Diabetes Prediction

Group Members:

Oliver Grudzinski

Pranaav Paladugu

Alec Weinbender

Research Question

With this project, we are attempting to see whether there any variables that we can use to diagnostically predict whether or not a patient has diabetes, based on the data in the dataset.

Hypotheses

Hypotheses Statements:

Null Hypothesis Example - A given variable does not influence whether a patient may have diabetes.

Alternative Hypothesis Example - A given variable does influence whether a patient may have diabetes.

These are all the variables that we will be looking at: Pregnancies, Glucose, Blood Pressure, Skin Thickness, Insulin, BMI, Diabetes Pedigree Function, Age.

Examples:

H0 (Glucose) – Glucose levels do not influence whether a patient may have diabetes.

HA (Glucose) – Glucose levels do have an influence on whether a patient may have diabetes.

H0 (BMI) – The patient’s BMI does not influence whether a patient may have diabetes.

HA (BMI) – The patient’s BMI does influence on whether a patient may have diabetes.

H0 (Age) – The patient’s age does not influence whether a patient may have diabetes.

HA (Age) – The patient’s age does influence on whether a patient may have diabetes.

H0 (Diabetes Pedigree Function) – The patient’s DPF levels do not influence whether a patient may have diabetes.

HA (Diabetes Pedigree Function) – The patient’s DPF levels do have an influence on whether a patient may have diabetes.

Dataset

Explanation of the Dataset:

This dataset was created to do what we are doing, which is to diagnostically predict whether or not a patient has diabetes. All of the data in this dataset is taken from female patients that are at least 21 years old and of Pima Indian heritage.

Source: https://www.kaggle.com/datasets/uciml/pima-indians-diabetes-database?resource=download

Description of Features:

Pregnancies: Number of times pregnant

Glucose: Plasma glucose concentration at 2 hours in an oral glucose tolerance test

Blood Pressure: Diastolic blood pressure (mm Hg)

Skin Thickness: Triceps skin fold thickness (mm)

Insulin: 2-Hour serum insulin (mu U/ml)

BMI: Body mass index (weight in kg/(height in m)^2)

Diabetes Pedigree Function: Diabetes pedigree function

Age: patient's (years)

Target Variable:

The target variable is Outcomes:

0 = Negative for Diabetes (500)

1 = Positive for Diabetes (268)

In [1]:
```
import pandas as pd

df = pd.read_csv("data/diabetes.csv")
df.head()
```

Out[1]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

Data Preprocessing

Identifying Features

In [2]:
```
df.describe()
```

Out[2]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	C
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.471876	33.240885	(
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.331329	11.760232	(
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.000000	(
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.243750	24.000000	(
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.372500	29.000000	(
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.000000	1
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	1

We are going to move forward with using all of the features given in the dataframe as they are all distinct and have the ability to be used for testing and modeling.

In [3]:
```
features = df.columns.drop('Outcome').tolist()
features
```

Out[3]:
```
['Pregnancies',
 'Glucose',
 'BloodPressure',
 'SkinThickness',
 'Insulin',
 'BMI',
 'DiabetesPedigreeFunction',
 'Age']
```

Handling Missing and/or Null Values

In [4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Pregnancies               768 non-null    int64  
 1   Glucose                   768 non-null    int64  
 2   BloodPressure             768 non-null    int64  
 3   SkinThickness             768 non-null    int64  
 4   Insulin                   768 non-null    int64  
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64  
 8   Outcome                   768 non-null    int64  
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

No null values in the dataframe!

Searching for Miscalculated Zeros

In [5]:
```python
for column in features:
    num_zeros = (df[column] == 0).sum()
    print(f"Number of 0s in '{column}': {num_zeros}")
```

```
Number of 0s in 'Pregnancies': 111
Number of 0s in 'Glucose': 5
Number of 0s in 'BloodPressure': 35
Number of 0s in 'SkinThickness': 227
Number of 0s in 'Insulin': 374
Number of 0s in 'BMI': 11
Number of 0s in 'DiabetesPedigreeFunction': 0
Number of 0s in 'Age': 0
```

Looking at the columns with 0's, we should replace or drop the values that are continous variables that don't make sense having zero's.

For this, it makes sense to replace the columns of Glucose, Blood Pressure, Skin Thickness, Insulin, and BMI as they are all variables that should be above zero.

We are going to replace the values by entering the mean values of the columns where there the values are 0.

In [6]:
```python
columns_to_check = ['Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI']

for column in columns_to_check:
    mean_value = df.loc[df[column] != 0, column].mean() # skip 0 values
    df[column] = df[column].replace(0, mean_value)
    num_zeros = (df[column] == 0).sum()
    print(f"Number of 0s in '{column}': {num_zeros}")
```

```
Number of 0s in 'Glucose': 0
Number of 0s in 'BloodPressure': 0
Number of 0s in 'SkinThickness': 0
Number of 0s in 'Insulin': 0
Number of 0s in 'BMI': 0
```

In [7]:
```python
df.describe()
```

Out[7]:

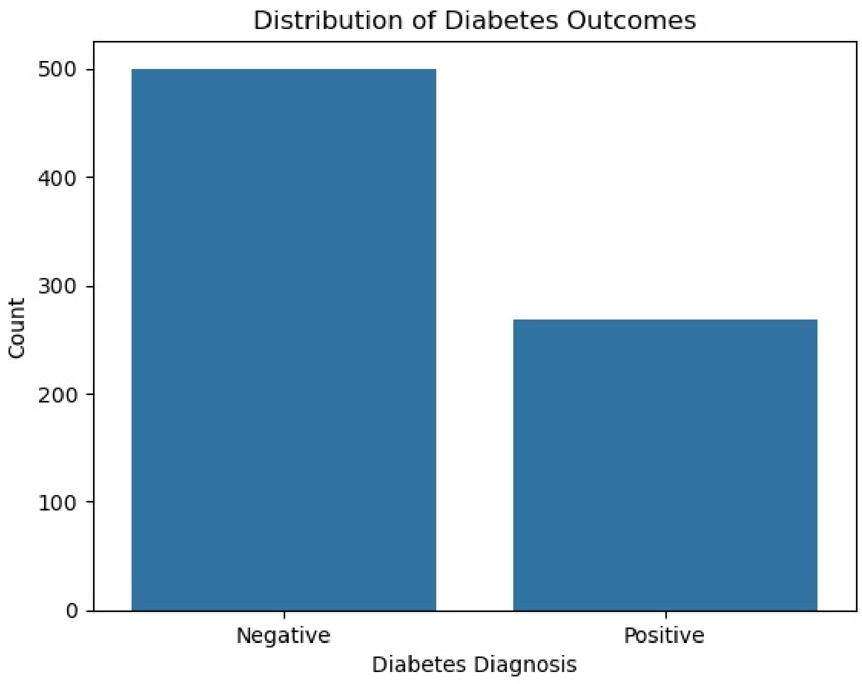
	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	121.686763	72.405184	29.153420	155.548223	32.457464	0.471876	33.240885
std	3.369578	30.435949	12.096346	8.790942	85.021108	6.875151	0.331329	11.760232
min	0.000000	44.000000	24.000000	7.000000	14.000000	18.200000	0.078000	21.000000
25%	1.000000	99.750000	64.000000	25.000000	121.500000	27.500000	0.243750	24.000000
50%	3.000000	117.000000	72.202592	29.153420	155.548223	32.400000	0.372500	29.000000
75%	6.000000	140.250000	80.000000	32.000000	155.548223	36.600000	0.626250	41.000000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000

Data Analysis and Visualization

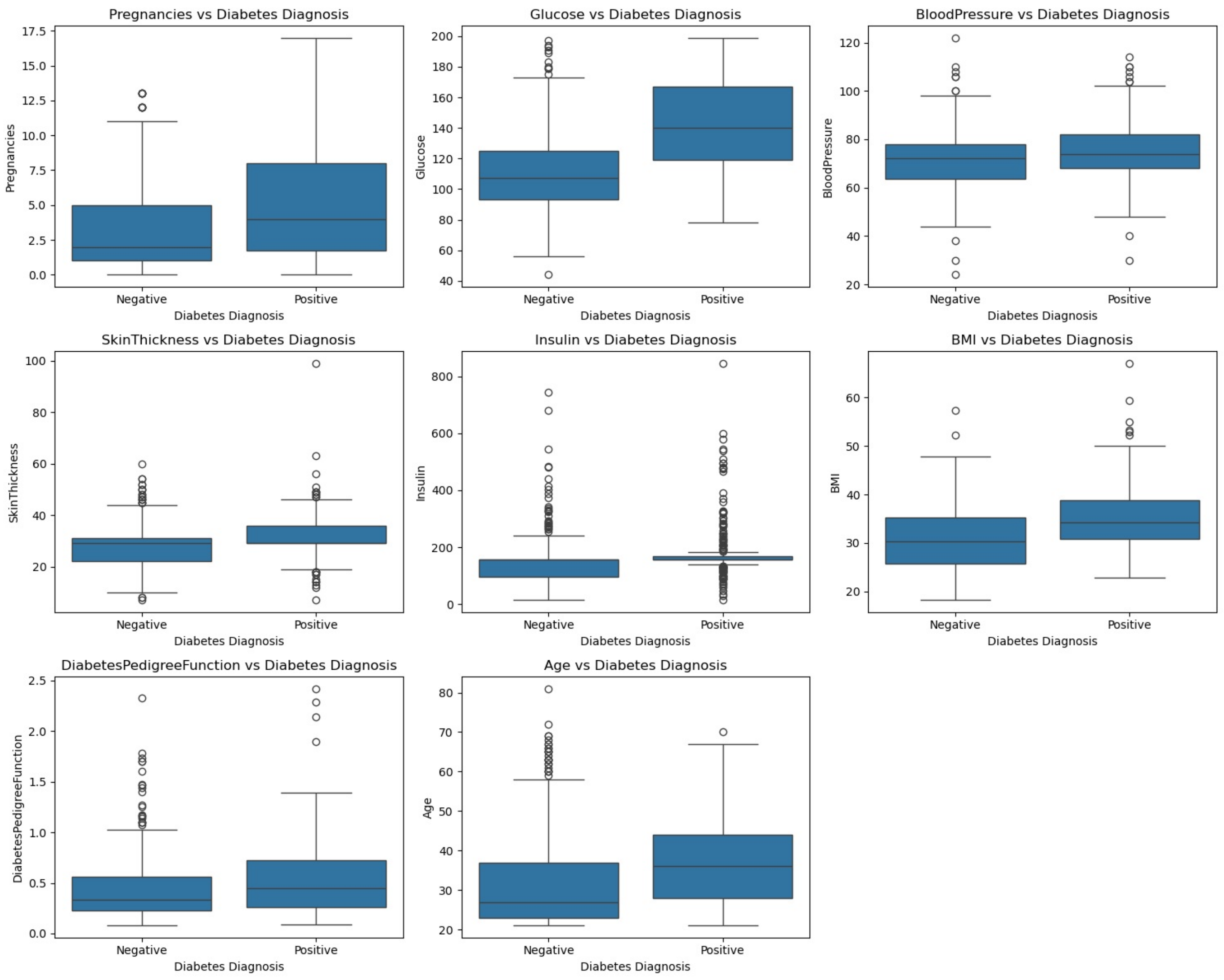
In [8]:
```python
import matplotlib.pyplot as plt
import seaborn as sns
```

Viewing the Data after Preprocessing

In [9]:
```python
outcomes = df['Outcome']
sns.countplot(x=outcomes)
plt.xticks([0, 1], ["Negative", "Positive"])
plt.xlabel("Diabetes Diagnosis")
plt.ylabel("Count")
plt.title("Distribution of Diabetes Outcomes")
plt.show()
```



In [10]:
```python
features = df.columns.drop('Outcome')
num_features = len(features)
cols = 3
rows = (num_features + cols - 1) // cols
plt.figure(figsize=(5 * cols, 4 * rows))
for i, feature in enumerate(features):
    plt.subplot(rows, cols, i + 1)
    sns.boxplot(x=outcomes, y=df[feature])
    plt.xlabel("Diabetes Diagnosis")
    plt.ylabel(feature)
    plt.xticks([0, 1], ["Negative", "Positive"])
    plt.title(f"{feature} vs Diabetes Diagnosis")

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



Hypothesis Testing

In [11]:

```
from scipy import stats
import pandas as pd

target = df["Outcome"]
features = df.columns.drop("Outcome")
n_samples = len(df)

alpha = 0.05
num_tests = len(features)
threshold = alpha/num_tests #Bonferroni Correction


Dict = {}
var = []
pval = []
Dict = {"Variable":[], "T-stat": [], "P-value": [], "Significant":[]}
count = 0
for variable in features:
        samples_by_group = []
        for value in set(df[variable]):
            mask = df[variable] == value
            samples_by_group.append(df['Outcome'][mask])
        kstat, Kp = stats.kruskal(*samples_by_group)
        significant = True
        if(Kp < ((threshold)/n_samples)):
            significant = True
        elif(Kp > ((threshold)/n_samples)):
            significant = False
        Dict["T-stat"].append(kstat)
        Dict["P-value"].append(Kp)
        pval.append(Kp)
        var.append(variable)
        Dict["Significant"].append(significant)
        Dict["Variable"].append(variable)

table = pd.DataFrame(Dict)

sorted_df = table.sort_values(by=['P-value'], ascending=True)
```

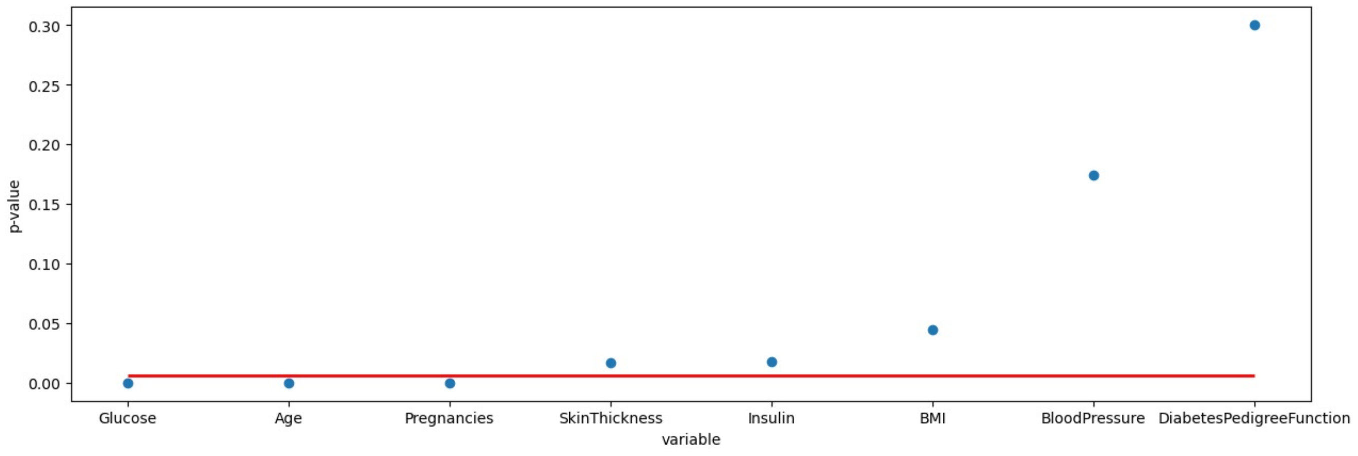
```
sorted_df.head(10)
```

Out[11]:

	Variable	T-stat	P-value	Significant
1	Glucose	269.382027	5.586210e-11	True
7	Age	140.754007	2.450833e-10	True
0	Pregnancies	64.510701	8.940856e-08	True
3	SkinThickness	73.467109	1.699207e-02	False
4	Insulin	227.473254	1.818808e-02	False
5	BMI	286.097245	4.422738e-02	False
2	BloodPressure	54.862435	1.738525e-01	False
6	DiabetesPedigreeFunction	532.330054	3.002203e-01	False

Visualizing Significant Variables

In [12]:
```
import matplotlib.pyplot as plt
import numpy as np

x = np.array(sorted_df['Variable'])
y = np.array(sorted_df['P-value'])

f = plt.figure()
f.set_figwidth(15)

plt.xlabel('variable')
plt.ylabel('p-value')

plt.scatter(x, y)
plt.hlines(y=threshold, xmin=0, xmax=7, linewidth=2, color='r')
plt.show()
```



In [13]:
```
significant_variables = sorted_df[sorted_df["Significant"] == True]["Variable"].tolist()
significant_variables
```

Out[13]:
```
['Glucose', 'Age', 'Pregnancies']
```

We will reject the null hypotheses for the variables Glucose, Age, and Pregnancies.

We have failed to reject the null hypotheses for the variables Skin Thickness, Insulin, BMI, Blood Pressure, and DiabetesPedigreeFunction.

Data Modeling and Prediction

Scaling/Normalizing

We will not need to scale or normalize our data as we are going to be building a Random Forest Classifier model.

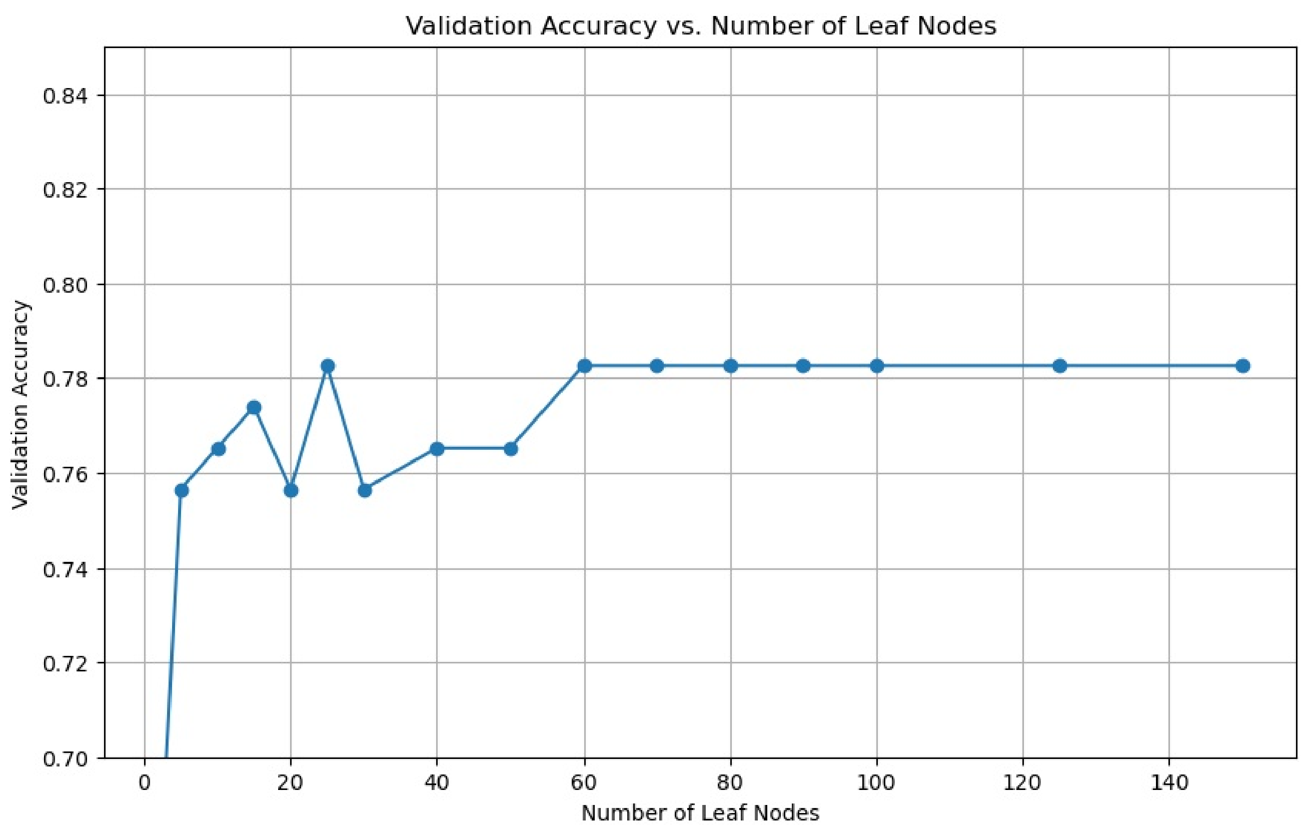
Baseline Model with Optimized Parameters

Finding Optimal max_leaf_nodes

In [14]:
```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
```

```python
import matplotlib.pyplot as plt

# Define features and target variable
X = df[features]
y = df["Outcome"]

X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.3, random_state=42, stratify=y)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42, stratify=y_temp

# Range of max_leaf_nodes to test
leaf_nodes_range = [2, 5, 10, 15, 20, 25, 30, 40, 50, 60, 70, 80, 90, 100, 125, 150]

# Dictionary to store validation accuracies
validation_accuracies = {}

for node_value in leaf_nodes_range:
    model = RandomForestClassifier(random_state=42, n_estimators=50, max_leaf_nodes=node_value)
    model.fit(X_train, y_train)
    y_val_pred = model.predict(X_val)
    accuracy = accuracy_score(y_val, y_val_pred)
    validation_accuracies[node_value] = accuracy
    print(f"max_leaf_nodes: {node_value}, Validation Accuracy: {accuracy:.4f}")


# Find the best max_leaf_nodes from the best validation accuracy
max_leaf_nodes = max(validation_accuracies, key=validation_accuracies.get)
print(f"\nOptimal number of max_leaf_nodes: {max_leaf_nodes}")

# Plot validation accuracies
plt.figure(figsize=(10, 6))
plt.plot(leaf_nodes_range, validation_accuracies.values(), marker='o', linestyle='-')
plt.title("Validation Accuracy vs. Number of Leaf Nodes")
plt.xlabel("Number of Leaf Nodes")
plt.ylabel("Validation Accuracy")
plt.ylim(0.7, 0.85)  # Set y-axis to start at 0
plt.grid()
plt.show()
```

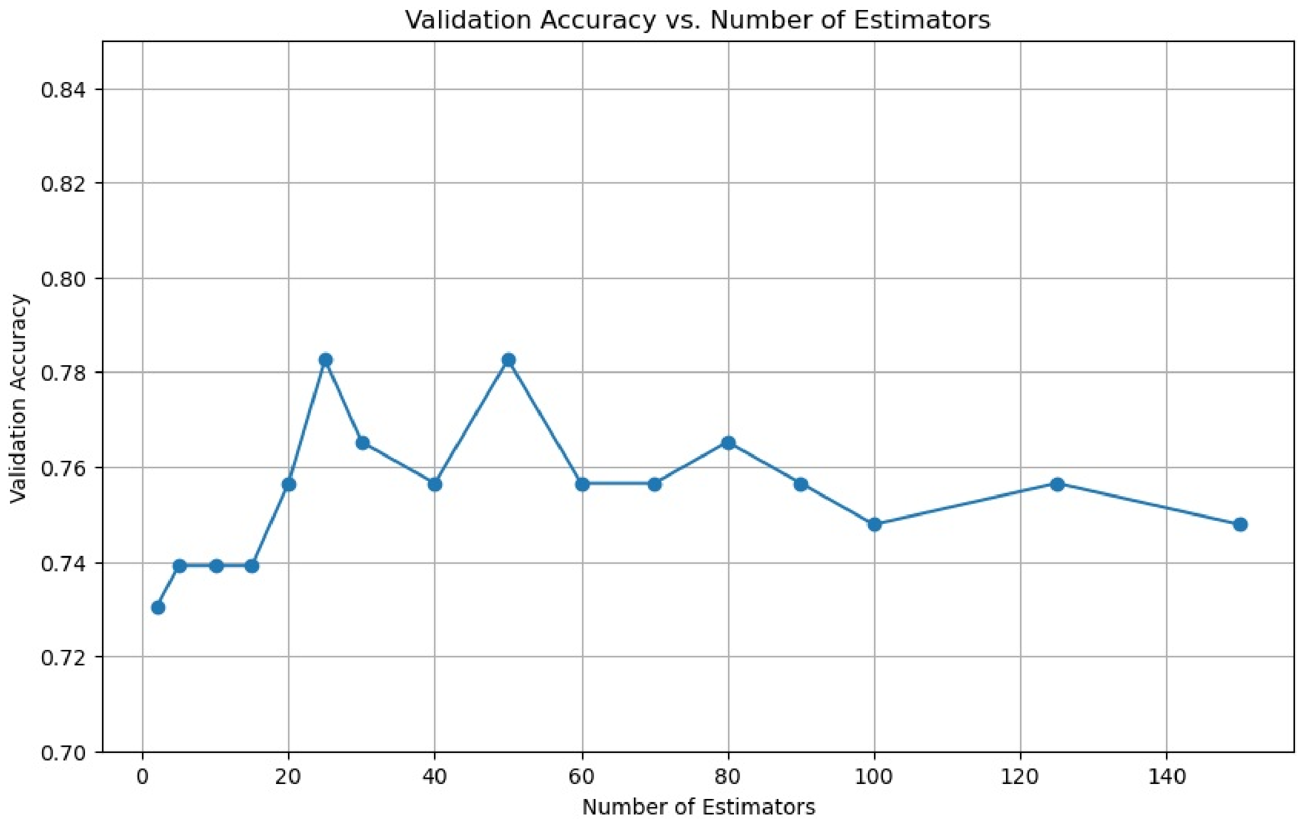
```
max_leaf_nodes: 2, Validation Accuracy: 0.6696
max_leaf_nodes: 5, Validation Accuracy: 0.7565
max_leaf_nodes: 10, Validation Accuracy: 0.7652
max_leaf_nodes: 15, Validation Accuracy: 0.7739
max_leaf_nodes: 20, Validation Accuracy: 0.7565
max_leaf_nodes: 25, Validation Accuracy: 0.7826
max_leaf_nodes: 30, Validation Accuracy: 0.7565
max_leaf_nodes: 40, Validation Accuracy: 0.7652
max_leaf_nodes: 50, Validation Accuracy: 0.7652
max_leaf_nodes: 60, Validation Accuracy: 0.7826
max_leaf_nodes: 70, Validation Accuracy: 0.7826
max_leaf_nodes: 80, Validation Accuracy: 0.7826
max_leaf_nodes: 90, Validation Accuracy: 0.7826
max_leaf_nodes: 100, Validation Accuracy: 0.7826
max_leaf_nodes: 125, Validation Accuracy: 0.7826
max_leaf_nodes: 150, Validation Accuracy: 0.7826

Optimal number of max_leaf_nodes: 25
```



Finding Optimal n_estimators

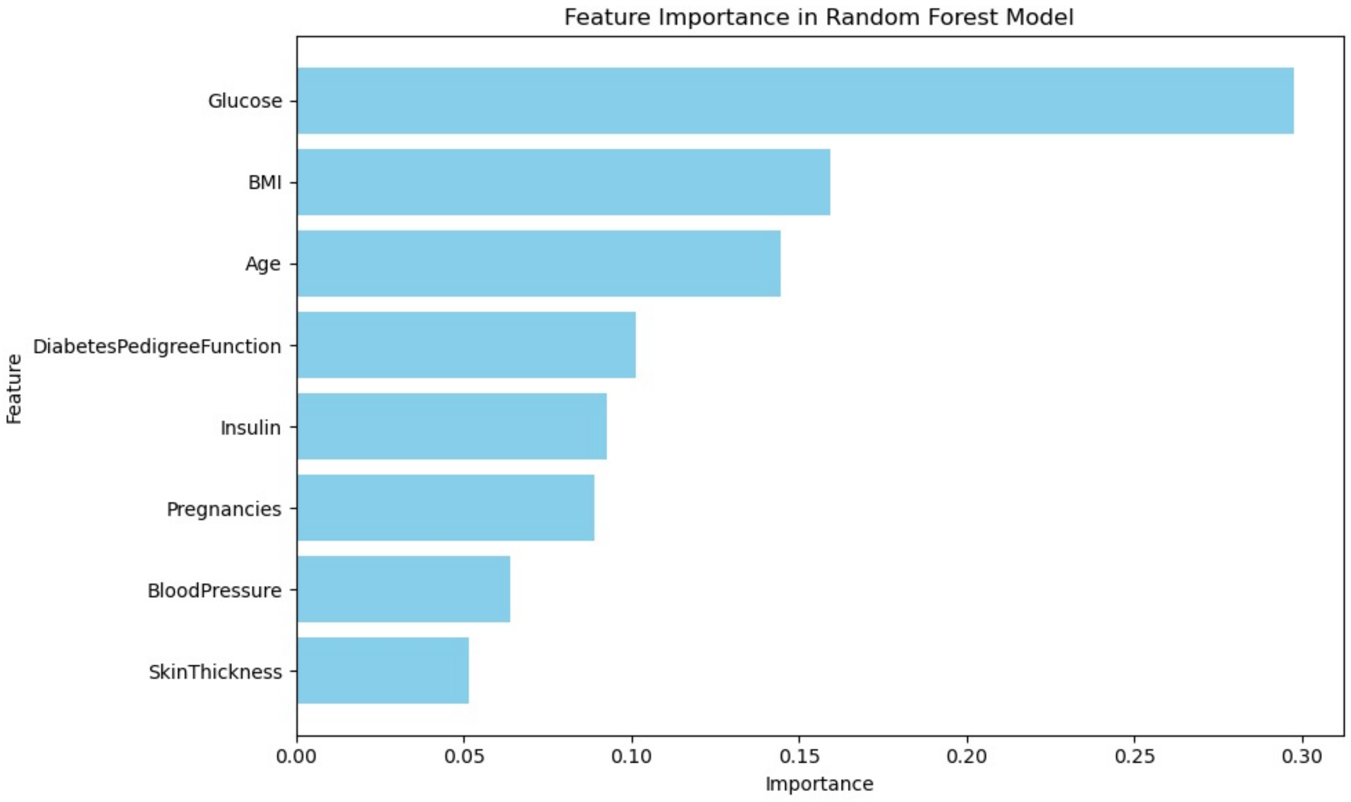
```
In [15]: # Define features and target variable
X = df[features]
y = df["Outcome"]

# Split the data into training, validation, and test sets
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.3, random_state=42, stratify=y)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42, stratify=y_temp

# Range of n_estimators to test
n_estimators_range = [2, 5, 10, 15, 20, 25, 30, 40, 50, 60, 70, 80, 90, 100, 125, 150]

# Dictionary to store validation accuracies
validation_accuracies = {}

# Find the optimal number of estimators
for n_estimators in n_estimators_range:
    model = RandomForestClassifier(random_state=42, n_estimators=n_estimators, max_leaf_nodes=max_leaf_nodes)
    model.fit(X_train, y_train)
    y_val_pred = model.predict(X_val)
    accuracy = accuracy_score(y_val, y_val_pred)
    validation_accuracies[n_estimators] = accuracy
    print(f"n_estimators: {n_estimators}, Validation Accuracy: {accuracy:.4f}")

# Find the n_estimators with the best validation accuracy
optimal_n_estimators = max(validation_accuracies, key=validation_accuracies.get)
print(f"\nOptimal number of estimators: {optimal_n_estimators}")

# Plot validation accuracies
plt.figure(figsize=(10, 6))
plt.plot(n_estimators_range, validation_accuracies.values(), marker='o', linestyle='-')
plt.title("Validation Accuracy vs. Number of Estimators")
plt.xlabel("Number of Estimators")
plt.ylabel("Validation Accuracy")
plt.ylim(0.7,0.85)  # Set y-axis to start at 0
plt.grid()
plt.show()
```

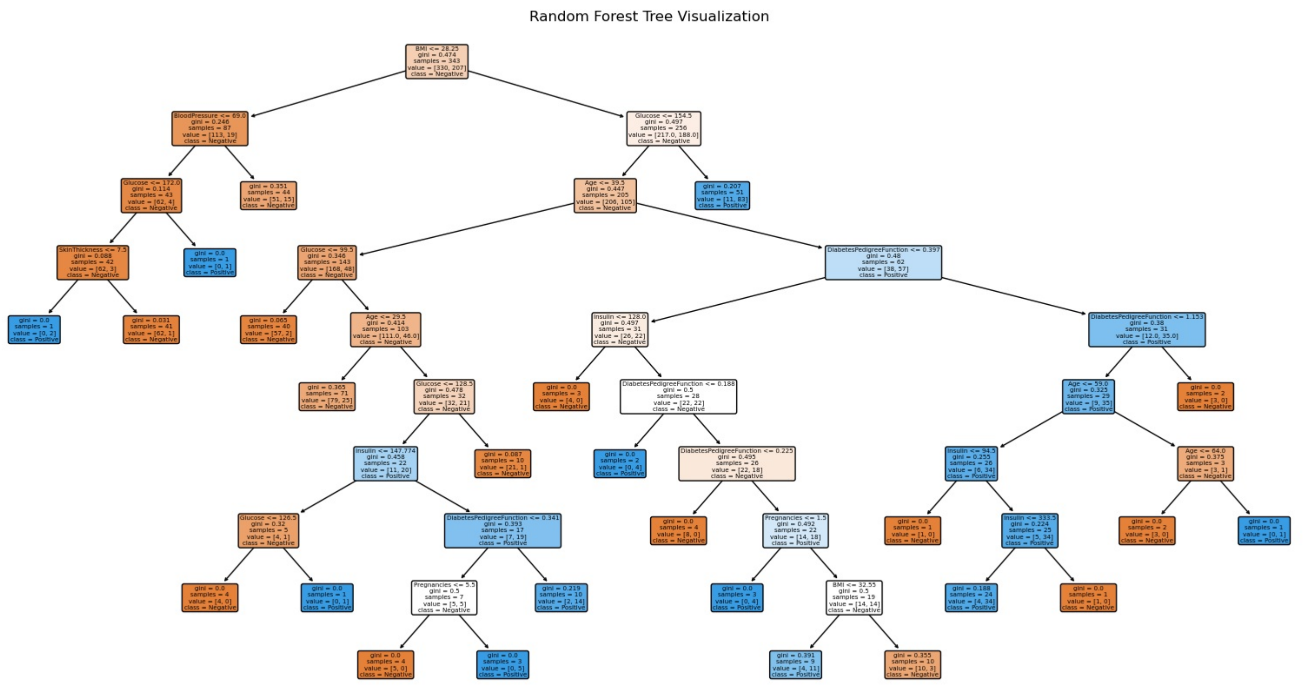
```
n_estimators: 2, Validation Accuracy: 0.7304
n_estimators: 5, Validation Accuracy: 0.7391
n_estimators: 10, Validation Accuracy: 0.7391
n_estimators: 15, Validation Accuracy: 0.7391
n_estimators: 20, Validation Accuracy: 0.7565
n_estimators: 25, Validation Accuracy: 0.7826
n_estimators: 30, Validation Accuracy: 0.7652
n_estimators: 40, Validation Accuracy: 0.7565
n_estimators: 50, Validation Accuracy: 0.7826
n_estimators: 60, Validation Accuracy: 0.7565
n_estimators: 70, Validation Accuracy: 0.7565
n_estimators: 80, Validation Accuracy: 0.7652
n_estimators: 90, Validation Accuracy: 0.7565
n_estimators: 100, Validation Accuracy: 0.7478
n_estimators: 125, Validation Accuracy: 0.7565
n_estimators: 150, Validation Accuracy: 0.7478

Optimal number of estimators: 25
```



In [16]:
```
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.tree import plot_tree

def visualize_forest(forest, feature_names):
    importance_df = pd.DataFrame({
        "Feature": feature_names,
        "Importance": forest.feature_importances_
    }).sort_values("Importance", ascending=False)

    plt.figure(figsize=(10, 6))
    plt.barh(importance_df["Feature"], importance_df["Importance"], color="skyblue")
    plt.gca().invert_yaxis()  # Invert y-axis to show the most important feature at the top
    plt.title("Feature Importance in Random Forest Model")
    plt.xlabel("Importance")
    plt.ylabel("Feature")
    plt.tight_layout()
    plt.show()

    plt.figure(figsize=(20, 10))
    plot_tree(forest.estimators_[0], feature_names=feature_names, class_names=["Negative", "Positive"], filled=
    plt.title("Random Forest Tree Visualization")
    plt.show()

X = df[features]
y = df["Outcome"]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42, stratify=y)

# Train the final model with the optimal number of estimators
optimized_model = RandomForestClassifier(random_state=42, n_estimators=optimal_n_estimators, max_leaf_nodes=max_
optimized_model.fit(X_train, y_train)

# Evaluate the model on the test set
optimized_y_test_pred = optimized_model.predict(X_test)
test_accuracy = accuracy_score(y_test, optimized_y_test_pred)
print(f"\nTest Accuracy with {max_leaf_nodes} max_leaf_nodes and {optimal_n_estimators} n_estimators: {test_accu

# Feature importance visualization
visualize_forest(optimized_model, features)
```

```
Test Accuracy with 25 max_leaf_nodes and 25 n_estimators: 0.7706
```



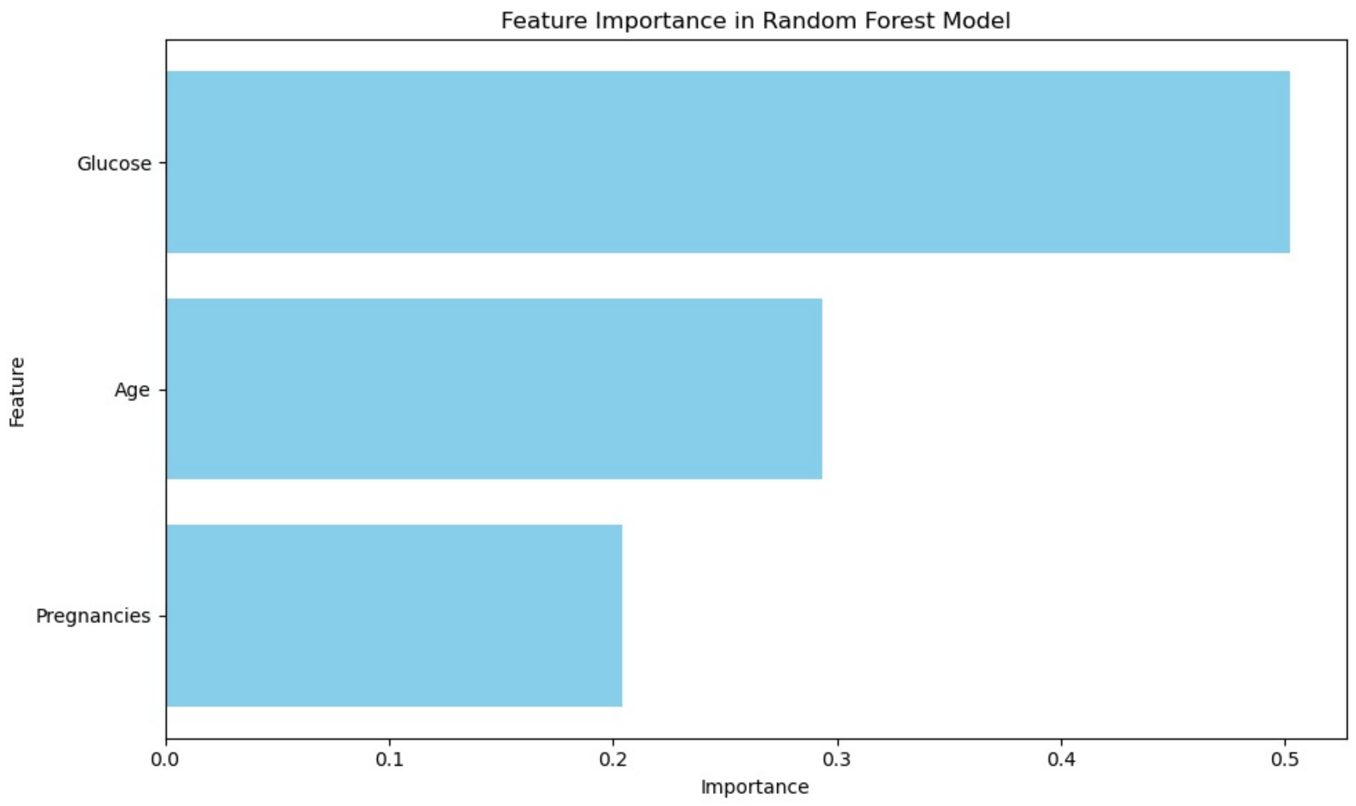


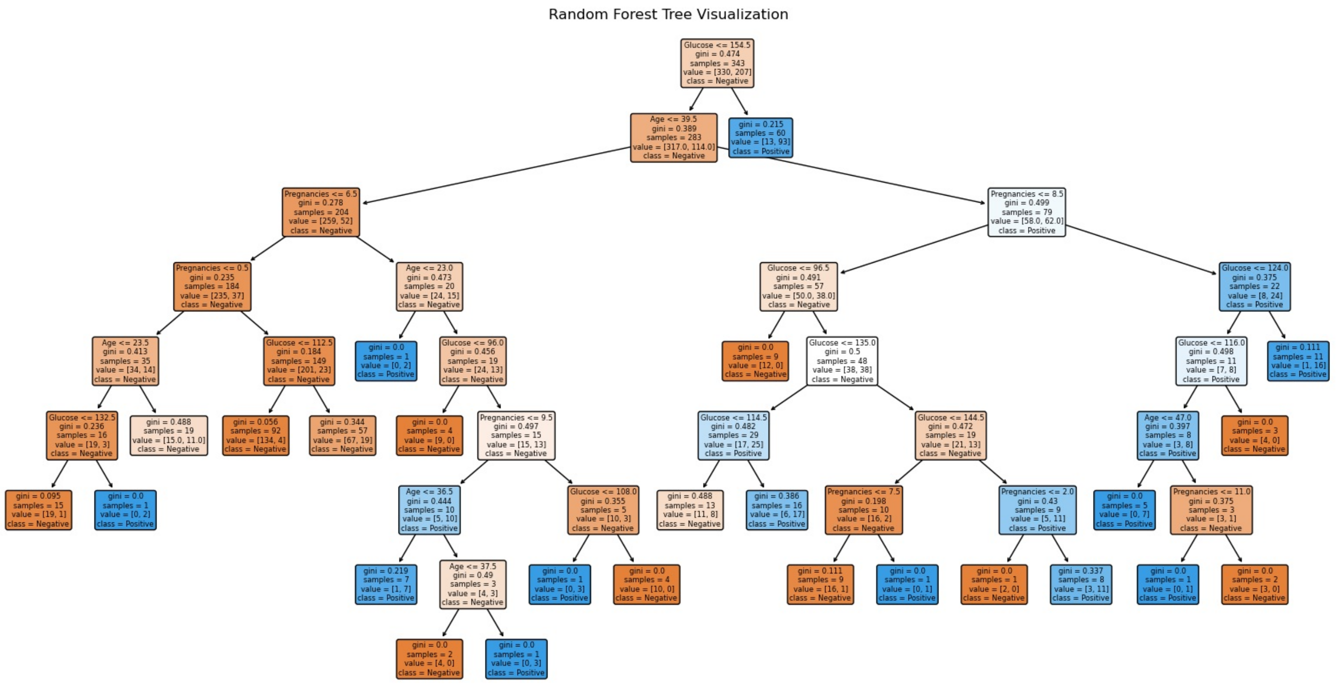
Random Forest Tree Visualization
BMI <= 28.25 gini = 0.474 samples = 343 value = [330, 207] class = Negative
BloodPressure <= 69.0 gini = 0.246 samples = 87 value = [113, 19] class = Negative
Glucose <= 154.5 gini = 0.497 samples = 256 value = [217.0, 188.0] class = Negative
Glucose <= 172.0 gini = 0.114 samples = 43 value = [62, 4] class = Negative
gini = 0.351 samples = 44 value = [51, 15] class = Negative
Age <= 39.5 gini = 0.447 samples = 205 value = [206, 105] class = Negative
gini = 0.207 samples = 51 value = [11, 83] class = Positive
SkinThickness <= 7.5 gini = 0.088 samples = 42 value = [62, 3] class = Negative
gini = 0.0 samples = 1 value = [0, 1] class = Positive
Glucose <= 99.5 gini = 0.346 samples = 143 value = [168, 48] class = Negative
DiabetesPedigreeFunction <= 0.397 gini = 0.48 samples = 62 value = [38, 57] class = Positive
gini = 0.0 samples = 1 value = [0, 2] class = Positive
gini = 0.031 samples = 41 value = [62, 1] class = Negative
gini = 0.065 samples = 40 value = [57, 2] class = Negative
Age <= 29.5 gini = 0.414 samples = 103 value = [111.0, 46.0] class = Negative
Insulin <= 128.0 gini = 0.497 samples = 31 value = [26, 22] class = Negative
DiabetesPedigreeFunction <= 1.153 gini = 0.38 samples = 31 value = [12.0, 35.0] class = Positive
gini = 0.365 samples = 71 value = [79, 25] class = Negative
Glucose <= 128.5 gini = 0.478 samples = 32 value = [32, 21] class = Negative
gini = 0.0 samples = 3 value = [4, 0] class = Negative
DiabetesPedigreeFunction <= 0.188 gini = 0.5 samples = 28 value = [22, 22] class = Negative
Age <= 59.0 gini = 0.325 samples = 29 value = [9, 35] class = Positive
gini = 0.0 samples = 2 value = [3, 0] class = Negative
Insulin <= 147.774 gini = 0.458 samples = 22 value = [11, 20] class = Positive
gini = 0.087 samples = 10 value = [21, 1] class = Negative
gini = 0.0 samples = 2 value = [0, 4] class = Positive
DiabetesPedigreeFunction <= 0.225 gini = 0.495 samples = 26 value = [22, 18] class = Negative
Insulin <= 94.5 gini = 0.255 samples = 26 value = [6, 34] class = Positive
Age <= 64.0 gini = 0.375 samples = 3 value = [3, 1] class = Negative
Glucose <= 126.5 gini = 0.32 samples = 5 value = [4, 1] class = Negative
DiabetesPedigreeFunction <= 0.341 gini = 0.393 samples = 17 value = [7, 19] class = Positive
gini = 0.0 samples = 4 value = [8, 0] class = Negative
Pregnancies <= 1.5 gini = 0.492 samples = 22 value = [14, 18] class = Positive
gini = 0.0 samples = 1 value = [1, 0] class = Negative
Insulin <= 333.5 gini = 0.224 samples = 25 value = [5, 34] class = Positive
gini = 0.0 samples = 2 value = [3, 0] class = Negative
gini = 0.0 samples = 1 value = [0, 1] class = Positive
gini = 0.0 samples = 4 value = [4, 0] class = Negative
gini = 0.0 samples = 1 value = [0, 1] class = Positive
Pregnancies <= 5.5 gini = 0.5 samples = 7 value = [5, 5] class = Negative
gini = 0.219 samples = 10 value = [2, 14] class = Positive
gini = 0.0 samples = 3 value = [0, 4] class = Positive
BMI <= 32.55 gini = 0.5 samples = 19 value = [14, 14] class = Negative
gini = 0.188 samples = 24 value = [4, 34] class = Positive
gini = 0.0 samples = 1 value = [1, 0] class = Negative
gini = 0.0 samples = 4 value = [5, 0] class = Negative
gini = 0.0 samples = 3 value = [0, 5] class = Positive
gini = 0.391 samples = 9 value = [4, 11] class = Positive
gini = 0.355 samples = 10 value = [10, 3] class = Negative

Significant Feature Selection Model

```
In [21]: X = df[significant_variables]
         y = df["Outcome"]
         X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42, stratify=y)
         significant_model = RandomForestClassifier(random_state=42, n_estimators=optimal_n_estimators, max_leaf_nodes=ma
         significant_model.fit(X_train, y_train)

         significant_y_test_pred = significant_model.predict(X_test)
         test_accuracy = accuracy_score(y_test, significant_y_test_pred)
         print(f"\nSignificant Model Test Accuracy with {max_leaf_nodes} max_leaf_nodes and {optimal_n_estimators} n_esti

         # Feature importance visualization
         visualize_forest(significant_model, significant_variables)
```

```
Significant Model Test Accuracy with 25 max_leaf_nodes and 25 n_estimators: 0.7186
```





Random Forest Tree Visualization
Glucose <= 154.5 gini = 0.474 samples = 343 value = [330, 207] class = Negative
Age <= 39.5 gini = 0.389 samples = 283 value = [317.0, 114.0] class = Negative
gini = 0.215 samples = 60 value = [13, 93] class = Positive
Pregnancies <= 6.5 gini = 0.278 samples = 204 value = [259, 52] class = Negative
Pregnancies <= 8.5 gini = 0.499 samples = 79 value = [58.0, 62.0] class = Positive
Pregnancies <= 0.5 gini = 0.235 samples = 184 value = [235, 37] class = Negative
Age <= 23.0 gini = 0.473 samples = 20 value = [24, 15] class = Negative
Glucose <= 96.5 gini = 0.491 samples = 57 value = [50.0, 38.0] class = Negative
Glucose <= 124.0 gini = 0.375 samples = 22 value = [8, 24] class = Positive
Age <= 23.5 gini = 0.413 samples = 35 value = [34, 14] class = Negative
Glucose <= 112.5 gini = 0.184 samples = 149 value = [201, 23] class = Negative
gini = 0.0 samples = 1 value = [0, 2] class = Positive
Glucose <= 96.0 gini = 0.456 samples = 19 value = [24, 13] class = Negative
gini = 0.0 samples = 9 value = [12, 0] class = Negative
Glucose <= 135.0 gini = 0.5 samples = 48 value = [38, 38] class = Negative
Glucose <= 116.0 gini = 0.498 samples = 11 value = [7, 8] class = Positive
gini = 0.111 samples = 11 value = [1, 16] class = Positive
Glucose <= 132.5 gini = 0.236 samples = 16 value = [19, 3] class = Negative
gini = 0.488 samples = 19 value = [15.0, 11.0] class = Negative
gini = 0.056 samples = 92 value = [134, 4] class = Negative
gini = 0.344 samples = 57 value = [67, 19] class = Negative
gini = 0.0 samples = 4 value = [9, 0] class = Negative
Pregnancies <= 9.5 gini = 0.497 samples = 15 value = [15, 13] class = Negative
Glucose <= 114.5 gini = 0.482 samples = 29 value = [17, 25] class = Positive
Glucose <= 144.5 gini = 0.472 samples = 19 value = [21, 13] class = Negative
Age <= 47.0 gini = 0.397 samples = 8 value = [3, 8] class = Positive
gini = 0.0 samples = 3 value = [4, 0] class = Negative
gini = 0.095 samples = 15 value = [19, 1] class = Negative
gini = 0.0 samples = 1 value = [0, 2] class = Positive
Age <= 36.5 gini = 0.444 samples = 10 value = [5, 10] class = Positive
Glucose <= 108.0 gini = 0.355 samples = 5 value = [10, 3] class = Negative
gini = 0.488 samples = 13 value = [11, 8] class = Negative
gini = 0.386 samples = 16 value = [6, 17] class = Positive
Pregnancies <= 7.5 gini = 0.198 samples = 10 value = [16, 2] class = Negative
Pregnancies <= 2.0 gini = 0.43 samples = 9 value = [5, 11] class = Positive
gini = 0.0 samples = 5 value = [0, 7] class = Positive
Pregnancies <= 11.0 gini = 0.375 samples = 3 value = [3, 1] class = Negative
gini = 0.219 samples = 7 value = [1, 7] class = Positive
Age <= 37.5 gini = 0.49 samples = 3 value = [4, 3] class = Negative
gini = 0.0 samples = 1 value = [0, 3] class = Positive
gini = 0.0 samples = 4 value = [10, 0] class = Negative
gini = 0.111 samples = 9 value = [16, 1] class = Negative
gini = 0.0 samples = 1 value = [0, 1] class = Positive
gini = 0.0 samples = 1 value = [2, 0] class = Negative
gini = 0.337 samples = 8 value = [3, 11] class = Positive
gini = 0.0 samples = 1 value = [0, 1] class = Positive
gini = 0.0 samples = 2 value = [3, 0] class = Negative
gini = 0.0 samples = 2 value = [4, 0] class = Negative
gini = 0.0 samples = 1 value = [0, 3] class = Positive

Greedy Model

```
In [22]: # Rank features by importance
         importances = optimized_model.feature_importances_
         importances_sorted_indices = importances.argsort()[::-1]
         features_sorted = [features[i] for i in importances_sorted_indices]
         features_sorted
```

```
Out[22]: ['Glucose',
  'BMI',
  'Age',
  'DiabetesPedigreeFunction',
  'Insulin',
  'Pregnancies',
  'BloodPressure',
  'SkinThickness']
```

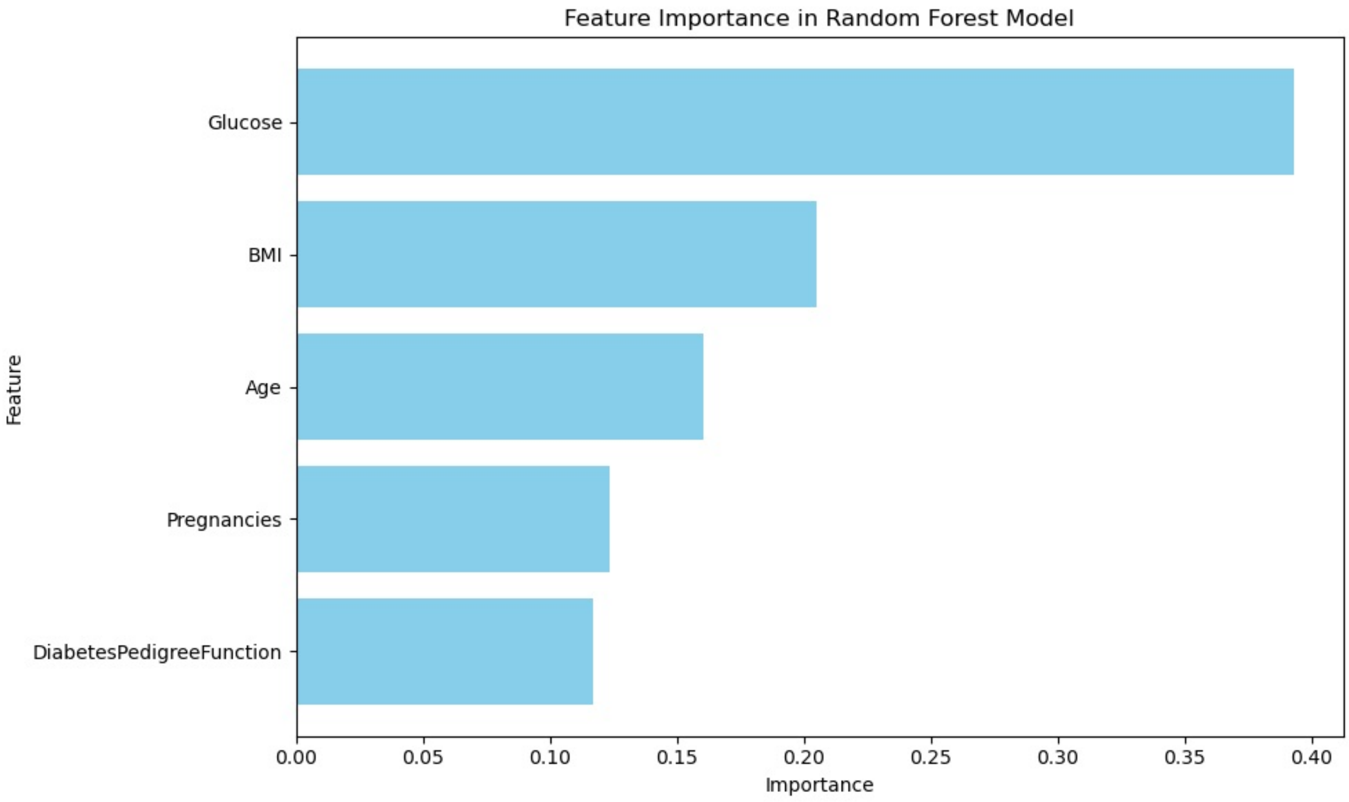
```
In [23]: # Add new features to the model one by one (greedy approach)

greedy_features = []
accuracy = 0
for feature in features_sorted:
    X = df[greedy_features + [feature]]
    y = df["Outcome"]
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42, stratify=y)
    model = RandomForestClassifier(random_state=42, n_estimators=optimal_n_estimators, max_leaf_nodes=max_leaf_
    model.fit(X_train, y_train)

    y_test_pred = model.predict(X_test)
    test_accuracy = accuracy_score(y_test, y_test_pred)
    if test_accuracy > accuracy:
        accuracy = test_accuracy
        greedy_features.append(feature)
        print(f"Added feature: {feature}, Test Accuracy: {test_accuracy:.4f}")
    else:
        print(f"Not added feature: {feature}. Test Accuracy: {test_accuracy:.4f}")

greedy_features
```

```
Added feature: Glucose, Test Accuracy: 0.7013
Added feature: BMI, Test Accuracy: 0.7100
Added feature: Age, Test Accuracy: 0.7446
Added feature: DiabetesPedigreeFunction, Test Accuracy: 0.7706
Not added feature: Insulin. Test Accuracy: 0.7532
Added feature: Pregnancies, Test Accuracy: 0.7749
Not added feature: BloodPressure. Test Accuracy: 0.7619
Not added feature: SkinThickness. Test Accuracy: 0.7532
```

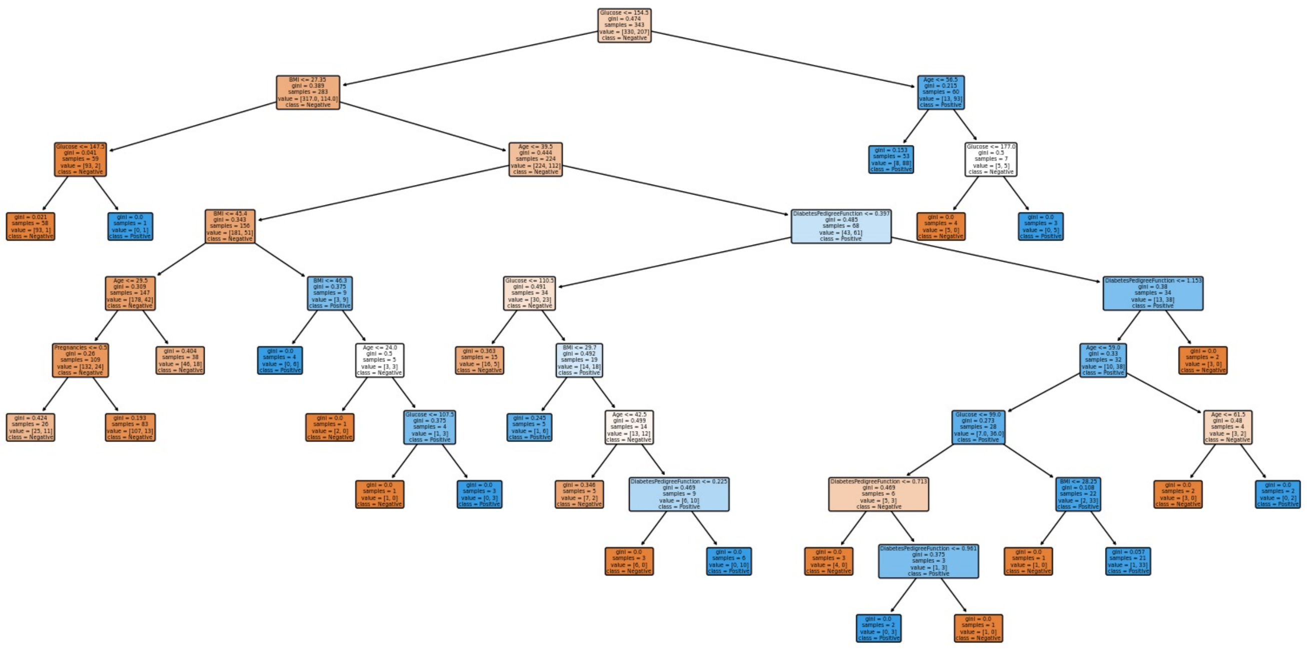
```
Out[23]: ['Glucose', 'BMI', 'Age', 'DiabetesPedigreeFunction', 'Pregnancies']
```

```
In [24]: X = df[greedy_features]
y = df["Outcome"]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42, stratify=y)

greedy_model = RandomForestClassifier(random_state=42, n_estimators=optimal_n_estimators, max_leaf_nodes=max_lea
greedy_model.fit(X_train, y_train)

greedy_y_test_pred = greedy_model.predict(X_test)
test_accuracy = accuracy_score(y_test, y_test_pred)
print(f"\nGreedy Model Test Accuracy with {max_leaf_nodes} max_leaf_nodes and {optimal_n_estimators} n_estimato

# Feature importance visualization
visualize_forest(greedy_model, greedy_features)
```

```
Greedy Model Test Accuracy with 25 max_leaf_nodes and 25 n_estimators: 0.7532
```


Feature Importance in Random Forest Model
Glucose
BMI
Age
Pregnancies
DiabetesPedigreeFunction
Feature
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
0.40
Importance

Random Forest Tree Visualization



Glucose <= 154.5 gini = 0.474 samples = 343 value = [330, 207] class = Negative
BMI <= 27.35 gini = 0.389 samples = 283 value = [317.0, 114.0] class = Negative
Age <= 56.5 gini = 0.215 samples = 60 value = [13, 93] class = Positive
Glucose <= 147.5 gini = 0.041 samples = 59 value = [93, 2] class = Negative
Age <= 39.5 gini = 0.444 samples = 224 value = [224, 112] class = Negative
gini = 0.153 samples = 53 value = [8, 88] class = Positive
Glucose <= 177.0 gini = 0.5 samples = 7 value = [5, 5] class = Negative
gini = 0.021 samples = 58 value = [93, 1] class = Negative
gini = 0.0 samples = 1 value = [0, 1] class = Positive
BMI <= 45.4 gini = 0.343 samples = 156 value = [181, 51] class = Negative
DiabetesPedigreeFunction <= 0.397 gini = 0.485 samples = 68 value = [43, 61] class = Positive
gini = 0.0 samples = 4 value = [5, 0] class = Negative
gini = 0.0 samples = 3 value = [0, 5] class = Positive
Age <= 29.5 gini = 0.309 samples = 147 value = [178, 42] class = Negative
BMI <= 46.3 gini = 0.375 samples = 9 value = [3, 9] class = Positive
Glucose <= 110.5 gini = 0.491 samples = 34 value = [30, 23] class = Negative
DiabetesPedigreeFunction <= 1.153 gini = 0.38 samples = 34 value = [13, 38] class = Positive
Pregnancies <= 0.5 gini = 0.26 samples = 109 value = [132, 24] class = Negative
gini = 0.404 samples = 38 value = [46, 18] class = Negative
gini = 0.0 samples = 4 value = [0, 6] class = Positive
Age <= 24.0 gini = 0.5 samples = 5 value = [3, 3] class = Negative
gini = 0.363 samples = 15 value = [16, 5] class = Negative
BMI <= 29.7 gini = 0.492 samples = 19 value = [14, 18] class = Positive
Age <= 59.0 gini = 0.33 samples = 32 value = [10, 38] class = Positive
gini = 0.0 samples = 2 value = [3, 0] class = Negative
gini = 0.424 samples = 26 value = [25, 11] class = Negative
gini = 0.193 samples = 83 value = [107, 13] class = Negative
gini = 0.0 samples = 1 value = [2, 0] class = Negative
Glucose <= 107.5 gini = 0.375 samples = 4 value = [1, 3] class = Positive
gini = 0.245 samples = 5 value = [1, 6] class = Positive
Age <= 42.5 gini = 0.499 samples = 14 value = [13, 12] class = Negative
Glucose <= 99.0 gini = 0.273 samples = 28 value = [7.0, 36.0] class = Positive
Age <= 61.5 gini = 0.48 samples = 4 value = [3, 2] class = Negative
gini = 0.0 samples = 1 value = [1, 0] class = Negative
gini = 0.0 samples = 3 value = [0, 3] class = Positive
gini = 0.346 samples = 5 value = [7, 2] class = Negative
DiabetesPedigreeFunction <= 0.225 gini = 0.469 samples = 9 value = [6, 10] class = Positive
DiabetesPedigreeFunction <= 0.713 gini = 0.469 samples = 6 value = [5, 3] class = Negative
BMI <= 28.25 gini = 0.108 samples = 22 value = [2, 33] class = Positive
gini = 0.0 samples = 2 value = [3, 0] class = Negative
gini = 0.0 samples = 2 value = [0, 2] class = Positive
gini = 0.0 samples = 3 value = [6, 0] class = Negative
gini = 0.0 samples = 6 value = [0, 10] class = Positive
gini = 0.0 samples = 3 value = [4, 0] class = Negative
DiabetesPedigreeFunction <= 0.961 gini = 0.375 samples = 3 value = [1, 3] class = Positive
gini = 0.0 samples = 1 value = [1, 0] class = Negative
gini = 0.057 samples = 21 value = [1, 33] class = Positive
gini = 0.0 samples = 2 value = [0, 3] class = Positive
gini = 0.0 samples = 1 value = [1, 0] class = Negative

Model Accuracies

In [25]:
```python
# F1 Score
from sklearn.metrics import f1_score
from sklearn.metrics import precision_score, recall_score
test_predictions = {
    "Baseline Model": optimized_y_test_pred,
    "Significant Model": significant_y_test_pred,
    "Greedy Model": greedy_y_test_pred
}
f1_scores = {}
for model_name, predictions in test_predictions.items():
    f1 = f1_score(y_test, predictions)
    f1_scores[model_name] = f1
    print(f"{model_name} F1 Score: {f1:.4f}")

    # Precision and Recall

    precision_scores = {}
    recall_scores = {}

    for model_name, predictions in test_predictions.items():
        precision = precision_score(y_test, predictions)
        recall = recall_score(y_test, predictions)
        precision_scores[model_name] = precision
        recall_scores[model_name] = recall
        print(f"{model_name} Precision: {precision:.4f}")
        print(f"{model_name} Recall: {recall:.4f}")
```

```
Baseline Model F1 Score: 0.6241
Baseline Model Precision: 0.7333
Baseline Model Recall: 0.5432
Significant Model Precision: 0.6212
Significant Model Recall: 0.5062
Greedy Model Precision: 0.7101
Greedy Model Recall: 0.6049
Significant Model F1 Score: 0.5578
Baseline Model Precision: 0.7333
Baseline Model Recall: 0.5432
Significant Model Precision: 0.6212
Significant Model Recall: 0.5062
Greedy Model Precision: 0.7101
Greedy Model Recall: 0.6049
Greedy Model F1 Score: 0.6533
Baseline Model Precision: 0.7333
Baseline Model Recall: 0.5432
Significant Model Precision: 0.6212
Significant Model Recall: 0.5062
Greedy Model Precision: 0.7101
Greedy Model Recall: 0.6049
```