Importing the Libraries

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler

from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier

from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score, recall_score, f1_score

from imblearn.over_sampling import RandomOverSampler
from imblearn.under_sampling import RandomUnderSampler
from collections import Counter

plt.style.use('fivethirtyeight')

import warnings
warnings.filterwarnings('ignore')
```

Reading & Exploring the dataset

In [2]:
```
df = pd.read_csv('/kaggle/input/diabetes-dataset/diabetes.csv')
df
```

Out[2]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outc
0	6	148	72	35	0	33.6	0.627	50	
1	1	85	66	29	0	26.6	0.351	31	
2	8	183	64	0	0	23.3	0.672	32	
3	1	89	66	23	94	28.1	0.167	21	
4	0	137	40	35	168	43.1	2.288	33	
...	...	...	...	...	...	...	...	...	
763	10	101	76	48	180	32.9	0.171	63	
764	2	122	70	27	0	36.8	0.340	27	
765	5	121	72	23	112	26.2	0.245	30	
766	1	126	60	0	0	30.1	0.349	47	
767	1	93	70	31	0	30.4	0.315	23	

768 rows × 9 columns

In [3]:
```
df.isna().sum()
```

```
Pregnancies                 0
```

Out[3]:
```
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

In [4]:
```python
df.duplicated().sum()
```

Out[4]:
```
0
```

In [5]:
```python
df.info()
```

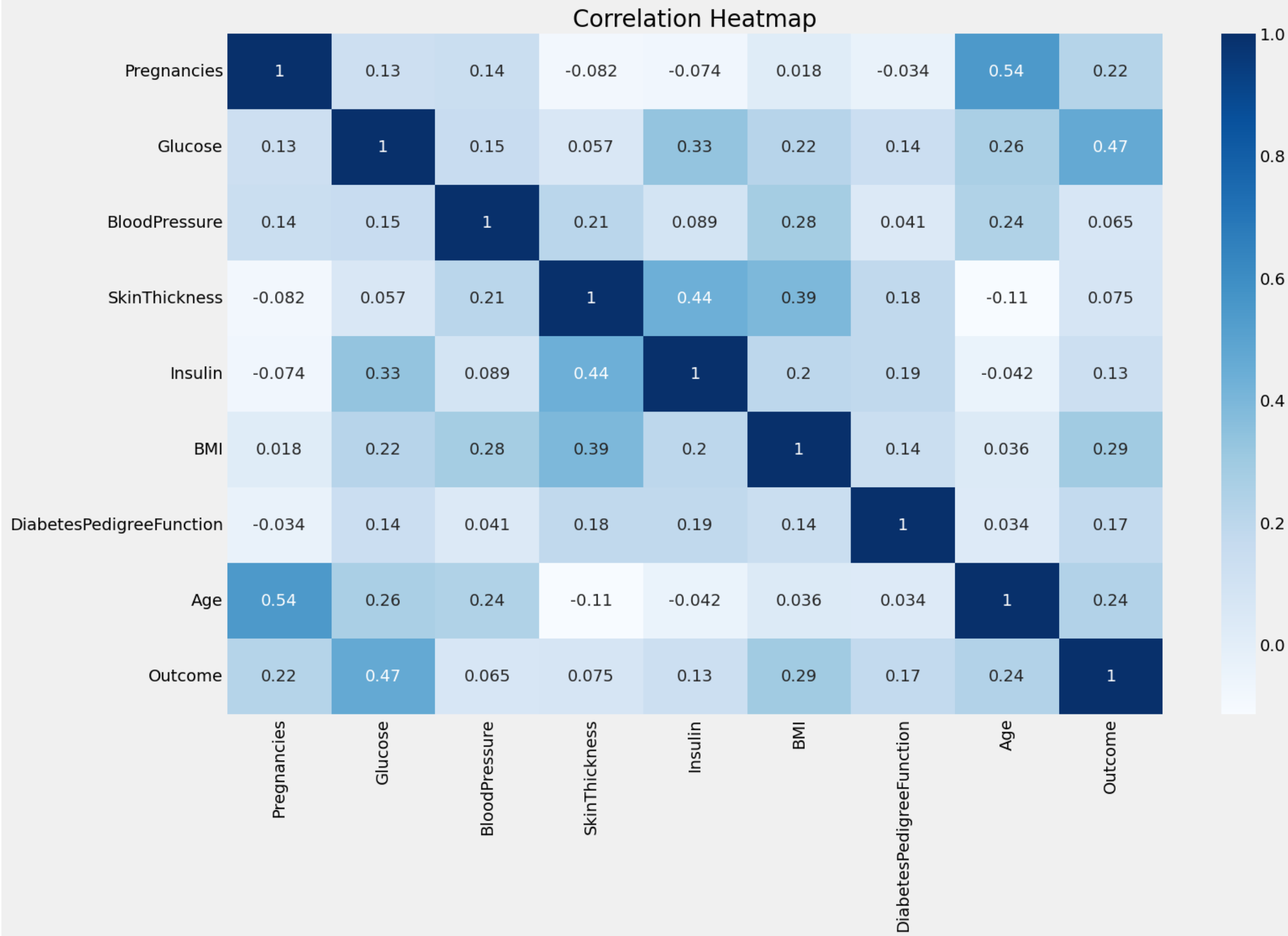
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Pregnancies               768 non-null    int64  
 1   Glucose               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

In [6]:
```python
# Showing the Correlations between colomns
df.corr()
```

Out[6]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Diabetes
Pregnancies	1.000000	0.129459	0.141282	-0.081672	-0.073535	0.017683	
Glucose	0.129459	1.000000	0.152590	0.057328	0.331357	0.221071	
BloodPressure	0.141282	0.152590	1.000000	0.207371	0.088933	0.281805	
SkinThickness	-0.081672	0.057328	0.207371	1.000000	0.436783	0.392573	
Insulin	-0.073535	0.331357	0.088933	0.436783	1.000000	0.197859	
BMI	0.017683	0.221071	0.281805	0.392573	0.197859	1.000000	
DiabetesPedigreeFunction	-0.033523	0.137337	0.041265	0.183928	0.185071	0.140647	
Age	0.544341	0.263514	0.239528	-0.113970	-0.042163	0.036242	
Outcome	0.221898	0.466581	0.065068	0.074752	0.130548	0.292695	

In [7]:
```python
# Ploting the correlation
plt.figure(figsize=(16, 10))
sns.heatmap(df.corr(), annot=True, cmap='Blues')
plt.title("Correlation Heatmap")
plt.show()
```



We can notice that the Glucose colomn highly affect the outcome.
the BMI and Age colomns may affect also the Outcome.
For the rest of the colomns they rarely affect the Outcome.

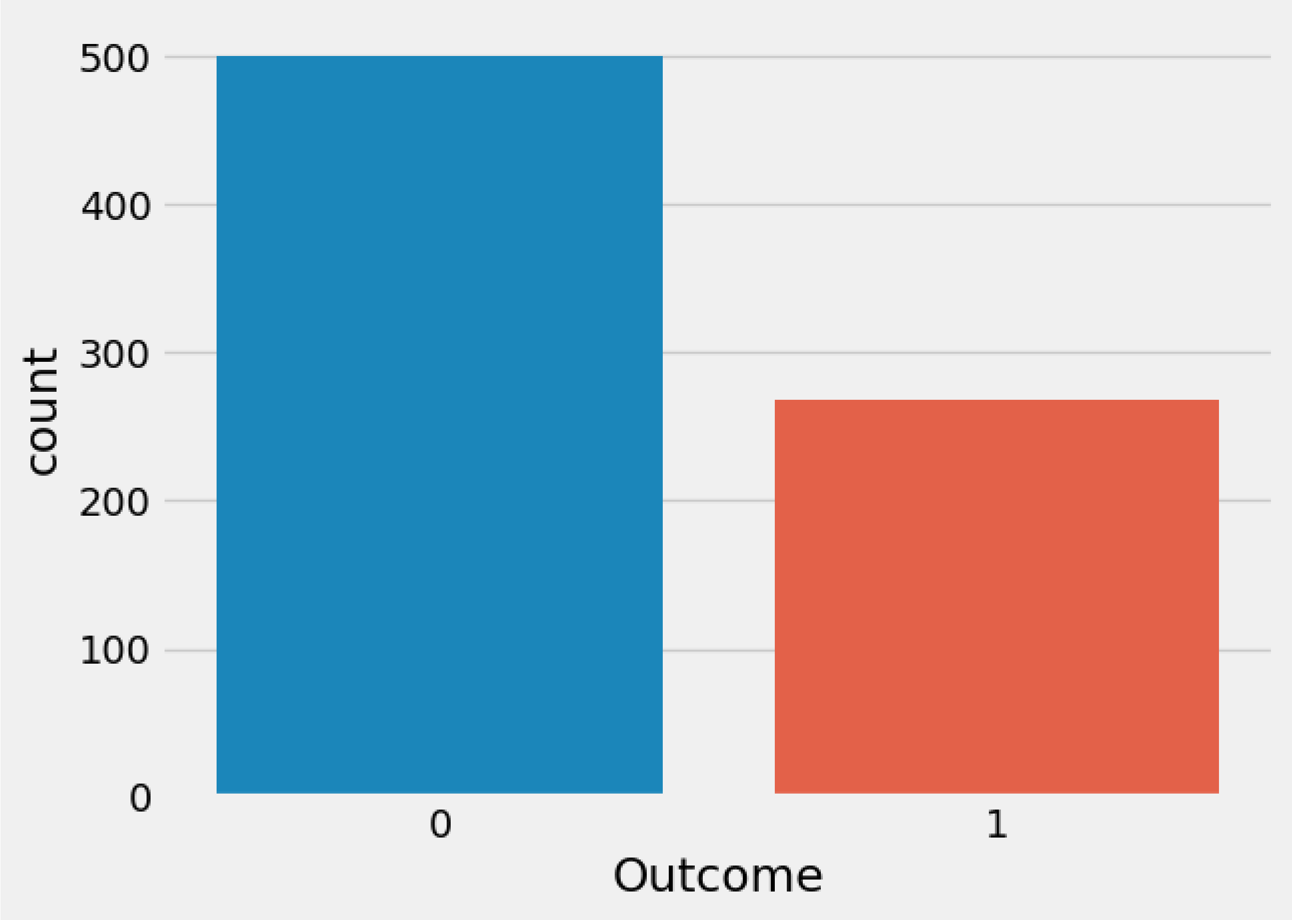
EDA

In [8]:
```
df['Outcome'].value_counts()
```

Out[8]:
```
Outcome
0    500
1    268
Name: count, dtype: int64
```

In [9]:
```
sns.countplot(x='Outcome', data = df)
```

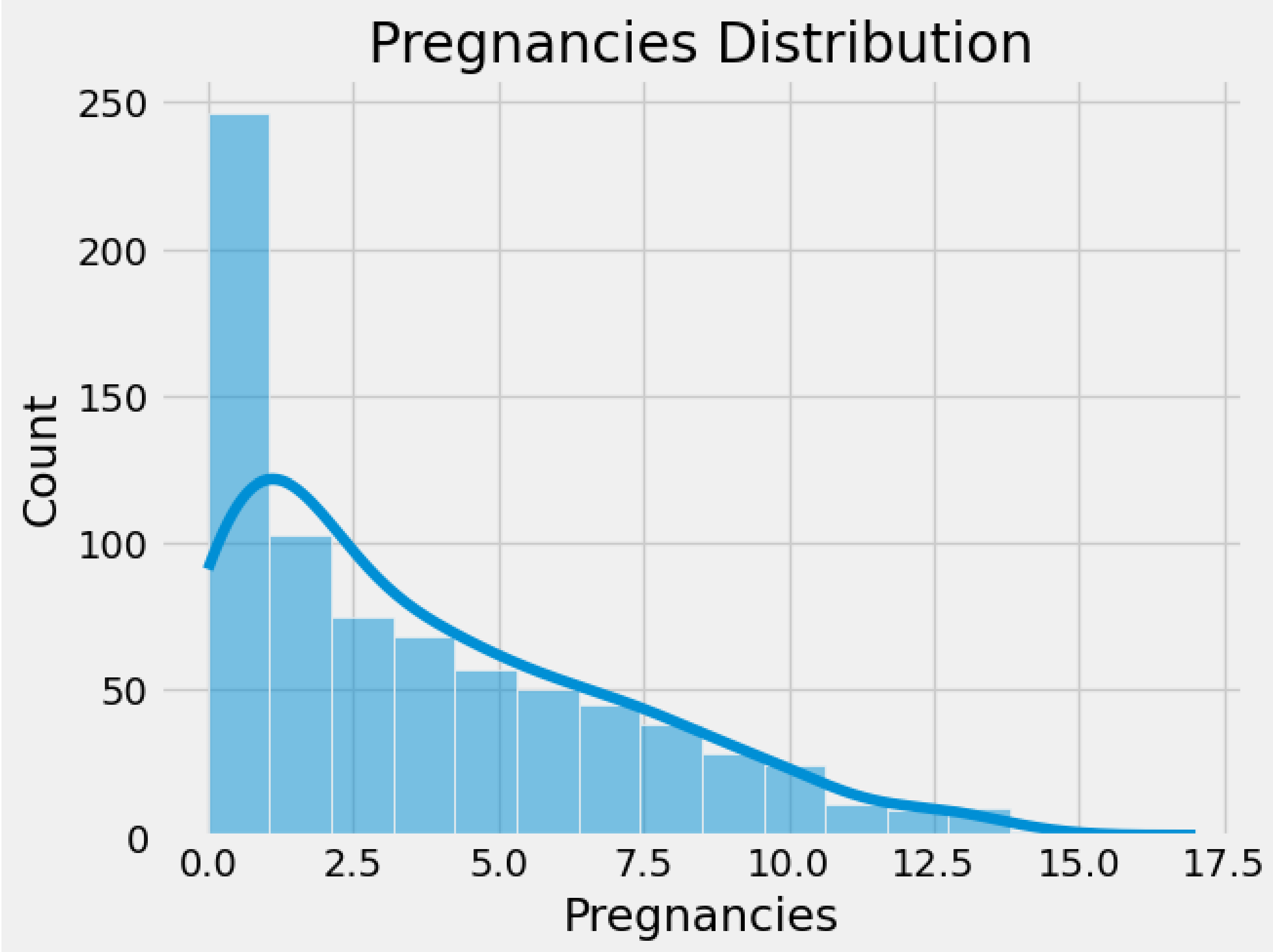
Out[9]:
```
<Axes: xlabel='Outcome', ylabel='count'>
```

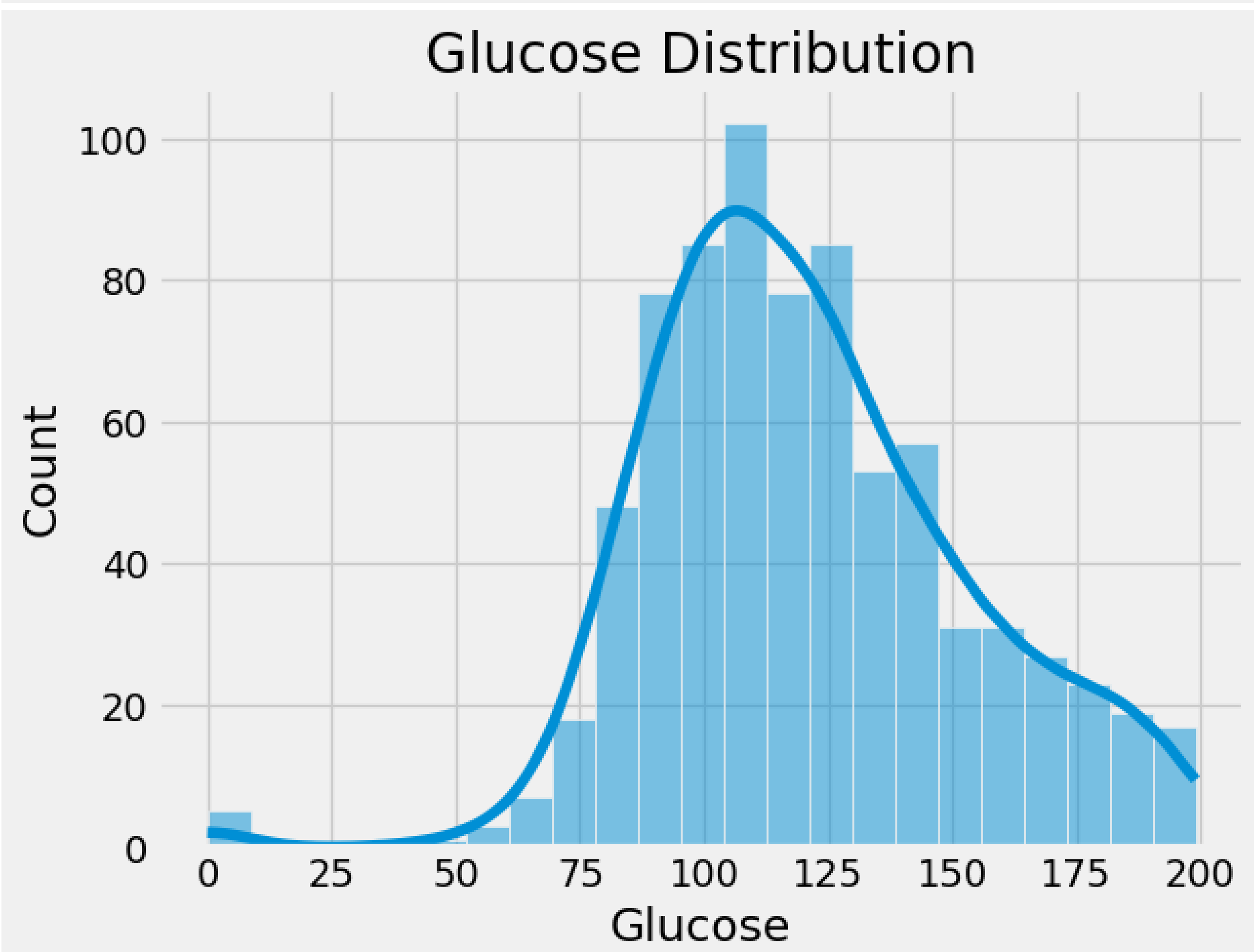


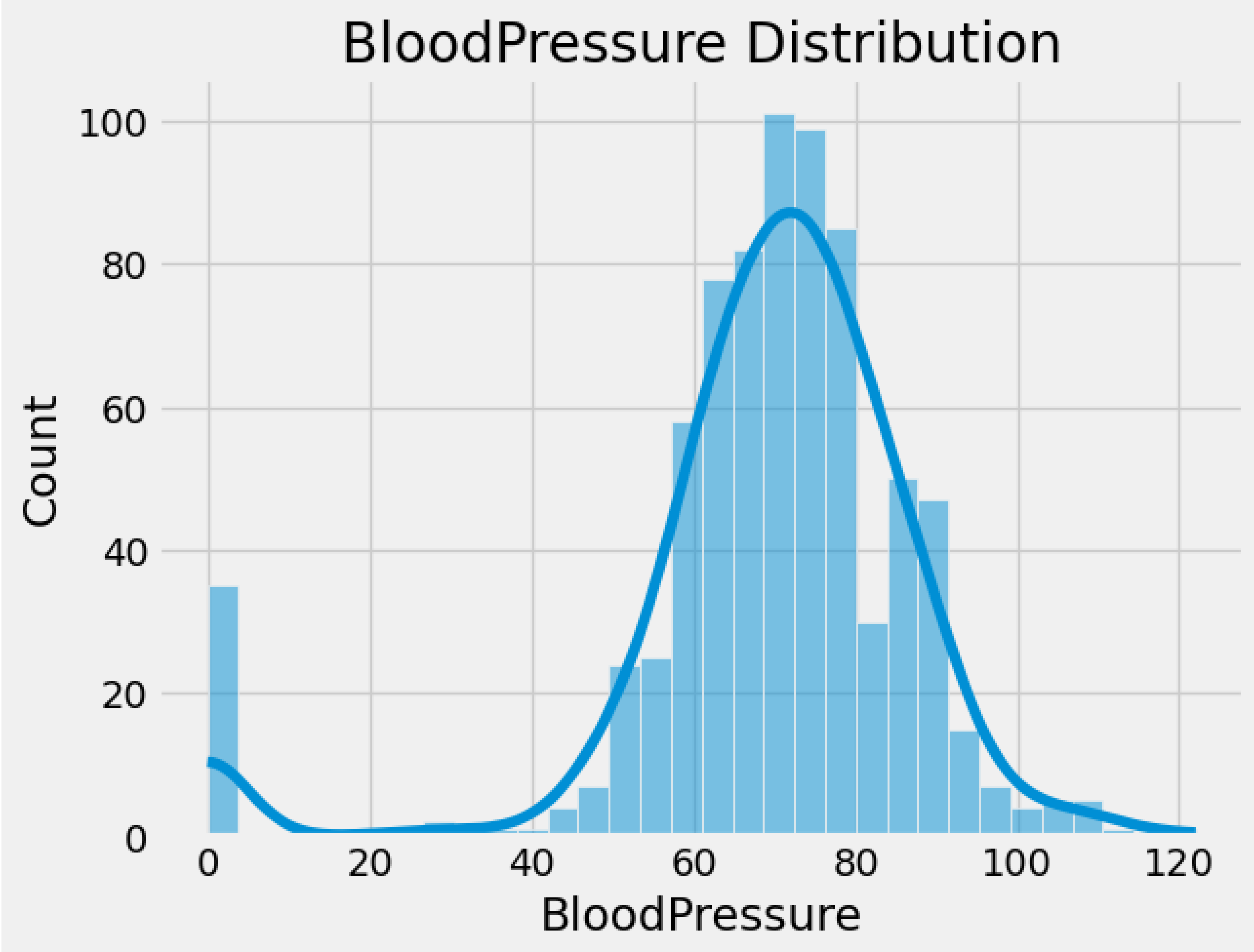
In [10]:
```
numerical_columns = df.select_dtypes(exclude=object).columns.tolist()
```

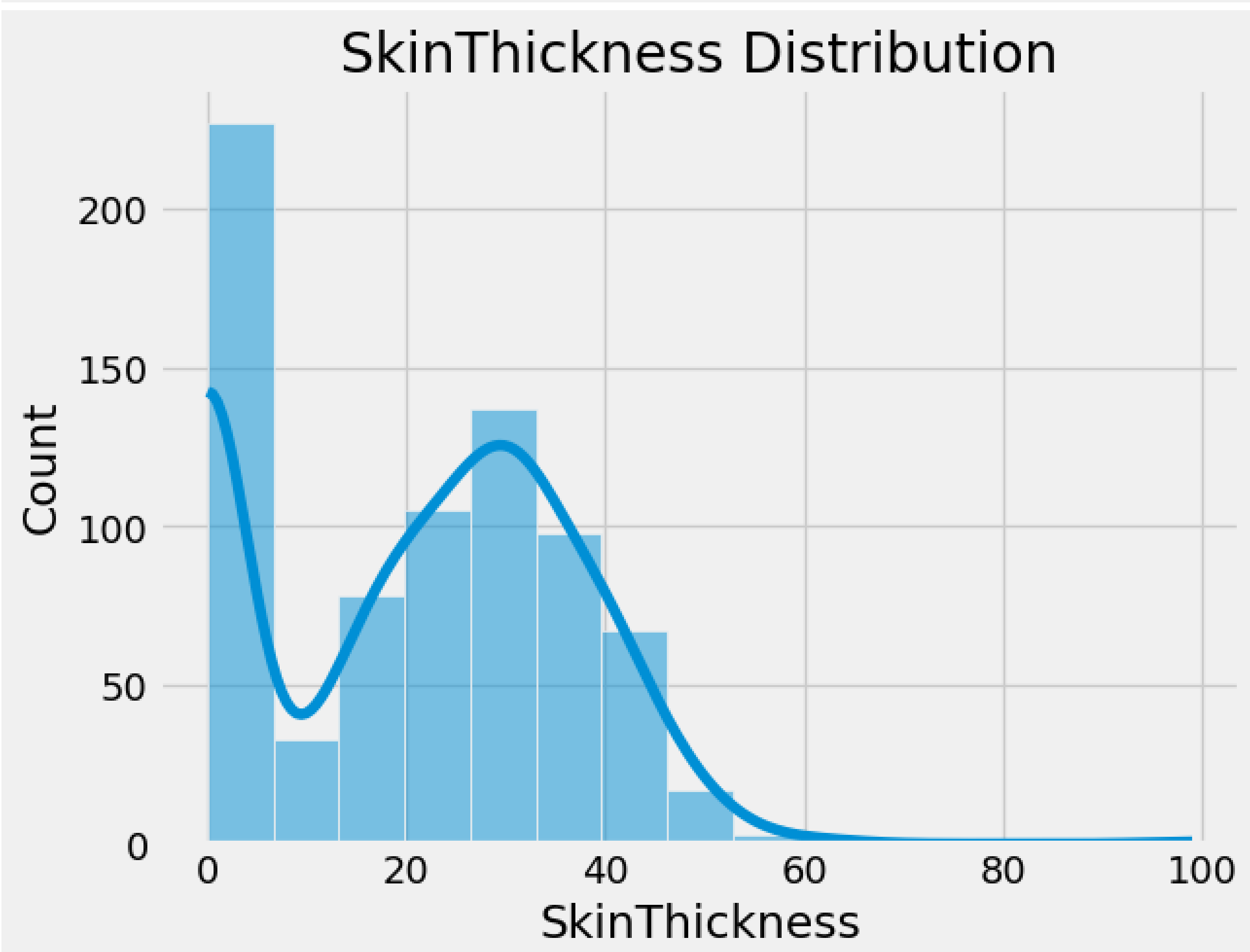
In [11]:
```
def num_cols_vis(col):
    sns.histplot(data=df, x=col, kde=True)
    plt.title(f'{col} Distribution')
    plt.show()
```

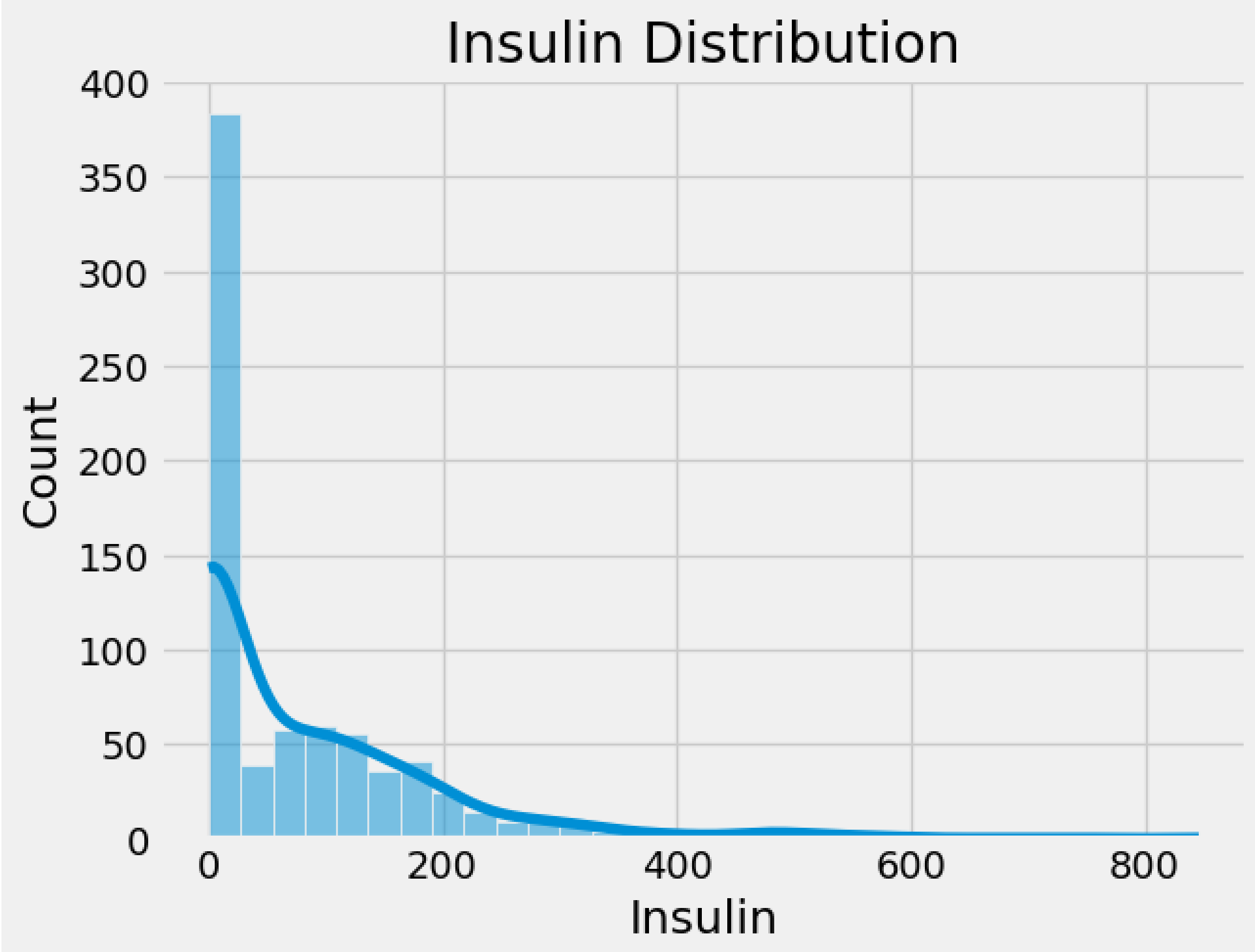
In [12]:
```
for col in numerical_columns:
    num_cols_vis(col)
```

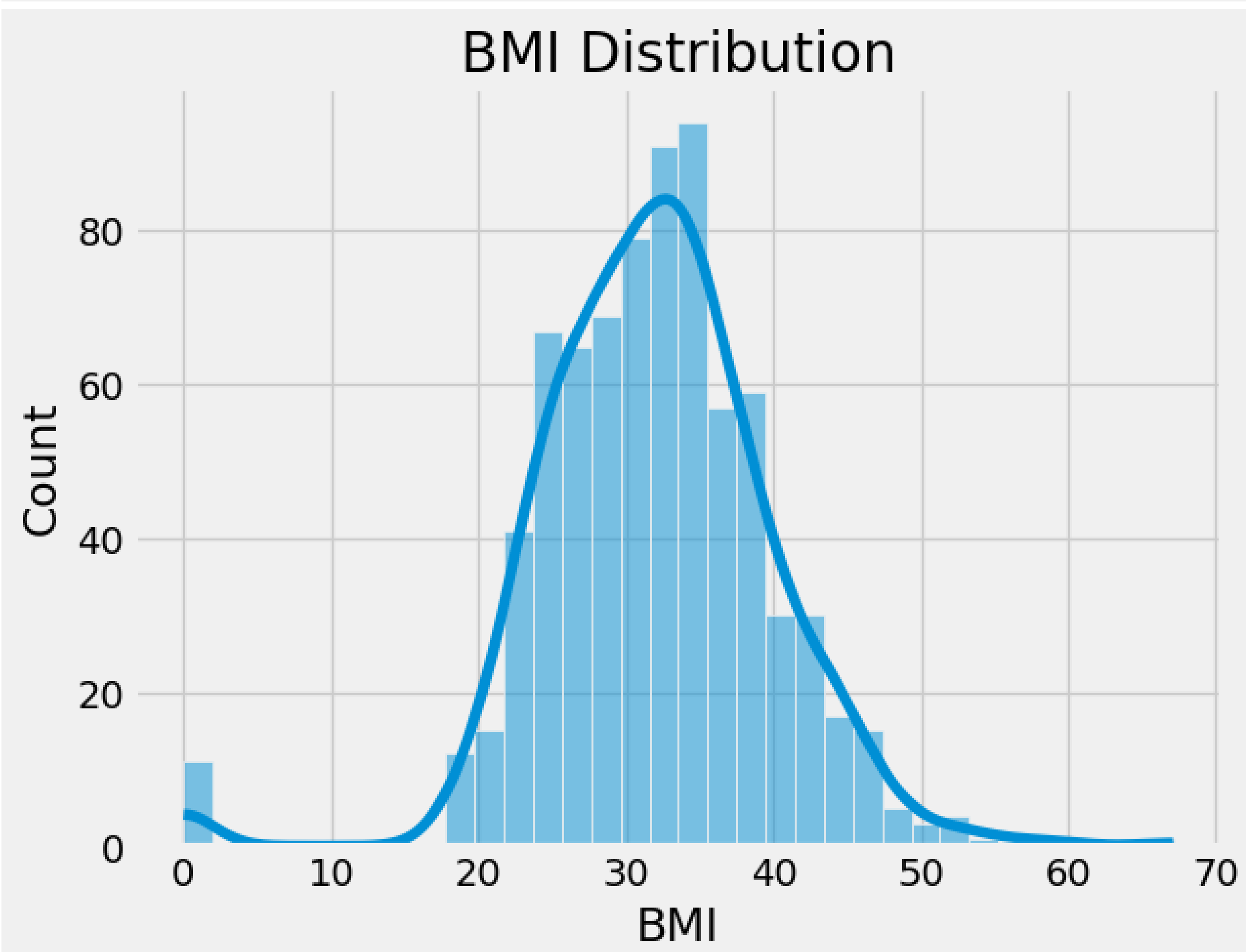

Pregnancies Distribution
Count
250
200
150
100
50
0
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Pregnancies

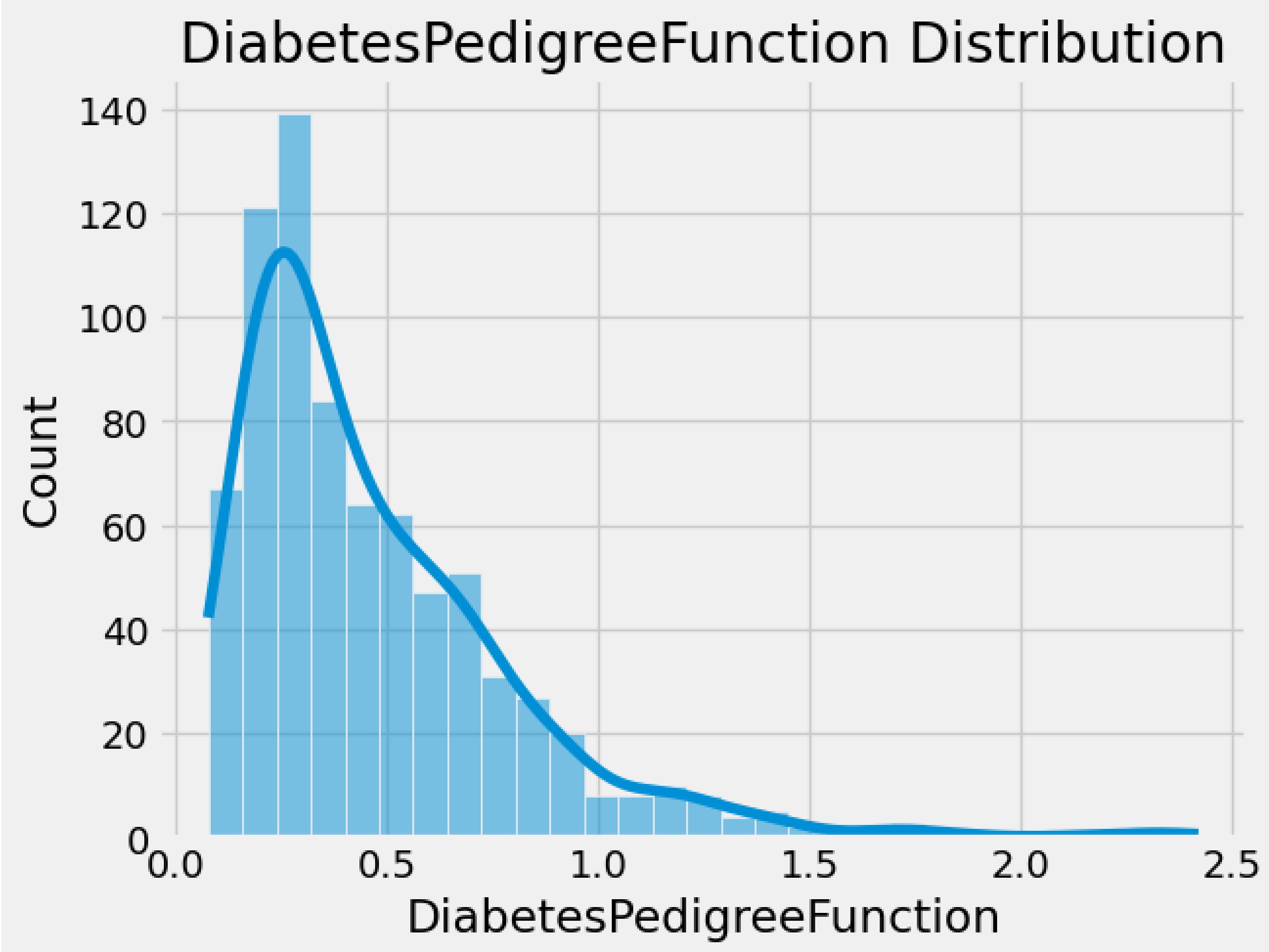

Glucose Distribution
Count
100
80
60
40
20
0
0
25
50
75
100
125
150
175
200
Glucose

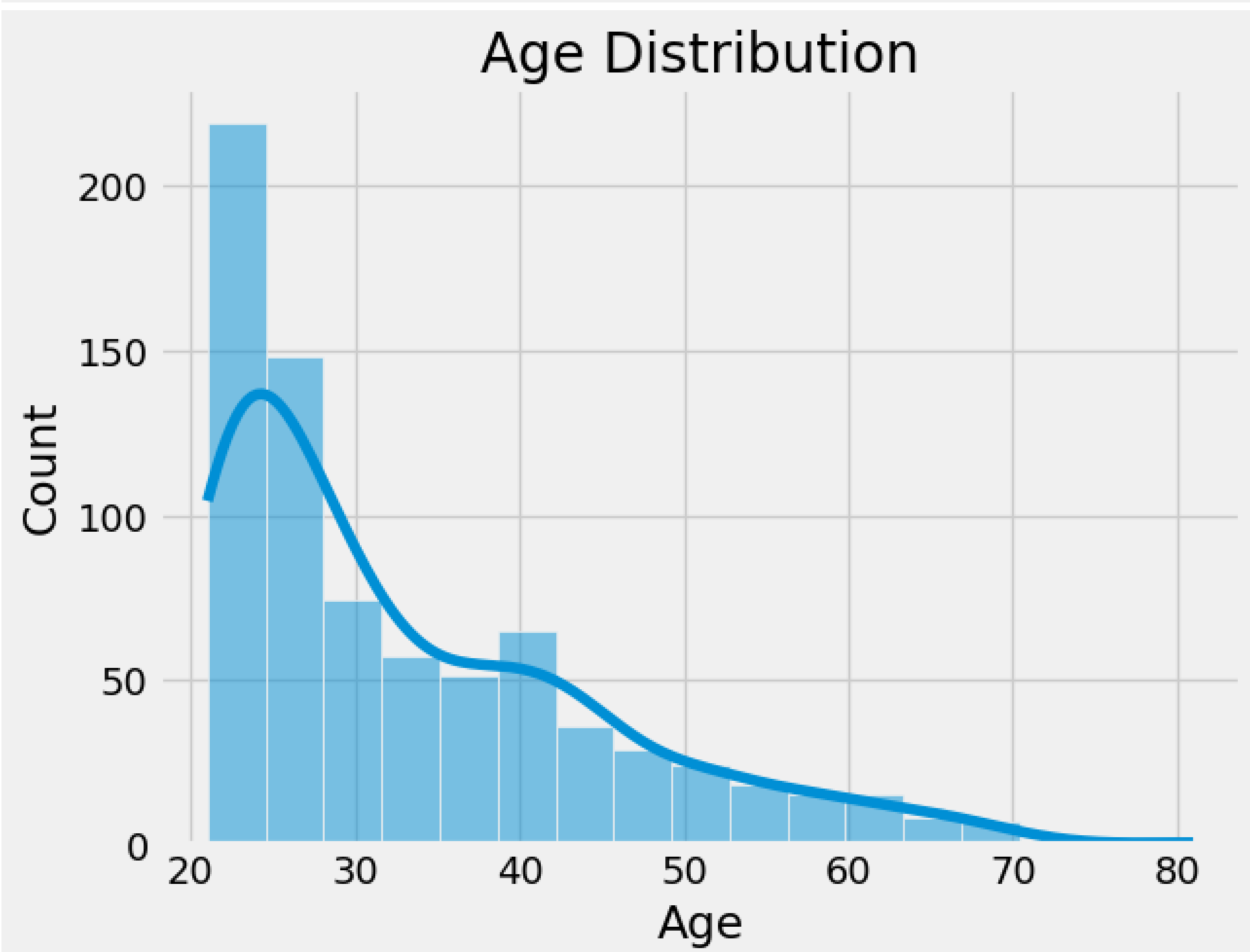

BloodPressure Distribution
Count
0
20
40
60
80
100
0
20
40
60
80
100
120
BloodPressure

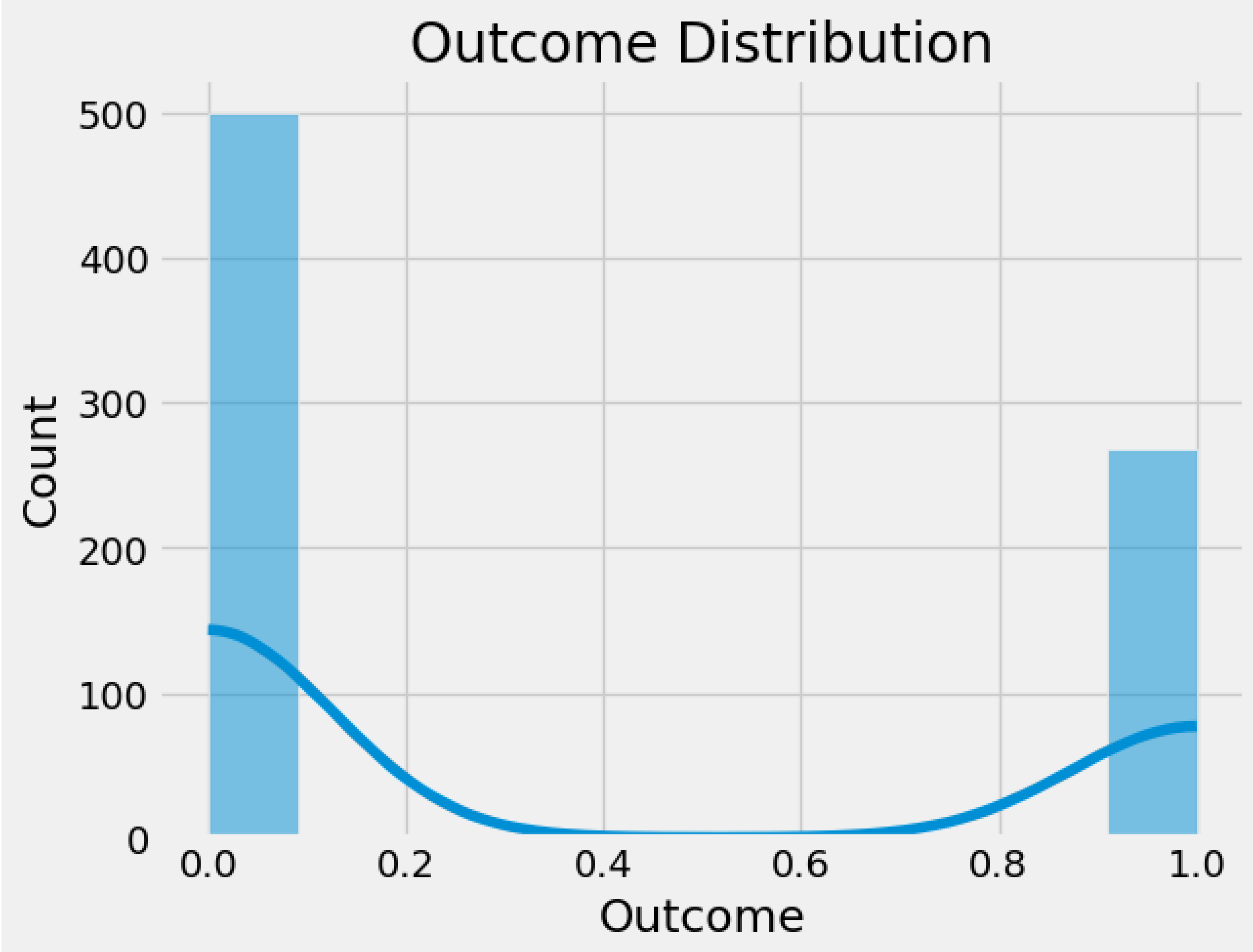

SkinThickness Distribution
Count
0
50
100
150
200
0
20
40
60
80
100
SkinThickness


Insulin Distribution
Count
Insulin
0
50
100
150
200
250
300
350
400
0
200
400
600
800


BMI Distribution
Count
BMI
0
20
40
60
80
0
10
20
30
40
50
60
70


DiabetesPedigreeFunction Distribution
Count
DiabetesPedigreeFunction
0
20
40
60
80
100
120
140
0.0
0.5
1.0
1.5
2.0
2.5


Age Distribution
Count
Age
0
50
100
150
200
20
30
40
50
60
70
80



Spliting the data

In [13]:
```
x = df.drop('Outcome', axis= 1)
y = df['Outcome']
```

In [14]:
```
rm = RandomOverSampler(random_state=41)
x_res,y_res = rm.fit_resample(x,y)
```

In [15]:
```
x_train,x_test,y_train,y_test = train_test_split(x_res,y_res,test_size= 0.2)
```

Building the Models & Evaluation

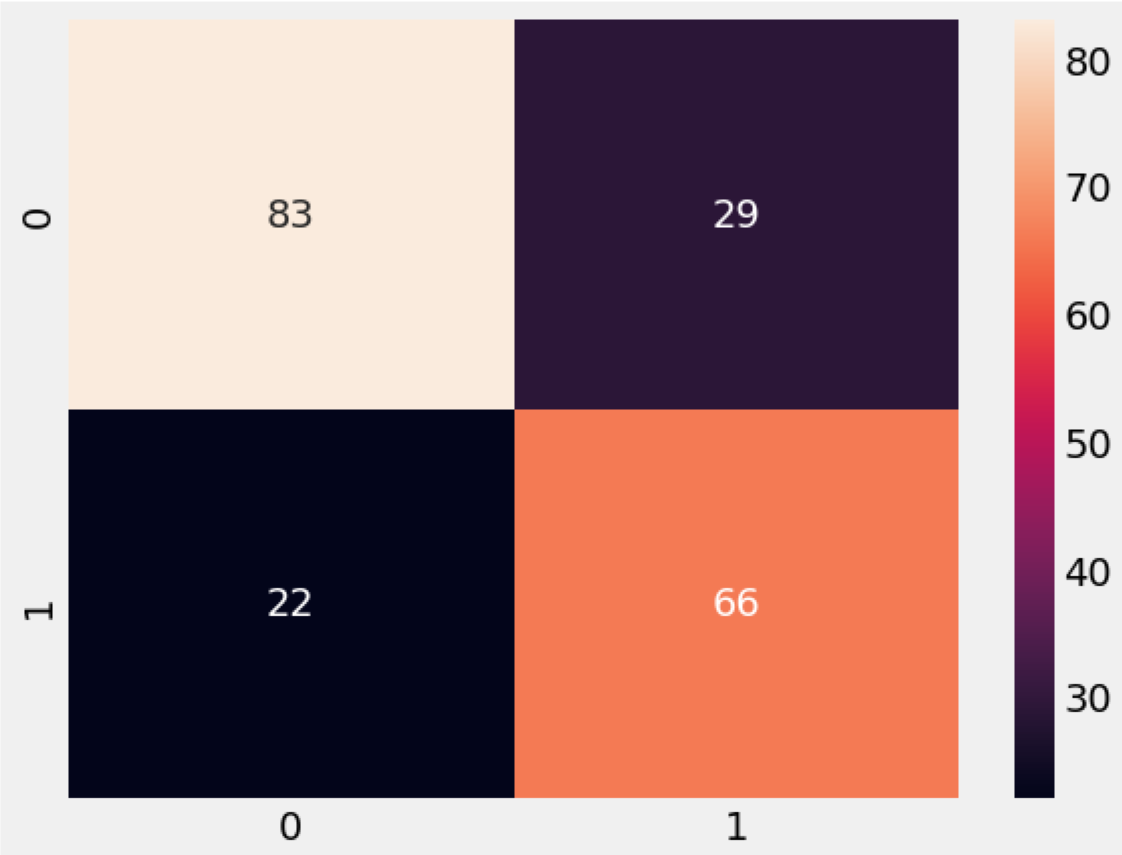
In [16]:
```
model_1 = LogisticRegression()
model_2 = SVC()
model_3 = RandomForestClassifier(n_estimators= 100,class_weight= 'balanced')
model_4 = GradientBoostingClassifier(n_estimators=1000)
```

In [17]:
```
col = ['LogisticRegression','SVC','RandomForestClassifier','GradientBoostingClassifier']
result_1 = []
result_2 = []
result_3 = []
```

In [18]:
```
def cal(model):
    model.fit(x_train,y_train)
    pre = model.predict(x_test)
    accuracy = accuracy_score(pre,y_test)
    recall = recall_score(pre,y_test)
```

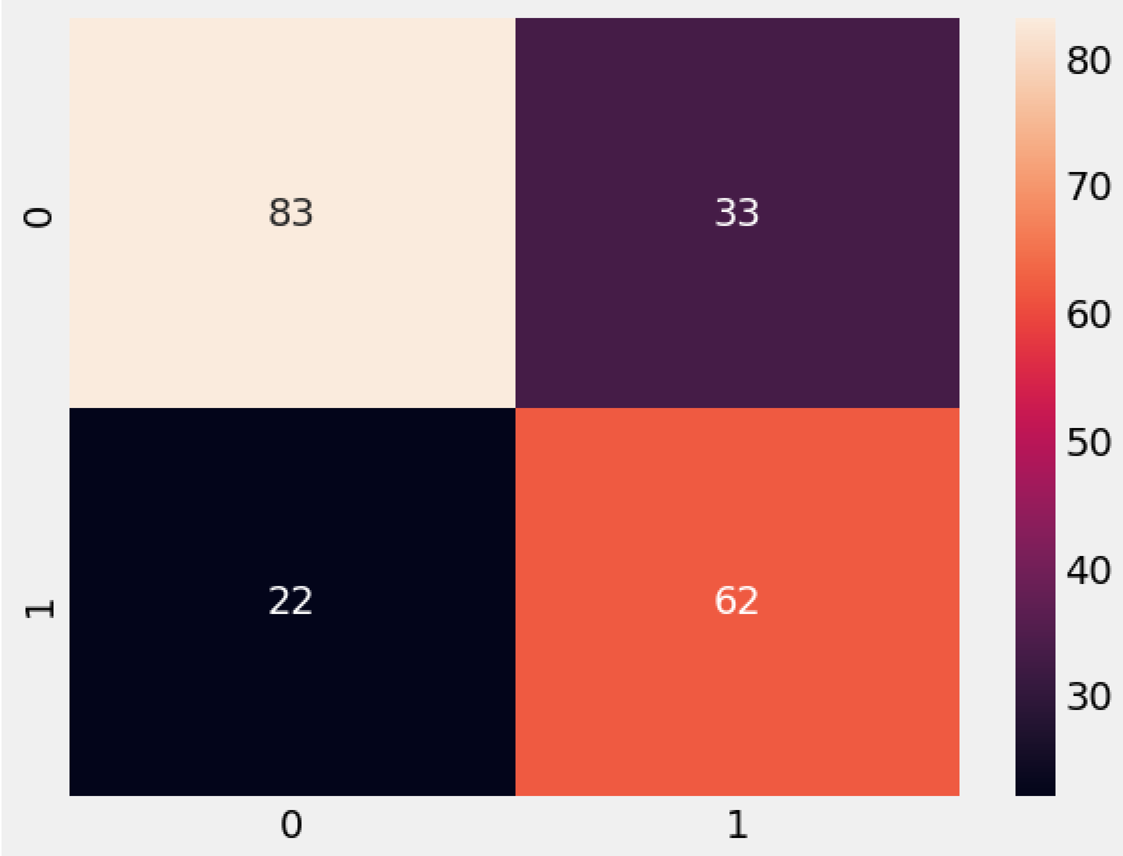
```python
    f1 = f1_score(pre,y_test)

    result_1.append(accuracy)
    result_2.append(recall)
    result_3.append(f1)

    sns.heatmap(confusion_matrix(pre,y_test),annot=True)
    print(model)
    print('Accuracy is: ',accuracy,'Recall is: ',recall,"F1 is: ",f1)
cal(model_1)
```

```
LogisticRegression()
Accuracy is:  0.745 Recall is:  0.75 F1 is:  0.7213114754098362
```



In [19]:
```python
cal(model_2)
```

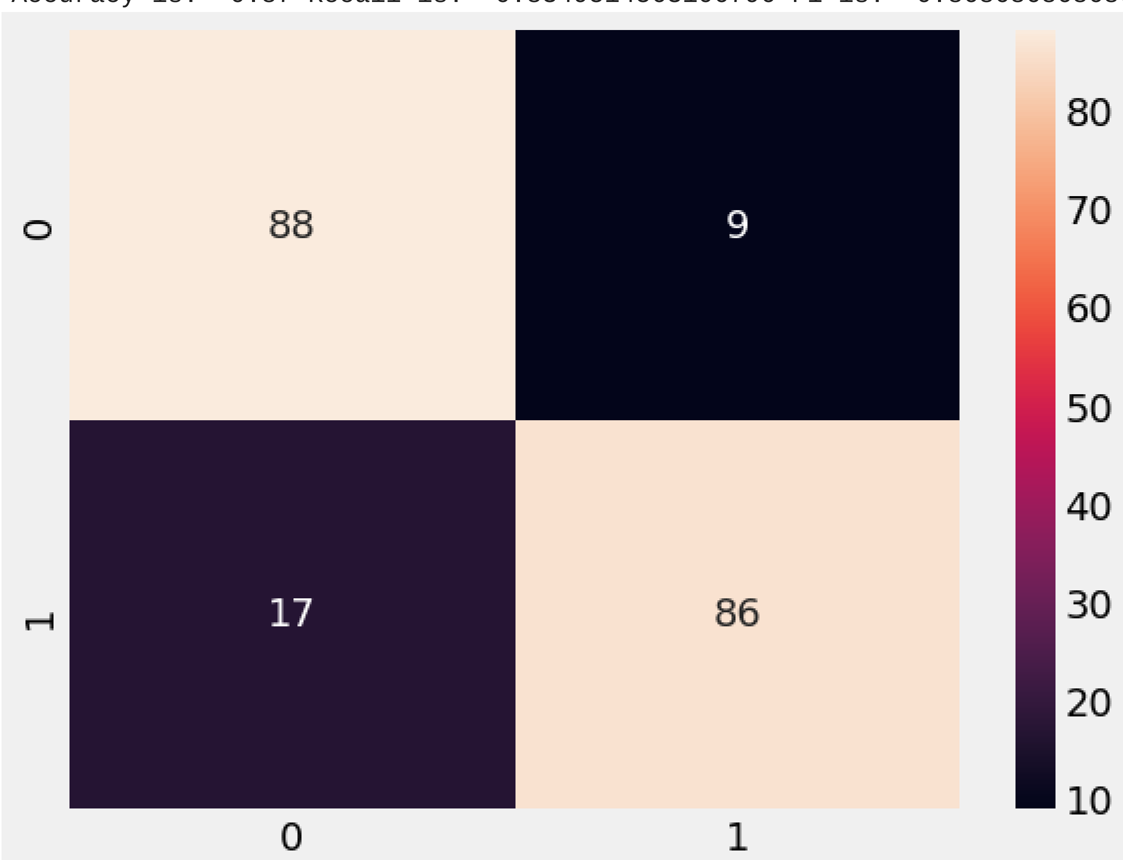
```
SVC()
Accuracy is:  0.725 Recall is:  0.7380952380952381 F1 is:  0.6927374301675978
```



In [20]:

```
cal(model_3)
```

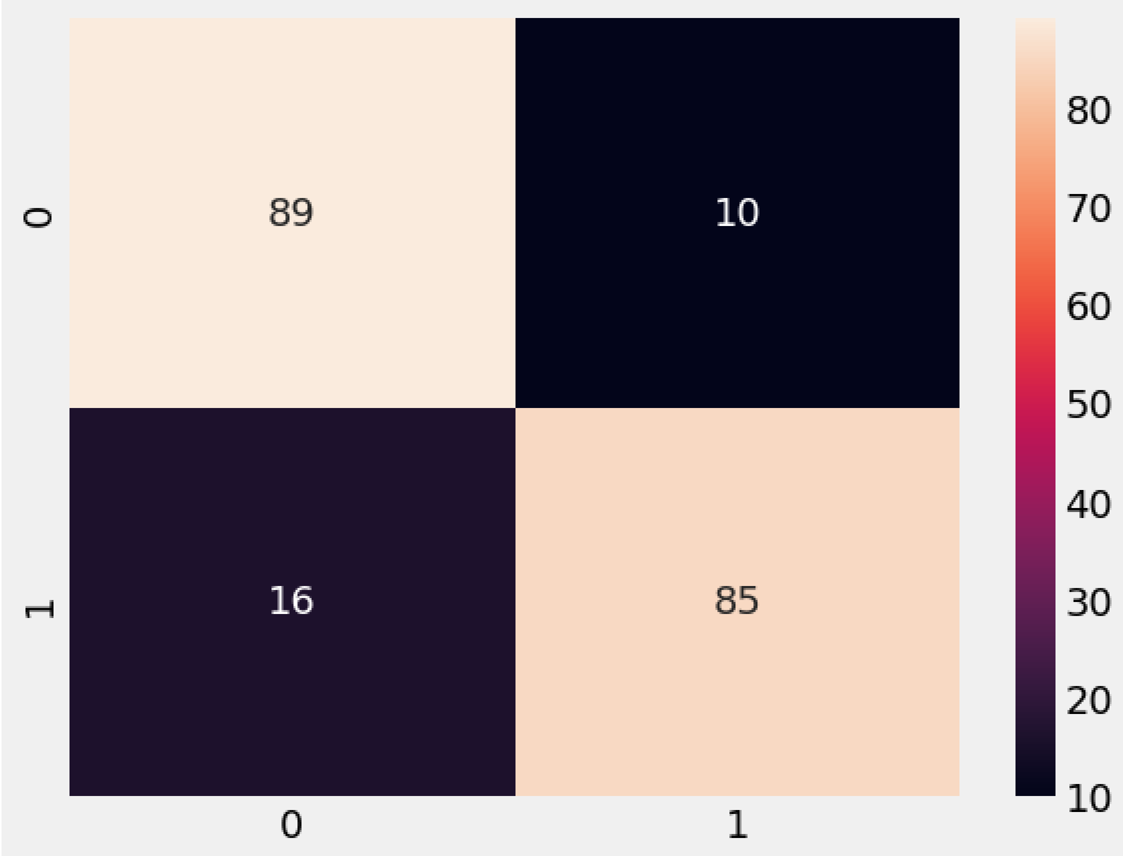
```
RandomForestClassifier(class_weight='balanced')
Accuracy is:  0.87 Recall is:  0.8349514563106796 F1 is:  0.8686868686868687
```



In [21]:

```
cal(model_4)
```

```
GradientBoostingClassifier(n_estimators=1000)
Accuracy is:  0.87 Recall is:  0.8415841584158416 F1 is:  0.8673469387755102
```



In [22]:
```
final_result = pd.DataFrame({"Algorithm":col ,'Accuarcy':result_1,"recall":result_2,"F1_
final_result
```

Out[22]:

	Algorithm	Accuarcy	recall	F1_score
0	LogisticRegression	0.745	0.750000	0.721311
1	SVC	0.725	0.738095	0.692737
2	RandomForestClassifier	0.870	0.834951	0.868687
3	GradientBoostingClassifier	0.870	0.841584	0.867347

In [23]:
```
# Performance Comparison of Classification Metrics Across Algorithms
fig,ax = plt.subplots(figsize=(15,5))
plt.plot(final_result.Algorithm,result_1,label="Accuracy")
plt.plot(final_result.Algorithm,result_2,label="Recall")
plt.plot(final_result.Algorithm,result_3,label="F1_Score")
plt.legend()
plt.show()
```