**Lab 1 :**

**Implement simple linear regression on a dataset and evaluate its performance.**

**Code:**

#simple linear regression lab :

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

#dataset

x = np.array([5,15,25,35,45,55]).reshape((-1,1))#reshape funcn 2 d ma lagna lai

y = np.array([5,20,14,32,22,38])

x.shape #to check the shape

from sklearn.linear\_model import LinearRegression

model=LinearRegression() #creating model

#training model ( fit)

model.fit(x,y)

#print intercept and coefficient

print(model.intercept\_)

print(model.coef\_) #slope

print(f"y={model.intercept\_ :.2f} + {model.coef\_[0]}x") # y=mx+c

#predict

y\_pred=model.predict(x)

y,y\_pred #actual vs predicted

#using regression matrix

from sklearn.metrics import mean\_absolute\_error,mean\_squared\_error,r2\_score

print(f"MAE={mean\_absolute\_error(y,y\_pred):.2f}")#actual vs predicted ko MAE

print(f"MSE={mean\_squared\_error(y,y\_pred):.2f}")#actual vs predicted ko MsE

print(f"r2={r2\_score(y,y\_pred):.2f}")#actual vs predicted ko r2

#plot garna #scatter data print garna lai ho

plt.scatter(x,y)

plt.plot(x,y\_pred,color='violet' ,label='regression line')

plt.xlabel('x')

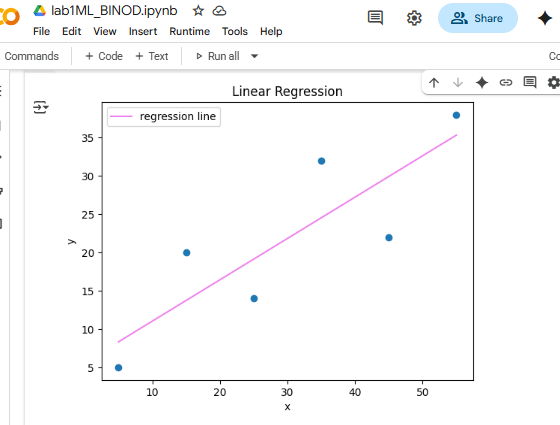
plt.ylabel('y')

plt.title('Linear Regression')

plt.legend()

plt.show()

**Output:**

****

**lab 2:**

**Implement linear regression on a dataset by uploading it and evaluate its performance.**

Code:

import numpy as np

import pandas as pd

import matplotlib as plt

import matplotlib.pyplot as plt#scatter funcn use garna

#load file

from google.colab import files

uploaded = files.upload()

# read data set

salary\_dataset = pd.read\_csv("Salary\_Data.csv")

salary\_dataset.shape#size row and colm

salary\_dataset.head()#top 5 value print garna

salary\_dataset.tail()#last 5 value print garna

x = salary\_dataset.iloc[:,0].values.reshape(-1,1)#first colm

y = salary\_dataset.iloc[:,1].values.reshape(-1,1)#second colm

x,y

# safe method

# x = salary\_dataset.iloc["YearsExperience"].values

# y = salary\_dataset.iloc["Salary"].values

# x,y

# create model

from sklearn.linear\_model import LinearRegression

model=LinearRegression() #creating model

# train or fit a model

model.fit(x,y)

#predict

y\_pred=model.predict(x)

y,y\_pred #actual vs predicted

#model evaluation

#using regression matrix

from sklearn.metrics import mean\_absolute\_error,mean\_squared\_error,r2\_score

print(f"MAE={mean\_absolute\_error(y,y\_pred):.2f}")#actual vs predicted ko MAE

print(f"MSE={mean\_squared\_error(y,y\_pred):.2f}")#actual vs predicted ko MsE

print(f"r2={r2\_score(y,y\_pred):.2f}")#actual vs predicted ko r2

#plot garna #scatter data print garna lai ho

plt.scatter(x,y ,color='red')

plt.plot(x,y\_pred,color='black' ,label='regression line')

plt.xlabel('x')

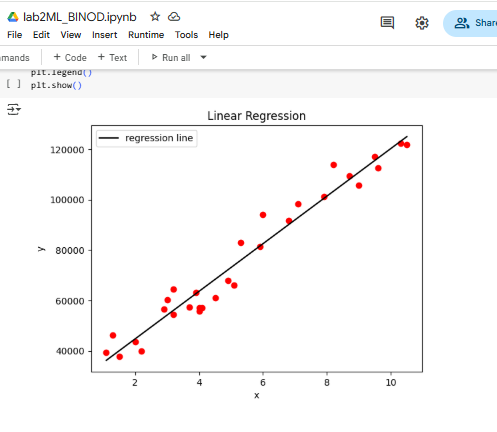
plt.ylabel('y')

plt.title('Linear Regression')

plt.legend()

plt.show()

**Output:**

****

**lab 3:**

**Implement Multiple linear regression on a dataset (e.g., housing prices) and evaluate its performance also plot it.**

**Code:**

#multiple linear regression using housing data,

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.datasets import fetch\_california\_housing #dataset

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

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_squared\_error, r2\_score

# load dta set

housing = fetch\_california\_housing()

#convert to data frame

df = pd.DataFrame(housing.data, columns=housing.feature\_names)

df['target'] = housing.target #target

df.head()

#features and target

X = df.drop('target', axis=1) #drop column except target #features

y = df['target'] #target

#train test split

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

# model create

model = LinearRegression()

# model train

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

# prediction

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

y,y\_pred

#evaluation

mse = mean\_squared\_error(y\_test, y\_pred)

r2 = r2\_score(y\_test, y\_pred)

print(f'Mean Squared Error: {mse}')

print(f'R-squared: {r2}')

# plotting

plt.scatter(y\_test, y\_pred)

plt.plot([y\_test.min(), y\_test.max()], [y\_test.min(), y\_test.max()], 'k--', lw=2 ,color='yellow')

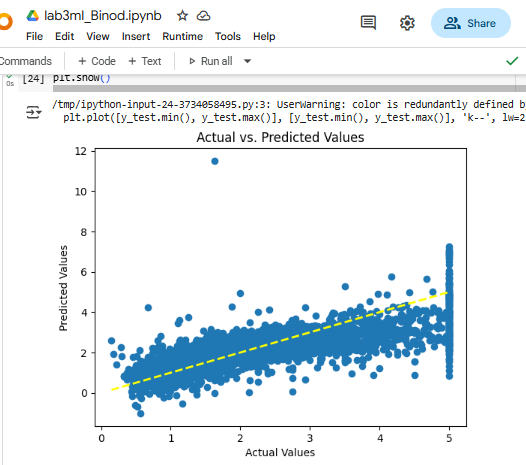
plt.xlabel('Actual Values')

plt.ylabel('Predicted Values')

plt.title('Actual vs. Predicted Values')

plt.show()

**output :**



**Lab 4 :**

**Implement linear regression on a dataset (e.g., housing prices) and evaluate its performance. Apply ridge and lasso regression to prevent overfitting and compare results.**

**Code:**

# linear regression using lasso and ridge regularization

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.datasets import fetch\_california\_housing

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

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

from sklearn.metrics import r2\_score,mean\_squared\_error

#loading dataset

housing=fetch\_california\_housing()

#convert dataset into datafrfame

df=pd.DataFrame(housing.data,columns=housing.feature\_names)

df['target']=housing.target

#feature and target

X=df.drop('target',axis=1)

y=df['target']

#TRAIN TEST SPLIT

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

#USING linear Regression(NO regularization)

lr=LinearRegression()

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

lr\_pred=lr.predict(X\_test)   #y predict gareko ho esma

op:

R2 score: 0.5757877060324508

MSE: 0.5558915986952444

# metrics using linear regression ,also print (w/o regularization)

print('R2 score:',r2\_score(y\_test,lr\_pred))

print('MSE:',mean\_squared\_error(y\_test,lr\_pred))

# using lasso regularization

lasso=Lasso(alpha=1)

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

lasso\_pred=lasso.predict(X\_test)

#metrics using linear regression

print('R2 score:',r2\_score(y\_test,lasso\_pred))

print('MSE:',mean\_squared\_error(y\_test,lasso\_pred))

**output:**

**R2 score: 0.2841671821008396**

**MSE: 0.9380337514945427**

# using ridge regularization

ridge=Ridge(alpha=1)

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

ridge\_pred=ridge.predict(X\_test)

#metrics using linear regression

print('R2 score:',r2\_score(y\_test,ridge\_pred))

print('MSE:',mean\_squared\_error(y\_test,ridge\_pred))

**output:**

**R2 score: 0.5758549611440126**

**MSE: 0.5558034669932211**

**Lab 5 :**

**Implement classification using logistic regression on a dataset and evaluate its performance by Visualizing Confusion matrix.**

**Code:**

#classification using logistic regression

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.datasets import load\_iris

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

from sklearn.linear\_model import LogisticRegression

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

from sklearn.preprocessing import StandardScaler #for standarization

#load dataset

dataset=load\_iris()

x = dataset.data

y = dataset.target

# standarzation dataset

sc = StandardScaler()

x = sc.fit\_transform(x)

# train test split

x\_train,x\_test,y\_train,y\_test = train\_test\_split(x,y,test\_size=0.2,random\_state=0)

#create logistic model

model = LogisticRegression(max\_iter=1000)

model.fit(x\_train,y\_train)

# predict the value

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

# evaluation metrics

print("confusion\_matrix: ",confusion\_matrix(y\_test,y\_pred))

print("accuracy\_score : ",accuracy\_score(y\_test,y\_pred)\*100) # ACCURACY OUT OF 100 %

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

# Visualize OF confusion MATRIX

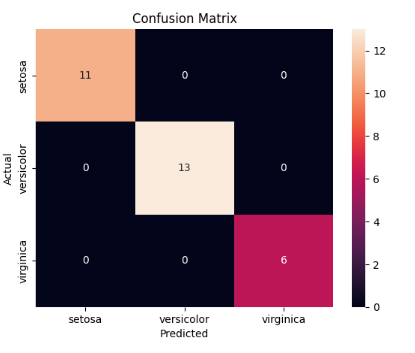
sns.heatmap(confusion\_matrix(y\_test,y\_pred),annot=True,fmt='d',xticklabels=dataset.target\_names,yticklabels=dataset.target\_names )

plt.title("Confusion Matrix")

plt.xlabel("Predicted")

plt.ylabel("Actual")

plt.show()

**Output:**

**Lab 6 :**

**Implement support vecctor machine for classification on a dataset and evaluate its performance by Visualizing Confusion matrix.**

**Code:**

#implement support vecctor machine for classification

import numpy as np

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.datasets import load\_iris

from sklearn.svm import SVC

from sklearn.preprocessing import StandardScaler

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

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

# load dataset

dataset = load\_iris()

x=dataset.data

y=dataset.target

# standardization

sc=StandardScaler()

x=sc.fit\_transform(x)

# train test split

x\_train,x\_test,y\_train,y\_test=train\_test\_split(x,y,test\_size=0.2,random\_state=0)

# create and train svc

svc=SVC(kernel='rbf',C=1.0,gamma='scale')#Radaial bias

svc.fit(x\_train,y\_train)

# predict

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

# evaluation

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

print(confusion\_matrix(y\_test,y\_pred))

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

# conhfusion matrix using heatmap

plt.figure(figsize=(4,2))

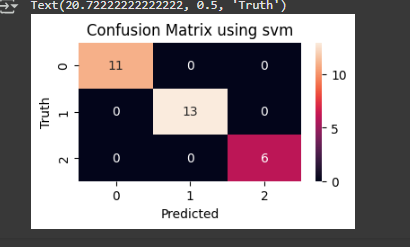
sns.heatmap(confusion\_matrix(y\_test,y\_pred),annot=True)

plt.title('Confusion Matrix using svm')

plt.xlabel('Predicted')

plt.ylabel('Truth')

**Output:**



**Lab 7 :**

**Implement K\_fold cross validation for RandomForest Classification on a dataset and evaluate its performance by Visualizing Confusion matrix.**

**Code:**

#K\_fold cross validation for RandomForest Classification

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.datasets import load\_iris

from sklearn.model\_selection import cross\_val\_score,cross\_val\_predict,KFold

from sklearn.ensemble import RandomForestClassifier

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

#load Dataset

dataset = load\_iris()

x = dataset.data

y = dataset.target

#initialize model

rf\_model = RandomForestClassifier(n\_estimators=100)

#k\_fold cross validation

kf = KFold(n\_splits=5, shuffle=True, random\_state=42)

cv\_results = cross\_val\_score(rf\_model, x, y, cv=kf, scoring='accuracy')

print(cv\_results)

**Output :**

[1. 0.96666667 0.93333333 0.93333333 0.96666667]

#prediction

y\_pred = cross\_val\_predict(rf\_model, x, y, cv=kf)

#Evaluation matric

accuracy = accuracy\_score(y, y\_pred)\*100

confusion\_mat = confusion\_matrix(y, y\_pred)

classification\_rep = classification\_report(y, y\_pred)

print("Accuracy:", accuracy)

print("Confusion Matrix:\n", confusion\_mat)

print("Classification Report:\n", classification\_rep)

#Visualization using heatmap

plt.figure(figsize=(8, 6))

sns.heatmap(confusion\_mat, annot=True, fmt='d', cmap='Blues', xticklabels=dataset.target\_names, yticklabels=dataset.target\_names)

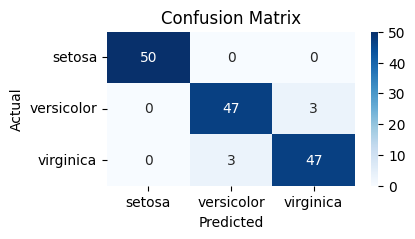
plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.title('Confusion Matrix')

plt.show()

**Output :**



**Lab 8 :**

**Implement k-mean clustering using PCA , evaluate its performance also plot it.**

**Code:**

# kmean clustering using pca\

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import sklearn.datasets as load\_iris

from sklearn.decomposition import PCA

from sklearn.cluster import KMeans

from sklearn.metrics import silhouette\_score

from sklearn.preprocessing import StandardScaler

# load dataset

iris = load\_iris.load\_iris()

x = iris.data

y = iris.target

# standarization

x = StandardScaler().fit\_transform(x)

# reduced 2d to pca

pca = PCA(n\_components=2)

x\_pca = pca.fit\_transform(x)

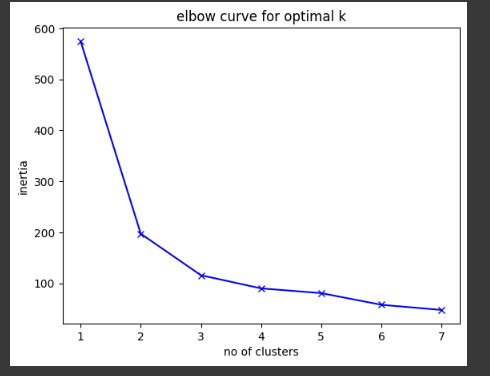
print("explained variance ratio (first two components): %s"

      % str(pca.explained\_variance\_ratio\_)) #for  % in component

# computer inertia ( within cluster sum of squares for differnet k , elbow method)

inertia =[]

for k in range(1,8):

    kmeans = KMeans(n\_clusters=k)

    kmeans.fit(x\_pca)

    inertia.append(kmeans.inertia\_)

# plot elbow curve

plt.plot(range(1,8),inertia,'bx-')

plt.xlabel('no of clusters')

plt.ylabel('inertia')

plt.title('elbow curve for optimal k')

plt.show()

# kmeans clustering

kmeans = KMeans(n\_clusters=3,random\_state=42)

kmeans.fit(x\_pca)

# evaluate silhoutte score

score=silhouette\_score(x\_pca,kmeans.labels\_)

print(f"silhoutte score : ",{score})

plt.figure(figsize=(4,2))

plt.scatter(x\_pca[:,0],x\_pca[:,1],c=y)

plt.xlabel('pca 1')

plt.ylabel('pca 2')

plt.title('kmeans cluster for Actual labels')

plt.show()

plt.figure(figsize=(4,2))

plt.scatter(x\_pca[:,0],x\_pca[:,1],c=kmeans.labels\_)

plt.scatter(kmeans.cluster\_centers\_[:,0],kmeans.cluster\_centers\_[:,1],marker='X',s=100,c='red')

plt.xlabel('pca 1')

plt.ylabel('pca 2')

plt.title('k-means clustering')

plt.show()

