**Diabetes Prediction Model**

**Machine Learning**

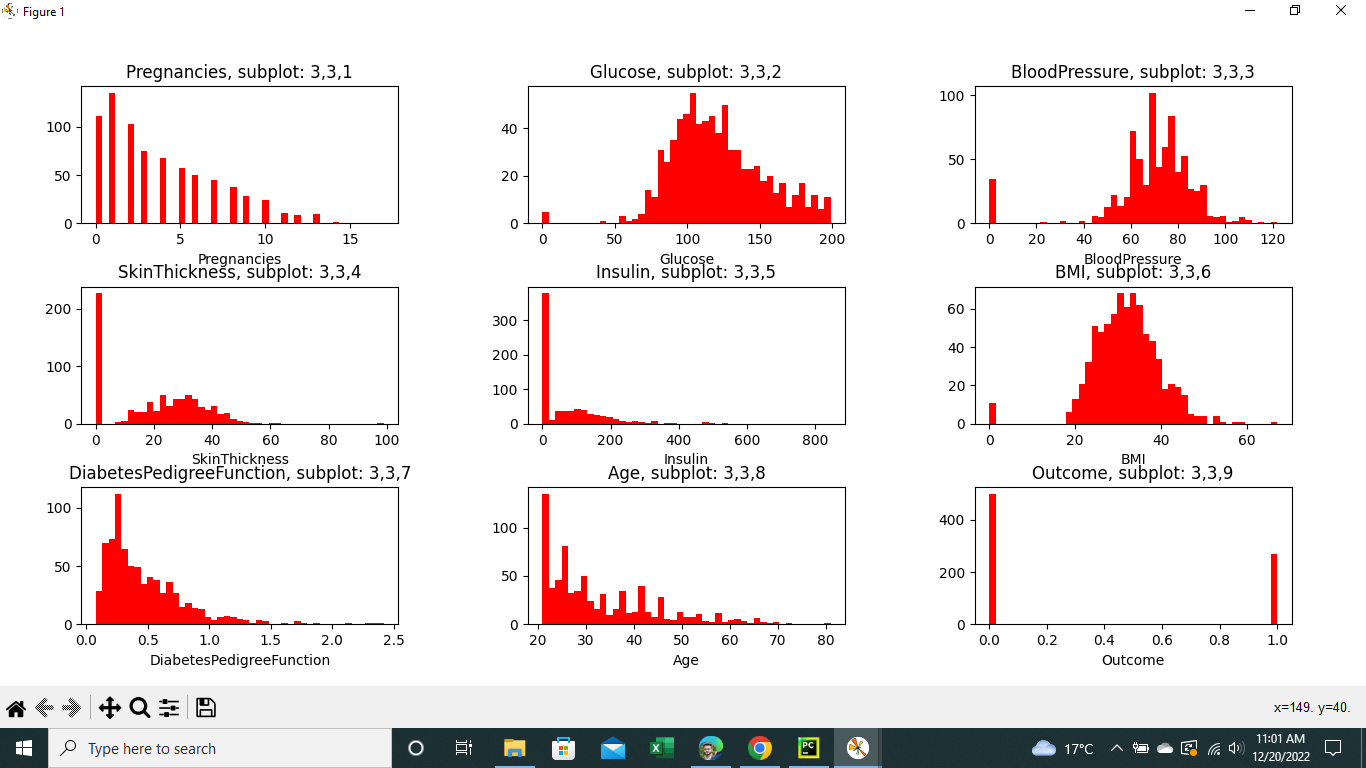
**(Python-Pycharm)**

*#Importing the basic libraries for analysis*import pandas as pd  
import numpy as np  
import matplotlib.pyplot as plt  
import seaborn as sns  
from sklearn.preprocessing import StandardScaler  
from sklearn.model\_selection import train\_test\_split  
from sklearn.tree import DecisionTreeClassifier  
from sklearn.ensemble import RandomForestClassifier  
from sklearn.ensemble import GradientBoostingClassifier  
from sklearn.linear\_model import LogisticRegression  
from sklearn.neighbors import KNeighborsClassifier  
from sklearn.svm import SVC  
from sklearn.metrics import r2\_score  
  
*#Import the dataset*df =pd.read\_csv("C:\\Users\\neeld\Desktop\Study\Tutorial\_1\pythonProjectTestDataScientist\diabetes - DS.csv")

**Dataset**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Pregnancies** | **Glucose** | **BloodPressure** | **SkinThickness** | **Insulin** | **BMI** | **DiabetesPedigreeFunction** | **Age** | **Outcome** |
| **6** | **148** | **72** | **35** | **0** | **33.6** | **0.627** | **50** | **1** |
| **1** | **85** | **66** | **29** | **0** | **26.6** | **0.351** | **31** | **0** |
| **8** | **183** | **64** | **0** | **0** | **23.3** | **0.672** | **32** | **1** |
| **1** | **89** | **66** | **23** | **94** | **28.1** | **0.167** | **21** | **0** |
| **0** | **137** | **40** | **35** | **168** | **43.1** | **2.288** | **33** | **1** |
| **5** | **116** | **74** | **0** | **0** | **25.6** | **0.201** | **30** | **0** |
| **3** | **78** | **50** | **32** | **88** | **31** | **0.248** | **26** | **1** |
| **10** | **115** | **0** | **0** | **0** | **35.3** | **0.134** | **29** | **0** |
| **2** | **197** | **70** | **45** | **543** | **30.5** | **0.158** | **53** | **1** |
| **8** | **125** | **96** | **0** | **0** | **0** | **0.232** | **54** | **1** |
| **4** | **110** | **92** | **0** | **0** | **37.6** | **0.191** | **30** | **0** |
| **10** | **168** | **74** | **0** | **0** | **38** | **0.537** | **34** | **1** |
| **10** | **139** | **80** | **0** | **0** | **27.1** | **1.441** | **57** | **0** |
| **1** | **189** | **60** | **23** | **846** | **30.1** | **0.398** | **59** | **1** |
| **5** | **166** | **72** | **19** | **175** | **25.8** | **0.587** | **51** | **1** |
| **7** | **100** | **0** | **0** | **0** | **30** | **0.484** | **32** | **1** |
| **0** | **118** | **84** | **47** | **230** | **45.8** | **0.551** | **31** | **1** |
| **7** | **107** | **74** | **0** | **0** | **29.6** | **0.254** | **31** | **1** |
| **1** | **103** | **30** | **38** | **83** | **43.3** | **0.183** | **33** | **0** |

*#print first five rows from the data set*print(df.head())  
  
*#print last five rows from the data set*print(df.head())  
  
*#Check the data types of columns*print(df.info())  
*#Check the null values of columns*print(df.isnull().sum())  
*#Check the na values of columns*print(df.isna().sum())  
*#Describe the data set*print(df.describe())  
  
*#We can see.. there are no missing values  
#two types of data (int,float)  
  
#..........perform the EDA.........*Columns\_all = ['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age','Outcome']  
  
*#using histogram identifying the distribution of each columns data*fig = plt.figure(figsize=(3,3))  
a = 3 *# number of rows*b = 3 *# number of columns*c = 1 *# initialize plot counter*for i in Columns\_all:  
 plt.subplot(a, b, c)  
 plt.title('{}, subplot: {},{},{}'.format(i, a, b, c))  
 plt.xlabel(i)  
 plt.hist(x=df[i],bins=45,color='red')  
 c = c + 1  
  
plt.tight\_layout()  
plt.show()

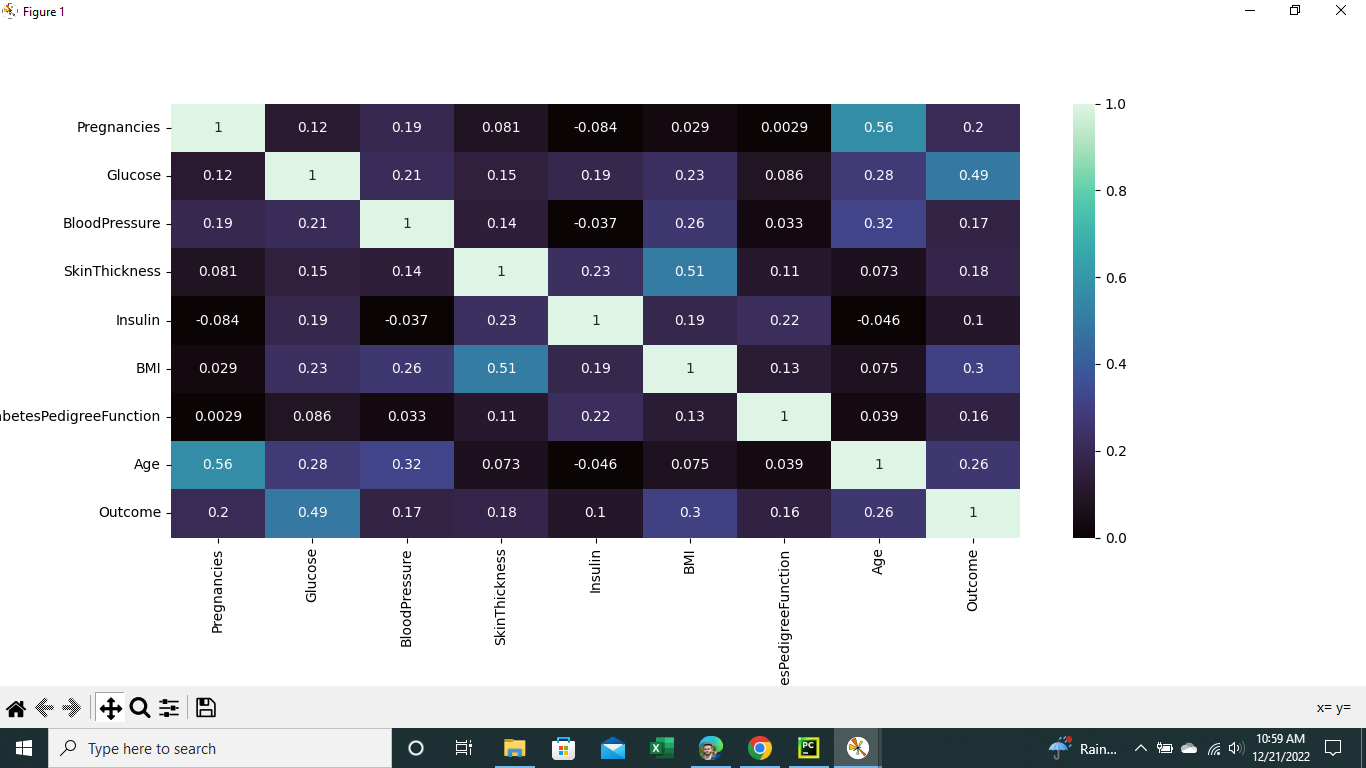
  
*#print the outcome count*print(df.Outcome.value\_counts())  
  
*#plot that outcome count*count\_outcome = pd.value\_counts(df['Outcome'],sort=True)  
count\_outcome.plot(kind='bar',rot=0,)  
plt.title('Outcome Distribution')  
plt.ylabel('Count')  
plt.xlabel('outcome')  
plt.show()  
Chart, bar chart

Description automatically generated  
*#push columns name into a list Columns*Columns = ['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age']  
  
*#Count the Zeros in each rows*for i in Columns:  
 count = (df[i] == 0).sum()  
 print('Zero count in column :',i,' are :::', count, )  
  
*#Zero count in column : Pregnancies are ::: 111  
#Zero count in column : Glucose are :::.... 5  
#Zero count in column : BloodPressure are ::: 35  
#Zero count in column : SkinThickness are ::: 227  
#Zero count in column : Insulin are ::: 374  
#Zero count in column : BMI are ::: 11  
#Zero count in column : DiabetesPedigreeFunction are ::: 0  
#Zero count in column : Age are ::: 0  
  
#pregnancies can be have zero values  
#But 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin' and 'BMI' can not have zero values..  
#Then I will fill them using median value from respective columns median*df['Glucose'] = df['Glucose'].replace(0, df['Glucose'].median())  
df['BloodPressure'] = df['BloodPressure'].replace(0, df['BloodPressure'].median())  
df['SkinThickness'] = df['SkinThickness'].replace(0, df['SkinThickness'].median())  
df['Insulin'] = df['Insulin'].replace(0, df['Insulin'].median())  
df['BMI'] = df['BMI'].replace(0, df['BMI'].median())  
  
print('After zero values replace with median---------------------')  
for i in Columns:  
 count = (df[i] == 0).sum()  
 print('Zero count in column :',i,' are :::', count, )  
  
*#Now check the outliers*Columns\_all = ['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age','Outcome']  
  
*#using boxplot identifying the outliers*fig = plt.figure(figsize=(3,3))  
a = 3 *# number of rows*b = 3 *# number of columns*c = 1 *# initialize plot counter*for i in Columns\_all:  
 plt.subplot(a, b, c)  
 plt.title('{}, subplot: {},{},{}'.format(i, a, b, c))  
 plt.xlabel(i)  
 sns.boxplot(x=df[i])  
 c = c + 1  
  
plt.tight\_layout()  
plt.show()

Graphical user interface, diagram

Description automatically generated  
  
*#we can see 'Glucose' and 'Outcome' columns not included outliers..  
#But 'Pregnancies', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI', 'DiabetesPedigreeFunction' and 'Age' are included Outliers  
#Then we have to remove those outliers from the each columns  
#Columns\_WithOutlier = ['Pregnancies', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age']  
  
#now find the outliers in 'Insulin'*q25 = np.percentile(df['Insulin'], 25)  
q75 = np.percentile(df['Insulin'], 75)  
iqr = q75 - q25  
print('Percentiles of: 25th=%.3f, 75th=%.3f, IQR=%.3f' % (q25, q75, iqr))  
  
cut\_off = iqr \* 1.5  
lower = q25 - cut\_off  
upper = q75 + cut\_off  
  
outliers = [x for x in df['Insulin'] if x < lower or x > upper]  
print('Identified outliers: %d' % len(outliers))  
  
outliers\_remove = [x for x in df['Insulin'] if x >= lower and x <= upper]  
print('Non-outlier observations: %d' % len(outliers\_remove))  
  
*#replace outliers with mean of the 'outliers\_remove'*df['Insulin'] = df['Insulin'].replace(outliers,np.mean(outliers\_remove))  
  
*#now find the outliers in 'Pregnancies'*q25 = np.percentile(df['Pregnancies'], 25)  
q75 = np.percentile(df['Pregnancies'], 75)  
iqr = q75 - q25  
print('Percentiles of: 25th=%.3f, 75th=%.3f, IQR=%.3f' % (q25, q75, iqr))  
  
cut\_off = iqr \* 1.5  
lower = q25 - cut\_off  
upper = q75 + cut\_off  
  
outliers = [x for x in df['Pregnancies'] if x < lower or x > upper]  
print('Identified outliers Pregnancies: %d' % len(outliers))  
  
outliers\_remove = [x for x in df['Pregnancies'] if x >= lower and x <= upper]  
print('Non-outlier observations Pregnancies: %d' % len(outliers\_remove))  
  
*#replace outliers with mean of the 'outliers\_remove'*df['Pregnancies'] = df['Pregnancies'].replace(outliers,np.mean(outliers\_remove))  
  
*#now find the outliers in 'BloodPressure'*q25 = np.percentile(df['BloodPressure'], 25)  
q75 = np.percentile(df['BloodPressure'], 75)  
iqr = q75 - q25  
print('BloodPressure of: 25th=%.3f, 75th=%.3f, IQR=%.3f' % (q25, q75, iqr))  
  
cut\_off = iqr \* 1.5  
lower = q25 - cut\_off  
upper = q75 + cut\_off  
  
outliers = [x for x in df['BloodPressure'] if x < lower or x > upper]  
print('Identified outliers: %d' % len(outliers))  
  
outliers\_remove = [x for x in df['BloodPressure'] if x >= lower and x <= upper]  
print('Non-outlier observations BloodPressure: %d' % len(outliers\_remove))  
  
*#replace outliers with mean of the 'outliers\_remove'*df['BloodPressure'] = df['BloodPressure'].replace(outliers,np.mean(outliers\_remove))  
  
*#now find the outliers in 'SkinThickness'*q25 = np.percentile(df['SkinThickness'], 25)  
q75 = np.percentile(df['SkinThickness'], 75)  
iqr = q75 - q25  
print('SkinThickness of: 25th=%.3f, 75th=%.3f, IQR=%.3f' % (q25, q75, iqr))  
  
cut\_off = iqr \* 1.5  
lower = q25 - cut\_off  
upper = q75 + cut\_off  
  
outliers = [x for x in df['SkinThickness'] if x < lower or x > upper]  
print('Identified outliers SkinThickness: %d' % len(outliers))  
  
outliers\_remove = [x for x in df['SkinThickness'] if x >= lower and x <= upper]  
print('Non-outlier observations SkinThickness: %d' % len(outliers\_remove))  
  
*#replace outliers with mean of the 'outliers\_remove'*df['SkinThickness'] = df['SkinThickness'].replace(outliers,np.mean(outliers\_remove))  
  
*#now find the outliers in 'BMI'*q25 = np.percentile(df['BMI'], 25)  
q75 = np.percentile(df['BMI'], 75)  
iqr = q75 - q25  
print('BMI of: 25th=%.3f, 75th=%.3f, IQR=%.3f' % (q25, q75, iqr))  
  
cut\_off = iqr \* 1.5  
lower = q25 - cut\_off  
upper = q75 + cut\_off  
  
outliers = [x for x in df['BMI'] if x < lower or x > upper]  
print('Identified outliers BMI: %d' % len(outliers))  
  
outliers\_remove = [x for x in df['BMI'] if x >= lower and x <= upper]  
print('Non-outlier observations BMI: %d' % len(outliers\_remove))  
  
*#replace outliers with mean of the 'outliers\_remove'*df['BMI'] = df['BMI'].replace(outliers,np.mean(outliers\_remove))  
  
*#now find the outliers in 'DiabetesPedigreeFunction'*q25 = np.percentile(df['DiabetesPedigreeFunction'], 25)  
q75 = np.percentile(df['DiabetesPedigreeFunction'], 75)  
iqr = q75 - q25  
print('DiabetesPedigreeFunction of: 25th=%.3f, 75th=%.3f, IQR=%.3f' % (q25, q75, iqr))  
  
cut\_off = iqr \* 1.5  
lower = q25 - cut\_off  
upper = q75 + cut\_off  
  
outliers = [x for x in df['DiabetesPedigreeFunction'] if x < lower or x > upper]  
print('Identified outliers: %d' % len(outliers))  
  
outliers\_remove = [x for x in df['DiabetesPedigreeFunction'] if x >= lower and x <= upper]  
print('Non-outlier observations: %d' % len(outliers\_remove))  
  
*#replace outliers with mean of the 'outliers\_remove'*df['DiabetesPedigreeFunction'] = df['DiabetesPedigreeFunction'].replace(outliers,np.mean(outliers\_remove))  
  
*#now find the outliers in 'Age'*q25 = np.percentile(df['Age'], 25)  
q75 = np.percentile(df['Age'], 75)  
iqr = q75 - q25  
print('Age of: 25th=%.3f, 75th=%.3f, IQR=%.3f' % (q25, q75, iqr))  
  
cut\_off = iqr \* 1.5  
lower = q25 - cut\_off  
upper = q75 + cut\_off  
  
outliers = [x for x in df['Age'] if x < lower or x > upper]  
print('Identified outliers Age: %d' % len(outliers))  
  
outliers\_remove = [x for x in df['Age'] if x >= lower and x <= upper]  
print('Non-outlier observations Age: %d' % len(outliers\_remove))  
  
*#replace outliers with mean of the 'outliers\_remove'*df['Age'] = df['Age'].replace(outliers,np.mean(outliers\_remove))  
  
  
*#After removed outliers drowing the boxplots*fig = plt.figure(figsize=(3,3))  
a = 3 *# number of rows*b = 3 *# number of columns*c = 1 *# initialize plot counter*for i in Columns\_all:  
 plt.subplot(a, b, c)  
 plt.title('{}, subplot: {},{},{}'.format(i, a, b, c))  
 plt.xlabel(i)  
 sns.boxplot(x=df[i])  
 c = c + 1  
  
plt.tight\_layout()  
plt.show()  
Graphical user interface

Description automatically generated  
*#Now check the multicoliniarity of the data  
# Extreme Multicollinearity occurs whenever the  
# independent variables are very high and Correlated  
# with one or more other independent variables.*print(df.corr())  
sns.heatmap(df.corr(),annot=True,cmap="mako", vmin=0, vmax=1)  
plt.title("Correlation Matrix", fontsize=20)  
plt.show()

  
  
*#Check the imbalanced data*count\_outcome = pd.value\_counts(df['Outcome'],sort=True)  
  
print(count\_outcome\*100/len(df))  
*#0 65.104167% =~ 65%  
#1 34.895833% =~35%  
#Many datasets will have an uneven number of instances in each class,  
# but a small difference is usually acceptable. As a rule of thumb,  
# if a two-class dataset has a difference of greater than 65% to 35%,  
# than it should be looked at as a dataset with class imbalance  
# Then we don't need to balance this data set,  
  
  
# the necessary libraries are imported beginning  
#Extracting Independent and dependent Variable*y= df['Outcome']  
print(y.head())  
x= df.drop(['Outcome'],axis=1)  
print(x.head())  
  
*# Splitting the dataset into training and test set.*x\_train, x\_test, y\_train, y\_test= train\_test\_split(x, y, test\_size= 0.2, random\_state=0)  
  
*# Seeing the split across training and testing datasets*print('Number of records and shape in the original dataset: ', len(x),'--shape :',x.shape)  
print('Number of records and shape in the training dataset: ', len(x\_train),'--shape :',x\_train.shape)  
print('Number of records and shape in the testing dataset: ', len(x\_test),'--shape :',x\_test.shape)  
print('Number of records and shape in the original dataset: ', len(y),'--shape :',y.shape)  
print('Number of records and shape in the training dataset: ', len(y\_train),'--shape :',y\_train.shape)  
print('Number of records and shape in the testing dataset: ', len(y\_test),'--shape :',y\_test.shape)  
  
*# Now we can scaler standerdization independent variables*st\_x= StandardScaler()  
x\_train= st\_x.fit\_transform(x\_train)  
x\_test= st\_x.transform(x\_test)  
  
print(x\_test)  
print(x\_train)  
  
*# Now build the model using below algorithms,  
# 1) Decision tree  
# 2) Random forest  
# 3) Support vector machine  
# 4) Gradient boosting  
# 5) Logistic regression  
# 6) K-nearest neighbors*DT = DecisionTreeClassifier()  
RF = RandomForestClassifier()  
SVM = SVC()  
GB = GradientBoostingClassifier()  
LR = LogisticRegression()  
KNN = KNeighborsClassifier()  
  
*#Create list of algorithms*li = [DT,RF,SVM,GB,LR,KNN]  
d = {}  
  
  
for i in li:  
 *# fitting the training data set into the the each model* i.fit(x\_train,y\_train)  
 *#predict the outcome using testing data set* ypred = i.predict(x\_test)  
 *#print the r squar values for each algorithms* print(i,":",r2\_score(ypred,y\_test)\*100)  
 d.update({str(i):i.score(x\_test,y\_test)\*100})  
  
*# now comparison of the models and find the best model from the list of model*plt.figure(figsize=(20,6))  
plt.title("Algorithm vs Accuracy")  
plt.xlabel("Algorithm")  
plt.ylabel("Accuracy")  
plt.plot(d.keys(),d.values(),marker='o',color='black')  
plt.show()  
Graphical user interface, chart, line chart

Description automatically generated

*# We can see the best model is "Gradient boosting" clasification model.*