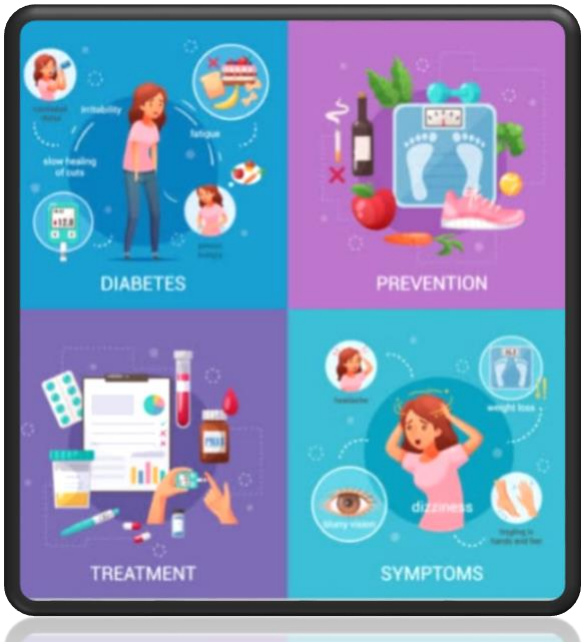
PREDICTING DIABETES SYSTEM USING MACHINE LEARNING

TEAM MEMBER: R. MAHAALAKSHMY

PHASE 2 SUBMISSION DOCUMENT

Project: Diabetes Prediction System



INTRODUTION:

A diabetes prediction system is a technological advancement designed to forecast the likelihood of an individual developing diabetes or experiencing fluctuations in blood glucose levels. By harnessing the power of data analytics, including factors like genetics, lifestyle, and medical history, these systems aim to provide early warnings and personalized

recommendations for diabetes prevention and management. Such innovations have the potential to revolutionize healthcare by empowering individuals to take proactive measures to reduce their risk of diabetes or to better control their condition, ultimately leading to improved overall health outcomes.

An innovation in diabetes prediction systems involves leveraging advanced machine learning algorithms and wearable health technology to monitor an individual's physiological data, such as blood glucose levels, heart rate, and activity patterns. By analyzing this data, the system can provide real-time predictions and alerts for potential fluctuations in blood sugar levels, enabling individuals with diabetes to proactively manage their condition and make informed decisions about their lifestyle and medication. This innovation aims to improve the quality of life for people living with diabetes by enhancing their ability to maintain optimal blood sugar control and minimize the risk of complications.

Content for Project Phase 2:

Consider exploring advanced regression techniques like Gradient Boosting or XGBoost for improved Prediction accuracy.

Data Source:

A Good data source for diabetes prediction using machine learning should be Accurate, Complete, covering the total Data given by the dataset.

DATASET LINK:

https://www.kaggle.com/code/mahaalakshmv/diabetes-prediction-using-different-models/edit

	A	B	C	D	E	F	G	H	I	J
1	Pregnanci	Glucose	BloodPres	SkinThickr	Insulin	BMI	DiabetesF	Age	Outcome	
2	6	148	72	35	0	33.6	0.627	50	1	
3	1	85	66	29	0	26.6	0.351	31	0	
4	8	183	64	0	0	23.3	0.672	32	1	
5	1	89	66	23	94	28.1	0.167	21	0	
6	0	137	40	35	168	43.1	2.288	33	1	
7	5	116	74	0	0	25.6	0.201	30	0	
8	3	78	50	32	88	31	0.248	26	1	
9	10	115	0	0	0	35.3	0.134	29	0	
10	2	197	70	45	543	30.5	0.158	53	1	
11	8	125	96	0	0	0	0.232	54	1	
12	4	110	92	0	0	37.6	0.191	30	0	
13	10	168	74	0	0	38	0.537	34	1	
14	10	139	80	0	0	27.1	1.441	57	0	
15	1	189	60	23	846	30.1	0.398	59	1	
16	5	166	72	19	175	25.8	0.587	51	1	
17	7	100	0	0	0	30	0.484	32	1	
18	0	118	84	47	230	45.8	0.551	31	1	
19	7	107	74	0	0	29.6	0.254	31	1	
20	1	103	30	38	83	43.3	0.183	33	0	
21	1	115	70	30	96	34.6	0.529	32	1	
22	3	126	88	41	235	39.3	0.704	27	0	
23	8	99	84	0	0	35.4	0.388	50	0	
24	7	196	90	0	0	39.8	0.451	41	1	

Data Collection:

Patient Records: Gather historical health records, including blood glucose measurements, medical history, and medication usage.

Wearable Devices: Collect real-time data from wearable devices like continuous glucose monitors (CGMs), fitness trackers, and smartwatches.

Surveys and Questionnaires: Administer surveys to collect lifestyle information, dietary habits, physical activity, and family history.

Genetics: Incorporate genetic data, if available, to assess the genetic predisposition to diabetes.

Data Preprocessing:

Data Cleaning: Remove duplicates, correct errors, and handle missing values in the collected data.

Feature Selection: Identify relevant features (variables) for prediction, such as glucose levels, BMI, age, and family history.

Data Transformation: Normalize or scale numerical features to bring them to a similar range and encode categorical data.

Time Series Data: If using time series data (e.g., CGM data), handle irregular sampling rates, and possibly apply smoothing techniques.

Outlier Detection: Identify and handle outliers that could skew predictions.

Imbalanced Data: Address class imbalance if there are significantly more non-diabetic cases than diabetic cases.

Exploratory Data Analysis (EDA):

Exploratory Data Analysis (EDA) is a critical step in understanding and gaining insights from the data used in a diabetes prediction system. Here's how you can conduct EDA for such a system:

- Data Summary
- For categorical variables like gender or medication use, calculate frequency counts and percentages.
- Data Visualization
- Box plots can reveal the spread of data and highlight outliers.
- Scatter plots can be useful for understanding relationships between two numerical variables.
- Bar charts and pie charts can represent categorical data, showing the distribution of classes or categories.
- Correlation
- Create a correlation matrix and visualize it using a heatmap to identify strong correlations.
- Visualize the class distribution to determine if any resampling techniques are needed.
- Feature Importance
- Outlier Detection.
- Missing Data

EDA helps you gain a deeper understanding of the data, identify patterns, and inform decisions about data preprocessing, feature engineering, and model selection. It plays a crucial role in building accurate and interpretable diabetes prediction models

.

Feature Engineering:

Feature engineering is a crucial step in developing a diabetes prediction system. It involves creating new features or modifying existing ones to improve the predictive performance of your model. Here are some feature engineering techniques for a diabetes prediction system:

✓ Glucose Metrics
✓ Time-Based Features
✓ BMI Categories
✓ Lifestyle Factors
✓ Medication Features
✓ Family History
✓ Age Groups
✓ Interaction Features
✓ Medical Comorbidities

Feature engineering is the process of transforming raw data into features that are more informative and predictive for machine learning models. It is an essential step in machine learning, as the quality and relevance of the features can have a significant impact on the performance of the model.

There are many different feature engineering techniques that can be used, depending on the specific problem and dataset. Some of the most common techniques include:

- **Handling missing values:** Missing values are a common problem in real-world datasets, and there are a variety of techniques that can be used to handle them, such as imputation, deletion, and feature encoding.
- **Encoding categorical variables:** Categorical variables, such as colors and countries, need to be encoded into numerical values before they can be used by machine learning models. There are two main encoding techniques: label encoding and one-hot encoding.
- **Transforming variables:** Some variables may need to be transformed before they can be used in a machine learning model. For example, a variable that is skewed may need to be log-transformed.
- **Creating new features:** new features can be created by combining existing features or by extracting new information from the data. For example, a new feature could be created to represent the ratio of two other features.
- **Selecting features:** Not all features are created equal. Some features may be more informative and predictive than others. Feature selection techniques can be used to identify the most important features and remove the less important ones.

Here are some examples of feature engineering techniques in action:

- **Predicting customer churn:** A company might want to predict which customers are likely to churn (cancel their subscription). One feature that they could use is the customer's tenure (how long they have been a customer). However, tenure is a continuous variable, and machine learning models often perform better with categorical variables. So, the company could

convert tenure into a categorical variable by creating bins, such as "less than 6 months", "6 months to 1 year", and "more than 1 year".

- **Recommending products:** An online retailer might want to recommend products to customers based on their past purchases. One feature that they could use is the customer's purchase history. However, purchase history is a sparse matrix, with many empty values. So, the retailer could create new features by counting the number of times that the customer has purchased each product category.

Feature engineering is an important part of machine learning, and it can have a significant impact on the performance of your model. By taking the time to understand your data and apply the appropriate feature engineering techniques, you can improve the accuracy and reliability of your predictions.

Advanced Regression Techniques:

There are many different types of regression, each of which is suited for different types of data and different types of relationships. The main types of regression include:

- Linear regression: Linear regression is the most basic type of regression, and it is used to model linear relationships between variables. For example, you could use linear regression to predict the price of a house based on its square footage or to predict the number of customers who will visit your store on a given day based on the weather forecast.

- Polynomial regression: Polynomial regression is a more complex type of regression that can be used to model non-linear relationships between variables. For example, you could use polynomial regression to predict the growth of a plant over time or the spread of a disease through a population.
- Logistic regression: Logistic regression is a type of regression that is used to model binary outcomes. For example, you could use logistic regression to predict whether a customer will click on an ad or whether a patient has a particular disease.
- Ridge regression: Ridge regression is a type of regression that is used to reduce overfitting. Overfitting occurs when a model learns the training data too well and is unable to generalize to new data. Ridge regression works by adding a penalty term to the loss function, which encourages the model to learn simpler relationships between the variables.
- Lasso regression: Lasso regression is another type of regression that is used to reduce overfitting. Lasso regression works by shrinking the coefficients of the model, which forces the model to focus on the most important variables.
- Bayesian regression: Bayesian regression is a type of regression that uses Bayesian statistics to model the relationships between variables. Bayesian regression is more computationally expensive than other types of regression, but it can be more accurate and robust in some cases.
- In addition to the main types of regression listed above, there are many other specialized regression techniques that can be used for specific tasks. For example, there are regression techniques for time series data, spatial data, and count data.

- The best type of regression to use will depend on the specific problem that you are trying to solve and the type of data that you have. If you are not sure which type of regression to use, it is always a good idea to consult with a data scientist or statistician.

Model Evaluation and Selection:

Model evaluation and **selection** is the process of assessing the performance of different machine learning models on a given dataset and choosing the best model for the task. It is an important step in the machine learning workflow, as the performance of the model will have a direct impact on the accuracy and reliability of the predictions.

There are a number of different ways to evaluate machine learning models. Some of the most common metrics include:

1)Accuracy: Accuracy is the percentage of predictions that are correct.

Precision: Precision is the percentage of positive predictions that are actually positive.

2)Recall: Recall is the percentage of positive cases that are correctly identified.

3)F1 score: The F1 score is a harmonic mean of precision and recall.

4)Mean squared error (MSE): MSE is a measure of the average error between the predicted and actual values.

5)Root mean squared error (RMSE): RMSE is the square root of MSE.

Model Interpretability:

Model interpretability is the degree to which a machine learning model can be understood by humans. Interpretable models are easier to trust and debug, and they can help users to understand how the model is making decisions.

There are a number of different ways to make machine learning models more interpretable. Some common techniques include:

Using simpler models: Simpler models, such as linear regression models, are generally more interpretable than complex models, such as deep learning models.

Using feature importance: Feature importance techniques can be used to identify the most important features in a model. This can help users to understand which features are having the biggest impact on the model's predictions.

Using partial dependence plots: Partial dependence plots can be used to visualize how the model's predictions change in response to changes in the input features. This can help users to understand how the model is using the different features to make predictions.

Using counterfactual explanations: Counterfactual explanations can be used to explain why the model made a particular prediction. This is done by identifying the smallest possible change to the input features that would cause the model to make a different prediction.

Model interpretability is an important consideration for many machine learning applications. For example, interpretable models are often used in high-stakes applications such as healthcare and finance, where it is important to understand why the model is making certain decisions.

Here are some examples of model interpretability in action:

A doctor might want to use an interpretable machine learning model to predict the risk of a patient developing a particular disease. This would allow the doctor to understand which factors are contributing to the patient's risk, and to make more informed decisions about treatment.

A bank might want to use an interpretable machine learning model to decide whether to approve a loan application. This would allow the bank to explain to the applicant why their loan was approved or denied, and to help them to improve their chances of getting approved in the future.

Model interpretability is a complex and challenging area of research, but it is an important area for future development. By making machine learning models more interpretable, we can increase trust in these models and enable them to be used in a wider range of applications.

Deployment and Prediction:

- ❖ Deploy the chosen regression model to prediction of diabetes system.
- ❖ Develop the user-friendly interface for users to input data features and diabetes predictions.

Program:

Diabetes Prediction System

```
import numpy as np
import pandas as pd
import os
from sklearn.preprocessing import StandardScaler
```

```
from sklearn.model_selection import train_test_split, RandomizedSearchCV, GridSearch
CV
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, confusion_matrix
from mlxtend.plotting import plot_confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
import xgboost as xgb
from sklearn.ensemble import AdaBoostClassifier
from sklearn.gaussian_process import GaussianProcessClassifier
from sklearn.gaussian_process.kernels import RBF
from sklearn.ensemble import GradientBoostingClassifier
```

```
/opt/conda/lib/python3.10/site-packages/scipy/__init__.py:146: UserWarning: A NumP
y version >=1.16.5 and <1.23.0 is required for this version of SciPy (detected version 1.2
3.5
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
```

In [2]:

```
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

```
/kaggle/input/pima-indians-diabetes-database/diabetes.csv
```

In [3]:

```
df = pd.read_csv('/kaggle/input/pima-indians-diabetes-database/diabetes.csv')
print(list(df.columns))
df.head(5)
```

```
['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI', 'DiabetesPedig
reeFunction', 'Age', 'Outcome']
```

Out[3]:

	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

In [4]:

linkcode

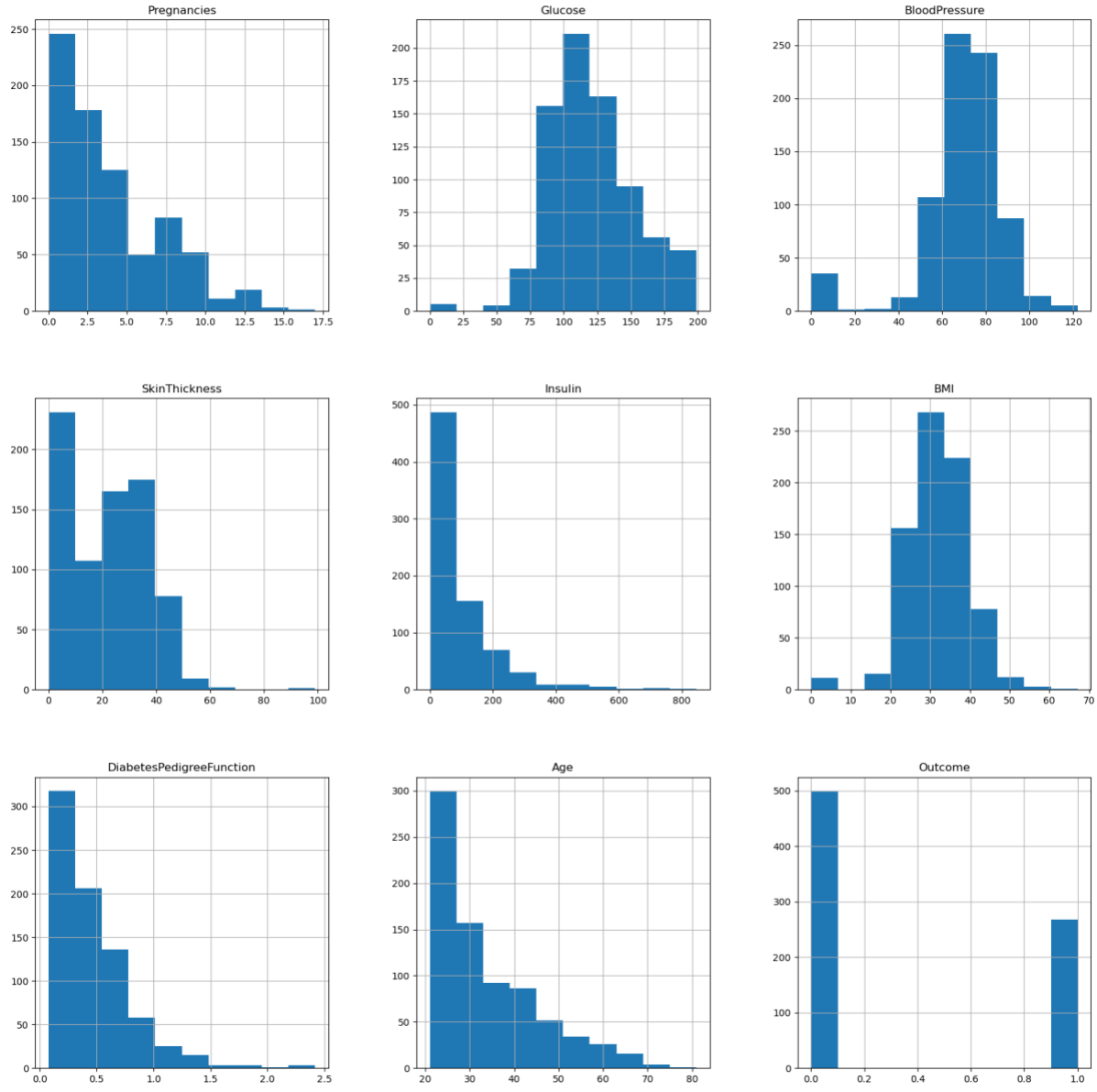
```
df_copy = df.copy(deep = True)
df_copy[['Glucose','BloodPressure','SkinThickness','Insulin','BMI']] = df_copy[['Glucose','
BloodPressure','SkinThickness','Insulin','BMI']].replace(0,np.NaN)

## showing the count of Nans
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 [5]:

```
p = df.hist(figsize = (20,20))
df_copy['Glucose'].fillna(df_copy['Glucose'].mean(), inplace = True)
```

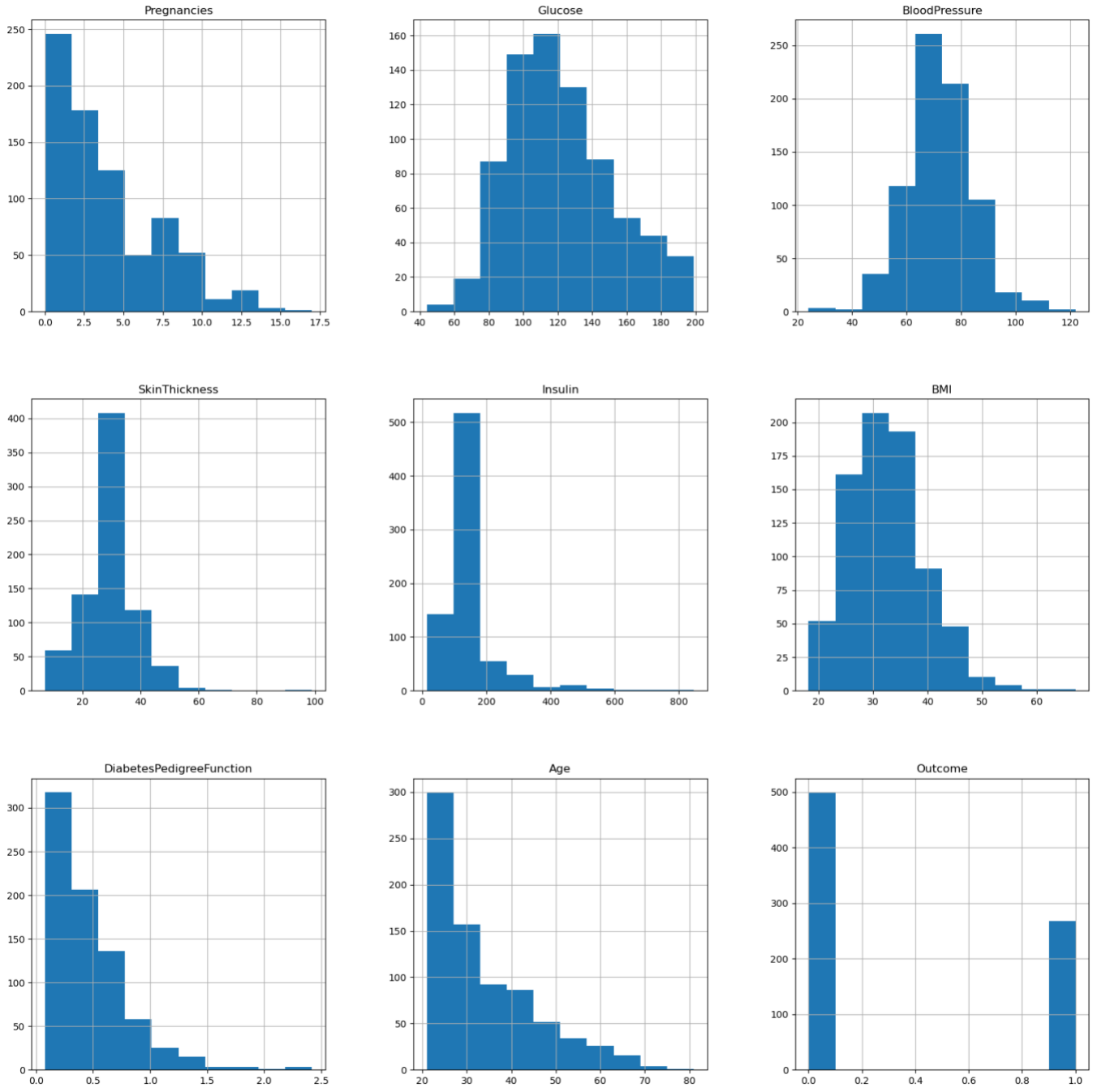



```
df_copy['BloodPressure'].fillna(df_copy['BloodPressure'].mean(), inplace = True)
df_copy['SkinThickness'].fillna(df_copy['SkinThickness'].median(), inplace = True)
```

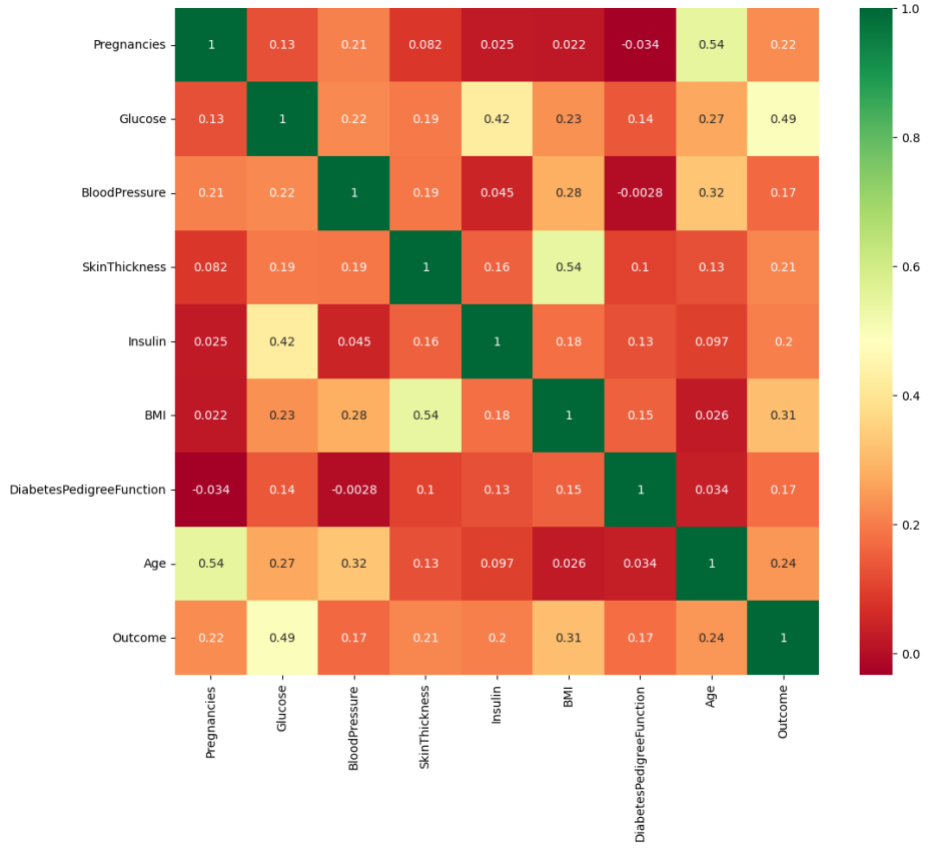
```
df_copy['Insulin'].fillna(df_copy['Insulin'].median(), inplace = True)
df_copy['BMI'].fillna(df_copy['BMI'].median(), inplace = True)
```

In [7]:

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



```
plt.figure(figsize=(12,10))  # on this line I just set the size of figure to
```



12 by 10.

```
p=sns.heatmap(df_copy.corr(), annot=True,cmap ='RdYlGn')
X = df_copy.drop(['Outcome'], axis = 1)
y = df_copy["Outcome"]

std = StandardScaler()
X = std.fit_transform(X)
data = pd.DataFrame(X)
data.head()
```

Out[9]:

	0	1	2	3	4	5	6	7
0	0.639947	0.865108	-0.033518	0.670643	-0.181541	0.166619	0.468492	1.425995

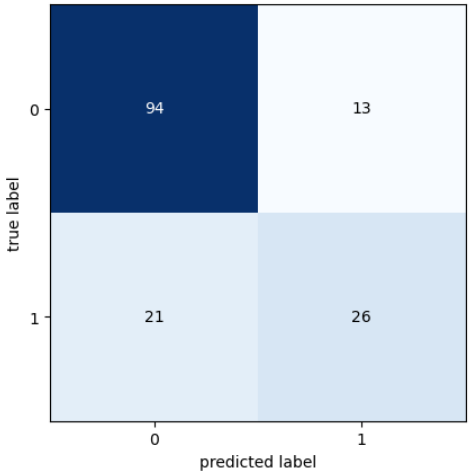
	0	1	2	3	4	5	6	7
1	-0.844885	-1.206162	-0.529859	-0.012301	-0.181541	-0.852200	-0.365061	-0.190672
2	1.233880	2.015813	-0.695306	-0.012301	-0.181541	-1.332500	0.604397	-0.105584
3	-0.844885	-1.074652	-0.529859	-0.695245	-0.540642	-0.633881	-0.920763	-1.041549
4	-1.141852	0.503458	-2.680669	0.670643	0.316566	1.549303	5.484909	-0.020496

```
X_train, X_test, y_train, y_test = train_test_split(data, y, test_size = 0.2, random_state = 0)
models = [[SVC(), "Support vector"],
      [LogisticRegression(), "Logistic regression"],
       [RandomForestClassifier(n_estimators = 10), "Random Forest"],
       [DecisionTreeClassifier(max_depth = 7), "Decision Trees"],
       [KNeighborsClassifier(n_neighbors = 7), "KNeighbourClassifier"],
       [xgb.XGBClassifier(), "XGBoost"],
       [AdaBoostClassifier(n_estimators = 25), "Adaboost"],
       [GradientBoostingClassifier(n_estimators=50, learning_rate=0.1, max_depth=10, random_state=0), "Gradient Boosting"],
       [GaussianProcessClassifier(kernel=RBF(1.0),random_state=0), 'Gaussian Process']]
for i in models:
```

```
    name = i[1]
    model = i[0]
    print(name)
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    print(accuracy_score(y_test, y_pred) * 100)
    cnf = confusion_matrix(y_test,y_pred)
    fig, ax = plot_confusion_matrix(conf_mat = cnf)
plt.show()
        print("\n")
```

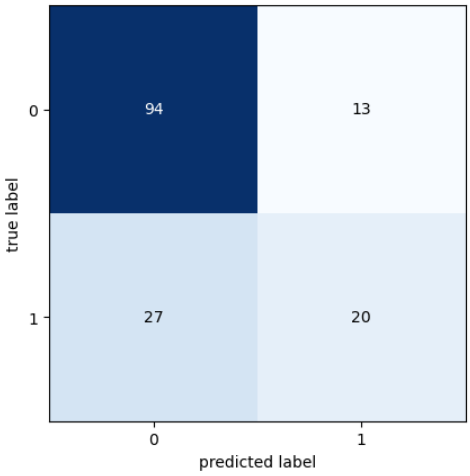
Support vector

74.02597402597402



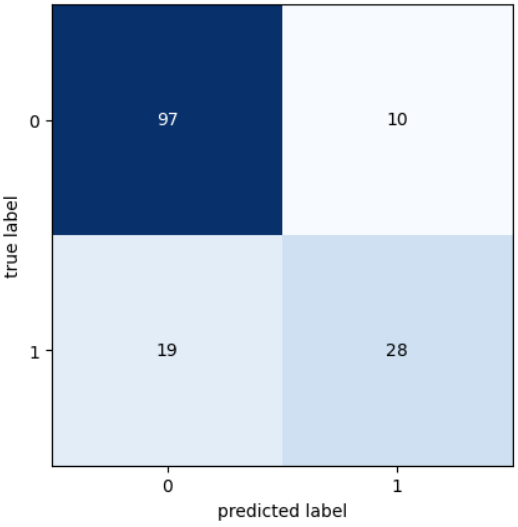
Random Forest

77.92207792207793



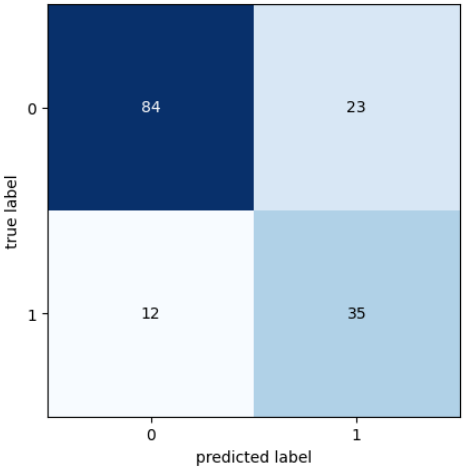
Logistic regression

81.16883116883116

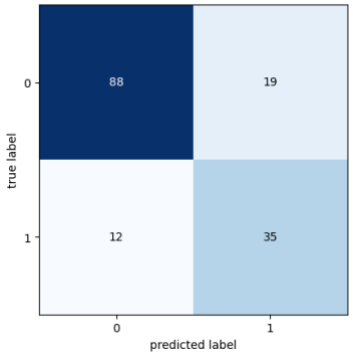


Decision Trees

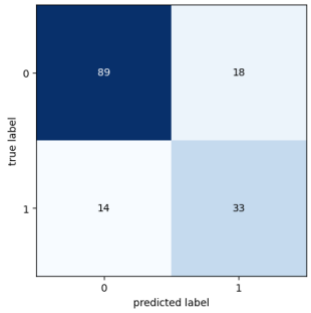
77.27272727272727



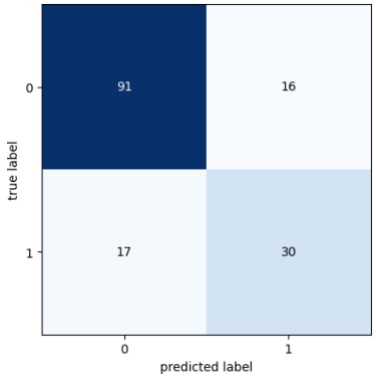
KNeighbourClassifier
78.57142857142857



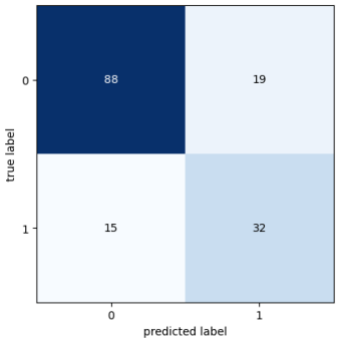
Adaboost
79.22077922077922



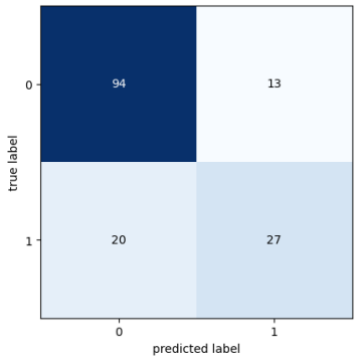
XGBoost
79.87012987012987



Gradient Boosting
77.92207792207793



Gaussian Process
78.57142857142857



```
n_estimators = [int(x) for x in np.linspace(start = 200, stop = 2000, num = 10)]
max_features = ['auto', 'sqrt']
max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
max_depth.append(None)
min_samples_split = [2, 5, 10]
min_samples_leaf = [1, 2, 4]
bootstrap = [True, False]
min_weight_fraction_leaf = [0, 1, 2, 3]
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap,
              'min_weight_fraction_leaf': min_weight_fraction_leaf}
print(random_grid)
```

```
{'n_estimators': [200, 400, 600, 800, 1000, 1200, 1400, 1600, 1800, 2000], 'max_feature
s': ['auto', 'sqrt'], 'max_depth': [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, None], 'min_s
amples_split': [2, 5, 10], 'min_samples_leaf': [1, 2, 4], 'bootstrap': [True, False], 'min_wei
ght_fraction_leaf': [0, 1, 2, 3]}
```

```
rf = RandomForestClassifier()
rf_random = RandomizedSearchCV(estimator = rf, param_distributions = random_grid, n
_iter = 100, cv = 3, verbose=0, random_state=42, n_jobs = -1)
rf_random.fit(X_train, y_train)
import warnings
warnings.filterwarnings('ignore')
params = rf_random.best_params_
```

In [14]:

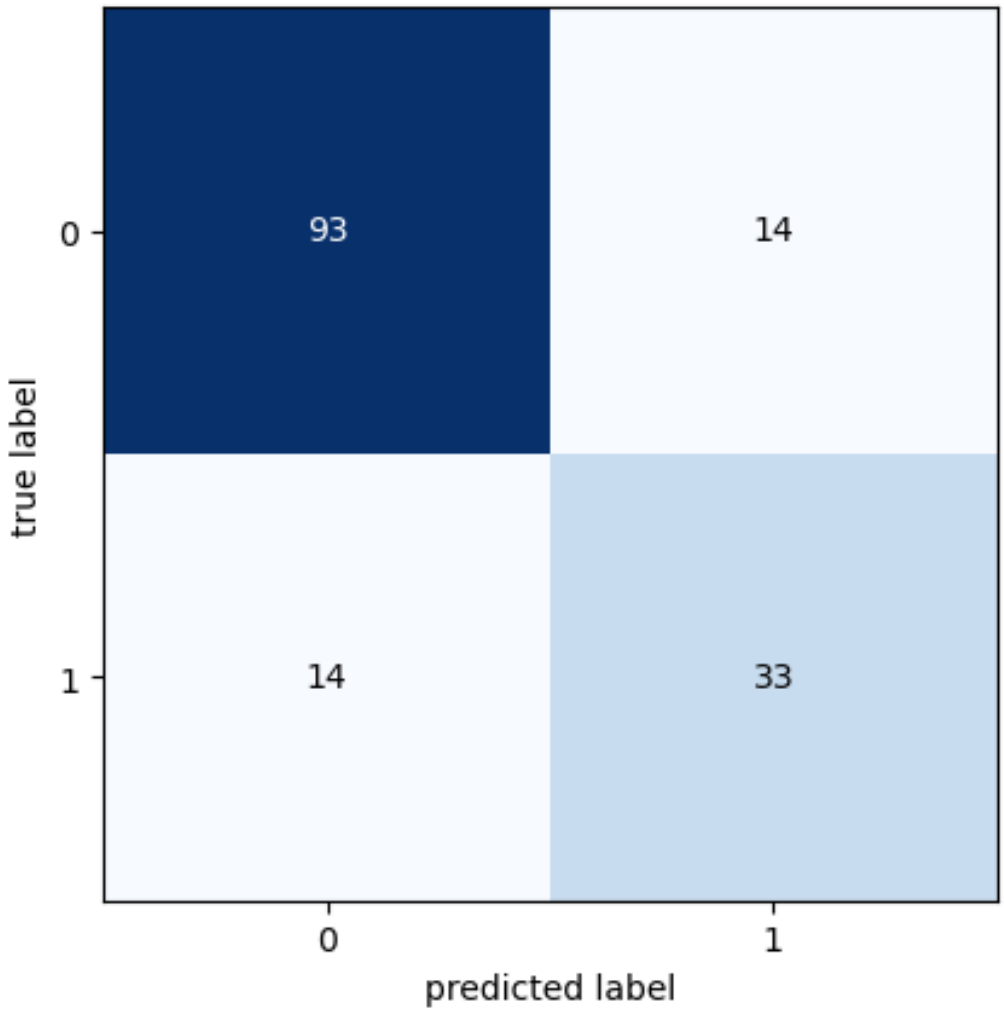
```
print(rf_random.best_params_)
print(rf_random.best_score_)
best_model = rf_random.best_estimator_
y_pred = best_model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(accuracy)
```

{'n_estimators': 600, 'min_weight_fraction_leaf': 0, 'min_samples_split': 5, 'min_sample
s_leaf': 1, 'max_features': 'sqrt', 'max_depth': None, 'bootstrap': True}
0.7458871353419417
0.8181818181818182

In [15]:

linkcode

```
cnf = confusion_matrix(y_test,y_pred)
fig, ax = plot_confusion_matrix(conf_mat = cnf)
plt.show()
```



Conclusion:

- In the phase 2 conclusion, we will summarize the key findings and insights from the advanced regression techniques. We will reiterate the impact of these techniques on improving the accuracy and robustness of diabetes prediction.
- Future Work: We will discuss potential avenues for future work, duchesse as incorporating additional data sources (e g., real-time economic indicators), exploring deep learning models for prediction, or expanding the project into a web application with more features and interactivity.