```
import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

import numpy as np
import pandas as pd
import seaborn as sns
import statsmodels.api as sm
import matplotlib.pyplot as plt
from pandas.plotting import scatter_matrix
from sklearn import metrics
from sklearn.preprocessing import scale, StandardScaler
from sklearn.model_selection import train_test_split, GridSearchCV, cross_val_score
from sklearn.metrics import confusion_matrix, accuracy_score, mean_squared_error, r2_score, roc_auc_score, roc_curve, classification_rep
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
import warnings
warnings.simplefilter(action='ignore')
sns.set()
plt.style.use("ggplot")
%matplotlib inline
```

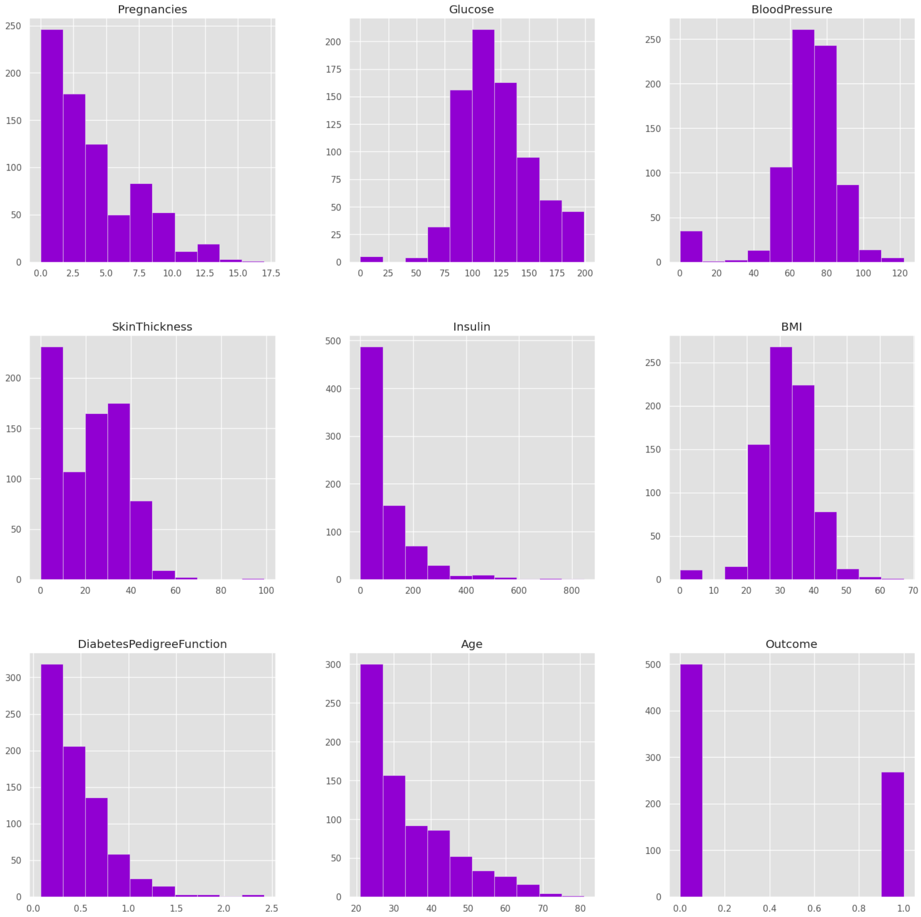
```
/kaggle/input/diabetes-data-set/diabetes.csv
```

```
df = pd.read_csv("/kaggle/input/diabetes-data-set/diabetes.csv")
```

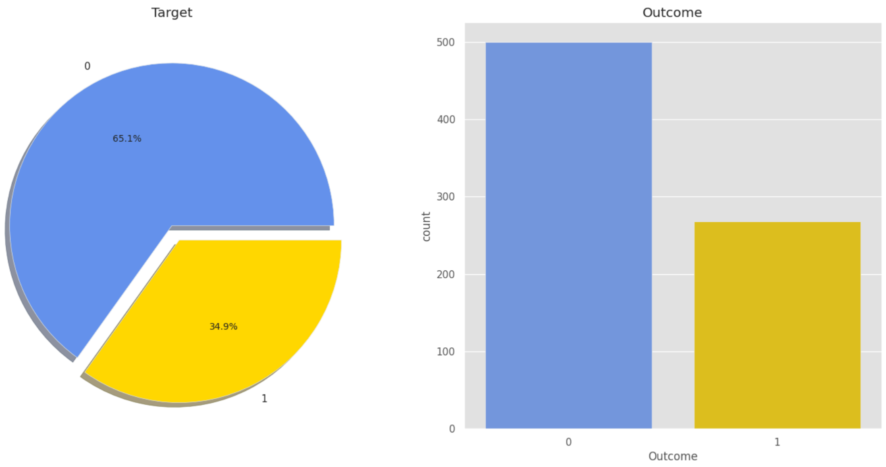
```
df.head()
```

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

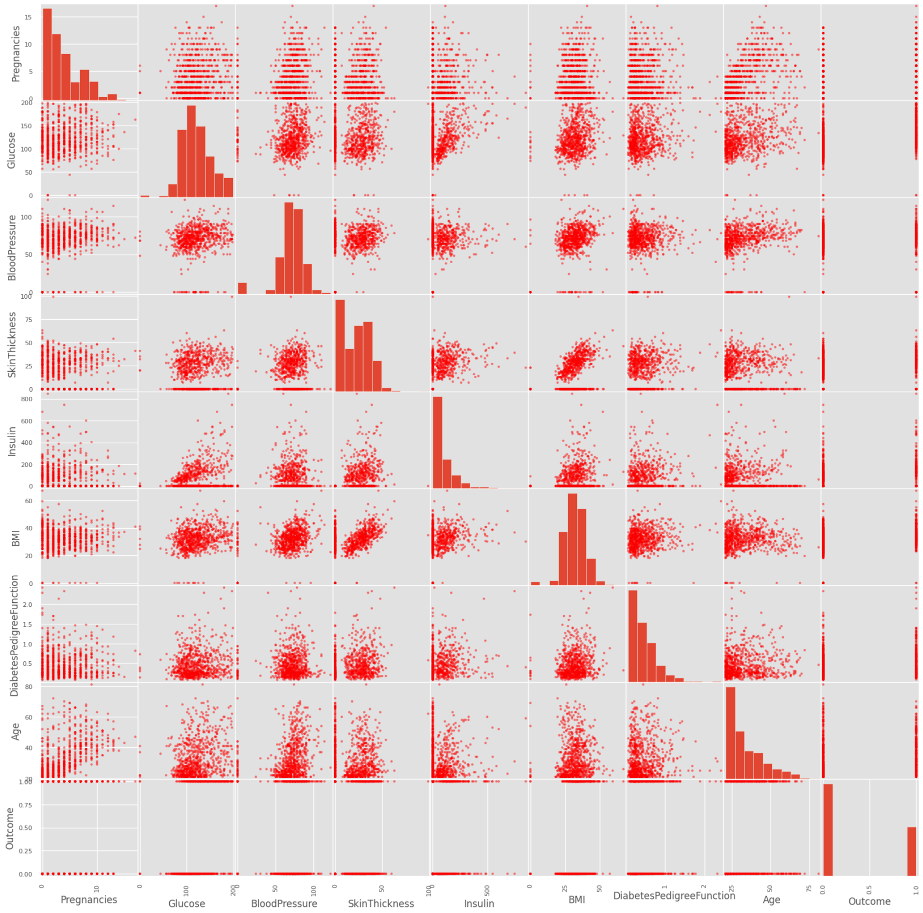
```
p = df.hist(figsize=(20,20), color = 'darkviolet')
```



```
custom_colors = ["#6495ED", "#FFD700"]
f, ax = plt.subplots(1, 2, figsize=(18, 8))
df['Outcome'].value_counts().plot.pie(explode=[0, 0.1], autopct="%1.1f%%", ax=ax[0], shadow=True, colors=custom_colors)
ax[0].set_title('Target')
ax[0].set_ylabel('')
sns.countplot(x='Outcome', data=df, ax=ax[1], palette=custom_colors)
ax[1].set_title('Outcome')
plt.show()
```

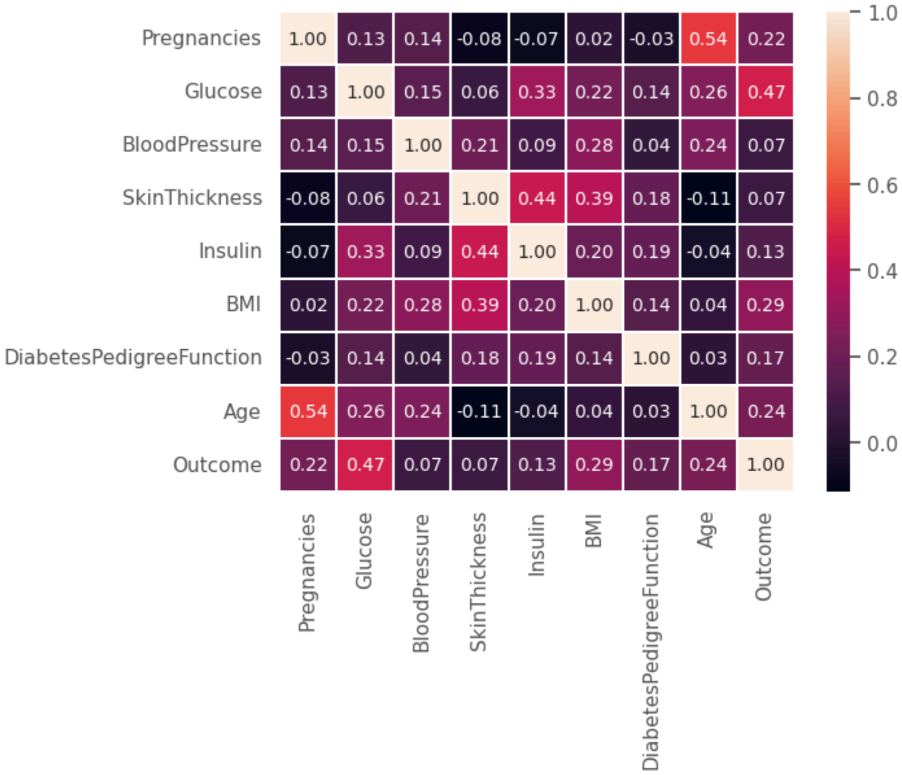


```
p = scatter_matrix(df, figsize=(20, 20), c='red', diagonal='hist')
```

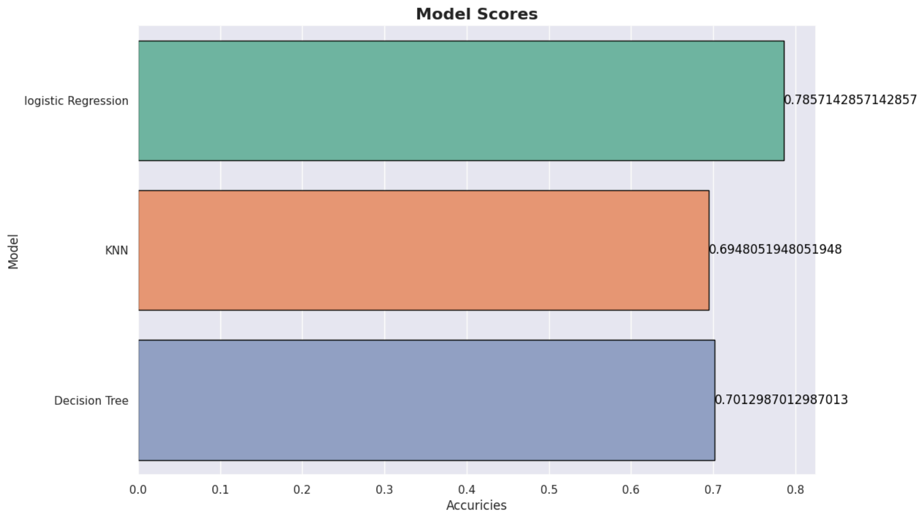


```
sns.heatmap(df.corr(),annot=True,fmt='0.2f',linewidths=0.3)
```

```
<Axes: >
```



```
x=df.drop("Outcome",axis =1).values
y = df['Outcome'].values
trained_x , tested_x , trained_y ,tested_y = train_test_split( x , y , test_size = 0.2,random_state = 42)
scaler = StandardScaler()
trained_scaled_x = scaler.fit_transform(trained_x)
tested_scaled_x = scaler.fit_transform(tested_x)
models = {"logistic Regression":LogisticRegression() , "KNN":KNeighborsClassifier(),"Decision Tree":DecisionTreeClassifier()}
res = {}
for name , model in models.items():
    model.fit(trained_scaled_x,trained_y)
    res[name] = model.score(tested_scaled_x,tested_y)
colors = sns.color_palette("Set2")
sns.set(style='darkgrid')
plt.figure(figsize=(12, 8))
ax=sns.barplot(x=list(res.values()), y=list(res.keys()), palette=colors, edgecolor='black')
for index, value in enumerate(res.values()):
    ax.text(value, index, str(value), color='black',  ha="left", va="center")
plt.title("Model Scores", fontsize=16, fontweight='bold')
plt.xlabel("Accuricies", fontsize=12)
plt.ylabel("Model", fontsize=12)
plt.xticks(fontsize=11)
plt.yticks(fontsize=11)
plt.show()
```



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

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

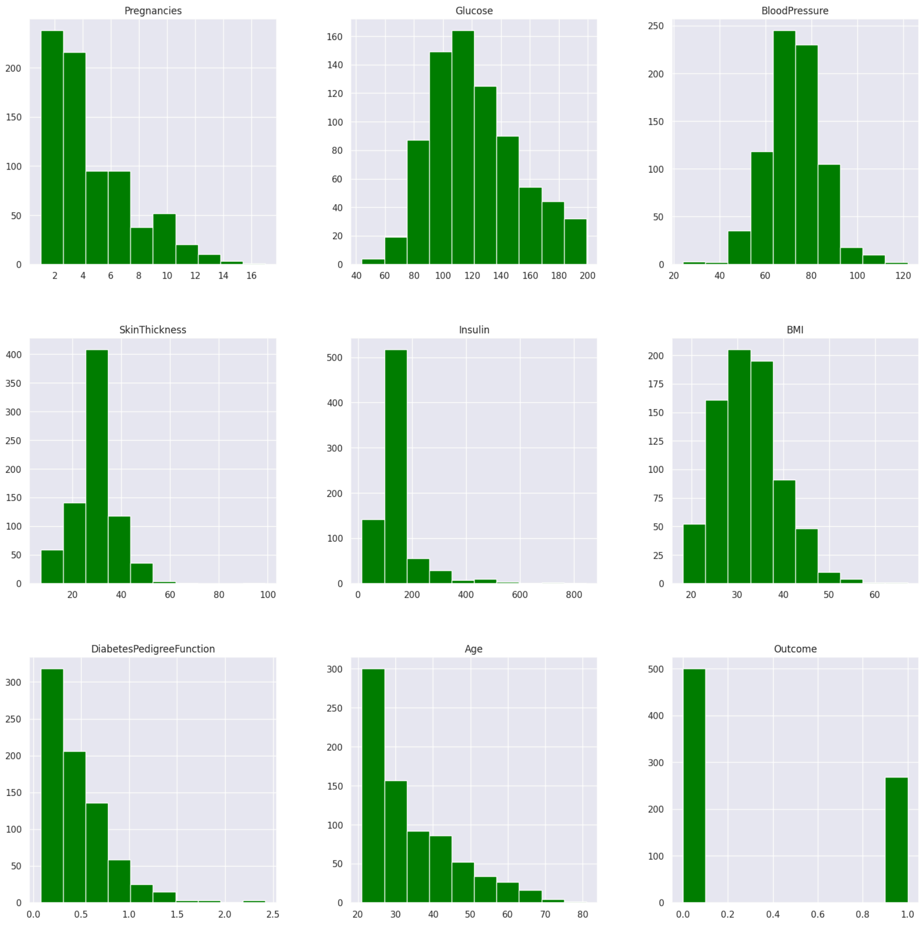
```
    Pregnancies                 111
    Glucose                       5
    BloodPressure                35
    SkinThickness               227
    Insulin                     374
    BMI                          11
    DiabetesPedigreeFunction      0
    Age                           0
    Outcome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

```
def median_target(var):
    temp = df[df[var].notnull()]
    temp = temp[[var, 'Outcome']].groupby(['Outcome'])[[var]].median().reset_index()
    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]
```

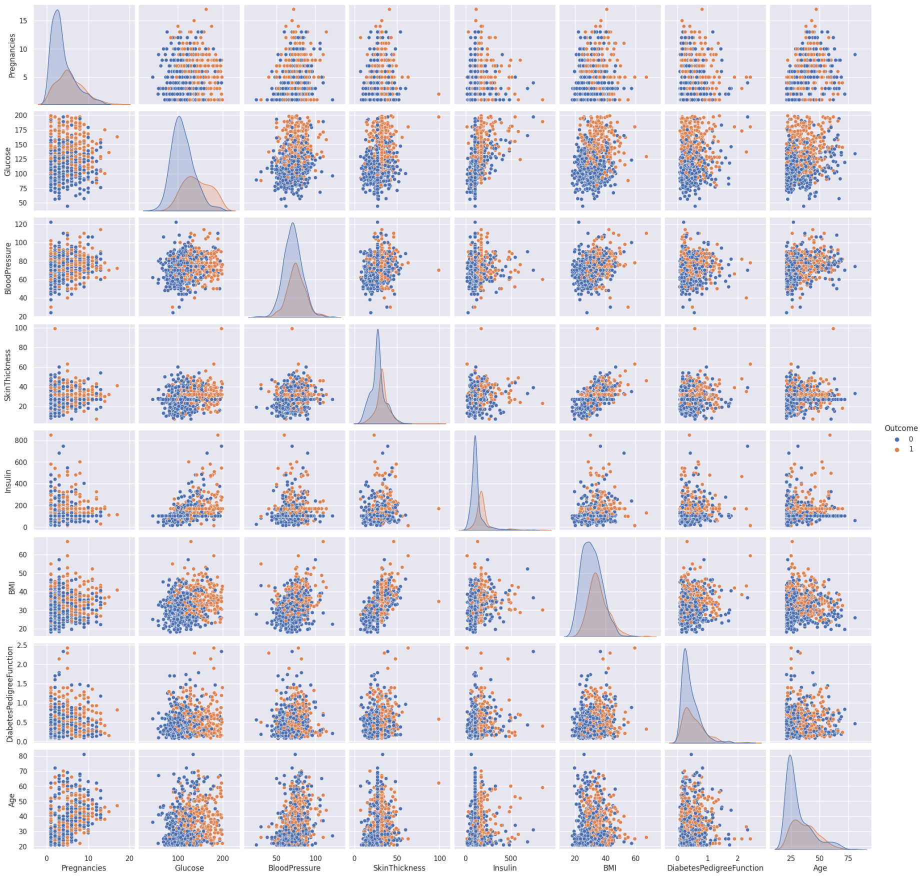
```
df.isnull().sum()
```

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

```
p = df.hist(figsize=(20,20), color='green')
```

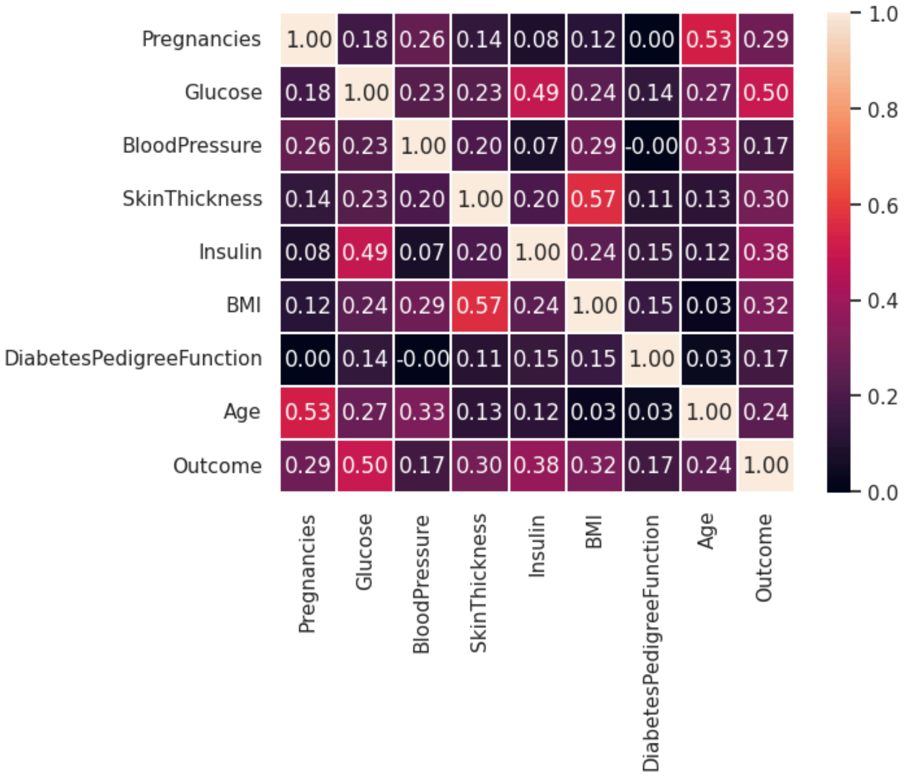


```
p =sns.pairplot(df , hue = 'Outcome')
```



```
sns.heatmap(df.corr(),annot=True,fmt='0.2f',linewidths=0.3)
```

```
<Axes: >
```



```
# Outlier Detection
for feature in df:
    Q1 = df[feature].quantile(0.25)
    Q3 = df[feature].quantile(0.75)
    IQR = Q3-Q1
    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
```

```
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
```

```
# 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_scores = lof.negative_outlier_factor_
np.sort(df_scores)[0:20]
thresold = np.sort(df_scores)[7]
outlier = df_scores>thresold
df = df[outlier]
```

```
#feature engineering
NewBMI = pd.Series(["Underweight","Normal", "Overweight","Obesity 1", "Obesity 2", "Obesity 3"], dtype = "category")
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]
```

```
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))
```

```
NewGlucose = pd.Series(["Low", "Normal", "Overweight", "Secret", "High"], dtype = "category")
df["NewGlucose"] = NewGlucose
df.loc[df["Glucose"] <= 70, "NewGlucose"] = NewGlucose[0]
df.loc[(df["Glucose"] > 70) & (df["Glucose"] <= 99), "NewGlucose"] = NewGlucose[1]
df.loc[(df["Glucose"] > 99) & (df["Glucose"] <= 126), "NewGlucose"] = NewGlucose[2]
df.loc[df["Glucose"] > 126 ,"NewGlucose"] = NewGlucose[3]
```

```
df = pd.get_dummies(df, columns = ["NewBMI", "NewInsulinScore", "NewGlucose"], drop_first=True)
```

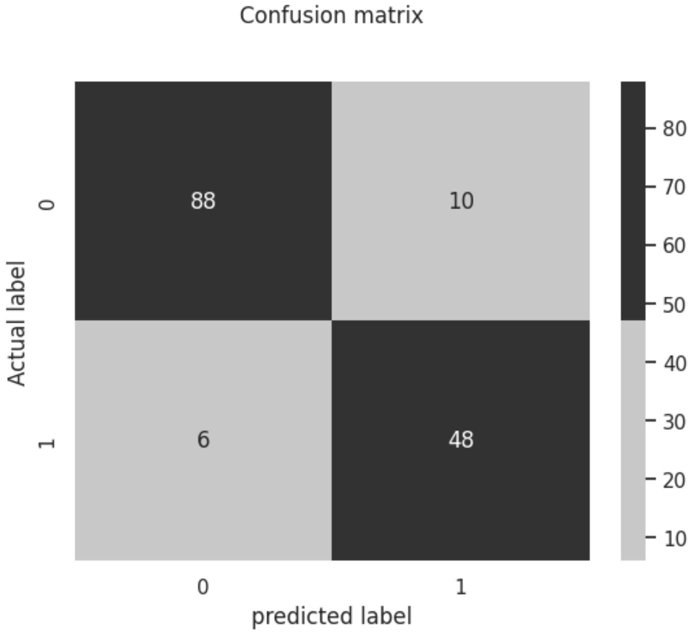
```
df.head()
```

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

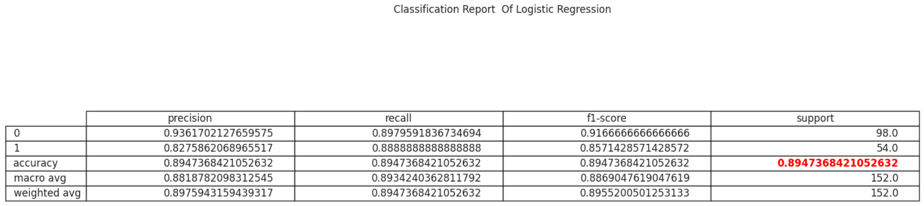
```
categorical_df = df[['NewBMI_Obesity 1',
       'NewBMI_Obesity 2', 'NewBMI_Obesity 3', 'NewBMI_Overweight',
       'NewBMI_Underweight', 'NewInsulinScore_Normal', 'NewGlucose_Low',
       'NewGlucose_Normal', 'NewGlucose_Overweight', 'NewGlucose_Secret']]
```

```
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=1)


cols = X.columns
index = X.index


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


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


X_train, X_test, y_train , y_test = train_test_split(X,y, test_size=0.2, random_state=0)


scaler =StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)


# LOGESTIC REGRESSION
log_reg = LogisticRegression()
log_reg.fit(X_train, y_train)
y_pred = log_reg.predict(X_test)
accuracy_score(y_train, log_reg.predict(X_train))
log_reg_acc = accuracy_score(y_test, log_reg.predict(X_test))


cnf_matrix = [[100, 20, 30],
              [10, 200, 40],
              [50, 60, 300]]

# Create a custom colormap
custom_cmap = sns.color_palette(["#CCCCCC", "#333333"])
cnf_matrix = metrics.confusion_matrix(y_test, y_pred)
p = sns.heatmap(pd.DataFrame(cnf_matrix), annot=True, cmap=custom_cmap, fmt='g')
plt. title ('Confusion matrix',y=1.1)
plt.ylabel('Actual label')
plt.xlabel('predicted label')
```

```
     Text(0.5, 19.049999999999997, 'predicted label')
```



```
report = classification_report(y_test, y_pred, output_dict=True)
report_df = pd.DataFrame(report).transpose()
plt.figure(figsize=(12, 6))
table = plt.table(cellText=report_df.values,
                  colLabels=report_df.columns,
                  rowLabels=report_df.index,
                  loc='center')
table.auto_set_font_size(False)
table.set_fontsize(12)
table.scale(1.5, 1.5)


accuracy_index = report_df.index.get_loc('accuracy')
accuracy_cells = table.get_celld()[(accuracy_index + 1, 3)]
accuracy_cells.set_text_props(fontweight='bold', color='red')

plt.axis('off')
plt.title('Classification Report  Of Logistic Regression')
plt.show()
```

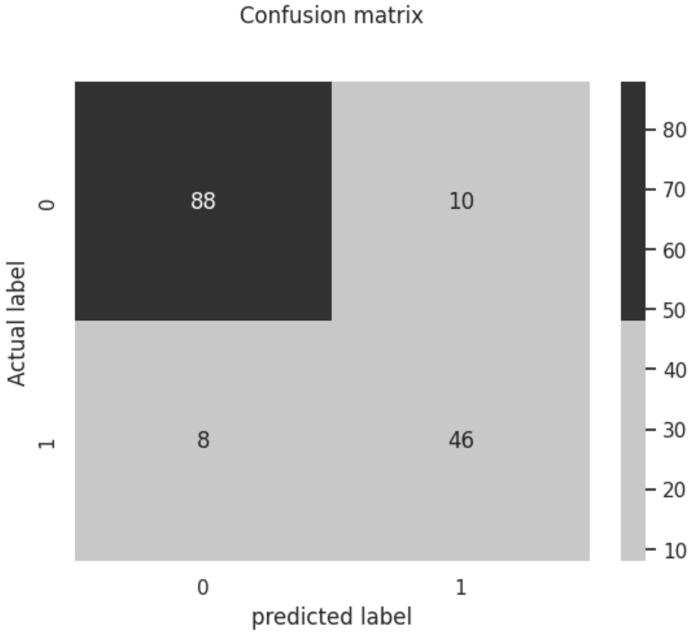


Classification Report Of Logistic Regression

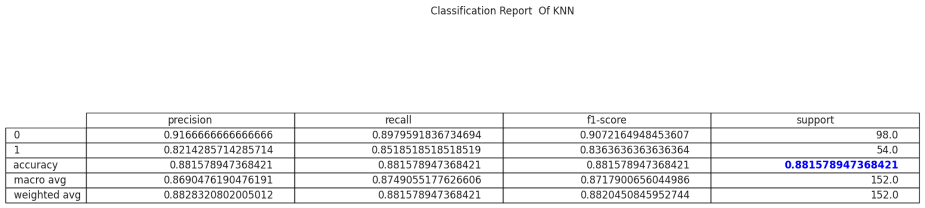
	precision	recall	f1-score	support
0	0.9361702127659575	0.8979591836734694	0.9166666666666666	98.0
1	0.8275862068965517	0.8888888888888888	0.8571428571428572	54.0
accuracy	0.8947368421052632	0.8947368421052632	0.8947368421052632	**0.8947368421052632**
macro avg	0.8818782098312545	0.8934240362811792	0.8869047619047619	152.0
weighted avg	0.8975943159439317	0.8947368421052632	0.8955200501253133	152.0

```
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)))
cnf_matrix = metrics.confusion_matrix(y_test, y_pred)
p = sns.heatmap(pd.DataFrame(cnf_matrix), annot=True, cmap=custom_cmap, fmt='g')
plt. title ('Confusion matrix',y=1.1)
plt.ylabel('Actual label')
plt.xlabel('predicted label')
```

```
    0.875
    0.881578947368421
    Text(0.5, 19.049999999999997, 'predicted label')
```



```
report = classification_report(y_test, y_pred, output_dict=True)
report_df = pd.DataFrame(report).transpose()
plt.figure(figsize=(12, 6))
table = plt.table(cellText=report_df.values,
                  colLabels=report_df.columns,
                  rowLabels=report_df.index,
                  loc='center')
table.auto_set_font_size(False)
table.set_fontsize(12)
table.scale(1.5, 1.5)


accuracy_index = report_df.index.get_loc('accuracy')
accuracy_cells = table.get_celld()[(accuracy_index + 1, 3)]
accuracy_cells.set_text_props(fontweight='bold', color='blue')

plt.axis('off')
plt.title('Classification Report  Of KNN')
plt.show()
```

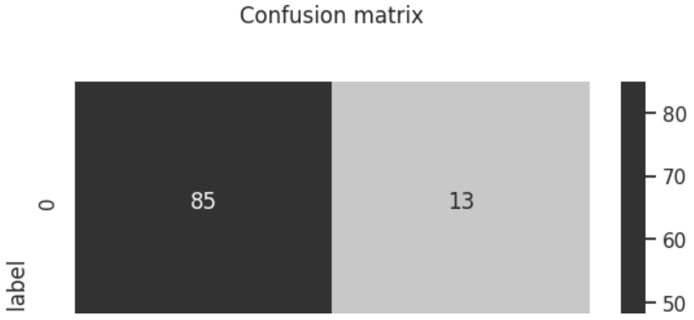


Classification Report Of KNN

	precision	recall	f1-score	support
0	0.9166666666666666	0.8979591836734694	0.9072164948453607	98.0
1	0.8214285714285714	0.8518518518518519	0.8363636363636364	54.0
accuracy	0.881578947368421	0.881578947368421	0.881578947368421	**0.881578947368421**
macro avg	0.8690476190476191	0.8749055177626606	0.8717900656044986	152.0
weighted avg	0.8828320802005012	0.881578947368421	0.8820450845952744	152.0

```
# Decision Tree
DT = DecisionTreeClassifier()
DT.fit(X_train, y_train)
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)))

cnf_matrix = metrics.confusion_matrix(y_test, y_pred)
p = sns.heatmap(pd.DataFrame(cnf_matrix), annot=True, cmap=custom_cmap, fmt='g')
plt. title ('Confusion matrix',y=1.1)
plt.ylabel('Actual label')
plt.xlabel('predicted label')
```

```
1.0
0.8552631578947368
Text(0.5, 19.049999999999997, 'predicted label')
```


Confusion matrix
label