Health Care Capstone Project

The objective of the dataset is to diagnostically predict whether or not a patient has diabetes, based on certain diagnostic measurements included in the dataset. Several constraints were placed on the selection of these instances from a larger database. In particular, all patients here are females at least 21 years old of Pima Indian heritage.

Importing all the required libraries

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
df = pd.read_csv('T:\Masters In Data Science\Capstone Project\Project 2\Healthcare - Diabetes\health care diabetes.csv')
```

In [3]:
```
df.head()
```

Out[3]:

	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

In [4]:
```
df.shape
```

Out[4]:
```
(768, 9)
```

In [5]:
```
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]:
```
df.columns
```

Out[6]:
```
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
      dtype='object')
```

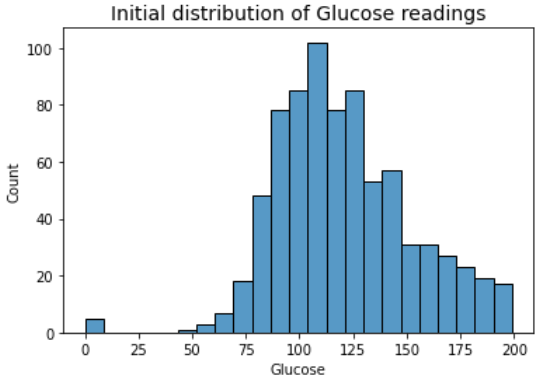
In [7]:
```
print('For Insulin',df[df['Insulin'].values == 0].count())

print('For Glucose',df[df['Glucose'].values == 0].count())

print('For SkinThickness',df[df['SkinThickness'].values == 0].count())

print('For Blood Pressure',df[df['BloodPressure'].values == 0].count())

print('For BMI',df[df['BMI'].values == 0].count())
```

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

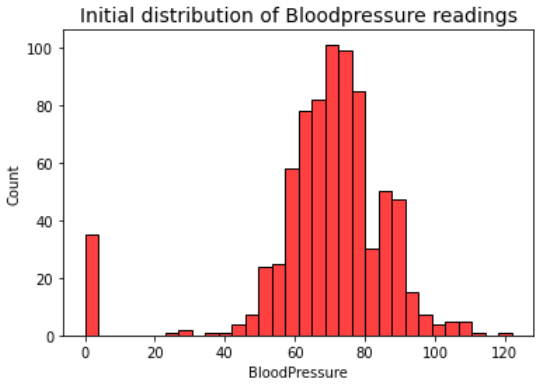
As told the values which are 0 are actually null values BMI, Blood_Pressure, Glucose have less amount of null values but Insulin and Skin Thickness variables have significantly more number of null values we have to treat them accordingly

In [8]:
```
import seaborn as sns
```

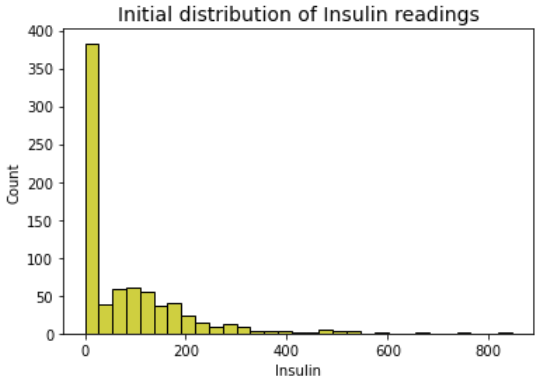
In [9]:
```
sns.histplot(data=df['Glucose'],palette='rainbow')
plt.title('Initial distribution of Glucose readings',fontsize=14)
plt.show()
```



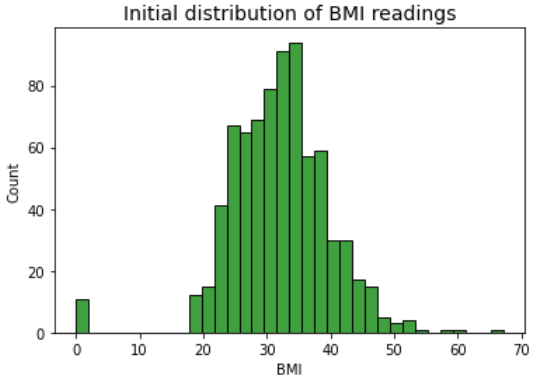
In [10]:
```
sns.histplot(data=df['BloodPressure'],color='r',palette='rainbow')
plt.title('Initial distribution of Bloodpressure readings',fontsize=14)
plt.show()
```



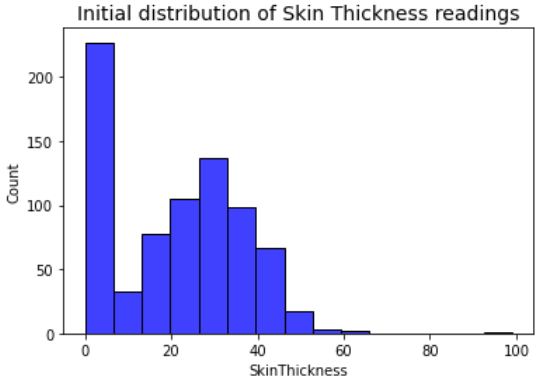
In [11]:
```
sns.histplot(data=df['Insulin'],color='y',palette='rainbow')
plt.title('Initial distribution of Insulin readings',fontsize=14)
plt.show()
```



In [12]:
```
sns.histplot(data=df['BMI'],color='g',palette='rainbow')
plt.title('Initial distribution of BMI readings',fontsize=14)
plt.show()
```



In [13]:
```
sns.histplot(data=df['SkinThickness'],color='b',palette='rainbow')
plt.title('Initial distribution of Skin Thickness readings',fontsize=14)
plt.show()
```



For all the variables replacing null values with median of that particular variable.

In [14]:
```
df['Glucose']=np.where(df.Glucose==0,df.Glucose.median(),df.Glucose)
```

In [15]:
```
df['BloodPressure']=np.where(df.BloodPressure==0,df.BloodPressure.median(),df.BloodPressure)
```

In [16]:
```
df['BMI']=np.where(df.BMI==0,df.BMI.median(),df.BMI)
```

In [17]:
```
df['SkinThickness']=np.where(df.SkinThickness==0,df.SkinThickness.median(),df.SkinThickness)
```

In [18]:
```
df['Insulin']=np.where(df.Insulin==0,df.Insulin.median(),df.Insulin)
```

In [19]:
```
df['Insulin'].head()
```

Out[19]:
```
0     30.5
1     30.5
2     30.5
3     94.0
4    168.0
Name: Insulin, dtype: float64
```

In [20]:
```
df['SkinThickness'].head()
```

Out[20]:
```
0    35.0
1    29.0
2    23.0
3    23.0
4    35.0
Name: SkinThickness, dtype: float64
```

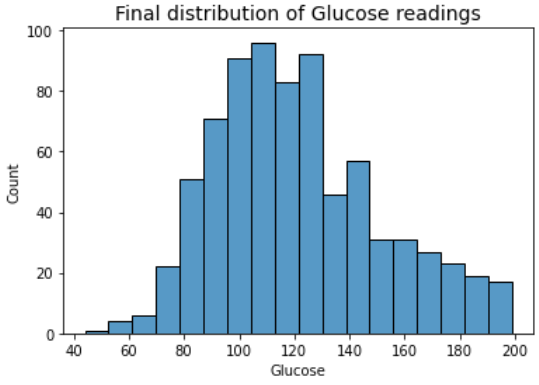
In [21]:
```
df.describe()
```

Out[21]:

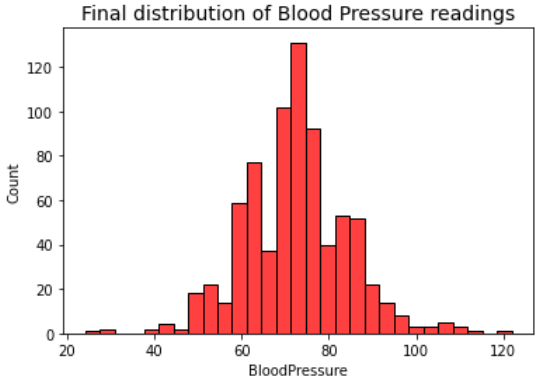
	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	121.656250	72.386719	27.334635	94.652344	32.450911	0.471876	33.240885	0.348958
std	3.369578	30.438286	12.096642	9.229014	105.547598	6.875366	0.331329	11.760232	0.476951
min	0.000000	44.000000	24.000000	7.000000	14.000000	18.200000	0.078000	21.000000	0.000000
25%	1.000000	99.750000	64.000000	23.000000	30.500000	27.500000	0.243750	24.000000	0.000000
50%	3.000000	117.000000	72.000000	23.000000	31.250000	32.000000	0.372500	29.000000	0.000000
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.000000	1.000000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	1.000000

Again checking the distribution of variables

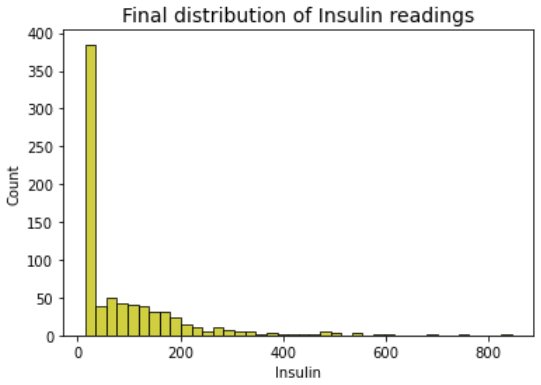
In [22]:
```
sns.histplot(data=df['Glucose'],palette='rainbow')
plt.title('Final distribution of Glucose readings',fontsize=14)
plt.show()
```



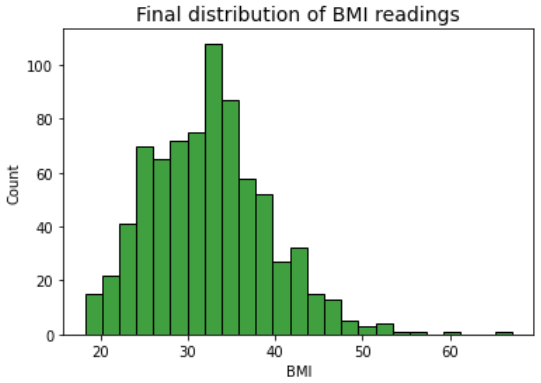
In [23]:
```
sns.histplot(data=df['BloodPressure'],color='r',palette='rainbow')
plt.title('Final distribution of Blood Pressure readings',fontsize=14)
plt.show()
```



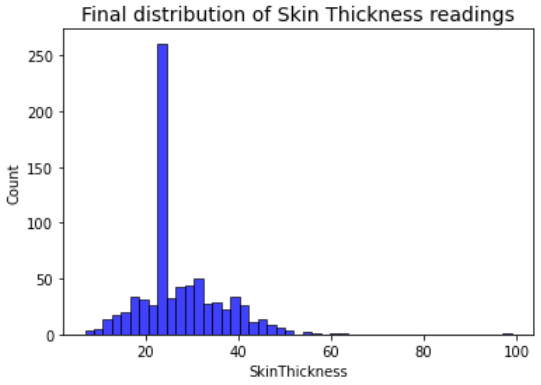
In [24]:
```
sns.histplot(data=df['Insulin'],color='y',palette='rainbow')
plt.title('Final distribution of Insulin readings',fontsize=14)
plt.show()
```



In [25]:
```
sns.histplot(data=df['BMI'],color='g',palette='rainbow')
plt.title('Final distribution of BMI readings',fontsize=14)
plt.show()
```



In [26]:
```
sns.histplot(data=df['SkinThickness'],color='b',palette='rainbow')
plt.title('Final distribution of Skin Thickness readings',fontsize=14)
plt.show()
```

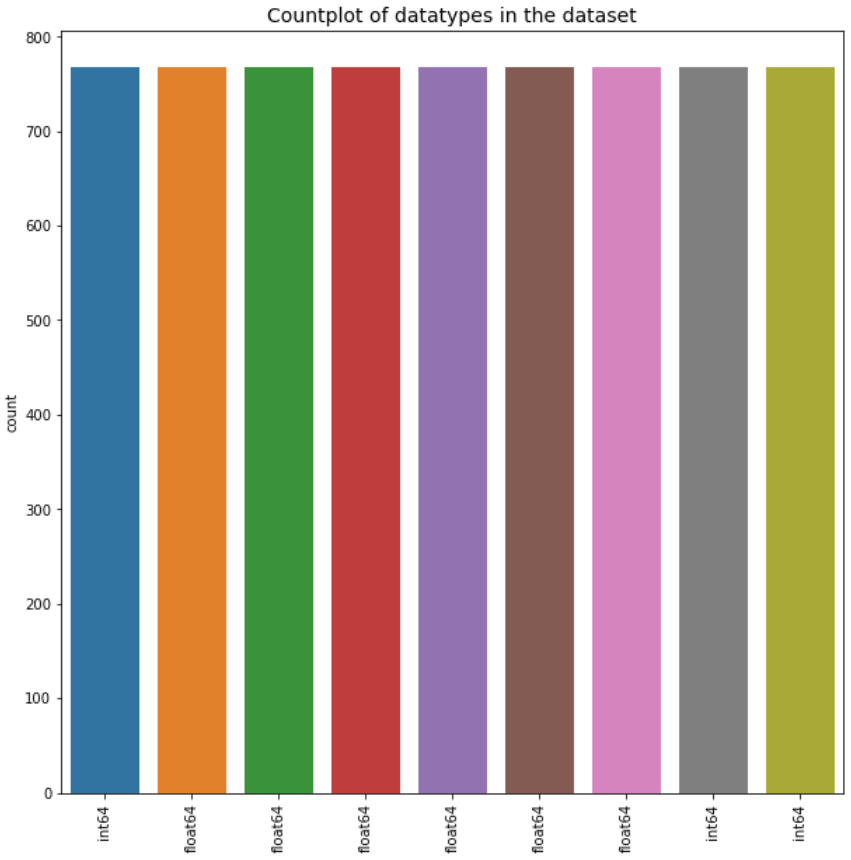


In [27]:
```
df.columns
```

Out[27]:
```
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
      dtype='object')
```

Create a count plot describing the data types and the count of variables.

In [28]:
```
plt.figure(figsize=(10,10))
sns.countplot(data=df).set_xticklabels((df.dtypes),rotation=90)
plt.title('Countplot of datatypes in the dataset',fontsize=14)
plt.show()
```



In [29]:
```
df.head()
```

Out[29]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148.0	72.0	35.0	30.5	33.6	0.627	50	1
1	1	85.0	66.0	29.0	30.5	26.6	0.351	31	0
2	8	183.0	64.0	23.0	30.5	23.3	0.672	32	1
3	1	89.0	66.0	23.0	94.0	28.1	0.167	21	0
4	0	137.0	40.0	35.0	168.0	43.1	2.288	33	1

In [30]:
```
df['SkinThickness'] = round(df.SkinThickness,2)
```

In [31]:
```
df['Insulin'] = round(df.Insulin,2)
```

Check the balance of the data by plotting the count of outcomes by their value

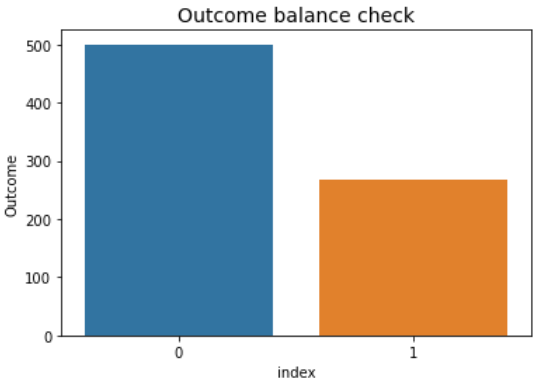
In [32]:
```
balance = df['Outcome'].value_counts().reset_index()
```

In [33]:
```
balance
```

Out[33]:

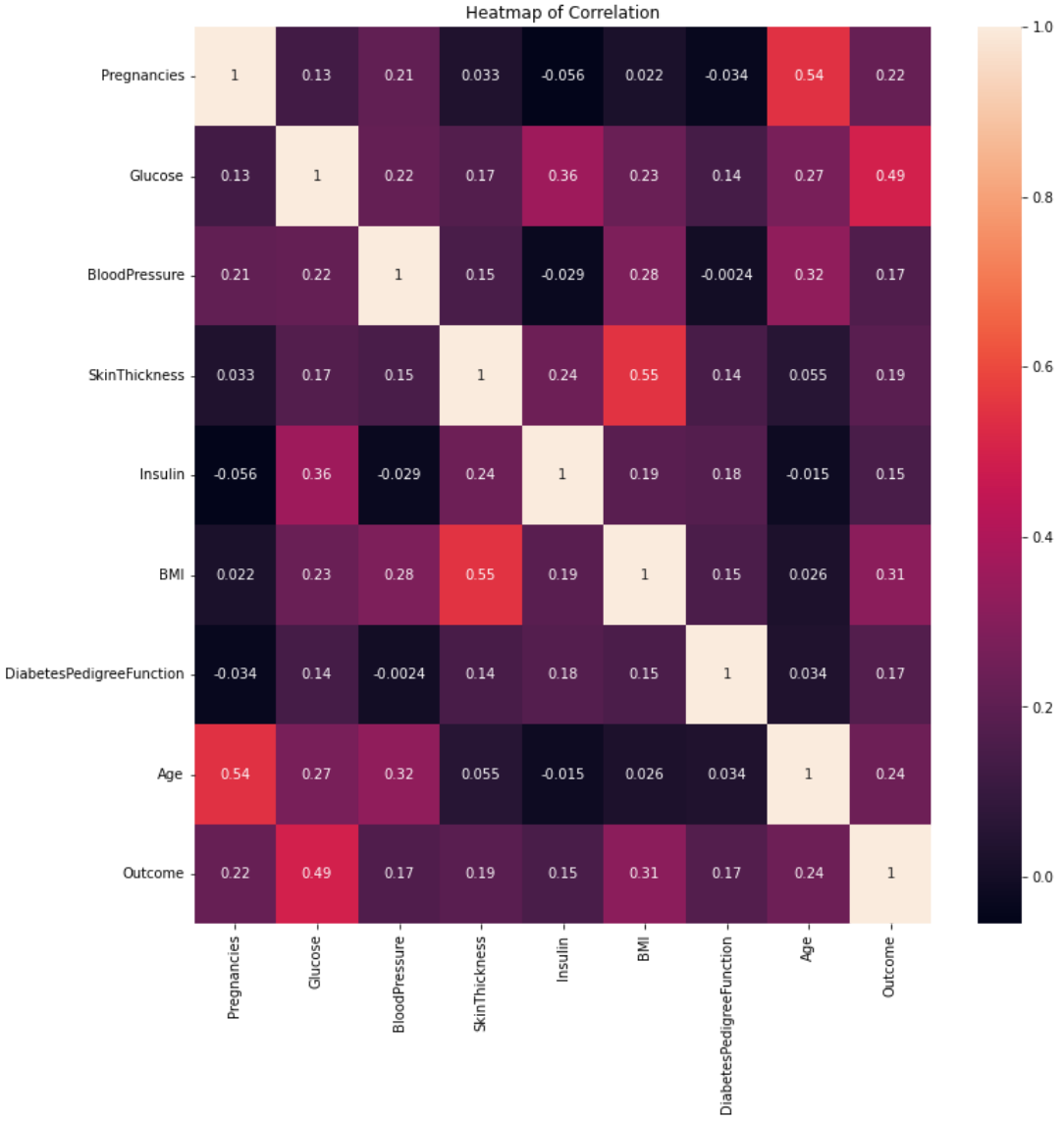
	index	Outcome
0	0	500
1	1	268

In [34]:
```
sns.barplot(x='index',y='Outcome',data=balance)
plt.title('Outcome balance check',fontsize=14)
plt.show()
```



From above plot it is clear that the data is imbalanced

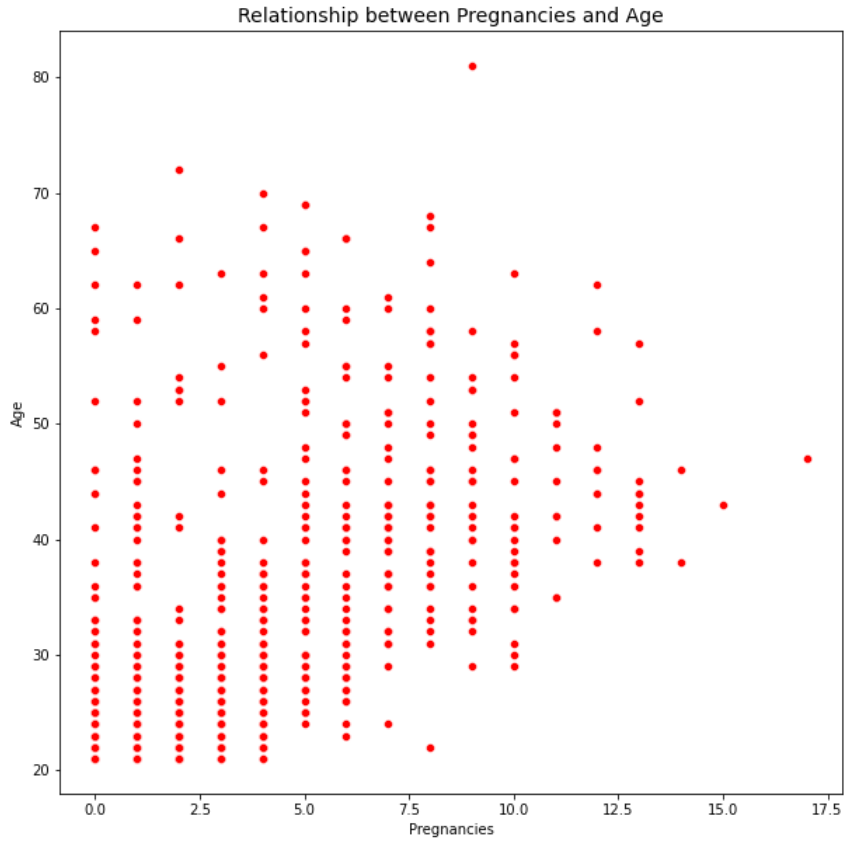
In [35]:
```
plt.figure(figsize=(12,12))
sns.heatmap(df.corr(),annot=True)
plt.title('Heatmap of Correlation')
plt.show()
```



We can see that Skin thickness and BMI have high positive correlation also Age and pregnancies have high correlation

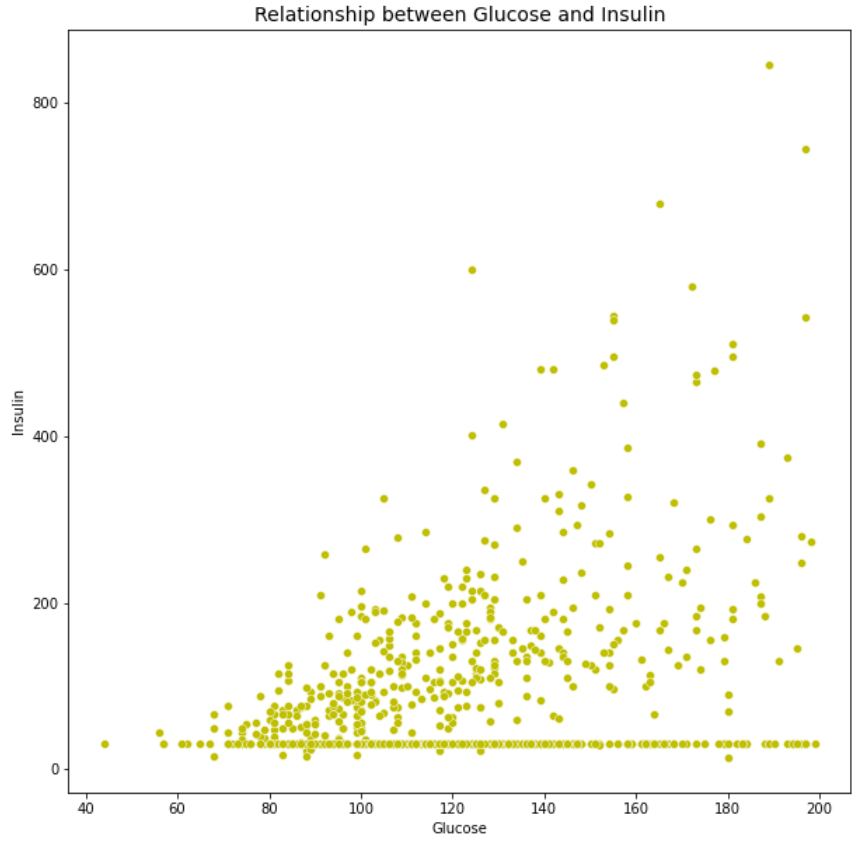
Checking the highly positively correlated variables relation by scatterplot

In [36]:
```
plt.figure(figsize=(10,10))
sns.scatterplot(x='Pregnancies',y='Age',data=df,color='r')
plt.title('Relationship between Pregnancies and Age',fontsize=14)
plt.show()
```



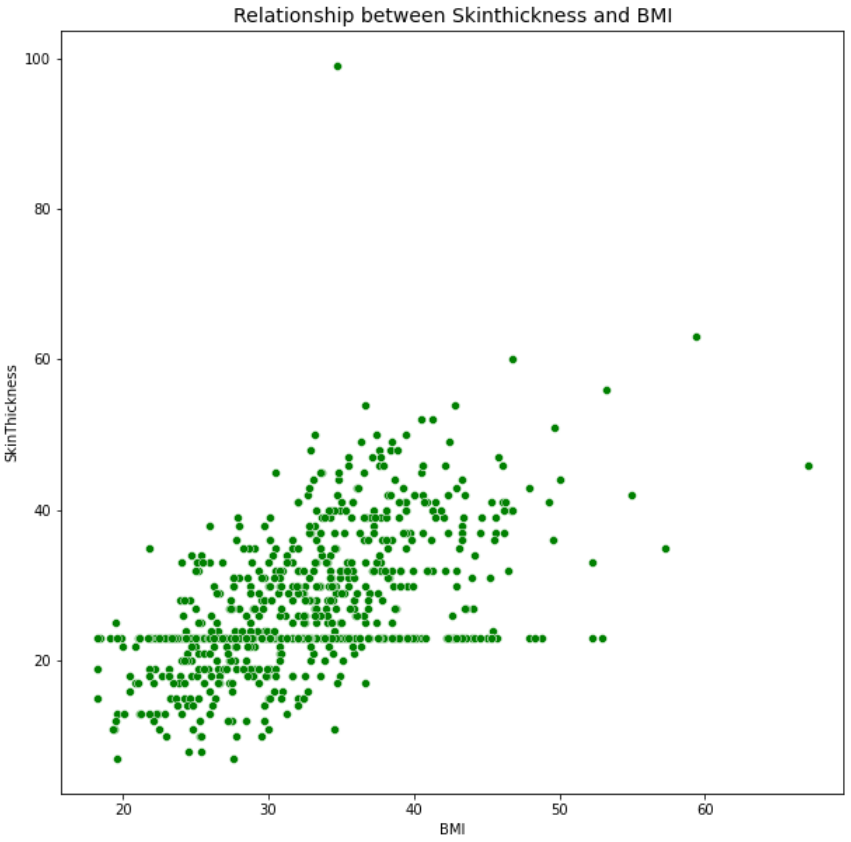
we can see as age increases number of pregnancies also increases

In [37]:
```
plt.figure(figsize=(10,10))
sns.scatterplot(x='Glucose',y='Insulin',data=df,color='y')
plt.title('Relationship between Glucose and Insulin',fontsize=14)
plt.show()
```



Relationship between Glucose and Insuline is somewhat linear

In [38]:
```
plt.figure(figsize=(10,10))
sns.scatterplot(x='BMI',y='SkinThickness',data=df,color='g')
plt.title('Relationship between Skinthickness and BMI',fontsize=14)
plt.show()
```



Similarly relation between Skinthickness and BMI is also somewhat linear

Data Preprocessing

In [39]:
```
df.columns
```

Out[39]:
```
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
      dtype='object')
```

In [40]:
```
X = df[['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
        'BMI', 'DiabetesPedigreeFunction', 'Age']]
Y = df['Outcome']
```

In [41]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(X,Y,random_state=500,test_size=0.3)
```

KNN model

In [42]:
```
from sklearn.neighbors import KNeighborsClassifier
KNN = KNeighborsClassifier(n_neighbors=27)
```

In [43]:
```
KNN_model = KNN.fit(x_test,y_test)
```

In [44]:
```
y_pred_KNN = KNN.predict(x_test)
y_pred_KNN
```

Out[44]:
```
array([0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0,
       0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0,
       0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0,
       1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
       0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0,
       1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0,
       1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0], dtype=int64)
```

In [45]:
```
from sklearn.metrics import classification_report
```

In [46]:
```
print(classification_report(y_test,y_pred_KNN))
```

```
              precision    recall  f1-score   support

           0       0.74      0.92      0.82       153
           1       0.70      0.38      0.50        78

    accuracy                           0.74       231
   macro avg       0.72      0.65      0.66       231
weighted avg       0.73      0.74      0.71       231
```

KNN model is having 74% Accuracy

Now we will try Logistic Regression model

In [47]:
```
from sklearn.metrics import accuracy_score
```

In [48]:
```
from sklearn.linear_model import LogisticRegression
logm = LogisticRegression()
```

In [49]:
```
Logistic_regression_model = logm.fit(x_train,y_train)
```

In [50]:
```
y_pred_logm = logm.predict(x_test)
```

In [51]:
```
y_pred_logm
```

Out[51]:
```
array([1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0,
       0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0,
       0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
       1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0,
       0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0,
       1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0,
       1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1,
       0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0], dtype=int64)
```

In [52]:
```
from sklearn.metrics import classification_report
print(classification_report(y_test,y_pred_logm))
```

```
              precision    recall  f1-score   support

           0       0.81      0.89      0.85       153
           1       0.73      0.60      0.66        78

    accuracy                           0.79       231
   macro avg       0.77      0.75      0.76       231
weighted avg       0.79      0.79      0.79       231
```

Logistic Regression model has 79% Accuracy

Now we will try Decision tree model

In [53]:
```
from sklearn.tree import DecisionTreeClassifier
dc = DecisionTreeClassifier()
```

In [54]:
```
Decision_tree_model = dc.fit(x_train,y_train)
y_pred_decision_tree = dc.predict(x_test)
```

In [55]:
```
y_pred_decision_tree
```

Out[55]:
```
array([1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0,
       0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0,
       0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0,
       0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1,
       1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0,
       0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1,
       0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
       1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0,
       1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1,
       1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0], dtype=int64)
```

In [56]:
```
print(classification_report(y_test,y_pred_decision_tree))
```

```
              precision    recall  f1-score   support

           0       0.81      0.78      0.79       153
           1       0.60      0.64      0.62        78

    accuracy                           0.73       231
   macro avg       0.70      0.71      0.71       231
weighted avg       0.74      0.73      0.73       231
```

Decision tree model has 73% Accuracy

Now we will try Random Forest Model

In [57]:
```
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(n_estimators=800)
```

In [58]:
```
Random_forest_model = rf.fit(x_train,y_train)
y_pred_random_forest = rf.predict(x_test)
```

In [59]:
```
y_pred_random_forest
```

Out[59]:
```
array([1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0,
       0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0,
       0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1,
       1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0,
       0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1,
       0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1,
       1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0,
       1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0], dtype=int64)
```

In [60]:
```
print(classification_report(y_test,y_pred_random_forest))
```

```
              precision    recall  f1-score   support

           0       0.83      0.85      0.84       153
           1       0.69      0.65      0.67        78

    accuracy                           0.78       231
   macro avg       0.76      0.75      0.75       231
weighted avg       0.78      0.78      0.78       231
```

We have achieved accuracy of 78% with Random forest model which is more than KNN model

In [61]:
```
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
kf = KFold(n_splits=20,random_state=25,shuffle=True)
```

In [62]:
```
scores = cross_val_score(rf,X,Y,scoring='accuracy',cv=kf,n_jobs=1)
scores
```

Out[62]:
```
array([0.76923077, 0.84615385, 0.71794872, 0.79487179, 0.82051282,
       0.74358974, 0.79487179, 0.82051282, 0.78947368, 0.71052632,
       0.68421053, 0.76315789, 0.76315789, 0.73684211, 0.78947368,
       0.73684211, 0.78947368, 0.73684211, 0.71052632, 0.73684211])
```

In [63]:
```
print('Accuracy : %.3f(%.3f)'%(np.mean(scores),np.std(scores)))
```

```
Accuracy : 0.763(0.042)
```

This much lower accuracy is not acceptable hence we will try scaling the data and deploy all models again

In [64]:
```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
```

In [65]:
```
X1=sc.fit_transform(X)
```

In [66]:
```
X11 = pd.DataFrame(X1)
```

In [67]:
```
X11.head()
```

Out[67]:

	0	1	2	3	4	5	6	7
0	0.639947	0.866045	-0.031990	0.831114	-0.608201	0.167240	0.468492	1.425995
1	-0.844885	-1.205066	-0.528319	0.180566	-0.608201	-0.851551	-0.365061	-0.190672
2	1.233880	2.016662	-0.693761	-0.469981	-0.608201	-1.331838	0.604397	-0.105584
3	-0.844885	-1.073567	-0.528319	-0.469981	-0.006185	-0.633239	-0.920763	-1.041549
4	-1.141852	0.504422	-2.679076	0.831114	0.695378	1.549885	5.484909	-0.020496

In [68]:
```
x1_train,x1_test,y1_train,y1_test = train_test_split(X11,Y,test_size=0.33)
```

KNN model

In [69]:
```
KNN_model_1 = KNN.fit(x1_train,y1_train)
y_pred_KNN_1 = KNN.predict(x1_test)
y_pred_KNN_1
```

Out[69]:
```
array([0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1,
       0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0,
       1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0,
       0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1,
       0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0,
       1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0,
       1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0,
       0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1,
       0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0,
       0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0], dtype=int64)
```

In [70]:
```
print(classification_report(y1_test,y_pred_KNN_1))
```

```
              precision    recall  f1-score   support

           0       0.84      0.88      0.86       174
           1       0.70      0.62      0.66        80

    accuracy                           0.80       254
   macro avg       0.77      0.75      0.76       254
weighted avg       0.79      0.80      0.80       254
```

KNN has 80% accuracy

Now we will try Logistic regression model

In [71]:
```
Logistic_regression_model_1 = logm.fit(x1_train,y1_train)
y_pred_logistic_regression_1 = logm.predict(x1_test)
y_pred_logistic_regression_1
```

Out[71]:
```
array([0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1,
       0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0,
       1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1,
       0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1,
       1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0,
       1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0,
       1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1,
       0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1,
       0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0,
       0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
       1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0], dtype=int64)
```

In [72]:
```
print(classification_report(y1_test,y_pred_logistic_regression_1))
```

```
              precision    recall  f1-score   support

           0       0.83      0.88      0.85       174
           1       0.70      0.61      0.65        80

    accuracy                           0.80       254
   macro avg       0.77      0.75      0.75       254
weighted avg       0.79      0.80      0.79       254
```

Logistic regression has 80% accuracy which is more than KNN

Now we will try Decision tree model

In [73]:
```
Decision_tree_model_1 = dc.fit(x1_train,y1_train)
y_pred_decision_tree_1 = dc.predict(x1_test)
y_pred_decision_tree_1
```

Out[73]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0,
       1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1,
       0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0,
       0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1,
       1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0,
       0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0,
       0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0,
       0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0,
       0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1,
       0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0,
       0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0,
       1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], dtype=int64)
```

In [74]:
```
print(classification_report(y1_test,y_pred_decision_tree_1))
```

```
              precision    recall  f1-score   support

           0       0.78      0.74      0.76       174
           1       0.49      0.55      0.52        80

    accuracy                           0.68       254
   macro avg       0.64      0.65      0.64       254
weighted avg       0.69      0.68      0.69       254
```

Decision tree has 68% accuracy which is less than KNN

Now we will try Random forest model

In [75]:
```
Random_forest_model_1 = rf.fit(x1_train,y1_train)
y_pred_random_forest_1 = rf.predict(x1_test)
y_pred_random_forest_1
```

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

In [76]:
```
print(classification_report(y1_test,y_pred_random_forest_1))
```

```
              precision    recall  f1-score   support

           0       0.85      0.82      0.83       174
           1       0.64      0.68      0.65        80

    accuracy                           0.78       254
   macro avg       0.74      0.75      0.74       254
weighted avg       0.78      0.78      0.78       254
```

random Forest has 78% accuracy which is less than KNN

In [77]:
```
from sklearn.metrics import confusion_matrix,ConfusionMatrixDisplay
```

In [90]:
```
conf = confusion_matrix(y1_test,y_pred_KNN_1)
```

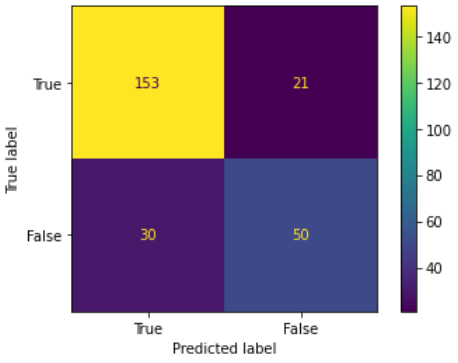
In [91]:
```
conf
```

Out[91]:
```
array([[153,  21],
       [ 30,  50]], dtype=int64)
```

Confusion Matrix for model with highest accuracy (KNN with scaling)

In [92]:
```
matrix = ConfusionMatrixDisplay(confusion_matrix=conf,display_labels=[True,False])
matrix.plot()
```

Out[92]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x23f9d6a7730>
```

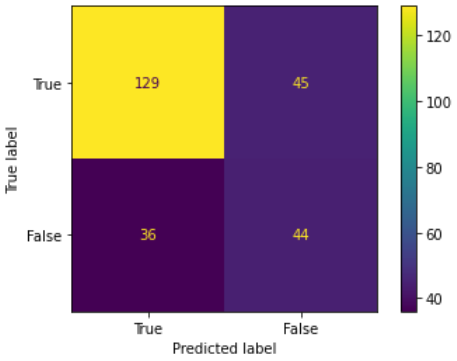


In [93]:
```
conf1 = confusion_matrix(y1_test,y_pred_decision_tree_1)
```

Confusion matrix for model with lowest accuracy (Decision Tree with Scaling)

In [94]:
```
matrix1 = ConfusionMatrixDisplay(confusion_matrix=conf1,display_labels=[True,False])
matrix1.plot()
```

Out[94]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x23f9d9333d0>
```



Collecting the results in a single dataframe

In [83]:
```
Results = [{'KNN_without_scaling': accuracy_score(y_test,y_pred_KNN),
            'Logistic_Regression_without_scaling': accuracy_score(y_test,y_pred_logm),
            'Decision_tree_without_scaling': accuracy_score(y_test,y_pred_decision_tree),
            'Random_forest_without_scaling': accuracy_score(y_test,y_pred_random_forest),
            '':'',
            'KNN_with_scaling': accuracy_score(y1_test,y_pred_KNN_1),
            'Logistic_Regression_with_scaling' : accuracy_score(y1_test,y_pred_logistic_regression_1),
            'Decision_tree_with_scaling' : accuracy_score(y1_test,y_pred_decision_tree_1),
            'Random_forest_with_scaling' : accuracy_score(y1_test,y_pred_random_forest_1)}]
```

In [84]:
```
Result = pd.DataFrame.from_dict(Results)
```

In [85]:
```
Result=np.transpose(Result)
```

In [86]:
```
Result = Result.reset_index()
```

In [87]:
```
col_names = ['Model','Accuracy']
Result.columns = col_names
```

In [88]:
```
Result
```

Out[88]:

	Model	Accuracy
0	KNN_without_scaling	0.735931
1	Logistic_Regression_without_scaling	0.792208
2	Decision_tree_without_scaling	0.731602
3	Random_forest_without_scaling	0.78355
4		
5	KNN_with_scaling	0.799213
6	Logistic_Regression_with_scaling	0.795276
7	Decision_tree_with_scaling	0.681102
8	Random_forest_with_scaling	0.775591

Converting the processed dataset df for further exploration purpose

In [89]:
```
df.to_csv('Health_care.csv')
```