Nour Ayman Alaassar 20200406
Rawan Mostafa Mahmoud 20200183

In [1]:
```python
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
```

In [2]:
```python
# Data Preprocessing
Drugs_data = pd.read_csv("./drug.csv")
print(Drugs_data)

print("Data Types:\n", Drugs_data.dtypes)
print("----------------------------------------------------------------------")

print("Missing Values:\n", Drugs_data.isnull().sum())


Drugs_data_cleaned = Drugs_data.dropna()

# Separate features and targets
X = Drugs_data_cleaned.drop(columns=['Drug'])
y_status = Drugs_data_cleaned['Drug']

# Categorical feature encoding for X
encoder = LabelEncoder()
categorical_cols = X.select_dtypes(include=['object']).columns
for col in categorical_cols:
    X[col] = encoder.fit_transform(X[col])

# Categorical target encoding for y
le_status = LabelEncoder()
y_status_encoded = le_status.fit_transform(y_status)

# Numerical standardization
scaler = StandardScaler()
numerical_cols = X.select_dtypes(include=['float64', 'int64']).columns
X[numerical_cols] = scaler.fit_transform(X[numerical_cols])

# Separate into training and testing sets for both X and y_status
# X_train, X_test, y_status_train_encoded, y_status_test_encoded = train_test_split(
#     X, y_status_encoded, test_size=0.3, random_state=90)
# print("\nEncoded Training Data:\n", X_train)
# print("\nEncoded Testing Data:\n", X_test)

print("----------------------------------------------------------------------")
```

```
     Age Sex      BP Cholesterol  Na_to_K   Drug
0     23   F    HIGH        HIGH   25.355  drugY
1     47   M     LOW        HIGH   13.093  drugC
2     47   M     LOW        HIGH   10.114  drugC
3     28   F  NORMAL        HIGH      NaN  drugX
4     61   F     LOW        HIGH   18.043  drugY
..   ...  ..     ...         ...      ...    ...
195   56   F     LOW        HIGH   11.567  drugC
196   16   M     LOW        HIGH   12.006  drugC
197   52   M  NORMAL        HIGH    9.894  drugX
198   23   M  NORMAL         NaN   14.020  drugX
199   40   F     LOW      NORMAL   11.349  drugX

[200 rows x 6 columns]
Data Types:
 Age              int64
Sex             object
BP              object
Cholesterol     object
Na_to_K        float64
Drug            object
dtype: object
----------------------------------------------------------------------
Missing Values:
 Age            0
Sex            0
BP             2
Cholesterol    2
Na_to_K        1
Drug           0
dtype: int64
----------------------------------------------------------------------
```

In [3]:
```python
#experiment no.1
experiment_number = 5
treeSize = []
All_acuracies = []

for i in range(experiment_number):
    X_train, X_test, y_status_train_encoded, y_status_test_encoded = train_test_split(
    X, y_status_encoded, test_size=0.3, random_state=i*40)

    # Create Decision_Tree
    clf = DecisionTreeClassifier(random_state=40)

    clf.fit(X_train, y_status_train_encoded)

    # decision tree size
    size = clf.tree_.node_count
    treeSize.append(size)

    y_pred = clf.predict(X_test)

    # accuracy calculation
    accuracy_i = accuracy_score(y_status_test_encoded, y_pred)
    All_acuracies.append(accuracy_i)

    print(f"Experiment no. {i + 1}:")
    print(f"Size of Decision Tree: {size}")
    print(f"Accuracy equals: {accuracy_i:.2f}")
    print("----------------------------------------------------------------------")
```

```
Experiment no. 1:
Size of Decision Tree: 11
Accuracy equals: 0.98
----------------------------------------------------------------------
Experiment no. 2:
Size of Decision Tree: 11
Accuracy equals: 1.00
----------------------------------------------------------------------
Experiment no. 3:
Size of Decision Tree: 11
Accuracy equals: 0.98
----------------------------------------------------------------------
Experiment no. 4:
Size of Decision Tree: 11
Accuracy equals: 1.00
----------------------------------------------------------------------
Experiment no. 5:
Size of Decision Tree: 11
Accuracy equals: 0.98
----------------------------------------------------------------------
```

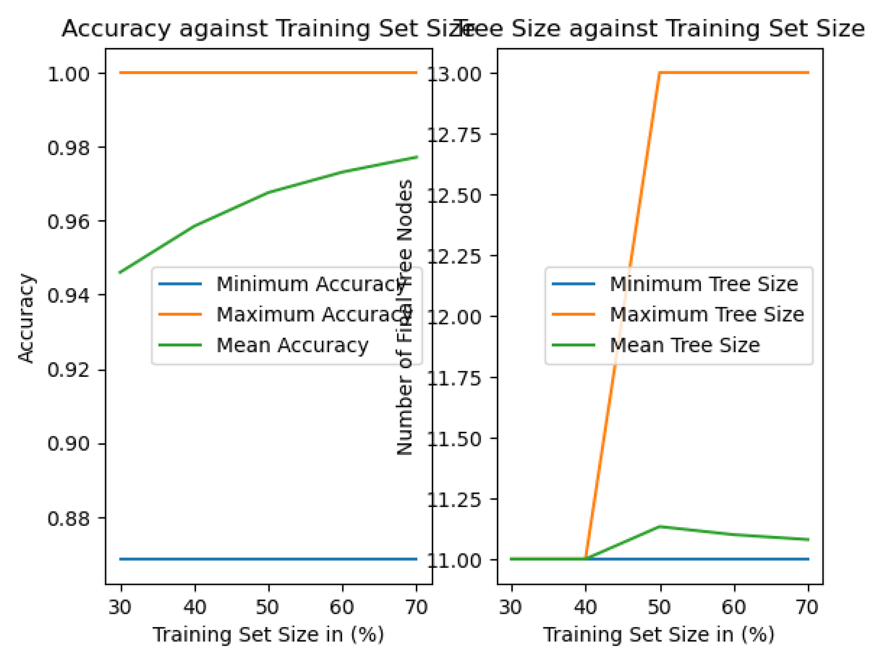
In [5]:
```python
#Best Performing
best_ofAll_experiment = All_acuracies.index(max(All_acuracies))
print(f" The Best Performing Model is fromExperiment number: {best_ofAll_experiment + 1}")
print(f"it's Size  : {treeSize[best_ofAll_experiment]}")
print(f"it's Accuracy : {All_acuracies[best_ofAll_experiment]:.2f}")


print("----------------------------------------------------------------------")
```

```
 The Best Performing Model is fromExperiment number: 2
it's Size  : 11
it's Accuracy : 1.00
----------------------------------------------------------------------
```

In [6]:
```python
#f experiment no.2
Accuracies_Mean = []
Accuracies_Maximum = []
Accuracies_Minimum= []
TreeSizes_Mean = []
TreeSizes_Maximum = []
TreeSizes_Minimum = []

SIze2 = []
All_accuracies2 = []

RandomSeeds_ = [42, 66, 33, 90, 125]
SplitingRRange = range(30, 80, 10)
for SplitRanges in SplitingRRange:
    SplitRanges /= 100.0
    TrainingSetSize=1-SplitRanges

    for seed_o in  RandomSeeds_:
        X_train, X_test, y_status_train_encoded, y_status_test_encoded = train_test_split(
            X, y_status_encoded, test_size=TrainingSetSize, random_state=seed_o)

        #Creation of DT
        clf = DecisionTreeClassifier(random_state=seed_o)
        clf.fit(X_train, y_status_train_encoded)

        # decision tree size
        size_i = clf.tree_.node_count
        SIze2.append(size_i)

        y_pred = clf.predict(X_test)

        # accuracy calculation
        accuracy_j = accuracy_score(y_status_test_encoded, y_pred)
        All_accuracies2.append(accuracy_j)


    Accuracies_Mean.append(np.mean(All_accuracies2))
    Accuracies_Maximum.append(np.max(All_accuracies2))
    Accuracies_Minimum.append(np.min(All_accuracies2))
    TreeSizes_Mean.append(np.mean(SIze2))
    TreeSizes_Maximum.append(np.max(SIze2))
    TreeSizes_Minimum.append(np.min(SIze2))

Reporting_ = pd.DataFrame({
    'TrainingSetSize': SplitingRRange,
    'MeanAccuracy': Accuracies_Mean,
    'MaximumAccuracy': Accuracies_Maximum,
    'MinimumAccuracy': Accuracies_Minimum,
    'MeanTreeSize': TreeSizes_Mean,
    'MaximumTreeSize': TreeSizes_Maximum,
    'MinimumTreeSize': TreeSizes_Minimum
})

print(Reporting_)
```

```
   TrainingSetSize  MeanAccuracy  MaximumAccuracy  MinimumAccuracy  \
0               30      0.945985              1.0         0.868613
1               40      0.958463              1.0         0.868613
2               50      0.967547              1.0         0.868613
3               60      0.973096              1.0         0.868613
4               70      0.977121              1.0         0.868613

   MeanTreeSize  MaximumTreeSize  MinimumTreeSize
0     11.000000               11               11
1     11.000000               11               11
2     11.133333               13               11
3     11.100000               13               11
4     11.080000               13               11
```

In [7]:
```python
#plotting



plt.subplot(1, 2, 1)
plt.title('Accuracy against Training Set Size')
plt.xlabel('Training Set Size in (%)')
plt.ylabel('Accuracy')
plt.plot(SplitingRRange, Accuracies_Minimum, label='Minimum Accuracy')
plt.plot(SplitingRRange, Accuracies_Maximum, label='Maximum Accuracy')
plt.plot(SplitingRRange, Accuracies_Mean, label='Mean Accuracy')



plt.legend()

plt.subplot(1, 2, 2)
plt.title('Tree Size against Training Set Size')
plt.xlabel('Training Set Size in (%)')
plt.ylabel('Number of Final Tree Nodes')

plt.plot(SplitingRRange, TreeSizes_Minimum, label='Minimum Tree Size')
plt.plot(SplitingRRange, TreeSizes_Maximum, label='Maximum Tree Size')
plt.plot(SplitingRRange, TreeSizes_Mean, label='Mean Tree Size')




plt.legend()


plt.show()

print("------------------------------------------------")
```




```
------------------------------------------------
```

In [12]:
```python
# Load the diabetes dataset
df = pd.read_csv('diabetes.csv')

# Function for Min-Max Scaling
def min_max_scaling(data):
    min_vals = np.min(data, axis=0)
    max_vals = np.max(data, axis=0)
    scaled_data = (data - min_vals) / (max_vals - min_vals)
    return scaled_data
```

In [13]:
```python
# Function for computing Euclidean distance
def euclidean_distance(x1, x2):
    return np.sqrt(np.sum((x1 - x2)**2))

# Function to perform KNN classification
def knn_classify(train_data, train_labels, test_instance, k):
    distances = np.zeros(len(train_data))

    for i in range(len(train_data)):
        distances[i] = euclidean_distance(train_data[i], test_instance)

    # Get indices of k nearest neighbors
    k_neighbors_indices = np.argsort(distances)[:k]

    # Use Distance-Weighted Voting to break ties
    class_votes = {}
    for idx in k_neighbors_indices:
        label = train_labels[idx]
        weight = 1 / distances[idx]  # Inverse of distance
        class_votes[label] = class_votes.get(label, 0) + weight

    # Return the class with the highest weighted votes
    return max(class_votes, key=class_votes.get)

# Function to split data into training and testing sets
def train_test_split(data, labels, split_ratio=0.7):
    split_index = int(split_ratio * len(data))
    train_data, test_data = data[:split_index], data[split_index:]
    train_labels, test_labels = labels[:split_index], labels[split_index:]
    return train_data, train_labels, test_data, test_labels

# Function to evaluate KNN with different k values
def evaluate_knn(data, labels, k_values):
    accuracies = []

    for k in k_values:
        correct_classifications = 0

        for i in range(len(test_data)):
            predicted_label = knn_classify(train_data, train_labels, test_data[i], k)
            if predicted_label == test_labels[i]:
                correct_classifications += 1

        accuracy = correct_classifications / len(test_data) * 100
        accuracies.append(accuracy)

        print(f'k value: {k}\nNumber of correctly classified instances: {correct_classifications}\n'
              f'Total number of instances: {len(test_data)}\nAccuracy: {accuracy:.2f}%\n')

    # Calculate and print average accuracy
    avg_accuracy = np.mean(accuracies)
    print(f'Average Accuracy Across All Iterations: {avg_accuracy:.2f}%')
```

In [14]:
```python
# Extract features and labels from the dataset
features = df.drop('Outcome', axis=1).values
labels = df['Outcome'].values

# Normalize features using Min-Max Scaling
scaled_features = min_max_scaling(features)

# Split data into training and testing sets
train_data, train_labels, test_data, test_labels = train_test_split(scaled_features, labels)

# Define k values for iterations
k_values = [26, 27, 28]

# Evaluate KNN with different k values
evaluate_knn(test_data, test_labels, k_values)
```

```
k value: 26
Number of correctly classified instances: 183
Total number of instances: 231
Accuracy: 79.22%

k value: 27
Number of correctly classified instances: 181
Total number of instances: 231
Accuracy: 78.35%

k value: 28
Number of correctly classified instances: 181
Total number of instances: 231
Accuracy: 78.35%

Average Accuracy Across All Iterations: 78.64%
```

In [ ]: