Lab Assignment No.	1B
Title	Apply LDA Algorithm on Iris Dataset and classify which species a given flower belongs to. Dataset Link:https://www.kaggle.com/datasets/uciml/iris
Roll No.	
Class	BE
Date of Completion	
Subject	Computer Laboratory-II :Quantum AI
Assessment Marks	
Assessor's Sign	

EXPERIMENT NO. 1 B

Aim: Apply LDA Algorithm on Iris Dataset and classify which species a given flower belongs to. Dataset Link:https://www.kaggle.com/datasets/uciml/iris

Hardware Requirement:

- 6 GB free disk space.
- 2 GB RAM.
- 2 GB of RAM, plus additional RAM for virtual machines.
- 6 GB disk space for the host, plus the required disk space for the virtual machine(s).
- Virtualization is available with the KVM hypervisor
- Intel 64 and AMD64 architectures

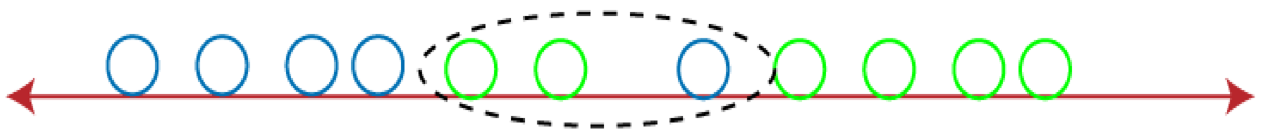
Software Requirement:
Jupyter Nootbook/Ubuntu

Theory:

Linear Discriminant Analysis (LDA) is one of the commonly used dimensionality reduction techniques in machine learning to solve more than two-class classification problems. It is also known as Normal Discriminant Analysis (NDA) or Discriminant Function Analysis (DFA).

This can be used to project the features of higher dimensional space into lower-dimensional space in order to reduce resources and dimensional costs. In this topic, "Linear Discriminant Analysis (LDA) in machine learning”, we will discuss the LDA algorithm for classification predictive modeling problems, limitation of logistic regression, representation of linear Discriminant analysis model, how to make a prediction using LDA, how to prepare data for LDA, extensions to LDA and much more. So, let's start with a quick introduction to Linear Discriminant Analysis (LDA) in machine learning.

Although the logistic regression algorithm is limited to only two-class, linear Discriminant analysis is applicable for more than two classes of classification problems.
Linear Discriminant analysis is one of the most popular dimensionality reduction techniques used for supervised classification problems in machine learning. It is also considered a pre-processing step for modeling differences in ML and applications of pattern classification.
Whenever there is a requirement to separate two or more classes having multiple features efficiently, the Linear Discriminant Analysis model is considered the most common technique to solve such classification problems. For e.g., if we have two classes with multiple features and need to separate them efficiently. When we classify them using a single feature, then it may show overlapping.

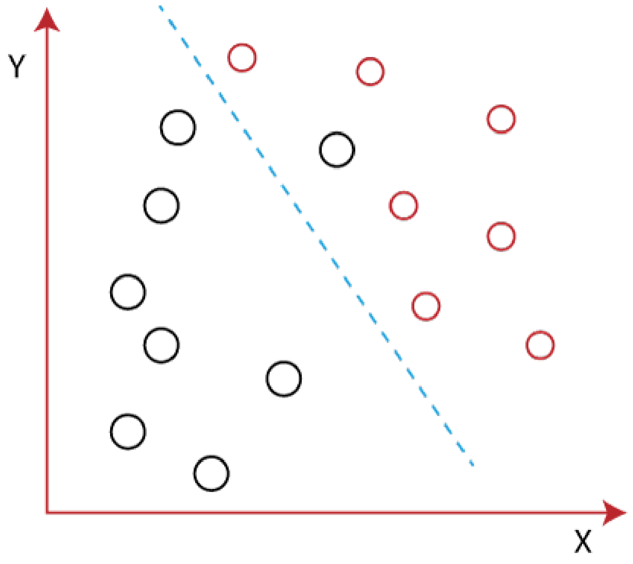


Overlapping

To overcome the overlapping issue in the classification process, we must increase the number of features regularly.

Example:

Let's assume we have to classify two different classes having two sets of data points in a 2-dimensional plane as shown below image:



However, it is impossible to draw a straight line in a 2-d plane that can separate these data points efficiently but using linear Discriminant analysis; we can dimensionally reduce the 2-D plane into the 1-D plane. Using this technique, we can also maximize the separability between multiple classes.

Implementation:

```
import pandas as pd
```

Reference Link: https://medium.com/@betulmesci/dimensionality-reduction-with-principal-component-analysis-and-linear-discriminant-analysis-on-iris-dc1731c07fad

```
df = pd.read_csv("Iris.csv")

print(df)
```

```
Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm  \
0    1            5.1           3.5            1.4           0.2
1    2            4.9           3.0            1.4           0.2
2    3            4.7           3.2            1.3           0.2
3    4            4.6           3.1            1.5           0.2
4    5            5.0           3.6            1.4           0.2
..   ...          ...           ...            ...           ...
145  146          6.7           3.0            5.2           2.3
146  147          6.3           2.5            5.0           1.9
147  148          6.5           3.0            5.2           2.0
148  149          6.2           3.4            5.4           2.3
149  150          5.9           3.0            5.1           1.8

            Species
1      Iris-setosa
2      Iris-setosa
3      Iris-setosa
4      Iris-setosa
5      Iris-setosa
..              ...
145  Iris-virginica
146  Iris-virginica
147  Iris-virginica
148  Iris-virginica
149  Iris-virginica

[150 rows x 6 columns]
```

```
df.Species.unique()
```

```
array(['Iris-setosa', 'Iris-versicolor', 'Iris-virginica'], dtype=object)
```

```python
X = df.drop(['Id','Species'],axis=1)
y = df['Species']

from sklearn.preprocessing import StandardScaler

# Scale the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

from sklearn.discriminant_analysis import LinearDiscriminantAnalysis

# Create an instance of LDA
lda = LinearDiscriminantAnalysis(n_components=2)

# Apply LDA on the scaled features
X_lda = lda.fit_transform(X_scaled, y)
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split

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

# Train a
logistic
regressio
n
classifier
classifier
=
LogisticR
egression
()
classifier.
fit(X_trai
n,
y_train)

LogisticRegression()

        # Suppose you have a new flower with the following measurements:
        new_flower = [[6.7,3.0,5.2,2.3 ]]  # Sepal length, sepal width, petal length, petal
        width

        # Scale the new flower using the same scaler used for training
        new_flower_scaled = scaler.transform(new_flower)

        # Apply LDA on the scaled new flower
        new_flower_lda = lda.transform(new_flower_scaled)

        # Predict the species of the new flower
        predicted_species = classifier.predict(new_flower_lda)

        # Map the predicted label to the actual species
        species_mapping = {'Iris-setosa': 0, 'Iris-
        versicolor': 1, 'Iris-virginica': 2}
        predicted_species_name =
        species_mapping[predicted_species[0]]

        # Print the predicted species
        print("Predicted species:",

predicted_species_name)Predicted
```

```
species: 2

/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does
not have valid feature names,but StandardScaler was fitted with feature names
  warnings.warn
```

EXPERIMENT NO. 2 B

- **Aim**: Use the diabetes data set from UCI and Pima Indians Diabetes data set for performing the following: a. Univariate analysis: Frequency, Mean, Median, Mode, Variance, Standard Deviation, Skewness and Kurtosis b. Bivariate analysis: Linear and logistic regression modeling c. Multiple Regression analysis d. Also compare the results of the above analysis for the two data sets Dataset link: https://www.kaggle.com/datasets/uciml/pima-indians-diabetes-database

- **Hardware Requirement:**

- 6 GB free disk space.
- 2 GB RAM.
- 2 GB of RAM, plus additional RAM for virtual machines.
- 6 GB disk space for the host, plus the required disk space for the virtual machine(s).
- Virtualization is available with the KVM hypervisor
- Intel 64 and AMD64 architectures

- **Software Requirement:**
  Jupyter Nootbook/Ubuntu

- **Theory:**

Descriptive statistics are brief informational coefficients that summarize a given data set, which can be either a representation of the entire population or a sample of a population. Descriptive statistics are broken down into measures of central tendency and measures of variability (spread). Measures of central tendency include the mean, median, and mode, while measures of variability include standard deviation, variance, minimum and maximum variables, kurtosis, and skewness.

Types of Descriptive Statistics

All descriptive statistics are either measures of central tendency or measures of variability, also known as measures of dispersion.

Central Tendency

Measures of central tendency focus on the average or middle values of data sets, whereas measures of variability focus on the dispersion of data. These two measures use graphs, tables and general discussions to help people understand the meaning of the analyzed data.

Measures of central tendency describe the center position of a distribution for a data set. A person analyzes the frequency of each data point in the distribution and describes it using the mean, median, or mode, which measures the most common patterns of the analyzed data set.

Measures of Variability

Measures of variability (or the measures of spread) aid in analyzing how dispersed the distribution is for a set of data. For example, while the measures of central tendency may give a person the average of a data set, it does not describe how the data is distributed within the set.

So while the average of the data maybe 65 out of 100, there can still be data points at both 1 and 100. Measures of variability help communicate this by describing the shape and spread of the data set. Range, quartiles, absolute deviation, and variance are all examples of measures of variability.

Consider the following data set: 5, 19, 24, 62, 91, 100. The range of that data set is 95, which is calculated by subtracting the lowest number (5) in the data set from the highest (100).

Distribution

Distribution (or frequency distribution) refers to the quantity of times a data point occurs. Alternatively, it is the measurement of a data point failing to occur. Consider a data set: male, male, female, female, female, other. The distribution of this data can be classified as:

- The number of males in the data set is 2.
- The number of females in the data set is 3.
- The number of individuals identifying as other is 1.
- The number of non-males is 4.

Univariate vs. Bivariate

In descriptive statistics, univariate data analyzes only one variable. It is used to identify characteristics of a single trait and is not used to analyze any relationships or causations.

For example, imagine a room full of high school students. Say you wanted to gather the average age of the individuals in the room. This univariate data is only dependent on one factor: each person's age. By gathering this one piece of information from each person and dividing by the total number of people, you can determine the average age.

Bivariate data, on the other hand, attempts to link two variables by searching for correlation. Two types of data are collected, and the relationship between the two pieces of information is analyzed together. Because multiple variables are analyzed, this approach may also be referred to as multivariate.

Descriptive Statistics vs. Inferential Statistics

Descriptive statistics have a different function than inferential statistics, data sets that are used to make decisions or apply characteristics from one data set to another.

Imagine another example where a company sells hot sauce. The company gathers data such as the count of sales, average quantity purchased per transaction, and average sale per day of the week. All of this information is descriptive, as it tells a story of what actually happened in the past. In this case, it is not being used beyond being informational.

Let's say the same company wants to roll out a new hot sauce. It gathers the same sales data above, but it crafts the information to make predictions about what the sales of the new hot sauce will be. The act of using descriptive statistics and applying characteristics to a different data set makes the data set inferential statistics. We are no longer simply summarizing data; we are using it predict what will happen regarding an entirely different body of data (the new hot sauce product).

Implementation:

```
import numpy as np
import pandas as pd

df = pd.read_csv("C:/Users/HP/Dropbox/PC/Downloads/diabetes.csv")

df.shape

(768, 9)

df.head()

  Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin  BMI  \
0        6     148        72          35       0  33.6
1        1      85        66         29       0  26.6
2        8     183        64           0       0  23.3
3        1      89        66         23      94  28.1
4        0     137        40          35     168  43.1

   DiabetesPedigreeFunction  Age  Outcome
0              0.627         1
50
1              0.351         0
31
2              0.672         1
32
3              0.167         0
21
4              2.288         1
33

df.describe()

       Pregnancies     Glucose  BloodPressure  SkinThickness     Insulin  \
count  768.000000  768.000000             768.000000 768.000000  768.000000
mean     3.845052                 69.105469 20.536458
         120.894531                         79.799479
std     3.369578               19.355807 15.952218
        31.972618                        115.244002
min      0.0000000.000000  0.000000  0.000000   0.000000
25% 1.000000  99.00000   62.000000  0.000000 0.000000
                 0
50% 3.000000 117.0000    72.000000 23.000000
                    00               30.500000
75% 6.000000 140.2500    80.000000 32.000000
                    00               127.250000
max   17.00000 199.0000    122.00000  99.000000
             0        00    0             846.000000

           BMI  DiabetesPedigreeFunction        Age     Outcome
count 768.000000        768.000000  768.000000
```

```
                              768.000000
mean31.992578                   0.471876   33.240885
                                                0.348958
std      7.884160            0.331329   11.760232
                                             0.476951
min      0.000000              0.078000 21.00000  0.000000
25       27.300000              0.2437500             0.000000
%                                          24.00000
                                           0
50%32.000000               0.372500 29.00000    0.000000
                                                0
75%36.600000               0.626250 41.00000    1.000000
                                                0
max      67.100000           2.420000 81.00000    1.000000
                                                0
```

Univariate analysis: Frequency, Mean, Median, Mode, Variance, Standard Deviation,

Skewness and Kurtosis

```
for column in df.columns:
    print(f"Column: {column}")
    print(f"Frequency:\n{df[column].value_counts()}\n"
    )print(f"Mean: {df[column].mean()}")
    print(f"Median: {df[column].median()}")
    print(f"Mode:\n{df[column].mode()}")
    print(f"Variance: {df[column].var()}")
    print(f"Standard Deviation:
    {df[column].std()}")print(f"Skewness:
    {df[column].skew()}")
    print(f"Kurtosis: {df[column].kurt()}")
    print("            \n")
```

```
Column: Pregnancies
Frequency:
1    135
0    111
2    103
3     75
4     68
5     57
6     50
7     45
8     38
9     28
10    24
11    11
13    10
12     9
14     2
15     1
17     1
Name: Pregnancies, dtype: int64

Mean: 3.8450520833333335
Median:
3.0Mode:
0   1
Name: Pregnancies, dtype:
int64 Variance:
11.35405632062142
Standard Deviation: 3.3695780626988623
Skewness: 0.9016739791518588
Kurtosis: 0.15921977754746486
------------

Column: Glucose
```

Frequency:
99   17

```
100  17
111  14
129  14
125  14
    ..
191   1
177   1
44    1
62    1
190   1
Name: Glucose, Length: 136, dtype: int64

Mean: 120.89453125
Median:
117.0Mode:
0   99
1  100
Name: Glucose, dtype: int64
Variance: 1022.2483142519557
Standard Deviation: 31.97261819513622
Skewness: 0.17375350179188992
Kurtosis: 0.6407798203735053
------------

Column: BloodPressure
Frequency:
70  57
74  52
78  45
68  45
72  44
64  43
80  40
76  39
60  37
0   35
62  34
66  30
82  30
88  25
84  23
90  22
86  21
58  21
50  13
56  12
52  11
54  11
```

```
75    8
92    8
65    7
85    6
94    6
48    5
96    4
44    4
100   3
106   3
98    3
110   3
55    2
108   2
104   2
46    2
30    2
122   1
95    1
102   1
61    1
24    1
38    1
40    1
114   1
Name: BloodPressure, dtype: int64

Mean: 69.10546875
Median:
72.0Mode:
0   70
Name: BloodPressure, dtype:
int64Variance:
374.6472712271838
Standard Deviation: 19.355807170644777
Skewness: -1.8436079833551302
Kurtosis: 5.180156560082496
------------

Column: SkinThickness
Frequency:
0  227
32  31
30  27
27  23
23  22
33  20
28  20
31    19
19    18
39    18
29    17
40    16
25    16
26    16
```

18 20 22 16

```
37   16
41   15
35   15
36   14
15   14
17   14
20   13
24   12
42   11
13   11
21   10
46    8
34    8
12    7
38    7
11    6
43    6
16    6
45   6
14   6
44   5
10   5
48   4
47   4
49   3
50   3
8    2
7    2
52   2
54   2
63   1
60   1
56   1
51   1
99   1
```

```
Name: SkinThickness, dtype: int64

Mean: 20.536458333333332
Median:
23.0Mode:
0   0
Name: SkinThickness, dtype:
int64Variance:
254.47324532811953
Standard Deviation: 15.952217567727677
Skewness: 0.10937249648187608
Kurtosis: -0.520071866153013
------------

Column: Insulin
Frequency:
0   374
105  11
130   9
140   9
120   8
     ..
73    1
171   1
255   1
52    1
112   1
Name: Insulin, Length: 186, dtype:
```

int64Mean: `79.79947916666667`

```
Median:
30.5Mode:
0   0
Name: Insulin, dtype: int64
Variance:
13281.180077955281
Standard Deviation: 115.24400235133837
Skewness: 2.272250858431574
Kurtosis: 7.2142595543487715
-----------

Column: BMI
Frequency:
32.0 13
31.6 12
31.2 12
0.0   11
32.4 10
      ..
36.7  1
41.8  1
42.6  1
42.8  1
46.3  1
Name: BMI, Length: 248, dtype:

int64Mean:

31.992578124999998
Median:
32.0Mode:
0   32.0
Name: BMI, dtype: float64
Variance:
62.15998395738257
Standard Deviation: 7.8841603203754405
Skewness: -0.428981588845356543
Kurtosis: 3.290442900816981
-----------

Column: DiabetesPedigreeFunction
Frequency:
0.258   6
0.254   6
0.268   5
0.207   5
0.261   5
       ..
```

1.353	1
0.655	1
0.092	1
0.926	1

```
0.171 1
Name: DiabetesPedigreeFunction, Length: 517, dtype: int64

Mean: 0.47187630208333325
Median:
0.3725Mode:
0   0.254
1   0.258
Name: DiabetesPedigreeFunction, dtype:
float64Variance: 0.10977863787313938
Standard Deviation: 0.33132859501277484
Skewness: 1.919911066307204
Kurtosis: 5.5949535279830584


Column: Age
Frequency:
22  72
21  63
25  48
24  46
23  38
28  35
26  33
27  32
29  29
31  24
41  22
30  21
37  19
42  18
33  17
38  16
36  16
32  16
45  15
34  14
46  13
43  13
40  13
39  12
35  10
50   8
51   8
52   8
44   8
58   7
47   6
54   6
49   5
48   5
57   5
53   5
60   5
66   4
63   4
62   4
55   4
67   3
56   3
59   3
65   3
69   2
61   2
72   1
81   1
64   1
70   1
68   1

Name: Age, dtype: int64

Mean: 33.240885416666664
Median: 29.0
```

```
Mode:
0   22
Name: Age, dtype: int64
Variance:
138.30304589037365
Standard Deviation: 11.76023154067868
Skewness: 1.1295967011444805
Kurtosis: 0.6431588885398942
------------

Column: Outcome
Frequency:
0   500
1   268
Name: Outcome, dtype: int64

Mean: 0.3489583333333333
Median:
0.0Mode:
0   0
Name: Outcome, dtype: int64
Variance:
0.22748261625380098
Standard Deviation: 0.4769513772427971
Skewness: 0.635016643444986
Kurtosis: -1.600929755156027
------------
```

Bivariate analysis: Linear and logistic regression modeling

```python
from sklearn.linear_model import LinearRegression, LogisticRegression

# Prepare the data
X_linear = df[['Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age']]
y_linear = df['Outcome']

# Fit the linear regression model
model_linear =
LinearRegression()
model_linear.fit(X_linear, y_linear)

# Print the coefficients
print('Linear Regression Coefficients:')
for feature, coef in zip(X_linear.columns, model_linear.coef_):
   print(f'{feature}: {coef}')

# Make predictions
predictions_linear = model_linear.predict(X_linear)
```

Linear Regression Coefficients:
Glucose:

```
0.005932504680360896
```

```
BloodPressure: -0.00227883712542089
SkinThickness: 0.00016697889986787442
Insulin: -0.0002096169514137912
BMI: 0.013310837289280066
DiabetesPedigreeFunction: 0.1376781570786881
Age: 0.005800684345071733
```

```python
# Prepare the data
X_logistic = df[['Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age']]
y_logistic = df['Outcome']

# Fit the logistic regression model
model_logistic = LogisticRegression()
model_logistic.fit(X_logistic, y_logistic)

# Print the coefficients
print('Logistic Regression Coefficients:')
for feature, coef in zip(X_logistic.columns,
   model_logistic.coef_[0]):print(f'{feature}: {coef}')

# Make predictions
predictions_logistic = model_logistic.predict(X_logistic)
```

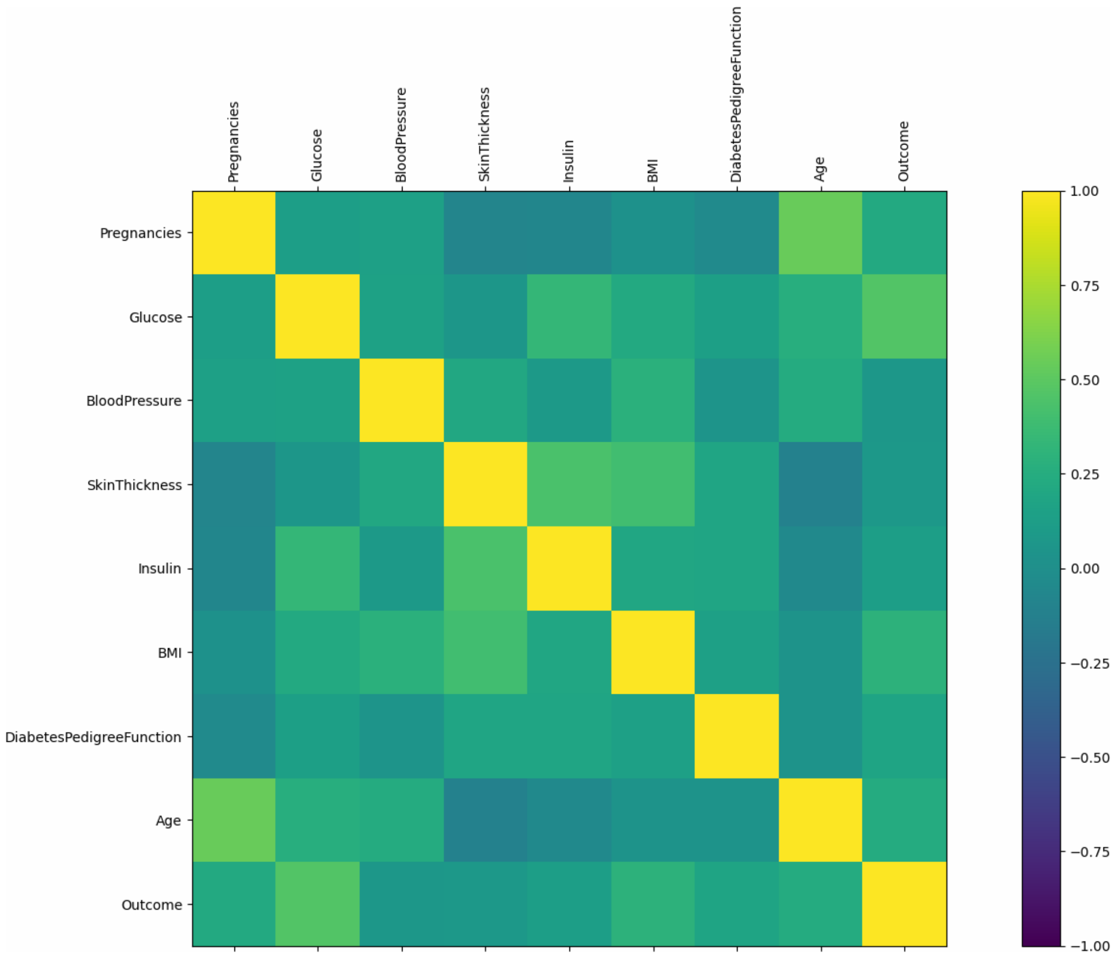
```
Logistic Regression Coefficients:
Glucose: 0.03454477124790582
BloodPressure: -0.01220824032665116
SkinThickness: 0.0010051963882454211
Insulin: -0.0013499454083243116
BMI: 0.08780751006435426
DiabetesPedigreeFunction: 0.8191678019528903
Age: 0.032699759788267134
```

Multiple Regression analysis

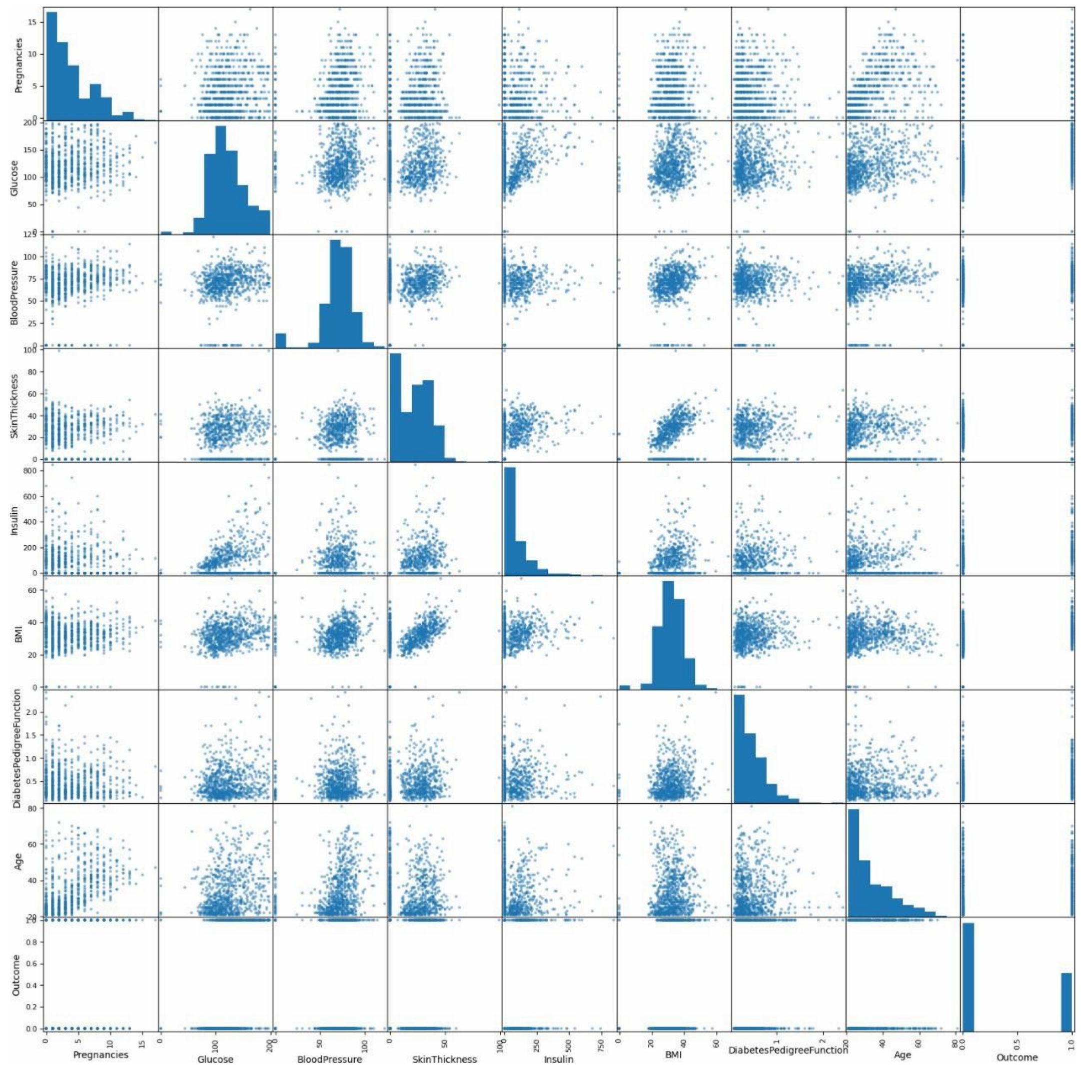
```python
import statsmodels.api as sm

# Split the dataset into the independent variables (X) and the dependent variable (y)
X = df.drop('Outcome', axis=1)  # Independent variables
y = df['Outcome']  # Dependent variable

# Add a constant column to the independent variables
X = sm.add_constant(X)

# Fit the multiple regression model
model = sm.OLS(y, X)
results = model.fit()

# Print the regression results
print(results.summary())
```

```
                    OLS Regression Results
==============================================================================
===================
Dep. Variable:              Outcome  R-squared:                 0.303
Model:                          OLS  Adj. R-squared:            0.296
Method:               Least Squares  F-statistic:               41.29
Date:              Sat, 08 Jul 2023  Prob (F-statistic):        7.36e-55
Time:                      15:59:17  Log-Likelihood:          -381.91
No. Observations:               768  AIC:                       781.8
Df Residuals:                   759  BIC:                       823.6
Df Model:                         8
Covariance Type:          nonrobust
==============================================================================
======================
===========
                        coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------------------
const                    -        0.085     -9.989      0.000      -1.022      -0.686
                    0.8539
Pregnancies           0.0206      0.00       4.01       0.000       0.011       0.031
                                   5          4
Glucose               0.0059      0.00      11.493      0.000       0.005       0.007
                                   1
BloodPressure            -        0.001     -2.873      0.004      -0.004      -0.001
                      0.0023
SkinThickness         0.0002      0.001      0.139      0.890      -0.002       0.002
Insulin                  -        0.000     -1.205      0.229      -0.000       0.000
                      0.0002
BMI                   0.0132      0.002      6.344      0.000       0.009       0.017
DiabetesPedigreeFunction 0.1472   0.045      3.268      0.001       0.05        0.236
                                                                    9
Age                   0.0026      0.002      1.693      0.091      -0.000       0.006
==============================================================================
===================
Omnibus:                     41.539  Durbin-Watson:             1.982
Prob(Omnibus):                0.000  Jarque-Bera (JB):         31.183
Skew:                         0.395  Prob(JB):               1.69e-07
Kurtosis:                     2.408  Cond. No.               1.10e+03
==============================================================================
===================

Notes:
[1]   Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2]The condition number is large, 1.1e+03. This might indicate that there
arestrong multicollinearity or other numerical problems.
```

```
fig = pyplot.figure()
ax = fig.add_subplot(111)
cax = ax.matshow(corr, vmin=-1,
vmax=1)fig.colorbar(cax)
ticks =
np.arange(0,9,1)
ax.set_xticks(ticks)
ax.set_yticks(ticks)
names = df.columns
# Rotate x-tick labels by 90 degrees
ax.set_xticklabels(names,rotation=9
0)ax.set_yticklabels(names)
pyplot.show()
```



```
# Import required package
from pandas.plotting import scatter_matrix
pyplot.rcParams['figure.figsize'] = [20,
20] # Plotting Scatterplot Matrix
scatter_matrix(df)
pyplot.show()
```



Pregnancies
Glucose
BloodPressure
SkinThickness
Insulin
BMI
DiabetesPedigreeFunction
Age
Outcome

EXPERIMENT NO. 3 A

- ☐ **Aim**: Implementation of Support Vector Machines (SVM) for classifying images of handwritten digits into their respective numerical classes (0 to 9).

- ☐ **Hardware Requirement:**

- ■ 6 GB free disk space.
- ■ 2 GB RAM.
- ■ 2 GB of RAM, plus additional RAM for virtual machines.
- ■ 6 GB disk space for the host, plus the required disk space for the virtual machine(s).
- ■ Virtualization is available with the KVM hypervisor
- ■ Intel 64 and AMD64 architectures

- ☐ **Software Requirement:**
  Jypiter Nootbook/Ubuntu

- ☐ **Theory:**

Classification Analysis: Definition

This analysis is a data mining technique used to determine the structure and categories within a given dataset. Classification analysis is commonly used in machine learning, text analytics, and statistical modelling. Above all, it can help identify patterns or groupings between individual observations, enabling researchers to understand their datasets better and make more accurate predictions.

Classification analysis is used to group or classify objects according to shared characteristics. Moreover, this analysis can be used in many applications, from segmenting customers for marketing campaigns to forecasting stock market trends.

Classification Analysis Example

- ● Classifying images

One example of a classification analysis is the use of supervised learning algorithms to classify images. In this case, the algorithm is provided with an image dataset (the training set) that contains labelled images.
The algorithm uses labels to learn how to distinguish between different types of objects in the picture. Once trained, it can then be used to classify new images as belonging to one category or another.

- ● Customer Segmentation

Another example of classification analysis would be customer segmentation for marketing campaigns. Classification algorithms group customers into segments based on their characteristics and behaviours.
This helps marketers target specific groups with tailored content, offers, and promotions that are more likely to appeal to them.

- ● Stock Market Prediction

Finally, classification analysis can also be used for stock market prediction. Classification algorithms can identify patterns between past stock prices and other economic indicators, such as interest rates or unemployment figures. By understanding these correlations, analysts can better predict future market trends and make more informed investment decisions.

These are just some examples of how classification analysis can be applied to various scenarios. Unquestionably, classification algorithms can be used to analyse datasets in any domain, from healthcare and finance to agriculture and logistics.

Classification Analysis Techniques

This analysis is a powerful technique used in data science to analyse and categorise data. Classification techniques are used in many areas, from predicting customer behaviours to finding patterns and trends in large datasets.

This analysis can help businesses make informed decisions about marketing strategies, product development, and more. So, let‟s delve into the various techniques

1. Supervised Learning

Supervised learning algorithms require labelled data. This means the algorithm is provided with a dataset that has already been categorised or labelled with class labels. The algorithm then uses this label to learn how to distinguish between different class objects in the data. Once trained, it can use its predictive power to classify new datasets.

2. Unsupervised Learning

Unsupervised learning algorithms do not require labelled data. Instead, they use clustering and dimensionality reduction techniques to identify patterns in the dataset without any external guidance. These algorithms help segment customers or identify outlier items in a dataset.

3. Deep Learning

Deep learning is a subset/division of machine learning technologies that use artificial neural networks. These algorithms are capable of learning from large datasets and making complex decisions. Deep learning can be used for tasks such as image classification, natural language processing, and predictive analytics.

Classification algorithms can help uncover patterns in the data that could not be detected using traditional methods. By using classification analysis, businesses can gain valuable insights into their customers‟ behaviours and preferences, helping them make more informed decisions.

Implementation:

```
# Import Libraries

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

Handwritten Digit Recognition

Use the sklearn.dataset load_digits() method. It loads the handwritten digits dataset. The returned data is in the form of a Dictionary. The 'data' attribute contains a flattenned array of 64 (each digit image is of 8*8 pixels) elements representing the digits.

The 'target' attribute is the 'class' of Digit (0-9) Each individual digit is represented through a flattendded 64 digit array numbers of Greyscale values. There are 1797 samples in total and each class or digit has roughly 180 samples.

```
from sklearn.datasets import load_digits
digits = load_digits(n_class=10)

digits

{'data': array([[ 0.,  0.,  5., ...,  0.,  0.,  0.],
[ 0.,  0.,  0., ..., 10.,  0.,  0.],
   [ 0., 0., 0., ..., 16.,  9.,
            0.],
    ...,
    [     0., 1., ...,  6.,  0.,
    0.,       0.],
   [ 0., 0., 2., ..., 12.,  0.,
            0.],

    [ 0.,  0., 10., ..., 12.,  1.,
    0.]]),
'target': array([0, 1, 2, ..., 8, 9,
8]),'frame': None,
'feature_names': ['pixel_0_0',
 'pixel_0_1',
 'pixel_0_2',
 'pixel_0_3',
 'pixel_0_4',
 'pixel_0_5',
 'pixel_0_6',
 'pixel_0_7',
 'pixel_1_0',
 'pixel_1_1',
 'pixel_1_2',
 'pixel_1_3',
 'pixel_1_4',
 'pixel_1_5',
 'pixel_1_6',
 'pixel_1_7',
 'pixel_2_0',
 'pixel_2_1',
 'pixel_2_2',
 'pixel_2_3',
 'pixel_2_4',
 'pixel_2_5',
 'pixel_2_6',
 'pixel_2_7',
 'pixel_3_0',
 'pixel_3_1',
 'pixel_3_2',
 'pixel_3_3',
 'pixel_3_4',
 'pixel_3_5',
 'pixel_3_6',
 'pixel_3_7',
 'pixel_4_0',
'pixel_4_1',
'pixel_4_2',
'pixel_4_3',
'pixel_4_4',
'pixel_4_5',
'pixel_4_6',
'pixel_4_7',
'pixel_5_0',
'pixel_5_1',
'pixel_5_2',
'pixel_5_3',
'pixel_5_4',
'pixel_5_5',
'pixel_5_6',
'pixel_5_7',
'pixel_6_0',
'pixel_6_1',
'pixel_6_2',
'pixel_6_3',
'pixel_6_4',
'pixel_6_5',
'pixel_6_6',
'pixel_6_7',
'pixel_7_0',
'pixel_7_1',
'pixel_7_2',
'pixel_7_3',
'pixel_7_4',
'pixel_7_5',
'pixel_7_6',
'pixel_7_7'],

'target_names': array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9]),
'images': array([[[ 0.,  0.,  5., ...,  1.,  0.,  0.],
        [      0., 13., ...,  5.,  0.],
        0.,        15.,
        [      3., 15., ...,  8.,  0.],
        0.,        11.,
        ...,
```

```
    [   0., 6., ...,  0.,  0.,
  0.,       0.]],

  [[   0.,  0.,       5.,  0.
  [    0., 0.,    9., 0., 0.],
  0.,       ...,
  [    0., 3.,    6., 0., 0.],
  0.,       ...,
  ...,
  [    0., 1.,    6., 0., 0.],
  0.,       ...,
  [    0., 1.,    6., 0., 0.],
  0.,       ...,
 [ 0., 0., 0., ...,     0., 0.]],
            10.,

[[ 0., 0., 0., ...,     0., 0.],
            12.,
 [ 0.,  0.,  3., ...,   0.,  0.],
        14.,
 [ 0.,  0.,  8., ...,   0.,  0.],
        16.,
  ...,
  [     9., 16., ...,  0.,  0.],
  0.,   0.,
 [ 0.,  3., 13., ...,   5.,  0.],
        11.,
 [ 0.,  0.,  0., ..., 16.,  9.,
        0.]],

 ...,

[[ 0.,  0.,  1., ...,  1.,  0.,
        0.],
 [ 0.,  0., 13., ...,  2.,  1.,
        0.],
 [ 0.,  0., 16., ..., 16.,  5.,
        0.],
  ...,
 [ 0.,  0., 16., ..., 15.,  0.,
        0.],
 [ 0.,  0., 15., ..., 16.,  0.,
        0.],
 [ 0.,  0.,  2., ...,  6.,  0.,
        0.]],

[[ 0.,  0.,  2., ...,  0.,  0.,
        0.],
 [ 0.,  0., 14., ..., 15.,  1.,
        0.],
 [ 0.,  4., 16., ..., 16.,  7.,
        0.],
  ...,
 [ 0.,  0.,  0., ..., 16.,  2.,
        0.],
 [ 0., 0.,  4., ..., 16.,  2.,
        0.],
 [ 0., 0.,  5., ..., 12.,  0.,
        0.]],

[[ 0.,  0., 10., ...,  1.,  0.,
        0.],
 [ 0., 2., 16., ...,  1.,  0.,
        0.],
 [ 0.,  0., 15., ..., 15.,  0.,
        0.],
  ...,
 [ 0.,  4., 16., ..., 16.,  6.,
        0.],
 [ 0., 8., 16., ..., 16.,  8.,
        0.],
 [ 0., 1.,  8., ..., 12.,  1.,
        0.]]]),
```

'DESCR': ".. _digits_dataset:\n\nOptical recognition of handwritten digits dataset\n--\n\n**Data Set Characteristics:**\n\n   :Number of Instances: 1797\n   :Number of Attributes: 64\n   :Attribute Information: 8x8 image of integer pixels in the range 0..16.\n :Missing Attribute Values: None\n   :Creator: E. Alpaydin (alpaydin '@' boun.edu.tr)\n :Date: July; 1998\n\nThisis a copy of the test set of the UCI ML hand-written digits datasets\nhttps://archive.ics.uci.edu/ml/datasets/Optical+Recognition+of+Handwritten+Digits\n\nThe data set contains images of hand-written digits: 10 classes where\neach class refers to a digit.\n\nPreprocessing programs made available by NIST were used to extract\nnormalized bitmaps of handwritten digits from a preprinted form. From a\ntotal of 43 people, 30 contributed to the training set and different 13\nto the test set. 32x32 bitmaps are divided into nonoverlapping blocks of\n4x4 and the number of on pixels are counted in each block. This generates\nan input matrix of 8x8 where each element is an integer in the range\n0..16. This reduces dimensionality and gives invariance to small\ndistortions.\n\nFor info on NIST preprocessing routines, see M. D. Garris, J. L. Blue, G.\nT. Candela, D. L. Dimmick, J. Geist, P. J. Grother, S. A. Janet, and C.\nL. Wilson, NIST Form-Based Handprint Recognition System, NISTIR 5469,\n1994.\n\n.. topic:: References\n\n - C. Kaynak (1995) Methods of Combining Multiple Classifiers and Their\n   Applications to Handwritten Digit Recognition, MSc Thesis, Institute of\n Graduate Studies in Science and Engineering, Bogazici University.\n - E. Alpaydin, C. Kaynak (1998) Cascading Classifiers, Kybernetika.\n - Ken Tang and Ponnuthurai N. Suganthan and Xi Yao and A. Kai Qin.\n Linear dimensionalityreduction using relevance weighted LDA. School of\n Electrical and Electronic Engineering Nanyang Technological University.\n   2005.\n - Claudio Gentile. A New Approximate Maximal Margin Classification\n   Algorithm. NIPS. 2000.\n"}

```
digits['data'][0].reshape(8,8)
```

```
array([[ 0.,  0.,  5., 13.,  9.,  1.,  0.,  0.],
    [ 0.,  0., 13., 15., 10., 15.,  5.,
          0.],
    [ 0.,  3., 15.,  2.,  0., 11.,  8.,
          0.],
    [ 0.,  4., 12.,  0.,  0.,  8.,  8.,
          0.],
    [ 0.,  5.,  8.,  0.,  0.,  9.,  8.,
          0.],
    [ 0.,  4., 11.,  0.,  1., 12.,  7.,
          0.],
    [ 0.,  2., 14.,  5., 10., 12.,  0.,
          0.],
    [ 0.,  0.,  6., 13., 10.,  0.,  0.,
          0.]])
```

```
digits['data'][0]
```

```
array([ 0.,  0.,  5., 13.,  9., 1.,  0.,  0.,  0.,  0., 13., 15., 10.,
      15., 5.,  0.,   3., 15.,  2.,  0., 11.,  8.,  0.,
          0.,                        0.,  4.,
      12., 0.,  0.,    8.,  0.,  0.,  5.,  8.,  0.,  0.,
          8.,                       9.,  8.,
       0.,  0.,  4.,    0.,  1., 12.,  7.,  0.,  0.,  2.,
         11.,                       14.,  5.,
     10., 12.,  0., 0.,  0.,  6., 13., 10.,  0.,  0.,
          0.,                         0.])
```

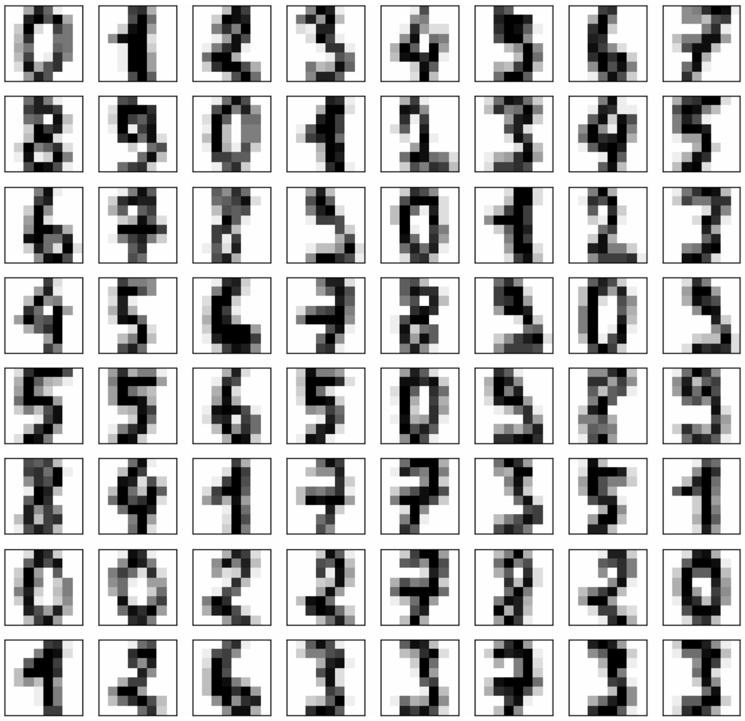
```
digits['images'][1]
```

```
array([[ 0.,  0.,  0., 12., 13.,  5.,  0.,  0.],
    [ 0.,  0.,  0., 11., 16.,  9.,  0.,
          0.],
    [ 0.,  0.,  3., 15., 16.,  6.,  0.,
          0.],
    [ 0.,  7., 15., 16., 16.,  2.,  0.,
          0.],
    [ 0.,  0.,   1., 16.,    3., 0., 0.],
                     16.,
    [ 0.,  0.,   1., 16.,    6., 0., 0.],
                     16.,
    [ 0.,  0.,   1., 16.,    6., 0., 0.],
                     16.,
    [ 0.,  0.,   0., 11.,   10.  0., 0.]])
                     16.,      ,
```

```
digits['target'][0:9]
```

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

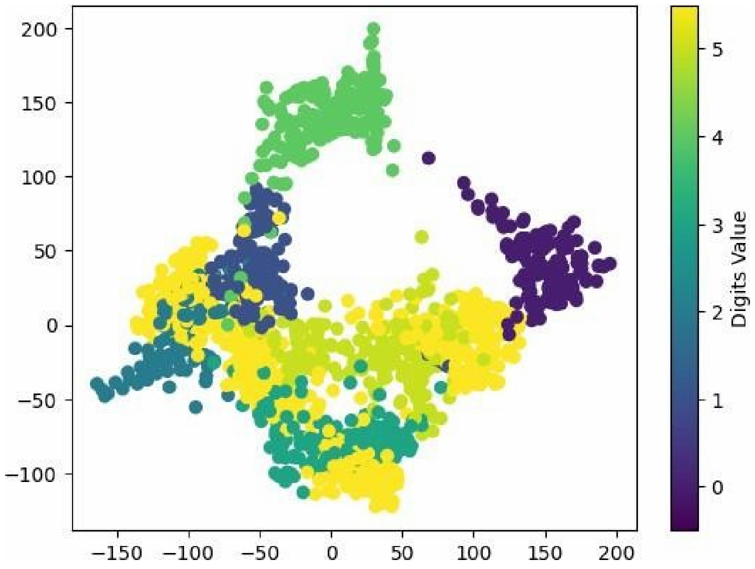
```
digits['target'][0]

0

digits.images[0]

array([[ 0.,  0.,  5., 13.,  9.,  1.,  0.,  0.],
    [ 0.,  0., 13., 15., 10., 15.,  5.,
         0.],
    [ 0.,  3., 15.,  2.,  0., 11.,  8.,
         0.],
    [ 0.,  4., 12.,  0.,  0.,  8.,  8.,
         0.],
    [ 0.,  5.,  8.,  0.,  0.,  9.,  8.,
         0.],
    [ 0.,  4., 11.,  0.,  1., 12.,  7.,
         0.],
    [ 0.,  2., 14.,  5., 10., 12.,  0.,
         0.],
    [ 0.,  0.,  6., 13., 10.,  0.,  0.,
         0.]])

# Each Digit is represented in digits.images as a matrix of 8x8 = 64 pixels. Each of the 64 values
represent
# a greyscale. The Greyscale are then plotted in the right scale by the imshow method.

fig, ax = plt.subplots(8,8, figsize=(10,10))
for i, axi in enumerate(ax.flat):
  axi.imshow(digits.images[i],
  cmap='binary')axi.set(xticks=[],
  yticks=[])
```



```
# Plotting - Clustering the data points after using Manifold Learning

from sklearn.manifold import Isomap

iso = Isomap(n_components=2)
```

```
projection = iso.fit_transform(digits.data)    # digits.data - 64 dimensions to 2 dimensions

plt.scatter(projection[:, 0], projection[:, 1], c=digits.target,

cmap="viridis")plt.colorbar(ticks=range(10), label='Digits Value')
plt.clim(-0.5, 5.5)
```

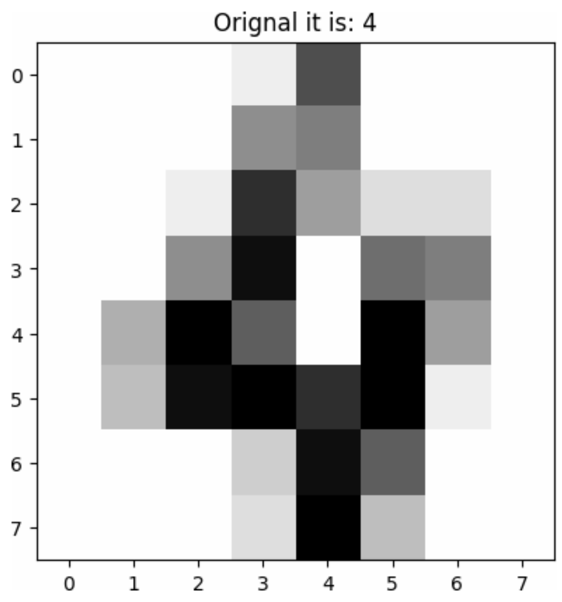
```
/usr/local/lib/python3.10/dist-packages/sklearn/manifold/_isomap.py:373: UserWarning: The number of connected components of the neighbors graph is 2 > 1. Completing the graph to fit Isomap might be slow.Increase the number of neighbors to avoid this issue.
  self._fit_transform(X)
/usr/local/lib/python3.10/dist-packages/scipy/sparse/_index.py:103: SparseEfficiencyWarning: Changing the sparsity structure of a csr_matrix is expensive. lil_matrix is more efficient.
  self._set_intXint(row, col, x.flat[0])
```



```
print(projection[:, 0][70], projection[:, 1][70])
```

```
-56.60683580684862 61.95022367117501
```

```
def view_digit(index):
    plt.imshow(digits.images[index] , cmap =
    plt.cm.gray_r)plt.title('Orignal it is: '+
    str(digits.target[index])) plt.show()

view_digit(4)
```



Use the Support Vector Machine Classifier to train the Data

Use part of the data for train and part of the data for test (predicion)

```
main_data = digits['data']
targets = digits['target']

from sklearn import svm

svc = svm.SVC(gamma=0.001 , C = 100)

# GAMMA is a parameter for non linear hyperplanes.
# The higher the gamma value it tries to exactly fit the training data set
# C is the penalty parameter of the error term.
# It controls the trade off between smooth decision boundary and classifying the training points correctly.

svc.fit(main_data[:1500] ,

targets[:1500]) predictions =

svc.predict(main_data[1501:])

list(zip(predictions , targets[1501:]))
```

```
[(7, 7),                    (8,                 (6, 6),
                            8),
 (4, 4),                    (4,                 (1, 1),
                            4),
 (6, 6),                    (3,                 (7, 7),
                            3),
 (3, 3),                    (1,                 (5, 5),
                            1),
```

(1, 1),	(4, 4),	(4, 4),
(3, 3),	(0, 0),	(4, 4),
(9, 9),	(5, 5),	(7, 7),
(1, 1),	(3, 3),	(2, 2),
(7, 7),	(6, 6),	(8, 8),
(6, 6),	(9, 9),	(2, 2),

(2, 2),	(0, 0),	(7, 7),
(5, 5),	(9, 9),	(9, 4),
(7, 7),	(8, 8),	(6, 6),
(9, 9),	(9, 9),	(3, 3),
(5, 5),	(8, 8),	(1, 1),
(4, 4),	(4, 4),	(3, 3),
(8, 8),	(1, 1),	(9, 9),
(8, 8),	(7, 7),	(1, 1),
(4, 4),	(7, 7),	(7, 7),
(9, 9),	(3, 3),	(6, 6),
(0, 0),	(5, 5),	(8, 8),
(8, 8),	(1, 1),	(4, 4),
(9, 9),	(0, 0),	(3, 3),
(8, 8),	(0, 0),	(1, 1),
(0, 0),	(2, 2),	(4, 4),
(1, 1),	(2, 2),	(0, 0),
(2, 2),	(7, 7),	(5, 5),
(3, 3),	(8, 8),	(3, 3),
(4, 4),	(2, 2),	(6, 6),
(5, 5),	(0, 0),	(9, 9),
(6, 6),	(1, 1),	(6, 6),
(7, 7),	(2, 2),	(1, 1),
(1, 8),	(6, 6),	(7, 7),
(9, 9),	(8, 3),	(5, 5),
(0, 0),	(3, 3),	(4, 4),
(1, 1),	(7, 7),	(4, 4),

(2, 2),	(3, 3),	(7, 7),
(3, 3),	(3, 3),	(2, 2),
(4, 4),	(4, 4),	(2, 2),
(5, 5),	(6, 6),	(5, 5),
(6, 6),	(6, 6),	(7, 7),
(9, 9),	(6, 6),	(8, 9),
(0, 0),	(9, 4),	(5, 5),
(1, 1),	(9, 9),	(9, 4),
(2, 2),	(1, 1),	(4, 4),
(3, 3),	(5, 5),	(5, 9),
(4, 4),	(0, 0),	(0, 0),
(5, 5),	(9, 9),	(8, 8),
(6, 6),	(5, 5),	(9, 9),
(7, 7),	(2, 2),	(8, 8),
(1, 8),	(8, 8),	(0, 0),
(9, 9),	(0, 0),	(1, 1),
(4, 0),	(1, 1),	(2, 2),
(9, 9),	(7, 7),	(3, 3),
(5, 5),	(6, 6),	(4, 4),
(5, 5),	(3, 3),	(5, 5),
(6, 6),	(2, 2),	(6, 6),
(5, 5),	(1, 1),	(7, 7),

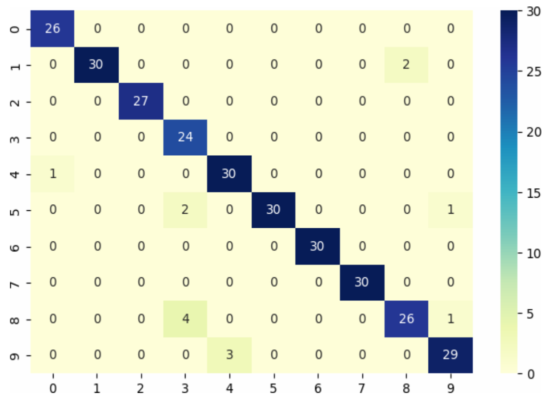
(8, 8),	(0, 0),	(1, 1),
(9, 9),	(2, 2),	(3, 3),
(0, 0),	(2, 2),	(9, 9),
(1, 1),	(7, 7),	(1, 1),
(2, 2),	(8, 8),	(7, 7),
(3, 3),	(2, 2),	(6, 6),
(4, 4),	(0, 0),	(8, 8),
(5, 5),	(1, 1),	(4, 4),
(6, 6),	(2, 2),	(5, 3),
(7, 7),	(6, 6),	(1, 1),
(8, 8),	(8, 3),	(4, 4),
(9, 9),	(8, 3),	(0, 0),
(0, 0),	(7, 7),	(5, 5),
(1, 1),	(5, 3),	(3, 3),
(2, 2),	(3, 3),	(6, 6),
(8, 3),	(4, 4),	(9, 9),
(4, 4),	(6, 6),	(6, 6),
(5, 5),	(6, 6),	(1, 1),
(6, 6),	(6, 6),	(7, 7),
(7, 7),	(4, 4),	(5, 5),
(8, 8),	(9, 9),	(4, 4),
(9, 9),	(1, 1),	(4, 4),
(0, 0),	(5, 5),	(7, 7),
(9, 9),	(0, 0),	(2, 2),
(5, 5),	(9, 9),	(8, 8),
(5, 5),	(5, 5),	(2, 2),

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

Create the Confusion Matric for Performance Evaluation

```
from sklearn.metrics import confusion_matrix
import seaborn as sns

cm = confusion_matrix(predictions, targets[1501:])

conf_matrix = pd.DataFrame(data = cm)
```

```
plt.figure(figsize = (8,5))

sns.heatmap(conf_matrix, annot=True,fmt='d',cmap="YlGnBu");
```



```
cm

array([[26, 0,  0,  0,  0,  0,  0,  0,  0,  0],
      [ 0, 30,  0,  0, 0, 0, 0, 2, 0],
      0,
      [ 0,  0, 27,  0, 0, 0, 0, 0, 0],
      0,
      [ 0,  0,  0, 24, 0, 0, 0, 0, 0],
      0,
      [ 1,  0,  0,  0,   0, 0, 0, 0, 0],
      30,
     [ 0, 0, 0, 2,  0, 30,  0,  0,  0,
                    1],
     [ 0, 0, 0, 0,  0,  0, 30,  0,  0,
                    0],
     [ 0, 0, 0, 0,  0,  0,  0, 30,  0,
                    0],
     [ 0, 0, 0, 4,  0,  0,  0,  0, 26,
                    1],
     [ 0, 0, 0, 0,  3,  0,  0,  0,  0,
                    29]])
```

Print the Classification Report

```
from sklearn.metrics import classification_report

print(classification_report(predictions,

        targets[1501:]))precision recall  f1-score

        support

      0     0.96  1.00     0.98     26
      1     1.00  0.94     0.97     32
      2     1.00  1.00     1.00     27
      3     0.80  1.00     0.89     24
```

```
       4      0.91   0.97      0.94     31
       5      1.00   0.91      0.95     33
       6      1.00   1.00      1.00     30
       7      1.00   1.00      1.00     30
       8      0.93   0.84      0.88     31
       9      0.94   0.91      0.92     32

accuracy                      0.95    296
```

```
  macro avg      0.95    0.96    0.95     296
weighted avg      0.96    0.95    0.95     296
```

Lab Assignment No.	4A
Title	Implement K-Means clustering on Iris.csv dataset. Determine the number of clustersusing the elbow method.Dataset Link: https://www.kaggle.com/datasets/uciml/iris
Roll No.	
Class	BE
Date of Completion	
Subject	Computer Laboratory-II :Quantum AI
Assessment Marks	
Assessor's Sign	

EXPERIMENT NO. 4 A

Aim: Implement K-Means clustering on Iris.csv dataset. Determine the number of clusters using the elbow method.Dataset Link: https://www.kaggle.com/datasets/uciml/iris

Hardware Requirement:

- 6 GB free disk space.
- 2 GB RAM.
- 2 GB of RAM, plus additional RAM for virtual machines.
- 6 GB disk space for the host, plus the required disk space for the virtual machine(s).
- Virtualization is available with the KVM hypervisor
- Intel 64 and AMD64 architectures

Software Requirement:
Jupyter Nootbook/Ubuntu

Theory:

K-means clustering algorithm computes the centroids and iterates until we it finds optimal centroid. It assumes that the number of clusters are already known. It is also called flat clustering algorithm. The number of clusters identified from data by algorithm is represented by „K" in K-means.

In this algorithm, the data points are assigned to a cluster in such a manner that the sum of the squared distance between the data points and centroid would be minimum. It is to be understood that less variation within the clusters will lead to more similar data points within same cluster.

Working of K-Means Algorithm

We can understand the working of K-Means clustering algorithm with the help of following steps −

Step 1 − First, we need to specify the number of clusters, K, need to be generated by this algorithm.
Step 2 − Next, randomly select K data points and assign each data point to a cluster. In simple words, classify the data based on the number of data points.
Step 3 − Now it will compute the cluster centroids.
Step 4 − Next, keep iterating the following until we find optimal centroid which is the assignment of data points to the clusters that are not changing any more −

4.1 − First, the sum of squared distance between data points and centroids would be computed.

4.2 − Now, we have to assign each data point to the cluster that is closer than other cluster (centroid).

4.3 − At last compute the centroids for the clusters by taking the average of all data points of that cluster.

K-means follows Expectation-Maximization approach to solve the problem. The Expectation-step is used for assigning the data points to the closest cluster and the Maximization-step is used for computing the centroid of each cluster.

While working with K-means algorithm we need to take care of the following things −

While working with clustering algorithms including K-Means, it is recommended to standardize the data because such algorithms use distance-based measurement to determine the similarity between data points.

Due to the iterative nature of K-Means and random initialization of centroids, K-Means may stick in a local optimum and may not converge to global optimum. That is why it is recommended to use different initializations of centroids

Implementation:

Importing the libraries and the data

```
import pandas as pd # Pandas (version : 1.1.5)
 import numpy as np # Numpy (version : 1.19.2)
 import matplotlib.pyplot as plt # Matplotlib (version :  3.3.2)
 from sklearn.cluster import KMeans # Scikit Learn (version : 0.23.2)
 import seaborn as sns # Seaborn (version : 0.11.1)
 plt.style.use('seaborn')
```

Importing the data from .csv file

First we read the data from the dataset using read_csv from the pandas library.

```
data = pd.read_csv('data\iris.csv')
```

Viewing the data that we imported to pandas dataframe object

```
data
```

```
      Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm  \
0      1            5.1           3.5            1.4           0.2
1      2            4.9           3.0            1.4           0.2
2    3           4.7      3.2            1.3            0.2
3      4            4.6           3.1            1.5           0.2
4      5            5.0           3.6            1.4           0.2
..  ...            ...      ...     ...      ...
145  146            6.7           3.0            5.2           2.3
146  147            6.3           2.5            5.0           1.9
147  148            6.5           3.0            5.2           2.0
148  149            6.2        3.4           5.4            2.3
149  150            5.9           3.0            5.1           1.8

        Species
1     Iris-setosa
2     Iris-setosa
3     Iris-setosa
4     Iris-setosa
5     Iris-setosa
..          ...
145  Iris-virginica
146  Iris-virginica
147  Iris-virginica
148  Iris-virginica
149  Iris-virginica

[150 rows x 6 columns]
```

Viewing and Describing the data

Now we view the Head and Tail of the data using head() and tail() respectively.

data.head()

```
   Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm      Species
```

```
 0  1             5.1            3.5            1.4            0.2  Iris-setosa
 1  2             4.9            3.0            1.4            0.2  Iris-setosa
 2  3             4.7            3.2            1.3            0.2  Iris-setosa
 3  4             4.6            3.1            1.5            0.2  Iris-setosa
 4  5             5.0            3.6            1.4            0.2  Iris-setosa
```

data.tail()

```
       Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm  \
 145  146       6.7            3.0            5.2            2.3
 146  147       6.3            2.5            5.0            1.9
 147  148       6.5            3.0            5.2       2.0
 148  149       6.2            3.4            5.4            2.3
 149  150       5.9            3.0            5.1            1.8


        Species
 145  Iris-virginica
 146  Iris-virginica
 147  Iris-virginica
 148  Iris-virginica
 149  Iris-virginica
```

Checking the sample size of data - how many samples are there in the dataset using len().

```
len(data)
150
```

Checking the dimensions/shape of the dataset using shape.

```
data.shape
(150, 6)
```

Viewing Column names of the dataset using columns

```
data.columns
Index(['Id', 'SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm',
       'Species'],
      dtype='object')
for i,col in enumerate(data.columns):
    print(f'Column number {1+i} is {col}')
Column number 1 is Id
 Column number 2 is SepalLengthCm
 Column number 3 is SepalWidthCm
 Column number 4 is PetalLengthCm
 Column number 5 is PetalWidthCm
 Column number 6 is Species
```

So, our dataset has 5 columns named:

- Id
- SepalLengthCm
- SepalWidthCm
- PetalLengthCm
- PetalWidthCm
- Species.

View datatypes of each column in the dataset using dtype.

```
data.dtypes
Id              int64
 SepalLengthCm    float64
 SepalWidthCm     float64
 PetalLengthCm    float64
 PetalWidthCm     float64
 Species          object
 dtype: object
```

Gathering Further information about the dataset using info()

```
data.info()
<class 'pandas.core.frame.DataFrame'>
 RangeIndex: 150 entries, 0 to 149
 Data columns (total 6 columns):
  #   Column          Non-Null Count  Dtype
 ---  -------         --------------  ------
  1   Id              150 non-null    int64
 2   SepalLengthCm  150 non-null     float64
 3   SepalWidthCm   150 non-null     float64
 4   PetalLengthCm  150 non-null     float64
 5   PetalWidthCm   150 non-null     float64
 6   Species      150 non-null    object
dtypes: float64(4), int64(1), object(1)
memory usage: 7.2+ KB
```

Describing the data as basic statistics using describe()

```
data.describe()
```

```
           Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm
count  150.000000     150.000000     150.000000     150.000000     150.000000
mean    75.500000     5.843333       3.054000       3.758667       1.198667
std     43.445368     0.828066       0.433594       1.764420       0.763161
min      1.000000     4.300000       2.000000       1.000000       0.100000
25%     38.250000     5.100000       2.800000       1.600000       0.300000
50%     75.500000     5.800000       3.000000       4.350000       1.300000
75%    112.750000     6.400000       3.300000       5.100000       1.800000
max    150.000000     7.900000       4.400000       6.900000       2.500000
```

Checking the data for inconsistencies and further cleaning the data if needed.

Checking data for missing values using isnull().

```
data.isnull()
```

```
           Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm  Species
0      False          False          False          False          False   False
1      False          False          False          False          False   False
2      False          False          False      False      False   False
3      False          False          False          False          False   False
4      False          False          False          False          False   False
..      ...     ...     ...     ...     ...     ...
145  False     False          False          False          False   False
146  False     False          False          False          False   False
147  False     False          False          False          False   False
```

```
 148  False      False              False              False              False    False
 149  False      False              False              False              False    False

 [150 rows x 6 columns]
```

Checking summary of missing values

```
data.isnull().sum()
Id                    0
 SepalLengthCm    0
 SepalWidthCm     0
 PetalLengthCm    0
 PetalWidthCm     0
 Species          0
 dtype: int64
```

The 'Id' column has no relevence therefore deleting it would be better.

Deleting 'customer_id' colummn using drop().

```
data.drop('Id', axis=1, inplace=True)
 data.head()
   SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm            Species
 0            5.1            3.5            1.4            0.2  Iris-setosa
 1            4.9            3.0            1.4            0.2  Iris-setosa
 2            4.7            3.2            1.3            0.2  Iris-setosa
 3            4.6            3.1            1.5            0.2  Iris-setosa
 4            5.0            3.6            1.4            0.2  Iris-setosa
```

Modelling

K - Means Clustering

K-means clustering is a clustering algorithm that aims to partition n observations into k clusters. Initialisation – K initial "means" (centroids) are generated at random Assignment – K clusters are created by associating each observation with the nearest centroid Update – The centroid of the clusters becomes the new mean, Assignment and Update are repeated iteratively until convergence The end result is that the sum of squared errors is minimised between points and their respective centroids. We will use KMeans Clustering. At first we will find the optimal clusters based on inertia and using elbow method. The distance between the centroids and the data points should be less.

First we need to check the data for any missing values as it can ruin our model.

```
data.isna().sum()
```

```
SepalLengthCm      0
SepalWidthCm     0
PetalLengthCm    0
PetalWidthCm     0
Species        0
dtype: int64
```

We conclude that we don't have any missing values therefore we can go forward and start the clustering procedure.

We will now view and select the data that we need for clustering.

```
data.head()
```

```
   SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm       Species
0            5.1           3.5            1.4           0.2  Iris-setosa
1            4.9           3.0            1.4           0.2  Iris-setosa
2            4.7           3.2            1.3           0.2  Iris-setosa
3            4.6           3.1            1.5           0.2  Iris-setosa
4            5.0           3.6            1.4           0.2  Iris-setosa
```

Checking the value count of the target column i.e. 'Species' using value_counts()

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

```
Iris-setosa        50
Iris-versicolor    50
```

```
Iris-virginica     50
Name: Species, dtype: int64
```

Splitting into Training and Target data

Target Data

```
target_data = data.iloc[:,4]
target_data.head()
```

```
1          Iris-setosa
 2         Iris-setosa
 3         Iris-setosa
 4         Iris-setosa
 5         Iris-setosa
 Name: Species, dtype: object
```

Training data

```
clustering_data = data.iloc[:,[0,1,2,3]]
clustering_data.head()
```

```
   SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm
 0          5.1           3.5            1.4           0.2
 1          4.9           3.0            1.4           0.2
 2          4.7           3.2            1.3           0.2
 3        4.6             3.1            1.5           0.2
 4          5.0           3.6            1.4           0.2
```

Now, we need to visualize the data which we are going to use for the clustering. This will give us a fair idea about the data we're working on.

```
fig, ax = plt.subplots(figsize=(15,7))
 sns.set(font_scale=1.5)
 ax = sns.scatterplot(x=data['SepalLengthCm'],y=data['SepalWidthCm'], s=70, color='#f73434',
edgecolor='#f73434', linewidth=0.3)
 ax.set_ylabel('Sepal Width (in cm)')
 ax.set_xlabel('Sepal Length (in cm)')
 plt.title('Sepal Length vs Width', fontsize = 20)
 plt.show()
```

This gives us a fair Idea and patterns about some of the data.

Determining No. of Clusters Required

The Elbow Method

The Elbow method runs k-means clustering on the dataset for a range of values for k (say from 1-10) and then for each value of k computes an average score for all clusters. By default, the distortion score is computed, the sum of square distances from each point to its assigned center.

When these overall metrics for each model are plotted, it is possible to visually determine the best value for k. If the line chart looks like an arm, then the “elbow” (the point of inflection on the curve) is the best value of k. The “arm” can be either up or down, but if there is a strong inflection point, it is a good indication that the underlying model fits best at that point.

We use the Elbow Method which uses Within Cluster Sum Of Squares (WCSS) against the the number of clusters (K Value) to figure out the optimal number of clusters value. WCSS measures sum of distances of observations from their cluster centroids which is given by the below formula.

formula

where Yi is centroid for observation Xi. The main goal is to maximize number of clusters and in limiting case each data point becomes its own cluster centroid.

With this simple line of code we get all the inertia value or the within the cluster sum of square.

```
from sklearn.cluster import KMeans
 wcss=[]
 for i in range(1,11):
        km = KMeans(i)
    km.fit(clustering_data)
    wcss.append(km.inertia_)
 np.array(wcss)
```

```
array([680.8244      , 152.36870648,  78.94084143,  57.31787321,
        46.53558205,  38.93096305,  34.29998554,  30.21678683,
        28.23999745,  25.95204113])
```

Inertia can be recognized as a measure of how internally coherent clusters are.

Now, we visualize the Elbow Method so that we can determine the number of optimal clusters for our dataset.

```
fig, ax = plt.subplots(figsize=(15,7))
 ax = plt.plot(range(1,11),wcss, linewidth=2, color="red", marker ="8")
 plt.axvline(x=3, ls='--')
 plt.ylabel('WCSS')
 plt.xlabel('No. of Clusters (k)')
 plt.title('The Elbow Method', fontsize = 20)
 plt.show()
```

It is clear, that the optimal number of clusters for our data are 3, as the slope of the curve is not steep enough after it. When we observe this curve, we see that last elbow comes at k = 3, it would be difficult to visualize the elbow if we choose the higher range.

Clustering

Now we will build the model for creating clusters from the dataset. We will use n_clusters = 3 i.e. 3 clusters as we have determined by the elbow method, which would be optimal for our dataset.

Our data set is for unsupervised learning therefore we will use fit_predict() Suppose we were working with supervised learning data set we would use fit_tranform()

```
from sklearn.cluster import KMeans
```

```
 kms = KMeans(n_clusters=3, init='k-means++')
 kms.fit(clustering_data)
```

```
KMeans(n_clusters=3)
```

Now that we have the clusters created, we will enter them into a different column

```
clusters = clustering_data.copy()
 clusters['Cluster_Prediction'] = kms.fit_predict(clustering_data)
 clusters.head()
```

```
   SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm  \
 0           5.1           3.5            1.4           0.2
 1           4.9           3.0            1.4           0.2
 2           4.7           3.2            1.3           0.2
 3           4.6           3.1            1.5           0.2
 4           5.0           3.6            1.4           0.2


    Cluster_Prediction
 0                   1
 1                   1
 2                   1
 3                   1
 4                   1
```

We can also get the centroids of the clusters by the cluster_centers_ attribute of KMeans algorithm.

```
kms.cluster_centers_
```

```
array([[5.9016129 , 2.7483871 , 4.39354839, 1.43387097],
       [5.006     , 3.418     , 1.464     , 0.244     ],
       [6.85      , 3.07368421, 5.74210526, 2.07105263]])
```

Now we have all the data we need, we just need to plot the data. We will plot the data using scatterplotwhich will allow us to observe different clusters in different colours.

```
fig, ax = plt.subplots(figsize=(15,7))
 plt.scatter(x=clusters[clusters['Cluster_Predictio
 n'] == 0]['SepalLengthCm'],
            y=clusters[clusters['Cluster_Prediction']
            == 0]['SepalWidthCm'],
            s=70,edgecolor='teal', linewidth=0.3,
            c='teal', label='Iris-versicolor')



 plt.scatter(x=clusters[clusters['Cluster_Prediction']
            == 1]['SepalLengthCm'],
            y=clusters[clusters['Cluster_Prediction']
            == 1]['SepalWidthCm'],
            s=70,edgecolor='lime', linewidth=0.3,
            c='lime', label='Iris-setosa')



 plt.scatter(x=clusters[clusters['Cluster_Prediction'] ==
            2]['SepalLengthCm'], y=clusters[clusters['Cluster_Prediction']
            == 2]['SepalWidthCm'], s=70,edgecolor='magenta',
            linewidth=0.3, c='magenta', label='Iris-virginica')

 plt.scatter(x=kms.cluster_centers_[:, 0], y=kms.cluster_centers_[:, 1], s = 170, c
= 'yellow', label ='Centroids',edgecolor='black', linewidth=0.3)
 plt.legend(loc='upper
 r
 i
```

g

h

t

'

)

p

l

t

.

x

l

i

m

(

4

,

8

)

p

l

t

.

y

l

i

m

(

1

.

8

```
,
4
.
5
)
ax.set_ylabel('Sepal
Width
(in
cm)')
ax.set_x
label('Se
pal
Length
(in cm)')
plt.title('
Clusters
',
fontsize
= 20)
plt.show
()
```

Lab Assignment No.	5B
Title	Use different voting mechanism and Apply AdaBoost (Adaptive Boosting), Gradient Tree Boosting (GBM), XGBoost classification on Iris dataset and compare the performance of three models using different evaluation measures. Dataset Link https://www.kaggle.com/datasets/uciml/iris
Roll No.	
Class	BE
Date of Completion	
Subject	Computer Laboratory-II :Quantum AI
Assessment Marks	
Assessor's Sign	

EXPERIMENT NO. 5 B

Aim: Use different voting mechanism and Apply AdaBoost (Adaptive Boosting), Gradient TreeBoosting (GBM), XGBoost classification on Iris dataset and compare the performance of three models using different evaluation measures. Dataset Link https://www.kaggle.com/datasets/uciml/iris

Hardware Requirement:

- 6 GB free disk space.
- 2 GB RAM.
- 2 GB of RAM, plus additional RAM for virtual machines.
- 6 GB disk space for the host, plus the required disk space for the virtual machine(s).
- Virtualization is available with the KVM hypervisor
- Intel 64 and AMD64 architectures

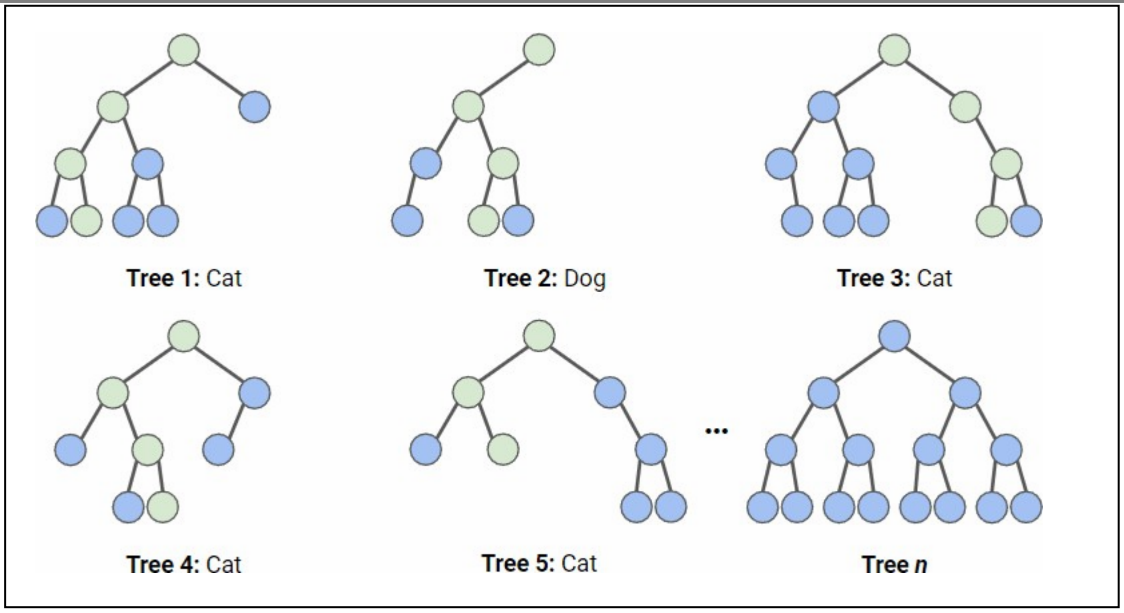
Software Requirement:
Jypiter Nootbook/Ubuntu

Theory:

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

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

In the diagram below, we have a random forest with n decision trees, and we‟ve shown the first 5, along with their predictions (either “Dog” or “Cat”). Each tree is exposed to a different number of features and a different sample of the original dataset, and as such, every tree can be different. Each tree makes a prediction. Looking at the first 5trees, we can see that 4/5 predicted the sample was a Cat. The green circles indicate a hypothetical path the tree took to reach its decision. The random forest would count the number of predictions from decision trees for Cat andfor Dog, and choose the most popular prediction.



Implementation:

```
import pandas as pd
from sklearn.datasets import load_digitsdigits =
load_digits()

dir(digits)

['DESCR', 'data', 'feature_names', 'frame', 'images', 'target', 'target_names']

%matplotlib inline
import matplotlib.pyplot as plt

plt.gray()
for i in range(4): plt.matshow(digits.images[i])

<Figure size 640x480 with 0 Axes>

df = pd.DataFrame(digits.data)df.head()

0                 1    2    3       4    5    6       7       8    9 ...  54   55   56  \
          0 0.0  0.0  5.0  13.0   9.0  1.0  0.0  0.0  0.0  0.0  ... 0.0  0.0  0.0
          1 0.0  0.0  0.0  12.0  13.0 5.0  0.0  0.0  0.0  0.0  ... 0.0  0.0  0.0
          2 0.0  0.0  0.0 4.0  15.0  12.0  0.0  0.0  0.0  0.0  ... 5.0  0.0  0.0
          3 0.0  0.0  7.0 15.0  13.0  1.0  0.0  0.0  0.0  8.0  ...  9.0  0.0  0.0
          4 0.0  0.0  0.0 1.0  11.0  0.0  0.0  0.0  0.0  0.0  ...  0.0  0.0  0.0
```

```
              57   58    59    60       61  62   63
         0  0.0 6.0  13.0  10.0  0.0  0.0  0.0
         1  0.0 0.0  11.0  16.0  10.0  0.0  0.0
         2  0.0 0.0  3.0  11.0  16.0  9.0  0.0
         3  0.0 7.0  13.0  13.0  9.0  0.0  0.0
         4  0.0 0.0  2.0  16.0  4.0  0.0  0.0
```

[5 rows x 64 columns] df['target'] =

digits.targetdf[0:12]

```
0                    1    2    3      4       5    6   7       8    9 ...  55   56   57  \
         0  0.0   0.0   5.0 13.0   9.0    1.0  0.0  0.0  0.0  0.0  ... 0.0  0.0  0.0
         1  0.0   0.0   0.0  12.0  13.0    5.0    0.0 0.0  0.0  0.0  ...  0.0  0.0  0.0
         2  0.0   0.0   0.0  4.0  15.0   12.0    0.0 0.0  0.0  0.0  ...  0.0  0.0  0.0
         3  0.0   0.0  7.0  15.0  13.0    1.0    0.0 0.0  0.0  8.0  ...  0.0  0.0  0.0
         4  0.0   0.0   0.0  1.0  11.0    0.0  0.0  0.0 0.0  0.0  ...  0.0  0.0  0.0
         5  0.0   0.0  12.0  10.0   0.0    0.0    0.0  0.0  0.0  0.0  ... 0.0  0.0  0.0
         6  0.0   0.0   0.0 12.0  13.0    0.0    0.0  0.0  0.0  0.0  ... 0.0  0.0  0.0
         7  0.0   0.0   7.0  8.0  13.0   16.0 15.0  1.0  0.0  0.0  ...  0.0  0.0  0.0
         8  0.0   0.0   9.0 14.0   8.0    1.0  0.0  0.0  0.0  0.0  ... 0.0  0.0  0.0
         9  0.0   0.0  11.0  12.0   0.0    0.0    0.0  0.0  0.0  2.0  ... 0.0  0.0  0.0
10  0.0  0.0   1.0   9.0  15.0  11.0   0.0  0.0  0.0  0.0  ...  0.0  0.0  0.0
11  0.0  0.0   0.0   0.0  14.0  13.0   1.0  0.0  0.0  0.0  ...  0.0  0.0  0.0
```

```
        58     59    60         61   62    63    target
0      6.0     13.0   10.0   0.0 0.0  0.0            0
1      0.0     11.0  16.0  10.0  0.0  0.0     1
2      0.0     3.0  11.0  16.0  9.0  0.0      2
3      7.0     13.0  13.0  9.0  0.0  0.0      3
4      0.0     2.0  16.0  4.0  0.0  0.0      4
5      9.0     16.0  16.0  10.0  0.0  0.0     5
6      1.0     9.0  15.0  11.0  3.0  0.0      6
7  13.0   5.0   0.0   0.0  0.0  0.0          7
8  11.0  16.0  15.0  11.0  1.0  0.0         8
9          9.0  12.0  13.0  3.0  0.0  0.0     9
10  1.0  10.0  13.0   3.0  0.0  0.0         0
11  0.0   1.0  13.0  16.0  1.0  0.0         1
```

[12 rows x 65 columns]

Train and the model and prediction

```
X = df.drop('target',axis='columns')
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=0.2)

from sklearn.ensemble import
RandomForestClassifiermodel =
RandomForestClassifier(n_estimators=20)
model.fit(X_train, y_train)

RandomForestClassifier(n_estimators=20)

model.score(X_test, y_test)

0.9805555555555555

y_predicted = model.predict(X_test)
```

Confusion Matrix

```
from sklearn.metrics import
confusion_matrixcm =
confusion_matrix(y_test,
y_predicted) cm

array([[32, 0,  0,  0,  0,  0,  0,  0,  0,  0],
                                        [ 0, 30,  0,  0,  0,  0,  0,  0,  0, 0],
                                        [ 0,  0, 32,  0,  0,  0,  0,  0,  0, 0],
                                        [ 0,  0,  0, 37,  0,  0,  0,  0,  0, 0],
                                        [ 0,  0,  0,  0, 35,  0,  0,  0,  0, 0],
[ 0,  0,  0,  0,  0, 41,  1,  0,  0,  1],
[ 0,  0,  0,  0,  1,  0, 35,  0,  0,  0],
[ 0,  0,  0,  0,  0,  0,  0, 52,  0,  2],
[ 1,  0,  0,  0,  0,  0,  0,  0, 32,  0],
[ 0,  0,  0,  0,  1,  0,  0,  0,  0, 27]])

%matplotlib inline
import
matplotlib.pyplot
as pltimport
seaborn as sn
plt.figure(figsize=
(10,7))
sn.heatmap(cm,
annot=True)
plt.xlabel('Predicte
d')
plt.ylabel('Truth')

Text(95.72222222222221, 0.5, 'Truth')
```

EXPERIMENT NO. 6 C

- **Aim**: Build a Tic-Tac-Toe game using reinforcement learning in Python by using following tasks
  a. Setting up the environment
  b. Defining the Tic-Tac-Toe game
  c. Building the reinforcement learning model
  d. Training the model
  e. Testing the model

- **Hardware Requirement:**

▪ 6 GB free disk space.
▪ 2 GB RAM.
▪ 2 GB of RAM, plus additional RAM for virtual machines.
▪ 6 GB disk space for the host, plus the required disk space for the virtual machine(s).
▪ Virtualization is available with the KVM hypervisor
▪ Intel 64 and AMD64 architectures

- **Software Requirement:**
  Jypiter Nootbook/Ubuntu

- **Theory:**

In reinforcement learning, developers devise a method of rewarding desired behaviors and punishing negative behaviors. This method assigns positive values to the desired actions to encourage the agent and negative values to undesired behaviors. This programs the agent to seek long-term and maximum overall reward to achieve an optimal solution.

These long-term goals help prevent the agent from stalling on lesser goals. With time, the agent learns to avoid the negative and seek the positive. This learning method has been adopted in artificial intelligence (AI) as a way of directing unsupervised machine learning through rewards and penalties.

Common reinforcement learning algorithms

Rather than referring to a specific algorithm, the field of reinforcement learning is made up of several algorithms that take somewhat different approaches. The differences are mainly due to their strategies for exploring their environments.

- State-action-reward-state-action (SARSA). This reinforcement learning algorithm starts by giving the agent what's known as a *policy*. The policy is essentially a probability that tells it the odds of certain actions resulting in rewards, or beneficial states.

- Q-learning. This approach to reinforcement learning takes the opposite approach. The agent receives no policy, meaning its exploration of its environment is more self-directed.
- Deep Q-Networks. These algorithms utilize neural networks in addition to reinforcement learning techniques. They utilize the self-directed environment exploration of reinforcement learning. Future actions are based on a random sample of past beneficial actions learned by the neural network.

Implementation:

```
import numpy as np

class TicTacToeEnvironment:
      def_init_(self):
      self.state = [0] * 9
      self.is_terminal = False

      def reset(self):
      self.state = [0] * 9
      self.is_terminal = False

      def get_available_moves(self):
      return [i for i, mark in enumerate(self.state) if mark == 0]

      def make_move(self, move, player_mark):
      self.state[move] = player_mark

      def check_win(self, player_mark):
     winning_states = [
      [0, 1, 2], [3, 4, 5], [6, 7, 8],  # rows
      [0, 3, 6], [1, 4, 7], [2, 5, 8],  # columns
      [0, 4, 8], [2, 4, 6]  # diagonals
      ]
      for state_indices in winning_states:
      if all(self.state[i] == player_mark for i in state_indices):
          self.is_terminal = True
          return True
      return False
```

```
        def is_draw(self):
        return 0 not in self.state

class QLearningAgent:
        def_init_(self, learning_rate=0.9, discount_factor=0.9, exploration_rate=0.3):
        self.learning_rate = learning_rate
        self.discount_factor = discount_factor
        self.exploration_rate = exploration_rate
        self.q_table = np.zeros((3**9, 9))

        def get_state_index(self, state):
      state_index = 0
        for i, mark in enumerate(state):
         state_index += (3 ** i) * (mark + 1)
        return state_index

        def choose_action(self, state, available_moves):
      state_index = self.get_state_index(state)
        if np.random.random() < self.exploration_rate:
        return np.random.choice(available_moves)
        else:
        return np.argmax(self.q_table[state_index, available_moves])

        def update_q_table(self, state, action, next_state, reward):
      state_index = self.get_state_index(state)
      next_state_index = self.get_state_index(next_state) if next_state is not None else None
      max_q_value = np.max(self.q_table[next_state_index]) if next_state is not None else 0
        self.q_table[state_index, action] = (1 - self.learning_rate) * self.q_table[state_index, action] + \
                                self.learning_rate * (reward + self.discount_factor * max_q_value)

def evaluate_agents(agent1, agent2, num_episodes=1000):
    environment = TicTacToeEnvironment()
    agent1_wins = 0
    agent2_wins = 0
        draws = 0

        for _ in range(num_episodes):
      environment.reset()
      current_agent = agent1
        while not environment.is_terminal:
         available_moves = environment.get_available_moves()
```

```
            current_state = environment.state.copy()
            action = current_agent.choose_action(current_state, available_moves)
            environment.make_move(action, 1 if current_agent == agent1 else -1)

            if environment.check_win(1 if current_agent == agent1 else -1):
                current_agent.update_q_table(current_state, action, None, 10)
                if current_agent == agent1:
                    agent1_wins += 1
                else:
                    agent2_wins += 1
                break
            elif environment.is_draw():
                current_agent.update_q_table(current_state, action, None, 0)
                draws += 1
                break

            next_state = environment.state.copy()
            reward = 0
            if environment.check_win(1 if current_agent == agent1 else -1):
                reward = -10
            current_agent.update_q_table(current_state, action, next_state, reward)

            current_agent = agent2 if current_agent == agent1 else agent1

        return agent1_wins, agent2_wins, draws

# Create agents
agent1 = QLearningAgent()
agent2 = QLearningAgent()

# Evaluate agents
agent1_wins, agent2_wins, draws = evaluate_agents(agent1, agent2)

# Print results
print(f"Agent 1 wins: {agent1_wins}")
print(f"Agent 2 wins: {agent2_wins}")
print(f"Draws: {draws}")
```

```
Agent 1 wins: 458
Agent 2 wins: 470
Draws: 72
```

TicTacToeEnvironment:

This class represents the Tic-Tac-Toe game environment. It maintains the current state of the game,checks for a win or draw, and provides methods to reset the game and make moves.

The__init___method initializes the game state and sets the terminal flag to False.The reset method resets the game state and the terminal flag.

The get_available_moves method returns a list of indices representing the available moves in the currentgame state.

The make_move method updates the game state by placing a player's mark at the specified move index.The check_win method checks if a player has won the game by examining the current state.

The is_draw method checks if the game has ended in a draw.

QLearningAgent:

This class represents the Q-learning agent. It learns to play Tic-Tac-Toe by updating a Q-table based onthe rewards received during gameplay.

The___init___method initializes the learning rate, discount factor, exploration rate, and the Q-table.

The get_state_index method converts the current game state into a unique index for indexing the Q-table.

The choose_action method selects the action (move) to be taken based on the current game state and theexploration-exploitation tradeoff using the epsilon-greedy policy.

The update_q_table method updates the Q-table based on the current state, action, next state, and thereward received.

evaluate_agents:

This function performs the evaluation of two Q-learning agents by playing multiple episodes of Tic-Tac-Toegames.

It takes the two agents and the number of episodes to play as input.

In each episode, the environment is reset, and the agents take turns making moves until the game is over(either a win or a draw).

The agents update their Q-tables based on the rewards received during the episode. The function keeps track of the wins and draws for each agent and returns the counts.

Main code:

The main code creates two Q-learning agents, agent1 and agent2, using the QLearningAgent class.

The evaluate_agents function is called to evaluate the agents by playing a specified number of episodes.The results (number of wins and draws) for each agent are printed.

The Q-learning algorithm involves the following steps:

The agents choose their moves based on the current game state and the exploration-exploitation policy.The environment updates the game state based on the chosen moves.
The environment checks if the game has ended (win or draw). The agents update their Q-tables based on the rewards received.
The agents continue playing until the specified number of episodes is completed.