EX: 1

Linear Regression

DATE:

AIM:

To write a python program to implement linear regression

1) Using datapoint.
2) Using dataset.

ALGORITHM:

STEP 1: Start by importing the necessary libraries

STEP 2: Load the dataset into a Pandas DataFrame

STEP 3: Assuming you have a CSV file with two columns named 'x' and 'y', where 'x' represents the input variable and 'y' represents the target variable. You can split the dataset into training and testing data using the train_test_split function

STEP 4: Create an instance of the LinearRegression class

STEP 5: We can fit the regressor to our training data

STEP 6: We can now use the trained model to make predictions on our test data

STEP 7: Let's evaluate the performance of our model using mean squared error(MSE) and the coefficient of determination (R2)

Program Code:

1) Using datapoints:

```
import numpy as np
import matplotlib.pyplot as plt
def estimate_coef(x, y):
    # number of observations/points
    n = np.size(x)
    # mean of x and y vector
    m_x = np.mean(x)
    m_y = np.mean(y)
    # calculating cross-deviation and deviation about x
    s_xy = np.sum(y*x) - n*m_y*m_x
    s_xx = np.sum(x*x) - n*m_x*m_x
    # calculating regression coefficients
    k_1 = s_xy / s_xx
    k_0 = m_y - k_1*m_x
    return (k_0, k_1)
def plot_regression_line(x, y, k):
    # plotting the actual points as scatter plot
    plt.scatter(x, y, color = "r",
            marker = "o", s = 50)
    # predicted response vector
    y_pred = k[0] + k[1]*x
```

```
    # plotting the regression line
    plt.plot(x, y_pred, color = "r")


    # putting labels
    plt.xlabel('x')
    plt.ylabel('y')
    # function to show plot
    plt.show()
def main():
    # observations / data
    x = np.array([1,2,3,4,5,6,7,8,9,10])
    y = np.array([0,1,12,13,14,5,16,7,8,9])
    # estimating coefficients
    k = estimate_coef(x, y)
    print("Estimated coefficients:\nk_0 = {}  \
        \nk_1 = {}".format(k[0], k[1]))
    # plotting regression line
    plot_regression_line(x, y, k)
if__name__ == "_main_":
    main()
```

2) <u>Using dataset:</u>

```
import pandas as pd
```

```
import numpy as np
import matplotlib.pyplot as plt
from google.colab import files
uploaded = files.upload()
import io
# reading csv file as pandas dataframe
data = pd.read_csv(io.BytesIO(uploaded['headbrain.csv']))
data.head()
x=data.iloc[:,2:3].values
y=data.iloc[:,2:3].values
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=1/4,random_state=0)

#fitting simple linear regression to the training set
from sklearn.linear_model import LinearRegression
regressor=LinearRegression()
regressor.fit(x_train,y_train)

#predict the test result
y_pred=regressor.predict(x_test)
#to see the relationship between the training data values
plt.scatter(x_train,y_train,c='red')
plt.show()
```

```
#to see the relationship between the predicted brain weight values using scattered grap
h

plt.plot(x_test,y_pred)

plt.scatter(x_test,y_test,c='red')

plt.xlabel('headsize')

plt.ylabel('brain weight')
```

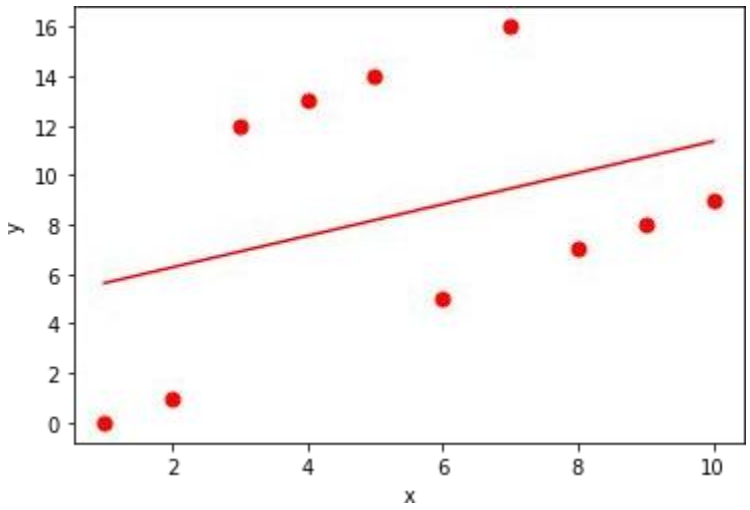
OUTPUT:

USING DATAPOINT:

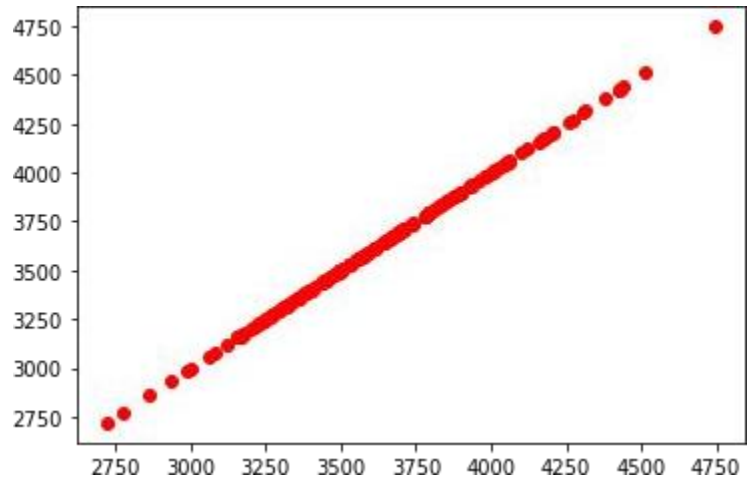
Estimated coefficients:

k_0 = 5.0

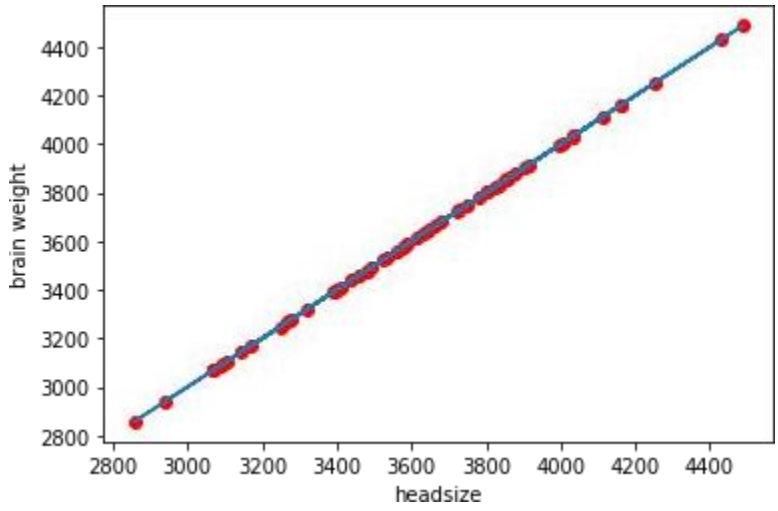
k_1 = 0.6363636363636364



USING DATASET:



Text(0, 0.5, 'brain weight')



RESULT:

Thus,to write a python program to implement linear regression is successfully created.

EX: 2

Polynomial Regression

DATE:

AIM:

To write a python program to implement polynomial regression

ALGORITHM:

STEP 1: Start by importing the necessary libraries

STEP 2: Prepare the data.

STEP 3: Reshape the data into a 2D array.

STEP 4: Create a Polynomial Features object and transform the input feature

STEP 5: Create a Linear Regression object and fit the transformed data

STEP 6: We can now use the trained model to make predictions on our test data

STEP 7: Let's evaluate the performance of our model using mean squared error(MSE) and the coefficient of determination (R2)

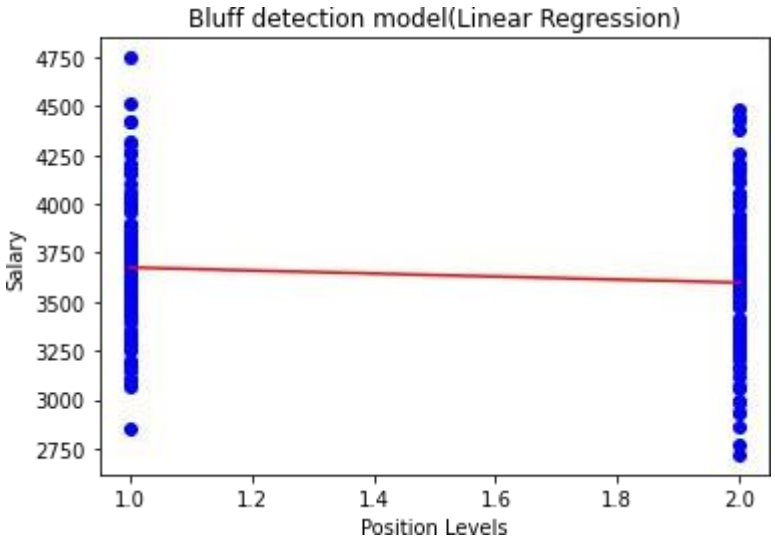
Program Code:

```
import numpy as nm

import matplotlib.pyplot as mtp

import pandas as pd
```

```
from google.colab import files

uploaded = files.upload()

import io


# reading csv file as pandas dataframe

data = pd.read_csv(io.BytesIO(uploaded['Position_Salaries.csv']))

data.head()


#Extracting Independent and dependent Variable

x= data_set.iloc[:, 1:2].values

y= data_set.iloc[:, 2].values


from sklearn.linear_model import LinearRegression

lin_regs= LinearRegression()

lin_regs.fit(x,y)


from sklearn.preprocessing import PolynomialFeatures

poly_regs= PolynomialFeatures(degree= 2)

x_poly= poly_regs.fit_transform(x)

lin_reg_2 =LinearRegression()

lin_reg_2.fit(x_poly, y)


from sklearn.preprocessing import PolynomialFeatures
```

```
poly_regs= PolynomialFeatures(degree= 3)

x_poly= poly_regs.fit_transform(x)

lin_reg_2 =LinearRegression()

lin_reg_2.fit(x_poly, y)


mtp.scatter(x,y,color="blue")

mtp.plot(x,lin_regs.predict(x), color="red")

mtp.title("Bluff detection model(Linear Regression)")

mtp.xlabel("Position Levels")

mtp.ylabel("Salary")

mtp.show()


poly_pred = lin_reg_2.predict(poly_regs.fit_transform([[6.5]]))

print(poly_pred)
```

OUTPUT:



[1.22711932e+16]

RESULT:

Thus,to write a python program to implement polynomial regression is successfully created.

EX: 3

Decision Tree Classification

DATE:

AIM:

To write a python program to implement Decision tree classification.

ALGORITHM:

STEP 1: Start by importing the necessary libraries

STEP 2: Prepare the data.

STEP 3: Reshape the data into a 2D array.

STEP 4: Choose the input feature to split the dataset

STEP 5: Create a decision node based on the selected input feature. The decision node contains a test condition that splits the dataset into two or more subsets.

STEP 6: Prune the decision tree,the decision tree may be pruned to prevent overfitting. Pruning involves removing decision nodes that do not improve the accuracy of the model on the testing dataset.

STEP 7: Use the decision tree to predict the class label of new instances:Use the decision tree to predict the class label of new instances.

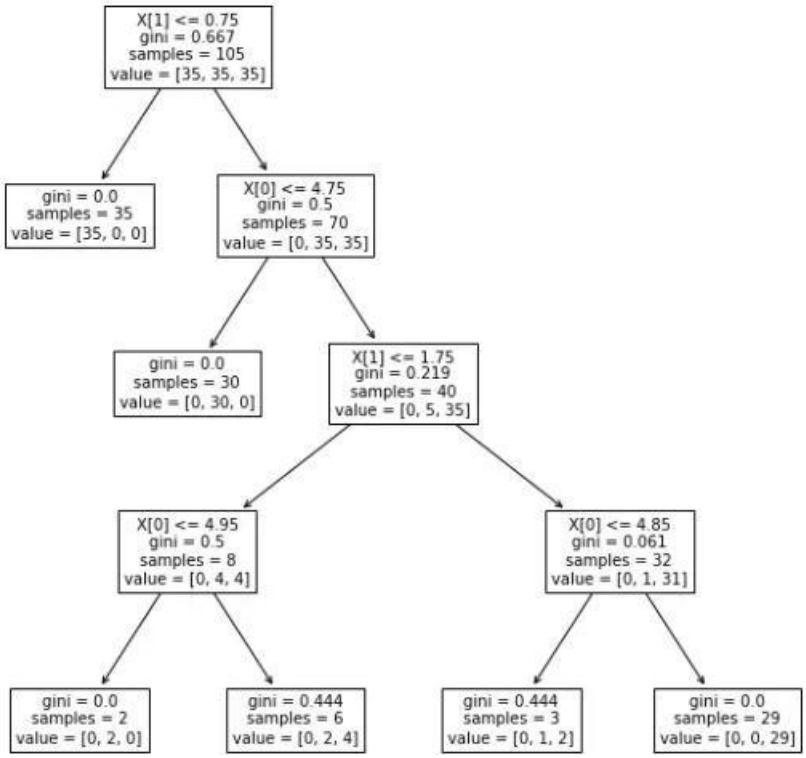
Program Code:

```
import numpy as nm

import matplotlib.pyplot as mtp
```

```
import pandas as pd
from google.colab import files
uploaded = files.upload()
import io
# reading csv file as pandas dataframe
data = pd.read_csv(io.BytesIO(uploaded['heart.csv']))
data.head()
x= data.iloc[:, [2,3]].values
y= data.iloc[:, 4].values
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test= train_test_split(x, y, test_size= 0.25, random_state=0)


#feature Scaling
from sklearn.preprocessing import StandardScaler
st_x= StandardScaler()
x_train= st_x.fit_transform(x_train)
x_test= st_x.transform(x_test)
from sklearn.tree import DecisionTreeClassifier
classifier= DecisionTreeClassifier(criterion='entropy', random_state=0)
classifier.fit(x_train, y_train)
y_pred= classifier.predict(x_test)
y_test= classifier.predict(x_test)
from sklearn.metrics import confusion_matrix
```

```
cm= confusion_matrix(y_test, y_pred)

from matplotlib.colors import ListedColormap

x_set, y_set = x_train, y_train

x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop = x_set[:, 0].max()
+ 1, step  =0.01),

nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))

mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(),
x2.ravel()]).T).reshape(x1.shape),

alpha = 0.75, cmap = ListedColormap(('purple','green' )))

mtp.xlim(x1.min(), x1.max())

mtp.ylim(x2.min(), x2.max())

for i, j in enumerate(nm.unique(y_set)):

 mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],c = ListedColormap(('purple',
'green'))(i), label = j)

 mtp.title('Decision Tree Algorithm (Training set)')

 mtp.xlabel('Skewness')

 mtp.ylabel('Cutosis')

 mtp.legend()

 mtp.show()
```

OUTPUT:



[1.22711932e+16]

RESULT:

Thus,to write a python program to implement Decision Tree classification is successfully created.

EX: 4

Logistic regression

DATE:

AIM:

To write a python program to implement logistic regression

ALGORITHM:

STEP 1: Start by importing the necessary libraries

STEP 2: Prepare the data.

STEP 3: Split the data into two sets

STEP 4: Choose the logistic regression model and the learning algorithm you want to use. The most common algorithm used is gradient descent.

STEP 5: Train the model using the training set. This involves finding the optimal values for the coefficients of the independent variables.

STEP 6: Evaluate the performance of the model using the testing set

STEP 7: Interpret the coefficients of the independent variables to determine which variables are most strongly associated with the dependent variable.

Program Code:

```
import pandas as pd

from sklearn.linear_model import LogisticRegression
```

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from sklearn.datasets import load_wine
wine=load_wine()
df=pd.DataFrame(data=wine.data,columns=wine.feature_names)
df['target']=wine.target
df
# Drop the missing values
df.dropna(inplace=True)

# Convert categorical variables to numerical variables
df= pd.get_dummies(df, drop_first=True)

# Split the dataset into the predictor and target variables
X = df.drop('target', axis=1)
y = df['target']
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=25, random_state=42)
from sklearn.linear_model import LogisticRegression

# Instantiate the model
logreg = LogisticRegression().fit(X_train, y_train)
```

```
# Fit the model to the training data
#logreg.
from sklearn.metrics import accuracy_score, confusion_matrix

# Predict the target variable for the testing data
y_pred = logreg.predict(X_test)

# Calculate the accuracy score
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

# Calculate the confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
print("Confusion matrix:\n", conf_matrix)
print(classification_report(y_test, y_pred))
```

OUTPUT:

```
Accuracy: 0.96
Confusion matrix:
 [[ 8  1  0]
 [ 0 10  0]
 [ 0  0  6]]
              precision    recall  f1-score   support

           0       1.00      0.89      0.94         9
           1       0.91      1.00      0.95        10
           2       1.00      1.00      1.00         6

    accuracy                           0.96        25
   macro avg       0.97      0.96      0.96        25
weighted avg       0.96      0.96      0.96        25
```

[1.22711932e+16]

RESULT:

Thus,to write a python program to implement Logistic regression is successfully created.

EX: 5 # EDA(Exploratory Data Analysis)

DATE:

AIM:

To write a python program to implement EDA(Exploratory Data Analysis).

ALGORITHM:

STEP 1: Start by importing the necessary libraries

STEP 2: Prepare the data.

STEP 3: Data Cleaning: In this step, you will clean and preprocess the data to ensure it's ready for analysis. This includes dealing with missing values, removing duplicates, and checking for outliers.

STEP 4: Exploratory Data Analysis: This is the core step of the analysis. You will use various statistical and visualization techniques to understand the data and draw insights from it.

STEP 5: You can use various communication tools like reports, dashboards, and visualizations to convey the insights and recommendations effectively.

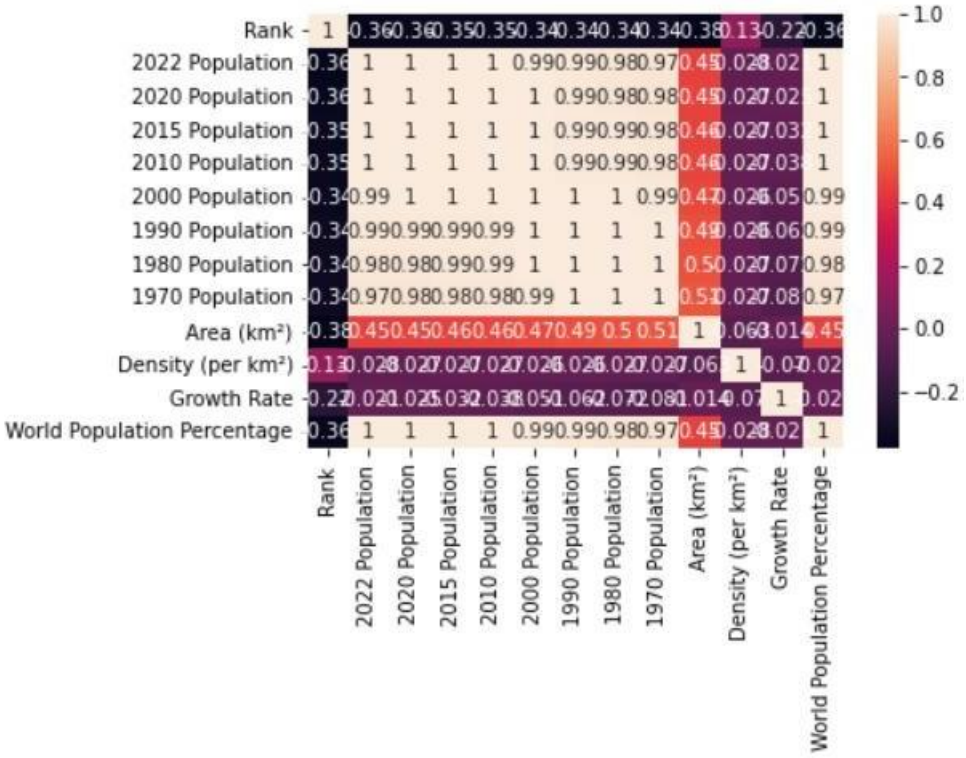
Program Code:

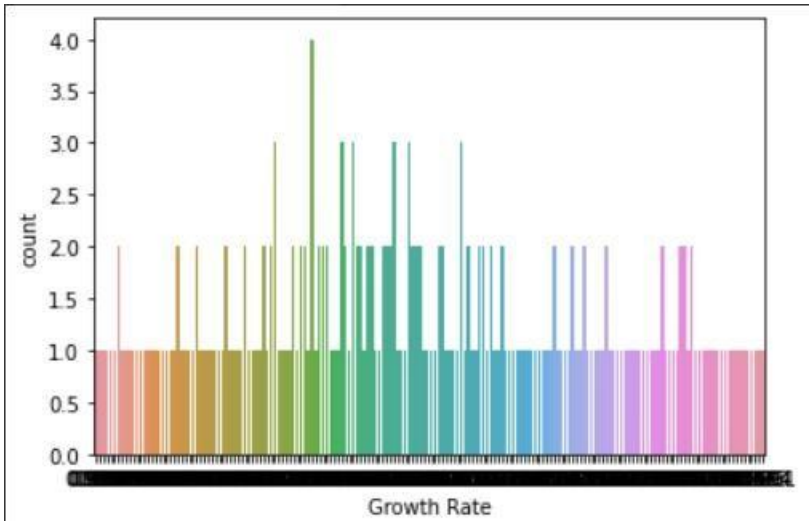
```
import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns
```

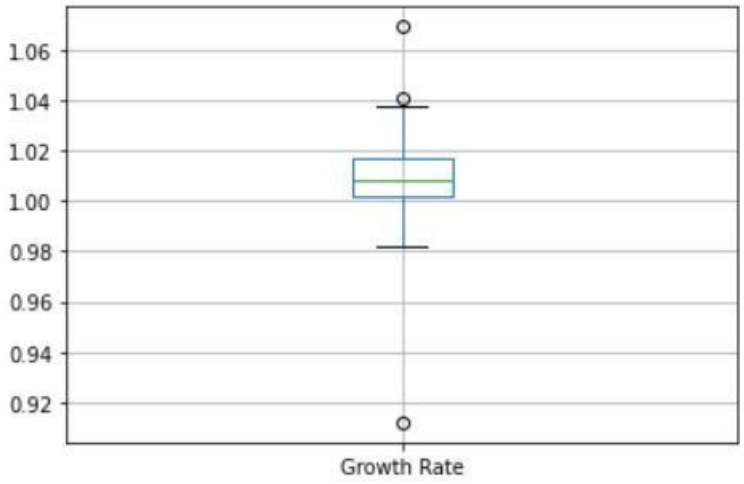
```
from google.colab import files

uploaded = files.upload()

import io

# reading csv file as pandas dataframe

data = pd.read_csv(io.BytesIO(uploaded['world_population.csv']))

data.head()

data.info()

data.duplicated().sum()

print(data.describe())

# Plot histograms

data.hist(bins=20, figsize=(15, 10))

plt.show()

# Plot scatter plots

sns.scatterplot(x='Growth Rate', y='World Population Percentage', data=data)

plt.show()

# Plot correlation matrix

sns.heatmap(data.corr(), annot=True)

plt.show()

#Plot the unique values

data["World Population Percentage"].unique()

sns.countplot(data['Growth Rate']).unique()

#Boxplot
```

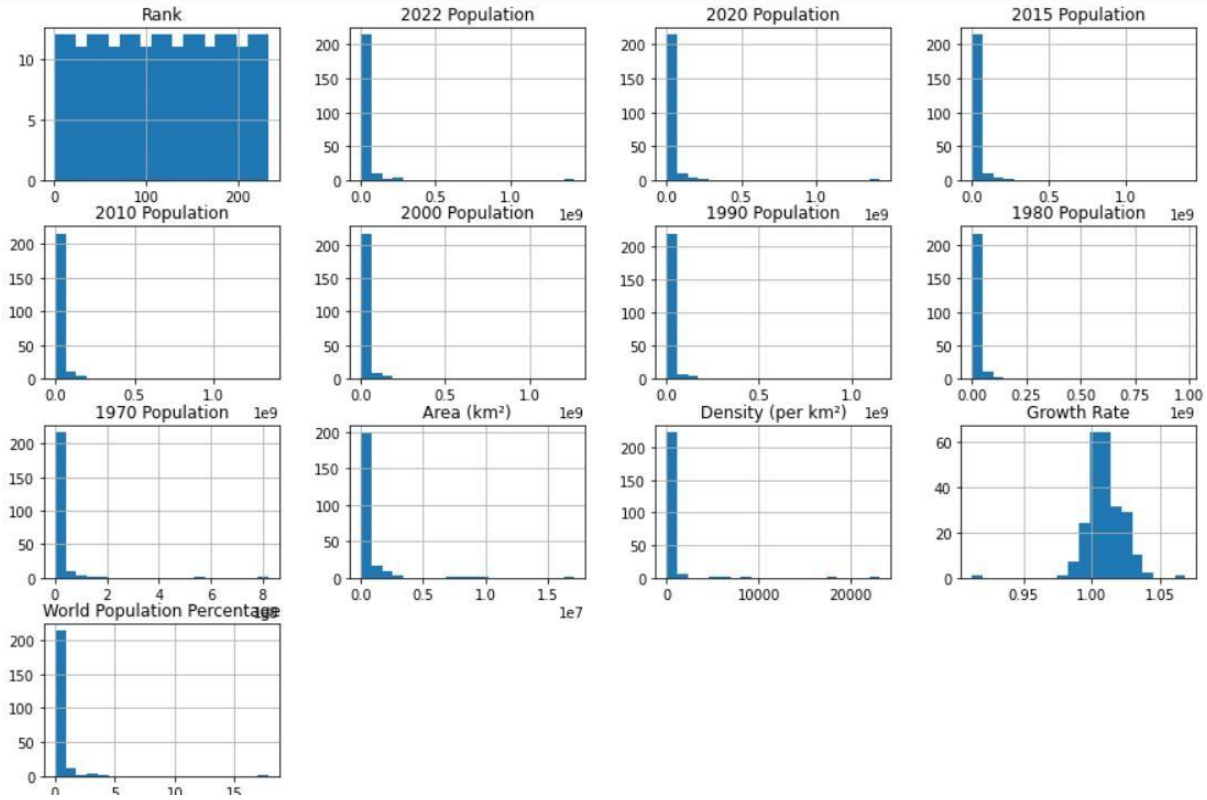
```
data[['Growth Rate']].boxplot()
```

OUTPUT:









RESULT:

Thus,to write a python program to implement EDA(Exploratory Data Analysis) is successfully created.

EX: 6

Text Pre-processing

DATE:

AIM:

To write a python program to implement Text Pre-processing.

ALGORITHM:

STEP 1: Start by importing the necessary libraries

STEP 2: Prepare the data.

STEP 3: Data Cleaning: In this step, you will clean and preprocess the data to ensure it's ready for analysis. This includes dealing with missing values, removing duplicates, and checking for outliers.

STEP 4: Exploratory Data Analysis: This is the core step of the analysis. You will use various statistical and visualization techniques to understand the data and draw insights from it.

STEP 5: You can use various communication tools like reports, dashboards, and visualizations to convey the insights and recommendations effectively.

Program Code:

```
import numpy as np

import pandas as pd

import re
```

```
import nltk
import spacy
import string
pd.options.mode.chained_assignment = None


full_df = pd.read_csv('/content/drive/MyDrive/sample.csv', encoding="latin-1")
full_df
df = full_df[["text"]]
df["text"] = df["text"].astype(str)
full_df.head()
import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords
", ".join(stopwords.words('english'))
PUNCT_TO_REMOVE = string.punctuation
def remove_punctuation(text):
    """custom function to remove the punctuation"""
    return text.translate(str.maketrans(", ", PUNCT_TO_REMOVE))


df["text_wo_punct"] = df["text"].apply(lambda text: remove_punctuation(text))
df.head()
STOPWORDS = set(stopwords.words('english'))
def remove_stopwords(text):
```

```
    """custom function to remove the stopwords"""
    return " ".join([word for word in str(text).split() if word not in STOPWORDS])


df["text_wo_stop"] = df["text_wo_punct"].apply(lambda text: remove_stopwords(text)
)
df.head()
from nltk.stem.porter import PorterStemmer


# Drop the two columns
df.drop(["text_wo_punct", "text_wo_stop"], axis=1, inplace=True)


stemmer = PorterStemmer()
def stem_words(text):
    return " ".join([stemmer.stem(word) for word in text.split()])


df["text_stemmed"] = df["text"].apply(lambda text: stem_words(text))
df.head()
from nltk.stem.snowball import SnowballStemmer
SnowballStemmer.languages
import nltk
nltk.download('wordnet')
nltk.download('omw-1.4')
from nltk.stem import WordNetLemmatizer
```

```
lemmatizer = WordNetLemmatizer()
def lemmatize_words(text):
    return " ".join([lemmatizer.lemmatize(word) for word in text.split()])

df["text_lemmatized"] = df["text"].apply(lambda text: lemmatize_words(text))
df.head()
from spellchecker import SpellChecker

spell = SpellChecker()
def correct_spellings(text):
    corrected_text = []
    misspelled_words = spell.unknown(text.split())
    for word in text.split():
        if word in misspelled_words:
            corrected_text.append(spell.correction(word))
        else:
            corrected_text.append(word)
    return " ".join(corrected_text)
correct_spellings(text)
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
CountVec = CountVectorizer(ngram_range=(1,1),stop_words='english')
#transform
Count_data = CountVec.fit_transform([text])
```

```
#create dataframe

cv_dataframe=pd.DataFrame(Count_data.toarray(),columns=CountVec.get_feature_n
ames_out())

print(cv_dataframe)
```

OUTPUT:

	text	text_wo_punct
0	@AppleSupport causing the reply to be disregar...	AppleSupport causing the reply to be disregard...
1	@105835 Your business means a lot to us. Pleas...	105835 Your business means a lot to us Please ...
2	@76328 I really hope you all change but I'm su...	76328 I really hope you all change but Im sure...
3	@105836 LiveChat is online at the moment - htt...	105836 LiveChat is online at the moment https...
4	@VirginTrains see attached error message. I've...	VirginTrains see attached error message Ive tr...

	text	text_wo_punct	text_wo_stop
0	@AppleSupport causing the reply to be disregar...	AppleSupport causing the reply to be disregard...	AppleSupport causing reply disregarded tapped ...
1	@105835 Your business means a lot to us. Pleas...	105835 Your business means a lot to us Please ...	105835 Your business means lot us Please DM na...
2	@76328 I really hope you all change but I'm su...	76328 I really hope you all change but Im sure...	76328 I really hope change Im sure wont Becaus...
3	@105836 LiveChat is online at the moment - htt...	105836 LiveChat is online at the moment https...	105836 LiveChat online moment httpstcoSY94VtU8...
4	@VirginTrains see attached error message. I've...	VirginTrains see attached error message Ive tr...	VirginTrains see attached error message Ive tr...

	text	text_stemmed
0	@AppleSupport causing the reply to be disregar...	@applesupport caus the repli to be disregard a...
1	@105835 Your business means a lot to us. Pleas...	@105835 your busi mean a lot to us. pleas dm y...
2	@76328 I really hope you all change but I'm su...	@76328 i realli hope you all chang but i'm sur...
3	@105836 LiveChat is online at the moment - htt...	@105836 livechat is onlin at the moment - http...
4	@VirginTrains see attached error message. I've...	@virgintrain see attach error message. i'v tri...

	text	text_stemmed	text_lemmatized
0	@AppleSupport causing the reply to be disregar...	@applesupport caus the repli to be disregard a...	@AppleSupport causing the reply to be disregar...
1	@105835 Your business means a lot to us. Pleas...	@105835 your busi mean a lot to us. pleas dm y...	@105835 Your business mean a lot to us. Please...
2	@76328 I really hope you all change but I'm su...	@76328 i realli hope you all chang but i'm sur...	@76328 I really hope you all change but I'm su...
3	@105836 LiveChat is online at the moment - htt...	@105836 livechat is onlin at the moment - http...	@105836 LiveChat is online at the moment - htt...
4	@VirginTrains see attached error message. I've...	@virgintrain see attach error message. i'v tri...	@VirginTrains see attached error message. I've...

	correctin	speling
0	1	1

RESULT:

Thus,to write a python program to implement Text Pre-processing is successfully created.

EX: 7

Image processing

DATE:

AIM:

To write a python program to implement a machine learning model to perform image processing for a given image.

ALGORITHM:

.

STEP 1: Start by importing the necessary libraries

STEP 2: Load the image.

STEP 3: Resize the image: Resizing the image is essential for reducing computational complexity and increasing the speed of your computer vision models. Split the ‘x’ and ‘y’ vectors into test and train data withtest split of 0.2.

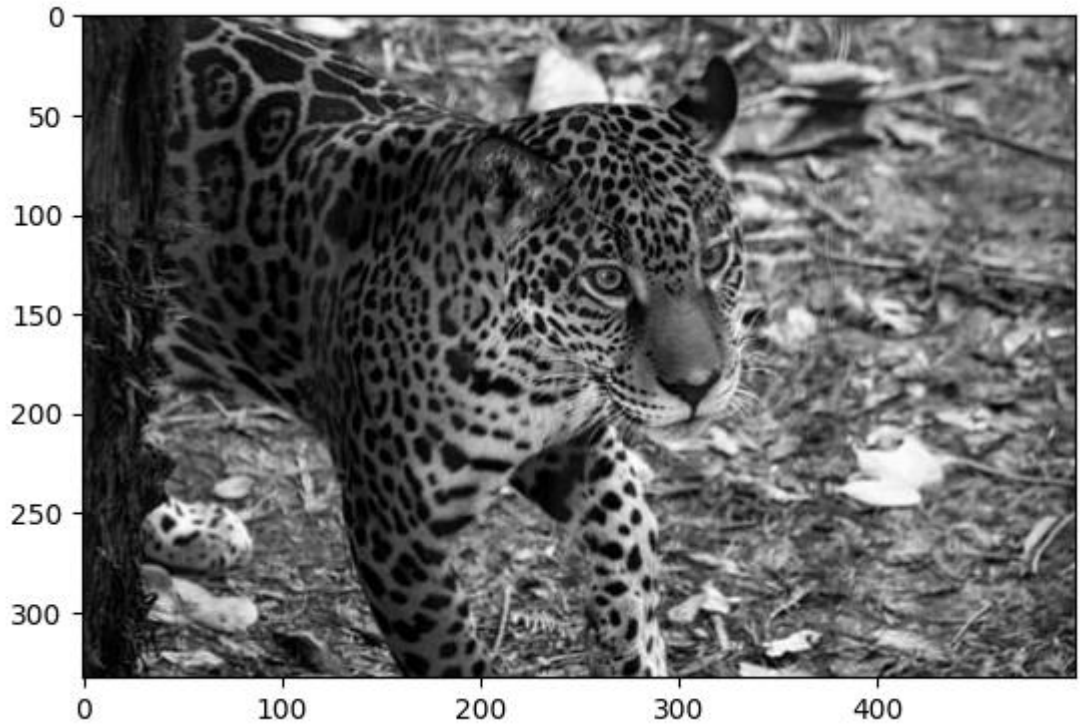
STEP 4: Normalize the image: Normalize the pixel values of the image to make them in the range of [0, 1]. Use the following code to normalize the image. Fit the training data in the model and train the model.

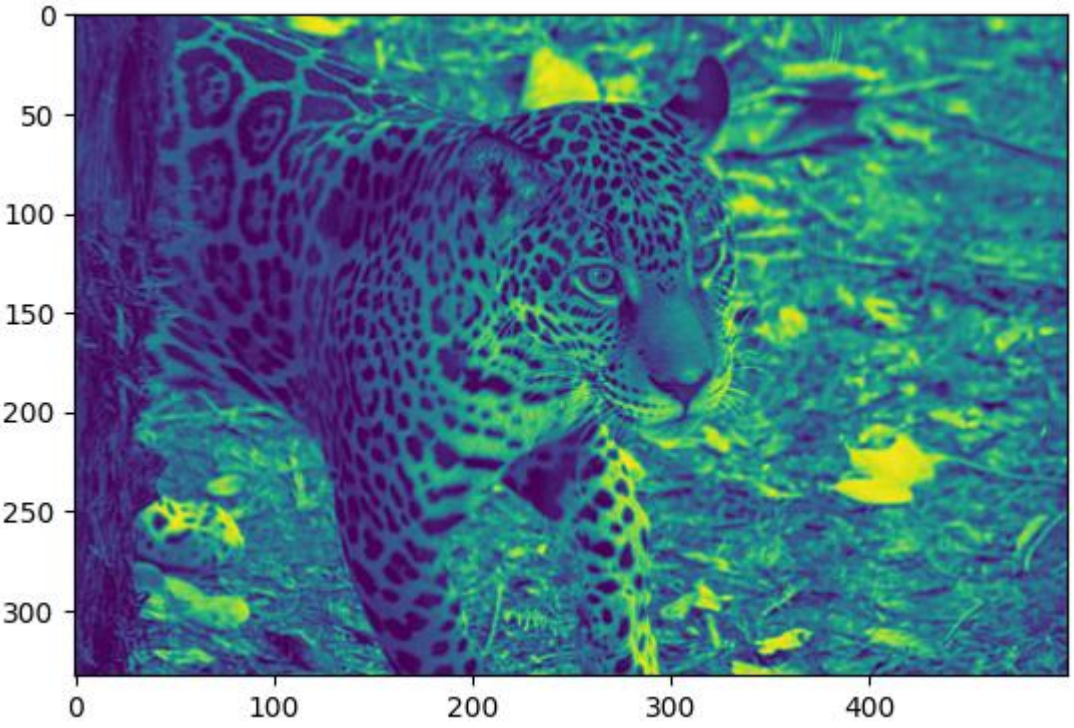
Program Code:

```
# importing libraries
import tensorflow
import keras
import os
import glob
import skimage
from skimage import io
import random
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
dataset_path = '/content/drive/MyDrive/Colab Notebooks/Animals'
class_names = ['Cheetah', 'Jaguar', 'Leopard', 'Lion','Tiger']
```

```
# apply glob module to retrieve files/pathnames

animal_path = os.path.join(dataset_path, class_names[1], '*')
animal_path = glob.glob(animal_path)
animal_path = list(animal_path)
image = io.imread(animal_path[4])
# plotting the original image
i, (im1) = plt.subplots(1)
i.set_figwidth(15)
im1.imshow(image)
import cv2
resz = cv2.resize(image,(10000, 10000), fx = 9, fy = 9)
bigger = cv2.resize(image, (1050, 1610))
plt.imshow(resz)
# plt.imshow(bigger)
gray_image = skimage.color.rgb2gray(image)
plt.imshow(gray_image, cmap = 'gray')
norm_image = (gray_image - np.min(gray_image)) / (np.max(gray_image) -
np.min(gray_image))
plt.imshow(norm_image)
from keras.preprocessing.image import ImageDataGenerator
datagen = ImageDataGenerator(horizontal_flip=True, vertical_flip=True)
Datagen
```

OUTPUT:





RESULT:

Thus, the program to implement a machine learning model to perform image processing for a given image is successfully executed.

EX: 8

SENTIMENTAL ANALYSIS

DATE:

AIM:

To write a python program to implement a machine learning model to perform sentimental analysis for a sample dataset.

ALGORITHM:

.

STEP 1: Start by importing the necessary libraries

STEP 2: Load the data: Load the textual data on which you want to perform sentiment analysis. You can load the data from a text file or from a database. Set Experience column as vector ‘x’ and Salary column as vector ‘y’.

STEP 3: Preprocess the data: Preprocess the textual data to remove any irrelevant information or noise. Split the ‘x_poly’ and ‘y’ vectors into test and train data with test split of 0.2.

STEP 4: Perform sentiment analysis using TextBlob: TextBlob is a Python library that provides a simple API for performing sentiment analysis. Fit the training data in the model and train the model.

Program Code:

```
from google.colab import drive
drive.mount("/content/drive/")
import pandas as pd
df = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/Reviews.csv')
df.head()
# Imports
import matplotlib.pyplot as plt
import seaborn as sns
color = sns.color_palette()
%matplotlib inline
import plotly.offline as py
```

```
py.init_notebook_mode(connected=True)
import plotly.graph_objs as go
import plotly.tools as tls
import plotly.express as px
# Product Scores
fig = px.histogram(df, x="Score")
fig.update_traces(marker_color="turquoise",marker_line_color='rgb(8,48,107)',
                  marker_line_width=1.5)
fig.update_layout(title_text='Product Score')
fig.show()

import nltk
from nltk.corpus import stopwords

import re
import nltk
nltk.download('stopwords')

# assign reviews with score > 3 as positive sentiment
# score < 3 negative sentiment
# remove score = 3
df = df[df['Score'] != 3]
df['sentiment'] = df['Score'].apply(lambda rating : +1 if rating > 3 else -1)

df.head()
df['Text'].head()
positive = df[df['sentiment'] == 1]
negative = df[df['sentiment'] == -1]
positive
positive['sentiment']
negative
def remove_punctuation(text):
```

```
    final = "".join(u for u in text if u not in ("?", ".", ";", ":",  "!",""))
    return final
df['Text'] = df['Text'].apply(remove_punctuation)
df = df.dropna(subset=['Summary'])
df['Summary'] = df['Summary'].apply(remove_punctuation)
dfNew = df[['Summary','sentiment']]
dfNew.head()
import numpy as np
# random split train and test data
index = df.index
df['random_number'] = np.random.randn(len(index))
train = df[df['random_number'] <= 0.8]
test = df[df['random_number'] > 0.8]
# count vectorizer:
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(token_pattern=r'\b\w+\b')
train_matrix = vectorizer.fit_transform(train['Summary'])
test_matrix = vectorizer.transform(test['Summary'])
# Logistic Regression
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression()
X_train = train_matrix
X_test = test_matrix
y_train = train['sentiment']
y_test = test['sentiment']
lr.fit(X_train,y_train)
predictions = lr.predict(X_test)
from sklearn.metrics import confusion_matrix,classification_report
new = np.asarray(y_test)
confusion_matrix(predictions,y_test)
print(classification_report(predictions,y_test))
```

OUTPUT:

	Summary	sentiment
0	Good Quality Dog Food	1
1	Not as Advertised	-1
2	Delight says it all	1
3	Cough Medicine	-1
4	Great taffy	1

```
array([[11639,  2386],
       [ 5899, 91874]])
```

	precision	recall	f1-score	support
-1	0.66	0.83	0.74	14025
1	0.97	0.94	0.96	97773
accuracy			0.93	111798
macro avg	0.82	0.88	0.85	111798
weighted avg	0.94	0.93	0.93	111798

RESULT:

Thus, the program to implement a machine learning model to performsentimental analysis for a sample dataset is successfully executed.

EX: 9

NEURAL NETWORKS

DATE:

AIM:

To write a python program to implement machine learning model to implement Neural Networks for a sample dataset.

ALGORITHM:

.

STEP 1: Start by importing the necessary libraries

STEP 2: Load the data: Load the textual data on which you want to perform sentiment analysis. You can load the data from a text file or from a database. Set Experience column as vector ‘x’ and Salary column as vector ‘y’.

STEP 3: Preprocess the data: Preprocess the textual data to remove any irrelevant information or noise. Split the ‘x_poly’ and ‘y’ vectors into test and train data with test split of 0.2.

STEP 4: Split the data: Split the data into training and testing sets. This will help you evaluate the performance of the neural network on unseen data.

STEP 5: Evaluate the neural network: Evaluate the performance of the neural network using the testing data.

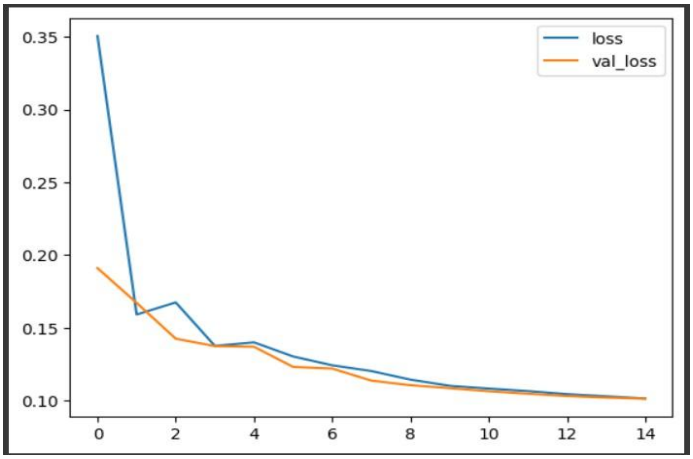
Program Code:

```
from google.colab import drive
drive.mount("/content/drive/")
import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout
from keras.optimizers import RMSprop

# Load the MNIST dataset
```

```
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# Preprocess the data
x_train = x_train.reshape(60000, 784).astype('float32') / 255
x_test = x_test.reshape(10000, 784).astype('float32') / 255
y_train = keras.utils.to_categorical(y_train, 10)
y_test = keras.utils.to_categorical(y_test, 10)

# Define the neural network model
model = Sequential()
model.add(Dense(512, activation='relu', input_shape=(784,)))
model.add(Dropout(0.2))
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(10, activation='softmax'))

# Compile the model
model.compile(loss='categorical_crossentropy',
              optimizer=RMSprop(),
              metrics=['accuracy'])

# Train the model
history = model.fit(x_train, y_train,
                    batch_size=128,
                    epochs=20,
                    verbose=1,
                    validation_data=(x_test, y_test))

# Evaluate the model
score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

OUTPUT:

```
Epoch 8/20
469/469 [==============================] - 8s 17ms/step - loss: 0.0355 - accuracy: 0.9896 - val_loss: 0.0747 - val_accuracy: 0.9833
Epoch 9/20
469/469 [==============================] - 9s 19ms/step - loss: 0.0320 - accuracy: 0.9904 - val_loss: 0.0877 - val_accuracy: 0.9817
Epoch 10/20
469/469 [==============================] - 9s 20ms/step - loss: 0.0292 - accuracy: 0.9913 - val_loss: 0.0923 - val_accuracy: 0.9829
Epoch 11/20
469/469 [==============================] - 8s 17ms/step - loss: 0.0267 - accuracy: 0.9920 - val_loss: 0.1074 - val_accuracy: 0.9802
Epoch 12/20
469/469 [==============================] - 9s 20ms/step - loss: 0.0250 - accuracy: 0.9927 - val_loss: 0.0997 - val_accuracy: 0.9820
Epoch 13/20
469/469 [==============================] - 9s 19ms/step - loss: 0.0247 - accuracy: 0.9931 - val_loss: 0.0918 - val_accuracy: 0.9835
Epoch 14/20
469/469 [==============================] - 11s 24ms/step - loss: 0.0215 - accuracy: 0.9938 - val_loss: 0.1053 - val_accuracy: 0.9828
Epoch 15/20
469/469 [==============================] - 8s 17ms/step - loss: 0.0217 - accuracy: 0.9940 - val_loss: 0.1205 - val_accuracy: 0.9824
Epoch 16/20
469/469 [==============================] - 9s 20ms/step - loss: 0.0210 - accuracy: 0.9944 - val_loss: 0.1085 - val_accuracy: 0.9837
Epoch 17/20
469/469 [==============================] - 9s 20ms/step - loss: 0.0209 - accuracy: 0.9947 - val_loss: 0.1142 - val_accuracy: 0.9830
Epoch 18/20
469/469 [==============================] - 8s 17ms/step - loss: 0.0206 - accuracy: 0.9948 - val_loss: 0.1266 - val_accuracy: 0.9799
Epoch 19/20
469/469 [==============================] - 9s 19ms/step - loss: 0.0183 - accuracy: 0.9948 - val_loss: 0.1292 - val_accuracy: 0.9843
Epoch 20/20
469/469 [==============================] - 10s 20ms/step - loss: 0.0177 - accuracy: 0.9954 - val_loss: 0.1208 - val_accuracy: 0.9842
Test loss: 0.12083691358566284
Test accuracy: 0.9842000007629395
```



RESULT:

Thus, the program to implement a machine learning model to performsentimental analysis for a sample dataset is successfully executed.

EX: 10

NAIVE BAYES

DATE:

AIM:

To write a python program to implement Naive Bayes classification for a sample dataset.

ALGORITHM:

.

STEP 1: Start by importing the necessary libraries

STEP 2: Load your dataset into a Pandas Data Frame.

STEP 3: Split the ‘x’ and ‘y’ vectors into test and train data with test split of 0.25.

STEP 4: Create an instance of the Gaussian Naive Bayes classifier.

STEP 5: Fit the training data in the model and train the model.

STEP 6: Make predictions on the testing data.

Program Code:

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn import preprocessing
df = pd.read_csv('/content/drive/MyDrive/FOML/drug.csv')
df
df['Drug'].value_counts()
label_encoder = preprocessing.LabelEncoder()
df['Sex']= label_encoder.fit_transform(df['Sex'])
df['BP']= label_encoder.fit_transform(df['BP'])
```

```
df['Cholesterol']= label_encoder.fit_transform(df['Cholesterol'])
x = df.drop(['Drug'], axis=1)
y = df['Drug']
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.25, random_state = 0)
from sklearn.naive_bayes import GaussianNB
classifier = GaussianNB()
classifier.fit(x_train, y_train)
y_pred = classifier.predict(x_test)
print("Accuracy = ", classifier.score(x_test, y_test)*100)
print("Classification report:\n", classification_report(y_test, y_pred))
print("Confusion matrix:\n", confusion_matrix(y_test, y_pred))
```

OUTPUT:

```
Accuracy =  86.0
```

```
Classification report:
              precision    recall  f1-score   support

       DrugY       0.95      0.76      0.84        25
       drugA       0.71      1.00      0.83         5
       drugB       0.50      1.00      0.67         1
       drugC       0.60      1.00      0.75         3
       drugX       0.94      0.94      0.94        16

    accuracy                           0.86        50
   macro avg       0.74      0.94      0.81        50
weighted avg       0.89      0.86      0.86        50
```

```
Confusion matrix:
 [[19  2  1  2  1]
 [ 0  5  0  0  0]
 [ 0  0  1  0  0]
 [ 0  0  0  3  0]
 [ 1  0  0  0 15]]
```

RESULT:

Thus, the program to implement Naive Bayes classification for a sampledataset is successfully executed.

EX: 11

FACIAL RECOGNITION

DATE:

AIM:

To write a python program to implement Facial recognition.

ALGORITHM:

.

STEP 1: Start by importing the necessary libraries

STEP 2: Collect and prepare a dataset of images that contain faces.

STEP 3: Preprocess the images to ensure that they are of consistent size, resolution, and orientation.

STEP 4: Use a face detection algorithm to locate the faces in the images.

STEP 5: Test the trained model on a separate dataset of images and evaluate its performance using metrics such as accuracy, precision, and recall.

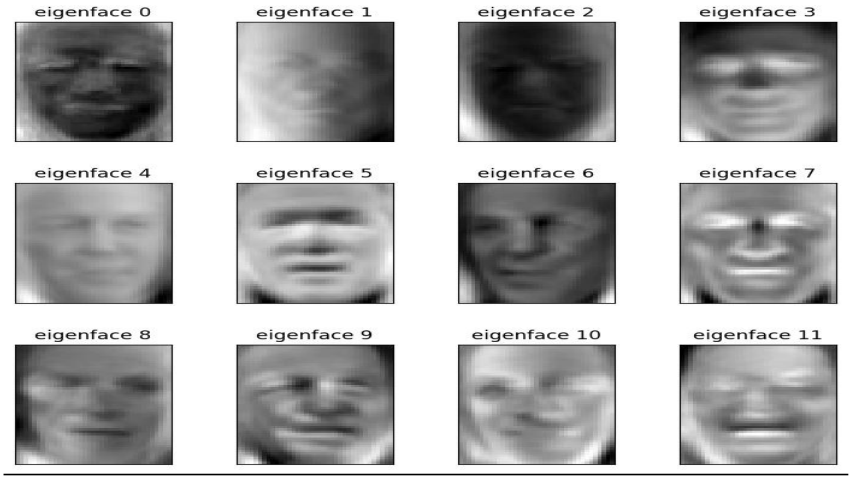
Program Code:

```
import pylab as pl import numpy as np
from matplotlib import pyplot as plt
from sklearn.model_selection import train_test_split from sklearn.datasets import
fetch_lfw_people
from sklearn.model_selection import GridSearchCV from sklearn.metrics import
classification_report from sklearn.metrics import confusion_matrix
from sklearn.decomposition import PCA as RandomizedPCA from sklearn.svm import SVC
lfw_people = fetch_lfw_people(min_faces_per_person=70, resize=0.4) n_samples, h, w =
lfw_people.images.shape
np.random.seed(42)


y = lfw_people.target
target_names = lfw_people.target_names n_classes = target_names.shape[0] print("Total
dataset size:") print("n_samples: %d", n_samples) print("n_features: %d", n_features)
print("n_classes: %d", n_classes)
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)
X_train_pca = pca.transform(X_train) X_test_pca = pca.transform(X_test) print("Fitting the classifier to the training set") param_grid = {
'C': [1e3, 5e3, 1e4, 5e4, 1e5],
'gamma': [0.0001, 0.0005, 0.001, 0.005, 0.01, 0.1],
}
clf = GridSearchCV(SVC(kernel='rbf', class_weight='balanced'), param_grid) clf = clf.fit(X_train_pca, y_train)
print("Best estimator found by grid search:") print(clf.best_estimator_)
print("Predicting the people names on the testing set") y_pred = clf.predict(X_test_pca)
print(confusion_matrix(y_test, y_pred, labels=range(n_classes)))
```

OUTPUT:

	precision	recall	f1-score	support
Ariel Sharon	0.77	0.77	0.77	13
Colin Powell	0.78	0.90	0.84	60
Donald Rumsfeld	0.67	0.59	0.63	27
George W Bush	0.87	0.90	0.89	146
Gerhard Schroeder	0.68	0.68	0.68	25
Hugo Chavez	0.80	0.53	0.64	15
Tony Blair	0.83	0.67	0.74	36
accuracy			0.81	322
macro avg	0.77	0.72	0.74	322
weighted avg	0.81	0.81	0.81	322



RESULT:

Thus, the program to implement Facial recognition is successfullyexecuted.

EX: 12 # SINGLE LAYER PERCEPTRON

DATE:

AIM:

To write a python program to implement single layer perceptron.

ALGORITHM:

.

STEP 1: Start by importing the necessary libraries

STEP 2: Load your dataset into a Pandas DataFrame.

STEP 3: Preprocess the data by scaling the features to have zero mean and unit variance.

STEP 4: Split the data into training and testing sets.

STEP 5: Train the model on the training data using the training loop.

STEP 6: Make predictions on the testing data.

Program Code:

```
from google.colab import drive
drive.mount("/content/drive/")
import sklearn.model_selection
from sklearn.model_selection import train_test_split
import numpy as np
import pandas as pd
from sklearn.preprocessing import StandardScaler


df = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/adult.csv', header=None,
skipinitialspace=True)


df.columns = ['age', 'workclass', 'fnlwgt', 'education', 'education-num', 'marital-status',
        'occupation', 'relationship', 'race', 'sex', 'capital-gain', 'capital-loss',
```

```
            'hours-per-week', 'native-country', 'income']

df = pd.get_dummies(df, columns=['workclass', 'education', 'marital-status', 'occupation',
                                 'relationship', 'race', 'sex', 'native-country'])
df['income'] = df['income'].apply(lambda x: 1 if x.strip() == '>50K' else 0)

X = df.drop('income', axis=1).values
y = df['income'].values
scaler = StandardScaler()
X = scaler.fit_transform(X)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

def step_function(x):
    return np.where(x >= 0, 1, 0)

# Define the perceptron function
def perceptron(X, y, learning_rate=0.1, epochs=100):
    # Initialize the weights and bias to zero
    w = np.zeros(X.shape[1])
    b = 0

    # Loop over the specified number of epochs
    for epoch in range(epochs):
        # Loop over each data point in the training set
        for i in range(X.shape[0]):
            # Calculate the activation potential for the current data point
            z = np.dot(w, X[i]) + b
            # Apply the activation function to get the predicted output
            y_pred = step_function(z)
            # Update the weights and bias based on the prediction error
            w += learning_rate * (y[i] - y_pred) * X[i]
            b += learning_rate * (y[i] - y_pred)
```

```
    # Return the final weights and bias
    return w, b

# Train the perceptron on the training data
w, b = perceptron(X_train, y_train, learning_rate=0.1, epochs=100)

# Make predictions on the test data
predictions = step_function(np.dot(X_test, w) + b)

# Calculate the accuracy of the classifier
accuracy = np.mean(predictions == y_test)

# Print the accuracy
print('Accuracy: %.2f' % accuracy)
def activation_func(value):    #Tangent Hypotenuse
    #return (1/(1+np.exp(-value)))
    return ((np.exp(value)-np.exp(-value))/(np.exp(value)+np.exp(-value)))
def perceptron_train(in_data,labels,alpha):
    X=np.array(in_data)
    y=np.array(labels)
    weights=np.random.random(X.shape[1])
    original=weights
    bias=np.random.random_sample()
    for key in range(X.shape[0]):
        a=activation_func(np.matmul(np.transpose(weights),X[key]))
        yn=0
        if a>=0.7:
            yn=1
        elif a<=(-0.7):
            yn=-1
        weights=weights+alpha*(yn-y[key])*X[key]
```

```
            print('Iteration '+str(key)+': '+str(weights))
        print('Difference: '+str(weights-original))
        return weights
def perceptron_test(in_data,label_shape,weights):
    X=np.array(in_data)
    y=np.zeros(label_shape)
    for key in range(X.shape[1]):
        a=activation_func((weights*X[key]).sum())
        y[key]=0
        if a>=0.7:
            y[key]=1
        elif a<=(-0.7):
            y[key]=-1
    return y
def score(result,labels):
    difference=result-np.array(labels)
    correct_ctr=0
    for elem in range(difference.shape[0]):
        if difference[elem]==0:
            correct_ctr+=1
    score=correct_ctr*100/difference.size
    print('Score='+str(score))
# Dividing DataFrame "data" into "d_train" (60%) and "d_test" (40%)
divider = np.random.rand(len(df)) < 0.70
d_train=df[divider]
d_test=df[~divider]
# Dividing d_train into data and labels/targets
d_train_y=d_train['income']
d_train_X=d_train.drop(['income'],axis=1)

# Dividing d_train into data and labels/targets
d_test_y=d_test['income']
```

```
d_test_X=d_test.drop(['income'],axis=1)
# Learning rate
alpha = 0.01
# Train
weights = perceptron_train(d_train_X, d_train_y, alpha)
# Test
result_test=perceptron_test(d_test_X,d_test_y.shape,weights)
# Calculate score
score(result_test,d_test_y)
```

OUTPUT:

```
Accuracy: 0.82
```

```
Score=75.89120131877189
<ipython-input-22-22fcf04698ab>:3: RuntimeWarning: overflow encountered in exp
  return ((np.exp(value)-np.exp(-value))/(np.exp(value)+np.exp(-value)))
<ipython-input-22-22fcf04698ab>:3: RuntimeWarning: invalid value encountered in double_scalars
  return ((np.exp(value)-np.exp(-value))/(np.exp(value)+np.exp(-value)))
```

RESULT:

Thus, the program to implement single layer perceptron is successfullyexecuted.