EDA LAB-1

import numpy as np

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

array\_Sum = np.sum(array)

array\_mean = np.mean(array)

print("Array:", array)

print("Sum;", array\_Sum)

print(" Mean:", array\_mean)

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

transpose = matrix.T

determinant = np.linalg.det(matrix)

print("Matrix: \n", matrix)

print("Tramspose: \n", transpose)

Print("Determinant : \n", determinant)

import pandas as pd

data = {'name': ['Alice', 'Baby', 'Charlie'], 'Age': [20, 32, 34]}

df = pd.DataFrame(data)

Print(df)

print(df['name'])

Print(df.describe())

data = {'name': ['Alice', 'Bob', 'Charlie'], 'Age': [24, 27, 22], 'Score': [85, 83, 79]}

df = pd.DataFrame(data)

df['Pass'] = df['Score'] > 80

print("Data Frame: \n", df)

print("Average Age:", df['Age'].mean())

import numpy as np

import matplotlib.pyplot as plt

y1 = [2, 1, 2, 5]

y2 = [1, 1.5, 4]

plt.plot(y1)

plt.plot(y2)

plt.legend(["blue", "green"], loc="lower right")

plt.show()

import seaborn as sns

import matplotlib.pyplot as plt

tips = sns.load\_dataset("tips")

sns.scatterplot(data=tips, x="total\_bill", y="tip", hue="sex")

plt.title("Tips dataset scatter plot")

plt.show()

import seaborn as sns

import matplotlib.pyplot as plt

tips = sns.load\_dataset("tips")

sns.boxplot(x="day", y="total\_bill", data=tips)

plt.title("Total bill distribution by day")

plt.show()

from sklearn.datasets import load\_iris

iris = load\_iris()

X = iris.data

y = iris.target

feature\_names = iris.feature\_names

target\_names = iris.target\_names

Print("feature names:", feature\_names)

print("target names:", target\_names)

print("\n Type of x is:", type(X))

print("\n first 5 rows of x:\n", X[:5])

from sklearn.datasets import load\_iris from sklearn.model\_selection import train\_test\_split from sklearn.tree import DecisionTreeClassifier from sklearn.metrics import accuracy\_score

iris = load\_iris()

X, y = iris.data, iris.target

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.30)

clf = DecisionTreeClassifier()

clf.fit(X\_train, y\_train)

y\_pred = clf.predict(X\_test)

accuracy = accuracy\_score(y\_test, y\_pred) print("Accuracy of the Decision Tree classifier", accuracy)

LAB2:

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

dataset = pd.read\_csv("C:/users/android/Desktop/diabetes.csv")

Print(dataset)

dataset = dataset.drop\_duplicates()

dataset.isnull()

dataset.sum()

dataset.tail()

dataset.describe()

dataset.info()

import pandas as pd

from sklearn.impute import SimpleImputer

data = pd.DataFrame({

'name': ['ranya', 'prithvi', 'Siri', 'Vanya', None],

'age': [21, 34, None, 21, 22],

'purchase amount': [100.5, None, None, 100.5, 50.5],

'Date of Purchase': ['2023/12/01', '2023/12/02', '2023/12/01', '2023/12/01', '2023/12/03']

})

imputer = SimpleImputer(strategy='mean')

data[['age', 'purchase amount']] = imputer.fit\_transform(data[['age', 'purchase amount']])

data['Date of Purchase'] = pd.to\_datetime(data['Date of Purchase'], errors='coerce')

Print(data)

import pandas as pd

data1 = pd.DataFrame({ 'Cust\_id': [1, 2, 3], 'name': ['Sam', 'Tom', 'Ram'], 'age': [28, 34, 29] })

data2 = pd.DataFrame({ 'Cust\_id': [1, 3, 4], 'purchase amount': [100.5, 85.3, 45.0], 'purchase-date': ['2023-02-01', '2023-09-02', '2023-12-01'] })

merged\_data = pd.merge(data1, data2, on='Cust\_id', how='inner') Print(merged\_data)

merged\_data = pd.merge(data1, data2, on='Cust\_id', how='outer') print(merged\_data)

merged\_data = pd.merge(data1, data2, on='Cust\_id', how='right') Print(merged\_data)

merged\_data = pd.merge(data1, data2, on='Cust\_id', how='left') print(merged\_data)

from sklearn.preprocessing import StandardScaler, OneHotEncoder

data = pd.DataFrame({'category': ['A', 'B', 'A', 'C', 'B'], 'numerical\_column': [10, 15, 10, 20, 15]})

scaler = StandardScaler() data['scaled\_numerical\_column'] = scaler.fit\_transform(data[['numerical\_column']])

encoder = OneHotEncoder(sparse\_output=False, drop='first') encoded\_data = pd.DataFrame(encoder.fit\_transform(data[['category']]), columns=encoder.get\_feature\_names\_out(['category']))

data = pd.concat([data, encoded\_data], axis=1) print(data)

from sklearn.decomposition import PCA from sklearn.feature\_selection import SelectKBest, chi2

data = pd.DataFrame({'feature1': [10, 20, 30, 40, 50], 'feature2': [2, 3, 2, 1, 2], 'feature3': [100, 200, 300, 400, 500], 'target': [0, 1, 0, 1, 0]})

selector = SelectKBest(chi2, k=2)

selected\_features\_chi2 = selector.fit\_transform(data[['feature1', 'feature2', 'feature3']], data['target'])

print("Selected features (SelectKBest chi2):") print(selected\_features\_chi2)

pca = PCA(n\_components=2) pca\_data = pca.fit\_transform(data[['feature1', 'feature2', 'feature3']])

print("PCA reduced data:") print(pca\_data)

Lab- 4 (13/02/2025)

#data Transformation

import pandas as pd

data = {'Name':['Tom', 'Nick', 'Krish', 'Jack'],

'Age': [20,21,19,18]}

df = pd.DataFrame(data)

print(df)

Name Age  
0 Tom 20  
1 Nick 21  
2 Krish 19  
3 Jack 18

import pandas as pd

data = {'Name' :['Jai', 'Princi', 'Gaurav', 'Anuj'],

'Age': [27,24,22,32],

'Address': ['Delhi', 'Kanpur', 'Allahabad', 'Kannauj'],

'Qualification': ['Msc', 'MA', 'MCA', 'Phd']}

df = pd.DataFrame(data)

print(df[['Name', 'Qualification']])

Name Qualification  
0 Jai Msc  
1 Princi MA  
2 Gaurav MCA  
3 Anuj Phd

#groupby

import pandas as pd

df=pd.DataFrame({'column1':['A', 'B', 'C', 'A', 'C', 'C', 'B', 'D', 'D', 'A'],

'column2': [5,10,15,20,25,30,35,40,45,50]})

df = df.groupby('column1')['column2'].apply(list)

df

column1  
A [5, 20, 50]  
B [10, 35]  
C [15, 25, 30]  
D [40, 45]  
Name: column2, dtype: object

#groupby and mean

df = pd.DataFrame({'Animal': ['Falcon', 'Falcon', 'Parrot', 'Parrot'],

'Max speed': [380,370,24,26]})

df.groupby(['Animal']).mean()

|  |  |
| --- | --- |
|  | Max speed |
| Animal |  |
| Falcon | 375.0 |
| Parrot | 25.0 |

import numpy as np

import pandas as pd

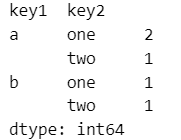
df=pd.DataFrame({'key1':['a','a','b','b','a'],

'key2':['one','two','one','two','one'],

'data1':np.random.randn(5),

'data1':np.random.randn(5)})

df.groupby(['key1','key2']).size()



import pandas as pd

import numpy as np

columns = pd.MultiIndex.from\_array([['US', 'US', 'US', 'JP', 'JP'], [1, 3, 5, 1, 3]], names=['cty', 'tenor'])

hier\_df = pd.DataFrame(np.random.randn(4, 5), columns=columns)

print(hier\_df)

#aggregation function

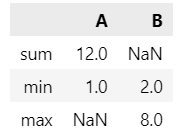
#aggereagete these function over the row

import numpy as np

df=pd.DataFrame([[1,2,3],[4,5,6],[7,8,9],[np.nan,np.nan,np.nan]],

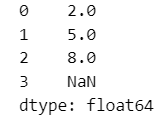
columns=['A','B','C'])

df.agg(['sum','min'])



#aggregation over the column

df.agg("mean",axis="columns")

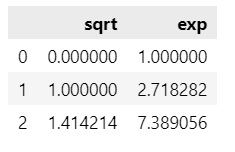


#data tranformation

df=pd.DataFrame({'A':range(3),'B':range(1,4)})

s=pd.Series(range(3))

s.transform([np.sqrt,np.exp])



df=pd.DataFrame({

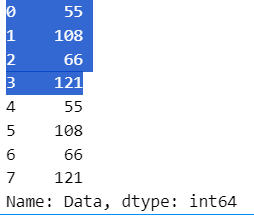
"Date":[

"2015-05-08","2015-05-07","2015-05-06","2015-05-05","2015-05-08","2015-05-07","2015-05-06","2015-05-05"],

"Data":[5,8,6,1,50,100,60,120],

})

df.groupby('Date')['Data'].transform('sum')

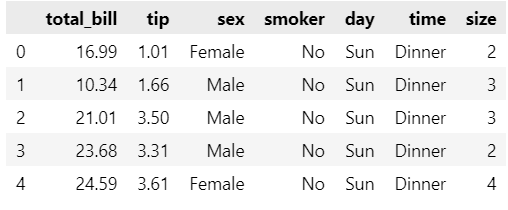


#pivot table

import seaborn as sns

tips=sns.load\_dataset("tips")

tips.head()



pd.pivot\_table(tips, index="sex",values=["tip"])

pd.pivot\_table(tips, index="sex",values=["tip"] ,aggfunc="sum")

pd.pivot\_table(tips, index="sex",values=["tip"],columns="time")

pd.pivot\_table(tips, index="sex",values=["tip"],columns="time",margins=1)

pd.pivot\_table(tips, index=["sex","smoker"],values=["total\_bill"],columns="time")

pd.pivot\_table(tips, index=["sex","smoker"],values=["total\_bill","tip"],columns="time")

pd.crosstab(tips["sex"],tips["time"])

pd.crosstab(tips["sex"],tips["time"],normalize="index")

pd.crosstab(tips["sex"],tips["time"],normalize="columns")

pd.crosstab(tips["sex"],tips["time"],normalize="all")

pd.crosstab(tips["sex"],tips["time"],values=tips["tip"], aggfunc="mean")

pd.crosstab(tips["sex"],tips["time"],values=tips["tip"], aggfunc="mean",rownames=["Gender"],colnames=["Time"])

pd.crosstab(tips["sex"],[tips["time"],tips["smoker"]])

pd.crosstab([tips["sex"],tips["time"]],[tips["time"],tips["smoker"]])

pd.crosstab(tips["day"],tips["sex"],normalize="index").plot.bar(stacked=True)

pd.crosstab(tips["day"],tips["sex"],normalize="index").plot.bar(stacked=False)

Lab- 5

#LINEAA REGRESSION

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LinearRegression,Ridge,Lasso,ElasticNet

from sklearn.preprocessing import PolynomialFeatures

from sklearn.metrics import mean\_squared\_error

#generate synatatic dataset

np.random.seed(42)

x=2\*np.random.rand(100,1)

y=4+3\*x+np.random.randn(100,1)

#train \_test\_split

X\_train,X\_test,y\_train,y\_test=train\_test\_split(x,y,test\_size=0.2,random\_state=40)

lin\_reg=LinearRegression()

lin\_reg.fit(X\_train,y\_train)

y\_pred=lin\_reg.predict(X\_test)

print("simple linear regression MSE",mean\_squared\_error(y\_test,y\_pred))

x\_multi=np.c\_[x,x\*\*2]

X\_train\_m,X\_test\_m,y\_train\_m,y\_test\_m=train\_test\_split(x\_multi,y,test\_size=0.2,random\_state=40)

lin\_reg\_multi=LinearRegression()

lin\_reg\_multi.fit(X\_train\_m,y\_train\_m)

y\_pred\_multi=lin\_reg\_multi.predict(X\_test\_m)

print("multiple linear regression MSE",mean\_squared\_error(y\_test\_m,y\_pred\_multi))

poly\_features=PolynomialFeatures(degree=2,include\_bias=False)

X\_poly=poly\_features.fit\_transform(x)

X\_train\_p,X\_test\_p,y\_train\_p,y\_test\_p=train\_test\_split(X\_poly,y,test\_size=0.2,random\_state=40)

lin\_reg\_poly=LinearRegression()

lin\_reg\_poly.fit(X\_train\_p,y\_train\_p)

y\_pred\_poly=lin\_reg\_poly.predict(X\_test\_p)

print("polynomial linear regression MSE",mean\_squared\_error(y\_test\_p,y\_pred\_poly))

ridge\_reg=Ridge(alpha=1.0)

ridge\_reg.fit(X\_train\_p,y\_train\_p)

y\_pred\_ridge=ridge\_reg.predict(X\_test\_p)

print("ridge regression MSE",mean\_squared\_error(y\_test\_p,y\_pred\_ridge))

lasso\_reg=Lasso(alpha=0.1)

lasso\_reg.fit(X\_train\_p,y\_train\_p)

y\_pred\_lasso=lasso\_reg.predict(X\_test\_p)

print("lasso regression",mean\_squared\_error(y\_test\_p,y\_pred\_lasso))

elastic\_net=ElasticNet(alpha=0.1,l1\_ratio=0.5)

elastic\_net.fit(X\_train\_p,y\_train\_p)

y\_pred\_elastic=elastic\_net.predict(X\_test\_p)

print("elastic regression",mean\_squared\_error(y\_test\_p,y\_pred\_elastic))

plt.scatter(x,y,color='blue',label='actual data')

plt.plot(X\_test,y\_pred,color='red',label='simple linear regression')

plt.xlabel('x')

plt.ylabel('y')

plt.legend()

plt.show()

plt.scatter(x,y,color='blue',label='actual data')

plt.plot(X\_test\_p,y\_pred,color='red',label='polynomial linear regression')

plt.xlabel('x')

plt.ylabel('y')

plt.legend()

plt.show()

plt.scatter(x,y,color='blue',label='actual data')

plt.plot(X\_test\_m,y\_pred\_multi,color='red',label='multi linear regression')

plt.xlabel('x')

plt.ylabel('y')

plt.legend()

plt.show()

plt.scatter(x,y,color='blue',label='actual data')

plt.plot(X\_test\_p,y\_pred\_lasso,color='red',label='lasso linear regression')

plt.xlabel('x')

plt.ylabel('y')

plt.legend()

plt.show()

plt.scatter(x,y,color='blue',label='actual data')

plt.plot(X\_test\_p,y\_pred\_ridge,color='red',label='redige linear regression')

plt.xlabel('x')

plt.ylabel('y')

plt.legend()

plt.show()

plt.scatter(x,y,color='blue',label='actual data')

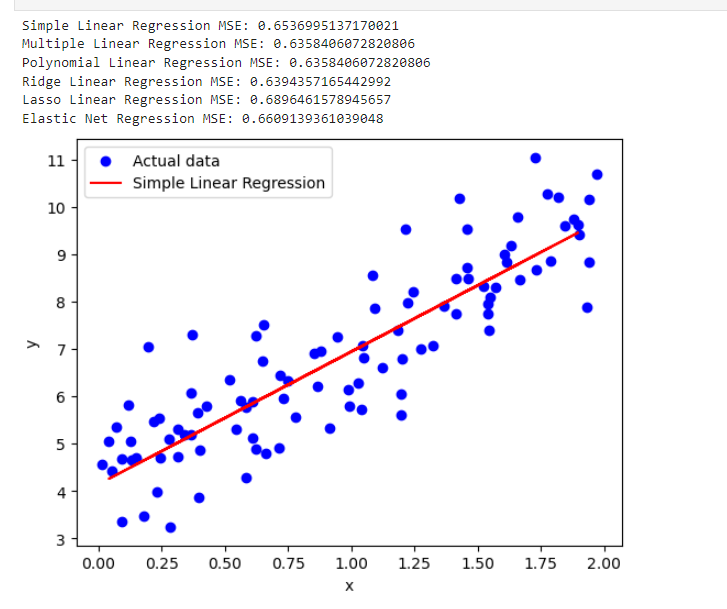
plt.plot(X\_test\_p,y\_pred\_elastic,color='red',label='elastic linear regression')

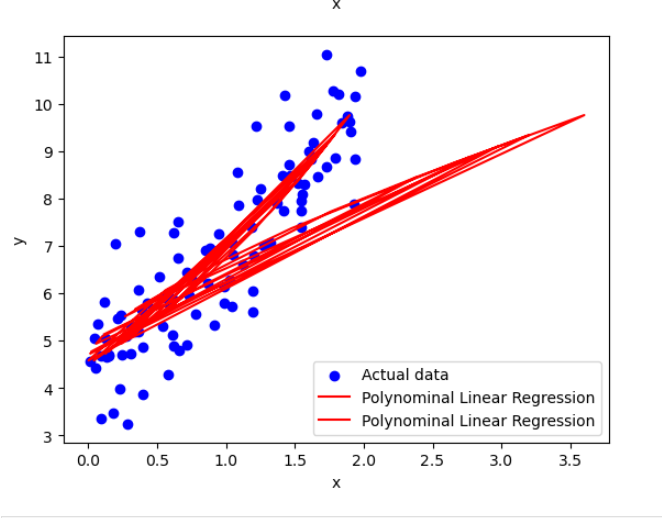
plt.xlabel('x')

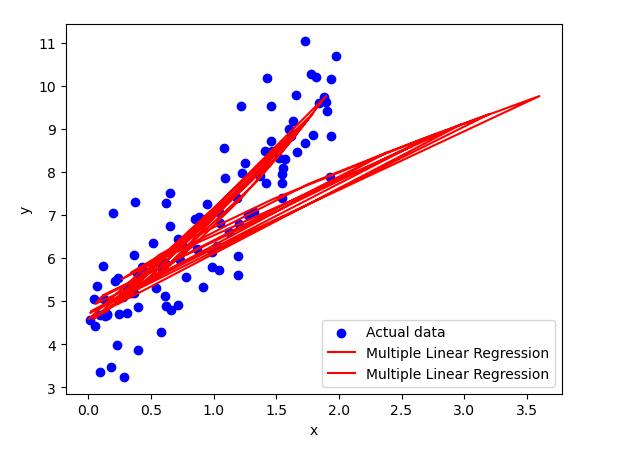
plt.ylabel('y')

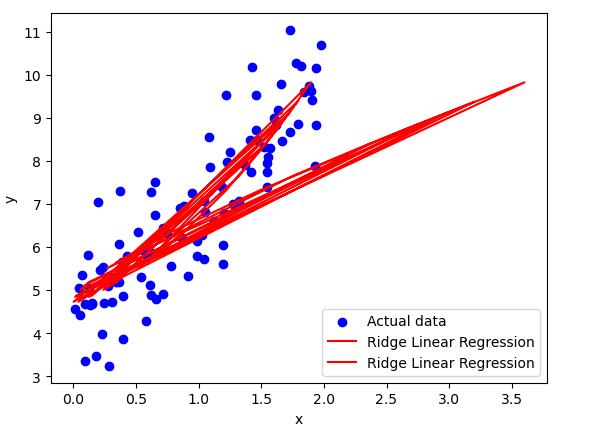
plt.legend()

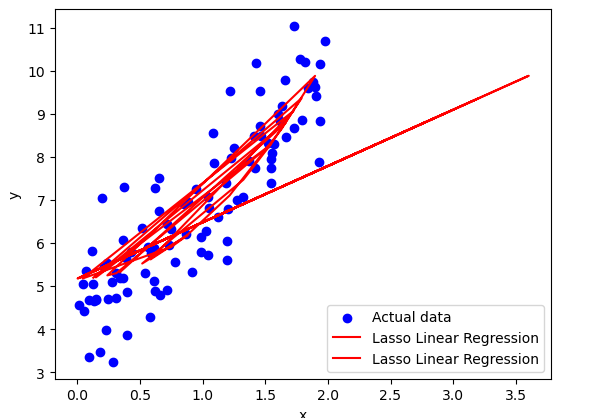
plt.show()

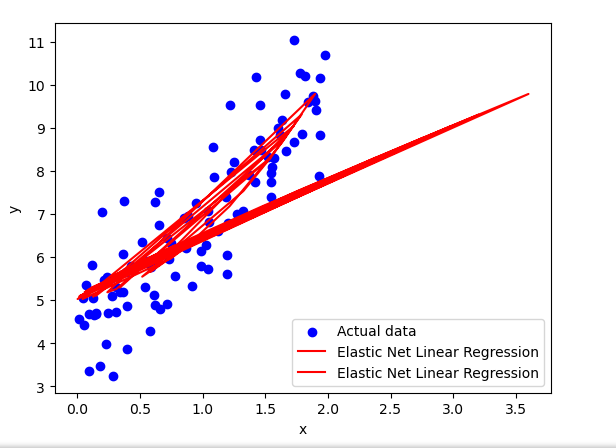












LAB-06

import numpy as np

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import accuracy\_score

from sklearn.datasets import load\_iris

data=load\_iris()

x,y=data.data,data.target

x\_train,x\_test,y\_train,y\_test=train\_test\_split(x,y,test\_size=0.2,random\_state=42)

scaler=StandardScaler()

x\_train=scaler.fit\_transform(x\_train)

x\_test=scaler.transform(x\_test)

#binary logisic regression

binary\_y\_train=(y\_train==1).astype(int)

binary\_y\_test=(y\_test==0).astype(int)

model=LogisticRegression()

model.fit(x\_train,binary\_y\_train)

y\_pred=model.predict(x\_test)

print("bnary logistic regression",accuracy\_score(binary\_y\_test,y\_pred))

#multiclass logistic regression(one vs rest & softmax)

log\_reg=LogisticRegression(multi\_class='multinomial',solver='lbfgs')

log\_reg.fit(x\_train,y\_train)

y\_pred=log\_reg.predict(x\_test)

print("multiclass logistic regression",accuracy\_score(y\_test,y\_pred))

log\_reg\_softmax=LogisticRegression(multi\_class='multinomial',solver='lbfgs',max\_iter=200)

log\_reg\_softmax.fit(x\_train,y\_train)

y\_pred=log\_reg\_softmax.predict(x\_test)

print("multiclass logistic regression with softmax",accuracy\_score(y\_test,y\_pred))

#regularized logistic regression l1,l2,elastic net

log\_reg\_l1=LogisticRegression(penalty='l1',solver='liblinear')

log\_reg\_l1.fit(x\_train,y\_train)

y\_pred=log\_reg\_l1.predict(x\_test)

print("regularized logistic regression l1",accuracy\_score(y\_test,y\_pred))

log\_reg\_l2=LogisticRegression(penalty='l2',solver='liblinear')

log\_reg\_l2.fit(x\_train,y\_train)

y\_pred=log\_reg\_l2.predict(x\_test)

print("regularized logistic regression l2",accuracy\_score(y\_test,y\_pred))

log\_reg\_elastic=LogisticRegression(penalty='elasticnet',solver='saga',l1\_ratio=0.5)

log\_reg\_elastic.fit(x\_train,y\_train)

y\_pred=log\_reg\_elastic.predict(x\_test)

print("regularized logistic regression elastic net",accuracy\_score(y\_test,y\_pred))

OUTPUT:

Binary Logitic Regression Accuracy: 1.0  
one-vs-Rest Logistic Regression Accuracy: 0.9666666666666667  
Softmax Logistic Regression Accuracy: 1.0  
L1 Regularized Logistic Regression Accuracy: 0.9666666666666667  
L2 Regularized Logistic Regression Accuracy: 1.0  
Elastic Net Logistic Regression Accuracy: 1.0

LAB – 07  
import numpy as np import pandas as pd from sklearn.model\_selection import train\_test\_split from sklearn.preprocessing import StandardScaler from sklearn.linear\_model import LogisticRegression from sklearn.metrics import accuracy\_score from sklearn.datasets import load\_iris import matplotlib.pyplot as plt

# #Load dataset (Iris for demonstration)

data = load\_iris() x, y = data.data, data.target # independent variables and dependent variables x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size=0.2, random\_state=42)

# #Standardize features

scaler = StandardScaler() x\_train = scaler.fit\_transform(x\_train) x\_test = scaler.transform(x\_test)

# #1. Binary Logistic Regression

binary\_y\_train = (y\_train == 0).astype(int) binary\_y\_test = (y\_test == 0).astype(int) log\_reg\_binary = LogisticRegression() log\_reg\_binary.fit(x\_train, binary\_y\_train) preds\_binary = log\_reg\_binary.predict(x\_test) acc\_binary = accuracy\_score(binary\_y\_test, preds\_binary)

# #2. Multiclass Logistic Regression (one-vs-Rest & softmax)

log\_reg\_ovr = LogisticRegression(multi\_class='ovr', solver='lbfgs', max\_iter=200) log\_reg\_ovr.fit(x\_train, y\_train) preds\_ovr = log\_reg\_ovr.predict(x\_test) acc\_ovr = accuracy\_score(y\_test, preds\_ovr)

log\_reg\_softmax = LogisticRegression(multi\_class='multinomial', solver='lbfgs', max\_iter=200) log\_reg\_softmax.fit(x\_train, y\_train) preds\_softmax = log\_reg\_softmax.predict(x\_test) acc\_softmax = accuracy\_score(y\_test, preds\_softmax)

# #3. Regularized logistic regression (L1, L2, Elastic Net)

log\_reg\_l1 = LogisticRegression(penalty='l1', solver='liblinear', max\_iter=200) log\_reg\_l1.fit(x\_train, y\_train) preds\_l1 = log\_reg\_l1.predict(x\_test) acc\_l1 = accuracy\_score(y\_test, preds\_l1)

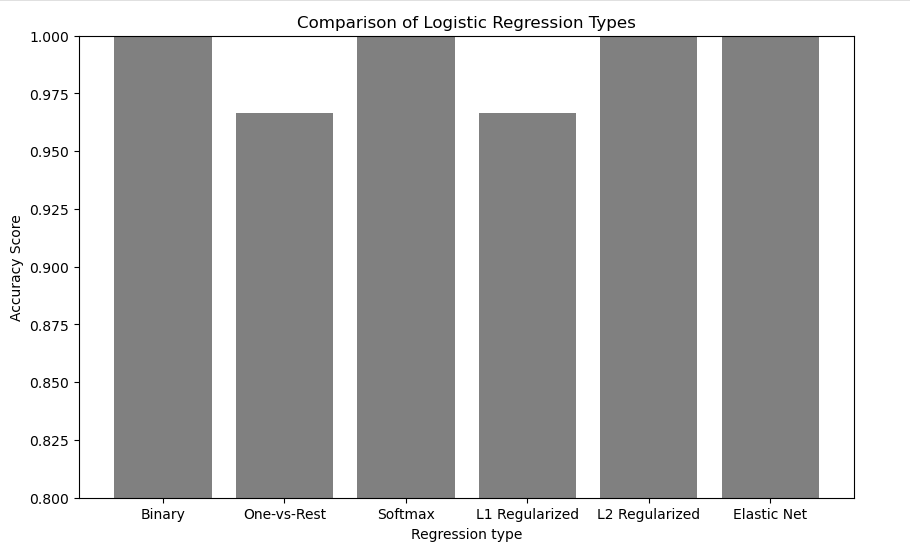
log\_reg\_l2 = LogisticRegression(penalty='l2', solver='lbfgs', max\_iter=200) log\_reg\_l2.fit(x\_train, y\_train) preds\_l2 = log\_reg\_l2.predict(x\_test) acc\_l2 = accuracy\_score(y\_test, preds\_l2)

log\_reg\_elastic = LogisticRegression(penalty='elasticnet', solver='saga', l1\_ratio=0.5, max\_iter=200) log\_reg\_elastic.fit(x\_train, y\_train) preds\_elastic = log\_reg\_elastic.predict(x\_test) acc\_elastic = accuracy\_score(y\_test, preds\_elastic)

# #Plotting

labels = ['Binary', 'One-vs-Rest', 'Softmax', 'L1 Regularized', 'L2 Regularized', 'Elastic Net'] accuracies = [acc\_binary, acc\_ovr, acc\_softmax, acc\_l1, acc\_l2, acc\_elastic]

plt.figure(figsize=(10, 6)) plt.bar(labels, accuracies, color=['grey']) plt.xlabel("Regression type") plt.ylabel("Accuracy Score") plt.title("Comparison of Logistic Regression Types") plt.ylim(0.8, 1.0) plt.show()

Output:  
  


LAB – 08

import numpy as np

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import accuracy\_score, classification\_report

#Generate a syntetic dataset

from sklearn.datasets import make\_classification

X, y = make\_classification(n\_samples=300, n\_features=5, n\_classes=2, random\_state=42)

#split dataset into training and testing sets(80% and 20%)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

scaler = StandardScaler()

X\_train = scaler.fit\_transform(X\_train)

X\_test = scaler.transform(X\_test)

#train knn model

k = 5

knn = KNeighborsClassifier(n\_neighbors=k)

knn.fit(X\_train, y\_train)

#make prediction

y\_pred = knn.predict(X\_test)

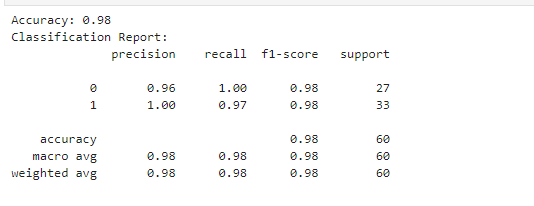
accuracy = accuracy\_score(y\_test, y\_pred)

print(f"Accuracy: {accuracy:.2f}")

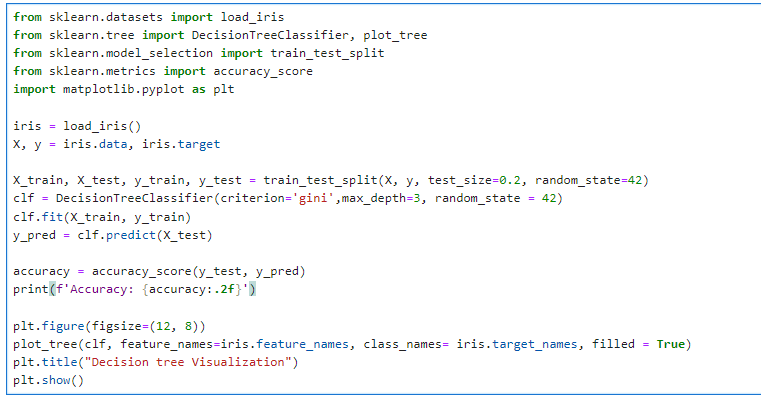
print("Classification Report:")

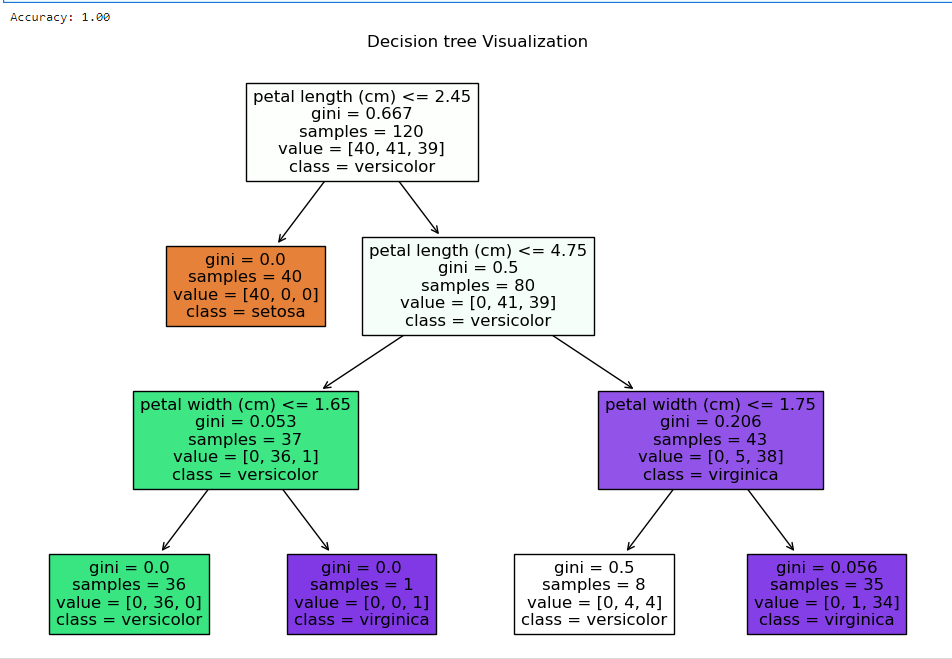
print(classification\_report(y\_test, y\_pred))



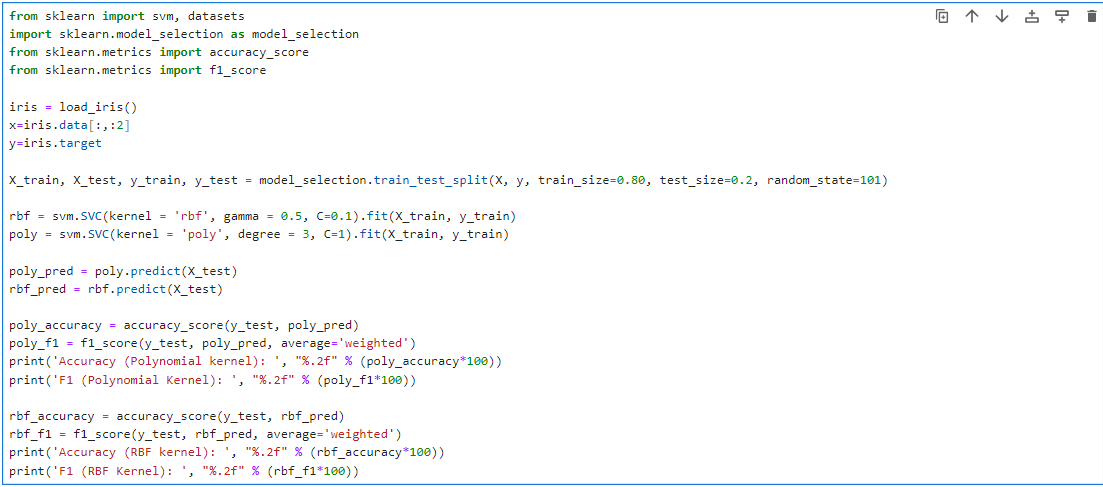


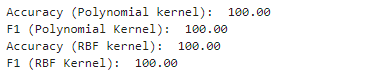
Decision Tree





SVM





Lab-09

K-means

