```
# *practical no: 5 ( Adjust accordingly )*

import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_sco

# Load the dataset
data = pd.read_csv('NSEI data set.csv')

# Convert the data into a DataFrame
df = pd.DataFrame(data)

# Clean columns with numeric data stored as strings with commas
df = df.replace({',': ''}, regex=True)  # Remove commas
df = df.apply(pd.to_numeric, errors='coerce')  # Convert to numeric, invalid pars

# Drop columns that are not needed (Rule1, Rule2, Rule3, Classifier if they are n
# If 'Classifier' is the target variable, we won't drop it
df = df.drop(columns=['Date', 'Volume', 'Rule1', 'Rule2', 'TP', 'Rule3'])  # Drop

# Step 2: Select features and target variable
# Let's assume 'Classifier' is the target variable
X = df.drop(columns=['Classifier'])  # Features (drop 'Classifier' from the featu
y = df['Classifier']  # Target variable (the 'Classifier' column)

# Check for any NaN values after conversion
print(df.isna().sum())  # This will show you any columns with missing values

# Handle NaN values (e.g., fill with mean
df = df.fillna(df.mean())  # Or df.dropna(

# Define the different training and testi
ratios = [(0.8, 0.2), (0.7, 0.3), (0.6, 0.4)]

# Initialize classifiers
classifiers = {
    'Decision Tree': DecisionTreeClassifier(random_state=42),
    'KNN': KNeighborsClassifier(),
    'Logistic Regression': LogisticRegression(random_state=42, max_iter=1000)
}

# Store metrics for each classifier
for clf_name, clf in classifiers.items():
    print(f"\nEvaluating {clf_name}:")

    # Store results for each ratio
    for ratio in ratios:
```

```
    train_size, test_size = ratio
    print(f"\nResults for {int(train_size*100)}:{int(test_size*100)} train:te

    # Split data into training and test sets based on the current ratio
    X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=trai

    # Train the classifier
    clf.fit(X_train, y_train)

    # Make predictions
    y_pred = clf.predict(X_test)

    # Calculate evaluation metrics
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred, average='weighted')  # Change
    recall = recall_score(y_test, y_pred, average='weighted')  # Change as pe
    f1 = f1_score(y_test, y_pred, average='weighted')  # Change as per the ty

    # Print results for this ratio and classifier
    print(f"Accuracy: {accuracy:.4f}")
    print(f"Precision: {precision:.4f}")
    print(f"Recall: {recall:.4f}")
    print(f"F1 Score: {f1:.4f}")
```

```
Results for 80:20 train:test ratio:
Accuracy: 0.6710
Precision: 0.6723
Recall: 0.6710
F1 Score: 0.6715

Results for 70:30 train:test ratio:
Accuracy: 0.6669
Precision: 0.6671
Recall: 0.6669
F1 Score: 0.6670

Results for 60:40 train:test ratio:
Accuracy: 0.6481
Precision: 0.6489
Recall: 0.6481
F1 Score: 0.6484

Evaluating KNN:

Results for 80:20 train:test ratio:
Accuracy: 0.7203
Precision: 0.7197
Recall: 0.7203
F1 Score: 0.7199
```

Run this cell to mount your Google Drive.
Learn more

```
Accuracy: 0.6904
Precision: 0.6889
Recall: 0.6904
F1 Score: 0.6891

Evaluating Logistic Regression:

Results for 80:20 train:test ratio:
Accuracy: 0.7897
Precision: 0.7908
Recall: 0.7897
F1 Score: 0.7878

Results for 70:30 train:test ratio:
Accuracy: 0.7884
Precision: 0.7892
Recall: 0.7884
F1 Score: 0.7862

Results for 60:40 train:test ratio:
Accuracy: 0.7820
Precision: 0.7833
Recall: 0.7820
F1 Score: 0.7796
```

Start coding or generate with AI.

Double-click (or enter) to edit

```
from google.colab import drive
drive.mount('/content/drive')
```

Run this cell to mount your Google Drive.
Learn more