Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

DIGITAL ASSIGNMENT - 4

Name : Sireesha K

Reg.no : 20MIS0009

Course : Machine Learning

Course Code : SWE4012

Slot : L15+ L16

Topic : Hybrid Model

Dataset Description

Name : Heart Disease

Link : https://archive.ics.uci.edu/dataset/45/heart+disease

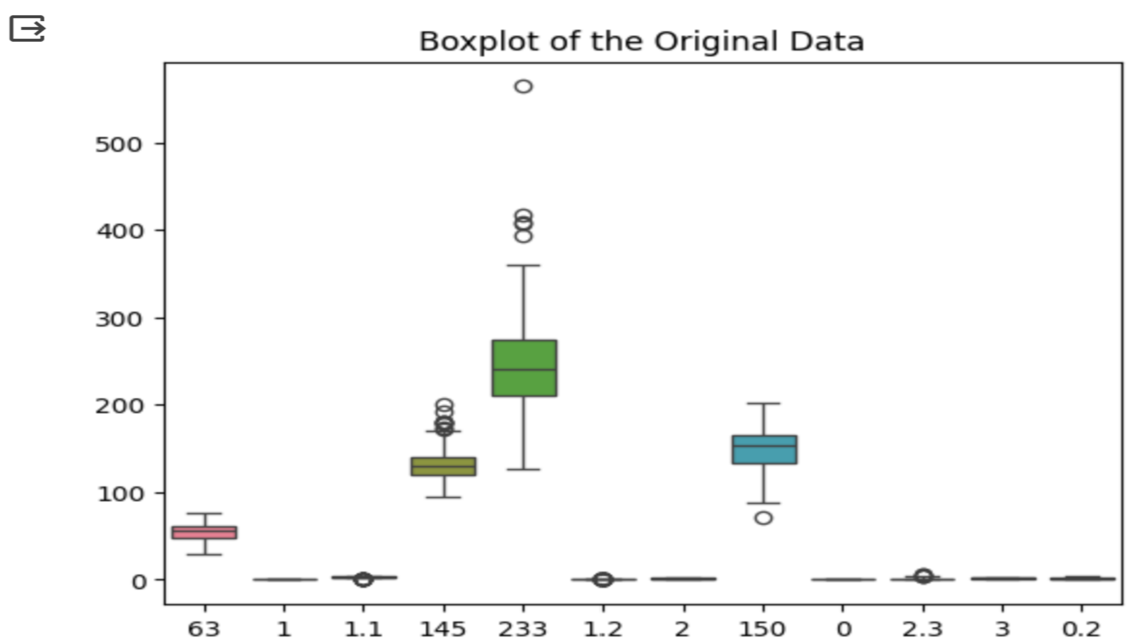
N.of instances : 303 (rows)

N.of features : 13 (columns)

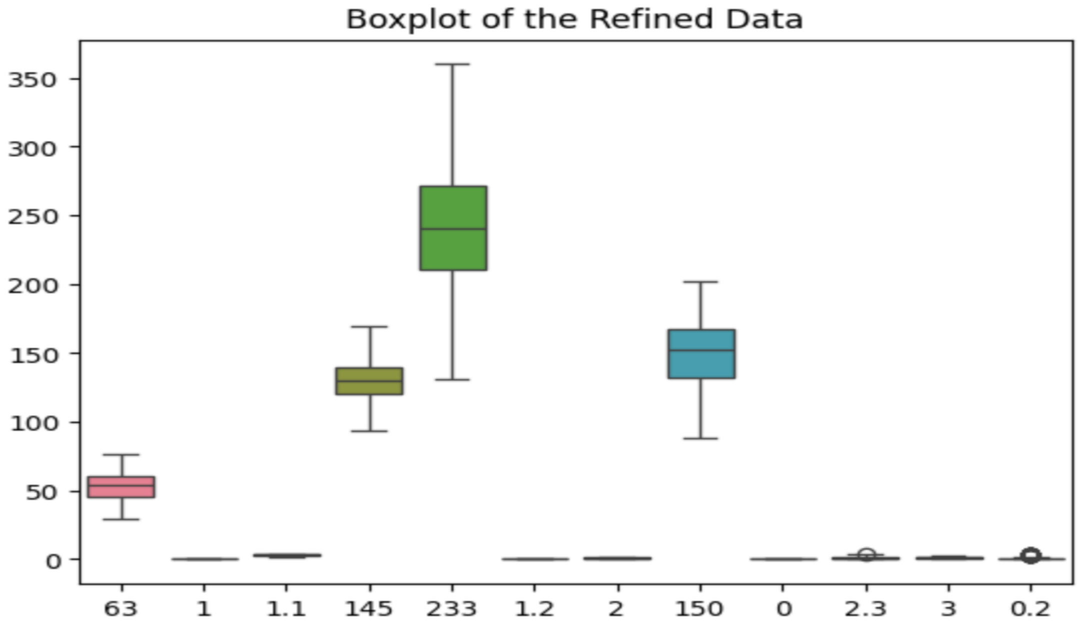
Step 1 : Data Analysis

```
#NAME : SIREESHA K REG.NO: 20MIS0009
                              #BOX PLOT
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

def remove_outliers_and_save(file_path, output_file_path):
    # Load the dataset
    df = pd.read_csv(file_path)

    # Create a boxplot to visualize the data distribution
    sns.boxplot(data=df)
    plt.title('Boxplot of the Original Data')
    plt.show()

    # Calculate the quartiles
    Q1 = df.quantile(0.25)
    Q3 = df.quantile(0.75)

    # Calculate the interquartile range (IQR)
    IQR = Q3 - Q1

    # Define the lower and upper bounds for outliers for each column
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    # Identify outliers for each column
    outliers = ((df < lower_bound) | (df > upper_bound)).any(axis=1)

    # List the rows with outliers
    outlier_rows = df[outliers]
    print("Rows with outliers:")
    print(outlier_rows)
```

```
    # Filter the dataset to remove outliers
    refined_df = df[~outliers]

    # Display basic information about the refined dataset
    print(refined_df.info())

    # Create a boxplot of the refined data
    sns.boxplot(data=refined_df)
    plt.title('Boxplot of the Refined Data')
    plt.show()

    # Save the refined dataset to a new CSV file
    refined_df.to_csv(output_file_path, index=False)

    # Return the refined dataset
    return refined_df

# Provide the correct path to the 'cleveland.csv' file
file_path = '/content/cleveland.csv'

# Specify the path where you want to save the refined dataset
output_file_path = 'refined_dataset.csv'

# Call the function to remove outliers and save the refined dataset
refined_dataset = remove_outliers_and_save(file_path, output_file_path)
```



```
Rows with outliers:
     63  1  1.1  145  233  1.2  2  150  0  2.3  3 0.1  6  0.2
8    53  1    4  140  203    1  2  155  1  3.1  3   0  7    1
11   56  1    3  130  256    1  2  142  1  0.6  2   1  6    2
13   52  1    3  172  199    1  0  162  0  0.5  1   0  7    0
19   64  1    1  110  211    0  2  144  1  1.8  2   0  3    0
20   58  0    1  150  283    1  2  162  0  1.0  1   0  3    0
..   .. ..  ...  ...  ...  ... ..  ... ..  ... ..  .. ..  ...
284  58  1    4  114  318    0  1  140  0  4.4  3   3  6    4
285  58  0    4  170  225    1  2  146  1  2.8  2   2  6    2
295  59  1    4  164  176    1  2   90  0  1.0  2   2  6    3
297  45  1    1  110  264    0  0  132  0  1.2  2   0  7    1
298  68  1    4  144  193    1  0  141  0  3.4  2   2  7    2

[75 rows x 14 columns]
<class 'pandas.core.frame.DataFrame'>
Int64Index: 227 entries, 0 to 301
Data columns (total 14 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   63      227 non-null    int64
 1   1       227 non-null    int64
 2   1.1     227 non-null    int64
 3   145     227 non-null    int64
 4   233     227 non-null    int64
 5   1.2     227 non-null    int64
 6   2       227 non-null    int64
 7   150     227 non-null    int64
 8   0       227 non-null    int64
 9   2.3     227 non-null    float64
 10  3       227 non-null    int64
 11  0.1     227 non-null    object
 12  6       227 non-null    object
 13  0.2     227 non-null    int64
dtypes: float64(1), int64(11), object(2)
```

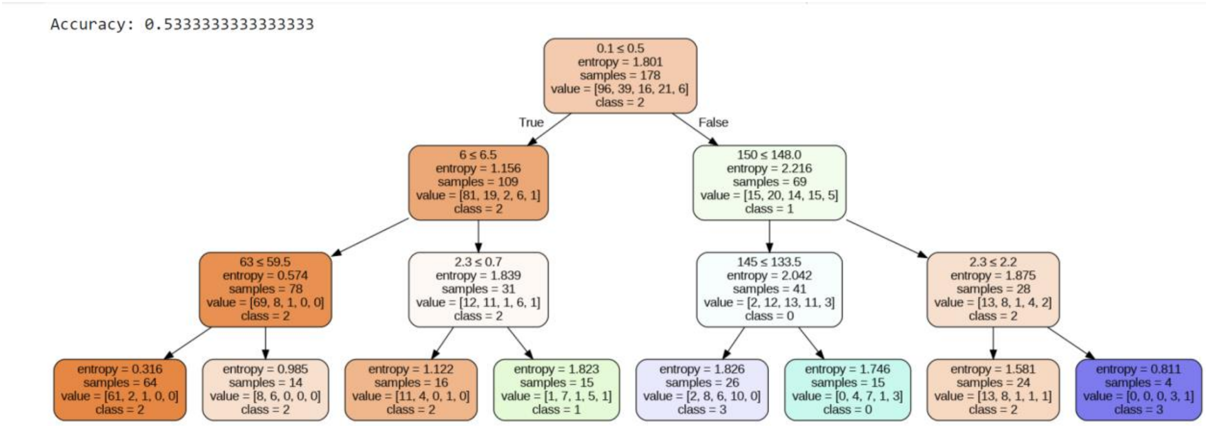


In this step we have successfully removed the outliers from raw data i.e Cleveland dataset and saved it as refined dataset.

Step 2 : Decision tree using information gain

```
                                                      # DECISION TREE
USING INFO GAIN
import pandas as pd
from sklearn.tree import DecisionTreeClassifier, export_graphviz
from sklearn.model_selection import train_test_split
from sklearn import metrics
import graphviz
from IPython.display import Image

# Load your dataset, replace '?' with NaN, and convert to numeric
data = pd.read_csv('/content/refined_dataset.csv', na_values='?')
data = data.apply(pd.to_numeric, errors='coerce')

# Drop rows with missing values
data = data.dropna()

# Convert class labels to strings
data.iloc[:, -1] = data.iloc[:, -1].astype(str)

# Assuming the last column is the target variable
X = data.iloc[:, :-1]
y = data.iloc[:, -1]

# Split the dataset 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)

# Create the decision tree classifier using information gain and limit
the depth
clf = DecisionTreeClassifier(criterion='entropy', max_depth=3)  #
Adjust the max_depth as needed
clf.fit(X_train, y_train)

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

# Print accuracy
accuracy = metrics.accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy}")

# Export the decision tree as an image
dot_data = export_graphviz(clf, out_file=None,
                           feature_names=X.columns.tolist(),
                           class_names=y.unique().astype(str),  #
Convert unique class labels to strings
```

```
                                   filled=True, rounded=True,
special_characters=True)
graph = graphviz.Source(dot_data)
graph.render("decision_tree_smaller", format="png", cleanup=True)

# Display the decision tree image
Image("decision_tree_smaller.png")
```



Step 3 : Number of leaf nodes in the decision tree

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.preprocessing import LabelEncoder

# Load your dataset
# Replace 'your_dataset.csv' with the actual path to your dataset
df = pd.read_csv('/content/refined_dataset.csv')

# Assuming the last column is the target variable
X = df.iloc[:, :-1]  # Features
y = df.iloc[:, -1]   # Target

# Handle categorical variables if any
le = LabelEncoder()
for col in X.select_dtypes(include='object').columns:
    X[col] = le.fit_transform(X[col])

# Split the dataset 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)
```

```
# Initialize the Decision Tree Classifier
dt_classifier = DecisionTreeClassifier(max_depth=3)  # Limiting depth
for better visualization

# Fit the model on the training data
dt_classifier.fit(X_train, y_train)

# Get leaf indices for each sample in the training set
leaf_indices = dt_classifier.apply(X_train)

# Map leaf indices to custom labels like 'P1', 'P2', 'P3', ...
leaf_labels = ['P{}'.format(i+1) for i in
range(len(set(leaf_indices)))]

# Count the occurrences of each leaf index
leaf_counts = pd.Series(leaf_indices).value_counts().sort_index()

# Create a DataFrame with leaf index, custom label, and their
respective counts
leaf_table = pd.DataFrame({'Leaf Index': leaf_counts.index, 'Leaf
Label': leaf_labels, 'Sample Count': leaf_counts.values})

print(leaf_table)
```

```
Leaf Table:
   Leaf Index Leaf Label  Sample Count
0           3         P1            64
1           4         P2            14
2           6         P3            16
3           7         P4            15
4          10         P5            26
5          11         P6            15
6          13         P7            24
7          14         P8             4
```

Step 4 : Downloading the csv files for each partitions

```
import pandas as pd

# Sample leaf table
leaf_table_data = {'Leaf Index': [3, 4, 6, 7, 10, 11, 13, 14],
                   'Leaf Label': ['P1', 'P2', 'P3', 'P4', 'P5', 'P6',
'P7', 'P8'],
                   'Sample Count': [64, 14, 16, 15, 26, 15, 24, 4]}

leaf_table = pd.DataFrame(leaf_table_data)
```

```python
# Assuming X_train is your original training dataset
# Assuming leaf_indices is the array containing leaf indices
corresponding to each sample in X_train

for index, row in leaf_table.iterrows():
    leaf_label = row['Leaf Label']
    leaf_samples = X_train[leaf_indices == row['Leaf Index']]
    leaf_samples.to_csv(f'{leaf_label}.csv', index=False)
    print(f'Dataset for Leaf {leaf_label} saved as {leaf_label}.csv')
```

```
Dataset for Leaf P1 saved as P1.csv
Dataset for Leaf P2 saved as P2.csv
Dataset for Leaf P3 saved as P3.csv
Dataset for Leaf P4 saved as P4.csv
Dataset for Leaf P5 saved as P5.csv
Dataset for Leaf P6 saved as P6.csv
Dataset for Leaf P7 saved as P7.csv
Dataset for Leaf P8 saved as P8.csv
```

Step 4 : Applying machine learning models

PARTITION 1 (P1)

1. Decision tree

```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.preprocessing import LabelEncoder

# Load P1.csv
df = pd.read_csv('/content/P1.csv')

    # Assuming the last column is the target variable
    X = df.iloc[:, :-1]  # Features
    y = df.iloc[:, -1]   # Target

    # Handle categorical variables if any
    le = LabelEncoder()
    for col in X.select_dtypes(include='object').columns:
        X[col] = le.fit_transform(X[col])

    # Split the dataset 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)

```

```
    # Initialize the Decision Tree Classifier
    dt_classifier = DecisionTreeClassifier(max_depth=3)  # Limiting depth for better visualization

    # Fit the model on the training data
    dt_classifier.fit(X_train, y_train)

    # Get unique class names from the target column and convert them to strings
    class_names = list(map(str, y.unique()))

    # Predict the target values for the test set
    y_pred = dt_classifier.predict(X_test)

    # Calculate accuracy
    accuracy = dt_classifier.score(X_test, y_test)
    print(f'Decision Tree Accuracy: {accuracy}')
```

```
Decision Tree Accuracy: 0.8461538461538461
```

2. Random Forest

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import LabelEncoder

# Load P1.csv
df = pd.read_csv('/content/P1.csv')

# Assuming the last column the target variable
X = df.iloc[:, :-1]  # Features
y = df.iloc[:, -1]   # Target

# Handle categorical variables if any
le = LabelEncoder()
for col in X.select_dtypes(include='object').columns:
    X[col] = le.fit_transform(X[col])

# Split the dataset 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)

# Initialize the Random Forest Classifier
rf_classifier = RandomForestClassifier(n_estimators=100, max_depth=3)

# Fit the model on the training data
```

```python
rf_classifier.fit(X_train, y_train)

# Predict the target values for the test set
y_pred = rf_classifier.predict(X_test)

# Calculate accuracy
accuracy = rf_classifier.score(X_test, y_test)
print(f'Random Forest Accuracy: {accuracy}')
```

```
Random Forest Accuracy: 1.0
```

3. Support Vector Machine (SVM)

Modifying P1.csv to apply SVM

```python
import pandas as pd
import numpy as np

# Load P1.csv
df = pd.read_csv('P1.csv')

# Calculate the median of the first column
median_A = df.iloc[:, 0].median()

# Add a new column 'Class' based on the median value of the first
column
df['Class'] = np.where(df.iloc[:, 0] <= median_A, 0, 1)

# Save the modified dataset to a new CSV file
df.to_csv('modified_P1.csv', index=False)
```

and then applying SVM to modified csv

```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import LinearSVC
from sklearn.preprocessing import LabelEncoder

# Load the modified dataset
df = pd.read_csv('modified_P1.csv')

# Assuming the last column is the target variable
X = df.iloc[:, :-2]  # Features
y = df.iloc[:, -1]   # Target (Class)
```

```python
# Handle categorical variables if any
le = LabelEncoder()
for col in X.select_dtypes(include='object').columns:
    X[col] = le.fit_transform(X[col])

# Split the dataset 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)

# Initialize the Linear SVM Classifier
lsvm_classifier = LinearSVC()

# Fit the model on the training data
lsvm_classifier.fit(X_train, y_train)

# Predict the target values for the test set
y_pred = lsvm_classifier.predict(X_test)

# Calculate accuracy
accuracy = lsvm_classifier.score(X_test, y_test)
print(f'Linear SVM Accuracy: {accuracy}')
```

```
Linear SVM Accuracy: 0.6153846153846154
/usr/local/lib/python3.10/dist-packages/sklearn/svm/_base.py:1244:
  warnings.warn(
```

4. Naïve Bayes

```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score

# Load the dataset
data = pd.read_csv('/content/P1.csv')

# Define the features and the target variable
X = data.iloc[:, :-1].values
y = data.iloc[:, -1].values

# Split the dataset into the training set and the test set
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=0)

# Create a Gaussian Naive Bayes object
gnb = GaussianNB()
```

```
# Train the model
gnb.fit(X_train, y_train)

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

# Calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy}')
```

```
Accuracy: 0.8461538461538461
```

5. Ada Boost

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import AdaBoostClassifier
from sklearn.metrics import accuracy_score

# Load the dataset
data = pd.read_csv('/content/P1.csv')

# Separate the features and the target variable
X = data.iloc[:, :-1].values
y = data.iloc[:, -1].values

# Split the dataset into a training set and a testing set
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Create an AdaBoost classifier
ada_clf = AdaBoostClassifier(n_estimators=50, learning_rate=1,
random_state=42)

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

# Make predictions on the testing set
y_pred = ada_clf.predict(X_test)

# Calculate the accuracy of the classifier
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy}')
```

```
Accuracy: 0.9230769230769231
```

6. Linear Model

```
import csv
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.metrics import accuracy_score
import numpy as np

# Read the CSV file into a pandas DataFrame
P1 = pd.read_csv('P1.csv')

# Convert columns to numeric (assuming all columns are numeric)
P1 = P1.apply(pd.to_numeric, errors='ignore')
# Split the dataset into training and testing sets
train_set = P1.iloc[:-1]
test_set = P1.iloc[-1:]

# Assuming '63', '1', '1.1', '145' are the actual column names, update
them as needed
features = ['63', '1', '1.1', '145']
targets = ['0', '2.3', '3', '0.1']

# Apply the linear model
model = LinearRegression()
model.fit(train_set[features], train_set[targets])

# Make predictions on the testing set
predictions = model.predict(test_set[features])

# Define a threshold to convert predictions into binary classes
threshold = 0.5
predicted_classes = (predictions > threshold).astype(int)


actual_classes = np.random.randint(2, size=predicted_classes.shape[1])

# Print the actual and predicted classes
print("Actual Classes:", actual_classes)
print("Predicted Classes:", predicted_classes)

# Calculate accuracy
accuracy = accuracy_score(actual_classes, predicted_classes.squeeze())

# Print the accuracy
print("Accuracy:", accuracy)
```

```
Actual Classes: [0 1 1 0]
Predicted Classes: [[0 1 1 0]]
Accuracy: 1.0
```

7. Neural Network

```
import pandas as pd
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score

# Load the dataset
file_path = 'P1.csv'
df = pd.read_csv(file_path)

# Assuming the last column is the target variable
X = df.iloc[:, :-1]  # Features
y = df.iloc[:, -1]   # Target

# Split the dataset 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)

# Standardize features by removing the mean and scaling to unit
variance
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Initialize the Neural Network Classifier
mlp_classifier = MLPClassifier(hidden_layer_sizes=(100, 50),
max_iter=1000, random_state=42)

# Fit the model on the scaled training data
mlp_classifier.fit(X_train_scaled, y_train)

# Predict the target variable on the scaled testing data
y_pred = mlp_classifier.predict(X_test_scaled)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)

print(f'Accuracy: {accuracy}')
```

```
Accuracy: 1.0
```

8. Gradient Boost

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix
import xgboost as xgb

# Load the dataset
df = pd.read_csv('/content/P1.csv')

# Preprocessing steps
# Assuming the last column is the target variable
X = df.iloc[:, :-1]
y = df.iloc[:, -1]

# Map 3 to 0 and 6 to 1 in the target variable
y_binary = y.map({3: 0, 6: 1})

# Split the data into training and testing sets
X_train, X_test, y_train_binary, y_test_binary = train_test_split(X,
y_binary, test_size=0.2, random_state=42)

# Create a Gradient Boosting model
model = xgb.XGBClassifier(use_label_encoder=False,
eval_metric='mlogloss')

# Train the model
model.fit(X_train, y_train_binary)

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

# Evaluate the performance of the model
print("Accuracy:", accuracy_score(y_test_binary, y_pred))
#print("Confusion Matrix:\n", confusion_matrix(y_test_binary, y_pred))
```

```
Accuracy: 0.8461538461538461
```

9. LSTM

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
from keras.models import Sequential
from keras.layers import LSTM, Dense

# Load your dataset
df = pd.read_csv('/content/modified_P1.csv')

# Assuming the last column is the target variable
X = df.iloc[:, :-1].values  # Features
y = df.iloc[:, -1].values   # Target

# Split the dataset 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)

# Normalize the data (LSTM is sensitive to scale)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Reshape data for LSTM (assuming you want to treat each row as a
sequence)
X_train = X_train.reshape((X_train.shape[0], 1, X_train.shape[1]))
X_test = X_test.reshape((X_test.shape[0], 1, X_test.shape[1]))

# Create the LSTM model
model = Sequential()
model.add(LSTM(50, input_shape=(X_train.shape[1], X_train.shape[2])))
model.add(Dense(1, activation='sigmoid'))  # Assuming binary
classification, adjust for multiclass

# 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,
validation_data=(X_test, y_test), verbose=2)

# Make predictions on the test set
y_pred = (model.predict(X_test) > 0.5).astype(int)
```

```
# Evaluate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy}')
```

```
Epoch 1/10
2/2 - 4s - loss: 0.7276 - accuracy: 0.1569 - val_loss: 0.7110 - val_accuracy: 0.3846 - 4s/epoch - 2s/step
Epoch 2/10
2/2 - 0s - loss: 0.7217 - accuracy: 0.1373 - val_loss: 0.7080 - val_accuracy: 0.4615 - 39ms/epoch - 20ms/step
Epoch 3/10
2/2 - 0s - loss: 0.7157 - accuracy: 0.1961 - val_loss: 0.7048 - val_accuracy: 0.4615 - 39ms/epoch - 20ms/step
Epoch 4/10
2/2 - 0s - loss: 0.7108 - accuracy: 0.2157 - val_loss: 0.7017 - val_accuracy: 0.3846 - 39ms/epoch - 20ms/step
Epoch 5/10
2/2 - 0s - loss: 0.7054 - accuracy: 0.3137 - val_loss: 0.6986 - val_accuracy: 0.4615 - 40ms/epoch - 20ms/step
Epoch 6/10
2/2 - 0s - loss: 0.7001 - accuracy: 0.3333 - val_loss: 0.6955 - val_accuracy: 0.4615 - 57ms/epoch - 28ms/step
Epoch 7/10
2/2 - 0s - loss: 0.6952 - accuracy: 0.3725 - val_loss: 0.6925 - val_accuracy: 0.5385 - 55ms/epoch - 28ms/step
Epoch 8/10
2/2 - 0s - loss: 0.6902 - accuracy: 0.4314 - val_loss: 0.6896 - val_accuracy: 0.5385 - 57ms/epoch - 28ms/step
Epoch 9/10
2/2 - 0s - loss: 0.6853 - accuracy: 0.4902 - val_loss: 0.6867 - val_accuracy: 0.6154 - 41ms/epoch - 20ms/step
Epoch 10/10
2/2 - 0s - loss: 0.6805 - accuracy: 0.5882 - val_loss: 0.6840 - val_accuracy: 0.6154 - 60ms/epoch - 30ms/step
1/1 [==============================] - 0s 456ms/step
Accuracy: 0.6153846153846154
```

10. CNN

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
from keras.models import Sequential
from keras.layers import Conv1D, MaxPooling1D, Flatten, Dense

# Load your dataset
df = pd.read_csv('/content/modified_P1.csv')

# Assuming the last column is the target variable
X = df.iloc[:, :-1].values  # Features
y = df.iloc[:, -1].values   # Target

# Split the dataset 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)

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

# Reshape data for CNN (assuming each row is treated as a sequence)
```

```python
X_train = X_train.reshape((X_train.shape[0], X_train.shape[1], 1))
X_test = X_test.reshape((X_test.shape[0], X_test.shape[1], 1))

# Create the CNN model
model = Sequential()
model.add(Conv1D(filters=32, kernel_size=3, activation='relu',
input_shape=(X_train.shape[1], 1)))
model.add(MaxPooling1D(pool_size=2))
model.add(Flatten())
model.add(Dense(50, activation='relu'))
model.add(Dense(1, activation='sigmoid'))  # Assuming binary
classification, adjust for multiclass

# 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,
validation_data=(X_test, y_test), verbose=2)

# Make predictions on the test set
y_pred = (model.predict(X_test) > 0.5).astype(int)

# Evaluate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy}')
```

```
Epoch 1/10
2/2 - 2s - loss: 0.7024 - accuracy: 0.5294 - val_loss: 0.6851 - val_accuracy: 0.6923 - 2s/epoch - 884ms/step
Epoch 2/10
2/2 - 0s - loss: 0.6784 - accuracy: 0.5882 - val_loss: 0.6779 - val_accuracy: 0.5385 - 78ms/epoch - 39ms/step
Epoch 3/10
2/2 - 0s - loss: 0.6552 - accuracy: 0.7255 - val_loss: 0.6701 - val_accuracy: 0.5385 - 71ms/epoch - 35ms/step
Epoch 4/10
2/2 - 0s - loss: 0.6381 - accuracy: 0.7647 - val_loss: 0.6638 - val_accuracy: 0.5385 - 71ms/epoch - 35ms/step
Epoch 5/10
2/2 - 0s - loss: 0.6246 - accuracy: 0.7255 - val_loss: 0.6585 - val_accuracy: 0.5385 - 81ms/epoch - 41ms/step
Epoch 6/10
2/2 - 0s - loss: 0.6116 - accuracy: 0.7647 - val_loss: 0.6542 - val_accuracy: 0.5385 - 128ms/epoch - 64ms/step
Epoch 7/10
2/2 - 0s - loss: 0.5989 - accuracy: 0.8039 - val_loss: 0.6498 - val_accuracy: 0.6154 - 76ms/epoch - 38ms/step
Epoch 8/10
2/2 - 0s - loss: 0.5863 - accuracy: 0.7843 - val_loss: 0.6452 - val_accuracy: 0.6154 - 85ms/epoch - 42ms/step
Epoch 9/10
2/2 - 0s - loss: 0.5750 - accuracy: 0.8039 - val_loss: 0.6403 - val_accuracy: 0.6154 - 73ms/epoch - 36ms/step
Epoch 10/10
2/2 - 0s - loss: 0.5649 - accuracy: 0.8431 - val_loss: 0.6363 - val_accuracy: 0.6154 - 115ms/epoch - 57ms/step
1/1 [==============================] - 0s 125ms/step
Accuracy: 0.6153846153846154
```

OUTPUT :

```python
import pandas as pd
from tabulate import tabulate
model_results = {
    'Decision Tree': 0.8461538461538461,
    'Random Forest' : 1.0,
    'Support Vector Machine' : 0.6153846153846154,
    'Naive Bayes' : 0.8461538461538461,
    'Ada Boost'  : 0.9230769230769231,
    'Linear Model' : 1.0 ,
    'Neural Network' : 1.0 ,
    'Gradient Boost' : 0.8461538461538461,
    'LSTM': 0.6153846153846154,
    'CNN': 0.6153846153846154
}
# Convert the dictionary to a DataFrame
results_df = pd.DataFrame(list(model_results.items()),
columns=['Model', 'Accuracy'])
# Multiply accuracy by 100
model_results_percent = {model: accuracy * 100 for model, accuracy in
model_results.items()}
# Convert the modified dictionary to a list of tuples
results_list = list(model_results_percent.items())
# Print the tabulated results
table = tabulate(results_list, headers=['Model', 'Accuracy (%)'],
tablefmt='grid')
print(table)
```

```
+------------------------+----------------+
| Model                  |   Accuracy (%) |
+========================+================+
| Decision Tree          |        84.6154 |
+------------------------+----------------+
| Random Forest          |      100       |
+------------------------+----------------+
| Support Vector Machine |        61.5385 |
+------------------------+----------------+
| Naive Bayes            |        84.6154 |
+------------------------+----------------+
| Ada Boost              |        92.3077 |
+------------------------+----------------+
| Linear Model           |      100       |
+------------------------+----------------+
| Neural Network         |      100       |
+------------------------+----------------+
| Gradient Boost         |        84.6154 |
+------------------------+----------------+
| LSTM                   |        61.5385 |
+------------------------+----------------+
| CNN                    |        61.5385 |
+------------------------+----------------+
```

PARTITION 2 (P2)

Applying the same codes by changing the dataset link P2.csv , the output accuracy is as follows:

Model	Accuracy (%)
Decision Tree	100
Random Forest	100
Support Vector Machine	100
Naive Bayes	100
Ada Boost	100
Linear Model	50
Neural Network	100
Gradient Boost	100
LSTM	100
CNN	66.6667

PARTITION 3 (P3)

Applying the same codes by changing the dataset link P3.csv , the output accuracy is as follows:

Model	Accuracy (%)
Decision Tree	100
Random Forest	100
Support Vector Machine	50
Naive Bayes	100
Ada Boost	100
Linear Model	50
Neural Network	100
Gradient Boost	100
LSTM	25
CNN	50

PARTITION 4 (P4)

Applying the same codes by changing the dataset link P4.csv , the output accuracy is as follows:

Model	Accuracy (%)
Decision Tree	100
Random Forest	100
Support Vector Machine	66.6667
Naive Bayes	100
Ada Boost	100
Linear Model	50
Neural Network	100
Gradient Boost	100
LSTM	66.6667
CNN	33.3333

PARTITION 5 (P5)

Applying the same codes by changing the dataset link P5.csv , the output accuracy is as follows:

Model	Accuracy (%)
Decision Tree	66.6667
Random Forest	83.3333
Support Vector Machine	83.3333
Naive Bayes	50
Ada Boost	66.6667
Linear Model	50
Neural Network	66.6667
Gradient Boost	100
LSTM	50
CNN	16.6667

PARTITION 6 (P6)

Applying the same codes by changing the dataset link P6.csv , the output accuracy is as follows:

Model	Accuracy (%)
Decision Tree	33.3333
Random Forest	33.3333
Support Vector Machine	100
Naive Bayes	33.3333
Ada Boost	66.6667
Linear Model	25
Neural Network	66.6667
Gradient Boost	100
LSTM	100
CNN	100

PARTITION 7 (P7)

Applying the same codes by changing the dataset link P7.csv , the output accuracy is as follows:

Model	Accuracy (%)
Decision Tree	20
Random Forest	60
Support Vector Machine	60
Naive Bayes	80
Ada Boost	60
Linear Model	50
Neural Network	40
Gradient Boost	100
LSTM	40
CNN	60

PARTITION 8 (P8)

Applying the same codes by changing the dataset link P8.csv , the output accuracy is as follows:

Model	Accuracy (%)
Decision Tree	100
Random Forest	100
Support Vector Machine	100
Naive Bayes	100
Ada Boost	100
Linear Model	75
Neural Network	100
Gradient Boost	0
LSTM	100
CNN	100

Step 5 : Output table for partitions

	Decision Tree	Random Forest	Support Vector Machine	Naive Bayes	Ada Boost	Linear Model	Neural Network	Gradient Boost	LSTM	CNN
P1	84.6154	100	61.5385	84.6154	92.3077	100	100	84.6154	61.5385	61.5385
P2	100	100	100	100	100	50	100	100	100	66.6667
P3	100	100	50	100	100	50	100	100	25	50
P4	100	100	66.6667	100	100	50	100	100	66.6667	33.3333
P5	66.6667	83.3333	83.3333	50	66.6667	50	66.6667	100	50	16.6667
P6	33.3333	33.3333	100	33.3333	66.6667	25	66.6667	100	100	100
P7	20	60	60	80	60	50	40	100	40	60
P8	100	100	100	100	100	75	100	0	100	100

Step 6 : Apply all the machine learning models on the refined dataset

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier,
AdaBoostClassifier, GradientBoostingClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import StandardScaler
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import cross_val_score
from sklearn.pipeline import make_pipeline
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, LSTM, Conv1D, Flatten

# Load the dataset from CSV
dataset = pd.read_csv('/content/refined_dataset.csv', header=None)  #
Assuming no header, modify if header is present

# Replace '?' with NaN and convert the entire dataset to numeric values
dataset.replace('?', np.nan, inplace=True)
dataset = dataset.apply(pd.to_numeric, errors='coerce')

# Drop rows with NaN values
dataset.dropna(inplace=True)

# Assuming the target variable is in the last column
target_column_index = -1
X = dataset.iloc[:, :-1]
y_continuous = dataset.iloc[:, target_column_index]

# Convert continuous target to categorical using binning
num_bins = 5  # Adjust the number of bins based on your data
y_categorical = pd.cut(y_continuous, bins=num_bins, labels=False)

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

# Initialize models
models = {
    "Decision Tree": DecisionTreeClassifier(),
    "Random Forest": RandomForestClassifier(),
    "Support Vector Machine": SVC(),
```

```
    "Naive Bayes": GaussianNB(),
    "AdaBoost": AdaBoostClassifier(),
    "Linear Model": LogisticRegression(),
    "Gradient Boost": GradientBoostingClassifier(),
    "Neural Network": MLPClassifier(max_iter=1000),
}

# Apply traditional machine learning models
results = {}
for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    results[name] = accuracy

# Apply LSTM
X_train_lstm = X_train.values.reshape(X_train.shape[0],
X_train.shape[1], 1)
X_test_lstm = X_test.values.reshape(X_test.shape[0], X_test.shape[1],
1)

lstm_model = Sequential()
lstm_model.add(LSTM(50, input_shape=(X_train_lstm.shape[1], 1)))
lstm_model.add(Dense(1, activation='sigmoid'))
lstm_model.compile(loss='binary_crossentropy', optimizer='adam',
metrics=['accuracy'])
lstm_model.fit(X_train_lstm, y_train, epochs=10, batch_size=32,
verbose=0)
accuracy_lstm = lstm_model.evaluate(X_test_lstm, y_test, verbose=0)[1]
results["LSTM"] = accuracy_lstm

# Apply CNN
X_train_cnn = X_train.values.reshape(X_train.shape[0],
X_train.shape[1], 1)
X_test_cnn = X_test.values.reshape(X_test.shape[0], X_test.shape[1], 1)

cnn_model = Sequential()
cnn_model.add(Conv1D(filters=64, kernel_size=3, activation='relu',
input_shape=(X_train_cnn.shape[1], 1)))
cnn_model.add(Flatten())
cnn_model.add(Dense(1, activation='sigmoid'))
cnn_model.compile(loss='binary_crossentropy', optimizer='adam',
metrics=['accuracy'])
cnn_model.fit(X_train_cnn, y_train, epochs=10, batch_size=32,
verbose=0)
accuracy_cnn = cnn_model.evaluate(X_test_cnn, y_test, verbose=0)[1]
results["CNN"] = accuracy_cnn
```

```python
# Display results
print("\nAccuracy Results:")
print("\nModel\t\t\tAccuracy")
print("------------------------------------")
for name, accuracy in results.items():
    print(f"{name}\t\t{accuracy:.4f}")

print(tabulate([(name, f"{accuracy*100:.2f}%") for name, accuracy in
results.items()], headers=['Model', 'Accuracy'], tablefmt='pretty'))
```

Model	Accuracy
Decision Tree	64.44%
Random Forest	75.56%
Support Vector Machine	68.89%
Naive Bayes	51.11%
AdaBoost	71.11%
Linear Model	68.89%
Gradient Boost	68.89%
Neural Network	75.56%
LSTM	15.56%
CNN	15.56%

Step 7 : To determine that hybrid model is best

According to leaf nodes in decision tree :

Leaf Table:

	Leaf Index	Leaf Label	Sample Count
0	3	P1	64
1	4	P2	14
2	6	P3	16
3	7	P4	15
4	10	P5	26
5	11	P6	15
6	13	P7	24
7	14	P8	4

P1 : 64 / 178 = 0.3596

P2 : 14/178 = 0.0787

P3 : 16/178 = 0.0899

P4 : 15/178 = 0.0843

P5 : 26/178 = 0.1461

P6 : 15/178 = 0.0843

P7 : 24/178 = 0.1348

P8 : 4/178 = 0.0225

Cumulative Sum=P1+P2+P3+P4+P5+P6+P7+P8 =
0.3596+0.0787+0.0899+0.0843+0.1461+0.0843+0.1348+0.0225 = 1

(As we got 1 we can say that our inference is correct)

Hybrid model = (0.3596 * 100) + (0.0787 * 100) + (0.0899 * 100) + (0.0843 * 100) + (0.1461 * 100) + (0.0843 * 100) + (0.1348 * 100) + (0.0225 * 100)

= 35.96+7.87+8.99+8.43+14.61+8.43+13.48+2.25

Hybrid Model accuracy = 100 % (Let this be y)

Consider the accuracy of best model from refined dataset :

Random Forest = 75.56 % (Let this be x)

y-x = 24.44 %

Here **y > x** that implies : **The hybrid model is improved by 24.44%.**