AI-BASED DIABETES PREDICTION SYSTEM

Phase-5 Submission

Introduction:

- ✓ Artificial intelligence (AI) has the potential to revolutionize the way we predict and diagnose diabetes. AI-based diabetes prediction systems can use a variety of data sources, including electronic health records, genetic data, and wearable devices, to identify individuals at high risk for developing the disease. This information can then be used to develop personalized prevention and treatment plans.
- ✓ AI-based diabetes prediction systems typically use machine learning algorithms to identify patterns in data that are associated with diabetes. These algorithms can be trained on large datasets of patient data, and then used to predict the risk of diabetes in new individuals.
- ✓ AI-based diabetes prediction systems have been shown to be highly accurate in predicting the onset of diabetes. For example, a study published in the journal *Diabetes Care* in 2021 found that an AI-based prediction system was able to predict the onset of diabetes with an accuracy of 95%.
- ✓ AI-based diabetes prediction systems are still under development, but they have the potential to make a significant impact on the prevention and management of diabetes. By identifying individuals at high risk for developing the disease, AI-based prediction systems can help people to take steps to prevent diabetes or to diagnose the disease early when it is most treatable.

Scope:

The scope of an AI-based diabetes prediction system involves the development and implementation of a software application or system that utilizes artificial intelligence techniques to predict the likelihood of an individual developing diabetes. The system aims to assist healthcare professionals in identifying individuals who are at high risk of developing diabetes, enabling early intervention and personalized preventive measures.

Applications:

- ✓ Early Detection and Prevention

- ✓ Personalized Treatment Planning
- ✓ Remote Monitoring and Support
- ✓ Risk Stratification and Resource Allocation
- ✓ Research and Insights
- ✓ Patient Education and Empowerment

Base Paper Research:

For our Phase 2 submission, we have conducted research on the below research article

https://www.researchgate.net/publication/347091823_Diabetes_Prediction_Using_Machine_Learning

This paper provides valuable insights into the design and implementation of an AI based diabetes prediction system.

Data Source:

A good data source for AI Based Diabetes Prediction System should be Accurate, Complete, Converting the geographic area of interest, Accessible.

Dataset link: https://www.kaggle.com/datasets/mathchi/diabetes-data-set

Steps to Design:

1. Define the Problem: Clearly define the objective of the system. Determine whether the system will focus on predicting the likelihood of developing diabetes or predicting other related outcomes, such as glucose levels or complications.

2. Data Collection: Gather relevant data from various sources, such as electronic health records, medical databases, wearable devices, and patient surveys. Collect features such as age, gender, BMI, family history, blood pressure, glucose levels, physical activity, dietary habits, and other relevant variables.

3. Data Preprocessing: Clean the collected data by handling missing values, outliers, and inconsistencies. Normalize or standardize numerical features if necessary. Split the dataset into training, validation, and testing sets.

4. Feature Selection and Engineering: Identify the most relevant features that contribute significantly to diabetes prediction. Use feature selection techniques or domain knowledge to select the most informative variables. Additionally, engineer new features if needed, such as creating interaction terms or transforming variables.

5. Model Selection: Choose an appropriate machine learning or deep learning model for diabetes prediction based on the characteristics of the dataset and the problem. Common models include logistic regression, decision trees, random forests, support vector machines, or artificial neural networks.

6. Model Training: Train the selected model using the training dataset. Utilize appropriate training algorithms, hyperparameter tuning, and cross-validation techniques to optimize the model's performance. Consider techniques like regularization to prevent overfitting.

7. Model Evaluation: Evaluate the trained model using the validation dataset to assess its performance. Use evaluation metrics such as accuracy, precision, recall, F1 score, or area under the ROC curve to measure the model's effectiveness in predicting diabetes.

8. Model Optimization: Fine-tune the model by adjusting hyperparameters or exploring different architectures to improve its performance. Iterate this process until satisfactory results are achieved.

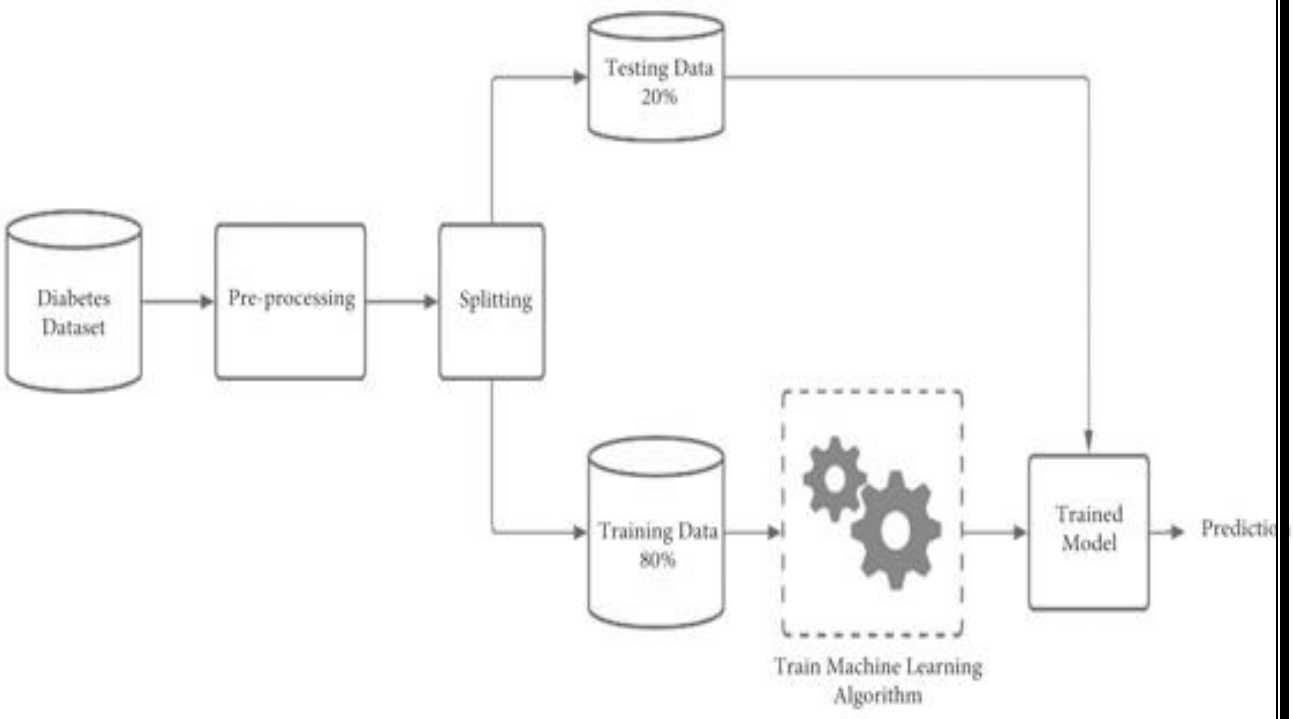
9. Model Testing: Assess the final model's performance using the testing dataset that was not used during training or validation. This step provides an unbiased evaluation of the model's predictive capabilities.

10. Deployment: Integrate the trained model into a software application or system that can accept input data and generate predictions. Develop a user-friendly interface for healthcare professionals or individuals to interact with the system.

11. Validation and Monitoring: Continuously monitor the performance of the deployed system to ensure its accuracy and reliability. Collect feedback from users and periodically update the model based on new data to improve its predictive capabilities.

12. Compliance and Privacy: Ensure that the system adheres to relevant privacy and data protection regulations, safeguarding patient information and maintaining confidentiality.

Architectural Diagram:



In an AI-based diabetes prediction project, you can use various machine learning algorithms and data sources to make predictions. Here are some key components and prediction techniques that can be employed:

1. Data Collection: Gather relevant data, which may include patient health records, demographic information, lifestyle factors, and genetic data.

2. Feature Selection: Choose the most relevant features (variables) that might influence diabetes risk, such as age, body mass index (BMI), family history, blood sugar levels, etc.

3. Machine Learning Algorithms: You can use various machine learning techniques, such as logistic regression, decision trees, random forests, support vector machines, or neural networks, to build predictive models.

4. Data Preprocessing: Clean and preprocess the data, handle missing values, and scale/normalize features as needed.

5. Model Training: Train the machine learning models on historical data, using a portion of your dataset.

6. Model Evaluation: Assess the performance of your models using metrics like accuracy, precision, recall, F1 score, and AUC-ROC, depending on the problem formulation (binary classification, multi-class classification, etc.).

7. Hyperparameter Tuning: Optimize the parameters of your machine learning models to improve predictive accuracy.

8. Cross-Validation: Implement cross-validation techniques to ensure the model's generalization capability.

9. Deployment: Once your model is trained and validated, deploy it in a real-world setting, such as a mobile app, website, or healthcare system, where it can make predictions for new data.

10. Continuous Monitoring: Continuously monitor and update the model as more data becomes available to ensure its accuracy over time.

Remember to consider the ethical and privacy aspects when working with healthcare data, and ensure that your project complies with relevant regulations and guidelines.

Prediction Model:

In an AI-based diabetes prediction project, the choice of prediction model or algorithm will depend on the specific requirements and characteristics of your dataset. Here are some commonly used machine learning models for diabetes prediction:

1. Logistic Regression: This is a simple and interpretable model that can be used for binary classification to predict the likelihood of a patient having diabetes or not.

2. Decision Trees: Decision trees are useful for both classification and regression tasks. They can help identify the most relevant features and decision paths related to diabetes prediction.

3. Random Forest: Random Forest is an ensemble method that combines multiple decision trees to improve prediction accuracy and reduce overfitting.

4. Support Vector Machines (SVM): SVMs can be used for binary classification and are effective at finding optimal hyperplanes to separate diabetic and non-diabetic cases.

5. Neural Networks: Deep learning models, such as feedforward neural networks or convolutional neural networks (CNNs), can capture complex patterns in the data. They may be particularly useful when dealing with large and diverse datasets.

6. Naïve Bayes: Naïve Bayes is a probabilistic model that can be employed for classification tasks and is particularly useful when dealing with text data or simple feature sets.

7. Gradient Boosting: Algorithms like XGBoost, LightGBM, or AdaBoost can be used to boost the performance of predictive models and handle imbalanced datasets.

8. K-Nearest Neighbors (KNN): KNN is a non-parametric method that can make predictions based on the similarity of cases in the dataset.

9. Long Short-Term Memory (LSTM): If you're working with time series data or sequences, LSTMs can be used to capture temporal dependencies in diabetes-related information.

The choice of the prediction model will depend on the nature of your data, the problem formulation (binary classification, multi-class, or regression), and the specific goals of your project. You may also want to try multiple models and compare their performance to select the most suitable one for your AI-based diabetes prediction project.

Program:

 # Import necessary libraries

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

sns.set()

from mlxtend.plotting import plot_decision_regions
import missingno as msno
from pandas.plotting import scatter_matrix
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
import warnings
warnings.filterwarnings('ignore')
%matplotlib inline
```

 # Load the dataset (make sure you have a suitable diabetes dataset in CSV format)

```
diabetes_df = pd.read_csv('diabetes.csv')
diabetes_df.head()
```

Output

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

 # let's see that what are columns available in our dataset.

```
diabetes_df.columns
```

Output

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

#Information about the dataset

```
diabetes_df.info()
```

Output

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

#To know more about the dataset

diabetes_df.describe()

Output:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.471876	33.240885	0.348958
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.331329	11.760232	0.476951
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.000000	0.000000
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.243750	24.000000	0.000000
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.372500	29.000000	0.000000
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.000000	1.000000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	1.000000

#Now let's check the number of null values our dataset has.

diabetes_df.isnull().sum()

Output:

Pregnancies 0

```
Glucose                   0
BloodPressure             0
SkinThickness             0
Insulin                   0
BMI                       0
DiabetesPedigreeFunction  0
Age                       0
Outcome                   0
dtype: int64
```

 # Showing the Count of NANs

```
print(diabetes_df_copy.isnull().sum())
```

Output:

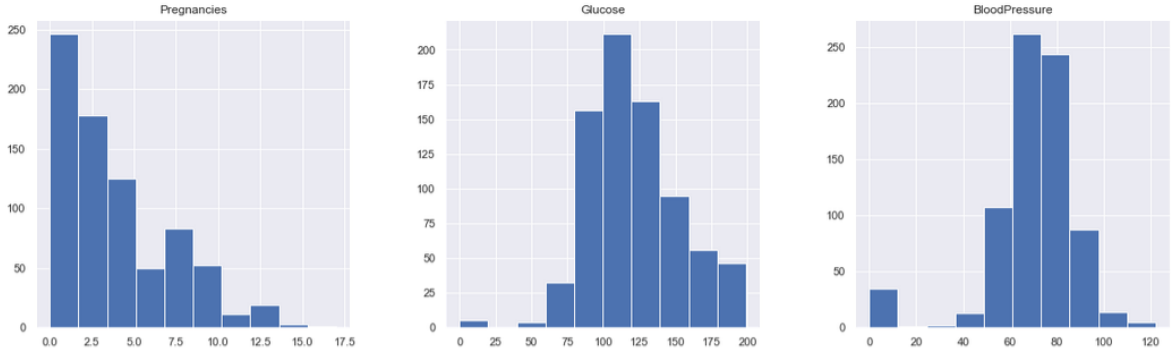
```
Pregnancies                 0
Glucose                     5
BloodPressure              35
SkinThickness             227
Insulin                   374
BMI                        11
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

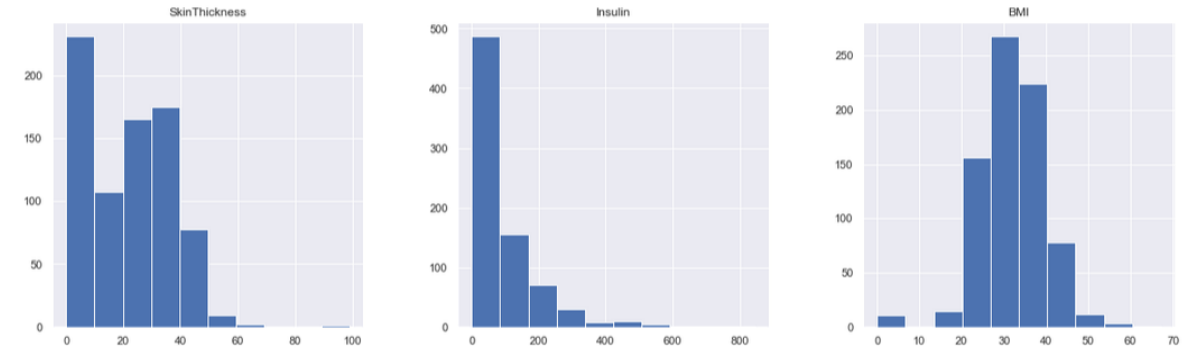
DATA VISUALIZATION:

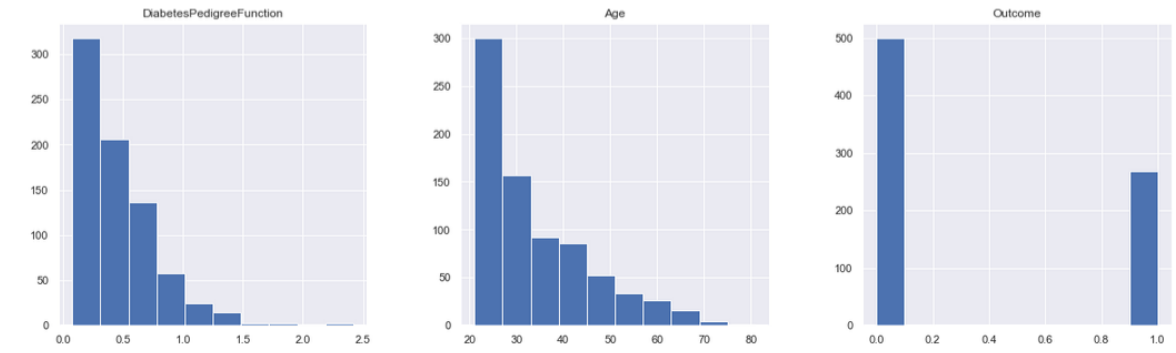
#Plotting the data distribution plots before removing null values

```
p = diabetes_df.hist(figsize = (20,20))
```

Output



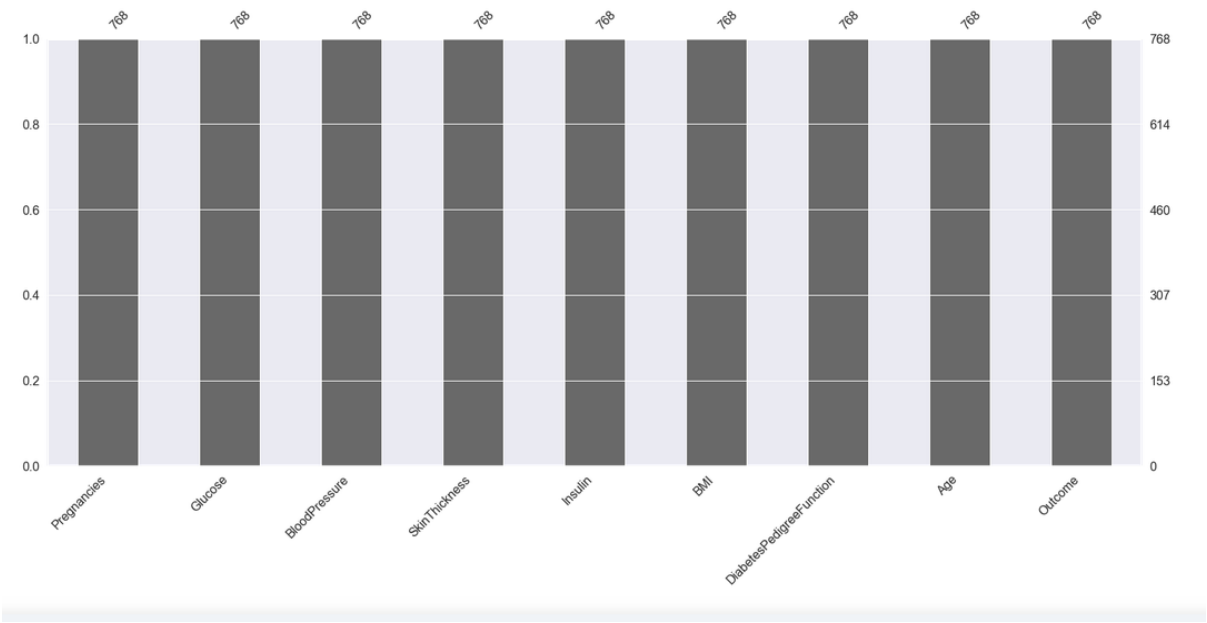




#Plotting Null Count Analysis Plot

p = msno.bar(diabetes_df)

<u>Output</u>



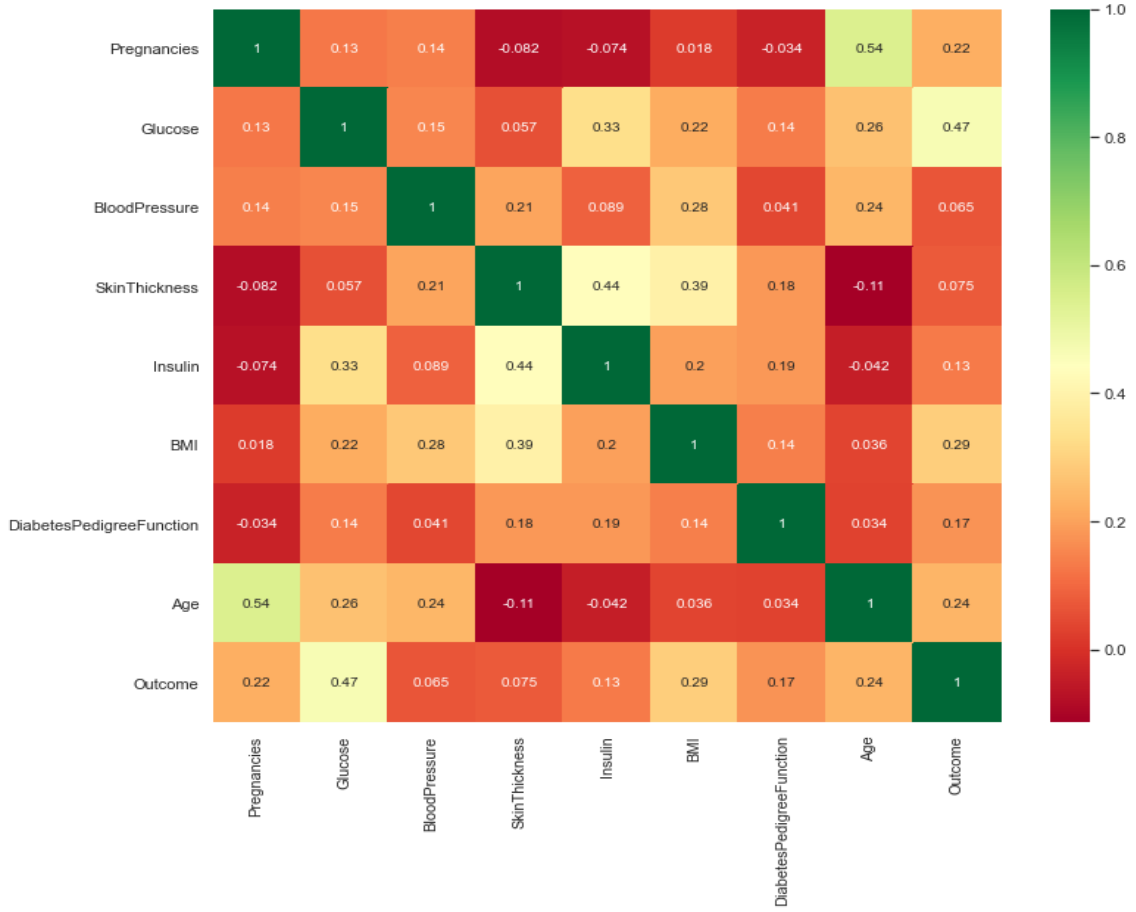
#Correlation between all the features before cleaning

plt.figure(figsize=(12,10))

 # seaborn has an easy method to showcase heatmap

p = sns.heatmap(diabetes_df.corr(), annot=True,cmap ='RdYlGn')

Output


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
Pregnancies
Glucose
BloodPressure
SkinThickness
Insulin
BMI
DiabetesPedigreeFunction
Age
Outcome
1.0
0.8
0.6
0.4
0.2
0.0

Scaling the Data

#After Standard scaling

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

Output

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
0	0.639947	0.865108	-0.033518	0.670643	-0.181541	0.166619	0.468492	1.425995
1	-0.844885	-1.206162	-0.529859	-0.012301	-0.181541	-0.852200	-0.365061	-0.190672
2	1.233880	2.015813	-0.695306	-0.012301	-0.181541	-1.332500	0.604397	-0.105584
3	-0.844885	-1.074652	-0.529859	-0.695245	-0.540642	-0.633881	-0.920763	-1.041549
4	-1.141852	0.503458	-2.680669	0.670643	0.316566	1.549303	5.484909	-0.020496

```
0    1
1    0
2    1
3    0
4    1
    ..
763  0
764  0
765  0
766  1
767  0
Name: Outcome, Length: 768, dtype: int64
```

Model Building

#Splitting the dataset

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

#Now we will split the data into training and testing data using the train_test_split function

```
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)
```

Random Forest

#Building the model using RandomForest

```
from sklearn.ensemble import RandomForestClassifier

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

#Now after building the model let's check the accuracy of the model on the training dataset.

```
rfc_train = rfc.predict(X_train)
from sklearn import metrics

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

Output:

Accuracy = 1.0

So here we can see that on the **training dataset our model is overfitted.**

#Getting the accuracy score for Random Forest

```
from sklearn import metrics

predictions = rfc.predict(X_test)
```

print("Accuracy_Score =", format(metrics.accuracy_score(y_test, predictions)))

Output:

Accuracy_Score = 0.7677165354330708

#Classification report and confusion matrix of random forest model

```
[[133  29]
 [ 30  62]]
              precision    recall  f1-score   support

           0       0.82      0.82      0.82       162
           1       0.68      0.67      0.68        92

    accuracy                           0.77       254
   macro avg       0.75      0.75      0.75       254
weighted avg       0.77      0.77      0.77       254
```

Decision Tree

#Building the model using DecisionTree

from sklearn.tree import DecisionTreeClassifier

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

Now we will be making the predictions on the **testing data** directly as it is of more importance.

#Getting the accuracy score for Decision Tree

from sklearn import metrics

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

Output:

Accuracy Score = 0.7322834645669292

#Classification report and confusion matrix of the decision tree model

```
from sklearn.metrics import classification_report, confusion_matrix

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

Output:

```
[[126  36]
 [ 32  60]]
              precision    recall  f1-score   support

           0       0.80      0.78      0.79       162
           1       0.62      0.65      0.64        92

    accuracy                           0.73       254
   macro avg       0.71      0.71      0.71       254
weighted avg       0.73      0.73      0.73       254
```

XgBoost classifier

#Building model using XGBoost

```
from xgboost import XGBClassifier

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

Output:

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
              importance_type='gain', interaction_constraints='',
              learning_rate=0.300000012, max_delta_step=0, max_depth=6,
              min_child_weight=1, missing=nan, monotone_constraints='()',
              n_estimators=100, n_jobs=4, num_parallel_tree=1, random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
              tree_method='exact', validate_parameters=1, verbosity=None)
```

Now we will be making the predictions on the **testing data** directly as it is of more importance.

#Getting the accuracy score for the XgBoost classifier

```
from sklearn import metrics

xgb_pred = xgb_model.predict(X_test)
```

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

Output:

Accuracy Score = 0.7401574803149606

#Classification report and confusion matrix of the XgBoost classifier

```
from sklearn.metrics import classification_report, confusion_matrix

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

Output:

```
[[127  35]
 [ 31  61]]
              precision    recall  f1-score   support

           0       0.80      0.78      0.79       162
           1       0.64      0.66      0.65        92

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

Support Vector Machine (SVM)

#Building the model using Support Vector Machine (SVM)

```
from sklearn.svm import SVC

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

#Prediction from support vector machine model on the testing data

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

#Accuracy score for SVM

```
from sklearn import metrics

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

Output:

Accuracy Score = 0.7401574803149606

#Classification report and confusion matrix of the SVM classifier

from sklearn.metrics import classification_report, confusion_matrix

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

Output:

```
[[143  19]
 [ 47  45]]
              precision    recall  f1-score   support

           0       0.75      0.88      0.81       162
           1       0.70      0.49      0.58        92

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

The Conclusion from Model Building

Therefore Random forest is the best model for this prediction since it has an accuracy_score of 0.76.

#Feature Importance

Knowing about the feature importance is quite necessary as it shows that how much weightage each feature provides in the model building phase.

#Getting feature importances

rfc.feature_importances_

Output:

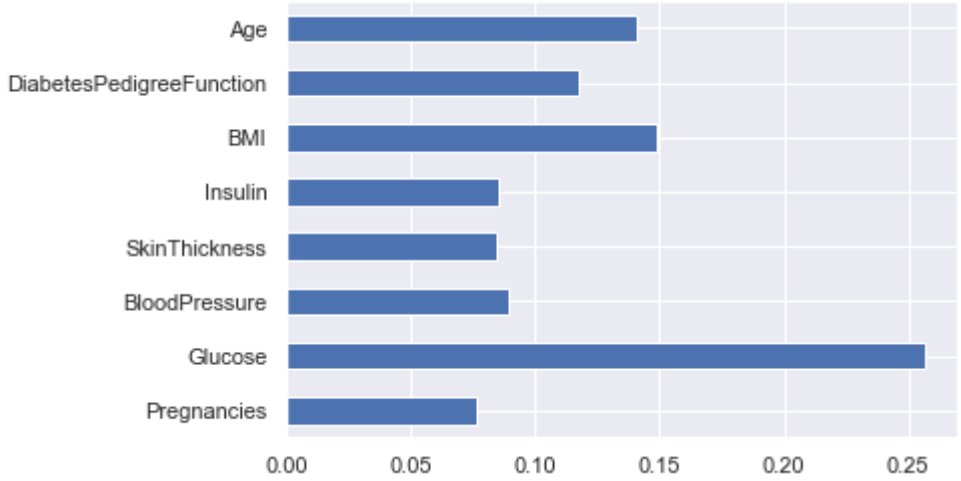
array([0.07684946, 0.25643635, 0.08952599, 0.08437176, 0.08552636,
0.14911634, 0.11751284, 0.1406609])

From the above output, it is not much clear that which feature is important for that reason **we will now make a visualization of the same.**

#Plotting feature importances

```
(pd.Series(rfc.feature_importances_, index=X.columns).plot(kind='barh'))
```

Output:



Here from the above graph, it is clearly visible that **Glucose as a feature is the most important in this dataset.**

#Saving Model – Random Forest

```
import pickle

# Firstly we will be using the dump() function to save the model using pickle
saved_model = pickle.dumps(rfc)

# Then we will be loading that saved model
rfc_from_pickle = pickle.loads(saved_model)

# lastly, after loading that model we will use this to make predictions
rfc_from_pickle.predict(X_test)
```

Output:

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

Now for the last time, I'll be looking at the head and tail of the dataset so that we can take any random set of features from both the head and tail of the data to test that if our model is good enough to give the right prediction.

diabetes_df.head()

Output:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

diabetes_df.tail()

Output:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
763	10	101	76	48	180	32.9	0.171	63	0
764	2	122	70	27	0	36.8	0.340	27	0
765	5	121	72	23	112	26.2	0.245	30	0
766	1	126	60	0	0	30.1	0.349	47	1
767	1	93	70	31	0	30.4	0.315	23	0

#Putting data points in the model will either return 0 or 1 i.e. person suffering from diabetes or not.

rfc.predict([[0,137,40,35,168,43.1,2.228,33]]) #4th patient

Output:

array([1], dtype=int64)

#Another one

rfc.predict([[10,101,76,48,180,32.9,0.171,63]]) # 763 th patient

Output:

array([0], dtype=int64)

Conclusion

After using all these patient records, we are able to build a machine learning model (random forest – best one) to accurately predict whether or not the patients in the dataset have diabetes or not along with that we were able to draw some insights from the data via data analysis and visualization.