Diabetes Prediction using classification method

Import Libraries

In [84]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
```

Load Datasets

In [85]:
```
data = pd.read_csv("diabetes.csv")
```

Shape of Data

In [86]:
```
data.shape
```

Out[86]:
```
(768, 9)
```

First 8 Rows of Data

In [87]:
```
data.head(8)
```

Out[87]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunctior
0	6	148	72	35	0	33.6	0.627
1	1	85	66	29	0	26.6	0.351
2	8	183	64	0	0	23.3	0.672
3	1	89	66	23	94	28.1	0.167
4	0	137	40	35	168	43.1	2.288
5	5	116	74	0	0	25.6	0.201
6	3	78	50	32	88	31.0	0.248
7	10	115	0	0	0	35.3	0.134

Last 7 Rows of Data

In [88]:
```
data.tail(7)
```

Out[88]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFuncti
761	9	170	74	31	0	44.0	0.4
762	9	89	62	0	0	22.5	0.1
763	10	101	76	48	180	32.9	0.1
764	2	122	70	27	0	36.8	0.3
765	5	121	72	23	112	26.2	0.2
766	1	126	60	0	0	30.1	0.3
767	1	93	70	31	0	30.4	0.3

In [89]:
```
data.describe()
```

Out[89]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Diabete
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	

In [90]:
```
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 [91]:
```
data.value_counts()
```

Out[91]:
```
Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin  BMI   DiabetesP
edigreeFunction  Age  Outcome
0            57       60             0              0        21.7  0.735
67   0          1
             67       76             0              0        45.3  0.194
46   0          1
5            103      108            37             0        39.2  0.305
65   0          1
             104      74             0              0        28.8  0.153
48   0          1
             105      72             29             325      36.9  0.159
28   0          1


..
2            84       50             23             76       30.4  0.968
21   0          1
             85       65             0              0        39.6  0.930
27   0          1
             87       0              23             0        28.9  0.773
25   0          1
                      58             16             52       32.7  0.166
25   0          1
17           163      72             41             114      40.9  0.817
47   1          1
Name: count, Length: 768, dtype: int64
```

In [92]:
```
data.columns
```

Out[92]:
```
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insuli
n',
       'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
      dtype='object')
```

Checking Null Values

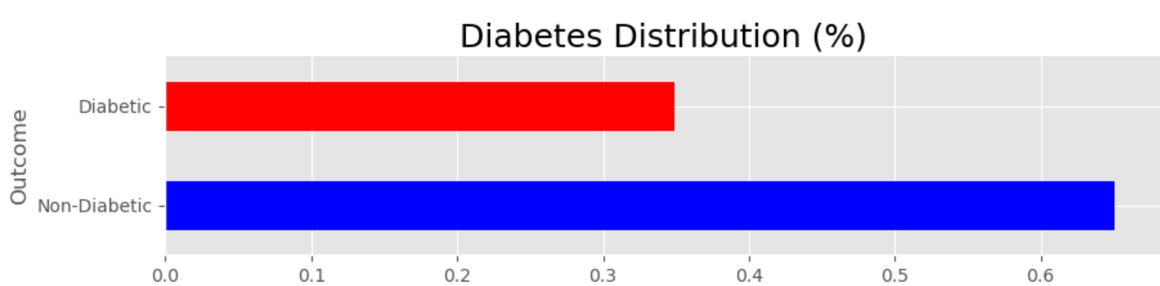
In [93]:
```
data.isnull().sum()
```

Out[93]:
```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

Diabetes Distribution

In [94]:
```
#Finding Class Distribution Percentage
print(data['Outcome'].value_counts(ascending=True))
print(data['Outcome'].value_counts(1,ascending=True).apply(lambda x: format(x
print()
# Plot the bar chart
data['Outcome'].value_counts(normalize=True).plot(kind='barh',figsize=(10, 2)
plt.title('Diabetes Distribution (%)', fontsize=18)
plt.yticks(ticks=[0,1], labels=['Non-Diabetic', 'Diabetic'])
plt.show()
```

```
Outcome
1    268
0    500
Name: count, dtype: int64
Outcome
1    34.895833%
0    65.104167%
Name: proportion, dtype: object
```



Exploratory Data Analysis

In [95]:
```
data.corr()
```

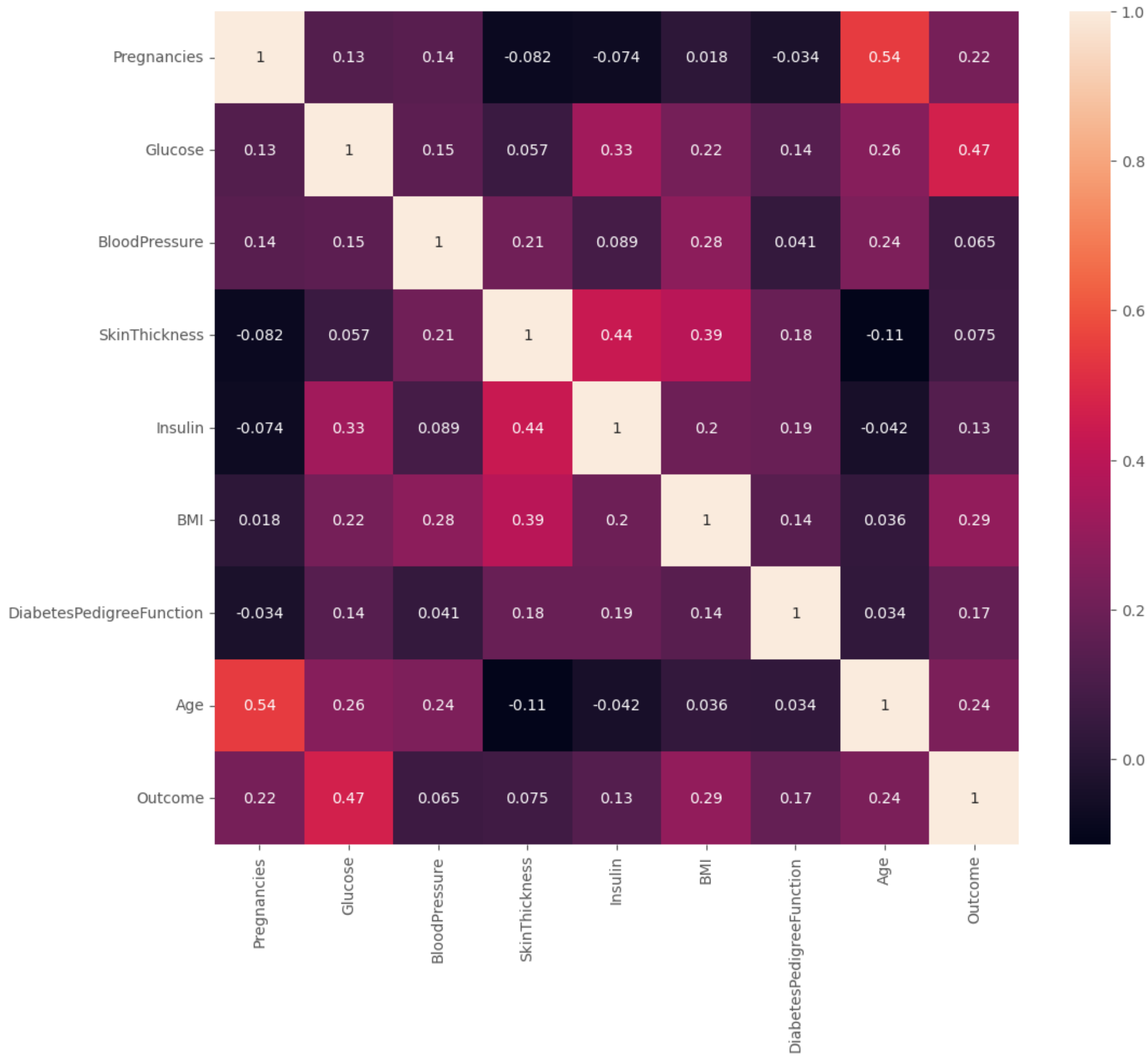
Out[95]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	
Pregnancies	1.000000	0.129459	0.141282	-0.081672	-0.073535	0.01
Glucose	0.129459	1.000000	0.152590	0.057328	0.331357	0.22
BloodPressure	0.141282	0.152590	1.000000	0.207371	0.088933	0.28
SkinThickness	-0.081672	0.057328	0.207371	1.000000	0.436783	0.39
Insulin	-0.073535	0.331357	0.088933	0.436783	1.000000	0.19
BMI	0.017683	0.221071	0.281805	0.392573	0.197859	1.00
DiabetesPedigreeFunction	-0.033523	0.137337	0.041265	0.183928	0.185071	0.14
Age	0.544341	0.263514	0.239528	-0.113970	-0.042163	0.03
Outcome	0.221898	0.466581	0.065068	0.074752	0.130548	0.29

Correlation Matrix

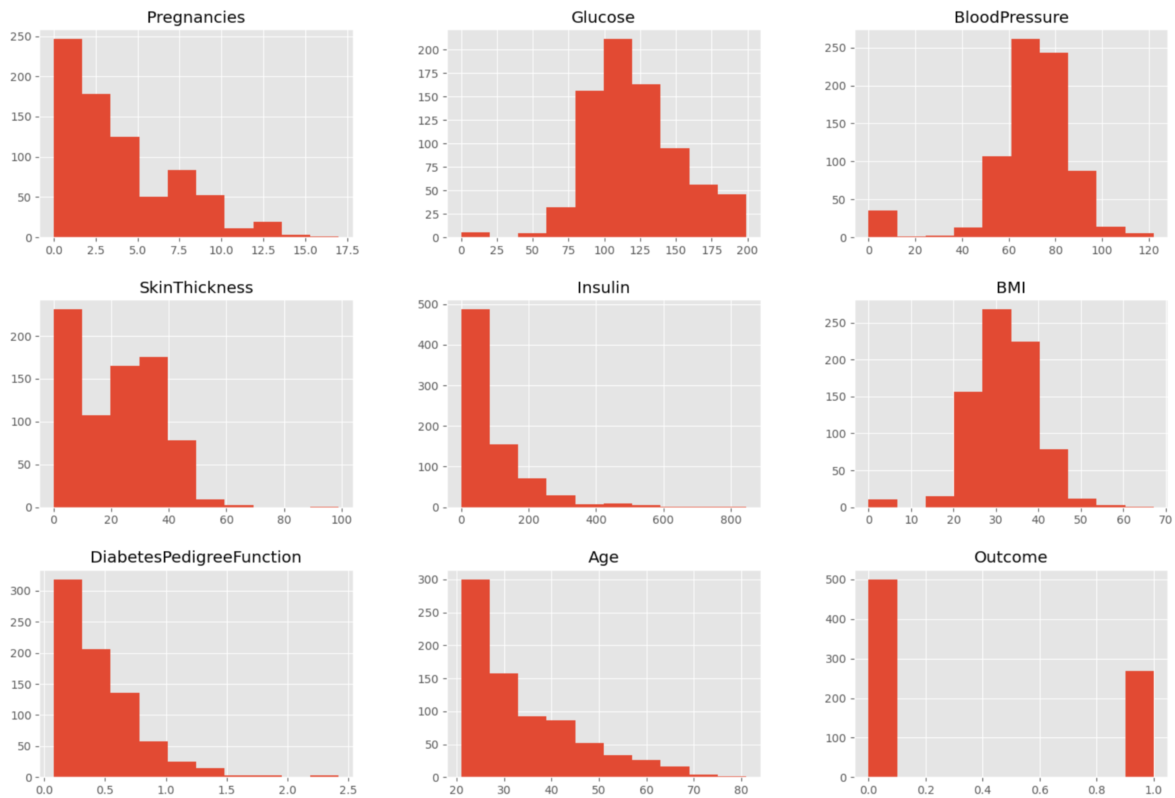
In [96]:
```
plt.figure(figsize = (12,10))

sns.heatmap(data.corr(), annot =True)
```

Out[96]: <Axes: >

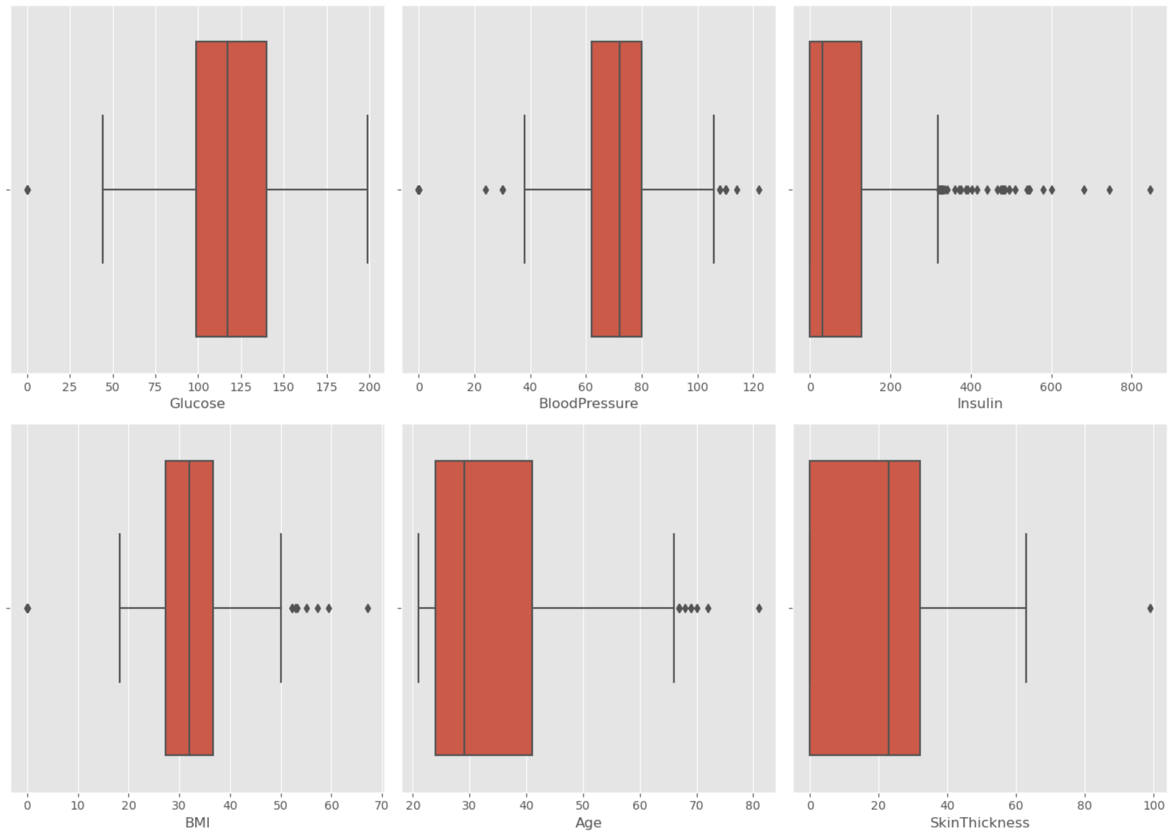


	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
Pregnancies	1	0.13	0.14	-0.082	-0.074	0.018	-0.034	0.54	0.22
Glucose	0.13	1	0.15	0.057	0.33	0.22	0.14	0.26	0.47
BloodPressure	0.14	0.15	1	0.21	0.089	0.28	0.041	0.24	0.065
SkinThickness	-0.082	0.057	0.21	1	0.44	0.39	0.18	-0.11	0.075
Insulin	-0.074	0.33	0.089	0.44	1	0.2	0.19	-0.042	0.13
BMI	0.018	0.22	0.28	0.39	0.2	1	0.14	0.036	0.29
DiabetesPedigreeFunction	-0.034	0.14	0.041	0.18	0.19	0.14	1	0.034	0.17
Age	0.54	0.26	0.24	-0.11	-0.042	0.036	0.034	1	0.24
Outcome	0.22	0.47	0.065	0.075	0.13	0.29	0.17	0.24	1

In [97]:
```
data.hist(figsize=(18,12))
plt.show()
```



In [98]:
```
features = ['Glucose', 'BloodPressure', 'Insulin', 'BMI', 'Age', 'SkinThicknes
plt.figure(figsize=(14, 10))

for i, feature in enumerate(features, start=1):
    plt.subplot(2, 3, i)
    sns.boxplot(x=feature, data=data)

plt.tight_layout()
plt.show()
```

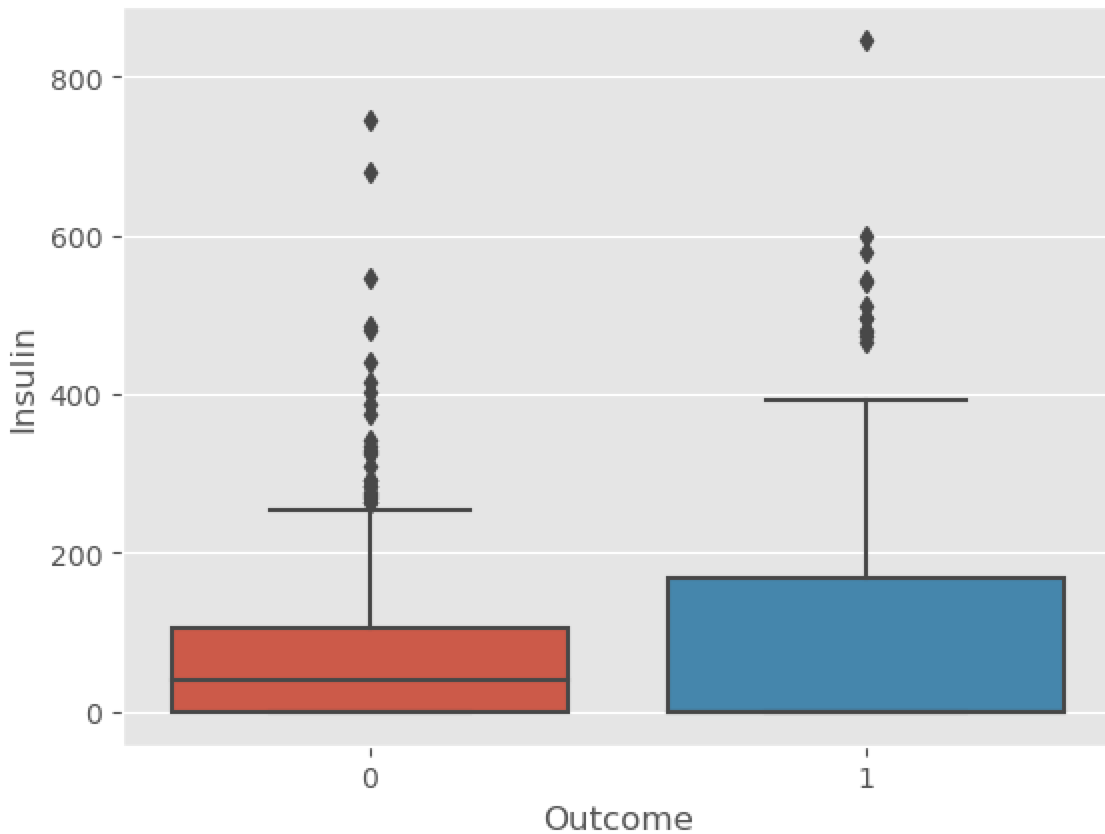


In [99]:
```
import seaborn as sns
import matplotlib.pyplot as plt
mean_col = ['Glucose', 'BloodPressure', 'Insulin', 'Age', 'Outcome', 'BMI']
sns.pairplot(data[mean_col])
plt.show()
```

```
C:\Users\Dell\anaconda3\Lib\site-packages\seaborn\axisgrid.py:118: UserWarni
ng: The figure layout has changed to tight
  self._figure.tight_layout(*args, **kwargs)
```

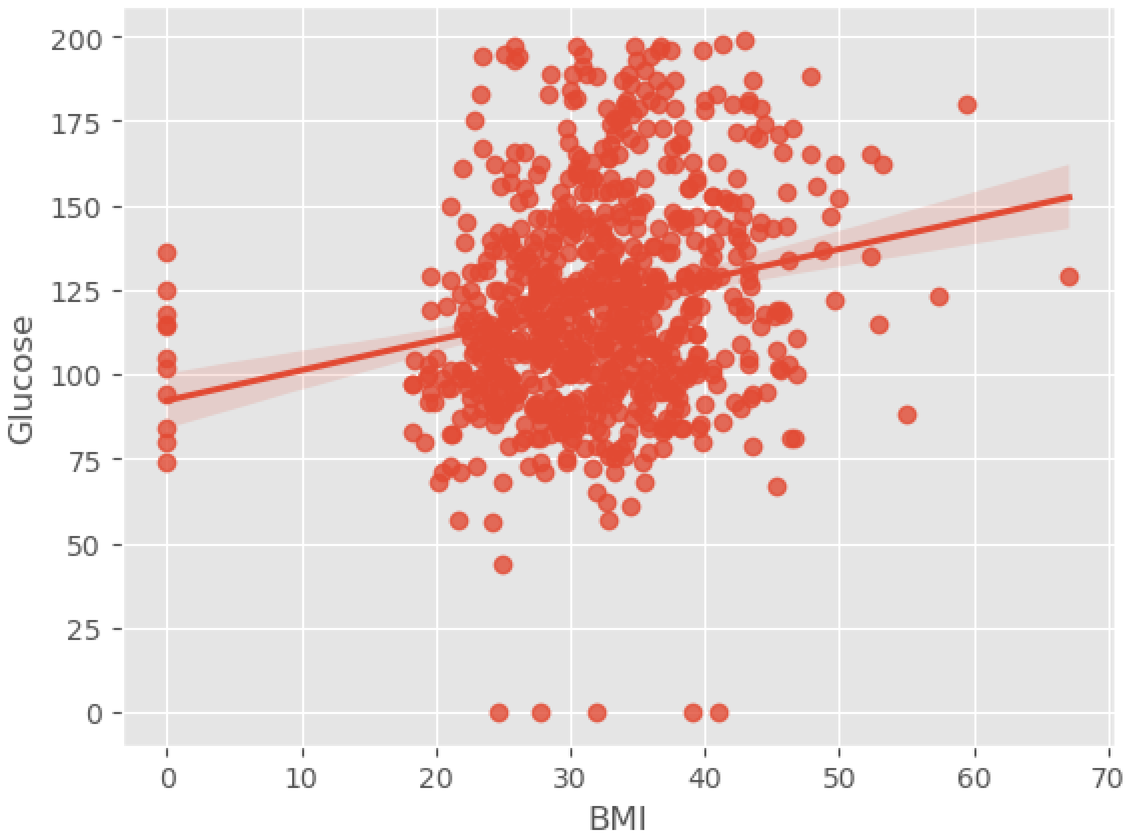
In [100]:
```
sns.boxplot(x='Outcome',y='Insulin',data=data)
```

Out[100]:
```
<Axes: xlabel='Outcome', ylabel='Insulin'>
```



In [101]:
```
sns.regplot(x='BMI', y= 'Glucose', data=data)
```

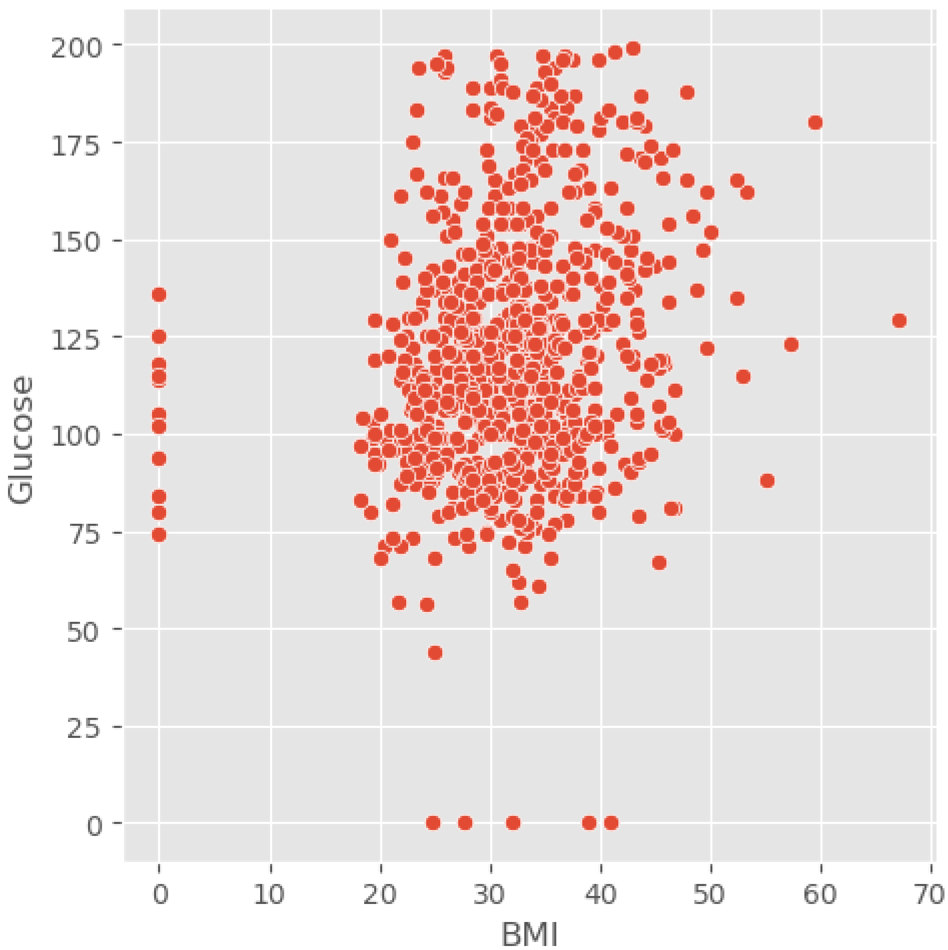
Out[101]: <Axes: xlabel='BMI', ylabel='Glucose'>



In [102]:
```
sns.relplot(x='BMI', y= 'Glucose', data=data)
```

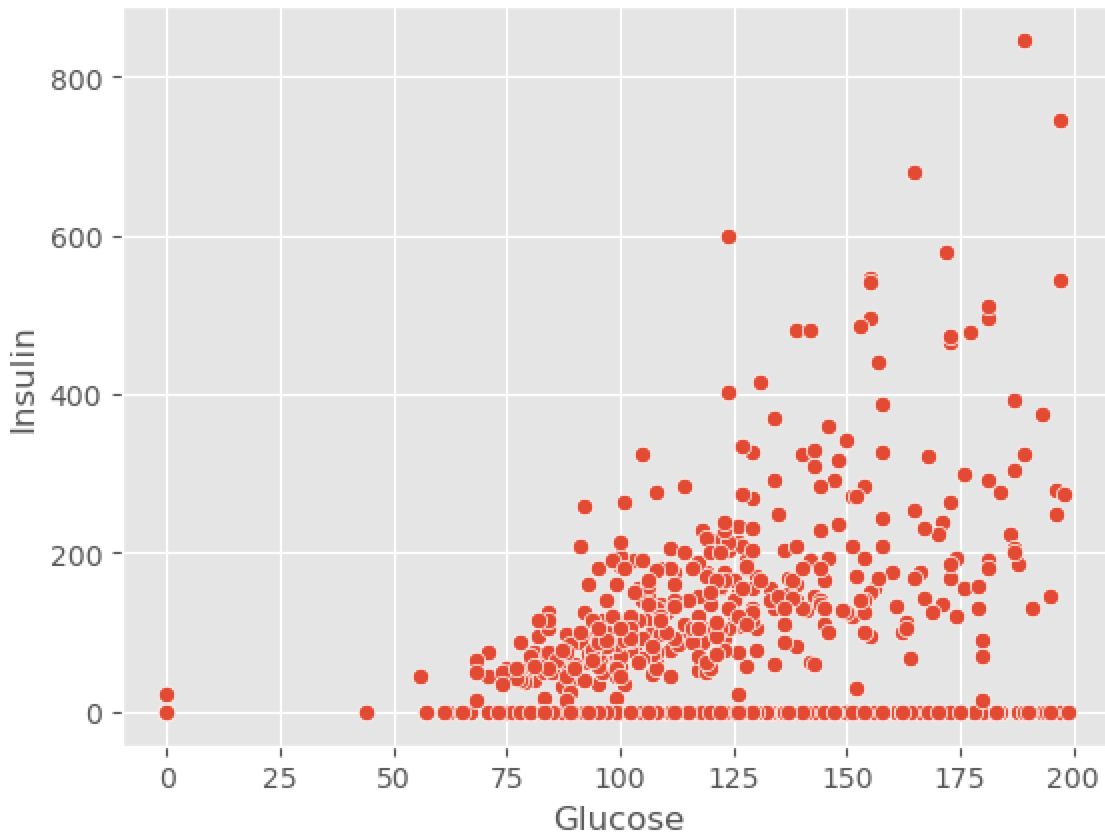
```
C:\Users\Dell\anaconda3\Lib\site-packages\seaborn\axisgrid.py:118: UserWarni
ng: The figure layout has changed to tight
  self._figure.tight_layout(*args, **kwargs)
```

Out[102]: <seaborn.axisgrid.FacetGrid at 0x1b7c8ad2050>



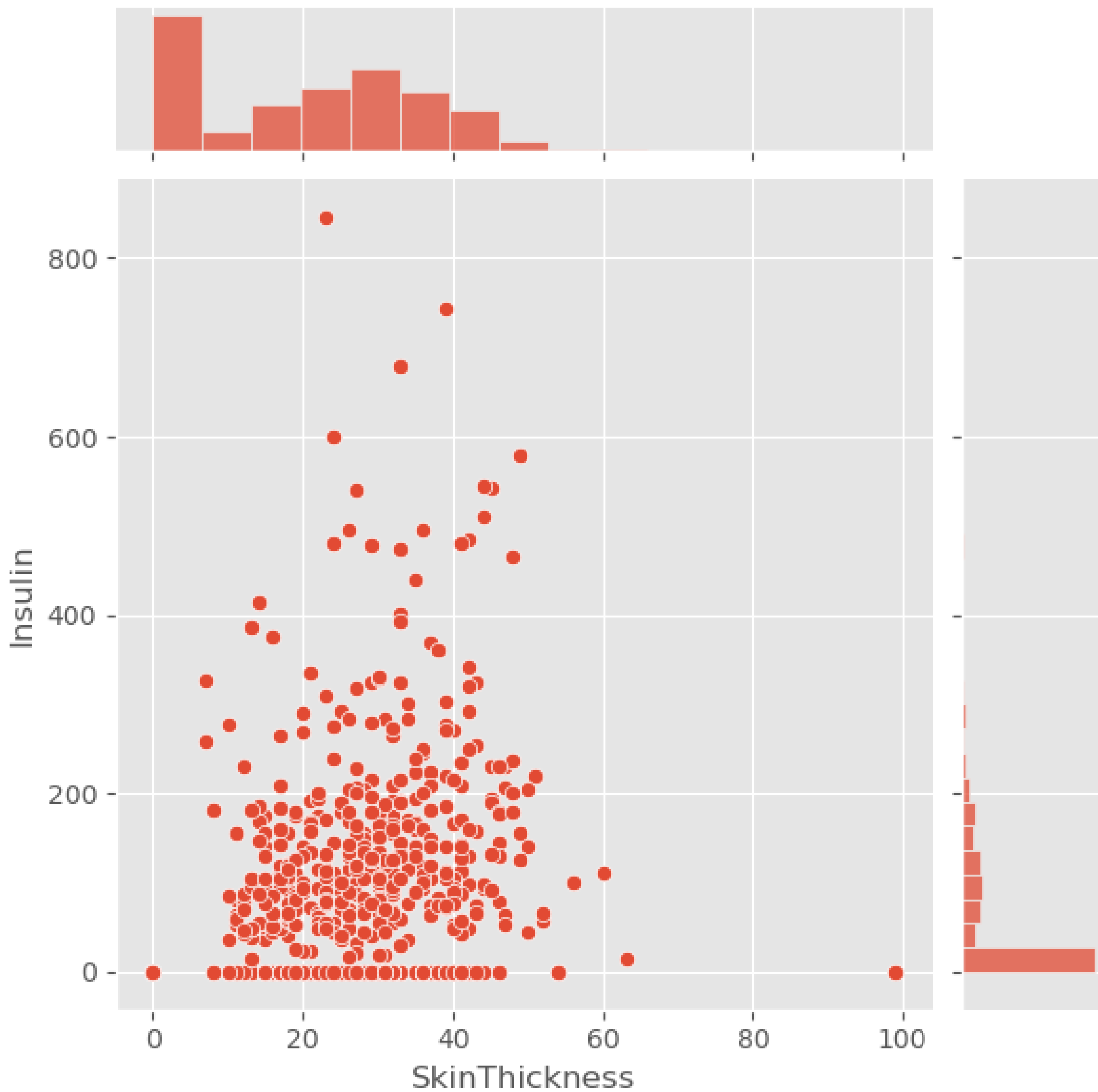
In [103]:
```
sns.scatterplot(x='Glucose', y= 'Insulin', data=data)
```

Out[103]:
```
<Axes: xlabel='Glucose', ylabel='Insulin'>
```



In [104]:
```
sns.jointplot(x='SkinThickness', y= 'Insulin', data=data)
```

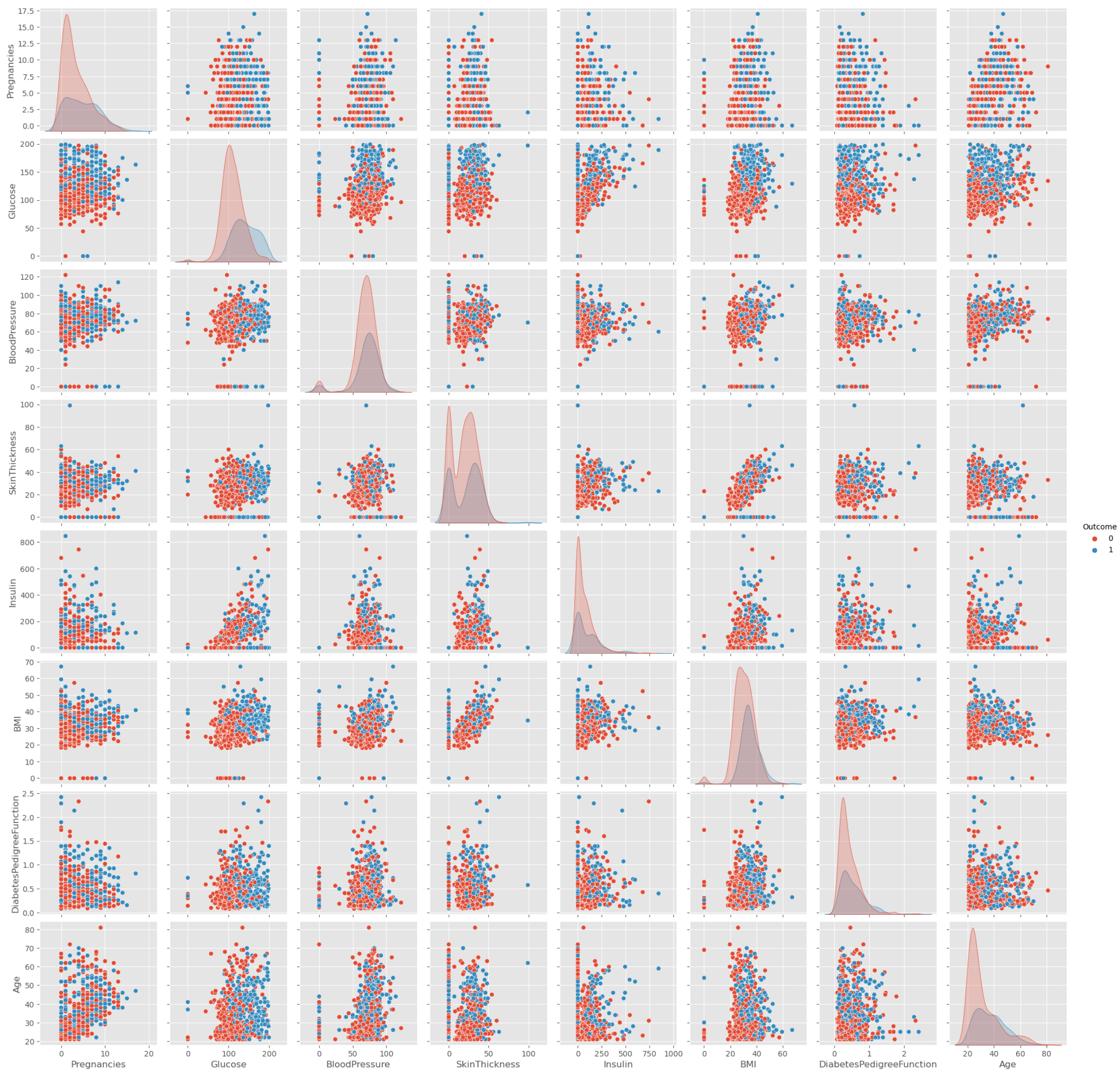
Out[104]: <seaborn.axisgrid.JointGrid at 0x1b7ca5450d0>



In [105]:
```
sns.pairplot(data,hue='Outcome')
```

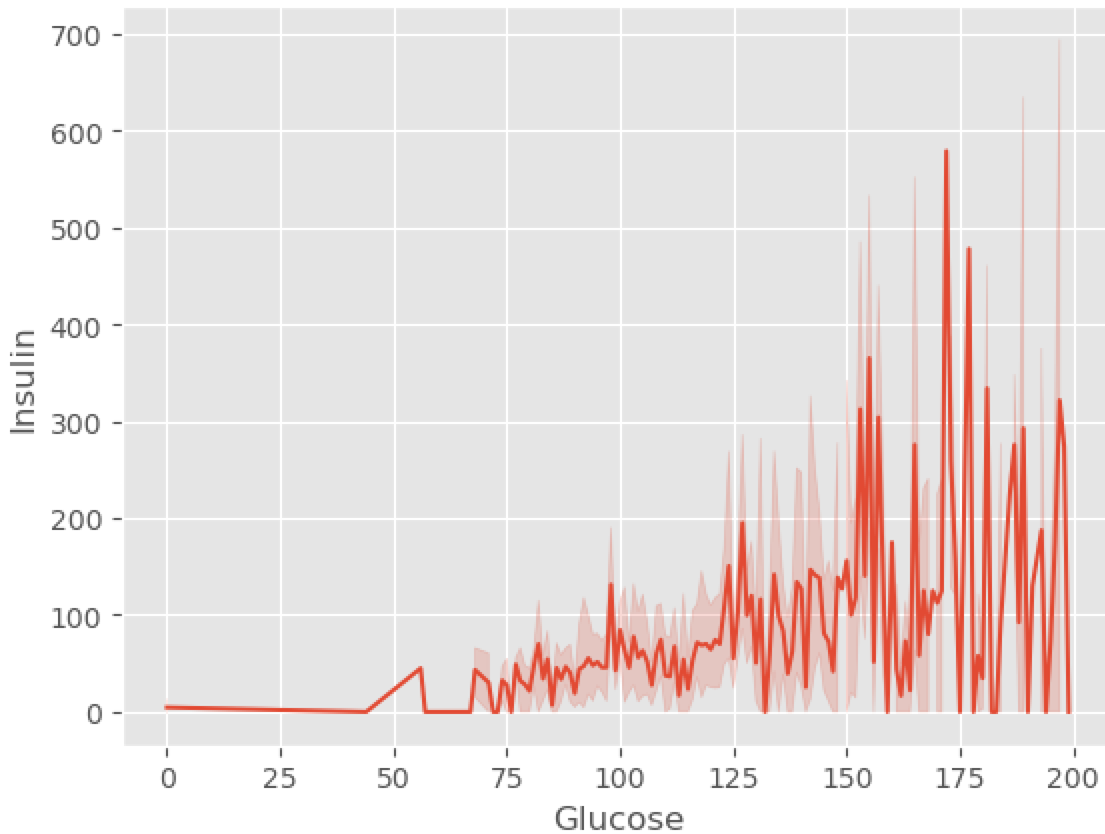
```
C:\Users\Dell\anaconda3\Lib\site-packages\seaborn\axisgrid.py:118: UserWarni
ng: The figure layout has changed to tight
  self._figure.tight_layout(*args, **kwargs)
```

Out[105]:
```
<seaborn.axisgrid.PairGrid at 0x1b7c6f9bc10>
```



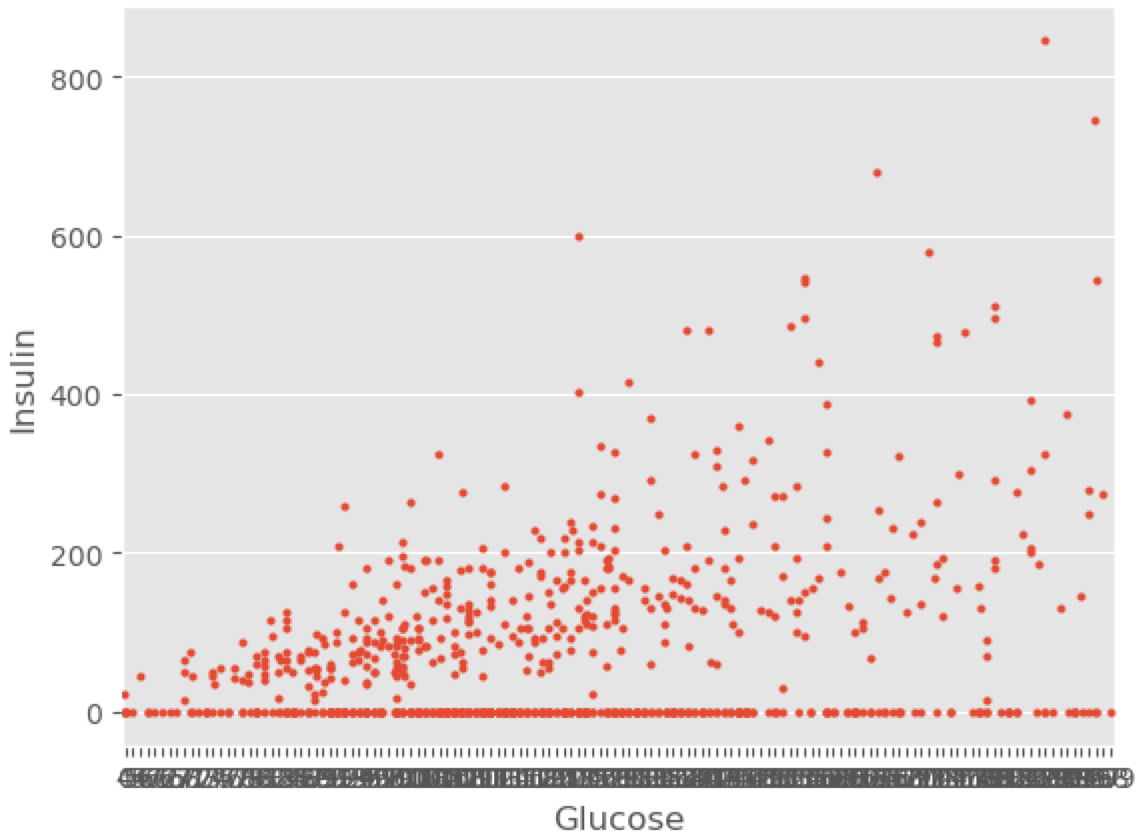
In [106]:
```
sns.lineplot(x='Glucose', y= 'Insulin', data=data)
```

Out[106]: <Axes: xlabel='Glucose', ylabel='Insulin'>

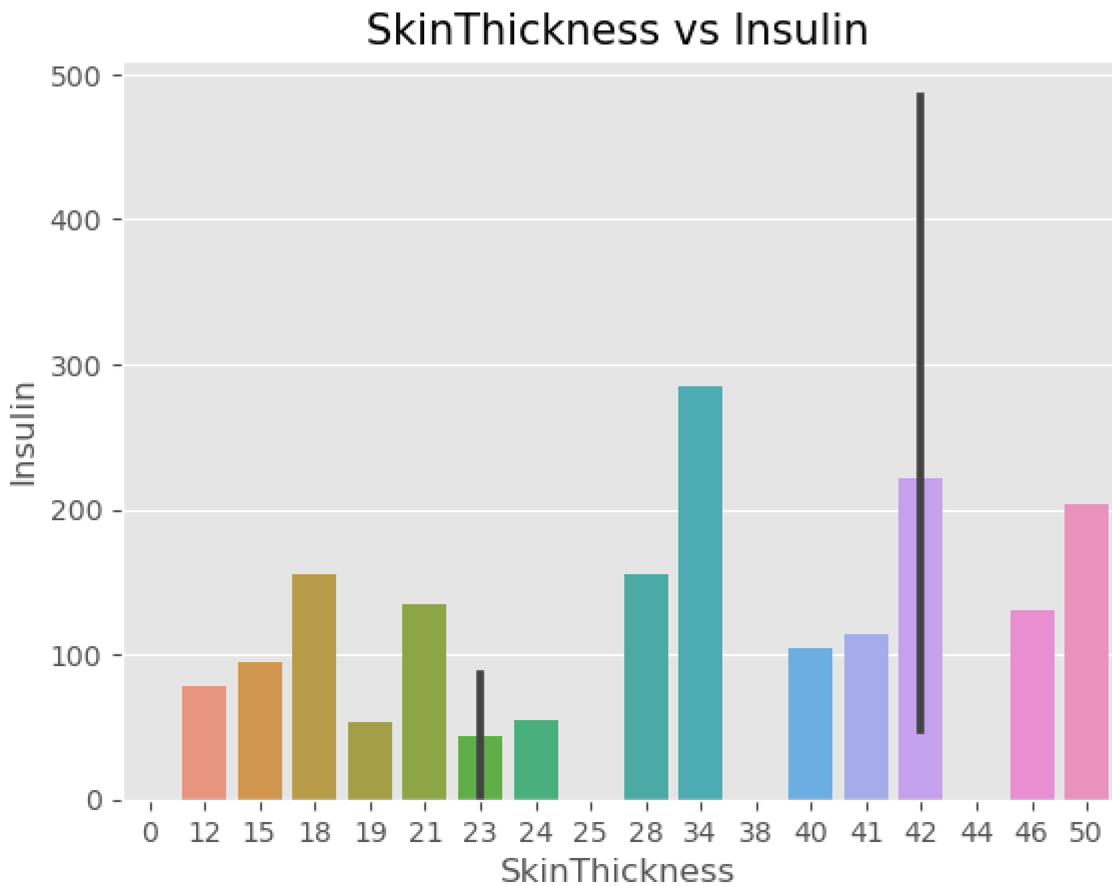


In [107]:
```
sns.stripplot(x='Glucose', y='Insulin', data=data, jitter=True, size=3)
```

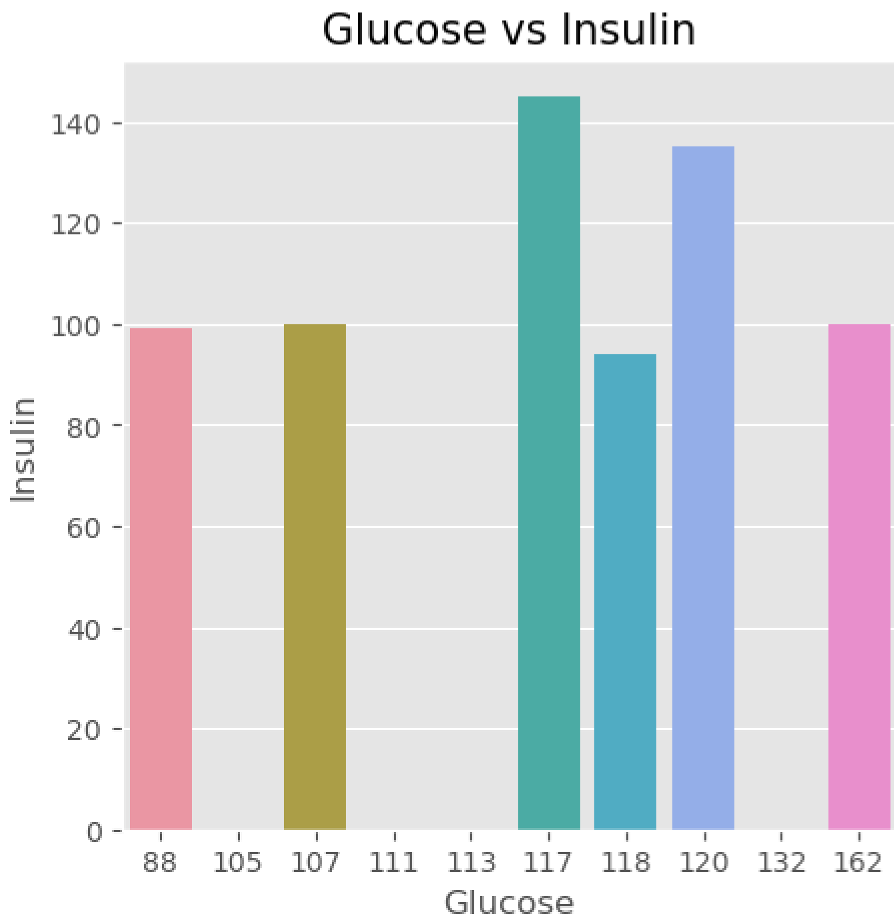
Out[107]: <Axes: xlabel='Glucose', ylabel='Insulin'>



In [108]:
```
sns.barplot(x="SkinThickness", y="Insulin", data=data[150:180])
plt.title("SkinThickness vs Insulin",fontsize=15)
plt.xlabel("SkinThickness")
plt.ylabel("Insulin")
plt.show()
plt.style.use("ggplot")
```



In [109]:
```
plt.figure(figsize=(5,5))
sns.barplot(x="Glucose", y="Insulin", data=data[120:130])
plt.title("Glucose vs Insulin",fontsize=15)
plt.xlabel("Glucose")
plt.ylabel("Insulin")
plt.show()
```



Pre-process,Training and Testing Data

In [123]:
```
x = data.drop(columns = 'Outcome')

y = data['Outcome']

from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(x,y,test_size=0.3,random_sta
```

Train the Neural Network model

In [124]:
```
from sklearn.neural_network import MLPClassifier
nn_model = MLPClassifier(hidden_layer_sizes=(100,50), max_iter=500, activatio
nn_model.fit(X_train, y_train)
```

Out[124]:
```
MLPClassifier
MLPClassifier(hidden_layer_sizes=(100, 50), max_iter=500, random_state=42)
```

MODELS

1. Logistic Regression

In [125]:
```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))

from sklearn.metrics import accuracy_score
LRAcc = accuracy_score(y_pred,y_test)
print('Logistic Regression accuracy is: {:.2f}%'.format(LRAcc*100))
```

```
              precision    recall  f1-score   support

           0       0.80      0.80      0.80       151
           1       0.62      0.62      0.62        80

    accuracy                           0.74       231
   macro avg       0.71      0.71      0.71       231
weighted avg       0.74      0.74      0.74       231

[[121  30]
 [ 30  50]]
Logistic Regression accuracy is: 74.03%

C:\Users\Dell\anaconda3\Lib\site-packages\sklearn\linear_model\_logistic.py:
460: 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 (https://scik
it-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 (https://scikit-learn.org/stable/modules/linear_model.html#logistic-re
gression)
  n_iter_i = _check_optimize_result(
```

2. SVM

In [126]:
```
from sklearn.svm import SVC  # Correct import statement
from sklearn.metrics import classification_report, confusion_matrix, accuracy_

model = SVC()
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))

SVMAcc = accuracy_score(y_test, y_pred)
print('SVM accuracy is: {:.2f}%'.format(SVMAcc * 100))
```

```
              precision    recall  f1-score   support

           0       0.76      0.87      0.81       151
           1       0.66      0.49      0.56        80

    accuracy                           0.74       231
   macro avg       0.71      0.68      0.69       231
weighted avg       0.73      0.74      0.72       231

[[131  20]
 [ 41  39]]
SVM accuracy is: 73.59%
```

3. Decison Tree

In [127]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report, confusion_matrix, accuracy_

model = DecisionTreeClassifier()
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))

DTAcc = accuracy_score(y_test, y_pred)
print('Decision Tree accuracy is: {:.2f}%'.format(DTAcc * 100))
```

```
              precision    recall  f1-score   support

           0       0.79      0.71      0.75       151
           1       0.54      0.64      0.58        80

    accuracy                           0.68       231
   macro avg       0.66      0.67      0.66       231
weighted avg       0.70      0.68      0.69       231

[[107  44]
 [ 29  51]]
Decision Tree accuracy is: 68.40%
```

Compare Models

In [128]:
```
compare = pd.DataFrame({'Models Trained': ['Logistic Regression', 'SVM', 'Dec
                        'Accuracy': [LRAcc*100, SVMAcc*100, DTAcc*100]})
compare.sort_values(by='Accuracy', ascending=False)
```

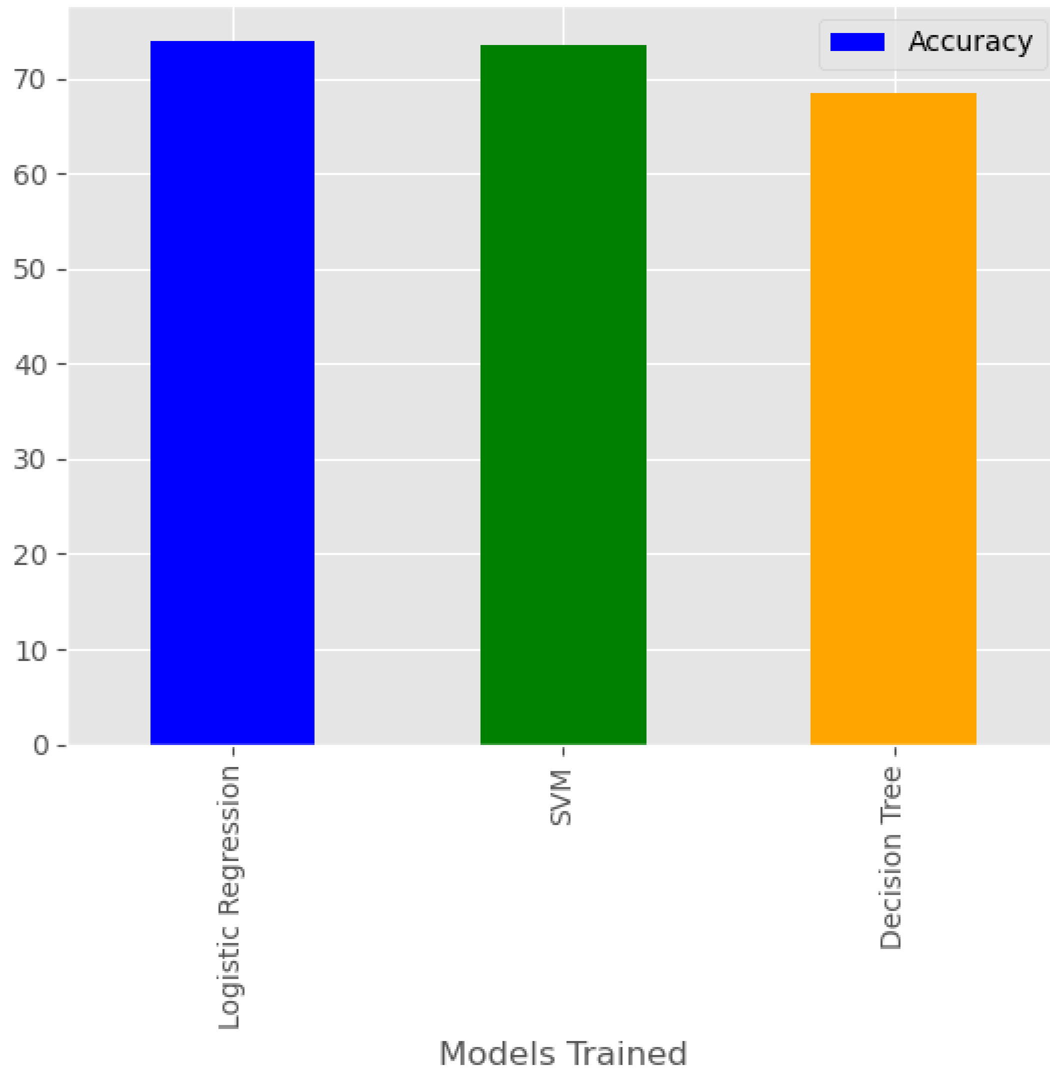
Out[128]:

	Models Trained	Accuracy
0	Logistic Regression	74.025974
1	SVM	73.593074
2	Decision Tree	68.398268

Plotting Model Comparison

In [130]:
```
compare.plot(x='Models Trained', y='Accuracy', kind='bar', color=['blue', 'gre
```

Out[130]:
```
<Axes: xlabel='Models Trained'>
```



From the comparison plot, among the 3 Machine Learning Models, Logistic Regression had achieved the highest accuracy of 74.025974%.