About Dataset

*Dependent Column : * Outcome (1: diabetic Patient , 0 : Non-diabetic patient)

*Independent Columns(8) : *

Pregnancies Glucose BloodPressure SkinThickness Insulin BMI DiabetesPedigreeFunction Age

Aim of the Project : **Can you build a model (Machine learning or deep learning) to accurately predict whether or not the patients in the dataset have diabetes or not?**

```
#Normal Library
#-----------------------
import numpy as np
import pandas as pd


#Visulization Library
#--------------------
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
sns.set()

#--------------------------------
#Warning library
import warnings
warnings.filterwarnings('ignore')


#anova testing
import scipy.stats as stats

#scaling the data
from sklearn.preprocessing import StandardScaler

#Train Test split
from sklearn.model_selection import train_test_split


#ML model - regeression
from sklearn.linear_model import LinearRegression

#ML model - regeression - Performance matrix
from sklearn import metrics
from sklearn.metrics import r2_score,mean_absolute_percentage_error, mean_squared_error



#ML model - Classification

#ML model - Classification - Performance matrix
from sklearn import metrics
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score, roc_auc_score


#Improve the accuracy
from sklearn.model_selection import cross_val_score
```

```
df= pd.read_csv(r"/content/diabetes.csv")
```

```
df.head()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunctio
0	6	148	72	35	0	33.6	0.62
1	1	85	66	29	0	26.6	0.35
2	8	183	64	0	0	23.3	0.67
3	1	89	66	23	94	28.1	0.16
4	0	137	40	35	168	43.1	2.28

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Pregnancies               768 non-null    int64
 1   Glucose                   768 non-null    int64
 2   BloodPressure             768 non-null    int64
 3   SkinThickness             768 non-null    int64
 4   Insulin                   768 non-null    int64
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64
 8   Outcome                   768 non-null    int64
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

Checking Duplicate rows

```
def drop_dup(df):
  if df.duplicated().any() == True:
    df.drop_duplicates(inplace= True, Keep = "Last",reset_index = True)
    print("Data after removing duplicates row :" , df.dupdated().sum())
  else:
    return "No action required( No duplicate found)"

drop_dup(df)
```

```
'No action required( No duplicate found)'
```

Checking Null Values

```
print(df.isnull().sum())
print("*****")
print(df.isnull().sum()/len(df)*100)
```

```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
*****
Pregnancies                 0.0
Glucose                     0.0
BloodPressure               0.0
SkinThickness               0.0
Insulin                     0.0
BMI                         0.0
DiabetesPedigreeFunction    0.0
Age                         0.0
Outcome                     0.0
dtype: float64
```

Check unique counts

```
def check_unique_count(df):
  unique_counts = df.nunique()
  print(unique_counts)


check_unique_count(df)
```

```
Pregnancies                  17
Glucose                     136
BloodPressure                47
SkinThickness                51
Insulin                     186
BMI                         248
DiabetesPedigreeFunction    517
Age                          52
Outcome                       2
dtype: int64
```

Check unique counts data entry in columns

```
for i in df.columns:
  print(i)
  print("******")
  print(set(df[i].tolist()))
  print("___________")
```

```
Pregnancies
******
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 17}
___________
Glucose
******
{0, 44, 56, 57, 61, 62, 65, 67, 68, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94,
___________
BloodPressure
******
{0, 24, 30, 38, 40, 44, 46, 48, 50, 52, 54, 55, 56, 58, 60, 61, 62, 64, 65, 66, 68, 70, 72, 74, 75, 76, 78, 80, 82, 84, 85, 86, 88,
___________
SkinThickness
******
{0, 7, 8, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 4
___________
Insulin
******
{0, 14, 15, 16, 18, 22, 23, 25, 540, 29, 543, 32, 545, 36, 37, 38, 40, 41, 42, 43, 44, 45, 46, 48, 49, 50, 51, 52, 53, 54, 55, 56,
___________
BMI
******
{0.0, 37.3, 18.4, 19.9, 19.4, 19.6, 22.2, 23.3, 24.4, 23.2, 25.8, 27.6, 27.4, 28.0, 28.9, 28.6, 32.8, 29.0, 29.7, 26.6, 27.1, 28.1,
___________
DiabetesPedigreeFunction
******
{0.351, 0.484, 1.189, 0.375, 0.875, 0.851, 0.665, 1.101, 1.476, 0.391, 1.391, 2.42, 0.398, 0.828, 0.218, 0.101, 0.226, 0.234, 0.21,
___________
Age
******
{21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53
___________
Outcome
******
{0, 1}
___________
```

data analyzized : columns -['Glucose', 'BloodPressure', 'SkinThickness''Insulin', 'BMI'] has value 0 which is wrong.

```
Consider_null_values =  ['Glucose', 'BloodPressure', 'SkinThickness','Insulin', 'BMI' ]
for col in Consider_null_values:
  df[col].replace(0, np.nan, inplace=True)
```

```
df.isnull().sum()
```

```
Pregnancies                   0
Glucose                       5
BloodPressure                35
SkinThickness               227
Insulin                     374
BMI                          11
DiabetesPedigreeFunction      0
Age                           0
Outcome                       0
dtype: int64
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Pregnancies               768 non-null    int64
 1   Glucose                   763 non-null    float64
 2   BloodPressure             733 non-null    float64
 3   SkinThickness             541 non-null    float64
 4   Insulin                   394 non-null    float64
 5   BMI                       757 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64
 8   Outcome                   768 non-null    int64
dtypes: float64(6), int64(3)
memory usage: 54.1 KB
```

```
#Using imputer method to fill the values.

df["Glucose"] = df["Glucose"].fillna(df["Glucose"].median())
df["BloodPressure"] = df["BloodPressure"].fillna(df["BloodPressure"].median())
df["SkinThickness"] = df["SkinThickness"].fillna(df["SkinThickness"].median())
df["Insulin"] = df["Insulin"].fillna(df["Insulin"].median())
df["BMI"] = df["BMI"].fillna(df["BMI"].median())
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Pregnancies               768 non-null    int64
 1   Glucose                   768 non-null    float64
 2   BloodPressure             768 non-null    float64
 3   SkinThickness             768 non-null    float64
 4   Insulin                   768 non-null    float64
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64
 8   Outcome                   768 non-null    int64
dtypes: float64(6), int64(3)
memory usage: 54.1 KB
```

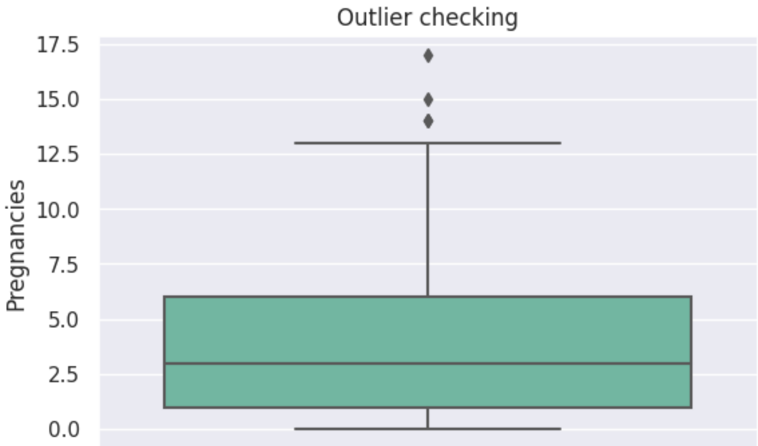
```
for i in df.columns:
  print(i)
  print("******")
  print(set(df[i].tolist()))
  print("___________")
```

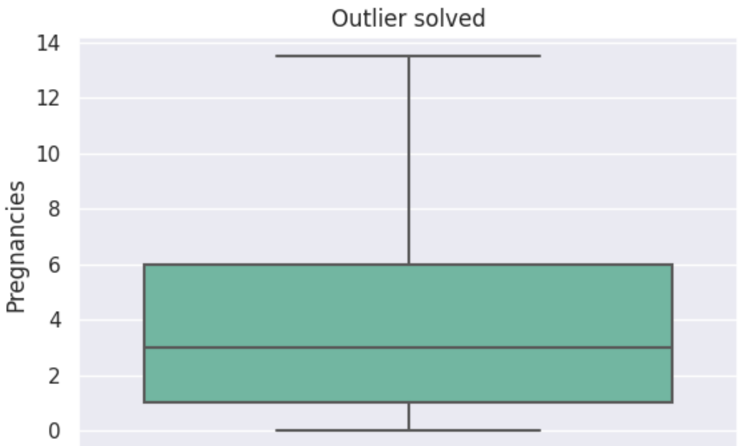
```
Pregnancies
******
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 17}
___________
Glucose
******
{44.0, 56.0, 57.0, 61.0, 62.0, 65.0, 67.0, 68.0, 71.0, 72.0, 73.0, 74.0, 75.0, 76.0, 77.0, 78.0, 79.0, 80.0, 81.0, 82.0, 83.0, 84.0
___________
BloodPressure
******
{24.0, 30.0, 38.0, 40.0, 44.0, 46.0, 48.0, 50.0, 52.0, 54.0, 55.0, 56.0, 58.0, 60.0, 61.0, 62.0, 64.0, 65.0, 66.0, 68.0, 70.0, 72.0
___________
SkinThickness
******
{7.0, 8.0, 10.0, 11.0, 12.0, 13.0, 14.0, 15.0, 16.0, 17.0, 18.0, 19.0, 20.0, 21.0, 22.0, 23.0, 24.0, 25.0, 26.0, 27.0, 28.0, 29.0,
___________
Insulin
******
{14.0, 15.0, 16.0, 18.0, 22.0, 23.0, 25.0, 540.0, 29.0, 543.0, 32.0, 545.0, 36.0, 37.0, 38.0, 40.0, 41.0, 42.0, 43.0, 44.0, 45.0, 4
___________
BMI
******
{37.3, 18.4, 19.9, 19.4, 19.6, 22.2, 23.3, 24.4, 23.2, 25.8, 27.6, 27.4, 28.0, 28.9, 28.6, 32.8, 29.0, 29.7, 26.6, 27.1, 28.1, 30.1
___________
DiabetesPedigreeFunction
******
{0.351, 0.484, 1.189, 0.375, 0.875, 0.851, 0.665, 1.101, 1.476, 0.391, 1.391, 2.42, 0.398, 0.828, 0.218, 0.101, 0.226, 0.234, 0.21,
___________
Age
******
{21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53
___________
Outcome
******
{0, 1}
___________
```

Outliers Check -

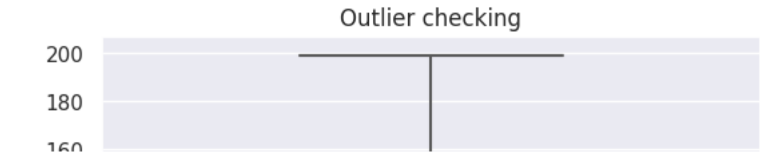
```
#column name - Pregnancies as Preg
plt.figure(figsize=(14,4))
plt.subplot(121)
sns.boxplot(y = "Pregnancies", data = df, palette ='Set2' )
plt.title("Outlier checking")

#######################
```

```
Preg_q1 = df['Pregnancies'].quantile(0.25)
Preg_q3 = df['Pregnancies'].quantile(0.75)
Preg_iqr = Preg_q3 - Preg_q1
Preg_upper = Preg_q3 + 1.5 * Preg_iqr
Preg_lower = Preg_q1 - 1.5 * Preg_iqr

df['Pregnancies'] = np.where(df['Pregnancies'] > Preg_upper,Preg_upper,
                                     np.where(df['Pregnancies'] < Preg_lower, Preg_lower,
                                            df['Pregnancies']) )

###############################################
plt.figure(figsize=(14,4))
plt.subplot(122)
sns.boxplot(y = "Pregnancies", data = df, palette ='Set2')
plt.title("Outlier solved")


plt.show()
```

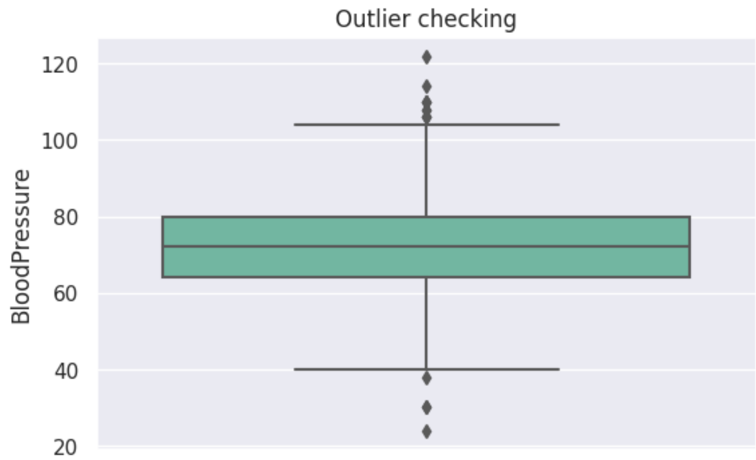


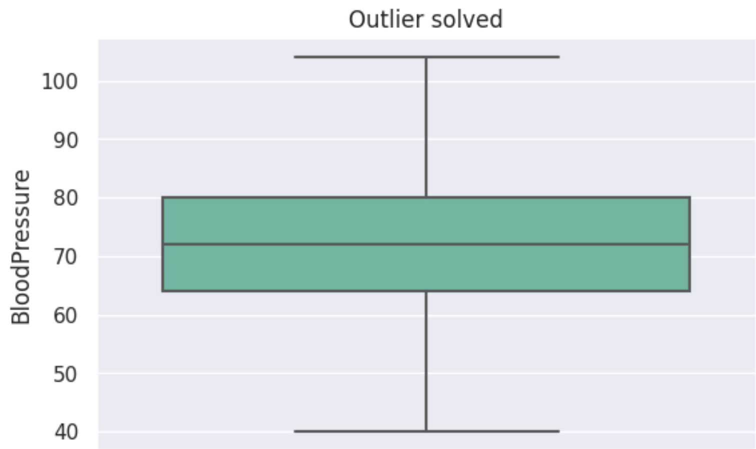



```
#column name - Glucose as Glu
plt.figure(figsize=(14,4))
plt.subplot(121)
sns.boxplot(y = "Glucose", data = df, palette ='Set2' )
plt.title("Outlier checking")

#######################


plt.show()
```



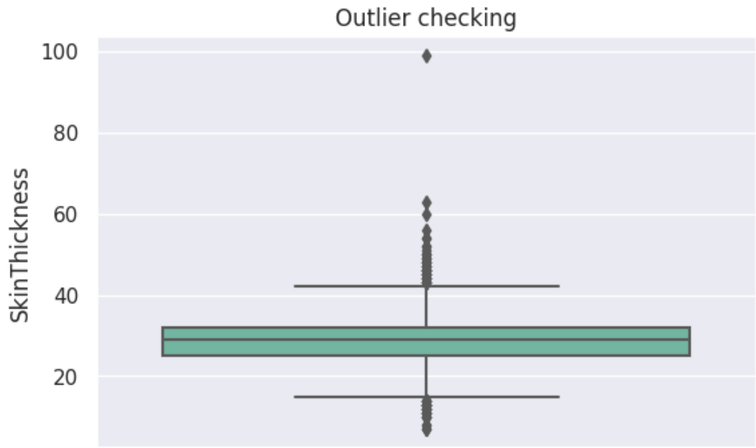

```
#column name - BloodPressure as blood
plt.figure(figsize=(14,4))
plt.subplot(121)
sns.boxplot(y = "BloodPressure", data = df, palette ='Set2' )
plt.title("Outlier checking")

#######################
blood_q1 = df['BloodPressure'].quantile(0.25)
blood_q3 = df['BloodPressure'].quantile(0.75)
blood_iqr = blood_q3 - blood_q1
blood_upper = blood_q3 + 1.5 * blood_iqr
blood_lower = blood_q1 - 1.5 * blood_iqr

df['BloodPressure'] = np.where(df['BloodPressure'] > blood_upper,blood_upper,
                                     np.where(df['BloodPressure'] < blood_lower, blood_lower,
                                            df['BloodPressure']) )

###############################################
plt.figure(figsize=(14,4))
plt.subplot(122)
sns.boxplot(y = "BloodPressure", data = df, palette ='Set2')
plt.title("Outlier solved")

plt.show()
```

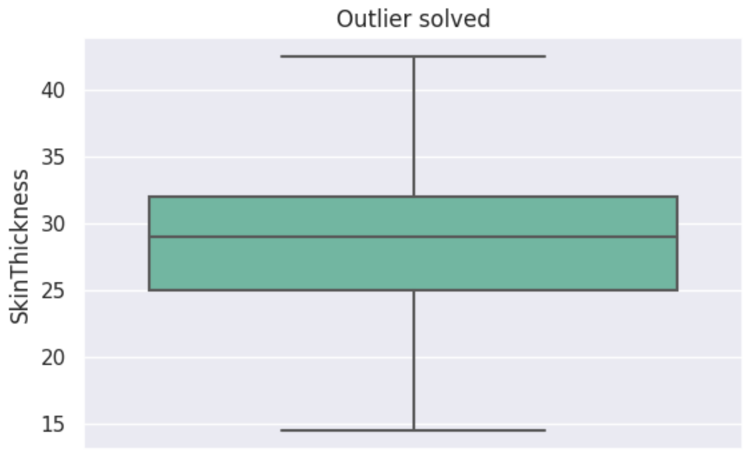




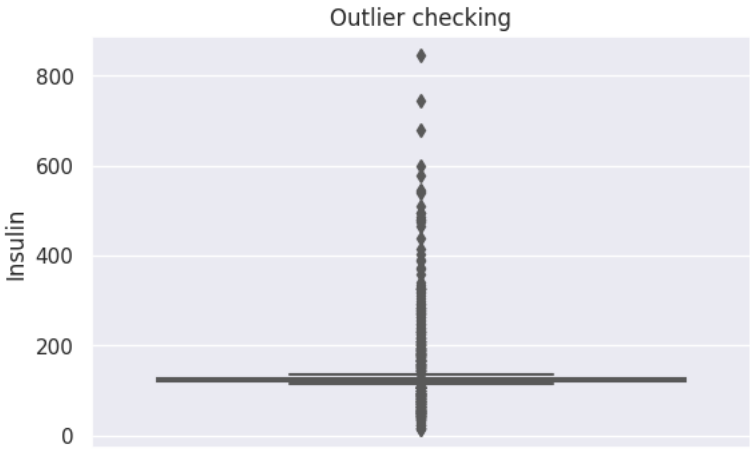

```
 #Column SkinThickness as Skin

plt.figure(figsize=(14,4))
plt.subplot(121)
sns.boxplot(y = "SkinThickness", data = df, palette ='Set2' )
plt.title("Outlier checking")


#######################
Skin_q1 = df['SkinThickness'].quantile(0.25)
Skin_q3 = df['SkinThickness'].quantile(0.75)
```

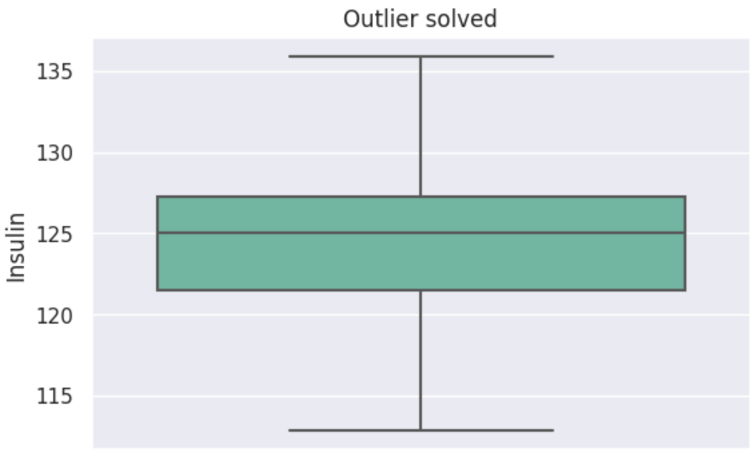
```
Skin_iqr = Skin_q3 - Skin_q1
Skin_upper = Skin_q3 + 1.5 * Skin_iqr
Skin_lower = Skin_q1 - 1.5 * Skin_iqr

df['SkinThickness'] = np.where(df['SkinThickness'] > Skin_upper,Skin_upper,
                                     np.where(df['SkinThickness'] < Skin_lower, Skin_lower,
                                            df['SkinThickness']) )

###############################################
plt.figure(figsize=(14,4))
plt.subplot(122)
sns.boxplot(y = "SkinThickness", data = df, palette ='Set2')
plt.title("Outlier solved")

plt.show()
```



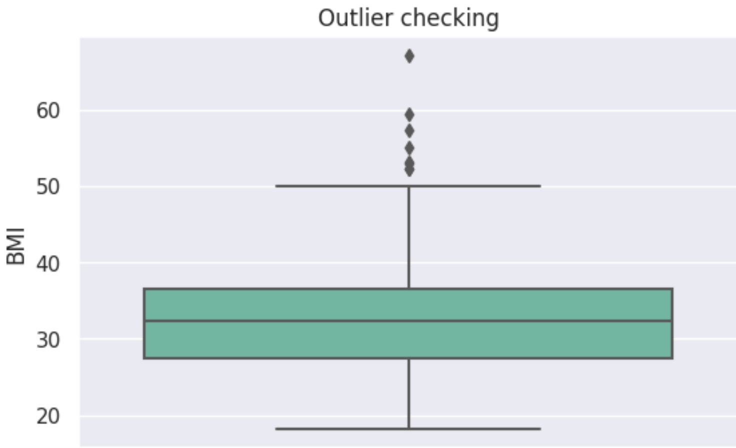



```
# Column Insulin as Insulin

plt.figure(figsize=(14,4))
plt.subplot(121)
sns.boxplot(y = "Insulin", data = df, palette ='Set2' )
plt.title("Outlier checking")


#######################
Insulin_q1 = df['Insulin'].quantile(0.25)
Insulin_q3 = df['Insulin'].quantile(0.75)
Insulin_iqr = Insulin_q3 -Insulin_q1
Insulin_upper = Insulin_q3 + 1.5 * Insulin_iqr
Insulin_lower = Insulin_q1 - 1.5 * Insulin_iqr

df['Insulin'] = np.where(df['Insulin'] > Insulin_upper,Insulin_upper,
                                     np.where(df['Insulin'] < Insulin_lower, Insulin_lower,
                                            df['Insulin']) )

###############################################
plt.figure(figsize=(14,4))
plt.subplot(122)
sns.boxplot(y = "Insulin", data = df, palette ='Set2')
plt.title("Outlier solved")

plt.show()
```





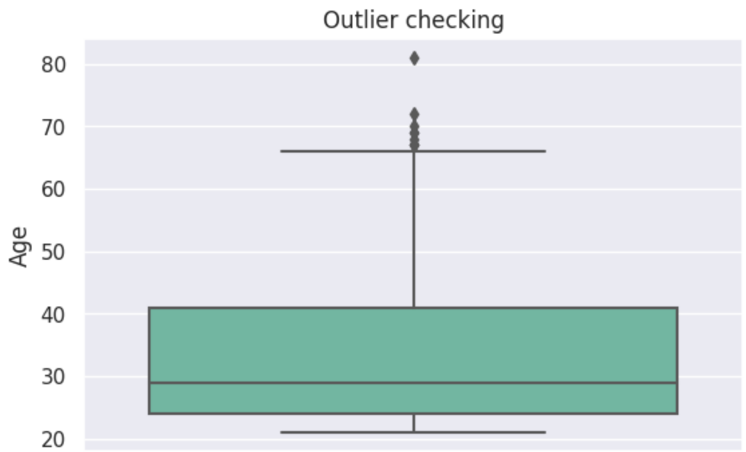

```
# Column BMI

plt.figure(figsize=(14,4))
plt.subplot(121)
sns.boxplot(y = "BMI", data = df, palette ='Set2' )
plt.title("Outlier checking")

#######################
BMI_q1 = df['BMI'].quantile(0.25)
BMI_q3 = df['BMI'].quantile(0.75)
BMI_iqr = BMI_q3 -BMI_q1
BMI_upper = BMI_q3 + 1.5 * BMI_iqr
BMI_lower = BMI_q1 - 1.5 * BMI_iqr

df['BMI'] = np.where(df['BMI'] > BMI_upper,BMI_upper,
                                     np.where(df['BMI'] < BMI_lower, BMI_lower,
                                            df['BMI']) )

###############################################
plt.figure(figsize=(14,4))
plt.subplot(122)
sns.boxplot(y = "BMI", data = df, palette ='Set2')
plt.title("Outlier solved")

plt.show()
```

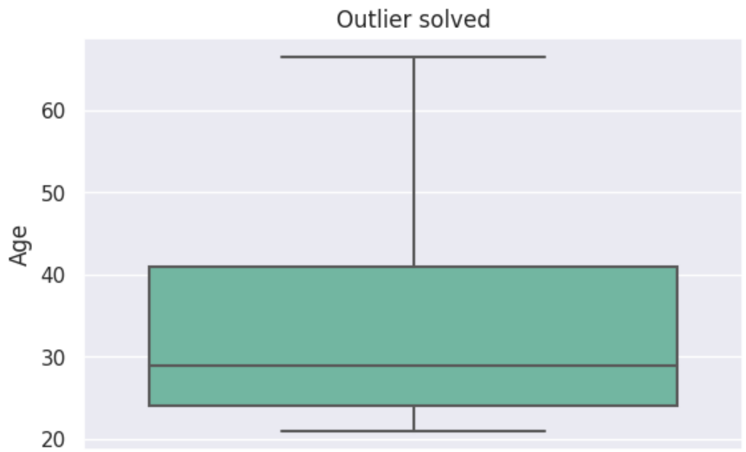


Outlier solved

```
# column - DiabetesPedigreeFunction

plt.figure(figsize=(14,4))
plt.subplot(121)
sns.boxplot(y = "DiabetesPedigreeFunction", data = df, palette ='Set2' )
plt.title("Outlier checking")

#######################
DiabetesPedigreeFunction_q1 = df['DiabetesPedigreeFunction'].quantile(0.25)
DiabetesPedigreeFunction_q3 = df['DiabetesPedigreeFunction'].quantile(0.75)
DiabetesPedigreeFunction_iqr = DiabetesPedigreeFunction_q3 -DiabetesPedigreeFunction_q1
DiabetesPedigreeFunction_upper = DiabetesPedigreeFunction_q3 + 1.5 * DiabetesPedigreeFunction_iqr
DiabetesPedigreeFunction_lower = DiabetesPedigreeFunction_q1 - 1.5 * DiabetesPedigreeFunction_iqr

df['DiabetesPedigreeFunction'] = np.where(df['DiabetesPedigreeFunction'] > DiabetesPedigreeFunction_upper,DiabetesPedigreeFunction_upper,
                                     np.where(df['DiabetesPedigreeFunction'] < DiabetesPedigreeFunction_lower, DiabetesPedigreeFunction_l
                                            df['DiabetesPedigreeFunction']) )

###############################################
plt.figure(figsize=(14,4))
plt.subplot(122)
sns.boxplot(y = "DiabetesPedigreeFunction", data = df, palette ='Set2')
plt.title("Outlier solved")

plt.show()
```

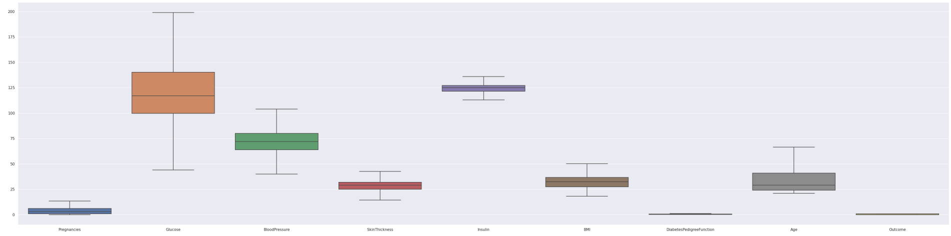

```
#Column - Age

plt.figure(figsize=(14,4))
plt.subplot(121)
sns.boxplot(y = "Age", data = df, palette ='Set2' )
plt.title("Outlier checking")

#######################
Age_q1 = df['Age'].quantile(0.25)
Age_q3 = df['Age'].quantile(0.75)
Age_iqr = Age_q3 -Age_q1
Age_upper = Age_q3 + 1.5 * Age_iqr
Age_lower = Age_q1 - 1.5 * Age_iqr

df['Age'] = np.where(df['Age'] > Age_upper,Age_upper,
                                     np.where(df['Age'] < Age_lower, Age_lower,
                                            df['Age']) )

###############################################
plt.figure(figsize=(14,4))
plt.subplot(122)
sns.boxplot(y = "Age", data = df, palette ='Set2')
plt.title("Outlier solved")

plt.show()
```






```
plt.figure(figsize=(50,12))
sns.boxplot(df)
```

```
<Axes: >
```



Normal Distribution check Skewenss Check***

```
for i in df.columns:
  print(df[i].skew())

#skewness available
```

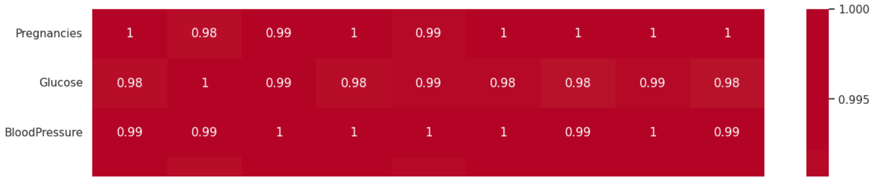
```
0.8539617478323778
0.5355873034111183
0.10566506007005463
-0.05764290402870092
-0.11822095592136406
0.34988208669393905
1.0244278033317116
1.0671703233262797
0.635016643444986
```

```
df.columns
```

```
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
      dtype='object')
```

Finding correlation it is not required for clinical dataset but just t see the graph

```
plt.figure(figsize=(15,8))
corr = df.describe().corr()
sns.heatmap(corr, annot=True, cmap='coolwarm',center = 0)
plt.show()
```



Visulize the data

```
sns.pairplot(df, size = 5, kind = 'scatter')
plt.show()
```

Pair plot is indicating to use logistic classification model

```
pip install movecolumn
```

```
Requirement already satisfied: movecolumn in /usr/local/lib/python3.10/dist-packages (0.0.7)
```

```
import movecolumn as mc
mc.MoveToLast(df,'Outcome')
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunct
0	6.0	148.0	72.0	35.0	125.000	33.6	0.
1	1.0	85.0	66.0	29.0	125.000	26.6	0.
2	8.0	183.0	64.0	29.0	125.000	23.3	0.
3	1.0	89.0	66.0	23.0	112.875	28.1	0.
4	0.0	137.0	40.0	35.0	135.875	43.1	1.
...	...	...	...	...	...	...	
763	10.0	101.0	76.0	42.5	135.875	32.9	0.
764	2.0	122.0	70.0	27.0	125.000	36.8	0.
765	5.0	121.0	72.0	23.0	112.875	26.2	0.
766	1.0	126.0	60.0	29.0	125.000	30.1	0.
767	1.0	93.0	70.0	31.0	125.000	30.4	0.

768 rows × 9 columns

Spliting train test

```
x = df.iloc[:,0:-1]
y = df['Outcome']
```

Scailing the dataset

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
sc_x = sc.fit_transform(x)
pd.DataFrame(sc_x)

variable = sc_x
variable.shape
```

```
(768, 8)
```

imbalance dataset handling

```
y.value_counts()
```

```
0    500
1    268
Name: Outcome, dtype: int64
```

```
# Smote method to handle imbalanced dataset

from imblearn.over_sampling import SMOTE
smote = SMOTE()
```

```
x, y = smote.fit_resample(x,y)
y.value_counts()
```

```
1    500
0    500
Name: Outcome, dtype: int64
```

```
#scailing dataset

from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
sc_x = sc.fit_transform(x)
pd.DataFrame(sc_x)

variable = sc_x
variable.shape
```

```
(1000, 8)
```

Split the data into train and test

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(sc_x, y, test_size=0.2, random_state=101)
```

transform the data through yeo-johnson method

```
from sklearn.preprocessing import PowerTransformer

pt1 = PowerTransformer(method='yeo-johnson')

x_train_transformed = pt1.fit_transform(x_train)
x_test_transformed = pt1.fit_transform(x_test)
```

Logistic regeression model

```
from sklearn.linear_model import LogisticRegression

logistic = LogisticRegression()
logistic.fit(x_train_transformed, y_train)
y_train_transformed_pred = logistic.predict(x_train_transformed)
y_test_transformed_pred = logistic.predict(x_test_transformed)

print(accuracy_score(y_train, y_train_transformed_pred))
print(accuracy_score(y_test, y_test_transformed_pred))
```

```
0.78125
0.76
```

Result : model accuracy without CVS - train(78 %) , test (76 %)

```
#CVS
```

```
training_accuracy = cross_val_score(logistic, x_train_transformed, y_train, cv=15)
test_accuracy = cross_val_score(logistic, x_test_transformed, y_test, cv=15)
print(training_accuracy[14])
print(test_accuracy[14])
```

```
0.8301886792452831
0.8461538461538461
```

Result : model accuracy with CVS - train(83 %) , test (84 %)

Performace matrix

```
print(classification_report(y_train, y_train_transformed_pred))
print(classification_report(y_test, y_test_transformed_pred))
```

```
print("________________________________________________________")

print(confusion_matrix(y_train, y_train_transformed_pred))
print(confusion_matrix(y_test, y_test_transformed_pred))

print("________________________________________________________")

print(roc_auc_score(y_train, y_train_transformed_pred))
print(roc_auc_score(y_test, y_test_transformed_pred))
```

```
              precision    recall  f1-score   support

           0       0.79      0.78      0.79       409
           1       0.77      0.78      0.78       391

    accuracy                           0.78       800
   macro avg       0.78      0.78      0.78       800
weighted avg       0.78      0.78      0.78       800

              precision    recall  f1-score   support

           0       0.72      0.77      0.74        91
           1       0.80      0.75      0.77       109

    accuracy                           0.76       200
   macro avg       0.76      0.76      0.76       200
weighted avg       0.76      0.76      0.76       200

________________________________________________________
[[320  89]
 [ 86 305]]
[[70 21]
 [27 82]]
________________________________________________________
0.7812236194573503
0.7607621736062102
```

KNN Model - Mostly Data scientist used to prefer this model for cinical sector datset

```
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier()
knn.fit(x_train_transformed, y_train)

y_train_pred_knn = knn.predict(x_train_transformed)
y_test_pred_knn = knn.predict(x_test_transformed)

accuracy_knn_train = accuracy_score(y_train, y_train_pred_knn)
accuracy_knn_test = accuracy_score(y_test, y_test_pred_knn)
print(accuracy_knn_train)
print(accuracy_knn_test)
```

```
0.8275
0.76
```

```
#CVS

training_accuracy = cross_val_score(knn, x_train_transformed, y_train, cv=15)
test_accuracy = cross_val_score(knn, x_test_transformed, y_test, cv=15)
print(training_accuracy[2])
print(test_accuracy[2])
```

```
0.7777777777777778
0.7857142857142857
```