Importing the Dependencies

In [3]:
```
import numpy as np
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn import svm
from sklearn.metrics import accuracy_score
from mlxtend.plotting import plot_decision_regions
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix
sns.set()
import warnings
warnings.filterwarnings('ignore')
%matplotlib inline
```

Data Collection and Analysis

PIMA Diabetes Dataset

In [4]:
```
diabetes_data = pd.read_csv('diabetes.csv')
```

In [5]:
```
diabetes_data.head()
```

Out[5]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFuncti
0	6	148	72	35	0	33.6	0.6
1	1	85	66	29	0	26.6	0.3
2	8	183	64	0	0	23.3	0.6
3	1	89	66	23	94	28.1	0.1
4	0	137	40	35	168	43.1	2.2

Basic EDA and statistical analysis

In [6]:
```
diabetes_data.shape
```

Out[6]: (768, 9)

In [7]:
```
# getting the statistical measures of the data
diabetes_data.describe()
```

Out[7]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Diabe
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	

1st Benchmark: Training the model with the raw data

In [8]:
```
# separating the data and labels
X = diabetes_data.drop(columns = 'Outcome', axis=1)
Y = diabetes_data['Outcome']
```

In [9]:
```
X_train, X_test, Y_train, Y_test = train_test_split(X,Y, test_size = 1/5,shuf
```

In [10]:
```
classifier = svm.SVC(kernel='linear')
```

In [11]:
```
#training the support vector Machine Classifier
classifier.fit(X_train, Y_train)
```

Out[11]:
```
SVC(kernel='linear')
```

In [12]:

```
# accuracy score on the training data
X_train_prediction = classifier.predict(X_train)
training_data_accuracy = accuracy_score(X_train_prediction, Y_train)

print('Accuracy score of the training data : ', training_data_accuracy)

# accuracy score on the test data
X_test_prediction = classifier.predict(X_test)
test_data_accuracy = accuracy_score(X_test_prediction, Y_test)

print('Accuracy score of the test data : ', test_data_accuracy)
```

```
Accuracy score of the training data :  0.7833876221498371
Accuracy score of the test data :  0.7532467532467533
```

Pre-processing:

On these columns, a value of zero does not make sense and thus indicates missing value. Following columns or variables have an invalid zero value: Glucose BloodPressure SkinThickness Insulin BMI

In [13]:

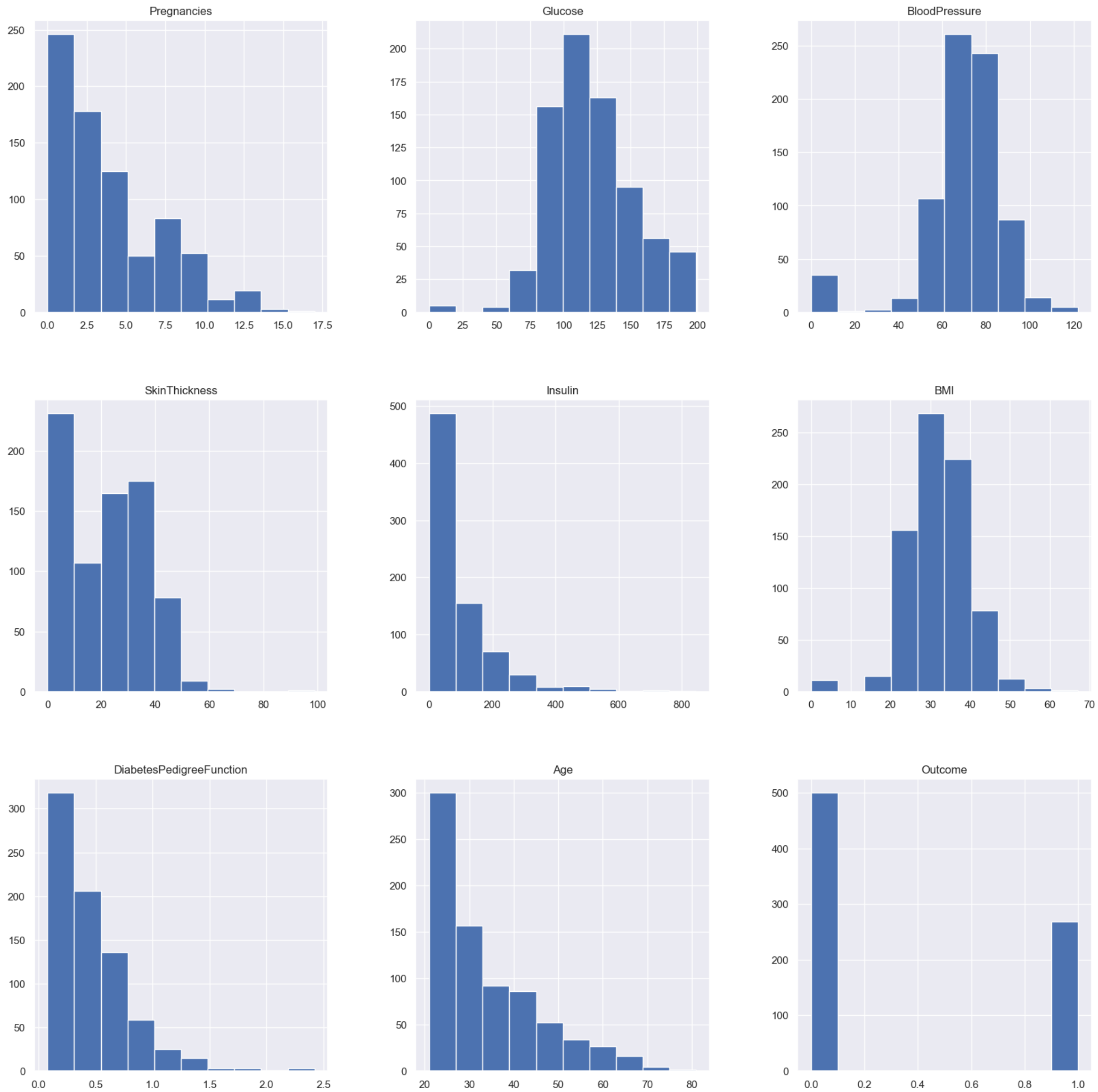
```
diabetes_data_copy = diabetes_data.copy(deep = True)
diabetes_data_copy[['Glucose','BloodPressure','SkinThickness','Insulin','BMI'


print(diabetes_data_copy.isnull().sum())
```

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

To fill these Nan values the data distribution needs to be understood

In [12]:
```
p = diabetes_data.hist(figsize = (20,20))
```



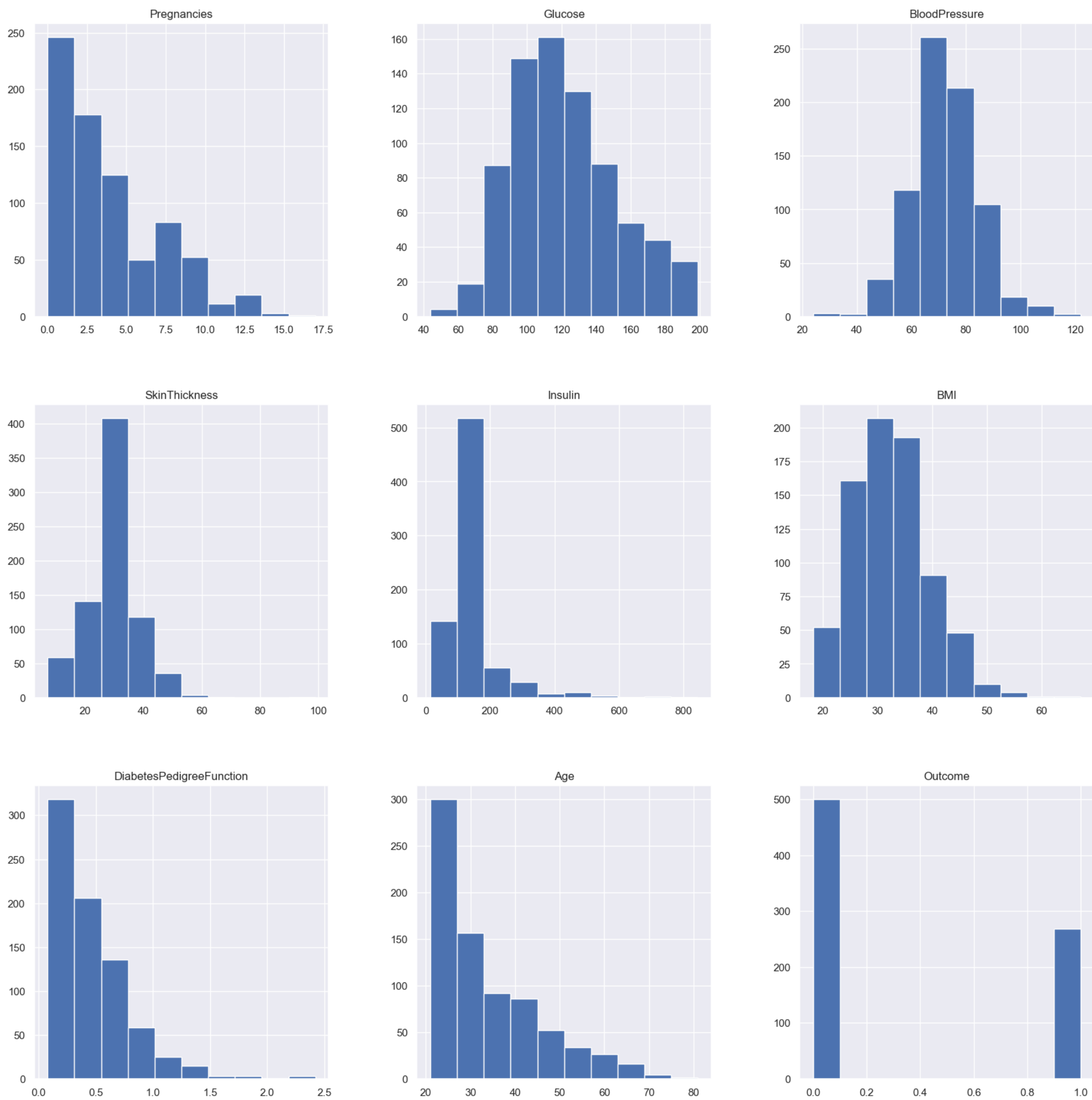
Replacing null values for columns considering their distribution

In [14]:

```
diabetes_data_copy['Glucose'].fillna(diabetes_data_copy['Glucose'].mean(), ir
diabetes_data_copy['BloodPressure'].fillna(diabetes_data_copy['BloodPressure'
diabetes_data_copy['SkinThickness'].fillna(diabetes_data_copy['SkinThickness'
diabetes_data_copy['Insulin'].fillna(diabetes_data_copy['Insulin'].median(),
diabetes_data_copy['BMI'].fillna(diabetes_data_copy['BMI'].median(), inplace
```

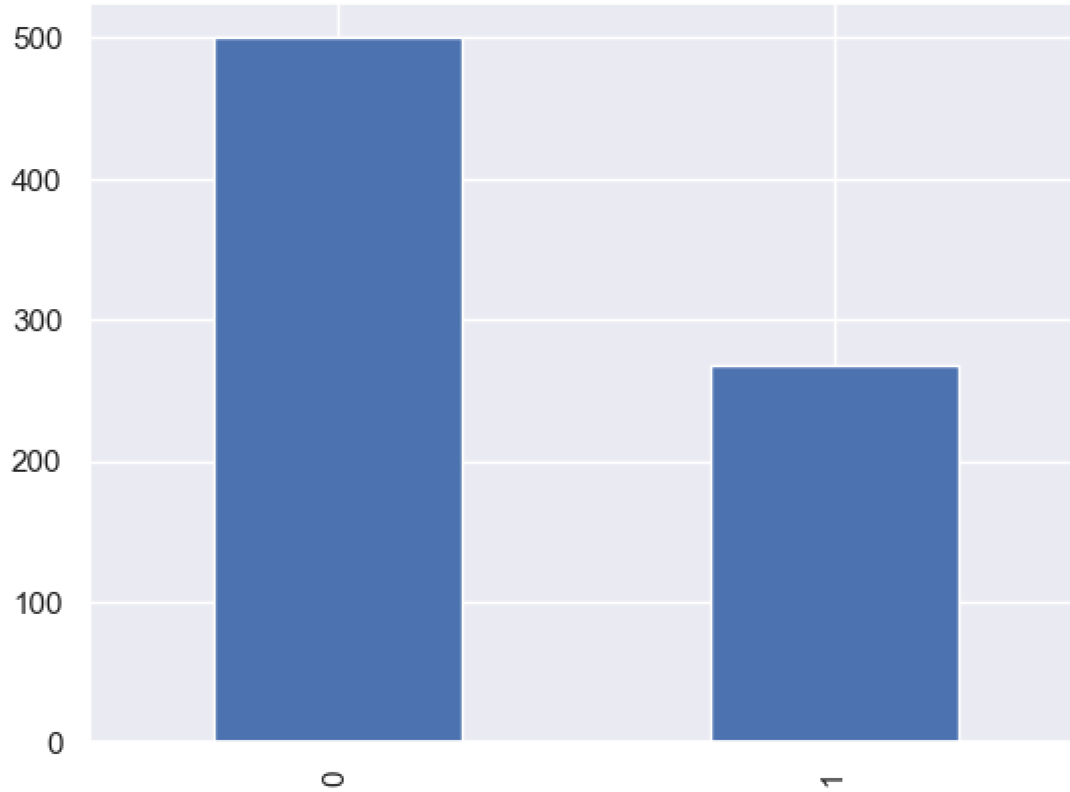
In [14]:

```
p = diabetes_data_copy.hist(figsize = (20,20))
```



In [15]:
```
color_wheel = {1: "#0392cf",
               2: "#7bc043"}
colors = diabetes_data["Outcome"].map(lambda x: color_wheel.get(x + 1))
print(diabetes_data.Outcome.value_counts())
p=diabetes_data.Outcome.value_counts().plot(kind="bar")
```

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



From the above graph we can say that the number of non-diabetics is almost twice the number of diabetic patients.

0 --> Non-Diabetic

1 --> Diabetic

In [15]:
```
diabetes_data_copy.groupby('Outcome').mean()
```

Out[15]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Dia
Outcome							
0	3.298000	110.710121	70.935397	27.726000	127.792000	30.885600	
1	4.865672	142.165573	75.147324	31.686567	164.701493	35.383582	

In [16]:
```
# separating the data and labels
X = diabetes_data_copy.drop(columns = 'Outcome', axis=1)
Y = diabetes_data_copy['Outcome']
```

In [18]:
```
print(X)
```

```
     Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0              6    148.0           72.0           35.0    125.0  33.6   
1              1     85.0           66.0           29.0    125.0  26.6   
2              8    183.0           64.0           29.0    125.0  23.3   
3              1     89.0           66.0           23.0     94.0  28.1   
4              0    137.0           40.0           35.0    168.0  43.1   
..           ...      ...            ...            ...      ...   ...   
763           10    101.0           76.0           48.0    180.0  32.9   
764            2    122.0           70.0           27.0    125.0  36.8   
765            5    121.0           72.0           23.0    112.0  26.2   
766            1    126.0           60.0           29.0    125.0  30.1   
767            1     93.0           70.0           31.0    125.0  30.4   

     DiabetesPedigreeFunction  Age  
0                       0.627   50  
1                       0.351   31  
2                       0.672   32  
3                       0.167   21  
4                       2.288   33  
..                        ...  ...  
763                     0.171   63  
764                     0.340   27  
765                     0.245   30  
766                     0.349   47  
767                     0.315   23  

[768 rows x 8 columns]
```

In [19]:
```
print(Y)
```

```
0      1
1      0
2      1
3      0
4      1
      ..
763    0
764    0
765    0
766    1
767    0
Name: Outcome, Length: 768, dtype: int64
```

Data Standardization

In [17]:
```
scaler = StandardScaler()
```

In [18]:
```
scaler.fit(X)
```

Out[18]:
```
StandardScaler()
```

In [19]:
```
standardized_data = scaler.transform(X)
```

In [20]:
```
print(standardized_data)
```

```
[[ 0.63994726  0.86510807 -0.03351824 ...  0.16661938  0.46849198
   1.4259954 ]
 [-0.84488505 -1.20616153 -0.52985903 ... -0.85219976 -0.36506078
  -0.19067191]
 [ 1.23388019  2.0158134  -0.69530596 ... -1.33250021  0.60439732
  -0.10558415]
 ...
 [ 0.3429808  -0.0225789  -0.03351824 ... -0.910418   -0.68519336
  -0.27575966]
 [-0.84488505  0.14180757 -1.02619983 ... -0.34279019 -0.37110101
   1.17073215]
 [-0.84488505 -0.94314317 -0.19896517 ... -0.29912651 -0.47378505
  -0.87137393]]
```

In [21]:
```
X = standardized_data
Y = diabetes_data_copy['Outcome']
```

In [22]:
```
print(X)
print(Y)
```

```
[[ 0.63994726  0.86510807 -0.03351824 ...  0.16661938  0.46849198
   1.4259954 ]
 [-0.84488505 -1.20616153 -0.52985903 ... -0.85219976 -0.36506078
  -0.19067191]
 [ 1.23388019  2.0158134  -0.69530596 ... -1.33250021  0.60439732
  -0.10558415]
 ...
 [ 0.3429808  -0.0225789  -0.03351824 ... -0.910418   -0.68519336
  -0.27575966]
 [-0.84488505  0.14180757 -1.02619983 ... -0.34279019 -0.37110101
   1.17073215]
 [-0.84488505 -0.94314317 -0.19896517 ... -0.29912651 -0.47378505
  -0.87137393]]
0      1
1      0
2      1
3      0
4      1
      ..
763    0
764    0
765    0
766    1
767    0
Name: Outcome, Length: 768, dtype: int64
```

Train Test Split

Linear

In [23]:
```
X_train, X_test, Y_train, Y_test = train_test_split(X,Y, test_size = 1/5, shu
```

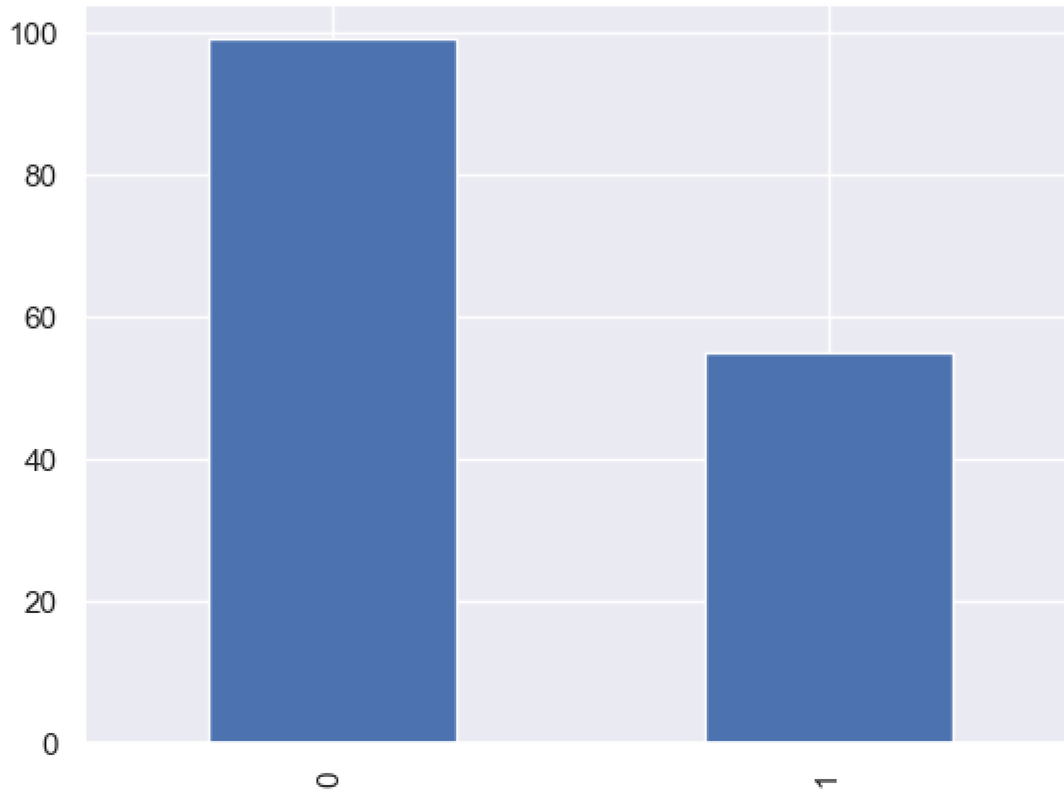
In [24]:
```
print(X.shape, X_train.shape, X_test.shape)
```

```
(768, 8) (614, 8) (154, 8)
```

```
Proportion of class variables
```

In [32]:
```
print(Y_test[Y_test == 1].count())
print(Y_test[Y_test == 0].count())
p=Y_test.value_counts().plot(kind="bar")
```

```
55
99
```



Training the Model - Linear SVM, before hyperparameter optimization

In [28]:
```
classifier = svm.SVC(kernel='linear')
```

In [29]:
```
#training the support vector Machine Classifier
classifier.fit(X_train, Y_train)
```

Out[29]:
```
SVC(kernel='linear')
```

In [30]:

```
# accuracy score on the training data
X_train_prediction = classifier.predict(X_train)
training_data_accuracy = accuracy_score(X_train_prediction, Y_train)

print('Accuracy score of the training data : ', training_data_accuracy)

# accuracy score on the test data
X_test_prediction = classifier.predict(X_test)
test_data_accuracy = accuracy_score(X_test_prediction, Y_test)

print('Accuracy score of the test data : ', test_data_accuracy)
```

```
Accuracy score of the training data :  0.7801302931596091
Accuracy score of the test data :  0.7597402597402597
```

SVM - Hyperparameter optimization

In [55]:

```
from sklearn.model_selection import GridSearchCV

# defining parameter range
param_grid = {'C': [0.1, 1, 10, 100, 1000],
              'gamma': [1, 0.1, 0.01, 0.001, 0.0001],
              'kernel': ['liner','rbf','poly']}

grid = GridSearchCV(svm.SVC(), param_grid, refit = True, verbose = 3)

#fitting the model for grid search
grid.fit(X_train, Y_train)
```

```
[CV 4/5] END ...C=0.1, gamma=0.0001, kernel=rbf;, score=0.659 total time
=   0.0s

[CV 5/5] END ...C=0.1, gamma=0.0001, kernel=rbf;, score=0.656 total time
=   0.0s
[CV 1/5] END ..C=0.1, gamma=0.0001, kernel=poly;, score=0.650 total time
=   0.0s
[CV 2/5] END ..C=0.1, gamma=0.0001, kernel=poly;, score=0.650 total time
=   0.0s
[CV 3/5] END ..C=0.1, gamma=0.0001, kernel=poly;, score=0.650 total time
=   0.0s
[CV 4/5] END ..C=0.1, gamma=0.0001, kernel=poly;, score=0.659 total time
=   0.0s
[CV 5/5] END ..C=0.1, gamma=0.0001, kernel=poly;, score=0.656 total time
=   0.0s
[CV 1/5] END .........C=1, gamma=1, kernel=liner;, score=nan total time
=   0.0s
[CV 2/5] END .........C=1, gamma=1, kernel=liner;, score=nan total time
=   0.0s
[CV 3/5] END .........C=1, gamma=1, kernel=liner;, score=nan total time
```

In [56]:
```
# print best parameter after tuning
print(grid.best_params_)

# print how our model looks after hyper-parameter tuning
print(grid.best_estimator_)
```

```
{'C': 1, 'gamma': 0.01, 'kernel': 'rbf'}
SVC(C=1, gamma=0.01)
```

In [57]:
```
from sklearn.metrics import classification_report, confusion_matrix
grid_predictions = grid.predict(X_test)

# print classification report
print(classification_report(Y_test, grid_predictions))
```

```
              precision    recall  f1-score   support

           0       0.75      0.90      0.82        99
           1       0.72      0.47      0.57        55

    accuracy                           0.75       154
   macro avg       0.74      0.69      0.70       154
weighted avg       0.74      0.75      0.73       154
```

Model 3 - KNN classification

In [34]:
```
from sklearn.neighbors import KNeighborsClassifier

test_scores = []
train_scores = []

for i in range(1,30):

    knn = KNeighborsClassifier(i)
    knn.fit(X_train,Y_train)

    train_scores.append(knn.score(X_train,Y_train))
    test_scores.append(knn.score(X_test,Y_test))
```

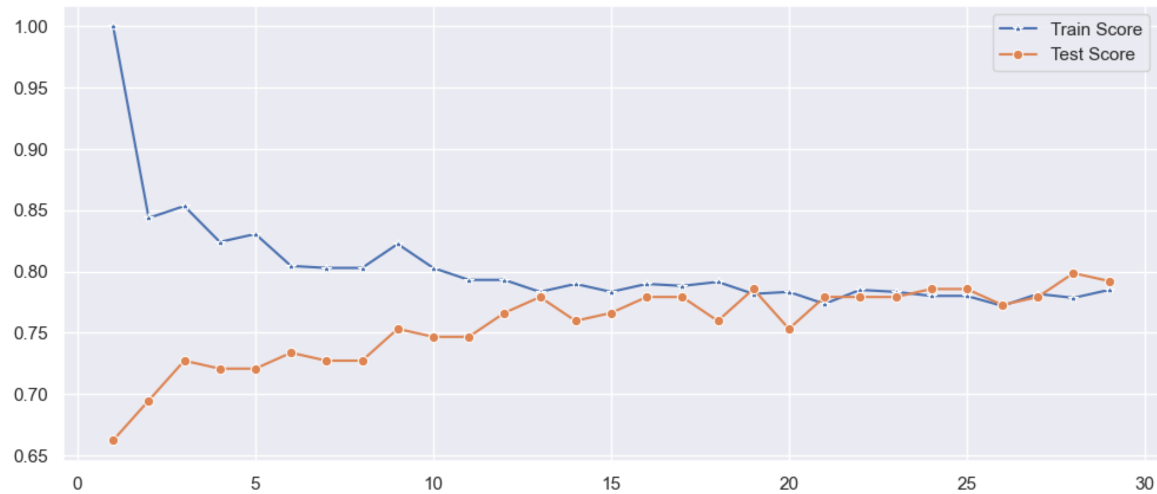
In [35]:
```
max_train_score = max(train_scores)
train_scores_ind = [i for i, v in enumerate(train_scores) if v == max_train_s
print('Max train score {} % and k = {}'.format(max_train_score*100,list(map(l
```

```
Max train score 100.0 % and k = [1]
```

In [36]:
```
max_test_score = max(test_scores)
test_scores_ind = [i for i, v in enumerate(test_scores) if v == max_test_scor
print('Max test score {} % and k = {}'.format(max_test_score*100,list(map(lam
```

```
Max test score 79.87012987012987 % and k = [28]
```

In [37]:
```
plt.figure(figsize=(12,5))
p = sns.lineplot(range(1,30),train_scores,marker='*',label='Train Score')
p = sns.lineplot(range(1,30),test_scores,marker='o',label='Test Score')
```



In [38]:
```
knn = KNeighborsClassifier(n_neighbors = 28,metric='euclidean',p=2)

knn.fit(X_train,Y_train)
knn.score(X_test,Y_test)
```

Out[38]:
```
0.7987012987012987
```

In [41]:
```
y_pred=knn.predict(X_test)

mat = confusion_matrix(Y_test, y_pred)
mat
```

Out[41]:
```
array([[93,  6],
       [25, 30]], dtype=int64)
```

In [52]:
```
from sklearn.datasets import load_iris
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import f_classif

print(X.shape)
X_new = SelectKBest(f_classif, k=2).fit_transform(X_train, Y_train)
X_new
X_test_new = SelectKBest(f_classif, k=2).fit_transform(X_test, Y_test)
```

```
(768, 8)
```

Predictive system

In [53]:

```
knn = KNeighborsClassifier(n_neighbors = 28)

knn.fit(X_new,Y_train)
knn.score(X_test_new,Y_test)
```

Out[53]: 0.7662337662337663

In [42]:

```
#input_data = (5,166,72,19,175,25.8,0.587,51)

input_data = []

print("Enter Pregnancy Month:")
input_data.append(input())

print("Enter Glucose Level:")
input_data.append(input())

print("Enter Blood Pressure Level:")
input_data.append(input())

print("Enter Skin Thickness of the Patient:")
input_data.append(input())

print("Enter Insulin Level:")
input_data.append(input())

print("Enter BMI of the Patient:")
input_data.append(input())

print("Enter Diabetese Pedegree Function:")
input_data.append(input())

print("Enter Patient Age:")
input_data.append(input())

# changing the input_data to numpy array
input_data_as_numpy_array = np.asarray(input_data)
print(input_data_as_numpy_array)
# reshape the array as we are predicting for one instance
input_data_reshaped = input_data_as_numpy_array.reshape(1,-1)

# standardize the input data
std_data = scaler.transform(input_data_reshaped)

prediction = knn.predict(std_data)
print(prediction)

if (prediction[0] == 0):
  print('The person is not diabetic')
else:
  print('The person is diabetic')
```

```
Enter Pregnancy Month:
4
Enter Glucose Level:
23
Enter Blood Pressure Level:
100
Enter Skin Thickness of the Patient:
14
Enter Insulin Level:
67.2
Enter BMI of the Patient:
```

```
172
Enter Diabetese Pedegree Function:
0.2
Enter Patient Age:
22
['4' '23' '100' '14' '67.2' '172' '0.2' '22']
[1]
The person is diabetic
```