eblsjaojf

February 6, 2023

[1]:
```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.metrics import accuracy_score, confusion_matrix , roc_curve,␣
 ↪roc_auc_score
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

[2]:
```
data = pd.read_csv('diabetes.csv')

data.head()
```

[2]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	\
0	6	148	72	35	0	33.6	
1	1	85	66	29	0	26.6	
2	8	183	64	0	0	23.3	
3	1	89	66	23	94	28.1	
4	0	137	40	35	168	43.1	

	DiabetesPedigreeFunction	Age	Outcome
0	0.627	50	1
1	0.351	31	0
2	0.672	32	1
3	0.167	21	0
4	2.288	33	1

[3]:
```
data.shape
```

[3]:
```
(768, 9)
```

[4]:
```
data.describe()
```

[4]:

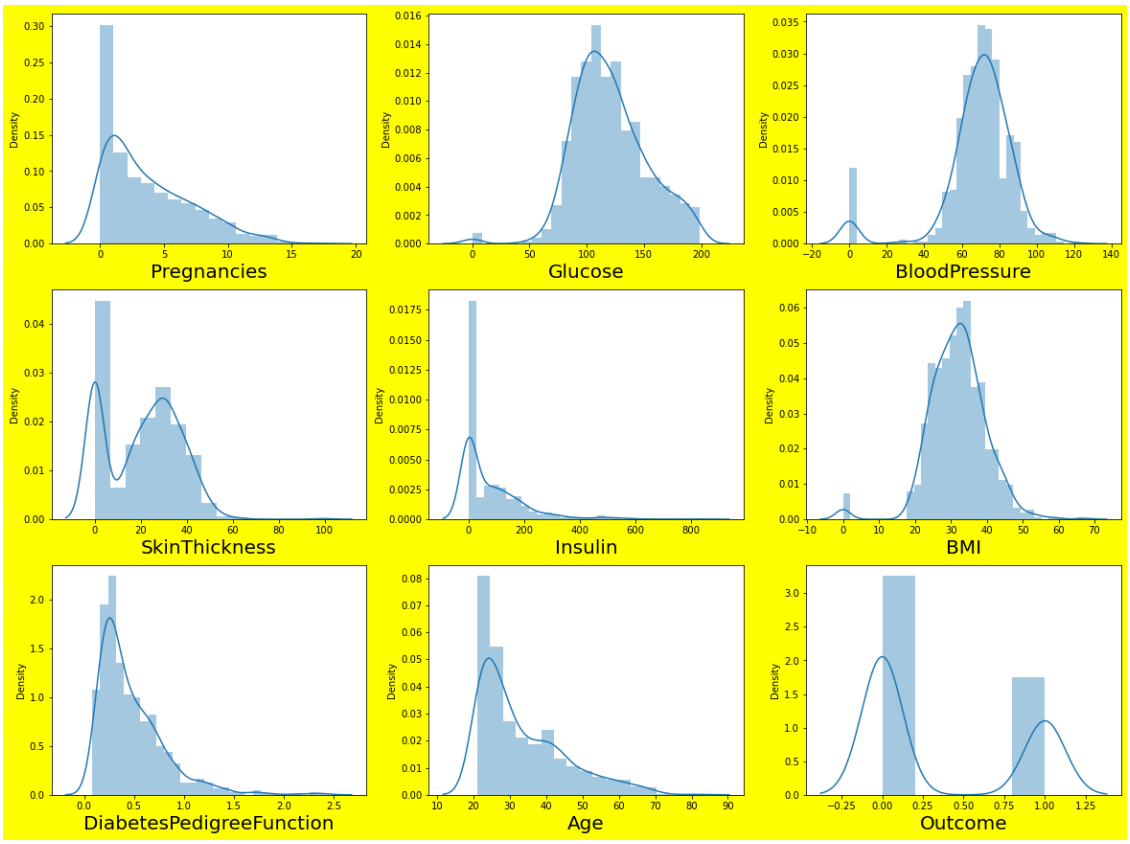
	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin
count	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	120.894531	69.105469	20.536458	79.799479
std	3.369578	31.972618	19.355807	15.952218	115.244002
min	0.000000	0.000000	0.000000	0.000000	0.000000
25%	1.000000	99.000000	62.000000	0.000000	0.000000
50%	3.000000	117.000000	72.000000	23.000000	30.500000
75%	6.000000	140.250000	80.000000	32.000000	127.250000
max	17.000000	199.000000	122.000000	99.000000	846.000000

	BMI	DiabetesPedigreeFunction	Age	Outcome
count	768.000000	768.000000	768.000000	768.000000
mean	31.992578	0.471876	33.240885	0.348958
std	7.884160	0.331329	11.760232	0.476951
min	0.000000	0.078000	21.000000	0.000000
25%	27.300000	0.243750	24.000000	0.000000
50%	32.000000	0.372500	29.000000	0.000000
75%	36.600000	0.626250	41.000000	1.000000
max	67.100000	2.420000	81.000000	1.000000

It seems that there are no missing values in our data. Great,Let's see the distribution of data:

1 Data Preprocessing Analysis (EDA)

[5]:
```
plt.figure(figsize=(20,15), facecolor = "yellow")
plotnumber = 1  # take 1 column at a time plot it and then go to next

for column in data:
    if plotnumber<= 9:
        ax = plt.subplot(3,3,plotnumber)
        sns.distplot(data[column])
        plt.xlabel(column,fontsize = 20)

    plotnumber+=1
plt.show()
```

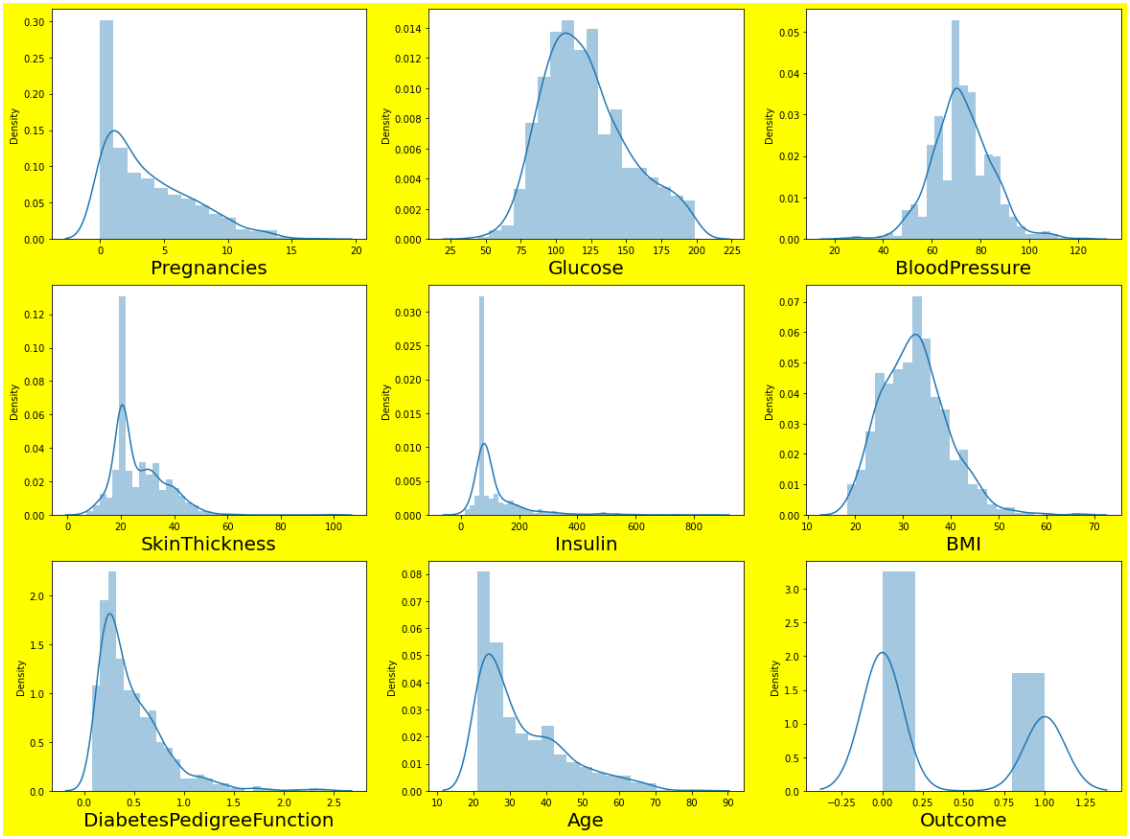


we can see there is some skewness in the data, let'sdeal with data.

Also,we can see there few data for columns 'Glucose','Insulin','skin thickness','BMI' and 'Blood Pressure' which have values as 0. That's not possible. you can do a quick search to see that one cannot have 0 values for these. Let's deal with that. we can either remove such data or simply replace it with their respective mean values.

[6]:
```
# Replacing zero values with mean of the column

data['BMI'] = data['BMI'].replace(0 , data['BMI'].mean())

data['BloodPressure'] = data['BloodPressure'].replace(0 , data['BloodPressure'].
↪mean())

data['Glucose'] = data['Glucose'].replace(0 , data['Glucose'].mean())

data['Insulin'] = data['Insulin'].replace(0 , data['Insulin'].mean())

data['SkinThickness'] = data['SkinThickness'].replace(0 , data['SkinThickness'].
↪mean())
```

[7]:
```
plt.figure(figsize=(20,15), facecolor = "yellow")
plotnumber = 1  # take 1 column at a time plot it and then go to next

for column in data:
    if plotnumber<= 9:
        ax = plt.subplot(3,3,plotnumber)
        sns.distplot(data[column])
        plt.xlabel(column,fontsize = 20)

    plotnumber+=1
plt.show()
```

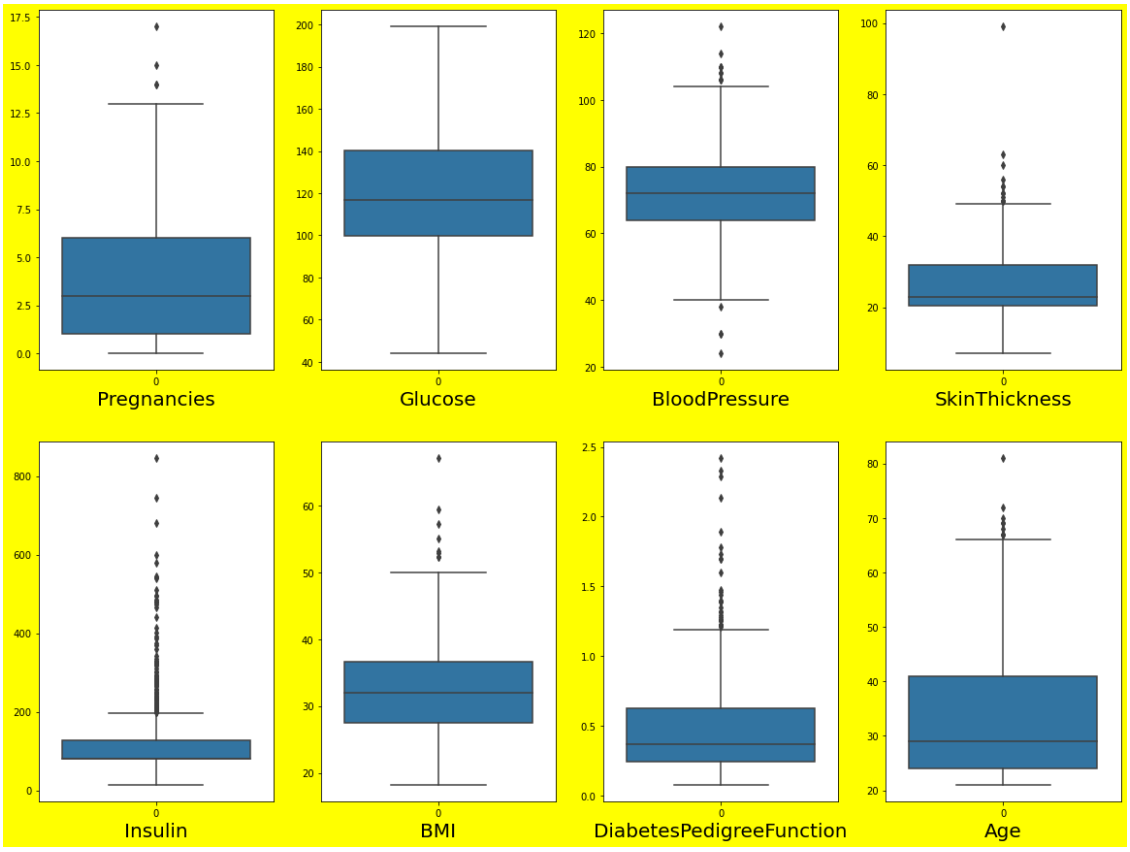


Now we have deal with the 0 values and data looks better.But there still are outliers present in some columns. Let's deal with them

[8]:
```
df = data.drop('Outcome', axis =1)
```

[9]:
```
plt.figure(figsize=(20,15), facecolor = "yellow")
plotnumber = 1  # take 1 column at a time plot it and then go to next

for column in df:
```

```
    if plotnumber<= 8:
        plt.subplot(2,4,plotnumber)
        ax = sns.boxplot(data = df[column])
        plt.xlabel(column,fontsize = 20)

    plotnumber+=1
plt.show()
```



[10]:
```
data.shape
```

[10]: (768, 9)

1.1 Find the IQR (inter quantile range) to identify outliers

[11]:
```
# 1st quantile

q1 = data.quantile(0.25)

# 3rd quantile
```

```
q3 = data.quantile(0.75)

# IQR

iqr = q3-q1
```

2 Outlier Detection Formula

2.1 higher side ==> Q3 + (1.5 * IQR)

2.2 lower side ==> Q1 - (1.5 * IQR)

[12]:
```
# Validating one outlier

preg = (q3.Pregnancies + (1.5 * iqr.Pregnancies))

preg
```

[12]:
```
13.5
```

[13]:
```
# Check the indexes which have higher values

index = np.where(data['Pregnancies'] > preg)
index
```

[13]:
```
(array([ 88, 159, 298, 455], dtype=int64),)
```

[14]:
```
# Drop the index which we found in the above cell

data = data.drop(data.index[index]) # drop the data/outlier on the basis of␣
 ↪their index numbers
data.shape
```

[14]:
```
(764, 9)
```

[15]:
```
# Resetting the indexes

data.reset_index()
```

[15]:

	index	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	\
0	0	6	148.0	72.0	35.000000	79.799479	
1	1	1	85.0	66.0	29.000000	79.799479	
2	2	8	183.0	64.0	20.536458	79.799479	
3	3	1	89.0	66.0	23.000000	94.000000	
4	4	0	137.0	40.0	35.000000	168.000000	
..	…	…	…	…	…	…	
759	763	10	101.0	76.0	48.000000	180.000000	

```
760    764            2    122.0           70.0      27.000000   79.799479
761    765            5    121.0           72.0      23.000000  112.000000
762    766            1    126.0           60.0      20.536458   79.799479
763    767            1     93.0           70.0      31.000000   79.799479

      BMI  DiabetesPedigreeFunction  Age  Outcome
0    33.6                     0.627   50        1
1    26.6                     0.351   31        0
2    23.3                     0.672   32        1
3    28.1                     0.167   21        0
4    43.1                     2.288   33        1
..    …                       …  …      …
759  32.9                     0.171   63        0
760  36.8                     0.340   27        0
761  26.2                     0.245   30        0
762  30.1                     0.349   47        1
763  30.4                     0.315   23        0

[764 rows x 10 columns]
```

[16]:
```
bp = (q3.BloodPressure + (1.5 * iqr.BloodPressure))
print(bp)

index = np.where(data['BloodPressure'] > bp)

data = data.drop(data.index[index])
print(data.shape)

data.reset_index()
```

```
104.0
(754, 9)
```

[16]:
```
     index  Pregnancies  Glucose  BloodPressure  SkinThickness     Insulin  \
0        0            6    148.0           72.0      35.000000   79.799479
1        1            1     85.0           66.0      29.000000   79.799479
2        2            8    183.0           64.0      20.536458   79.799479
3        3            1     89.0           66.0      23.000000   94.000000
4        4            0    137.0           40.0      35.000000  168.000000
..     …          …      …             …             …          …
749    763           10    101.0           76.0      48.000000  180.000000
750    764            2    122.0           70.0      27.000000   79.799479
751    765            5    121.0           72.0      23.000000  112.000000
752    766            1    126.0           60.0      20.536458   79.799479
753    767            1     93.0           70.0      31.000000   79.799479

      BMI  DiabetesPedigreeFunction  Age  Outcome
```

```
0    33.6                     0.627   50        1
1    26.6                     0.351   31        0
2    23.3                     0.672   32        1
3    28.1                     0.167   21        0
4    43.1                     2.288   33        1
..    …                       …  …      …
749  32.9                     0.171   63        0
750  36.8                     0.340   27        0
751  26.2                     0.245   30        0
752  30.1                     0.349   47        1
753  30.4                     0.315   23        0

[754 rows x 10 columns]
```

[17]:
```
sk = (q3.SkinThickness + (1.5 * iqr.SkinThickness))
print(sk)

index = np.where(data['SkinThickness'] > sk)

data = data.drop(data.index[index])
print(data.shape)

data.reset_index()
```

```
49.1953125
(742, 9)
```

[17]:
```
     index  Pregnancies  Glucose  BloodPressure  SkinThickness     Insulin  \
0        0            6    148.0           72.0      35.000000   79.799479
1        1            1     85.0           66.0      29.000000   79.799479
2        2            8    183.0           64.0      20.536458   79.799479
3        3            1     89.0           66.0      23.000000   94.000000
4        4            0    137.0           40.0      35.000000  168.000000
..     …          …       …             …          …         …
737    763           10    101.0           76.0      48.000000  180.000000
738    764            2    122.0           70.0      27.000000   79.799479
739    765            5    121.0           72.0      23.000000  112.000000
740    766            1    126.0           60.0      20.536458   79.799479
741    767            1     93.0           70.0      31.000000   79.799479

      BMI  DiabetesPedigreeFunction  Age  Outcome
0    33.6                     0.627   50        1
1    26.6                     0.351   31        0
2    23.3                     0.672   32        1
3    28.1                     0.167   21        0
4    43.1                     2.288   33        1
..    …                       …  …      …
```

```
737  32.9                     0.171   63        0
738  36.8                     0.340   27        0
739  26.2                     0.245   30        0
740  30.1                     0.349   47        1
741  30.4                     0.315   23        0

[742 rows x 10 columns]
```

[18]:
```
itn = (q3.Insulin + (1.5 * iqr.Insulin))
print(itn)

index = np.where(data['Insulin'] > itn)

data = data.drop(data.index[index])
print(data.shape)

data.reset_index()
```

```
198.42578125
(657, 9)
```

[18]:

	index	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	\
0	0	6	148.0	72.0	35.000000	79.799479	
1	1	1	85.0	66.0	29.000000	79.799479	
2	2	8	183.0	64.0	20.536458	79.799479	
3	3	1	89.0	66.0	23.000000	94.000000	
4	4	0	137.0	40.0	35.000000	168.000000	
..	…	…	…	…	…	…	
652	763	10	101.0	76.0	48.000000	180.000000	
653	764	2	122.0	70.0	27.000000	79.799479	
654	765	5	121.0	72.0	23.000000	112.000000	
655	766	1	126.0	60.0	20.536458	79.799479	
656	767	1	93.0	70.0	31.000000	79.799479	

	BMI	DiabetesPedigreeFunction	Age	Outcome
0	33.6	0.627	50	1
1	26.6	0.351	31	0
2	23.3	0.672	32	1
3	28.1	0.167	21	0
4	43.1	2.288	33	1
..	…	… …	…	
652	32.9	0.171	63	0
653	36.8	0.340	27	0
654	26.2	0.245	30	0
655	30.1	0.349	47	1
656	30.4	0.315	23	0

[657 rows x 10 columns]

[19]:
```
bm = (q3.BMI + (1.5 * iqr.BMI))
print(bm)

index = np.where(data['BMI'] > bm)

data = data.drop(data.index[index])
print(data.shape)

data.reset_index()
```

```
50.25
(654, 9)
```

[19]:

	index	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	0	6	148.0	72.0	35.000000	79.799479	33.6	0.627	50	1
1	1	1	85.0	66.0	29.000000	79.799479	26.6	0.351	31	0
2	2	8	183.0	64.0	20.536458	79.799479	23.3	0.672	32	1
3	3	1	89.0	66.0	23.000000	94.000000	28.1	0.167	21	0
4	4	0	137.0	40.0	35.000000	168.000000	43.1	2.288	33	1
..	…	…	…	…	…	…	…	…	…	…
649	763	10	101.0	76.0	48.000000	180.000000	32.9	0.171	63	0
650	764	2	122.0	70.0	27.000000	79.799479	36.8	0.340	27	0
651	765	5	121.0	72.0	23.000000	112.000000	26.2	0.245	30	0
652	766	1	126.0	60.0	20.536458	79.799479	30.1	0.349	47	1
653	767	1	93.0	70.0	31.000000	79.799479	30.4	0.315	23	0

[654 rows x 10 columns]

[20]:
```
dp = (q3.DiabetesPedigreeFunction + (1.5 * iqr.DiabetesPedigreeFunction))
print(dp)

index = np.where(data['DiabetesPedigreeFunction'] > dp)
```

```
data = data.drop(data.index[index])
print(data.shape)

data.reset_index()
```

```
1.2
(631, 9)
```

[20]:

	index	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	0	6	148.0	72.0	35.000000	79.799479	33.6	0.627	50	1
1	1	1	85.0	66.0	29.000000	79.799479	26.6	0.351	31	0
2	2	8	183.0	64.0	20.536458	79.799479	23.3	0.672	32	1
3	3	1	89.0	66.0	23.000000	94.000000	28.1	0.167	21	0
4	5	5	116.0	74.0	20.536458	79.799479	25.6	0.201	30	0
..	…	…	…	…	…	…	…	…	…	…
626	763	10	101.0	76.0	48.000000	180.000000	32.9	0.171	63	0
627	764	2	122.0	70.0	27.000000	79.799479	36.8	0.340	27	0
628	765	5	121.0	72.0	23.000000	112.000000	26.2	0.245	30	0
629	766	1	126.0	60.0	20.536458	79.799479	30.1	0.349	47	1
630	767	1	93.0	70.0	31.000000	79.799479	30.4	0.315	23	0

[631 rows x 10 columns]

[21]:
```
ag = (q3.Age + (1.5 * iqr.Age))
print(ag)

index = np.where(data['Age'] > ag)

data = data.drop(data.index[index])
print(data.shape)

data.reset_index()
```

```
66.5
```

```
(622, 9)
```

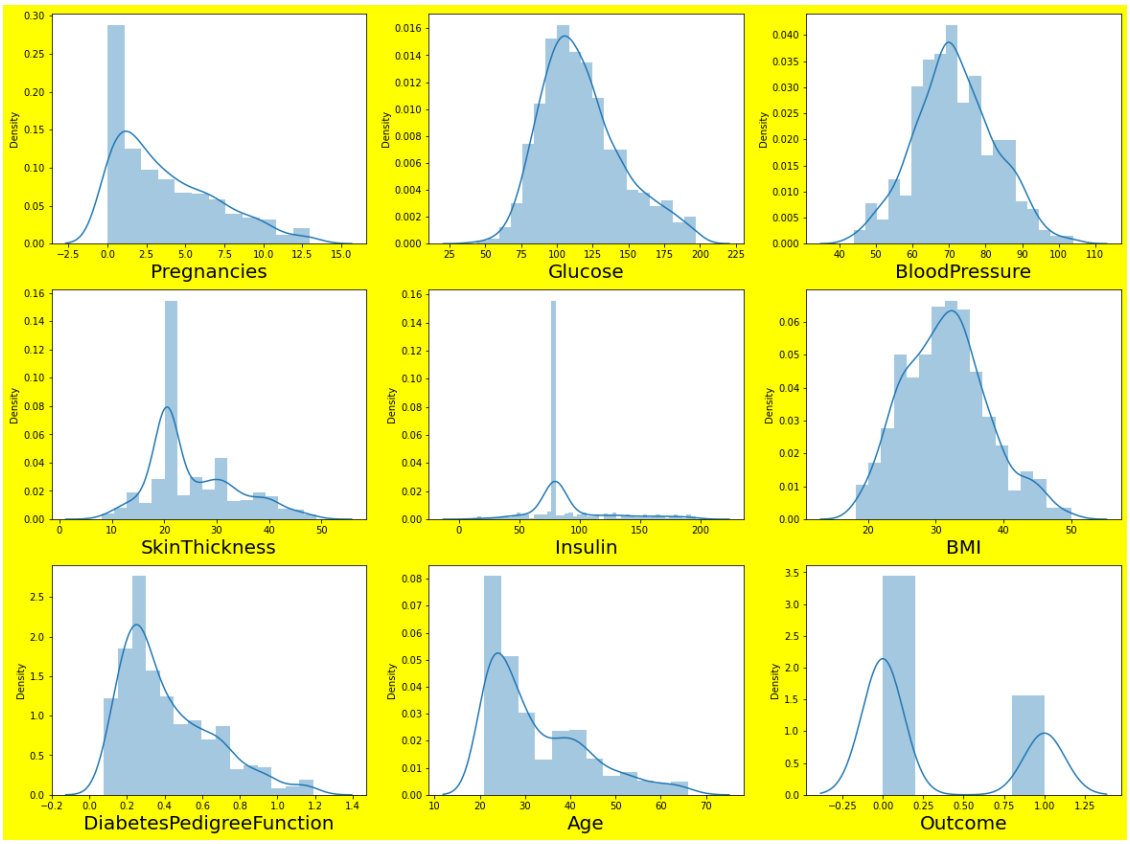
[21]:
```
     index  Pregnancies  Glucose  BloodPressure  SkinThickness     Insulin  \
0        0            6    148.0           72.0      35.000000   79.799479
1        1            1     85.0           66.0      29.000000   79.799479
2        2            8    183.0           64.0      20.536458   79.799479
3        3            1     89.0           66.0      23.000000   94.000000
4        5            5    116.0           74.0      20.536458   79.799479
..     ...          ...      ...            ...            ...         ...
617    763           10    101.0           76.0      48.000000  180.000000
618    764            2    122.0           70.0      27.000000   79.799479
619    765            5    121.0           72.0      23.000000  112.000000
620    766            1    126.0           60.0      20.536458   79.799479
621    767            1     93.0           70.0      31.000000   79.799479

      BMI  DiabetesPedigreeFunction  Age  Outcome
0    33.6                     0.627   50        1
1    26.6                     0.351   31        0
2    23.3                     0.672   32        1
3    28.1                     0.167   21        0
4    25.6                     0.201   30        0
..    ...                       ...  ...      ...
617  32.9                     0.171   63        0
618  36.8                     0.340   27        0
619  26.2                     0.245   30        0
620  30.1                     0.349   47        1
621  30.4                     0.315   23        0

[622 rows x 10 columns]
```

[22]:
```
# Because Bloodpressure than outliers on both high side and low side
# Removing Outliers of low side

bp1 = (q1.BloodPressure - (1.5 * iqr.BloodPressure))
print(bp1)

index = np.where(data['BloodPressure'] < bp1)

data = data.drop(data.index[index])
print(data.shape)

data.reset_index()
```

```
40.0
(619, 9)
```

[22]:
```
     index  Pregnancies  Glucose  BloodPressure  SkinThickness     Insulin  \
0        0            6    148.0           72.0      35.000000   79.799479
1        1            1     85.0           66.0      29.000000   79.799479
2        2            8    183.0           64.0      20.536458   79.799479
3        3            1     89.0           66.0      23.000000   94.000000
4        5            5    116.0           74.0      20.536458   79.799479
..     ...          ...      ...            ...            ...         ...
614    763           10    101.0           76.0      48.000000  180.000000
615    764            2    122.0           70.0      27.000000   79.799479
616    765            5    121.0           72.0      23.000000  112.000000
617    766            1    126.0           60.0      20.536458   79.799479
618    767            1     93.0           70.0      31.000000   79.799479

      BMI  DiabetesPedigreeFunction  Age  Outcome
0    33.6                     0.627   50        1
1    26.6                     0.351   31        0
2    23.3                     0.672   32        1
3    28.1                     0.167   21        0
4    25.6                     0.201   30        0
..    ...                       ...  ...      ...
614  32.9                     0.171   63        0
615  36.8                     0.340   27        0
616  26.2                     0.245   30        0
617  30.1                     0.349   47        1
618  30.4                     0.315   23        0

[619 rows x 10 columns]
```

[23]:
```
plt.figure(figsize=(20,15), facecolor = "yellow")
plotnumber = 1  # take 1 column at a time plot it and then go to next

for column in data:
    if plotnumber<= 9:
        ax= plt.subplot(3,3,plotnumber)
        sns.distplot(data[column])
        plt.xlabel(column,fontsize = 20)

    plotnumber+=1
plt.show()
```



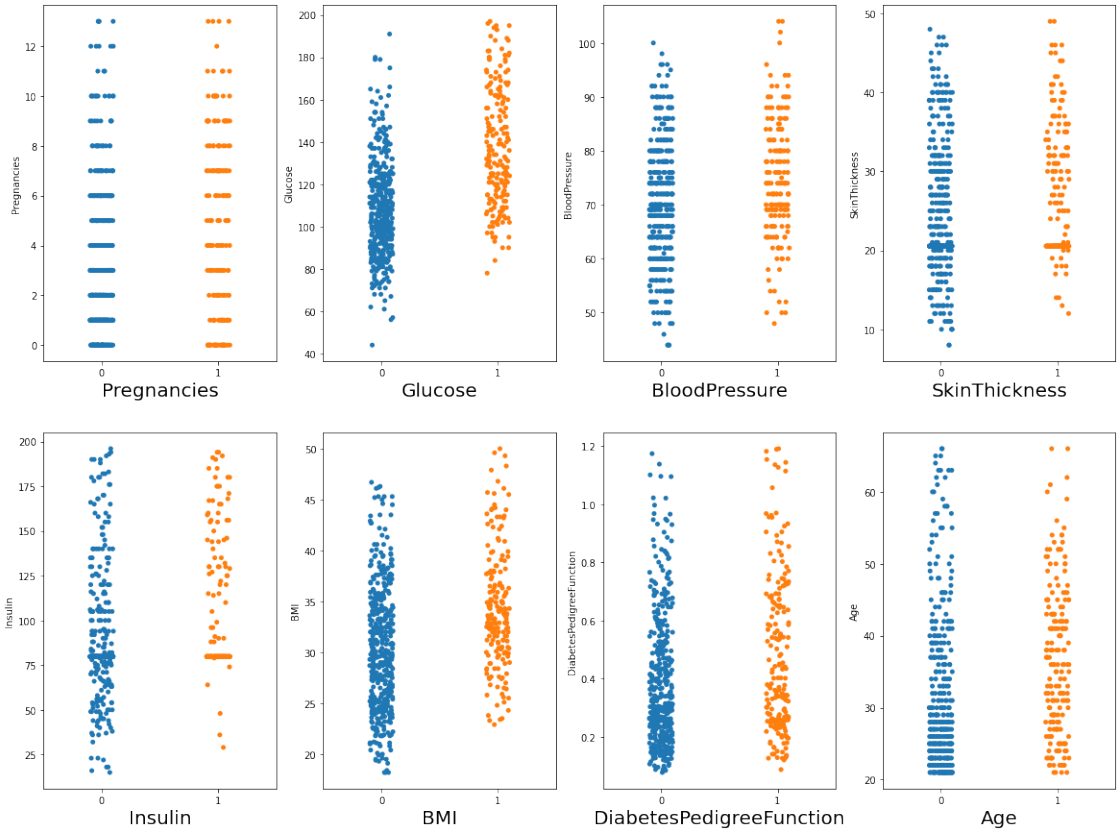
The data looks much better now than before we will start our analysis with this data now as we don't want to lose important information. if our model doesn't work with accuracy,we will come back for more preprocessing

[24]:
```
x = data.drop(columns = ['Outcome']) # Features
y = data['Outcome']
```

Before we fit our data to a model , let's visualize the relationship between our independent variables and the categories

[28]:
```
# let's see how features are related to class
plt.figure(figsize=(20,15))
plotnumber = 1  # take 1 column at a time plot it and then go to next

for column in x:
    if plotnumber<= 8:
        ax= plt.subplot(2,4,plotnumber)
        sns.stripplot(y,x[column])
        plt.xlabel(column,fontsize = 20)

    plotnumber+=1
```

```
plt.show()
```



Great! Let's proceed by checking multicolinearity in the dependent variables .Before that we should scale out data. Let's use the standar scaler for that

[29]:
```
scaler = StandardScaler()
x_scaled = scaler.fit_transform(x)
```

[30]:
```
x_scaled.shape[1]  # using[1] to print only columns
```

[30]: 8

[31]:
```
# Finding variance inflation factor in each scaled column i.e x_scaled.shape[1]

# Formula :- (1/(1-R2))
vif = pd.DataFrame() # creating a blank dataframe
vif['vif'] = [variance_inflation_factor(x_scaled,i) for i in range(x_scaled.
  ↪shape[1])]
vif['Features'] = x.columns

# Let's check the values
```

```
vif
```

[31]:

	vif	Features
0	1.448654	Pregnancies
1	1.250247	Glucose
2	1.258898	BloodPressure
3	1.411508	SkinThickness
4	1.200759	Insulin
5	1.447599	BMI
6	1.038530	DiabetesPedigreeFunction
7	1.659799	Age

All the VIF values are less than 5 and are very low. That means no multicollinearity. Now we can go ahend with fitting our data to the model. before that let's split our data in test and training set

[32]:
```
x_train,x_test,y_train,y_test = train_test_split(x_scaled,y, test_size =0.25 ,␣
↪random_state = 50)
```

[33]:
```
log = LogisticRegression()

log.fit(x_train,y_train)
```

[33]:
```
LogisticRegression()
```

[34]:
```
# Let's see how well our model performs on the test data set.

y_pred = log.predict(x_test)
```

[35]:
```
y_pred
```

[35]:
```
array([0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0,
       0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0,
       0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0,
       0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0,
       1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1,
       1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0,
       1], dtype=int64)
```

[36]:
```
# Model Accuracy
accuracy = accuracy_score(y_test,y_pred)
accuracy
```

[36]:
```
0.7741935483870968
```

[37]:
```
# Confusion Matrix
c_mat = confusion_matrix(y_test,y_pred)
```

```
c_mat
```

[37]:
```
array([[94, 10],
       [25, 26]], dtype=int64)
```

[39]:
```
from sklearn.metrics import classification_report
```

[40]:
```
print(classification_report(y_test,y_pred))
```

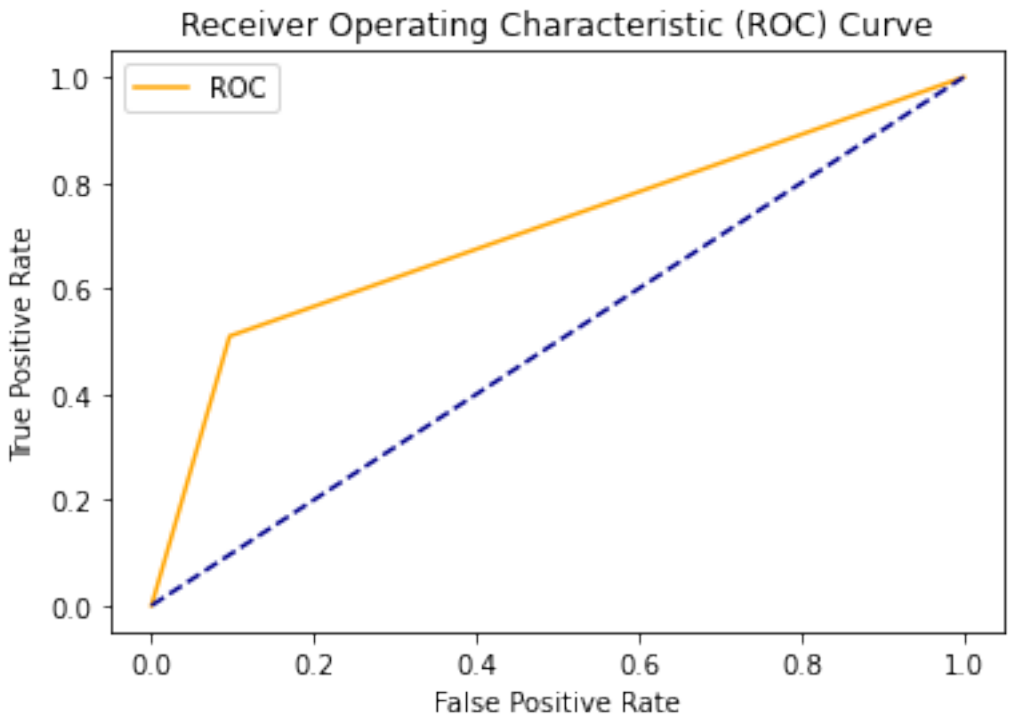
```
              precision    recall  f1-score   support

           0       0.79      0.90      0.84       104
           1       0.72      0.51      0.60        51

    accuracy                           0.77       155
   macro avg       0.76      0.71      0.72       155
weighted avg       0.77      0.77      0.76       155
```

[41]:
```
# ROC curve
fpr,tpr,th = roc_curve(y_test,y_pred)
```

[42]:
```
# thresholds[0]means no instances predicted ( it should be read from 0 - max)
print('Threshold =',th)
print('True positive rate =',tpr)
print('False positive rate =',fpr)
```

```
Threshold = [2 1 0]
True positive rate = [0.         0.50980392 1.        ]
False positive rate = [0.         0.09615385 1.        ]
```

[44]:
```
plt.plot(fpr,tpr,color='orange',label='ROC')
plt.plot ([0,1],[0,1] , color ='darkblue', linestyle ='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend()
plt.show()
```



[45]:
```
# How much area it is covering (AUC)
auc_score = roc_auc_score(y_test,y_pred)
print(auc_score)
```

```
0.7068250377073907
```

[ ]:

[ ]:

[ ]: