BANGLADESH UNIVERSITY OF BUSINESS AND TECHNOLOGY

Committed to Academic Excellence

Lab Report

Course Code: CSE 476

Course Title: Data Mining Lab

Submitted to:	Submitted by:
Name: Badhan Chandra Das Lecturer Dept. of CSE at Bangladesh University of Business and Technology.	Name: Syeda Nowshin Ibnat ID: 17183103020 Intake: 39 Section: 02 Program: B.Sc. in CSE Semester: Spring 2022

Date of Submission: 27-07-2022

Lab-1 (NumPy, Pandas)

Objective:

To be familiar with NumPy and Pandas operations.

- - - Numpy - - -

(1)

Sample Code:

a = np.array([[1, 2], [3, 4],[5,6]])

print(a)

a

Output:

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

(2)

Sample Code:
b = a.reshape(2,3)
print(b.shape)
b
Output:

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

(3)

Sample Code:

```
x = np.arange(10,20,4)
x
```

Output:

```
array([10, 14, 18])
```

(4)

Sample Code:

```
a = np.array([[0.0,0.0,0.0],[10.0,10.0,10.0],[20.0,20.0,20.0],[30.0,30.0,30.0]])
b = np.array([1.0,2.0,3.0])

print ('First array:')
print (a)
print ('\n')

print ('Second array:')
print (b)
print ('\n')

print ('First Array + Second Array')  # broadcasting
print (a + b)
```

Output:

```
First array:
[[ 0.  0.  0.]
 [10. 10. 10.]
 [20. 20. 20.]
 [30. 30. 30.]]


Second array:
[1. 2. 3.]


First Array + Second Array
[[ 1.  2.  3.]
 [11. 12. 13.]
 [21. 22. 23.]
 [31. 32. 33.]]
```

(5)

Sample Code:

```
a = np.array([[1,2],[3,4]])
print ('First array:')
print (a)
print ('\n')
b = np.array([[5,6],[7,8]])
print ('Second array:' )
print (b)
print ('\n'  )
# both the arrays are of same dimensions
print ('Joining the two arrays along axis 0:' )
print (np.concatenate((a,b), axis =0)) #by default axis =0
print ('\n'  )
print ('Joining the two arrays along axis 1:' )
print (np.concatenate((a,b),axis = 1))
```

Output:

```
First array:
[[1 2]
 [3 4]]


Second array:
[[5 6]
 [7 8]]


Joining the two arrays along axis 0:
[[1 2]
 [3 4]
 [5 6]
 [7 8]]


Joining the two arrays along axis 1:
[[1 2 5 6]
 [3 4 7 8]]
```

(6)

String Operation

Sample Code:

```
print (np.char.capitalize('hello world'))

print (np.char.lower('HELLO WORLD'))

print (np.char.upper('hello world'))

print (np.char.split ('hello how are you?'))
```

Output:

```
Hello world
hello world
HELLO WORLD
['hello', 'how', 'are', 'you?']
```

(7)

Statistical Function

Sample Code:

```
a = np.array([1,2,3,4])

print ('Our array is:')

print (a )

print ('\n'  )

print ('Applying average() function:' )

print (np.average(a) )

print ('\n' )

# this is same as mean when weight is not specified

wts = np.array([4,3,2,1])

print ('Applying average() function again:' )

print (np.average(a,weights = wts) )

print ('\n'  )

# Returns the sum of weights, if the returned parameter is set to True.

print ('Sum of weights' )

print (np.average([1,2,3,4],weights = [4,3,2,1], returned = True))
```

Output:

```
Our array is:
[1 2 3 4]


Applying average() function:
2.5


Applying average() function again:
2.0


Sum of weights
(2.0, 10.0)
```

- - - Pandas - - -

(1)

Sample Code:

```
data = np.array(['a','b','c','d'])
s = pd.Series(data,index=[100,101,102,103])
print (s)
print() #retrieve the elements
print (s[101])
print() #retrieve the first three element
print (s[:3])
#retrieve from nth element
print()
print (s[2:])
```

Output:

```
100    a
101    b
102    c
103    d
dtype: object

b

100    a
101    b
102    c
dtype: object

102    c
103    d
dtype: object
```

(2)
Dataframe

Sample Code:
```
data = [['Alex',10,25000],['Bob',12,30000],['Clarke',13,20000],['John',20,50000]]
df = pd.DataFrame(data,columns=['Name','Age','Salary'])
df
```

Output:

	Name	Age	Salary
0	Alex	10	25000
1	Bob	12	30000
2	Clarke	13	20000
3	John	20	50000

(3)

Sample Code:
```
# Adding a new column to an existing DataFrame object with column label by passing new series
print ("Adding a new column by passing as Series:")
df['three']=pd.Series([10,20,30],index=['a','b','c'])
df
```

Output:

Adding a new column by passing as Series:

	one	two	three
a	1.0	1	10.0
b	2.0	2	20.0
c	3.0	3	30.0
d	NaN	4	NaN

(4)

Sample Code:

```
del df['one']
df
```

Output:

	two	three	four
a	1	40	41.0
b	2	30	32.0
c	3	20	23.0
d	4	10	NaN

(5)

Sample Code:

```
df = df.drop('b')
df
```

Output:

	two	four
a	1	41.0
b	2	32.0
c	3	23.0
d	4	NaN

→

	two	four
a	1	41.0
c	3	23.0
d	4	NaN

Lab -2 Data Visualization (Matplotlib, Seaborn)

Objective:

To be familiar with data visualization using Matplotlib and Seaborn.

About the dataset:

- For this lab work we used numeric dataset and dataset format is *.csv*.
- Name of the dataset: iris.csv
- Total number of data: 150

1	Id	sepal.length	sepal.width	petal.length	petal.width	variety
2	1	5.1	3.5	1.4	0.2	Setosa
3	2	4.9	3	1.4	0.2	Setosa
4	3	4.7	3.2	1.3	0.2	Setosa
5	4	4.6	3.1	1.5	0.2	Setosa
6	5	5	3.6	1.4	0.2	Setosa

Figure1: Dataset view

- - - Matplotlib - - -

1) Line Graph

Sample Code:

```
x = np.linspace(0,20)

y= x**2

plt.plot(x, y)

plt.xlabel('x-values')

plt.ylabel('x^2-values')

plt.title('Fig. 1 Line GRaph ')
```

plt.grid(True)

plt.show()

Output:

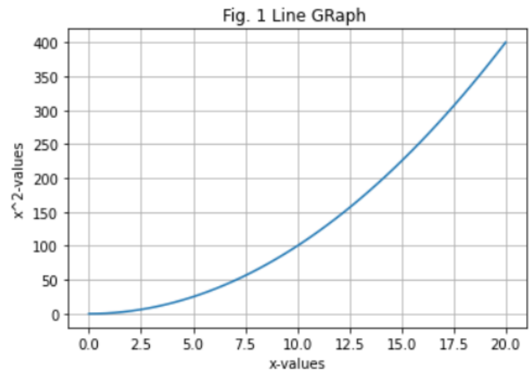


Figure2: Line Graph 1

Sample Code:

plt.plot(np.sin(x),label='sin(x)',color='orange')

plt.plot(np.cos(x),label='cos(x)',color='green')

plt.xlim(10,50)

plt.legend()

plt.title('math functions')

plt.show()

Output:

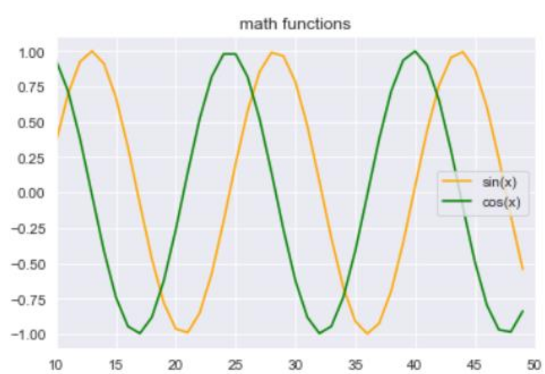


Figure3: Line Graph 2

2) Sub Plot

Sample Code:

```
fig, axs = plt.subplots(3, 2,gridspec_kw={'hspace': 0.5, 'wspace': 0.5})

x = np.linspace(0, 20, 400)

y = np.sin(x)

z=np.cos(x)

m = (x**3)

n=(x**2)

axs[0, 0].plot(x, y)

axs[0, 0].set_title('sin(x)')

axs[0, 1].plot(x, z, 'tab:orange')

axs[0, 1].set_title('cos(x)')

axs[1, 0].plot(x, m, 'tab:green')

axs[1, 0].set_title('x**3')

axs[1, 1].plot(x,n, 'tab:red')

axs[1, 1].set_title('x**2')

plt.show()
```

Output:

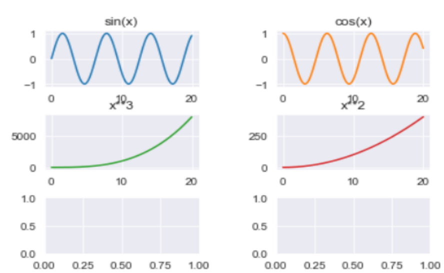


Figure4: Subplot

3) Scatter Plots

Sample Code:

```
colours = {'Setosa':'orange', 'Versicolor':'green', 'Virginica':'blue'}

for i in range(len(iris['sepal.length'])):

plt.scatter(iris['petal.length'][i],iris['petal.width'][i], color = colours[iris['variety'][i]])

plt.title('Scatter Plot')

plt.xlabel('petal length')

plt.ylabel('petal width')

plt.grid(True)

plt.show()
```

Output:

Figure5: Scatter plot

4) Bar plots

Sample Code:

```
a= iris['variety'].value_counts()

species_types = a.index

count = a.values

plt.bar(species,count,color = 'lightgreen')

plt.xlabel('species_types')

plt.ylabel('count')

plt.show()
```

Output:

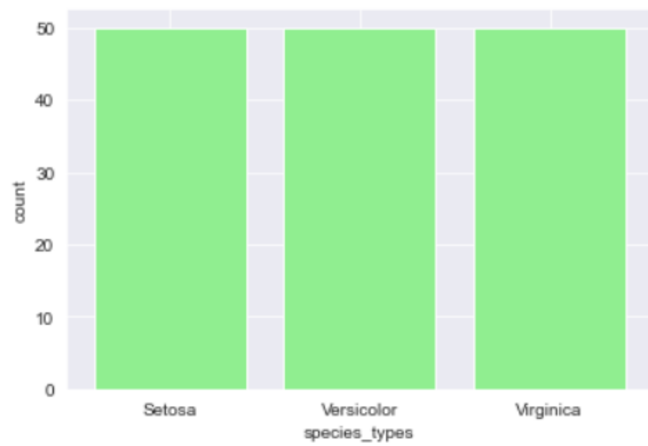


Figure6: Bar plot

5) Boxplot

Sample Code:

```
length_width = iris[['petal.length','petal.width','sepal.length','sepal.width']] #excluding species column

length_width.boxplot()

plt.xlabel('Flower measurements')
```

```
plt.ylabel('values')
plt.title("Iris dataset analysis")
plt.show()
```

Output:

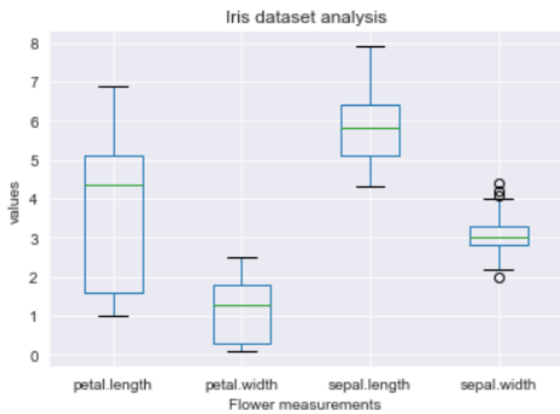


Figure7: Boxplot

6) Histograms

Sample Code:

```
data_ = np.random.randn(1000)
plt.hist(data_,bins = 40,color='blue')
plt.grid(True)
plt.xlabel('points')
plt.title("Histogram")
plt.show()
```

Output:

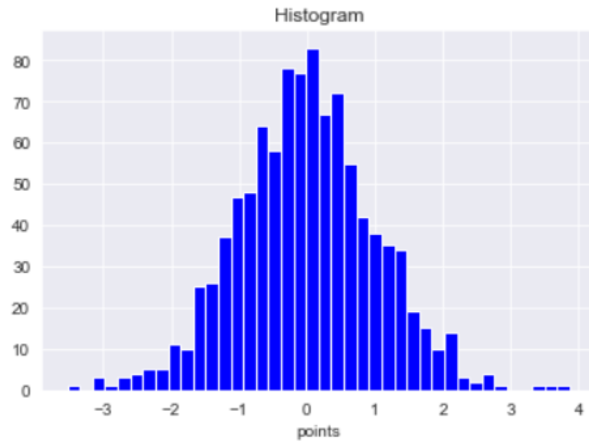


Figure8: Histogram

7) Pie Charts

Sample Code:

```
a= iris['variety'].value_counts()

species = a.index

count = a.values

colors= ['lightblue','lightgreen','gold']

explode = (0,0.6,0)

plt.pie(count, labels=species,shadow=False,

colors=colors,explode = explode, autopct='%1.1f%%')

plt.xlabel('species')

plt.axis('equal')

plt.show()
```

Output:

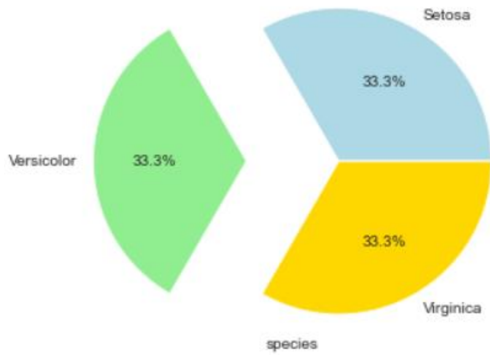


Figure9: Pie chart

- - - Seaborn - - -

1) Line Graph

Sample Code:

```
sns.set_style('darkgrid')

sns.lineplot(data=iris.drop(['variety'], axis=1))

plt.show()
```

Output:

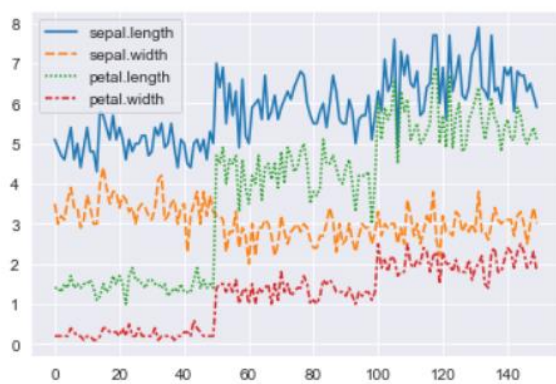


Figure10: Line Graph 3

2) Scatter Plot

Sample Code:

```
sns.FacetGrid(iris, hue="variety", height=4).map(plt.scatter, "sepal.length", "sepal.width").add_legend()

plt.show()
```

Output:

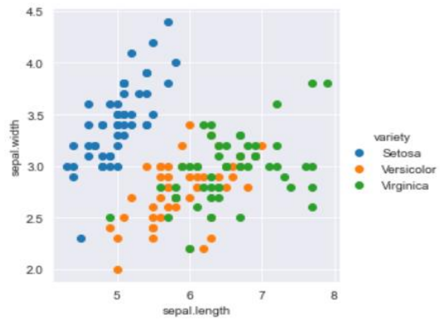


Figure11: Scatter plot 2

3) Bar Plot

Sample Code:

```
sns.FacetGrid(iris, hue="variety", height=5) \

.map(sns.distplot, "petal.length") \

.add_legend()

plt.show()
```

Output:

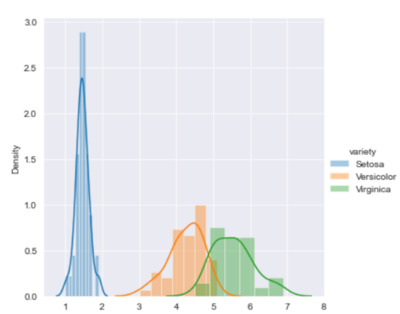


Figure12: Bar plot 2

Lab - 3 Data cleaning (Handling missing values, Outlier)

Objective:

To be familiar with handling missing values of data and outlier.

About the Dataset:

Total number of data: 9576

Datatype: Numerical

Data format: .csv file

car	price	body	mileage	engV	engType	registratio	year	model	drive
Ford	15500	crossover	68	2.5	Gas	yes	2010	Kuga	full
Mercedes-Benz	20500	sedan	173	1.8	Gas	yes	2011	E-Class	rear
Mercedes-Benz	35000	other	135	5.5	Petrol	yes	2008	CL 550	rear
Mercedes-Benz	17800	van	162	1.8	Diesel	yes	2012	B 180	front
Mercedes-Benz	33000	vagon	91	NA	Other	yes	2013	E-Class	

Figure13: Dataset view

1) Handling missing values

Sample Code:

data_1=data.copy()

data_1.isnull().sum()

Output:

```
car               0
price             0
body              0
mileage           0
engV            434
engType           0
registration      0
year              0
model             0
drive           511
dtype: int64
```

- **Eliminating missing values(row)**

Sample Code:

```
data_without_missing_values=data_1.dropna(subset=["engV","drive"])
data_without_missing_values
```

Output:

	car	price	body	mileage	engV	engType	registration	year	model	drive
0	Ford	15500.0	crossover	68	2.5	Gas	yes	2010	Kuga	full
1	Mercedes-Benz	20500.0	sedan	173	1.8	Gas	yes	2011	E-Class	rear
2	Mercedes-Benz	35000.0	other	135	5.5	Petrol	yes	2008	CL 550	rear
3	Mercedes-Benz	17800.0	van	162	1.8	Diesel	yes	2012	B 180	front
5	Nissan	16600.0	crossover	83	2.0	Petrol	yes	2013	X-Trail	full
...	...	...	...	...	...	...	...	...	...	...

- **Eliminating missing values(columns)**

Sample Code:

```
data_1.drop(['engV', 'drive'], axis=1)
```

Output:

	car	price	body	mileage	engType	registration	year	model
0	Ford	15500.0	crossover	68	Gas	yes	2010	Kuga
1	Mercedes-Benz	20500.0	sedan	173	Gas	yes	2011	E-Class
2	Mercedes-Benz	35000.0	other	135	Petrol	yes	2008	CL 550
3	Mercedes-Benz	17800.0	van	162	Diesel	yes	2012	B 180
4	Mercedes-Benz	33000.0	vagon	91	Other	yes	2013	E-Class
...	...	...	...	...	...	...	...	...

- **Estimate missing values**

Sample Code:

```
data_1=data_1.iloc[:,:].values
data_1
imputer = SimpleImputer(missing_values=np.nan, strategy='most_frequent')
imputer=imputer.fit(data_1[0:,0:10]) #1st portion is for row (start:end): 2nd portion is for columns (start:end)
```

data_1[0:,0:10]=imputer.transform(data_1[0:,0:10])

data_1[0:,0:10]

data_1

data_without_missing_values_1=pd.DataFrame(data_1)

data_without_missing_values_1

...	...	...	...	...	...	...	...	...	...	...
9571	Hyundai	14500	crossover	140	2	Gas	yes	2011	Tucson	front
9572	Volkswagen	2200	vagon	150	1.6	Petrol	yes	1986	Passat B2	front
9573	Mercedes-Benz	18500	crossover	180	3.5	Petrol	yes	2008	ML 350	full
9574	Lexus	16999	sedan	150	3.5	Gas	yes	2008	ES 350	front
9575	Audi	22500	other	71	3.6	Petrol	yes	2007	Q7	full

data_without_missing_values_2=data_without_missing_values_1.copy()

data_without_missing_values_2.isnull().sum()

```
0    0
1    0
2    0
3    0
4    0
5    0
6    0
7    0
8    0
9    0
dtype: int64
```

data_without_missing_values_2

Output:

	0	1	2	3	4	5	6	7	8	9
0	Ford	15500	crossover	68	2.5	Gas	yes	2010	Kuga	full
1	Mercedes-Benz	20500	sedan	173	1.8	Gas	yes	2011	E-Class	rear
2	Mercedes-Benz	35000	other	135	5.5	Petrol	yes	2008	CL 550	rear
3	Mercedes-Benz	17800	van	162	1.8	Diesel	yes	2012	B 180	front
4	Mercedes-Benz	33000	vagon	91	2	Other	yes	2013	E-Class	front
...	...	...	...	...	...	...	...	...	...	...

Outlier

Sample Code:

q1=data_without_missing_values_1[4].quantile(0.998)

q2=data_without_missing_values_1[4].quantile(0.0013)

data_without_outlier_top=data_without_missing_values_1[data_without_missing_values_1[4]<
q1]

data_without_outlier=data_without_outlier_top[data_without_outlier_top[4]>q2]

data_without_outlier

Output:

	0	1	2	3	4	5	6	7	8	9
0	Ford	15500	crossover	68	2.5	Gas	yes	2010	Kuga	full
1	Mercedes-Benz	20500	sedan	173	1.8	Gas	yes	2011	E-Class	rear
2	Mercedes-Benz	35000	other	135	5.5	Petrol	yes	2008	CL 550	rear
3	Mercedes-Benz	17800	van	162	1.8	Diesel	yes	2012	B 180	front
4	Mercedes-Benz	33000	vagon	91	2	Other	yes	2013	E-Class	front
...	...	...	...	...	...	...	...	...	...	...

Data Aggregation

(1)

Sample Code:

df1=pd.DataFrame({'A':['A0','A1','A2','A3'],

'B':['B0','B1','B2','B3'],

'C':['C0','C1','C2','C3'],

'D':['D0','D1','D2','D3'],},

index=[0,1,2,3])

df1

Output:

	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3

(2)

Sample Code:

pd.concat([df1,df2])

Output:

	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3
4	A4	B4	C4	D4
5	A5	B5	C5	D5
6	A6	B6	C6	D6
7	A7	B7	C7	D7

Join

Left Outer Join

pd.merge(left, right, how="left", on=["key1", "key2"])

	key1	key2	A	B	C	D
0	K0	K0	A0	B0	C0	D0
1	K0	K1	A1	B1	NaN	NaN
2	K1	K0	A2	B2	C1	D1
3	K1	K0	A2	B2	C2	D2
4	K2	K1	A3	B3	NaN	NaN

Right Outer Join

pd.merge(left, right, how="right", on=["key1", "key2"])

	key1	key2	A	B	C	D
0	K0	K0	A0	B0	C0	D0
1	K1	K0	A2	B2	C1	D1
2	K1	K0	A2	B2	C2	D2
3	K2	K0	NaN	NaN	C3	D3

Full Outer Join

pd.merge(left, right, how="outer", on=["key1", "key2"])

	key1	key2	A	B	C	D
0	K0	K0	A0	B0	C0	D0
1	K0	K1	A1	B1	NaN	NaN
2	K1	K0	A2	B2	C1	D1
3	K1	K0	A2	B2	C2	D2
4	K2	K1	A3	B3	NaN	NaN
5	K2	K0	NaN	NaN	C3	D3

Inner Join

pd.merge(left, right, how="inner", on=["key1", "key2"])

	key1	key2	A	B	C	D
0	K0	K0	A0	B0	C0	D0
1	K1	K0	A2	B2	C1	D1
2	K1	K0	A2	B2	C2	D2

Lab - 4 (Decision Tree, Naive Bayes, KNN)

Objective:

To be familiar with some algorithms such as Decision Tree, Naive Bayes, and KNN.

- - - Decision Tree - - -

Introduction:

Decision Tree is a Supervised learning technique that can be used for both classification and Regression problems. It is a tree-structured classifier, where internal nodes represent the features of a dataset, branches represent the decision rules and each leaf node represents the outcome.

Sample Code:

```
import pandas as pd

import numpy as np

from sklearn.datasets import load_iris

#load in the data

data = load_iris()

#convert to a dataframe

df = pd.DataFrame(data.data, columns = data.feature_names)

#create the species column

df['Species'] = data.target
```

#replace this with the actual names

target = np.unique(data.target)

target_names = np.unique(data.target_names)

targets = dict(zip(target, target_names))

df['Species'] = df['Species'].replace(targets)

df

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	Species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa
...	...	...	...	...	...

#extract features and target variables

x = df.drop(columns="Species")

y = df["Species"]

#save the feature name and target variables

feature_names = x.columns

labels = y.unique()

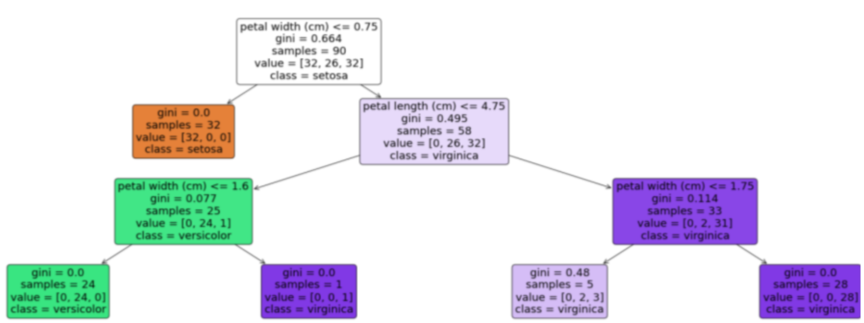
#split the dataset

from sklearn.model_selection import train_test_split

X_train, test_x, y_train, test_lab = train_test_split(x,y, test_size = 0.4, random_state = 10)

from sklearn.tree import DecisionTreeClassifier

```
#import relevant packages

from sklearn import tree

import matplotlib.pyplot as plt

#plt the figure, setting a black background

plt.figure(figsize=(30,10), facecolor ='w')

#create the tree plot

a = tree.plot_tree(clf,

#use the feature names stored

feature_names = feature_names,

#use the class names stored

class_names = labels,

rounded = True,

filled = True,

fontsize=18)

#show the plot

plt.show()
```



#import relevant functions

from sklearn.tree import export_text

#export the decision rules

tree_rules = export_text(clf,

feature_names = list(feature_names))

#print the result

print(tree_rules)

```
|--- petal width (cm) <= 0.75
|   |--- class: setosa
|--- petal width (cm) >  0.75
|   |--- petal length (cm) <= 4.75
|   |   |--- petal width (cm) <= 1.60
|   |   |   |--- class: versicolor
|   |   |--- petal width (cm) >  1.60
|   |   |   |--- class: virginica
|   |--- petal length (cm) >  4.75
|   |   |--- petal width (cm) <= 1.75
|   |   |   |--- class: virginica
|   |   |--- petal width (cm) >  1.75
|   |   |   |--- class: virginica
```

test_pred_decision_tree = clf.predict(test_x)

from sklearn import metrics

import seaborn as sns

import matplotlib.pyplot as plt

#get the confusion matrix

confusion_matrix = metrics.confusion_matrix(test_lab, test_pred_decision_tree)

#turn this into a dataframe

matrix_df = pd.DataFrame(confusion_matrix)

#plot the result

ax = plt.axes()

sns.set(font_scale=1.3)

plt.figure(figsize=(10,7))

sns.heatmap(matrix_df, annot=True, fmt="g", ax=ax, cmap="magma")

#set axis titles

ax.set_title('Confusion Matrix - Decision Tree')

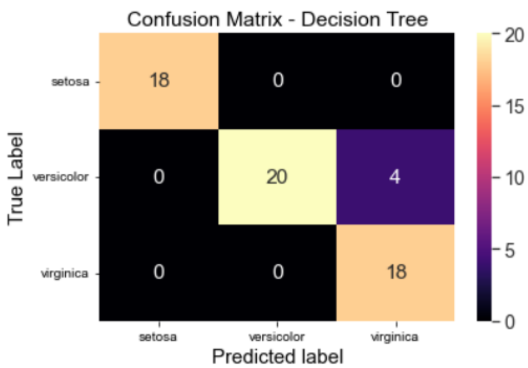
ax.set_xlabel("Predicted label", fontsize =15)

ax.set_xticklabels([''+labels)

ax.set_ylabel("True Label", fontsize=15)

ax.set_yticklabels(list(labels), rotation = 0)

plt.show()



metrics.accuracy_score(test_lab, test_pred_decision_tree)

test_pred_decision_tree = clf.predict(test_x)

metrics.accuracy_score(test_lab, test_pred_decision_tree)

#get the precision score

precision = metrics.precision_score(test_lab,

test_pred_decision_tree,

average=None)

#turn it into a dataframe

precision_results = pd.DataFrame(precision, index=labels)

#rename the results column

precision_results.rename(columns={0:'precision'}, inplace =True)

precision_results

	precision
setosa	1.000000
versicolor	1.000000
virginica	0.818182

recall = metrics.recall_score(test_lab, test_pred_decision_tree,

average =None)

recall_results = pd.DataFrame(recall, index= labels)

recall_results.rename(columns ={0:'Recall'}, inplace =True)

recall_results

	Recall
setosa	1.000000
versicolor	0.833333
virginica	1.000000

f1 = metrics.f1_score(test_lab, test_pred_decision_tree, average=None)

f1_results = pd.DataFrame(f1, index=labels)

f1_results.rename(columns={0:'f1'}, inplace=True)

f1_results

	f1
setosa	1.000000
versicolor	0.909091
virginica	0.900000

print(metrics.classification_report(test_lab, test_pred_decision_tree))

Output:

```
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        18
  versicolor       1.00      0.83      0.91        24
   virginica       0.82      1.00      0.90        18

    accuracy                           0.93        60
   macro avg       0.94      0.94      0.94        60
weighted avg       0.95      0.93      0.93        60
```

#extract importance

importance = pd.DataFrame({'feature': X_train.columns,

'importance' : np.round(clf.feature_importances_, 3)})

importance.sort_values('importance', ascending=False, inplace = True)

print(importance)

```
             feature  importance
3   petal width (cm)       0.599
2  petal length (cm)       0.401
0  sepal length (cm)       0.000
1   sepal width (cm)       0.000
```

- - - Naïve Bayes - - -

Introduction:

Naïve Bayes algorithm is a supervised learning algorithm, which is based on Bayes theorem and used for solving classification problems.

About the Dataset:

Total number of data: 400

Datatype: Numerical

Data format: .csv file

Sample Code:

```
import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

import sklearn

dataset = pd.read_csv(r(r'N:\STUDY\University\1-Undergrad\4th-Year\Semester-12\DM-Lab\archive\Social_Network_Ads.csv'))

X = dataset.iloc[:, [1, 2, 3]].values

y = dataset.iloc[:, -1].values

dataset
```

	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0
...	...	...	...	...	...

from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()

X[:,0] = le.fit_transform(X[:,0])

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20, random_state = 0)

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

X_train = sc.fit_transform(X_train)

X_test = sc.transform(X_test)

```
array([0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1,
       0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
       1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1,
       0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1], dtype=int64)
```

from sklearn.naive_bayes import GaussianNB

classifier = GaussianNB()

classifier.fit(X_train, y_train)

y_pred = classifier.predict(X_test)

y_pred

from sklearn.metrics import confusion_matrix,accuracy_score

cm = confusion_matrix(y_test, y_pred)

ac = accuracy_score(y_test,y_pred)

ac

```
0.925
```

Making the Confusion Matrix

from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score, f1_score

cm = confusion_matrix(y_test, y_pred)

ac = accuracy_score(y_test,y_pred)

pr= precision_score(y_test,y_pred)

rc=recall_score(y_test,y_pred)

f1= f1_score(y_test,y_pred)

cm

```
array([[56,  2],
       [ 4, 18]], dtype=int64)
```

print("Accuracy", ac);

print("Precision", pr)

print("Recall", rc)

print("F1 Score", f1)

Final Output:

```
Accuracy 0.925
Precision 0.9
Recall 0.8181818181818182
F1 Score 0.8571428571428572
```

- - - KNN - - -

Introduction:

K-Nearest Neighbor is one of the simplest algorithms based on Supervised Learning technique. K-NN algorithm can be used for Regression as well as for Classification but mostly it is used for the Classification problems.

About the dataset:

- For this lab work we used numeric dataset and dataset format is *.csv*.
- Name of the dataset: iris.csv
- Total number of data: 150

Sample Code:

import pandas as pd

import numpy as np

import math

import operator

import pandas as pd

import numpy as np

from sklearn.datasets import load_iris

#load in the data

data = load_iris()

#convert to a dataframe

data = pd.read_csv(r"N:\STUDY\University\1-Undergrad\4th-Year\Semester-12\DM-Lab\iris.csv")

data

	Id	sepal.length	sepal.width	petal.length	petal.width	variety
0	1	5.1	3.5	1.4	0.2	Setosa
1	2	4.9	3.0	1.4	0.2	Setosa
2	3	4.7	3.2	1.3	0.2	Setosa
3	4	4.6	3.1	1.5	0.2	Setosa
4	5	5.0	3.6	1.4	0.2	Setosa
...	...	...	...	...	...	...

print(data.describe())

	Id	sepal.length	sepal.width	petal.length	petal.width
count	150.000000	150.000000	150.000000	150.000000	150.000000
mean	75.500000	5.843333	3.057333	3.758000	1.199333
std	43.445368	0.828066	0.435866	1.765298	0.762238
min	1.000000	4.300000	2.000000	1.000000	0.100000
25%	38.250000	5.100000	2.800000	1.600000	0.300000
50%	75.500000	5.800000	3.000000	4.350000	1.300000
75%	112.750000	6.400000	3.300000	5.100000	1.800000
max	150.000000	7.900000	4.400000	6.900000	2.500000

feature_columns = ['sepal.length', 'sepal.width', 'petal.length','petal.width']

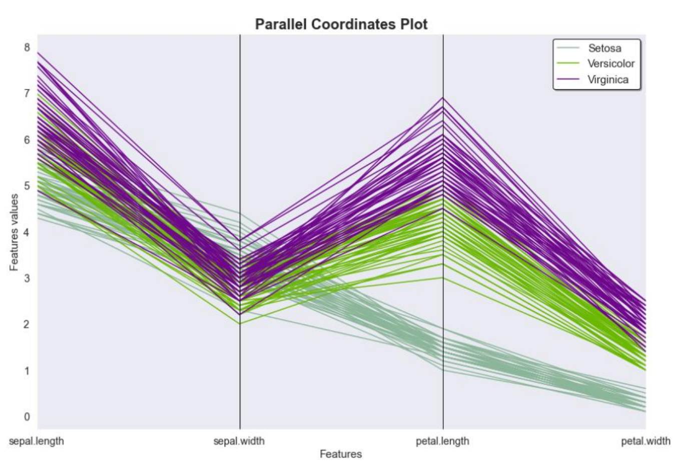
X = data[feature_columns].values

y = data['variety'].values

from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()

```
y = le.fit_transform(y)

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)

import matplotlib.pyplot as plt

import seaborn as sns

%matplotlib inline

from pandas.plotting import parallel_coordinates

plt.figure(figsize=(15,10))

parallel_coordinates(data.drop("Id", axis=1), "variety")

plt.title('Parallel Coordinates Plot', fontsize=20, fontweight='bold')

plt.xlabel('Features', fontsize=15)

plt.ylabel('Features values', fontsize=15)

plt.legend(loc=1, prop={'size': 15}, frameon=True,shadow=True, facecolor="white",
edgecolor="black")

plt.show()
```



```
# Fitting clasifier to the Training set

# Loading libraries

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import confusion_matrix, accuracy_score

from sklearn.model_selection import cross_val_score

# Instantiate learning model (k = 3)

classifier = KNeighborsClassifier(n_neighbors=3)

# Fitting the model

classifier.fit(X_train, y_train)

y_pred = classifier.predict(X_test) # Predicting the Test set results

cm = confusion_matrix(y_test, y_pred)

cm
```

```
array([[11,  0,  0],
       [ 0, 12,  1],
       [ 0,  0,  6]], dtype=int64)
```

```
accuracy = accuracy_score(y_test, y_pred)*100

print('Accuracy of our model is equal ' + str(round(accuracy, 2)) + ' %.')
```

```
Accuracy of our model is equal 96.67 %.
```

```
# creating list of K for KNN

k_list = list(range(1,50,2))

cv_scores = [] # creating list of cv scores
```

```
for k in k_list: # perform 10-fold cross validation

knn = KNeighborsClassifier(n_neighbors=k)

scores = cross_val_score(knn, X_train, y_train, cv=10, scoring='accuracy')

cv_scores.append(scores.mean())

MSE = [1 - x for x in cv_scores]

plt.figure()

plt.figure(figsize=(15,10))

plt.title('The optimal number of neighbors', fontsize=20, fontweight='bold')

plt.xlabel('Number of Neighbors K', fontsize=15)

plt.ylabel('Misclassification Error', fontsize=15)

sns.set_style("whitegrid")

plt.plot(k_list, MSE)

plt.show()
```

Output: