What is Python?

Python is a **high-level**, **interpreted**, and **general-purpose programming language** known for its **simple and readable syntax**. It was created by **Guido van Rossum** and first released in **1991**.

Python is very popular among beginners and professionals because it is:

- Easy to learn and write
- Versatile (used in many fields)
- Open-source and has a large community
- Rich in libraries and frameworks

Uses of Python

Python is used in a wide range of areas:

1. Artificial Intelligence & Machine Learning

- Libraries: `TensorFlow`, `PyTorch`, `Scikit-learn`
- Used to build models that can learn and make predictions

2. Web Development

- Frameworks: `Django`, `Flask`, `FastAPI`
- Used to create dynamic websites and REST APIs

3. Data Science & Data Analysis

- Libraries: `Pandas`, `NumPy`, `Matplotlib`, `Seaborn`
- Used to analyze, visualize, and manipulate data

4. Automation & Scripting

- Automate boring or repetitive tasks
- Example: Web scraping, file management, email sending

5. Desktop Applications

- Libraries: `Tkinter`, `PyQt`
- Used to create GUI-based applications

6. Game Development

- Libraries: `Pygame`
- Used to build simple 2D games

What is NumPy?

NumPy (short for Numerical Python) is a powerful Python library used for numerical and scientific computing. It provides support for:

- Multidimensional arrays (ndarrays)
- Mathematical functions
- Linear algebra
- Random number generation

NumPy is fast and efficient, and it's the foundation of many other libraries like Pandas, TensorFlow, and Scikit-learn.

Key Features of NumPy

- Powerful **N-dimensional arrays**
- Very **fast** operations (written in C under the hood)
- Supports **vectorized operations** (no need for loops)
- Tools for working with **matrices, Fourier transforms, and statistics**

What is Pandas?

Pandas is a powerful **open-source Python library** used for **data analysis and data manipulation**. It is built on top of **NumPy** and makes working with **structured data** (like tables or spreadsheets) easy and efficient.

Core Data Structures in Pandas

1. **Series** – 1D labeled array (like a single column)
2. **DataFrame** – 2D table with rows and columns (like an Excel sheet)

Key Features of Pandas

Feature	Description
DataFrame Support	Handles labeled 2D data like SQL tables or Excel files
Data Cleaning	Handle missing data, filter, replace, or fill values
Data Aggregation	Grouping, summarizing, counting, mean, etc.
Data Analysis	Perform statistical operations and data exploration
File Handling	Easily read/write from CSV, Excel, JSON, SQL, etc.
Feature	**Description**
Time Series Handling	Built-in support for date/time functions and indexing

Data Visualization	Integrates well with libraries like Matplotlib and Seaborn for plotting
Fast Operations	Built on top of NumPy, optimized for performance

---

What is Matplotlib?

Matplotlib is a Python library used for **data visualization**. It lets you create **line charts, bar charts, scatter plots**, and many other types of graphs.

Key Features of Matplotlib

Feature	Description
Plotting Variety	Line, bar, scatter, pie, histogram, etc.
Customization	Control over colors, labels, legends, etc. Subplots
Support	Create multiple plots in one figure
Exporting	Save plots as PNG, PDF, SVG, etc. Annotating Graphs
	Add labels, arrows, and custom styling

What is Seaborn?

Seaborn is a Python visualization library built **on top of Matplotlib**. It provides a **high-level interface** for making beautiful and informative statistical graphics.

Key Features of Seaborn

Feature	Description
Easy Plotting	One-liner plots like histograms and box plots
Statistical Graphs	Built-in support for correlation and regression
Beautiful Themes	Pre-set styles and color palettes Works with
Pandas	Supports DataFrames directly Complex Plots Made
Easy	Heatmaps, pairplots, violin plots, etc.

What is SciPy?

SciPy (Scientific Python) is a Python library used for **scientific and technical computing**. It builds on NumPy and adds advanced features.

Key Features of SciPy

Feature	Description

Advanced Math	Integration, differentiation, optimization Statistics
	Probability distributions, statistical tests Linear Algebra Matrix operations, eigenvalues, etc.
Signal & Image Processing	Tools for filtering, image analysis, etc.
Performance	Fast calculations built in C/Fortran

What is Scikit-learn (sklearn)?

Scikit-learn is a powerful **machine learning library** in Python. It provides simple tools for **classification, regression, clustering**, and more.

Key Features of Scikit-learn

Feature	Description
Machine Learning	Classification, regression, clustering
Model Tools	Training, testing, model evaluation
Preprocessing	Feature scaling, encoding, normalization
Model Selection	Cross-validation, hyperparameter tuning
Pipeline Support	Combine multiple steps into one workflow

1.Write a Python program to calculate the Mean, Median, and Standard Deviation of a dataset using a NumPy array.

[1]:
```
import numpy as np
```

[14]:
```
marks = [23,23,12,45,50,22,21,45,34,36]
print(type(marks))
print(marks)
marks = np.array(marks)
print(marks)
print(type(marks))
```

```
<class 'list'>
[23, 23, 12, 45, 50, 22, 21, 45, 34, 36]
[23 23 12 45 50 22 21 45 34 36]
<class 'numpy.ndarray'>
```

[8]:
```
avg_marks = np.mean(marks)
middle_marks = np.median(marks)
stdv_marks = np.std(marks)
```

[12]:
```
print(f"Marks:\nAverage: {avg_marks} \nMiddle_marks: {middle_marks} \nDeviation_marks: {stdv_marks}")
```

```
Marks:
Average: 31.1
Middle_marks: 28.5
Deviation_marks: 12.070211265756702
```

[ ]:

2)Write a Python program to read a CSV file, clean the data by handling missing values, and perform basic data analysis using the Pandas library.

[36]:
```
import numpy as np
import pandas as pd
```

[5]:
```
medical_data = pd.read_csv('medical.csv')
```

[7]:
```
print("Total null values column vise:\n\n",medical_data.isnull().sum())
```

```
Total null values column vise:

 Patient_ID        0
Name               1
Age                2
Gender             1
Diagnosis          1
Blood_Pressure     1
Heart_Rate         1
Cholesterol        1
dtype: int64
```

Cleaning of data

Method1: Droping all the null value

[8]:
```
med_data_final = medical_data.dropna()
```

[9]:
```
med_data_final
```

[9]:

	Patient_ID	Name	Age	Gender	Diagnosis	Blood_Pressure	Heart_Rate	Cholesterol

Method1: Droping all the null value

[8]:
```
med_data_final = medical_data.dropna()
```

[9]:
```
med_data_final
```

[9]:

	Patient_ID	Name	Age	Gender	Diagnosis	Blood_Pressure	Heart_Rate	Cholesterol
0	101	Alice	29.0	F	Diabetes	120/80	78.0	190.0
1	102	Bob	45.0	M	Hypertension	140/90	85.0	220.0

Method2: Replacing with the Central tendencies

[10]:
```
medical_data
```

[10]:

	Patient_ID	Name	Age	Gender	Diagnosis	Blood_Pressure	Heart_Rate	Cholesterol
0	101	Alice	29.0	F	Diabetes	120/80	78.0	190.0
1	102	Bob	45.0	M	Hypertension	140/90	85.0	220.0
2	103	Charlie	34.0	M	Asthma	110/70	90.0	NaN
3	104	David	NaN	M	Diabetes	135/85	88.0	240.0
4	105	Eva	41.0	NaN	Heart Disease	125/80	NaN	210.0
5	106	Frank	50.0	M	NaN	145/95	80.0	200.0
6	107	NaN	38.0	F	Hypertension	NaN	76.0	230.0

[35]:

	Patient_ID	Name	Age	Gender	Diagnosis	Blood_Pressure	Heart_Rate	Cholesterol
0	101	Alice	29.0	F	Diabetes	120/80	78.0	190.0
1	102	Bob	45.0	M	Hypertension	140/90	85.0	220.0
2	103	Charlie	34.0	M	Asthma	110/70	90.0	205.0
3	104	David	39.8	M	Diabetes	135/85	88.0	240.0
4	105	Eva	41.0	M	Heart Disease	125/80	82.5	210.0
5	106	Frank	50.0	M	Diabetes	145/95	80.0	200.0
7	108	Hannah	39.8	F	Diabetes	130/85	72.0	180.0

[70]:
```
#funciton to check is data is cleaned or not
def Check_clean(data):
    d = dict(data.isnull().sum())
    null_values = []
    for key,value in d.items():
        if value != 0:
            null_values.append((key,value))
    return 'Cleaned' if data.isnull().sum().sum() == 0 else null_value
```

[71]:
```
Check_clean(data)
```

[71]:
```
'Cleaned'
```

3.) Create a Python program to plot a scatter plot, histogram, and boxplot using Matplotlib and Seaborn.

[5]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

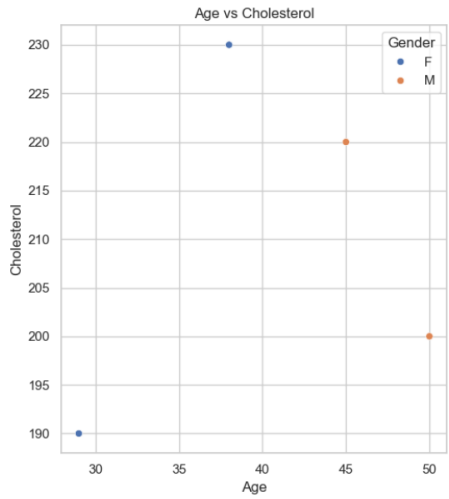
[6]:
```
data = pd.read_csv("medical.csv")
```

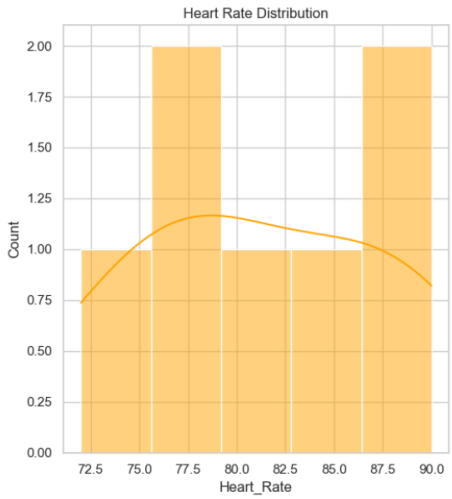
[8]:
```
df = pd.DataFrame(data)
```

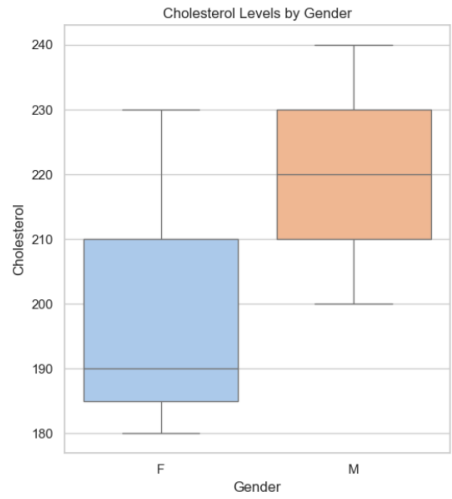
[11]:
```
def plotshow(data):
    # Set the plotting style
    sns.set(style="whitegrid")
    plt.figure(figsize=(16, 6))

    # Scatter Plot: Age vs Cholesterol
    plt.subplot(1, 3, 1)
    sns.scatterplot(data=df, x='Age', y='Cholesterol', hue='Gender')
    plt.title("Age vs Cholesterol")

    # Histogram: Heart Rate distribution
    plt.subplot(1, 3, 2)
    sns.histplot(df['Heart_Rate'].dropna(), kde=True, bins=5, color='orange')
    plt.title("Heart Rate Distribution")

    # Boxplot: Cholesterol by Gender
    plt.subplot(1, 3, 3)
    sns.boxplot(data=df, x='Gender', y='Cholesterol', palette='pastel')
    plt.title("Cholesterol Levels by Gender")

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

[12]:
```
plotshow(data)
```

```
C:\Users\Keshav Barawal\AppData\Local\Temp\ipykernel_2124\3806559445.py:18: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  sns.boxplot(data=df, x='Gender', y='Cholesterol', palette='pastel')
```







4) Write Python Programme to implement linear regression.

[1]:
```
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt
import seaborn as sns
```

[2]:
```
data = pd.read_csv("medical.csv")
```

[6]:
```
df = pd.DataFrame(data)
```

[7]:
```
# Drop rows with missing Age or Cholesterol for regression
df_clean = df[['Age', 'Cholesterol']].dropna()
```

[8]:
```
# Split into features and target
X = df_clean[['Age']]  # feature
y = df_clean['Cholesterol']  # target
```

[9]:
```

# Split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

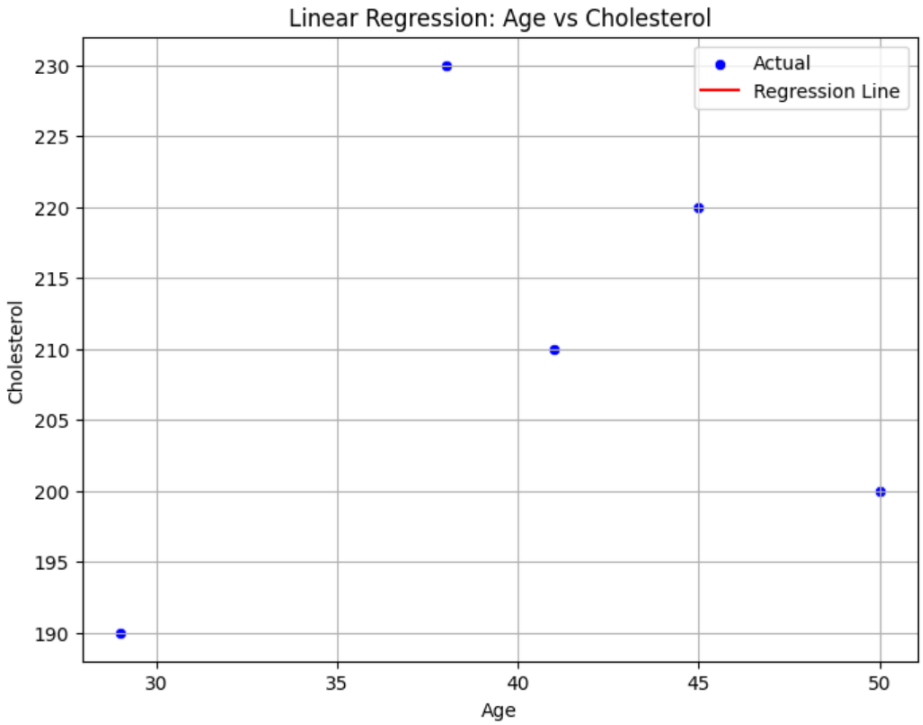
[10]:
```
# Linear Regression Model
model = LinearRegression()
model.fit(X_train, y_train)
```

[11]:
```
# Predictions
y_pred = model.predict(X_test)
```

[12]:
```
# Evaluation
print(f"Mean Squared Error: {mean_squared_error(y_test, y_pred):.2f}")
print(f"R² Score: {r2_score(y_test, y_pred):.2f}")
```

```
Mean Squared Error: 113.78
R² Score: nan
C:\MY_PC\4thsem\Dataminginglab\dm_env\Lib\site-packages\sklearn\metrics\_regression.py:1266: Undef
less than two samples.
  warnings.warn(msg, UndefinedMetricWarning)
```

[13]:
```
# Plotting
plt.figure(figsize=(8, 6))
sns.scatterplot(x='Age', y='Cholesterol', data=df_clean, color='blue', label='Actual')
plt.plot(X_test, y_pred, color='red', label='Regression Line')
plt.title('Linear Regression: Age vs Cholesterol')
plt.xlabel('Age')
plt.ylabel('Cholesterol')
plt.legend()
plt.grid(True)
plt.show()
```



5) Write a python programme to implement logistic regression.

[1]:
```
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
```

[2]:
```
data = pd.read_csv("medical.csv")
```

[3]:
```
df = pd.DataFrame(data)
```

[4]:
```
# Create target column: 1 if Diagnosis is Diabetes, else 0
df['Diabetes'] = df['Diagnosis'].apply(lambda x: 1 if x == 'Diabetes' else 0)
```

[5]:
```

# Keep only rows where we have no missing values for Age and Cholesterol
df_clean = df[['Age', 'Cholesterol', 'Diabetes']].dropna()
```

[6]:
```

# Features and target
X = df_clean[['Age', 'Cholesterol']]
y = df_clean['Diabetes']
```

[7]:
```
# Split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

[8]:
```
# Logistic Regression model
logreg = LogisticRegression()
logreg.fit(X_train, y_train)
```

[8]:
```
LogisticRegression
LogisticRegression()
```

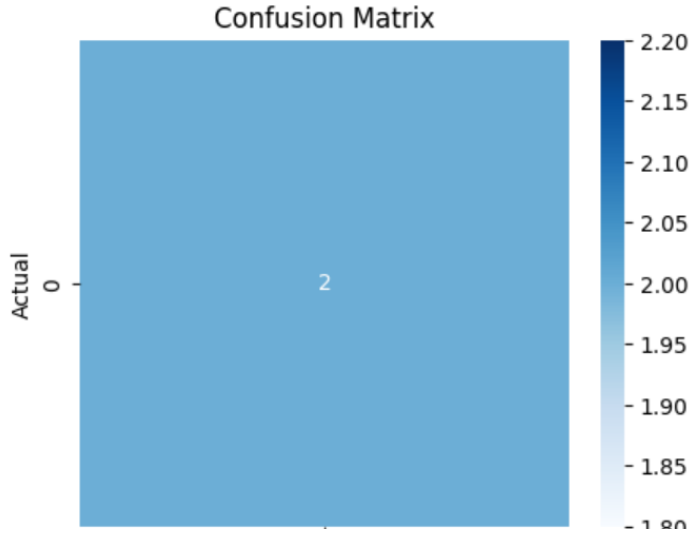
[9]:
```
# Predict
y_pred = logreg.predict(X_test)
```

[10]:
```
# Evaluation
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

```
Confusion Matrix:
 [[2]]

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         2

    accuracy                           1.00         2
   macro avg       1.00      1.00      1.00         2
weighted avg       1.00      1.00      1.00         2
```

[11]:
```
# Optional: Heatmap of confusion matrix
plt.figure(figsize=(5, 4))
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt='d', cmap='Blues')
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```

```
C:\MY_PC\4thsem\Datamininglab\dm_env\Lib\site-packages\sklearn\metrics\_classification.py:407: UserWarning: A single l
_pred'. For the confusion matrix to have the correct shape, use the 'labels' parameter to pass all known labels.
  warnings.warn(
```



6) Write python programme to implement decision tree.

[1]:
```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, accuracy_score
import matplotlib.pyplot as plt
import seaborn as sns
```

[2]:
```
data = pd.read_csv("medical.csv")
```

[3]:
```
df = pd.DataFrame(data)
```

[4]:
```
# Target variable: 1 for Diabetes, 0 otherwise
df['Diabetes'] = df['Diagnosis'].apply(lambda x: 1 if x == 'Diabetes' else 0)
```

[5]:
```
# Clean data: drop rows with missing values in features
df_clean = df[['Age', 'Cholesterol', 'Heart_Rate', 'Diabetes']].dropna()
```

[6]:
```
# Features and target
X = df_clean[['Age', 'Cholesterol', 'Heart_Rate']]
y = df_clean['Diabetes']
```

[7]:
```
# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1)
```

[8]:
```
# Create and train decision tree model
clf = DecisionTreeClassifier(criterion='entropy', random_state=1)
clf.fit(X_train, y_train)
```

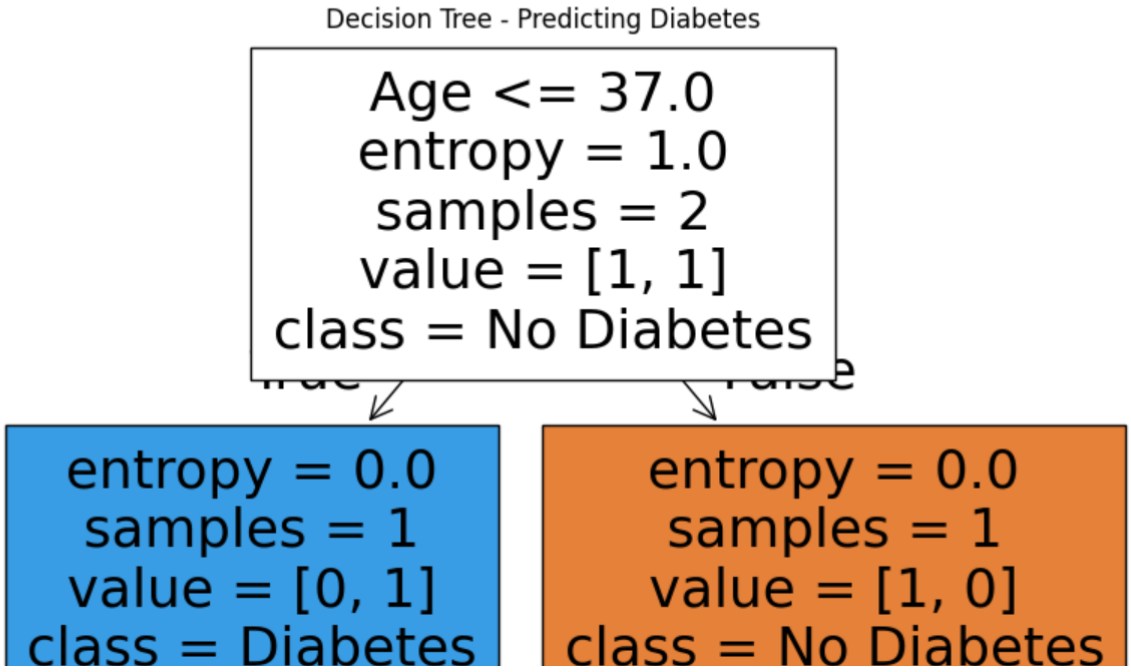
[8]:
```
DecisionTreeClassifier(criterion='entropy', random_state=1)
```

[9]:
```
# Predict and evaluate
y_pred = clf.predict(X_test)
```

[10]:
```
print("Accuracy Score:", accuracy_score(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

```
Accuracy Score: 1.0

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         2

    accuracy                           1.00         2
   macro avg       1.00      1.00      1.00         2
weighted avg       1.00      1.00      1.00         2
```

[11]:
```
# Plot the decision tree
plt.figure(figsize=(10, 6))
plot_tree(clf, feature_names=['Age', 'Cholesterol', 'Heart_Rate'], class_names=['No Diabetes', 'Diabetes'], filled=True)
plt.title("Decision Tree - Predicting Diabetes")
plt.show()
```



7) Write python programme to implement Ensemble Techniques.

[1]:
```
import pandas as pd
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier, VotingClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report
from sklearn.preprocessing import LabelEncoder
import numpy as np
```

[2]:
```
data = pd.read_csv("medical.csv")
```

[15]:
```
df = pd.DataFrame(data)
```

[16]:
```
# Create target column: 1 if Diagnosis is 'Diabetes', else 0
df['Diabetes'] = df['Diagnosis'].apply(lambda x: 1 if x == 'Diabetes' else 0)
```

[17]:
```
# Drop rows with missing values in necessary columns
df_clean = df[['Age', 'Gender', 'Cholesterol', 'Heart_Rate', 'Diabetes']].dropna()
```

[14]:
```
# Encode Gender
le = LabelEncoder()
df_clean['Gender'] = le.fit_transform(df_clean['Gender'])  # F=0, M=1
```

[13]:
```
# Features and Target
X = df_clean[['Age', 'Gender', 'Cholesterol', 'Heart_Rate']]
y = df_clean['Diabetes']
```

[12]:
```
# Split dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

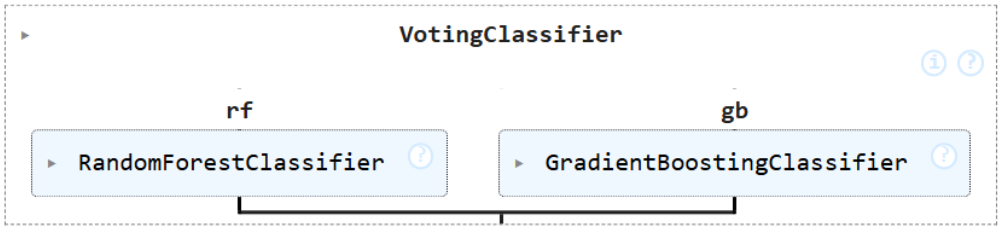
[11]:
```
# 1. Random Forest
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
```

[19]:
```
# 2. Gradient Boosting
gb_model = GradientBoostingClassifier(n_estimators=100, random_state=42)
```

[21]:
```
# 3. Voting Classifier (Hard Voting)
voting_clf = VotingClassifier(estimators=[
    ('rf', rf_model),
    ('gb', gb_model)
], voting='hard')
```

[22]:
```
# Train all models
rf_model.fit(X_train, y_train)
gb_model.fit(X_train, y_train)
voting_clf.fit(X_train, y_train)
```

[22]:



[23]:
```
# Predictions
rf_pred = rf_model.predict(X_test)
gb_pred = gb_model.predict(X_test)
voting_pred = voting_clf.predict(X_test)
```

[24]:

```
# Results
print("Random Forest Accuracy:", accuracy_score(y_test, rf_pred))
print("Gradient Boosting Accuracy:", accuracy_score(y_test, gb_pred))
print("Voting Classifier Accuracy:", accuracy_score(y_test, voting_pred))
```

```
Random Forest Accuracy: 0.5
Gradient Boosting Accuracy: 0.5
Voting Classifier Accuracy: 0.5
```

[25]:

```
print("\nVoting Classifier Classification Report:\n", classification_report(y_test, voting_pred))
```

```
Voting Classifier Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.50      0.67         2
           1       0.00      0.00      0.00         0

    accuracy                           0.50         2
   macro avg       0.50      0.25      0.33         2
weighted avg       1.00      0.50      0.67         2
```

8) Write a python program to implement clustering.

[1]:
```
import pandas as pd
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
import seaborn as sns
```

[2]:
```
data = pd.read_csv("medical.csv")
```

[3]:
```
df = pd.DataFrame(data)
```

[4]:
```
# Select features for clustering and handle missing values
df_cluster = df[['Age', 'Cholesterol', 'Heart_Rate']].dropna()
```

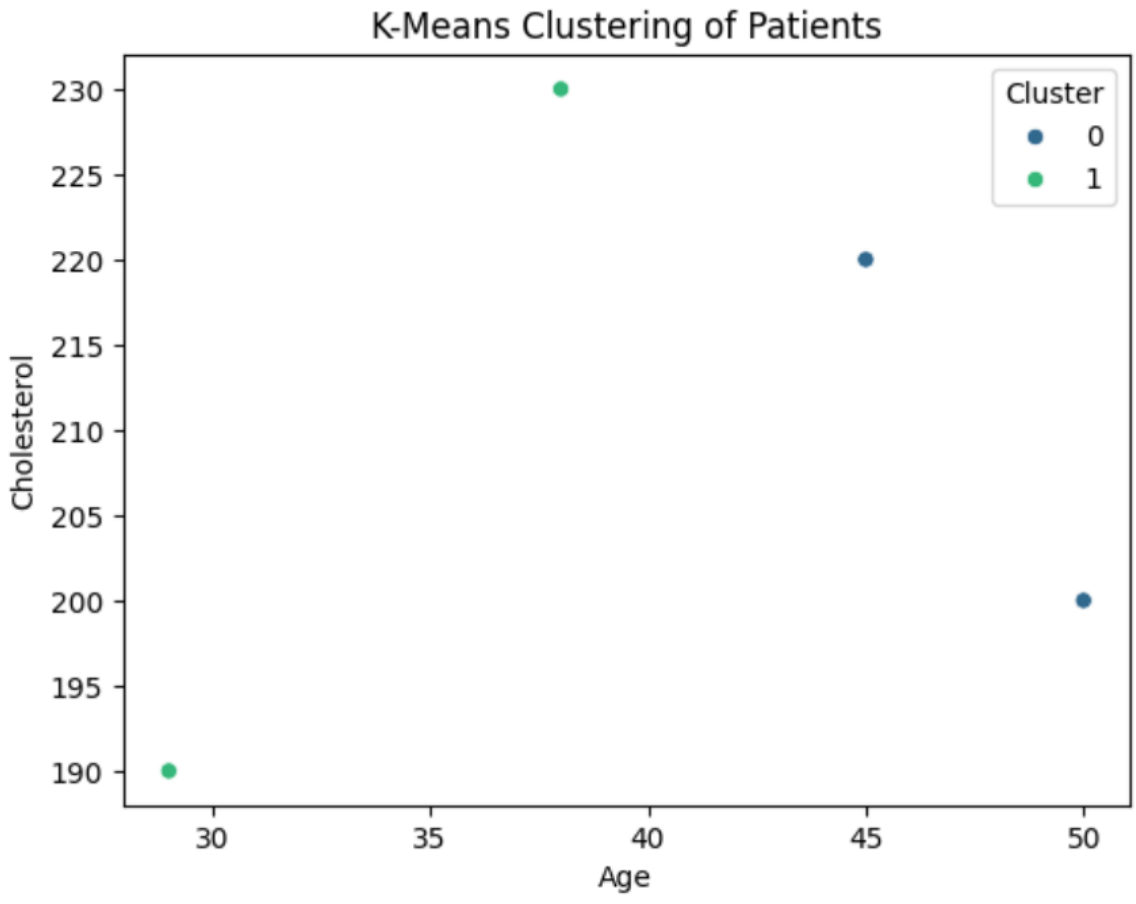
[5]:
```
# Standardize the features
scaler = StandardScaler()
scaled_features = scaler.fit_transform(df_cluster)
```

[7]:
```
# Apply KMeans clustering
kmeans = KMeans(n_clusters=2, random_state=42)
df_cluster['Cluster'] = kmeans.fit_predict(scaled_features)
```

[8]:
```
# Print clustered data
print("Clustered Data:\n")
print(df_cluster)
```

```
Clustered Data:

    Age  Cholesterol  Heart_Rate  Cluster
0  29.0        190.0        78.0        1
1  45.0        220.0        85.0        0
5  50.0        200.0        80.0        0
6  38.0        230.0        76.0        1
```

[9]:
```
# Optional: Visualize the clusters
sns.scatterplot(x='Age', y='Cholesterol', hue='Cluster', data=df_cluster, palette='v
plt.title("K-Means Clustering of Patients")
plt.xlabel("Age")
plt.ylabel("Cholesterol")
plt.show()
```



9) Write a python programme to implement SVM.

[1]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import classification_report, confusion_matrix
```

[2]:
```
data = pd.read_csv("medical.csv")
```

[3]:
```
df = pd.DataFrame(data)
```

[5]:
```
# Create binary labels: 1 if Diagnosis is Diabetes, else 0
df['Target'] = df['Diagnosis'].apply(lambda x: 1 if x == 'Diabetes' else 0)
```

[6]:
```
# Select features and target, dropping rows with missing values
features = df[['Age', 'Cholesterol', 'Heart_Rate']]
df_model = pd.concat([features, df['Target']], axis=1).dropna()
```

[7]:
```
X = df_model[['Age', 'Cholesterol', 'Heart_Rate']]
y = df_model['Target']
```

[8]:
```
# Split dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

[9]:
```
# Standardize the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

[10]:
```
# Train SVM classifier
svm_model = SVC(kernel='linear')  # You can try 'rbf', 'poly' too
svm_model.fit(X_train_scaled, y_train)
```

[10]:
```
SVC
SVC(kernel='linear')
```

[11]:
```
# Predict
y_pred = svm_model.predict(X_test_scaled)
```

[12]:
```
# Evaluation
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

```
Confusion Matrix:
 [[2]]

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         2

    accuracy                           1.00         2
   macro avg       1.00      1.00      1.00         2
weighted avg       1.00      1.00      1.00         2
```

10) Write a python programme to implement K-nn.

[1]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, confusion_matrix
```

[2]:
```
data = pd.read_csv("medical.csv")
```

[3]:
```
df = pd.DataFrame(data)
```

[5]:
```
# Create binary target: 1 if Diabetes, else 0
df['Target'] = df['Diagnosis'].apply(lambda x: 1 if x == 'Diabetes' else 0)
```

[6]:
```
# Use selected features and drop missing values
features = df[['Age', 'Cholesterol', 'Heart_Rate']]
df_model = pd.concat([features, df['Target']], axis=1).dropna()
```

[7]:
```
X = df_model[['Age', 'Cholesterol', 'Heart_Rate']]
y = df_model['Target']
```

[8]:
```
# Split data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

[9]:
```
# Scale features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

[12]:
```
k = min(3, len(X_train))  # use 3 or less based on available training samples
knn = KNeighborsClassifier(n_neighbors=k)
```

[13]:
```
# Ensure k <= number of training samples
k = min(3, len(X_train))
knn = KNeighborsClassifier(n_neighbors=k)
knn.fit(X_train_scaled, y_train)

# Predictions
y_pred = knn.predict(X_test_scaled)
```

[15]:
```
# Evaluation
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

```
Confusion Matrix:
 [[2]]

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         2

    accuracy                           1.00         2
   macro avg       1.00      1.00      1.00         2
weighted avg       1.00      1.00      1.00         2
```