Diabetes Health Indicators: A model that uses self-reported health indicators to predict incidents of diabetes/prediabetes

Data set taken from https://www.kaggle.com/datasets/alexteboul/diabetes-health-indicators-dataset/data The data set "diabetes *binary* health *indicators* BRFSS2015.csv" contains survey responses from the 2015 Behavioral Risk Factor Surveillance System survey conducted by the CDC. This telephone survey collects data health related data from American residents, including information about diabetes and diabetes risk factors/behaviours. The target variable is "diabetes_binary" which classifies responses into no diabetes (0) or diabetes/prediabetes (1).

DATA CLEANING

In [ ]:

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
```

In [ ]:

```
df=pd.read_csv('/content/diabetes_binary_health_indicators_BRFSS2015.csv')
```

In [ ]:

```
df.head()
```

Out[ ]:

	Diabetes_binary	HighBP	HighChol	CholCheck	BMI	Smoker	Stroke	HeartDiseaseorAttack	PhysActivity	Fruits	...	AnyH
0	0.0	1.0	1.0	1.0	40.0	1.0	0.0	0.0	0.0	0.0	...	
1	0.0	0.0	0.0	0.0	25.0	1.0	0.0	0.0	1.0	0.0	...	
2	0.0	1.0	1.0	1.0	28.0	0.0	0.0	0.0	0.0	1.0	...	
3	0.0	1.0	0.0	1.0	27.0	0.0	0.0	0.0	1.0	1.0	...	
4	0.0	1.0	1.0	1.0	24.0	0.0	0.0	0.0	1.0	1.0	...	

5 rows × 22 columns

In [ ]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 253680 entries, 0 to 253679
Data columns (total 22 columns):
 #   Column                Non-Null Count   Dtype
---  ------                --------------   -----
 0   Diabetes_binary       253680 non-null  float64
 1   HighBP                253680 non-null  float64
 2   HighChol              253680 non-null  float64
 3   CholCheck             253680 non-null  float64
 4   BMI                   253680 non-null  float64
 5   Smoker                253680 non-null  float64
 6   Stroke                253680 non-null  float64
 7   HeartDiseaseorAttack  253680 non-null  float64
 8   PhysActivity          253680 non-null  float64
 9   Fruits                253680 non-null  float64
 10  Veggies               253680 non-null  float64
 11  HvyAlcoholConsump     253680 non-null  float64
 12  AnyHealthcare         253680 non-null  float64
 13  NoDocbcCost           253680 non-null  float64
```

```
 13   NoDocbcCost           253680 non-null  float64
 14   GenHlth               253680 non-null  float64
 15   MentHlth              253680 non-null  float64
 16   PhysHlth              253680 non-null  float64
 17   DiffWalk              253680 non-null  float64
 18   Sex                   253680 non-null  float64
 19   Age                   253680 non-null  float64
 20   Education             253680 non-null  float64
 21   Income                253680 non-null  float64
dtypes: float64(22)
memory usage: 42.6 MB
```

In [ ]:

```
df.describe()
```

Out[ ]:

	Diabetes_binary	HighBP	HighChol	CholCheck	BMI	Smoker	Stroke	HeartDis
count	253680.000000	253680.000000	253680.000000	253680.000000	253680.000000	253680.000000	253680.000000	25
mean	0.139333	0.429001	0.424121	0.962670	28.382364	0.443169	0.040571	
std	0.346294	0.494934	0.494210	0.189571	6.608694	0.496761	0.197294	
min	0.000000	0.000000	0.000000	0.000000	12.000000	0.000000	0.000000	
25%	0.000000	0.000000	0.000000	1.000000	24.000000	0.000000	0.000000	
50%	0.000000	0.000000	0.000000	1.000000	27.000000	0.000000	0.000000	
75%	0.000000	1.000000	1.000000	1.000000	31.000000	1.000000	0.000000	
max	1.000000	1.000000	1.000000	1.000000	98.000000	1.000000	1.000000	

8 rows × 22 columns

checking for outliers in each columns

In [ ]:

```
## Run for other columns/ variables
df.loc[df['Diabetes_binary'] >2]
```

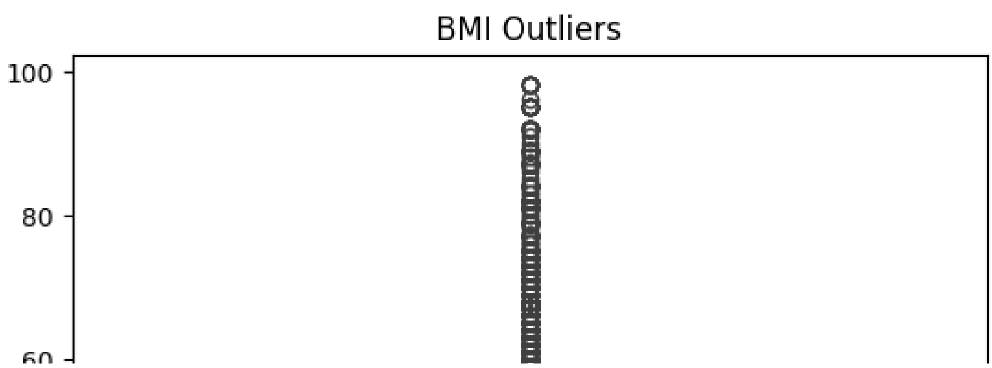
Out[ ]:

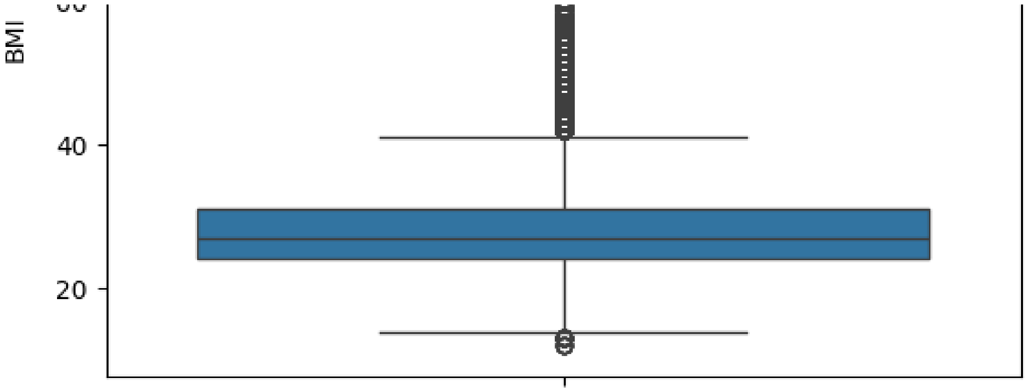
Diabetes_binary	HighBP	HighChol	CholCheck	BMI	Smoker	Stroke	HeartDiseaseorAttack	PhysActivity	Fruits	...	AnyHe

0 rows × 22 columns

In [ ]:

```
# Using Boxplot to identify outliers.
sns.boxplot(df['BMI'])
plt.title("BMI Outliers")
plt.show()
```





In [ ]:

```
from scipy.stats import iqr
```

In [ ]:

```
iqr(df['BMI'])
```

Out[ ]:

```
7.0
```

In [ ]:

```
Q1 = df['BMI'].quantile(0.25)
Q3 = df['BMI'].quantile(0.75)
```

In [ ]:

```
##lower outlier treshold
Q1-1.5*iqr(df['BMI'])
```

Out[ ]:

```
13.5
```

In [ ]:

```
##higher outlier treshold
Q3+1.5*iqr(df['BMI'])
```

Out[ ]:

```
41.5
```

In [ ]:

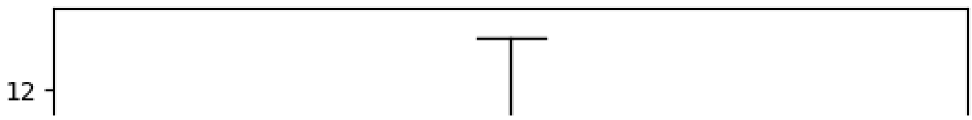
```
##we remove BMI values that are below 13.5 and above 41.5
df=df[(df['BMI']>=13.5) & (df['BMI']<=41.5)]
```

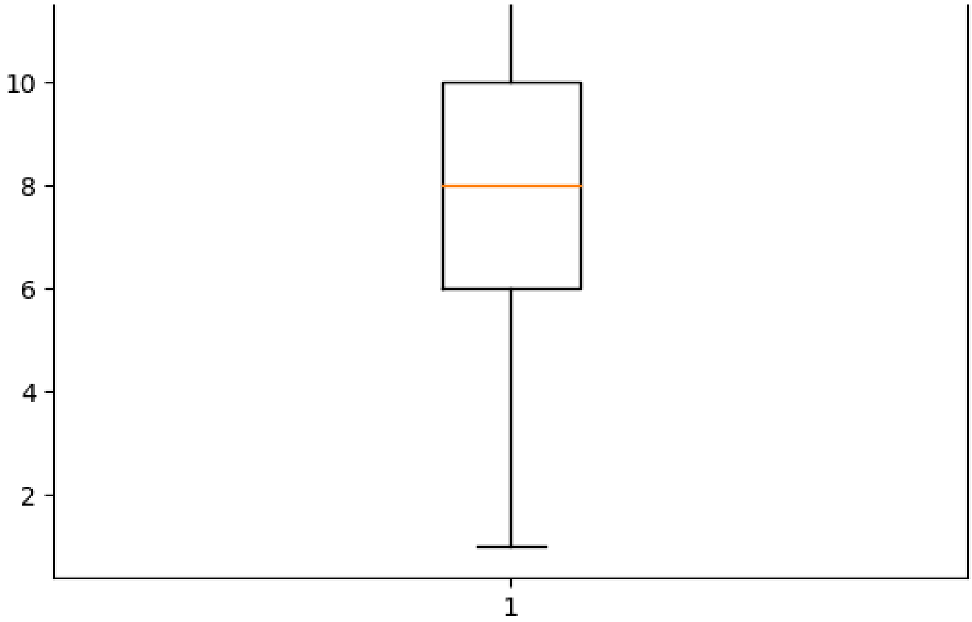
In [ ]:

```
plt.boxplot(df['Age'])
```

Out[ ]:

```
{'whiskers': [<matplotlib.lines.Line2D at 0x7d28f95c9ab0>,
  <matplotlib.lines.Line2D at 0x7d28f95c9d50>],
 'caps': [<matplotlib.lines.Line2D at 0x7d28f95c9ff0>,
  <matplotlib.lines.Line2D at 0x7d28f95ca290>],
 'boxes': [<matplotlib.lines.Line2D at 0x7d28f95c9930>],
 'medians': [<matplotlib.lines.Line2D at 0x7d28f95ca530>],
 'fliers': [<matplotlib.lines.Line2D at 0x7d28f95ca7d0>],
 'means': []}
```





Removing NoDocbcCost, Education and Incnome columns

As we are only focusing on self-reported health indicators, we have decided to remove columns NoDocbcCost (this represents patience who reported not going to the doctor due to medical costs) as well as income and education. As this data is take from a survey of U.S. residents, we also feel that financial barriers to accessing medical care likely differ in Canada and the US.

In [ ]:

```
##removing columns
df=df.drop(['NoDocbcCost','Education','Income'],axis=1)
```

In [ ]:

```
df.describe()
```

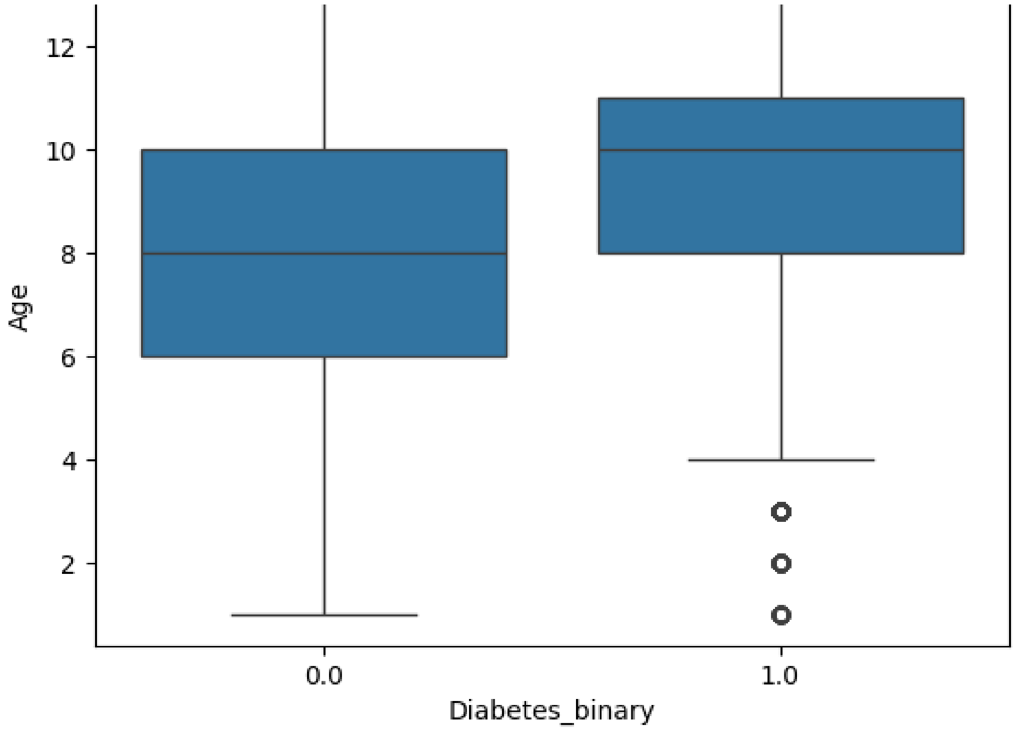
Out[ ]:

	Diabetes_binary	HighBP	HighChol	CholCheck	BMI	Smoker	Stroke	HeartDis
count	243833.000000	243833.000000	243833.000000	243833.000000	243833.000000	243833.000000	243833.000000	2
mean	0.131151	0.420140	0.422027	0.962187	27.569492	0.443398	0.040122	
std	0.337566	0.493582	0.493884	0.190744	4.964920	0.496787	0.196245	
min	0.000000	0.000000	0.000000	0.000000	14.000000	0.000000	0.000000	
25%	0.000000	0.000000	0.000000	1.000000	24.000000	0.000000	0.000000	
50%	0.000000	0.000000	0.000000	1.000000	27.000000	0.000000	0.000000	
75%	0.000000	1.000000	1.000000	1.000000	31.000000	1.000000	0.000000	
max	1.000000	1.000000	1.000000	1.000000	41.000000	1.000000	1.000000	

EXPLORATORY DATA ANALYSIS (EDA)

In [ ]:

```
# checking for relationship between diabetes and age
sns.boxplot(x='Diabetes_binary', y='Age', data=df)
plt.title("Age vs Diabetes Status")
plt.show()
```

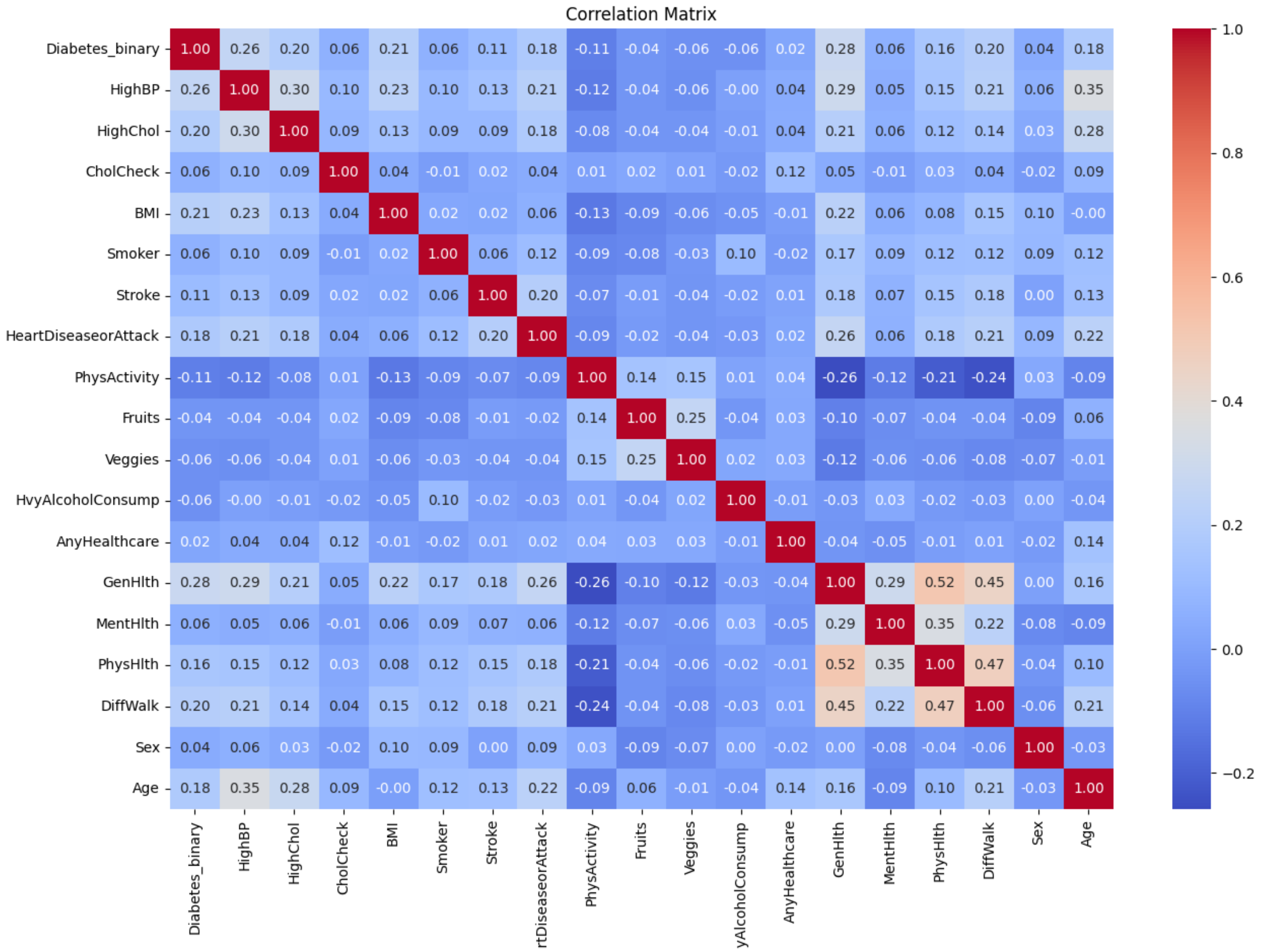


In [ ]:

```
##There seems to be some correlation between diabetes incidence and higher age group.
```

In [ ]:

```
# Correlation matrix
plt.figure(figsize=(15,10))
sns.heatmap(df.corr(), annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Matrix')
plt.show()
```



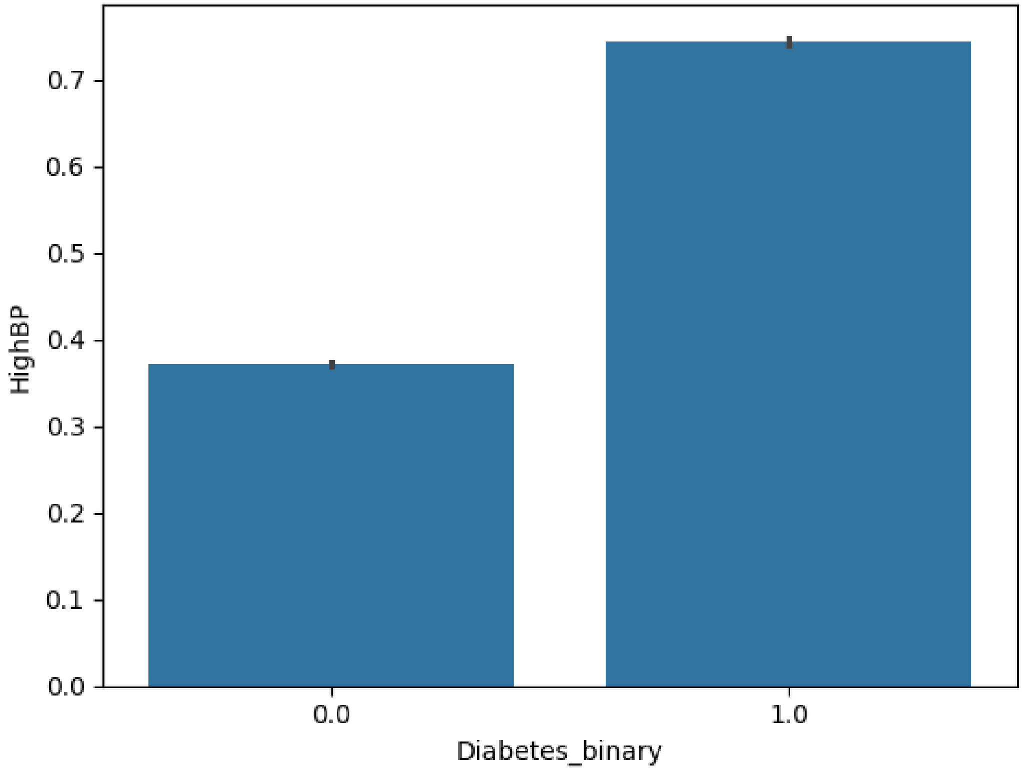
Given the correlation matrix of all the variables, we can see that the variables do not indipendently have a high corralation with each other. We can observe that Age and HighBP have one of the highest correlation, as well as HighCOL and HighBP.

In [ ]:

```
##High Blood Pressure vs diabetes incidence
sns.barplot(x='Diabetes_binary', y='HighBP', data=df)
```

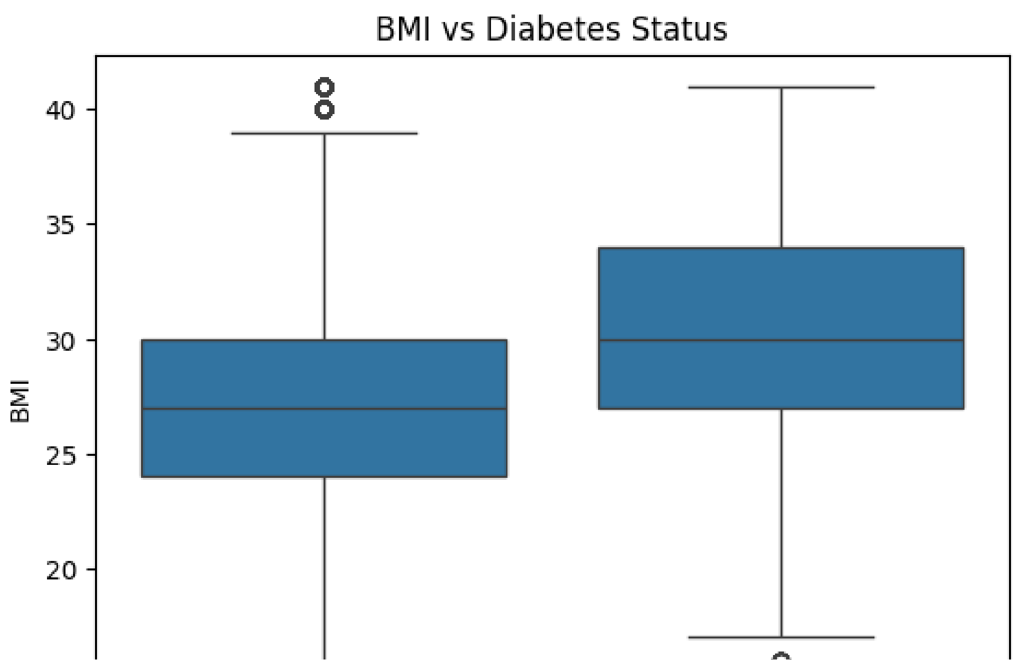
Out[ ]:

```
<Axes: xlabel='Diabetes_binary', ylabel='HighBP'>
```



In [ ]:

```
##BMI vs diabetes incidence: creating a box plot to analyze the incidence between BMI and Diabetes
sns.boxplot(x='Diabetes_binary', y='BMI', data=df)
plt.title("BMI vs Diabetes Status")
plt.show()
```

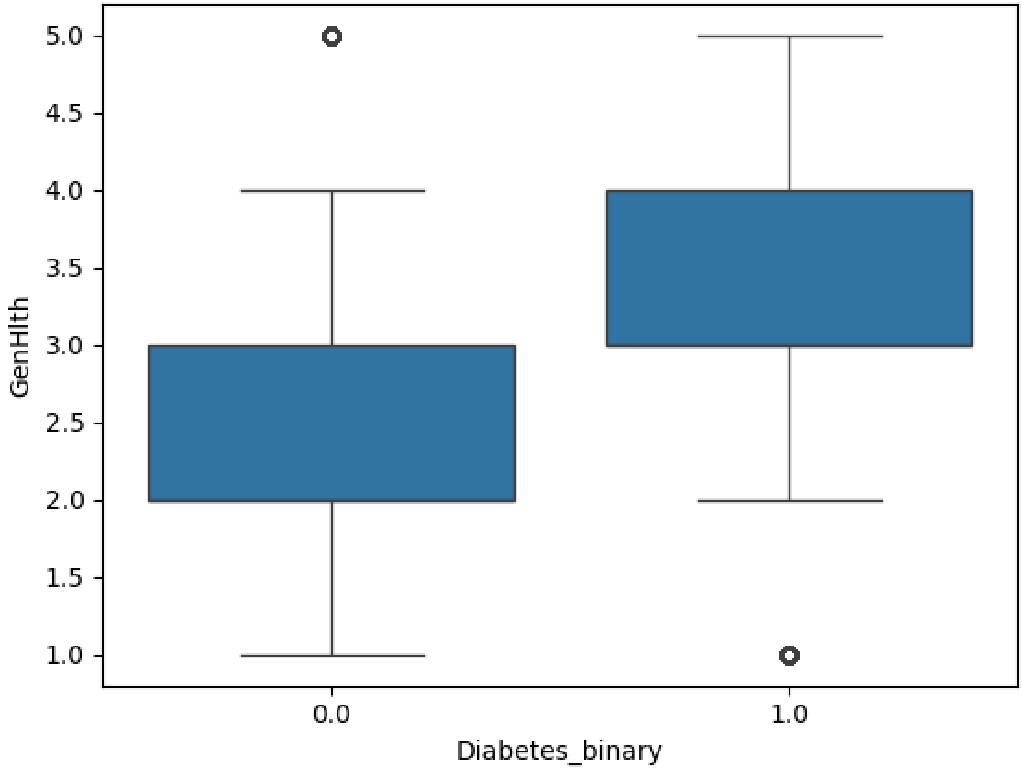


In [ ]:

```
sns.boxplot(x='Diabetes_binary', y='GenHlth', data=df)
```

Out[ ]:

```
<Axes: xlabel='Diabetes_binary', ylabel='GenHlth'>
```



Given the performed Exploratory Data Analysis (EDA), we were able to examine the dataset to highlight its key characteristics and gain insights through visualizations of the variables, as well as their relationships with other variables. We successfully identified correlations between variables and the key relationships among them.

Model Building and Tuning: We proceed to create a model that, once trained, can generalize and predict outcomes on unseen data, aiding in effective decision-making.

In [ ]:

```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from sklearn.preprocessing import StandardScaler
```

In [ ]:

```
# Select features (X) and the target (y)
X = df.drop(columns=['Diabetes_binary'])
y = df['Diabetes_binary']
# Droping rows with missing target values
df = df.dropna(subset=['Diabetes_binary'])
# Selecting the features (X) and the target (y)
X = df.drop(columns=['Diabetes_binary'])
y = df['Diabetes_binary']
```

In [ ]:

```
# Split the dataset into training and testing sets (80% training, 20% testing)
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42
) ## we can change the random forest vale and set between 10 -30

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

We proceeded to adopt the Random Forest classifier due to its ability to handle various data types and complexities. Given the size and number of variables in our dataset, as well as the amount of outliers in our data set, Random Forest was chosen for its robustness, as it is less sensitive to noise and outliers.

In [ ]:

```
# Initializing the Random Forest Classifier
rf_model = RandomForestClassifier(n_estimators=100, random_state=42) # we can change thi
s

# Train the model on the training set
rf_model.fit(X_train_scaled, y_train)
```

Out[ ]:

```
▾       RandomForestClassifier     i ?

RandomForestClassifier(random_state=42)
```

In [ ]:

```
# Make predictions on the test set
y_pred = rf_model.predict(X_test_scaled)
```

In [ ]:

```
# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")
```

```
Accuracy: 0.86
```

In [ ]:

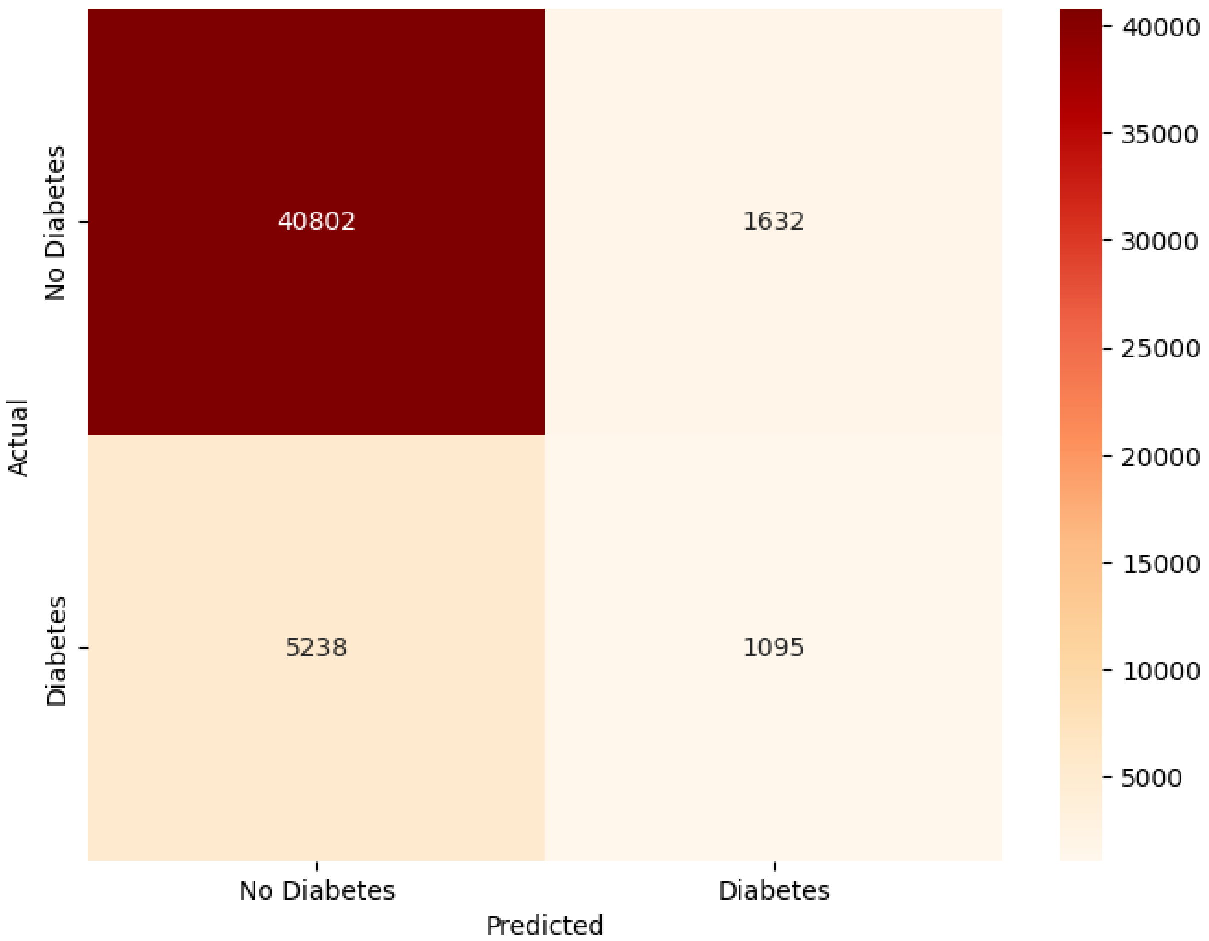
```
# Confusion matrix
cm = confusion_matrix(y_test, y_pred)
print(f"Confusion Matrix:\n{cm}")

# Classification report
report = classification_report(y_test, y_pred)
print(f"Classification Report:\n{report}")
```

```
Confusion Matrix:
[[40802  1632]
 [ 5238  1095]]
Classification Report:
              precision    recall  f1-score   support

         0.0       0.89      0.96      0.92     42434
         1.0       0.40      0.17      0.24      6333

    accuracy                           0.86     48767
   macro avg       0.64      0.57      0.58     48767
weighted avg       0.82      0.86      0.83     48767
```

In [ ]:

```
##Confusion matrix plot
```

```
plt.figure(figsize=(8,6))
sns.heatmap(cm, annot=True, fmt='d', cmap='OrRd', xticklabels=['No Diabetes', 'Diabetes'
], yticklabels=['No Diabetes', 'Diabetes'])

plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.show()
```



Summary of the Model's Performance: Based on the 2X2 matrix, we can summarize each score as: 1) True Negatives: 40,802 - The model correctly identified these cases as people who do not have diabetes. 2) False Positives: 1,632 - These cases were incorrectly predicted as having diabetes when they actually didn't. 3) False Negatives: 5,238 - This number represents people who actually have diabetes but were incorrectly predicted as not having it. 4) True Positives: 1,095 - The model correctly identified these cases as people who have diabetes.

To improve our predictions, we decided to do some hyperparameter tuning. This step is super neccessary because it helps us optimize the settings of our model, By adjusting these hyperparameters, we can find the best configuration that boosts our model's accuracy and performance.

In [ ]:

```
```

In [ ]:

```
#Hyper parameter tuning for Random Forest Classifier
from sklearn.model_selection import GridSearchCV
param_grid=[{
    'n_estimators':[1,10,20],
    'min_samples_split': [5, 10, 15],
    'min_samples_leaf': [ 1,2,4],

}]

grid_search=GridSearchCV(rf_model,
                         param_grid,
                         cv=2,
```

```
                             scoring='accuracy',
                             n_jobs=-1)
grid_search.fit(X_train_scaled, y_train)
```

Out[ ]:

```
▸             GridSearchCV          i ?
▸ best_estimator_: RandomForestClassifier
  ▸       RandomForestClassifier      ?
```

In [ ]:

```
#accuracy scores
best_accuracy=grid_search.best_score_
best_parameters=grid_search.best_params_
print(best_accuracy)
print(best_parameters)
```

```
0.8707104262147171
{'min_samples_leaf': 4, 'min_samples_split': 15, 'n_estimators': 20}
```

In [ ]:

```
##Hyperparameter tuning
param_grid=[{
    'n_estimators':[20,30,50],
    'min_samples_split': [15, 50, 100],
    'min_samples_leaf': [ 4,10,20],

}]

grid_search=GridSearchCV(rf_model,
                             param_grid,
                             cv=2,
                             scoring='accuracy',
                             n_jobs=-1)

grid_search.fit(X_train_scaled, y_train)
```

Out[ ]:

```
▸             GridSearchCV          i ?
▸ best_estimator_: RandomForestClassifier
  ▸       RandomForestClassifier      ?
```

In [ ]:

```
#accuracy scores
best_accuracy=grid_search.best_score_
best_parameters=grid_search.best_params_
print(best_accuracy)
print(best_parameters)
```

```
0.8716690761075738
{'min_samples_leaf': 4, 'min_samples_split': 100, 'n_estimators': 30}
```

In [ ]:

```
#Random Forest with new parameters
# Initializing the Random Forest Classifier
rf_model = RandomForestClassifier(n_estimators=30, min_samples_split=50, min_samples_lea
f=10, random_state=42) # we can change this

# Train the model on the training set
```

```
rf_model.fit(X_train_scaled, y_train)
```

Out[ ]:

▾ RandomForestClassifier i ?

```
RandomForestClassifier(min_samples_leaf=10, min_samples_split=50,
                       n_estimators=30, random_state=42)
```

In [ ]:

```
y_pred = rf_model.predict(X_test_scaled)
```

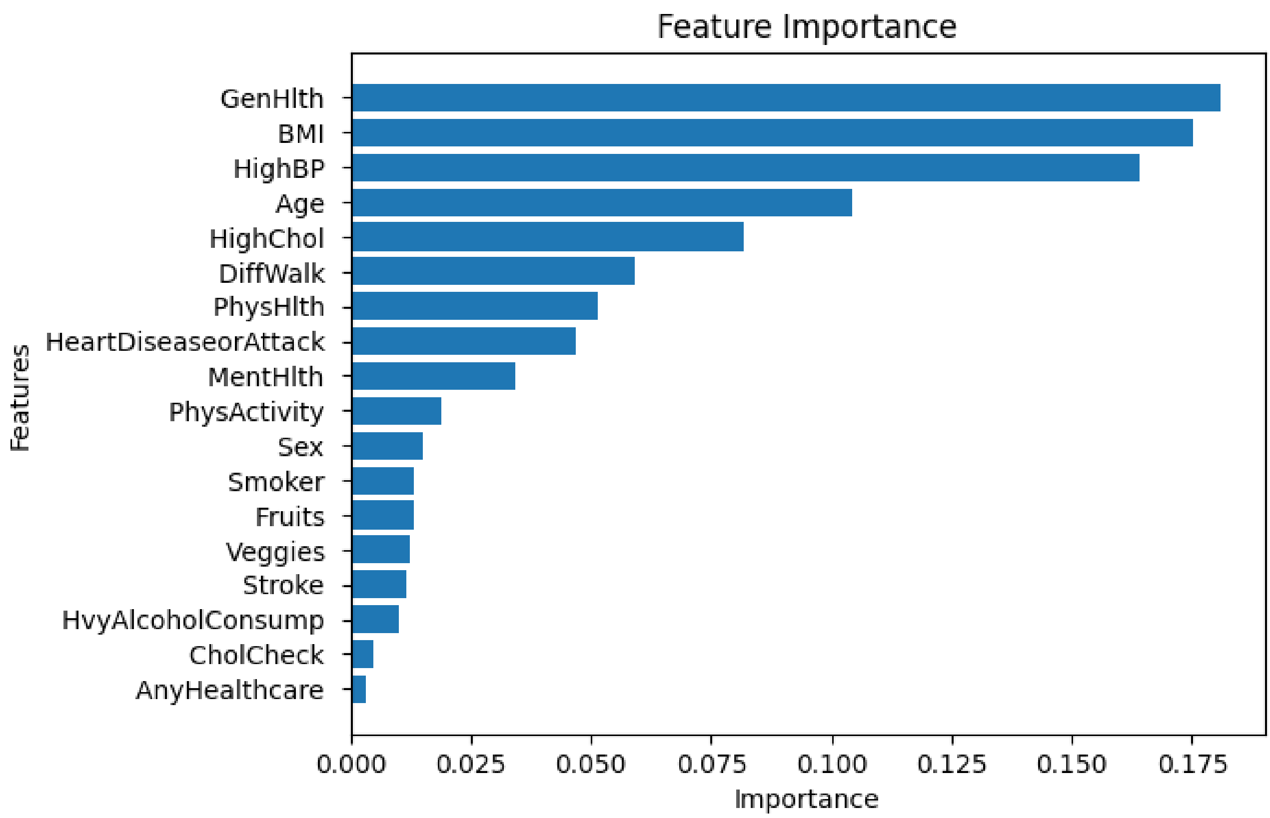
In [ ]:

```
##New Accuracy score
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")
```

```
Accuracy: 0.87
```

In [ ]:

```
##random forest feature selection
from sklearn.feature_selection import SelectFromModel
from sklearn.feature_selection import RFECV

feature_names = [f"{col} " for i, col in enumerate(X.columns)]

f_i = list(zip(feature_names,rf_model.feature_importances_))
f_i.sort(key = lambda x : x[1])

fig, ax = plt.subplots()
plt.barh([x[0] for x in f_i],[x[1] for x in f_i])


ax.set_ylabel("Features")
ax.set_xlabel("Importance")
ax.set_title("Feature Importance")

plt.show()
```



In [ ]:

```
##GenHlth is the most important feature, followed by BMI, highBP and Age
```

Logistic regression is also great for binary classification as it helps in understanding the relationship between features and outcomes.

In [ ]:

```
##Logistic Regression
from sklearn.linear_model import LogisticRegression
LR_model = LogisticRegression()

#fiting the model
LR_model.fit(X_train_scaled, y_train)
```

Out[ ]:

```
▼ LogisticRegression i ?

LogisticRegression()
```

In [ ]:

```
#prediction
y_pred = LR_model.predict(X_test_scaled)
```

In [ ]:

```
#Evaluate the accuracy of the model
print("Accuracy ", LR_model.score(X_test_scaled, y_test)*100)
```

```
Accuracy  87.22086656960649
```

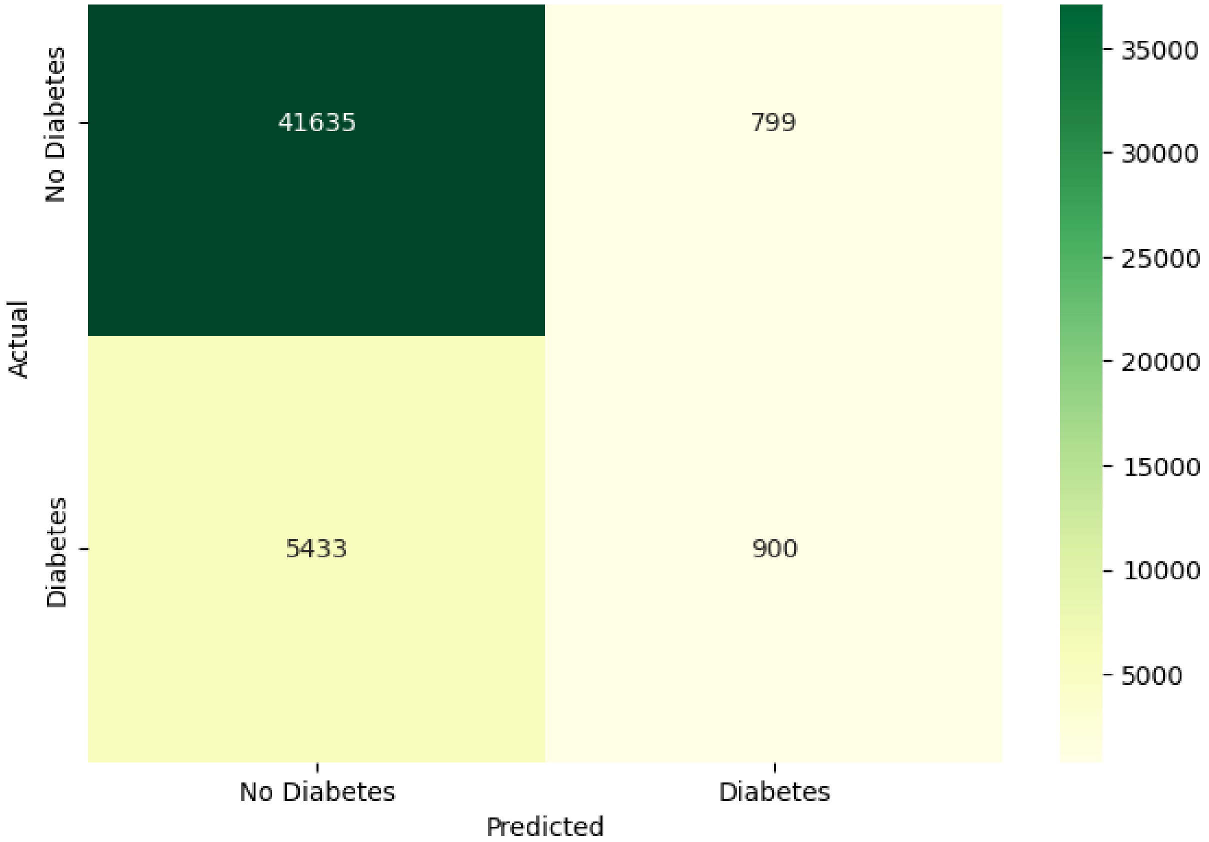
In [ ]:

```
##confusion matrix
cm = confusion_matrix(y_test, y_pred)
print(f"Confusion Matrix:\n{cm}")
# Classification report
report = classification_report(y_test, y_pred)
print(f"Classification Report:\n{report}")
```

```
Confusion Matrix:
[[41635   799]
 [ 5433   900]]
Classification Report:
              precision    recall  f1-score   support

         0.0       0.88      0.98      0.93     42434
         1.0       0.53      0.14      0.22      6333

    accuracy                           0.87     48767
   macro avg       0.71      0.56      0.58     48767
weighted avg       0.84      0.87      0.84     48767
```

In [ ]:

```
#Confusion matrix plot
plt.figure(figsize=(8,6))
sns.heatmap(cm, annot=True, fmt='d', cmap='YlGn', xticklabels=['No Diabetes', 'Diabetes'
], yticklabels=['No Diabetes',  'Diabetes'])

plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.show()
```



Hyper Parameter Tuning for Logistic Regression

In [ ]:

```
##Hypertuning, gridsearch
from scipy.stats import loguniform
from sklearn.linear_model import Ridge
from sklearn.model_selection import RepeatedKFold
from sklearn.model_selection import GridSearchCV
```

In [ ]:

```
logreg_model = LogisticRegression()
logreg_model.fit(X_train_scaled,y_train)
```

Out[ ]:

▾ LogisticRegression i ?

LogisticRegression()

In [ ]:

```
param_grid = [
    {##'penalty':['l1', 'l2'],
    'solver': ['lbfgs','newton-cg','sag','saga'],
    'max_iter'   : [100,200,500,1000]
}
]
```

In [ ]:

```
gridsearch = GridSearchCV(logreg_model,param_grid = param_grid, cv = 3, verbose=True,n_j
obs=-1)
gridsearch
```

Out[ ]:

▸ GridSearchCV i ?

▸ estimator: LogisticRegression

▸ LogisticRegression

In [ ]:

```
best_search = gridsearch.fit(X_train_scaled,y_train)
```

```
Fitting 3 folds for each of 16 candidates, totalling 48 fits
```

In [ ]:

```
print(best_search.best_params_)
```

```
{'max_iter': 100, 'solver': 'lbfgs'}
```

In [ ]:

```
print(f'Accuracy: {best_search.score(X_train_scaled,y_train):.3f}')
```

```
Accuracy: 0.871
```

In [ ]:

```
logreg_model = LogisticRegression(max_iter=500, solver='saga')
logreg_model.fit(X_train_scaled,y_train)
```

Out[ ]:

▾ LogisticRegression

```
LogisticRegression(max_iter=500, solver='saga')
```

In [ ]:

```
y_pred = logreg_model.predict(X_test_scaled)
```

In [ ]:

```
print("Accuracy ", logreg_model.score(X_test_scaled, y_test)*100)
```

```
Accuracy  87.12654048844506
```

Summary of the Model's Performance:

- **True Negatives: 41,635**
- **False Positives: 799**
- **False Negatives: 5,433**
- **True Positives: 900**

***Building* a third classification MODEL :- K-Nearest Neighbors (KNN) classifier:**

In [ ]:

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

X = df.drop(columns='Diabetes_binary')   # All features
y = df['Diabetes_binary']                #  (diabetes status)

# Spliting the dataset into 2: (training and test sets (80% train, 20% test))
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state= 3
0) #trying a radmon state of 30

# Scaling  the features

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

In [ ]:

```
# Initializing the KNN model with a chosen value for k (e.g., k= 15) seting k at 15 due t
o the size of the data
knn = KNeighborsClassifier(n_neighbors=20)

# Training  the model using the training data
knn.fit(X_train_scaled, y_train)
```

Out[ ]:

```
▾ KNeighborsClassifier  i ?

KNeighborsClassifier(n_neighbors=20)
```

In [ ]:

```
# Making predictions on the test set
y_pred = knn.predict(X_test_scaled)

# making predictions and printing out the accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")
```

```
Accuracy: 0.87
```

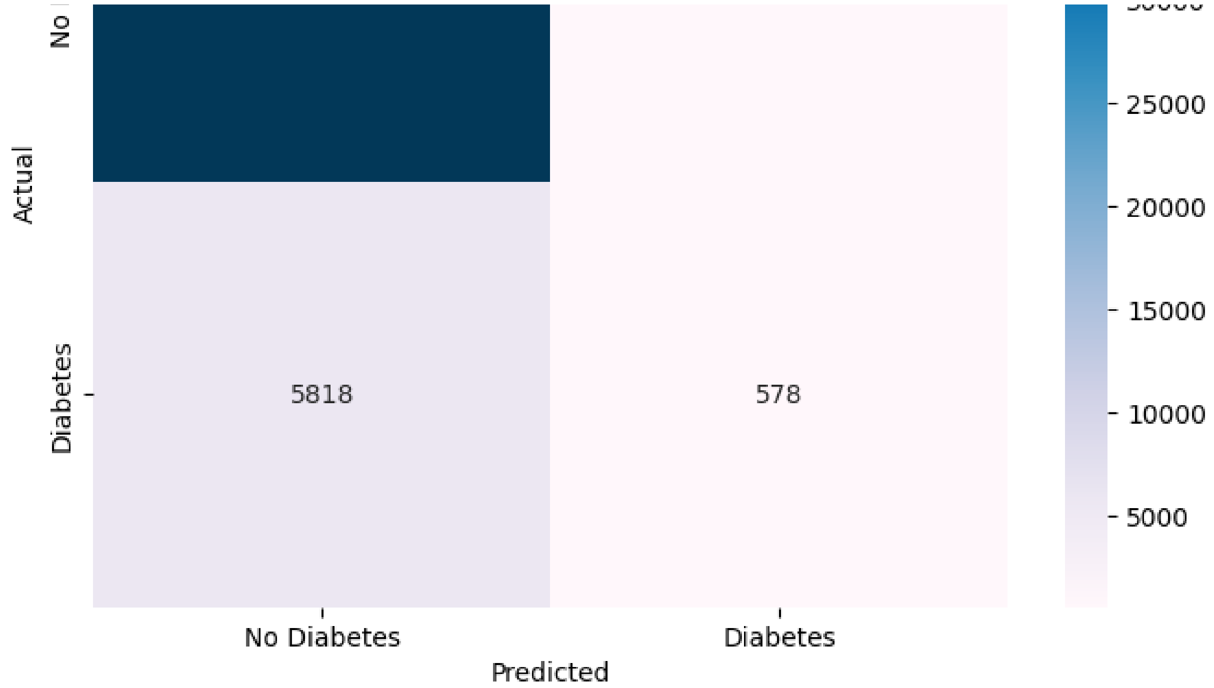
In [ ]:

```
# Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(conf_matrix)

# Classification Report (Precision, Recall, F1-Score for each class)
report = classification_report(y_test, y_pred)
print("Classification Report:")
print(report)
```

```
Confusion Matrix:
[[41820   551]
 [ 5818   578]]
Classification Report:
              precision    recall  f1-score   support

         0.0       0.88      0.99      0.93     42371
         1.0       0.51      0.09      0.15      6396

    accuracy                           0.87     48767
   macro avg       0.69      0.54      0.54     48767
weighted avg       0.83      0.87      0.83     48767
```

In [ ]:

```
plt.figure(figsize=(8,6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='PuBu', xticklabels=['No Diabetes', '
Diabetes'], yticklabels=['No Diabetes',  'Diabetes'])

plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.show()
```



In [ ]:

```
knn_model = KNeighborsClassifier()
knn_model.fit(X_train_scaled,y_train)
```

Out[ ]:

▾ KNeighborsClassifier i ?

KNeighborsClassifier()

In [ ]:

```
param_grid = {
    'n_neighbors': [100,200,500],
    ##'weights': ['uniform', 'distance'],
    ##'leaf_size': [10,20,30],
    ##'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute']
}
```

In [ ]:

```
gridsearch = GridSearchCV(knn_model,param_grid = param_grid, cv = 3, verbose=True,n_jobs
=-1)
gridsearch
```

Out[ ]:

▸ GridSearchCV i ?

▸ estimator: KNeighborsClassifier

▸ KNeighborsClassifier ?

In [ ]:

```
best_search = gridsearch.fit(X_train_scaled,y_train)
```

Fitting 3 folds for each of 3 candidates, totalling 9 fits

In [ ]:

```
print(best_search.best_score_)
```

0.8703618262536783

In [ ]:

```
print(best_search.best_params_)
```

```
{'n_neighbors': 100}
```

Summary of the Model's Performance:

The model performed similarly to Logistic regression.

- **Accuracy Score: 87**
- **True Negatives: 41,820**
- **False Positives: 551**
- **False Negatives: 5,818**
- **True Positives: 578**

Comparing the Three Models: All three models achieved similar accuracy percentages, with only slight variations in performance. Our goal is to reach an accuracy rate of 90% to 95%. To achieve this, we could consider adopting Support Vector Machine (SVM), which is effective at handling large and complex datasets. However, it’s important to note that SVM can be quite expensive to run.

In [ ]:

```
```