Diabetes Prediction

- Goutham Kumar Aare

Determining the Likelihood of Diabetes in Individuals

In this project, I'm working on predicting whether someone might have diabetes. Using machine learning, I'm creating a model that looks at certain details to estimate the chance of diabetes. Throughout this notebook, I'll explore the data, build prediction models, and check how well they work. Let's start digging into the analysis!

In this project, I am utilizing a dataset from Kaggle, specifically sourced from the following link: [Kaggle - Diabetes Dataset (https://www.kaggle.com/datasets/johndasilva/diabetes)](https://www.kaggle.com/datasets/johndasilva/diabetes)

In [241]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
```

In [242]:
```
import io
%cd "C:\Users\gouth\Downloads"
```

```
C:\Users\gouth\Downloads
```

In [243]:
```
diabetes=pd.read_csv("diabetes.csv")
```

Data Exploration

In [244]:
```
# Number of rows and columns
diabetes.shape
```

Out[244]:
```
(2000, 9)
```

In [245]:
```
# Column Names of the data
diabetes.columns
```

Out[245]:
```
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
      dtype='object')
```

In [246]:
```
# Information of Dataset
diabetes.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2000 entries, 0 to 1999
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Pregnancies               2000 non-null   int64  
 1   Glucose                   2000 non-null   int64  
 2   BloodPressure             2000 non-null   int64  
 3   SkinThickness             2000 non-null   int64  
 4   Insulin                   2000 non-null   int64  
 5   BMI                       2000 non-null   float64
 6   DiabetesPedigreeFunction  2000 non-null   float64
 7   Age                       2000 non-null   int64  
 8   Outcome                   2000 non-null   int64  
dtypes: float64(2), int64(7)
memory usage: 140.8 KB
```

In [247]:
```
# Finding Null Values
diabetes.isnull().any()
```

Out[247]:
```
Pregnancies                 False
Glucose                     False
BloodPressure               False
SkinThickness               False
Insulin                     False
BMI                         False
DiabetesPedigreeFunction    False
Age                         False
Outcome                     False
dtype: bool
```

In [248]:
```
# First Five Rows
diabetes.head()
```

Out[248]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction
0	2	138	62	35	0	33.6	0.127
1	0	84	82	31	125	38.2	0.233
2	0	145	0	0	0	44.2	0.630
3	0	135	68	42	250	42.3	0.365
4	1	139	62	41	480	40.7	0.536

In [249]:
```
# Last Five rows
diabetes.tail()
```

Out[249]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunctio
1995	2	75	64	24	55	29.7	0.37
1996	8	179	72	42	130	32.7	0.71
1997	6	85	78	0	0	31.2	0.38
1998	0	129	110	46	130	67.1	0.31
1999	2	81	72	15	76	30.1	0.54

In [250]:
```
# Renaming a Column
diabetes = diabetes.rename(columns={'DiabetesPedigreeFunction':'DPF'})
```

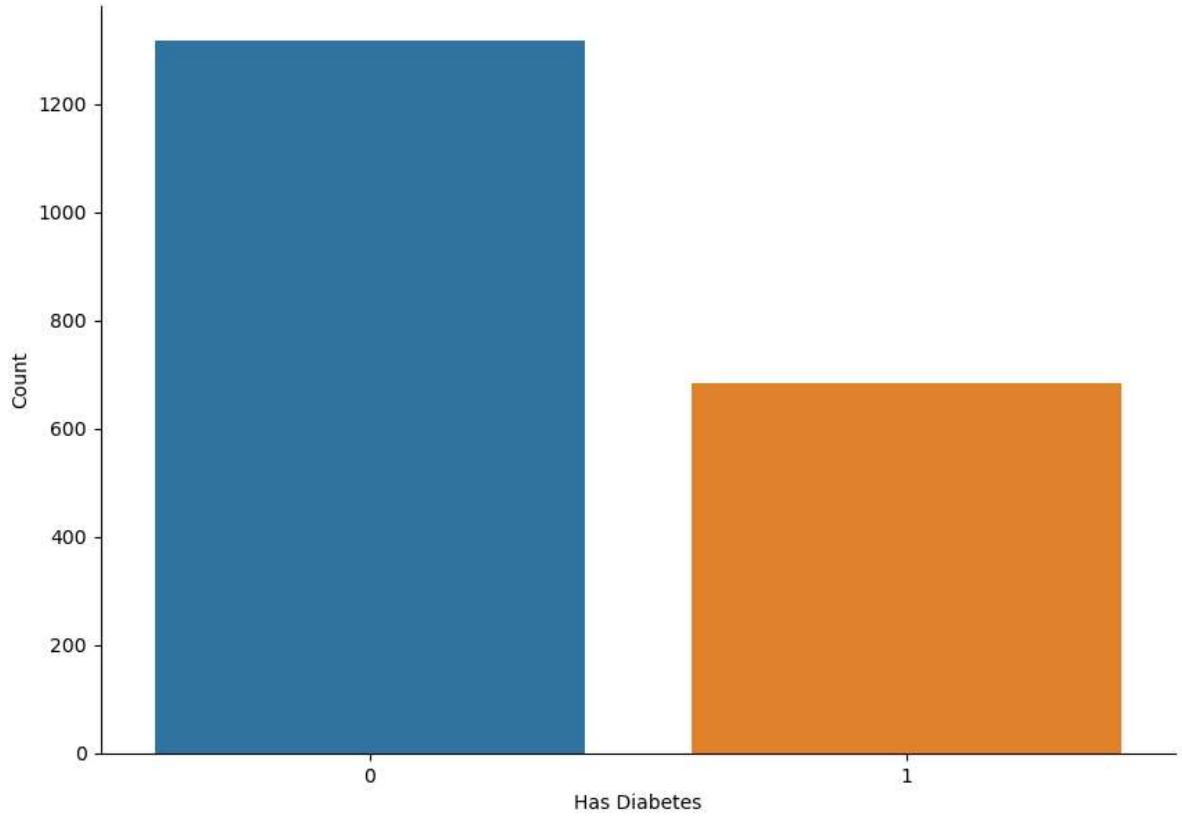
In [251]:
```
diabetes.head()
```

Out[251]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DPF	Age	Outcome
0	2	138	62	35	0	33.6	0.127	47	1
1	0	84	82	31	125	38.2	0.233	23	0
2	0	145	0	0	0	44.2	0.630	31	1
3	0	135	68	42	250	42.3	0.365	24	1
4	1	139	62	41	480	40.7	0.536	21	0

In [252]:
```
# Importing libraries for visualization
import seaborn as sns
```

In [253]:
```
# Plotting the Outcomes based on the number of dataset entries
plt.figure(figsize=(10,7))
sns.countplot(x='Outcome', data=diabetes)

# Removing the unwanted spines
plt.gca().spines['top'].set_visible(False)
plt.gca().spines['right'].set_visible(False)

# Headings
plt.xlabel('Has Diabetes')
plt.ylabel('Count')

plt.show()
```

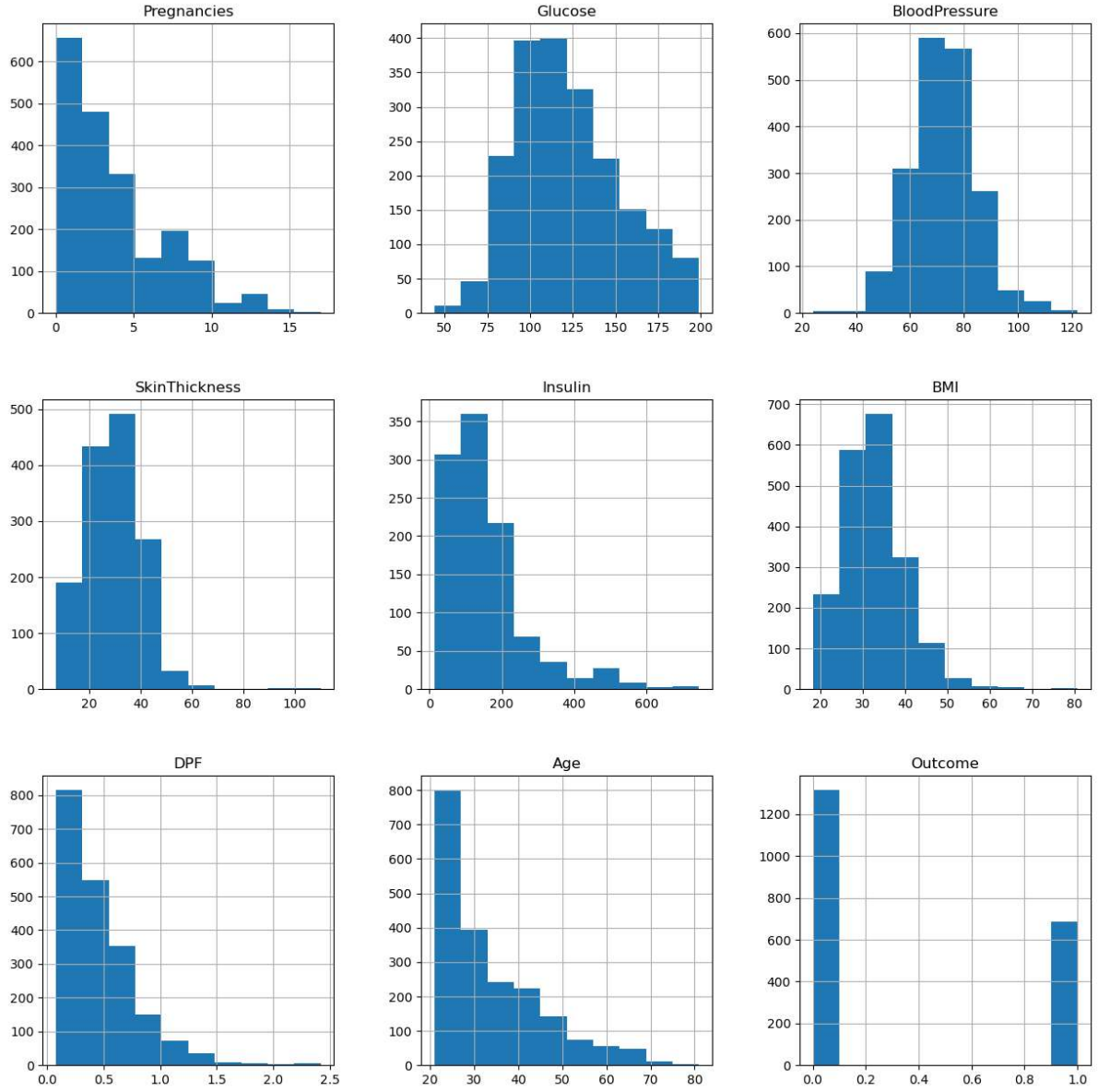


Data Cleaning

In [254]:
```
# Replacing 0 values from ['Glucose','BloodPressure','SkinThickness','Insulin',
diabetes_copy = diabetes.copy(deep=True)
diabetes_copy[['Glucose','BloodPressure','SkinThickness','Insulin','BMI']] = di
                                                                              'Sk
diabetes_copy.isnull().sum()
```

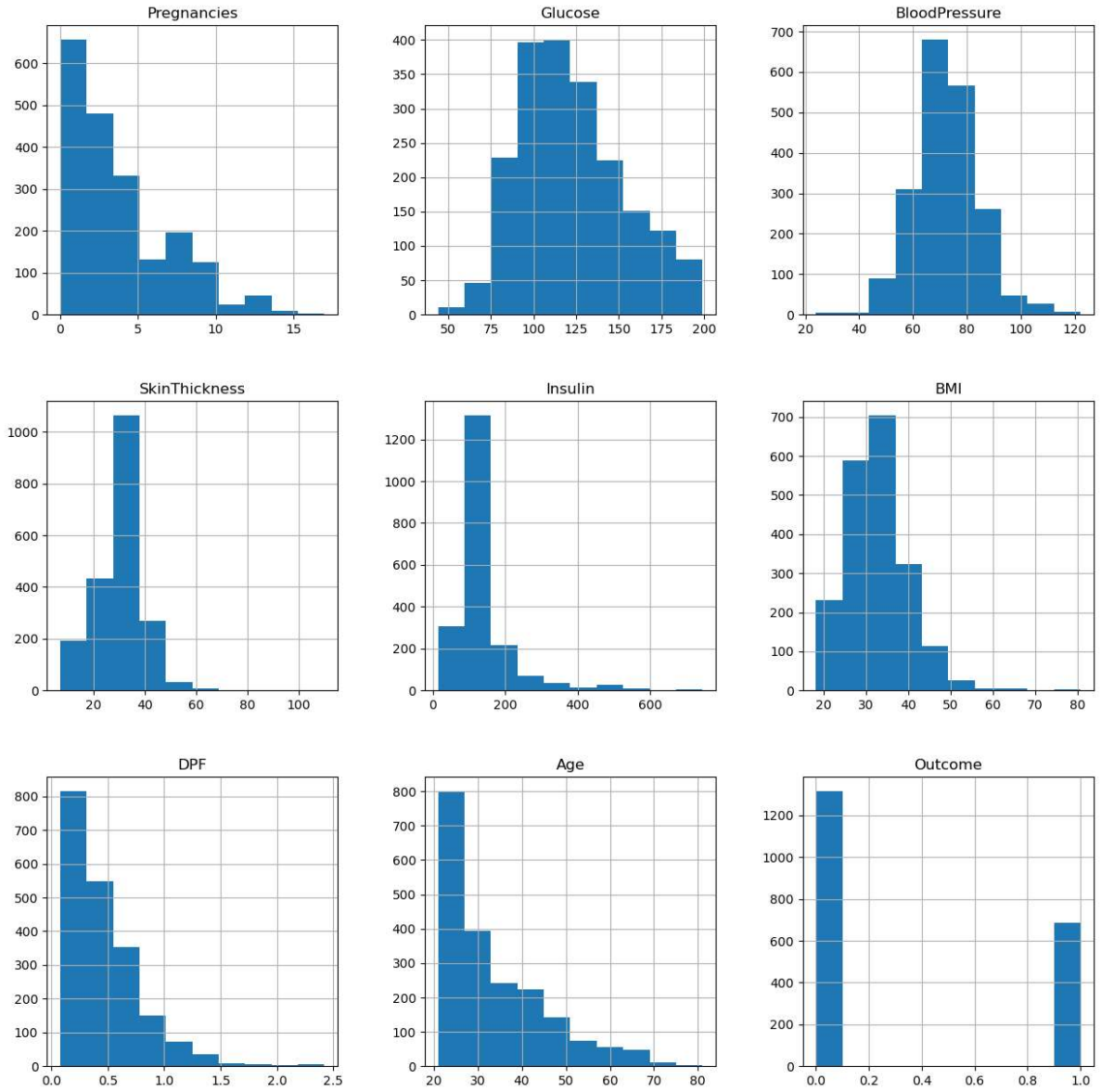
Out[254]:
```
Pregnancies         0
Glucose            13
BloodPressure      90
SkinThickness     573
Insulin           956
BMI                28
DPF                 0
Age                 0
Outcome             0
dtype: int64
```

In [255]:
```
# To fill these Nan values the data distribution needs to be understood
# Plotting histogram of dataset before replacing NaN values
p = diabetes_copy.hist(figsize = (15,15))
```



In [256]:
```
# Imputing missing values in 'Glucose', 'BloodPressure', 'SkinThickness', 'Insu
diabetes_copy['Glucose'].fillna(diabetes_copy['Glucose'].mean(), inplace=True)
diabetes_copy['BloodPressure'].fillna(diabetes_copy['BloodPressure'].mean(), in
diabetes_copy['SkinThickness'].fillna(diabetes_copy['SkinThickness'].median(),
diabetes_copy['Insulin'].fillna(diabetes_copy['Insulin'].median(), inplace=True
diabetes_copy['BMI'].fillna(diabetes_copy['BMI'].median(), inplace=True)
```

In [257]:
```
# Plotting histogram of dataset after replacing NaN values
p = diabetes_copy.hist(figsize=(15,15))
```



In [258]:
```
diabetes_copy.isnull().sum()
```

Out[258]:
```
Pregnancies       0
Glucose           0
BloodPressure     0
SkinThickness     0
Insulin           0
BMI               0
DPF               0
Age               0
Outcome           0
dtype: int64
```

Model Building

In [259]:
```
from sklearn.model_selection import train_test_split
```

In [260]:
```
X = diabetes.drop(columns='Outcome')
y = diabetes['Outcome']
```

In [261]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, rando
print('X_train size: {}, X_test size: {}'.format(X_train.shape, X_test.shape))
```

```
X_train size: (1600, 8), X_test size: (400, 8)
```

In [262]:
```
# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

In [263]:
```
# Using GridSearchCV to find the best algorithm for this problem
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import ShuffleSplit
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
```

In [264]:
```
# Creating a function to calculate the best model for this problem
def find_best_model(X, y):
    models = {
        'logistic_regression': {
            'model': LogisticRegression(solver='lbfgs', multi_class='auto'),
            'parameters': {
                'C': [1, 5, 10]
              }
        },

        'decision_tree': {
            'model': DecisionTreeClassifier(splitter='best'),
            'parameters': {
                'criterion': ['gini', 'entropy'],
                'max_depth': [5, 10]
            }
        },

        'random_forest': {
            'model': RandomForestClassifier(criterion='gini'),
            'parameters': {
                'n_estimators': [10, 15, 20, 50, 100, 200]
            }
        },

        'svm': {
            'model': SVC(gamma='auto'),
            'parameters': {
                'C': [1, 10, 20],
                'kernel': ['rbf', 'linear']
            }
        }
    }

    scores = []
    cv_shuffle = ShuffleSplit(n_splits=5, test_size=0.20, random_state=0)

    for model_name, model_params in models.items():
        gs = GridSearchCV(model_params['model'], model_params['parameters'], cv
        gs.fit(X, y)
        scores.append({
            'model': model_name,
            'best_parameters': gs.best_params_,
            'score': gs.best_score_
        })

    return pd.DataFrame(scores, columns=['model', 'best_parameters', 'score'])

# Running the function with the training data
find_best_model(X_train, y_train)
```

Out[264]:

	model	best_parameters	score
0	logistic_regression	{'C': 10}	0.763125
1	decision_tree	{'criterion': 'gini', 'max_depth': 10}	0.901250
2	random_forest	{'n_estimators': 200}	0.950625
3	svm	{'C': 20, 'kernel': 'rbf'}	0.869375

In [265]:
```
# Using cross_val_score for gaining average accuracy
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestClassifier
```

In [266]:
```
# Creating a Random Forest classifier
classifier = RandomForestClassifier(n_estimators=20, random_state=0)
classifier.fit(X_train, y_train)
```

Out[266]:
```
RandomForestClassifier(n_estimators=20, random_state=0)
```

In [267]:
```
# Performing cross-validation
scores = cross_val_score(classifier, X_train, y_train, cv=5)
```

In [268]:
```
# Calculating and printing the average accuracy
average_accuracy = round(sum(scores) * 100 / len(scores), 3)
print('Average Accuracy: {}%'.format(average_accuracy))
```

```
Average Accuracy: 95.0%
```

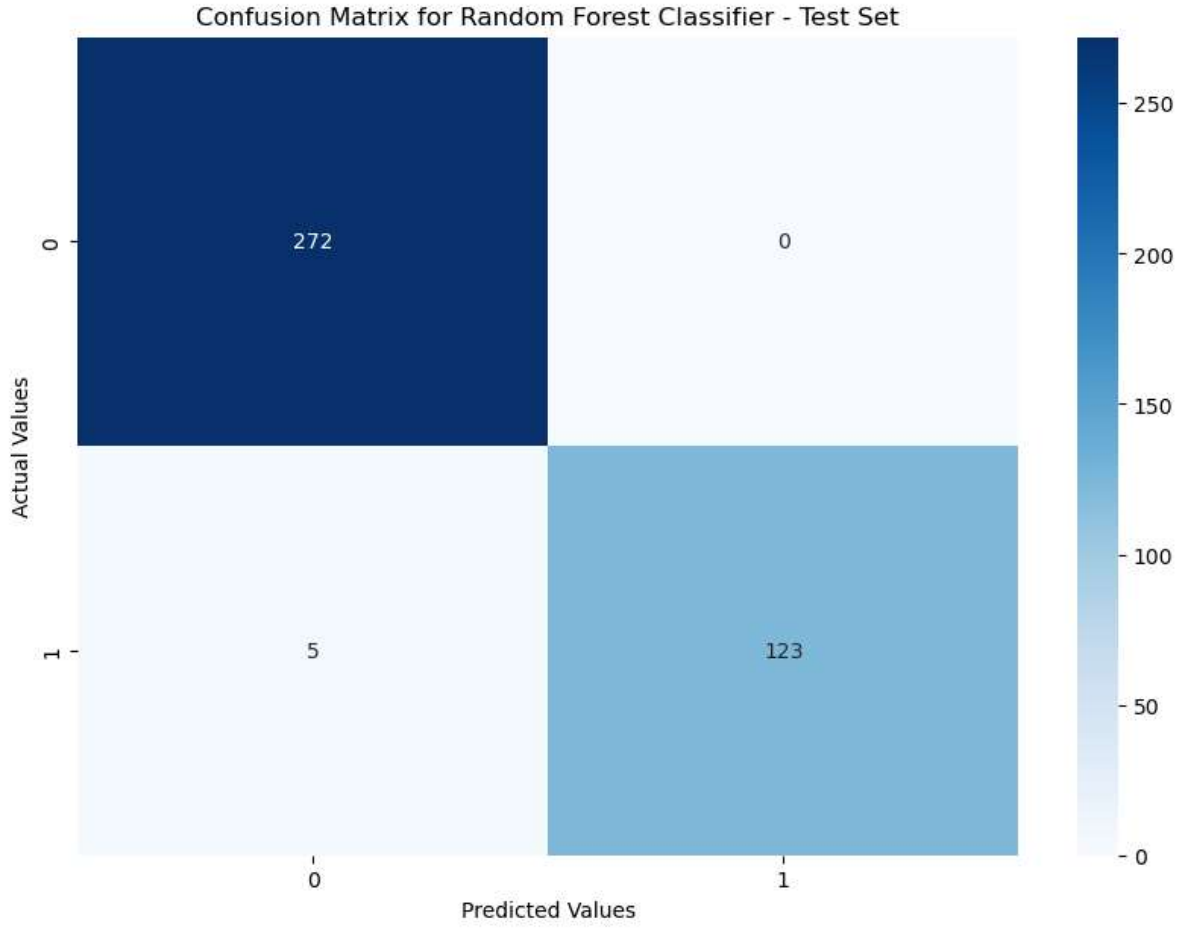
Model Evaluation

Confusion Matrix

In [269]:
```
from sklearn.metrics import confusion_matrix

# Assuming you have y_test and y_pred defined
y_pred = classifier.predict(X_test)
cm = confusion_matrix(y_test, y_pred)

# Display the confusion matrix
print("Confusion Matrix:")
print(cm)
```

```
Confusion Matrix:
[[272   0]
 [  5 123]]
```

In [270]:
```
# Visualizing the Confusion Matrix

plt.figure(figsize=(10, 7))
sns.heatmap(cm, annot=True, cmap="Blues", fmt='g')
plt.title('Confusion Matrix for Random Forest Classifier - Test Set')
plt.xlabel('Predicted Values')
plt.ylabel('Actual Values')
plt.show()
```



Accuracy Score

In [271]:
```
from sklearn.metrics import accuracy_score

# Assuming you have y_test and y_pred defined
y_pred = classifier.predict(X_test)

# Calculate accuracy and round to four decimal places
accuracy = round(accuracy_score(y_test, y_pred), 4) * 100

# Print the accuracy on the test set
print("Accuracy on the test set: {}%".format(accuracy))
```

```
Accuracy on the test set: 98.75%
```

Classification Report for Test Set

In [272]:
```
from sklearn.metrics import classification_report

# Assuming you have y_test and y_pred defined
y_pred = classifier.predict(X_test)

# Print the classification report for the test set
print("Classification Report for Test Set:")
print(classification_report(y_test, y_pred))
```

```
Classification Report for Test Set:
              precision    recall  f1-score   support

           0       0.98      1.00      0.99       272
           1       1.00      0.96      0.98       128

    accuracy                           0.99       400
   macro avg       0.99      0.98      0.99       400
weighted avg       0.99      0.99      0.99       400
```

Creating a Confusion Matrix for Training Set

In [273]:
```
# Assuming you have y_train and y_train_pred defined
y_train_pred = classifier.predict(X_train)
cm = confusion_matrix(y_train, y_train_pred)

# Display the confusion matrix for the training set
print("Confusion Matrix for Training Set:")
print(cm)
```

```
Confusion Matrix for Training Set:
[[1044    0]
 [   1  555]]
```

Accuracy Report

In [274]:
```
# Assuming you have y_train and y_train_pred defined
y_train_pred = classifier.predict(X_train)

# Calculate accuracy and round to four decimal places
accuracy_train = round(accuracy_score(y_train, y_train_pred), 4) * 100

# Print the accuracy on the training set
print("Accuracy on the training set: {}%".format(accuracy_train))
```

```
Accuracy on the training set: 99.94%
```

Classification Report for Training Set

In [275]:
```
# Assuming you have y_train and y_train_pred defined
y_train_pred = classifier.predict(X_train)

# Print the classification report for the training set
print("Classification Report for Training Set:")
print(classification_report(y_train, y_train_pred))
```

```
Classification Report for Training Set:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      1044
           1       1.00      1.00      1.00       556

    accuracy                           1.00      1600
   macro avg       1.00      1.00      1.00      1600
weighted avg       1.00      1.00      1.00      1600
```

Predictions

In [276]:
```
def predict_diabetes(Pregnancies, Glucose, BloodPressure, SkinThickness, Insuli
    preg = int(Pregnancies)
    glucose = float(Glucose)
    bp = float(BloodPressure)
    st = float(SkinThickness)
    insulin = float(Insulin)
    bmi = float(BMI)
    dpf = float(DPF)
    age = int(Age)

    x = [[preg, glucose, bp, st, insulin, bmi, dpf, age]]
    x = sc.transform(x)

    return classifier.predict(x)
```

Prediction

In [280]:
```
prediction = predict_diabetes(2, 81, 72, 15, 76, 30.1, 0.547, 25)
if prediction:
    print('Oh snap! Your body decided to host a sweet party. You have diabetes.
else:
    print("Well done! Your pancreas deserves a gold star. No diabetes for you!"
```

```
Well done! Your pancreas deserves a gold star. No diabetes for you!

C:\Users\gouth\anaconda3\lib\site-packages\sklearn\base.py:450: UserWarning: X
does not have valid feature names, but StandardScaler was fitted with feature
names
  warnings.warn(
```

In [281]:
```
prediction = predict_diabetes(1, 117, 88, 24, 145, 34.5, 0.403, 40)
if prediction:
    print('Oh snap! Your body decided to host a sweet party. You have diabetes.
else:
    print("Well done! Your pancreas deserves a gold star. No diabetes for you!"
```

```
Oh snap! Your body decided to host a sweet party. You have diabetes.

C:\Users\gouth\anaconda3\lib\site-packages\sklearn\base.py:450: UserWarning: X
does not have valid feature names, but StandardScaler was fitted with feature
names
  warnings.warn(
```

In [ ]: