Capstone 2 - Diabetes prediction modeling

Project Idea:

- Predicting Diabetes:

Diabetes Mellitus is an increasingly prevalent chronic disease characterized by the body's inability to metabolize glucose. Building Machine Learning model to predict whether a person has diabetes or not, based on information about the patient such as blood pressure, body mass index (BMI), age, etc.

The model is to help physicians in predicting patients with future occurrence of diabetes and providing necessary preventive interventions. Fasting blood glucose, body mass index, high-density lipoprotein, and triglycerides were the most important predictors in these models. Compare performance of classification algorithms

Dataset source: The data was collected and made available by "National Institute of Diabetes and Digestive and Kidney Diseases" as part of the Pima Indians Diabetes Database.

Introduction

We will be using Python as our programming language, and making use of some popular python machine learning and data science related packages. First of all, we will import pandas to read our data from a CSV file and manipulate it for further use. We will also use numpy to convert out data into a format suitable to feed our classification model. We'll use seaborn and matplotlib for visualizations. We will then import Logistic Regression algorithm from sklearn. This algorithm will help us build our classification model. Lastly, we will use joblib available in sklearn to save our model for future use.

This model will predict which people are likely to develop diabetes.

This dataset is originally from the National Institute of Diabetes and Digestive and Kidney Diseases. The objective of the dataset is to diagnostically predict whether or not a patient has diabetes, based on certain diagnostic measurements included in the dataset. Several constraints were placed on the selection of these instances from a larger database. In particular, all patients here are females at least 21 years old of Pima Indian heritage.

Import Libraries

In [3]:
```
%%time
import warnings
warnings.filterwarnings('ignore')

import pandas as pd  # data processing, CSV file I/O (e.g. pd.read_csv)
import numpy as np # linear algebra
import seaborn as sns
import matplotlib.pyplot as plt  # matplotlib.pyplot plots data
%matplotlib inline

from pandas.api.types import is_string_dtype, is_numeric_dtype

from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
import joblib


from tabulate import tabulate
from mpl_toolkits.mplot3d import Axes3D
from sklearn.preprocessing import StandardScaler

import pandas_profiling as pp
import pycaret
#print('Using PyCaret Version', pycaret.__version__)
#print('Path to PyCaret: ', pycaret.__file__)
from pycaret.classification import *
```

```
Wall time: 2.59 s
```

In [4]:
```
import warnings
warnings.filterwarnings("ignore")

# settings
import os
from sb_utils import save_file
#from utils import
datapath = './data'
dataraw = '../data/raw'

if __name__ == '__main__' and __package__ is None:
    from os import sys, path
    sys.path.append(path.dirname(path.dirname(path.abspath("__file__"))))

%load_ext autoreload
%autoreload 2
```

In [ ]:
```
"""Attributes:

Dataset information:

This dataset is originally from the National Institute of Diabetes and Digestive and Kidney
Several constraints were placed on the selection of these instances from a larger database
In particular, all patients here are females at least 21 years old of Pima Indian heritage


Pregnancies: Number of times pregnant

Glucose: Plasma glucose concentration a 2 hours in an oral glucose tolerance test

BloodPressure: Diastolic blood pressure (mm Hg)

SkinThickness: Triceps skin fold thickness (mm)

Insulin: 2-Hour serum insulin (mu U/ml)

BMI: Body mass index (weight in kg/(height in m)^2)

DiabetesPedigreeFunction: Diabetes pedigree function

Age: Age (years)

Outcome: Class variable (0 or 1)

"""
```

2.4 Objectives

There are some fundamental questions to resolve in this notebook before you move on.

- Do you think you may have the data you need to tackle the desired question?
  - Have you identified the required target value?
  - Do you have potentially useful features?
- Do you have any fundamental issues with the data?

2.5 Load and review data - Data Description

Data cleaning - missing values, Outliers

In [6]:
```
## load the data
diabetesDF1 = pd.read_csv('diabetes.csv') #pima-indians-diabetes
diabetesDF1.head()
```

Out[6]:

	preg	plas	pres	skin	insu	mass	pedi	age	class
0	6	148	72	35	0	33.6	0.627	50	tested_positive
1	1	85	66	29	0	26.6	0.351	31	tested_negative
2	8	183	64	0	0	23.3	0.672	32	tested_positive
3	1	89	66	23	94	28.1	0.167	21	tested_negative
4	0	137	40	35	168	43.1	2.288	33	tested_positive

Content The datasets consists of several medical predictor variables and one target variable, Outcome. Columns are following :-

Pregnancies :- Number of times pregnant

Glucose:- Plasma glucose concentration a 2 hours in an oral glucose tolerance test

BloodPressure:- Diastolic blood pressure (mm Hg)

SkinThickness:- Triceps skin fold thickness (mm)

Insulin:- 2-Hour serum insulin (mu U/ml)

BMI:- Body mass index (weight in kg/(height in m)²)

DiabetesPedigreeFunction:- Diabetes pedigree function

Age:-Age in years

Outcome:- Class variable (0 or 1) 268 of 768 are 1, the others are 0

In [7]:
```
#1

#pp.ProfileReport(diabetesDF1)
```

Distinct	17
Distinct (%)	2.2%
Missing	0
Missing (%)	0.0%
Infinite	0
Infinite (%)	0.0%

Mean	3.845052083
Minimum	0
Maximum	17
Zeros	111

It is observed that the following are missing values / zeros / outliers

__count %__

insu 374 48.697917 , 374 zeros and outliers will be changed to average(79.79) / median(30.5)

skin 227 29.557292 , 227 zeros values keep as it is (these are correct values), max value 99

preg 111 14.453125 , 111 zeros are keeping as it is , max number of preg is 17

pres 35 4.557292 , 35 zeros will be change to average value

mass 11 1.432292 , 11 zeros (outliers) will be changed

plas 5 0.65104 , 5 zeros will be changed to average values

missing values

In [8]:
```
diabetesDF1.head(20)
```

Out[8]:

	preg	plas	pres	skin	insu	mass	pedi	age	class
0	6	148	72	35	0	33.6	0.627	50	tested_positive
1	1	85	66	29	0	26.6	0.351	31	tested_negative
2	8	183	64	0	0	23.3	0.672	32	tested_positive
3	1	89	66	23	94	28.1	0.167	21	tested_negative
4	0	137	40	35	168	43.1	2.288	33	tested_positive
5	5	116	74	0	0	25.6	0.201	30	tested_negative
6	3	78	50	32	88	31.0	0.248	26	tested_positive
7	10	115	0	0	0	35.3	0.134	29	tested_negative
8	2	197	70	45	543	30.5	0.158	53	tested_positive
9	8	125	96	0	0	0.0	0.232	54	tested_positive
10	4	110	92	0	0	37.6	0.191	30	tested_negative
11	10	168	74	0	0	38.0	0.537	34	tested_positive
12	10	139	80	0	0	27.1	1.441	57	tested_negative
13	1	189	60	23	846	30.1	0.398	59	tested_positive
14	5	166	72	19	175	25.8	0.587	51	tested_positive
15	7	100	0	0	0	30.0	0.484	32	tested_positive
16	0	118	84	47	230	45.8	0.551	31	tested_positive
17	7	107	74	0	0	29.6	0.254	31	tested_positive
18	1	103	30	38	83	43.3	0.183	33	tested_negative
19	1	115	70	30	96	34.6	0.529	32	tested_positive

nulls

In [9]:
```
# To check the missing values in the dataset

diabetesDF1.isnull().values.any()
```

Out[9]:
```
False
```

In [10]:
```
diabetesDF1.columns
```

Out[10]:
```
Index(['preg', 'plas', 'pres', 'skin', 'insu', 'mass', 'pedi', 'age', 'class'], dtype='ob
ject')
```

In [11]:
```
diabetesDF1.isnull().sum()[['preg', 'plas', 'pres', 'skin', 'insu', 'mass', 'pedi', 'age']]
```

Out[11]:
```
preg    0
plas    0
pres    0
skin    0
insu    0
mass    0
pedi    0
age     0
dtype: int64
```

In [10]:
```
missing = pd.concat([diabetesDF1.isnull().sum(), 100 * diabetesDF1.isnull().mean()], axis=1
missing.columns=['count','%']
missing.sort_values(by='count',ascending=False, na_position='first')
```

Out[10]:

	count	%
preg	0	0.0
plas	0	0.0
pres	0	0.0
skin	0	0.0
insu	0	0.0
mass	0	0.0
pedi	0	0.0
age	0	0.0
class	0	0.0

`insu` has the most missing values, at just over 50%. Unfortunately, you see you're also missing quite a few of your desired target quantity, the ticket price, which is missing 15-16% of values. `AdultWeekday` is missing in a few more records than `AdultWeekend`. What overlap is there in these missing values? This is a question you'll want to investigate. You should also point out that `isnull()` is not the only indicator of missing data. Sometimes 'missingness' can be encoded, perhaps by a -1 or 999. Such values are typically chosen because they are "obviously" not genuine values. If you were capturing data on people's heights and weights but missing someone's height, you could certainly encode that as a 0 because no one has a height of zero (in any units). Yet such entries would not be revealed by `isnull()`. Here, you need a data dictionary and/or to spot such values as part of looking for outliers. Someone with a height of zero should definitely show up as an outlier!

zeros

In [12]:
```
#Replace 0 to NaN

diabetesDF2=diabetesDF1[[ 'plas', 'pres',  'insu', 'mass', 'pedi', 'age']]=diabetesDF1[[ '
diabetesDF2.head()
```

Out[12]:

	plas	pres	insu	mass	pedi	age
0	148.0	72.0	NaN	33.6	0.627	50
1	85.0	66.0	NaN	26.6	0.351	31
2	183.0	64.0	NaN	23.3	0.672	32
3	89.0	66.0	94.0	28.1	0.167	21
4	137.0	40.0	168.0	43.1	2.288	33

In [13]:
```
# Find the number of Missing values - check again

diabetesDF2.isnull().sum()[[ 'plas', 'pres', 'insu', 'mass', 'pedi', 'age']]
```

Out[13]:
```
plas      5
pres     35
insu    374
mass     11
pedi      0
age       0
dtype: int64
```

Handling the Missing values by replacing NaN to median

In [14]:
```
#Replace NaN to mean value to explore dataset

# MEDIAN <--> MEAN

diabetesDF2['plas'].fillna(diabetesDF2['plas'].median(),inplace=True)
diabetesDF2['pres'].fillna(diabetesDF2['pres'].median(),inplace=True)

diabetesDF2['insu'].fillna(diabetesDF2['insu'].median(),inplace=True)
diabetesDF2['mass'].fillna(diabetesDF2['mass'].median(),inplace=True)
diabetesDF2.head()
```

Out[14]:

	plas	pres	insu	mass	pedi	age
0	148.0	72.0	125.0	33.6	0.627	50
1	85.0	66.0	125.0	26.6	0.351	31
2	183.0	64.0	125.0	23.3	0.672	32
3	89.0	66.0	94.0	28.1	0.167	21
4	137.0	40.0	168.0	43.1	2.288	33

In [15]:
```
diabetesDF1.head()
```

Out[15]:

	preg	plas	pres	skin	insu	mass	pedi	age	class
0	6	148.0	72.0	35	NaN	33.6	0.627	50	tested_positive
1	1	85.0	66.0	29	NaN	26.6	0.351	31	tested_negative
2	8	183.0	64.0	0	NaN	23.3	0.672	32	tested_positive
3	1	89.0	66.0	23	94.0	28.1	0.167	21	tested_negative
4	0	137.0	40.0	35	168.0	43.1	2.288	33	tested_positive

In [16]:
```
# Find the number of Missing values - check

diabetesDF2.isnull().sum()[[ 'plas', 'pres',  'insu', 'mass', 'pedi', 'age']]
```

Out[16]:
```
plas    0
pres    0
insu    0
mass    0
pedi    0
age     0
dtype: int64
```

In [17]:
```
diabetesDF = pd.concat((diabetesDF1[['preg','skin']],diabetesDF2,diabetesDF1['class']),axis
#([diabetesDF.isnull().sum(), 100 * diabetesDF.isnull().mean()], axis=1)
diabetesDF
```

Out[17]:

	preg	skin	plas	pres	insu	mass	pedi	age	class
0	6	35	148.0	72.0	125.0	33.6	0.627	50	tested_positive
1	1	29	85.0	66.0	125.0	26.6	0.351	31	tested_negative
2	8	0	183.0	64.0	125.0	23.3	0.672	32	tested_positive
3	1	23	89.0	66.0	94.0	28.1	0.167	21	tested_negative
4	0	35	137.0	40.0	168.0	43.1	2.288	33	tested_positive
...	...	...	...	...	...	...	...	...	...
763	10	48	101.0	76.0	180.0	32.9	0.171	63	tested_negative
764	2	27	122.0	70.0	125.0	36.8	0.340	27	tested_negative
765	5	23	121.0	72.0	112.0	26.2	0.245	30	tested_negative
766	1	0	126.0	60.0	125.0	30.1	0.349	47	tested_positive
767	1	31	93.0	70.0	125.0	30.4	0.315	23	tested_negative

768 rows × 9 columns

In [18]:
```
diabetesDF = diabetesDF2.copy()
diabetesDF[['preg','skin', 'class']] = diabetesDF1[['preg','skin', 'class']]
diabetesDF
```

Out[18]:

	plas	pres	insu	mass	pedi	age	preg	skin	class
0	148.0	72.0	125.0	33.6	0.627	50	6	35	tested_positive
1	85.0	66.0	125.0	26.6	0.351	31	1	29	tested_negative
2	183.0	64.0	125.0	23.3	0.672	32	8	0	tested_positive
3	89.0	66.0	94.0	28.1	0.167	21	1	23	tested_negative
4	137.0	40.0	168.0	43.1	2.288	33	0	35	tested_positive
...	...	...	...	...	...	...	...	...	...
763	101.0	76.0	180.0	32.9	0.171	63	10	48	tested_negative
764	122.0	70.0	125.0	36.8	0.340	27	2	27	tested_negative
765	121.0	72.0	112.0	26.2	0.245	30	5	23	tested_negative
766	126.0	60.0	125.0	30.1	0.349	47	1	0	tested_positive
767	93.0	70.0	125.0	30.4	0.315	23	1	31	tested_negative

768 rows × 9 columns

In [19]:
```
missing = pd.concat([diabetesDF.isin([0]).sum(), 100 * diabetesDF.isin([0]).mean()], axis=1
missing.columns=['count','%']
missing.sort_values(by='count',ascending=False, na_position='first')
```

Out[19]:

	count	%
skin	227	29.557292
preg	111	14.453125
plas	0	0.000000
pres	0	0.000000
insu	0	0.000000
mass	0	0.000000
pedi	0	0.000000
age	0	0.000000
class	0	0.000000

In [20]:
```
diabetesDF = diabetesDF[['preg', 'plas', 'pres', 'skin', 'insu', 'mass', 'pedi', 'age', 'cl
```

In [20]:

```
# save cleaned data
save_file(diabetesDF, 'diabetesDF.csv', datapath)
```

```
A file already exists with this name.

Do you want to overwrite? (Y/N)y
Writing file.  "./data\diabetesDF.csv"
```

In [21]:
```
%%time
#2

#pp.ProfileReport(diabetesDF)
```

```
Wall time: 61 ms
```

A Jupyter widget could not be displayed because the widget state could not be found. This could happen if the kernel storing the widget is no longer available, or if the widget state was not saved in the notebook. You may be able to create the widget by running the appropriate cells.

A Jupyter widget could not be displayed because the widget state could not be found. This could happen if the kernel storing the widget is no longer available, or if the widget state was not saved in the notebook. You may be able to create the widget by running the appropriate cells.

A Jupyter widget could not be displayed because the widget state could not be found. This could happen if the kernel storing the widget is no longer available, or if the widget state was not saved in the notebook. You may be able to create the widget by running the appropriate cells.

Overview

Dataset statistics

Number of variables	9
Number of observations	768
Missing cells	0
Missing cells (%)	0.0%
Duplicate rows	0
Duplicate rows (%)	0.0%
Total size in memory	54.1 KiB
Average record size in memory	72.2 B

Variable types

Numeric	8
Categorical	1

Warnings

`preg` has 111 (14.5%) zeros	**Zeros**
`skin` has 227 (29.6%) zeros	**Zeros**

Reproduction

Analysis	2021-04-26 21:10:26.930818

Out[21]:

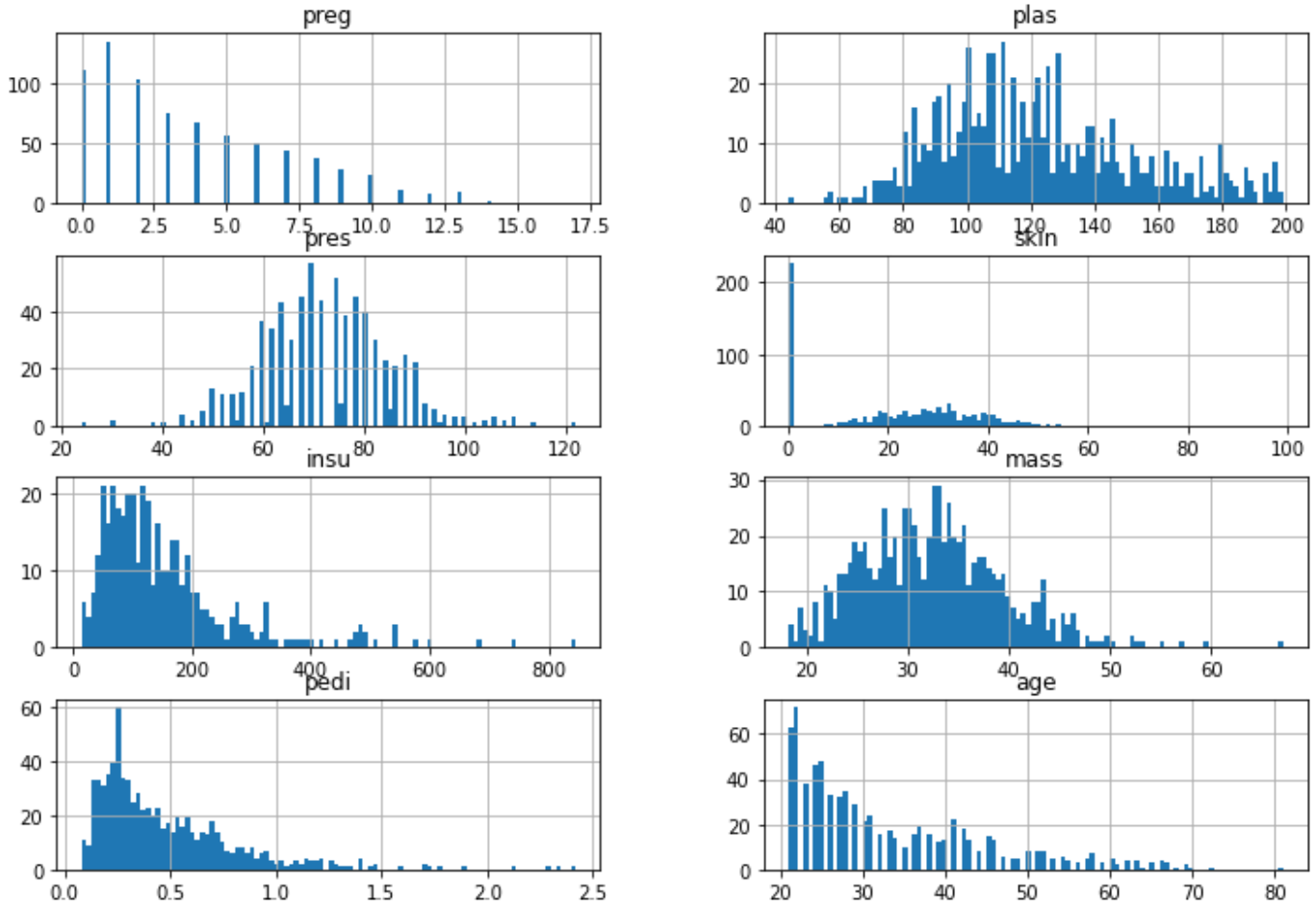
In [22]:
```
diabetesDF.shape,diabetesDF1.shape,diabetesDF2.shape, # Check number of columns and rows in
```

Out[22]:
```
((768, 9), (768, 9), (768, 6))
```

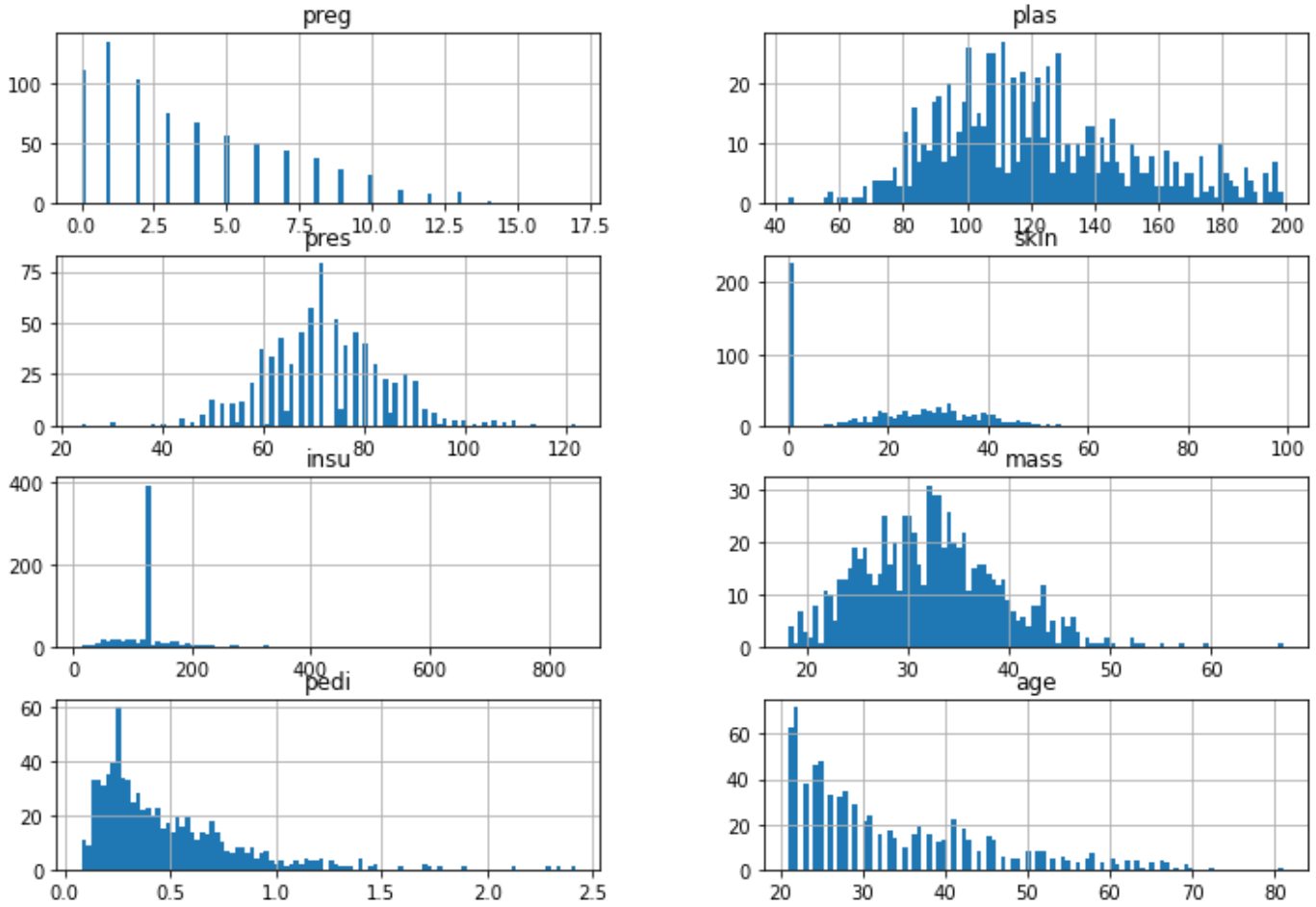
In [23]:
```
diabetesDF.isnull().values.any() # If there are any null values in data set
```

Out[23]:
```
False
```

In [24]:
```
# original dataset
columns = list(diabetesDF1)[0:-1] # Excluding 'class' column which has only
diabetesDF1[columns].hist(stacked=False, bins=100, figsize=(12,30), layout=(14,2));
# Histogram of first 8 columns
```



In [25]:
```
# cleaned dataset
columns = list(diabetesDF)[0:-1] # Excluding 'class' column which has only
diabetesDF[columns].hist(stacked=False, bins=100, figsize=(12,30), layout=(14,2));
# Histogram of first 8 columns
```



Univaraite Analysis

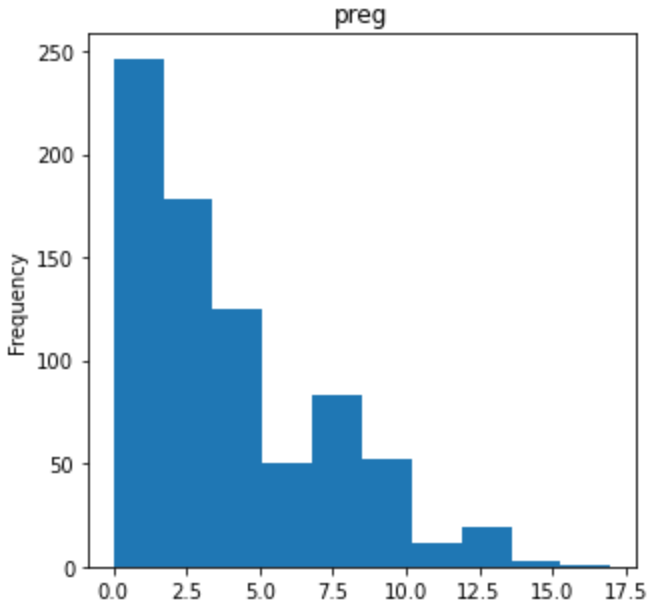
In [26]:
```
diabetesDF.dtypes
```

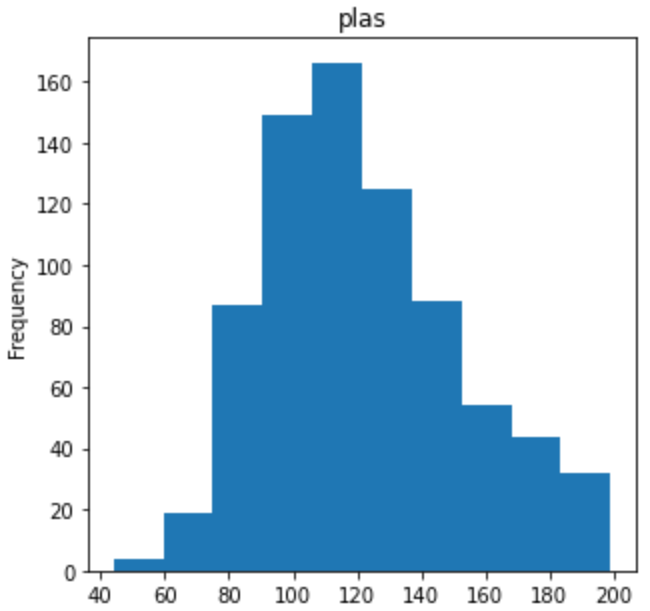
Out[26]:
```
preg       int64
plas     float64
pres     float64
skin       int64
insu     float64
mass     float64
pedi     float64
age        int64
class     object
dtype: object
```

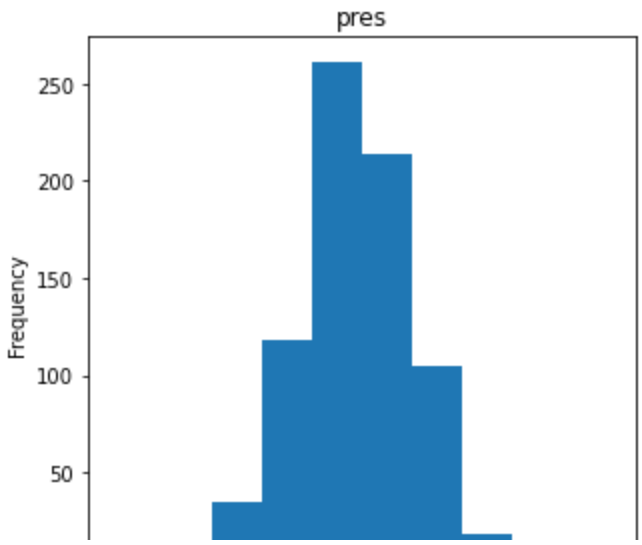
In [27]:
```
# populate the list of numeric attributes and categorical attributes
num_list = []
cat_list = []

for column in diabetesDF:
 if is_numeric_dtype(diabetesDF[column]):
        num_list.append(column)
 elif is_string_dtype(diabetesDF[column]):
        cat_list.append(column)

print("Numerical: ",num_list)
print("Categorical:  ",cat_list)
```

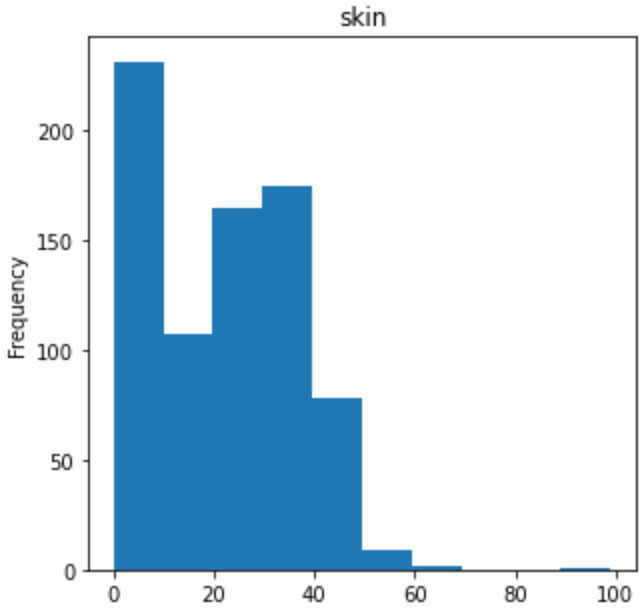
```
Numerical:  ['preg', 'plas', 'pres', 'skin', 'insu', 'mass', 'pedi', 'age']
Categorical:   ['class']
```

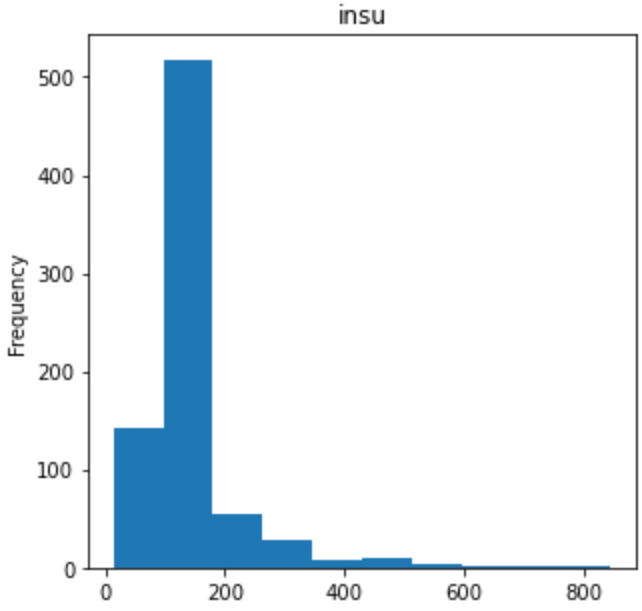
In [28]:
```
# bar chart and histogram
for column in diabetesDF:
    plt.figure(column, figsize = (4.9,4.9))
    plt.title(column)
    if is_numeric_dtype(diabetesDF[column]):
        diabetesDF[column].plot(kind = 'hist')
    elif is_string_dtype(diabetesDF[column]):
 # show only the TOP 10 value count in each categorical data
        diabetesDF[column].value_counts()[:10].plot(kind = 'bar')
```

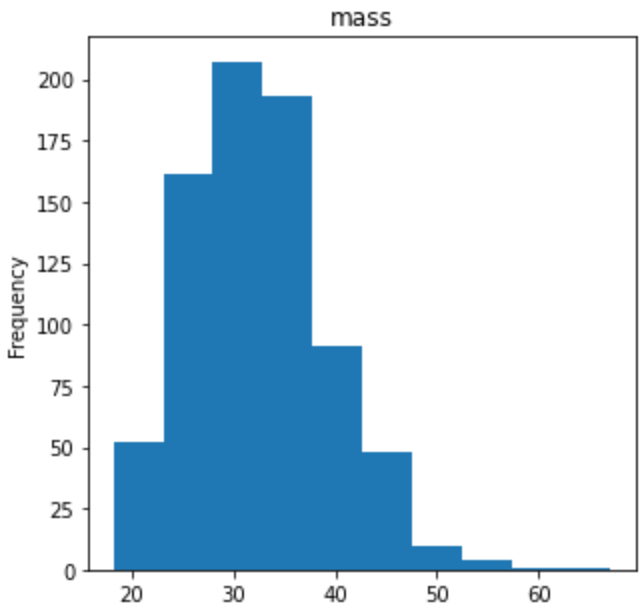


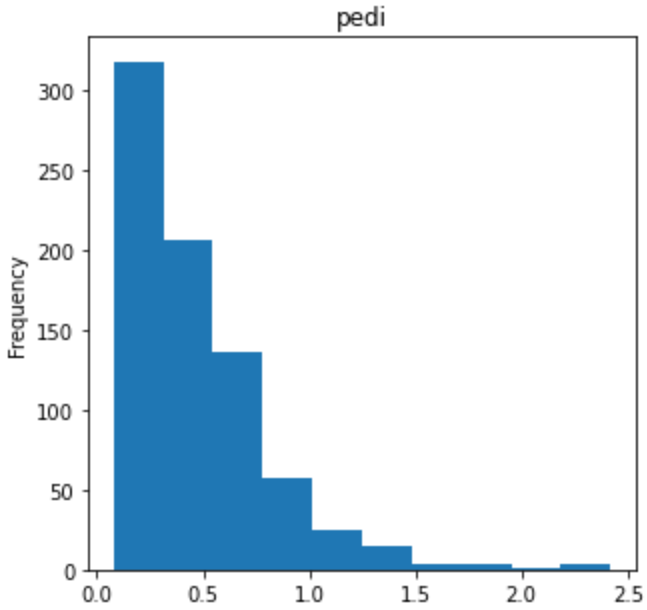


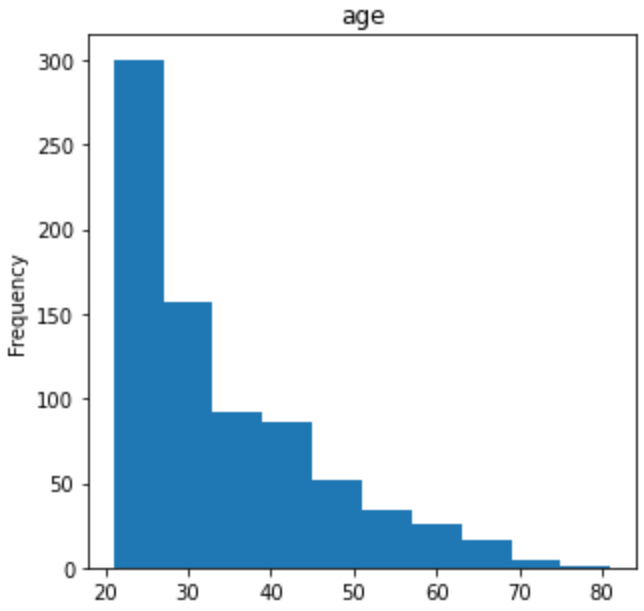

pres
Frequency
250
200
150
100
50

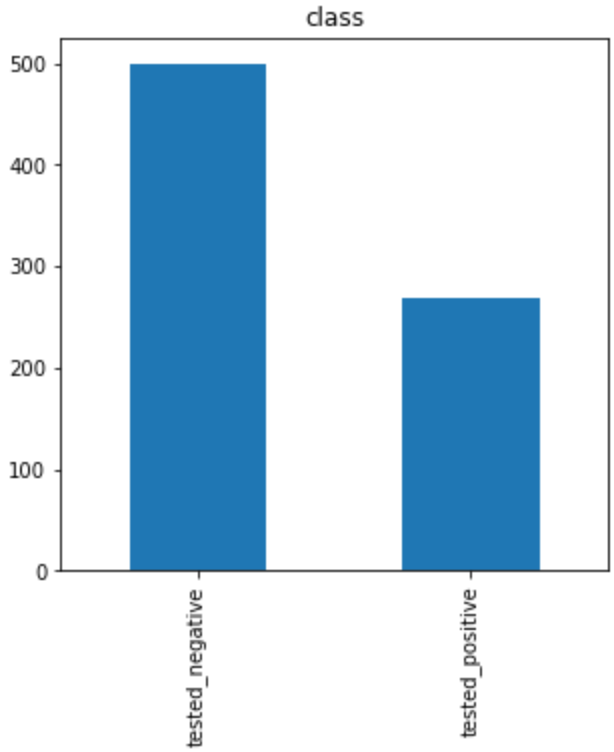

skin
Frequency
200
150
100
50
0
0
20
40
60
80
100


insu
Frequency
500
400
300
200
100
0
0
200
400
600
800


mass
Frequency
200
175
150
125
100
75
50
25
0
20
30
40
50
60


pedi
Frequency
300
250
200
150
100
50
0
0.0
0.5
1.0
1.5
2.0
2.5

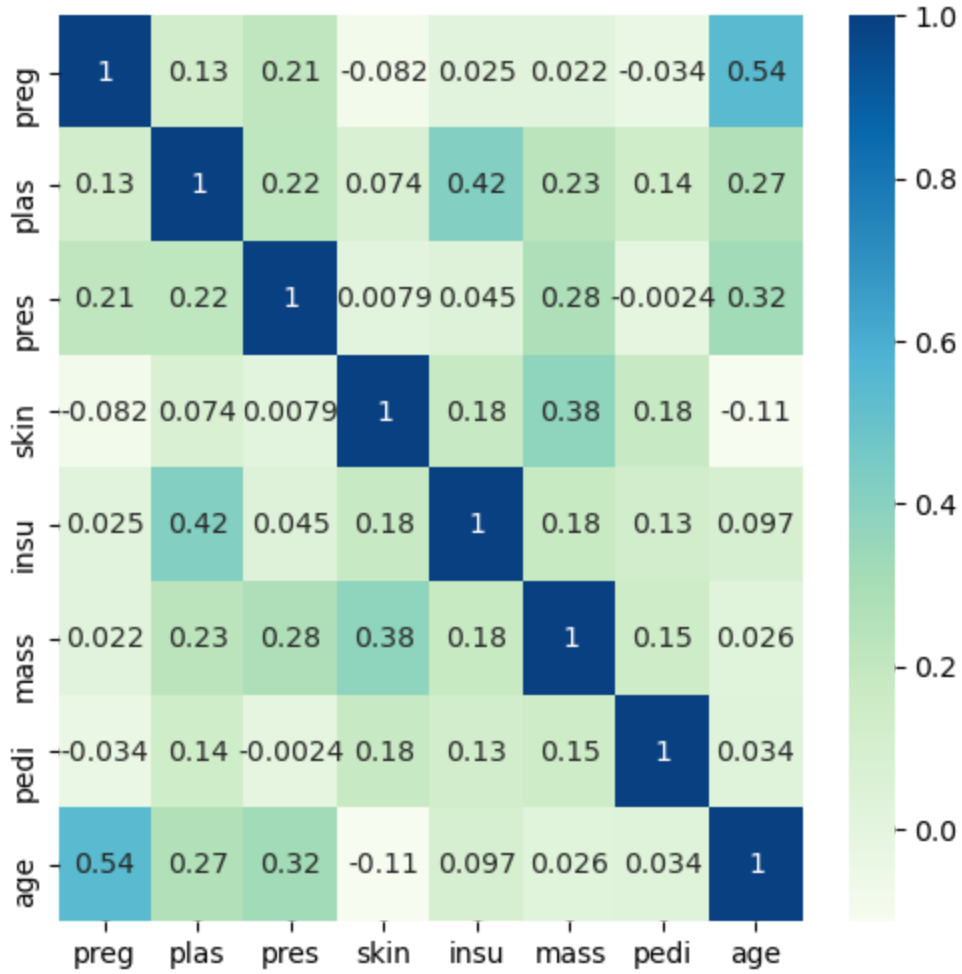

age
Frequency
300
250
200
150
100
50
0
20
30
40
50
60
70
80



Multivariate Analysis

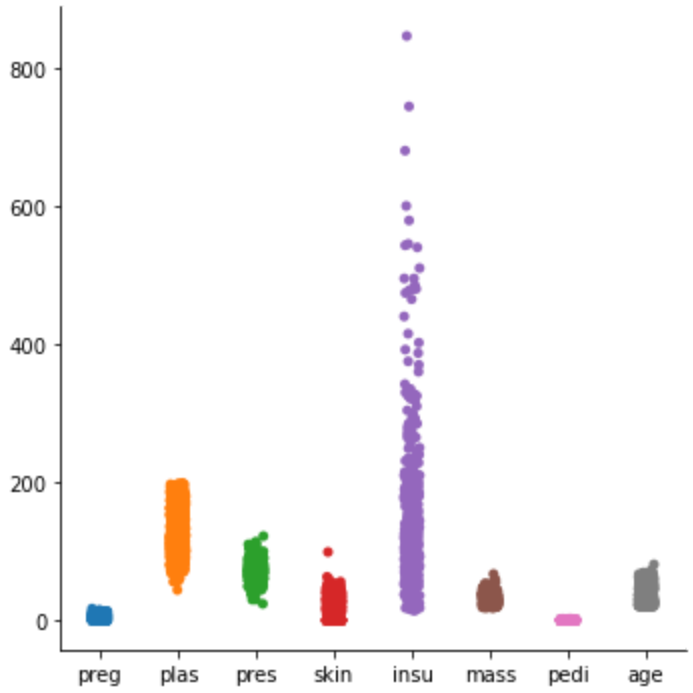
In [29]:
```
## Identify Correlation in data

# correation matrix and heatmap
correlation = diabetesDF.corr()

plt.figure(figsize= (6,6), dpi=100)
sns.heatmap(correlation, cmap = "GnBu", annot = True)
```

Out[29]: <AxesSubplot:>



In [30]:
```
sns.catplot(data=diabetesDF1)
#
```

Out[30]:
```
<seaborn.axisgrid.FacetGrid at 0x26dc80eed30>
```

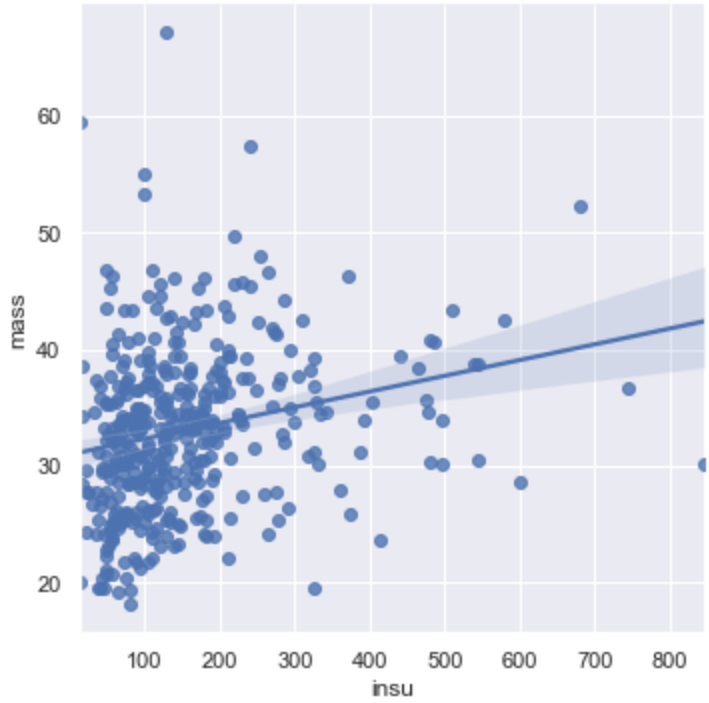


In [31]:
```
diabetesDF1.columns
```

Out[31]:
```
Index(['preg', 'plas', 'pres', 'skin', 'insu', 'mass', 'pedi', 'age', 'class'], dtype='ob
ject')
```

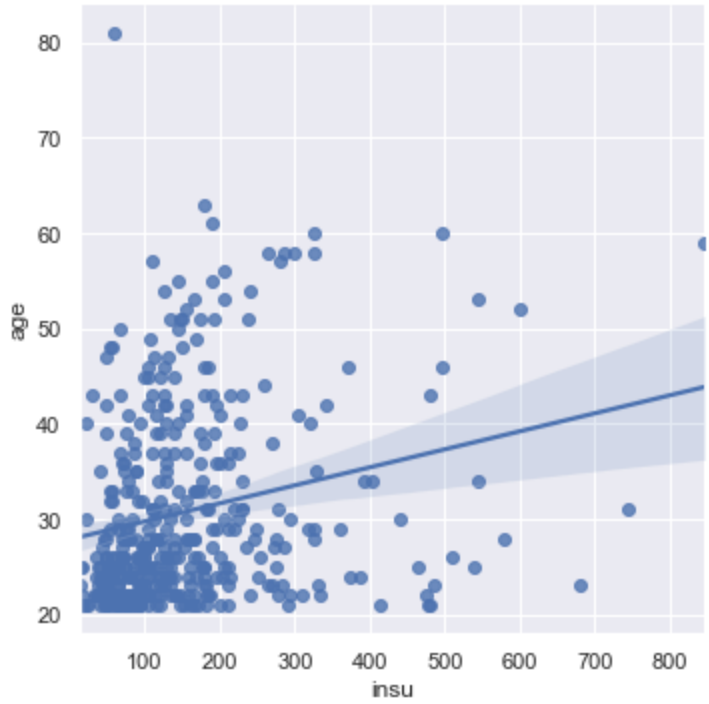
In [32]:
```
import seaborn as sns; sns.set_theme(color_codes=True)
sns.lmplot(x = 'insu', y = 'mass',data = diabetesDF1)
```

Out[32]: <seaborn.axisgrid.FacetGrid at 0x26dc8184640>



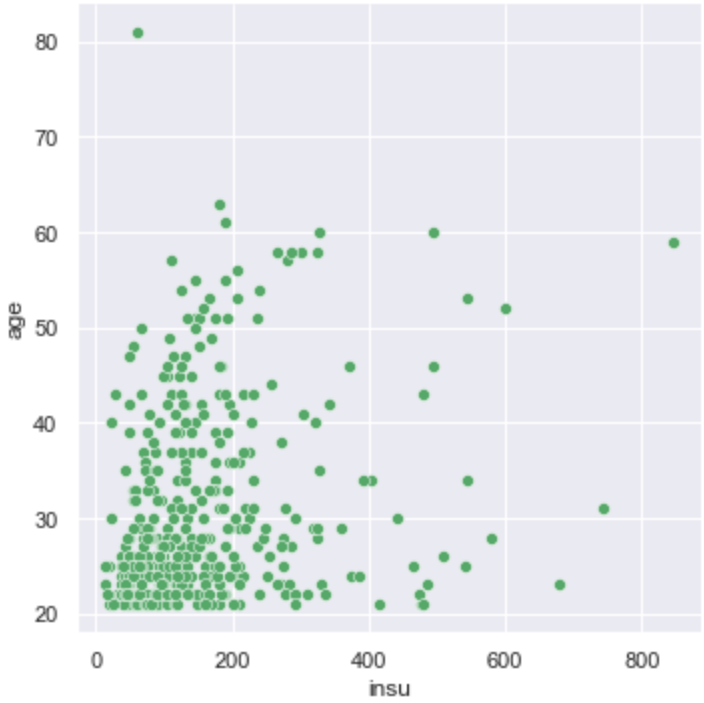
In [33]:
```
sns.lmplot(x = 'insu', y = 'age',data = diabetesDF1)
```

Out[33]: <seaborn.axisgrid.FacetGrid at 0x26dc818f7c0>



In [34]:
```
sns.relplot(x = 'insu', y = 'age',data = diabetesDF1, color = "g")
#https://seaborn.pydata.org/tutorial/regression.html
```

Out[34]: <seaborn.axisgrid.FacetGrid at 0x26dc829ebe0>



In above plot yellow colour represents maximum correlation and blue colour represents minimum correlation. We can see none of variable have correlation with any other variables.

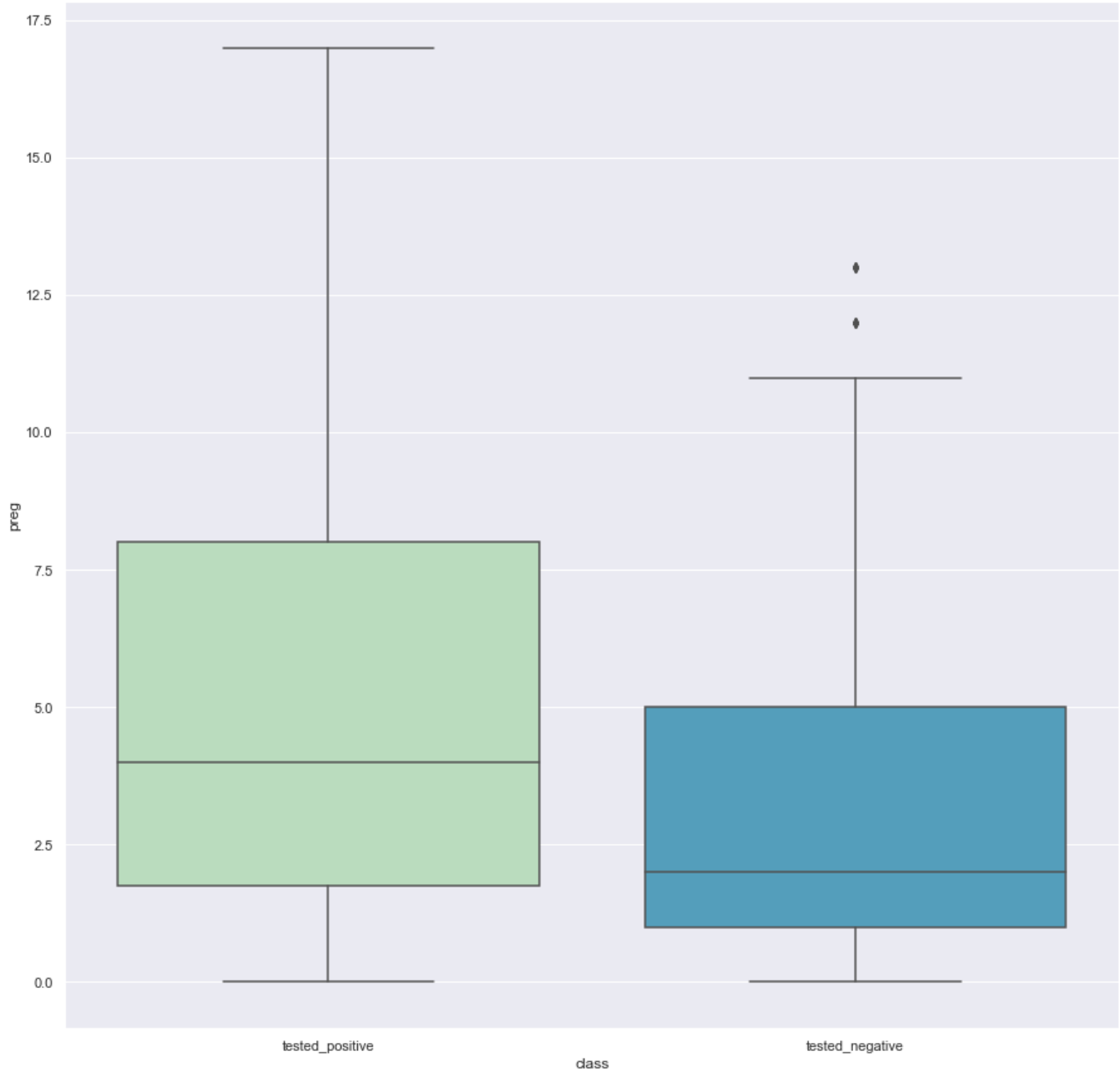
In [35]:
```
sns.pairplot(diabetesDF1,hue = 'class')
```

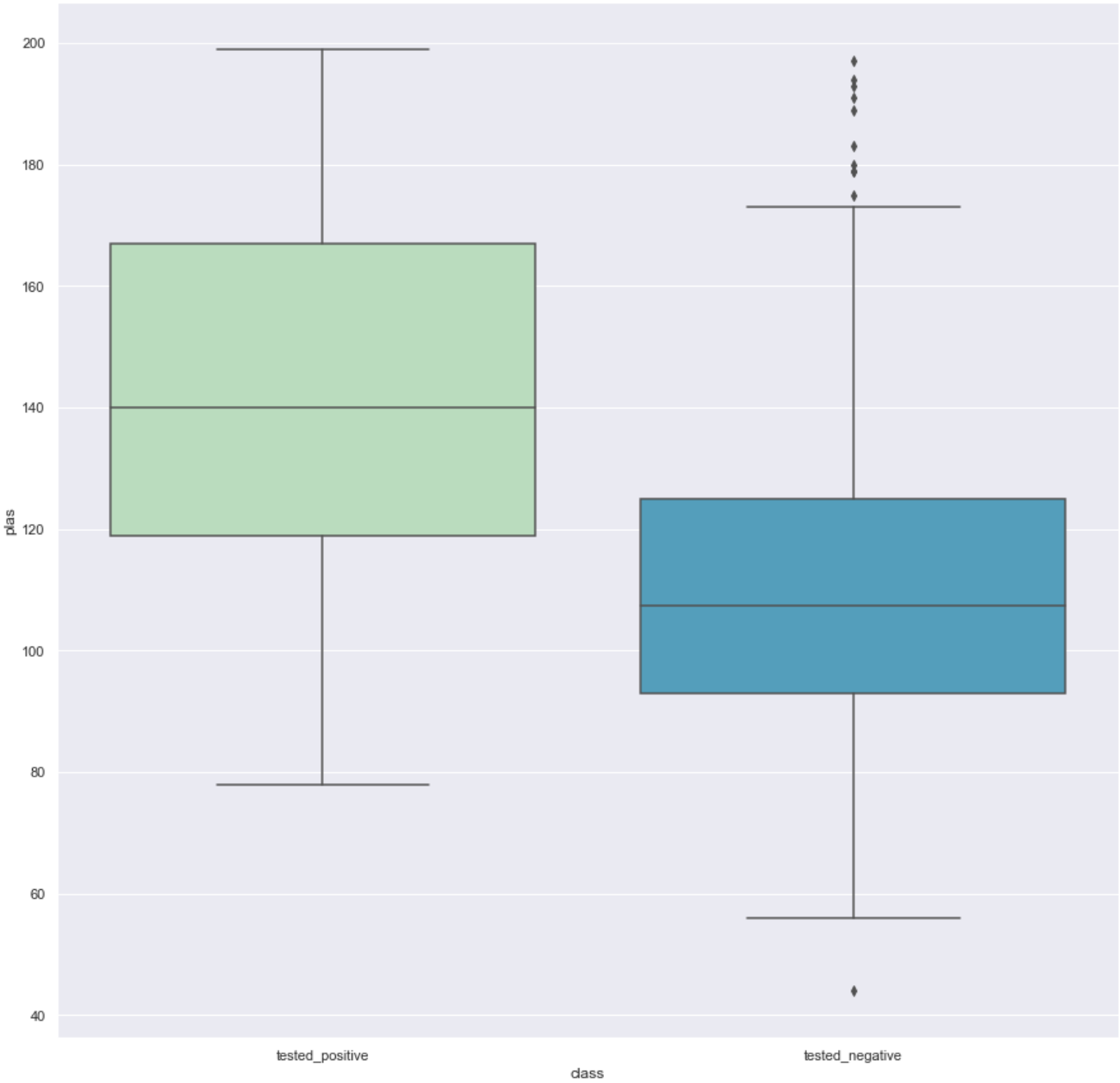
Out[35]: <seaborn.axisgrid.PairGrid at 0x26dc9472820>

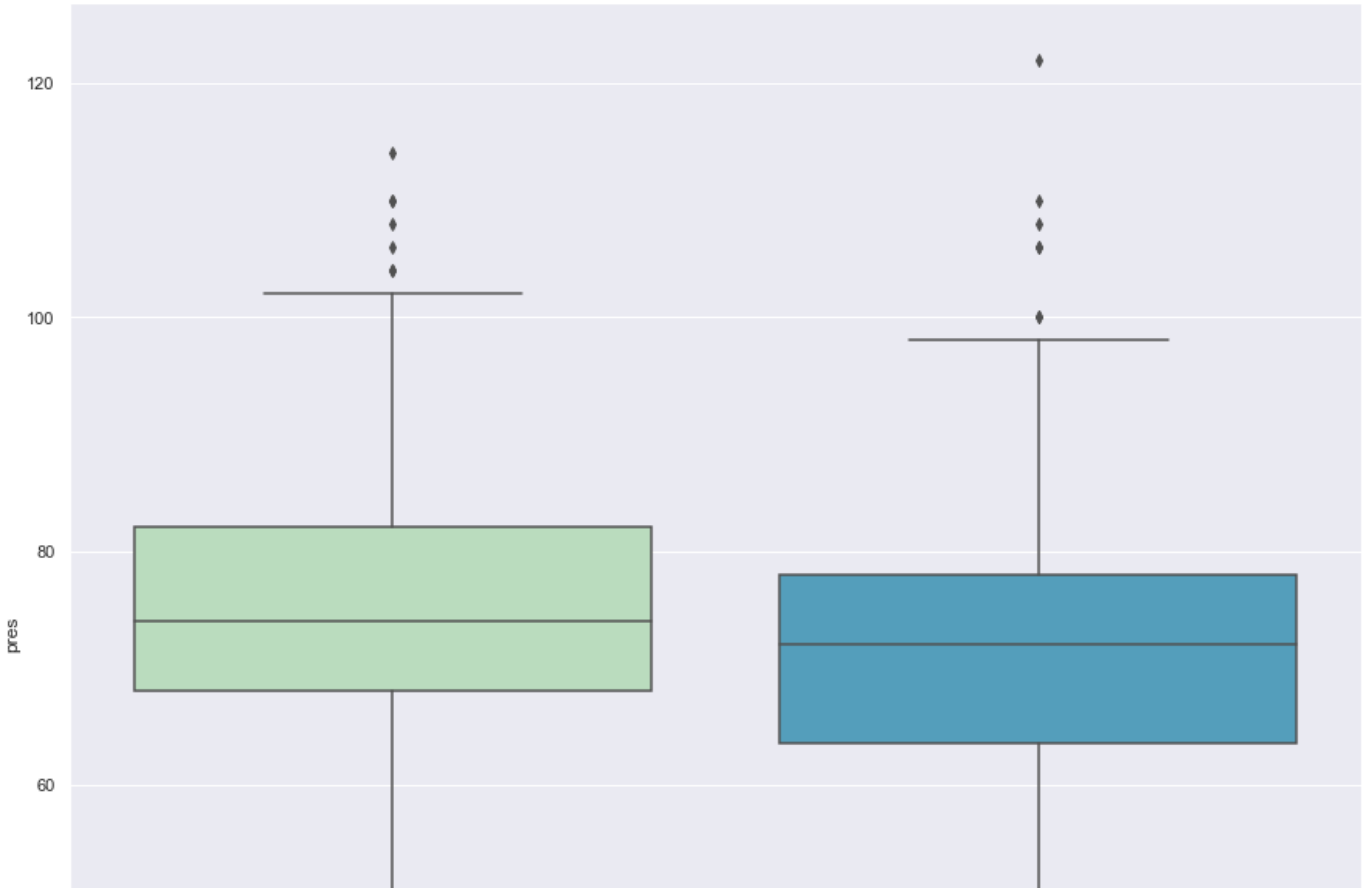
In [36]:
```
sns.pairplot(diabetesDF,diag_kind='kde',hue = 'class')
```

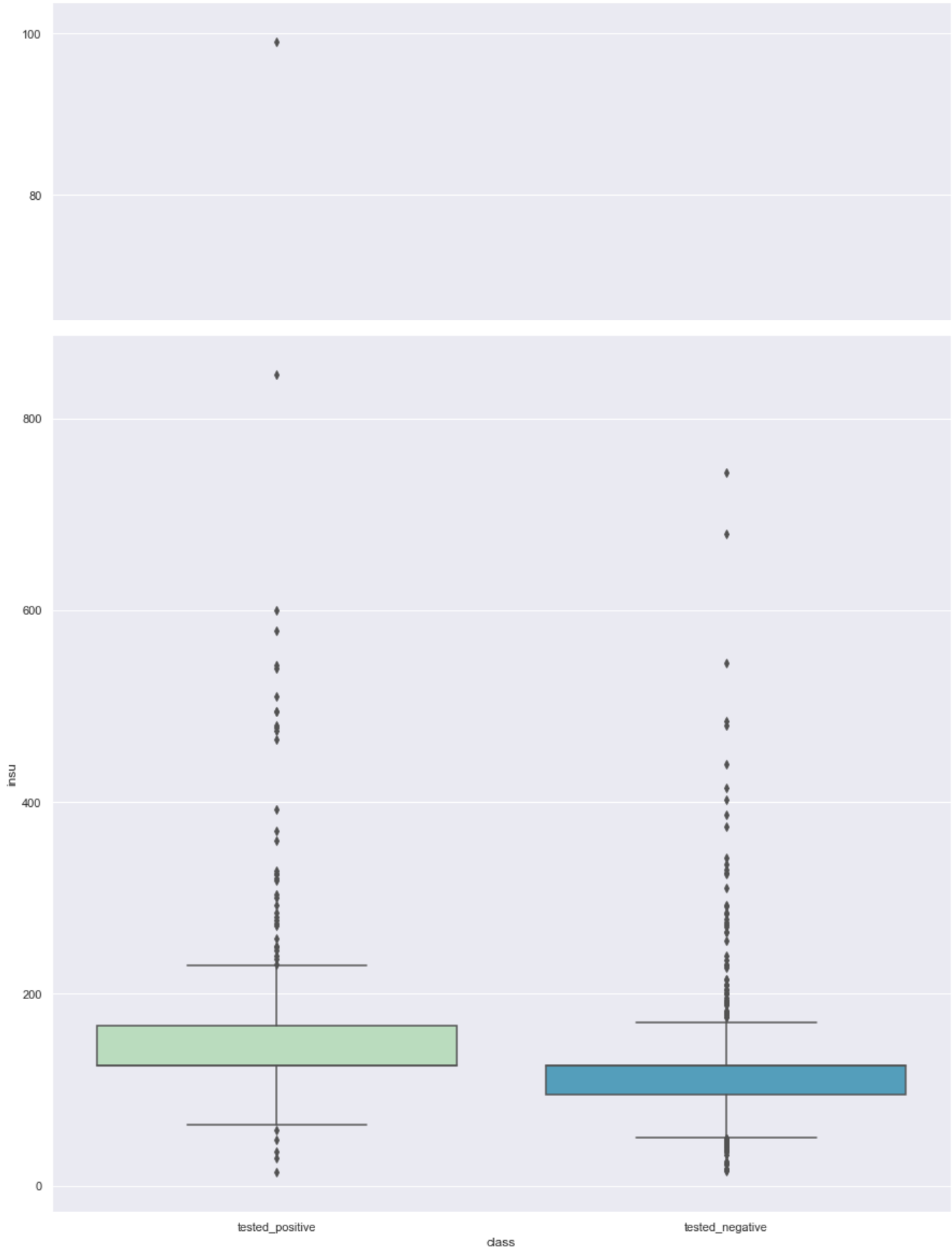
Out[36]: <seaborn.axisgrid.PairGrid at 0x26dc6e66a60>

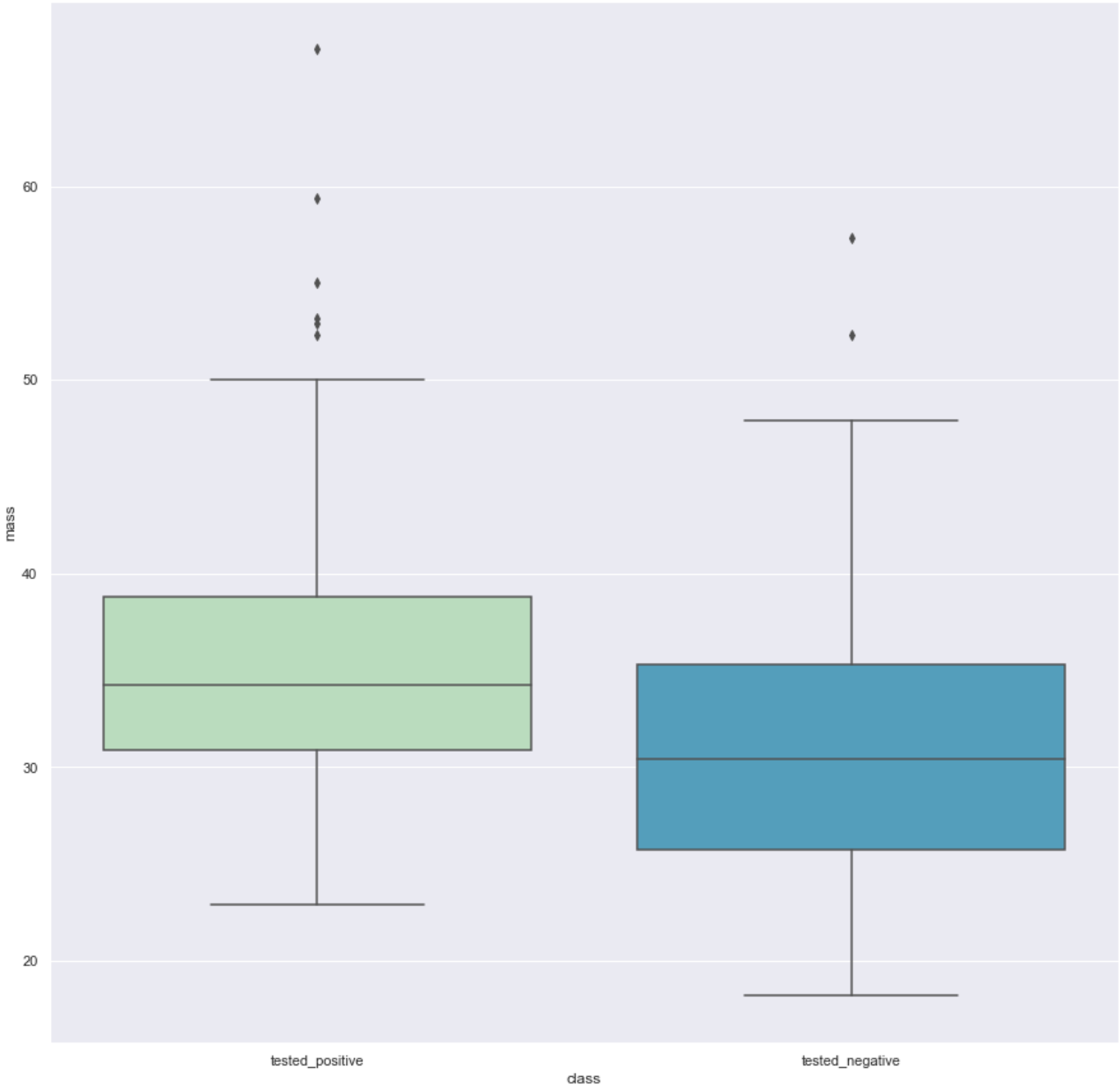
In [37]:
```
# box plot
for i in range(0, len(cat_list)):
    cat = cat_list[i]
    for j in range(0, len(num_list)):
        num = num_list[j]
        plt.figure (figsize = (15,15))
        sns.boxplot( x = cat, y = num, data = diabetesDF, palette = "GnBu")
```

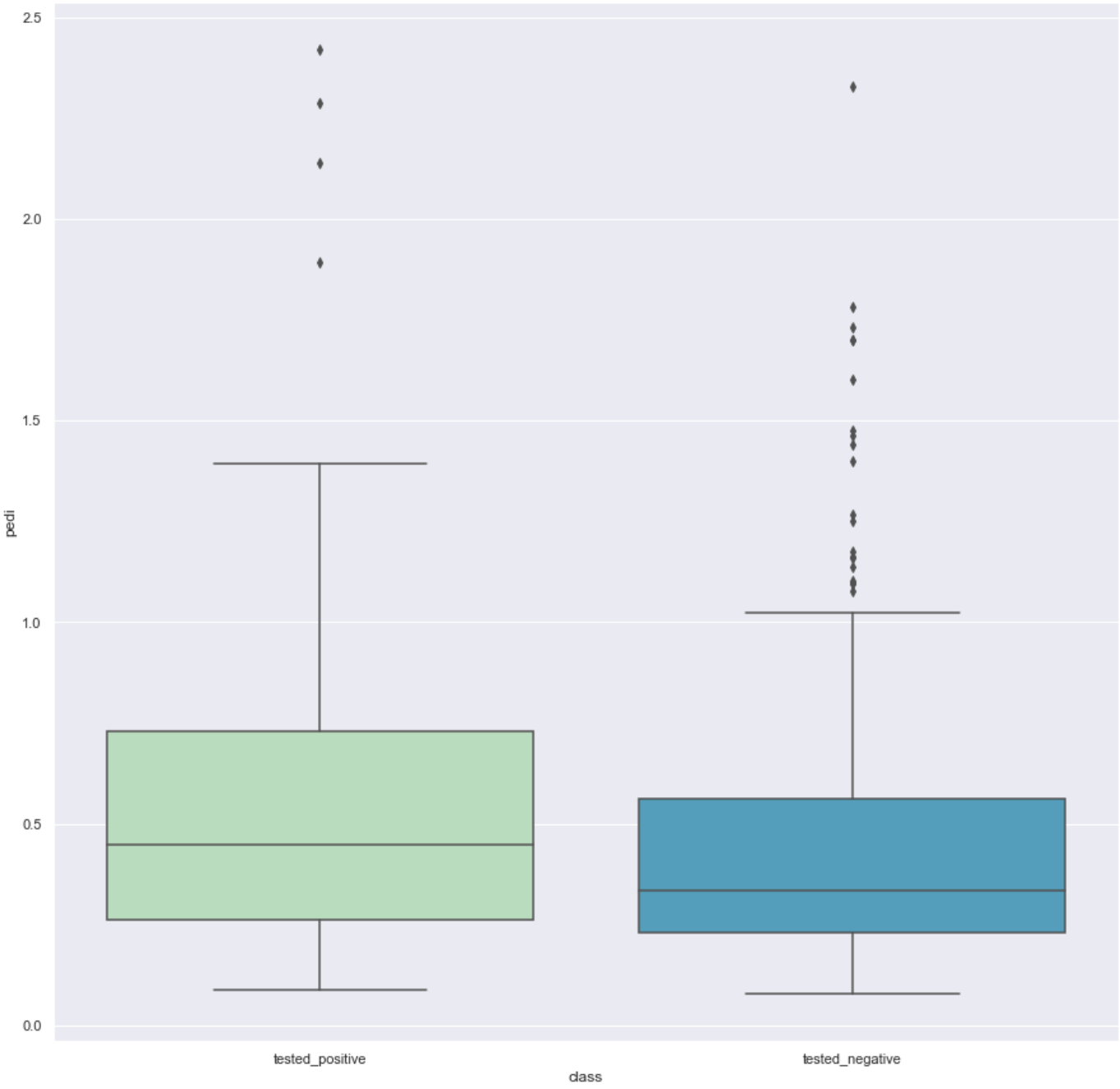


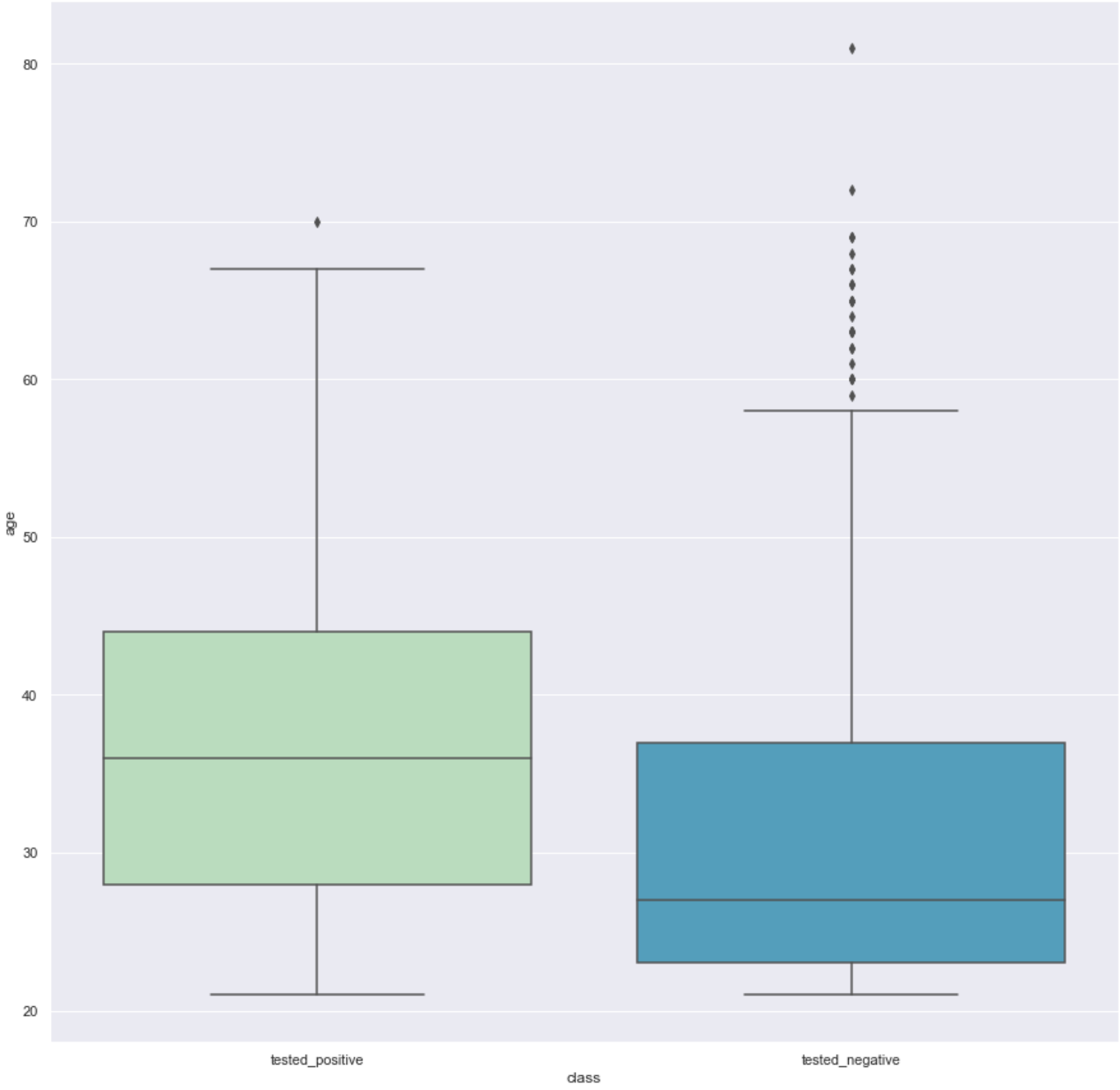

200
180
160
140
120
100
80
60
40
plas
tested_positive
tested_negative
class


120
100
80
60
pres


100
80
800
600
400
200
0
insu
tested_positive
tested_negative
class


60
50
40
30
20
mass
tested_positive
tested_negative
class


2.5
2.0
1.5
1.0
0.5
0.0
pedi
tested_positive
tested_negative
class


80
70
60
50
40
30
20
age
tested_positive
tested_negative
class

finding outliers

In [23]:
```
from scipy import stats
import numpy as np

z = np.abs(stats.zscore(diabetesDF[diabetesDF.select_dtypes(exclude ='object').columns]))

print(z)
```

```
[[0.63994726 0.86604475 0.03198993 ... 0.16661938 0.46849198 1.4259954 ]
 [0.84488505 1.20506583 0.5283186  ... 0.85219976 0.36506078 0.19067191]
 [1.23388019 2.01666174 0.69376149 ... 1.33250021 0.60439732 0.10558415]
 ...
 [0.3429808  0.02157407 0.03198993 ... 0.910418   0.68519336 0.27575966]
 [0.84488505 0.14279979 1.02464727 ... 0.34279019 0.37110101 1.17073215]
 [0.84488505 0.94206766 0.19743282 ... 0.29912651 0.47378505 0.87137393]]
```

In [24]:
```
#An outlier of a dataset is defined as a value that is more than 3 standard deviations fro
diabetes_outlierOUT = diabetesDF[(z <= 3).all(axis=1)]
```

In [25]:
```
diabetes_outlierOUT.shape, diabetesDF.shape
```

Out[25]: ((719, 9), (768, 9))

#An outlier of a dataset is defined as a value that is more than 3 standard deviations from the mean diabetes_WITHoutlier = diabetesDF[((z >= 3) & (z >= -3)).all(axis=1)] diabetes_WITHoutlier.shape

In [27]:
```
import warnings
warnings.filterwarnings('ignore')
diabetesOUT_o1 = diabetesDF

Q1 = diabetesOUT_o1.quantile(0.25)
Q3 = diabetesOUT_o1.quantile(0.75)
IQR = Q3 - Q1
print(IQR)

diabetes_outliersONLY = diabetesOUT_o1[((diabetesOUT_o1 < (Q1 - 25 * IQR)) | (diabetesOUT_

diabetes_outliersONLY.shape
```

```
preg     5.0000
plas    40.5000
pres    16.0000
skin    32.0000
insu     5.7500
mass     9.1000
pedi     0.3825
age     17.0000
dtype: float64
```

Out[27]: (50, 9)

In [28]:
```
#3
#pp.ProfileReport(diabetes_outliersONLY)
```

A Jupyter widget could not be displayed because the widget state could not be found. This could happen if the kernel storing the widget is no longer available, or if the widget state was not saved in the notebook. You may be able to create the widget by running the appropriate cells.

A Jupyter widget could not be displayed because the widget state could not be found. This could happen if the kernel storing the widget is no longer available, or if the widget state was not saved in the notebook. You may be able to create the widget by running the appropriate cells.

A Jupyter widget could not be displayed because the widget state could not be found. This could happen if the kernel storing the widget is no longer available, or if the widget state was not saved in the notebook. You may be able to create the widget by running the appropriate cells.

Overview

Dataset statistics

Number of variables	10
Number of observations	50
Missing cells	0
Missing cells (%)	0.0%
Duplicate rows	0
Duplicate rows (%)	0.0%
Total size in memory	4.0 KiB
Average record size in memory	82.6 B

Variable types

Numeric	9
Categorical	1

Warnings

`df_index` has unique values	**Unique**
`preg` has 7 (14.0%) zeros	**Zeros**

Reproduction

Analysis	2021-04-26 21:14:40.865743

Out[28]:

In [29]:
```
diabetes_outliersONLY
```

Out[29]:

	preg	plas	pres	skin	insu	mass	pedi	age	class
8	2	197.0	70.0	45	543.0	30.5	0.158	53	tested_positive
13	1	189.0	60.0	23	846.0	30.1	0.398	59	tested_positive
53	8	176.0	90.0	34	300.0	33.7	0.467	58	tested_positive
54	7	150.0	66.0	42	342.0	34.7	0.718	42	tested_negative
56	7	187.0	68.0	39	304.0	37.7	0.254	41	tested_positive
111	8	155.0	62.0	26	495.0	34.0	0.543	46	tested_positive
139	5	105.0	72.0	29	325.0	36.9	0.159	28	tested_negative
144	4	154.0	62.0	31	284.0	32.8	0.237	23	tested_negative
153	1	153.0	82.0	42	485.0	40.6	0.687	23	tested_negative
162	0	114.0	80.0	34	285.0	44.2	0.167	27	tested_negative
186	8	181.0	68.0	36	495.0	30.1	0.615	60	tested_positive
199	4	148.0	60.0	27	318.0	30.9	0.150	29	tested_positive
206	8	196.0	76.0	29	280.0	37.5	0.605	57	tested_positive
220	0	177.0	60.0	29	478.0	34.6	1.072	21	tested_positive
228	4	197.0	70.0	39	744.0	36.7	2.329	31	tested_negative
231	6	134.0	80.0	37	370.0	46.2	0.238	46	tested_positive
247	0	165.0	90.0	33	680.0	52.3	0.427	23	tested_negative
248	9	124.0	70.0	33	402.0	35.4	0.282	34	tested_negative
258	1	193.0	50.0	16	375.0	25.9	0.655	24	tested_negative
279	2	108.0	62.0	10	278.0	25.3	0.881	22	tested_negative
286	5	155.0	84.0	44	545.0	38.7	0.619	34	tested_negative
296	2	146.0	70.0	38	360.0	28.0	0.337	29	tested_positive
360	5	189.0	64.0	33	325.0	31.2	0.583	29	tested_positive
364	4	147.0	74.0	25	293.0	34.9	0.385	30	tested_negative
370	3	173.0	82.0	48	465.0	38.4	2.137	25	tested_positive
375	12	140.0	82.0	43	325.0	39.2	0.528	58	tested_positive
388	5	144.0	82.0	26	285.0	32.0	0.452	58	tested_positive
392	1	131.0	64.0	14	415.0	23.7	0.389	21	tested_negative
395	2	127.0	58.0	24	275.0	27.7	1.600	25	tested_negative
409	1	172.0	68.0	49	579.0	42.4	0.702	28	tested_positive
412	1	143.0	84.0	23	310.0	42.4	1.076	22	tested_negative
415	3	173.0	84.0	33	474.0	35.7	0.258	22	tested_positive
425	4	184.0	78.0	39	277.0	37.0	0.264	31	tested_positive
480	3	158.0	70.0	30	328.0	35.5	0.344	35	tested_positive
486	1	139.0	62.0	41	480.0	40.7	0.536	21	tested_negative
519	6	129.0	90.0	7	326.0	19.6	0.582	60	tested_negative
561	0	198.0	66.0	32	274.0	41.3	0.502	28	tested_positive

	preg	plas	pres	skin	insu	mass	pedi	age	class
574	1	143.0	86.0	30	330.0	30.1	0.892	23	tested_negative
584	8	124.0	76.0	24	600.0	28.7	0.687	52	tested_positive
606	1	181.0	78.0	42	293.0	40.0	1.258	22	tested_positive
608	0	152.0	82.0	39	272.0	41.5	0.270	27	tested_negative
612	7	168.0	88.0	42	321.0	38.2	0.787	40	tested_positive
645	2	157.0	74.0	35	440.0	39.4	0.134	30	tested_negative
655	2	155.0	52.0	27	540.0	38.7	0.240	25	tested_positive
695	7	142.0	90.0	24	480.0	30.4	0.128	43	tested_positive
707	2	127.0	46.0	21	335.0	34.4	0.176	22	tested_negative
710	3	158.0	64.0	13	387.0	31.2	0.295	24	tested_negative
713	0	134.0	58.0	20	291.0	26.4	0.352	21	tested_negative
715	7	187.0	50.0	33	392.0	33.9	0.826	34	tested_positive
753	0	181.0	88.0	44	510.0	43.3	0.222	26	tested_positive

diabetes_outliersONLY : all these are accepted as valuable data

In [44]:
```
#4
#pp.ProfileReport(diabetes_outlierOUT)
```

A Jupyter widget could not be displayed because the widget state could not be found. This could happen if the kernel storing the widget is no longer available, or if the widget state was not saved in the notebook. You may be able to create the widget by running the appropriate cells.

A Jupyter widget could not be displayed because the widget state could not be found. This could happen if the kernel storing the widget is no longer available, or if the widget state was not saved in the notebook. You may be able to create the widget by running the appropriate cells.

A Jupyter widget could not be displayed because the widget state could not be found. This could happen if the kernel storing the widget is no longer available, or if the widget state was not saved in the notebook. You may be able to create the widget by running the appropriate cells.

Overview

Dataset statistics

Number of variables	10
Number of observations	719
Missing cells	0
Missing cells (%)	0.0%
Duplicate rows	0
Duplicate rows (%)	0.0%
Total size in memory	56.3 KiB
Average record size in memory	80.2 B

Variable types

Numeric	9
Categorical	1

Warnings

`df_index` is uniformly distributed	**Uniform**
`df_index` has unique values	**Unique**
`preg` has 101 (14.0%) zeros	**Zeros**
`skin` has 221 (30.7%) zeros	**Zeros**

Out[44]:

++

diabetes types classification

In [30]:
```
t1_t2 = diabetesDF[['preg', 'plas', 'pres', 'skin', 'insu', 'mass', 'pedi', 'age']].astype(
t1_t2_clean = t1_t2[ (diabetesDF.pres > 0) & (diabetesDF.mass > 0) ]
t1_t2_clean
```

Out[30]:

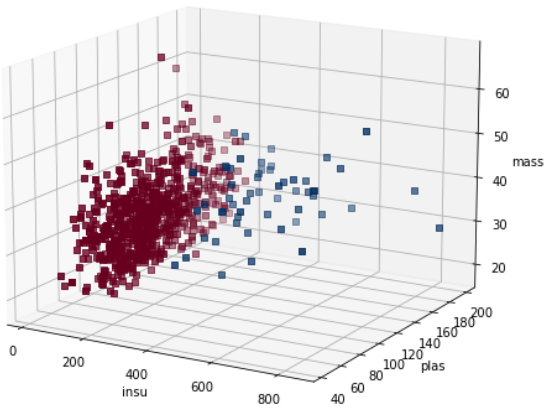
	preg	plas	pres	skin	insu	mass	pedi	age
0	6.0	148.0	72.0	35.0	125.0	33.6	0.627	50.0
1	1.0	85.0	66.0	29.0	125.0	26.6	0.351	31.0
2	8.0	183.0	64.0	0.0	125.0	23.3	0.672	32.0
3	1.0	89.0	66.0	23.0	94.0	28.1	0.167	21.0
4	0.0	137.0	40.0	35.0	168.0	43.1	2.288	33.0
...	...	...	...	...	...	...	...	...
763	10.0	101.0	76.0	48.0	180.0	32.9	0.171	63.0
764	2.0	122.0	70.0	27.0	125.0	36.8	0.340	27.0
765	5.0	121.0	72.0	23.0	112.0	26.2	0.245	30.0
766	1.0	126.0	60.0	0.0	125.0	30.1	0.349	47.0
767	1.0	93.0	70.0	31.0	125.0	30.4	0.315	23.0

768 rows × 8 columns

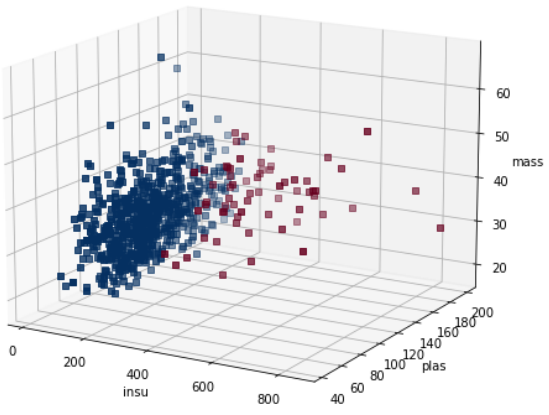
In [31]:

```
from sklearn.cluster import KMeans, AgglomerativeClustering

"""
Clustering to distinguish between type - I and type - II
"""

# Use standard K-Means with two clusters (type I/II) as a first step
kmeans = KMeans(n_clusters=2).fit(t1_t2_clean)

# For comparison, also use Agglomerative Clustering with two clusters (type I/II)
agglo = AgglomerativeClustering(n_clusters=2).fit(t1_t2_clean)


def plot_3D(df, label1, label2, title1, title2):
    #---- Parameters which are the same for both figures
    fig = plt.figure(figsize=(20,8))
    x = np.array(diabetesDF['insu'])
    y = np.array(diabetesDF['plas'])
    z = np.array(diabetesDF['mass'])

    #---- First subplot
    ax = fig.add_subplot(121, projection='3d')

    ax.scatter(x, y, z, marker="s", c=label1, s=25, cmap="RdBu")
    ax.set_title("%s" % title1, fontsize = 16)
    ax.set_xlabel('insu')
    ax.set_ylabel('plas')
    ax.set_zlabel('mass')
    ax.view_init(15) # rotation of the 3D plot

    #---- Second subplot
    ax = fig.add_subplot(122, projection='3d')

    ax.scatter(x, y, z, marker="s", c=label2, s=25, cmap="RdBu")
    ax.set_title("%s" % title2, fontsize = 16)
    ax.set_xlabel('insu')
    ax.set_ylabel('plas')
    ax.set_zlabel('mass')
    ax.view_init(15) # rotation of the 3D plot

    plt.show()


# Visualize the results of both clustering algorithms in 3D
plot_3D(t1_t2_clean, kmeans.labels_, agglo.labels_, "K-Means Clustering", "Agglomerative C
```

K-Means Clustering

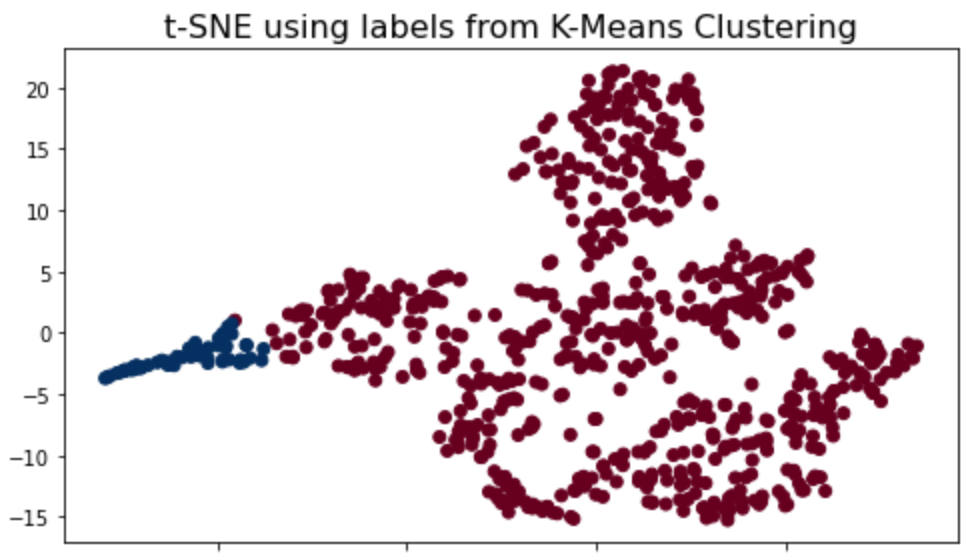


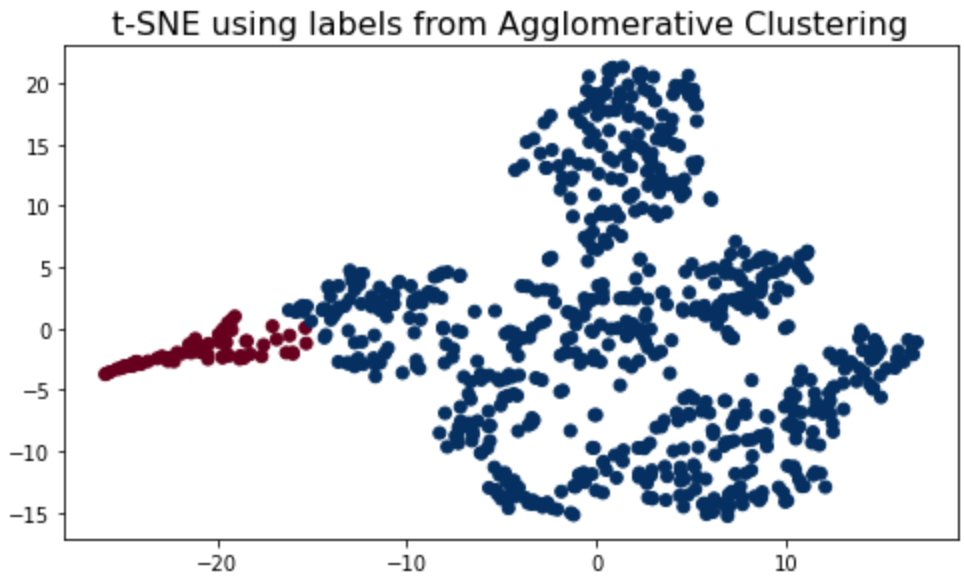
Agglomerative Clustering



In [32]:

```
from sklearn.manifold import TSNE

"""
Use dimensionality reduction to visualize the separation of the two clusters
"""

# Perform tSNE on the values with two clusters (type I/II) as a first step
tsne = TSNE(n_components=2, n_iter=300).fit_transform(t1_t2_clean)



def plot_TSNE(df, label1, label2, title1, title2):
    fig, ax = plt.subplots(2, sharex=True, figsize=(8,10))

    ax[0].scatter(df[:,0], df[:,1], c = label1, cmap="RdBu")
    ax[0].set_title('t-SNE using labels from %s Clustering' % title1, fontsize = 16)

    ax[1].scatter(df[:,0], df[:,1], c = label2, cmap="RdBu")
    ax[1].set_title('t-SNE using labels from %s Clustering' % title2, fontsize = 16)

    fig.subplots_adjust(hspace=0.2)
    plt.show()



plot_TSNE(tsne, kmeans.labels_, agglo.labels_, "K-Means", "Agglomerative")
```

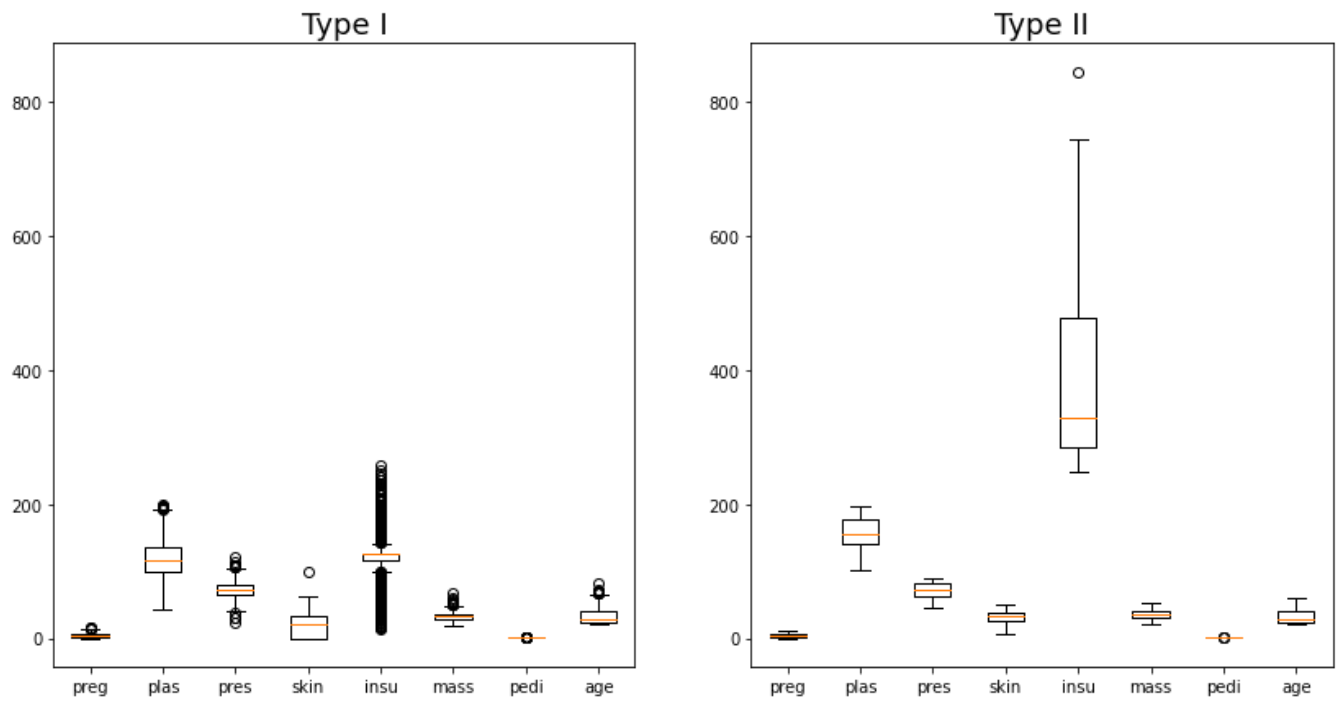

t-SNE using labels from K-Means Clustering
20
15
10
5
0
−5
−10
−15


t-SNE using labels from Agglomerative Clustering
20
15
10
5
0
−5
−10
−15
−20
−10
0
10

In [33]:
```
# Find the average BMI, BloodPressure and Age with their STD of each class; output a table


def print_tab(df, label):
    # Use the labels of KMeans, as they gave a better separation.
    mean1 = df[label == 0].mean(axis=0)
    std1 = df[label == 0].std(axis=0)

    mean2 = df[label == 1].mean(axis=0)
    std2 = df[label == 1].std(axis=0)

    print("")
    print("Kmeans clustering")

    print("")


    # Print the values in a table
    print( tabulate([['Type - I', "%i +/- %i"%(mean1[0],std1[0]), "%i +/- %i"%(mean1[1],st
                      ['Type - II', "%i +/- %i"%(mean2[0],std2[0]), "%i +/- %i"%(mean2[1],s
                     headers=list(df), tablefmt="fancy_grid") )


print_tab(t1_t2_clean, kmeans.labels_)




def plot_boxplot(df, label):
    # Create a figure instance
    fig = plt.figure(figsize=(14,7))

    # Create the subplots, sharing the y axis
    ax1 = plt.subplot(121)
    plt.boxplot(df[label == 0].values)
    plt.setp(ax1, xticklabels=list(df))
    plt.title('Type I', fontsize=18)

    ax2 = plt.subplot(122, sharey=ax1)
    plt.boxplot(df[label == 1].values)
    plt.setp(ax2, xticklabels=list(df))
    plt.title('Type II', fontsize=18)

    plt.show()


# Visualize the result in a boxplot!
plot_boxplot(t1_t2_clean, kmeans.labels_)
```

```
Kmeans clustering
```

	preg	plas	pres	skin
Type - I	3 +/- 3	118 +/- 29	72 +/- 12
Type - II	3 +/- 3	156 +/- 25	72 +/- 11



Insulin resistance often develops before type 2 diabetes. At first, insulin resistance causes the body to make extra insulin, to make up for ineffective insulin. Extra insulin in the bloodstream can cause hypoglycemia. But insulin resistance tends to get worse over time. Eventually, it decreases your body's ability to make insulin. As insulin levels drop, blood sugar levels rise. If levels don't return to normal, you may get type 2 diabetes.

ref. https://medlineplus.gov/lab-tests/insulin-in-blood/ (https://medlineplus.gov/lab-tests/insulin-in-blood/)

Summary

Using only the features BloodPressure, BMI and Age, it is possible to visualize the data. Using tSNE on the data leads to a separation of the points in two dimensions; however, it is still not possible to clearly separate the data into certain clusters.

In order to reveal the difference between Diabetes Type I and Type II, the data is clustered with both KMeans and Agglomerative Clustering, using two as the number of clusters. KMeans is further used, as it manages to separate the data better than Agglomerative Clustering.

The boxplots as well as the tables show the differences between the two clusters well. As expected, one of the classes has higher values for the blood pressure, the BMI and the age. This is in accordance to the table above, which shows the major causes for Diabetes Type II.

Before jumping to quick conclusions, it must be mentioned that the separation of the data into two clusters might have other causes. In order to analyze those potential causes, one could do further analysis with different number of clusters; like that, it is possible to find out the optimal number of clusters. Alternatively, having a dataset with labels for Diabetes Type I and Type II would increase the possibilities for analysis drastically. To concentrate on other aspects of this dataset, I will not cover those approaches here.

++++++++++++++++++++++++++++++++++++

PyCaret Classification

MODEL 0: The "Kitchen Sink" approach

predicting on the test dataset to record 2 main outcomes:

1. Capture feature importances from the fit procedure. The plan is to use the most important features later on with further preprocessing and parameter tuning
2. Capture classification results of this "base model" to compare future models and see which ones perform best. Compare different models based on Accuracy, AUC, Recall, Prec., F1, Kappa, MCC,

In [34]:
```
dataset_pycaret = diabetesDF.copy()
dataset_pycaret.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   preg    768 non-null    int64
 1   plas    768 non-null    float64
 2   pres    768 non-null    float64
 3   skin    768 non-null    int64
 4   insu    768 non-null    float64
 5   mass    768 non-null    float64
 6   pedi    768 non-null    float64
 7   age     768 non-null    int64
 8   class   768 non-null    object
dtypes: float64(5), int64(3), object(1)
memory usage: 54.1+ KB
```

In [35]:
```
dataset_pycaret.columns
```

Out[35]:
```
Index(['preg', 'plas', 'pres', 'skin', 'insu', 'mass', 'pedi', 'age', 'class'], dtype='ob
ject')
```

Experiment Setup

In [36]:
```
clf1 = setup(dataset_pycaret,
             target = 'class', # Replace with your target variable.
             session_id=123,
             log_experiment=True,
             experiment_name='diabetes ML', # Optionally replace with your desired experime
             silent=True # Runs the command without waiting for user input.
            )
```

	Description	Value
0	session_id	123
1	Target	class
2	Target Type	Binary
3	Label Encoded	tested_negative: 0, tested_positive: 1
4	Original Data	(768, 9)
5	Missing Values	False
6	Numeric Features	7
7	Categorical Features	1
8	Ordinal Features	False
9	High Cardinality Features	False
10	High Cardinality Method	None

In [44]:
```
### Compare Baseline Models
models()
```

Out[44]:

ID	Name	Reference	Turbo
lr	Logistic Regression	sklearn.linear_model._logistic.LogisticRegression	True
knn	K Neighbors Classifier	sklearn.neighbors._classification.KNeighborsCl...	True
nb	Naive Bayes	sklearn.naive_bayes.GaussianNB	True
dt	Decision Tree Classifier	sklearn.tree._classes.DecisionTreeClassifier	True
svm	SVM - Linear Kernel	sklearn.linear_model._stochastic_gradient.SGDC...	True
rbfsvm	SVM - Radial Kernel	sklearn.svm._classes.SVC	False
gpc	Gaussian Process Classifier	sklearn.gaussian_process._gpc.GaussianProcessC...	False
mlp	MLP Classifier	sklearn.neural_network._multilayer_perceptron....	False
ridge	Ridge Classifier	sklearn.linear_model._ridge.RidgeClassifier	True
rf	Random Forest Classifier	sklearn.ensemble._forest.RandomForestClassifier	True
qda	Quadratic Discriminant Analysis	sklearn.discriminant_analysis.QuadraticDiscrim...	True
ada	Ada Boost Classifier	sklearn.ensemble._weight_boosting.AdaBoostClas...	True
gbc	Gradient Boosting Classifier	sklearn.ensemble._gb.GradientBoostingClassifier	True
lda	Linear Discriminant Analysis	sklearn.discriminant_analysis.LinearDiscrimina...	True
et	Extra Trees Classifier	sklearn.ensemble._forest.ExtraTreesClassifier	True
xgboost	Extreme Gradient Boosting	xgboost.sklearn.XGBClassifier	True
lightgbm	Light Gradient Boosting Machine	lightgbm.sklearn.LGBMClassifier	True
catboost	CatBoost Classifier	catboost.core.CatBoostClassifier	True

In [53]:
```
models()['Name'][2]
```

Out[53]:
```
'Naive Bayes'
```

In [54]:
```
models()[:3]
```

Out[54]:

ID	Name	Reference	Turbo
lr	Logistic Regression	sklearn.linear_model._logistic.LogisticRegression	True
knn	K Neighbors Classifier	sklearn.neighbors._classification.KNeighborsCl...	True
nb	Naive Bayes	sklearn.naive_bayes.GaussianNB	True

In [45]:
```
best_model = compare_models()
```

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	TT (Sec)
catboost	CatBoost Classifier	0.7786	0.8392	0.6000	0.7067	0.6418	0.4843	0.4929	1.2500
gbc	Gradient Boosting Classifier	0.7674	0.8344	0.5778	0.6948	0.6207	0.4565	0.4673	0.0440
ada	Ada Boost Classifier	0.7619	0.8053	0.6056	0.6556	0.6245	0.4519	0.4566	0.0420
lr	Logistic Regression	0.7564	0.8046	0.5111	0.6876	0.5815	0.4160	0.4279	0.4580
ridge	Ridge Classifier	0.7545	0.0000	0.5000	0.6954	0.5747	0.4095	0.4247	0.0070
lda	Linear Discriminant Analysis	0.7527	0.7942	0.4944	0.6929	0.5700	0.4041	0.4197	0.0080
rf	Random Forest Classifier	0.7526	0.8089	0.5222	0.6861	0.5810	0.4119	0.4268	0.1020
lightgbm	Light Gradient Boosting Machine	0.7524	0.8022	0.5833	0.6475	0.6101	0.4301	0.4343	0.0520
xgboost	Extreme Gradient Boosting	0.7414	0.7898	0.5944	0.6251	0.6053	0.4139	0.4173	0.1190
et	Extra Trees Classifier	0.7338	0.7690	0.4444	0.6643	0.5209	0.3494	0.3690	0.0920
knn	K Neighbors Classifier	0.7209	0.7432	0.5389	0.6081	0.5607	0.3587	0.3678	0.0150
dt	Decision Tree Classifier	0.6966	0.6644	0.5667	0.5426	0.5525	0.3237	0.3252	0.0070
nb	Naive Bayes	0.6798	0.7022	0.2222	0.5411	0.3096	0.1532	0.1775	0.0070
svm	SVM - Linear Kernel	0.6390	0.0000	0.1667	0.3905	0.2020	0.0614	0.0699	0.0080
qda	Quadratic Discriminant Analysis	0.5120	0.5633	0.6556	0.3907	0.4549	0.0866	0.0988	0.0090

In [46]:
```
#Compare ensemble models only.
print('Ensemble models: ', models(type='ensemble').index.tolist())

models_to_include = models(type='ensemble').index.tolist()#.remove('gbr')
print(models_to_include)

ensembled_models = compare_models(include = models_to_include, fold = 3)
```

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	TT (Sec)
catboost	CatBoost Classifier	0.7523	0.8261	0.5389	0.6703	0.5940	0.4189	0.4267	1.1600
ada	Ada Boost Classifier	0.7505	0.7998	0.6000	0.6374	0.6151	0.4314	0.4342	0.0467
gbc	Gradient Boosting Classifier	0.7467	0.7945	0.5722	0.6350	0.5973	0.4149	0.4189	0.0467
et	Extra Trees Classifier	0.7467	0.7662	0.4778	0.6841	0.5583	0.3887	0.4038	0.0967
rf	Random Forest Classifier	0.7374	0.8016	0.4722	0.6581	0.5462	0.3687	0.3812	0.1100
lightgbm	Light Gradient Boosting Machine	0.7356	0.7944	0.5389	0.6293	0.5756	0.3864	0.3921	0.0467
xgboost	Extreme Gradient Boosting	0.7207	0.7763	0.5111	0.6020	0.5505	0.3503	0.3543	0.1200

In [47]:
```
# Select the best model based on the chosen metric
best = automl(optimize = 'Accuracy')
best
```

Out[47]:
```
<catboost.core.CatBoostClassifier at 0x205026b3bb0>
```

In [ ]:
```
# Select the best model based on the chosen metric

#best = automl(optimize = 'Recall')
#best
```

Analyze Models With Plots

In [48]:
```
%%time
evaluate_model(best)
```

A Jupyter widget could not be displayed because the widget state could not be found. This could happen if the kernel storing the widget is no longer available, or if the widget state was not saved in the notebook. You may be able to create the widget by running the appropriate cells.

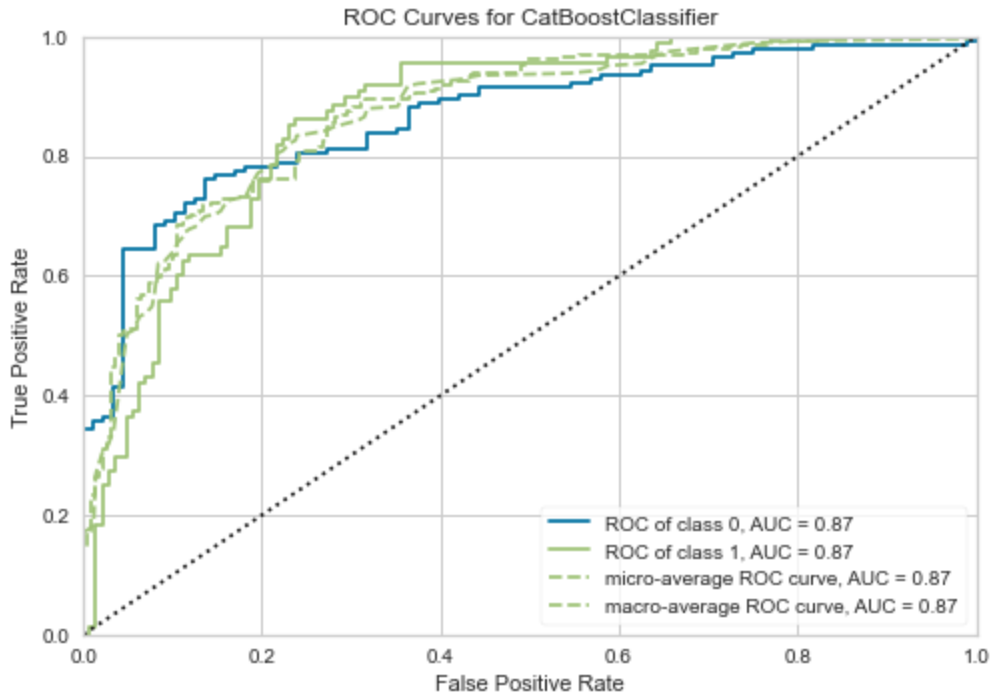
```
Wall time: 258 ms
```

In [59]:
```
best.get_feature_importance(prettified=True)
```

Out[59]:

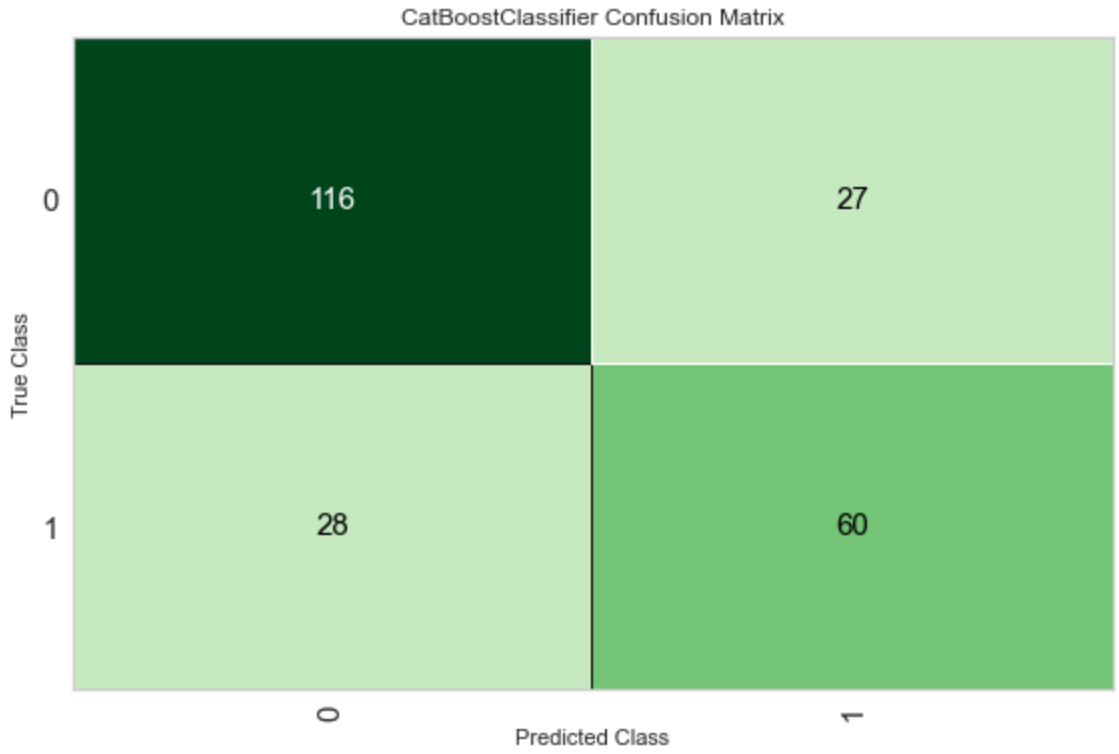
	Feature Id	Importances
0	plas	21.933147
1	mass	17.836461
2	age	17.453641
3	pedi	11.927685
4	pres	8.243372
5	skin	7.732584
6	insu	7.538819
7	preg_1	1.479110
8	preg_4	1.474747
9	preg_6	0.955257
10	preg_3	0.951695
11	preg_7	0.516305
12	preg_0	0.503253
13	preg_2	0.401271
14	preg_9	0.395351
15	preg_5	0.185288
16	preg_8	0.158292
17	preg_10	0.126760
18	preg_11	0.062982
19	preg_12	0.049809
20	preg_13	0.047675
21	preg_15	0.015942
22	preg_14	0.010553

In [60]:
```
plot_model(best)
print('best model plot', best)
```

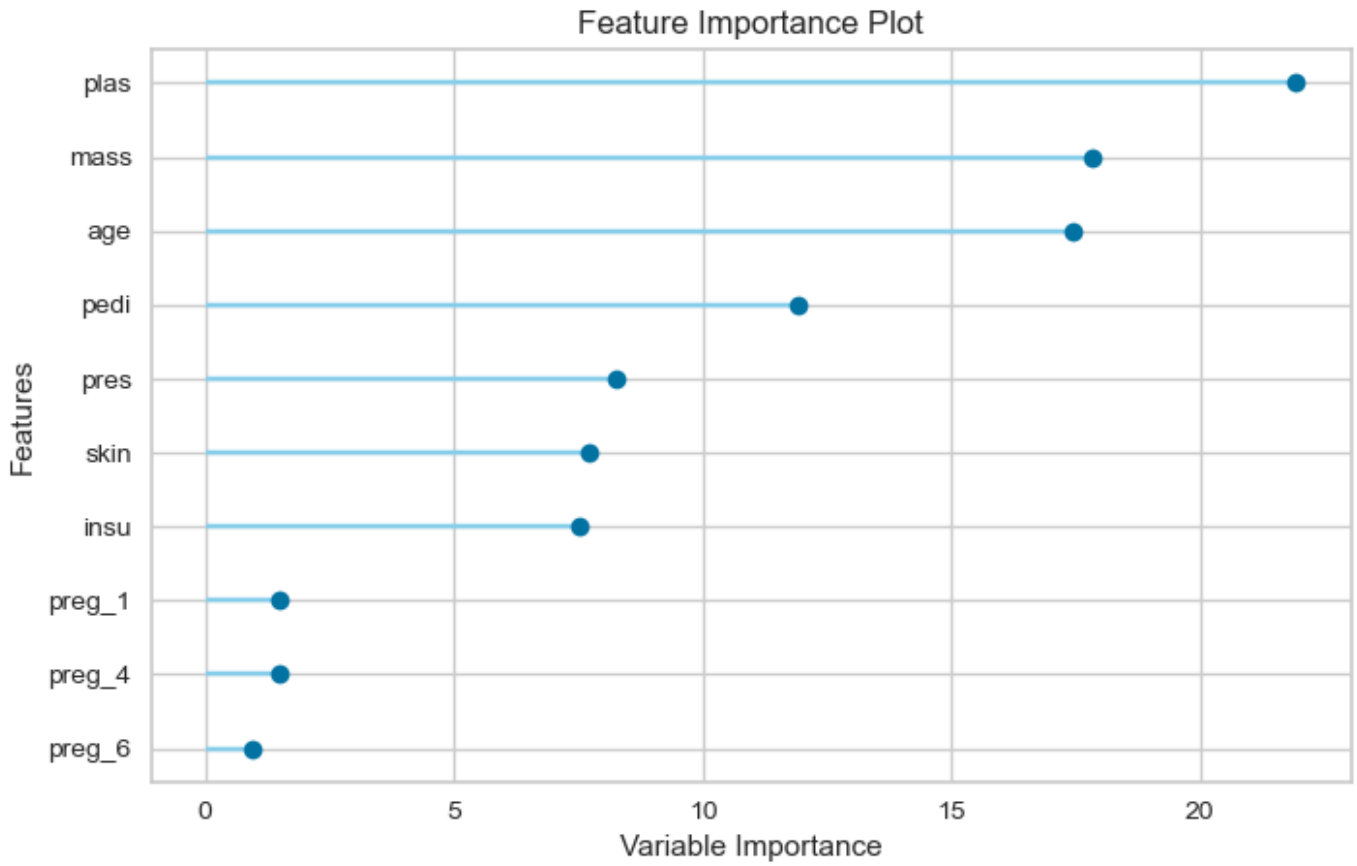



```
best model plot <catboost.core.CatBoostClassifier object at 0x0000026DDADAD280>
```

In [61]:
```
plot_model(best, plot = 'confusion_matrix')
```



In [62]:
```
plot_model(best, plot = 'feature')
```

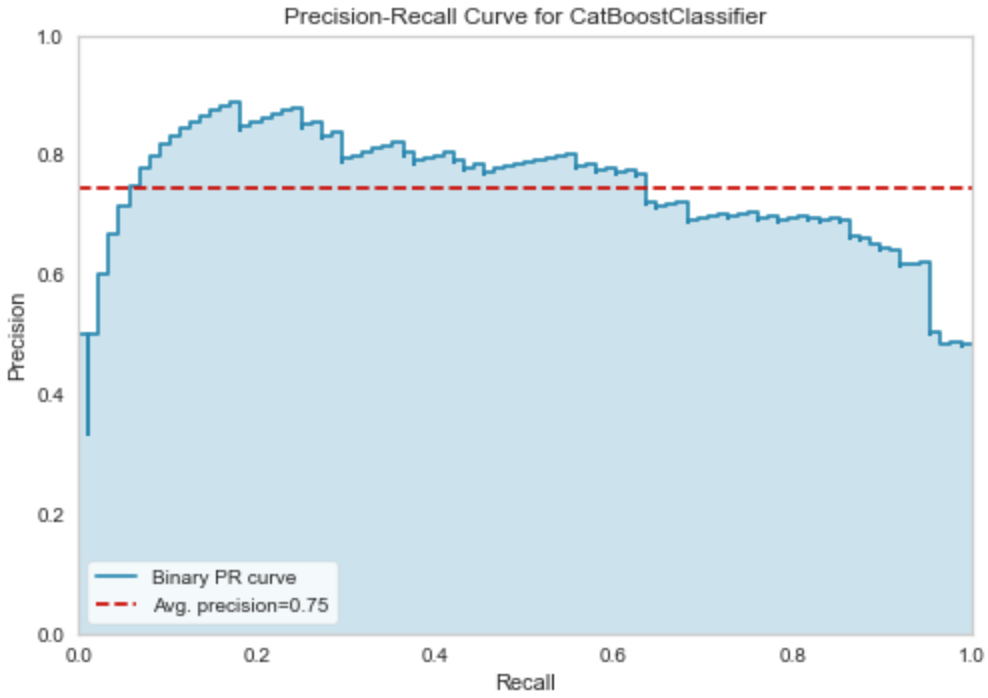


In [63]:
```
best.get_feature_importance(prettified=True)
```

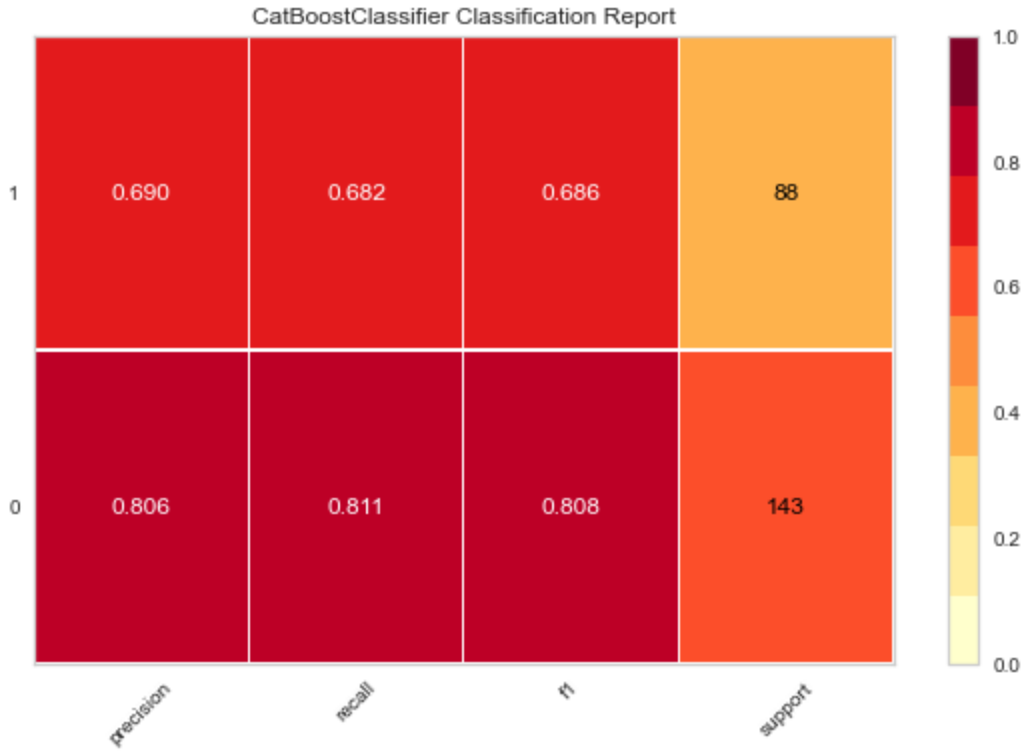
Out[63]:

	Feature Id	Importances
0	plas	21.933147
1	mass	17.836461
2	age	17.453641
3	pedi	11.927685
4	pres	8.243372
5	skin	7.732584
6	insu	7.538819
7	preg_1	1.479110
8	preg_4	1.474747
9	preg_6	0.955257
10	preg_3	0.951695
11	preg_7	0.516305
12	preg_0	0.503253
13	preg_2	0.401271
14	preg_9	0.395351
15	preg_5	0.185288
16	preg_8	0.158292
17	preg_10	0.126760
18	preg_11	0.062982
19	preg_12	0.049809
20	preg_13	0.047675
21	preg_15	0.015942
22	preg_14	0.010553

In [64]:
```
plot_model(best, plot = 'pr')
```



In [65]:
```
plot_model(best, plot = 'class_report')
```



Interpret Models With SHAP

In [ ]:
```
interpret_model(best)
```

In [ ]:
```
#interpret_model(best, plot = 'correlation')
```

In [ ]:
```
interpret_model(best, plot = 'reason', observation = 12)
```

Predict

In [49]:
```
predict_model(best).head(10)
```

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	CatBoost Classifier	0.7619	0.8655	0.6818	0.6897	0.6857	0.4941	0.4941

Out[49]:

	plas	pres	skin	insu	mass	pedi	age	preg_0	preg_1	preg_10	...	preg_3	preg_4	preg_5	pre
0	181.0	84.0	21.0	192.0	35.900002	0.586	51.0	0.0	0.0	0.0	...	0.0	0.0	0.0	
1	127.0	58.0	24.0	275.0	27.700001	1.600	25.0	0.0	0.0	0.0	...	0.0	0.0	0.0	
2	138.0	76.0	0.0	125.0	33.200001	0.420	35.0	0.0	0.0	0.0	...	0.0	0.0	0.0	
3	81.0	60.0	22.0	125.0	27.700001	0.290	25.0	0.0	0.0	0.0	...	0.0	0.0	0.0	
4	84.0	82.0	31.0	125.0	38.200001	0.233	23.0	1.0	0.0	0.0	...	0.0	0.0	0.0	
5	140.0	94.0	0.0	125.0	32.700001	0.734	45.0	0.0	0.0	0.0	...	0.0	0.0	0.0	
6	197.0	74.0	0.0	125.0	25.900000	1.191	39.0	0.0	0.0	0.0	...	0.0	0.0	0.0	
7	120.0	72.0	0.0	125.0	30.000000	0.183	38.0	0.0	0.0	0.0	...	0.0	0.0	0.0	
8	161.0	86.0	0.0	125.0	30.400000	0.165	47.0	0.0	0.0	0.0	...	0.0	0.0	0.0	
9	120.0	68.0	0.0	125.0	29.600000	0.709	34.0	0.0	0.0	0.0	...	0.0	1.0	0.0	

10 rows × 26 columns

Save and Load Model

In [ ]:
```
save_model(best, model_name='best-model')
```

In [ ]:
```
loaded_bestmodel = load_model('best-model')
print(loaded_bestmodel)
```

((()))))))))))))))))))))))))))))))))))

In [52]:
```
from sklearn.model_selection import train_test_split
y = diabetesDF['class']
X = diabetesDF.drop('class', axis=1)


#we will be splitting the following data into labels and features. Labels are the data whic
X_train, X_test, y_train, y_test = train_test_split(X, y,test_size=0.4)
```

In [55]:
```
models
```

Out[55]:
```
<function pycaret.classification.models(type: Union[str, NoneType] = None, internal: bool
= False, raise_errors: bool = True) -> pandas.core.frame.DataFrame>
```

In [56]:
```
#GradientBoostingClassifier
from sklearn.ensemble import GradientBoostingClassifier
clf = GradientBoostingClassifier(n_estimators=100, learning_rate=1.0,
       max_depth=1, random_state=0).fit(X_train, y_train)
clf.score(X_test, y_test)
print("GBC Accuracy = ", clf.score(X_test, y_test) * 100 , "%")

coef_df = pd.DataFrame(models.coef_)
coef_df['intercept'] = model.intercept_
print(coef_df)
print(diabetesDF.columns)
```

```
GBC Accuracy =  73.05194805194806 %

----------------------
AttributeErrorTraceback (most recent call last)
<ipython-input-56-e58a79706ac9> in <module>
      6 print("GBC Accuracy = ", clf.score(X_test, y_test) * 100 , "%")
      7
----> 8 coef_df = pd.DataFrame(models.coef_)
      9 coef_df['intercept'] = model.intercept_
     10 print(coef_df)

AttributeError: 'function' object has no attribute 'coef_'
```

In [57]:
```
from sklearn.linear_model import LogisticRegression

# Fit the model on train
model = LogisticRegression(solver="liblinear")
model.fit(X_train, y_train)
#predict on test
y_predict = model.predict(X_test)

model_accuracy = model.score(X_test, y_test)
print("Logistic Regression Accuracy = ", model_accuracy * 100 , "%")


from sklearn.metrics import classification_report

print(classification_report(y_test, y_predict))
```

```
Logistic Regression Accuracy =  75.0 %
                precision    recall  f1-score   support

tested_negative       0.75      0.89      0.82       192
tested_positive       0.74      0.52      0.61       116

       accuracy                           0.75       308
      macro avg       0.75      0.70      0.71       308
   weighted avg       0.75      0.75      0.74       308
```

catboost.core.CatBoostClassifier

In [58]:
```
from catboost import Pool, CatBoostClassifier, cv
import numpy as np
np.set_printoptions(precision=4)
import catboost
from catboost import *
```

In [59]:
```
rnd_state = 42

categorical_features_indices = np.where(X.dtypes != np.float)[0]

clf = CatBoostClassifier(random_seed=rnd_state)

clf.fit(X_train, y_train, cat_features=categorical_features_indices)
clf.score(X_test, y_test)
```

```
Learning rate set to 0.007395
0:      learn: 0.6889167        total: 16.8ms   remaining: 16.8s
1:      learn: 0.6846055        total: 33.2ms   remaining: 16.6s
2:      learn: 0.6815775        total: 49.9ms   remaining: 16.6s
3:      learn: 0.6777552        total: 66.1ms   remaining: 16.5s
4:      learn: 0.6744665        total: 82.6ms   remaining: 16.4s
5:      learn: 0.6706388        total: 98.7ms   remaining: 16.3s
6:      learn: 0.6678875        total: 116ms    remaining: 16.4s
7:      learn: 0.6646381        total: 129ms    remaining: 16s
8:      learn: 0.6609194        total: 145ms    remaining: 16s
9:      learn: 0.6577722        total: 161ms    remaining: 16s
10:     learn: 0.6540570        total: 178ms    remaining: 16s
11:     learn: 0.6515433        total: 197ms    remaining: 16.2s
12:     learn: 0.6490030        total: 209ms    remaining: 15.9s
13:     learn: 0.6471718        total: 216ms    remaining: 15.2s
14:     learn: 0.6434356        total: 239ms    remaining: 15.7s
15:     learn: 0.6407486        total: 256ms    remaining: 15.7s
16:     learn: 0.6387178        total: 265ms    remaining: 15.3s
17:     learn: 0.6363511        total: 279ms    remaining: 15.2s
```

42= 0.75 30 = 76.94 F 0.7786 S 0.7523

In [60]:
```
# Submission 1: catboost submission with all training data and early stopping on Accuracy
#test_df = pd.read_csv('../input/test.csv', index_col='PassengerId')
#test_df.fillna(-999, inplace=True)
clf_od = CatBoostClassifier(random_seed=rnd_state, od_type='Iter', od_wait=20, eval_metric=
clf_od.fit(X, y, cat_features=categorical_features_indices)

clf_od.score(X_test, y_test)
```

```
Learning rate set to 0.009204
0:      learn: 0.7513021        total: 15.2ms   remaining: 15.2s
1:      learn: 0.7682292        total: 32ms     remaining: 16s
2:      learn: 0.7630208        total: 47ms     remaining: 15.6s
3:      learn: 0.7643229        total: 63.4ms   remaining: 15.8s
4:      learn: 0.7656250        total: 80ms     remaining: 15.9s
5:      learn: 0.7643229        total: 96.1ms   remaining: 15.9s
6:      learn: 0.7708333        total: 113ms    remaining: 16s
7:      learn: 0.7734375        total: 129ms    remaining: 16s
8:      learn: 0.7773438        total: 145ms    remaining: 16s
9:      learn: 0.7773438        total: 162ms    remaining: 16s
10:     learn: 0.7877604        total: 181ms    remaining: 16.3s
11:     learn: 0.7916667        total: 202ms    remaining: 16.6s
12:     learn: 0.7812500        total: 215ms    remaining: 16.3s
13:     learn: 0.7877604        total: 234ms    remaining: 16.4s
14:     learn: 0.7890625        total: 250ms    remaining: 16.4s
15:     learn: 0.7851563        total: 268ms    remaining: 16.5s
16:     learn: 0.7929688        total: 284ms    remaining: 16.4s
17:     learn: 0.7903646        total: 301ms    remaining: 16.4s
```

Model Comments:

•1. The model returned a list of features that appear to explain majority of the variance in the dataset.

•2. After comparing these 3 models found that catbooster is best model with average accuracy score of 75 %.

•3. We will use this as a reference point to judge later models and reference this classification report for subsequent models.

Recommendations:

1. Insulin in blood decided type of diabetes, high insulin found in blood (as a medical reason insulin resistance) among type – II positive diabetes compare to other clustered people type – I.

2. Most important features are plas, mass, age, pedi , pres, skin, insu and then preg.

1. It is possible to predict with an accuracy of 75%
2. People with more number of pregnancies are more prone to diabetes.
3. People with high blood pressure and plasma glucose are more diabetic.
4. Age is also one factor in diabetes.

Future exploration could also include other models that handle categorical variables like LightGBM and H2oGBM.

*****