Student Name: Sawera Fazal_____Roll No:21A-026-se_____Section: 21A_____

CS334 - Machine Learning

Lab 01

Instructor: Ms. Maham Ashraf
E-mail: mashraf@uit.edu

Semester: Fall, 2023

Objective

The purpose of this lab session is to introduce data preparation techniques for Machine Learning(ML) projects.

Instructions

You have to perform the following tasks yourselves. Raise your hand if you face any difficulty in understanding and solving these tasks. **Plagiarism** is an abhorrent practice and you should not engage in it.

How to Submit

- Submit lab work in a single .py file on Microsoft Teams. (No other format will be accepted)
- Lab work file name should be saved with your roll number (e.g. 19a-001-SE_LW01.py)
- Submit home work in a single .py file on Microsoft Teams. (No other format will be accepted)
- Home work file name should be saved with your roll number (e.g. 19a-001-SE_HW01.py)

1 Task 1 - Load Data From CSV

In this tutorial you will discover how to load your data in Python from scratch, including:

1. How to load a CSV file.
2. How to convert strings from a file to floating point numbers.
3. How to convert class values from a file to integers.

1.1 Load CSV file

The standard file format for small datasets is Comma Separated Values or CSV. In its simplest form, CSV files are comprised of rows of data. Each row is divided into columns using a comma (,).
Dataset 1: The *Pima Indians Diabetes* Dataset involves the prediction of the onset of diabetes within 5 years. Download the dataset and save it into your current working directory with the filename pima-indians-diabetes.csv ().
Dataset 2: Iris Flower Species Dataset involves the prediction of iris flower species. Download the dataset and save it into your current working directory with the filename iris.csv (`https://www.kaggle.com/uciml/iris`).

```
# Example of loading Pima Indians CSV dataset
from csv import reader

# Load a CSV file
  def load_csv ( filename ):
    file = open (filename , "r")
    lines = reader ( file )
    dataset = list ( lines )
    return dataset

# Load dataset
filename = 'pima-indians-diabetes.csv'
dataset = load_csv (filename)
print ('Loaded data file {0} with {1} rows and {2} columns'.format (filename ,
    len (dataset), len (dataset [0])))
```

A limitation of this function is that it will load empty lines from data files and add them to our list of rows. We can overcome this by adding rows of data one at a time to our dataset and skipping empty rows. Below is the updated example with this new improved version of the load csv() function.

```
# Example of loading Pima Indians CSV dataset
from csv import reader

# Load a CSV file
  def load_csv (filename):
    dataset = list ()
    with open (filename , 'r') as file :
      csv_reader = reader (file)
      for row in csv_reader :
        if not row:
          continue
        dataset.append (row)
    return dataset
# Load dataset
filename = 'pima-indians-diabetes.csv'
dataset = load_csv (filename)
print ('Loaded data file {0} with {1} rows and {2} columns'.format (filename ,
    len (dataset), len (dataset [0])))
```

Listing 2: Improved Example of Loading the Pima Indians Diabetes Dataset CSV File.

Task 1: Load iris.csv file

Download *iris.csv* file and load it into your system. Ignore empty lines if the are present in the dataset.

2 Convert String to Floats

Most, if not all machine learning algorithms prefer to work with numbers. Specifically, floating point numbers are preferred. Our code for loading a CSV file returns a dataset as a list of lists, but each value is a string.

We can write a small function to convert specific columns of our loaded dataset to floating point values. Below is this function called str column to float(). It will convert a given column in the dataset to oating point values, careful to strip any whitespace from the value before making the conversion.

```
def str_column_to_float (dataset , column ):
  for row in dataset :
    row [column ] = float (row [column ].strip ())
```

Listing 3: Function For Converting String Data To Floats.

Task 2: Test your data

After converting string data into float values, test and verify your data for its successful conversion.

Table 1: Iris dataset

```
5.1,3.5,1.4,0.2,Iris-setosa
4.9,3.0,1.4,0.2,Iris-setosa
4.7,3.2,1.3,0.2,Iris-setosa
4.6,3.1,1.5,0.2,Iris-setosa
5.0,3.6,1.4,0.2,Iris-setosa
```

3 Convert String to Integers

The iris owers dataset is like the Pima Indians dataset, in that the columns contain numeric data. The difference is the final column, traditionally used to hold the outcome or value to be predicted for a given row. The final column in the iris flowers data is the iris flower species as a string. For example, in Table 1, the first 5 rows of the iris dataset contains string values. Some machine learning algorithms prefer all values to be numeric, including the outcome or predicted value. We can convert the class value in the iris flowers dataset to an integer by creating a map.

1. First, we locate all of the unique class values, which happen to be: Iris-setosa, Iris-versicolor and Iris-virginica.
2. Next, we assign an integer value to each, such as: 0, 1 and 2.
3. Finally, we replace all occurrences of class string values with their corresponding integer values.

Below is a function to do just that called str column to int(). Like the previously introduced str column to float() it operates on a single column in the dataset.

```
# Convert string column to integer
def str_column_to_int ( dataset , column ):
  class_values = [row[column] for row in dataset]
  unique = set(class_values)
  lookup = dict()
  for i, value in enumerate(unique):
    lookup[value] = i
  for row in dataset:
    row[column] = lookup[row[column]]
  return lookup
```

Listing 4: Function To Integer Encode String Class Values.

Task 3: Test your function

Test this new function in addition to the previous two functions for loading a CSV file and converting columns to floating point values. It should also return the dictionary mapping of class values to integer values, in case any users downstream want to convert predictions back to string values again. Running this example produces the output below. We can see the first row of the dataset before and after the data type conversions. We can also see the dictionary mapping of class values to integers.

```
Loaded data file iris.csv with 150 rows and 5 columns
['5.1', '3.5', '1.4', '0.2', 'Iris-setosa']
[5.1, 3.5, 1.4, 0.2, 1]
{'Iris-virginica': 0, 'Iris-setosa': 1, 'Iris-versicolor': 2}
```

4 Exercises (Homework)

You learned how to load CSV files and perform basic data conversions. Data loading can be a difficult task given the variety of data cleaning and conversion that may be required from problem to problem. There are many extensions that you could make to make these examples more robust to new and different data files. Below are just a few ideas That you to implement yourself and submit as homework file on MS Teams:

- Detect and remove empty lines at the top or bottom of the file.

- Detect and handle missing values in a column.
- Detect and handle rows that do not match expectations for the rest of the file.
- Support for other delimiters such as pipe (|) or white space.
- Support more efficient data structures such as arrays.

Two libraries you may wish to use in practice for loading CSV data are NumPy and Pandas. NumPy offers the loadtxt()[1] function for loading data files as NumPy arrays. Pandas offers the read csv()[2] function that offers a lot of flexibility regarding data types, file headers and more.

Task 1

```
@author: Sawera Fazal
"""

from csv import reader
def load_csv(filename):
    #This line opens the specified CSV file in read-only mode ("r")
    file=open(filename, "r")
    #This reader function is part of the csv module inn python
    lines = reader(file)
    dataset = list(lines)
    return dataset

filename = 'Iris.csv'
dataset = load_csv(filename)
print("Loaded data file {0} with {1} rows and {2} columns".format(filename, len(dataset), len(datas

#print the first 5 data rows
for row in dataset[0:6]:#Skip the header and print the next 5 rows
    print(row)
```

OUTPUT

```
Loaded data file Iris.csv with 151 rows and 6 columns
['Id', 'SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm', 'Species']
['1', '5.1', '3.5', '1.4', '0.2', 'Iris-setosa']
['2', '4.9', '3', '1.4', '0.2', 'Iris-setosa']
['3', '4.7', '3.2', '1.3', '0.2', 'Iris-setosa']
['4', '4.6', '3.1', '1.5', '0.2', 'Iris-setosa']
['5', '5', '3.6', '1.4', '0.2', 'Iris-setosa']
```

Task 2

```
@author: Sawera Fazal
"""

from csv import reader

def load_csv(filename):
    dataset = list()
    with open (filename, "r") as file:
        csv_reader = reader (file)
        for row in csv_reader:
            if not row:
                continue
            dataset.append(row)
        return dataset


filename = 'Iris.csv'
dataset = load_csv(filename)
print("Loaded data file {0} with {1} rows and {2} columns".format(filename, len(dataset), len(dataset[0])))

#print the first 5 data rows
for row in dataset[0:15]:#Skip the header and print the next 5 rows
    print(row)
```

OUTPUT

```
Loaded data file Iris.csv with 151 rows and 6 columns
['Id', 'SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm'
'Species']
['1', '5.1', '3.5', '1.4', '0.2', 'Iris-setosa']
['2', '4.9', '3', '1.4', '0.2', 'Iris-setosa']
['3', '4.7', '3.2', '1.3', '0.2', 'Iris-setosa']
['4', '4.6', '3.1', '1.5', '0.2', 'Iris-setosa']
['5', '5', '3.6', '1.4', '0.2', 'Iris-setosa']
['6', '5.4', '3.9', '1.7', '0.4', 'Iris-setosa']
['7', '4.6', '3.4', '1.4', '0.3', 'Iris-setosa']
['8', '5', '3.4', '1.5', '0.2', 'Iris-setosa']
['9', '4.4', '2.9', '1.4', '0.2', 'Iris-setosa']
['10', '4.9', '3.1', '1.5', '0.1', 'Iris-setosa']
['11', '5.4', '3.7', '1.5', '0.2', 'Iris-setosa']
['12', '4.8', '3.4', '1.6', '0.2', 'Iris-setosa']
['13', '4.8', '3', '1.4', '0.1', 'Iris-setosa']
['14', '4.3', '3', '1.1', '0.1', 'Iris-setosa']
```

Task 3

```
@author: Sawera fazal
"""

from csv import reader

def load_csv(filename):
    dataset = list()
    with open (filename, "r") as file:
        csv_reader = reader (file)
        for row in csv_reader:
            if not row:
                continue
            dataset.append(row)
        return dataset

def str_column_to_int(dataset, column_name):
    column_index = dataset[0].index(column_name)
    class_values = set()
    for row in dataset[1:]:
        class_values.add(row[column_index])

    class_mapping = {value: index for index, value in enumerate(class_values)}

    for row in dataset[1:]:
        row[column_index] = class_mapping[row[column_index]]

    return class_mapping

filename = 'Iris.csv'
dataset = load_csv(filename)
print(dataset[0])

class_mapping = str_column_to_int(dataset, column_name="SepalLengthCm")
for row in dataset[1:10]:
    print(row)

print(class_mapping)

def int_column_to_float(dataset, column_name):
    column_index = dataset[0].index(column_name)
    for row in dataset[1:]:
        row[column_index] = float(row[column_index])


int_column_to_float(dataset, column_name="SepalLengthCm")
print(dataset[0])
for row in dataset[1:10]:
    print(row)
```

OUTPUT

```
['Id', 'SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm'
'Species']
['1', 2, '3.5', '1.4', '0.2', 'Iris-setosa']
['2', 9, '3', '1.4', '0.2', 'Iris-setosa']
['3', 32, '3.2', '1.3', '0.2', 'Iris-setosa']
['4', 13, '3.1', '1.5', '0.2', 'Iris-setosa']
['5', 27, '3.6', '1.4', '0.2', 'Iris-setosa']
['6', 14, '3.9', '1.7', '0.4', 'Iris-setosa']
['7', 13, '3.4', '1.4', '0.3', 'Iris-setosa']
['8', 27, '3.4', '1.5', '0.2', 'Iris-setosa']
['9', 10, '2.9', '1.4', '0.2', 'Iris-setosa']
{'4.3': 0, '5.9': 1, '5.1': 2, '5.8': 3, '7.3': 4, '6.6': 5, '7.1': 6,
'7.2': 7, '5.7': 8, '4.9': 9, '4.4': 10, '4.8': 11, '6.1': 12, '4.6':
13, '5.4': 14, '6.2': 15, '5.5': 16, '7.6': 17, '4.5': 18, '5.6': 19,
'7.7': 20, '5.3': 21, '7': 22, '6.7': 23, '6.9': 24, '6.8': 25, '5.2':
26, '5': 27, '6.5': 28, '6': 29, '7.9': 30, '6.3': 31, '4.7': 32, '6.4'
33, '7.4': 34}
['Id', 'SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm'
'Species']
['1', 2.0, '3.5', '1.4', '0.2', 'Iris-setosa']
['2', 9.0, '3', '1.4', '0.2', 'Iris-setosa']
['3', 32.0, '3.2', '1.3', '0.2', 'Iris-setosa']
['4', 13.0, '3.1', '1.5', '0.2', 'Iris-setosa']
['5', 27.0, '3.6', '1.4', '0.2', 'Iris-setosa']
['6', 14.0, '3.9', '1.7', '0.4', 'Iris-setosa']
['7', 13.0, '3.4', '1.4', '0.3', 'Iris-setosa']
['8', 27.0, '3.4', '1.5', '0.2', 'Iris-setosa']
['9', 10.0, '2.9', '1.4', '0.2', 'Iris-setosa']
```

LAB TASK

```python
@author: Sawera Fazal
"""

import csv
from statistics import mean, median
from collections import Counter

# Open the CSV file
with open('Iris.csv', 'r') as csvfile:
    # Read the CSV file
    reader = csv.DictReader(csvfile)

    # Extract SepalLengthCm values into a list
    sepal_lengths = [float(row['SepalLengthCm']) for row in reader]

# Calculate mean
mean_sepal_length = mean(sepal_lengths)

# Calculate median
median_sepal_length = median(sepal_lengths)

# Calculate mode
sepal_length_counts = Counter(sepal_lengths)
mode_sepal_length = sepal_length_counts.most_common(1)[0][0]

print(f"Mean Sepal Length: {mean_sepal_length:.2f}")
print(f"Median Sepal Length: {median_sepal_length}")
print(f"Mode Sepal Length: {mode_sepal_length}")
```

OUTPUT

```
Mean Sepal Length: 5.84
Median Sepal Length: 5.8
Mode Sepal Length: 5.0
```

HOME TASK 1

```
@author: Sawera Fazal
"""

import pandas as pd

dataset = pd.read_csv("Iris.csv")
data = pd.DataFrame(dataset)

#remove empty lines at the top of the file
df = data.dropna(how="all")

print(df)
```

OUTPUT

```
      Id  SepalLengthCm  ...  PetalWidthCm         Species
0      1            5.1  ...           0.2     Iris-setosa
1      2            4.9  ...           0.2     Iris-setosa
2      3            4.7  ...           0.2     Iris-setosa
3      4            4.6  ...           0.2     Iris-setosa
4      5            5.0  ...           0.2     Iris-setosa
..   ...            ...  ...           ...             ...
145  146            6.7  ...           2.3  Iris-virginica
146  147            6.3  ...           1.9  Iris-virginica
147  148            6.5  ...           2.0  Iris-virginica
148  149            6.2  ...           2.3  Iris-virginica
149  150            5.9  ...           1.8  Iris-virginica

[150 rows x 6 columns]
```

HOME TASK 2

```
@author: Sawera Fazal
"""

import pandas as pd

dataset = pd.read_csv("Iris.csv")
data = pd.DataFrame(dataset)

#detecting column with missing values
missing_values = data.isnull().sum()
print(missing_values)

print("\n")
#Handling column with missing value
# Fill missing values with 0
data.fillna(0, inplace=True)
#again check fro missing values
missing_values = data.isnull().sum()
print(missing_values)
```

OUTPUT

```
Id              0
SepalLengthCm   0
SepalWidthCm    0
PetalLengthCm   1
PetalWidthCm    1
Species         0
dtype: int64


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

HOME TASK 3

```
@author: Saawera Fazal
"""

import pandas as pd

dataset = pd.read_csv("Iris.csv")
data = pd.DataFrame(dataset)

expected_SepalLengthCm_range = (1, 6)
# Detect rows where the 'age' column does not meet expectations
unexpected_rows = data[~data['SepalLengthCm'].between(*expected_SepalLengthCm_range)]
print(unexpected_rows)

# Fill unexpected Sepal length values with a default value (e.g., 2)
filled = data.loc[~data['SepalLengthCm'].between(*expected_SepalLengthCm_range), 'SepalLengthCm'] = 0
unexpected_rows = data[~data['SepalLengthCm'].between(*expected_SepalLengthCm_range)]
print(unexpected_rows)
```

OUTPUT

```
      Id  SepalLengthCm  ...  PetalWidthCm          Species
50    51            7.0  ...           1.4  Iris-versicolor
51    52            6.4  ...           1.5  Iris-versicolor
52    53            6.9  ...           1.5  Iris-versicolor
54    55            6.5  ...           1.5  Iris-versicolor
56    57            6.3  ...           1.6  Iris-versicolor
..   ...            ...  ...           ...              ...
144  145            6.7  ...           2.5   Iris-virginica
145  146            6.7  ...           2.3   Iris-virginica
146  147            6.3  ...           1.9   Iris-virginica
147  148            6.5  ...           2.0   Iris-virginica
148  149            6.2  ...           2.3   Iris-virginica

[61 rows x 6 columns]
      Id  SepalLengthCm  ...  PetalWidthCm          Species
50    51            0.0  ...           1.4  Iris-versicolor
51    52            0.0  ...           1.5  Iris-versicolor
52    53            0.0  ...           1.5  Iris-versicolor
54    55            0.0  ...           1.5  Iris-versicolor
56    57            0.0  ...           1.6  Iris-versicolor
..   ...            ...  ...           ...              ...
144  145            0.0  ...           2.5   Iris-virginica
145  146            0.0  ...           2.3   Iris-virginica
146  147            0.0  ...           1.9   Iris-virginica
147  148            0.0  ...           2.0   Iris-virginica
148  149            0.0  ...           2.3   Iris-virginica
```

HOME TASK 4

```python
@author: Sawera Fazal
"""

import pandas as pd

# Read pipe-delimited file
data_pipe = pd.read_csv('Iris.csv', delimiter='|')
print(data_pipe)
# Read whitespace-separated file (e.g., tab-separated or space-separated)
data_whitespace = pd.read_csv('Iris.csv', sep='\s+')
print(data_whitespace)
```

OUTPUT

```
    Id,SepalLengthCm,SepalWidthCm,PetalLengthCm,PetalWidthCm,Species
0                          1,5.1,3.5,1.4,0.2,Iris-setosa
1                            2,4.9,3,1.4,0.2,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                            5,5,3.6,1.4,0.2,Iris-setosa
..                                                   ...
145                     146,6.7,3,5.2,2.3,Iris-virginica
146                     147,6.3,2.5,5,1.9,Iris-virginica
147                       148,6.5,3,5.2,2,Iris-virginica
148                   149,6.2,3.4,5.4,2.3,Iris-virginica
149                     150,5.9,3,5.1,1.8,Iris-virginica

[150 rows x 1 columns]
    Id,SepalLengthCm,SepalWidthCm,PetalLengthCm,PetalWidthCm,Species
0                          1,5.1,3.5,1.4,0.2,Iris-setosa
1                            2,4.9,3,1.4,0.2,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                            5,5,3.6,1.4,0.2,Iris-setosa
..                                                   ...
145                     146,6.7,3,5.2,2.3,Iris-virginica
146                     147,6.3,2.5,5,1.9,Iris-virginica
147                       148,6.5,3,5.2,2,Iris-virginica
148                   149,6.2,3.4,5.4,2.3,Iris-virginica
149                     150,5.9,3,5.1,1.8,Iris-virginica

[150 rows x 1 columns]
```

HOME TASK 5

```python
@author: Sawera Fazal
"""

import numpy as np

# Read data from a file into a 1D NumPy array (replace delimiter and dtype accordingly)
data_array = np.genfromtxt('Iris.csv', delimiter='|', dtype=int, skip_header=1)

# Reshape the 1D array into a 2D array with one column
data_array = data_array.reshape(-1, 1)
print(data_array)

# Calculate the mean of a specific column in the filtered data
column_index = 0  # Replace this with the actual index of the column you want to calculate the mean for
column_mean = np.mean(data_array[:, column_index])
print(column_mean)
```

OUTPUT

```
 [-1]
 [-1]
 [-1]
 [-1]
 [-1]
 [-1]
 [-1]
 [-1]
 [-1]
 [-1]
 [-1]
 [-1]
 [-1]
 [-1]
 [-1]
 [-1]
 [-1]
 [-1]
 [-1]
 [-1]
 [-1]
 [-1]
 [-1]
 [-1]
 [-1]
 [-1]
 [-1]
 [-1]]
-1.0
```

Student Name: Sawera Fazal_____Roll No:21A-026-se_____Section: 21A_____

CS334 - Machine Learning

Lab 02

Instructor: Dr. Maham Ashraf
E-mail: mashraf@uit.edu

Semester: Fall, 2023

Objective

The purpose of this lab session is to introduce data preparation techniques for Machine Learning(ML) projects.

Instructions

You have to perform the following tasks yourselves. Raise your hand if you face any difficulty in understanding and solving these tasks. **Plagiarism** is an abhorrent practice and you should not engage in it.

How to Submit

- Submit lab work in a single .py file on Microsoft Teams. (No other format will be accepted)
- Lab work file name should be saved with your roll number (e.g. 19a-001-SE_LW01.py)
- Submit home work in a single .py file on Microsoft Teams. (No other format will be accepted)
- Home work file name should be saved with your roll number (e.g. 19a-001-SE_HW01.py)

1 Summarize and Visualize your data

1.1 Data summary

Descriptive statistics is packed with information and insights. A pair of experienced eyes will take a look at every single data point and extract valuable information from the summary table. Let's first see what a table of summary statistics looks like for a given dataset. We will use a built-in dataset that comes with *seaborn* library in Python.

```
import seaborn as sns
import pandas as pd
df = sns.load_dataset('tips')
df.head()
```

Listing 1: Load data in data frame.

Each observation (row) in this dataset represents dining in a restaurant. The columns names here are self-explanatory. Among the numeric columns, 'total_bill' refers to how much bills the diners paid and 'tip' represents the amount of tip they paid.

Just a simple method call `df.describe()` gives you the summary statistics for the numeric columns (I'll touch upon categorical columns towards the end).

Your Tasks

Task 1 Run the example as given in Listing - 1. Print top 10 rows of the data and also print the data summary as given in Figure - 2.

Task 2 Repeat Task 1 on *pima-indians-diabetes.csv* dataset.

Task 3 Generate dot plot and distribution plot on *heart.csv* dataset (https://www.kaggle.com/fedesoriano/heart-failure-prediction?select=heart.csv).

Task 4 Run the example as given in Listing - 3 and review the number of missing values in the dataset before and after the data imputation transform.

Task 5 Repeat Task 4 on *heart.csv* dataset.

```
import pandas as pd
```
[32] ✓ 0.0s

```
data=pd.read_csv("heart.csv")
data
```
[33] ✓ 0.0s

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
298	57	0	0	140	241	0	1	123	1	0.2	1	0	3	0
299	45	1	3	110	264	0	1	132	0	1.2	1	0	3	0
300	68	1	0	144	193	1	1	141	0	3.4	1	2	3	0
301	57	1	0	130	131	0	1	115	1	1.2	1	1	3	0
302	57	0	1	130	236	0	0	174	0	0.0	1	1	2	0

303 rows × 14 columns

```
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```
[34] ✓ 0.0s

```
data['target'].value_counts()
```
[35] ✓ 0.0s

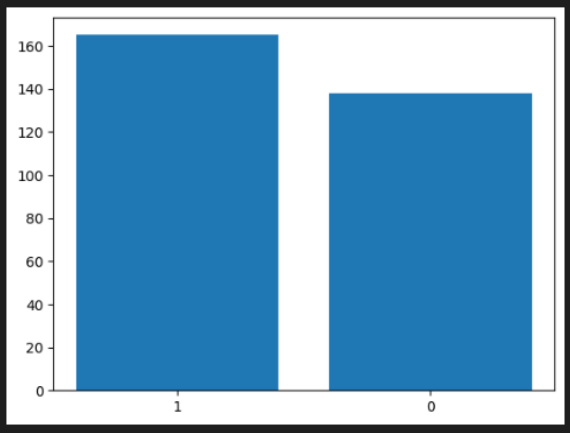
```
target
1    165
0    138
Name: count, dtype: int64
```

```
plt.bar(['1','0',], height=data.target.value_counts())
```
[36] ✓ 0.1s

```
plt.bar(['1','0',], height=data.target.value_counts())
```

[36] ✓ 0.1s

```
<BarContainer object of 2 artists>
```



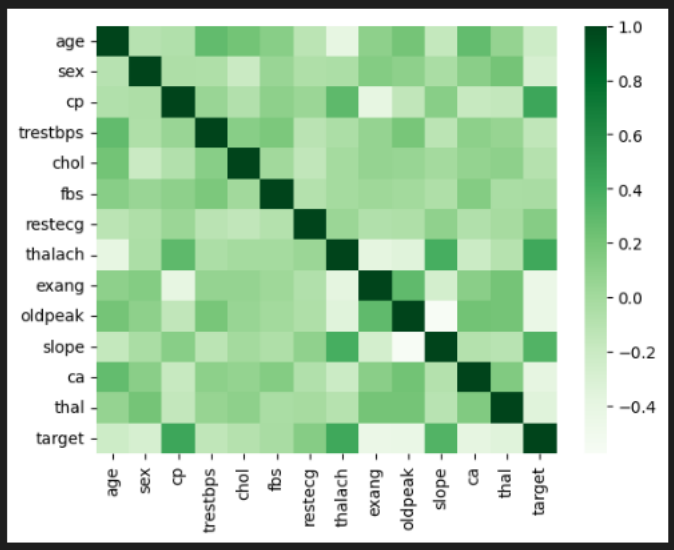

```
data.corr()
```

[37] ✓ 0.0s

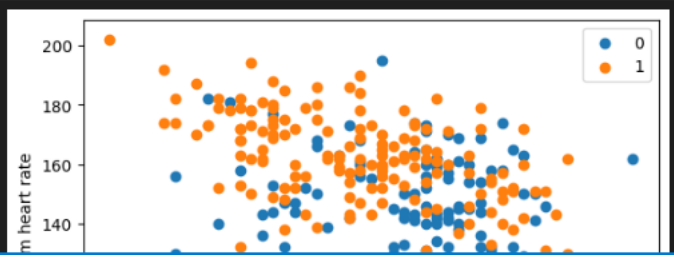
	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
age	1.000000	-0.098447	-0.068653	0.279351	0.213678	0.121308	-0.116211	-0.398522	0.096801	0.210013	-0.168814	0.276326	0.068001	-0.225439
sex	-0.098447	1.000000	-0.049353	-0.056769	-0.197912	0.045032	-0.058196	-0.044020	0.141664	0.096093	-0.030711	0.118261	0.210041	-0.280937
cp	-0.068653	-0.049353	1.000000	0.047608	-0.076904	0.094444	0.044421	0.295762	-0.394280	-0.149230	0.119717	-0.181053	-0.161736	0.433798
trestbps	0.279351	-0.056769	0.047608	1.000000	0.123174	0.177531	-0.114103	-0.046698	0.067616	0.193216	-0.121475	0.101389	0.062210	-0.144931
chol	0.213678	-0.197912	-0.076904	0.123174	1.000000	0.013294	-0.151040	-0.009940	0.067023	0.053952	-0.004038	0.070511	0.098803	-0.085239
fbs	0.121308	0.045032	0.094444	0.177531	0.013294	1.000000	-0.084189	-0.008567	0.025665	0.005747	-0.059894	0.137979	-0.032019	-0.028046
restecg	-0.116211	-0.058196	0.044421	-0.114103	-0.151040	-0.084189	1.000000	0.044123	-0.070733	-0.058770	0.093045	-0.072042	-0.011981	0.137230
thalach	-0.398522	-0.044020	0.295762	-0.046698	-0.009940	-0.008567	0.044123	1.000000	-0.378812	-0.344187	0.386784	-0.213177	-0.096439	0.421741
exang	0.096801	0.141664	-0.394280	0.067616	0.067023	0.025665	-0.070733	-0.378812	1.000000	0.288223	-0.257748	0.115739	0.206754	-0.436757
oldpeak	0.210013	0.096093	-0.149230	0.193216	0.053952	0.005747	-0.058770	-0.344187	0.288223	1.000000	-0.577537	0.222682	0.210244	-0.430696
slope	-0.168814	-0.030711	0.119717	-0.121475	-0.004038	-0.059894	0.093045	0.386784	-0.257748	-0.577537	1.000000	-0.080155	-0.104764	0.345877
ca	0.276326	0.118261	-0.181053	0.101389	0.070511	0.137979	-0.072042	-0.213177	0.115739	0.222682	-0.080155	1.000000	0.151832	-0.391724
thal	0.068001	0.210041	-0.161736	0.062210	0.098803	-0.032019	-0.011981	-0.096439	0.206754	0.210244	-0.104764	0.151832	1.000000	-0.344029

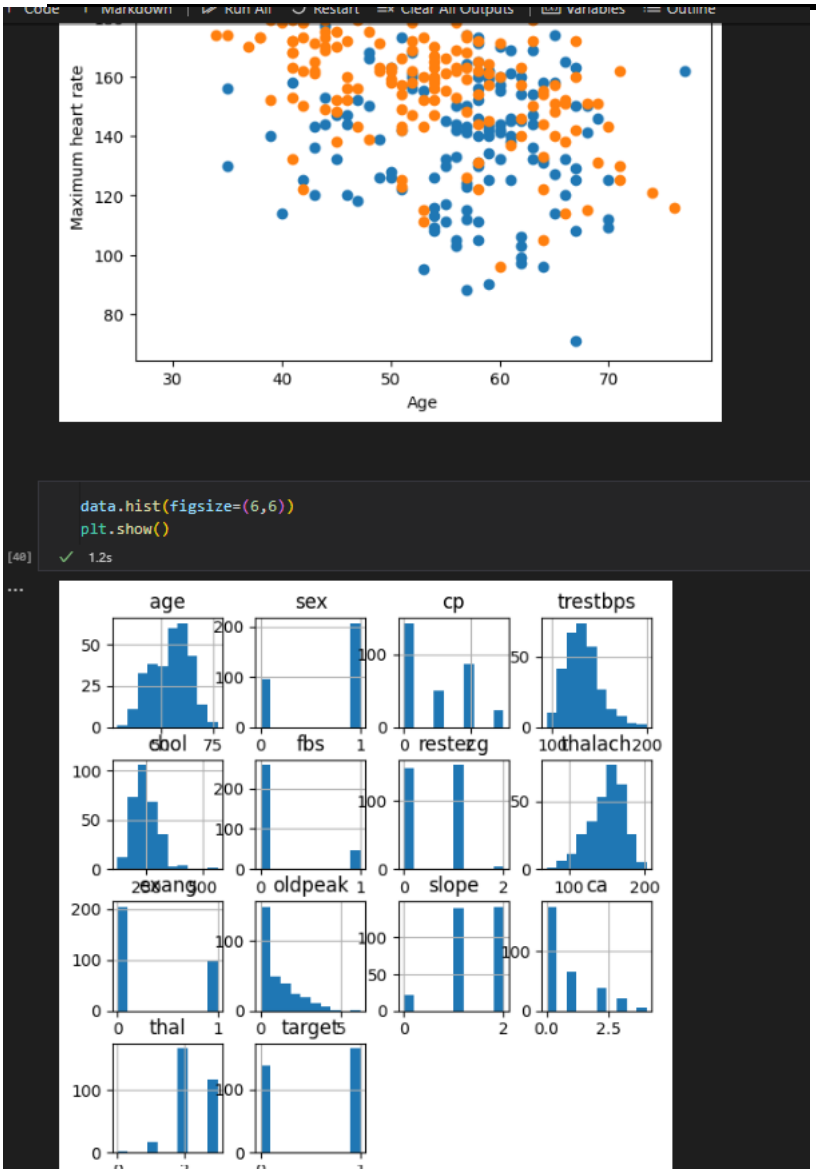
```
sns.heatmap(data.corr(), cmap="Greens")
```
[38] ✓ 0.3s

```
<Axes: >
```




```
plt.scatter(x=data.age[data.target==0],y=data.thalach[data.target==0])
plt.scatter(x=data.age[data.target==1],y=data.thalach[data.target==1])
plt.xlabel("Age")
plt.ylabel("Maximum heart rate")
plt.legend(['0','1'])
plt.show()
```
[39] ✓ 0.1s




Maximum heart rate
160
140
120
100
80
30
40
50
60
70
Age
data.hist(figsize=(6,6))
plt.show()
[40]
1.2s
age
sex
cp
trestbps
chol
fbs
restecg
thalach
exang
oldpeak
slope
ca
thal
target

```
new_df=pd.get_dummies(data,columns=['cp','slope','thal'])
new_df.head()
```
[41] ✓ 0.0s

	age	sex	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	ca	...	cp_1	cp_2	cp_3	slope_0	slope_1	slope_2	thal_0	thal_1	thal_2	thal_3
0	63	1	145	233	1	0	150	0	2.3	0	...	False	False	True	True	False	False	False	True	False	False
1	37	1	130	250	0	1	187	0	3.5	0	...	False	True	False	True	False	False	False	False	True	False
2	41	0	130	204	0	0	172	0	1.4	0	...	True	False	False	False	False	True	False	False	True	False
3	56	1	120	236	0	1	178	0	0.8	0	...	True	False	False	False	False	True	False	False	True	False
4	57	0	120	354	0	1	163	1	0.6	0	...	False	False	False	False	False	True	False	False	True	False

5 rows × 22 columns

```
X=data.drop(columns='target',axis=1)
Y=data['target']
```
[42] ✓ 0.0s

```
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,stratify=Y,random_state=2)
```
[43] ✓ 0.0s

```
print(X.shape,X_train.shape,X_test.shape)
```
[44] ✓ 0.0s

```
(303, 13) (242, 13) (61, 13)
```

```
from matplotlib.pyplot import sca
from sklearn.discriminant_analysis import StandardScaler


scalar=StandardScaler()
X_train=scalar.fit_transform(X_train)
X_test=scalar.transform(X_train)
X_test
```
[45] ✓ 0.7s

```
C:\Users\DELL\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.11_qbz5n2kfra8p0\LocalCache\local-packages\Python311\site-packages\sklearn\ba
  warnings.warn(
```


```
from matplotlib.pyplot import sca
from sklearn.discriminant_analysis import StandardScaler


scalar=StandardScaler()
X_train=scalar.fit_transform(X_train)
X_test=scalar.transform(X_train)
X_test
```
45] ✓ 0.7s

```
C:\Users\DELL\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.1
  warnings.warn(
```


```
array([[-5.9841691 ,  0.0482465 , -0.89429863, ..., -0.72410136,
        -1.39383985, -1.87634007],
       [-5.70607802,  0.0482465 , -1.8603665 , ..., -0.72410136,
         1.57453212, -4.73758689],
       [-5.95998727,  0.0482465 , -0.89429863, ..., -6.07216765,
        -1.39383985, -4.73758689],
       ...,
       [-5.81489627,  0.0482465 , -1.8603665 , ..., -3.39813451,
         0.5850748 , -1.87634007],
       [-5.9116236 ,  0.0482465 ,  0.07176923, ..., -3.39813451,
        -1.39383985, -4.73758689],
       [-6.04462369, -4.49970721, -0.89429863, ..., -3.39813451,
        -1.39383985, -4.73758689]])
```

Student Name: Sawera Fazal_____Roll No:21A-026-se_____Section: 21A_____

CS334 - Machine Learning

Lab 03

Instructor: Ms. Maham Ashraf
E-mail: mashraf@uit.edu

Semester: Fall, 2023

Objective

The purpose of this lab session is to introduce Analytical Base Table (ABT) for Machine Learning(ML) projects.

Instructions

You have to perform the following tasks yourselves. Raise your hand if you face any difficulty in understanding and solving these tasks. **Plagiarism** is an abhorrent practice and you should not engage in it.

How to Submit

- Submit lab work in a single .py file on Microsoft Teams. (No other format will be accepted)
- Lab work file name should be saved with your roll number (e.g. 19a-001-SE_LW01.py)
- Submit home work in a single .py file on Microsoft Teams. (No other format will be accepted)
- Home work file name should be saved with your roll number (e.g. 19a-001-SE_HW01.py)

1 Data Processing

1.1 Analytics Base Table (ABT)

An **Analytics Base Table (ABT)** is a simple, flat, tabular data structure made up of rows and columns. The columns are divided into a set of descriptive features and a single target feature. Each row contains a value for each descriptive feature and the target feature and represents an instance about which a prediction can be made.
Pandas-DataFrame is a Python library specially designed to load and manipulate data in ABT.

```
import pandas as pd
data = pd.read_csv("Motor Insurance Fraud Claim ABTFull .csv")
df = pd.DataFrame(data)
df.head()
```

Listing 1: Load data in data frame.

The above code load data from a csv file and place in a pandas data frame object. Now data is in ready for further processing.

2 Lab Tasks

1 Download Zameen.com property data from https://www.kaggle.com/datasets/huzzefakhan/zameencom-property-data-pakistan

2 Describe the data properties of each column,

- Data type of each column
- missing values in each column
- null values in each column
- outliers in each columns

3 Handle null values by replacing with suitable values

4 Suppose you have to predict the cost of a house, for this purpose select the appropriate coulmns that will help you to develop machine laerning model. Save the selected coulmns dataset in a separate csv file.

5 List down the descriptive variables and target variable.

6 Describe the statistics of the new data.

7 Compute the covariance and correlation matrix among descriptive variables.

8 Group the data by city, location, and area.

9 Count the total values of each item of all attributes.

10 Encode categorical values of *'property_type'* and *'province_name'* fearures with numbers.

```
import pandas as pd
import numpy as np
import seaborn as sns
data1= pd.read_csv("property.csv")
df = pd.DataFrame(data1)
df.head()
```

```
#data properties of each column
df.dtypes
```

```
# missing  values  in each column
df.isna().sum()
```

```
#null values in each column
df.isnull().sum()
```

```
sns.boxplot(y=df.columns)
```

```
import numpy as np
import pandas as pd

def detect_outliers(data, column_name):
    # Sort the data in ascending order
    sorted_data = data[column_name].sort_values()
```

11

```
    # Calculate the first quartile and third quartile
    q1 = np.percentile(sorted_data, 25)
    q3 = np.percentile(sorted_data, 75)

    # Calculate the interquartile range (IQR)
    IQR = 3 - 1

    # Define the lower and upper bounds for outliers
    lower_bound = q1 - 1.5 * IQR
    upper_bound = q3 + 1.5 * IQR

    # Find the outliers
    outliers = data[(data[column_name] < lower_bound) | (data[column_name] > upper_bound)]

    print( outliers)

# Assuming you have a DataFrame called 'f' with a column 'column_name' where you want to detect outliers.
outliers = detect_outliers(df, 'latitude')
```

```
# 2) Handle null values by replacing with suitable values
agency_value = "Forces Properties"
df['agency'].fillna(agency_value, inplace=True)
agent_value = "Khalid Saeed Khan"
df['agent'].fillna(agent_value, inplace=True)
```

```
#to predict the cost of a house
data = df.loc[:, ['property_type', 'price', 'location', 'city','province_name','locality','latitude','longitude',
data.head()
```

```
#statistics of new data
data.describe()
```

[ ]

```
data.cov()
```

[ ]

```
data.corr()
```

[ ]

```
#  Group the data by city, location, and area.
grouped_data=data.groupby(['city','location','area'])
print(grouped_data.size())
```

[ ]

```
#Count the total values of each item of all attributes.
data.count()
```

[ ]

```
#Encode categorical values of 'property_type' and 'province_name' fearures with numbers.
from sklearn.preprocessing import LabelEncoder
label_encoder = LabelEncoder()
data['property_type'] = label_encoder.fit_transform(data['property_type'])
data['province_name'] = label_encoder.fit_transform(data['province_name'])
```

[ ]

```
data.head()
```

[ ]

Student Name: Sawera Fazal_____Roll No:21A-026-se_____Section: 21A_____

CS334 - Machine Learning

Lab 04 - Feature Selection Methods in ML (Part - 1)

Instructor: Ms. Maham Ashraf
E-mail: mashraf@uit.edu

Semester: Fall, 2023

Objective

The purpose of this lab session is to introduce feature selection methods for machine learning model. This lab is divided into two parts, in Part-1 we will use only Filter for feature selection. In Part-2, we will use Wrapper, Embedded and Hybrid methods for feature selections.

Instructions

You have to perform the following tasks yourselves. Raise your hand if you face any difficulty in understanding and solving these tasks. **Plagiarism** is an abhorrent practice and you should not engage in it.

How to Submit

- Submit lab work in a single .py file on Microsoft Teams. (No other format will be accepted)
- Lab work file name should be saved with your roll number (e.g. 19a-001-SE_LW01.py)
- Submit home work in a single .py file on Microsoft Teams. (No other format will be accepted)
- Home work file name should be saved with your roll number (e.g. 19a-001-SE_HW01.py)

1 Homework

1. Apply *Information Gain (IG)* methods for feature selection on *Motor Insurance Fraud* data set and discover the selected features.

```
#1 Apply Information Gain (IG) methods for feature selection on Motor Insurance Fraud data set and  discover  the  selected  features.
import pandas as pd
data=pd.read_csv("car.csv")
data.head()
```

[1] ✓ 5.0s

	Month	WeekOfMonth	DayOfWeek	Make	AccidentArea	DayOfWeekClaimed	MonthClaimed	Sex	MaritalStatus	Age	...	RepNumber	Deductibl
0	Dec	5	Wednesday	Honda	Urban	Tuesday	Jan	Female	Single	21	...	12	3(
1	Jan	3	Wednesday	Honda	Urban	Monday	Jan	Male	Single	34	...	15	4(
2	Oct	5	Friday	Honda	Urban	Thursday	Nov	Male	Married	47	...	7	4(
3	Jun	2	Saturday	Toyota	Rural	Friday	Jul	Male	Married	65	...	4	4(
4	Jan	5	Monday	Honda	Urban	Tuesday	Feb	Female	Single	27	...	3	4(

5 rows × 23 columns

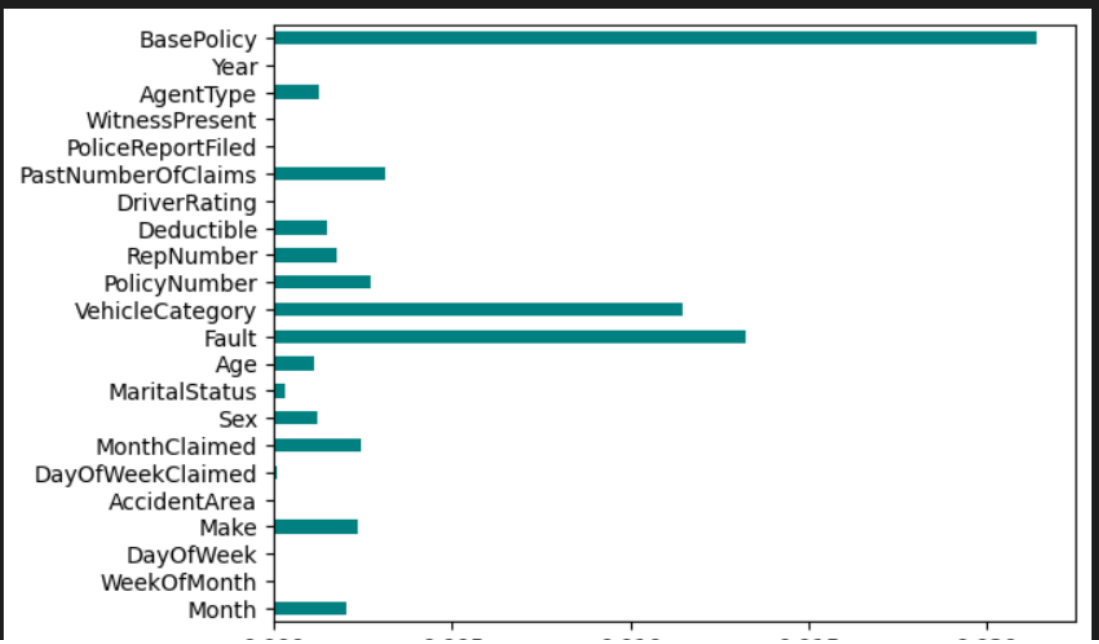
```
from sklearn.preprocessing import LabelEncoder
label_encoder = LabelEncoder()
data['Month'] = label_encoder.fit_transform(data['Month'])
data['DayOfWeek'] = label_encoder.fit_transform(data['DayOfWeek'])
data['Make'] = label_encoder.fit_transform(data['Make'])
data['AccidentArea'] = label_encoder.fit_transform(data['AccidentArea'])
data['DayOfWeekClaimed'] = label_encoder.fit_transform(data['DayOfWeekClaimed'])
data['MonthClaimed'] = label_encoder.fit_transform(data['MonthClaimed'])
data['Sex'] = label_encoder.fit_transform(data['Sex'])
data['MaritalStatus'] = label_encoder.fit_transform(data['MaritalStatus'])
data['Fault'] = label_encoder.fit_transform(data['Fault'])
data['VehicleCategory'] = label_encoder.fit_transform(data['VehicleCategory'])
data['PoliceReportFiled'] = label_encoder.fit_transform(data['PoliceReportFiled'])
data['WitnessPresent'] = label_encoder.fit_transform(data['WitnessPresent'])
data['AgentType'] = label_encoder.fit_transform(data['AgentType'])
data['BasePolicy'] = label_encoder.fit_transform(data['BasePolicy'])
data['FraudFound'] = label_encoder.fit_transform(data['FraudFound'])
X = data.drop('FraudFound', axis=1)
Y = data['FraudFound']

X
```

✓ 6.3s

	Month	WeekOfMonth	DayOfWeek	Make	AccidentArea	DayOfWeekClaimed	MonthClaimed	Sex	MaritalStatus	Age	...	PolicyNur
0	2	5	6	6	1	6	5	0	2	21	...	
1	4	3	6	6	1	2	5	1	2	34	...	
2	10	5	0	6	1	5	10	1	1	47	...	
3	6	2	2	17	0	1	6	1	1	65	...	
4	4	5	1	6	1	6	4	0	2	27	...	
...	...	...	...	...	...	...	...	...	...	...	...	...
15415	9	4	0	17	1	6	10	1	1	35	...	1
15416	9	5	4	13	1	1	3	1	1	30	...	1
15417	9	5	4	17	0	1	3	1	2	24	...	1
15418	2	1	1	17	1	5	3	0	1	34	...	1

```
from sklearn.feature_selection import mutual_info_classif
import pandas as pd
import matplotlib.pyplot as plt
imp= mutual_info_classif(X,Y)
feat_importance=pd.Series(imp,data.columns[0:len(data.columns)-1])

feat_importance.plot(kind='barh',color='teal')
plt.show()
```

[5] ✓ 5.3s



2. Apply χ^2 methods for feature selection on *diabetes* data set and discover the selected features.

```
#2 Apply χ2 methods for feature selection on diabetes data set and discover the
selected features.
import pandas as pd

df = pd.read_csv('diabetes.csv')

# Select specific columns for analysis
#The χ² test is particularly suitable for analyzing the association between
categorical variables.
df = df.loc[:,
['Pregnancies','Glucose','BloodPressure','SkinThickness','Insulin','BMI','Diabet
esPedigreeFunction','Age','Outcome']]
df
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1
...	...	...	...	...	...	...	...	...	...
763	10	101	76	48	180	32.9	0.171	63	0
764	2	122	70	27	0	36.8	0.340	27	0
765	5	121	72	23	112	26.2	0.245	30	0
766	1	126	60	0	0	30.1	0.349	47	1
767	1	93	70	31	0	30.4	0.315	23	0

768 rows × 9 columns

```
df.shape
```

✓ 0.0s

```
(768, 9)
```

```
import pandas as pd
from sklearn.feature_selection import SelectPercentile, chi2

# Define X and y
x = df.drop('Outcome', axis=1)
y =df['Outcome']
x
```

✓ 0.0s

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
0	6	148	72	35	0	33.6	0.627	50
1	1	85	66	29	0	26.6	0.351	31
2	8	183	64	0	0	23.3	0.672	32
3	1	89	66	23	94	28.1	0.167	21
4	0	137	40	35	168	43.1	2.288	33
...	...	...	...	...	...	...	...	...
763	10	101	76	48	180	32.9	0.171	63
764	2	122	70	27	0	36.8	0.340	27
765	5	121	72	23	112	26.2	0.245	30
766	1	126	60	0	0	30.1	0.349	47
767	1	93	70	31	0	30.4	0.315	23

768 rows × 8 columns

```
    chi_scores = chi2(x, y)
    print(chi_scores)
```

✓ 0.0s

```
(array([ 111.51969064, 1411.88704064,   17.60537322,   53.10803984,
        2175.56527292,  127.66934333,    5.39268155,  181.30368904]), array([4.
        0.00000000e+000, 1.32590849e-029, 2.02213728e-002, 2.51638830e-041]))
```

```
    p_values = pd.Series(chi_scores[1], index=x.columns)
    print(p_values[:])
```

✓ 0.0s

```
Pregnancies                 4.552610e-26
Glucose                    5.487286e-309
BloodPressure               2.718193e-05
SkinThickness               3.156977e-13
Insulin                     0.000000e+00
BMI                         1.325908e-29
DiabetesPedigreeFunction    2.022137e-02
Age                         2.516388e-41
dtype: float64
```

```
p_values.sort_values(ascending=True, inplace=True)

# to select top 50% features
x_new = SelectPercentile(chi2, percentile=50)  # Use percentile method
#transforming X to only selected features
x_new.fit_transform(x, y)

# Get the selected features
feature_names = ['Pregnancies','Glucose','BloodPressure','SkinThickness','Insulin','BMI','DiabetesPedigreeFunction','Ag
new_features = []  # The list of your K best features

#creating a filter to traverse and filter best features
mask = x_new.get_support()

# Print selected features
new_features = []  # The list of your K best features
for bool, feature in zip(mask, feature_names):
    if bool:
        new_features.append(feature)

print(new_features)
```

✓ 0.0s

```
['Glucose', 'Insulin', 'BMI', 'Age']
```

3Apply *Fisher score* methods for feature selection on *Motor Insurance Fraud* data set and discover the selected features.

```
#3 Apply Fisher score methods for feature selection on Motor Insurance Fraud data set
and discover the  selected  features
import pandas as pd
import numpy as np
from regex import B
from sklearn.feature_selection import f_classif

# Load the data
data = pd.read_csv('car.csv')
```

```
# Select the desired features
selected_features = ['Month',   'WeekOfMonth',  'DayOfWeek' ,'Make'
,'AccidentArea',    'DayOfWeekClaimed', 'MonthClaimed',
'Sex', 'MaritalStatus',
'Age','RepNumber'   ,'Deductible'   ,'DriverRating','PastNumberOfClaims','PoliceReportF
iled'
,'WitnessPresent'   ,'AgentType',   'Year','BasePolicy',    'FraudFound']

# Data preprocessing if necessary

# Encoding categorical variables if needed
label_encoder = LabelEncoder()
data['Month'] = label_encoder.fit_transform(data['Month'])
data['DayOfWeek'] = label_encoder.fit_transform(data['DayOfWeek'])
data['Make'] = label_encoder.fit_transform(data['Make'])
data['AccidentArea'] = label_encoder.fit_transform(data['AccidentArea'])
data['DayOfWeekClaimed'] = label_encoder.fit_transform(data['DayOfWeekClaimed'])
data['MonthClaimed'] = label_encoder.fit_transform(data['MonthClaimed'])
data['Sex'] = label_encoder.fit_transform(data['Sex'])
data['MaritalStatus'] = label_encoder.fit_transform(data['MaritalStatus'])
data['Fault'] = label_encoder.fit_transform(data['Fault'])
data['VehicleCategory'] = label_encoder.fit_transform(data['VehicleCategory'])
data['PoliceReportFiled'] = label_encoder.fit_transform(data['PoliceReportFiled'])
data['WitnessPresent'] = label_encoder.fit_transform(data['WitnessPresent'])
data['AgentType'] = label_encoder.fit_transform(data['AgentType'])
data['BasePolicy'] = label_encoder.fit_transform(data['BasePolicy'])
data['FraudFound'] = label_encoder.fit_transform(data['FraudFound'])
a = data.drop('FraudFound', axis=1)
b = data['FraudFound']

# Fill missing values if any
data = data.fillna(0) # You may want to use a more suitable method for your data
# Extract the features and the target variable
a = data[selected_features[:-1]]  # Features
b = data[selected_features[-1]]   # Target

# Calculate Fisher Score
fisher_score, p_value = f_classif(X, Y)

# Print the results
for i in range(len(selected_features) - 1):
    print("Feature:", selected_features[i])
    print("Fisher Score:", fisher_score[i])
    print("p-value:", p_value[i])
    print("----------")
#p values small ==most imp
```

```
Feature: Month
Fisher Score: 1.159019579580683
p-value: 0.2816858087248288
----------
Feature: WeekOfMonth
Fisher Score: 2.169535347352119
p-value: 0.14078917522963263
----------
Feature: DayOfWeek
Fisher Score: 4.699457865273744
p-value: 0.03018738297453482
----------
Feature: Make
Fisher Score: 5.677963162541267
p-value: 0.017191341773980746
----------
Feature: AccidentArea
Fisher Score: 17.321734961875425
p-value: 3.172692690041703e-05
----------
Feature: DayOfWeekClaimed
Fisher Score: 0.06572867199351251
p-value: 0.7976639104778226
----------
Feature: MonthClaimed
...
Feature: BasePolicy
Fisher Score: 1.001001267916479
p-value: 0.31708405320770233
```

4Apply *Correlation & Variance* methods for feature selection on *Property* data set and discover the selected features.

```
#4 Apply Correlation & Variance methods for feature selection on Property data set and
discover the  selected  features
# correlation should be more with target value and covariance with with features

import pandas as pd
from sklearn.preprocessing import LabelEncoder

# Load the data
data = pd.read_csv('Property.csv')
data.head()

# Calculate correlation coefficients for the entire dataset
correlation_matrix = data.corr()

# Calculate variance for the entire dataset
variance = data.var()

# Select the desired features
selected_features =
['location_id','page_url','property_type','price','price_bin','location','city','provin
ce_name','locality','latitude','longitude','baths','area','area_marla','area_sqft',
                    'purpose','bedrooms','date_added','year','month','day','agency','a
gent']
```

```
# Applying LabelEncoder to the actual DataFrame columns
label_encoder = LabelEncoder()
data['location_id'] = label_encoder.fit_transform(data['location_id'])
data['page_url'] = label_encoder.fit_transform(data['page_url'])
data['property_type'] = label_encoder.fit_transform(data['property_type'])
data['price'] = label_encoder.fit_transform(data['price'])
data['price_bin'] = label_encoder.fit_transform(data['price_bin'])
data['location'] = label_encoder.fit_transform(data['location'])
data['city'] = label_encoder.fit_transform(data['city'])
data['province_name'] = label_encoder.fit_transform(data['province_name'])
data['locality'] = label_encoder.fit_transform(data['locality'])
data['latitude'] = label_encoder.fit_transform(data['latitude'])
data['longitude'] = label_encoder.fit_transform(data['longitude'])
data['baths'] = label_encoder.fit_transform(data['baths'])
data['area'] = label_encoder.fit_transform(data['area'])
data['area_marla'] = label_encoder.fit_transform(data['area_marla'])
data['area_sqft'] = label_encoder.fit_transform(data['area_sqft'])
data['purpose'] = label_encoder.fit_transform(data['purpose'])
data['bedrooms'] = label_encoder.fit_transform(data['bedrooms'])
data['date_added'] = label_encoder.fit_transform(data['date_added'])
data['year'] = label_encoder.fit_transform(data['year'])
data['month'] = label_encoder.fit_transform(data['month'])
data['day'] = label_encoder.fit_transform(data['day'])
data['agency'] = label_encoder.fit_transform(data['agency'])
data['agent'] = label_encoder.fit_transform(data['agent'])

# Calculate correlation coefficients for the selected features
selected_corr_matrix = data[selected_features].corr()

# Calculate variance for the selected features
selected_variance = data[selected_features].var()

# Print the results
print("Correlation Coefficients for the Entire Dataset:")
print(correlation_matrix)
print("\nVariance for the Entire Dataset:")
print(variance)

print("\nCorrelation Coefficients for the Selected Features:")
print(selected_corr_matrix)
print("\nVariance for the Selected Features:")
print(selected_variance)
```

5Apply *Mean Absolute Difference (MAD)* method for feature selection on *diabetes* data set and discover the selected features.

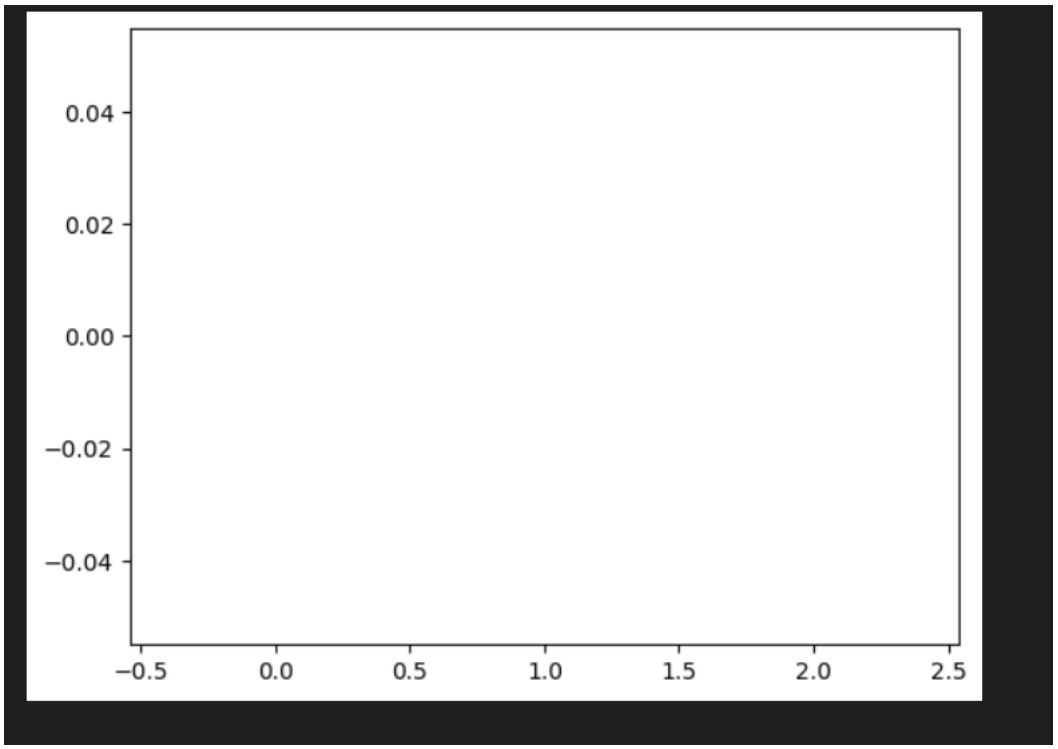
```
#5 Apply Mean Absolute Difference (MAD) method for feature selection on diabetes data
set and discover  the  selected  features.
```

```
import pandas as pd

# Load data
datas = pd.read_csv('diabetes.csv')
datas.head()
datas.columns
# Load data
import pandas as pd
datas = 'diabetes.csv'
names = ['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],

d= pd.read_csv(datas, names=names)

# Print the first few rows of the DataFrame
print(d.head())

# Extracting the features
X = d.iloc[:, :4]

# Calculating Mean Absolute Deviation
mad = X.mad(axis=0)
print("Mean absolute deviation of columns:")
print(mad)
```

6Apply *Dispersion Ratio* method for feature selection on *Churn* data set and discover the selected features.

```
#6. Apply Dispersion  Ratio  method for feature selection on Churn  data set and
discover the selected features
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# Load data
data = 'Churn_Modelling.csv'
names = ['Geography','Gender','Age']
df = pd.read_csv(data, names=names)

# Convert the DataFrame to numeric values
df = df.apply(pd.to_numeric, errors='coerce')

# Extracting the features
X = df.iloc[:, :4]
X = X + 1

# Calculating Arithmetic Mean
am = np.mean(X, axis=0)

# Calculating Geometric Mean
```

```
gm = np.power(np.product(X, axis=0), 1 / X.shape[0])

# Ratio of arithmetic mean and geometric mean
disp_ratio = am / gm

# Plotting the bar graph
plt.bar(np.arange(X.shape[1]), disp_ratio, color='teal')
plt.show()
```



Student Name: Sawera Fazal_____Roll No:21A-026-se_____Section: 21A_____

CS334 - Machine Learning

Lab 05 - Feature Selection Methods in ML (Part - 2)

Instructor: Ms. Maham Ashraf
E-mail: mashraf@uit.edu

Semester: Fall, 2023

Objective

The purpose of this lab session is to introduce feature selection methods for machine learning model. This lab is divided into two parts, in Part-1 we have used Filter methods for extracting important features from data sets. In Part-2, we will use Wrapper, Embedded and Hybrid methods for feature selections.

Instructions

You have to perform the following tasks yourselves. Raise your hand if you face any difficulty in understanding and solving these tasks. **Plagiarism** is an abhorrent practice and you should not engage in it.

How to Submit

- Submit lab work in a single .py file on Microsoft Teams. (No other format will be accepted)
- Lab work file name should be saved with your roll number (e.g. 19a-001-SE_LW01.py)
- Submit home work in a single .py file on Microsoft Teams. (No other format will be accepted)
- Home work file name should be saved with your roll number (e.g. 19a-001-SE_HW01.py)

1 Wrapper Methods: for Features Selection

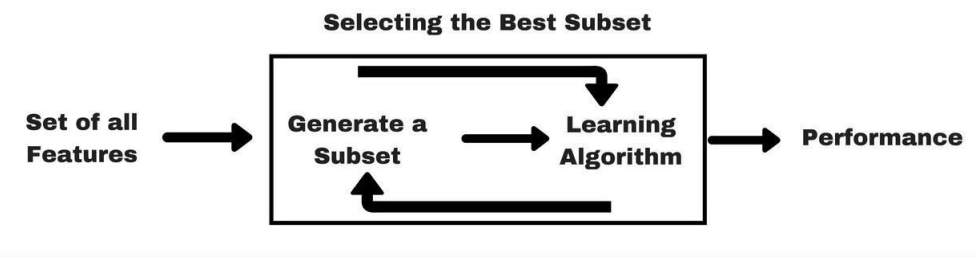


Figure 1: Wrapper Method

Wrappers require some method to search the space of all possible subsets of features, assessing their quality by learning and evaluating a classifier with that feature subset. The feature selection

2 Homework

1. Apply *Forward Feature Selection* methods on **housing** data set and discover the selected features.

```
#1. Apply Forward Feature Selection methods on housing data set and discover the
selected features
# 1. Load  data
import pandas as pd
dataframe = pd.read_csv("housing.csv")  # Replace 'pima-indians-diabetes.csv'
with your dataset's filename

# Assume 'target_feature' is your target feature, modify it as per your dataset
X_df = dataframe.drop('MEDV', axis=1)  # Assuming 'target_feature' is the target
feature
y_series = dataframe['MEDV']  # Assuming 'target_feature' is the target feature

X = X_df  # Features
y = y_series  # Target variable


# 2. Split data into descriptive and target features in X and y variables
respectively.

# 3. Load important libraries
from sklearn.linear_model import LogisticRegression
from mlxtend.feature_selection import SequentialFeatureSelector as SFS

# 4. Apply Model
lr = LogisticRegression(class_weight='balanced', solver='lbfgs',random_state=42,
n_jobs=-1, max_iter=500)
lr.fit(X, y)

# 5. Select best features
bfs = SFS(lr,
          k_features='best',
          forward=True,
          floating=False,
          verbose=2,
          scoring='accuracy',
          cv=0)
bfs = bfs.fit(X, y)

# 6. Print feature list
features = list(bfs.k_feature_names_)
print(features)
```

2. Apply *Backward Feature Elimination* methods on **diabetes** data set and discover the selected

features.

```
#2Apply Backward Feature Elimination methods on diabetes data set and discover
the selected features.

from sklearn.neighbors import KNeighborsClassifier

import pandas as pd

dataframe = pd.read_csv("diabetes.csv")  # Replace 'pima-indians-diabetes.csv'
with your dataset's filename

# Assume 'target_feature' is your target feature, modify it as per your dataset
X_df = dataframe.drop('Outcome', axis=1)  # Assuming 'target_feature' is the
target feature
y_series = dataframe['Outcome']  # Assuming 'target_feature' is the target
feature

# 4. Apply Model
lr = LogisticRegression(class_weight='balanced', solver='lbfgs',
random_state=42, n_jobs=-1, max_iter=500)
lr.fit(X, y)

# 5. Select best features
bfs = SFS(lr,
          k_features='best',
          forward=False,
          floating=False,
          verbose=2,
          scoring='accuracy',
          cv=0)
bfs = bfs.fit(X, y)

# 6. Print feature list
features = list(bfs.k_feature_names_)
print(features)
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  30 out of  30 | elapsed:    2.4s finished

[2023-10-26 10:42:25] Features: 29/1 -- score: 0.9630931458699473[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurr
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  29 out of  29 | elapsed:    2.2s finished

[2023-10-26 10:42:27] Features: 28/1 -- score: 0.9630931458699473[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurr
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  28 out of  28 | elapsed:    2.3s finished

[2023-10-26 10:42:30] Features: 27/1 -- score: 0.9630931458699473[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurr
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  27 out of  27 | elapsed:    2.0s finished

[2023-10-26 10:42:32] Features: 26/1 -- score: 0.9648506151142355[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurr
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  26 out of  26 | elapsed:    2.0s finished

[2023-10-26 10:42:34] Features: 25/1 -- score: 0.9666080843585237[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurr
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  25 out of  25 | elapsed:    1.9s finished

[2023-10-26 10:42:36] Features: 24/1 -- score: 0.9666080843585237[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurr
...
[Parallel(n_jobs=1)]: Done   6 out of   6 | elapsed:    0.0s finished

[2023-10-26 10:42:53] Features: 5/1 -- score: 0.961335676625659[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurre
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
```

3. Apply *Exhaustive Feature Selection* methods on **Motor Insurance Fraud** data set and discover the selected features.

```
#3.  Apply Exhaustive  Feature Selection  methods on Motor Insurance Fraud
dataset and discover the  selected features.
from sklearn.neighbors import KNeighborsClassifier
from mlxtend.feature_selection import ExhaustiveFeatureSelector as EFS
import pandas as pd

dataframee = pd.read_csv("insurance_claims.csv")  # Replace 'pima-indians-
diabetes.csv' with your dataset's filename

# Assume 'target_feature' is your target feature, modify it as per your dataset
X_d = dataframee.drop('Policy_state', axis=1)  # Assuming 'target_feature'is the
target feature
y_serie = dataframee['Policy_state']  # Assuming 'target_feature' is thetarget
feature
```

```
# 2 Split data into descriptive and target features in X and y
variablesrespectively.
# Assuming X_df and y_series are your DataFrame and Series, respectively.

knn = KNeighborsClassifier(n_neighbors=3)
efs1 = EFS(knn,
           min_features=1,
           max_features=4,
           scoring='accuracy',
           print_progress=True,
           cv=5)
efs1 = efs1.fit(X_d, y_serie)
```

```
print('Best accuracy score: %.2f' % efs1.best_score_)
print('Best subset (indices): ', efs1.best_idx_)
print('Best subset (corresponding names): ', efs1.best_feature_names_)
```

4. Apply *Recursive Feature Elimination* methods on **Breast Cancer** data set and discover the selected features.

```
#4. Apply Recursive Feature Elimination methods on Breast Cancer data set and discover theselected features
from sklearn.datasets import load_breast_cancer
data = load_breast_cancer()
X_df = data.data
y_train = data.target


from sklearn.pipeline import Pipeline
from sklearn.model_selection import RepeatedStratifiedKFold, cross_val_score
from sklearn.feature_selection import RFE
import numpy as np
from sklearn.ensemble import GradientBoostingClassifier

# initialize recursive feature elimination class
rfe = RFE(estimator=GradientBoostingClassifier(), n_features_to_select=6)
model = GradientBoostingClassifier()

# make a pipeline for execution
pipe = Pipeline([('Feature Selection', rfe), ('Model', model)])

cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=5, random_state=36851234)

# compute and print score of each feature
n_scores = cross_val_score(pipe, X_df, y_train, scoring='accuracy', cv=cv, n_jobs=-1)
print(np.mean(n_scores))

# Execute RFE model
pipe.fit(X_df, y_train)
print(rfe.support_)
```

```
0.9588471177944862
[False False False False False False False  True False False False False
 False False False False False False False  True  True  True  True
 False False False  True False False]
```

5. Apply *Lasso regression (L_1)* method for feature selection on **housing** data set and discover the selected features.

```python
# 5. Apply Lasso regression ( 1) method for feature selection on housing data set and discover theselected features
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import Lasso
import numpy as np
import pandas as pd
dataframe = pd.read_csv("housing.csv")  # Replace 'pima-indians-diabetes.csv' with your dataset's filename

# Assume 'target_feature' is your target feature, modify it as per your dataset
X_df = dataframe.drop('MEDV', axis=1)  # Assuming 'target_feature' is the target feature
y_series = dataframe['MEDV']  # Assuming 'target_feature' is the target feature

X = data.data
y = data.target

# Assuming X and y are your dataset

X_train, X_test, y_train, y_test = train_test_split(X_df, y_series, test_size=0.33, random_state=42)
features = ['LSTAT','INDUS','NOX','RM','MEDV']

pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('model', Lasso())
])

search = GridSearchCV(
    pipeline,
    {'model__alpha': np.arange(0.1, 10, 0.1)},
    cv=5,
    scoring="neg_mean_squared_error",
    verbose=3
)

search.fit(X_train, y_train)

coefficients = search.best_estimator_.named_steps['model'].coef_
importance = np.abs(coefficients)
```

```
    print(importance)
    print(np.array(features)[importance > 0])
```

```
Fitting 5 folds for each of 99 candidates, totalling 495 fits
[CV 1/5] END .................model__alpha=0.1;, score=-39.617 total time=   0.0s
[CV 2/5] END .................model__alpha=0.1;, score=-25.881 total time=   0.0s
[CV 3/5] END .................model__alpha=0.1;, score=-37.118 total time=   0.0s
[CV 4/5] END .................model__alpha=0.1;, score=-23.807 total time=   0.0s
[CV 5/5] END .................model__alpha=0.1;, score=-32.385 total time=   0.0s
[CV 1/5] END .................model__alpha=0.2;, score=-39.504 total time=   0.0s
[CV 2/5] END .................model__alpha=0.2;, score=-25.857 total time=   0.0s
[CV 3/5] END .................model__alpha=0.2;, score=-36.877 total time=   0.0s
[CV 4/5] END .................model__alpha=0.2;, score=-24.086 total time=   0.0s
[CV 5/5] END .................model__alpha=0.2;, score=-32.363 total time=   0.0s
[CV 1/5] END model__alpha=0.3000000000000004;, score=-39.487 total time=   0.0s
[CV 2/5] END model__alpha=0.3000000000000004;, score=-25.835 total time=   0.0s
[CV 3/5] END model__alpha=0.3000000000000004;, score=-37.104 total time=   0.0s
[CV 4/5] END model__alpha=0.3000000000000004;, score=-24.360 total time=   0.0s
[CV 5/5] END model__alpha=0.3000000000000004;, score=-32.359 total time=   0.0s
[CV 1/5] END .................model__alpha=0.4;, score=-39.522 total time=   0.0s
[CV 2/5] END .................model__alpha=0.4;, score=-25.848 total time=   0.0s
[CV 3/5] END .................model__alpha=0.4;, score=-37.491 total time=   0.0s
[CV 4/5] END .................model__alpha=0.4;, score=-24.667 total time=   0.0s
[CV 5/5] END .................model__alpha=0.4;, score=-32.388 total time=   0.0s
[CV 1/5] END .................model__alpha=0.5;, score=-39.599 total time=   0.0s
[CV 2/5] END .................model__alpha=0.5;, score=-25.896 total time=   0.0s
[CV 3/5] END .................model__alpha=0.5;, score=-37.935 total time=   0.0s
[CV 4/5] END .................model__alpha=0.5;, score=-25.007 total time=   0.0s
...
[CV 3/5] END .................model__alpha=2.7;, score=-57.418 total time=   0.0s
[CV 4/5] END .................model__alpha=2.7;, score=-33.591 total time=   0.0s
[CV 5/5] END .................model__alpha=2.7;, score=-37.472 total time=   0.0s
[CV 1/5] END .model__alpha=2.8000000000000003;, score=-50.671 total time=   0.0s
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
[CV 2/5] END .model__alpha=2.8000000000000003;, score=-30.212 total time=   0.0s
```

6. Apply *Random forest features selection* method with 50 random states on **diabetes** data set and discover the selected features.

```
    import pandas as pd
    import numpy as np

    dataframe = pd.read_csv("diabetes.csv")  # Replace 'pima-indians-diabetes.csv' with your dataset's filenam

    # Assume 'target_feature' is your target feature, modify it as per your dataset
    X_df = dataframe.drop('Outcome', axis=1)  # Assuming 'target_feature' is the target feature
    y_series = dataframe['Outcome']  # Assuming 'target_feature' is the target feature
    features = ['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
           'BMI', 'DiabetesPedigreeFunction', 'Age']


    from sklearn.model_selection import train_test_split
    X_train, X_test, y_train, y_test = train_test_split(X_df, y_series, test_size=0.33, random_state=42)

    # Apply Random Forest regression algorithm we use ensemble where we will use multiple trees
    from sklearn.ensemble import RandomForestRegressor
    rf = RandomForestRegressor(random_state=0)
    rf.fit(X_df, y_series)

    # Print selected features
    print(rf.feature_importances_)
    print(np.array(features)[rf.feature_importances_ > 0.1])
```

```
[0.06609831 0.32609463 0.08472232 0.05072631 0.05168459 0.17229999
 0.1249409  0.12343295]
['Glucose' 'BMI' 'DiabetesPedigreeFunction' 'Age']
```

Student Name:_____Roll No:_____Section:_____

CS334 - Machine Learning

Lab 06

Instructor: Ms. Maham Ashraf
E-mail: mashraf@uit.edu

Semester: Fall, 2023

Objective

The purpose of this lab session is to introduce learn machine learning model using Decision Tree Algorithms.

Instructions

You have to perform the following tasks yourselves. Raise your hand if you face any difficulty in understanding and solving these tasks. **Plagiarism** is an abhorrent practice and you should not engage in it.

How to Submit

- Submit lab work in a single .py file on Microsoft Teams. (No other format will be accepted)
- Lab work file name should be saved with your roll number (e.g. 19a-001-SE_LW01.py)
- Submit home work in a single .py file on Microsoft Teams. (No other format will be accepted)
- Home work file name should be saved with your roll number (e.g. 19a-001-SE_HW01.py)

1 Lab Tasks

1. Implement *GINI* method to compute the feature entropy.
2. Compute the remainder of the feature by *GINI* method.
3. Compute information gain by *GINI* method.
4. Compute Gain Ratio (GR), for *GINI* method.
5. write the method that print your decision tree

Lab Task

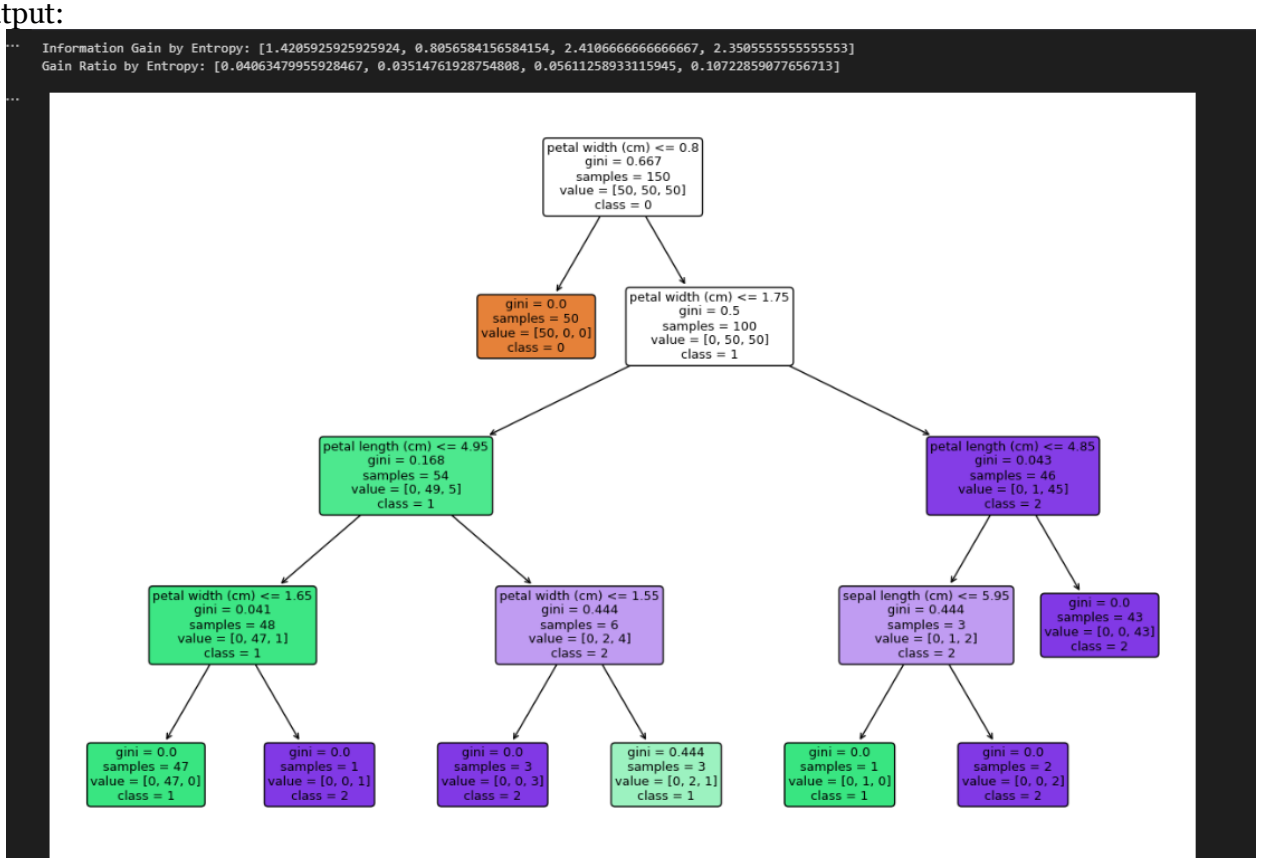
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier, plot_tree

class Gini:
```

```
    def __init__(self, data, t_label):
        self.data = data
        self.t_label = t_label
        self.clf = None

    def preprocess_data(self):
        pass

    def get_feature_entropy(self, data=None, t_label=None):
        if data is None:
            data = self.data
        if t_label is None:
            t_label = self.t_label

        target = data[t_label]
        class_list = target.unique()
        total_row = data.shape[0]
        total_entr = 0

        for c in class_list:
            total_class_count = data[data[t_label] == c].shape[0]
            total_class_entropy = 1 - (total_class_count / total_row) ** 2
            total_entr += total_class_entropy

        return total_entr

    def get_rem_by_entropy(self):
        desc_features = self.data.drop([self.t_label], axis=1)
        target_feature = self.data[self.t_label]
        target_list = target_feature.unique()
        class_count = desc_features.shape[0]
        rem_list = []

        for item in desc_features.columns:
            rem_feature_entropy = 0
            class_list = desc_features[item].unique()
            new_feature = desc_features[item]

            for level in class_list:
                label_class_count = desc_features[desc_features[item] ==
level].shape[0]
                entropy_class = 0
                feature_level_entropy = 0
                sum_feature_entropy = 0

                if label_class_count != 0:
                    probability_class = label_class_count / class_count

                    for tvalue in target_list:
                        count_level_frequency = 0
```

```
                    for i in range(class_count):
                        if (new_feature[i] == level) and (target_feature[i] ==
tvalue):
                            count_level_frequency += 1

                    if count_level_frequency != 0:
                        feature_prob = count_level_frequency / label_class_count
                        feature_level_entropy = 1 - (feature_prob ** 2)
                        sum_feature_entropy += feature_level_entropy

                prob_Xfeature_entropy = probability_class * sum_feature_entropy
                rem_feature_entropy += prob_Xfeature_entropy

            rem_list.append(rem_feature_entropy)

        return rem_list

    def get_info_gain_by_entropy(self):
        target_entropy = self.get_feature_entropy()
        rem = self.get_rem_by_entropy()
        IG_list = [target_entropy - rem[i] for i in range(len(rem))]
        return IG_list

    def get_GR_by_entropy(self):
        feature_list = self.data.drop([self.t_label], axis=1).columns
        GR_list = []
        count = 0
        IG = self.get_info_gain_by_entropy()

        for item in feature_list:
            feat_entropy = self.get_feature_entropy(None, item)
            GR_list.append(IG[count] / feat_entropy)
            count += 1

        return GR_list

    def build_decision_tree(self, max_depth=None, random_state=None):
        # Build Decision Tree classifier
        self.clf = DecisionTreeClassifier(criterion='gini', max_depth=max_depth,
random_state=random_state)
        X = self.data.drop(self.t_label, axis=1)
        y = self.data[self.t_label]
        self.clf.fit(X, y)

    def visualize_tree(self):
        if self.clf is not None:
            plt.figure(figsize=(15, 10))
            plot_tree(self.clf, filled=True, feature_names=self.data.drop(self.t_label,
axis=1).columns,
```

```
                    class_names=list(map(str, self.clf.classes_)), rounded=True)
            plt.show()
        else:
            print("Decision tree not built. Please run build_decision_tree first.")

# Load the Iris dataset
iris = load_iris()
iris_df = pd.DataFrame(data=np.c_[iris['data'], iris['target']],
columns=iris['feature_names'] + ['target'])
iris_df['target'] = iris_df['target'].astype(int)

# Example usage with the Iris dataset
gini_instance = Gini(iris_df, 'target')

# Get Information Gain by Entropy
info_gain_list = gini_instance.get_info_gain_by_entropy()
print("Information Gain by Entropy:", info_gain_list)

# Get Gain Ratio by Entropy
gain_ratio_list = gini_instance.get_GR_by_entropy()
print("Gain Ratio by Entropy:", gain_ratio_list)

gini_instance.build_decision_tree(max_depth=4, random_state=1)

gini_instance.visualize_tree()
```

Output:



2 Homework

1. Modify your ID3 algorithm that can accept continuous (e.g. descriptive and target) features.
2. Compute threshold of each continuous descriptive feature.
3. Select continuous feature for split the tree based on its variance.
4. If the target feature is continuous, split your tree by weighted variance method.
5. write the method that print your decision tree.

Home Task

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier, plot_tree

class Gini:
    def __init__(self, data, t_label):
        self.data = data
        self.t_label = t_label
        self.clf = None

    def preprocess_data(self):
        pass

    def get_feature_entropy(self, data=None, t_label=None):
        if data is None:
            data = self.data
        if t_label is None:
            t_label = self.t_label

        target = data[t_label]
        class_list = target.unique()
        total_row = data.shape[0]
        total_entr = 0

        for c in class_list:
            total_class_count = data[data[t_label] == c].shape[0]
            total_class_entropy = 1 - (total_class_count / total_row) ** 2
            total_entr += total_class_entropy

        return total_entr

    def get_rem_by_entropy(self):
        desc_features = self.data.drop([self.t_label], axis=1)
        target_feature = self.data[self.t_label]
        target_list = target_feature.unique()
        class_count = desc_features.shape[0]
        rem_list = []
```

```
        for item in desc_features.columns:
            rem_feature_entropy = 0
            class_list = desc_features[item].unique()
            new_feature = desc_features[item]

            for level in class_list:
                label_class_count = desc_features[desc_features[item] ==
level].shape[0]
                entropy_class = 0
                feature_level_entropy = 0
                sum_feature_entropy = 0

                if label_class_count != 0:
                    probability_class = label_class_count / class_count

                    for tvalue in target_list:
                        count_level_frequency = 0

                        for i in range(class_count):
                            if (new_feature[i] == level) and (target_feature[i] ==
tvalue):
                                count_level_frequency += 1

                        if count_level_frequency != 0:
                            feature_prob = count_level_frequency / label_class_count
                            feature_level_entropy = 1 - (feature_prob ** 2)
                            sum_feature_entropy += feature_level_entropy

                    prob_Xfeature_entropy = probability_class * sum_feature_entropy
                    rem_feature_entropy += prob_Xfeature_entropy

            rem_list.append(rem_feature_entropy)

        return rem_list

    def get_info_gain_by_entropy(self):
        target_entropy = self.get_feature_entropy()
        rem = self.get_rem_by_entropy()
        IG_list = [target_entropy - rem[i] for i in range(len(rem))]
        return IG_list

    def get_GR_by_entropy(self):
        feature_list = self.data.drop([self.t_label], axis=1).columns
        GR_list = []
        count = 0
        IG = self.get_info_gain_by_entropy()

        for item in feature_list:
            feat_entropy = self.get_feature_entropy(None, item)
            GR_list.append(IG[count] / feat_entropy)
```

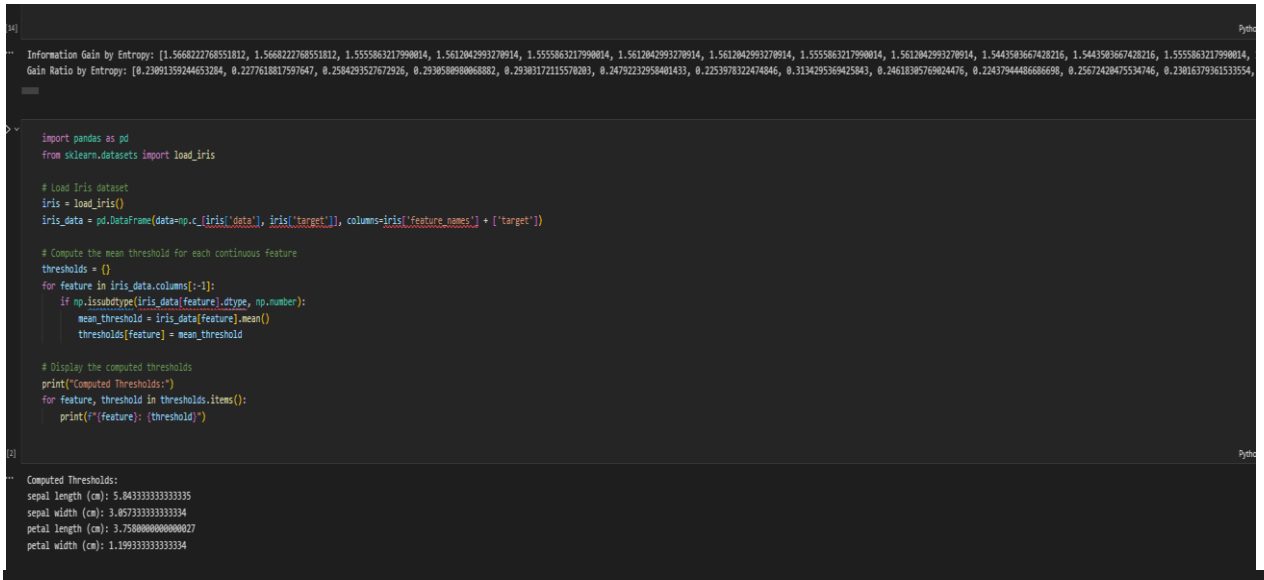
```
            count += 1

        return GR_list

    def build_decision_tree(self, max_depth=None, random_state=None):
        # Build Decision Tree classifier
        self.clf = DecisionTreeClassifier(criterion='gini', max_depth=max_depth,
random_state=random_state)
        X = self.data.drop(self.t_label, axis=1)
        y = self.data[self.t_label]
        self.clf.fit(X, y)

    def visualize_tree(self):
        if self.clf is not None:
            plt.figure(figsize=(15, 10))
            plot_tree(self.clf, filled=True, feature_names=self.data.drop(self.t_label,
axis=1).columns,
                      class_names=list(map(str, self.clf.classes_)), rounded=True)
            plt.show()
        else:
            print("Decision tree not built. Please run build_decision_tree first.")

# Load the Iris dataset
iris = load_iris()
iris_df = pd.DataFrame(data=np.c_[iris['data'], iris['target']],
columns=iris['feature_names'] + ['target'])
iris_df['target'] = iris_df['target'].astype(int)

# Example usage with the Iris dataset
gini_instance = Gini(iris_df, 'target')

# Get Information Gain by Entropy
info_gain_list = gini_instance.get_info_gain_by_entropy()
print("Information Gain by Entropy:", info_gain_list)

# Get Gain Ratio by Entropy
gain_ratio_list = gini_instance.get_GR_by_entropy()
print("Gain Ratio by Entropy:", gain_ratio_list)

gini_instance.build_decision_tree(max_depth=4, random_state=1)

gini_instance.visualize_tree()

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier, plot_tree

class Gini:
```

```
    def __init__(self, data, t_label):
        self.data = data
        self.t_label = t_label
        self.clf = None

    def preprocess_data(self):
        pass

    def get_feature_entropy(self, data=None, t_label=None):
        if data is None:
            data = self.data
        if t_label is None:
            t_label = self.t_label

        target = data[t_label]
        class_list = target.unique()
        total_row = data.shape[0]
        total_entr = 0

        for c in class_list:
            total_class_count = data[data[t_label] == c].shape[0]
            total_class_entropy = 1 - (total_class_count / total_row) ** 2
            total_entr += total_class_entropy

        return total_entr

    def get_rem_by_entropy(self):
        desc_features = self.data.drop([self.t_label], axis=1)
        target_feature = self.data[self.t_label]
        target_list = target_feature.unique()
        class_count = desc_features.shape[0]
        rem_list = []

        for item in desc_features.columns:
            rem_feature_entropy = 0
            class_list = desc_features[item].unique()
            new_feature = desc_features[item]

            for level in class_list:
                label_class_count = desc_features[desc_features[item] ==
level].shape[0]
                entropy_class = 0
                feature_level_entropy = 0
                sum_feature_entropy = 0

                if label_class_count != 0:
                    probability_class = label_class_count / class_count

                    for tvalue in target_list:
                        count_level_frequency = 0
```

```
                    for i in range(class_count):
                        if (new_feature[i] == level) and (target_feature[i] ==
tvalue):
                            count_level_frequency += 1

                    if count_level_frequency != 0:
                        feature_prob = count_level_frequency / label_class_count
                        feature_level_entropy = 1 - (feature_prob ** 2)
                        sum_feature_entropy += feature_level_entropy

                prob_Xfeature_entropy = probability_class * sum_feature_entropy
                rem_feature_entropy += prob_Xfeature_entropy

            rem_list.append(rem_feature_entropy)

        return rem_list

    def get_info_gain_by_entropy(self):
        target_entropy = self.get_feature_entropy()
        rem = self.get_rem_by_entropy()
        IG_list = [target_entropy - rem[i] for i in range(len(rem))]
        return IG_list

    def get_GR_by_entropy(self):
        feature_list = self.data.drop([self.t_label], axis=1).columns
        GR_list = []
        count = 0
        IG = self.get_info_gain_by_entropy()

        for item in feature_list:
            feat_entropy = self.get_feature_entropy(None, item)
            GR_list.append(IG[count] / feat_entropy)
            count += 1

        return GR_list

    def build_decision_tree(self, max_depth=None, random_state=None):
        # Build Decision Tree classifier
        self.clf = DecisionTreeClassifier(criterion='gini', max_depth=max_depth,
random_state=random_state)
        X = self.data.drop(self.t_label, axis=1)
        y = self.data[self.t_label]
        self.clf.fit(X, y)

    def visualize_tree(self):
        if self.clf is not None:
            plt.figure(figsize=(15, 10))
            plot_tree(self.clf, filled=True, feature_names=self.data.drop(self.t_label,
axis=1).columns,
```

```
                        class_names=list(map(str, self.clf.classes_)), rounded=True)
            plt.show()
        else:
            print("Decision tree not built. Please run build_decision_tree first.")

# Load the Iris dataset
iris = load_iris()
iris_df = pd.DataFrame(data=np.c_[iris['data'], iris['target']],
columns=iris['feature_names'] + ['target'])
iris_df['target'] = iris_df['target'].astype(int)

# Example usage with the Iris dataset
gini_instance = Gini(iris_df, 'target')

# Get Information Gain by Entropy
info_gain_list = gini_instance.get_info_gain_by_entropy()
print("Information Gain by Entropy:", info_gain_list)

# Get Gain Ratio by Entropy
gain_ratio_list = gini_instance.get_GR_by_entropy()
print("Gain Ratio by Entropy:", gain_ratio_list)

gini_instance.build_decision_tree(max_depth=4, random_state=1)

gini_instance.visualize_tree()
```

Output:



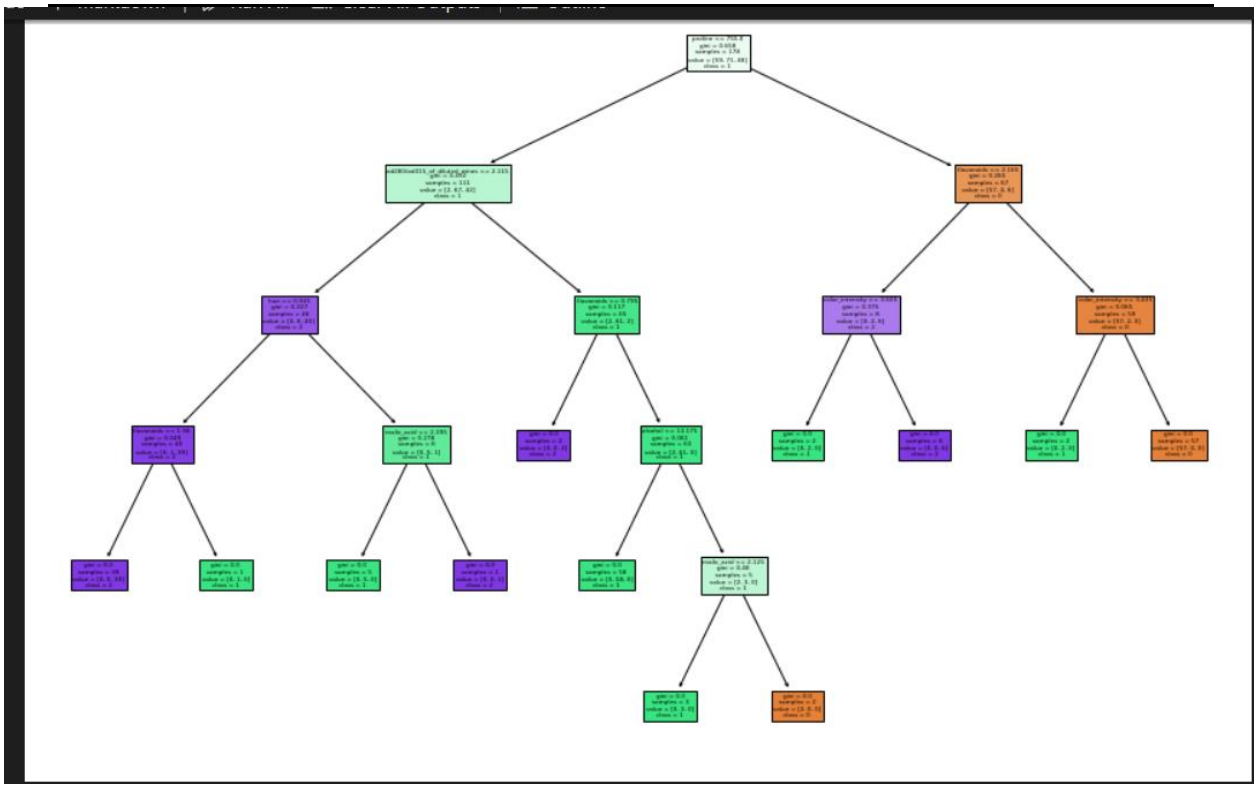

```
import pandas as pd
```

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier, plot_tree

class Gini:
    def __init__(self, data, t_label):
        self.data = data
        self.t_label = t_label
        self.clf = None

    def preprocess_data(self):
        pass

    def get_feature_entropy(self, data=None, t_label=None):
        if data is None:
            data = self.data
        if t_label is None:
            t_label = self.t_label

        target = data[t_label]
        class_list = target.unique()
        total_row = data.shape[0]
        total_entr = 0

        for c in class_list:
            total_class_count = data[data[t_label] == c].shape[0]
            total_class_entropy = 1 - (total_class_count / total_row) ** 2
            total_entr += total_class_entropy

        return total_entr

    def get_rem_by_entropy(self):
        desc_features = self.data.drop([self.t_label], axis=1)
        target_feature = self.data[self.t_label]
        target_list = target_feature.unique()
        class_count = desc_features.shape[0]
        rem_list = []

        for item in desc_features.columns:
            rem_feature_entropy = 0
            class_list = desc_features[item].unique()
            new_feature = desc_features[item]

            for level in class_list:
                label_class_count = desc_features[desc_features[item] ==
level].shape[0]
                entropy_class = 0
                feature_level_entropy = 0
                sum_feature_entropy = 0
```

```
                if label_class_count != 0:
                    probability_class = label_class_count / class_count

                    for tvalue in target_list:
                        count_level_frequency = 0

                        for i in range(class_count):
                            if (new_feature[i] == level) and (target_feature[i] ==
tvalue):
                                count_level_frequency += 1

                        if count_level_frequency != 0:
                            feature_prob = count_level_frequency / label_class_count
                            feature_level_entropy = 1 - (feature_prob ** 2)
                            sum_feature_entropy += feature_level_entropy

                    prob_Xfeature_entropy = probability_class * sum_feature_entropy
                    rem_feature_entropy += prob_Xfeature_entropy

            rem_list.append(rem_feature_entropy)

        return rem_list

    def get_info_gain_by_entropy(self):
        target_entropy = self.get_feature_entropy()
        rem = self.get_rem_by_entropy()
        IG_list = [target_entropy - rem[i] for i in range(len(rem))]
        return IG_list

    def get_GR_by_entropy(self):
        feature_list = self.data.drop([self.t_label], axis=1).columns
        GR_list = []
        count = 0
        IG = self.get_info_gain_by_entropy()

        for item in feature_list:
            feat_entropy = self.get_feature_entropy(None, item)
            GR_list.append(IG[count] / feat_entropy)
            count += 1

        return GR_list

    def build_decision_tree(self, max_depth=None, random_state=None):
        # Build Decision Tree classifier
        self.clf = DecisionTreeClassifier(criterion='gini', max_depth=max_depth,
random_state=random_state)
        X = self.data.drop(self.t_label, axis=1)
        y = self.data[self.t_label]
        self.clf.fit(X, y)
```

```
    def visualize_tree(self):
        if self.clf is not None:
            plt.figure(figsize=(15, 10))
            plot_tree(self.clf, filled=True, feature_names=self.data.drop(self.t_label,
axis=1).columns,
                      class_names=list(map(str, self.clf.classes_)), rounded=True)
            plt.show()
        else:
            print("Decision tree not built. Please run build_decision_tree first.")

# Load the Iris dataset
iris = load_iris()
iris_df = pd.DataFrame(data=np.c_[iris['data'], iris['target']],
columns=iris['feature_names'] + ['target'])
iris_df['target'] = iris_df['target'].astype(int)

# Example usage with the Iris dataset
gini_instance = Gini(iris_df, 'target')

# Get Information Gain by Entropy
info_gain_list = gini_instance.get_info_gain_by_entropy()
print("Information Gain by Entropy:", info_gain_list)

# Get Gain Ratio by Entropy
gain_ratio_list = gini_instance.get_GR_by_entropy()
print("Gain Ratio by Entropy:", gain_ratio_list)

gini_instance.build_decision_tree(max_depth=4, random_state=1)

gini_instance.visualize_tree()
```

Output:



.

Student Name: Sawera Fazal_____Roll No:21A-026-se_____Section: 21A_____

CS334 - Machine Learning

Lab 07 - Similarity Based Learning

Instructor: Ms. Maham Ashraf
E-mail: mashraf@uit.edu

Semester: Fall, 2023

Objective

The purpose of this lab session is to introduce similarity based learning models, such as Nearest Neighbor (NN), KNN, and Weighted KNN .

Instructions

You have to perform the following tasks yourselves. Raise your hand if you face any difficulty in understanding and solving these tasks. **Plagiarism** is an abhorrent practice and you should not engage in it.

How to Submit

- Submit lab work in a single .py file on Microsoft Teams. (No other format will be accepted)
- Lab work file name should be saved with your roll number (e.g. 19a-001-SE_LW01.py)
- Submit home work in a single .py file on Microsoft Teams. (No other format will be accepted)
- Home work file name should be saved with your roll number (e.g. 19a-001-SE_HW01.py)

1 K Nearest Neighbor Algorithm

K Nearest Neighbor algorithm falls under the Supervised Learning category and is used for classification (most commonly) and regression. It is a versatile algorithm also used for imputing missing values and resampling datasets. As the name (K Nearest Neighbor) suggests it considers K Nearest Neighbors (Data points) to predict the class or continuous value for the new Data point.

The algorithm's learning is:

1. Instance-based learning: Here we do not learn weights from training data to predict output (as in model-based algorithms) but use entire training instances to predict output for unseen data.
2. Lazy Learning: Model is not learned using training data prior and the learning process is postponed to a time when prediction is requested on the new instance.
3. Non -Parametric: In KNN, there is no predefined form of the mapping function.

2 Lab Tasks

1. Develop a ML model based on KNN algorithm that can predict the heart disease in a person

Task1:

Develop a ML model based on KNN algorithm that can predict the heart disease in a person. Use heart.csv data set for developing ML mo

```
# Import necessary libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

# Load the dataset
data = pd.read_csv('heart_disease_data.csv')

# Separate features and target variable
X = data.drop('target', axis=1)
y = data['target']

# Split 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)

# Standardize the features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Choose the value of k (number of neighbors)
k = 7

knn = KNeighborsClassifier(n_neighbors=k)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)

# Choose the best value of k based on the plot
```

```
  # Choose the best value of k based on the plot
  print(f'Accuracy: {accuracy}')
```

```
Accuracy: 0.9180327868852459
```

.

Use heart.csv data set for developing ML model.

2. Using Cosine similarity matrix, develop a movie recommender system that recommend similar types of movies based on their genre, plot, and language. Use IMDBdata_MainData.csv dataset for the recommender system.

```
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics.pairwise import cosine_similarity

# Load the dataset
movies_data = pd.read_csv('imdb_movies.csv')

# Combine relevant features into a single column for text-based similarity
movies_data['combined_features'] = movies_data['genre'] + ' ' + movies_data['overview']
+ ' ' + movies_data['orig_lang']

# Handle missing values in 'combined_features'
movies_data['combined_features'].fillna('', inplace=True)
```

```
# Use CountVectorizer to convert text data into numerical vectors
count_vectorizer = CountVectorizer(stop_words='english')
count_matrix = count_vectorizer.fit_transform(movies_data['combined_features'])

# Compute cosine similarity matrix
cosine_sim_matrix = cosine_similarity(count_matrix, count_matrix)

# Function to get movie recommendations based on similarity
def get_recommendations(movie_title, cosine_sim_matrix, movies_data):
    movie_index = movies_data.index[movies_data['names'] == movie_title].tolist()[0]
    similar_movies = list(enumerate(cosine_sim_matrix[movie_index]))

    # Sort the movies based on similarity scores
    sorted_similar_movies = sorted(similar_movies, key=lambda x: x[1], reverse=True)

    # Get the top 5 recommendations (excluding the input movie itself)
    recommended_movies = [(movies_data['names'][i], sorted_similar_movies[i][1]) for i
in range(1, 6)]

    return recommended_movies

# Example: Get recommendations for a movie title
movie_title = "Heart and Souls"
recommendations = get_recommendations(movie_title, cosine_sim_matrix, movies_data)

# Display the recommendations
print(f"Recommendations for '{movie_title}':")
for movie, similarity in recommendations:
    print(f"{movie} (Similarity Score: {similarity:.2f})")
```

```
Recommendations for 'Heart and Souls':
Avatar: The Way of Water (Similarity Score: 0.31)
The Super Mario Bros. Movie (Similarity Score: 0.30)
Mummies (Similarity Score: 0.29)
Supercell (Similarity Score: 0.28)
Cocaine Bear (Similarity Score: 0.27)
```

3. Use Mahalanobis distance and detects the outlier rows in the Diabetes data set. Choose 95% confidence interval for detecting outlier rows.

Task3:

Use Mahalanobis distance and detects the outlier rows in the Diabetes data set. Choose 95% confidence interval for detecting outlier rows

```
import pandas as pd
import numpy as np
from scipy.stats import chi2
from sklearn.covariance import MinCovDet

# Load the Diabetes dataset (replace 'path/to/diabetes.csv' with the actual path)
diabetes_data = pd.read_csv('diabetes.csv')

# Extract the features (excluding the outcome variable)
X = diabetes_data.drop('Outcome', axis=1)

# Fit Minimum Covariance Determinant (MCD) to estimate the robust covariance matrix
mcd = MinCovDet()
mcd.fit(X)

# Calculate Mahalanobis distances
mahalanobis_distances = mcd.mahalanobis(X)

# Set the significance level for outlier detection (e.g., 95% confidence interval)
alpha = 0.05
threshold = chi2.ppf(1 - alpha, df=X.shape[1])  # Chi-squared critical value

# Detect outliers
outliers = diabetes_data[mahalanobis_distances > threshold]

# Display the outliers
print("Outlier Rows:")
outliers
```

```
Outlier Rows:
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
2	8	183	64	0	0	23.3	0.672	32	1
4	0	137	40	35	168	43.1	2.288	33	1
7	10	115	0	0	0	35.3	0.134	29	0
8	2	197	70	45	543	30.5	0.158	53	1
9	8	125	96	0	0	0.0	0.232	54	1
...	...	...	...	...	...	...	...	...	...
753	0	181	88	44	510	43.3	0.222	26	1
757	0	123	72	0	0	36.3	0.258	52	1
759	6	190	92	0	0	35.5	0.278	66	1
763	10	101	76	48	180	32.9	0.171	63	0
766	1	126	60	0	0	30.1	0.349	47	1

218 rows × 9 columns

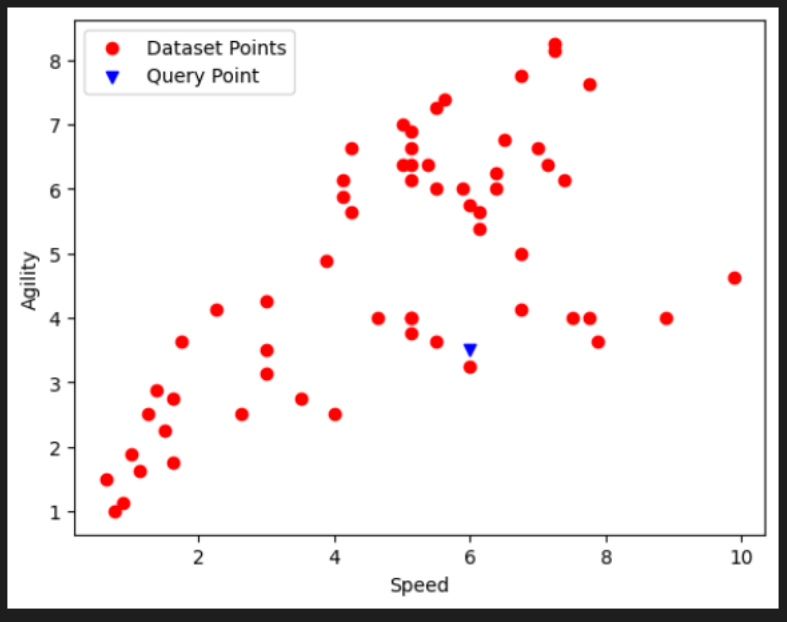
1. Draw KD-tree using Speed-Agility data.

Task4:

Draw KD-tree using Speed-Agility data.

```
import pandas as pd
import numpy as np
from sklearn.neighbors import KDTree
import matplotlib.pyplot as plt

# Read the data from Agility_Speed.csv
data = pd.read_excel('toughestsport.xlsx')
df = pd.DataFrame(data)

# Extract 'Speed' and 'Agility' columns
x = df['SPD']
y = df['AGI']

# Create a list of points
X = list(zip(x, y))

# Build KD-tree
tree = KDTree(X, leaf_size=2)

# Query for the 2 nearest neighbors to [6, 3.5]
dist, ind = tree.query([[6, 3.5]], k=2)
print("Indices of 2 closest neighbors:", ind)
print("Distances to 2 closest neighbors:", dist)

# Scatter plot of the dataset and the query point
plt.scatter(x, y, color='red', marker='o', label='Dataset Points')
plt.scatter(6, 3.5, color='blue', marker='v', label='Query Point')
plt.xlabel('Speed')
plt.ylabel('Agility')
plt.legend()
plt.show()
```

```
Indices of 2 closest neighbors: [[40 35]]
Distances to 2 closest neighbors: [[0.25       0.51662365]]
```

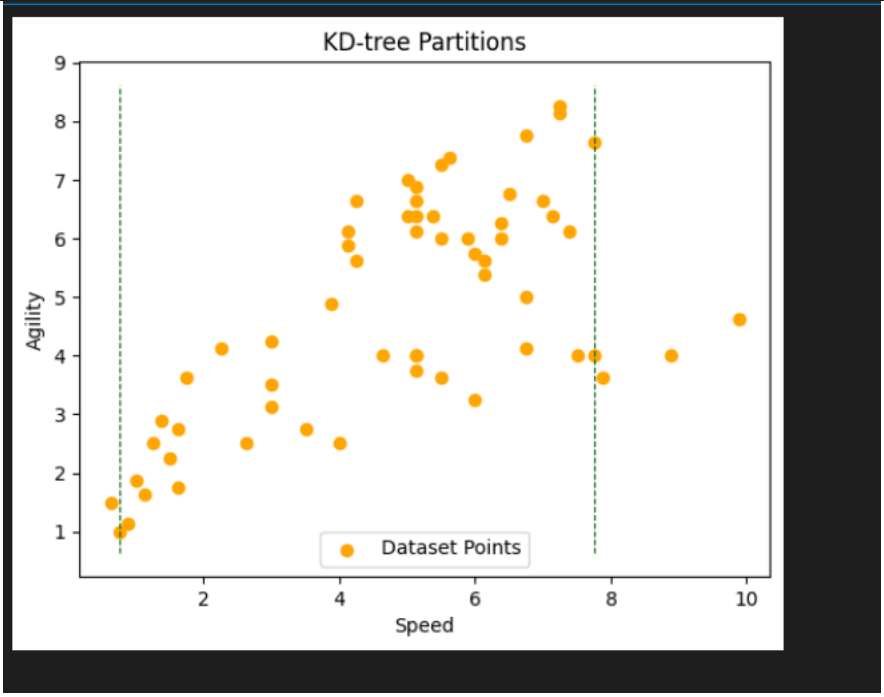


5. plot KD-tree partitions on graph using Speed-Agility data

```
import pandas as pd
import numpy as np
from sklearn.neighbors import KDTree
```

```
import matplotlib.pyplot as plt

# Load your dataset
data = pd.read_excel('toughestsport.xlsx')
df = pd.DataFrame(data)

# Extract 'SPD' and 'AGI' columns
x = df['SPD']
y = df['AGI']

# Create a list of points
X = np.array(list(zip(x, y)))

# Build KD-tree
tree = KDTree(X, leaf_size=2)

# Function to plot KD-tree partitions
def plot_kdtree_partitions(node, depth, bbox):
    if node.data.shape[0] > 1:
        bbox = np.array(bbox).reshape(1, -1)  # Reshape bbox to 2D
        left_indices = node.query(bbox[:, [0, 2]], return_distance=False)[0]
        right_indices = node.query(bbox[:, [1, 3]], return_distance=False)[0]

        left_bbox = [np.min(X[left_indices, 0]), np.max(X[left_indices, 0]),
np.min(X[left_indices, 1]), np.max(X[left_indices, 1])]
        right_bbox = [np.min(X[right_indices, 0]), np.max(X[right_indices, 0]),
np.min(X[right_indices, 1]), np.max(X[right_indices, 1])]

        if depth % 2 == 0:
            plt.plot([X[left_indices[-1], 0], X[left_indices[-1], 0]], [bbox[0, 2],
bbox[0, 3]], c='darkgreen', linestyle='--', linewidth=0.8)
        else:
            plt.plot([bbox[0, 0], bbox[0, 1]], [X[left_indices[-1], 1],
X[left_indices[-1], 1]], c='darkgreen', linestyle='--', linewidth=0.8)

        plot_kdtree_partitions(tree.query(X[left_indices], return_distance=False),
depth + 1, left_bbox)

        if depth % 2 == 0:
            plt.plot([X[right_indices[0], 0], X[right_indices[0], 0]], [bbox[0, 2],
bbox[0, 3]], c='darkgreen', linestyle='--', linewidth=0.8)
        else:
            plt.plot([bbox[0, 0], bbox[0, 1]], [X[right_indices[0], 1],
X[right_indices[0], 1]], c='darkgreen', linestyle='--', linewidth=0.8)

        plot_kdtree_partitions(tree.query(X[right_indices], return_distance=False),
depth + 1, right_bbox)

# Scatter plot of the dataset
plt.scatter(x, y, color='orange', marker='o', label='Dataset Points')
```

```
# Plot KD-tree partitions
plot_kdtree_partitions(tree, 0, plt.axis())

plt.xlabel('Speed')
plt.ylabel('Agility')
plt.title('KD-tree Partitions')
plt.legend()
plt.show()
```



Student Name: sawera Roll No: 21A-026-se Section: 21A

CS334 - Machine Learning

Lab 08 - Ensemble Models (Part-2)

Instructor: Ms. Maham Ashraf E-mail:
mashraf@uit.edu

Semester: Fall, 2023

Objective

The purpose of this lab session is to introduce Ensemble models that use in Machine Learning for better performance.

Instructions

You have to perform the following tasks yourselves. Raise your hand if you face any difficulty in understanding and solving these tasks. **Plagiarism** is an abhorrent practice and you should not engage in it.

How to Submit

- Submit lab work in a single .py file on Microsoft Teams. (No other format will be accepted)
- Lab work file name should be saved with your roll number (e.g. 19a-001-SE_LW01.py)
- Submit home work in a single .py file on Microsoft Teams. (No other format will be accepted)
- Home work file name should be saved with your roll number (e.g. 19a-001-SE_HW01.py)

Lab Tasks

1. Execute tasks of Section, 2, 3, and 4 on *make_classification* dataset of sklearn library.
2. Train and evaluate models as discussed in Section 2, 3, and 4 on *Diabetes* data set select the bestmodel based on evaluation scores.

Task 1

Execute tasks of Section, 2, 3, and 4 on make_classification dataset of sklearn library.

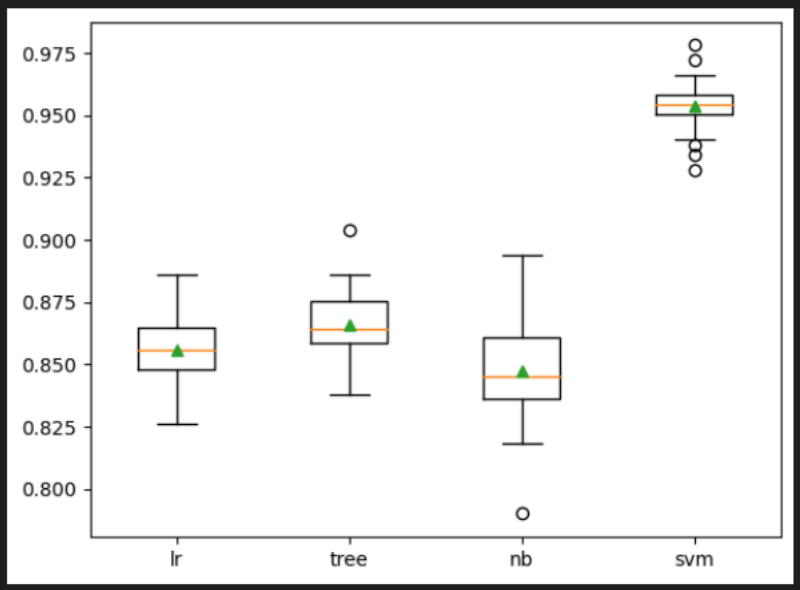
[ ]

Baseline Models and Voting

```
# evaluate standard models on the synthetic dataset
from numpy import mean
from numpy import std
from sklearn.datasets import make_classification
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from matplotlib import pyplot
from sklearn.ensemble import VotingClassifier
# get the dataset

def get_dataset () :
    X, y = make_classification(n_samples=5000 , n_features=20, n_informative=10 ,
n_redundant=10 , random_state=1)
    return X, y
# get a list of models to evaluate
def get_models ():
    models = list()
    models.append(('lr', LogisticRegression()))
    models.append(('tree', DecisionTreeClassifier()))
    models.append (('nb', GaussianNB()))
    models.append(('svm', SVC(probability=True)))
    return models

# evaluate a give model using cross - validation
def evaluate_model (model , X, y):
    # define the model evaluation procedure
    cv = RepeatedStratifiedKFold( n_splits=10 , n_repeats=3 , random_state=1)
    # evaluate the model
    scores = cross_val_score(model, X, y, scoring='accuracy', cv=cv, n_jobs= -1)
    return scores
# define dataset
X, y = get_dataset()
# get the models to evaluate
models = get_models()
# evaluate the models and store results
```

```
results , names = list() , list()
for name , model in models :
    # evaluate model
    scores =evaluate_model ( model ,X , y)
    # Store results
    results.append(scores)
    names.append(name)
    #summarize result
    print('>%s %.3f (%.3f)' % (name, mean(scores), std(scores)))
# plot model performance for comparison
pyplot.boxplot (results , labels=names , showmeans=True )
pyplot.show ()

#create the ensemble
ensemble = VotingClassifier( estimators = models , voting = 'soft')
# define the evaluation procedure
cv = RepeatedStratifiedKFold ( n_splits =10 , n_repeats =3 , random_state =1)
# evaluate the ensemble
scores = cross_val_score (ensemble , X, y, scoring ='accuracy', cv=cv , n_jobs=-1)
# summarize the result
print('Mean Accuracy: %.3f (%.3f)' % (mean(scores), std(scores)))
```

```
>lr 0.856 (0.014)
>tree 0.866 (0.013)
>nb 0.847 (0.021)
>svm 0.953 (0.010)
```




```
Mean Accuracy: 0.916 (0.013)
```

Ensemble Pruning Example

```
# evaluate a list of models
def evaluate_ensemble (models , X, y):
    # check for no models
    if len(models) == 0:
        return 0.0
    # create the ensemble
    ensemble = VotingClassifier(estimators = models , voting = 'soft')
    # define the evaluation procedure
    cv = RepeatedStratifiedKFold ( n_splits=10 , n_repeats=3 , random_state=1)
    # evaluate the ensemble
    scores = cross_val_score (ensemble , X, y, scoring ='accuracy', cv=cv ,n_jobs= -1)
    # return mean score
    return mean(scores)

# perform a single round of pruning the ensemble
def prune_round (models_in , X, y):
    # establish a baseline
    baseline = evaluate_ensemble (models_in , X, y)
    best_score , removed = baseline , None
    # enumerate removing each candidate and see if we can improve performance
    for m in models_in :
        # copy the list of chosen models
        dup = models_in.copy()
        # remove this model
        dup.remove(m)
        #evaluate new ensemble
        result = evaluate_ensemble (dup, X , y)
        # check for new best
        if result > best_score :
            # store the new best
            best_score , removed = result , m
    return best_score , removed
# prune an ensemble from scratch
def prune_ensemble ( models , X , y):
    best_score = 0.0
    # prune ensemble until no further improvement
    while True :
        # remove one model to the ensemble
        score , removed = prune_round(models ,X, y)
        # check for no improvement
        if removed is None :
            print ('>no further improvement')
            break
        # keep track of best score
        best_score = score
        # remove model from the list
        models.remove( removed )
        # report results along the way
        print('>%.3f (removed: %s)' % (score, removed[0]))
        return best_score , models

# define dataset
X, y = get_dataset ()
# get the models to evaluate
models = get_models ()
# prune the ensemble
score , model_list = prune_ensemble(models,X, y)
names = ','. join ([ n for n , _ in model_list ])
print ('Models : %s '% names )
print ('Final Mean Accuracy : %.3f' % score )
```

```
>0.922 (removed: lr)
Models : tree,nb,svm
Final Mean Accuracy : 0.922
```

Ensemble Growing Example

```
# perform a single round of growing the ensemble
def grow_round (models_in , models_candidate , X, y):
    # establish a baseline
    baseline = evaluate_ensemble (models_in , X, y)
    best_score , addition = baseline , None
    # enumerate adding each candidate and see if we can improve performance
    for m in models_candidate :
        # copy the list of chosen models
        dup = models_in.copy()
        # add the candidate
        dup.append(m)
        # evaluate new ensemble
        result = evaluate_ensemble(dup,X,y)
        # check for new best
        if result > best_score :
            # store the new best
            best_score , addition = result , m
    return best_score , addition

# grow an ensemble from scratch
def grow_ensemble (models , X, y):
    best_score , best_list = 0.0 , list()
    # grow ensemble until no further improvement
    while True :
        # add one model to the ensemble
        score , addition = grow_round (best_list ,models ,X, y)
        # check for no improvement
        if addition is None :
            print ('>no further improvement')
            break
        # keep track of best score
        best_score = score
        # remove new model from the list of candidates
        models.remove(addition)
        # add new model to the list of models in the ensemble
        best_list.append(addition)
        # report results along the way
        names = ','. join ([n for n,_ in best_list ])
        print ('>%.3f (%s)' % (score , names ))
    return best_score , best_list

# define dataset
X, y = get_dataset ()
# get the models to evaluate
models = get_models ()
# prune the ensemble
score , model_list = grow_ensemble(models,X, y)
```

```
names = ','. join ([ n for n , _ in model_list ])
print ('Models : %s '% names )
print ('Final Mean Accuracy : %.3f' % score )
```

```
>0.953 (svm)
>no further improvement
Models : svm
Final Mean Accuracy : 0.953
```

Task 2

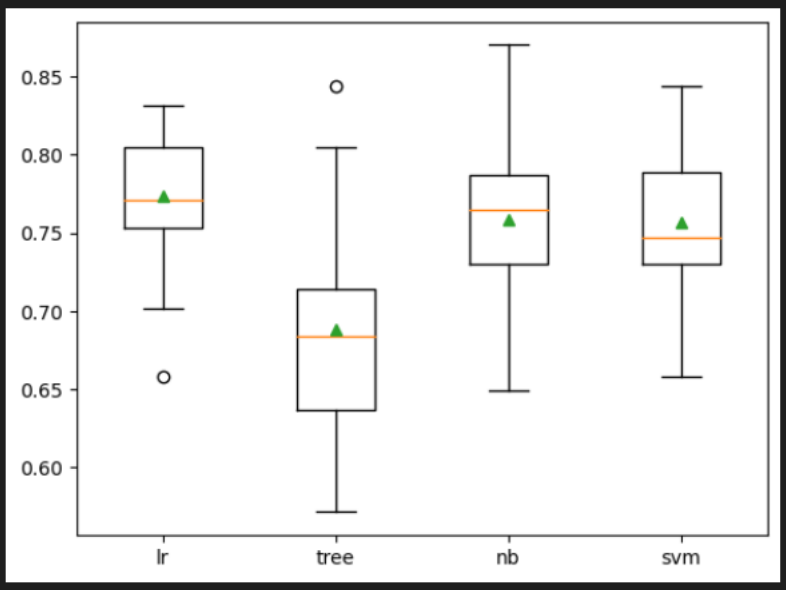
Train and evaluate models as discussed in Section 2, 3, and 4 on Diabetes data set select the bestmodel based on evaluation scores.

```
 # evaluate standard models on the synthetic dataset
from numpy import mean
from numpy import std
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from matplotlib import pyplot
from sklearn.ensemble import VotingClassifier
from sklearn.datasets import load_diabetes
import pandas as pd
# get the dataset

def get_dataset () :
    # Import the dataset
    data = pd.read_csv('diabetes.csv')
    # Extracting features and target variable
    X = data.drop("Outcome", axis=1)
    y = data['Outcome']

    return X, y
# get a list of models to evaluate
def get_models ():
    models = list()
    models.append(('lr', LogisticRegression()))
    models.append(('tree', DecisionTreeClassifier()))
    models.append (('nb', GaussianNB()))
    models.append(('svm', SVC(probability=True)))
    return models

# evaluate a give model using cross - validation
def evaluate_model (model , X, y):
    # define the model evaluation procedure
    cv = RepeatedStratifiedKFold( n_splits=10 , n_repeats=3 , random_state=1)
    # evaluate the model
    scores = cross_val_score(model, X, y, scoring='accuracy', cv=cv, n_jobs= -1)
```

```
    return scores
# define dataset
X, y = get_dataset()
# get the models to evaluate
models = get_models()
# evaluate the models and store results
results , names = list() , list()
for name , model in models :
    # evaluate model
    scores =evaluate_model ( model ,X , y)
    # Store results
    results.append(scores)
    names.append(name)
    #summarize result
    print('>%s %.3f (%.3f)' % (name, mean(scores), std(scores)))
# plot model performance for comparison
pyplot.boxplot (results , labels=names , showmeans=True )
pyplot.show ()

#create the ensemble
ensemble = VotingClassifier( estimators = models , voting = 'soft')
# define the evaluation procedure
cv = RepeatedStratifiedKFold ( n_splits =10 , n_repeats =3 , random_state =1)
# evaluate the ensemble
scores = cross_val_score (ensemble , X, y, scoring ='accuracy', cv=cv , n_jobs=-1)
# summarize the result
print('Mean Accuracy: %.3f (%.3f)' % (mean(scores), std(scores)))
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

```
>lr 0.773 (0.040)
>tree 0.688 (0.059)
>nb 0.759 (0.055)
>svm 0.757 (0.040)
```




```
# evaluate a list of models
def evaluate_ensemble (models , X, y):
```

```
    # check for no models
    if len(models) == 0:
        return 0.0
    # create the ensemble
    ensemble = VotingClassifier(estimators = models , voting = 'soft')
    # define the evaluation procedure
    cv = RepeatedStratifiedKFold ( n_splits=10 , n_repeats=3 , random_state=1)
    # evaluate the ensemble
    scores = cross_val_score (ensemble , X, y, scoring ='accuracy', cv=cv ,n_jobs= -1)
    # return mean score
    return mean(scores)

# perform a single round of pruning the ensemble
def prune_round (models_in , X, y):
    # establish a baseline
    baseline = evaluate_ensemble (models_in , X, y)
    best_score , removed = baseline , None
    # enumerate removing each candidate and see if we can improve performance
    for m in models_in :
        # copy the list of chosen models
        dup = models_in.copy()
        # remove this model
        dup.remove(m)
        #evaluate new ensemble
        result = evaluate_ensemble (dup, X , y)
        # check for new best
        if result > best_score :
            # store the new best
            best_score , removed = result , m
    return best_score , removed

# prune an ensemble from scratch
def prune_ensemble ( models , X , y):
    best_score = 0.0
    # prune ensemble until no further improvement
    while True :
        # remove one model to the ensemble
        score , removed = prune_round(models ,X, y)
        # check for no improvement
        if removed is None :
            print ('>no further improvement')
            break
        # keep track of best score
        best_score = score
        # remove model from the list
        models.remove( removed )
        # report results along the way
        print('>%.3f (removed: %s)' % (score, removed[0]))
        return best_score , models
```

```
# define dataset
X, y = get_dataset ()
# get the models to evaluate
models = get_models ()
# prune the ensemble
score , model_list = prune_ensemble(models,X, y)
names = ','. join ([ n for n , _ in model_list ])
print ('Models : %s '% names )
print ('Final Mean Accuracy : %.3f' % score )
# perform a single round of growing the ensemble
def grow_round (models_in , models_candidate , X, y):
    # establish a baseline
    baseline = evaluate_ensemble (models_in , X, y)
    best_score , addition = baseline , None
    # enumerate adding each candidate and see if we can improve performance
    for m in models_candidate :
        # copy the list of chosen models
        dup = models_in.copy()
        # add the candidate
        dup.append(m)
        # evaluate new ensemble
        result = evaluate_ensemble(dup,X,y)
        # check for new best
        if result > best_score :
            # store the new best
            best_score , addition = result , m
    return best_score , addition

# grow an ensemble from scratch
def grow_ensemble (models , X, y):
    best_score , best_list = 0.0 , list()
    # grow ensemble until no further improvement
    while True :
        # add one model to the ensemble
        score , addition = grow_round (best_list ,models ,X, y)
        # check for no improvement
        if addition is None :
            print ('>no further improvement')
            break
        # keep track of best score
        best_score = score
        # remove new model from the list of candidates
        models.remove(addition)
        # add new model to the list of models in the ensemble
        best_list.append(addition)
        # report results along the way
        names = ','. join ([n for n,_ in best_list ])
        print ('>%.3f (%s)' % (score , names ))
    return best_score , best_list
```

```
# define dataset
X, y = get_dataset ()
# get the models to evaluate
models = get_models ()
# prune the ensemble
score , model_list = grow_ensemble(models,X, y)
names = ','. join ([ n for n , _ in model_list ])
print ('Models : %s '% names )
print ('Final Mean Accuracy : %.3f' % score )
```

```
/opt/anaconda3/envs/myenv/lib/python3.9/site-packages/sklearn/linear_model/_logistic.py:460: ConvergenceWarning: lbfgs fai
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/opt/anaconda3/envs/myenv/lib/python3.9/site-packages/sklearn/linear_model/_logistic.py:460: ConvergenceWarning: lbfgs fai
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/opt/anaconda3/envs/myenv/lib/python3.9/site-packages/sklearn/linear_model/_logistic.py:460: ConvergenceWarning: lbfgs fai
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/opt/anaconda3/envs/myenv/lib/python3.9/site-packages/sklearn/linear_model/_logistic.py:460: ConvergenceWarning: lbfgs fai
...
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
```

Student Name:_____Roll No:_____Section:_____

CS334 - Machine Learning

Lab 09 - Ensemble Models (Part-2)

Instructor: Ms. Maham Ashraf E-mail:
mashraf@uit.edu

Semester: Fall, 2023

Objective

The purpose of this lab session is to introduce Ensemble models that use in Machine Learning for better performance.

Instructions

You have to perform the following tasks yourselves. Raise your hand if you face any difficulty in understanding and solving these tasks. **Plagiarism** is an abhorrent practice and you should not engage in it.

How to Submit

- Submit lab work in a single .py file on Microsoft Teams. (No other format will be accepted)
  - Lab work file name should be saved with your roll number (e.g. 19a-001-SE_LW01.py)
- Submit home work in a single .py file on Microsoft Teams. (No other format will be accepted)
  - Home work file name should be saved with your roll number (e.g. 19a-001-SE_HW01.py)

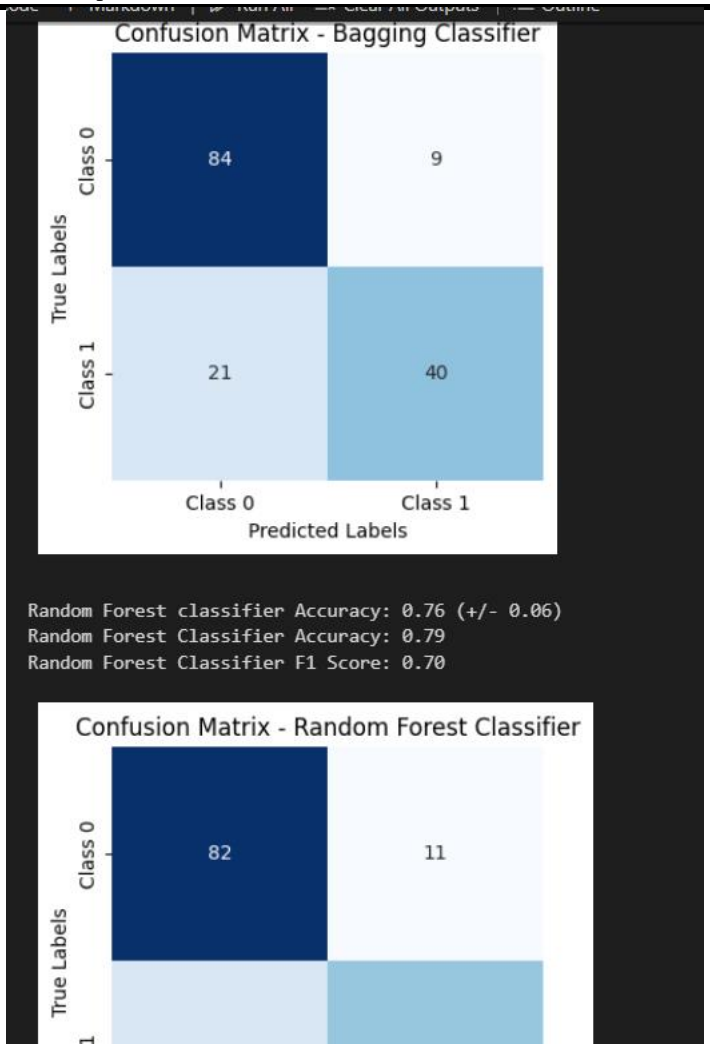
Lab Tasks

1. Use *BaggingClassifier* and *RandomForestClassifier* from *sklearn* library and implement them on *diabetes* data set.

```
from sklearn.model_selection import KFold, cross_val_score
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn . ensemble import RandomForestClassifier
from sklearn.datasets import load_diabetes
from sklearn.metrics import f1_score, accuracy_score
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
```

```
import pandas as pd

dataframe = pd.read_csv("diabetes.csv")
X = dataframe.drop('Outcome', axis=1)
y = dataframe['Outcome']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=2020)
max_features = 3
kfold = KFold(n_splits=10, shuffle=True, random_state=2020)
decision_tree = DecisionTreeClassifier(max_features = max_features)
num_trees = 100

# Decision Tree base estimator
bagging_model = BaggingClassifier(base_estimator=decision_tree, n_estimators = num_trees, random_state=2020)

results = cross_val_score(bagging_model, X_train, y_train, cv=kfold)
print("Bagging classifier Accuracy: %0.2f (+/- %0.2f)" % (results.mean(), results.std()))

# Bagging Classifier with Decision Tree base estimator
bagging_model.fit(X_train, y_train)
y_pred_bagging = bagging_model.predict(X_test)

accuracy_bagging = accuracy_score(y_test, y_pred_bagging)
f1_bagging = f1_score(y_test, y_pred_bagging)

print("Bagging Classifier Accuracy: %0.2f" % accuracy_bagging)
print("Bagging Classifier F1 Score: %0.2f" % f1_bagging)

# Bagging Classifier with Decision Tree base estimator
conf_matrix_bagging = confusion_matrix(y_test, y_pred_bagging)

# Plot confusion matrix for Bagging Classifier
plt.figure(figsize=(4, 4))
sns.heatmap(conf_matrix_bagging, annot=True, fmt="d", cmap="Blues", cbar=False,
            xticklabels=['Class 0', 'Class 1'], yticklabels=['Class 0', 'Class 1'])
plt.title("Confusion Matrix - Bagging Classifier")
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.show()

# Random Forest Classifier
num_trees_rf = 100
max_features_rf = 3
kfold_rf = KFold(n_splits=10, shuffle=True, random_state=2020)

```

```
rf_model = RandomForestClassifier(n_estimators=num_trees_rf, max_features=
max_features)
# Cross-validation
results_rf = cross_val_score(rf_model, X_train, y_train, cv=kfold_rf)
print("Random Forest classifier Accuracy: %0.2f (+/- %0.2f)" %
(results_rf.mean(),results_rf.std()))

#accuracy and F1 score
rf_model.fit(X_train, y_train)
y_pred_rf = rf_model.predict(X_test)

accuracy_rf = accuracy_score(y_test, y_pred_rf)
f1_rf = f1_score(y_test, y_pred_rf)

print("Random Forest Classifier Accuracy: %0.2f" % accuracy_rf)
print("Random Forest Classifier F1 Score: %0.2f" % f1_rf)

# Random Forest Classifier
conf_matrix_rf = confusion_matrix(y_test, y_pred_rf)

# Plot confusion matrix for Random Forest Classifier
plt.figure(figsize=(4, 4))
sns.heatmap(conf_matrix_rf, annot=True, fmt="d", cmap="Blues", cbar=False,
            xticklabels=['Class 0', 'Class 1'], yticklabels=['Class 0', 'Class
1'])
plt.title("Confusion Matrix - Random Forest Classifier")
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.show()
```

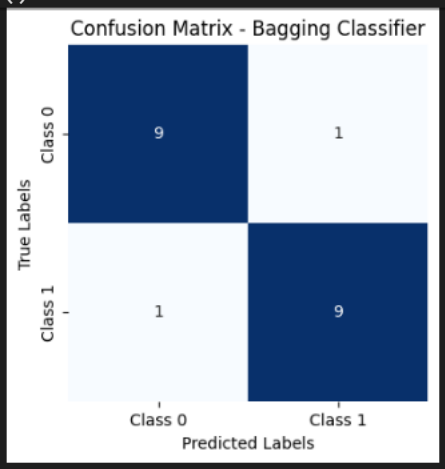


2Compare the results of Task - 1 classification tasks using accuracy, F1, and confusion matrix

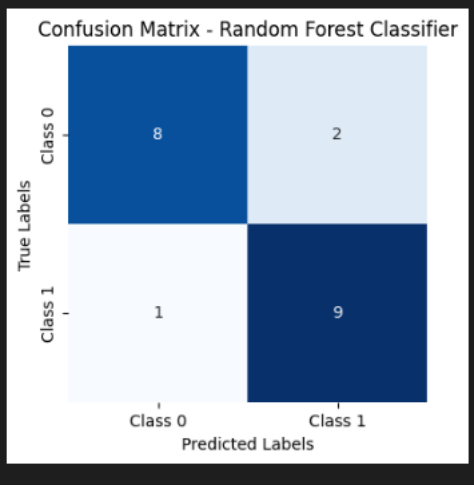
```
from sklearn.model_selection import KFold, cross_val_score
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.datasets import make_classification
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import f1_score, accuracy_score
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

X, y = make_classification(n_samples=100, n_features=20, n_classes=2,
random_state=2020)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=2020)
max_features = 3
kfold = KFold(n_splits=10, shuffle=True, random_state=2020)
decision_tree = DecisionTreeClassifier(max_features = max_features)
num_trees = 100

# Decision Tree base estimator
```

```
bagging_model = BaggingClassifier(base_estimator=decision_tree, n_estimators =
num_trees, random_state=2020)

# Cross-validation
results = cross_val_score(bagging_model, X_train, y_train, cv=kfold)
print("Bagging Classifier cross validation Accuracy: %0.2f (+/- %0.2f)" %
(results.mean(), results.std()))

# Bagging Classifier with Decision Tree base estimator
bagging_model.fit(X_train, y_train)
y_pred_bagging = bagging_model.predict(X_test)

accuracy_bagging = accuracy_score(y_test, y_pred_bagging)
f1_bagging = f1_score(y_test, y_pred_bagging)

print("Bagging Classifier Accuracy: %0.2f" % accuracy_bagging)
print("Bagging Classifier F1 Score: %0.2f" % f1_bagging)

# Bagging Classifier with Decision Tree base estimator
conf_matrix_bagging = confusion_matrix(y_test, y_pred_bagging)

# Plot confusion matrix for Bagging Classifier
plt.figure(figsize=(4, 4))
sns.heatmap(conf_matrix_bagging, annot=True, fmt="d", cmap="Blues", cbar=False,
            xticklabels=['Class 0', 'Class 1'], yticklabels=['Class 0', 'Class 1'])
plt.title("Confusion Matrix - Bagging Classifier")
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.show()

# Random Forest Classifier
num_trees_rf = 100
max_features_rf = 3
kfold_rf = KFold(n_splits=10, shuffle=True, random_state=2020)

rf_model = RandomForestClassifier(n_estimators=num_trees_rf, max_features=
max_features)

# Cross-validation
results_rf = cross_val_score(rf_model, X_train, y_train, cv=kfold_rf)
print("Random Forest Accuracy: %0.2f (+/- %0.2f)" %
(results_rf.mean(),results_rf.std()))

#accuracy and F1 score
rf_model.fit(X_train, y_train)
y_pred_rf = rf_model.predict(X_test)

accuracy_rf = accuracy_score(y_test, y_pred_rf)
f1_rf = f1_score(y_test, y_pred_rf)
```

```
print("Random Forest Classifier Accuracy: %0.2f" % accuracy_rf)
print("Random Forest Classifier F1 Score: %0.2f" % f1_rf)

# Random Forest Classifier
conf_matrix_rf = confusion_matrix(y_test, y_pred_rf)

# Plot confusion matrix for Random Forest Classifier
plt.figure(figsize=(4, 4))
sns.heatmap(conf_matrix_rf, annot=True, fmt="d", cmap="Blues", cbar=False,
            xticklabels=['Class 0', 'Class 1'], yticklabels=['Class 0', 'Class 1'])
plt.title("Confusion Matrix - Random Forest Classifier")
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.show()
```




```
Random Forest Accuracy: 0.81 (+/- 0.20)
Random Forest Classifier Accuracy: 0.85
Random Forest Classifier F1 Score: 0.86
```

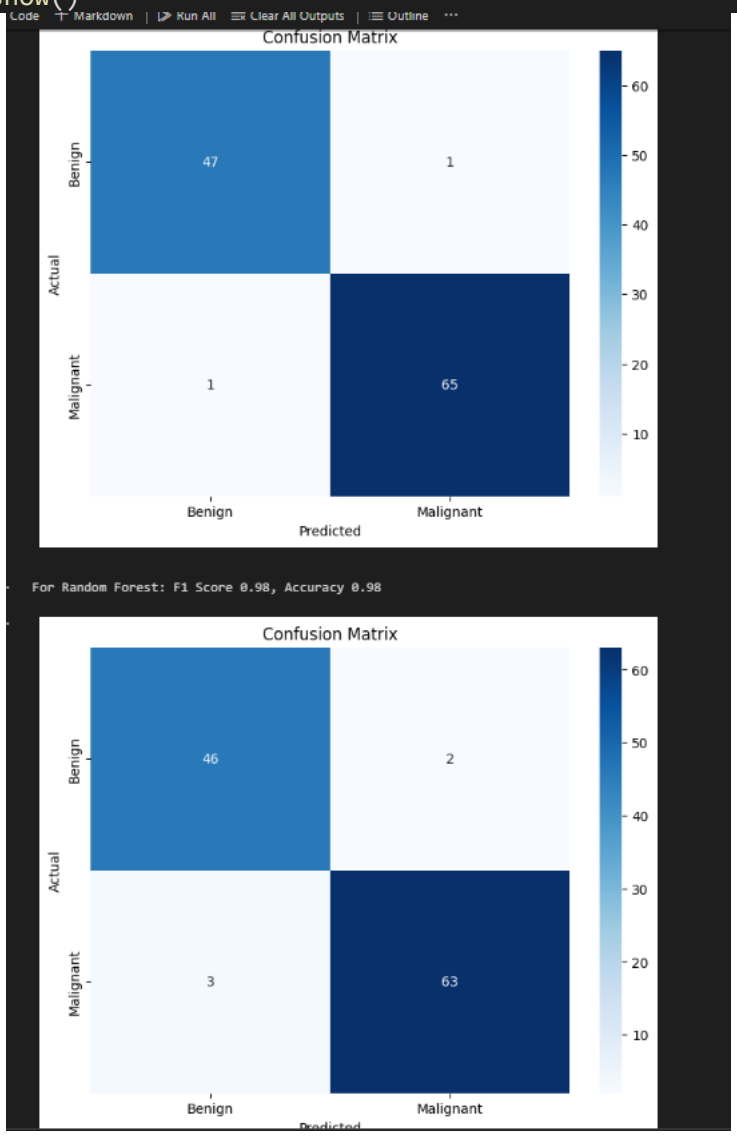


3Use Use *AdaBoostClassifier* and *RandomForstClassifier* from *sklearn* library and implment both on *Breast Cancer* data set.

```
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import f1_score, accuracy_score
from sklearn.datasets import load_breast_cancer
from sklearn.metrics import confusion_matrix
```

```
from sklearn.ensemble import RandomForestClassifier
import seaborn as sns
import matplotlib.pyplot as plt

cancer = load_breast_cancer()
X = cancer.data
y = cancer.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=2020)

# Ada Boost classifier

clf_boosting = AdaBoostClassifier( base_estimator=DecisionTreeClassifier(max_depth=1),
n_estimators=200
)
# Fit the model
clf_boosting.fit(X_train, y_train)
# Make predictions
predictions = clf_boosting.predict(X_test)
# Calculate and print F1 Score and Accuracy
print("For Boosting: F1 Score {}, Accuracy {}".format(
    round(f1_score(y_test, predictions), 2),
    round(accuracy_score(y_test, predictions), 2)
))

# Confusion Matrix
conf_mat = confusion_matrix(y_test, predictions)

# Plot Confusion Matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_mat, annot=True, fmt="d", cmap="Blues", xticklabels=["Benign",
"Malignant"], yticklabels=["Benign", "Malignant"])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()

# Random forest classifier

# Model training using RandomForestClassifier as an example
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)
# Make predictions
rf_predictions = rf_model.predict(X_test)
# Calculate and print F1 Score and Accuracy
print("For Random Forest: F1 Score {}, Accuracy {}".format(
    round(f1_score(y_test, predictions), 2),
    round(accuracy_score(y_test, predictions), 2)
))
```

```
# Confusion Matrix
conf_mat = confusion_matrix(y_test, rf_predictions)

# Plot Confusion Matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_mat, annot=True, fmt="d", cmap="Blues", xticklabels=["Benign",
"Malignant"], yticklabels=["Benign", "Malignant"])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

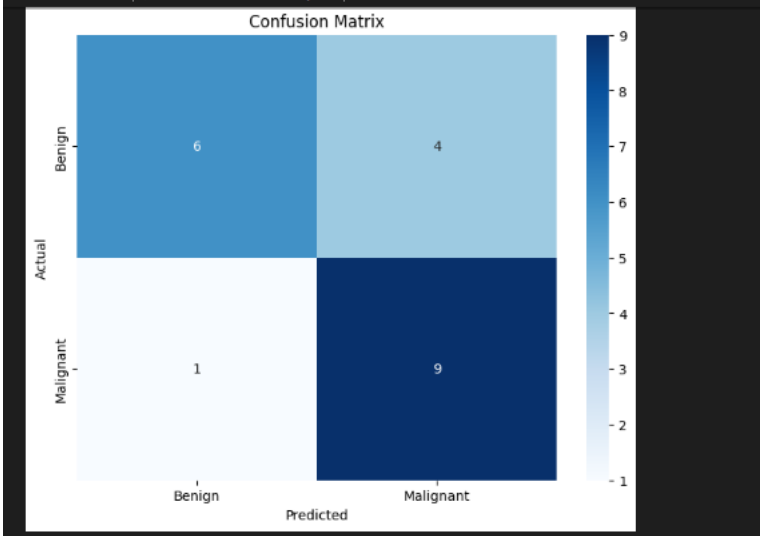


4Compare the results of Task - 3 classification tasks using accuracy, F1, and confusion matrix.

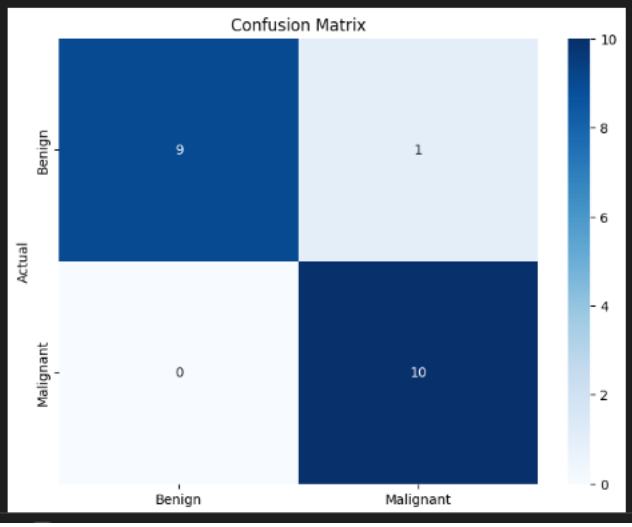
```
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import f1_score, accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.ensemble import RandomForestClassifier
```

```
import seaborn as sns
import matplotlib.pyplot as plt

X, y = make_classification(n_samples=100, n_features=20, n_classes=2,
random_state=2020)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=2020)

clf_boosting = AdaBoostClassifier( base_estimator=DecisionTreeClassifier(max_depth=1),
n_estimators=200
)
# Fit the model
clf_boosting.fit(X_train, y_train)
# Make predictions
predictions = clf_boosting.predict(X_test)
# Calculate and print F1 Score and Accuracy
print("For Boosting: F1 Score {}, Accuracy {}".format(
    round(f1_score(y_test, predictions), 2),
    round(accuracy_score(y_test, predictions), 2)
))

# Confusion Matrix
conf_mat = confusion_matrix(y_test, predictions)

# Plot Confusion Matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_mat, annot=True, fmt="d", cmap="Blues", xticklabels=["Benign",
"Malignant"], yticklabels=["Benign", "Malignant"])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()

# Random forest classifier

# Model training using RandomForestClassifier as an example
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)
# Make predictions
rf_predictions = rf_model.predict(X_test)
# Calculate and print F1 Score and Accuracy
print("For Random Forest: F1 Score {}, Accuracy {}".format(
    round(f1_score(y_test, predictions), 2),
    round(accuracy_score(y_test, predictions), 2)
))

# Confusion Matrix
conf_mat = confusion_matrix(y_test, rf_predictions)

# Plot Confusion Matrix
```

```
plt.figure(figsize=(8, 6))
sns.heatmap(conf_mat, annot=True, fmt="d", cmap="Blues", xticklabels=["Benign",
"Malignant"], yticklabels=["Benign", "Malignant"])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```




```
For Random Forest: F1 Score 0.78, Accuracy 0.75
```



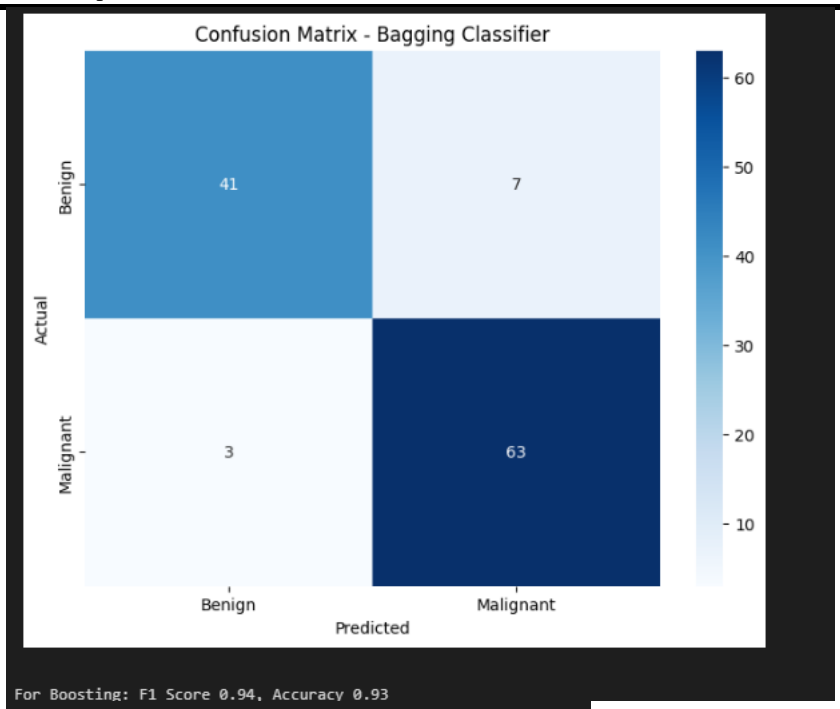
5To check and compare the performance of *Bagging, Boosting, and Stacking* classification algorithms, implement *AdaBoostClassifier, RandomForestClassifier, and LogisticRegrassion* (for Stacking) on *Breast Cancer* data set.

```
import numpy as np
from sklearn.ensemble import AdaBoostClassifier, RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import f1_score, accuracy_score
from sklearn.datasets import load_breast_cancer
from sklearn.metrics import confusion_matrix
from sklearn.ensemble import RandomForestClassifier
import seaborn as sns
import matplotlib.pyplot as plt
```

```
class NumberOfClassifierException(Exception):
    pass

class Stacking():
    def __init__(self, classifiers):
        if len(classifiers) < 2:
            raise numberOfClassifierException (" You must fit your classifier with 2
classifiers at least ");
        else:
            self._classifiers = classifiers

    def fit(self, data_x, data_y):
        stacked_data_x = data_x.copy()
        for classifier in self._classifiers[:-1]:
            classifier.fit(data_x, data_y)
            stacked_data_x = np.column_stack((stacked_data_x
,classifier.predict_proba(data_x)))
        last_classifier = self._classifiers[-1]
        last_classifier.fit(stacked_data_x, data_y)

    def predict(self, data_x):
        stacked_data_x = data_x.copy()
        for classifier in self._classifiers[:-1]:
            prob_predictions = classifier.predict_proba(data_x)
            stacked_data_x = np.column_stack ((stacked_data_x , prob_predictions))
        last_classifier = self._classifiers[-1]
        return last_classifier.predict(stacked_data_x)

cancer = load_breast_cancer()
X = cancer.data
y = cancer.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=2020)
# Creating classifiers
boosting_clf_ada_boost = AdaBoostClassifier(
base_estimator=DecisionTreeClassifier(max_depth=1), n_estimators=3)
clf_rf = RandomForestClassifier( n_estimators=200, max_depth=1, random_state=2020)
clf_adaboost = AdaBoostClassifier( base_estimator=DecisionTreeClassifier(max_depth=1,
random_state=2020), n_estimators=3)

clf_logistic_reg = LogisticRegression(solver='liblinear', random_state=2020)

# Customizing and Exception message
classifiers_list = [clf_rf, clf_adaboost, clf_logistic_reg]
clf_stacking = Stacking(classifiers_list)
# Fit models
clf_rf.fit(X_train, y_train)
```

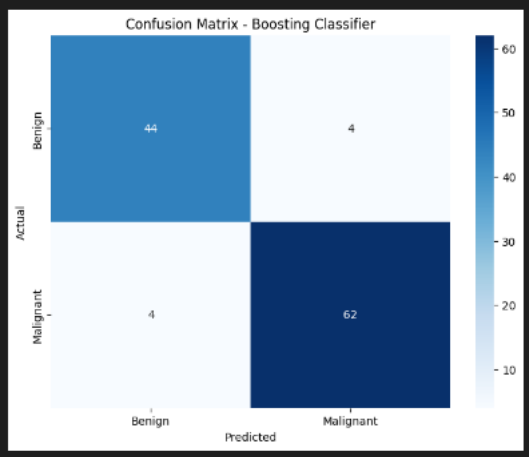
```
boosting_clf_ada_boost.fit(X_train, y_train)
clf_stacking.fit(X_train, y_train)

# Make predictions
predictions_bagging = clf_rf.predict(X_test)
predictions_boosting = boosting_clf_ada_boost.predict(X_test)
predictions_stacking = clf_stacking.predict(X_test)

# Print results
print("For Bagging: F1 Score {}, Accuracy {}".format(
    round(f1_score(y_test, predictions_bagging), 2),
    round(accuracy_score(y_test, predictions_bagging), 2)
))

# Confusion Matrix for Bagging
conf_mat_bagging = confusion_matrix(y_test, predictions_bagging)

# Plot Confusion Matrix for Bagging
plt.figure(figsize=(8, 6))
sns.heatmap(conf_mat_bagging, annot=True, fmt="d", cmap="Blues", xticklabels=["Benign",
"Malignant"], yticklabels=["Benign", "Malignant"])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix - Bagging Classifier')
plt.show()

print("For Boosting: F1 Score {}, Accuracy {}".format(
    round(f1_score(y_test, predictions_boosting), 2),
    round(accuracy_score(y_test, predictions_boosting), 2)
))

# Confusion Matrix for Bagging
conf_mat_bagging = confusion_matrix(y_test, predictions_boosting)

# Plot Confusion Matrix for Bagging
plt.figure(figsize=(8, 6))
sns.heatmap(conf_mat_bagging, annot=True, fmt="d", cmap="Blues", xticklabels=["Benign",
"Malignant"], yticklabels=["Benign", "Malignant"])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix - Boosting Classifier')
plt.show()

print("For Stacking: F1 Score {}, Accuracy {}".format(
    round(f1_score(y_test, predictions_stacking), 2),
    round(accuracy_score(y_test, predictions_stacking), 2)
))

# Confusion Matrix for Bagging
conf_mat_bagging = confusion_matrix(y_test, predictions_stacking)
```

```
# Plot Confusion Matrix for Bagging
plt.figure(figsize=(8, 6))
sns.heatmap(conf_mat_bagging, annot=True, fmt="d", cmap="Blues", xticklabels=["Benign",
"Malignant"], yticklabels=["Benign", "Malignant"])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix - Stacking Classifier')
plt.show()
```

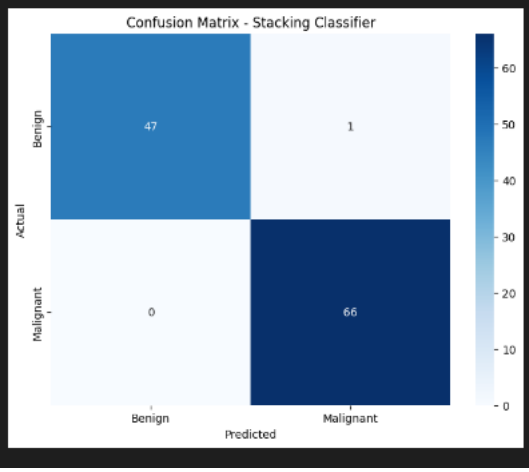


For Boosting: F1 Score 0.94, Accuracy 0.93

For Boosting: F1 Score 0.94, Accuracy 0.93



For Stacking: F1 Score 0.99, Accuracy 0.99



6Compare the results of Task - 5 classification tasks using accuracy, F1, and confusion matrix.

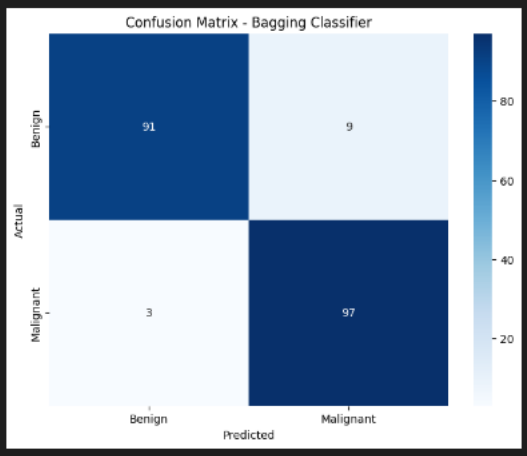
```
import numpy as np
from sklearn.ensemble import AdaBoostClassifier, RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
```

```
from sklearn.metrics import f1_score, accuracy_score
from sklearn.datasets import load_breast_cancer
from sklearn.metrics import confusion_matrix
from sklearn.ensemble import RandomForestClassifier
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.datasets import make_classification

class NumberOfClassifierException(Exception):
    pass

class Stacking():
    def __init__(self, classifiers):
        if len(classifiers) < 2:
            raise numberOfClassifierException (" You must fit your classifier with 2
classifiers at least ");
        else:
            self._classifiers = classifiers

    def fit(self, data_x, data_y):
        stacked_data_x = data_x.copy()
        for classifier in self._classifiers[:-1]:
            classifier.fit(data_x, data_y)
            stacked_data_x = np.column_stack((stacked_data_x
,classifier.predict_proba(data_x)))
        last_classifier = self._classifiers[-1]
        last_classifier.fit(stacked_data_x, data_y)

    def predict(self, data_x):
        stacked_data_x = data_x.copy()
        for classifier in self._classifiers[:-1]:
            prob_predictions = classifier.predict_proba(data_x)
            stacked_data_x = np.column_stack ((stacked_data_x , prob_predictions))
        last_classifier = self._classifiers[-1]
        return last_classifier.predict(stacked_data_x)

X, y = make_classification(n_samples=1000, n_features=20, n_classes=2,
random_state=2020)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=2020)
# Creating classifiers
boosting_clf_ada_boost = AdaBoostClassifier(
base_estimator=DecisionTreeClassifier(max_depth=1), n_estimators=3)
clf_rf = RandomForestClassifier( n_estimators=200, max_depth=1, random_state=2020)
clf_adaboost = AdaBoostClassifier( base_estimator=DecisionTreeClassifier(max_depth=1,
random_state=2020), n_estimators=3)

clf_logistic_reg = LogisticRegression(solver='liblinear', random_state=2020)

# Customizing and Exception message
```

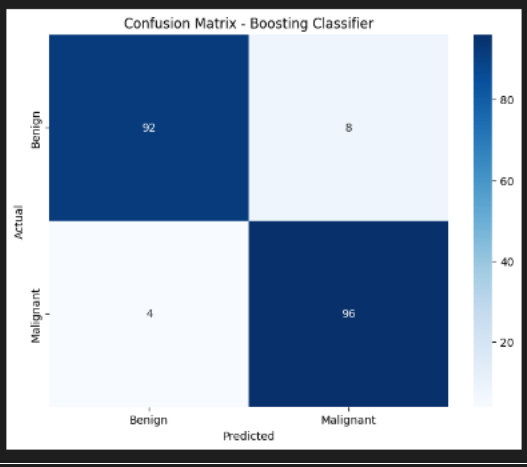
```
classifiers_list = [clf_rf, clf_adaboost, clf_logistic_reg]
clf_stacking = Stacking(classifiers_list)
# Fit models
clf_rf.fit(X_train, y_train)
boosting_clf_ada_boost.fit(X_train, y_train)
clf_stacking.fit(X_train, y_train)

# Make predictions
predictions_bagging = clf_rf.predict(X_test)
predictions_boosting = boosting_clf_ada_boost.predict(X_test)
predictions_stacking = clf_stacking.predict(X_test)

# Print results
print("For Bagging: F1 Score {}, Accuracy {}".format(
    round(f1_score(y_test, predictions_bagging), 2),
    round(accuracy_score(y_test, predictions_bagging), 2)
))

# Confusion Matrix for Bagging
conf_mat_bagging = confusion_matrix(y_test, predictions_bagging)

# Plot Confusion Matrix for Bagging
plt.figure(figsize=(8, 6))
sns.heatmap(conf_mat_bagging, annot=True, fmt="d", cmap="Blues", xticklabels=["Benign",
"Malignant"], yticklabels=["Benign", "Malignant"])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix - Bagging Classifier')
plt.show()

print("For Boosting: F1 Score {}, Accuracy {}".format(
    round(f1_score(y_test, predictions_boosting), 2),
    round(accuracy_score(y_test, predictions_boosting), 2)
))

# Confusion Matrix for Bagging
conf_mat_bagging = confusion_matrix(y_test, predictions_boosting)

# Plot Confusion Matrix for Bagging
plt.figure(figsize=(8, 6))
sns.heatmap(conf_mat_bagging, annot=True, fmt="d", cmap="Blues", xticklabels=["Benign",
"Malignant"], yticklabels=["Benign", "Malignant"])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix - Boosting Classifier')
plt.show()

print("For Stacking: F1 Score {}, Accuracy {}".format(
    round(f1_score(y_test, predictions_stacking), 2),
    round(accuracy_score(y_test, predictions_stacking), 2)
```

```
))

# Confusion Matrix for Bagging
conf_mat_bagging = confusion_matrix(y_test, predictions_stacking)

# Plot Confusion Matrix for Bagging
plt.figure(figsize=(8, 6))
sns.heatmap(conf_mat_bagging, annot=True, fmt="d", cmap="Blues", xticklabels=["Benign",
"Malignant"], yticklabels=["Benign", "Malignant"])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix - Stacking Classifier')
plt.show()
```

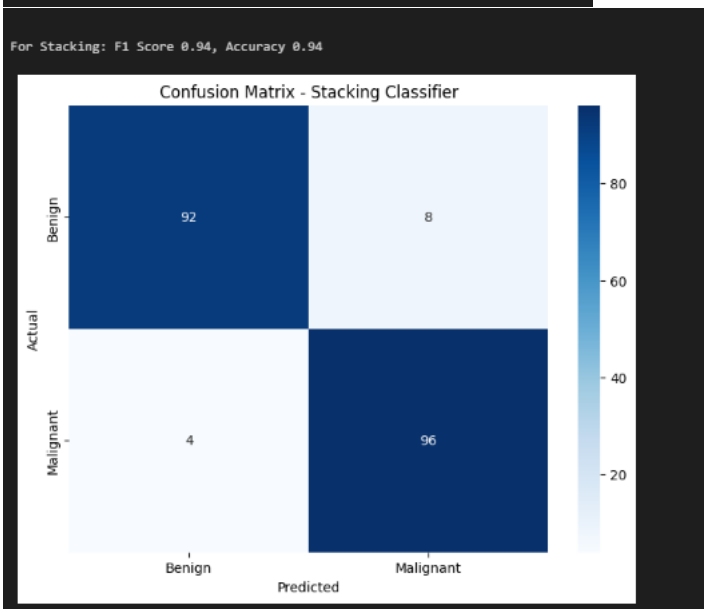
```
  warnings.warn(
For Bagging: F1 Score 0.94, Accuracy 0.94
```




```
For Boosting: F1 Score 0.94, Accuracy 0.94
```




```
For Stacking: F1 Score 0.94, Accuracy 0.94
```



Student Name: sawera fazal Roll No: 21A-026-se Section: 21A

CS334 - Machine Learning

Lab 10 - Probability Based Learning

Instructor: Ms. Maham Ashraf E-mail: mashraf@uit.edu

Semester: Fall, 2023

Objective

The purpose of this lab session is to introduce probability based learning models, such as Naive-Bayes classifier for categorical and continues features, and Naive Bayes classifier with Laplace Smoothing.

Instructions

You have to perform the following tasks yourselves. Raise your hand if you face any difficulty in understanding and solving these tasks. **Plagiarism** is an abhorrent practice and you should not engage in it.

How to Submit

- Submit lab work in a single .py file on Microsoft Teams. (No other format will be accepted)
- Lab work file name should be saved with your roll number (e.g. 19a-001-SE_LW01.py)
- Submit home work in a single .py file on Microsoft Teams. (No other format will be accepted)
- Home work file name should be saved with your roll number (e.g. 19a-001-SE_HW01.py)

3 Lab Tasks

1. Use the hints as provided in Section -2 for implementing Naive Bayes classifier for categorical variable and implement a class NBClassifier.

```
from sklearn.preprocessing import LabelEncoder
import pandas as pd

class NaiveBayesClassifier:
    def __init__(self, data, target):
        # Constructor to initialize the classifier with input data and target variable
        self.data = data
        self.target = target
        self.target_prob_dict = {}  # Dictionary to store target class probabilities
        self.cond_prob_list = []  # List to store conditional probabilities
```

```
        self.smoothing = False  # Flag to enable/disable Laplace smoothing
        self.k = 1  # Laplace Smoothing parameter

    def compute_target_probabilities(self):
        # Compute probabilities of each target class in the dataset
        total_rows = len(self.data)
        target_counts = self.data[self.target].value_counts()

        for level, count in target_counts.items():
            self.target_prob_dict[level] = count / total_rows

    def compute_conditional_probabilities(self):
        # Compute conditional probabilities for each feature given the target class
        for feature in self.data.columns:
            if feature != self.target:
                feature_levels = self.data[feature].unique()

                for level in feature_levels:
                    for target_level in self.target_prob_dict.keys():
                        count_f_v_t = len(self.data[(self.data[feature] == level) & (self.data[self.target] == target_level)])
                        count_f_t = len(self.data[self.data[self.target] == target_level])

                        if self.smoothing:
                            # Laplace Smoothing
                            prob = (count_f_v_t + self.k) / (count_f_t + self.k * len(feature_levels))
                        else:
                            prob = count_f_v_t / count_f_t

                        # Store the computed probability in the list
                        self.cond_prob_list.append((feature, level, target_level, round(prob, 3)))

    def fit(self, smoothing=False, k=1):
        # Method to train the Naive Bayes classifier
        self.smoothing = smoothing
        self.k = k

        self.compute_target_probabilities()
        self.compute_conditional_probabilities()

    def predict_instance(self, instance):
        # Predict the target class for a given instance
        prob_dict = {}

        for target_level in self.target_prob_dict.keys():
            prob_prod = 1
```

```
57.
58.             for feature, level, t_level, prob in self.cond_prob_list:
59.                 if feature in instance and instance[feature] == level and t_level
  == target_level:
60.                     prob_prod *= prob
61.
62.             # Multiply the conditional probability with the prior probability
63.             prob_dict[target_level] = round(prob_prod *
  self.target_prob_dict[target_level], 4)
64.
65.         # Return the predicted target class with the highest probability
66.         return max(prob_dict, key=prob_dict.get)
67.
68.     def get_probabilities(self):
69.         # Display the computed target and conditional probabilities
70.         print("Target probabilities:", self.target_prob_dict)
71.         print("Conditional features probabilities:", self.cond_prob_list)
72.

73. # Example usage:
74. data = {
75.     'CrdtHistory': ['current', 'paid', 'arrears', 'none', 'current', 'paid',
  'arrears', 'none'],
76.     'GCoApplicant': ['none', 'guarantor', 'coapplicant', 'none', 'none',
  'guarantor', 'coapplicant', 'none'],
77.     'Accommodation': ['own', 'rent', 'free', 'own', 'rent', 'free', 'own',
  'rent'],
78.     'Target': [True, True, True, True, False, False, False, False]
79. }
80.
81. df = pd.DataFrame(data)
82.
83. nbc = NaiveBayesClassifier(df, 'Target')
84. nbc.fit(smoothing=True, k=1)
85.
86. # Prediction for a new instance
87. query_instance = {'CrdtHistory': 'current', 'GCoApplicant': 'none',
  'Accommodation': 'own'}
88. prediction = nbc.predict_instance(query_instance)
89. print("Prediction:", prediction)
90.
91. # Display all probabilities
92. nbc.get_probabilities()
```

```
Prediction: True
Target probabilities: {True: 0.5, False: 0.5}
Conditional features probabilities: [('CrdtHistory', 'current', True, 0.25), ('CrdtHistory', 'current', False, 0.25), ('CrdtHistory', 'paid', True, 0.25), ('C
```

2Modify your NBClassifier that can work on continues descriptive and target variables.

```
import pandas as pd
import numpy as np
from scipy.stats import norm
from sklearn.preprocessing import LabelEncoder

class NaiveBayesClassifier:
    def __init__(self, data, target):
        # Constructor to initialize the classifier with input data and target variable
        self.data = data
        self.target = target
        self.target_prob_dict = {}  # Dictionary to store target class probabilities
        self.cond_prob_dict = {}  # Dictionary to store conditional probabilities
        self.continuous_features = []  # List to store names of continuous features
        self.k = 1  # Laplace Smoothing parameter

    def compute_target_probabilities(self):
        # Compute probabilities of each target class in the dataset
        total_rows = len(self.data)
        target_counts = self.data[self.target].value_counts()

        for level, count in target_counts.items():
            self.target_prob_dict[level] = count / total_rows

    def compute_conditional_probabilities(self):
        # Compute conditional probabilities for each feature given the target class
        for feature in self.data.columns:
            if feature != self.target:
                if self.data[feature].dtype == np.float64 or self.data[feature].dtype
== np.int64:
                    # Continuous variable
                    self.continuous_features.append(feature)
                else:
                    # Categorical variable
                    feature_levels = self.data[feature].unique()

                    for level in feature_levels:
                        for target_level in self.target_prob_dict.keys():
                            count_f_v_t = len(self.data[(self.data[feature] == level) &
(self.data[self.target] == target_level)])
                            count_f_t = len(self.data[self.data[self.target] ==
target_level])

                            prob = count_f_v_t / count_f_t

                            # Store the conditional probability in the dictionary
                            self.cond_prob_dict[(feature, level, target_level)] = prob

    def fit(self, smoothing=False, k=1):
        # Method to train the Naive Bayes classifier
```

```
        self.k = k
        self.compute_target_probabilities()
        self.compute_conditional_probabilities()

    def predict_instance(self, instance):
        # Predict the target class for a given instance
        prob_dict = {}

        for target_level in self.target_prob_dict.keys():
            prob_prod = 1

            for feature, level, t_level in self.cond_prob_dict.keys():
                if feature in instance:
                    if feature in self.continuous_features:
                        # Use Gaussian distribution for continuous variables
                        mean = self.data[self.data[self.target] ==
t_level][feature].mean()
                        std = self.data[self.data[self.target] ==
t_level][feature].std()
                        prob_density = norm.pdf(instance[feature], mean, std)
                        prob_prod *= prob_density
                    else:
                        # Categorical variable
                        if instance[feature] == level and t_level == target_level:
                            prob_prod *= self.cond_prob_dict[(feature, level,
target_level)]

            # Multiply the conditional probability with the prior probability
            prob_dict[target_level] = round(prob_prod *
self.target_prob_dict[target_level], 4)

        # Return the predicted target class with the highest probability
        return max(prob_dict, key=prob_dict.get)

    def get_probabilities(self):
        # Display the computed target and conditional probabilities
        print("Target probabilities:", self.target_prob_dict)
        print("Conditional features probabilities:", self.cond_prob_dict)

# Read your dataset
data = pd.read_csv('loan.csv')

# Initialize a LabelEncoder for encoding categorical variables
label_encoder = LabelEncoder()

# Convert 'Education', 'Self_Employed', and 'Loan_Status' columns to numerical labels
data['Education'] = label_encoder.fit_transform(data['Education'])
data['Self_Employed'] = label_encoder.fit_transform(data['Self_Employed'])
data['Loan_Status'] = label_encoder.fit_transform(data['Loan_Status'])
```

```
# Assuming 'Credit_History' is your target variable

# Create an instance of the NaiveBayesClassifier class with the preprocessed data and
target variable
nbc = NaiveBayesClassifier(data, 'Loan_Status')

# Fit the Naive Bayes classifier to the data with Laplace smoothing (smoothing=True)
and Laplace Smoothing parameter k=1
nbc.fit(smoothing=True, k=1)

# Prediction for a new instance
query_instance = {'Education': 0, 'Self_Employed': 0, 'Credit_History': 1,
'ApplicantIncome': 5000, 'LoanAmount': 128}
prediction = nbc.predict_instance(query_instance)
print("Prediction:", prediction)

# Display all target and conditional probabilities computed by the classifier
nbc.get_probabilities()
```

```
Prediction: 1
Target probabilities: {1: 0.6872964169381107, 0: 0.3127035830618892}
Conditional features probabilities: {('Loan_ID', 'LP001002', 1): 0.002369668246445498, ('Loan_ID', 'LP001002', 0): 0.0, ('Loan_ID', 'LP001003', 1): 0.0, ('Loan_ID', 'LP0
```

Student Name: sawera fazal Roll No: 21A-026-se Section: 21A

CS334 - Machine Learning

Lab 11 - Error Based Learning

Instructor: Ms. Maham Ashraf E-mail: mashraf@uit.edu

Semester: Fall, 2023

Objective

The purpose of this lab session is to introduce Error based learning models, such as Gradient Descent algorithm, Regression based classifier for categorical and continues features, and Support Vector Machine (SVM).

Instructions

You have to perform the following tasks yourselves. Raise your hand if you face any difficulty in understanding and solving these tasks. **Plagiarism** is an abhorrent practice and you should not engage in it.

How to Submit

- Submit lab work in a single .py file on Microsoft Teams. (No other format will be accepted)
- Lab work file name should be saved with your roll number (e.g. 19a-001-SE_LW01.py)
- Submit home work in a single .py file on Microsoft Teams. (No other format will be accepted)
- Home work file name should be saved with your roll number (e.g. 19a-001-SE_HW01.py)

- **Lab Tasks**

1. Write a complete Linear Regression class as defined in Section 2.

```
# Importing necessary libraries
import pandas as pd
import numpy as np

# Loading the training dataset from 'train.csv' into a Pandas DataFrame
df_train = pd.read_csv('train.csv')

# Loading the testing dataset from 'test.csv' into a Pandas DataFrame
```

```
df_test = pd.read_csv('test.csv')

# Extracting the 'x' column (features) from the training dataset
x_train = df_train['x']

# Extracting the 'y' column (labels) from the training dataset
y_train = df_train['y']

# Extracting the 'x' column (features) from the testing dataset
x_test = df_test['x']

# Extracting the 'y' column (labels) from the testing dataset
y_test = df_test['y']

# Converting the 'x_train' and 'y_train' data to NumPy arrays
x_train = np.array(x_train)
y_train = np.array(y_train)

# Converting the 'x_test' and 'y_test' data to NumPy arrays
x_test = np.array(x_test)
y_test = np.array(y_test)

# Reshaping the 'x_train' array to have a single feature column (-1 indicates that
the size of that dimension is inferred)
x_train = x_train.reshape(-1, 1)

# Reshaping the 'x_test' array to have a single feature column (-1 indicates that the
size of that dimension is inferred)
x_test = x_test.reshape(-1, 1)
```

```
# Calculating the number of null values in each column of the 'df_train' DataFrame
# The method 'isnull()' returns a DataFrame of the same shape as 'df_train', where each element is a boolean indicating whether the corre
# The method 'sum()' is then used to sum up the boolean values (True=1, False=0) along each column, providing the count of null values fo
null_values = df_train.isnull().sum()

# Displaying the count of null values for each column in 'df_train'
# The result is a Pandas Series where the index represents the column names, and the values represent the count of null values in each co
null_values
```

```
x    0
y    0
dtype: int64
```

```
# Calculating the number of null values in each column of the 'df_test' DataFrame
# The method 'isnull()' returns a DataFrame of the same shape as 'df_test', where each element is a boolean indicating whether the corres
# The method 'sum()' is then used to sum up the boolean values (True=1, False=0) along each column, providing the count of null values fo
null_values = df_test.isnull().sum()

# Displaying the count of null values for each column in 'df_test'
# The result is a Pandas Series where the index represents the column names, and the values represent the count of null values in each co
null_values
```

```
x    0
y    0
dtype: int64
```

```
# Importing necessary libraries
import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression  # Importing the Linear Regression model from scikit-learn
from sklearn.metrics import r2_score  # Importing the R2 score metric from scikit-learn
import matplotlib.pyplot as plt  # Importing matplotlib for plotting

# Creating a Linear Regression model instance
clf = LinearRegression()

# Training the Linear Regression model with the training data
# 'x_train' contains the input features, and 'y_train' contains the corresponding labels
clf.fit(x_train, y_train)

# Using the trained model to make predictions on the test data
y_pred = clf.predict(x_test)

# Calculating and printing the R-squared (R2) score, a measure of the model's performance
# R2 score indicates the proportion of the variance in the dependent variable that is predictable from the independent variable(s).
print('R2 Score (scikit-learn):', r2_score(y_test, y_pred))
```

[12]

```
R2 Score (scikit-learn): 0.9610404854275703
```

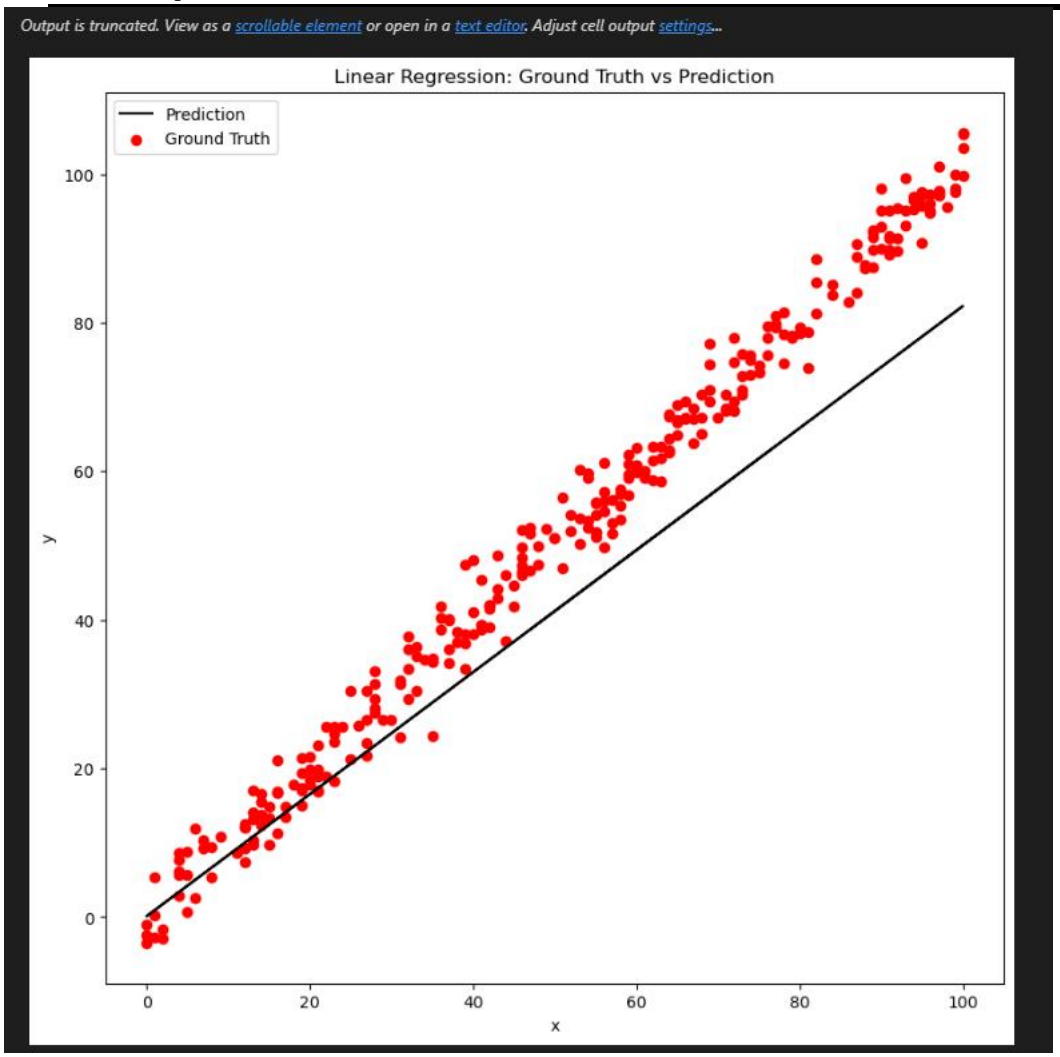
```
# Getting the number of training examples
n = len(x_train)

# Setting the learning rate for gradient descent
alpha = 0.00000001

# Initializing coefficients for the linear regression equation
a_0 = 0.0
a_1 = 0.0

# Initializing the epoch counter
epochs = 0
```

```
# Performing gradient descent for a specified number of epochs (500 in this case)
while epochs < 500:
    # Calculating the predicted values using the current coefficients
    y = a_0 + a_1 * x_train

    # Calculating the error between predicted and actual values
    error = y - y_train

    # Calculating the mean squared error
    mean_sq_err = np.sum(error ** 2)
    mean_sq_err = mean_sq_err / n

    # Updating coefficients using gradient descent
    a_0 = a_0 - alpha * 2 * np.sum(error) / n
    a_1 = a_1 - alpha * 2 * np.sum(error * x_train) / n

    # Incrementing the epoch counter
    epochs += 1

    # Printing mean squared error every 10 epochs
    if epochs % 10 == 0:
        print(f'Mean Squared Error after {epochs} epochs:', mean_sq_err)

# Reshaping coefficients for the test set
a_0_reshaped = a_0.reshape(-1, 1)
a_1_reshaped = a_1.reshape(-1, 1)

# Predicting values on the test set
y_prediction = a_0_reshaped + a_1_reshaped * x_test

# Flattening y_test for consistent dimensions
y_test = y_test.flatten()

# Calculating R2 Score for manually implemented gradient descent
r2 = r2_score(y_test, y_prediction.flatten())
print('R2 Score (Manual Gradient Descent):', r2)
```

```
# Plotting
plt.figure(figsize=(10, 10))
plt.scatter(x_test, y_test, color='red', label='Ground Truth')
plt.plot(x_test, y_prediction, color='black', label='Prediction')
plt.legend()
plt.xlabel('x')
plt.ylabel('y')
plt.title('Linear Regression: Ground Truth vs Prediction')
plt.show()
```

```
Mean Squared Error after 10 epochs: 13883298.94969759
Mean Squared Error after 20 epochs: 13471433.170016164
Mean Squared Error after 30 epochs: 13313228.878976261
Mean Squared Error after 40 epochs: 13252445.995607812
Mean Squared Error after 50 epochs: 13229078.846782872
Mean Squared Error after 60 epochs: 13220081.612343177
Mean Squared Error after 70 epochs: 13216603.298722873
Mean Squared Error after 80 epochs: 13215244.587235669
Mean Squared Error after 90 epochs: 13214699.933205076
Mean Squared Error after 100 epochs: 13214467.928243179
Mean Squared Error after 110 epochs: 13214356.001098987
Mean Squared Error after 120 epochs: 13214290.192725077
Mean Squared Error after 130 epochs: 13214242.098358626
Mean Squared Error after 140 epochs: 13214200.80884676
Mean Squared Error after 150 epochs: 13214162.134412589
Mean Squared Error after 160 epochs: 13214124.465927389
Mean Squared Error after 170 epochs: 13214087.18538799
Mean Squared Error after 180 epochs: 13214050.055443538
Mean Squared Error after 190 epochs: 13214012.984936792
Mean Squared Error after 200 epochs: 13213975.938857725
Mean Squared Error after 210 epochs: 13213938.903760258
Mean Squared Error after 220 epochs: 13213901.874480154
Mean Squared Error after 230 epochs: 13213864.849033942
Mean Squared Error after 240 epochs: 13213827.826659707
Mean Squared Error after 250 epochs: 13213790.80706475
...
Mean Squared Error after 480 epochs: 13212940.075238282
```

PROBLEMS 150 OUTPUT DEBUG CONSOLE TERMINAL PORTS JUPYTER


Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
Linear Regression: Ground Truth vs Prediction
Prediction
Ground Truth
y
x
0
20
40
60
80
100

Student Name: Sawera Fazal Roll No: 21A-026-se2 Section: 21A

CS334 - Machine Learning

Lab 12 - Error Based Learning - Part - 2

Instructor: Ms. Maham Ashraf E-mail: mashraf@uit.edu

Semester: Fall, 2023

Objective

The purpose of this lab session is to introduce Error based learning models, such as Gradient Descent algorithm, Regression based classifier for categorical and continues features, and Support Vector Machine (SVM).

Instructions

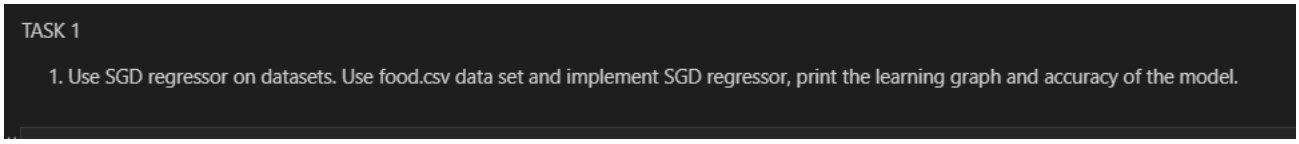
You have to perform the following tasks yourselves. Raise your hand if you face any difficulty in understanding and solving these tasks. **Plagiarism** is an abhorrent practice and you should not engage in it.

How to Submit

- Submit lab work in a single .py file on Microsoft Teams. (No other format will be accepted)
- Lab work file name should be saved with your roll number (e.g. 19a-001-SE_LW01.py)
- Submit home work in a single .py file on Microsoft Teams. (No other format will be accepted)
- Home work file name should be saved with your roll number (e.g. 19a-001-SE_HW01.py)

5 Lab Tasks

1. Use SGD regressor on datasets. Use `food.csv` data set and implement SGD regressor, print the learning graph and accuracy of the model.

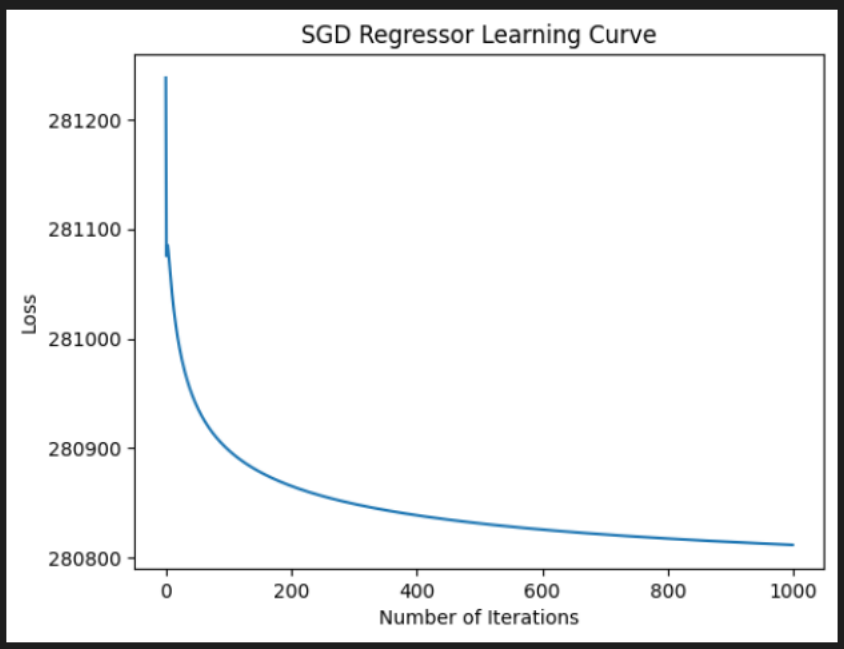



```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import SGDRegressor
```

```
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import r2_score

# Load the dataset (assuming it's in the same directory as the script)
data = pd.read_csv("food.csv")

# Assume that 'target_column' is the column you want to predict
target_column = "score"
X = data.drop(target_column, axis=1)
y = data[target_column]

# Label encode 'title' and 'id' columns
le = LabelEncoder()
X['title'] = le.fit_transform(X['title'])
X['id'] = le.fit_transform(X['id'])
X['url'] = le.fit_transform(X['url'])
X['body'] = le.fit_transform(X['body'])
X['timestamp'] = le.fit_transform(X['timestamp'])

# 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)

# Feature scaling (important for SGD)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Create an SGDRegressor instance
regressor = SGDRegressor(max_iter=1000, tol=1e-3, random_state=42)

# Manually capture the loss values during training
loss_values = []
for epoch in range(1000):  # Adjust the number of epochs as needed
    regressor.partial_fit(X_train_scaled, y_train)
    y_pred = regressor.predict(X_train_scaled)
    mse = mean_squared_error(y_train, y_pred)
    loss_values.append(mse)

# Predict on the test set
y_pred = regressor.predict(X_test_scaled)

# Calculate and print the Mean Squared Error (MSE) as a measure of accuracy
mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error:", mse)

# Calculate R-squared score
score = r2_score(y_test, y_pred)
```

```
print("R-squared Score:", score)

# Plot the learning graph
plt.plot(loss_values)
plt.title("SGD Regressor Learning Curve")
plt.xlabel("Number of Iterations")
plt.ylabel("Loss")
plt.show()
```

```
Mean Squared Error: 56517.4880073163
R-squared Score: 0.33137559527091787
```



2. To learn how to handle descriptive categorical variable in Linear Regression, develop the model as described below,

   (a) Load `flight.csv` data set.

   (b) Apply pre-processing on data set.

   (c) Encode categorical features into numerical values.

   (d) Apply linear regression model from `sklearn` library.

   (e) Predict the delay of flight departure.

TASK 2 To learn how to handle descriptive categorical variable in Linear Regression, develop the model as described below, (a) values. (d) Apply linear regression model from sklearn library. (e) Predict the delay of flight departure.

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline

# (a) Load flight.csv data set
df = pd.read_csv("flight.csv")
df.head()
```

[2]

	UniqueCarrier	Origin	Distance	CRSDepTime	DepDelay
0	AA	JFK	2353	1003	-2
1	DL	JFK	4567	356	45
2	WN	JFK	344	383	-3
3	DL	DFW	317	2298	-5
4	B6	ATL	1905	1839	45

```
# (b) Apply pre-processing on data set
null_values = df.isnull().sum()
print(null_values)
```

[3]

```
UniqueCarrier    0
Origin           0
Distance         0
CRSDepTime       0
```

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.metrics import mean_squared_error

# (a) Load flight.csv data set
data = pd.read_csv("flight.csv")

# (b) Apply pre-processing on the data set
# Assuming 'DepDelay' is the target column to predict
target_column = "DepDelay"
X = data.drop(target_column, axis=1)
y = data[target_column]

# Assume 'UniqueCarrier' and 'Origin' are categorical features
categorical_features = ['UniqueCarrier', 'Origin']
X_categorical = X[categorical_features]
X_numerical = X.drop(categorical_features, axis=1)

# (c) Encode categorical features into numerical values
```

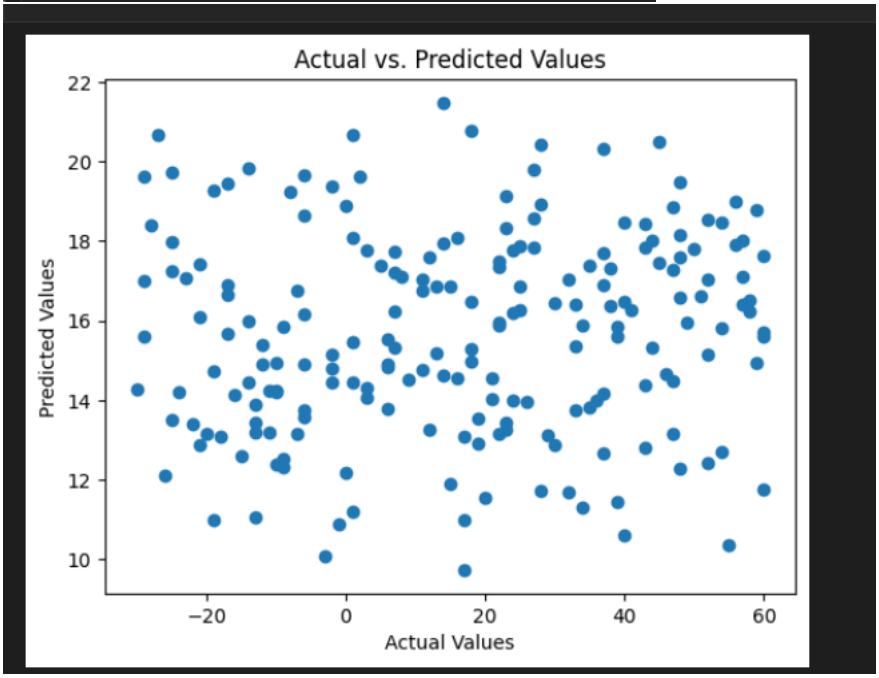
```
# Use OneHotEncoder for encoding categorical variables
preprocessor = ColumnTransformer(
    transformers=[
        ('cat', OneHotEncoder(), categorical_features)
    ],
    remainder='passthrough'
)

# (d) Apply linear regression model from sklearn library
# Create a pipeline with preprocessing and linear regression
model = Pipeline([
    ('preprocessor', preprocessor),
    ('regressor', LinearRegression())
])

# (e) 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 model on the training data
model.fit(X_train, y_train)

# Predict the delay of flight departure on the test set
y_pred = model.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error:", mse)
```

```
Mean Squared Error: 653.07859743293
```

```
# (b) Apply pre-processing on data set
null_values = df.isnull().sum()
print(null_values)
```

```
UniqueCarrier    0
Origin           0
Distance         0
CRSDepTime       0
DepDelay         0
dtype: int64
```

```
import matplotlib.pyplot as plt

plt.scatter(y_test, y_pred)
plt.xlabel("Actual Values")
plt.ylabel("Predicted Values")
plt.title("Actual vs. Predicted Values")
plt.show()
```



3. To learn how to handle continues target variable in Linear Regression Model, Read the following example and develop the proper model.

   The `ALCustomers.csv` data contains data of airline customer. The main purpose of this dataset is to predict whether a future customer would be satisfied with their service given the details of the other parameters values.

   Also the airlines need to know on which aspect of the services offered by them have to be emphasized more to generate more satisfied customers.

   (a) Develop a prediction model that predict that either customer would be satisfy with the current services or not.

   (b) Predict the factors (e.g. descriptive variables) that need improvements to increase customer satisfaction rate.

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# Load the dataset
data = pd.read_csv("Alcustomer.csv")

# Assume 'satisfaction' is the target variable
target_column = "satisfaction"
X = data.drop(target_column, axis=1)
y = data[target_column]

# 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)

# Define numerical and categorical features
numeric_features = X.select_dtypes(include=['int64', 'float64']).columns
categorical_features = X.select_dtypes(include=['object']).columns

# Create a preprocessor using ColumnTransformer
preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numeric_features),
        ('cat', OneHotEncoder(), categorical_features)
    ],
    remainder='passthrough'
)

# Create a Logistic Regression model pipeline with imputation
model = Pipeline([
    ('preprocessor', preprocessor),
    ('imputer', SimpleImputer(strategy='mean')),  # Choose imputation strategy
    ('classifier', LogisticRegression(random_state=42))
])

# Fit the model on the training data
model.fit(X_train, y_train)

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

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
```

```
conf_matrix = confusion_matrix(y_test, y_pred)
classification_rep = classification_report(y_test, y_pred)

print("Accuracy:", accuracy)
print("\nConfusion Matrix:")
print(conf_matrix)
print("\nClassification Report:")
print(classification_rep)
# Get feature names after one-hot encoding (if applicable)
feature_names = X.columns.tolist()

```

```
Confusion Matrix:
[[10626  1087]
 [ 1432  7636]]

Classification Report:
                       precision    recall  f1-score   support

neutral or dissatisfied       0.88      0.91      0.89     11713
              satisfied       0.88      0.84      0.86      9068

               accuracy                           0.88     20781
              macro avg       0.88      0.87      0.88     20781
           weighted avg       0.88      0.88      0.88     20781
```

```
# Get coefficients and their corresponding features
coefficients = model.named_steps['classifier'].coef_[0]

# Combine feature names and coefficients
feature_coefficients = list(zip(feature_names, coefficients))

# Sort features by their coefficients
feature_coefficients.sort(key=lambda x: abs(x[1]), reverse=True)

# Display feature coefficients
print("Feature Coefficients are:")
for feature, coef in feature_coefficients:
    print(f"{feature}: {coef}")
```

[15]

```
Feature Coefficients are:
Arrival Delay in Minutes: -1.0004768976281235
Departure Delay in Minutes: 0.9959136439275648
Departure/Arrival time convenient: 0.838390860400836
Age: 0.5106502210803967
Inflight entertainment: 0.4133585750484487
Food and drink: 0.3992164279258385
Checkin service: -0.3919740379466015
Online boarding: 0.33633740475028717
Leg room service: 0.29337922639568154
Class: -0.20239096805719684
Baggage handling: 0.1990962498487794
Type of Travel: -0.1834439934274642
id: -0.17206442719037915
```

Student Name: Sawera Fazal Roll No: 21A-026-se Section: 21A

CS334 - Machine Learning

Lab 13

Instructor: Ms. Maham Ashraf E-mail: mashraf@uit.edu Semester: Fall, 2023

Objective

Open Ended Lab to perform all the concepts of previous lab and work on a dataset to implement Analysis, process, feature Selection , Model Implementation and model Evaluation with Visualization

Instructions

You have to perform the following tasks yourselves. Raise your hand if you face any difficulty in understanding and solving these tasks. **Plagiarism** is an abhorrent practice and you should not engage in it.

How to Submit

- Submit lab work in a single .py file on Microsoft Teams. (No other format will be accepted)
- Lab work file name should be saved with your roll number (e.g. 19a-001SE_LW01.py)
- Submit home work in a single .py file on Microsoft Teams. (No other format will be accepted)
- Home work file name should be saved with your roll number (e.g. 19a-001SE_HW01.py)

Lab task:

1)Explore Hyper parameter tuning on any dataset from Kaggle.

2)Explore other hyper parameter tuning techniques and use them in your code on your own dataset.

3)Evaluate the performance of different hyperparameter configurations on the validation set.

Diabetes dataset

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
dataset_path = 'diabetes.csv'
df = pd.read_csv(dataset_path)
df.head()
```

[3] ✓ 0.0s

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

+ Code + Mark

```
df.dtypes
```

[4] ✓ 0.0s

```
Pregnancies                 int64
Glucose                     int64
BloodPressure               int64
SkinThickness               int64
Insulin                     int64
BMI                       float64
DiabetesPedigreeFunction  float64
Age                         int64
Outcome                     int64
```

```
# Splitting data into features (X) and target variable (y)
X = df.drop('Outcome', axis=1)  # Adjust 'target_variable_column' to your dataset
y = df['Outcome']

# Splitting 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)

# Implementing a simple machine learning model (e.g., Decision Tree)
# Instantiate the Decision Tree classifier
model = DecisionTreeClassifier(random_state=42)

# Fit the model on the training data
model.fit(X_train, y_train)

# Make predictions on the testing data
y_pred = model.predict(X_test)

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

✓ 0.0s

```
Accuracy of the Decision Tree model: 0.7467532467532467
```

GridSearchCV & RandomizedSearchCV

```
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
```

```
param_grid = {'criterion': ['gini', 'entropy'],
              'splitter': ['best', 'random'],
              'max_depth': [None, 10, 20, 30, 40, 50],
              'min_samples_split': [2, 5, 10],
              'min_samples_leaf': [1, 2, 4]}

param_dist = {'criterion': ['gini', 'entropy'],
              'splitter': ['best', 'random'],
              'max_depth': [None, 10, 20, 30, 40, 50],
              'min_samples_split': [2, 5, 10],
              'min_samples_leaf': [1, 2, 4]}



# Grid Search
grid_search = GridSearchCV(model, param_grid, cv=5)
grid_search.fit(X_train, y_train)

# Random Search
random_search = RandomizedSearchCV(model, param_distributions=param_dist, n_iter=10,
cv=5)
random_search.fit(X_train, y_train)


# Evaluate models on the testing set
y_pred_grid = grid_search.predict(X_test)
y_pred_random = random_search.predict(X_test)

# Calculate accuracy scores
accuracy_grid = accuracy_score(y_test, y_pred_grid)
accuracy_random = accuracy_score(y_test, y_pred_random)

# Print accuracy scores
print("Accuracy with Grid Search:", accuracy_grid)
print("Accuracy with Random Search:", accuracy_random)
```

```
✓ 7.2s
Accuracy with Grid Search: 0.7597402597402597
Accuracy with Random Search: 0.7467532467532467
```

Student Name: Sawera fazal Roll No: 21A-026-se Section: 21A

CS334 - Machine Learning

Lab 14

Instructor: Ms. Maham Ashraf E-mail: mashraf@uit.edu Semester: Fall, 2023

Objective

Open Ended Lab to perform all the concepts of previous lab and work on a dataset to implement Analysis, process, feature Selection , Model Implementation and model Evaluation with Visualization

Instructions

You have to perform the following tasks yourselves. Raise your hand if you face any difficulty in understanding and solving these tasks. **Plagiarism** is an abhorrent practice and you should not engage in it.

How to Submit

- Submit lab work in a single .py file on Microsoft Teams. (No other format will be accepted)
- Lab work file name should be saved with your roll number (e.g. 19a-001SE_LW01.py)
- Submit home work in a single .py file on Microsoft Teams. (No other format will be accepted)
- Home work file name should be saved with your roll number (e.g. 19a-001SE_HW01.py)

Objective:

This open-ended lab project aims to consolidate and apply the diverse range of AI concepts learned in previous labs. The project encourages students to work on a real-world dataset, performing tasks such as analysis, data processing, feature selection, model implementation, and evaluation. Visualization techniques should be employed to enhance the understanding and interpretation of the results.

Project Tasks:

1. **Data Exploration and Understanding:**
   - Begin by thoroughly exploring the contents of the provided dataset.
   - Examine the structure, data types, and key attributes within the dataset.
   - Document any initial observations or patterns identified during exploration.
2. **Analysis and Visualization:**
   - Conduct a comprehensive analysis of the dataset, exploring statistical measures, and correlations.
   - Utilize visualization libraries to create insightful charts, graphs, or plots.
   - Provide visualizations that enhance understanding and reveal patterns within the data.
3. **Data Preprocessing and Cleaning:**
   - Implement necessary preprocessing steps to clean and organize the dataset.
   - Handle missing values, outliers, or any other data imperfections.
   - Discuss the impact of preprocessing on the overall quality of the data.
4. **Feature Selection and Engineering:**
   - Explore feature selection techniques to identify and retain the most relevant features.
   - Consider the creation of new features through feature engineering.
   - Discuss the rationale behind feature selection and engineering decisions.
5. **Model Implementation:**
   - Choose a suitable machine learning or deep learning model for the task at hand.
   - Implement the selected model using appropriate libraries or frameworks.
   - Justify the choice of the model architecture based on the characteristics of the dataset.
6. **Training and Evaluation:**
   - Train the implemented model using the preprocessed dataset.
   - Evaluate the model's performance using relevant metrics.
   - Discuss the significance of the evaluation results in the context of the project.
7. **Results Visualization and Analysis:**
   - Visualize the predictions of the model on sample data.
   - Analyze the results, emphasizing the strengths and limitations of the implemented model.
   - Consider the interpretability of the model and its implications.

Submission Guidelines:

- Submit in a well-documented Jupyter notebook containing the implemented code, explanations, and visualizations.
- Include comments and markdown cells for clarity and understanding.
- Ensure the file provides a clear narrative of the project, from data exploration to model evaluation.

EDA AND VISUALISATION

In [52]:

```
import pandas as pd

data = pd.read_csv("loan_approval_dataset.csv")
dataFrame = pd.DataFrame(data)

dataFrame.head()
```

Out[52]:

	loan_id	no_of_dependents	education	self_employed	income_annum	loan_amount	loan_term	cibil_score	residential_assets_value
0	1	2	Graduate	No	9600000	29900000	12	778	2400000
1	2	0	Not Graduate	Yes	4100000	12200000	8	417	2700000
2	3	3	Graduate	No	9100000	29700000	20	506	7100000
3	4	3	Graduate	No	8200000	30700000	8	467	18200000
4	5	5	Not Graduate	Yes	9800000	24200000	20	382	12400000

In [5]:

```
dataFrame.shape
```

Out[5]:

(4269, 13)

In [6]:

```
dataFrame.describe()
```

Out[6]:

	loan_id	no_of_dependents	income_annum	loan_amount	loan_term	cibil_score	residential_assets_value	commercial_a
count	4269.000000	4269.000000	4.269000e+03	4.269000e+03	4269.000000	4269.000000	4.269000e+03	4.
mean	2135.000000	2.498712	5.059124e+06	1.513345e+07	10.900445	599.936051	7.472617e+06	4.
std	1232.498479	1.695910	2.806840e+06	9.043363e+06	5.709187	172.430401	6.503637e+06	4.
min	1.000000	0.000000	2.000000e+05	3.000000e+05	2.000000	300.000000	-1.000000e+05	0.
25%	1068.000000	1.000000	2.700000e+06	7.700000e+06	6.000000	453.000000	2.200000e+06	1.
50%	2135.000000	3.000000	5.100000e+06	1.450000e+07	10.000000	600.000000	5.600000e+06	3.
75%	3202.000000	4.000000	7.500000e+06	2.150000e+07	16.000000	748.000000	1.130000e+07	7.
max	4269.000000	5.000000	9.900000e+06	3.950000e+07	20.000000	900.000000	2.910000e+07	1

In

```
dataFrame.dtypes
```

Out[7]:

```
loan_id                    int64
 no_of_dependents          int64
 education                object
 self_employed            object
 income_annum              int64
 loan_amount               int64
 loan_term                 int64
 cibil_score               int64
 residential_assets_value  int64
 commercial_assets_value   int64
 luxury_assets_value       int64
 bank_asset_value          int64
 loan_status              object
dtype: object
```

In [8]:

```
null = dataFrame.isnull().sum() print(null.count)
```

```
<bound method Series.count of  loan_id                    0
 no_of_dependents          0
 education                 0
 self_employed             0
 income_annum              0
 loan_amount               0
 loan_term                 0
 cibil_score               0
 residential_assets_value  0
 commercial_assets_value   0
 luxury_assets_value       0
 bank_asset_value          0
 loan_status               0
dtype: int64>
```

In [9]:

```
dataFrame.columns
```

Out[9]:

```
Index(['loan_id', ' no_of_dependents', ' education', ' self_employed',
       ' income_annum', ' loan_amount', ' loan_term', ' cibil_score',
       ' residential_assets_value', ' commercial_assets_value',
       ' luxury_assets_value', ' bank_asset_value', ' loan_status'],
      dtype='object')
```

In [10]:

```
#checking relation btw columns
dataFrame.corr()
```

Out[10]:

	loan_id	no_of_dependents	income_annum	loan_amount	loan_term	cibil_score	residential_assets_value
loan_id	1.000000	0.005326	0.012592	0.008170	0.009809	0.016323	0.020936
no_of_dependents	0.005326	1.000000	0.007266	-0.003366	-0.020111	-0.009998	0.007376
income_annum	0.012592	0.007266	1.000000	0.927470	0.011488	-0.023034	0.636841
loan_amount	0.008170	-0.003366	0.927470	1.000000	0.008437	-0.017035	0.594596
loan_term	0.009809	-0.020111	0.011488	0.008437	1.000000	0.007810	0.008016
cibil_score	0.016323	-0.009998	-0.023034	-0.017035	0.007810	1.000000	-0.019947
residential_assets_value	0.020936	0.007376	0.636841	0.594596	0.008016	-0.019947	1.000000
commercial_assets_value	0.018595	-0.001531	0.640328	0.603188	-0.005478	-0.003769	0.414786
luxury_assets_value	-0.000862	0.002817	0.929145	0.860914	0.012490	-0.028618	0.590932
bank_asset_value	0.010765	0.011163	0.851093	0.788122	0.017177	-0.015478	0.527418

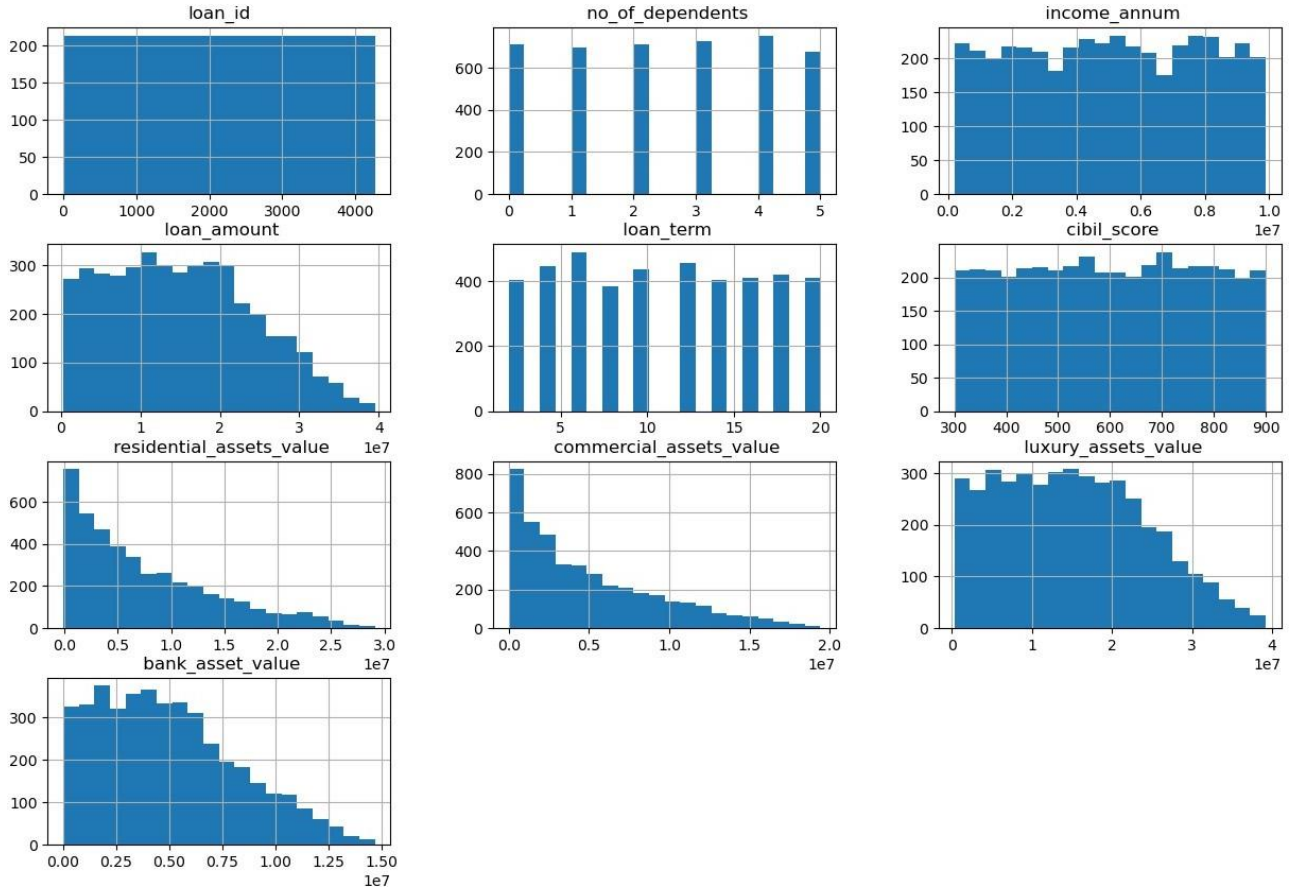
In

```
dataFrame.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4269 entries, 0 to 4268
Data columns (total 13 columns):
 #   Column                    Non-Null Count  Dtype 
---  ------                    --------------  ----- 
 0   loan_id                   4269 non-null   int64 
 1    no_of_dependents         4269 non-null   int64 
 2    education                4269 non-null   object
 3    self_employed            4269 non-null   object
 4    income_annum             4269 non-null   int64 
 5    loan_amount              4269 non-null   int64 
 6    loan_term                4269 non-null   int64 
 7    cibil_score              4269 non-null   int64 
 8    residential_assets_value 4269 non-null   int64 
 9    commercial_assets_value  4269 non-null   int64 
 10   luxury_assets_value      4269 non-null   int64 
 11   bank_asset_value         4269 non-null   int64 
 12   loan_status              4269 non-null   object
dtypes: int64(10), object(3)
memory usage: 433.7+ KB
```

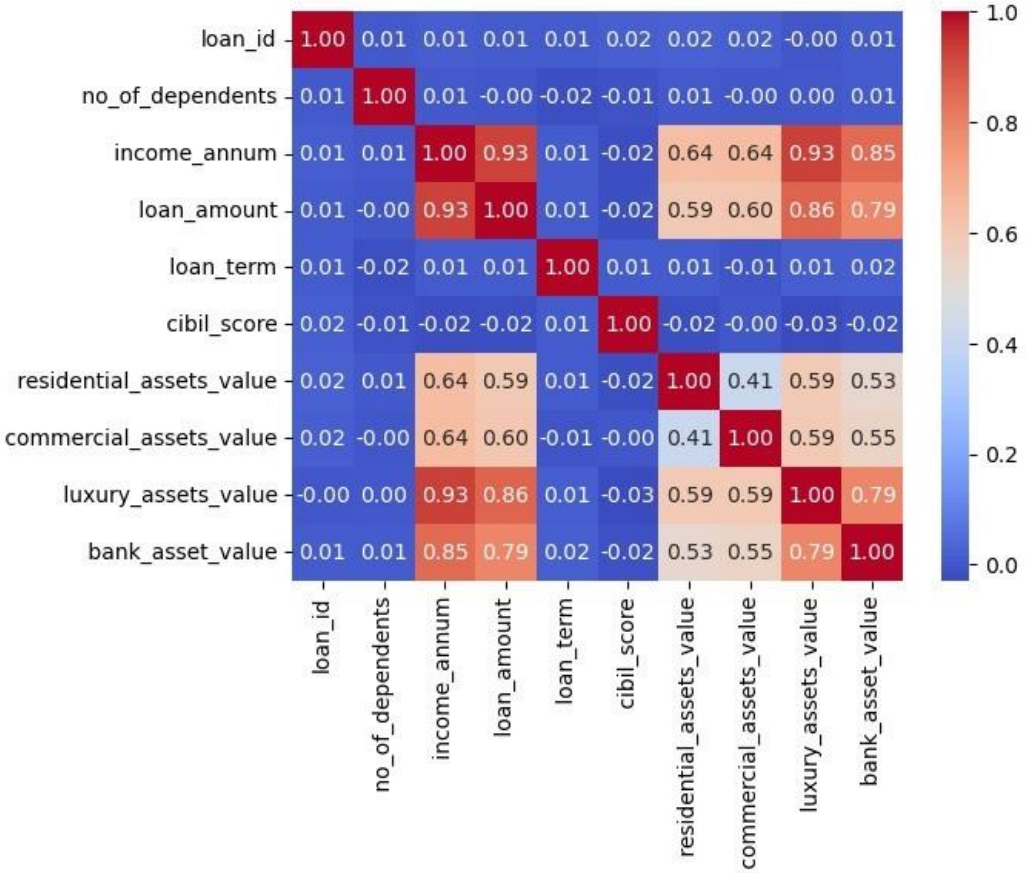
In [12]:

```
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
numerical_columns = dataFrame.select_dtypes(include=np.number).columns
dataFrame[numerical_columns].hist(bins=20, figsize=(15, 10)) plt.show()
```



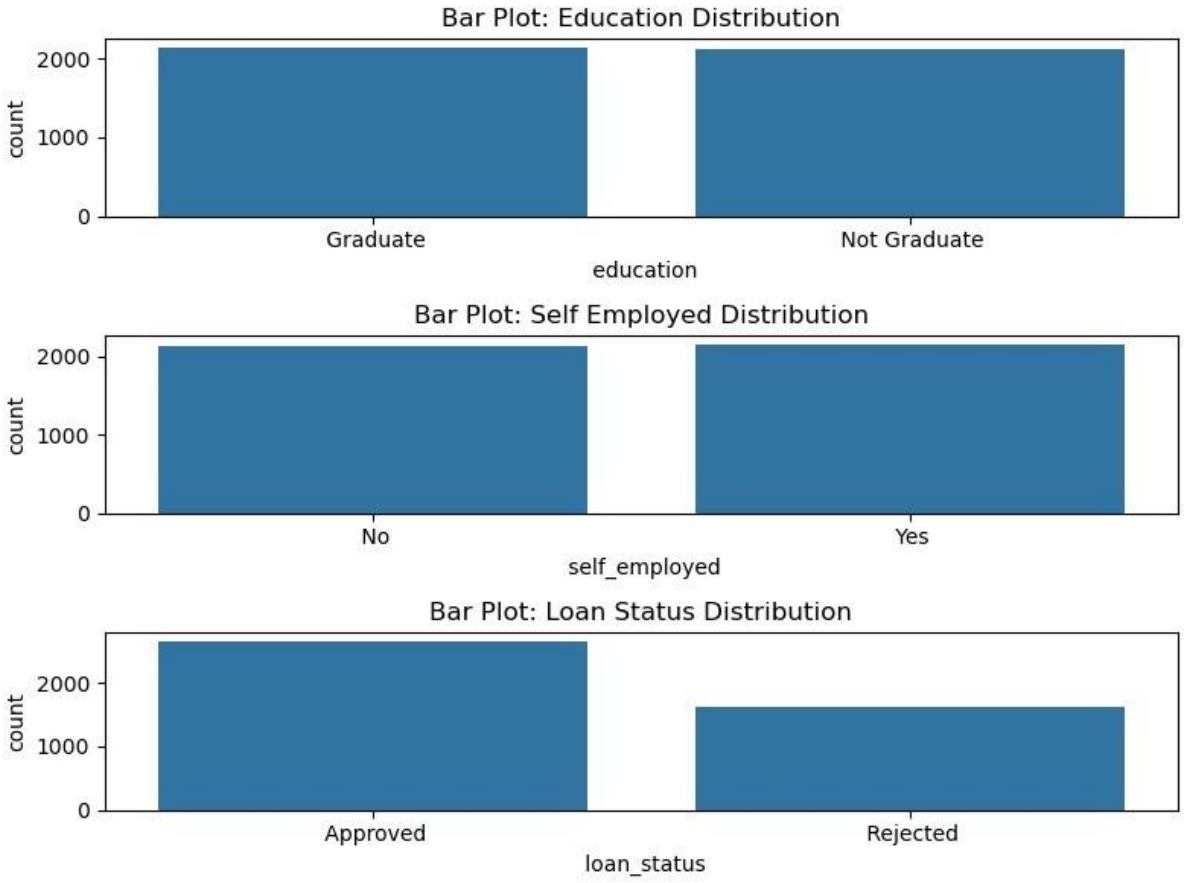
In

```
correlation_matrix = dataFrame.corr()
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f") plt.show()
```



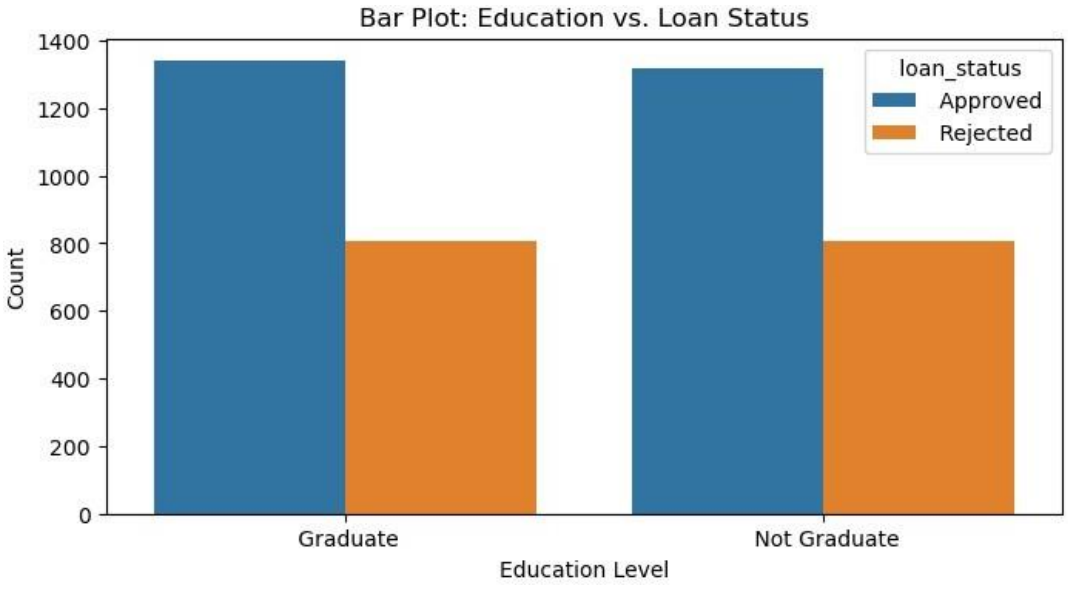
In

```
# Bar plot between 'education', 'self_employed', and 'loan_status'
plt.figure(figsize=(8, 6))

# Bar plot for 'education'
plt.subplot(3, 1, 1)
sns.countplot(x='education', data=dataFrame)
plt.title('Bar Plot: Education Distribution')

# Bar plot for 'self_employed'
plt.subplot(3, 1, 2)
sns.countplot(x='self_employed', data=dataFrame)
plt.title('Bar Plot: Self Employed Distribution')

# Bar plot for 'loan_status'
plt.subplot(3, 1, 3)
sns.countplot(x='loan_status', data=dataFrame)
plt.title('Bar Plot: Loan Status Distribution')

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



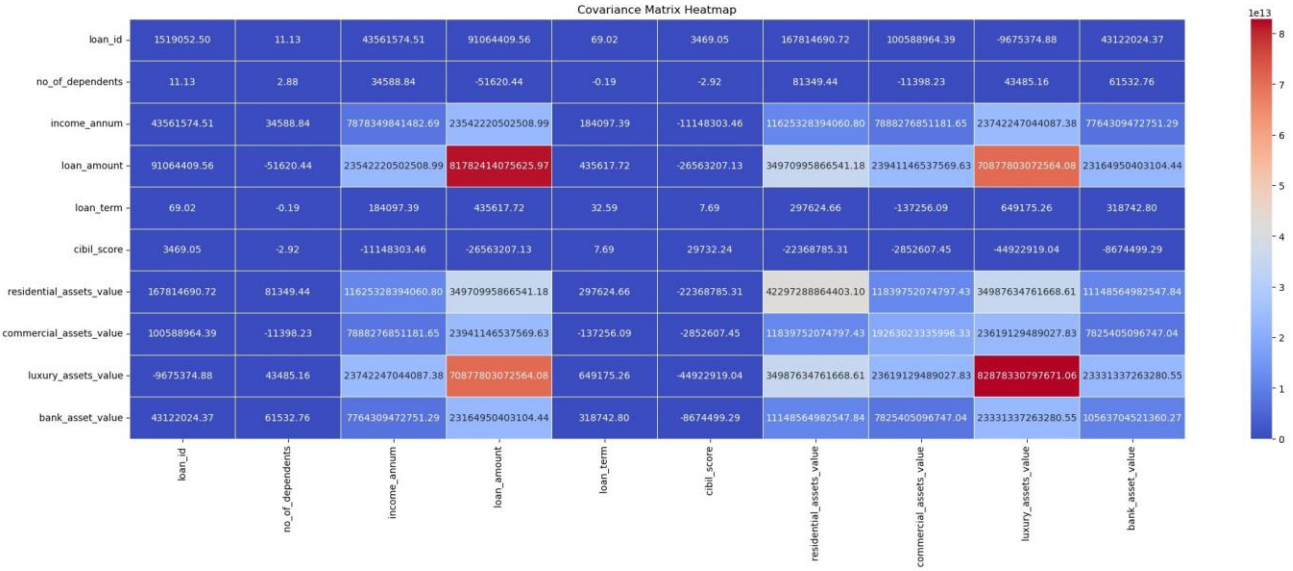
In

```
# Bar plot between 'education' and 'loan_status'
plt.figure(figsize=(8, 4))
sns.countplot(x=' education', hue=' loan_status', data=dataFrame)
plt.title('Bar Plot: Education vs. Loan Status') plt.xlabel('Education
Level')
plt.ylabel('Count')
plt.show()
```



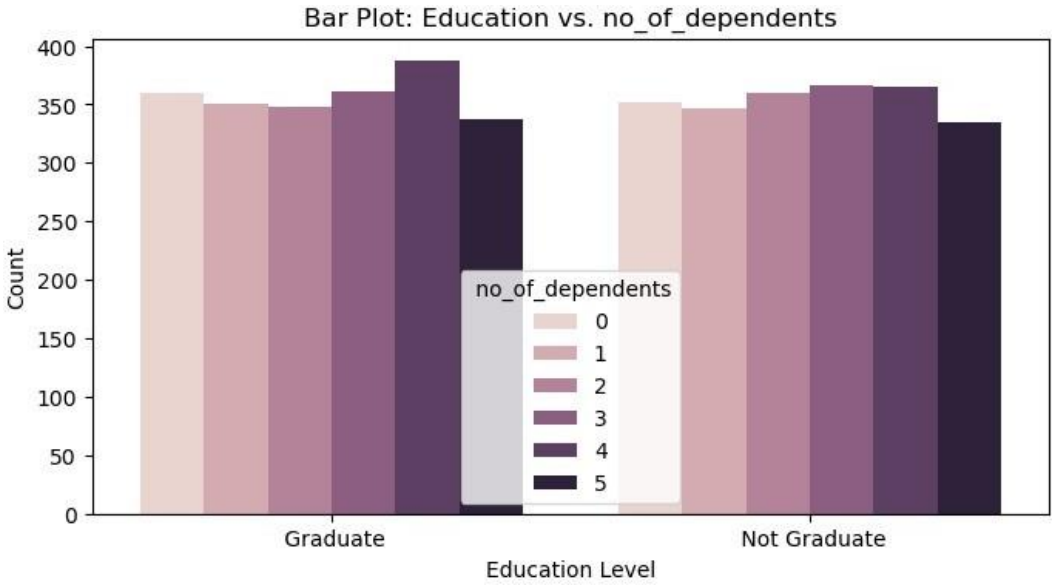
In [17]:

```
# Calculate the covariance matrix
covariance_matrix = dataFrame[numerical_columns].cov()

# Plot the covariance matrix as a heatmap
plt.figure(figsize=(25, 8))
sns.heatmap(covariance_matrix, annot=True, cmap='coolwarm', fmt='.2f', linewidths=0.5) plt.title('Covariance Matrix
Heatmap')
plt.show()
```



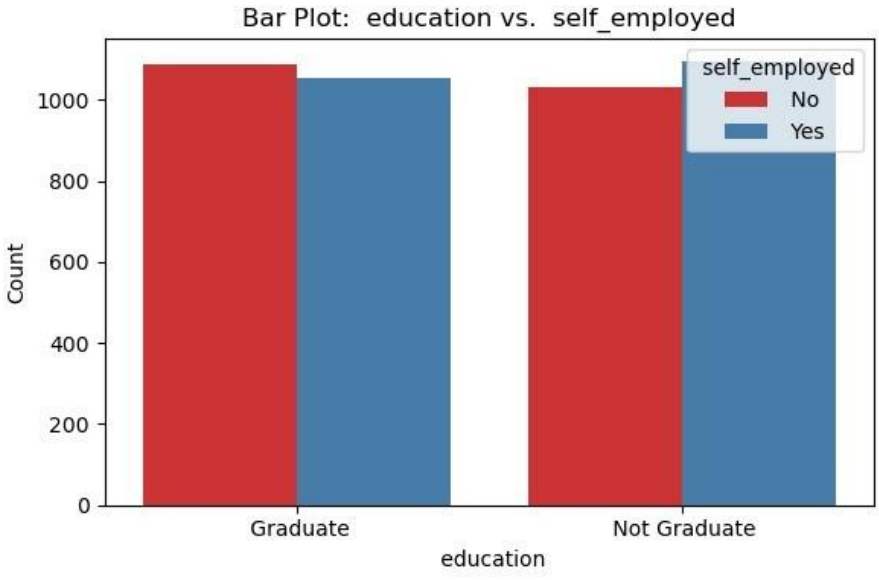
In

```
# Bar plot between 'education' and 'loan_status'
plt.figure(figsize=(8, 4))
sns.countplot(x=' education', hue=' no_of_dependents', data=dataFrame)
plt.title('Bar Plot: Education vs. no_of_dependents') plt.xlabel('Education
Level')
plt.ylabel('Count')
plt.show()
```



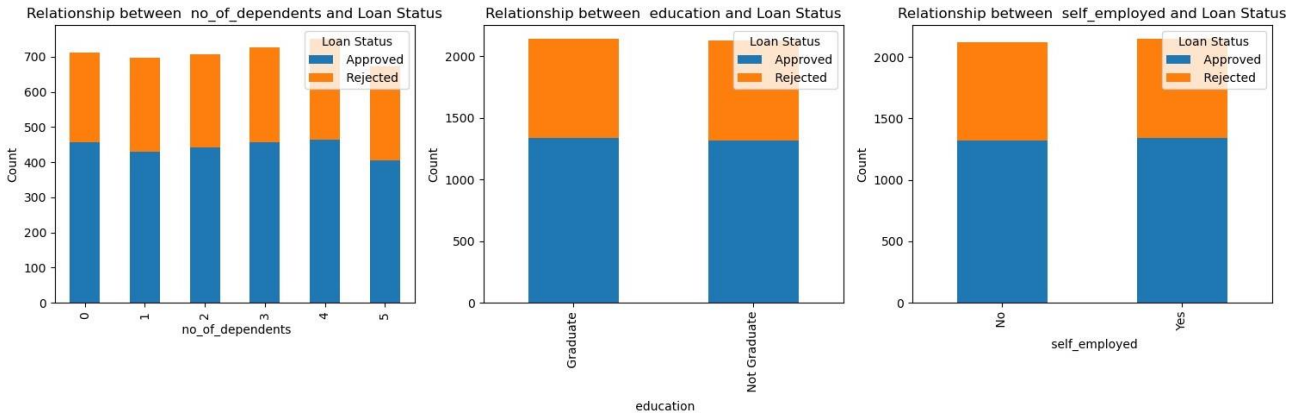
In [58]:

```
# Select three categorical columns for the bar plot
categorical_columns = [' education', ' self_employed', ' no_of_dependents']

# Create a bar plot
plt.figure(figsize=(6, 4))
sns.countplot(x=categorical_columns[0], hue=categorical_columns[1], data=dataFrame, palette='Set1') plt.title(f'Bar Plot:
{categorical_columns[0]} vs. {categorical_columns[1]}') plt.xlabel(categorical_columns[0])
plt.ylabel('Count')
plt.legend(title=categorical_columns[1])

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



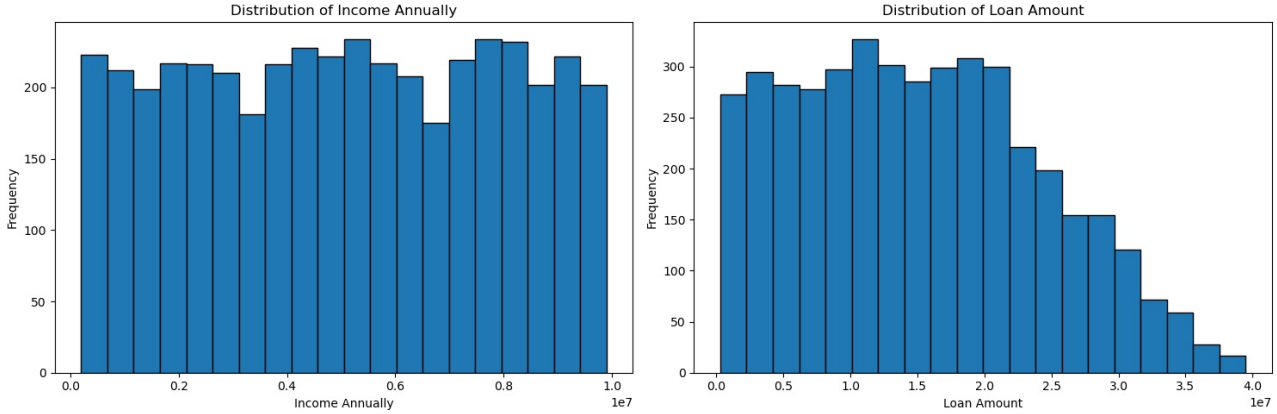
In

```
# List of variables to analyze
variables_to_analyze = [' no_of_dependents', ' education', ' self_employed']

# Set up subplots
fig, axes = plt.subplots(nrows=1, ncols=len(variables_to_analyze), figsize=(15, 5))

# Loop through variables and create bar plots
for i, variable in enumerate(variables_to_analyze): ax =
    axes[i]
    df_grouped = dataFrame.groupby([variable, ' loan_status']).size().unstack()
    df_grouped.plot(kind='bar', stacked=True, ax=ax) ax.set_title(f'Relationship
    between {variable} and Loan Status') ax.set_xlabel(variable)
    ax.set_ylabel('Count') ax.legend(title='Loan
    Status')

# Adjust layout
plt.tight_layout()

# Show the plots
plt.show()
```



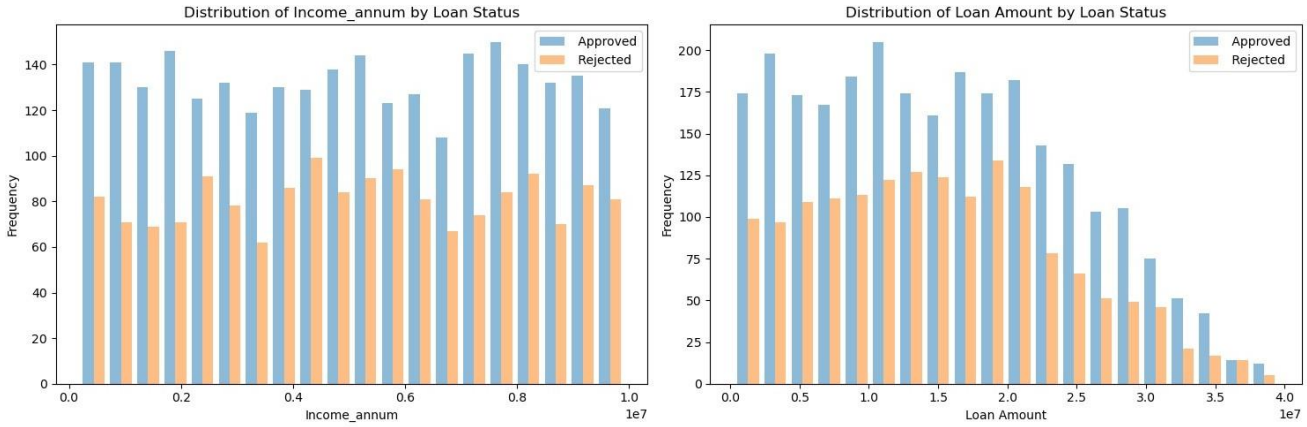
In [21]:

```
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(15, 5))

# Create histograms
dataFrame[' income_annum'].plot(kind='hist', bins=20, edgecolor='black', ax=axes[0]) axes[0].set_title('Distribution
of Income Annually')
axes[0].set_xlabel('Income Annually')
axes[0].set_ylabel('Frequency')

dataFrame[' loan_amount'].plot(kind='hist', bins=20, edgecolor='black', ax=axes[1]) axes[1].set_title('Distribution of
Loan Amount')
axes[1].set_xlabel('Loan Amount')
axes[1].set_ylabel('Frequency')

# Adjust layout
plt.tight_layout()

# Show the plots
plt.show()
```



In

```
import matplotlib.pyplot as plt

# Set up subplots
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(15, 5))

# Histogram for income_annum
axes[0].hist([dataFrame[dataFrame[' loan_status'] == ' Approved'][' income_annum'],
              dataFrame[dataFrame[' loan_status'] == ' Rejected'][' income_annum']],
              bins=20, alpha=0.5, label=[' Approved', ' Rejected'])
axes[0].set_title('Distribution of Income_annum by Loan Status')
axes[0].set_xlabel("Income_annum")
axes[0].set_ylabel('Frequency')
axes[0].legend()

# Histogram for loan_amount
axes[1].hist([dataFrame[dataFrame[' loan_status'] == ' Approved'][' loan_amount'],
              dataFrame[dataFrame[' loan_status'] == ' Rejected'][' loan_amount']],
              bins=20, alpha=0.5, label=[' Approved', ' Rejected'])
axes[1].set_title('Distribution of Loan Amount by Loan Status')
axes[1].set_xlabel('Loan Amount')
axes[1].set_ylabel('Frequency')
axes[1].legend()

# Adjust layout
plt.tight_layout()

# Show the plots
plt.show()
```



Handling Outliers

In

```
import pandas as pd

# Assuming df is your DataFrame
# Replace with your actual dataset loading mechanism
# df = pd.read_csv('your_dataset.csv')

# Define a function to print and handle outliers using IQR for all numerical columns
def handle_outliers_iqr(dataframe):
    outliers = pd.DataFrame(columns=["Column", 'Outliers'])

    for column in dataframe.select_dtypes(include="number").columns:
        Q1 = dataframe[column].quantile(0.25)
        Q3 = dataframe[column].quantile(0.75)
        IQR = Q3 - Q1
        lower_bound = Q1 - 1.5 * IQR
        upper_bound = Q3 + 1.5 * IQR

        column_outliers = dataframe[(dataframe[column] < lower_bound) | (dataframe[column] > upper_bound)]
        outliers = pd.concat([outliers, pd.DataFrame({"Column": [column], 'Outliers': [column_outliers]})])

        # Clip the values to remove outliers
        dataframe[column] = dataframe[column].clip(lower=lower_bound, upper=upper_bound)

    return outliers

# Get outliers and print them
outliers_info = handle_outliers_iqr(dataFrame)
outliers_info
```

Out[62]:

	Column	Outliers
0	loan_id	Empty DataFrame Columns: [loan_id, no_of_depe...
0	no_of_dependents	Empty DataFrame Columns: [loan_id, no_of_depe...
0	income_annum	Empty DataFrame Columns: [loan_id, no_of_depe...
0	loan_amount	Empty DataFrame Columns: [loan_id, no_of_depe...
0	loan_term	Empty DataFrame Columns: [loan_id, no_of_depe...
0	cibil_score	Empty DataFrame Columns: [loan_id, no_of_depe...
0	residential_assets_value	Empty DataFrame Columns: [loan_id, no_of_depe...
0	commercial_assets_value	Empty DataFrame Columns: [loan_id, no_of_depe...
0	luxury_assets_value	Empty DataFrame Columns: [loan_id, no_of_depe...
0	bank_asset_value	Empty DataFrame Columns: [loan_id, no_of_depe...

In [24]:

```
dataFrame.shape
```

Out[24]:

(4269, 13)

Filter Methods for feature selection

information gain

first converting categorical columns into numerial ones to apply filter methods

In

```
from sklearn.preprocessing import LabelEncoder

# Create a copy of the original DataFrame
encoded_df = dataFrame.copy()

# Initialize LabelEncoder
label_encoder = LabelEncoder()

# Iterate over columns and apply label encoding for categorical columns
for column in dataFrame.select_dtypes(include='object').columns:
    if len(dataFrame[column].unique()) <= 2:
        # For binary (two-level) categorical columns, use LabelEncoder
        encoded_df[column] = label_encoder.fit_transform(dataFrame[column])
    else:
        # For multi-level categorical columns, consider other encoding methods (e.g., one-hot encoding)
        print(f"Column '{column}' has more than two unique values. Consider other encoding methods.")

print("Encoded DataFrame:")
encoded_df
```

Encoded DataFrame:

Out[65]:

	loan_id	no_of_dependents	education	self_employed	income_annum	loan_amount	loan_term	cibil_score	residential_assets_value
0	1	2	0	0	9600000	29900000	12	778	2400000.0
1	2	0	1	1	4100000	12200000	8	417	2700000.0
2	3	3	0	0	9100000	29700000	20	506	7100000.0
3	4	3	0	0	8200000	30700000	8	467	18200000.0
4	5	5	1	1	9800000	24200000	20	382	12400000.0
...	...	...	...	...	...	...	...	...	..
4264	4265	5	0	1	1000000	2300000	12	317	2800000.0
4265	4266	0	1	1	3300000	11300000	20	559	4200000.0
4266	4267	2	1	0	6500000	23900000	18	457	1200000.0
4267	4268	1	1	0	4100000	12800000	8	780	8200000.0
4268	4269	1	0	0	9200000	29700000	10	607	17800000.0

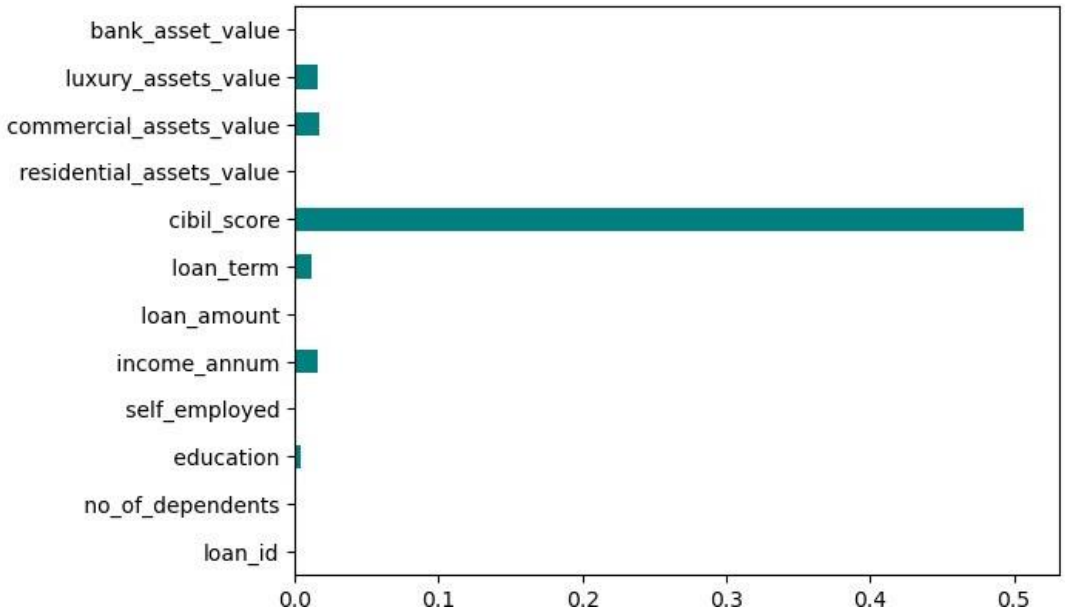
4269 rows × 13 columns

In [66]:

```
x = encoded_df.drop(' loan_status', axis=1)  # Assuming 'target_feature' is the target feature
y = encoded_df[' loan_status']
```

In

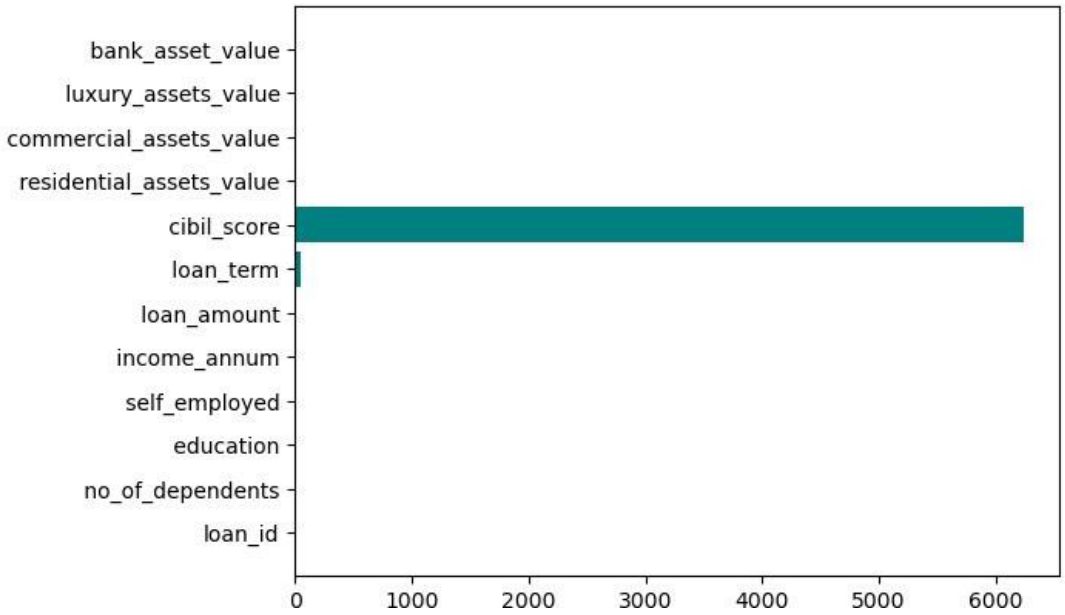
```
#for feature selection based on mutual information for a classification task.
from sklearn.feature_selection import mutual_info_classif

importance = mutual_info_classif(x, y)
feat_importance = pd.Series(importance, encoded_df.columns[0:len(encoded_df.columns) - 1]) feat_importance.plot(kind='barh', color='teal')
plt.show()
```



Select KBest

In

```
from sklearn.feature_selection import SelectKBest, f_classif

# Select features using SelectKBest and f_classif
selector = SelectKBest(score_func=f_classif, k=3) X_new
= selector.fit_transform(x, y)

# Get the selected features
mask = selector.get_support()
selected_features = x.columns[mask]

# Plot a histogram graph
plt.barh(range(len(selector.scores_)), selector.scores_, color='teal')
plt.yticks(range(len(x.columns)), x.columns)
plt.show()

# Print selected features
print("Selected features:", selected_features)
```



Selected features: Index([' no_of_dependents', ' loan_term', ' cibil_score'], dtype='object')

Fisher Score

In

```
from sklearn.feature_selection import f_classif

# Perform feature selection using f_classif
fisher_score, p_value = f_classif(x, y)

# Print the results
for i, column in enumerate(encoded_df.columns[:-1]):  # Exclude the target column
    print("Feature:", column)
    print("Fisher Score:", fisher_score[i])
    print("p-value:", p_value[i])
    print("          ----------")
```

```
Feature: loan_id
Fisher Score: 1.3349674319819718
p-value: 0.2479881190731431
----------
Feature: no_of_dependents
Fisher Score: 1.4006001729702904
p-value: 0.23668903607767336
----------
Feature: education
Fisher Score: 0.10320162169880803
p-value: 0.7480366078913321
----------
Feature: self_employed
Fisher Score: 0.0005064308399286643
p-value: 0.9820469589094434
----------
Feature: income_annum
Fisher Score: 0.9846688347948409
p-value: 0.32110512407104685
----------
Feature: loan_amount
Fisher Score: 1.1131763752970667
p-value: 0.2914522504516055
----------
Feature: loan_term
Fisher Score: 55.22545796738981
p-value: 1.291185463829176e-13
----------
Feature: cibil_score
Fisher Score: 6235.054590534256
p-value: 0.0
----------
Feature: residential_assets_value
Fisher Score: 0.958402864749625
p-value: 0.32764510326774676
----------
Feature: commercial_assets_value
Fisher Score: 0.33035269381476207
p-value: 0.565481784235544
----------
Feature: luxury_assets_value
Fisher Score: 1.020728281020977
p-value: 0.3124036068561609
----------
Feature: bank_asset_value
Fisher Score: 0.19418768809834083
p-value: 0.6594761809444952
----------
```

Mean absooute difference

In

```
mad = x.mad(axis=0)
print("Mean absolute deviation of columns:")
print(mad)
```

```
Mean absolute deviation of columns:
loan_id                     1.067250e+03
 no_of_dependents           1.488298e+00
 education                  4.999901e-01
 self_employed              4.999736e-01
 income_annum               2.424446e+06
 loan_amount                7.580930e+06
 loan_term                  4.963065e+00
 cibil_score                1.492619e+02
 residential_assets_value   5.291511e+06
 commercial_assets_value    3.588533e+06
 luxury_assets_value        7.632964e+06
 bank_asset_value           2.668145e+06
dtype: float64
```

Forward feature selection

In [33]:

```
from sklearn.linear_model import LogisticRegression
from mlxtend.feature_selection import SequentialFeatureSelector as SFS

# 4. Apply Model
lr = LogisticRegression(class_weight='balanced', solver='lbfgs', random_state=42, n_jobs=-1, max_iter=500) lr.fit(x,
y)

# 5. Select best features
bfs = SFS(lr,
          k_features='best',
          forward=True,
          floating=False,
          verbose=2,
          scoring='accuracy', cv=0)
bfs = bfs.fit(x, y)

# 6. Print feature list
features = list(bfs.k_feature_names_)
print(features)
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  1 out of  1 | elapsed:    0.5s remaining:    0.0s
[Parallel(n_jobs=1)]: Done 12 out of 12 | elapsed:    2.8s finished

[2024-01-02 22:37:05] Features: 1/12 -- score: 0.951042398688217 4[Parallel(n_jobs=1)]: Using backend
SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  1 out of  1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done 11 out of 11 | elapsed:    0.7s finished

[2024-01-02 22:37:06] Features: 2/12 -- score: 0.9505739048957601[Parallel(n_jobs=1)]: Using backend
SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  1 out of  1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done 10 out of 10 | elapsed:    0.2s finished

[2024-01-02 22:37:06] Features: 3/12 -- score: 0.9508081517919887[Parallel(n_jobs=1)]: Using backend
SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  1 out of  1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  9 out of  9 | elapsed:    0.6s finished

[2024-01-02 22:37:07] Features: 4/12 -- score: 0.9501054111033029[Parallel(n_jobs=1)]: Using backend
SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  1 out of  1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  8 out of  8 | elapsed:    0.2s finished

[2024-01-02 22:37:07] Features: 5/12 -- score: 0.949402670414617[Parallel(n_jobs=1)]: Using backend
SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  1 out of  1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  7 out of  7 | elapsed:    1.0s finished

[2024-01-02 22:37:08] Features: 6/12 -- score: 0.9226985242445538[Parallel(n_jobs=1)]: Using backend
SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  1 out of  1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  6 out of  6 | elapsed:    0.1s finished

[2024-01-02 22:37:08] Features: 7/12 -- score: 0.7390489576013117[Parallel(n_jobs=1)]: Using backend
SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  1 out of  1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  5 out of  5 | elapsed:    0.1s finished

[2024-01-02 22:37:08] Features: 8/12 -- score: 0.7390489576013117[Parallel(n_jobs=1)]: Using backend
SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  1 out of  1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  4 out of  4 | elapsed:    0.0s finished

[2024-01-02 22:37:08] Features: 9/12 -- score: 0.7388147107050832[Parallel(n_jobs=1)]: Using backend
SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  1 out of  1 | elapsed:    0.0s remaining:    0.0s

[' cibil_score']

[Parallel(n_jobs=1)]: Done  3 out of  3 | elapsed:    0.1s finished

[2024-01-02 22:37:08] Features: 10/12 -- score: 0.7434996486296557[Parallel(n_jobs=1)]: Using backen
d SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  1 out of  1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  2 out of  2 | elapsed:    0.0s finished

[2024-01-02 22:37:08] Features: 11/12 -- score: 0.624033731553057[Parallel(n_jobs=1)]: Using backend
SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  1 out of  1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  1 out of  1 | elapsed:    0.0s finished

[2024-01-02 22:37:08] Features: 12/12 -- score: 0.5427500585617241
```

Exhaustive Feature Selection:

In

```
from sklearn.neighbors import KNeighborsClassifier
from mlxtend.feature_selection import ExhaustiveFeatureSelector as EFS

x = encoded_df.drop(' loan_status', axis=1)  # Assuming 'target_feature' is the target feature
y = encoded_df[' loan_status']

knn = KNeighborsClassifier(n_neighbors=3) efs1 =
EFS(knn,
                min_features=1,
                max_features=4,
                scoring='accuracy',
                print_progress=True, cv=5)
efs1 = efs1.fit(x, y)

print('Best accuracy score: %.2f' % efs1.best_score_) print('Best
subset (indices): ', efs1.best_idx_)
print('Best subset (corresponding names): ', efs1.best_feature_names_)
```

```
Features:  793/793

Best accuracy score: 0.95
Best subset (indices):  (2, 3, 6, 7)
Best subset (corresponding names):  (' education', ' self_employed', ' loan_term', ' cibil_score')
```

Random Forest

In [36]:

```
# Apply Random Forest regression algorithm
from sklearn.ensemble import RandomForestRegressor

rf = RandomForestRegressor(random_state=0)
rf.fit(X_train, y_train)

# Print selected features
print(rf.feature_importances_)
print(np.array(features)[rf.feature_importances_ > 0.1])
```

```
[8.34715611e-03 3.46046217e-03 3.96891375e-04 8.49391812e-04
 2.19010605e-02 2.84329259e-02 7.66550462e-02 8.32448350e-01
 8.28227062e-03 6.78039033e-03 7.91008880e-03 4.53596631e-03]
[' cibil_score']
```

Genetic algorithm

In

```
import random
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import pandas as pd

logmodel = LogisticRegression(max_iter=1000)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(x)
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.33, random_state=42)

# defining various steps required for the genetic algorithm
def initialization_of_population(size, n_feat):
    population = []
    for i in range(size):
        chromosome = np.ones(n_feat, dtype=bool)
        chromosome[:int(0.3 * n_feat)] = False
        np.random.shuffle(chromosome)
        population.append(chromosome)
    return population

def fitness_score(population, X_train, X_test, y_train, y_test, logmodel):
    scores = []
    for chromosome in population:
        logmodel.fit(X_train[:, chromosome], y_train)
        predictions = logmodel.predict(X_test[:, chromosome])
        scores.append(accuracy_score(y_test, predictions))
    scores, population = np.array(scores), np.array(population)
    inds = np.argsort(scores)
    return list(scores[inds][::-1]), list(population[inds, :][::-1])

def selection(pop_after_fit, n_parents):
    population_next_gen = []
    for i in range(n_parents):
        population_next_gen.append(pop_after_fit[i])
    return population_next_gen

def crossover(pop_after_sel):
    population_next_gen = pop_after_sel.copy()
    for i in range(len(pop_after_sel)):
        child = pop_after_sel[i].copy()
        child[3:7] = pop_after_sel[(i + 1) % len(pop_after_sel)][3:7]
        population_next_gen.append(child)
    return population_next_gen

def mutation(pop_after_cross, mutation_rate):
    population_next_gen = []
    for i in range(0, len(pop_after_cross)):
        chromosome = pop_after_cross[i].copy()
        for j in range(len(chromosome)):
            if random.random() < mutation_rate:
                chromosome[j] = not chromosome[j]
        population_next_gen.append(chromosome)
    return population_next_gen

def generations(size, n_feat, n_parents, mutation_rate, n_gen, X_train, X_test, y_train, y_test, logmodel):
    best_chromo = []
    best_score = []
    population_next_gen = initialization_of_population(size, n_feat)
    for i in range(n_gen):
        scores, pop_after_fit = fitness_score(population_next_gen, X_train, X_test, y_train, y_test, logmodel)
        print(scores[:2])
        pop_after_sel = selection(pop_after_fit, n_parents)
        pop_after_cross = crossover(pop_after_sel)
        population_next_gen = mutation(pop_after_cross, mutation_rate)
        best_chromo.append(pop_after_fit[0])
        best_score.append(scores[0])
    return best_chromo, best_score

# Implementing GA
# Replace the following placeholders with your actual data
# X_train, X_test, y_train, y_test, logmodel
chromo, score = generations(size=200, n_feat=12, n_parents=100, mutation_rate=0.10, n_gen=5, X_train=X_train,
                            X_test=X_test, y_train=y_train, y_test=y_test, logmodel=logmodel)
logmodel.fit(X_train[:, chromo[-1]], y_train)
predictions = logmodel.predict(X_test[:, chromo[-1]])
print("Accuracy score after genetic algorithm is = " + str(accuracy_score(y_test, predictions)))
print('List of important features:', chromo[-1])
```

```
[0.9247693399574166, 0.9240596167494677]
[0.9254790631653655, 0.9247693399574166]
[0.9254790631653655, 0.9247693399574166]
[0.9254790631653655, 0.9247693399574166]
[0.9254790631653655, 0.9254790631653655]
Accuracy score after genetic algorithm is = 0.9254790631653655
List of important features: [ True False False  True False  True False  True  True  True  True  True
]
```

In [38]:

```
x.columns
```

Out[38]:

```
Index(['loan_id', ' no_of_dependents', ' education', ' self_employed',
       ' income_annum', ' loan_amount', ' loan_term', ' cibil_score',
       ' residential_assets_value', ' commercial_assets_value',
       ' luxury_assets_value', ' bank_asset_value'],
      dtype='object')
```

so according to all the feature selection method cibil_score is best one for prediction

dropping those columns which have zero impact on the prediction of loan_status

In [39]:

```
# Drop specified columns
columns_to_drop = ['loan_id', ' income_annum', ' commercial_assets_value'] final_df =
encoded_df.drop(columns=columns_to_drop)

# Display the resulting DataFrame
final_df
```

Out[39]:

	no_of_dependents	education	self_employed	loan_amount	loan_term	cibil_score	residential_assets_value	luxury_assets_value
0	2	0	0	29900000	12	778	2400000.0	22700000
1	0	1	1	12200000	8	417	2700000.0	8800000
2	3	0	0	29700000	20	506	7100000.0	33300000
3	3	0	0	30700000	8	467	18200000.0	23300000
4	5	1	1	24200000	20	382	12400000.0	29400000
...	...	...	...	...	...	...	...	...
4264	5	0	1	2300000	12	317	2800000.0	3300000
4265	0	1	1	11300000	20	559	4200000.0	11000000
4266	2	1	0	23900000	18	457	1200000.0	18100000
4267	1	1	0	12800000	8	780	8200000.0	14100000
4268	1	0	0	29700000	10	607	17800000.0	35700000

4269 rows × 10 columns

Decision Tree

In [70]:

```
x = final_df.drop(' loan_status', axis=1)  # Assuming 'target_feature' is the target feature
y = final_df[' loan_status']
```

In [71]:

```
# Import necessary libraries
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split

# Perform train and test split
X_train, X_test, y_train, y_test = train_test_split(x, y, random_state=0)

y_train_reshaped = y_train.values.reshape(-1, 1)

# Build Decision Tree classifier
clf = DecisionTreeClassifier(criterion='gini', max_depth=4, random_state=1)
clf.fit(X_train, y_train_reshaped)
```

Out[71]:

```
DecisionTreeClassifier(max_depth=4, random_state=1)
```

In [44]:

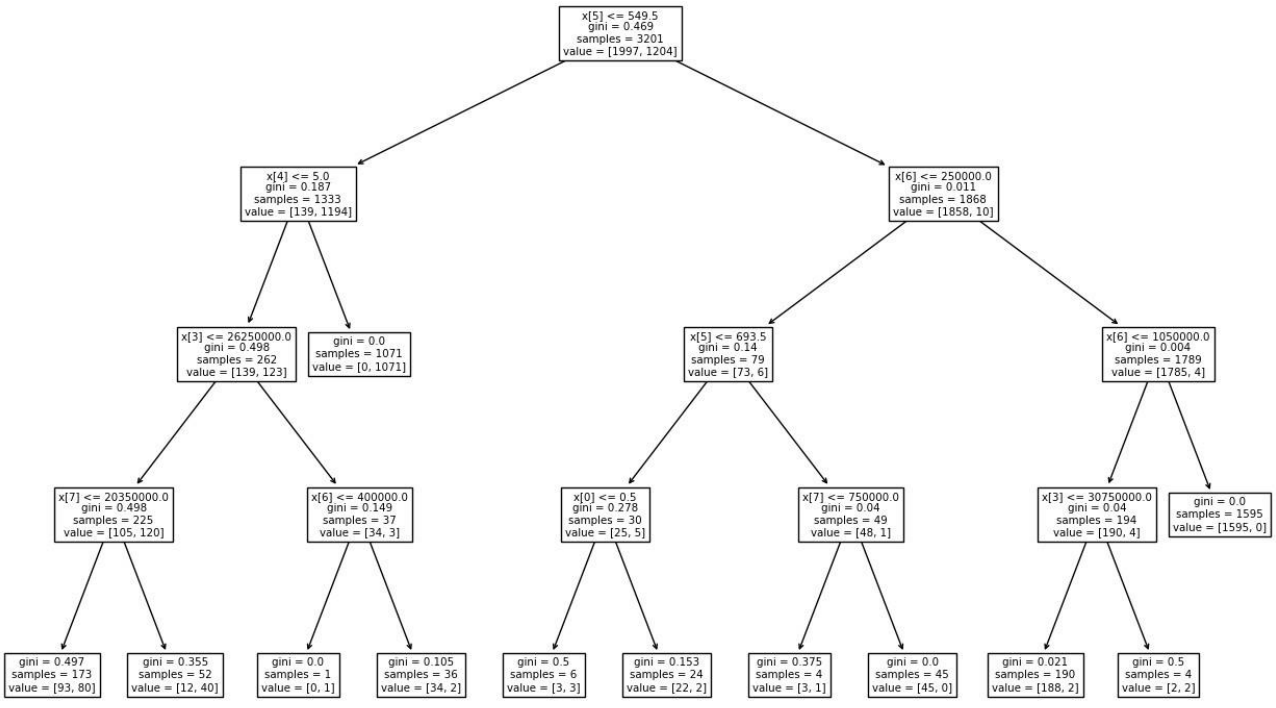
```
# Extract relevant attributes from the tree structure
n_nodes = clf.tree_.node_count
children_left = clf.tree_.children_left
children_right = clf.tree_.children_right
feature = clf.tree_.feature
threshold = clf.tree_.threshold
node_depth = np.zeros(shape=n_nodes, dtype=np.int64)
is_leaves = np.zeros(shape=n_nodes, dtype=bool)
stack = [(0, 0)]  # start with the root node id (0) and its depth (0)

# Traverse the tree structure to compute depth and identify leaves
while len(stack) > 0:
    # 'pop' ensures each node is only visited once
    node_id, depth = stack.pop()
    node_depth[node_id] = depth

    # If the left and right child of a node is not the same, we have a split node
    is_split_node = children_left[node_id] != children_right[node_id]

    # If a split node, append left and right children and depth to 'stack'
    # so we can loop through them
    if is_split_node:
        stack.append((children_left[node_id], depth + 1))
        stack.append((children_right[node_id], depth + 1))
    else:
        is_leaves[node_id] = True

# Print the binary tree structure
print("The binary tree structure has {n} nodes and has the following tree structure:\n".format(n=n_nodes))
for i in range(n_nodes):
    if is_leaves[i]:
        print("{space}node={node} is a leaf node.".format(space=node_depth[i] * "\t", node=i))
    else:
        print("{space}node={node} is a split node: "
              "go to node {left} if X[:, {feature}] <= {threshold} "
              "else to node {right}.".format(space=node_depth[i] * "\t",
                                             node=i,
                                             left=children_left[i],
                                             feature=feature[i],
                                             threshold=threshold[i],
                                             right=children_right[i]))
```

The binary tree structure has 23 nodes and has the following tree structure:

```
node=0 is a split node: go to node 1 if X[:, 5] <= 549.5 else to node 10.
        node=1 is a split node: go to node 2 if X[:, 4] <= 5.0 else to node 9.
                node=2 is a split node: go to node 3 if X[:, 3] <= 26250000.0 else to node 6.
                        node=3 is a split node: go to node 4 if X[:, 7] <= 20350000.0 else to node 5
.
                                node=4 is a leaf node.
                                node=5 is a leaf node.
                        node=6 is a split node: go to node 7 if X[:, 6] <= 400000.0 else to node 8.
                                node=7 is a leaf node.
                                node=8 is a leaf node.
                node=9 is a leaf node.
        node=10 is a split node: go to node 11 if X[:, 6] <= 250000.0 else to node 18.
                node=11 is a split node: go to node 12 if X[:, 5] <= 693.5 else to node 15.
                        node=12 is a split node: go to node 13 if X[:, 0] <= 0.5 else to node 14.
                                node=13 is a leaf node.
                                node=14 is a leaf node.
                        node=15 is a split node: go to node 16 if X[:, 7] <= 750000.0 else to node 1
7.
                                node=16 is a leaf node.
                                node=17 is a leaf node.
                node=18 is a split node: go to node 19 if X[:, 6] <= 1050000.0 else to node 22.
                        node=19 is a split node: go to node 20 if X[:, 3] <= 30750000.0 else to node
21.
                                node=20 is a leaf node.
                                node=21 is a leaf node.
                        node=22 is a leaf node.
```

In [45]:

```
from sklearn.tree import plot_tree
import matplotlib.pyplot as plt

# Plot the decision tree
plt.figure(figsize=(16, 10))
plot_tree(clf)
plt.show()
```



KNN Algorithm

In

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# Initialize and train the KNN classifier
knn_classifier = KNeighborsClassifier(n_neighbors=3)
knn_classifier.fit(X_train, y_train)

# Make predictions with KNN on the test set
knn_predictions = knn_classifier.predict(X_test)

#Evaluate the performane of KNN
knn_accuracy = accuracy_score(y_test, knn_predictions)
knn_classification_report = classification_report(y_test, knn_predictions)
knn_confusion_matrix = confusion_matrix(y_test, knn_predictions)

# Display results
print("KNN Accuracy:", knn_accuracy)
print("\nKNN Classification Report:")
print(knn_classification_report)
print("\nKNN Confusion Matrix:")
print(knn_confusion_matrix)
```

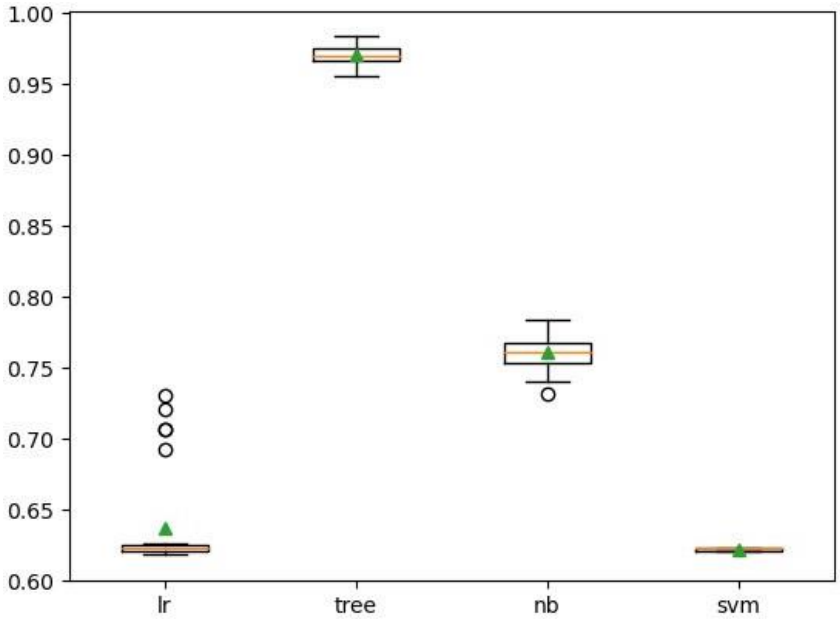
```
KNN Accuracy: 0.5430711610486891

KNN Classification Report:
              precision    recall  f1-score   support

           0       0.62      0.69      0.65       659
           1       0.38      0.30      0.34       409

    accuracy                           0.54      1068
   macro avg       0.50      0.50      0.49      1068
weighted avg       0.52      0.54      0.53      1068


KNN Confusion Matrix:
[[456 203]
 [285 124]]
```

Base models and voting

In

```
# evaluate standard models on the synthetic dataset
from numpy import mean
from numpy import std
from  sklearn.datasets  import  make_classification
from sklearn.model_selection import cross_val_score
from  sklearn.model_selection  import  RepeatedStratifiedKFold
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from matplotlib import pyplot
from  sklearn.ensemble  import  VotingClassifier
# get the dataset

# get a list of models to evaluate
def get_models ():
    models = list()
    models.append(("lr", LogisticRegression()))
    models.append(("tree", DecisionTreeClassifier()))
    models.append (('nb', GaussianNB()))
    models.append(("svm", SVC(probability=True)))
    return models

# evaluate a give model using cross - validation
def evaluate_model (model , X, y):
    # define the model evaluation procedure
    cv = RepeatedStratifiedKFold( n_splits=10 , n_repeats=3 , random_state=1)
    # evaluate the model
    scores = cross_val_score(model, X, y, scoring='accuracy', cv=cv, n_jobs= -1)
    return scores
# get the models to evaluate
models = get_models()
# evaluate the models and store results
results , names = list() , list()
for name , model in models :
    # evaluate model
    scores =evaluate_model ( model ,x , y)
    # Store results
    results.append(scores)
    names.append(name)
    #summarize result
    print('>%s %.3f (%.3f)' % (name, mean(scores), std(scores)))
# plot model performance for comparison
pyplot.boxplot (results , labels=names , showmeans=True )
pyplot.show ()

#create the ensemble
ensemble = VotingClassifier( estimators = models , voting = 'soft')
# define the evaluation procedure
cv = RepeatedStratifiedKFold ( n_splits =10 , n_repeats =3 , random_state =1)
# evaluate the ensemble
scores = cross_val_score (ensemble , x, y, scoring ='accuracy', cv=cv , n_jobs=-1)
# summarize the result
print('Mean Accuracy: %.3f (%.3f)' % (mean(scores), std(scores)))
```

>lr 0.637 (0.034)
>tree 0.971 (0.007)
>nb 0.761 (0.011)
>svm 0.622 (0.001)



Mean Accuracy: 0.966 (0.012)

Ensemble pruning

In

```
# evaluate a list of models
def evaluate_ensemble (models , X, y):
    # check for no models
    if len(models) == 0:
        return 0.0
    # create the ensemble
    ensemble = VotingClassifier(estimators = models , voting = 'soft')
    # define the evaluation procedure
    cv = RepeatedStratifiedKFold ( n_splits=10 , n_repeats=3 , random_state=1)
    # evaluate the ensemble
    scores = cross_val_score (ensemble , X, y, scoring ='accuracy', cv=cv ,n_jobs= -1)
    # return mean score
    return mean(scores)

# perform a single round of pruning the ensemble
def prune_round (models_in , X, y):
    # establish a baseline
    baseline = evaluate_ensemble (models_in , X, y)
    best_score , removed = baseline , None
    # enumerate removing each candidate and see if we can improve performance
    for m in models_in :
        # copy the list of chosen models
        dup = models_in.copy()
        # remove this model
        dup.remove(m)
        #evaluate new ensemble
        result = evaluate_ensemble (dup, X , y)
        # check for new best
        if result > best_score :
            # store the new best
            best_score , removed = result , m
    return best_score , removed

# prune an ensemble from scratch
def prune_ensemble ( models , X , y):
    best_score = 0.0
    # prune ensemble until no further improvement
    while True :
        # remove one model to the ensemble
        score , removed = prune_round(models ,X, y)
        # check for no improvement
        if removed is None :
            print ('>no further improvement')
            break
        # keep track of best score
        best_score = score
        # remove model from the list
        models.remove( removed )
        # report results along the way
        print('>%.3f (removed: %s)' % (score, removed[0]))
        return best_score , models

# get the models to evaluate
models = get_models ()
# prune the ensemble
score , model_list = prune_ensemble(models,x, y)
names = ','. join ([ n for n , _ in model_list ])
print ('Models : %s '% names )
print ('Final Mean Accuracy : %.3f' % score )
```

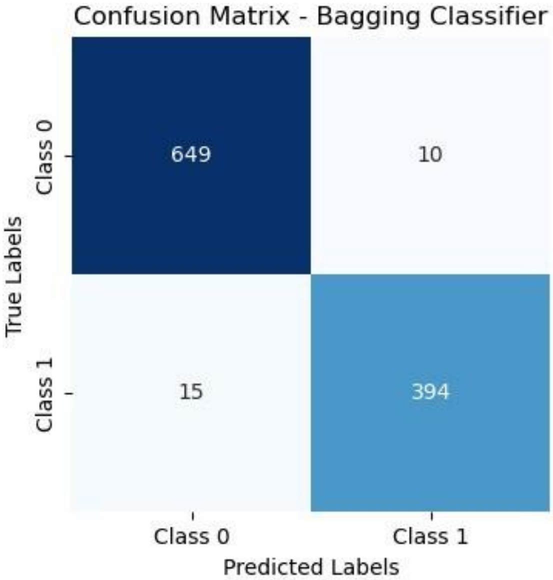
```
>0.972 (removed: svm)
Models : lr,tree,nb
Final Mean Accuracy : 0.972
```

Bagging and Random Forest classifier

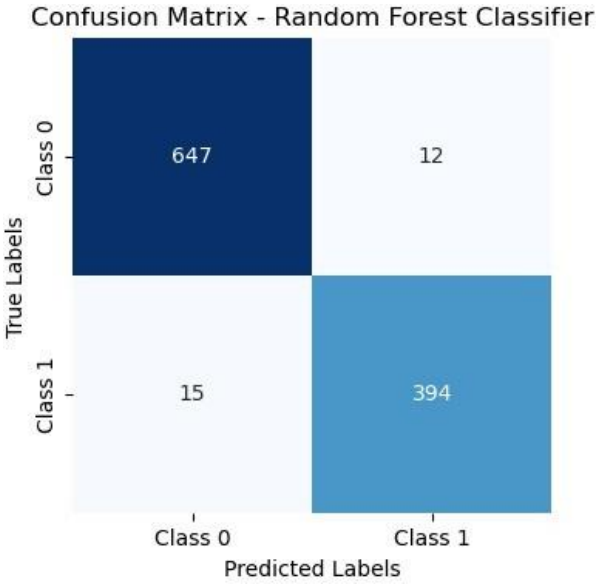
In

```
from sklearn.model_selection import KFold, cross_val_score
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn . ensemble import RandomForestClassifier
from sklearn.datasets import load_diabetes
from sklearn.metrics import f1_score, accuracy_score
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd

max_features = 3
kfold = KFold(n_splits=10, shuffle=True, random_state=2020)
decision_tree = DecisionTreeClassifier(max_features = max_features)
num_trees = 100

# Decision Tree base estimator
bagging_model = BaggingClassifier(base_estimator=decision_tree, n_estimators = num_trees, random_state=2020)

results = cross_val_score(bagging_model, X_train, y_train, cv=kfold)
print("Bagging classifier Accuracy: %0.2f (+/- %0.2f)" % (results.mean(), results.std()))

# Bagging Classifier with Decision Tree base estimator
bagging_model.fit(X_train, y_train)
y_pred_bagging = bagging_model.predict(X_test)

accuracy_bagging = accuracy_score(y_test, y_pred_bagging)
f1_bagging = f1_score(y_test, y_pred_bagging)

print("Bagging Classifier Accuracy: %0.2f" % accuracy_bagging)
print("Bagging Classifier F1 Score: %0.2f" % f1_bagging)

# Bagging Classifier with Decision Tree base estimator
conf_matrix_bagging = confusion_matrix(y_test, y_pred_bagging)

# Plot confusion matrix for Bagging Classifier
plt.figure(figsize=(4, 4))
sns.heatmap(conf_matrix_bagging, annot=True, fmt="d", cmap="Blues", cbar=False,
            xticklabels=['Class 0', 'Class 1'], yticklabels=['Class 0', 'Class 1'])
plt.title("Confusion Matrix - Bagging Classifier")
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.show()

# Random Forest Classifier
num_trees_rf = 100
max_features_rf = 3
kfold_rf = KFold(n_splits=10, shuffle=True, random_state=2020)

rf_model = RandomForestClassifier(n_estimators=num_trees_rf, max_features= max_features)
# Cross-validation
results_rf = cross_val_score(rf_model, X_train, y_train, cv=kfold_rf)
print("Random Forest classifier Accuracy: %0.2f (+/- %0.2f)" % (results_rf.mean(),results_rf.std()))

#accuracy and F1 score
rf_model.fit(X_train, y_train)
y_pred_rf = rf_model.predict(X_test)

accuracy_rf = accuracy_score(y_test, y_pred_rf)
f1_rf = f1_score(y_test, y_pred_rf)

print("Random Forest Classifier Accuracy: %0.2f" % accuracy_rf)
print("Random Forest Classifier F1 Score: %0.2f" % f1_rf)

# Random Forest Classifier
conf_matrix_rf = confusion_matrix(y_test, y_pred_rf)

# Plot confusion matrix for Random Forest Classifier
plt.figure(figsize=(4, 4))
sns.heatmap(conf_matrix_rf, annot=True, fmt="d", cmap="Blues", cbar=False,
            xticklabels=['Class 0', 'Class 1'], yticklabels=['Class 0', 'Class 1'])
plt.title("Confusion Matrix - Random Forest Classifier")
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.show()
```

```
/opt/anaconda3/envs/myenv/lib/python3.9/site-packages/sklearn/ensemble/_base.py:156: FutureWarning:
`base_estimator` was renamed to `estimator` in version 1.2 and will be removed in 1.4.
  warnings.warn(
/opt/anaconda3/envs/myenv/lib/python3.9/site-packages/sklearn/ensemble/_base.py:156: FutureWarning:
`base_estimator` was renamed to `estimator` in version 1.2 and will be removed in 1.4.
  warnings.warn(
/opt/anaconda3/envs/myenv/lib/python3.9/site-packages/sklearn/ensemble/_base.py:156: FutureWarning:
`base_estimator` was renamed to `estimator` in version 1.2 and will be removed in 1.4.
  warnings.warn(
/opt/anaconda3/envs/myenv/lib/python3.9/site-packages/sklearn/ensemble/_base.py:156: FutureWarning:
`base_estimator` was renamed to `estimator` in version 1.2 and will be removed in 1.4.
  warnings.warn(
/opt/anaconda3/envs/myenv/lib/python3.9/site-packages/sklearn/ensemble/_base.py:156: FutureWarning:
`base_estimator` was renamed to `estimator` in version 1.2 and will be removed in 1.4.
  warnings.warn(
/opt/anaconda3/envs/myenv/lib/python3.9/site-packages/sklearn/ensemble/_base.py:156: FutureWarning:
`base_estimator` was renamed to `estimator` in version 1.2 and will be removed in 1.4.
  warnings.warn(
/opt/anaconda3/envs/myenv/lib/python3.9/site-packages/sklearn/ensemble/_base.py:156: FutureWarning:
`base_estimator` was renamed to `estimator` in version 1.2 and will be removed in 1.4.
  warnings.warn(
/opt/anaconda3/envs/myenv/lib/python3.9/site-packages/sklearn/ensemble/_base.py:156: FutureWarning:
`base_estimator` was renamed to `estimator` in version 1.2 and will be removed in 1.4.
  warnings.warn(
/opt/anaconda3/envs/myenv/lib/python3.9/site-packages/sklearn/ensemble/_base.py:156: FutureWarning:
`base_estimator` was renamed to `estimator` in version 1.2 and will be removed in 1.4.
  warnings.warn(
/opt/anaconda3/envs/myenv/lib/python3.9/site-packages/sklearn/ensemble/_base.py:156: FutureWarning:
`base_estimator` was renamed to `estimator` in version 1.2 and will be removed in 1.4.
  warnings.warn(

Bagging classifier Accuracy: 0.97 (+/- 0.01)

/opt/anaconda3/envs/myenv/lib/python3.9/site-packages/sklearn/ensemble/_base.py:156: FutureWarning:
`base_estimator` was renamed to `estimator` in version 1.2 and will be removed in 1.4.
  warnings.warn(

Bagging Classifier Accuracy: 0.98
Bagging Classifier F1 Score: 0.97
```




```
Random Forest classifier Accuracy: 0.97 (+/- 0.01)
Random Forest Classifier Accuracy: 0.97
Random Forest Classifier F1 Score: 0.97
```



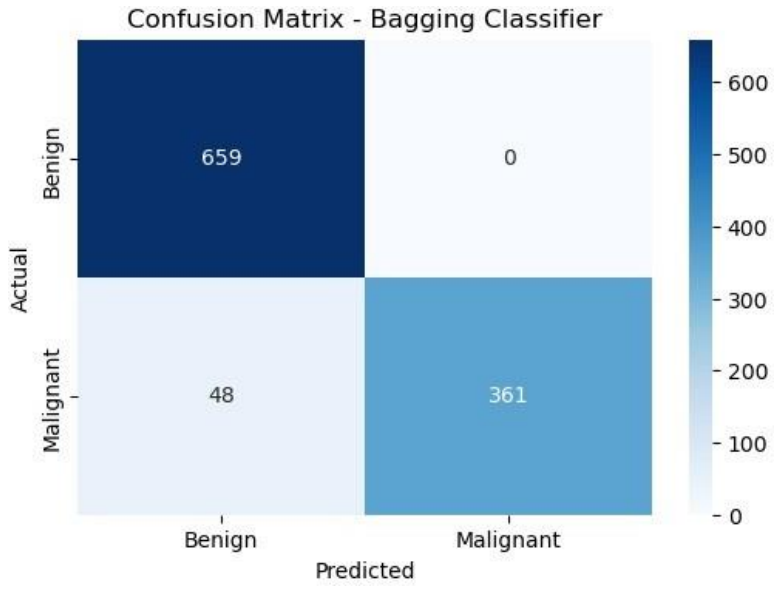
Stacking

In [76]:

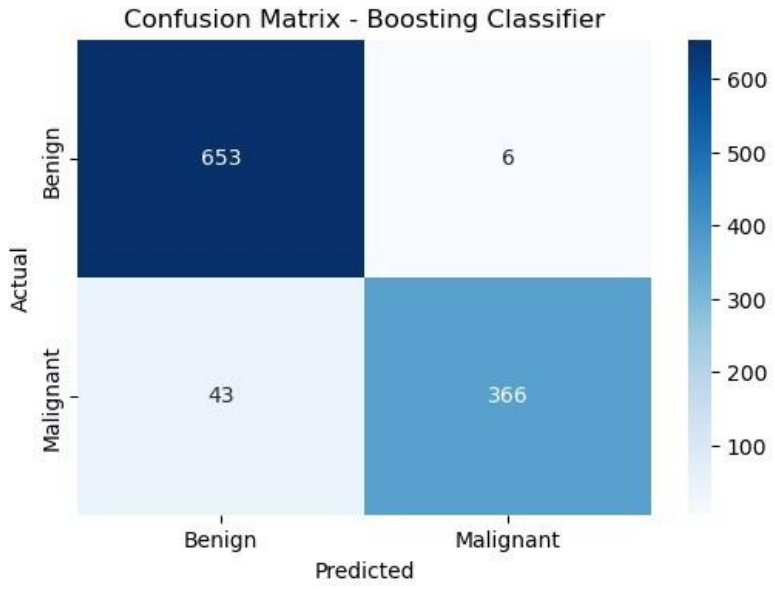
```
import numpy as np
from sklearn.ensemble import AdaBoostClassifier, RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import f1_score, accuracy_score
from sklearn.datasets import load_breast_cancer
from sklearn.metrics import confusion_matrix
from sklearn.ensemble import RandomForestClassifier
import seaborn as sns
import matplotlib.pyplot as plt



class NumberOfClassifierException(Exception):
    pass

class Stacking():
    def __init__(self, classifiers):
        if len(classifiers) < 2:
            raise numberOfClassifierException (" You must fit your classifier with 2 classifiers at least ");
        else:
            self._classifiers = classifiers

    def fit(self, data_x, data_y):
        stacked_data_x = data_x.copy()
        for classifier in self._classifiers[:-1]:
            classifier.fit(data_x, data_y)
            stacked_data_x = np.column_stack((stacked_data_x ,classifier.predict_proba(data_x)))
        last_classifier = self._classifiers[-1]
        last_classifier.fit(stacked_data_x, data_y)

    def predict(self, data_x):
        stacked_data_x = data_x.copy()
        for classifier in self._classifiers[:-1]:
            prob_predictions = classifier.predict_proba(data_x)
            stacked_data_x = np.column_stack ((stacked_data_x , prob_predictions))
        last_classifier = self._classifiers[-1]
        return  last_classifier.predict(stacked_data_x)


# Creating classifiers
boosting_clf_ada_boost = AdaBoostClassifier( base_estimator=DecisionTreeClassifier(max_depth=1), n_estimators=3)
clf_rf = RandomForestClassifier( n_estimators=200, max_depth=1, random_state=2020)
clf_adaboost = AdaBoostClassifier( base_estimator=DecisionTreeClassifier(max_depth=1, random_state=2020), n_estim
ators=3)

clf_logistic_reg = LogisticRegression(solver='liblinear', random_state=2020)

# Customizing and Exception message
classifiers_list = [clf_rf, clf_adaboost, clf_logistic_reg]
clf_stacking = Stacking(classifiers_list)
# Fit models
clf_rf.fit(X_train, y_train)
boosting_clf_ada_boost.fit(X_train, y_train)
```

```
clf_stacking.fit(X_train, y_train)

# Make predictions
predictions_bagging = clf_rf.predict(X_test)
predictions_boosting = boosting_clf_ada_boost.predict(X_test)
predictions_stacking = clf_stacking.predict(X_test)

# Print results
print("For Bagging: F1 Score {}, Accuracy {}".format(
    round(f1_score(y_test, predictions_bagging), 2),
    round(accuracy_score(y_test, predictions_bagging), 2)
))

# Confusion Matrix for Bagging
conf_mat_bagging = confusion_matrix(y_test, predictions_bagging)

# Plot Confusion Matrix for Bagging
plt.figure(figsize=(6, 4))
sns.heatmap(conf_mat_bagging, annot=True, fmt="d", cmap="Blues", xticklabels=["Benign", "Malignant"], yticklabels
=["Benign", "Malignant"])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix - Bagging Classifier')
plt.show()

print("For Boosting: F1 Score {}, Accuracy {}".format(
    round(f1_score(y_test, predictions_boosting), 2),
    round(accuracy_score(y_test, predictions_boosting), 2)
))

# Confusion Matrix for Bagging
conf_mat_bagging = confusion_matrix(y_test, predictions_boosting)

# Plot Confusion Matrix for Bagging
plt.figure(figsize=(6, 4))
sns.heatmap(conf_mat_bagging, annot=True, fmt="d", cmap="Blues", xticklabels=["Benign", "Malignant"], yticklabels
=["Benign", "Malignant"])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix - Boosting Classifier')
plt.show()

print("For Stacking: F1 Score {}, Accuracy {}".format(
    round(f1_score(y_test, predictions_stacking), 2),
    round(accuracy_score(y_test, predictions_stacking), 2)
))

# Confusion Matrix for Bagging
conf_mat_bagging = confusion_matrix(y_test, predictions_stacking)

# Plot Confusion Matrix for Bagging
plt.figure(figsize=(6, 4))
sns.heatmap(conf_mat_bagging, annot=True, fmt="d", cmap="Blues", xticklabels=["Benign", "Malignant"], yticklabels
=["Benign", "Malignant"])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix - Stacking Classifier')
plt.show()
```

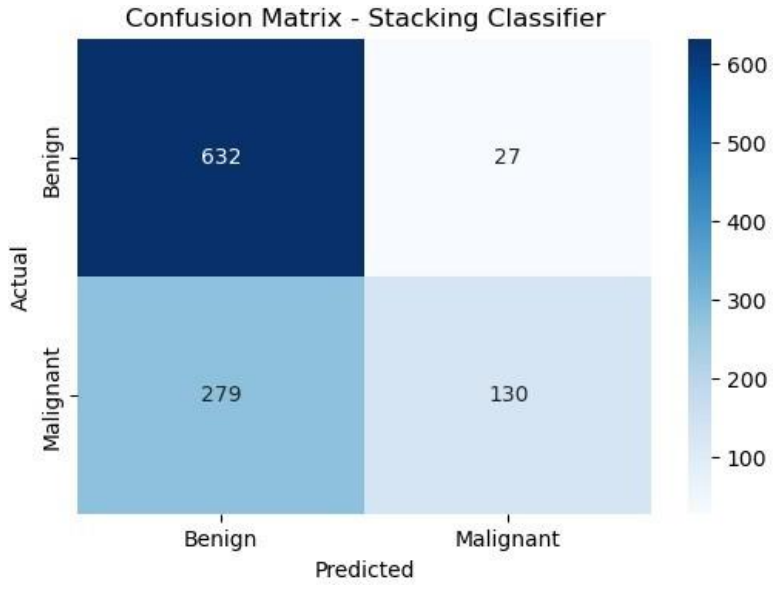
```
/opt/anaconda3/envs/myenv/lib/python3.9/site-packages/sklearn/ensemble/_base.py:156: FutureWarning:
`base_estimator` was renamed to `estimator` in version 1.2 and will be removed in 1.4.
  warnings.warn(
/opt/anaconda3/envs/myenv/lib/python3.9/site-packages/sklearn/ensemble/_base.py:156: FutureWarning:
`base_estimator` was renamed to `estimator` in version 1.2 and will be removed in 1.4.
  warnings.warn(

For Bagging: F1 Score 0.94, Accuracy 0.96
```



For Boosting: F1 Score 0.94, Accuracy 0.95



For Stacking: F1 Score 0.46, Accuracy 0.71



Naive Bayes Classifier

In [77]:

```
class NaiveBayesClassifier:
    def __init__(self, data, target):
        # Constructor to initialize the classifier with input data and target variable
```

```
        self.data = data
        self.target = target
        self.target_prob_dict = {}  # Dictionary to store target class probabilities
        self.cond_prob_list = []  # List to store conditional probabilities
        self.smoothing = False  # Flag to enable/disable Laplace smoothing
        self.k = 1  # Laplace Smoothing parameter

    def compute_target_probabilities(self):
        # Compute probabilities of each target class in the dataset
        total_rows = len(self.data)
        target_counts  =  self.data[self.target].value_counts()

        for level, count in target_counts.items():
            self.target_prob_dict[level] = count / total_rows

    def compute_conditional_probabilities(self):
        # Compute conditional probabilities for each feature given the target class
         for feature in self.data.columns:
             if feature != self.target:
                 feature_levels = self.data[feature].unique()

                for level in feature_levels:
                    for target_level in self.target_prob_dict.keys():
                        count_f_v_t = len(self.data[(self.data[feature] == level) & (self.data[self.target] == ta
rget_level)])
                        count_f_t = len(self.data[self.data[self.target] == target_level])

                        if self.smoothing:
                            # Laplace Smoothing
                            prob = (count_f_v_t + self.k) / (count_f_t + self.k * len(feature_levels))
                        else:
                            prob = count_f_v_t / count_f_t

                        # Store the computed probability in the list
                        self.cond_prob_list.append((feature, level, target_level, round(prob, 3)))

    def fit(self, smoothing=False, k=1):
        # Method to train the Naive Bayes classifier
        self.smoothing = smoothing
        self.k = k

        self.compute_target_probabilities()
        self.compute_conditional_probabilities()

    def predict_instance(self, instance):
        # Predict the target class for a given instance
        prob_dict = {}

        for target_level in self.target_prob_dict.keys():
            prob_prod = 1

            for feature, level, t_level, prob in self.cond_prob_list:
                if feature in instance and instance[feature] == level and t_level == target_level:
                    prob_prod *= prob

            # Multiply the conditional probability with the prior probability
            prob_dict[target_level] = round(prob_prod * self.target_prob_dict[target_level], 4)

        # Return the predicted target class with the highest probability
        return max(prob_dict, key=prob_dict.get)

    def get_probabilities(self):
        # Display the computed target and conditional probabilities
        print("Target probabilities:", self.target_prob_dict)
        print("Conditional features probabilities:", self.cond_prob_list)


# Example usage:
data = {
    'CrdtHistory': ['current', 'paid', 'arrears', 'none', 'current', 'paid', 'arrears', 'none'],
    'GCoApplicant': ['none', 'guarantor', 'coapplicant', 'none', 'none', 'guarantor', 'coapplicant', 'none'],
    'Accommodation': ['own', 'rent', 'free', 'own', 'rent', 'free', 'own', 'rent'],
    'Target': [True, True, True, True, False, False, False, False]
}

df = pd.DataFrame(data)

nbc = NaiveBayesClassifier(final_df, ' loan_status')
nbc.fit(smoothing=True, k=1)

# Prediction for a new instance
query_instance = {' no_of_dependents': 5, ' education': 1, ' self_employed': 1, ' loan_amount': 24300000, ' loan_
term': 12, ' cibil_score': 317, ' residential_assests_value': 18200000.0, ' luxury_assests_value': 29400000, ' ba
```

```
nk_asset_value': 800000.0}

prediction = nbc.predict_instance(query_instance)
print("Prediction:", prediction)
```

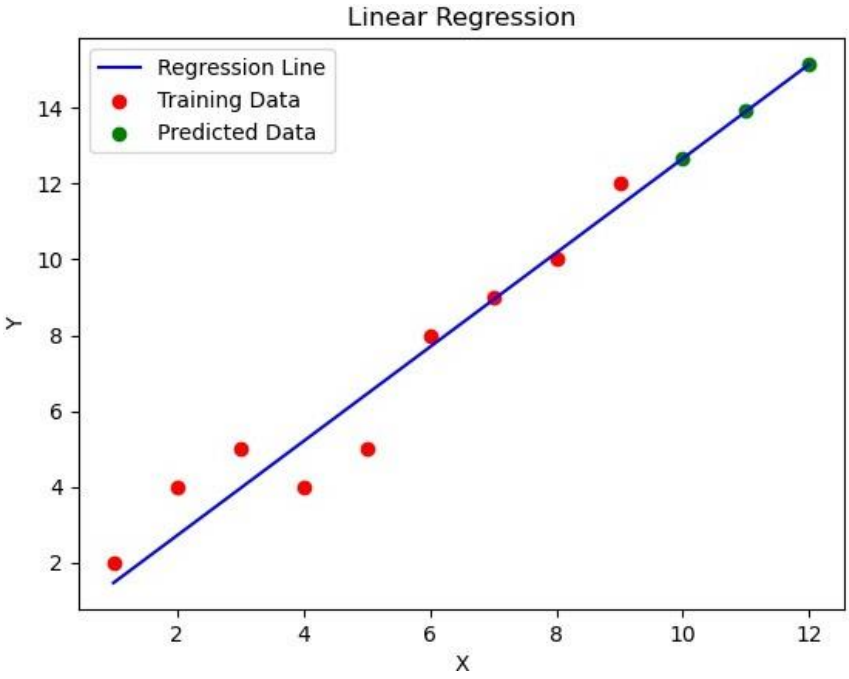
```
Prediction:  0
```

Linear Regression Model

In [90]:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import r2_score

def gradient_descent(X_train, y_train, learning_rate=0.0001, epochs=1000):
    n = len(X_train)
    a_0 = 0.0
    a_1 = 0.0

    for epoch in range(epochs):
        y_pred = a_0 + a_1 * X_train
        error = y_pred - y_train
        mean_sq_err = np.sum(error ** 2) / n

        a_0 = a_0 - learning_rate * 2 * np.sum(error) / n
        a_1 = a_1 - learning_rate * 2 * np.sum(error * X_train) / n

        if epoch % 100 == 0:
            print(f'Mean Squared Error after {epoch} epochs:', mean_sq_err)

    return a_0, a_1

def linear_regression(X_train, y_train, X_test):
    a_0, a_1 = gradient_descent(X_train, y_train)

    y_pred_train = a_0 + a_1 * X_train
    y_pred_test = a_0 + a_1 * X_test

    return y_pred_train, y_pred_test, a_0, a_1



# Performing linear regression
y_pred_train, y_pred_test, a_0, a_1 = linear_regression(X_train, y_train, X_test)

# Combine training and test sets for plotting
X_combined = np.concatenate((X_train, X_test))
y_combined = np.concatenate((y_train, y_pred_test))

# Calculating R2 Score
r2 = r2_score(y_train, y_pred_train)
print('R2 Score:', r2)

# Plotting
plt.scatter(X_train, y_train, color='red', label='Training Data')
plt.plot(X_combined, a_0 + a_1 * X_combined, color='blue', label='Regression Line')
plt.scatter(X_test, y_pred_test, color='green', label='Predicted Data')
plt.legend()
plt.xlabel('X')
plt.ylabel('Y')
plt.title('Linear Regression')
plt.show()
```

```
Mean Squared Error after 0 epochs: 52.77777777777778
Mean Squared Error after 100 epochs: 14.917031978341372
Mean Squared Error after 200 epochs: 4.62610266991101
Mean Squared Error after 300 epochs: 1.8286803492572576
Mean Squared Error after 400 epochs: 1.0680036202264156
Mean Squared Error after 500 epochs: 0.8609191543258855
Mean Squared Error after 600 epochs: 0.8043043436911523
Mean Squared Error after 700 epochs: 0.7885900306638032
Mean Squared Error after 800 epochs: 0.7839951845567169
Mean Squared Error after 900 epochs: 0.7824252690486773
R2 Score: 0.9202568099499092
```

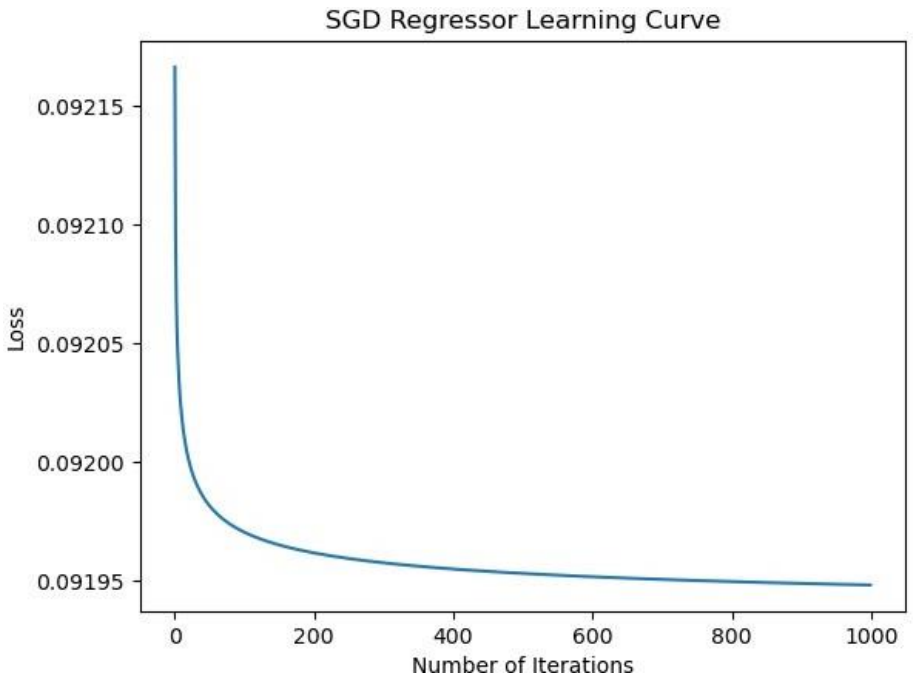


SDG Regressor

In [63]:

```
from sklearn.linear_model import SGDRegressor
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score

# Feature scaling (important for SGD)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Create an SGDRegressor instance
regressor = SGDRegressor(max_iter=1000, tol=1e-3, random_state=42)

# Manually capture the loss values during training
loss_values = []
for epoch in range(1000):  # Adjust the number of epochs as needed
    regressor.partial_fit(X_train_scaled, y_train)
    y_pred = regressor.predict(X_train_scaled)
    mse = mean_squared_error(y_train, y_pred)
    loss_values.append(mse)

# Predict on the test set
y_pred = regressor.predict(X_test_scaled)

# Calculate and print the Mean Squared Error (MSE) as a measure of accuracy
mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error:", mse)

# Calculate R-squared score
score = r2_score(y_test, y_pred)
print("R-squared Score:", score)

# Plot the learning graph
plt.plot(loss_values)
plt.title("SGD Regressor Learning Curve")
plt.xlabel("Number of Iterations")
plt.ylabel("Loss")
plt.show()
```

```
Mean Squared Error: 0.09042501884190475
R-squared Score: 0.6173317848725053
```



In [ ]: