Name	ID
Nore Mohamed Hussein	20201196
Kholoud Mohamed Alkamkhli	20200846
Mariam Mohamed Elmoazen	20200528
Heba Abdelwahab Sayed Abdelwahab	20201208

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.utils import shuffle
from sklearn.linear_model import LinearRegression
from sklearn import tree

data = pd.read_csv("drug.csv")
print ("\ndata\n",data.head(20))
```

```
data
     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
5    22   F  NORMAL        HIGH    8.607  drugX
6    49   F  NORMAL        HIGH   16.275  drugY
7    41   M     LOW        HIGH   11.037  drugC
8    60   M  NORMAL        HIGH   15.171  drugY
9    43   M     NaN      NORMAL   19.368  drugY
10   47   F     LOW        HIGH   11.767  drugC
11   34   F    HIGH      NORMAL   19.199  drugY
12   43   M     LOW        HIGH   15.376  drugY
13   74   F     LOW        HIGH   20.942  drugY
14   50   F  NORMAL        HIGH   12.703  drugX
15   16   F    HIGH      NORMAL   15.516  drugY
16   69   M     LOW      NORMAL   11.455  drugX
17   43   M    HIGH        HIGH   13.972  drugA
18   23   M     LOW        HIGH    7.298  drugC
19   32   F    HIGH      NORMAL   25.974  drugY
```

```
datanull= data.isnull().any(axis = 1)
data[datanull]
count_null_rows = datanull.sum()

print("Number of rows with at least one null value:", count_null_rows)
```

```
Number of rows with at least one null value: 5
```

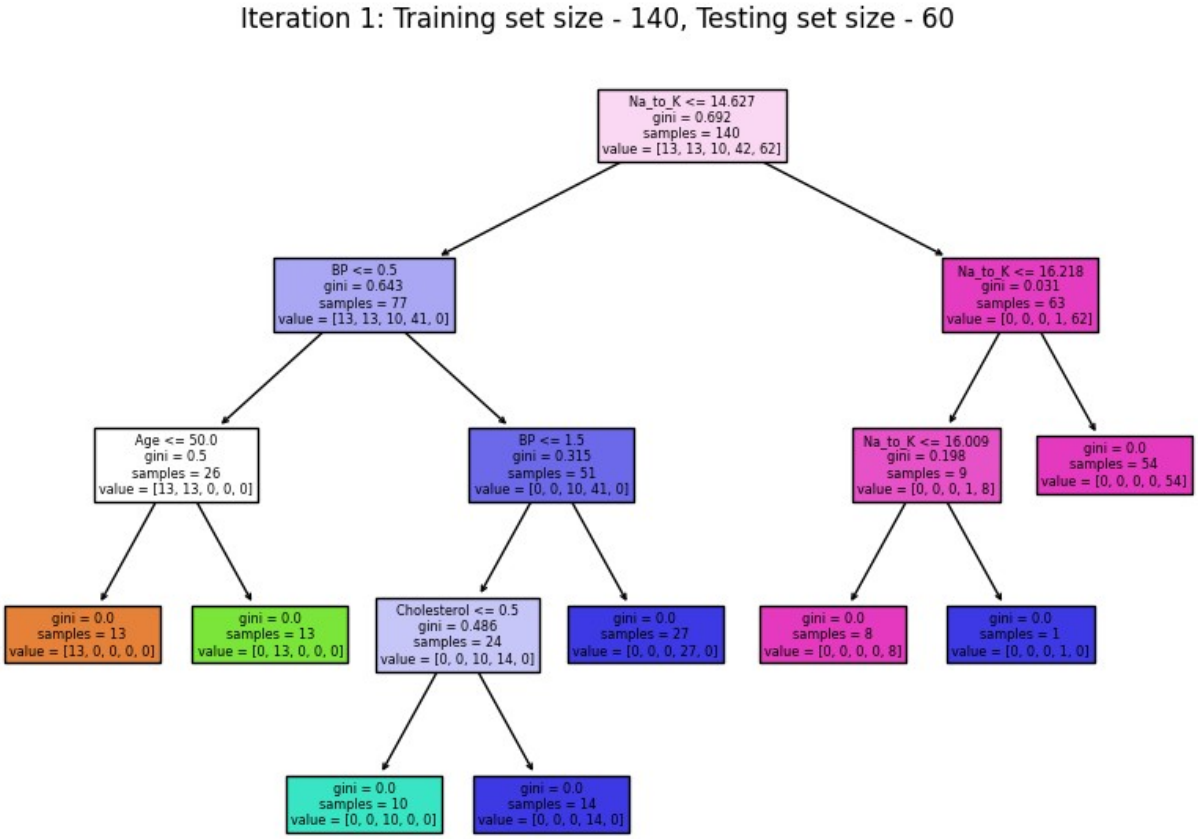
```
categorical_columns = data.select_dtypes(include='object').columns
numerical_columns = data.select_dtypes(exclude='object').columns
#replace categorical missing values with the mode and the numercial
missing values with the mean
```

```
data[categorical_columns] =
data[categorical_columns].fillna(data[categorical_columns].mode().iloc
[0])

data[numerical_columns] =
data[numerical_columns].fillna(data[numerical_columns].mean())

# we notice that missing values have been replaced
print ("\ndata\n",data.head(20))


data
     Age Sex      BP Cholesterol    Na_to_K   Drug
0    23   F    HIGH        HIGH  25.355000  drugY
1    47   M     LOW        HIGH  13.093000  drugC
2    47   M     LOW        HIGH  10.114000  drugC
3    28   F  NORMAL        HIGH  16.126126  drugX
4    61   F     LOW        HIGH  18.043000  drugY
5    22   F  NORMAL        HIGH   8.607000  drugX
6    49   F  NORMAL        HIGH  16.275000  drugY
7    41   M     LOW        HIGH  11.037000  drugC
8    60   M  NORMAL        HIGH  15.171000  drugY
9    43   M    HIGH      NORMAL  19.368000  drugY
10   47   F     LOW        HIGH  11.767000  drugC
11   34   F    HIGH      NORMAL  19.199000  drugY
12   43   M     LOW        HIGH  15.376000  drugY
13   74   F     LOW        HIGH  20.942000  drugY
14   50   F  NORMAL        HIGH  12.703000  drugX
15   16   F    HIGH      NORMAL  15.516000  drugY
16   69   M     LOW      NORMAL  11.455000  drugX
17   43   M    HIGH        HIGH  13.972000  drugA
18   23   M     LOW        HIGH   7.298000  drugC
19   32   F    HIGH      NORMAL  25.974000  drugY

categorical_columns = data.select_dtypes(include='object').columns
numerical_columns = data.select_dtypes(exclude='object').columns

sex_encoder = LabelEncoder()
bp_encoder = LabelEncoder()
cholesterol_encoder = LabelEncoder()
# Fit and transform the categorical variables using the label encoders
data['Sex'] = sex_encoder.fit_transform(data['Sex'])
data['BP'] = bp_encoder.fit_transform(data['BP'])
data['Cholesterol'] =
cholesterol_encoder.fit_transform(data['Cholesterol'])

print ("\ndata\n",data.head(20))


data
     Age  Sex  BP  Cholesterol    Na_to_K   Drug
```

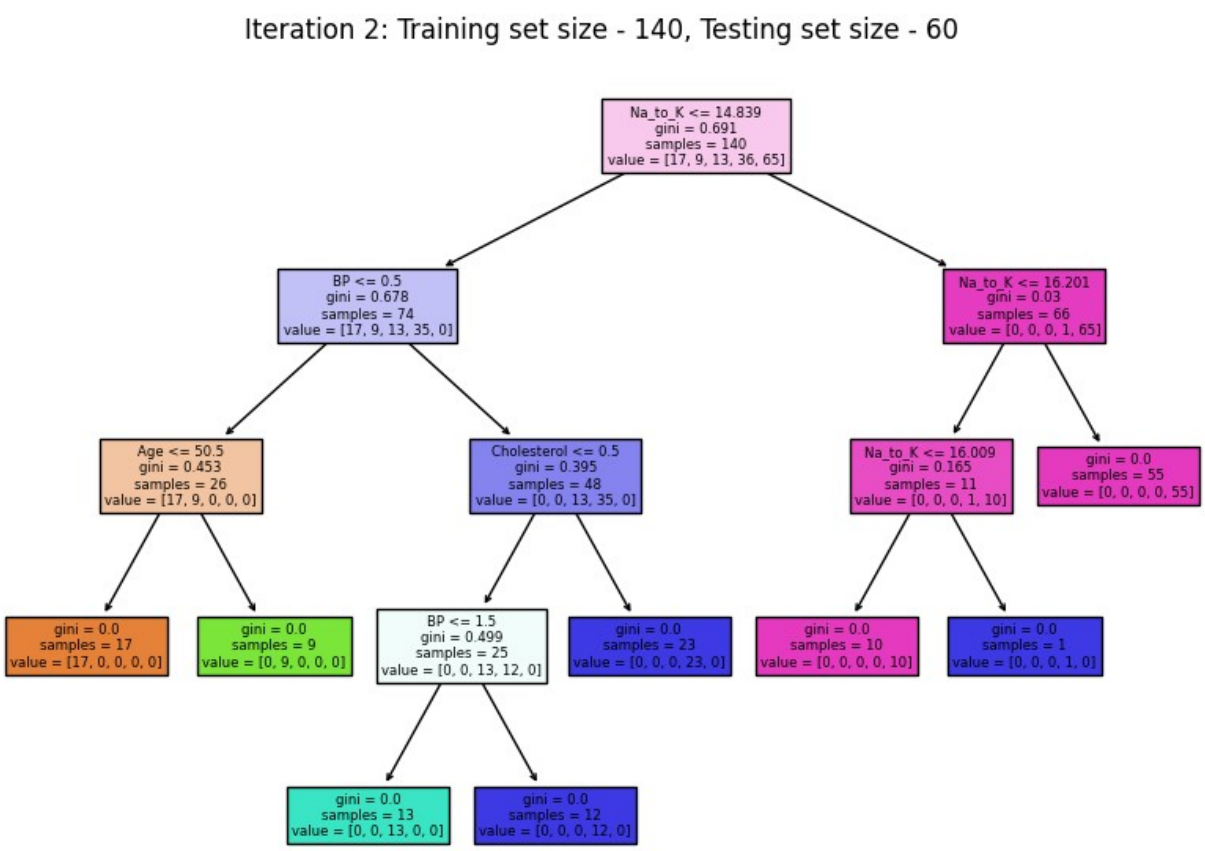
```
0    23    0   0            0  25.355000  drugY
1    47    1   1            0  13.093000  drugC
2    47    1   1            0  10.114000  drugC
3    28    0   2            0  16.126126  drugX
4    61    0   1            0  18.043000  drugY
5    22    0   2            0   8.607000  drugX
6    49    0   2            0  16.275000  drugY
7    41    1   1            0  11.037000  drugC
8    60    1   2            0  15.171000  drugY
9    43    1   0            1  19.368000  drugY
10   47    0   1            0  11.767000  drugC
11   34    0   0            1  19.199000  drugY
12   43    1   1            0  15.376000  drugY
13   74    0   1            0  20.942000  drugY
14   50    0   2            0  12.703000  drugX
15   16    0   0            1  15.516000  drugY
16   69    1   1            1  11.455000  drugX
17   43    1   0            0  13.972000  drugA
18   23    1   1            0   7.298000  drugC
19   32    0   0            1  25.974000  drugY
```

```
x = data.drop(columns=['Drug'])
y = data[['Drug']]

from sklearn.model_selection import train_test_split

import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score

bestAccuracy = 0
for i in range(5):
    X_train, X_test, y_train, y_test = train_test_split(x, y,
test_size=0.3, random_state=np.random.randint(100 + 2*i))

    clf = tree.DecisionTreeClassifier()

    clf = clf.fit(X_train, y_train)

    plt.figure(figsize=(10, 7))
    tree.plot_tree(clf, filled=True, feature_names=X_train.columns)
    plt.title(f"Iteration {i+1}: Training set size - {len(X_train)},
Testing set size - {len(X_test)}")
    plt.show()
    num_nodes = clf.tree_.node_count
    num_leaves = clf.get_n_leaves()
    tree_depth = clf.get_depth()
    y_pred = clf.predict(X_test)

    accuracy = accuracy_score(y_test, y_pred)*100
    if accuracy>bestAccuracy:
```

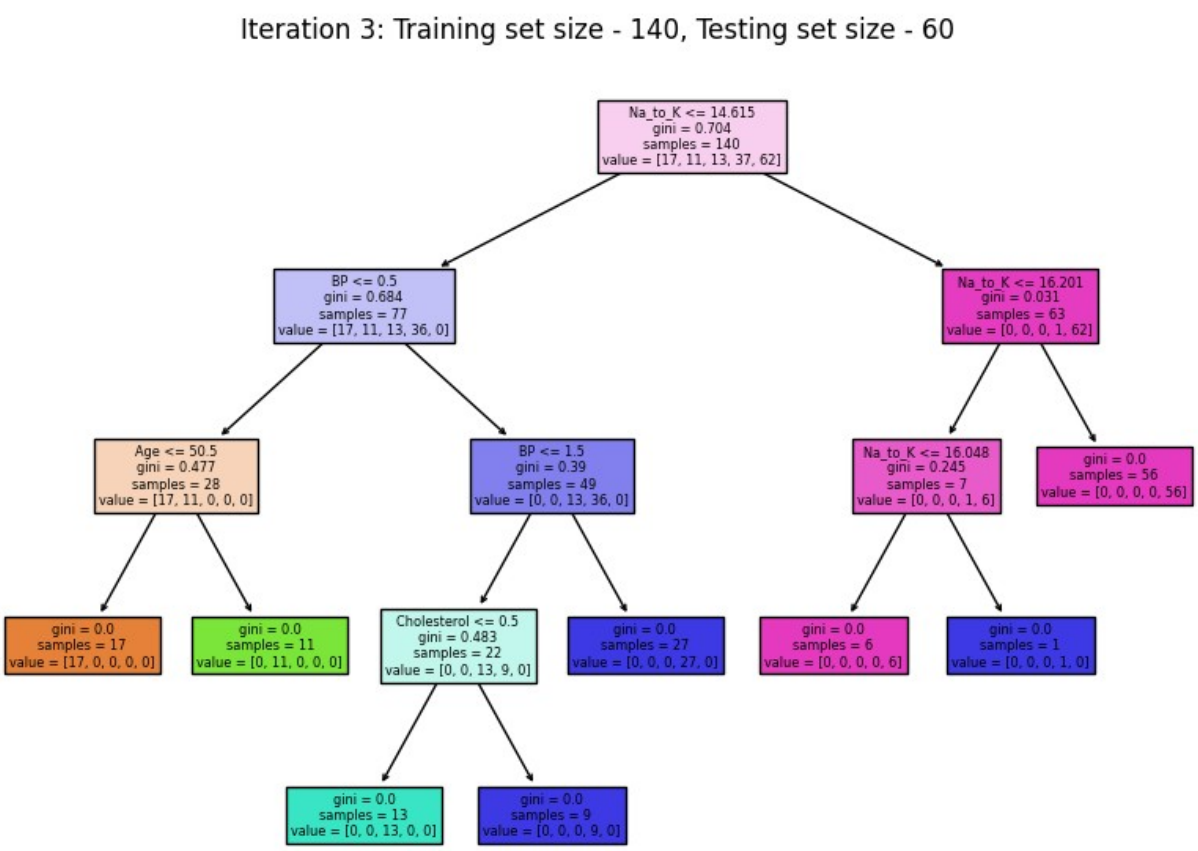
```
        bestAccuracy = accuracy
        clfBest = tree.DecisionTreeClassifier()
        clfBest = clf
    print(f"Iteration {i+1}: Accuracy - {accuracy:.4f}, Number of
nodes - {num_nodes}, Number of leaves - {num_leaves}, Tree depth -
{tree_depth}")
print(f"Best Accuracy: {bestAccuracy}")
plt.figure(figsize=(10, 7))
tree.plot_tree(clfBest, filled=True, feature_names=X_train.columns)
plt.title(f"Best Decision tree")
plt.show()
```



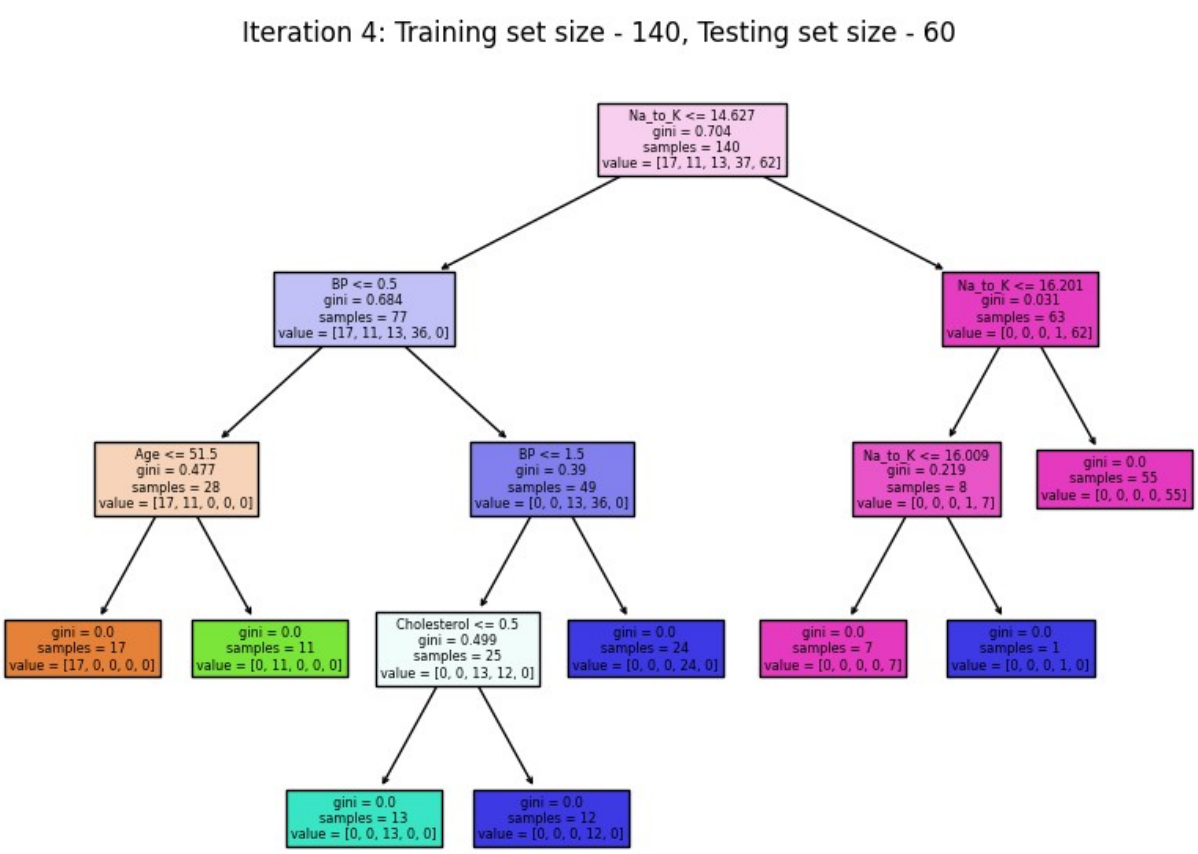

```
Iteration 1: Accuracy - 98.3333, Number of nodes - 15, Number of
leaves - 8, Tree depth - 4
```



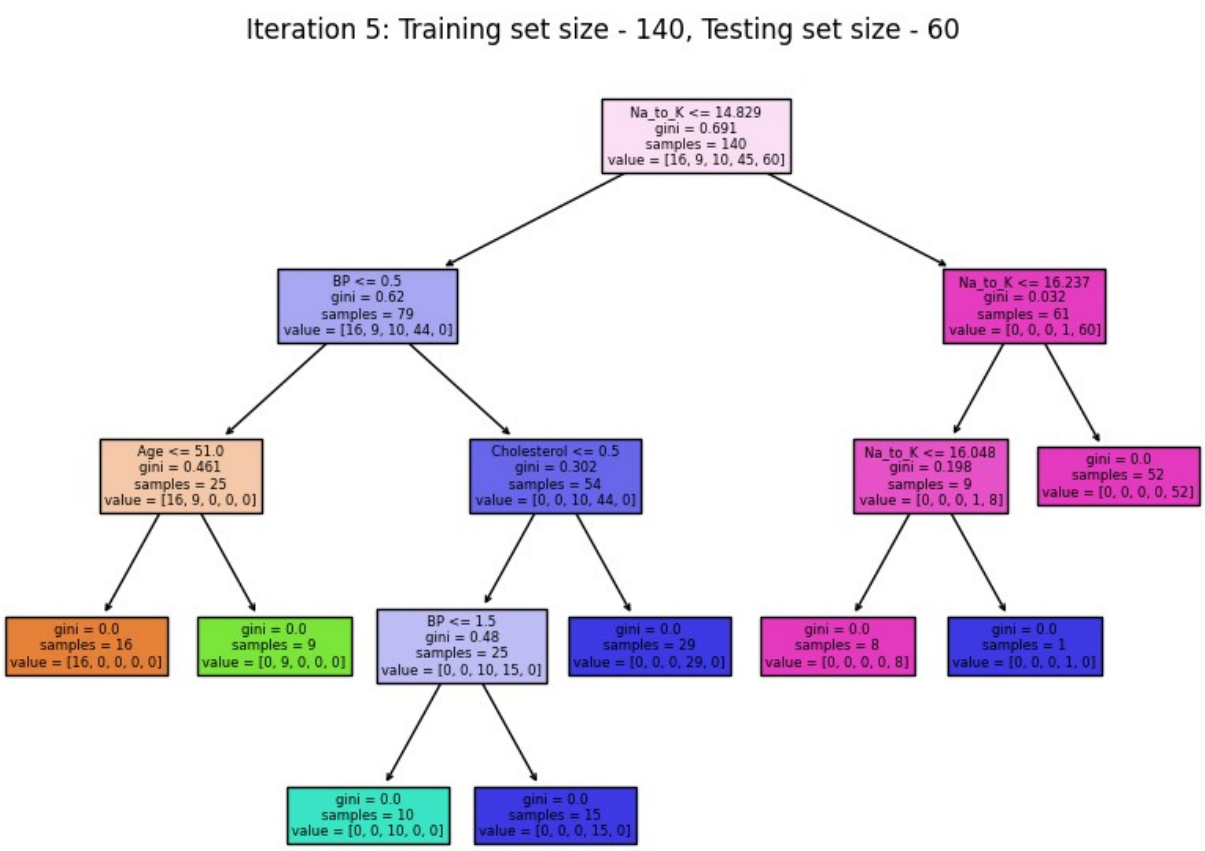

```
Iteration 2: Accuracy - 100.0000, Number of nodes - 15, Number of
leaves - 8, Tree depth - 4
```



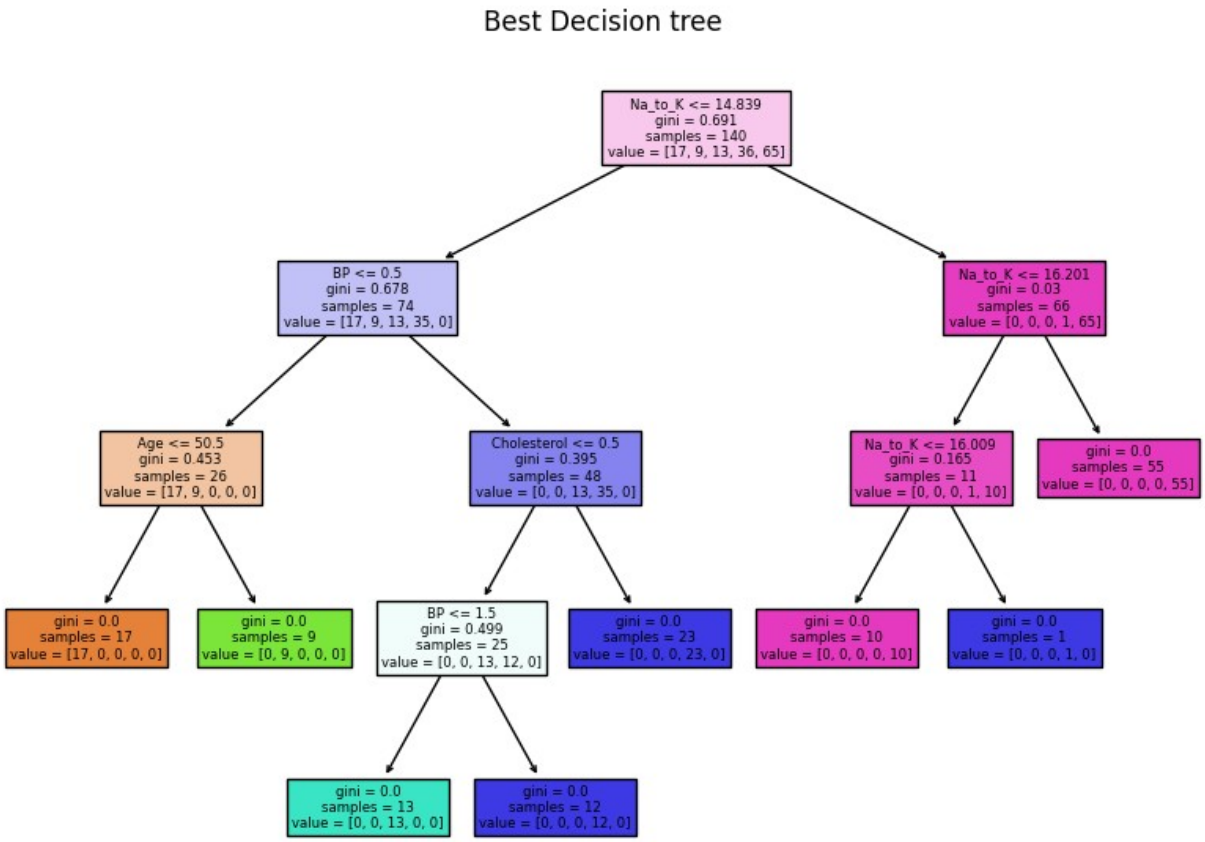

```
Iteration 3: Accuracy - 98.3333, Number of nodes - 15, Number of
leaves - 8, Tree depth - 4
```




```
Iteration 4: Accuracy - 96.6667, Number of nodes - 15, Number of
leaves - 8, Tree depth - 4
```




```
Iteration 5: Accuracy - 98.3333, Number of nodes - 15, Number of
leaves - 8, Tree depth - 4
Best Accuracy: 100.0
```

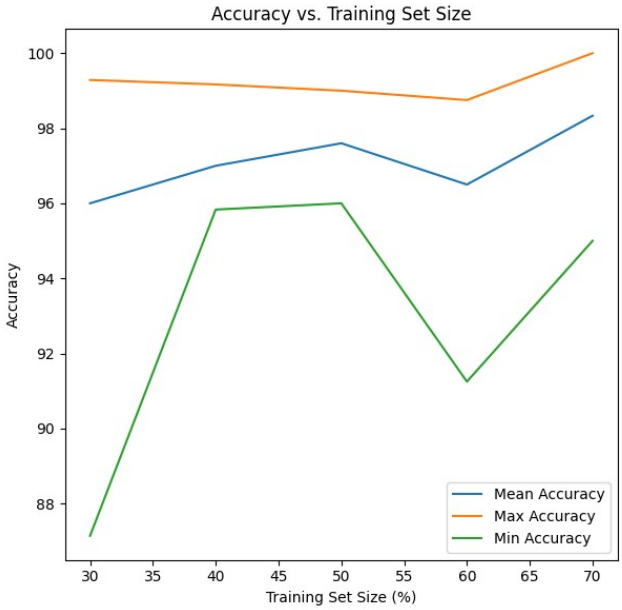


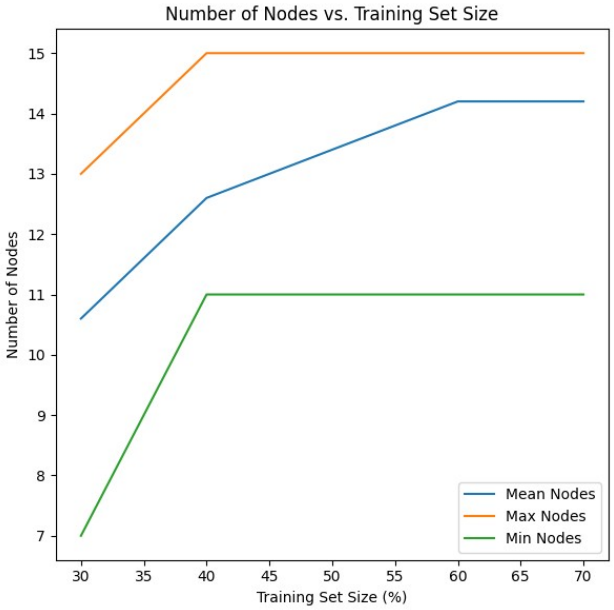

```
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from fpdf import FPDF
from tabulate import tabulate
import matplotlib.backends.backend_pdf as pdf_backend
training_sizes = np.arange(0.3, 0.8, 0.1)

mean_accuracies = []
max_accuracies = []
min_accuracies = []
mean_nodes = []
max_nodes = []
min_nodes = []

for train_size in training_sizes:
    accuracies = []
    nodes = []

    for i in range(5):
        X_train, X_test, y_train, y_test = train_test_split(x, y,
test_size=(1 - train_size), random_state=np.random.randint(100))
```

```
        clf = DecisionTreeClassifier()

        clf.fit(X_train, y_train)

        y_pred = clf.predict(X_test)

        accuracy = accuracy_score(y_test, y_pred)
        num_nodes = clf.tree_.node_count

        # Append to the lists
        accuracies.append(accuracy)
        nodes.append(num_nodes)

    mean_accuracies.append(np.mean(accuracies)*100)
    max_accuracies.append(np.max(accuracies)*100)
    min_accuracies.append(np.min(accuracies)*100)

    mean_nodes.append(np.mean(nodes))
    max_nodes.append(np.max(nodes))
    min_nodes.append(np.min(nodes))

report_data = {
    'Training Size': training_sizes,
    'Mean Accuracy': mean_accuracies,
    'Max Accuracy': max_accuracies,
    'Min Accuracy': min_accuracies,
    'Mean Nodes': mean_nodes,
    'Max Nodes': max_nodes,
    'Min Nodes': min_nodes
}
#save report to excel sheet
report_df = pd.DataFrame(report_data)

report_df.to_excel('report.xlsx', index=False, sheet_name='report')
#save report to pdf
latex_table = data.to_latex(index=False)
with pdf_backend.PdfPages("report.pdf") as pdf:
    fig, ax = plt.subplots(figsize=(12, 6))
    ax.axis('off')
    ax.table(cellText=report_df.values, colLabels=report_df.columns,
cellLoc='center', loc='center')
    pdf.savefig(fig, bbox_inches='tight')

import matplotlib.pyplot as plt

plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)
plt.plot(training_sizes * 100, mean_accuracies, label='Mean Accuracy')
```

```
plt.plot(training_sizes * 100, max_accuracies, label='Max Accuracy')
plt.plot(training_sizes * 100, min_accuracies, label='Min Accuracy')
plt.xlabel('Training Set Size (%)')
plt.ylabel('Accuracy')
plt.title('Accuracy vs. Training Set Size')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(training_sizes * 100, mean_nodes, label='Mean Nodes')
plt.plot(training_sizes * 100, max_nodes, label='Max Nodes')
plt.plot(training_sizes * 100, min_nodes, label='Min Nodes')
plt.xlabel('Training Set Size (%)')
plt.ylabel('Number of Nodes')
plt.title('Number of Nodes vs. Training Set Size')
plt.legend()

plt.tight_layout()
plt.show()
```

Training Size	Mean Accuracy	Max Accuracy	Min Accuracy	Mean Nodes	Max Nodes	Min Nodes
0.3	96.0	99.28571428571429	87.14285714285714	10.6	13.0	7.0
0.4	97.00000000000001	99.16666666666667	95.83333333333334	12.6	15.0	11.0
0.5	97.6	99.0	96.0	13.4	15.0	11.0
0.6000000000000001	96.50000000000001	98.75	91.25	14.2	15.0	11.0
0.7000000000000002	98.33333333333331	100.0	95.0	14.2	15.0	11.0


Accuracy vs. Training Set Size
Accuracy
Training Set Size (%)
Mean Accuracy
Max Accuracy
Min Accuracy


Number of Nodes vs. Training Set Size
Number of Nodes
Training Set Size (%)
Mean Nodes
Max Nodes
Min Nodes

```
import pandas
import numpy as np
import matplotlib.pyplot as plot
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler

data = pandas.read_csv('diabetes.csv')
print(data.head())
# print(data.shape[0])
```

```
   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin
BMI  \
0            6      148             72             35        0  33.6

1            1       85             66             29        0  26.6

2            8      183             64              0        0  23.3

3            1       89             66             23       94  28.1

4            0      137             40             35      168  43.1


   DiabetesPedigreeFunction  Age  Outcome
0                     0.627   50        1
1                     0.351   31        0
2                     0.672   32        1
3                     0.167   21        0
4                     2.288   33        1
```

Data preprocessing

1. Removing missing values

```
def RemoveMissingValue(data):
    datanull= data.isnull().any(axis = 1)  #remove row with missing
value
    return data[~datanull]
data = RemoveMissingValue(data)
print(data.shape[0])
```

```
767
```

1. splitting data

```
x = data.drop('Outcome',axis=1)
y = data['Outcome']
x_train, x_test, y_train, y_test = train_test_split(x, y,
test_size=0.3, random_state=100)
print("x_train\n ",x_train.head()) , print("x_test\
n",x_test.head()),print("y_train\n",y_train.head()),print("y test\
```

```
n",y_test.head())
print(data.shape[0])
print(x_train.shape[0])
print(x_test.shape[0])

x_train
      Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin
BMI  \
155             7      152             88             44        0  50.0

150             1      136             74             50      204  37.4

78              0      131              0              0        0  43.2

9               8      125             96              0        0   0.0

142             2      108             52             26       63  32.5


     DiabetesPedigreeFunction  Age
155                     0.337   36
150                     0.399   24
78                      0.270   26
9                       0.232   54
142                     0.318   22
x_test
      Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin
BMI  \
173             1       79             60             42       48  43.5

253             0       86             68             32        0  35.8

207             5      162            104              0        0  37.7

433             2      139             75              0        0  25.6

191             9      123             70             44       94  33.1


     DiabetesPedigreeFunction  Age
173                     0.678   23
253                     0.238   25
207                     0.151   52
433                     0.167   29
191                     0.374   40
y_train
 155    1
150    0
78     1
9      1
142    0
```

```
Name: Outcome, dtype: int64
y test
 173    0
253    0
207    1
433    0
191    0
Name: Outcome, dtype: int64
767
536
231
```

1. data normalization

```
min = x_train.min(axis=0)
max = x_train.max(axis=0)
x_train = (x_train - min) / (max- min)
x_test = (x_test - min) / (max - min)

x_train = x_train.to_numpy().reshape((-1,8))
x_test = x_test.to_numpy().reshape((-1,8))
y_train = y_train.to_numpy()
y_test = y_test.to_numpy()


print("x_train\n",x_train)
print("x_test\n",x_test)
```

```
x_train
 [[0.46666667 0.7638191  0.77192982 ... 0.74515648 0.11058924 0.25
]
 [0.06666667 0.68341709 0.64912281 ... 0.55737705 0.13706234 0.05
]
 [0.         0.65829146 0.         ... 0.6438152  0.08198121
0.08333333]
 ...
 [0.26666667 0.46231156 0.70175439 ... 0.62891207 0.06789069
0.13333333]
 [0.13333333 0.56281407 0.57894737 ... 0.37257824 0.09777968 0.05
]
 [0.13333333 0.34170854 0.61403509 ... 0.37257824 0.04654142
0.06666667]]
x_test
 [[0.06666667 0.39698492 0.52631579 ... 0.64828614 0.25619129
0.03333333]
 [0.         0.4321608  0.59649123 ... 0.53353204 0.06831768
0.06666667]
 [0.33333333 0.81407035 0.9122807  ... 0.56184799 0.03116994
0.51666667]
 ...
```

```
 [0.53333333 0.77889447 0.54385965 ... 0.50670641 0.19854825
0.41666667]
 [0.26666667 0.65829146 0.59649123 ... 0.49329359 0.03501281
0.11666667]
 [0.06666667 0.55778894 0.8245614  ... 0.48882265 0.07984629 0.4
]]
```

KNN Algorithm

```
def euclidean_distance(x1, x2):
    return np.sqrt(np.sum(x1-x2)**2)

class KNN:
    def __init__(self, k=3):
        self.k = k

    def fit(self, X, y):
        self.X_train = X
        self.y_train = y

    def predict(self, X):
        predictions = [self._predict(x) for x in X]
        return predictions

    def _predict(self, x):
        # Compute the distances
        distances = [euclidean_distance(x, x_train) for x_train in
self.X_train]
        # print(distances)

        # Get indices of the closest k neighbors
        k_indices = np.argsort(distances)[:self.k]
        k_nearest_labels = [self.y_train[i] for i in k_indices]

        # Extract distances for the k nearest neighbors
        k_distances = [distances[i] for i in k_indices]

        # Compute weighted distances for the k nearest neighbors
        k_weighted_distances = [1/d if d != 0 else float('inf') for d
in k_distances]

        # Count the occurrences of each label in the k nearest
neighbors
        label_counts = np.bincount(k_nearest_labels)
        final_prediction = label_counts.argmax()

        # Check for ties
        if (label_counts == label_counts[final_prediction]).sum() > 1:
```

```
            # Tie detected, resolve using weighted distances
            sorted_indices_by_weight = np.argsort([-w for w in
k_weighted_distances], kind='mergesort')

            # Update k_indices and k_nearest_labels based on the
sorted weighted distances
            k_indices = [k_indices[i] for i in
sorted_indices_by_weight]
            k_nearest_labels = [self.y_train[i] for i in k_indices]
            final_prediction = k_nearest_labels[0]

        return final_prediction

knnalgorithm = KNN()
knnalgorithm.fit(x_train, y_train)
sum_Accuracy=0
cnt=0;

for k in range(1, 20):
    knnalgorithm.k = k;
    y_predicts = knnalgorithm.predict(x_test)
    correct_prediction = np.sum(y_predicts == y_test)
    Accuracy = correct_prediction / len(y_test)
    sum_Accuracy=sum_Accuracy+Accuracy
    cnt=cnt+1

    print("k value: ",knnalgorithm.k)
    print("Number of correctly classified instances:
",correct_prediction)
    print("Total number of instances: ",len(y_test))
    print("Accuracy: ",Accuracy*100)


print("average accuracy: ", sum_Accuracy/cnt)
print("average accuracy %100: ", (sum_Accuracy/cnt)*100)
```

```
k value:  1
Number of correctly classified instances:  153
Total number of instances:  231
Accuracy:  66.23376623376623
k value:  2
Number of correctly classified instances:  153
Total number of instances:  231
Accuracy:  66.23376623376623
k value:  3
Number of correctly classified instances:  160
Total number of instances:  231
Accuracy:  69.26406926406926
k value:  4
Number of correctly classified instances:  163
```

```
Total number of instances:  231
Accuracy:  70.56277056277057
k value:  5
Number of correctly classified instances:  163
Total number of instances:  231
Accuracy:  70.56277056277057
k value:  6
Number of correctly classified instances:  168
Total number of instances:  231
Accuracy:  72.72727272727273
k value:  7
Number of correctly classified instances:  168
Total number of instances:  231
Accuracy:  72.72727272727273
k value:  8
Number of correctly classified instances:  165
Total number of instances:  231
Accuracy:  71.42857142857143
k value:  9
Number of correctly classified instances:  166
Total number of instances:  231
Accuracy:  71.86147186147186
k value:  10
Number of correctly classified instances:  166
Total number of instances:  231
Accuracy:  71.86147186147186
k value:  11
Number of correctly classified instances:  174
Total number of instances:  231
Accuracy:  75.32467532467533
k value:  12
Number of correctly classified instances:  172
Total number of instances:  231
Accuracy:  74.45887445887446
k value:  13
Number of correctly classified instances:  171
Total number of instances:  231
Accuracy:  74.02597402597402
k value:  14
Number of correctly classified instances:  170
Total number of instances:  231
Accuracy:  73.59307359307358
k value:  15
Number of correctly classified instances:  167
Total number of instances:  231
Accuracy:  72.2943722943723
k value:  16
Number of correctly classified instances:  165
Total number of instances:  231
```

```
Accuracy:  71.42857142857143
k value:  17
Number of correctly classified instances:  162
Total number of instances:  231
Accuracy:  70.12987012987013
k value:  18
Number of correctly classified instances:  172
Total number of instances:  231
Accuracy:  74.45887445887446
k value:  19
Number of correctly classified instances:  167
Total number of instances:  231
Accuracy:  72.2943722943723
average accuracy:  0.7165641376167692
average accuracy %100:  71.65641376167692
```