

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

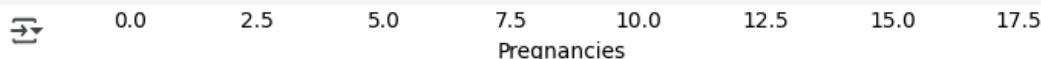
```
▶ categorical_columns = df.select_dtypes(include=['object']).columns  
df = pd.get_dummies(df, columns=categorical_columns, drop_first=True)
```

7. Find out outliers.

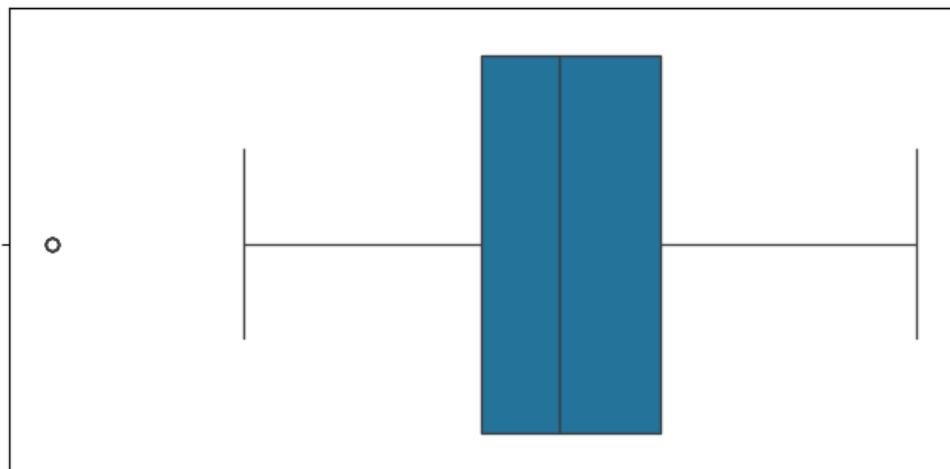
Used IQR method:

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 $Q1 - 1.5 \times IQR$ or above $Q3 + 1.5 \times IQR$.

```
▶ 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()
```



Box Plot for Glucose



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	-0.68442
1.2338801856003137	1.9437238810747468	-0.2639412465385531	-1.2882122129452358	-0.6928905722954675	-1.10325
-0.8448850534430228	-0.9982077796701243	-0.16054574674686284	0.15453319216918512	0.12330164444496892	-0.49404
-1.1418515161634994	0.5040551960293843	-1.5046872440388366	0.9072699252723613	0.7658359427299933	1.40974
0.34298079743888377	-0.15318485583915073	0.2530362524198983	-1.2882122129452358	-0.6928905722954675	-0.81134
-0.2509521280020695	-1.3424763782679283	-0.9877097450803851	0.7190857419965673	0.07120426890834532	-0.12597
1.8278131110412668	-0.18448200116622385	-3.572597239872642	-1.2882122129452358	-0.6928905722954675	0.41977
-0.5479185907225461	2.38188391565377	0.046245252836517724	1.5345505361916747	4.021921913768968	-0.18943
1.2338801856003137	0.12848945210450713	1.3903867501284914	-1.2882122129452358	-0.6928905722954675	-4.06047
0.04601433471840714	-0.3409677278015893	1.183595750545111	-1.2882122129452358	-0.6928905722954675	0.71168
1.8278131110412668	1.4742667011686503	0.2530362524198983	-1.2882122129452358	-0.6928905722954675	0.76245
1.8278131110412668	0.5666494866835304	0.5632227517949692	-1.2882122129452358	-0.6928905722954675	-0.62096
-0.8448850534430228	2.1315067530371854	-0.47073224612193365	0.15453319216918512	6.65283937836846	-0.24020
0.34298079743888377	1.411672410514504	0.149640752628208	-0.09637905219854027	0.8266162141893876	-0.78595
0.9369137228798371	-0.6539391810723203	-3.572597239872642	-1.2882122129452358	-0.6928905722954675	-0.25289
-1.1418515161634994	-0.09059056518500455	0.7700137513783497	1.6600066583755375	1.3041754899417703	1.75242
0.9369137228798371	-0.4348591637828086	0.2530362524198983	-1.2882122129452358	-0.6928905722954675	-0.30366

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)
```

Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	B
0.3529411764705882	0.7437185929648241	0.5901639344262295	0.3535353535353536	0.0	0.50074515
0.058823529411764705	0.4271356783919598	0.5409836065573771	0.29292929292929293	0.0	0.39642324
0.47058823529411764	0.9195979899497487	0.5245901639344263	0.0	0.0	0.34724292
0.058823529411764705	0.4472361809045226	0.5409836065573771	0.23232323232323235	0.1111111111111111	0.41877794
0.0	0.6884422110552764	0.3278688524590164	0.3535353535353536	0.19858156028368795	0.64232488
0.29411764705882354	0.5829145728643216	0.6065573770491803	0.0	0.0	0.38152011
0.1764705882352941	0.3919597989949749	0.4098360655737705	0.32323232323232326	0.10401891252955082	0.46199701
0.5882352941176471	0.577889447236181	0.0	0.0	0.0	0.52608047
0.11764705882352941	0.9899497487437187	0.5737704918032788	0.4545454545454546	0.6418439716312057	0.45454548
0.47058823529411764	0.628140703517588	0.7868852459016393	0.0	0.0	0.0
0.23529411764705882	0.5527638190954774	0.7540983606557378	0.0	0.0	0.56035767
0.5882352941176471	0.8442211055276382	0.6065573770491803	0.0	0.0	0.56631892
0.5882352941176471	0.6984924623115578	0.6557377049180328	0.0	0.0	0.40387481
0.058823529411764705	0.949748743718593	0.49180327868852464	0.23232323232323235	1.0	0.44858420
0.29411764705882354	0.8341708542713568	0.5901639344262295	0.19191919191919193	0.20685579196217493	0.38450074
0.4117647058823529	0.5025125628140703	0.0	0.0	0.0	0.44709388
0.0	0.592964824120603	0.6885245901639344	0.4747474747474748	0.2718676122931442	0.68256333
0.4117647058823529	0.5376884422110553	0.6065573770491803	0.0	0.0	0.44113263

```

▶ 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.")

```

Conclusion:

Thus we have performed pre-processing on the dataset of Alzheimers 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

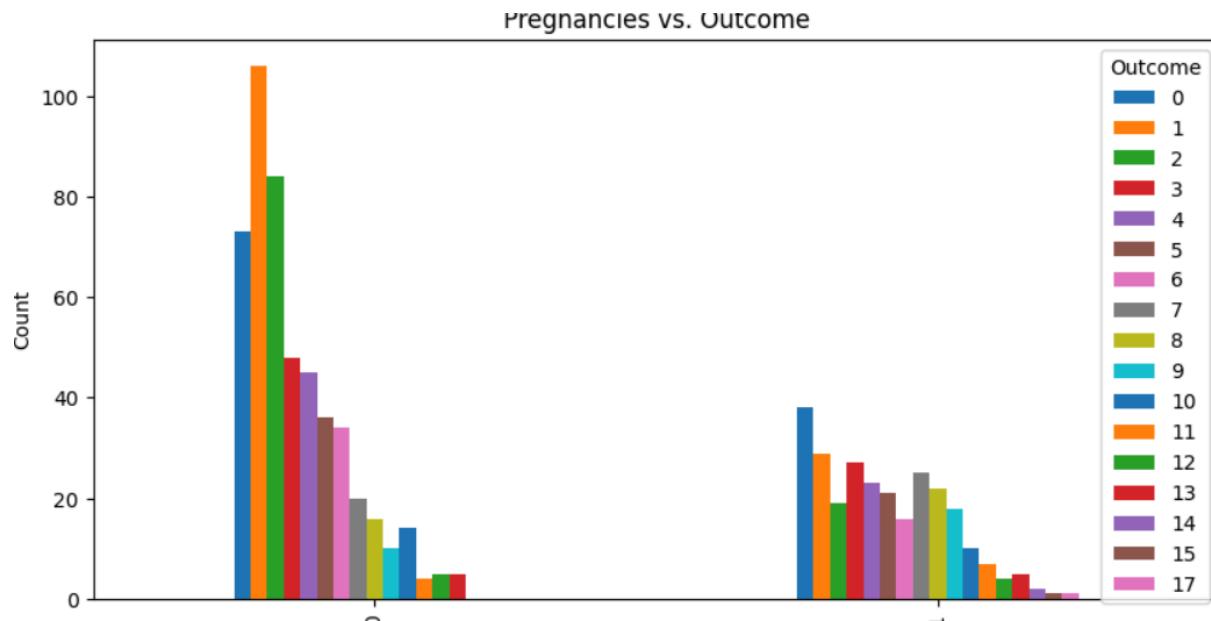
Steps:

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

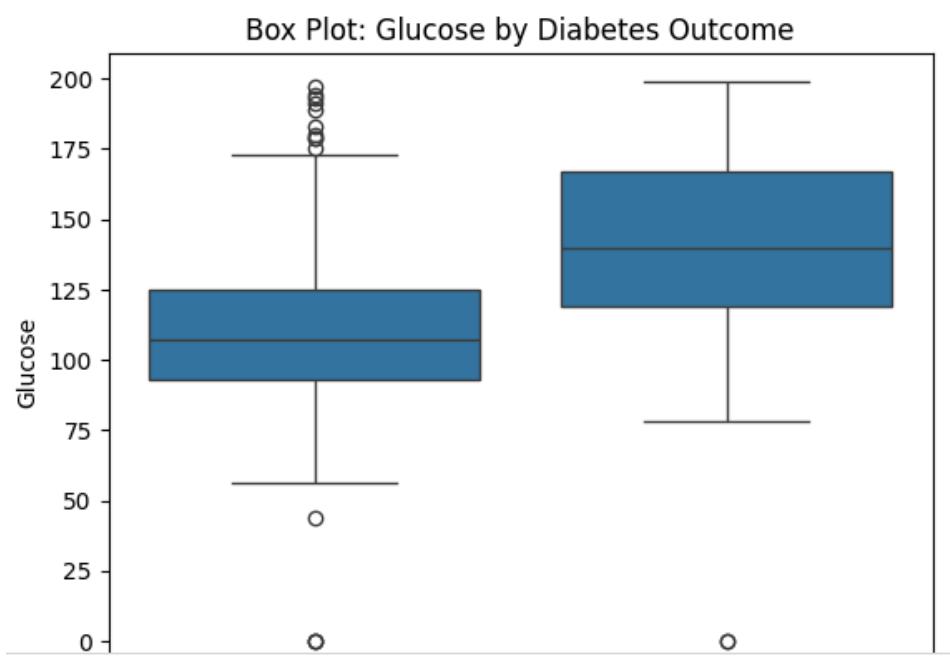
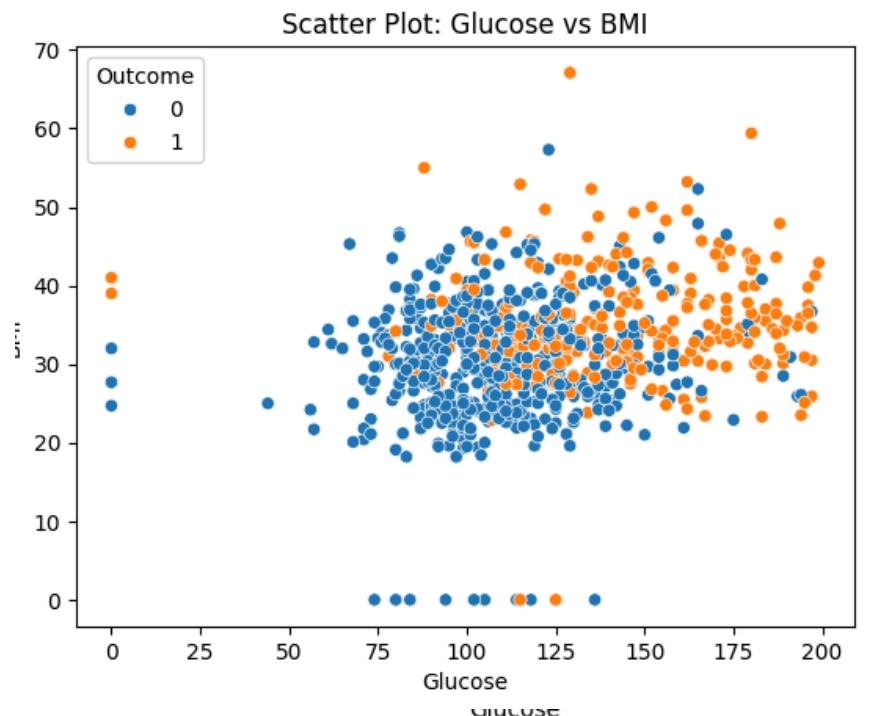
```

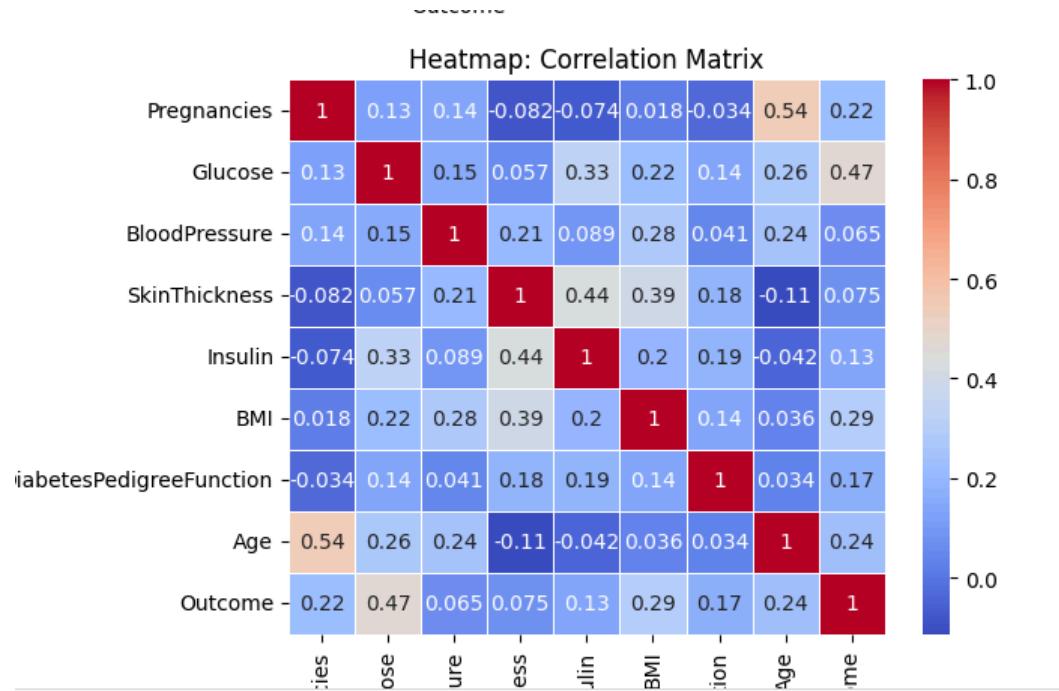
sns.scatterplot(x=df['Glucose'], y=df['BMI'], hue=df['Outcome'])
plt.title("Glucose vs. BMI")
plt.show()

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()

```

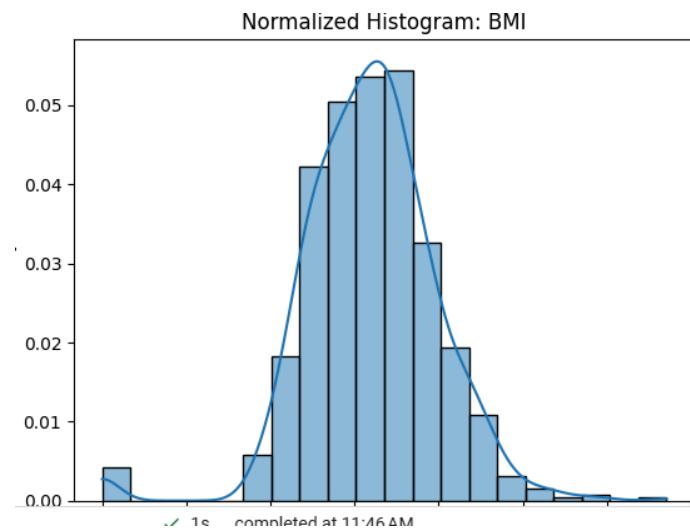
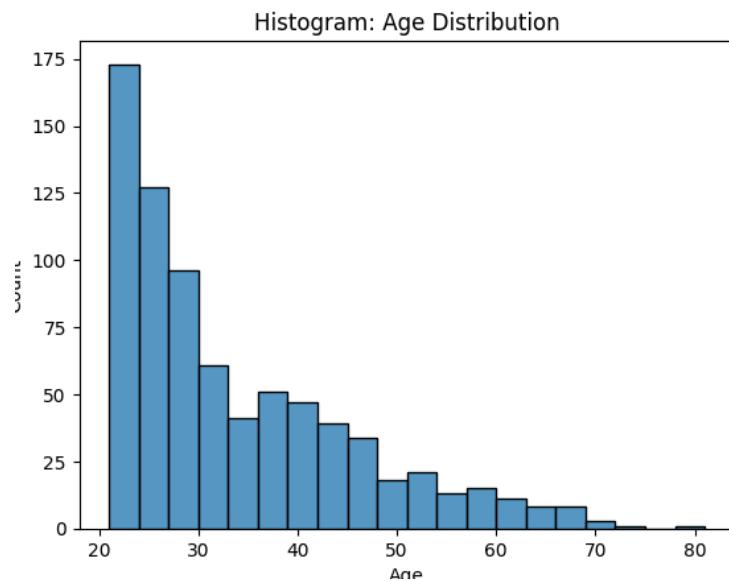




3. Create Histogram and Normalized Histogram

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 $Q1 - 1.5 * \text{IQR}$ and $Q3 + 1.5 * \text{IQR}$ 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.

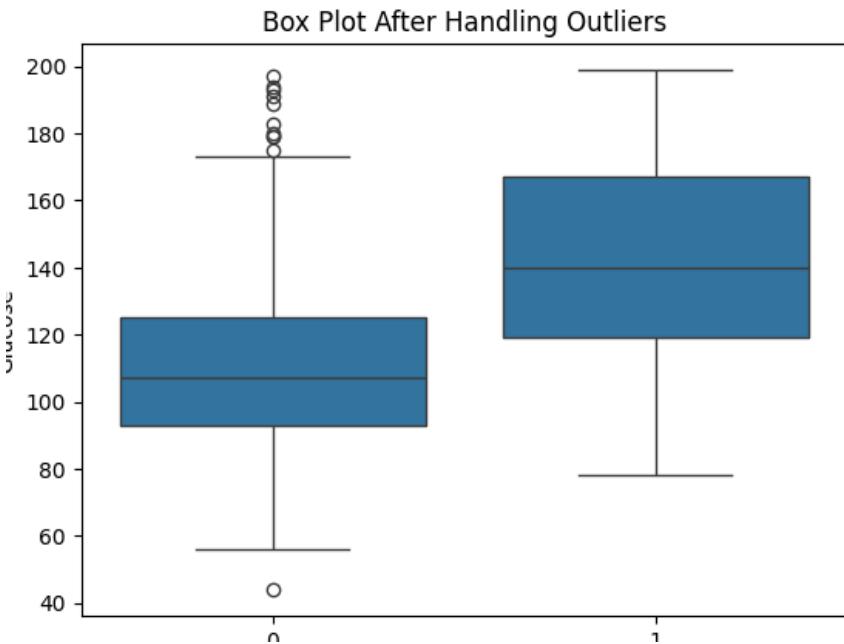
```
Q1 = df.quantile(0.25)
Q3 = df.quantile(0.75)
IQR = Q3 - Q1
```

```
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.

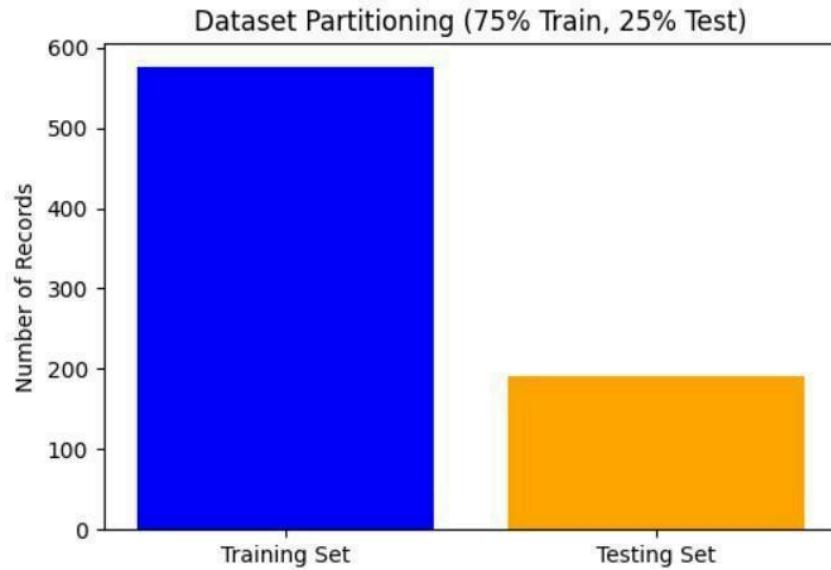
```
▶ # Step 1: Import Libraries
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from statsmodels.stats.proportion import proportions_ztest

# 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.

```
# 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 \times 0.75 and Testing Size = Total Data \times 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.

```
# 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.

```
▶ # 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:

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

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

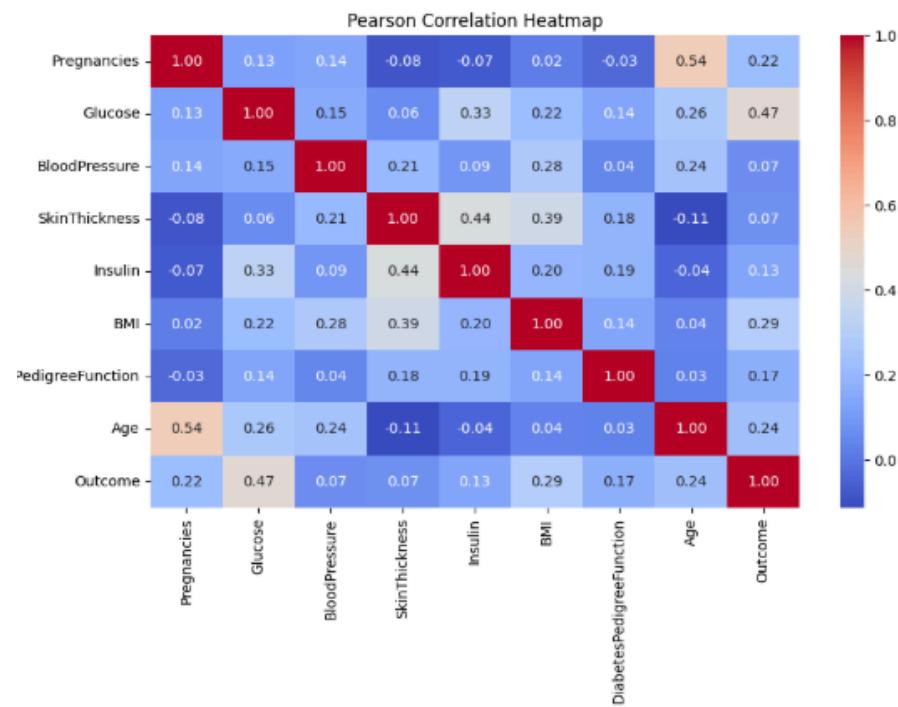
    # Heatmap of Pearson correlation
    plt.figure(figsize=(10, 6))
    sns.heatmap(pearson_corr, annot=True, cmap="coolwarm", fmt=".2f")
    plt.title("Pearson Correlation Heatmap")
    plt.show()
```

```
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:

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

```
▶ # 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	Age	Outcome
Pregnancies	1.000000	0.091323	0.135440	-0.064401	0.458272	0.170370
Glucose	0.091323	1.000000	0.159961	0.039046	0.196510	0.390565
BloodPressure	0.135440	0.159961	1.000000	0.094868	0.246056	0.119206
SkinThickness	-0.064401	0.039046	0.094868	1.000000	-0.044754	0.076297
Insulin	-0.096417	0.163645	-0.003682	0.420066	0.004183	0.253676
BMI	0.004183	0.155862	0.205222	0.331532	-0.029959	0.143359
DiabetesPedigreeFunction	-0.029959	0.061871	0.019448	0.126457	0.161652	0.028042
Age	0.458272	0.196510	0.246056	-0.044754	0.094644	0.058531
Outcome	0.170370	0.390565	0.119206	0.076297	1.000000	0.253676

	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
Pregnancies	-0.096417	0.004183	-0.029959	0.458272	0.170370
Glucose	0.163645	0.155862	0.061871	0.196510	0.390565
BloodPressure	-0.003682	0.205222	0.019448	0.246056	0.119206
SkinThickness	0.420066	0.331532	0.126457	-0.044754	0.058531
Insulin	1.000000	0.141587	0.161652	0.080176	0.253676
BMI	0.141587	1.000000	0.094644	0.088678	0.143359
DiabetesPedigreeFunction	0.161652	0.094644	1.000000	0.028042	0.253676
Age	-0.080176	0.088678	0.028042	1.000000	0.253676
Outcome	0.058531	0.253676	0.143359	0.253676	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 :

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

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

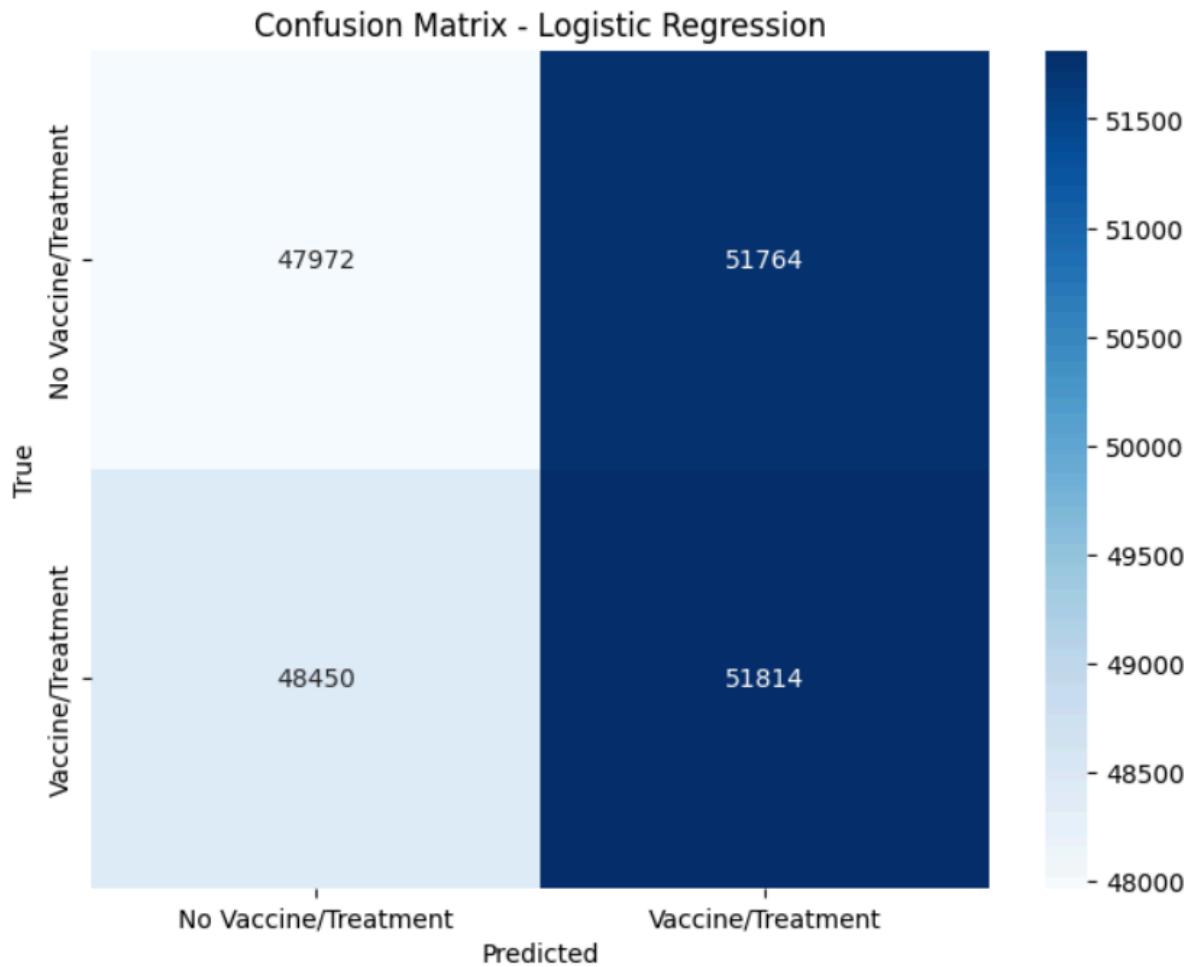
Confusion Matrix:

```
[[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



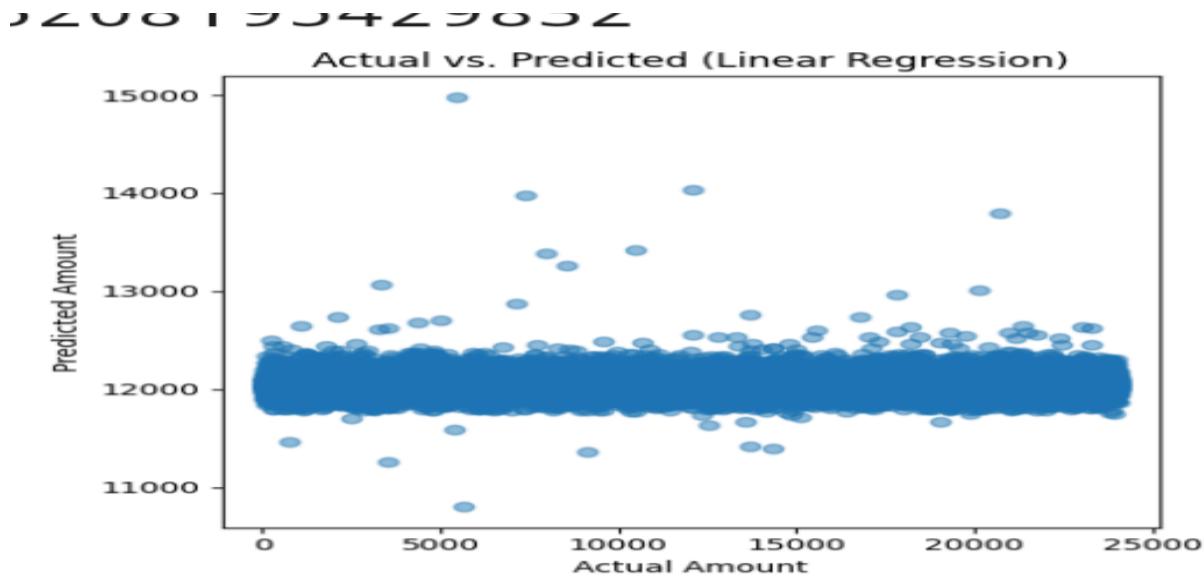
6. Linear Regression (Regression Analysis)

- 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 R² score.

Regression Model Results:
Mean Squared Error: 8.1939
 R^2 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 R² 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 implemented 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:

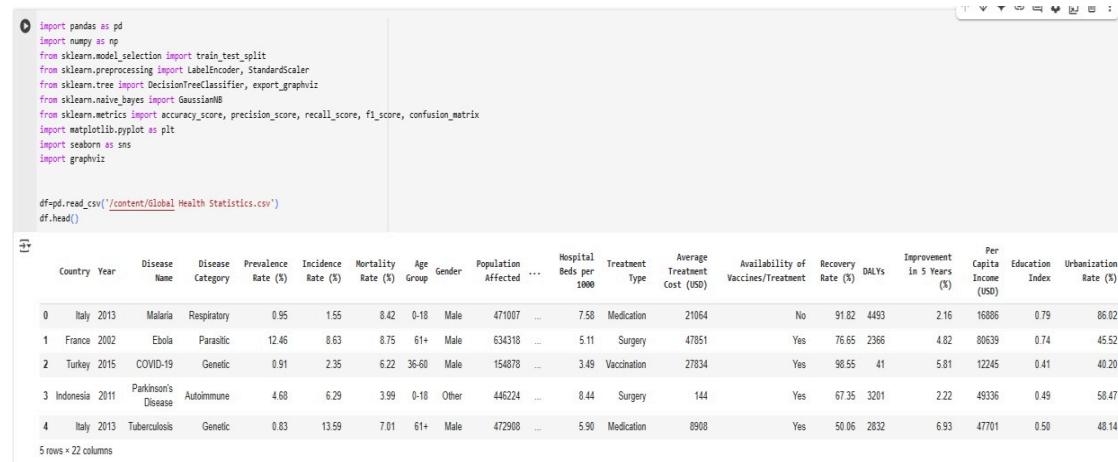
The **Global Health Statistics** dataset contains **1,000,000 records** with **22 attributes** related to disease prevalence, healthcare access, and socio-economic factors across different countries and years. It includes details on diseases, their categories, prevalence, incidence, mortality rates, 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.



The screenshot shows a Jupyter Notebook cell with Python code for importing libraries and reading the CSV file. Below the code, the first five rows of the dataset are displayed as a Pandas DataFrame.

```
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

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
```

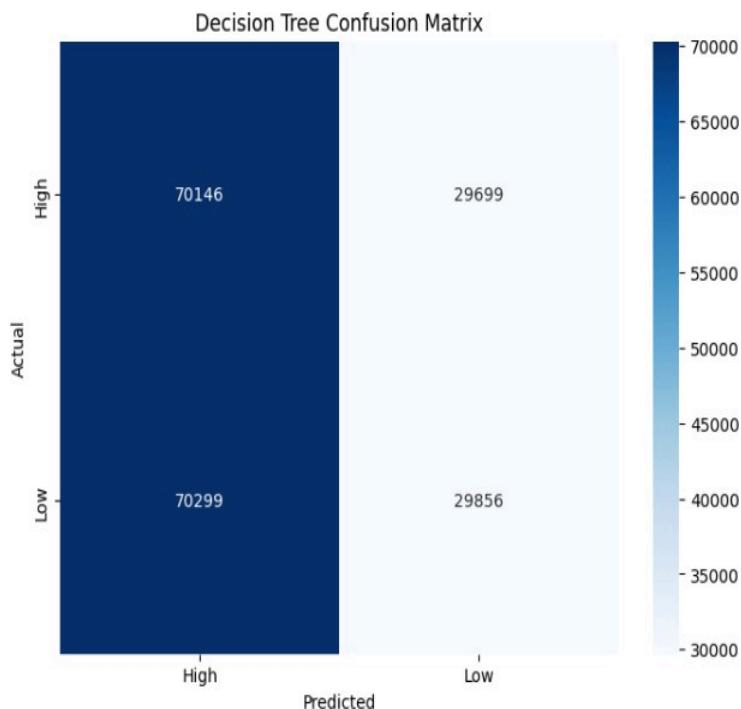
Experiment-6

4. Model Evaluation:

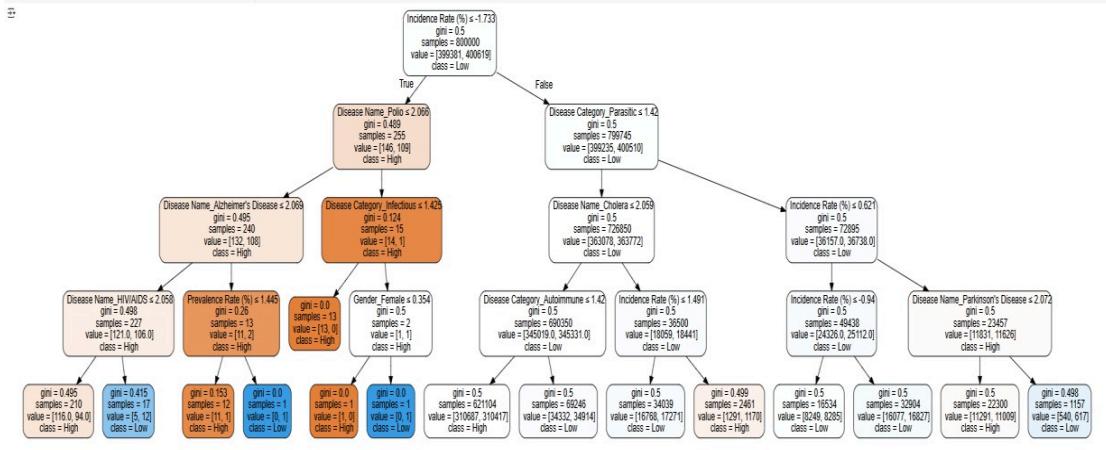
The heatmap visualizes classification performance, where diagonal values represent correct predictions, and off-diagonal values indicate misclassifications.

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

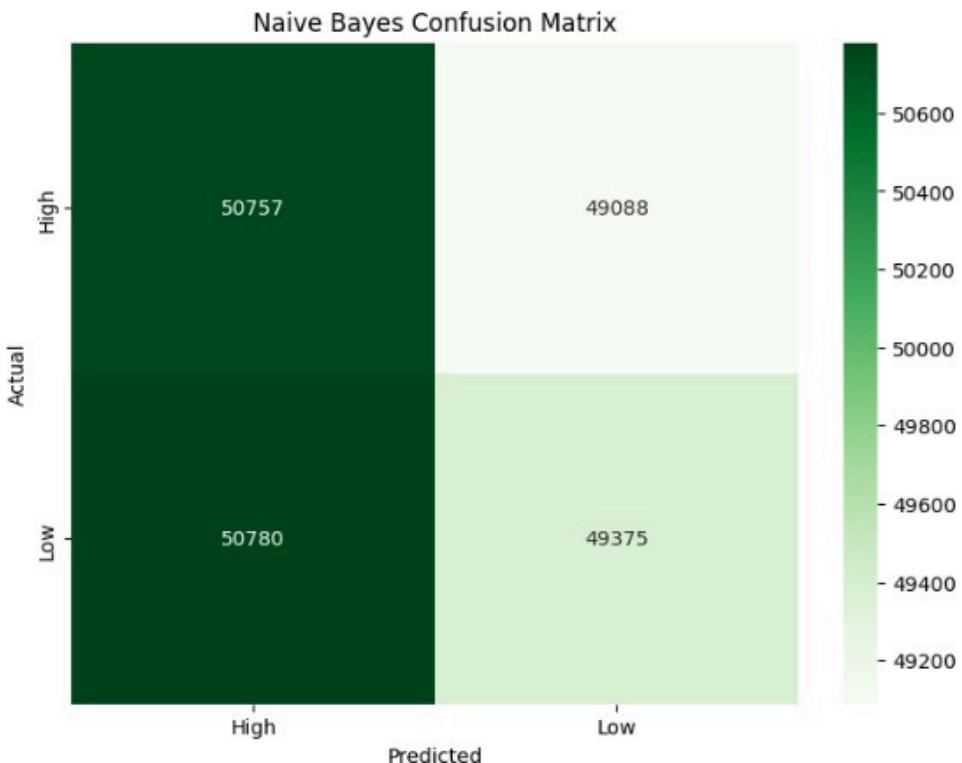
```
▶ # Predictions on the test set
y_pred_nb = nb_classifier.predict(X_test)

# 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)

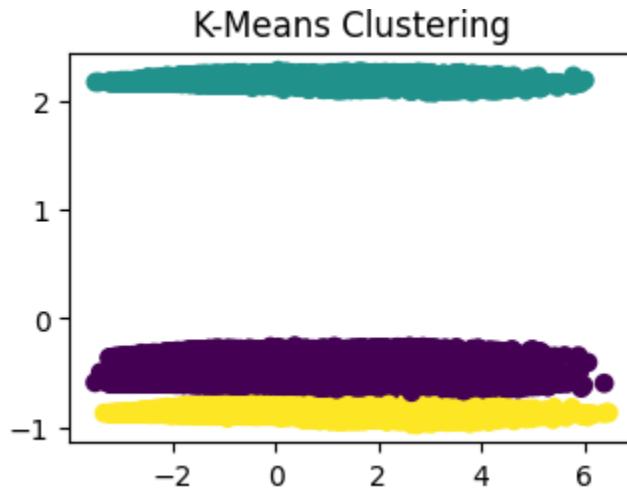
pca = PCA(n_components=2)
df_pca = pca.fit_transform(df_scaled)

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:

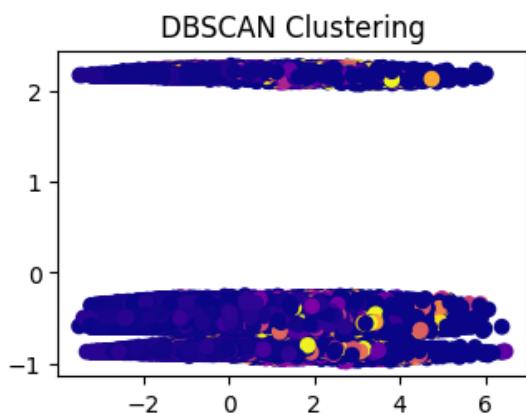
- DBSCAN(eps=0.5, min_samples=2): Groups points based on density.
- Noise points (outliers) are labeled as -1.
- Cluster labels are stored in df['dbscan_cluster'].

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:

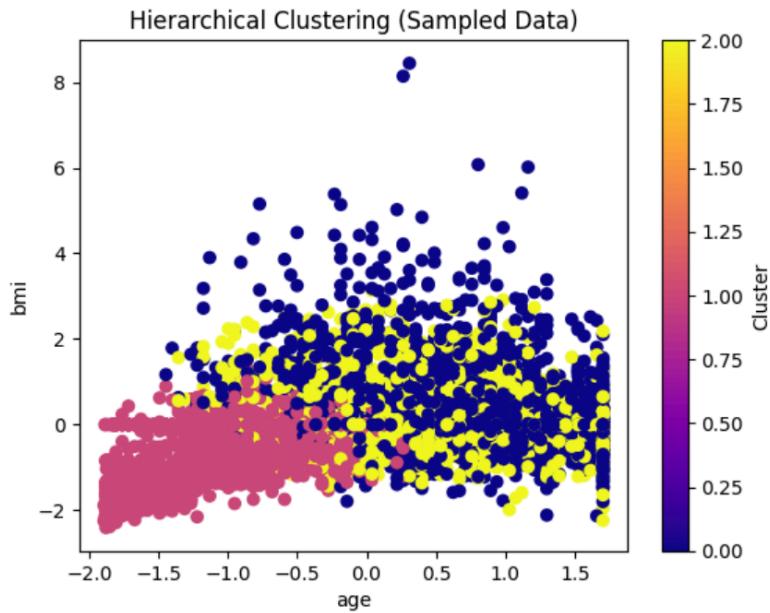
- Selected relevant numerical features (age, bmi, hbA1c_level, blood_glucose_level).
- Took a random sample of 5000 from the dataset to reduce memory usage.
- Scaled the data using StandardScaler() to normalize feature values.

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

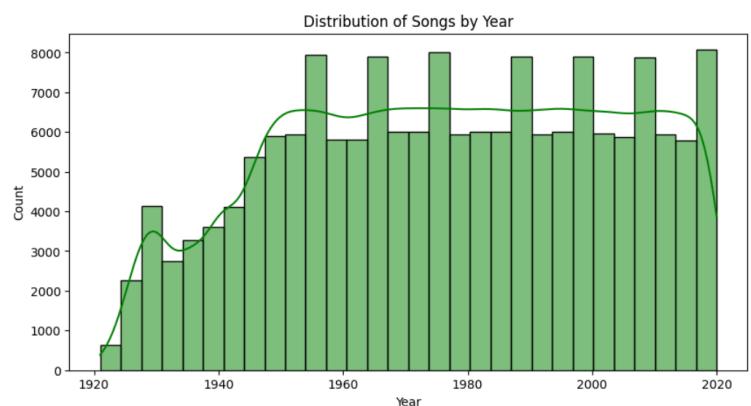
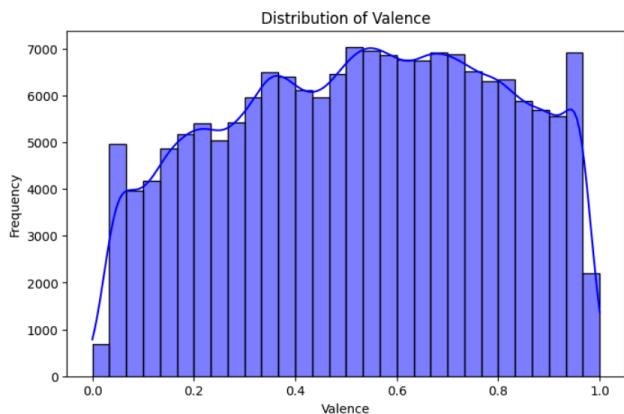
```
import pandas as pd
# Load the dataset
df = pd.read_csv("data.csv")
# Display first few rows
df.head()
```

	valence	year	acousticness	artists	danceability	duration_ms	energy	explicit	id	instrumentalness	key	liveness
0	0.0594	1921	0.982	['Sergei Rachmaninoff', 'James Levine', 'Berlin...']	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 Had...']	0.328	500062	0.166	0	1o6l8BglA6yIDMrlELygv1	0.913000	3	0.101

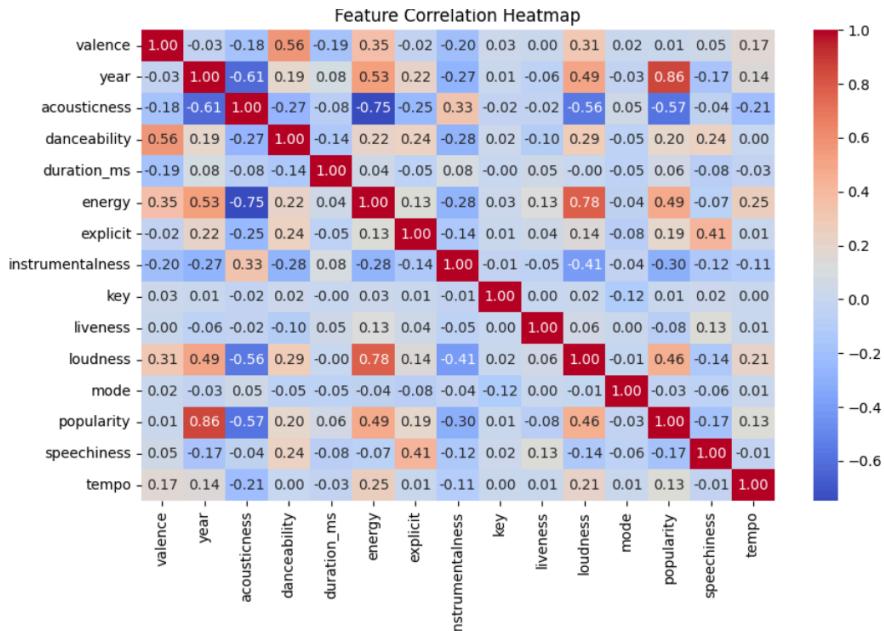
```
# Check for missing values
df.isnull().sum()
# Summary statistics
df.describe()
# Check data types
df.dtypes
```

```
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.

NAME : KAUSHAL S GALAV

DISC 11

AIDS ASSIGNMENT 1

Teacher: Ravita Mam

Date _____
Page _____

(1)

Q.1] What is AI? How AI helped During covid 19

AI refers to machines simulating human intelligence to perform tasks like learning, problem-solving, and decision-making.

AI applications during COVID 19.

- Healthcare: AI - assisted diagnosis (e.g. CT scan analysis for COVID 19 detection).
- Vaccine Development: AI helped speed up drug discovery.
- Robotics: AI powered robots assisted in hospitals and sanitization.
- Chatbots: Provided Covid 19 guidelines and answered queries.
- Remote Work & Education: AI improved online learning and virtual meeting.

Q.2] What are AI Agents terminology, explain with examples.

An AI agent perceives the environment and takes action to achieve goal.

- Simple Reflex Agent: Traffic light system.
- Model Based Agent: Google Maps.
- Goal Based Agent: Chess AI.
- Utility Based Agent: Autonomous
- Learning Agent: AI personal assistants.

Q.3] How AI techniques is used to solve 8 sum problem.

The 8 puzzle is solved using search algo eithms.

- Breadth First Search (BFS): Explores all possible moves level wise.
- Depth First Search (DFS): Explores paths deeply before backtracking.
- A* Algorithm: Uses heuristics function to find the shortest path efficiently.

Q.4] What is PEAS descriptor? Give PEAS descriptor for the following.

Taxi Driver.

Performance: Safe, timely trips.

Environment: Roads, traffic.

Actuators: Steering, brakes.

Sensors: Cameras, GPS.

Medical Diagnosis System.

P: Correct diagnosis.

E: Patient data.

A: Display reports.

S: Symptoms, test results.

Music Composer.

P: Melodic output

E: Music notes.

A: Sound synthesis.

S: User input, music trends

Aircraft Autoland

P: Smooth landing

E: Airport

A: Flaps, brakes.

S: Altimeter, radar.

Essay Evaluator.

P: Accuracy, coherence.

E: Essays.

A: Text processing.

S: NLP techniques.

Robotic Sentry Gun.

P: Accuracy, safety

E: Lab premises.

A: Gun, alarms

S: Motion sensors.

Q5] Categorise a shopping bot for an offline bookstore according to each of the six dimensions (Fully / partially observable, deterministic / stochastic, episodic / sequential, static / dynamic, discrete / continuous, single / multi agent).

Ans: 1) Fully observable - Can access stock availability.

2) Deterministic - Stock updates happen predictability.

3) Sequential - Each action affects future choices.

4) Dynamic! Stock and customer queries change in real time.

5) Discrete: Limited actions (search, buy, suggest)

6) Multi agent: Interacts with customer and supplier.

(c) Differentiate Model based and Utility based Agent

Model Based Agent

- Maintains an internal representation of the environment
- Uses past observations to update its model and predict future states

Example: A self driving car uses a map and sensor data to predict traffic conditions.

Utility Based Agent

- Chooses action based on a utility function that quantifies performance
- Tries to maximise expected utility rather than just achieving goal.

Example: A recommendation system suggests products based on customer preferences to maximise engagement.

Q.7] Explain the architecture of a knowledge based agent and learning agent.

knowledge - Based Agent

- Uses knowledge base that stores facts, rules, and logic
- Uses an Inference Engine to derive new knowledge.

Components:

- ① Perceptual System: Collects data from the environment.

- Date _____
Page _____
- 3
20/08
- ② knowledge Base: Stores facts and rules.
 - ③ Inference Engine: Applies logic to make decisions.
 - ④ Actuator: Executes actions based on reasoning.

Example: A medical diagnosis system that suggests treatments based on symptoms and medical knowledge.

- Learning Agent
 - Can improve its performance over time using feedback.
 - Components:
 - ① Learning Agent: Modifies the knowledge base using new data.
 - ② Performance Element: Uses existing knowledge to make decisions.
 - ③ Critic: Evaluates the performance of the agent.
 - ④ Problem Generator: Suggest new experiences to learn from
 - Example: A speech recognition system that adapts to a user's voice over time.

Q.8 What is AI? Considering the covid 19 pandemic situation, how AI helped to survive and renovated our way of life with different applications?

AI played a crucial role in handling the pandemic across different sectors:

- Healthcare & Diagnosis

AI powered CT scans helped detect Covid 19 in seconds.

- Drug discovery using AI accelerated vaccine deployment.

- Predicting & Controlling Outbreaks.

- AI models (like Blue Dot) predicted COVID 19 outbreaks before WHO alerts.

- AI based contact tracing apps helped track infections.

- Healthcare Automation.

- AI powered chatbots provide medical advice and reduced hospital workload.

- Robots assisted in disinfecting hospitals and delivering medicines.

- Remote Work & Education

- AI driven virtual meeting platforms helped business and schools operate remotely.

- AI enhanced e-learning platforms adapted lessons based on user performance.

Drones and AI powered robots delivered essential supplies to remote areas.

Q Convert the following to predicates.

① Anita travels by car if available otherwise travel by bus.

Step 1: Convert Sentences into Predicates.

① Anita travels by car if available, otherwise by bus

- Travels(Anita, car) \leftarrow Available(car)
- Travels(Anita, bus) \rightarrow Available(car)

② Bus goes via Andheri and Goregaon

• Cross(Bus, Andheri)

• Cross(Bus, Goregaon).

③ Car has a puncture, so it is not available

• Puncture(car) $\rightarrow \neg$ Available(car)

• Given: Puncture(car) is True.

Step 2: Apply forward Chaining (Reasoning).

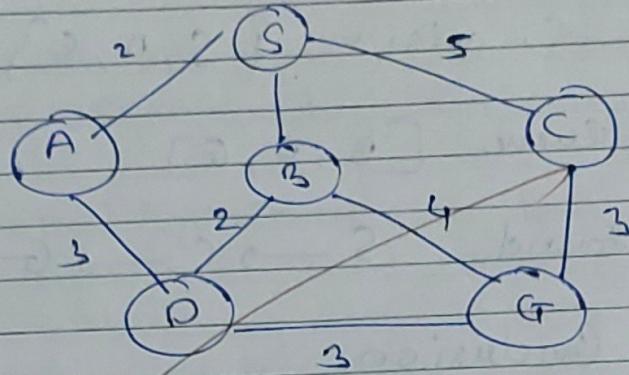
① Puncture(car) $\rightarrow \neg$ Available(car) \rightarrow True(car is not available)

② Since \neg Available(car), then Travels(Anita, Bus) is true.

3. Since Travels (Anita, Bus) and Bus goes via Goregaon, then Anita will travel via Goregaon.

Conclusion: Yes, Anita will travel via Goregaon.

10. Find the route from S to G using BFS.



Step 1: Understanding the graph.

Nodes and edges with weights.

$$S \rightarrow A(2)$$

$$S \rightarrow B(5)$$

$$A \rightarrow D(3)$$

$$B \rightarrow D(2)$$

$$B \rightarrow G(4)$$

$$C \rightarrow G(3)$$

BFS does not consider weights. It explores all neighbours level by level.

Step 2: Perform BFS traversal

Algorithm for BFS

- ① Start from node S, enqueue it in a queue.
- ② Visit its neighbours (A, C, B) and enqueue them.
- ③ Continue visiting and enqueueing neighbours until we reach G.

Execution

- ① Start from $S \rightarrow$ Queue [S])
- ② Dequeue S , visit $\{S\} \rightarrow$ Enqueue its
neighbours A, C, B
 \rightarrow Queue [A, C, B])
- ③ Dequeue A , visit $\{S, A\}$
 \rightarrow Queue [C, B, D])
- ④ Dequeue C , visit $\{S, A, C\} \rightarrow$ enqueue G
 \rightarrow Queue [B, D, G])
- ⑤ Path found $S \rightarrow C \rightarrow G$

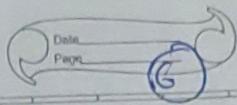
Step 3 Conclusion

BFS path ~~for~~ from S to G : $S \rightarrow C \rightarrow G$.

- (11) what do you mean by depth limited search?
? Explain iterative deepening search with example

Depth Limited Search (DLS)

- A variation of Depth First Search with a depth limit to avoid infinite recursion in deep trees.
- It avoids going beyond a specific limit.



Iterative Deepening Search.

- Combines BFS and DFS. It runs DFS repeatedly with increasing depth limit
- Efficient for finding the shortest path in an uninformed search.
- Example : Finding a goal in a tree with increasing depth ($0, 1, 2, 3 \dots$).

a) Explain Hill Climbing and its drawbacks in detail with example. Also state limitations of steepest ascent hill climbing

Hill Climbing Algorithm

- Greedy algorithm that moves towards increasing heuristic values.
- Stops when no further improvement is possible.

Drawbacks

- Local Maxima: Get stuck at peak that is not the global maximum.
- Plateau: No improvement in heuristic function.
- Ridges : A flat region with no clear gradient direction.

Steepest ascent Hill Climbing limitations.

- Always chooses the best possible move, which can lead to getting stuck in local maxima.

a) Explain Simulated Annealing & its Algorithm

Concept:

- Inspired by the annealing process in metals.
- Allows bad moves with decreasing probability to escape local maxima.

Algorithm:

- ① Start with an initial solution
- ② Select a neighbour solution randomly
- ③ If the neighbour is better, move to it, otherwise, accept it with a probability that decreases over time.
- ④ Repeat until the system cools down.

(Q.12) Explain A* Algorithm with example.

- Uses $f(n) = g(n) + h(n)$
- where g is cost and h is heuristics.
- Guarantees the shortest path if $h(n)$ is admissible.

Example

- Solving a shortest path problem in a grid or graph

(Q.15) Explain Min Max Algorithm and draw game tree for Tic Tac Toe game.

Minimax is a decision making algorithm used in two-player adversarial games like Chess & tic tac toe. It assumes:

- One player (max) tries to maximise the score
- The other player (MIN) tries to minimise the opponent's score.

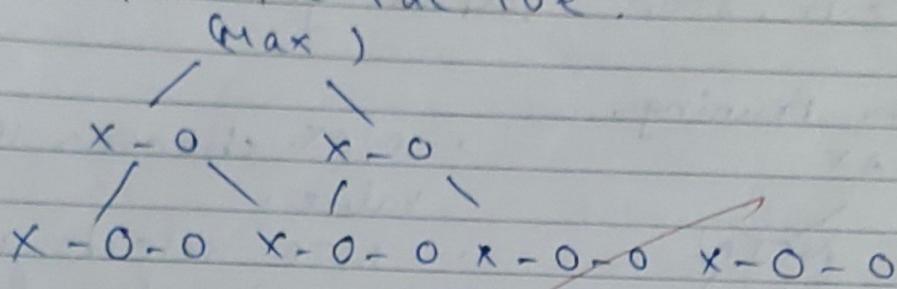
Minimax Algorithm Steps:

- ① Generate the game tree upto terminal states.
- ② Assign values (win/loss/draw) at leaf nodes.

3. Backpropagate scores using
- Max node: Chooses the maximum score from child nodes.
 - Min node: Chooses the minimum score from child nodes.

Q) The root node gets the best possible move.

Example Tic Tac Toe.



- Terminal nodes have value like +1 (win), 0 (draw), -1 (loss)
- The algorithm propagates these values to decide best move.

Why Minimax Important?

- Ensures best possible move for players.
- Used in AI powered bots for Chess, Tic-Tac-Toe, Checkers, etc.

Q) Explain Alpha Beta pruning algorithms for adversarial search with example.

Why Alpha - Beta Pruning?

- Optimise Minimax by eliminating unnecessary branches.

: Does not affect the final result but speed up computation.

Concept of α (Alpha) and β (Beta)

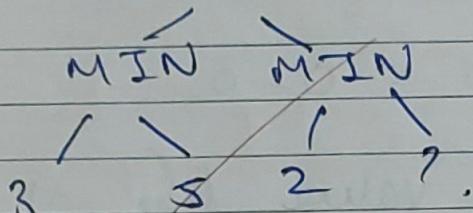
. α (Alpha): The best Max can guarantee so far.

. β (Beta): The best MIN can guarantee so far.

. If $\alpha \geq \beta$, stop exploring the branch

Alpha Beta Pruning.

MAX



. MAX chooses the highest value.

. MIN selects the lowest value from its children.

. If one branch already

proves worse than an explored branch, we prune it.

Key Benefit: Reduces the number of nodes examined, speeding up adversarial search.

a. i.) Wumpus World & PFAIS Representation.

What is Wumpus World?

A grid based environment where an agent

. Finds gold, avoids pits, and avoids the Wumpus.

. Uses percepts to decide moves.

PEAS Representation.

Performance

Finds gold, avoids dangers, minimizes steps.

Environment

A 4x4 grid with pits, Wumpus and gold.

Actuators
Sensors

Move, grab, shoot.
Breeze, Stench, Glitter

Percept Sequence Generation.

Agent perceives sensory data.
Example.

- If a Breeze is detected \rightarrow A pit is nearby.
- If a Stench is detected \rightarrow Wumpus nearby.

Q-18] Crypto Arithmetic SEND + MORE = MONEY.

Crypto-arithmetic problem involves mapping letters to unique digits (0-9) to satisfy equation.

$$\begin{array}{r} \text{SEND} \\ + \text{MORE} \\ \hline \text{MONEY} \end{array}$$

- ① Each letter represents a unique digit
- ② Carry proposition must be considered.
- ③ Solving leads to.

$$9567 + 1085 = 10652$$

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

Teacher's Sign.: T = 2,

(19)

First Order Logic & Resolution

Given Statement-

- ① All people who are graduating are happy.
 - ② All happy people are smiling.
 - ③ Someone is graduating.
- . $\forall n (\text{graduating}(n) \rightarrow \text{happy}(n))$
 - . $\forall n (\text{happy}(n) \rightarrow \text{smiling}(n))$
 - . $\exists n (\text{graduating}(n))$

Conversion to Clause Form

- . convert implications \rightarrow Remove universal quantifiers \rightarrow Skolemization \rightarrow Convert to CNF.

Resolution Proving Is someone smiling?

- . Apply unification and resolution rule to infer that someone must be smiling.
- . Draw the resolution tree showing clause cancellation leading to proof.

(22) Modus Ponens In Propositional Logic

A fundamental rule in logic

If $A \rightarrow B$ and A is true then B must be true.

Example

- ① If it rains, the ground will be wet.
• Rain(n) \rightarrow Wet(n),
- ② It is raining.
• Rain(n),
- ③ Therefore ground is wet

(21) Forward Chaining & Backward Chaining.

Forward Chaining

• Starts with facts \rightarrow Applies rules \rightarrow Derives new facts \rightarrow Reaches conclusion.

- Example: Expert systems like MYCIN
 - Rule: If fever & cough \rightarrow Flu,
 - Given: Patient has fever & cough,
 - Conclusion: Patient has flu.

Backward Chaining

~~• Starts with goal \rightarrow Work backwards \rightarrow Checks if facts support it.~~

• Example: Theorem proving in Prolog

• Goal: Prove "Flu"

• Check: Does patient has fever & cough?

• If yes, Flu is confirmed.

AIDS-I Assignment No: 2

Q.1: Statistical Analysis of the Dataset

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

ANS - Mean (10 pts): The mean is calculated by summing all values and dividing by the total number of values. $\text{Mean} = (82 + 66 + 70 + 59 + 90 + 78 + 76 + 95 + 99 + 84 + 88 + 76 + 82 + 81 + 91 + 64 + 79 + 76 + 85 + 90) / 20 = 1601 / 20 = \mathbf{80.05}$

Median (10 pts): Arrange the data in ascending order: 59, 64, 66, 70, 76, 76, 76, 78, 79, 81, 82, 82, 84, 85, 88, 90, 90, 91, 95, 99 Median = (10th value + 11th value) / 2 = (81 + 82)/2 = **81.5**

Mode (10 pts): The most frequent number is 76, which appears 3 times. Mode = **76**

Interquartile Range (20 pts): Q1 (25th percentile) = median of first half = (5th + 6th)/2 = (76 + 76)/2 = **76** Q3 (75th percentile) = median of second half = (15th + 16th)/2 = (88 + 90)/2 = **89** IQR = Q3 - Q1 = **89 - 76 = 13**

Q.2: Comparison of Machine Learning Tools for Kids

ANS - 1. Machine Learning for Kids

- **Target Audience:** School-aged children (ages 8–16), educators
- **Use by Audience:** Allows students to create ML projects using image, text, or number-based data. Integrated with Scratch and Python.
- **Benefits:** Intuitive, easy to use, free, integrates with existing educational platforms.
- **Drawbacks:** Limited model complexity; less suitable for advanced users.
- **Analytic Type:** **Predictive Analytic** - as it uses trained models to make predictions.
- **Learning Type:** **Supervised Learning** - users provide labeled examples for training.

2. Teachable Machine

- **Target Audience:** Beginners, educators, artists, and developers.
- **Use by Audience:** Users can train models using images, audio, or poses through a webcam/microphone.
- **Benefits:** No coding required, real-time interaction, fast model training.
- **Drawbacks:** Models may overfit, not suitable for large-scale applications.
- **Analytic Type:** **Predictive Analytic** - it predicts based on real-time input.
- **Learning Type:** **Supervised Learning** - trained using labeled input data.

Q.3: Misinformation in Data Visualization

ANS - Data visualization is a powerful tool to convey complex information quickly and intuitively. However, when misused—intentionally or unintentionally—it can lead to serious misinformation. This often happens during crises such as the COVID-19 pandemic, where public perception directly influences behavior and policy.

Key Articles and Insights:

1. Arthur Kakande – "What's in a chart?" (Medium)
 - This article provides a step-by-step guide to identifying misleading charts. It highlights common red flags such as:
 - Truncated or manipulated axes
 - Misleading use of color
 - Unlabeled data points or scales
 - Charts without context
 - The goal is to teach readers how to critically evaluate visual data, emphasizing that beautiful graphics can still be deceptive.
2. Katherine Ellen Foley – “How bad Covid-19 data visualizations mislead the public” (Quartz)
 - The article discusses real examples where poor design choices contributed to public misunderstanding.
 - It stresses the responsibility of media and data scientists in avoiding:
 - Overuse of cumulative data (which hides trends)
 - Inappropriate chart types (e.g., pie charts for time series)
 - Inconsistent scales and color coding

Example of Real-World Misinformation:

Case Study: Misleading COVID-19 Charts in U.S. States During the early months of the pandemic, some state governments (notably Georgia and Florida) published bar graphs of COVID-19 cases where:

- The x-axis (dates) was out of order (e.g., jumping from May 7 to May 4 and back to May 6).
- The colors representing counties were reused inconsistently.
- Y-axes were truncated, making trends appear flat even as case numbers surged.

Why This Was Misleading:

- The non-chronological order gave the illusion of declining cases.
- Truncated y-axes minimized visual differences, suggesting improvement
- Inconsistent color mapping confused which regions were most affected.

Impact:

- These visualizations gave citizens and policymakers a false sense of security.
- Some interpreted the misleading visuals as evidence that restrictions could be lifted safely.
- Public health experts criticized these charts as dangerous.

Better Practices:

- Always keep axes clearly labeled and in proper order.
- Avoid truncated y-axes unless justified—and always indicate truncation.

- Use consistent and meaningful color schemes.
- Present both raw and per-capita numbers for clearer comparison.

Cited Source:

- The New York Times, "The Worst Charts of the Pandemic—and How We Can Do Better"
<https://www.nytimes.com/2020/07/02/upshot/coronavirus-data-charts.html>

Q. 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

ANS -

```

 1s  import numpy as np
 2s  from sklearn.model_selection import train_test_split
 3s  from sklearn.preprocessing import StandardScaler
 4s
 5s  # Features where 0 is invalid and should be considered missing
 6s  cols_with_zero_invalid = ['Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI']
 7s
 8s  # Replace 0s with NaN
 9s  df[cols_with_zero_invalid] = df[cols_with_zero_invalid].replace(0, np.nan)
10s
11s  # Impute missing values with median
12s  df.fillna(df.median(), inplace=True)
13s
14s  # Separate features and target
15s  X = df.drop('Outcome', axis=1)
16s  y = df['Outcome']
17s
18s  # Scale features
19s  scaler = StandardScaler()
20s  X_scaled = scaler.fit_transform(X)
21s
22s  # Train-Validation-Test Split
23s  X_train_val, X_test, y_train_val, y_test = train_test_split(X_scaled, y, test_size=0.10, random_state=42, stratify=y)
24s  X_train, X_val, y_train, y_val = train_test_split(X_train_val, y_train_val, test_size=2/9, random_state=42, stratify=y_train_val)  # (2
25s
26s  # Final shapes
27s  print("Train set:", X_train.shape)
28s  print("Validation set:", X_val.shape)
29s  print("Test set:", X_test.shape)

 3s  ↗ Train set: (537, 8)
 4s  Validation set: (154, 8)
 5s  Test set: (77, 8)

 6s  [7]  from imblearn.over_sampling import SMOTE
 7s  from collections import Counter

```

```

✓ 0s  from imblearn.over_sampling import SMOTE
    from collections import Counter

    # Apply SMOTE only on the training data (NOT on validation or test)
    smote = SMOTE(random_state=42)
    X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)

    # Check the class distribution before and after
    print("Before SMOTE:", Counter(y_train))
    print("After SMOTE:", Counter(y_train_resampled))

→ Before SMOTE: Counter({0: 350, 1: 187})
After SMOTE: Counter({0: 350, 1: 350})

```

```

print("\nMissing values:\n", df.isnull().sum())

→ Shape of dataset: (768, 9)

Data types:
Pregnancies          int64
Glucose              int64
BloodPressure        int64
SkinThickness        int64
Insulin              int64
BMI                  float64
DiabetesPedigreeFunction float64
Age                  int64
Outcome              int64
dtype: object

Sample data:
   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin    BMI \
0           6      148            72            35         0  33.6
1           1       85            66            29         0  26.6
2           8      183            64             0         0  23.3
3           1       89            66            23         94  28.1
4           0      137            40            35        168  43.1

   DiabetesPedigreeFunction  Age  Outcome
0                   0.627  50       1
1                   0.351  31       0
2                   0.672  32       1
3                   0.167  21       0
4                   2.288  33       1

Missing values:
Pregnancies          0
Glucose              0
BloodPressure        0
SkinThickness        0
Insulin              0
BMI                  0
DiabetesPedigreeFunction 0
Age                  0
Outcome              0
dtype: int64

```

The screenshot shows two code cells in a Jupyter Notebook.

Code Cell 1:

```
# best_SVM = grid_search.best_estimator_
# Evaluate on validation set
y_val_pred = best_svm.predict(X_val)
print("◆ SVM Validation Accuracy:", accuracy_score(y_val, y_val_pred))
print("◆ Classification Report:\n", classification_report(y_val, y_val_pred))
```

Output 1:

```
◆ SVM Validation Accuracy: 0.6948051948051948
◆ Classification Report:
precision    recall   f1-score   support
          0       0.78      0.73      0.76      100
          1       0.56      0.63      0.59       54

   accuracy                           0.69      154
  macro avg       0.67      0.68      0.67      154
weighted avg     0.71      0.69      0.70      154
```

Code Cell 2:

```
[10] from sklearn.naive_bayes import GaussianNB
nb = GaussianNB()
nb.fit(X_train_resampled, y_train_resampled)

# Evaluate on validation set
y_val_pred_nb = nb.predict(X_val)
print("◆ Naive Bayes Validation Accuracy:", accuracy_score(y_val, y_val_pred_nb))
print("◆ Classification Report:\n", classification_report(y_val, y_val_pred_nb))
```

Output 2:

```
◆ Naive Bayes Validation Accuracy: 0.7662337662337663
◆ Classification Report:
precision    recall   f1-score   support
          0       0.83      0.80      0.82      100
          1       0.66      0.70      0.68       54

   accuracy                           0.77      154
  macro avg       0.74      0.75      0.75      154
weighted avg     0.77      0.77      0.77      154
```

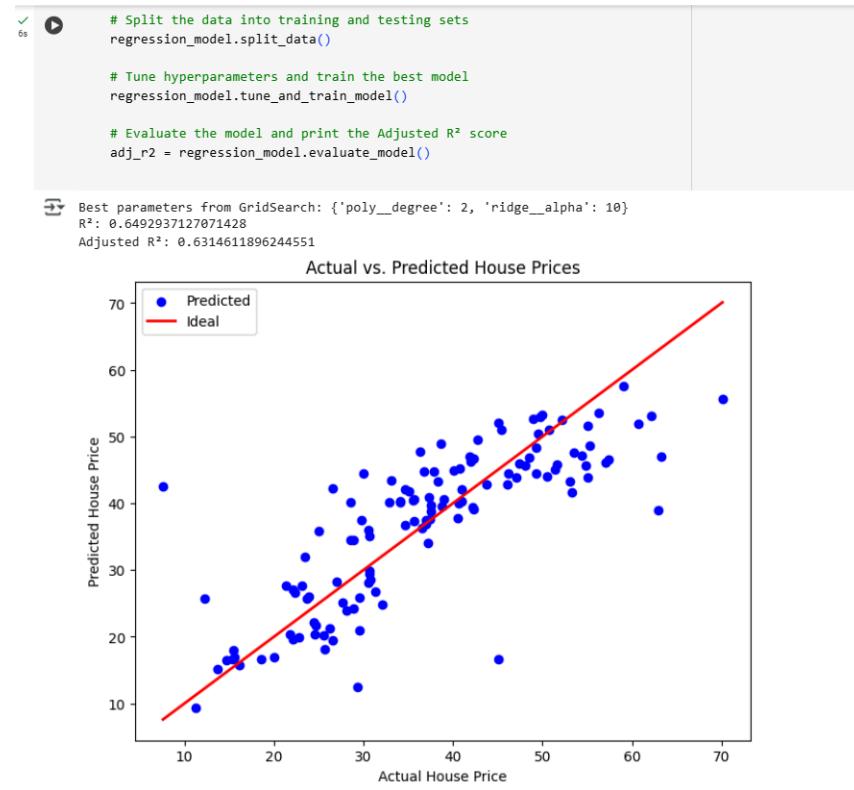
Q.5 Train Regression Model and visualize the prediction performance of trained model

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

Use any Regression model to predict the values of all Dependent variables using values of 1st column.

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

ANS -

Q.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).

ANS - Key Features:

- Fixed acidity, Volatile acidity, Citric acid, Residual sugar, Chlorides, Free sulfur dioxide, Total sulfur dioxide, Density, pH, Sulphates, Alcohol
- Importance:
 - Alcohol: Most correlated with quality (higher alcohol, better quality)
 - Volatile Acidity: Negative correlation (lower acidity, better quality)
 - Sulphates and Citric Acid: Add to flavor; positively correlated
- Handling Missing Data:
 - Technique Used: Mean/median imputation for numerical columns
 - Alternatives: KNN Imputation, model-based imputation
- Advantages and Disadvantages:
 - Mean/Median: Easy, fast; may reduce variability
 - KNN: Captures local structure; computationally expensive
 - Model-based: More accurate; risk of bias, overfitting
- Dataset Source: Kaggle Wine Quality Dataset
(<https://www.kaggle.com/datasets/uciml/red-wine-quality-cortez-et-al-2009>)

Kaushal Sahebrao Galav

Roll no - 11

D15C