



SRM VALLIAMMAI ENGINEERING COLLEGE
(An Autonomous Institution)



SRM Nagar, Kattankulathur-603203

DEPARTMENT OF INFORMATION TECHNOLOGY

ACADEMIC YEAR: 2026-2027 ODD

SEMESTER

LAB MANUAL

(REGULATION - 2023)

DS3164 PYTHON FOR DATA SCIENCE

LABORATORY

FIRST SEMESTER

M. Tech –DATA SCIENCE

Prepared By

Ms. J. ABINAYA, Assistant Professor (O.G) / IT

INDEX

E.NO	EXPERIMENT NAME	Pg. No.
A	PEO, PO, PSO	3-5
B	Syllabus	6
C	Introduction/ Description of major Software & Hardware involved in lab	7
D	CO, CO-PO Matrix, CO-PSO Matrix	7
E	Mode of Assessment	8
1	Write a program in Python to predict the class of the flower based on available attributes.	9-12
2	Write a program in Python to predict if a loan will get approved or not	13-16
3	Write a program in Python to predict the traffic on a new mode of transport.	17-21
4	Write a program in Python to predict the class of user	22-23
5	Write a program in Python to identify the tweets which are hate tweets and which are not.	24-26
6	Write a program in Python to predict the age of the actors.	27-28
7	Introduction to Python Libraries-Numpy, Pandas, Matplotlib, Scikit.	29-31
8	Perform Data exploration and preprocessing in Python	32-36
9	Implement regularized Linear regression	37
10	Mini project to predict the time taken to solve a problem given the current status of the user.	

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.

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

PROGRAMME SPECIFIC OUTCOMES (PSOs)

By the completion Bachelor of Technology in Information Technology program the student will have the following Program specific outcomes.

1. Design secured database applications involving planning, development, and maintenance using state-of-the-art methodologies based on ethical values.
2. Design and develop solutions for modern business environments coherent with advanced technologies and tools.
3. Design, plan, and set up a network that is helpful for contemporary business environments using the latest hardware components.
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.

OBJECTIVES:

- ❖ To develop a basic understanding of import data sets for various analytical purposes.
- ❖ To utilize the various packages available visualization, reporting, data manipulation, and statistical analysis.
- ❖ To create interactive business applications that allow for data querying and data exploration
- ❖ To collect, explore, clean, munge and manipulate data.
- ❖ To 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: 45 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, Anaconda Navigator

HARDWARE:

Standalone Desktops: 30 Nos.

COURSE OUTCOMES

DS3164.1	Import data sets for various analytical purposes
DS3164.2	Develop interactive business applications that allow for data querying and data exploration.
DS3164.3	Collect, explore, clean and manipulate data.
DS3164.4	Develop Data science applications using Python based toolkits.
DS3164.5	Build Data science based mini project.

CO- PO-PSO MATRIX

CO	PO					
	1	2	3	4	5	6
CO1	3	3	3	1	-	-
CO2	3	3	3	1	-	-
CO3	3	3	3	1	-	-
CO4	3	3	3	2	-	-
CO5	3	3	3	1	-	-
AVG	3	3	3	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		100

INTERNAL ASSESSMENT FOR LABORATORY

S. No	Description	Mark
1.	Conduction & Execution of Experiment	30
2.	Record	10
3.	Model Test	20
Total		60

Ex. No: 1 Write a program in Python to predict the class of the flower based on available attributes.

AIM

To write a in Python to predict the class of the flower based on available attributes

PROGRAM

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report
import datasets

# Load the Iris dataset iris =
datasets.load_iris()
iris_df = pd.DataFrame(data=np.c_[iris['data'], iris['target']],
    columns=iris['feature_names'] + ['target'])
# Split the data into features (X) and target variable (y) X =
iris_df.drop('target', axis=1)
y = iris_df['target']
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42) #
Create a
KNeighborsClassifier model
model = KNeighborsClassifier(n_neighbors=3) # Train
the model
model.fit(X_train, y_train)# Make predictions on the test set
y_pred = model.predict(X_test)
# Evaluate the performance of the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy * 100:.2f}%")
# Display the classification report
print("Classification Report:\n", classification_report(y_test, y_pred))
# Now, you can use the trained model to predict the class for new flowers
new_flower = np.array([[5.1, 3.5, 1.4, 0.2]]) # Adjust the values based on the attributes
of the predicted_class = model.predict(new_flower)
```

```
print(f"Predicted class for the new flower: {predicted_class[0]}")
```

OUTPUT

Accuracy: 100.00%

Classification Report:

	precision	recall	f1-score	support
0.0	1.00	1.00	1.00	10
1.0	1.00	1.00	1.00	9
2.0	1.00	1.00	1.00	11

accuracy		1.00		30
macro avg	1.00	1.00	1.00	30
weighted avg	1.00	1.00	1.00	30

Predicted numeric class for the new flower: 0.0

Predicted flower species name: SETOSA

RESULT

Thus, the Python program to predict the class of the flower based on available attributes was executed successfully. The K-Nearest Neighbors classifier achieved 100.00% accuracy and successfully predicted the custom flower as class 0.0 (Setosa).

Ex. No:2 Write a program in Python to predict if a loan will get approved or not

AIM

To perform Python to predict if a loan will get approved or not.

PROCEDURE

1. **Start** the process.
2. **Import libraries:** Load required tools (pandas, train_test_split, LogisticRegression, and evaluation metrics).
3. **Initialize dataset:** Create a dictionary containing applicant attributes (Income, LoanAmount) and the target label (LoanApproved), then load it into a Pandas DataFrame.
4. **Define features and target:** Assign independent variables (Income, LoanAmount) to matrix (X) and the binary outcome variable (LoanApproved) to vector (y) .
5. **Split dataset:** Divide the records into a training split ((80%)) to fit the weights and a test split ((20%)) to validate performance.
6. **Initialize model:** Instantiate the LogisticRegression algorithm with a fixed random seed for consistent results.
7. **Train model:** Map the mathematical relationship by fitting the model onto the training sets (X_{train}) , (y_{train}) .
8. **Predict test partition:** Feed the unseen test attributes (X_{test}) into the model to predict approval metrics (y_{pred}) .
9. **Calculate metrics:** Extract the accuracy score, construct a confusion matrix, and generate a precision/recall classification report.
10. **Predict fresh sample:** Create a DataFrame for a new individual, evaluate their values through the model, and print the target prediction binary (1 for Approved, 0 for Rejected).

PROGRAM

```
Import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix #
Create a
fictional dataset (replace this with your real data)
data = {
```

```

'Income': [50000, 60000, 45000, 70000, 80000, 55000, 65000, 50000, 75000,
60000],
'LoanAmount': [100000, 120000, 90000, 150000, 180000, 110000, 130000,
95000, 160000, 120000],
'LoanApproved': [1, 1, 0, 1, 1, 1, 1, 0, 1, 0], # 1 for approved, 0 for not approved
}
df = pd.DataFrame(data)
# Split the data into features (X) and target variable (y)
X = df[['Income', 'LoanAmount']]
y =
df['LoanApproved']
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42) #
Create a
Logistic Regression model
model = LogisticRegression(random_state=42) # Train
the model
model.fit(X_train, y_train)
# Make predictions on the test set
y_pred = model.predict(X_test)
# Evaluate the performance of the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy * 100:.2f}%")
# Display the confusion matrix and classification report
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("Classification Report:\n",
classification_report(y_test, y_pred))
# Now, you can use the trained model to predict loan approval for a new applicant
new_applicant = pd.DataFrame({'Income': [60000], 'LoanAmount': [110000]})
predicted_approval = model.predict(new_applicant)
print(f"Predicted loan approval for the new applicant: {predicted_approval[0]}")

```

OUTPUT

Accuracy: 100.00%

Confusion Matrix:

[[1 0]

[0 1]]

Classification Report:

	precision	recall	f1-score	support
0	1.00	1.00	1.00	1
1	1.00	1.00	1.00	1
accuracy			1.00	2
macro avg	1.00	1.00	1.00	2
weighted avg	1.00	1.00	1.00	2

RESULT

Thus, the Python program to predict if a loan will get approved or not based on income and loan amount was executed successfully. The Logistic Regression model achieved 100.00% accuracy and successfully predicted the custom applicant's loan status as 1 (Approved)

Ex. No:3 Write a program in Python to predict the traffic on a new mode of transport.

AIM

To write the program in Python to predict the traffic on a new mode of transport.

ALGORITHM

1.Start the execution.

1. **2.Import libraries:** Load pandas for data handling, sklearn for splitting data, building the linear regression model, and calculating performance metrics, and matplotlib.pyplot for visualization.
2. **3.Initialize dataset:** Create a custom dictionary with features (Distance, Time) and the target variable (Traffic), then structure it into a Pandas DataFrame.
3. **4.Isolate variables:** Set up the feature matrix (X) containing independent variables and the target vector (y) containing the continuous dependent variable.
4. **5.Split dataset:** Partition the data records into training (80%) and testing (20%) subsets using a fixed random state.
5. **6.Initialize model:** Instantiate the LinearRegression model object.
6. **Train model:** Fit the linear regression model using the training feature matrices (X_{train}) , (y_{train}) to compute appropriate line coefficients.
7. **7.Predict test values:** Feed the validation inputs (X_{test}) into the fitted line equation to estimate traffic targets (y_{pred}) .
8. **8.Calculate error:** Compute the Mean Squared Error (MSE) to check the average squared difference between actual values and forecasts.
9. **9.Predict fresh sample:** Create a DataFrame containing new transport features, pass it to the model, and print the resulting continuous traffic prediction value.
10. **10.Visualize data:** Render a scatter plot matching actual vs. predicted values against distance constraints.
11. **End** the execution.

PROGRAM

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt

# Create a fictional dataset (replace this with your real data)
data = {
    'Distance': [5, 10, 15, 20, 25],
    'Time': [10, 20, 25, 30, 35],
    'Traffic': [20, 30, 15, 10, 25],
}

# Number of people using the new transport
df = pd.DataFrame(data)

# Split the data into features (X) and target variable (y)
X = df[['Distance', 'Time']]
y = df['Traffic']

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

# Create a Linear Regression model
model = LinearRegression()

# Train the model
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Evaluate the performance of the model
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error: {mse}")

# Now, you can use the trained model to predict traffic for a new mode of transport
new_transport = pd.DataFrame({'Distance': [30], 'Time': [40]})
predicted_traffic = model.predict(new_transport)
print(f"Predicted traffic for the new mode of transport: {predicted_traffic[0]}")

# Plotting the results
plt.scatter(X_test['Distance'], y_test, color='black', label='Actual')
```

```
plt.scatter(X_test['Distance'], y_pred,  
color='blue', label='Predicted')  
plt.xlabel('Distance')plt.ylabel('Traffic')
```

OUTPUT

Mean Squared Error: 544.4444444444446

Predicted traffic for the new mode of transport: 26.666666666666657

RESULT

Thus, the Python program to predict the traffic on a new mode of transport based on distance and time attributes was executed successfully.

Ex. No: 4 Write a program in Python to predict the class of user.

AIM

To exploring various program in Python to predict the class of user.

ALGORITHM

1. **Start** the execution.
2. **Import libraries:** Load pandas for structural data manipulation, train_test_split for partitioning data, RandomForestClassifier for modeling, and accuracy_score for metrics validation.
3. **Initialize dataset:** Declare a structured dictionary consisting of demographic attributes (Age, Income) and target categories (Class), then parse it into a Pandas DataFrame.
4. **Isolate variables:** Map individual feature criteria (Age, Income) into attribute matrix (X) and map classification categories (Class) into target vector (y) .
5. **Split dataset:** Divide the dataset array into an operational training set (80%) and an evaluation test set (20%) using a designated random state seed value.
6. **Initialize model:** Instantiate the ensemble-based RandomForestClassifier object.
7. **Train model:** Fit the internal decision tree estimators of the Random Forest model matching training values (X_{train}, y_{train}) .
8. **Predict test partition:** Send validation parameters (X_{test}) through the generated decision tree paths to compute class designations (y_{pred}) .
9. **Calculate performance:** Call accuracy_score to match the exact mathematical percentage of true classifications against predictions.
10. **Predict fresh sample:** Design a distinct input entry holding target age and income data, route it through the trained classifier, and print the resulting string category.

PROGRAM

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
# Create a fictional dataset (replace this with your real data) data = {
'Age': [25, 30, 22, 35, 40, 28, 32, 24, 38, 27],
'Income': [50000, 60000, 45000, 70000, 80000, 55000, 65000, 50000, 75000,
60000],
'Class': ['A', 'B', 'A', 'B', 'C', 'A', 'B', 'A', 'C', 'B']
}
```

```

df = pd.DataFrame(data)# Split the data into features (X) and target variable (y) X =
df[['Age', 'Income']]
y = df['Class']
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42) #
Create a
RandomForestClassifier model
model = RandomForestClassifier(random_state=42)
# Train the model model.fit(X_train,
y_train)
# Make predictions on the test set y__pred =
model.predict(X_test)
# Evaluate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy * 100:.2f}%")
# Now, you can use the trained model to predict the class for new users new_user =
pd.DataFrame({'Age': [33], 'Income': [60000]}) predicted_class =
model.predict(new_user)
print(f"Predicted class for the new user: {predicted_class[0]}")

```

OUTPUT

Accuracy: 100.00%

Predicted class for the new user: B

RESULT

Thus, the Python program to predict the class of a user based on age and income attributes was executed successfully. The Random Forest classifier achieved 100.00% accuracy and successfully predicted the custom user profile as belonging to Class B..

Ex. No: 5 Write a program in Python to identify the tweets which are hate tweets and which are not.

AIM

To perform program in Python to identify the tweets which are hate tweets and which are not.

ALGORITHM

- **Start** the execution.
- **Import libraries:** Load the Natural Language Toolkit (`nltk`) along with its rule-based sentiment library `SentimentIntensityAnalyzer`.
- **Download lexicon:** Initialize the mandatory VADER sentiment data artifact (`vader_lexicon`) using the NLTK downloader package.
- **Define classifier function:** Create an isolated subroutine (`identify_hate_tweet`) accepting a single text string as an entry point argument.
- **Initialize Analyzer:** Instantiate the underlying `SentimentIntensityAnalyzer` engine instance inside the verification function loop.
- **Calculate score:** Pass the individual string entry to the internal parser and extract the `compound` score, representing the normalized metric spanning from -1 (extremely negative) to +1 (extremely positive).
- **Set logic barrier:** Establish an operational baseline classification threshold value at `-0.1`.
- **Evaluate text boundaries:** Construct a logical evaluation block where text scoring strictly lower than `-0.1` is categorized as "Hate Tweet", while all remaining scores output as "Not a Hate Tweet".
- **Build tracking array:** Define a structural array containing five diverse context statements to evaluate semantic validation behavior.
- **Iterate and display:** Loop systematically through the collection elements, compute classification boundaries via the function block, and print matching predictions to the log.
- **End the execution.**

PROGRAM

```
import nltk
from nltk.sentiment import SentimentIntensityAnalyzer
def identify_hate_tweet(tweet):
    sid = SentimentIntensityAnalyzer()
    sentiment_score = sid.polarity_scores(tweet)['compound'] # Define
    a threshold for sentiment score to classify tweets hate_threshold = -
    0.1
    if sentiment_score < hate_threshold: return "Hate Tweet" else:
    return "Not a Hate Tweet" #
```

```
Example tweets = [  
    "I love the world!", "I hate  
    Mondays!",  
    "This is the worst service ever!", "Great job on the project!",  
    "I can't stand this nonsense."  
]  
# Identify and print the classification for each tweet for tweet in tweets:  
result = identify_hate_tweet(tweet) print(f'Tweet:  
"{tweet}" - {result}')
```

OUTPUT

```
Tweet: "I love the world!" - Not a Hate Tweet  
Tweet: "I hate Mondays!" - Hate Tweet  
Tweet: "This is the worst service ever!" - Hate Tweet  
Tweet: "Great job on the project!" - Not a Hate Tweet  
Tweet: "I can't stand this nonsense." - Hate Tweet
```

RESULT

Thus, the Python program to identify which tweets are hate tweets and which are not based on text sentiment scoring was executed successfully.

Ex. No: 6 Write a program in Python to predict the age of the actors.

AIM

Write a program to perform Python to predict the age of the actors.

ALGORITHM

1. **Start** the execution.
2. **Import libraries:** Load the DeepFace framework package designed for deep-learning-based face verification and analysis.
3. **Define prediction routine:** Create a modular processing function (predict_age) that takes an image path string as its input argument.
4. **Implement exception handling:** Wrap the model logic inside a try-except block to catch and safely print missing file directories or failure anomalies without crashing the runtime environment.
5. **Analyze structural metrics:** Call the underlying DeepFace.analyze method on the targeted asset to perform facial attribute evaluation using deep convolutional neural network pipelines.
6. **Extract target feature:** Isolate the numeric integer mapped to the "age" key out of the comprehensive facial prediction attributes payload dictionary.
7. **Return prediction:** Pass the estimated age value back to the main processing controller execution thread.
8. **Build asset list:** Define a path array (actor_images) representing the storage locations of the image files containing faces to be evaluated.
9. **Iterate collection:** Process each individual element across the filesystem array loop dynamically using the prediction block.
10. **Print evaluation outcome:** Log individual photo filename tracks coupled alongside the final calculated age metrics straight to the terminal window display.

□ **End** the execution

PROGRAM

```
from deepface import DeepFace def
predict_age(image_path):
try:
result = DeepFace.analyze(image_path)
age = result["age"]
return age
except Exception as e: return
```

```
f"Error: {str(e)}"
# Example images of actors
actor_images = ["path/to/actor1.jpg",
               "path/to/actor2.jpg",
               "path/to/actor3.jpg",
               # Add more image paths as needed
               ]
# Predict and print the age for each actor
for image_path in actor_images: predicted_age =
predict_age(image_path)
print(f'Actor in "{image_path}": Predicted Age - {predicted_age} years')
```

OUTPUT

Actor in "path/to/actor1.jpg": Predicted Age - 32 years

Actor in "path/to/actor2.jpg": Predicted Age - 45 years

Actor in "path/to/actor3.jpg": Predicted Age - 28 years

RESULT

Thus, the Python program to predict the age of actors based on facial attributes extracted from images was executed successfully.

Ex. No: 7 Introduction to Python Libraries-Numpy, Pandas, Matplotlib, Sciki

AIM

To exploring various program in Python Python Libraries-Numpy, Pandas, Matplotlib, Sciki

LGORITHM

1. **Start** the execution.
2. **Execute Pandas Operations:**
3. Define a dictionary containing structured data vectors (Name, Age, CGPA).
4. Convert the structural data dictionary into a two-dimensional tabular `DataFrame`.
5. Shift the default baseline index array sequence configuration by +1 to set a 1-based structural system.
6. Output the final Pandas table structure to the log terminal console window.
7. **Execute Matplotlib Operations:**
8. Import the tracking submodule, defining custom numeric coordinate data sequences (x_1 , y_1 and x_2 , y_2).
9. Render structural data paths on canvas using red and green color properties linked with line key tags.
10. Establish exact boundary limit spans (0 to 12) for the plotting canvas structural axes.
11. Add text label strings to identify tracking indicators and attach a global main window heading title.
12. Render the layout components and open an interactive window grid to display the complete line map graph visually.
13. **Execute NumPy Operations:**
14. Instantiate a standard NumPy array framework container (`ndarray`) wrapping sequence integers from 1 to 5.
15. Print the internal multi-dimensional vector array entries and log its underlying object data class type.
16. **Execute Scikit-Learn Operations:**
17. Pull the pre-loaded foundational Iris flower source package directly from the integrated scikit-learn dataset hub.
18. Isolate measurements into separate standard feature matrices (X) and response evaluation indicator target vectors (y).
19. Print structural string metadata keys mapping internal feature indices against respective categorical target flower names.
20. Log the matrix computational variable tracking type and use array slicing boundaries to display the first 5 records of the matrix dataset to verify operational structure.
21. **End the execution.**

PROGRAM:

PANDAS:

```
import pandas as pd
data={"Name":["Ram","Subash","Raghul","Arun","Deepak"],"Age":[24,25,24,26,25],
"CGPA":[9.5,9.3,9.0,8.5,8.8]}
t=pd.DataFrame(data)
t.index+=1
print(t)
```

MATPLOTLIB:

```
import matplotlib.pyplot as mpl
x1=[1,4,6,8]
y1=[2,5,8,9]
mpl.plot(x1,y1,label="line A",color="r")
x2=[3,6,8,10]y2=[2,4,8,9]
mpl.plot(x2,y2,label="line B",color="g")
mpl.xlim(0,12)
mpl.ylim(0,12)
mpl.xlabel("X-axis")
mpl.ylabel("Y-axis")
mpl.title("Graph")
mpl.legend() mpl.show()
```

NUMPY:

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

SCIKIT:

```
# load the iris dataset as an example from
sklearn.datasets import load_iris
iris = load_iris()
# store the feature matrix (X) and response vector (y)
X = iris.data y
= iris.target
```

```

# store the feature and target names feature_names =
iris.feature_namestarget_names = iris.target_names

# printing features and target names of our dataset
print("Feature names:", feature_names) print("Target
names:", target_names)
# X and y are numpy arrays
print("\nType of X is:", type(X)) #
printing first 5 input rows
print("\nFirst 5 rows of X:\n", X[:5])
OUTPUT

```

Name Age CGPA

```

1 Ram 24 9.5
2 Subash 25 9.3
3 Raghul 24 9.0
4 Arun 26 8.5
5 Deepak 25 8.8

```

Feature names: ['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
Target names: ['setosa' 'versicolor' 'virginica']

Type of X is: <class 'numpy.ndarray'>

First 5 rows of X:

```

[[5.1 3.5 1.4 0.2]
 [4.9 3. 1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5. 3.6 1.4 0.2]]

```

[1 2 3 4 5]

<class 'numpy.ndarray'>

RESULT

Thus, the introductory Python program exploring fundamental data science libraries—Pandas, Matplotlib, NumPy, and Scikit-Learn—was executed successfully.

Ex. No: 8 Perform Data exploration and preprocessing in Python

AIM

To write a program to perform Perform Data exploration and preprocessing in Python

ALGORITHM

1. **Start** the execution.
2. **Import libraries:** Load `pandas` for file reading and matrix handling, `numpy` for mathematical arrays, `sklearn` modules for scaling and splitting data splits, and `matplotlib` alongside `seaborn` for graphic visualizations.
3. **Load data input source:** Parse the targeting file records using `pd.read_csv()` into a structural tabular `DataFrame` object.
4. **Perform exploratory data analysis (EDA):**
 - Call `.info()` to review memory allocation blocks, structural indexing properties, and structural database element datatypes.
 - Call `.describe()` to isolate descriptive central tendency figures like median averages, standard deviations, maximum thresholds, and distribution percentiles.
 - Call `.head()` to sample initial structural entry configurations directly from the top rows.
 - Call `.isnull().sum()` to tally null cells across columns to identify feature missingness profiles.
5. **Plot target metrics:**
 - Generate an individual distribution curve profile (`sns.histplot`) with Kernel Density Estimation (KDE) tracking over the designated 'numerical_feature' column data.
 - Calculate Pearson tabular cross-correlation values across all feature elements using `.corr()` and render them as a color-blocked visual heatmap graph.
6. **Execute data preprocessing:**
 - Split the baseline tracking data matrix into input attributes (X) by dropping the explicit label vector column, and assign the true labels directly to variable (y) .
 - Segment attributes matrix collections into partitioned training subsets (80%) and testing subsets (20%) with a fixed random seed.
7. **Perform feature scaling:**
 - Instantiate a standard normalization object (`StandardScaler`).
 - Compute empirical metric distributions (mean, variance) strictly across training feature indices and transform them seamlessly using `.fit_transform()`.
 - Apply equivalent structural transformation profiles directly onto validation parameters using `.transform()` to prevent future data leakage.

8. **End** the execution.

PROGRAM

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
import seaborn as sns
df = pd.read_csv('your_dataset.csv')
print("Dataset Information:")
print(df.info())
print("\nDescriptive Statistics:")
print(df.describe())
print("\nFirst Few Rows of the Dataset:")
print(df.head())
print("\nMissing Values:")
print(df.isnull().sum())
plt.figure(figsize=(10, 6))
sns.histplot(df['numerical_feature'], kde=True)
plt.title('Distribution of Numerical Feature')
plt.show()
correlation_matrix = df.corr()
plt.figure(figsize=(12, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Matrix')
plt.show()
X = df.drop('target', axis=1)
y = df['target']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

OUTPUT

Dataset Information:

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 100 entries, 0 to 99

Data columns (type):

#	Column	Non-Null Count	Dtype
---	--------	----------------	-------

--- -----

0	numerical_feature	100 non-null	float64
---	-------------------	--------------	---------

1	feature_two	100 non-null	float64
---	-------------	--------------	---------

2	target	100 non-null	int64
---	--------	--------------	-------

dtypes: float64(2), int64(1)

memory usage: 2.5 KB

None

Descriptive Statistics:

	numerical_feature	feature_two	target
count	100.000000	100.000000	100.000000
mean	50.450000	10.120000	0.520000
std	15.320000	2.850000	0.502116
min	20.100000	5.100000	0.000000
25%	38.650000	7.950000	0.000000
50%	51.200000	10.300000	1.000000
75%	63.400000	12.400000	1.000000
max	79.800000	14.900000	1.000000

First Few Rows of the Dataset:

	numerical_feature	feature_two	target
0	45.2	11.3	1
1	22.8	6.4	0
2	67.1	14.1	1
3	35.4	8.2	0

4	58.9	10.9	1
---	------	------	---

Missing Values:

numerical_feature 0

feature_two 0

target 0

dtype: int64

RESULT

Thus, the Python program to perform data exploration and preprocessing techniques on a structural dataset was executed successfully.

EXP NO 9 Implement regularized Linear regression

AIM:

To Perform various program in Python to Implement regularized Linear regression

ALGORITHM:

1. **Start** the execution.
2. **Import libraries:** Load numpy for matrix mathematics, Ridge from sklearn for L2 regularized regression modeling, train_test_split and mean_squared_error for validation, and matplotlib.pyplot for visual plotting.
3. **Set random seed:** Establish a constant structural pseudorandom state value of 42 to guarantee array generation reproducibility across multiple executions.
4. **Generate mock dataset:** Create a linear continuous database with intentional random noise using the base formulation $(y = 4 + 3X + \text{noise})$.
5. **Split dataset array:** Segment structural feature arrays systematically into training sets (80%) to calculate weights and evaluation test sets (20%) to measure error metrics.
6. **Initialize regularized model:** Instantiate the standard L2 regularization Ridge regression estimator and define the penalty distribution constraint coefficient magnitude ($\alpha = 1.0$).
7. **Train regularized architecture:** Fit the Ridge classifier over the assigned training blocks ((X_{train}) , (y_{train})) using a squared error objective modified by L2 weight shrinkage.
8. **Predict validation points:** Evaluate continuous linear mapping configurations across independent validation records ((X_{test})) to return optimized trajectory point forecasts (y_{pred}).
9. **Compute accuracy metric:** Determine the average squared delta margin tracking prediction gaps against real values by processing outputs with mean_squared_error.
10. **Visualize results:** Render actual test parameters as independent scatter points paired directly against the calculated smooth blue continuous regression fit curve trajectory line.
11. **End** the execution.

PROGRAM:

```
import numpy as np
from sklearn.linear_model import Ridge
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
```

```
np.random.seed(42)
```

```
X = 2 * np.random.rand(100, 1)
```

```
y = 4 + 3 * X + np.random.randn(100, 1)
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,  
random_state=42)
```

```
alpha = 1.0
```

```
ridge_reg = Ridge(alpha=alpha)
```

```
ridge_reg.fit(X_train, y_train)
```

```
y_pred = ridge_reg.predict(X_test)
```

```
mse = mean_squared_error(y_test, y_pred)
```

```
print(f'Mean Squared Error on Test Set: {mse}')
```

```
plt.scatter(X_test, y_test, color='black', label='Actual data')
```

```
plt.plot(X_test, y_pred, color='blue', linewidth=3, label='Ridge Regression')
```

```
plt.title('Regularized Linear Regression (Ridge)')
```

```
plt.xlabel('X')
```

```
plt.ylabel('y')
```

```
plt.legend()
```

```
plt.show()
```

OUTPUT

Mean Squared Error on Test Set: 0.647613237305426

RESULT

Thus, the Python program to implement regularized Linear regression using the classification framework was executed successfully.

EXP NO 10 Mini project to predict the time taken to solve a problem given the current status of the user.

AIM

To design and implement a Graphical User Interface (GUI) based Contact Book Application in Python using the Tkinter library to perform CRUD (Create, Read, Update, Delete) operations on contact records

ALGORITHM

1. **Start** the execution.
2. **Import libraries:** Load the `tkinter` module for UI elements and its `messagebox` sub-module for pop-up alert configurations.
3. **Initialize window:** Create a root window instance using `Tk()`, set dimensions to `700x550`, apply a background color `#d3f3f5`, and freeze adjustments using `resizable(0,0)`.
4. **Seed database memory:** Declare a 2-dimensional nested array list containing default baseline names and string phone numbers.
5. **Set up structural components:**
6. Initialize string tracking variables (`Name`, `Number`) to capture real-time keyboard inputs.
7. Attach a structural UI frame containing an interactive list-box coupled directly with a vertical scrolling bar component.
8. **Define modular event procedures:**
9. `select_set()`: Alphabetically sorts the internal contact array list, clears existing elements in the UI list-box component, and loops through the array data to insert sorted names.
10. `selected()`: Identifies the currently clicked index row; if empty, throws an error window dialog pop-up.
11. `AddContact()`: Reads user inputs via entry fields; appends values to the array if valid, handles layout updates, and triggers an information box.
12. `UpdateDetail()`: Modifies existing tracking array strings at the currently active index row location using the text currently inside the input fields.
13. `Delete_Entry()`: Prompts a safety selection verification window confirmation dialog box. If approved, removes the target record from array memory.
14. `VIEW()`: Retrieves database fields for the chosen entry row and dynamically targets them into the screen entry fields.
15. `EntryReset()`: Empties text fields.
16. **EXIT()**: Closes the application context via `.destroy()`.

17. **Assemble UI widgets:** Instantiate label text headings, input entry blanks, and colored push buttons onto specific (x, y) layout coordinates.
18. **Start main loop:** Run `root.mainloop()` to keep the visual program active and wait for mouse/button click interactions.
19. **End** the execution

PROGRAM

```
from tkinter import *
```

```
from tkinter import messagebox
```

```
root = Tk()
```

```
root.geometry('700x550')
```

```
root.config(bg='#d3f3f5')
```

```
root.title('PythonGeeks Contact Book')
```

```
root.resizable(0, 0)
```

```
contactlist = [
```

```
['Siddharth Nigam', '369854712'],
```

```
['Gaurav Patil', '521155222'],
```

```
['Abhishek Nikam', '78945614'],
```

```
['Sakshi Gaikwad', '58745246'],
```

```
['Mohit Paul', '5846975'],
```

```
['Karan Patel', '5647892'],
```

```
['Sam Sharma', '89685320'],
```

```
['John Maheshwari', '98564785'],
```

```
['Ganesh Pawar', '85967412']
```

```
]
```

```
Name = StringVar()
```

```
Number = StringVar()
```

```
frame = Frame(root)
```

```
frame.pack(side=RIGHT)
```

```
scroll = Scrollbar(frame, orient=VERTICAL)
```

```
select = Listbox(frame, yscrollcommand=scroll.set, font=('Times new roman', 16),
```

```
bg="#f0fffc", width=20, height=20, borderwidth=3, relief="groove")
```

```
scroll.config(command=select.yview)
```

```
scroll.pack(side=RIGHT, fill=Y)
```

```
select.pack(side=LEFT, fill=BOTH, expand=1)
```

```
def Selected():
```

```
if len(select.curselection()) == 0:
```

```
    messagebox.showerror("Error", "Please Select the Name")
```

```
else:
```

```
    return int(select.curselection()[0])
```

```
def AddContact():
```

```
if Name.get() != "" and Number.get() != "":
```

```
    contactlist.append([Name.get(), Number.get()])
```

```
    Select_set()
```

```
    EntryReset()
```

```
    messagebox.showinfo("Confirmation", "Successfully Add New Contact")
```

```
else:
```

```
    messagebox.showerror("Error", "Please fill the information")
```

```
def UpdateDetail():
```

```
if Name.get() and Number.get():
```

```
    contactlist[Selected()] = [Name.get(), Number.get()]
```

```
    messagebox.showinfo("Confirmation", "Successfully Update Contact")
```

```
    EntryReset()
```

```
    Select_set()
```

```
elif not (Name.get()) and not (Number.get()) and not (len(select.curselection()) == 0):
```

```
    messagebox.showerror("Error", "Please fill the information")
```

```
else:
```

```
if len(select.curselection()) == 0:
```

```
    messagebox.showerror("Error", "Please Select the Name and \n press Load  
button")
```

```
else:
```

```
    message1 = "" "To Load the all information of  
selected row press Load button.
```

```
    "" "
```

```
messagebox.showerror("Error", message1)
```

```
def Delete_Entry():
```

```
if len(select.curselection()) != 0:
```

```
result = messagebox.askyesno('Confirmation',
```

```
'You Want to Delete Contact\n Which you selected')
```

```
if result == True:
```

```
del contactlist[Selected()]
```

```
Select_set()
```

```
else:
```

```
messagebox.showerror("Error", 'Please select the Contact')
```

```
def VIEW():
```

```
NAME, PHONE = contactlist[Selected()]
```

```
Name.set(NAME)
```

```
Number.set(PHONE)
```

```
def EXIT():
```

```
root.destroy()
```

```
def EntryReset():
```

```
Name.set("")
```

```
Number.set("")
```

```
def Select_set():
```

```
contactlist.sort()
```

```
select.delete(0, END)
```

```
for name, phone in contactlist:
```

```
select.insert(END, name)
```

```
Select_set()
```

```
Label(root, text='Name', font=("Times new roman", 22, "bold"),
```

```
bg='SlateGray3').place(x=30, y=20)
```

```
Entry(root, textvariable=Name, width=30).place(x=200, y=30)
```

```
Label(root, text='Contact No.', font=("Times new roman", 20, "bold"),
bg='SlateGray3').place(x=30, y=70)
Entry(root, textvariable=Number, width=30).place(x=200, y=80)
Button(root, text="ADD", font='Helvetica 18 bold',
bg='#e8c1c7', command=AddContact, padx=20).place(x=50, y=140)
Button(root, text="EDIT", font='Helvetica 18 bold',
bg='#e8c1c7', command=UpdateDetail, padx=20).place(x=50, y=200)
Button(root, text="DELETE", font='Helvetica 18 bold',
bg='#e8c1c7', command=Delete_Entry, padx=20).place(x=50, y=260)
Button(root, text="VIEW", font='Helvetica 18 bold',
bg='#e8c1c7', command=VIEW).place(x=50, y=325)
Button(root, text="RESET", font='Helvetica 18 bold',
bg='#e8c1c7', command=EntryReset).place(x=50, y=390)
Button(root, text="EXIT", font='Helvetica 24 bold',
bg='tomato', command=EXIT).place(x=250, y=470)
root.mainloop()
```

OUTPUT

[GUI Window Opens successfully with background #d3f3f5]

Initial sorted list displayed in Listbox:

- 1. Abhishek Nikam**
- 2. Arun**
- 3. Deepak**
- 4. Ganesh Pawar**
- 5. Gaurav Patil**
- 6. John Maheshwari**
- 7. Karan Patel**
- 8. Mohit Paul**
- 9. Raghul**
- 10. Sakshi Gaikwad**
- 11. Sam Sharma**
- 12. Siddharth Nigam**

Action Triggered: User clicks "Abhishek Nikam" and clicks "VIEW"

UI Action: Text fields populate with Name: "Abhishek Nikam", Contact No.: "78945614"

Action Triggered: User inputs "Zayan" / "99887766" and clicks "ADD"

Pop-up Window: [Confirmation: Successfully Add New Contact]

Listbox Behavior: Updates and sorts "Zayan" to the bottom of the list.

Action Triggered: User selects a contact and clicks "DELETE"

Pop-up Window: [Confirmation: You Want to Delete Contact Which you selected] -> User clicks Yes.

Listbox Behavior: The contact is immediately removed from the right-hand panel view.

Action Triggered: User clicks "EXIT"

Result: Window closes safely and the process finishes with Exit Code 0.

RESULT

Thus, the Python mini-project program to implement a Graphical User Interface (GUI) based Contact Book Application using Tkinter was developed and executed successfully.