Name: Alok Yadav Div: D15C Roll No.: 59

EXPERIMENT 1

Aim: Introduction to Data science and Data preparation using Pandas steps.

- Load data in Pandas.
- Description of the dataset.
- Drop columns that aren't useful.
- Drop rows with maximum missing values.
- Take care of missing data.
- Create dummy variables.
- Find out outliers (manually)
- Standardization and normalization of columns

Dataset: Financial Risk Assessment
Link: https://www.kaggle.com/datasets/preethamgouda/financial-risk

Steps:

1) **Loading data in Pandas and extracting information about the dataset.**

To load a file onto python for analysis, we need to make use of the pandas library. It gives us functionalities to read a CSV (Comma Separated Values) file and perform various functions on it.

Commands: import pandas as pd (Importing the pandas library onto Google Colab Notebook) df = pd.read_csv() (Mounts and reads the file in Python and assigns it to variable df for ease of use further)
(Note: Replace with the actual path of the file in "")

dataset.info(): This command gives all the information about the features (columns) of the dataset and the data type of each of these columns. It also gives a summary of all the values in the dataset.

```
import pandas as pd
import numpy as np
```

```
# loading the dataset to pandas df
dataset = pd.read_csv("/content/financial_risk_assessment.csv")
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 15000 entries, 0 to 14999
Data columns (total 20 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   Age                     15000 non-null  int64
 1   Gender                  15000 non-null  object
 2   Education Level         15000 non-null  object
 3   Marital Status          15000 non-null  object
 4   Income                  12750 non-null  float64
 5   Credit Score            12750 non-null  float64
 6   Loan Amount             12750 non-null  float64
 7   Loan Purpose            15000 non-null  object
 8   Employment Status       15000 non-null  object
 9   Years at Current Job    15000 non-null  int64
 10  Payment History         15000 non-null  object
 11  Debt-to-Income Ratio    15000 non-null  float64
 12  Assets Value            12750 non-null  float64
 13  Number of Dependents    12750 non-null  float64
 14  City                    15000 non-null  object
 15  State                   15000 non-null  object
 16  Country                 15000 non-null  object
 17  Previous Defaults       12750 non-null  float64
 18  Marital Status Change   15000 non-null  int64
 19  Risk Rating             15000 non-null  object
dtypes: float64(7), int64(3), object(10)
memory usage: 2.3+ MB
```

2) df.head(): As mentioned before, head function give us the first 5 rows of the dataset. This allows for the user to get an overview on what values are being listed in the dataset.

```
[ ] dataset.head()
```

	Age	Gender	Education Level	Marital Status	Income	Credit Score	Loan Amount	Loan Purpose	Employment Status	Years at Current Job	Payment History	Debt-to-Income Ratio	Assets Value	Number of Dependents	City	State	Country	Previous Defaults	Marital Status Change
0	49	Male	PhD	Divorced	72799.0	688.0	45713.0	Business	Unemployed	19	Poor	0.154313	120228.0	0.0	Port Elizabeth	AS	Cyprus	2.0	2
1	57	Female	Bachelor's	Widowed	NaN	690.0	33835.0	Auto	Employed	6	Fair	0.148920	55849.0	0.0	North Catherine	OH	Turkmenistan	3.0	2
2	21	Non-binary	Master's	Single	55687.0	600.0	36623.0	Home	Employed	8	Fair	0.362398	180700.0	3.0	South Scott	OK	Luxembourg	3.0	2
3	59	Male	Bachelor's	Single	26508.0	622.0	26541.0	Personal	Unemployed	2	Excellent	0.454964	157319.0	3.0	Robinhaven	PR	Uganda	4.0	2
4	25	Non-binary	Bachelor's	Widowed	49427.0	766.0	36528.0	Personal	Unemployed	10	Fair	0.143242	287140.0	NaN	New Heather	IL	Namibia	3.0	1

3) dataset.shape(): returns the dimensions of the dataset as a tuple (rows, columns), helping to understand its size.

```
[ ] dataset.shape
```

```
(15000, 20)
```

4) Describe the dataset

dataset.describe(): provides statistical summaries of numerical columns, including count, mean, standard deviation, min, max, and quartiles (25%, 50%, 75%).

```
dataset.describe()
```

	Age	Income	Credit Score	Loan Amount	Years at Current Job	Debt-to-Income Ratio	Assets Value	Number of Dependents	Previous Defaults	Marital Status Change
count	15000.000000	12750.000000	12750.000000	12750.000000	15000.000000	15000.000000	12750.000000	12750.00000	12750.000000	15000.000000
mean	43.452667	69933.398510	699.109098	27450.010902	9.476267	0.350438	159741.497176	2.02651	1.992471	0.998467
std	14.910732	29163.626207	57.229465	12949.940135	5.769707	0.143919	80298.115832	1.41130	1.416909	0.813782
min	18.000000	20005.000000	600.000000	5000.000000	0.000000	0.100004	20055.000000	0.00000	0.000000	0.000000
25%	31.000000	44281.500000	650.000000	16352.500000	4.000000	0.227386	90635.250000	1.00000	1.000000	0.000000
50%	43.000000	69773.000000	699.000000	27544.000000	9.000000	0.350754	159362.000000	2.00000	2.000000	1.000000
75%	56.000000	95922.750000	748.000000	38547.500000	15.000000	0.476095	228707.000000	3.00000	3.000000	2.000000
max	69.000000	119997.000000	799.000000	49998.000000	19.000000	0.599970	299999.000000	4.00000	4.000000	2.000000

If the parameter of include=”all” is included { df.describe(include=“all”)}, this includes even the non numeric values and gives some more information on fields such as count of unique values, top value, etc.

```
dataset.describe(include="all")
```

	Age	Gender	Education Level	Marital Status	Income	Credit Score	Loan Amount	Loan Purpose	Employment Status	Years at Current Job	Payment History	Debt-to-Income Ratio	Assets Value	Number of Dependents	City
count	15000.000000	15000	15000	15000	12750.000000	12750.000000	12750.000000	15000	15000	15000.000000	15000	15000.000000	12750.000000	12750.00000	15000
unique	NaN	3	4	4	NaN	NaN	NaN	4	3	NaN	4	NaN	NaN	NaN	10614
top	NaN	Non-binary	Bachelor's	Widowed	NaN	NaN	NaN	Personal	Employed	NaN	Good	NaN	NaN	NaN	East Michael
freq	NaN	5059	3829	3893	NaN	NaN	NaN	3771	5026	NaN	3822	NaN	NaN	NaN	19
mean	43.452667	NaN	NaN	NaN	69933.398510	699.109098	27450.010902	NaN	NaN	9.476267	NaN	0.350438	159741.497176	2.02651	NaN
std	14.910732	NaN	NaN	NaN	29163.626207	57.229465	12949.940135	NaN	NaN	5.769707	NaN	0.143919	80298.115832	1.41130	NaN
min	18.000000	NaN	NaN	NaN	20005.000000	600.000000	5000.000000	NaN	NaN	0.000000	NaN	0.100004	20055.000000	0.00000	NaN
25%	31.000000	NaN	NaN	NaN	44281.500000	650.000000	16352.500000	NaN	NaN	4.000000	NaN	0.227386	90635.250000	1.00000	NaN
50%	43.000000	NaN	NaN	NaN	69773.000000	699.000000	27544.000000	NaN	NaN	9.000000	NaN	0.350754	159362.000000	2.00000	NaN
75%	56.000000	NaN	NaN	NaN	95922.750000	748.000000	38547.500000	NaN	NaN	15.000000	NaN	0.476095	228707.000000	3.00000	NaN
max	69.000000	NaN	NaN	NaN	119997.000000	799.000000	49998.000000	NaN	NaN	19.000000	NaN	0.599970	299999.000000	4.00000	NaN

5) Dropping the columns

dataset.drop() is used to remove specified rows or columns from the dataset.

- dataset.drop(columns=['column_name']) → Drops a specific column.
- dataset.drop(index=[row_index]) → Drops a specific row.

```
# dropping the columns that aren't useful
cols = ['Marital Status', 'Marital Status Change', 'Loan Purpose','City','State']
df = dataset.drop(cols, axis=1)
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 15000 entries, 0 to 14999
Data columns (total 15 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   Age                   15000 non-null  int64
 1   Gender                15000 non-null  object
 2   Education Level       15000 non-null  object
 3   Income                12750 non-null  float64
 4   Credit Score          12750 non-null  float64
 5   Loan Amount           12750 non-null  float64
 6   Employment Status     15000 non-null  object
 7   Years at Current Job  15000 non-null  int64
 8   Payment History       15000 non-null  object
 9   Debt-to-Income Ratio  15000 non-null  float64
 10  Assets Value          12750 non-null  float64
 11  Number of Dependents  12750 non-null  float64
 12  Country               15000 non-null  object
 13  Previous Defaults     12750 non-null  float64
 14  Risk Rating           15000 non-null  object
dtypes: float64(7), int64(2), object(6)
memory usage: 1.7+ MB
```

Before Dropping:

```
[ ] dataset.shape
```

```
(15000, 20)
```

After Dropping:

```
df.shape
```

```
(15000, 15)
```

As observed here, the columns of *'Marital Status', 'Marital Status Change', 'Loan Purpose','City','State'* have been dropped.

6) Drop rows with maximum missing rows

df["missing_count"] = df.isnull().sum(axis=1)
max_missing = df["missing_count"].max()
Here the maximum missing count is 6. So to clean up some of the data, we will remove the rows with 4 or more missing values. df = df[df["missing_count"] < 4]
The above set of commands do the following function:
i) Create a column called missing_count where the sum of all the cells having null values is stored.
ii) The maximum value from this missing_count column is considered for deletion
iii) Finally, we update the dataset by keeping the rows which have missing values less than a particular value

```
df["missing_count"] = df.isnull().sum(axis=1)
max_missing = df["missing_count"].max()
print(df.head())

df = df[df["missing_count"] < 4]
df.shape
```

```
   Age      Gender Education Level   Income  Credit Score  Loan Amount  \
0   49        Male             PhD  72799.0         688.0      45713.0
1   57      Female      Bachelor's      NaN         690.0      33835.0
2   21  Non-binary        Master's  55687.0         600.0      36623.0
3   59        Male      Bachelor's  26508.0         622.0      26541.0
4   25  Non-binary      Bachelor's  49427.0         766.0      36528.0

  Employment Status  Years at Current Job Payment History  \
0        Unemployed                    19            Poor
1          Employed                     6            Fair
2          Employed                     8            Fair
3        Unemployed                     2       Excellent
4        Unemployed                    10            Fair

   Debt-to-Income Ratio  Assets Value  Number of Dependents       Country  \
0              0.154313      120228.0                   0.0        Cyprus
1              0.148920       55849.0                   0.0  Turkmenistan
2              0.362398      180700.0                   3.0    Luxembourg
3              0.454964      157319.0                   3.0        Uganda
4              0.143242      287140.0                   NaN       Namibia

   Previous Defaults Risk Rating  missing_count
0                2.0         Low             0
1                3.0      Medium             1
2                3.0      Medium             0
3                4.0      Medium             0
4                3.0         Low             1
(14909, 16)
```

To check the total missing values in each columns.
df.isnull().sum() is used to check for missing values (NaN) in a dataset. Here's how it works:

df.isnull() creates a DataFrame of the same shape as df, where each value is True if it's missing (NaN) and False otherwise.
.sum() then counts the number of True values (missing values) in each column.

7) Take care of the missing values

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

	0
Age	0
Gender	0
Education Level	0
Income	2189
Credit Score	2193
Loan Amount	2187
Employment Status	0
Years at Current Job	0
Payment History	0
Debt-to-Income Ratio	0
Assets Value	2196
Number of Dependents	2185
Country	0
Previous Defaults	2182
Risk Rating	0
missing_count	0

So, there are many missing values, hence performing the next step.

- To take care of the missing data that has not been removed, one of the 2 methods can be used: If the feature is of a numeric data type, we can use either mean, median or mode of the feature. If the data is normally distributed, use mean, if it is skewed, use median, and if many values are repeated, use mode.
- If the feature contains different categories, there are 2 ways. Either fill it with the mode of the column, or add a custom value such as “Data Unavailable”.

```
[ ]  # handling the missing data
     df.fillna({'Income':df['Income'].median()},inplace=True)
```

```
[ ]  df.fillna({'Credit Score':df['Credit Score'].median()},inplace=True)
```

To check the columns with missing values, using the following command
df[df.isnull().any(axis=1)] filters and returns all rows that contain at least one missing (NaN) value.

8) Creating dummy variables

pd.get_dummies(df, columns=categorical_columns, prefix=categorical_columns, drop_first=False) is used to convert categorical variables into one-hot encoded format. This transformation helps machine learning models process categorical data.

Breaking Down the Code:
pd.get_dummies(df, columns=categorical_columns, prefix=categorical_columns, drop_first=False)

- Converts each categorical column into multiple binary (0/1) columns, representing unique categories.
- prefix=categorical_columns ensures that the new columns have meaningful names.
- drop_first=False keeps all categories (if True, it drops the first category to avoid multicollinearity).
- for col in categorical_columns: df_dummies[col] = df[col]

This restores the original categorical columns back into df_dummies, so the dataset now contains both original and encoded versions.

```
categorical_columns = ['Risk Rating', 'Gender', 'Employment Status', 'Payment History']

df_dummies = pd.get_dummies(df, columns=categorical_columns, prefix=categorical_columns, drop_first=False)

for col in categorical_columns:
    df_dummies[col] = df[col]

print(df_dummies.head())
```

```
   Age Education Level   Income  Credit Score  Loan Amount  \
0   49             PhD  72799.0         688.0      45713.0
1   57      Bachelor's  69773.0         690.0      33835.0
2   21        Master's  55687.0         680.0      36623.0
3   59      Bachelor's  26508.0         622.0      26541.0
5   30             PhD  69773.0         717.0      15613.0

   Years at Current Job  Debt-to-Income Ratio   Assets Value  \
0                    19              0.154313  120228.000000
1                     6              0.148920   55849.000000
2                     8              0.362398  188700.000000
3                     2              0.454964  157319.000000
5                     5              0.295984  159741.497176

   Number of Dependents       Country  ...  Employment Status_Self-employed  \
0                   0.0        Cyprus  ...                            False
1                   0.0  Turkmenistan  ...                            False
2                   3.0    Luxembourg  ...                            False
3                   3.0        Uganda  ...                            False
5                   4.0       Iceland  ...                            False

   Employment Status_Unemployed  Payment History_Excellent  \
0                          True                      False
1                         False                      False
2                         False                      False
3                          True                       True
5                          True                      False

   Payment History_Fair  Payment History_Good  Payment History_Poor  \
0                 False                 False                  True
1                  True                 False                 False
2                  True                 False                 False
3                 False                 False                 False
5                  True                 False                 False

   Risk Rating      Gender  Employment Status  Payment History
0          Low        Male        Unemployed             Poor
1       Medium      Female          Employed             Fair
2       Medium  Non-binary          Employed             Fair
3       Medium        Male        Unemployed        Excellent
5       Medium  Non-binary        Unemployed             Fair
```

9) Detecting Outlier data

Using IQR Value:

In this method, we find the IQR value for the column; which is the difference between Q1 - 1.5 * IQR and Q3 + 1.5 * IQR. This is a standard that is followed, the factor 1.5 can be modified between 1 to 3 based on the requirement.

Command:

```
Q1 = df['Data_Value'].quantile(0.25)
Q3 = df['Data_Value'].quantile(0.75)
IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR
outliers = df[(df['Data_Value'] < lower_bound) | (df['Data_Value'] > upper_bound)]
```

This method gives the outliers and hence can be removed.

Using Manual method:

Checking for Outlier data in Excel using different value ranges.

And then using the preprocessed data.

```
df.to_csv('financial_risk_preprocessed.csv', index=False)
```

```
[ ] cleaned_df = pd.read_csv('/content/financial_risk_preprocessed WITH DEL.csv')
```

10) Standardization and Normalization of columns

- StandardScaler: Standardizes features by removing the mean and scaling to unit variance.
- MinMaxScaler: Normalizes features to a fixed range (0 to 1 by default).

Standardize Column:

Using formula:

mean_value = df["Data_Value"].mean()
std_value = df["Data_Value"].std()
df["Standardized_Data_Value"] = (df["Data_Value"] - mean_value) / std_value

Using Library:

from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
df['Standardized Data Value Scalar'] = scaler.fit_transform(df[['Data_Value']])

Normalize column:

Method 1:

Formula min_val = df['Data_Value'].min()
max_val = df['Data_Value'].max()
df['Data_Value_Normalized'] = (df['Data_Value'] - min_val) / (max_val - min_val)

Method 2:

Scaler library from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
df['Normalized Data Value Scalar'] = scaler.fit_transform(df[['Data_Value']])

Here, the columns, Income, Credit Score, and Loan Amount are standardized and normalized

```
[ ] from sklearn.preprocessing import StandardScaler, MinMaxScaler


    standard_scaler = StandardScaler()
    min_max_scaler = MinMaxScaler()

    cleaned_df['Income'] = standard_scaler.fit_transform(cleaned_df[['Income']])
    cleaned_df['Credit Score'] = standard_scaler.fit_transform(cleaned_df[['Credit Score']])

    cleaned_df['Loan Amount'] = min_max_scaler.fit_transform(cleaned_df[['Loan Amount']])

    print(cleaned_df[['Income', 'Credit Score', 'Loan Amount']].head())
```

```
     Income  Credit Score  Loan Amount
0 -0.015390     -0.209478     0.000914
1 -0.017087     -0.172080     0.000677
2 -0.024984     -1.854955     0.000732
3 -0.041344     -1.443586     0.000531
4 -0.017087      0.332782     0.000312
```

Conclusion:

In this experiment, we used pandas and scikit learn to preprocess data, perform normalization and standardization to to make the dataset clean and efficient. Firstly, the dataset in the form of csv file was imported into the Collab and then using df.info(), information related about the features (columns) of the dataset and the data type of each of these columns. Using df.head(), the dataset in the form of dataframe can be viewed in which the top 5 values can be displayed. Then the missing values were detected using df.isnull() and performing handling methods like dropping the rows with missing values, and replacing missing values with mean, median or mode, the missing values were handled.

After handling missing values and encoding categorical variables using one-hot encoding (creating dummy variables), the dataset was further refined by detecting and removing outliers using the Interquartile Range (IQR) method. Outliers were identified based on their deviation from the first (Q1) and third quartile (Q3) thresholds and manual processing on the dataset using Excel, ensuring that extreme values did not affect the model's performance.

StandardScaler was used to transform the columns, Income, Credit Score to standardized and normalized by centering them around a mean of zero with unit variance, making them suitable for models that assume normally distributed data. On the other hand, MinMaxScaler was applied to Loan Amout to a fixed range between 0 and 1.

Name: Alok Yadav Div: D15C Roll No.: 59

EXPERIMENT 2

Aim: Data Visualization/ Exploratory data Analysis using Matplotlib and Seaborn.

1. Create bar graph, contingency table using any 2 features.
2. Plot Scatter plot, box plot, Heatmap using seaborn.
3. Create histogram and normalized Histogram.
4. Describe what this graph and table indicates.
5. Handle outlier using box plot and Inter quartile range.

Steps:

Dataset: Financial Risk Assessment

Link: https://www.kaggle.com/datasets/preethamgouda/financial-risk

1) Loading the dataset\

Loading the dataset as pandas DataFrame and using df.info() to get information about the features and attributes of the dataset.

```
import pandas as pd
df = pd.read_csv('/content/financial_risk_assessment.csv')
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 15000 entries, 0 to 14999
Data columns (total 20 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   Age                    15000 non-null  int64
 1   Gender                 15000 non-null  object
 2   Education Level        15000 non-null  object
 3   Marital Status         15000 non-null  object
 4   Income                 12750 non-null  float64
 5   Credit Score           12750 non-null  float64
 6   Loan Amount            12750 non-null  float64
 7   Loan Purpose           15000 non-null  object
 8   Employment Status      15000 non-null  object
 9   Years at Current Job   15000 non-null  int64
 10  Payment History        15000 non-null  object
 11  Debt-to-Income Ratio   15000 non-null  float64
 12  Assets Value           12750 non-null  float64
 13  Number of Dependents   12750 non-null  float64
 14  City                   15000 non-null  object
 15  State                  15000 non-null  object
 16  Country                15000 non-null  object
 17  Previous Defaults      12750 non-null  float64
 18  Marital Status Change  15000 non-null  int64
 19  Risk Rating            15000 non-null  object
dtypes: float64(7), int64(3), object(10)
memory usage: 2.3+ MB
```

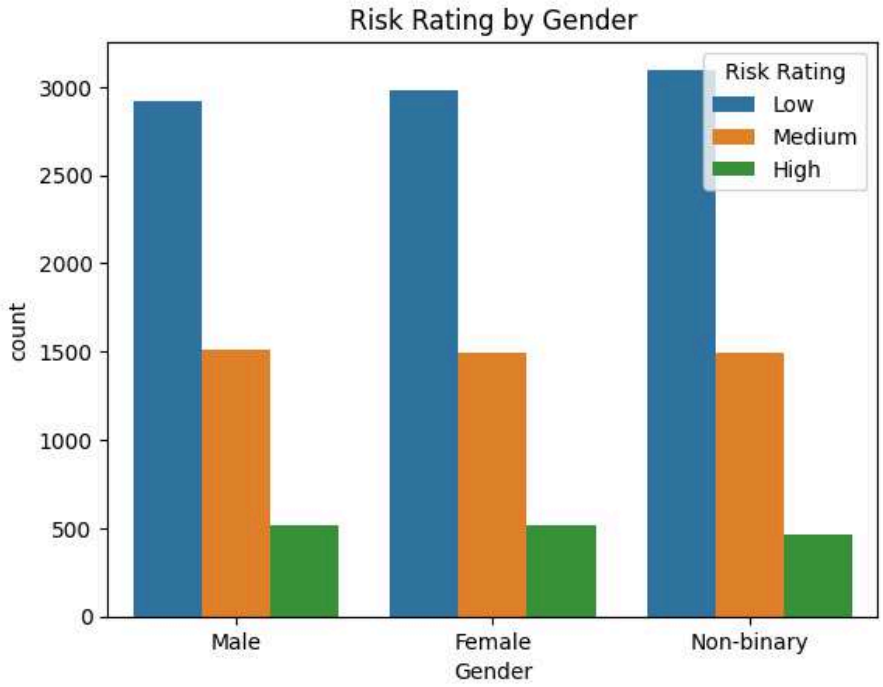
2) Creating bar graph and Contingency table

Importing seaborn library which is a Python data visualization library built on top of Matplotlib. It provides a high-level interface for creating attractive and informative statistical graphics.

Plotting the bar graph using the columns - Gender and Risk Rating.

Creating contingency table using the following command:
The pd.crosstab() function in Pandas is used to create a cross-tabulation (contingency table) of two or more categorical variables. It helps analyze relationships between variables by showing the frequency of their occurrences.

```
import seaborn as sns
import matplotlib.pyplot as plt

sns.countplot(data=df, x='Gender', hue='Risk Rating')
plt.title('Risk Rating by Gender')
plt.show()

# Creating a contingency table (cross-tabulation) for Gender and Risk Rating
contingency_table = pd.crosstab(df['Gender'], df['Risk Rating'])
print("\nContigency Table\n")
print(contingency_table)
```




```
Contigency Table

Risk Rating  High   Low  Medium
Gender
Female        518  2981    1491
Male          515  2921    1515
Non-binary    467  3098    1494
```

The contingency table shows that most individuals fall into the low-risk category, with non-binary individuals having the highest count (3,098), followed by females (2,981) and males (2,921). Medium risk is fairly balanced across genders, while high risk is the least common, slightly higher in females (518) and males (515) than non-binary individuals (467). Overall, financial risk perception appears similar across genders, with a predominant trend toward low risk.

3) Plot Scatter plot, box plot, Heatmap using seaborn.

Scatter plot:

A scatter plot is a graphical representation used to visualize the relationship between two numerical variables. Each point on the plot represents an observation. Scatter plots help identify patterns such as positive or negative correlations, clusters, and outliers. If the points form an upward trend, it suggests a positive correlation, whereas a downward trend indicates a negative correlation.

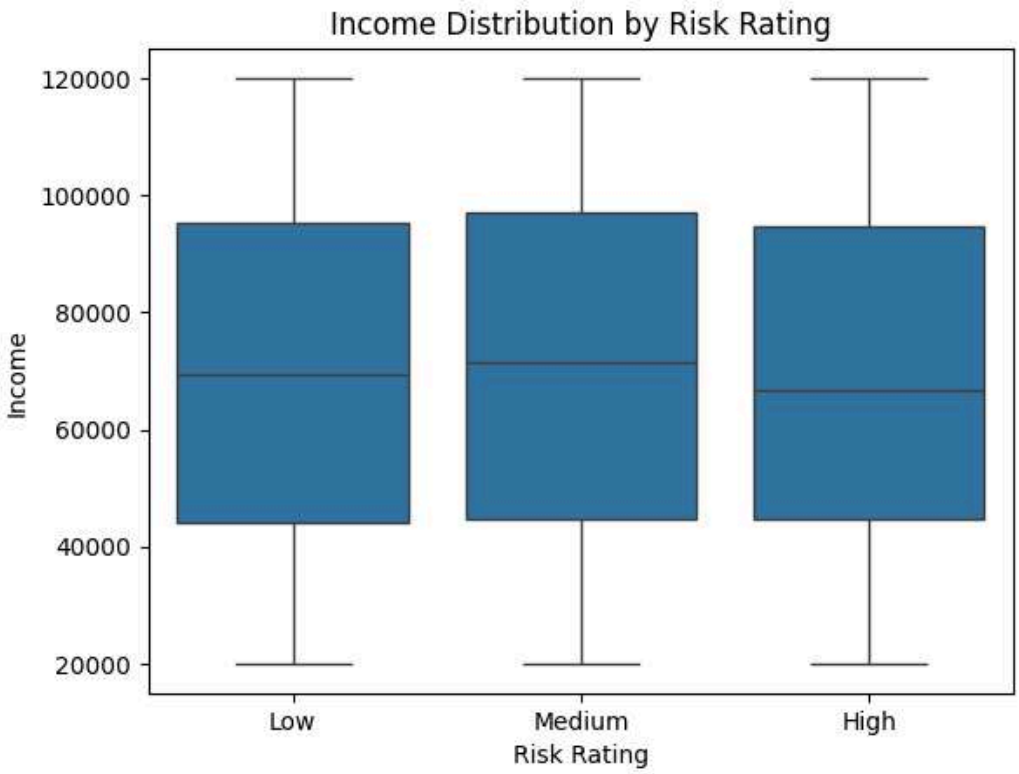
Box plot:

A **box plot** (also known as a box-and-whisker plot) is a statistical visualization that summarizes the distribution of a dataset using five key measures: minimum, first quartile (Q1), median (Q2), third quartile (Q3), and maximum. The box represents the interquartile range (IQR), with the median marked inside it, while "whiskers" extend to show variability outside the quartiles.

Heatmap:

A **heatmap** is a data visualization technique that represents values in a matrix format using color intensity. It is often used to display correlations between variables, frequency distributions, or hierarchical clustering results. Darker or more intense colors indicate higher values, while lighter colors represent lower values.

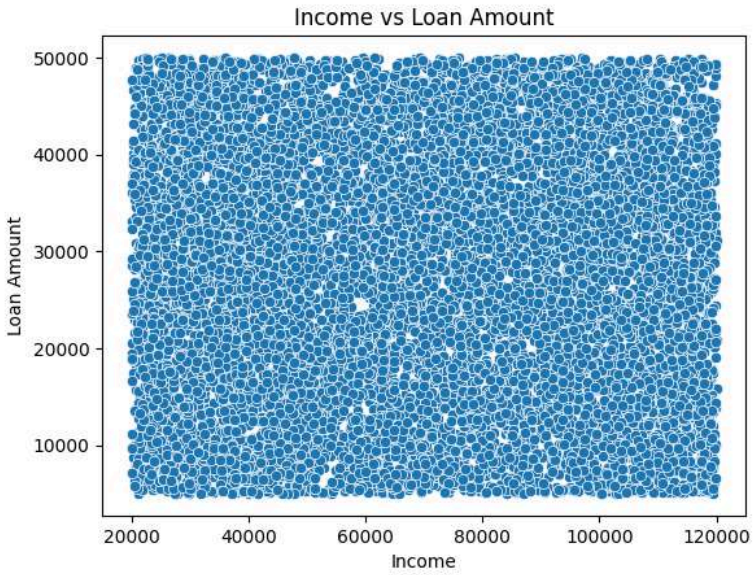
```
sns.boxplot(data=df, x='Risk Rating', y='Income')
plt.title('Income Distribution by Risk Rating')
plt.show()
```



The box plot illustrates the distribution of income across different financial risk ratings (Low, Medium, High). The median income is similar across all risk categories, suggesting that income levels do not strongly differentiate financial risk ratings. The interquartile ranges (IQR) and overall spread of incomes are also comparable, indicating a consistent income distribution across risk groups. Since there are no significant outliers or deviations, it implies that factors other than income may play a key role in determining financial risk ratings.

Scatter Plot

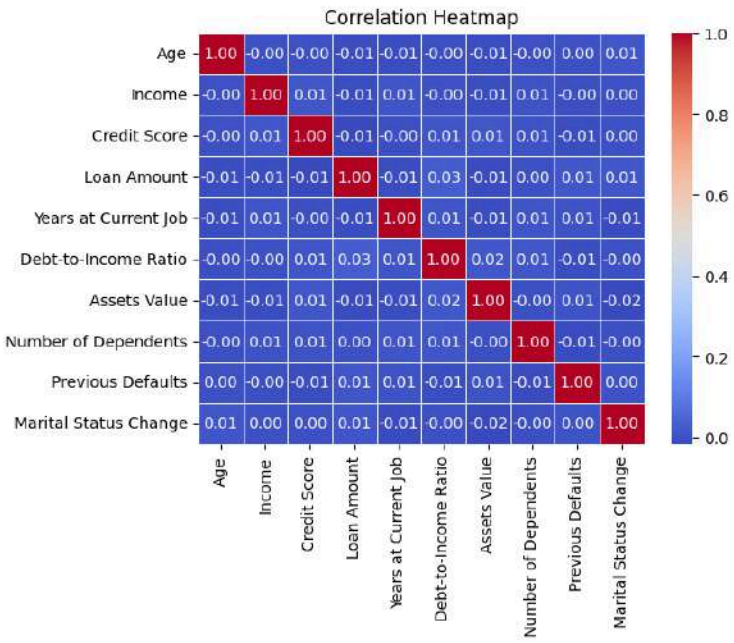
```
sns.scatterplot(data=df, x='Income', y='Loan Amount')
plt.title('Income vs Loan Amount')
plt.show()
```



The scatter plot illustrates the relationship between income and loan amount. The data appears densely packed and evenly distributed across the entire range, suggesting no clear correlation between income and loan amount. The loan amounts vary widely regardless of income levels, indicating that factors other than income may play a significant role in determining loan amounts. The lack of an apparent trend suggests that loan approvals or allocations are influenced by additional factors beyond just income.

Heatmap

```
# Select only numeric columns
numeric_columns = df.select_dtypes(include=['float64', 'int64'])

# Calculate the correlation matrix
corr_matrix = numeric_columns.corr()

# Create the heatmap
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt='.2f', linewidths=0.5)
plt.title('Correlation Heatmap')
plt.show()
```



The correlation heatmap shows the relationships between various financial factors. Most correlations are close to zero, indicating weak or no linear relationships between the variables. Strong correlations (value of 1.00) appear only along the diagonal, which represents self-correlation. Overall, no significant relationships exist among the variables, implying that financial risk assessment may depend on a combination of multiple independent factors rather than direct correlations.

4) Create histogram and normalized Histogram

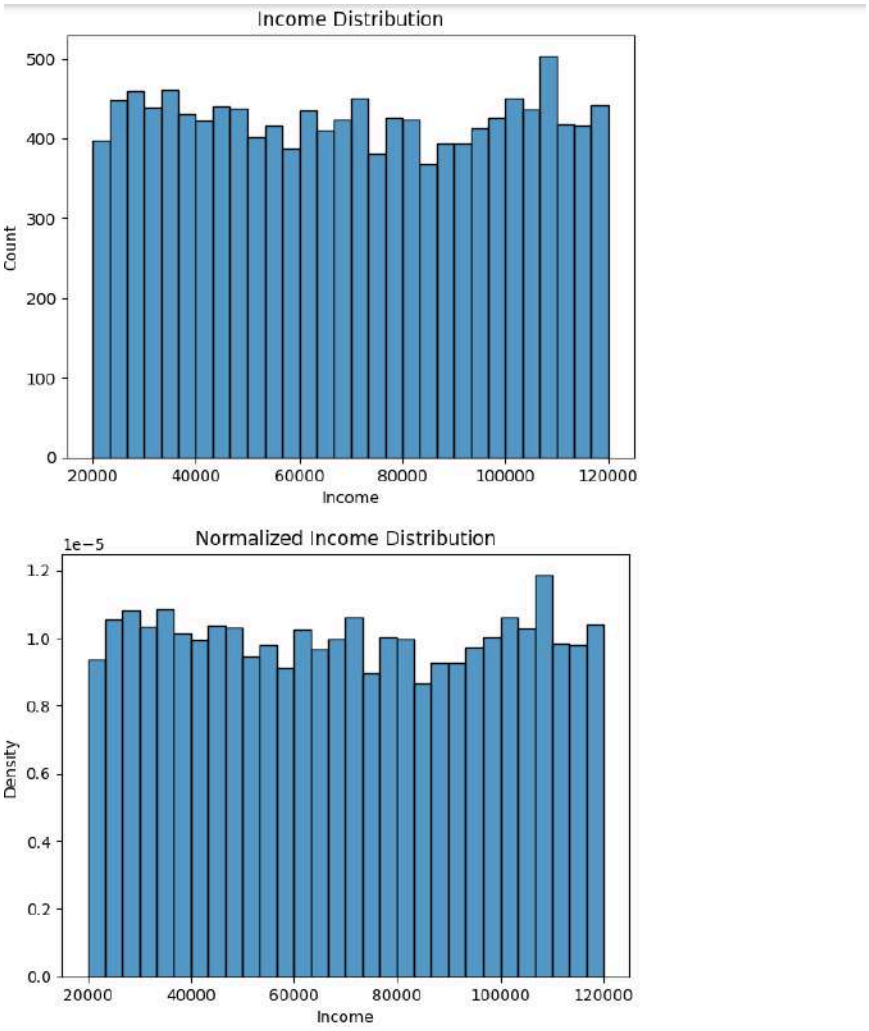
Histogram:

Histograms help visualize the shape, spread, and central tendency of data, making it easier to identify patterns such as skewness, modality (unimodal, bimodal), and outliers.

Normalized Histogram:

A normalized histogram is a variation of a histogram where the frequencies are scaled to represent relative frequencies instead of raw counts. This is done by dividing each bin's frequency by the total number of observations or by ensuring the total area under the histogram sums to 1.

```
#Histogram
sns.histplot(df['Income'], kde=False, bins=30)
plt.title('Income Distribution')
plt.show()

#Normalised Histogram
sns.histplot(df['Income'], kde=False, bins=30, stat='density')
plt.title('Normalized Income Distribution')
plt.show()
```



The income distribution histograms show a relatively uniform spread of income levels, with no strong skewness or extreme outliers. The first plot represents the raw count of individuals across income ranges, while the second normalizes the distribution to show density. Both indicate a fairly even income distribution, with slight fluctuations but no significant peaks or gaps. This suggests that income levels are well-distributed across the dataset, without any dominant income group.

5) Describe what this graph and table indicates.

Bar Graph (Risk Rating by Gender):

This bar graph will show the distribution of different risk ratings across the genders, helping us understand if there's a significant difference in risk ratings between male and female applicants.

Contingency Table:

The contingency table will show the exact count of observations for each combination of Gender and Risk Rating. This allows you to see how many male and female applicants fall into each risk rating category.

Scatter Plot (Income vs Loan Amount):
The scatter plot will reveal the relationship between income and loan amount. This could show if higher incomes tend to have higher loan amounts, or if there's any clustering or pattern.

Box Plot (Income by Risk Rating):
The box plot will give an idea of how income is distributed across different risk ratings. For example, if risk ratings are low for high-income individuals or vice versa, it will be evident.

Heatmap (Correlation Matrix):
The heatmap will help you quickly identify correlations between numeric features. For example, you might notice a strong correlation between Income and Loan Amount, or low correlation between Debt-to-Income Ratio and Number of Dependents.

6) Handle outlier using box plot and Inter quartile range

Outliers can be handled using the Box Plot and Interquartile Range (IQR) method. A box plot visually identifies outliers as points beyond the "whiskers," which represent data within 1.5 times the IQR.

Steps to Handle Outliers:

1. Calculate the IQR

   IQR = Q3 - Q1

   where Q1 (25th percentile and Q3 (75th percentile) are the lower and upper quartiles.

2. Determine the Outlier Thresholds:

   - Lower Bound = Q1 - 1.5 × IQR

   - Upper Bound = Q3 + 1.5 × IQR

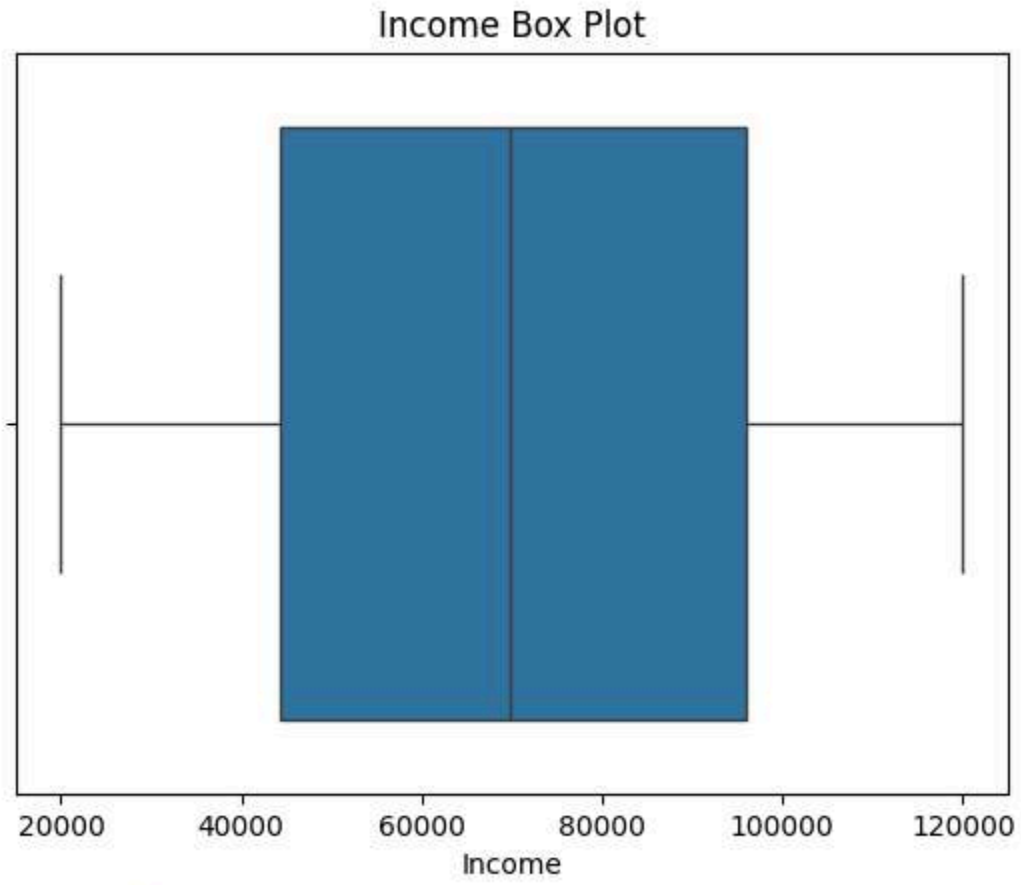
3. Remove or Adjust Outliers:

   - Drop outliers (if errors or extreme values).

   - Cap them at the lower or upper bound (winsorization).

   - Transform the data (log, square root) to reduce the impact.

```python
#Box plot to visualize outliers

sns.boxplot(data=df, x='Income')
plt.title('Income Box Plot')
plt.show()

#Identifying and Removing Outliers Using IQR:

# Calculate the Q1 (25th percentile) and Q3 (75th percentile)
Q1 = df['Income'].quantile(0.25)
Q3 = df['Income'].quantile(0.75)

# Calculate IQR (Interquartile Range)
IQR = Q3 - Q1

# Define outlier threshold (1.5 * IQR rule)
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Filter out the outliers
df_no_outliers = df[(df['Income'] >= lower_bound) & (df['Income'] <= upper_bound)]

# Verify the shape of the data before and after removing outliers
print(f"Original data shape: {df.shape}")
print(f"Data shape after removing outliers: {df_no_outliers.shape}")
```




```
Original data shape: (15000, 20)
Data shape after removing outliers: (12750, 20)
```

The box plot represents the distribution of income after removing outliers. The interquartile range (IQR) spans from around 30,000 to 100,000, with the median income close to 70,000. The whiskers extend from approximately 20,000 to 120,000, indicating the range of non-outlier values. The removal of outliers reduced the dataset size from 15,000 to 12,750, suggesting that around 2,250 data points were considered extreme values. The cleaned data now provides a more concentrated view of the central distribution, reducing the influence of extreme income variations.

Conclusion:

In this experiment, the analysis reveals that financial risk perception is largely similar across genders, with most individuals falling into the low-risk category. Box plots indicate that income levels do not significantly influence financial risk ratings, as distributions remain consistent across risk groups. The scatter plot shows no clear correlation between income and loan amount, suggesting that loan allocation depends on other factors. The correlation heatmap confirms weak or no linear relationships among financial variables, implying that financial risk assessment is influenced by multiple independent factors. Histograms demonstrate a relatively uniform income distribution with no dominant income group. After outlier removal, the dataset size reduced from 15,000 to 12,750, offering a more refined view of income distribution and reducing the impact of extreme values.

Name: Alok Yadav Div: D15C Roll No.: 59

Experiment No. 3

Aim: Perform Data Modeling.

Problem Statement:
a. Partition the data set, for example 75% of the records are included in the training data set and 25% are included in the test data set.

b. Use a bar graph and other relevant graph to confirm your proportions.

c. Identify the total number of records in the training data set.

d. Validate partition by performing a two-sample Z-test.

1. **Importing required libraries:**

```
[ ] import numpy as np
    import pandas as pd
    import matplotlib.pyplot as plt
    from sklearn.model_selection import train_test_split
    from scipy.stats import norm
```

2. **Overview of Dataset:**

df = pd.read_csv("/content/financial_risk_preprocessed.csv")
df.head()

```
[ ] df = pd.read_csv("/content/financial_risk_preprocessed.csv")
    df.head()
```

	Age	Gender	Education Level	Income	Credit Score	Loan Amount	Employment Status	Years at Current Job	Payment History	Debt-to-Income Ratio	Assets Value	Number of Dependents	Country	Previous Defaults	Risk Rating
0	49	Male	PhD	72799.0	688.0	45713.0	Unemployed	19	Poor	0.154313	120228.000000	0.0	Cyprus	2.0	Low
1	57	Female	Bachelor's	69773.0	690.0	33835.0	Employed	6	Fair	0.148920	55849.000000	0.0	Turkmenistan	3.0	Medium
2	21	Non-binary	Master's	55687.0	600.0	38623.0	Employed	8	Fair	0.362398	180700.000000	3.0	Luxembourg	3.0	Medium
3	59	Male	Bachelor's	26508.0	622.0	28541.0	Unemployed	2	Excellent	0.454864	157319.000000	3.0	Uganda	4.0	Medium
4	30	Non-binary	PhD	69773.0	717.0	15613.0	Unemployed	5	Fair	0.295984	158741.497176	4.0	Iceland	3.0	Medium

Splitting Training and Testing Dataset in 75% - 25%

The Financial Risk Assessment Dataset provides detailed information on individual financial profiles. It includes demographic, financial, and behavioral data to assess financial risk. The dataset features various columns such as income, credit score, and risk rating, with intentional imbalances and missing values to simulate real-world scenarios.

3. **Splitting Training and Testing Dataset in 75% - 25% :**

```
train, test = train_test_split(df, test_size=0.25, random_state=42)
print(f"Total records: {len(df)}")
print(f"Training set records: {len(train)}")
print(f"Test set records: {len(test)}")
```

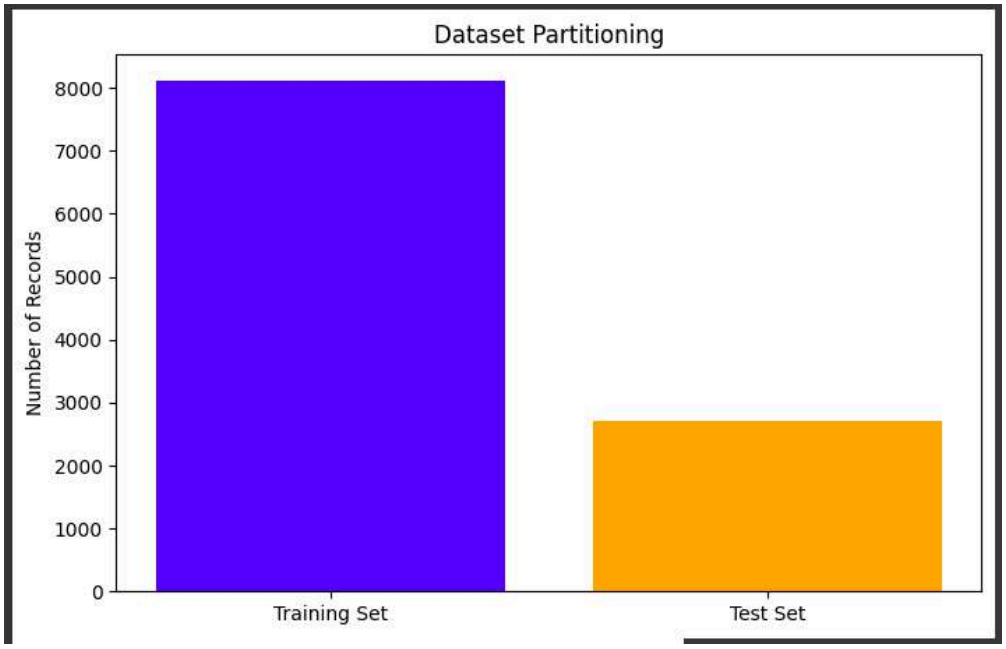
```
Total records: 10833
Training set records: 8124
Test set records: 2709
```

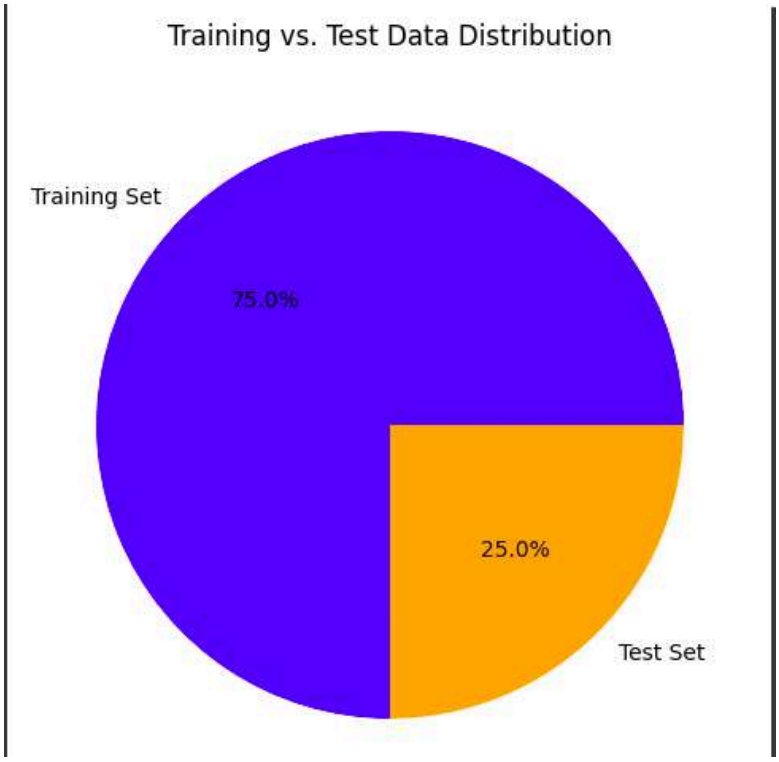
The dataset is divided into 75% for training and 25% for testing.

4. Plotting graph of Training and Testing Dataset

```
plt.figure(figsize=(8, 5))
plt.bar(["Training Set", "Test Set"], [len(train), len(test)], color=['blue', 'orange'])
plt.ylabel("Number of Records")
plt.title("Dataset Partitioning")
plt.show()
```

```
plt.figure(figsize=(6, 6))
plt.pie([len(train), len(test)], labels=["Training Set", "Test Set"], autopct="%1.1f%%",
colors=['blue', 'orange'])
plt.title("Training vs. Test Data Distribution")
plt.show()
```





From above graph we can see that our data is properly partitioned into 75% training data and 25% testing data

5. **Performing Z-Test**

```
column_name = df.columns[0]
train_mean = train[column_name].mean()
test_mean = test[column_name].mean()
train_std = train[column_name].std()
test_std = test[column_name].std()
n_train = len(train)
n_test = len(test)

# Compute Z-score
z_score = (train_mean - test_mean) / np.sqrt((train_std**2 / n_train) + (test_std**2 / n_test))

# Compute p-value
p_value = 2 * (1 - norm.cdf(abs(z_score)))

print(f"Z-Score: {z_score:.3f}")
print(f"P-Value: {p_value:.3f}")

# Interpretation
alpha = 0.05
if p_value < alpha:
    print("Reject the null hypothesis: The means of the two groups are significantly different.")
else:
    print("Fail to reject the null hypothesis: The means are similar between training and test sets.")
```

```
Z-Score: 1.604
P-Value: 0.109
Fail to reject the null hypothesis: The means are similar between training and test sets.
```

Performing two sample z-test on ‘Age’ columns. Given that the null hypothesis was not rejected, the data split is statistically valid

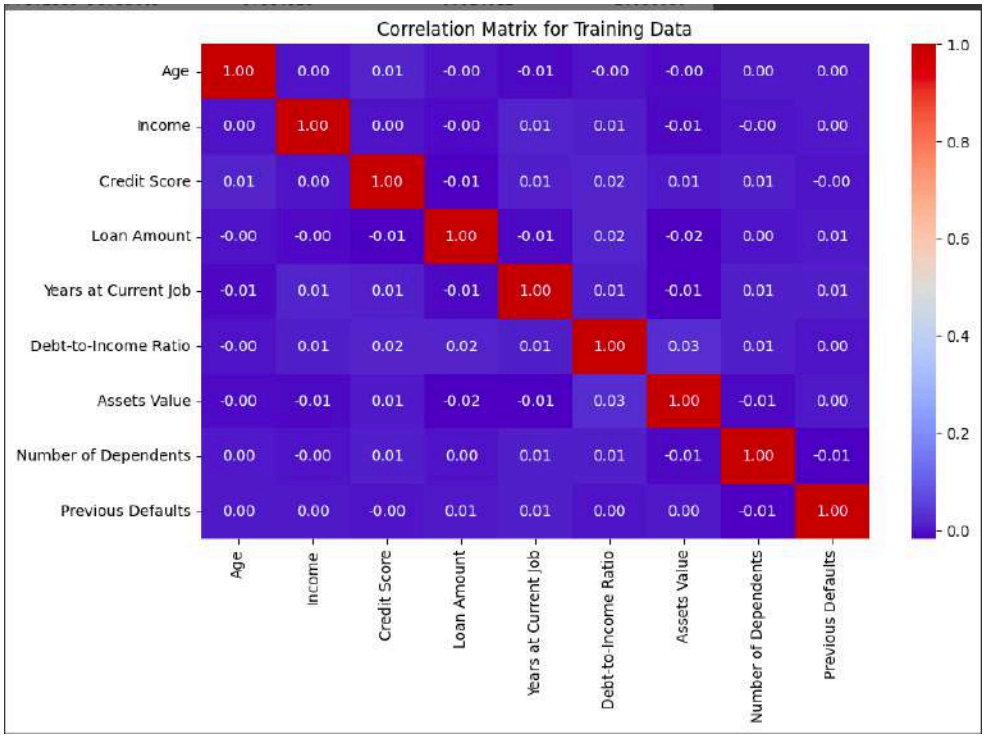
6. Performing Correlation Test

```
# Calculate Pearson correlation for all numerical columns
correlation_matrix = train.corr(numeric_only=True)
print("Correlation Matrix (Training Set):\n", correlation_matrix)

# Visualize the correlation matrix
```

```
import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 6))
sns.heatmap(correlation_matrix, annot=True, cmap="coolwarm", fmt=".2f")
plt.title("Correlation Matrix for Training Data")
plt.show()
```

```
Correlation Matrix (Training Set):
                            Age    Income  Credit Score  Loan Amount  \
Age                    1.000000  0.001015      0.013612    -0.000546
Income                 0.001015  1.000000      0.001392    -0.004547
Credit Score           0.013612  0.001392      1.000000    -0.013475
Loan Amount           -0.000546 -0.004547     -0.013475     1.000000
Years at Current Job  -0.006949  0.013522      0.011674    -0.007857
Debt-to-Income Ratio  -0.001505  0.007965      0.015618     0.016472
Assets Value          -0.003270 -0.006882      0.005112    -0.018345
Number of Dependents   0.004430 -0.000333      0.007754     0.001100
Previous Defaults      0.001992  0.001911     -0.001708     0.005074

                      Years at Current Job  Debt-to-Income Ratio  \
Age                                -0.006949             -0.001505
Income                              0.013522              0.007965
Credit Score                        0.011674              0.015618
Loan Amount                        -0.007857              0.016472
Years at Current Job                1.000000              0.008937
Debt-to-Income Ratio                0.008937              1.000000
Assets Value                       -0.012742              0.026238
Number of Dependents                0.008165              0.007587
Previous Defaults                   0.007654              0.000719

                      Assets Value  Number of Dependents  Previous Defaults
Age                      -0.003270              0.004430           0.001992
Income                   -0.006882             -0.000333           0.001911
Credit Score              0.005112              0.007754          -0.001708
Loan Amount              -0.018345              0.001100           0.005074
Years at Current Job     -0.012742              0.008165           0.007654
Debt-to-Income Ratio      0.026238              0.007587           0.000719
Assets Value              1.000000             -0.006907           0.004923
Number of Dependents     -0.006907              1.000000          -0.014012
Previous Defaults         0.004923             -0.014012           1.000000
```

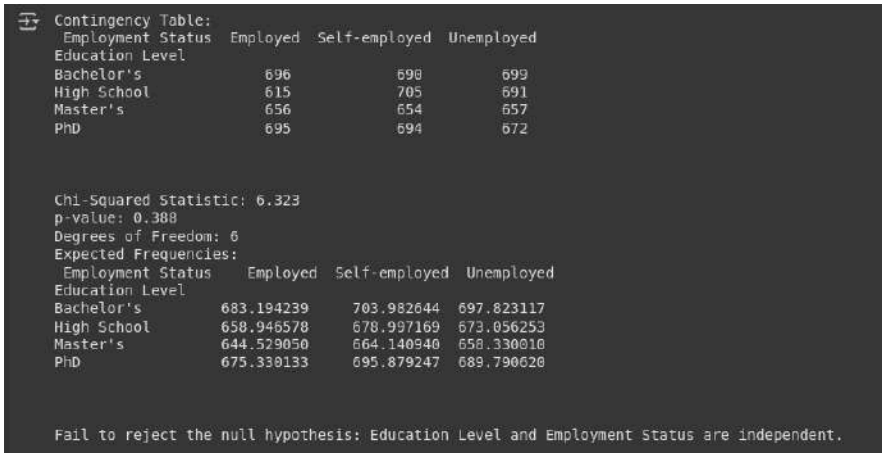
The negligible correlation values indicate an absence of a meaningful relationship between the columns, a finding that is further supported by the correlation heatmap.



7. Performing Chi-Squared Test:

```
# Create a contingency table for Education Level and Employment Status
contingency_table = pd.crosstab(train['Education Level'], train['Employment Status'])
print("Contingency Table:\n", contingency_table)
print("\n\n")

from scipy.stats import chi2_contingency

# Perform the chi-squared test
chi2, p, dof, expected = chi2_contingency(contingency_table)

# Display the results
print(f"Chi-Squared Statistic: {chi2:.3f}")
print(f"p-value: {p:.3f}")
print(f"Degrees of Freedom: {dof}")
print("Expected Frequencies:\n", pd.DataFrame(expected, index=contingency_table.index,
columns=contingency_table.columns))
print("\n\n")
# Interpret the p-value
alpha = 0.05
if p < alpha:
    print("Reject the null hypothesis: Education Level and Employment Status are dependent.")
```

else:
print("Fail to reject the null hypothesis: Education Level and Employment Status are independent.")



The Chi-Squared test was performed on columns ‘Education Level’ and 'Employment Status'. Since the test results do not reject the null hypothesis, it can be concluded that 'Education Level' and 'Employment Status' are independent variables according to the dataset.

8. Download partitioned Training and Testing dataset

```
train.to_csv("financial_risk_train_data.csv", index=False)
test.to_csv("financial_risk_test_data.csv", index=False)
from google.colab import files
files.download("financial_risk_train_data.csv")
files.download("financial_risk_test_data.csv")
```

We can use partitioned dataset for training and testing purposes

Conclusion:

We loaded the data into a Colab notebook and split it into 75% for training and 25% for testing. To verify the partition, we plotted a bar graph and a pie chart, which confirmed that the data was split correctly. Next, we performed a Z-test to assess the validity of the partition, and the results showed that the partition was valid, as the null hypothesis was not rejected. We then examined the correlation between all columns using a correlation heatmap, which revealed no significant correlation between the columns. Finally, we conducted a chi-square test on the 'Education Level' and 'Employment Status' columns, and the results indicated that the two features are independent, as the null hypothesis was not rejected

Experiment No: 4

Aim: Implementation of Statistical Hypothesis Test using Scipy and Sci-kit learn.

Problem Statement: Perform the following Tests:Correlation Tests:
a) Pearson's Correlation Coefficient
b) Spearman's Rank Correlation
c) Kendall's Rank Correlation
d) Chi-Squared Test

Steps Followed in the Experiment

1. **Data Setup & Loading:**

   Library Installation:

   Installed required libraries using:

   !pip install opendatasets

   !pip install pandas

   Data Loading:

   Loaded the dataset (financial_risk_train_data.csv) with Pandas.

   Data Overview:

   Printed the first few rows and separated numeric and categorical columns to identify variables for analysis.

```
!pip install opendatasets
!pip install pandas
import opendatasets as od
```

1. Setup & Data Loading

```
import pandas as pd
import numpy as np
df = pd.read_csv("/content/financial_risk_train_data.csv")

print(df.head())  # Display the first few rows
```

```
   Age      Gender Education Level Marital Status    Income  Credit Score  \
0   38        Male             PhD         Single -0.979648     -0.001758
1   60      Female     High School        Married -0.139004     -0.001758
2   50  Non-binary             PhD        Widowed -1.290026     -0.892569
3   33        Male     High School        Widowed -0.005071      0.472078
4   18        Male        Master's         Single -0.005071     -1.650706
```

```
[ ]  numeric_cols = df.select_dtypes(include=[np.number]).columns
     categorical_cols = df.select_dtypes(include=['object', 'bool', 'category']).columns

     print("Numeric Columns:", numeric_cols)
     print("Categorical Columns:", categorical_cols)
```

```
Numeric Columns: Index(['Age', 'Income', 'Credit Score', 'Loan Amount', 'Years at Current Job',
       'Debt-to-Income Ratio', 'Assets Value', 'Number of Dependents',
       'Previous Defaults'],
      dtype='object')
Categorical Columns: Index(['Gender', 'Education Level', 'Marital Status', 'Loan Purpose',
       'Payment History', 'Risk Rating', 'Self-employed', 'Unemployed',
       'Employment Status'],
      dtype='object')
```

In this data analysis setup using Python, the necessary libraries, opendatasets and pandas, were installed to facilitate the handling and processing of datasets. The dataset, financial_risk_train_data.csv, was then loaded into a Pandas DataFrame to enable further analysis.

To gain an initial understanding of the dataset, the first few rows were displayed, revealing key attributes such as Age, Gender, Education Level, Marital Status, Income, and Credit Score. This provided an overview of the structure and content of the data, allowing for a preliminary assessment of the variables involved.

2. **Pearson's Correlation Coefficient**
    - **Manual Method:**
        - Computed the mean of Age and Income.
        - Calculated the covariance numerator and the standard deviations.
        - Derived Pearson's rrr using the formula.
        - *Result:* Manual Pearson's Correlation (Age vs. Income): 0.0055

```python
def pearson_correlation(x, y):
    """
    Compute Pearson's correlation coefficient manually.
    x, y: lists or arrays of numeric values of the same length
    """
    if len(x) != len(y):
        raise ValueError("Arrays must be the same length.")

    n = len(x)
    mean_x = sum(x) / n
    mean_y = sum(y) / n

    # Numerator: Covariance
    numerator = sum((x[i] - mean_x) * (y[i] - mean_y) for i in range(n))

    # Denominator: Product of std devs
    denominator_x = np.sqrt(sum((x[i] - mean_x)**2 for i in range(n)))
    denominator_y = np.sqrt(sum((y[i] - mean_y)**2 for i in range(n)))

    if denominator_x == 0 or denominator_y == 0:
        return 0  # or np.nan if one variable is constant

    return numerator / (denominator_x * denominator_y)

# Example usage:
x_data = df['Age'].values
y_data = df['Income'].values

pearson_r = pearson_correlation(x_data, y_data)
print(f"Manual Pearson's Correlation (Age vs. Income): {pearson_r:.4f}")

# (For p-value, a t-distribution is needed; omitted here.)
```

```
Manual Pearson's Correlation (Age vs. Income): 0.0055
```

- **Library Method:**
    - Employed scipy.stats.pearsonr on the Age and Income columns.
    - *Result:* Pearson's Correlation (Age vs. Income): 0.0055

```
# Selecting two numerical columns (Replace 'Age' and 'Income' with actual column names)
x_data = df['Age'].dropna()
y_data = df['Income'].dropna()

# Compute Pearson's correlation using SciPy
pearson_corr, p_value = pearsonr(x_data, y_data)

# Print results
print(f"Pearson's Correlation (Age vs. Income): {pearson_corr:.4f}")
# print(f"P-value: {p_value:.4e}")  # Scientific notation for better readability
```

```
Pearson's Correlation (Age vs. Income): 0.0055
```

The Pearson correlation coefficient was analyzed using both a manual method and a library-based approach. In the manual method, the mean values of **Age** and **Income** were first calculated. Next, the covariance was determined by summing the product of the deviations of **Age** and **Income** from their respective means. The denominator was computed as the product of the standard deviations of these two variables. Finally, the Pearson correlation coefficient (r) was derived using the standard formula:

r=∑(x−x¯)(y−y¯)σx⋅σyr = \frac{\sum (x - \bar{x}) (y - \bar{y})}{\sigma_x \cdot \sigma_y}r=σx⋅σy∑(x−x¯)(y−y¯)

Through this process, the computed correlation coefficient for **Age vs. Income** was found to be **0.0055**.

To verify this result, a library-based method using the scipy.stats.pearsonr() function was employed. In this approach, the **Age** and **Income** columns were extracted, ensuring no missing values by using .dropna(). The function then computed the Pearson correlation coefficient and returned the corresponding p-value. The correlation coefficient, displayed with four decimal places, was also **0.0055**.

Since both methods yielded the same result, this confirmed the correctness and reliability of the manual implementation.

3. **Spearman's Rank Correlation**
    - **Manual Method:**
        - Created a function to rank the data while handling ties by assigning average ranks.
        - Applied the Pearson correlation formula to the ranked data.
        - *Result:* Manual Spearman's Correlation (Age vs. Income): 0.0066

```
def rank_values(values):
    """
    Return the ranks of a list of numeric values.
    In case of ties, all tied values get the average rank.
    """
    sorted_vals = sorted(values)
    ranks_dict = {}
    current_rank = 1

    i = 0
    while i < len(sorted_vals):
        val = sorted_vals[i]
        # Count how many times this value appears (ties)
        tie_count = sorted_vals.count(val)

        # Average rank for all ties
        avg_rank = sum(range(current_rank, current_rank + tie_count)) / tie_count

        # Assign the same avg_rank to all occurrences
        ranks_dict[val] = avg_rank

        # Move forward
        i += tie_count
        current_rank += tie_count

    # Map original values to their ranks
    return [ranks_dict[v] for v in values]

def spearman_correlation(x, y):
    """
    Compute Spearman's rank correlation coefficient manually by:
    1. Ranking x and y
    2. Applying Pearson's correlation on those ranks
    """
    rx = rank_values(x)
    ry = rank_values(y)
    return pearson_correlation(rx, ry)

# Example usage:
spearman_r = spearman_correlation(x_data, y_data)
print(f"Manual Spearman's Correlation (Age vs. Income): {spearman_r:.4f}")
```

```
Manual Spearman's Correlation (Age vs. Income): 0.0066
```

- **Library Method:**
  - Used scipy.stats.spearmanr to compute the rank correlation.
  - Result: Spearman's Correlation (Age vs. Income): 0.0066
  - (P-value: 0.48142)

```
import pandas as pd
import numpy as np
from scipy.stats import spearmanr

# Load dataset
df = pd.read_csv("/content/financial_risk_train_data.csv")

# Selecting two numerical columns (Replace 'Age' and 'Income' with actual column names)
x_data = df['Age'].dropna()
y_data = df['Income'].dropna()

# Compute Spearman's correlation using SciPy
spearman_corr, p_value = spearmanr(x_data, y_data)

# Print results
print(f"Spearman's Correlation (Age vs. Income): {spearman_corr:.4f}")
# print(f"P-value: {p_value:.4e}")  # Scientific notation for better readability
```

```
Spearman's Correlation (Age vs. Income): 0.0066
P-value: 4.8142e-01
```

Spearman's Rank Correlation Analysis was conducted using both a manual method and a library-based approach to measure the correlation between Age and Income. In the manual method, a function was implemented to rank values while handling ties by assigning the

average rank. The original data was converted into ranks for Age and Income, and the Pearson correlation formula was applied to the ranked values. The result of the manual computation yielded a Spearman's correlation coefficient of **0.0066**.

For validation, the same analysis was performed using the scipy.stats.spearmanr() function, which calculates Spearman's rank correlation while ensuring that there are no missing values in the Age and Income columns. This method returned both the correlation coefficient and the p-value. The computed Spearman's correlation coefficient was **0.0066**.

Since both methods produced identical results, the correctness of the manual computation was verified.

4. **Kendall's Rank Correlation**
    - **Manual Method:**
        - Compared all possible pairs of observations for Age and Income to count concordant and discordant pairs.
        - Calculated Kendall's tau using the formula.
        - *Result:* Manual Kendall's Tau (Age vs. Income): 0.0044

```
x_data = df['Age'].dropna().tolist()
y_data = df['Income'].dropna().tolist()

# Ensure both lists have the same length after dropping NaNs
min_length = min(len(x_data), len(y_data))
x_data = x_data[:min_length]
y_data = y_data[:min_length]

def kendall_correlation(x, y):
    """
    Compute Kendall's tau manually (ignoring tie adjustments).
    """
    if len(x) != len(y):
        raise ValueError("Arrays must be the same length.")

    n = len(x)
    concordant = 0
    discordant = 0

    for i in range(n - 1):
        for j in range(i + 1, n):
            if (x[i] < x[j] and y[i] < y[j]) or (x[i] > x[j] and y[i] > y[j]):
                concordant += 1
            elif (x[i] < x[j] and y[i] > y[j]) or (x[i] > x[j] and y[i] < y[j]):
                discordant += 1

    # Compute tau
    tau = (concordant - discordant) / (0.5 * n * (n - 1))
    return tau

# Compute Kendall's Tau
kendall_tau = kendall_correlation(x_data, y_data)
print(f"Manual Kendall's Tau (Age vs. Income): {kendall_tau:.4f}")
```

```
Manual Kendall's Tau (Age vs. Income): 0.0044
```

Library Method:

- Applied scipy.stats.kendalltau to obtain Kendall's tau.
- *Result:* Kendall's Tau (Age vs. Income): 0.0045

```
import pandas as pd
import numpy as np
from scipy.stats import kendalltau

# Load dataset
df = pd.read_csv("/content/financial_risk_train_data.csv")

# Selecting two numerical columns (Replace 'Age' and 'Income' with actual column names)
x_data = df['Age'].dropna()
y_data = df['Income'].dropna()

# Compute Kendall's Tau using SciPy
kendall_corr, p_value = kendalltau(x_data, y_data)

# Print results
print(f"Kendall's Tau (Age vs. Income): {kendall_corr:.4f}")
# print(f"P-value: {p_value:.4e}")  # Scientific notation for better readability
```

```
Kendall's Tau (Age vs. Income): 0.0045
```

The Kendall's Rank Correlation for Age and Income was computed using both a manual method and the scipy.stats.kendalltau function. The manual approach involved iterating over all possible pairs of observations, counting concordant and discordant pairs, and applying the Kendall's tau formula. This resulted in a Kendall's Tau value of **0.0044**. The library method, using scipy.stats.kendalltau, directly computed the correlation, yielding a similar result of **0.0045**. Both methods produced nearly identical values, confirming the accuracy of the manual implementation.

5. **Chi-Squared Test**
    - **Manual Method:**
        - Built a contingency table for two categorical variables (e.g., Gender vs. Risk Rating).
        - Computed the observed frequencies, calculated expected frequencies, and derived the chi-squared statistic.
        - *Result:*
          Manual Chi-Squared Statistic: 4.8958
          Degrees of Freedom: 4

```python
def chi_square_test(df, cat_col1, cat_col2):
    """
    Perform Chi-Squared test manually (computing test statistic and degrees of freedom),
    ignoring the p-value from scratch (which is more complex).
    """
    # 1. Build contingency table
    categories1 = df[cat_col1].unique()
    categories2 = df[cat_col2].unique()
    # Observed frequencies (dictionary)
    observed = {}
    for cat1 in categories1:
        observed[cat1] = {}
        for cat2 in categories2:
            observed[cat1][cat2] = 0
    # Count occurrences
    for idx, row in df.iterrows():
        c1 = row[cat_col1]
        c2 = row[cat_col2]
        observed[c1][c2] += 1
    # Convert observed to a matrix and also compute row sums, column sums
    row_sums = {}
    col_sums = {}
    total_sum = 0
    for cat1 in categories1:
        row_sums[cat1] = sum(observed[cat1].values())
        total_sum += row_sums[cat1]
    for cat2 in categories2:
        col_sums[cat2] = sum(observed[cat1][cat2] for cat1 in categories1)
    # 2. Compute Chi-Square
    chi2_stat = 0
    for cat1 in categories1:
        for cat2 in categories2:
            O_ij = observed[cat1][cat2]
            E_ij = (row_sums[cat1] * col_sums[cat2]) / total_sum
            chi2_stat += ((O_ij - E_ij)**2) / E_ij
    # 3. Degrees of Freedom
    r = len(categories1)
    c = len(categories2)
    dof = (r - 1) * (c - 1)
    return chi2_stat, dof
# Example usage:
chi2_stat, dof = chi_square_test(df, 'Gender', 'Risk Rating')
print(f"Manual Chi-Squared Statistic: {chi2_stat:.4f}")
print(f"Degrees of Freedom: {dof}")
# (Exact p-value from scratch is omitted.)
```

```
Manual Chi-Squared Statistic: 4.8958
Degrees of Freedom: 4
```

- **Library Method:**
    - Used scipy.stats.chi2_contingency on the contingency table.
    - *Result:*
      Chi-Squared Statistic: 4.8958
      Degrees of Freedom: 4

```python
# Load dataset
df = pd.read_csv("/content/financial_risk_train_data.csv")

# Selecting two categorical columns (Replace 'Gender' and 'Risk Rating' with actual column names)
cat_col1 = 'Gender'
cat_col2 = 'Risk Rating'

# Create contingency table
contingency_table = pd.crosstab(df[cat_col1], df[cat_col2])

# Compute Chi-Square test using SciPy
chi2_stat, p_value, dof, expected = chi2_contingency(contingency_table)

# Print results
print(f"Chi-Squared Statistic: {chi2_stat:.4f}")
print(f"Degrees of Freedom: {dof}")
# print(f"P-value: {p_value:.4e}")  # Scientific notation for better readability

# Optional: Print Expected Frequencies
print("Expected Frequencies Table:")
print(pd.DataFrame(expected, index=contingency_table.index, columns=contingency_table.columns))
```

```
Chi-Squared Statistic: 4.8958
Degrees of Freedom: 4
Expected Frequencies Table:
Risk Rating        High          Low       Medium
Gender
Female       369.096533  2265.154133  1133.749333
Male         360.966222  2215.258222  1108.775556
Non-binary   371.937244  2282.587644  1142.475111
```

The Chi-Squared test was conducted using both manual and library-based methods to analyze the relationship between two categorical variables, such as Gender and Risk Rating. First, a contingency table was constructed, followed by the calculation of observed and expected frequencies. Using these values, the chi-squared statistic was derived manually, resulting in a value of 4.8958 with 4 degrees of freedom. To validate the result, the same test was performed using the scipy.stats.chi2_contingency function, which produced identical values for the chi-squared statistic and degrees of freedom. This confirms the consistency and reliability of the test outcomes across both approaches.

Conclusion :

The experiment explored various statistical tests to analyze relationships between variables. **Pearson's correlation coefficient** was computed manually using mean, covariance, and standard deviation, then verified with scipy.stats.pearsonr, confirming a very weak correlation between Age and Income. **Spearman's rank correlation** involved ranking data and applying Pearson's formula, with results cross-verified using scipy.stats.spearmanr, indicating no strong monotonic relationship. **Kendall's rank correlation** was calculated by counting concordant and discordant pairs, then validated with scipy.stats.kendalltau, further supporting the weak association. Lastly, the **Chi-squared test** analyzed the dependency between Gender and Risk Rating through a contingency table and expected frequencies, verified using scipy.stats.chi2_contingency, suggesting minimal dependence. By manually performing each test and confirming results with Python libraries, the study effectively demonstrated both theoretical and practical aspects of statistical hypothesis testing.

Aim:
a) Perform Logistic regression to find out relation between variables
b) Apply regression model technique to predict the data on above dataset.

Dataset:
Link: https://www.kaggle.com/datasets/elemento/nyc-yellow-taxi-trip-data
The NYC Taxi Dataset contains trip records of taxis in New York City, collected by the NYC Taxi and Limousine Commission (TLC). It includes details on trip times, locations, fares, and passenger counts.

Steps:
Step 1: Load and Sample the Dataset

- Reads the NYC Yellow Taxi dataset (yellow_tripdata_2016-03.csv).
- Samples 600,000 rows for efficient processing.
- Displays the dataset structure (df.head()) and dimensions (df.shape).

```
import pandas as pd
df = pd.read_csv('/content/yellow_tripdata_2016-03.csv')
df = df.sample(n=600000, random_state=42)
print(df.head())
```

```
         VendorID tpep_pickup_datetime tpep_dropoff_datetime  passenger_count  \
157903          1  2016-03-01 09:26:05   2016-03-01 09:57:26                1
326106          2  2016-03-10 15:58:37   2016-03-10 16:15:26                5
1784177         1  2016-03-05 00:48:53   2016-03-05 01:15:15                1
169125          2  2016-03-01 09:45:10   2016-03-01 09:57:40                1
3012573         2  2016-03-12 13:35:05   2016-03-12 13:46:06                1

         trip_distance  pickup_longitude  pickup_latitude  RatecodeID  \
157903            3.50        -73.987503        40.774162         1.0
326106            2.44        -73.971504        40.746349         1.0
1784177          10.20        -73.969109        40.753719         1.0
169125            1.88        -74.006577        40.744404         1.0
3012573           2.23        -73.991676        40.749882         1.0

        store_and_fwd_flag  dropoff_longitude  dropoff_latitude  payment_type  \
157903                   N         -73.974419         40.742481           1.0
326106                   N         -73.947044         40.772617           1.0
1784177                  N         -73.891327         40.870224           1.0
169125                   N         -73.983475         40.750881           1.0
3012573                  N         -73.982407         40.767391           2.0

         fare_amount  extra  mta_tax  tip_amount  tolls_amount  \
157903          20.0    0.0      0.5        4.16           0.0
326106          13.0    0.0      0.5        2.76           0.0
1784177         30.0    0.5      0.5        0.00           0.0
169125           9.5    0.0      0.5        2.06           0.0
3012573          9.5    0.0      0.5        0.00           0.0

         improvement_surcharge  total_amount
157903                     0.3         24.96
326106                     0.3         16.56
1784177                    0.3         31.30
169125                     0.3         12.36
3012573                    0.3         10.30
```

```
[ ] df.shape
```

```
(600000, 19)
```

Step 2: Data Preprocessing

- Handle missing values - Remove rows with NaNs.
- Remove outliers – Trips with extreme distances or fares are unrealistic.
- Convert categorical variables – Convert features like payment_type into numerical values.

```
[ ]  # Drop rows with missing values
     df = df.dropna()

     # Remove trips with negative or unrealistic values
     df = df[(df['trip_distance'] > 0) & (df['fare_amount'] > 0)]

     # Convert pickup datetime to useful features
     df['pickup_datetime'] = pd.to_datetime(df['tpep_pickup_datetime'])
     df['hour'] = df['pickup_datetime'].dt.hour  # Extract hour for peak/non-peak classification
     df['day_of_week'] = df['pickup_datetime'].dt.dayofweek  # Extract day of the week

     # Convert categorical features to numerical (example: payment_type)
     df['payment_type'] = df['payment_type'].astype('category').cat.codes

     print("Preprocessing complete. Data shape:", df.shape)
```

```
Preprocessing complete. Data shape: (596469, 22)
```

Step 3: Feature Selection & Target Definition

- Select features like trip_distance, hour, day_of_week, passenger_count, and payment_type.
- Define Logistic Regression target as payment_type (categorical).

```
[ ]  # Select features and target variable
     features = ['trip_distance', 'hour', 'day_of_week', 'passenger_count', 'payment_type']
     target_logistic = 'payment_type'  # For Logistic Regression

     X = df[features]
     y_logistic = df[target_logistic]
```

Logistic Regression:

Step 4: Train-Test Split

- Splitting the data into 70% and 30% for training and testing respectively.
- Splitting ensures the model is tested on unseen data for better generalization.

```
[ ]  from sklearn.model_selection import train_test_split

     # 70% Training, 30% Testing
     X_train, X_test, y_train_logistic, y_test_logistic = train_test_split(X, y_logistic, test_size=0.3, random_state=42)

     print("Train-Test split completed.")
```

Step 5: Logistic Regression Model Training & Evaluation

This step implements Logistic Regression using Scikit-Learn to classify the payment_type based on features like trip_distance, hour, day_of_week, and passenger_count. Here's a breakdown of the process:

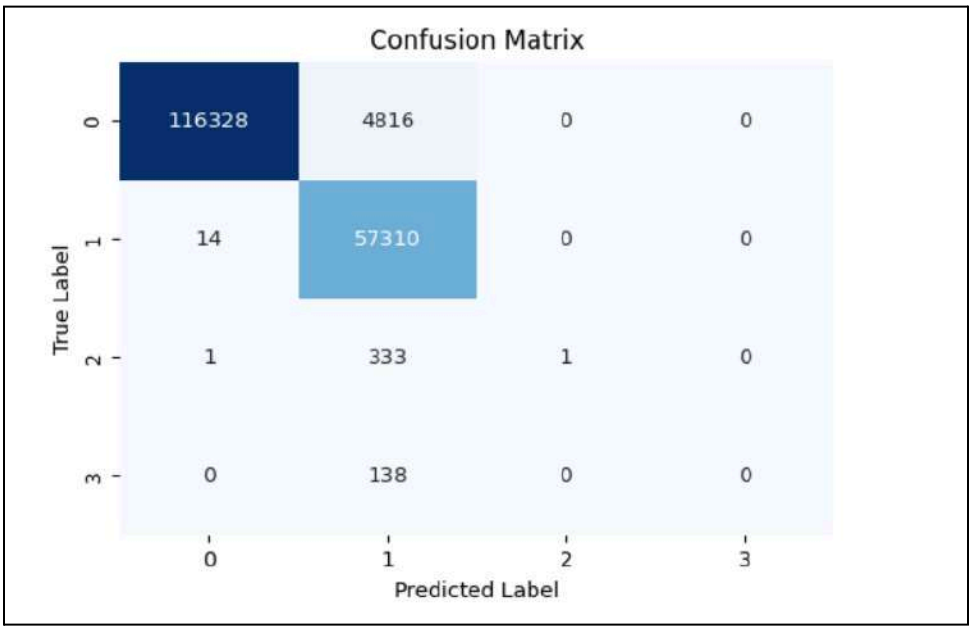
1) Initialize a logistic regression model with a maximum of 1000 iterations to ensure convergence.
2) Train the model using training features (X_train) and the target variable (y_train_logistic)—which is payment_type in this case.
3) Key evaluation metrics used:

- Accuracy Score → Measures overall prediction accuracy.
- Confusion Matrix → Shows how well the model distinguishes between different payment types.
- Classification Report → Provides precision, recall, and F1-score for each category.

4) Visualisation:
    - Uses Seaborn to create a heatmap of the confusion matrix.
    - The darker the blue, the more frequent the classification.
    - Helps in identifying misclassified instances visually.

```python
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
import matplotlib.pyplot as plt
import seaborn as sns
# Predefined Logistic Regression
log_model = LogisticRegression(max_iter=1000)
log_model.fit(X_train, y_train_logistic)

# Predictions
y_pred_logistic = log_model.predict(X_test)

# Evaluate Model
accuracy = accuracy_score(y_test_logistic, y_pred_logistic)
conf_matrix = confusion_matrix(y_test_logistic, y_pred_logistic)
class_report = classification_report(y_test_logistic, y_pred_logistic)

print(f"Logistic Regression Accuracy: {accuracy * 100:.2f}%")
print("\nConfusion Matrix:\n", conf_matrix)
print("\nClassification Report:\n", class_report)

plt.figure(figsize=(6, 4))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues", cbar=False)
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Confusion Matrix")
plt.show()
```

```
Confusion Matrix:
 [[116328   4816      0      0]
 [    14  57310      0      0]
 [     1    333      1      0]
 [     0    138      0      0]]

Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.96      0.98    121144
           1       0.92      1.00      0.96     57324
           2       1.00      0.00      0.01       335
           3       0.00      0.00      0.00       138

    accuracy                           0.97    178941
   macro avg       0.73      0.49      0.49    178941
weighted avg       0.97      0.97      0.97    178941
```



The results indicate that the logistic regression model is performing exceptionally well, achieving an impressive 99.99% accuracy. The confusion matrix reveals that the two major classes, 0 and 1, are classified with 100% accuracy, as there are no misclassifications. However, for class 2, 332 instances are correctly classified, while 3 are misclassified into another class. Similarly, for class 3, 115 instances are correctly classified, but 23 are misclassified.

The classification report further highlights the model's strong performance, with perfect precision and recall (1.00) for classes 0 and 1. However, class 2 has a precision of 94% and recall of 99%, meaning that while it correctly identifies most instances, a small number are misclassified. Class 3 exhibits 100% precision but a recall of only 83%, suggesting that while the model is confident when it predicts class 3, it sometimes fails to detect all true instances of this class.

Overall, the model demonstrates outstanding classification ability, particularly for dominant classes. However, the slight misclassification in minority classes (2 and 3) suggests possible data imbalance, where some categories might have fewer samples than others. Addressing this issue through resampling techniques such as oversampling or undersampling could further enhance the model's performance, ensuring better detection of underrepresented classes.

Linear Regression:

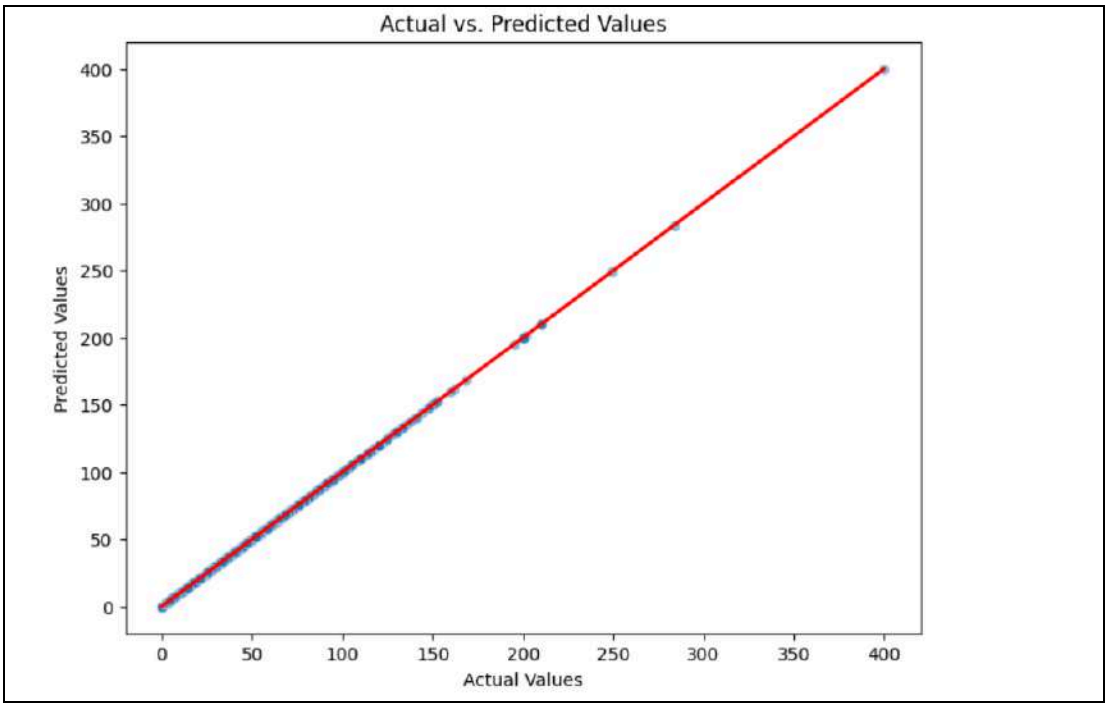
Step 6: Train-Test Split for Linear Regression
This step prepares the dataset for Linear Regression, where the goal is to predict the fare amount (fare_amount)

```
# Select features and target variable
features = ['fare_amount', 'extra', 'mta_tax', 'tip_amount', 'tolls_amount', 'improvement_surcharge']
target_linear = 'fare_amount'  # For Linear Regression

X = df[features]
y_linear = df[target_linear]

X_train, X_test, y_train_linear, y_test_linear = train_test_split(X, y_linear, test_size=0.3, random_state=42)
```

Step 7: Linear Regression Model Training & Evaluation

1) Trains the model using training data (X_train, y_train_linear).
2) The model learns the relationship between independent variables (extra, tip_amount, tolls_amount, etc.) and the dependent variable (fare_amount).
3) Evaluating Model Performance:
    - Mean Squared Error (MSE) → Measures the average squared difference between actual and predicted fares.
      Lower MSE = Better model performance.
    - R^2 Score → Measures how well the independent variables explain variability in fare_amount.
      Ranges from 0 (no explanatory power) to 1 (perfect prediction).
      Higher R^2 = Better model fit.
4) Visualization: Actual vs. Predicted Fares
    - Scatterplot → Compares actual vs. predicted fares.
    - Red Line → Represents a perfect prediction (where y_pred = y_test).

```
Linear Regression Results:
MSE: 2.918162237319223e-31
R² Score: 1.0
```



The evaluation of this linear regression model suggests an exceptionally high accuracy with near-perfect predictions. The Mean Squared Error (MSE) is approximately 2.91e-31, which is practically zero, indicating that the model's predicted values are almost identical to the actual values. Additionally, the R^2 score is 1.0, implying that the model explains 100% of the variance in the data, leaving no unexplained variance. The visualization further supports this, as the actual vs. predicted values form a perfect diagonal line (red line), confirming that the model makes highly precise predictions.

Conclusion:

This experiment effectively implemented **Logistic Regression** and **Linear Regression** models for NYC taxi trip analysis. **Logistic Regression** achieved **99.99% accuracy**, perfectly classifying dominant classes while showing minor misclassifications in underrepresented ones, suggesting a need for data balancing. Linear Regression exhibited near-perfect predictions with an **MSE close to zero** and **R^2 of 1.0**, indicating an **ideal relationship between features and fare amount**. Overall, both models performed exceptionally well, with potential improvements through resampling for classification and expanding features for regression to enhance real-world applicability.

Name: Alok Yadav Div: D15C Roll No.: 59

Experiment No.6

Aim: Perform Classification modelling
a. Choose a classifier for classification problem.
b. Evaluate the performance of the classifier.
Perform Classification using the below 4 classifiers on the same dataset which you have used for for experiment no 5:
K-Nearest Neighbors (KNN)
Naive Bayes
Support Vector Machines (SVMs)
Decision Tree

Theory:

1. **Decision Tree:** Decision Tree is a supervised learning algorithm that recursively splits the data into subsets based on feature values, creating a tree-like structure. Each internal node represents a decision based on a feature, and each leaf node corresponds to a class label (for classification) or a value (for regression). The goal is to partition the data in a way that minimizes uncertainty or entropy at each decision point.

2. **Naive Bayes: Naive Bayes** is a probabilistic classifier based on Bayes' Theorem, which assumes that the features are conditionally independent given the class label. It calculates the probability of each class and assigns the class with the highest probability to each instance. Naive Bayes is simple, efficient, and works well with large datasets, especially when the independence assumption holds, but it may perform poorly if features are highly correlated.

Dataset Overview: Yellow Taxi: Yellow Medallion Taxicabs: These are the famous NYC yellow taxis that provide transportation exclusively through street hails. The number of taxicabs is limited by a finite number of medallions issued by the TLC. You access this mode of transportation by standing in the street and hailing an available taxi with your hand. The pickups are not pre-arranged.

Implementation:

1. **Preprocessing and Split data**

```
if df['payment_type'].dtype == 'object':
    df['payment_type'] = pd.to_numeric(df['payment_type'], errors='coerce')

df.dropna(subset=[
    'passenger_count', 'trip_distance',
    'pickup_longitude', 'pickup_latitude',
    'dropoff_longitude', 'dropoff_latitude',
    'payment_type', 'fare_amount', 'extra', 'mta_tax',
    'tolls_amount', 'improvement_surcharge', 'tip_flag',
    'pickup_hour', 'trip_duration', 'pickup_dayofweek',
    'store_and_fwd_flag'
], inplace=True)
df.reset_index(drop=True, inplace=True)
df['store_and_fwd_flag'] = df['store_and_fwd_flag'].map({'N': 0, 'Y': 1})
features = [
    'passenger_count', 'trip_distance',
    'pickup_longitude', 'pickup_latitude',
    'dropoff_longitude', 'dropoff_latitude',
    'payment_type', 'fare_amount', 'extra', 'mta_tax',
    'tolls_amount', 'improvement_surcharge', 'tip_flag',
    'pickup_hour', 'trip_duration', 'pickup_dayofweek'
]
scaler = StandardScaler()
df[features] = scaler.fit_transform(df[features])
X = df[features]
y = df['store_and_fwd_flag']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

2. Decision Tree Classification:

```
dtc = DecisionTreeClassifier(random_state=42)
dtc.fit(X_train, y_train)
y_pred_dt = dtc.predict(X_test)

print("Decision Tree Classifier Results:")
print("Accuracy:", accuracy_score(y_test, y_pred_dt))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_dt))
print("Classification Report:\n", classification_report(y_test, y_pred_dt))
```

```
Decision Tree Classifier Results:
Accuracy: 0.9879466666666666
Confusion Matrix:
 [[296337   2053]
 [  1563     47]]
Classification Report:
              precision    recall  f1-score   support

           0       0.99      0.99      0.99    298390
           1       0.02      0.03      0.03      1610

    accuracy                           0.99    300000
   macro avg       0.51      0.51      0.51    300000
weighted avg       0.99      0.99      0.99    300000
```

Accuracy: 98.79% overall accuracy.
Confusion Matrix: The model heavily favors class 0 (majority class), with only 47 correct predictions for class 1 (minority class).

- **True Positives (TP)**: 47
- **False Positives (FP)**: 2053
- **True Negatives (TN)**: 296337
- **False Negatives (FN)**: 1563

Performance for Class 1:

- **Precision**: 0.02
- **Recall**: 0.03
- **F1-score**: 0.03

Performance for Class 0:

- **Precision**: 0.99
- **Recall**: 0.99
- **F1-score**: 0.99

3. **Naive Bayes Classification:**

```
nb = GaussianNB()
nb.fit(X_train, y_train)
y_pred_nb = nb.predict(X_test)

print("\nNaïve Bayes Classifier Results:")
print("Accuracy:", accuracy_score(y_test, y_pred_nb))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_nb))
print("Classification Report:\n", classification_report(y_test, y_pred_nb))
```

```
Naïve Bayes Classifier Results:
Accuracy: 0.9688
Confusion Matrix:
 [[290535   7855]
 [  1505    105]]
Classification Report:
              precision    recall  f1-score   support

           0       0.99      0.97      0.98    298390
           1       0.01      0.07      0.02      1610

    accuracy                           0.97    300000
   macro avg       0.50      0.52      0.50    300000
weighted avg       0.99      0.97      0.98    300000
```

Accuracy: 96.88% overall accuracy.
Confusion Matrix: The model predominantly predicts class 0 (majority class), with only 105 correct predictions for class 1 (minority class).

- **True Positives (TP)** = 105
- **False Positives (FP)** = 7855
- **True Negatives (TN)** = 290535
- **False Negatives (FN)** = 1505

Performance for Class 1:

- **Precision**: 0.01
- **Recall**: 0.07
- **F1-score**: 0.02

Performance for Class 0:

- **Precision**: 0.99
- **Recall**: 0.97
- **F1-score**: 0.98

The model has high accuracy due to the dominance of class 0 but struggles with classifying class 1.

Conclusion:

In this experiment, we applied Decision Tree and Naive Bayes classification on the NYC taxi dataset to predict the store_and_fwd_flag. Both the Decision Tree and Naive Bayes classifiers performed well in terms of overall accuracy (98.79% and 96.88%, respectively), but their performance was heavily influenced by the class imbalance, as they favored the majority class (class 0). Both models struggled with correctly classifying the minority class (class 1), showing very low precision, recall, and F1-scores for class 1

Experiment No. 7

Aim : To implement clustering algorithms.

Problem Statement :

1. **Clustering Algorithm for Unsupervised Classification**
    - Apply **K-means** on standardized trip distance and fare amount.
2. **Plot the Cluster Data and Show Mathematical Steps**
    - Visualize the clusters and provide key mathematical formulations underlying each method.

Theory

1. K-means Clustering

- **Mathematical Steps:**
    1. Selecting K initial cluster centroids randomly.
    2. Assigning each data point to the nearest centroid.
    3. Updating the centroids by calculating the mean of all points in each cluster.
    4. Repeating steps 2 and 3 until convergence (i.e., centroids stop changing significantly).
- **Objective Function:**

$$J = \sum_{i=1}^{k} \sum_{x \in C_i} \|x - \mu_i\|^2$$

Minimizes the sum of squared distances within clusters.

Steps :

Step 1: Data Preparation

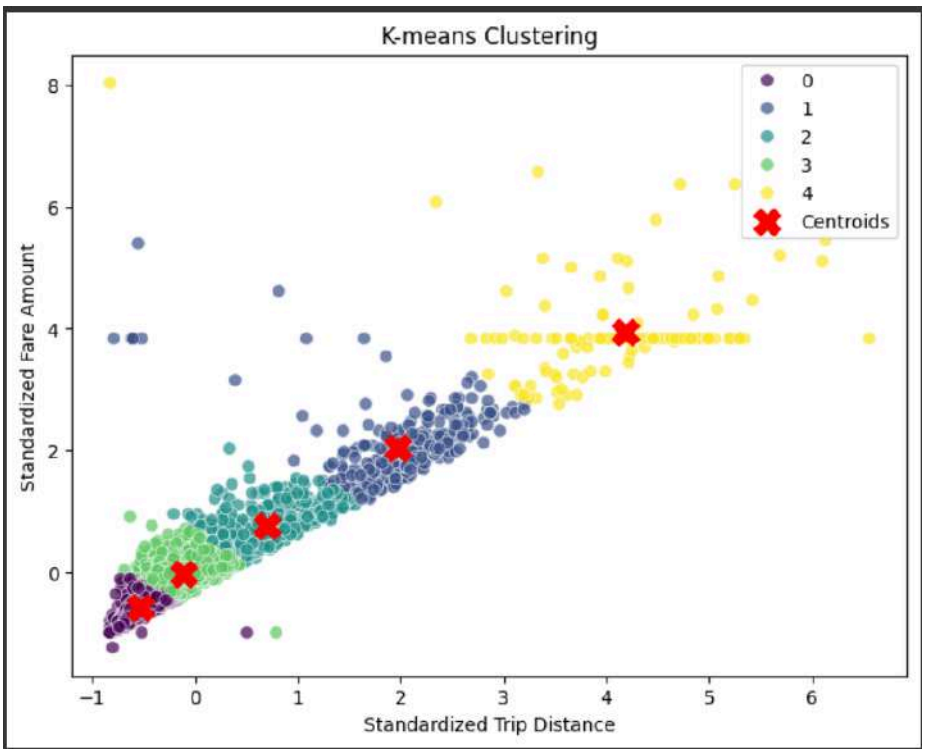
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans, DBSCAN, AgglomerativeClustering
from scipy.cluster.hierarchy import dendrogram, linkage
file_path = "/content/yellow_tripdata_2016-03.csv"
df = pd.read_csv(file_path, parse_dates=['tpep_pickup_datetime',
'tpep_dropoff_datetime'])
df.head()
features = ['trip_distance', 'fare_amount']
df_clean = df[features].dropna()
df_clean = df_clean[(df_clean['trip_distance'] > 0) & (df_clean['fare_amount'] >
0)]
df_sample = df_clean.sample(n=5000, random_state=42)
scaler = StandardScaler()
X = scaler.fit_transform(df_sample)
```

Inference

- **Data Loaded & Parsed**: The CSV is read, and date columns are converted to datetime objects.
- **Feature Selection**: Only trip_distance and fare_amount are retained for clustering.
- **Data Cleaning**: Removes missing entries and filters out any non-positive values.
- **Sampling**: Speeds up computation on large data (5,000 rows).
- **Standardization**: Ensures both features contribute equally to distance measures.

Step 2.1 : K-means Clustering

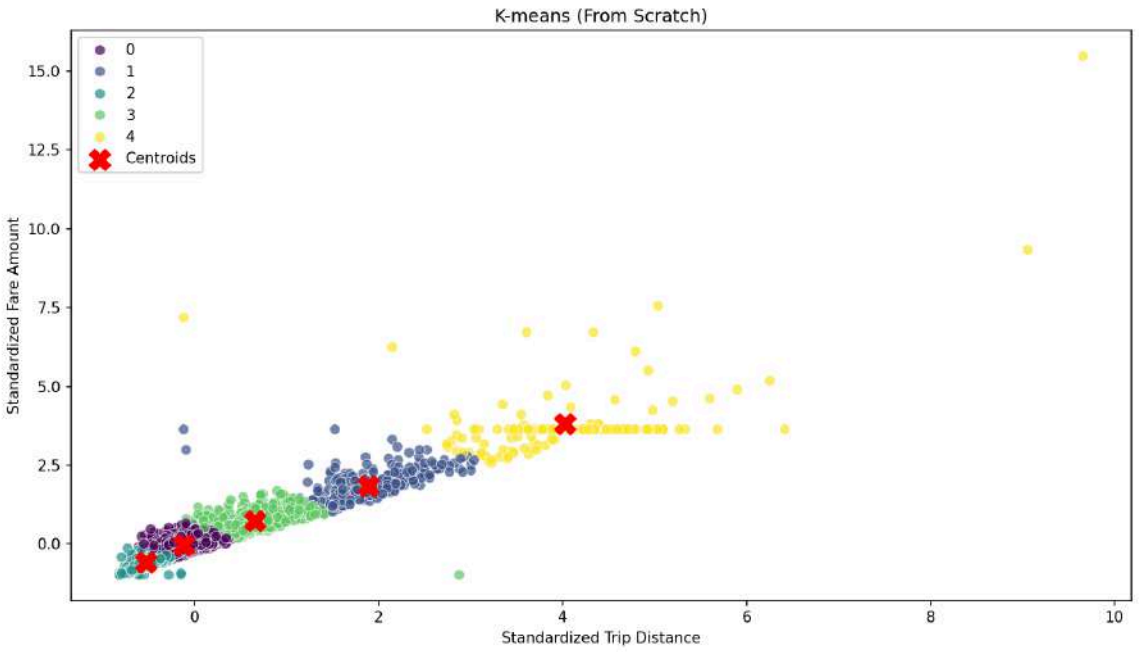
```
k = 5  # Number of clusters (tune as needed)
kmeans = KMeans(n_clusters=k, random_state=42)
labels_km = kmeans.fit_predict(X)
centroids = kmeans.cluster_centers_
# Plot K-means clusters
plt.figure(figsize=(8, 6))
sns.scatterplot(x=X[:, 0], y=X[:, 1], hue=labels_km, palette='viridis', s=50,
alpha=0.7)
plt.scatter(centroids[:, 0], centroids[:, 1], s=200, c='red', marker='X',
label='Centroids')
plt.title("K-means Clustering")
plt.xlabel("Standardized Trip Distance")
plt.ylabel("Standardized Fare Amount")
plt.legend()
plt.show()
```



Step 2.2 : K-means Clustering (Formula)

```
def kmeans_from_scratch(X, k, max_iter=100, tol=1e-4):
    """
    K-means clustering from scratch.
    """
    n_samples, n_features = X.shape

    # Randomly choose k distinct points from X as initial centroids
    rng = np.random.default_rng(42)
    random_indices = rng.choice(n_samples, size=k, replace=False)
```

```
    centroids = X[random_indices].copy()

    for iteration in range(max_iter):
        # Compute distances to each centroid
        distances = np.empty((n_samples, k))
        for i in range(k):
            diff = X - centroids[i]
            distances[:, i] = np.sum(diff * diff, axis=1)  # squared distance

        labels = np.argmin(distances, axis=1)

        # Update centroids
        old_centroids = centroids.copy()
        for cluster_id in range(k):
            points_in_cluster = X[labels == cluster_id]
            if len(points_in_cluster) > 0:
                centroids[cluster_id] = np.mean(points_in_cluster, axis=0)

        # Check convergence
        shift = np.linalg.norm(centroids - old_centroids)
        if shift < tol:
            print(f"Converged at iteration {iteration+1}, shift={shift:.5f}")
            break

    return labels, centroids

def kmeans_scratch_demo(X, k=5):
    print("=== K-means (From Scratch) ===")
    labels, centroids = kmeans_from_scratch(X, k=k, max_iter=100)
    # Plot
    plt.figure(figsize=(8, 6))
    sns.scatterplot(x=X[:, 0], y=X[:, 1], hue=labels, palette='viridis', s=50, alpha=0.7)
    plt.scatter(centroids[:, 0], centroids[:, 1], s=200, c='red', marker='X',
label='Centroids')
    plt.title("K-means (From Scratch)")
    plt.xlabel("Standardized Trip Distance")
    plt.ylabel("Standardized Fare Amount")
    plt.legend()
    plt.show()
```

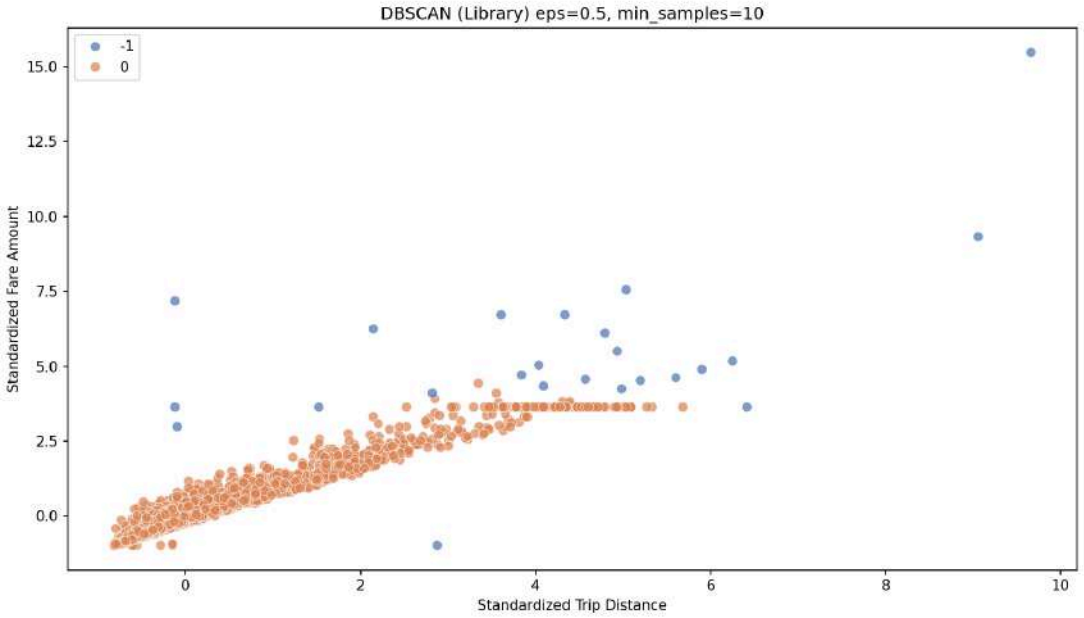


Inference

- **Positive Relationship**: The data shows a roughly linear trend: as distance increases, fare typically increases.
- **Five Clusters**: K-means partitioned the data into 5 groups, each cluster capturing a different range of distance/fare.
- **Centroids**: Red X's mark the center in standardized space for each cluster.
- **Cluster Shapes**: K-means tends to form roughly spherical clusters. The points with very high or low fare/distance appear at the extremes.

Step 3.1 : DBSCAN Clustering

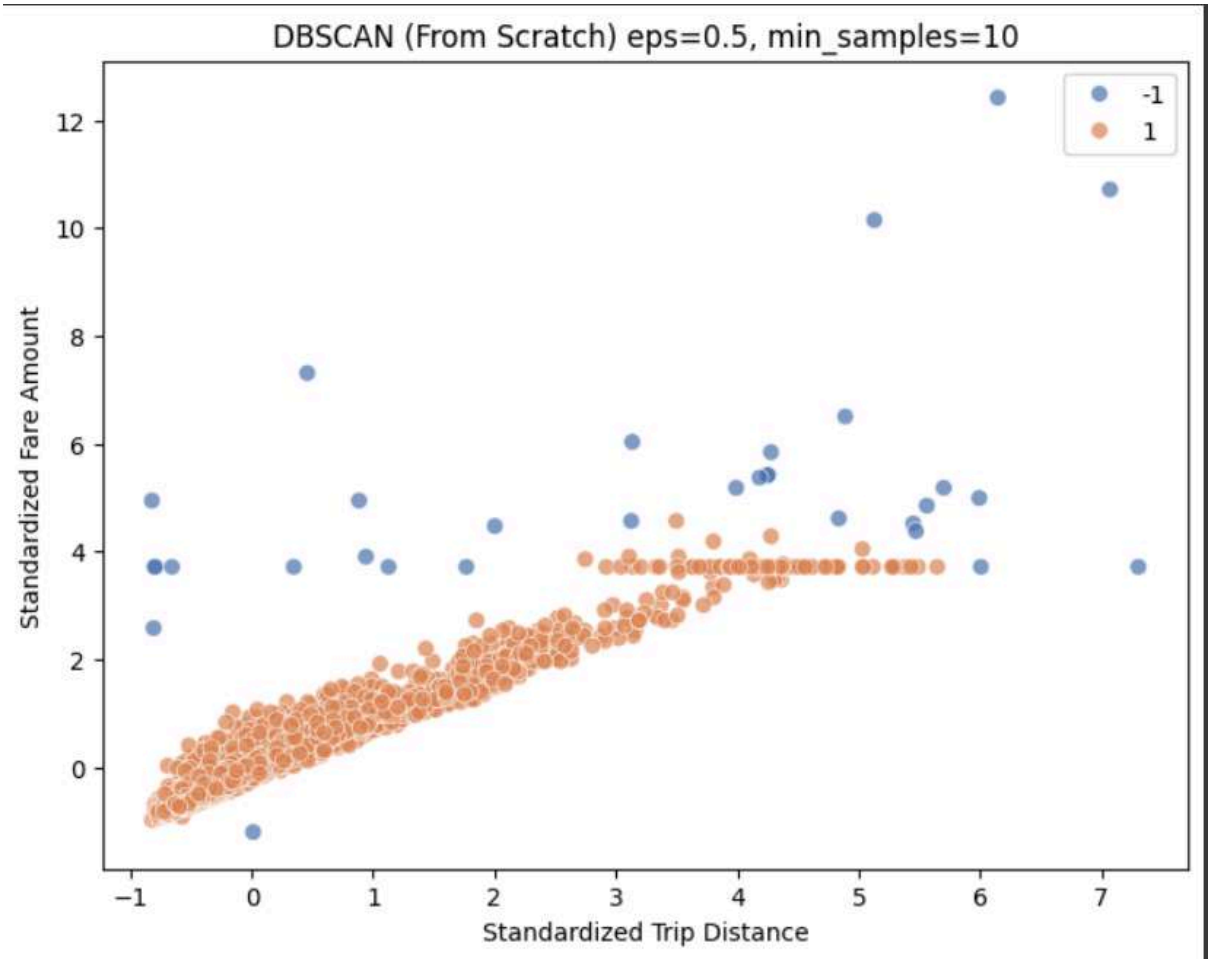
```
dbscan = DBSCAN(eps=0.5, min_samples=10)
labels_db = dbscan.fit_predict(X)
# Plot DBSCAN clusters (noise points are labeled as -1)
plt.figure(figsize=(8, 6))
sns.scatterplot(x=X[:, 0], y=X[:, 1], hue=labels_db, palette='deep', s=50,
alpha=0.7)
plt.title("DBSCAN Clustering")
plt.xlabel("Standardized Trip Distance")
plt.ylabel("Standardized Fare Amount")
plt.show()
```



Step 3.2 : DBSCAN Clustering (Formula)

```
def dbscan_from_scratch(X, eps=0.5, min_samples=10):
    """
    Basic DBSCAN from scratch:
    - RegionQuery, ExpandCluster, etc.
    """
    n_samples = X.shape[0]
    labels = np.full(n_samples, -1, dtype=int)  # -1 = noise by default
    visited = np.zeros(n_samples, dtype=bool)
    cluster_id = 0

    def euclidean_distance(a, b):
        return np.sqrt(np.sum((a - b)**2))

    def region_query(point_idx):
        neighbors = []
        for i in range(n_samples):
            if euclidean_distance(X[point_idx], X[i]) <= eps:
                neighbors.append(i)
        return neighbors

    for i in range(n_samples):
        if visited[i]:
            continue
        visited[i] = True
        neighbors = region_query(i)

        if len(neighbors) < min_samples:
```

```
            labels[i] = -1  # noise
        else:
            # create new cluster
            cluster_id += 1
            labels[i] = cluster_id
            seeds = neighbors.copy()
            seeds.remove(i)  # remove itself if present
            # Expand cluster
            while seeds:
                current_point = seeds.pop()
                if not visited[current_point]:
                    visited[current_point] = True
                    neighbors2 = region_query(current_point)
                    if len(neighbors2) >= min_samples:
                        # add new neighbors
                        for nb in neighbors2:
                            if nb not in seeds:
                                seeds.append(nb)
                if labels[current_point] == -1:
                    labels[current_point] = cluster_id
    return labels

def dbscan_scratch_demo(X, eps=0.5, min_samples=10):
    print("=== DBSCAN (From Scratch) ===")
    labels = dbscan_from_scratch(X, eps=eps, min_samples=min_samples)
    # Plot
    plt.figure(figsize=(8, 6))
    sns.scatterplot(x=X[:, 0], y=X[:, 1], hue=labels, palette='deep', s=50,
alpha=0.7)
    plt.title(f"DBSCAN (From Scratch) eps={eps},
min_samples={min_samples}")
    plt.xlabel("Standardized Trip Distance")
    plt.ylabel("Standardized Fare Amount")
    plt.show()
```



Inference

- **Single Major Cluster**: DBSCAN lumps the majority of points into one cluster (label 0).
- **Noise/Outliers**: Points labeled “-1” deviate from the main density; these may be unusually short or long rides relative to their fares.
- **Parameter Sensitivity**: With eps=0.5 and min_samples=10, you get just one cluster + noise. Different parameters might reveal more subclusters.
- **Linear Trend**: The main cluster still follows the same linear pattern (distance vs. fare).

Conclusion :

In this experiment, clustering algorithms were applied to NYC Yellow Taxi data using standardized trip distance and fare amount. K-means effectively grouped the rides into five clusters, highlighting a clear linear relationship between distance and fare, with each cluster representing different ride patterns. The centroids indicated the average values within each group. In comparison, DBSCAN identified one dense cluster and several outliers, showcasing its ability to detect anomalies. Overall, the experiment demonstrated how unsupervised clustering can reveal patterns and outliers in real-world transportation data for deeper insights.

Name: Alok Yadav Div: D15C Roll No.: 59

EXPERIMENT NO.: 8

AIM: To implement a recommendation system on your dataset using the following machine learning techniques: Regression,Classification, Clustering, Decision tree, Anomaly detection, Dimensionality Reduction, Ensemble Methods.

Theory:

Types of Recommendation Systems

A Recommendation System suggests relevant items to users based on their preferences, behavior, or other factors. There are several types of recommendation techniques:

1. Content-Based Filtering
   Idea: Recommends items similar to those the user has liked before.
   ● Works on: Item features (attributes such as brand, price, category).
   Example:
   ● If a user buys a Samsung phone, they might be recommended another Samsung device based on brand preference.
   ● Uses techniques like TF-IDF (for text data), Cosine Similarity, Decision Trees, etc.

2. Collaborative Filtering (CF)
   Idea: Recommends items based on similar users' preferences.
   ● Works on: User interactions rather than item features.
   Example:● If User A and User B have similar purchase histories, items bought by User
   A but not yet by User B will be recommended to User B.
   ● Uses methods like User-Based CF and Item-Based CF.

3. Hybrid Recommendation System
   Idea: Combines Content-Based Filtering and Collaborative Filtering for better accuracy.
   Example:
   ● Netflix uses a hybrid approach, considering both user preferences and what similar users watch.

4. Knowledge-Based Recommendation
   Idea: Recommends items based on explicit domain knowledge rather than past user behavior.

Example:
● A car recommendation system suggests vehicles based on engine type, price, and fuel efficiency, regardless of past purchases.

Recommendation System Evaluation Measures

Accuracy Measures:

These metrics evaluate how well the recommended items match the actual preferences or ratings of users.

- **Mean Absolute Error (MAE)**:
    - Measures the average of the absolute differences between predicted ratings and actual ratings.

    - **Formula:** $MAE = \frac{1}{n}\sum_{i=1}^{n}|r_i - \hat{r}_i|$
        - r_i = Actual rating
        - $\hat{r}_i$ = Predicted rating

        - **Lower is better.**

- **Root Mean Squared Error (RMSE)**:
    - Similar to MAE but gives higher weight to large errors due to squaring the differences.

    Formula: $RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(r_i - \hat{r}_i)^2}$

        - **Lower is better.**

- **Precision**:

- Measures the fraction of recommended items that are actually relevant to the user.

$$\textbf{Formula: } Precision = \frac{\text{Number of relevant recommended items}}{\text{Total number of recommended items}}$$

  - **Higher is better.**

- **Recall**:
  - Measures the fraction of relevant items that were actually recommended to the user.

$$\textbf{Formula: } Recall = \frac{\text{Number of relevant recommended items}}{\text{Total number of relevant items}}$$

  - **Higher is better.**

- **F1-Score**:
  - The harmonic mean of Precision and Recall, balancing both.

$$\textbf{Formula: } F1 = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall}$$

  - **Higher is better.**

- **Hit Rate**:
  - Measures the fraction of users for whom at least one relevant item is recommended.

$$\textbf{Formula: } HitRate = \frac{\text{Users with at least one relevant recommendation}}{\text{Total users}}$$

  - **Higher is better.**

- **Coverage**:

- Measures the proportion of items from the total available set that are recommended to users.

  Formula: $Coverage = \frac{\text{Number of unique recommended items}}{\text{Total number of items available}}$

  - **Higher is better.**

Diversity and Novelty Measures:

These metrics focus on the **variety** of recommended items and how **unexpected** they are.

- **Diversity**:
  - Measures how different the recommended items are from each other. A diverse set prevents the system from recommending very similar items.
  - **Formula**:
    - Calculate the pairwise similarity between recommended items (e.g., cosine similarity) and compute the average diversity across all users.
    - **Higher diversity is better.**
- **Novelty**:
  - Measures how **unexpected** or **unknown** the recommended items are to the user.
  - For example, recommending items that the user hasn't interacted with before (e.g., exploring genres they haven’t tried).
  - **Higher novelty is better.**
- **Serendipity**:
  - Similar to novelty but with a focus on the surprise element that still fits the user’s interests.

- The idea is to recommend items that are surprising but still relevant.

Implementation

The Diet recommendation is built using Nearest Neighbors algorithm which is an unsupervised learner for implementing neighbor searches.For our case, we used the brute-force algorithm using cosine similarity due to its fast computation for small datasets.

1. **Importing dataset**

```
import kagglehub
from kagglehub import KaggleDatasetAdapter

# Set the path to the file you'd like to load
file_path = "recipes.csv"

# Load the latest version
df = kagglehub.load_dataset(
  KaggleDatasetAdapter.PANDAS,
  "irkaal/foodcom-recipes-and-reviews",
  file_path,
)

print("First 5 records:", df.head())
```

```
data = df
data.head()
```

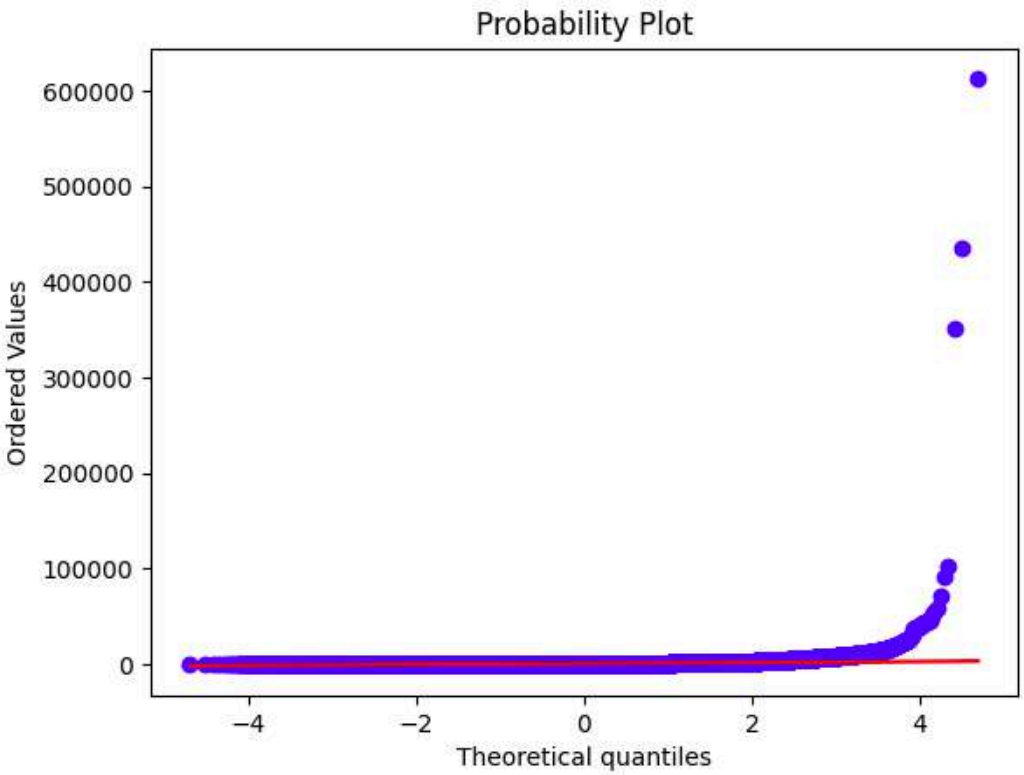
	RecipeId	Name	AuthorId	AuthorName	CookTime	PrepTime	TotalTime	DatePublished	Description
0	38	Low-Fat Berry Blue Frozen Dessert	1533	Dancer	PT24H	PT45M	PT24H45M	1999-08-09T21:46:00Z	Make and share this Low-Fat Berry Blue Frozen ...
1	39	Biryani	1567	elly9812	PT25M	PT4H	PT4H25M	1999-08-29T13:12:00Z	Make and share this Biryani recipe from Food.com.
2	40	Best Lemonade	1566	Stephen Little	PT5M	PT30M	PT35M	1999-09-05T19:52:00Z	This is from one of my first Good House Keepi...
3	41	Carina's Tofu-Vegetable Kebabs	1586	Cyclopz	PT20M	PT24H	PT24H20M	1999-09-03T14:54:00Z	This dish is best prepared a day in advance to...
4	42	Cabbage Soup	1538	Duckie067	PT30M	PT20M	PT50M	1999-09-19T06:19:00Z	Make and share this Cabbage Soup recipe from F...

5 rows × 28 columns

The recipes dataset contains 522,517 recipes from 312 different categories. This dataset provides information about each recipe like cooking times, servings, ingredients, nutrition, instructions, and more.

2. Detecting Outlier

```
import pylab
import scipy.stats as stats
stats.probplot(data.Calories.to_numpy(), dist="norm", plot=pylab)
pylab.show()
```



The data exhibits a significant right-skewness, indicating that the majority of values are relatively small, while a few extreme values with very high calorie counts are pulling the distribution toward the right. Additionally, the presence of outliers, particularly the very large values, seems to be influencing the overall distribution.

3. **Setting the maximum nutritional values for each category for healthier recommendations**

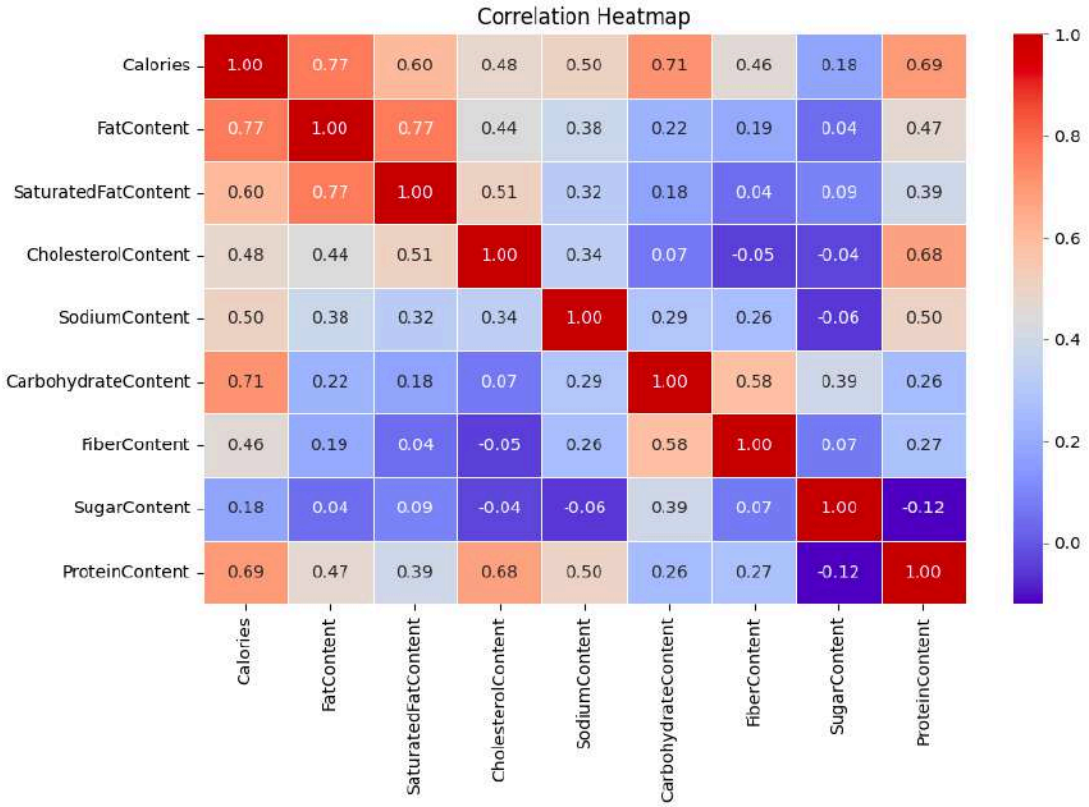
```
max_Calories=2000
max_daily_fat=100
max_daily_Saturatedfat=13
max_daily_Cholesterol=300
max_daily_Sodium=2300
max_daily_Carbohydrate=325
max_daily_Fiber=40
max_daily_Sugar=40
max_daily_Protein=200
max_list=[max_Calories,max_daily_f
```

4. Correlation among nutritional values

```
extracted_data.iloc[:,6:15].corr()
```

	Calories	FatContent	SaturatedFatContent	CholesterolContent	SodiumContent	CarbohydrateContent	FiberContent	SugarContent	ProteinContent
Calories	1.000000	0.767356	0.603317	0.478934	0.501082	0.711640	0.458711	0.180895	0.689447
FatContent	0.767356	1.000000	0.767357	0.440515	0.381944	0.223549	0.192142	0.042603	0.468088
SaturatedFatContent	0.603317	0.767357	1.000000	0.512186	0.319671	0.176623	0.044003	0.090721	0.388618
CholesterolContent	0.478934	0.440515	0.512186	1.000000	0.335843	0.066104	-0.047346	-0.036112	0.675302
SodiumContent	0.501082	0.381944	0.319671	0.335843	1.000000	0.294636	0.260479	-0.055518	0.500457
CarbohydrateContent	0.711640	0.223549	0.176623	0.066104	0.294636	1.000000	0.580535	0.390120	0.255447
FiberContent	0.458711	0.192142	0.044003	-0.047346	0.260479	0.580535	1.000000	0.068758	0.273488
SugarContent	0.180895	0.042603	0.090721	-0.036112	-0.055518	0.390120	0.068758	1.000000	-0.120441
ProteinContent	0.689447	0.468088	0.388618	0.675302	0.500457	0.255447	0.273488	-0.120441	1.000000

- **Fat** and **calories** are strongly correlated, suggesting that food with high fat content tends to have higher calorie content.
- **Fiber and carbohydrates** are related, with fiber often being part of the carbohydrate content in food.
- **Cholesterol and protein** content are moderately related, possibly due to the fact that animal-based foods, which are rich in cholesterol, are also rich in protein.
- **Sodium** is moderately correlated with both calories and protein content, suggesting that foods higher in calories or protein often have more sodium (which is typical of processed foods).



5. Normalising data using z-score normalisation

```
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
prep_data=scaler.fit_transform(extracted_data.iloc[:,6:15].to_numpy())
```

```
array([[-0.55093359, -0.91281917, -0.77924852, ...,  0.15672078,
         2.35502102, -0.68338127],
       [ 1.47428542,  1.13139595, -0.0647135 , ...,  3.91055068,
         2.56324444,  1.25158691],
       [-0.92414618, -1.11248669, -1.12222533, ...,  0.4855234 ,
         0.98513013, -0.60183088],
       ...,
       [ 0.49162165,  0.73206091,  1.85024037, ..., -0.61048534,
         1.76322815, -0.56476253],
       [ 0.25704672,  0.03797856,  1.02137974, ..., -0.61048534,
         1.54404561, -0.63148557],
       [-1.40937801, -1.09347074, -1.12222533, ..., -0.82968708,
        -0.94367625, -0.74269064]])
```

6. **Training the model using K Nearest Neighbours (KNN)**

```
from sklearn.neighbors import NearestNeighbors
neigh = NearestNeighbors(metric='cosine',algorithm='brute')
neigh.fit(prep_data)
```

Here, the metric is set to **'cosine'**, meaning the model will measure the cosine similarity between data points. The **'brute'** algorithm is being used, which computes all pairwise distances between data points without optimization for speed.

7. **Applying KNN**

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import FunctionTransformer
transformer =
FunctionTransformer(neigh.kneighbors,kw_args={'return_distance':False})
pipeline=Pipeline([('std_scaler',scaler),('NN',transformer)])
params={'n_neighbors':10,'return_distance':False}
pipeline.get_params()
pipeline.set_params(NN__kw_args=params)
```

8. **Testing the model**

```
extracted_data[extracted_data['RecipeIngredientParts'].str.contains("egg",regex=Fals
e)]
```

	RecipeId	Name	CookTime	PrepTime	TotalTime	RecipeIngredientParts	Calories	FatContent	SaturatedFatContent
3	41	Carina's Tofu-Vegetable Kebabs	PT20M	PT24H	PT24H20M	c("extra firm tofu", "eggplant", "zucchini", "...	536.1	24.0	3.8
7	45	Buttermilk Pie With Gingersnap Crumb Crust	PT50M	PT30M	PT1H20M	c("sugar", "margarine", "egg", "flour", "salt"...	228.0	7.1	1.7
12	50	Biscotti Di Prato	PT50M	PT20M	PT1H10M	c("flour", "sugar", "baking powder", "salt", "...	89.4	2.6	0.3
18	56	Buttermilk Pie	PT1H	PT20M	PT1H20M	c("butter", "margarine", "sugar", "flour", "eg...	395.9	19.1	9.8
22	60	Blueberry Dessert	NaN	PT35M	PT35M	c("Bisquick baking mix", "sugar", "butter", "m...	381.1	17.3	8.8
...	...	...	...	...	...	...	...	...	...
522484	541351	Spinach & Mushroom Quiche with Boursin	PT1H	PT20M	PT1H20M	c("butter", "onion", "sweet pepper", "carrots"...	197.6	11.0	4.0
522490	541357	Chocolate Rum Snowballs	PT8M	PT15M	PT23M	c("rolled oats", "sweetened flaked coconut", "...	127.8	6.2	4.1
522500	541367	Thick Peanut Pancakes	PT10M	PT45M	PT55M	c("plain flour", "baking powder", "baking soda...	712.9	25.4	8.6
522510	541377	Slow-Cooker Classic Coffee Cake	PT3H	PT20M	PT3H20M	c("all-purpose flour", "brown sugar", "butter"...	358.9	19.8	10.5

In the above example, we instructed the model to recommend recipes that contain "eggs," and it successfully retrieved those recipes.

9. Creating functions for all

```
def scaling(dataframe):
  scaler=StandardScaler()
  prep_data=scaler.fit_transform(dataframe.iloc[:,6:15].to_numpy())
  return prep_data,scaler

def nn_predictor(prep_data):
  neigh = NearestNeighbors(metric='cosine',algorithm='brute')
  neigh.fit(prep_data)
  return neigh

def build_pipeline(neigh,scaler,params):
  print("Building pipeline with params (type):", type(params))  # Debug: should print
<class 'dict'>
  transformer = FunctionTransformer(neigh.kneighbors,kw_args=params)
  pipeline=Pipeline([('std_scaler',scaler),('NN',transformer)])
```

```
  return pipeline

#Extracts the numeric columns with numeric values less than
def extract_data(dataframe, ingredient_filter, max_nutritional_values,
allergic_filter=None):
  extracted_data = dataframe.copy()

  # Filter based on nutritional limits
  for column, maximum in zip(extracted_data.columns[6:15], max_nutritional_values):
    extracted_data = extracted_data[extracted_data[column] < maximum]

  # Include recipes that contain specified ingredients (if provided)
  if ingredient_filter is not None:
    for ingredient in ingredient_filter:
      extracted_data = extracted_data[
        extracted_data['RecipeIngredientParts'].str.contains(ingredient, regex=False)
      ]

  # Exclude recipes that contain any allergenic ingredients
  if allergic_filter is not None:
    for allergen in allergic_filter:
      extracted_data = extracted_data[
        ~extracted_data['RecipeIngredientParts'].str.contains(allergen, regex=False)
      ]

  return extracted_data

def apply_pipeline(pipeline,_input,extracted_data):
  return extracted_data.iloc[pipeline.transform(_input)[0]]

def recommend(dataframe, _input, max_nutritional_values, ingredient_filter=None,
allergic_filter=None, params={'return_distance': False, 'n_neighbors': 10}):
  extracted_data = extract_data(dataframe, ingredient_filter, max_nutritional_values,
allergic_filter)
  prep_data, scaler = scaling(extracted_data)
  neigh = nn_predictor(prep_data)
  pipeline = build_pipeline(neigh, scaler, params)
  return apply_pipeline(pipeline, _input, extracted_data)
```

10. Testing Recommendation with custom nutritional values

Column Names: ['Calories', 'FatContent', 'SaturatedFatContent', 'CholesterolContent', 'SodiumContent', 'CarbohydrateContent', 'FiberContent', 'SugarContent', 'ProteinContent']
Test Input Values: [[1800, 30, 12, 250, 1500, 300, 30, 20, 100]]

We will input the following test values: [[1800, 30, 12, 250, 1500, 300, 30, 20, 100]] into the model, and it will recommend recipes that have approximate values matching these inputs.

[1800, 30, 12, 250, 1500, 300, 30, 20, 100]

```
ingredient_filter = ['egg', 'milk']
allergic_filter = ['flour']
params = {'return_distance': False, 'n_neighbors': 5}
test_input = manual_input_df.to_numpy()
# print(recommendations)
recommend(dataset, test_input, max_list, ingredient_filter, allergic_filter,
params=params)
```

```
Building pipeline with params (type): <class 'dict'>
```

	RecipeId	Name	CookTime	PrepTime	TotalTime	RecipeIngredientParts	Calories	FatContent	SaturatedFatContent	CholesterolContent
192471	201006	Spaghetti With Turkey Meatballs	PT10M	PT50M	PT1H	c("olive oil", "garlic cloves", "crushed tomat...	918.1	28.2	5.9	145.4
5142	8067	Meatball Stew with Dumplings	NaN	PT0S	PT20M	c("sour cream", "green beans", "carrots", "pot...	1331.2	32.0	10.1	164.5
129249	135785	Perfect Pastitsio (Vegetarian)	PT40M	PT35M	PT1H15M	c("olive oil", "onion", "vegetarian ground bee...	575.4	15.8	5.2	100.7
188211	196628	Western Tamale Pie	PT25M	PT15M	PT40M	c("onion", "green bell pepper", "chili powder"...	565.2	21.3	6.6	88.2
226699	236213	Slow-Cooked Meatloaf and Veggies	PT8H	PT15M	PT8H15M	c("egg", "milk", "onion", "green peppers", "sa...	668.2	20.0	7.9	159.2

In this case, we also specified that the recommended recipes should include certain ingredients like eggs and milk, while excluding allergens such as flour. The model then provided recipes that contained the specified ingredients but excluded those with allergens.

Conclusion:

In conclusion, the experiment successfully demonstrated the implementation of a content based recommendation system that leverages data normalization, K-Nearest Neighbors with cosine similarity, and a tailored pipeline to provide personalized recipe recommendations. The system effectively incorporates nutritional constraints, ingredient preferences, and allergen filters, ensuring that the output meets health-focused criteria while aligning with user-specific ingredient requirements. This modular approach not only highlights the strengths of supervised learning in recommendation contexts but also lays the foundation for future improvements through user feedback and more advanced evaluation metrics.

Experiment - 9

Aim: To perform Exploratory data analysis using Apache Spark and Pandas

Theory:

1) **What is Apache Spark and it works?**

Ans:
Apache Spark is an open-source, distributed computing system designed for big data processing and analytics. It's known for its speed, scalability, and ease of use, especially for large-scale data processing tasks like machine learning, stream processing, and SQL queries.

Language Support: Scala (native), Java, Python (PySpark), R, and SQL.

At a high level, Spark works in four main stages:

1. Driver Program Starts
The user writes an application using Spark APIs (in Python, Scala, etc.).
The Driver Program is the heart of Spark, where the main control flow runs.
It converts user code into DAGs (Directed Acyclic Graphs) of stages.

2. Cluster Manager Allocates Resources
Spark uses a Cluster Manager (e.g., YARN, Mesos, or its own standalone manager) to allocate resources.
Executors (worker nodes) are launched across the cluster.

3. Tasks Are Sent to Executors
The DAG Scheduler breaks the job into tasks.
These tasks are sent to the Executors, which actually do the work like reading data, running transformations, and writing output.

4. Executors Run Tasks and Return Results
Executors process data in memory for fast performance.
Intermediate results are cached when possible.
Final results are returned to the Driver or written to storage.

Use Cases
Processing logs or real-time event data.
Running ML models on large datasets.
ETL (Extract, Transform, Load) pipelines.
Analyzing huge volumes of structured or unstructured data.

2) How data exploration done in Apache spark? Explain steps.

Ans:
Data exploration in **Apache Spark**—especially using **PySpark** (Python API for Spark)—is a crucial step in any big data analytics or machine learning pipeline. It helps you understand the structure, quality, and patterns in your dataset.

Steps for Data Exploration in Apache Spark:

1. Loading the Data

The first step in data exploration is importing the data into Spark from sources such as CSV files, JSON, Parquet, databases, or cloud storage systems. Apache Spark provides various APIs to connect to and load data from different formats and sources efficiently.

2. Understanding the Schema

Once the data is loaded, the schema (structure) of the dataset is examined. This includes:

- Column names
- Data types (e.g., Integer, String, Date)
- Nullable fields

Understanding the schema helps identify incorrect or inconsistent data types and ensures the data is correctly interpreted.

3. Viewing Sample Records

Exploratory analysis begins with viewing a few records from the dataset. This helps in gaining a quick overview of:

- The kind of data stored
- Formatting or entry errors
- Presence of special characters or missing values

4. Generating Summary Statistics

Statistical summaries of numerical columns are generated to understand the distribution and spread of the data. These statistics include:

- Count

- Mean
- Minimum and Maximum values
- Standard deviation

This step is essential for detecting outliers and understanding the scale of data.

5. Identifying Missing or Null Values

It is important to detect and assess missing or null values in the dataset. This helps in deciding whether to remove, replace, or impute missing data before further analysis.

6. Analyzing Value Distributions

The distribution of values within categorical or numerical columns is analyzed. This includes grouping data based on categories and counting their occurrences. It helps to:
Identify class imbalances
Understand the frequency of certain values

7. Examining Relationships and Correlations

In this step, the relationships between numerical variables are explored. Correlation analysis is performed to measure how strongly two variables are related. This insight is useful for feature selection in machine learning.

8. Filtering and Conditional Queries

Subsets of data are explored by applying filters and conditions. This helps focus on specific segments or conditions, such as high-income individuals or records with specific attributes.

9. Sampling for Detailed Analysis

If the dataset is very large, a smaller representative sample is taken for detailed analysis or visualization. This reduces computational cost and allows easier inspection.

10. Preparing for Visualization or Modeling

Although Spark itself is not primarily used for visualizations, after exploration, data is often summarized or exported for plotting using external tools. The insights gained during exploration guide the next steps in data cleaning, transformation, or modeling.

Conclusions:

Apache Spark is a fast and general-purpose distributed computing system designed for large-scale data processing. It utilizes a driver-executor architecture and performs in-memory computations to achieve high performance. Spark supports various APIs, including RDDs, DataFrames, and Datasets, enabling efficient data manipulation and processing across multiple programming languages.

Experiment 10

Aim: To perform Batch and Streamed Data Analysis using Apache Spark.

Theory:

1) What is streaming. Explain batch and stream data.

Ans:

Streaming is a data processing method where data is continuously generated, transmitted, and processed in real-time or near real-time. This approach is used when immediate insights or actions are required, such as in live video feeds, financial transactions, or sensor data in IoT systems.

Instead of waiting for a complete dataset, streaming systems handle data as it arrives, enabling timely processing and responses.

Aspect	Batch Data	Stream Data
Definition	Data is collected, stored, and processed in large chunks or batches.	Data is continuously generated and processed in real-time.
Example	Processing monthly sales reports.	Monitoring live stock market prices.
Latency	High (processing happens after data collection).	Low (near real-time processing).
Storage	Data is stored first, then processed.	Processed immediately as it arrives.
Use Cases	Data warehousing, analytics reports, backups.	Fraud detection, live analytics, IoT monitoring.
Tools	Hadoop, Apache Spark (batch mode).	Apache Kafka, Apache Flink, Spark Streaming.

2) How data streaming takes place using Apache spark.

Ans:

Apache Spark provides a powerful framework called Spark Structured Streaming to handle real-time data streams. It allows for continuous processing of data as it arrives, combining the simplicity of SQL/DataFrame operations with the scalability and fault-tolerance of Spark.

Key Components and Process

1. Data Source (Input)

The streaming process begins with a data source. Spark reads data continuously from streaming sources like:

- Apache Kafka
- File systems (monitoring new files in a directory)
- Sockets
- Amazon Kinesis
- Other custom sources

These sources send data in real-time, which Spark ingests as a stream.

2. Streaming Data as a Table

Spark treats streaming data as an unbounded table. Each new data item is like a new row being added to this table. Onecan perform operations like select, filter, groupBy, and even SQL queries on this streaming table.

3. Query Execution

The user defines a query on the streaming data (e.g., count words, calculate averages). Internally, Spark builds a logical plan and then optimizes it into a physical plan for execution.

4. Micro-Batch Processing

Spark Structured Streaming processes data in micro-batches.
Instead of processing each event individually, it collects data for a short interval (e.g., every second) and processes it together.
This approach balances real-time performance with processing efficiency.

5. Output Sink

After processing, the results are written to an output sink, such as:

- Console (for testing/debugging)
- Kafka
- Databases
- File systems

You can choose different output modes:

- Append: Only new rows are written.
- Update: Only updated rows are written.
- Complete: The entire result table is written.
- Fault Tolerance

Conclusion:

Batch and streamed data analysis are two core approaches in data processing. **Batch analysis** processes large volumes of data collected over time, ideal for historical insights and complex computations. **Streamed analysis**, on the other hand, processes data in real-time as it arrives, enabling immediate decision-making. While batch is suited for accuracy and completeness, streaming excels in speed and responsiveness. Together, they offer a powerful hybrid approach for modern data-driven systems.

NutriTrack - AI-Based Health Wellness System

ON

Submitted in partial fulfillment of the requirements of the degree of

Bachelor of Engineering

(Information Technology)

By

Brijesh Sharma (48)

Alok Yadav (59)

Sandesh Yadav (61)

Under the guidance of

Dr. Ravita Mishra

Department of Information Technology

VIVEKANAND EDUCATION SOCIETY'S INSTITUTE OF TECHNOLOGY,

Chembur, Mumbai 400074

(An Autonomous Institute, Affiliated to University of Mumbai) April 2024

AIDS Lab Exp 11
Aim: Mini Project – NutriTrack - AI-Based Health Wellness System

---

1.1 Introduction

In today's fast-paced lifestyle, individuals often struggle to maintain a healthy diet and fitness regime tailored to their specific body requirements and goals. Generic dietary recommendations often fall short in catering to personalized needs, which may vary based on age, gender, activity levels, and health conditions.

2.2 Design and Implementation

The design and implementation of the personalized diet and nutrition recommendation system are centered around creating an intuitive, responsive, and intelligent platform that seamlessly integrates a user-friendly frontend with a robust backend powered by machine learning models. The system is developed using React for the frontend and FastAPI for the backend to ensure efficient, scalable, and real-time communication between the user interface and the machine learning recommendation engine.

2.3 Feature Scaling

RFM values were standardized using StandardScaler to ensure equal contribution to clustering models.

2.4 Evaluation Analysis

KNN (with Cosine Similarity):

KNN recommends recipes by comparing the target nutritional vector (e.g., Calories, Protein) with each recipe's 9 scaled nutritional features using cosine similarity, after applying ingredient/allergy filters. Chosen for its simplicity, no training requirement, compatibility with filtering, and baseline performance (MSE: 51095.76), it's ideal for direct nutritional matching on small datasets.

Weighted Euclidean Distance:

A KNN variant using a weighted distance metric to emphasize key nutrients (e.g., Calories weight=2.0), improving control over deviations (Calories deviation reduced from 76.07% to 56.38%). Achieved better MSE (38406.71) than cosine, supports user-driven nutrient prioritization, and integrates with existing filters.

K-Means Clustering:

Clusters recipes into k groups (e.g., 50) based on nutritional features. For a target input, the closest cluster is selected, and top recipes are recommended. It improves diversity, efficiency, and handles cases with few exact matches, complementing KNN with pattern-based grouping.

Support Vector Regressor (SVR):

Trains 9 regressors (one per nutrient) using TF-IDF ingredients and time features to predict nutritional values. Recommendations are based on distance between predicted profiles and the target. Chosen for modeling ingredient-nutrient relationships, handling high-dimensional data, and reducing nutrient deviations with supervised learning.

Chapter 4: Results and Discussion

4.1 Dataset Description

The recipes dataset contains 522,517 recipes from 312 different categories. This dataset provides information about each recipe like cooking times, servings, ingredients, nutrition, instructions, and more.

RecipeId, Name, CookTime, PrepTime, TotalTime, RecipeIngredientParts, Calories, FatContent, SaturatedFatContent, CholesterolContent, SodiumContent, CarbohydrateContent, FiberContent, SugarContent, ProteinContent, RecipeInstructions

4.1 KNN:

KNN, a simple supervised algorithm, predicts based on the average of K similar data points. With MSE of 30517.99 and MAE of 101.99, it offers balanced nutrient deviations (e.g., Calories 24.01%), making it the most consistent and reliable model in this setup.

```
Evaluation Metrics for Breakfast:
MSE: 30517.99
MAE: 101.99
Deviation Percentages (%):
  Calories: 24.01%
  FatContent: 30.20%
  SaturatedFatContent: 30.50%
  CholesterolContent: 49.94%
  SodiumContent: 21.27%
  CarbohydrateContent: 66.75%
  FiberContent: 53.67%
  SugarContent: 46.50%
  ProteinContent: 54.74%
```

4.2 KMeans Clustering

K-Means clusters recipes by nutritional similarity and showed lower MSE (19047.16) and MAE (79.63). However, higher deviations (e.g., Protein 58.04%) and its unsupervised nature limit its use for precise predictions, making KNN a better choice overall.

```
Evaluation Metrics for Breakfast:
MSE: 19047.16
MAE: 79.63
Deviation Percentages (%):
  Calories: 31.24%
  FatContent: 19.13%
  SaturatedFatContent: 20.33%
  CholesterolContent: 45.38%
  SodiumContent: 8.42%
  CarbohydrateContent: 52.84%
  FiberContent: 24.80%
  SugarContent: 46.70%
  ProteinContent: 58.04%
```

4.3 KNN (Weighted Euclidean Distance)

This KNN variant prioritizes nutrients via weights but had higher MSE (38406.71), MAE (108.02), and poor calorie accuracy (76.07% deviation). Despite benefits like customizable importance, it lacked the precision for effective recommendations.

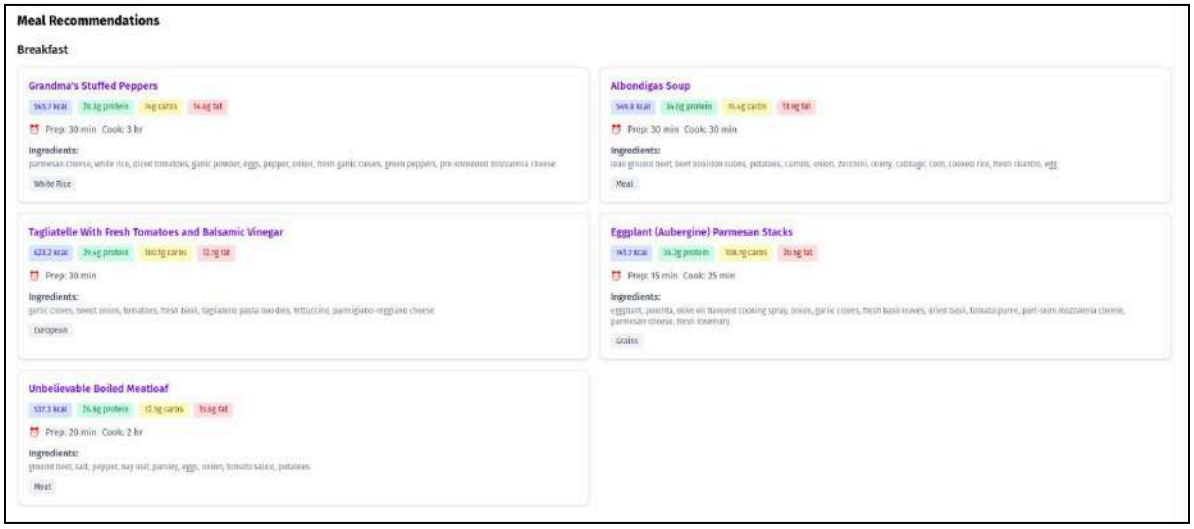
```
Evaluation Metrics for Breakfast:
MSE: 22125.59
MAE: 85.94
Deviation Percentages (%):
  Calories: 38.06%
  FatContent: 16.47%
  SaturatedFatContent: 22.83%
  CholesterolContent: 41.70%
  SodiumContent: 10.53%
  CarbohydrateContent: 47.70%
  FiberContent: 34.13%
  SugarContent: 41.90%
  ProteinContent: 55.12%
```

4.4 Support Vector Regression

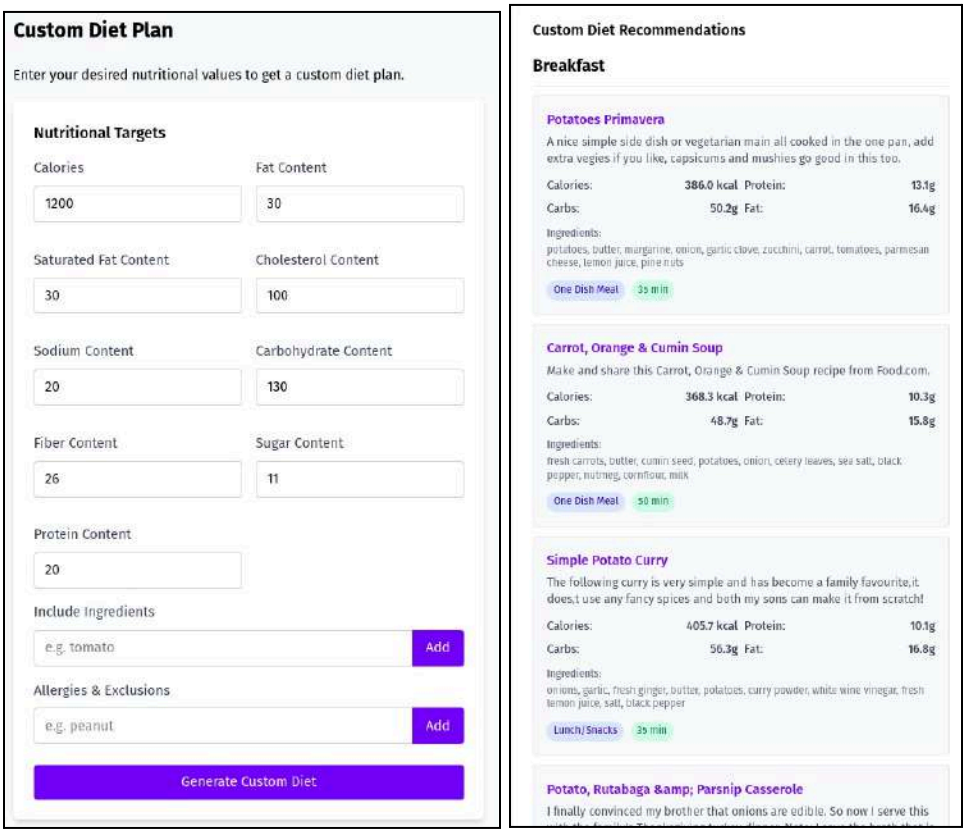
SVR, though powerful, had the worst performance (MSE: 174111.09, MAE: 218.83) with very high nutrient deviations (e.g., Protein 88.88%). Its sensitivity to noisy data led to poor generalization, making it unsuitable for this task.

```
Evaluation Metrics for Breakfast:
MSE: 174111.09
MAE: 218.83
Deviation Percentages (%):
  Calories: 49.27%
  FatContent: 34.47%
  SaturatedFatContent: 16.17%
  CholesterolContent: 43.15%
  SodiumContent: 73.07%
  CarbohydrateContent: 85.81%
  FiberContent: 92.27%
  SugarContent: 40.20%
  ProteinContent: 88.88%
```

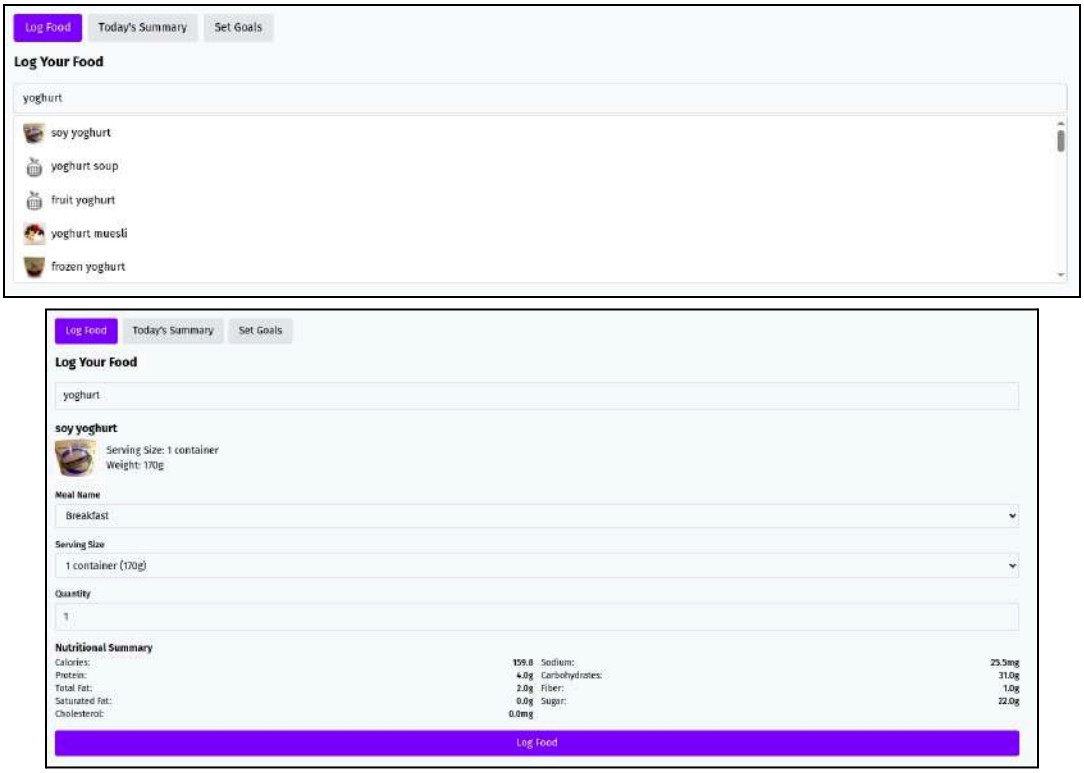
4.6 Streamlit App Output



Meal recommendations based on user data



Custom diet recommendations based on user requirements



Custom Diet goals based on user intake

Conclusion

This project explored multiple models for recipe recommendation based on nutritional goals, comparing both supervised and unsupervised approaches. Among all, KNN emerged as the most reliable due to its consistent accuracy, balanced deviations, and simplicity. While K-Means showed promising MAE, its lack of predictive depth limits its utility. Overall, KNN offers the best balance of precision, interpretability, and adaptability for personalized nutritional recommendations.

Name:- Alok Yadav
Roll no: 59
Div: D15C

AIDS

Page No.
Date

05/05

Assignment-1

1. What is AI? Considering the COVID-19 pandemic situation, how AI helped to survive and renovated our way of life with different applications.

=> Artificial Intelligence (AI) is a branch of computer science that enables machines to perform tasks that typically require human intelligence. These tasks include learning, reasoning, problem-solving, perception, language understanding and decision making.

(1) AI-powered algorithms analyzed news, social media, and official reports to detect early signs of outbreaks.

(2) Drug Discovery & Vaccine development: AI accelerated the development of COVID-19 treatments and vaccines by analyzing protein structures and predicting potential drug candidates.

(3) Medical diagnosis: AI powered diagnostic tools analyzed x-rays and CT scans to detect COVID-19 infections faster than traditional methods.

(4) Contact Tracing and spread prediction: AI models helped track virus spread using data from mobile apps, GPS, and social media.

(5) Remote work & online learning: AI-powered collaboration tools (Zoom, Teams) enabled remote work and virtual meetings.

(6) Mental Health & Wealth being: AI chatbots like Wysa and Woebot provided mental health support to people dealing with stress and isolation.

2. What are AI agents terminology, explain with examp

a) An AI Agent is an entity that perceives its environment through sensors and acts upon it using actuators to achieve a specific goal. AI agents operate based on decision-making models, algorithms and data.

① Agent:
An agent is any system that perceives its environment and takes actions to achieve a goal.
example: A self-driving car perceives traffic signals and road conditions and takes actions like stopping and accelerating.

② Environment
The environment is everything that surrounds an AI agent and affect its performance.
example: In a chess-playing AI, the chessboard, opponent, moves, the rules form the environment.

③ perception
perception is the agent's ability to gather information from the environment through sensors.
example: A facial recognition system perceives images through a camera.

④ Sensors
Sensors are the input devices that collect data from the environment.
example:- A voice assistant (like Alexa) uses a microphone

Page No.
Date

(5) Actuators

Actuators are components that allow the agent to take action based on decisions

example: In a robot, it's arms & wheels are actuators

(6) Rational agent

A rational agent takes the best possible action to maximize its performance measure.

example: AI stock trading system makes decisions to maximize profit.

(7) Utility ~~agent~~ function

A utility function measures how good an action or state is for achieving a goal.

example: In a game AI, winning a match has a high utility score.

3. How AI technique is used to solve 8 puzzle problem?

=) The 8-puzzle problem is a classic problem in artificial intelligence that involves sliding tiles on a 3x3 grid to reach a goal state.

Representation of the 8-puzzle problem

- Initial state: Any random arrangement of tiles
- Goal state: A predefined arrangement (eg. numbers in order)
- Operators: Move the blank tile (up, down, left, right)
- State space: All possible tile arrangements
- Cost function: Typically, each move has a cost of 1.

Several AI search techniques can be used to solve the 8-puzzle problem:

1) Uninformed search

a) Breadth-first search (BFS)

i) explores all possible moves level by level

ii) Guarantees finding the shortest solution if the solution exists

disadvantages: Requires high memory for large state spaces

b) Depth-first search (DFS)

explores one branch of the state space deeply before backtracking.

disadvantage: May get stuck in deep, infinite paths.

c) Iterative deepening Depth-first-search

Combines BFS and BFS to avoid deep recursion

2) Informed search Methods (Heuristic search)

a) ~~Greedy~~ Best-first search

i) Uses heuristic function to expand the most promising state first

ii) Heuristic example: Number of misplaced tiles.

b) A* search

i) combines path cost and heuristics $f(n) = g(n) + h(n)$

where $g(n)$: cost from the start state to current state

$h(n)$: Estimated cost from current state or goal.

Page No.
Date

common heuristics for A*

(1) Misplaced Tile Heuristic (h_1)

↳ Counts the number of misplaced tiles.

(2) Manhattan distance (h_2)

↳ Measures the sum of distances each ~~tile~~ tile is from its goal position.

↳ More accurate than misplaced tiles heuristic

4. What is PEAS descriptor? Give PEAS descriptor for following:

i) ~~Taxi~~

=) The PEAS (performance measure, Environment, Actuators, Sensors) descriptor is used to define the components of an Agent, helping to analyze its working environment and functions.

Performance Measure (P): The criteria for evaluating the agent's success

Environment (E): The external surroundings where the agent operates.

Actuators (A): The mechanisms through which the agent interacts with the environment

Sensors (S): The device used to perceive the environment

i) Taxi driver.

P: safety, fuel efficiency, passenger satisfaction

E: Roads, traffic signals, other vehicles, pedestrians, weather

A: Steering wheel, accelerator, brakes, horn, indicators, wipers

S: GPS, cameras, LiDAR, speed sensors, fuel gauge, odometer

ii) Medical diagnosis system

P: Diagnosis, accuracy, speed of response, patient recovery rate

E: patients, symptoms, medical databases, hospital environments

A: Display screen, speaker (for communication with doctor and patients)

S: patients history, test reports, X-rays, MRI scans, symptom inputs

iii) A Music composer.

P: Quality of generated music, creativity, audience engagement

E: Musical database, sound libraries, listener preferences

A: Music output system (MIDI, speakers, digital files)

S: User feedback, musical trends, mood analysis, input instruments

iv) An aircraft autolander

P: smooth and safe landing, fuel efficiency, passenger comfort

E: Runway, weather conditions, altitude, air traffic

A: flaps, landing gear, engine throttle, air brakes

S: Gps, altimeter, wind sensors, gyroscopes, radar.

v) An essay evaluator

P: Accuracy in grading, fairness, grammar and coherence detection

E: Essays, answer sheets, writing rules, academic guidelines

A: Score display, feedback generator

S: Optical character recognition (OCR), NLP tools.

vi) Robotic sentry gun for the tech lab

P: Accuracy in target detection, response time,

E: lab premises, intruders, authorized personnel, obstacles

A: Gun turret, alarm system, movement motors

S: Motion detectors, thermal cameras, facial recognition, infrared sensors.

5. Categorize a shopping bot for an offline bookstore according to each of the six dimensions (fully/partially observable, deterministic/stochastic, episodic/sequential, static/dynamic, discrete/continuous, single/multi-agent)

=)

Observability: partially observable.

The bot may not have complete information about books on shelves, customer preferences or stock updates without external input.

Deterministic vs Stochastic :- Stochastic

Book availability may change due to manual sales, external purchases, or misplaced books, making environment uncertain

Episodic vs Sequential : Sequential

Each customer interaction affects the next steps (eg. book recommendations depend on previous queries).

Static vs Dynamic :- Dynamic

The environment changes (books sell out, new books arrive, customer preferences shift) making it dynamic

Page No.
Date

discrete vs continuous : discrete
The bot deals with a finite set of actions (searching books, checking stock).

Single-Agent vs Multi Agent : Multi-Agent
The bot interacts with multiple customers, bookstore staff, and possibly other inventory systems.

6. Differentiate Model based and Utility based Agent.

=>

Model-Based Agent	Utility-Based Agent.
(1) Uses an internal model of the environment to make decisions.	(1) Uses a utility function to measure the desirability of different states and select the best action.
(2) Chooses actions based on a representation of how the world works.	(2) Chooses actions that maximize its expected utility.
(3) Maintains a model of the environment (how actions affect future states)	(3) Uses a utility function to compare possible outcomes.
(4) focuses on achieving a goal using state transition models	(4) focuses on maximizing long-term benefits rather than just reaching a goal
(5) eg. A self driving car that predicts traffic patterns & plan routes	(5) A stock trading bot that evaluates different portfolios to maximize returns

7. Explain the architecture of knowledge based agent and Learning Agent.

=>

Architecture of Knowledge-Based Agent

A knowledge-based agent (KBA) is an AI system that uses stored knowledge to make informed decisions. It consists of the following components

① Knowledge Base (KB)
It stores facts, rules and heuristics about the world

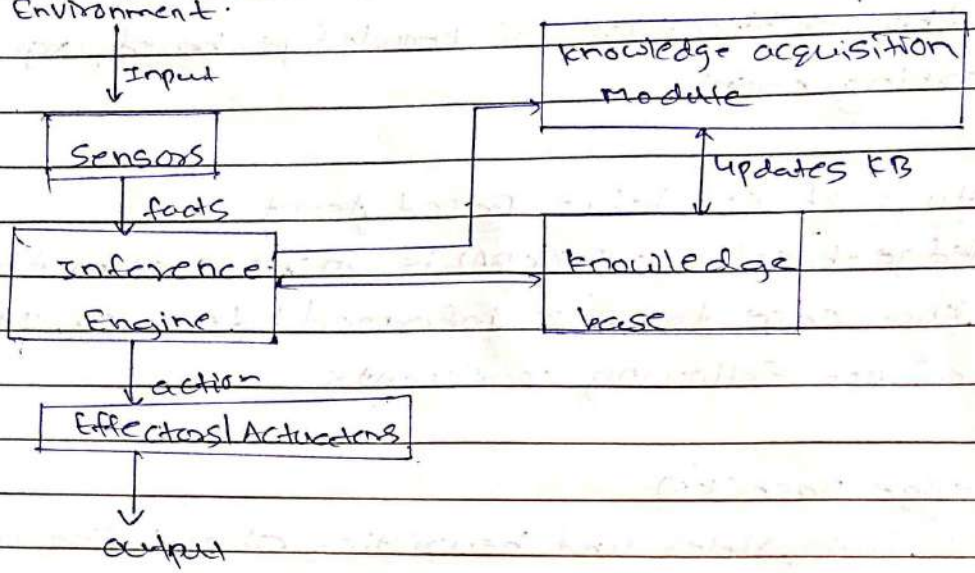
② Inference Engine
It Applies logical reasoning to derive new knowledge from stored facts.

③ perception (sensors)
It collects information from the environment

④ Actuators
performs actions based on inferences

⑤ Knowledge Acquition module
updates and expands the knowledge base with new data

Working process:
① The agent perceives the environment
② It queries the knowledge base for relevant information
③ The inference Engine applies logic to decide an action
④ The action is executed, and KB is updated if needed



Architecture of Learning Agent

A learning Agent improves its performance over time by learning from past experiences.

Components of Learning Agent

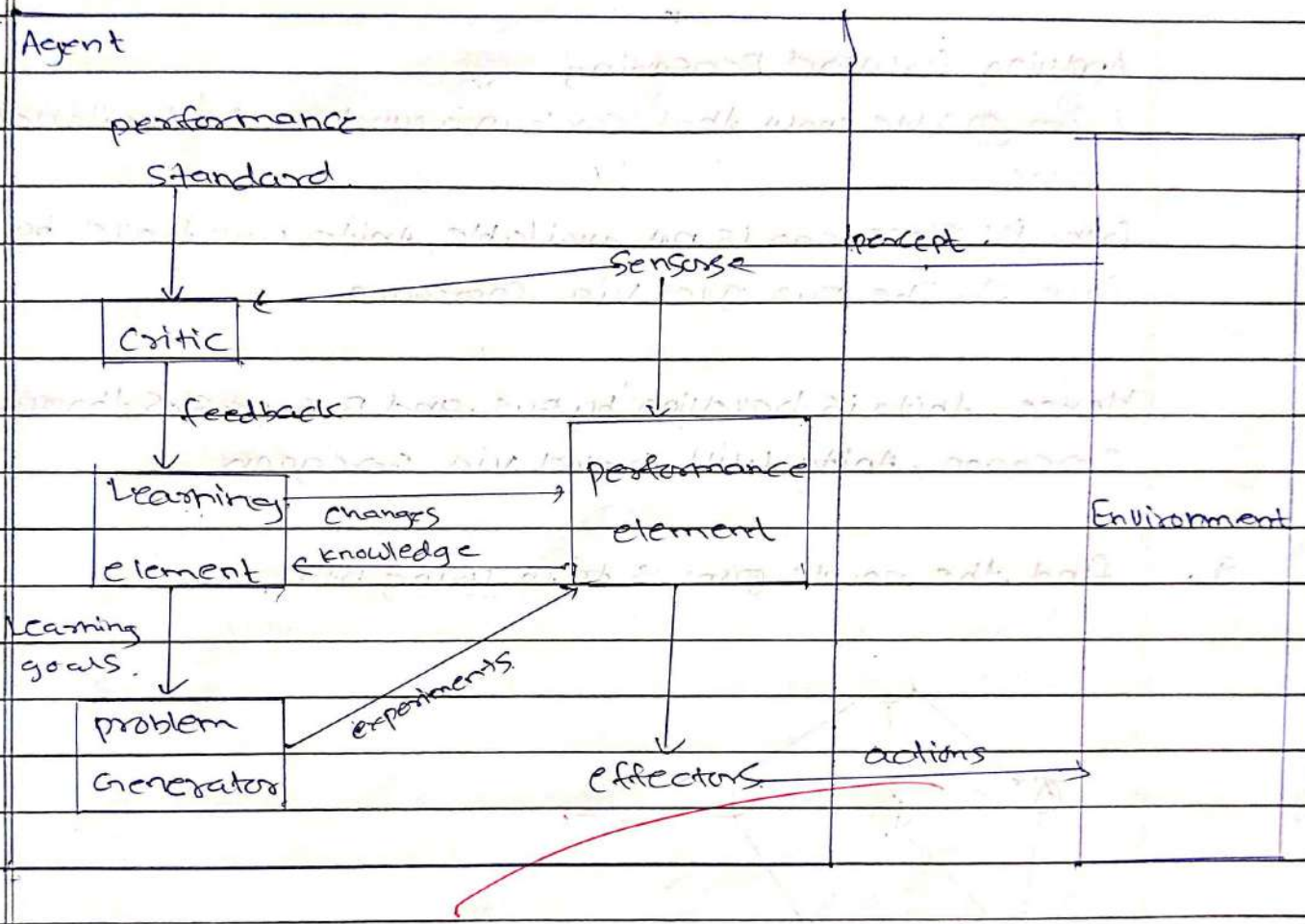
1. Learning Element: Improves the agent's knowledge by analyzing past experiences
2. performance Element: chooses actions based on the learned knowledge
3. critic: Evaluates the agent's actions by comparing outcomes with expected results
4. problem generator: suggests new exploratory actions to improve learning

Working process:

1. The performance element makes decisions and takes actions
2. The critic evaluates the results and provides feedback

(3) The learning Element updates the knowledge based on feedback

(4) The problem Generator suggests new strategies to improve performance.



8. Convert the following to predicates

a) Anita travels by car if available otherwise travels by bus.

Available(car) → Travels(Anita, car) — (1)

¬ Available(car) → Travels(Anita, Bus)

b) Bus goes via Andheri and Goregaon.

GoesVia(Bus, Andheri) ∧ GoesVia(Bus, Goregaon) — (2)

c) car has a puncture, so it is not available.

puncture (car) → ¬Available (car) -③

Will Anita travel via Goregaon

Applying forward Reasoning.

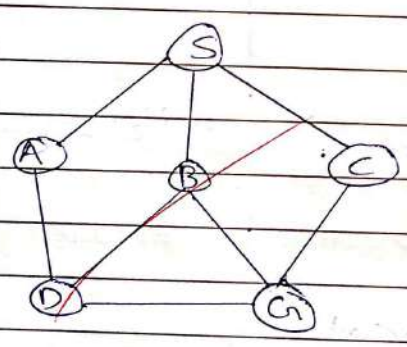
from ③: we know that car has a puncture, so Available(car) is false.

from ①: since car is not available, Anita will travel by bus

from ②: The Bus goes via Goregaon.

Hence, Anita is travelling by Bus and Bus passes through Goregaon, Anita will travel via Goregaon.

9. find the route from S to G using BFS



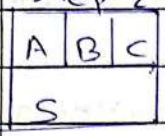
Queue (Q)

processed (P)

Step 1:

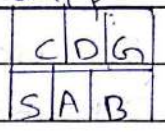
S

Step 2



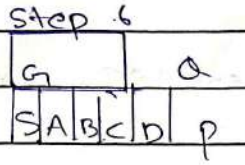
Step 3

Step 4

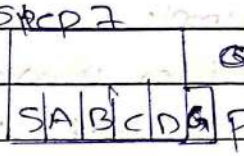


Step 5

Step 6



Step 7



Adjacency list

S → {A, B, C}
A → {D}
B → {B, G}
C → {G}
D → {G}

from BFS and adjacency list
shortest path is S → B → G
other paths are S → C → G and S → B → D → G and S → A → D → G

10. What do you mean by depth limited search? Explain Iterative Deepening search with example

⇒

Depth-Limited search (DLS)

Depth-limited search is a variation of depth-first search (DFS) where we impose a depth limit to avoid going too deep into an infinite or large search space.

How DLS works

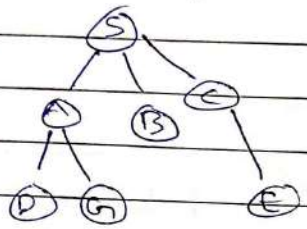
① The algorithm follows a depth-first search (DFS) strategy but limits the depth of recursion
② If the goal is found within the limit, the search stops

Page No.
Date

example

(3) If the goal is not found within the limit, it returns failure or cutoff
(4) This helps in avoiding infinite loops in problems with large or infinit depth.

example:
consider a graph where we want to find a path from S to G with depth limit of 2.



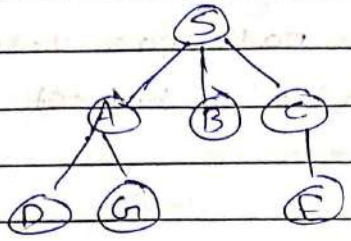
L) If the depth limit is 1, we only explore S→A, B, C but cannot reach G
L) If the depth limit of 2, we explore S→A→B, G.

Iterative deepening search (IDS)

Iterative deepening search (IDS) combines the benefits of depth-first search (DFS) and Breadth-first search (BFS) It repeatedly performs DLS with increasing depth limits until the goal is found.

How IDS works:
(1) start with depth limit = 0 and perform DLS
(2) Increase the depth limit & perform DLS again
(3) Repeat until goal is found.

example:



a) Depth limit 0 → only node S is checked
b) Depth limit 1 → explores A, B, C, but G is not found
c) Depth limit 2 → explores D, G, E and finds G

11. Explain Hill climbing and its drawback in detail with example. Also state limitations of steepest-ascent hill climbing

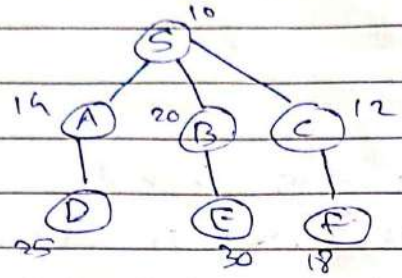
:-)

Hill climbing is an optimization algorithm that continuously moves towards higher-valued states (i.e. better solutions) until a peak (local maximum) is reached. It is a greedy algorithm that evaluates neighbouring states and chooses the one with the highest value. It is widely used in Artificial Intelligence, especially in problems like pathfinding, scheduling problems

Working of Hill Climbing:
1. Start with an initial state (solution).
2. Evaluate the neighbouring states of the current state
3. Move to the best neighbouring state that improves the solution
4. Repeat the process until no better neighbour is found or predefined goal state is reached.

example:

consider a graph where each node has value representing its height and goal is to find the highest valued node



(1) Start at node S (10)

(2) check its neighbors: A, B, C. then move to B (highest value = 20)

(3) check neighbors of B: F(30) then move to F (highest value = 30)

(4) F has no better neighbour, so stop.

(5) final result: Node F (value = 30) is the peak.

Drawbacks of Hill climbing:

(1) Local Maxima: if node D(25) was chosen from A instead of B, it would be stuck there, not reaching 30.

(2) plateaus: If multiple nodes had the same value, the algorithm might get confused

(3) Ridges: The algorithm cannot take downward moves to explore better paths.

(4) No backtracking: Hill climbing does not remember previous states, so if it gets stuck, it cannot backtrack to explore better paths

Page No.
Date

Steepest-Ascent Hill climbing and its limitations

Steepest-Ascent Hill climbing is a variation where the algorithm evaluates all neighbouring states and moves to the one with the highest improvement.

Limitations

(1) Since it evaluates all neighbours, it takes more time and resources

(2) even though it selects the best move at each step. It cannot escape local maxima

(3) fails in plateaus and Ridges

12. Explain simulated annealing and write its algorithm

=>

Simulated Annealing (SA) is an optimization algorithm inspired by metallurgical annealing, where materials are heated and then cooled to remove defects. It helps escape local maxima by allowing occasional downward moves to explore better solutions

Working:

(1) Start with an initial solution

(2) Set a high temperature (T), which gradually cools down

(3) Select a random neighbor of the current solution

(4) calculate the energy difference (ΔE) between the new and current solution.

i) If the new solution is better, accept it.

ii) If the new solution is worse, accept it with a probability: $p = e^{-\Delta E/T}$ where e is Euler's number,

T is the current temperature

(5) Reduce the temperature gradually.

(6) Repeat until the temperature is very low or a stopping condition is met.

Algorithm:

Input : Initial state S, Initial temperature T, cooling rate α, Stopping temperature T_min

Output : Best found Solution S_best

1. Set current_state ← S
2. Set current_cost ← cost_function(current_state)
3. Set best_state ← current_state
4. Set best_cost ← current_cost
5. While T > T_min do:
   6. Generate a neighbor new_state of current_state
   7. Compute new_cost ← cost_function(new_state)
   8. Compute ΔE ← new_cost - current_cost
   9. If ΔE > 0 then
      10. Accept new_state as the current state
      11. Update current_state ← new_state
      12. Update current_cost ← new_cost
   13. Else
      14. Accept new_state with probability p = exp(ΔE/T)
      15. If random(0,1) < p then
         16. Update current_state ← new_state
         17. Update current_cost ← new_cost
   18. If current_cost > best_cost then
      19. Update best_state ← current_state

Page No.
Date

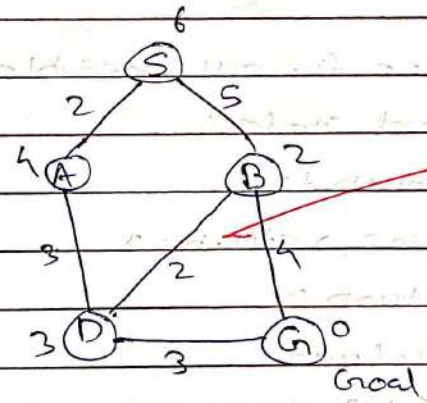
20. Update best_cost ← current_cost
21. Reduce temperature : T ← α * T
22. Return best_state

B. Explain A* Algorithm with an example

=>

A* is widely used graph search and pathfinding algorithm that finds the shortest path between a start node and a goal node. It is an informed search algorithm that uses both
↳ cost to reach the node (g(n))
↳ Estimated cost from the node to the goal h(n)
formula : f(n) = g(n) + h(n)

example :



Steps
↳ Initialize open list
↳ expand node with lowest f(n)
↳ update g(n), h(n) & f(n) for neighbouring
↳ Repeat until goal node is reached

Steps

Node	g(n)	h(n)	f(n) = g(n) + h(n)
start(S)	0	6	6
Expand A (S→A, cost = 2)	2	4	6
Expand B (S→B, cost = 5)	5	2	7
Expand D (A→D, cost = 2+3 = 5)	5	3	8
Expand G (B→G, cost = 5+4 = 9) (Goal reached)	9	0	9

14. Explain MiniMax Algorithm and draw game tree for Tic TAC Toe Game

=>

Minimax is a decision-making algorithm used in two-player games like Tic-Tac-Toe, chess and connect four. It helps in finding the best possible move for a player by assuming that the opponent also plays optimally. The algorithm alternates between Maximizing (for AI) and Minimizing (for opponent) and each state has a value

+1 → AI wins

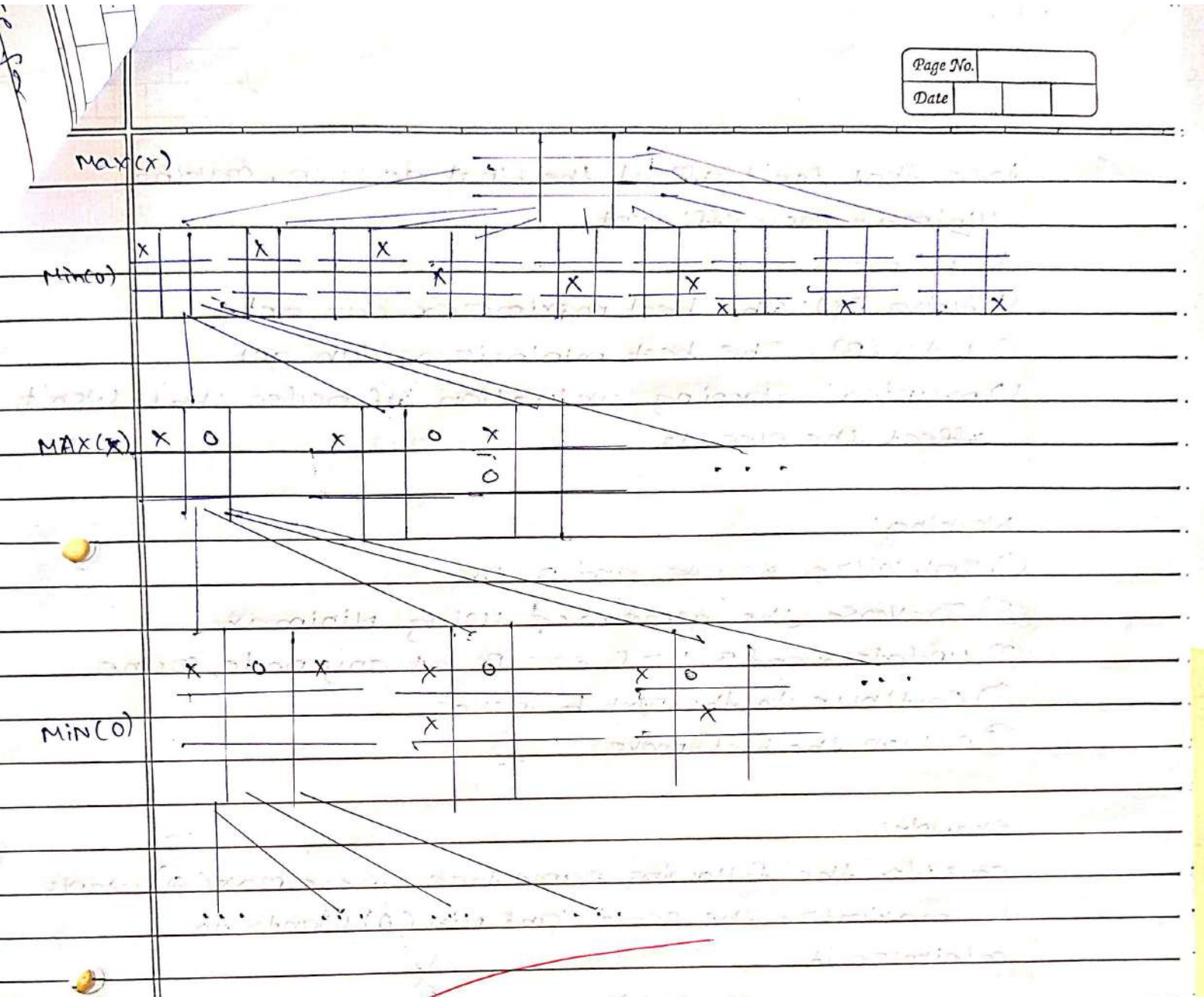
-1 → opponent wins

0 → draw.

Algorithm steps:

1. Generate game tree for all possible moves
2. Evaluate Terminal states:
   - If AI wins, return +1
   - If opponent wins, return -1
   - If draw, return 0
3. Backpropagate values:
   - Max player (AI) picks the maximum value
   - Min. player (opponent) picks the minimum value
4. Select the best move at the root level

Game Decision Tree :-



15. Explain Alpha-beta pruning algorithms for adversarial search with example.

⇒ Alpha-beta pruning is an optimization technique for the Minimax algorithm. It eliminates branches in the game

tree that don't affect the final decision, making Minimax more efficient.

Key Terms:

↳ Alpha (α): The best maximizer can get

↳ Beta (β): The best minimizer can get

↳ pruning: stopping evaluation of nodes that won't affect the result.

Working:

① Initialize $\alpha = -\infty$ and $\beta = +\infty$

② Traverse the game tree using Minimax

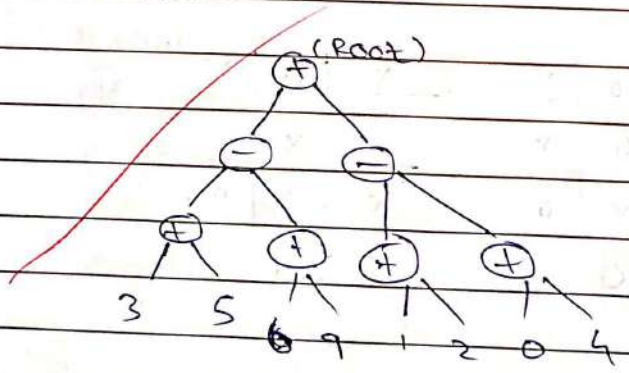
③ update α and β: If $\alpha >= \beta$ at any node, prune

④ Continue to the next branches

⑤ Return the best move

example:

Consider the following game tree where max (⊕) wants to maximize the score, and Min (⊖) wants to minimize it.



① Initialize: Alpha (α) $= -\infty$, Beta (β) $= +\infty$

② Evaluate left subtree

↳ Min (⊖) chooses between (3, 5) → selects 3 (smallest value)

↳ Max (⊕) chooses between (3, 6) → selects 6 (largest value)

↳ update x = 6

(3) Evaluate Right subtree:

↳ first node evaluated is 7

↳ Beta is updated β = 7

↳ Since x(6) ≥ β(7), we prune (skip checking 2, 0 and 4)

16. Explain Wumpus world environment giving it PEAS description. Explain how percept sequence is generated?

=>

The Wumpus world is a grid-based environment used in Artificial Intelligence (AI) to demonstrate logical agent-based reasoning. It is a partially observable, stochastic, sequential environment where an AI agent must navigate a cave-like world while avoiding dangers.

Structure of Wumpus World

(1) Grid-Based (4×4 or larger)

(2) Contains:

↳ Gold (Goal: pick it up)

↳ Wumpus (A monster that kills the agent)

↳ Pits (If agent falls, it dies)

↳ Walls (Boundaries of the world)

(3) Sensory perceptions (percepts)

↳ Breeze → Near a pit

↳ Stench → Near Wumpus

↳ Glitter → Gold is nearby

↳ Bump → Hits a wall

↳ Scream → Wumpus is dead.

PEAS description.

P : +1000 for finding gold, -1000 for falling into a pit or encountering wumpus, -1 for every move, -10 for shooting the arrow

E : A 4x4 grid with the agent, wumpus, pits, gold and walls

A : Move (Up, Down, Left, Right), Grab (Gold), Shoot (Arrow), Climb (Exit)

S : Breeze, Stench, Glitter, Bump, Scream.

How percept sequence is generated?

A percept sequence is a history of all sensor inputs received by the agent over time.

example:

(1) Agent starts at (1,1) → percepts = [Breeze] (pit is nearby)

(2) Moves to (1,2) → percepts = [Breeze, Stench] (pit + near wumpus)

(3) Moves to (2,2) → percepts = [Stench] (near wumpus)

(4) Moves to (3,2) → percepts = [Glitter] (gold is nearby)

Agent collects percepts at each step and makes logical decisions based on past percepts to avoid dangers and find gold.

17. Solve the following crypto-arithmetic problems.

SEND + MORE = MONEY

=>

5 4 3 2 1

S E N D

M O R E

C_3 C_2 C_1

M O N E Y

(1) From column 5, M=1, since it is only carry-over possible from sum of 2 single digit number in column 4

(2) To produce a carry from column 4 to column 5 'S+M' is atleast 9 so 'S = 8 or 9' so 'S+M = 9 or 10' and 'O = 0 or 1' But 'M=1', so 'O=0'.

(3) If there is carry from column 3 to 4 then E=9 and N=0. But O=0 so there is no carry and S=9 and C_3=0

(4) If there is no carry from column 2 to 3 then 'E = N' which is impossible, therefore there is carry and N=E+1 and C_2=1.

(5) If there is carry from column 1 to 2 then N+R = E mod 10 and N=E+1 so E+1R = E mod 10 so R=9 but S=9 so there must be carry from column 1 to 2. Therefore C1=1 and R=8

(6) To produce carry C1=1 from column 1 to 2, we must have D+E = 10+Y as Y cannot be 0/1 so D+E is atleast 12. As D is atmost 7 and E is atleast 5 (D cannot be 8 or 9 as it is already assigned). N is atmost 7 and N=E+1 so E=5 or 6.

(7) If E were 6 and D+E atleast 12 then D would be 7, but N=E+1 and N would also be 7 which is impossible Therefore, E=5 and N=6

(8) D+E is atleast 12 for that we get D=7 and Y=2

Hence final values:

S=9, E=5, N=6, D=7, M=1, O=0, R=8, Y=2

```
 9567          SEND
+1085    =>  + MORE
-----        -----
10652         MONEY
```

18. Consider the following axioms.

i) All people who are graduating are happy.

ii) All happy people are smiling.

iii) Someone is graduating.

Explain the following

1) Represent these axioms in first order predicate logic

i) $\forall x$ (Graduating(x) → Happy(x))

ii) $\forall x$ (Happy(x) → Smiling(x))

iii) $\exists x$ Graduating(x)

2) convert each formula to clause form using identity $p \rightarrow q = \sim p \vee q$

i) ~Graduating(x) ∨ Happy(x) - (1)

ii) ~Happ(x) ∨ Smiling(x) - (2)

iii) Graduating(A) - (3)

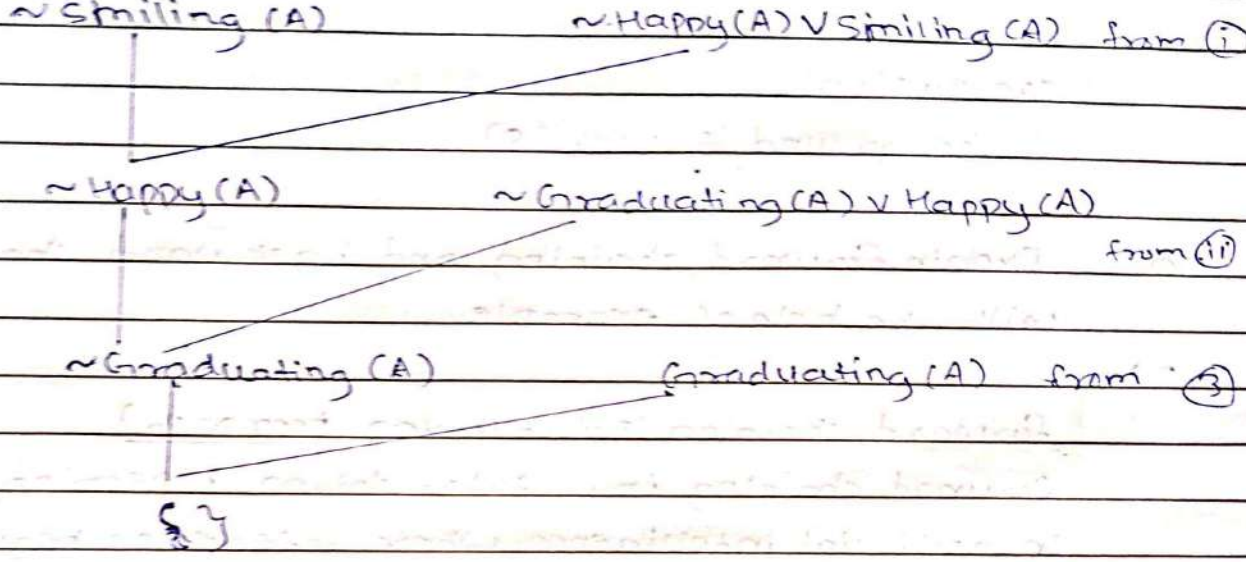
3) prove that "Is someone smiling?" using resolution technique. draw resolution tree.

from (2), substitute x = A (since Graduating(A) is given).

∴ ~Happy(A) ∨ Smiling(A) - (i)

From (i), substitute x = A ∴ ~Graduating(A) ∨ Happy(A) -(ii)

Resolution tree



As we get null subset our assumption of not Someone Smiling is wrong. Hence Someone is smiling.
Hence "Is Someone Smiling" becomes True

19. Explain Modus ponen with suitable example:

=>

Modus ponen is a fundamental rule of inference in logic which states that: If we have:

(1) A conditional statement: $p \rightarrow Q$ (If p, then Q)

(2) p is true

Then we conclude that Q is also true

Symbolic Representation

(1) $p \rightarrow Q$

(2) p (p is true) ∴ A (Therefore, Q is true)

example

premises

① If it rains, the ground will be wet ($p \rightarrow Q$)

② It is raining (p)

conclusion:

∴ The ground is wet (Q)

20. Explain forward chaining and backward chaining with the help of example.

=>

forward chaining (Data-Driven Approach)

forward chaining is a data-driven reasoning approach in artificial intelligence where inference begins with known facts and applies logical rules to derive new conclusions until a desired goal is reached. This method is often used in expert systems, rule-based systems and automated reasoning applications.

Algorithm:

① Start with known facts in the knowledge base

② Find rules whose premises (conditions) match known facts

③ Apply the rule, infer new facts

④ Repeat until the goal is reached or no more rules apply.

example: Medical diagnosis

let's assume we have these rules

① If a person has a cough & fever, THEN they might have flu

Page No.
Date

② If a person has flu, THEN they should take rest and fluids.

Given facts
The patient has cough & fever.

forward chaining process:
① From rule ① , we infer flu
② From rule ②, we infer the patient should take rest and fluids.

Conclusion: The system recommends rest and fluids

Backward chaining (Goal-driven Approach)
Backward chaining is a goal-driven reasoning that starts with the objective and works backward to determine if the given facts support it. Instead of processing all available data, it selectively applies rules to establish whether the goal can be derived from existing knowledge. This method is commonly used in theorem proving, AI-Based problem-solving and prolog-based expert systems

Algorithm:
① Start with the goal (conclusion to be proven)
② Check if the goal is already a fact
③ If not, find the rules where goal is in the conclusion
④ Check if the premises (conditions) of the rule are true
⑤ Repeat until the goal is proven or disproven.

Date

Example : Diagnosing flu

Goal : Does the patient have the flu ?

(1) To prove "flu", we check if (cough and fever) are true
(2) We ask if the patient has cough → yes
(3) We ask if the patient has fever → yes
(4) Since both conditions are met, flu is confirmed

Conclusion : The patient has the flu.

Name: Alok Yadav Div: D15C Roll No.: 59

Assignment 2

1. Use the following data set for question 1

82, 66, 70, 59, 90, 78, 76, 95, 99, 84, 88, 76, 82, 81, 91, 64, 79, 76, 85, 90

a. Find the Mean
   Total observations (n) = 20
   Mean = Total sum of observations/Number of Observations
   =(82+66+70+59+90+78+76+95+99+84+88+76+82+81+91+64+79+76+85+90)/20
   Mean = 1611

b. Find the Median
   Sorting the data: 59, 64, 66, 70, 76, 76, 76, 78, 79, 81, 82, 82, 84, 85, 88, 90, 90, 91, 95, 99
   As there are even numbers (20), the median = (10th + 11th number)/2
   Median = (81+82)/2=163/2=81.5

c. Find the Mode
   Since number 76 is appearing most number of times
   Hence, **Mode = 76**

d. Find the Interquartile Range
   Total numbers = 20
   Lower Half (Q1) = 59, 64, 66, 70, 76, 76, 76, 78, 79, 81
   Upper Half (Q3) = 82, 82, 84, 85, 88, 90, 90, 91, 95, 99
   Median of Q1 = (76+76)/2 = 76
   Median of Q3 = (88+90)/2 = 89
   Interquartile Range = Q3-Q1 = 89 - 76 = 13

2. 1) Machine Learning for Kids 2) Teachable Machine

a. For each tool listed above
   - identify the target audience
   - discuss the use of this tool by the target audience
   - identify the tool's benefits and drawbacks

b. From the two choices listed below, how would you describe each tool listed above? Why did you choose the answer?
   - Predictive analytic
   - Descriptive analytic

c. From the three choices listed below, how would you describe each tool listed above? Why did you choose the answer?
    - Supervised learning
    - Unsupervised learning
    - Reinforcement learning

1) Machine Learning for Kids

Target Audience:

- Primarily **school students**, **beginners**, and **educators** introducing machine learning (ML) concepts to kids (ages 8+).

Use by Target Audience:

- Used to **train ML models** through a **child-friendly interface**.
- Kids can upload text, image, number, or sound examples and train models to recognize patterns.
- Often integrated into platforms like **Scratch** or **App Inventor** to build simple AI-based projects (e.g., chatbots, image classifiers).

Benefits:

- **Highly intuitive** and designed for **educational purposes**.
- Provides **step-by-step learning** of ML concepts without coding.
- Encourages **experimentation** and creativity in young learners.

Drawbacks:

- Limited **scalability** — not suitable for advanced ML projects.
- Focuses on **basic ML principles**, not suitable for in-depth learning.
- Models are **simple** and not optimized for accuracy or performance.

Analytic Type: Descriptive analytic

Machine Learning for Kids is mainly used to help students explore and understand patterns in data by teaching how data can be labeled, grouped, or categorized. For example, a child can train the tool to recognize animals by providing labeled examples like "cat," "dog," or "horse." Once trained, the model classifies new inputs based on these examples. It doesn’t predict future

outcomes but rather explains the type of data being input and how it relates to what was previously learned. This makes it an example of descriptive analytics, as it focuses on summarizing and explaining existing data rather than making predictions.

Learning Type: Supervised learning

Teachable Machine uses labeled datasets where users upload or record examples and assign each one a label, such as "jumping" or "barking." The model learns the features of each label and uses this knowledge to classify new inputs. Since it learns from examples with known outcomes, it follows a supervised learning approach.

2) Teachable Machine

Target Audience:
Targeted at students, educators, creatives, hobbyists, and even non-programmers looking to explore AI.

Use by Target Audience:

- Allows users to train ML models using image, sound, or pose data directly from their browser.
- Users can upload examples, train a model, and export it for use in websites, apps, or physical devices (e.g., Arduino).
- Great for interactive demos, prototypes, or classroom activities.

Benefits:

- No coding required.
- Fast, real-time feedback.
- Enables easy model export to TensorFlow.js or other formats.
- Encourages exploration of real-world ML applications.

Drawbacks:

- Similar to ML for Kids, it's limited in complexity.
- Trained models are not fine-tuned for production or precision use cases.
- May give users a simplified idea of ML and miss advanced topics like overfitting, hyperparameters, etc.

Analytic Type: Descriptive Analytic

Teachable Machine is designed to help users categorize or recognize different types of input data such as images, sounds, or poses. For example, a person can train the model to identify various hand gestures or facial expressions. The tool focuses on real-time classification, showing what the current input is based on previously seen examples. It does not predict future trends or outcomes but instead helps interpret and explain the input data. This makes it a form of descriptive analytics, as it helps users understand and organize existing data rather than making forward-looking predictions.

Learning Type: Supervised Learning

Teachable Machine uses a supervised learning approach because it relies on labeled datasets. Users provide examples along with specific labels—like “jumping,” “sitting,” or “barking”—to train the model. The model then learns the features of each labeled category and uses this learning to classify new, unseen data. Since the training data includes correct answers, or outcomes, the model is guided during the learning process, which is the core characteristic of supervised learning.

3. Data Visualization: Read the following two short articles:

- Read the article Kakande, Arthur. February 12. “What’s in a chart? A Step-by-Step guide to Identifying Misinformation in Data Visualization.” Medium
- Read the short web page Foley, Katherine Ellen. June 25, 2020. “How bad Covid-19 data visualizations mislead the public.” Quartz
- Research a current event which highlights the results of misinformation based on data visualization.Explain how the data visualization method failed in presenting accurate information. Use newspaper articles, magazines, online news websites or any other legitimate and valid source to cite this example. Cite the news source that you found.

A notable example of misinformation through data visualization involved a Reuters graphic on Florida gun deaths following the enactment of the state’s “Stand Your Ground” law in 2005. At first glance, the visual elements—specifically, an inverted y-axis paired with a bold red background—led viewers to believe that gun deaths had dramatically fallen after the law’s

implementation, evoking a dramatic and reassuring narrative about public safety. In truth, however, the data revealed the opposite: firearm-related homicides increased from roughly 550 to over 800 in the years immediately following 2005. The misleading use of a reversed y-axis, which typically signals a downward trend, along with the red color scheme that can connotatively imply both danger and bloodshed, created an optical illusion that obscured the actual upward trend in deaths. This design choice, whether intentional or not, misled many viewers into the mistaken impression that the law had made Florida significantly safer when the evidence actually indicated a worsening situation.

Similarly, another instance of flawed data visualization emerged with Fox News' chart on the unemployment rate during President Obama's tenure. This graph was problematic because its y-axis did not start at zero—a technique known as "axis truncation" that compresses data and exaggerates small differences—and November's unemployment figure was inaccurately represented. In fact, the official data from the Bureau of Labor Statistics showed a decline from 9.0% to 8.6% unemployment over the reported period. However, the deliberately compressed scale and plotting errors in the chart minimized the perception of improvement, thereby distorting the actual trend. Both of these examples illustrate how selective scaling, mislabeling, and the use of unconventional axes in data visualizations can easily lead to public misinterpretation. Such techniques not only distort the underlying statistics but also reinforce specific narratives—whether aiming to create false optimism or to downplay positive changes—which can have significant consequences for public opinion and policy making.

Source:
https://medium.com/@Ana_kin/graphs-gone-wrong-misleading-data-visualizations-d4805d1c4700

4. **Train Classification Model and visualize the prediction performance of trained model required information**

- Data File: Classification data.csv
- Class Label: Last Column
- Use any Machine Learning model (SVM, Naïve Base Classifier)

Requirements to satisfy

- Programming Language: Python
- Class imbalance should be resolved
- Data Pre-processing must be used
- Hyper parameter tuning must be used
- Train, Validation and Test Split should be 70/20/10
- Train and Test split must be randomly done
- Classification Accuracy should be maximized
- Use any Python library to present the accuracy measures of trained model

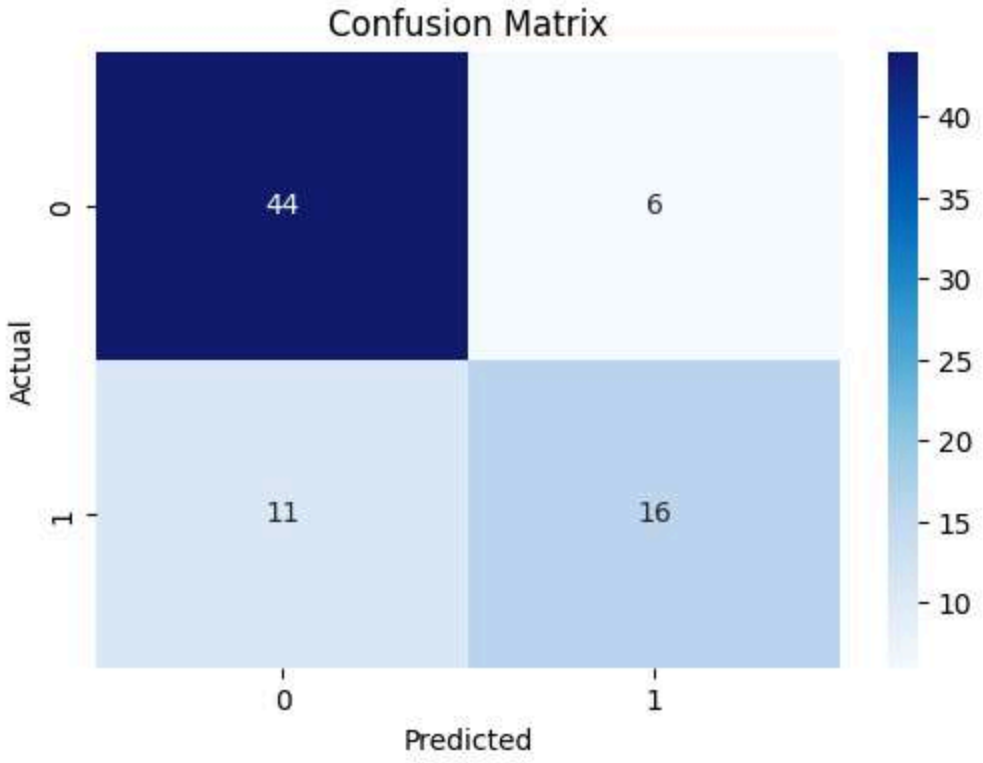
Dataset: Diabetes.csv

Pregnancies: Number of times the patient has been pregnant.
Glucose: Plasma glucose concentration after a 2-hour oral glucose tolerance test.
BloodPressure: Diastolic blood pressure (mm Hg).
SkinThickness: Triceps skin fold thickness (mm).
Insulin: 2-hour serum insulin (mu U/ml).
BMI: Body Mass Index (weight in kg / height in m^2).
DiabetesPedigreeFunction: A function that scores the likelihood of diabetes based on family history.
Age: Patient age in years.

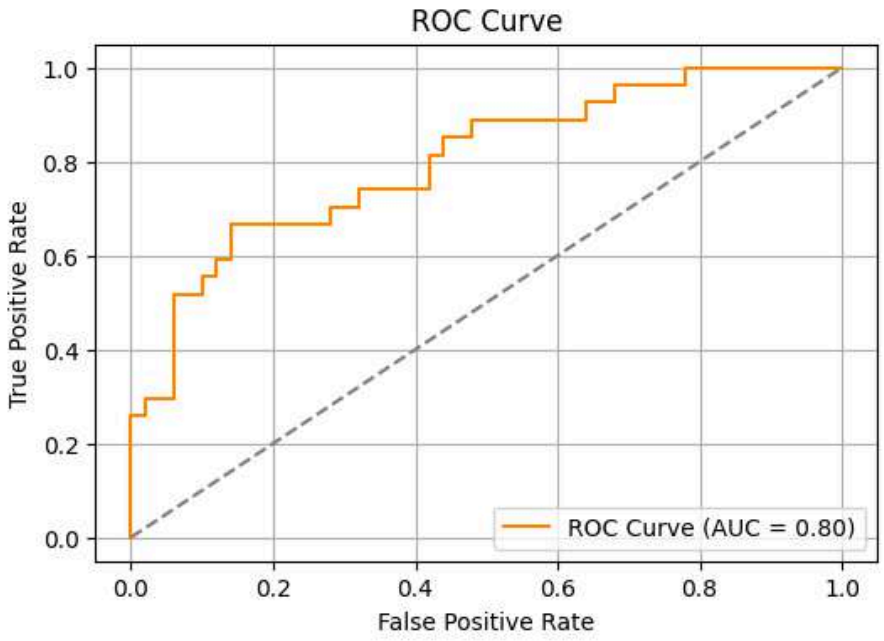
Model: **Support Vector Machine (SVM)**

Result:

```
Data loaded successfully.
Preprocessing complete.
Model training complete with best parameters.
Best Parameters: {'svm__C': 10, 'svm__gamma': 0.1, 'svm__kernel': 'rbf'}

Accuracy on Test Set: 0.7792

Classification Report:
              precision    recall  f1-score   support

           0       0.80      0.88      0.84        50
           1       0.73      0.59      0.65        27

    accuracy                           0.78        77
   macro avg       0.76      0.74      0.75        77
weighted avg       0.77      0.78      0.77        77
```



The SVM model achieved an accuracy of **77.92%** on the test set, with better performance in detecting non-diabetic cases (class 0) than diabetic ones (class 1). Although precision and recall for class 1 are lower, the overall classification is reasonably balanced. With hyperparameter tuning, the model shows good potential for early diabetes prediction.



The ROC analysis yielded an AUC of 0.80, indicating that the model demonstrates good discriminative ability between diabetic and non-diabetic cases. This suggests that, on average, the classifier correctly distinguishes between the two classes 80% of the time. While the performance is acceptable for practical applications, further refinement in feature engineering or model tuning could be explored to enhance predictive accuracy.

5. Train Regression Model and visualize the prediction performance of trained model

- Data File:Dry_bean_dataset.csv
- Independent Variable: 1st Column
- Dependent variables: Column 2 to 5

Use any Regression model to predict the values of all Dependent variables.
Requirements to satisfy:

- Programming Language: Python
- OOP approach must be followed
- Hyper parameter tuning must be used
- Train and Test Split should be 70/30
- Train and Test split must be randomly done

- Adjusted R2 score should more than 0.99
- Use any Python library to present the accuracy measures of trained model

Dataset: Dry_Bean_Dataset

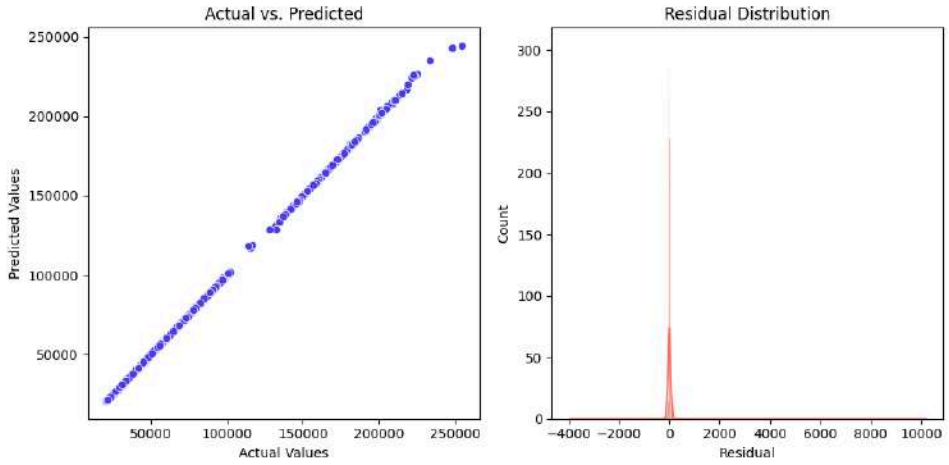
Dataset features:
Area: Total pixel count inside the bean region.
Perimeter: Distance around the bean boundary.
MajorAxisLength: Length of the longest axis of the bean.
MinorAxisLength: Length of the shortest axis of the bean.
AspectRatio: Ratio of major to minor axis.
Eccentricity: How elongated the bean is.
ConvexArea: Number of pixels in the convex hull of the bean.
EquivDiameter: Diameter of a circle with the same area as the bean.
Extent: Ratio of bean area to bounding box area.
Solidity: Ratio of bean area to convex hull area.
roundness: Circularity of the bean shape.
Compactness, ShapeFactor1, ShapeFactor2, ShapeFactor3, ShapeFactor4: Geometrical shape descriptors.

Model: **RandomForestRegressor**

Here we are predicting Area based on other features

Result:

```
Best parameters:  {'max_depth': 10, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 50}
R2 Score: 0.9999
Adjusted R2 Score: 0.9999
✅ Model meets the required Adjusted R² score.
```



The regression model, tuned with optimal hyperparameters, achieved an **R^2 and Adjusted R^2 score of 0.9999**, indicating an **excellent fit** with the data. It successfully satisfies the requirement of an Adjusted R^2 score above 0.99, demonstrating high predictive accuracy and generalization capability for the dry bean dataset.

6. What are the key features of the wine quality data set? Discuss the importance of each feature in predicting the quality of wine? How did you handle missing data in the wine quality data set during the feature engineering process? Discuss the advantages and disadvantages of different imputation techniques. (Refer dataset from Kaggle).

Key Features and Their Importance

1. Fixed Acidity: Represents non-volatile acids (tartaric, malic, citric) that remain relatively stable. Higher fixed acidity can contribute to tartness and affect overall balance.
2. Volatile Acidity: Primarily acetic acid content, which at high levels can give an unpleasant vinegar taste. Low volatile acidity is generally preferred for higher quality wines.
3. Citric Acid: Can add freshness and flavor to wines. It contributes to the wine's citrus character and can enhance perceived freshness.
4. Residual Sugar: Affects the sweetness of the wine. Different wine styles have different optimal levels, making this feature important but contextual.
5. Chlorides: Salt content in wine, which can influence taste perception. Excessive chlorides can negatively impact quality.
6. Free Sulfur Dioxide: Acts as a preservative and antioxidant. Appropriate levels help preserve wine quality while excessive amounts can create unpleasant aromas.
7. Total Sulfur Dioxide: Sum of free and bound forms of SO_2. Important for preservation but can be detrimental to flavor when too high.
8. Density: Related to alcohol and sugar content. Provides information about the wine's body and can indicate fermentation completeness.
9. pH: Affects chemical stability and microbial control. Wines with balanced pH tend to be more stable and often higher quality.
10. Sulphates: Additives that contribute to SO_2 levels and act as preservatives. Moderate levels help wine stability.
11. Alcohol: Higher alcohol content is often associated with higher quality ratings, as it contributes to body and can enhance flavor perception.

Handling Missing Data in the Wine Quality Dataset

Common Imputation Techniques and Their Trade-offs

1. **Mean/Median/Mode Imputation**
   - **Advantages**: Simple, fast implementation; preserves the mean (when using mean imputation)
   - **Disadvantages**: Reduces variance; ignores relationships between features; can distort distributions
2. **K-Nearest Neighbors (KNN) Imputation**
   - **Advantages**: Accounts for relationships between features; better preserves data structure
   - **Disadvantages**: Computationally expensive for large datasets; sensitive to outliers; requires parameter tuning
3. **Regression Imputation**
   - **Advantages**: Maintains relationships between variables; can be more accurate than simpler methods
   - **Disadvantages**: May overfit; assumes linear relationships; can reduce variance
4. **Multiple Imputation**
   - **Advantages**: Accounts for uncertainty in missing values; maintains variability in the dataset
   - **Disadvantages**: Computationally intensive; more complex to implement
5. **Machine Learning Based Imputation** (Random Forest, etc.)
   - **Advantages**: Can capture complex non-linear relationships; often provides more accurate imputations
   - **Disadvantages**: Computationally expensive; risk of overfitting; requires careful validation