```
#**importing libraries**
import numpy as np
import pandas as pd
from sklearn.metrics import accuracy_score


#**importing datsets**
import pandas as pd
# Load each CSV file separately
X_train = pd.read_csv('/content/X_Train_Data_Input.csv')
Y_train = pd.read_csv('/content/Y_Train_Data_Target.csv')
X_test = pd.read_csv('/content/X_Test_Data_Input.csv')
Y_test = pd.read_csv('/content/Y_Test_Data_Target.csv')


#**Concating Training datasets**
data=pd.concat([X_train, Y_train], axis=1)
data.head()
```

	ID	Column0	Column1	Column2	Column3	Column4	Co
0	ad1a67e4cbddc767a3456b0d94299b9e	2.0	2495	3726.0	0.678139	0.701403	-0.0
1	7246d2f76ac0c217ec25e72ea5f014cb	0.0	2495	3454.0	0.452580	0.701403	-0.0
2	22ba388e7dd14c13342c49e75fc29dda	2.0	2495	4543.0	-1.577453	-1.429540	-0.0
3	59f9b981472d97342587fb3e6392aeb1	0.0	211	59.0	NaN	NaN	
4	f6317cf7ecf126859804eddff279aead	0.0	718	950.0	-2.028572	-1.855728	

5 rows × 25 columns

```
#**Concating Testing datasets**
data=pd.concat([X_test, Y_test], axis=1)
data.head()
```

	ID	target	ID	target
0	07cf2025382f6325b316e128b1b90999	0	07cf2025382f6325b316e128b1b90999	0
1	eb972eb3a1f8d0d1a13f45e7c07d37d4	0	eb972eb3a1f8d0d1a13f45e7c07d37d4	0
2	ee35e164b3ddc25a9f40243b81ad290d	0	ee35e164b3ddc25a9f40243b81ad290d	0
3	28229ccd7bad7dd83324a4175a7e0531	0	28229ccd7bad7dd83324a4175a7e0531	0
4	2f94873da2c332d28f111742818e0fbb	0	2f94873da2c332d28f111742818e0fbb	0

```
#**Precision for datasets**
from sklearn.metrics import precision_score
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
model = LogisticRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
precision = precision_score(y_test, y_pred)
print(precision)
```

```
0.7970500030601628
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:469: Converger
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

```
#**Recall for datasets**
from sklearn.metrics import recall_score
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
model = LogisticRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
recall = recall_score(y_test, y_pred)
print(recall)
```

```
0.8827955531453362
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:469: Converger
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

```
#**F1 score for datasets**
from sklearn.metrics import f1_score
f1 = f1_score(y_test, y_pred)
print(f1)
```

```
0.8377343925894953
```

```
#**LOGISTIC REGRESSION Training & Testing accuracy.**
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
```

```
from sklearn.metrics import accuracy_score

# Assuming X and y are defined
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

model = LogisticRegression()
model.fit(X_train, y_train)

y_pred_train = model.predict(X_train)
y_pred_test = model.predict(X_test)

train_accuracy = accuracy_score(y_train, y_pred_train)
test_accuracy = accuracy_score(y_test, y_pred_test)

print(f"Training accuracy: {train_accuracy}")
print(f"Testing accuracy: {test_accuracy}")
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:469: Converger
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
Training accuracy: 0.9680292816817544
Testing accuracy: 0.9678717672756915
```

```
#**Accuracy, Precision, Recall, F1 score for LOGISTIC REGRESSION**
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from sklearn.impute import SimpleImputer

# Load the datasets
X_train = pd.read_csv('/content/X_Train_Data_Input.csv')
Y_train = pd.read_csv('/content/Y_Train_Data_Target.csv')
X_test = pd.read_csv('/content/X_Test_Data_Input.csv')
Y_test = pd.read_csv('/content/Y_Test_Data_Target.csv')

# Check shapes of loaded dataframes to understand if there's a size mismatch
print("X_train shape:", X_train.shape)
print("Y_train shape:", Y_train.shape)
print("X_test shape:", X_test.shape)
print("Y_test shape:", Y_test.shape)

# Ensure Y_train and Y_test have the same number of rows as X_train and X_test respectively
# This might involve filtering or adjusting the target dataframes
```

```
# Example: If Y_train and Y_test have an 'ID' column that matches X_train and X_test, you ca
Y_train = Y_train[Y_train['ID'].isin(X_train['ID'])]
Y_test = Y_test[Y_test['ID'].isin(X_test['ID'])]

# Extract target variables
y_train = Y_train['target']
y_test = Y_test['target']

# Handle non-numeric columns (drop for now)
X_train = X_train.select_dtypes(include=['number'])
X_test = X_test.select_dtypes(include=['number'])

# Impute missing values with the mean
imputer = SimpleImputer(strategy='mean')
X_train = imputer.fit_transform(X_train)
X_test = imputer.transform(X_test)

# Scale the data
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Create and train the Logistic Regression model
model = LogisticRegression(random_state=42)
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Calculate metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

# Print metrics
print(f"Accuracy: {accuracy}")
print(f"Precision: {precision}")
print(f"Recall: {recall}")
print(f"F1 Score: {f1}")
```

```
X_train shape: (49019, 23)
Y_train shape: (299593, 2)
X_test shape: (44108, 23)
Y_test shape: (261712, 2)
Accuracy: 0.96830506937517
Precision: 0.8021762785636561
Recall: 0.8828742514970059
F1 Score: 0.8405929304446979
```

```
#**DECISION TREE CLASSIFIER - Training & Testing Accuracy**
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Assuming X and y are defined
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

model = DecisionTreeClassifier()
model.fit(X_train, y_train)

y_pred_train = model.predict(X_train)
y_pred_test = model.predict(X_test)

train_accuracy = accuracy_score(y_train, y_pred_train)
test_accuracy = accuracy_score(y_test, y_pred_test)

print(f"Training accuracy: {train_accuracy}")
print(f"Testing accuracy: {test_accuracy}")
```

```
Training accuracy: 0.9996959111997019
Testing accuracy: 0.9675915606870156
```

```
#**Accuracy, Precision, Recall, F1 score for DECISION TREE CLASSIFIER**
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.tree import DecisionTreeClassifier  # Import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from sklearn.impute import SimpleImputer

# Load the datasets
X_train = pd.read_csv('/content/X_Train_Data_Input.csv')
Y_train = pd.read_csv('/content/Y_Train_Data_Target.csv')
X_test = pd.read_csv('/content/X_Test_Data_Input.csv')
Y_test = pd.read_csv('/content/Y_Test_Data_Target.csv')

# Check shapes of loaded dataframes to understand if there's a size mismatch
print("X_train shape:", X_train.shape)
print("Y_train shape:", Y_train.shape)
print("X_test shape:", X_test.shape)
print("Y_test shape:", Y_test.shape)

# Ensure Y_train and Y_test have the same number of rows as X_train and X_test respectively
# This might involve filtering or adjusting the target dataframes
# Example: If Y_train and Y_test have an 'ID' column that matches X_train and X_test, you ca
Y_train = Y_train[Y_train['ID'].isin(X_train['ID'])]
Y_test = Y_test[Y_test['ID'].isin(X_test['ID'])]

# Extract target variables
y_train = Y_train['target']
```

```
y_test = Y_test['target']

# Handle non-numeric columns (drop for now)
X_train = X_train.select_dtypes(include=['number'])
X_test = X_test.select_dtypes(include=['number'])

# Impute missing values with the mean
imputer = SimpleImputer(strategy='mean')
X_train = imputer.fit_transform(X_train)
X_test = imputer.transform(X_test)

# Scale the data
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Create and train the Decision Tree Classifier model
model = DecisionTreeClassifier(random_state=42)  # Instantiate DecisionTreeClassifier
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Calculate metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

# Print metrics
print(f"Accuracy: {accuracy}")
print(f"Precision: {precision}")
print(f"Recall: {recall}")
print(f"F1 Score: {f1}")
```

```
X_train shape: (63693, 23)
Y_train shape: (389471, 2)
X_test shape: (58805, 23)
Y_test shape: (261712, 2)
Accuracy: 0.9673667205169628
Precision: 0.8266215253029223
Recall: 0.8306177260519249
F1 Score: 0.8286148075377333
```

```
#**STANDARD SCALER - Training & Testing Accuracy**
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.impute import SimpleImputer # Import the SimpleImputer module
```

```
# Load the datasets
X_train = pd.read_csv('/content/X_Train_Data_Input.csv')
Y_train = pd.read_csv('/content/Y_Train_Data_Target.csv')
X_test = pd.read_csv('/content/X_Test_Data_Input.csv')
Y_test = pd.read_csv('/content/Y_Test_Data_Target.csv')

# Combine features and target variables
X = pd.concat([X_train, X_test])
y = pd.concat([Y_train, Y_test])

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Check for non-numeric columns in X_train
non_numeric_cols = X_train.select_dtypes(exclude=['number']).columns

# Handle non-numeric columns (example: drop or encode)
X_train = X_train.drop(columns=non_numeric_cols)
X_test = X_test.drop(columns=non_numeric_cols)

# Remove the ID column from y_train and y_test
y_train = y_train['target']
y_test = y_test['target']

# Impute missing values using SimpleImputer
imputer = SimpleImputer(strategy='mean') # Create an imputer object with the strategy you wa
X_train = imputer.fit_transform(X_train) # Fit and transform on the training data
X_test = imputer.transform(X_test) # Transform the test data using the same imputer


# Scale the data
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Create and train the model
model = LogisticRegression()
model.fit(X_train, y_train)

# Make predictions
y_pred_train = model.predict(X_train)
y_pred_test = model.predict(X_test)

# Calculate accuracy
train_accuracy = accuracy_score(y_train, y_pred_train)
test_accuracy = accuracy_score(y_test, y_pred_test)

print(f"Training accuracy: {train_accuracy}")
print(f"Testing accuracy: {test_accuracy}")
```

```
Training accuracy: 0.968832539678749
Testing accuracy: 0.9685435761741232
```

```
#**STANDARD SCALER Precision, Recall, F1 score**
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from sklearn.impute import SimpleImputer # Import SimpleImputer to handle missing values

# Load the datasets
X_train = pd.read_csv('/content/X_Train_Data_Input.csv')
Y_train = pd.read_csv('/content/Y_Train_Data_Target.csv')
X_test = pd.read_csv('/content/X_Test_Data_Input.csv')
Y_test = pd.read_csv('/content/Y_Test_Data_Target.csv')

# Combine features and target variables
X = pd.concat([X_train, X_test])
y = pd.concat([Y_train, Y_test])

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Check for non-numeric columns in X_train
non_numeric_cols = X_train.select_dtypes(exclude=['number']).columns

# Handle non-numeric columns (example: drop or encode)
X_train = X_train.drop(columns=non_numeric_cols)
X_test = X_test.drop(columns=non_numeric_cols)

# Remove the ID column from y_train and y_test
y_train = y_train['target']
y_test = y_test['target']

# Impute missing values using SimpleImputer
imputer = SimpleImputer(strategy='mean') # Create an imputer object with the strategy you wa
X_train = imputer.fit_transform(X_train) # Fit and transform on the training data
X_test = imputer.transform(X_test) # Transform the test data using the same imputer

# Scale the data
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Create and train the model
model = LogisticRegression()
model.fit(X_train, y_train)

# Make predictions
```

```
y_pred = model.predict(X_test)

# Calculate metrics
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

# Print metrics
print(f"Precision: {precision}")
print(f"Recall: {recall}")
print(f"F1-score: {f1}")
```

```
Precision: 0.8028821648672443
Recall: 0.8813774861386642
F1-score: 0.8403006789524733
```

```
#**ANN & DECISION TREE CLASSIFIER - Training & Testing Accuracy**
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.impute import SimpleImputer
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

# Load the datasets
X_train = pd.read_csv('/content/X_Train_Data_Input.csv')
Y_train = pd.read_csv('/content/Y_Train_Data_Target.csv')
X_test = pd.read_csv('/content/X_Test_Data_Input.csv')
Y_test = pd.read_csv('/content/Y_Test_Data_Target.csv')

# Combine features and target variables
X = pd.concat([X_train, X_test])
y = pd.concat([Y_train, Y_test])

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Check for non-numeric columns in X_train
non_numeric_cols = X_train.select_dtypes(exclude=['number']).columns

# Handle non-numeric columns (example: drop or encode)
X_train = X_train.drop(columns=non_numeric_cols)
X_test = X_test.drop(columns=non_numeric_cols)

# Remove the ID column from y_train and y_test
y_train = y_train['target']
y_test = y_test['target']

# Impute missing values using SimpleImputer
```

```
imputer = SimpleImputer(strategy='mean')
X_train = imputer.fit_transform(X_train)
X_test = imputer.transform(X_test)

# Scale the data
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Decision Tree Classifier
dt_model = DecisionTreeClassifier()
dt_model.fit(X_train, y_train)
dt_y_pred_train = dt_model.predict(X_train)
dt_y_pred_test = dt_model.predict(X_test)
dt_train_accuracy = accuracy_score(y_train, dt_y_pred_train)
dt_test_accuracy = accuracy_score(y_test, dt_y_pred_test)

print(f"Decision Tree - Training accuracy: {dt_train_accuracy}")
print(f"Decision Tree - Testing accuracy: {dt_test_accuracy}")

# ANN
ann_model = Sequential()
ann_model.add(Dense(128, activation='relu', input_shape=(X_train.shape[1],)))
ann_model.add(Dense(64, activation='relu'))
ann_model.add(Dense(1, activation='sigmoid'))
ann_model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
ann_model.fit(X_train, y_train, epochs=10, batch_size=32)
_, ann_train_accuracy = ann_model.evaluate(X_train, y_train)
_, ann_test_accuracy = ann_model.evaluate(X_test, y_test)

print(f"ANN - Training accuracy: {ann_train_accuracy}")
print(f"ANN - Testing accuracy: {ann_test_accuracy}")
```

```
Decision Tree - Training accuracy: 0.999615511369878
Decision Tree - Testing accuracy: 0.9680898318280166
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: UserWarning:
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Epoch 1/10
26172/26172 ━━━━━━━━━━━━━━━━━━━━ 52s 2ms/step - accuracy: 0.9711 - loss: 0.0664
Epoch 2/10
26172/26172 ━━━━━━━━━━━━━━━━━━━━ 76s 2ms/step - accuracy: 0.9750 - loss: 0.0572
Epoch 3/10
26172/26172 ━━━━━━━━━━━━━━━━━━━━ 47s 2ms/step - accuracy: 0.9752 - loss: 0.0564
Epoch 4/10
26172/26172 ━━━━━━━━━━━━━━━━━━━━ 49s 2ms/step - accuracy: 0.9754 - loss: 0.0555
Epoch 5/10
26172/26172 ━━━━━━━━━━━━━━━━━━━━ 46s 2ms/step - accuracy: 0.9756 - loss: 0.0566
Epoch 6/10
26172/26172 ━━━━━━━━━━━━━━━━━━━━ 46s 2ms/step - accuracy: 0.9758 - loss: 0.0555
Epoch 7/10
26172/26172 ━━━━━━━━━━━━━━━━━━━━ 82s 2ms/step - accuracy: 0.9753 - loss: 0.0558
Epoch 8/10
26172/26172 ━━━━━━━━━━━━━━━━━━━━ 45s 2ms/step - accuracy: 0.9758 - loss: 0.0553
```

```
Epoch 9/10
26172/26172 ━━━━━━━━━━━━━━━━━━━━ 80s 2ms/step - accuracy: 0.9758 - loss: 0.0586
Epoch 10/10
26172/26172 ━━━━━━━━━━━━━━━━━━━━ 43s 2ms/step - accuracy: 0.9760 - loss: 0.0563
26172/26172 ━━━━━━━━━━━━━━━━━━━━ 32s 1ms/step - accuracy: 0.9762 - loss: 0.0549
6543/6543 ━━━━━━━━━━━━━━━━━━━━ 9s 1ms/step - accuracy: 0.9754 - loss: 0.0569
ANN - Training accuracy: 0.9761282801628113
ANN - Testing accuracy: 0.9754213690757751
```

```
#**ANN - Training & Testing Accuracy**
!pip install tensorflow

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
from sklearn.impute import SimpleImputer
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

# Load the datasets
X_train = pd.read_csv('/content/X_Train_Data_Input.csv')
Y_train = pd.read_csv('/content/Y_Train_Data_Target.csv')
X_test = pd.read_csv('/content/X_Test_Data_Input.csv')
Y_test = pd.read_csv('/content/Y_Test_Data_Target.csv')

# ---Ensure that X_train and Y_train have the same number of rows before concatenation---
X_train = X_train.iloc[:min(X_train.shape[0], Y_train.shape[0])]
Y_train = Y_train.iloc[:min(X_train.shape[0], Y_train.shape[0])]
X_test = X_test.iloc[:min(X_test.shape[0], Y_test.shape[0])]
Y_test = Y_test.iloc[:min(X_test.shape[0], Y_test.shape[0])]

# ---Extract the target variable from Y_train and Y_test---
y_train = Y_train['target']
y_test = Y_test['target']

# ---Now you can split the data using X_train, y_train---
# Split data into training and testing sets (if necessary, you might skip this step)
#X_train, X_test, y_train, y_test = train_test_split(X_train, y_train, test_size=0.2, random

# Handle non-numeric columns (drop for now)
X_train = X_train.select_dtypes(include=['number'])
X_test = X_test.select_dtypes(include=['number'])

# Impute missing values with the mean
imputer = SimpleImputer(strategy='mean')
X_train = imputer.fit_transform(X_train)
X_test = imputer.transform(X_test)

# Scale the data
```

```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Create the ANN model
model = Sequential()
model.add(Dense(128, activation='relu', input_shape=(X_train.shape[1],)))
model.add(Dense(64, activation='relu'))
model.add(Dense(1, activation='sigmoid'))  # Output layer for binary classification

# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Train the model
model.fit(X_train, y_train, epochs=10, batch_size=32, verbose=0)  # verbose=0 to suppress tr

# Predict on training and testing data
y_pred_train_probs = model.predict(X_train)
y_pred_test_probs = model.predict(X_test)

# Convert probabilities to binary predictions (0 or 1) using a threshold of 0.5
y_pred_train = (y_pred_train_probs > 0.5).astype(int)
y_pred_test = (y_pred_test_probs > 0.5).astype(int)

# Calculate training and testing accuracy
train_accuracy = accuracy_score(y_train, y_pred_train)
test_accuracy = accuracy_score(y_test, y_pred_test)

# Print training and testing accuracy
print(f"Training Accuracy: {train_accuracy}")
print(f"Testing Accuracy: {test_accuracy}")
```

```
Requirement already satisfied: tensorflow in /usr/local/lib/python3.10/dist-packages (2.
Requirement already satisfied: absl-py>=1.0.0 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: astunparse>=1.6.0 in /usr/local/lib/python3.10/dist-packa
Requirement already satisfied: flatbuffers>=24.3.25 in /usr/local/lib/python3.10/dist-pa
Requirement already satisfied: gast!=0.5.0,!=0.5.1,!=0.5.2,>=0.2.1 in /usr/local/lib/pyt
Requirement already satisfied: google-pasta>=0.1.1 in /usr/local/lib/python3.10/dist-pac
Requirement already satisfied: h5py>=3.10.0 in /usr/local/lib/python3.10/dist-packages (
Requirement already satisfied: libclang>=13.0.0 in /usr/local/lib/python3.10/dist-packag
Requirement already satisfied: ml-dtypes<0.5.0,>=0.3.1 in /usr/local/lib/python3.10/dist
Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.10/dist-packa
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (fro
Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.
Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python3.10/dist-pac
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages (fr
Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.10/dist-packages (f
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.10/dist-packag
Requirement already satisfied: typing-extensions>=3.6.6 in /usr/local/lib/python3.10/dis
Requirement already satisfied: wrapt>=1.11.0 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python3.10/dist-pac
Requirement already satisfied: tensorboard<2.18,>=2.17 in /usr/local/lib/python3.10/dist
Requirement already satisfied: keras>=3.2.0 in /usr/local/lib/python3.10/dist-packages (
```

```
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in /usr/local/lib/py
Requirement already satisfied: numpy<2.0.0,>=1.23.5 in /usr/local/lib/python3.10/dist-pa
Requirement already satisfied: wheel<1.0,>=0.23.0 in /usr/local/lib/python3.10/dist-pack
Requirement already satisfied: rich in /usr/local/lib/python3.10/dist-packages (from ker
Requirement already satisfied: namex in /usr/local/lib/python3.10/dist-packages (from ke
Requirement already satisfied: optree in /usr/local/lib/python3.10/dist-packages (from k
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dis
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-pack
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-pack
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.10/dist-package
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in /usr/local/lib/p
Requirement already satisfied: werkzeug>=1.0.1 in /usr/local/lib/python3.10/dist-package
Requirement already satisfied: MarkupSafe>=2.1.1 in /usr/local/lib/python3.10/dist-packa
Requirement already satisfied: markdown-it-py>=2.2.0 in /usr/local/lib/python3.10/dist-p
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /usr/local/lib/python3.10/dist
Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/python3.10/dist-packages (fr
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: UserWarning:
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
21285/21285 ━━━━━━━━━━━━━━━━━━━━ 33s 2ms/step
8179/8179 ━━━━━━━━━━━━━━━━━━━━ 12s 1ms/step
Training Accuracy: 0.9758298842001559
Testing Accuracy: 0.9760079782356178
```

```
#**Accuracy, Precision, Recall, F1score for ANN**
!pip install tensorflow

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from sklearn.impute import SimpleImputer
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

# Load the datasets
X_train = pd.read_csv('/content/X_Train_Data_Input.csv')
Y_train = pd.read_csv('/content/Y_Train_Data_Target.csv')
X_test = pd.read_csv('/content/X_Test_Data_Input.csv')
Y_test = pd.read_csv('/content/Y_Test_Data_Target.csv')

# ---Ensure that X_train and Y_train have the same number of rows before concatenation---
X_train = X_train.iloc[:min(X_train.shape[0], Y_train.shape[0])]
Y_train = Y_train.iloc[:min(X_train.shape[0], Y_train.shape[0])]
X_test = X_test.iloc[:min(X_test.shape[0], Y_test.shape[0])]
Y_test = Y_test.iloc[:min(X_test.shape[0], Y_test.shape[0])]

# ---Extract the target variable from Y_train and Y_test---
y_train = Y_train['target']
y_test = Y_test['target']
```

```
# ---Now you can split the data using X_train, y_train---
# Split data into training and testing sets (if necessary, you might skip this step)
#X_train, X_test, y_train, y_test = train_test_split(X_train, y_train, test_size=0.2, random

# Handle non-numeric columns (drop for now)
X_train = X_train.select_dtypes(include=['number'])
X_test = X_test.select_dtypes(include=['number'])

# Impute missing values with the mean
imputer = SimpleImputer(strategy='mean')
X_train = imputer.fit_transform(X_train)
X_test = imputer.transform(X_test)

# Scale the data
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Create the ANN model
model = Sequential()
model.add(Dense(128, activation='relu', input_shape=(X_train.shape[1],)))
model.add(Dense(64, activation='relu'))
model.add(Dense(1, activation='sigmoid'))  # Output layer for binary classification

# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Train the model
model.fit(X_train, y_train, epochs=10, batch_size=32, verbose=0)  # verbose=0 to suppress tr

# Predict probabilities
y_pred_probs = model.predict(X_test)

# Convert probabilities to binary predictions (0 or 1) using a threshold of 0.5
y_pred = (y_pred_probs > 0.5).astype(int)

# Calculate metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

# Print metrics
print(f"Accuracy: {accuracy}")
print(f"Precision: {precision}")
print(f"Recall: {recall}")
print(f"F1 Score: {f1}")
```

```
Requirement already satisfied: tensorflow in /usr/local/lib/python3.10/dist-packages (2.
Requirement already satisfied: absl-py>=1.0.0 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: astunparse>=1.6.0 in /usr/local/lib/python3.10/dist-packa
Requirement already satisfied: flatbuffers>=24.3.25 in /usr/local/lib/python3.10/dist-pa
```

```
Requirement already satisfied: gast!=0.5.0,!=0.5.1,!=0.5.2,>=0.2.1 in /usr/local/lib/pyt
Requirement already satisfied: google-pasta>=0.1.1 in /usr/local/lib/python3.10/dist-pac
Requirement already satisfied: h5py>=3.10.0 in /usr/local/lib/python3.10/dist-packages (
Requirement already satisfied: libclang>=13.0.0 in /usr/local/lib/python3.10/dist-packag
Requirement already satisfied: ml-dtypes<0.5.0,>=0.3.1 in /usr/local/lib/python3.10/dist
Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.10/dist-packa
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (fro
Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.
Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python3.10/dist-pac
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages (fr
Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.10/dist-packages (f
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.10/dist-packag
Requirement already satisfied: typing-extensions>=3.6.6 in /usr/local/lib/python3.10/dis
Requirement already satisfied: wrapt>=1.11.0 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python3.10/dist-pac
Requirement already satisfied: tensorboard<2.18,>=2.17 in /usr/local/lib/python3.10/dist
Requirement already satisfied: keras>=3.2.0 in /usr/local/lib/python3.10/dist-packages (
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in /usr/local/lib/py
Requirement already satisfied: numpy<2.0.0,>=1.23.5 in /usr/local/lib/python3.10/dist-pa
Requirement already satisfied: wheel<1.0,>=0.23.0 in /usr/local/lib/python3.10/dist-pack
Requirement already satisfied: rich in /usr/local/lib/python3.10/dist-packages (from ker
Requirement already satisfied: namex in /usr/local/lib/python3.10/dist-packages (from ke
Requirement already satisfied: optree in /usr/local/lib/python3.10/dist-packages (from k
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dis
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-pack
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-pack
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.10/dist-package
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in /usr/local/lib/p
Requirement already satisfied: werkzeug>=1.0.1 in /usr/local/lib/python3.10/dist-package
Requirement already satisfied: MarkupSafe>=2.1.1 in /usr/local/lib/python3.10/dist-packa
Requirement already satisfied: markdown-it-py>=2.2.0 in /usr/local/lib/python3.10/dist-p
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /usr/local/lib/python3.10/dist
Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/python3.10/dist-packages (fr
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: UserWarning:
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
8179/8179 ━━━━━━━━━━━━━━━━━━━━ 14s 2ms/step
Accuracy: 0.975625878828636
Precision: 0.8202373035595534
Recall: 0.9496312505065241
F1 Score: 0.8802043230858796
```

```
#**NAIVE BAYES CLASSIFIER - Training & Testing Accuracy**
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.naive_bayes import GaussianNB  # Import GaussianNB
from sklearn.metrics import accuracy_score
from sklearn.impute import SimpleImputer

# Load the datasets
X_train = pd.read_csv('/content/X_Train_Data_Input.csv')
Y_train = pd.read_csv('/content/Y_Train_Data_Target.csv')
```

```
X_test = pd.read_csv('/content/X_Test_Data_Input.csv')
Y_test = pd.read_csv('/content/Y_Test_Data_Target.csv')

# ---Ensure that X_train and Y_train have the same number of rows before concatenation---
X_train = X_train.iloc[:min(X_train.shape[0], Y_train.shape[0])]
Y_train = Y_train.iloc[:min(X_train.shape[0], Y_train.shape[0])]
X_test = X_test.iloc[:min(X_test.shape[0], Y_test.shape[0])]
Y_test = Y_test.iloc[:min(X_test.shape[0], Y_test.shape[0])]

# ---Extract the target variable from Y_train and Y_test---
y_train = Y_train['target']
y_test = Y_test['target']

# ---Now you can split the data using X_train, y_train---
# Split data into training and testing sets (if necessary, you might skip this step)
#X_train, X_test, y_train, y_test = train_test_split(X_train, y_train, test_size=0.2, random

# Handle non-numeric columns (drop for now)
X_train = X_train.select_dtypes(include=['number'])
X_test = X_test.select_dtypes(include=['number'])

# Impute missing values with the mean
imputer = SimpleImputer(strategy='mean')
X_train = imputer.fit_transform(X_train)
X_test = imputer.transform(X_test)

# Scale the data
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Create and train the Naive Bayes model
model = GaussianNB()  # Create a Gaussian Naive Bayes model
model.fit(X_train, y_train)

# Make predictions on training and testing data
y_pred_train = model.predict(X_train)
y_pred_test = model.predict(X_test)

# Calculate training and testing accuracy
train_accuracy = accuracy_score(y_train, y_pred_train)
test_accuracy = accuracy_score(y_test, y_pred_test)

# Print training and testing accuracy
print(f"Training Accuracy: {train_accuracy}")
print(f"Testing Accuracy: {test_accuracy}")
```

```
Training Accuracy: 0.9586375811486716
Testing Accuracy: 0.9589510607079538
```

```
#**Precision, Recall, F1score for NAIVE BAYES CLASSIFIER.**
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from sklearn.impute import SimpleImputer

# Load the datasets
X_train = pd.read_csv('/content/X_Train_Data_Input.csv')
Y_train = pd.read_csv('/content/Y_Train_Data_Target.csv')
X_test = pd.read_csv('/content/X_Test_Data_Input.csv')
Y_test = pd.read_csv('/content/Y_Test_Data_Target.csv')

# ---Ensure that X_train and Y_train have the same number of rows before concatenation---
X_train = X_train.iloc[:min(X_train.shape[0], Y_train.shape[0])]
Y_train = Y_train.iloc[:min(X_train.shape[0], Y_train.shape[0])]
X_test = X_test.iloc[:min(X_test.shape[0], Y_test.shape[0])]
Y_test = Y_test.iloc[:min(X_test.shape[0], Y_test.shape[0])]

# ---Extract the target variable from Y_train and Y_test---
y_train = Y_train['target']
y_test = Y_test['target']

# ---Now you can split the data using X_train, y_train---
# Split data into training and testing sets (if necessary, you might skip this step)
#X_train, X_test, y_train, y_test = train_test_split(X_train, y_train, test_size=0.2, random

# Handle non-numeric columns (drop for now)
X_train = X_train.select_dtypes(include=['number'])
X_test = X_test.select_dtypes(include=['number'])

# Impute missing values with the mean
imputer = SimpleImputer(strategy='mean')
X_train = imputer.fit_transform(X_train)
X_test = imputer.transform(X_test)

# Scale the data
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Create and train the Naive Bayes model
model = GaussianNB()
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Calculate precision, recall, and F1 score
precision = precision_score(y_test, y_pred)
```

```
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

# Print the results
print(f"Precision: {precision}")
print(f"Recall: {recall}")
print(f"F1 Score: {f1}")
```

```
Precision: 0.7304448486852985
Recall: 0.8949266553205284
F1 Score: 0.8043632655291096
```

```
#**RANDOM FOREST - Training & Testing accuraccy**
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from sklearn.impute import SimpleImputer

# Load the datasets
X_train = pd.read_csv('/content/X_Train_Data_Input.csv')
Y_train = pd.read_csv('/content/Y_Train_Data_Target.csv')
X_test = pd.read_csv('/content/X_Test_Data_Input.csv')
Y_test = pd.read_csv('/content/Y_Test_Data_Target.csv')

# Combine and split data
X = pd.concat([X_train, X_test])
y = pd.concat([Y_train['target'], Y_test['target']])
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Handle non-numeric columns (drop for now)
X_train = X_train.select_dtypes(include=['number'])
X_test = X_test.select_dtypes(include=['number'])

# Impute missing values with the mean
imputer = SimpleImputer(strategy='mean')
X_train = imputer.fit_transform(X_train)
X_test = imputer.transform(X_test)

# Scale the data
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Create and train the model
model = RandomForestClassifier(random_state=42)
model.fit(X_train, y_train)

# Make predictions
y_pred_train = model.predict(X_train)
```

```
y_pred_test = model.predict(X_test)

# Calculate accuracy
train_accuracy = accuracy_score(y_train, y_pred_train)
test_accuracy = accuracy_score(y_test, y_pred_test)

print(f"Training Accuracy: {train_accuracy}")
print(f"Testing Accuracy: {test_accuracy}")
```

```
Training Accuracy: 0.9975629152357799
Testing Accuracy: 0.9765581342032488
```

```
#**Acuracy, Precision, Recall, F1score for RANDOM FOREST**
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from sklearn.impute import SimpleImputer

# Load the datasets
X_train = pd.read_csv('/content/X_Train_Data_Input.csv')
Y_train = pd.read_csv('/content/Y_Train_Data_Target.csv')
X_test = pd.read_csv('/content/X_Test_Data_Input.csv')
Y_test = pd.read_csv('/content/Y_Test_Data_Target.csv')

# ---Ensure that X_train and Y_train have the same number of rows before concatenation---
X_train = X_train.iloc[:min(X_train.shape[0], Y_train.shape[0])]
Y_train = Y_train.iloc[:min(X_train.shape[0], Y_train.shape[0])]
X_test = X_test.iloc[:min(X_test.shape[0], Y_test.shape[0])]
Y_test = Y_test.iloc[:min(X_test.shape[0], Y_test.shape[0])]

# ---Extract the target variable from Y_train and Y_test---
y_train = Y_train['target']
y_test = Y_test['target']

# ---Now you can split the data using X_train, y_train---
# Split data into training and testing sets (if necessary, you might skip this step)
#X_train, X_test, y_train, y_test = train_test_split(X_train, y_train, test_size=0.2, random_



# Handle non-numeric columns (drop for now)
X_train = X_train.select_dtypes(include=['number'])
X_test = X_test.select_dtypes(include=['number'])

# Impute missing values with the mean
imputer = SimpleImputer(strategy='mean')
X_train = imputer.fit_transform(X_train)
X_test = imputer.transform(X_test)

# Scale the data
```

```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Create and train the model
model = RandomForestClassifier(random_state=42)
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Calculate metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

# Print metrics
print(f"Accuracy: {accuracy}")
print(f"Precision: {precision}")
print(f"Recall: {recall}")
print(f"F1 Score: {f1}")
```

```
Accuracy: 0.9758069712003092
Precision: 0.8437368642286676
Recall: 0.9129050596930074
F1 Score: 0.876959204849544
```