Diabetes Project - Simplilearn

By P.Harish

In [1]:
```
#importing libraries
import pandas as pd
import seaborn as sb
import matplotlib.pyplot as plt
```

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

In [3]:
```
data.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
```

In [4]:
```
data
```

Out[4]:

	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.
...	...	...	...	...	...	...	
763	10	101	76	48	180	32.9	0.
764	2	122	70	27	0	36.8	0.
765	5	121	72	23	112	26.2	0.
766	1	126	60	0	0	30.1	0.
767	1	93	70	31	0	30.4	0.

768 rows × 9 columns

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

Out[5]:

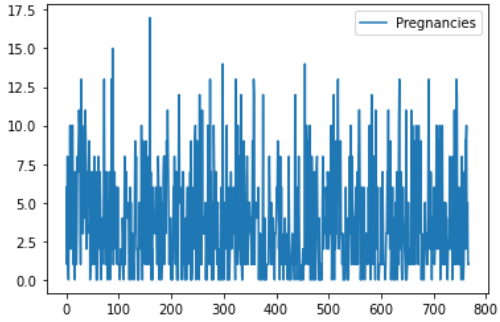
	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunctio
0	6	148	72	35	0	33.6	0.62
1	1	85	66	29	0	26.6	0.35
2	8	183	64	0	0	23.3	0.67
3	1	89	66	23	94	28.1	0.16
4	0	137	40	35	168	43.1	2.28

In [6]:
```
data.count
```

Out[6]:
```
<bound method DataFrame.count of      Pregnancies  Glucose  BloodPressure  S
kinThickness  Insulin   BMI  \
0               6      148             72             35        0  33.6
1               1       85             66             29        0  26.6
2               8      183             64              0        0  23.3
3               1       89             66             23       94  28.1
4               0      137             40             35      168  43.1
..            ...      ...            ...            ...      ...   ...
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

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

[768 rows x 9 columns]>
```

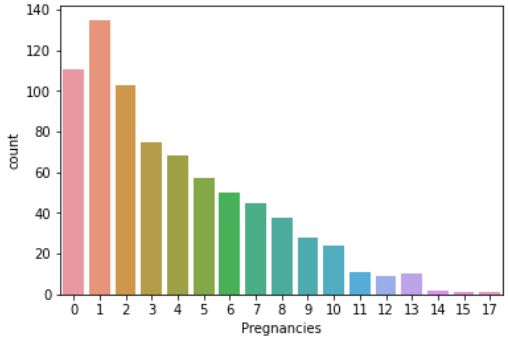
In [7]:
```
data.iloc[:,:1].plot()
```

Out[7]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x229f15ee460>
```



In [8]:
```
sb.countplot(data['Pregnancies'])
```

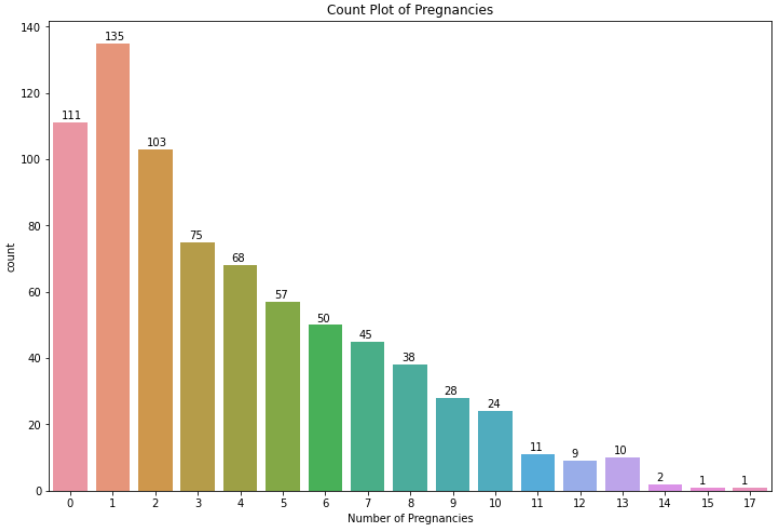
Out[8]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x229f1688d60>
```



In [9]:
```
data['Pregnancies'].value_counts()
```

Out[9]:
```
1     135
0     111
2     103
3      75
4      68
5      57
6      50
7      45
8      38
9      28
10     24
11     11
13     10
12      9
14      2
15      1
17      1
Name: Pregnancies, dtype: int64
```

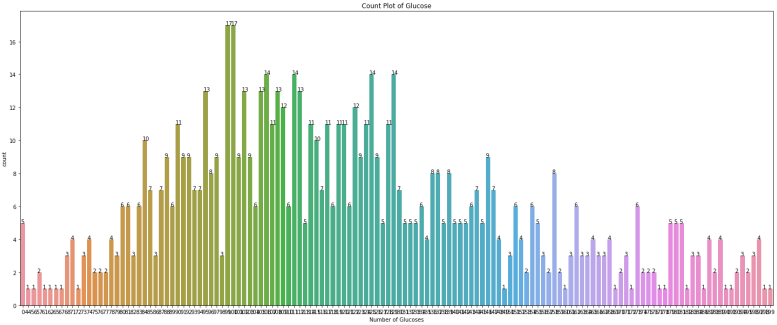
In [10]:
```
import matplotlib.pyplot as plt
plt.figure(figsize=(12,8))
ax = sb.countplot(x='Pregnancies',data = data)
plt.title('Count Plot of Pregnancies')
plt.xlabel('Number of Pregnancies')
for p in ax.patches:
        ax.annotate(format(p.get_height()), (p.get_x()+0.2, p.get_height()+1
))
```



In [11]:
```
data['Pregnancies'][data['Pregnancies']==0].count()
```

Out[11]: 111

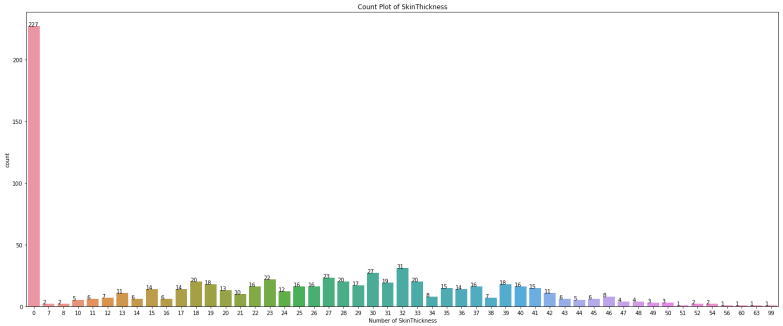
In [12]:
```
plt.figure(figsize=(25,10))
ax = sb.countplot(x='Glucose',data = data)
plt.title('Count Plot of Glucose')
plt.xlabel('Number of Glucoses')
for p in ax.patches:
        ax.annotate(format(p.get_height()), (p.get_x(), p.get_height()))
```



In [13]:
```
data['Glucose'][data['Glucose']==0].count()
```

Out[13]: 5

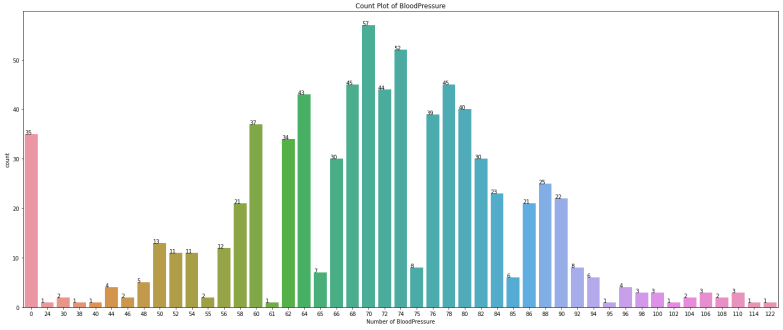
In [14]:
```
plt.figure(figsize=(25,10))
ax = sb.countplot(x='SkinThickness',data = data)
plt.title('Count Plot of SkinThickness')
plt.xlabel('Number of SkinThickness')
for p in ax.patches:
        ax.annotate(format(p.get_height()), (p.get_x(), p.get_height()))
```



In [15]:
```
data['SkinThickness'][data['SkinThickness']==0].count()
```

Out[15]: 227

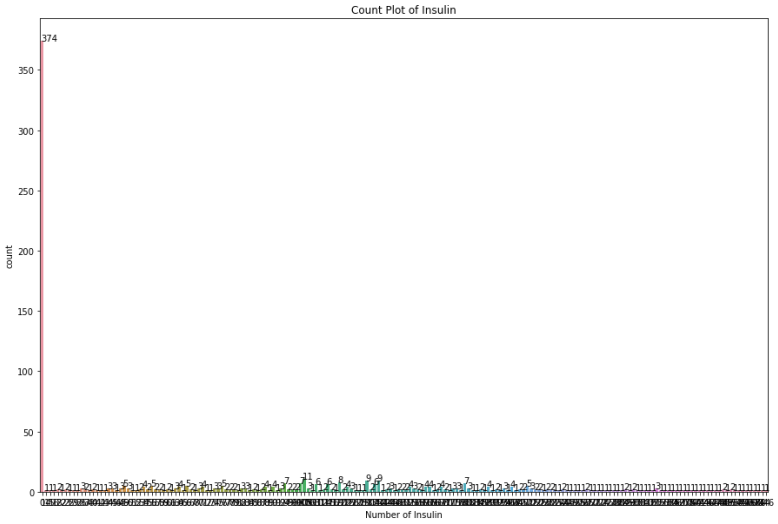
In [16]:
```
plt.figure(figsize=(25,10))
ax = sb.countplot(x='BloodPressure',data = data)
plt.title('Count Plot of BloodPressure')
plt.xlabel('Number of BloodPressure')
for p in ax.patches:
        ax.annotate(format(p.get_height()), (p.get_x(), p.get_height()))
```



In [17]:
```
data['BloodPressure'][data['BloodPressure']==0].count()
```

Out[17]: 35

In [18]:
```
plt.figure(figsize=(15,10))
ax = sb.countplot(x='Insulin',data=data)
plt.title('Count Plot of Insulin')
plt.xlabel('Number of Insulin')
for p in ax.patches:
        ax.annotate(format(p.get_height()), (p.get_x(), p.get_height()))
```



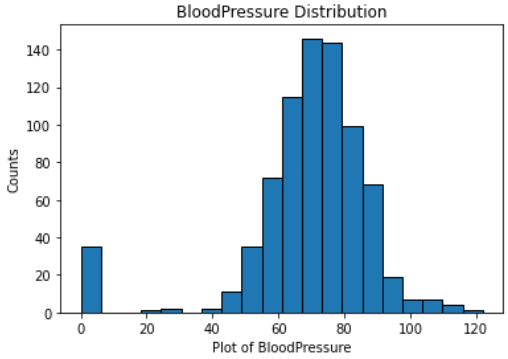
In [19]:
```
data['Insulin'][data['Insulin']==0].count()
```

Out[19]: 374

In [20]:
```
# plt.hist(data['BloodPressure'],bins=20, align='left', color='purple', edgec
olor='black')
# # plt.xticks(bins)
# plt.xlabel('Plot of BloodPressure')
# plt.ylabel('Counts')
# plt.title('BloodPressure Distribution')
```

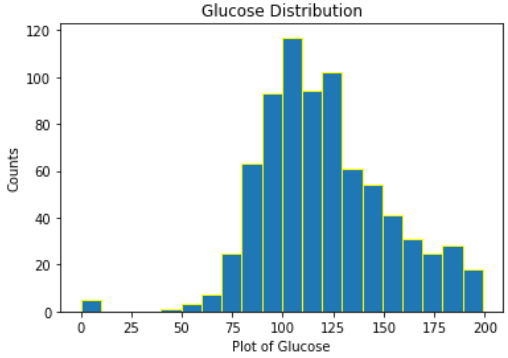
In [21]:
```
plt.show()
plt.hist(data.BloodPressure, bins=20, edgecolor='black')
plt.xlabel('Plot of BloodPressure')
plt.ylabel('Counts')
plt.title('BloodPressure Distribution')
```

Out[21]: Text(0.5, 1.0, 'BloodPressure Distribution')



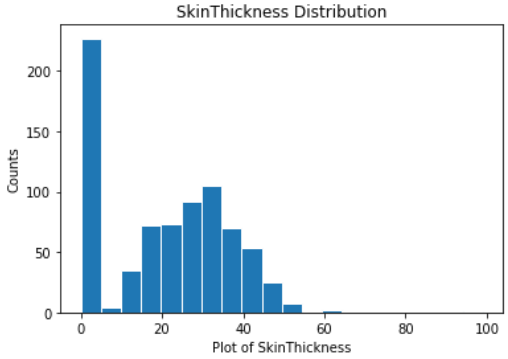
In [22]:
```
plt.show()
plt.hist(data.Glucose, bins=20, edgecolor='yellow')
plt.xlabel('Plot of Glucose')
plt.ylabel('Counts')
plt.title('Glucose Distribution')
```

Out[22]: Text(0.5, 1.0, 'Glucose Distribution')



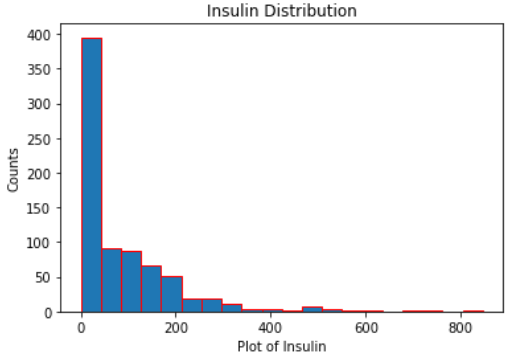
In [23]:
```
plt.show()
plt.hist(data.SkinThickness, bins=20, edgecolor='white')
plt.xlabel('Plot of SkinThickness')
plt.ylabel('Counts')
plt.title('SkinThickness Distribution')
```

Out[23]:
```
Text(0.5, 1.0, 'SkinThickness Distribution')
```



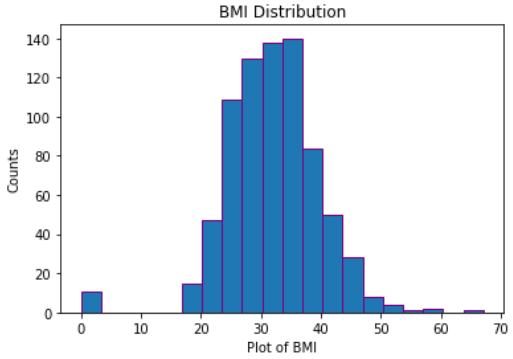
In [24]:
```
plt.show()
plt.hist(data.Insulin, bins=20, edgecolor='red')
plt.xlabel('Plot of Insulin')
plt.ylabel('Counts')
plt.title('Insulin Distribution')
```

Out[24]:
```
Text(0.5, 1.0, 'Insulin Distribution')
```



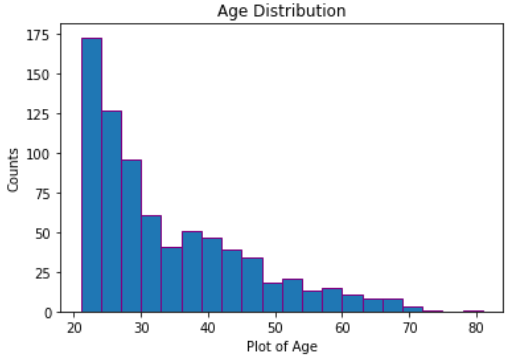
In [25]:
```
plt.show()
plt.hist(data.BMI, bins=20, edgecolor='purple')
plt.xlabel('Plot of BMI')
plt.ylabel('Counts')
plt.title('BMI Distribution')
```

Out[25]:
```
Text(0.5, 1.0, 'BMI Distribution')
```



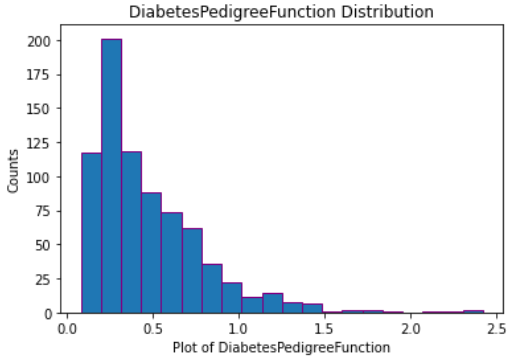
In [26]:
```
plt.show()
plt.hist(data.Age, bins=20, edgecolor='purple')
plt.xlabel('Plot of Age')
plt.ylabel('Counts')
plt.title('Age Distribution')
```

Out[26]:
```
Text(0.5, 1.0, 'Age Distribution')
```



In [27]:
```
plt.show()
plt.hist(data.DiabetesPedigreeFunction, bins=20, edgecolor='purple')
plt.xlabel('Plot of DiabetesPedigreeFunction')
plt.ylabel('Counts')
plt.title('DiabetesPedigreeFunction Distribution')
```

Out[27]: Text(0.5, 1.0, 'DiabetesPedigreeFunction Distribution')



In [28]:
```
import numpy as np
data[data.iloc[:,1:6]==0]=np.nan
```

In [29]:
```
data
```

Out[29]:

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

768 rows × 9 columns

In [30]:
```
data.fillna(data.median(),inplace=True)
```

In [31]:
```
data.info()
```

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

In [32]:
```
data.iloc[:,1:5]=data.iloc[:,1:5].astype(np.int64)
```

In [33]:
```
data.info()
```

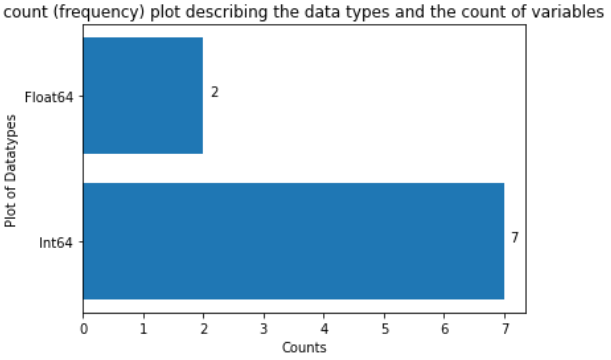
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Pregnancies               768 non-null    int64  
 1   Glucose                   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
```

In [34]:
```
# plt.show()
# plt.hist(data.Glucose, bins=20, edgecolor='white')
# plt.xlabel('Plot of SkinThickness')
# plt.ylabel('Counts')
# plt.title('SkinThickness Distribution')
```

In [35]:
```
# plt.show()
# sb.countplot(data.dtypes.value_counts())

# plt.xlabel('Plot of Datatypes')
# plt.ylabel('Counts')
# plt.title('count (frequency) plot describing the data types and the count o
f variables')
```

In [36]:
```
k=['Int64','Float64']
v=data.dtypes.value_counts().values
plt.barh(k,v)
for index, value in enumerate(v):
    plt.text(value+0.1, index, str(value))
plt.xlabel('Counts')
plt.ylabel('Plot of Datatypes')
plt.title('count (frequency) plot describing the data types and the count of
 variables')
```

Out[36]:
```
Text(0.5, 1.0, 'count (frequency) plot describing the data types and the cou
nt of variables')
```



In [37]:
```
data.dtypes.value_counts()
```

Out[37]:
```
int64      7
float64    2
dtype: int64
```

In [38]:
```
data.iloc[:,[0,6,7,8]]
```

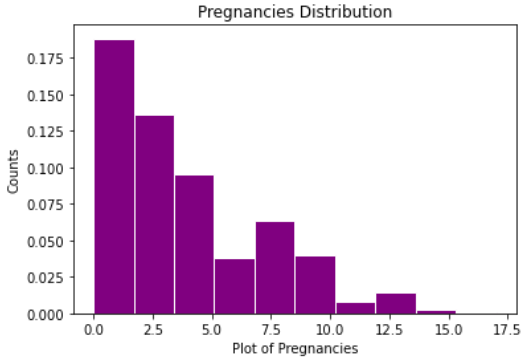
Out[38]:

	Pregnancies	DiabetesPedigreeFunction	Age	Outcome
0	6	0.627	50	1
1	1	0.351	31	0
2	8	0.672	32	1
3	1	0.167	21	0
4	0	2.288	33	1
...	...	...	...	...
763	10	0.171	63	0
764	2	0.340	27	0
765	5	0.245	30	0
766	1	0.349	47	1
767	1	0.315	23	0

768 rows × 4 columns

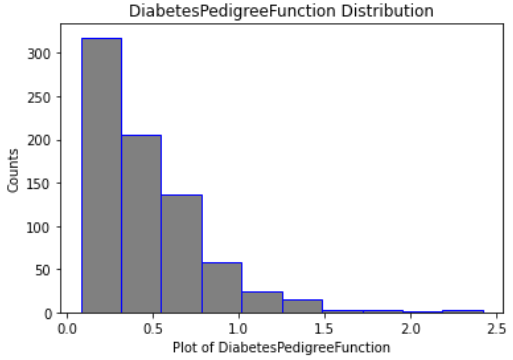
In [39]:
```
plt.show()
data1=data.Pregnancies
bins,density,_ = plt.hist(data.Pregnancies,density=True,color='purple',edgeco
lor='White')
plt.xlabel('Plot of Pregnancies')
plt.ylabel('Counts')
plt.title('Pregnancies Distribution')
# count, _ = np.histogram(data, bins)
# for x,y,num in zip(bins, density, count):
#     if num != 0:
#         plt.text(x, y+0.05, num, fontsize=10, rotation=-90) # x,y,str
```

Out[39]: Text(0.5, 1.0, 'Pregnancies Distribution')

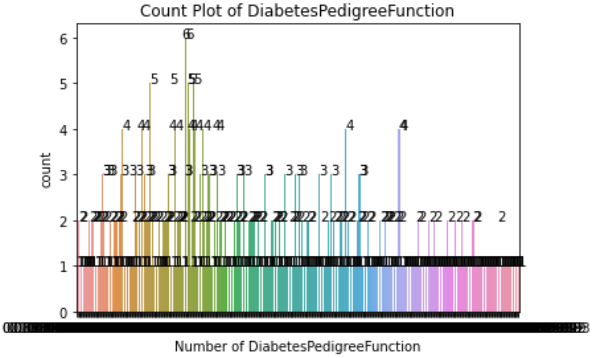


In [40]:
```
plt.show()
plt.hist(data.DiabetesPedigreeFunction,color='grey',edgecolor='blue')
plt.xlabel('Plot of DiabetesPedigreeFunction')
plt.ylabel('Counts')
plt.title('DiabetesPedigreeFunction Distribution')
```

Out[40]: Text(0.5, 1.0, 'DiabetesPedigreeFunction Distribution')



In [41]:
```
# plt.figure(figsize=(50,15))
ax = sb.countplot(x='DiabetesPedigreeFunction',data = data)
plt.title('Count Plot of DiabetesPedigreeFunction')
plt.xlabel('Number of DiabetesPedigreeFunction')
for p in ax.patches:
        ax.annotate(format(p.get_height()), (p.get_x(), p.get_height()))
```



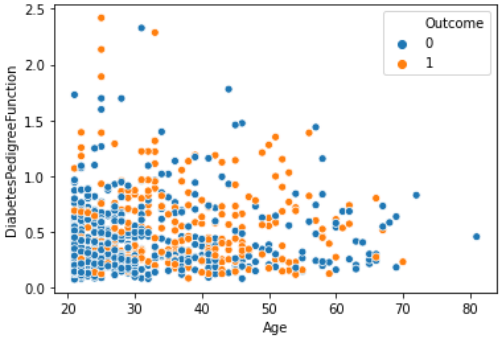
In [42]:
```
# sb.scatterplot(data=data, x="total_bill", y="tip")
data.head()
```

Out[42]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunctio
0	6	148	72	35	125	33.6	0.62
1	1	85	66	29	125	26.6	0.35
2	8	183	64	29	125	23.3	0.67
3	1	89	66	23	94	28.1	0.16
4	0	137	40	35	168	43.1	2.28

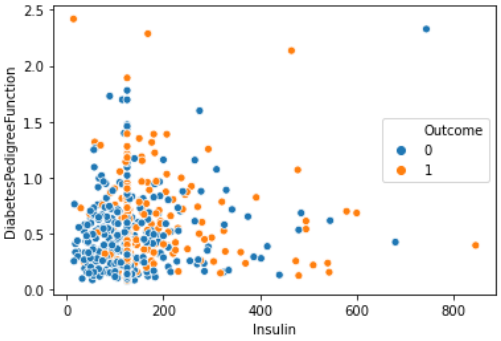
In [43]:
```
sb.scatterplot(data=data,x='Age',y='DiabetesPedigreeFunction',hue="Outcome")
```

Out[43]: <matplotlib.axes._subplots.AxesSubplot at 0x229f1f61940>



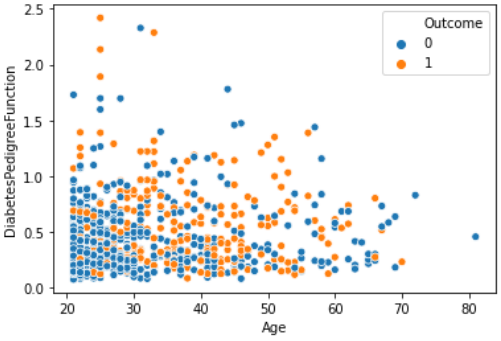
In [44]:
```
sb.scatterplot(data=data,x='Insulin',y='DiabetesPedigreeFunction',hue="Outcome")
```

Out[44]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x229f26f4ee0>
```



In [45]:
```
sb.scatterplot(data=data,x='Age',y='DiabetesPedigreeFunction',hue="Outcome")
```

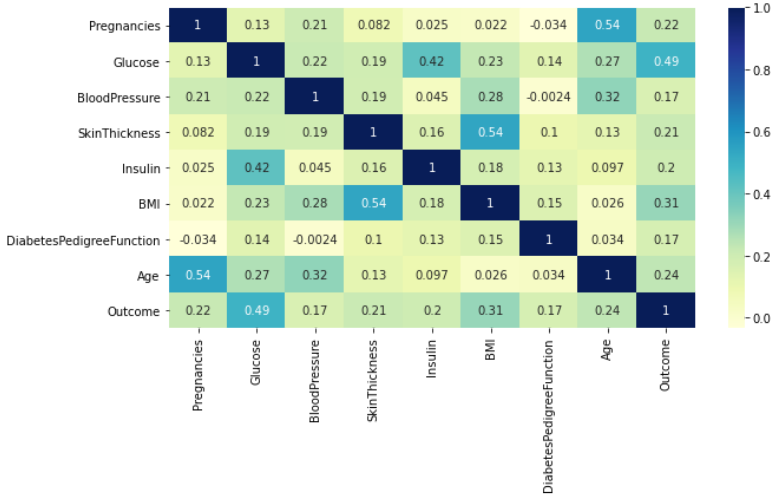
Out[45]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x229f1c5b490>
```



In [46]:
```
print(data.corr())
```

```
                          Pregnancies   Glucose  BloodPressure  SkinThicknes
s  \
Pregnancies                  1.000000  0.128213       0.208615       0.08177
0
Glucose                      0.128213  1.000000       0.218937       0.19261
5
BloodPressure                0.208615  0.218937       1.000000       0.19189
2
SkinThickness                0.081770  0.192615       0.191892       1.00000
0
Insulin                      0.025047  0.419451       0.045363       0.15561
0
BMI                          0.021559  0.231049       0.281257       0.54320
5
DiabetesPedigreeFunction    -0.033523  0.137327      -0.002378       0.10218
8
Age                          0.544341  0.266909       0.324915       0.12610
7
Outcome                      0.221898  0.492782       0.165723       0.21487
3

                           Insulin       BMI  DiabetesPedigreeFunction  \
Pregnancies               0.025047  0.021559                 -0.033523
Glucose                   0.419451  0.231049                  0.137327
BloodPressure             0.045363  0.281257                 -0.002378
SkinThickness             0.155610  0.543205                  0.102188
Insulin                   1.000000  0.180241                  0.126503
BMI                       0.180241  1.000000                  0.153438
DiabetesPedigreeFunction  0.126503  0.153438                  1.000000
Age                       0.097101  0.025597                  0.033561
Outcome                   0.203790  0.312038                  0.173844

                               Age   Outcome
Pregnancies               0.544341  0.221898
Glucose                   0.266909  0.492782
BloodPressure             0.324915  0.165723
SkinThickness             0.126107  0.214873
Insulin                   0.097101  0.203790
BMI                       0.025597  0.312038
DiabetesPedigreeFunction  0.033561  0.173844
Age                       1.000000  0.238356
Outcome                   0.238356  1.000000
```

In [47]:
```
import seaborn as sb
plt.figure(figsize=(10,5))
# plotting correlation heatmap
dataplot = sb.heatmap(data.corr(), cmap="YlGnBu", annot=True)

# displaying heatmap
plt.show()
```



Model Modeling

In [48]:
```
from sklearn.model_selection import train_test_split
```

In [49]:
```
X=data.iloc[:,:-1]
y=data.iloc[:,-1]
```

In [50]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30,
random_state=101)
```

In [51]:
```
from sklearn.tree import DecisionTreeClassifier
dtree = DecisionTreeClassifier()
dtree.fit(X_train,y_train)
```

Out[51]:
```
DecisionTreeClassifier()
```

In [52]:
```
predictions = dtree.predict(X_test)
from sklearn.metrics import classification_report,confusion_matrix,accuracy_s
core
print(classification_report(y_test,predictions))
dtree_score=accuracy_score(y_test,predictions)
print(dtree_score)
```

```
              precision    recall  f1-score   support

           0       0.83      0.81      0.82       150
           1       0.67      0.69      0.68        81

    accuracy                           0.77       231
   macro avg       0.75      0.75      0.75       231
weighted avg       0.77      0.77      0.77       231

0.7705627705627706
```

In [53]:
```
print(confusion_matrix(y_test,predictions))
```

```
[[122  28]
 [ 25  56]]
```

In [54]:
```
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier(n_estimators=1000)
rfc.fit(X_train,y_train)
```

Out[54]:
```
RandomForestClassifier(n_estimators=1000)
```

In [55]:
```
predictions2 = rfc.predict(X_test)
from sklearn.metrics import classification_report,confusion_matrix,accuracy_s
core
print(classification_report(y_test,predictions2))
RandomForest_score=accuracy_score(y_test,predictions2)
print(RandomForest_score)
```

```
              precision    recall  f1-score   support

           0       0.81      0.85      0.83       150
           1       0.69      0.63      0.66        81

    accuracy                           0.77       231
   macro avg       0.75      0.74      0.74       231
weighted avg       0.77      0.77      0.77       231

0.7705627705627706
```

In [56]:
```
print(confusion_matrix(y_test,predictions2))
```

```
[[127  23]
 [ 30  51]]
```

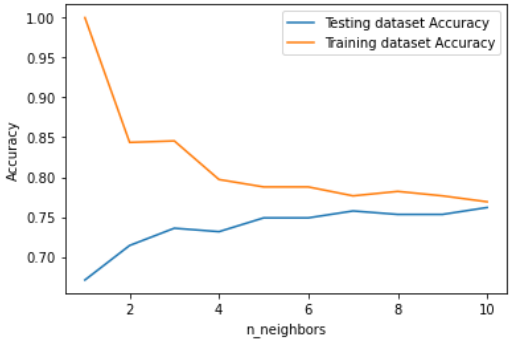
In [57]:
```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=7)
knn.fit(X_train, y_train)

# Predict on dataset which model has not seen before
print(knn.predict(X_test))
```

```
[1 1 0 0 0 0 1 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 1 0 0 1 0 0 0 1 1 0 0 0 0 0 1
 1 1 0 0 0 0 1 0 0 1 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 1 0 1 0 0 0 0 0 1 0 0 0
 1 1 0 1 0 0 0 0 1 0 1 0 1 0 1 1 1 0 1 0 0 0 1 0 1 1 1 0 0 1 0 0 0 0 0 0 1
 0 0 0 1 0 0 0 0 0 1 0 0 1 1 0 0 0 0 0 0 0 1 0 1 0 1 0 0 0 0 0 1 0 1 1 0 1 0
 0 1 0 1 0 0 0 1 1 0 0 1 1 0 0 1 0 1 0 1 1 0 0 1 0 0 0 0 0 0 1 0 0 1 0 1 0
 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1
 1 0 0 1 0 1 1 0 1]
```

In [58]:
```
knn_score=knn.score(X_test, y_test)
print(knn_score)
```

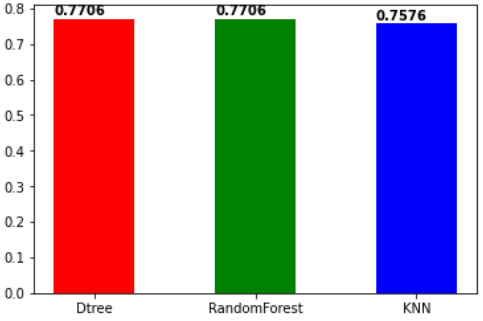
```
0.7575757575757576
```

In [59]:
```
neighbors = np.arange(1, 11)
train_accuracy = np.empty(len(neighbors))
test_accuracy = np.empty(len(neighbors))

# Loop over K values
for i, k in enumerate(neighbors):
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)

    # Compute traning and test data accuracy
    train_accuracy[i] = knn.score(X_train, y_train)
    test_accuracy[i] = knn.score(X_test, y_test)

# Generate plot
plt.plot(neighbors, test_accuracy, label = 'Testing dataset Accuracy')
plt.plot(neighbors, train_accuracy, label = 'Training dataset Accuracy')

plt.legend()
plt.xlabel('n_neighbors')
plt.ylabel('Accuracy')
plt.show()
```



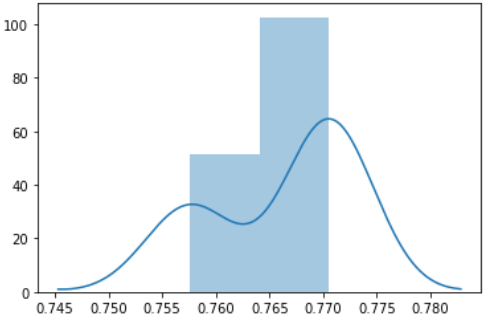
In [60]:
```
x=[dtree_score,RandomForest_score,knn_score]
fig, ax = plt.subplots()
# x-coordinates of left sides of bars
left = [1, 2, 3]

# heights of bars
height = [dtree_score,RandomForest_score,knn_score]

# labels for bars
tick_label = ['Dtree', 'RandomForest', 'KNN']

# plotting a bar chart
b=ax.bar(left, height, tick_label = tick_label,
        width = 0.5, color = ['red', 'green','blue'], label=height)

for p in b.patches:
    print("{}".format(str(p.get_height()*100)))
    ax.text(p.get_x(), p.get_height()+0.01,round(float(format(p.get_height
()())),4),color = 'black', fontweight = 'bold')
plt.show()
```

```
77.05627705627705
77.05627705627705
75.75757575757575
```



In [61]:
```
sb.distplot(x)
```

Out[61]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x229f5165490>
```



Model evaluation metrics

Classification accuracy: percentage of correct predictions

In [62]:
```
from sklearn import metrics
print(metrics.accuracy_score(y_test,predictions2))
```

```
0.7705627705627706
```

Null accuracy:

In [63]:
```
# examine the class distribution of the testing set (using a Pandas Series me
thod)
y_test.value_counts()
```

Out[63]:
```
0    150
1     81
Name: Outcome, dtype: int64
```

In [64]:
```
# calculate the percentage of zeros
1 - y_test.mean()
```

Out[64]:
```
0.6493506493506493
```

In [65]:
```
# calculate null accuracy in a single line of code
# only for binary classification problems coded as 0/1
max(y_test.mean(), 1 - y_test.mean())
```

Out[65]:
```
0.6493506493506493
```

This means that a dumb model that always predicts 0 would be right 65% of the time

This shows how classification accuracy is not that good as it's close to a dumb model

It's a good way to know the minimum we should achieve with our models

Confusion Matrix

In [66]:
```
print(metrics.confusion_matrix(y_test, predictions2))
```

```
[[127  23]
 [ 30  51]]
```

In [67]:
```
confusion = metrics.confusion_matrix(y_test, predictions2)
print(confusion)
#[row, column]
TP = confusion[1, 1]
TN = confusion[0, 0]
FP = confusion[0, 1]
FN = confusion[1, 0]
```

```
[[127  23]
 [ 30  51]]
```

Metrics computed from a confusion matrix

Classification Accuracy: Overall, how often is the classifier correct?

In [68]:
```
print((TP + TN) / float(TP + TN + FP + FN))
print(metrics.accuracy_score(y_test,predictions2 ))
```

```
0.7705627705627706
0.7705627705627706
```

Classification Error: Overall, how often is the classifier incorrect?

In [69]:
```
classification_error = (FP + FN) / float(TP + TN + FP + FN)

print(classification_error)
print(1 - metrics.accuracy_score(y_test, predictions2))
```

```
0.22943722943722944
0.22943722943722944
```

Sensitivity: When the actual value is positive, how often is the prediction correct?

In [70]:
```
sensitivity = TP / float(FN + TP)

print(sensitivity)
print(metrics.recall_score(y_test, predictions2))
```

```
0.6296296296296297
0.6296296296296297
```

Specificity: When the actual value is negative, how often is the prediction correct?

In [71]:
```
specificity = TN / float(TN + FP)

print(specificity)
```

```
0.8466666666666667
```

Our classifier

Highly specific Not sensitive

In [72]:
```
# train a logistic regression model on the training set
from sklearn.linear_model import LogisticRegression

# instantiate model
logreg = LogisticRegression()

# fit model
logreg.fit(X_train, y_train)
```

```
C:\Users\pakalamanda.harish\Anaconda3\lib\site-packages\sklearn\linear_model
\_logistic.py:762: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regre
ssion
  n_iter_i = _check_optimize_result(
```

Out[72]:
```
LogisticRegression()
```

In [73]:
```
# make class predictions for the testing set
y_pred_class = logreg.predict(X_test)
```

In [74]:
```
# calculate accuracy
from sklearn import metrics
print(metrics.accuracy_score(y_test, y_pred_class))
```

```
0.7878787878787878
```

In [75]:
```
# print the first 10 predicted responses
# 1D array (vector) of binary values (0, 1)
logreg.predict(X_test)[0:10]
```

Out[75]:
```
array([0, 1, 0, 0, 0, 1, 1, 0, 0, 0], dtype=int64)
```

In [76]:
```
# print the first 10 predicted probabilities of class membership
logreg.predict_proba(X_test)[0:10]
```

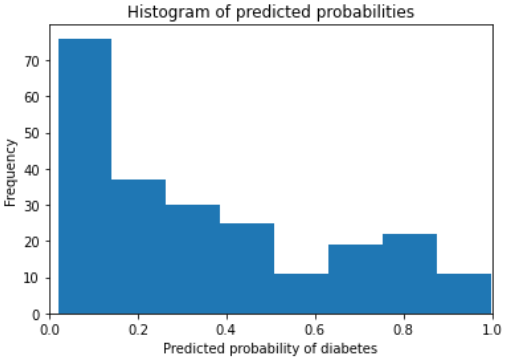
Out[76]:
```
array([[0.63882615, 0.36117385],
       [0.16100006, 0.83899994],
       [0.90556801, 0.09443199],
       [0.57053248, 0.42946752],
       [0.94949388, 0.05050612],
       [0.00149873, 0.99850127],
       [0.20324683, 0.79675317],
       [0.96402753, 0.03597247],
       [0.65522383, 0.34477617],
       [0.55127476, 0.44872524]])
```

In [77]:
```
# store the predicted probabilities for class 1
y_pred_prob = logreg.predict_proba(X_test)[:, 1]
```

In [78]:
```
# allow plots to appear in the notebook
%matplotlib inline
import matplotlib.pyplot as plt

# adjust the font size
plt.rcParams['font.size'] = 10
```

In [79]:
```
# histogram of predicted probabilities

# 8 bins
plt.hist(y_pred_prob, bins=8)

# x-axis limit from 0 to 1
plt.xlim(0,1)
plt.title('Histogram of predicted probabilities')
plt.xlabel('Predicted probability of diabetes')
plt.ylabel('Frequency')
```

Out[79]:
```
Text(0, 0.5, 'Frequency')
```



We can see from the first bar

About 35% of observations have probability from 0 to 0.2

Small number of observations with probability > 0.5

This is below the threshold of 0.5

Most would be predicted "no diabetes" in this case

In [80]:
```
# predict diabetes if the predicted probability is greater than 0.3
from sklearn.preprocessing import binarize
# it will return 1 for all values above 0.3 and 0 otherwise
# results are 2D so we slice out the first column
y_pred_class1 = binarize([y_pred_prob],threshold=0.3)
```

In [81]:
```
# print the first 10 predicted probabilities
y_pred_prob[0:10]
```

Out[81]:
```
array([0.36117385, 0.83899994, 0.09443199, 0.42946752, 0.05050612,
       0.99850127, 0.79675317, 0.03597247, 0.34477617, 0.44872524])
```

In [82]:
```
# print the first 10 predicted classes with the lower threshold
y_pred_class[0:10]
```

Out[82]:
```
array([0, 1, 0, 0, 0, 1, 1, 0, 0, 0], dtype=int64)
```

In [83]:
```
# previous confusion matrix (default threshold of 0.5)
print(confusion)
```

```
[[127  23]
 [ 30  51]]
```

In [84]:
```
# new confusion matrix (threshold of 0.3)
print(metrics.confusion_matrix(y_test, y_pred_class))
```

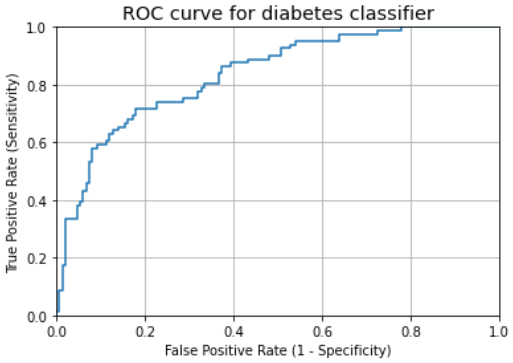
```
[[134  16]
 [ 33  48]]
```

In [85]:
```
# sensitivity has decreased (used to be 0.64)
print (48 / float(33 + 48))
```

```
0.5925925925925926
```

Receiver Operating Characteristic (ROC)

In [86]:
```
# we pass y_test and y_pred_prob
# we do not use y_pred_class, because it will give incorrect results without
 generating an error
# roc_curve returns 3 objects fpr, tpr, thresholds
# fpr: false positive rate
# tpr: true positive rate
fpr, tpr, thresholds = metrics.roc_curve(y_test, y_pred_prob)

plt.plot(fpr, tpr)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.0])
plt.rcParams['font.size'] = 12
plt.title('ROC curve for diabetes classifier')
plt.xlabel('False Positive Rate (1 - Specificity)')
plt.ylabel('True Positive Rate (Sensitivity)')
plt.grid(True)
```



In [87]:
```
# define a function that accepts a threshold and prints sensitivity and speci
ficity
def evaluate_threshold(threshold):
    print('Sensitivity:', tpr[thresholds > threshold][-1])
    print('Specificity:', 1 - fpr[thresholds > threshold][-1])
```

In [88]:
```
evaluate_threshold(0.5)
```

```
Sensitivity: 0.5925925925925926
Specificity: 0.9066666666666666
```

In [89]:
```
evaluate_threshold(0.3)
```

```
Sensitivity: 0.7530864197530864
Specificity: 0.6799999999999999
```

In [90]:
```
evaluate_threshold(0.2)
```

```
Sensitivity: 0.8888888888888888
Specificity: 0.5666666666666667
```

In [91]:
```
evaluate_threshold(0.6)
```

```
Sensitivity: 0.5308641975308642
Specificity: 0.92
```

AUC is the percentage of the ROC plot that is underneath the curve:

In [92]:
```
print(metrics.roc_auc_score(y_test, y_pred_prob))
```

```
0.8354732510288065
```

In [93]:
```
from sklearn.model_selection import cross_val_score
cross_val_score(logreg, X, y, cv=10, scoring='roc_auc').mean()
```

```
C:\Users\pakalamanda.harish\Anaconda3\lib\site-packages\sklearn\linear_model
\_logistic.py:762: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regre
ssion
  n_iter_i = _check_optimize_result(
C:\Users\pakalamanda.harish\Anaconda3\lib\site-packages\sklearn\linear_model
\_logistic.py:762: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regre
ssion
  n_iter_i = _check_optimize_result(
C:\Users\pakalamanda.harish\Anaconda3\lib\site-packages\sklearn\linear_model
\_logistic.py:762: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regre
ssion
  n_iter_i = _check_optimize_result(
C:\Users\pakalamanda.harish\Anaconda3\lib\site-packages\sklearn\linear_model
\_logistic.py:762: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regre
ssion
  n_iter_i = _check_optimize_result(
C:\Users\pakalamanda.harish\Anaconda3\lib\site-packages\sklearn\linear_model
\_logistic.py:762: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regre
ssion
  n_iter_i = _check_optimize_result(
C:\Users\pakalamanda.harish\Anaconda3\lib\site-packages\sklearn\linear_model
\_logistic.py:762: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regre
ssion
  n_iter_i = _check_optimize_result(
C:\Users\pakalamanda.harish\Anaconda3\lib\site-packages\sklearn\linear_model
\_logistic.py:762: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

```
Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regre
ssion
  n_iter_i = _check_optimize_result(
C:\Users\pakalamanda.harish\Anaconda3\lib\site-packages\sklearn\linear_model
\_logistic.py:762: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regre
ssion
  n_iter_i = _check_optimize_result(
C:\Users\pakalamanda.harish\Anaconda3\lib\site-packages\sklearn\linear_model
\_logistic.py:762: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regre
ssion
  n_iter_i = _check_optimize_result(
C:\Users\pakalamanda.harish\Anaconda3\lib\site-packages\sklearn\linear_model
\_logistic.py:762: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regre
ssion
  n_iter_i = _check_optimize_result(
```

Out[93]: 0.8292250712250713

In [ ]: