Diabetes Prediction using Machine Learning — Python

Introduction

Diabetes is a health condition that affects how your body turns food into energy. Most of the food you eat is broken down into sugar (also called glucose) and released into your bloodstream. When your blood sugar goes up, it signals your pancreas to release insulin.

Without ongoing, careful management, diabetes can lead to a buildup of sugars in the blood, which can increase the risk of dangerous complications, including stroke and heart disease. So that i decide to predict using Machine Learning in Python.

Innovation

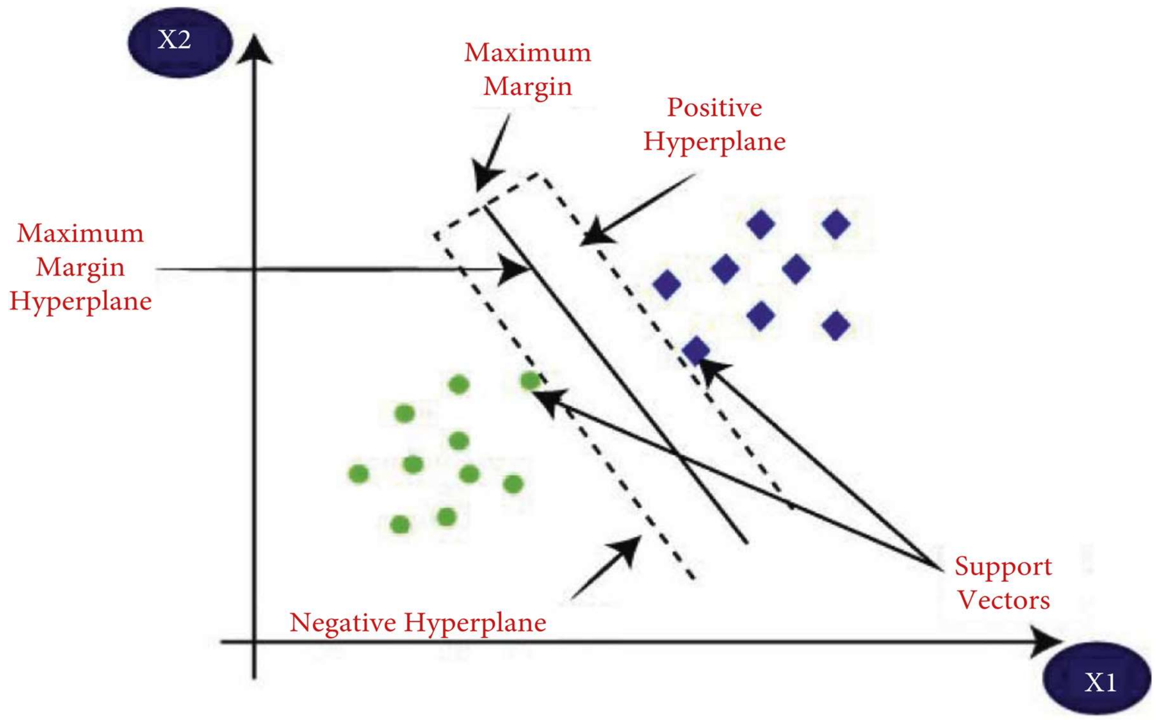
In this subsection, we briefly describe the different machine learning methods as well as the Deep Neural Networks that we used for evaluating the proposed system.

Logistic Regression

Logistic Regression (LR) is a subset of generalized linear models which deals with the analysis of binary data, which seeks out the best-fitting model for describing the connection between dependent and independent predictors [40, 41]. When it comes to predicting sickness or health status, the LR model is most commonly used [42, 43]. Based on the risk factors given, the LR model can calculate the likelihood of an individual acquiring diabetes disease

Support Vector Machine

SVM is a nonprobabilistic classifier with a separating hyperplane as its formal definition. The technique creates an ideal hyperplane with the greatest distance from the support vectors based on the available training data (supervised learning). This hyperplane is a line that divides a plane into two classes in two-dimensional space. The epsilon ε, regularization, and kernel parameters are the SVM classifier's tuning parameters [6, 44]. The principle of SVM is shown below in Figure 2.

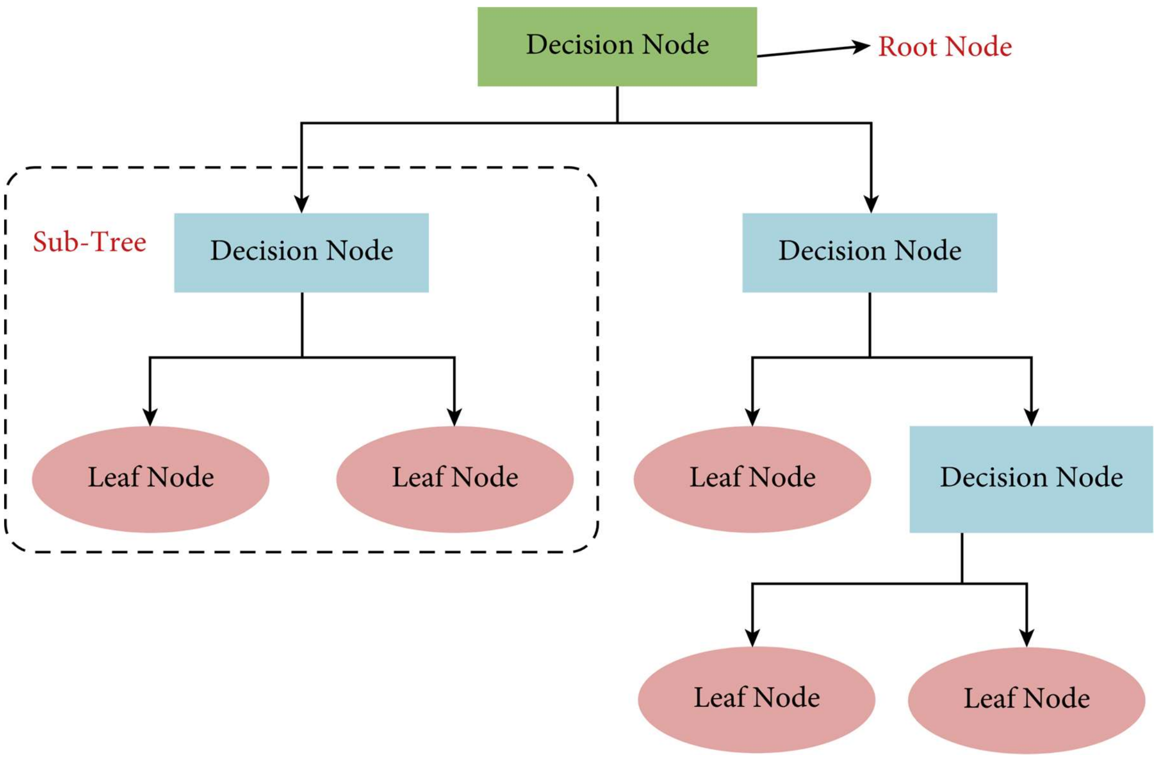


Extreme Gradient Boosting (XGBoost)

The Extreme Gradient Boosting is an improved supervised algorithm proposed by Chen and Guestrin [45] based on the Gradient Boosting Decision Tree algorithm [46]. XGBoost can be used to solve problems for regression and classification, which has been chosen to be used by data scientists because of its high execution speed and the high accuracy that it supplies [47]. The XGBoost objective function includes its loss function and regularization term, which can help to prevent overfitting by smoothing the final learned weights to obtain an optimal solution [48]. The loss function controls the ability of the prediction, which determines the deviation between predicted label and the actual label . The regularization term controls the complexity of the model and it can also handle the overfitting issue [48, 49]. XGBoost can also optimize the loss function using first-order and second-order gradient statistics. The objective function for XGBoost is defined as follows [49]:

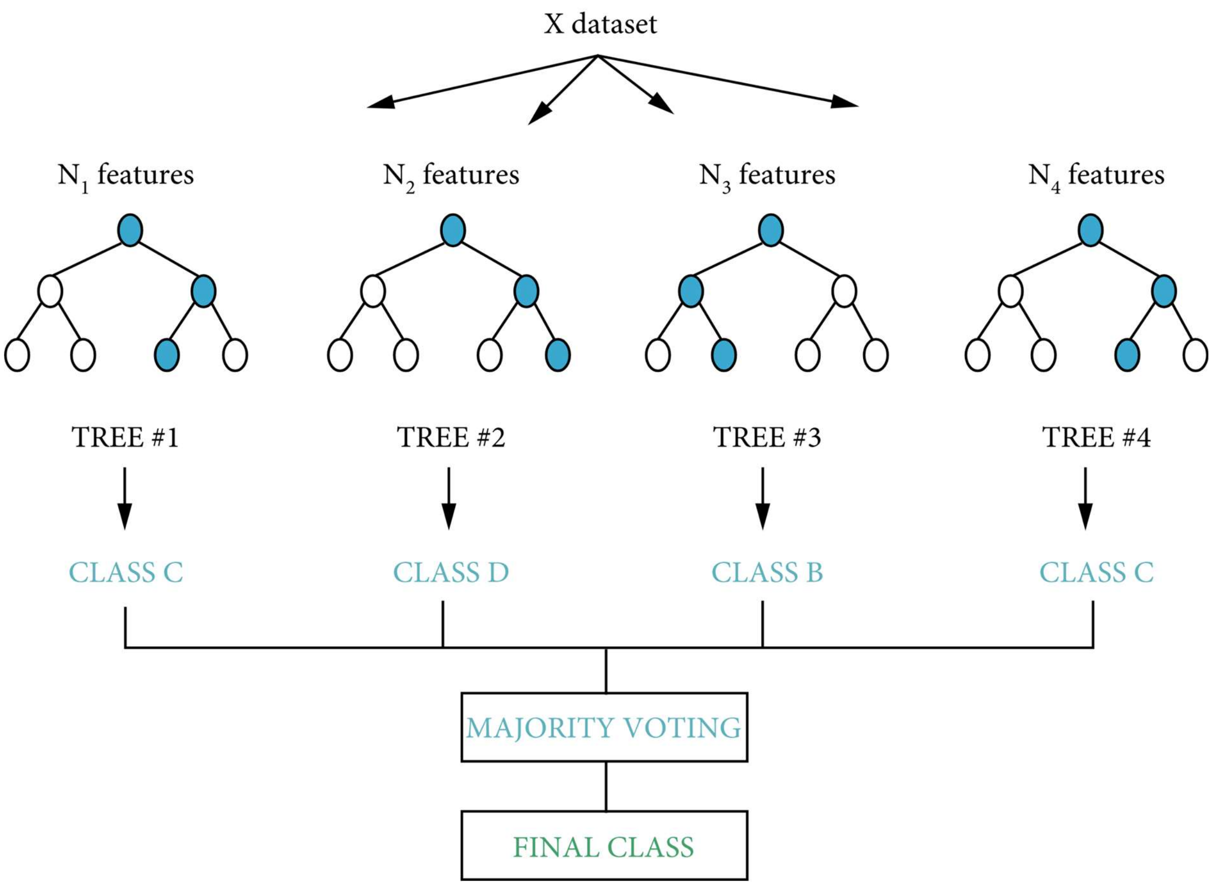
Decision Tree

DT is a nonparametric supervised learning algorithm for regression and classification tasks. DT (Figure 3) can be seen as a construction model that includes root node, division, and leaf node. Each internal node represents a test on an attribute, each division represents the outcome of test, and each leaf node grips the class label. The opening node in the tree is the root node. First, an attribute is selected and sited at the root node. Then, a division is made for each possible value. This splits dataset into subgroups, one for every value of the attribute. The tree process is recursively repeated for each division using only those cases that reach the branch. When all cases on a node have the same classification, the tree progress can be stopped. Usually, entropy or classification error is used to define the best tree division [51, 52].



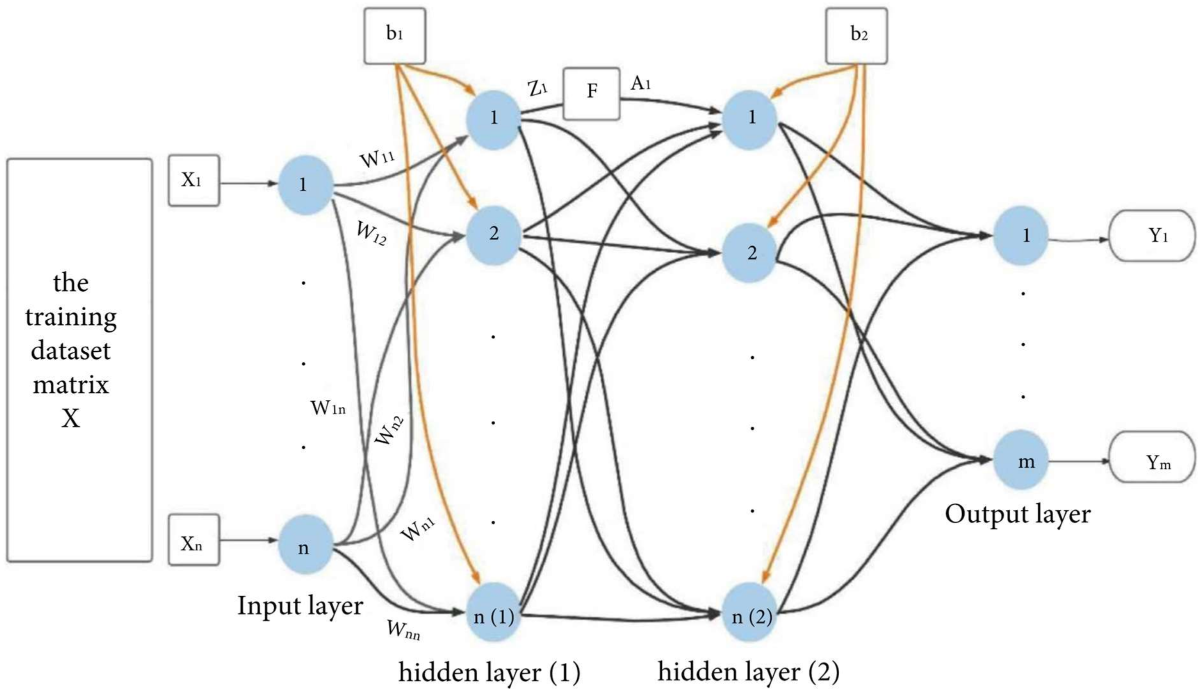
Random Forest

RF is one of the most common uses of classifier integration. As shown in Figure 4, RF is made up of numerous separate Decision Tree classifiers that vote on test samples according to a set of criteria [53, 54]. The steps are as follows:(i)Extracting some samples from the training set as a training subset using the bootstrap method, which is a self-help sampling approach.(ii)A number of features are randomly picked from the feature set for the training subset as the basis for splitting each node of the Decision Tree.(iii)Repeat steps (i)-(ii) to generate a large number of training subsets and Decision Trees, which are then combined to build a Random Forest.(iv)The test set's samples are fed into the Random Forest, where each Decision Tree makes a choice based on the data. After receiving the findings, the results are voted on using a voting technique to determine the sample categorization results.(v)Repeat step (iv) until all of the test sets have been classified [55].



Deep Neural Networks

The Deep Neural Networks (DNNs) are one of the architectures of Deep Learning [56]. DNNs have the same basic architecture as ANNs, with the exception that DNNs may have several hidden layers; that is why we use the term "deep." A Deep Neural Network can hold almost 150 hidden layers [1], and each layer can have several neurons as shown in Figure 5 and, in each layer of neurons, the input of a layer depends on the previous layer's output and so on until we get the prediction of our model in the output layer [57].



The workflow to build the end-to-end Machine Learning Project to predict Diabetes is as follows-

1.Collection of data

2.Exploring the data

3.Splitting the data

4.Training the model

5.Evaluating the model

6.Deploying the model

Objectives

1. Predict if person is diabetes patient or not
2. Find most indicative features of diabetes
3. Try different classification methods to find highest accuracy

Installing Libraries

In this first step I have imported most common libraries used in python for machine learning such as Pandas, Seaborn, Matplitlib etc.

I am using Python because if very flexible and effective programming language i ever used. I used Python in software development field too.

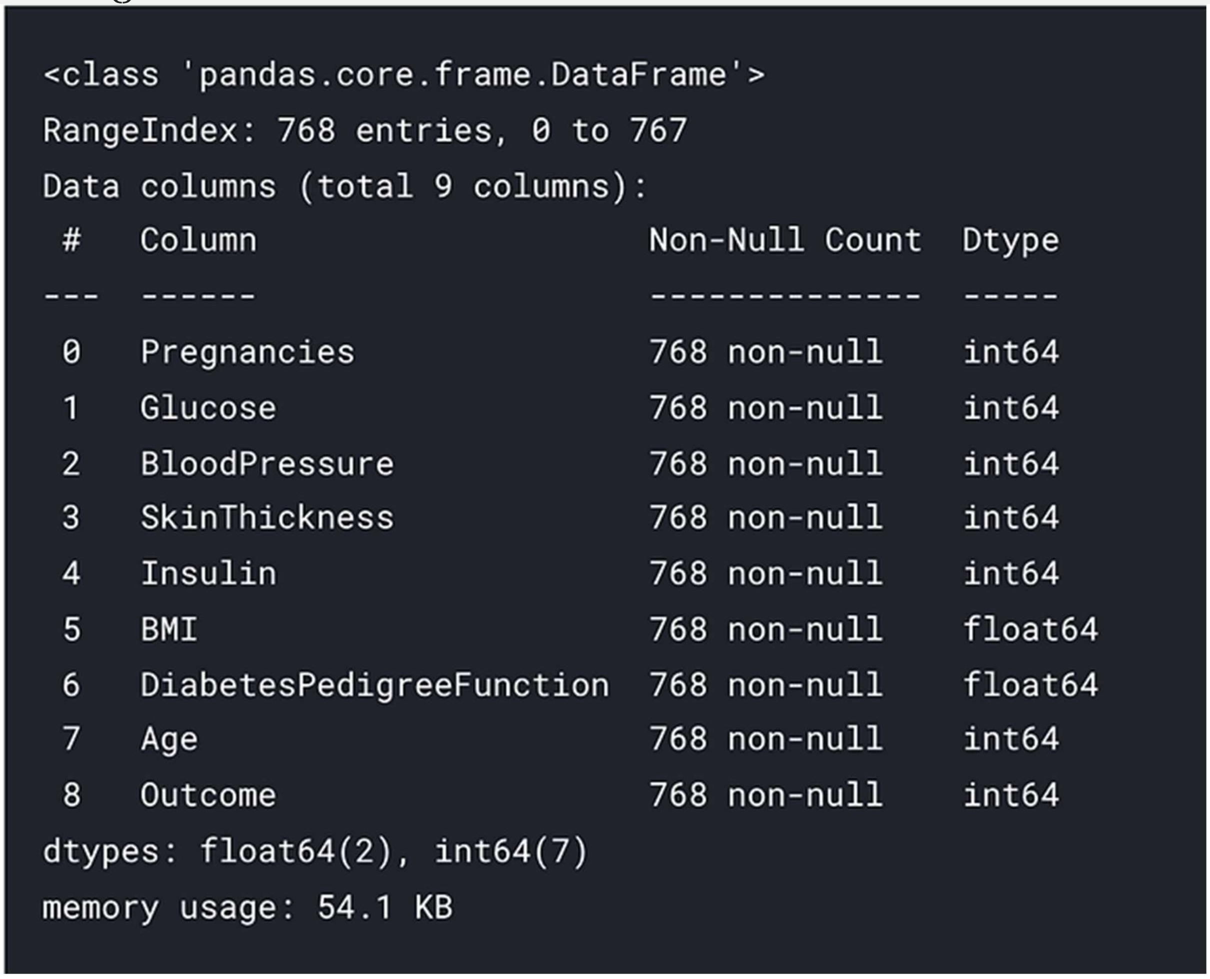
```
# Import libraries
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import seaborn as sns # for data visualization
import matplotlib.pyplot as plt # to plot charts
from collections import Counter
import os

# Modeling Libraries
from sklearn.preprocessing import QuantileTransformer
from sklearn.metrics import confusion_matrix, accuracy_score,
precision_score
from sklearn.ensemble import RandomForestClassifier,
AdaBoostClassifier, GradientBoostingClassifier, VotingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV, cross_val_score,
StratifiedKFold, learning_curve, train_test_split
```

The sklearn library is very versatile and handy and serves real-world purposes. It provides wide range of ML algorithms and Models.

Importing Data

In this project i used Pima Indians Diabetes Database from Kaggle. This dataset is originally from the National Institute of Diabetes and Digestive and Kidney Diseases.

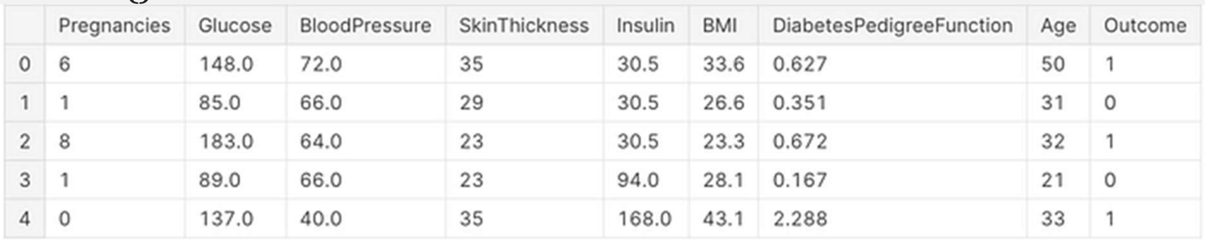
```
# Import dataset
df = pd.read_csv("../input/pima-indians-diabetes-
database/diabetes.csv")# Get familier with dataset structure
df.info()
```



```
<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
```

Excepting BMI and DiabetesPedigreeFunction all the columns are integer. Outcome is the label containing 1 and 0 values. 1 means person has diabetes and 0 mean person is not diabetic.

```
# Show top 5 rows
df.head()
```

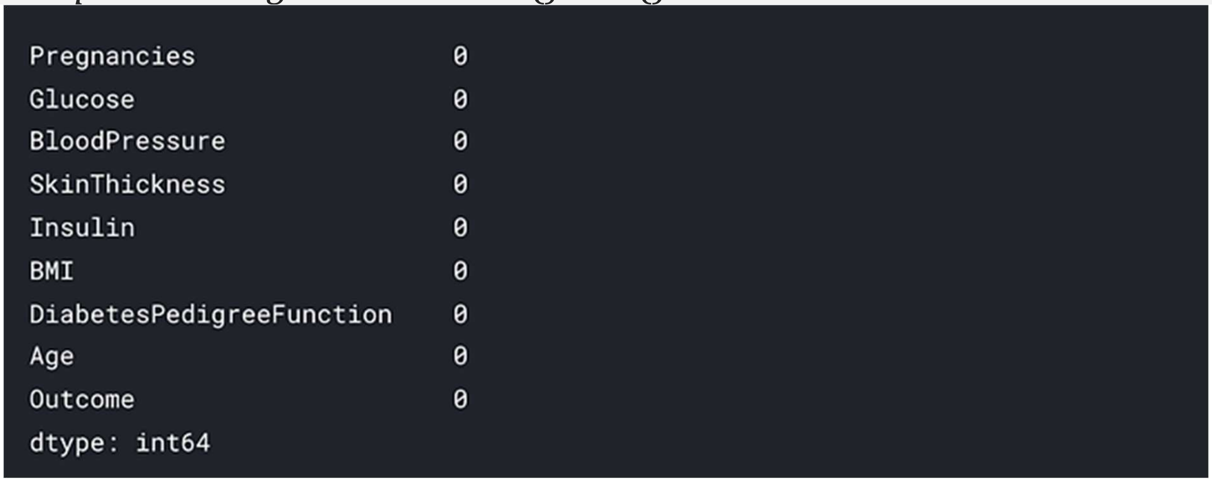


	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148.0	72.0	35	30.5	33.6	0.627	50	1
1	1	85.0	66.0	29	30.5	26.6	0.351	31	0
2	8	183.0	64.0	23	30.5	23.3	0.672	32	1
3	1	89.0	66.0	23	94.0	28.1	0.167	21	0
4	0	137.0	40.0	35	168.0	43.1	2.288	33	1

Missing Value Analysis

Next, i will cleanup the dataset which is the important part of data science. Missing data can lead to wrong statistics during modeling and predictions.

```
# Explore missing valuesdf.isnull().sum()
```



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

Missing Value Analysis

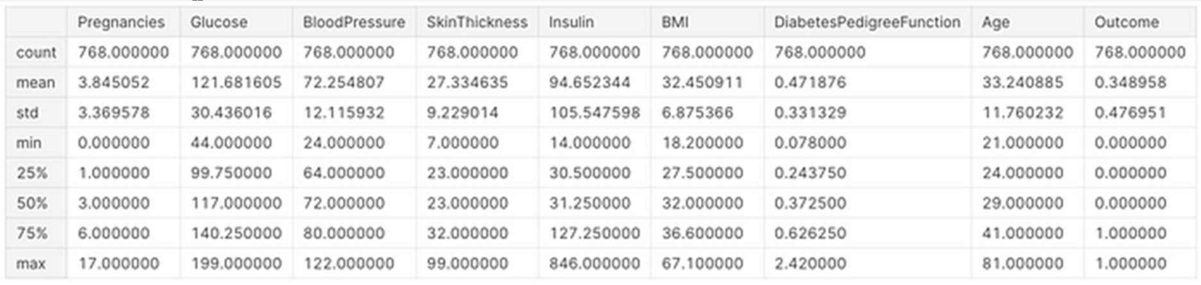
I observed that there is no missing values in dataset however the features like Glucose, BloodPressure, Insulin, SkinThickness has 0 values which is not possible. We have to replace 0 values with either mean or median values of specific column.

```
df['Glucose'] = df['Glucose'].replace(0, df['Glucose'].mean())# Correcting missing values in blood pressure
```

```
df['BloodPressure'] = df['BloodPressure'].replace(0,
df['BloodPressure'].mean()) # There are 35 records with 0
BloodPressure in dataset# Correcting missing values in BMI
df['BMI'] = df['BMI'].replace(0, df['BMI'].median())# Correct missing
values in Insulin and SkinThickness

df['SkinThickness'] = df['SkinThickness'].replace(0,
df['SkinThickness'].median())
df['Insulin'] = df['Insulin'].replace(0, df['Insulin'].median())
```

Now, lets review the dataset statistics

```
# Review dataset statistics
df.describe()
```



	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	121.681605	72.254807	27.334635	94.652344	32.450911	0.471876	33.240885	0.348958
std	3.369578	30.436016	12.115932	9.229014	105.547598	6.875366	0.331329	11.760232	0.476951
min	0.000000	44.000000	24.000000	7.000000	14.000000	18.200000	0.078000	21.000000	0.000000
25%	1.000000	99.750000	64.000000	23.000000	30.500000	27.500000	0.243750	24.000000	0.000000
50%	3.000000	117.000000	72.000000	23.000000	31.250000	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 i have clean dataset without missing values in features which is good.

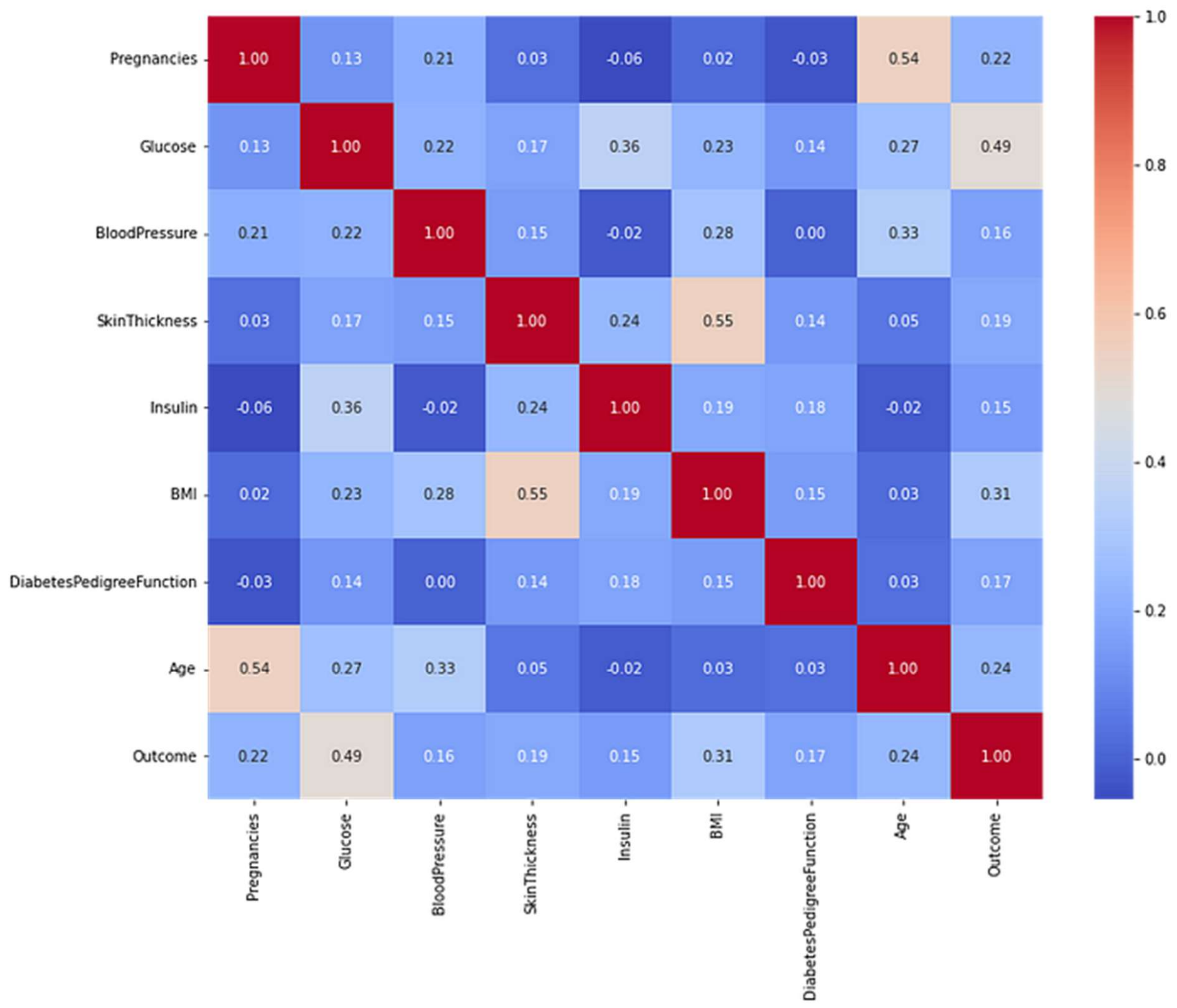
Exploratory Data Analysis

In this step, i showcased anlytics using GUI using Seaborn.

Correlation

Correlation is **one or more variables are related** to each other. It also helps to find the feature importance and clean the dataset before i start Modeling

```
plt.figure(figsize=(13,10))
sns.heatmap(df.corr(),annot=True, fmt = ".2f", cmap = "coolwarm")
```

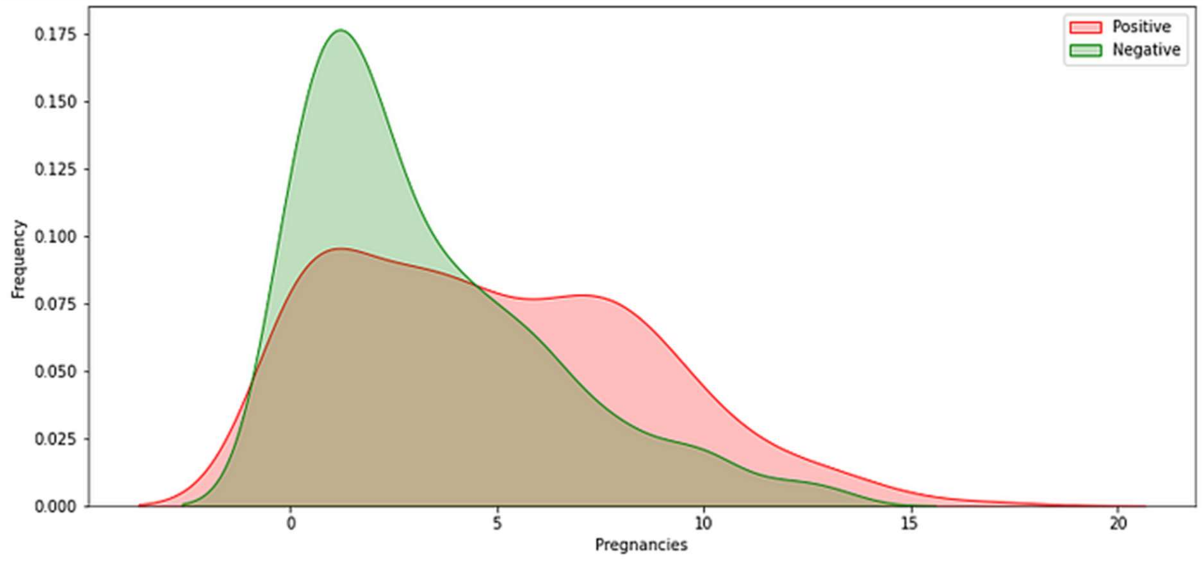


According to observation, features like Pregnancies, Gluecose, BMI, and Age is more correlated with Outcome. In next steps, i showcased details representation of these features.

Pregnancies

Women with diabetes can and do have healthy pregnancies and healthy babies. Managing diabetes can help reduce your risk for complications. Untreated diabetes increases your risk for pregnancy complications, like high blood pressure, depression, premature birth, birth defects and pregnancy loss.

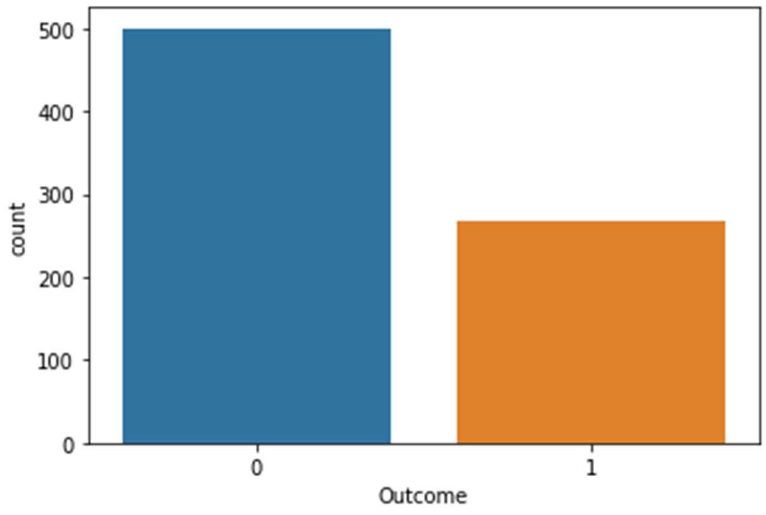
```
# Explore Pregnancies vs Outcomeplt.figure(figsize=(13,6))
g = sns.kdeplot(df["Pregnancies"][df["Outcome"] == 1],
    color="Red", shade = True)
g = sns.kdeplot(df["Pregnancies"][df["Outcome"] == 0],
    ax =g, color="Green", shade= True)g.set_xlabel("Pregnancies")
g.set_ylabel("Frequency")
g.legend(["Positive","Negative"])
```



Outcome

Outcome has 1 and 0 values where 1 indicates that person has diabetes and 0 shows person has no diabetes. This is my label column in dataset.

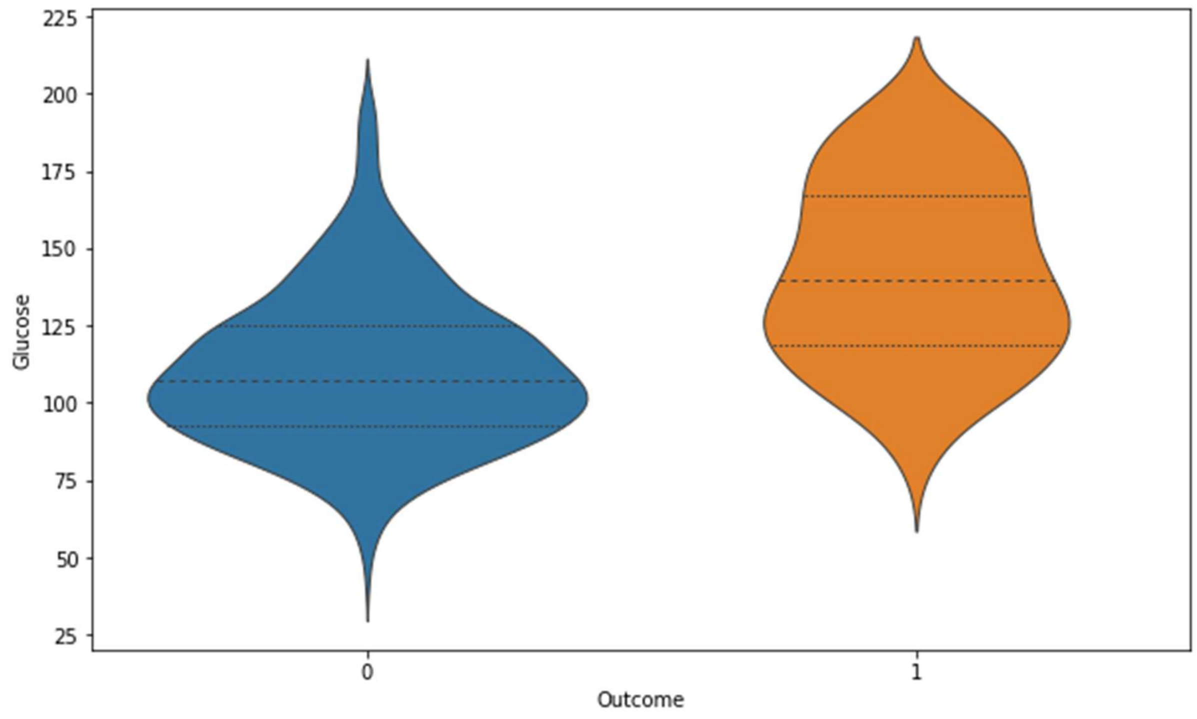
```
sns.countplot('Outcome', data = df)
```



It indicates, There are more people who do not have diabetes in dataset which is around 65% and 35% people has diabetes.

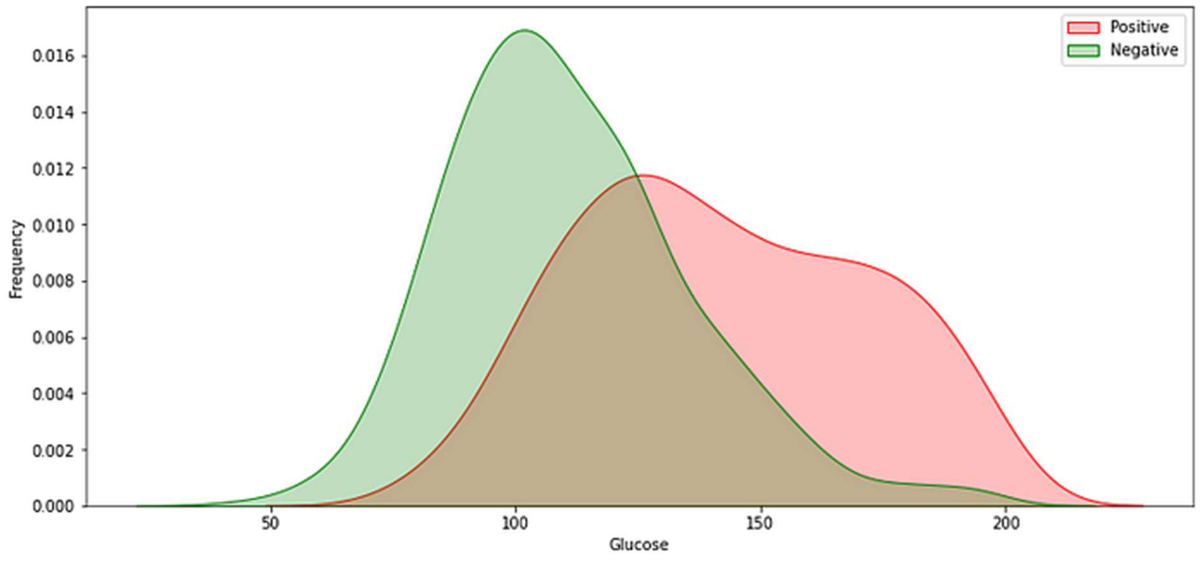
Glucose

```
# Explore Gluecose vs Outcomeplt.figure(figsize=(10,6))
sns.violinplot(data=df, x="Outcome", y="Glucose",
           split=True, inner="quart", linewidth=1)
```



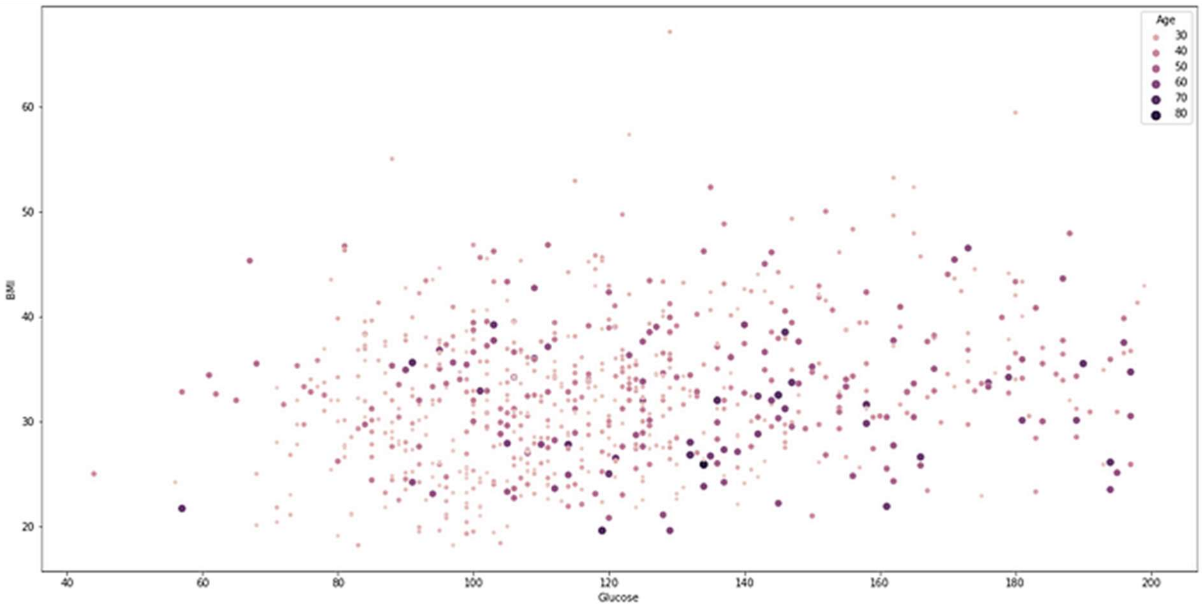
The chances of diabetes is gradually increasing with level of Glucose.

```
# Explore Glucose vs Outcome

plt.figure(figsize=(13,6))
g = sns.kdeplot(df["Glucose"][df["Outcome"] == 1], color="Red", shade =
True)
g = sns.kdeplot(df["Glucose"][df["Outcome"] == 0], ax =g, color="Green",
shade= True)
g.set_xlabel("Glucose")
g.set_ylabel("Frequency")
g.legend(["Positive","Negative"])
```



Explore Glucose vs BMI vs Age

```
# Glucose vs BMI vs Age

plt.figure(figsize=(20,10))
sns.scatterplot(data=df, x="Glucose", y="BMI", hue="Age", size="Age")
```



As per observation there are some outliers in features. We need to remove outliers in feature engineering.

Feature Engineering

Till now, i explored the dataset, did missing value corrections and data visualization. Next, i have started feature engineering. Feature engineering is useful to improve the performance of machine learning algorithms and is often considered as applied machine learning.

Selecting the important features and reducing the size of the feature set makes computation in machine learning and data analytic algorithms more feasible.

Outlier Detection

In this part i removed all the records outlined in dataset. Outliers impacts Model accuracy. I used *Tukey Method* used for outlier detection.

```
def detect_outliers(df,n,features):
    outlier_indices = []
    """
    Detect outliers from given list of features. It returns a list of the indices
    according to the observations containing more than n outliers according
    to the Tukey method
    """
    # iterate over features(columns)
    for col in features:
        Q1 = np.percentile(df[col], 25)
        Q3 = np.percentile(df[col],75)
        IQR = Q3 - Q1

        # outlier step
        outlier_step = 1.5 * IQR

        # Determine a list of indices of outliers for feature col
        outlier_list_col = df[(df[col] < Q1 - outlier_step) | (df[col] > Q3 +
outlier_step )].index
```

```
        # append the found outlier indices for col to the list of outlier indices
        outlier_indices.extend(outlier_list_col)

    # select observations containing more than 2 outliers
    outlier_indices = Counter(outlier_indices)
    multiple_outliers = list( k for k, v in outlier_indices.items() if v > n )

    return multiple_outliers

# detect outliers from numeric features
outliers_to_drop = detect_outliers(df, 2 ,["Pregnancies", 'Glucose',
'BloodPressure', 'BMI', 'DiabetesPedigreeFunction', 'SkinThickness',
'Insulin', 'Age'])
```

Here, I find outliers from all the features such as Pregnancies, Glucose, BloodPressure, BMI, DiabetesPedigreeFunction, SkinThickness, Insulin, and Age.

```
df.drop(df.loc[outliers_to_drop].index, inplace=True)
```

I have successfully removed all outliers from dataset now. The next step is to split the dataset in train and test and proceed the modeling.

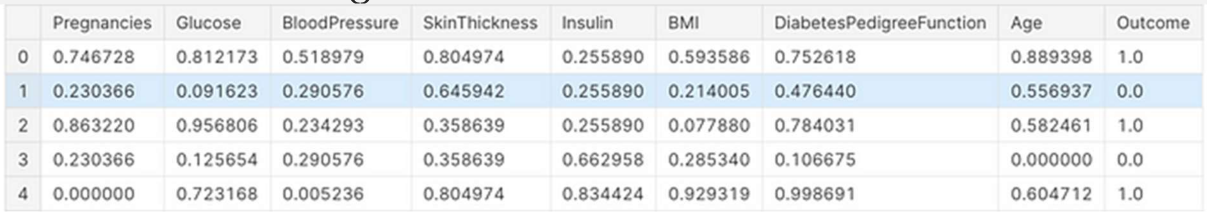
Modeling

In this sections, i tried different models and compare the accuracy for each. Then, i performed Hyperparameter Tuning on Models that has high accuracy.

Before i split the dataset i need to transform the data into quantile using `sklearn.preprocessing` .

```
# Data Transformation
q  = QuantileTransformer()
X = q.fit_transform(df)
transformedDF = q.transform(X)
transformedDF = pd.DataFrame(X)
```

```
transformedDF.columns =['Pregnancies', 'Glucose', 'BloodPressure',
'SkinThickness', 'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age',
'Outcome']# Show top 5 rows
transformedDF.head()
```



	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	0.746728	0.812173	0.518979	0.804974	0.255890	0.593586	0.752618	0.889398	1.0
1	0.230366	0.091623	0.290576	0.645942	0.255890	0.214005	0.476440	0.556937	0.0
2	0.863220	0.956806	0.234293	0.358639	0.255890	0.077880	0.784031	0.582461	1.0
3	0.230366	0.125654	0.290576	0.358639	0.662958	0.285340	0.106675	0.000000	0.0
4	0.000000	0.723168	0.005236	0.804974	0.834424	0.929319	0.998691	0.604712	1.0

Data Transformation

Data Splitting

Next, i split data in test and train dataset. Train dataset will be used in Model training and evaluation and test dataset will be used in prediction. Before i predict the test data, i performed cross validation for various models.

```
features = df.drop(["Outcome"], axis=1)
labels = df["Outcome"]x_train, x_test, y_train, y_test =
train_test_split(features, labels, test_size=0.30, random_state=7)
```

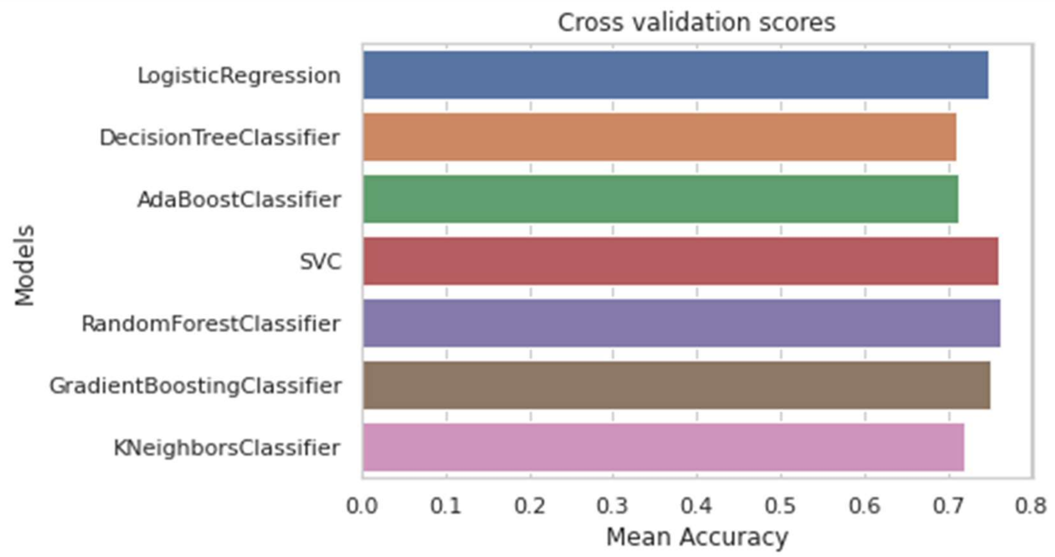
Above code splits dataset into train (70%) and test (30%) dataset.

Cross Validate Models

```
def evaluate_model(models):
    """
    Takes a list of models and returns chart of cross validation scores using
mean accuracy
    """

    # Cross validate model with Kfold stratified cross val
    kfold = StratifiedKFold(n_splits = 10)

    result = []
    for model in models :
        result.append(cross_val_score(estimator = model, X = x_train, y =
y_train, scoring = "accuracy", cv = kfold, n_jobs=4))

   cv_means = []
   cv_std = []
   for cv_result in result:
      cv_means.append(cv_result.mean())
      cv_std.append(cv_result.std())

   result_df = pd.DataFrame({
      "CrossValMeans":cv_means,
      "CrossValerrors": cv_std,
      "Models":[
         "LogisticRegression",
         "DecisionTreeClassifier",
         "AdaBoostClassifier",
         "SVC",
         "RandomForestClassifier",
         "GradientBoostingClassifier",
         "KNeighborsClassifier"
      ]
   })

   # Generate chart
   bar = sns.barplot(x = "CrossValMeans", y = "Models", data = result_df,
orient = "h")
   bar.set_xlabel("Mean Accuracy")
   bar.set_title("Cross validation scores")
   return result_df
```

Method `evaluate_model` takes a list of models and returns chart of cross validation scores using mean accuracy.

```
# Modeling step Test differents algorithms
random_state = 30
models = [
   LogisticRegression(random_state = random_state, solver='liblinear'),
   DecisionTreeClassifier(random_state = random_state),
   AdaBoostClassifier(DecisionTreeClassifier(random_state =
random_state), random_state = random_state, learning_rate = 0.2),
   SVC(random_state = random_state),
   RandomForestClassifier(random_state = random_state),
   GradientBoostingClassifier(random_state = random_state),
   KNeighborsClassifier(),
]
evaluate_model(models)
```



Cross Validate Models

As per above observation, i found that SVC, RandomForestClassifier, and LogisticRegression model has more accuracy. Next, i will do hyper parameter tuning on three models.

Hyperparameter Tuning

Hyperparameter tuning is choosing a set of optimal hyperparameters for a learning algorithm. A hyperparameter is a model argument whose value is set before the learning process begins. The key to machine learning algorithms is hyperparameter tuning.

I have done tuning process for SVC, RandomForestClassifier, and LogisticRegression models one by one.

```
# Import libraries
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_reportdef
analyze_grid_result(grid_result):
    '''
    Analysis of GridCV result and predicting with test dataset
    Show classification report at last
    '''    # Best parameters and accuracy
    print("Tuned hyperparameters: (best parameters) ",
grid_result.best_params_)
    print("Accuracy :", grid_result.best_score_)

    means = grid_result.cv_results_["mean_test_score"]
    stds = grid_result.cv_results_["std_test_score"]
    for mean, std, params in zip(means, stds,
grid_result.cv_results_["params"]):
        print("%0.3f (+/-%0.03f) for %r" % (mean, std * 2, params))
    print()    print("Detailed classification report:")
    y_true, y_pred = y_test, grid_result.predict(x_test)
    print(classification_report(y_true, y_pred))
    print()
```

First of all i have imported GridSearchCV and classification_report from sklearn library. Then, i have defined `analyze_grid_result` method which will show prediction result. I called this method for each Model used in SearchCV. In next step, i will perform tuning for each model.

LogisticRegression

```
# Define models and parameters for LogisticRegression
model = LogisticRegression(solver='liblinear')
solvers = ['newton-cg', 'liblinear']
penalty = ['l2']
c_values = [100, 10, 1.0, 0.1, 0.01]# Define grid search
grid = dict(solver = solvers, penalty = penalty, C = c_values)
cv = StratifiedKFold(n_splits = 50, random_state = 1, shuffle = True)
grid_search = GridSearchCV(estimator = model, param_grid = grid, cv =
cv, scoring = 'accuracy', error_score = 0)
```

```
logi_result = grid_search.fit(x_train, y_train)# Logistic Regression
Hyperparameter Result
analyze_grid_result(logi_result)
```

Output:

```
Tuned hyperparameters: (best parameters)  {'C': 10, 'penalty': 'l2',
'solver': 'liblinear'}
Accuracy : 0.7883636363636363
Detailed classification report:

           precision    recall  f1-score   support

         0      0.78      0.84      0.81       147
         1      0.68      0.58      0.62        83

   accuracy                          0.75       230
  macro avg       0.73      0.71      0.72       230
weighted avg      0.74      0.75      0.74       230
```

As per my observation, in LogisticRegression it returned best score 0.78 with `*{'C': 10, 'penalty': 'l2', 'solver': 'liblinear'}*` parameters. Next i will perform tuning for other models.

SVC

```
# Define models and parameters for LogisticRegression
model = SVC()# Define grid search
tuned_parameters = [
   {"kernel": ["rbf"], "gamma": [1e-3, 1e-4], "C": [1, 10, 100, 1000]},
   {"kernel": ["linear"], "C": [1, 10, 100, 1000]},
]
cv = StratifiedKFold(n_splits = 2, random_state = 1, shuffle = True)
grid_search = GridSearchCV(estimator = model, param_grid =
tuned_parameters, cv = cv, scoring = 'accuracy', error_score = 0)
scv_result = grid_search.fit(x_train, y_train)# SVC Hyperparameter Result
analyze_grid_result(scv_result)
```

Output:

```
Tuned hyperparameters: (best parameters)  {'C': 10, 'kernel': 'linear'}
Accuracy : 0.7775797976410084
Detailed classification report:

           precision    recall  f1-score   support

        0       0.78      0.84      0.81       147
        1       0.67      0.57      0.61        83

   accuracy                           0.74       230
  macro avg       0.72      0.70      0.71       230
weighted avg       0.74      0.74      0.74       230
```

SVC Model gave max 0.77 accuracy which is bit less than LogisticRegression. I will not use this model anymore.

RandomForestClassifier

```
# Define models and parameters for LogisticRegression
model = RandomForestClassifier(random_state=42)# Define grid search
tuned_parameters = {
   'n_estimators': [200, 500],
   'max_features': ['auto', 'sqrt', 'log2'],
   'max_depth' : [4,5,6,7,8],
   'criterion' :['gini', 'entropy']
}
cv = StratifiedKFold(n_splits = 2, random_state = 1, shuffle = True)
grid_search = GridSearchCV(estimator = model, param_grid =
tuned_parameters, cv = cv, scoring = 'accuracy', error_score = 0)
grid_result = grid_search.fit(x_train, y_train)# SVC Hyperparameter
Result
analyze_grid_result(grid_result)
```

Output:

```
Tuned hyperparameters: (best parameters)  {'criterion': 'entropy',
'max_depth': 5, 'max_features': 'log2', 'n_estimators': 200}
Accuracy : 0.7663648051875454
Detailed classification report:

          precision    recall  f1-score   support

        0       0.78      0.83      0.80       147
        1       0.66      0.58      0.62        83

   accuracy                           0.74       230
  macro avg       0.72      0.70      0.71       230
weighted avg       0.73      0.74      0.74       230
```

Randomforest model gave max 0.76% accuracy which is not best comparing to other model. So i decided to use LogisticRegression Model for prediction.

Prediction

Till now, i worked on EDA, Feature Engineering, Cross Validation of Models, and Hyperparameter Tuning and find the best working Model for my dataset. Next, I did prediction from my test dataset and storing the result in CSV.

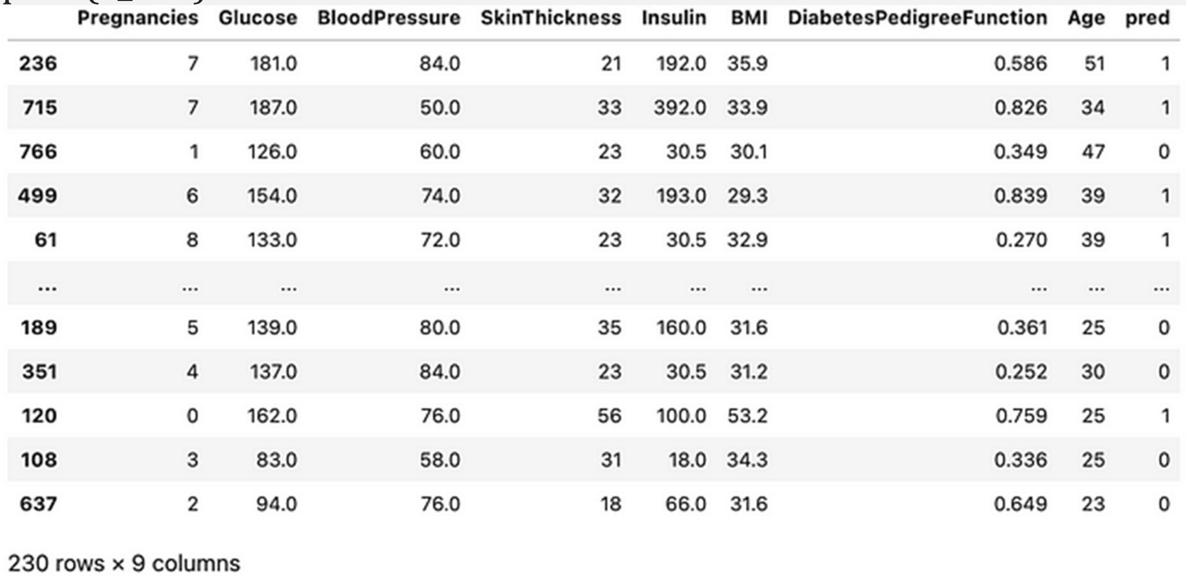
```
# Test predictions
y_pred = logi_result.predict(x_test)
print(classification_report(y_test, y_pred))
# output
          precision    recall  f1-score   support

        0       0.78      0.84      0.81       147
        1       0.68      0.58      0.62        83
```

```
   accuracy                     0.75      230
  macro avg      0.73     0.71     0.72      230
weighted avg      0.74     0.75     0.74      230
```

Finally append new feature column in test dataset

called Prediction and print the dataset.

```
x_test['pred'] = y_pred
print(x_test)
```



	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	pred
236	7	181.0	84.0	21	192.0	35.9	0.586	51	1
715	7	187.0	50.0	33	392.0	33.9	0.826	34	1
766	1	126.0	60.0	23	30.5	30.1	0.349	47	0
499	6	154.0	74.0	32	193.0	29.3	0.839	39	1
61	8	133.0	72.0	23	30.5	32.9	0.270	39	1
...	...	...	...	...	...	...	...	...	...
189	5	139.0	80.0	35	160.0	31.6	0.361	25	0
351	4	137.0	84.0	23	30.5	31.2	0.252	30	0
120	0	162.0	76.0	56	100.0	53.2	0.759	25	1
108	3	83.0	58.0	31	18.0	34.3	0.336	25	0
637	2	94.0	76.0	18	66.0	31.6	0.649	23	0

230 rows × 9 columns