

VISVESVARAYA TECHNOLOGICAL UNIVERSITY
“JnanaSangama”, Belgaum -590014, Karnataka.



LAB REPORT
ON
MACHINE LEARNING

Submitted by:

NAMYA R D(1BM21CS111)

in partial fulfillment for the award of the degree of

BACHELOR OF ENGINEERING
in
COMPUTER SCIENCE AND ENGINEERING



B.M.S. COLLEGE OF ENGINEERING
(Autonomous Institution under VTU)
BENGALURU-560019
March 2024-June 2024



**B. M. S. College of Engineering,
Bull Temple Road, Bangalore 560019**
(Affiliated To Visvesvaraya Technological University, Belgaum)
Department of Computer Science and Engineering

CERTIFICATE

This is to certify that the Lab work entitled “**MACHINE LEARNING**” carried out by **NAMYA R D(1BM21CS111)**, who is bonafide student of **B. M. S. College of Engineering**. It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum during the year 2023-24. The Lab report has been approved as it satisfies the academic requirements in respect of Machine Learning Lab - **(22CS3PCMAL)** work prescribed for the said degree.

Sonika Sharma D

Assistant Professor
Department of CSE
BMSCE, Bengaluru

Dr. Jyothi S Nayak

Professor and Head
Department of CSE
BMSCE, Bengaluru

INDEX

Sl. No.	Experiment Title	Page No.
1	Write a python program to import and export data using Pandas library functions	3-5
2	Demonstrate various data pre-processing techniques for a given dataset	6-11
3	Use an appropriate data set for building the decision tree (ID3) and apply this knowledge to classify a new sample.	12-18
4	Implement Linear and Multi-Linear Regression algorithm using appropriate dataset	19-25
5	Build Logistic Regression Model for a given dataset	26-30
6	Build KNN Classification model for a given dataset.	31-34
7	Build Support vector machine model for a given dataset	35-38
8	Build Artificial Neural Network model with back propagation on a given dataset	39-41
9	a) Implement Random forest ensemble method on a given dataset. b) Implement Boosting ensemble method on a given dataset.	42-51
10	Build k-Means algorithm to cluster a set of data stored in a .CSV file.	52-55
11	Implement Dimensionality reduction using Principle Component Analysis (PCA) method.	56-59

PROGRAM 1

Date:05-04-2024

Write a python program to import and export data using Pandas library functions

05/04/24

Program -1

Write a python program to import and export data using Pandas library functions

IMPORT:

```
import pandas as pd  
airbnb_data = pd.read_csv("listings.csv")  
airbnb_data.head()
```

EXPORT:

```
airbnb_data.to_csv("exported-listings.csv")
```

READING DATA FROM URL:

```
url = "https://archive.ics.uci.edu/ml/machine-learning-  
databases/iris/iris.data"  
colnames = ["sepal-length-in-cm", "sepal-width-  
in-cm", "petal-length-in-cm",  
"petal-width-in-cm", "class"]  
iris_data = pd.read_csv(url, names=colnames)  
iris_data.head()
```

Import:

```
import pandas as pd
# Read the CSV file
airbnb_data = pd.read_csv("listings.csv")
# View the first 5 rows
airbnb_data.head()
```

[1]:		id	name	host_id	host_name	neighbourhood_group	neighbourhood	latitude	longitude	room_type	price	minimum_nights	number_of_reviews	last_review
0	329172	Hillside designer home, 10 min. dwntrn	1680871	Janet		Nan	78746	30.30085	-97.80794	Entire home/apt	495	3	7	2022-
1	329306	Urban Homestead, 5 minutes to downtown	880571	Angel		Nan	78702	30.27232	-97.72579	Private room	63	2	570	2022-
2	331549	One Room with Private Bathroom	1690383	Sandra		Nan	78725	30.23911	-97.58625	Private room	100	2	0	
3	333815	Solar Sanctuary - Austin Room	372962	Kim		Nan	78704	30.25381	-97.75262	Private room	102	2	164	2022-
4	333442	Rare Secluded 1940s Estate	1698318	Virginia		Nan	78703	30.31267	-97.76641	Entire home/apt	286	3	163	2022-

Export:

```
airbnb_data.to_csv("exported_listings.csv")
```

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
1																			
2	0	329172	Hillside designer home, 10 min. dwntrn	1680871	Janet		78746	30.30085	-97.80794	Entire home	495	3	7	0.05	1	363	1		
3	1	329306	Urban Homestead, 5 minutes to downtown	880571	Angel		Nan	78702	30.27232	Private room	63	2	570	4.36	5	55	45		
4	2	331549	One Room with Private Bathroom	1690383	Sandra		Nan	78725	30.23911	Entire home	100	2	0		1	0	0		
5	3	333815	Solar Sanctuary - Austin Room	372962	Kim		Nan	78704	30.25381	Entire home	102	2	164	1.26	1	36	18		
6	4	333442	Rare Secluded 1940s Estate	1698318	Virginia		Nan	78703	30.31267	Entire home	286	3	163	1.32	1	271	15		
7	5	335815	Solar Sanc	1690383	Sandra		Nan	78701	30.25381	Entire home	300	28	43	0.33	1	90	0		
8	6	333442	Rare Secluded 1940s Estate	1698318	Virginia		Nan	78702	30.27232	Entire home	250	1	2	0.02	1	0	0		
9	7	337125	1800 Sqft, Great SKS!	1680871	Stephen		Nan	78741	30.24481	Entire home	80	32	33	0.25	1	83	0		
10	8	340039	Greenbell, 172572 Shanti	1698318	Carolyn		Nan	78759	30.42035	Entire home	400	1	1	0.09	1	0	1		
11	9	335945	The Sherri	1555683	Patrick		Nan	78741	30.42026	Entire home	221	3	34	0.26	1	27	2		
12	10	340630	2 Bedroom	531267	Shelley		Nan	78757	30.33061	Entire home	50	30	23	0.18	3	25	3		
13	11	341596	South Aus	1733004	Julie		Nan	78706	30.24623	Entire home	120	2	90	0.69	1	122	15		
14	12	341382	SXSW i	1205884	Luke And Rachel		Nan	78702	30.26543	Entire home	750	4	124	0.95	1	0	0		
15	13	342243	Sparcious f	1736662	Teresa		Nan	78759	30.41633	Entire home	45	31	3	0.06	1	317	0		
16	14	342039	Garden Gt	1735494	Steph And Joey		Nan	78702	30.25158	Entire home	116	1	342	2.62	2	120	30		
17	15	343889	Close-In C	1744639	Darcy		Nan	78702	30.72793	Entire home	47	30	32	0.24	1	214	4		
18	16	343462	Charming	1742984	Rachel		Nan	78721	30.26895	Entire home	99	1	255	1.95	1	143	12		
19	17	345473	Quaint	1751224	Whitney		Nan	78706	30.24619	Entire home	308	2	164	1.38	1	86	44		
20	18	345221	Austin His	1644218	Cecily		Nan	78701	30.27247	Entire home	299	3	8	0.06	2	0	0		
21	19	5456	Walk to 6t	8028	Sylvia		Nan	78702	30.26057	Entire home	96	2	623	3.71	1	318	44		
22	20	347572	East Austi	1761506	Savanna		Nan	78702	30.26448	Entire home	125	2	29	0.22	1	0	0		
23	21	5769	NW Austi	8186	Elizabeth		Nan	78729	30.45697	Entire home	41	1	273	1.77	1	14	9		
24	22	345536	Artist's Hr	1752493	Gigi		Nan	78704	30.2466	Entire home	156	3	284	2.19	2	319	29		
25	23	347935	for formul	1763742	Amber		Nan	78704	30.25152	Entire home	643	5	1	0.01	1	0	0		

Reading data from URL:

```
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"
```

```
# Define the column names
```

```
col_names = ["sepal_length_in_cm",
             "sepal_width_in_cm",
             "petal_length_in_cm",
             "petal_width_in_cm",
             "class"]
```

```
# Read data from URL
```

```
iris_data = pd.read_csv(url, names=col_names)
```

```
iris_data.head()
```

	sepal_length_in_cm	sepal_width_in_cm	petal_length_in_cm	petal_width_in_cm	class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

PROGRAM 2

Date:05-04-2024

Demonstrate various data pre-processing techniques for a given dataset

Code and Output

2. Importing and Exploration of the dataset

```
In [2]: # Loading the data and setting the unique client_id as the index::  
df = pd.read_csv('/content/loans.csv', index_col = 'client_id')
```

```
In [3]: # # showing the first 5 rows of the dataset:  
df.head()
```

```
Out[3]:   loan_type  loan_amount  repaid  loan_id  loan_start  loan_end  rate  
client_id  
46109    home        13672      0  10243  2002-04-16  2003-12-20  2.15  
46109    credit       9794      0  10984  2003-10-21  2005-07-17  1.25  
46109    home        12734      1  10990  2006-02-01  2007-07-05  0.68  
46109    cash         12518      1  10596  2010-12-08  2013-05-05  1.24  
46109    credit       14049      1  11415  2010-07-07  2012-05-21  3.13
```

```
In [4]: # To check the Dimensions of the dataset:  
df.shape
```

```
Out[4]: (443, 7)
```

```
In [5]: # Checking the info of the data:  
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>  
Index: 443 entries, 46109 to 26945  
Data columns (total 7 columns):  
 #   Column      Non-Null Count  Dtype     
---  --          --          --  
 0   loan_type    443 non-null   object    
 1   loan_amount  443 non-null   int64    
 2   repaid       443 non-null   int64    
 3   loan_id      443 non-null   int64    
 4   loan_start   443 non-null   object    
 5   loan_end     443 non-null   object    
 6   rate         443 non-null   float64  
dtypes: float64(1), int64(3), object(3)  
memory usage: 27.7+ KB
```

3. Checking the datatypes of the columns

```
In [6]: df.dtypes
```

```
Out[6]: loan_type      object
loan_amount     int64
repaid         int64
loan_id        int64
loan_start     object
loan_end       object
rate           float64
dtype: object
```

4. Converting the data types of columns

- loan_id to object
- repaid to category dtype
- loan_start and loan_end to date type

```
In [7]: # Loan_id:
```

```
df['loan_id'] = df['loan_id'].astype('object')

# repaid:

df['repaid'] = df['repaid'].astype('category')
```

```
In [8]: # Loan_start:
```

```
df['loan_start'] = pd.to_datetime(df['loan_start'], format = '%Y-%m-%d')

# Loan_end:

df['loan_end'] = pd.to_datetime(df['loan_end'], format = '%Y-%m-%d')
```

Checking the datatypes again:

```
In [9]: df.dtypes
```

```
Out[9]: loan_type      object
loan_amount     int64
repaid         category
loan_id        object
loan_start     datetime64[ns]
loan_end       datetime64[ns]
rate           float64
dtype: object
```

5. Summary Statistics of the data

```
In [10]: # Summary Statistics for Numerical data:
df.describe()
```

	loan_amount	loan_start	loan_end	rate
count	443.000000	443	443	443.000000
mean	7982.311512	2007-08-02 12:56:53.092550912	2009-08-23 11:35:37.246049536	3.217156
min	559.000000	2000-01-26 00:00:00	2001-08-02 00:00:00	0.010000
25%	4232.500000	2003-10-19 00:00:00	2005-09-12 12:00:00	1.220000
50%	8320.000000	2007-03-10 00:00:00	2009-03-19 00:00:00	2.780000
75%	11739.000000	2011-07-31 00:00:00	2013-09-11 12:00:00	4.750000
max	14971.000000	2014-11-11 00:00:00	2017-05-07 00:00:00	12.620000
std	4172.891992	NaN	NaN	2.397168

```
In [11]: # Summary Statistics for Categorical data:  
df.describe(exclude=[np.number])
```

```
Out[11]:
```

	loan_type	repaid	loan_id	loan_start	loan_end
count	443	443.0	443.0	443	443
unique	4	2.0	443.0	NaN	NaN
top	home	1.0	10243.0	NaN	NaN
freq	121	237.0	1.0	NaN	NaN
mean	NaN	NaN	NaN	2007-08-02 12:56:53.092550912	2009-08-23 11:35:37.246049536
min	NaN	NaN	NaN	2000-01-26 00:00:00	2001-08-02 00:00:00
25%	NaN	NaN	NaN	2003-10-19 00:00:00	2005-09-12 12:00:00
50%	NaN	NaN	NaN	2007-03-10 00:00:00	2009-03-19 00:00:00
75%	NaN	NaN	NaN	2011-07-31 00:00:00	2013-09-11 12:00:00
max	NaN	NaN	NaN	2014-11-11 00:00:00	2017-05-07 00:00:00

6. Missing Values

```
In [12]: # use isnull().sum() to check for missing values  
df.isnull().sum()
```

```
Out[12]:
```

loan_type	0
loan_amount	0
repaid	0
loan_id	0
loan_start	0
loan_end	0
rate	0
dtype: int64	

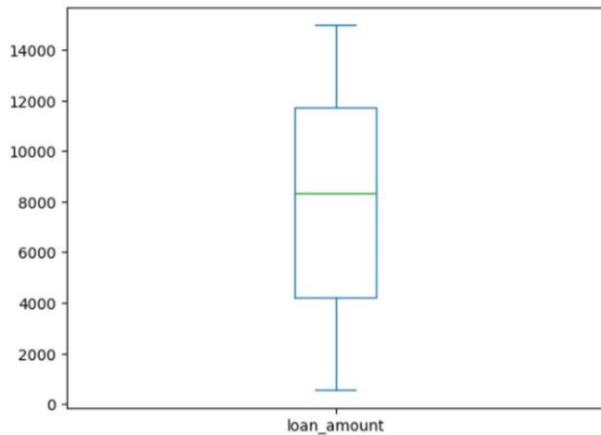
There are no missing values in the data.

Sk-learn library has an in-built function called Iterative Imputer to impute the missing values. Its sklearn documentation: <https://scikit-learn.org/stable/modules/generated/sklearn.impute.IterativeImputer.html>

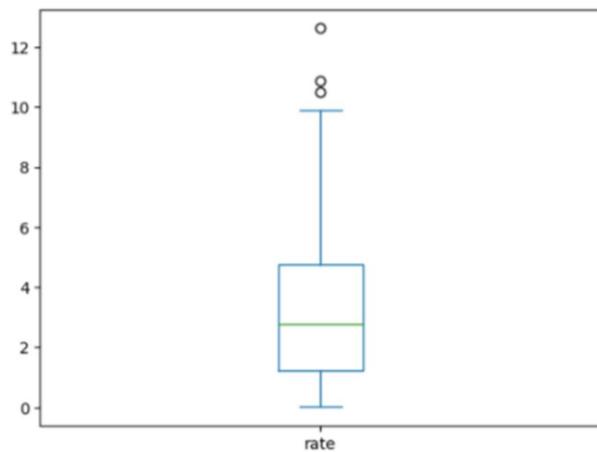
7. Outliers Treatment

To check for the presence of outliers, we plot Boxplot.

```
In [13]: # For loan_amount  
df['loan_amount'].plot(kind='box')  
plt.show()
```



```
In [14]: # For rate  
df['rate'].plot(kind='box')  
plt.show()
```



We can see that there are no outliers in the loan_amount column and some outliers are present in the rate column. To treat for outliers can either cap the values or transform the data. Shall demonstrate both the approaches here.

8. Transformation

8a. SQRT transformation

```
In [15]: df['SQRT_RATE'] = df['rate']**0.5
```

```
In [16]: df['sqrt_rate'] = np.sqrt(df['rate'])
```

```
In [17]: df.head()
```

```
Out[17]:   loan_type  loan_amount  repaid  loan_id  loan_start  loan_end  rate  SQRT_RATE  sqrt_rate  
client_id  
46109    home       13672      0  10243  2002-04-16  2003-12-20  2.15  1.466288  1.466288  
46109   credit       9794      0  10984  2003-10-21  2005-07-17  1.25  1.118034  1.118034  
46109    home       12734      1  10990  2006-02-01  2007-07-05  0.68  0.824621  0.824621  
46109   cash        12518      1  10596  2010-12-08  2013-05-05  1.24  1.113553  1.113553  
46109   credit       14049      1  11415  2010-07-07  2012-05-21  3.13  1.769181  1.769181
```

```
In [18]: #checking the skewness, kurtosis between the original and transformed data:
print("The skewness of the original data is {}".format(df.rate.skew()))
print('The skewness of the SQRT transformed data is {}'.format(df.SQRT_RATE.skew()))

print('')

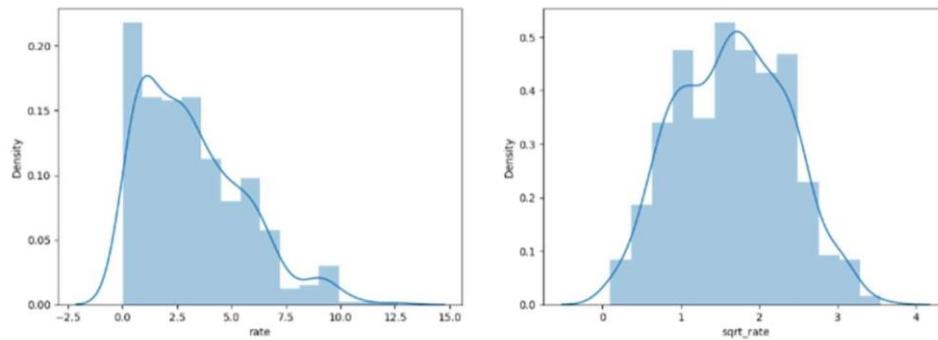
print("The kurtosis of the original data is {}".format(df.rate.kurt()))
print("The kurtosis of the SQRT transformed data is {}".format(df.SQRT_RATE.kurt()))
```

The skewness of the original data is 0.884204614329943
The skewness of the SQRT transformed data is 0.04964154055528862
The kurtosis of the original data is 0.42437165143736433
The kurtosis of the SQRT transformed data is -0.6318437642052039

```
In [19]: # plotting the distribution

fig, axes = plt.subplots(1,2, figsize=(15,5))
sns.distplot(df['rate'], ax=axes[0])
sns.distplot(df['sqrt_rate'], ax=axes[1])

plt.show()
```



Result:

The Rate column was right skewed earlier. The skewness and kurtosis has reduced significantly. The transformed SQRT rate, on the right graph resembles normal distribution now.

8b. Log Transformation

```
In [20]: df['Log Rate'] = np.log(df['rate'])
```

```
In [21]: df.head()
```

```
Out[21]:   loan_type  loan_amount  repaid  loan_id  loan_start  loan_end  rate  SQRT_RATE  sqrt_rate  Log Rate
client_id
46109    home        13672      0  10243  2002-04-16  2003-12-20  2.15  1.466288  1.466288  0.765468
46109    credit       9794      0  10984  2003-10-21  2005-07-17  1.25  1.118034  1.118034  0.223144
46109    home        12734      1  10990  2006-02-01  2007-07-05  0.68  0.824621  0.824621  -0.385662
46109    cash         12518      1  10596  2010-12-08  2013-05-05  1.24  1.113553  1.113553  0.215111
46109    credit       14049      1  11415  2010-07-07  2012-05-21  3.13  1.769181  1.769181  1.141033
```

```
In [22]: print("The skewness of the original data is {}".format(df.rate.skew()))
print('The skewness of the SQRT transformed data is {}'.format(df.SQRT_RATE.skew()))
print("The skewness of the LOG transformed data is {}".format(df['Log Rate'].skew()))

print('')

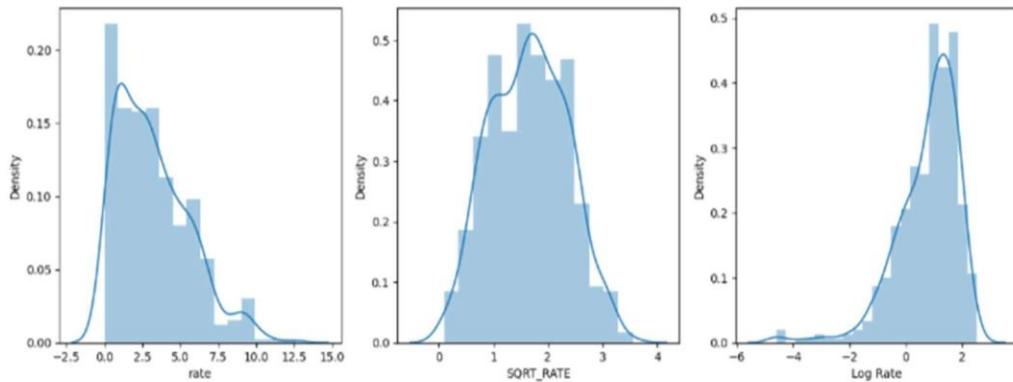
print("The kurtosis of the original data is {}".format(df.rate.kurt()))
print("The kurtosis of the SQRT transformed data is {}".format(df.SQRT_RATE.kurt()))
print("The kurtosis of the LOG transformed data is {}".format(df['Log Rate'].kurt()))
```

The skewness of the original data is 0.884204614329943
The skewness of the SQRT transformed data is 0.04964154055528862
The skewness of the LOG transformed data is -1.5943217626331552
The kurtosis of the original data is 0.42437165143736433
The kurtosis of the SQRT transformed data is -0.6318437642052039
The kurtosis of the LOG transformed data is 4.157026150198228

```
In [23]: # plot the graph:
fig, axes = plt.subplots(1,3,figsize=(15,5))

sns.distplot(df['rate'], ax=axes[0])
sns.distplot(df['SQRT_RATE'], ax=axes[1])
sns.distplot(df['Log Rate'], ax=axes[2])

plt.show()
```



Inference:

Log Transformation made the rate left skewed and more peaked.

However, Log transformation is more closer to 0 and hence is more normal. Though it heavily manipulates the data.

In our case, square root transformation is more suitable.

```
In [24]: ## Using Lambda function :
df['LOG_Rate'] = df['rate'].apply(lambda x:np.log(x))
```

```
In [25]: df.head()
```

```
Out[25]:   loan_type  loan_amount  repaid  loan_id  loan_start  loan_end  rate  SQRT_RATE  sqrt_rate  Log Rate  LOG_Rate
client_id
46109    home        13672      0  10243  2002-04-16  2003-12-20  2.15  1.466288  1.466288  0.765468  0.765468
46109    credit       9794      0  10984  2003-10-21  2005-07-17  1.25  1.118034  1.118034  0.223144  0.223144
46109    home        12734      1  10990  2006-02-01  2007-07-05  0.68  0.824621  0.824621 -0.385662 -0.385662
46109    cash         12518      1  10596  2010-12-08  2013-05-05  1.24  1.113553  1.113553  0.215111  0.215111
46109    credit       14049      1  11415  2010-07-07  2012-05-21  3.13  1.769181  1.769181  1.141033  1.141033
```

PROGRAM 3

Date: 12-04-2024

Use an appropriate data set for building the decision tree (ID3) and apply this knowledge to classify a new sample

Algorithm:

12/04/24

Program - 2 Decision Tree ID3

Algorithm:

ID3(Examples, Target-attribute, Attributes)

- Create a Root node for the tree.
- Return If all Examples are positive,
Return the single-node tree Root, with
label = +
- If all Examples are negative, Return
the single-node tree Root, with label = -
- If attributes is empty, Return the single-
node tree Root, with label = most common
value of Target-attribute in Examples
- Otherwise Begin :
 - A \leftarrow the attribute from Attributes that
best* classifies Examples.
 - The decision attribute for Root \leftarrow A.
 - For each possible value, v_i , of A,
 - Add a new tree branch below Root,
corresponding to the test $A = v_i$.
 - Let Examples $_{v_i}$ be the subset of
Examples that have values v_i for A.
 - If Examples $_{v_i}$ is empty.
 - Then below this new branch add
a leaf node with label = most
common value of Target-attribute
 - Else below this new branch add
the subtree ID3.
 - End
 - Return Root

Code:

Importing Database

```
In [2]: # Importing the required Libraries
import pandas as pd
import numpy as np
import math

# Reading the dataset (Tennis-dataset)
data = pd.read_csv('/content/PlayTennis.csv')

In [ ]: from google.colab import drive
drive.mount('/content/drive')

In [3]: def highlight(cell_value):
    ...
    Highlight yes / no values in the dataframe
    ...
    color_1 = 'background-color: pink;'
    color_2 = 'background-color: lightgreen;'

    if cell_value == 'no':
        return color_1
    elif cell_value == 'yes':
        return color_2

data.style.applymap(highlight)\n    .set_properties(subset=data.columns, **{'width': '100px'})\n    .set_table_styles([{'selector': 'th', 'props': [('background-color', 'lightgray'), ('border', '1px solid gray'),
                                                    ('font-weight', 'bold')]}],\n                    {'selector': 'tr:hover', 'props': [('background-color', 'white'), ('border', '1.5px solid black')]}])
```

Out[3]:

	outlook	temp	humidity	windy	play
0	sunny	hot	high	False	no
1	sunny	hot	high	True	no
2	overcast	hot	high	False	yes
3	rainy	mild	high	False	yes
4	rainy	cool	normal	False	yes
5	rainy	cool	normal	True	no
6	overcast	cool	normal	True	yes
7	sunny	mild	high	False	no
8	sunny	cool	normal	False	yes
9	rainy	mild	normal	False	yes
10	sunny	mild	normal	True	yes
11	overcast	mild	high	True	yes
12	overcast	hot	normal	False	yes
13	rainy	mild	high	True	no

Entropy of the dataset

```
In [4]:  
def find_entropy(data):  
    """  
    Returns the entropy of the class or features  
    formula: - Σ P(X)logP(X)  
    """  
    entropy = 0  
    for i in range(data.nunique()):  
        x = data.value_counts()[i]/data.shape[0]  
        entropy += (-x * math.log(x,2))  
    return round(entropy,3)  
  
def information_gain(data, data_):  
    """  
    Returns the information gain of the features  
    """  
    info = 0  
    for i in range(data_.nunique()):  
        df = data[data_ == data_.unique()[i]]  
        w_avg = df.shape[0]/data.shape[0]  
        entropy = find_entropy(df.play)  
        x = w_avg * entropy  
        info += x  
    ig = find_entropy(data.play) - info  
    return round(ig, 3)  
  
def entropy_and_infogain(datax, feature):  
    """  
    Grouping features with the same class and computing their  
    entropy and information gain for splitting  
    """  
    for i in range(data[feature].nunique()):  
        df = datax[datax[feature]==data[feature].unique()[i]]  
        if df.shape[0] < 1:  
            continue  
  
        display(df[[feature, 'play']].style.applymap(highlight)\  
            .set_properties(subset=[feature, 'play'], **{'width': '80px'})\ \  
            .set_table_styles([{'selector': 'th', 'props': [('background-color', 'lightgray'),  
                                              ('border', '1px solid gray'),  
                                              ('font-weight', 'bold')]}],  
                           {'selector': 'td', 'props': [('border', '1px solid gray')]},  
                           {'selector': 'tr:hover', 'props': [('background-color', 'white'),  
                                              ('border', '1.5px solid black')]}])  
  
        print(f'Entropy of {feature} - {data[feature].unique()[i]} = {find_entropy(df.play)}')  
        print(f'Information Gain for {feature} = {information_gain(datax, datax[feature])}')
```

```
In [5]:  
print(f'Entropy of the entire dataset: {find_entropy(data.play)})'
```

Entropy of the entire dataset: 0.94

Entropy and Information Gain of temperature

```
In [6]: entropy_and_infogain(data, 'temp')
```

```
temp play
0 hot no
1 hot no
2 hot yes
12 hot yes
Entropy of temp - hot = 1.0
temp play
3 mild yes
7 mild no
9 mild yes
10 mild yes
11 mild yes
13 mild no
Entropy of temp - mild = 0.918
temp play
4 cool yes
5 cool no
6 cool yes
8 cool yes
Entropy of temp - cool = 0.811
Information Gain for temp = 0.029
```

Entropy and Information Gain of humidity

```
In [7]: entropy_and_infogain(data, 'humidity')
```

```
humidity play
0 high no
1 high no
2 high yes
3 high yes
7 high no
11 high yes
13 high no
Entropy of humidity - high = 0.985
humidity play
4 normal yes
5 normal no
6 normal yes
8 normal yes
9 normal yes
10 normal yes
12 normal yes
Entropy of humidity - normal = 0.592
Information Gain for humidity = 0.151
```

Entropy and Information Gain of windy

```
In [8]: entropy_and_infogain(data, 'windy')
```

```
windy play
```

```
0 False no
2 False yes
3 False yes
4 False yes
7 False no
8 False yes
9 False yes
12 False yes
```

```
Entropy of windy - False = 0.811
```

```
windy play
```

```
1 True no
5 True no
6 True yes
10 True yes
11 True yes
13 True no
```

```
Entropy of windy - True = 1.0
```

```
Information Gain for windy = 0.048
```

Rainy Outlook

Rainy -outlook

```
In [9]: rainy = data[data['outlook'] == 'rainy']
rainy.style.applymap(highlight)
.set_properties(subset=data.columns, **{'width': '100px'})
.set_table_styles([{'selector': 'th', 'props': [('background-color', 'lightgray'), ('border', '1px solid gray'),
                                              ('font-weight', 'bold')]}],
                  {'selector': 'tr:hover', 'props': [('background-color', 'white'), ('border', '1.5px solid black')]})
```

```
Out[9]:   outlook  temp  humidity  windy  play
          3    rainy   mild      high  False   yes
          4    rainy   cool     normal  False   yes
          5    rainy   cool     normal   True    no
          9    rainy   mild     normal  False   yes
         13    rainy   mild      high   True    no
```

```
In [10]: print(f'Entropy of the Rainy dataset: {find_entropy(rainy.play)}')

Entropy of the Rainy dataset: 0.971
```

```
In [11]: entropy_and_infogain(rainy, 'temp')
```

temp	play	
3	mild	yes
9	mild	yes
13	mild	no

Entropy of temp - mild = 0.918

temp	play	
4	cool	yes
5	cool	no

Entropy of temp - cool = 1.0
Information Gain for temp = 0.02

```
In [12]: entropy_and_infogain(rainy, 'humidity')
```

humidity	play	
3	high	yes
13	high	no

Entropy of humidity - high = 1.0

humidity	play	
4	normal	yes
5	normal	no
9	normal	yes

Entropy of humidity - normal = 0.918
Information Gain for humidity = 0.02

```
In [13]: entropy_and_infogain(rainy, 'windy')
```

windy	play	
3	False	yes
4	False	yes
9	False	yes

Entropy of windy - False = 0.0

windy	play	
5	True	no
13	True	no

Entropy of windy - True = 0.0
Information Gain for windy = 0.971

wind has highest information gain

Output

Output:

Entropy of the dataset : 0.9331

Pregnancies - Entropy : 3.482 , IG : 0.062

Glucose - Entropy : 6.751 , IG : 0.304

BloodPressure - Entropy : 4.792 , IG : 0.059

SkinThickness - Entropy : 4.586 , IG : 0.082

Insulin - Entropy : 4.682 , IG : 0.277

BMI - Entropy : 7.594 , IG : 0.344

DiabetesPedigreeFunction - Entropy : 8.829 , IG : 0.65

Age - Entropy : 5.029 , IG : 0.141

PROGRAM 4

Date: 19-04-2024

Implement Linear and Multi-Linear Regression algorithm using appropriate dataset

LINEAR REGRESSION:

Algorithm

03/05/2024

Program - 4

Implement linear and multi-linear Regression algorithm.

Linear Regression :

```
function linear_regression (x, y, learning_rate, num_iterations)
    Initialize random values for slope (m)
    & intercept (b)
    for i = 1 to num_iterations :
        predictions = m * x + b
        errors = predictions - y
        loss = mean_squared_error(errors)
        gradient_m = (2/N) * sum(errors * x)
        gradient_m = (2N) * sum(errors)
        m = m - learning_rate * gradient_m
        b = b - learning_rate * gradient_b
    Return m, b

function mean_squared_error (errors) :
    squared_errors = errors^2
    mse = sum(squared_errors) / sum(errors)
    return mse
```

Code

Importing Dataset

```
In [1]: import pandas as pd  
import numpy as np  
import matplotlib.pyplot as plt
```

```
In [18]: df=pd.read_csv("/Salary_Data.csv")  
df
```

```
Out[18]:   YearsExperience    Salary  
0           1.1    39343.0  
1           1.3    46205.0  
2           1.5    37731.0  
3           2.0    43525.0  
4           2.2    39891.0  
5           2.9    56642.0  
6           3.0    60150.0  
7           3.2    54445.0  
8           3.2    64445.0  
9           3.7    57189.0  
10          3.9    63218.0  
11          4.0    55794.0  
12          4.0    56957.0  
13          4.1    57081.0  
14          4.5    61111.0  
15          4.9    67938.0  
16          5.1    66029.0  
17          5.3    83088.0  
18          5.9    81363.0  
19          6.0    93940.0  
20          6.8    91738.0  
21          7.1    98273.0  
22          7.9    101302.0  
23          8.2    113812.0  
24          8.7    109431.0  
25          9.0    105582.0  
26          9.5    116969.0  
27          9.6    112635.0  
28         10.3    122391.0  
29         10.5    121872.0
```

Slope and Intercept calculation

```
In [19]: def linear(X, b0, b1):
    return [b0+b1*x for x in X]
```

```
In [20]: # b0 - Intercept
def intercept(X, Y, b1):
    X_ = np.mean(X)
    Y_ = np.mean(Y)

    return Y_-b1*X_
```

```
In [21]: # b1 - Slope
def slope(X, Y):
    X_ = np.mean(X)
    Y_ = np.mean(Y)

    rise = sum([(x-X_) * (y-Y_) for x,y in zip(X,Y)])
    run = sum([(x-X_)**2 for x,y in zip(X,Y)])

    return rise / run
```

Predicted Values Graph

```
In [23]: plt.figure(figsize=(8,5))
plt.title("YearsExperience vs Salary")
plt.scatter(predictor, target, color = "#247ba0")
plt.xlabel("YearsExperience")
plt.ylabel("Salary")
plt.show()
```



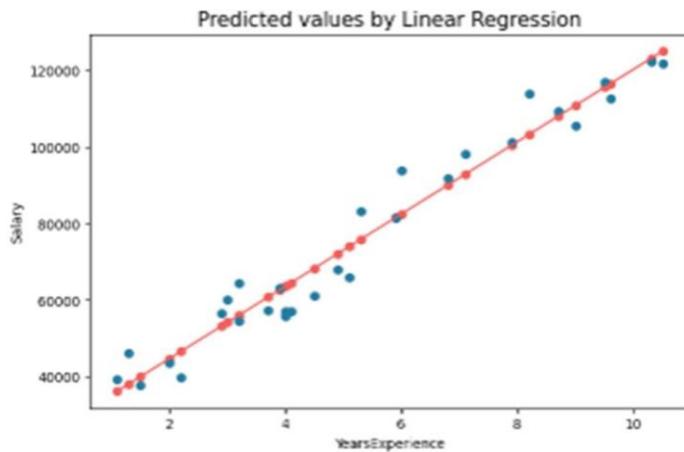
```
In [24]: b1 = slope(predictor, target)
b0 = intercept(predictor, target, b1)
predicted = linear(predictor, b0, b1)
```

```
In [25]: plt.figure(figsize = (8, 5))
plt.plot(predictor, predicted, color = "#ff25f5c")
plt.scatter(predictor, predicted, color = "#ff25f5c") #red Line is predicted data
plt.title('Predicted values by Linear Regression', fontsize = 15)
plt.xlabel("YearsExperience")
plt.ylabel("Salary")
plt.scatter(predictor, target, color = "#247ba0")
plt.show()
```

Output

```
In [26]:  
print("Coefficients:\n=====")  
print("b0 : ", b0)  
print("b1 : ", b1)
```

```
Coefficients:  
=====  
b0 : 25792.20019866869  
b1 : 9449.962321455077
```



MULTIPLE LINEAR REGRESSION:

Algorithm

Date _____
Page _____

Multi-linear Regression

```
function initialize_parameters():
    randomly initialize  $\beta_0, \beta_1, \dots$ 
```

```
function hypothesis_function( $X, \beta$ ):
     $h = \beta_0 + \beta_1 * X[1] + \beta_2 * X[2] + \dots + \beta_m * X[m]$ 
    return  $h$ .
```

```
function cost_function( $X, y, \beta$ ):
    n = length( $X$ )
    total_error = 0
    for i = 1 to n:
         $h = \text{hypothesis\_function}(X[i], \beta)$ 
        total_error += ( $h - y[i]$ )^2
    cost = (1/(2*n)) * total_error
    return cost.
```

```
function gradient_descent( $X, y, \beta, \alpha, \text{iterations}, \text{threshold}$ ):
    n = length( $X$ )
    for i=0 = 1 to iterations:
        error_sum = 0
        for i = 1 to n:
             $h = \text{hypothesis\_function}(X[i], \beta)$ 
            error =  $h - y[i]$ 
            error_sum += error
        error_sum /= n
        for j = 0 to m:
             $\beta[j] = \beta[j] - \alpha * (1/n) * error * X[i][j]$ 
    cost = cost_function( $X, y, \beta$ )
    if cost < threshold: break
    return  $\beta$ .
```

Code

```
In [19]: house = pd.read_csv('https://github.com/YBIFoundation/Dataset/raw/main/Boston.csv')
```

```
In [20]: house.head()
```

```
Out[20]:
```

	CRIM	ZN	INDUS	CHAS	NX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296.0	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242.0	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242.0	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222.0	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222.0	18.7	396.90	5.33	36.2

```
In [21]: house.describe()
```

```
Out[21]:
```

	CRIM	ZN	INDUS	CHAS	NX	RM	AGE	DIS	RAD	TAX	PTRAT
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000
mean	3.613524	11.363636	11.136779	0.069170	0.554695	6.284634	68.574901	3.795043	9.549407	408.237154	18.4555
std	8.601545	23.322453	6.860353	0.253994	0.115878	0.702617	28.148861	2.105710	8.707259	168.537116	2.1649
min	0.006320	0.000000	0.460000	0.000000	0.385000	3.561000	2.900000	1.129600	1.000000	187.000000	12.6000
25%	0.082045	0.000000	5.190000	0.000000	0.449000	5.885500	45.025000	2.100175	4.000000	279.000000	17.4000
50%	0.256510	0.000000	9.690000	0.000000	0.538000	6.208500	77.500000	3.207450	5.000000	330.000000	19.0500
75%	3.677083	12.500000	18.100000	0.000000	0.624000	6.623500	94.075000	5.188425	24.000000	666.000000	20.2000
max	88.976200	100.000000	27.740000	1.000000	0.871000	8.780000	100.000000	12.126500	24.000000	711.000000	22.0000

```
In [22]: house.columns
```

```
Out[22]: Index(['CRIM', 'ZN', 'INDUS', 'CHAS', 'NX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX',  
'PTRATIO', 'B', 'LSTAT', 'MEDV'],  
dtype='object')
```

```
In [23]: y = house['MEDV']
```

```
In [24]: X = house.drop(['MEDV'],axis=1)
```

```
In [25]: from sklearn.model_selection import train_test_split  
X_train, X_test, y_train, y_test = train_test_split(X,y, train_size=0.7, random_state=2529)
```

```
In [26]: X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

```
Out[26]: (354, 13), (152, 13), (354, 1), (152, 1)
```

```
In [25]: from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y, train_size=0.7, random_state=2529)

In [26]: X_train.shape, X_test.shape, y_train.shape, y_test.shape

Out[26]: ((354, 13), (152, 13), (354,), (152,))

In [27]: from sklearn.linear_model import LinearRegression
model = LinearRegression()

In [28]: # Step 6 : train or fit model
model.fit(X_train,y_train)

Out[28]: LinearRegression()
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [29]: model.intercept_

Out[29]: 34.21916368862993

In [30]: model.coef_

Out[30]: array([-1.29e-01,  3.65e-02,  1.54e-02,  2.35e+00, -2.04e+01,  4.41e+00,
       4.61e-03, -1.59e+00,  2.51e-01, -9.60e-03, -9.64e-01,  1.01e-02,
      -5.43e-01])

In [31]: # Step 7 : predict model
y_pred = model.predict(X_test)
```

Output

```
In [32]: y_pred

Out[32]: array([31.72, 22.02, 21.17, 39.78, 20.1 , 22.86, 18.36, 14.79, 22.56,
       21.35, 18.38, 27.97, 29.86, 6.45, 10.68, 26.25, 21.89, 25.23,
       3.62, 36.22, 24.08, 22.94, 14.27, 20.79, 24.23, 16.74, 18.75,
       20.97, 28.51, 20.86, 9.23, 17.07, 22.07, 22.23, 39.26, 26.17,
       42.5 , 19.35, 34.52, 14.07, 13.81, 23.28, 11.79, 9.01, 21.65,
       25.55, 18.17, 16.82, 14.66, 14.86, 33.79, 33.27, 15.49, 24.08,
       27.64, 19.58, 45.02, 20.97, 20.07, 27.67, 34.59, 12.71, 23.66,
       31.66, 28.97, 32.46, 13.93, 35.49, 19.36, 19.6 , 1.44, 24.1 ,
       33.67, 20.62, 26.89, 21.29, 31.95, 29.74, 13.93, 13.82, 19.76,
       21.54, 20.87, 23.63, 28.0 , 23.64, 6.95, 22.2 , -6.82, 16.97,
       16.77, 25.44, 14.95, 3.72, 15.03, 16.91, 21.46, 31.66, 30.72,
       23.73, 22.19, 13.76, 18.47, 18.15, 36.6 , 27.49, 11. , 17.26,
       22.49, 16.53, 29.49, 22.89, 24.68, 20.38, 19.69, 22.55, 27.32,
       24.86, 20.2 , 29.14, 7.43, 5.85, 25.35, 38.73, 23.94, 25.28,
       20.11, 19.75, 25.07, 35.16, 27.32, 27.26, 31.4 , 16.55, 14.3 ,
       23.77, 7.65, 23.35, 21.37, 26.12, 25.32, 13.12, 17.67, 36.2 ,
       20.5 , 27.95, 22.46, 18.15, 31.24, 20.85, 27.36, 30.53])
```

```
In [33]: # Step 8 : model accuracy
from sklearn.metrics import mean_absolute_error, mean_absolute_percentage_error, mean_squared_error
```

```
In [34]: mean_absolute_error(y_test,y_pred)
```

```
Out[34]: 3.155030927602485
```

PROGRAM 5

Date:03-05-2024

Build Logistic Regression Model for a given dataset

Algorithm

03/05/24
Program-5
Build logistic regression model for a given dataset.

```
function logistic-regression(x, y, learning-rate, num-itr):
    Initialize random values for weights (w) & bias (b)
    for i = 1 to num-itr:
        logits = x * w + b
        pred = sigmoid(logits)
        loss = compute-loss(y, pred)
        update weights & bias using gradients
    return w, b

function sigmoid(x):
    return 1 / (1 + exp(-x))

function compute-loss(y-true, y-pred):
    loss = mean(y-true * log(y-pred) + (1-y-true) * log(1-y-pred))
    return loss
```

Code

```
In [2]: import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt
# Input data files are available in the "../input/" directory.
# For example, running this (by clicking run or pressing Shift+Enter) will list the files in the input directory

import os

In [4]: data = pd.read_csv('/content/data.csv')

In [5]: data.drop(['Unnamed: 32','id'], axis=1, inplace=True)
data.diagnosis = [1 if each == "M" else 0 for each in data.diagnosis]
y = data.diagnosis.values
x_data = data.drop(['diagnosis'], axis=1)

In [7]: # Assuming x_data is a numpy array or pandas DataFrame
x = (x_data - np.min(x_data)) / (np.max(x_data) - np.min(x_data))

In [8]: from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.15, random_state=42)

x_train = x_train.T
x_test = x_test.T
y_train = y_train.T
y_test = y_test.T

print("x train: ",x_train.shape)
print("x test: ",x_test.shape)
print("y train: ",y_train.shape)
print("y test: ",y_test.shape)

x train: (30, 483)
x test: (30, 86)
y train: (483,)
y test: (86,)

In [9]: def initialize_weights_and_bias(dimension):
    w = np.full((dimension,1),0.01)
    b = 0.0
    return w, b

In [10]: def sigmoid(z):
    y_head = 1/(1+np.exp(-z))
    return y_head

In [ ]: def forward_backward_propagation(w,b,x_train,y_train):
    # forward propagation
    z = np.dot(w.T,x_train) + b
    y_head = sigmoid(z)
    loss = -y_train*np.log(y_head)-(1-y_train)*np.log(1-y_head)
    cost = (np.sum(loss))/x_train.shape[1]      # x_train.shape[1] is for scaling
    # backward propagation
    derivative_weight = (np.dot(x_train,((y_head-y_train).T)))/x_train.shape[1] # x_train.shape[1] is for scaling
    derivative_bias = np.sum(y_head-y_train)/x_train.shape[1]                      # x_train.shape[1] is for scaling
    gradients = {"derivative_weight": derivative_weight,"derivative_bias": derivative_bias}
    return cost,gradients
```

```
In [1]: def update(w, b, x_train, y_train, learning_rate, number_of_iterarion):
    cost_list = []
    cost_list2 = []
    index = []
    # updating(learning) parameters is number_of_iterarion times
    for i in range(number_of_iterarion):
        # make forward and backward propagation and find cost and gradients
        cost,gradients = forward_backward_propagation(w,b,x_train,y_train)
        cost_list.append(cost)
        w = w - learning_rate * gradients["derivative_weight"]
        b = b - learning_rate * gradients["derivative_bias"]
        if i % 10 == 0:
            cost_list2.append(cost)
            index.append(i)
            print ("Cost after iteration %i: %f" %(i, cost))
    # we update(Learn) parameters weights and bias
    parameters = {"weight": w,"bias": b}
    plt.plot(index,cost_list2)
    plt.xticks(index,rotation='vertical')
    plt.xlabel("Number of Iterarion")
    plt.ylabel("Cost")
    plt.show()
    return parameters, gradients, cost_list
```

```
In [11]: def predict(w,b,x_test):
    # x_test is a input for forward propagation
    z = sigmoid(np.dot(w.T,x_test)+b)
    Y_prediction = np.zeros((1,x_test.shape[1]))
    # if z is bigger than 0.5, our prediction is sign one (y_head=1),
    # if z is smaller than 0.5, our prediction is sign zero (y_head=0),
    for i in range(z.shape[1]):
        if z[0,i]<= 0.5:
            Y_prediction[0,i] = 0
        else:
            Y_prediction[0,i] = 1
    return Y_prediction
```

```
In [18]: def sigmoid(z):
    return 1 / (1 + np.exp(-z))

def initialize_weights_and_bias(dim):
    w = np.zeros((dim, 1))
    b = 0
    return w, b

def compute_cost(w, b, x, y):
    m = x.shape[1]
    A = sigmoid(np.dot(w.T, x) + b)
    cost = -1 / m * np.sum(y * np.log(A) + (1 - y) * np.log(1 - A))
    return cost

def propagate(w, b, x, y):
    m = x.shape[1]
    A = sigmoid(np.dot(w.T, x) + b)
    dw = 1 / m * np.dot(x, (A - y).T)
    db = 1 / m * np.sum(A - y)
    return dw, db
```

```

def logistic_regression(x_train, y_train, x_test, y_test, learning_rate, num_iterations):
    # Initialize
    dimension = x_train.shape[0] # Number of features
    w, b = initialize_weights_and_bias(dimension)
    costs = []

    # Gradient Descent
    for i in range(num_iterations):
        # Forward and Backward Propagation
        dw, db = propagate(w, b, x_train, y_train)

        # Update parameters
        w -= learning_rate * dw
        b -= learning_rate * db

        # Record the costs
        if i % 100 == 0:
            cost = compute_cost(w, b, x_train, y_train)
            costs.append(cost)
            print(f"Cost after iteration {i}: {cost}")

    # Evaluate model
    y_prediction_train = predict(w, b, x_train)
    y_prediction_test = predict(w, b, x_test)

    train_accuracy = 100 - np.mean(np.abs(y_prediction_train - y_train)) * 100
    test_accuracy = 100 - np.mean(np.abs(y_prediction_test - y_test)) * 100

    print("Train accuracy: {}".format(train_accuracy))
    print("Test accuracy: {}".format(test_accuracy))

    return w, b

# Assuming you have defined the predict function
# def predict(w, b, x):
#     ...

# Assuming you have defined x_train, y_train, x_test, y_test, Learning_rate, and num_iterations
logistic_regression(x_train, y_train, x_test, y_test, learning_rate=1, num_iterations=100)

```

Output

```
Cost after iteration 0: 0.6782740160052536
Train accuracy: 80.74534161490683 %
Test accuracy: 81.3953488372093 %

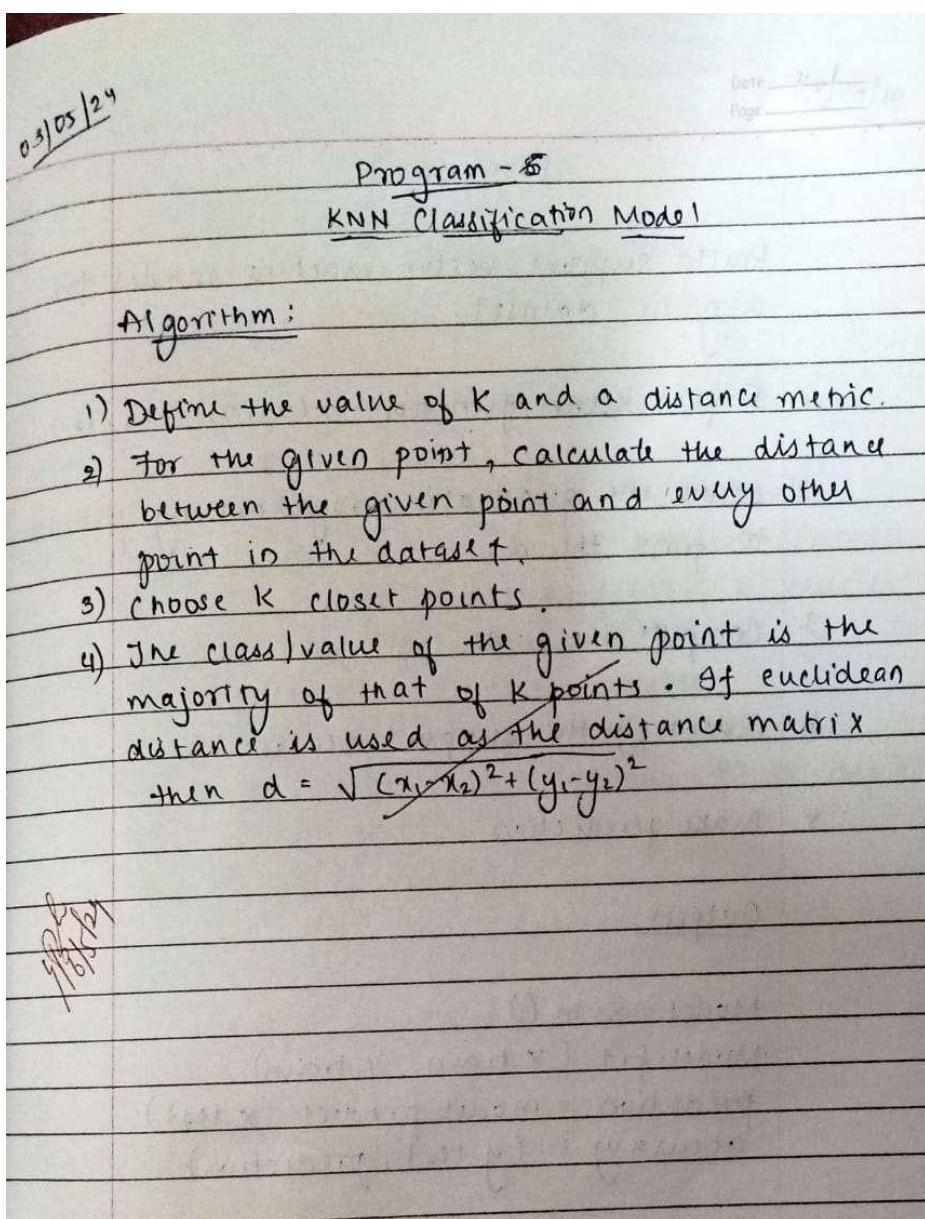
Out[18]: (array([[ 1.77806654e-02],
       [ 1.10160388e-02],
       [ 1.27806976e-01],
       [ 1.95749649e+00],
       [ 1.85931875e-05],
       [ 2.68863405e-04],
       [ 4.89020048e-04],
       [ 2.63106803e-04],
       [ 3.49357933e-05],
       [-2.02145931e-05],
       [ 1.25690784e-03],
       [-3.98285024e-04],
       [ 8.96937014e-03],
       [ 2.02426962e-01],
       [-3.60718647e-06],
       [ 4.19150446e-05],
       [ 6.03411729e-05],
       [ 2.00740406e-05],
       [-6.24803672e-06],
       [ 6.24944780e-07],
       [ 2.79506973e-02],
       [ 1.99326360e-02],
       [ 1.98774929e-01],
       [ 3.39189908e+00],
       [ 5.79135019e-05],
       [ 8.53041205e-04],
       [ 1.25862280e-03],
       [ 4.60695564e-04],
       [ 1.89671301e-04],
       [ 3.52490835e-05]]),
 -1.5161875221606185)
```

PROGRAM 6

Date: 19-04-2024

Build KNN Classification model for a given dataset.

Algorithm



Code

```
In [1]: import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt # for data visualization purposes
import seaborn as sns # for data visualization
%matplotlib inline

In [2]: data = '/content/cancer_detector.txt'
df = pd.read_csv(data, header=None)

In [3]: df.shape

Out[3]: (699, 11)

In [4]: col_names = ['Id', 'Clump_thickness', 'Uniformity_Cell_Size', 'Uniformity_Cell_Shape', 'Marginal_Adhesion',
   'Single_Epithelial_Cell_Size', 'Bare_Nuclei', 'Bland_Chromatin', 'Normal_Nucleoli', 'Mitoses', 'Class']

df.columns = col_names

df.columns

Out[4]: Index(['Id', 'Clump_thickness', 'Uniformity_Cell_Size',
   'Uniformity_Cell_Shape', 'Marginal_Adhesion',
   'Single_Epithelial_Cell_Size', 'Bare_Nuclei', 'Bland_Chromatin',
   'Normal_Nucleoli', 'Mitoses', 'Class'],
   dtype='object')

In [5]: df.head()

Out[5]:    Id Clump_thickness Uniformity_Cell_Size Uniformity_Cell_Shape Marginal_Adhesion Single_Epithelial_Cell_Size Bare_Nuclei Bla
  0  1000025          5              1                  1                  1                      2                  1
  1  1002945          5              4                  4                  5                      7                 10
  2  1015425          3              1                  1                  1                      2                  2
  3  1016277          6              8                  8                  1                      3                  4
  4  1017023          4              1                  1                  3                      2                  1
```

```
In [18]: import numpy as np
```

```
In [21]: # view summary statistics in numerical variables
print(round(df.describe(),2))

   Clump_Thickness Uniformity_Cell_Size Uniformity_Cell_Shape \
count          699.00            699.00            699.00
mean           4.42             3.13             3.21
std            2.82             3.05             2.97
min            1.00             1.00             1.00
25%           2.00             1.00             1.00
50%           4.00             1.00             1.00
75%           6.00             5.00             5.00
max           10.00            10.00            10.00

   Marginal_Adhesion Single_Epithelial_Cell_Size Bare_Nuclei \
count          699.00            699.00            683.00
mean           2.81             3.22             3.54
std            2.86             2.21             3.64
min            1.00             1.00             1.00
25%           1.00             2.00             1.00
50%           1.00             2.00             1.00
75%           4.00             4.00             6.00
max           10.00            10.00            10.00

   Bland_Chromatin Normal_Nucleoli Mitoses Class
count          699.00            699.00            699.00
mean           3.44             2.87             1.59             2.69
std            2.44             3.05             1.72             0.95
min            1.00             1.00             1.00             2.00
25%           2.00             1.00             1.00             2.00
50%           3.00             1.00             1.00             2.00
75%           5.00             4.00             1.00             4.00
max           10.00            10.00            10.00            4.00

In [22]: X = df.drop(['Class'], axis=1)

y = df['Class']

In [23]: 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, random_state = 0)

In [24]: X_train.shape, X_test.shape

Out[24]: ((559, 9), (140, 9))

In [25]: for col in X_train.columns:
    if X_train[col].isnull().mean()>0:
        print(col, round(X_train[col].isnull().mean(),4))

Bare_Nuclei 0.0233

In [26]: for df1 in [X_train, X_test]:
    for col in X_train.columns:
        col_median=X_train[col].median()
        df1[col].fillna(col_median, inplace=True)

In [27]: cols = X_train.columns

In [28]: from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

X_train = scaler.fit_transform(X_train)

X_test = scaler.transform(X_test)

In [29]: X_train = pd.DataFrame(X_train, columns=[cols])

In [30]: X_test = pd.DataFrame(X_test, columns=[cols])
```

Output

```
In [33]: y_pred = knn.predict(X_test)

y_pred
```

```
Out[33]: array([2, 2, 4, 2, 4, 2, 4, 2, 2, 2, 4, 4, 4, 2, 2, 4, 4, 2, 4, 4,
2, 2, 2, 4, 2, 2, 4, 4, 2, 2, 2, 2, 2, 2, 2, 2, 4, 2, 2, 2, 2, 2, 2, 2,
```

```
4, 4, 2, 4, 2, 4, 4, 2, 2, 2, 4, 2, 2, 2, 2, 4, 2, 2, 4, 4, 4, 4, 4,
```

```
4, 2, 2, 4, 2, 2, 4, 4, 2, 2, 2, 2, 4, 2, 2, 2, 4, 2, 2, 2, 4, 2, 2,
```

```
4, 4, 2, 2, 2, 4, 2, 2, 2, 4, 2, 4, 4, 2, 2, 2, 4, 2, 2, 2, 4, 2, 2,
```

```
4, 4, 4, 2, 2, 2, 2, 2, 4, 4, 4, 4, 4, 2, 2, 4, 4, 4, 4, 4, 4, 2,
```

```
2, 4, 4, 2, 2, 4, 2, 2])
```

```
In [34]: knn.predict_proba(X_test)[:,0]
```

```
Out[34]: array([1.        , 1.        , 0.33333333, 1.        , 0.        ,
1.        , 0.        , 1.        , 0.        , 0.66666667,
```

```
1.        , 1.        , 0.        , 0.33333333, 0.        ,
```

```
1.        , 1.        , 0.        , 0.        , 1.        , 1.        ,
```

```
0.        , 0.        , 1.        , 1.        , 1.        , 1.        ,
```

```
0.        , 1.        , 1.        , 0.        , 0.        , 0.        ,
```

```
1.        , 1.        , 1.        , 1.        , 1.        , 1.        ,
```

```
0.66666667, 1.        , 0.        , 1.        , 1.        , 1.        ,
```

```
1.        , 1.        , 1.        , 1.        , 0.        , 0.        ,
```

```
0.        , 1.        , 0.        , 1.        , 0.        , 0.        ,
```

```
0.        , 1.        , 1.        , 0.        , 1.        , 1.        ,
```

```
1.        , 1.        , 1.        , 0.66666667, 1.        ,
```

```
0.        , 1.        , 1.        , 0.        , 0.        , 0.        ,
```

```
0.33333333, 0.        , 1.        , 1.        , 0.        , 0.        ,
```

```
1.        , 1.        , 0.        , 0.        , 1.        , 1.        ,
```

```
1.        , 1.        , 1.        , 0.        , 0.        , 1.        ,
```

```
1.        , 1.        , 0.        , 1.        , 1.        , 1.        ,
```

```
1.        , 0.        , 1.        , 0.        , 0.        , 0.        ,
```

```
1.        , 1.        , 0.66666667, 0.        , 1.        , 1.        ,
```

```
1.        , 1.        , 0.        , 0.        , 1.        , 0.        ,
```

```
0.        , 1.        , 1.        , 1.        , 0.        , 0.        ,
```

```
1.        , 1.        , 0.33333333, 0.        , 1.        , 1.        ,
```

```
1.        , 1.        , 1.        , 0.        , 0.        , 0.        ,
```

```
0.        , 0.33333333, 1.        , 0.        , 1.        , 1.        ,
```

```
1.        , 0.33333333, 0.33333333, 0.        , 0.        , 0.        ,
```

```
0.        , 1.        , 1.        , 0.33333333, 0.        , 0.        ,
```

```
1.        , 1.        , 0.        , 1.        , 1.        , 1.        ,])
```

```
In [35]: from sklearn.metrics import accuracy_score

print('Model accuracy score: {:.4f}'.format(accuracy_score(y_test, y_pred)))
```

```
Model accuracy score: 0.9714
```

```
In [36]: y_pred_train = knn.predict(X_train)
```

```
In [37]: print('Training-set accuracy score: {:.4f}'.format(accuracy_score(y_train, y_pred_train)))
```

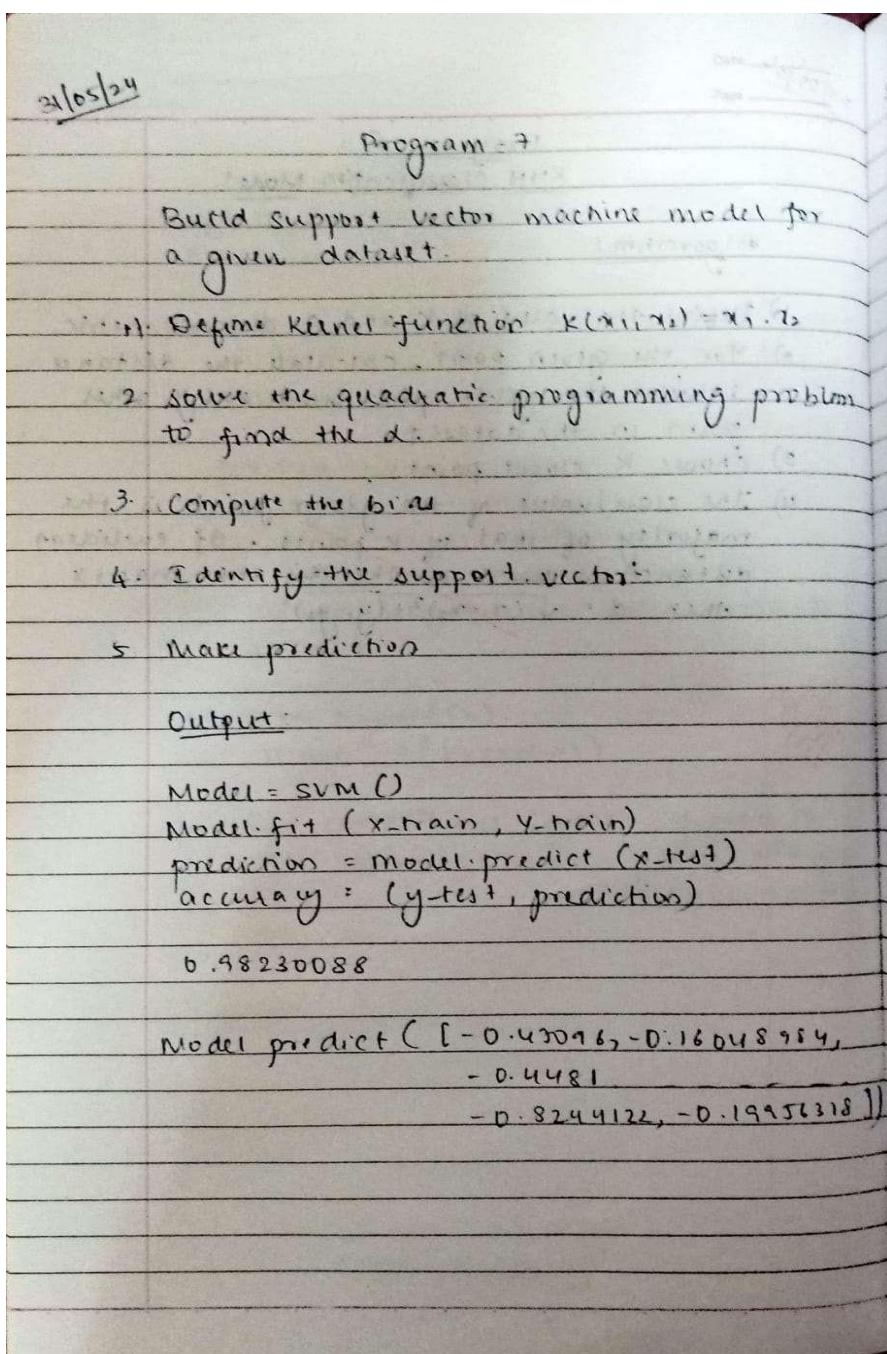
```
Training-set accuracy score: 0.9821
```

PROGRAM 7

Date:24-05-2024

Build Support vector machine model for a given dataset

Algorithm



Code

```
In [1]: import seaborn as sns
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import plotly.express as px

In [2]: df = pd.read_csv('/content/breast-cancer.csv')
df.head()

Out[2]:  
id diagnosis radius_mean texture_mean perimeter_mean area_mean smoothness_mean compactness_mean concavity_mean  
0 842302 M 17.99 10.38 122.80 1001.0 0.1840 0.2760 0.300  
1 842517 M 20.57 17.77 132.90 1326.0 0.08474 0.07864 0.0869  
2 84300903 M 19.69 21.25 130.00 1209.0 0.10960 0.15990 0.197  
3 84348801 M 11.42 20.38 77.58 386.1 0.14250 0.28390 0.241  
4 84358402 M 20.29 14.34 135.10 1297.0 0.10080 0.13280 0.1981  
5 rows × 32 columns  
  
In [3]: df.drop('id', axis=1, inplace=True) #drop redundant columns  
  
In [4]: df.describe().T  
  
Out[4]:  
count mean std min 25% 50% 75% max  
radius_mean 569.0 14.127292 3.524049 6.981000 11.700000 13.370000 15.780000 28.11000  
texture_mean 569.0 19.289649 4.301086 9.710000 16.170000 18.840000 21.800000 39.28000  
perimeter_mean 569.0 91.969083 24.298981 43.790000 75.170000 86.240000 104.100000 188.50000  
area_mean 569.0 654.889104 351.914129 143.500000 420.900000 551.100000 782.700000 2501.00000  
smoothness_mean 569.0 0.096360 0.014064 0.052630 0.086170 0.095870 0.105300 0.16340  
compactness_mean 569.0 0.104341 0.052813 0.019980 0.054920 0.092630 0.130400 0.34540  
concavity_mean 569.0 0.088799 0.079720 0.000000 0.029560 0.061540 0.130700 0.42680  
concave points_mean 569.0 0.048919 0.058808 0.000000 0.020310 0.033500 0.074000 0.20120  
symmetry_mean 569.0 0.181162 0.027414 0.106000 0.161900 0.179200 0.195700 0.30400  
fractal_dimension_mean 569.0 0.062798 0.007060 0.049960 0.057700 0.061540 0.066120 0.09744  
radius_se 569.0 0.405172 0.277313 0.111500 0.212400 0.324200 0.478900 2.87300  
texture_se 569.0 1.216853 0.551648 0.366200 0.833900 1.108000 1.474000 4.88500  
perimeter_se 569.0 2.866059 2.021855 0.757000 1.606000 2.287000 3.357000 21.98000  
area_se 569.0 40.337079 45.491006 6.802000 17.850000 24.530000 45.190000 542.20000  
smoothness_se 569.0 0.007041 0.003003 0.001713 0.005169 0.006580 0.008146 0.08113  
compactness_se 569.0 0.025478 0.017908 0.002252 0.019080 0.020450 0.032450 0.13540  
concavity_se 569.0 0.031894 0.030186 0.000000 0.015090 0.025890 0.042050 0.39600  
concave points_se 569.0 0.011796 0.006170 0.000000 0.007638 0.010930 0.014710 0.05279  
symmetry_se 569.0 0.030542 0.008266 0.007882 0.015160 0.018730 0.023480 0.07895  
fractal_dimension_se 569.0 0.003795 0.002646 0.000695 0.002248 0.003187 0.004558 0.02964  
radius_worst 569.0 16.269190 4.833242 7.930000 13.010000 14.970000 18.790000 36.04000  
texture_worst 569.0 25.677223 6.146258 12.020000 21.080000 25.410000 29.720000 49.54000  
perimeter_worst 569.0 107.261213 33.602542 50.410000 84.110000 97.660000 125.400000 251.20000  
area_worst 569.0 880.583128 569.356993 185.200000 515.300000 686.500000 1064.000000 4254.00000  
smoothness_worst 569.0 0.132369 0.022832 0.071170 0.116600 0.181300 0.146000 0.22260  
compactness_worst 569.0 0.254265 0.157336 0.027290 0.147200 0.211900 0.339100 1.05800  
concavity_worst 569.0 0.272188 0.208624 0.000000 0.114500 0.226700 0.382900 1.25200  
concave points_worst 569.0 0.114606 0.065732 0.000000 0.064980 0.099980 0.161400 0.29100  
symmetry_worst 569.0 0.290076 0.061867 0.156500 0.250400 0.282200 0.317900 0.66380  
fractal_dimension_worst 569.0 0.083946 0.018061 0.055040 0.071460 0.080040 0.092080 0.20750
```

```

In [1]: df['diagnosis'] = (df['diagnosis'] == 'M').astype(int) #encode the label into 1/0

In [2]: corr = df.corr()

In [3]: # Get the absolute value of the correlation
cor_target = abs(corr["diagnosis"])

# Select highly correlated features (threshold = 0.2)
relevant_features = cor_target[cor_target>0.2]

# Collect the names of the features
names = [Index for index, value in relevant_features.items()]

# Drop the target variable from the results
names.remove('diagnosis')

# Display the results
print(names)

['radius_mean', 'texture_mean', 'perimeter_mean', 'area_mean', 'smoothness_mean', 'compactness_mean', 'concave points_mean', 'symmetry_mean', 'radius_se', 'perimeter_se', 'area_se', 'compactness_se', 'concave points_se', 'radius_worst', 'texture_worst', 'perimeter_worst', 'area_worst', 'smoothness_worst', 'compactness_worst', 'concave points_worst', 'symmetry_worst', 'fractal_dimension_worst']

In [4]: X = df[names].values
y = df['diagnosis']

In [5]: def scale(X):
    """
    Standardizes the data in the array X.

    Parameters:
    X (numpy.ndarray): Features array of shape (n_samples, n_features).

    Returns:
    numpy.ndarray: The standardized features array.

    # Calculate the mean and standard deviation of each feature
    mean = np.mean(X, axis=0)
    std = np.std(X, axis=0)

    # Standardize the data
    X = (X - mean) / std
    return X

In [6]: X = scale(X)

In [7]: def train_test_split(X, y, random_state=42, test_size=0.2):
    """
    Splits the data into training and testing sets.

    Parameters:
    X (numpy.ndarray): Features array of shape (n_samples, n_features).
    y (numpy.ndarray): Target array of shape (n_samples,).
    random_state (int): Seed for the random number generator. Default is 42.
    test_size (float): Proportion of samples to include in the test set. Default is 0.2.

    Returns:
    Tuple[numpy.ndarray]: A tuple containing X_train, X_test, y_train, y_test.

    # Get number of samples
    n_samples = X.shape[0]

    # Set the seed for the random number generator
    np.random.seed(random_state)

    # Shuffle the indices
    shuffled_indices = np.random.permutation(np.arange(n_samples))

    # Determine the size of the test set
    test_size = int(n_samples * test_size)

    # Split the indices into test and train
    test_indices = shuffled_indices[:test_size]
    train_indices = shuffled_indices[test_size:]

    # Split the features and target arrays into test and train
    X_train, X_test = X[train_indices], X[test_indices]
    y_train, y_test = y[train_indices], y[test_indices]

    return X_train, X_test, y_train, y_test

In [8]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state=42) #split the data into training and testing sets

```

```
In [ ]: class SVM:

    def __init__(self, iterations=1000, lr=0.01, lambdaa=0.01):
        self.lambdaa = lambdaa
        self.iterations = iterations
        self.lr = lr
        self.w = None
        self.b = None
    def initialize_parameters(self, X):
        m, n = X.shape
        self.w = np.zeros(n)
        self.b = 0

    def gradient_descent(self, X, y):
        y_ = np.where(y <= 0, -1, 1)
        for i, x in enumerate(X):
            if y_[i] * (np.dot(x, self.w) - self.b) >= 1:
                dw = 2 * self.lambdaa * self.w
                db = 0
            else:
                dw = 2 * self.lambdaa * self.w - np.dot(x, y_[i])
                db = y_[i]
            self.update_parameters(dw, db)

    def update_parameters(self, dw, db):
        self.w = self.w - self.lr * dw
        self.b = self.b - self.lr * db
    def fit(self, X, y):
        self.initialize_parameters(X)
        for i in range(self.iterations):
            self.gradient_descent(X, y)

    def predict(self, X):
        # get the outputs
        output = np.dot(X, self.w) - self.b
        # get the signs of the labels depending on if it's greater/less than zero
        label_signs = np.sign(output)
        # set predictions to 0 if they are less than or equal to -1 else set them to 1
        predictions = np.where(label_signs <= -1, 0, 1)
        return predictions
```

```
In [ ]: def accuracy(y_true, y_pred):
    total_samples = len(y_true)
    correct_predictions = np.sum(y_true == y_pred)
    return (correct_predictions / total_samples)
```

Output

```
In [ ]: model = SVM()
model.fit(X_train,y_train)
predictions = model.predict(X_test)

accuracy(y_test, predictions)
```

Out[]: 0.9823008849557522

PROGRAM 8

Date: 31-05-2024

Build Artificial Neural Network model with back propagation on a given dataset

Algorithm

31/5/24 Date: 31/5/24 Page: 1

Program 8

Build Artificial Neural Network model with back propagation on a given dataset

Algorithm :

1. Create a feed-forward network with n_i inputs, n_{hidden} hidden units, and n_{out} output units.
2. Initialize all network weights to small random numbers.
3. Until the termination condition is met, Do
 - For each (x, E) , in training examples, Do
 - Propagate the input forward through the network:
 1. Input the instance x , to the network and compute the output o_u of every unit u in the network.
 - Propagate the error backward through the network:
 2. For each network output unit k , calculate its error term δ_k
 $\delta_k \leftarrow o_k(1-o_k)(t_k - o_k)$
 3. For each hidden unit h , calculate its error term δ_h
 $\delta_h \leftarrow o_h(1-o_h) \sum_{k \in \text{outputs}} w_{hk} \delta_k$
 - Update each network weight w_{ji}
 $w_{ji} \leftarrow w_{ji} + \Delta w_{ji}$
where
$$\Delta w_{ji} = \eta \delta_j x_{ij}$$

Output : Learning Accuracy = 0.66233, Testing Accuracy = 1.0

Code

```
In [1]: import numpy as np
from sklearn.model_selection import train_test_split

db = np.loadtxt("/content/duke-breast-cancer.txt")
print("Database raw shape (%s,%s)" % np.shape(db))

Database raw shape (86,7130)

In [2]: np.random.shuffle(db)
y = db[:, 0]
x = np.delete(db, [0], axis=1)
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.1)
print(np.shape(x_train),np.shape(x_test))

(77, 7129) (9, 7129)

In [3]: hidden_layer = np.zeros(72)
weights = np.random.random((len(x[0]), 72))
output_layer = np.zeros(2)
hidden_weights = np.random.random((72, 2))

In [4]: def sum_function(weights, index_locked_col, x):
    result = 0
    for i in range(0, len(x)):
        result += x[i] * weights[i][index_locked_col]
    return result

In [5]: def activate_layer(layer, weights, x):
    for i in range(0, len(layer)):
        layer[i] = 1.7159 * np.tanh(2.0 * sum_function(weights, i, x) / 3.0)

In [6]: def soft_max(layer):
    soft_max_output_layer = np.zeros(len(layer))
    for i in range(0, len(layer)):
        denominator = 0
        for j in range(0, len(layer)):
            denominator += np.exp(layer[j] - np.max(layer))
        soft_max_output_layer[i] = np.exp(layer[i] - np.max(layer)) / denominator
    return soft_max_output_layer

In [7]: def recalculate_weights(learning_rate, weights, gradient, activation):
    for i in range(0, len(weights)):
        for j in range(0, len(weights[i])):
            weights[i][j] = (learning_rate * gradient[j] * activation[i]) + weights[i][j]

In [8]: def back_propagation(hidden_layer, output_layer, one_hot_encoding, learning_rate, x):
    output_derivative = np.zeros(2)
    output_gradient = np.zeros(2)
    for i in range(0, len(output_layer)):
        output_derivative[i] = (1.0 - output_layer[i]) * output_layer[i]
    for i in range(0, len(output_layer)):
        output_gradient[i] = output_derivative[i] * (one_hot_encoding[i] - output_layer[i])
    hidden_derivative = np.zeros(72)
    hidden_gradient = np.zeros(72)
    for i in range(0, len(hidden_layer)):
        hidden_derivative[i] = (1.0 - hidden_layer[i]) * (1.0 + hidden_layer[i])
    for i in range(0, len(hidden_layer)):
        sum_ = 0
        for j in range(0, len(output_gradient)):
            sum_ += output_gradient[j] * hidden_weights[i][j]
        hidden_gradient[i] = sum_ * hidden_derivative[i]
    recalculate_weights(learning_rate, hidden_weights, output_gradient, hidden_layer)
    recalculate_weights(learning_rate, weights, hidden_gradient, x)
```

Output

```
In [9]: one_hot_encoding = np.zeros((2,2))
for i in range(0, len(one_hot_encoding)):
    one_hot_encoding[i][i] = 1
training_correct_answers = 0
for i in range(0, len(x_train)):
    activate_layer(hidden_layer, weights, x_train[i])
    activate_layer(output_layer, hidden_weights, hidden_layer)
    output_layer = soft_max(output_layer)
    training_correct_answers += 1 if y_train[i] == np.argmax(output_layer) else 0
    back_propagation(hidden_layer, output_layer, one_hot_encoding[int(y_train[i])], -1, x_train[i])
print("MLP Correct answers while learning: %s / %s (Accuracy = %s) on %s database." % (training_correct_answers, len(x_train),
                                                                                     training_correct_answers/len(x_train), "Duke breast cancer"))
```

MLP Correct answers while learning: 51 / 77 (Accuracy = 0.6623376623376623) on Duke breast cancer database.

```
In [10]: testing_correct_answers = 0
for i in range(0, len(x_test)):
    activate_layer(hidden_layer, weights, x_test[i])
    activate_layer(output_layer, hidden_weights, hidden_layer)
    output_layer = soft_max(output_layer)
    testing_correct_answers += 1 if y_test[i] == np.argmax(output_layer) else 0
print("MLP Correct answers while testing: %s / %s (Accuracy = %s) on %s database" % (testing_correct_answers, len(x_test),
                                                                                      testing_correct_answers/len(x_test), "Duke breast cancer"))
```

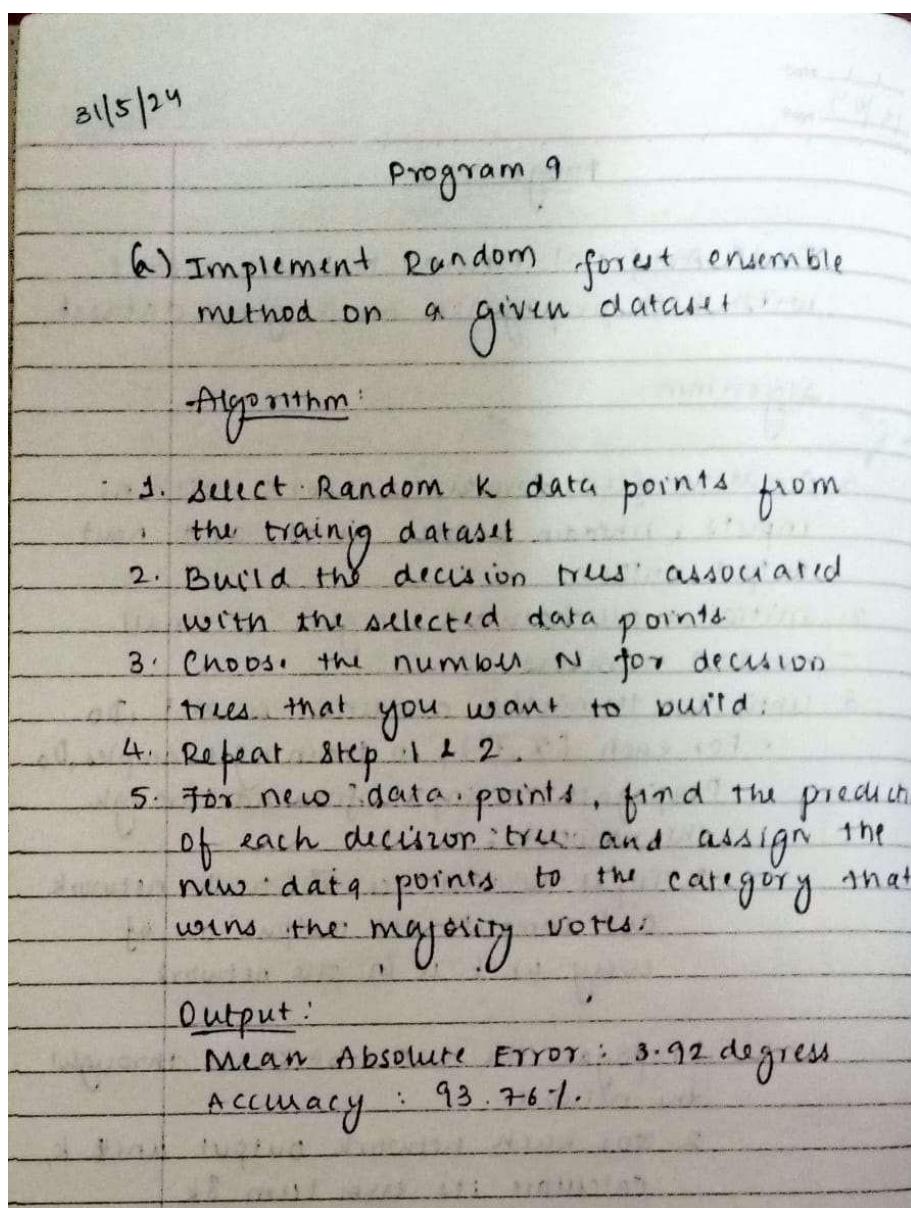
MLP Correct answers while testing: 9 / 9 (Accuracy = 1.0) on Duke breast cancer database

PROGRAM 9

Date: 31-05-2024

a) Implement Random forest ensemble method on a given dataset.

Algorithm



Code

[Open in Colab](#)

```
In [1]: # Pandas is used for data manipulation
import pandas as pd
# Read in data and display first 5 rows
features = pd.read_csv('/content/temp.csv')
features.head(5)
```

```
Out[1]:   year month day week temp_2 temp_1 average actual forecast_noaa forecast_acc forecast_under friend
0 2016 1 1 Fri 45 45 45.6 45 43 50 44 29
1 2016 1 2 Sat 44 45 45.7 44 41 50 44 61
2 2016 1 3 Sun 45 44 45.8 41 43 46 47 56
3 2016 1 4 Mon 44 41 45.9 40 44 48 46 53
4 2016 1 5 Tues 41 40 46.0 44 46 46 46 41
```

```
In [2]: print('The shape of our features is:', features.shape)
```

The shape of our features is: (348, 12)

```
In [3]: # Descriptive statistics for each column
features.describe()
```

```
Out[3]:   year month day temp_2 temp_1 average actual forecast_noaa forecast_acc forecast_under
count 348.0 348.000000 348.000000 348.000000 348.000000 348.000000 348.000000 348.000000 348.000000 348.000000
mean 2016.0 6.477011 15.514368 62.652299 62.701149 59.760632 62.543103 57.238506 62.373563 59.772989 60.0
std 0.0 3.498380 8.772982 12.165398 12.120542 10.527306 11.794146 10.605746 10.549381 10.705256 15.1
min 2016.0 1.000000 1.000000 35.000000 35.000000 45.100000 35.000000 41.000000 46.000000 44.000000 28.0
25% 2016.0 3.000000 8.000000 54.000000 54.000000 49.975000 54.000000 48.000000 53.000000 50.000000 47.0
50% 2016.0 6.000000 15.000000 62.500000 62.500000 58.200000 62.500000 56.000000 61.000000 58.000000 60.0
75% 2016.0 10.000000 23.000000 71.000000 71.000000 69.025000 71.000000 66.000000 72.000000 69.000000 71.0
max 2016.0 12.000000 31.000000 117.000000 117.000000 77.400000 92.000000 77.000000 82.000000 79.000000 95.0
```

```
In [4]: # One-hot encode the data using pandas get_dummies
features = pd.get_dummies(features)
# Display the first 5 rows of the last 12 columns
features.iloc[:,5:].head(5)
```

```
Out[4]:   average actual forecast_noaa forecast_acc forecast_under friend week_Fri week_Mon week_Sat week_Sun week_Thurs week_Tu
0 45.6 45 43 50 44 29 True False False False False Fa
1 45.7 44 41 50 44 61 False False True False False Fa
2 45.8 41 43 46 47 56 False False False True False Fa
3 45.9 40 44 48 46 53 False True False False False Fa
4 46.0 44 46 46 46 41 False False False False False Fa
```

```
In [5]: # Use numpy to convert to arrays
import numpy as np
# Labels are the values we want to predict
labels = np.array(features['actual'])
# Remove the labels from the features
# axis 1 refers to the columns
features= features.drop('actual', axis = 1)
# Saving feature names for later use
feature_list = list(features.columns)
# Convert to numpy array
features = np.array(features)

In [6]: # Using Skicit-learn to split data into training and testing sets
from sklearn.model_selection import train_test_split
# Split the data into training and testing sets
train_features, test_features, train_labels, test_labels = train_test_split(features, labels, test_size = 0.25, random_state = 42)

In [7]: print('Training Features Shape:', train_features.shape)
print('Training Labels Shape:', train_labels.shape)
print('Testing Features Shape:', test_features.shape)
print('Testing Labels Shape:', test_labels.shape)

Training Features Shape: (261, 17)
Training Labels Shape: (261,)
Testing Features Shape: (87, 17)
Testing Labels Shape: (87,)

In [8]: # The baseline predictions are the historical averages
baseline_preds = test_features[:, feature_list.index('average')]
# Baseline errors, and display average baseline error
baseline_errors = abs(baseline_preds - test_labels)
print('Average baseline error: ', round(np.mean(baseline_errors), 2))

Average baseline error:  5.06

In [9]: # Import the model we are using
from sklearn.ensemble import RandomForestRegressor
# Instantiate model with 1000 decision trees
rf = RandomForestRegressor(n_estimators = 1000, random_state = 42)
# Train the model on training data
rf.fit(train_features, train_labels);

In [10]: # Use the forest's predict method on the test data
predictions = rf.predict(test_features)
# Calculate the absolute errors
errors = abs(predictions - test_labels)
# Print out the mean absolute error (mae)
print('Mean Absolute Error:', round(np.mean(errors), 2), 'degrees.')

Mean Absolute Error: 3.87 degrees.

In [11]: # Calculate mean absolute percentage error (MAPE)
mape = 100 * (errors / test_labels)
# Calculate and display accuracy
accuracy = 100 - np.mean(mape)
print('Accuracy:', round(accuracy, 2), '%')

Accuracy: 93.93 %.
```

```
In [12]: # Import tools needed for visualization
from sklearn.tree import export_graphviz
import pydot
# Pull out one tree from the forest
tree = rf.estimators_[5]
# Import tools needed for visualization
from sklearn.tree import export_graphviz
import pydot
# Pull out one tree from the forest
tree = rf.estimators_[5]
# Export the image to a dot file
export_graphviz(tree, out_file = 'tree.dot', feature_names = feature_list, rounded = True, precision = 1)
# Use dot file to create a graph
(graph, ) = pydot.graph_from_dot_file('tree.dot')
# Write graph to a png file
graph.write_png('tree.png')
```

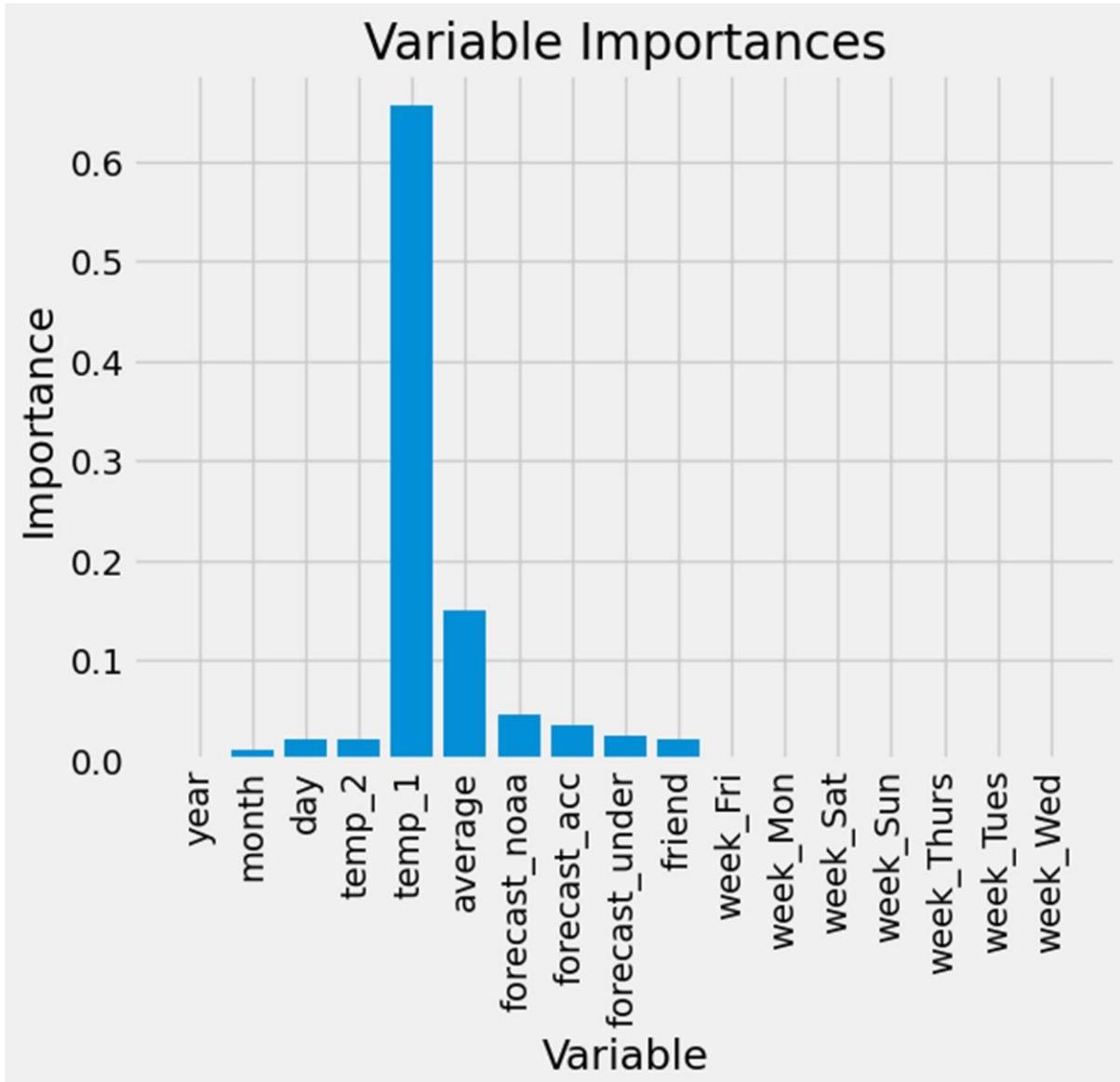
```
In [13]: # Limit depth of tree to 3 levels
rf_small = RandomForestRegressor(n_estimators=10, max_depth = 3)
rf_small.fit(train_features, train_labels)
# Extract the small tree
tree_small = rf_small.estimators_[5]
# Save the tree as a png image
export_graphviz(tree_small, out_file = 'small_tree.dot', feature_names = feature_list, rounded = True, precision = 1)
(graph, ) = pydot.graph_from_dot_file('small_tree.dot')
graph.write_png('small_tree.png');
```

```
In [14]: # Get numerical feature importances
importances = list(rf.feature_importances_)
# List of tuples with variable and importance
feature_importances = [(feature, round(importance, 2)) for feature, importance in zip(feature_list, importances)]
# Sort the feature importances by most important first
feature_importances = sorted(feature_importances, key = lambda x: x[1], reverse = True)
# Print out the feature and importances
[print('Variable: {:20} Importance: {}'.format(*pair)) for pair in feature_importances];
```

Variable	Importance
temp_1	0.66
average	0.15
forecast_noaa	0.05
forecast_acc	0.03
day	0.02
temp_2	0.02
forecast_under	0.02
friend	0.02
month	0.01
year	0.0
week_Fri	0.0
week_Mon	0.0
week_Sat	0.0
week_Sun	0.0
week_Thurs	0.0
week_Tues	0.0
week_Wed	0.0

```
In [15]: # New random forest with only the two most important variables
rf_most_important = RandomForestRegressor(n_estimators= 1000, random_state=42)
# Extract the two most important features
important_indices = [feature_list.index('temp_1'), feature_list.index('average')]
train_important = train_features[:, important_indices]
test_important = test_features[:, important_indices]
# Train the random forest
rf_most_important.fit(train_important, train_labels)
# Make predictions and determine the error
predictions = rf_most_important.predict(test_important)
errors = abs(predictions - test_labels)
# Display the performance metrics
print('Mean Absolute Error:', round(np.mean(errors), 2), 'degrees.')
mape = np.mean(100 * (errors / test_labels))
accuracy = 100 - mape
print('Accuracy:', round(accuracy, 2), '%.')
```

Output



b) Implement Boosting ensemble method on a given dataset.

Algorithm

Date _____
Page _____

(b) Implement Boosting Ensemble on a given dataset

Algorithm:

1. Initialize the dataset and assign equal weight to each of the data point.
2. Provide this as input to the model and identify the wrongly classified datapoints.
3. Increase the weight of the wrongly classified data points and decrease the weights of correctly classified data points. And then normalize the weights of all data points.
4. If (got required results)
 Goto step - 5
Else
 Goto step - 2
5. End.

Output

Confusion Matrix : $\begin{bmatrix} 116 & 35 \\ 26 & 54 \end{bmatrix}$

Accuracy score : 0.7359.

Code

```
In [1]:  
import pandas as pd  
import numpy as np  
import matplotlib.pyplot as plt  
import seaborn as sns  
  
%matplotlib inline  
sns.set_style("whitegrid")  
plt.style.use("fivethirtyeight")
```

```
In [2]:  
df = pd.read_csv("/content/diabetes.csv")  
df.head()
```

```
Out[2]:  
   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
```

```
In [3]:  
df.info()  
  
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 768 entries, 0 to 767  
Data columns (total 9 columns):  
 #   Column           Non-Null Count  Dtype     
---  --  
 0   Pregnancies      768 non-null    int64    
 1   Glucose          768 non-null    int64    
 2   BloodPressure    768 non-null    int64    
 3   SkinThickness    768 non-null    int64    
 4   Insulin          768 non-null    int64    
 5   BMI              768 non-null    float64  
 6   DiabetesPedigreeFunction 768 non-null    float64  
 7   Age              768 non-null    int64    
 8   Outcome          768 non-null    int64    
dtypes: float64(2), int64(7)  
memory usage: 54.1 KB
```

```
In [4]:  
df.isnull().sum()
```

```
Out[4]:  
Pregnancies      0  
Glucose          0  
BloodPressure    0  
SkinThickness    0  
Insulin          0  
BMI              0  
DiabetesPedigreeFunction 0  
Age              0  
Outcome          0  
dtype: int64
```

```
In [5]:  
pd.set_option('display.float_format', '{:.2f}'.format)  
df.describe()
```

```
Out[5]:  
   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin  BMI  DiabetesPedigreeFunction  Age  Outcome  
count    768.00    768.00      768.00      768.00    768.00             768.00  768.00    768.00  
mean     3.85    120.89      69.11      20.54    79.80    31.99             0.47  33.24    0.35  
std      3.37     31.97      19.36      15.95   115.24     7.88             0.33  11.76    0.48  
min      0.00     0.00      0.00      0.00     0.00     0.00             0.08  21.00    0.00  
25%     1.00     99.00      62.00      0.00     0.00    27.30             0.24  24.00    0.00  
50%     3.00    117.00      72.00      23.00    30.50    32.00             0.37  29.00    0.00  
75%     6.00    140.25      80.00      32.00   127.25    36.60             0.63  41.00    1.00  
max    17.00    199.00     122.00      99.00   846.00    67.10             2.42  81.00    1.00
```

```
In [6]: categorical_val = []
continuous_val = []
for column in df.columns:
    #     print('*****')
    #     print(f'{column} : {df[column].unique()}')
    if len(df[column].unique()) <= 10:
        categorical_val.append(column)
    else:
        continuous_val.append(column)

In [7]: df.columns

Out[7]: Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
       dtype='object')

In [8]: # How many missing zeros are missing in each feature
feature_columns = [
    'Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness',
    'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age'
]

for column in feature_columns:
    print("*****")
    print(f"{column} ==> Missing zeros : {len(df.loc[df[column] == 0])}")

*****
Pregnancies ==> Missing zeros : 111
*****
Glucose ==> Missing zeros : 5
*****
BloodPressure ==> Missing zeros : 35
*****
SkinThickness ==> Missing zeros : 227
*****
Insulin ==> Missing zeros : 374
*****
BMI ==> Missing zeros : 11
*****
DiabetesPedigreeFunction ==> Missing zeros : 0
*****
Age ==> Missing zeros : 0

In [9]: from sklearn.impute import SimpleImputer

fill_values = SimpleImputer(missing_values=0, strategy="mean", copy=False)
df[feature_columns] = fill_values.fit_transform(df[feature_columns])

for column in feature_columns:
    print("*****")
    print(f"{column} ==> Missing zeros : {len(df.loc[df[column] == 0])}")

*****
Pregnancies ==> Missing zeros : 0
*****
Glucose ==> Missing zeros : 0
*****
BloodPressure ==> Missing zeros : 0
*****
SkinThickness ==> Missing zeros : 0
*****
Insulin ==> Missing zeros : 0
*****
BMI ==> Missing zeros : 0
*****
DiabetesPedigreeFunction ==> Missing zeros : 0
*****
Age ==> Missing zeros : 0
```

```
In [10]: from sklearn.model_selection import train_test_split

X = df[feature_columns]
y = df.Outcome

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

```
In [11]: from sklearn.metrics import confusion_matrix, accuracy_score, classification_report

def evaluate(model, X_train, X_test, y_train, y_test):
    y_test_pred = model.predict(X_test)
    y_train_pred = model.predict(X_train)

    print("TRAINING RESULTS: \n====")
    clf_report = pd.DataFrame(classification_report(y_train, y_train_pred, output_dict=True))
    print(f"CONFUSION MATRIX:\n{confusion_matrix(y_train, y_train_pred)}")
    print(f"ACCURACY SCORE:\n{accuracy_score(y_train, y_train_pred):.4f}")
    print(f"CLASSIFICATION REPORT:\n{clf_report}")

    print("TESTING RESULTS: \n====")
    clf_report = pd.DataFrame(classification_report(y_test, y_test_pred, output_dict=True))
    print(f"CONFUSION MATRIX:\n{confusion_matrix(y_test, y_test_pred)}")
    print(f"ACCURACY SCORE:\n{accuracy_score(y_test, y_test_pred):.4f}")
    print(f"CLASSIFICATION REPORT:\n{clf_report}")
```

```
In [12]: from sklearn.ensemble import AdaBoostClassifier

ada_boost_clf = AdaBoostClassifier(n_estimators=30)
ada_boost_clf.fit(X_train, y_train)
evaluate(ada_boost_clf, X_train, X_test, y_train, y_test)
```

Output-AdaBoost

```
TRAINING RESULTS:
=====
CONFUSION MATRIX:
[[310  39]
 [ 51 137]]
ACCURACY SCORE:
0.8324
CLASSIFICATION REPORT:
      0      1  accuracy  macro avg  weighted avg
precision  0.86   0.78      0.83      0.82      0.83
recall    0.89   0.73      0.83      0.81      0.83
f1-score   0.87   0.75      0.83      0.81      0.83
support   349.00 188.00      537.00      537.00
TESTING RESULTS:
=====
CONFUSION MATRIX:
[[123  28]
 [ 27  53]]
ACCURACY SCORE:
0.7619
CLASSIFICATION REPORT:
      0      1  accuracy  macro avg  weighted avg
precision  0.82   0.65      0.76      0.74      0.76
recall    0.81   0.66      0.76      0.74      0.76
f1-score   0.82   0.66      0.76      0.74      0.76
support   151.00  80.00      231.00      231.00
```

Output- GradientBoost

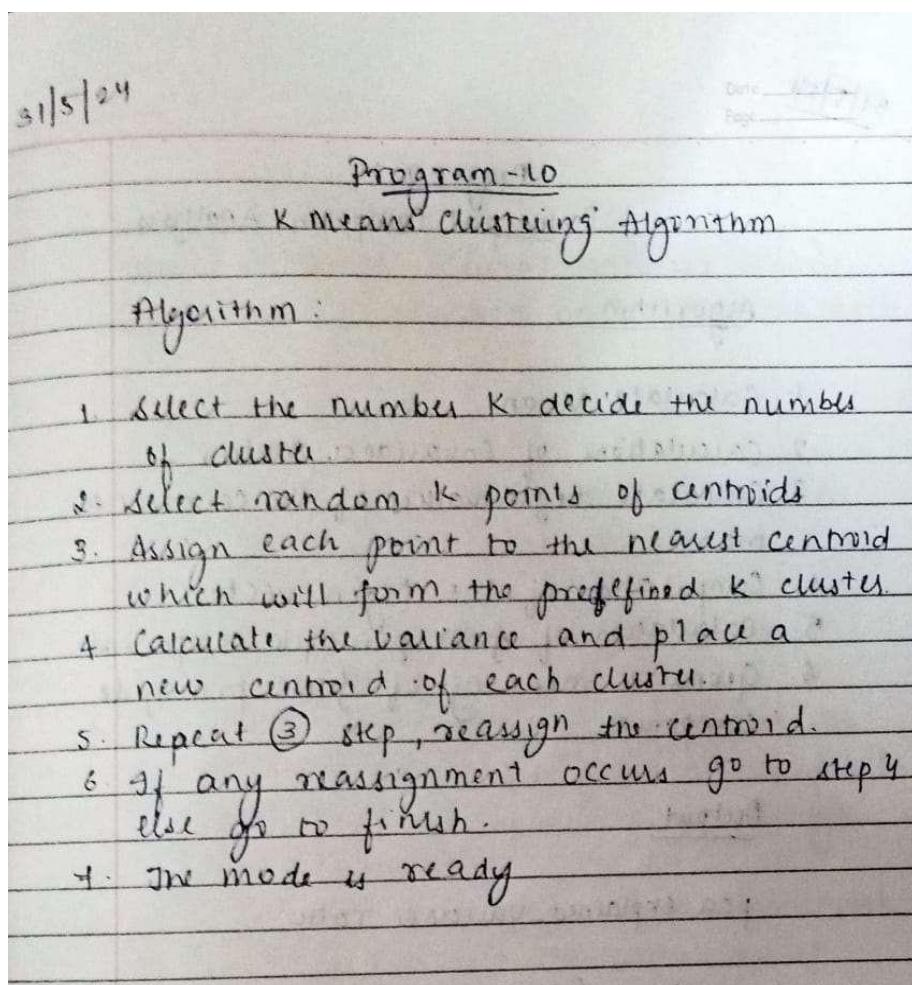
```
TRAINING RESULTS:  
=====  
CONFUSION MATRIX:  
[[342  7]  
 [ 19 169]]  
ACCURACY SCORE:  
0.9516  
CLASSIFICATION REPORT:  
          0      1  accuracy  macro avg  weighted avg  
precision  0.95  0.96      0.95      0.95      0.95  
recall    0.98  0.90      0.95      0.94      0.95  
f1-score   0.96  0.93      0.95      0.95      0.95  
support   349.00 188.00      0.95      537.00      537.00  
TESTING RESULTS:  
=====  
CONFUSION MATRIX:  
[[116  35]  
 [ 26  54]]  
ACCURACY SCORE:  
0.7359  
CLASSIFICATION REPORT:  
          0      1  accuracy  macro avg  weighted avg  
precision  0.82  0.61      0.74      0.71      0.74  
recall    0.77  0.68      0.74      0.72      0.74  
f1-score   0.79  0.64      0.74      0.72      0.74  
support   151.00 80.00      0.74      231.00      231.00
```

PROGRAM 10

Date: 24-05-2024

Build k-Means algorithm to cluster a set of data stored in a .CSV file.

Algorithm



Code

Importing and initializing the data points

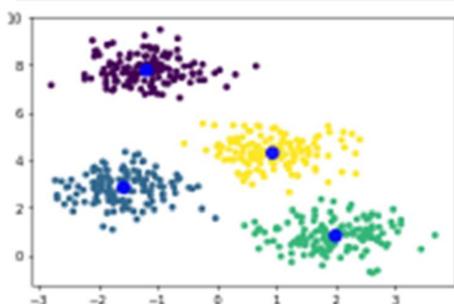
```
In [5]: import matplotlib.pyplot as plt  
import numpy as np  
from sklearn.cluster import KMeans  
  
In [6]: from sklearn.datasets import make_blobs  
X, y_true = make_blobs(n_samples=500, centers=4, cluster_std=0.60, random_state=0)  
  
In [7]: import plotly.express as px  
fig = px.scatter(x=X[:, 0], y=X[:, 1], width=800, height=500)  
fig.show()
```

Elbow Method to find optimal K

```
In [8]: cost = []  
for i in range(1, 11):  
    KM = KMeans(n_clusters = i, max_iter = 500)  
    KM.fit(X)  
  
    cost.append(KM.inertia_)  
  
# plot the cost against K values  
fig = px.line(x=range(1, 11), y=cost, width=600, height=400)  
fig.show()  
# the point of the elbow is the  
# most optimal value for choosing k
```

Defining Model and fitting the same

```
In [9]: kmeans = KMeans(n_clusters=4)  
kmeans.fit(X)  
y_kmeans = kmeans.predict(X)  
  
In [10]: fig = px.scatter(x=X[:, 0], y=X[:, 1], color=y_kmeans, width=700, height=400)  
trace = px.scatter(x=X[:, 0], y=X[:, 1], width=700, height=400)  
fig.show()  
  
In [11]: plt.scatter(X[:, 0], X[:, 1], c=y_kmeans, s=20)  
centers = kmeans.cluster_centers_  
plt.scatter(centers[:, 0], centers[:, 1], c="blue", s=100, alpha=0.9);  
plt.show()
```



Iris Dataset

```
In [12]: import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn import datasets

In [24]: iris = datasets.load_iris()
df = pd.DataFrame(iris.data)
df['class']=iris.target
df.columns=['sepal_len', 'sepal_wid', 'petal_len', 'petal_wid', 'class']
df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column      Non-Null Count  Dtype  
 --- 
  0   sepal_len    150 non-null   float64 
  1   sepal_wid    150 non-null   float64 
  2   petal_len    150 non-null   float64 
  3   petal_wid    150 non-null   float64 
  4   class        150 non-null   int64  
dtypes: float64(4), int64(1)
memory usage: 6.8 kB

In [52]: px.histogram(df, x='class', color='class')

In [56]: from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X = df.iloc[:,0:4].values

In [58]: scaled_x = scaler.fit_transform(X)

In [74]: model = KMeans(n_clusters=3,init='k-means++',random_state=0)
labels = model.fit_predict(scaled_x)

In [81]: import plotly.graph_objects as go
fig = go.Figure()

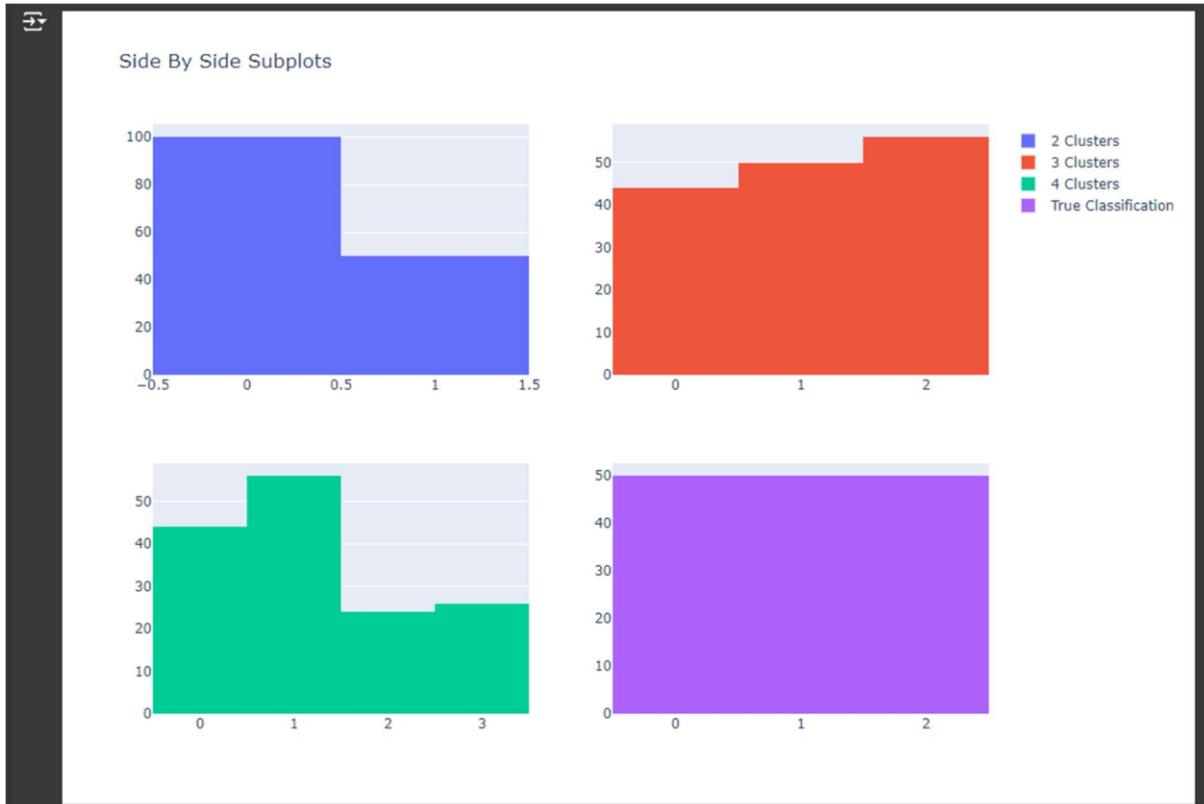
# Add trace
fig.add_trace(go.Histogram(x=labels,name="Predicted Labels"))
fig.add_trace(go.Histogram(x=df['class'],name="True Labels"))

# Overlay both histograms
fig.update_layout(barmode='overlay')
# Reduce opacity to see both histograms
fig.update_traces(opacity=0.75)
fig.show()

In [89]: labels = []
for i in range(2, 5):
    model = KMeans(n_clusters = i, max_iter = 500)
    model.fit(scaled_x)
    labels.append(model.fit_predict(scaled_x))

In [97]: from plotly.subplots import make_subplots
import plotly.graph_objects as go
fig = make_subplots(rows=2, cols=2)
for i in range(0, 3):
    fig.add_trace(go.Histogram(x=labels[i],name="{} Clusters".format(i+2)),
                 row=(i/2 + 1), col=(i%2 + 1))
    fig.add_trace(go.Histogram(x=df['class'],name="True Classification"),
                 row=(2), col=(2))
fig.update_layout(height=700, width=1000, title_text="Side By Side Subplots")
fig.show()
```

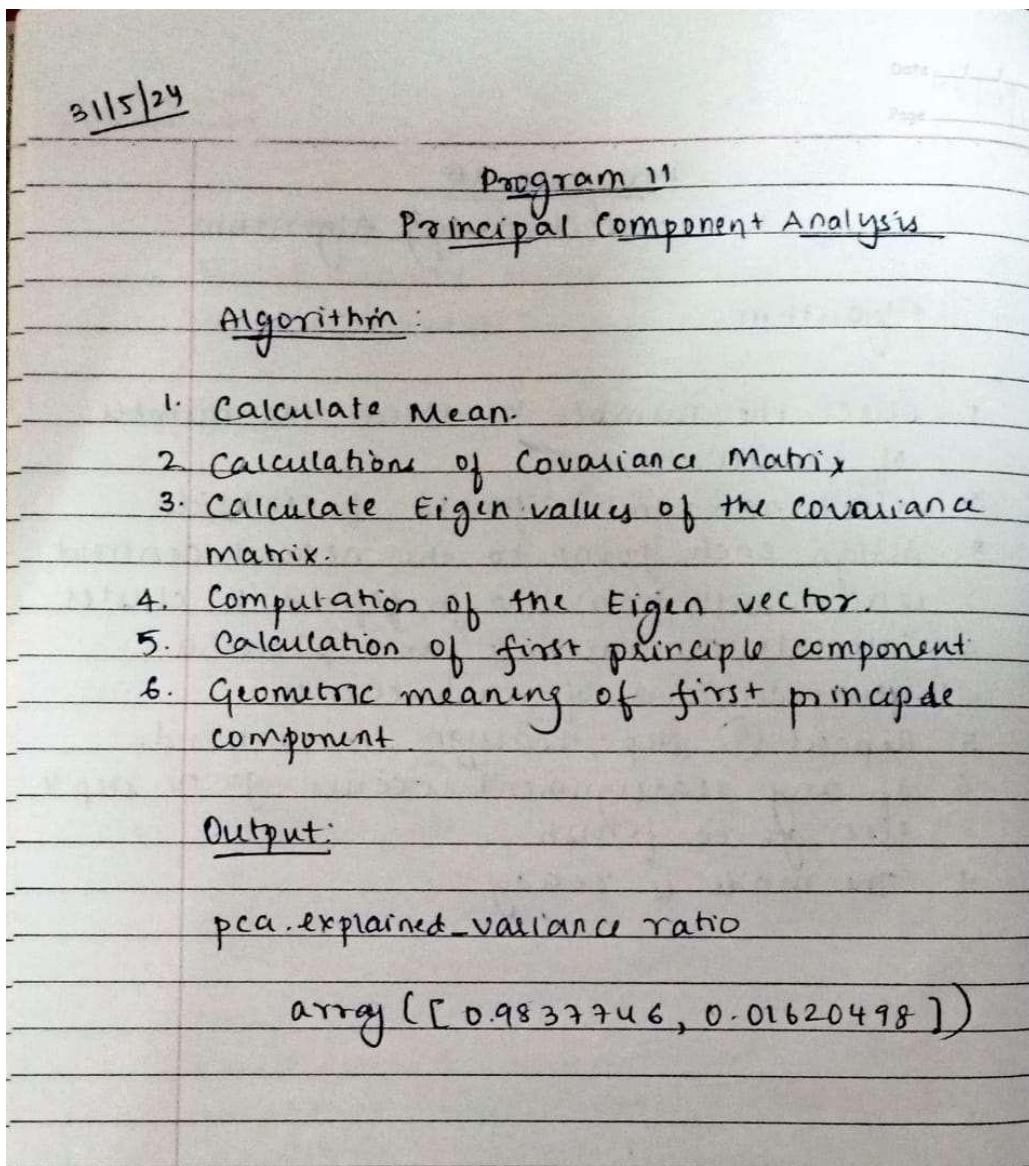
Output



PROGRAM 11

Date: 24-05-2024

Implement Dimensionality reduction using Principle Component Analysis (PCA) method.



Code

```
Open in Colab

In [ ]: from google.colab import drive
drive.mount('/content/drive')

Mounted at /content/drive

In [ ]: import seaborn as sns
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import plotly.express as px
import plotly.graph_objects as go
from plotly.subplots import make_subplots

In [ ]: df = pd.read_csv('/content/drive/MyDrive/breast-cancer.csv')
df.head()

Out[ ]:
   id diagnosis radius_mean texture_mean perimeter_mean area_mean smoothness_mean compactness_mean concavity_mean
0  842302      M       17.99      10.38      122.80     1001.0      0.11840      0.27760      0.3001
1  842517      M       20.57      17.77      132.90     1326.0      0.08474      0.07864      0.0866
2  84300908     M       19.69      21.25      130.00     1203.0      0.10960      0.15990      0.1974
3  84348301     M       11.42      20.38      77.58      386.1      0.14250      0.28390      0.2414
4  84358402     M       20.29      14.34      135.10     1297.0      0.10030      0.13280      0.1980

5 rows × 32 columns

In [ ]: df.drop('id', axis=1, inplace=True) #drop redundant columns

In [ ]: df['diagnosis'] = (df['diagnosis'] == 'M').astype(int) #encode the Label into 1/0

In [ ]: corr = df.corr()

In [ ]:
# Get the absolute value of the correlation
cor_target = abs(corr["diagnosis"])

# Select highly correlated features (threshold = 0.2)
relevant_features = cor_target[cor_target>0.2]

# Collect the names of the features
names = [index for index, value in relevant_features.items()]

# Drop the target variable from the results
names.remove('diagnosis')

# Display the results
print(names)

['radius_mean', 'texture_mean', 'perimeter_mean', 'area_mean', 'smoothness_mean', 'compactness_mean', 'concavity_mean', 'concave points_mean', 'symmetry_mean', 'radius_se', 'perimeter_se', 'area_se', 'compactness_se', 'concavity_se', 'concave points_se', 'radius_worst', 'texture_worst', 'perimeter_worst', 'area_worst', 'smoothness_worst', 'compactness_worst', 'concavity_worst', 'concave points_worst', 'symmetry_worst', 'fractal_dimension_worst']

In [ ]: X = df[names].values
```

```
In [ ]:
class PCA:
    """
    Principal Component Analysis (PCA) class for dimensionality reduction.
    """

    def __init__(self, n_components):
        """
        Constructor method that initializes the PCA object with the number of components to retain.

        Args:
        - n_components (int): Number of principal components to retain.
        """
        self.n_components = n_components

    def fit(self, X):
        """
        Fits the PCA model to the input data and computes the principal components.

        Args:
        - X (numpy.ndarray): Input data matrix with shape (n_samples, n_features).

        # Compute the mean of the input data along each feature dimension.
        mean = np.mean(X, axis=0)

        # Subtract the mean from the input data to center it around zero.
        X = X - mean

        # Compute the covariance matrix of the centered input data.
        cov = np.cov(X.T)

        # Compute the eigenvectors and eigenvalues of the covariance matrix.
        eigenvalues, eigenvectors = np.linalg.eigh(cov)
        # Reverse the order of the eigenvalues and eigenvectors.
        eigenvalues = eigenvalues[::-1]
        eigenvectors = eigenvectors[:, ::-1]

        # Keep only the first n_components eigenvectors as the principal components.
        self.components_ = eigenvectors[:, :self.n_components]

        # Compute the explained variance ratio for each principal component.
        # Compute the total variance of the input data
        total_variance = np.sum(np.var(X, axis=0))

        # Compute the variance explained by each principal component
        self.explained_variances_ = eigenvalues[:self.n_components]

        # Compute the explained variance ratio for each principal component
        self.explained_variance_ratio_ = self.explained_variances_ / total_variance
    """

    def transform(self, X):
        """
        Transforms the input data by projecting it onto the principal components.

        Args:
        - X (numpy.ndarray): Input data matrix with shape (n_samples, n_features).

        Returns:
        - transformed_data (numpy.ndarray): Transformed data matrix with shape (n_samples, n_components).
        """
        # Center the input data around zero using the mean computed during the fit step.
        X = X - np.mean(X, axis=0)

        # Project the centered input data onto the principal components.
        transformed_data = np.dot(X, self.components_)

        return transformed_data

    def fit_transform(self, X):
        """
        Fits the PCA model to the input data and computes the principal components then
        transforms the input data by projecting it onto the principal components.

        Args:
        - X (numpy.ndarray): Input data matrix with shape (n_samples, n_features).
        """
        self.fit(X)
        transformed_data = self.transform(X)
        return transformed_data
```

```
In [ ]: pca = PCA(2)

In [ ]: pca.fit(X)

In [ ]: pca.explained_variance_ratio_

Out[ ]: array([0.98377428, 0.01620498])

In [ ]: X_transformed = pca.transform(X)

In [ ]: X_transformed[:,1].shape

Out[ ]: (569,)

In [ ]: fig = px.scatter(x=X_transformed[:,0], y=X_transformed[:,1])
fig.update_layout(
    title="PCA transformed data for breast cancer dataset",
    xaxis_title="PC1",
    yaxis_title="PC2"
)
fig.show()
```

Output

