Data preparation: Download heart dataset from following link.
https://www.kaggle.com/zhaoyingzhu/heartcsv Perform followi
operation on
given dataset.
a) Find Shape of Data
b) Find Missing Values
c) Find data type of each column
d) Finding out Zero's
e) Find Mean age of patients
f) Now extract only Age, Sex, ChestPain, RestBP, Chol.
Randomly divide dataset in training (75%) and testing (25%). Th
the diagnosis
test I predicted 100 report as COVID positive, but only 45 of th
were actually
positive. Total 50 people in my sample were actually COVID pos
have total
500 samples. Create confusion matrix based on above data and
I. Accuracy
II. Precision
III. Recall
IV. F-1 score
more theory about this topic
more information about theory

Data Preparation: Heart Disease Dataset

To begin with, let's walk through the necessary steps involved in data preparation and analy the heart disease dataset, followed by a deeper understanding of the confusion matrix met (Accuracy, Precision, Recall, and F1 score) in the context of a classification task.

1. **Find Shape of Data**

In Python (using pandas), the shape of a dataset refers to the number of rows and columns shape can be found using:

python

```python
import pandas as pd # Load the dataset df = pd.read_csv('heart.csv') # Make sure
dataset is downloaded and in the correct path # Get the shape of the dataset df.s
```

This will return a tuple of the form `(number of rows, number of columns)`.

2. **Find Missing Values**

Checking for missing values is essential to understand the quality of the data and handle th properly (either by filling or removing them).

python

```python
# Check for missing values df.isnull().sum()
```

This will show how many missing values exist in each column. If there are any, they can be using techniques like imputation (filling missing values with mean, median, or mode) or by rows or columns.

3. **Find Data Type of Each Column**

To check the data type of each column, which is important for understanding how each fea represented and whether any type conversion is needed:

```python
# Get the data types of each column df.dtypes
```

This will show the data type of each column (e.g., `int64`, `float64`, `object` for strings). En numeric features are of numeric types and categorical features are of type `object` or `cate`

4. **Finding Out Zero's**

To find columns with zero values (which may not make sense for certain features like age, b pressure, etc.), you can check if any feature contains zero values:

```python
# Check for zero values in the dataset df[df == 0].count()
```

You can also analyze specific columns by doing:

```python
# Find zero values in a specific column, e.g., 'Chol' df[df['Chol'] == 0]
```

This could help identify if zero values are valid or if they need to be corrected.

5. **Find Mean Age of Patients**

The average or mean age of patients can be calculated as follows:

```python
# Find the mean of the 'Age' column mean_age = df['Age'].mean() print("Mean Age:'
mean_age)
```

This gives the average age of the patients in the dataset.

6. **Extract Selected Features (Age, Sex, ChestPain, RestBP, Chol)**

You can select only a subset of features to work with:

```python
# Extract the selected columns selected_columns = df[['Age', 'Sex', 'ChestPain',
'RestBP', 'Chol']] print(selected_columns.head())
```

This will give you a new dataframe containing just the features you want.

7. **Split Dataset into Training and Testing Data**

Randomly split the dataset into training (75%) and testing (25%) sets. This is important for the model performance on unseen data.

python

```
from sklearn.model_selection import train_test_split # Split data into features (
target (y) X = df.drop(columns=['Target']) # assuming 'Target' is the diagnosis c
= df['Target'] # Split data into training and testing sets X_train, X_test, y_tra
y_test = train_test_split(X, y, test_size=0.25, random_state=42)
```

8. **Confusion Matrix and Performance Metrics**

You provided a diagnostic test scenario for COVID detection. Let's interpret this scenario ar use a confusion matrix to evaluate the model:

- **Predicted COVID Positive = 100**
- **True COVID Positive = 45**
- **Total actual COVID Positive = 50**
- **Total samples = 500**

From this information, you can create the confusion matrix and compute various performar metrics.

Confusion Matrix

A confusion matrix has the following structure:

	Predicted Positive	**Predicted Negative**
Actual Positive	True Positive (TP)	False Negative (FN)
Actual Negative	False Positive (FP)	True Negative (TN)

Given the scenario:

- **True Positives (TP)** = 45 (COVID positive and predicted positive)
- **False Negatives (FN)** = 5 (COVID positive but predicted negative, i.e., 50 total actual p minus 45 TP)
- **False Positives (FP)** = 55 (COVID negative but predicted positive, i.e., 100 predicted po minus 45 TP)
- **True Negatives (TN)** = 400 (the remaining samples, as there are 500 samples in total)

You can create the confusion matrix and calculate the metrics as follows:

python

```
from sklearn.metrics import confusion_matrix # Assuming the model prediction for
test set is stored in y_pred y_pred = [1] * 100 + [0] * 400 # Predicted positives
100, the rest negatives y_true = [1] * 50 + [0] * 450 # Actual 50 positives, rest
negatives # Create confusion matrix cm = confusion_matrix(y_true, y_pred)
print("Confusion Matrix:\n", cm)
```

Metrics:

1. **Accuracy**: The accuracy measures how often the model is correct. It is calculated as:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

2. **Precision**: Precision measures the proportion of positive predictions that are actually c calculated as:

$$\text{Precision} = \frac{TP}{TP + FP}$$

3. **Recall (Sensitivity)**: Recall measures the proportion of actual positives that are correctl identified. It is calculated as:

$$\text{Recall} = \frac{TP}{TP + FN}$$

4. **F1 Score**: The F1 Score is the harmonic mean of Precision and Recall, and is used when a balance between Precision and Recall. It is calculated as:

$$\text{F1 Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

Using Python, you can calculate these metrics using `sklearn.metrics`:

python

```python
from sklearn.metrics import precision_score, recall_score, f1_score, accuracy_scc
Calculate accuracy accuracy = accuracy_score(y_true, y_pred) # Calculate precisic
precision = precision_score(y_true, y_pred) # Calculate recall recall =
recall_score(y_true, y_pred) # Calculate F1 score f1 = f1_score(y_true, y_pred)
print(f"Accuracy: {accuracy}") print(f"Precision: {precision}") print(f"Recall:
{recall}") print(f"F1 Score: {f1}")
```

Theory Behind the Confusion Matrix Metrics:

1. **Accuracy**: Accuracy is the simplest evaluation metric. It works well when the dataset is but may not be reliable in cases where the dataset is imbalanced (e.g., 95% negatives, positives).

2. **Precision**: Precision answers the question: *Of all the instances where the model predicte how many were actually positive?* It is particularly useful when the cost of a false positi (e.g., falsely diagnosing a patient with a disease).

3. **Recall (Sensitivity)**: Recall answers the question: *Of all the actual positive instances, hov were correctly identified by the model?* It is important when the cost of a false negative (e.g., failing to identify a disease in a patient who actually has it).

4. **F1 Score**: The F1 score is the harmonic mean of precision and recall. It is useful when y balance between precision and recall, especially in imbalanced datasets. A high F1 sco indicates both high precision and high recall.

Conclusion

By following these steps, you'll be able to prepare the data, perform basic exploratory data (EDA), and calculate key performance metrics using the confusion matrix. These steps are fundamental when assessing the performance of machine learning classification models.

more theory

Deep Dive: Theory Behind Data Preparation and Evaluation Metrics

To gain a clearer understanding of the theory behind data preparation and evaluation metr accuracy, precision, recall, and F1 score, let's break down each step and concept in more de

Data Preparation and Preprocessing

Data preparation is a crucial step in any data analysis or machine learning task, especially ir world scenarios where data is often messy and incomplete. Proper preprocessing is essenti making meaningful predictions and ensuring that the model can learn effectively from the are some key concepts:

1. **Shape of Data**

The shape of a dataset refers to its dimensions, specifically the number of rows (observatio columns (features). For example:

- Rows represent individual samples or records (e.g., individual patient data).
- Columns represent different features (e.g., age, sex, blood pressure, cholesterol, etc.).

Knowing the shape helps to understand the scale of the data and how much information y work with.

2. **Missing Values**

Missing data is common in many datasets, and it can arise due to a variety of reasons:

- Data not collected or recorded
- Errors in data entry
- Non-response in surveys or studies

Dealing with missing values is important because many machine learning algorithms can't missing data directly. Common strategies include:

- **Imputation**: Filling missing values with a calculated value, such as the mean, median, c the column.
- **Removal**: Deleting rows or columns that contain too many missing values (if they are r critical).
- **Prediction**: Using models to predict missing values based on other features.

3. **Data Types**

Different types of data require different handling:

- **Numerical data** (e.g., age, blood pressure, cholesterol levels) can be used directly in m models, though they may require scaling or normalization.
- **Categorical data** (e.g., sex, chest pain type) may need to be encoded into numerical fo techniques like one-hot encoding or label encoding) to be usable in most machine lea algorithms.
- **Ordinal data** (e.g., levels of severity) has a natural order and may require special handl assigning numerical values based on the order.

4. **Zero Values**

In some contexts, zeros might be valid values (e.g., a patient with zero cholesterol). Howeve contexts, zeros may indicate missing data or errors (e.g., age or blood pressure values of ze be unrealistic). Identifying zeros is important to understand whether they are meaningful o be corrected.

5. **Feature Selection**

Extracting relevant features (columns) is crucial for the efficiency and accuracy of your mod selecting only the most important features (e.g., age, sex, chest pain type), you can reduce complexity of your model and improve its interpretability.

Confusion Matrix and Performance Metrics

Once you have a trained model, the next step is to evaluate its performance. The confusion provides a detailed view of how well the model is performing, and from that, we can derive important metrics that help in model evaluation.

Confusion Matrix

A **confusion matrix** is a table that shows the performance of a classification algorithm by c the actual versus predicted values. It breaks down the predictions into four categories:

- **True Positives (TP)**: These are cases where the model correctly predicts a positive class predicting a patient has heart disease when they actually do).
- **False Positives (FP)**: These are cases where the model incorrectly predicts a positive cla predicting a patient has heart disease when they do not).
- **True Negatives (TN)**: These are cases where the model correctly predicts a negative cla predicting a patient does not have heart disease when they really don't).
- **False Negatives (FN)**: These are cases where the model incorrectly predicts a negative predicting a patient does not have heart disease when they actually do).

The confusion matrix helps to break down the performance of the model into more specifi categories, rather than just providing an overall accuracy.

Performance Metrics

1. Accuracy

Accuracy is the most straightforward evaluation metric. It is the proportion of correct predi (both positive and negative) out of all predictions made.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

- **Pros**: Simple to calculate and understand.
- **Cons**: It may be misleading when dealing with imbalanced datasets (e.g., in fraud dete rare disease detection, where the majority of cases are negative).

2. Precision (Positive Predictive Value)

Precision measures how many of the instances predicted as positive are actually positive. It on the performance of the model in predicting the positive class.

$$\text{Precision} = \frac{TP}{TP + FP}$$

- **High precision** means that when the model predicts a positive class, it is usually correc
- **Use case**: Precision is particularly important when false positives are costly. For exampl medical diagnoses, a false positive may lead to unnecessary treatments or tests.

3. Recall (Sensitivity or True Positive Rate)

Recall measures how many of the actual positive instances were correctly predicted by the focuses on the model's ability to identify positive cases.

$$\text{Recall} = \frac{TP}{TP + FN}$$

- **High recall** means that the model correctly identifies most of the positive cases.
- **Use case**: Recall is especially important when false negatives are costly. For example, ir detection, failing to detect cancer (false negative) could have severe consequences.

4. F1 Score

The F1 score is the harmonic mean of precision and recall. It provides a balance between tl metrics, especially when there is an uneven class distribution (e.g., when one class is much prevalent than the other).

$$\text{F1 Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

- **Use case**: The F1 score is useful when you need a balance between precision and recal especially useful when the dataset has imbalanced classes (e.g., a rare disease with onl percentage of positive cases).

Other Evaluation Metrics

Besides the standard metrics discussed above, there are other evaluation metrics that can b valuable depending on the task and context:

1. **Specificity (True Negative Rate)**: Measures how well the model identifies negative cas complement of recall:

$$\text{Specificity} = \frac{TN}{TN + FP}$$

2. **Area Under the ROC Curve (AUC-ROC)**: The ROC curve is a graphical representation o classifier's performance, plotting the true positive rate (recall) against the false positive area under the ROC curve (AUC) provides an aggregate measure of the classifier's perf across all possible thresholds.
3. **Log Loss (Cross-Entropy Loss)**: This metric evaluates the probability output of a classif model rather than just the final predicted class. It is used primarily in probabilistic mod logistic regression).
4. **Matthews Correlation Coefficient (MCC)**: MCC is a balanced measure that takes into a true and false positives and negatives. It ranges from -1 (perfect inverse prediction) to (perfect prediction), with 0 indicating random prediction.

Conclusion

- **Data Preparation**: The quality of your data largely determines the effectiveness of you Proper preprocessing, handling of missing values, feature selection, and understanding shape and types of data are all foundational to building robust models.
- **Confusion Matrix and Evaluation Metrics**: The confusion matrix offers a detailed breal a model's performance, and metrics like accuracy, precision, recall, and F1 score help e different aspects of the model's effectiveness. Precision and recall are especially useful imbalanced datasets, where accuracy alone may not provide an accurate picture of mc performance.

Ultimately, choosing the right evaluation metrics depends on the problem at hand and the consequences of different types of errors. For example, in medical applications, you may pr recall to ensure that you detect as many positive cases as possible, even at the cost of som positives. In other cases, you might prioritize precision to minimize the cost of false positive

Download temperature data from below link.
https://www.kaggle.com/venky73/temperaturesofindia?select=temperatures.csv This data consists of temperatures of I averaging the temperatures of all places month wise. Temperat values are
recorded in CELSIUS
a. Apply Linear Regression using suitable library function and p the Monthwisetemperature.

b. Assessthe performance of regression models using MSE, MA
R-Square
metrics
c. Visualize simple regression model

Temperature Prediction Using Linear Regression

In this task, we will work with the temperature data of India, which includes monthly averag temperatures recorded in Celsius for various places across India. Our objective is to apply **L Regression** to predict the monthly temperature, assess the model's performance using vari evaluation metrics, and visualize the results.

Step-by-Step Breakdown

1. **Download the Dataset**
   - First, download the temperature dataset from the given Kaggle link.
   - The dataset likely contains columns like "Month" and "Temperature" (in Celsius), w temperature represents the average temperature for each month across various lo India.
2. **Data Loading and Preprocessing**
   - Load the dataset into a pandas DataFrame.
   - Preprocess the data (check for missing values, ensure correct data types, etc.).
   - Split the data into features (X) and target (Y), where the feature will be the month, target will be the temperature.
3. **Apply Linear Regression**
   - Use a linear regression model to predict the monthly temperature based on the g
   - Fit the model using a suitable library function (like `sklearn.linear_model.LinearRegression`).
4. **Assess Model Performance**
   - Evaluate the model performance using the following metrics:
     - **Mean Squared Error (MSE)**
     - **Mean Absolute Error (MAE)**
     - **R-Squared (R^2)**
5. **Visualize the Regression Model**
   - Plot the predicted values against the actual values.
   - Visualize the linear regression line over the data points to better understand how model fits the data.

Code Implementation

Let’s break it down into code.

Step 1: Data Loading and Preprocessing

First, make sure you've downloaded the dataset and have the necessary libraries installed:

bash

```bash
pip install pandas matplotlib scikit-learn
```

Now, let's load the data and inspect it:

python

```python
import pandas as pd # Load the dataset df = pd.read_csv('temperatures.csv') # Ins
the first few rows of the dataset print(df.head()) # Check for missing values anc
```

```
types print(df.info()) # Drop any rows with missing values (if present)
df.dropna(inplace=True)
```

Assuming the dataset has a "Month" and "Temperature" column, let's move on to applying regression model.

Step 2: Apply Linear Regression

We will use `scikit-learn` to create a linear regression model. The target variable (`y`) is the **Temperature**, and the feature (`X`) is the **Month**.

python

```
from sklearn.linear_model import LinearRegression from sklearn.model_selection in
train_test_split # Extract feature (X) and target variable (y) X = df[['Month']]
'Month' as the independent variable y = df['Temperature'] # 'Temperature' as the
dependent variable # Split the data into training and testing sets (80% training,
testing) X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42) # Initialize the linear regression model model = LinearRegressio
Fit the model on the training data model.fit(X_train, y_train) # Predict the
temperatures on the test set y_pred = model.predict(X_test)
```

Step 3: Model Evaluation

We will use **Mean Squared Error (MSE)**, **Mean Absolute Error (MAE)**, and **R-Squared (R^2)** t the model's performance.

python

```
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score #
Calculate Mean Squared Error (MSE) mse = mean_squared_error(y_test, y_pred) # Cal
Mean Absolute Error (MAE) mae = mean_absolute_error(y_test, y_pred) # Calculate R
Squared (R²) r_squared = r2_score(y_test, y_pred) # Print the performance metrics
print(f"Mean Squared Error (MSE): {mse}") print(f"Mean Absolute Error (MAE): {mae
print(f"R-Squared (R²): {r_squared}")
```

- **MSE**: Measures the average of the squares of the errors—that is, the average squared between the actual and predicted values. Lower values are better.
- **MAE**: Measures the average magnitude of errors in a set of predictions, without consic their direction. Like MSE, lower values are better.
- **R^2 (R-Squared)**: This is the proportion of the variance in the dependent variable that is predictable from the independent variable. R^2 values range from 0 to 1, where 1 indica perfect fit.

Step 4: Visualize the Simple Regression Model

Now, let's visualize the fitted linear regression line over the actual data points. We'll use **Matplotlib** to plot the data.

python

```
import matplotlib.pyplot as plt # Plot the actual data points plt.scatter(X_test,
y_test, color='blue', label='Actual Data') # Plot the regression line (Predicted
plt.plot(X_test, y_pred, color='red', linewidth=2, label='Regression Line') # Set
and labels plt.title('Linear Regression: Temperature Prediction') plt.xlabel('Mor
```

```
plt.ylabel('Temperature (Celsius)') # Show the legend plt.legend() # Show the plo
plt.show()
```

In this plot:

- **Blue dots** represent the actual data points (i.e., the real temperatures for each month).
- **Red line** represents the linear regression model's predictions.

Summary of Results

After running the code:

1. **Model Performance**: You'll get values for MSE, MAE, and R^2, which help assess the per of the model.
   - **MSE and MAE** indicate the average error in the model's predictions.
   - **R^2** tells us how well the model explains the variance in the temperature data. A hig value suggests the model fits the data well.
2. **Visualizing the Model**: The regression plot will show how well the linear model fits the temperature data. A straight line indicates a linear relationship between the month anc temperature.

Additional Insights

- **Data Trend**: In the real-world, temperature data typically shows some seasonality or cy behavior (e.g., higher temperatures in summer months and lower temperatures in wint months). A simple linear regression might not fully capture these seasonal patterns, wl require more advanced models like **Polynomial Regression** or **Time Series Models** (e.g SARIMA).
- **Model Evaluation**: If the model's performance metrics (MSE, MAE) are high, or the R^2 s low, this suggests that a simple linear model may not be the best choice for this datas could explore other regression techniques or even feature engineering (e.g., encoding as sine/cosine values to capture seasonal behavior).