```
# Data Loading
from google.colab import drive
import pandas as pd
drive.mount('/content/drive')
file_path = '/content/drive/My Drive/diabetes.csv'
df = pd.read_csv(file_path)
```

```
Mounted at /content/drive
```

```
df.head()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Diabete
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	

Next steps: Generate code with df | View recommended plots

```
df.tail()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Diabet
763	10	101	76	48	180	32.9	
764	2	122	70	27	0	36.8	
765	5	121	72	23	112	26.2	
766	1	126	60	0	0	30.1	
767	1	93	70	31	0	30.4	

```
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
```

```
df.columns
```

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

```
df.describe()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.00
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.99
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.88
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.00
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.30
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.00
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.60
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.10

```
df.shape
```

```
(768, 9)
```

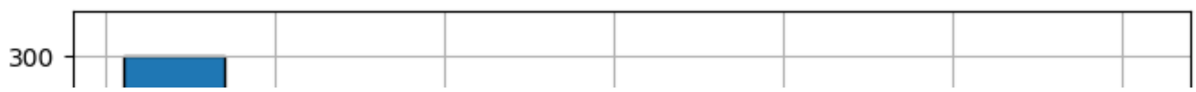
```
# distribution of outcome variable
df.Outcome.value_counts()*100/len(df)
df['Outcome'].value_counts()*100/len(df)
```

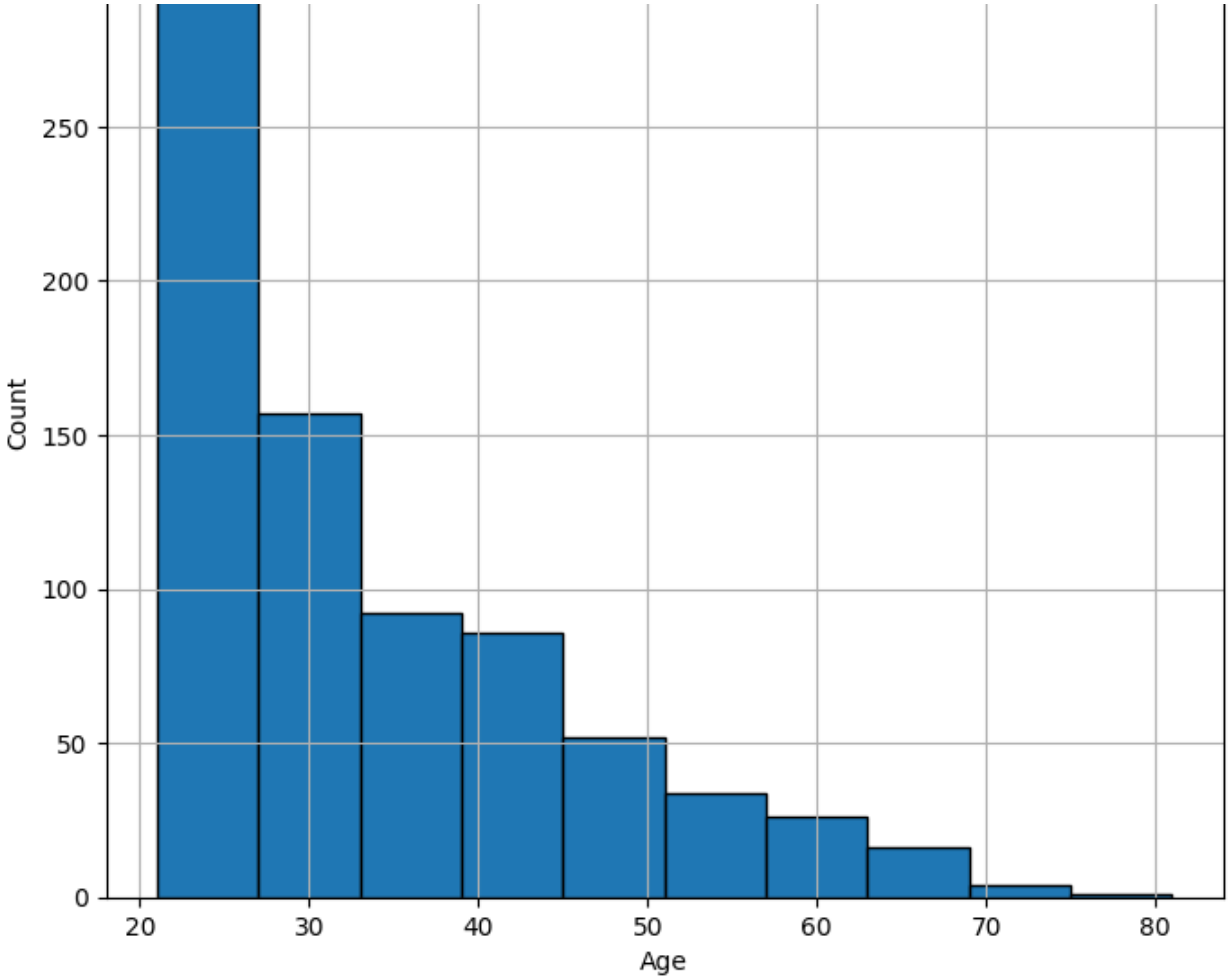
```
Outcome
0    65.104167
1    34.895833
```

```
Name: count, dtype: float64
```

```
# plot the hist of the age variable
import matplotlib.pyplot as plt
plt.figure(figsize=(8,7))
plt.xlabel('Age', fontsize=10)
plt.ylabel('Count', fontsize=10)
df['Age'].hist(edgecolor="black")
```

```
<Axes: xlabel='Age', ylabel='Count'>
```



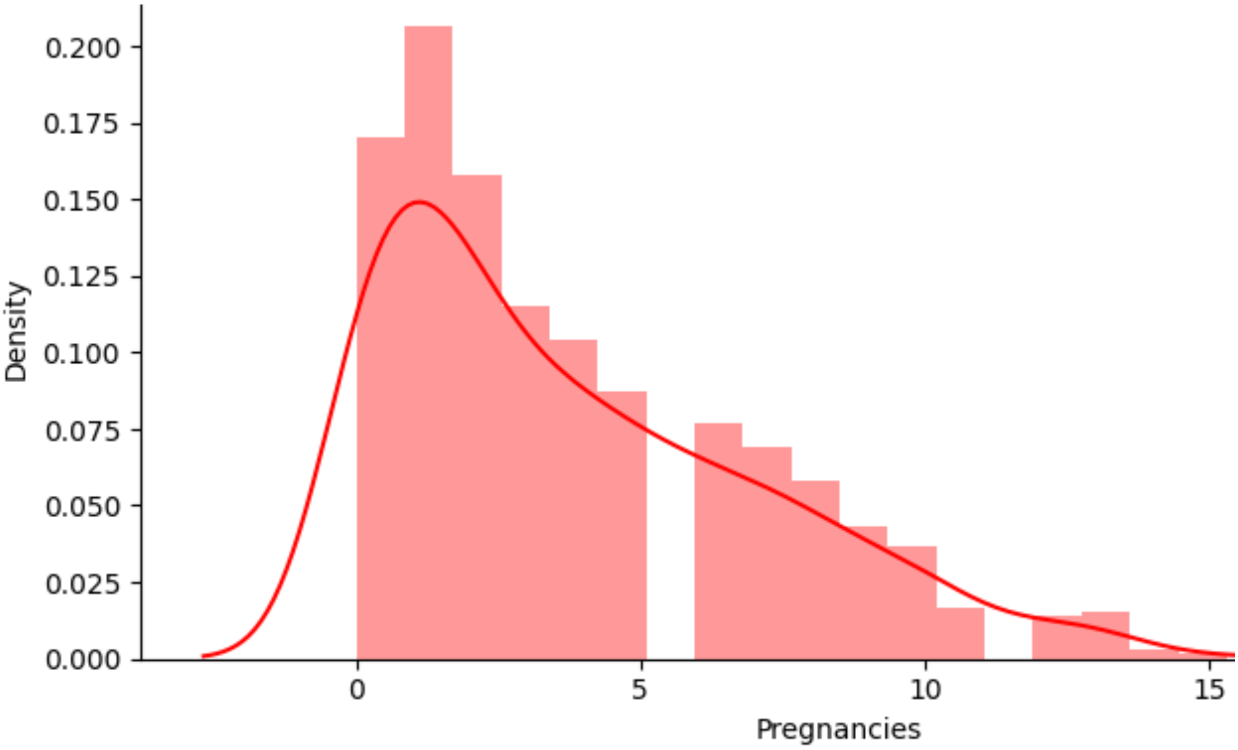


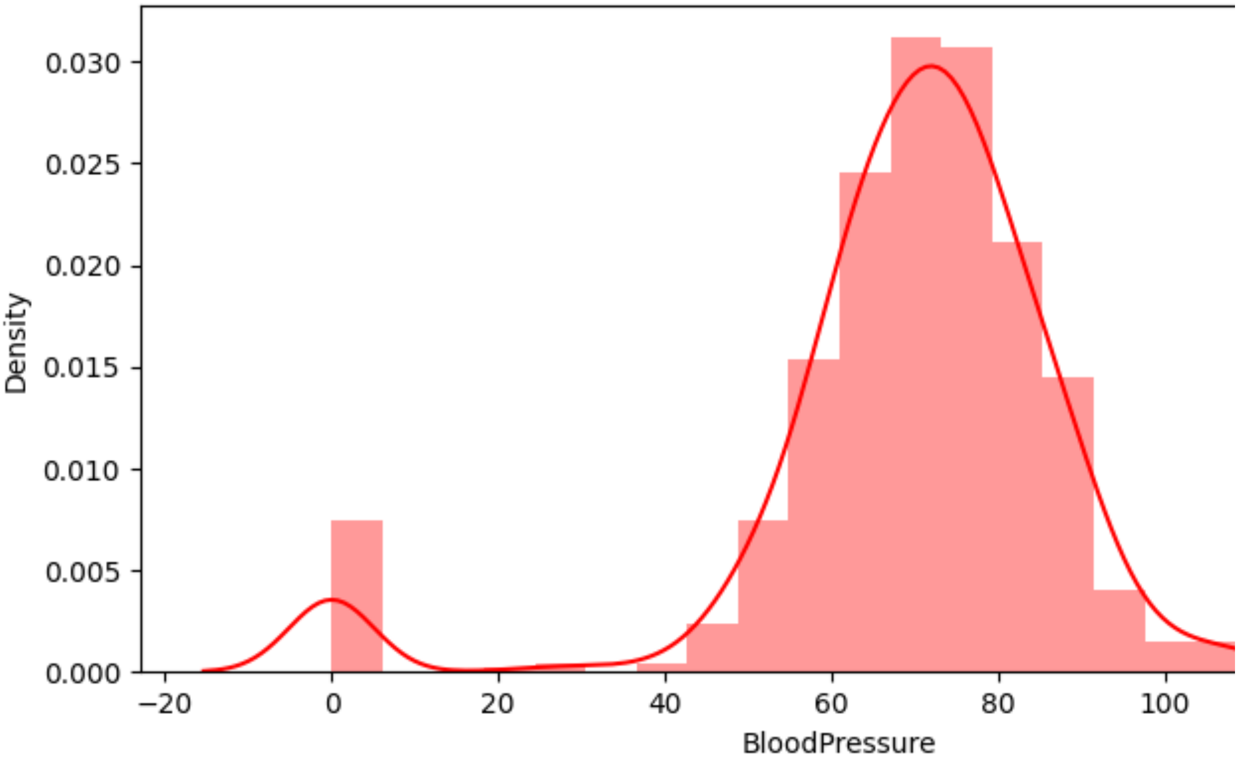

```
print("MAX AGE: "+str(df['Age'].max()))
print("MIN AGE: "+str(df['Age'].min()))
```

```
MAX AGE: 81
MIN AGE: 21
```

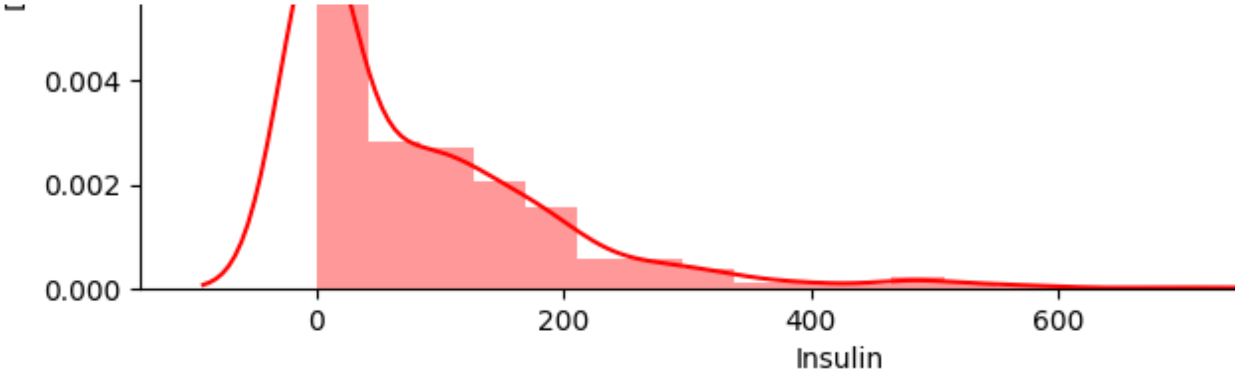
```
import seaborn as sns
import warnings
warnings.simplefilter(action='ignore')
fig,ax = plt.subplots(4,2, figsize=(20,20))
sns.distplot(df.Pregnancies, bins=20, ax=ax[0,0], color="red")
sns.distplot(df.Glucose, bins=20, ax=ax[0,1], color="red")
sns.distplot(df.BloodPressure, bins=20, ax=ax[1,0], color="red")
sns.distplot(df.SkinThickness, bins=20, ax=ax[1,1], color="red")
sns.distplot(df.Insulin, bins=20, ax=ax[2,0], color="red")
sns.distplot(df.BMI, bins=20, ax=ax[2,1], color="red")
sns.distplot(df.DiabetesPedigreeFunction, bins=20, ax=ax[3,0], color="red")
sns.distplot(df.Age, bins=20, ax=ax[3,1], color="red")
```

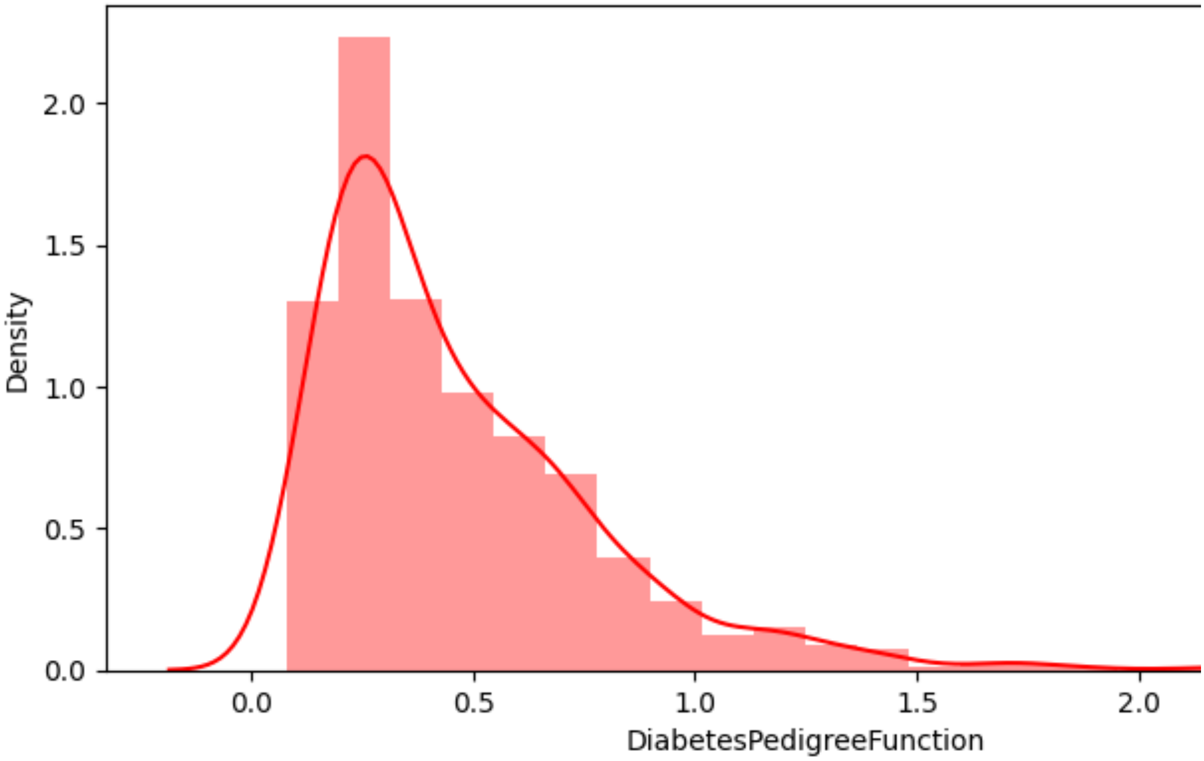
```
<Axes: xlabel='Age', ylabel='Density'>
```


0.200
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
Density
0
5
10
15
Pregnancies


0.030
0.025
0.020
0.015
0.010
0.005
0.000
Density
−20
0
20
40
60
80
100
BloodPressure

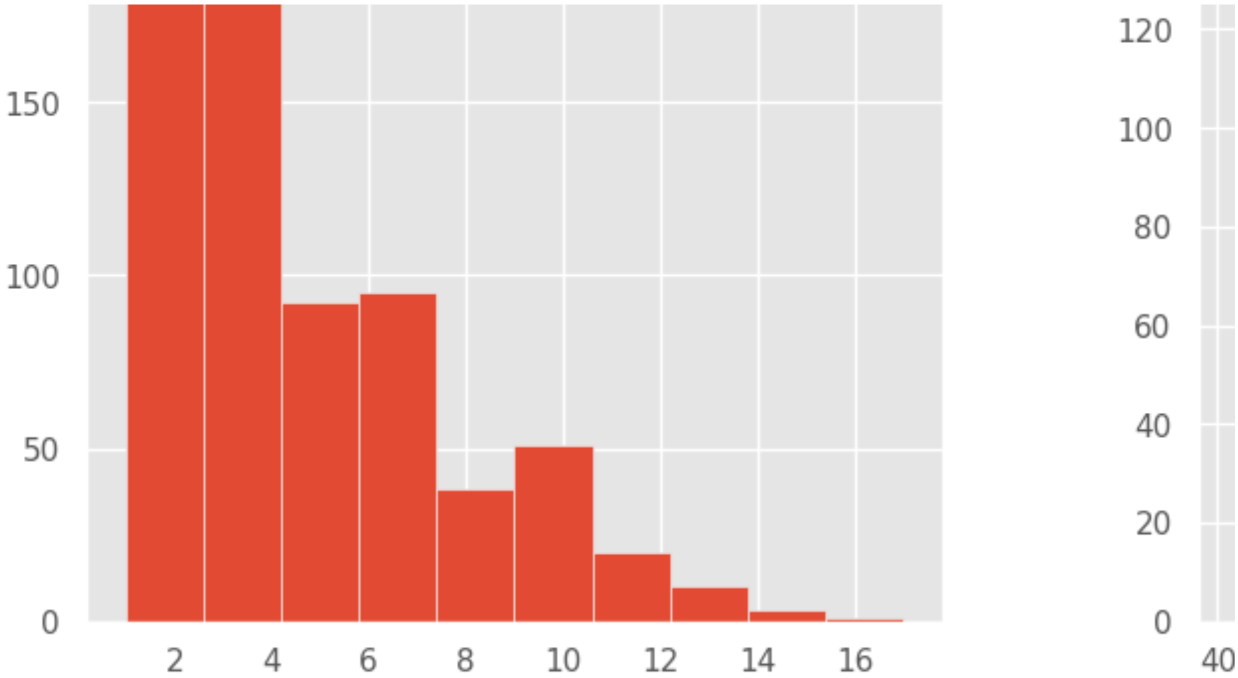

0.012
0.010
0.008
0.006

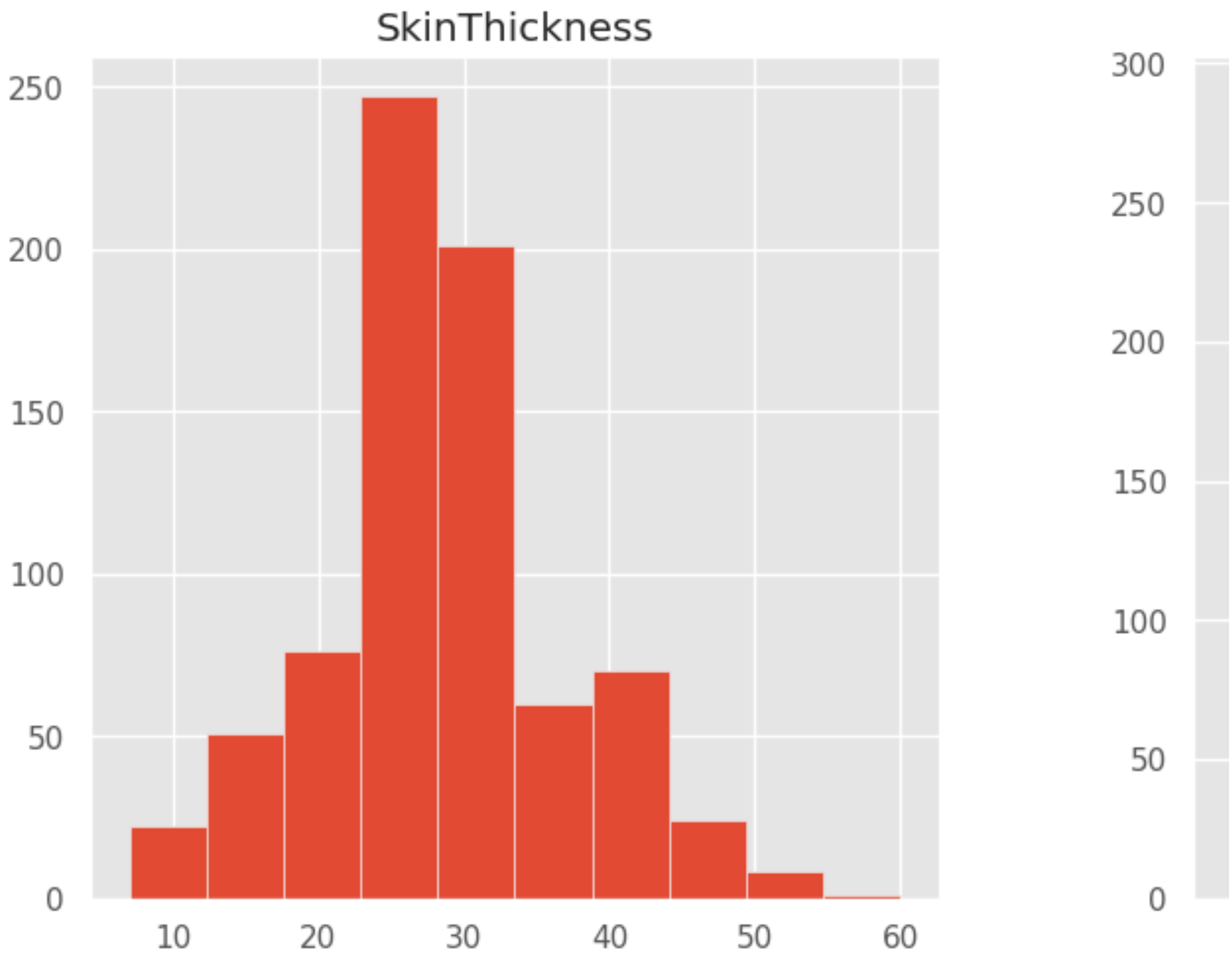




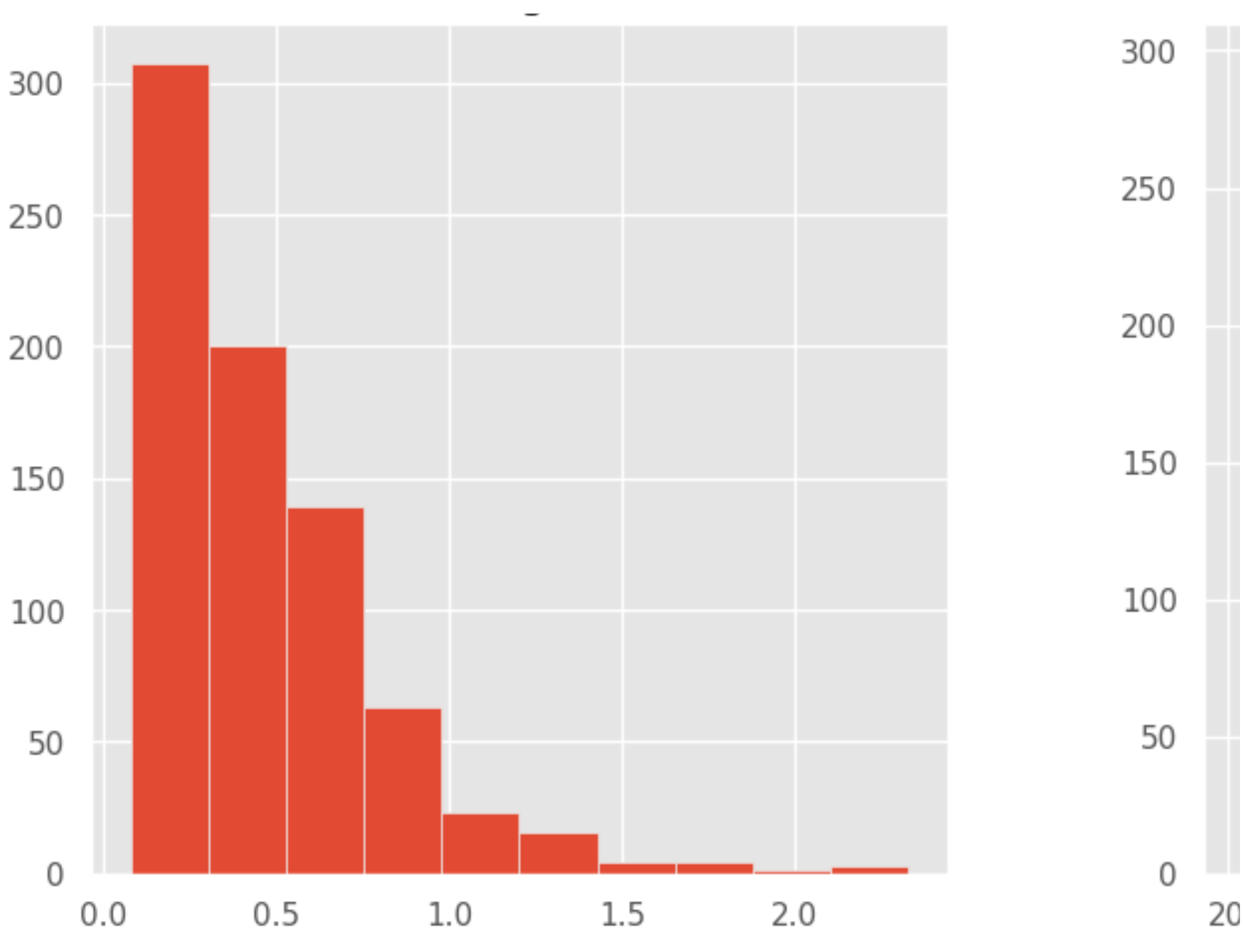

```
df.hist(figsize = (20,20))
```

```
array([[<Axes: title={'center': 'Pregnancies'}>,
        <Axes: title={'center': 'Glucose'}>,
        <Axes: title={'center': 'BloodPressure'}>],
       [<Axes: title={'center': 'SkinThickness'}>,
        <Axes: title={'center': 'Insulin'}>,
        <Axes: title={'center': 'BMI'}>],
       [<Axes: title={'center': 'DiabetesPedigreeFunction'}>,
        <Axes: title={'center': 'Age'}>,
        <Axes: title={'center': 'Outcome'}>]], dtype=object)
```



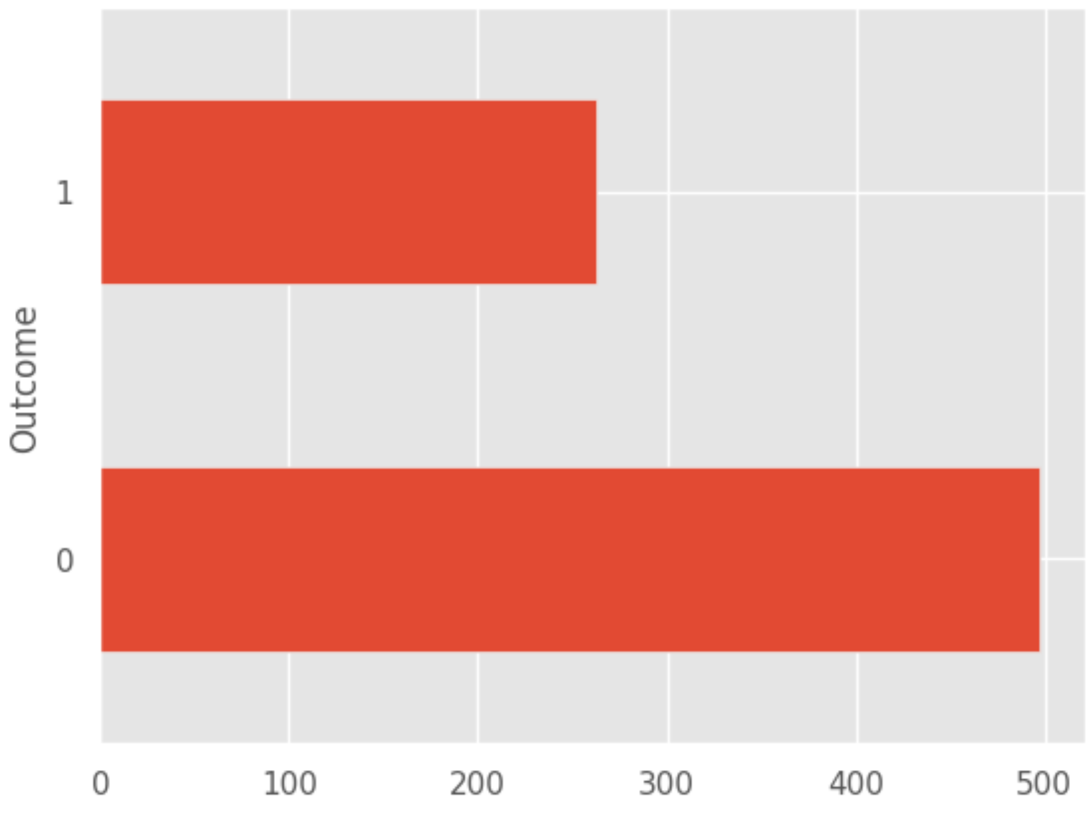


DiabetesPedigreeFunction




```
df['Outcome'].value_counts().plot(kind='barh')
```

```
<Axes: ylabel='Outcome'>
```



```
plt.figure(figsize=(5, 5))
sns.countplot(x='Outcome', hue='Outcome', data=df, palette='deep')
plt.title("Outcome")
plt.xlabel("Has Diabetes")
plt.xticks([0, 1], ('False', 'True'))
plt.ylabel("Count")
plt.show()
```

Has Diabetes

```
df.groupby("Outcome").agg({'Pregnancies':'mean'})
```

Outcome	Pregnancies
0	3.298000
1	4.865672

```
df.groupby("Outcome").agg({'Pregnancies':'max'})
```

Outcome	Pregnancies
0	13
1	17

```
df.groupby("Outcome").agg({'Glucose':'mean'})
```

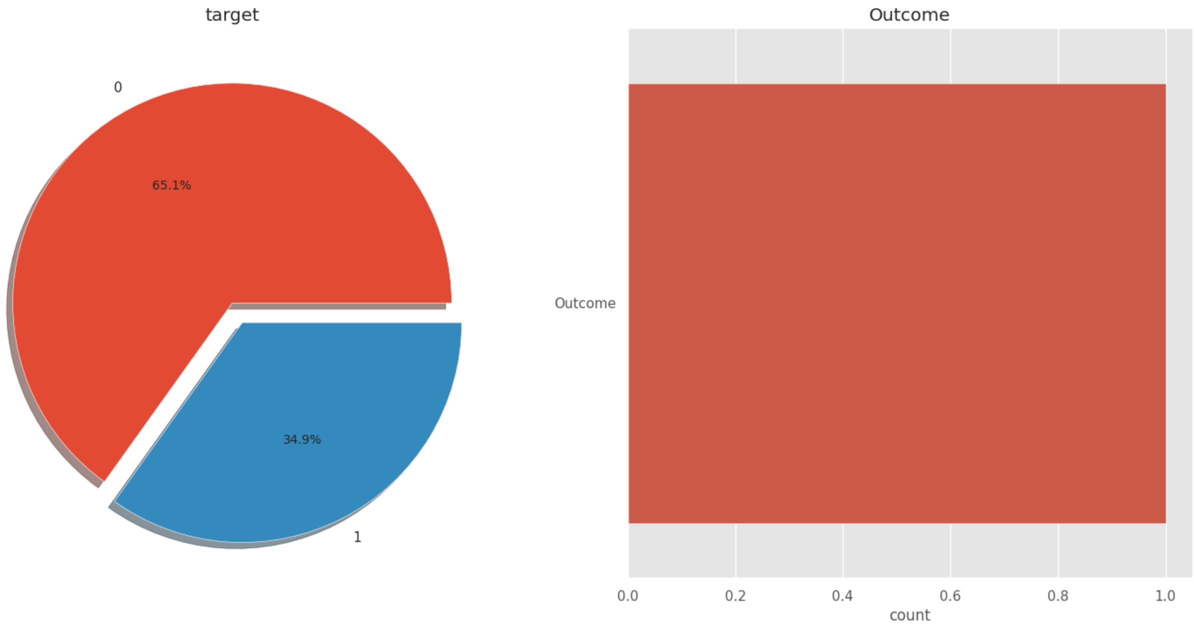
Outcome	Glucose
0	109.980000
1	141.257463

```
df.groupby("Outcome").agg({'Glucose':'max'})
```

Outcome	Glucose
0	197
1	199

```
import numpy as np
import statsmodels.api as sm
sns.set()
plt.style.use("ggplot")
%matplotlib inline
```

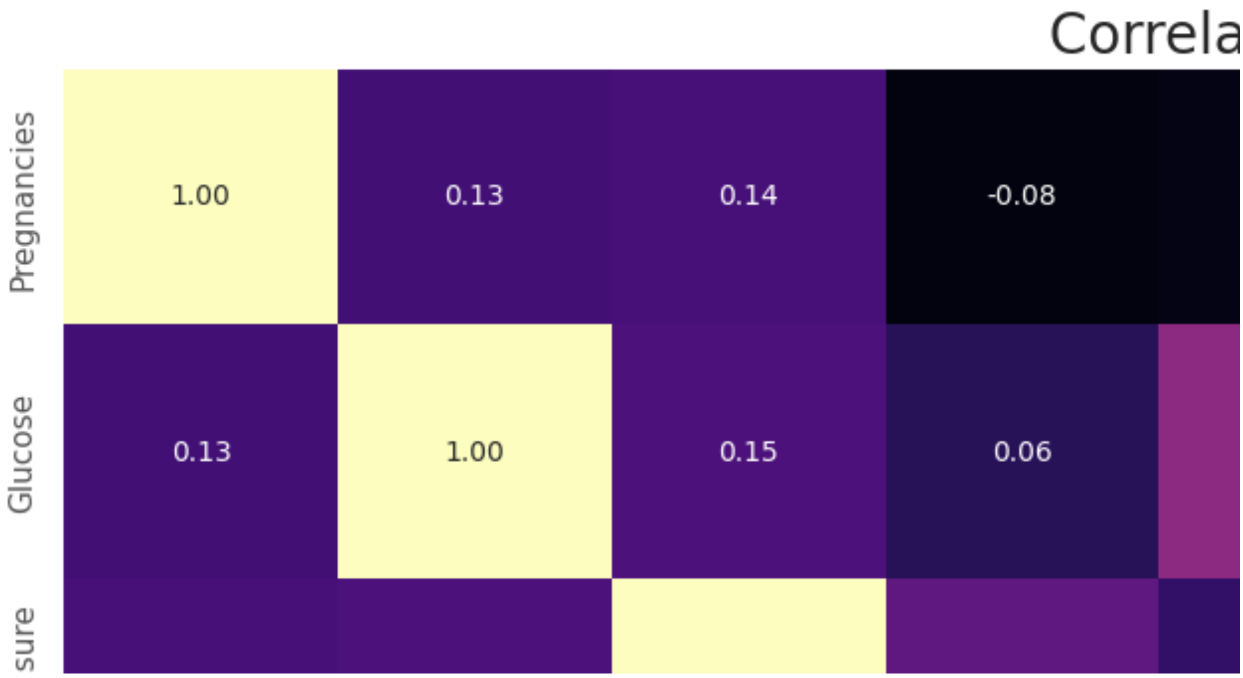
```
f,ax = plt.subplots(1,2, figsize=(18,8))
df['Outcome'].value_counts().plot.pie(explode=[0,0.1],autopct = "%1.1f%%", ax=a
ax[0].set_title('target')
ax[0].set_ylabel('')
sns.countplot('Outcome', ax=ax[1])
ax[1].set_title('Outcome')
plt.show()
```

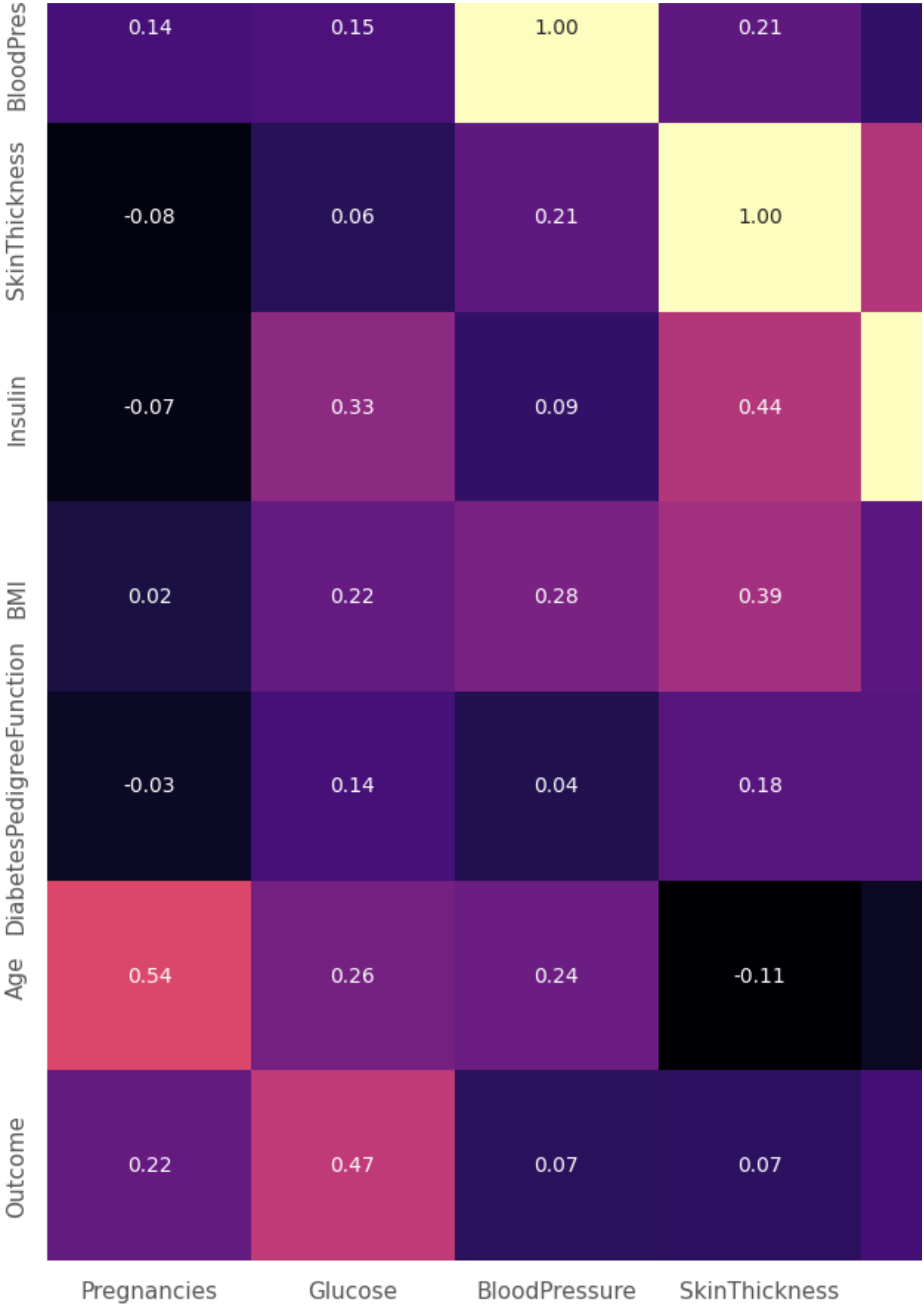


```
df.corr()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness
Pregnancies	1.000000	0.129459	0.141282	-0.081672
Glucose	0.129459	1.000000	0.152590	0.057328
BloodPressure	0.141282	0.152590	1.000000	0.207371
SkinThickness	-0.081672	0.057328	0.207371	1.000000
Insulin	-0.073535	0.331357	0.088933	0.436783
BMI	0.017683	0.221071	0.281805	0.392573
DiabetesPedigreeFunction	-0.033523	0.137337	0.041265	0.183928
Age	0.544341	0.263514	0.239528	-0.113970
Outcome	0.221898	0.466581	0.065068	0.074752

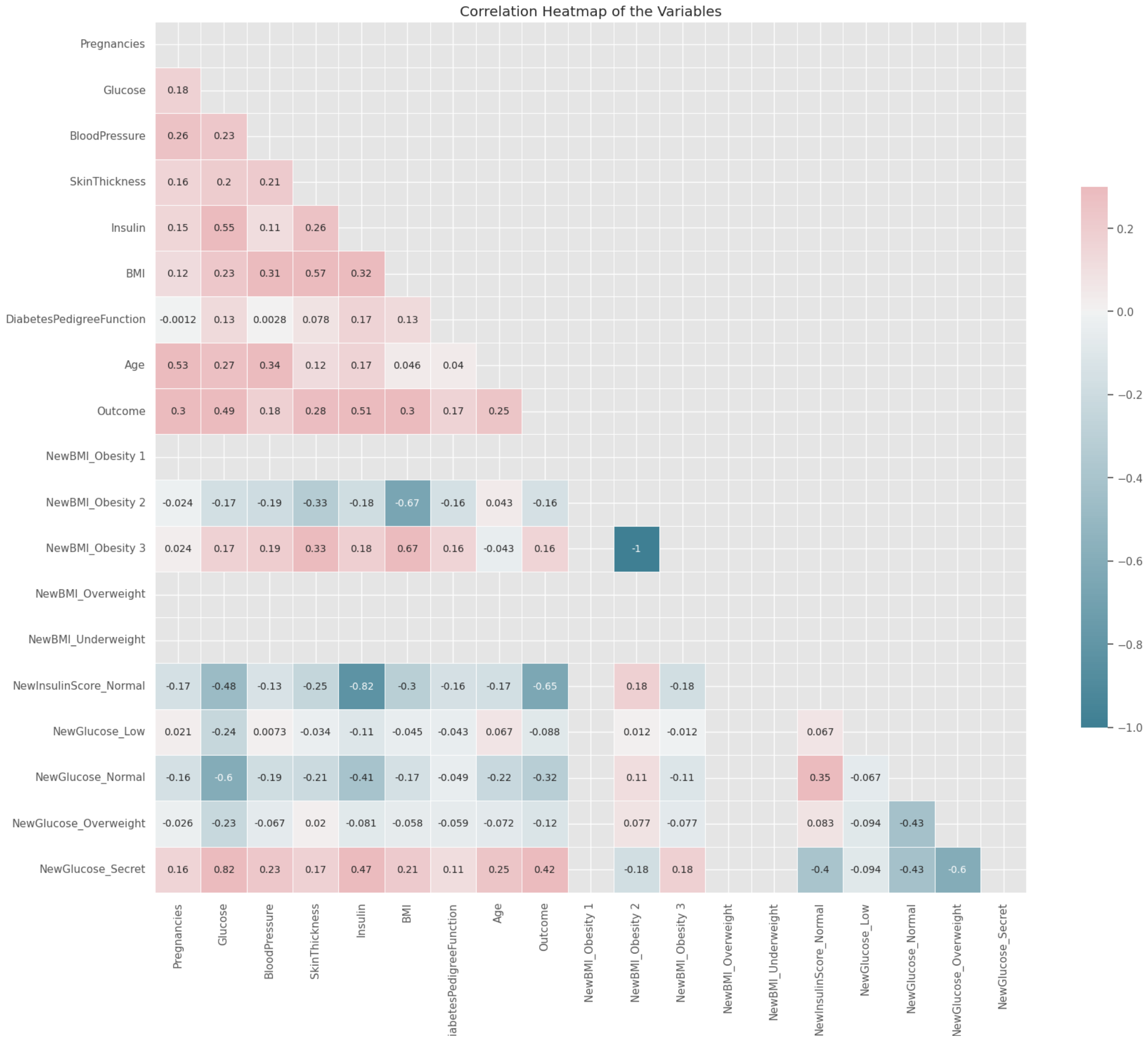
```
f,ax = plt.subplots(figsize=[20,15])
sns.heatmap(df.corr(), annot=True, fmt = '.2f', ax=ax, cmap='magma')
ax.set_title("Correlation Matrix", fontsize=20)
plt.show()
```






```
# Computer correlation matrix
corr = df.corr()
```

```
# Generate a mask for the upper triangle
mask = np.zeros_like(corr)
mask[np.triu_indices_from(mask)] = True

# Set up the matplotlib figure
fig, ax = plt.subplots(figsize=(20, 20))

# Generate a custom diverging colourmap
cmap = sns.diverging_palette(220, 10, as_cmap=True)

# Draw the heatmap with the mask and correct aspect ratio
sns.heatmap(
    corr, mask=mask, cmap=cmap, vmax=.3, center=0,
    square=True, linewidths=.5, cbar_kws={"shrink": .5},
    annot=True
)

ax.set_title('Correlation Heatmap of the Variables')

plt.show()
```



```
df.columns
```

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

```
df[['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age']] = df[['Pregnancies', 'Glucose
       'BMI', 'DiabetesPedigreeFunction', 'Age']].replace(0, np.NaN)
```

```
# Data preprocessing Part
```

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

```
Pregnancies                 111
Glucose                       5
BloodPressure                35
SkinThickness               227
Insulin                     374
BMI                          11
DiabetesPedigreeFunction      0
Age                           0
Outcome                       0
dtype: int64
```

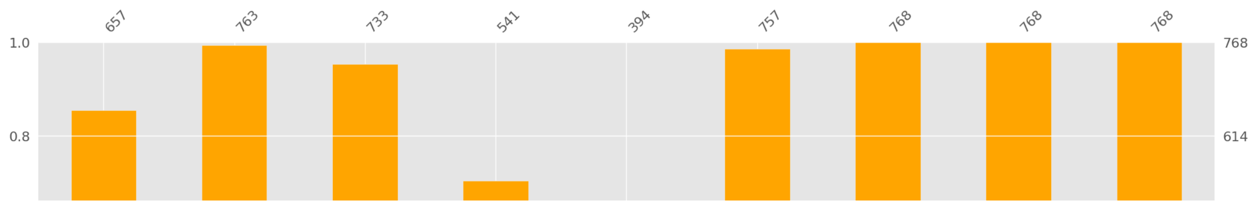
```
df.head()
```

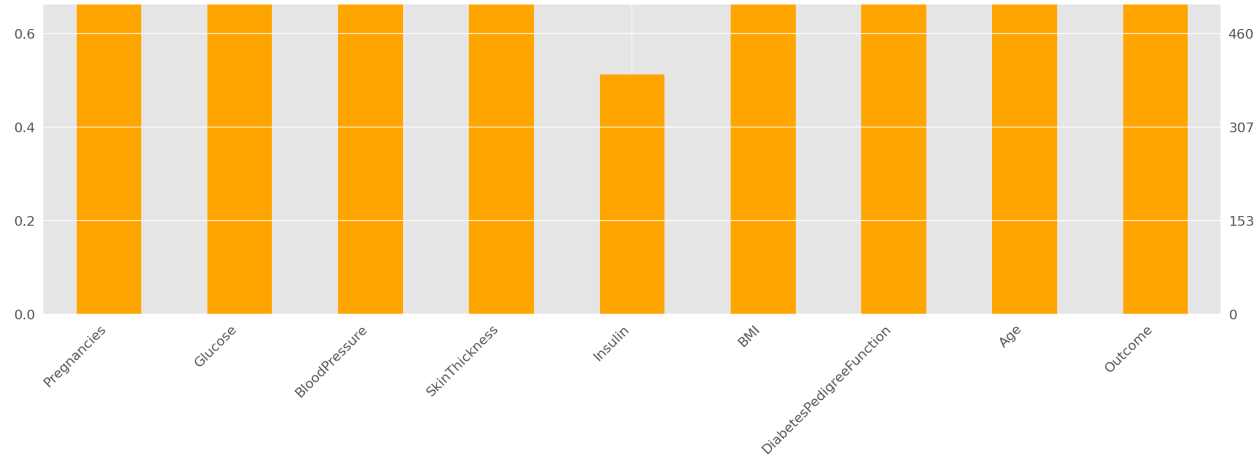
	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Diabete
0	6.0	148.0	72.0	35.0	NaN	33.6	
1	1.0	85.0	66.0	29.0	NaN	26.6	
2	8.0	183.0	64.0	NaN	NaN	23.3	
3	1.0	89.0	66.0	23.0	94.0	28.1	
4	NaN	137.0	40.0	35.0	168.0	43.1	

Next steps: Generate code with df View recommended plots

```
import missingno as msno
msno.bar(df, color="orange")
```

```
<Axes: >
```






```
# Handling Missing values :
def median_target(var):
    temp = df[df[var].notnull()]
    temp = temp[[var, 'Outcome']].groupby(['Outcome'])[[var]].median().reset_ir
    return temp


columns = df.columns
columns = columns.drop("Outcome")
for i in columns:
    median_target(i)
    df.loc[(df['Outcome'] == 0 ) & (df[i].isnull()), i] = median_target(i)[i][0
    df.loc[(df['Outcome'] == 1 ) & (df[i].isnull()), i] = median_target(i)[i][1


df.head()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Diabete:
0	6.0	148.0	72.0	35.0	169.5	33.6	

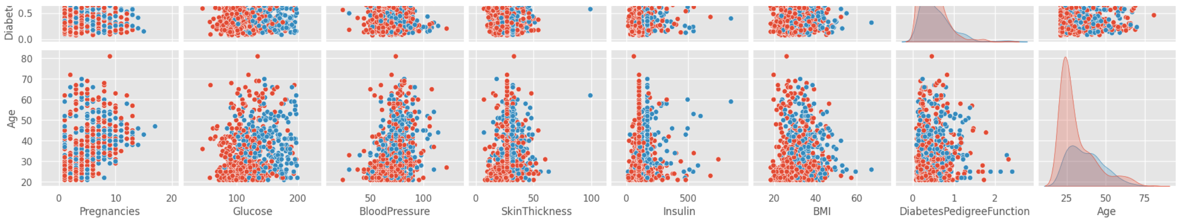
1	1.0	85.0	66.0	29.0	102.5	26.6
2	8.0	183.0	64.0	32.0	169.5	23.3
3	1.0	89.0	66.0	23.0	94.0	28.1
4	5.0	137.0	40.0	35.0	168.0	43.1

Next steps: Generate code with df View recommended plots

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

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

```
# pair plot
p = sns.pairplot(df, hue="Outcome")
```



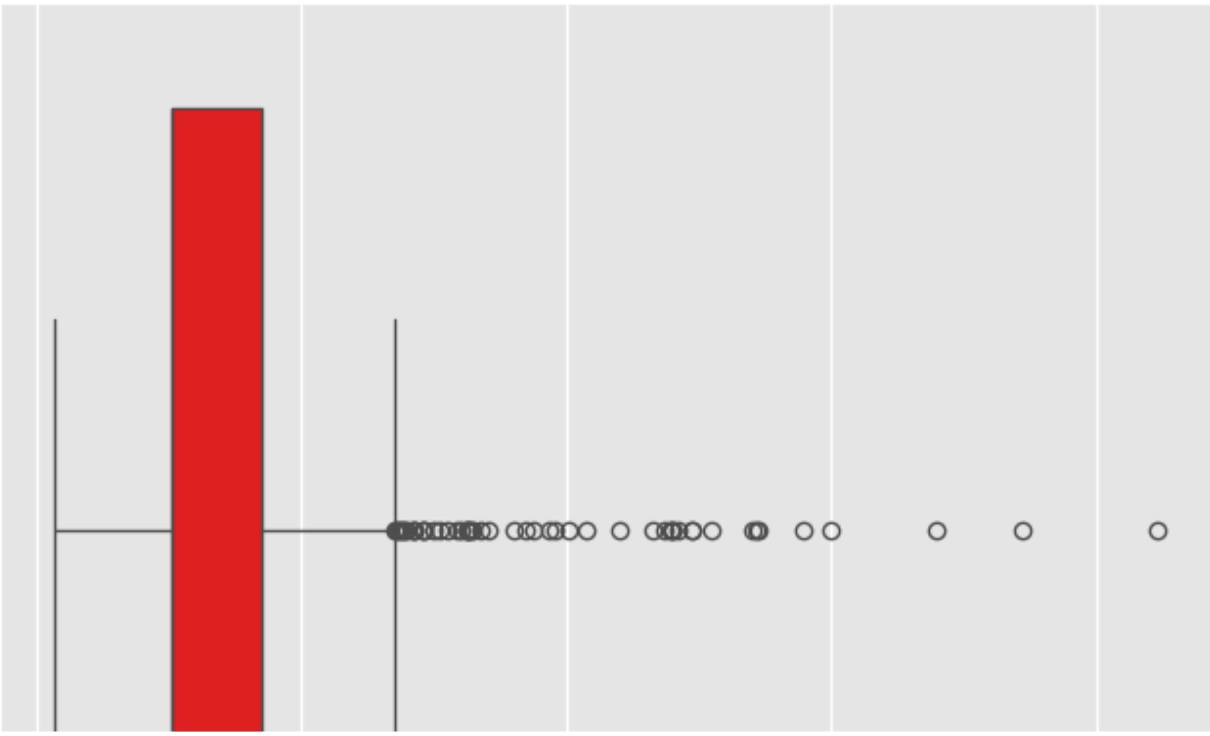

```
import statsmodels.api as sm
# Outlier Detection
# IQR+Q1
# 50%
# 24.65->25%+50%
# 24.65->25%
for feature in df:
    Q1 = df[feature].quantile(0.25)
    Q3 = df[feature].quantile(0.75)
    IQR = Q3-Q1
    lower = Q1-1.5*IQR
```

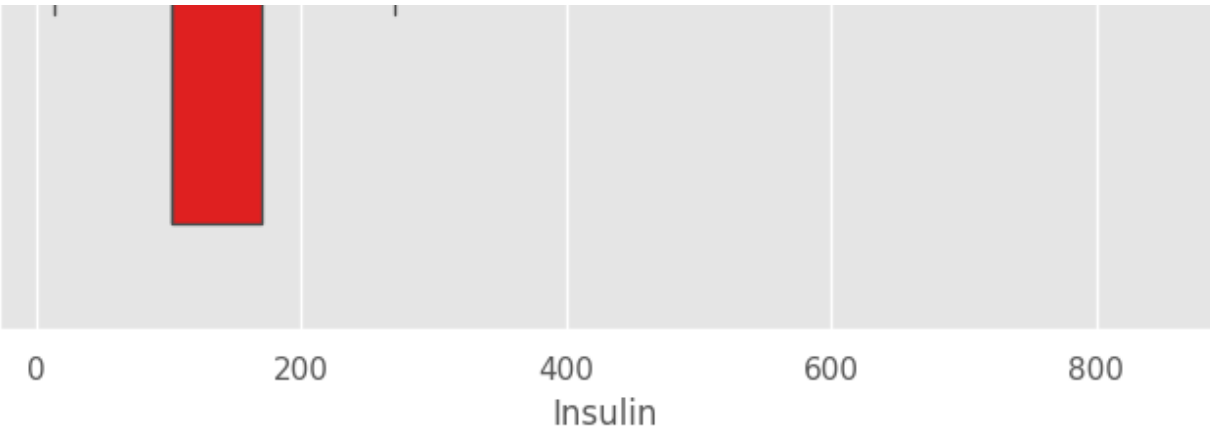
```
    lower = Q1-1.5*IQR
    upper = Q3+1.5*IQR
    if df[(df[feature]>upper)].any(axis=None):
        print(feature, "yes")
    else:
        print(feature, "no")
```

```
Pregnancies yes
Glucose no
BloodPressure yes
SkinThickness yes
Insulin yes
BMI yes
DiabetesPedigreeFunction yes
Age yes
Outcome no
```

```
plt.figure(figsize=(8,7))
sns.boxplot(x= df["Insulin"], color="red")
```

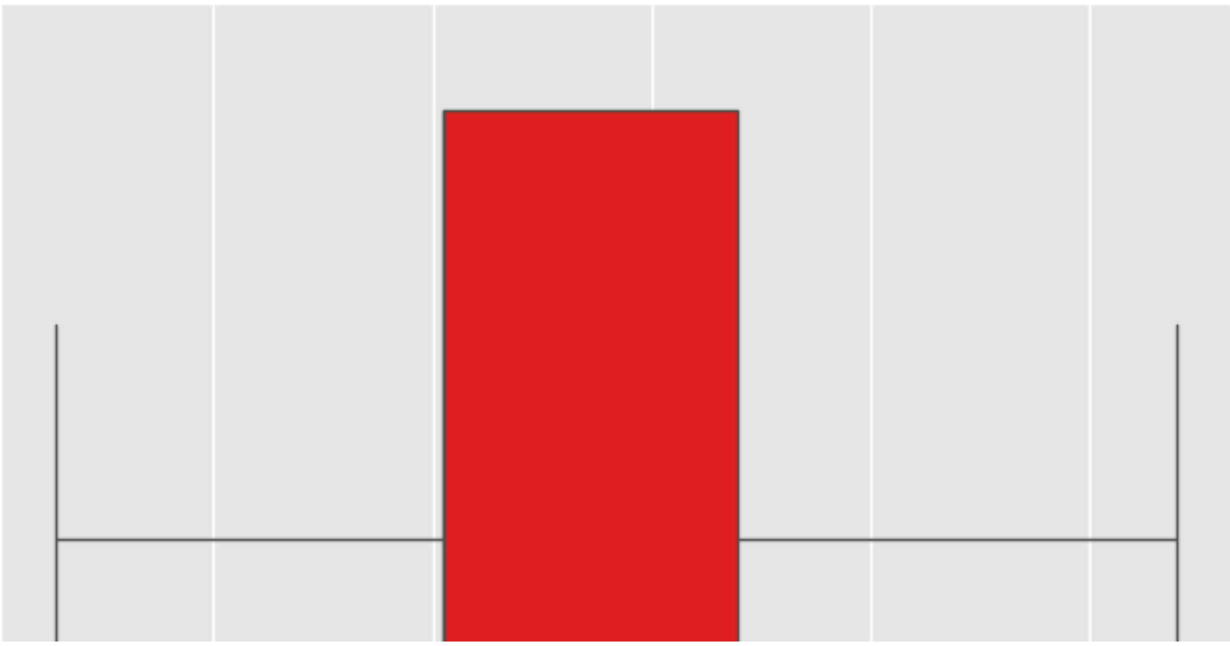
```
<Axes: xlabel='Insulin'>
```

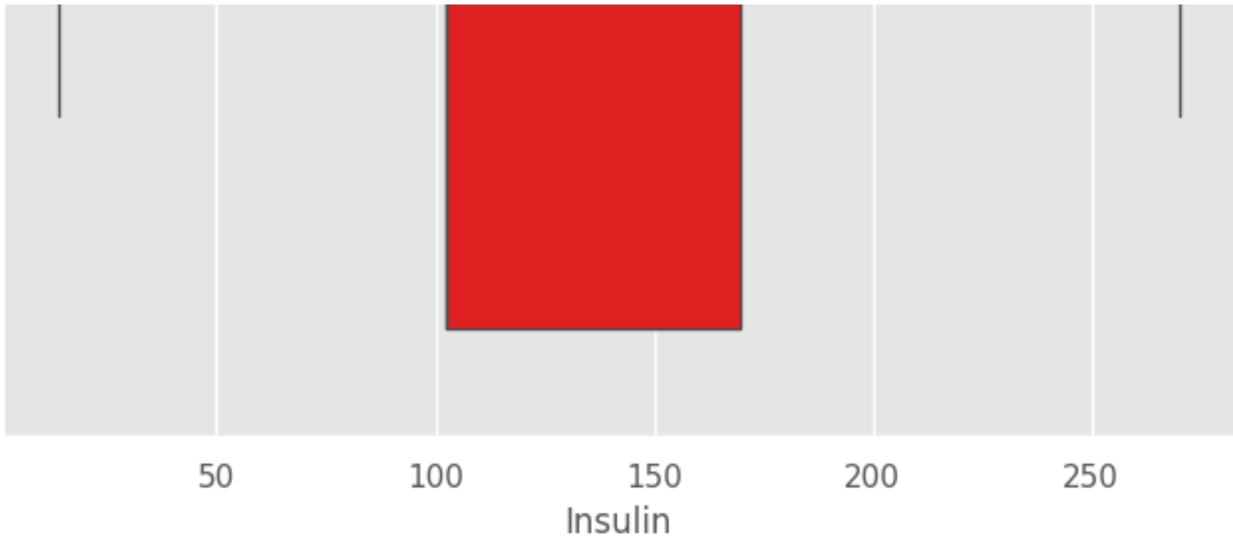




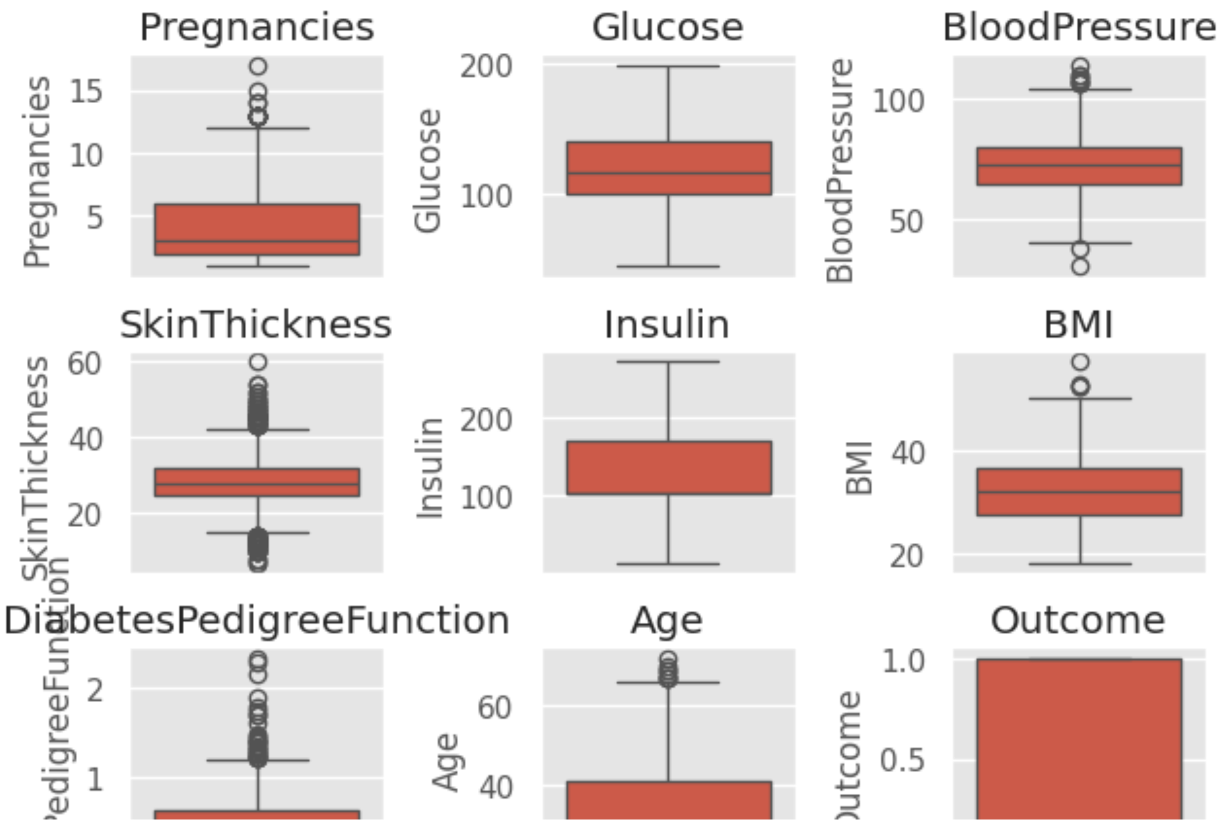

```
Q1 = df.Insulin.quantile(0.25)
Q3 = df.Insulin.quantile(0.75)
IQR = Q3-Q1
lower = Q1-1.5*IQR
upper = Q3+1.5*IQR
df.loc[df['Insulin']>upper, "Insulin"] = upper
plt.figure(figsize=(8,7))
sns.boxplot(x= df["Insulin"], color="red")
```

```
<Axes: xlabel='Insulin'>
```






```
repeat = 1
for col in df.select_dtypes(exclude = 'O').columns[:9]:
    plt.subplot(3,3, repeat)
    sns.boxplot(df[col])
    plt.title(col)
    repeat +=1
plt.tight_layout()
plt.show()
```



```
# local outlier factor
from sklearn.neighbors import LocalOutlierFactor
lof = LocalOutlierFactor(n_neighbors=10)
lof.fit_predict(df)
```

```
array([ 1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1, -1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1, -1,  1,  1,  1,  1, -1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1, -1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1, -1,  1,  1,  1,  1, -1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1, -1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1, -1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1, -1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1, -1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1, -1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1, -1,  1,  1, -1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1, -1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
       -1, -1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1, -1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1, -1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1, -1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
```

```
1,  1,  1,  1,  1,  1, -1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
1, -1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
1,  1, -1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
1,  1,  1,  1,  1, -1,  1,  1,  1, -1,  1,  1,  1,  1,  1,  1,  1,
1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1, -1,  1,  1,
1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
1,  1,  1])
```

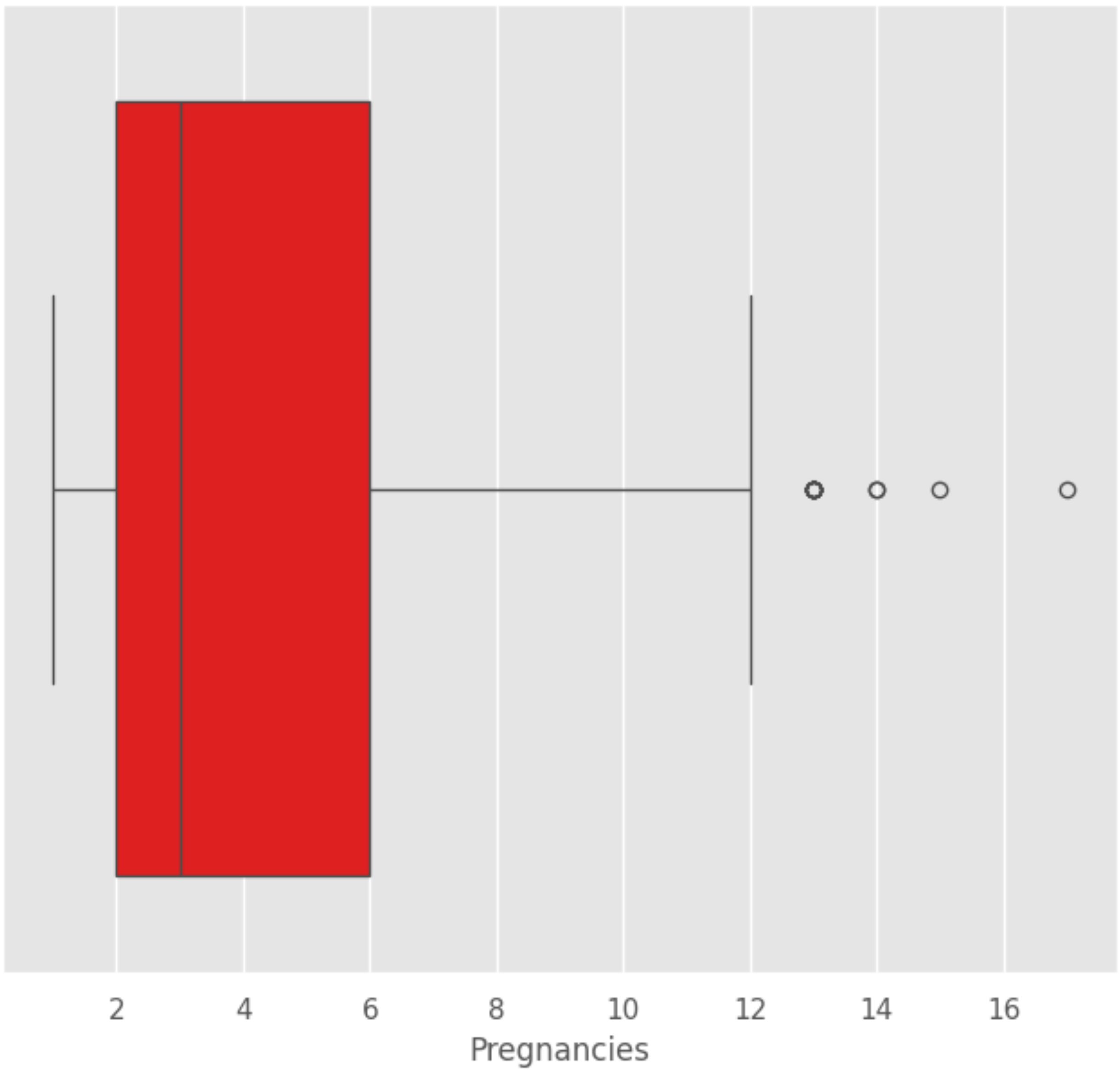
```
df.head()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Diabete:
0	6.0	148.0	72.0	35.0	169.5	33.6	
1	1.0	85.0	66.0	29.0	102.5	26.6	
2	8.0	183.0	64.0	32.0	169.5	23.3	
3	1.0	89.0	66.0	23.0	94.0	28.1	
4	5.0	137.0	40.0	35.0	168.0	43.1	

Next steps: Generate code with df | View recommended plots

```
plt.figure(figsize=(8,7))
sns.boxplot(x= df["Pregnancies"], color="red")
```

```
<Axes: xlabel='Pregnancies'>
```




```
df_scores = lof.negative_outlier_factor_
np.sort(df_scores)[0:20]
```

```
array([-3.06509976, -2.38250393, -2.15557018, -2.11501347, -2.08356175,
```

```
       -1.95386655, -1.83559384, -1.74974237, -1.7330214 , -1.71017168,
       -1.70215105, -1.68722889, -1.64294601, -1.64180205, -1.61181746,
       -1.61067772, -1.60925053, -1.60214364, -1.59998552, -1.58761193])
```

```
thresold = np.sort(df_scores)[7]
thresold
```

```
-1.7497423670960557
```

```
outlier = df_scores > thresold
df = df[outlier]
```

```
df.head()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Diabete
0	6.0	148.0	72.0	35.0	169.5	33.6	
1	1.0	85.0	66.0	29.0	102.5	26.6	
2	8.0	183.0	64.0	32.0	169.5	23.3	
3	1.0	89.0	66.0	23.0	94.0	28.1	
4	5.0	137.0	40.0	35.0	168.0	43.1	

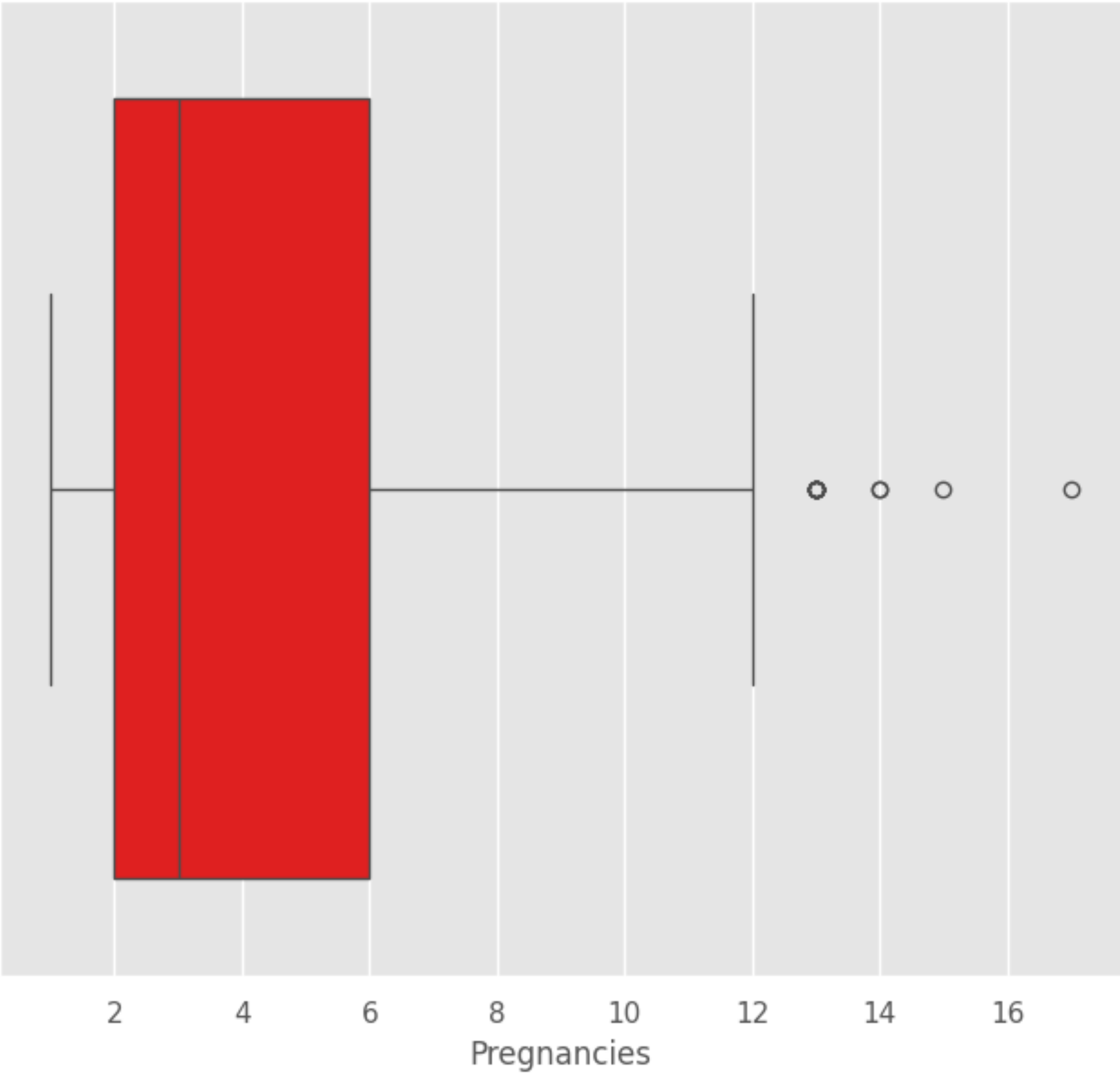
Next steps: Generate code with df View recommended plots

```
df.shape
```

```
(760, 9)
```

```
plt.figure(figsize=(8,7))
sns.boxplot(x= df["Pregnancies"], color="red")
```

```
<Axes: xlabel='Pregnancies'>
```




```
# Feature Enginnering
```

```
# Feature Engineering
NewBMI = pd.Series(["Underweight","Normal", "Overweight","Obesity 1", "Obesity
NewBMI
```

```
0    Underweight
1         Normal
2     Overweight
3      Obesity 1
4      Obesity 2
5      Obesity 3
dtype: category
Categories (6, object): ['Normal', 'Obesity 1', 'Obesity 2', 'Obesity 3',
'Overweight',
                          'Underweight']
```

```
df['NewBMI'] = NewBMI
df.loc[df["BMI"]<18.5, "NewBMI"] = NewBMI[0]
df.loc[(df["BMI"]>18.5) & df["BMI"]<=24.9, "NewBMI"] = NewBMI[1]
df.loc[(df["BMI"]>24.9) & df["BMI"]<=29.9, "NewBMI"] = NewBMI[2]
df.loc[(df["BMI"]>29.9) & df["BMI"]<=34.9, "NewBMI"] = NewBMI[3]
df.loc[(df["BMI"]>34.9) & df["BMI"]<=39.9, "NewBMI"] = NewBMI[4]
df.loc[df["BMI"]>39.9, "NewBMI"] = NewBMI[5]
```

```
df.head()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Diabete
0	6.0	148.0	72.0	35.0	169.5	33.6	
1	1.0	85.0	66.0	29.0	102.5	26.6	
2	8.0	183.0	64.0	32.0	169.5	23.3	

Next steps: Generate code with df View recommended plots

```
# if insulin>=16 & insuline<=166->normal
def set_insuline(row):
    if row["Insulin"]>=16 and row["Insulin"]<=166:
        return "Normal"
    else:
        return "Abnormal"
```

```
df = df.assign(NewInsulinScore=df.apply(set_insuline, axis=1))
df.head()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Diabete

0	6.0	148.0	72.0	35.0	169.5	33.6
1	1.0	85.0	66.0	29.0	102.5	26.6
2	8.0	183.0	64.0	32.0	169.5	23.3

Next steps: Generate code with df View recommended plots

```
# Some intervals were determined according to the glucose variable and these we
NewGlucose = pd.Series(["Low", "Normal", "Overweight", "Secret", "High"], dtype
df["NewGlucose"] = NewGlucose
df.loc[df["Glucose"] <= 70, "NewGlucose"] = NewGlucose[0]
df.loc[(df["Glucose"] > 70) & (df["Glucose"] <= 99), "NewGlucose"] = NewGlucose
df.loc[(df["Glucose"] > 99) & (df["Glucose"] <= 126), "NewGlucose"] = NewGlucos
df.loc[df["Glucose"] > 126 ,"NewGlucose"] = NewGlucose[3]
```

```
df.head()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Diabete
0	6.0	148.0	72.0	35.0	169.5	33.6	
1	1.0	85.0	66.0	29.0	102.5	26.6	
2	8.0	183.0	64.0	32.0	169.5	23.3	

Next steps: Generate code with df View recommended plots

```
# One hot encoding
df = pd.get_dummies(df, columns = ["NewBMI", "NewInsulinScore", "NewGlucose"],
```

```
df.head()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Diabete
0	6.0	148.0	72.0	35.0	169.5	33.6	
1	1.0	85.0	66.0	29.0	102.5	26.6	

2	8.0	183.0	64.0	32.0	169.5	23.3
3	1.0	89.0	66.0	23.0	94.0	28.1
4	5.0	137.0	40.0	35.0	168.0	43.1

Next steps: Generate code with df View recommended plots

```
df.columns
```

```
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness',
'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome',
'NewBMI_Obesity 1',
       'NewBMI_Obesity 2', 'NewBMI_Obesity 3', 'NewBMI_Overweight',
       'NewBMI_Underweight', 'NewInsulinScore_Normal', 'NewGlucose_Low',
       'NewGlucose_Normal', 'NewGlucose_Overweight', 'NewGlucose_Secret'],
      dtype='object')
```

```
categorical_df = df[['NewBMI_Obesity 1',
       'NewBMI_Obesity 2', 'NewBMI_Obesity 3', 'NewBMI_Overweight',
       'NewBMI_Underweight', 'NewInsulinScore_Normal', 'NewGlucose_Low',
       'NewGlucose_Normal', 'NewGlucose_Overweight', 'NewGlucose_Secret']]
```

```
categorical_df.head()
```

	NewBMI_Obesity 1	NewBMI_Obesity 2	NewBMI_Obesity 3	NewBMI_Overweight	NewBMI
0	False	True	False	False	
1	False	True	False	False	
2	False	True	False	False	
3	False	True	False	False	

4	False	False	True	False

Next steps: Generate code with categorical_df | View recommended plots

```
y=df['Outcome']
X=df.drop(['Outcome','NewBMI_Obesity 1',
       'NewBMI_Obesity 2', 'NewBMI_Obesity 3', 'NewBMI_Overweight',
       'NewBMI_Underweight', 'NewInsulinScore_Normal', 'NewGlucose_Low',
       'NewGlucose_Normal', 'NewGlucose_Overweight', 'NewGlucose_Secret'], axis
```

```
cols = X.columns
index = X.index
```

```
X.head()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Diabete:
0	6.0	148.0	72.0	35.0	169.5	33.6	
1	1.0	85.0	66.0	29.0	102.5	26.6	
2	8.0	183.0	64.0	32.0	169.5	23.3	
3	1.0	89.0	66.0	23.0	94.0	28.1	
4	5.0	137.0	40.0	35.0	168.0	43.1	

Next steps: Generate code with X | View recommended plots

```
from sklearn.preprocessing import RobustScaler
transformer = RobustScaler().fit(X)
X=transformer.transform(X)
X=pd.DataFrame(X, columns = cols, index = index)
```

```
X.head()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Dia
0	0.75	0.775	0.000	1.000000	1.000000	0.177778	
1	-0.50	-0.800	-0.375	0.142857	0.000000	-0.600000	
2	1.25	1.650	-0.500	0.571429	1.000000	-0.966667	
3	-0.50	-0.700	-0.375	-0.714286	-0.126866	-0.433333	

4	0.50	0.500	-2.000	1.000000	0.977612	1.233333

Next steps: Generate code with X | View recommended plots

```
X = pd.concat([X, categorical_df], axis=1)
```

```
X.head()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Dia
0	0.75	0.775	0.000	1.000000	1.000000	0.177778	
1	-0.50	-0.800	-0.375	0.142857	0.000000	-0.600000	
2	1.25	1.650	-0.500	0.571429	1.000000	-0.966667	
3	-0.50	-0.700	-0.375	-0.714286	-0.126866	-0.433333	
4	0.50	0.500	-2.000	1.000000	0.977612	1.233333	

5 rows × 28 columns

```
from sklearn.preprocessing import scale, StandardScaler
from sklearn.model_selection import train_test_split, GridSearchCV, cross_val_s
from sklearn.metrics import confusion_matrix, accuracy_score, mean_squared_errc
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.neural_network import MLPClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.model_selection import KFold
```

```
X_train, X_test, y_train , y_test = train_test_split(X,y, test_size=0.2, randon
scaler =StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
log_reg = LogisticRegression()
log_reg.fit(X_train, y_train)
```

```
▾ LogisticRegression
LogisticRegression()
```

```
y_pred = log_reg.predict(X_test)
accuracy_score(y_train, log_reg.predict(X_train))
```

```
0.8470394736842105
```

```
log_reg_acc = accuracy_score(y_test, log_reg.predict(X_test))
confusion_matrix(y_test, y_pred)
```

```
array([[88, 10],
       [ 6, 48]])
```

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

```
              precision    recall  f1-score   support

           0       0.94      0.90      0.92        98
           1       0.83      0.89      0.86        54

    accuracy                           0.89       152
   macro avg       0.88      0.89      0.89       152
weighted avg       0.90      0.89      0.90       152
```

```
#KNN-classifier approach:
knn = KNeighborsClassifier()
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
print(accuracy_score(y_train, knn.predict(X_train)))
knn_acc = accuracy_score(y_test, knn.predict(X_test))
print(accuracy_score(y_test, knn.predict(X_test)))
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
0.8782894736842105
0.8947368421052632
[[88 10]
 [ 6 48]]
              precision    recall  f1-score   support

           0       0.94      0.90      0.92        98
           1       0.83      0.89      0.86        54

    accuracy                           0.89       152
   macro avg       0.88      0.89      0.89       152
weighted avg       0.90      0.89      0.90       152
```

```
# SVM-classifier approach:
svc = SVC(probability=True)
parameter = {
    "gamma":[0.0001, 0.001, 0.01, 0.1],
    'C': [0.01, 0.05,0.5, 0.01, 1, 10, 15, 20]
}
grid_search = GridSearchCV(svc, parameter)
grid_search.fit(X_train, y_train)
```

▸ **GridSearchCV**

▸ **estimator: SVC**

▸ SVC

```
# best_parameter
grid_search.best_params_
```

```
{'C': 1, 'gamma': 0.1}
```

```
grid_search.best_score_
```

```
0.8602086438152012
```

```
svc = SVC(C=10, gamma = 0.01, probability=True)
svc.fit(X_train, y_train)
y_pred = svc.predict(X_test)
print(accuracy_score(y_train, svc.predict(X_train)))
svc_acc = accuracy_score(y_test, svc.predict(X_test))
print(accuracy_score(y_test, svc.predict(X_test)))
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
0.8848684210526315
0.9078947368421053
[[90  8]
 [ 6 48]]
              precision    recall  f1-score   support

           0       0.94      0.92      0.93        98
           1       0.86      0.89      0.87        54

    accuracy                           0.91       152
   macro avg       0.90      0.90      0.90       152
weighted avg       0.91      0.91      0.91       152
```

```
# Decision Tree Approach:
DT = DecisionTreeClassifier()
DT.fit(X_train, y_train)
y_pred = DT.predict(X_test)
print(accuracy_score(y_train, DT.predict(X_train)))

print(accuracy_score(y_test, DT.predict(X_test)))
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
1.0
0.868421052631579
[[85 13]
 [ 7 47]]
              precision    recall  f1-score   support

           0       0.92      0.87      0.89        98
           1       0.78      0.87      0.82        54

    accuracy                           0.87       152
   macro avg       0.85      0.87      0.86       152
weighted avg       0.87      0.87      0.87       152
```

```
# hyperparameter tuning of Decision Theoretic Model:
grid_param = {
    'criterion':['gini','entropy'],
    'max_depth' :  [3,5,7,10],
    'splitter' : ['best','radom'],
    'min_samples_leaf':[1,2,3,5,7],
    'min_samples_split':[1,2,3,5,7],
    'max_features':['auto','sqrt','log2']
}
grid_search_dt = GridSearchCV(DT, grid_param, cv=50, n_jobs=-1, verbose = 1)
grid_search_dt.fit(X_train, y_train)
```

```
Fitting 50 folds for each of 1200 candidates, totalling 60000 fits
```

▾ **GridSearchCV**

```
GridSearchCV(cv=50, estimator=DecisionTreeClassifier(), n_jobs=-1,
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': [3, 5, 7, 10],
                         'max_features': ['auto', 'sqrt', 'log2'],
                         'min_samples_leaf': [1, 2, 3, 5, 7],
                         'min_samples_split': [1, 2, 3, 5, 7],
                         'splitter': ['best', 'radom']},
             verbose=1)
```

▾ **estimator: DecisionTreeClassifier**

```
DecisionTreeClassifier()
```

▾ DecisionTreeClassifier

```
DecisionTreeClassifier()
```

```
grid_search_dt.best_params_
```

```
{'criterion': 'gini',
 'max_depth': 5,
```

```
      'max_features': 'sqrt',
      'min_samples_leaf': 7,
      'min_samples_split': 2,
      'splitter': 'best'}
```

```
grid_search_dt.best_score_
```

```
0.8741025641025643
```

```
DT = grid_search_dt.best_estimator_
y_pred = DT.predict(X_test)
print(accuracy_score(y_train, DT.predict(X_train)))
dt_acc = accuracy_score(y_test, DT.predict(X_test))
print(accuracy_score(y_test, DT.predict(X_test)))
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
0.8601973684210527
0.8355263157894737
[[78 20]
 [ 5 49]]
              precision    recall  f1-score   support

           0       0.94      0.80      0.86        98
           1       0.71      0.91      0.80        54

    accuracy                           0.84       152
   macro avg       0.82      0.85      0.83       152
weighted avg       0.86      0.84      0.84       152
```

```
# Random Forest Classifier Aprroach:
rand_clf = RandomForestClassifier(criterion = 'entropy', max_depth = 15, max_fe
rand_clf.fit(X_train, y_train)
```

```
▾                         RandomForestClassifier
RandomForestClassifier(criterion='entropy', max_depth=15, max_features=0.75
                       min_samples_leaf=2, min_samples_split=3,
                       n_estimators=130)
```

```
y_pred = rand_clf.predict(X_test)
```

```
y_pred = rand_clf.predict(X_test)
print(accuracy_score(y_train, rand_clf.predict(X_train)))
rand_acc = accuracy_score(y_test, rand_clf.predict(X_test))
print(accuracy_score(y_test, rand_clf.predict(X_test)))
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
0.9901315789473685
0.9013157894736842
[[89  9]
 [ 6 48]]
              precision    recall  f1-score   support

           0       0.94      0.91      0.92        98
           1       0.84      0.89      0.86        54

    accuracy                           0.90       152
   macro avg       0.89      0.90      0.89       152
weighted avg       0.90      0.90      0.90       152
```

```
gbc = GradientBoostingClassifier()

parameters = {
    'loss': ['deviance', 'exponential'],
    'learning_rate': [0.001, 0.1, 1, 10],
    'n_estimators': [100, 150, 180, 200]
}

grid_search_gbc = GridSearchCV(gbc, parameters, cv = 10, n_jobs = -1, verbose =
grid_search_gbc.fit(X_train, y_train)
```

```
Fitting 10 folds for each of 32 candidates, totalling 320 fits
```

GridSearchCV

```
GridSearchCV(cv=10, estimator=GradientBoostingClassifier(), n_jobs=-1,
             param_grid={'learning_rate': [0.001, 0.1, 1, 10],
                         'loss': ['deviance', 'exponential'],
                         'n_estimators': [100, 150, 180, 200]},
             verbose=1)
```

estimator: GradientBoostingClassifier

```
GradientBoostingClassifier()
```

```
▾ GradientBoostingClassifier
GradientBoostingClassifier()
```

```
grid_search_gbc.best_params_
```

```
{'learning_rate': 0.1, 'loss': 'exponential', 'n_estimators': 150}
```

```
grid_search_gbc.best_score_
```

```
0.888032786885246
```

```
gbc = GradientBoostingClassifier(learning_rate = 0.1, loss = 'exponential', n_e
gbc.fit(X_train, y_train)
```

```
▾ GradientBoostingClassifier
GradientBoostingClassifier(loss='exponential', n_estimators=150)
```

```
gbc = grid_search_gbc.best_estimator_
y_pred = gbc.predict(X_test)
print(accuracy_score(y_train, gbc.predict(X_train)))
gbc_acc = accuracy_score(y_test, gbc.predict(X_test))
print(accuracy_score(y_test, gbc.predict(X_test)))
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
0.9983552631578947
0.9144736842105263
[[91  7]
 [ 6 48]]
              precision    recall  f1-score   support

           0       0.94      0.93      0.93        98
           1       0.87      0.89      0.88        54

    accuracy                           0.91       152
   macro avg       0.91      0.91      0.91       152
weighted avg       0.91      0.91      0.91       152
```

```
from xgboost import XGBClassifier
xgb = XGBClassifier(objective = 'binary:logistic', learning_rate = 0.01, max_de

xgb.fit(X_train, y_train)
```

```
▾                                XGBClassifier
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=Nor
              enable_categorical=False, eval_metric=None, feature_types=Nor
              gamma=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=0.01, max_bin=Nor
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=10, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None
              multi_strategy=None, n_estimators=180, n_jobs=None,
              num_parallel_tree=None, random_state=None, ...)
```

```
y_pred = xgb.predict(X_test)
print(accuracy_score(y_train, xgb.predict(X_train)))
xgb_acc = accuracy_score(y_test, xgb.predict(X_test))
print(accuracy_score(y_test, xgb.predict(X_test)))
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
0.9736842105263158
0.881578947368421
[[89  9]
 [ 9 45]]
              precision    recall  f1-score   support

           0       0.91      0.91      0.91        98
           1       0.83      0.83      0.83        54

    accuracy                           0.88       152
   macro avg       0.87      0.87      0.87       152
weighted avg       0.88      0.88      0.88       152
```

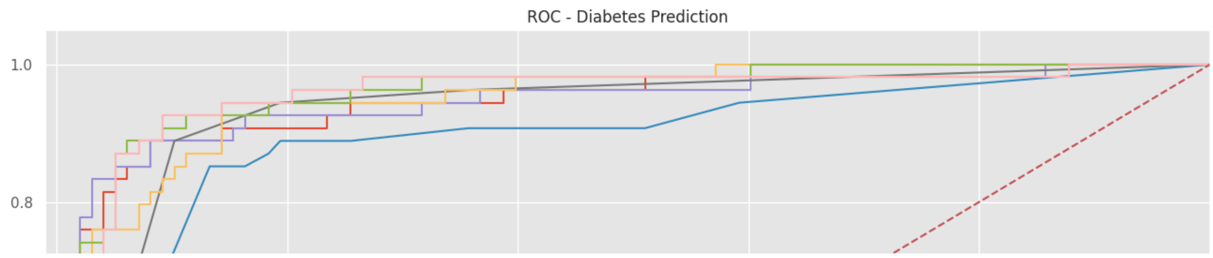
```
# Model Comparison
models = pd.DataFrame({
    'Model': ['Logistic Regression', 'KNN', 'SVM', 'Decision Tree Classifier',
    'Score': [100*round(log_reg_acc,4), 100*round(knn_acc,4), 100*round(svc_acc
              100*round(gbc_acc,4), 100*round(xgb_acc,4)]
})
models.sort_values(by = 'Score', ascending = False)
```

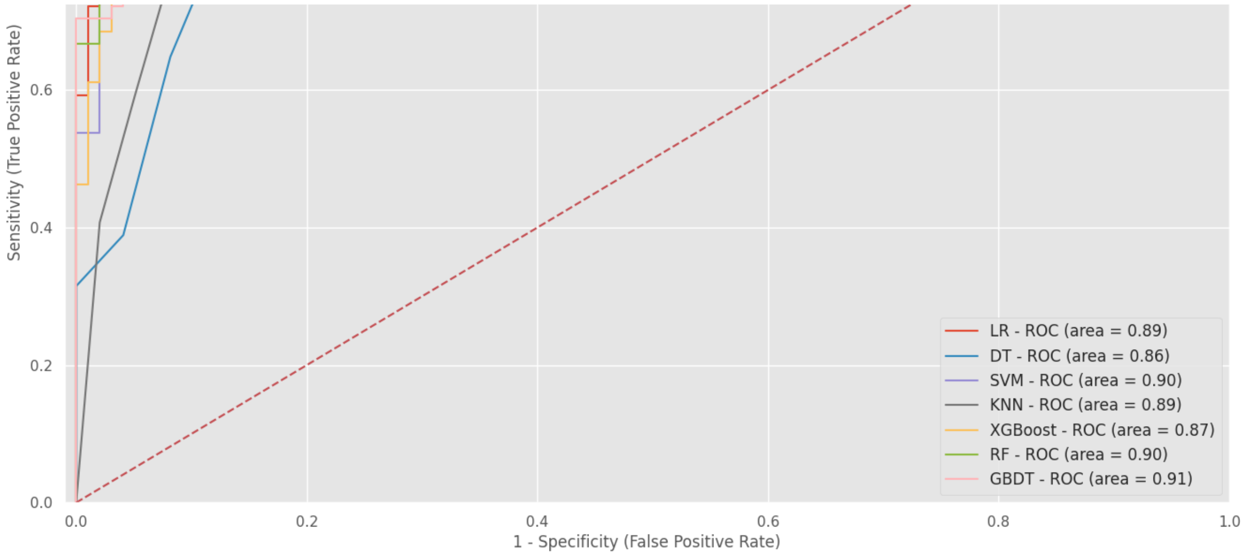
	Model	Score
5	Gradient Boosting Classifier	91.45
2	SVM	90.79
4	Random Forest Classifier	90.13
0	Logistic Regression	89.47
1	KNN	89.47
6	XgBoost	88.16
3	Decision Tree Classifier	83.55

```
import pickle
model = gbc_acc
pickle.dump(model, open("diabetes.pkl",'wb'))
```

```
from sklearn import metrics
plt.figure(figsize=(16,10))
models = [
{
    'label': 'LR',
    'model': log_reg,
},
{
    'label': 'DT',
    'model': DT,
},
{
    'label': 'SVM',
    'model': svc,
},
{
    'label': 'KNN',
    'model': knn,
},
{

    'label': 'XGBoost',
    'model': xgb,
},
{
```

```
{
    'label': 'RF',
    'model': rand_clf,
},
{
    'label': 'GBDT',
    'model': gbc,
}
]
for m in models:
    model = m['model']
    model.fit(X_train, y_train)
    y_pred=model.predict(X_test)
    fpr1, tpr1, thresholds = metrics.roc_curve(y_test, model.predict_proba(X_te
    auc = metrics.roc_auc_score(y_test,model.predict(X_test))
    plt.plot(fpr1, tpr1, label='%s - ROC (area = %0.2f)' % (m['label'], auc))

plt.plot([0, 1], [0, 1],'r--')
plt.xlim([-0.01, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('1 - Specificity (False Positive Rate)', fontsize=12)
plt.ylabel('Sensitivity (True Positive Rate)', fontsize=12)
plt.title('ROC - Diabetes Prediction', fontsize=12)
plt.legend(loc="lower right", fontsize=12)
plt.savefig("roc_diabetes.jpeg", format='jpeg', dpi=400, bbox_inches='tight')
plt.show()
```



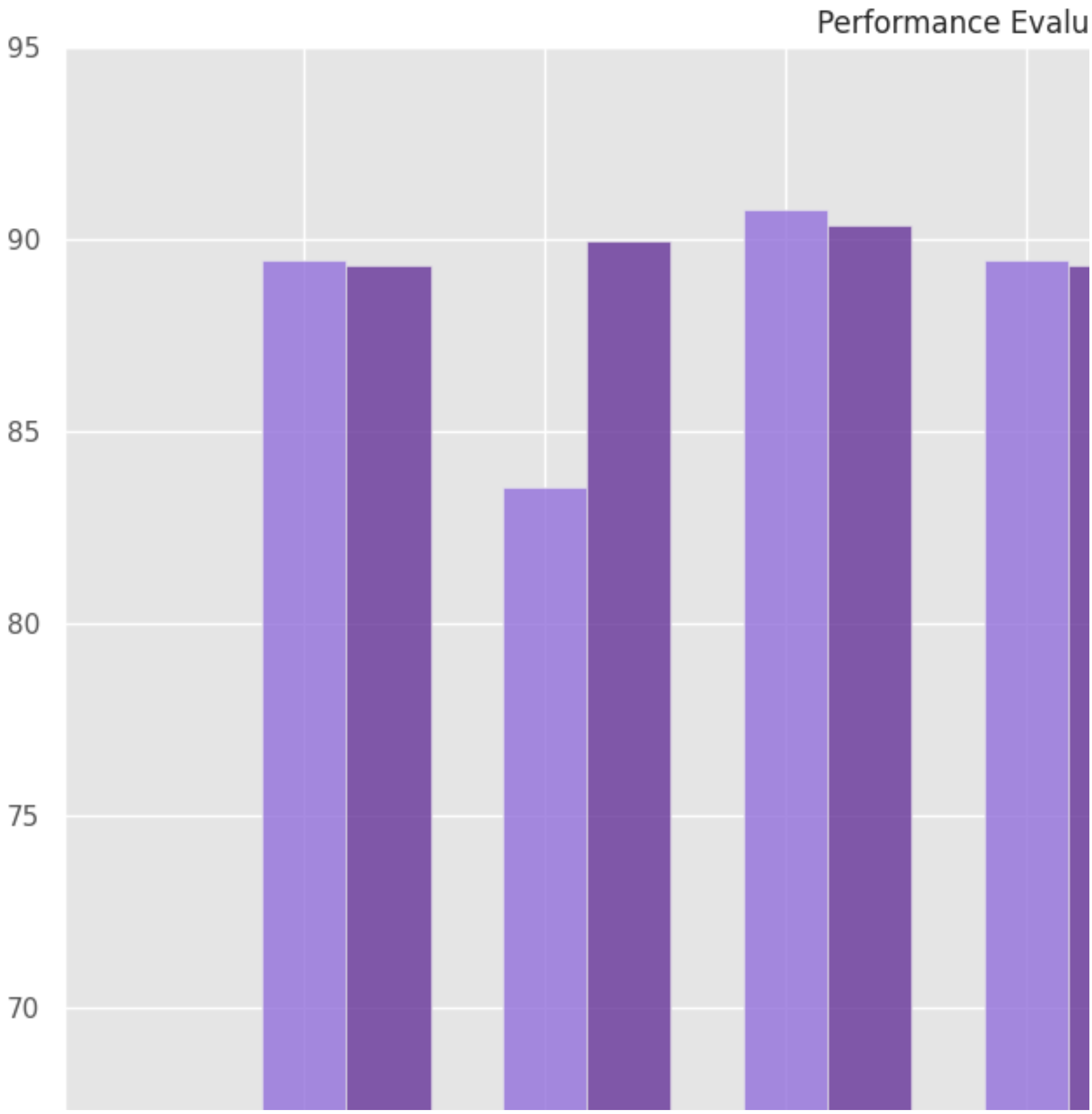


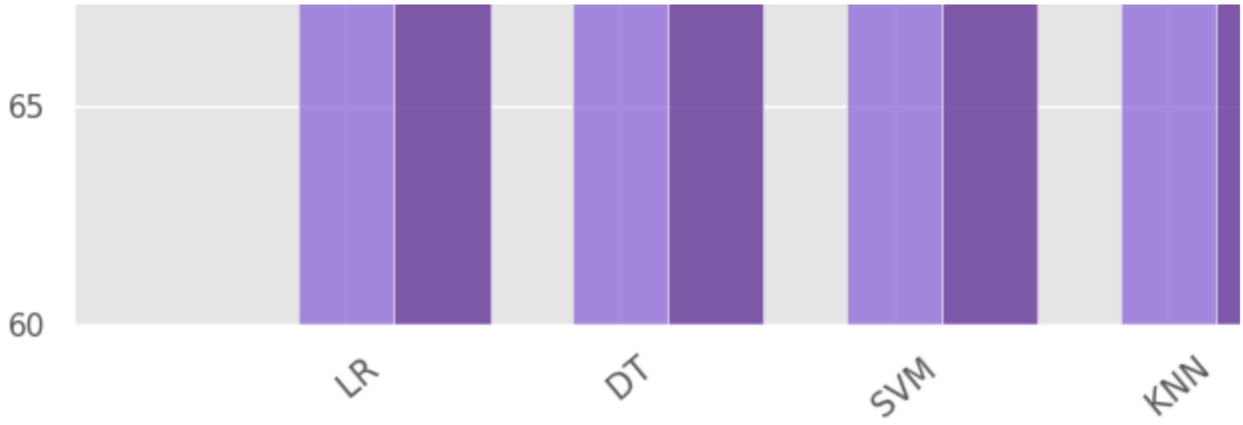

```
from sklearn import metrics

import numpy as np
import matplotlib.pyplot as plt
models = [
{
    'label': 'LR',
    'model': log_reg,
},
{
    'label': 'DT',
    'model': DT,
},
```

```
},
{
    'label': 'SVM',
    'model': svc,
},
{
    'label': 'KNN',
    'model': knn,
},
{
    'label': 'XGBoost',
    'model': xgb,
},
{
    'label': 'RF',
    'model': rand_clf,
},
{
    'label': 'GBDT',
    'model': gbc,
}
]

means_roc = []
means_accuracy = [100*round(log_reg_acc,4), 100*round(dt_acc,4), 100*round(svc_
                  100*round(rand_acc,4), 100*round(gbc_acc,4)]

for m in models:
    model = m['model']
    model.fit(X_train, y_train)
    y_pred=model.predict(X_test)
    fpr1, tpr1, thresholds = metrics.roc_curve(y_test, model.predict_proba(X_te
    auc = metrics.roc_auc_score(y_test,model.predict(X_test))
    auc = 100*round(auc,4)
    means_roc.append(auc)

print(means_accuracy)
print(means_roc)


# data to plot
n_groups = 7
means_accuracy = tuple(means_accuracy)
means_roc = tuple(means_roc)

# create plot
fig, ax = plt.subplots(figsize=(16,10))
index = np.arange(n_groups)
bar_width = 0.35
opacity = 0.8

rects1 = plt.bar(index, means_accuracy, bar_width,
alpha=opacity,
```

```
color='mediumpurple',
label='Accuracy (%)')

rects2 = plt.bar(index + bar_width, means_roc, bar_width,
alpha=opacity,
color='rebeccapurple',
label='ROC (%)')

plt.xlim([-1, 8])
plt.ylim([60, 95])

plt.title('Performance Evaluation - Diabetes Prediction', fontsize=12)
plt.xticks(index, ('   LR', '   DT', '   SVM', '   KNN', 'XGBoost' , '   RF', '
plt.legend(loc="upper right", fontsize=10)
plt.savefig("PE_diabetes.jpeg", format='jpeg', dpi=400, bbox_inches='tight')
plt.show()
```

```
[89.47, 83.55, 90.79, 89.47, 88.16000000000001, 90.13, 91.45]
[89.34, 89.95, 90.36, 89.34, 87.07000000000001, 88.92999999999999, 90.86999
```

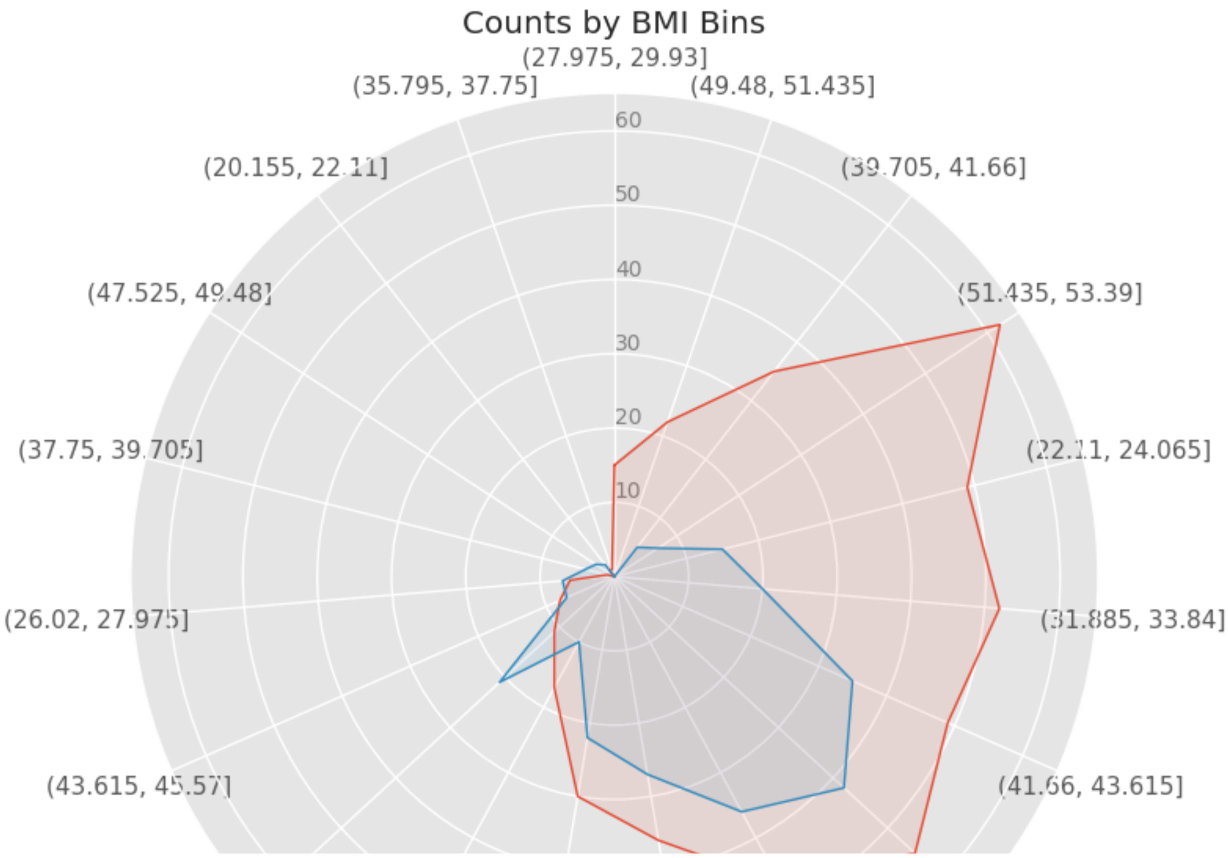


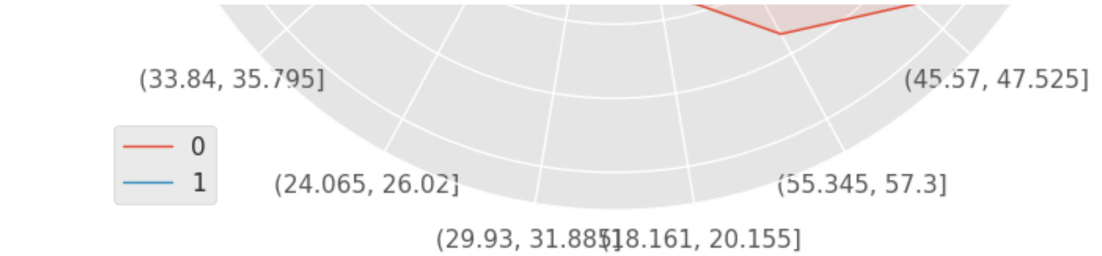



```
# Statistical Learning plus analysis:
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

def plot_radar(df, bins, column,data):
    # SET DATA
    data_counts = pd.crosstab(pd.cut(df[column], bins=bins), df[data])

    # CREATE BACKGROUND
    datas = set(pd.cut(df[column], bins=bins))

    # Angle of each axis in the plot
    angles = [(n / len(datas)) * 2 * np.pi for n in range(len(datas)+1)]

    subplot_kw = {
        'polar': True
    }

    fig, ax = plt.subplots(figsize=(8, 8), subplot_kw=subplot_kw)
    ax.set_theta_offset(np.pi / 2)
```

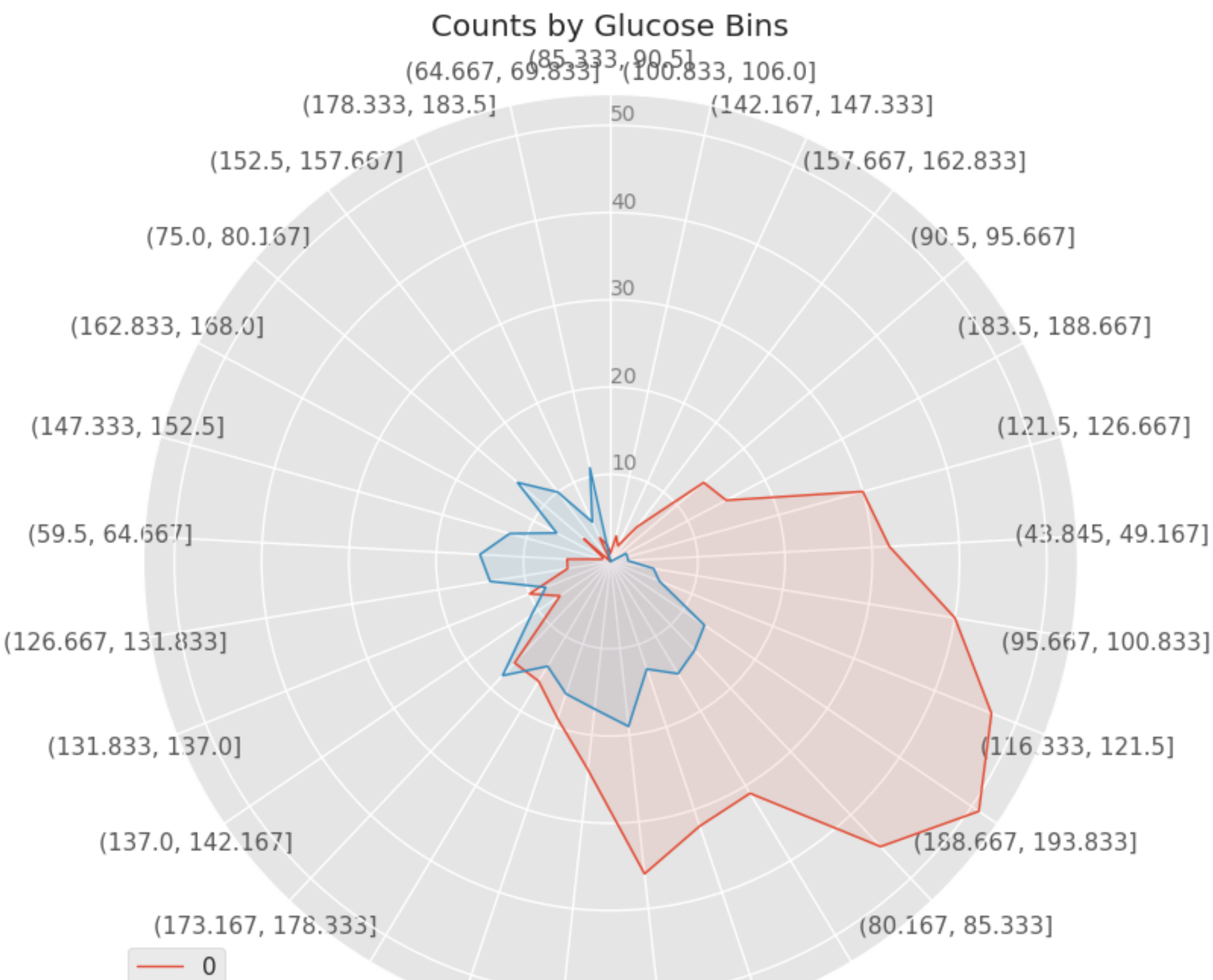
```
    ax.set_theta_direction(-1)
    ax.set_rlabel_position(0)

    plt.xticks(angles[:-1], datas)
    plt.yticks(color="grey", size=10)

    # ADD PLOTS
    for outcome in data_counts.columns:
        counts = data_counts[outcome].tolist()
        counts += counts[:1]  # Properly loops the circle back
        ax.plot(angles, counts, linewidth=1, linestyle='solid', label=outcome)
        ax.fill(angles, counts, alpha=0.1)

    plt.title(f"Counts by {column} Bins")
    plt.legend(loc='upper right', bbox_to_anchor=(0.1, 0.1))

    plt.show()
```

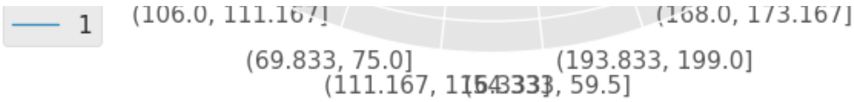
```
plot_radar(df, 20, "BMI","Outcome")
```






```
plot_radar(df, 30, "Glucose","Outcome")
```






```
# Outlier Detection:
outliers_cols = ["Glucose","BloodPressure","Insulin","BMI"]
outliers_df = pd.DataFrame()

for col in outliers_cols:
    stats = df[col].describe()


    IQR = stats['75%'] - stats['25%']
    upper_limit = stats['75%'] + 1.5 * IQR
    lower_limit = stats['25%'] - 1.5 * IQR

    outliers = df[(df[col] > upper_limit) | (df[col] < lower_limit)]

    if  outliers.empty:
        print(f'\n No outlier found in: {col}')
    else:
        print(f'\n -------------- {col} --------------')
        outliers_df = pd.concat([outliers_df, outliers])
    print(f"Number of Outliers: {len(outliers)}")
    print(f'\n IQR: {IQR}')
    print(f'\n Outliers Lower Bound: {lower_limit}')
    print(f'\n Outliers Upper Bound: {upper_limit}')

print(f'Total number of outlier: {len(outliers_df)}')
```

```
 No outlier found in: Glucose
Number of Outliers: 0

 IQR: 40.0

 Outliers Lower Bound: 40.0

 Outliers Upper Bound: 200.0

 -------------- BloodPressure --------------
Number of Outliers: 10

 IQR: 16.0

 Outliers Lower Bound: 40.0

 Outliers Upper Bound: 104.0

 No outlier found in: Insulin
Number of Outliers: 0

 IQR: 67.0

 Outliers Lower Bound: 2.0

 Outliers Upper Bound: 270.0

 -------------- BMI --------------
Number of Outliers: 4

 IQR: 9.0

 Outliers Lower Bound: 14.0

 Outliers Upper Bound: 50.0
Total number of outlier: 14
```

```
# Z-score determine:
all_outliers = pd.DataFrame()
# Compute mean and standard deviation
for col in outliers_cols:
    mean = df[col].mean()
    std_dev = df[col].std()

    # Calculating z-scores
    df['Z_Score'] = (df[col] - mean) / std_dev

    # outliers threshold
    threshold = 2

    # Identify outliers
    outliers = df[abs(df['Z_Score']) > threshold]
    all_outliers = pd.concat([all_outliers,outliers])

    print(len(outliers),col)
print(f'Total number of outlier: {len(all_outliers)}')
```

```
38 Glucose
35 BloodPressure
59 Insulin
29 BMI
Total number of outlier: 161
```

```
df.corr()['Outcome'] #other features correlation to Outcome
```

```
Pregnancies                 0.296356
Glucose                     0.494213
BloodPressure               0.182309
SkinThickness               0.281523
Insulin                     0.509097
BMI                         0.299717
DiabetesPedigreeFunction    0.166545
Age                         0.246939
Outcome                     1.000000
NewBMI_Obesity 1                 NaN
NewBMI_Obesity 2           -0.155184
NewBMI_Obesity 3            0.155184
NewBMI_Overweight                NaN
NewBMI_Underweight               NaN
NewInsulinScore_Normal     -0.650555
NewGlucose_Low             -0.088157
NewGlucose_Normal          -0.317394
NewGlucose_Overweight      -0.119731
NewGlucose_Secret           0.419215
Z_Score                     0.299717
Name: Outcome, dtype: float64
```

```
X = df.iloc[:,:-1].values
y = df.iloc[:,-1].values
```

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X = scaler.fit_transform(X)
X.shape
```

```
(760, 19)
```

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_state
```

```
import tensorflow
from tensorflow import keras
from tensorflow.keras import Sequential
from keras.layers import Dense,Dropout
```

```
model = Sequential()
model.add(Dense(32,activation='relu',input_dim=19))
model.add(Dense(1,activation='sigmoid'))

model.compile(optimizer='Adam',loss='binary_crossentropy',metrics=['accuracy'])


model.fit(X_train,y_train,batch_size=32,epochs=100,validation_data=(X_test,y_te
```

```
19/19 [==============================] - 0s 17ms/step - loss: -178.7066 - a
Epoch 72/100
19/19 [==============================] - 0s 10ms/step - loss: -183.7772 - a
Epoch 73/100
19/19 [==============================] - 0s 17ms/step - loss: -188.6419 - a
Epoch 74/100
19/19 [==============================] - 0s 7ms/step - loss: -193.8483 - ac
Epoch 75/100
19/19 [==============================] - 0s 7ms/step - loss: -199.0373 - ac
Epoch 76/100
19/19 [==============================] - 0s 6ms/step - loss: -204.3369 - ac
Epoch 77/100
19/19 [==============================] - 0s 7ms/step - loss: -209.4307 - ac
Epoch 78/100
19/19 [==============================] - 0s 6ms/step - loss: -214.9252 - ac
Epoch 79/100
19/19 [==============================] - 0s 8ms/step - loss: -220.3127 - ac
Epoch 80/100
19/19 [==============================] - 0s 7ms/step - loss: -225.8326 - ac
Epoch 81/100
19/19 [==============================] - 0s 7ms/step - loss: -231.2605 - ac
Epoch 82/100
19/19 [==============================] - 0s 5ms/step - loss: -236.7471 - ac
Epoch 83/100
19/19 [==============================] - 0s 5ms/step - loss: -242.4043 - ac
Epoch 84/100
19/19 [==============================] - 0s 6ms/step - loss: -248.0512 - ac
Epoch 85/100
19/19 [==============================] - 0s 9ms/step - loss: -253.8454 - ac
Epoch 86/100
19/19 [==============================] - 0s 6ms/step - loss: -259.4611 - ac
Epoch 87/100
19/19 [==============================] - 0s 6ms/step - loss: -265.3672 - ac
Epoch 88/100
19/19 [==============================] - 0s 7ms/step - loss: -271.2856 - ac
Epoch 89/100
19/19 [==============================] - 0s 6ms/step - loss: -277.1169 - ac
Epoch 90/100
19/19 [==============================] - 0s 8ms/step - loss: -283.1459 - ac
Epoch 91/100
19/19 [==============================] - 0s 5ms/step - loss: -289.1490 - ac
Epoch 92/100
19/19 [==============================] - 0s 8ms/step - loss: -295.0878 - ac
```

```
Epoch 93/100
19/19 [==============================] - 0s 6ms/step - loss: -301.5058 - ac
Epoch 94/100
19/19 [==============================] - 0s 7ms/step - loss: -307.5862 - ac
Epoch 95/100
19/19 [==============================] - 0s 8ms/step - loss: -313.8767 - ac
Epoch 96/100
19/19 [==============================] - 0s 6ms/step - loss: -320.1636 - ac
Epoch 97/100
19/19 [==============================] - 0s 8ms/step - loss: -326.5222 - ac
Epoch 98/100
19/19 [==============================] - 0s 7ms/step - loss: -332.7726 - ac
Epoch 99/100
19/19 [==============================] - 0s 15ms/step - loss: -339.4695 - a
Epoch 100/100
19/19 [==============================] - 0s 17ms/step - loss: -345.9737 - a
<keras.src.callbacks.History at 0x7e013a5dd360>
```

```
model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_4 (Dense)             (None, 32)                640

 dense_5 (Dense)             (None, 1)                 33

=================================================================
Total params: 673 (2.63 KB)
Trainable params: 673 (2.63 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
model.fit(X_train,y_train,batch_size=32,epochs=100,initial_epoch=6,validation_c
```

```
Epoch 7/100
19/19 [==============================] - 0s 12ms/step - loss: -359.5589 - a
Epoch 8/100
19/19 [==============================] - 0s 8ms/step - loss: -366.1521 - ac
Epoch 9/100
19/19 [==============================] - 0s 7ms/step - loss: -373.0780 - ac
Epoch 10/100
19/19 [==============================] - 0s 6ms/step - loss: -379.9747 - ac
Epoch 11/100
19/19 [==============================] - 0s 7ms/step - loss: -386.5775 - ac
Epoch 12/100
19/19 [==============================] - 0s 9ms/step - loss: -393.6030 - ac
Epoch 13/100
19/19 [==============================] - 0s 7ms/step - loss: -400.1753 - ac
Epoch 14/100
19/19 [==============================] - 0s 7ms/step - loss: -407.5401 - ac
Epoch 15/100
19/19 [==============================] - 0s 7ms/step - loss: -414.5459 - ac
Epoch 16/100
19/19 [==============================] - 0s 6ms/step - loss: -421.6665 - ac
Epoch 17/100
```

```
Epoch 17/100
19/19 [==============================] - 0s 8ms/step - loss: -428.6198 - ac
Epoch 18/100
19/19 [==============================] - 0s 7ms/step - loss: -435.9584 - ac
Epoch 19/100
19/19 [==============================] - 0s 9ms/step - loss: -443.1318 - ac
Epoch 20/100
19/19 [==============================] - 0s 6ms/step - loss: -450.3473 - ac
Epoch 21/100
19/19 [==============================] - 0s 7ms/step - loss: -457.7197 - ac
Epoch 22/100
19/19 [==============================] - 0s 5ms/step - loss: -465.2077 - ac
Epoch 23/100
19/19 [==============================] - 0s 6ms/step - loss: -472.4452 - ac
Epoch 24/100
19/19 [==============================] - 0s 5ms/step - loss: -479.8652 - ac
Epoch 25/100
19/19 [==============================] - 0s 6ms/step - loss: -487.6178 - ac
Epoch 26/100
19/19 [==============================] - 0s 6ms/step - loss: -495.0153 - ac
Epoch 27/100
19/19 [==============================] - 0s 6ms/step - loss: -502.6337 - ac
Epoch 28/100
19/19 [==============================] - 0s 6ms/step - loss: -510.2431 - ac
Epoch 29/100
19/19 [==============================] - 0s 6ms/step - loss: -517.8948 - ac
Epoch 30/100
19/19 [==============================] - 0s 6ms/step - loss: -525.6293 - ac
Epoch 31/100
19/19 [==============================] - 0s 6ms/step - loss: -533.4838 - ac
Epoch 32/100
19/19 [==============================] - 0s 5ms/step - loss: -541.3406 - ac
Epoch 33/100
19/19 [==============================] - 0s 4ms/step - loss: -549.1597 - ac
Epoch 34/100
19/19 [==============================] - 0s 4ms/step - loss: -556.9739 - ac
Epoch 35/100
19/19 [==============================] - 0s 4ms/step - loss: -565.1715 - ac
Epoch 36/100
```

```
def build_model(hp):

    model = Sequential()

    units = hp.Int('units',min_value = 8,max_value = 128,step = 8)
```

```
    model.add(Dense(units=units,activation='relu',input_dim=19))
    model.add(Dense(1,activation='sigmoid'))

    model.compile(optimizer='rmsprop',loss='binary_crossentropy', metrics=['acc
    return model
```

```
model.fit(X_train,y_train,batch_size=32, epochs=100, initial_epoch=5,validatior
```

```
Epoch 71/100
19/19 [==============================] - 0s 4ms/step - loss: -1965.0654 - a
Epoch 72/100
19/19 [==============================] - 0s 5ms/step - loss: -1977.7638 - a
Epoch 73/100
19/19 [==============================] - 0s 5ms/step - loss: -1991.3127 - a
Epoch 74/100
19/19 [==============================] - 0s 4ms/step - loss: -2004.8585 - a
Epoch 75/100
19/19 [==============================] - 0s 3ms/step - loss: -2017.7714 - a
Epoch 76/100
19/19 [==============================] - 0s 4ms/step - loss: -2031.3949 - a
Epoch 77/100
19/19 [==============================] - 0s 4ms/step - loss: -2044.4111 - a
Epoch 78/100
19/19 [==============================] - 0s 5ms/step - loss: -2058.3169 - a
Epoch 79/100
19/19 [==============================] - 0s 3ms/step - loss: -2071.3398 - a
Epoch 80/100
19/19 [==============================] - 0s 4ms/step - loss: -2085.1768 - a
Epoch 81/100
19/19 [==============================] - 0s 4ms/step - loss: -2098.5354 - a
Epoch 82/100
19/19 [==============================] - 0s 3ms/step - loss: -2112.4033 - a
Epoch 83/100
19/19 [==============================] - 0s 3ms/step - loss: -2125.6433 - a
Epoch 84/100
19/19 [==============================] - 0s 4ms/step - loss: -2139.7600 - a
Epoch 85/100
19/19 [==============================] - 0s 5ms/step - loss: -2152.7939 - a
Epoch 86/100
19/19 [==============================] - 0s 6ms/step - loss: -2166.9785 - a
Epoch 87/100
19/19 [==============================] - 0s 7ms/step - loss: -2180.7422 - a
Epoch 88/100
19/19 [==============================] - 0s 7ms/step - loss: -2194.5342 - a
Epoch 89/100
19/19 [==============================] - 0s 5ms/step - loss: -2208.1719 - a
Epoch 90/100
19/19 [==============================] - 0s 6ms/step - loss: -2222.2021 - a
Epoch 91/100
19/19 [==============================] - 0s 5ms/step - loss: -2236.0034 - a
Epoch 92/100
19/19 [==============================] - 0s 7ms/step - loss: -2249.9890 - a
Epoch 93/100
19/19 [==============================] - 0s 5ms/step - loss: -2264.0325 - a
Epoch 94/100
19/19 [==============================] - 0s 6ms/step - loss: -2277.7388 - a
Epoch 95/100
```

```
Epoch 95/100
19/19 [==============================] - 0s 5ms/step - loss: -2291.8398 - a
Epoch 96/100
19/19 [==============================] - 0s 6ms/step - loss: -2306.1133 - a
Epoch 97/100
19/19 [==============================] - 0s 6ms/step - loss: -2319.8721 - a
Epoch 98/100
19/19 [==============================] - 0s 7ms/step - loss: -2334.0767 - a
Epoch 99/100
19/19 [==============================] - 0s 5ms/step - loss: -2348.3430 - a
Epoch 100/100
19/19 [==============================] - 0s 5ms/step - loss: -2362.2266 - a
```

```
model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_4 (Dense)             (None, 32)                640

 dense_5 (Dense)             (None, 1)                 33

=================================================================
Total params: 673 (2.63 KB)
Trainable params: 673 (2.63 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
def build_model(hp):

    model = Sequential()

    model.add(Dense(64,activation='relu',input_dim=19))

    for i in range(hp.Int('num_layers',min_value=1,max_value=10)):

        model.add(Dense(64,activation='relu'))


    model.add(Dense(1,activation='sigmoid'))

    model.compile(optimizer='rmsprop', loss='binary_crossentropy',metrics=['acc

    return model
model.fit(X_train,y_train,epochs=100,initial_epoch=6,validation_data=(X_test,y_
```

```
19/19 [==============================] - 0s 12ms/step - loss: -5061.8696 -
Epoch 72/100
19/19 [==============================] - 0s 12ms/step - loss: -5081.4297 -
Epoch 73/100
19/19 [==============================] - 0s 11ms/step - loss: -5101.0430 -
Epoch 74/100
19/19 [==============================] - 0s 9ms/step - loss: -5121.1108 - a
Epoch 75/100
19/19 [==============================] - 0s 13ms/step - loss: -5141.2559 -
```

```
19/19 [==============================] - 0s 13ms/step - loss: -5141.2339 -
Epoch 76/100
19/19 [==============================] - 0s 10ms/step - loss: -5160.4673 -
Epoch 77/100
19/19 [==============================] - 0s 7ms/step - loss: -5180.8267 - a
Epoch 78/100
19/19 [==============================] - 0s 8ms/step - loss: -5200.9785 - a
Epoch 79/100
19/19 [==============================] - 0s 9ms/step - loss: -5220.7808 - a
Epoch 80/100
19/19 [==============================] - 0s 8ms/step - loss: -5241.3086 - a
Epoch 81/100
19/19 [==============================] - 0s 11ms/step - loss: -5261.0015 -
Epoch 82/100
19/19 [==============================] - 0s 9ms/step - loss: -5281.1011 - a
Epoch 83/100
19/19 [==============================] - 0s 7ms/step - loss: -5301.3477 - a
Epoch 84/100
19/19 [==============================] - 0s 11ms/step - loss: -5321.8037 -
Epoch 85/100
19/19 [==============================] - 0s 7ms/step - loss: -5341.5508 - a
Epoch 86/100
19/19 [==============================] - 0s 10ms/step - loss: -5361.9570 -
Epoch 87/100
19/19 [==============================] - 0s 11ms/step - loss: -5382.6289 -
Epoch 88/100
19/19 [==============================] - 0s 11ms/step - loss: -5402.7012 -
Epoch 89/100
19/19 [==============================] - 0s 16ms/step - loss: -5423.1416 -
Epoch 90/100
19/19 [==============================] - 0s 11ms/step - loss: -5443.2788 -
Epoch 91/100
19/19 [==============================] - 0s 7ms/step - loss: -5463.9888 - a
Epoch 92/100
19/19 [==============================] - 0s 7ms/step - loss: -5484.6450 - a
Epoch 93/100
19/19 [==============================] - 0s 8ms/step - loss: -5504.8940 - a
Epoch 94/100
19/19 [==============================] - 0s 7ms/step - loss: -5525.3643 - a
Epoch 95/100
19/19 [==============================] - 0s 8ms/step - loss: -5545.8008 - a
Epoch 96/100
19/19 [==============================] - 0s 8ms/step - loss: -5566.6001 - a
Epoch 97/100
19/19 [==============================] - 0s 13ms/step - loss: -5587.5200 -
Epoch 98/100
19/19 [==============================] - 0s 11ms/step - loss: -5607.8042 -
Epoch 99/100
19/19 [==============================] - 0s 7ms/step - loss: -5628.7319 - a
Epoch 100/100
19/19 [==============================] - 0s 7ms/step - loss: -5649.7192 - a
<keras.src.callbacks.History at 0x7e013a399840>
```

```
model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
```

```
=================================================================
 dense_4 (Dense)             (None, 32)                640

 dense_5 (Dense)             (None, 1)                 33

=================================================================
Total params: 673 (2.63 KB)
Trainable params: 673 (2.63 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
def build_model(hp):

    model = Sequential()
    counter = 0

    for i in range(hp.Int('num_layers',min_value = 1,max_value = 10)):
        if counter == 0:
            model.add(
                Dense(
                    hp.Int('units' + str(i),min_value = 8,max_value = 128,step
                    activation= hp.Choice('activation' + str(i), values=['relu'
                    input_dim=8
                    )
                )
            model.add(Dropout(hp.Choice('dropout'+ str(i), values=[0.1,0.2,0.3,
        else:
            model.add(
                Dense(
                    hp.Int('units' + str(i),min_value = 8,max_value = 128,step
                    activation= hp.Choice('activation' + str(i), values=['relu'
                    )

                )
            model.add(Dropout(hp.Choice('dropout'+ str(i), values=[0.1,0.2,0.3,
        counter += 1

    model.add(Dense(1,activation='sigmoid'))


    model.compile(optimizer=hp.Choice('optimizer', values=['rmsprop','adam','sg
                                  loss='binary_crossentropy',
                                  metrics=['accuracy'])
    return model
model.fit(X_train,y_train,epochs=200, initial_epoch=5,validation_data=(X_test,y
```

```
Epoch 171/200
19/19 [==============================] - 0s 6ms/step - loss: -9557.0674 - a
Epoch 172/200
19/19 [==============================] - 0s 16ms/step - loss: -9583.4990 -
Epoch 173/200
19/19 [==============================] - 0s 8ms/step - loss: -9609.4092 - a
Epoch 174/200
19/19 [==============================] - 0s 6ms/step - loss: -9636.5801 - a
Epoch 175/200
```

```
19/19 [==============================] - 0s 8ms/step - loss: -9662.4141 - a
Epoch 176/200
19/19 [==============================] - 0s 11ms/step - loss: -9689.2002 -
Epoch 177/200
19/19 [==============================] - 0s 6ms/step - loss: -9716.1211 - a
Epoch 178/200
19/19 [==============================] - 0s 6ms/step - loss: -9742.0820 - a
Epoch 179/200
19/19 [==============================] - 0s 8ms/step - loss: -9768.8848 - a
Epoch 180/200
19/19 [==============================] - 0s 8ms/step - loss: -9795.3506 - a
Epoch 181/200
19/19 [==============================] - 0s 11ms/step - loss: -9822.0537 -
Epoch 182/200
19/19 [==============================] - 0s 7ms/step - loss: -9849.6113 - a
Epoch 183/200
19/19 [==============================] - 0s 8ms/step - loss: -9875.5967 - a
Epoch 184/200
19/19 [==============================] - 0s 6ms/step - loss: -9902.2588 - a
Epoch 185/200
19/19 [==============================] - 0s 6ms/step - loss: -9929.0010 - a
Epoch 186/200
19/19 [==============================] - 0s 6ms/step - loss: -9956.2715 - a
Epoch 187/200
19/19 [==============================] - 0s 7ms/step - loss: -9982.9941 - a
Epoch 188/200
19/19 [==============================] - 0s 8ms/step - loss: -10009.8203 -
Epoch 189/200
19/19 [==============================] - 0s 7ms/step - loss: -10037.3955 -
Epoch 190/200
19/19 [==============================] - 0s 7ms/step - loss: -10064.0557 -
Epoch 191/200
19/19 [==============================] - 0s 9ms/step - loss: -10091.4014 -
Epoch 192/200
19/19 [==============================] - 0s 7ms/step - loss: -10117.2627 -
Epoch 193/200
19/19 [==============================] - 0s 22ms/step - loss: -10145.4756 -
Epoch 194/200
19/19 [==============================] - 0s 13ms/step - loss: -10172.5273 -
Epoch 195/200
19/19 [==============================] - 0s 12ms/step - loss: -10200.1123 -
Epoch 196/200
19/19 [==============================] - 0s 12ms/step - loss: -10226.5576 -
Epoch 197/200
19/19 [==============================] - 0s 12ms/step - loss: -10253.8857 -
Epoch 198/200
19/19 [==============================] - 0s 24ms/step - loss: -10281.0830 -
Epoch 199/200
19/19 [==============================] - 1s 33ms/step - loss: -10308.6787 -
Epoch 200/200
19/19 [==============================] - 1s 40ms/step - loss: -10336.1689 -
```

```
model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
```

```
 dense_4 (Dense)             (None, 32)                640

 dense_5 (Dense)             (None, 1)                 33

=================================================================
Total params: 673 (2.63 KB)
Trainable params: 673 (2.63 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```