PROJECT TITLE: AI BASED DIABETES PREDICTION SYSTEM

Phase 1: Project Documentation & Submission

PROBLEM STATEMENT:

The problem is to build an AI-powered diabetes prediction system that uses machine learning algorithms to analyze medical data and predict the likelihood of an individual developing diabetes. The system aims to provide early risk assessment and personalized preventive measures, allowing individuals to take proactive actions to manage their health.

OBJECTIVE:

The objective of the project to develop an AI-powered diabetes prediction system is to provide early risk assessment and personalized preventive measures for individuals, allowing them to take proactive actions to manage their health.

DATASET:

In this project, we use the dataset from **kaggle.**

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

- Pregnancies: Number of times pregnant
- Glucose: Plasma glucose concentration 2 hours in an oral glucose tolerance test
- Blood Pressure: Diastolic blood pressure (mm Hg)
- Skin Thickness: Triceps skin fold thickness (mm)
- Insulin: 2-Hour serum insulin (mu U/ml)
- BMI: Body mass index (weight in kg/ (height in m) ^2)
- Diabetes Pedigree Function: Diabetes pedigree function
- Age: Age (years)
- Outcome: Class variable (0 or 1)

DESIGN THINKING AND PHASES OF DEVELOPMENT:

Design thinking is an iterative and human-centered approach to problem-solving. In the context of building an AI-powered diabetes prediction system, here's a design thinking framework and phases of development that includes

key stages like data collection, data preprocessing, model selection, deployment and iterative improvement

1. ***Data Collection:***

Data collection is a critical step in building your AI-powered diabetes prediction system. You'll need a dataset that contains the necessary medical features and labels indicating whether individuals have diabetes or not. Here's how you can approach data collection:

- Identify Data Sources:

Collaborate with healthcare institutions, clinics, or research organizations to access medical data. Ensure that you comply with all legal and ethical requirements for data access and usage.

- Electronic Health Records (EHRs):

EHRs can be a valuable source of medical data. They typically contain patient information such as glucose levels, blood pressure, BMI, and other relevant medical features.

- Lifestyle and Demographic Data:

Collect information on lifestyle factors such as diet, exercise habits, smoking status, and family medical history. These factors can play a significant role in diabetes prediction.

- Lab Tests and Diagnostic Data:

Include results of lab tests, such as fasting blood glucose levels, HbA1c, and lipid profiles, which are commonly used for diabetes diagnosis and risk assessment.

- Data Privacy and Consent:

Ensure that you have proper consent and data anonymization procedures in place to protect patient privacy and comply with regulations like HIPAA (for the U.S.) or GDPR (for Europe).

2. ***Data Preprocessing:***

Data preprocessing is a critical step in preparing your medical data for training machine learning models. It involves cleaning the data, handling missing values, normalizing features, and performing other transformations to make the data suitable for modeling. Here's a step-by-step approach to data preprocessing:

- Data Cleaning:

Identify and handle missing values: Use techniques like imputation (filling missing values with meaningful estimates) or removal of rows/columns with excessive missing data.

Detect and address outliers: Outliers can adversely affect model performance. You can choose to remove outliers, transform them, or treat them separately.

Check for duplicate records and remove them if necessary.

- Feature Engineering:

Create new features that might provide valuable information. For example, you can calculate the body mass index (BMI) if it's not already present in the dataset.

Encode categorical variables: Convert categorical features into numerical representations using techniques like one-hot encoding or label encoding.

- Data Normalization:

Scale numerical features to have a consistent range. Common techniques include Min-Max scaling or Z-score standardization. This ensures that all features contribute equally to the model.

- Data Splitting:

Split the preprocessed data into training, validation, and test sets. The typical split ratio is 70-80% for training, 10-15% for validation, and 10-15% for testing. Ensure that the class distribution is maintained in each split.

- Feature Scaling:

Scale numerical features to have a consistent range. Common techniques include Min-Max scaling or Z-score standardization. This ensures that all features contribute equally to the model.

3. Model Selection:

Model selection is a critical step in building your diabetes risk prediction system. Experimenting with various machine learning algorithms can help you identify the model that performs best for your specific problem. Here's how you can approach model selection:

- Select a Set of Candidate Models:

Start by choosing a set of candidate machine learning algorithms. In your case, you mentioned Logistic Regression, Random Forest, and Gradient Boosting. These are excellent choices as they cover a range of algorithm types (linear, ensemble, boosting).

- Baseline Models:

Train baseline models for each of the selected algorithms without any hyperparameter tuning. This provides a starting point to assess their initial performance.

- Hyperparameter Tuning:

Perform hyperparameter tuning for each algorithm using techniques like grid search or random search. Adjust hyperparameters to optimize model performance. Hyperparameters may include learning rates, tree depths, regularization strengths, etc.

- Cross-Validation:

Utilize k-fold cross-validation to evaluate model performance. This technique helps you assess how well each model generalizes to unseen data.

- Evaluation Metrics:

Choose appropriate evaluation metrics for assessing model performance. For binary classification tasks like diabetes prediction, common metrics include accuracy, precision, recall, F1-score, ROC-AUC, and precision-recall curves.

4. Deployment:

Deployment in the context of machine learning refers to the process of making a trained model available for use in real-world applications. Here's a brief explanation of the deployment process:

- *Scalability:*

Ensure that your deployment infrastructure can handle the expected load. Consider factors like the number of concurrent users and the volume of prediction requests.

- *API Development:*

Create an API (Application Programming Interface) to expose your trained model. This API will receive input data and return predictions. Popular frameworks for building APIs include Flask and FastAPI in Python.

- *Model Serialization:*

Serialize your trained model to a format suitable for deployment. Common formats include pickle, joblib, or ONNX for compatibility with various deployment platforms.

- *Monitoring and Logging:*

Implement monitoring and logging mechanisms to track the system's performance in real-time. Monitor model drift, data quality, and system health to identify issues early.

- *Testing in Production:*

Perform testing in a production-like environment before full deployment. This helps uncover any deployment-specific issues that may not have been apparent during development.

- *Security Measures:*

Implement security measures to protect against potential threats, including data breaches or attacks on the deployed system.

- *Documentation:*

Create thorough documentation for the deployment process, API usage, and troubleshooting procedures. This documentation is essential for the system's maintenance and support.

- *Regulatory Compliance:*

Ensure that your system complies with any relevant regulations and obtain necessary approvals if handling medical data or operating in a regulated environment.

- *User Training:*

If applicable, provide training to users who will interact with the system, such as healthcare professionals who use the predictions for patient care.

5. Iterative Improvement:

Iterative improvement is a crucial part of the machine learning development process, and it's essential for continuously enhancing the performance of your diabetes risk prediction system. Here's how you can approach iterative improvement:

- Fine-Tuning Model Parameters:

Continue to experiment with different hyperparameters for your chosen machine learning algorithms. Use techniques like grid search or random search to systematically explore the hyperparameter space.

Consider conducting hyperparameter tuning in combination with cross-validation to find the best parameter values that generalize well to unseen data.

- Feature Scaling and Transformation:

Revisit feature scaling and transformation methods to ensure that the data preprocessing steps are optimized for the chosen model(s).

Experiment with different scaling techniques to see if they have an impact on model performance.

- Feature Selection Refinement:

Continue to monitor the performance of your selected features. If new data becomes available or if you discover that certain features are no longer relevant, adjust your feature selection accordingly.

- Ensemble Models:

Explore more sophisticated ensemble techniques, such as stacking multiple models, to harness the strengths of different algorithms. Ensemble models can often lead to improved performance.

- Cross-Validation and Validation Set:

Maintain a robust cross-validation strategy to ensure that you have a good estimate of your model's generalization performance.

Regularly validate your model on a separate validation set to monitor its performance over time.

DATA PREPROCESSING STEPS:

```
# Step 1:Import necessary libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# Step 2:Load the dataset
df = pd.read_csv('/kaggle/input/diabetes-data-set/diabetes.csv')

# Step 3: Data Cleaning

# Check for Missing Values
missing_values = df.isnull().sum()
print("Missing Values:")
print(missing_values)

# Handle missing values (if any)
# For example, fill missing values with the mean of the column
mean_fill = df.mean()
df.fillna(mean_fill, inplace=True)

# Check for Duplicate Rows
```

```
duplicate_rows = df[df.duplicated()]
print("\nDuplicate Rows:")
print(duplicate_rows)

# Handle duplicate rows (if any)
# For example, drop duplicate rows
df.drop_duplicates(inplace=True)

# Step 4: Data Analysis

# Summary Statistics
summary_stats = df.describe()
print("\nSummary Statistics:")
print(summary_stats)

# Class Distribution (for binary classification problems)
class_distribution = df['Outcome'].value_counts()
print("\nClass Distribution:")
print(class_distribution)

# Step 5:Visualization (e.g., histograms, box plots, correlation matrix)
sns.pairplot(df, hue='Outcome')
plt.show()

# Step 6: Support Vector Machine (SVM) Modeling

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

# Split the dataset into a training and testing set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardize features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Initialize and train the SVM model
model = SVC(kernel='linear', random_state=42)
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)
```

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

# Classification report and confusion matrix
print(classification_report(y_test, y_pred))
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt='d')
plt.show()
```

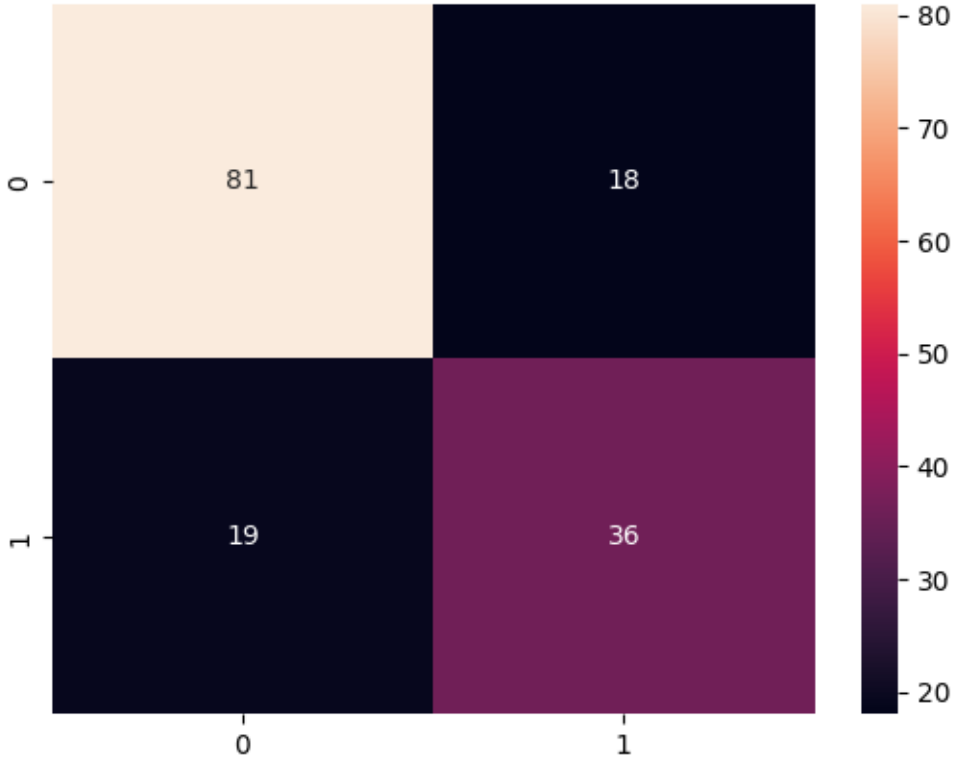
```
Missing Values:
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64

Duplicate Rows:
Empty DataFrame
Columns: [Pregnancies, Glucose, BloodPressure, SkinThickness, Insulin, BMI, DiabetesPedigr
eeFunction, Age, Outcome]
Index: []

Summary Statistics:
       Pregnancies     Glucose  BloodPressure  SkinThickness     Insulin  \
count   768.000000  768.000000     768.000000     768.000000  768.000000
mean      3.845052  120.894531      69.105469      20.536458   79.799479
std       3.369578   31.972618      19.355807      15.952218  115.244002
min       0.000000    0.000000       0.000000       0.000000    0.000000
25%       1.000000   99.000000      62.000000       0.000000    0.000000
50%       3.000000  117.000000      72.000000      23.000000   30.500000
75%       6.000000  140.250000      80.000000      32.000000  127.250000
max      17.000000  199.000000     122.000000      99.000000  846.000000

              BMI  DiabetesPedigreeFunction         Age     Outcome
count  768.000000                768.000000  768.000000  768.000000
mean    31.992578                  0.471876   33.240885    0.348958
std      7.884160                  0.331329   11.760232    0.476951
```

min	0.000000	0.078000	21.000000	0.000000
25%	27.300000	0.243750	24.000000	0.000000
50%	32.000000	0.372500	29.000000	0.000000
75%	36.600000	0.626250	41.000000	1.000000
max	67.100000	2.420000	81.000000	1.000000

Class Distribution:
Outcome
0 500
1 268
Name: count, dtype: int64
/opt/conda/lib/python3.10/site-packages/seaborn/axisgrid.py:118: UserWarning: The figure layout has changed to tight
self._figure.tight_layout(*args, **kwargs)

Accuracy: 0.76

	precision	recall	f1-score	support
0	0.81	0.82	0.81	99
1	0.67	0.65	0.66	55
accuracy			0.76	154
macro avg	0.74	0.74	0.74	154
weighted avg	0.76	0.76	0.76	154



FEATURE ENGINEERING TECHNIQUES:

Feature selection is a crucial step in building an effective diabetes risk prediction model. It involves choosing the most relevant features from your dataset to reduce dimensionality and improve the model's performance. Here's how you can approach feature selection for our diabetes prediction system:

- Understand the Domain:

Start by gaining a deep understanding of the domain. Collaborate with healthcare experts to identify which features are most clinically relevant for diabetes risk prediction.

- Explore Feature Importance:

Utilize techniques like feature importance scores from tree-based models (e.g., Random Forest or XGBoost) to identify features that contribute the most to prediction accuracy.

- Correlation Analysis:

Calculate pairwise correlations between features. Remove features that are highly correlated with each other as they may introduce multicollinearity. Retain the feature that is more clinically significant.

- Univariate Feature Selection:

Use statistical tests (e.g., chi-squared, ANOVA) to select features that have a significant impact on the target variable (diabetes diagnosis). These tests can identify features with the strongest association with diabetes.

- Recursive Feature Elimination (RFE):

Employ RFE with machine learning models like Logistic Regression or SVM to iteratively remove the least important features until you reach the desired number of features.

MACHINE LEARNING ALGORITHM:

In this project, we have used **Random Forest** Machine Learning Algorithm because it is a **versatile and powerful machine learning algorithm**, often used for various **predictive tasks, including diabetes prediction**. Random Forest is an ensemble learning method that combines multiple decision trees to make predictions. It's based on the idea that **aggregating the predictions of multiple models can lead to a more accurate and robust model**. Random Forest introduces randomness during the training process by bootstrapping (sampling with replacement) the data and randomly selecting a subset of features for each tree. This randomness **helps reduce overfitting and increases the diversity** among the individual trees, making the ensemble more accurate.

Some of its uses:

1. Classification: Random Forest is well-suited for classification tasks, such as predicting whether an individual has diabetes (binary classification) in a diabetes prediction project. It can effectively handle imbalanced datasets and provide accurate predictions.

2. Feature Importance: Random Forest can measure the importance of each feature in the prediction process. This is valuable for feature selection and understanding which features are most relevant for diabetes prediction.

3. Handling Complex Interactions: Diabetes prediction may involve complex interactions between different medical features (e.g., glucose levels, BMI, age). Random Forest can capture these complex relationships due to its ability to model non-linear and interactive effects.

4. Robustness: Random Forest is robust against outliers and noisy data, making it suitable for real-world healthcare datasets, which can often have data quality issues.

5. High-Dimensional Data: If the dataset contains a large number of features, Random Forest can handle high-dimensional data effectively, making it suitable for medical datasets with numerous variables.

6. Hyperparameter Tuning: Random Forest provides various hyperparameters (e.g., number of trees, tree depth, feature selection methods) that can be fine-tuned to optimize the model's performance.

7. Ensemble Learning: Random Forest is an ensemble method by design, which means it combines multiple decision trees. It can further improve predictive accuracy when compared to individual decision trees.

MODEL TRAINING:

Model training is a crucial step in the development of machine learning models. It involves the process of teaching the model to recognize patterns, make predictions, or perform a specific task based on data. Here's a brief overview of the model training process:

- *Data Splitting:*

Before any testing, split your dataset into three parts: training data, validation data, and test data. Common splits are 70-80% for training, 10-15% for validation, and 10-15% for testing. The validation set is used during model development and hyperparameter tuning, while the test set is kept separate for final evaluation.

- *Model Evaluation:*

Evaluate your trained diabetes prediction model using various evaluation metrics, including accuracy, precision, recall, F1-score, and ROC AUC. Use the test dataset for this evaluation to assess how well the model generalizes to unseen data.

- *Cross-Validation:*

Perform cross-validation (e.g., k-fold cross-validation) to assess the model's stability and consistency. This involves splitting the training data into multiple

subsets, training the model on different subsets, and evaluating its performance. Cross-validation helps identify potential overfitting.

- *Performance Optimization:*

Fine-tune your model's hyperparameters based on validation results. You can use techniques like grid search or random search to find the best combination of hyperparameters.

- *Bias and Fairness Testing:*

Check for biases in your model predictions, especially if the dataset exhibits bias or if certain groups are underrepresented. Ensure that the model's predictions are fair and unbiased across different demographic groups.

- *Robustness Testing:*

Assess how well the model performs under various conditions, including noisy data, missing values, or outliers. Robustness testing helps ensure the model's reliability in real-world scenarios.

- *Security and Privacy Testing:*

Verify that the system follows best practices for data security and privacy protection. Ensure that sensitive patient data is handled securely and that the system complies with relevant regulations (e.g., GDPR or HIPAA).

EVALUATION METRICS:

Evaluating your diabetes risk prediction model using a variety of metrics is crucial to assess its performance comprehensively. The metrics you mentioned—accuracy, precision, recall, F1-score, and ROC-AUC—are indeed appropriate for a binary classification problem like this. Here's how to interpret and use these metrics:

- Accuracy:

Accuracy measures the proportion of correctly predicted instances (both true positives and true negatives) out of all instances. It's a commonly used metric, but it may not be the best choice if your dataset has imbalanced class distribution.

- Precision:

Precision measures the proportion of true positive predictions out of all positive predictions. It is useful when you want to minimize false positive predictions. High precision indicates that the model has a low rate of false positives.

- Recall (Sensitivity):

Recall measures the proportion of true positive predictions out of all actual positives. It is important when you want to minimize false negatives. High recall indicates that the model has a low rate of false negatives.

- F1-Score:

The F1-score is the harmonic mean of precision and recall. It provides a balance between precision and recall. It's particularly useful when there is an uneven class distribution or when both false positives and false negatives are important.

- ROC-AUC (Receiver Operating Characteristic - Area Under the Curve):

ROC-AUC measures the ability of the model to distinguish between positive and negative classes across different probability thresholds. A higher ROC-AUC value indicates better discrimination ability. It's a valuable metric when you want to assess the overall performance of a classifier regardless of the chosen threshold.

PROGRAM CODE FOR FEATURE ENGINEERING,MODEL TRAINING AND EVALUATION:

```
# Import necessary libraries

import pandas as pd

import numpy as np

from sklearn.model_selection import train_test_split, cross_val_score, GridSearchCV

from sklearn.preprocessing import StandardScaler, OneHotEncoder

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score

# Step 1: Load the dataset

data = pd.read_csv('diabetes_data.csv')

# Step 2: Feature Engineering

# Categorize BMI as 'BMI_Category'

def categorize_bmi(bmi):

    if bmi < 18.5:
```

```
        return 'Underweight'
    elif 18.5 <= bmi < 24.9:
        return 'Normal Weight'
    elif 24.9 <= bmi < 29.9:
        return 'Overweight'
    else:
        return 'Obese'
data['BMI_Category'] = data['BMI'].apply(categorize_bmi)
# Step 3: Split the data into features (X) and target (y)
X = data.drop('Outcome', axis=1)
y = data['Outcome']
# Step 4: Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
# Step 5: Feature Engineering and Preprocessing
# Apply feature scaling to numerical features
numerical_features = ['Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI',
'DiabetesPedigreeFunction', 'Age']
categorical_features = ['BMI_Category']
# Feature Scaling
scaler = StandardScaler()
X_train[numerical_features] = scaler.fit_transform(X_train[numerical_features])
X_test[numerical_features] = scaler.transform(X_test[numerical_features])
# One-Hot Encoding for categorical features
encoder = OneHotEncoder(drop='first', sparse=False)
X_train_encoded = encoder.fit_transform(X_train[categorical_features])
X_test_encoded = encoder.transform(X_test[categorical_features])
```

```
# Concatenate the encoded features with the original features
X_train = np.concatenate([X_train.drop(categorical_features, axis=1).values, X_train_encoded], axis=1)
X_test = np.concatenate([X_test.drop(categorical_features, axis=1).values, X_test_encoded], axis=1)
```

 # Step 6: Model Training and Hyperparameter Tuning

```
param_grid = {
    'n_estimators': [50, 100, 150],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}
rf = RandomForestClassifier(random_state=42)
grid_search = GridSearchCV(estimator=rf, param_grid=param_grid, cv=5, scoring='accuracy')
grid_search.fit(X_train, y_train)
# Get the best model from the hyperparameter tuning
best_model = grid_search.best_estimator_
```

 # Step 7: Model Evaluation with Cross-Validation

```
cv_scores = cross_val_score(best_model, X_train, y_train, cv=5, scoring='accuracy')
print("Cross-Validation Scores:", cv_scores)
print("Mean CV Accuracy:", np.mean(cv_scores))
# Evaluate the model on the test set
y_pred = best_model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
```

f1 = f1_score(y_test, y_pred)

roc_auc = roc_auc_score(y_test, best_model.predict_proba(X_test)[:, 1])

 # Step 8: Display Results

print("Test Set Accuracy:", accuracy)

print("Test Set Precision:", precision)

print("Test Set Recall:", recall)

print("Test Set F1 Score:", f1)

print("Test Set ROC AUC Score:", roc_auc)

CODING:

#Importing 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
```

#Here we will be reading the dataset which is in the CSV format

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

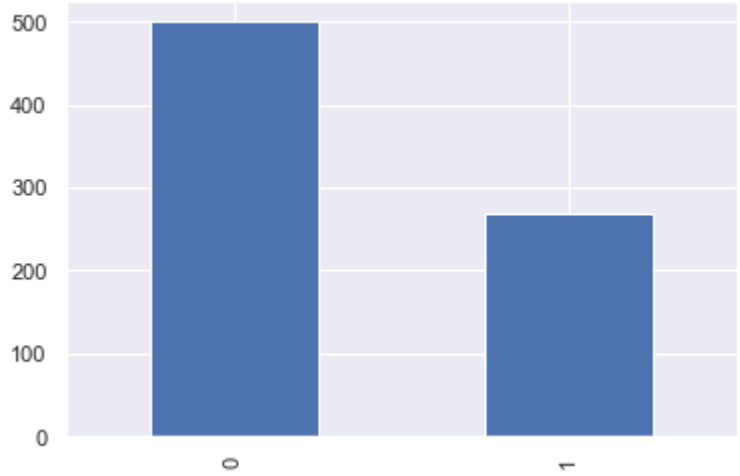
Output:

	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

```
color_wheel = {1: "#0392cf", 2: "#7bc043"}
colors = diabetes_df["Outcome"].map(lambda x: color_wheel.get(x + 1))
print(diabetes_df.Outcome.value_counts())
p=diabetes_df.Outcome.value_counts().plot(kind="bar")
```

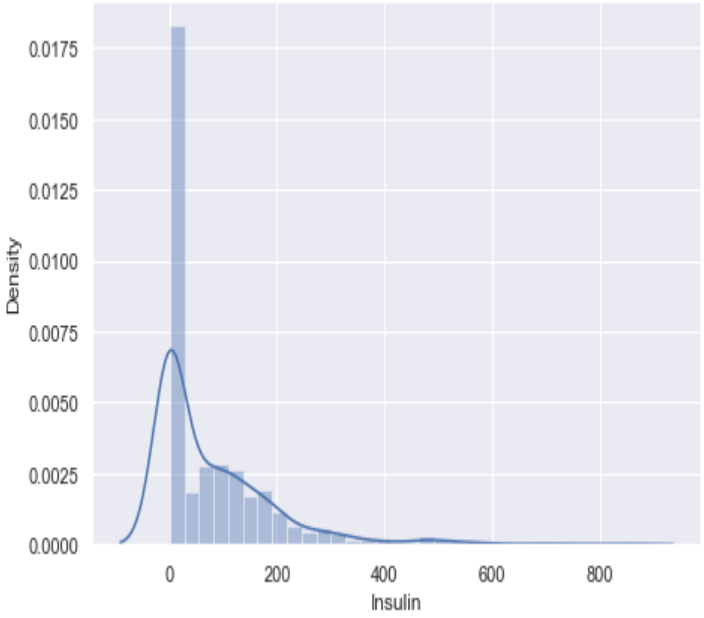
Output:

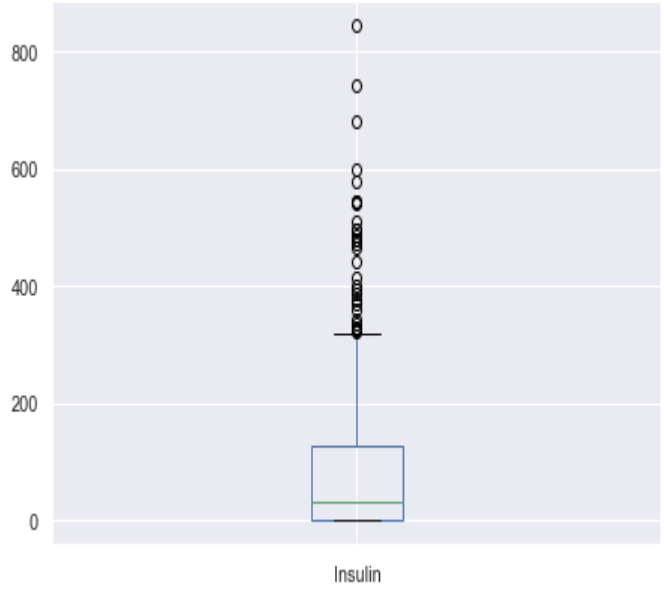
```
0    500
1    268
Name: Outcome, dtype: int64
```




```
plt.subplot(121), sns.distplot(diabetes_df['Insulin'])
plt.subplot(122), diabetes_df['Insulin'].plot.box(figsize=(16,5))
plt.show()
```

Output:

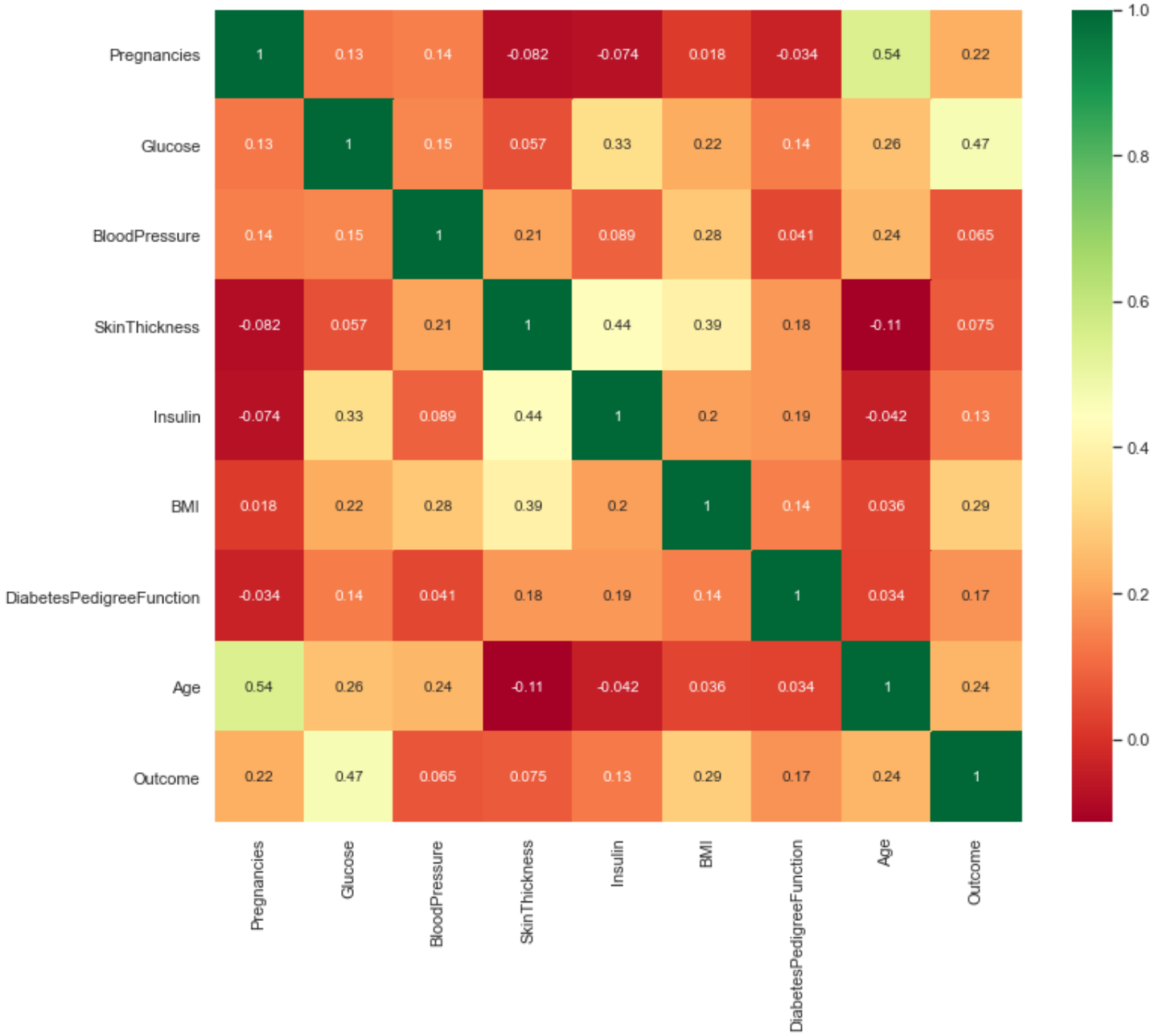




Correlation between all the features before cleaning

```
plt.figure(figsize=(12,10))
# seaborn has an easy method to showcase heatmap
p = sns.heatmap(diabetes_df.corr(), annot=True,cmap ='RdYlGn')
```

Output:



Scaling the Data

diabetes_df_copy.head()

Output:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148.0	72.0	35.0	125.0	33.6	0.627	50	1
1	1	85.0	66.0	29.0	125.0	26.6	0.351	31	0
2	8	183.0	64.0	29.0	125.0	23.3	0.672	32	1
3	1	89.0	66.0	23.0	94.0	28.1	0.167	21	0
4	0	137.0	40.0	35.0	168.0	43.1	2.288	33	1

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

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)
```

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
```