

## Assignment-2:

Abdelrahman Sayed\_20201114

Seif El-Din Mohamed\_20200239

Shahd Fekry ali\_20201101

Mariam Alaa Eldeen\_20200525

Wessam Fawzy\_20201215

### Problem-1

#### Import Libraries

In [ ]:

```
import pandas as pd # Pandas for data manipulation
from sklearn.model_selection import train_test_split # Train-test split for model evaluation
from sklearn.tree import DecisionTreeClassifier # Decision tree classifier for modeling
from sklearn.metrics import accuracy_score # Metric for evaluating model performance
from sklearn.compose import ColumnTransformer # Used for transforming specific columns
from sklearn.preprocessing import OneHotEncoder # One-hot encoding for categorical variables
import numpy as np # NumPy for numerical operations
from sklearn.preprocessing import LabelEncoder # Label encoding for categorical variables
import matplotlib.pyplot as plt # Matplotlib for plotting
```

#### import Data from Dataset

```
In [ ]: dataset = pd.read_csv('drug.csv') # Read the dataset from a CSV file named 'drug.csv' using Pandas
```

## Data Preprocessing

### Count Missing Values Occurs

```
In [ ]: missing_values = dataset.isnull().sum() # Check for missing values in the dataset
print("Missing Values:\n", missing_values) # Print the count of missing values for each column
```

Missing Values:

|             |   |
|-------------|---|
| Age         | 0 |
| Sex         | 0 |
| BP          | 2 |
| Cholesterol | 2 |
| Na_to_K     | 1 |
| Drug        | 0 |

dtype: int64

### Handling Missing Data Features

```
In [ ]: # We Will Drop Records which have no BP and cholesterol Features
dataset.dropna(subset=['BP', 'Cholesterol'], inplace=True)

# Then We Will Fill Records has no Na_to_K Feature with Average of Na_to_K Values
dataset['Na_to_K'].fillna(dataset['Na_to_K'].mean(), inplace=True)
```

### Double-Check Missing Features

```
In [ ]: missing_values = dataset.isnull().sum() # Check for missing values in the dataset
print("Missing Values:\n", missing_values) # Print the count of missing values for each column
```

Missing Values:

|             |   |
|-------------|---|
| Age         | 0 |
| Sex         | 0 |
| BP          | 0 |
| Cholesterol | 0 |

```
Na_to_K      0  
Drug         0  
dtype: int64
```

## Split Dataset Into Feature set and Target set

In [ ]:

```
# Extract the feature matrix (X) and target variable (Y) from the dataset  
X = dataset.iloc[:, :-1].values # Features (all columns except the last one)  
Y = dataset.iloc[:, -1].values # Target variable (last column)  
  
# Display the feature matrix (X)  
print("Feature Matrix (X):\n", X)
```

Feature Matrix (X):

```
[[23 'F' 'HIGH' 'HIGH' 25.355]  
[47 'M' 'LOW' 'HIGH' 13.093]  
[47 'M' 'LOW' 'HIGH' 10.114]  
[28 'F' 'NORMAL' 'HIGH' 16.08220512820512]  
[61 'F' 'LOW' 'HIGH' 18.043]  
[22 'F' 'NORMAL' 'HIGH' 8.607]  
[49 'F' 'NORMAL' 'HIGH' 16.275]  
[41 'M' 'LOW' 'HIGH' 11.037]  
[60 'M' 'NORMAL' 'HIGH' 15.171]  
[47 'F' 'LOW' 'HIGH' 11.767]  
[34 'F' 'HIGH' 'NORMAL' 19.199]  
[43 'M' 'LOW' 'HIGH' 15.376]  
[74 'F' 'LOW' 'HIGH' 20.942]  
[50 'F' 'NORMAL' 'HIGH' 12.703]  
[16 'F' 'HIGH' 'NORMAL' 15.516]  
[69 'M' 'LOW' 'NORMAL' 11.455]  
[43 'M' 'HIGH' 'HIGH' 13.972]  
[23 'M' 'LOW' 'HIGH' 7.298]  
[32 'F' 'HIGH' 'NORMAL' 25.974]  
[57 'M' 'LOW' 'NORMAL' 19.128]  
[63 'M' 'NORMAL' 'HIGH' 25.917]  
[47 'M' 'LOW' 'NORMAL' 30.568]  
[48 'F' 'LOW' 'HIGH' 15.036]  
[33 'F' 'LOW' 'HIGH' 33.486]  
[28 'F' 'HIGH' 'NORMAL' 18.809]  
[31 'M' 'HIGH' 'HIGH' 30.366]  
[49 'F' 'NORMAL' 'NORMAL' 9.381]  
[39 'F' 'LOW' 'NORMAL' 22.697]  
[45 'M' 'LOW' 'HIGH' 17.951]
```

```
[18 'F' 'NORMAL' 'NORMAL' 8.75]
[74 'M' 'HIGH' 'HIGH' 9.567]
[49 'M' 'LOW' 'NORMAL' 11.014]
[65 'F' 'HIGH' 'NORMAL' 31.876]
[53 'M' 'NORMAL' 'HIGH' 14.133]
[46 'M' 'NORMAL' 'NORMAL' 7.285]
[32 'M' 'HIGH' 'NORMAL' 9.445]
[39 'M' 'LOW' 'NORMAL' 13.938]
[39 'F' 'NORMAL' 'NORMAL' 9.709]
[15 'M' 'NORMAL' 'HIGH' 9.084]
[73 'F' 'NORMAL' 'HIGH' 19.221]
[58 'F' 'HIGH' 'NORMAL' 14.239]
[50 'M' 'NORMAL' 'NORMAL' 15.79]
[23 'M' 'NORMAL' 'HIGH' 12.26]
[50 'F' 'NORMAL' 'NORMAL' 12.295]
[66 'F' 'NORMAL' 'NORMAL' 8.107]
[37 'F' 'HIGH' 'HIGH' 13.091]
[68 'M' 'LOW' 'HIGH' 10.291]
[23 'M' 'NORMAL' 'HIGH' 31.686]
[28 'F' 'LOW' 'HIGH' 19.796]
[58 'F' 'HIGH' 'HIGH' 19.416]
[67 'M' 'NORMAL' 'NORMAL' 10.898]
[62 'M' 'LOW' 'NORMAL' 27.183]
[24 'F' 'HIGH' 'NORMAL' 18.457]
[68 'F' 'HIGH' 'NORMAL' 10.189]
[26 'F' 'LOW' 'HIGH' 14.16]
[65 'M' 'HIGH' 'NORMAL' 11.34]
[40 'M' 'HIGH' 'HIGH' 27.826]
[60 'M' 'NORMAL' 'NORMAL' 10.091]
[34 'M' 'HIGH' 'HIGH' 18.703]
[38 'F' 'LOW' 'NORMAL' 29.875]
[24 'M' 'HIGH' 'NORMAL' 9.475]
[67 'M' 'LOW' 'NORMAL' 20.693]
[45 'M' 'LOW' 'NORMAL' 8.37]
[60 'F' 'HIGH' 'HIGH' 13.303]
[68 'F' 'NORMAL' 'NORMAL' 27.05]
[29 'M' 'HIGH' 'HIGH' 12.856]
[17 'M' 'NORMAL' 'NORMAL' 10.832]
[54 'M' 'NORMAL' 'HIGH' 24.658]
[18 'F' 'HIGH' 'NORMAL' 24.276]
[70 'M' 'HIGH' 'HIGH' 13.967]
[28 'F' 'NORMAL' 'HIGH' 19.675]
[24 'F' 'NORMAL' 'HIGH' 10.605]
[41 'F' 'NORMAL' 'NORMAL' 22.905]
```

```
[31 'M' 'HIGH' 'NORMAL' 17.069]
[26 'M' 'LOW' 'NORMAL' 20.909]
[36 'F' 'HIGH' 'HIGH' 11.198]
[26 'F' 'HIGH' 'NORMAL' 19.161]
[19 'F' 'HIGH' 'HIGH' 13.313]
[32 'F' 'LOW' 'NORMAL' 10.84]
[60 'M' 'HIGH' 'HIGH' 13.934]
[64 'M' 'NORMAL' 'HIGH' 7.761]
[32 'F' 'LOW' 'HIGH' 9.712]
[38 'F' 'HIGH' 'NORMAL' 11.326]
[47 'F' 'LOW' 'HIGH' 10.067]
[59 'M' 'HIGH' 'HIGH' 13.935]
[51 'F' 'NORMAL' 'HIGH' 13.597]
[69 'M' 'LOW' 'HIGH' 15.478]
[37 'F' 'HIGH' 'NORMAL' 23.091]
[50 'F' 'NORMAL' 'NORMAL' 17.211]
[62 'M' 'NORMAL' 'HIGH' 16.594]
[41 'M' 'HIGH' 'NORMAL' 15.156]
[29 'F' 'HIGH' 'HIGH' 29.45]
[42 'F' 'LOW' 'NORMAL' 29.271]
[56 'M' 'LOW' 'HIGH' 15.015]
[36 'M' 'LOW' 'NORMAL' 11.424]
[58 'F' 'LOW' 'HIGH' 38.247]
[56 'F' 'HIGH' 'HIGH' 25.395]
[20 'M' 'HIGH' 'NORMAL' 35.639]
[15 'F' 'HIGH' 'NORMAL' 16.725]
[31 'M' 'HIGH' 'NORMAL' 11.871]
[45 'F' 'HIGH' 'HIGH' 12.854]
[28 'F' 'LOW' 'HIGH' 13.127]
[56 'M' 'NORMAL' 'HIGH' 8.966]
[22 'M' 'HIGH' 'NORMAL' 28.294]
[37 'M' 'LOW' 'NORMAL' 8.968]
[22 'M' 'NORMAL' 'HIGH' 11.953]
[42 'M' 'LOW' 'HIGH' 20.013]
[72 'M' 'HIGH' 'NORMAL' 9.677]
[23 'M' 'NORMAL' 'HIGH' 16.85]
[50 'M' 'HIGH' 'HIGH' 7.49]
[47 'F' 'NORMAL' 'NORMAL' 6.683]
[35 'M' 'LOW' 'NORMAL' 9.17]
[65 'F' 'LOW' 'NORMAL' 13.769]
[20 'F' 'NORMAL' 'NORMAL' 9.281]
[51 'M' 'HIGH' 'HIGH' 18.295]
[67 'M' 'NORMAL' 'NORMAL' 9.514]
[40 'F' 'NORMAL' 'HIGH' 10.103]
```

```
[32 'F' 'HIGH' 'NORMAL' 10.292]
[61 'F' 'HIGH' 'HIGH' 25.475]
[28 'M' 'NORMAL' 'HIGH' 27.064]
[15 'M' 'HIGH' 'NORMAL' 17.206]
[36 'F' 'NORMAL' 'HIGH' 16.753]
[53 'F' 'HIGH' 'NORMAL' 12.495]
[19 'F' 'HIGH' 'NORMAL' 25.969]
[66 'M' 'HIGH' 'HIGH' 16.347]
[35 'M' 'NORMAL' 'NORMAL' 7.845]
[47 'M' 'LOW' 'NORMAL' 33.542]
[32 'F' 'NORMAL' 'HIGH' 7.477]
[70 'F' 'NORMAL' 'HIGH' 20.489]
[52 'M' 'LOW' 'NORMAL' 32.922]
[49 'M' 'LOW' 'NORMAL' 13.598]
[24 'M' 'NORMAL' 'HIGH' 25.786]
[42 'F' 'HIGH' 'HIGH' 21.036]
[74 'M' 'LOW' 'NORMAL' 11.939]
[55 'F' 'HIGH' 'HIGH' 10.977]
[35 'F' 'HIGH' 'HIGH' 12.894]
[51 'M' 'HIGH' 'NORMAL' 11.343]
[69 'F' 'NORMAL' 'HIGH' 10.065]
[49 'M' 'HIGH' 'NORMAL' 6.269]
[64 'F' 'LOW' 'NORMAL' 25.741]
[60 'M' 'HIGH' 'NORMAL' 8.621]
[74 'M' 'HIGH' 'NORMAL' 15.436]
[39 'M' 'HIGH' 'HIGH' 9.664]
[61 'M' 'NORMAL' 'HIGH' 9.443]
[37 'F' 'LOW' 'NORMAL' 12.006]
[26 'F' 'HIGH' 'NORMAL' 12.307]
[61 'F' 'LOW' 'NORMAL' 7.34]
[22 'M' 'LOW' 'HIGH' 8.151]
[49 'M' 'HIGH' 'NORMAL' 8.7]
[68 'M' 'HIGH' 'HIGH' 11.009]
[55 'M' 'NORMAL' 'NORMAL' 7.261]
[72 'F' 'LOW' 'NORMAL' 14.642]
[37 'M' 'LOW' 'NORMAL' 16.724]
[49 'M' 'LOW' 'HIGH' 10.537]
[31 'M' 'HIGH' 'NORMAL' 11.227]
[53 'M' 'LOW' 'HIGH' 22.963]
[59 'F' 'LOW' 'HIGH' 10.444]
[34 'F' 'LOW' 'NORMAL' 12.923]
[30 'F' 'NORMAL' 'HIGH' 10.443]
[57 'F' 'HIGH' 'NORMAL' 9.945]
[43 'M' 'NORMAL' 'NORMAL' 12.859]
```

```
[21 'F' 'HIGH' 'NORMAL' 28.632]
[16 'M' 'HIGH' 'NORMAL' 19.007]
[38 'M' 'LOW' 'HIGH' 18.295]
[58 'F' 'LOW' 'HIGH' 26.645]
[57 'F' 'NORMAL' 'HIGH' 14.216]
[51 'F' 'LOW' 'NORMAL' 23.003]
[20 'F' 'HIGH' 'HIGH' 11.262]
[28 'F' 'NORMAL' 'HIGH' 12.879]
[45 'M' 'LOW' 'NORMAL' 10.017]
[41 'F' 'LOW' 'NORMAL' 18.739]
[42 'M' 'HIGH' 'NORMAL' 12.766]
[73 'F' 'HIGH' 'HIGH' 18.348]
[48 'M' 'HIGH' 'NORMAL' 10.446]
[25 'M' 'NORMAL' 'HIGH' 19.011]
[39 'M' 'NORMAL' 'HIGH' 15.969]
[67 'F' 'NORMAL' 'HIGH' 15.891]
[22 'F' 'HIGH' 'NORMAL' 22.818]
[59 'F' 'NORMAL' 'HIGH' 13.884]
[20 'F' 'LOW' 'NORMAL' 11.686]
[36 'F' 'HIGH' 'NORMAL' 15.49]
[18 'F' 'HIGH' 'HIGH' 37.188]
[57 'F' 'NORMAL' 'NORMAL' 25.893]
[70 'M' 'HIGH' 'HIGH' 9.849]
[47 'M' 'HIGH' 'HIGH' 10.403]
[65 'M' 'HIGH' 'NORMAL' 34.997]
[64 'M' 'HIGH' 'NORMAL' 20.932]
[58 'M' 'HIGH' 'HIGH' 18.991]
[23 'M' 'HIGH' 'HIGH' 8.011]
[72 'M' 'LOW' 'HIGH' 16.31]
[72 'M' 'LOW' 'HIGH' 6.769]
[46 'F' 'HIGH' 'HIGH' 34.686]
[56 'F' 'LOW' 'HIGH' 11.567]
[16 'M' 'LOW' 'HIGH' 12.006]
[52 'M' 'NORMAL' 'HIGH' 9.894]
[40 'F' 'LOW' 'NORMAL' 11.349]]
```

## Encode Categorical Data in Features (One-Hot Encoding)

```
In [ ]:
# Define a ColumnTransformer for applying transformations to specific columns
# - 'encoder': OneHotEncoder is used for one-hot encoding
# - [1, 2, 3]: Columns 1, 2, and 3 are one-hot encoded, specified by their indices
# - 'remainder': 'passthrough' indicates that the remaining columns are passed through without any transformations
ct = ColumnTransformer(transformers=[('encoder', OneHotEncoder(), [1, 2, 3])], remainder='passthrough')
```

```
# Apply the transformations defined by ColumnTransformer to the feature matrix (X)
X = np.array(ct.fit_transform(X))
```

## Encode Target Data (Label Encoding)

```
In [ ]: le = LabelEncoder() # Use LabelEncoder to encode the target variable (Y)

Y = np.array(le.fit_transform(Y)) # Transform and overwrite the original target variable (Y) with encoded values

# Display the Target matrix (Y)
Y
```

```
Out[ ]: array([4, 2, 2, 3, 4, 3, 4, 2, 4, 2, 4, 4, 4, 3, 4, 3, 0, 2, 4, 4, 4, 4,
   4, 4, 4, 3, 4, 4, 3, 1, 3, 4, 3, 3, 0, 3, 3, 3, 3, 4, 1, 4, 3, 3,
   3, 0, 2, 4, 4, 4, 3, 4, 4, 1, 2, 1, 4, 3, 4, 4, 0, 4, 3, 1, 4, 0,
   3, 4, 4, 1, 4, 3, 4, 4, 4, 0, 4, 0, 3, 1, 3, 2, 0, 2, 1, 3, 4, 4,
   4, 4, 4, 4, 4, 3, 4, 4, 4, 4, 0, 0, 2, 3, 4, 3, 3, 4, 1, 4, 0,
   3, 3, 3, 3, 4, 3, 3, 0, 4, 4, 4, 4, 1, 4, 4, 3, 4, 3, 4, 4, 3, 4,
   4, 3, 1, 0, 1, 3, 0, 4, 1, 4, 0, 3, 3, 0, 3, 2, 0, 1, 3, 3, 4, 2,
   0, 4, 2, 3, 3, 1, 3, 4, 4, 4, 3, 4, 0, 3, 3, 4, 0, 4, 0, 4, 4,
   4, 4, 3, 3, 4, 4, 4, 1, 0, 4, 4, 0, 4, 2, 4, 2, 3, 3])
```

## 1st Experiment

```
In [ ]: # Define a list of random states which have high variance to ensure Randomization Factor
random_states = [1736, 0, 123, 789, 987, 654]

# Initialize Indicator Variables with -ve values to detect the maximum Accuracy, best experiment, Tree size of the best experiment
max_accuracy = -100
best_experiment = 0
best_treesize = -1

# Loop Over Random States
for i in range(5):

    # Split the dataset into training and testing sets using a random state
    X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.3, random_state=random_states[i])

    # Create and train a Decision Tree Classifier model
```

```

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

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

# Calculate and print the accuracy of the current experiment
accuracy = accuracy_score(y_test, y_pred)

# Update variables if the current experiment has higher accuracy
if accuracy >= max_accuracy:
    max_accuracy = accuracy
    best_experiment = i+1
    best_treesize = model.tree_.node_count

# Print Each Experiment with It's Accuracy and Tree Size
print(f"Experiment {i+1} with Tree Size: {model.tree_.node_count} and Accuracy: {accuracy}")

print("\n")
# Print Details of Experiment having best accuracy
print(f"Best Experiment: {best_experiment}\nAccuracy: {max_accuracy}\nTree Size: {best_treesize}")

```

Experiment 1 with Tree Size: 15 and Accuracy: 1.0  
Experiment 2 with Tree Size: 15 and Accuracy: 0.9661016949152542  
Experiment 3 with Tree Size: 15 and Accuracy: 0.9322033898305084  
Experiment 4 with Tree Size: 11 and Accuracy: 0.9661016949152542  
Experiment 5 with Tree Size: 15 and Accuracy: 0.9661016949152542

Best Experiment: 1  
Accuracy: 1.0  
Tree Size: 15

In [ ]:

```

def run_decision_tree_experiment(X, Y, random_states, trn_size):

    accuracies = []
    tree_sizes = []

    for i in range(len(random_states)):
        # Split the dataset into training and testing sets using a random state
        X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size = 1-trn_size, random_state=random_states[i])

        # Create and train a Decision Tree Classifier model

```

```

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

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

# Calculate accuracy of the current experiment
accuracy = accuracy_score(y_test, y_pred)
accuracies.append(accuracy)
tree_sizes.append(model.tree_.node_count)

# Return details of the best experiment as a tuple
return np.mean(accuracies), np.max(accuracies), np.min(accuracies), np.mean(tree_sizes), np.max(tree_sizes), np.min(tree_sizes)

```

## 2nd Experiment

In [ ]:

```

train_size = 0.3

# Initialize lists to store statistics
mean_accuracies = []
max_accuracies = []
min_accuracies = []
mean_tree_sizes = []
max_tree_sizes = []
min_tree_sizes = []

# Loop through different training set sizes using while loop
while train_size <= 0.7:

    a_mean, a_max, a_min, t_mean, t_max, t_min = run_decision_tree_experiment(X,Y,random_states,train_size)

    # Calculate mean, max, and min statistics for the current training set size
    mean_accuracies.append(a_mean) # Append Mean of each experiment with specific train_size
    max_accuracies.append(a_max)
    min_accuracies.append(a_min)
    mean_tree_sizes.append(t_mean)
    max_tree_sizes.append(t_max)
    min_tree_sizes.append(t_min)

    # Increment training_size
    train_size += 0.1

```

```
# Display the statistics
report = pd.DataFrame({
    'Training Set Size': [0.3, 0.4, 0.5, 0.6, 0.7],
    'Mean Accuracy': mean_accuracies,
    'Max Accuracy': max_accuracies,
    'Min Accuracy': min_accuracies,
    'Mean Tree Size': mean_tree_sizes,
    'Max Tree Size': max_tree_sizes,
    'Min Tree Size': min_tree_sizes
})

print(report)

# Define the size of the first plot (accuracy plot) to be 9 units in width and 5 units in height
plt.figure(figsize=(9, 5))

# Create a line plot for mean accuracy with black color and add a legend label
plt.plot([0.3, 0.4, 0.5, 0.6, 0.7], mean_accuracies, label='Mean Accuracy', color='black')

# Fill the area between the minimum and maximum accuracy curves with blue color and transparency, and add a legend label
plt.fill_between([0.3, 0.4, 0.5, 0.6, 0.7], min_accuracies, max_accuracies, alpha=0.3, color='blue', label='Accuracy Range')

# Set the title of the accuracy plot
plt.title('Accuracy vs Training Set Size')

# Label the x-axis as 'Training Set Size'
plt.xlabel('Training Set Size')

# Label the y-axis as 'Accuracy'
plt.ylabel('Accuracy')

# Display the legend in the plot
plt.legend()

# Show the accuracy plot
plt.show()

# Define the size of the second plot (tree size plot) to be 9 units in width and 5 units in height
plt.figure(figsize=(9, 5))

# Create a line plot for mean tree size with blue color and add a legend label
plt.plot([0.3, 0.4, 0.5, 0.6, 0.7], mean_tree_sizes, label='Mean Tree Size', color='blue')

# Fill the area between the minimum and maximum tree size curves with black color and transparency, and add a legend label
```

```
plt.fill_between([0.3, 0.4, 0.5, 0.6, 0.7], min_tree_sizes, max_tree_sizes, alpha=0.3, color='black', label='Tree Size Range')

# Set the title of the tree size plot
plt.title('Tree Size vs Training Set Size')

# Label the x-axis as 'Training Set Size'
plt.xlabel('Training Set Size')

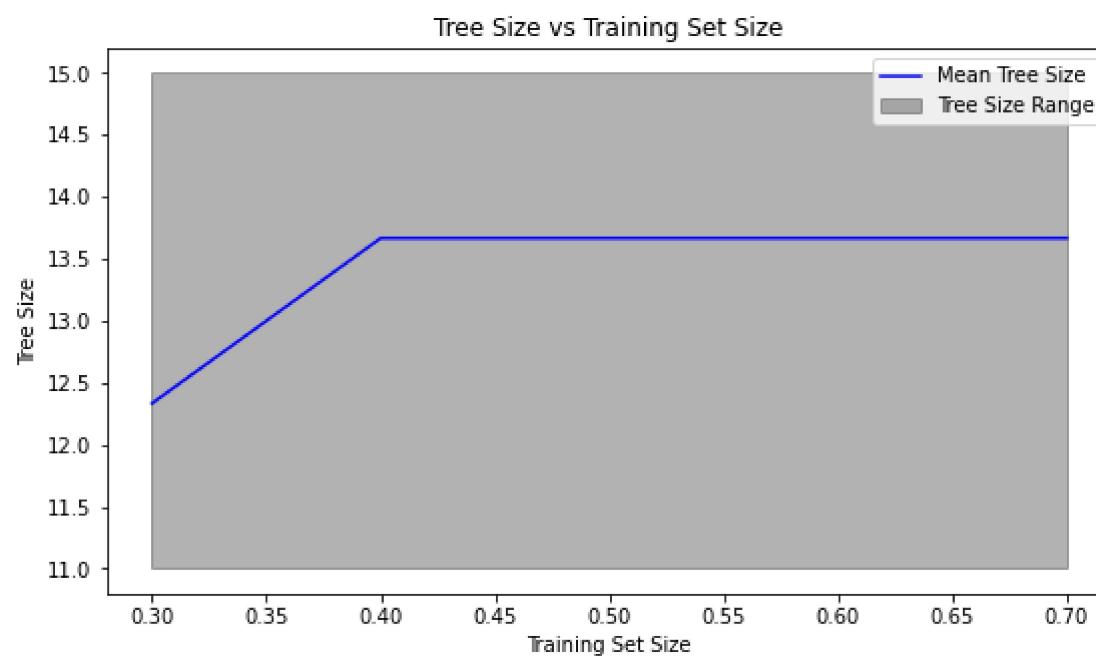
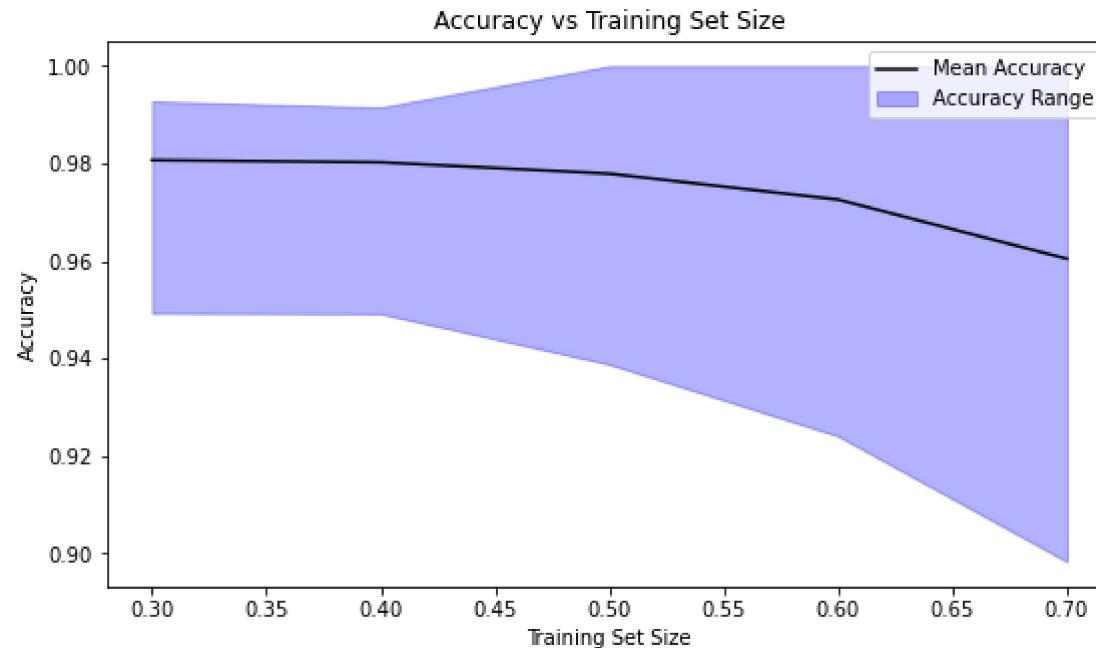
# Label the y-axis as 'Tree Size'
plt.ylabel('Tree Size')

# Display the Legend in the plot
plt.legend()

# Show the tree size plot
plt.show()
```

|   | Training Set Size | Mean Accuracy | Max Accuracy | Min Accuracy | \ |
|---|-------------------|---------------|--------------|--------------|---|
| 0 | 0.3               | 0.980676      | 0.992754     | 0.949275     |   |
| 1 | 0.4               | 0.980226      | 0.991525     | 0.949153     |   |
| 2 | 0.5               | 0.977891      | 1.000000     | 0.938776     |   |
| 3 | 0.6               | 0.972574      | 1.000000     | 0.924051     |   |
| 4 | 0.7               | 0.960452      | 1.000000     | 0.898305     |   |

|   | Mean Tree Size | Max Tree Size | Min Tree Size |
|---|----------------|---------------|---------------|
| 0 | 12.333333      | 15            | 11            |
| 1 | 13.666667      | 15            | 11            |
| 2 | 13.666667      | 15            | 11            |
| 3 | 13.666667      | 15            | 11            |
| 4 | 13.666667      | 15            | 11            |



# Problem 2

## KNN Class

In [ ]:

```
class KNN():

    def __init__(self) -> None:
        self.train=list()

    def __distance(self,testRow,trainRow):
        distance = 0.0
        for i in range(len(testRow)-1):
            distance += (testRow[i] - trainRow[i])**2
        return distance**(1/2)

    def __KDist(self,testRow, k):
        allDistances = list()
        for trainRow in self.train:
            dist = self.__distance(testRow, trainRow)
            allDistances.append((trainRow, dist))
        allDistances.sort(key=lambda tup: tup[1])
        neighborsDistances=allDistances[:k]
        return neighborsDistances #Tuples of k neighbors and their distances

    def predict(self, testRow, k):
        nwd = self.__KDist(testRow, k)
        total_weighted_votes = {}
        total_weights = 0

        for neighbor, dist in nwd:
            weight = 1 / (dist + 0.000001) #to not divide by 0
            total_weights += weight

            class_label = neighbor[-1]

            if class_label in total_weighted_votes:
                total_weighted_votes[class_label] += weight
            else:
                total_weighted_votes[class_label] = weight
```

```

normalized_votes = {label: weight / total_weights for label, weight in total_weighted_votes.items()} #Weight of every class
return max(normalized_votes, key=normalized_votes.get) #Max voted class

def evaluation(self, true, predicted):
    correctPredictions = 0

    for i in range(len(true)):
        if true[i] == predicted[i]:
            correctPredictions += 1
    return correctPredictions

def fit(self, train, test, k):
    self.train=train
    allPredicted = list()
    for row in test:
        prediction = self.predict(row, k)
        allPredicted.append(prediction)
    trueValues = [row[-1] for row in test]
    corr=self.evaluation(trueValues,allPredicted)
    print("Model fit successfully using:- \nK: {}\nCorrect predictions: {} \nTest set instances: {} \nAccuracy: {}%".format(k, co

```

## Preprocessing Class

In [ ]:

```

class preprocessing():

    def train_test_split(dataset, ratio):

        ratio_idx=round(1-ratio*len(dataset))
        train = dataset[:ratio_idx]
        test = dataset[ratio_idx:]
        return train, test

    def normalization(dataset):

        features = [list(map(float, row[:-1])) for row in dataset]
        labels = [row[-1] for row in dataset]

        features_transposed = list(map(list, zip(*features)))

        # Normalization min max

```

```

normalized_features = []
for feature_values in features_transposed:
    min_value = min(feature_values)
    max_value = max(feature_values)
    normalized_feature = [(float(value) - min_value) / (max_value - min_value) for value in feature_values]
    normalized_features.append(normalized_feature)

normalized_features = list(map(list, zip(*normalized_features)))

normalized_dataset = [normalized_feature + [label] for normalized_feature, label in zip(normalized_features, labels)]

return normalized_dataset

```

## Pandas Class

In [ ]:

```

class pd:

    def read_csv(filename, header=True):
        with open(filename, 'r') as f:
            if header:
                next(f)
            results = []
            for line in f:
                line = line.strip()
                words = line.split(',')
                results.append(words)

        return results

```

## Load The Dataset

In [ ]:

```
dataset=pd.read_csv('diabetes.csv')
```

**Normalize each feature column separately for training and test objects using Log Transformation or Min-Max Scaling**

In [ ]:

```
dataset=preprocessing.normalization(dataset)
```

## Split Dataset into Train and Test

```
In [ ]: train,test=preprocessing.train_test_split(dataset,0.3)
```

## Train The KNN Classifier

```
In [ ]: knn=KNN()  
knn.fit(train,test,5)
```

Model fit successfully using:-  
K: 5  
Correct predictions: 173  
Test set instances: 229  
Accuracy: 75.55%

## Train With Different values for K

```
In [ ]: for k in [3,6,9,12,14]:  
    knn.fit(train,test,k)  
    print('-' * 20)
```

Model fit successfully using:-  
K: 3  
Correct predictions: 175  
Test set instances: 229  
Accuracy: 76.42%

---

Model fit successfully using:-  
K: 6  
Correct predictions: 176  
Test set instances: 229  
Accuracy: 76.86%

---

Model fit successfully using:-  
K: 9  
Correct predictions: 171  
Test set instances: 229  
Accuracy: 74.67%

Model fit successfully using:-

K: 12

Correct predictions: 177

Test set instances: 229

Accuracy: 77.29%

---

Model fit successfully using:-

K: 14

Correct predictions: 178

Test set instances: 229

Accuracy: 77.73%

---