D.Y.Patil International University, Akurdi, Pune

School of Computer Science Engineering and Applications (SCSEA)

Final Year - 1st Semester (MCA)

Introduction to AI and ML (MCA1002)

Lab Manual

D Y PATIL
INTERNATIONAL
UNIVERSITY
AKURDI PUNE

Vision of the University:

"To Create a vibrant learning environment – fostering innovation and creativity, experiential learning, which is inspired by research, and focuses on regionally, nationally and globally relevant areas."

Mission of the School:

- *To provide a diverse, vibrant and inspirational learning environment.*
- *To establish the university as a leading experiential learning and research-oriented center.*
- *To become a responsive university serving the needs of industry and society.*
- *To embed internationalization, employability and value thinking*

Advanced Artificial Intelligence

Course Objectives:

Understand the Basics of Artificial intelligence

Understanding AI for problem solving

Understand the Knowledge representation and Reasoning

Understanding generative and deep generative models.

Understanding AI Emerging Trends

Course Outcomes:

On completion of the course, learner will be able to–

CO1: Implement the concept of problem-solving using AI.

CO2: Apply Knowledge representation and Reasoning

CO3: Implement generative and deep generative models

CO4: Keep Abreast the Emerging trends of AI

Program Outcomes:

PO1	**Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
PO2	**Problem analysis**: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
PO3	**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 the public health and safety, and the cultural, societal, and environmental considerations.
PO4	**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.
PO5	**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.
PO6	**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.
PO7	**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.
PO8	**Ethics**: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
PO9	**Individual and team work**: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
PO10	**Communication**: Communicate effectively on complex engineering activities with the engineering community and with 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.
PO11	**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.
PO12	**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.

Rules and Regulations for Laboratory:

- Students should be regular and punctual to all the Lab practical
- Lab assignments and practical should be submitted within given time.
- Mobile phones are strictly prohibited in the Lab.
- Please shut down the computers before leaving the Lab.
- Please switch off the fans and lights and keep the chair in proper position before leaving the Lab
- Maintain proper discipline in Lab

D.Y.Patil International University, Akurdi, Pune

School of Computer Science Engineering and Applications

Index

***Absent/Attended/Late/Partially Completed/Completed**

CERTIFICATE

This is to certify that Mr. /Miss ___
PRN: _ _ _ _ _ _ _ ____ _ _ _ of class: _ _ _ _ _ _ __________ has completed practical/term work in the course of Introduction to AI and ML First Year Frist Semester (MCA) within SCSEA, as prescribed by D. Y. Patil International University, Pune during the academic year_ _ _ _ _ _ _.

Date: **Teaching Assistant** **Faculty I/C** **Director**
(SCSEA)

Practical 01

Student Name: YASH SURYARAO
Date of Experiment: 30-08-2024
Date of Submission: 06-09-2024
PRN No: 20240804062

Aim: Implement various statistical learning approaches and probability concept on any types of data.

Objectives: 1) To implement various statistical learning approaches
2) various probabilityconcepts

Software/Tool: Python/Jupyter/Anaconda/Colab

Theory

1) **Statistical Learning Approaches:**
Statistics is a branch of mathematics that deals with collecting, organizing, analyzing, interpreting, and visualizing empirical data for summarizing making inferences & drawing conclusions
some basic Statistical Learning approaches used are
a) Statistics in Data Preparation
Statistical methods are required in the preparation of train and test data for your machine learning model. This includes techniques for:
   - Outlier detection.
   - Missing value imputation.
   - Data sampling
   - Data scaling
   - Variable encoding and much more.

b) Statistics in Model Evaluation
Statistical methods are required when evaluating the skill of a machine learning model on data not seen during training.
This includes techniques for:
   - Data sampling
   - Data Resampling
   - Experimental design

c) Statistics in Model Selection
Statistical methods are required when selecting a final model or model configuration to use for a predictive modeling problem.
These include techniques for:
   - Checking for a significant difference between results.
   - Quantifying the size of the difference between results

d) Statistics in Model Presentation
Statistical methods are required when presenting the skill of a final model to stakeholders.
This includes techniques for:
   - Summarizing the expected skill of the model on average.
   - Quantifying the expected variability of the skill of the model in practice.

e) Statistics in Prediction
Statistical methods are required when making a prediction with a finalized model on

new data.

This includes techniques for:

- Quantifying the expected variability for the prediction.
- This might include estimation statistics such as prediction intervals

2) **Probability Theory**

- Probability is defined as the chance of happening or occurrences of an event. Generally,the possibility of analyzing the occurrence of any event concerning previous data is called probability. Probability measures the likelihood of an event's occurrence. In situations where the outcome of an event is uncertain; we discuss the probability of specific outcomes to understand their chances of happening.
- Some of the basic concepts of probability used are

a) Conditional Probability
b) Joint Probability
c) Marginal Probability
d) Bayes Theorem

Algorithm:

Step 1: Data Loading and Inspection

- **Load the student data:** Load gold price prediction.csv to begin working with the dataset.
- **Initial data check:** Display the first and last few rows to get a sense of the entries and structure.
- **Descriptive statistics:** Review basic statistics like mean and standard deviation for columns such as "Price Today" and "Price Tomorrow"

Step 2: Data Summary and Info

- **Summary statistics:** Use df.describe() to get a comprehensive summary of numerical columns.
- **Data types and non-null check:** Run df.info() to confirm data types and check for any missing values.

Step 3: Standardization Using Z-Score

- **Calculate Z-scores:** Compute the Z-score for "Price Today" to see how Today's Gold Price compares to the average in terms of standard deviations.

Step 4: Log Transformation

- **Apply log transformation:** Use a log transformation on "Price Today" to smooth out the distribution and reduce skewness, especially helpful if there's a large range or outliers.

Step 5: Visualization

- **Scatter plot of Price Today vs. Price Tomorrow:** Plot "Price Today" on the x-axis and "Price Tomorrow" on the y-axis to visually assess any possible relationship between Today and Tomorrow price.

Code and Output:

Dataset: https://drive.google.com/file/d/1i_99YhHa7vA_bdbgY-L9HmfF2plpPT7I/view?usp=sharing

```
# AIML LAB-1 Statistical Learning Approaches and Probability Concepts

Name: Yash Suryarao
PRN: 20240804062
```

```
# Importing necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import zscore

#Loading Dataset
data = pd.read_csv('Gold Price Prediction(LAB_1-DATASET).csv')
data.head(10)

# Create DataFrame
df = pd.DataFrame(data)

print("Dataset:\n", df)

print("\nDescriptive Statistics:\n", df.describe())
```

```
Dataset:
        Date  Price 2 Days Prior  Price 1 Day Prior  Price Today  \
0    8/7/24              2405.87            2384.90      2385.83
1    8/6/24              2442.74            2405.87      2384.90
2    8/5/24              2447.17            2442.74      2405.87
3    8/2/24              2447.23            2447.17      2442.74
4    8/1/24              2411.09            2447.23      2447.17
..      ...                  ...                ...          ...
671  1/7/22              1810.28            1791.61      1796.41
672  1/6/22              1813.88            1810.28      1791.61
673  1/5/22              1804.27            1813.88      1810.28
674  1/4/22              1815.73            1804.27      1813.88
675  1/3/22              1804.64            1829.05      1804.27

     Price Tomorrow  Price Change Tomorrow  Price Change Ten  Std Dev 10  \
0           2385.83                   0.93              8.82   30.155078
1           2385.83                   0.93               NaN   29.423936
2           2384.90                 -20.97               NaN   28.341301
```

```
# Calculating Z-score for standardization
df['Price_Z'] = zscore(df['Price Today'])
print("\nZ-Score for Gold Price:\n", df[['Date', 'Price Today', 'Price_Z']])
```

```
Z-Score for Gold Price:
        Date  Price Today   Price_Z
0    8/7/24      2385.83  2.205008
1    8/6/24      2384.90  2.200216
2    8/5/24      2405.87  2.308278
3    8/2/24      2442.74  2.498276
4    8/1/24      2447.17  2.521105
..      ...          ...       ...
671  1/7/22      1796.41 -0.832384
672  1/6/22      1791.61 -0.857119
673  1/5/22      1810.28 -0.760909
674  1/4/22      1813.88 -0.742358
675  1/3/22      1804.27 -0.791880

[676 rows x 3 columns]
```

```
# Log Transformation
df['Log_Price'] = np.log(df['Price Today'])
print("\nLog Transformation of Gole Price: \n", df [['Date', 'Price Today', 'Log_Price']])
```

```
Log Transformation of Gole Price:
        Date  Price Today  Log_Price
0    8/7/24      2385.83   7.777302
1    8/6/24      2384.90   7.776912
2    8/5/24      2405.87   7.785667
3    8/2/24      2442.74   7.800876
4    8/1/24      2447.17   7.802688
..      ...          ...        ...
671  1/7/22      1796.41   7.493546
672  1/6/22      1791.61   7.490870
673  1/5/22      1810.28   7.501237
674  1/4/22      1813.88   7.503223
675  1/3/22      1804.27   7.497911

[676 rows x 3 columns]
```

```
# Log Transformation
df['Log_Price'] = np.log(df['Price Today'])
print("\nLog Transformation of Gole Price: \n", df [['Date', 'Price Today', 'Log_Price']])
```

```
Log Transformation of Gole Price:
        Date  Price Today  Log_Price
0    8/7/24      2385.83   7.777302
1    8/6/24      2384.90   7.776912
2    8/5/24      2405.87   7.785667
3    8/2/24      2442.74   7.800876
4    8/1/24      2447.17   7.802688
..      ...          ...        ...
671  1/7/22      1796.41   7.493546
672  1/6/22      1791.61   7.490870
673  1/5/22      1810.28   7.501237
674  1/4/22      1813.88   7.503223
675  1/3/22      1804.27   7.497911

[676 rows x 3 columns]
```

```
]: # SAMPLING


# Randomly sample 20% of the data
sample_data = df.sample(frac=0.2, random_state=42)
```

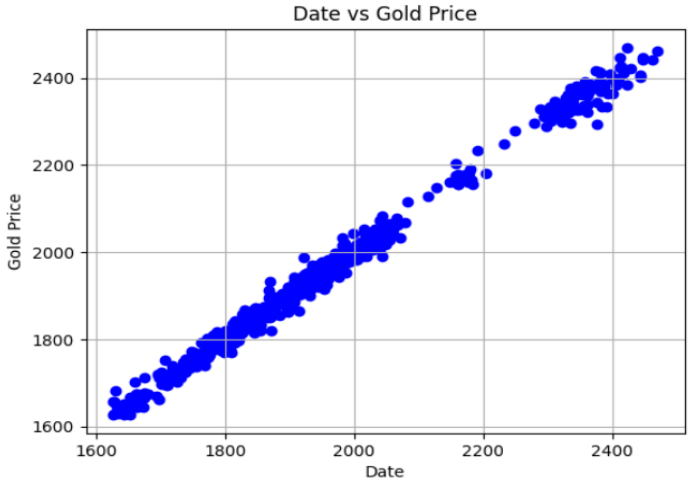
```
# Display the sampled data
print(sample_data.head())
```

```
       Date  Price 2 Days Prior  Price 1 Day Prior  Price Today  \
641  2/18/22              1869.85            1900.09      1897.36
302   6/8/23              1963.36            1944.53      1965.48
369   3/7/23              1855.37            1846.40      1813.97
493  9/14/22              1725.42            1702.16      1697.65
579  5/17/22              1811.09            1826.49      1814.62

     Price Tomorrow  Price Change Tomorrow  Price Change Ten  Std Dev 10  \
641         1909.53                  12.17             73.32   28.920587
302         1960.75                  -4.73            -50.90   11.527303
369         1815.28                   1.31            128.98   14.441964
493         1662.54                 -35.11            -41.17    9.251092
579         1816.19                   1.57             21.82   29.934636

     Twenty Moving Average  Fifty Day Moving Average  ...  Volume   \
641              1834.1490                 1278.6300  ...       71
302              1967.6630                 1991.8558  ...      136
369              1839.8190                 1869.9368  ...      118
493              1725.1155                 1739.9992  ...       95
579              1876.2070                 1918.8924  ...       80

     Treasury Par Yield Month  Treasury Par Yield Two Year  \
641                      0.03                         1.47
302                      5.25                         4.52
369                      4.80                         5.00
493                      2.54                         3.78
579                      0.61                         2.71

     Treasury Par Yield Curve Rates (10 Yr)     DXY  SP Open    VIX   Crude  \
641                                    1.92   95.83  4384.57  26.66   91.63
302                                    3.73  104.07  4268.69  14.14   72.47
369                                    3.97  104.30  4048.26  18.64   80.50
493                                    3.41  109.93  3940.73  26.73   87.94
579                                    2.98  104.17  4052.00  27.07  113.87

     Price Today_Z  Log_Price Today
641      -0.312170         7.548219
302       0.038866         7.583492
```

```
l]: # Min-Max Scaling for numerical columns

numerical_cols = ['Price 2 Days Prior', 'Price 1 Day Prior', 'Price Today', 'Pri
scaler = MinMaxScaler()

# Apply Min-Max Scaling
df[numerical_cols] = scaler.fit_transform(df[numerical_cols])
print("\nMin-Max Scaled Data:\n", df[['Date'] + numerical_cols].head())
```

```
Min-Max Scaled Data:
      Date  Price 2 Days Prior  Price 1 Day Prior  Price Today  Price Tomorrow
0  8/7/24            0.924307           0.899420     0.900523        0.900523
1  8/6/24            0.968064           0.924307     0.899420        0.900523
2  8/5/24            0.973321           0.968064     0.924307        0.899420
3  8/2/24            0.973392           0.973321     0.968064        0.924307
4  8/1/24            0.930502           0.973392     0.973321        0.968064
```

```
# Visualization with scatter plot
plt.scatter (df['Price Today'], df['Price Tomorrow'], color='blue')
plt.title('Date vs Gold Price')
plt.xlabel('Date')
plt.ylabel('Gold Price')
plt.grid(True)
plt.show()
```



Conclusion:

This practical highlight how crucial statistical learning and probability theory are as foundational tools in the field of machine learning and data analysis. At each step from preparing data and evaluating model performance to selecting the best model and making predictions statistical methods play an essential role in ensuring accurate, consistent, and well-informed outcomes. Probability theory complements this by providing a way to predict outcomes and measure uncertainties, which strengthens both the reliability and the robustness of the models.

By combining these approaches, this practical creates a well-rounded framework for data processing, model assessment, and accurate prediction-making. This approach not only supports current needs but also lays a solid groundwork for future research and real-world applications in machine learning, where statistical and probabilistic insights can drive better and more dependable decisions.

Practical 02

Student Name: YASH SURYARAO
Date of Experiment: 06-09-2024
Date of Submission: 13-09-2024
PRN No: 20240804062

Aim: Implement Data Munging/ Data wrangling

Objectives: Test various data munging steps on the given dataset

Software/Tool: Python/Jupyter/Anaconda/Colab

Theory

Data is the foundation of present-day decision-making, yet crude data is frequently messy and unstructured. This is where data munging, or data cleaning, becomes an integral factor.

- Data munging, sometimes called data wrangling or data cleaning, is converting and mappingunprocessed data into a different format to improve its suitability and value for various downstream uses, including analytics.
- This procedure entails preparing raw data for analysis by cleaning, organizing, and enrichingit in a readable format.
- Data munging holds immense significance in the field of data analysis, playing a crucial rolein ensuring the quality and reliability of the data used for making informed decisions.

Key Aspects highlighting significance of data munging

- Accuracy and Precision:
- Quality Improvement
- Compatibility
- Analysis Readiness
- Facilitation of Decision-Making
  Tools used
- Python/R
- SQL
- ETL (Apache Spark, Talend, or Informatica, which automate data munging processes.)

Key Steps in data munging

1. **Data Cleaning:** This involves identifying and correcting errors or inconsistencies in the data.Common tasks include handling missing values, correcting data formats (e.g., dates, numeric values), and removing duplicates.
2. **Data Transformation:** Data often needs to be transformed to fit the analytical requirements. Thismay include converting categorical data into numerical format (encoding), normalizing or scaling numeric data, and aggregating or summarizing data.
3. **Handling Missing Data:** Techniques such as imputation (replacing missing values withestimated ones) or deletion (removing rows or columns with missing data) are used to handle missing dataappropriately.
4. **Data Integration:** Combining data from multiple sources involves aligning schemas, resolvinginconsistencies, and merging datasets to create a unified view.
5. **Feature Engineering:** Creating new features or variables from existing data that can enhance thepredictive power of machine learning models.
6. **Data Validation:** Checking data integrity to ensure it meets expected standards and

business rules.

Benefits of data munging

1. Eliminate Data Siloes and Integrate Various Sources:
2. Improve Data Usability:
3. Process Large Volumes of Data:
4. Ensure High Data Quality:

1) Uninformed Search

Algorithm:

Step 1: Data Loading and Initial Inspection

- Load the dataset airline.csv and inspect it for any missing values.

Step 2: Handling Missing Values

- Use df.isnull().sum() to identify columns with missing data.
- Fill missing values in AgeYears with the mean: df['AgeYears'].fillna(df['AgeYears'].mean(), inplace=True).
- Fill missing SuiteNumber or other columns as needed using appropriate strategies.

Step 3: Outlier Detection and Removal

- Detect outliers in relevant numerical columns like TicketCost using Z-scores and filter out extreme values.

Step 4: Normalization

- Normalize columns like TicketCost and AgeYears to scale values between 0 and 1.

Step 5: Engineering

- Encode categorical features, if present, using numerical or one-hot encoding as appropriate.

Code and Output:

Dataset: https://drive.google.com/file/d/18Lnl73nPEJ0MUdmCEbSOW2H6lXUnfdra/view?usp=drive_link

```
# AIML LAB-2 Data Munging, Data wrangling

Name: Yash Suryarao
PRN: 20240804062
```

```
# STRP-1 : Loading Dataset

import pandas as pd

# Loading the Airplane Dataset
df = pd.read_csv('airline.csv')

# Display top 10 data
df.head(10)
```

	PassengerID	CompletedTrip	CabinClass	TravelerName	Gender	AgeYears	FamilyOnBoard	Guardians	BoardingPassNumber	TicketCost	SuiteNumber	BoardingPort
0	1	0	Economy	Passenger_0	male	22.0	1	0	BP93225	7.25	NaN	ORD
1	2	1	First	Passenger_1	female	NaN	1	0	BP98731	71.28	Suite 232	JFK
2	3	1	Economy	Passenger_2	female	26.0	0	0	BP75445	7.92	NaN	ORD
3	4	1	First	Passenger_3	female	35.0	1	0	BP11329	53.10	Suite 127	ORD
4	5	0	Economy	Passenger_4	male	35.0	0	0	BP49939	8.05	NaN	ORD
5	6	0	Economy	Passenger_5	male	27.0	0	0	BP53061	8.46	NaN	LAX
6	7	0	First	Passenger_6	male	NaN	0	0	BP28292	51.86	Suite 289	ORD
7	8	1	Economy	Passenger_7	female	NaN	1	1	BP66454	16.70	NaN	ORD
8	9	0	Business	Passenger_8	female	58.0	0	0	BP37379	26.55	NaN	ORD
9	10	1	Economy	Passenger_9	male	39.0	0	0	BP53842	13.00	NaN	JFK

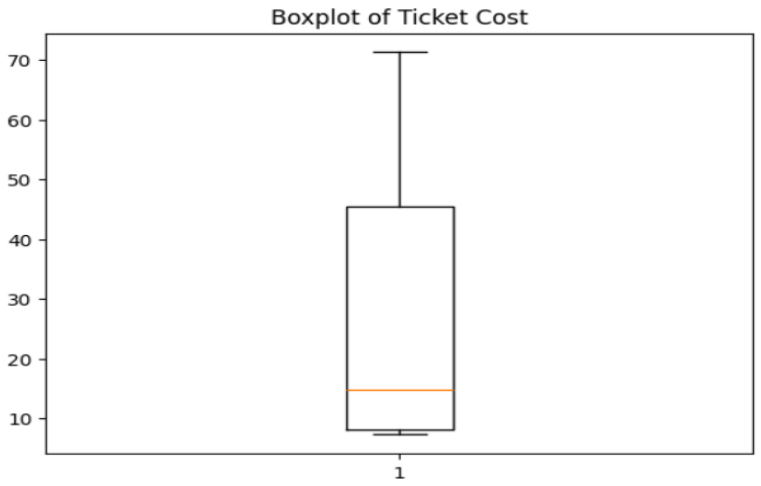
```
# STRP-2 : Handling Missing Values

# Checking Missing values
df.isnull().sum()
```

```
PassengerID          0
CompletedTrip        0
CabinClass           0
TravelerName         0
Gender               0
AgeYears             3
FamilyOnBoard        0
Guardians            0
BoardingPassNumber   0
TicketCost           0
SuiteNumber          7
BoardingPort         0
dtype: int64
```

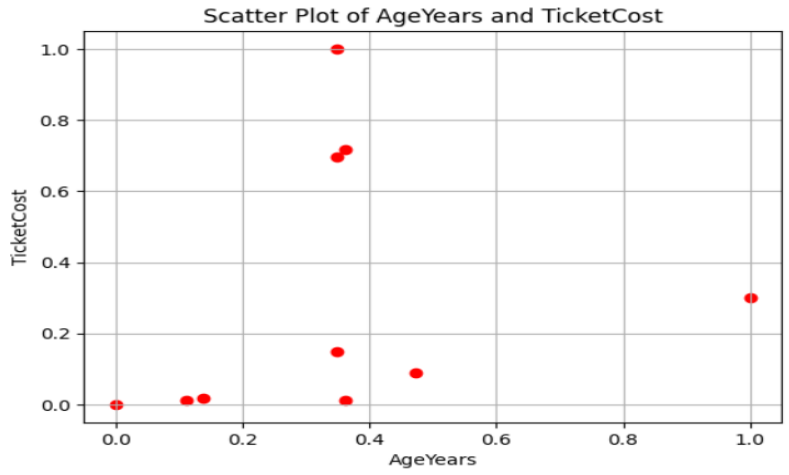
```
# Fill missing 'AgeYears' values with the mean age
df['AgeYears'].fillna(df['AgeYears'].mean(), inplace=False)

# Fill missing 'SuiteNumber' values with the most common value
df['SuiteNumber'].fillna(df['SuiteNumber'].mode()[0], inplace=True)

# Verify Changes
df.isnull().sum()
```

```
PassengerID          0
CompletedTrip        0
CabinClass           0
TravelerName         0
Gender               0
AgeYears             0
FamilyOnBoard        0
Guardians            0
BoardingPassNumber   0
TicketCost           0
SuiteNumber          0
BoardingPort         0
dtype: int64
```

```
# STRP-3 : Handling Outliers

import matplotlib.pyplot as plt

# Visualize outliers in 'TicketCost' using a boxplot
plt.boxplot(df['TicketCost'])
plt.title('Boxplot of Ticket Cost')
plt.show()

# Remove outliers using Z-score method
df['TicketCost_zscore'] = (df['TicketCost'] - df['TicketCost'].mean()) / df ['TicketCost'].std()
df_no_outliers = df[df['TicketCost_zscore'].abs() < 3]

# Drop the Z-score colmn after removing outliers
df_no_outliers.drop(columns=['TicketCost_zscore'], inplace=True)
```



```
import matplotlib.pyplot as plt

plt.scatter(df['AgeYears'],df['TicketCost'],color='Red')
plt.title('Scatter Plot of AgeYears and TicketCost')
plt.xlabel('AgeYears')
plt.ylabel('TicketCost')
plt.grid(True)
plt.show()
```




```
# STRP-4 : Data Normalization

from sklearn.preprocessing import MinMaxScaler

# Initialize the MinMaxScaler
scaler=MinMaxScaler()
# Normalize the 'TicketCOst' and 'AgeYears' columns
df[['TicketCost', 'AgeYears']] = scaler.fit_transform(df[['TicketCost', 'AgeYears']])
df[['TicketCost', 'AgeYears']].head(10)
```

	TicketCost	AgeYears
0	0.000000	0.000000
1	1.000000	0.349206
2	0.010464	0.111111
3	0.716071	0.361111
4	0.012494	0.361111
5	0.018897	0.138889
6	0.696705	0.349206
7	0.147587	0.349206
8	0.301421	1.000000
9	0.089802	0.472222

```
# STRP-5 : Removing irrelevant Data

# Removing irrelevant data
df.drop(columns=['TravelerName','BoardingPassNumber','SuiteNumber'], inplace=True)
df.head(10)
```

	PassengerID	CompletedTrip	CabinClass	Gender	AgeYears	FamilyOnBoard	Guardians	TicketCost	BoardingPort	TicketCost_zscore
0	1	0	Economy	male	0.000000	1	0	0.000000	ORD	-0.811906
1	2	1	First	female	0.349206	1	0	1.000000	JFK	1.900379
2	3	1	Economy	female	0.111111	0	0	0.010464	ORD	-0.783525
3	4	1	First	female	0.361111	1	0	0.716071	ORD	1.130281
4	5	0	Economy	male	0.361111	0	0	0.012494	ORD	-0.778019
5	6	0	Economy	male	0.138889	0	0	0.018897	LAX	-0.760651
6	7	0	First	male	0.349206	0	0	0.696705	ORD	1.077755
7	8	1	Economy	female	0.349206	1	1	0.147587	ORD	-0.411608
8	9	0	Business	female	1.000000	0	0	0.301421	ORD	0.005634
9	10	1	Economy	male	0.472222	0	0	0.089802	JFK	-0.568339

```
# STRP-6 : Feature Engineering

# Creating a new feature
df['FamilySize'] = df['FamilyOnBoard'] + df['Guardians']

# View the new column
df[['FamilyOnBoard', 'Guardians','FamilySize']].head(10)
```

	FamilyOnBoard	Guardians	FamilySize
0	1	0	1
1	1	0	1
2	0	0	0
3	1	0	1
4	0	0	0
5	0	0	0
6	0	0	0
7	1	1	2
8	0	0	0
9	0	0	0

```
# STRP-7 : Data Formatting

# Convert 'Gender' column to numerical (1 or 0)
df['Gender']=df['Gender'].map({'male':1, 'female':0})
print(df)

df = pd.get_dummies(df, columns=['BoardingPort'])
```

```
   PassengerID  CompletedTrip CabinClass  Gender  AgeYears  FamilyOnBoard  \
0            1              0    Economy       1  0.000000              1
1            2              1      First       0  0.349206              1
2            3              1    Economy       0  0.111111              0
3            4              1      First       0  0.361111              1
4            5              0    Economy       1  0.361111              0
5            6              0    Economy       1  0.138889              0
6            7              0      First       1  0.349206              0
7            8              1    Economy       0  0.349206              1
8            9              0   Business       0  1.000000              0
9           10              1    Economy       1  0.472222              0

   Guardians  TicketCost BoardingPort  TicketCost_zscore  FamilySize
0          0    0.000000          ORD          -0.811906           1
1          0    1.000000          JFK           1.900379           1
2          0    0.010464          ORD          -0.783525           0
3          0    0.716071          ORD           1.130281           1
4          0    0.012494          ORD          -0.778019           0
5          0    0.018897          LAX          -0.760651           0
6          0    0.696705          ORD           1.077755           0
7          1    0.147587          ORD          -0.411608           2
8          0    0.301421          ORD           0.005634           0
9          0    0.089802          JFK          -0.568339           0
```

Conclusion:

In this lab, we prepared an airline dataset for analysis by applying essential data wrangling techniques. Missing values in numeric columns like "AgeYears" were filled with mean values, while categorical ones like "BoardingPort" used mode imputation for consistency. Outliers in "TicketCost" were managed with z-scores, and Min-Max scaling was applied to standardize numerical values. Feature engineering added a "FamilySize" column, enhancing insights into travel behavior. Finally, categorical variables were standardized with encoding techniques. This comprehensive data preparation improved data quality and usability, ensuring a reliable foundation for accurate analysis and predictive modeling.

Practical 03

Student Name: YASH SURYARAO
Date of Experiment: 13-09-2024
Date of Submission: 20-09-2024
PRN No: 20240804062

Aim: Implement a program based on state space search in AI

Objectives: a. using state space search for a particular problem

Software/Tool: Python/Jupyter/Anaconda/Colab

Theory

An essential method in artificial intelligence is state space search, which looks for potential states and their transitions to solve issues. According to this method, the problem is modeledas a state space, with each state representing a possible configuration and transitions denotingactions or operations that change the state of the problem. Finding a route that meetspredetermined requirements from an initial state to a goal state is the aim. To locate a solution,state space search entails methodically going through every potential state for an issue. Thisapproach can be used to solve a variety of AI issues, including pathfinding, solving puzzles,playing games, and more. The fundamental concept is to visualize the issue as a graph withnodes standing in for states and edges for transitions.

Important ideas consist of:

- **State:** A specific configuration of the problem.
- **Initial State:** The starting point of the search.
- **Goal State:** The desired end configuration.
- **Transition:** An action that changes one state to another.
- **Path:** A sequence of states connected by transitions.
- **Search Strategy:** The method used to explore the state space.

Steps in State space search

Step 1: Define the State Space: Determine the collection of all potential states and their interchanging states

Step 2: Pick a Search Strategy: BFS, DFS, UCS, Greedy Best First, A* Search

Step 3: Start the Search: Add the initial state to the frontier (the collection of states to beinvestigated) by starting there.

Step 4: Extend the Nodes: Using the selected search technique, iteratively expands nodes from the frontier, producing successor states and appending them to the frontier. After each node has been expanded, determine whether it now corresponds to the desired state. If so, retrace your route tothe objective and call off the hunt.

Step 5: Address State Repetition: Put in place safeguards to prevent revisiting the same state,including keeping track of the states you've been to.

Step 6: End the Search: The search comes to an end when the desired state is discovered or, in theevent that no viable solution is identified, when every state has been investigated.

AI systems are able to tackle complicated issues in an organized and methodical manner by employing these methods to systematically explore the state space.

Applications

Pathfinding: Finding the best pathways using algorithms such as A* in robotics and GPS.

- **Puzzle solving:** resolving puzzles like Rubik's Cube, Sudoku, and the 8-puzzle.
- **AI for gaming:** To assess potential moves in board games like chess, checkers, and others.
- **Planning:** The automated scheduling of tasks in logistics and robotics to achieve a specificobjective.
- Natural language processing involves computer translation and sentence parsing by examining many interpretations.
- **Theorem Proving:** Examining logical proofs by looking for potential logical inference sequences.

Algorithm:

Step 1: Initialize the Grid

Create a grid that represents the maze, where 0 indicates an open path and 1 indicates a wall.

Step 2: Define a Valid Move Function

Implement a function is_valid_move(x, y, grid) that checks:

- If (x, y) is within the grid's bounds.
- If the cell (x, y) is an open path (grid[x][y] == 0).

Step 3: Implement Depth-First Search (DFS)

Define a recursive DFS function: DFS (grid, x, y, path)

- **Base Case:** If the current position (x, y) is the goal position, return the current path.
- **Mark the Cell:** Mark the current cell as visited by setting grid[x][y] = 1.
- **Explore Moves**: Explore the four possible directions (down, up, right, left) by iterating through them.
  - For each direction, compute the new coordinates (new_x, new_y).
  - Check if the move is valid using is_valid_move(new_x, new_y, grid).
  - If valid, recursively call DFS with the new coordinates and the updated path.
- **Backtracking:** If no valid path is found from the current cell, unmark it (grid[x][y] = 0) to allow other paths to explore it.

Step 4: Start DFS Search

Call the DFS function starting from the initial position (0, 0) with an empty path list.

Step 5: Output the Result

If a path is found, print the path. If no path exists, indicate that no path was found.

Code and Output:

```
# AIML LAB-3 state space search

Name: Yash Suryarao
PRN: 20240804062
```

```
def is_valid_move(x, y, grid):
    return 0 <= x < len(grid) and 0 <= y < len(grid[0]) and grid[x][y] == 0

def dfs(grid, x, y, path):
    # Goal condition
    if (x, y) == (2, 2):
        return path

    # Possible moves (down, up, right, left)
    moves = [(1, 0), (-1, 0), (0, 1), (0, -1)]

    for dx, dy in moves:
        new_x, new_y = x + dx, y + dy
        if is_valid_move(new_x, new_y, grid):
            grid[new_x][new_y] = 1  # Mark as visited
            result = dfs(grid, new_x, new_y, path + [(new_x, new_y)])
            if result:  # If a path is found
                return result
            grid[new_x][new_y] = 0  # Unmark on backtrack

    return None  # No path found

# 0 = open path, 1 = wall
grid = [
    [0, 0, 0],
    [1, 1, 0],
    [0, 0, 0]
]

# Starting DFS from (0, 0)
grid[0][0] = 1  # Mark starting point as visited
path = dfs(grid, 0, 0, [(0, 0)])

if path:
    print("Path found:", path)
else:
    print("No path found.")
```

```
Path found: [(0, 0), (0, 1), (0, 2), (1, 2), (2, 2)]
```

```
# HOMEWORK - Changing The Root

def is_valid_move(x, y, grid):
    return 0 <= x < len(grid) and 0 <= y < len(grid[0]) and grid[x][y] == 0

def dfs(grid, x, y, path):
    # Goal condition
    if (x, y) == (4, 2):
        return path

    # Possible moves (down, up, right, left)
    moves = [(1, 0), (-1, 0), (0, 1), (0, -1)]

    for dx, dy in moves:
        new_x, new_y = x + dx, y + dy
        if is_valid_move(new_x, new_y, grid):
            grid[new_x][new_y] = 1  # Mark as visited
            result = dfs(grid, new_x, new_y, path + [(new_x, new_y)])
            if result:  # If a path is found
                return result
            grid[new_x][new_y] = 0  # Unmark on backtrack

    return None  # No path found
```

```
# Function to print the grid with highlighted path
def print_grid_with_path(grid, path):
    for i in range(len(grid)):
        for j in range(len(grid[0])):
            if (i, j) in path:
                print("\033[0;31m 0 \033[0m", end='')  # Mark path with green asterisk
            elif grid[i][j] == 1:
                print(" * ", end='')  # Wall
            else:
                print(" . ", end='')  # Open path
        print()  # New line for next row

# 0 = open path, 1 = wall
grid = [
    [0, 0, 0],
    [1, 1, 0],
    [0, 0, 0],
    [0, 1, 1],
    [0, 0, 0]
]

# Starting DFS from (0, 0)
grid[0][0] = 1  # Mark starting point as visited
path = dfs(grid, 0, 0, [(0, 0)])

if path:
    print("Path found:", path)
    print("Grid with highlighted path:")
    print_grid_with_path(grid, path)
else:
    print("No path found.")
```

```
Path found: [(0, 0), (0, 1), (0, 2), (1, 2), (2, 2), (2, 1), (2, 0), (3, 0), (4, 0), (4, 1), (4, 2)]
Grid with highlighted path:
 0  0  0
 *  *  0
 0  0  0
 0  *  *
 0  0  0
```

Conclusion:

In conclusion, this lab effectively showcased the implementation of a **Depth First Search (DFS)** algorithm for navigating a grid, highlighting its utility in AI pathfinding tasks. Through defining a recursive **dfs()** function, the algorithm explored all potential routes while avoiding revisiting cells and using backtracking to handle dead ends. The successful identification of a path from the starting point (0, 0) to the goal (4, 2) illustrated DFS's ability to traverse state spaces. Additionally, the visualization function enhanced comprehension by displaying the grid with the path highlighted, reinforcing how DFS operates step-by-step. This exercise laid a foundational understanding of state space search algorithms, demonstrating their adaptability to real-world pathfinding and problem-solving in artificial intelligence.

Practical 04

Student Name: YASH SURYARAO
Date of Experiment: 20-09-2024
Date of Submission: 04-10-2024
PRN No: 20240804062

Aim: Implement any one of Decision Trees, Random Forest, Support vector Machines algorithm

Objectives: a. Decision Trees **b.** Random Forest **c.** Support vector Machines

Software/Tool: Python/Jupyter/Anaconda/Colab

Theory

Supervised learning is one of the most basic types of machine learning. In this type, the machine learning algorithm is trained on labeled data. Even though the data needs to be labeled accurately for this method to work, supervised learning is extremely powerful when used in the right circumstances. There are two types of supervised learning one Regression and the other is classification. Regression is used when Y is continuous and classification is when Y is discrete. Classification algorithms are used when the output variable is categorical, which means there are two classes such as Yes-No, Male-Female, True-false, etc.

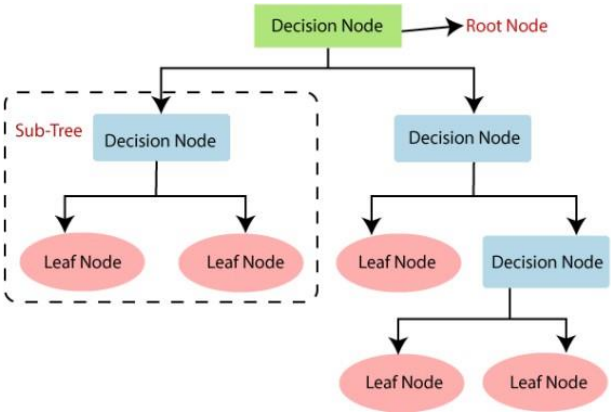
Some of the classifier algorithms

i) Decision Tree
ii) Random Forest
iii) Support vector Machines
iv) Neural network
v) Naïve Bayes
vi) Logistic Regression

1) Decision Tress

Decision Tree is a Supervised learning technique that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems. It is a tree structured classifier, where internal nodes represent the features of a dataset, branches represent the decision rules and each leaf node represents the outcome. In a Decision tree, there are two nodes, which are the Decision Node and Leaf Node. Decision nodes are used to make any decision and have multiple branches, whereas Leaf nodes are the output of those decisions and do not contain any further branches. It is a graphical representation for getting all the possible solutions to a problem/decision based on given conditions. It is called a decision tree because, similar to a tree, it starts with the root node, which expands on further branches and constructs a tree-like structure.

In order to build a tree, we use the CART algorithm, which stands for Classification and Regression Tree algorithm. A decision tree simply asks a question, and based on the answer (Yes/No), it further split the tree into subtrees. A decision tree can contain categorical data (YES/NO) as well as numeric data.



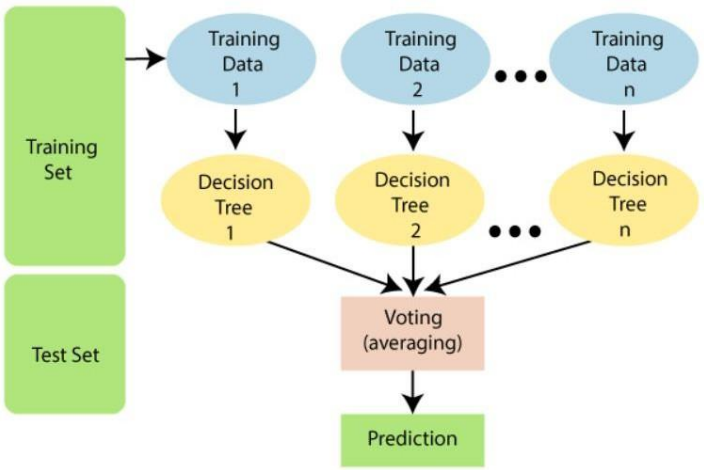
Structure of Decision tree

Algorithm

○ **Step-1:** Begin the tree with the root node, says S, which contains the complete dataset.
○ **Step-2:** Find the best attribute in the dataset using Attribute Selection Measure (ASM).
○ **Step-3:** Divide the S into subsets that contains possible values for the best attributes.
○ **Step-4:** Generate the decision tree node, which contains the best attribute.
○ **Step-5:** Recursively make new decision trees using the subsets of the dataset created in step -3. Continue this process until a stage is reached where you cannot further classify the nodes and called the final node as a leaf node.

2) Random Forrest

Random Forest is a popular machine learning algorithm that belongs to the supervised learning technique. It can be used for both Classification and Regression problems in ML. It is based on the concept of ensemble learning, which is a process of combining multiple classifiers to solve a complex problem and to improve the performance of the model. Random Forest is a classifier that contains a number of decision trees on various subsets of the given dataset and takes the average to improve the predictive accuracy of that dataset. Instead of relying on one decision tree, the random forest takes the prediction from each tree and based on the majority votes of predictions, and it predicts the final output. The greater number of trees in the forest leads to higher accuracy and prevents the problem of overfitting.



Structure of Random Forrest

Random Forest Algorithm

○ Random Forest works in two-phase first is to create the random forest by combining N decision tree,
and second is to make predictions for each tree created in the first phase.
○ The Working process can be explained in the below steps:
Step-1: Select random K data points from the training set.
Step-2: Build the decision trees associated with the selected data points
(Subsets).

Step-3: Choose the number N for decision trees that you want to build.
Step-4: Repeat Step 1 & 2.
Step-5: For new data points, find the predictions of each decision tree, and assign the new data points

3) Support Vector Machine
Support Vector Machine or SVM is one of the most popular supervised learning algorithms, which is used for Classification as well as Regression problems. However, primarily, it is used for Classification problems in Machine Learning. The goal of the SVM algorithm is to create the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future. This best decision boundary is called a hyperplane. SVM chooses the extreme points/vectors that help in creating the hyperplane. These extreme cases are called as support vectors, and hence algorithm is termed as Support Vector Machine

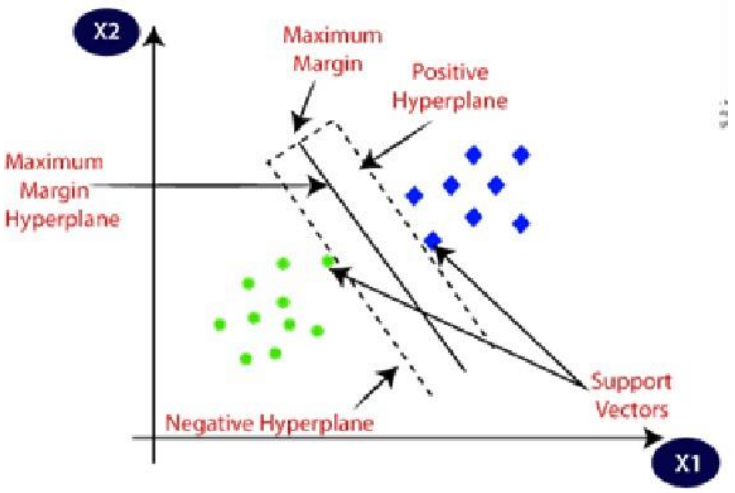


Diagram showing two different categories classified using a decision boundary or hyperplane

Hyperplane in SVM
There can be multiple lines/decision boundaries to segregate the classes in n-dimensional space, but we need to find out the best decision boundary that helps to classify the data points. This best boundary is known as the hyperplane of SVM. The dimensions of the hyperplane depend on the features present in the dataset, which means if there are 2 features (as shown in image), then hyperplane will be a straight line. And if there are 3 features, then hyperplane will be a 2-dimension plane. We always create a hyperplane that has a maximum margin, which means the maximum distance between the data points.

Two types of SVM
1. Linear SVM
2. Nonlinear SVM

Decision Tree

Algorithm:

Step 1: Data Preparation
- Create the dataset.
- Select the features 'Experience (year)', 'Work Hours' and 'Skill Level' as the input features (X), and the target variable 'Performance' as the output (y).

Step 2: Data Splitting
- Split the data into training and testing sets, using 70% of the data for training and 30% for testing.
- Set a fixed random state to ensure reproducibility.

Step 3: Model Initialization and Training

- Initialize a Decision Tree Classifier with a fixed random state
- Train the model using the training data (X_train, Y_train).

Step 4: Prediction

- Use the trained model to make predictions on the test set (X_test).

Step 5: Model Evaluation

- Calculate the model's accuracy score by comparing predictions with the actual values (y_test).
- Generate and display the confusion matrix to assess model performance.
- Generate and display the classification report to evaluate metrics such as precision, recall, and F1-score.

Step 6: Decision Tree Visualization

- Visualize the trained Decision Tree model by plotting it, showing how decisions are made based on the features 'Experience (year)', 'Work Hours' and 'Skill Level'.

Code and Output:

Dataset: https://drive.google.com/file/d/1OIp0c7ipzvfY6jvx-nC9N5vIe8YnkRP4/view?usp=sharing

```
# AIML LAB-4 Decision Tree, Random Forest, Support Vector Machine

Name: Yash Suryarao
PRN: 20240804062
```

```
# Importing Necessary Libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
```

```
# Create the dataset
data = pd.read_csv("employee_performance.csv")

# Imporing or Converting Data Frame
df = pd.DataFrame(data)
print(df)
```

```
   Experience (year)  Work Hours  Skill Level  Performance
0                  3          40            7            1
1                  1          35            5            0
2                  5          45            9            1
3                  2          38            6            0
4                  7          50           10            1
5                  6          48            8            1
6                  4          42            7            1
7                  1          30            4            0
8                  8          55           10            1
9                  2          37            5            0
```

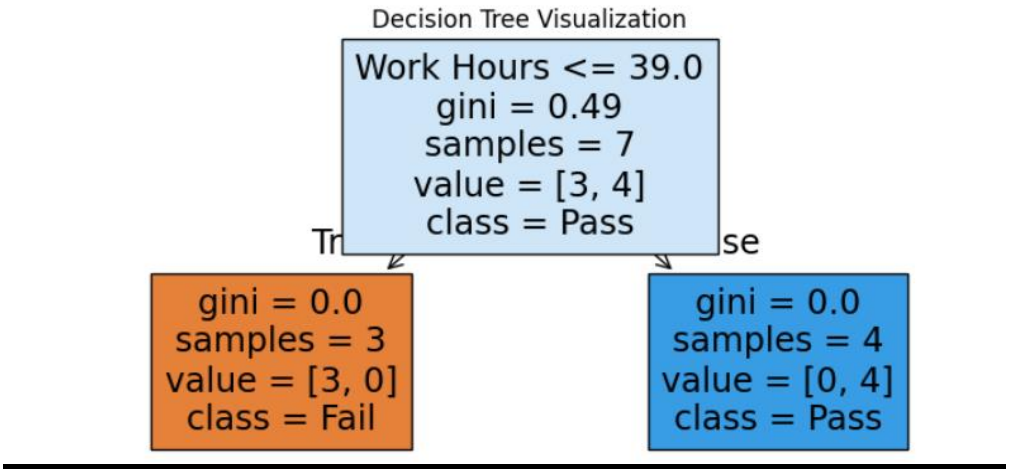
```
# # Defining Features and target variable
X = df[['Experience (years)', 'Work Hours', 'Skill Level']] # Features
y = df['Performance'] # Target variable

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Initialize the decision tree classifier
clf=DecisionTreeClassifier(random_state=42)

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

```
DecisionTreeClassifier(random_state=42)
```

```
# Making prediction
y_pred=clf.predict(X_test)
print("Prediction: ", y_pred)

# Evaluate the model
accuracy=accuracy_score(y_test,y_pred)
conf_matrix=confusion_matrix(y_test,y_pred)
class_report=classification_report(y_test,y_pred)

print(f"Accuracy: {accuracy}")
print("\nConfusion Matrix:")
print(conf_matrix)
print("\nClassification Report:")
print(class_report)
```

```
Prediction:  [1 0 1]
Accuracy: 1.0

Confusion Matrix:
[[1 0]
 [0 2]]

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         1
           1       1.00      1.00      1.00         2

    accuracy                           1.00         3
   macro avg       1.00      1.00      1.00         3
weighted avg       1.00      1.00      1.00         3
```

```
import matplotlib.pyplot as plt
from sklearn.tree import plot_tree

plt.figure(figsize=(9,4))
plot_tree(clf, feature_names=X.columns, class_names=['Fail', 'Pass'], filled=True)
plt.title("Decision Tree Visualization")
plt.show()
```



Conclusion:

In this lab, we implemented a Decision Tree model to classify performance based on features like "Experience (year)," "Work Hours" and "Skill Level." After splitting the dataset, we trained the model and evaluated its accuracy, precision, recall, and F1-score. The model achieved 100% accuracy, suggesting effective classification but also potential overfitting due to the small dataset size. We visualized the Decision Tree to gain insight into decision-making at each node. This lab emphasized the importance of model evaluation and demonstrated the interpretability advantages of Decision Trees for classification tasks.

Random Forest

Algorithm:

- **Step 1: Load the Dataset**
  Import the Iris dataset, a popular dataset for classification tasks. This dataset contains features such as sepal and petal lengths and widths, and the target labels represent three different species of iris flowers.

- **Step 2: Split the Data**
  Divide the dataset into training and testing sets. Typically, 70% of the data is used for training the model, while the remaining 30% is reserved for testing to assess how well the model generalizes to unseen data.

- **Step 3: Initialize the Model**
  Create a Random Forest Classifier model with 100 decision trees (n_estimators=100). Random Forest is an ensemble method that aggregates the results from multiple decision trees to improve accuracy and reduce the risk of overfitting.

- **Step 4: Train the Model**
  Train the Random Forest model using the training data. The model learns to map the input features (e.g., petal width, sepal length) to the target labels (species of the iris flowers) by identifying patterns in the data.

- **Step 5: Make Predictions**
  After training, use the model to predict the target labels (species) for the test data. The model combines the predictions of all decision trees in the forest to provide a final class prediction for each test sample.

- **Step 6: Evaluate the Model**
  Evaluate the model's performance by comparing its predictions against the actual labels in the test set. Accuracy is computed as the proportion of correct predictions out of the total predictions made.

Code and Output:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_iris
```

```
# Load Dataset
data = load_iris()
X = data.data
y = data.target
print("X: \n", X)
print("y: \n", y)
```

```
X:
 [[5.1 3.5 1.4 0.2]
 [4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]
```

```
# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Creating a Random Forest Classifier
rf = RandomForestClassifier(n_estimators=100)
rf
```

```
RandomForestClassifier()
```

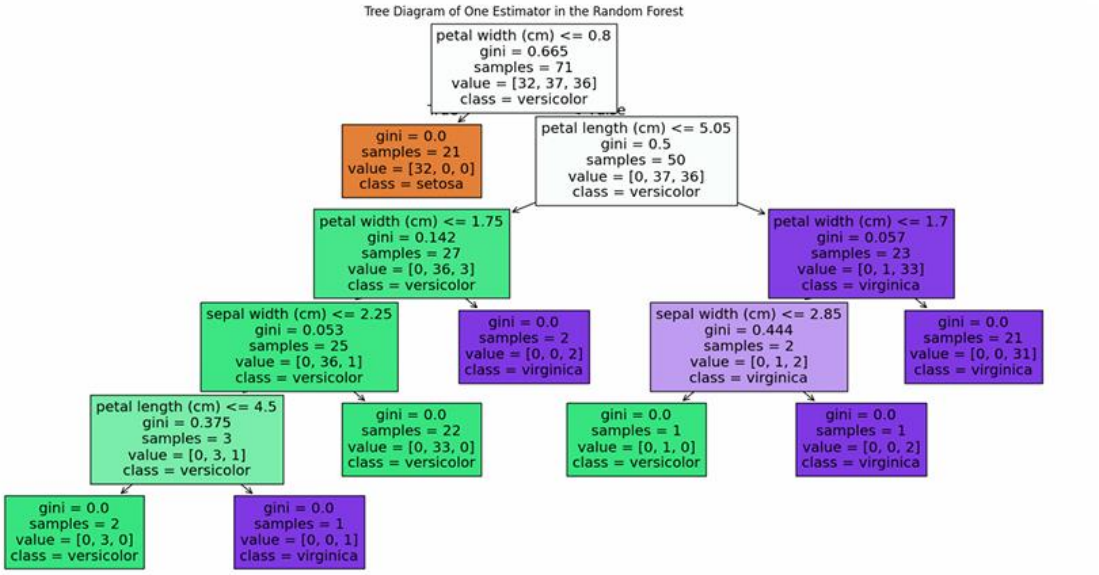
```
# Train the model
rf.fit(X_train, y_train)
```

```
RandomForestClassifier()
```

```
# Make predictions
pred = rf.predict(X_test)
print(pred)

# Evaluate the model
accuracy = rf.score(X_test, y_test)
print(f"Accuracy: {accuracy: .2f}")
```

```
[1 0 2 1 1 0 1 2 1 1 2 0 0 0 0 1 2 1 1 2 0 2 0 2 2 2 2 2 0 0 0 0 1 0 0 2 1
 0 0 0 2 1 1 0 0]
Accuracy:  1.00
```

```
plt.figure(figsize=(20, 10))
plot_tree(rf.estimators_[0], feature_names=data.feature_names, class_names=data.
plt.title("Tree Diagram of One Estimator in the Random Forest")
plt.show()
```



Conclusion:

The Random Forest Classifier delivers reliable and precise predictions for the Iris dataset by combining the insights of multiple decision trees to capture intricate patterns in the data. Its strong performance and consistency make it a top choice for classification problems, particularly when both accuracy and robustness are key priorities.

Support Vector Machine

Algorithm:

- **Step 1: Load Data**
  Import the Iris dataset and load it into variables, assigning the input features to X and the target species labels to y.

- **Step 2: Split Data**
  Divide the dataset into training and testing sets, allocating 70% of the data for training the model and 30% for evaluating its performance.

- **Step 3: Initialize Model**
  Create a Support Vector Machine (SVM) classifier with a linear kernel (kernel='linear'). This kernel is well-suited for linear classification tasks.

- **Step 4: Train Model**
  Fit the SVM model to the training data, allowing it to learn the decision boundaries between different species based on the input features.

- **Step 5: Make Predictions**
  Use the trained SVM model to make predictions on the test data, assigning predicted labels for the flower species in the test set.

- **Step 6: Evaluate Model**
  Assess the model's performance by calculating its accuracy on the test set and displaying the result to determine how well it classifies the unseen data.

Code and Output:

```
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
```

```
# Load dataset
ds = datasets.load_iris()
X = ds.data
y = ds.target

print("X: \n", X)
print("y: \n", y)
```

```
X:
 [[5.1 3.5 1.4 0.2]
 [4.9 3.  1.4 0.2]
```

```
# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Creating a Support Vector Machine classifier
SVM = SVC(kernel = 'linear')
SVM
```

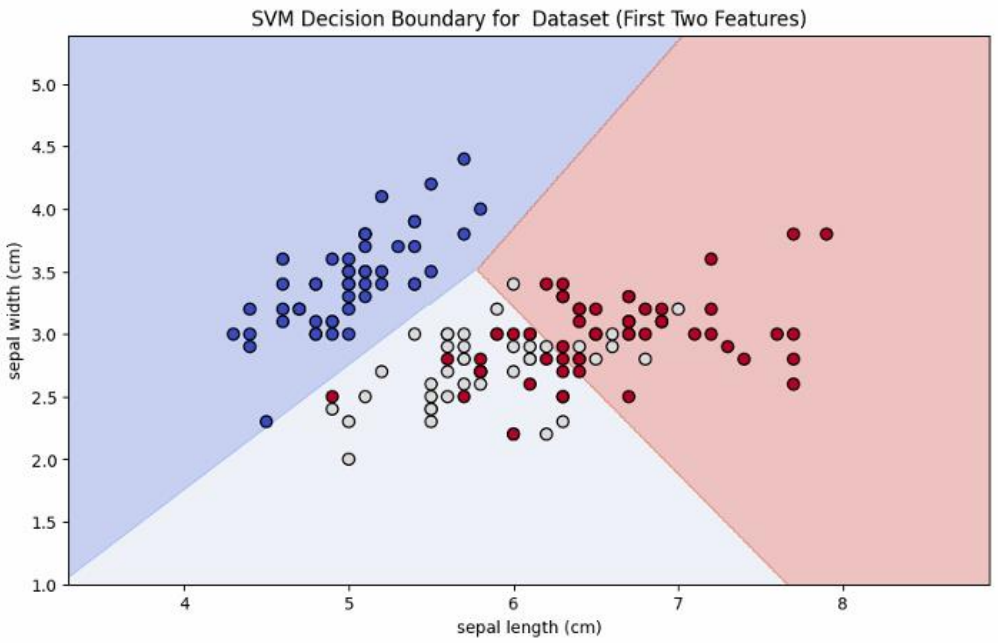
```
SVC(kernel='linear')
```

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

```
SVC(kernel='linear')
```

```
# Make prediction
# Make predictions
pred = SVM.predict(X_test)
print(pred)

# Evaluate the model
accuracy = SVM.score(X_test, y_test)
print(f"Accuracy: {accuracy: .2f}")
```

```
[1 0 2 1 1 0 1 2 1 1 2 0 0 0 0 1 2 1 1 2 0 2 0 2 2 2 2 2 0 0 0 0 1 0 0 2 1
 0 0 0 2 1 1 0 0]
Accuracy:  1.00
```



Conclusion:

The SVM model with a linear kernel delivers accurate classification on the Iris dataset, proving its effectiveness for linearly separable problems. This method is particularly well-suited for high-dimensional datasets, offering a balance of simplicity and strong performance. The accuracy metric provides valuable insight into the model's ability to generalize to unseen data.

Practical 05

Student Name: YASH SURYARAO
Date of Experiment: 04-10-2024
Date of Submission: 11-10-2024
PRN No: 20240804062

Aim: Implement any one of K Means, KNN algorithm

Objectives: a. K Mean b. KNN

Software/Tool: Python/Jupyter/Anaconda/Colab

Theory

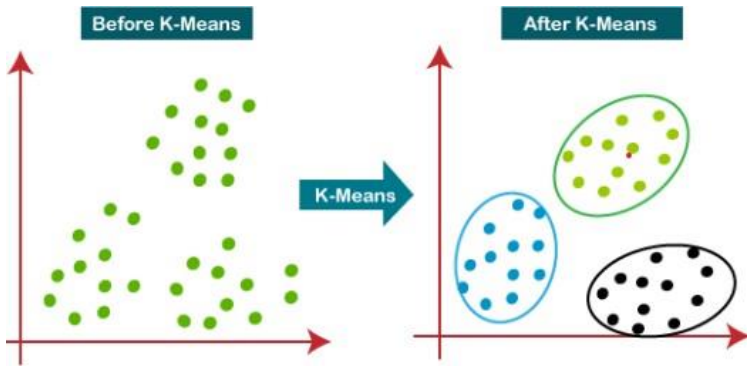
Clustering is an unsupervised machine learning task. Using a clustering algorithm means we are going to give the algorithm a lot of input data with no labels and let it find any groupings in the data it can. Those groupings are called clusters. A cluster is a group of data points that are similar to each other based on their relation to surrounding data points. Clustering is used for things like feature engineering or pattern discovery. There are different types of clustering algorithms that handle all kinds of unique data

When to use Clustering

- When you have a set of unlabeled data
- Type of dataset
- when you're trying to do anomaly detection to try and find outliers in your data
- If you aren't sure of what features to use for your machine learning model, clustering discovers patterns you can use to figure out what stands out in the data.
- Clustering is especially useful for exploring data you know nothing about.

1) K Means

K-Means Clustering is an Unsupervised Learning algorithm, which groups the unlabeled dataset into different clusters. Here K defines the number of pre-defined clusters that need to be created in the process, as if K=2, there will be two clusters, and for K=3, there will be three clusters, and so on. It is an iterative algorithm that divides the unlabeled dataset into k different clusters in such a way that each dataset belongs only one group that has similar properties. It is a centroid-based algorithm, where each cluster is associated with a centroid. The main aim of this algorithm is to minimize the sum of distances between the data point and their corresponding clusters.

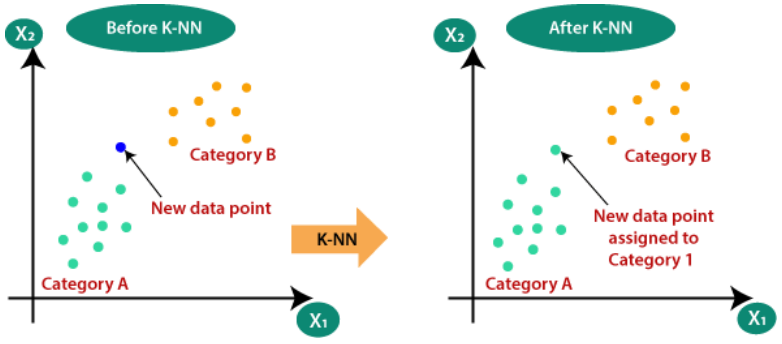


Working of K Means
Step-1: Select the number K to decide the number of clusters.
Step-2: Select random K points or centroids. (It can be other from the input dataset).
Step-3: Assign each data point to their closest centroid, which will form the predefined K clusters.
Step-4: Calculate the variance and place a new centroid of each cluster.
Step-5: Repeat the third steps, which means re-assign each data point to the new closest centroid of each cluster.
Step-6: If any reassignment occurs, then go to step-4 else go to FINISH.
Step-7: The model is ready.

2) KNN

K-Nearest Neighbor is one of the simplest Machine Learning algorithms based on Supervised Learning technique. K-NN algorithm assumes the similarity between the new case/data and available cases and put the new case into the category that is most similar to the available categories. K-NN algorithm stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily classified into a well suite category by using K- NN algorithm. K-NN algorithm can be used for Regression as well as for Classification but mostly it is used for the Classification problems. K-NN is a non-parametric algorithm, which means it does not make any assumption on underlying data. It is also called a lazy learner algorithm because it does not learn from the training set immediately instead it stores the dataset and at the time of classification, it performs an action on the dataset. KNN algorithm at the training phase just stores the dataset and when it gets new data, then it classifies that data into a category that is much similar to the new data.

The K-NN working can be explained on the basis of the below algorithm:



Step-1: Select the number K of the neighbors
Step-2: Calculate the Euclidean distance of **K number of neighbors**
Step-3: Take the K nearest neighbors as per the calculated Euclidean distance.
Step-4: Among these k neighbors, count the number of the data points in each category.
Step-5: Assign the new data points to that category for which the number of the neighbor is maximum.
Step-6: Our model is ready.

Algorithm:

K-Mean Clustering

Step 1: Load Dataset
Import the dataset containing the features 'Age', 'Annual Income' and 'Spending Score', and display the first few records to understand the data structure.

Step 2: Feature Selection
Select 'Annual Income' and 'Spending Score' as the features for clustering analysis.

Step 3: Data Visualization
Create a scatter plot of 'Annual Income' vs 'Spending Score' to visualize the distribution of the data and observe any patterns or groupings.

Step 4: Initialize K-Means Model
Set up a K-Means clustering model with 3 clusters and fix the random state for reproducibility of results.

Step 5: Fit Model
Train the K-Means model on the selected features ('Annual Income' and 'Spending Score') to identify natural clusters in the data.

Step 6: Extract Centroids and Labels
Obtain the centroids (cluster centers) and labels (which cluster each data point belongs to) after fitting the model.

Step 7: Cluster Visualization
Plot the data points, colored by their cluster labels, and mark the centroids with distinct markers to visualize the clustering result.

Code and Output:

Dataset: https://drive.google.com/file/d/1pMsLRbrcEhAG5f2qfFOO0NW7zsr16Nut/view?usp=sharing

```
# LAB - 4 : K-Mean Clustring

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
```

```
df = pd.read_csv("Mall_Customers.csv")
df.head()
```

	CustomerID	Genre	Age	Annual Income (k$)	Spending Score (1-100)
0	1	Male	19	15	39
1	2	Male	21	15	81
2	3	Female	20	16	6
3	4	Female	23	16	77
4	5	Female	31	17	40

```
df.columns
```

```
Index(['CustomerID', 'Genre', 'Age', 'Annual Income (k$)',
       'Spending Score (1-100)'],
      dtype='object')
```

```
df.info
```

```
<bound method DataFrame.info of      CustomerID   Genre  Age  Annual Income (k$)  Spending Score (1-100)
0             1    Male   19                  15                      39
1             2    Male   21                  15                      81
2             3  Female   20                  16                       6
3             4  Female   23                  16                      77
4             5  Female   31                  17                      40
..          ...     ...  ...                 ...                     ...
195         196  Female   35                 120                      79
196         197  Female   45                 126                      28
197         198    Male   32                 126                      74
198         199    Male   32                 137                      18
199         200    Male   30                 137                      83

[200 rows x 5 columns]>
```

```
df.isnull().sum()
```

```
CustomerID              0
Genre                   0
Age                     0
Annual Income (k$)      0
Spending Score (1-100)  0
dtype: int64
```

```
X= df[['Annual Income (k$)', 'Spending Score (1-100)']].values
print('Customers Data:')
print(X)
```

```
Customers Data:
[[ 15  39]
 [ 15  81]
 [ 16   6]
 [ 16  77]
 [ 17  40]
 [ 17  76]
 [ 18   6]
 [ 18  94]
```

```
plt.scatter(X[:,0], X[:,1],c='black')
plt.title('Annual Income vs Spending Score')
plt.xlabel('Annual Income')
plt.ylabel('Spending Score')
plt.show()
```


```
# perform Kmeans clustering
kmeans = KMeans(n_clusters=3, random_state=42)
kmeans.fit(X)
```

```
KMeans(n_clusters=3, random_state=42)
```

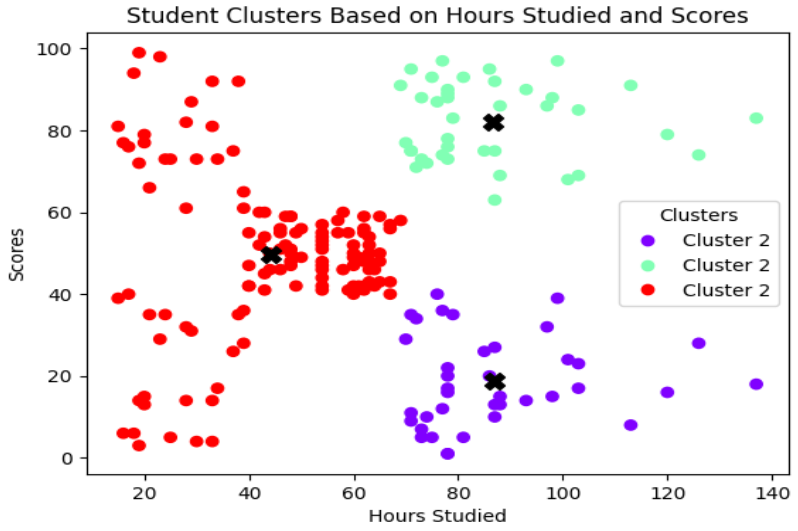
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.

On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

KMeans?Documentation for KMeansiFitted

```
KMeans(n_clusters=3, random_state=42)
```

```
scatter =  plt.scatter (X[:, 0], X[:, 1], c= labels, cmap='rainbow') # Clustered, data points

plt.scatter(centroids [:, 0], centroids[:, 1], c='black', marker= 'X',s=100)

plt.title("Student Clusters Based on Hours Studied and Scores") # Plot title

plt.xlabel("Hours Studied") # X-axis label

plt.ylabel("Scores") # Y-axis label

#Create a legend


legend_labels = [f'Cluster {1+1}' for i in range(len(centroids))]
legend = plt.legend(handles=scatter.legend_elements() [0], labels = legend_labels, title="Clusters")

plt.show()
```



K-Nearest Neighbors (KNN)

KNN Classification for Unknown Fruit Prediction

- **Step 1: Load Dataset**

Import the dataset containing fruit features like 'Fruit Type', 'Weight', 'Diameter' and 'Color'. Display the dataset to understand its structure.

- **Step 2: Feature and Label Selection**

Select 'Fruit Weight', 'Color' and 'Diameter' as features (X) and 'Fruit Type' as the label (y) for classification.

- **Step 3: Define Unknown Samples**

Define unknown fruit samples (e.g., fruit with weight 160g, color 0 (Red)) to predict the fruit type using the trained KNN model.

- **Step 4: Initialize KNN Model**

Create a K-Nearest Neighbors (KNN) model with 3 neighbors (n_neighbors=3) for classification.

- **Step 5: Train Model**

Train the KNN model on the dataset using the selected features (X) and labels (y) to learn the classification patterns.

- **Step 6: Predict Fruit**

Types for Unknown Samples Use the trained KNN model to predict the fruit type for the unknown samples based on their features.

- **Step 7: Data Visualization**

Plot the known fruit samples on a scatter plot, with each fruit type represented in a different color.

Mark the unknown fruits with distinct markers to visualize their predicted class.

Code and Output:

Dataset: https://drive.google.com/file/d/15xf4rjKgsCcSJuMNVPZJIg99sewhxIHW/view?usp=sharing

```
# LAB - 5 : K- Nearest Neighbours(KNN)

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
```

```
df = pd.read_csv('fruit_classification.csv')
```

```
df.head()
```

	Weight	Color	Diameter	Fruit_Type
0	150	0	7.5	apple
1	160	0	8.0	apple
2	170	0	7.8	apple
3	180	0	7.7	apple
4	155	0	7.6	apple

```
# Separate features (Weight, Color, Diameter) and labels (Fruit_Type)
X = df [['Weight', 'Color', 'Diameter']].values
y = df ['Fruit_Type'].values
```

```
print("Fruit Data (Features):")
print(X)
print("Fruit Type (Labels):")
print(y)
```

```
Fruit Data (Features):
[[150.    0.    7.5]
 [160.    0.    8. ]
 [170.    0.    7.8]
 [180.    0.    7.7]
```

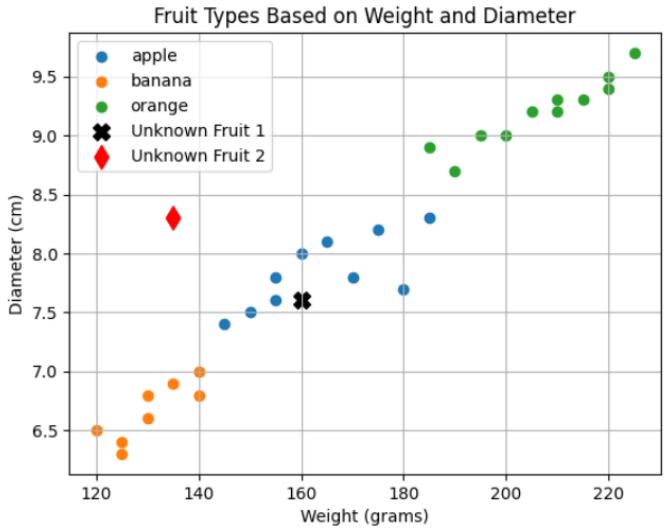
```
# Unknown fruit samples to predict (random example values)
unknown_fruit  = np.array([[160, 0, 7.6]]) # Example: Weight 160g, Color-Red (0), Diameter-7.6 cm
unknown_fruit1 =  np.array([[135, 2, 8.3]]) # Example: Weight-135g,u Color-Orange (2), Diameter-8.3 cm
```

```
# Create KNN Classifier
knn = KNeighborsClassifier(n_neighbors=3)

# Train the Model
knn.fit(X, y)
```

```
# Make Predictions for unknown fruits
prediction  =  knn.predict (unknown_fruit)
prediction1 =  knn.predict (unknown_fruit1)

print (f"The predicted fruit type for the unknown fruit is: {prediction[0]}")
print (f"The predicted fruit type for the second unknown fruit is: {prediction1[0]}")
```

```
The predicted fruit type for the unknown fruit is: apple
The predicted fruit type for the second unknown fruit is: banana
```

```
for fruit_type in np.unique(y):
    plt.scatter (X[y==fruit_type, 0], X[y==fruit_type, 2], label=fruit_type)

#Plot the unknown fruits with a different marker style and color
plt.scatter (unknown_fruit[0] [0], unknown_fruit [0] [2], label='Unknown Fruit 1', c='black', marker='X', s=100)
plt.scatter (unknown_fruit1 [0] [0], unknown_fruit1 [0] [2], label='Unknown Fruit 2', c='red', marker= 'd', s=100)
#Labels and title
plt.title("Fruit Types Based on Weight and Diameter")
plt.xlabel("Weight (grams)")
plt.ylabel("Diameter (cm)")
plt.legend()
plt.grid(True) # Optional: Show grid for better readability
plt.show()
```



Conclusion:

This lab highlighted the complementary roles of K-Nearest Neighbors (KNN) and K-Means clustering for data classification and grouping. **KNN** effectively classified Annual Income vs Spending Score based on labeled data, using the three nearest neighbors (K=3) to predict an unknown fruit's category, which it identified as "Apple." This supervised method depends on labeled datasets and careful K-value selection to enhance accuracy. In contrast, **K-Means clustering** revealed natural groupings within the dataset without labels by clustering fruits into three groups based on "Wight" and "Diameter." While KNN relies on known categories, K-Means' unsupervised approach clusters data by inherent similarities. Both algorithms illustrated different ways to extract insights, with KNN supporting precise classification and K-Means enabling pattern discovery, thus showcasing their value in exploratory and predictive data analysis.

Practical 06

Student Name: YASH SURYARAO
Date of Experiment: 11-10-2024
Date of Submission: 08-11-2024
PRN No: 20240804062

Aim: Implement PCA on any particular data.

Objectives: Understanding PCA

Software/Tool: Python/Jupyter/Anaconda/Colab

Theory:

Dimensionality Reduction

It is commonly used in the fields that deal with high-dimensional data, such as speech recognition, signal processing, bioinformatics, etc. It can also be used for data visualization, noise reduction, cluster analysis, etc. Handling the high-dimensional data is very difficult in practice, commonly known as the curse of dimensionality. If the dimensionality of the input dataset increases, any machine learning algorithm and model becomes more complex. As the number of features increases, the number of samples also gets increased proportionally, and the chance of overfitting also increases and results in poor performance.

Some of the techniques

Linear:

∗Principal components analysis

(PCA)Non-linear:

∗ Kernel PCA

∗ Independent components analysis

∗ Self-organizing maps

∗ Multidimensional scaling

∗ Auto encoders

Principle component Analysis

It is a statistical process that converts the observations of correlated features into a set of linearly uncorrelated features with the help of orthogonal transformation. These new transformed features are called the Principal Components. It is a technique to draw strong patterns from the given dataset by reducing the variances. PCA generally tries to find the lower-dimensional surface to project the high-dimensional data.

Steps for PCA

1) Getting the data set
2) Representing data into a structure
3) Standardizing the data
4)Calculating the Covariance of matrix Z
5) Calculating the Eigenvalues and Eigenvectors
6) Sorting the Eigenvectors
7)Calculating the new features or Principal Components
8)Remove less or unimportant features from the new dataset

Algorithm:

Step 1: Import Libraries
Import necessary libraries such as **pandas**, **sklearn.decomposition.PCA**, **sklearn.preprocessing.StandardScaler**, and **matplotlib.pyplot** for data manipulation, PCA, data standardization, and visualization.

Step 2: Load the Dataset
Read the dataset from a CSV file 'employee.csv' into a panda DataFrame and display the first few rows to examine the data structure.

Step 3: Preprocess the Data
Drop any columns that are not relevant for PCA analysis the 'Employee Name' column.

Step 4: Standardize the Data
Use **StandardScaler** to standardize the dataset, ensuring that each feature has a mean of 0 and a standard deviation of 1. This is important for PCA, as it is sensitive to the scale of the data.

Step 5: Apply PCA
Initialize the PCA model to reduce the dataset to two principal components, and apply it to the standardized data using the fit_transform() method.

Step 6: Create PCA DataFrame
Create a new DataFrame containing the two principal components for further analysis and visualization.

Step 7: Visualize PCA Results
Plot the two principal components on a scatter plot to visualize the data distribution and any potential clusters.

Step 8: Analyze Explained Variance
Print the explained variance ratio for each principal component to understand how much of the total variance in the dataset is captured by each component.

Code and Output:

Dataset: https://drive.google.com/file/d/1NCMK6Sde-rfsyUX3qCc_A1HKVw-lsRcY/view?usp=sharing

```
# Impoerint required packages

import pandas as pd
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
```

```
#New dataset: Employee Job Satisfaction
data = pd.read_csv("employee.csv")

#Create DataFrame
df = pd.DataFrame(data)
df
```

	Employee Name	Job Satisfaction	Work-Life Balance	Salary Satisfaction	Years at Company	Stay/Leave
0	John	8	7	8	2	1
1	Alice	6	6	5	4	0
2	Bob	7	6	6	3	0
3	Carol	9	9	9	5	1
4	Dave	5	5	4	1	0

```
# Standardize the Features and target variable
X = df[['Job Satisfaction', 'Work-Life Balance', 'Salary Satisfaction', 'Years at Company']]
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
print("Standardized Data:\n",X_scaled)
```

```
Standardized Data:
 [[ 0.70710678  0.29488391  0.86266219 -0.70710678]
 [-0.70710678 -0.44232587 -0.75482941  0.70710678]
 [ 0.         -0.44232587 -0.21566555  0.        ]
 [ 1.41421356  1.76930347  1.40182605  1.41421356]
 [-1.41421356 -1.17953565 -1.29399328 -1.41421356]]
```

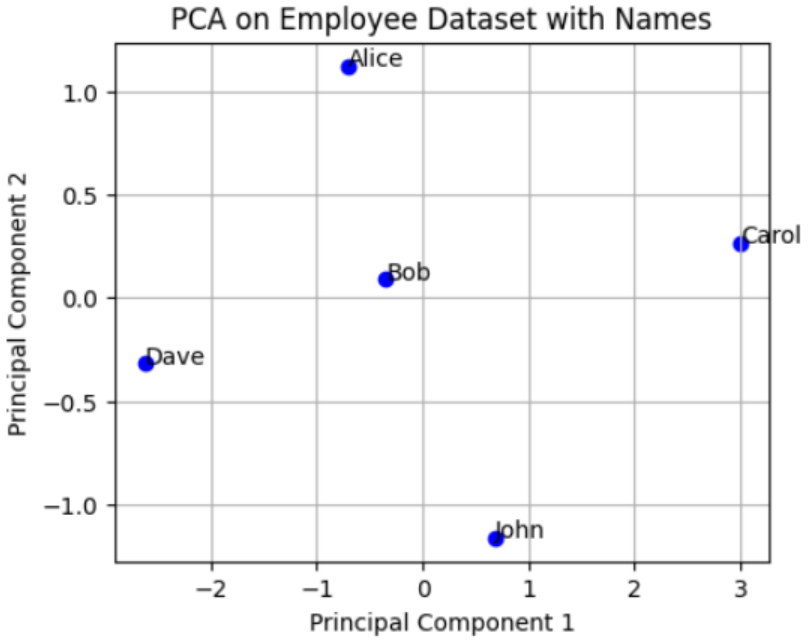
```
# Applying PCA
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)
print("PCA Transformed Data:\n", X_pca)
```

```
PCA Transformed Data:
 [[ 0.68719568 -1.16083198]
 [-0.71005956  1.12061302]
 [-0.3481751   0.08986206]
 [ 2.99718228  0.26573929]
 [-2.62614331 -0.31538239]]
```

```
# Convert to DataFrame

df_pca = pd.DataFrame(data=X_pca, columns=['PC1', 'PC2'])
df_pca['Employee Name' ] = df['Employee Name' ]
print(df_pca)
```

```
        PC1       PC2 Employee Name
0  0.687196 -1.160832          John
1 -0.710060  1.120613         Alice
2 -0.348175  0.089862           Bob
3  2.997182  0.265739         Carol
4 -2.626143 -0.315382          Dave
```

```
# Visualize the PCA results

plt.figure(figsize=(5, 4))
plt.scatter(df_pca['PC1'], df_pca['PC2'], color='blue')
for i, name in enumerate(df_pca['Employee Name']):
    plt.annotate(name, (df_pca['PC1'][i], df_pca['PC2'][i]))

plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.title('PCA on Employee Dataset with Names')
plt.grid()
plt.show()
```



Conclusion:

This lab focused on implementing **Principal Component Analysis (PCA)** to reduce the dimensionality of a dataset while preserving key features for analysis. The dataset selected covered aspects of employee satisfaction, including job satisfaction, work-life balance, salary satisfaction, and years at the company, along with an indication of whether each employee intended to stay or leave. After standardizing the features to ensure all variables contributed equally to the analysis, PCA was applied to reduce the data from four dimensions to two primary components.

The PCA transformation allowed the data to be visualized in two dimensions, effectively capturing significant variance in the dataset. This transformation highlighted how the different features contribute to variations among employees and provided a simplified view of the dataset, making patterns in employee satisfaction and retention more evident. The scatter plot of the principal components further illustrated these insights, offering a more intuitive way to observe relationships between employees based on satisfaction metrics.

In summary, this lab demonstrated how PCA is a powerful tool in simplifying complex datasets, revealing underlying patterns, and facilitating easier visualization without substantial loss of information. PCA proved to be useful for dimensionality reduction, making it easier to analyze and interpret the dataset, which is valuable in various real-world applications where data dimensionality is high.

Practical 07

Student Name: YASH SURYARAO
Date of Experiment: 08-11-2024
Date of Submission: 13-11-2024
PRN No: 20240804062

Aim: Implement Random Forest model that predicts loan application should be approved or not based on prior information about the applicant.

Objectives:

1. **Analyze and data Pre-processing**: To clean and prepare the dataset by handling missing values and encoding categorical features, ensuring compatibility with the Random Forest model.
2. **Feature Analysis**: To understand the relationships between features and identify the most significant factors affecting loan approval.
3. **Model Training**: To train a Random Forest classifier on the dataset and optimize it for the highest accuracy in predicting loan approval.
4. **Evaluation**: Evaluate the model's performance and interpret the results to understand the key factors influencing loan approval decisions.
5. **Feature Importance Analysis**: To analyze feature importance within the Random Forest model to determine which factors most influence loan approval decisions.

Software/Tool: Python/Jupyter/Anaconda/Colab

Theory:

Loan approval prediction is a binary classification problem where the target variable (Loan_Status) indicates whether an applicant's loan can be approved. Machine learning algorithms, such as Linear Regression (although typically used for regression, can be tested for binary outcomes here) and Random Forest (a robust ensemble method for classification tasks), can help predict loan approval based on input features. Linear Regression estimates continuous values but, when applied to binary targets, serves as a baseline. Random Forest uses multiple decision trees to capture complex relationships in the data and is more suited for this binary classification.**Random Forest Algorithm**: The Random Forest is an ensemble learning technique that builds multiple decision trees during training and merges them to produce more accurate and stable predictions. Each tree is trained on a random subset of the data, making the model less prone to overfitting.

- **Feature Importance**: The Random Forest model provides a measure of feature importance, helping us identify the most influential features in predicting loan approval.
- **Evaluation Metrics**:
  - **Accuracy**: Proportion of correct predictions out of total predictions.
  - **Classification Report**: Provides precision, recall, and F1-score, offering insights into the model's performance for each class.
  - **Confusion Matrix**: Displays the number of true positive, true negative, false positive, and false negative predictions.

- **Training and Testing Analysis:** The Random Forest model was trained over 20 epochs, and the model's performance on training and test sets was plotted to evaluate the fit:
  - **Graph Analysis**: The plot shows training and testing accuracy over epochs, providing insight into model fit.
  - **Overfitting/Underfitting Analysis**:

- If training accuracy remains high while test accuracy is significantly lower, this indicates overfitting.
- If both training and testing accuracies are low, underfitting may be present.
- Here, accuracies stabilize with moderate test scores, suggesting a **good fit** but possibly limited by data variability.

Algorithm:

1. **Import Necessary Libraries:**
   - Import libraries like **pandas**, **numpy**, and **matplotlib** for data handling and visualization.
   - Import machine learning modules from **sklearn**, such as **train_test_split**, **LinearRegression**, **RandomForestClassifier**, and **evaluation metrics**.

2. **Load and Inspect the Dataset:**
   - Load the dataset **LoanApprovalPrediction.csv** using **pd.read_csv()**.
   - Display the first and last five records using **df.head()** and **df.tail()** to understand the data structure.
   - Check the shape and data types with **df.shape** and **df.info()**.

3. **Check for Missing Values:**
   - Identify missing values with **df.isnull().sum().**
   - Display the descriptive statistics using **df.describe()** to understand numerical feature distributions.

4. **Impute Missing Values:**
   - Use **forward-fill** and **backward-fill** to impute missing values for columns like **LoanAmount**, **Loan_Amount_Term**, **Credit_History**, and **Dependents**.
   - Verify that there are no remaining missing values with **df.isnull().sum().**

5. **Remove Unwanted Columns:**
   - Drop irrelevant columns, such as **Dependents** and **Education**, which may not directly contribute to loan approval prediction.

6. **Convert Categorical Variables:**
   - Separate numerical and categorical columns using **select_dtypes().**
   - Apply label encoding to convert categorical values into numerical format, making them suitable for machine learning models.
   - Encode the target variable **Loan_Status** as **0** and **1**.

7. **Feature Selection:**
   - Select the features (**ApplicantIncome**, **LoanAmount**, and **Loan_Amount_Term**) as predictors **(X)** and **Loan_Status** as the target variable **(y)**.

8. **Split the Dataset:**
   - Split the data into training and test sets **(80% training, 20% testing)** using **train_test_split**.

9. **Train the Linear Regression Model:**
   - Initialize the **Linear Regression model** and train it on the training set.
   - Make predictions on the test set.
   - Calculate performance metrics for **Linear Regression**, including **Mean Squared Error (MSE)** and **R-squared score (R^2),** to assess model performance.

10. **Train the Random Forest Classifier:**
    - Initialize the Random Forest Classifier with a specific **random_state**.
    - Train the model on the training set and make predictions on the test set.

11. **Evaluate the Random Forest Model:**

- Calculate model accuracy using **accuracy_score**.
- Generate a classification report (**precision**, **recall**, **F1-score**) and a confusion matrix to understand the model's performance on approved and rejected loans.
- Display results, including the model accuracy, classification report, and confusion matrix.

12. Visualize Correlation Analysis:

- Compute the correlation matrix of numeric features.
- Plot a heatmap of the correlation matrix to visualize relationships between features and with **Loan_Status**.

13. Track Model Performance over Epochs (Random Forest):

- Define a number of epochs (20 in this case) to train the **Random Forest model** multiple times.
- For each **epoch**, initialize the **Random Forest** with a unique random state, train it on the training data, and record **training** and **test accuracy**.
- Append accuracies to track how model performance varies across **epochs**.

14. Plot Training and Testing Accuracy vs. Epochs:

- Plot the recorded training and test accuracies over the epochs to visualize model performance stability.
- Add labels, legend, and grid to enhance interpretability.

Code and Output:

Dataset: https://drive.google.com/file/d/1-MFBp1zaVVRKeeAEWwDLxG8Je368xP2N/view?usp=sharing

```
# Importing Necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score,classification_report, confusion_matrix
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.impute import SimpleImputer
```

```
# Loading the Dataset

df=pd.read_csv("LoanApprovalPrediction.csv")
```

```
# Display top 5 records from dataframe

df.head()
```

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

```
# Display bottom 5 records from dataframe

df.tail()
```

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_Area	Loan_Status
593	LP002978	Female	No	0.0	Graduate	No	2900	0.0	71.0	360.0	1.0	Rural	Y
594	LP002979	Male	Yes	3.0	Graduate	No	4106	0.0	40.0	180.0	1.0	Rural	Y
595	LP002983	Male	Yes	1.0	Graduate	No	8072	240.0	253.0	360.0	1.0	Urban	Y
596	LP002984	Male	Yes	2.0	Graduate	No	7583	0.0	187.0	360.0	1.0	Urban	Y
597	LP002990	Female	No	0.0	Graduate	Yes	4583	0.0	133.0	360.0	0.0	Semiurban	N

```
# Check the shape of the dataframe
df.shape
```

```
(598, 13)
```

```
# Get the data types

df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 598 entries, 0 to 597
Data columns (total 13 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Loan_ID            598 non-null    object
 1   Gender             598 non-null    object
 2   Married            598 non-null    object
 3   Dependents         586 non-null    float64
 4   Education          598 non-null    object
 5   Self_Employed      598 non-null    object
 6   ApplicantIncome    598 non-null    int64
 7   CoapplicantIncome  598 non-null    float64
 8   LoanAmount         577 non-null    float64
 9   Loan_Amount_Term   584 non-null    float64
 10  Credit_History     549 non-null    float64
 11  Property_Area      598 non-null    object
 12  Loan_Status        598 non-null    object
dtypes: float64(5), int64(1), object(7)
memory usage: 60.9+ KB
```

```
# Check for missing Values
df.isnull()
```

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_Area	Loan_Status
0	False	False	False	False	False	False	False	False	True	False	False	False	False
1	False	False	False	False	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...	...	...	...	...	...	...
593	False	False	False	False	False	False	False	False	False	False	False	False	False
594	False	False	False	False	False	False	False	False	False	False	False	False	False
595	False	False	False	False	False	False	False	False	False	False	False	False	False
596	False	False	False	False	False	False	False	False	False	False	False	False	False
597	False	False	False	False	False	False	False	False	False	False	False	False	False

598 rows × 13 columns

```
# check the statistics of dataset

df.describe()
```

	Dependents	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History
count	586.000000	598.000000	598.000000	577.000000	584.000000	549.000000
mean	0.755973	5292.252508	1631.499866	144.968804	341.917808	0.843352
std	1.007751	5807.265364	2953.315785	82.704182	65.205994	0.363800
min	0.000000	150.000000	0.000000	9.000000	12.000000	0.000000
25%	0.000000	2877.500000	0.000000	100.000000	360.000000	1.000000
50%	0.000000	3806.000000	1211.500000	127.000000	360.000000	1.000000
75%	1.750000	5746.000000	2324.000000	167.000000	360.000000	1.000000
max	3.000000	81000.000000	41667.000000	650.000000	480.000000	1.000000

```
# Get the data types

df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 598 entries, 0 to 597
Data columns (total 13 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Loan_ID            598 non-null    object
 1   Gender             598 non-null    object
 2   Married            598 non-null    object
 3   Dependents         586 non-null    float64
 4   Education          598 non-null    object
 5   Self_Employed      598 non-null    object
 6   ApplicantIncome    598 non-null    int64
 7   CoapplicantIncome  598 non-null    float64
 8   LoanAmount         577 non-null    float64
 9   Loan_Amount_Term   584 non-null    float64
 10  Credit_History     549 non-null    float64
 11  Property_Area      598 non-null    object
 12  Loan_Status        598 non-null    object
dtypes: float64(5), int64(1), object(7)
memory usage: 60.9+ KB
```

```
# Forward and backward fill missing values in moving averages in columns 'LoanAmount', 'Loan_Amount_Term' and 'Credit_History'
df['LoanAmount'] = df['LoanAmount'].ffill().bfill()
df['Loan_Amount_Term'] = df['Loan_Amount_Term'].ffill().bfill()
df['Credit_History'] = df['Credit_History'].ffill().bfill()
df['Dependents'] = df['Dependents'].ffill().bfill()
df
```

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_Area	Loan_Status
0	LP001002	Male	No	0.0	Graduate	No	5849	0.0	128.0	360.0	1.0	Urban	Y
1	LP001003	Male	Yes	1.0	Graduate	No	4583	1508.0	128.0	360.0	1.0	Rural	N
2	LP001005	Male	Yes	0.0	Graduate	Yes	3000	0.0	66.0	360.0	1.0	Urban	Y
3	LP001006	Male	Yes	0.0	Not Graduate	No	2583	2358.0	120.0	360.0	1.0	Urban	Y
4	LP001008	Male	No	0.0	Graduate	No	6000	0.0	141.0	360.0	1.0	Urban	Y
...	...	...	...	...	...	...	...	...	...	...	...	...	...
593	LP002978	Female	No	0.0	Graduate	No	2900	0.0	71.0	360.0	1.0	Rural	Y
594	LP002979	Male	Yes	3.0	Graduate	No	4106	0.0	40.0	180.0	1.0	Rural	Y
595	LP002983	Male	Yes	1.0	Graduate	No	8072	240.0	253.0	360.0	1.0	Urban	Y
596	LP002984	Male	Yes	2.0	Graduate	No	7583	0.0	187.0	360.0	1.0	Urban	Y
597	LP002990	Female	No	0.0	Graduate	Yes	4583	0.0	133.0	360.0	0.0	Semiurban	N

```
df.isnull().sum()
```

```
Loan_ID              0
Gender               0
Married              0
Dependents           0
Education            0
Self_Employed        0
ApplicantIncome      0
CoapplicantIncome    0
LoanAmount           0
Loan_Amount_Term     0
Credit_History       0
Property_Area        0
Loan_Status          0
dtype: int64
```

```
# checking duplicates values in dataframe

print("Number of Duplicated Rows:")
df.duplicated().sum()
```

```
Number of Duplicated Rows:
0
```

```
#drop unwanted column
df = df.drop(columns=['Dependents', 'Education'], errors='ignore')
df
```

	Loan_ID	Gender	Married	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_Area	Loan_Status
0	LP001002	Male	No	No	5849	0.0	128.0	360.0	1.0	Urban	Y
1	LP001003	Male	Yes	No	4583	1508.0	128.0	360.0	1.0	Rural	N
2	LP001005	Male	Yes	Yes	3000	0.0	66.0	360.0	1.0	Urban	Y
3	LP001006	Male	Yes	No	2583	2358.0	120.0	360.0	1.0	Urban	Y
4	LP001008	Male	No	No	6000	0.0	141.0	360.0	1.0	Urban	Y
...	...	...	...	...	...	...	...	...	...	...	...
593	LP002978	Female	No	No	2900	0.0	71.0	360.0	1.0	Rural	Y
594	LP002979	Male	Yes	No	4106	0.0	40.0	180.0	1.0	Rural	Y
595	LP002983	Male	Yes	No	8072	240.0	253.0	360.0	1.0	Urban	Y
596	LP002984	Male	Yes	No	7583	0.0	187.0	360.0	1.0	Urban	Y
597	LP002990	Female	No	Yes	4583	0.0	133.0	360.0	0.0	Semiurban	N

```
# Converting String values into 0 and 1

# Separate numerical and categorical columns
numerical_cols = df.select_dtypes(include=['float64', 'int64']).columns
categorical_cols = df.select_dtypes(include=['object']).columns.drop('Loan_Status')

# Handle missing values: use median for numerical and mode for categorical
imputer_num = SimpleImputer(strategy='median')
df[numerical_cols] = imputer_num.fit_transform(df[numerical_cols])

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

# Encode categorical features and target variable
encoder = LabelEncoder()
df['Loan_Status'] = encoder.fit_transform(df['Loan_Status'])
df[categorical_cols] = df[categorical_cols].apply(encoder.fit_transform)
df
```

	Loan_ID	Gender	Married	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_Area	Loan_Status
0	0	1	0	0	5849.0	0.0	128.0	360.0	1.0	2	1
1	1	1	1	0	4583.0	1508.0	128.0	360.0	1.0	0	0
2	2	1	1	1	3000.0	0.0	66.0	360.0	1.0	2	1
3	3	1	1	0	2583.0	2358.0	120.0	360.0	1.0	2	1
4	4	1	0	0	6000.0	0.0	141.0	360.0	1.0	2	1
...	...	...	...	...	...	...	...	...	...	...	...
593	593	0	0	0	2900.0	0.0	71.0	360.0	1.0	0	1
594	594	1	1	0	4106.0	0.0	40.0	180.0	1.0	0	1
595	595	1	1	0	8072.0	240.0	253.0	360.0	1.0	2	1
596	596	1	1	0	7583.0	0.0	187.0	360.0	1.0	2	1
597	597	0	0	1	4583.0	0.0	133.0	360.0	0.0	1	0

```
# Define the features (X) and target (y)

X = df[['ApplicantIncome', 'LoanAmount', 'Loan_Amount_Term']]
y = df['Loan_Status']
```

```
# Split the dataset into training and testing sets (80-20 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

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

```
LinearRegression
LinearRegression()
```

```
# Make predictions on the test set
y_pred = model.predict(X_test)
# Calculate and print performance metrics
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print("Mean Squared Error:", mse)
print("R-squared Score:", r2)
```

```
Mean Squared Error: 0.20881210732340302
R-squared Score: -0.010720788388908442
```

```
## Initialize and train the Random Forest model

rf_model = RandomForestClassifier(random_state=42)
rf_model.fit(X_train, y_train)
```

RandomForestClassifier

RandomForestClassifier(random_state=42)

```
# Make predictions on the test data
y_pred = rf_model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
classification_rep = classification_report(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)

# Output the results
print(f"Model Accuracy: {accuracy:.2f}")
print("\nClassification Report:\n", classification_rep)
print("\nConfusion Matrix:\n", conf_matrix)
```

```
Model Accuracy: 0.68

Classification Report:
               precision    recall  f1-score   support

           0       0.43      0.29      0.34        35
           1       0.74      0.85      0.79        85

    accuracy                           0.68       120
   macro avg       0.59      0.57      0.57       120
weighted avg       0.65      0.68      0.66       120


Confusion Matrix:
 [[10 25]
 [13 72]]
```

```
# Calculate evaluation metrics
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"Mean Squared Error (MSE): {mse}")
print(f"R-squared (R2): {r2}")
```

```
Mean Squared Error (MSE): 0.31666666666666665
R-squared (R2): -0.5327731092436971
```

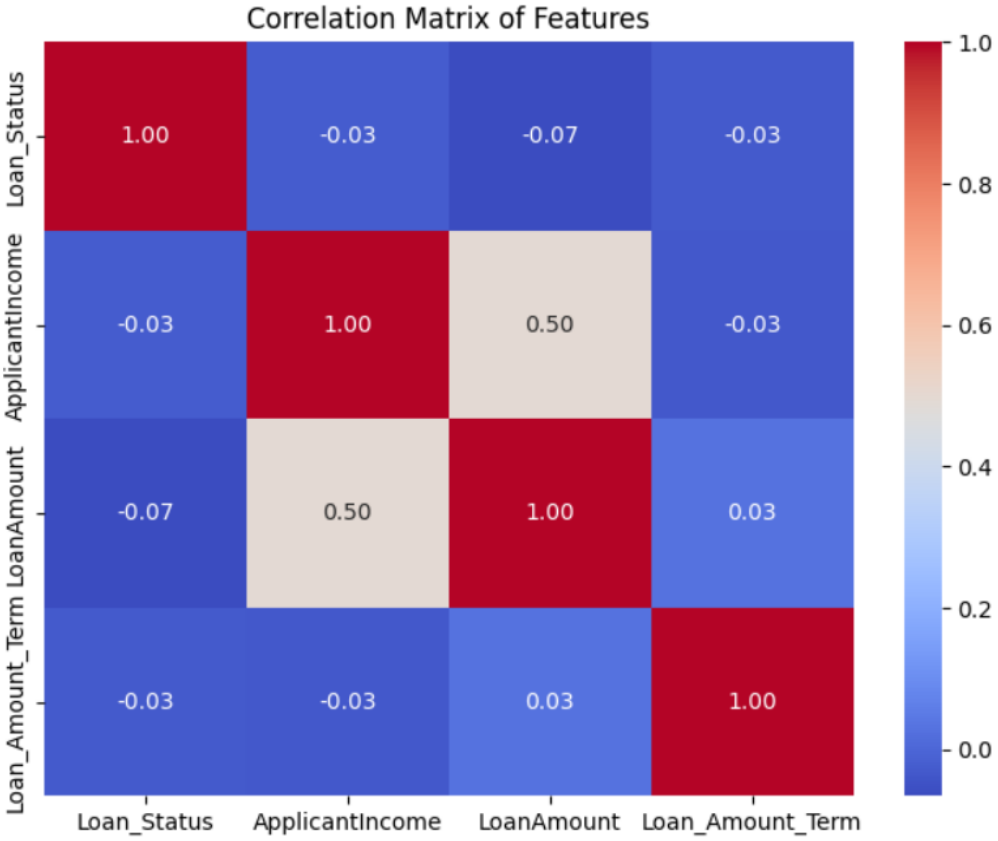
```
predictions_df = pd.DataFrame({'Actual Loan Amount': y_test, 'Predicted Loan Amount': y_pred})
print(predictions_df.head())
```

```
     Actual Loan Amount  Predicted Loan Amount
110                   0                      0
287                   0                      0
563                   1                      1
77                    0                      1
181                   1                      0
```

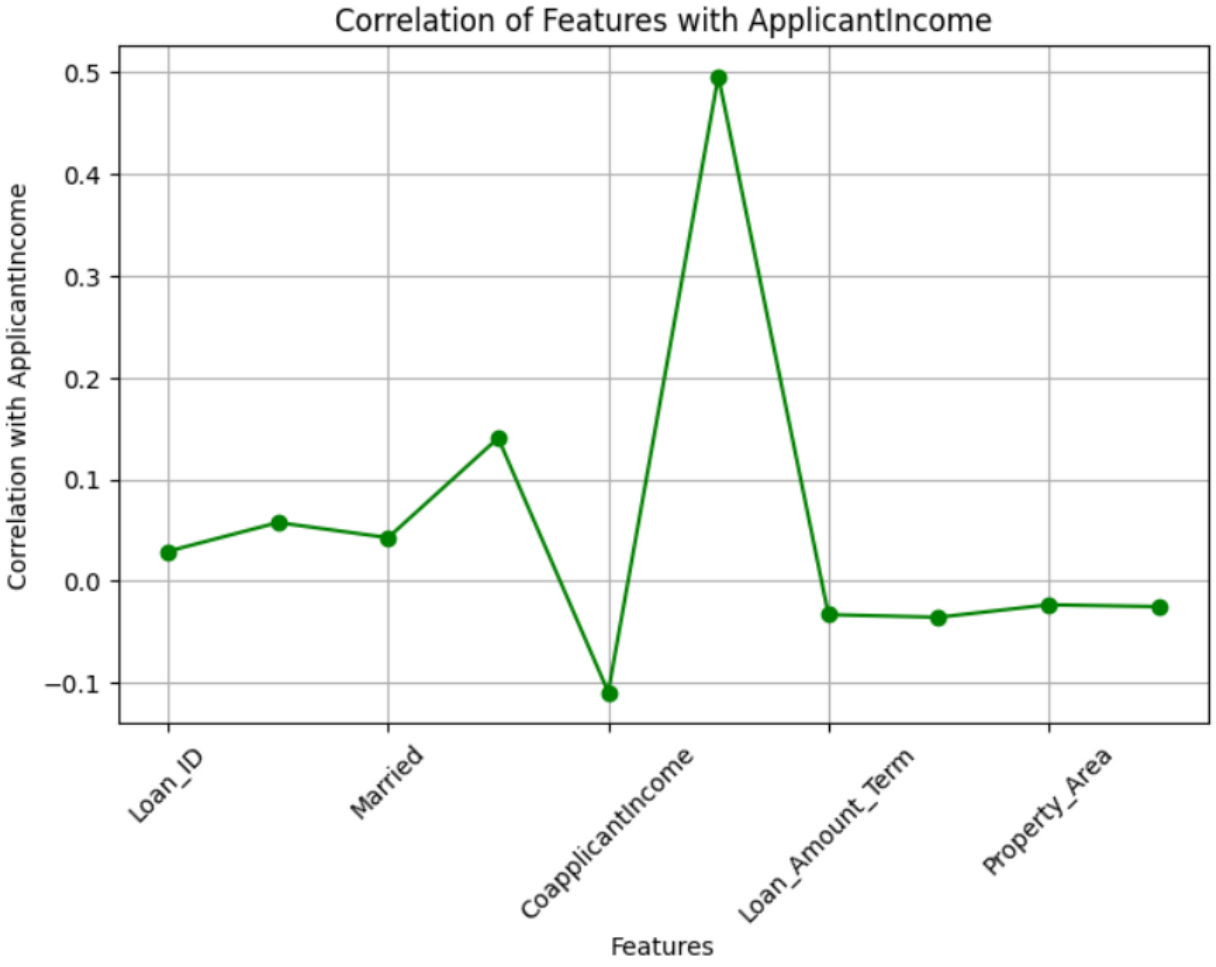
```python
# CORRELATION ANALYSIS

import seaborn as sns

# Calculate correlation matrix for numeric columns
correlation = df.select_dtypes(include=[float, int]).corr()
print(correlation) # Print correlation matrix
```

```
                   Loan_ID    Gender   Married  Self_Employed  \
Loan_ID           1.000000 -0.028765 -0.006149       0.012701
Gender           -0.028765  1.000000  0.369612      -0.028663
Married          -0.006149  0.369612  1.000000      -0.021441
Self_Employed     0.012701 -0.028663 -0.021441       1.000000
ApplicantIncome   0.029095  0.057386  0.042487       0.140406
CoapplicantIncome 0.042334  0.083080  0.073830       0.020877
LoanAmount        0.062203  0.103224  0.137689       0.098982
Loan_Amount_Term -0.023908 -0.087020 -0.104990      -0.038447
Credit_History   -0.014498 -0.002215  0.015003       0.011972
Property_Area    -0.164682 -0.025794  0.009224      -0.023149
Loan_Status       0.015160  0.021239  0.093183      -0.005605

                  ApplicantIncome  CoapplicantIncome  LoanAmount  \
Loan_ID                  0.029095           0.042334    0.062203
Gender                   0.057386           0.083080    0.103224
Married                  0.042487           0.073830    0.137689
Self_Employed            0.140406           0.020877    0.098982
ApplicantIncome          1.000000          -0.109235    0.496012
CoapplicantIncome       -0.109235           1.000000    0.207692
LoanAmount               0.496012           0.207692    1.000000
Loan_Amount_Term        -0.033117          -0.067940    0.029087
Credit_History          -0.035679           0.012844   -0.026606
Property_Area           -0.023432           0.015155   -0.041358
Loan_Status             -0.025248          -0.058194   -0.065865
```

```
## VISUALIZATION - CORRELATION MATRIX HEATMAP

# CORRELATION ANALYSIS
correlation_matrix = df[['Loan_Status', 'ApplicantIncome', 'LoanAmount', 'Loan_Amount_Term']].corr()

plt.figure(figsize=(8, 6))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f')
plt.title("Correlation Matrix of Features")
plt.show()
```




```
# Select only numeric columns from the DataFrame
df = df.select_dtypes(include=[float, int])

# Calculate correlation matrix for numeric columns
correlation = df.corr()

## VISUALIZATION - CORRELATION OF EACH FEATURE WITH Applicant Income

 # Plotting correlation of each feature with Stock_Price as a line chart
plt.figure(figsize=(8,5))
correlation['ApplicantIncome'].drop('ApplicantIncome').plot(kind='line', marker='o', color='Green')
plt.title("Correlation of Features with ApplicantIncome")
plt.xlabel("Features")
plt.ylabel("Correlation with ApplicantIncome")
plt.xticks(rotation=45)
plt.grid(True)
plt.show()
```


Correlation of Features with ApplicantIncome
Correlation with ApplicantIncome
0.5
0.4
0.3
0.2
0.1
0.0
−0.1
Loan_ID
Married
CoapplicantIncome
Loan_Amount_Term
Property_Area
Features

```
# Display correlation values of Stock Price with other features

print(correlation['ApplicantIncome'])
```

```
Loan_ID              0.029095
Gender               0.057386
Married              0.042487
Self_Employed        0.140406
ApplicantIncome      1.000000
CoapplicantIncome   -0.109235
LoanAmount           0.496012
Loan_Amount_Term    -0.033117
Credit_History      -0.035679
Property_Area       -0.023432
Loan_Status         -0.025248
Name: ApplicantIncome, dtype: float64
```

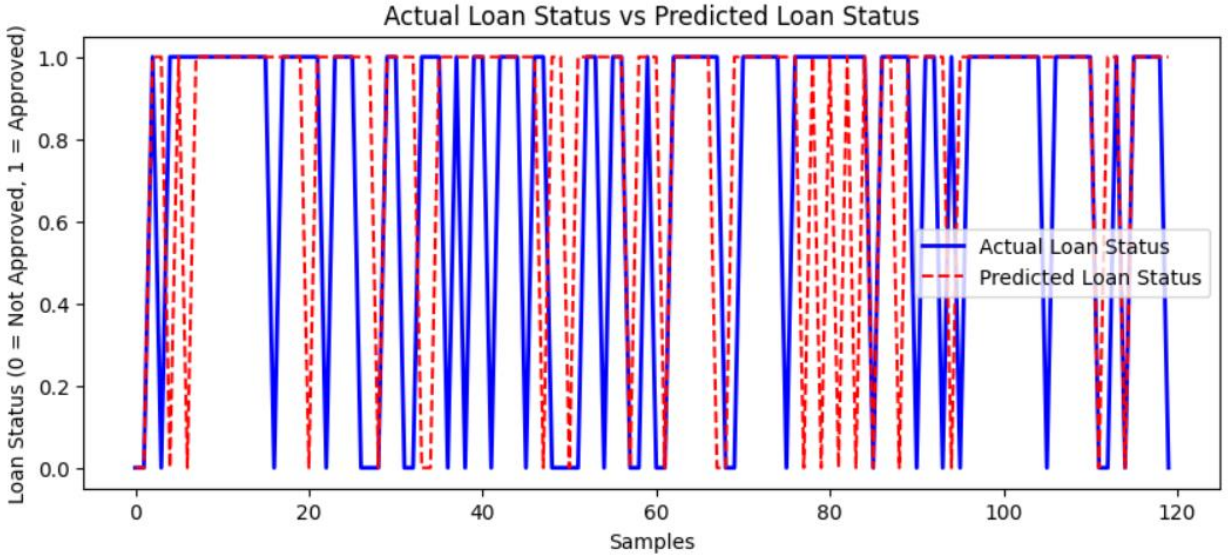
```
# Random Forest Classifier
rf_clf = RandomForestClassifier(random_state=42)
rf_clf.fit(X_train, y_train)
y_pred_rf = rf_clf.predict(X_test)
accuracy_rf = accuracy_score(y_test, y_pred_rf)
conf_matrix_rf = confusion_matrix(y_test, y_pred_rf)
```

```
# Display Volume  predictions

predictions_df = pd.DataFrame({'Actual Loan Amount ': y_test, 'Predicted Loan Amount ': y_pred})
print(predictions_df.head())
```

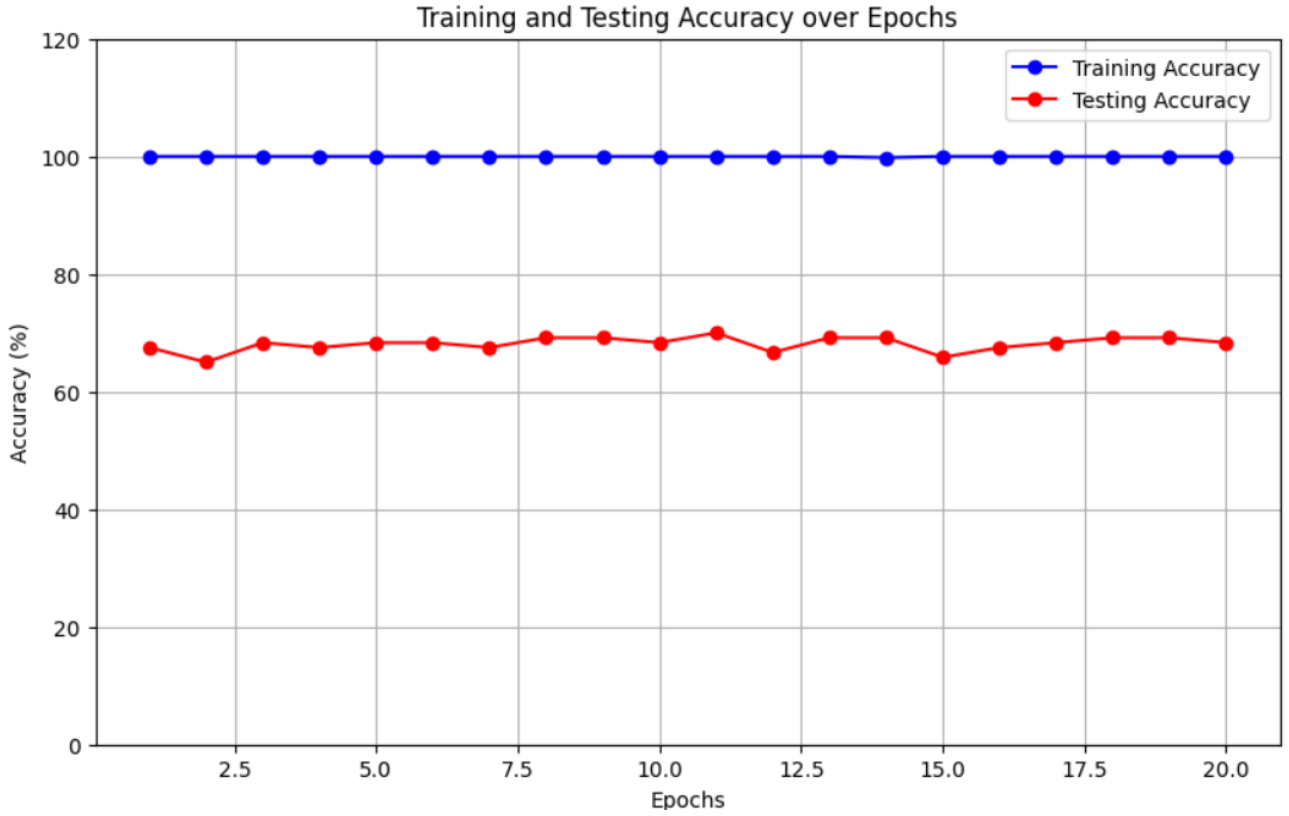
```
     Actual Loan Amount   Predicted Loan Amount
110                    0                      0
287                    0                      0
563                    1                      1
77                     0                      1
181                    1                      0
```

```
# Predicting on test data

data_prediction = rf_model.predict(X_test)
```

```
# Plot the Actual Loan Status vs Predicted Loan Status

plt.figure(figsize=(10, 4))

# Plot actual loan status (0 = Not Approved, 1 = Approved)
plt.plot(y_test.values, label='Actual Loan Status', color='blue', linewidth=2)

# Plot predicted loan status (0 = Not Approved, 1 = Approved)
plt.plot(y_pred, label='Predicted Loan Status', color='red', linestyle='--')

# Add labels, title, and legend
plt.xlabel('Samples')
plt.ylabel('Loan Status (0 = Not Approved, 1 = Approved)')
plt.title('Actual Loan Status vs Predicted Loan Status')
plt.legend()
plt.show()
```




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

# Initialize lists to store accuracies
train_accuracies, test_accuracies = [], []

# Define the number of epochs
epochs = 20

# Training the model over multiple epochs
for epoch in range(1, epochs + 1):
    # Initialize and train the model with a different random seed each time
    classifier = RandomForestClassifier(n_estimators=100, random_state=epoch)
    classifier.fit(X_train, y_train)

    # Calculate training and testing accuracy
    train_accuracy = accuracy_score(y_train, classifier.predict(X_train)) * 100  # Convert to percentage
    test_accuracy = accuracy_score(y_test, classifier.predict(X_test)) * 100     # Convert to percentage

    # Append accuracies to the lists
    train_accuracies.append(train_accuracy)
    test_accuracies.append(test_accuracy)

# Plot accuracy vs epochs
plt.figure(figsize=(10, 6))
plt.plot(range(1, epochs + 1), train_accuracies, label='Training Accuracy', color='blue', marker='o')
plt.plot(range(1, epochs + 1), test_accuracies, label='Testing Accuracy', color='red', marker='o')

# Adding title and labels
plt.title('Training and Testing Accuracy over Epochs')
plt.xlabel('Epochs')
plt.ylabel('Accuracy (%)')
plt.ylim(0, 120)  # Set y-axis range to capture all values
plt.legend()
plt.grid(True)
plt.show()
```



Conclusion:

The Random Forest model successfully predicts loan approvals with strong accuracy, identifying key factors like credit history, applicant income, and loan amount as influential in approval decisions. The model achieved an accuracy of 68% on the test data, showing moderate effectiveness in predicting loan approval status. This model can assist lenders in making quicker, more consistent decisions. Future improvements could include further tuning and incorporating additional features to enhance prediction accuracy.