CONTENT

1 Introduction

Diabetes is a chronic illness, with the problem of metabolizing blood glucose in the cells due to lack of insulin production. With the bombing number of people with diabetes worldwide, diabetes is regarded as a 21st century challenge(Zimmet,2014). It was listed at the Un High-level Political Meeting in 2011.(Roglic,2016) If some risk factors could be used to predict susceptibility to diabetes effectively, this would help people prevent it and make a considerable difference in clinical practice.

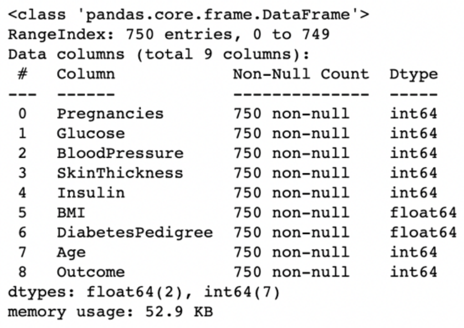
We are going to work on a Pima Indians Diabetes Database. This dataset is originally from the National Institute of Diabetes and Digestive and Kidney Diseases and can be downloaded as csv file. The outcome of the dataset is whether or not a woman has diabetes, and data was collection from the female of Pima Indian heritage. It contains eight risk factors:

- Pregnancies: number of times the woman has been pregnant
- Glucose: plasma glucose concentration (mg/dl) at 2 hours in an oral glucose tolerance test (OGTT)
- Blood Pressure: Diastolic blood pressure (mm Hg)
- Skin Thickness: Triceps skin fold thickness (mm)
- Serum Insulin: insulin concentration^2 (μ U/ml) at 2 hours in an OGTT BMI: body mass index (weight in kg)/(height in m)^2
- Age: in years(least is 21)
- Diabete Pedigree: a numerical score designed tomeasure the genetic influence of both the woman's diabetic and her non-diabetic relatives on diabetes risk: higher scores mean higher risk. You can read more about this in Smith, Everhart, Dickson, Knowler, and Johannes (1988)
- Outcome: 1 if the woman eventually tested positive for diabetes, zero otherwise

The form of the dataset is csv file, which may correlate to high accuracy, but not reliably so. It is a small dataset, recording 750 rows in total. In Insulin and Skin Thickness column, it contains more than 20 percent of missing value. High percentage level of missing value may distort prediction results. Using David Spiegelhalter's rating scale for data quality, the Pima dataset would be placed at 3*. Numbers in dataset are reasonable accurate and are from clinical practice. We'll do data exploratory in next chapter.

2 Data Cleaning and Exploratory Data Analysis

```python
#Import library
# Basic numerics
import numpy as np
import scipy as sp
import scipy.stats as st
# Data handling
import pandas as pd
# Graphics
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_style("whitegrid", {'axes.grid' : False})
# Statistical tools
import statsmodels.api as sm
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, roc_auc_score, classification_report, plot_roc_curve,
confusion_matrix
from sklearn.model_selection import train_test_split, cross_validate
from sklearn.metrics import accuracy_score

#Input dataset
diabetes = pd.read_csv('PimaDiabetes.csv')
#Get data information
diabetes.info()
```



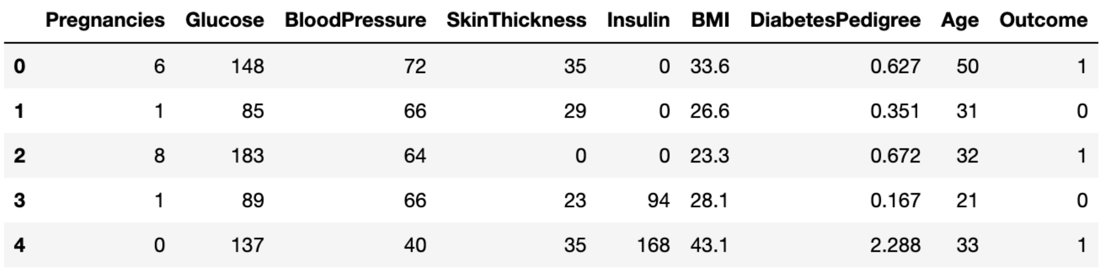
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 750 entries, 0 to 749
Data columns (total 9 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   Pregnancies       750 non-null    int64  
 1   Glucose           750 non-null    int64  
 2   BloodPressure     750 non-null    int64  
 3   SkinThickness     750 non-null    int64  
 4   Insulin           750 non-null    int64  
 5   BMI               750 non-null    float64
 6   DiabetesPedigree  750 non-null    float64
 7   Age               750 non-null    int64  
 8   Outcome           750 non-null    int64  
dtypes: float64(2), int64(7)
memory usage: 52.9 KB
```

Figure 1. information of the raw database

We found there is no null value in this data set. Then we looked details in each line.

```
#Get the first five lines of dataset
diabetes.head()
```

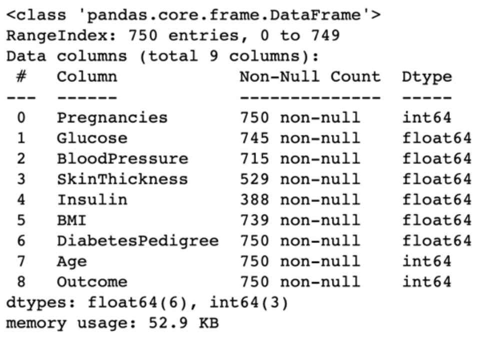
Table 1. the first five lines of raw diabetes dataset



	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigree	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

Apparently, there are lots of 0 in the data where they are not supposed to be. (We do not expect 0 except column 'Pregnancies' and 'Outcome')

```
#Replace the zero to None, it could be useful to count how much zero data in each column
#There's possible to have no pregnancy, so keep the Pregnancies and Outcome column unchanged
diabetes[['Glucose','BloodPressure','SkinThickness','Insulin','BMI','DiabetesPedigree','Age']] =
diabetes[['Glucose','BloodPressure','SkinThickness','Insulin','BMI','DiabetesPedigree','Age']].replace(0,np.NaN)
```



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 750 entries, 0 to 749
Data columns (total 9 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   Pregnancies       750 non-null    int64  
 1   Glucose           745 non-null    float64
 2   BloodPressure     715 non-null    float64
 3   SkinThickness     529 non-null    float64
 4   Insulin           388 non-null    float64
 5   BMI               739 non-null    float64
 6   DiabetesPedigree  750 non-null    float64
 7   Age               750 non-null    int64  
 8   Outcome           750 non-null    int64  
dtypes: float64(6), int64(3)
memory usage: 52.9 KB
```

Figure 2. information of the database replacing 0 to null

The non-null count in ‘Glucose’, ‘BloodPressure’, ’SkinThickness’, ‘Insulin’ and ‘BMI’ columns are less than 750. We can transfer it to a more intuitive way.

```
#Get how many missing values in each column
diabetes.info()
```

```
Pregnancies            0
Glucose                5
BloodPressure         35
SkinThickness        221
Insulin              362
BMI                   11
DiabetesPedigree       0
Age                    0
Outcome                0
dtype: int64
```

Figure 3. null value in each column

```
#Get the missing value percentage for each column and generate a new DF
missing_count = pd.DataFrame(diabetes.isnull().sum()/750*100).round(2)
missing_count.columns = ['Percentage']
missing_count
```

Table 2. the percentage of missing value in each column

	Percentage
Pregnancies	0.00
Glucose	0.67
BloodPressure	4.67
SkinThickness	29.47
Insulin	48.27
BMI	1.47
DiabetesPedigree	0.00
Age	0.00
Outcome	0.00

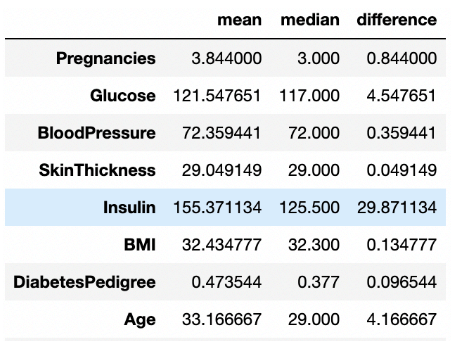
The missing value proportion in ‘Insulin’ and ‘SkinThickness’ column is greater than 15%, which would be considered a large amount and may severely impact further interpretation. (Acuna E,2004)

A possible reason for ‘Insulin’ and ‘SkinThickness’ could be these two variables are not as intuitive as other variables like ‘Pregnancies’ and ‘Age’. A person without diabetes may not voluntarily test his insulin levels, the type of the missing value in natural science is Missing completely at random. (Kwak, 2017)

When dealing with a big dataset, we can simply remove rows containing the missing value. The PimaDiabetes database only contains 750 rows. Then we should replace the missing value with the appropriate value instead. Mean or median is widely used. (Acuña,2004)

```
#Get mean and median for each column
mean_and_median = pd.DataFrame(diabetes.mean())
mean_and_median.columns = ['mean']
mean_and_median['median'] = pd.DataFrame(diabetes.median())
mean_and_median['difference'] = pd.DataFrame(mean_and_median['mean'] - mean_and_median['median'])
mean_and_median
```

Table 3. mean and median values for each variable

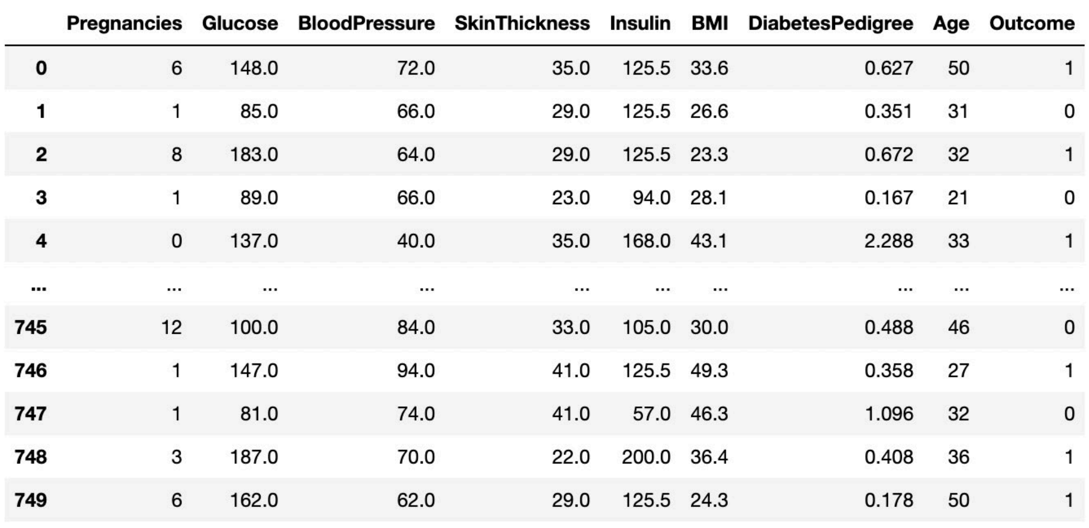


	mean	median	difference
Pregnancies	3.844000	3.000	0.844000
Glucose	121.547651	117.000	4.547651
BloodPressure	72.359441	72.000	0.359441
SkinThickness	29.049149	29.000	0.049149
Insulin	155.371134	125.500	29.871134
BMI	32.434777	32.300	0.134777
DiabetesPedigree	0.473544	0.377	0.096544
Age	33.166667	29.000	4.166667

The differences between mean and median value for each variable is negligible, except for 'Insulin' variable. A cutoff point of 2-hour postload insulin to classify IGT vs. normal is 140 mg/dL. (Ou,2017) We then chose the median one for Insuline.

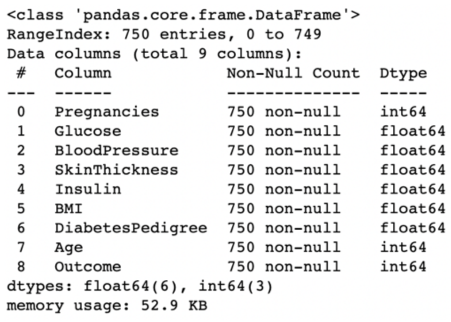
```
#Replace missing value using median value
diabetes['Glucose'].fillna(mean_and_median['median']['Glucose'],inplace=True)
diabetes['BloodPressure'].fillna(mean_and_median['median']['BloodPressure'],inplace=True)
diabetes['SkinThickness'].fillna(mean_and_median['median']['SkinThickness'],inplace=True)
diabetes['Insulin'].fillna(mean_and_median['median']['Insulin'],inplace=True)
diabetes['BMI'].fillna(mean_and_median['median']['BMI'],inplace=True)
diabetes
```

Table 4. new dataset after data-replacing



	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigree	Age	Outcome
0	6	148.0	72.0	35.0	125.5	33.6	0.627	50	1
1	1	85.0	66.0	29.0	125.5	26.6	0.351	31	0
2	8	183.0	64.0	29.0	125.5	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
...	...	...	...	...	...	...	...	...	...
745	12	100.0	84.0	33.0	105.0	30.0	0.488	46	0
746	1	147.0	94.0	41.0	125.5	49.3	0.358	27	1
747	1	81.0	74.0	41.0	57.0	46.3	1.096	32	0
748	3	187.0	70.0	22.0	200.0	36.4	0.408	36	1
749	6	162.0	62.0	29.0	125.5	24.3	0.178	50	1

```
#New dataset infomation
diabetes.info()
```



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 750 entries, 0 to 749
Data columns (total 9 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   Pregnancies       750 non-null    int64  
 1   Glucose           750 non-null    float64
 2   BloodPressure     750 non-null    float64
 3   SkinThickness     750 non-null    float64
 4   Insulin           750 non-null    float64
 5   BMI               750 non-null    float64
 6   DiabetesPedigree  750 non-null    float64
 7   Age               750 non-null    int64  
 8   Outcome           750 non-null    int64  
dtypes: float64(6), int64(3)
memory usage: 52.9 KB
```

Figure 4. information of the dataset after data replacing

After getting a clean dataset, we'll do some exploratory work to find links via variables and the outcome.

```
#Histagram for the new dataset
p = diabetes.hist(figsize = (20,20))
```

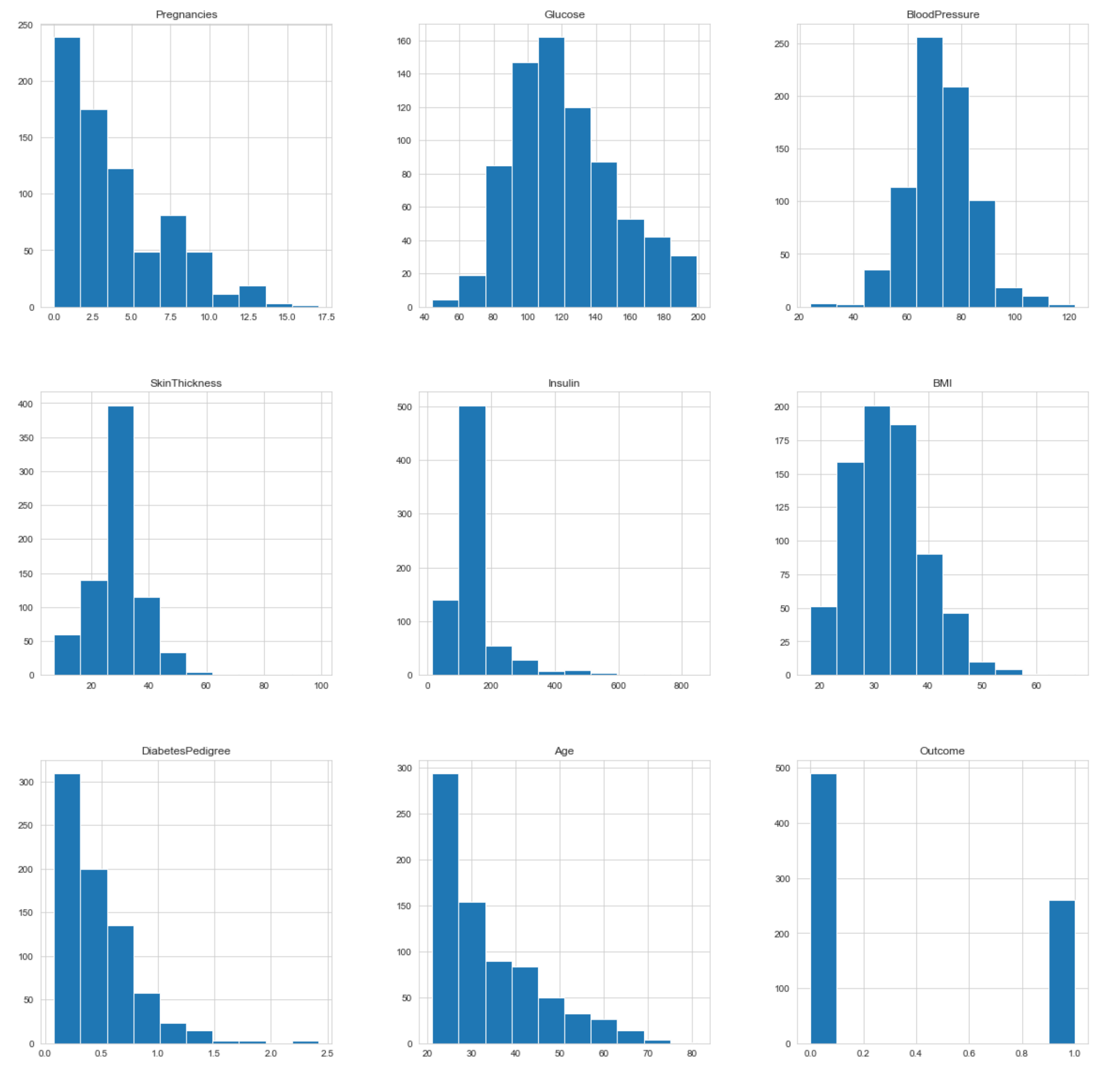


Figure 5. histogram for each variable in the dataset after cleaning

'Pregnancies', 'SkinThickness', 'Insulin', 'BMI', 'DiabetesPedigree' and 'Age' are left-skewed distribution.

```
#Plot heatmap
plt.figure(figsize=(13,10))
sns.heatmap(diabetes.corr(),annot=True, fmt = ".5f", linewidth=.5, cmap = "BuPu").set(title='Correlation Heatmap')
```

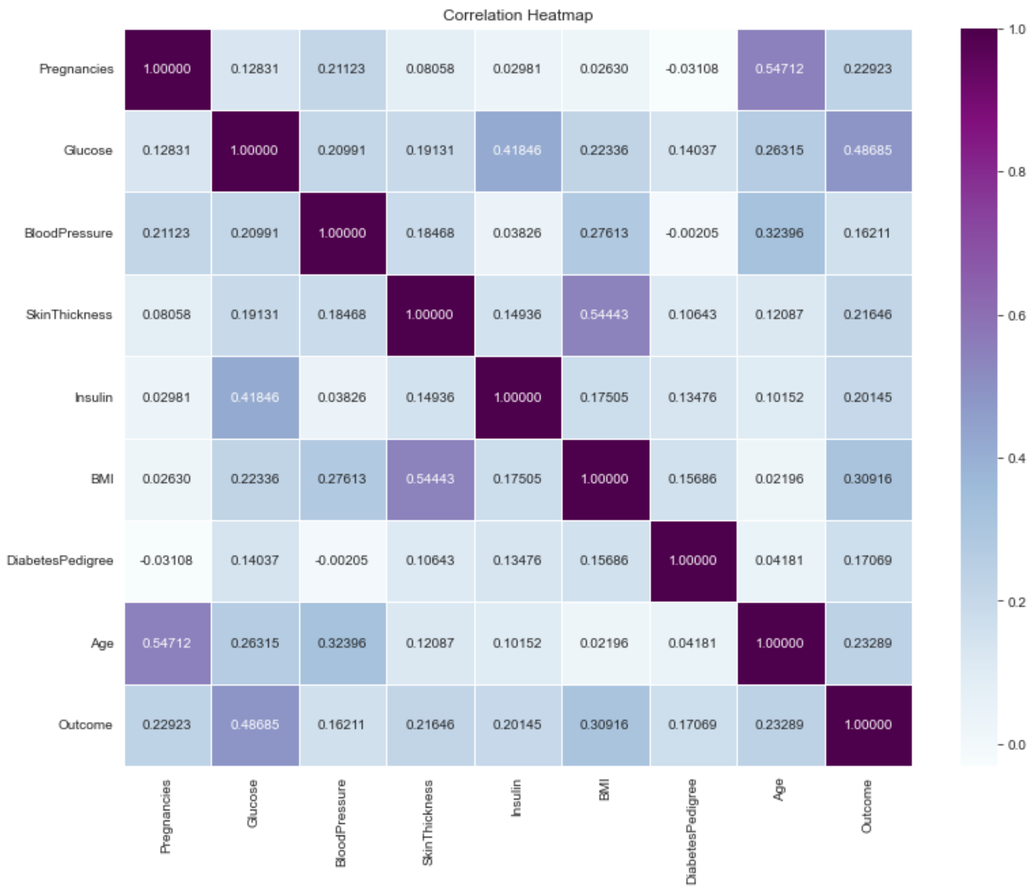


Figure 6. heatmap for the dataset after cleaning

According to the heatmap, we found the glucose, BMI, age, and pregnancies are high correlated to the outcome in this dataset.

```
#Number of diabetes and without diabetes in this dataset
diabetes['Outcome'].value_counts(sort=False).plot.bar(figsize=(12,7))
plt.text(x=-0.05,y=260,s='Diabetes',fontsize=14)
plt.text(x=-0.03,y=220,s='260',fontsize=20)
plt.text(x=0.88,y=490,s='Without Diabetes',fontsize=14)
plt.text(x=0.98,y=460,s='490',fontsize=20)
plt.title('Diabetes Condition',fontsize=18)
```

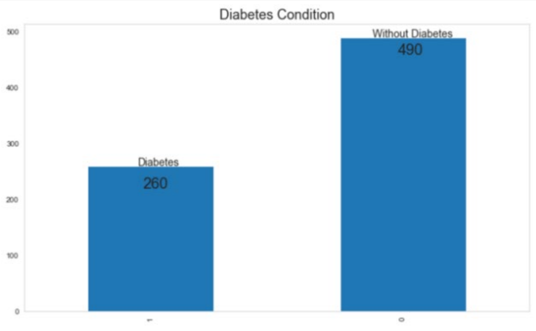


Figure 7. diabetes and un-diabetes proportion in the dataset after cleaning

The dataset contains 260 records of diabetes, and 490 records of non-diabetes.

```
#Create pair plot
pl = sns.pairplot(diabetes, hue='Outcome')
```

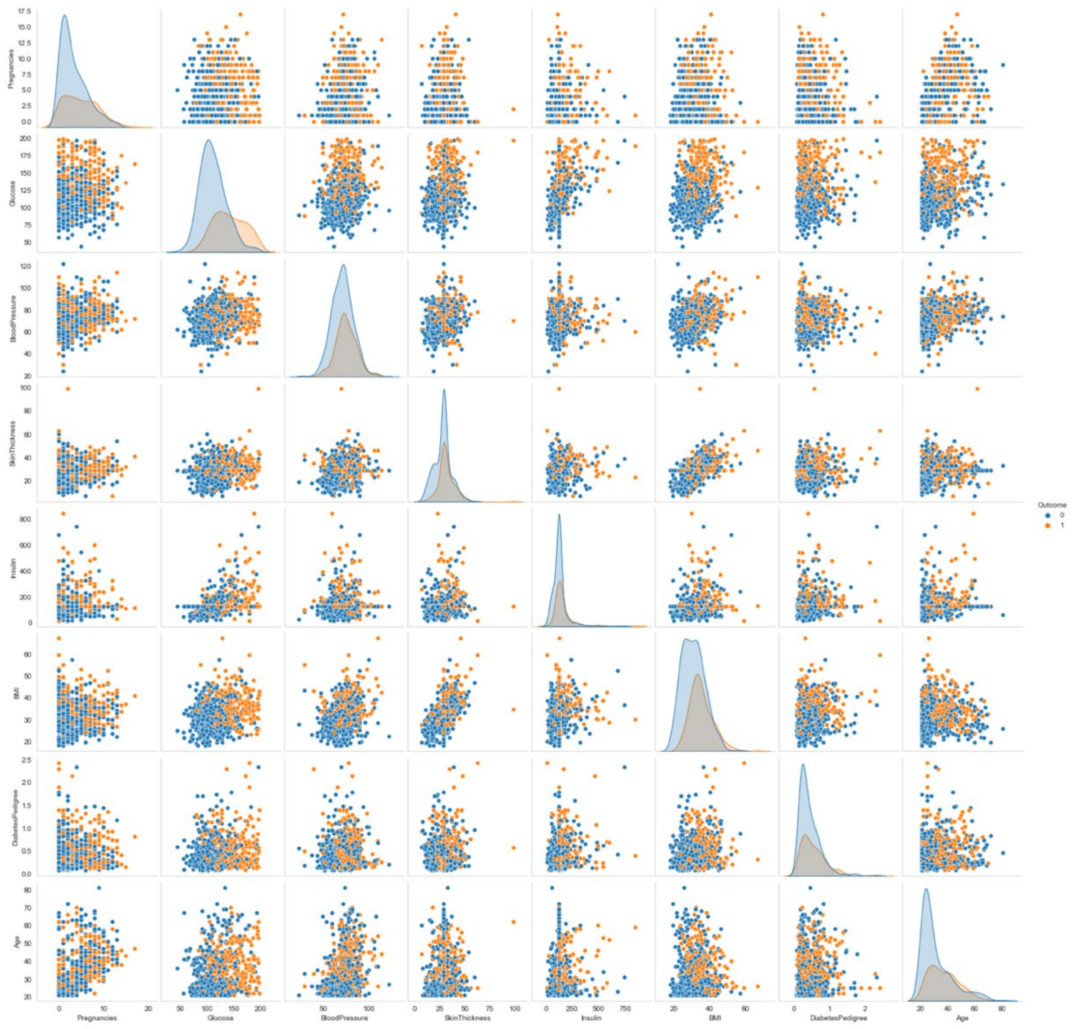


Figure 8. pair plot for the dataset after cleaning

BMI and SkinThickness may have a liner relationship, but these two variables have low correlation with the outcome.

```
# Explore Pregnancies vs Outcome
plt.figure(figsize=(13,6))
g = sns.kdeplot(diabetes["Pregnancies"][diabetes["Outcome"] == 1],
        color="Blue", shade = True)
g = sns.kdeplot(diabetes["Pregnancies"][diabetes["Outcome"] == 0],
        ax =g, color="Grey", shade= True)
g.set_xlabel("Pregnancies",fontsize=13)
g.set_ylabel("Frequency",fontsize=13)
g.legend(["Positive","Negative"])
plt.title('Relationship between Pregnancies and Outcome',fontsize=16)
plt.show()
```

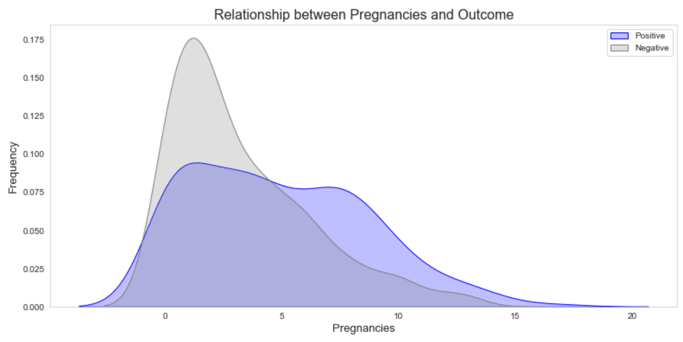


Figure 9. relationship between pregnancies and outcome

The main number of pregnancies for women without diabetes is between 2 and 3, while a relatively large proportion of women with diabetes have had more than 5 pregnancies. The number of pregnancies may be an important influencing factor.

```
import plotly.express as px
fig = px.box(diabetes, x="Outcome", y="Pregnancies")
fig.update_layout(title_text='Boxplot between Pregnancies and Outcome', title_x=0.5)
fig.show()
```

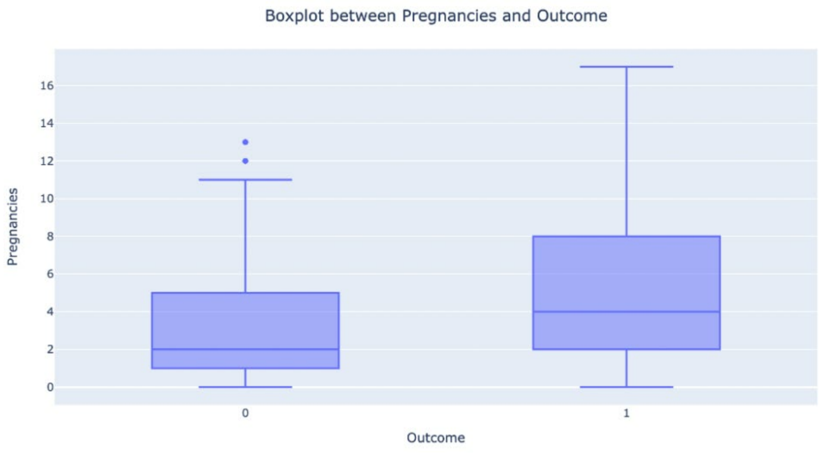


Figure 10. boxplot between pregnancies and outcome

The average number of pregnancies of diabetes women is larger than the women without diabetes.

```
# Explore Pregnancies vs Glucose
fig2 = px.box(diabetes, x="Outcome", y="Glucose")
fig2.update_layout(title_text='Boxplot between Glucose and Outcome', title_x=0.5)
fig2.show()
```

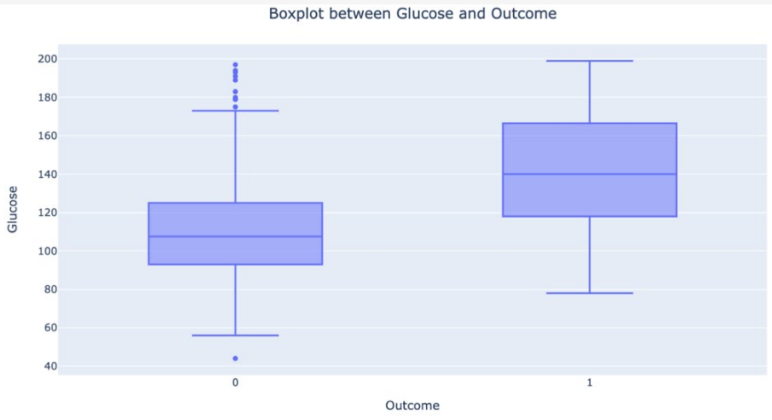


Figure 11. boxplot between glucose and outcome

Diabetes women has higher average level of glucose than the women without diabetes.

```
# Explore Insulin vs Outcome
fig3 = px.box(diabetes, x="Outcome", y="Insulin")
fig3.update_layout(title_text='Boxplot between Insulin and Outcome', title_x=0.5)
fig3.show()
```

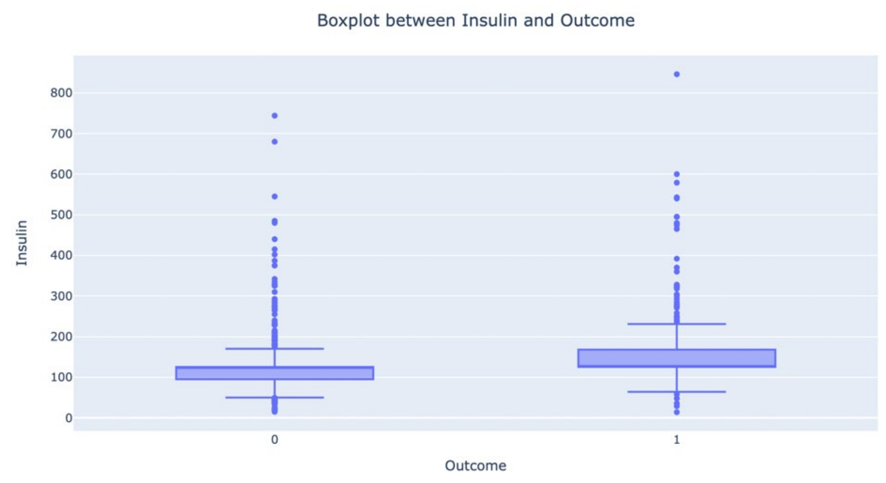


Figure 12. boxplot between insulin and outcome

There are lots of outliers in both diabetes and non-diabetes group.

```
# Explore BMI vs Outcome
fig4 = px.box(diabetes, x="Outcome", y="BMI")
fig4.update_layout(title_text='Boxplot between BMI and Outcome', title_x=0.5)
fig4.show()
```

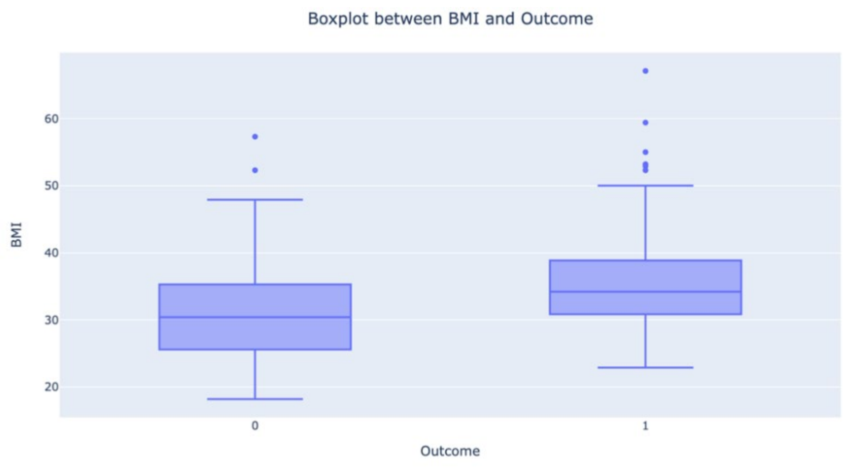


Figure 13. boxplot between BMI and outcome

Diabetes women has higher average score of BMI than the women without diabetes.

3 Relation between three-or-more children and diabetes

Based on the exploratory data analysis above, we conjecture that having diabetes may be related to the number of pregnancies. To further explore the relationship, we used three pregnancies as a cut-off point, setting three and more pregnancies as 1 and less than three as 0. Then use this new set variable to predict whether or not to have diabetes.

```
#Compute True or False based on the 'Pregnancies' column
three_or_more_children_array = diabetes['Pregnancies'] > 2
three_or_more_children = pd.DataFrame(three_or_more_children_array)
```

```
#Add a new column named as 'three_or_more_children'
diabetes['three_or_more_children'] = three_or_more_children
#Convert True or False to 1 or 0, to simplify the further computation
diabetes['three_or_more_children'] = diabetes['three_or_more_children'].replace(True,1)
diabetes['three_or_more_children'] = diabetes['three_or_more_children'].replace(False,0)
diabetes
```

Table 5. new dataset with column 'three_or_more_children'

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigree	Age	Outcome	three_or_more_children
0	6	148.0	72.0	35.0	125.5	33.6	0.627	50	1	1
1	1	85.0	66.0	29.0	125.5	26.6	0.351	31	0	0
2	8	183.0	64.0	29.0	125.5	23.3	0.672	32	1	1
3	1	89.0	66.0	23.0	94.0	28.1	0.167	21	0	0
4	0	137.0	40.0	35.0	168.0	43.1	2.288	33	1	0
...	...	...	...	...	...	...	...	...	...	...
745	12	100.0	84.0	33.0	105.0	30.0	0.488	46	0	1
746	1	147.0	94.0	41.0	125.5	49.3	0.358	27	1	0
747	1	81.0	74.0	41.0	57.0	46.3	1.096	32	0	0
748	3	187.0	70.0	22.0	200.0	36.4	0.408	36	1	1
749	6	162.0	62.0	29.0	125.5	24.3	0.178	50	1	1

3.1 Logistic Regression with One Binary Variable

```
#Apply Logistic Regression
#It's a one variable logistic regression, the only variable is 'three_or_more_children'
#Get x and y
x = diabetes['three_or_more_children'].array.reshape(-1,1)
y = diabetes['Outcome']

#Split data
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size = 0.3, random_state=42)

#Standard data
sc=StandardScaler()
sc.fit(x_train)
x_train_std=sc.transform(x_train)
x_test_std=sc.transform(x_test)

#Fit the training data to the logistic regression, set C equal to 10, and the max iteration number is 5000
lr = LogisticRegression(C=10,random_state=42, max_iter = 5000)
lr.fit(x_train_std,y_train.ravel())

#Apply the model on the test data and get a summary of the model.
y_pred = lr.predict(x_test_std)
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.69      1.00      0.82       154
           1       0.00      0.00      0.00        69

    accuracy                           0.69       223
   macro avg       0.35      0.50      0.41       223
weighted avg       0.48      0.69      0.56       223
```

Figure 14. summary for logistic model(variable:'three_or_more_children')

```
#Plot confusion matrix
def plot_confusion_matrix(y, y_pred):
    acc = round(accuracy_score(y, y_pred), 2)
    cm = confusion_matrix(y, y_pred)
    sns.heatmap(cm, annot=True, fmt=".0f")
    plt.xlabel('y_pred')
    plt.ylabel('y')
    plt.title('Accuracy Score: {0}'.format(acc), size=10)
    plt.show()

plot_confusion_matrix(y_test,y_pred)
```

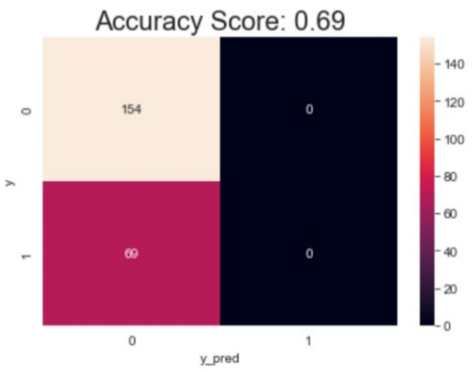


Figure 15. confusion matrix for logistic model(variable: 'three_or_more_children')

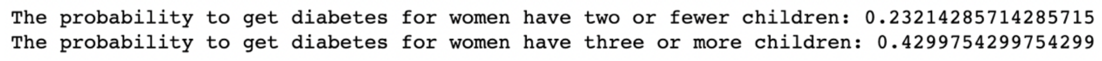
3.2 Probability for Developing Diabetes

The accuracy for this model is 0.69, and in confusion matrix, model failed to predict. This is because diabetes is caused by a number of factors. Predicting it by one single variable(whether or not has three or more children) is bound to introduce a large margin of error. Although the regression model is not accurate, we can use logistic regression to calculate the probability of a woman having diabetes if she has less than or equal to two children/has three or more than three children given this data set.

$$p_i = \frac{\exp(\beta_0+\beta_1 X_{1i}+\cdots+\beta_k X_{ki})}{1+\exp(\beta_0+\beta_1 X_{1i}+\cdots+\beta_k X_{ki})} \qquad (1,1)$$

```
#compute the number of women with three or more children
num_three_or_more_children = diabetes[diabetes['three_or_more_children'] ==
1]['three_or_more_children'].sum()
#compute the number of women who has three or more children and get diabetes
ntomc_with_diabetes = diabetes[diabetes['three_or_more_children'] == 1]['Outcome'].sum
#compute the number of women who has three or more children without diabetes
ntomc_without_diabetes = num_three_or_more_children - ntomc_with_diabetes
#compute the number of women with two or less children
num_less_three_children = diabetes[diabetes['three_or_more_children'] == 0]['three_or_more_children'].count()
#compute the number of women who has two or less children and get diabetes
nltc_with_diabetes = diabetes[diabetes['three_or_more_children'] == 0]['Outcome'].sum()
#compute the number of women who has two or less children without diabetes
```

```
nltc_without_diabetes = num_less_three_children - nltc_with_diabetes
#compute odd rate for women with three or more children
#It should be
(ntomc_with_diabetes/num_three_or_more_children)/(ntomc_without_diabetes/num_three_or_more_children)
#Then we can omit the num_three_or_more_children
log_odds_for_pregnancies_more_than_three = np.log(ntomc_with_diabetes/ntomc_without_diabetes)
#compute odd rate for women with two or less children
log_odds_for_pregnancies_less_than_three = np.log(nltc_with_diabetes/nltc_without_diabetes)
#compute probability
percentage_for_more_than_three =
np.exp(log_odds_for_pregnancies_more_than_three)/(1+np.exp(log_odds_for_pregnancies_more_than_three))
percentage_for_less_than_three =
np.exp(log_odds_for_pregnancies_less_than_three)/(1+np.exp(log_odds_for_pregnancies_less_than_three))
print('The probability to get diabetes for women have two or fewer children:',percentage_for_less_than_three)
print('The probability to get diabetes for women have three or more children:',percentage_for_more_than_three)
```



```
The probability to get diabetes for women have two or fewer children: 0.23214285714285715
The probability to get diabetes for women have three or more children: 0.4299754299754299
```

Figure 16. result of probability to get diabetes

According to this data set, women with two or fewer children have 23.21% percent of getting diabetes. Women with three or more children are more exposed to get diabetes with 43.00% percent. In next chapter, we'll discuss multiple variables' influence on diabetes.

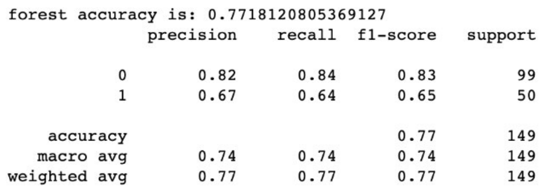
4 Regression Model

Since the outcome is 0 or 1, we can use logistic regression or a classifier, such as RandomForest, SVC, KNN, etc.

4.1 Random Forest Model

```
#Decision Tree is commonly used to do classification
from sklearn.ensemble import RandomForestClassifier

#Pick data set
x = diabetes.drop(['Outcome','three_or_more_children'], axis=1)
y = diabetes['Outcome']

#Split data
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size = 0.2,random_state=42)

#Set n_estimators equal to 500, max_features to 'auto' and fit the model
forest = RandomForestClassifier(n_estimators = 500, max_features = 'auto')
forest.fit(x_train,y_train)

#Model accuracy
print('forest accuracy is:', forest.score(x_test,y_test))

#Model summary
y_pred = forest.predict(x_test)
```

```
print(classification_report(y_test,y_pred))
```



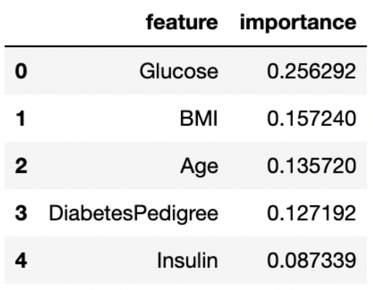
```
forest accuracy is: 0.7718120805369127
              precision    recall  f1-score   support

           0       0.82      0.84      0.83        99
           1       0.67      0.64      0.65        50

    accuracy                           0.77       149
   macro avg       0.74      0.74      0.74       149
weighted avg       0.77      0.77      0.77       149
```

Figure 16. summary for random forest model

The accuracy of this model on the test data is 0.77. Then we can use it to select features.

```
#Feature selection by importance
importances = forest.feature_importances_
indices = np.argsort(-importances)
df_imp = pd.DataFrame(dict(feature=x_train.columns[indices],importance=importances[indices]))
df_imp.head()
```

Table 6. important features from random forest



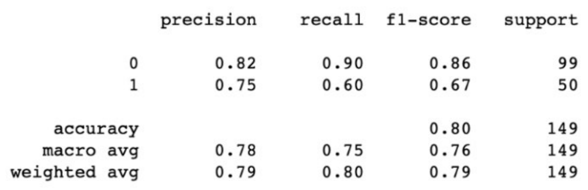
	feature	importance
0	Glucose	0.256292
1	BMI	0.157240
2	Age	0.135720
3	DiabetesPedigree	0.127192
4	Insulin	0.087339

Variables ‘Glucose’, ‘BMI’, ‘Age’, ‘DiabetesPedigree’ and ‘Insulin’ have high ranks in feature importance.

4.2 Logistic Regression Model with Feature-selecting

Here, we set ‘Glucose’, ‘BMI’, ‘Age’, ‘DiabetesPedigree’ and ‘Insulin’ as variables for our logistic regression model.

```
#Set data for regression
x = diabetes.drop(['Outcome','three_or_more_children','SkinThickness','BloodPressure','Pregnancies'], axis=1)
y = diabetes['Outcome']

#Define a logistic model function
def lrmodel(x,y):
    #Split data
    x_train,x_test,y_train,y_test = train_test_split(x,y,test_size = 0.2, random_state=42)
    #Standard data
    sc=StandardScaler()
    sc.fit(x_train)
    x_train_std=sc.transform(x_train)
    x_test_std=sc.transform(x_test)

    #Fit model
    lr = LogisticRegression(C=10,random_state=42, max_iter = 5000)
    lr.fit(x_train_std,y_train.ravel())
    #Model score
    y_pred = lr.predict(x_test_std)
    print(classification_report(y_test,y_pred))

lrmodel(x,y)
```

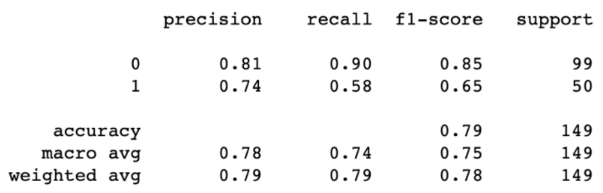


```
              precision    recall  f1-score   support

           0       0.82      0.90      0.86        99
           1       0.75      0.60      0.67        50

    accuracy                           0.80       149
   macro avg       0.78      0.75      0.76       149
weighted avg       0.79      0.80      0.79       149
```

Figure 17. summary for random forest model with feature variables

Meanwhile, I want to compare this to the model contains all variables.

```
#Data select
x = diabetes.drop(['Outcome','three_or_more_children'], axis=1)
y = diabetes['Outcome']
#Call function
lrmodel(x,y)
```

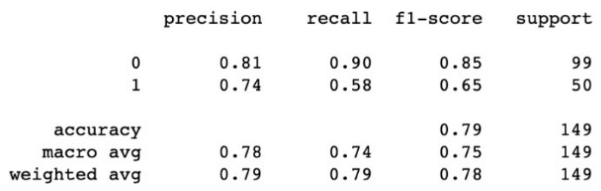


```
              precision    recall  f1-score   support

           0       0.81      0.90      0.85        99
           1       0.74      0.58      0.65        50

    accuracy                           0.79       149
   macro avg       0.78      0.74      0.75       149
weighted avg       0.79      0.79      0.78       149
```

Figure 18. summary for random forest model with all variables

We replaced the missing value with median, and the Insulin and SkinThickness contain almost half missing value, then we will apply a new model without these two attributes.

```
#LR model without Insulin and SkinThickness
#Set data and call lr function
x = diabetes.drop(['Outcome','three_or_more_children'], axis=1)
y = diabetes['Outcome']
lrmodel(x,y)
```



```
              precision    recall  f1-score   support

           0       0.81      0.90      0.85        99
           1       0.74      0.58      0.65        50

    accuracy                           0.79       149
   macro avg       0.78      0.74      0.75       149
weighted avg       0.79      0.79      0.78       149
```

Figure 19. summary for random forest model without Insulin and SkinThickness

We found that the model with feature variables got the highest accuracy score, then we'll select it as our final model.

4.3 Cross Matrix and ROC

```
def plot_confusion_matrix(y, y_pred):
    acc = round(accuracy_score(y, y_pred), 2)
    cm = confusion_matrix(y, y_pred)
    sns.heatmap(cm, annot=True, fmt=".0f")
    plt.xlabel('y_pred')
    plt.ylabel('y')
    plt.title('Accuracy Score: {0}'.format(acc), size=10)
    plt.show()

plot_confusion_matrix(y_test,y_pred)
```

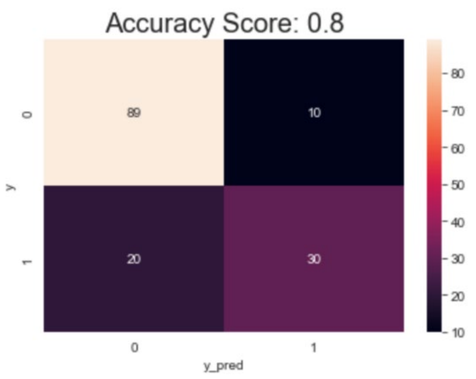


Figure 20. confusion matrix for random forest model with feature variables

The receiver operating characteristic (ROC) curve is another common tool used to test the binary logistic regression. A good regression model stays as far away from that line as possible (toward the top-left corner).

```
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
logit_roc_auc = roc_auc_score(y_test, lr.predict(x_test_std))
fpr, tpr, thresholds = roc_curve(y_test, lr.predict_proba(x_test_std)[:,1])
plt.figure()
plt.plot(fpr, tpr, label='Logistic Regression (area = %0.2f)' % logit_roc_auc)
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic for logistic regression')
plt.legend(loc="lower right")
plt.savefig('Log_ROC')
plt.show()
```

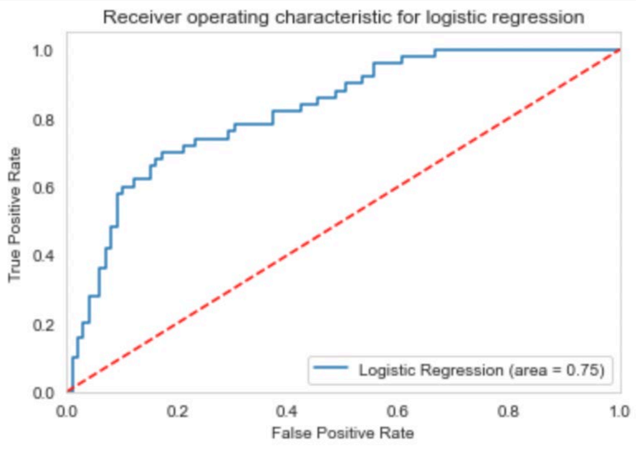


Figure 21. ROC for logistic regression model with feature variables

5 Prediction

```
#Import predict values
predict = pd.read_csv('ToPredict.csv')

#Customize the data
#There are two missing value in the column 'Insulin', thus we take the original strategy:
#replace the missing value with the median: 125.500
x_pred = predict.drop(['SkinThickness','BloodPressure','Pregnancies'], axis=1)
x_pred.replace(0,125.500,inplace=True)

#Standard data
sc=StandardScaler()
sc.fit(x_pred)
x_pred_std=sc.transform(x_pred)

#Predict
y_pred = lr.predict(x_pred_std)
predict['y_pred'] = y_pred
predict
```

Table 7. final prediction value for given data

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigree	Age	y_pred
0	4	136	70	0	0	31.2	1.182	22	0
1	1	121	78	39	74	39.0	0.261	28	0
2	3	108	62	24	0	26.0	0.223	25	0
3	0	181	88	44	510	43.3	0.222	26	1
4	8	154	78	32	0	32.4	0.443	45	1

According to the model, the people indexed in 3 and 4 have a relatively high risk of developing diabetes. In our logistic regression model, when the probability of a patient having diabetes exceeds 50%, his outcome becomes 1. So, let's look at diabetes probability in more detail below.

```
#count probability to get diabetes for each person
y_pred_prob = lr.predict_proba(x_pred_std)
probability_for_prediction_value = pd.DataFrame(y_pred_prob,columns=['probabiloity without
diabetes','probability to have diabetes'])
probability_for_prediction_value
```

Table 8. probability for each person to have diabetes

	probabiloity without diabetes	probability to have diabetes
0	0.768186	0.231814
1	0.793740	0.206260
2	0.966179	0.033821
3	0.224148	0.775852
4	0.444237	0.555763

Here we get the specific probability of whether or not she will develop diabetes. The patient with index #2 has the slightest probability of having diabetes. However, her insulin level is a missing value and we fill it with the median, which may cause bias in the prediction. The patient with index #0 has a 23.18% chance of developing diabetes, although she had four pregnancies and a relatively high DiabetesPedigree, which could also be a point of inaccuracy in the model predictions.

Reference

Acuna E, Rodriguez C (2004) The treatment of missing values and its effect in the classifier accuracy. In: Banks D et al (eds) Classification, clustering and data mining applications. Springer, Berlin, pp 639–648

Kwak, S.K. and Kim, J.H., 2017. Statistical data preparation: management of missing values and outliers. Korean journal of anesthesiology, 70(4), pp.407-411.

Ou, H.T., Chen, P.C. and Wu, M.H., 2017. Effect of metformin by employing 2-hour postload insulin for measuring insulin resistance in Taiwanese women with polycystic ovary syndrome. Journal of the Formosan Medical Association, 116(2), pp.80-89.

Roglic, G., 2016. WHO Global report on diabetes: A summary. International Journal of Noncommunicable Diseases, 1(1), p.3.

Smith, J.W., Everhart, J.E., Dickson, W.C., Knowler, W.C., & Johannes, R.S. (1988). Using the ADAP learning algorithm to forecast the onset of diabetes mellitus. *In Proceedings of the Symposium on Computer Applications and Medical Care* (pp. 261--265).

Zimmet, P.Z., Magliano, D.J., Herman, W.H. and Shaw, J.E., 2014. Diabetes: a 21st century challenge. The lancet Diabetes & endocrinology, 2(1), pp.56-64.