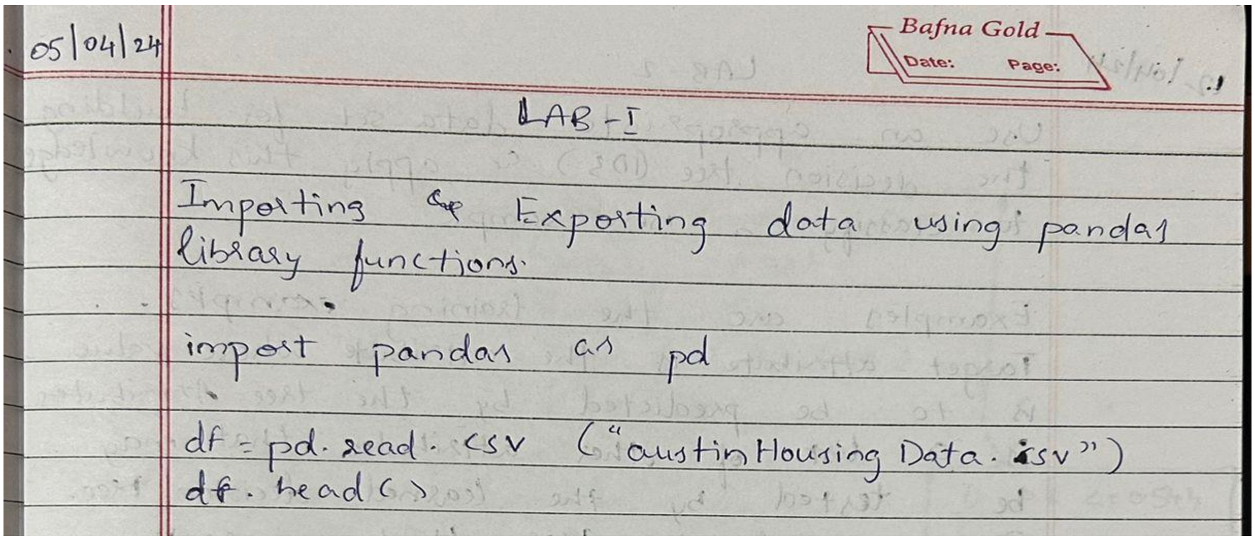
Lab1

Date: 5^{th} April , 2024

Observation

Program Title: 1. Write a python program to import and export data using Pandas library functions



05|04|24

Bafna Gold

Date: Page:

LAB - I

Importing & Exporting data using pandas library functions.

import pandas as pd

df = pd. read. csv ("austinHousing Data. csv")

df. head ()

Program Title:1. Write a python program to import and export data using Pandas library Functions

```
# 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. 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. Demonstrate various data pre-processing techniques for a given dataset

* Reading data from URL

url = "https:// arch.uv.icsivci .edu/me/
machinc 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)
iris _ data. head ()

* Exporting to another csv file
iris_ data, to _csv ("cleaned- iris_
data . csv")

Snapshot of the output

```
[5] 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	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

```
# 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

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

Lab2

Date: 12th April , 2024

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

Algorithm

12/04/24

LAB - 2

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

Examples are the training examples. Target - attribute is the attribute whose value is to be predicted by the tree. Attributes is a list of other attributes that may be tested by the learned decision tree. Returns a decision tree that correctly classifies the given examples.

* Create a root node for the tree
* If all examples are positive, return the single - node tree root, with label = most common value of Target - attribute in examples.

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	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

```
# 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   -2.787004
         ...
6.7   -2.983606
6.3   -3.790103
6.5   -3.127297
6.2   -3.387994
5.9   -3.345923
Length: 150, dtype: object
```

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

Lab3

Date: 3rd May , 2024

Title: Implement Linear Regression algorithm using appropriate dataset

Algorithm

03/05/24

LAB-3

Implement Linear regression algorithm using appropriate dataset.

i, Import necessary libraries
ii, Import dataset
iii, Visualization of dataset using different plots like heatmap, distribution plot etc.
iv, Preprocess the data, convert on encode categorical data
v, Split the dataset into training and testing set. from Sklearn.
vi, Build model.
vii, Fit dataset model & train it Linreg-fix(X-train, y-train).
viii, Calculate the accuracy using mean square error.

Code and output

```python
# 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
```

[9] ✓ 0.0s

```python
# 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

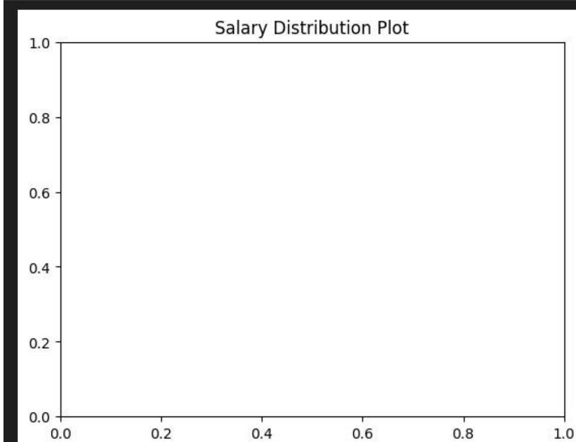
```python
# Describe data
df_sal.describe()
```

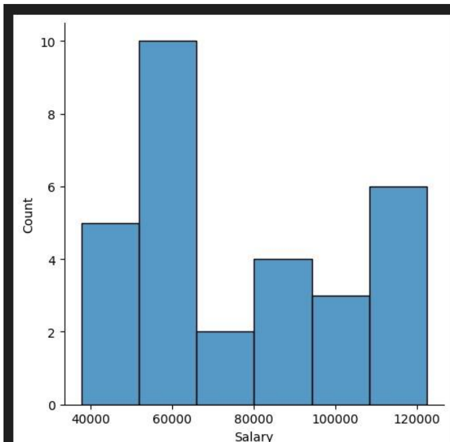
[11] ✓ 0.1s

	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

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

✓ 0.5s






```python
# 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()
```

```python
# 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 Python


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

✓ 0.0s Python

```python
# 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)
```

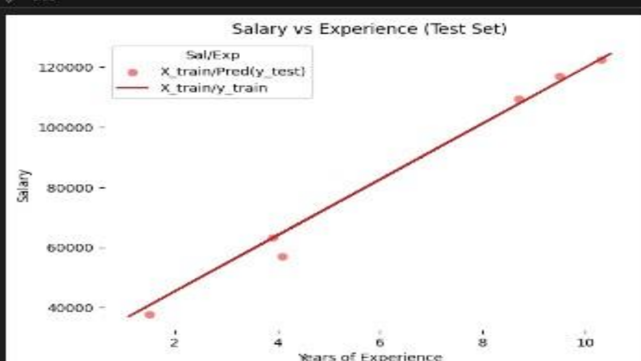
```python
# Regressor model
regressor = LinearRegression()
regressor.fit(X_train, y_train)
```
0.2s Python

```
LinearRegression
LinearRegression()
```

```python
# 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

```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', facecc
plt.box(False)
plt.show()
```


```python
# 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




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

```
Coefficient: [[9312.57512673]]
Intercept: [26780.09915063]
```

Title: Implement Multi-Linear Regression algorithm using appropriate dataset

Algorithm

Bafna Gold

Date: Page:

b) Implement Multi-Linear Regression.

~~Algorithm:~~

1) Import the required python package
2) Load the dataset
3) Do the data analysis
4) Split the dataset into dependent / independent variables.
5) One-slot encoding of categorical data.
   * It is a method to represent a categorical variable in a numerical way. ~~variable~~
6) Split dates into train/test sets
7) Train the regression model
8) Predict the results.

Code and output

```python
#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
```

```python
# Reading the Dataset
df = pd.read_csv("data.csv")
```

```python
df.head()
```

	Car	Model	Volume	Weight	CO2
0	Toyoty	Aygo	1000	790	99
1	Mitsubishi	Space Star	1200	1160	95
2	Skoda	Citigo	1000	929	95
3	Fiat	500	900	865	90
4	Mini	Cooper	1500	1140	105

```python
df.shape
```

```
(36, 5)
```

```python
df.corr(numeric_only=True)
```

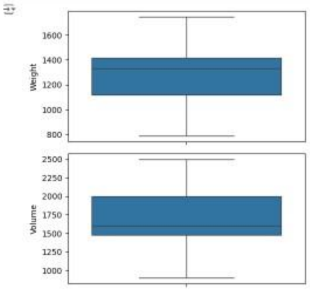
	Volume	Weight	CO2
Volume	1.000000	0.753537	0.592082
Weight	0.753537	1.000000	0.552150
CO2	0.592082	0.552150	1.000000

```python
print(df.describe())
```

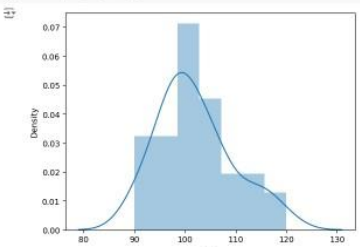
```
            Volume       Weight         CO2
count    36.000000    36.000000   36.000000
mean   1611.111111  1292.277778  102.027778
std     388.975047   242.123889    7.454571
min     900.000000   790.000000   90.000000
25%    1475.000000  1117.250000   97.750000
50%    1600.000000  1329.000000   99.000000
75%    2000.000000  1418.250000  105.000000
max    2500.000000  1746.000000  120.000000
```

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

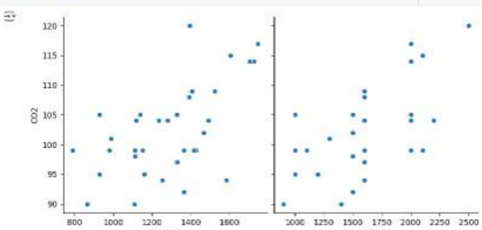
```python
fig, axs = plt.subplots(2, figsize = (5,5))
plt1 = sns.boxplot(df['Weight'], ax = axs[0])
plt2 = sns.boxplot(df['Volume'], ax = axs[1])
plt.tight_layout()
```



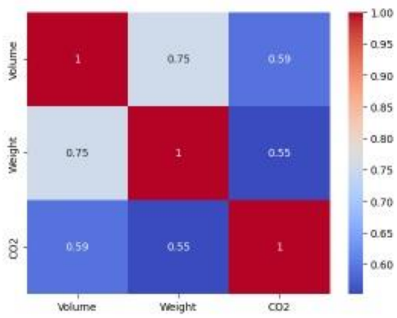

```python
sns.distplot(df['CO2']);
```




```python
sns.pairplot(df, x_vars=['Weight', 'Volume'], y_vars='CO2', height=4, aspect=1, kind='scatter')
plt.show()
```



```
[ ] # 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', 0.0171880645996374), ('Volume', 0.0025846399866482976)]
```

```
[ ] #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: [ 90.41571939 102.16323413  99.56363213 104.56661845 101.54657652
  95.94770019 108.64011848 102.22654214  92.80374837  97.27327129
  97.57074463]
```

```
[ ] #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	102.163234
32	104	99.563632
35	120	104.566618
7	92	101.546577
12	99	95.947700
29	114	108.640118
33	108	102.226542
5	105	92.803748
1	95	97.273271
18	104	97.570745

```
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

c) Implementation of KNN - Algorithm.

1) ~~Improving~~ Import the modules
2) Creating Dataset
3) Visualise the Dataset
4) Splitting data into training & testing datasets
5) KNN classify implementation
6) Prediction the KNN Classifiers
7) Predict Accuracy for both 'K' values

OUTPUT:

Accuracy with k = 5 => 93.600001

Accuracy with k = 1 => 90.4

Code and output

```python
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, confusion_matrix
from sklearn import datasets
iris = datasets.load_iris()

x = iris.data
y = iris.target

print('sepal-length', 'sepal-width', 'petal-length', 'petal-width')
print(x)
print('class: 0 - Iris-Setosa, 1 - Iris-Versicolour, 2 - Iris-Virginica')
print(y)
```

✓ 19.3s

```
sepal-length sepal-width petal-length petal-width
[[5.1 3.5 1.4 0.2]
 [4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5.  3.6 1.4 0.2]
 [5.4 3.9 1.7 0.4]
 [4.6 3.4 1.4 0.3]
 [5.  3.4 1.5 0.2]
 [4.4 2.9 1.4 0.2]
 [4.9 3.1 1.5 0.1]
 [5.4 3.7 1.5 0.2]
 [4.8 3.4 1.6 0.2]
 [4.8 3.  1.4 0.1]
 [4.3 3.  1.1 0.1]
 [5.8 4.  1.2 0.2]
 [5.7 4.4 1.5 0.4]
 [5.4 3.9 1.3 0.4]
 [5.1 3.5 1.4 0.3]
 [5.7 3.8 1.7 0.3]
 [5.1 3.8 1.5 0.3]
 [5.4 3.4 1.7 0.2]
 [5.1 3.7 1.5 0.4]
 [4.6 3.6 1.  0.2]
 [5.1 3.3 1.7 0.5]
...
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
```

```python
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.3)

#To Training the model and Nearest nighbors K=5
classifier = KNeighborsClassifier(n_neighbors=5)
classifier.fit(x_train, y_train)

#to make predictions on our test data
y_pred=classifier.predict(x_test)

print('Prediction -')

for i,test in enumerate(x_test) :
  print(f'{test} - {y_pred[i]}')

# print('Confusion Matrix')
# print(confusion_matrix(y_test,y_pred))
# print('Accuracy Metrics')
# print(classification_report(y_test,y_pred))
```

```
Prediction -
[5.2 4.1 1.5 0.1] - 0
[5.5 2.3 4.  1.3] - 1
[6.7 3.1 4.7 1.5] - 1
[7.  3.2 4.7 1.4] - 1
[6.2 2.8 4.8 1.8] - 2
[5.7 2.8 4.5 1.3] - 1
[6.  3.4 4.5 1.6] - 1
[5.1 3.8 1.6 0.2] - 0
[5.5 2.5 4.  1.3] - 1
[4.8 3.1 1.6 0.2] - 0
[6.1 3.  4.9 1.8] - 2
[4.7 3.2 1.6 0.2] - 0
[5.6 2.9 3.6 1.3] - 1
[5.4 3.9 1.3 0.4] - 0
[5.  3.2 1.2 0.2] - 0
[6.1 2.9 4.7 1.4] - 1
[5.  3.4 1.5 0.2] - 0
[7.7 2.8 6.7 2. ] - 2
[4.6 3.2 1.4 0.2] - 0
[5.7 2.9 4.2 1.3] - 1
[4.6 3.6 1.  0.2] - 0
[6.8 2.8 4.8 1.4] - 1
[6.8 3.2 5.9 2.3] - 2
[6.2 2.2 4.5 1.5] - 1
...
[5.2 3.4 1.4 0.2] - 0
[6.1 2.8 4.7 1.2] - 1
[6.5 3.  5.2 2. ] - 2
[6.5 3.  5.8 2.2] - 2
```

Lab4

Date: 17th May , 2024

Title: Build Logistic Regression Model for a given dataset

Algorithm

Bafna Gold
Date: Page:

03/05/24

LAB-4

* Build Logistic Regression Model

1) Import required libraries
2) Load, visualize and explore the dataset
3) Clean the dataset
4) Deal with the outliers
5) Define dependent & independent variables and then split the data into a training set and testing set.

6)

OUTPUT

Regression coefficients obtained are $b_0 = 68.83$

$b_1 = 0.192671$

Code and output

In [ ]:

```python
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 as py
import plotly.graph_objs as go
import time

init_notebook_mode(connected=True)
```

In [ ]:

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

In [ ]:

```python
def loss(h, y):
    return (-y * np.log(h) - (1 - y) * np.log(1 - h)).mean()
```

In [ ]:

```python
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 [ ]:

```python
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 [ ]:

```python
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 [ ]:

```python
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
```

Out[ ]:

	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 [ ]:

```python
df = data.copy()
```

In [ ]:

```python
churns = ["Yes", "No"]
fig = {
    'data': [
        {
            '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 [ ]:

```python
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 [ ]:

```python
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 [ ]:

```python
_ = 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(255,190,134,1)', '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 [ ]:

```python
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 [ ]:

```
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_loss(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):78.8485119342804 seconds
Learning rate: 0.1
Iteration: 100000
```

In [ ]:

```
result = sigmoid(X, theta)
```

In [ ]:

```
f = pd.DataFrame(np.around(result, decimals=6)).join(y)
f['pred'] = f[0].apply(lambda x : 0 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[ ]:

```
53.301150078091716
```

In [ ]:

```
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_ascent(X2, h2, y) #np.dot(X.T, (h - y)) / y.size
    theta2 = update_weight_mle(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-2-2eeea9337b29>:3: RuntimeWarning:

overflow encountered in exp
```

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

In [ ]:

```
result2 = sigmoid(X2, theta2)
```

```
<ipython-input-2-2eeea9337b29>:3: RuntimeWarning:

overflow encountered in exp
```

In [ ]:

```
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 [ ]:

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

In [ ]:

```
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[ ]:

```
78.44668465142695
```

Lab5

Date: 24th May,2024

Title: Build Support vector machine model for a given dataset

Algorithm (Handwritten)

24/5/24 LAB5

i) Support Vector Machine

1) Define Kernel function
Eg: $K(x_1, x_2) = x_1 \cdot x_2$

2) Solve the quadratic programming problem (QP) to find the α

3) Compute the bias

4) Identify the support vectors

5) Make prediction

OUTPUT

```
Model = SVM()
Model.fit (X_train, Y_train)
prediction = Model.predict (x_test)
accuracy = (ytext, prediction)

0.98230088

→ Model predict ([-0.47096, -0.1604584, -0.4481 -- --
-0.244122, -0.19956318, 0.1832441,
-0.1969574])
```

$\boxed{\text{array (0)}}$

Code and output

```
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

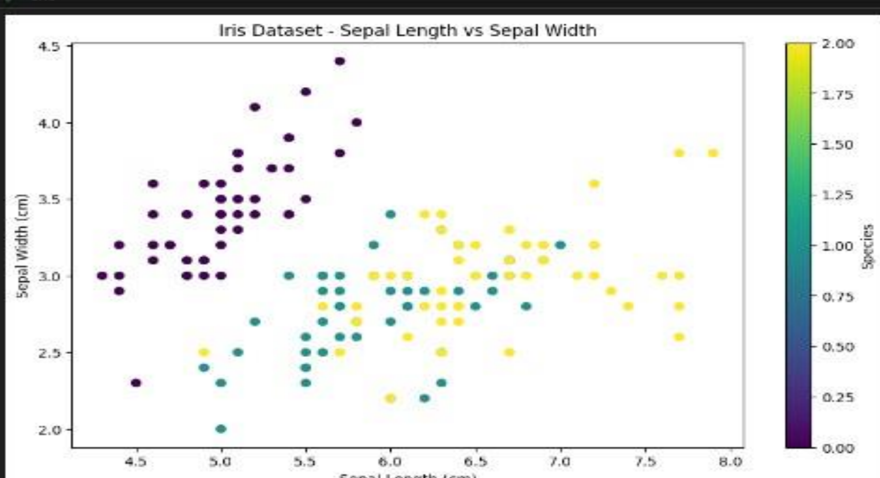
```
# Load the Iris dataset
iris = load_iris()
```
0.0s

```
# 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)  \
0                5.1               3.5                1.4               0.2
1                4.9               3.0                1.4               0.2
2                4.7               3.2                1.3               0.2
3                4.6               3.1                1.5               0.2
4                5.0               3.6                1.4               0.2

   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



```python
# 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)
```

```python
# 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)
```

```
Accuracy of SVM Classifier: 1.0
```

```python
y_pred
```

```
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, 2, 0, 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

2) K means clustering Algorithm

1) Select number K decide the no. of elements.

2) Select random k points or centroids.

3) Assign each point to the nearest centroid, which will form the predifined cluster

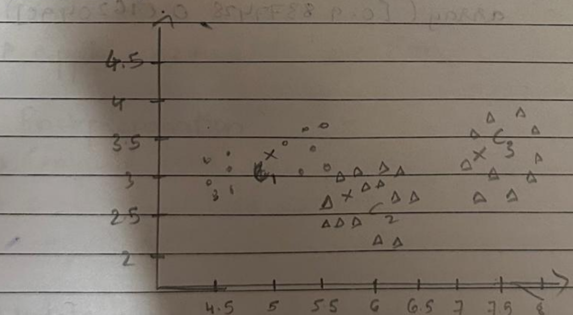
4) Calculate the variance and place a new centroid of each cluster.

5) Repeat step 3, reassign the centroid

6) If any rearrangement occurs perform step 4 else finish

7) The model is ready

OUTPUT:



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:870: FutureWarni
  warnings.warn(
```

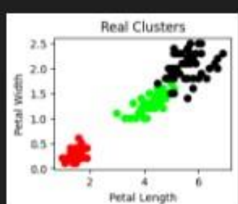
```
▾ KMeans
KMeans(n_clusters=3)
```

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

```
<Figure size 1400x1400 with 0 Axes>
```

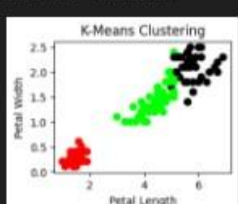
```
# Plot the Original Classifications using Petal features
plt.subplot(2, 2, 1)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[y.Targets], s=40)
plt.title('Real Clusters')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')
```

```
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)

3) PCA - Principle Component Analysis

1) Calculate the mean
2) Calculation of convinience matrix
3) Eigen value of the convinience matrix.
4) Computation of the eigen vecter - unit eigenvect of
5) Calculation of first principle component
6) Geometrical measuring of first principle component.

OUTPUT:

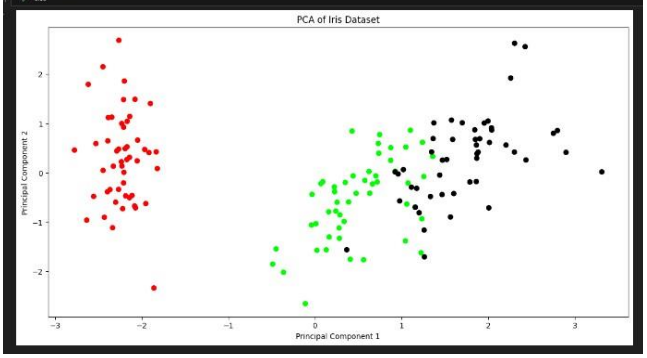
pca.explained variance ratio.

array([0.98377428, 0.01620498])

Code and output

```
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
import pandas as pd
import numpy as np

# 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'])

# Standardize the data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Apply PCA
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

# Convert PCA result to a DataFrame
X_pca_df = pd.DataFrame(X_pca, columns=['PCA1', 'PCA2'])

# Add the target column for visualization
X_pca_df['Targets'] = y.Targets

# Visualize the PCA result
plt.figure(figsize=(14, 7))
colormap = np.array(['red', 'lime', 'black'])

# Plot the PCA transformed data
plt.scatter(X_pca_df.PCA1, X_pca_df.PCA2, c=colormap[X_pca_df.Targets], s=40)
plt.title('PCA of Iris Dataset')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.show()
```

✓ 0.6s



Lab6

Date: 31st May,2024

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

Refer:https://docs.google.com/presentation/d/11UE61G27eOAynhc8ctHAqoEaeYLrVhoT/edit?usp=sharing&ouid=117926028109390959744&rtpof=true&sd=true

Algorithm (Handwritten)

31/05/24

LAB-6

Build an artificial ~~neutra~~ neural network model with back propagation

Algorithm:

* Initialize parameters
* Normalize input features
* Normalize the output
* Set hyper parameters: no. of epochs, no. of neurons
* Define activation functions

-> Training the network

Forward propagation

* Compute input to hidden layers
* Add bias
* apply actuation function

-> Backpropagation

* Compute error
* compute gradiant
* Compute delta

-> Update weights and biases

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
```

```
#Varialble 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
    hinp1 = np.dot(x,wh)
    hinp = hinp1 + bh
    hlayer_act = sigmoid(hinp)
    outinp1 = np.dot(hlayer_act,wout)
    outinp = outinp1 + bout
    output = sigmoid(outinp)

    # Backpropagation
    EO = y - output
    outgrad = 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: \n" + str(x))
    print("Actual output: \n" + 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)

Implement Random Forest Ensemble Method

Algorithm:

1) Import necessary libraries
2) Load & insert data
3) Pre process the data as in separating features and strengths
4) Split the data to training and test samples. Use 0.4 to allocate 40% of data to testing and use rest for training.
5) Initialize random forest classifier and train it using fit method.
6) Make predictions on test sample using method predict.
7) Evaluate the model

OUTPUT:

Accuracy: 0.98

Confusion Matrix:

```
[[23  0  0]
 [ 0 19  0]
 [ 0  1 17]]
```

Code and output

```python
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))
```

✓ 0.7s

```
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        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)

Bajna Gold

Date: Page:

Implement Boosting ensemble method on a given dataset.

Algorithm:

1) Import Libraries
2) Load the dataset
3) Data preprocessing involves separation of features and dataset
4) Split the dataset to train and test Samples
5) Initialize the adaboost classifier with Specified no. of estimates and base estimators
6) Train the model using the training data.
7) Make predictions for test sample using trained model.
8) Evaluate the model

OUTPUT:

Metrics. accuracy score:

0.9833333333

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)
```

```
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
```