

SRM VALLIAMMAI ENGINEERING COLLEGE

(An Autonomous Institution)

SRM Nagar, Kattankulathur-603203

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

ACADEMIC YEAR: 2023-2024

EVEN SEMESTER

LABMANUAL

(REGULATION - 2019)

1922405-DATASCIENCE LABO

FOURTH SEMSTER

B. Tech – Artificial Intelligence and Data Science

Prepared By

V. ABARNA, A.P (O.G) / AI&DS

INDEX

| E.NO | EXPERIMENT NAME | Pg. No. |
|------|---|---------|
| A | PEO, PO, PSO | 3-5 |
| В | Syllabus | 6 |
| С | Introduction/ Description of major Software & Hardware involved in lab | 7 |
| D | CO, CO-PO Matrix, CO-PSO Matrix | 7 |
| Е | Mode of Assessment | 8 |
| 1 | Write a program in Python to predict the class of the flower based on available attributes. | 9-13 |
| 2 | Write a program in Python to predict if a loan will get approved or not. | 14-21 |
| 3 | Write a program in Python to predict the traffic on a new mode of transport. | 22-28 |
| 4 | Write a program in Python to predict the class of user. | 29-35 |
| 5 | Write a program in Python to identify the tweets which are hate tweets and which are not. | 36-41 |
| 6 | Write a program in Python to predict the age of the actors. | 42-48 |
| 7 | Introduction to Python Libraries-Numpy, Pandas, Matplotlib, Scikit | 49-51 |
| 8 | Perform Data exploration and preprocessing in Python | 52-59 |
| 9 | Implement regularized Linear regression | 60-65 |
| 10 | Mini project to predict the time taken to solve a problem given the current status of the user. | 66-72 |
| 11 | Topic Beyond Syllabus Implementation of Multivariate Regression | 73-77 |

PROGRAMME EDUCATIONAL OBJECTIVES (PEOs)

- 1. To afford the necessary background in the field of Information Technology to deal with engineering problems to excel as engineering professionals in industries.
- 2. To improve the qualities like creativity, leadership, teamwork and skill thus contributing towards the growth and development of society.
- 3. To develop ability among students towards innovation and entrepreneurship that caters to the needs of Industry and society.
- 4. To inculcate and attitude for life-long learning process through the use of information technology sources.
- 5. To prepare then to be innovative and ethical leaders, both in their chosen profession and in other activities.

PROGRAMME OUTCOMES (POs)

After going through the four years of study, Information Technology Graduates will exhibit ability to:

| PO# | Graduate Attribute | Programme Outcome |
|-----|--|--|
| 1 | Engineering knowledge | Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization for the solution of complex engineering problems. |
| 2 | Problem analysis | Identify, formulate, research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences. |
| 3 | Design/development of solutions | Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for public health and safety, and cultural, societal, and environmental considerations. |
| 4 | Conduct investigations of complex problems | Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and |

| | | synthesis of the information to provide valid conclusions | | | | | |
|----|--------------------------|---|--|--|--|--|--|
| | | Create, select, and apply appropriate techniques, resources, and | | | | | |
| | Modern tool usego | modern engineering and IT tools, including prediction an | | | | | |
| 5 | Modern tool usage | modeling to complex engineering activities, with an | | | | | |
| | | understanding of the limitations. | | | | | |
| | | Apply reasoning informed by the contextual knowledge to assess | | | | | |
| | The engineer and society | societal, health, safety, legal, and cultural issues and the | | | | | |
| 6 | The engineer and society | consequent responsibilities relevant to the professional | | | | | |
| | | engineering practice | | | | | |
| | Environment and | Understand the impact of the professional engineering solutions | | | | | |
| 7 | sustainability | in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development. | | | | | |
| | sustamaomity | knowledge of, and need for sustainable development. | | | | | |
| 0 | Ethics | Apply ethical principles and commit to professional ethics and | | | | | |
| 8 | Lunes | responsibilities and norms of the engineering practice | | | | | |
| | Individual and team work | Function effectively as an individual, and as a member or | | | | | |
| 9 | marviduai and team work | leader in diverse teams, and in multidisciplinary settings | | | | | |
| | | Communicate effectively on complex engineering activities with | | | | | |
| | | the engineering community and with the society at large, such | | | | | |
| 10 | Communication | as, being able to comprehend and write effective reports and | | | | | |
| | | design documentation, make effective presentations, and | | | | | |
| | | give and receive clear instructions | | | | | |
| | | Demonstrate knowledge and understanding of the engineering | | | | | |
| 11 | Project management and | and management principles and apply these to one's own work, | | | | | |
| 11 | finance | as a member and leader in a team, to manage projects and in | | | | | |
| | | multidisciplinary environments | | | | | |
| | | Recognize the need for, and have the preparation and ability to | | | | | |
| 12 | Life-long learning | engage in independent and life-long learning in the broadest | | | | | |
| | | context of technological change | | | | | |

PROGRAMME SPECIFIC OUTCOMES (PSOs)

After the completion of Bachelor of Technology in Artificial Intelligence and Data Science programme the student will have following Program specific outcomes

- 1. Design and develop secured database applications with data analytical approaches of data preprocessing, optimization, visualization techniques and maintenance using state of the art methodologies based on ethical values.
- 2. Design and develop intelligent systems using computational principles, methods and systems for extracting knowledge from data to solve real time problems using advanced technologies and tools.
- 3. Design, plan and setting up the network that is helpful for contemporary business environments using latest software and hardware.
- 4. Planning and defining test activities by preparing test cases that can predict and correct errors ensuring a socially transformed product catering all technological needs.



1922405

OBJECTIVES:

This course will enable students to

- Develop a basic understanding of import data sets for various analytical purposes.
- Utilize the various packages available visualization, reporting, data manipulation, and statistical analysis.
- Create interactive business applications that allow for data querying and data exploration.
- Collect, explore, clean, munge and manipulate data.
- Build data science applications using Python based toolkits.

LIST OF EXPERIMENTS:

- 1. Write a program in Python to predict the class of the flower based on available attributes.
- 2. Write a program in Python to predict if a loan will get approved or not.
- 3. Write a program in Python to predict the traffic on a new mode of transport.
- 4. Write a program in Python to predict the class of user.
- 5. Write a program in Python to identify the tweets which are hate tweets and which are not.
- 6. Write a program in Python to predict the age of the actors.
- 7. Introduction to Python Libraries-Numpy, Pandas, Matplotlib, Scikit
- 8. Perform Data exploration and preprocessing in Python
- 9. Implement regularized Linear regression
- 10. Mini project to predict the time taken to solve a problem given the current status of the user.

Total: 60 Periods

LIST OF EQUIPMENTS FOR A BATCH OF 30 STUDENTS

SOFTWARE:

Standalone desktops with Python 3 interpreter for Windows / Linux 30 Nos. (or) Server with Python 3 interpreter for Windows/Linux supporting 30 terminals or more.

HARDWARE:

Standalone Desktops: 30 Nos.

COURSE OUTCOMES

| 1922405.1 | Import data sets for various analytical purposes. |
|-----------|--|
| 1922405.2 | Suggest Develop an interactive business application that allow for data querying and data exploration. |
| 1922405.3 | Collect, explore, clean, munge and manipulate data. |
| 1922405.4 | Develop data science applications using Python based toolkits. |
| 1922405.5 | Implement models such as k-nearest Neighbours, Naive Bayes, linear and logistic regression, decision trees, neural networks and clustering |

CO-PO-PSO MATRIX

| CO | PO | | | | | | | | | | PSO | | | | | |
|----|----|---|---|---|---|---|---|---|---|----|-----|----|---|---|---|---|
| СО | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 1 | 2 | 3 | 4 |
| 1 | - | 2 | 3 | - | 3 | - | - | - | - | - | 1 | - | 2 | - | - | - |
| 2 | - | 2 | 3 | - | 3 | - | - | - | - | - | 1 | 2 | 2 | - | - | - |
| 3 | - | 2 | 3 | - | 3 | - | - | - | - | - | 1 | - | 3 | - | - | - |
| 4 | 2 | 2 | 3 | - | 3 | - | 1 | ı | ı | _ | 2 | 1 | 3 | 1 | 2 | 2 |
| 5 | 2 | 2 | 3 | - | 3 | - | 1 | 1 | 1 | _ | 2 | 1 | 3 | 1 | 1 | 2 |

EVALUATION PROCEDURE FOR EACH EXPERIMENT

| S. No | Description | Mark | | | | |
|-------|--------------------------|------|--|--|--|--|
| 1. | Aim & Procedure | 20 | | | | |
| 2. | Observation | 30 | | | | |
| 3. | Conduction and Execution | 30 | | | | |
| 4. | Output & Result | 10 | | | | |
| 5. | Viva | 10 | | | | |
| | Total | | | | | |

INTERNAL ASSESSMENT FOR LABORATORY

| S. No | Description | Mark | | | | |
|-------|--------------------------------------|------|--|--|--|--|
| 1. | Conduction & Execution of Experiment | 25 | | | | |
| 2. | Record | 10 | | | | |
| 3. | Model Test | 15 | | | | |
| | Total | | | | | |

Ex. No: 1 PREDICT THE CLASS OF THE FLOWER BASED ON AVAILABLE ATTRIBUTES

AIM

To write a Python program to predict the class of the flower based on available attributes using K-Nearest Neighbor.

PROCEDURE

- 1. Importing the necessary libraries, including 'pandas' for data manipulation, and 'numpy' for numerical operations.
- 2. Load the dataset containing flower attributes. For Example, iris dataset.
- 3. Explore the dataset to understand its structure and features. Preprocess the data if needed, such as handling missing values or encoding categorical variables.
- 4. Split the dataset into features (x) and target variable (y).
- 5. Further split the dataset into training and testing sets using 'train_test_split' from 'sklearn.model selection'.
- 6. Initialize a K-Nearest Neighbors model with a specified number of neighbors ('n_neighbors') from 'sklearn. Neighbors'.
- 7. Train the K-Nearest Neighbor model using the training data with the 'fit' method.
- 8. Use the trained KNN model to make predictions on the testing data with the 'predict' method.
- 9. Evaluate the performance of the model using **metrics** such as accuracy, precision, recall, F1_Score, or others based on the requirements.
- 10. Finally, predict the flower class for new data using the trained model.

PROGRAM

Import Libraries

import pandas as pd

import numpy as np

import warnings

warnings. Simplefilter ("ignore")

Load the Dataset

df = pd. read_csv("C:\\Users\\Documents\\iris.csv")
df. head ()

| | sepal_length | sepal_width | petal_length | petal_width | 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 |

df.info()

Output

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
Column Non-Null Count Dtype

| # | Column | Non-Null Count | Dtype |
|------------|--------------|----------------|---------|
| | | | |
| 0 | sepal_length | 150 non-null | float64 |
| 1 | sepal_width | 150 non-null | float64 |
| 2 | petal_length | 150 non-null | float64 |
| 3 | petal_width | 150 non-null | float64 |
| 4 | species | 150 non-null | object |
| The second | | -L-!(4) | |

dtypes: float64(4), object(1)

memory usage: 6.0+ KB

df. describe ()

| | sepal_length | sepal_width | petal_length | petal_width |
|-------------|--------------|-------------|--------------|-------------|
| count | 150.000000 | 150.000000 | 150.000000 | 150.000000 |
| mean | 5.843333 | 3.054000 | 3.758667 | 1.198667 |
| std | 0.828066 | 0.433594 | 1.764420 | 0.763161 |
| min | 4.300000 | 2.000000 | 1.000000 | 0.100000 |
| 25% | 5.100000 | 2.800000 | 1.600000 | 0.300000 |
| 50% | 5.800000 | 3.000000 | 4.350000 | 1.300000 |
| 75 % | 6.400000 | 3.300000 | 5.100000 | 1.800000 |
| max | 7.900000 | 4.400000 | 6.900000 | 2.500000 |

Explore and preprocess the data

df. isnull (). sum ()

Output

sepal_length 0
sepal_width 0
petal_length 0
petal_width 0
petal_width 0
species 0
dtype: int64

Split the data into features (x)

x = df. iloc [:, :-1]

X

| (| Output | | | SRM |
|-----|--------|-------------|--------------|-------------|
| | _ | sepal_width | petal_length | petal_width |
| 0 | 5.1 | 3.5 | 1.4 | 0.2 |
| 1 | 4.9 | 3.0 | 1.4 | 0.2 |
| 2 | 4.7 | 3.2 | 1.3 | 0.2 |
| 3 | 4.6 | 3.1 | 1.5 | 0.2 |
| 4 | 5.0 | 3.6 | 1.4 | 0.2 |
| | | | | |
| 145 | 6.7 | 3.0 | 5.2 | 2.3 |
| 146 | 6.3 | 2.5 | 5.0 | 1.9 |
| 147 | 6.5 | 3.0 | 5.2 | 2.0 |
| 148 | 6.2 | 3.4 | 5.4 | 2.3 |
| 149 | 5.9 | 3.0 | 5.1 | 1.8 |

150 rows × 4 columns

Split the data into target variable (y)

```
y = df. iloc [:, -1]
```

Output

```
0
          setosa
1
           setosa
2
           setosa
3
          setosa
4
          setosa
       virginica
145
146
       virginica
       virginica
147
       virginica
148
       virginica
149
Name: species, Length: 150, dtype: object
```

Further split the dataset into training and testing sets

from sklearn. model_selection **import** train_test_split x_train, x_test, y_train, y_test=train_test_split (x, y, random_state=0)

Intialize the KNN model

from sklearn. neighbors **import** KNeighborsClassifier model = KNeighborsClassifier(n_neighbors=3)

Train the KNN model

model.fit (x_train, y_train)

Output

KNeighborsClassifier(n_neighbors=3)

Make predictions

```
y_pred=model. predict(x_test)
y_pred
```

Evaluate the model

from sklearn. metrics **import** accuracy_score, confusion_matrix confusion_matrix (y_test, y_pred)

Output

accuracy=accuracy_score (y_test, y_pred) *100 print ("Accuracy of the model is {:.2f}". format(accuracy))

Output

Accuracy of the model is 97.37

from sklearn. metrics import classification_report
class_report = classification_report (y_test, y_pred)
print (f"\nClassification Report:\n{class_report}")

Output

Classification Report: precision recall f1-score support 1.00 1.00 1.00 setosa 13 versicolor 1.00 0.94 0.97 16 virginica 0.90 1.00 0.95 9 accuracy 0.97 38 0.97 0.98 0.97 38 macro avg weighted avg 0.98 0.97 0.97 38

Predict for new data

Output

array(['setosa'], dtype=object)

RESULT

Thus, the python program to predict the class of the flower based on available attributes was completed successfully using K-Nearest Neighbor model.

Ex. No: 2 PREDICT IF A LOAN WILL GET APPROVED OR NOT

AIM

To write a Python program to predict if a loan will get approved or not using Logistic Regression.

PROCEDURE

- 1. Importing the necessary libraries, including 'pandas' for data manipulation, and 'numpy' for numerical operations and warnings.
- 2. Load the dataset containing loan details. For Example, loan dataset.
- 3. Explore the dataset to understand its structure and features. Preprocess the data if needed, such as handling missing values.
- 4. Use 'LabelEncoder' to encode categorical columns like 'Gender', 'Married', 'Education', 'Self employed', etc.
- 5. Use 'SimpleImputer' to replace null values in numeric columns with the most frequent values.
- 6. Split the dataset into features (x) and target variable (y).
- 7. Replace '3+' with 3 and convert the column to float using 'df['Dependents'] = df['Dependents']. replace ('3+', 3). astype(float).
- 8. Further split the dataset into training and testing sets using 'train_test_split' from 'sklearn. model selection'.
- 9. Standardize the features using 'StandardScaler' from 'sklearn. preprocessing'.
- 10. Initialize a Logistic Regression model from 'sklearn. linear model'.
- 11. Train the Logistic Regression model using the training data with the 'fit' method.
- 12. Use the trained Logistic Regression model to make predictions on the testing data with the 'predict' method.
- 13. Evaluate the performance of the model using **metrics** such as accuracy, precision, recall, F1_Score, or others based on the requirements.
- 14. Finally, use the trained model to predict the outcome for the new data.

PROGRAM

Import Libraries

import pandas as pd

import numpy as np

import warnings

warnings. Simplefilter ("ignore")

Load the Dataset

df = pd. read_csv("C:\\Users\\Documents\\loan.csv")
df. head ()

| | Loan_ID | Gender | Married | Dependents | Education | Self_Employed | ApplicantIncome | CoapplicantIncome | LoanAmount | Loan_Amount_Term | Credit_History |
|---|----------|--------|---------|------------|-----------------|---------------|-----------------|-------------------|------------|------------------|----------------|
| 0 | LP001002 | Male | No | 0 | Graduate | No | 5849 | 0.0 | NaN | 360.0 | 1.0 |
| 1 | LP001003 | Male | Yes | 1 | Graduate | No | 4583 | 1508.0 | 128.0 | 360.0 | 1.0 |
| 2 | LP001005 | Male | Yes | 0 | Graduate | Yes | 3000 | 0.0 | 66.0 | 360.0 | 1.0 |
| 3 | LP001006 | Male | Yes | 0 | Not Graduate | No | 2583 | 2358.0 | 120.0 | 360.0 | 1.0 |
| 4 | LP001008 | Male | No | 0 | Graduate | No | 6000 | 0.0 | 141.0 | 360.0 | 1.0 |
| 4 | | | | | | | | | | | • |

df. info ()

Output

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 614 entries, 0 to 613
Data columns (total 13 columns):

| # | Column | Non-Null Count | Dtype | | | |
|---|------------------------|----------------|---------|--|--|--|
| | | | | | | |
| 0 | Loan_ID | 614 non-null | object | | | |
| 1 | Gender | 601 non-null | object | | | |
| 2 | Married | 611 non-null | object | | | |
| 3 | Dependents | 599 non-null | object | | | |
| 4 | Education | 614 non-null | object | | | |
| 5 | Self_Employed | 582 non-null | object | | | |
| 6 | ApplicantIncome | 614 non-null | int64 | | | |
| 7 | CoapplicantIncome | 614 non-null | float64 | | | |
| 8 | LoanAmount | 592 non-null | float64 | | | |
| 9 | Loan_Amount_Term | 600 non-null | float64 | | | |
| 10 | Credit_History | 564 non-null | float64 | | | |
| 11 | Property_Area | 614 non-null | object | | | |
| 12 | Loan_Status | 614 non-null | object | | | |
| dtypes: float64(4), int64(1), object(8) | | | | | | |
| memo | memory usage: 62.5+ KB | | | | | |
| | | | | | | |



df. describe ()

| | ApplicantIncome | CoapplicantIncome | LoanAmount | Loan_Amount_Term | Credit_History |
|-------|-----------------|-------------------|------------|------------------|----------------|
| count | 614.000000 | 614.000000 | 592.000000 | 600.00000 | 564.000000 |
| mean | 5403.459283 | 1621.245798 | 146.412162 | 342.00000 | 0.842199 |
| std | 6109.041673 | 2926.248369 | 85.587325 | 65.12041 | 0.364878 |
| min | 150.000000 | 0.000000 | 9.000000 | 12.00000 | 0.000000 |
| 25% | 2877.500000 | 0.000000 | 100.000000 | 360.00000 | 1.000000 |
| 50% | 3812.500000 | 1188.500000 | 128.000000 | 360.00000 | 1.000000 |
| 75% | 5795.000000 | 2297.250000 | 168.000000 | 360.00000 | 1.000000 |
| max | 81000.000000 | 41667.000000 | 700.000000 | 480.00000 | 1.000000 |

Explore and preprocess the data

df. isnull (). Sum ()

Output

| Loan_ID | 0 |
|-------------------|----|
| Gender | 13 |
| Married | 3 |
| Dependents | 15 |
| Education | 0 |
| Self_Employed | 32 |
| ApplicantIncome | 0 |
| CoapplicantIncome | 0 |
| LoanAmount | 22 |
| Loan_Amount_Term | 14 |
| Credit_History | 50 |
| Property_Area | 0 |
| Loan_Status | 0 |
| dtype: int64 | |

Drop the null values

df = df. drop ('Loan_ID', axis =1)
df. head ()

Output

| C | Sender | Married | Dependents | Education | Self_Employed | ApplicantIncome | CoapplicantIncome | LoanAmount | Loan_Amount_Term | Credit_History | Property_Ai |
|-----|--------|---------|------------|-----------------|---------------|-----------------|-------------------|------------|------------------|----------------|-------------|
| 0 | Male | No | 0 | Graduate | No | 5849 | 0.0 | NaN | 360.0 | 1.0 | Urt |
| 1 | Male | Yes | 1 | Graduate | No | 4583 | 1508.0 | 128.0 | 360.0 | 1.0 | Ru |
| 2 | Male | Yes | 0 | Graduate | Yes | 3000 | 0.0 | 66.0 | 360.0 | 1.0 | Urt |
| 3 | Male | Yes | 0 | Not Graduate | No | 2583 | 2358.0 | 120.0 | 360.0 | 1.0 | Urt |
| 4 | Male | No | 0 | Graduate | No | 6000 | 0.0 | 141.0 | 360.0 | 1.0 | Urt |
| 4 @ | | | | | | | | | | | > |

Encoding the categorical values

from sklearn. preprocessing import Label Encoder

label_encoder = LabelEncoder ()

categorical_columns = ['Gender', 'Married', 'Education', 'Self_Employed', 'Property_Area', 'Loan_Status'] df[categorical_columns] = df[categorical_columns]. apply(label_encoder.fit_transform) df[categorical_columns]

| | Gender | Married | Education | Self_Employed | Property_Area | Loan_Status |
|-----|--------|---------|-----------|---------------|---------------|-------------|
| 0 | 1 | 0 | 0 | 0 | 2 | 1 |
| 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| 2 | 1 | 1 | 0 | 1 | 2 | 1 |
| 3 | 1 | 1 | 1 | 0 | 2 | 1 |
| 4 | 1 | 0 | 0 | 0 | 2 | 1 |
| | | | | | | |
| 609 | 0 | 0 | 0 | 0 | 0 | 1 |
| 610 | 1 | 1 | 0 | 0 | 0 | 1 |
| 611 | 1 | 1 | 0 | 0 | 2 | 1 |
| 612 | 1 | 1 | 0 | 0 | 2 | 1 |
| 613 | 0 | 0 | 0 | 1 | 1 | 0 |

614 rows × 6 columns

Replace the null value

from sklearn. impute import SimpleImputer

numeric_columns = ['Gender', 'Married', 'Dependents', 'Self_Employed', 'LoanAmount', 'Loan_Amount_Term',
'Credit_History']

df[numeric_columns] = SimpleImputer(strategy='most_frequent'). fit_transform(df[numeric_columns])
df[numeric_columns]

Output

| | Gender | Married | Dependents | Self_Employed | LoanAmount | Loan_Amount_Term | Credit_History |
|-----|--------|---------|------------|---------------|------------|------------------|----------------|
| 0 | 1 | 0 | 0 | 0 | 120.0 | 360.0 | 1.0 |
| 1 | 1 | 1 | 1 | 0 | 128.0 | 360.0 | 1.0 |
| 2 | 1 | 1 | 0 | 1 | 66.0 | 360.0 | 1.0 |
| 3 | 1 | 1 | 0 | 0 | 120.0 | 360.0 | 1.0 |
| 4 | 1 | 0 | 0 | 0 | 141.0 | 360.0 | 1.0 |
| | | | | | | | |
| 609 | 0 | 0 | 0 | 0 | 71.0 | 360.0 | 1.0 |
| 610 | 1 | 1 | 3+ | 0 | 40.0 | 180.0 | 1.0 |
| 611 | 1 | 1 | 1 | 0 | 253.0 | 360.0 | 1.0 |
| 612 | 1 | 1 | 2 | 0 | 187.0 | 360.0 | 1.0 |
| 613 | 0 | 0 | 0 | 1 | 133.0 | 360.0 | 0.0 |

614 rows × 7 columns

Handle special case in 'Dependents' column

df['Dependents'] = df['Dependents']. replace ('3+', 3). astype(float) df['Dependents']

Output 0.0

0 1.0 1 2 0.0 3 0.0 4 0.0 609 0.0 610 3.0 611 1.0 2.0 612 613 0.0

Name: Dependents, Length: 614, dtype: float64

Split the data into features (x)

x = df. iloc [:, :-1]

X

Output

| | Gender | Married | Dependents | Education | Self_Employed | ApplicantIncome | CoapplicantIncome | LoanAmount | Loan_Amount_Term | Credit_History | Property_ |
|-------|----------|---------|------------|-----------|---------------|-----------------|-------------------|------------|------------------|----------------|-----------|
| 0 | 1 | 0 | 0.0 | 0 | 0 | 5849 | 0.0 | 120.0 | 360.0 | 1.0 | |
| 1 | 1 | 1 | 1.0 | 0 | 0 | 4583 | 1508.0 | 128.0 | 360.0 | 1.0 | |
| 2 | 1 | 1 | 0.0 | 0 | 1 | 3000 | 0.0 | 66.0 | 360.0 | 1.0 | |
| 3 | 1 | 1 | 0.0 | 1 | 0 | 2583 | 2358.0 | 120.0 | 360.0 | 1.0 | |
| 4 | 1 | 0 | 0.0 | 0 | 0 | 6000 | 0.0 | 141.0 | 360.0 | 1.0 | |
| | | | | | | | | | | | |
| 609 | 0 | 0 | 0.0 | 0 | 0 | 2900 | 0.0 | 71.0 | 360.0 | 1.0 | |
| 610 | 1 | 1 | 3.0 | 0 | 0 | 4106 | 0.0 | 40.0 | 180.0 | 1.0 | |
| 611 | 1 | 1 | 1.0 | 0 | 0 | 8072 | 240.0 | 253.0 | 360.0 | 1.0 | |
| 612 | 1 | 1 | 2.0 | 0 | 0 | 7583 | 0.0 | 187.0 | 360.0 | 1.0 | |
| 613 | 0 | 0 | 0.0 | 0 | 1 | 4583 | 0.0 | 133.0 | 360.0 | 0.0 | |
| 614 r | OWO × 11 | columns | | | | | | | | | |

614 rows × 11 columns

Split the data into target variable (y)

y = df. iloc [:, -1]

Output

Name: Loan_Status, Length: 614, dtype: int32

Further split the dataset into training and testing sets

```
from sklearn. model_selection import train_test_split x_train, x_test, y_train, y_test=train_test_split (x, y, random_state=0)
```

Feature scaling using StandardScaler

```
from sklearn. preprocessing import StandardScaler scaler = StandardScaler ()
x_train = scaler.fit_transform(x_train)
x_test = scaler. transform(x_test)
x_train
```

Output

```
array([[ 0.36547961,  0.6622422 ,  0.20631248, ...,  0.27778225,  0.41649656,  1.20186498],
[ 0.36547961, -1.42427431, -0.77207659, ...,  0.27778225,  0.41649656, -1.31684978],
[ 0.36547961, -1.42427431,  1.18470154, ...,  0.27778225,  0.41649656, -1.31684978],
...,
[ 0.36547961,  0.6622422 ,  2.1630906 , ...,  0.27778225,  0.41649656, -0.0574924 ],
[ 0.36547961,  0.6622422 , -0.77207659, ...,  0.27778225,  0.41649656,  1.20186498],
[ -2.00241646,  0.6622422 , -0.77207659, ...,  0.27778225,  0.41649656, -0.0574924 ]])
```

x test

Output

```
array([[ 0.36547961, -1.42427431, -0.77207659, ..., 0.27778225, 0.41649656, -0.0574924 ], [-2.00241646, -1.42427431, -0.77207659, ..., 0.27778225, 0.41649656, -0.0574924 ], [ 0.36547961, 0.6622422, -0.77207659, ..., 0.27778225, 0.41649656, 1.20186498], ..., [ 0.36547961, 0.6622422, -0.77207659, ..., 0.27778225, 0.41649656, 1.20186498], [ 0.36547961, -1.42427431, -0.77207659, ..., 0.27778225, 0.41649656, -0.0574924 ], [ 0.36547961, 0.6622422, 0.20631248, ..., 0.27778225, -2.40098019, -0.0574924 ]])
```

Intialize the Logistic Regression model

from sklearn. linear_model import LogisticRegression model = LogisticRegression ()

Train the Logistic Regression model

model.fit (x train, y train)

Output

LogisticRegression ()

Make predictions

y_pred=model. predict(x_test)
y_pred

Output

Evaluate the model

from sklearn. metrics import accuracy_score, classification_report accuracy = accuracy_score (y_test, y_pred) * 100 print ("Accuracy of the model is {:.2f}". format(accuracy))

Output

Accuracy: 83.77%

from sklearn. metrics import classification_report
class_report = classification_report (y_test, y_pred)
print (f"\nClassification Report:\n{class_report}")

Output

Classification Report:

| | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| 0 | 0.91 | 0.47 | 0.62 | 43 |
| 1 | 0.83 | 0.98 | 0.90 | 111 |
| accuracy | | | 0.84 | 154 |
| macro avg | 0.87 | 0.72 | 0.76 | 154 |
| weighted avg | 0.85 | 0.84 | 0.82 | 154 |

Predict for new data

new_data = [1, 1, 2, 1, 1, 4106.0, 240.0, 253.0, 360.0, 1, 2] predictions = model. predict([new_data]) print(predictions)

Output

[1]

Assuming a label encoder for decoding categorical predictions

decoded_predictions = label_encoder. inverse_transform(predictions)
print(decoded_predictions)

Output

['Y']



RESULT

Thus, the program to predict if a loan will get approved or not was completed successfully using Logistic Regression.

Ex. No:3 PREDICT THE TRAFFIC ON A NEW MODE OF TRANSPORT

AIM

To write a Python program to predict the traffic on a new mode of transport using Support Vector Machine.

PROCEDURE

- 1. Importing the necessary libraries, including 'pandas' for data manipulation, and 'numpy' for numerical operations and warnings.
- 2. Load the dataset containing traffic data. For Example, traffic dataset.
- 3. Explore the dataset to understand its structure and features. Preprocess the data if needed, such as handling missing values.
- 4. Use 'LabelEncoder' to encode categorical columns like 'Day of the week', and 'Traffic Situation'.
- 5. Split the dataset into features (x) and target variable (y).
- 6. Further split the dataset into training and testing sets using 'train_test_split' from 'sklearn. model selection'.
- 7. Standardize the features using 'StandardScaler' from 'sklearn. preprocessing'.
- 8. Initialize a Support Vector Machine model from 'sklearn.svm'.
- 9. Train the Support Vector Machine model using the training data with the 'fit' method.
- 10. Use the trained Support Vector Machine model to make predictions on the testing data with the 'predict' method.
- 11. Evaluate the performance of the model using **metrics** such as accuracy, precision, recall, F1_Score, or others based on the requirements.
- 12. Finally, use the trained model to predict the outcome for the new data.

PROGRAM

Import Libraries

import pandas as pd
import numpy as np

import warnings

warnings. Simplefilter ("ignore")

Load the Dataset

df = pd. read_csv("C:\\Users\\Documents\\Traffic.csv")
df. head()

| | Time | Date | Day of the week | CarCount | BikeCount | BusCount | TruckCount | Total | Traffic Situation |
|---|-------------|------|-----------------|----------|-----------|----------|------------|-------|-------------------|
| 0 | 12:00:00 AM | 10 | Tuesday | 31 | 0 | 4 | 4 | 39 | low |
| 1 | 12:15:00 AM | 10 | Tuesday | 49 | 0 | 3 | 3 | 55 | low |
| 2 | 12:30:00 AM | 10 | Tuesday | 46 | 0 | 3 | 6 | 55 | low |
| 3 | 12:45:00 AM | 10 | Tuesday | 51 | 0 | 2 | 5 | 58 | low |
| 4 | 1:00:00 AM | 10 | Tuesday | 57 | 6 | 15 | 16 | 94 | normal |

df.info()

Output

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2976 entries, 0 to 2975
Data columns (total 9 columns):

| # | Column | Non-Null Count | Dtype |
|---|-------------------|----------------|--------|
| | | | |
| 0 | Time | 2976 non-null | object |
| 1 | Date | 2976 non-null | int64 |
| 2 | Day of the week | 2976 non-null | object |
| 3 | CarCount | 2976 non-null | int64 |
| 4 | BikeCount | 2976 non-null | int64 |
| 5 | BusCount | 2976 non-null | int64 |
| 6 | TruckCount | 2976 non-null | int64 |
| 7 | Total | 2976 non-null | int64 |
| 8 | Traffic Situation | 2976 non-null | object |

dtypes: int64(6), object(3)
memory usage: 209.4+ KB

df. describe ()

| | Date | CarCount | BikeCount | BusCount | TruckCount | Total |
|-------|-------------|-------------|-------------|-------------|-------------|-------------|
| count | 2976.000000 | 2976.000000 | 2976.000000 | 2976.000000 | 2976.000000 | 2976.000000 |
| mean | 16.000000 | 68.696573 | 14.917339 | 15.279570 | 15.324933 | 114.218414 |
| std | 8.945775 | 45.850693 | 12.847518 | 14.341986 | 10.603833 | 60.190627 |
| min | 1.000000 | 6.000000 | 0.000000 | 0.000000 | 0.000000 | 21.000000 |
| 25% | 8.000000 | 19.000000 | 5.000000 | 1.000000 | 6.000000 | 55.000000 |
| 50% | 16.000000 | 64.000000 | 12.000000 | 12.000000 | 14.000000 | 109.000000 |
| 75% | 24.000000 | 107.000000 | 22.000000 | 25.000000 | 23.000000 | 164.000000 |
| max | 31.000000 | 180.000000 | 70.000000 | 50.000000 | 40.000000 | 279.000000 |

Explore and preprocess the data

df. isnull (). sum ()

Output

Time 0 Date 0 Day of the week 0 CarCount 0 BikeCount 0 BusCount 0 TruckCount Total 0 Traffic Situation 0 dtype: int64

Drop the null values

df = df. drop ('Time', axis =1)
df. head ()

Output

| | Date | Day of the week | CarCount | BikeCount | BusCount | TruckCount | Total | Traffic Situation |
|---|------|-----------------|----------|-----------|----------|------------|-------|-------------------|
| 0 | 10 | Tuesday | 31 | 0 | 4 | 4 | 39 | low |
| 1 | 10 | Tuesday | 49 | 0 | 3 | 3 | 55 | low |
| 2 | 10 | Tuesday | 46 | 0 | 3 | 6 | 55 | low |
| 3 | 10 | Tuesday | 51 | 0 | 2 | 5 | 58 | low |
| 4 | 10 | Tuesday | 57 | 6 | 15 | 16 | 94 | normal |

Encoding the categorical values

from sklearn. preprocessing import LabelEncoder

label_encoder = LabelEncoder ()

categorical_columns = ['Day of the week', 'Traffic Situation']

df[categorical_columns] = df[categorical_columns]. apply(label_encoder.fit_transform)

df[categorical_columns]

| | Day of the week | Traffic Situation |
|------|-----------------|-------------------|
| 0 | 5 | 2 |
| 1 | 5 | 2 |
| 2 | 5 | 2 |
| 3 | 5 | 2 |
| 4 | 5 | 3 |
| | | |
| 2971 | 4 | 3 |
| 2972 | 4 | 3 |
| 2973 | 4 | 3 |
| 2974 | 4 | 3 |
| 2975 | 4 | 3 |

2976 rows × 2 columns

Split the data into features (x)

x = df. iloc [:, :-1]

X

Output



| | Date | Day of the week | CarCount | BikeCount | BusCount | TruckCount | Total |
|------|------|-----------------|----------|-----------|----------|------------|-------|
| 0 | 10 | 5 | 31 | 0 | 4 | 4 | 39 |
| 1 | 10 | 5 | 49 | 0 | 3 | 3 | 55 |
| 2 | 10 | 5 | 46 | 0 | 3 | 6 | 55 |
| 3 | 10 | 5 | 51 | 0 | 2 | 5 | 58 |
| 4 | 10 | 5 | 57 | 6 | 15 | 16 | 94 |
| ••• | 500 | 1999 | *** | *** | | *** | *** |
| 2971 | 9 | 4 | 16 | 3 | 1 | 36 | 56 |
| 2972 | 9 | 4 | 11 | 0 | 1 | 30 | 42 |
| 2973 | 9 | 4 | 15 | 4 | 1 | 25 | 45 |
| 2974 | 9 | 4 | 16 | 5 | 0 | 27 | 48 |
| 2975 | 9 | 4 | 14 | 3 | 1 | 15 | 33 |

2976 rows × 7 columns

Split the data into target variable (y)

```
y = df. iloc [:, -1]
```

Output

```
2
0
1
          2
2
          2
3
          2
4
          3
2971
          3
2972
          3
2973
          3
          3
2974
```

Name: Traffic Situation, Length: 2976, dtype: int32

Further split the dataset into training and testing sets

from sklearn. model_selection **import** train_test_split x_train, x_test, y_train, y_test=train_test_split (x, y, random_state=0)

Feature scaling using StandardScaler

from sklearn. preprocessing import StandardScaler scaler = StandardScaler ()

x_train = scaler.fit_transform(x_train)

 $x_{test} = scaler. transform(x_{test})$

x_train

```
array([[-0.33600687, -1.59798223, 1.98010887, ..., 1.00899718, -1.1633432, 2.0733575],
        [ 0.43635908, -1.59798223, 2.30697357, ..., -0.31203497, -1.1633432, 2.20601196],
        [ 1.42940101, -0.10771914, 1.10846967, ..., 0.66135714, -0.68866549, 0.8628855],
        ...,
        [ 1.20872503, -1.59798223, -0.351526, ..., -0.590147, -0.02411669, -0.48024096],
        [-1.10837282, -1.10122786, -1.15779226, ..., -1.07684306, 0.83030319, -1.1766769],
        [-0.99803483, 0.88578959, 0.01892066, ..., -0.72920302, 1.30498091, -0.09885937]])
```

```
array([[-0.99803483, 0.88578959, -1.24495618, ..., -1.07684306, 1.21004537, -1.19325871],
[-1.66006278, 1.38254396, 0.38936732, ..., 0.17466108, 1.39991645, 0.43175849],
[-1.32904881, -0.6044735, -0.22078012, ..., 1.2175812, 0.92523874, 0.21619498],
...,
[ 1.31906302, -0.6044735, -1.07062834, ..., -0.17297896, 0.64043211, -0.91136797],
[ 0.32602109, 0.38903523, -1.13600128, ..., -1.00731505, 0.45056102, -1.20984052],
[ -1.4393868, -1.59798223, -0.351526, ..., -0.79873103, -0.02411669, -0.57973181]])
```

Intialize the Support Vector Machine model

from sklearn.svm import SVC model = SVC (kernel = 'linear', random_state = 0)

Train the Support Vector Machine model

model.fit (x_train, y_train)

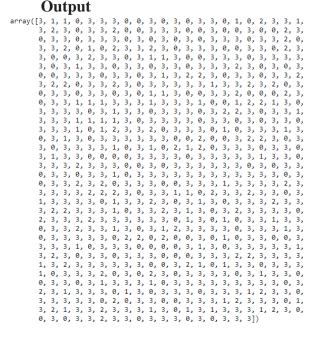
Output

SVC (kernel='linear', random_state=0)

Make predictions

y_pred=model. predict(x_test)
y_pred





Evaluate the model

from sklearn. metrics **import** accuracy_score, classification_report accuracy = accuracy_score (y_test, y_pred) * 100 print ("Accuracy of the model is {:.2f}". format(accuracy))

Output

Accuracy: 89.247312%

from sklearn. metrics import classification_report
class_report = classification_report (y_test, y_pred)
print (f"\nClassification Report:\n{class_report}")

Output

Classification Report:

| | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| 0 | 0.95 | 0.99 | 0.97 | 168 |
| 1 | 0.85 | 0.75 | 0.79 | 80 |
| 2 | 0.72 | 0.68 | 0.70 | 73 |
| 3 | 0.90 | 0.92 | 0.91 | 423 |
| accuracy | | | 0.89 | 744 |
| macro avg | 0.86 | 0.84 | 0.84 | 744 |
| weighted avg | 0.89 | 0.89 | 0.89 | 744 |



Predict for new data

new_data = [9, 5, 6, 3, 2, 5, 45]
predictions = model. predict([new_data])
print(predictions)

Output

[0]

Assuming a label encoder for decoding categorical predictions

decoded_predictions = label_encoder.inverse_transform(predictions)
print(decoded_predictions)

Output

['heavy']

RESULT

Thus, the program to predict the traffic on a new mode of transport was executed successfully using Support Vector Machine model.

Ex. No: 4 PREDICT THE CLASS OF USER

AIM

To write a Python program to predict the class of user using Decision Tree.

PROCEDURE

- 1. Importing the necessary libraries, including 'pandas' for data manipulation, and 'numpy' for numerical operations and warnings.
- 2. Load the dataset containing trip history data. For Example, Trip dataset.
- 3. Explore the dataset to understand its structure and features. Preprocess the data if needed, such as handling missing values.
- 4. Use 'LabelEncoder' to encode categorical columns like 'Start date', 'End station', 'Bike number' and 'Member type'.
- 5. Split the dataset into features (x) and target variable (y).
- 6. Further split the dataset into training and testing sets using 'train_test_split' from 'sklearn. model selection'.
- 7. Standardize the features using 'StandardScaler' from 'sklearn. preprocessing'.
- 8. Initialize a Decision Tree Classifier model from 'sklearn. tree'.
- 9. Train the Decision Tree Classifier model using the training data with the 'fit' method.
- 10. Use the trained Decision Tree Classifier model to make predictions on the testing data with the 'predict' method.
- 11. Evaluate the performance of the model using **metrics** such as accuracy, precision, recall, F1_Score, or others based on the requirements.
- 12. Finally, use the trained model to predict the outcome for the new data.

PROGRAM

Import Libraries

import pandas as pd

import numpy as np

import warnings

warnings. Simplefilter ("ignore")

Load the Dataset

df = pd. read_csv("C:\\Users\\Documents\\Trip.csv")
df. head ()

| | Duration | Start date | End date | Start station number | Start station | End station number | End station | Bike number | Member type |
|---|----------|---------------------|---------------------|----------------------|----------------------------------|--------------------|----------------------------------|----------------|----------------|
| 0 | 1012 | 20-09-2010 11:27 | 20-09-2010 11:43 | 31208 | M St & New Jersey Ave SE | 31108 | 4th & M St SW | W00742 | Member |
| 1 | 61 | 20-09-2010 11:41 | 20-09-2010 11:42 | 31209 | 1st & N St SE | 31209 | 1st & N St SE | W00032 | Member |
| 2 | 2690 | 20-09-2010 12:05 | 20-09-2010 12:50 | 31600 | 5th & K St NW | 31100 | 19th St & Pennsylvania Ave NW | W00993 | Member |
| 3 | 1406 | 20-09-2010 12:06 | 20-09-2010 12:29 | 31600 | 5th & K St NW | 31602 | Park Rd & Holmead Pl NW | W00344 | Member |
| 4 | 1413 | 20-09-2010 12:10 | 20-09-2010 12:34 | 31100 | 19th St & Pennsylvania Ave NW | 31201 | 15th & P St NW | W00883 | Member |

df.info()

Output

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 115597 entries, 0 to 115596
Data columns (total 9 columns):

| # | Column | Non-Null Count | Dtype |
|---|----------------------|-----------------|----------|
| | | | |
| 0 | Duration | 115597 non-null | int64 |
| 1 | Start date | 115597 non-null | object 📗 |
| 2 | End date | 115597 non-null | object 🖁 |
| 3 | Start station number | 115597 non-null | int64 |
| 4 | Start station | 115597 non-null | object |
| 5 | End station number | 115597 non-null | int64 |
| 6 | End station | 115597 non-null | object |
| 7 | Bike number | 115597 non-null | object |
| 8 | Member type | 115597 non-null | object |

dtypes: int64(3), object(6)
memory usage: 7.9+ MB

df. describe ()

| | Duration | Start station number | End station number |
|-------|---------------|----------------------|--------------------|
| count | 115597.000000 | 115597.000000 | 115597.000000 |
| mean | 1254.649956 | 31266.213431 | 31268.042250 |
| std | 2914.317998 | 187.645048 | 186.194316 |
| min | 60.000000 | 31000.000000 | 31000.000000 |
| 25% | 403.000000 | 31110.000000 | 31111.000000 |
| 50% | 665.000000 | 31213.000000 | 31214.000000 |
| 75% | 1120.000000 | 31301.000000 | 31238.000000 |
| max | 85644.000000 | 31805.000000 | 31805.000000 |

Explore and preprocess the data

df. isnull (). sum ()

Output

Duration 0
Start date 0
End date 0
Start station number 0
Start station 0
End station number 0
End station 0
Bike number 0
Member type 0
dtype: int64

Drop the null values

df = df. drop (['Start date', 'End date'], axis =1) df. head ()

Output

| | Duration | Start station number | Start station | End station number | End station | Bike number | Member type |
|---|----------|----------------------|-------------------------------|--------------------|-------------------------------|-------------|-------------|
| 0 | 1012 | 31208 | M St & New Jersey Ave SE | 31108 | 4th & M St SW | W00742 | Member |
| 1 | 61 | 31209 | 1st & N St SE | 31209 | 1st & N St SE | W00032 | Member |
| 2 | 2690 | 31600 | 5th & K St NW | 31100 | 19th St & Pennsylvania Ave NW | W00993 | Member |
| 3 | 1406 | 31600 | 5th & K St NW | 31602 | Park Rd & Holmead Pl NW | W00344 | Member |
| 4 | 1413 | 31100 | 19th St & Pennsylvania Ave NW | 31201 | 15th & P St NW | W00883 | Member |

Encoding the categorical values

from sklearn. preprocessing import LabelEncoder label_encoder = LabelEncoder () categorical_columns = ['Start station', 'End station', 'Bike number', 'Member type'] df[categorical_columns] = df[categorical_columns]. apply(label_encoder.fit_transform) df[categorical_columns]

| | Start station | End station | Bike number | Member type |
|--------|---------------|-------------|-------------|-------------|
| 0 | 85 | 50 | 614 | 1 |
| 1 | 32 | 33 | 41 | 1 |
| 2 | 52 | 31 | 836 | 1 |
| 3 | 52 | 94 | 282 | 1 |
| 4 | 30 | 21 | 734 | 1 |
| | | | | |
| 115592 | 35 | 65 | 716 | 0 |
| 115593 | 63 | 18 | 764 | 1 |
| 115594 | 93 | 18 | 819 | 1 |
| 115595 | 1 | 14 | 946 | 1 |
| 115596 | 1 | 1 | 636 | 0 |

115597 rows × 4 columns

Split the data into features (x)

x = df. iloc [:, :-1]

X





SRM

| 115592 | 2179 | 31110 | 35 | 31623 | 65 | 716 |
|--------|-------|-------|----|-------|----|-----|
| 115593 | 953 | 31106 | 63 | 31401 | 18 | 764 |
| 115594 | 737 | 31602 | 93 | 31401 | 18 | 819 |
| 115595 | 514 | 31111 | 1 | 31202 | 14 | 946 |
| 115596 | 51962 | 31111 | 1 | 31111 | 1 | 636 |

115597 rows × 6 columns

Split the data into target variable (y)

```
y = df. iloc [:, -1]
```

Output

```
0
           1
1
           1
2
            1
3
           1
           1
4
115592
           0
115593
           1
115594
           1
           1
115595
115596
           0
```

Name: Member type, Length: 115597, dtype: int32

Further split the dataset into training and testing sets

from sklearn. model_selection import train_test_split
x_train, x_test, y_train, y_test=train_test_split (x, y, random_state=0)

Feature scaling using StandardScaler

from sklearn. preprocessing import StandardScaler scaler = StandardScaler ()

x train = scaler.fit transform(x train)

 $x_{test} = scaler. transform(x_{test})$

x train

```
array([[ 0.54640202, -1.40804138, -0.50762432, 1.8909801 , 0.11621975, -1.36635173],
[-0.32533639, -1.41337456, -0.81044688, -1.36975376, 0.69092124, -1.47913197],
[-0.14892163, -0.28274243, -0.84409383, -0.89702793, -1.10079519, 1.07843294],
...,
[-0.21022131, -0.16007951, -0.13750786, -0.30074876, -0.35706384, -1.51915077],
[-0.18028425, 1.85052573, 0.77095983, 1.86949256, -0.05281011, 0.57637894],
[-0.35277869, -0.35207365, 1.30931104, -0.35983949, -0.93176534, 0.82740594]])
```

x_test

Output

Intialize the Decision Tree model

from sklearn.tree import DecisionTreeClassifier model = DecisionTreeClassifier ()

Train the Decision Tree model

model.fit (x_train, y_train)

Output

DecisionTreeClassifier ()

Make predictions

y_pred=model. predict(x_test)
y_pred

Output

array ([1, 1, 1, ..., 1, 1, 0])

Evaluate the model

from sklearn. metrics **import** accuracy_score, classification_report accuracy = accuracy_score (y_test, y_pred) * 100 print ("Accuracy of the model is {:.2f}". format(accuracy))

Output

Accuracy: 79.71%

from sklearn. metrics import classification_report
class_report = classification_report (y_test, y_pred)
print (f"\nClassification Report:\n{class_report}")

Output

Classification Report:

| | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| 0 | 0.52 | 0.54 | 0.53 | 6089 |
| 1 | 0.88 | 0.87 | 0.87 | 22811 |
| accuracy | | | 0.80 | 28900 |
| macro avg | 0.70 | 0.70 | 0.70 | 28900 |
| weighted avg | 0.80 | 0.80 | 0.80 | 28900 |

Predict for new data

new_data = [737, 31110, 63, 31623, 18, 636]
predictions = model. predict([new_data])
print(predictions)

Output

[1]

Assuming a label encoder for decoding categorical predictions

decoded_predictions = label_encoder. inverse_transform(predictions)
print(decoded_predictions)

Output

['Member']

RESULT

Thus, the program to predict the class of user was completed successfully using Decision Tree Classifier.

Ex. No: 5 IDENTIFY THE TWEETS WHICH ARE HATE TWEETS AND WHICH ARE NOT.

AIM

To write a Python program to identify the tweets which are hate tweets and which are not using Multinomial Naïve Bayes model.

PROCEDURE

- 1. Importing the necessary libraries, including 'pandas' for data manipulation, and 'numpy' for numerical operations and warnings.
- 2. Load the dataset containing twitter data. For Example, Twitter dataset.
- 3. Explore the dataset to understand its structure and features. Preprocess the data if needed, such as handling missing values.
- 4. Split the dataset into features (x) and target variable (y).
- 5. Further split the dataset into training and testing sets using 'train_test_split' from 'sklearn. model selection'.
- 6. Use CountVectorizer from sklearn to convert the text features in the training data into a matrix of token counts and apply the same CountVectorizer to transform the text feature in the test data.
- 7. Initialize a Multinomial Naïve Bayes model from 'sklearn. naive_bayes'.
- 8. Train the Multinomial Naïve Bayes model using the training data with the 'fit' method.
- 9. Use the trained Multinomial Naïve Bayes model to make predictions on the testing data with the 'predict' method.
- 10. Evaluate the performance of the model using **metrics** such as accuracy, precision, recall, F1_Score, or others based on the requirements.
- 11. Finally, use the trained model to predict the outcome for the new data.

PROGRAM

Import Libraries

import pandas as pd
import numpy as np
import warnings

warnings. Simplefilter ("ignore")

Load the Dataset

df = pd. read_csv("C:\\Users\\Documents\\Twitter.csv")
df. head ()

| | status_id | text | created_at | favorite_count | retweet_count | location | followers_count | friends_count | statuses_count | category |
|---|---------------------|---|--------------------------|----------------|---------------|-----------------|-----------------|---------------|----------------|----------|
| 0 | 1046207313588236290 | Entitled, obnoxious, defensive, lying weasel | 2018-09- 30T01:17:15Z | 5 | 1 | McAllen, TX | 2253 | 2303 | 23856 | 0 |
| 1 | 1046207328113086464 | Thank you and for what you did for the women | 2018-09- 30T01:17:19Z | 5 | 2 | Tampa, FL | 2559 | 4989 | 19889 | 0 |
| 2 | 1046207329589493760 | Knitting (s) & Description (s) & Amp; getting ready for January 1 | 2018-09- 30T01:17:19Z | 0 | 0 | St Cloud, MN | 16 | 300 | 9 | 0 |
| 3 | 1046207341283168256 | Yep just like triffeling women weaponized thei | 2018-09- 30T01:17:22Z | 1 | 0 | flyover | 3573 | 3732 | 38361 | 1 |
| 4 | 1046207347016826880 | No, the President wants to end movement posin | 2018-09- 30T01:17:23Z | 0 | 0 | World | 294 | 312 | 7635 | 0 |

df.info ()

Output

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 807174 entries, 0 to 807173
Data columns (total 10 columns):

| Data | cordinis (cocar i | o corumns). | |
|------|-------------------|-----------------|--------|
| # | Column | Non-Null Count | Dtype |
| | | | |
| 0 | status_id | 807174 non-null | int64 |
| 1 | text | 803638 non-null | object |
| 2 | created_at | 807174 non-null | object |
| 3 | favorite_count | 807174 non-null | int64 |
| 4 | retweet_count | 807174 non-null | int64 |
| 5 | location | 616394 non-null | object |
| 6 | followers_count | 807174 non-null | int64 |
| 7 | friends_count | 807174 non-null | int64 |
| 8 | statuses_count | 807174 non-null | int64 |
| 9 | category | 807174 non-null | int64 |



9 category 807174 dtypes: int64(7), object(3) memory usage: 61.6+ MB

df. describe ()

| | status_id | favorite_count | retweet_count | followers_count | friends_count | statuses_count | category |
|-------|--------------|----------------|---------------|-----------------|---------------|----------------|---------------|
| count | 8.071740e+05 | 807174.000000 | 807174.000000 | 8.071740e+05 | 807174.000000 | 8.071740e+05 | 807174.000000 |
| mean | 1.061411e+18 | 6.466671 | 2.557508 | 4.663755e+04 | 6306.597207 | 4.793559e+04 | 0.118108 |
| std | 1.428176e+16 | 159.865656 | 50.724140 | 6.111813e+05 | 40549.763583 | 1.646474e+05 | 0.322737 |
| min | 1.046207e+18 | 0.000000 | 0.000000 | 0.000000e+00 | 0.000000 | 1.000000e+00 | 0.000000 |
| 25% | 1.050184e+18 | 0.000000 | 0.000000 | 1.070000e+02 | 160.000000 | 1.663250e+03 | 0.000000 |
| 50% | 1.054643e+18 | 0.000000 | 0.000000 | 5.390000e+02 | 528.000000 | 8.007000e+03 | 0.000000 |
| 75% | 1.071177e+18 | 1.000000 | 0.000000 | 2.844000e+03 | 1756.000000 | 3.317500e+04 | 0.000000 |
| max | 1.097647e+18 | 70385.000000 | 17484.000000 | 5.457643e+07 | 899383.000000 | 9.565126e+06 | 1.000000 |

Explore and preprocess the data

df. isnull (). sum ()

Output

```
status id
                          0
text
                      3536
created at
                          0
favorite count
                          0
retweet count
                          0
location
                    190780
followers_count
                          0
friends count
                          0
statuses count
                          0
                          0
category
dtype: int64
```

Replace the null value

df['text']. fillna (", inplace=True)
df['text']

Output

```
Entitled, obnoxious, defensive, lying weasel. ...
0
1
          Thank you and for what you did for the women...
2
         Knitting (s) & amp; getting ready for January 1...
         Yep just like triffeling women weaponized thei...
3
         No, the President wants to end movement posin...
4
         Let's not forget that this "iconic kiss" was u...
807169
807170
         DEFINITELY....the only one any of us should su...
         Did the movement count the dollars of Erin An...
807171
         This is one of my all time fav songs & amp; vid...
807172
           I watched your news on the death of the sailo...
807173
Name: text, Length: 807174, dtype: object
```

df['location']. fillna (", inplace=True) df['location']

```
0
                  McAllen, TX
1
                    Tampa, FL
                 St Cloud, MN
2
3
              flyover country
4
                        World
807169
                South Florida
807170
807171
                       Philly
          Berlin, Deutschland
807172
           Massachusetts, USA
807173
Name: location, Length: 807174, dtype: object
```

Drop the null values

df = df. drop (['status_id', 'created_at'], axis =1)
df. head ()

Output

| | text | favorite_count | retweet_count | location | followers_count | friends_count | statuses_count | category |
|---|---|----------------|---------------|-----------------|-----------------|---------------|----------------|----------|
| 0 | Entitled, obnoxious, defensive, lying weasel | 5 | 1 | McAllen, TX | 2253 | 2303 | 23856 | 0 |
| 1 | Thank you and for what you did for the women | 5 | 2 | Tampa, FL | 2559 | 4989 | 19889 | 0 |
| 2 | Knitting (s) & amp; getting ready for January 1 | 0 | 0 | St Cloud, MN | 16 | 300 | 9 | 0 |
| 3 | Yep just like triffeling women weaponized thei | 1 | 0 | flyover country | 3573 | 3732 | 38361 | 1 |
| 4 | No, the President wants to end movement posin | 0 | 0 | World | 294 | 312 | 7635 | 0 |

Split the data into features (x)

x = df. iloc [:, :-1]

X

Output

| | text | favorite_count | retweet_count | location | followers_count | friends_count | statuses_count |
|--------|---|----------------|---------------|---------------------|-----------------|---------------|----------------|
| 0 | Entitled, obnoxious, defensive, lying weasel | 5 | 1 | McAllen, TX | 2253 | 2303 | 23856 |
| 1 | Thank you and for what you did for the women | 5 | 2 | Tampa, FL | 2559 | 4989 | 19889 |
| 2 | Knitting (s) & amp; getting ready for January 1 | 0 | 0 | St Cloud, MN | 16 | 300 | 9 |
| 3 | Yep just like triffeling women weaponized thei | 1 | 0 | flyover country | 3573 | 3732 | 38361 |
| 4 | No, the President wants to end movement posin | 0 | 0 | World | 294 | 312 | 7635 |
| | | *** | | | | | |
| 807169 | Let's not forget that this "iconic kiss" was u | 2 | 0 | South Florida | 206 | 412 | 1247 |
| 807170 | DEFINITELYthe only one any of us should su | 3 | 0 | | 63 | 6 | 137 |
| 807171 | Did the movement count the dollars of Erin An | 0 | 0 | Philly | 2721 | 3509 | 66966 |
| 807172 | This is one of my all time fav songs & amp; vid | 1 | 1 | Berlin, Deutschland | 2683 | 1011 | 15455 |
| 807173 | I watched your news on the death of the sailo | 1 | 0 | Massachusetts, USA | 237 | 741 | 789 |

807174 rows × 7 columns

Split the data into target variable (y)

y = df. iloc [:, -1]

Output

Name: category, Length: 807174, dtype: int64

Further split the dataset into training and testing sets

from sklearn. model_selection import train_test_split

x_train, x_test, y_train, y_test=train_test_split (x, y, random_state=0)

Use CountVectorizer for text feature on training data

from sklearn. feature_extraction.text import CountVectorizer vectorizer = CountVectorizer ()
x_train_text = vectorizer.fit_transform(x_train['text'])
x_train_text

Output

<605380x119250 sparse matrix of type '<class 'numpy.int64'>'
 with 10994435 stored elements in Compressed Sparse Row format>

Apply the same CountVectorizer to transform the text feature in the test data

x_test_text = vectorizer. transform(x_test['text'])
x_test_text

Output

<201794x119250 sparse matrix of type '<class 'numpy.int64'>'
 with 3641705 stored elements in Compressed Sparse Row format>

Intialize the Naïve Bayes model

from sklearn. naive_bayes import MultinomialNB model = MultinomialNB ()

Train the Naïve Bayes model

model.fit (x_train, y_train)

Output

MultinomialNB ()

Make predictions

y_pred = model. predict(x_test_text)
y_pred

Output

array ([0, 0, 0, ..., 0, 0, 0], dtype=int64)

Evaluate the model

from sklearn. metrics **import** accuracy_score, classification_report accuracy = accuracy_score (y_test, y_pred) * 100 print ("Accuracy of the model is {:.2f}". format(accuracy))

Output

Accuracy: 93.11%

from sklearn. metrics import classification_report
class_report = classification_report (y_test, y_pred)
print (f"\nClassification Report:\n{class_report}")

Output

| Classification | Report: precision | recall | f1-score | support |
|----------------|----------------------|--------|----------|---------|
| 0 | 0.96 | 0.96 | 0.96 | 178149 |
| 1 | 0.70 | 0.71 | 0.71 | 23645 |
| accuracy | | | 0.93 | 201794 |
| macro avg | 0.83 | 0.84 | 0.83 | 201794 |
| weighted avg | 0.93 | 0.93 | 0.93 | 201794 |



Thus, the program to identify the tweets which are hate tweets and which are not was completed successfully using Multinomial Naïve Bayes model.

PREDICT THE AGE OF THE ACTORS

AIM

To write a Python program to predict the age of the actors using Linear Regression.

PROCEDURE

Ex. No: 6

- 1. Importing the necessary libraries, including 'pandas' for data manipulation, and 'numpy' for numerical operations and warnings.
- 2. Load the dataset containing age of actor's data. For Example, age dataset.
- 3. Explore the dataset to understand its structure and features. Preprocess the data if needed, such as handling missing values.
- 4. In the preprocessing, convert the 'birthday' column to a datetime format and also extract the birth year from the 'birth year' from the 'birthday' column to create a new 'birth_year' column.
- 5. Use 'LabelEncoder' to encode categorical columns like 'name', and 'Character gender'.
- 6. Use 'SimpleImputer' fro, sklearn to replace null values in numerical columns ('birth_year', 'actor age') with the most frequent values.
- 7. Split the dataset into features (x) and target variable (y).
- 8. Further split the dataset into training and testing sets using 'train_test_split' from 'sklearn. model selection'.
- 9. Initialize a Linear Regression model from 'sklearn. linear_model'.
- 10. Train the Linear Regression model using the training data with the 'fit' method.
- 11. Use the trained Linear Regression model to make predictions on the testing data with the 'predict' method.
- 12. Evaluate the performance of the model using **metrics** such as mean_squared_error, or others based on the requirements.
- 13. Finally, use the trained model to predict the outcome for the new data.

PROGRAM

Import Libraries

import pandas **as** pd

import numpy as np

import warnings

warnings. Simplefilter ("ignore")

Load the Dataset

df = pd. read_csv("C:\\Users\\Documents\\age.csv")
df. head ()

| | name | birthday | title | character_name | character_year | characterage | character_gender | love_interest | release_date | actor_age |
|---|---------------------|----------------|-------------------|---------------------|----------------|--------------|------------------|---|--------------|-----------|
| 0 | Ben Platt | 24-09- 1993 | The Politician | Payton Hobart | hs senior | NaN | M | Alice Charles, Astrid Sloan, River Barkley | 27-09-2019 | 26.0 |
| 1 | Zoey Deutch | 10-11- 1994 | The Politician | Infinity Jackson | hs senior | NaN | F | NaN | 27-09-2019 | 24.0 |
| 2 | Lucy Boynton | 17-01- 1994 | The Politician | Astrid Sloan | hs senior | NaN | F | Payton Hobart, River Barkley | 27-09-2019 | 25.0 |
| 3 | Julia Schlaepfer | 03-03- 1995 | The Politician | Alice Charles | hs senior | NaN | F | Payton Hobart, James Sullivan | 27-09-2019 | 24.0 |
| 4 | Laura Dreyfuss | 22-08- 1988 | The Politician | McAfee Westbrook | hs senior | NaN | F | Skye Leighton | 27-09-2019 | 31.0 |

df.info()

Output

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 243 entries, 0 to 242
Data columns (total 10 columns):

| # | Column | Non-Null Count | Dtype | |
|------|-------------------|----------------|----------|--|
| | | | | |
| 0 | name | 243 non-null | object | |
| 1 | birthday | 227 non-null | object 🤲 | |
| 2 | title | 243 non-null | object | |
| 3 | character_name | 243 non-null | object | |
| 4 | character_year | 111 non-null | object | |
| 5 | characterage | 131 non-null | float64 | |
| 6 | character_gender | 243 non-null | object | |
| 7 | love_interest | 132 non-null | object | |
| 8 | release_date | 243 non-null | object 🣂 | |
| 9 | actor_age | 227 non-null | float64 | |
| dtun | oc. float(4/2) ob | inc+(0) | | |

dtypes: float64(2), object(8)

memory usage: 19.1+ KB

df. describe ()

| | characterage | actor_age |
|-------|--------------|------------|
| count | 131.000000 | 227.000000 |
| mean | 16.580153 | 21.977974 |
| std | 1.518877 | 3.908743 |
| min | 10.000000 | 11.000000 |
| 25% | 16.000000 | 20.000000 |
| 50% | 17.000000 | 22.000000 |
| 75% | 17.000000 | 24.000000 |
| max | 25.000000 | 32.000000 |

Explore and preprocess the data

df. isnull (). sum ()

Output

```
name
                       0
birthday
                      16
title
                       0
character_name
                       0
character_year
                     132
characterage
                     112
character_gender
                       0
love interest
                     111
release date
                       0
actor_age
                      16
dtype: int64
```

Preprocessing

Extract year from birthday

df['birthday'] = pd.to_datetime(df['birthday']) df['birthday']

Output

```
0
      1993-09-24
1
      1994-11-10
2
      1994-01-17
3
      1995-03-03
4
      1988-08-22
238
             NaT
239
      1991-10-19
240
      1994-11-09
241
      1998-04-20
242
             NaT
Name: birthday, Length: 243, dtype: datetime64[ns]
```

df['birth_year'] = df['birthday'].dt. year df['birth_year']

```
0
       1993.0
1
       1994.0
2
       1994.0
3
       1995.0
4
       1988.0
238
         NaN
239
       1991.0
240
       1994.0
241
       1998.0
242
         NaN
Name: birth_year, Length: 243, dtype: float64
```

Drop the null values

SRM

df

Output

| | name | character_gender | actor_age | birth_year |
|-----|------------------------|------------------|-----------|------------|
| 0 | Ben Platt | M | 26.0 | 1993.0 |
| 1 | Zoey Deutch | F | 24.0 | 1994.0 |
| 2 | Lucy Boynton | F | 25.0 | 1994.0 |
| 3 | Julia Schlaepfer | F | 24.0 | 1995.0 |
| 4 | Laura Dreyfuss | F | 31.0 | 1988.0 |
| | | | | |
| 238 | Thomas Mitchell Barnet | M | NaN | NaN |
| 239 | Kevin Alves | M | 28.0 | 1991.0 |
| 240 | Asha Bromfield | F | 25.0 | 1994.0 |
| 241 | Felix Mallard | М | 21.0 | 1998.0 |
| 242 | Hallea Jones | F | NaN | NaN |

243 rows × 4 columns

Encoding the categorical values

from sklearn. preprocessing import LabelEncoder label_encoder = LabelEncoder ()

categorical_columns = ['name', 'character_gender']

df[categorical_columns] = df[categorical_columns]. apply(label_encoder.fit_transform)
df[categorical_columns]

Output

| | name | character_gender |
|-----|------|------------------|
| 0 | 24 | 1 |
| 1 | 239 | 0 |
| 2 | 151 | 0 |
| 3 | 126 | 0 |
| 4 | 142 | 0 |
| | | |
| 238 | 229 | 1 |
| 239 | 135 | 1 |
| 240 | 18 | 0 |
| 241 | 80 | 1 |
| 242 | 94 | 0 |
| | | |

243 rows × 2 columns

Explore the null values in the data

df. isnull (). sum ()

Output

| Output | |
|------------------|----|
| name | 0 |
| character_gender | 0 |
| actor_age | 16 |
| birth_year | 16 |
| dtype: int64 | |

Replace the null values

from sklearn. impute import SimpleImputer
numeric_columns = ['birth_year','actor_age']
df[numeric_columns] = SimpleImputer(strategy='most_frequent'). fit_transform(df[numeric_columns])
df[numeric_columns]

Output

| | birth_year | actor_age |
|-----|------------|-----------|
| 0 | 1993.0 | 26.0 |
| 1 | 1994.0 | 24.0 |
| 2 | 1994.0 | 25.0 |
| 3 | 1995.0 | 24.0 |
| 4 | 1988.0 | 31.0 |
| | | |
| 238 | 1996.0 | 22.0 |
| 239 | 1991.0 | 28.0 |
| 240 | 1994.0 | 25.0 |
| 241 | 1998.0 | 21.0 |
| 242 | 1996.0 | 22.0 |
| | | |



243 rows × 2 columns

Split the data into features (x)

 $\begin{aligned} x &= df[\text{['name', 'character_gender', 'birth_year']}] \\ x \end{aligned}$

| | name | character_gender | birth_year |
|-----|------|------------------|------------|
| 0 | 24 | 1 | 1993.0 |
| 1 | 239 | 0 | 1994.0 |
| 2 | 151 | 0 | 1994.0 |
| 3 | 126 | 0 | 1995.0 |
| 4 | 142 | 0 | 1988.0 |
| | | | |
| 238 | 229 | 1 | 1996.0 |
| 239 | 135 | 1 | 1991.0 |
| 240 | 18 | 0 | 1994.0 |
| 241 | 80 | 1 | 1998.0 |
| 242 | 94 | 0 | 1996.0 |

243 rows × 3 columns

Split the data into target variable (y)

y = df['actor_age'] y

| | Output |
|-----|--------|
| 0 | 26.0 |
| 1 | 24.0 |
| 2 | 25.0 |
| 3 | 24.0 |
| 4 | 31.0 |
| | |
| 238 | 22.0 |
| 239 | 28.0 |
| 240 | 25.0 |
| 241 | 21.0 |

22.0

242

Name: actor_age, Length: 243, dtype: float64

Further split the dataset into training and testing sets

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=42)

Intialize the Linear Regression model

from sklearn. linear_model import LinearRegression model = LinearRegression ()

Train the Linear Regression model

model.fit (x_train, y_train)

Output

LinearRegression ()

Make predictions

```
y_pred = model. predict(x_test)
y_pred
```

Output

```
array([20.09904337, 23.90720926, 20.81859233, 17.27447575, 20.69497398, 19.17478424, 20.17372696, 19.32125732, 26.15894389, 20.06672071, 23.63164979, 26.99190733, 23.06216532, 20.95792124, 21.86855069, 20.60029112, 21.8311261, 26.20693847, 19.70247812, 20.88760364, 21.86063193, 21.31628731, 21.96911008, 20.55001143, 19.37308618, 20.47177507, 28.29322551, 20.24192505, 20.99893724, 20.76190441, 21.17414158, 20.93918962, 20.9083775, 22.36906249, 21.07435678, 21.26959902, 23.92308542, 20.81785638, 23.86149981, 21.93115378, 22.94291295, 20.06312931, 20.71599071, 21.13390016, 20.34685042, 22.32955702, 20.15422076, 20.61032939, 21.4240295])
```

Evaluate the model

from sklearn. metrics import mean_squared_error mse = mean_squared_error (y_test, y_pred) print (f"Mean Squared Error: {mse}")

Output

Mean Squared Error: 9.760123797821025

Predict the new data

```
new_data = pd. DataFrame ({"name": [142], "character_gender": [1], "birth_year": [1996.0]})
predicted_age = model. predict(new_data)
print (f"Predicted Actor Age: {predicted_age [0]}")
```

Output

Predicted Actor Age: 21.007426345958265

RESULT

Thus, the program to predict the age of the actors was completed successfully using Linear Regression.

Ex. No: 7 INTRODUCTION TO PYTHON LIBRARIES - NUMPY, PANDAS, MATPLOTLIB, SCIKIT

AIM

The aim to study the Python Libraries such as Numpy for numerical operations, Pandas for data manipulation and analysis, Matplotlib for data visualization, Scikit – Learn for machine learning tasks.

PROCEDURE

1. Numpy

- Numpy is used for numerical operations in Python.
- It provides support for large, multi-dimensional arrays, along with mathematical functions to operate on these arrays.

Program

```
import numpy as np
arr = np. array ([1, 2, 3, 4, 5])
print ("Numpy Array:", arr)
```

Output

Numpy Array: [1 2 3 4 5]



2. Pandas

- Pandas is a powerful library for data manipulation and analysis.
- It introduces two primary data structures: Series (1D labeled array) and DataFrame (2D labeled table).
- Procedures including loading data, exploring data using methods like info (), head (), and performing operations like dropping null values.

Program

```
import pandas as pd
data = {'Name': ['Alice', 'Bob', 'Charlie'], 'Age': [25, 30, 35]}
df = pd. DataFrame(data)
print ("Pandas DataFrame:")
print(df)
```

```
Pandas DataFrame:
Name Age
Alice 25
Bob 30
Charlie 35
```

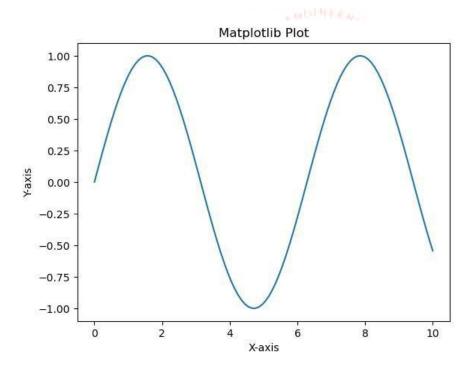
3. Matplotlib

- Matplotlib is a popular plotting library for creating static, interactive, and animated visualizations.
- The focus is on using Matplotlib to create basic plots like line plots, scatter plots, and bar plots.

Program

```
import numpy as np
import matplotlib. pyplot as plt
x = np. linspace (0, 10, 100)
y = np. sin (x)
plt. plot (x, y)
plt. title ("Matplotlib Plot")
plt. xlabel ("X-axis")
plt. ylabel ("Y-axis")
plt. show ()
```

Output



4. Scikit-Learn

- Scikit-Learn is a machine learning library that provides simple and efficient tools for data analysis and modeling.
- Procedures involve splitting data into training and testing sets using **train_test_split**, initializing and training a machine learning model (e.g., Linear regression), making predictions, and evaluating model performance.

Program

Dataset

```
x = np. array ([1, 2, 3, 4, 5]). reshape (-1, 1) 
y = np. array ([2, 4, 5, 4, 5])
```

Split the training and testing sets

```
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 = 42)
```

Initialize the Linear Regression Model

from sklearn. linear_model import LinearRegression model = LinearRegression ()

Train the Linear regression Model

model.fit (x_train, y_train)

Output

LinearRegression ()

Make Predictions

prediction = model. predict(x_test)
print ("Scikit-Learn Prediction:", prediction)

Evaluate the model

```
from sklearn. metrics import mean_absolute_error, mean_squared_error, root_mean_sqerr, R_squared mae = mean_absolute_error (y_test, y_pred) print (f'Mean Absolute Error: {mae}')
```

```
mse = mean_squared_error (y_test, y_pred)
print (f'Mean Squared Error: {mse}')
```

Output

Mean Absolute Error: 85.71428571428572 Mean Squared Error: 73.46938775510206

Result

Thus, the basic usage of Numpy, Pandas, Matplotlib, and Scikit-Lear for machine learning tasks was studied successfully.

Ex. No: 8 PERFORM DATA EXPLORATION AND PREPROCESSING IN PYTHON

AIM

The aim is to perform the data exploration and preprocessing techniques in Python using a real-world dataset.

PROCEDURE

1. Import necessary libraries

Importing the necessary libraries, including 'pandas' for data manipulation, and 'numpy' for numerical operations.

Program

import numpy as np import pandas as pd

2. Load the dataset

Load the real-world sample dataset to explore and preprocess the data for better understanding. For Example, Data.CSV dataset.

Program

 $dataset = pd. \ read_csv \ ("C:\\\\\\\\\\) Documents\ \ \ DSC \ LAB\ \ \ Dataset\ \ \ \ \ dataset$

| | Country | Age | Salary | Purchased |
|---|---------|------|---------|-----------|
| 0 | France | 44.0 | 72000.0 | No |
| 1 | Spain | 27.0 | 48000.0 | Yes |
| 2 | Germany | 30.0 | 54000.0 | No |
| 3 | Spain | 38.0 | 61000.0 | No |
| 4 | Germany | 40.0 | NaN | Yes |
| 5 | France | 35.0 | 58000.0 | Yes |
| 6 | Spain | NaN | 52000.0 | No |
| 7 | France | 48.0 | 79000.0 | Yes |
| 8 | Germany | 50.0 | 83000.0 | No |
| 9 | France | 37.0 | 67000.0 | Yes |

3. Display the few rows to understand the data structure

Display the few rows to understand the data structure of the data using **head** () method.

Program

dataset. head (10)

Output

| | Country | Age | Salary | Purchased |
|---|---------|------|---------|-----------|
| 0 | France | 44.0 | 72000.0 | No |
| 1 | Spain | 27.0 | 48000.0 | Yes |
| 2 | Germany | 30.0 | 54000.0 | No |
| 3 | Spain | 38.0 | 61000.0 | No |
| 4 | Germany | 40.0 | NaN | Yes |
| 5 | France | 35.0 | 58000.0 | Yes |
| 6 | Spain | NaN | 52000.0 | No |
| 7 | France | 48.0 | 79000.0 | Yes |
| 8 | Germany | 50.0 | 83000.0 | No |
| 9 | France | 37.0 | 67000.0 | Yes |

4. Analyze and view the information in tha data

To view the information of the data using **info** () method.

Program

dataset.info ()

Output

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10 entries, 0 to 9
Data columns (total 4 columns):

| Duca | COTAMINS (| cocar + coramins). | |
|------|------------|--------------------|---------|
| # | Column | Non-Null Count | Dtype |
| | | | |
| 0 | Country | 10 non-null | object |
| 1 | Age | 9 non-null | float64 |
| 2 | Salary | 9 non-null | float64 |
| 3 | Purchased | 10 non-null | object |
| | | | |

dtypes: float64(2), object(2)
memory usage: 452.0+ bytes

SHIM

5. Analyze and view the statistics information of the data

To view the statistics information of the data using **describe** () method.

Program

dataset. describe ()

Output

| | Age | Salary |
|-------|-----------|--------------|
| count | 9.000000 | 9.000000 |
| mean | 38.777778 | 63777.777778 |
| std | 7.693793 | 12265.579662 |
| min | 27.000000 | 48000.000000 |
| 25% | 35.000000 | 54000.000000 |
| 50% | 38.000000 | 61000.000000 |
| 75% | 44.000000 | 72000.000000 |
| max | 50.000000 | 83000.000000 |

6. Split the dataset into features (x)

From the dataset, split the independent variable (x).

Program

x = dataset. iloc [:, :-1].values
print(x)



Output

[['France' 44.0 72000.0]
['Spain' 27.0 48000.0]
['Germany' 30.0 54000.0]
['Spain' 38.0 61000.0]
['Germany' 40.0 nan]
['France' 35.0 58000.0]
['Spain' nan 52000.0]
['France' 48.0 79000.0]
['Germany' 50.0 83000.0]
['France' 37.0 67000.0]

7. Split the dataset into target variable (y)

From the dataset, split the dependent variable (y).

Program

y = dataset. iloc [:, 3].values print(y)

Output

['No' 'Yes' 'No' 'No' 'Yes' 'Yes' 'No' 'Yes' 'No' 'Yes']

8. Handling missing Data

Identify missing values using 'dataset. isnull (). sum ()'.

Program

```
print (dataset. isnull (). sum ())
```

Output

Country 0 Age 1 Salary 1 Purchased 0 dtype: int64

9. Drop missing values

Drop missing values from the dataset using 'dataset. dropna ()'.

Program

```
dataset. dropna (inplace = True)
print(dataset)
```

Output

| | Country | Age | Salary | Purchased |
|---|---------|------|---------|-----------|
| 0 | France | 44.0 | 72000.0 | No |
| 1 | Spain | 27.0 | 48000.0 | Yes |
| 2 | Germany | 30.0 | 54000.0 | No |
| 3 | Spain | 38.0 | 61000.0 | No |
| 5 | France | 35.0 | 58000.0 | Yes |
| 7 | France | 48.0 | 79000.0 | Yes |
| 8 | Germany | 50.0 | 83000.0 | No |
| 9 | France | 37.0 | 67000.0 | Yes |

10. Replace missing values

Use 'SimpleImputer' from scikit-learn to replace missing values with the mean of the respective columns.

Program

Taking care of missing data (replacing with the mean)

```
from sklearn. impute import SimpleImputer
imputer = SimpleImputer (missing_values = np.nan, strategy = 'mean')
```

Fitting the imputer object to the matrix of features X

imputer.fit(x[:,1:3])

Replacing the missing data by the mean of the column

```
x[:,1:3] = imputer.transform(x[:,1:3])
print(x[:,1:3])
```

```
[[44.0 72000.0]

[27.0 48000.0]

[30.0 54000.0]

[38.0 61000.0]

[40.0 63777.77777777778]

[35.0 58000.0]

[38.77777777777778 52000.0]

[48.0 79000.0]

[50.0 83000.0]

[37.0 67000.0]]
```

11. Encoding the categorical data

• Apply one-hot encoding to the 'Country' column using 'ColumnTransfomer' and 'OneHotEncoder'.

Program

```
from sklearn. compose import ColumnTransformer
from sklearn. preprocessing import OneHotEncoder
ct = ColumnTransformer (transformers = [('encoder', OneHotEncoder (), [0])], remainder =
"passthrough")
x = np. array(ct.fit_transform(x))
print(x)
```

Output

```
[[1.0 0.0 0.0 44.0 72000.0]

[0.0 0.0 1.0 27.0 48000.0]

[0.0 1.0 0.0 30.0 54000.0]

[0.0 0.0 1.0 38.0 61000.0]

[0.0 1.0 0.0 40.0 63777.77777777778]

[1.0 0.0 0.0 35.0 58000.0]

[0.0 0.0 1.0 38.77777777777778 52000.0]

[1.0 0.0 0.0 48.0 79000.0]

[0.0 1.0 0.0 50.0 83000.0]

[1.0 0.0 0.0 37.0 67000.0]]
```

• Apply Label encode to the 'Purchased' column using 'LabelEncoder'.

Program

```
from sklearn. preprocessing import LabelEncoder le = LabelEncoder ()
y=le.fit_transform(y)
print(y)
```

```
[0 1 0 0 1 1 0 1 0 1]
```

12. Splitting the Dataset

Use 'train_test_split' from scikit-learn to split the dataset into training and testing sets.

Program

```
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 = 1) print(x_train)
```

Output

```
[[0.0 1.0 0.0 40.0 63777.77777777778]

[1.0 0.0 0.0 44.0 72000.0]

[0.0 0.0 1.0 38.0 61000.0]

[0.0 0.0 1.0 27.0 48000.0]

[1.0 0.0 0.0 48.0 79000.0]

[0.0 1.0 0.0 50.0 83000.0]

[1.0 0.0 0.0 35.0 58000.0]]
```

Program

print(x_test)

Output

```
[[0.0 1.0 0.0 30.0 54000.0]
[1.0 0.0 0.0 37.0 67000.0]
[0.0 0.0 1.0 38.777777777778 52000.0]]
```

Program

print(y_train)

Output

[1 0 0 1 1 0 1]

Program

print(y_test)

Output

 $[0\ 1\ 0]$

13. Feature Scaling

Standardize the numerical features using 'StandardScaler' from scikit-learn.

Program

```
from sklearn. preprocessing import StandardScaler
scaler = StandardScaler ()
x_train[:,2:]=scaler.fit_transform(x_train[:,2:])
x_test[:,2:]=scaler.fit_transform(x_test[:,2:])
print(x_train)
print(x_test)
```

```
[[0.0 1.0 -0.6324555320336758 -0.038910211282047996 -0.22960023388015188]
[1.0 0.0 -0.6324555320336758 0.5058327466666259 0.49120534884662787]
[0.0 0.0 1.5811388300841895 -0.3112816902563849 -0.4731156334500103]
[0.0 0.0 1.5811388300841895 -1.809324824615238 -1.6127677034369463]
[1.0 0.0 -0.6324555320336758 1.0505757046152997 1.1048641557626704]
[0.0 1.0 -0.6324555320336758 1.3229471835896367 1.455526331143266]
[1.0 0.0 -0.6324555320336758 -0.7198389087178904 -0.736112264985457]]
[[0.0 1.0 -0.7071067811865475 -1.3880272079128577 -0.5513801778287937]
[1.0 0.0 -0.7071067811865475 0.4594174561401711 1.40351317992784]
[0.0 0.0 1.4142135623730951 0.9286097517726866 -0.8521330020990451]]
```

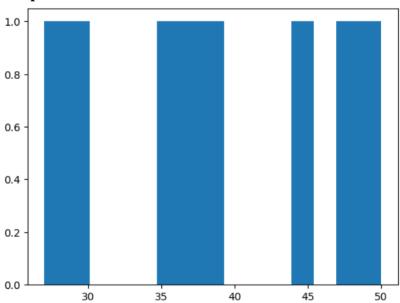
14. Data Visualization

Visualize the distribution of the 'Age' column using a histogram.

Program

import matplotlib. pyplot as plt
plt. hist(dataset['Age'], bins=15)
plt. show ()

Output



15. Identify outliers

Identify the outliers using the quantile method.

Program

lowerLimit=dataset['Age']. quantile (0.05) print(lowerLimit)

Output

28.05

Program

print(dataset[dataset['Age'] < lowerLimit])</pre>

Output

```
Country Age Salary Purchased
1 Spain 27.0 48000.0 Yes
```

Program

upperLimit=dataset['Age']. quantile (0.95) print(upperLimit)

Output

49.3

Program

print(dataset[dataset['Age']>upperLimit])

Output

```
Country Age Salary Purchased
8 Germany 50.0 83000.0 No
```

16.Removeoutliers from the dataset

Remove the outliers from the dataset using quantile method.

Program

```
dataset = dataset[(dataset['Age']>lowerLimit) & (dataset['Age'] <upperLimit)]
print(dataset)</pre>
```

Output

```
Country Age Salary Purchased
0 France 44.0 72000.0 No
2 Germany 30.0 54000.0 No
3 Spain 38.0 61000.0 No
5 France 35.0 58000.0 Yes
7 France 48.0 79000.0 Yes
9 France 37.0 67000.0 Yes
```

Result

Thus, the performance of the data exploration and preprocessing techniques in Python using a real-world dataset was successfully completed.

Ex. No: 9 IMPLEMENTATION OF REGULARIZED LINEAR REGRESSION

AIM

To write a Python program to implement a regularized linear regression on housing dataset.

PROCEDURE

- 1. Importing the necessary libraries, including 'pandas' for data manipulation, and 'numpy' for numerical operations and warnings.
- 2. Load the dataset containing housing dataset from the specified file location.
- 3. Explore the dataset to understand its structure and features. Preprocess the data if needed, such as handling missing values.
- 4. Replace missing values in specific numeric columns with the mean using 'SimpleImputer'.
- 5. Split the dataset into features (x) and target variable (y).
- 6. Further split the dataset into training and testing sets using 'train_test_split' from 'sklearn. model selection'.
- 7. Standardize the features using 'StandardScaler' from 'sklearn. preprocessing'.
- 8. Initialize a Ridge Regression model from 'sklearn. linear model'.
- 9. Train the Ridge Regression model using the training data with the 'fit' method.
- 10. Use the trained Ridge Regression model to make predictions on the testing data with the 'predict' method.
- 11. Evaluate the performance of the model using **metrics** such as mean_squared_error, or others based on the requirements.
- 12. Finally, use the trained model to predict the outcome for the new data.

PROGRAM

Import Libraries

import pandas as pd

import numpy as np

import warnings

warnings. Simplefilter ("ignore")

Load the Dataset

df = pd. read_csv("C:\\Users\\Documents\\Housing.csv")
df

| | CRIM | ZN | INDUS | CHAS | NOX | RM | AGE | DIS | RAD | TAX | PTRATIO | В | LSTAT | MEDV |
|-----|---------|------|-------|------|-------|-------|-------|--------|-----|------|---------|--------|-------|------|
| 0 | 0.00632 | 18.0 | 2.31 | 0.0 | 0.538 | 6.575 | 65.2 | 4.0900 | 1 | 296 | 15.3 | 396.90 | 4.98 | 24.0 |
| 1 | 0.02731 | 0.0 | 7.07 | 0.0 | 0.469 | 6.421 | 78.9 | 4.9671 | 2 | 242 | 17.8 | 396.90 | 9.14 | 21.6 |
| 2 | 0.02729 | 0.0 | 7.07 | 0.0 | 0.469 | 7.185 | 61.1 | 4.9671 | 2 | 242 | 17.8 | 392.83 | 4.03 | 34.7 |
| 3 | 0.03237 | 0.0 | 2.18 | 0.0 | 0.458 | 6.998 | 45.8 | 6.0622 | 3 | 222 | 18.7 | 394.63 | 2.94 | 33.4 |
| 4 | 0.06905 | 0.0 | 2.18 | 0.0 | 0.458 | 7.147 | 54.2 | 6.0622 | 3 | 222 | 18.7 | 396.90 | NaN | 36.2 |
| | 075 | | *** | (37) | *** | | 1.494 | 389 | 760 | .,,, | 1.424 | *** | *** | *** |
| 501 | 0.06263 | 0.0 | 11.93 | 0.0 | 0.573 | 6.593 | 69.1 | 2.4786 | 1 | 273 | 21.0 | 391.99 | NaN | 22.4 |
| 502 | 0.04527 | 0.0 | 11.93 | 0.0 | 0.573 | 6.120 | 76.7 | 2.2875 | 1 | 273 | 21.0 | 396.90 | 9.08 | 20.6 |
| 503 | 0.06076 | 0.0 | 11.93 | 0.0 | 0.573 | 6.976 | 91.0 | 2.1675 | 1 | 273 | 21.0 | 396.90 | 5.64 | 23.9 |
| 504 | 0.10959 | 0.0 | 11.93 | 0.0 | 0.573 | 6.794 | 89.3 | 2.3889 | 1 | 273 | 21.0 | 393.45 | 6.48 | 22.0 |
| 505 | 0.04741 | 0.0 | 11.93 | 0.0 | 0.573 | 6.030 | NaN | 2.5050 | 1 | 273 | 21.0 | 396.90 | 7.88 | 11.9 |

SRM

506 rows × 14 columns

df.info()

Output

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 14 columns):

| # | Column | Non-Null Count | Dtype |
|----|---------|----------------|---------|
| | | | |
| 0 | CRIM | 486 non-null | float64 |
| 1 | ZN | 486 non-null | float64 |
| 2 | INDUS | 486 non-null | float64 |
| 3 | CHAS | 486 non-null | float64 |
| 4 | NOX | 506 non-null | float64 |
| 5 | RM | 506 non-null | float64 |
| 6 | AGE | 486 non-null | float64 |
| 7 | DIS | 506 non-null | float64 |
| 8 | RAD | 506 non-null | int64 |
| 9 | TAX | 506 non-null | int64 |
| 10 | PTRATIO | 506 non-null | float64 |
| 11 | В | 506 non-null | float64 |
| 12 | LSTAT | 486 non-null | float64 |
| 13 | MEDV | 506 non-null | float64 |

dtypes: float64(12), int64(2)

memory usage: 55.5 KB

df. describe ()

Output

| | CRIM | ZN | INDUS | CHAS | NOX | RM | AGE | DIS | RAD | TAX | PTRATIO | В | L |
|-------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|--------|
| count | 486.000000 | 486.000000 | 486.000000 | 486.000000 | 506.000000 | 506.000000 | 486.000000 | 506.000000 | 506.000000 | 506.000000 | 506.000000 | 506.000000 | 486.00 |
| mean | 3.611874 | 11.211934 | 11.083992 | 0.069959 | 0.554695 | 6.284634 | 68.518519 | 3.795043 | 9.549407 | 408.237154 | 18.455534 | 356.674032 | 12.71 |
| std | 8.720192 | 23.388876 | 6.835896 | 0.255340 | 0.115878 | 0.702617 | 27.999513 | 2.105710 | 8.707259 | 168.537116 | 2.164946 | 91.294864 | 7.15 |
| min | 0.006320 | 0.000000 | 0.460000 | 0.000000 | 0.385000 | 3.561000 | 2.900000 | 1.129600 | 1.000000 | 187.000000 | 12.600000 | 0.320000 | 1.73 |
| 25% | 0.081900 | 0.000000 | 5.190000 | 0.000000 | 0.449000 | 5.885500 | 45.175000 | 2.100175 | 4.000000 | 279.000000 | 17.400000 | 375.377500 | 7.12 |
| 50% | 0.253715 | 0.000000 | 9.690000 | 0.000000 | 0.538000 | 6.208500 | 76.800000 | 3.207450 | 5.000000 | 330.000000 | 19.050000 | 391.440000 | 11.43 |
| 75% | 3.560263 | 12.500000 | 18.100000 | 0.000000 | 0.624000 | 6.623500 | 93.975000 | 5.188425 | 24.000000 | 666.000000 | 20.200000 | 396.225000 | 16.95 |
| max | 88.976200 | 100.000000 | 27.740000 | 1.000000 | 0.871000 | 8.780000 | 100.000000 | 12.126500 | 24.000000 | 711.000000 | 22.000000 | 396.900000 | 37.97 |
| 4 | | | | | | | | | | | | | b |

Explore and preprocess the data

df. isnull (). sum ()

Output

| CRIM | 20 |
|-----------|-----|
| ZN | 20 |
| INDUS | 20 |
| CHAS | 20 |
| NOX | 0 |
| RM | 0 |
| AGE | 20 |
| DIS | 0 |
| RAD | 0 |
| TAX | 0 |
| PTRATIO | 0 |
| В | 0 |
| LSTAT | 20 |
| MEDV | 0 |
| dtvne: in | t64 |



Replace the null values

 $from \ sklearn. \ impute \ import \ SimpleImputer \\ numeric_columns = ['CRIM', 'ZN', 'INDUS', 'CHAS', 'AGE', 'LSTAT'] \\ df[numeric_columns] = SimpleImputer(strategy='mean').fit_transform(df[numeric_columns]) \\ df[numeric_columns]$

| | CRIM | ZN | INDUS | CHAS | AGE | LSTAT |
|-----|---------|------|-------|------|-----------|-----------|
| 0 | 0.00632 | 18.0 | 2.31 | 0.0 | 65.200000 | 4.980000 |
| 1 | 0.02731 | 0.0 | 7.07 | 0.0 | 78.900000 | 9.140000 |
| 2 | 0.02729 | 0.0 | 7.07 | 0.0 | 61.100000 | 4.030000 |
| 3 | 0.03237 | 0.0 | 2.18 | 0.0 | 45.800000 | 2.940000 |
| 4 | 0.06905 | 0.0 | 2.18 | 0.0 | 54.200000 | 12.715432 |
| | | | | | | |
| 501 | 0.06263 | 0.0 | 11.93 | 0.0 | 69.100000 | 12.715432 |
| 502 | 0.04527 | 0.0 | 11.93 | 0.0 | 76.700000 | 9.080000 |
| 503 | 0.06076 | 0.0 | 11.93 | 0.0 | 91.000000 | 5.640000 |
| 504 | 0.10959 | 0.0 | 11.93 | 0.0 | 89.300000 | 6.480000 |
| 505 | 0.04741 | 0.0 | 11.93 | 0.0 | 68.518519 | 7.880000 |

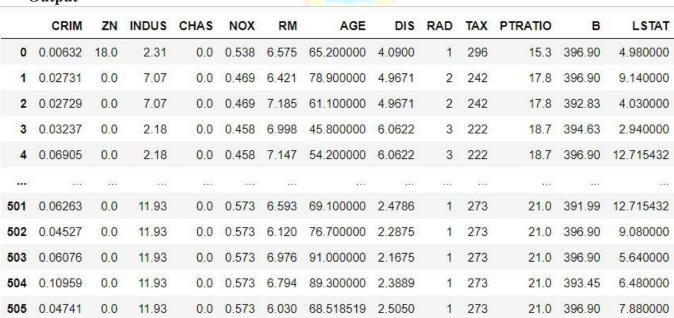
506 rows × 6 columns

Split the data into features (x)

x = df. iloc [:, :-1]

X

Output



506 rows × 13 columns

```
# Split the data into target variable (y)
y = df. iloc[:, -1]
y
    Output
0
        24.0
1
        21.6
2
        34.7
3
        33.4
        36.2
         . . .
501
        22.4
502
        20.6
503
        23.9
504
        22.0
505
        11.9
Name: MEDV, Length: 506, dtype: float64
# Further split the dataset into training and testing sets
from sklearn. model_selection import train_test_split
x_train, x_test, y_train, y_test=train_test_split (x, y, random_state=0)
    # Intialize the Regularized Linear Regression model
from sklearn. linear_model import Ridge
    # Train a Ridge Regression model
# Regularization strength, adjust as needed
alpha = 1.0
model = Ridge(alpha=alpha)
    # Train the Regularized Linear Regression model
model.fit (x_train, y_train)
    Output
Ridge ()
    # Make predictions
y_pred = model. predict(x_test)
```

y_pred

```
array([26.34156898, 22.20793785, 28.81921868, 11.47348568, 21.04408706,
       20.07024517, 19.92297674, 21.98467322, 19.4284628 , 19.78080258,
        4.16412665, 15.12289492, 17.09792351, 5.05900886, 38.91946091,
       32.92319545, 21.24037396, 36.10602752, 31.81994714, 23.86720827,
       25.23529707, 22.88616194, 20.79907381, 30.51097382, 22.69975385,
       7.89770805, 17.85693177, 18.79114443, 35.85644296, 20.64459013,
       17.22799563, 17.71776841, 19.12617187, 22.88297321, 28.53952139,
      19.82099868, 11.05567273, 23.90734107, 16.91704535, 14.40178292,
       25.90653761, 21.42101506, 23.7896341 , 13.97022376, 24.14528817,
       24.67733954, 19.26650784, 24.12468236, 11.09047876, 24.65867603,
      22.77081575, 19.26975343, 24.25327367, 31.6500543 , 12.84562266,
       22.28345771, 21.23812988, 16.08306186, 11.65101825, 22.23665653,
      18.60898999, 22.10455726, 32.35772086, 31.57758771, 17.12779565,
       32.87589474, 19.54848641, 19.28578883, 19.20291078, 24.09782582,
       21.94365685, 23.69076767, 30.63340076, 29.0446177, 24.44329987,
        5.35195943, 37.02566061, 24.09822246, 27.67967687, 19.90269748,
       28.43908868, 18.85735312, 17.29445352, 37.82901475, 39.31585098,
       24.74164818, 25.00022374, 16.02274095, 26.54988516, 16.73956103,
      16.46269817, 13.41520098, 24.63687876, 30.64557876, 22.71383635,
       20.562259 , 0.09343784, 25.64117227, 15.50351612, 17.61209673,
      25.92059772, 22.30531336, 32.48071611, 22.28998171, 27.57238428,
       23.45701196, 6.11176901, 14.10019865, 22.60348397, 29.02334184,
       31.92622327, 12.03657088, 19.53895825, 21.18421153, 12.1825899,
      23.82100592, 5.79235568, 19.29551109, 9.01036856, 45.15731178,
       30.5553796 , 17.34563703 , 17.515943 , 22.17927199 , 23.41151526 ,
       18.70788687, 34.97867185])
```

Evaluate the model

from sklearn. metrics import mean_squared_error mse = mean_squared_error (y_test, y_pred) print (f'Mean Squared Error: {mse}')

Output

Mean Squared Error: 32.12855445696262

Predict the new data

new_data = [0.02731, 0.0, 7.07, 0.0, 0.469, 6.421, 78.900000, 4.9671, 2, 242, 17.8, 396.90, 9.140000] predictions = model. predict([new_data]) print(predictions)

Output

[24.67733954]

RESULT

Thus, the implementation of regularized linear regression was completed successfully.

Ex. No: 10 MINI PROJECT TO PREDICT THE TIME TAKEN TO SOLVE A PROBLEM GIVEN THE CURRENT STATUS OF THE USER

AIM

The Mini Project to predict the time taken to solve a problem given the current status of the user using Random Forest Regressor Model.

PROCEDURE

- 1. Importing the necessary libraries, including 'pandas' for data manipulation, and 'numpy' for numerical operations and warnings.
- 2. Load the user data from the specified file location.
- 3. Explore the dataset to understand its structure and features. Preprocess the data if needed, such as handling missing values.
- 4. Replace missing values in specific numeric columns with the mean using 'SimpleImputer'.
- 5. Use 'LabelEncoder' to encode categorical columns like 'Country', and 'rank'.
- 6. Drop null values from the dataset, specifically the 'user id' column.
- 7. Split the dataset into features (x) and target variable (y).
- 8. Further split the dataset into training and testing sets using 'train_test_split' from 'sklearn. model selection'.
- 9. Initialize a Random Forest Regression model from 'sklearn. ensemble'.
- 10. Train the Random Forest Regression model using the training data with the 'fit' method.
- 11. Use the trained Random Forest Regression model to make predictions on the testing data with the 'predict' method.
- 12. Evaluate the performance of the model using **metrics** such as mean_squared_error, Score, or others based on the requirements.
- 13. Finally, use the trained model to predict the outcome for the new data.

PROGRAM

Import Libraries

import pandas as pd
import numpy as np

import warnings

warnings. Simplefilter ("ignore")

Load the Dataset

 $df = pd. read_csv("C:\Users\Documents\user_data.csv") \\ df$

| | user_id | submission_count | problem_solved | contribution | country | follower_count | last_online_time_seconds | max_rating | rating | rank | regis |
|------|-----------|------------------|----------------|--------------|----------------|----------------|--------------------------|------------|---------|--------------|-------|
| 0 | user_3311 | 47 | 40 | 0 | NaN | 4 | 1504111645 | 348.337 | 330.849 | intermediate | |
| 1 | user_3028 | 63 | 52 | 0 | India | 17 | 1498998165 | 405.677 | 339.450 | intermediate | |
| 2 | user_2268 | 226 | 203 | -8 | Egypt | 24 | 1505566052 | 307.339 | 284.404 | beginner | |
| 3 | user_480 | 611 | 490 | 1 | Ukraine | 94 | 1505257499 | 525.803 | 471.330 | advanced | |
| 4 | user_650 | 504 | 479 | 12 | Russia | 4 | 1496613433 | 548.739 | 486.525 | advanced | |
| | *** | | FAX: | (444.) | (4)4 | | | *** | 3404 | | |
| 3566 | user_2685 | 161 | 120 | 0 | Bangladesh | 42 | 1505409069 | 306.193 | 246.560 | beginner | |
| 3567 | user_1548 | 41 | 30 | 0 | NaN | 0 | 1504026868 | 331.135 | 218.463 | beginner | |
| 3568 | user_1929 | 58 | 51 | 0 | NaN | 0 | 1505552744 | 330.275 | 262.901 | beginner | |
| 3569 | user_2772 | 148 | 137 | 0 | NaN | 2 | 1496606504 | 409.977 | 345.757 | intermediate | |
| 3570 | user_2179 | 163 | 115 | 6 | South Korea | 40 | 1502074467 | 392.775 | 288.704 | beginner | |

3571 rows × 11 columns

df.info ()

Output

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3571 entries, 0 to 3570
Data columns (total 11 columns):

| # | Column | Non-Null Count | Dtype |
|-----|---------------------------|----------------|---------|
| | | | |
| 0 | user_id | 3571 non-null | object |
| 1 | submission_count | 3571 non-null | int64 |
| 2 | problem_solved | 3571 non-null | int64 |
| 3 | contribution | 3571 non-null | int64 |
| 4 | country | 2418 non-null | object |
| 5 | follower_count | 3571 non-null | int64 |
| 6 | last_online_time_seconds | 3571 non-null | int64 |
| 7 | max_rating | 3571 non-null | float64 |
| 8 | rating | 3571 non-null | float64 |
| 9 | rank | 3571 non-null | object |
| 10 | registration_time_seconds | 3571 non-null | int64 |
| da. | (1+c+/o) :-+c+/c) - | L-1+ (5) | |

dtypes: float64(2), int64(6), object(3)

memory usage: 307.0+ KB

df. describe ()

| | submission_count | problem_solved | contribution | follower_count | last_online_time_seconds | max_rating | rating | registration_time_seconds |
|-------|------------------|----------------|--------------|----------------|--------------------------|-------------|-------------|---------------------------|
| count | 3571.000000 | 3571.000000 | 3571.000000 | 3571.000000 | 3.571000e+03 | 3571.000000 | 3571.000000 | 3.571000e+03 |
| mean | 299.481098 | 267.894427 | 4.102492 | 46.690563 | 1.502680e+09 | 390.374392 | 350.165578 | 1.434961e+09 |
| std | 366.102887 | 344.139688 | 16.552256 | 211.494638 | 5.114850e+06 | 92.428788 | 106.592503 | 4.750758e+07 |
| min | 1.000000 | 0.000000 | -64.000000 | 0.000000 | 1.484237e+09 | 303.899000 | 0.000000 | 1.264761e+09 |
| 25% | 66.500000 | 53.000000 | 0.000000 | 4.000000 | 1.502691e+09 | 317.661000 | 279.243000 | 1.416323e+09 |
| 50% | 169.000000 | 146.000000 | 0.000000 | 13.000000 | 1.505054e+09 | 355.791000 | 329.702000 | 1.449085e+09 |
| 75% | 390.000000 | 349.000000 | 0.000000 | 40.000000 | 1.505551e+09 | 444.954000 | 413.417500 | 1.470379e+09 |
| max | 4570.000000 | 4476.000000 | 171.000000 | 10575.000000 | 1.505595e+09 | 983.085000 | 911.124000 | 1.484236e+09 |

Explore and preprocess the data

df. isnull (). sum ()

Output

| user_id | 0 |
|---------------------------|------|
| submission_count | 0 |
| problem_solved | 0 |
| contribution | 0 |
| country | 1153 |
| follower_count | 0 |
| last_online_time_seconds | 0 |
| max_rating | 0 |
| rating | 0 |
| rank | 0 |
| registration_time_seconds | 0 |
| dtype: int64 | |

Encoding the categorical values

from sklearn. preprocessing import LabelEncoder label_encoder = LabelEncoder () categorical_columns = ['country', 'rank'] df[categorical_columns] = df[categorical_columns].apply(label_encoder.fit_transform) df[categorical_columns]

Output

| | country | rank |
|------|---------|------|
| 0 | 79 | 3 |
| 1 | 31 | 3 |
| 2 | 21 | 1 |
| 3 | 73 | 0 |
| 4 | 57 | 0 |
| | | |
| 3566 | 5 | 1 |
| 3567 | 79 | 1 |
| 3568 | 79 | 1 |
| 3569 | 79 | 3 |
| 3570 | 62 | 1 |

3571 rows × 2 columns

Replace the null values

from sklearn. impute import SimpleImputer
numeric_columns = ['country']
df[numeric_columns] = SimpleImputer(strategy='mean'). fit_transform(df[numeric_columns])
df[numeric_columns]

Output country

| 0 | 79.0 |
|------|------|
| 1 | 31.0 |
| 2 | 21.0 |
| 3 | 73.0 |
| 4 | 57.0 |
| | |
| 3566 | 5.0 |
| 3567 | 79.0 |
| 3568 | 79.0 |
| 3569 | 79.0 |
| 3570 | 62.0 |

3571 rows × 1 columns



Drop the null values

df = df. drop ('user_id', axis =1)
df. head ()

| | submission_count | problem_solved | contribution | country | follower_count | last_online_time_seconds | max_rating | rating | rank | registration_time_seconds |
|---|------------------|----------------|--------------|---------|----------------|--------------------------|------------|---------|------|---------------------------|
| 0 | 47 | 40 | 0 | 79.0 | 4 | 1504111645 | 348.337 | 330.849 | 3 | 1466686436 |
| 1 | 63 | 52 | 0 | 31.0 | 17 | 1498998165 | 405.677 | 339.450 | 3 | 1441893325 |
| 2 | 226 | 203 | -8 | 21.0 | 24 | 1505566052 | 307.339 | 284.404 | 1 | 1454267603 |
| 3 | 611 | 490 | 1 | 73.0 | 94 | 1505257499 | 525.803 | 471.330 | 0 | 1350720417 |
| 4 | 504 | 479 | 12 | 57.0 | 4 | 1496613433 | 548.739 | 486.525 | 0 | 1395560498 |

Split the data into features (x)

x = df. iloc [:, :-1]

X

Output

| | submission_count | problem_solved | contribution | country | follower_count | last_online_time_seconds | max_rating | rating | rank |
|------|------------------|----------------|--------------|---------|----------------|--------------------------|------------|---------|------|
| 0 | 47 | 40 | 0 | 79.0 | 4 | 1504111645 | 348.337 | 330.849 | 3 |
| 1 | 63 | 52 | 0 | 31.0 | 17 | 1498998165 | 405.677 | 339.450 | 3 |
| 2 | 226 | 203 | -8 | 21.0 | 24 | 1505566052 | 307.339 | 284.404 | 1 |
| 3 | 611 | 490 | 1 | 73.0 | 94 | 1505257499 | 525.803 | 471.330 | 0 |
| 4 | 504 | 479 | 12 | 57.0 | 4 | 1496613433 | 548.739 | 486.525 | 0 |
| | | | | | | | | | |
| 3566 | 161 | 120 | 0 | 5.0 | 42 | 1505409069 | 306.193 | 246.560 | 1 |
| 3567 | 41 | 30 | 0 | 79.0 | 0 | 1504026868 | 331.135 | 218.463 | 1 |
| 3568 | 58 | 51 | 0 | 79.0 | 0 | 1505552744 | 330.275 | 262.901 | 1 |
| 3569 | 148 | 137 | 0 | 79.0 | 2 | 1496606504 | 409.977 | 345.757 | 3 |
| 3570 | 163 | 115 | 6 | 62.0 | 40 | 1502074467 | 392.775 | 288.704 | 1 |

3571 rows × 9 columns

Split the data into target variable (y)

y = df. iloc[:, -1] y

Output

| 0 | 1466686436 |
|--------------|--------------------------|
| 1 | 1441893325 |
| 2 | 1454267603 |
| 3 | 1350720417 |
| 4 | 1395560498 |
| | |
| | |
| 3566 | 1455055521 |
| 3566 3567 | 1455055521 1465142933 |
| 3300 | 1.33033321 |
| 3567 | 1465142933 |
| 3567 3568 | 1465142933 1480086231 |

Name: registration_time_seconds, Length: 3571, dtype: int64

Further split the dataset into training and testing sets

from sklearn. model_selection import train_test_split x_train, x_test, y_train, y_test=train_test_split (x, y, random_state=0)

Intialize the Random Forest Regression model

from sklearn. ensemble import RandomForestRegressor model = RandomForestRegressor (n_estimators=100, random_state=42)

Train the Random Forest Regression model

model.fit (x_train, y_train)

Output

RandomForestRegressor(random_state=42)

Make predictions

```
y_pred = model. predict(x_test)
y_pred
```

Output

```
array([1.45455496e+09, 1.43392280e+09, 1.39042627e+09, 1.45241261e+09,
       1.45483615e+09, 1.42245957e+09, 1.46665441e+09, 1.40576063e+09,
       1.45182356e+09, 1.40803296e+09, 1.44347665e+09, 1.45527652e+09,
       1.45833179e+09, 1.41039747e+09, 1.40353994e+09, 1.43335683e+09,
       1.45930505e+09, 1.48396266e+09, 1.45175827e+09, 1.46663339e+09,
       1.45883224e+09, 1.39686707e+09, 1.37510170e+09, 1.45583003e+09,
       1.43567271e+09, 1.46697796e+09, 1.42782285e+09, 1.35167334e+09,
       1.45055875e+09, 1.45104392e+09, 1.47095946e+09, 1.43119661e+09,
       1.39180426e+09, 1.45824164e+09, 1.44051569e+09, 1.46181146e+09,
       1.43774071e+09, 1.41057693e+09, 1.41982014e+09, 1.41032514e+09,
       1.46102952e+09, 1.44531362e+09, 1.44408272e+09, 1.43312909e+09,
       1.44923526e+09, 1.45480931e+09, 1.46493558e+09, 1.38739107e+09,
       1.46522130e+09, 1.34720321e+09, 1.44306016e+09, 1.43258882e+09,
       1.45973789e+09, 1.44872871e+09, 1.41464911e+09, 1.43173064e+09,
       1.47030693e+09, 1.41554286e+09, 1.45349140e+09, 1.45890839e+09,
       1.37963785e+09, 1.39024897e+09, 1.46494454e+09, 1.39503257e+09,
       1.36973396e+09, 1.46910568e+09, 1.45365958e+09, 1.44598027e+09,
       1.45992361e+09, 1.39585475e+09, 1.36215790e+09, 1.44300388e+09,
       1.42957886e+09, 1.45861093e+09, 1.39488787e+09, 1.46234938e+09,
```

Evaluate the model

from sklearn. metrics import mean_squared_error mse = mean_squared_error (y_test, y_pred) print (f'Mean Squared Error: {mse}')

Output

Mean Squared Error: 1269410250004051.2

Predict the new data

new_data = [161, 120, 0, 5.0, 42, 1505409069, 306.193, 246.560,1] predictions = model. predict([new_data]) print(predictions)

Output

[1.45716732e+09]



RESULT

Thus, the mini project to predict the time taken to solve a problem given the current status of the user was completed successfully using Random Forest Regression model.

Ex. No: 10 IMPLEMENTATION OF MULTIVARIATE REGRESSION

AIM

To create a machine learning model to predict the profit of a startup company using Multivariate Regression.

PROCEDURE

1. Multivariate linear regression resembles simple linear regression except that in multivariate linear regression, multiple independent variables contribute to the dependent variables and so multiple coefficients are used in the computation.

```
v = c + m_1x_1 + m_2x_2 + \dots + m_nx_n
```

- 2. Importing the necessary libraries, including 'pandas' for data manipulation, and 'numpy' for numerical operations and warnings.
- 3. Load the dataset containing startup company data. For Example, startup company dataset.
- 4. Explore the dataset to understand its structure and features. Preprocess the data if needed, such as handling missing values.
- 5. Split the dataset into features (x) and target variable (y).
- 6. Further split the dataset into training and testing sets using 'train_test_split' from 'sklearn. model selection'.
- 7. Initialize a Mutivariate Regression model from 'sklearn. linear model'.
- 8. Train the Mutivariate Regression model using the training data with the 'fit' method.
- 9. Use the trained Mutivariate Regression model to make predictions on the testing data with the 'predict' method.
- 10. Evaluate the performance of the model using metrics such as mean_absolute_error, mean_squared_error, root_mean_square, R_Squared, or others based on the requirements.
- 11. Finally, use the trained model to predict the outcome for the new data.

PROGRAM

Import Libraries

import pandas as pd
import numpy as np

import warnings

warnings. Simplefilter ("ignore")

Load the Dataset

df = pd. read_csv("C:\\Users\\Documents\\50_Startups.csv")
df. head ()

| | R&D Spend | Administration | Marketing Spend | State | Profit |
|---|-----------|----------------|-----------------|------------|-----------|
| 0 | 165349.20 | 136897.80 | 471784.10 | New York | 192261.83 |
| 1 | 162597.70 | 151377.59 | 443898.53 | California | 191792.06 |
| 2 | 153441.51 | 101145.55 | 407934.54 | Florida | 191050.39 |
| 3 | 144372.41 | 118671.85 | 383199.62 | New York | 182901.99 |
| 4 | 142107.34 | 91391.77 | 366168.42 | Florida | 166187.94 |

df.info()

Output

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50 entries, 0 to 49

Data columns (total 5 columns):

Column Non-Null Count Dtype

| # | COTUIIII | Non-Null Count | ucype |
|------|------------------|----------------|---------|
| | | | |
| 0 | R&D Spend | 50 non-null | float64 |
| 1 | Administration | 50 non-null | float64 |
| 2 | Marketing Spend | 50 non-null | float64 |
| 3 | State | 50 non-null | object |
| 4 | Profit | 50 non-null | float64 |
| d+vn | oc: float64/4) o | hioct(1) | |

dtypes: float64(4), object(1)

memory usage: 2.1+ KB

df. describe ()

| | R&D Spend | Administration | Marketing Spend | Profit |
|-------|---------------|----------------|-----------------|---------------|
| count | 50.000000 | 50.000000 | 50.000000 | 50.000000 |
| mean | 73721.615600 | 121344.639600 | 211025.097800 | 112012.639200 |
| std | 45902.256482 | 28017.802755 | 122290.310726 | 40306.180338 |
| min | 0.000000 | 51283.140000 | 0.000000 | 14681.400000 |
| 25% | 39936.370000 | 103730.875000 | 129300.132500 | 90138.902500 |
| 50% | 73051.080000 | 122699.795000 | 212716.240000 | 107978.190000 |
| 75% | 101602.800000 | 144842.180000 | 299469.085000 | 139765.977500 |
| max | 165349.200000 | 182645.560000 | 471784.100000 | 192261.830000 |

Explore and preprocess the data

df. isnull (). sum ()

Output

R&D Spend 0
Administration 0
Marketing Spend 0
State 0
Profit 0

dtype: int64

Encoding the categorical values

from sklearn import preprocessing

Using Label Encoder to convert the categorical values

le = preprocessing. LabelEncoder ()
state_encoded = le.fit_transform(data['State'])
data['State'] = state_encoded
data. head ()

Output

| | R&D Spend | Administration | Marketing Spend | State | Profit |
|---|-----------|----------------|-----------------|-------|-----------|
| 0 | 165349.20 | 136897.80 | 471784.10 | 2 | 192261.83 |
| 1 | 162597.70 | 151377.59 | 443898.53 | 0 | 191792.06 |
| 2 | 153441.51 | 101145.55 | 407934.54 | 1 | 191050.39 |
| 3 | 144372.41 | 118671.85 | 383199.62 | 2 | 182901.99 |
| 4 | 142107.34 | 91391.77 | 366168.42 | 1 | 166187.94 |

Split the data into features (x)

x = df. iloc [:, :-1]

X

| | R&D Spend | Administration | Marketing Spend | State |
|---|-----------|----------------|-----------------|-------|
| 0 | 165349.20 | 136897.80 | 471784.10 | 2 |
| 1 | 162597.70 | 151377.59 | 443898.53 | 0 |
| 2 | 153441.51 | 101145.55 | 407934.54 | 1 |
| 3 | 144372.41 | 118671.85 | 383199.62 | 2 |
| 4 | 142107.34 | 91391.77 | 366168.42 | 1 |

Split the data into target variable (y)

```
y = df. iloc[:, -1]
y
```

Output

- 0 192261.83 1 191792.06 2 191050.39 3 182901.99 4 166187.94
- Name: Profit, dtype: float64

Further split the dataset into training and testing sets

from sklearn. model_selection import train_test_split x_train, x_test, y_train, y_test=train_test_split (x, y, random_state=355)

Intialize the Multivariate Regression model

from sklearn. Linear_model import LinearRegression model = LinearRegression

Train the Random Forest Regression model

model.fit (x_train, y_train)

Output

LinearRegression ()

Make predictions

```
y_pred = model. predict(x_test)
y_pred
```

Output

```
array([126720.66150723, 84909.08961912, 98890.31854876, 46479.31240248, 129113.18318813, 50968.88397762, 109015.01626803, 100893.57078084, 97713.73821431, 113085.59056068])
```

Evaluate the model

```
from sklearn. metrics import mean_squared_error, mean_absolute_error mse = mean_squared_error (y_test, y_pred) print (f"Mean Squared Error: {mse}")
```

SHIM

Mean Squared Error: 80929465.49097784

mae = mean_absolute_error (y_test, y_pred)
print (f"Mean Absolute Error: {mae}")

Output

Mean Absolute Error: 6979.17574672139



RESULT

Thus, a machine learning model for profit prediction using Multivariate Regression model was implemented successfully.