PROGRAM-06

```
import numpy as np
import matplotlib.pyplot as plt

def local_regression(x0, X, Y, tau):
    x0 = [1, x0]
    X = [[1, i] for i in X]
    X = np.asarray(X)
    W=np.diag(np.exp(-np.sum((X-x0)**2,axis=1)/(2*tau*tau)))
    beta = np.linalg.pinv(X.T@ W @ X)@X.T@ W @Y
    y_pred=np.dot(beta,x0)
    return y_pred

def draw(tau):
    prediction = [local_regression(x0, X, Y, tau) for x0 in domain]
    plt.plot(X, Y, 'o', color = 'black')
    plt.plot(domain, prediction, color = 'red')
    plt.show()
X = np.linspace(-3, 3, num = 100)
domain = X
Y = np.log(np.abs(X ** 2 - 1)+ .5)
print("X values:",X)
print("Y values:",Y)
print("\n Regression Line Fit for different values of Tau- 10,0.1,0.01,0.001")
draw(10)
draw(0.1)
draw(0.01)
draw(0.001)
```

PROGRAM-07

7a)

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score

# Load dataset
data = pd.read_csv("boston_housing.csv")
print(data.head())
print(data.shape)
print(data.info())
```

```
X = data.drop('MEDV', axis=1) # ALL columns except 'MEDV'
y = data['MEDV'] # Target column

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train model
model = LinearRegression()
model.fit(X_train, y_train)

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

mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_pred)

print("Mean Squared Error:", mse)
print("Root Mean Squared Error (RMSE):", rmse)
print("R² Score:", r2)


rm_model = LinearRegression()
rm_model.fit(data[['RM']], data['MEDV'])
data['Predicted'] = rm_model.predict(data[['RM']])


sns.set(style='whitegrid')
sns.scatterplot(x='RM', y='MEDV', data=data, label='Actual data', color='blue', alpha=0.5)
sns.lineplot(x='RM', y='Predicted', data=data, label='Regression Line (RM only)', color='red')
plt.title("Linear Regression - RM vs MEDV (Visualization)")
plt.xlabel("Average Number of Rooms (RM)")
plt.ylabel("Median Home Value (MEDV)")
plt.legend()
plt.show()
```

7b)

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.model_selection import train_test_split
```

```
from sklearn.metrics import mean_squared_error, r2_score

# Load the dataset
data = pd.read_csv("mpg.csv")
print(data.head())
print(data.shape)
print(data.info())

# Drop rows with missing values
data.dropna(inplace=True)

# Convert 'horsepower' column to numeric
data['horsepower'] = data['horsepower'].astype(float)

# Select feature and target
X = data[['horsepower']]
y = data['mpg']

# Transform feature to polynomial (degree 2)
poly = PolynomialFeatures(degree=2)
X_poly = poly.fit_transform(X)

# Split into training and testing data
X_train, X_test, y_train, y_test = train_test_split(X_poly, y, test_size=0.2, random_state=42)

# Create and train the model
model = LinearRegression()
model.fit(X_train, y_train)

# Predict
y_pred = model.predict(X_test)

# Evaluate
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_pred)

print("Mean Squared Error:", mse)
print("Root Mean Squared Error (RMSE):", rmse)
print("R² Score:", r2)

# Create smooth curve for plotting
X_range = pd.DataFrame({'horsepower': range(int(X.min()), int(X.max())+1)})
X_range_poly = poly.transform(X_range)
y_range_pred = model.predict(X_range_poly)

# Plot
sns.set(style='whitegrid')
```

```
sns.scatterplot(x='horsepower', y='mpg', data=data, label='Actual data', alpha=0.4,
color='blue')
sns.lineplot(x='horsepower', y=y_range_pred, data=X_range, label='Polynomial Fit',
color='red')
plt.title("Polynomial Regression - Auto MPG (Horsepower vs MPG)")
plt.xlabel("Horsepower")
plt.ylabel("Miles Per Gallon (MPG)")
plt.legend()
plt.show()
```

PROGRAM-8

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier,plot_tree
from sklearn.metrics import accuracy_score,classification_report,confusion_matrix
data=load_breast_cancer()
x=data.data
y=data.target
df=pd.DataFrame(data.data,columns=data.feature_names)
df['diagnosis']=data.target
print("Head of dataset:")
print(df.head())
print("\nHead of dataset:")
print(df.info())
print("\nDataset Shape:")
print(df.shape)
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=42)
clf=DecisionTreeClassifier(criterion='entropy',random_state=42)
clf.fit(x_train,y_train)
y_pred=clf.predict(x_test)
accuracy=accuracy_score(y_test,y_pred)
print("\nAccuracy:",accuracy)
print("\nClassification Report:")
print(classification_report(y_test,y_pred,target_names=data.target_names))
print("\nConfusion Matrix:")
print(confusion_matrix(y_test,y_pred))
new_sample=np.array([[12.5,19.2,80.0,500.0,0.085,0.1,0.05,0.02,0.17,0.06,0.4,1.0,2.5,40.0,
0.25,0.31,0.15,0.006,0.02,0.03,16.0,25.0,105.0,900.0,0.13,0.25,0.28,0.12,0.29,0.08]])
prediction=clf.predict(new_sample)
print("\nNew sample prediction:")
print("Class:",data.target_names[prediction][0])
plt.figure(figsize=(20,10))
plot_tree(clf,filled=True,feature_names=data.feature_names,class_names=data.target_nam
es,fontsize=9)
```

```
plt.title("Decision Tree Visulaiztion(Entropy)",fontsize=16)
plt.show()
```

PROGRAM-9

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

data=fetch_olivetti_faces()
data.keys()

print("Data shape",data.data.shape)
print("Target Shapes",data.target.shape)
print("There are {} unique persons in the dataset".format(len(np.unique(data.target))))
print("Size of each image is {}x{}".format(data.images.shape[-1],data.images.shape[1]))

def print_faces(images,target,top_n):
        top_n=min(top_n,len(images))
        grid_size=int(np.ceil(np.sqrt(top_n)))
        fig,axes=plt.subplots(grid_size,grid_size,figsize=(15,15))
        fig.subplots_adjust(left=0,right=1,bottom=0,top=1,hspace=.2,wspace=.2)

        for i ,ax in enumerate(axes.ravel()):
        if i<top_n:
        ax.imshow(images[i],cmap="bone")
        ax.axis("off")
        ax.text(2,12,str(target[i]),fontsize=9,color="red")
        ax.text(2,55,f"face:{i}",fontsize=9,color="blue")
        else:
        ax.axis('off')
        plt.show()
print_faces(data.images,data.target,400)


def display_unique_faces(pics):
        fig=plt.figure(figsize=(24,10))
        col,row=10,4

        for i in range(1,col*row+1):
        img_index=10*i-1
        if img_index<pics.shape[0]:
        img=pics[img_index,:,:]
        ax=fig.add_subplot(row,col,i)
        ax.imshow(img,cmap="gray")
        ax.set_title(f"Person {i}",fontsize=14)
        ax.axis('off')
```

```
        plt.suptitle("There are 40 distinct person faces are there in the dataset",fontsize=24)
        plt.show()

display_unique_faces(data.images)

from sklearn.model_selection import train_test_split
x=data.data
y=data.target
xtrain,xtest,ytrain,ytest=train_test_split(x,y,test_size=.3,random_state=46)

print("x train :",xtrain.shape)
print("x test :",xtest.shape)

from sklearn.naive_bayes import GaussianNB,MultinomialNB
from sklearn.metrics import confusion_matrix,accuracy_score,classification_report

nb=GaussianNB()
nb.fit(xtrain,ytrain)

ypred=nb.predict(xtest)

nbaccuracy=round(accuracy_score(ytest,ypred)*100,2)
cm=confusion_matrix(ytest,ypred)
print("confusion matrix:",cm)

print(f"Naive bayes accuracy:{nbaccuracy}")

nb=MultinomialNB()
nb.fit(xtrain,ytrain)

ypred=nb.predict(xtest)
accuray=round(accuracy_score(ytest,ypred)*100,2)
print(f"Multinominal Naive Bayes acccuracy:{accuracy}")

missclass=np.where(ypred!=ytest)[0]
nummissclass=len(missclass)
print("number of missclassified images",nummissclass)
print("Total images in test set",len(ytest))
print("Accuray:",round((1-nummissclass/len(ytest))*100,2),"%")

nmissclass=min(nummissclass,5)
plt.figure(figsize=(10,5))
for i in range(nmissclass):
        idx=missclass[i]
        plt.subplot(1,nmissclass,i+1)
        plt.imshow(xtest[idx].reshape(64,64),cmap="gray")
        plt.axis("off")
plt.show()
```

PROGRAM-10

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import load_breast_cancer
# Load the Wisconsin Breast Cancer dataset
data = load_breast_cancer()
df = pd.DataFrame(data.data, columns=data.feature_names)
actual_labels = data.target # 0 = malignant, 1 = benign
print(df.head())
print(df.info())
print(df.shape)
# Standardize the dataset
scaler = StandardScaler()
df_scaled = scaler.fit_transform(df)
# Apply K-Means clustering
kmeans = KMeans(n_clusters=2, random_state=42, n_init=10)
kmeans.fit(df_scaled)
cluster_labels = kmeans.labels_

# Reduce dimensions using PCA for visualization
pca = PCA(n_components=2)
df_pca = pca.fit_transform(df_scaled)

# Create a DataFrame for visualization
df_visual = pd.DataFrame(df_pca, columns=['PC1', 'PC2'])
df_visual['Cluster'] = cluster_labels
df_visual['Actual'] = actual_labels

# Plot clusters and actual diagnoses
plt.figure(figsize=(12, 5))

# K-Means Clusters
plt.subplot(1, 2, 1)
sns.scatterplot(x='PC1', y='PC2', hue='Cluster', data=df_visual, palette='Set1')
plt.title('K-Means Clustering on Wisconsin Breast Cancer Dataset')

# Actual Diagnosis
plt.subplot(1, 2, 2)
sns.scatterplot(x='PC1', y='PC2', hue='Actual', data=df_visual, palette='Set2')
plt.title('Actual Diagnosis')
plt.tight_layout()
plt.show()
```