Double-click (or enter) to edit

Clearly define a business problem that can be addressed through the application of analytics. Who does the problem affect? What are the financial and social implications of a potential solution?

Our primary goal is to create a precise diabetes prediction model. This model will power companion robots with health smartwatches, gathering crucial health data for detailed electronic medical records. By analyzing various dataset variables, we aim to offer personalized risk assessments. Our approach leverages advanced predictive modeling techniques to tailor insights to individuals' unique health profiles and demographics.

The financial implications of our solution include potential cost savings within the healthcare system. By accurately identifying individuals at risk of developing diabetes, our predictive model facilitates targeted interventions and preventive measures, ultimately reducing the financial burden associated with diabetes-related healthcare expenditures.Furthermore, the deployment of companion robots equipped with health smartwatches offers opportunities for innovative revenue streams. Our predictive diabetes model aims to save costs in healthcare by identifying those at risk early. With companion robots and health smartwatches, we'll collect data for personalized reports. This helps reduce diabetes-related expenses like hospitalizations. Plus, we can offer premium monitoring services, creating new revenue streams for healthcare providers. Ultimately, our solution promotes financial sustainability while improving patient care

```
#Loading Packages

import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
import warnings
import pickle
import plotly.graph_objects as go
import joblib

from sklearn.metrics import accuracy_score, confusion_matrix, classification_report, mean_squared_error, \
                          mean_absolute_error, ConfusionMatrixDisplay, r2_score, roc_auc_score, roc_curve
from sklearn.tree import DecisionTreeRegressor, plot_tree
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import StratifiedKFold
from sklearn.naive_bayes import GaussianNB
from sklearn.preprocessing import LabelEncoder
from flask import Flask, request, jsonify
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Perceptron
from sklearn.neural_network import MLPClassifier, MLPRegressor
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.preprocessing import MinMaxScaler
from sklearn.linear_model import LogisticRegression
import seaborn as sns
from sklearn import tree

import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense,Activation,Dropout
```

```
#Loading Dataset

#Dataset:
# https://www.kaggle.com/datasets/alexteboul/diabetes-health-indicators-dataset

Diabetes = pd.read_csv('/content/Diabetes.csv')
```

```
Diabetes.head(12)
```

	Outcome	HighBP	HighChol	CholCheck	BMI	Smoker	Stroke	HeartDiseaseorAttack	PhysActivity	Fruits	...	AnyHealthcar
0	0	1	0	1	26	0	0	0	1	0	...	
1	0	1	1	1	26	1	1	0	0	1	...	
2	0	0	0	1	26	0	0	0	1	1	...	

3	0	1	1	1	28	1	0	0	1	1	...
4	0	0	0	1	29	1	0	0	1	1	...
5	0	0	0	1	18	0	0	0	1	1	...
6	0	0	1	1	26	1	0	0	1	1	...
7	0	0	0	1	31	1	0	0	0	1	...
8	0	0	0	1	32	0	0	0	1	1	...
9	0	0	0	1	27	1	0	0	0	1	...
10	0	1	1	1	24	1	0	1	1	1	...
11	0	0	0	1	21	0	0	0	1	1	...

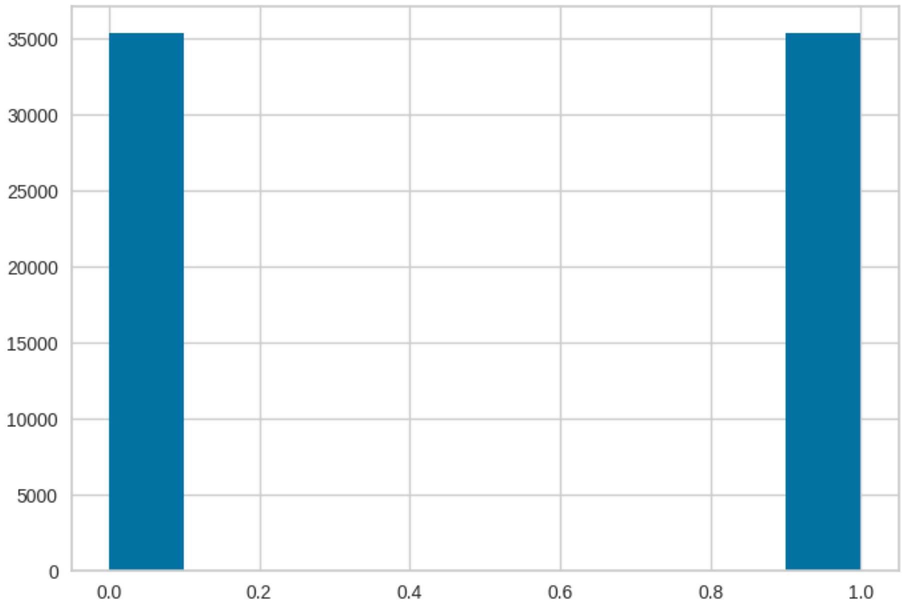
12 rows × 22 columns

```
Diabetes.columns
```

```
Index(['Outcome', 'HighBP', 'HighChol', 'CholCheck', 'BMI', 'Smoker', 'Stroke',
       'HeartDiseaseorAttack', 'PhysActivity', 'Fruits', 'Veggies',
       'HvyAlcoholConsump', 'AnyHealthcare', 'NoDocbcCost', 'GenHlth',
       'MentHlth', 'PhysHlth', 'DiffWalk', 'Sex', 'Age', 'Education',
       'Income'],
      dtype='object')
```

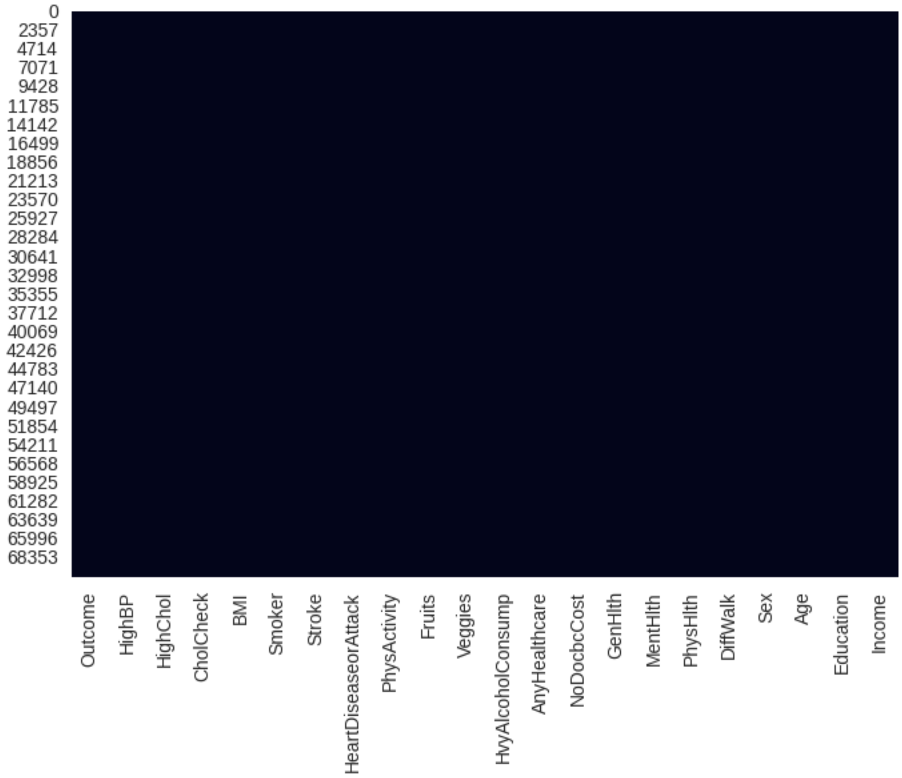
```
# Exploring the distribution of the target variable

Diabetes['Outcome'].hist()
```

```
<Axes: >
```




```
#Examine missing values

sns.heatmap(Diabetes.isnull(), cbar=False)
```

```
<Axes: >
```




```
#Cleaning the dataset to remove columns that will not be used in the analysis

Remove=['CholCheck','Fruits','Veggies','AnyHealthcare','NoDocbcCost','GenHlth','PhysHlth','DiffWalk','Sex', 'Education','Inc

Diabetes_clean = Diabetes.drop(columns=Remove)
```

```
Diabetes_clean.columns
```

```
Index(['Outcome', 'HighBP', 'HighChol', 'BMI', 'Smoker', 'Stroke',
       'HeartDiseaseorAttack', 'PhysActivity', 'HvyAlcoholConsump', 'MentHlth',
       'Age'],
      dtype='object')
```

```
# Summary statistics

Diabetes_clean.describe().transpose()
```

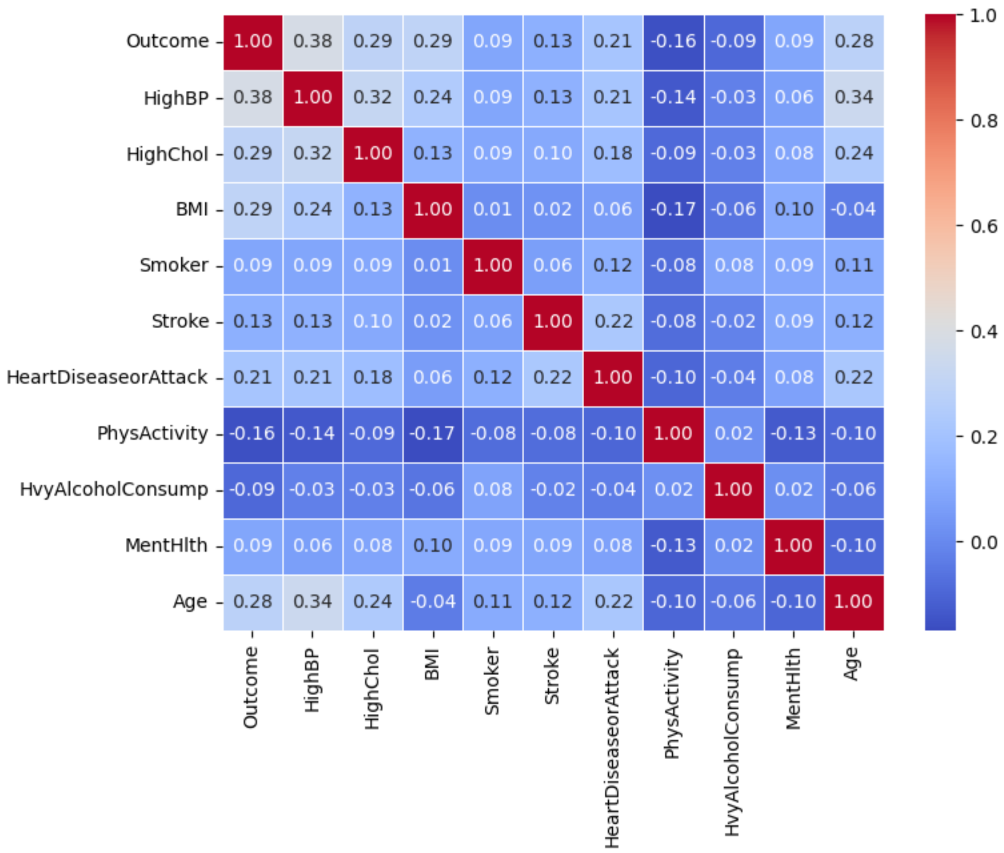
	count	mean	std	min	25%	50%	75%	max
Outcome	70692.0	0.500000	0.500004	0.0	0.0	0.5	1.0	1.0
HighBP	70692.0	0.563458	0.495960	0.0	0.0	1.0	1.0	1.0
HighChol	70692.0	0.525703	0.499342	0.0	0.0	1.0	1.0	1.0
BMI	70692.0	29.856985	7.113954	12.0	25.0	29.0	33.0	98.0
Smoker	70692.0	0.475273	0.499392	0.0	0.0	0.0	1.0	1.0
Stroke	70692.0	0.062171	0.241468	0.0	0.0	0.0	0.0	1.0
HeartDiseaseorAttack	70692.0	0.147810	0.354914	0.0	0.0	0.0	0.0	1.0
PhysActivity	70692.0	0.703036	0.456924	0.0	0.0	1.0	1.0	1.0
HvyAlcoholConsump	70692.0	0.042721	0.202228	0.0	0.0	0.0	0.0	1.0
MentHlth	70692.0	3.752037	8.155627	0.0	0.0	0.0	2.0	30.0
Age	70692.0	8.584055	2.852153	1.0	7.0	9.0	11.0	13.0

```
Diabetes_clean.info()
```

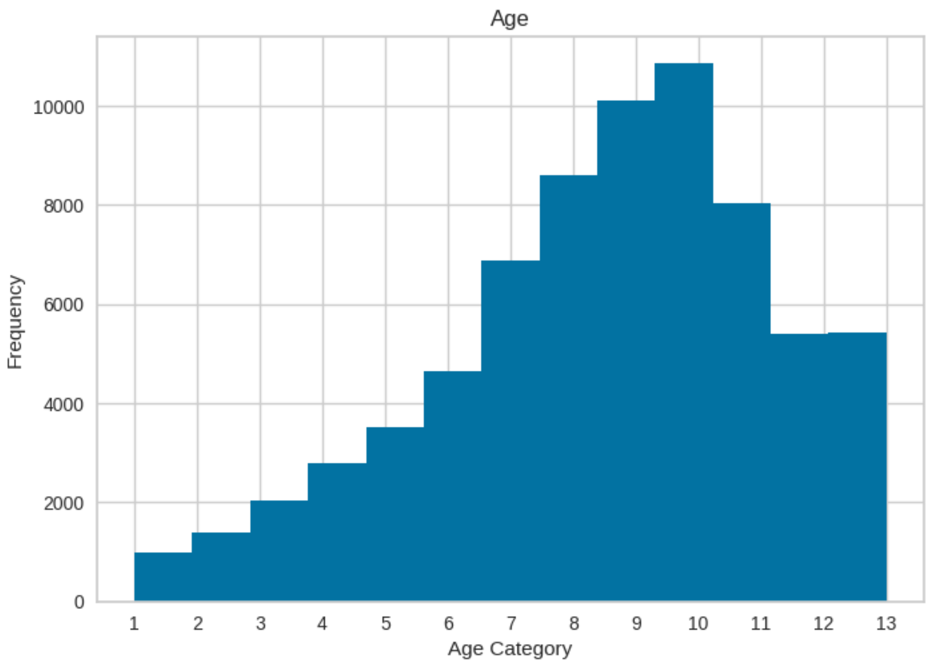
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 70692 entries, 0 to 70691
Data columns (total 11 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   Outcome               70692 non-null  int64
 1   HighBP                70692 non-null  int64
```

```
 2   HighChol              70692 non-null  int64
 3   BMI                   70692 non-null  int64
 4   Smoker                70692 non-null  int64
 5   Stroke                70692 non-null  int64
 6   HeartDiseaseorAttack  70692 non-null  int64
 7   PhysActivity          70692 non-null  int64
 8   HvyAlcoholConsump     70692 non-null  int64
 9   MentHlth              70692 non-null  int64
 10  Age                   70692 non-null  int64
dtypes: int64(11)
memory usage: 5.9 MB
```

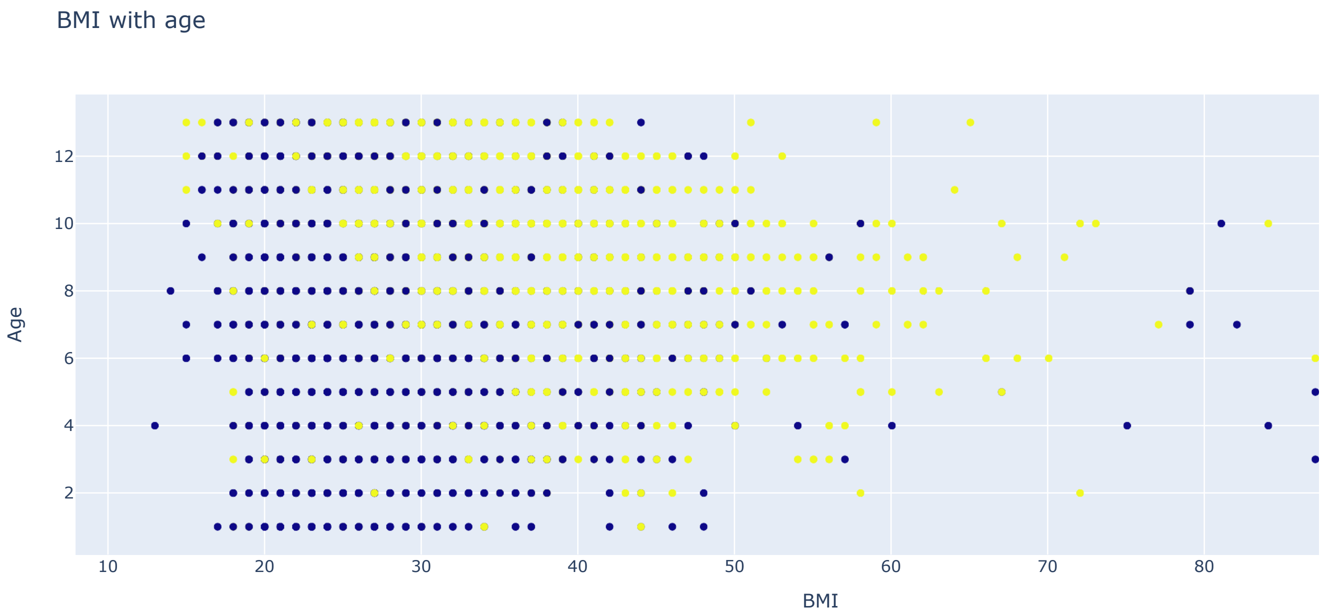
```
# Calculate correlations
correlations = Diabetes_clean.corr()

# Plot heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(correlations, annot=True, cmap='coolwarm', fmt=".2f", linewidths=.5)
plt.show()
```




```
# Plot Age histogram
graph_histograms = Diabetes.hist(column="Age", grid=True, bins=13)

# Calculate the tick positions for 13 bins
tick_positions = range(1, 14)

# Set the x-axis ticks
plt.xticks(tick_positions)

# Optionally, set x-axis and y-axis labels
plt.xlabel("Age Category")
plt.ylabel("Frequency")

# Show the plot
plt.show()
```

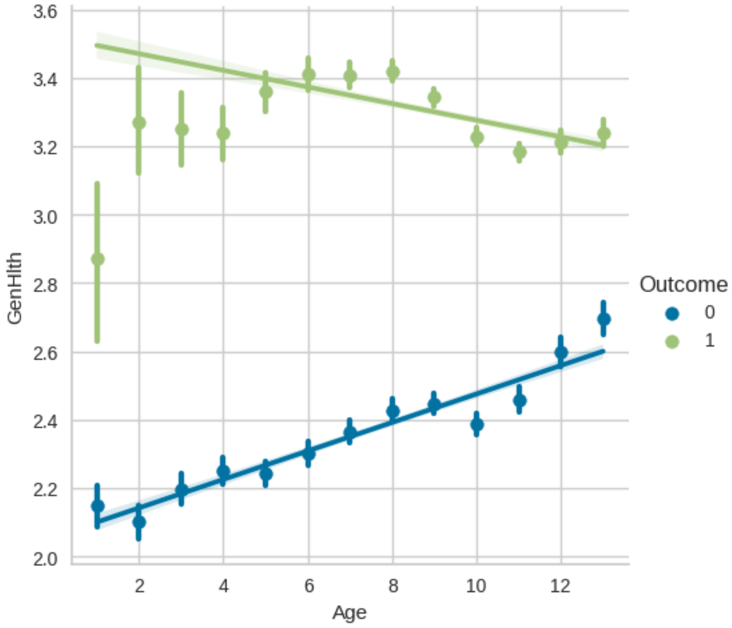



```
#plotting to show corr between bmi and age
import plotly.express as px
px.scatter(Diabetes.sample(10000),
          x='BMI',
           y='Age',
           title='BMI with age',
           color='Outcome')
```




```
# Age/General-Health Scatterplot
sns.lmplot(data=Diabetes, x="Age", y="GenHlth", hue="Outcome", x_bins=1000)
```

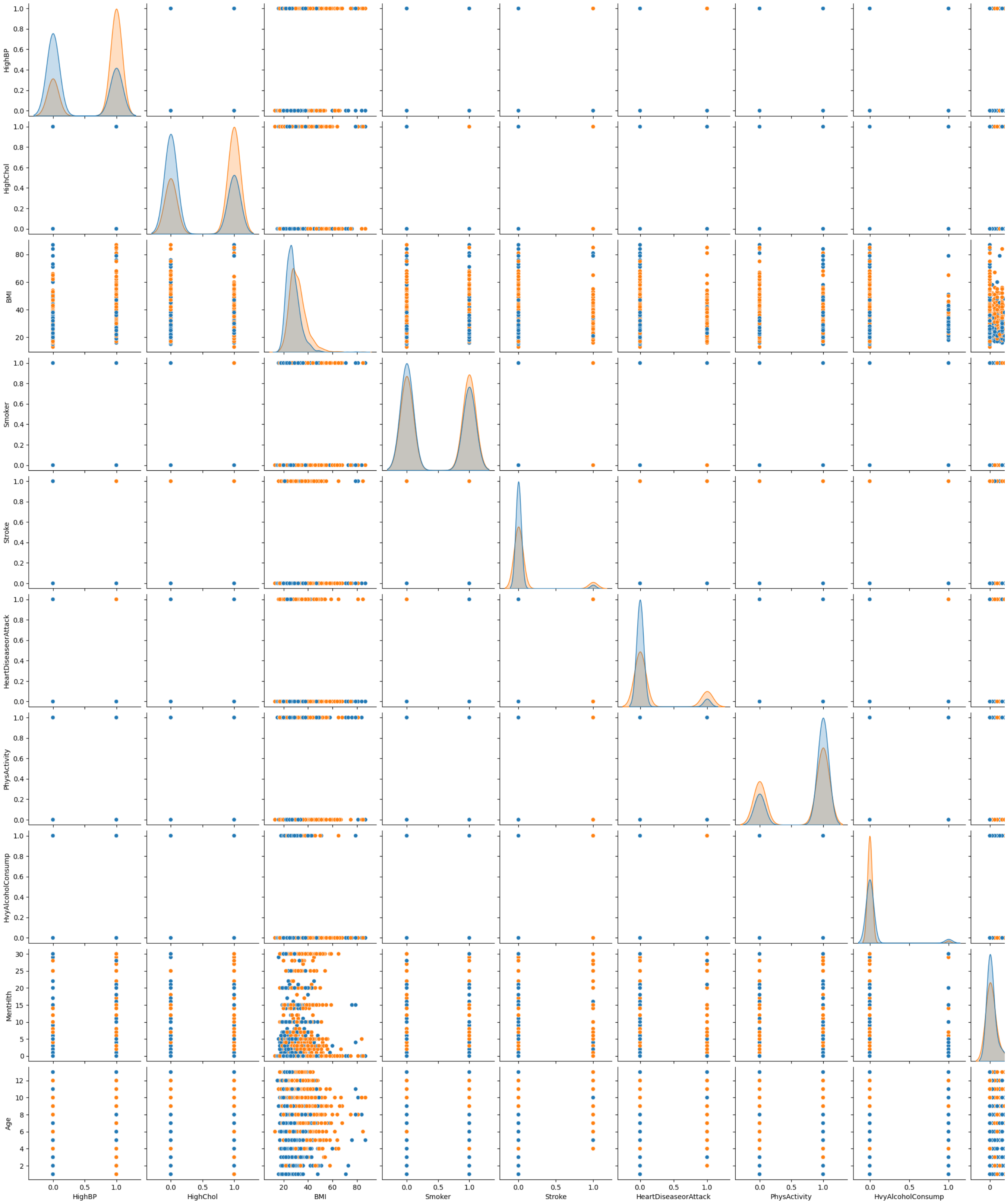
```
<seaborn.axisgrid.FacetGrid at 0x7cd87cd4d8d0>
```



General Health over Age The general health indicator is on a scale of 1 to 5, with a score of 5 as excellent. Here we see that diabetics have poorer general health. We also see a trend in healthy people where the older one gets, the poorer their overall health.

```
Diabetes_subset = Diabetes_clean.sample(n=5000)

# Pairplot with random samples

Diabetes_subset = Diabetes_clean.sample(n=5000)

sns.pairplot(Diabetes_subset, hue = 'Outcome')
```

```
<seaborn.axisgrid.PairGrid at 0x7cf5db27c490>
```



```
Diabetes_subset.shape
```

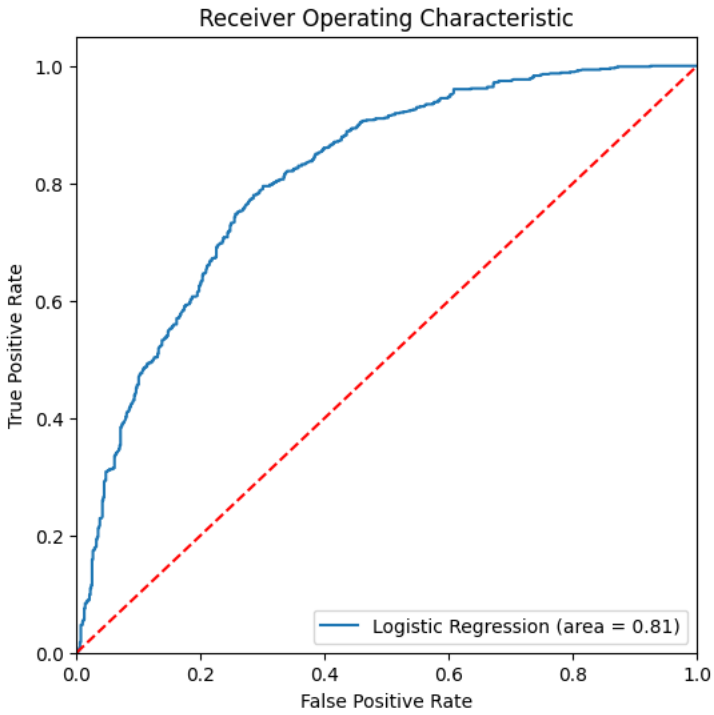
```
(5000, 11)
```

```
# MinMax Scaler transformation

X = Diabetes_subset.drop('Outcome', axis=1)
y = Diabetes_subset['Outcome']

scaler = MinMaxScaler()
X_ = scaler.fit_transform(X)
```

```
X_rescaled = pd.DataFrame(X_, columns=X.columns)


#Building a logistic regression model

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1)

logmodel = LogisticRegression(solver='liblinear')

logmodel.fit(X_train,y_train)

y_pred = logmodel.predict(X_test)

print(logmodel.coef_)
print("\n")
print(confusion_matrix(y_test,y_pred))
print("\n")
print(classification_report(y_test,y_pred))
print("\n")
print('ROC AUC: ', roc_auc_score(y_test,logmodel.predict_proba(X_test)[:,1]))
```

```
[[ 0.80158523  0.6152282   0.08340094  0.04990875  1.02276698  0.46609891
  -0.23209184 -0.96959485  0.01902464  0.17855856]]


[[526 193]
 [190 591]]


              precision    recall  f1-score   support

           0       0.73      0.73      0.73       719
           1       0.75      0.76      0.76       781

    accuracy                           0.74      1500
   macro avg       0.74      0.74      0.74      1500
weighted avg       0.74      0.74      0.74      1500



ROC AUC:  0.8116657970327974
```

```
y_pred_proba = logmodel.predict_proba(X_test)[:,1]

# ROC curve and ROC AUC
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba)
roc_auc = roc_auc_score(y_test, y_pred_proba)

# Plotting the ROC curve
plt.figure(figsize=(6, 6))
plt.plot(fpr, tpr, label='Logistic Regression (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], 'r--')  # Adds the reference line
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic')
plt.legend(loc="lower right")
plt.show()
```

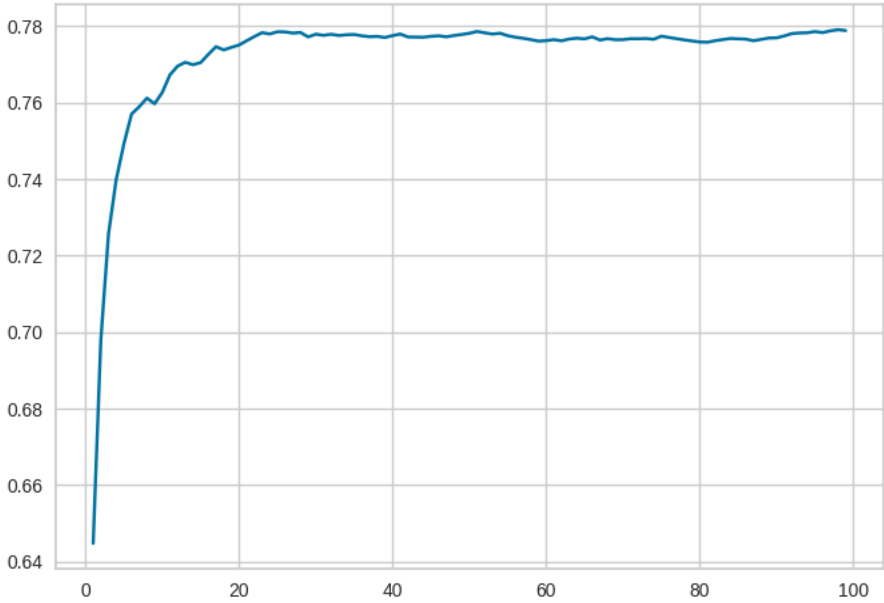



```
#Develop a kNN model using k = 5

X_train, X_test, y_train, y_test = train_test_split(X_rescaled, y, test_size=0.3, random_state=1)

knn = KNeighborsClassifier(n_neighbors=5, metric='euclidean')
knn.fit(X_train, y_train)

y_pred = knn.predict(X_test)

print(confusion_matrix(y_test,y_pred))
print("\n")
print(classification_report(y_test,y_pred))
print("\n")
print('ROC AUC: ', roc_auc_score(y_test,knn.predict_proba(X_test)[:,1]))
```

```
[[539 214]
 [246 501]]


              precision    recall  f1-score   support

           0       0.69      0.72      0.70       753
           1       0.70      0.67      0.69       747

    accuracy                           0.69      1500
   macro avg       0.69      0.69      0.69      1500
weighted avg       0.69      0.69      0.69      1500



ROC AUC:  0.7500635565724607
```

```
# K-optimized

X_train, X_test, y_train, y_test = train_test_split(X_rescaled, y, test_size=0.3, random_state=1)

max_K = 100
cv_scores = [ ]

for K in range(1, max_K):
    knn = KNeighborsClassifier(n_neighbors=K)
    scores = cross_val_score(knn, X_train, y_train.values.ravel(), cv=5, scoring="roc_auc")
    cv_scores.append(scores.mean())

optimal_K_index = np.argmax(cv_scores)
optimal_K = optimal_K_index + 1

print("Optimal K:", optimal_K)
print("\n")
sns.lineplot(x=range(1, max_K), y=cv_scores)
```

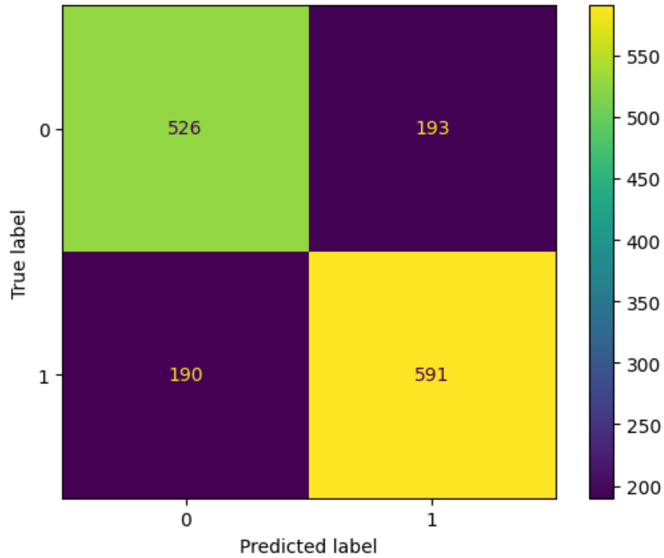
```
Optimal K: 98

<Axes: >
```




```
#Develop a kNN model using optimized k-value

X_train, X_test, y_train, y_test = train_test_split(X_rescaled, y, test_size=0.3, random_state=1)

knn = KNeighborsClassifier(n_neighbors=optimal_K, metric='euclidean')
knn.fit(X_train, y_train)

y_pred = knn.predict(X_test)

print(confusion_matrix(y_test,y_pred))
print("\n")
print(classification_report(y_test,y_pred))
print("\n")
print('ROC AUC: ', roc_auc_score(y_test,knn.predict_proba(X_test)[:,1]))
```

```
[[527 226]
 [218 529]]


              precision    recall  f1-score   support

           0       0.71      0.70      0.70       753
           1       0.70      0.71      0.70       747

    accuracy                           0.70      1500
   macro avg       0.70      0.70      0.70      1500
weighted avg       0.70      0.70      0.70      1500



ROC AUC:  0.7747190621716614
```

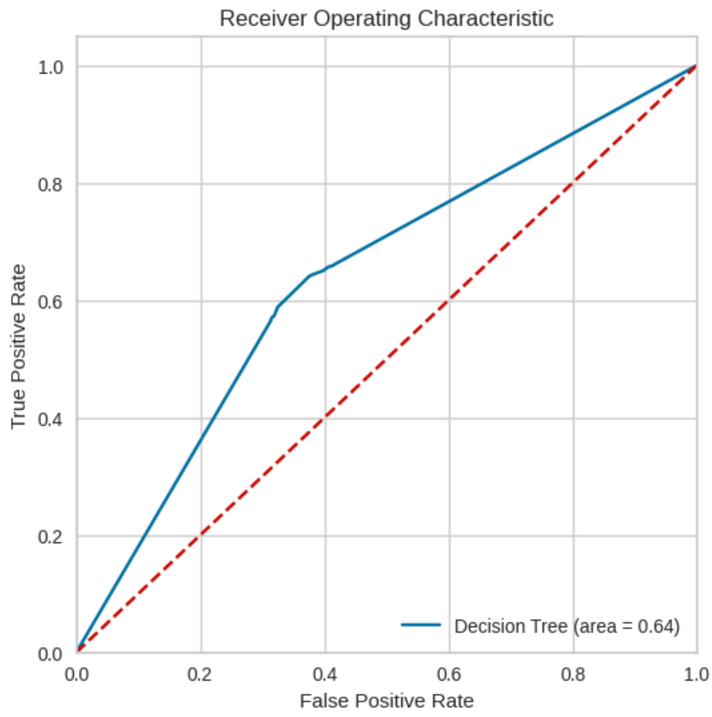
```
graph_confusion_matrix = ConfusionMatrixDisplay.from_predictions(y_test, y_pred)
```



Decision Tree Modelling

The random forest predictive regression model will serve as a foundation for identifying the key variables associated with the likelihood of experiencing diabetes . This model will offer valuable insights to aid in the screening process.

```
# Train a Decision Tree Model

from sklearn.tree import DecisionTreeClassifier

dt_model = DecisionTreeClassifier()
dt_model.fit(X_train, y_train)

y_pred_proba = dt_model.predict_proba(X_test)[:, 1]

# ROC curve and ROC AUC
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba)
roc_auc = roc_auc_score(y_test, y_pred_proba)

# Plotting the ROC curve
plt.figure(figsize=(6, 6))
plt.plot(fpr, tpr, label='Decision Tree (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], 'r--')  # Adds the reference line
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic')
plt.legend(loc="lower right")
plt.show()
```




```
# Random Forest

from sklearn.ensemble import RandomForestClassifier

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1)

rf_model  = RandomForestClassifier(max_depth=5, random_state=0)
rf_model.fit(X_train,y_train)

y_pred_rf = rf_model.predict(X_test)

print(confusion_matrix(y_test,y_pred_rf))
print("\n")
print(classification_report(y_test,y_pred_rf))
print("\n")
print('ROC AUC: ', roc_auc_score(y_test,rf_model.predict_proba(X_test)[:,1]))
```

```
[[532 221]
 [197 550]]


              precision    recall  f1-score   support

           0       0.73      0.71      0.72       753
           1       0.71      0.74      0.72       747

    accuracy                           0.72      1500
   macro avg       0.72      0.72      0.72      1500
weighted avg       0.72      0.72      0.72      1500



ROC AUC:  0.7987905584267125
```

```
# Boosted tree model

from sklearn.ensemble import AdaBoostClassifier

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1)

bt_model = AdaBoostClassifier(n_estimators=100)

bt_model.fit(X_train,y_train)

y_pred_bt = bt_model.predict(X_test)

print(confusion_matrix(y_test,y_pred_bt))
print("\n")
print(classification_report(y_test,y_pred_bt))
print("\n")
print('ROC AUC: ', roc_auc_score(y_test,bt_model.predict_proba(X_test)[:,1]))
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_weight_boosting.py:519: FutureWarning:

The SAMME.R algorithm (the default) is deprecated and will be removed in 1.6. Use the SAMME algorithm to circumvent this

[[550 203]
 [207 540]]

              precision    recall  f1-score   support

           0       0.73      0.73      0.73       753
           1       0.73      0.72      0.72       747

    accuracy                           0.73      1500
   macro avg       0.73      0.73      0.73      1500
weighted avg       0.73      0.73      0.73      1500


ROC AUC:  0.7995363481371258
```

```
# creating the Naive Bayes model
nb_model = GaussianNB()

# separating features and target variable
X = Diabetes_subset.drop('Outcome', axis=1)
y = Diabetes_subset['Outcome']

# using StratifiedKFold to extract the folds
stratified_kfold = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

# replacing data
scores = cross_val_score(nb_model, X, y, cv=stratified_kfold)
print(f'Mean accuracy: {scores.mean()}')
```

```
Mean accuracy: 0.7165999999999999
```

```
from sklearn.metrics import f1_score

# Assuming y_test and y_pred are your true labels and predicted labels respectively
f1 = f1_score(y_test, y_pred)

print(f'F1 Score: {f1:.2f}')
```

```
F1 Score: 0.70
```

```
# Spliting the data 70/30 into training and test datasets for ANN model

X = Diabetes_subset.drop('Outcome',axis=1).values
y = Diabetes_subset['Outcome'].values

X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.30,random_state=1)

scaler = MinMaxScaler()
scaler.fit(X_train)

X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

```
X_train.shape
```

```
(3500, 10)
```

```
# Sequential neural network

model = Sequential()

model.add(Dense(units=500,activation='relu'))
model.add(Dropout(0.5))

model.add(Dense(units=200,activation='relu'))
model.add(Dropout(0.5))

model.add(Dense(units=100,activation='relu'))
model.add(Dropout(0.5))

model.add(Dense(units=50,activation='relu'))
model.add(Dropout(0.5))
```

```
model.add(Dense(units=1,activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam')

from tensorflow.keras.callbacks import EarlyStopping

early_stop = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=5)


model.fit(x=X_train,
          y=y_train,
          batch_size=128,
          epochs=800,
          validation_data=(X_test, y_test), verbose=1,
          callbacks=[early_stop]
          )
```

```
Epoch 1/800
28/28 [==============================] - 3s 31ms/step - loss: 0.6703 - val_loss: 0.6203
Epoch 2/800
28/28 [==============================] - 0s 6ms/step - loss: 0.6200 - val_loss: 0.5874
Epoch 3/800
28/28 [==============================] - 0s 5ms/step - loss: 0.5989 - val_loss: 0.5776
Epoch 4/800
28/28 [==============================] - 0s 5ms/step - loss: 0.5984 - val_loss: 0.5692
Epoch 5/800
28/28 [==============================] - 0s 5ms/step - loss: 0.5841 - val_loss: 0.5678
Epoch 6/800
28/28 [==============================] - 0s 5ms/step - loss: 0.5840 - val_loss: 0.5617
Epoch 7/800
28/28 [==============================] - 0s 5ms/step - loss: 0.5752 - val_loss: 0.5647
Epoch 8/800
28/28 [==============================] - 0s 6ms/step - loss: 0.5666 - val_loss: 0.5574
Epoch 9/800
28/28 [==============================] - 0s 5ms/step - loss: 0.5723 - val_loss: 0.5548
Epoch 10/800
28/28 [==============================] - 0s 5ms/step - loss: 0.5657 - val_loss: 0.5516
Epoch 11/800
28/28 [==============================] - 0s 5ms/step - loss: 0.5587 - val_loss: 0.5557
Epoch 12/800
28/28 [==============================] - 0s 5ms/step - loss: 0.5562 - val_loss: 0.5498
Epoch 13/800
28/28 [==============================] - 0s 5ms/step - loss: 0.5562 - val_loss: 0.5589
Epoch 14/800
28/28 [==============================] - 0s 5ms/step - loss: 0.5511 - val_loss: 0.5437
Epoch 15/800
28/28 [==============================] - 0s 5ms/step - loss: 0.5574 - val_loss: 0.5479
Epoch 16/800
28/28 [==============================] - 0s 5ms/step - loss: 0.5582 - val_loss: 0.5583
Epoch 17/800
28/28 [==============================] - 0s 5ms/step - loss: 0.5472 - val_loss: 0.5476
Epoch 18/800
28/28 [==============================] - 0s 5ms/step - loss: 0.5466 - val_loss: 0.5454
Epoch 19/800
28/28 [==============================] - 0s 5ms/step - loss: 0.5473 - val_loss: 0.5515
Epoch 19: early stopping
<keras.src.callbacks.History at 0x7cd7d17fbc10>
```

```
# Plot of training and validation losses versus epochs

model_loss = pd.DataFrame(model.history.history)
model_loss.plot()
```

```
<Axes: >
```


```
# Model confusion matrix
# Model classification report
# Model ROC AUC

y_pred =(model.predict(X_test) > 0.5).astype("int32")

print(confusion_matrix(y_test,y_pred))
print(classification_report(y_test,y_pred))
print('ROC AUC: ', roc_auc_score(y_test,model.predict(X_test)))
```

```
47/47 [==============================] - 0s 2ms/step
[[548 205]
 [217 530]]
              precision    recall  f1-score   support

           0       0.72      0.73      0.72       753
           1       0.72      0.71      0.72       747

    accuracy                           0.72      1500
   macro avg       0.72      0.72      0.72      1500
weighted avg       0.72      0.72      0.72      1500

47/47 [==============================] - 0s 2ms/step
ROC AUC:  0.7995567929086865
```

```
!pip install pycaret --quiet
```