CS 513 C - Knowledge Discovery and Data Mining Project

Problem Definition: Algorithm Performance Analysis for Diabetes Classification

Objective: The primary goal is to analyze and compare the performance of various machine learning algorithms in accurately classifying individuals into categories such as diabetic, prediabetic, or non-diabetic. This involves understanding and quantifying how effectively each algorithm can handle the data provided, make predictions, and how their predictions align with actual clinical diagnoses.

In [ ]:

```

```

In [19]:

```
import pandas as pd
import numpy as np
from sklearn.metrics import accuracy_score, precision_score, recall_score,
import matplotlib.pyplot as plt
```

In [20]:

```
df = pd.read_csv("C:\\Users\prudh\Downloads\Project\Project\diabetes_binary
reduced_df = df.sample(n=10000, random_state=42)
```

Step 1: Exploratory Data Analysis

Data Understanding and Quality Checks

In [21]:

```
reduced_df.head() #Initial Inspection, essential to understand how the data
```

Out[21]:

	Diabetes_binary	HighBP	HighChol	CholCheck	BMI	Smoker	Stroke	HeartDisease
219620	0.0	0.0	0.0	1.0	21.0	0.0	0.0	
132821	0.0	1.0	1.0	1.0	28.0	0.0	0.0	
151862	0.0	0.0	0.0	1.0	24.0	0.0	0.0	
139717	0.0	0.0	0.0	1.0	27.0	1.0	0.0	
239235	0.0	0.0	1.0	1.0	31.0	1.0	0.0	

5 rows × 22 columns

In [22]:

```
reduced_df.shape #helps in understanding data dimensions, validation, quali
```

Out[22]:

```
(10000, 22)
```

In [23]:
```
reduced_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 10000 entries, 219620 to 125546
Data columns (total 22 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   Diabetes_binary       10000 non-null  float64
 1   HighBP                10000 non-null  float64
 2   HighChol              10000 non-null  float64
 3   CholCheck             10000 non-null  float64
 4   BMI                   10000 non-null  float64
 5   Smoker                10000 non-null  float64
 6   Stroke                10000 non-null  float64
 7   HeartDiseaseorAttack  10000 non-null  float64
 8   PhysActivity          10000 non-null  float64
 9   Fruits                10000 non-null  float64
 10  Veggies               10000 non-null  float64
 11  HvyAlcoholConsump     10000 non-null  float64
 12  AnyHealthcare         10000 non-null  float64
 13  NoDocbcCost           10000 non-null  float64
 14  GenHlth               10000 non-null  float64
 15  MentHlth              10000 non-null  float64
 16  PhysHlth              10000 non-null  float64
 17  DiffWalk              10000 non-null  float64
 18  Sex                   10000 non-null  float64
 19  Age                   10000 non-null  float64
 20  Education             10000 non-null  float64
 21  Income                10000 non-null  float64
dtypes: float64(22)
memory usage: 1.8 MB
```

In [24]:
```
reduced_df.describe()
```

Out[24]:

	Diabetes_binary	HighBP	HighChol	CholCheck	BMI	Smok
count	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.0000
mean	0.136000	0.428100	0.423300	0.963100	28.450800	0.4451
std	0.342806	0.494828	0.494107	0.188526	6.481403	0.4970
min	0.000000	0.000000	0.000000	0.000000	12.000000	0.0000
25%	0.000000	0.000000	0.000000	1.000000	24.000000	0.0000
50%	0.000000	0.000000	0.000000	1.000000	27.000000	0.0000
75%	0.000000	1.000000	1.000000	1.000000	31.000000	1.0000
max	1.000000	1.000000	1.000000	1.000000	95.000000	1.0000

8 rows × 22 columns

In [25]:
```
reduced_df.sample(20)
```

Out[25]:

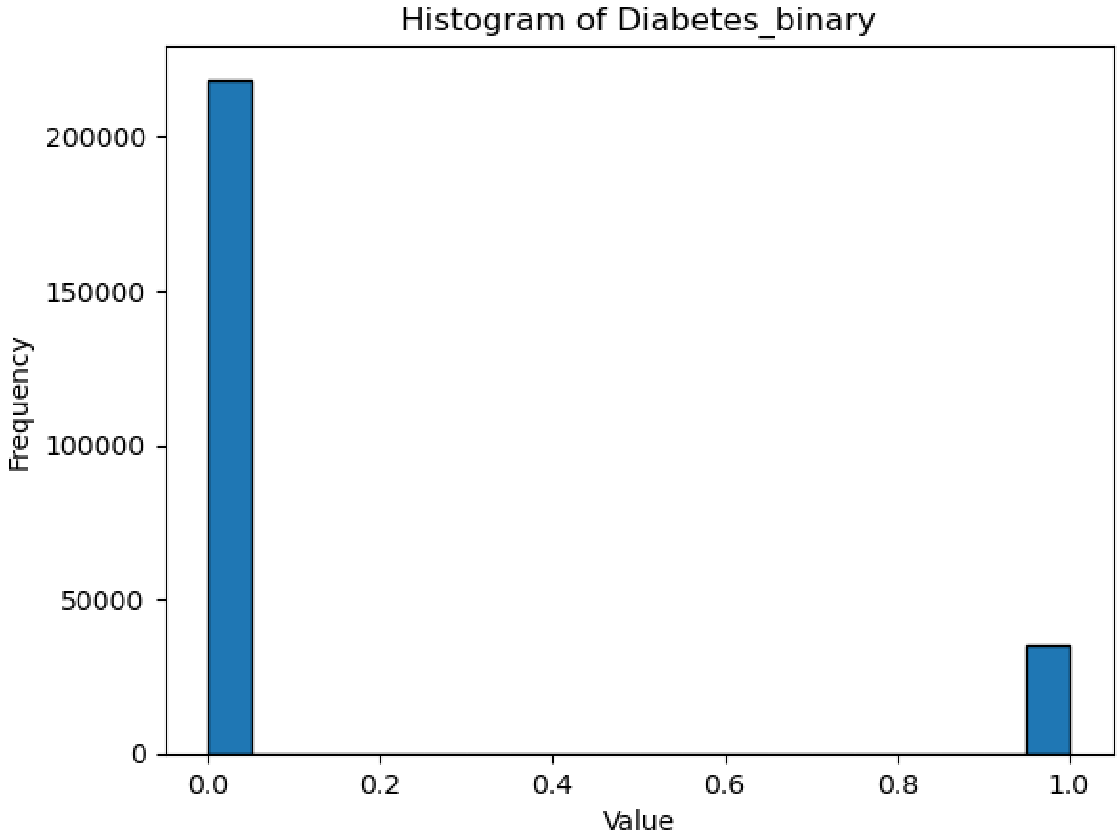
	Diabetes_binary	HighBP	HighChol	CholCheck	BMI	Smoker	Stroke	HeartDisease(
191955	0.0	1.0	1.0	1.0	37.0	0.0	0.0	
118464	0.0	0.0	0.0	1.0	25.0	1.0	0.0	
43112	0.0	1.0	0.0	1.0	17.0	1.0	0.0	
66532	0.0	0.0	0.0	1.0	26.0	0.0	0.0	
245567	0.0	0.0	0.0	1.0	48.0	1.0	0.0	
151028	1.0	1.0	1.0	1.0	28.0	0.0	0.0	
130405	0.0	0.0	0.0	1.0	31.0	0.0	0.0	
187701	0.0	0.0	0.0	1.0	30.0	0.0	0.0	
91650	0.0	0.0	0.0	1.0	27.0	1.0	0.0	
117086	0.0	0.0	1.0	1.0	32.0	1.0	0.0	
43946	0.0	0.0	0.0	1.0	24.0	0.0	0.0	
161908	0.0	0.0	0.0	1.0	27.0	0.0	0.0	
236364	0.0	0.0	0.0	1.0	49.0	0.0	0.0	
59747	0.0	0.0	1.0	0.0	21.0	0.0	0.0	
197068	0.0	0.0	1.0	1.0	24.0	0.0	0.0	
80261	0.0	1.0	1.0	1.0	26.0	0.0	0.0	
117141	0.0	0.0	0.0	1.0	41.0	0.0	0.0	
85241	0.0	0.0	0.0	1.0	28.0	1.0	0.0	
231302	0.0	0.0	0.0	1.0	28.0	0.0	0.0	
68787	0.0	1.0	1.0	1.0	23.0	0.0	0.0	

20 rows × 22 columns

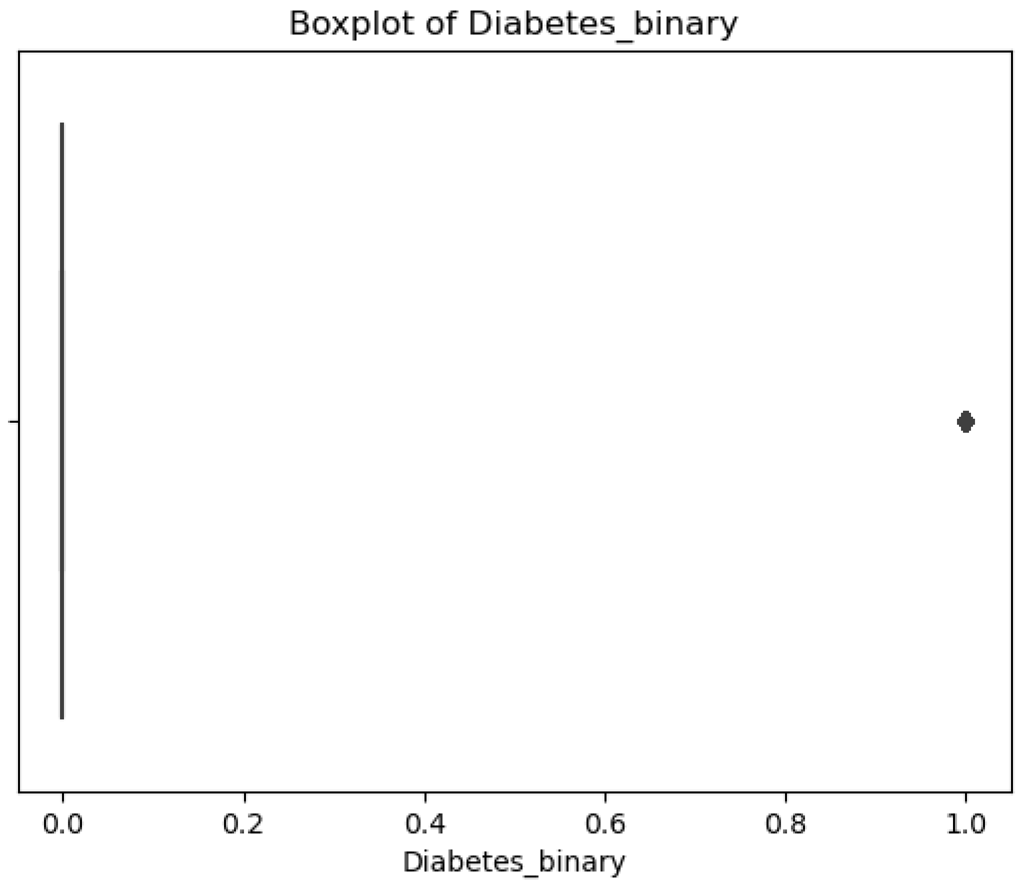
In [26]:
```
reduced_df.isnull().sum() #Checking for the duplicates in the features
```

Out[26]:
```
Diabetes_binary         0
HighBP                  0
HighChol                0
CholCheck               0
BMI                     0
Smoker                  0
Stroke                  0
HeartDiseaseorAttack    0
PhysActivity            0
Fruits                  0
Veggies                 0
HvyAlcoholConsump       0
AnyHealthcare           0
NoDocbcCost             0
GenHlth                 0
MentHlth                0
PhysHlth                0
DiffWalk                0
Sex                     0
Age                     0
Education               0
Income                  0
dtype: int64
```

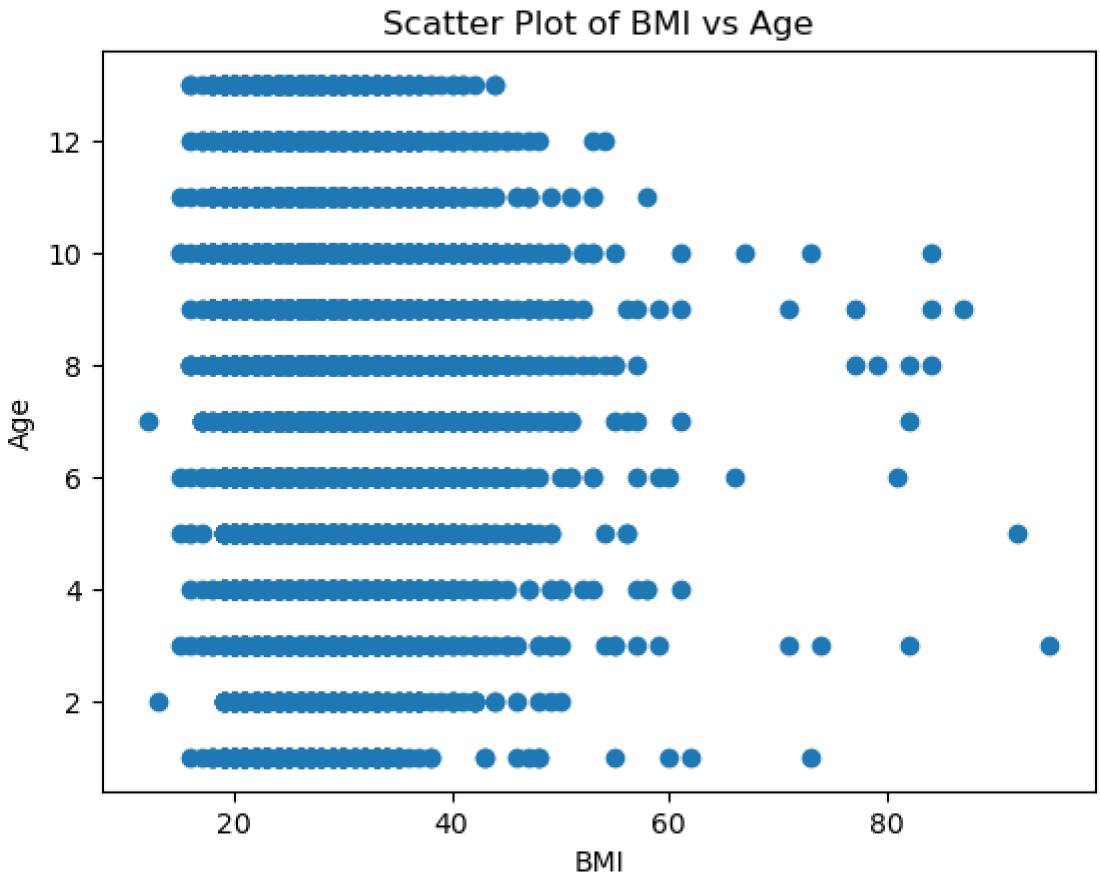
In [27]:
```
# Assuming 'data' is your DataFrame and 'feature_name' is the column you wa
plt.hist(df['Diabetes_binary'], bins=20, edgecolor='black')
plt.title('Histogram of Diabetes_binary')
plt.xlabel('Value')
plt.ylabel('Frequency')
plt.show()
```



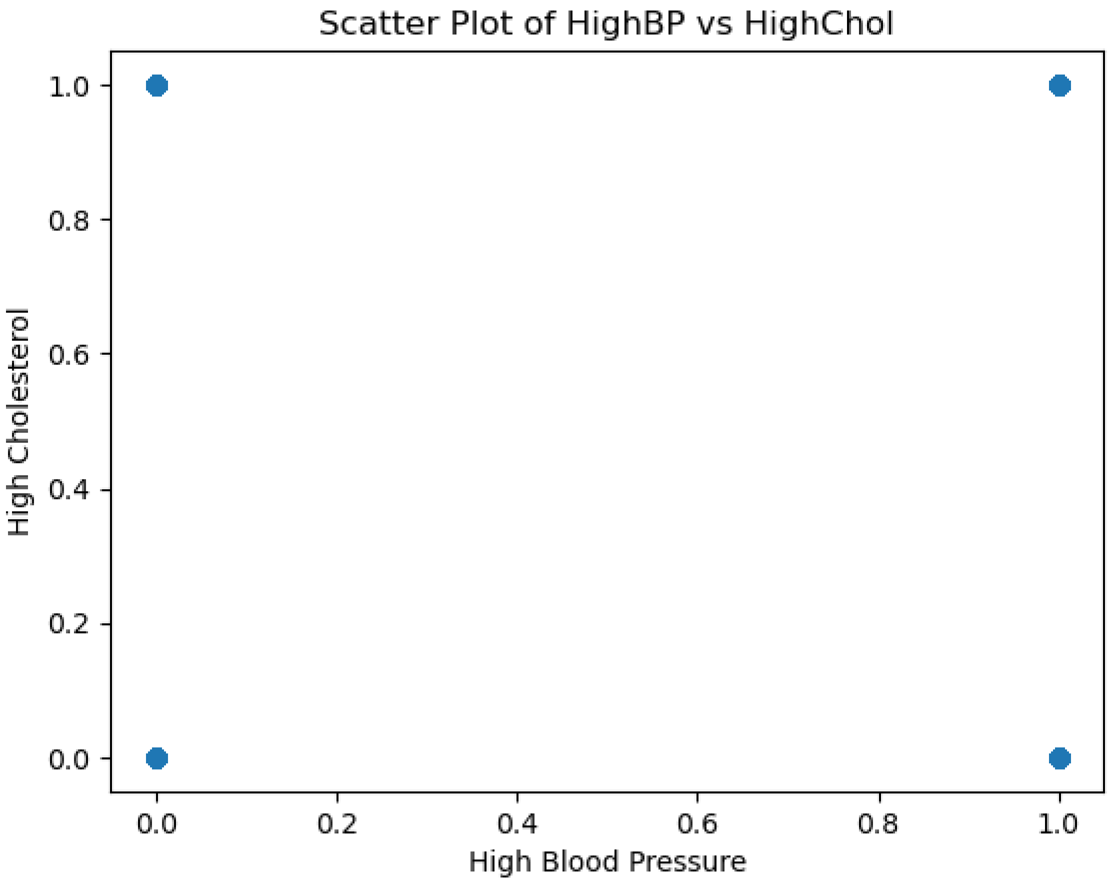
In [28]:
```
import seaborn as sns

sns.boxplot(x=reduced_df['Diabetes_binary'])
plt.title('Boxplot of Diabetes_binary')
plt.xlabel('Diabetes_binary')
plt.show()
```



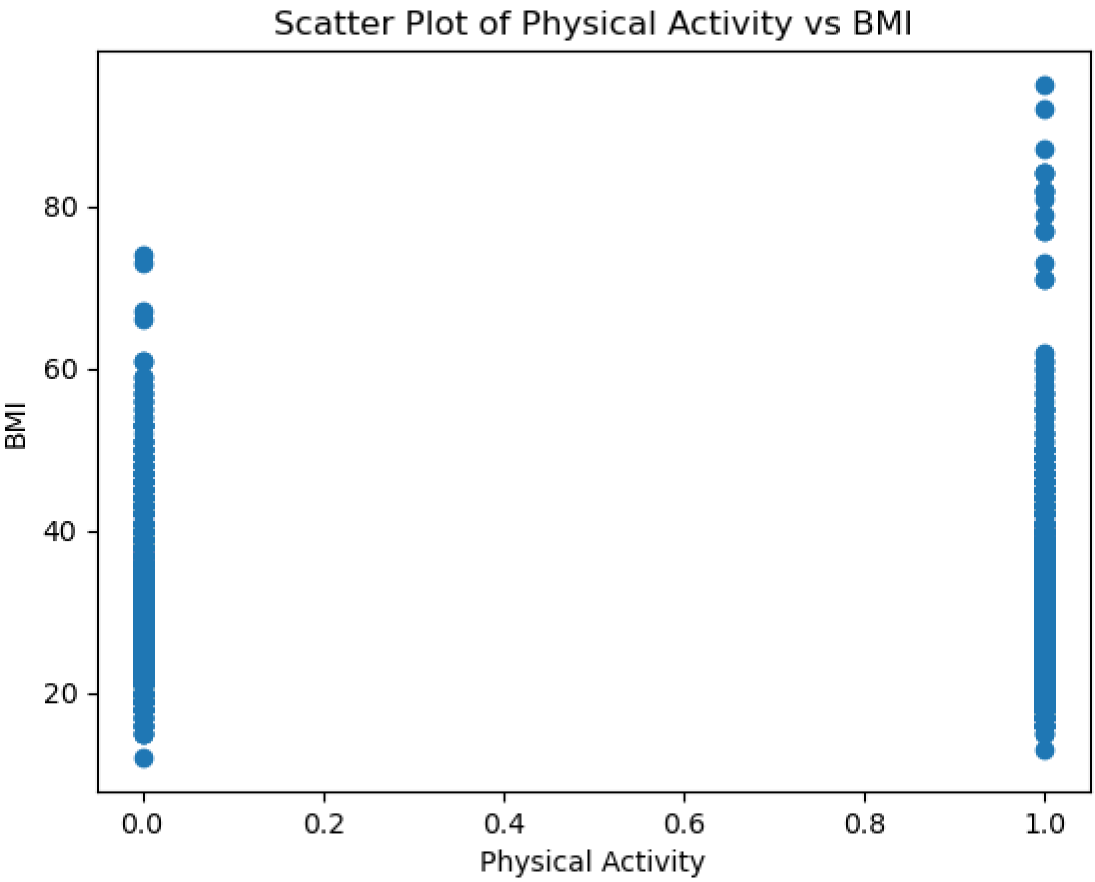
In [29]:
```
plt.scatter(reduced_df['BMI'], reduced_df['Age'])
plt.title('Scatter Plot of BMI vs Age')
plt.xlabel('BMI')
plt.ylabel('Age')
plt.show()
```



In [30]:
```python
plt.scatter(reduced_df['HighBP'], reduced_df['HighChol'])
plt.title('Scatter Plot of HighBP vs HighChol')
plt.xlabel('High Blood Pressure')
plt.ylabel('High Cholesterol')
plt.show()
```



In [31]:
```
plt.scatter(reduced_df['PhysActivity'], reduced_df['BMI'])
plt.title('Scatter Plot of Physical Activity vs BMI')
plt.xlabel('Physical Activity')
plt.ylabel('BMI')
plt.show()
```



In [32]:
```
x = reduced_df.drop(['Diabetes_binary'], axis = 1)
y = reduced_df['Diabetes_binary']
```

In [10]:
```
pip install scikit-learn --upgrade
```

```
Requirement already satisfied: scikit-learn in ./opt/anaconda3/lib/python
3.9/site-packages (1.3.2)
Requirement already satisfied: numpy<2.0,>=1.17.3 in ./opt/anaconda3/lib/p
ython3.9/site-packages (from scikit-learn) (1.21.5)
Requirement already satisfied: joblib>=1.1.1 in ./opt/anaconda3/lib/python
3.9/site-packages (from scikit-learn) (1.3.2)
Requirement already satisfied: scipy>=1.5.0 in ./opt/anaconda3/lib/python
3.9/site-packages (from scikit-learn) (1.9.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in ./opt/anaconda3/li
b/python3.9/site-packages (from scikit-learn) (2.2.0)
Note: you may need to restart the kernel to use updated packages.
```

In [33]:
```
from sklearn.model_selection import train_test_split
# Split the data into training and testing sets
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, ra
```

Naive Bayes

In [44]:

```python
from sklearn.naive_bayes import GaussianNB


# Create a Naive Bayes classifier
classifier = GaussianNB()

# Train the classifier on the training data
classifier.fit(x_train, y_train)

# Make predictions on the testing data
y_pred = classifier.predict(x_test)

# Compute confusion matrix
tn, fp, fn, tp = confusion_matrix(y_test, y_pred).ravel()


# Compute accuracy
accuracy_nb = accuracy_score(y_test, y_pred)
print('Accuracy:', accuracy_nb)

# Compute precision
precision_nb = precision_score(y_test, y_pred)
print('Precision:', precision_nb)

# Compute specificity
specificity_nb = tn / (tn + fp)
print('Specificity:', specificity_nb)


# Compute recall
recall_nb = recall_score(y_test, y_pred)
print('Recall:', recall_nb)

# Compute sensitivity
sensitivity_nb= tp / (tp + fn)
print('Sensitivity:', sensitivity_nb)

# Compute F1-score
f1_nb = f1_score(y_test, y_pred)
print('F1 Score:', f1_nb)

print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test,y_pred))

print(y_pred)
print(y_test)
```

```
Accuracy: 0.7736666666666666
Precision: 0.34347275031685676
Specificity: 0.7982866043613707
Recall: 0.6273148148148148
Sensitivity: 0.6273148148148148
F1 Score: 0.44389844389844385
              precision    recall  f1-score   support

         0.0       0.93      0.80      0.86      2568
         1.0       0.34      0.63      0.44       432

    accuracy                           0.77      3000
   macro avg       0.64      0.71      0.65      3000
weighted avg       0.84      0.77      0.80      3000

[[2050  518]
 [ 161  271]]
[1. 0. 0. ... 1. 0. 0.]
93489     1.0
119707    0.0
51691     0.0
247820    1.0
16592     0.0
         ...
221028    0.0
124217    1.0
43599     1.0
114510    0.0
172978    0.0
Name: Diabetes_binary, Length: 3000, dtype: float64
```

K-Nearest Neighbour

In [36]:
```
from sklearn.neighbors import KNeighborsClassifier
```

In [45]:

```
knn = KNeighborsClassifier(n_neighbors = 3)
knn.fit(x_train,y_train)
y_pred = knn.predict(x_test)


# Compute confusion matrix
tn, fp, fn, tp = confusion_matrix(y_test, y_pred).ravel()
print("Knn Output")

# Compute accuracy
accuracy_knn = accuracy_score(y_test, y_pred)
print('Accuracy:', accuracy_knn)

# Compute precision
precision_knn = precision_score(y_test, y_pred)
print('Precision:', precision_knn)

# Compute specificity
specificity_knn = tn / (tn + fp)
print('Specificity:', specificity_knn)

# Compute recall
recall_knn = recall_score(y_test, y_pred)
print('Recall:', recall_knn)

# Compute sensitivity
sensitivity_knn= tp / (tp + fn)
print('Sensitivity:', sensitivity_knn)

# Compute F1-score
f1_knn = f1_score(y_test, y_pred)
print('F1 Score:', f1_knn)

print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test,y_pred))
```

```
Knn Output
Accuracy: 0.8333333333333334
Precision: 0.3440366972477064
Specificity: 0.9443146417445483
Recall: 0.1736111111111111
Sensitivity: 0.1736111111111111
F1 Score: 0.23076923076923075
              precision    recall  f1-score   support

         0.0       0.87      0.94      0.91      2568
         1.0       0.34      0.17      0.23       432

    accuracy                           0.83      3000
   macro avg       0.61      0.56      0.57      3000
weighted avg       0.80      0.83      0.81      3000

[[2425  143]
 [ 357   75]]
```

CART

In [106]:
```
from sklearn.tree import DecisionTreeClassifier


CART = DecisionTreeClassifier()

CART.fit(x_train, y_train)
y_pred = CART.predict(x_test)

# Compute accuracy
accuracy_cart = accuracy_score(y_test, y_pred)
print('Accuracy:', accuracy_cart)

# Compute precision
precision_cart = precision_score(y_test, y_pred)
print('Precision:', precision_cart)

# Compute specificity
specificity_cart = tn / (tn + fp)
print('Specificity:', specificity_cart)

# Compute recall
recall_cart = recall_score(y_test, y_pred)
print('Recall:', recall_cart)

# Compute sensitivity
sensitivity_cart= tp / (tp + fn)
print('Sensitivity:', sensitivity_cart)

# Compute F1-score
f1_cart = f1_score(y_test, y_pred)
print('F1 Score:', f1_cart)

print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test,y_pred))
```

```
Accuracy: 0.8016666666666666
Precision: 0.3224400871459695
Specificity: 0.963006230529595
Recall: 0.3425925925925926
Sensitivity: 0.19907407407407407
F1 Score: 0.33221099887766553
              precision    recall  f1-score   support

         0.0       0.89      0.88      0.88      2568
         1.0       0.32      0.34      0.33       432

    accuracy                           0.80      3000
   macro avg       0.61      0.61      0.61      3000
weighted avg       0.81      0.80      0.80      3000

[[2257  311]
 [ 284  148]]
```

Decision Tree

In [107]:

```python
from sklearn.tree import DecisionTreeClassifier

DecisionTree = DecisionTreeClassifier(max_depth = 6)
DecisionTree.fit(x_train,y_train)
y_pred = DecisionTree.predict(x_test)

# Compute confusion matrix
tn, fp, fn, tp = confusion_matrix(y_test, y_pred).ravel()
print(" Decision Tree Output")

# Compute accuracy
accuracy_dt = accuracy_score(y_test, y_pred)
print('Accuracy:', accuracy_dt)

# Compute precision
precision_dt = precision_score(y_test, y_pred)
print('Precision:', precision_dt)

# Compute specificity
specificity_dt = tn / (tn + fp)
print('Specificity:', specificity_dt)

# Compute recall
recall_dt = recall_score(y_test, y_pred)
print('Recall:', recall_dt)

# Compute sensitivity
sensitivity_dt= tp / (tp + fn)
print('Sensitivity:', sensitivity_dt)

# Compute F1-score
f1_dt = f1_score(y_test, y_pred)
print('F1 Score:', f1_dt)


print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test,y_pred))
```

```
 Decision Tree Output
Accuracy: 0.8536666666666667
Precision: 0.4808743169398907
Specificity: 0.963006230529595
Recall: 0.2037037037037037
Sensitivity: 0.2037037037037037
F1 Score: 0.28617886178861784
              precision    recall  f1-score   support

         0.0       0.88      0.96      0.92      2568
         1.0       0.48      0.20      0.29       432

    accuracy                           0.85      3000
   macro avg       0.68      0.58      0.60      3000
weighted avg       0.82      0.85      0.83      3000

[[2473   95]
 [ 344   88]]
```

Random Forest

In [108]:
```python
from sklearn.ensemble import RandomForestClassifier

Random_forest = RandomForestClassifier()
Random_forest.fit(x_train,y_train)
y_pred = Random_forest.predict(x_test)
#classification_report(y_test,y_pred)

# Compute confusion matrix
tn, fp, fn, tp = confusion_matrix(y_test, y_pred).ravel()
print("Output Random Forest")

# Compute accuracy
accuracy_rf = accuracy_score(y_test, y_pred)
print('Accuracy:', accuracy_rf)

# Compute precision
precision_rf = precision_score(y_test, y_pred)
print('Precision:', precision_rf)

# Compute specificity
specificity_rf = tn / (tn + fp)
print('Specificity:', specificity_rf)


# Compute recall
recall_rf = recall_score(y_test, y_pred)
print('Recall:', recall_rf)

# Compute sensitivity
sensitivity_rf= tp / (tp + fn)
print('Sensitivity:', sensitivity_rf)

# Compute F1-score
f1_rf = f1_score(y_test, y_pred)
print('F1 Score:', f1_rf)


print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test,y_pred))
```

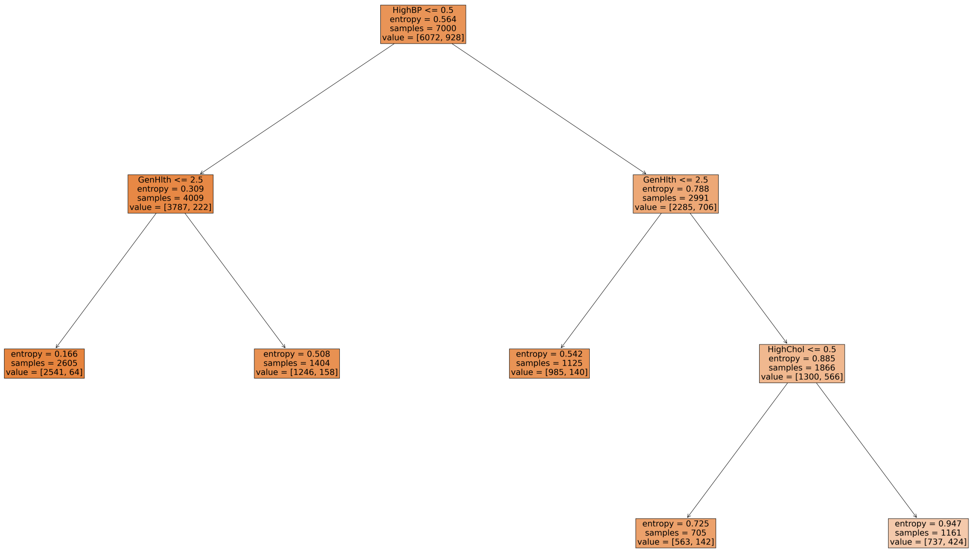
```
Output Random Forest
Accuracy: 0.856
Precision: 0.5
Specificity: 0.9809190031152648
Recall: 0.11342592592592593
Sensitivity: 0.11342592592592593
F1 Score: 0.18490566037735851
              precision    recall  f1-score   support

         0.0       0.87      0.98      0.92      2568
         1.0       0.50      0.11      0.18       432

    accuracy                           0.86      3000
   macro avg       0.68      0.55      0.55      3000
weighted avg       0.82      0.86      0.82      3000

[[2519   49]
 [ 383   49]]
```

C_50

In [109]:

```python
import seaborn as sns
from sklearn.tree import plot_tree


model = DecisionTreeClassifier(criterion='entropy', max_depth=3,splitter='b
model.fit(x_train,y_train)
target_pred = model.predict(x_test)

# Compute accuracy
accuracy_c50 = accuracy_score(y_test, y_pred)
print('Accuracy:', accuracy_c50)

# Compute precision
precision_c50 = precision_score(y_test, y_pred)
print('Precision:', precision_c50)

# Compute specificity
specificity_c50 = tn / (tn + fp)
print('Specificity:', specificity_c50)


# Compute recall
recall_c50 = recall_score(y_test, y_pred)
print('Recall:', recall_c50)

# Compute sensitivity
sensitivity_c50= tp / (tp + fn)
print('Sensitivity:', sensitivity_c50)

# Compute F1-score
f1_c50 = f1_score(y_test, y_pred)
print('F1 Score:', f1_c50)
print(f"\n Classification Report:")
print(classification_report(y_test,y_pred))

plt.figure(figsize=(50,30), dpi=250)
plot_tree(model,fontsize=20,filled=True,feature_names=x.columns);
```

```
Accuracy: 0.856
Precision: 0.5
Specificity: 0.9809190031152648
Recall: 0.11342592592592593
Sensitivity: 0.11342592592592593
F1 Score: 0.1849056603773585

 Classification Report:
              precision    recall  f1-score   support

         0.0       0.87      0.98      0.92      2568
         1.0       0.50      0.11      0.18       432

    accuracy                           0.86      3000
   macro avg       0.68      0.55      0.55      3000
weighted avg       0.82      0.86      0.82      3000
```



Support Vector Machine

In [110]:

```
from sklearn.svm import SVC


svm = SVC(gamma = 'auto')
svm.fit(x_train,y_train)
y_pred = svm.predict(x_test)


# Compute confusion matrix
tn, fp, fn, tp = confusion_matrix(y_test, y_pred).ravel()
print("Support Vector Machines (SVM) Output")

# Compute accuracy
accuracy_svm = accuracy_score(y_test, y_pred)
print('Accuracy:', accuracy_svm)

# Compute precision
precision_svm = precision_score(y_test, y_pred)
print('Precision:', precision_svm)

# Compute specificity
specificity_svm = tn / (tn + fp)
print('Specificity:', specificity_svm)

# Compute recall
recall_svm = recall_score(y_test, y_pred)
print('Recall:', recall_svm)

# Compute sensitivity
sensitivity_svm= tp / (tp + fn)
print('Sensitivity:', sensitivity_svm)

# Compute F1-score
f1_svm = f1_score(y_test, y_pred)
print('F1 Score:', f1_svm)

print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test,y_pred))
```

```
Support Vector Machines (SVM) Output
Accuracy: 0.858
Precision: 0.6071428571428571
Specificity: 0.9957165109034268
Recall: 0.03935185185185185
Sensitivity: 0.03935185185185185
F1 Score: 0.07391304347826087
              precision    recall  f1-score   support

         0.0       0.86      1.00      0.92      2568
         1.0       0.61      0.04      0.07       432

    accuracy                           0.86      3000
   macro avg       0.73      0.52      0.50      3000
weighted avg       0.82      0.86      0.80      3000

[[2557   11]
 [ 415   17]]
```

ANN

In [111]:

```
from sklearn.neural_network import MLPClassifier

ann = MLPClassifier(hidden_layer_sizes=(10,10,10), max_iter = 1000)
ann.fit(x_train,y_train.values.ravel())
y_pred = ann.predict(x_test)


# Compute confusion matrix
tn, fp, fn, tp = confusion_matrix(y_test, y_pred).ravel()
print(" Artifical Neural Network ANN Output")

# Compute accuracy
accuracy_ann = accuracy_score(y_test, y_pred)
print('Accuracy:', accuracy_ann)

# Compute precision
precision_ann = precision_score(y_test, y_pred)
print('Precision:', precision_ann)

# Compute specificity
specificity_ann = tn / (tn + fp)
print('Specificity:', specificity_ann)

# Compute recall
recall_ann = recall_score(y_test, y_pred)
print('Recall:', recall_ann)

# Compute sensitivity
sensitivity_ann= tp / (tp + fn)
print('Sensitivity:', sensitivity_ann)

# Compute F1-score
f1_ann = f1_score(y_test, y_pred)
print('F1 Score:', f1_ann)

print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test,y_pred))
```

```
 Artifical Neural Network ANN Output
Accuracy: 0.862
Precision: 0.6097560975609756
Specificity: 0.9875389408099688
Recall: 0.11574074074074074
Sensitivity: 0.11574074074074074
F1 Score: 0.19455252918287938
              precision    recall  f1-score   support

         0.0       0.87      0.99      0.92      2568
         1.0       0.61      0.12      0.19       432

    accuracy                           0.86      3000
   macro avg       0.74      0.55      0.56      3000
weighted avg       0.83      0.86      0.82      3000

[[2536   32]
 [ 382   50]]
```

In [116]:

```python
# Algorithms used
algorithms = ['NB', 'KNN','CART' ,'DT', 'RF','C_50', 'SVM', 'ANN']

accuracies = [accuracy_nb, accuracy_knn,accuracy_cart , accuracy_dt, accura
precisions = [precision_nb, precision_knn, precision_cart , precision_dt, p
recalls = [recall_nb, recall_knn,recall_cart , recall_dt, recall_rf, recall
f1_scores = [f1_nb, f1_knn,f1_cart ,  f1_dt, f1_rf, f1_svm,f1_c50 , f1_ann]
specificities = [specificity_nb, specificity_knn,specificity_cart , specifi

# Plotting
x = np.arange(len(algorithms))
width = 0.15

fig, ax = plt.subplots()
rects1 = ax.bar(x - width*2, accuracies, width, label='Accuracy')
rects2 = ax.bar(x - width, precisions, width, label='Precision')
rects3 = ax.bar(x, recalls, width, label='Recall')
rects4 = ax.bar(x + width, f1_scores, width, label='F1 Score')
rects5 = ax.bar(x + width*2, specificities, width, label='Specificity')

ax.set_xlabel('Algorithms')
ax.set_ylabel('Scores')
ax.set_title('Scores by Algorithm and Metric')
ax.set_xticks(x)
ax.set_xticklabels(algorithms)
ax.legend()

plt.show()
```