AVN INSTITUTE OF ENGINEERING & TECHNOLOGY
PATEL GUDA, IBRAHIMPATNAM (M), R.R.Dist, 501510, T.S
Ph.No. 08415-201345, www.avniet.ac.in, Email: avn.principal@gmail.com
NAAC NATIONAL ASSESSMENT AND ACCREDITATION COUNCIL
Accredited with 'B++' Grade by NAAC

DEPARTMENT OF COMPUTER SCIENCE ENGINEERING

MACHINE LEARNING

LAB MANUAL

Subject Code: CS604PC

Regulation: R22/JNTUH

III B. TECH II SEMESTER

COMPUTER SCIENCE AND ENGINEERING

AVN INSTITUTE OF ENGINEERING AND TECHNOLOGY

Affiliated o JNTUH, Patelguda,(V), Ibrahimpatnam(M) R. R Dist, TS-501510

AVN INSTITUTE OF ENGINEERING & TECHNOLOGY
PATEL GUDA, IBRAHIMPATNAM (M), R.R.Dist, 501510, T.S
Ph.No. 08415-201345, www.avniet.ac.in, Email: avn.principal@gmail.com
NAAC NATIONAL ASSESSMENT AND ACCREDITATION COUNCIL
Accredited with 'B++' Grade by NAAC

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

VISION AND MISSION OF THE INSTITUTION

VISION

To become self-sustainable institution this is recognized for its new age engineering through innovative teaching and learning culture, inculcating research and entrepreneurial ecosystem, and sustainable social impact in the community.

MISSION

- To offer undergraduate and post-graduate programs that is supported through industry relevant curriculum and innovative teaching and learning processes that would help students succeed in their professional careers.
- To provide necessary support structures for students, this will contribute to their personal and professional growth and enable them to become leaders in their respective fields.
- To provide faculty and students with an ecosystem that fosters research and development through strategic partnerships with government organisations and collaboration with industries.
- To contribute to the development of the region by using our technological expertise to work with nearby communities and support them in their social and economic growth.

VISION AND MISSION OF CSE DEPARTMENT

VISION

To be recognized as a department of excellence by stimulating a learning environment in which students andfaculty will thrive and grow to achieve their professional, institutional and societal goals.

MISSION

- To provide high quality technical education to students that will enable life-long learning and buildexpertise in advanced technologies in Computer Science and Engineering.
- To promote research and development by providing opportunities to solve complex engineeringproblems in collaboration with industry and government agencies.
- To encourage professional development of students that will inculcate ethical values and leadership skills while working with the community to address societal issues.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

PROGRAMEDUCATIONALOBJECTIVES(PEOS):

A graduate of the Computer Science and Engineering Program should:

PEO1	**ProgramEducationalObjective1:(PEO1)** The Graduates will provide solutions to difficult and challenging issues in their profession by applying computer science and engineering theory and principles.
PEO2	**ProgramEducationalObjective2:(PEO2)** The Graduates have successful careers in computer science and engineering fields or will be able to successfully pursue advanced degrees.
PEO3	**ProgramEducationalObjective3:(PEO3)** The Graduates will communicate effectively, work collaboratively and exhibit high levels of Professionalism, moral and ethical responsibility.
PEO4	**ProgramEducationalObjective4:(PEO4)** The Graduates will develop the ability to understand and analyze Engineering issues in a broader perspective with ethical responsibility towards sustainable development.

PROGRAMOUTCOMES(POS):

PO1	**Engineering knowledge**: Apply the knowledge of mathematics, science, engineering fundamentals and an engineering specialization to the solution of complex engineering problems.
PO2	**Problem analysis**: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
PO3	**Design/development of solutions**: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

PO4	**Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
PO5	**Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
PO6	**The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
PO7	**Environment and sustainability:** Understand the impact of the professional engineering Solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
PO8	**Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
PO9	**Individual and teamwork:** Function effectively as an individual, and as a member or leader in diverse teams, and in multi-disciplinary settings.
PO10	**Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
PO11	**Project management and finance:** Demonstrate knowledge and understanding of the Engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
PO12	**Life-long learning**: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

PROGRAMSPECIFICOUTCOMES(PSOS):

PSO1	**Problem Solving Skills –** Graduate will be able to apply computational techniques and software principles to solve complex engineering problems pertaining to software engineering.
PSO2	**Professional Skills** – Graduate will be able to think critically, communicate effectively, and collaborate in teams through participation in co and extra-curricular activities.
PSO3	**Successful Career**–Graduates will possess a solid foundation in computer science and engineering that will enable them to grow in their profession and pursue lifelong learning through post-graduation and professional development.

AVN INSTITUTE OF ENGINEERING & TECHNOLOGY
PATEL GUDA, IBRAHIMPATNAM (M), R.R.Dist, 501510, T.S
Ph.No. 08415-201345, www.avniet.ac.in, Email: avn.principal@gmail.com
NAAC
NATIONAL ASSESSMENT AND ACCREDITATION COUNCIL
Accredited with 'B++' Grade by NAAC

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

COURSE OBJECTIVE:

- The objective of this lab is to get an overview of the various machine learning techniques and can demonstrate them using python.

COURSE OUTCOMES:

- Understand modern notions in predictive data analysis.
- Select data, model selection, model complexity and identify the trends.
- Understand a range of machine learning algorithms along with their strengths and weaknesses.
- Build predictive models from data and analyze their performance.

AVN INSTITUTE OF ENGINEERING & TECHNOLOGY
PATEL GUDA, IBRAHIMPATNAM (M), R.R.Dist, 501510, T.S
Ph.No. 08415-201345, www.avniet.ac.in, Email: avn.principal@gmail.com
NAAC NATIONAL ASSESSMENT AND ACCREDITATION COUNCIL
Accredited with 'B++' Grade by NAAC

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Course Name: ML LAB **Course Code:CS604PC**

Year/Semester: III/II **Regulation: R22**

List of experiments:

1. Write a python program to compute Central Tendency Measures: Mean, Median, Mode Measure of Dispersion: Variance, Standard Deviation
2. Study of Python Basic Libraries such as Statistics, Math, Numpy and Scipy
3. Study of Python Libraries for ML application such as Pandas and Matplotlib
4. Write a Python program to implement Simple Linear Regression
5. Implementation of Multiple Linear Regression for House Price Prediction using sklearn
6. Implementation of Decision tree using sklearn and its parameter tuning
7. Implementation of KNN using sklearn
8. Implementation of Logistic Regression using sklearn
9. Implementation of K-Means Clustering
10. Performance analysis of Classification Algorithms on a specific dataset (Mini Project)

Additional list of experiments:

11. Build an artificial neural network by implementing the backpropagation algorithm and test the same using appropriate data sets.
12. Write a program to reduce the dimensions of a given dataset using PCA.

Faculty HoD,CSE

Week1:

Write a python program to compute Central Tendency Measures: Mean, Median, Mode Measure of Dispersion: Variance, Standard Deviation

Description:

Central Tendency Measures:

Central tendency is defined as "the statistical measure that identifies a single value as representative of an entire distribution. It aims to provide an accurate description of the entire data. It is the single value that is most typical/representative of the collected data.

Sure, let's define each measure with examples:

1. Mean:

The mean is the average of a set of numbers. It is calculated by adding up all the values in the dataset and then dividing by the total number of values.

Example:
Consider the following dataset: [10, 20, 30, 40, 50]
Mean = (10 + 20 + 30 + 40 + 50) / 5 = 30

So, the mean of this dataset is 30.

2. Median:

The median is the middle value in a sorted dataset. If there is an odd number of values, the median is simply the middle value. If there is an even number of values, the median is the average of the two middle values.

Example:
Consider the following dataset: [15, 20, 25, 30, 35, 40]
Since there are 6 values, the median is the average of the two middle values, which are 25 and 30.
Median = (25 + 30) / 2 = 27.5 So, the median of this dataset is 27.5.

3. Mode:

The mode is the value that appears most frequently in a dataset. A dataset can have one mode, more than one mode (multi-modal), or no mode if all values occur with the same frequency.

Example:

Consider the following dataset: [10, 20, 30, 20, 40, 50, 20]

In this dataset, the number 20 appears most frequently (3 times), so the mode is 20.

So, the mode of this dataset is 20.

These measures are used to understand the central tendency of a dataset (mean, median) and the most common value(s) in the dataset (mode). They provide insight into the typical or representative value(s) in the data.

Measures of Dispersion:

Measures of dispersion, also known as measures of variability, quantify the spread or dispersion of data points in a dataset. They provide information about how much the individual data points differ from the central tendency measures like the mean, median, or mode. Common measures of dispersion include:

Variance:

Variance measures how far a set of numbers are spread out from their average value (mean). It is calculated as the average of the squared differences from the Mean.

$$\text{Variance} = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2$$

where x_i represents each individual data point, $\bar{x}$ is the mean, and n is the number of data points.

Standard Deviation:

Standard deviation is the square root of the variance. It gives a measure of the amount of variation or dispersion of a set of values.

$$\text{Standard Deviation} = \sqrt{\text{Variance}}$$

Program:

```
import numpy as np
from scipy import stats

# Sample data
data = [12, 15, 18, 20, 22, 25, 30, 32, 35, 40]

# Central tendency measures
mean = np.mean(data)
median = np.median(data)
mode = stats.mode(data)[0][0]
```

```
# Measures of dispersion
variance = np.var(data)
std_deviation = np.std(data)

# Output
print("Central Tendency Measures:")
print("Mean:", mean)
print("Median:", median)
print("Mode:", mode)
print("\nMeasures of Dispersion:")
print("Variance:", variance)
print("Standard Deviation:", std_deviation)
```

Program Without using Modules:

```
def mean(data):
    return sum(data) / len(data)

def median(data):
    sorted_data = sorted(data)
    n = len(sorted_data)
    mid = n // 2
    if n % 2 == 0:
        return (sorted_data[mid - 1] + sorted_data[mid]) / 2
    else:
        return sorted_data[mid]

def mode(data):
    frequency = {}
    for value in data:
        frequency[value] = frequency.get(value, 0) + 1
    max_frequency = max(frequency.values())
    mode = [key for key, val in frequency.items() if val == max_frequency]
    return mode[0] if len(mode) == 1 else mode

def variance(data):
    n = len(data)
    data_mean = mean(data)
    return sum((x - data_mean) ** 2 for x in data) / n



def standard_deviation(data):
```

```
    return variance(data) ** 0.5

def main():
    # Example data
    data = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]

    # Compute central tendency measures
    mean_value = mean(data)
    median_value = median(data)
    mode_value = mode(data)
    print("Mean:", mean_value)
    print("Median:", median_value)
    print("Mode:", mode_value)

    # Compute measures of dispersion
    variance_value = variance(data)
    std_deviation_value = standard_deviation(data)
    print("Variance:", variance_value)
    print("Standard Deviation:", std_deviation_value)

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

Output:

```
Mean: 55.0
Median: 55.0
Mode: [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
Variance: 825.0
Standard Deviation: 28.722813232690143
```

Week2:

Study of Python Basic Libraries such as Statistics, Math, Numpy and Scipy

1. Statistics:

- The `statistics` module is part of the Python Standard Library and provides functions for calculating mathematical statistics of numeric data.
  - It includes functions for computing measures like mean, median, mode, variance, standard deviation, and more.
  - This module is useful for basic statistical analysis tasks and is easy to use for simple data sets.
-

Example:

```
import statistics
    data = [1, 2, 3, 4, 5]
    mean_value = statistics.mean(data)
    median_value = statistics.median(data)
    mode_value = statistics.mode(data)
    print(mean_value)
    print(median_value)
    print(mode_value)
```

Output:

```
3
3
1
```

2. Math:

- The `math` module is also part of the Python Standard Library and provides mathematical functions for operations beyond basic arithmetic.
  - It includes functions for mathematical constants (e.g., pi, e), trigonometric functions, logarithmic functions, exponentiation, and more.
  - This module is useful for more complex mathematical calculations.
  - **Example:**

```
import math
```

```
x = math.sin(math.pi / 2)
y = math.log(10)
z = math.sqrt(25)
print(x)
print(y)
print(z)
```

Output:

```
1.0
2.302585092994046
5.0
```

3. NumPy:

- NumPy (Numerical Python) is a popular library for numerical computing in Python.
- It provides support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays.
- NumPy is widely used in scientific computing, data analysis, and machine learning tasks due to its efficient array operations and mathematical functions.
- **Example:**

```
import numpy as np
data = np.array([1, 2, 3, 4, 5])
mean_value = np.mean(data)
print(mean_value)
median_value = np.median(data)
print(median_value)
variance_value = np.var(data)
print(variance_value)
```

Output:

```
3.0
3.0
2.0
```

4. SciPy:

- SciPy (Scientific Python) builds on top of NumPy and provides additional functionality for scientific computing tasks.
- It includes modules for optimization, interpolation, integration, linear algebra, signal processing, and more.
- SciPy is extensively used in various scientific fields and engineering disciplines for solving complex mathematical problems.

- Example:

```
from scipy import optimize
# Example optimization problem
def objective_function(x):
    return x**2 + 5*x + 6
result = optimize.minimize(objective_function, x0=0)
print(result)
```

Output:

```
      fun: -0.2499999999999991
 hess_inv: array([[0.5]])
      jac: array([-5.96046448e-08])
  message: 'Optimization terminated successfully.'
     nfev: 6
      nit: 2
     njev: 3
   status: 0
  success: True
        x: array([-2.50000002])
```

These libraries play crucial roles in various domains of Python programming, from simple statistical analysis to advanced scientific computations. Understanding how to utilize them effectively can greatly enhance your ability to work with numerical data and solve complex mathematical problems.

Week3:

Study of Python Libraries for ML application such as Pandas and Matplotlib

PANDAS

- Pandas stand for Panel Data System
- Pandas is an open source library for data analysis, Data manipulation and Data Visualization.
- (OR) Pandas provide powerful data structures for data analysis, time series and statistics.
- Pandas works on the top numpy and matplotlib.

Features of pandas

1. Handling huge amount data
2. Missing Data
3. Cleaning up data
4. Alignment and indexing
5. Merging and joining
6. Grouping and Visualizing data
7. Time Series Functionality
8. Allows to load data from multiple file formats
9. Input and Output Tools

Pandas library is used by scikit-learn for ML

Applications of Pandas

1. Recommendation Systems
2. Stock Prediction
3. Big Data and Data Science
4. NLP (Natural Language Processing)
5. Statistics and Analytics
6. Neuroscience

Important data structures of Pandas are,

1. Series
2. DataFrame

Q: What is data analysis?

Data analysis is process of collecting, transforming, cleaning and modeling the data with goal of discovering required information.
Data analysis process consists of the following steps.

1. Data Requirement Specifications
2. Data Collection
3. Data Processing
4. Data Cleaning
5. Data Analysis
6. Communication

What is Series?

Pandas series is a one dimensional array object, this object can hold data of any type. It can be integers, floats, string or python objects.
Pandas series represents or equal to a column in any data base (MsExcel, Oracle, MySQL, SQLServer,..)

What is DataFrame?

DataFrame is a two dimensional array object or data structure. Data stored tabular format, which is rows and columns.
The Dataframe consist of 3 components.

1. Data
2. Rows
3. Columns

How to install pandas?

Other than jupyter and googlecolab, it is required to install pandas lib.

pip install pandas

Pandas Series

Series is single dimension array like object with homogeneous or heterogeneous data.
Series object can be created in different ways.

1. Using array
2. Using Dictionary
3. Using Scalar values
4. Using other iterables

Series is name of the class or type which is used to construct Series object.

Syntax: Series(data,index,dtype)

Data : the source using which series object is created
Index : index values must hashable and must be unique/access labels
dtype: type of the series is defined using dtype.

Creating Empty Series

```
import pandas as pd
import numpy as np
s1=pd.Series(dtype=np.int8)
print(s1)
```

```
Series([], dtype: int8)
```

Creating Series using List object

```
s2=pd.Series([10,20,30,40,50])
print(s2)
s3=pd.Series([10,20,30,40,50],index=['a','b','c','d','e'])
print(s3)
```

```
0    10
1    20
2    30
3    40
4    50
dtype: int64
a    10
b    20
c    30
d    40
e    50
dtype: int64
```

Creating Series using ndarray

```
a=np.ndarray(shape=(5,))
i=0
for value in range(10,60,10):
    a[i]=value
    i+=1
print(a)
print(type(a))
s=pd.Series(a)
print(s)
```

```
[10. 20. 30. 40. 50.]
<class 'numpy.ndarray'>
0    10.0
1    20.0
2    30.0
3    40.0
4    50.0
dtype: float64
```

Creating Series Using Dictionary

We can create series using dictionary (OR) we can pass the dictionary object to series.

Series object is using dictionary values as data and dictionary keys as index labels.

```
sales_dict={2018:50000,2019:60000,2020:75000}
s=pd.Series(sales_dict)
print(s)
emp_dict={'naresh':5000,'suresh':6000,'kishore':9000}
s=pd.Series(emp_dict)
print(s)
```

```
2018    50000
2019    60000
2020    75000
dtype: int64
naresh     5000
suresh     6000
kishore    9000
dtype: int64
```

Creating Series using Scalar values

If the series is created using scalar values we must define index. This index defines the length of series.

```
s=pd.Series(15,index=[0,1,2,3,4])
print(s)
```

```
0    15
1    15
2    15
3    15
4    15
dtype: int64
```

Accessing Data from Series

Series is index based collection, we can read and manipulate data using index. This index starts with 0.

```
s1=pd.Series([100,200,300,400,500])
print(s1)
print(s1[0],s1[1],s1[2],s1[3],s1[4])
s2=pd.Series([1000,2000,3000,4000,5000],index=['a','b','c','d','e'])
print(s2['a'],s2['b'],s2['c'],s2['d'],s2['e'])
print(s2[0],s2[1],s2[2],s2[3],s2[4])
```

```
0    100
1    200
2    300
3    400
4    500
dtype: int64
100 200 300 400 500
1000 2000 3000 4000 5000
1000 2000 3000 4000 5000
```

Reading multiple elements/values from series

Series allows reading multiple elements by defining index labels within list.

```
s1=pd.Series(range(100,1000,100))
print(s1)
print(s1[[0,3,6,8]])
s2=pd.Series([100,200,300,400,500],index=['a','b','c','d','e'])
print(s2)
print(s2[['a','c','e']])
```

```
0    100
1    200
2    300
3    400
4    500
5    600
6    700
7    800
8    900
dtype: int64
0    100
3    400
6    700
8    900
dtype: int64
a    100
b    200
c    300
d    400
e    500
```

0s completed at 7:01 PM

Series allows slicing, to read multiple elements/values.

```
s1=pd.Series(range(100,1000,100))
print(s1)
print(s1[:3])
print(s1[-3:])
print(s1[-1::-1])
```

```
0    100
1    200
2    300
3    400
4    500
5    600
6    700
7    800
8    900
dtype: int64
0    100
1    200
2    300
dtype: int64
6    700
7    800
8    900
dtype: int64
8    900
7    800
6    700
5    600
```

DataFrame

DataFrame is two dimensional array object with heterogeneous data. In DataFrame data is stored in the form of rows and columns.

DataFrame represents a table in database.

How to create DataFrame?

DataFrame can be created in different ways.

1. Series
2. Lists
3. Dictionary
4. Numpy array
5. From another dataframe
6. Data can read from files or database

"DataFrame" is type or class name, to create dataframe object

Syntax:

DataFrame(data,index,columns,dtype)

data : data is taken from various sources
Index : row labels
columns : columns labels
dtype: data type of each column

Creating empty dataframe

```
import pandas as pd
#creating empty dataframe
df=pd.DataFrame()
print(df)
```

```
Empty DataFrame
Columns: []
Index: []
```

Creating DataFrame using dictionary

Dictionary consist of key and values.
Dictionary keys as columns headers and values are columns values

```
d={'empno':[1,2,3,4,5],'ename':['naresh','suresh','rajesh','kishore','raman'],'sal':[5000,6000,7000,9000,6000]}
df=pd.DataFrame(d)
print(df)
```

```
   empno    ename   sal
0      1   naresh  5000
1      2   suresh  6000
2      3   rajesh  7000
3      4  kishore  9000
4      5    raman  6000
```

Create DataFrame using List

A nested list represents the content of dataframe.
Each list within list is represented as row.

```
person_list=[['naresh',50],['suresh',45],['kishore',35]]
df=pd.DataFrame(person_list,columns=['name','age'],dtype=float)
print(df)
```

```
      name   age
0   naresh  50.0
1   suresh  45.0
2  kishore  35.0
```

DataFrame created with missing data

Missing data is identified with NaN(Not a Number)

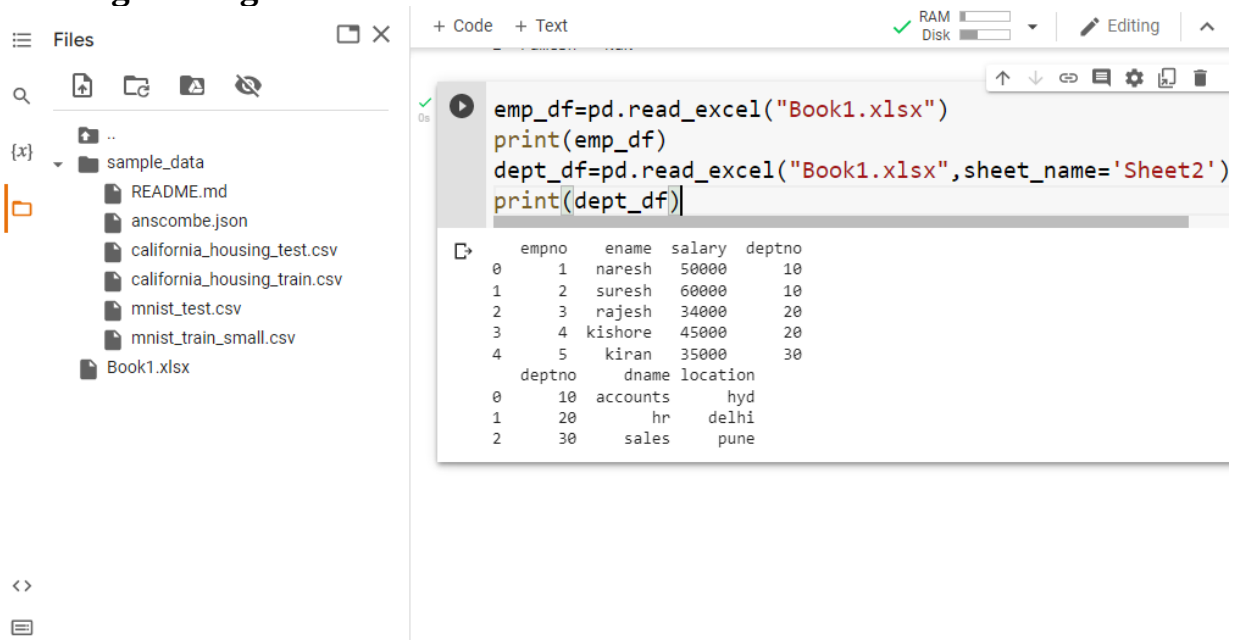
```
data=[['naresh',45],['suresh',56],['kishore',65],['rajesh']]
df=pd.DataFrame(data,columns=['name','age'])
print(df)
```

```
      name   age
0   naresh  45.0
1   suresh  56.0
2  kishore  65.0
3   rajesh   NaN
```

```
data=[{'name':'naresh','age':45},{'name':'kishore'},{'name':'suresh'},{'age':50},{}]
df=pd.DataFrame(data,index=['p1','p2','p3','p4','p5'])
print(df)
```

```
      name   age
p1   naresh  45.0
p2  kishore   NaN
p3   suresh   NaN
p4      NaN  50.0
p5      NaN   NaN
```

Reading/loading data from ms-excel



Selecting Data

1. Row Selection

2. Column Selection

Column Selection

Selecting columns from DataFrame can be done using column header.

```
data=[{'name':'naresh','age':45},{'name':'kishore'},{'name':'suresh'},{'age':50},{}]
df=pd.DataFrame(data)
print(df)
c1=df['name']
c2=df['age']
print(type(c1),type(c2))
print(c1,c2)
```

```
      name   age
0   naresh  45.0
1  kishore   NaN
2   suresh   NaN
3      NaN  50.0
4      NaN   NaN
<class 'pandas.core.series.Series'> <class 'pandas.core.series.Series'>
0     naresh
1    kishore
2     suresh
3        NaN
4        NaN
Name: name, dtype: object 0    45.0
1     NaN
2     NaN
3    50.0
4     NaN
Name: age, dtype: float64
```

Reading multiple columns from DataFrame

In order to read multiple columns, the column names must be defined as a list. It return multiple columns as a dataframe.

single column it read as a series.

```
data={'a':[1,2,3,4,5],'b':[100,200,300,400,500],'c':[1000,2000,3000,4000,5000],'d':[10000,20000,30000,40000,5000]}
df=pd.DataFrame(data)
print(df)
print(df[['a','c']])
r=df[['a','c']]
print(r)
print(type(r))
```

```
   a    b     c      d
0  1  100  1000  10000
1  2  200  2000  20000
2  3  300  3000  30000
3  4  400  4000  40000
4  5  500  5000   5000
   a     c
0  1  1000
1  2  2000
2  3  3000
3  4  4000
4  5  5000
   a     c
0  1  1000
1  2  2000
2  3  3000
3  4  4000
4  5  5000
<class 'pandas.core.frame.DataFrame'>
```

MATPLOTLIB

Matplotlib is a low level graph plotting library in python that serves as a visualization utility. Matplotlib was created by John D. Hunter. Matplotlib is open source and we can use it freely. Matplotlib is mostly written in python, a few segments are written in C, Objective-C and Javascript for Platform compatibility.

Types of Matplotlib

Matplotlib comes with a wide variety of plots. Plots help to understand trends, and patterns, and to make correlations. They're typically instruments for reasoning about quantitative information. Some of the sample plots are covered here.

- Matplotlib Line Plot
- Matplotlib Bar Plot
- Matplotlib Histograms Plot
- Matplotlib Scatter Plot
- Matplotlib Pie Charts
- Matplotlib Area Plot

Import Matplotlib

```
import matplotlib
```

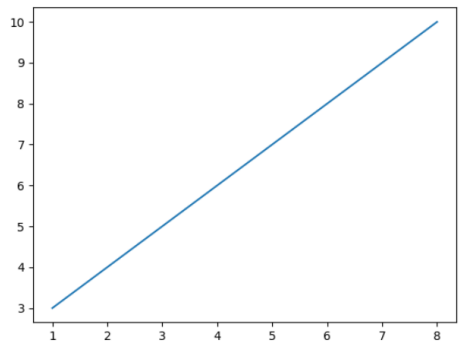
Checking Matplotlib Version

```
import matplotlib

print(matplotlib.__version__)

import matplotlib.pyplot as plt
```

LinePlot

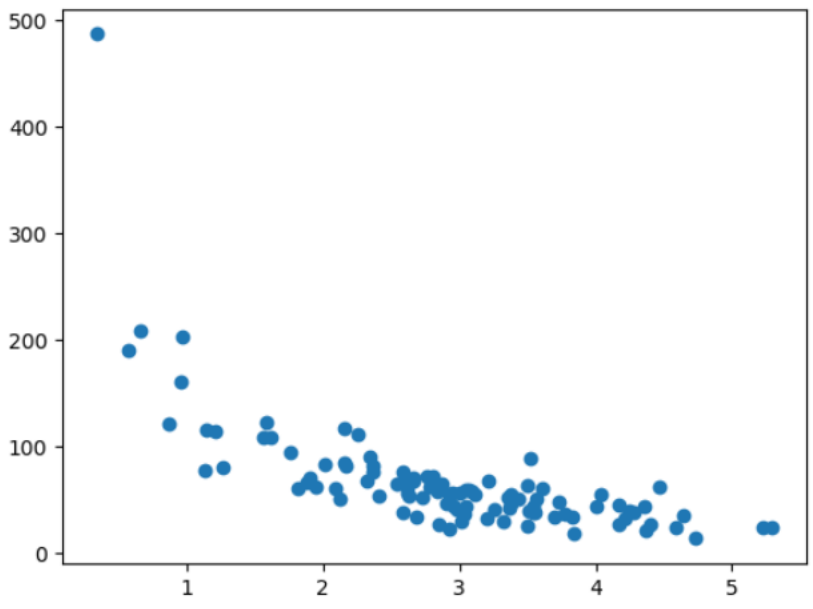
Draw a line in a diagram from position (0,0) to position (6,250):

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

xpoints = np.array([0, 6])
ypoints = np.array([0, 250])

plt.plot(xpoints, ypoints)
plt.show()
```



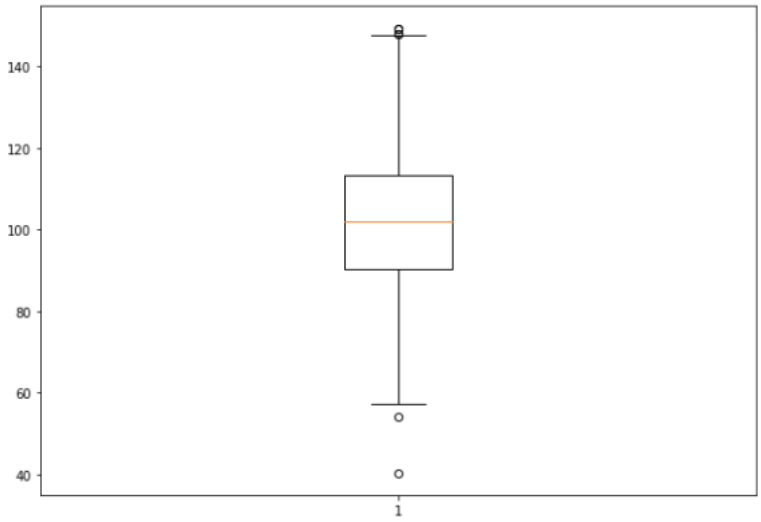
SCATTER PLOT

```
import numpy
import matplotlib.pyplot as plt
numpy.random.seed(2)

x = numpy.random.normal(3, 1, 100)
y = numpy.random.normal(150, 40, 100) / x

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



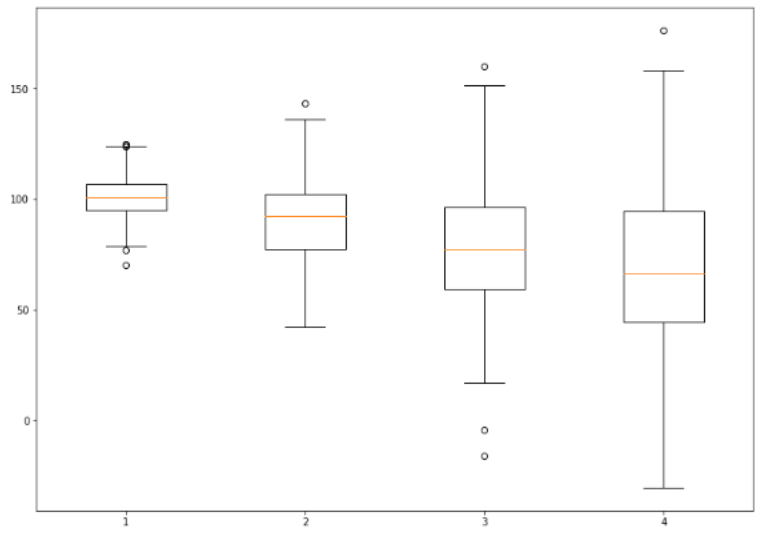
BOXPLOT

```
# Import libraries
import matplotlib.pyplot as plt
import numpy as np


# Creating dataset
np.random.seed(10)
data = np.random.normal(100, 20, 200)

fig = plt.figure(figsize =(10, 7))

# Creating plot
plt.boxplot(data)

# show plot
plt.show()
```



```
# Import libraries
import matplotlib.pyplot as plt
import numpy as np


# Creating dataset
np.random.seed(10)

data_1 = np.random.normal(100, 10, 200)
data_2 = np.random.normal(90, 20, 200)
data_3 = np.random.normal(80, 30, 200)
data_4 = np.random.normal(70, 40, 200)
data = [data_1, data_2, data_3, data_4]

fig = plt.figure(figsize =(10, 7))

# Creating axes instance
ax = fig.add_axes([0, 0, 1, 1])

# Creating plot
bp = ax.boxplot(data)

# show plot
plt.show()
```

Output:



Week-4

Write a Python program to implement Simple Linear Regression

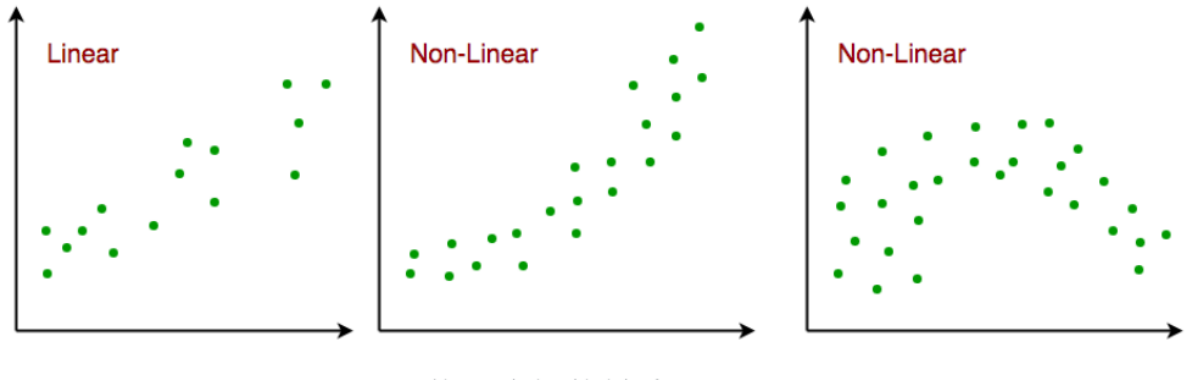
What is Linear Regression?

Linear regression is a statistical method that is used to predict a continuous dependent variable(target variable) based on one or more independent variables(predictor variables). This technique assumes a linear relationship between the dependent and independent variables, which implies that the dependent variable changes proportionally with changes in the independent variables. In other words, linear regression is used to determine the extent to which one or more variables can predict the value of the dependent variable.

Assumptions We Make in a Linear Regression Model:

Given below are the basic assumptions that a linear regression model makes regarding a dataset on which it is applied:

- **Linear relationship**: The relationship between response and feature variables should be linear. The linearity assumption can be tested using scatter plots. As shown below, 1st figure represents linearly related variables whereas variables in the 2nd and 3rd figures are most likely non-linear. So, 1st figure will give better predictions using linear regression.



Types of Linear Regression

There are two main types of linear regression:

- **Simple linear regression:** This involves predicting a dependent variable based on a single independent variable.
- **Multiple linear regression:** This involves predicting a dependent variable based on multiple independent variables.

Simple Linear Regression (Using Least Square Estimation):

Least squares estimation is a method used in statistics and regression analysis to estimate the parameters of a linear model. The goal of least squares estimation is to find the line (or hyperplane in higher dimensions) that best fits a set of data points by minimizing the sum of the squared differences between the observed and predicted values.

In the context of simple linear regression (which involves a single independent variable), the linear model is represented as:

[y = mx + b]

Where:

- y is the dependent variable,

- x is the independent variable,

- m is the slope of the line,

- b is the y-intercept.

Given a set of data points (x_i, y_i) for $i = 1, 2, ..., n$, the goal is to find the values of m and b that minimize the sum of the squared differences between the observed y_i and the predicted values $mx_i + b$. Mathematically, this can be expressed as:

$$\text{minimize} \sum_{i=1}^{n} (y_i - mx_i - b)^2$$

This is the sum of the squared residuals (the differences between the observed and predicted values), also known as the "sum of squared errors" or "SSE". The least squares method finds the values of m and b that minimize this sum.

The least squares estimates for m and b can be calculated using calculus. Specifically, the partial derivatives of the sum of squared residuals with respect to m and b are set to zero, and the resulting equations are solved to find the optimal values.

Once the optimal values of m and b are obtained, the linear regression line can be expressed as y=mx+b, where m is the slope and b is the y-intercept. This line represents the "best fit" line for the given data points in terms of minimizing the sum of squared errors.

```python
import numpy as np

def simple_linear_regression(x, y):
    n = len(x)
    x_mean = np.mean(x)
    y_mean = np.mean(y)

    # Calculate slope (m) and y-intercept (b) using least squares method
    numerator = sum((x[i] - x_mean) * (y[i] - y_mean) for i in range(n))
    denominator = sum((x[i] - x_mean) ** 2 for i in range(n))
    slope = numerator / denominator
    y_intercept = y_mean - slope * x_mean

    return slope, y_intercept

def main():
    # Example data
    x = np.array([1, 2, 3, 4, 5])
    y = np.array([2, 3, 4, 5, 6])

    # Perform linear regression
    slope, y_intercept = simple_linear_regression(x, y)

    # Predict y for a new x value
    new_x = 6
    predicted_y = slope * new_x + y_intercept
    print("Predicted y for x =", new_x, ":", predicted_y)
```

Output:

```
Predicted y for x = 6 : 7.0
```

Week 5:

Implementation of Multiple Linear Regression for House Price Prediction using sklearn

Step-by-step process for implementation of Multiple Linear Regression.

1. Input Data:

- Collect the data consisting of observations where each observation has multiple independent variables (xi1,xi2,.....xim) and a single dependent variable (yi) for i=1,2,...,n.

- Ensure that the data is organized in a tabular format where each row represents an observation and each column represents a variable

2. Preprocessing:

- Standardize or normalize the independent variables if necessary to ensure that they are on a similar scale.
- Handle missing values in the dataset, either by imputation or removal of incomplete records.

3. Model Specification:

Specify the multiple linear regression model with the form:

$$y = \beta_0 + \beta_1 \cdot x_1 + \beta_2 \cdot x_2 + ... + \beta_m \cdot x_m + \epsilon$$

where β_0 is the intercept term, $\beta_1, \beta_2, ..., \beta_m$ are the coefficients corresponding to each independent variable, $x_1, x_2, ..., x_m$, and ϵ is the error term.

4. Estimation of Coefficients:

Use a method such as ordinary least squares (OLS) to estimate the coefficients ($\beta_0, \beta_1, ..., \beta_m$) that minimize the sum of squared errors:

$$\sum_{i=1}^{n}(y_i - (\beta_0 + \beta_1 \cdot x_{i1} + \beta_2 \cdot x_{i2} + ... + \beta_m \cdot x_{im}))^2$$

5. Model Evaluation:

Assess the goodness of fit of the model using techniques such as:

- Coefficient of determination : Measures the proportion of the variance in the dependent variable that is predictable from the independent variables.
- Residual analysis: Examine the residuals (difference between observed and predicted values) to ensure they are randomly distributed around zero and do not exhibit any patterns.
- Cross-validation: Split the data into training and testing sets to evaluate the model's performance on unseen data.

6. Inference:

- Interpret the coefficients to understand the relationships between the independent variables and the dependent variable.
- Test the significance of the coefficients using hypothesis tests (e.g., t-tests) to determine whether they are statistically different from zero.

7**. Prediction:**

Use the estimated regression coefficients to make predictions for new observations by substituting the values of the independent variables into the regression equation.

Program:

```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

def multiple_linear_regression(X, y):
    # Split the data into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

    # Fit the linear regression model
    model = LinearRegression()
    model.fit(X_train, y_train)

    # Predict on the testing set
    y_pred = model.predict(X_test)

    # Evaluate the model
    mse = mean_squared_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)

    return model.coef_, model.intercept_, mse, r2
```

```
def main():
    # Example data
    X = np.array([[1, 2, 3],
                  [4, 5, 6],
                  [7, 8, 9],
                  [10, 11, 12]])
    y = np.array([4, 7, 10, 13])

    # Perform multiple linear regression
    coefficients, intercept, mse, r2 = multiple_linear_regression(X, y)

    # Print results
    print("Coefficients:", coefficients)
    print("Intercept:", intercept)
    print("Mean Squared Error:", mse)
    print("R^2 Score:", r2)
```

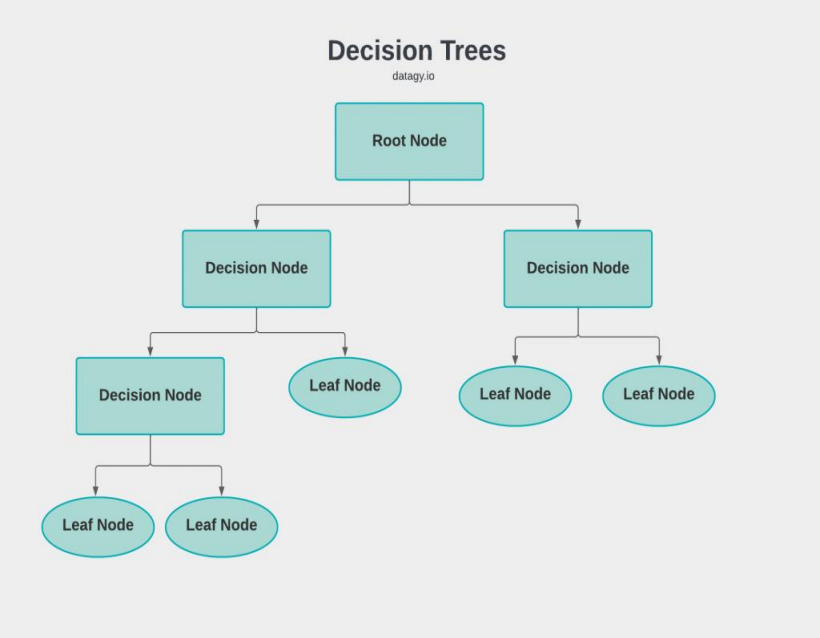
Output:

```
Coefficients: [0.33333333 0.33333333 0.33333333]
Intercept: 2.0
Mean Squared Error: 0.0
R^2 Score: nan
```

Week 6:

Implementation of Decision tree using sklearn and its parameter tuning

Decision tree classifiers are supervised machine learning models. This means that they use prelabelled data in order to train an algorithm that can be used to make a prediction. Decision trees can also be used for regression problems.
Much of the information that you'll learn in this tutorial can also be applied to regression problems.
Decision tree classifiers work like flowcharts. Each node of a decision tree represents a decision point that splits into two leaf nodes.
Each of these nodes represents the outcome of the decision and each of the decisions can also turn into decision nodes. Eventually, the different decisions will lead to a final classification.
The diagram below demonstrates how decision trees work to make decisions. The top node is called the root node. Each of the decision points are called decision nodes. The final decision point is referred to as a leaf node.



Algorithm
Step 1: Loads the Iris dataset.
Step 2: Splits the dataset into training and testing sets.
Step 3: Defines a DecisionTreeClassifier.
Step 4: Sets up a grid of hyperparameters to search over using GridSearchCV.
Step 5: Fits the grid search to the training data.
Step 6: Prints out the best parameters found and the corresponding best score.
Step 7: Uses the best model to make predictions on the test data.
Step 8: Evaluates the model's accuracy on the test data.

Step 9: Visualizes the decision tree using plot_tree function from sklearn.

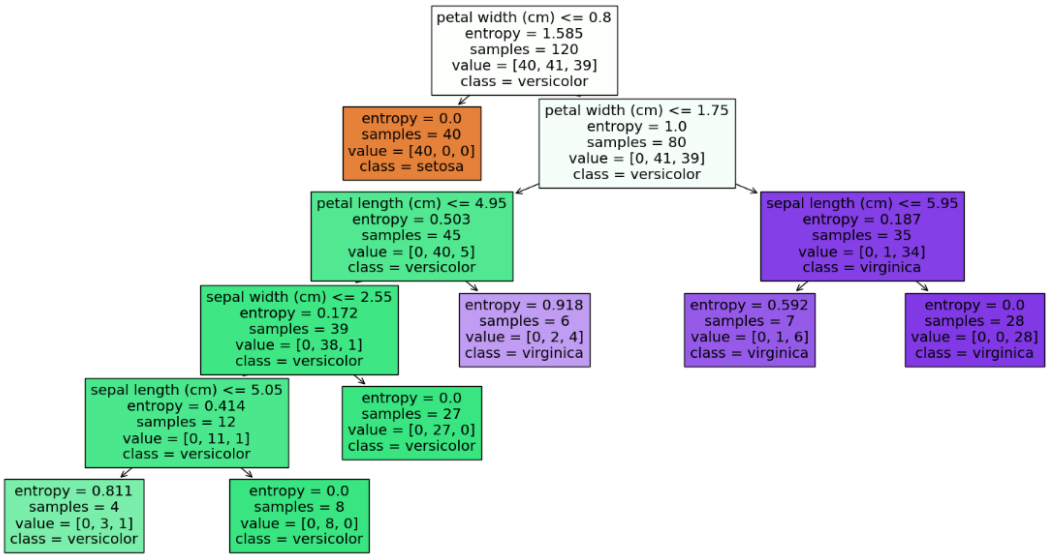
Source code

```
# Importing necessary libraries
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.metrics import accuracy_score

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

# Splitting the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Defining the Decision Tree Classifier
clf = DecisionTreeClassifier()

# Define the grid of parameters to search
param_grid = {
    'criterion': ['entropy','gini'],
    'max_depth': [None, 5, 10, 15, 20],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_features': ['auto', 'sqrt', 'log2']
}

# Create the grid search cross-validation object
grid_search = GridSearchCV(estimator=clf, param_grid=param_grid, cv=5, scoring='accuracy',
n_jobs=-1)

# Fit the grid search to the data
grid_search.fit(X_train, y_train)

# Get the best parameters and the best score
best_params = grid_search.best_params_
best_score = grid_search.best_score_

# Print the best parameters and the best score
```

```
print("Best Parameters:", best_params)
print("Best Score:", best_score)

# Using the best model to make predictions
best_clf = grid_search.best_estimator_
y_pred = best_clf.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

# Plotting the decision tree
plt.figure(figsize=(20,10))
plot_tree(best_clf, filled=True, feature_names=iris.feature_names,
class_names=iris.target_names)
plt.show()
```

Output

Best Parameters: {'criterion': 'entropy', 'max_depth': None, 'max_features': 'log2', 'min_samples_leaf': 4, 'min_samples_split': 2} Best Score: 0.9583333333333334
Accuracy: 1.0



Week-7

Implementation of KNN using sklearn

The K-NN algorithm works by finding the K nearest neighbors to a given data point based on a distance metric, such as Euclidean distance. The class or value of the data point is then determined by the majority vote or average of the K neighbors. This approach allows the algorithm to adapt to different patterns and make predictions based on the local structure of the data.

Distance Metrics

Euclidean Distance

This is nothing but the cartesian distance between the two points which are in the plane/hyperplane.

$$\text{distance}(x, X_i) = \sqrt{\sum_{j=1}^{d} (x_j - X_{i_j})^2]}$$

Manhattan Distance

Manhattan Distance metric is generally used when we are interested in the total distance traveled by the object instead of the displacement. This metric is calculated by summing the absolute difference between the coordinates of the points in n-dimensions.

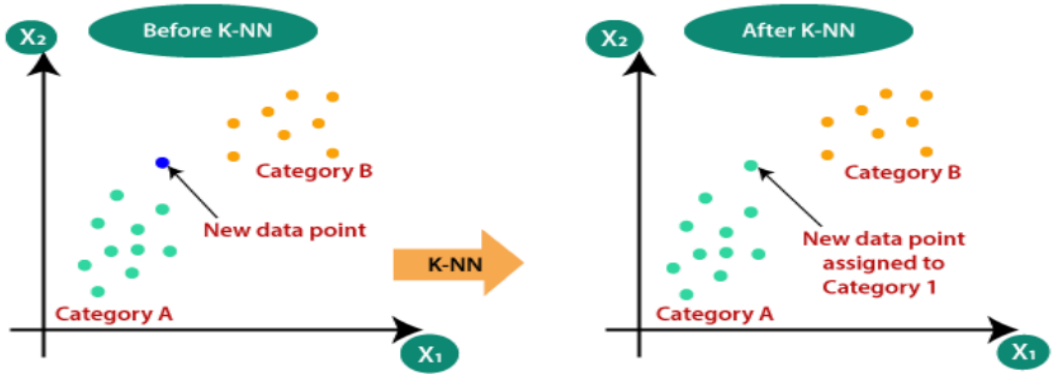
$$d(x, y) = \sum_{i=1}^{n} |x_i - y_i|$$

Minkowski Distance

We can say that the Euclidean, as well as the Manhattan distance, are special cases of the Minkowski distance.

$$d(x, y) = \left(\sum_{i=1}^{n} (x_i - y_i)^p\right)^{\frac{1}{p}}$$

From the formula above we can say that when p = 2 then it is the same as the formula for the Euclidean distance and when p = 1 then we obtain the formula for the Manhattan distance.



Algorithm

Step 1:Loads the Iris dataset.
Step 2:Splits the dataset into training and testing sets.
Step 3:Defines a KNN classifier with k=5.
Step 4:Trains the classifier using the training data.
Step 5:Makes predictions on the test data.
Step 6:Evaluates the model's accuracy on the test data.

You can adjust the value of the number of neighbors considered for classification. Additionally, you can replace the dataset with another dataset you want to work with

Source code

```
import pandas as pd
from sklearn.datasets import load_iris
iris = load_iris()
df = pd.concat([
    (pd.DataFrame(data=iris['data'], columns=iris['feature_names'])),
     (pd.DataFrame(data=iris['target'], columns=['target']))],
            axis=1)
# Split the dataset into features (X) and target (y)
X = df.drop('target', axis=1)
y = df['target']

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20)
```

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X_train)
```

```
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

```
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors=5)
classifier.fit(X_train, y_train)

y_pred = classifier.predict(X_test)

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

Output

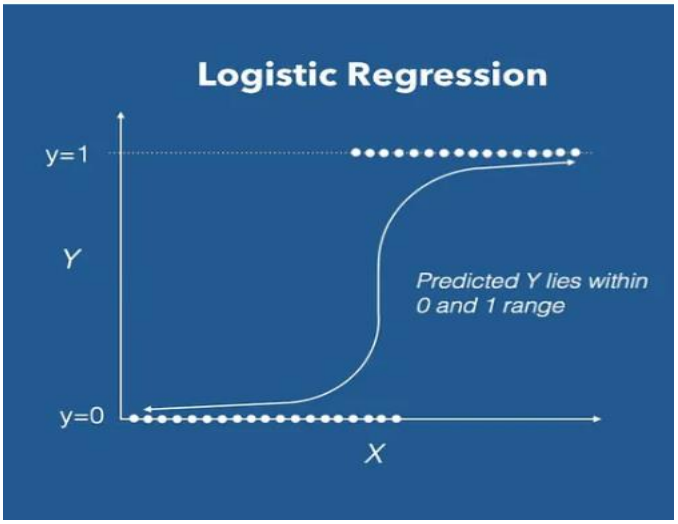
```
[[10  1  0]
 [ 0  8  0]
 [ 0  3  8]]
              precision    recall  f1-score   support

           0       1.00      0.91      0.95        11
           1       0.67      1.00      0.80         8
           2       1.00      0.73      0.84        11

    accuracy                           0.87        30
   macro avg       0.89      0.88      0.86        30
weighted avg       0.91      0.87      0.87        30
```

Week 8:

Implementation of Logistic Regression using sklearn

Logistic regression is a statistical method that is used for building machine learning models where the dependent variable is dichotomous: i.e. binary. Logistic regression is used to describe data and the relationship between one dependent variable and one or more independent variables. The independent variables can be nominal, ordinal, or of interval type.

The name "logistic regression" is derived from the concept of the logistic function that it uses. The logistic function is also known as the sigmoid function. The value of this logistic function lies between zero and one.



cost function can be defined as the 'Sigmoid function' or also known as the 'logistic function'
The hypothesis of logistic regression tends it to limit the cost function between 0 and 1.

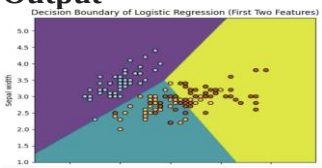
Algorithm

Step 1: Loads the Iris dataset.
Step 2:Uses only the first two features of the data for visualization purposes.
Step 3:Splits the dataset into training and testing sets.
Step 4:Fits a logistic regression model to the training data.
Step 5:Plots the decision boundary, which essentially visualizes the sigmoid curve for logistic regression.
The decision boundary separates the feature space into regions where the logistic regression model predicts different classes. It's a visualization of the decision boundary learned by the logistic regression model.

Source code

```
# Importing necessary libraries
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

# Load the iris dataset
iris = load_iris()
X = iris.data[:, :2]  # Using only the first two features for visualization
y = iris.target

# Splitting the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Defining the Logistic Regression model
clf = LogisticRegression()

# Training the model
clf.fit(X_train, y_train)

# Plotting decision boundary
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.01),
                     np.arange(y_min, y_max, 0.01))
Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
plt.contourf(xx, yy, Z, alpha=0.8)
plt.scatter(X[:, 0], X[:, 1], c=y, edgecolors='k', cmap=plt.cm.Paired)
plt.xlabel('Sepal length')
plt.ylabel('Sepal width')
plt.title('Decision Boundary of Logistic Regression (First Two Features)')
plt.show()
```

Output



Week-9

Implementation of K-Means Clustering

K-means clustering is a type of unsupervised learning, which is used when you have unlabeled data (i.e., data without defined categories or groups). The goal of this algorithm is to find groups in the data, with the number of groups represented by the variable K. The algorithm works iteratively to assign each data point to one of K groups based on the features that are provided. Data points are clustered based on feature similarity. The results of the K-means clustering algorithm are:

The centroids of the K clusters, which can be used to label new data
Labels for the training data (each data point is assigned to a single cluster)
Rather than defining groups before looking at the data, clustering allows you to find and analyze the groups that have formed organically. The "Choosing K" section below describes how the number of groups can be determined.

Each centroid of a cluster is a collection of feature values which define the resulting groups. Examining the centroid feature weights can be used to qualitatively interpret what kind of group each cluster represents.

Algorithm

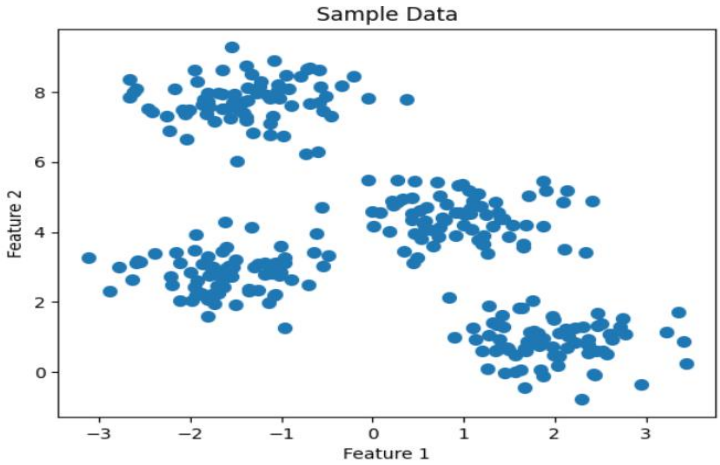
Step 1:Generates sample data using make_blobs function from scikit-learn.
Step 2:Visualizes the sample data.
Step 3:Defines the number of clusters (k).
Step 4:Initializes a KMeans object with the desired number of clusters.
Step 5:Fits the model to the sample data.
Step 6:Retrieves cluster centroids and cluster labels.
Step 7:Visualizes the clusters with cluster centroids.

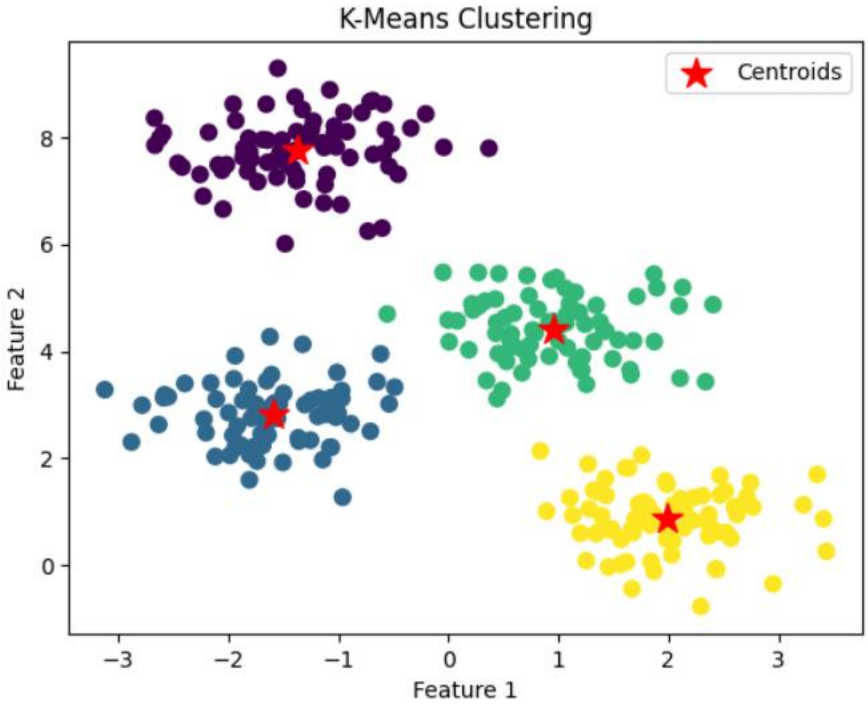
You can adjust parameters such as the number of clusters (k), the number of samples, the centers, and the standard deviation in the make_blobs function to generate different datasets.

Source code

```
# Importing necessary libraries
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans

# Generating sample data
X, _ = make_blobs(n_samples=300, centers=4, cluster_std=0.60, random_state=0)

# Visualizing the sample data
plt.scatter(X[:, 0], X[:, 1], s=50)
plt.title('Sample Data')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.show()

# Defining the number of clusters
k = 4

# Initializing the KMeans object
kmeans = KMeans(n_clusters=k)

# Fitting the model to the data
kmeans.fit(X)

# Getting the cluster centroids
centroids = kmeans.cluster_centers_

# Getting the cluster labels for each data point
labels = kmeans.labels_

# Visualizing the clusters
plt.scatter(X[:, 0], X[:, 1], c=labels, s=50, cmap='viridis')
plt.scatter(centroids[:, 0], centroids[:, 1], marker='*', s=200, c='red', label='Centroids')
plt.title('K-Means Clustering')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.legend()
plt.show()
```

Output


Sample Data
Feature 2
Feature 1
8
6
4
2
0
−3
−2
−1
0
1
2
3


K-Means Clustering
Centroids
Feature 2
Feature 1
8
6
4
2
0
−3
−2
−1
0
1
2
3

Additional Experiment 1

Build an Artificial Neural Network by implementing the Backpropagation algorithm and test the same using appropriate data sets.

Artificial Neural Networks (ANNs) are computational models inspired by the structure and function of the human brain. They are a fundamental building block of modern machine learning and artificial intelligence systems, capable of learning complex patterns and relationships from data. ANNs consist of interconnected nodes, known as neurons, organized into layers. Each neuron processes input signals, performs a computation, and passes the result to neurons in the next layer. This architecture enables ANNs to perform tasks such as classification, regression, clustering, and pattern recognition. Training an ANN involves adjusting the weights and biases of the neurons to minimize a loss function, which measures the difference between the predicted output and the actual output. This process typically involves an optimization algorithm such as gradient descent or its variants. Backpropagation, an efficient algorithm for computing gradients in layered networks, is commonly used to update the weights and biases during training.

Backpropagation is a supervised learning algorithm used for training artificial neural networks. It works by calculating the gradient of the loss function with respect to the weights of the network, and then updating the weights in the direction that minimizes the loss.

Steps of Backpropagation Algorithm:

- Initialize Weights: Initialize the weights and biases of the neural network with small random values.
- Forward Pass: Perform a feedforward pass through the network to compute the output for a given input.
- Compute Loss: Calculate the loss (error) between the predicted output and the actual output using a suitable loss function.
- Backpropagation: Compute the gradients of the loss function with respect to each weight and bias in the network using the chain rule.
- Update Weights: Update the weights and biases of the network using an optimization algorithm like gradient descent. The weights are adjusted in the direction that minimizes the loss.
- Repeat: Repeat steps 2-5 for each input-output pair in the training dataset for a fixed number of epochs or until convergence.
- Evaluate: After training, evaluate the performance of the network on a separate validation or test dataset to assess its generalization ability.

Source code:

```
import numpy as np

X = np.array(([2, 9], [1, 5], [3, 6]), dtype=float)

y = np.array(([92], [86], [89]), dtype=float)

X = X/np.amax(X,axis=0) # maximum of X array longitudinally

y = y/100

#Sigmoid Function

def sigmoid (x):

return 1/(1 + np.exp(-x))

#Derivative of Sigmoid Function

def derivatives_sigmoid(x):

return x * (1 - x)

#Variable initialization

epoch=7000 #Setting training iterations

lr=0.1 #Setting learning rate

inputlayer_neurons = 2 #number of features in data set

hiddenlayer_neurons = 3 #number of hidden layers neurons

output_neurons = 1 #number of neurons at output layer

#weight and bias initialization

wh=np.random.uniform(size=(inputlayer_neurons,hiddenlayer_neurons))

bh=np.random.uniform(size=(1,hiddenlayer_neurons))

wout=np.random.uniform(size=(hiddenlayer_neurons,output_neurons))
```

```
bout=np.random.uniform(size=(1,output_neurons))
#draws a random range of numbers uniformly of dim x*y
for i in range(epoch):
#Forward Propogation
hinp1=np.dot(X,wh)
hinp=hinp1 + bh
hlayer_act = sigmoid(hinp)
outinp1=np.dot(hlayer_act,wout)
outinp= outinp1+ bout
output = sigmoid(outinp)
#Backpropagation
EO = y-output
outgrad = derivatives_sigmoid(output)
d_output = EO* outgrad
EH = d_output.dot(wout.T)
hiddengrad = derivatives_sigmoid(hlayer_act)#how much hidden layer wts
contributed to error


d_hiddenlayer = EH * hiddengrad
wout += hlayer_act.T.dot(d_output) *lr# dotproduct of nextlayererror and
currentlayerop
# bout += np.sum(d_output, axis=0,keepdims=True) *lr
```

```
wh += X.T.dot(d_hiddenlayer) *lr

#bh += np.sum(d_hiddenlayer, axis=0,keepdims=True) *lr

print("Input: \n" + str(X))

print("Actual Output: \n" + str(y))

print("Predicted Output: \n" ,output)
```

output

```
Input:

[[ 0.66666667 1. ]

[ 0.33333333 0.55555556]

[ 1. 0.66666667]]

Actual Output:

[[ 0.92]

[ 0.86]

[ 0.89]]

Predicted Output:

[[ 0.89559591]

[ 0.88142069]

[ 0.8928407 ]]
```

Additional Experiment 2

Write a program to reduce the dimensions of a given dataset using PCA.

Principal Component Analysis is a technique of feature extraction that maps a higher dimensional feature space to a lower-dimensional feature space. While reducing the number of dimensions, PCA ensures that maximum information of the original dataset is retained in the dataset with the reduced no. of dimensions and the co-relation between the newly obtained Principal Components is minimum. The Principal Component Analysis (PCA) algorithm involves several steps to reduce the dimensionality of a dataset while retaining as much variance as possible. Below are the steps involved in PCA:

1. Standardization: If the features of the dataset are on different scales, it's essential to standardize them (subtract the mean and divide by the standard deviation) to ensure that each feature contributes equally to the analysis. This step is crucial for PCA as it is based on covariance, and scaling ensures that the algorithm is not biased towards features with larger scales.

2. Covariance Matrix Computation: Calculate the covariance matrix of the standardized dataset. The covariance matrix summarizes the relationships between the features and is symmetric, with variances on the diagonal and covariances off-diagonal.

3. Eigenvalue Decomposition: Perform eigenvalue decomposition on the covariance matrix to find its eigenvectors and eigenvalues. Eigenvectors represent the directions of maximum variance in the data, while eigenvalues represent the magnitude of variance along each eigenvector. The eigenvectors are also called principal components.

4. Selection of Principal Components: Sort the eigenvectors by their corresponding eigenvalues in descending order. The eigenvectors with the highest eigenvalues capture the most variance in the data and are selected as the principal components. The number of principal components to retain is a hyperparameter determined by the analyst and depends on the desired amount of variance explained.

5. Projection: Project the original data onto the subspace spanned by the selected principal components. This is achieved by multiplying the standardized data by the matrix of selected principal components.

6. Dimensionality Reduction: The final step is to reduce the dimensionality of the dataset by selecting a subset of the projected data. This subset contains the desired number of principal components, effectively reducing the dimensionality of the dataset while preserving as much variance as possible.

Source Code:

```
# Import necessary libraries

from sklearn import datasets # to retrieve the iris Dataset

import pandas as pd # to load the dataframe

from sklearn.preprocessing import StandardScaler # to standardize the features

from sklearn.decomposition import PCA # to apply PCA

import seaborn as sns # to plot the heat maps

#Load the Dataset

iris = datasets.load_iris()

#convert the dataset into a pandas data frame

df = pd.DataFrame(iris['data'], columns = iris['feature_names'])

#display the head (first 5 rows) of the dataset

df.head()

#Standardize the features

#Create an object of StandardScaler which is present in sklearn.preprocessing

scalar = StandardScaler()

scaled_data = pd.DataFrame(scalar.fit_transform(df)) #scaling the data

scaled_data

#Applying PCA

#Taking no. of Principal Components as 3

pca = PCA(n_components = 3)

pca.fit(scaled_data)
```

```
data_pca = pca.transform(scaled_data)

data_pca = pd.DataFrame(data_pca,columns=['PC1','PC2','PC3'])

data_pca.head()
```

Output:

	PC1	PC2	PC3
0	-2.264703	0.480027	-0.127706
1	-2.080961	-0.674134	-0.234609
2	-2.364229	-0.341908	0.044201
3	-2.299384	-0.597395	0.091290
4	-2.389842	0.646835	0.015738