Hi there. This is the reply report to my task from Nucleic Health. In order to predict periodical attributes like health or longevity of a bridge, we need to:

1)Analyze historical data 2)Explore data and attribute association (Statistical Analysis, as well as preprocessing) 3)Build a Machine learning model (This predict the next value, example what would be the Bridge's ductility after 3 months) 4)Save best model

In [5]:

```
# 1) Importing Diabetes data set and necessary libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
%matplotlib inline
```

In [6]:

```
sns.set()
```

In [7]:

```
#importing dataset
df= pd.read_csv(r'C:\Users\felix\Downloads\diabetes.csv')
```

In [8]:

```
#showing first 5 attributes of data
df.head()
```

Out[8]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunct
0	6	148	72	35	0	33.6	0.6
1	1	85	66	29	0	26.6	0.3
2	8	183	64	0	0	23.3	0.6
3	1	89	66	23	94	28.1	0.1
4	0	137	40	35	168	43.1	2.2

In [9]:

```
# 2) Exploring Data and Attribute Association
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
```

In [10]:

```
# to check the column names
df.columns
```

Out[10]:

```
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insuli
n',
       'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
      dtype='object')
```

In [11]:

```
#to check null values
df.isnull()
```

Out[11]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFu
0	False	False	False	False	False	False	
1	False	False	False	False	False	False	
2	False	False	False	False	False	False	
3	False	False	False	False	False	False	
4	False	False	False	False	False	False	
...	...	...	...	...	...	...	
763	False	False	False	False	False	False	
764	False	False	False	False	False	False	
765	False	False	False	False	False	False	
766	False	False	False	False	False	False	
767	False	False	False	False	False	False	

768 rows × 9 columns

In [12]:

```
#null values not means0, but value inputs like NIL, NAN are also counted. To displat thos
df_copy = df.copy(deep = True)
df_copy[['Glucose','BloodPressure','SkinThickness','Insulin','BMI']] = df_copy[['Glucose'
```

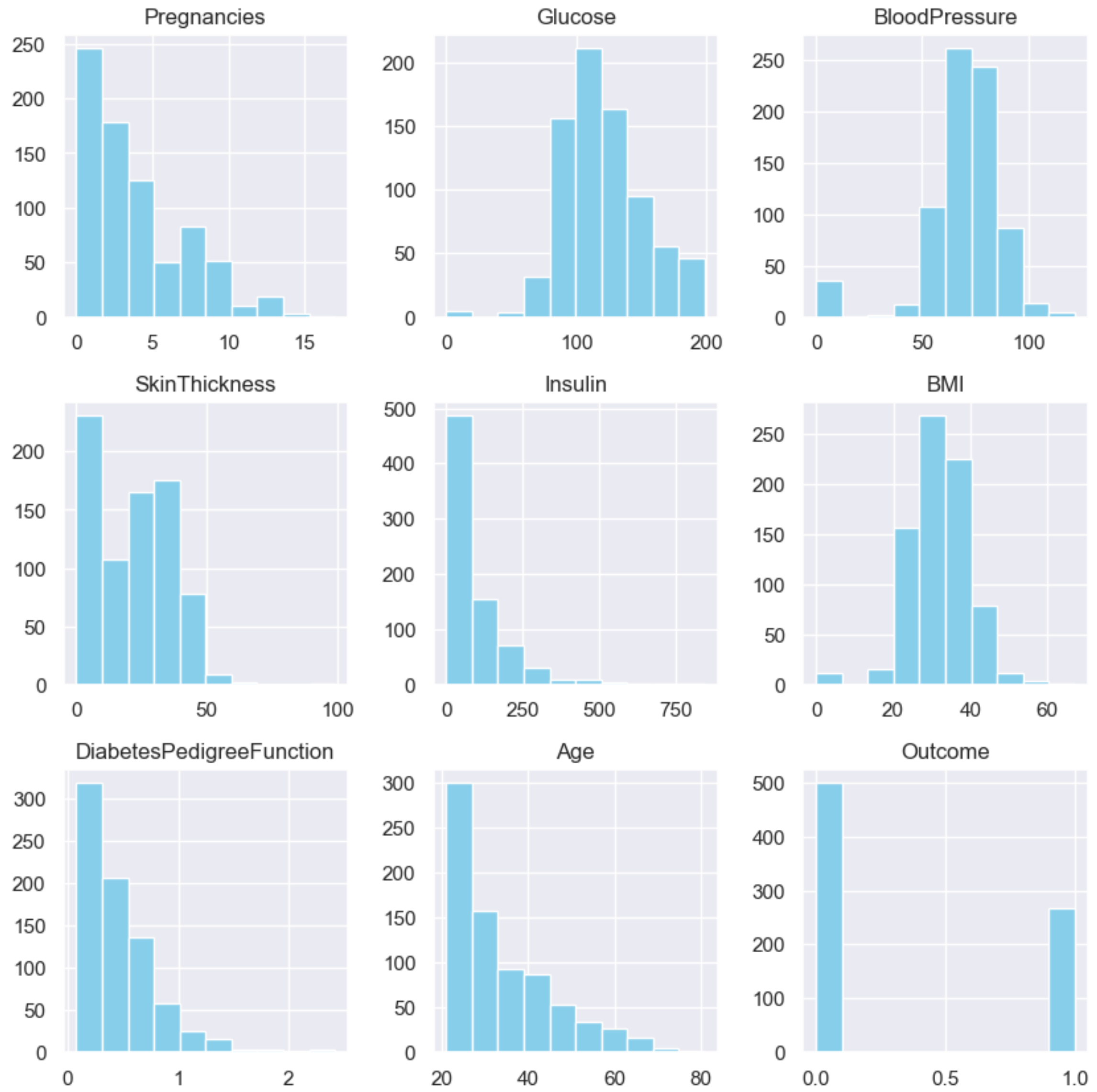
In [13]:

```
print(df_copy.isnull().sum())
```

```
Pregnancies                   0
Glucose                       5
BloodPressure                35
SkinThickness               227
Insulin                     374
BMI                          11
DiabetesPedigreeFunction      0
Age                           0
Outcome                       0
dtype: int64
```

In [14]:

```
#plotting a histogram to infer details about value pattern
h = df.hist(figsize=(10,10), color='skyblue')
```

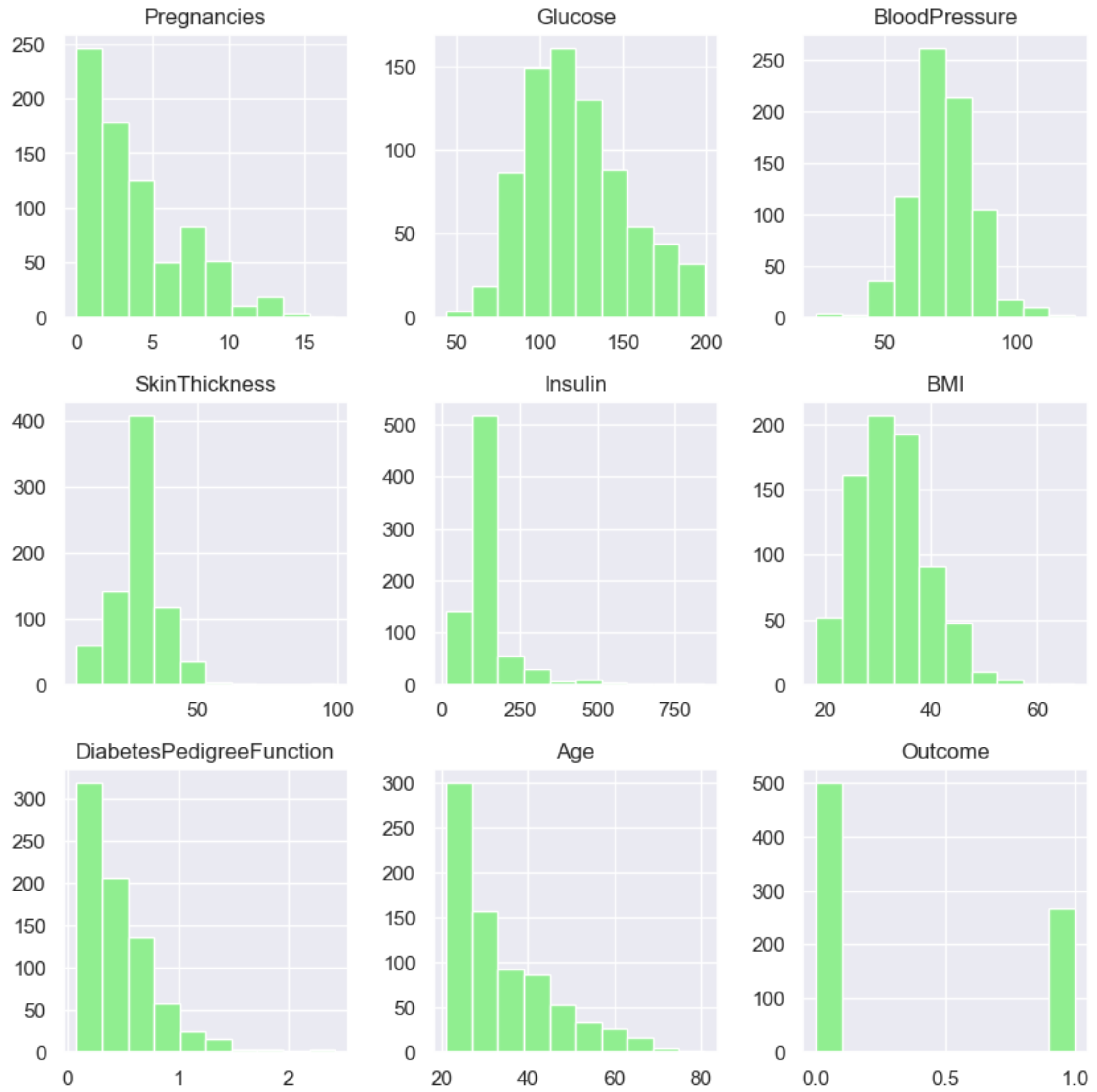


In [15]:

```
#replacing null value with mean values of each column respectively
df_copy['Glucose'].fillna(df_copy['Glucose'].mean(), inplace = True)
df_copy['BloodPressure'].fillna(df_copy['BloodPressure'].mean(), inplace = True)
df_copy['SkinThickness'].fillna(df_copy['SkinThickness'].median(), inplace = True)
df_copy['Insulin'].fillna(df_copy['Insulin'].median(), inplace = True)
df_copy['BMI'].fillna(df_copy['BMI'].median(), inplace = True)
```

In [16]:

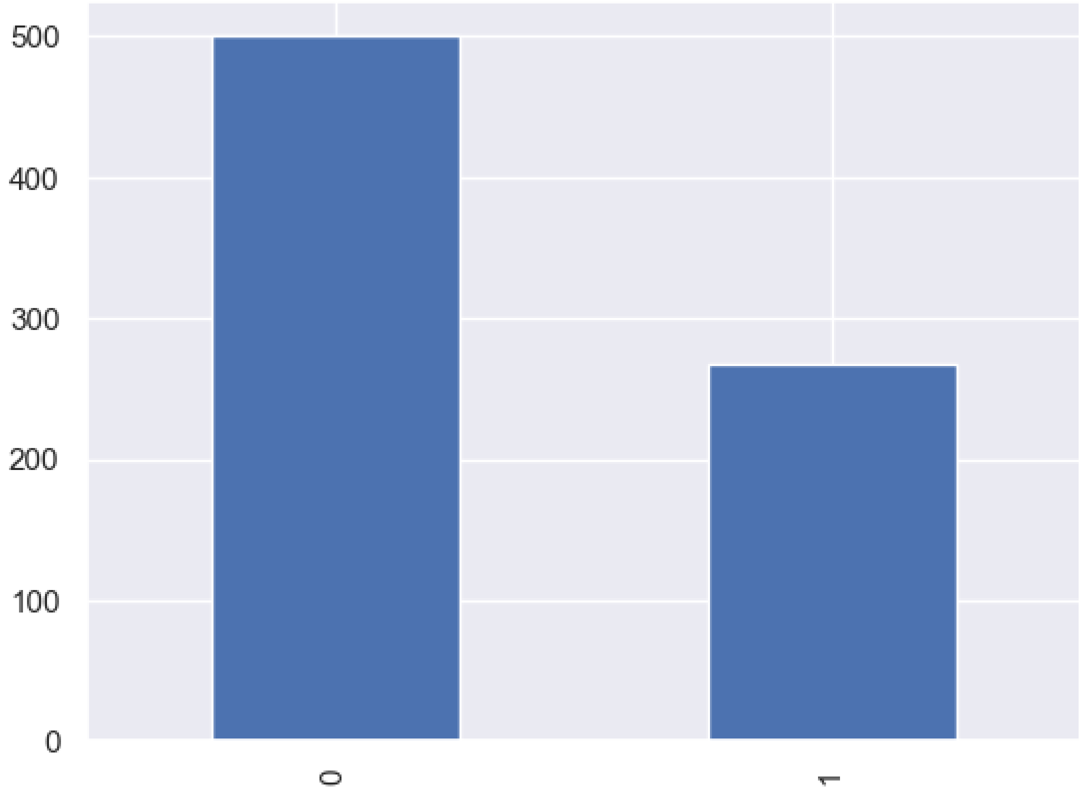
```
#plotting a histogram after removing null values
h = df_copy.hist(figsize=(10,10),color="lightgreen")
```



In [17]:

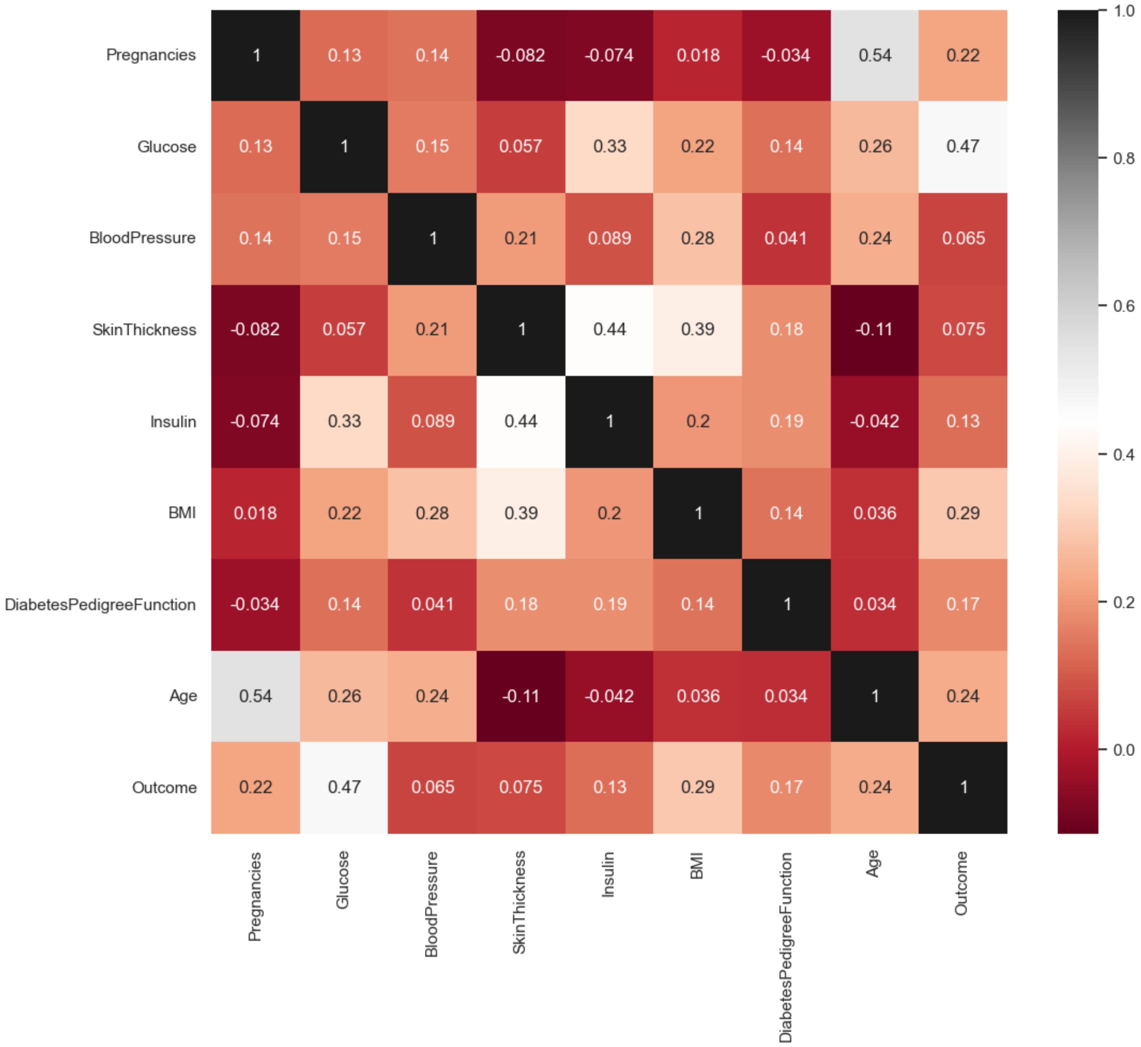
```
#Representing the outcome column balance (0 is non diabetic and 1 is diabetic patients)
color_wheel = {1: " ", 2: " "}
colors = df["Outcome"].map(lambda x: color_wheel.get(x + 1))
print(df.Outcome.value_counts())
p=df.Outcome.value_counts().plot(kind="bar")
```

```
0    500
1    268
Name: Outcome, dtype: int64
```



In [18]:

```
#As per above diagram, the diabetic patients are half as count of total patients. Next st
plt.figure(figsize=(12,10))
# calling seaborn showcase heatmap
p = sns.heatmap(df.corr(), annot=True,cmap ='RdGy')
```



	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
Pregnancies	1	0.13	0.14	-0.082	-0.074	0.018	-0.034	0.54	0.22
Glucose	0.13	1	0.15	0.057	0.33	0.22	0.14	0.26	0.47
BloodPressure	0.14	0.15	1	0.21	0.089	0.28	0.041	0.24	0.065
SkinThickness	-0.082	0.057	0.21	1	0.44	0.39	0.18	-0.11	0.075
Insulin	-0.074	0.33	0.089	0.44	1	0.2	0.19	-0.042	0.13
BMI	0.018	0.22	0.28	0.39	0.2	1	0.14	0.036	0.29
DiabetesPedigreeFunction	-0.034	0.14	0.041	0.18	0.19	0.14	1	0.034	0.17
Age	0.54	0.26	0.24	-0.11	-0.042	0.036	0.034	1	0.24
Outcome	0.22	0.47	0.065	0.075	0.13	0.29	0.17	0.24	1

In [19]:

```
#From the heatmap, we can associate values, for example, insulin in correlated with gluco
#Next step is scaling the data scaling, done to generalize the differences in data points
#Importing Scaling and pre-models required for ML process.
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import classification_report
```

In [20]:

```
sc_X = StandardScaler()
X =  pd.DataFrame(sc_X.fit_transform(df_copy.drop(["Outcome"],axis = 1),), columns=['Preg
'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI', 'DiabetesPedigreeFunction'

X.head()
```

Out[20]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedig
0	0.639947	0.865108	-0.033518	0.670643	-0.181541	0.166619	
1	-0.844885	-1.206162	-0.529859	-0.012301	-0.181541	-0.852200	
2	1.233880	2.015813	-0.695306	-0.012301	-0.181541	-1.332500	
3	-0.844885	-1.074652	-0.529859	-0.695245	-0.540642	-0.633881	
4	-1.141852	0.503458	-2.680669	0.670643	0.316566	1.549303	

In [21]:

```
# 3)Model Building
X = df.drop('Outcome', axis=1)
y = df['Outcome']
```

In [22]:

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.33,
                                                    random_state=7)
```

In [23]:

```
# Model 1 - Random Forest
from sklearn.ensemble import RandomForestClassifier

rfc = RandomForestClassifier(n_estimators=200)
rfc.fit(X_train, y_train)
```

Out[23]:

```
▾         RandomForestClassifier
RandomForestClassifier(n_estimators=200)
```

In [24]:

```
#Checking the accuracy of model
rfc_train = rfc.predict(X_train)
from sklearn import metrics

print("Accuracy_Score =", format(metrics.accuracy_score(y_train, rfc_train)))
```

```
Accuracy_Score = 1.0
```

In [25]:

```
#Getting Accuracy score of random forest
from sklearn import metrics

predictions = rfc.predict(X_test)
print("Accuracy_Score =", format(metrics.accuracy_score(y_test, predictions)))
```

```
Accuracy_Score = 0.7637795275590551
```

In [26]:

```
#Classification report and Confusion matrix of Random Forest
from sklearn.metrics import classification_report, confusion_matrix
print(confusion_matrix(y_test,predictions))
print(classification_report(y_test,predictions))
```

```
[[135  27]
 [ 33  59]]
              precision    recall  f1-score   support

           0       0.80      0.83      0.82       162
           1       0.69      0.64      0.66        92

    accuracy                           0.76       254
   macro avg       0.74      0.74      0.74       254
weighted avg       0.76      0.76      0.76       254
```

In [27]:

```
# Model 2- DecisionTree
from sklearn.tree import DecisionTreeClassifier

dtree = DecisionTreeClassifier()
dtree.fit(X_train, y_train)
```

Out[27]:

```
▾ DecisionTreeClassifier
DecisionTreeClassifier()
```

In [28]:

```
# Accuracy score of Decision Tree
from sklearn import metrics

predictions = dtree.predict(X_test)
print("Accuracy Score =", format(metrics.accuracy_score(y_test,predictions)))
```

```
Accuracy Score = 0.6968503937007874
```

In [29]:

```
#Classification report and Confusion matrix of Decision Tree
from sklearn.metrics import classification_report, confusion_matrix

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

```
[[125  37]
 [ 40  52]]
              precision    recall  f1-score   support

           0       0.76      0.77      0.76       162
           1       0.58      0.57      0.57        92

    accuracy                           0.70       254
   macro avg       0.67      0.67      0.67       254
weighted avg       0.69      0.70      0.70       254
```

In [30]:

```
# Model 3- XgBoost classifier
!pip install xgboost
from xgboost import XGBClassifier

xgb_model = XGBClassifier(gamma=0)
xgb_model.fit(X_train, y_train)
```

```
Requirement already satisfied: xgboost in c:\users\felix\anaconda3\lib\sit
e-packages (1.7.6)
Requirement already satisfied: scipy in c:\users\felix\anaconda3\lib\site-
packages (from xgboost) (1.10.0)
Requirement already satisfied: numpy in c:\users\felix\anaconda3\lib\site-
packages (from xgboost) (1.23.5)
```

Out[30]:

```
▾                                XGBClassifier
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types
=None,
              gamma=0, gpu_id=None, grow_policy=None, importance_type=N
one,
              interaction_constraints=None, learning_rate=None, max_bin
=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
```

In [31]:

```
# Accuracy score of XgBoost classifier
from sklearn import metrics

xgb_pred = xgb_model.predict(X_test)
print("Accuracy Score =", format(metrics.accuracy_score(y_test, xgb_pred)))
```

```
Accuracy Score = 0.7401574803149606
```

In [32]:

```
#Classification report and Confusion matrix of XgBoost classifier
from sklearn.metrics import classification_report, confusion_matrix

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

```
[[131  31]
 [ 35  57]]
              precision    recall  f1-score   support

           0       0.79      0.81      0.80       162
           1       0.65      0.62      0.63        92

    accuracy                           0.74       254
   macro avg       0.72      0.71      0.72       254
weighted avg       0.74      0.74      0.74       254
```

In [33]:

```
# Model 4- support Vector Machine (SVM)
from sklearn.svm import SVC

svc_model = SVC()
svc_model.fit(X_train, y_train)
```

Out[33]:

```
▸ SVC
```

In [34]:

```
svc_pred = svc_model.predict(X_test)
```

In [35]:

```
#Accuracy score of SVM
from sklearn import metrics

print("Accuracy Score =", format(metrics.accuracy_score(y_test, svc_pred)))
```

```
Accuracy Score = 0.7480314960629921
```

In [36]:

```
from sklearn.metrics import classification_report, confusion_matrix

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

```
[[145  17]
 [ 47  45]]
              precision    recall  f1-score   support

           0       0.76      0.90      0.82       162
           1       0.73      0.49      0.58        92

    accuracy                           0.75       254
   macro avg       0.74      0.69      0.70       254
weighted avg       0.74      0.75      0.73       254
```

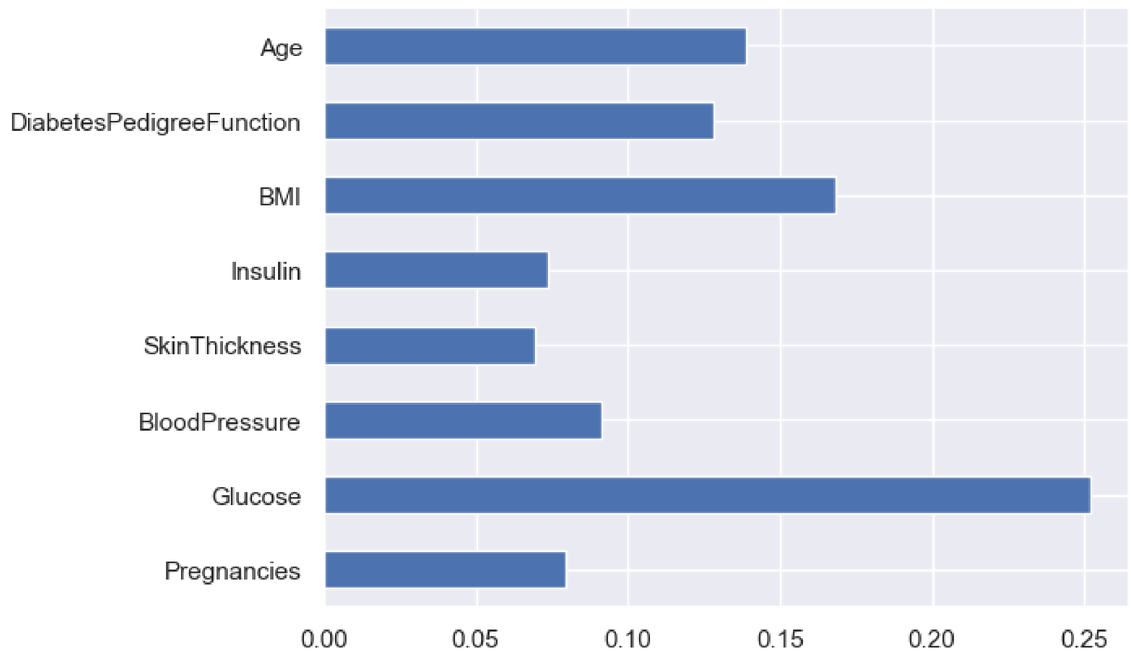
Based upom four models and their accuracy, Model 1 shows the highest accuracy, it is the 'Random Forest' machine learning model, with an accuracy of 0.75.

In [37]:

```
#finding feature importance to analyze which column feature holds most weightage in makin
(pd.Series(rfc.feature_importances_, index=X.columns).plot(kind='barh'))
```

Out[37]:

```
<Axes: >
```



It is visible from the above diagram that, Glucose holds highest weightage in this dataset.

In [38]:

```
# 4) Saving Model
import pickle
saved_model = pickle.dumps(rfc)
#loading the saved model
rfc_from_pickle = pickle.loads(saved_model)

# use this to make predictions
rfc_from_pickle.predict(X_test)
```

Out[38]:

```
array([0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0,
       0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1,
       0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
       1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
       0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1,
       0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
       0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0,
       1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0,
       0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1,
       1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1], dtype=int64)
```

In [45]:

```
#Begin prediction

df.head()
```

Out[45]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunct
0	6	148	72	35	0	33.6	0.6
1	1	85	66	29	0	26.6	0.3
2	8	183	64	0	0	23.3	0.6
3	1	89	66	23	94	28.1	0.1
4	0	137	40	35	168	43.1	2.2

In [53]:

```
df.tail()
```

Out[53]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFur
763	10	101	76	48	180	32.9	
764	2	122	70	27	0	36.8	
765	5	121	72	23	112	26.2	
766	1	126	60	0	0	30.1	
767	1	93	70	31	0	30.4	

In [ ]:

```
#Predicting the 3rd patient
print("Enter patient data one by one to test the prediction:")
while True:
    Pregnancies = int(input("Enter Pregnancies:"))
    glucose = int(input("Enter Glucose:"))
    bloodpressure = int(input("Enter Blood Pressue:"))
    skinthickness = int(input("Enter Skin Thickness:"))
    insulin = int(input("Enter Insulin:"))
    bmi = float(input("Enter BMI:"))
    DiabetesPedigreeFunction = float(input("Enter DiabetesPedigreeFunction:"))
    age = int(input("Enter Age:"))
    result = rfc.predict([[Pregnancies,glucose,bloodpressure,skinthickness,insulin,bmi,Di
    print("Current patient's chances of getting Diabetes is=",result)

    print("[0] means chances the patient to become Diabetic is 0 \n [1] means chances the
    continue
```

```
Enter BMI:23.3
Enter DiabetesPedigreeFunction:0.672
Enter Age:32
Current patient's chances of getting Diabetes is= [0]
[0] means chances the patient to become Diabetic is 0
 [1] means chances the patient to become Diabetic is 1 or maximum
Enter Pregnancies:1
Enter Glucose:126
Enter Blood Pressue:60
Enter Skin Thickness:0
Enter Insulin:0
Enter BMI:30.1
Enter DiabetesPedigreeFunction:0.349
Enter Age:47
Current patient's chances of getting Diabetes is= [1]
[0] means chances the patient to become Diabetic is 0
 [1] means chances the patient to become Diabetic is 1 or maximum

Enter Pregnancies:
```

```
#3 Patient
Enter Pregnancies:8
Enter Glucose:183
```

```
Enter Blood Pressue:64
Enter Skin Thickness:0
Enter Insulin:0
Enter BMI:23.3
Enter DiabetesPedigreeFunction:0.672
Enter Age:32
Current patient's chances of getting Diabetes is= [0]
[0] means chances the patient to become Diabetic is 0
 [1] means chances the patient to become Diabetic is 1 or maximum
```

3 patient Might not get affected by Diabetes.

```
#766 Patient
Enter Pregnancies:1
Enter Glucose:126
Enter Blood Pressue:60
Enter Skin Thickness:0
Enter Insulin:0
Enter BMI:30.1
Enter DiabetesPedigreeFunction:0.349
Enter Age:47
Current patient's chances of getting Diabetes is= [1]
[0] means chances the patient to become Diabetic is 0
 [1] means chances the patient to become Diabetic is 1 or maximum
```

766 patient Might get affected by Diabetes.

By Ajmal M S