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

Steps:

1.**Load data in Pandas.**

Load the file. 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 "")

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import MinMaxScaler, StandardScaler
import matplotlib.pyplot as plt
import seaborn as sns

# Step 1: Load the data
df = pd.read_csv('diabetes.csv')
```

2.**Description of dataset**

The description of the dataset gives the user an idea on what are the features, what is the count of rows and columns, etc. To achieve this, we can use the following commands.
Command 1: 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.
Command 2: df.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
Command 3: df.describe()
This command gives the details of all the values under all the features of the dataset. The command having no parameters gives information about count, max, min, standard deviation, top 25%ile, 50%ile, 75%ile and max value of the dataset.

```
print("Dataset Description:")
print(df.describe())
print("\nInfo:")
print(df.info())
```

```
Dataset Description:
       Pregnancies     Glucose  BloodPressure  SkinThickness     Insulin  \
count   768.000000  768.000000     768.000000     768.000000  768.000000
mean      3.845052  120.894531      69.105469      20.536458   79.799479
std       3.369578   31.972618      19.355807      15.952218  115.244002
min       0.000000    0.000000       0.000000       0.000000    0.000000
25%       1.000000   99.000000      62.000000       0.000000    0.000000
50%       3.000000  117.000000      72.000000      23.000000   30.500000
75%       6.000000  140.250000      80.000000      32.000000  127.250000
max      17.000000  199.000000     122.000000      99.000000  846.000000
```

3.**drop columns that aren't useful**

In data science, it is important to drop the columns that would not help the user while working on the dataset as it would make it cleaner to work with.

```
columns_to_drop = ['pregnancies']
df.drop(columns=columns_to_drop, inplace=True, errors='ignore')
```

4.**drop rows with maximum missing values**

It is important to drop the rows with maximum missing values as they would hinder the performance of the analysis and can lead to inaccuracies in the dataset

```
threshold = len(df.columns) // 2
df.dropna(thresh=threshold, inplace=True)
```

5.**Take care of the missing values** (filling it with the mean)

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.

```
[6] df.fillna(df.mean(), inplace=True)
```

6.**create dummy variables**

It is essential to create dummy variables to the columns that contain categorical data as most of the algorithms cannot understand the data directly. So they are classified as True and False or 0 and 1 which makes it easier. To create the dummy variables, we will list the columns that fall under categorical columns and then create another variable as pd_dummies to get the output of this. Pandas library provides a inbuilt function called as get_dummies which takes the data from the columns and create all the required dummy variables

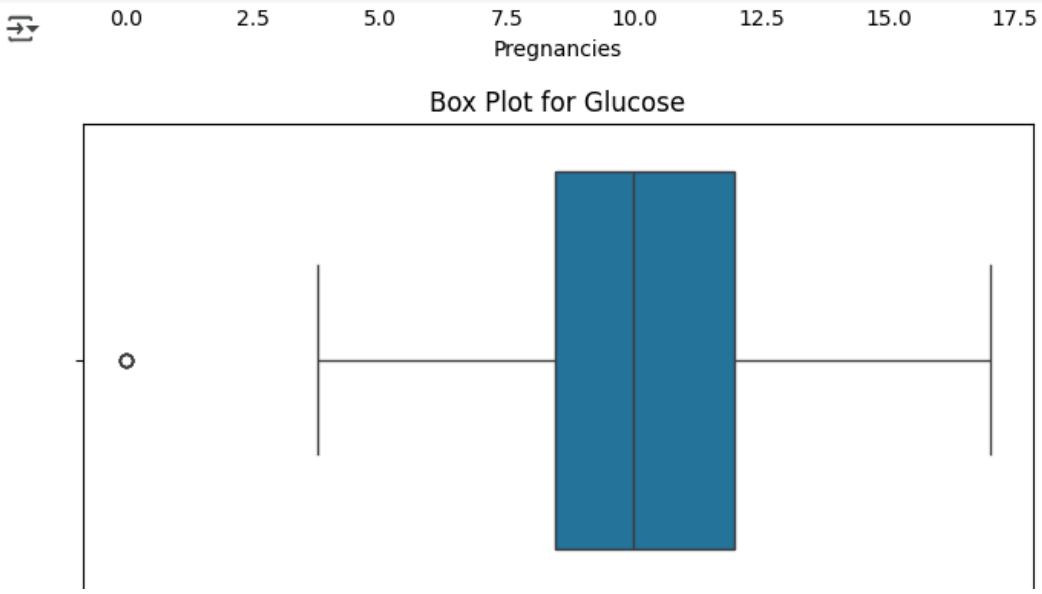
```
categorical_columns = df.select_dtypes(include=['object']).columns
df = pd.get_dummies(df, columns=categorical_columns, drop_first=True)
```

7.**Find out outliers.**

Used IQR method:

The interquartile range (IQR) is the range between the 1st quartile (Q1) and the 3rd quartile (Q3). Outliers are typically data points that fall below Q1 - 1.5 × IQR or above Q3 + 1.5 × IQR.

```
numerical_columns = df.select_dtypes(include=[np.number]).columns
for column in numerical_columns:
    plt.figure(figsize=(8, 4))
    sns.boxplot(x=df[column])
    plt.title(f"Box Plot for {column}")
    plt.show()
```



8.**Standardisation**

We can standardize columns using 1 of 2 methods. Either by their formulae, or by the SKLearn Library.

```
# Standardization
scaler_standard = StandardScaler()
standardized_data = scaler_standard.fit_transform(df)

standardized_df = pd.DataFrame(standardized_data, columns=df.columns)
```

Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	
0.6399472601593604	0.8483237946271883	0.149640752628208	0.9072699252723613	-0.6928905722954675	0.20401
-0.8448850534430228	-1.1233963609784168	-0.16054574674686284	0.5309015587207732	-0.6928905722954675	-0.6844
1.2338801856003137	1.9437238810747468	-0.2639412465385531	-1.2882122129452358	-0.6928905722954675	-1.1032
-0.8448850534430228	-0.9982077796701243	-0.16054574674686284	0.15453319216918512	0.12330164444496892	-0.4940
-1.1418515161634994	0.5040551960293843	-1.5046872440388366	0.9072699252723613	0.7658359427299933	1.4097
0.34298079743888377	-0.15318485583915073	0.2530362524198983	-1.2882122129452358	-0.6928905722954675	-0.8113
-0.2509521280020695	-1.3424763782679283	-0.9877097450803851	0.7190857419965673	0.07120426890834532	-0.1259
1.8278131110412668	-0.18448200116622385	-3.572597239872642	-1.2882122129452358	-0.6928905722954675	0.4197
-0.5479185907225461	2.38188391565377	0.046245252836517724	1.5345505361916747	4.021921913768968	-0.1894
1.2338801856003137	0.12848945210450713	1.3903867501284914	-1.2882122129452358	-0.6928905722954675	-4.0604
0.04601433471840714	-0.3409677278015893	1.183595750545111	-1.2882122129452358	-0.6928905722954675	0.7116
1.8278131110412668	1.4742667011686503	0.2530362524198983	-1.2882122129452358	-0.6928905722954675	0.7624
1.8278131110412668	0.5666494866835304	0.5632227517949692	-1.2882122129452358	-0.6928905722954675	-0.6209
-0.8448850534430228	2.1315067530371854	-0.47073224612193365	0.15453319216918512	6.65283937836846	-0.2402
0.34298079743888377	1.411672410514504	0.149640752628208	-0.09637905219854027	0.8266162141893876	-0.7859
0.9369137228798371	-0.6539391810723203	-3.572597239872642	-1.2882122129452358	-0.6928905722954675	-0.2528
-1.1418515161634994	-0.09059056518500455	0.7700137513783497	1.6600066583755375	1.3041754899417703	1.7524
0.9369137228798371	-0.4348591637828086	0.2530362524198983	-1.2882122129452358	-0.6928905722954675	-0.3036

9.**normalisation**.

We can normalize columns using 1 of 2 methods. Either by their formulae, or by the SKLearn Library

```
scaler_minmax = MinMaxScaler()
normalized_data = scaler_minmax.fit_transform(df)

normalized_df = pd.DataFrame(normalized_data, columns=df.columns)
```

1 to 25 of 768 entries Filter

Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	B
0.3529411764705882	0.7437185929648241	0.5901639344262295	0.3535353535353536	0.0	0.5007451
0.058823529411764705	0.4271356783919598	0.5409836065573771	0.29292929292929293	0.0	0.3964232
0.47058823529411764	0.9195979899497487	0.5245901639344263	0.0	0.0	0.3472429
0.058823529411764705	0.4472361809045226	0.5409836065573771	0.23232323232323235	0.1111111111111111	0.4187779
0.0	0.6884422110552764	0.3278688524590164	0.3535353535353536	0.19858156028368795	0.6423248
0.29411764705882354	0.5829145728643216	0.6065573770491803	0.0	0.0	0.3815201
0.1764705882352941	0.3919597989949749	0.4098360655737705	0.32323232323232326	0.10401891252955082	0.4619970
0.5882352941176471	0.577889447236181	0.0	0.0	0.0	0.5260804
0.11764705882352941	0.9899497487437187	0.5737704918032788	0.4545454545454546	0.6418439716312057	0.4545454
0.47058823529411764	0.628140703517588	0.7868852459016393	0.0	0.0	0.0
0.23529411764705882	0.5527638190954774	0.7540983606557378	0.0	0.0	0.5603576
0.5882352941176471	0.8442211055276382	0.6065573770491803	0.0	0.0	0.5663189
0.5882352941176471	0.6984924623115578	0.6557377049180328	0.0	0.0	0.4038748
0.058823529411764705	0.949748743718593	0.49180327868852464	0.23232323232323235	1.0	0.4485842
0.29411764705882354	0.8341708542713568	0.5901639344262295	0.19191919191919193	0.20685579196217493	0.3845007
0.4117647058823529	0.5025125628140703	0.0	0.0	0.0	0.4470938
0.0	0.592964824120603	0.6885245901639344	0.4747474747474748	0.2718676122931442	0.6825633
0.4117647058823529	0.5376884422110553	0.6065573770491803	0.0	0.0	0.4411326

```
standardized_df.to_csv('standardized_dataset.csv', index=False)
normalized_df.to_csv('normalized_dataset.csv', index=False)

print("Processing complete. Standardized and normalized datasets have been saved.")
```

```
Processing complete. Standardized and normalized datasets have been saved.
```

Conclusion:

Thus we have performed pre-processing on the dataset of Alzhiemers diseases and Healthy Aging data. To load the data into pandas, we used the read_csv() function of the pandas library to load it. For a description, we used various methods such as head(), info(), describe() which gave information such as data types, mean, max, min, count, etc. Using drop() command, we dropped the columns off the dataset that would not have had much impact on the analysis. For dropping rows with maximum missing values, we implemented a series of commands on our dataset that checked each entry for missing values, selected the max from amongst them and then deleted those rows with maximum missing values. This is done so that it does not bring up the skewness of the dataset, we analysed the data which is available by taking graphs of it, and used the apt method (mean, median, mode) to fill up the missing values of the database. After manual analysis, we decided to use the IQR index technique to check out the outliers of the dataset. Now, while analysis, data with higher values can tend to affect the analysis. To reduce this anomaly, we normalize and standardize the graph based on minmax/standard deviation methods to get the values to a reasonable range for the analysis to take place smoothly.

Aim: Perform following data visualization and exploration on your selected dataset.

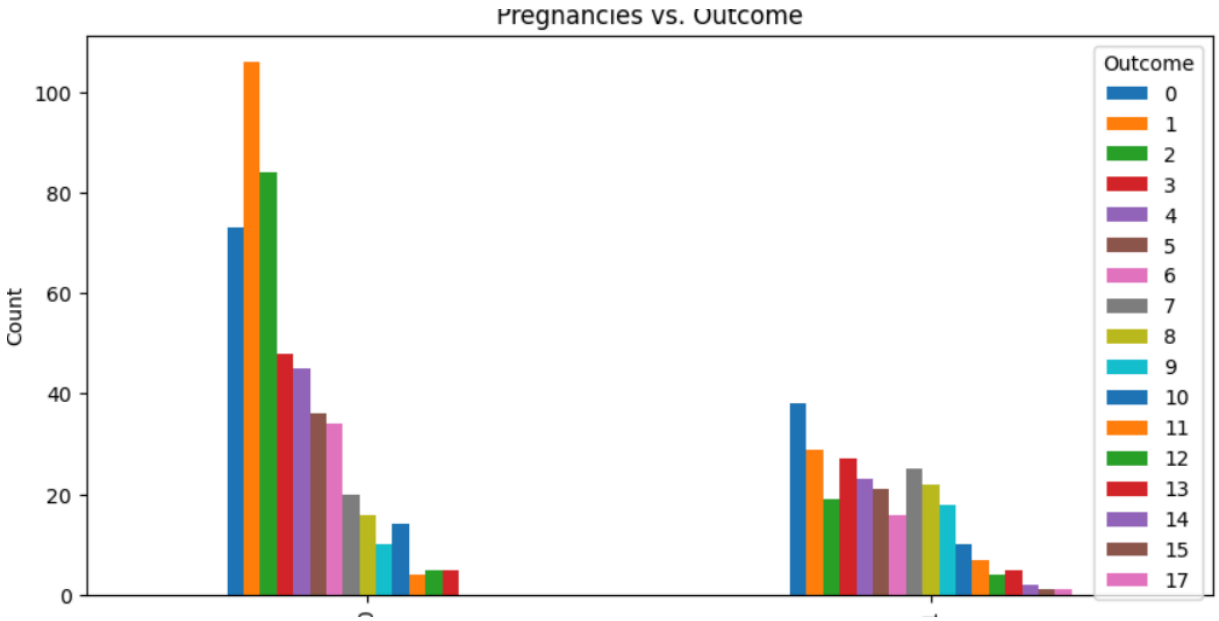
- Create bar graph, contingency table using any 2 features.
- Plot Scatter plot, box plot, Heatmap using seaborn.
- Create histogram and normalized Histogram.
- Describe what this graph and table indicates.
- Handle outlier using box plot and Inter quartile range

Steps:

1.**Create Bar Graph and Contingency Table**

Seaborn and Matplotlib are used together to create an effective bar graph. Matplotlib sets the figure size, labels, and titles to ensure clarity in the visualization. Seaborn's barplot function is used to plot categorical data, where the x-axis represents different categories (Class), and the y-axis represents the corresponding data values. The estimator='mean' argument allows automatic aggregation, providing the average value for each category. This helps in identifying trends and comparisons across different classes efficiently. The rotation of x-axis labels enhances readability.

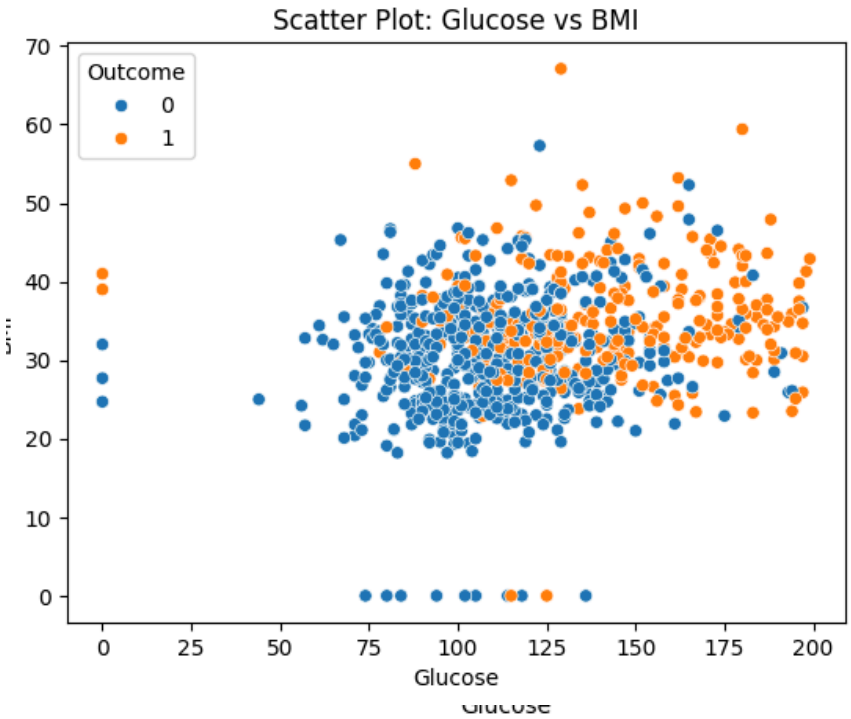
```
table = pd.crosstab(df['Outcome'], df['Pregnancies'])
print("Contingency Table:\n", table)

table.plot(kind='bar', figsize=(10, 5))
plt.title("Pregnancies vs. Outcome")
plt.xlabel("Pregnancies")
plt.ylabel("Count")
plt.legend(title='Outcome')
plt.show()
```

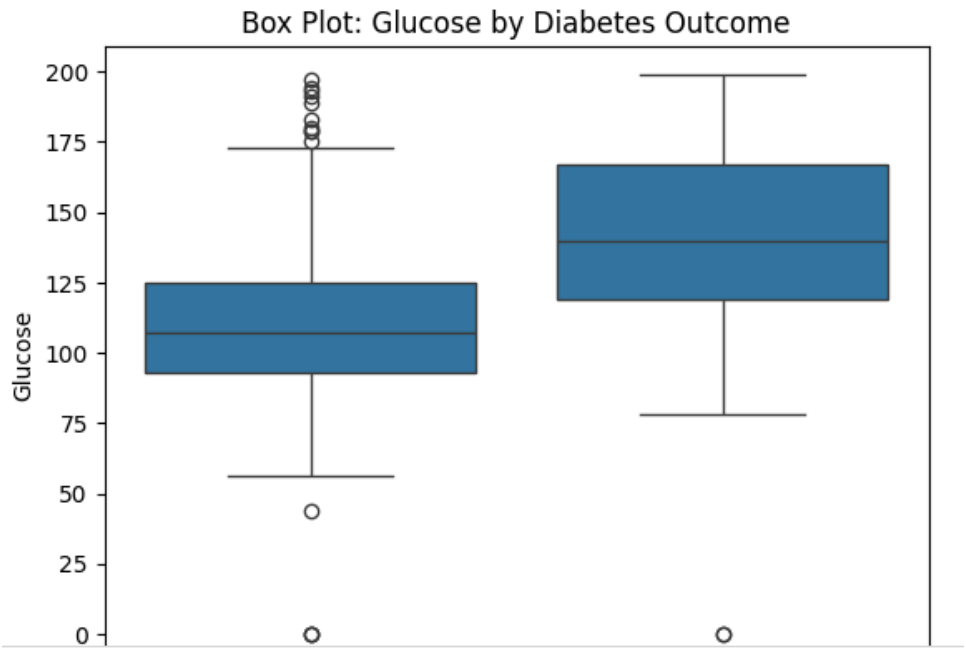


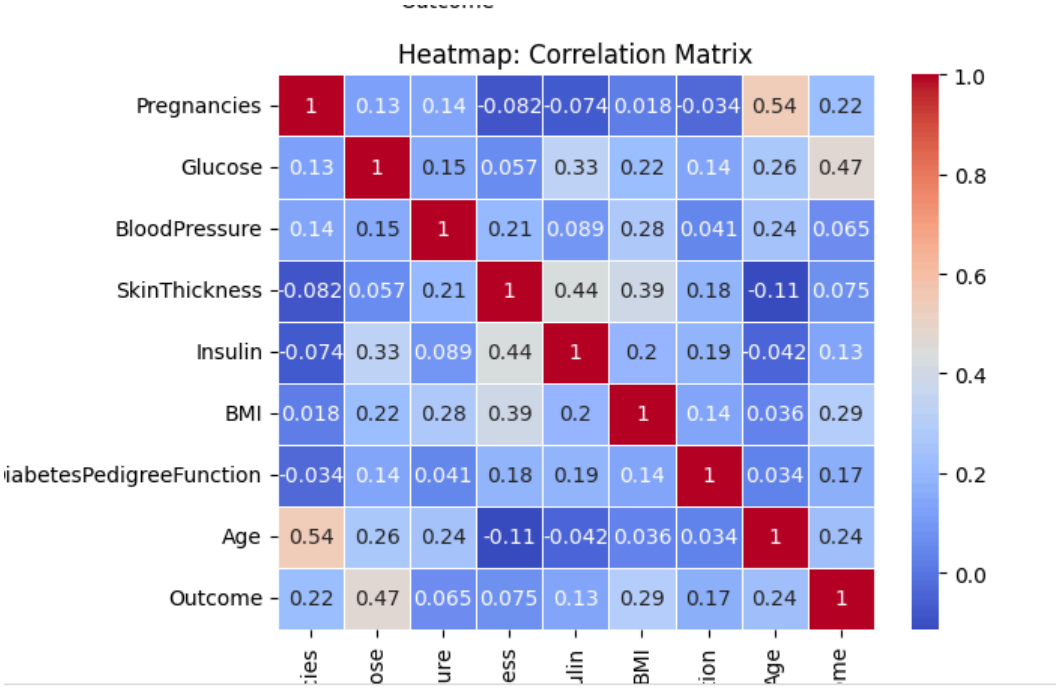
2.**Plot Scatter Plot, Box Plot, and Heatmap**

In this part, we have plotted Heatmap and seaborn as they are relevant to our dataset. A heatmap is used to visualize the contingency table, which represents the relationship between two categorical variables, Region and Class. Seaborn's heatmap function is applied to display data distribution using a color gradient (cmap="coolwarm"), where darker shades indicate higher values. The annot=True argument ensures that numerical values are displayed in each cell for clarity. Matplotlib is used to set figure size and labels, enhancing readability. This visualization helps in quickly identifying trends and correlations between categories

```
sns.scatterplot(x=df['Glucose'], y=df['BMI'], hue=df['Outcome'])
plt.title("Glucose vs. BMI")
plt.show()

sns.boxplot(x=df['Outcome'], y=df['BloodPressure'])
plt.title("Boxplot of Blood Pressure by Outcome")
plt.show()

plt.figure(figsize=(10, 6))
sns.heatmap(df.corr(), annot=True, cmap='coolwarm', linewidths=0.5)
plt.title("Correlation Heatmap")
plt.show()
```


Scatter Plot: Glucose vs BMI
Outcome
0
1
70
60
50
40
30
20
10
0
0
25
50
75
100
125
150
175
200
Glucose

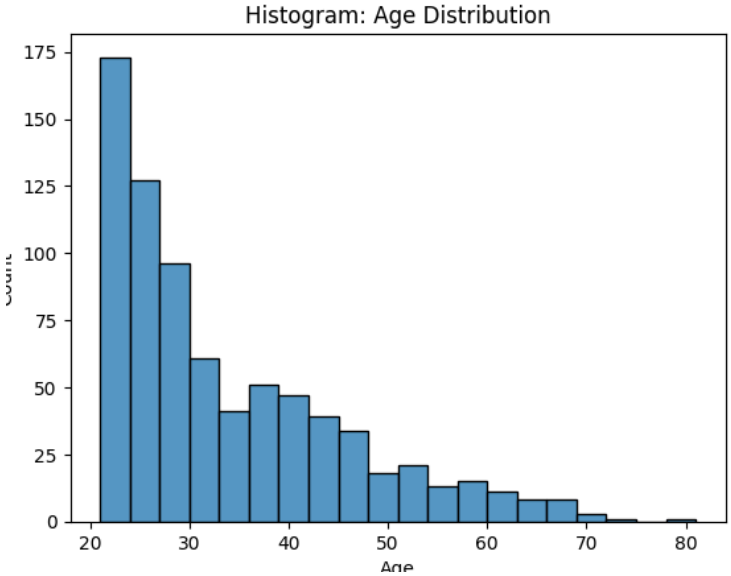

Box Plot: Glucose by Diabetes Outcome
Glucose
200
175
150
125
100
75
50
25
0

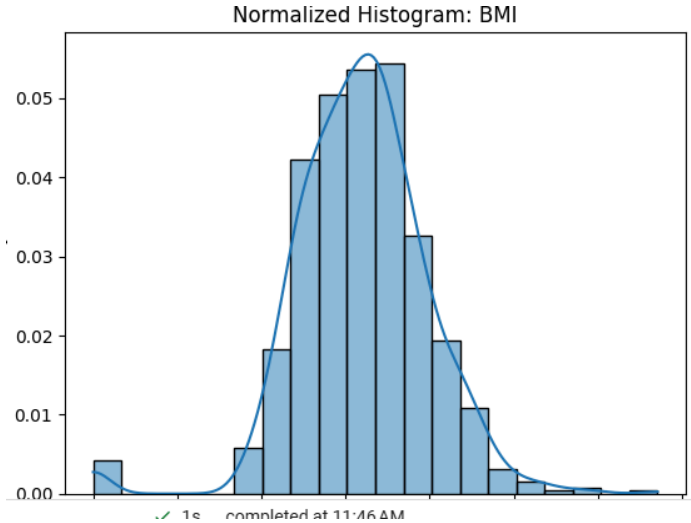


3.**Create Histogram and Normalized Histogram**

A histogram is used to visualize the distribution of Data_Value, showing how frequently different values occur. The sns.histplot function from Seaborn is used with bins=30 to divide data into 30 intervals, providing a detailed view of the distribution. The kde=True parameter adds a Kernel Density Estimate (KDE) curve for a smooth representation. The color is set to blue for better visibility. This helps in understanding data skewness, spread, and common value ranges.

```
plt.hist(df['Age'], bins=20, alpha=0.7, color='blue', edgecolor='black')
plt.title("Age Distribution Histogram")
plt.xlabel("Age")
plt.ylabel("Frequency")
plt.show()
```





4.Describe What This Graph and Table Indicates

Bar Graph: The bar graph shows the distribution of diabetes outcomes. You can easily see the number of individuals with and without diabetes.

Contingency Table: The table shows the relationship between `Pregnancies` and `Outcome`. It tells you how many people with different numbers of pregnancies have diabetes (1) or not (0).

Scatter Plot: This shows how `Glucose` and `BMI` are related, and whether people with and without diabetes have different patterns in these variables.
Box Plot: The box plot illustrates the distribution of `Glucose` levels for people with diabetes vs. those without it. It helps identify the spread, median, and potential outliers in the glucose levels.
Heatmap: The heatmap shows how correlated the numerical features are with each other. For example, `BMI` might be highly correlated with `Insulin` or `Age`.
Histograms: The regular histogram of `Age` shows the frequency distribution of age in the dataset, while the normalized histogram for `BMI` shows how BMI values are distributed across the population in terms of probability density.

5.**Handle Outlier Using Box Plot and Inter Quartile Range (IQR)**

Outliers in a dataset can significantly impact statistical analysis, making it essential to identify and handle them effectively. A boxplot is a useful visualization tool for detecting outliers, as it highlights the spread of data, median, and interquartile range (IQR). The IQR method is a robust technique for outlier detection, where values falling outside Q1 - 1.5 * IQR and Q3 + 1.5 * IQR are considered potential outliers. By computing Q1 (25th percentile) and Q3 (75th percentile), we determine the IQR and set boundaries to identify extreme values. Outliers can distort measures of central tendency and variability, potentially leading to misleading insights. Once detected, these values can be analyzed to determine if they result from data entry errors, anomalies, or natural variability. Proper handling, such as transformation, capping, or removal, depends on the dataset's context. The combination of a boxplot and IQR analysis ensures a balanced approach to managing outliers.

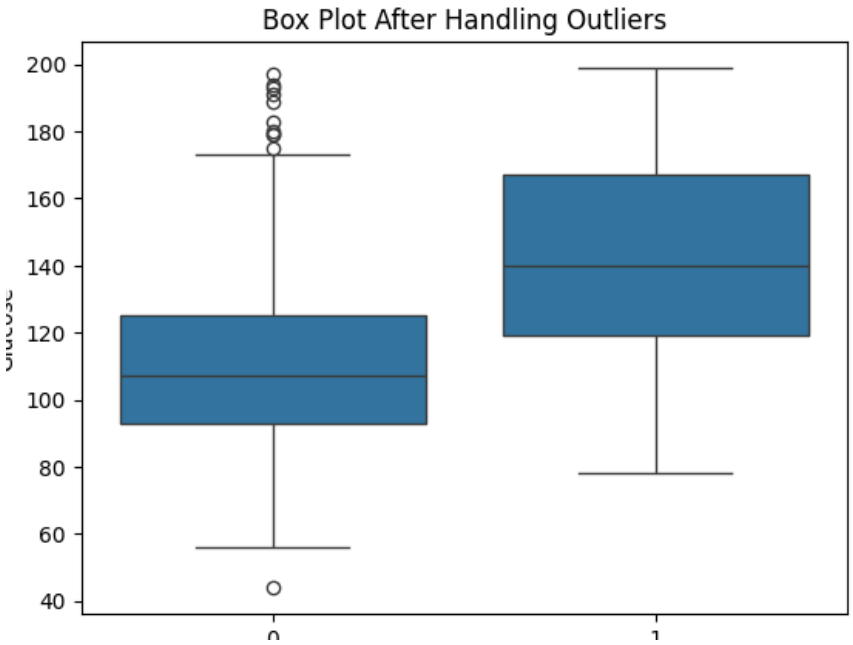
```
Q1 = df.quantile(0.25)
Q3 = df.quantile(0.75)
IQR = Q3 - Q1
```

```
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR
```

```
df_no_outliers = df[~((df < lower_bound) | (df > upper_bound)).any(axis=1)]
print("Data shape before outlier removal:", df.shape)
print("Data shape after outlier removal:", df_no_outliers.shape)
```

```
Data shape before outlier removal: (768, 9)
Data shape after outlier removal: (639, 9)
```

```
sns.boxplot(x='Outcome', y='Insulin', data=df_no_outliers)
plt.title('Box Plot of Insulin by Outcome (Outliers Removed)')
plt.show()
```



6.Conclusion: This experiment provided valuable insights into the dataset through various visualizations. The bar graph highlighted variations in Data_Value across different categories, helping identify dominant and underrepresented classes. The contingency table and heatmap effectively showcased regional differences, revealing areas with higher concentrations in specific categories.The histogram and KDE curve indicated a multimodal distribution, suggesting the presence of distinct subgroups within the data. The box plot and IQR method identified several outliers, with extreme values that could potentially skew statistical analysis. Detecting and managing these outliers is crucial to maintaining data accuracy and ensuring reliable conclusions. Overall, this study reinforced the importance of exploratory data analysis (EDA) in understanding data distribution, detecting patterns, and handling anomalies. These visualizations provide a clearer perspective on data trends, aiding in better decision-making and more precise interpretations of the dataset.

Experiment No - 3

Aim: To perform data modeling.

Theory:

Data partitioning is a crucial step in machine learning, where we divide the dataset into training and test sets. The training set, usually around 75% of the data, is used to develop the model, while the test set (25%) evaluates its performance. This ensures that the model generalizes well to new data rather than memorizing patterns from the training set. Proper partitioning prevents overfitting and improves real-world accuracy.

To verify that the partitioning is correct, we use visualization techniques such as bar graphs, histograms, and pie charts. These help confirm that the dataset maintains its proportions after splitting and that no class or feature distribution is unintentionally skewed. Counting the records in both sets ensures that the correct percentage of data has been allocated for training and testing.

A two-sample Z-test is a statistical method used to validate whether the training and test sets come from the same population. This test compares the means of numerical features in both sets to check if there is a significant difference. The Z-test is ideal for large datasets (sample size >30) and assumes normal distribution. If the p-value from the test is greater than 0.05, it indicates that the datasets are similar, confirming a good split. However, if the p-value is less than 0.05, it suggests that the partitioning may be biased, requiring a reassessment of the split method.

Overall, partitioning, visualization, and statistical validation together ensure a well-balanced dataset that helps in building an accurate and reliable machine learning model.

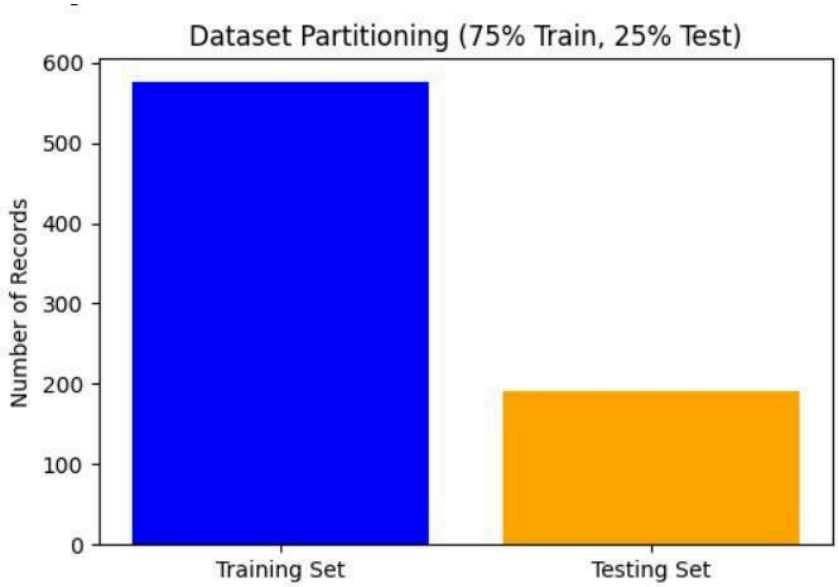
Steps:

1) Data partitioning is the process of dividing a dataset into two subsets: a training set and a test set. Typically, 75% of the data is used for training, where the model learns patterns, and 25% is used for testing to evaluate performance on unseen data. This division ensures that the model generalizes well and does not simply memorize the training examples. Proper partitioning helps in reducing overfitting and provides a fair evaluation of the model's effectiveness. The train_test_split function from sklearn.model_selection is commonly used to achieve this.

```python
# Step 1: Import Libraries
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from statsmodels.stats.proportion import proportions_ztest

# Step 2: Load Dataset (Upload your dataset to Colab)
from google.colab import files
uploaded = files.upload()

# Load dataset into a DataFrame
df = pd.read_csv("diabetes.csv")  # ✅ Uses your dataset name
```

2) Visualization of the SplitAfter splitting the dataset, visualizing the distribution of training and test sets ensures that the split maintains the original dataset's characteristics. Bar graphs can be used to compare the number of records in both sets, while histograms and pie charts help check whether numerical and categorical feature distributions remain balanced. If a class or feature is disproportionately represented in either subset, the split may need adjustment. The matplotlib.pyplot library in Python helps create such visualizations to confirm a proper split.

```python
# Step 5: Visualize the Partitioning
plt.figure(figsize=(6,4))
plt.bar(["Training Set", "Testing Set"], [train_size, test_size], color=["blue", "orange"])
plt.ylabel("Number of Records")
plt.title("Dataset Partitioning (75% Train, 25% Test)")
plt.show()
```



3) Counting the number of records in both training and test sets ensures that the split has been performed correctly.The expected number of samples in each set is calculated using simple percentage formulas, such as Training Size = Total Data × 0.75 and Testing Size = Total Data× 0.25. By printing the lengths of the training and test sets after splitting, we can verify if the proportions match the intended split. This step helps in detecting potential errors in dataset Partitioning.

```
# Step 6: Validate Partition with Two-Sample Z-Test
train_diabetes_count = train_df["Outcome"].sum()
test_diabetes_count = test_df["Outcome"].sum()

# Total counts in each set
train_total = len(train_df)
test_total = len(test_df)
```

```
Training Set Size: 576 records
Testing Set Size: 192 records
```

4) A two-sample Z-test is used to statistically verify whether the training and test sets come from the same distribution. It compares the means of numerical features in both subsets and checks for significant differences. If the p-value from the Z-test is greater than 0.05, the split is valid, meaning there is no significant difference between the two sets. However, if the p-value is below 0.05, the dataset may not be evenly distributed, requiring a reassessment of the split. The scipy.stats.z test function in Python is commonly used to perform this validation.

```python
# Perform the Z-test
count = [train_diabetes_count, test_diabetes_count]
nobs = [train_total, test_total]
z_stat, p_value = proportions_ztest(count, nobs)

print(f"Z-Statistic: {z_stat}")
print(f"P-Value: {p_value}")

# Step 7: Interpretation
if p_value > 0.05:
    print("No significant difference in class proportions. Partitioning is valid.")
else:
    print("Significant difference detected. Consider adjusting the split method.")
```

```
Z-Statistic: 0.0
P-Value: 1.0
No significant difference in class proportions. Partitioning is valid.
```

Conclusion: Proper dataset partitioning, visualization, and statistical validation ensure a balanced and unbiased split, leading to reliable model performance. This approach minimizes overfitting and improves the model's generalization to real-world data.

Experiment - 4

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

DataSet Link : Diabetes

Theory :

Correlation and association tests are used in statistics to measure relationships between variables. Pearson's Correlation Coefficient quantifies the linear relationship between two continuous variables, ranging from -1 (strong negative correlation) to +1 (strong positive correlation), with 0 indicating no correlation. It assumes normally distributed data and is sensitive to outliers. In contrast, Spearman's Rank Correlation is a non-parametric test that evaluates the monotonic relationship between two variables by ranking data points. It is useful when the relationship is non-linear and is less affected by extreme values, making it suitable for ordinal or skewed data.

Another non-parametric alternative is Kendall's Rank Correlation, which measures the strength of association based on the concordance of data pairs. It is more robust for small sample sizes and ties in data, providing a more reliable assessment of rank-based dependencies. Unlike Pearson's method, both Spearman and Kendall's tests do not assume normality and work well for ordinal data. These correlation tests help understand variable dependencies in various fields like finance, medicine, and social sciences.

The Chi-Squared Test is used for testing relationships between categorical variables. It evaluates whether observed frequencies significantly differ from expected frequencies under the assumption of independence. This test is widely used in feature selection, independence testing, and market research to determine associations in categorical data. Unlike correlation tests that measure strength and direction, the Chi-Square test assesses whether variables are statistically dependent without indicating the strength of association. These statistical methods collectively play a vital role in data-driven decision-making and hypothesis testing in research.

Output:

1.Importing Required Libraries :

Importing required libraries ensures that all necessary tools for data handling, analysis, visualization, and modeling are available. It enables efficient execution of tasks like data manipulation, statistical analysis, and machine learning.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
from sklearn.feature_selection import chi2
```

2. Loading dataset:

The purpose of loading a dataset is to import data into a Python environment for analysis, preprocessing, and visualization. It serves as the first step in data processing to enable further exploration and model building.

```
file_path = "/content/sample_data/Diabetes.xlsx"

# Load Excel file
df = pd.read_excel(file_path)

# Display the first 5 rows
df.head()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

3. Exploratory Data Analysis (EDA):

This process helps understand the dataset's structure, identify missing values, and detect data inconsistencies. It provides summary statistics to analyze distributions, central tendencies, and variability before further processing.

```python
# Display column names and data types
df.info()

# Check for missing values
print("\nMissing Values:\n", df.isnull().sum())

# Summary statistics
df.describe()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.471876	33.240885	0.348958
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.331329	11.760232	0.476951
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.000000	0.000000
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.243750	24.000000	0.000000
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.372500	29.000000	0.000000
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.000000	1.000000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	1.000000

1. Pearson's Correlation Coefficient:

Pearson's Correlation Coefficient (denoted as **r**) measures the **linear** relationship between two continuous variables.
Values range from **-1 to +1**:

- **+1**: Perfect positive correlation
- **0**: No correlation
- **-1**: Perfect negative correlation

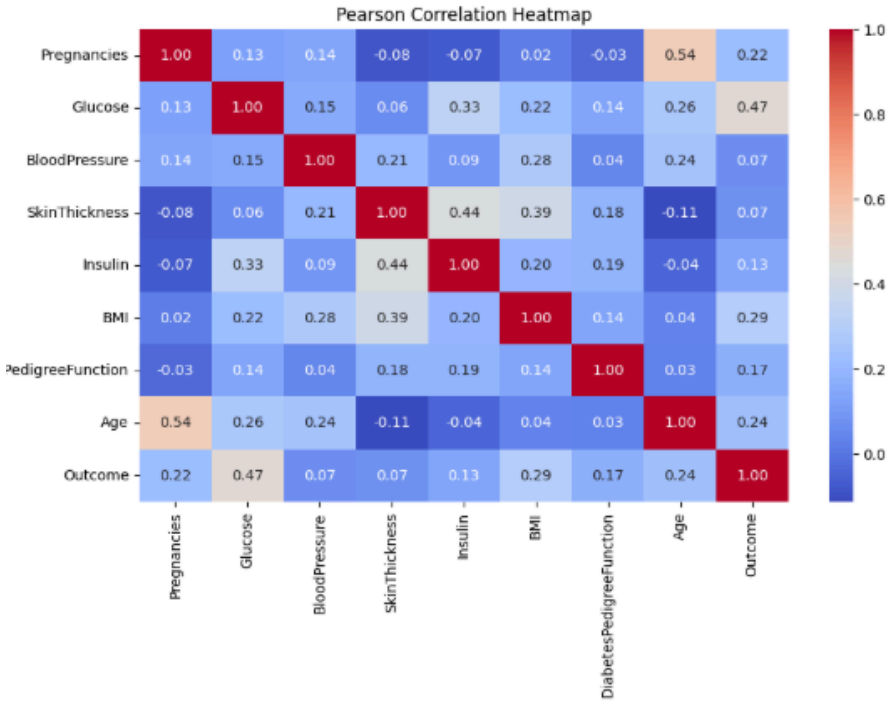
The formula for Pearson's Correlation Coefficient is:

$$r = \frac{\sum(X_i - \bar{X})(Y_i - \bar{Y})}{\sqrt{\sum(X_i - \bar{X})^2 \sum(Y_i - \bar{Y})^2}}$$

```python
#Pearson's Correlation Coefficient
pearson_corr = df.corr(method='pearson')
print("\nPearson Correlation Coefficient:\n", pearson_corr)

# Heatmap of Pearson correlation
plt.figure(figsize=(10, 6))
sns.heatmap(pearson_corr, annot=True, cmap="coolwarm", fmt=".2f")
plt.title("Pearson Correlation Heatmap")
plt.show()
```

```
Pearson Correlation Coefficient:
                          Pregnancies   Glucose  BloodPressure  SkinThickness  \
Pregnancies                  1.000000  0.129459       0.141282      -0.081672
Glucose                      0.129459  1.000000       0.152590       0.057328
BloodPressure                0.141282  0.152590       1.000000       0.207371
SkinThickness               -0.081672  0.057328       0.207371       1.000000
Insulin                     -0.073535  0.331357       0.088933       0.436783
BMI                          0.017683  0.221071       0.281805       0.392573
DiabetesPedigreeFunction    -0.033523  0.137337       0.041265       0.183928
Age                          0.544341  0.263514       0.239528      -0.113970
Outcome                      0.221898  0.466581       0.065068       0.074752

                           Insulin       BMI  DiabetesPedigreeFunction  \
Pregnancies              -0.073535  0.017683                 -0.033523
Glucose                   0.331357  0.221071                  0.137337
BloodPressure             0.088933  0.281805                  0.041265
SkinThickness             0.436783  0.392573                  0.183928
Insulin                   1.000000  0.197859                  0.185071
BMI                       0.197859  1.000000                  0.140647
DiabetesPedigreeFunction  0.185071  0.140647                  1.000000
Age                      -0.042163  0.036242                  0.033561
Outcome                   0.130548  0.292695                  0.173844

                               Age   Outcome
Pregnancies               0.544341  0.221898
Glucose                   0.263514  0.466581
BloodPressure             0.239528  0.065068
SkinThickness            -0.113970  0.074752
Insulin                  -0.042163  0.130548
BMI                       0.036242  0.292695
DiabetesPedigreeFunction  0.033561  0.173844
Age                       1.000000  0.238356
Outcome                   0.238356  1.000000
```

Heatmap of Pearson correlation :



2. Spearman's Rank Correlation

- Spearman's Rank Correlation (denoted as ρ, rho) measures the monotonic relationship between two variables.
- It does not require normally distributed data.
- If ranks of two variables are related, it indicates correlation.
- The formula is:

$$\rho = 1 - \frac{6 \sum d_i^2}{n(n^2 - 1)}$$

```python
# Spearman's Rank Correlation
spearman_corr = df.corr(method='spearman')
print("\nSpearman's Rank Correlation:\n", spearman_corr)
```

```
Spearman's Rank Correlation:
                          Pregnancies   Glucose  BloodPressure  SkinThickness  \
Pregnancies                  1.000000  0.130734       0.185127      -0.085222
Glucose                      0.130734  1.000000       0.235191       0.060022
BloodPressure                0.185127  0.235191       1.000000       0.126486
SkinThickness               -0.085222  0.060022       0.126486       1.000000
Insulin                     -0.126723  0.213206      -0.006771       0.541000
BMI                          0.000132  0.231141       0.292870       0.443615
DiabetesPedigreeFunction    -0.043242  0.091293       0.030046       0.180390
Age                          0.607216  0.285045       0.350895      -0.066795
Outcome                      0.198689  0.475776       0.142921       0.089728

                            Insulin       BMI  DiabetesPedigreeFunction  \
Pregnancies               -0.126723  0.000132                 -0.043242
Glucose                    0.213206  0.231141                  0.091293
BloodPressure             -0.006771  0.292870                  0.030046
SkinThickness              0.541000  0.443615                  0.180390
Insulin                    1.000000  0.192726                  0.221150
BMI                        0.192726  1.000000                  0.141192
DiabetesPedigreeFunction   0.221150  0.141192                  1.000000
Age                       -0.114213  0.131186                  0.042909
Outcome                    0.066472  0.309707                  0.175353

                                Age   Outcome
Pregnancies                0.607216  0.198689
Glucose                    0.285045  0.475776
BloodPressure              0.350895  0.142921
SkinThickness             -0.066795  0.089728
Insulin                   -0.114213  0.066472
BMI                        0.131186  0.309707
DiabetesPedigreeFunction   0.042909  0.175353
Age                        1.000000  0.309040
Outcome                    0.309040  1.000000
```

3.Kendall's Rank Correlation

Theory:

- Kendall's Tau (τ) measures the ordinal association between two variables.
- It counts concordant and discordant pairs:
  - Concordant pairs: If one variable increases, the other also increases.
  - Discordant pairs: One increases while the other decreases.
- The formula is:

$$\tau = \frac{(C - D)}{\frac{1}{2}n(n-1)}$$

```
[7]  #Kendall's Rank Correlation
     kendall_corr = df.corr(method='kendall')
     print("\nKendall's Rank Correlation:\n", kendall_corr)
```

```
Kendall's Rank Correlation:
                          Pregnancies   Glucose  BloodPressure  SkinThickness  \
Pregnancies                  1.000000  0.091323       0.135440      -0.064401
Glucose                      0.091323  1.000000       0.159961       0.039046
BloodPressure                0.135440  0.159961       1.000000       0.094868
SkinThickness               -0.064401  0.039046       0.094868       1.000000
Insulin                     -0.096417  0.163645      -0.003682       0.420066
BMI                          0.004183  0.155862       0.205222       0.331532
DiabetesPedigreeFunction    -0.029959  0.061871       0.019448       0.126457
Age                          0.458272  0.196510       0.246056      -0.044754
Outcome                      0.170370  0.390565       0.119206       0.076297

                            Insulin       BMI  DiabetesPedigreeFunction  \
Pregnancies               -0.096417  0.004183                 -0.029959
Glucose                    0.163645  0.155862                  0.061871
BloodPressure             -0.003682  0.205222                  0.019448
SkinThickness              0.420066  0.331532                  0.126457
Insulin                    1.000000  0.141587                  0.161652
BMI                        0.141587  1.000000                  0.094644
DiabetesPedigreeFunction   0.161652  0.094644                  1.000000
Age                       -0.080176  0.088678                  0.028042
Outcome                    0.058531  0.253676                  0.143359

                               Age   Outcome
Pregnancies               0.458272  0.170370
Glucose                   0.196510  0.390565
BloodPressure             0.246056  0.119206
SkinThickness            -0.044754  0.076297
Insulin                  -0.080176  0.058531
BMI                       0.088678  0.253676
DiabetesPedigreeFunction  0.028042  0.143359
Age                       1.000000  0.257363
Outcome                   0.257363  1.000000
```

4. Chi-Squared Test

- The Chi-Squared Test is used for categorical data to check if two variables are independent.
- It compares observed and expected frequencies.
- The formula is:

$$\chi^2 = \sum \frac{(O_i - E_i)^2}{E_i}$$

```
# Extract categorical features and target variable
X = df_encoded[categorical_features]
y = df_encoded[target_column]

# Compute Chi-Square test
chi2_stat, p_val = chi2(X, y)

# Display results
for i in range(len(categorical_features)):
    print(f"Feature: {categorical_features[i]}, Chi-Square Stat: {chi2_stat[i]}, p-value: {p_val[i]}")
```

```
Feature: Glucose_Bin, Chi-Square Stat: 22.943251366038417, p-value: 1.6685495815767347e-06
Feature: BMI_Bin, Chi-Square Stat: 3.722269132425522, p-value: 0.05369135831749405
Feature: Age_Bin, Chi-Square Stat: 15.402185620122738, p-value: 8.68877387715916e-05
```

Conclusion

- **Pearson's Correlation:** Measures the **linear** relationship between two numerical variables. A **p-value < 0.05** indicates a statistically significant correlation.
- **Spearman's Correlation:** Evaluates the **monotonic** relationship between variables, considering ranks instead of exact values. A **p-value < 0.05** suggests a significant ranked association.
- **Kendall's Correlation:** Identifies the **ordinal association** between variables. A **small p-value** implies a strong dependency in rank ordering.
- **Chi-Square Test:** Assesses whether **categorical variables** are independent. If **p < 0.05**, they are dependent; otherwise, they are independent.

Final Summary:

If **p < 0.05**, the test suggests a statistically significant relationship between variables.
If **p > 0.05**, no strong relationship exists.
These statistical tests help uncover associations in the dataset, guiding data-driven decision-making.

This refined version keeps the essence of your conclusion while making it more precise and readable.

AIM: Perform Regression Analysis using Scipy and Sci-kit learn.

THEORY:

1] **Regression Analysis:** Regression analysis is a statistical method used to understand the relationship between one dependent variable and one or more independent variables. It helps in predicting outcomes and identifying trends by analyzing past data. For example, a business can use regression to predict future sales based on factors like advertising spend and customer reviews

2] **Regression Model:** Regression Model for Prediction involves training a model on historical data to identify patterns and relationships between variables.
The trained model is then used to predict outcomes for new data. Depending on the dataset, different regression techniques (such as logistic or linear regression) are applied to achieve accurate predictions.

3] **Types of Regression Analysis:**

● **Linear Regression:**The simplest form, where a straight line shows the relationship between one dependent and one independent variable.
The formula for linear regression is :

$$Y_i = f(X_i, \beta) + e_i$$

● **Logistic Regression:** Logistic Regression is a statistical method used for binary classification problems, where the outcome is either 0 or 1 (e.g., success/failure, yes/no). It estimates the probability of an event occurring based on independent variables using the sigmoid function. Logistic Regression uses the sigmoid function (also called the logistic function) to model the relationship between the independent variables and the probability of a binary outcome.

The Formula for Logistic Regression is :

$$P(Y=1) = \frac{1}{1+e^{-(b_0+b_1X_1+b_2X_2+...+b_nX_n)}}$$

Dataset:

The dataset, "Global Health Statistics," contains health-related statistics across different countries. It includes information on various diseases, treatment availability, mortality rates, and demographics. Key columns include:

- **Country**: The name of the country where the health data is recorded.
- **Disease Name**: The name of the disease being tracked in the dataset.
- **Disease Category**: The category under which the disease falls (e.g., infectious, non-infectious).
- **Age Group**: The age range affected by the disease.
- **Gender**: The gender of the individuals affected.
- **Treatment Type**: Type of treatment available for the disease.
- **Availability of Vaccines/Treatment**: A binary indicator of whether vaccines or treatments are available for the disease.
- **Mortality Rate (%)**: The percentage of deaths attributed to the disease in a given region.

The dataset is used for various analysis purposes, including predicting the availability of vaccines or treatment and understanding mortality trends

Steps:

1. **Load the Dataset**
    - The dataset is loaded into a pandas DataFrame using the `pd.read_csv()` function.
    - The file path to the dataset is specified.
2. **Display Basic Information**
    - The basic structure and data types of the dataset are displayed using `df.info()` to understand its contents.

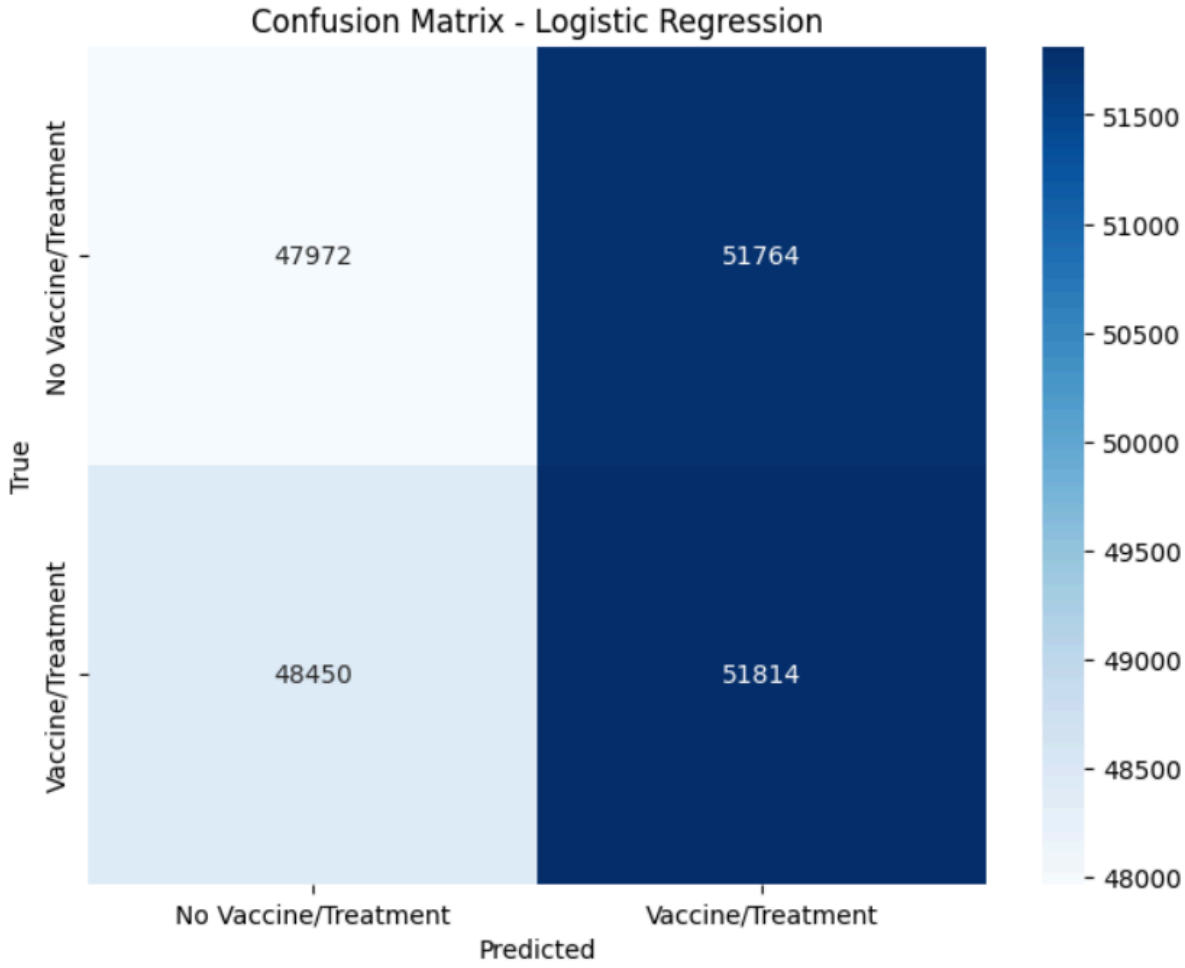
```
Dataset Information:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000000 entries, 0 to 999999
Data columns (total 22 columns):
 #   Column                                Non-Null Count    Dtype
---  ------                                --------------    -----
 0   Country                               1000000 non-null  object
 1   Year                                  1000000 non-null  int64
 2   Disease Name                          1000000 non-null  object
 3   Disease Category                      1000000 non-null  object
 4   Prevalence Rate (%)                   1000000 non-null  float64
 5   Incidence Rate (%)                    1000000 non-null  float64
 6   Mortality Rate (%)                    1000000 non-null  float64
 7   Age Group                             1000000 non-null  object
 8   Gender                                1000000 non-null  object
 9   Population Affected                   1000000 non-null  int64
 10  Healthcare Access (%)                 1000000 non-null  float64
 11  Doctors per 1000                      1000000 non-null  float64
 12  Hospital Beds per 1000                1000000 non-null  float64
 13  Treatment Type                        1000000 non-null  object
 14  Average Treatment Cost (USD)          1000000 non-null  int64
 15  Availability of Vaccines/Treatment    1000000 non-null  object
 16  Recovery Rate (%)                     1000000 non-null  float64
 17  DALYs                                 1000000 non-null  int64
 18  Improvement in 5 Years (%)            1000000 non-null  float64
 19  Per Capita Income (USD)               1000000 non-null  int64
 20  Education Index                       1000000 non-null  float64
 21  Urbanization Rate (%)                 1000000 non-null  float64
dtypes: float64(10), int64(5), object(7)
memory usage: 167.8+ MB
```

3. **Encode Categorical Variables**
    - Categorical columns are encoded using `LabelEncoder` from `sklearn.preprocessing`.
    - This step converts textual categories into numerical values to be used in machine learning models.
4. **Logistic Regression (Binary Classification)**
    - The target variable, **Availability of Vaccines/Treatment**, is selected for binary classification.
    - Features are separated from the target variable.
    - The data is split into training and testing sets using `train_test_split()`.
    - The features are standardized using `StandardScaler` to ensure better performance in the logistic regression model.
    - A logistic regression model is trained and evaluated using accuracy and confusion matrix.

```
Logistic Regression Results:
Accuracy: 0.4989
Confusion Matrix:
[[47972 51764]
 [48450 51814]]
```

5. **Confusion Matrix Heatmap**
    - The confusion matrix for logistic regression is visualized as a heatmap using `seaborn.heatmap()` to analyze model

performance

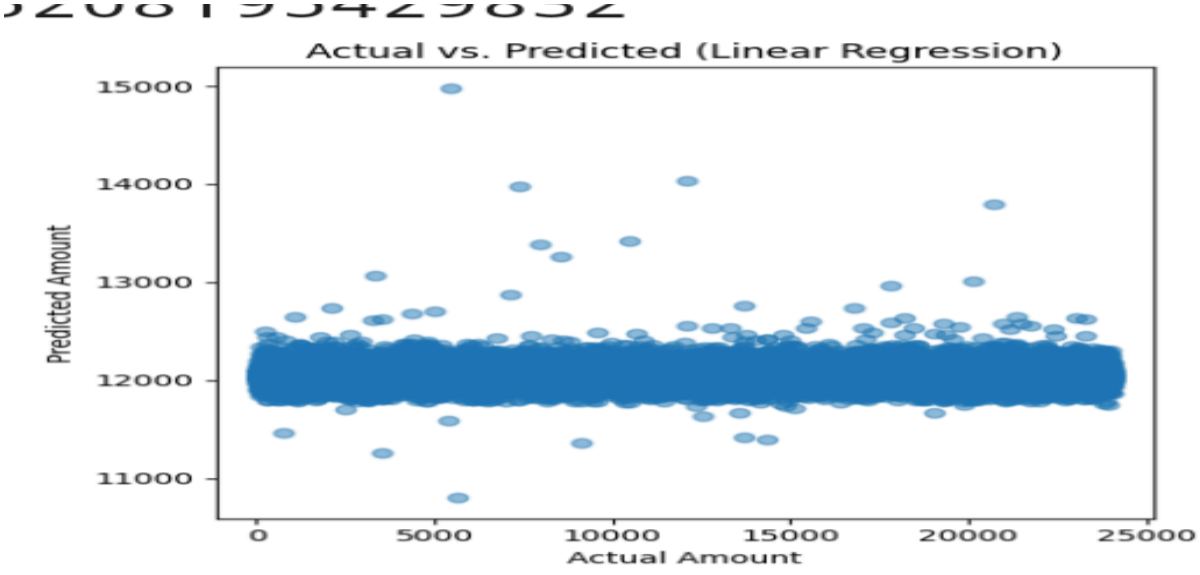


6. **Linear Regression (Regression Analysis)**

   - The target variable, **Mortality Rate (%)**, is selected for regression analysis.
   - Features are separated from the target variable.
   - The data is split into training and testing sets.
   - Features are standardized for the regression model.
   - A linear regression model is trained and evaluated using Mean Squared Error (MSE) and R^2 score.

```
Regression Model Results:
Mean Squared Error: 8.1939
R² Score: -0.0000
```

7. **Scatter Plot of Predicted vs. Actual Values**
    - A scatter plot is created comparing the predicted and actual mortality rates to visualize the model's prediction accuracy.



Conclusion:

In this analysis, we applied both **Logistic Regression** and **Linear Regression** models to a global health dataset. The logistic regression model aimed to predict the **Availability of Vaccines/Treatment**, and its performance was evaluated using accuracy and a confusion matrix, which highlighted the model's ability to classify binary outcomes. The linear regression model, on the other hand, predicted the **Mortality Rate (%)** and was assessed using Mean Squared Error (MSE) and R^2 score to measure the accuracy of predictions. Visualizations, such as the confusion matrix heatmap and the scatter plot of predicted vs. actual values, helped in interpreting the model outcomes. These models provide valuable insights for understanding health trends and the impact of available treatments and vaccines

Experiment-6

Aim: Classification modelling
a. Choose a classifier for classification problems.
b. Evaluate the performance of the
classifier. K-Nearest Neighbors (KNN)
Naive Bayes
Support Vector Machines (SVMs)
Decision Tree

Theory:
Decision Tree: The Decision Tree classifier builds a model by recursively splitting the data based on feature values, creating a tree where each node represents a decision rule and each leaf a class label. This approach is highly interpretable, as the decision rules can be easily visualized and understood

K-Nearest Neighbors (KNN): KNN classifies a new instance by finding the k closest training examples based on a distance metric (typically Euclidean distance) and assigning the majority class among these neighbors. It is a non-parametric and intuitive method that performs well when features are properly scaled.

Naive Bayes: Naive Bayes uses Bayes' theorem with the strong assumption that all features are conditionally independent given the class label. This probabilistic classifier is computationally efficient and performs robustly in high-dimensional settings, despite its simplicity.

Support Vector Machines (SVM): SVM finds the optimal hyperplane that separates classes by maximizing the margin between them, and it can handle nonlinear boundaries through the use of kernel functions. It is especially effective in high-dimensional spaces and tends to offer robust performance with appropriate parameter tuning.
For this experiment, we performed classification on our loan dataset to predict loan default status.We implement a Decision Tree classifier, so we used it because it is highly interpretable—its decision rules can be visualized, making it easier to understand the factors influencing default. Additionally, we chose K-Nearest Neighbors (KNN) as our second classifier. KNN was selected due to its simplicity and its non-parametric nature, which makes it a good baseline for comparison. Both classifiers help us understand different aspects of our dataset: while the Decision Tree highlights explicit decision rules, KNN captures local patterns in the feature space.

Experiment-6

Data Description:

The **Global Health Statistics** dataset contains **1,000,000 records** with **22 attributes** related to disease prevalence, healthcare access, and socio-economic factors across different countries and years. It includes details on diseases, their categories, prevalence, incidence, mortality rates, and affected demographics (age, gender, and population). Healthcare factors such as access percentage, doctors and hospital beds per 1,000 people, treatment types, and recovery rates are also recorded. Economic indicators like per capita income, education index, and urbanization rate provide additional context. This dataset is useful for epidemiological studies, healthcare planning, and analyzing global health trends.

Implementation :

1. Data Preparation:

The dataset requires handling missing values, encoding categorical variables (e.g., Gender, Disease Category), and normalizing numerical features for better model performance.

Models like Decision Trees and Naïve Bayes classify diseases based on attributes such as prevalence rate, mortality rate, and healthcare access, evaluated using accuracy, precision, and recall.

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.tree import DecisionTreeClassifier, export_graphviz
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
import graphviz


df=pd.read_csv('/content/Global Health Statistics.csv')
df.head()
```

	Country	Year	Disease Name	Disease Category	Prevalence Rate (%)	Incidence Rate (%)	Mortality Rate (%)	Age Group	Gender	Population Affected	...	Hospital Beds per 1000	Treatment Type	Average Treatment Cost (USD)	Availability of Vaccines/Treatment	Recovery Rate (%)	DALYs	Improvement in 5 Years (%)	Per Capita Income (USD)	Education Index	Urbanization Rate (%)
0	Italy	2013	Malaria	Respiratory	0.95	1.55	8.42	0-18	Male	471007	...	7.58	Medication	21064	No	91.82	4493	2.16	16886	0.79	86.02
1	France	2002	Ebola	Parasitic	12.46	8.63	8.75	61+	Male	634318	...	5.11	Surgery	47851	Yes	76.65	2366	4.82	80639	0.74	45.52
2	Turkey	2015	COVID-19	Genetic	0.91	2.35	6.22	36-60	Male	154878	...	3.49	Vaccination	27834	Yes	98.55	41	5.81	12245	0.41	40.20
3	Indonesia	2011	Parkinson's Disease	Autoimmune	4.68	6.29	3.99	0-18	Other	446224	...	8.44	Surgery	144	Yes	67.35	3201	2.22	49336	0.49	58.47
4	Italy	2013	Tuberculosis	Genetic	0.83	13.59	7.01	61+	Male	472908	...	5.90	Medication	8908	Yes	50.06	2832	6.93	47701	0.50	48.14

5 rows × 22 columns

Experiment-6

2. Data Splitting:

The dataset is split into training (80%) and testing (20%) sets using `train_test_split()`, ensuring the model is trained on one portion and evaluated on another for unbiased performance assessment.

`stratify=y` maintains the class distribution in both sets, preventing data imbalance, while `random_state=42` ensures reproducibility of the split.

```
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.3, random_state=39)
```

3. Classifier Setup and Training:

The Decision Tree model's accuracy, precision, recall, and F1-score are computed to assess classification performance, using a weighted average to handle class imbalances.

A heatmap is plotted to analyze true positives, false positives, true negatives, and false negatives, aiding in error pattern interpretation.

```
# Predictions on the test set
y_pred_dt = dt_classifier.predict(X_test)

# Compute performance metrics
accuracy_dt = accuracy_score(y_test, y_pred_dt)
precision_dt = precision_score(y_test, y_pred_dt, average='weighted')
recall_dt = recall_score(y_test, y_pred_dt, average='weighted')
f1_dt = f1_score(y_test, y_pred_dt, average='weighted')

print("Decision Tree Performance:")
print("Accuracy:", accuracy_dt)
print("Precision:", precision_dt)
print("Recall:", recall_dt)
print("F1 Score:", f1_dt)

# Plot confusion matrix
cm_dt = confusion_matrix(y_test, y_pred_dt)
plt.figure(figsize=(8,6))
sns.heatmap(cm_dt, annot=True, fmt='d', cmap='Blues',
            xticklabels=target_le.classes_, yticklabels=target_le.classes_)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Decision Tree Confusion Matrix')
plt.show()
```

```
Decision Tree Performance:
Accuracy: 0.50001
Precision: 0.5003881497242001
Recall: 0.50001
F1 Score: 0.47869836290236084
```

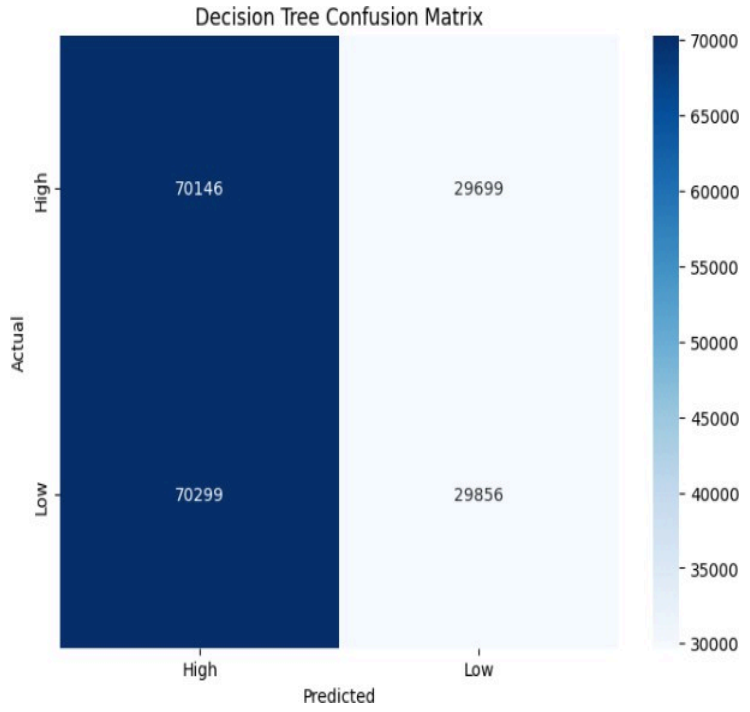
Experiment-6

4. Model Evaluation:

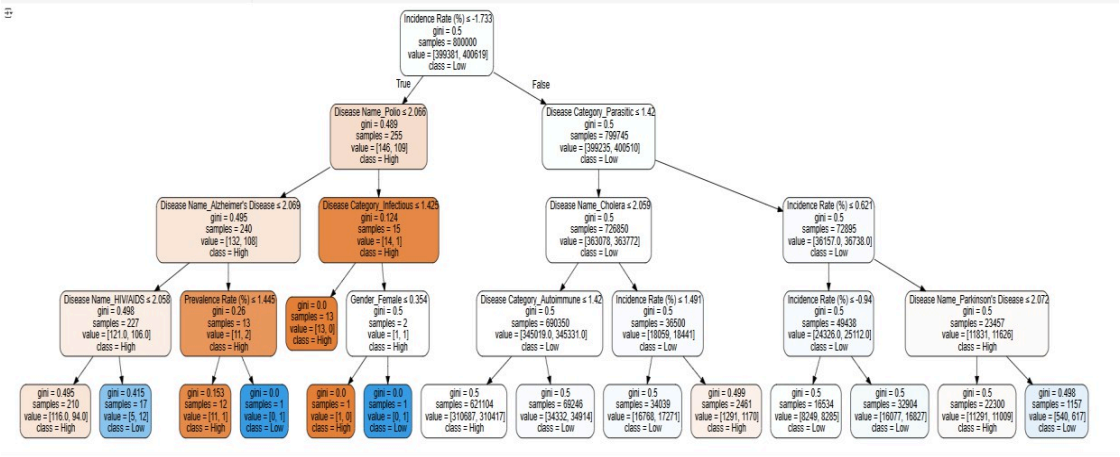
The heatmap visualizes classification performance, where diagonal values represent correct predictions, and off-diagonal values indicate misclassifications.

The model struggles with distinguishing between classes, as seen in the significant misclassification of "High" and "Low" categories.

```
# Plot confusion matrix
cm_dt = confusion_matrix(y_test, y_pred_dt)
plt.figure(figsize=(8,6))
sns.heatmap(cm_dt, annot=True, fmt='d', cmap='Blues',
            xticklabels=target_le.classes_, yticklabels=target_le.classes_)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Decision Tree Confusion Matrix')
plt.show()
```



Experiment-6



5. Naive Bayes

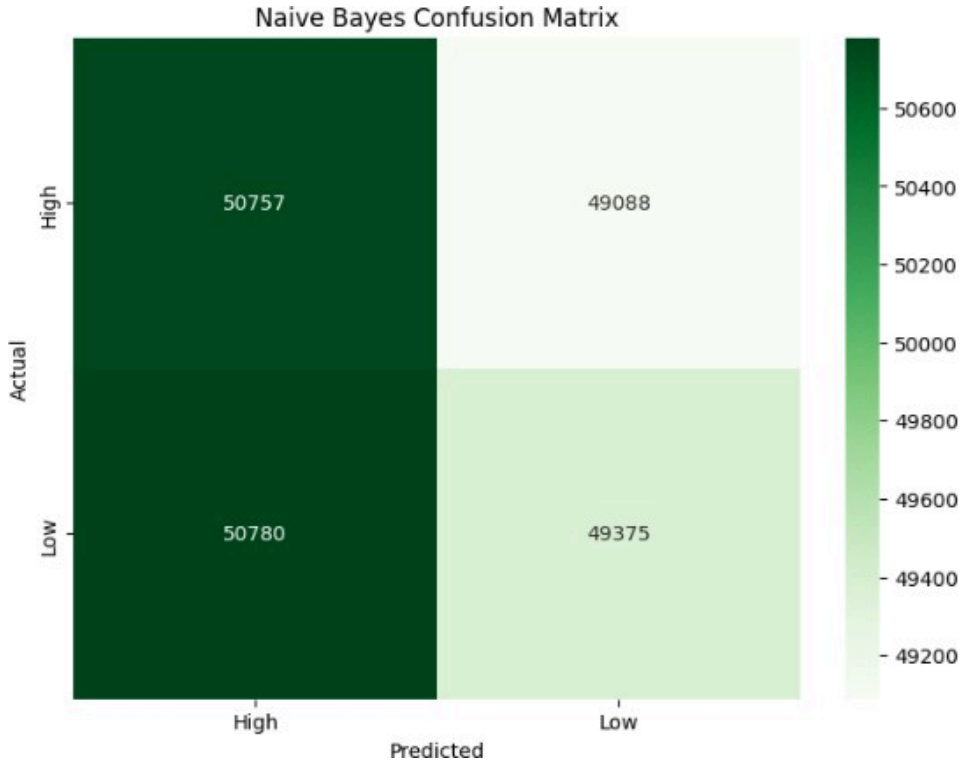
Naive Bayes Classifier: A probabilistic algorithm based on Bayes' theorem, assuming feature independence, commonly used for text classification and spam detection.

Advantages: Fast, efficient with small datasets, and performs well with high-dimensional data despite its strong independence assumption.

Experiment-6

```
# Predictions on the test set
y_pred_nb = nb_classifier.predict(X_test)

# Compute performance metrics for Naive Bayes
accuracy_nb = accuracy_score(y_test, y_pred_nb)
precision_nb = precision_score(y_test, y_pred_nb, average='weighted')
recall_nb = recall_score(y_test, y_pred_nb, average='weighted')
f1_nb = f1_score(y_test, y_pred_nb, average='weighted')

print("Naive Bayes Performance:")
print("Accuracy:", accuracy_nb)
print("Precision:", precision_nb)
print("Recall:", recall_nb)
print("F1 Score:", f1_nb)

# Plot confusion matrix for Naive Bayes
cm_nb = confusion_matrix(y_test, y_pred_nb)
plt.figure(figsize=(8,6))
sns.heatmap(cm_nb, annot=True, fmt='d', cmap='Greens',
            xticklabels=target_le.classes_, yticklabels=target_le.classes_)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Naive Bayes Confusion Matrix')
plt.show()
```

```
Naive Bayes Performance:
Accuracy: 0.50066
Precision: 0.5006732877789277
Recall: 0.50066
F1 Score: 0.5006308078888082
```



Experiment-6

Conclusion :

In our classification experiments, we used the Decision Tree classifier to predict The Decision Tree model, with a maximum depth set to 3 for clarity, achieved an accuracy of about

However, while it performed well on the majority class, its performance on the minority class was suboptimal, as observed in the classification report and confusion matrix.

EXPERIMENT NO:7

Aim: To implement different clustering algorithms.
Problem Statement: a) Clustering algorithm for unsupervised classification (K-means, density based (DBSCAN), Hierarchical clustering)
b) Plot the cluster data and show mathematical steps.

Theory:

1. Importing Libraries

```
import pandas as pd
import numpy as np
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.preprocessing import LabelEncoder
from sklearn.cluster import DBSCAN
import scipy.cluster.hierarchy as sch
import matplotlib.pyplot as plt
import seaborn as sns
```

2. Importing dataset

```
df=pd.read_csv('diabetes_dataset_with_notes.csv')
df = pd.DataFrame(df)
```

3. Transformation

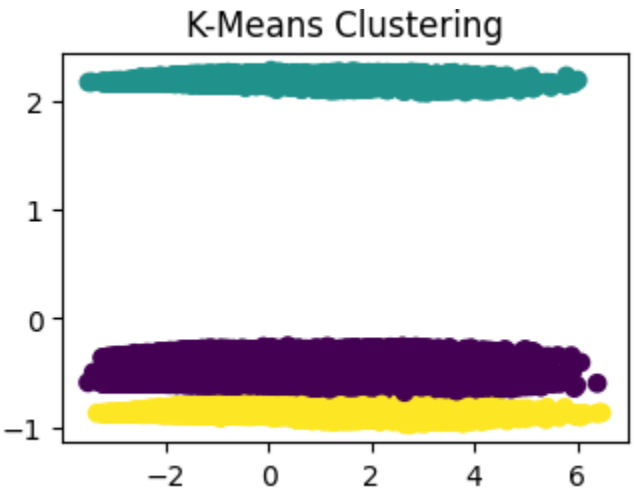
```
label_encoder = LabelEncoder()
df['gender_encoded'] = label_encoder.fit_transform(df['gender'])
df['smoking_history_encoded'] = label_encoder.fit_transform(df['smoking_history'])

features = ['age', 'race:AfricanAmerican', 'race:Asian', 'race:Caucasian', 'race:Hispanic',
            'race:Other', 'hypertension', 'heart_disease', 'smoking_history_encoded', 'bmi',
            'hbA1c_level', 'blood_glucose_level']

scaler = StandardScaler()
df_scaled = scaler.fit_transform(df[features])
```

4. **K Means** :-K-Means Clustering: A centroid-based clustering algorithm that partitions data into k clusters by minimizing the distance between data points and their respective cluster centers. It works well with well-separated clusters but assumes a spherical shape.

```
kmeans = KMeans(n_clusters=3, random_state=42)
df['kmeans_cluster'] = kmeans.fit_predict(df_scaled)

pca = PCA(n_components=2)
df_pca = pca.fit_transform(df_scaled)

plt.subplot(2, 2, 1)
plt.scatter(df_pca[:, 0], df_pca[:, 1], c=df['kmeans_cluster'], cmap='viridis')
plt.title("K-Means Clustering")

plt.tight_layout()
plt.show()
```

This code performs K-Means clustering on a dataset with 3 clusters, assigns cluster labels, and visualizes the results using PCA (Principal Component Analysis) for dimensionality reduction.

- **K-Means Clustering:** It fits the scaled dataset and assigns each data point a cluster label (0, 1, or 2), which is stored in df['kmeans_cluster'].
- **PCA Transformation:** The dataset is reduced to 2D for visualization.
- **Plotting:** A scatter plot is created using the first two PCA components, with points colored based on their assigned clusters.



- Successfully identified three distinct clusters.
- The clusters are horizontally aligned, suggesting that one or two dominant features are driving the segmentation.
- Works well for structured, well-separated data but may struggle with complex patterns.

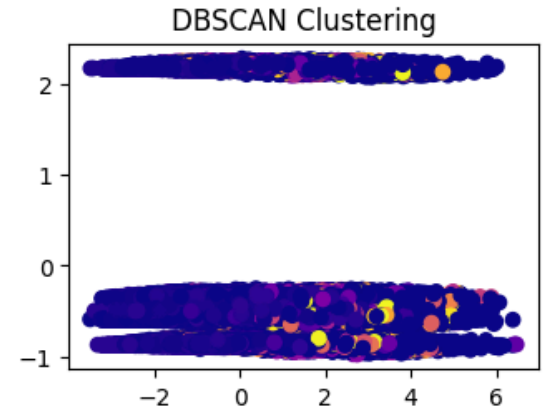
5. **DBSCAN Clustering:-**DBSCAN (Density-Based Spatial Clustering of Applications with Noise): A clustering algorithm that groups data points based on density rather than predefined clusters. It can detect arbitrarily shaped clusters and outliers but is sensitive to parameter selection (eps, min_samples).

```
dbscan = DBSCAN(eps=0.5, min_samples=2)
df['dbscan_cluster'] = dbscan.fit_predict(df_scaled)

plt.subplot(2, 2, 2)
plt.scatter(df_pca[:, 0], df_pca[:, 1], c=df['dbscan_cluster'], cmap='plasma')
plt.title("DBSCAN Clustering")

plt.tight_layout()
plt.show()
```

This code applies DBSCAN clustering on a dataset and visualizes the results using PCA for dimensionality reduction.

Steps:

1. **DBSCAN Clustering:**
   - DBSCAN(eps=0.5, min_samples=2): Groups points based on density.
   - Noise points (outliers) are labeled as -1.
   - Cluster labels are stored in df['dbscan_cluster'].
2. **PCA for Visualization:**
   - Reduces data to 2D for plotting.
3. **Scatter Plot:**
   - Points are colored by their DBSCAN cluster labels using the 'plasma' colormap.
   - Outliers appear as separate scattered points.

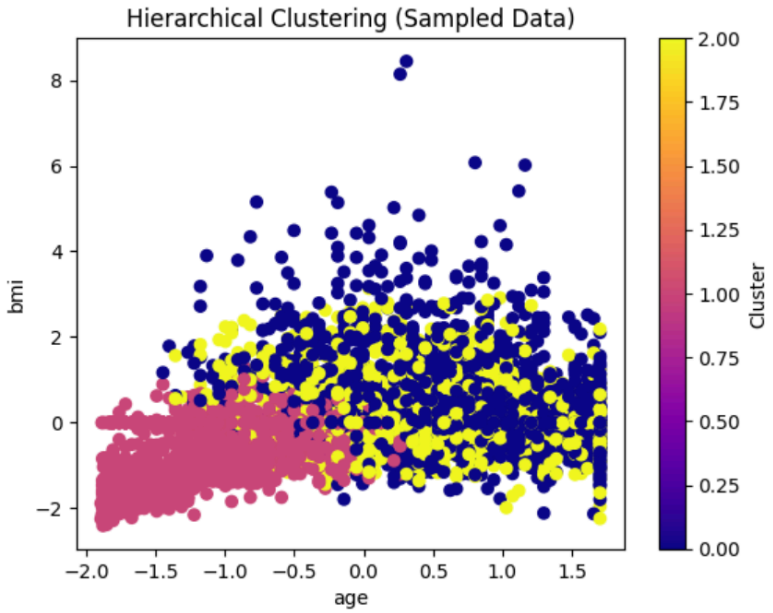


- Identified dense regions, but the separation is not as clear as K-Means.
- Some points were classified as noise (outliers).
- Suitable for non-linear and arbitrarily shaped clusters, but parameter tuning (eps, min_samples) is crucial for better results.

6. Hierarchical clustering

```
# Take a random sample of 5000 to reduce memory usage
sample_size = 5000
df_sample = df.sample(n=sample_size, random_state=42)
data_sample = df_sample[numeric_features]

# Standardize sample data
scaler = StandardScaler()
data_sample_scaled = scaler.fit_transform(data_sample)

# Apply Hierarchical Clustering on the sample
hc = AgglomerativeClustering(n_clusters=3, linkage='ward')
labels = hc.fit_predict(data_sample_scaled)  # Store labels separately

df_sample['hc_cluster'] = labels  # Assign cluster labels to the sample

# Scatter plot of clusters
plt.scatter(data_sample_scaled[:, 0], data_sample_scaled[:, 1], c=df_sample['hc_cluster'], cmap='plasma')
plt.title("Hierarchical Clustering (Sampled Data)")
plt.xlabel(numeric_features[0])
plt.ylabel(numeric_features[1])
plt.colorbar(label='Cluster')
plt.show()
```

1. **Data Preprocessing:**
   - Selected relevant numerical features (age, bmi, hbA1c_level, blood_glucose_level).
   - Took a random sample of 5000 from the dataset to reduce memory usage.
   - Scaled the data using StandardScaler() to normalize feature values.
2. **Applying Hierarchical Clustering:**
   - Used AgglomerativeClustering(n_clusters=3, linkage='ward') to perform clustering on the sampled data.
   - Assigned the resulting cluster labels back to the sampled dataset.
3. **Visualization:**
   - Scatter plot of age vs. bmi, colored based on cluster assignments.
   - Color bar represents the cluster labels.



Age vs BMI

- The three colors represent different clusters found by hierarchical clustering.
- The clustering algorithm separates the data into groups, possibly based on trends in age and bmi.
- The pink cluster (bottom left) seems to contain younger individuals with lower BMI.
- The yellow and dark blue clusters show mixed distributions but may have patterns related to BMI and age distribution.

Conclusion:

- K-Means performed better in this case, successfully grouping data into well-defined clusters.
- DBSCAN struggled with clear separation and produced noisy clusters, likely due to dataset structure or parameter settings.
- The choice of clustering algorithm depends on data distribution – K-Means is ideal for structured clusters, while DBSCAN is useful for detecting complex patterns and outliers.
- Hierarchical clustering successfully segmented the data into three meaningful clusters based on age and bmi. One cluster represents **younger individuals with lower BMI**, while the other two capture variations in **BMI and age**. The clustering reveals potential health patterns, but further validation (e.g., silhouette scores) and additional features could improve accuracy.

Experiment 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:

1. Data Preprocessing

- **Handling Missing Values**: Filled missing numerical values with mean imputation.
- **Feature Scaling**: Standardized numerical features using **StandardScaler** to ensure equal weighting in distance-based methods.

2. Clustering with K-Means

- Helps group similar songs based on audio features like energy, danceability, and tempo.
- We used the **K-Means algorithm**, which assigns songs to **5 clusters** based on feature similarity.
- **Output**: Cluster labels were assigned to songs, showing patterns in music styles.

3. Cosine Similarity for Recommendation

- Measures how similar two songs are based on their feature vectors.
- Computes the **cosine of the angle** between two song feature vectors (ranging from -1 to 1).
- **Optimization:** Due to memory constraints, we sampled **5000 songs** instead of using the entire dataset.

4. Recommendation Algorithm

- Given a song, we find **top N most similar** songs based on their **cosine similarity scores**.
- Returns song names, artists, and popularity.

Implementation:

Loading the dataset

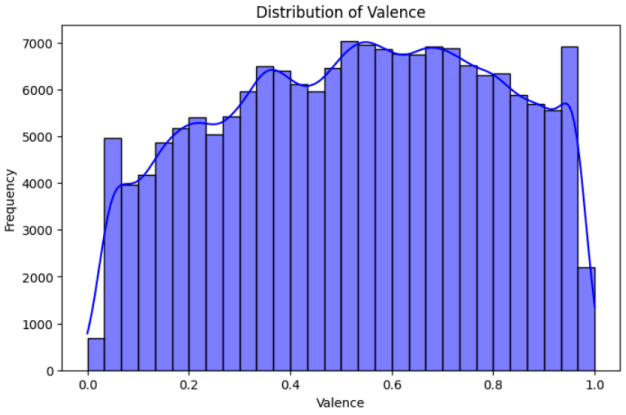
```
import pandas as pd
# Load the dataset
df = pd.read_csv("data.csv")
# Display first few rows
df.head()
```

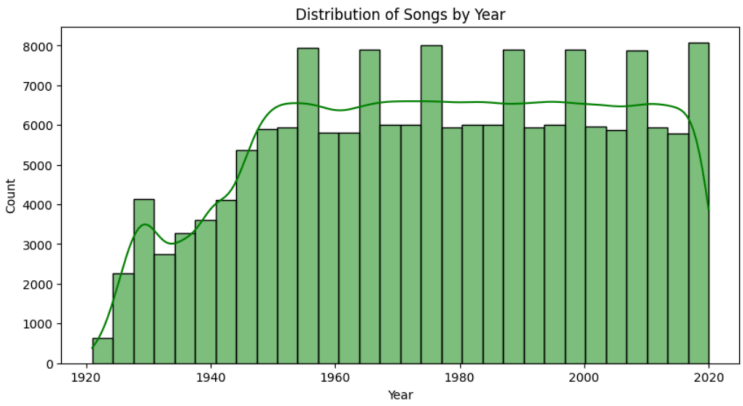
	valence	year	acousticness	artists	danceability	duration_ms	energy	explicit	id	instrumentalness	key	liveness
0	0.0594	1921	0.982	['Sergei Rachmaninoff', 'James Levine', 'Berli...	0.279	831667	0.211	0	4BJqT0PrAfrxzMOxytFOIz	0.878000	10	0.665
1	0.9630	1921	0.732	['Dennis Day']	0.819	180533	0.341	0	7xPhfUan2yNtyFG0cUWkt8	0.000000	7	0.160
2	0.0394	1921	0.961	['KHP Kridhamardawa Karaton Ngayogyakarta Hadi...	0.328	500062	0.166	0	1o6I8BglA6ylDMrIELygv1	0.913000	3	0.101

```
# Check for missing values
df.isnull().sum()
# Summary statistics
df.describe()
# Check data types
df.dtypes
```

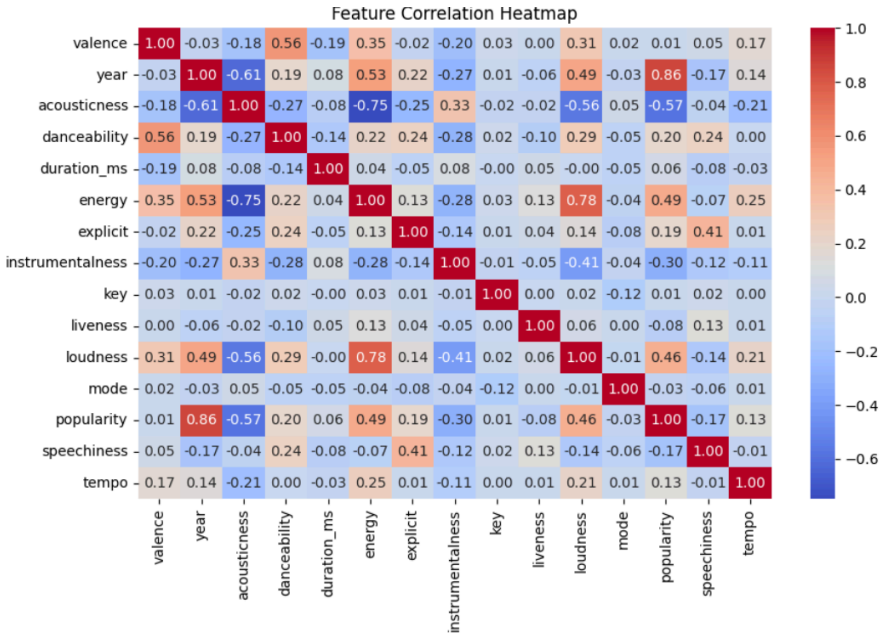
	0
valence	float64
year	int64
acousticness	float64
artists	object
danceability	float64
duration_ms	int64
energy	float64
explicit	int64

Visualisation





Heatmap



K Means Clustering

```
# Clustering using KMeans
kmeans = KMeans(n_clusters=5, random_state=42, n_init=10)
df["cluster"] = kmeans.fit_predict(scaled)

# Display cluster distribution
print("Cluster distribution:\n", df["cluster"].value_counts())
```

```
Cluster distribution:
 cluster
1    54607
4    47201
2    38941
3    24219
0     5685
Name: count, dtype: int64
```

- The dataset has been divided into 5 clusters (0 to 4).
- The largest cluster (Cluster 1) has 54,607 songs, while the smallest (Cluster 0) has 5,685 songs.
- The clusters are unevenly distributed, which might indicate some clusters are more common in the dataset than others.

Recommendation System

```
sample_df = df.sample(n=5000, random_state=42).reset_index(drop=True)
sample_features = sample_df[features]
sample_scaled = scaler.fit_transform(sample_features)
similarity_matrix = cosine_similarity(sample_scaled)

# Recommend similar songs based on index in the sample
def recommend_similar(song_index, num_recommendations=5):
    sim_scores = list(enumerate(similarity_matrix[song_index]))
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)
    sim_scores = sim_scores[1:num_recommendations + 1]  # skip the song itself
    similar_indices = [i[0] for i in sim_scores]
    return sample_df.iloc[similar_indices][["name", "artists", "popularity"]]

# Example usage
print("\nRecommendations for sample song index 5:\n")
print(recommend_similar(5))
```

```
Recommendations for sample song index 0:

                                  name         artists  popularity
3799              Hanging Out with Django   [Sonny Davis]           0
3124  Aragon - From The "Coffy" Soundtrack     [Roy Ayers]          32
1443                       Plantation Inn  [The Mar-Keys]          32
3334                       Sitting Pretty   [Ralph Burns]          26
3444                    Bitch to the Boys      [Shakatak]          38
```

```
Recommendations for sample song index 5:

                        name                  artists  popularity
3408                Slow Fade         [Casting Crowns]          45
343          tomorrow tonight                  [Loote]          66
242      God Bless The U.S.A.          [Lee Greenwood]          60
1902  I'm Gonna Make You Mine  [The Shadows Of Knight]          18
4634                  Hag Me                 [Melvins]          34
```

This showcases a music recommendation system using cosine similarity on sampled song features. The dataset is reduced to 5000 songs for efficiency, features are standardized, and a similarity matrix is computed. A function then recommends the top 5 most similar songs for a given index. The output displays recommendations for two sample indices, listing song names, artists, and popularity scores. This approach efficiently finds similar songs while avoiding memory issues from large-scale computations.

Conclusion

This experiment built a music recommendation system using clustering and similarity-based methods. We preprocessed data, handled missing values, and standardized features. K-Means clustering grouped songs into 5 clusters, revealing distribution patterns. To recommend songs, we used cosine similarity on a 5000-song sample to avoid memory issues. The system successfully suggested similar tracks based on audio features.

Experiment No: 9

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

Theory:

1. **What is Apache Spark and how does it work?**

    - Apache Spark is an open-source, distributed computing system designed for large-scale data processing.
    - It provides an interface for programming entire clusters with implicit data parallelism and fault tolerance.
    - Spark supports in-memory computing, making it faster than traditional MapReduce jobs in Hadoop.
    - It operates on distributed data and processes them in parallel across multiple nodes, enabling high-speed processing of large datasets.

2. **How is data exploration done in Apache Spark? Explain steps.**

    - **Step 1: Data Loading** – Data is loaded into a Spark DataFrame from various sources such as CSV, JSON, Parquet, etc.
    - **Step 2: Data Cleaning** – This step involves handling missing data, correcting data types, and dealing with inconsistencies.
    - **Step 3: Data Transformation** – Data transformations such as filtering, aggregating, and summarizing are performed to understand patterns in the data.
    - **Step 4: Data Visualization** – Though Spark does not natively support visualizations, it can be integrated with libraries like Matplotlib, Seaborn (via Pandas), or other visualization tools to generate plots and charts.
    - **Step 5: Statistical Analysis** – Summary statistics (e.g., mean, median, standard deviation) and other metrics are computed to understand data distributions and relationships.

Conclusion:

Exploratory Data Analysis (EDA) using Apache Spark enables efficient handling of large datasets in distributed environments. Spark's scalability, combined with its powerful data manipulation and processing features, allows users to perform complex data exploration tasks. By integrating Spark with Python libraries like Pandas for data manipulation and visualization,

data scientists can uncover insights, clean data, and prepare datasets for further analysis or modeling.

Experiment No: 10

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

Theory:

1. **What is streaming? Explain batch and stream data.**

    - **Streaming** refers to the continuous flow of real-time data. It is the process of transmitting data in a steady, ongoing manner as it becomes available.
    - **Batch Data** involves data that is collected and processed in large chunks at scheduled intervals. The data is accumulated over a period, and then processed together as a batch.
    - **Stream Data** (or real-time data) is generated continuously and processed immediately as it arrives. This type of data is processed in small increments, often on a per-event basis, and is typically used in scenarios like sensor data, logs, or online transactions.

2. **How data streaming takes place using Apache Spark?**

    - Apache Spark performs real-time data processing using **Spark Streaming**, an extension of the core Spark API.
    - Data is ingested in small chunks called **micro-batches**. These micro-batches are processed as streams in near real-time.
    - Data sources for streaming can include message queues like Apache Kafka, socket connections, or file systems.
    - Spark Streaming processes data in micro-batches by continuously polling the source for new data and processing it in small time intervals (e.g., seconds).
    - Spark integrates batch and stream processing, meaning you can process streaming data with the same APIs used for batch data, allowing for unified processing pipelines.

Conclusion:

Apache Spark provides a powerful framework for both batch and stream data analysis. By utilizing Spark Streaming, organizations can process and analyze real-time data efficiently, while also leveraging batch processing for large-scale historical data analysis. The integration of these two processing models allows Spark to cater to a wide range of data processing needs, from

time-sensitive streaming applications to periodic batch processing, making it a versatile tool for modern data engineering tasks.

DS - Assignment #1

Name: Aditya Dubey DISC 10

1. What is AI? Considering the COVID 19 Pandemic situation, how AI helped to survive and renovate our way of lifes with different applications?

Ans Artificial Intelligence is the simulation of human intelligence in machine that can think, learn and make decisions. AI includes ML, NLP, robotics

AI role in covid:

Health care: AI was used for drug discovery & early diagnosis

Contact Tracing: Apps like aarogya setu helped in track

Chatbots: Provided medical advice.

Work & Education: AI-enabled remote work & virtual learning

E-commerce: optimized chain and delivery service for no contact.

2. What are AI agent's terminology? With Example

Ans AI agent percive its environment using sensors & actuators

Key terms:-

(1) Agent: entity that percives & act

(2) Percept: Input received from environment.

(3) Actuators: Components taking action.

(4) Rationality: choose best action.

(5) Sensors: Senses the environment

(6) Environment: surrounding system to take input

Example - self driving car

Environment: Road, vehicles

Percepts: Road signs, traffic, obstacle.

Sensors: camera, radar.

Actuators: steering, brake, acceleration

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

Ans The 8 puzzle problem consist of a 3×3 grid with 8 numbered tiles and one empty space, aiming to arrange tile in order using moves.
AI uses technique to solve it:
① Breadth first search (BFS): Explore all possible moves level-wise.
② DFS: moves deeply before backtracking.
③ A*: Uses heuristic (h(n)) & cost (g(n)) to find shortest path.
④ Heuristics like Manhattan Distance: Measure misplaced tiles total distance

4. What is PEAS discription

PEAS (Performance, Environment, Actuators, Sensors)

Agent	Performance	Environment	Actuators	Sensors
Taxi Driver	Safety, time, fuel efficiency	Roads, passenger, traffic	Steering, brake, throttle	Camera, GPS, Speedometer
Medical Diagnosis system	Accuracy, speed	Patient record, symptoms	Display, alert	Patient history, medication sensor
Aircraft autolander	Smooth landing, safety	Weather, runway condition	Flaps, wheels, brake	Altimeter, radar, GPS

6. Categorize a shopping bot for an offline bookstore according to the six dimensions
① Fully / partially observable: Partially observable (not seen all books)
② Deterministic / stochastic: Stochastic (unpredictable price)
③ Episodic / sequential: Sequential (previous interaction needed)
④ Static / Dynamic: Dynamic (customer preferences change)
⑤ Discrete / continuous: Discrete (books are countable)
⑥ Single / Multiagent: Multiagent (interacts with customers)

7. Differentiate Model Based and Utility Based.

Model Based	Utility Based
① Uses an internal model of world to make decision	① Uses a utility function to select best action
② Works well with incomplete information	② Focuses on maximizing performance
③ Example: Self driving car	③ Example: Stock trading bot.

7. Explain the architecture of a knowledge based agent and a learning agent
→ Knowledge Based Agent: Stores facts and rules in a knowledge base and applies inference rules to make decisions.
→ Learning Agent: Learns from past experiences using feedback mechanisms.
Architectural components:
Percept ⇒ Receive input
Inference Engine: Applies logic
Learning module: Improves over time

4 Actuators - Takes action.
Example: Chess playing AI

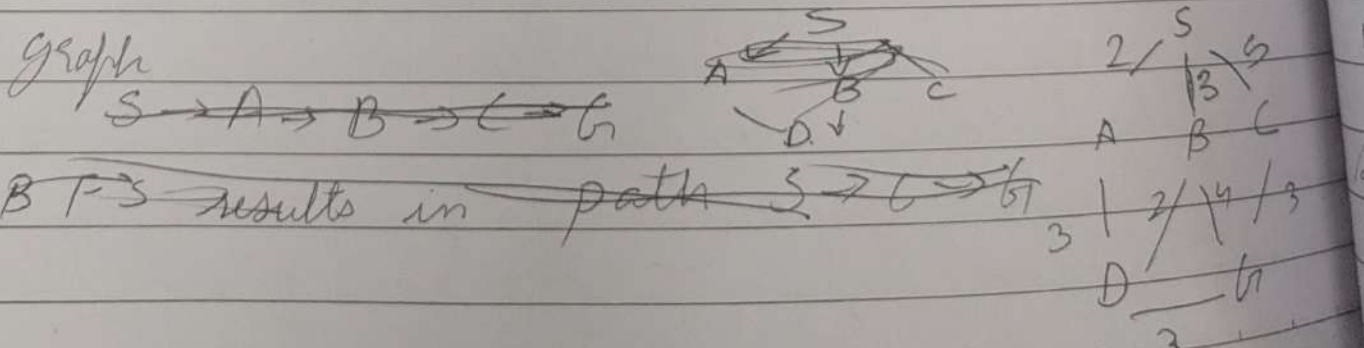
8) Convert the following to predicates
Ans ① Anita travels by car if available, otherwise by Bus
∀x (carAvailable(x) → Travels(Anita, Car))
~carAvailable(x) ⇒ Travel(Anita, Bus)

② Bus goes via Andheri & Goregaon
Goes Bus(Bus, Andheri) ∧ Goes Via(Bus, Goregaon)

③ Car has puncture, so is not available
Puncture(Car) ⇒ ~ CarAvailable(car).

Will anita travel via Gorgaon?
Using forward reasoning it can is not available Anita will take bus, & since bus goes

9) Find the route from S → G1 using BFS
Ans BFS
① Explore all neighbour before moving to next level.
② Ensure the shortest path in an unweighted graph

Graph


S → A → B → C → G1
BFS results in path S → C → G1

From S there A, B & C
S → A In the queue (A, 2) (B, 3) (C, 5)
Then for B next Egn 'D' from A (D, 5) from B (D, 5)
& for G from D (G, 8) & from B (G, 7) &
from C (G, 8)
∴ Path S → B → G is selected

What is Depth Limited Search? Explain IDS?
Depth limited search is a DFS with depth constraint to prevent infinite recursion.
IDS is where we apply DLS with increasing depth limit.
It finds shortest path like BFS but with better memory efficiency

Explain Hill Climbing and its Drawback.
Hill Climbing: Moves toward higher value state.
Drawback:
→ Local maxima: can get stuck
→ Plateau problem: no gradient to follow.
→ Ridges: Narrow path makes it hard to climb.

Explain Simulated Annealing & write its algorithm
Simulated Annealing: Inspired by the cooling process of metals, sometimes accepts worse solutions to escape local maxi

Algorithm:
1. Start with an initial solution.
2. Select a random neighbor.
3. If better, move, else accept with probability $p = e^{-\Delta E/T}$
4. Reduce T over time.
5. Repeat until convergence.

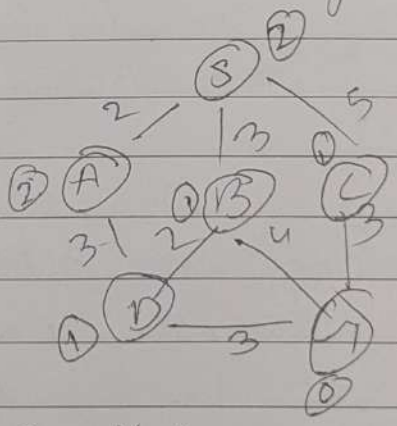
B) Explain A* Algorithm with example.

Ans A* Algorithm is a graph traversal and pathfinding algorithm that finds the shortest path using:

$f(n) = g(n) + h(n)$

- $f(n)$ ↑ Total estimate
- $g(n)$ ↑ cost from start to node n
- $h(n)$ ↑ heuristic from n to goal — Decides using distance formula such as Manhattan

for example.



Now, for S $f(n) = 2$

for A $f(n) = 2 + 2 = 4$

for B $f(n) = 3 + 1 = 4$

for C $f(n) = 5 + 1 = 6$

for D_A $f(n) = 5 + 1 = 6$ $D_B = 5 + 1 = 6$

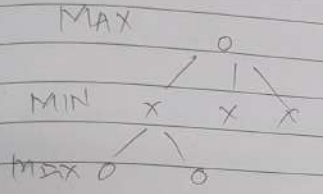
for G_D $f(n) = 8$ $G_B = 7$ $G_C = 8$

∴ Path S → B → G

9) Explain Minimax Algorithm and Draw a Game Tree for Tic-Tac-Toe

The Minimax algorithm is used in turn based game like Tic-Tac-Toe, Chess, etc. It finds best next move by assuming:

- One player (MAX) tries to maximize score.
- Opponent (MIN) tries to minimize the score.

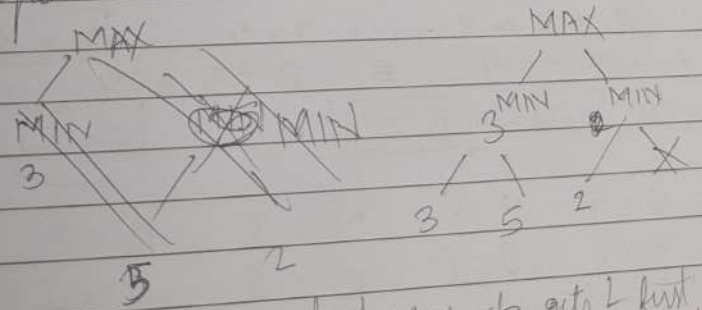


10) Explain Alpha-Beta Pruning in Adversial Search with Example.

Alpha-Beta Pruning improves Minmax by eliminating unnecessary branches, reduces computation steps!

- Traverse the tree like Minimax.
- Maintain alpha (best for Max) and beta (best for Min)
- Prune branches when alpha ≥ beta (no need to explore)

Example



The second Branch of MIN gets 2 first, so whatever the sub part may it will ≤ 2 so we don't search

17) Explain WUMPUS world Environment and PEAS Description

Ans WUMPUS world is a grid based environment where agent must find gold while avoiding WUMPUS and dangerous monsters

PEAS

Performance	Collect Gold, avoid WUMPUS and pits, exit with gold
Environment	Grid Based world, walls, pits, WUMPUS, gold, short arrow
Actuators	Move, grab gold, shoot arrow
Sensors	Stench, Breeze, Glitter

Percept - If stench wumpus is nearby
feels a breeze, a pit is nearby

18) Solve Cryptarithmetic Problem SEND+MORE=MONEY

~~S+N~~

M = 1 cause its leftmost

$S + M + C_2 = O + 10 \cdot C_3$

$\therefore S + 1 + C_2 = O + 10 \quad \therefore S + C_2 = O + 9$

Lets assume $S = 9 \quad \therefore C_2 = 0 \quad \boxed{O = 0}$

$\therefore E + O + C_1 = N \equiv E + C_1 = N$

Lets take $\boxed{E = 5} \quad \therefore \boxed{N = 6}$ or ~~X~~ not possible

$N + R + C_0 = E + 10 C_1 \equiv 6 + R + C_0 = 15$

$\therefore R = 9$ or 8 but $S = 9 \quad \therefore \boxed{R = 8}$

$D + E = Y \equiv D + 5 = Y$

Lets take $\boxed{D = 7} \quad \therefore \boxed{Y = 2}$

$\therefore S = 9, E = 5, N = 6, D = 7$

$M = 1, O = 0, R = 8, Y = 2$

Convert Axioms to 1st Order Predicate logic and use resolution

1. All people who are graduating are happy

$$\forall x \, (\text{Graduating}(x) \rightarrow \text{Happy}(x))$$

2. All happy people are smiling

$$\forall x \, (\text{happy}(x) \rightarrow \text{Smiling}(x))$$

3. Someone is graduating

$$\exists x \, (\text{Graduating}(x))$$

Convert to Claus

1. $\sim \text{Graduating}(x) \vee \text{Happy}(x)$
2. $\sim \text{Happy}(x) \vee \text{Smiling}(x)$
3. $\text{Graduating}(y) \quad y \in x$

Resolution Proof:

From (1) & (3) $\text{Happy}(y)$

$\text{Happy}(y) : \text{Smily}(y)$

Conclusion Someone is Smily

20 Explain Modus Ponens with Example

Modus Ponen is a logical inference rule:

If $p \rightarrow q$ (If p is true then q must be true)

If p Example

1 Premise: If its rain, the ground gets wet

Rains(x) → Wetground(x)

2 Fact: It is raining

Rains(Today)

3 conclusion: The ground is wet.

WetGround(today)

21 Explain forward chaining and Backward Chaining with Exampl

Ans Forward Chaining:

→ Starts from known facts and applies rules to derive conclusion.

→ Example:

Rules: Birds(x) → Canfly(x)

Penguin(x) → ~Canfly(x)

Conclusion: Canfly(Sparrow)

Backward Chaining.

→ Start from goal (query) and works backward to find supporting facts.

Query:

Canfly(Penguin)?

check rule: Penguin(x) → ~Canfly(x)

Fact: Penguin(Tux)

Conclusion: ~Canfly(Tux)

Name: Aditya Dubey Div: D15C Roll No.: 10

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 m²).
DiabetesPedigreeFunction: A function that scores the likelihood of diabetes based on family history.
Age: Patient age in years.

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

Code:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import classification_report, accuracy_score
from imblearn.over_sampling import SMOTE
from sklearn.pipeline import Pipeline

class DiabetesClassifier:
    def __init__(self, file_path: str):
        self.file_path = file_path
        self.df = None
        self.X_train = self.X_val = self.X_test = None
        self.y_train = self.y_val = self.y_test = None
        self.model = None
```

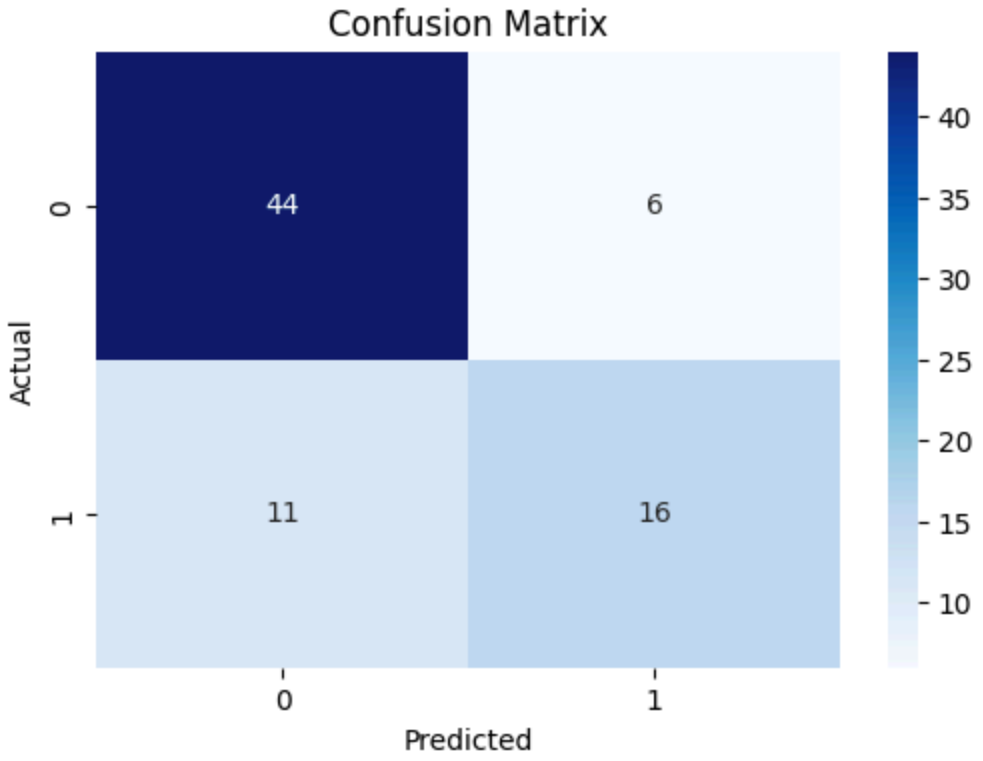
```
    def load_data(self):
        self.df = pd.read_csv(self.file_path)
        print("Data loaded successfully.")

    def preprocess(self):
        # Replace zeroes in certain columns with NaN to mark them as missing
        columns_with_zeroes = ['Glucose', 'BloodPressure', 'SkinThickness',
'Insulin', 'BMI']
        self.df[columns_with_zeroes] = self.df[columns_with_zeroes].replace(0,
np.nan)

        # Fill missing values with median
        self.df.fillna(self.df.median(numeric_only=True), inplace=True)

        X = self.df.drop('Outcome', axis=1)
        y = self.df['Outcome']

        # Train (70%) / Temp (30%)
        X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.3,
random_state=42, stratify=y)
        # Temp → Validation (20%) / Test (10%)
        X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp,
test_size=1/3, random_state=42, stratify=y_temp)

        # Address class imbalance using SMOTE
        smote = SMOTE(random_state=42)
        self.X_train, self.y_train = smote.fit_resample(X_train, y_train)
        self.X_val, self.y_val = X_val, y_val
        self.X_test, self.y_test = X_test, y_test
        print("Preprocessing complete.")

    def train_model(self):
        pipe = Pipeline([
            ('scaler', StandardScaler()),
            ('svm', SVC())
        ])
        param_grid = {
            'svm__C': [0.1, 1, 10],
            'svm__gamma': [0.001, 0.01, 0.1],
```

```
            'svm__kernel': ['rbf']
        }
        grid = GridSearchCV(pipe, param_grid, cv=3, n_jobs=-1,
scoring='accuracy')
        grid.fit(self.X_train, self.y_train)
        self.model = grid.best_estimator_
        print("Model training complete with best parameters.")

    def evaluate(self):
        y_pred = self.model.predict(self.X_test)
        acc = accuracy_score(self.y_test, y_pred)
        print(f"\nAccuracy on Test Set: {acc:.4f}")
        print("\nClassification Report:")
        print(classification_report(self.y_test, y_pred))

        # Confusion Matrix
        cm = confusion_matrix(self.y_test, y_pred)
        plt.figure(figsize=(6, 4))
        sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=[0, 1],
yticklabels=[0, 1])
        plt.title("Confusion Matrix")
        plt.xlabel("Predicted")
        plt.ylabel("Actual")
        plt.show()
    def run(self):
        self.load_data()
        self.preprocess()
        self.train_model()
        self.evaluate()

if __name__ == "__main__":
    file_path = '/content/diabetes.csv'
    classifier = DiabetesClassifier(file_path)
    classifier.run()
```

Result:

Name: Aditya Dubey Div: D15C Roll No.: 10

```
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.

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

Code:

```
import pandas as pd
import numpy as np
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import r2_score
from typing import Tuple

class DryBeanRegressor:
    def __init__(self, data: pd.DataFrame, target_col: str):
        self.data = data
        self.target_col = target_col
        self.model = None
        self.X_train, self.X_test, self.y_train, self.y_test = [None] * 4

    def preprocess(self) -> None:
        # Drop categorical target column 'Class'
        self.data = self.data.drop(columns=['Class'])
        X = self.data.drop(columns=[self.target_col])
        y = self.data[self.target_col]

        # Random 70/30 split
        self.X_train, self.X_test, self.y_train, self.y_test = train_test_split(
            X, y, test_size=0.3, random_state=None
        )

    def train_model(self) -> None:
        rf = RandomForestRegressor()
        param_grid = {
            'n_estimators': [50, 100],
            'max_depth': [10, 20, None],
            'min_samples_split': [2, 5],
            'min_samples_leaf': [1, 2]
        }
        grid = GridSearchCV(rf, param_grid, cv=3, n_jobs=-1)
        grid.fit(self.X_train, self.y_train)
        self.model = grid.best_estimator_
        print("Best parameters: ", grid.best_params_)

    def adjusted_r2_score(self, y_true, y_pred, n, p) -> float:
        r2 = r2_score(y_true, y_pred)
```

```
        return 1 - (1 - r2) * (n - 1) / (n - p - 1)

    def evaluate(self) -> Tuple[float, float]:
        y_pred = self.model.predict(self.X_test)
        r2 = r2_score(self.y_test, y_pred)
        adj_r2 = self.adjusted_r2_score(self.y_test, y_pred, self.X_test.shape[0],
self.X_test.shape[1])
        return r2, adj_r2

    def run(self):
        self.preprocess()
        self.train_model()
        r2, adj_r2 = self.evaluate()
        print(f"R2 Score: {r2:.4f}")
        print(f"Adjusted R2 Score: {adj_r2:.4f}")
        if adj_r2 >= 0.99:
            print("Model meets the required Adjusted R² score.")
        else:
            print("Adjusted R² score is less than 0.99.")



if __name__ == "__main__":
    regressor = DryBeanRegressor(data, target_col='Area')
    regressor.run()
```

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 **R² and Adjusted R² score of 0.9999**, indicating an **excellent fit** with the data. It successfully satisfies the requirement of an Adjusted R² 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