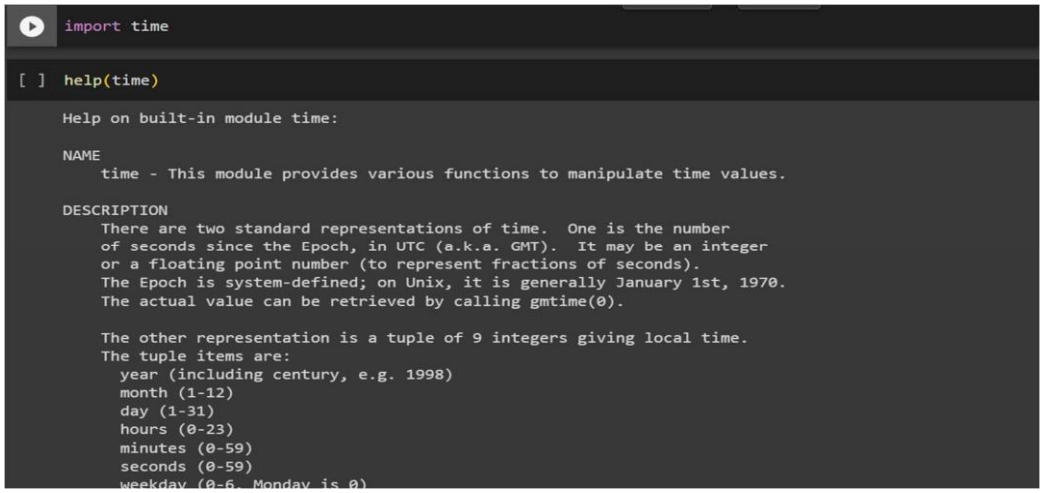
Day – 3: Built-in Packages and models.

Python comes with a comprehensive standard library that includes built-in packages and modules, providing a wide range of functionality for various tasks. Some notable built-in packages and modules include:

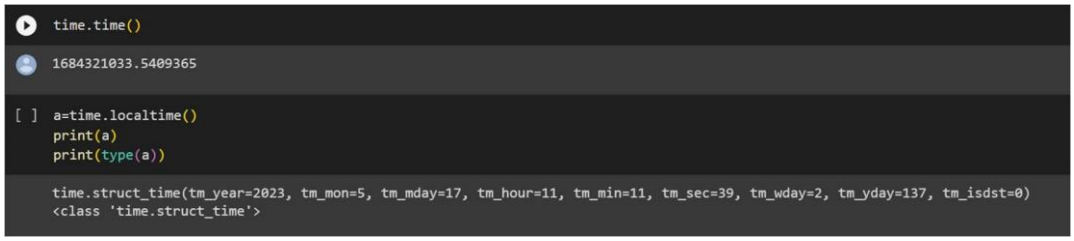
- math: Offers mathematical functions and constants for numeric operations.
- random: Provides functions for generating random numbers and making random selections.
- datetime: Allows manipulation of dates, times, and time intervals.
- os: Facilitates interaction with the operating system, including file operations and directory management.
- re: Supports regular expressions for pattern matching and text manipulation.

These built-in packages and modules provide essential tools for common programming tasks, reducing the need for external dependencies. They contribute to the versatility, convenience, and power of Python, making it a robust language for various applications.

Time module:



Accessing local time:



Calculating difference between times:

```
start= time.time()
for i in range(15) :
    print(i**2)
end= time.time()
print(start, end)
print(end-start)
```

```
0
1
4
9
16
25
36
49
64
81
100
121
144
169
196
1684321518.1716094 1684321518.1745577
0.002948284149169922
```

Printing date:

```
day = a.tm_mday
month= a.tm_mon
year= a.tm_year
print(f'date is {day} - {month} - {year}')
```

```
date is 17 - 5 - 2023
```

Random Module:

```
[ ] import random

    #help(random)
    x=random.randint(2,10)
    x
```

```
7
```

```
[ ] x=3**(1/2)
    x
```

```
1.7320508075688772
```

Python Libraries for Data Analysis:

Popular Python toolboxes/ libraries:

1. NumPy
2. Pandas
3. Scikit-Learn

Visualization libraries:

1. Matplotlib
2. Seaborn
3. Plotly

NumPy Library:

NumPy, short for Numerical Python, is one of the most important foundational packages for numerical computing in Python.

- NumPy performs complex Computations on entire arrays without the need for Python for loops.
- Tools for reading/writing array data to disk and working with memory-mapped files.
- pip install NumPy in command prompt is used to install NumPy in our system.

Pandas Library:

Pandas is an open-source Python Library providing high-performance data manipulation and analysis tool using its powerful data structures.

- The name Pandas is derived from the word Panel Data – an Econometrics from Multidimensional data, as it includes observations over multiple time periods.
- It can work with any format of file like .xlsx,.csv, .html, .json formats. Pandas stores data in two types: Series and Data Frame.
- pip install pandas in command prompt is used to install Pandas in our system.

Key Points to remember:

- Dataset: Collection of data
- Model: Representation of what a ML System has learned from training data
- Training: Process of determining the ideal parameters
- Learning: Output of the training process, which a Machine Learning model can use to make predictions
- Training dataset: subset of the data set to train a model
- Validation dataset: Also, a subset of dataset, disjunct from training dataset, that we use to adjust hyperparameters
- Testing dataset: Used to test model which has gone through initial vetting by validation set.
- Hyperparameter: Values which are set before training a model
- Parameter: A variable of a model that ML System trains on its own.
- Target: Output of the input variable also called as Dependent variable or Response variable.
- Feature: Individual Independent variables that act as input in your system. These are also called as independent variables or predictor variables.
- Label: Labels are the final output, such as output classes.
- Fit: Capture patterns from provided data. Heart of Modelling.
- Evaluate: Determine how accurate the model's predictions are.
- Regularization: Method to estimate a preferred complexity of Machine Learning so that the model generalizes, by avoiding underfitting and overfitting.

Day – 4: Working on NumPy:

NumPy arrays are faster in execution and perform element wise operations.

```
[1] import numpy as np

[ ] #Floating point array --> output will be in float

    data = [15,35,45,6,12]
    c = np.array(data, dtype = 'float')
    c

    array([15., 35., 45.,  6., 12.])

[ ] c+c

    array([30., 70., 90., 12., 24.])
```

```
    #ndim() --> returns dimensions of the array
    print(c.ndim)
    print(c.size)   #returns the no. of elements in the array
    print(c.shape)

    1
    5
    (5,)
```

Creation of 2-Dimensional Array:

```
[ ] x = [[1,5,8],[15,5,6],[-2,5,9]]
    len((x))

    3

    d = np.array(x)
    d

    array([[ 1,  5,  8],
           [15,  5,  6],
           [-2,  5,  9]])

axis 0 --> rows

axis 1 --> columns

[ ] type(d)
    print(d.ndim)
    print(d.size)
    print(d.shape)

    2
    9
    (3, 3)
```

Indexing and slicing on NumPy arrays:

```
d[1]
print(d[1:])
```

```
[[15  5  6]
 [-2  5  9]]
```

```
#subindexing

print(d[1][1])
```

```
5
```

```
#Accessing column wise data

d[:, 0]
print(d[1:,1:])
```

```
[[5 6]
 [5 9]]
```

Transformation of arrays (by using reshape() function):

Reshaping arr array to (2,5) and then back to original

```
arr = np.arange(10)
print(arr)
arr.ndim
```

```
[0 1 2 3 4 5 6 7 8 9]
1
```

```
#reshape() --> transorms the given array with possible number of elements in array

e = arr.reshape(2,5)
print(e)
e.ndim
```

```
[[0 1 2 3 4]
 [5 6 7 8 9]]
2
```

```
f = e.reshape(1,-1)
print(f)
f.shape
```

```
[[0 1 2 3 4 5 6 7 8 9]]
(1, 10)
```

Transforming a (4,3) array to (2,6) array

```
a = np.array([[1,8,5],
              [2,3,6],
              [5,1,2],
              [4,6,8]])
print(a)
a.shape
```

```
[[1 8 5]
 [2 3 6]
 [5 1 2]
 [4 6 8]]
(4, 3)
```

```
a = a.reshape(2,6)
print(a)
print(a.shape)
```

```
[[1 8 5 2 3 6]
 [5 1 2 4 6 8]]
(2, 6)
```

Transforming a 3-Dimensional array (by using reshape() function):

```
[ ]  # 3 dimensional

     a = a.reshape(2,3,2)  #2-splits,3-rows,2-columns
     print(a)
     print(f'a.ndim = {a.ndim}')
```

```
[[[1 8]
  [5 2]
  [3 6]]

 [[5 1]
  [2 4]
  [6 8]]]
a.ndim = 3
```

```
# Relating 1D arrays with ones and zeros
#np.zeros(shape) --> creates an array of given shape with all zeros

zero = np.zeros(5)
print(zero)
zer = np.zeros((2,6))
print(zer)
```

```
[0. 0. 0. 0. 0.]
[[0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]]
```

```
[ ]  zer[1][2] = -25
     print(zer)
```

```
[[  0.   0.   0.   0.   0.   0.]
 [  0.   0. -25.   0.   0.   0.]]
```

```
#np.ones(shape) --> creates an array of given shape with all ones
one = np.ones(5)
print(one)
one.ndim
```

```
[1. 1. 1. 1. 1.]
1
```

```
[ ]  on = np.ones(5, dtype='int')
     print(on)
```

```
[1 1 1 1 1]
```

Arithmetic Operations on arrays:

```
[ ]  a = np.array([[1,5,6],
                   [1,5,8],
                   [6,2,4]])
     print(a)
```

```
[[1 5 6]
 [1 5 8]
 [6 2 4]]
```

```
[ ]  b = np.arange(9).reshape(3,3)
     b
```

```
array([[0, 1, 2],
       [3, 4, 5],
       [6, 7, 8]])
```

Addition:

```
print(a+b)
```

```
[[ 1  6  8]
 [ 4  9 13]
 [12  9 12]]
```

By using add() from NumPy.

Addition:

```
print(np.add(a,b))
```

```
[[ 1  6  8]
 [ 4  9 13]
 [12  9 12]]
```

arange() and linspace():

arange() -->

- returns evenly spaced values within a given interval
- output is an array

```
a = np.arange(12)
a
```

```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11])
```

linspace() -->

- returns evenly spaced values over a specified interval
- mostly used for graphical representation

```
a = np.linspace(1,12)   #by default interval is 50
a
b = np.linspace(1,12,5) #returns exact 5 values within given interva;
b
c = np.linspace(1,12,5,endpoint = False)
c
d = np.linspace(1,12,5, endpoint = False, retstep = True)   #retstep = True --> will display repetition interval
d
```

```
(array([1. , 3.2, 5.4, 7.6, 9.8]), 2.2)
```

Indexing with Boolean arrays: Selection process -

```
a = np.array([8,12,5,3,4])
print(a)
```

```
[ 8 12  5  3  4]
```

```
print(a==5)
```

```
[False False  True False False]
```

```
print(a<5)
```

```
[False False False  True  True]
```

Selecting specific values:

```
val = a<10
print(val)
print(a[val])
```

```
[ True False  True  True  True]
[8 5 3 4]
```

Expressing conditional logic in arrays (by using where()):

```
a = np.array([1,6,8,12,25])
print(a)
```

```
[ 1  6  8 12 25]
```

```
b = np.where(a%2==0)
b
```

```
(array([1, 2, 3]),)
```

```
b = np.where(a%2==0,a,a*5)
print(b)
```

```
[  5   6   8  12 125]
```

```
c = np.where(a%5==0,a*3,a)
c
```

```
array([ 1,  6,  8, 12, 75])
```

Matrix multiplication / Dot Product of array:

- matrix multiplication rule--> number of columns in 1st matrix should be equal to number of rows of 2nd matrix

```
a = np.array([[1,2],[2,6]])
print(a)
b = np.array([[5,8],[1,6]])
print(b)
```

```
[[1 2]
 [2 6]]
[[5 8]
 [1 6]]
```

```
np.dot(a,b)
```

```
array([[ 7, 20],
       [16, 52]])
```

Inverse function in NumPy arrays:

```
#2*2 arrays --> linalg() module
inv_array = np.array([[1,5],[6,2]])
inv_array
```

```
array([[1, 5],
       [6, 2]])
```

```
np.linalg.inv(inv_array)
```

```
array([[-0.07142857,  0.17857143],
       [ 0.21428571, -0.03571429]])
```

universal functions -->log(), sqrt(), exp()

```
a=np.arange(1,5)
print(a)
print(np.log(a))
print(np.sqrt(a))
print(np.square(a))
print(np.exp(a))
```

```
[1 2 3 4]
[0.         0.69314718 1.09861229 1.38629436]
[1.         1.41421356 1.73205081 2.        ]
[ 1  4  9 16]
[ 2.71828183  7.3890561  20.08553692 54.59815003]
```

statistical functions -- > mean(), var(), std():

```
b = np.array([1,4,8,6])
print(b)
```

```
[1 4 8 6]
```

```
[ ] print(np.mean(b))
```

```
4.75
```

```
[ ] print(np.median(b))
```

```
5.0
```

```
[ ] print(np.std(b))
```

```
2.5860201081971503
```

```
[ ] print(np.var(b))  #var --> summation (x-xmean)/n-1
```

```
6.6875
```

Maximum and minimum values in an array:

```
a = np.array([[1,5,6],
              [14,4,8],
              [6,-2,24]])
print(a)
```

```
[[ 1  5  6]
 [14  4  8]
 [ 6 -2 24]]
```

```
[ ] print(f'min = {a.min()}')
    print(f'max = {a.max()}')
```

```
min = -2
max = 24
```

We can specify the axis:

```
[ ]  # we can specify the axis
     print(f'min in each row = {a.min(axis = 1)}')
     print(f'max in each col = {a.max(axis = 0)}')

     min in each row = [ 1  4 -2]
     max in each col = [14  5 24]
```

Comparing with multiple arrays:

```
[ ]  print(a)

     [[ 1  5  6]
      [14  4  8]
      [ 6 -2 24]]

     b = np.arange(9).reshape(3,3)
     print(b)

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

```
     print(np.minimum(a,b))

     [[ 0  1  2]
      [ 3  4  5]
      [ 6 -2  8]]
```

```
     print(np.maximum(a,b))

     [[ 1  5  6]
      [14  4  8]
      [ 6  7 24]]
```

Flattening of arrays;

We can flatten our array's by using: ravel() and flatten() functions.

```
     a = np.array([[1,8,5,-3],[15,6,4,2]])
     print(a)

     [[ 1  8  5 -3]
      [15  6  4  2]]
```

ravel() --> it is like a reference

```
     b = a.ravel()  #change will be reflected in original array also
     print(b)

     [ 1  8  5 -3 15  6  4  2]
```

```
b[1] = 45
print(b)
print(a)
```

```
[ 1 45  5 -3 15  6  4  2]
[[ 1 45  5 -3]
 [15  6  4  2]]
```

flatten() --> it is like a copy

```
c = a.flatten()   #chnage will not be reflected in original array
print(c)
```

```
[ 1  8  5 -3 15  6  4  2]
```

```
c[1] = 145

print(c)
print(a)
```

```
[  1 145   5  -3  15   6   4   2]
[[ 1 45  5 -3]
 [15  6  4  2]]
```

Day – 5: Working on Pandas

Installing and importing Pandas:

```
!pip install pandas
```

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

1-Dimensional data representation (Series):

```
#Creation of Series --> List, arrays, dictionaries, scalar

a = np.array([5,8,12,36,54])
print(a)
print(type(a))
```

```
[ 5  8 12 36 54]
<class 'numpy.ndarray'>
```

Converting the array into Pandas series:

```
#pd.Series(data,index,datatype)

b = pd.Series(a)    #if no index is mentioned it takes default RangeIndex(0,1,.....,n)
print(b)
print(type(b))
print(f'b.index = {b.index}')    #returns index
print(f'b.values = {b.values}')   #returns values in the array
print(type(b.values))
print(b.ndim)
```

```
0     5
1     8
2    12
3    36
4    54
dtype: int64
<class 'pandas.core.series.Series'>
b.index = RangeIndex(start=0, stop=5, step=1)
b.values = [ 5  8 12 36 54]
<class 'numpy.ndarray'>
1
```

Indexing Pandas series:

```
[ ]  c = pd.Series(a,index = list(range(1,6)))
     print(c)
     print(c.index)
```

```
1     5
2     8
3    12
4    36
5    54
dtype: int64
Int64Index([1, 2, 3, 4, 5], dtype='int64')
```

Creating Series from dictionary:

```
#Creating Series from a dictionary
dict = {'a':2, 'c':4, 'k':5}
dict
series = pd.Series(dict)    #key will be the index for this series and this series is mutable
print(series)
print(type(series))
print(f'series.index = {series.index}')
```

```
a    2
c    4
k    5
dtype: int64
<class 'pandas.core.series.Series'>
series.index = Index(['a', 'c', 'k'], dtype='object')
```

```
new = pd.Series(dict,index = [1,5,6])
new   #retuns NaN as no key is matching with the index
```

```
1   NaN
5   NaN
6   NaN
dtype: float64
```

```
new = pd.Series(dict,index=[1,5,6,'k'])
new
```

```
1    NaN
5    NaN
6    NaN
k    5.0
dtype: float64
```

Vectorized Operations in Series:

```
first_series = pd.Series([1,5,8,12],index = ['a','b','c','d'])
second_series = pd.Series([10,20,30,40],index = ['e','f','g','h'])
first_series + second_series    #returns Nan as both indexes are different
```

```
a   NaN
b   NaN
c   NaN
d   NaN
e   NaN
f   NaN
g   NaN
h   NaN
dtype: float64
```

```
first_series = pd.Series([1,5,8,12],index = ['a','b','c','d'])
second_series = pd.Series([10,20,30,40],index = list('abcd'))
d = first_series + second_series
d
```

```
a    11
b    25
c    38
d    52
dtype: int64
```

```
print(d.index)
print(d['a'])
```

```
Index(['a', 'b', 'c', 'd'], dtype='object')
11
```

```
print(d[0])
```

```
11
```

```
print(d[0:2])
```

```
a    11
b    25
dtype: int64
```

2-Dimensional data representation (Data Frames):

- Creating a data frame from an array.

```
data = np.arange(12).reshape(4,3)
print(data)

df = pd.DataFrame(data)   #default index and columns
df
```

```
[[ 0  1  2]
 [ 3  4  5]
 [ 6  7  8]
 [ 9 10 11]]
```

	0	1	2
0	0	1	2
1	3	4	5
2	6	7	8
3	9	10	11

```
print(f'df.index = {df.index}')
print(f'df.values = {df.values}')
print(f'df.columns = {df.columns}')
print(f'df.ndim = {df.ndim}')
```

```
df.index = RangeIndex(start=0, stop=4, step=1)
df.values = [[ 0  1  2]
 [ 3  4  5]
 [ 6  7  8]
 [ 9 10 11]]
df.columns = RangeIndex(start=0, stop=3, step=1)
df.ndim = 2
```

Creating a data frame from a series:

```
olympic_series_part = pd.Series([205,204,201,200,197], index = [2012,2008,2004,2000,1996])
olympic_series_country = pd.Series(['London','Beijing','Athens','Sydney','Atlanta'],index = [2012,2008,2004,2000,1996])
df_olympics = pd.DataFrame({'No. of countries':olympic_series_part,'host cities':olympic_series_country})
df_olympics
```

	No. of countries	host cities
2012	205	London
2008	204	Beijing
2004	201	Athens
2000	200	Sydney
1996	197	Atlanta

```
df_olympics.index
```

```
Int64Index([2012, 2008, 2004, 2000, 1996], dtype='int64')
```

```
df_olympics.columns
```

```
Index(['No. of countries', 'host cities'], dtype='object')
```

We can get precise summary from the data frame by using info().

```
df_olympics.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 5 entries, 2012 to 1996
Data columns (total 2 columns):
 #   Column            Non-Null Count  Dtype 
---  ------            --------------  ----- 
 0   No. of countries  5 non-null      int64 
 1   host cities       5 non-null      object
dtypes: int64(1), object(1)
memory usage: 120.0+ bytes
```

Accessing Columns:

```
df_olympics['No. of countries']
```

```
2012    205
2008    204
2004    201
2000    200
1996    197
Name: No. of countries, dtype: int64
```

```
df_olympics['No. of countries'][2012]   #pass some index
```

```
205
```

Accessing multiple columns:

```
#Accessing multiple columns

df_olympics[['host cities','No. of countries']][:2]
```

	host cities	No. of countries
2012	London	205
2008	Beijing	204

Accessing row-wise data using loc(), iloc():

```
df_olympics.index
```

```
Int64Index([2012, 2008, 2004, 2000, 1996], dtype='int64')
```

- loc() --> needs index as it is. eg: 2012,2004, etc.

```
df_olympics.loc[[2012,2004,2000]]
```

	No. of countries	host cities
2012	205	London
2004	201	Athens
2000	200	Sydney

Accessing rows and columns together:

```
#Accessing rows and columns together

df_olympics.loc[[1996,2012,2008]]['host cities']
```

```
1996    Atlanta
2012     London
2008    Beijing
Name: host cities, dtype: object
```

```
[ ] df_olympics['host cities'].loc[[1996,2012,2008]]
```

```
1996    Atlanta
2012     London
2008    Beijing
Name: host cities, dtype: object
```

- iloc() --> integer index location --> we can pass default index.

```
df_olympics.iloc[0]
```

```
No. of countries       205
host cities         London
Name: 2012, dtype: object
```

```
df_olympics.iloc[[0,2,4]]['No. of countries']
```

```
2012    205
2004    201
1996    197
Name: No. of countries, dtype: int64
```

```
df_olympics.iloc[:3]
```

	No. of countries	host cities
2012	205	London
2008	204	Beijing
2004	201	Athens

Filling values in a data frame.

- Forward filling

- Backward filling.

```
df1 = pd.DataFrame(np.arange(12).reshape(3,4), columns = list('abcd'))
df2 = pd.DataFrame(np.arange(20).reshape(4,5),columns = list('abcde'))
df1
```

	a	b	c	d
0	0	1	2	3
1	4	5	6	7
2	8	9	10	11

```
df2
```

	a	b	c	d	e
0	0	1	2	3	4
1	5	6	7	8	9
2	10	11	12	13	14
3	15	16	17	18	19

```
df1+df2
```

	a	b	c	d	e
0	0.0	2.0	4.0	6.0	NaN
1	9.0	11.0	13.0	15.0	NaN
2	18.0	20.0	22.0	24.0	NaN
3	NaN	NaN	NaN	NaN	NaN

As we got NaN values because two data frames are different in indexes and columns.

```
df1.add(df2,fill_value = 0)   #it fills missing values with 0 and perfroms add operation
```

	a	b	c	d	e
0	0.0	2.0	4.0	6.0	4.0
1	9.0	11.0	13.0	15.0	9.0
2	18.0	20.0	22.0	24.0	14.0
3	15.0	16.0	17.0	18.0	19.0

```
#To understand fillna() --> contains both bfill and ffill

df = pd.DataFrame(np.arange(15).reshape(5,3),
                  index = ['a','c', 'e', 'f', 'h'],
                  columns = ['one','two','three'])
df
```

	one	two	three
a	0	1	2
c	3	4	5
e	6	7	8
f	9	10	11
h	12	13	14

redindex() □ changes the index of the data frame.

```
df = df.reindex(list('abcdefgh'))
df
```

	one	two	three
a	0.0	1.0	2.0
b	NaN	NaN	NaN
c	3.0	4.0	5.0
d	NaN	NaN	NaN
e	6.0	7.0	8.0
f	9.0	10.0	11.0
g	NaN	NaN	NaN
h	12.0	13.0	14.0

Now filling missing values using fillna()

```
#filling missing values using fillna()

df3 = df.fillna(method = 'bfill',axis = 0)    #bfill-->takes next valid observation
df3
```

	one	two	three
a	0.0	1.0	2.0
b	3.0	4.0	5.0
c	3.0	4.0	5.0
d	6.0	7.0	8.0
e	6.0	7.0	8.0
f	9.0	10.0	11.0
g	12.0	13.0	14.0
h	12.0	13.0	14.0

```
df4 = df.fillna(method = 'ffill',axis = 0)    #ffill-->takes previous valid observation
df4
```

	one	two	three
a	0.0	1.0	2.0
b	0.0	1.0	2.0
c	3.0	4.0	5.0
d	3.0	4.0	5.0
e	6.0	7.0	8.0
f	9.0	10.0	11.0
g	9.0	10.0	11.0
h	12.0	13.0	14.0

Filling missing values in individual columns with mean:

```
df.fillna(df.mean(),inplace=True)   #inplace = True will make permanent changes
df
```

	one	two	three
a	0.0	1.0	2.0
b	6.0	7.0	8.0
c	3.0	4.0	5.0
d	6.0	7.0	8.0
e	6.0	7.0	8.0
f	9.0	10.0	11.0
g	6.0	7.0	8.0
h	12.0	13.0	14.0

Week – 2 (22-05-2023 to 27-05-2023)

Day & Date	Learning Outcomes
Day – 1	Working on Pandas (contd.)
Day – 2	EDA Case Study
Day – 3	Introduction to Scikit-Learn, Feature Scaling
Day – 4	Working on Underfitting and Overfitting Scenarios
Day – 5	Bias and variance trade off
Day – 6	Implementation Linear Regression, Polynomial Regression using case study.

Day – 1: Working on Pandas (contd.) and case study

Working on Pandas-

Importing packages:

```
#import the packages
import numpy as np
import pandas as pd
```

Aggregation and groups:

```
#Aggregations and Grouping
df_president_names = pd.DataFrame({'first':['George','Bill','Ronald','Jimmy','George'],
                                   'last':['Bush','Clinton','Regan','Carton','Washington']})
df_president_names
```

	first	last
0	George	Bush
1	Bill	Clinton
2	Ronald	Regan
3	Jimmy	Carton
4	George	Washington

Usage of group by:

```
[ ] grouped = df_president_names.groupby('first')
    grouped

    <pandas.core.groupby.generic.DataFrameGroupBy object at 0x000002A6A1AA47C0>

[ ] #as above dataframe is related to names we will above groupby() and group the data
    grp_data = grouped.get_group('George')
    grp_data
```

	first	last
0	George	Bush
4	George	Washington

Sorting according to values to: Sorting according to first name:

```
#Sorting according to values or index
df_president_names.sort_values('first')
```

	first	last
1	Bill	Clinton
0	George	Bush
4	George	Washington
3	Jimmy	Carton
2	Ronald	Regan

Usage of join():

- joining data frames in this case we have given common index.

```
df1 = pd.DataFrame({'Int_rate':[2,8,5,6],
                    'IND_GDP':[58,45,36,40]}
                    ,index=[2002,2003,2004,2005])
df2 = pd.DataFrame({'Low_Tier_HPI':[85,65,75,80],
                    'Unemployment':[5,2.3,4,6]},
                    index=[2002,2003,2005,2006])
joined = df1.join(df2)
joined
```

	Int_rate	IND_GDP	Low_Tier_HPI	Unemployment
2002	2	58	85.0	5.0
2003	8	45	65.0	2.3
2004	5	36	NaN	NaN
2005	6	40	75.0	4.0

Data Loading: Read from text files, Excel (.xlsx, .csv) and Web APIs.

```
data = pd.read_table('sample.txt',delimiter=' ') #we can use delimiter (default \t)
data
```

	sno	name	place
0	1	saketh	knr
1	2	sai	vzg
2	3	rahul	vjwda
3	4	sairam	gntr

Accessing data from opensource Web APIs and store the data in a table:

```
[ ] import requests

[ ] #take the Github pandas issues url
    url = "https://api.github.com/repos/pandas-dev/pandas/issues"

[ ] #connect to the webpage (url and check for the response code)
    data = requests.get(url)
    data

    <Response [200]>
```

As webpage data is always in string format we use json().

```
g = data.json()
g
type(g)

list
```

When we get Mutable collection first thing to do is check len().

```
len(g)

30

[ ] g[0]

{'url': 'https://api.github.com/repos/pandas-dev/pandas/issues/53332',
 'repository_url': 'https://api.github.com/repos/pandas-dev/pandas',
 'labels_url': 'https://api.github.com/repos/pandas-dev/pandas/issues/53332/labels{/name}',
 'comments_url': 'https://api.github.com/repos/pandas-dev/pandas/issues/53332/comments',
 'events_url': 'https://api.github.com/repos/pandas-dev/pandas/issues/53332/events',
 'html_url': 'https://github.com/pandas-dev/pandas/pull/53332',
 'id': 1719283559,
```

Creating a data frame by using above data:

```
#now let's create a dataframe by using above data
data = pd.DataFrame(g,columns=['id','number','title','state','created_at'])
data
```

	id	number	title	state	created_at
0	1719283559	53332	Update default value for `fill_value` paramete...	open	2023-05-22T09:26:44Z
1	1719207308	53331	DOC: Fixing EX01 - Added examples	open	2023-05-22T08:47:31Z
2	1718667266	53330	ENH: Deprecate literal json string input to re...	open	2023-05-21T22:52:26Z

Day – 2: EDA Case Study on titanic dataset

EDA stands for Exploratory Data Analysis.

```
data = pd.read_csv('titanic.csv')
data
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

count of unique rows(values) in the data frame.

```
[ ]  data['Survived'].value_counts()

     0    549
     1    342
     Name: Survived, dtype: int64
```

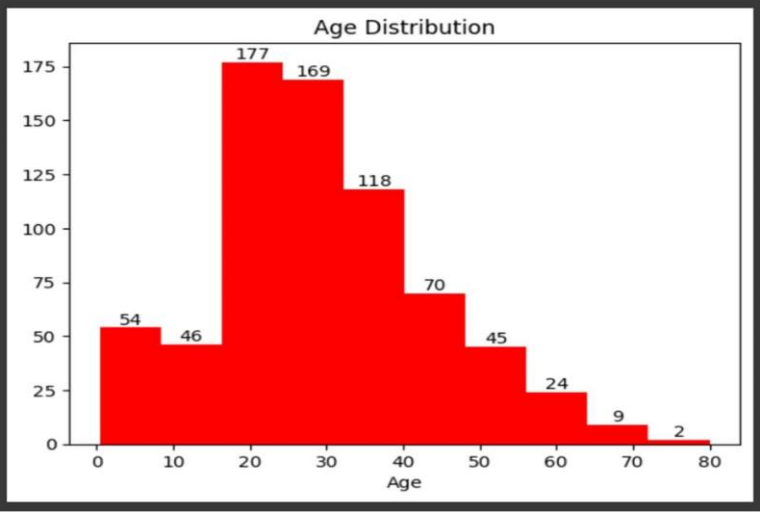
Data Visualization: using Matplotlib

Installing and importing matplotlib:

```
[ ]  pip install matplotlib
     import matplotlib
     import matplotlib.pyplot as plt
```

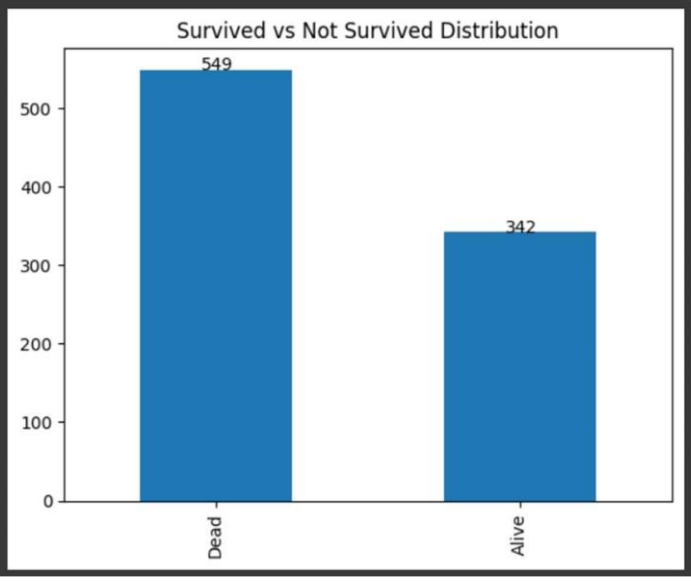
Visualizing age column using hist plot

```
#To display count on bars in the histogram
counts,edges,bars = plt.hist(data['Age'],color='red')
#print(counts)
plt.bar_label(bars) #adds label on the bars
plt.title("Age Distribution")
plt.xlabel("Age")#labelling the axes
plt.show()
```



Visualizing Survived column using bar graph:

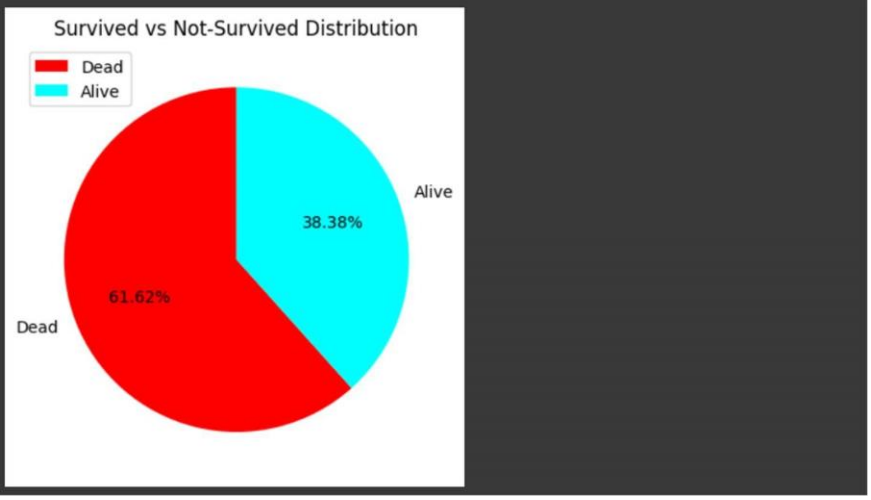
```
ax = data['Survived'].value_counts().plot(kind='bar')
plt.xticks([0,1],['Dead','Alive'])
plt.title("Survived vs Not Survived Distribution")
for i in ax.patches:
    ax.annotate((i.get_height()),(i.get_x()+0.2,i.get_height()))
plt.show()
```



Visualizing survived Vs Not-Survived using pie plot:

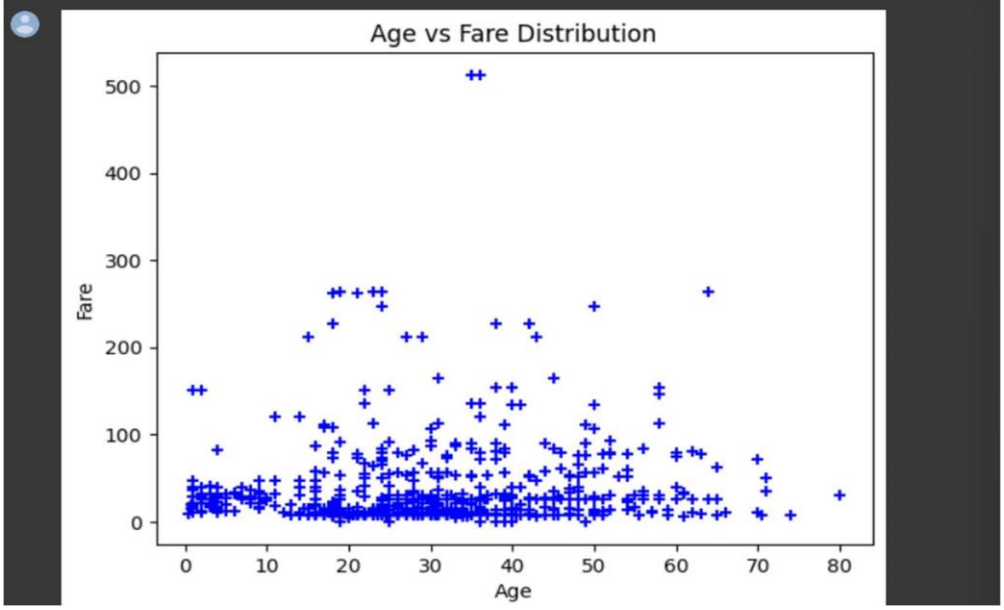
```
[ ]  d = data['Survived'].value_counts().index.tolist()
     print(d)
     e = data['Survived'].value_counts().values.tolist()
     print(e)

     [0, 1]
     [549, 342]
```

```
plt.pie(e,labels=['Dead','Alive'],colors=['red','cyan'],
        startangle=90,rotatelabels=False,autopct='%.2f%%')
plt.title("Survived vs Not-Survived Distribution")
plt.legend() #to displaythe labels on the graph
plt.show()
```



Visualizing relationship between two variables using scatter plot:

```
#Relationship between two variables -->Scatterplot()
plt.scatter(data['Age'],data['Fare'],color='blue',marker='+')
plt.title("Age vs Fare Distribution")
plt.xlabel("Age")
plt.ylabel("Fare")
plt.show()
```



To see descriptive statistics:

```
data['Age'].describe()
```

```
count    714.000000
mean      29.699118
std       14.526497
min        0.420000
25%       20.125000
50%       28.000000
75%       38.000000
max       80.000000
Name: Age, dtype: float64
```

Handling missing values: for numerical data

```
[ ] data['Age'].isnull().sum()

    177

[ ] data['Age'].fillna(data['Age'].mean(),inplace=True)
    data['Age'].isnull().sum()

    0
```

Similarly, we can fill all the values accordingly.

Handling categorical attributes using pandas get_dummies():

- It creates a new dataframe with the columns as unique values from the original column.

```
sex = pd.get_dummies(data['Sex'],drop_first=True)
sex
```

	male
0	1
1	0
2	0
3	0
4	1
...	...

Similarly, we use get_dummies() for the columns: embarked and pclass.

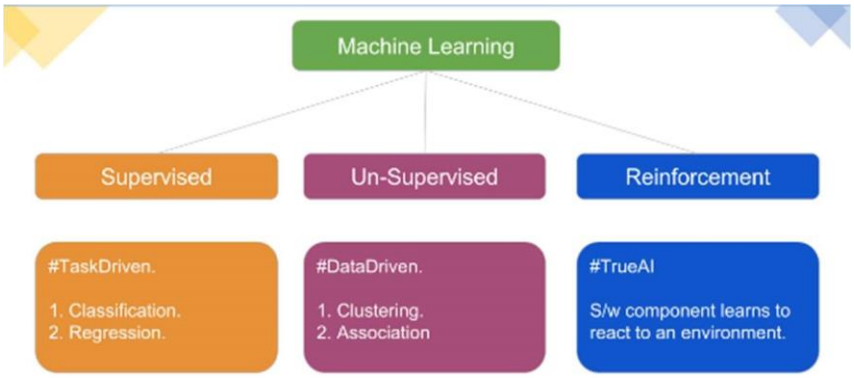
Day – 3: Introduction to scikit-learn and feature scaling

Scikit-Learn Design:

The main design principles are :

- Consistency - Estimators (fit()),Transformers (fit_transform()) and Predictors (predict() and score()) methods.
- Inspection - Estimators hyper-parameters.
- Non-proliferation of classes – Datasets are represented as NumPy arrays, instead of homemade classes.
- Composition - To create a Pipeline Estimator.
- Sensible defaults - Reasonable default values for most parameters.

Machine Learning techniques are mainly of three types:



Main steps to follow in building a machine learning model:

1. Gather the data (real world data).
2. Discover and visualize the data to gain insights.

3. Prepare the data for Machine Learning algorithms.
4. Select a model and train it.
5. Fine tune the model.
6. Present the solution.
7. Launch, monitor and maintain the system.

Feature Engineering:

- Features are individual independent variables. It is an attribute of a dataset.
- Feature Engineering - Translating a dataset into features which will be able to represent the dataset more effectively and result in a better learning performance
  It has two major elements:
  1. Feature Transformation: Transforms the data- structured or unstructured into a new set of features which can represent the underlying problem which machine learning is trying to solve. There are two variants of Feature Transformation:

  - Feature Construction: Feature Construction process discovers missing information about the relationships between features and augments the feature space by creating additional features.

  - Feature Extraction: Feature Extraction is the process of extracting or creating a new set of features from the original set of features by doing some transformation in order to remove redundancy. Thus, new features can represent the data very likely to the old features.

  2. Feature Subset-selection: Unlike Feature Transformation, in case of Feature Subset-selection (Feature Selection) no new feature is generated. The objective of feature selection is to derive a subset of features from the full feature set which is most meaningful in the context of a specific machine learning problem.

Feature Scaling:

Feature scaling in machine learning is one of the most critical steps during the pre-processing of data before creating a machine learning model.

Scaling can make a difference between a weak machine learning model and a better one.

The most common techniques of feature scaling are Normalization and Standardization.

Normalization is used when we want to bound our values between two numbers, typically, between [0,1] or [-1,1]. While Standardization transforms the data to have zero mean and a variance of 1, they make our data unitless.

Min-max normalization (or simply Normalization)

$$x_{scaled} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

Standard scalar normalization (or simple Standardization):

$$z = \frac{x - \mu}{\sigma}$$

μ = Mean
σ = Standard Deviation

Day – 4: Working on Underfitting and Overfitting Scenarios

Importing required libraries:

```
#importing the libraries
import numpy as np
import pandas as pd
%matplotlib inline
import matplotlib.pyplot as plt
```

```
[2] import warnings
    warnings.filterwarnings('ignore')
```

Creating synthetic data to understand underfitting, overfitting and ideal model scenario.

```
n_points = 20
#defining the input
x = np.random.uniform(0,11,n_points)
x
x = np.sort(x)
x
```

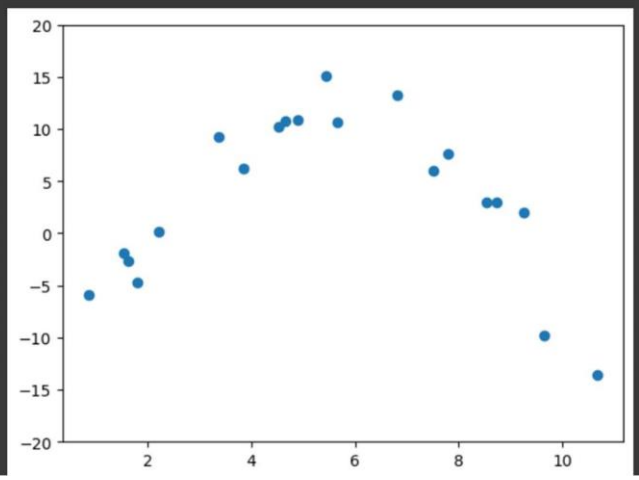
```
array([ 0.84675699,  1.54100053,  1.6157265 ,  1.7980201 ,  2.21421831,
        3.35931447,  3.85646975,  4.53276807,  4.66184153,  4.89188348,
        5.43258425,  5.65629111,  6.82324671,  7.50695369,  7.79770552,
        8.54435966,  8.73826711,  9.25092683,  9.65595179, 10.68745365])
```

```
#defining dependent variable
y = (-x+2)*(x-9) + np.random.normal(0,3,n_points)
y
```

```
array([ -5.85425173,  -1.90165988,  -2.70438844,  -4.719554  ,
         0.21147196,   9.22035905,   6.22044002,  10.26554694,
        10.81020205,  10.91952074,  15.10523944,  10.70004635,
        13.21318473,   5.99539816,   7.67340372,   2.97903923,
         2.98006778,   1.98946202,  -9.75128927, -13.54427923])
```

Visualizing x and y using scatter plot.

```
[5] import matplotlib as mpl
    mpl.rcParams['figure.dpi'] = 100
    plt.scatter(x,y)
    plt.ylim(-20,20)
    plt.show()
```

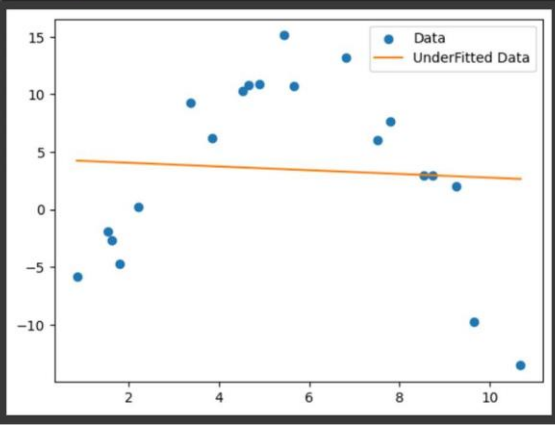


This looks like the shape of a parabola, as expected for a quadratic transformation of x. However, we can see the noise in the fact that not all the points appear that they would lie perfectly on the parabola.

In our synthetic example, we know the data generating process: the response variable y is a quadratic transformation of the feature x. In general, when building machine learning models, the data generating process is not known. Instead, several candidate features are proposed, a model is proposed, and an exploration is made of how well these features and this model can explain the data.

In this case, we would likely plot the data, observe the apparent quadratic relationship, and use a quadratic model. However, for the sake of illustration of an underfit model, what does a linear model for these data look like?

```
lin_fit = np.polyfit(x,y,1) #least Squares Polynomial Fit
#let's visualize the above scenario
cmap = mpl.cm.get_cmap('tab10') #cmap -->color map(pick on your own)
plt.scatter(x,y,label = "Data",color=cmap(0))
plt.plot(x,np.polyval(lin_fit,x),label = "UnderFitted Data",
        color = cmap(1)) #polyval -->Evaluating Polynomial with specific givenvalues
plt.legend() #display the labels
plt.show()
```



We can fit a polynomial model of degree 1, in other words a linear model, with NumPy's polyfit.

Doesn't look like a very good fit, does it!

Now let's imagine we have many features available, that are polynomial transformations of x, specifically x2, x3, … x15. This would be a large number of features relative to the number of samples (20) that we have. Again, in "real life", we probably wouldn't consider a model with all these features, since by observation we can see that a quadratic model, or 2nd degree polynomial, would likely be sufficient. However, identifying the ideal features isn't always so straightforward. Our illustrative example serves to show what happens when we are very clearly using too many features.

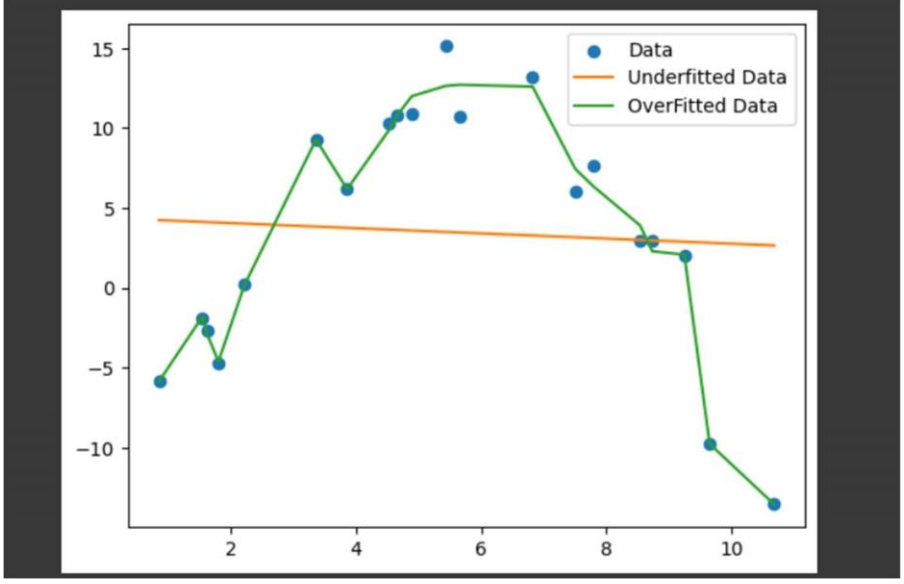
Let's make a 15 - degree polynomial fit:

```
high_poly_fit = np.polyfit(x,y,15)
high_poly_fit
#These are the coefficients of all the powers of x from 1 through 15 in this model
```

```
array([ 8.40459678e-07, -6.18651496e-05,  2.02121489e-03, -3.83836771e-02,
        4.63587479e-01, -3.61035539e+00,  1.66491290e+01, -2.27662581e+01,
       -2.51771401e+02,  2.03633506e+03, -7.92425163e+03,  1.90155249e+04,
       -2.90584791e+04,  2.73324161e+04, -1.42964732e+04,  3.14933671e+03])
```

These are the coefficients of all the powers of x from 1 through 15 in this model, and the intercept. Notice the widely different scales of the coefficients: some are close to zero, while others are fairly large in terms of magnitude (absolute value). Let's plot this model as well. First, we generate a large number of evenly spaced points over the range of x values, so we can see how this model looks not only for the particular values of x used for model training, but also for values in between.

```
[9] plt.scatter(x,y,label = "Data",color=cmap(0))
    plt.plot(x,np.polyval(lin_fit,x),label="Underfitted Data",color = cmap(1))
    plt.plot(x,np.polyval(high_poly_fit,x),label = "OverFitted Data",
             color = cmap(2))
    plt.legend() #displaying the labels
    plt.show()
```



This is a classic case of overfitting. The overfit model passes nearly perfectly through all the training data. However, it's easy to see that for values in between, the overfit model

does not look like a realistic representation of the data generating process. Rather, the overfit model has become tuned to the noise of the training data. This matches the definition of high variance given above.

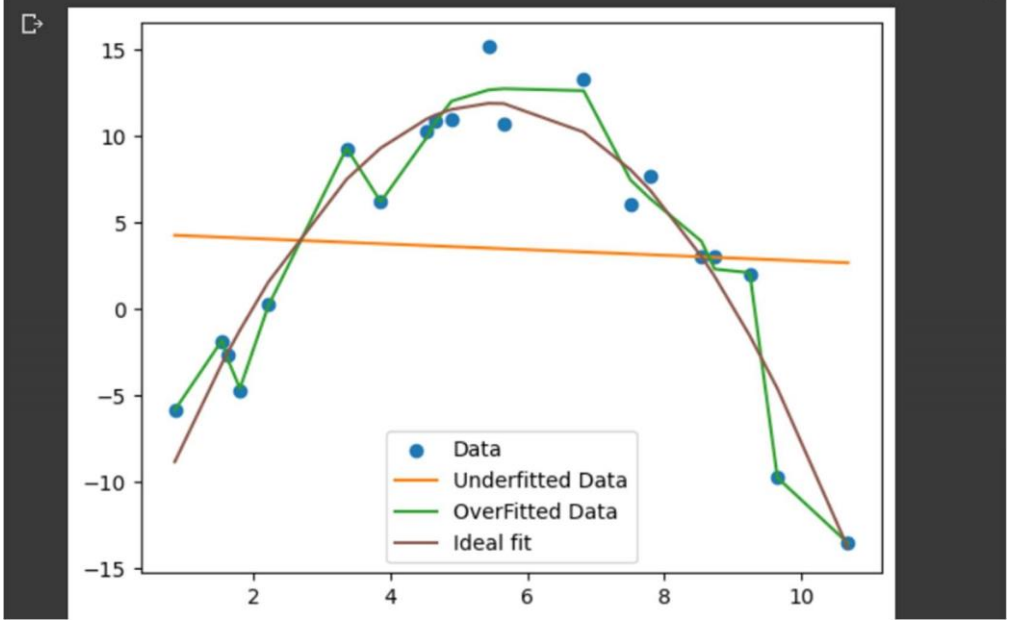
In this last graph, you can see another definition of high variance: a small change in the input x can result in a large change in the output y.

Further, you can imagine that if we generated a new, larger synthetic data set, using the same quadratic function $y=(-x+2)(x-9)$, but added new randomly generated noise ϵ according to the same distribution we used above, then randomly sampled 20 points and fit the highdegree polynomial, the resulting model would look much different. It would pass almost perfectly through these new noisy points and the coefficients of the 15-degree polynomial would be very different to allow this to happen. Repeating this process with different samples of 20 points, would continue to result in highly variable coefficient estimates. In other words, the coefficients would have high variance between samples of the data used for model training. This is yet another definition of a model with high variance.

With our synthetic data, since in this case we know the data generating process, we can see how a 2nd degree polynomial fit looks in comparison with the underfit and overfit models.

Ideal Fit Scenario:

```
##Ideal Fit Scenario
plt.scatter(x,y,label = "Data",color=cmap(0))
plt.plot(x,np.polyval(lin_fit,x),label="Underfitted Data",color = cmap(1))
plt.plot(x,np.polyval(high_poly_fit,x),label = "OverFitted Data",
        color = cmap(2))
plt.plot(x,np.polyval(np.polyfit(x,y,2),x),label = "Ideal fit",color=cmap(5))
plt.legend() #displaying the labels
plt.show()
```



Day – 5: Bias and variance trade off

Bias - Difference between the actual prediction of our model to the correct value trying to predict.

Variance - It is complete opposite to Bias. We can define it as a Model's Sensitivity to the fluctuations in the data.

Importing required libraries:

```
[ ] import numpy as np
    import pandas as pd
    %matplotlib inline
    import matplotlib.pyplot as plt

[ ] import warnings
    warnings.filterwarnings('ignore')
```

Importing seaborn as it also contains inbuilt datasets.

```
[ ] import seaborn as sn
    #Seaborn makes hard things easier (default labelling,styling and so many factors)

[ ] print(sn.get_dataset_names())

    ['anagrams', 'anscombe', 'attention', 'brain_networks', 'car_crashes', 'diamonds', 'dots', 'dowjones', 'exerci
```

Loading tips dataset.

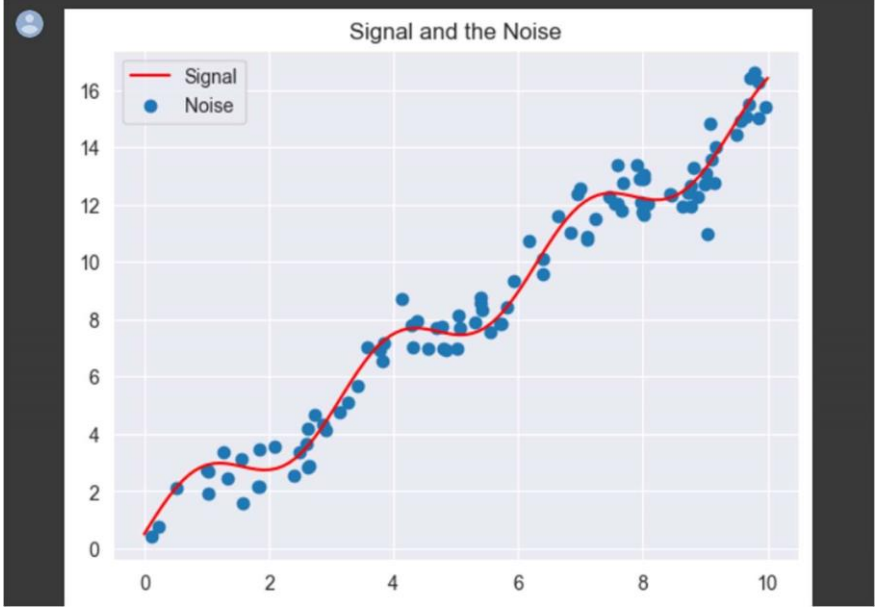
```
#load any dataset from above
sn.load_dataset('tips')
```

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4
...	...	...	...	...	...	...	...

Setting up the styling part for visualization.

```
[ ] #we will setup the styling part
    sn.set_style('darkgrid')

    #create some data and plotting it
    def generate_data(x,noise_threshold = 0.1):
        signal = np.sin(2*x)+(1.5*x) + 0.5
        noise = np.random.normal(0,1,size=len(x))*noise_threshold
        return signal + noise
    x = np.random.uniform(0,10,100)
    y = generate_data(x=x,noise_threshold=0.7)
```

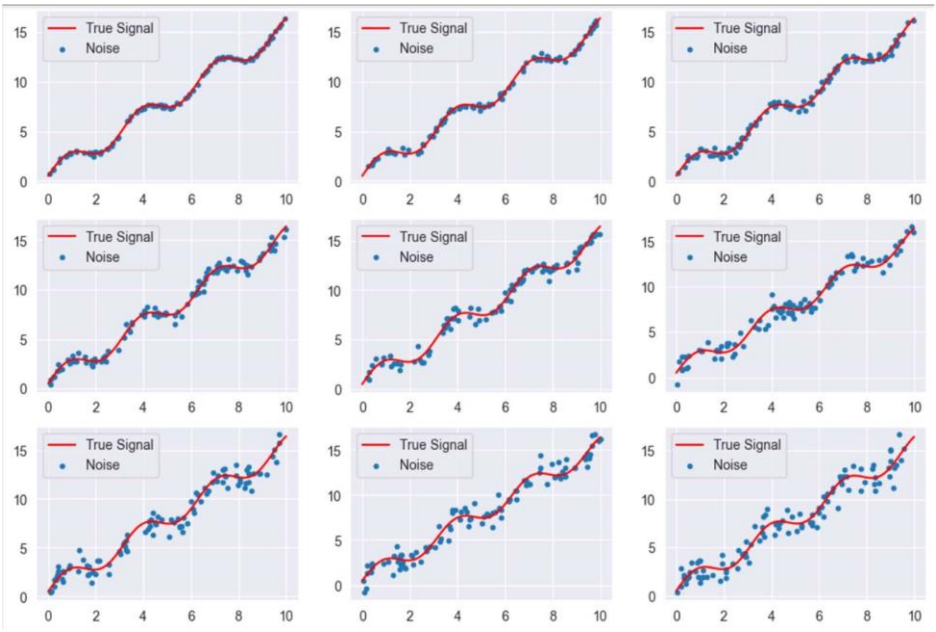
```
signal_x = np.linspace(0,10,100)
signal_y = generate_data(x=signal_x,noise_threshold=0.0)
plt.plot(signal_x,signal_y,color='red',label="Signal")

x = np.random.uniform(0,10,100)
y = generate_data(x=x,noise_threshold=0.7)
plt.scatter(x,y,label="Noise")
plt.title("Signal and the Noise")
plt.legend()
plt.show()
```



In the presence of an imbalanced dataset, using weighted or penalized models can be considered as an alternative. The most common source of error is generally due to the training dataset not being diverse in nature and hence the model being created not having enough training data to clearly identify or differentiate between a problem. Introducing more data increases the data to noise ratio which may help reduce the variance of the present model. When the model is fed with more data, it has shown to be able to come up with a better understanding of the data which is then also applied to newly introduced data points.

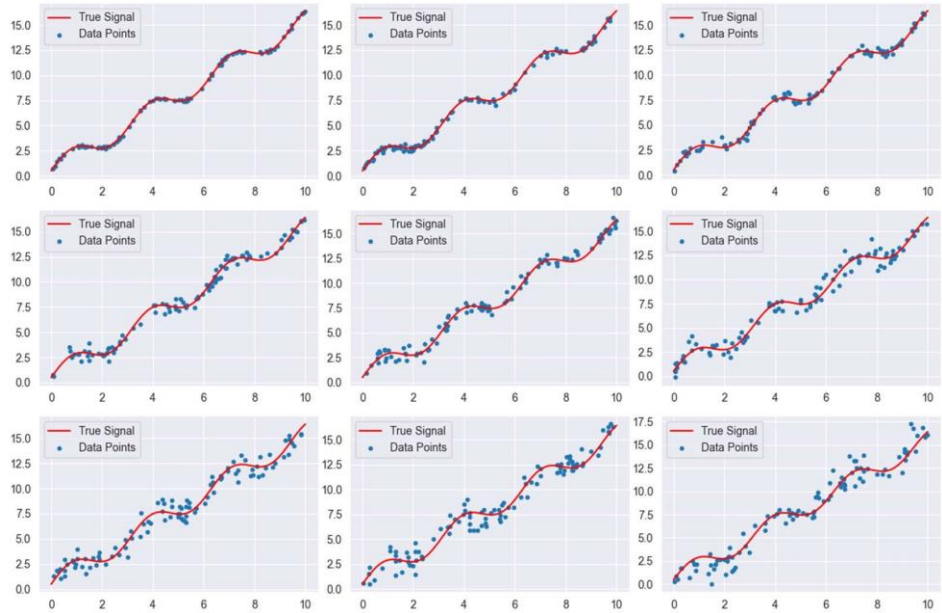
Creating a high bias and low variance model.

```
plt.figure(figsize=(12,8))
for i in range(9):
    noise_threshold = 0.1*(i+1) #varying noise levels
    x = np.random.uniform(0,10,100)
    y = generate_data(x=x,noise_threshold=noise_threshold)

    plt.subplot(3,3,i+1)
    plt.plot(signal_x,signal_y,color='red',label = "True Signal")
    plt.scatter(x=x,y=y,s=10,label = "Noise")
    plt.legend()
plt.show()
```



Creating low bias and high variance model:

```
plt.figure(figsize=(12,8))
for i in range(9):
    noise_threshold = 0.1 * (i+1)  #varying noise levels
    x = np.random.uniform(0,10,100)
    y = generate_data(x = x,noise_threshold=noise_threshold)

    plt.subplot(3,3,i+1)
    plt.plot(signal_x,signal_y,color='red',label = "True Signal")
    plt.scatter(x=x,y=y,s =10,label="Data Points")
    plt.legend()

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



Day – 6: Implementation of Linear Regression and Polynomial Regression using case study.

Implementing using pure mathematical approach from scratch.

Importing required libraries:

```
[ ]  #Import the libraries and read the dataset
     import numpy as np
     import pandas as pd
     import matplotlib.pyplot as plt
```

Reading 'Advertisement' dataset:

```
#read the dataset
data = pd.read_csv('Advertising.csv')
data
```

	Unnamed: 0	TV	radio	newspaper	sales
0	1	230.1	37.8	69.2	22.1
1	2	44.5	39.3	45.1	10.4
2	3	17.2	45.9	69.3	9.3
3	4	151.5	41.3	58.5	18.5
4	5	180.8	10.8	58.4	12.9
...	...	...	...	...	...

Getting precise summary of dataset using info():

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 5 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Unnamed: 0  200 non-null    int64
 1   TV          200 non-null    float64
 2   radio       200 non-null    float64
 3   newspaper   200 non-null    float64
 4   sales       200 non-null    float64
dtypes: float64(4), int64(1)
memory usage: 7.9 KB
```

Dropping unnecessary columns:

```
data.drop(columns=['Unnamed: 0'],inplace=True)
```

Considering TV column as input and sales column as output:

```
[ ]  X = data['TV'].values
     type(X)
```

```
numpy.ndarray
```

```
[ ]  Y = data['sales'].values
     type(Y)
```

```
numpy.ndarray
```

```
[ ] mean_x = X.mean()
    mean_y = Y.mean()
```

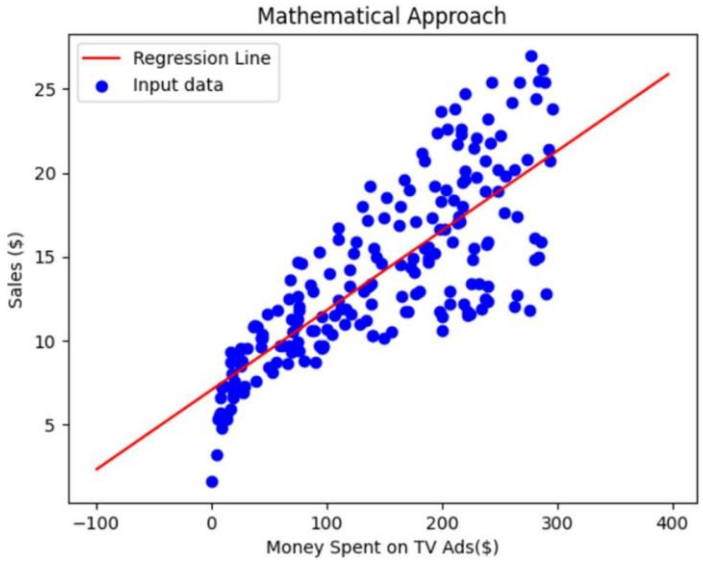
```
[ ] # y = m*x + c (or) y = b1*x + b0
    #Acc.to the formula we will take two variables numerator and denominator
    #slope = (X[0] - mean_x)*(Y[0] - mean_y)/(X[0] - mean_x)**2 #first point in similar way
    #we will generalize it
    n = len(X)
    numer = 0;denom = 0
    for i in range(n):
        numer += (X[i] - mean_x)*(Y[i] - mean_y)
        denom += (X[i] - mean_x)**2
    b1 = numer/denom
    print(b1)
    #To find the intercept which is b0
    b0 = mean_y - (b1*mean_x)
    print(b0)
```

```
0.04753664043301969
7.032593549127704
```

```
[ ] #Line equation can be represented as
    print("Linear Model is Y=",b1,"*x+",b0)
```

```
Linear Model is Y= 0.04753664043301969 *x+ 7.032593549127704
```

```
#Now we will try to add new data points(Samples) into out input and then
#sub. b1 and b0 values
min_x = np.min(X) - 100
max_x = np.max(X) + 100
#print(min_x,max_x) #for boundary values
#based on the boundary we will create new values
x = np.linspace(min_x,max_x,1000)
#substitute the above values in our line equaton
y = b0 + b1*x
#plot the graphs for Actual data and the new values
plt.plot(x,y,color='red',label="Regression Line")
plt.scatter(X,Y,color='blue',label = "Input data")
plt.xlabel("Money Spent on TV Ads($)")
plt.ylabel("Sales ($)")
plt.title("Mathematical Approach")
plt.legend()
plt.show()
```



```
[ ] #let's check the predicted values
    print(X[2])
```

```
17.2
```

```
[ ] print(Y[2])
```

```
9.3
```

```
[ ] #check with our model approach
    y[2] = b1*X[2]+b0
    y[2]
```

```
7.8502237645756425
```

R-Squared value:

R-squared value is the statistical measure of how close the data are to the fitted regression line. It is also known as the Coefficient of Determination or Coefficient of Multiple Determination for multiple regression.

R-squared always between 0 and 100%.

0% indicates that the model explains none of the variability of the response data around its mean.

100% indicates that the model explains all the variability of the response data around its mean.

```
#1 - ∑(y predicted - y )^2(ss_res)/ ∑(y - mean of y)^2 (ss_tot)
#we can pick two variables sum of residuals and sum of squares
ss_tot = 0
ss_res = 0
n = len(X)
for i in range(n):
    y_pred = b0+b1*X[i]
    ss_tot += (Y[i] - mean_y)**2
    ss_res += (y_pred - Y[i])**2
r2 = 1-(ss_res/ss_tot)
print(r2)
print(r2*100)
```

```
0.6118750508500708
61.187505085007075
```

Implementing using scikit-learn:

```
[ ] #Implemenation using scikit-learn
    from sklearn.linear_model import LinearRegression
```

```
[ ] data
```

	TV	radio	newspaper	sales
0	230.1	37.8	69.2	22.1
1	44.5	39.3	45.1	10.4
2	17.2	45.9	69.3	9.3
3	151.5	41.3	58.5	18.5
4	180.8	10.8	58.4	12.9
...	...	...	...	...

Dividing dataset into dependent and independent variables:

```
[ ] X = data['TV'].values
    X.ndim
```

```
1
```

```
[ ] Y = data['sales'].values
    Y.ndim
```

```
1
```

```
[ ] #building the model
    lin_reg = LinearRegression()
    #lin_reg.fit(X,Y) #as it results valueerror need to transform to 2-Darray
```

```
    X = data['TV'].values.reshape(-1,1) #we are having only one feature with many samples
    X.shape
```

```
(200, 1)
```

```
[ ] Y = data['sales'].values.reshape(-1,1)
```

```
[ ] lin_reg.fit(X,Y)
```

```
LinearRegression()
```

```
[ ] #Let's check the coefficient and intercept
    print(lin_reg.coef_)
    print(lin_reg.intercept_)
```

```
[[0.04753664]]
[7.03259355]
```

```
[ ] #now let's go for predictions
    pred = lin_reg.predict(X[2].reshape(-1,1))
    pred
```

```
array([[7.85022376]])
```

```
[ ] #Now let's check the score value or coefficient of determination
    score = lin_reg.score(X,Y)
    score
```

```
0.611875050850071
```

Now let’s consider all 3 columns as input:

```
[ ] X = data[['TV','newspaper','sales']]
    #X
    X = data.iloc[:,:-1]
    #X
```

```
[ ] Y = data.iloc[:,-1]
    Y
```

```
0      22.1
1      10.4
2       9.3
3      18.5
4      12.9
       ...
195     7.6
196     9.7
197    12.8
198    25.5
199    13.4
Name: sales, Length: 200, dtype: float64
```

Splitting data into testing and training:

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

```
[ ] x_train,x_test,y_train,y_test = train_test_split(X,Y,test_size=0.25,random_state=42)
```

```
[ ] x_train.shape

    (150, 3)
```

```
[ ] #building the model
    lin_reg = LinearRegression()
    lin_reg.fit(x_train,y_train)

    LinearRegression()
```

```
    #we can check the predictions
    predictions = lin_reg.predict(x_test)
```

Calculating metrics using mean squared error and r2 score:

```
[ ] from sklearn.metrics import mean_squared_error,r2_score
```

```
[ ] mean_squared_error(y_test,predictions)

    2.880023730094191
```

```
[ ] r2_score(y_test,predictions)

    0.8935163320163658
```

Implementing Polynomial Regression:

Implementing from scratch:

```
[ ] #To generate some non-linear data
    m=100
    x = 6*np.random.rand(m,1)-1 #rand()which includes both +ve & -ve
```

```
[ ] #let's create a quadratic equation
    y = 0.5*-x**2 + x + 2 +np.random.rand(m,1) #ax^2 + bx+c
```

Applying linear regression:

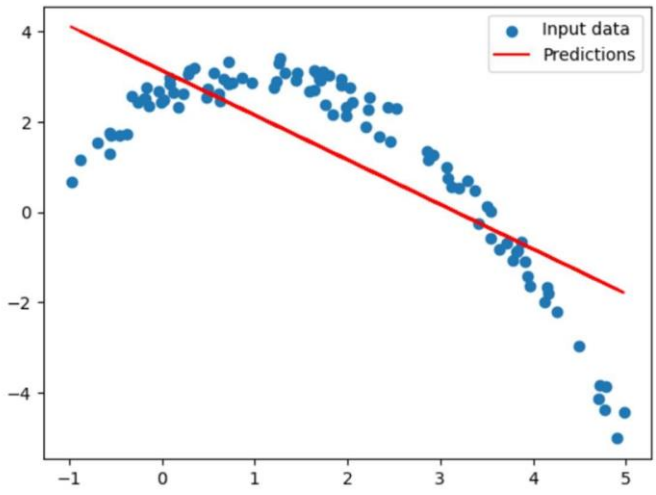
```
[ ] lin_reg = LinearRegression()
    lin_reg.fit(x,y)

    LinearRegression()
```

```
[ ] #we can check the predictions
    lin_pred = lin_reg.predict(x)
```

```
[ ] #cal.the metrics
    print("Mean Squared Error",mean_squared_error(y,lin_pred))
    print("R-Squared Value is",r2_score(y,lin_pred))
```

```
Mean Squared Error 1.656535345406941
R-Squared Value is 0.6146605168809411
```

```
[ ] #let's visualize the data
    plt.scatter(x,y,label="Input data")
    plt.plot(x,lin_pred,color="red",label="Predictions")
    plt.legend()
    plt.show()
```



So, by observing the above graphs we transform our data using PolynomialFeatures of scikitlearn

```
[ ] from sklearn.preprocessing import PolynomialFeatures
```

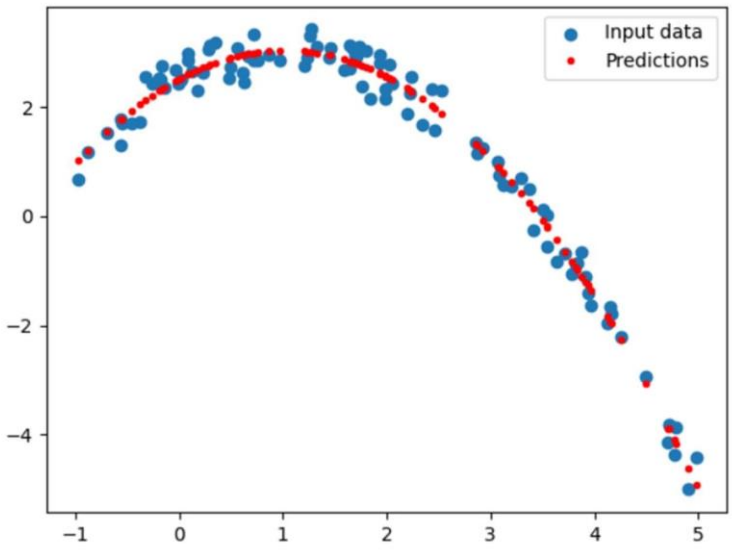
```
[ ] poly_features = PolynomialFeatures(degree=2,include_bias=False)
    x_poly = poly_features.fit_transform(x) #tranforming input
```

```
#Once transformation is done we will apply Linear Regression
poly_lr = LinearRegression()
poly_lr.fit(x_poly,y)
```

```
LinearRegression()
```

```
[ ] #check the predictions
    poly_pred = poly_lr.predict(x_poly)
```

```
[ ]  plt.scatter(x,y,label="Input data")
     plt.plot(x,poly_pred,'.',color="red",label="Predictions")
     plt.legend()
     plt.show()
```




```
[ ]  #cal.the metrics
     print("Mean Squared Error",mean_squared_error(y,poly_pred))
     print("R-Squared Value is",r2_score(y,poly_pred))
```

```
Mean Squared Error 0.07099645060251165
R-Squared Value is 0.9834849671911232
```

Week – 3 (29-05-2023 to 03-06-2023)

Day & Date	Learning Outcomes
Day – 1	Web Scraping and Model Building
Day – 2	Web Scraping and Model Building contd.
Day – 3	Implementation of Regularization with House Pricing Case Study
Day – 4	Linear Regression, Ridge Regression, LASSO Regression
Day – 5	Logistic Regression

Day – 6	Introduction to Decision Tree

Day – 1: Web Scraping and Model Building

Web scraping is the process of extracting data from websites automatically. It involves writing a program or script that retrieves web pages, parses the HTML or XML content, and extracts the desired information. Here are the main steps involved in web scraping:

1. Identify the Target Website: Determine the website from which you want to scrape data. Understand its structure, the pages containing the desired information, and any rules or limitations set by the website.

2. Choose a Web Scraping Tool or Library: There are several tools and libraries available in various programming languages to facilitate web scraping. Popular choices include Python libraries like BeautifulSoup, Scrapy, and Selenium.

3. Analyze the Website's Structure: Examine the HTML structure of the web pages you want to scrape. Identify the specific elements (tags, classes, IDs) that contain the data you need.

4. Send HTTP Requests: Use the web scraping tool or library to send HTTP requests to the target website's server. Retrieve the HTML content of the web pages.

5. Parse the HTML Content: Parse the HTML content of the web pages using the chosen library or tool. Extract the desired data by selecting specific elements based on their tags, classes, or IDs.

6. Data Extraction: Extract the relevant data from the parsed HTML content. This may involve navigating through the HTML tree structure, using CSS selectors, or applying regular expressions to locate and extract the desired information.

7. Data Cleaning and Transformation: Clean and preprocess the extracted data as needed. This may involve removing unnecessary characters, converting data types, handling missing values, or performing any necessary transformations.

8. Storage and Analysis: Store the extracted data in a suitable format, such as a database, CSV file, or JSON file. Perform any further analysis, visualization, or processing on the collected data.

9. Handle Website Access Policies: Be mindful of the website's terms of service and any access policies. Respect the website's robots.txt file, avoid overwhelming the server with too many requests, and incorporate delays or timeouts between requests to avoid being blocked.

Remember to always comply with legal and ethical guidelines when performing web scraping. Respect the website's terms of service, ensure that you are scraping publicly available data, and be mindful of the impact of your scraping activities on the website's server.

Same above steps technically

-->Sending a HTTP GET request to the url of the webpage by using request library

-->Fetching and Parsing the data using BeautifulSoup and maintain the data in some dicts/lists

-->Analyzing the HTML tags and their attributes

-->Output the data in any file format(.csv,.xlsx,.json)

Implementation:

```
[ ]  #import the packages
     import requests
     import bs4
```

```
[ ]  #help(bs4)
```

```
[ ]  from bs4 import BeautifulSoup
```

```
     #Data Collection/Gathering process of ML life cycle we are using dynamic website (makaan)
     url = "https://www.makaan.com/vijayawada-residential-property/buy-property-in-vijayawada-city"
```

```
     #connect to the webpage and get the response
     data = requests.get(url)
     data
```

```
<Response [200]>
```

```
[ ]  #accessing content from the webpage
     soup = BeautifulSoup(data.text,'html')
     #soup
```

```
     #let's extract the property details one by one
     soup.find('div',attrs={'class':'title-line'}).text
```

```
'3 BHK Apartment'
```

```
     #we want only specific number
     soup.find('a',attrs={'class':'typelink'}).span.text.strip()
```

```
'3'
```

```
[ ]  soup.find('span',attrs={'itemprop':'addressLocality'}).text
```

```
[ ]  #As prices are in Lakhs and Crores -->we will label them to same scale
     Price = []
     for i in c:
         i = i.text.strip()
         if "Cr" in i:
             i = i.replace(' Cr','') #removing Cr from the string
             #convert the number to lakhs
             i = float(i) * 100 #convert values to lakhs
         else:
             i = i.replace(' L','')#remove 'L' from the string
             i = float(i)
         Price.append(i)
         #print(Price)
     print(Price)
```

```
[55.5, 48.0, 50.0, 89.0, 59.86, 65.0, 40.0, 56.0, 95.0, 48.0, 110.00000000000001, 16.83, 33.0, 20.0, 166.0, 74.0, 56.0, 166.0, 57.05, 37.18]
```

```
     #We do same process for total area,construction status and details of the property
     c = soup.find_all('td',attrs={'class':'size'})
     size = [i.text.strip() for i in c]
     print(size)
     print(len(size))
```

```
['1587', '1450', '1290', '2062', '1663', '2000', '1200', '1500', '1600', '1150', '3700', '1782', '1800', '1500', '2317', '1560', '1000', '2317', '1100', '648']
20
```

```
[ ]  d = soup.find_all('td',attrs={'class':'val'})
     status = [i.text for i in d]
     print(status)
     print(len(status))
```

```
['Ready to move', 'Ready to move', 'Under Construction', 'Ready to move', 'Ready to move', 'Ready to move', 'Ready to move', 'Ready to move', 'Under Construction', 'Under
20
```

```
[ ] #to get only type of building separately
    d = soup.find_all('div',attrs={'class':'title-line'})
    property_types = ['Apartment','Villa','Independent House','Residential Plot','Builder Floor','Independent Floor'
                      'Studio Apartment']
    Type = []
    for i in d:
        i = i.text
        for p_type in property_types:
            if p_type in i:
                Type.append(p_type)
    print(Type)
    print(len(Type))
```

```
['Apartment', 'Apartment', 'Independent House', 'Apartment', 'Apartment', 'Apartment', 'Apartment', 'Independent House', 'Independent House', '
18
```

```
url = "https://www.makaan.com/vijayawada-residential-property/buy-property-in-vijayawada-city?page="
for i in range(1,11):
    print(url+str(i))
```

```
https://www.makaan.com/vijayawada-residential-property/buy-property-in-vijayawada-city?page=1
https://www.makaan.com/vijayawada-residential-property/buy-property-in-vijayawada-city?page=2
https://www.makaan.com/vijayawada-residential-property/buy-property-in-vijayawada-city?page=3
https://www.makaan.com/vijayawada-residential-property/buy-property-in-vijayawada-city?page=4
https://www.makaan.com/vijayawada-residential-property/buy-property-in-vijayawada-city?page=5
https://www.makaan.com/vijayawada-residential-property/buy-property-in-vijayawada-city?page=6
https://www.makaan.com/vijayawada-residential-property/buy-property-in-vijayawada-city?page=7
https://www.makaan.com/vijayawada-residential-property/buy-property-in-vijayawada-city?page=8
https://www.makaan.com/vijayawada-residential-property/buy-property-in-vijayawada-city?page=9
https://www.makaan.com/vijayawada-residential-property/buy-property-in-vijayawada-city?page=10
```

Day – 2: Web Scraping and Model Building contd.

```
#we create empty lists for final data from multiple pages

import time
from IPython.display import clear_output
property_types = ['Apartment','Villa','Independent House',
                  'Residential Plot','Builder Floor',
                  'Independent Floor',
                  'Studio Apartment']

base_url = "https://www.makaan.com/vijayawada-residential-property/buy-property-in-vijayawada-city?page="
for i in range(3,65):
    url = base_url + str(i)
    print(f'Scraping info from page {i}') #fstring represtn
    clear_output(wait=True)

    #we will pause for atleast 3 sec
    time.sleep(2)
    response = requests.get(url)
    soup = BeautifulSoup(response.text,'html')

    a = soup.find_all("div",attrs={'class':'title-line'})
    for i in a:
        i = i.span.text
        i=i.replace('Residential Plot','0').replace(' ','')
        Bedrooms.append(i)

    b = soup.find_all('span',attrs={'itemprop':'addressLocality'})
    for i in b:
        Location.append(i.text)

    c = soup.find_all('td',attrs={'class':'price'})
    for i in c:
        i = i.text.strip()
        if "Cr" in i:
            i = i.replace(' Cr','') #removing Cr from the string
            #convert the number to lakhs
            i = float(i) * 100 #convert values to lakhs
        else:
            i = i.replace(' L','')#remove 'L' from the string
            i = float(i)
        Price.append(i)

    d = soup.find_all('td',attrs={'class':'size'})
```

```
f = soup.find_all('ul',attrs={'class':'listing-details'})
for i in f:
    i = i.text
    #print(i)
    bathroom_count = re.findall(r'(\d+) Bathrooms',i)
    if bathroom_count:
        Bathrooms.append(int(bathroom_count[0]))
    else:
        Bathrooms.append(0)
    facing_direction = re.findall(r'(North|South|East|West|NorthEast|Northwest|SouthEast|Southwest) Facing',i)
    if facing_direction:
        Facing.append(facing_direction[0])
    else:
        Facing.append(0)

g = soup.find_all('div',attrs={'class':'title-line'})
for i in g:
    i = i.text
    for p_type in property_types:
        if p_type in i:
            Type.append(p_type)
```

```
Scraping info from page 62
```

```
print(len(Bedrooms))
print(len(Location))
print(len(Size))
print(len(Availability))
print(len(Price))
print(len(Bathrooms))
print(len(Facing))
print(len(Type))
```

```
1221
1221
1221
1221
```

```
# Find the length of the longest list
max_length = max(len(Bedrooms), len(Bathrooms), len(Location), len(Size), len(Availability), len(Facing), len(Type), len(Price))

# Pad the shorter lists with None values to match the length of the longest list
for key in data.keys():
    data[key] += [None] * (max_length - len(data[key]))

# Create a Pandas DataFrame from the dictionary
final_data = pd.DataFrame.from_dict(data)

# Print the DataFrame
final_data
```

	Bedrooms	Place	Size	Price	Status	Bathrooms	Facing	Type
0	3	Poranki	1587	55.50	Ready to move	3.0	West	Apartment
1	2	Poranki	1450	48.00	Ready to move	2.0	0	Apartment
2	2	Kankipadu	1290	50.00	Under Construction	2.0	NorthEast	Independent House
3	2	Poranki	1200	40.00	Ready to move	2.0	0	Apartment
4	3	Gannavaram	1663	59.86	Ready to move	3.0	East	Apartment
...	...	...	...	...	...	...	...	...
1237	None	None	None	NaN	None	NaN	None	Residential Plot
1238	None	None	None	NaN	None	NaN	None	Residential Plot
1239	None	None	None	NaN	None	NaN	None	Residential Plot
1240	None	None	None	NaN	None	NaN	None	Residential Plot
1241	None	None	None	NaN	None	NaN	None	Residential Plot

1242 rows × 8 columns

```
[ ] #check the count of missing values in the final dataframe
    final_data.isnull().sum()

Bedrooms      21
Place         21
Size          21
Price         21
Status        21
Bathrooms     21
Facing        21
Type           0
dtype: int64

[ ] #As we are having missing values we can drop them

    final_data.dropna(inplace=True)

    final_data.info()

<class 'pandas.core.frame.DataFrame'>
Int64Index: 1221 entries, 0 to 1220
Data columns (total 8 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Bedrooms    1221 non-null   object
 1   Place       1221 non-null   object
 2   Size        1221 non-null   object
 3   Price       1221 non-null   float64
 4   Status      1221 non-null   object
 5   Bathrooms   1221 non-null   float64
 6   Facing      1221 non-null   object
 7   Type        1221 non-null   object
dtypes: float64(2), object(6)
memory usage: 85.9+ KB
```

Day – 3: Implementation of Regularization with House Pricing Case Study

Regularization is a technique used in Machine learning to prevent Overfitting and improve the generalization ability of the model.

Overfitting occurs when the model fits the training data too well and captures noise in the data, resulting in poor performance on new, unseen data.

Regularization helps to address this issue by adding a penalty term to the loss function that encourages the model to learn simpler and smoother patterns.

Ridge Regression is the regularized version of Linear Regression where we add a reguaralized term to the cost function

Cost Function $J(\theta) = MSE(\theta) + \alpha * 1/2 \sum_{i=1 \text{ to } n} \theta^2$

Implementation:

```
[ ] boston_df.corr() #checking  Correlation of features with Price
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	Price
CRIM	1.000000	-0.200469	0.406583	-0.055892	0.420972	-0.219247	0.352734	-0.379670	0.625505	0.582764	0.289946	-0.385064	0.455621	-0.388305
ZN	-0.200469	1.000000	-0.533828	-0.042697	-0.516604	0.311991	-0.569537	0.664408	-0.311948	-0.314563	-0.391679	0.175520	-0.412995	0.360445
INDUS	0.406583	-0.533828	1.000000	0.062938	0.763651	-0.391676	0.644779	-0.708027	0.595129	0.720760	0.383248	-0.356977	0.603800	-0.483725
CHAS	-0.055892	-0.042697	0.062938	1.000000	0.091203	0.091251	0.086518	-0.099176	-0.007368	-0.035587	-0.121515	0.048788	-0.053929	0.175260
NOX	0.420972	-0.516604	0.763651	0.091203	1.000000	-0.302188	0.731470	-0.769230	0.611441	0.668023	0.188933	-0.380051	0.590879	-0.427321
RM	-0.219247	0.311991	-0.391676	0.091251	-0.302188	1.000000	-0.240265	0.205246	-0.209847	-0.292048	-0.355501	0.128069	-0.613808	0.695360
AGE	0.352734	-0.569537	0.644779	0.086518	0.731470	-0.240265	1.000000	-0.747881	0.456022	0.506456	0.261515	-0.273534	0.602339	-0.376955
DIS	-0.379670	0.664408	-0.708027	-0.099176	-0.769230	0.205246	-0.747881	1.000000	-0.494588	-0.534432	-0.232471	0.291512	-0.496996	0.249929
RAD	0.625505	-0.311948	0.595129	-0.007368	0.611441	-0.209847	0.456022	-0.494588	1.000000	0.910228	0.464741	-0.444413	0.488676	-0.381626
TAX	0.582764	-0.314563	0.720760	-0.035587	0.668023	-0.292048	0.506456	-0.534432	0.910228	1.000000	0.460853	-0.441808	0.543993	-0.468536
PTRATIO	0.289946	-0.391679	0.383248	-0.121515	0.188933	-0.355501	0.261515	-0.232471	0.464741	0.460853	1.000000	-0.177383	0.374044	-0.507787
B	-0.385064	0.175520	-0.356977	0.048788	-0.380051	0.128069	-0.273534	0.291512	-0.444413	-0.441808	-0.177383	1.000000	-0.366087	0.333461
LSTAT	0.455621	-0.412995	0.603800	-0.053929	0.590879	-0.613808	0.602339	-0.496996	0.488676	0.543993	0.374044	-0.366087	1.000000	-0.737663
Price	-0.388305	0.360445	-0.483725	0.175260	-0.427321	0.695360	-0.376955	0.249929	-0.381626	-0.468536	-0.507787	0.333461	-0.737663	1.000000

```
[ ] #building the model
    lin_model = LinearRegression()
    lin_model.fit(x_train,y_train)
```

```
LinearRegression()
```

```
#check the coefficients
df = pd.DataFrame({'Features':X.columns,
                   'Coefficients':lin_model.coef_})
df
```

	Features	Coefficients
0	CRIM	-0.098542
1	ZN	0.060784
2	INDUS	0.059172
3	CHAS	2.439560
4	NOX	-21.469965
5	RM	2.795814
6	AGE	0.003575
7	DIS	-1.516272
8	RAD	0.307542
9	TAX	-0.011280
10	PTRATIO	-1.005466
11	B	0.006450
12	LSTAT	-0.568835

Day – 4: Implementation of Ridge Regression, LASSO Regression

Ridge Regression:

Ridge Regression is a regularized linear regression algorithm that helps mitigate overfitting by introducing a penalty term to the loss function. The penalty term is proportional to the sum of squared coefficients, which encourages the model to have smaller coefficient values.

By including the regularization term, ridge regression strikes a balance between model complexity and overfitting. It is particularly useful when dealing with datasets that have a large number of predictors or when multicollinearity is present.

Ridge regression can be solved using various optimization algorithms, such as closed-form solutions or gradient descent methods. The regularization parameter α is a hyperparameter that needs to be tuned to find the optimal trade-off between bias and variance.

Overall, ridge regression provides a regularized approach to linear regression, promoting stability and improving generalization performance by reducing the impact of multicollinearity and preventing excessive model complexity.

Implementation of Ridge Regression:

```
[ ] #Implement Ridge Regression
    from sklearn.linear_model import Ridge
```

```
[ ] #start building the model
    rr = Ridge(alpha = 0.01) #L2 Normalization term in the Ridge Regression is weighted by the
    #regularization parameter

    rr.fit(x_train,y_train)

    Ridge(alpha=0.01)
```

```
[ ] rr.coef_

    array([-0.83869061,  1.42802799,  0.40491125,  0.67948994, -2.52990688,
            1.93399057,  0.1007731 , -3.23562877,  2.70195867, -1.91611996,
           -2.15568443,  0.58229622, -4.13408504])
```

evaluating the training and test scores for Ridge Regression model:

```
[ ] #evaluate the training and test scores both for Linear and Ridge Regression models
    train_score =linear_model.score(x_train,y_train)
    test_score = linear_model.score(x_test,y_test)
    print(train_score,test_score)

    0.7103879080674731 0.7836295385076292
```

```
[ ] ridge_train_score = rr.score(x_train,y_train)
    ridge_test_score = rr.score(x_test,y_test)
    print(ridge_train_score,ridge_test_score)

    0.7103879045252288 0.7836323841970448
```

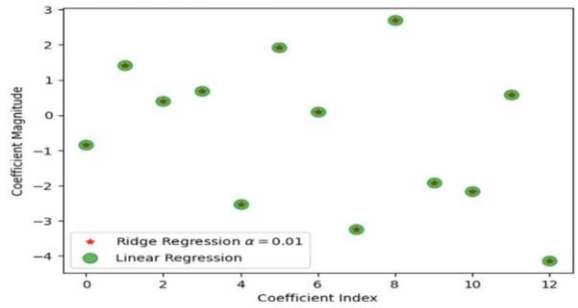
```
    #check the metrics
    from sklearn.metrics import mean_squared_error,r2_score
    #first check with Linear Regression
    pred_lin_reg = linear_model.predict(x_test)
    print(np.sqrt(mean_squared_error(y_test,pred_lin_reg)))
    print(r2_score(y_test,pred_lin_reg))

    4.4532374371981485
    0.7836295385076292
```

```
    #checking metrics the Ridge Regression
    pred_rr_reg =with  rr.predict(x_test)
    print(np.sqrt(mean_squared_error(y_test,pred_rr_reg)))
    print(r2_score(y_test,pred_rr_reg))

    4.453208152765743
    0.7836323841970448
```

Visualisation of Ridge Regression:

```
#let's check with the visualization
import matplotlib.pyplot as plt
plt.plot(rr.coef_,color="red",alpha=0.8,linestyle='none',
         marker='*',markersize=6,
         label=r"Ridge Regression $\alpha=0.01$") #alpha is thickness of markers
plt.plot(linear_model.coef_,alpha=0.6,linestyle='none',
         marker = 'o',markersize=10,
         color='green',label="Linear Regression")
plt.legend()
plt.xlabel("Coefficient Index",fontsize=10)
plt.ylabel("Coefficient Magnitude",fontsize=10)
```

```
Text(0, 0.5, 'Coefficient Magnitude')
```



LASSO Regression:

Lasso Regression, also known as L1 regularization, is a linear regression algorithm that introduces a penalty term proportional to the absolute values of the coefficients. This penalty term encourages sparsity in the model by forcing some coefficients to become exactly zero, effectively performing feature selection.

Lasso regression has a unique property compared to ridge regression: it can produce sparse models. By setting some coefficients to exactly zero, lasso regression performs feature selection by effectively eliminating less important predictors from the model.

The choice of the regularization parameter α in lasso regression is critical. It determines the amount of regularization applied and influences the sparsity of the resulting model. Techniques such as cross-validation or information criteria (e.g., AIC, BIC) can be used to find the optimal α value.

Lasso regression can be solved using optimization algorithms such as coordinate descent or least-angle regression (LARS). These algorithms iteratively update the coefficients to find the optimal solution.

In summary, lasso regression is a linear regression technique that introduces a regularization term to promote sparsity in the model. It performs feature selection by pushing some coefficients to zero, allowing for interpretable and more compact models.

Implementation of LASSO Regression:

```
[ ] #let's check the coefficients
    lasso.coef_
```

```
array([-0.        , -0.        , -0.        , -0.        , -0.        ,
       -0.        , -0.        , -0.        , -0.        , -0.        ,
       -0.        , -0.        , -0.        ,  0.        , -0.        ,
       -0.        , -0.        , -0.        , -0.        , -0.        ,
       -0.        , -0.        , -0.00425308, -0.00036339, -0.        ,
       -0.        , -0.        , -0.        , -0.        , -0.        ])
```

```
#let's check the coefficients which are used out of (30)
coeff_used = np.sum(lasso.coef_!=0)
coeff_used
```

```
2
```

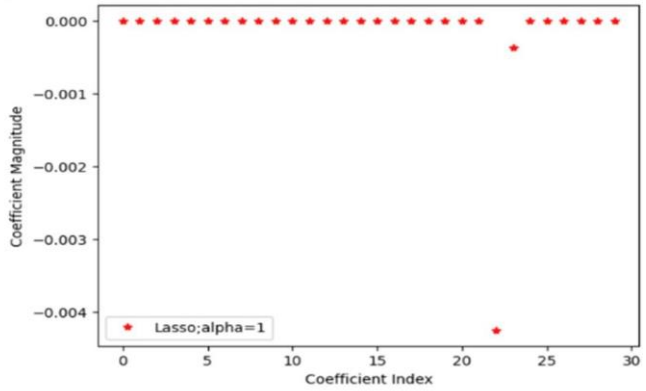
```
Lasso()
```

```
[ ] #let's check the training and test scores
    train_score = lasso.score(x_train,y_train)
    test_score = lasso.score(x_test,y_test)
    print(train_score,test_score)
```

```
0.549685114797433 0.6276868622116495
```

```
plt.plot(lasso.coef_,alpha=0.9,
         linestyle = "none",marker='*',
         color='red',markersize=6,
         label = "Lasso;alpha=1")
plt.legend()
plt.xlabel("Coefficient Index",fontsize=10)
plt.ylabel("Coefficient Magnitude",fontsize=10)
```

```
Text(0, 0.5, 'Coefficient Magnitude')
```



Day – 5: Logistic Regression

Logistic Regression is a classification algorithm. It is used to predict a binary outcome (1 / 0, Yes / No, True / False) given a set of independent variables one or many. To represent binary/categorical outcome, we use dummy variables. You can also think of logistic regression as a special case of linear regression when the outcome variable is categorical, where we are using log of odds as dependent variable.

The fundamental equation of generalized linear model is:

$g(E(y)) = \alpha + \beta x1 + \gamma x2$

Here, g() is the link function,

E(y) is the expectation of target variable and $\alpha + \beta x1 + \gamma x2$ is the linear predictor

α,β,γ to be predicted). The role of link function is to 'link' the expectation of y to linear predictor.

Once the coefficients are estimated, the predicted probability can be converted into a binary decision by applying a threshold. For example, if the predicted probability is above 0.5, the instance is classified as belonging to the positive class; otherwise, it is classified as belonging to the negative class.

Logistic regression can be extended to handle multiclass classification problems using techniques like one-vs-rest or softmax regression.

Logistic regression is widely used due to its simplicity, interpretability, and robustness. It is a linear model that can handle both categorical and continuous features. However, it assumes a linear relationship between the features and the log-odds of the outcome, which may not always be appropriate for complex data patterns.

In logistic regression, the logistic function (also called the sigmoid function) is used to transform the linear combination of the input features and their corresponding coefficients into a value between 0 and 1, representing the probability of belonging to the positive class. The logistic function is defined as:

$p = 1 / (1 + e^{(-z)})$

where p is the predicted probability, and z is the linear combination of the input features and their coefficients.

The linear combination of the input features and their coefficients is expressed as:

$z = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + ... + \beta_p x_p$

where $\beta_0, \beta_1, \beta_2, ..., \beta_p$ are the coefficients (also known as weights) associated with the features $x_1, x_2, ..., x_p$, respectively.

The coefficients $\beta_0, \beta_1, \beta_2, ..., \beta_p$ are estimated by maximizing the likelihood of the observed data. This is typically done using optimization algorithms like maximum likelihood estimation or gradient descent.

Logistic Regression Implementation:

```
#building the model
from sklearn.linear_model import LogisticRegression
```

```
logmodel = LogisticRegression()
```

```
#fitting the data to the model
logmodel.fit(x_train,y_train)
```

```
LogisticRegression()
```

```
#we can check the predictions by passing test data
predictions = logmodel.predict(x_test)
```

```
#metrics calculation
from sklearn.metrics import classification_report,confusion_matrix,accuracy_score
```

```
print("Accuracy Score is",accuracy_score(y_test,predictions))
```

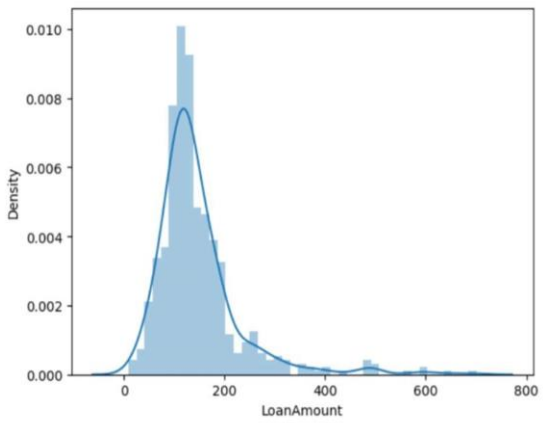
```
Accuracy Score is 0.7837837837837838
```

```
print("Classification report")
print(classification_report(y_test,predictions))
```

```
Classification report
              precision    recall  f1-score   support

           0       0.93      0.42      0.57        65
           1       0.76      0.98      0.86       120
```

```
import warnings
warnings.filterwarnings('ignore')
```

```
#Distribution of the amount of data -->distplot()
sn.distplot(data['LoanAmount'])
plt.show()
```




```
#building the model
from sklearn.linear_model import LogisticRegression
```

```
logmodel = LogisticRegression()
```

```
#fitting the data to the model
logmodel.fit(x_train,y_train)
```

```
LogisticRegression()
```

```
#we can check the predictions by passing test data
predictions = logmodel.predict(x_test)
```

```
#metrics calculation
from sklearn.metrics import classification_report,confusion_matrix,accuracy_score
```

```
print("Accuracy Score is",accuracy_score(y_test,predictions))
```

```
Accuracy Score is 0.7837837837837838
```


Day – 6: Introduction to Decision Tree

Decision Tree:

As the name suggests it is a classifier based on a series of logical decisions, which resembles a tree with the branches.

A decision tree is used for multi-dimensional analysis with multiple class.

It is characterized by fast execution time and ease in the implementation of the rules.

In decision trees, there are several key terminologies that describe the different elements and concepts within the tree. Here are some important terms related to decision trees:

1. Root Node: The topmost node in a decision tree, from which the tree starts branching out. It represents the entire dataset or a subset of it.
2. Internal Node: Nodes within the tree that have child nodes branching out from them. Internal nodes represent attribute or feature test conditions.
3. Leaf Node (or Terminal Node): The final nodes of a decision tree where no further splits occur. Each leaf node represents a class label or a predicted outcome.
4. Branch (or Edge): The connection between nodes in a decision tree. It represents the outcome of a feature test.
5. Splitting: The process of dividing the dataset into subgroups based on attribute conditions at internal nodes.
6. Feature (or Attribute): The input variable or characteristic used for making decisions at each node. Features can be categorical or continuous.
7. Splitting Criterion: The measure used to determine the attribute and condition to split the data at each internal node. Common splitting criteria include Gini impurity, information gain, or gain ratio.
8. Pruning: The process of reducing the size of a decision tree by removing unnecessary nodes or branches. Pruning helps prevent overfitting and improves the generalization ability of the model.
9. Parent Node: The node that branches out to one or more child nodes.
10. Child Node: The nodes that are connected to a parent node via branches.
11. Depth: The length of the longest path from the root node to a leaf node in a decision tree. It represents the number of attribute tests required to reach a leaf node.
12. Decision Path: The sequence of attribute tests taken from the root node to a specific leaf node.

Popular Decision Tree Algorithms:

There are several popular decision tree algorithms that are widely used in machine learning. Here are some of the most well-known decision tree algorithms:

1. ID3 (Iterative Dichotomiser 3): ID3 is one of the earliest decision tree algorithms developed by Ross Quinlan. It uses the concept of information gain to select the best attribute for splitting the data at each node. ID3 is primarily used for categorical data.
2. C4.5: C4.5 is an extension of the ID3 algorithm, also developed by Ross Quinlan. It handles both categorical and continuous data, and it uses the concept of information gain ratio as the splitting criterion. C4.5 can handle missing values and performs attribute selection based on information gain ratio.
3. CART (Classification and Regression Trees): CART is a versatile decision tree algorithm that can be used for both classification and regression tasks. It uses the Gini impurity (for classification) or mean squared error (for regression) as the splitting criterion. CART constructs binary decision trees, where each internal node has two child nodes.
4. Random Forest: Random Forest is an ensemble learning method that combines multiple decision trees to make predictions. It creates a set of decision trees, each trained on a random subset of features and a random subset of the training data. The final prediction is determined by aggregating the predictions of all the individual trees.

The main steps to build a decision tree involve the following:

1. Data Preparation: Gather and preprocess the data that will be used to build the decision tree. This includes handling missing values, encoding categorical variables, and splitting the data into a training set and a testing set.

2. Attribute Selection: Determine the best attribute to use for splitting the data at each internal node of the tree. Different algorithms use different criteria, such as information gain, Gini impurity, or gain ratio, to measure the quality of the splits.

3. Splitting: Once the attribute has been selected, split the data into subsets based on the attribute's values. Each subset corresponds to a branch or child node in the tree.

4. Recursion: Repeat the attribute selection and splitting process recursively for each subset or branch, creating child nodes and further splitting the data until a stopping condition is met. Stopping conditions may include reaching a maximum depth, having a minimum number of samples per leaf node, or achieving a certain level of purity.

5. Leaf Node Labeling: Assign a class label or a predicted outcome to each leaf node based on the majority class or the predicted value of the samples within that node.

6. Pruning (Optional): Prune the decision tree to reduce its complexity and prevent overfitting. Pruning involves removing unnecessary nodes or branches that do not significantly improve the tree's performance on unseen data.

7. Evaluation: Evaluate the performance of the decision tree using appropriate metrics such as accuracy, precision, recall, or mean squared error. This is typically done using the testing set that was separated in the data preparation step.

8. Prediction: Once the decision tree is built and evaluated, it can be used to make predictions on new, unseen data by traversing the tree from the root node to a leaf node based on the attribute conditions.

These steps outline the general process of building a decision tree. The specific details and variations may depend on the algorithm being used and the specific requirements of the problem at hand.

Week – 4 (05-06-2023 to 10-06-2023)

Day & Date	Learning Outcomes
Day – 1	Decision Tree Implementation.
Day – 2	Case study on Cross Validation Random and Forest Classifier.
Day – 3	Deployment using Streamlit.
Day – 4	Ensemble models.
Day – 5	XGBoost Classifier.
Day – 6	Naïve Bayes Classifier.

Day – 1: Decision Tree Implementation (Using Mushroom dataset) Importing libraries:

```
[ ]  #import the libraries
     import numpy as np
     import pandas as pd
     import matplotlib.pyplot as plt
     %matplotlib inline
     import matplotlib.pyplot as plt
```

```
[ ]  from sklearn.metrics import accuracy_score,classification_report
```

Reading mushrooms dataset.

```
#reading the daaset
data = pd.read_csv('mushrooms.csv')
data
```

	class	cap-shape	cap-surface	cap-color	bruises	odor	gill-attachment	gill-spacing	gill-size	gill-color	...	stalk-surface-below-ring	stalk-color-above-ring	stalk-color-below-ring	veil-type	veil-color	ring-number	ring-type	spore-print-color	population	habitat
0	p	x	s	n	t	p	f	c	n	k	...	s	w	w	p	w	o	p	k	s	u
1	e	x	s	y	t	a	f	c	b	k	...	s	w	w	p	w	o	p	n	n	g
2	e	b	s	w	t	l	f	c	b	n	...	s	w	w	p	w	o	p	n	n	m
3	p	x	y	w	t	p	f	c	n	n	...	s	w	w	p	w	o	p	k	s	u
4	e	x	s	g	f	n	f	w	b	k	...	s	w	w	p	w	o	e	n	a	g
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8124 entries, 0 to 8123
Data columns (total 23 columns):
 #   Column                    Non-Null Count  Dtype 
---  ------                    --------------  ----- 
 0   class                     8124 non-null   object
 1   cap-shape                 8124 non-null   object
 2   cap-surface               8124 non-null   object
 3   cap-color                 8124 non-null   object
 4   bruises                   8124 non-null   object
 5   odor                      8124 non-null   object
 6   gill-attachment           8124 non-null   object
 7   gill-spacing              8124 non-null   object
 8   gill-size                 8124 non-null   object
 9   gill-color                8124 non-null   object
 10  stalk-shape               8124 non-null   object
 11  stalk-root                8124 non-null   object
 12  stalk-surface-above-ring  8124 non-null   object
 13  stalk-surface-below-ring  8124 non-null   object
 14  stalk-color-above-ring    8124 non-null   object
 15  stalk-color-below-ring    8124 non-null   object
 16  veil-type                 8124 non-null   object
 17  veil-color                8124 non-null   object
 18  ring-number               8124 non-null   object
 19  ring-type                 8124 non-null   object
 20  spore-print-color         8124 non-null   object
 21  population                8124 non-null   object
 22  habitat                   8124 non-null   object
dtypes: object(23)
memory usage: 1.4+ MB
```

Performing Feature encoding:

- Converting categorical attributes into numerical attributes.
- Using native python approach instead of one hot encoding or get_dummies()

```
[ ] def normalize(col):
        uniques = list(set(col))
        d = {u:i for (i,u) in enumerate(uniques)}
        print(d)
        for i in range(len(col)):
            col[i] = d[col[i]]
        return d
    dicts=[]
```

```
for i in range(len(data.columns)):
    dicts.append(normalize(data.values[:,i]))
```

```
{'e': 0, 'p': 1}
{'b': 0, 'f': 1, 'k': 2, 'c': 3, 's': 4, 'x': 5}
{'y': 0, 'g': 1, 's': 2, 'f': 3}
{'b': 0, 'p': 1, 'u': 2, 'e': 3, 'n': 4, 'w': 5, 'c': 6, 'y': 7, 'g': 8, 'r': 9}
{'t': 0, 'f': 1}
{'p': 0, 'l': 1, 'n': 2, 'f': 3, 'a': 4, 'c': 5, 'y': 6, 'm': 7, 's': 8}
{'f': 0, 'a': 1}
{'c': 0, 'w': 1}
{'n': 0, 'b': 1}
{'b': 0, 'p': 1, 'u': 2, 'e': 3, 'n': 4, 'w': 5, 'k': 6, 'h': 7, 'o': 8, 'y': 9, 'g': 10, 'r': 11}
{'e': 0, 't': 1}
{'?': 0, 'b': 1, 'e': 2, 'c': 3, 'r': 4}
{'y': 0, 's': 1, 'f': 2, 'k': 3}
{'y': 0, 's': 1, 'f': 2, 'k': 3}
{'b': 0, 'p': 1, 'e': 2, 'n': 3, 'w': 4, 'c': 5, 'o': 6, 'y': 7, 'g': 8}
{'b': 0, 'p': 1, 'e': 2, 'n': 3, 'w': 4, 'c': 5, 'o': 6, 'y': 7, 'g': 8}
{'p': 0}
{'y': 0, 'n': 1, 'w': 2, 'o': 3}
{'n': 0, 't': 1, 'o': 2}
{'p': 0, 'l': 1, 'e': 2, 'n': 3, 'f': 4}
{'b': 0, 'u': 1, 'n': 2, 'w': 3, 'k': 4, 'h': 5, 'o': 6, 'y': 7, 'r': 8}
{'v': 0, 'n': 1, 'a': 2, 'c': 3, 'y': 4, 's': 5}
{'p': 0, 'u': 1, 'l': 2, 'w': 3, 'd': 4, 'g': 5, 'm': 6}
```

Dividing dataset into dependent and independent variables:

```
[ ]  X = data.values[:,1:]
```

```
[ ]  Y = data.values[:,0]
```

```
[ ]  #As we have target in object type we will transform into integer
     Y = Y.astype("int")
     Y.dtype
```

```
dtype('int32')
```

Splitting into training and testing:

```
[ ]  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)
```

Checking with Gini Index:

```
Gini_classifier = tree.DecisionTreeClassifier(criterion='gini',
                                               random_state=9,
                                               max_depth=4,
                                               min_samples_leaf=2)
```

```
[ ]  gini_classifier.fit(X_train,Y_train)
```

```
DecisionTreeClassifier(max_depth=4, min_samples_leaf=2, random_state=9)
```

Checking with Entropy/Information gain:

```
entropy_classifier = tree.DecisionTreeClassifier(criterion='entropy',
                                                  random_state=9,
                                                  max_depth=4,
                                                  min_samples_leaf=2)
```

```
[ ]  entropy_classifier.fit(X_train,Y_train)
```

```
DecisionTreeClassifier(criterion='entropy', max_depth=4, min_samples_leaf=2,
                       random_state=9)
```

Predicting test data:

```
[ ]  gini_predict = gini_classifier.predict(X_test)
     entropy_predict = entropy_classifier.predict(X_test)
```

Metrics Calculation:

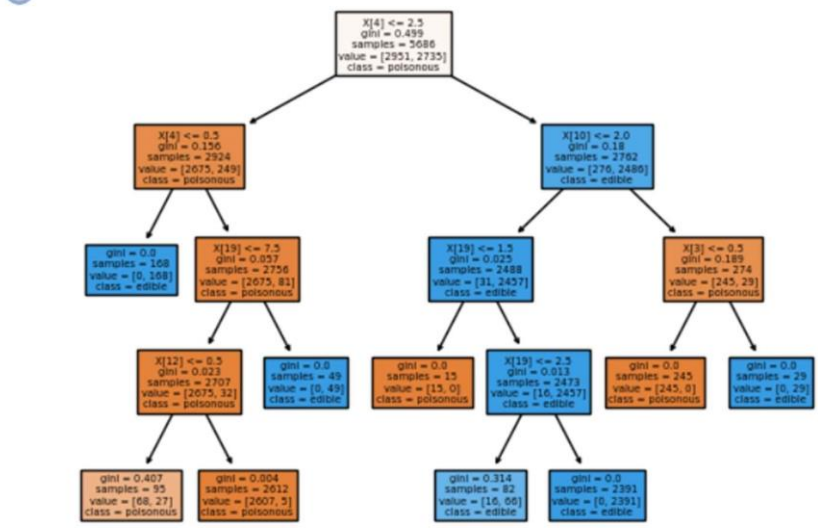
```
gini_accu = accuracy_score(gini_predict,Y_test)
print(gini_accu*100)
entro_accu = accuracy_score(entropy_predict,Y_test)
print(entro_accu*100)
```

```
99.01558654634947
99.34372436423298
```

Visualizing tree:

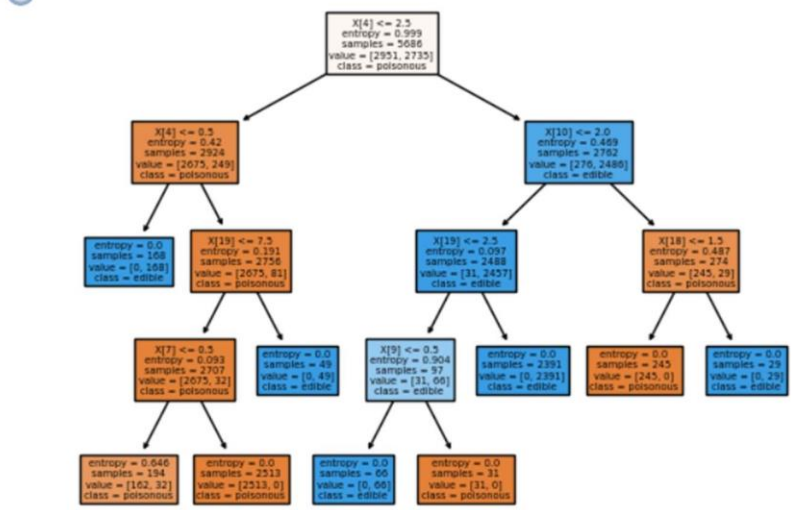
Using gini_classifier-

```
tree.plot_tree(gini_classifier,class_names=['poisonous','edible'],
               filled=True)
plt.show()
```



Using entropy_classifier-

```
tree.plot_tree(entropy_classifier,class_names=['poisonous','edible'],
               filled=True)
plt.show()
```



Day – 2: Case study on Cross Validation Random and Forest Classifier.

Train-Test Split and Cross-Validation are the techniques used in Machine Learning to assess the performance of a model.

Training Set: It learns patterns and relationships within the data during modelling

Test Set: Once the model is trained, it is evaluated on the test set to assess the performance. By evaluating the model's performance on unseen data, we can estimate how well it generalizes to new, unseen examples.

Cross Validation is a technique used to assess the performance of the model in a more robust and reliable manner compared to a single train-test split. The basic idea behind crossvalidation is to divide the dataset into multiple subsets or folds. The model is trained on a combination of folds and evaluated on the remaining fold, repeating this process several times. The most common type of cross-validation is k-fold cross-validation, where the data is divided into k equally sized folds. The training and evaluation process is performed k times, each time using a different fold as the evaluation set.

Cross-validation is especially useful when the dataset is small, and it provides a more comprehensive evaluation of the model by utilizing different subsets of data for training and testing.

Training: For each iteration, the model is trained on the combination of k-1 folds.
Evaluation: The model's performance is then evaluated on the fold that was left out. After all iterations are completed, the performance metrics (e.g., accuracy, error, etc.) are averaged to provide a more reliable estimate of the model's performance. This approach helps to reduce the variance and provides a better assessment of how the model is likely to perform on unseen data.

In summary, train-test split is a simpler approach where the data is divided into two parts, whereas cross-validation divides the data into multiple subsets (folds) and performs multiple training and evaluation cycles to provide a more robust performance estimate.

Cross Validation – Case Study (Red wine quality dataset) Importing

libraries:

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

Reading the red wine quality dataset:

```
[ ] #read the dataset
    df = pd.read_csv('winequality-red.csv')
    df
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	9.4	5
1	7.8	0.880	0.00	2.6	0.098	25.0	67.0	0.99680	3.20	0.68	9.8	5
2	7.8	0.760	0.04	2.3	0.092	15.0	54.0	0.99700	3.26	0.65	9.8	5
3	11.2	0.280	0.56	1.9	0.075	17.0	60.0	0.99800	3.16	0.58	9.8	6
4	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	9.4	5
...	...	...	...	...	...	...	...	...	...	...	...	...

```
[ ] df['quality'].value_counts()
```

```
5    681
6    638
7    199
4     53
8     18
3     10
Name: quality, dtype: int64
```

```
[ ] #We will map the quality values from 0-5
    #A mapping dictionary is created
    quality_mapping = {3:0,4:1,5:2,6:3,7:4,8:5}
    #you can use map_function with pandas (small home task)
    df.loc[:,'quality'] = df.quality.map(quality_mapping)
```

```
[ ] df['quality'].value_counts()
```

```
2    681
3    638
4    199
1     53
5     18
0     10
Name: quality, dtype: int64
```

```
[ ] #we use sample with frac=1 to shuffle the dataframe we reset the indices since they
    #change after shuffling the dataframe
    df = df.sample(frac=1).reset_index(drop=True)
    df

    #top 1000 rows into training
    df_train = df.head(1000)
    #bottom 599 values into testing
    df_test = df.tail(599)
```

```
[ ] #We will now build Decision Tree Classifier
    #import from scikit-learn
    from sklearn import tree
    from sklearn import metrics

    #initialize Decision Tree Classifier with a max_depth of 3
    clf = tree.DecisionTreeClassifier(max_depth=3)

    #choose the columns you want to train on the model
    clf.fit(df_train.iloc[:,:-1],df_train.quality)
```

```
DecisionTreeClassifier(max_depth=3)
```

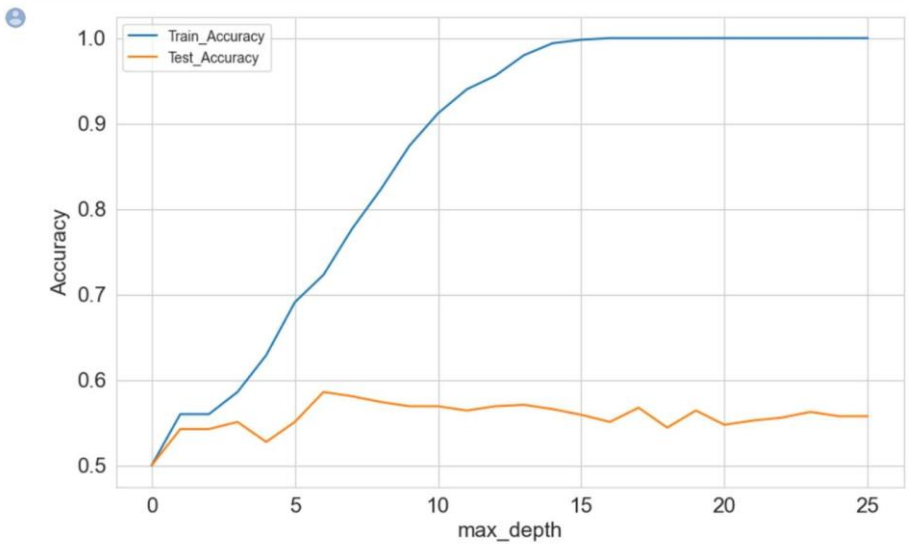
```
#generate predictions on the training set
train_predictions = clf.predict(df_train.iloc[:,:-1])

#generate predictions on the test set
test_predictions = clf.predict(df_test.iloc[:,:-1])

#Calculate the training accuracy of predictions on training dataset
train_accuracy = metrics.accuracy_score(df_train.quality,
                                        train_predictions)

#Calculate the test accuracy of predictions on test dataset
test_accuracy = metrics.accuracy_score(df_test.quality,
                                       test_predictions)
print(train_accuracy,test_accuracy)
```

```
0.586 0.5509181969949917
```

Calculating these accuracies with different values of max_depth and plotting them.

```
#plotting libraries
import seaborn as sn

#To make sure plot is displayed inside the notebook
%matplotlib inline

#to give the global size of label text on the plots

matplotlib.rc('xtick',labelsize=15)
matplotlib.rc('ytick',labelsize=15)

#initialize the lists to store the training and test accuracies
train_acc = [0.5]
test_acc = [0.5]
```

```
#iterate over a few depth values
for depth in range(1,26):
    #init the model
    clf = tree.DecisionTreeClassifier(max_depth=depth)

    #cols for training
    clf.fit(df_train.iloc[:,:-1],df_train.quality)
    #create training and testing predictions
    train_predictions = clf.predict(df_train.iloc[:,:-1])
    #generate predictions on the test set
    test_predictions = clf.predict(df_test.iloc[:,:-1])

    #Calculate the training accuracy of predictions on training dataset
    train_accuracy = metrics.accuracy_score(df_train.quality,
                                            train_predictions)

    #Calculate the test accuracy of predictions on test dataset
    test_accuracy = metrics.accuracy_score(df_test.quality,
                                           test_predictions)

    #append accuracies
    train_acc.append(train_accuracy)
    test_acc.append(test_accuracy)
#visualization
plt.figure(figsize=(10,6))
sn.set_style("whitegrid") #check the available styles
plt.plot(train_acc,label="Train_Accuracy")
plt.plot(test_acc,label="Test_Accuracy")
plt.legend()
plt.xlabel("max_depth",size=15)
plt.ylabel("Accuracy",size=15)
plt.show()
```



Implementing k-fold cross validation:

```
[ ] #Laod the wine_quality
    wine_data = pd.read_csv('winequality-red.csv')
    #separate features (X) and target variable (y)
    X = wine_data.drop('quality',axis=1)
    y = wine_data['quality']
```

```
[ ] model = tree.DecisionTreeClassifier()
    #Define the number of folds for Cross-Validation
    from sklearn.model_selection import cross_val_score,KFold,GridSearchCV
```

```
    k = 5
    #create a k-fold crossvalidation object
    kf = KFold(n_splits=k,shuffle=True,random_state=42)

    #define the range of maxdepth values to be explored
    param_grid = {'max_depth':[3,5,7,9]}

    #Perform GridSearchCV with k-fold cross-validation
    grid_search = GridSearchCV(estimator=model,
                               param_grid=param_grid,
                               cv=kf,n_jobs=-1,
                               verbose=4,scoring="accuracy")
    grid_search.fit(X,y)
    #print the best parameter and its correspondingscore
    print("Best Max_Depth:",grid_search.best_params_['max_depth'])
    print("Best Accuracy:",grid_search.best_score_)
```

```
Fitting 5 folds for each of 4 candidates, totalling 20 fits
Best Max_Depth: 9
Best Accuracy: 0.6010266457680251
```

```
[ ] #Now let's chck with Preprocessing -->StandardScaler
    from sklearn.preprocessing import StandardScaler
```

```
[ ] scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X)
```

```
[ ] model = tree.DecisionTreeClassifier()

    #define the number of folds
    k = 5
    #create a k-fold crossvalidation object
    kf = KFold(n_splits=k,shuffle=True,random_state=42)

    #Define the range of hyperparameters to explore
    param_grid = {'max_depth':[3,5,7,9],
                  'min_samples_leaf':[1,2,3]}
```

```
    #Perform GridSearchCV with k-fold cross-validation
    grid_search = GridSearchCV(estimator=model,
                               param_grid=param_grid,
                               cv=kf,n_jobs=-1,
                               verbose=4,scoring="accuracy")
    grid_search.fit(X_scaled,y)
    #print the best parameter and its correspondingscore
    print("Best Max_Depth:",grid_search.best_params_)
    print("Best Accuracy:",grid_search.best_score_)
```

```
Fitting 5 folds for each of 12 candidates, totalling 60 fits
Best Max_Depth: {'max_depth': 9, 'min_samples_leaf': 2}
Best Accuracy: 0.599149686520376
```

Random Forest belongs to a class of algorithms called ensemble methods.

Ensemble learning combines multiple models to make more accurate predictions than any individual model.

Random Forest combines multiple decision trees to form a forest, where each tree independently learns from a randomly selected subset of the training data.

- The "random" in Random Forest refers to the introduction of randomness in two key aspects:

Random Subset of Features: For each tree in the forest, only a random subset of features is considered at each node during the tree's construction. This randomness helps to introduce diversity among the trees and prevents them from relying too heavily on any specific feature.

Voting and Prediction: Once the Random Forest is trained, it can make predictions by aggregating the results from individual trees. For classification tasks, each tree "votes" for a class, and the class with the majority of votes becomes the predicted class. For regression tasks, the predictions from individual trees are averaged to obtain the final prediction.

- Random Forest makes predictions by combining the predictions of all the individual trees.
- In classification tasks, it uses a majority voting scheme, where each "tree" votes for a class label, and the class with the most votes becomes the final prediction.
- In Regression tasks, it takes the average of the predicted values from all the trees.

Random Forest Classifier – Case Study (Red wine quality dataset) Importing

required libraries:

```
import numpy as np
import pandas as pd
import matplotlib
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score,KFold,GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.feature_selection import SelectKBest,chi2
```

Reading the dataset:

```
#read the dataset
wine_data = pd.read_csv('winequality-red.csv')
wine_data
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	9.4	5
1	7.8	0.880	0.00	2.6	0.098	25.0	67.0	0.99680	3.20	0.68	9.8	5
2	7.8	0.760	0.04	2.3	0.092	15.0	54.0	0.99700	3.26	0.65	9.8	5
3	11.2	0.280	0.56	1.9	0.075	17.0	60.0	0.99800	3.16	0.58	9.8	6
4	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	9.4	5
...	...	...	...	...	...	...	...	...	...	...	...	...

Dividing dataset into dependent and independent variables:

```
[ ]  X = wine_data.drop('quality',axis=1)
     y = wine_data['quality']
```

Feature selection:

```
[ ]  selector = SelectKBest(chi2,k=5)
     X_selected = selector.fit_transform(X,y)
```

Performing feature scaling using standard scaler:

```
[ ]  scaler = StandardScaler()
     X_scaled = scaler.fit_transform(X_selected)
```

Creating a random forest classifier:

```
[ ]  #Create a Random Forest Classifier
     model = RandomForestClassifier()
```

Defining number of folds and performing cross-validation:

```
[ ]  #Define the number of folds
     k = 5
     #create a k-fold crossvalidation object
     kf = KFold(n_splits=k,shuffle=True,random_state=42)

     #Define the range of hyperparameters
     param_grid = {'n_estimators':[100,200,300],
                   'max_depth':[5,7,9],
                    'min_samples_split':[2,5,10],
                    'max_features':['sqrt','log2']
                   }
```

```
#Perform GridSearchCV with k-fold cross-validation
grid_search = GridSearchCV(estimator=model,
                           param_grid=param_grid,
                           cv=kf,n_jobs=-1,
                           verbose=4,scoring="accuracy")
grid_search.fit(X_scaled,y)
#print the best parameter and its correspondingscore
print("Best Max_Depth:",grid_search.best_params_)
print("Best Accuracy:",grid_search.best_score_)
```

```
Fitting 5 folds for each of 54 candidates, totalling 270 fits
Best Max_Depth: {'max_depth': 9, 'max_features': 'sqrt', 'min_samples_split': 2, 'n_estimators': 200}
Best Accuracy: 0.6423079937304075
```

Day – 3: Deployment using Streamlit:

Streamlit is an open-source Python library used for building interactive web applications and data dashboards. It simplifies the process of creating and sharing data-driven applications by allowing developers to write simple Python scripts. With Streamlit, developers can quickly create visualizations, input widgets, and interactive components, enabling users to explore and interact with data easily. It provides a straightforward and intuitive API, automatically updating the application as users interact with the interface. Streamlit is well-suited for prototyping, data exploration, and creating lightweight applications with minimal effort. It integrates seamlessly with popular data science libraries like Pandas, Matplotlib, and TensorFlow. With its user-friendly approach, Streamlit has gained popularity among data scientists and developers for quickly deploying and sharing interactive applications.

Git hub repository link: https://github.com/ManasaLakshmi1802/mlmodeldeploy.git

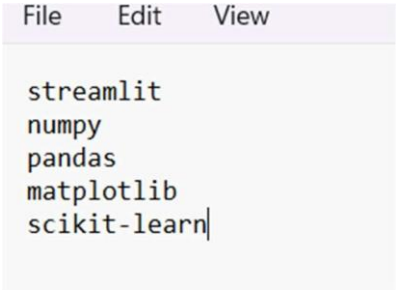
Streamlit app link: https://manasalakshmi1802-mlmodeldeploy-model-xmku8a.streamlit.app/

model.py

```
import streamlit as st
import numpy as np
import pandas as pd
from sklearn import metrics
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import LabelEncoder,OneHotEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import ConfusionMatrixDisplay,RocCurveDisplay,PrecisionRecallDisplay
from sklearn.metrics import precision_score,recall_score

#Start mapping the logic to built the interface
st.set_option('deprecation.showPyplotGlobalUse',False)
def main():
    st.title("Mushroom Classifier")
    st.sidebar.title("App Sidebar")
    st.sidebar.markdown("Let's start")

    @st.cache(persist = True)
    def load(): #Data Loading
        data=pd.read_csv("mushrooms.csv")
        label = LabelEncoder() #you do with OneHotEncoder
        for i in data.columns:
            data[i] = label.fit_transform(data[i])
        return data
    df = load()#call the function
    if st.sidebar.checkbox("Display data",False):
        st.subheader("Data is displayed")
        st.write(df)

    @st.cache(persist = True)
    def split(df):
        y = df['class']
        x = df.drop(columns=['class'])
        x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.3,
                                                         random_state = 42)
        return x_train,x_test,y_train,y_test
    x_train,x_test,y_train,y_test = split(df)
```

```
    def plot_metrics(metrics_list):
        if "Confusion Matrix" in metrics_list:
            st.subheader("Confusion Matrix")
            ConfusionMatrixDisplay.from_estimator(model,x_test,y_test)
            st.pyplot()
        if "ROC Curve" in metrics_list:
            st.subheader("ROC Curve")
            RocCurveDisplay.from_estimator(model,x_test,y_test)
            st.pyplot()
        if "Precision-Recall Curve" in metrics_list:
            st.subheader("Precision-Recall Curve")
            PrecisionRecallDisplay.from_estimator(model,x_test,y_test)
            st.pyplot()
    class_names = ['edible','poisonous']
    st.sidebar.subheader("Choose Classifier")
    classifier = st.sidebar.selectbox("Classifier",("Logistic Regression","Random Forest"))
    if classifier == "Logistic Regression":
        st.sidebar.subheader("Hyperparameters")
        C = st.sidebar.number_input("C (Regularization parameter)",0.01,10.0,
                                    step = 0.01,key="C_LR")
        max_iter = st.sidebar.slider("Maximum iterations",100,500,key="max_iter")
        metrics = st.sidebar.multiselect("What Metrics to plot?",
                                         ("Confusion Matrix","ROC Curve","Precision-Recall Curve"))
        if st.sidebar.button("Classify",key = "classify"):
            st.subheader("Logistic Regression Results")
            model = LogisticRegression(C=C,max_iter =max_iter)
            model.fit(x_train,y_train)
            accuracy = model.score(x_test,y_test)
            y_pred = model.predict(x_test)
            st.write("Accuracy: ",accuracy.round(2))
            st.write("Precision:",precision_score(y_test,y_pred,
                                                  labels = class_names).round(2))
            st.write("Recall:",recall_score(y_test,y_pred,
                                            labels = class_names).round(2))
            plot_metrics(metrics)


    if classifier == "Random Forest":
        st.sidebar.subheader("Hyperparameters")
        n_estimators = st.sidebar.number_input("The number of trees in the \
                                               Forest",100,5000,step=10,key="n_estimators")
        max_depth = st.sidebar.number_input("The maximum depth of the \
                                            tree" ,1,20,step = 1,key="max_depth")
        metrics = st.sidebar.multiselect("What Metrics to plot?",
                                         ("Confusion Matrix","ROC Curve","Precision-Recall Curve"))
        if st.sidebar.button("Classify",key = "classify"):
            st.subheader("Random Forest Results")
            model = RandomForestClassifier(n_estimators = n_estimators,
                                           max_depth = max_depth,n_jobs=-1)
            model.fit(x_train,y_train)
            accuracy = model.score(x_test,y_test)
            y_pred = model.predict(x_test)
            st.write("Accuracy: ",accuracy.round(2))
            st.write("Precision:",precision_score(y_test,y_pred,
                                                  labels = class_names).round(2))
            st.write("Recall:",recall_score(y_test,y_pred,
                                            labels = class_names).round(2))
            plot_metrics(metrics)


if __name__ == "__main__":
    main()
```

requirements.txt:



Day – 4: Ensemble Models

Bagging, also known as bootstrap aggregation, is the ensemble learning method that is commonly used to reduce variance within a noisy dataset. In bagging, a random sample of data in a training set is selected with replacement—meaning that the individual data points can be chosen more than once. After several data samples are generated, these weak models are then trained independently, and depending on the type of task—regression or classification, for example—the average or majority of those predictions yield a more accurate estimate.

As a note, the random forest algorithm is considered an extension of the bagging method, using both bagging and feature randomness to create an uncorrelated forest of decision trees.

Ensemble learning gives credence to the idea of the “wisdom of crowds,” which suggests that the decision-making of a larger group of people is typically better than that of an individual expert. Similarly, ensemble learning refers to a group (or ensemble) of base learners, or models, which work collectively to achieve a better final prediction. A single model, also known as a base or weak learner, may not perform well individually due to high variance or high bias. However, when weak learners are aggregated, they can form a strong learner, as their combination reduces bias or variance, yielding better model performance.

Steps:

1. Consider there are 'n' observations and 'm' features in the training set. You need to select a random sample from the training dataset without replacement.
2. A subset of m features is chosen randomly to create a model using sample observations.
3. The feature offering the best split out of the lot is used to split the nodes. The tree is grown, so you have the best root nodes.
4. The above steps are repeated n times, it aggregates the output of individual decision trees to give the best solution.

Evaluating a base classifier to visualize the importance of bagging, we first need to see how a base classifier performs on a dataset.

```
[ ] #import the necessary libraries
    from sklearn import datasets
    from sklearn.model_selection import train_test_split
    from sklearn.metrics import accuracy_score,classification_report,ConfusionMatrixDisplay
```

```
[ ] #base classifier -->Decision Tree Classifier
    from sklearn.tree import DecisionTreeClassifier
```

```
[ ] #We are going to use the sklearn's wine dataset to classify different types of wine using
    #the decision tree model
    data = datasets.load_wine()
    #print(data)
    data = datasets.load_wine(as_frame=True)
    #print(data)
    data.keys()
    #print(data['DESCR'])
    print(data['target_names'])
```

```
['class_0' 'class_1' 'class_2']
```

```
[ ] X = data.data
    y = data.target
```

```
[ ] #Check with the Feature Scaling part
```

```
[ ] #Split the dataset into training and testing
    x_train,x_test,y_train,y_test = train_test_split(X,y,test_size=0.25,
                                                     random_state=22)
    #load the model and fit it with the training data
    decision_tree = DecisionTreeClassifier(random_state=22)
    decision_tree.fit(x_train,y_train)
```

```
DecisionTreeClassifier(random_state=22)
```

```
[ ] #Get the predictions from the trained model
    y_pred = decision_tree.predict(x_test)
```

```
[ ] train_accuracy = accuracy_score(y_train,y_pred = decision_tree.predict(x_train))
    test_accuracy = accuracy_score(y_test,y_pred=y_pred)
    print("Training Data Accuracy",train_accuracy)
    print("Test data Accuracy",test_accuracy)
```

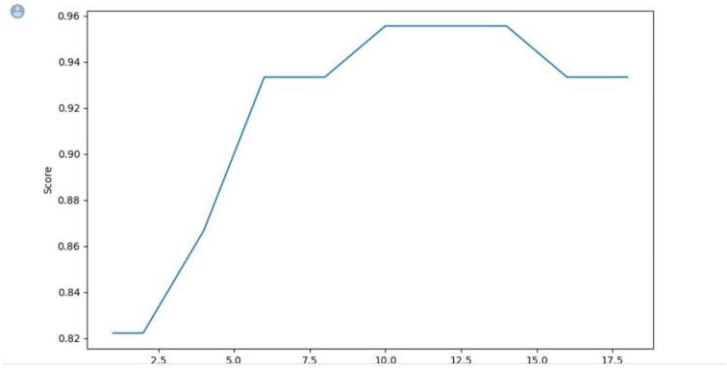
```
Training Data Accuracy 1.0
Test data Accuracy 0.8222222222222222
```

So, we get an 82% accuracy on the test dataset with the available parameters. For Bagging, we need to set the hyperparameters n_estimators . It is the number of base classifiers that the model will aggregate together.

```
#import the library
from sklearn.ensemble import BaggingClassifier
#choose a list of n_estimators
estimator_range = [1,2,4,6,8,10,12,14,16,18]

models =[]
scores =[]

for n_estimators in estimator_range:
    #create the bagging classifier
    classifier = BaggingClassifier(decision_tree,n_estimators=n_estimators,random_state=22)
    #fit the model
    classifier.fit(x_train,y_train)
    #appnd the model and score to their respective list
    models.append(classifier)
    scores.append(accuracy_score(y_true = y_test,y_pred = classifier.predict(x_test)))
print(models)
print(scores)
```

```
[BaggingClassifier(base_estimator=DecisionTreeClassifier(random_state=22),
                   n_estimators=1, random_state=22), BaggingClassifier(base_estimator=DecisionTreeClassifier(random_state=22),
                   n_estimators=2, random_state=22), BaggingClassifier(base_estimator=DecisionTreeClassifier(random_state=22),
                   n_estimators=4, random_state=22), BaggingClassifier(base_estimator=DecisionTreeClassifier(random_state=22),
                   n_estimators=6, random_state=22), BaggingClassifier(base_estimator=DecisionTreeClassifier(random_state=22),
                   n_estimators=8, random_state=22), BaggingClassifier(base_estimator=DecisionTreeClassifier(random_state=22),
                   random_state=22), BaggingClassifier(base_estimator=DecisionTreeClassifier(random_state=22),
                   n_estimators=12, random_state=22), BaggingClassifier(base_estimator=DecisionTreeClassifier(random_state=22),
                   n_estimators=14, random_state=22), BaggingClassifier(base_estimator=DecisionTreeClassifier(random_state=22),
                   n_estimators=16, random_state=22), BaggingClassifier(base_estimator=DecisionTreeClassifier(random_state=22),
                   n_estimators=18, random_state=22)]
[0.8222222222222222, 0.8222222222222222, 0.8666666666666667, 0.9333333333333333, 0.9333333333333333, 0.9555555555555556, 0.9555555555555556, 0.9555555555555556,
```

```
import matplotlib.pyplot as plt
#generate the plot of scores against several estimators

plt.figure(figsize=(10,6))
plt.plot(estimator_range,scores)
plt.xlabel("Number of Estimators")
plt.ylabel("Score")
plt.show()
```



Day – 5: XGBoost Classifier

Boosting (originally called as Hypothesis boosting) refers to any ensemble method that can combine several weak learners to a strong learner. The general idea is of most of the boosting methods is to train predictors sequentially and correct from its predecessors.

It is a powerful and widely-used ml algorithm known for its efficiency and performance in various tasks. It is based on Gradient Boosting Framework, which combines the predictions of multiple weak prediction models, typical decision trees, to create a stronger and more accurate final model.

1. Boosting: It is an ensemble learning technique that combines multiple weak models to create a strong model. It builds the weak models sequentially, where each new model is trained to correct the mistakes made by the previous models.
2. Gradient Boosting: It is a special type of boosting that minimizes a loss function by adding new models in a gradient descent manner. The key idea is to fit each new model to the residual errors (difference b/w actual and predicted)
3. XGBoost Algo: It is an optimized implementation of gradient boosting that introduces several enhancements to improve its performance and accuracy. These Key aspects are:

a. Regularization: It is used to control overfitting and improve the generalization ability of the model. It includes both L1(LASSO) & L2(RIDGE).

b. Customized Loss Function: Allows the use of customized loss functions to fit specific problem domains.

c. Feature Importance: Provides a feature importance score that indicates the relevant importance of each feature in a dataset.

d. Parallelization & Optimization: It is designed to be highly efficient & scalable. It includes parallelization techniques to speed up training, and it optimizes the computational resources.

Case Study – hotel_bookings dataset:

```
[ ] #import the packages
    import numpy as np
    import pandas as pd
    import matplotlib.pyplot as plt
    import seaborn as sns
```

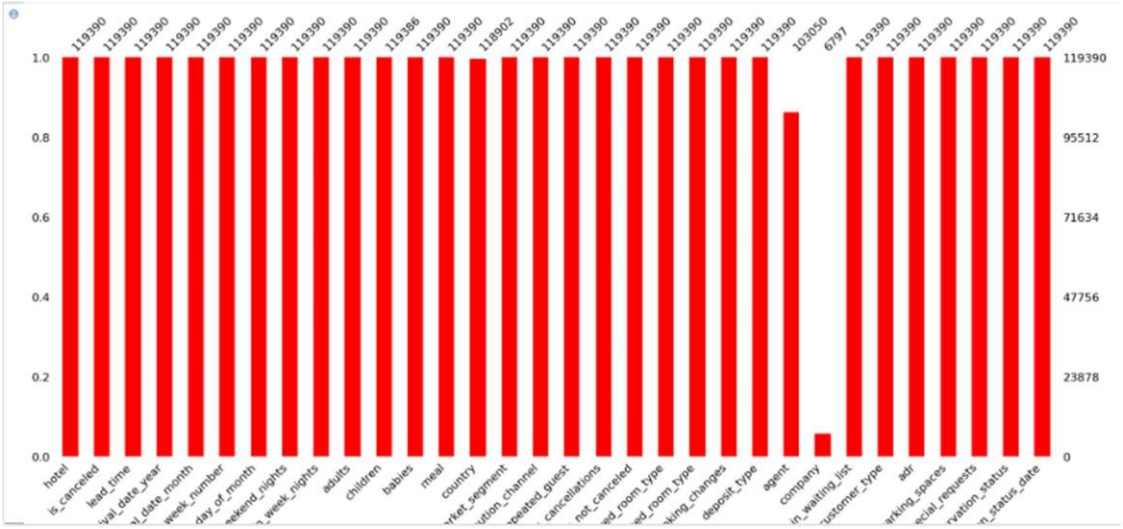
```
[ ] #read the data
    data = pd.read_csv('hotel_bookings.csv')
    data
```

	hotel	is_canceled	lead_time	arrival_date_year	arrival_date_month	arrival_date_week_number	arrival_date_day_of_month	stays_in_weekend_nights	stays_in_week_nights	adults	...
0	Resort Hotel	0	342	2015	July	27	1	0	0	2	...
1	Resort Hotel	0	737	2015	July	27	1	0	0	2	...
2	Resort Hotel	0	7	2015	July	27	1	0	1	1	...
3	Resort Hotel	0	13	2015	July	27	1	0	1	1	...
4	Resort Hotel	0	14	2015	July	27	1	0	2	2	...
...	...	...	...	...	...	...	...	...	...	...	...

Visualizing missing values:

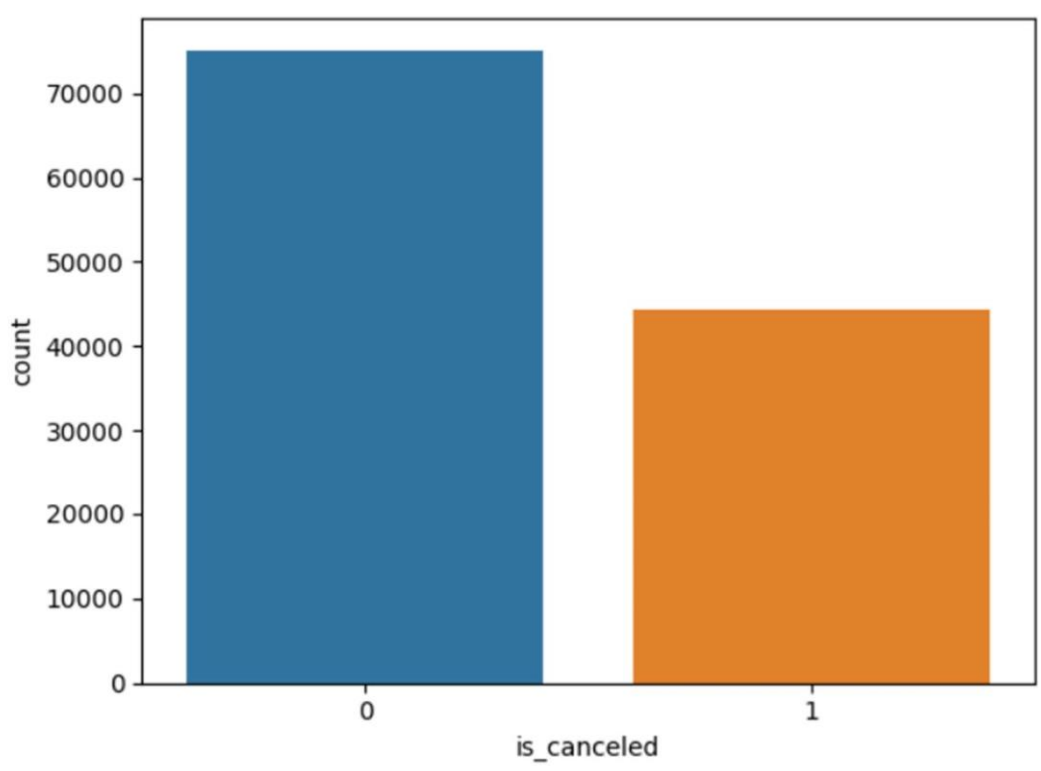
```
#visualize the missingvalues -->missingno
import missingno as ms
```

```
[ ] ms.bar(data,color='red')
    plt.show()
```




```
[ ]  data.drop(['company','reservation_status_date'],axis=1,inplace=True)
```

```
sns.countplot(x=data['is_canceled'])
plt.show()
```




```
[ ]  # creating a new dataframe that contains the original fields of data with th categorical variables replaced
     new_data = pd.get_dummies(data, drop_first=True)
```

```
[ ]  x = new_data.drop('is_canceled', axis=1)
     y = new_data['is_canceled']
```

```
[ ]  from sklearn.model_selection import train_test_split

     x_train, x_test, y_train, y_test = train_test_split(x, y, stratify=y, random_state=42)
```

```
[ ]  print(x_train.shape)
     print(x_test.shape)
```

```
(89542, 585)
(29848, 585)
```

Common Parameters for fitting XGBoost x -> input feature y -> target varable eval_set -> Optional argument that takes a list of (x,y) pairs, used to monitor model performance
eval_metric -> Specifies the evaluation metric to be used for model monitoring (accuracy, log loss, AUC, etc.)
early_stopping_rounds -> Allows for early stopping based on the evaluation
metrics params -> Dictionary containing various XGBoost Parameters
num_boost_round --> No of boosting rounds(trees) to build during training.

```
[ ]  from xgboost import XGBClassifier
```

```
[ ]  clf_xgb = XGBClassifier(objective='binary:logistic', seed=42)
     clf_xgb.fit(x_train, y_train,
                 verbose=True,
                 early_stopping_rounds=10,
                 eval_metric = 'auc',
                 eval_set=[(x_test,y_test)])
```

```
/usr/local/lib/python3.10/dist-packages/xgboost/sklearn.py:835: UserWarning: `eval_metric` in `f
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/xgboost/sklearn.py:835: UserWarning: `early_stopping_rou
  warnings.warn(
[0]     validation_0-auc:1.00000
[1]     validation_0-auc:1.00000
[2]     validation_0-auc:1.00000
[3]     validation_0-auc:1.00000
[4]     validation_0-auc:1.00000
[5]     validation_0-auc:1.00000
[6]     validation_0-auc:1.00000
[7]     validation_0-auc:1.00000
[8]     validation_0-auc:1.00000
[9]     validation_0-auc:1.00000
[10]    validation_0-auc:1.00000
```

```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, gpu_id=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=None, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=None, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              n_estimators=100, n_jobs=None, num_parallel_tree=None
```

```
[ ] from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
```

```
[ ] prediction_tr = clf_xgb.predict(x_train)
    print('Training Score :', accuracy_score(y_train, prediction_tr))

    prediction_ts = clf_xgb.predict(x_test)
    print('Testing Score :', accuracy_score(y_test, prediction_ts))
```

```
Training Score : 1.0
Testing Score : 1.0
```

```
[ ] predictions = clf_xgb.predict(x_test)

    print(classification_report(predictions, y_test))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     18792
           1       1.00      1.00      1.00     11056

    accuracy                           1.00     29848
   macro avg       1.00      1.00      1.00     29848
weighted avg       1.00      1.00      1.00     29848
```

```
[ ] print(confusion_matrix(predictions, y_test))
```

```
[[18792     0]
 [    0 11056]]
```

Day – 6: Naïve Bayes Classifier

Bayes Theorem:
It is a fundamental concept in Propability that is widely used in ML and Data Science. It provides a way to update our beliefs about the probality of an event occuring, given new evidence or information.

$$P(A|B) = P(B|A) * P(A) / P(B)$$

P(A|B) -> Event A occuring given that event B has occured.
P(B|A) --> Event B occuring that event A has occurred
P(A) --> Probability of A
P(B) --> Propability of B

Ex: Disease to occur in 1% (0.01) of population, diagnostic test that is 90% (0.9) accurate, if the person doesnt have disease there is only 10% (0.1) chance.

P(Disease | Test Positive) = P(Test Positive | Disease) * P(Disease) / P(Test Positive)
P(Disease) = 0.01
P(Test Positive | Disease) = 0.9
P(Test Positive) = P(Test Positive | Disease) * P(Disease) + P(Test Positive |No Disease) * P(No Disease)
P(Test Positive) = 0.9 * 0.01 + 0.1 * 0.99 = 0.108
P(Disease | Test Positive) = 0.9 * 0.01 / 0.108 = 0.0833 = 8.33 %
Only 8.33% chance that person actually has the disease.

Case Study using Spam-Ham dataset:

```
[ ] import pandas as pd
```

```
[ ] data = pd.read_csv('spamham.csv', encoding='latin-1')
```

```
data
```

	v1	v2	Unnamed: 2	Unnamed: 3	Unnamed: 4
0	ham	Go until jurong point, crazy.. Available only ...	NaN	NaN	NaN
1	ham	Ok lar... Joking wif u oni...	NaN	NaN	NaN
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...	NaN	NaN	NaN
3	ham	U dun say so early hor... U c already then say...	NaN	NaN	NaN
4	ham	Nah I don't think he goes to usf, he lives aro...	NaN	NaN	NaN
...	...	...	...	...	...

```
[ ] data.drop(columns=['Unnamed: 2', 'Unnamed: 3', 'Unnamed: 4'], axis=1, inplace=True)
```

Multinomial - Discrete features (word frequencies, document word counts, etc.), ignores the order or words in the text.
Binomial - Binary features (word presence or absence), ignores the frequency of words in the text.
CountVectorizer -> Considers only the count of terms in the document. TF-IDF --> Consider the count of terms and significance.

```
[ ] from sklearn.feature_extraction.text import CountVectorizer
```

```
[ ] import numpy as np
```

```
[ ] x = np.array(data['v2'])
    y = np.array(data['v1'])
```

```
[ ] cv = CountVectorizer()
```

```
[ ] X = cv.fit_transform(x)
```

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

```
[ ] x_train, x_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state=42)
```

```
[ ] from sklearn.naive_bayes import MultinomialNB
```

```
[ ] clf = MultinomialNB().fit(x_train, y_train)
```

```
[ ] clf.score(x_train, y_train)
```

```
0.9944962909787031
```

```
[ ] clf.score(x_test, y_test)
```

```
0.9763101220387652
```

```
[ ]  data = input('Enter a message : ')

     user_df = cv.transform([data]).toarray()

     clf.predict(user_df)[0]

     Enter a message : You need to submit the work on time
     'ham'
```

NLP:

It is a branch of AI that focuses on the interaction between computers and human language.

1. Text Preprocessing
2. Language Understanding
3. Sentiment Analysis
4. Information Extraction
5. Text Summarization

```
[ ] !pip install nltk
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: nltk in /usr/local/lib/python3.10/dist-packages (3.8.1)
Requirement already satisfied: click in /usr/local/lib/python3.10/dist-packages (from nltk) (8.1.3)
Requirement already satisfied: joblib in /usr/local/lib/python3.10/dist-packages (from nltk) (1.2.0)
Requirement already satisfied: regex>=2021.8.3 in /usr/local/lib/python3.10/dist-packages (from nltk) (2022.10.31)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from nltk) (4.65.0)
```

```
[ ] # Text Tokenization -> breaking down words into pharses

    import nltk
    nltk.download('punkt') # punkt is a tokenizer

    text ='Hey, my name is Rahul. Im here to teach you ML'

    tokens = nltk.word_tokenize(text) # Word tokenization
    sentences = nltk.sent_tokenize(text) # Sentence Tokenization

    print(tokens)
    print(sentences)
```

```
['Hey', ',', 'my', 'name', 'is', 'Rahul', '.', 'Im', 'here', 'to', 'teach', 'you', 'ML']
['Hey, my name is Rahul.', 'Im here to teach you ML']
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

```
# Lemmatization & Stemming --> Techniques used to reduce word to there root or base form

from nltk.stem import WordNetLemmatizer, PorterStemmer

nltk.download('wordnet')
nltk.download('averaged_perceptron_tagger')
nltk.download('punkt')

lemmatizer = WordNetLemmatizer()
stemmer = PorterStemmer()

word = 'drinking'

lemma = lemmatizer.lemmatize(word, pos='v')
stem = stemmer.stem(word)


print(word)
print(lemma)
print(stem)
```

```
drinking
drink
drink
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

Week – 5: (24-05-2023 to 27-05-2023)

Day & Date	Learning Outcomes
Day – 1	Recommendation System
Day – 2	Clustering Techniques
Day – 3	Deployment using AWS

Day – 4	Project Allocation and Explanation

Day 1: Over view on Recommendation System

A recommendation system is a subclass of information filtering system that seeks to predict the rating or the preference a user might give to an item. In simple words, it is an algorithm that suggests relevant items to users.

There are two main types of recommendation systems:

- Collaborative filtering systems recommend items based on the ratings or preferences of other users. For example, if you like the movie "The Shawshank Redemption", a collaborative filtering system might recommend other movies that have been rated highly by people who also liked "The Shawshank Redemption".
- Content-based filtering systems recommend items based on the content of the items themselves. For example, if you have rated movies highly that are about prison breaks, a content-based filtering system might recommend other movies about prison breaks.

Recommendation systems are used in a wide variety of applications, including:

- E-commerce: Recommending products to customers based on their past purchases, browsing history, and other factors.
- Online streaming: Recommending movies, TV shows, and music to users based on their watch history and ratings.
- Social media: Recommending friends, groups, and content to users based on their interests and relationships.
- Education: Recommending learning resources to students based on their past performance, interests, and goals.

Recommendation systems are a powerful tool for helping users discover new and relevant items. They are becoming increasingly important as the amount of available information continues to grow.

Here are some of the benefits of using recommendation systems:

- Increased customer satisfaction: Recommendation systems can help customers find the products and services they are looking for more easily, which can lead to increased satisfaction.

- Increased sales: Recommendation systems can help businesses increase sales by suggesting products and services that customers are likely to be interested in.
- Improved user experience: Recommendation systems can help improve the user experience by making it easier for users to find the information and content they are looking for.

Recommendation systems are a complex and evolving field. There are many different algorithms and techniques that can be used to build recommendation systems, and the best approach will vary depending on the specific application. However, recommendation systems are a powerful tool that can be used to improve the user experience and increase sales for businesses.

It is a type of information filtering system that suggests relevant items or content to users based on their preferences, interests, or past behavious. Its main goal is to assist users in discovering new items, such as products, movies, music, or articles, that they might find interesting or useful.

Types:

1. Content-Based Filtering --> Tis approach recommends items to users based on their previous preferences and the similarity of item characteristics.
2. Collaborative Filtering --> This technique recommends items based on the behaviour and preferences of similar users.

a. User-Based -> It recommends items to the user based on the opinion of other users who have similar preferences or behavior

b. Item-Based -> It recommends items to a user based on the similarity between items

3. Hybrid Approaches --> Combines multiple techniques, such as content-based and collaborative, to provide more accurate and diverse recommendations
4. Knowledge-Based --> It utilizes domain-specific knowledge or expert systems to generate recommendations

Cosine Similarity:

It is a metric used to determine the similarity b/w 2 vectors or documents. To calculate Cosine Similarity, steps are:

1. Compute Dot Product -> Sum of the elements-wise multiplication of corresponding dimensions
2. Calculating Magnitude -> Sqrt of the sum of squares of the vectors elements (scaler)
3. Cosine Similarity -> Dividing the dot product by the product of the magnitudes of the vectors

Parameters:

1. Vectors --> It represent the object / document to be compared Ex: Document A = [1,2,3]
2. Dot Product --> Calculates the sum of element wise multiplication Ex: (12) + (23) + (3*1) = 11
3. Magnitude --> Represents the length of vector Ex: Magnitude of A = sqrt((1^2) + (2^2) + (3^2)) = 3.74
4. Cosine Similarity --> Measure the angle b/w 2 vectors Cosine Similarity = Dot Product / (Magnitude A * Magnitude B)
5. Range & Interpretation --> Cosine Similarity range from -1 to 1 -1 --> Opposite 0 --> No corelation 1 --> Identical

Working on Cosine Similarity:

```
[ ] from sklearn.metrics.pairwise import cosine_similarity

[ ] vector1 = [0.5, 0.2, 0.1]

    vector2 = [0.3, 0.6, 0.2]

[ ] similarity = cosine_similarity([vector1], [vector2])

[ ] similarity

    array([[0.75637877]])
```

Example Use Case

Importing libraries

```
import nltk
from nltk.corpus import stopwords # is, are, the, to, a
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
from nltk.corpus import wordnet

from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
```

Download the necessary NLTK resources

```
nltk.download('punkt') nltk.download('stopwords') nltk.download('wordnet')
```

Sample data - documents for recommendations

```
documents = [
    "I love to watch movies",
    "I like action movies",
    "I enjoy romantic movies",
    "I hate horror movies",
    "I love to read books",
    "I like adventure books",
    "I enjoy romantic novels",
    "I prefer comedies over dramas",
    "I enjoy science fiction novels",
    "I like to travel and explore new places",
    "I like to play sports",
    "I am a food lover",
    "I enjoy spending time outdoors",
    "I love to listen to music",
    "I am interested in history",
    "I enjoy cooking and trying new recipes",
    "I like to go hiking",
    "I enjoy watching TV shows",
    "I love spending time with my family",
    "I am passionate about photography",
    "I like to go swimming",
    "I enjoy going to concerts",
    "I love to go shopping",
    "I am a pet lover",
    "I enjoy solving puzzles",
    "I like to go cycling",
    "I am a nature enthusiast",
```

Preprocess the documents

```
[ ] def preprocess_documents(documents):
        preprocessed_docs = []
        lemmatizer = WordNetLemmatizer()
        stop_words = set(stopwords.words('english'))

        for doc in documents:
            tokens = word_tokenize(doc.lower())
            filtered_tokens = [lemmatizer.lemmatize(token) for token in tokens if token.isalpha() and token not in stop_words]
            preprocessed_docs.append(' '.join(filtered_tokens))

        return preprocessed_docs
```

```
[ ] preprocessed_documents = preprocess_documents(documents)
```

```
[ ] preprocessed_documents
```

```
['love watch movie',
 'like action movie',
 'enjoy romantic movie',
 'hate horror movie',
 'love read book',
 'like adventure book',
 'enjoy romantic novel',
 'prefer comedy drama',
 'enjoy science fiction novel',
 'like travel explore new place',
 'like play sport',
 'food lover',
 'enjoy spending time outdoors',
 'love listen music',
 'interested history',
```

Build TF-IDF matrix

```
[ ] # build TF-IDF matrix

    vectorizer = TfidfVectorizer()
    tfidf_matrix = vectorizer.fit_transform(preprocessed_documents)
```

```
print(tfidf_matrix)
```

```
  (0, 47)       0.5436550340535424
  (0, 90)       0.7315612806215285
  (0, 44)       0.41140891658255413
  (1, 0)        0.7388151705619556
  (1, 41)       0.39076968421099
  (1, 47)       0.5490457154455826
  (2, 69)       0.7109078012810341
  (2, 17)       0.39240484084492555
  (2, 47)       0.5836339083357259
  (3, 36)       0.625946688907279
  (3, 33)       0.625946688907279
  (3, 47)       0.4651682333221266
  (4, 6)        0.6194129022023412
  (4, 66)       0.6842814718383395
  (4, 44)       0.3848201188659822
  (5, 1)        0.6902068145321107
  (5, 6)        0.6247765337860405
  (5, 41)       0.3650600443813826
  (6, 53)       0.6587108273710274
  (6, 69)       0.6587108273710274
  (6, 17)       0.3635933055054141
  (7, 15)       0.5773502691896257
```

Function to get recommendations

```python
# Function to get recommendations
def get_recommendations(query, documents, tfidf_matrix, top_n=3):
    # Preprocess query
    preprocessed_query = preprocess_documents([query])

    # Transform query to TF-IDF vector
    query_vector = vectorizer.transform(preprocessed_query)

    # Compute cosine similarity between query vector and document vectors
    similarities = cosine_similarity(query_vector, tfidf_matrix)

    # Get top n recommendations
    indices = similarities.argsort()[0][-top_n:][::-1]
    recommendations = [documents[idx] for idx in indices]

    return recommendations
```

Test the recommendation system

```python
query = input('Enter Query to Recommend : ')
recommendations = get_recommendations(query, documents, tfidf_matrix)

print('Recommendation --------')
for rec in recommendations:
  print(rec)
```

```
Enter Query to Recommend : i like books
Recommendation --------
I like adventure books
I love to read books
I like to go cycling
```

Importing matplotlib

```python
import matplotlib.pyplot as plt
from wordcloud import WordCloud
```

```python
text = ' '.join(preprocessed_documents)

wordcloud = WordCloud(width=800, height=400, background_color='black').generate(text)
```

```python
plt.figure(figsize=(10, 6))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis=('off')
plt.show()
```

Day-2: Clustering Techniques

Clustering is a type of unsupervised machine learning that involves grouping data points together based on their similarity. Clustering algorithms are used to find patterns in data that would not be apparent if the data were simply analyzed individually.

There are many different clustering techniques available, each with its own strengths and weaknesses. Some of the most common clustering techniques include:

K-means clustering is a simple and efficient clustering algorithm that divides the data into a predefined number of clusters.

Hierarchical clustering is a more complex clustering algorithm that builds a hierarchy of clusters, starting with individual data points and merging them together until there is only one cluster left.

Density-based clustering identifies clusters of data points that are densely packed together, but have low density in between.

Fuzzy clustering allows data points to belong to multiple clusters at the same time.

The choice of clustering technique will depend on the specific application and the nature of the data. For example, k-means clustering is a good choice for data that is well-separated into distinct clusters, while hierarchical clustering is a good choice for data that is more continuous.

Clustering techniques are used in a wide variety of applications, including:

Market research and customer segmentation : Clustering can be used to segment customers into groups based on their interests, demographics, or purchase history.

Biological data analysis : Clustering can be used to identify patterns in biological data, such as gene expression data or protein interactions.

Image analysis : Clustering can be used to segment images into different objects or regions.

Social network analysis : Clustering can be used to identify groups of users in social networks who are connected to each other.

Clustering techniques are a powerful tool for discovering patterns in data. They can be used to improve the user experience, increase sales, and make better decisions.

Here are some of the benefits of using clustering techniques:

Discovering patterns in data : Clustering techniques can help to identify patterns in data that would not be apparent if the data were simply analyzed individually.

Improving the user experience : Clustering techniques can be used to improve the user experience by grouping similar items together, making it easier for users to find what they are looking for.

Increasing sales : Clustering techniques can be used to increase sales by grouping customers together based on their interests, making it easier to target them with marketing campaigns.

Making better decisions : Clustering techniques can be used to make better decisions by grouping data points together based on their similarity, which can help to identify trends and patterns.

Clustering techniques are a powerful tool that can be used to improve the user experience, increase sales, and make better decisions. However, it is important to choose the right clustering technique for the specific application. There are many different clustering techniques available, each with its own strengths and weaknesses. The best approach will vary depending on the specific application.

It is a technique in ML that groups similar data points together based on their characteristics. It is an unsupervised learning method, meaning it doesnt rely on predefined labels or outcomes. Instead, clustering algorithm identify patterns and similarities in the data to form groups.

1. Centroid-Based –
    - It is like sorting candies into colorful groups based on average colors
    - It groups data around representative points called centroids
    - use it when Data has well defined clusters / Data has similar variance within clusters / No. of clusters is known or estimated
    - Dont use it when Data has irregular shapes or Clusters have varying sizes or Outliers are present

- K-means, K-medoids, Fuzzy C-means, etc
- Ex: Customer segemntation based on age and income

2. Density-Based –
    - It is like organizing colorful stickers into groups based on how close they are to each other
    - It groups data based on density and proximity
    - Use it when data has irregular shapes and varying cluster sizes / Clusters have varying densities / Can handle noise and outliers
    - Dont use it when data has global patterns or shapes / high dimensional data / when clusters have similar densities
    - DBSCAN, OPTICS, HDBSCAN, etc.
    - Ex: Identifying hotspots in crime data.

3. Distribution-Based –
    - It is like grouping balloons on how they are spread out or distributed in a bag
    - It groups data based on how they are spread and distributed
    - Use it when data follows specific distributions / can handle irregularity shaped clusters / Cluster sizes are unkown
    - Dont use it when data has overlapping clusters / Clusters have variable densities Gaussian Mixture Models, Expectation-Maximization(EM), etc.

    - Ex: Segmenting customers based on purchasing behaviour

4. Heirarchial –
    - It is like building a tower of toys, stacking them together based on their similarities.
    - Creates a heirarchy of clusters, forming nested subgroups
    - Use it when data has a hirarchial structure / want to explore clusters at different granularities
    - Dont use it when data has varying densities / Scalability is a concern
    - BIRCH, CURE, Agglomerative, Divisive, etc.,

K-MEANS

Parameters in K-Means:

1. n_clusters --> No, of clusters to be formed (integer>1), we use it when approximate number of clusters is known, we dont use it when numbr of clusters are unknown or irrlevant

2. init --> It is initialization method for centroid seeds(k-means++, random, ndarray), we use it 'k-means++' for general purpose, 'random' for larger datasets, we dont use it when we have a specific prior knowledge about centroids.

3. n_init --> No of times the k-means algorithm will be running (integer>0), higher values (>=0) to improve result robustness, we dont use it for fater execution or if the result is already satisfactory

4. max_iter --> No of iterations for a single run (integer >= 1), we use it when to limit the communication time

5. tol --> Tolerance for convergence (Float >= 0.0)

6. algorithm --> Algorithm to compute the k-means (auto, full or elkan)

importing dataset

```
from sklearn.datasets import make_blobs
```

Generate sample data

```
x,y_true = make_blobs(n_samples = 500, centers = 4, cluster_std = 0.6, random_state=0)
```

Importing K-means clustering

```
from sklearn.cluster import KMeans
```

Python

```
kmeans = KMeans(n_clusters = 4,
                init = 'k-means++',
                n_init = 10,
                max_iter = 300,
                tol = 1e-4,
                algorithm = 'auto',
                random_state = 0)
```

Python

```
kmeans.fit(x)
```

Python

```
C:\Users\manas\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\cluster\_kmeans.py:1366: FutureWarning: algorithm='auto' is deprecated, it will be removed in 1.3. Using
  warnings.warn(
```

Dividing labels and centre

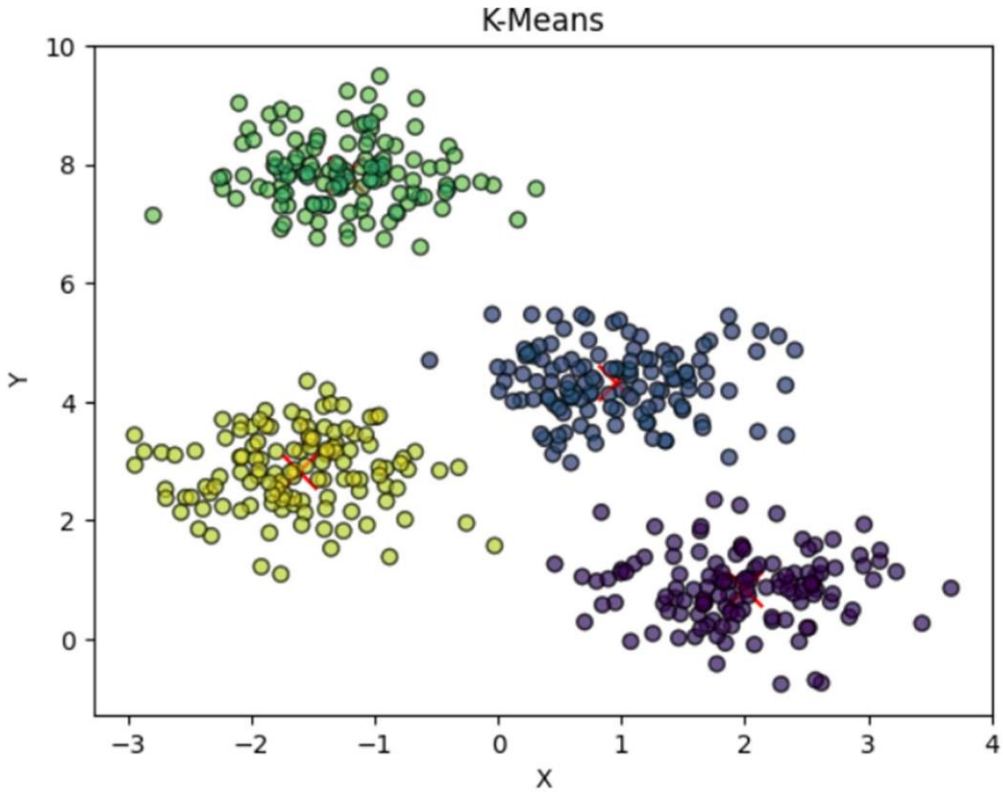
```
labels = kmeans.labels_
centers = kmeans.cluster_centers_
```

]

Importing Matplotlib Library

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

```
# original
plt.scatter(x[:, 0], x[:,1], c=y_true, cmap='viridis', alpha=0.5, edgecolors='k')

# cluster centers
plt.scatter(centers[:, 0], centers[:, 1], c='red', marker='x', s=200)

# predicted labels
plt.scatter(x[:, 0], x[:, 1], c=labels, cmap='viridis', alpha=0.5, edgecolors='k')

plt.title('K-Means')
plt.xlabel('X')
plt.ylabel('Y')
plt.show()
```



DBSCAN

1. eps -The max distance between 2 samples to be considered as neighbors (Floatr >0.0), we dont use when data has unknown distace scale
2. min_samples -Min no of samples required to form a dense region (integer >=1)

3. metric - Distance metric used to measure the similarity b/w samples (string / callable), we dont use it when we have specific requirements
    - euclidean -default and widely used for general purpose
    - manhattan - suitable when you are dealing with attributes of different scales
    - cosine - measures the angular difference b/w vectors
    - callable function - use when non of the provided options meet your requirements
4. algorithm - Algorithm used to compute the dbscan clustering (auto, ball_tree, kd_tree, brute), we dont use it when we have specific requirements
    - auto - default and automatically selects the most appropriate algorithm
    - ball_tree / kd_tree - high dimensional data
    - brute -computes full distance matrix, but not efficient for larger for datasets
5. leaf_size - Leaf size of KD-Tree / Ball-Tree(integer>0), smaller values for better performance, dont use it when using brute algorithm.
6. p - Power parameter for Minkowaski metric (float >=1.0)

1. Manhattan
2. Euclideam

Importing make_moon dataset, DBSCAN

```
from sklearn.datasets import make_moons
```

```
x,y_true = make_moons(n_samples = 200, noise = 0.1, random_state = 0)
```

```
from sklearn.cluster import DBSCAN
```

```
dbscan = DBSCAN(eps = 0.3,
                min_samples = 5,
                metric = 'euclidean',
                algorithm = 'auto')
```

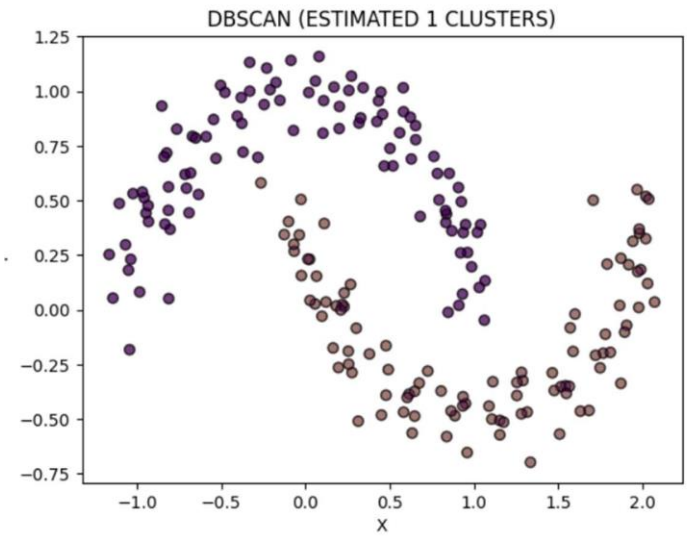
```
dbscan.fit(x)
```

```
labels = dbscan.labels_
n_clusters = len(set(labels)) - (1 if -1 in labels else 0)
```

Plotting the values

```
#original
plt.scatter(x[:,0], x[:,1], c = y_true, cmap='viridis', alpha = 0.5, edgecolors = 'k')

# predicted labels
plt.scatter(x[:,0], x[:,1], c = labels, cmap='viridis', alpha = 0.5, edgecolors = 'k')

plt.title(f'DBSCAN (ESTIMATED {n_clusters} CLUSTERS)')
plt.xlabel('X')
plt.ylabel('Y')
plt.show()
```

Plot



Day – 3: Deployment in AWS

Deployment in AWS using Elastic Beanstalk and AWS Code pipeline.

GitHub repository: https://github.com/ManasaLakshmi1802/House-price-prediction.git Flask

code: app.py

```python
# pip install flask

from flask import Flask, render_template, request
import numpy as np
import joblib
import os
app = Flask(__name__)

# Load the pre-trained model
path=os.path.dirname(os.path.abspath(__file__))
model = joblib.load(os.path.join(path,'house_prediction_model.pkl'))

# Define the route for the home page
@app.route('/')
def home():
    return render_template('index.html')

# Define the route for the prediction page
@app.route('/predict', methods=['POST'])
def predict():
    # Get input values from user
    bed = int(request.form['bedrooms'])
    bath = int(request.form['bathrooms'])
    loc = int(request.form['location'])
    size = int(request.form['size'])
    status = int(request.form['status'])
    face = int(request.form['facing'])
    Type = int(request.form['type'])

    # Create input data array
    input_data = np.array([[bed, bath, loc, size, status, face, Type]])

    # Predict the price using the pre-trained model
    predicted_price = model.predict(input_data)[0]

    # Render the result page with predicted price
    return render_template('index.html', predicted_price=predicted_price)

if __name__ == '__main__':
    app.run()
```

Index.html

```html
<!DOCTYPE html>
<html>
<head>
    <title>House Price Prediction</title>
</head>
<body>
    <h1>House Price Prediction</h1>
    <form action="/predict" method="post">
        <label for="bedrooms">Number of Bedrooms:</label>
        <input type="number" id="bedrooms" name="bedrooms" required><br><br>
        <label for="bathrooms">Number of Bathrooms:</label>
        <input type="number" id="bathrooms" name="bathrooms" required><br><br>
        <label for="location">Location:</label>
        <select id="location" name="location" required>
            <option value="0">Poranki</option>
            <option value="1">Kankipadu</option>
            <option value="2">Benz Circle</option>
            <option value="3">Gannavaram</option>
            <option value="4">PNT Colony</option>
            <option value="5">Labbipet</option>
            <option value="6">Gunadala</option>
            <option value="7">Gollapudi</option>
            <option value="8">Enikepadu</option>
            <option value="9">Vidhyadharpuram</option>
            <option value="10">Penamaluru</option>
            <option value="11">Payakapuram</option>
        </select><br><br>
        <label for="size">Size (in sqft):</label>
        <input type="number" id="size" name="size" required><br><br>
        <label for="status">Status of Property:</label>
        <select id="status" name="status" required>
            <option value="0">Resale</option>
            <option value="1">Under Construction</option>
            <option value="2">Ready to move</option>
            <!-- Add more options for other status values -->
        </select><br><br>
        <label for="facing">Direction of Facing:</label>
        <select id="facing" name="facing" required>
            <option value="0">Not Mentioned</option>
            <option value="1">East</option>
            <option value="2">West</option>
            <option value="3">NorthEast</option>
            <!-- Add more options for other facing values -->
        </select><br><br>
        <label for="type">Type of Property:</label>
        <select id="type" name="type" required>
            <option value="0">Apartment</option>
            <option value="1">Independent House</option>
            <option value="2">Residential Plot</option>
            <!-- Add more options for other property types -->
        </select><br><br>
        <input type="submit" value="Predict">
    </form>

    <h1>House Price Prediction Result</h1>
    <p>Predicted House Price: ₹ {{ predicted_price }} Lakhs</p>

</body>
</html>
```

requirements.txt:

File Edit View

```
flask==2.2.5
scikit-learn==1.2.2
numpy==1.24.3
joblib==1.2.0
```

.ebextensions: 01_flask.config

```
option_settings:
  aws:elasticbeanstalk:application:environment:
    PYTHONPATH: "/var/app/current:$PYTHONPATH"
  aws:elasticbeanstalk:container:python:
    WSGIPath: "app:app"
```

Day – 4: Project Allocation and Explanation

Allotted Project: Optimizing Workplace Health: Predicting Employee Stress for Pre-emptive Remediation using Machine Learning

Week – 6 (19-06-2023 to 24-06-2023)

Day & Date	Learning Outcomes
Day – 1	Dataset Collection
Day – 2	Dataset Collection contd.
Day – 3	Pre-processing and Feature Scaling
Day – 4	Pre-processing and Feature Scaling contd.
Day – 5	Model Building
Day – 6	Model Building contd.

Day – 1: Dataset Collection

Data collection is a crucial step in the research process that involves gathering relevant information to address a specific research question or objective.

The following attributes are used to predict whether a employee has stress or not:

1. age

2.average daily hours

3. department

4.education

5.education field

6.gender

7.has flexible timings

8.is individual contributor

9.JobInvolvement

10.JobRole

11.JobSatisfaction

12.LeavesTaken

13.MaritalStatus

14.MicromanagedAtWork

15.MonthlyIncome

16.NumCompaniesWorked

17.PercentSalaryHike

18.PerformanceRating

19.RelationshipSatisfaction

20.RemoteWorkSatistfaction

21.SelfMotivationLevel

22.TotalWorkingYears

23.TrainingTimesLastYear

24.WorkLifeBalance

25.WorkLoadLevel

26.YearsAtCompany

27.YearsSinceLastPromotion

28.YearsWithCurrManager

It's important for researchers to select data collection methods and techniques that align with their research objectives, resources, and constraints. The chosen theory or combination of theories will depend on the research design, methodology, and nature of the research question or hypothesis being investigated.

With the above mentioned attributes as input features, whether the employee has stress or not is predicted.

Day – 2: Dataset Collection contd.

	EmployeeID	Target	Age	AvgDailyHours	Department	Education	EducationField	Gender	HasFlexibleTimings	IsIndividualContributor	...	RelationshipSatisfaction	RemoteWorkSati
0	100001	0	36.0	6.45	Sales	5	Technical Degree	Male	No	Yes	...	1	
1	100002	0	24.0	8.48	Sales	5	Technical Degree	Male	No	Yes	...	1	
2	100003	0	45.0	6.93	Research & Development	2	Technical Degree	Female	No	Yes	...	4	
3	100004	1	29.0	7.10	Sales	4	Medical	Male	No	No	...	4	
4	100005	0	30.0	7.30	Human Resources	3	Life Sciences	Female	No	Yes	...	1	
...	...	...	...	...	...	...	...	...	...	...	...	...	
15995	115996	0	19.0	6.81	Sales	2	Other	Male	Yes	Yes	...	3	
15996	115997	0	56.0	7.60	Sales	2	Technical Degree	Female	No	No	...	2	
15997	115998	0	26.0	6.52	Human Resources	1	Other	Female	No	Yes	...	3	
15998	115999	0	29.0	6.11	Research & Development	4	Human Resources	Female	Yes	No	...	1	
15999	116000	0	41.0	10.03	Human Resources	3	Human Resources	Male	No	No	...	2	

16000 rows × 30 columns

Day – 3: Pre-processing and Feature Scaling

Info of training data before pre-processing:

```
#Info of Train Set
TrainData.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 16000 entries, 0 to 15999
Data columns (total 30 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   EmployeeID                16000 non-null  int64
 1   Target                    16000 non-null  int64
 2   Age                       15393 non-null  float64
 3   AvgDailyHours             16000 non-null  float64
 4   Department                16000 non-null  object
 5   Education                 16000 non-null  int64
 6   EducationField            16000 non-null  object
 7   Gender                    15652 non-null  object
 8   HasFlexibleTimings        15831 non-null  object
 9   IsIndividualContributor   15675 non-null  object
 10  JobInvolvement            16000 non-null  int64
 11  JobRole                   15630 non-null  object
 12  JobSatisfaction           16000 non-null  int64
 13  LeavesTaken               15951 non-null  float64
 14  MaritalStatus             15766 non-null  object
 15  MicromanagedAtWork        15862 non-null  float64
```

Pre-processing of training data:

```
#Pre-processing
le = LabelEncoder()
for column in TrainData.columns:
  print(column)
  TempVal = le.fit_transform(TrainData[column].astype('category'))
  TrainData.drop(labels=[column], axis="columns", inplace=True)
  TrainData[column] = TempVal
  '''
  encoded_labels = le.classes_
  encoded_values = range(len(encoded_labels))
  labelMap = dict(zip(encoded_labels, encoded_values))
  print(labelMap)
  '''
```

```
EmployeeID
Target
Age
AvgDailyHours
Department
Education
EducationField
Gender
HasFlexibleTimings
IsIndividualContributor
JobInvolvement
```

Info of training data after Pre-processing:

```
#Info of Train Set after Pre-processing
TrainData.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 16000 entries, 0 to 15999
Data columns (total 30 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   EmployeeID                16000 non-null  int64
 1   Target                    16000 non-null  int64
 2   Age                       16000 non-null  int64
 3   AvgDailyHours             16000 non-null  int64
 4   Department                16000 non-null  int64
 5   Education                 16000 non-null  int64
 6   EducationField            16000 non-null  int64
 7   Gender                    16000 non-null  int64
 8   HasFlexibleTimings        16000 non-null  int64
 9   IsIndividualContributor   16000 non-null  int64
 10  JobInvolvement            16000 non-null  int64
 11  JobRole                   16000 non-null  int64
 12  JobSatisfaction           16000 non-null  int64
 13  LeavesTaken               16000 non-null  int64
 14  MaritalStatus             16000 non-null  int64
 15  MicromanagedAtWork        16000 non-null  int64
```

Day – 4: Pre-processing and Feature Scaling contd.

Info test data before pre-processing:

```
#Info of Test data before Pre-preocessing
TestData.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4000 entries, 0 to 3999
Data columns (total 29 columns):
 #   Column                     Non-Null Count  Dtype  
---  ------                     --------------  -----  
 0   EmployeeID                 4000 non-null   int64  
 1   Age                        3873 non-null   float64
 2   AvgDailyHours              4000 non-null   float64
 3   Department                 4000 non-null   object 
 4   Education                  4000 non-null   int64  
 5   EducationField             4000 non-null   object 
 6   Gender                     3917 non-null   object 
 7   HasFlexibleTimings         3971 non-null   object 
 8   IsIndividualContributor    3921 non-null   object 
 9   JobInvolvement             4000 non-null   int64  
 10  JobRole                    3907 non-null   object 
 11  JobSatisfaction            4000 non-null   int64  
 12  LeavesTaken                3987 non-null   float64
```

Pre-processing on test data:

```
#Pre-processing on Test Data
le = LabelEncoder()
for column in TestData.columns:
  print(column)
  TempVal = le.fit_transform(TestData[column].astype('category'))
  TestData.drop(labels=[column], axis="columns", inplace=True)
  TestData[column] = TempVal
  '''
  encoded_labels = le.classes_
  encoded_values = range(len(encoded_labels))
  labelMap = dict(zip(encoded_labels, encoded_values))
  print(labelMap)
  '''
```

```
EmployeeID
Age
AvgDailyHours
Department
Education
EducationField
Gender
HasFlexibleTimings
IsIndividualContributor
JobInvolvement
JobRole
```

Info of test data after Pre-processing:

```
#Info after Pre-processing
TestData.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4000 entries, 0 to 3999
Data columns (total 29 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   EmployeeID                4000 non-null   int64
 1   Age                       4000 non-null   int64
 2   AvgDailyHours             4000 non-null   int64
 3   Department                4000 non-null   int64
 4   Education                 4000 non-null   int64
 5   EducationField            4000 non-null   int64
 6   Gender                    4000 non-null   int64
 7   HasFlexibleTimings        4000 non-null   int64
 8   IsIndividualContributor   4000 non-null   int64
 9   JobInvolvement            4000 non-null   int64
 10  JobRole                   4000 non-null   int64
 11  JobSatisfaction           4000 non-null   int64
 12  LeavesTaken               4000 non-null   int64
 13  MaritalStatus             4000 non-null   int64
 14  MicromanagedAtWork        4000 non-null   int64
 15  MonthlyIncome             4000 non-null   int64
 16  NumCompaniesWorked        4000 non-null   int64
 17  PercentSalaryHike         4000 non-null   int64
 18  PerformanceRating         4000 non-null   int64
 19  RelationshipSatisfaction  4000 non-null   int64
 20  RemoteWorkSatistfaction   4000 non-null   int64
 21  SelfMotivationLevel       4000 non-null   int64
```

Day – 5: Model Building

Model building:

1. Train the model: Feed the training data into the selected model and optimize its parameters. This process involves finding the best values for the model's parameters to minimize the error or maximize the performance metric.
2. Evaluate the model: Assess the performance of the trained model on the testing/validation set. Common evaluation metrics include accuracy, precision, recall, F1 score, mean squared error, or others, depending on the problem type.
3. Fine-tune the model: Adjust the model and its parameters to improve its performance. This may involve techniques like hyperparameter tuning, regularization, or model ensemble methods.
4. Validate the model: Once you are satisfied with the model's performance, evaluate it on a separate, unseen dataset (the validation set) to ensure its generalization capability.

Splitting the data:

```
#Splitting the Data
X = TrainData.drop('Target', axis=1)
y = TrainData['Target']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Model building using random forest:

```
#Model = RandomForest
from sklearn.ensemble import RandomForestClassifier
RFClassifier = RandomForestClassifier(n_estimators=100, random_state=42)
RFClassifier.fit(X_train, y_train)
y_pred = RFClassifier.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy(RF ): {accuracy}")
```

```
Accuracy(RF): 0.88375
```

Model building using logistic regression:

```
[ ]  #Model = LogisticRegression
     from sklearn.linear_model import LogisticRegression
     LRClassifier = LogisticRegression(max_iter = 2000)
     LRClassifier.fit(X_train, y_train)
     y_pred = LRClassifier.predict(X_test)
     accuracy = accuracy_score(y_test, y_pred)
     print(f"Accuracy(LR): {accuracy}")
```

```
Accuracy(LR): 0.8853125
```

Model building using decision tree:

```
#Model = DecisionTree
from sklearn.tree import DecisionTreeClassifier
DTClassifier = DecisionTreeClassifier()
DTClassifier.fit(X_train, y_train)
y_pred = DTClassifier.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy(DT): {accuracy}")
```

```
Accuracy(DT): 0.815625
```

Day – 6: Model Building contd.

Model building using SupportVectorMachine:

```
#Model = SupportVectorMachine
from sklearn.svm import SVC
SVMClassifier = SVC()
SVMClassifier.fit(X_train, y_train)
y_pred = SVMClassifier.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy(SVM): {accuracy}")
```

```
Accuracy(SVM): 0.8296875
```

Model building using KNN:

```
#Model = KNN
from sklearn.neighbors import KNeighborsClassifier
KNNClassifier = KNeighborsClassifier()
KNNClassifier.fit(X_train, y_train)
y_pred = KNNClassifier.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy(KNN): {accuracy}")
```

```
Accuracy(KNN): 0.8071875
```

Model building using neural networks:

```
[ ] from sklearn.neural_network import MLPClassifier
    model = MLPClassifier(hidden_layer_sizes=(64, 64), activation='relu', solver='sgd', random_state=42)
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    print(f"Accuracy(NN): {accuracy}")
```

```
Accuracy(NN): 0.8296875
```