

Lab 1

Date: 05.04.2024

Observation (Handwritten)

```
(1) Observation

# import the pandas library
import pandas as pd

# read the csv file
airbnb_data = pd.read_csv("/content/sample-data/mnist-test.csv")

# using url to take dataset
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"

col_names = ["sepal-length-in-cm", "sepal-width-in-cm", "petal-length-in-cm", "petal-width-in-cm", "class"]

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

# export the data as a csv file
iris_data.to_csv("cleaned_iris.csv")
```

Program Title: 2. Write a python program to import and export data using Pandas library functions

3. Demonstrate various data pre-processing techniques used for a given dataset. (IvR)

→ (1) Observation

- (i) Download the dataset
- (ii) Read the csv file with the help of pandas library

- `df = pd.read_csv("at.csv")`
- (iii) Observing dataset using `head()`, `info()`, and `describe()`
 - (iv) Visualization of dataset using `matplotlib`.
 - (v) Creating test set and training set by splitting the dataset.
- `train_set, test_set = train_test_split(housing, test_size=0.2, random_state=42)`
- (vi) Visualizing the data for insights.
 - (vii) Finding the correlation b/w the categories.
 - (viii) Data cleaning by dropping null values.
- `housing.dropna()`
- (ix) Imputation of missing values.
 - (x) Encoding categorical values with numbers.
- `imputer = SimpleImputer(strategy='median')`
- (xi) Scaling data by standardization.
- `scaler = StandardScaler(strategy='min-max')`
- (xii) Training the linear regression model.
 - (xiii) Calculating the root mean square error.
- `linear_regression = LinearRegression()`
- (xiv) Training decision tree.
 - (xv) Cross validating using mean and standard deviation.
- `cross_val_score(tree, housing, target)`
- (xvi) Fitting test dataset and calculating accuracy.

Program Title: 3. Demonstrate various data pre-processing techniques for a given dataset

Code

```
# import the pandas library
import pandas as pd

# Read the CSV file
airbnb_data = pd.read_csv("/content/sample_data/mnist_test.csv")

# View the first 5 rows
airbnb_data.head()

# Webpage 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()

# Export the file to the current working directory
iris_data.to_csv("cleaned_iris_data.csv")
```

Program Title: 2. Write a python program to import and export data using Pandas library functions

Program Title: 3. Demonstrate various data pre-processing techniques for a given dataset

Snapshot of the output

```
✓ 0s [5] import pandas as pd

✓ 1s ⏎ # Read the CSV file
airbnb_data = pd.read_csv("/content/sample_data/mnist_test.csv")

# View the first 5 rows
airbnb_data.head()



|   | 7 | 0 | 0.1 | 0.2 | 0.3 | 0.4 | 0.5 | 0.6 | 0.7 | 0.8 | ... | 0.658 | 0.659 | 0.660 | 0.661 | 0.662 | 0.663 | 0.664 | 0.665 | 0.666 | 0.667 |
|---|---|---|-----|-----|-----|-----|-----|-----|-----|-----|-----|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|
| 0 | 2 | 0 | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | ... | 0     | 0     | 0     | 0     | 0     | 0     | 0     | 0     | 0     | 0     |
| 1 | 1 | 0 | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | ... | 0     | 0     | 0     | 0     | 0     | 0     | 0     | 0     | 0     | 0     |
| 2 | 0 | 0 | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | ... | 0     | 0     | 0     | 0     | 0     | 0     | 0     | 0     | 0     | 0     |
| 3 | 4 | 0 | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | ... | 0     | 0     | 0     | 0     | 0     | 0     | 0     | 0     | 0     | 0     |
| 4 | 1 | 0 | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | ... | 0     | 0     | 0     | 0     | 0     | 0     | 0     | 0     | 0     | 0     |


5 rows × 785 columns
```

```
✓ 0s ⏎ # Webpage 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 |



Next steps:  View recommended plots



```
✓ 0s [8] # Export the file to the current working directory
iris_data.to_csv("cleaned_iris_data.csv")
```


```

Program Title: 2. Write a python program to import and export data using Pandas library functions

Program Title: 3. Demonstrate various data pre-processing techniques for a given dataset

Lab 2

Date: 12.04.2024

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

Algorithm (Handwritten)

Lab - 2
Date: 12/04/24
Page No.: 3

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

→ Algorithm :-

```
def ID3(D, A):
    if D is pure or A is empty:
        return a leaf node with the majority class
    in D
    else:
        A-best = argmax(InformationGain(D, A))
        root = Node(A-best)
        for r in values(A-best):
            D-r = subset(D, A-best, r)
            child = ID3(D-r, A - {A-best})
            root.add_child(r, child)
        return root.
```

Entropy = $- \sum_i P_i \log_2(P_i)$

P_i → fraction of samples within a particular node.

$$IG(A, D) = H(S) - \sum_r \frac{|S_r|}{|S|} * H(S_r)$$

S → Total instances
S_r → no. of instances for which attribute D has value r

Code and output

```
[ ] # import the pandas library
import pandas as pd

[ ] # Read the CSV file
airbnb_data = pd.read_csv("/content/sample_data/mnist_test.csv")

# View the first 5 rows
airbnb_data.head()

[ ] 7 0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 ... 0.658 0.659 0.660 0.661 0.662 0.663 0.664 0.665 0.666 0.667
[ ] 0 2 0 0 0 0 0 0 0 0 ... 0 0 0 0 0 0 0 0 0 0
[ ] 1 1 0 0 0 0 0 0 0 0 ... 0 0 0 0 0 0 0 0 0 0
[ ] 2 0 0 0 0 0 0 0 0 0 ... 0 0 0 0 0 0 0 0 0 0
[ ] 3 4 0 0 0 0 0 0 0 0 ... 0 0 0 0 0 0 0 0 0 0
[ ] 4 1 0 0 0 0 0 0 0 0 ... 0 0 0 0 0 0 0 0 0 0
5 rows × 785 columns

[ ] # Webpage 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

[ ] iris_data.info()

[ ] <class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 # Column          Non-Null Count  Dtype  
---  ...
 0  sepal_length_in_cm 150 non-null   float64 
 1  sepal_width_in_cm  150 non-null   float64 
 2  petal_length_in_cm 150 non-null   float64 
 3  petal_width_in_cm  150 non-null   float64 
 4  class            150 non-null   object  
dtypes: float64(4), object(1)
memory usage: 6.0+ KB

[ ] iris_data.describe()

[ ] sepal_length_in_cm sepal_width_in_cm petal_length_in_cm petal_width_in_cm
[ ] count    150.000000 150.000000 150.000000 150.000000
[ ] mean     5.843333 3.054000 3.758667 1.198667
[ ] std      0.828066 0.433594 1.764420 0.763161
[ ] min      4.300000 2.000000 1.000000 0.100000
[ ] 25%     5.100000 2.800000 1.600000 0.300000
[ ] 50%     5.800000 3.000000 4.350000 1.300000
[ ] 75%     6.400000 3.300000 5.100000 1.800000
[ ] max      7.900000 4.400000 6.900000 2.500000

[ ] iris_data.isnull().sum()

[ ] sepal_length_in_cm 0
sepal_width_in_cm 0
petal_length_in_cm 0
petal_width_in_cm 0
class 0
dtype: int64

[ ] data=iris_data.to_numpy()
dataset=data[:, :-1]
df = pd.DataFrame(dataset, index=dataset[:, 0])
df.kurt(axis=1)

[ ] 5.1 -2.368842
4.9 -1.091924
4.7 -2.276657
4.6 -1.57517
5.0 -2.787804
...
6.7 -2.983606
6.3 -3.799183
6.5 -3.127297
6.2 -3.365994
5.9 -3.345923
Length: 150, dtype: object

[ ] # Export the file to the current working directory
iris_data.to_csv("cleaned_iris_data.csv")
```

Lab 3

Date: 03.05.2024

Title: Implement Linear Regression algorithm using appropriate dataset

Algorithm (Handwritten)

Lab - 3

Date : 3/5/24
Page No : 1

Q:- Implement linear and multi-linear regression algorithm using appropriate dataset.

→ Algorithm :-

- ① Import necessary libraries
- ② Import dataset
- ③ Pre-process the data, convert or encode categorical data.
- ④ Split the dataset into training and testing set. from sklearn.model_selection, import train-test-split, X-train, X-test, y-train, y-test = train-test-split (X, y, test_size=0.3, random_state=23)
- ⑤ Build Model:
from sklearn.linear_model import LinearRegression
lin-reg = LinearRegression()
- ⑥ Fit dataset to model and train it
lin-reg.fit (X-train, y-train).
- ⑦ Calculate the accuracy using mean square error.

"/" to implement multi-linear regression

- ⑧ Encode categorical data
- ⑨ df = pd.get_dummies (df, columns = [c for c in df if c != 'Outcome'], dropfirst = True)
- ⑩ split dataset into training and testing set.

(i) We can see multiple independent variables

(ii) Create regression model

regressor = LinearRegression()

(iii) Fit train set

(iv) Test model using test set

(v) Compare actual and predicted values.

SPD

3/5/24

Code and output

```
# Import libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from pandas.core.common import random_state
from sklearn.linear_model import LinearRegression
```

[10] ✓ 0.0s

```
# Get dataset
df_sal = pd.read_csv('salary_data.csv')
df_sal.head()
```

[10] ✓ 0.0s

	YearsExperience	Salary
0	1.1	39343
1	1.3	46205
2	1.5	37731
3	2.0	43525
4	2.2	39891

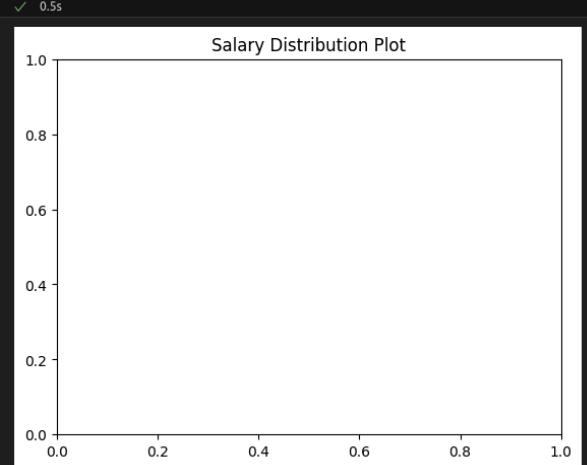
```
# Describe data
df_sal.describe()
```

[11] ✓ 0.1s

	YearsExperience	Salary
count	30.000000	30.000000
mean	5.313333	76003.000000
std	2.837888	27414.429785
min	1.100000	37731.000000
25%	3.200000	56720.750000
50%	4.700000	65237.000000
75%	7.700000	100544.750000
max	10.500000	122391.000000

```
# Data distribution
plt.title('Salary Distribution Plot')
sns.distplot(df_sal['Salary'])
plt.show()
```

[✓] 0.5s



A histogram titled "Salary Distribution Plot" showing the distribution of salaries. The x-axis ranges from 0.0 to 1.0, and the y-axis ranges from 0.0 to 1.0. The distribution is roughly uniform across the range.

```
# Relationship between Salary and Experience
plt.scatter(df_sal['YearsExperience'], df_sal['Salary'], color = 'lightcoral')
```

[✓] 0.1s



A scatter plot titled "Salary vs Experience" showing the relationship between years of experience and salary. The x-axis is labeled "Years of Experience" and ranges from 0 to 10. The y-axis is labeled "Salary" and ranges from 40,000 to 120,000. The data points show a positive correlation, with salary increasing as years of experience increase.

```
plt.title('Salary vs Experience')
plt.xlabel('Years of Experience')
plt.ylabel('Salary')
plt.box(False)
plt.show()
```

```
# Relationship between Salary and Experience
plt.scatter(df_sal['YearsExperience'], df_sal['Salary'], color = 'lightcoral')
plt.title('Salary vs Experience')
plt.xlabel('Years of Experience')
plt.ylabel('Salary')
plt.box(False)
plt.show()
```

[✓] 0.1s

```
# Splitting variables
X = df_sal.iloc[:, 1:] # independent
y = df_sal.iloc[:, 1:] # dependent
```

[✓] 0.0s

```
# Splitting dataset into test/train
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)
```

```

# Regressor model
regressor = LinearRegression()
regressor.fit(X_train, y_train)
✓ 0.2s
LinearRegression()
LinearRegression()

# Prediction result
y_pred_test = regressor.predict(X_test)      # predicted value of y_test
y_pred_train = regressor.predict(X_train)     # predicted value of y_train
✓ 0.0s
Python

# Prediction on training set
plt.scatter(X_train, y_train, color = 'lightcoral')
plt.plot(X_train, y_pred_train, color = 'firebrick')
plt.title('Salary vs Experience (Training Set)')
plt.xlabel('Years of Experience')
plt.ylabel('Salary')
plt.legend(['X_train/Pred(y_test)', 'X_train/y_train'], title = 'Sal/Exp', loc='best', facecolor='white')
plt.box(False)
plt.show()

```

```

# Prediction on test set
plt.scatter(X_test, y_test, color = 'lightcoral')
plt.plot(X_train, y_pred_train, color = 'firebrick')
plt.title('Salary vs Experience (Test Set)')
plt.xlabel('Years of Experience')
plt.ylabel('Salary')
plt.legend(['X_train/Pred(y_test)', 'X_train/y_train'], title = 'Sal/Exp', loc='best', facecolor='white')
plt.box(False)
plt.show()
✓ 0.0s

```

```

# Regressor coefficients and intercept
print(f'Coefficient: {regressor.coef_}')
print(f'Intercept: {regressor.intercept_}')
✓ 0.0s

```

```

Coefficient: [[9312.57512673]]
Intercept: [26788.09915063]

```

Title: Implement Multi-Linear Regression algorithm using appropriate dataset

Code and output

```
#Importing the libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# import warnings
import warnings
warnings.filterwarnings("ignore")

# We will use some methods from the sklearn module
from sklearn import linear_model
from sklearn.linear_model import LinearRegression
from sklearn import metrics
from sklearn.metrics import mean_squared_error, mean_absolute_error
from sklearn.model_selection import train_test_split, cross_val_score

[] # Reading the Dataset
df = pd.read_csv("data.csv")

df.head()

Car      Model Volume Weight CO2
0   Toyoly     Aygo    1000    790   99
1 Mitsubishi Space Star    1200   1100   95
2   Skoda     Citigo    1000    929   95
3     Fiat       500     900    805   90
4     Mini    Cooper    1500   1140  105

[] df.shape
(36, 5)

[] df.corr(numeric_only=True)

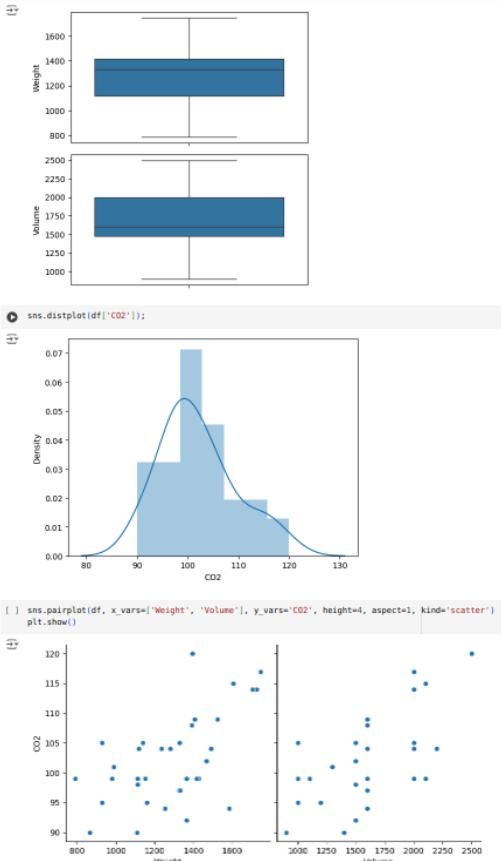
Volume  Weight   CO2
Volume  1.000000  0.753537  0.592082
Weight   0.753537  1.000000  0.552150
CO2     0.592082  0.552150  1.000000

[] print(df.describe())

   Volume      Weight      CO2
count  36.000000  36.000000  36.000000
mean  1611.111111 1292.277778 182.827778
std   388.975847  242.123889  7.454571
min   900.000000  790.000000  90.000000
25%  1475.000000 1117.250000  97.750000
50%  1600.000000 1329.000000  99.000000
75%  2000.000000 1418.250000 105.000000
max   2500.000000 1746.000000 120.000000

[] #Setting the value for X and Y
X = df[['Weight', 'Volume']]
y = df['CO2']

[] fig, axs = plt.subplots(2, figsize = (5,5))
plt1 = sns.boxplot(df['Weight'], ax = axs[0])
plt2 = sns.boxplot(df['Volume'], ax = axs[1])
plt.tight_layout()
```



```
[ ] # Create the correlation matrix and represent it as a heatmap.
sns.heatmap(df.corr(numeric_only=True), annot = True, cmap = 'coolwarm')
plt.show()
```

```
[ ] X_train,X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = 100)
```

```
[ ] y_train.shape
```

```
(25,)
```

```
[ ] y_test.shape
```

```
(11,)
```

```
reg_model = linear_model.LinearRegression()
```

```
[ ] #Fitting the Multiple Linear Regression model
reg_model = LinearRegression().fit(X_train, y_train)
```

```
[ ] #Printing the model coefficients
print('Intercept:', reg_model.intercept_)
# pair the feature names with the coefficients
list(zip(X, reg_model.coef_))
```

```
Intercept: 74.33882836589245
[('Weight', 0.0171800645996374), ('Volume', 0.0025046399866482976)]
```

```
[ ] #Predicting the Test and Train set result
y_pred= reg_model.predict(X_test)
x_pred= reg_model.predict(X_train)
```

```
[ ] print("Prediction for test set: {}".format(y_pred))
```

```
Prediction for test set: [ 98.41571939 102.16323413  99.56363213 104.56661845 101.54657652
 95.94770819 108.64011848 102.22654214  92.88374837  97.27327129
 97.57874463]
```

```
[ ] #Actual value and the predicted value
reg_model_diff = pd.DataFrame({'Actual value': y_test, 'Predicted value': y_pred})
reg_model_diff
```

	Actual value	Predicted value
0	99	90.415719
19	105	102.103234
32	104	99.563632
35	120	104.566618
7	92	101.546577
12	99	95.947700
29	114	108.040118
33	108	102.226542
5	105	92.883748
1	95	97.273271
18	104	97.578744

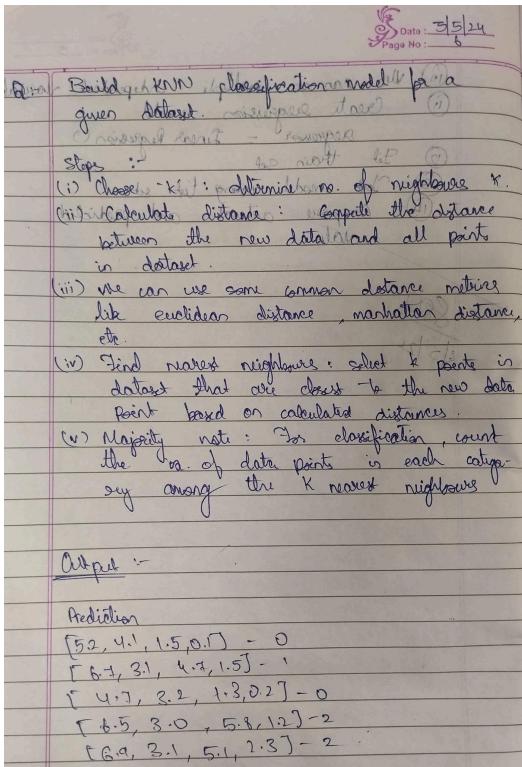
```
mae = metrics.mean_absolute_error(y_test, y_pred)
mse = metrics.mean_squared_error(y_test, y_pred)
r2 = np.sqrt(metrics.mean_squared_error(y_test, y_pred))

print('Mean Absolute Error:', mae)
print('Mean Square Error:', mse)
print('Root Mean Square Error:', r2)
```

```
Mean Absolute Error: 6.901980901636316
Mean Square Error: 63.39765310998794
Root Mean Square Error: 7.96226432053018
```

Title: Build KNN Classification model for a given dataset

Algorithm (Handwritten)



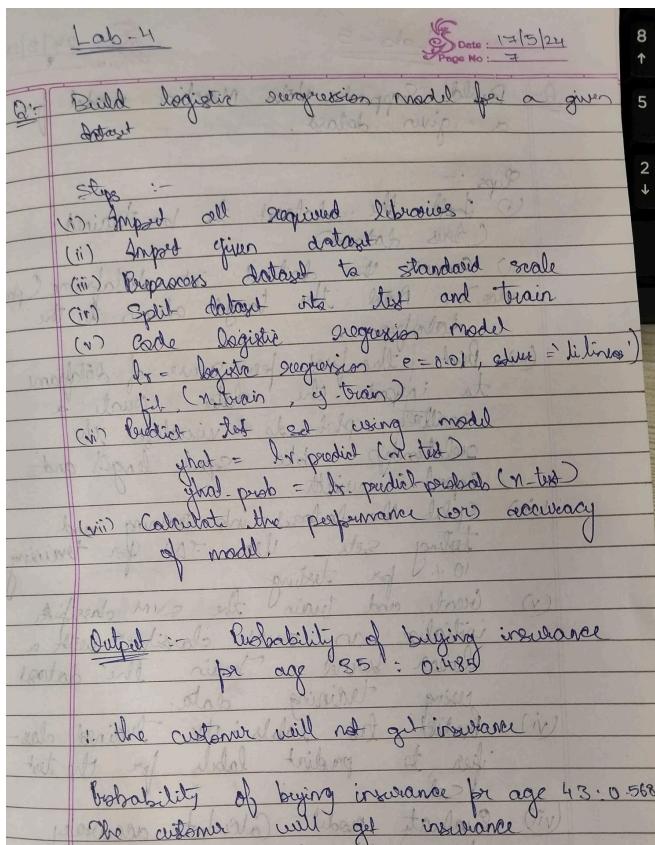
Code and output

Lab 4

Date: 17.05.2024

Title: Build Logistic Regression Model for a given dataset

Algorithm (Handwritten)



Code and output

```
In [1]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from plotly.offline import download_plotlyjs, init_notebook_mode, plot, iplot
import plotly.graph_objs as go
import time

init_notebook_mode(connected=True)
```

```
In [2]: def sigmoid(X, weight):
    z = np.dot(X, weight)
    return 1 / (1 + np.exp(-z))
```

```
In [3]: def loss(h, y):
    return -y * np.log(h) - (1 - y) * np.log(1 - h)).mean()
```

```
In [4]: def gradient_descent(X, h, y):
    return np.dot(X.T, (h - y)) / y.shape[0]
def update_weight_loss(weight, learning_rate, gradient):
    return weight - learning_rate * gradient
```

```
In [5]: def log_likelihood(x, y, weights):
    z = np.dot(x, weights)
    ll = np.sum(y*z - np.log(1 + np.exp(z)))
    return ll
```

```
In [6]: def gradient_ascent(X, h, y):
    return np.dot(X.T, y - h)
def update_weight_mle(weight, learning_rate, gradient):
    return weight + learning_rate * gradient
```

```
In [7]: data = pd.read_csv("./content/WA_Fn-UseC_-Telco-Customer-Churn.csv")
print("Dataset size")
print("Rows {} Columns {}".format(data.shape[0], data.shape[1]))
print("Columns and data types")
pd.DataFrame(data.dtypes).rename(columns = {0:'dtype'})
```

Dataset size
Rows 7043 Columns 21
Columns and data types

	dtype
customerID	object
gender	object
SeniorCitizen	int64
Partner	object
Dependents	object
tenure	int64
PhoneService	object
MultipleLines	object
InternetService	object
OnlineSecurity	object
OnlineBackup	object
DeviceProtection	object
TechSupport	object
StreamingTV	object
StreamingMovies	object
Contract	object
PaperlessBilling	object
PaymentMethod	object
MonthlyCharges	float64
TotalCharges	object
Churn	object

```
In [8]: df = data.copy()
churns = ["Yes", "No"]
fig = [
    {
        'x': df.loc[(df['Churn']==churn), 'MonthlyCharges'],
        'y': df.loc[(df['Churn']==churn), 'tenure'],
        'name': churn,
        'mode': 'markers',
    } for churn in churns
]
layout = {
    'title': 'Tenure vs Monthly Charges',
    'xaxis': {'title': 'Monthly Charges'},
    'yaxis': {'title': 'Tenure'}
}
py.offline.iplot(fig)
```

```
In [9]: figs = []
for churn in churns:
    figs.append(
        go.Box(
            y = df.loc[(df['Churn']==churn), 'tenure'],
            name = churn
        )
    )
layout = go.Layout(
    title = "Tenure",
    xaxis = {"title": "Churn?"},
    yaxis = {"title": "Tenure"},
    width=800,
    height=500
)
fig = go.Figure(data=figs, layout=layout)
py.offline.iplot(fig)
```

```
In [10]: figs = []
for churn in churns:
    figs.append(
        go.Box(
            y = df.loc[(df['Churn']==churn), 'MonthlyCharges'],
            name = churn
        )
    )
layout = go.Layout(
    title = "MonthlyCharges",
    xaxis = {"title": "Churn?"},
    yaxis = {"title": "MonthlyCharges"},
    width=800,
    height=500
)
fig = go.Figure(data=figs, layout=layout)
py.offline.iplot(fig)
```

```
In [11]: _ = df.groupby('Churn').size().reset_index()
#.sort_values(by='tenure', ascending=True)

data = [go.Bar(
    x = ['Churn'].tolist(),
    y = [0].tolist(),
    marker=dict(
        color=['rgba(255,199,134,1)', 'rgba(142,186,217,1)']
)]
layout = go.Layout(
    title = "Churn distribution",
    xaxis = {"title": "Churn?"},
    width=800,
    height=500
)
fig = go.Figure(data=data, layout=layout)
py.offline.iplot(fig)
```

```
In [12]: df['class'] = df['Churn'].apply(lambda x : 1 if x == "Yes" else 0)
# features will be saved as X and our target will be saved as y
X = df[['tenure', 'MonthlyCharges']].copy()
X2 = df[['tenure', 'MonthlyCharges']].copy()
y = df['class'].copy()
```

```
In [1]: start_time = time.time()
num_iter = 100000

intercept = np.ones((X.shape[0], 1))
X = np.concatenate((intercept, X), axis=1)
theta = np.zeros(X.shape[1])

for i in range(num_iter):
    h = sigmoid(X, theta)
    gradient = gradient_descent(X, h, y)
    theta = update_weight(theta, 0.1, gradient)

print("Training time (Log Reg using Gradient descent): " + str(time.time() - start_time) + " seconds")
print("Learning rate: {} \nIteration: {}".format(0.1, num_iter))

Training time (Log Reg using Gradient descent):70.8485119342804 seconds
Learning rate: 0.1
Iteration: 100000
```

```
In [2]: result = sigmoid(X, theta)
```

```
In [3]: f = pd.DataFrame(np.around(result, decimals=6)).join(y)
f['pred'] = f[0].apply(lambda x : 0.1 if x < 0.5 else 1)
print("Accuracy (Loss minimization):")
f.loc[f['pred'] == f['class']].shape[0] / f.shape[0] * 100
```

```
Accuracy (Loss minimization):
```

```
Out[3]: 53.301150078891716
```

```
In [4]: start_time = time.time()
num_iter = 100000

intercept2 = np.ones((X2.shape[0], 1))
X2 = np.concatenate((intercept2, X2), axis=1)
theta2 = np.zeros(X2.shape[1])

for i in range(num_iter):
    h2 = sigmoid(X2, theta2)
    gradient2 = gradient_descent(X2, h2, y) #np.dot(X.T, (h - y)) / y.size
    theta2 = update_weight(theta2, 0.1, gradient2)

print("Training time (Log Reg using MLE): " + str(time.time() - start_time) + " seconds")
print("Learning rate: {} \nIteration: {}".format(0.1, num_iter))

<iPython>-input-2-2eaaa9337b29>:3: RuntimeWarning:
overflow encountered in exp
```

```
Training time (Log Reg using MLE):81.35162234306335 seconds
Learning rate: 0.1
Iteration: 100000
```

```
In [5]: result2 = sigmoid(X2, theta2)
```

```
<iPython>-input-2-2eaaa9337b29>:3: RuntimeWarning:
overflow encountered in exp
```

```
In [6]: from sklearn.linear_model import LogisticRegression

clf = LogisticRegression(fit_intercept=True, max_iter=100000)
clf.fit(df[['tenure', 'MonthlyCharges']], y)
print("Training time (sklearn's LogisticRegression module): " + str(time.time() - start_time) + " seconds")
print("Learning rate: {} \nIteration: {}".format(0.1, num_iter))

Training time (sklearn's LogisticRegression module):83.02515387535095 seconds
Learning rate: 0.1
Iteration: 100000
```

```
In [7]: result3 = clf.predict(df[['tenure', 'MonthlyCharges']])
```

```
In [8]: print("Accuracy (sklearn's Logistic Regression):")
f3 = pd.DataFrame(result3).join(y)
f3.loc[f3[0]==f3['class']].shape[0] / f3.shape[0] * 100
```

```
Accuracy (sklearn's Logistic Regression):
```

```
Out[8]: 78.44668465142695
```

Lab 5

Date: 24.05.2024

Title: Build Support vector machine model for a given dataset

Algorithm (Handwritten)

Labs - 5

Date : 24/5/24
Page No : 8

Q:- Build a Support Vector Machine model for a given dataset.

Tips :-

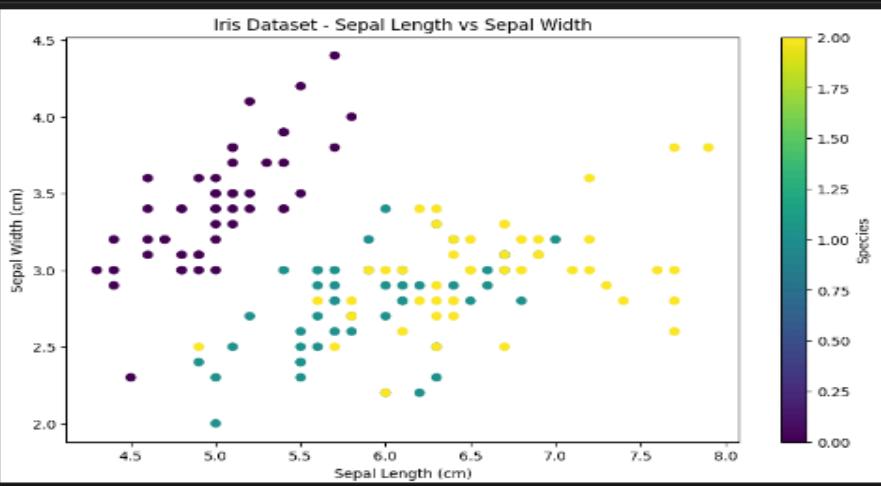
- (i) Load the dataset to be trained (iris dataset).
- (ii) Convert the dataset into Dataframe (pandas). Add the target column to the Dataframe.
- (iii) Print the first few rows of Dataframe to inspect the data. Create a scatter plot to visualize the relationship between Sepal Length and Sepal Width.
- (iv) Split the dataset into training and testing sets. Use 80% for training and 20% for testing.
- (v) Create and train the SVM classifier. Initialize an SVM classifier with a linear kernel. Train the dataset using training data.
- (vi) Predict test labels: Use Trained classifier to predict labels for the test set.
- (vii) Evaluate Model: Calculate accuracy of model by comparison prediction labels with actual test labels.

Output :- Accuracy of SVM classifier is 1.0

Date : 24/5/24
Page No : 9

y-pred is array ([1, 0, 2, 1, 0, 1, 2, 1, 1, 2, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0])

Code and output

```
Code | Markdown | Run All | Restart | Clear All Outputs | Variables | Outline | ...  
import pandas as pd  
import matplotlib.pyplot as plt  
from sklearn.datasets import load_iris  
from sklearn.model_selection import train_test_split  
from sklearn.svm import SVC  
from sklearn.metrics import accuracy_score  
0.5s  
  
# Load the Iris dataset  
iris = load_iris()  
0.0s  
  
# Convert the dataset into a pandas DataFrame  
iris_df = pd.DataFrame(data=iris.data, columns=iris.feature_names)  
iris_df['target'] = iris.target  
0.0s  
  
# Display the first few rows of the DataFrame  
print(iris_df.head())  
0.0s  
sepal length (cm) sepal width (cm) petal length (cm) petal width (cm) \\\n0 5.1 3.5 1.4 0.2  
1 4.9 3.0 1.4 0.2  
2 4.7 3.2 1.3 0.2  
3 4.6 3.1 1.5 0.2  
4 5.0 3.6 1.4 0.2  
  
target  
0 0  
1 0  
2 0  
3 0  
4 0  
  
# Plotting to visualize the data  
plt.figure(figsize=(10, 6))  
plt.scatter(iris_df['sepal length (cm)'], iris_df['sepal width (cm)'],  
           c=iris_df['target'], cmap='viridis')  
plt.xlabel('Sepal Length (cm)')  
plt.ylabel('Sepal Width (cm)')  
plt.title('Iris Dataset - Sepal Length vs Sepal Width')  
plt.colorbar(label='Species')  
plt.show()  
0.1s  

```

```

# Splitting the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(iris.data, iris.target, test_size=0.3, random_state=42)
✓ 0.0s

# Creating and training the SVM classifier
svm_classifier = SVC(kernel='linear')
svm_classifier.fit(X_train, y_train)

# Predicting the labels for the test set
y_pred = svm_classifier.predict(X_test)

# Calculating the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy of SVM Classifier:", accuracy)
✓ 0.0s

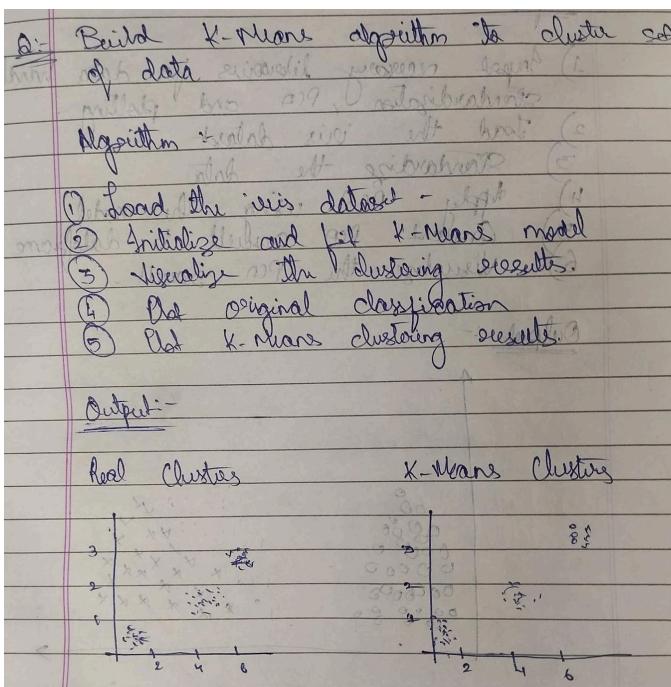
Accuracy of SVM Classifier: 1.0

y_pred
✓ 0.0s
array([1, 0, 2, 1, 1, 0, 1, 2, 1, 1, 2, 0, 0, 0, 0, 1, 2, 1, 1, 2, 0, 2,
       0, 2, 2, 2, 2, 0, 0, 0, 1, 0, 0, 2, 1, 0, 0, 0, 2, 1, 1, 0,
       0])

```

Title: Build K-Means algorithm to cluster a set of data stored in a .CSV file

Algorithm (Handwritten)



Code and output

```
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.cluster import KMeans
import pandas as pd
import numpy as np

# Import some data to play with
iris = datasets.load_iris()
X = pd.DataFrame(iris.data)
X.columns = ['Sepal_Length', 'Sepal_Width', 'Petal_Length', 'Petal_Width']
y = pd.DataFrame(iris.target)
y.columns = ['Targets']

# Build the K Means Model
model = KMeans(n_clusters=3)
model.fit(X) # model.labels_ : Gives cluster no for which samples belongs

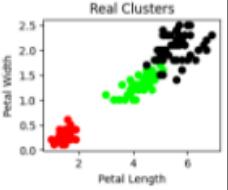
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:878: FutureWarning:
warnings.warn(
    * KMeans
KMeans(n_clusters=3)

# Visualise the clustering results
plt.figure(figsize=(14,14))
colormap = np.array(['red', 'lime', 'black'])

<Figure size 1400x1400 with 0 Axes>

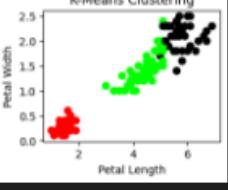
# Plot the Original Classifications using Petal features
plt.subplot(2, 2, 1)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[y.Targets], s=40)
plt.title('Real Clusters')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')

Text(0, 0.5, 'Petal Width')


```
Plot the Models Classifications
plt.subplot(2, 2, 2)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[model.labels_], s=40)
plt.title('K-Means Clustering')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')

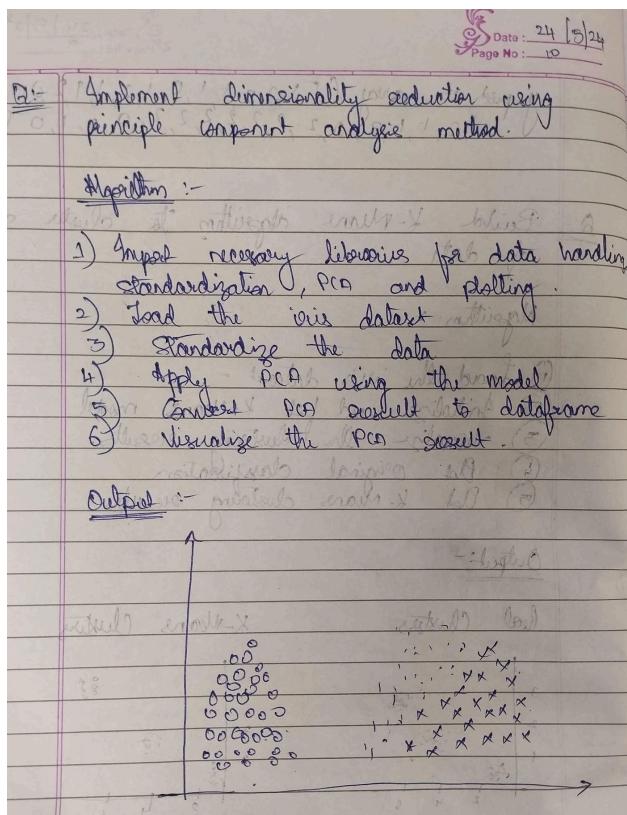
Text(0, 0.5, 'Petal Width')


```


```

Title: Implement Dimensionality reduction using Principle Component Analysis (PCA)
Method

Algorithm (Handwritten)



Code and output



Lab 6

Date: 31.05.2024

Title: Build Artificial Neural Network model with back propagation on a given dataset. Refer:-
<https://docs.google.com/presentation/d/11UE61G27eOAYnhe8ctHAqoEaeYLrVhoT/edit?usp=sharing&ouid=117926028109390959744&rtpof=true&sd=true>

Algorithm (Handwritten)

Lab - 6	
	Date : 31/5/24 Page No. : 12
Q1) Build a feed forward neural network model with back propagation.	
Algorithm :-	
1) Initialize parameters	
- Normalize input feature matrix X , - Normalize the output y , - Set hyperparameters (no. of epochs, no. of neurons in each layer, learning rate).	
2) Define activation functions	
- sigmoid function ($\sigma(x) = \frac{1}{1 + e^{-x}}$)	
3) Training the network	
- Forward propagation of test data	
+ Compute i/p to hidden layer 1 (add bias)	
+ apply activation function (sigmoid)	
4) Back propagation	
+ compute error	
+ compute gradient (backward pass)	
+ compute delta	
5) Update weights and biases	
Output :- I/p	$\begin{bmatrix} [0.667 & 1] \\ [0.333 & 0.667] \\ [0.1 & 0.667] \end{bmatrix}$
Actual Output :-	$[[0.92] [0.86] [0.87]]$

Lab - 6	
	Date : 31/5/24 Page No. : 12
Predicted Output :-	$\begin{bmatrix} [0.80056175] \\ [0.79392837] \\ [0.8011234] \end{bmatrix}$

Code and output

```
import numpy as np
x = np.array(([2,9],[1,5],[3,6]),dtype = float)
y = np.array(([92],[86],[89]),dtype = float)
x = x/np.amax(x,axis=0)
y = y/100

#Variable Initialization
epoch = 5000
lr = 0.1
inputlayer_neurons = 2
hiddenlayer_neurons = 3
output_neurons = 1

# weight and bias Initialization
wh = np.random.uniform(size=(inputlayer_neurons,hiddenlayer_neurons))
bh = np.random.uniform(size=(1,hiddenlayer_neurons))
wout = np.random.uniform(size=(hiddenlayer_neurons,output_neurons))
bout = np.random.uniform(size=(1,output_neurons))

#sigmoid function
def sigmoid(x):
| return 1/(1+np.exp(-x))

# Derivative of Sigmoid
def der_sigmoid(x):
| return x*(1-x)

# Draws a random range of numbers uniformly of dim x*y
for i in range(epoch):

    # forward propagation
    hinpl = np.dot(x,wh)
    hinp = hinpl + bh
    hlayer_act = sigmoid(hinp)
    outinpl = np.dot(hlayer_act,wout)
    outinp = outinpl + bout
    output = sigmoid(outinp)

    # Backpropagation
    EO = y - output
    outgrad = der_sigmoid(output)
    d_output = EO*outgrad
    EH = d_output.dot(wout.T)

    # how much hidden layer weights contributed to error
    hiddengrad = der_sigmoid(hlayer_act)
    d_hiddenlayer = EH*hiddengrad

    #dotproduct of nextlayererror and current layer op
    wout += hlayer_act.T.dot(d_output)*lr
    wh += x.T.dot(d_hiddenlayer)*lr

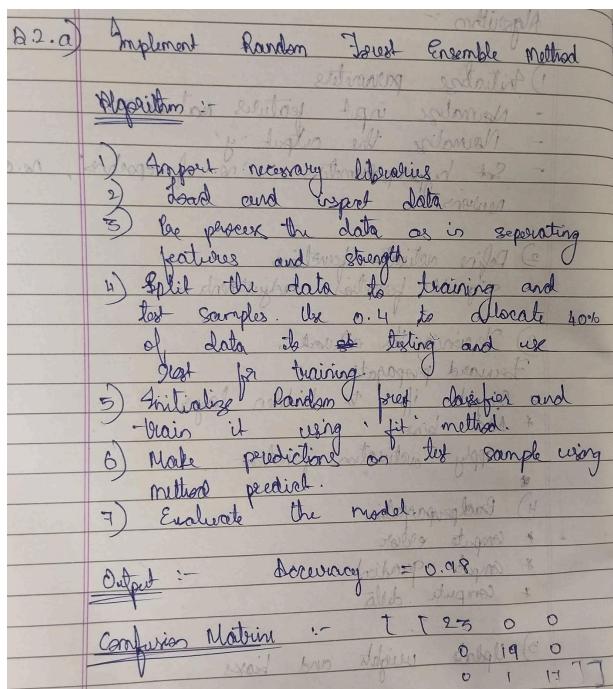
    print("Input: " + str(x))
    print("Actual output: " + str(y))
    print("Predicted Output: \n",output)
```

Input:
[[0.66666667 1.]
 [0.33333333 0.55555556]
 [1. 0.66666667]]
Actual output:
[[0.92]
 [0.86]
 [0.89]]
Predicted Output:
[[0.80056875]
 [0.79393831]
 [0.80112347]]

Title: Implement Random forest ensemble method on a given dataset.

Ref- <https://towardsdatascience.com/random-forest-in-python-24d0893d51c0>

Algorithm (Handwritten)



Code and output

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn import datasets

# Load the data
iris_data = datasets.load_iris()

X = pd.DataFrame(iris_data.data, columns=['Sepal_Length', 'Sepal_Width', 'Petal_Length', 'Petal_Width'])
y = pd.DataFrame(iris_data.target, columns=['Targets'])

# Check the info of the modified data
# print(iris_data.info())

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4, random_state=42)

# Initialize the Random Forest classifier
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)

# Fit the classifier to the training data
rf_classifier.fit(X_train, y_train)

# Predict on the test data
y_pred = rf_classifier.predict(X_test)

# Evaluate the classifier
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")

# Print classification report
print("Classification Report:")
print(classification_report(y_test, y_pred))

# Print confusion matrix
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

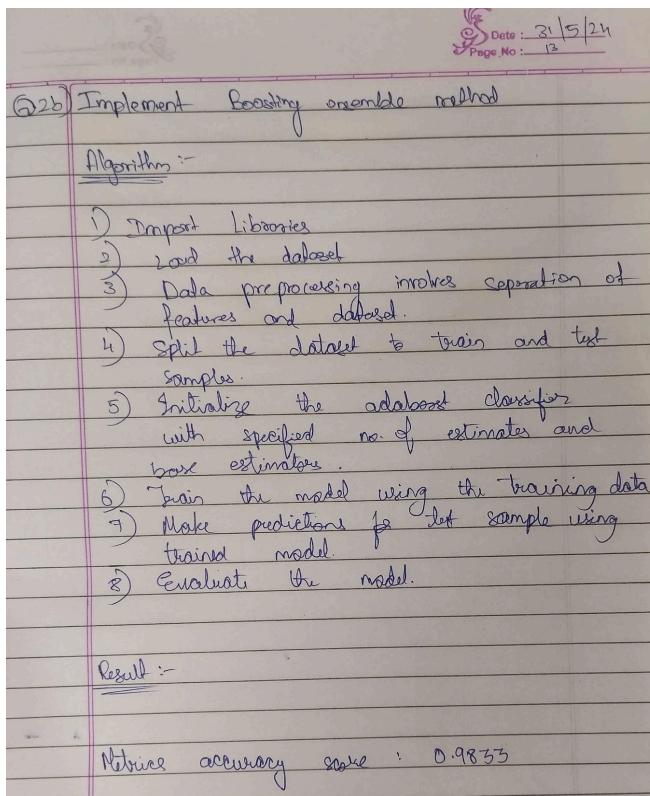
Accuracy: 0.98
Classification Report:
precision    recall  f1-score   support
          0       1.00      1.00      1.00      23
          1       0.95      1.00      0.97      19
          2       1.00      0.94      0.97      18

    accuracy         0.98      0.98      0.98      60
   macro avg       0.98      0.98      0.98      60
weighted avg       0.98      0.98      0.98      60

Confusion Matrix:
[[23  0  0]
 [ 0 19  0]
 [ 0  1 17]]
```

Title: Implement Boosting ensemble method on a given dataset

Algorithm (Handwritten)



Code and output

```
✓ [10] from sklearn.linear_model import LogisticRegression
      from sklearn.ensemble import AdaBoostClassifier
      from sklearn import metrics
      from sklearn import datasets

✓ [11] import pandas as pd
      import matplotlib.pyplot as plt
      from sklearn.model_selection import train_test_split

✓ [12] # Load the iris dataset
      iris = datasets.load_iris()
      X = pd.DataFrame(iris.data, columns=['Sepal_Length', 'Sepal_Width', 'Petal_Length', 'Petal_Width'])
      y = pd.DataFrame(iris.target, columns=['Targets'])

✓ [13] X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.4,random_state=42)

✓ [14] mylogregmodel = LogisticRegression()

✓ [15] adabc = AdaBoostClassifier(n_estimators = 150, estimator = mylogregmodel, learning_rate = 1)

✓ [2a] ⏎ model = adabc.fit(X_train, y_train)
      ↗ /usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:1143: DataConversionWarning: A
          y = column_or_1d(y, warn=True)

✓ [17] y_pred = model.predict(X_test)

✓ [18] metrics.accuracy_score(y_test, y_pred)
      ↗ 0.9833333333333333
```