CSE17040

October 20, 2020

1 Lab 12 - Performance Metrics

-CB.EN.U4CSE17040

(Manojkumar V K)

1.1 Import necessary libraries

[1]:
```
import numpy as np
import pandas as pd
from scipy.stats import iqr
import seaborn as sns
import matplotlib.pyplot as plt

%matplotlib inline
sns.set_style("whitegrid")
```

[2]:
```
data = pd.read_csv('diabetes.csv')
data.head()
```

[2]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

1.2 Exploratory Data Analysis

Dataset Desription To get the central tendency of different fields of the dataset we can simply use describe() method. The central tendency includes mean, median, mode. The describe function lists out: total count, mean, standard deviation, minimum value, First quartile(Q1), Median(Q2), Third Quartile(Q3), maximum value. In the dataset we have 'Outcome' as the target variable. The value 1 or 0 which indicates whether or not the subject has diabetes.

[3]:
```
data.describe()
```

[3]:

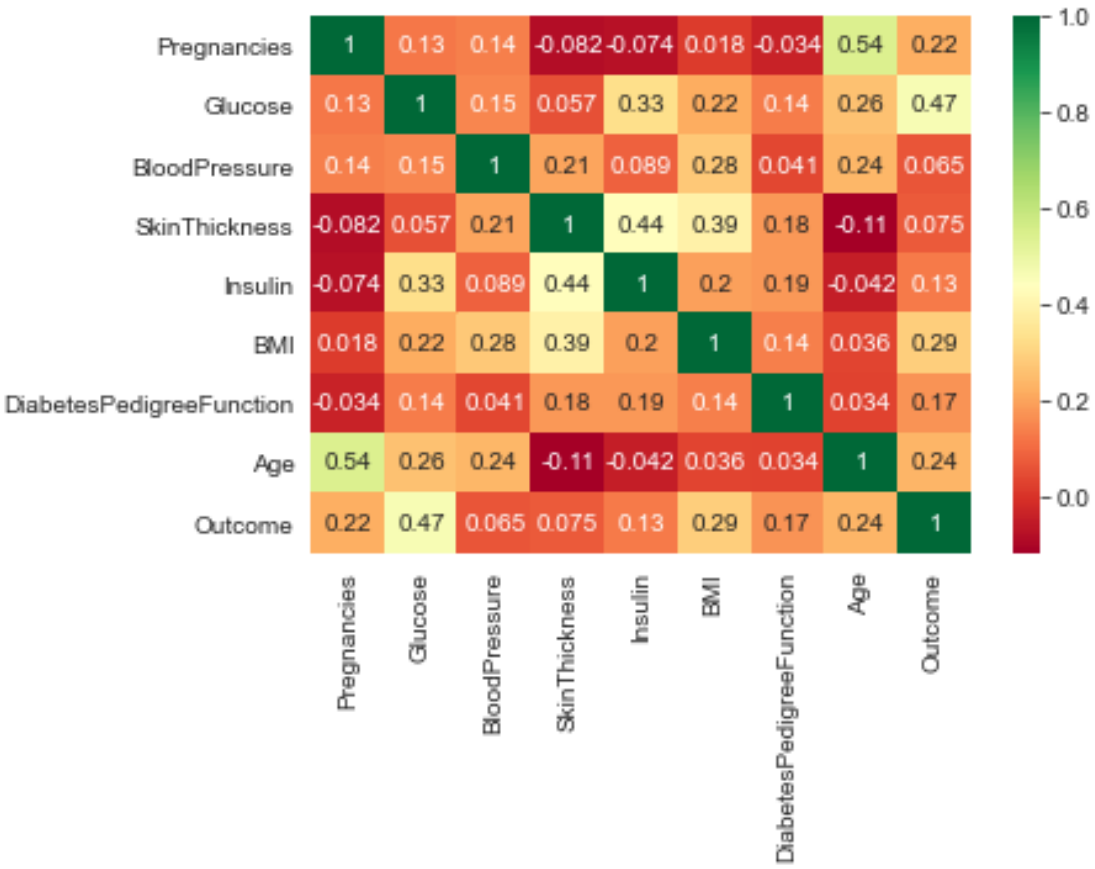
	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	\
count	768.000000	768.000000	768.000000	768.000000	768.000000	
mean	3.845052	120.894531	69.105469	20.536458	79.799479	
std	3.369578	31.972618	19.355807	15.952218	115.244002	
min	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	1.000000	99.000000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

	BMI	DiabetesPedigreeFunction	Age	Outcome
count	768.000000	768.000000	768.000000	768.000000
mean	31.992578	0.471876	33.240885	0.348958
std	7.884160	0.331329	11.760232	0.476951
min	0.000000	0.078000	21.000000	0.000000
25%	27.300000	0.243750	24.000000	0.000000
50%	32.000000	0.372500	29.000000	0.000000
75%	36.600000	0.626250	41.000000	1.000000
max	67.100000	2.420000	81.000000	1.000000

Correlation between different fields To find the correlation of different fields we use corr() and plot it using heatmap() function in seaborn. By looking at the corelation we can infer which fields may be used to predict the targe variable. We can even remove redundant fields if we find too much correlation amongst them, hence reducing dimensionality in the dataset which may make it easier for different prediction algorithms to process and provide effective results. From the below heatmap we can see the correlation between the fields. Lighter areas suggest more correlation and similarly darker areas suggest very little or no correlation.

[4]:
```
corr= data.corr()
sns.heatmap(corr,annot = True,cmap = "RdYlGn")
```

[4]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd18bfe5250>
```



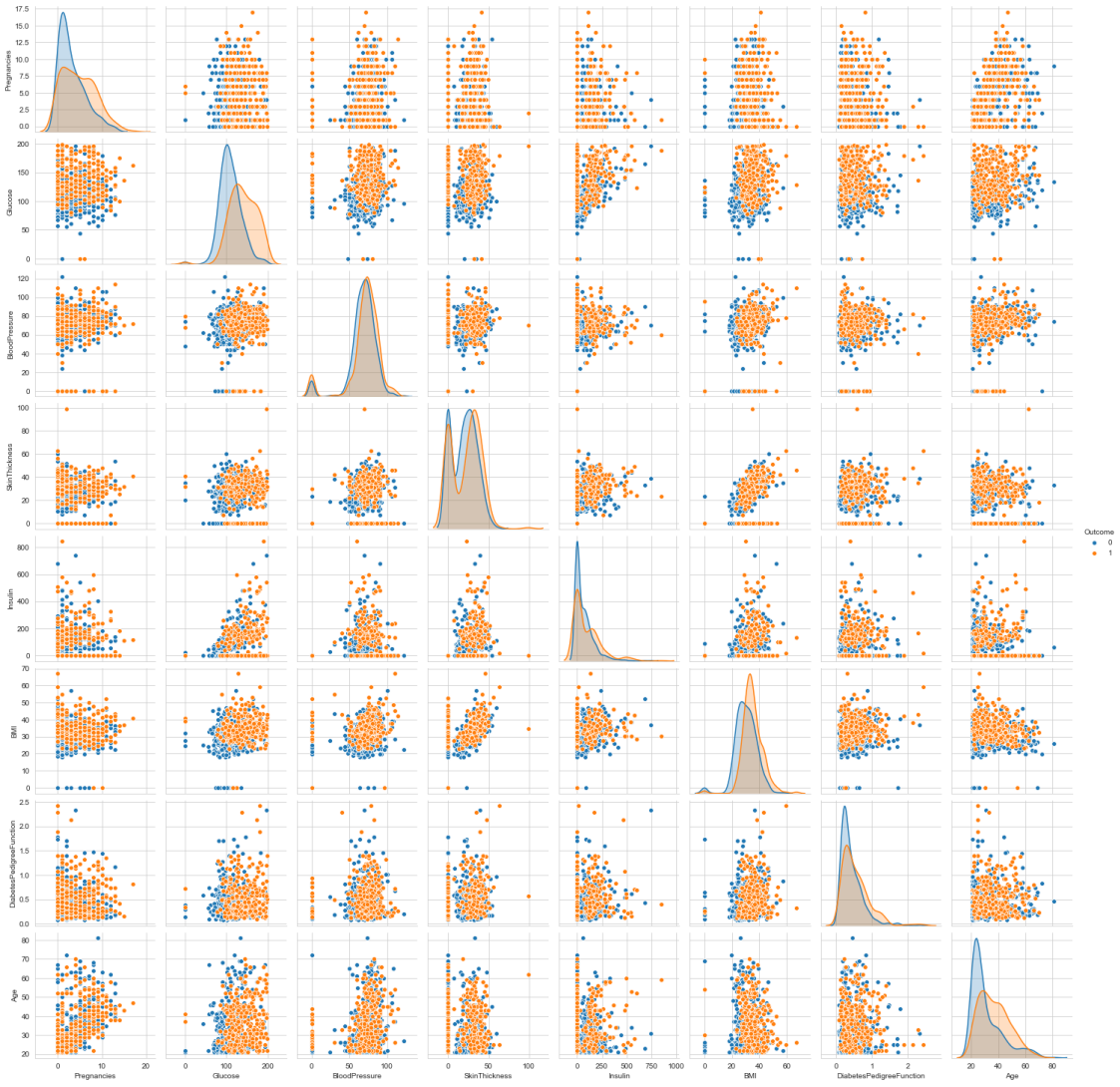
From above heatmap we can infer that 'Glucose' and 'Outcome' have a correlation coefficient of 0.47. We also see a prominent correlation between 'Age' and 'Pregnancies' i.e. 0.54 which is self explanatory as the age of a woman increases the number of pregnancies she had would tend to increase.

****Visualizing data using different plots****

Pairplot

[5]:
```
sns.pairplot(data,hue='Outcome')
```

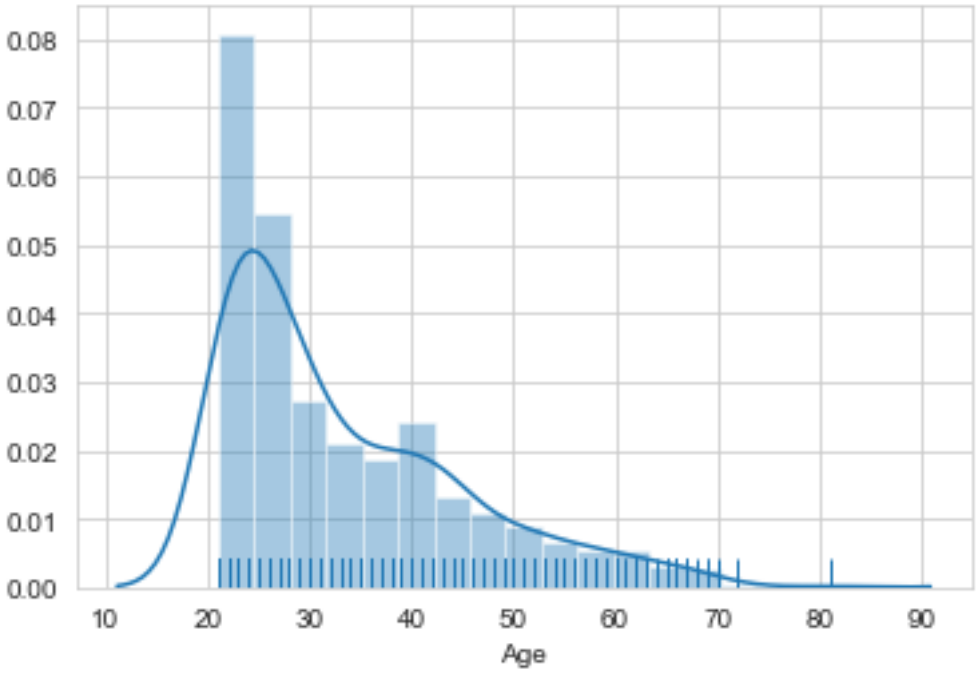
[5]:
```
<seaborn.axisgrid.PairGrid at 0x7fd18977dcd0>
```



Distplot

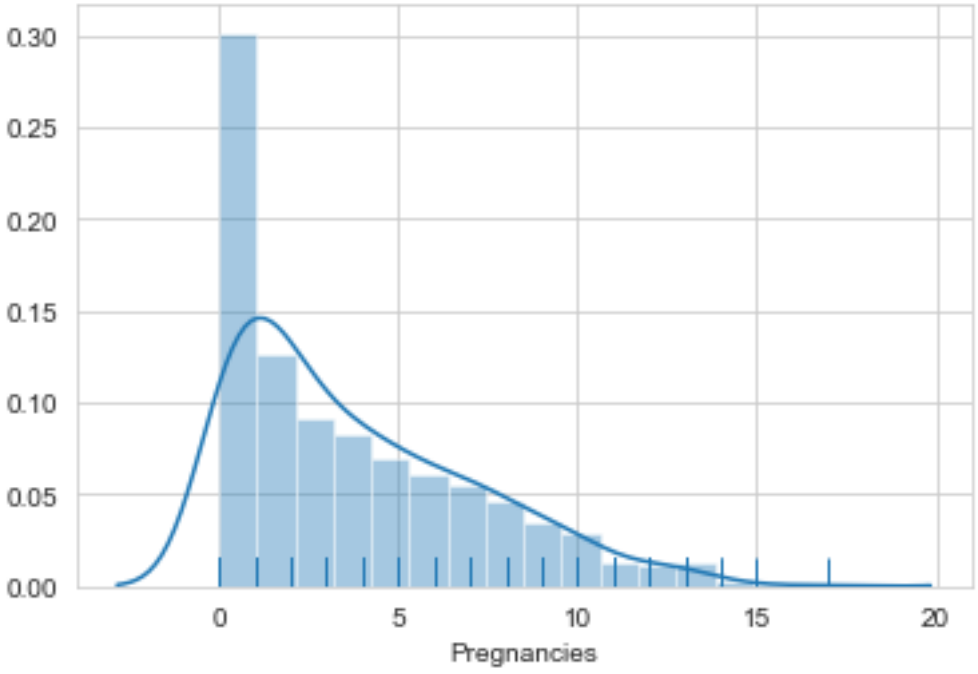
[6]:
```
sns.distplot(data['Age'],kde=True,rug=True)
```

[6]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd187e2bf90>
```



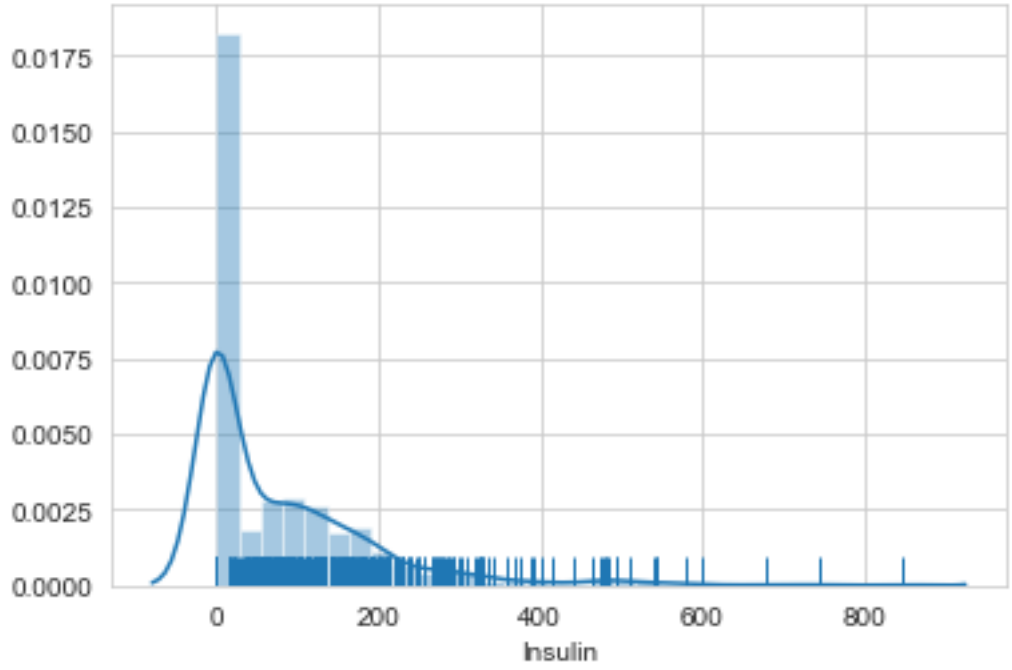
[7]:
```
sns.distplot(data['Pregnancies'],kde=True,rug=True)
```

[7]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd187722250>
```



[8]:
```
sns.distplot(data['Insulin'],kde=True,rug=True)
```

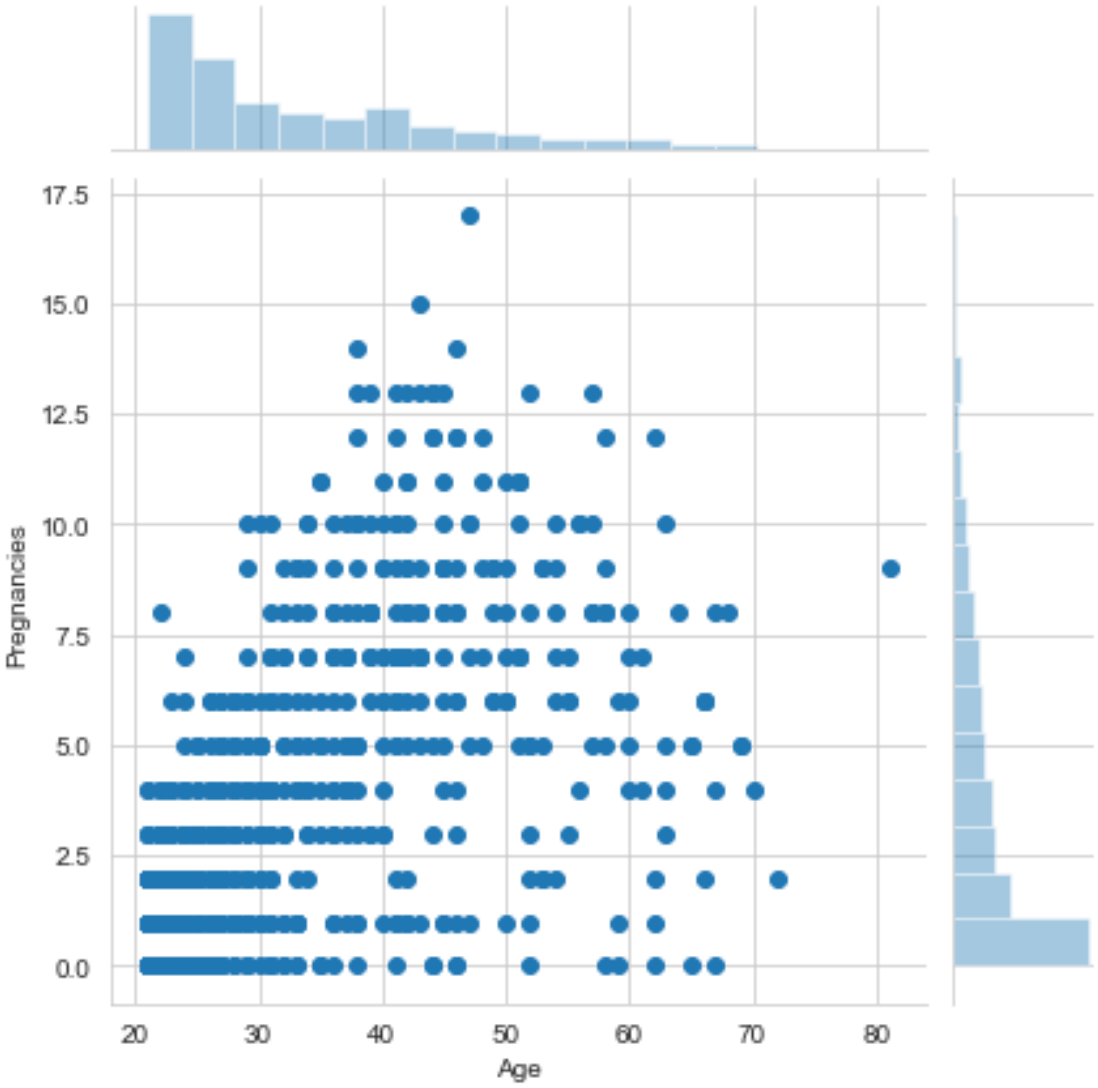
[8]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd184a83450>
```



Jointplot

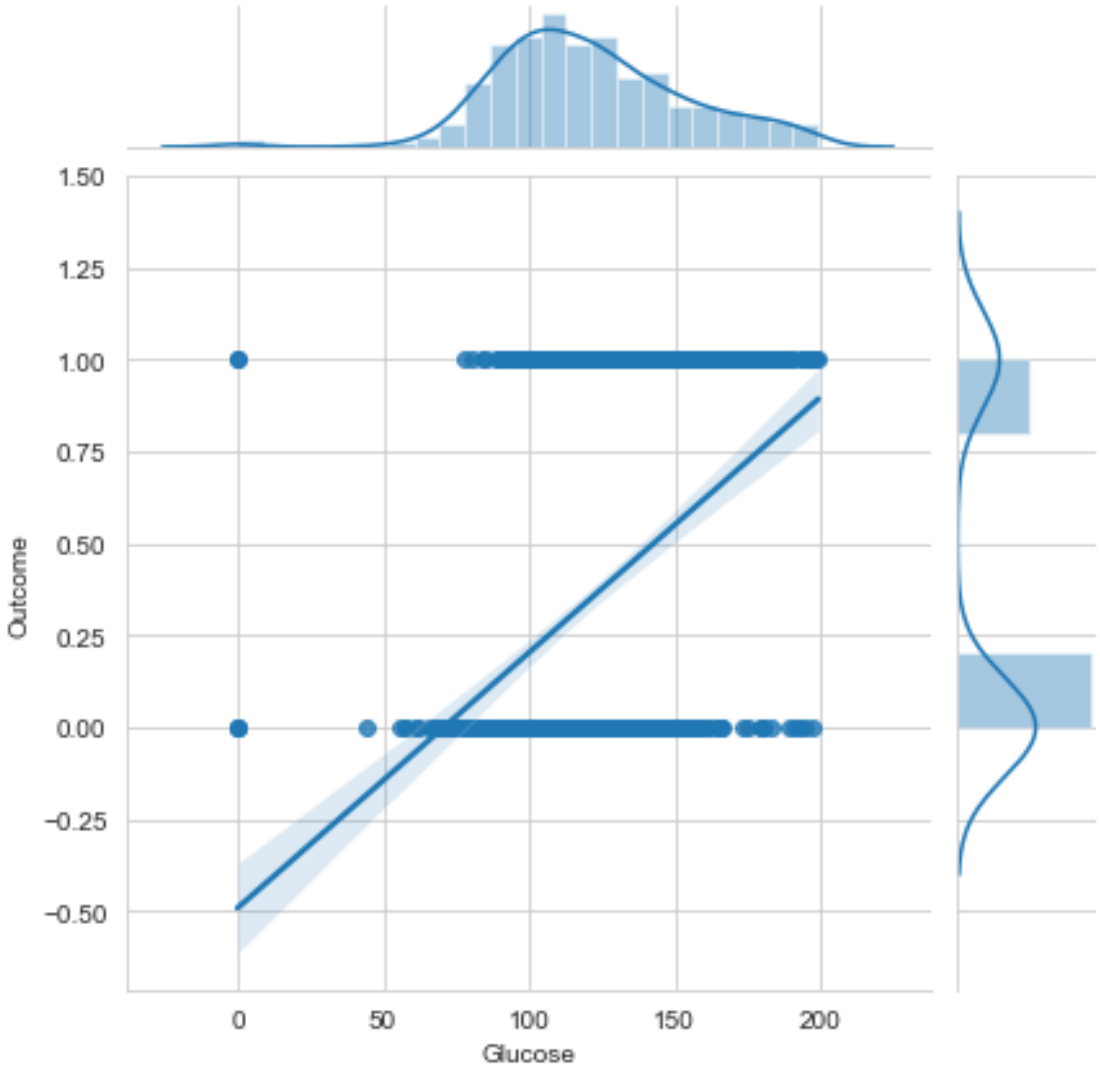
[9]:
```
sns.jointplot(data['Age'],data['Pregnancies'])
```

[9]:
```
<seaborn.axisgrid.JointGrid at 0x7fd184940cd0>
```



[10]:
```
sns.jointplot(data['Glucose'],data['Outcome'],kind='reg')
```

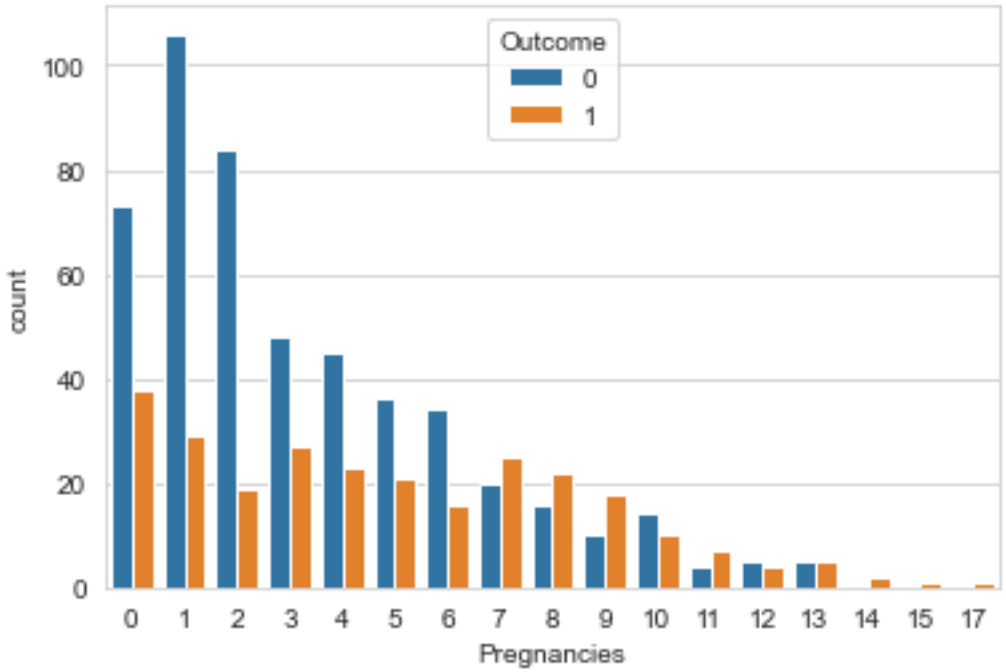
[10]: <seaborn.axisgrid.JointGrid at 0x7fd1847fbed0>



Countplot

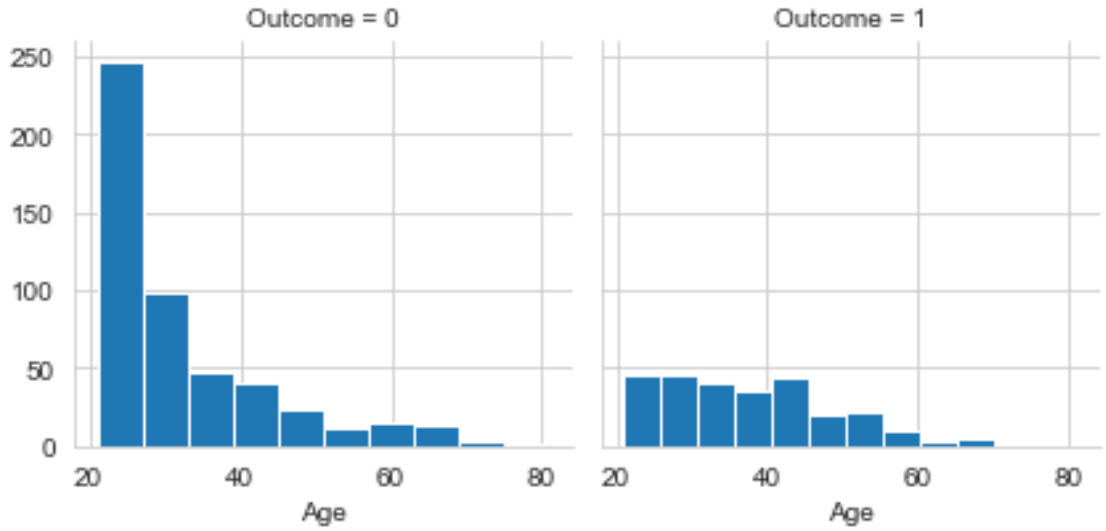
[11]:
```
sns.countplot(x='Pregnancies',hue='Outcome',data=data)
```

[11]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd18462d190>
```



[12]:
```
g = sns.FacetGrid(data,col='Outcome')
g.map(plt.hist,'Age')
```

[12]: <seaborn.axisgrid.FacetGrid at 0x7fd184567750>



Here insulin_0 represents the mean value of Insulin for the subjects for which Outcome is 0 and insulin_1 represents the mean value of Insulin for the subjects for which Outcome is 1. We see

considerable difference in both the values

[13]:
```
insulin_0 = data[(data['Outcome']==0) & (data['Insulin']!=0)]['Insulin'].mean()
insulin_1 = data[(data['Outcome']==1) & (data['Insulin']!=0)]['Insulin'].mean()
print(insulin_0)
print(insulin_1)
```

```
130.28787878787878
206.84615384615384
```

[14]:
```
glucose_0 = data[(data['Outcome']==0) & (data['Glucose']!=0)]['Glucose'].mean()
glucose_1 = data[(data['Outcome']==1) & (data['Glucose']!=0)]['Glucose'].mean()
print(glucose_0)
print(glucose_1)
```

```
110.64386317907444
142.31954887218046
```

[15]:
```
skin_0 = data[(data['Outcome']==0) & (data['SkinThickness']!
 ↪=0)]['SkinThickness'].mean()
skin_1 = data[(data['Outcome']==1) & (data['SkinThickness']!
 ↪=0)]['SkinThickness'].mean()
print(skin_0)
print(skin_1)
```

```
27.235457063711912
33.0
```

The interquartile range for the Glucose levels for Outcome 0 and 1 respectively are listed below. Again we disregard the 0 values for Glucose. The interquartile value handles the outliers i.e. the extremely high values efficiently as compared to mean.

[16]:
```
print(iqr(data[(data['Outcome']==0) & (data['Glucose']!
 ↪=0)]['Glucose'],rng=(25,75)))
print(iqr(data[(data['Outcome']==1) & (data['Glucose']!
 ↪=0)]['Glucose'],rng=(25,75)))
```

```
32.0
48.0
```

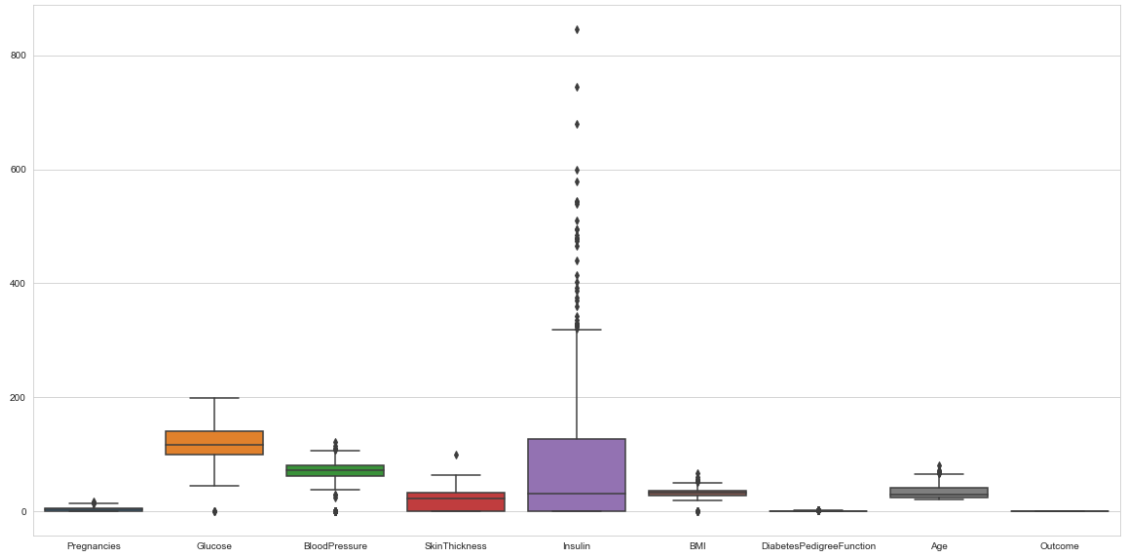
[17]:
```
print(iqr(data[(data['Outcome']==0) & (data['Insulin']!
 ↪=0)]['Insulin'],rng=(25,75)))
print(iqr(data[(data['Outcome']==1) & (data['Insulin']!
 ↪=0)]['Insulin'],rng=(25,75)))
```

```
95.25
111.75
```

Boxplot

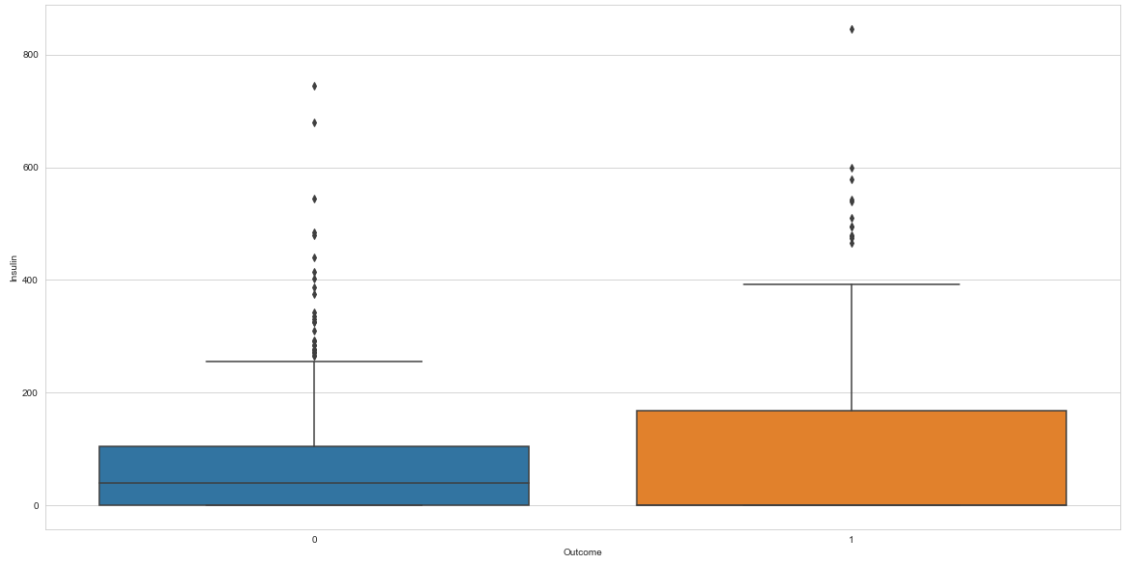
[18]:
```
plt.figure(figsize=(20,10))
sns.boxplot(data=data)
```

[18]: <matplotlib.axes._subplots.AxesSubplot at 0x7fd18446e310>



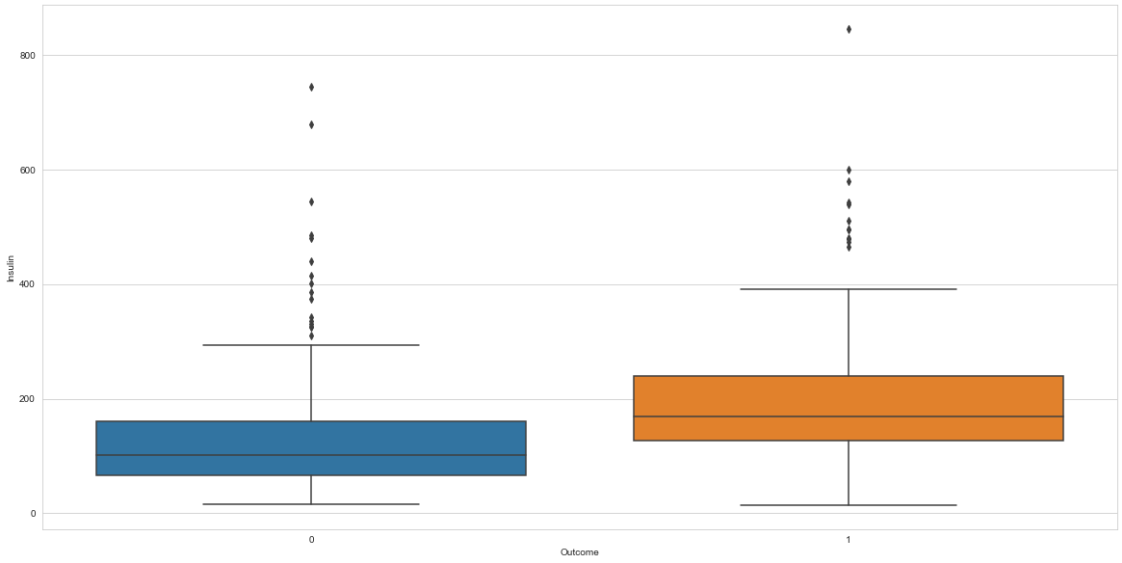
[19]:
```
plt.figure(figsize=(20,10))
sns.boxplot(x='Outcome',y='Insulin',data=data)
```

[19]: <matplotlib.axes._subplots.AxesSubplot at 0x7fd184b6d590>



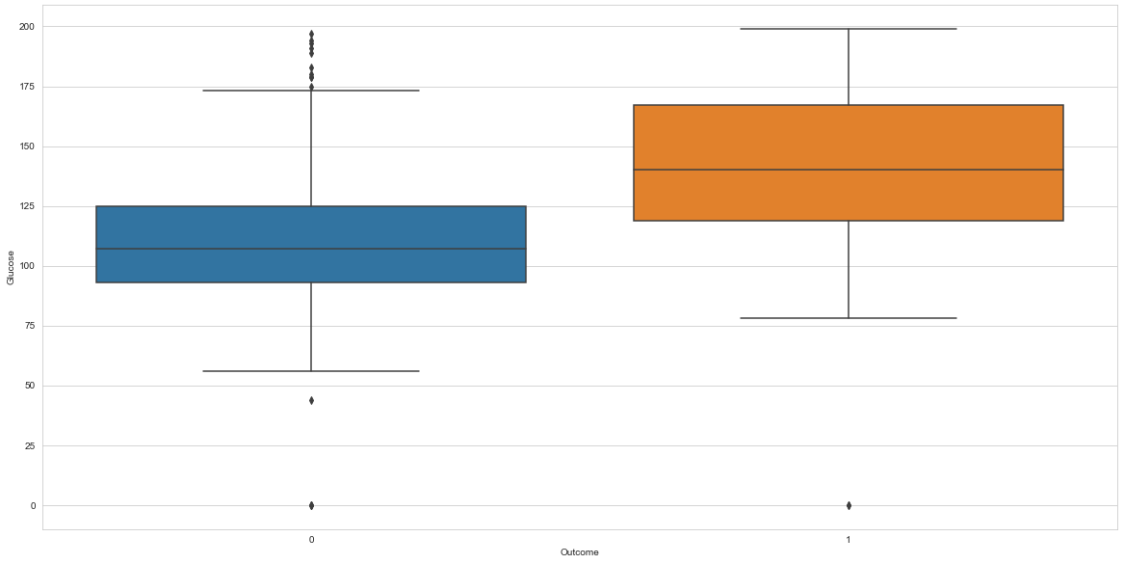
[20]:
```
data_new = data[data['Insulin']!=0]
plt.figure(figsize=(20,10))
sns.boxplot(x='Outcome',y='Insulin',data=data_new)
```

[20]: <matplotlib.axes._subplots.AxesSubplot at 0x7fd1873b5550>



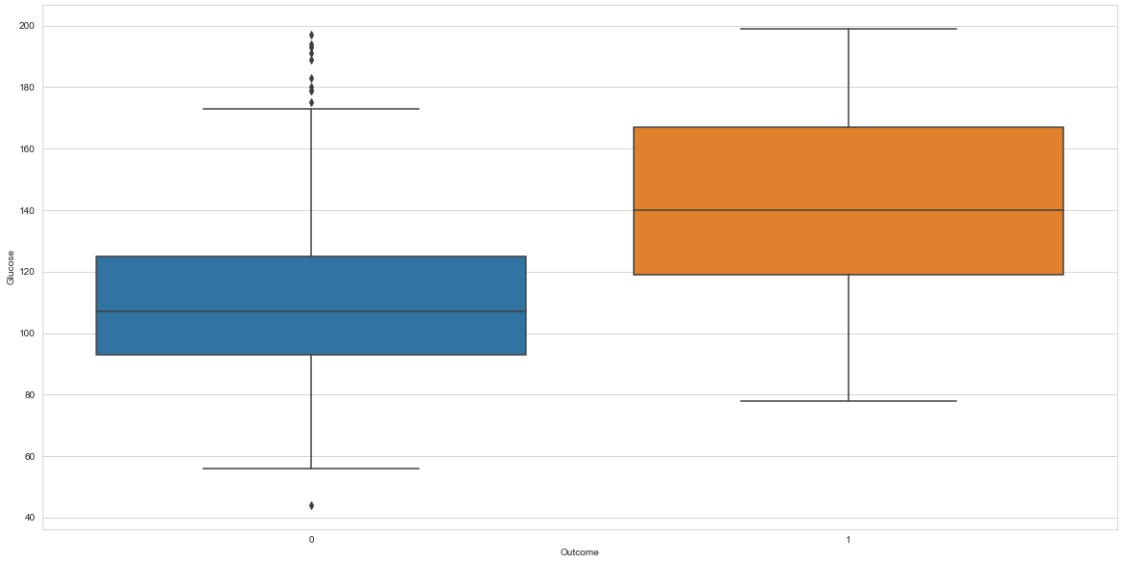
[21]:
```
plt.figure(figsize=(20,10))
sns.boxplot(x='Outcome',y='Glucose',data=data)
```

[21]: <matplotlib.axes._subplots.AxesSubplot at 0x7fd18733edd0>



[22]:
```
data_new = data[data['Glucose']!=0]
plt.figure(figsize=(20,10))
sns.boxplot(x='Outcome',y='Glucose',data=data_new)
```

[22]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd1873c3c90>
```



For Outcome=1 the median is close to 140 and for Outcome=0 the median is close to 105 though having some outliers.

Filling the zero values

Now in this data set we see a lot of zero values. What we can do is for better observations we can replace the zero values with the mean or median values of the columns. After that we look into the same results. So lets replace the zeroes with medians

[23]:
```
data_new = pd.DataFrame()
data_new['Glucose'] = data['Glucose'].replace(0,data['Glucose'].median())
data_new['Insulin'] = data['Insulin'].replace(0,data['Insulin'].median())
data_new['BMI'] = data['BMI'].replace(0,data['BMI'].median())
data_new['SkinThickness'] = data['SkinThickness'].
 ↪replace(0,data['SkinThickness'].median())
data_new['BloodPressure'] = data['BloodPressure'].
 ↪replace(0,data['BloodPressure'].median())
data_new['Outcome'] = data['Outcome']
```

[24]:
```
data_new.head()
```

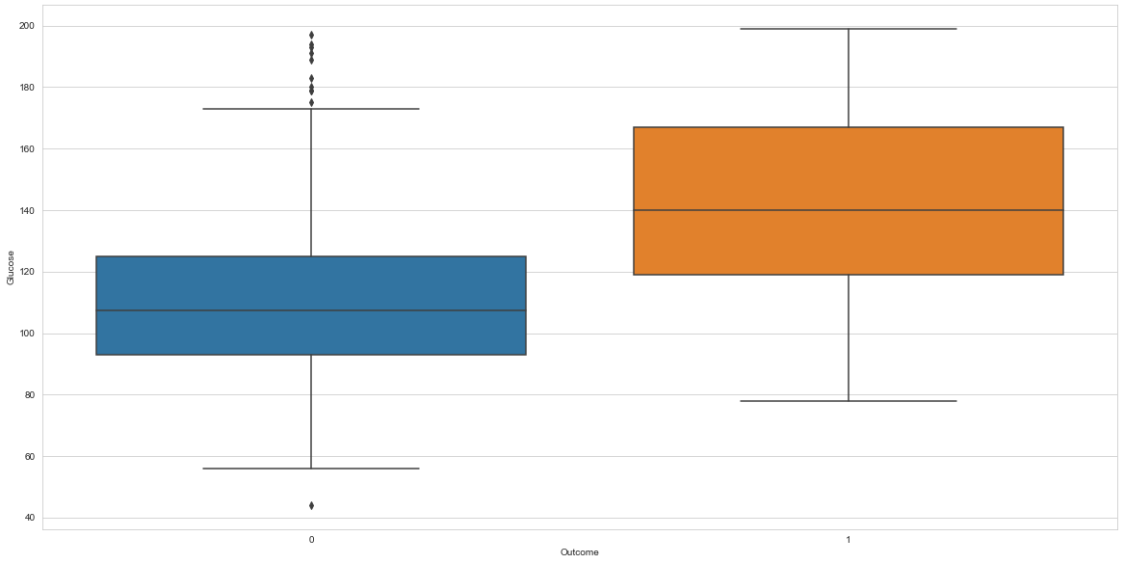
[24]:

	Glucose	Insulin	BMI	SkinThickness	BloodPressure	Outcome
0	148	30.5	33.6	35	72	1

1	85	30.5	26.6	29	66	0
2	183	30.5	23.3	23	64	1
3	89	94.0	28.1	23	66	0
4	137	168.0	43.1	35	40	1

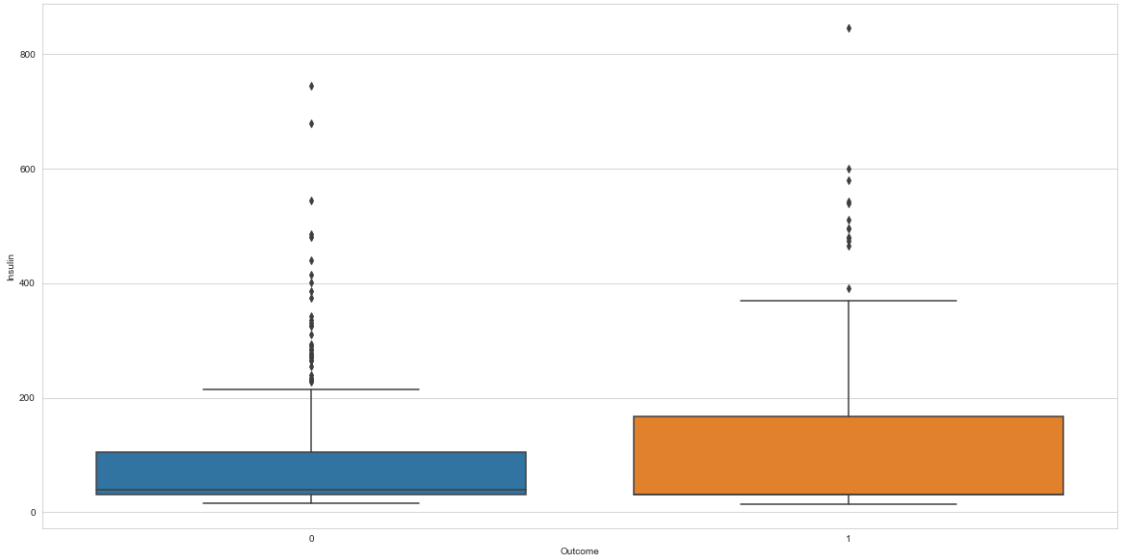
[25]:
```
plt.figure(figsize=(20,10))
sns.boxplot(x='Outcome',y='Glucose',data=data_new)
```

[25]: <matplotlib.axes._subplots.AxesSubplot at 0x7fd1846a2750>



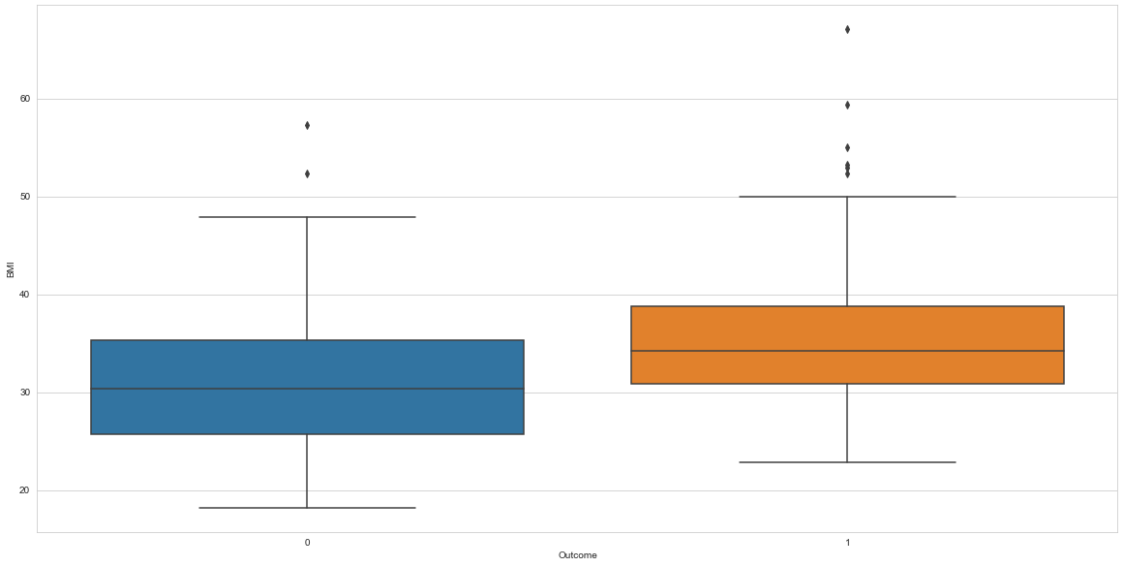
[26]:
```
plt.figure(figsize=(20,10))
sns.boxplot(x='Outcome',y='Insulin',data=data_new)
```

[26]: <matplotlib.axes._subplots.AxesSubplot at 0x7fd187210190>



[27]:
```
plt.figure(figsize=(20,10))
sns.boxplot(x='Outcome',y='BMI',data=data_new)
```

[27]: <matplotlib.axes._subplots.AxesSubplot at 0x7fd184484150>



[28]:
```
print(iqr(data_new[(data_new['Outcome']==0)]['Glucose'],rng=(25,75)))

print(iqr(data_new[(data_new['Outcome']==1)]['Glucose'],rng=(25,75)))
```

```
32.0
```

```
48.0
```

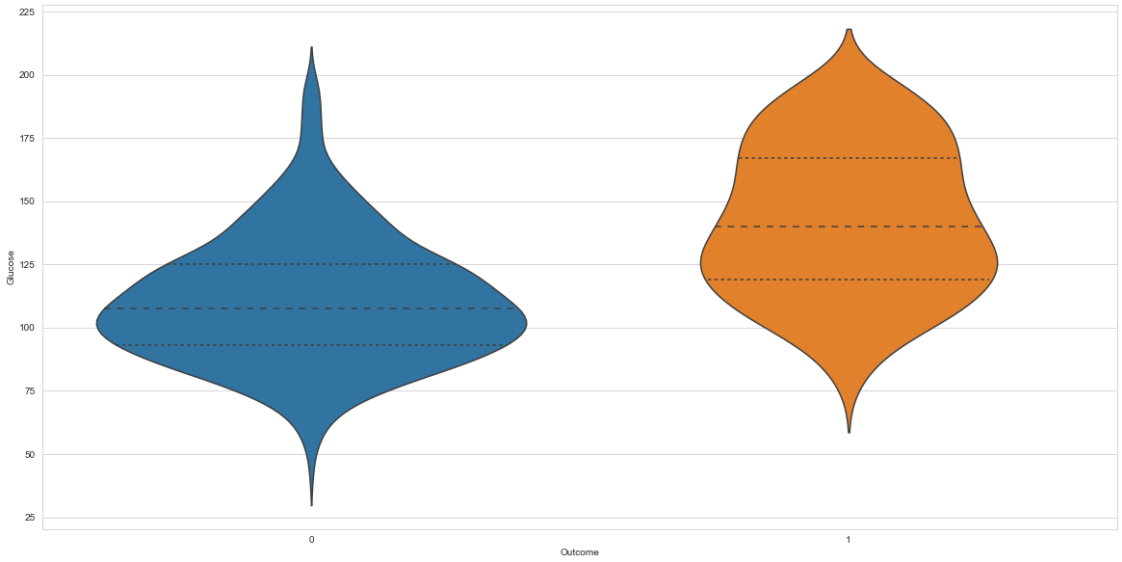
[29]:
```
print(iqr(data_new[(data_new['Outcome']==0)]['Insulin'],rng=(25,75)))

print(iqr(data_new[(data_new['Outcome']==1)]['Insulin'],rng=(25,75)))
```

```
74.5
136.75
```

Violin Plots

[30]:
```
plt.figure(figsize=(20, 10))
sns.violinplot(x='Outcome',y='Glucose',data=data_new,inner='quartile')
```

[30]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd1871c4050>
```



[31]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score,confusion_matrix,precision_score
from sklearn.metrics import recall_score,f1_score, roc_auc_score, roc_curve
from sklearn.metrics import confusion_matrix, make_scorer, classification_report
from sklearn.model_selection import train_test_split, cross_val_score,␣
↪RandomizedSearchCV
```

[32]:
```
X = data.drop('Outcome',axis = 1)
y = data['Outcome']
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size = 0.
↪25,random_state = 200)
```

[33]:
```
clf = RandomForestClassifier(max_depth=10, oob_score = True,n_jobs =␣
↪-1,random_state = 100)
clf.fit(X_train,y_train)
```

[33]:
```
RandomForestClassifier(max_depth=10, n_jobs=-1, oob_score=True,
                       random_state=100)
```

[34]:
```
scores = cross_val_score(clf,X_train,y_train,cv = 5,scoring = "f1_macro")
print("Cross Validation Score: ",round(scores.mean(), 3))
```

```
Cross Validation Score:  0.69
```

[35]:
```
train_pred = clf.predict(X_train)
test_pred = clf.predict(X_test)

print("Training F1 score: ",f1_score(train_pred,y_train))
print("Testing F1 score : ",f1_score(test_pred,y_test))
```

```
Training F1 score:  0.9976019184652278
Testing F1 score :  0.7107438016528926
```

[36]:
```
parameters = {
              "max_depth":[5,6,7,8,9,10],
              "n_estimators":[100,104,106,107,108,109],
              "min_samples_split":[3,4,5,6,7,8],
              "min_samples_leaf":[2,4,6,8,9,10]
              }

scorer = make_scorer(f1_score)
```

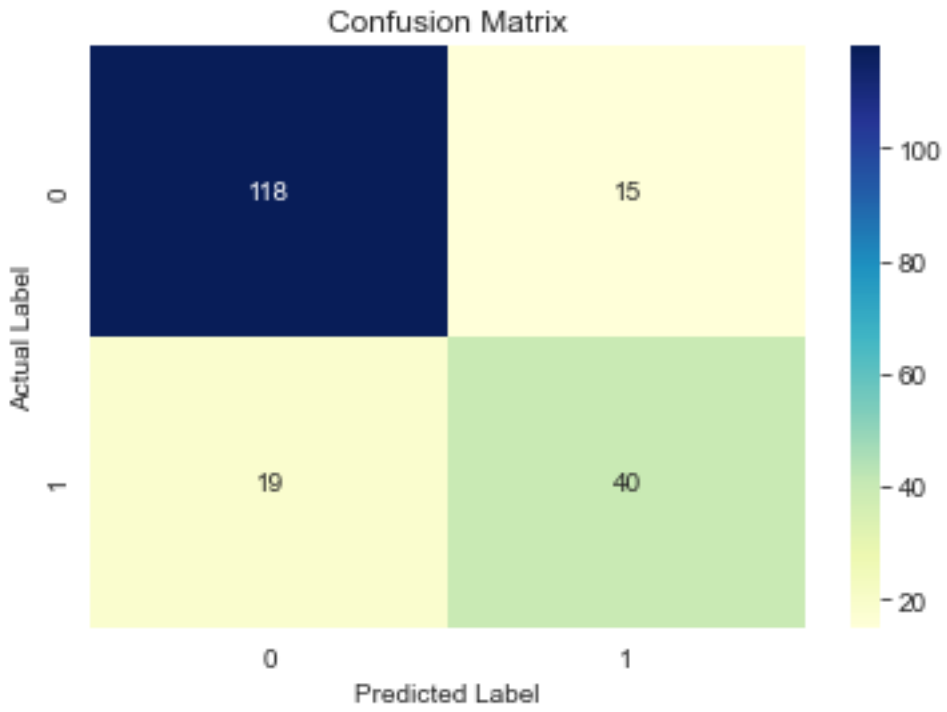
[37]:
```
clf1 = RandomizedSearchCV(clf,parameters,scoring = scorer)
clf1.fit(X_train,y_train)
best_clf_random = clf1.best_estimator_
best_clf_random.fit(X_train,y_train)
```

[37]:
```
RandomForestClassifier(max_depth=5, min_samples_leaf=8, min_samples_split=5,
                       n_jobs=-1, oob_score=True, random_state=100)
```

[38]:
```
train_pred = best_clf_random.predict(X_train)
test_pred = best_clf_random.predict(X_test)
```

[39]:
```
cnf_matrix = confusion_matrix(y_test,test_pred)
p = sns.heatmap(pd.DataFrame(cnf_matrix),annot = True,cmap = "YlGnBu",fmt = 'g')
plt.title("Confusion Matrix",y = 1.1)
plt.xlabel('Predicted Label')
plt.ylabel('Actual Label')
```

[39]: Text(34.0, 0.5, 'Actual Label')



[40]:
```
print("Training Accuracy    : ", round(accuracy_score(y_train, train_pred),3))
print("Training Precision   : ", round(precision_score(y_train, train_pred),3))
print("Training Recall      : ", round(recall_score(y_train, train_pred),3))
print("Training F1 score is : ", round(f1_score(train_pred,y_train),3))
```

```
Training Accuracy    :  0.828
Training Precision   :  0.812
Training Recall      :  0.684
Training F1 score is :  0.743
```

[41]:
```
print("Testing Accuracy    : ", round(accuracy_score(y_test, test_pred),3))
print("Testing Precision   : ", round(precision_score(y_test, test_pred),3))
print("Testing Recall      : ", round(recall_score(y_test, test_pred),3))
print("Testing F1 score is : ", round(f1_score(test_pred,y_test),3))
```

```
Testing Accuracy    :  0.823
Testing Precision   :  0.727
Testing Recall      :  0.678
Testing F1 score is :  0.702
```

[42]:
```
print("ROC Score: ",round(roc_auc_score(y_test,best_clf_random.
 ↪predict_proba(X_test)[:,1]),3))
```

```
ROC Score:  0.875
```

[43]:
```
print("Classification Report")
print(classification_report(y_test, test_pred, target_names=['class 0', 'class␣
↪1']))
```

```
Classification Report
              precision    recall  f1-score   support

     class 0       0.86      0.89      0.87       133
     class 1       0.73      0.68      0.70        59

    accuracy                           0.82       192
   macro avg       0.79      0.78      0.79       192
weighted avg       0.82      0.82      0.82       192
```

[44]:
```
fpr,tpr,thresholds = roc_curve(y_test,best_clf_random.predict_proba(X_test)[:
↪,1])
plt.plot([0,1],[0,1],'k--')
plt.plot(fpr,tpr)
plt.xlabel('fpr')
plt.ylabel('tpr')
plt.show()
```