Project_2

March 31, 2022

[1]:
```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
sns.set()
```

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

0.1 1. Perform descriptive analysis. Understand the variables and their corresponding values. On the columns below, a value of zero does not make sense and thus indicates missing value:

Glucose BloodPressure SkinThickness Insulin BMI

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

[3]:
```
   Pregnancies  Glucose  BloodPressure  SkinThickness  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

   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
```

[4]:
```python
df.info() # we can note that the data is of datatype int and float
```

```
<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
```

[5]:
```
df.describe() # we can note the summary of the dataset below
```

[5]:
```
       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
```

[6]:
```
df.isna().any() # Verified that no column has missing values
```

[6]:
```
Pregnancies                 False
Glucose                     False
BloodPressure               False
SkinThickness               False
Insulin                     False
BMI                         False
DiabetesPedigreeFunction    False
Age                         False
Outcome                     False
dtype: bool
```

[7]:
```python
df.isnull().sum()
```

[7]:
```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

[8]:
```python
# The five mentioned columns consists of zeros and we can obtain the count␣
↪using below
print('Missing value (zeros) in Columns')
print('Column Glucose: {}'.format((df['Glucose']==0).sum()))
print('Column BloodPressure: {}'.format((df['BloodPressure']==0).sum()))
print('Column SkinThickness: {}'.format((df['SkinThickness']==0).sum()))
print('Column Insulin: {}'.format((df['Insulin']==0).sum()))
print('Column BMI: {}'.format((df['BMI']==0).sum()))
```

```
Missing value (zeros) in Columns
Column Glucose: 5
Column BloodPressure: 35
Column SkinThickness: 227
Column Insulin: 374
Column BMI: 11
```

[9]:
```python
df.shape # shape of the dataset
```

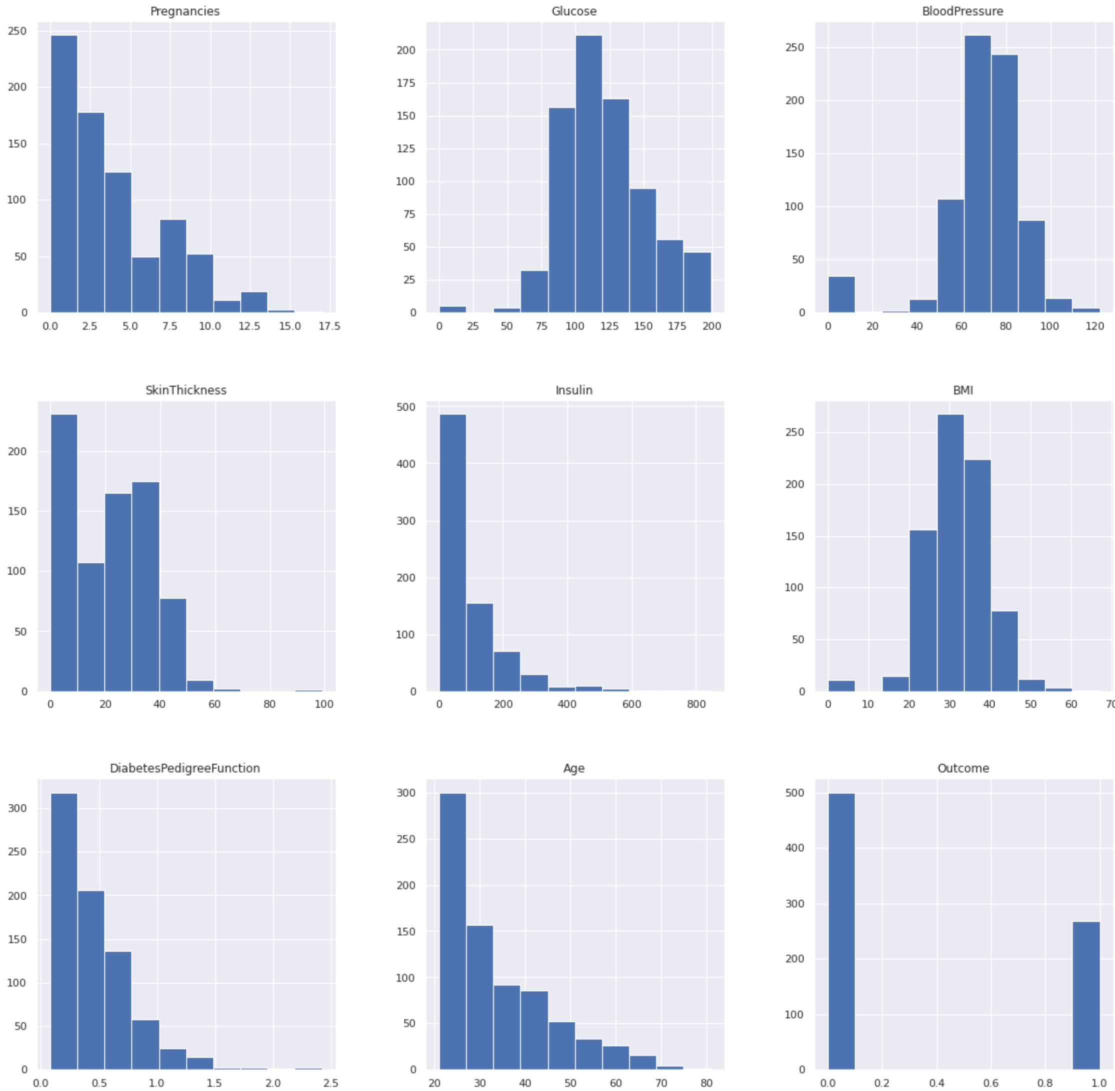
[9]:
```
(768, 9)
```

0.2 2. Visually explore these variables using histograms. Treat the missing values accordingly

[10]:
```python
df.hist(figsize = (20,20))
```

[10]:
```
array([[<AxesSubplot:title={'center':'Pregnancies'}>,
        <AxesSubplot:title={'center':'Glucose'}>,
        <AxesSubplot:title={'center':'BloodPressure'}>],
       [<AxesSubplot:title={'center':'SkinThickness'}>,
        <AxesSubplot:title={'center':'Insulin'}>,
        <AxesSubplot:title={'center':'BMI'}>],
       [<AxesSubplot:title={'center':'DiabetesPedigreeFunction'}>,
        <AxesSubplot:title={'center':'Age'}>,
```

```
<AxesSubplot:title={'center':'Outcome'}>]], dtype=object)
```



Glucose and BloodPressure are normally distributed SkinThickness, Insulin, Age and Diabetes-PedigreeFunction are skewed to Left

[11]:
```
df.head(10) # we can note that the specified columns comprises of zeros and␣
↪requires to be treated
```

[11]:
```
   Pregnancies  Glucose  BloodPressure  SkinThickness  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
```

```
5          5      116             74              0         0  25.6
6          3       78             50             32        88  31.0
7         10      115              0              0         0  35.3
8          2      197             70             45       543  30.5
9          8      125             96              0         0   0.0

   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
5                     0.201   30        0
6                     0.248   26        1
7                     0.134   29        0
8                     0.158   53        1
9                     0.232   54        1
```

[12]:
```
# The Zero in columns are replaced with NaN and then proceeded with imputation
```

[13]:
```
df1 = df.copy(deep=True)
```

[14]:
```
df1[['Glucose','BloodPressure','SkinThickness','Insulin','BMI']]=df1[['Glucose','BloodPressure
↪replace(0,np.NaN)
```

[15]:
```
df1.isnull().sum()
```

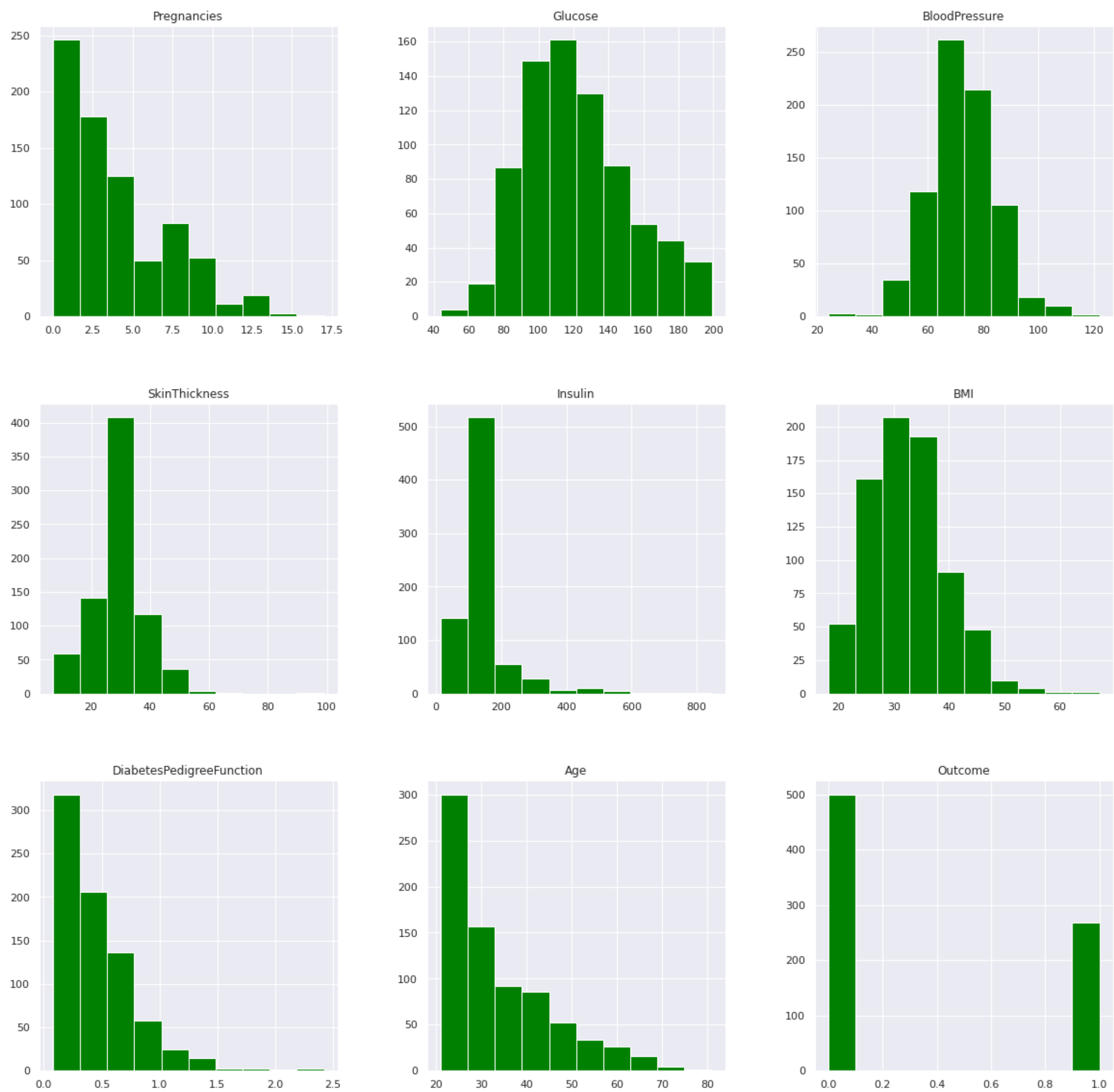
[15]:
```
Pregnancies                   0
Glucose                       5
BloodPressure                35
SkinThickness               227
Insulin                     374
BMI                          11
DiabetesPedigreeFunction      0
Age                           0
Outcome                       0
dtype: int64
```

Since Glucose and BloodPressure are normally distributed, we replace NaN with mean SkinThickness, Insulin are skewed, hence we replace NaN with median

[16]:
```
df1['Glucose'].fillna(df1['Glucose'].mean(),inplace=True)
df1['BloodPressure'].fillna(df1['BloodPressure'].mean(),inplace=True)
df1['SkinThickness'].fillna(df1['SkinThickness'].median(),inplace=True)
df1['Insulin'].fillna(df1['Insulin'].median(),inplace=True)
df1['BMI'].fillna(df1['BMI'].median(),inplace=True)
```

[17]:
```python
df1.hist(figsize = (20,20),color='green') # Plotting Histogram after treating␣
↪the missing values
```

[17]:
```
array([[<AxesSubplot:title={'center':'Pregnancies'}>,
        <AxesSubplot:title={'center':'Glucose'}>,
        <AxesSubplot:title={'center':'BloodPressure'}>],
       [<AxesSubplot:title={'center':'SkinThickness'}>,
        <AxesSubplot:title={'center':'Insulin'}>,
        <AxesSubplot:title={'center':'BMI'}>],
       [<AxesSubplot:title={'center':'DiabetesPedigreeFunction'}>,
        <AxesSubplot:title={'center':'Age'}>,
        <AxesSubplot:title={'center':'Outcome'}>]], dtype=object)
```



0.3 3. There are integer and float data type variables in this dataset. Create a count (frequency) plot describing the data types and the count of variables.

[18]:
```
df1.info() # Verified that there are integer and float data type in the dataset
```

```
<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
```

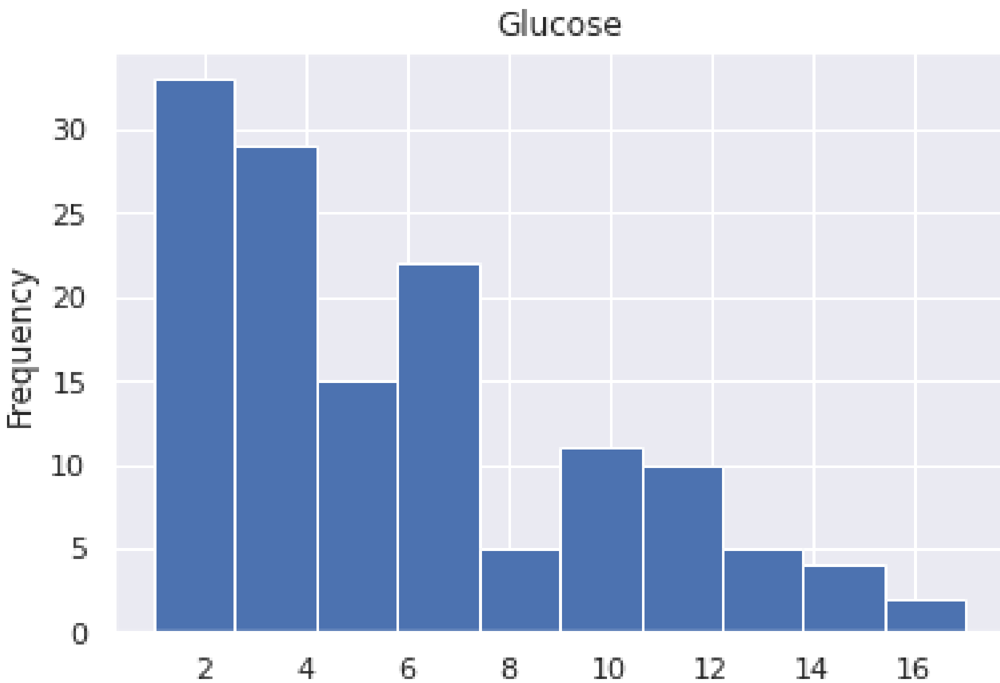
[19]:
```
series1=df1.apply(lambda x: x.value_counts()).T.stack()

series1 # count of variables
```

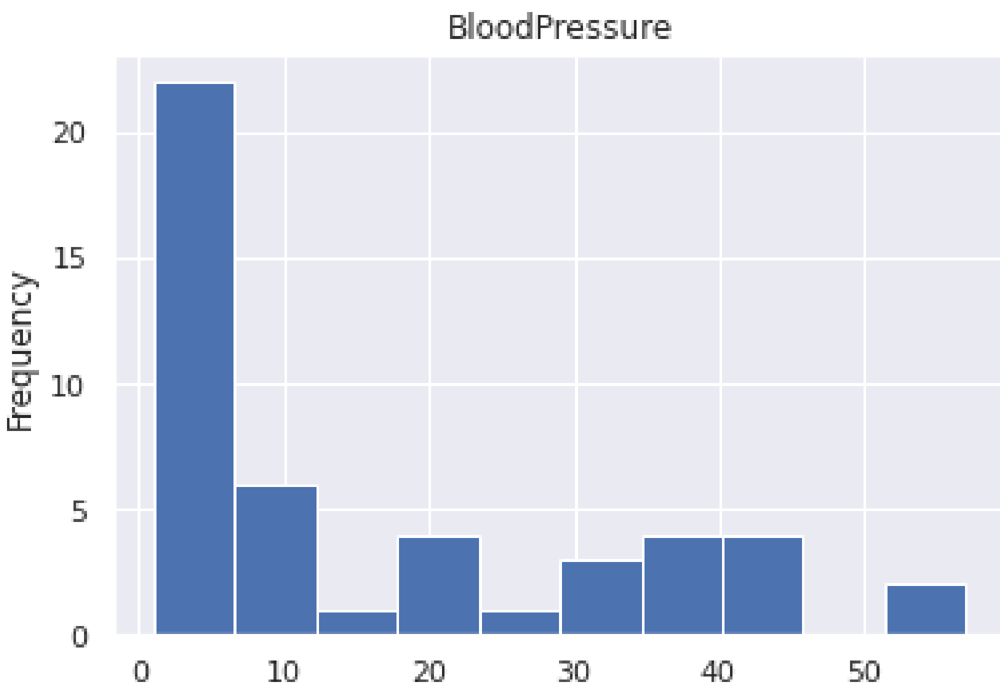
[19]:
```
Pregnancies  0.0     111.0
             1.0     135.0
             2.0     103.0
             3.0      75.0
             4.0      68.0
                      …
Age          70.0      1.0
             72.0      1.0
             81.0      1.0
Outcome      0.0     500.0
             1.0     268.0
Length: 1253, dtype: float64
```

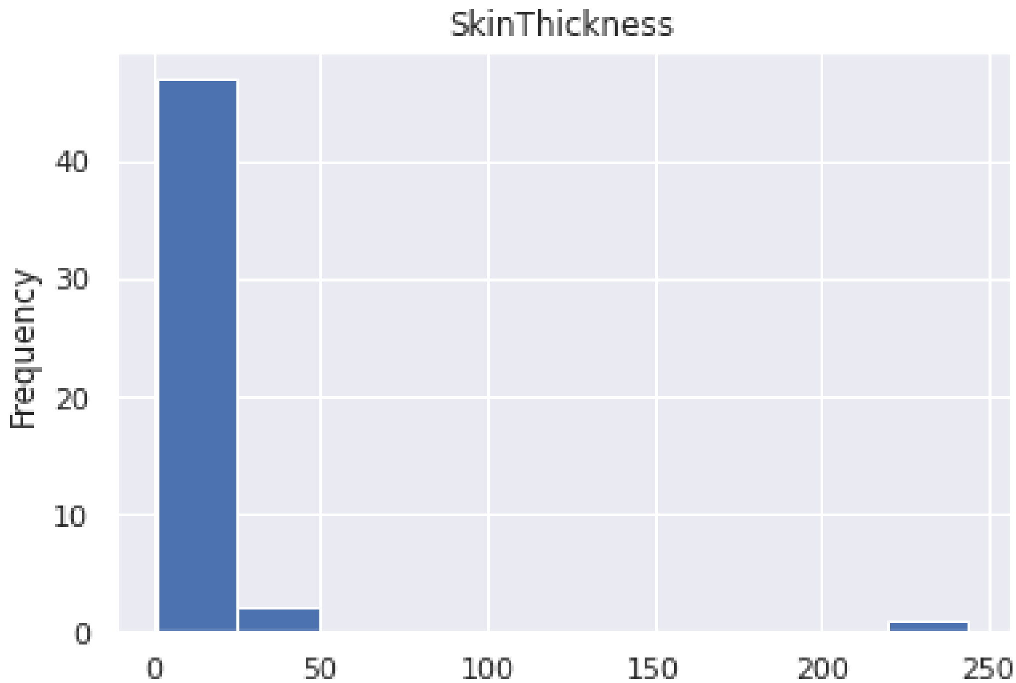
[20]:
```
# Count (Frequency plots for the variables)
```

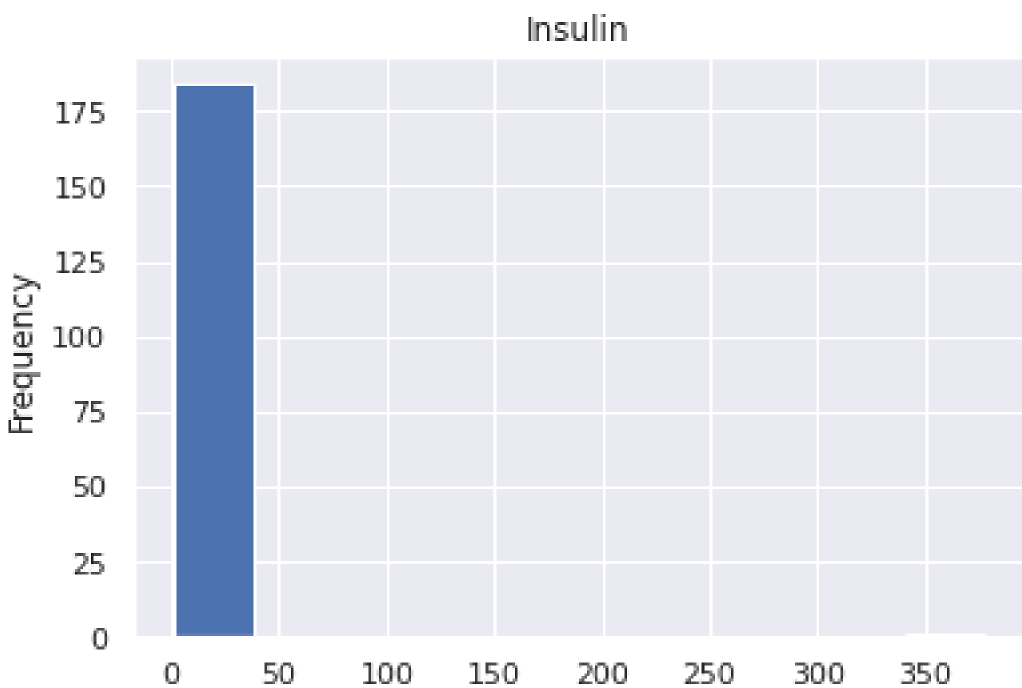
[21]:
```
df2=df1[['Glucose','BloodPressure','SkinThickness','Insulin','BMI']]
```

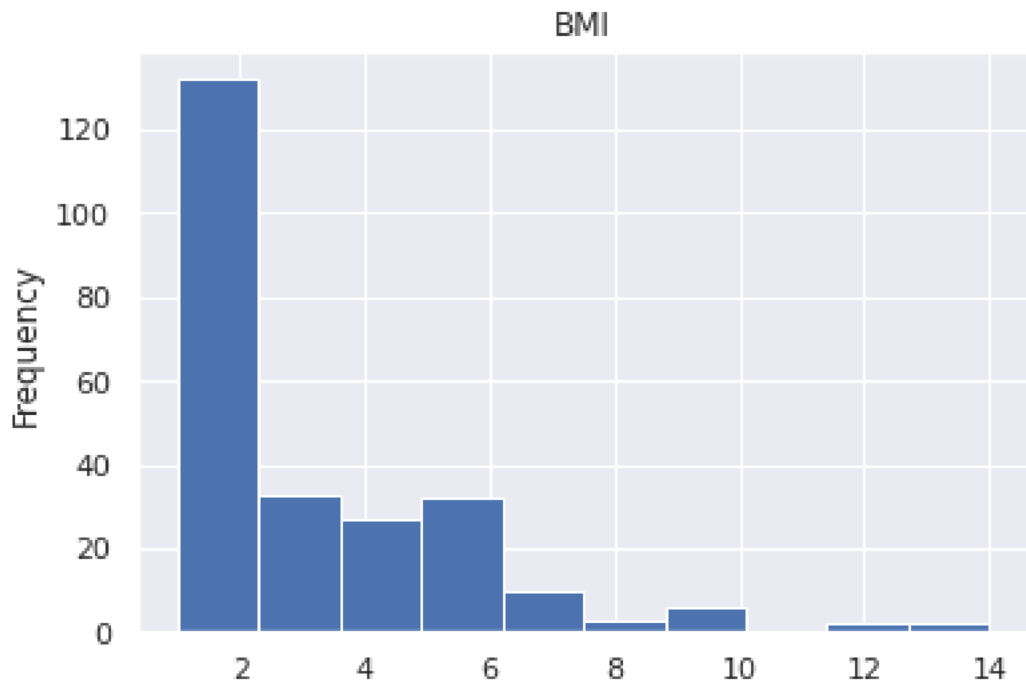
[22]:
```
for i in df2.columns:
    plt.figure()
    df2[i].value_counts().plot(kind='hist',title=i)
    plt.show()
```


Glucose
Frequency
0
5
10
15
20
25
30
2
4
6
8
10
12
14
16


BloodPressure
Frequency
0
5
10
15
20
0
10
20
30
40
50


SkinThickness
Frequency
0
10
20
30
40
0
50
100
150
200
250


Insulin
Frequency
0
25
50
75
100
125
150
175
0
50
100
150
200
250
300
350



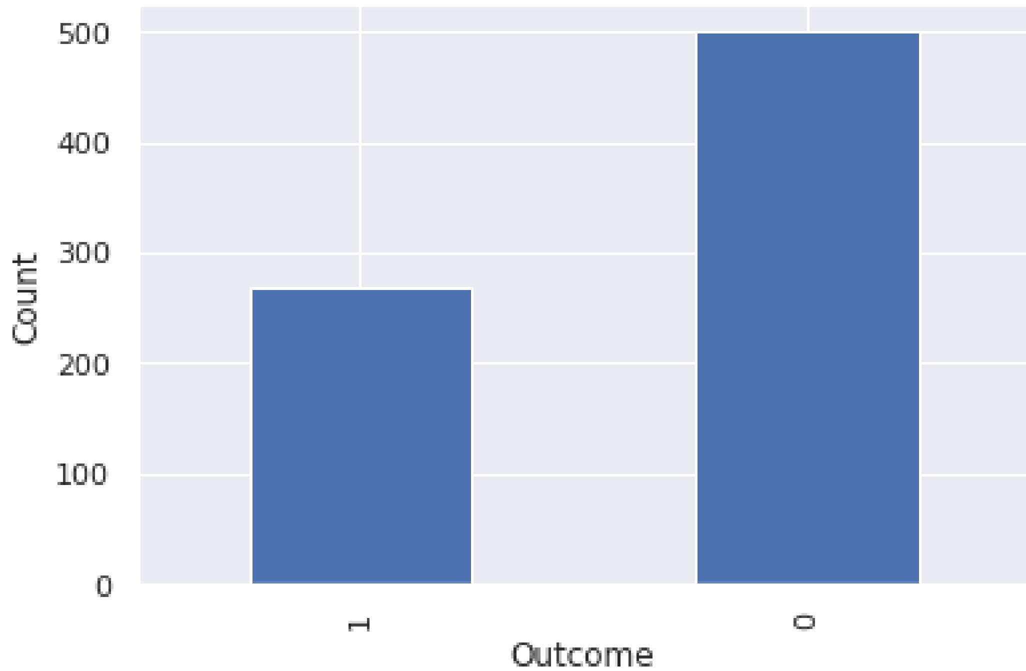
0.4 4. Check the balance of the data by plotting the count of outcomes by their value. Describe your findings and plan future course of action.

[23]:
```
df.Outcome.value_counts()
```

[23]:
```
0    500
1    268
Name: Outcome, dtype: int64
```

[24]:
```
df.Outcome.value_counts().sort_values().plot(kind = 'bar').
↪set(xlabel='Outcome',ylabel='Count')
```

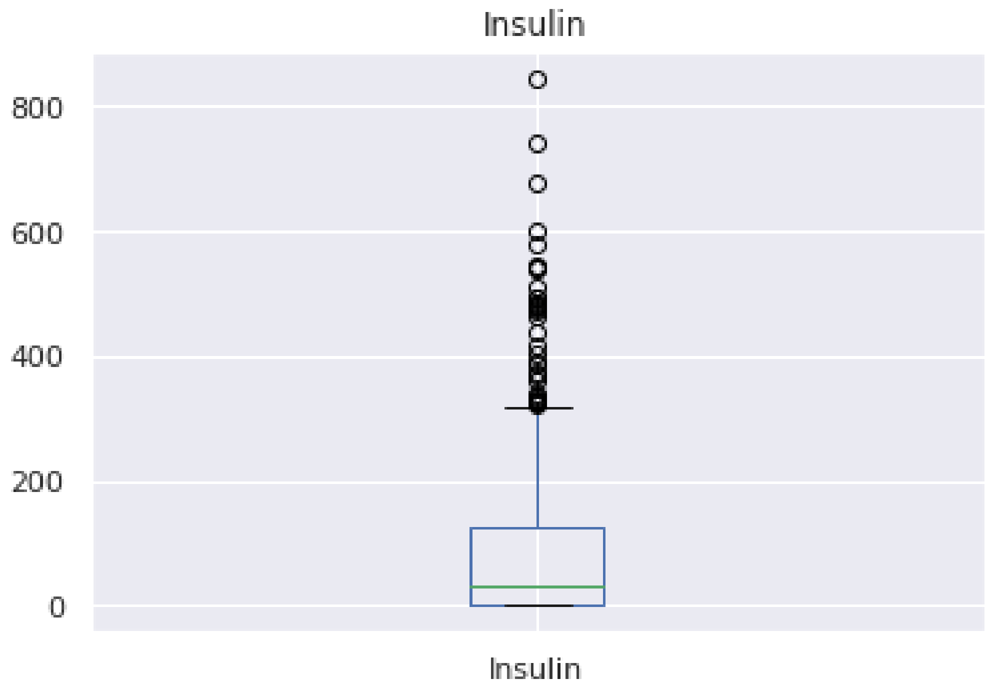
[24]:
```
[Text(0.5, 0, 'Outcome'), Text(0, 0.5, 'Count')]
```



Number of patients who are diabetic is half the Number of patients who are non-diabetic

[25]:
```
df['Insulin'].plot(kind='box',title='Insulin') # Outliers in Insulin are
↪visible in Box plot
```

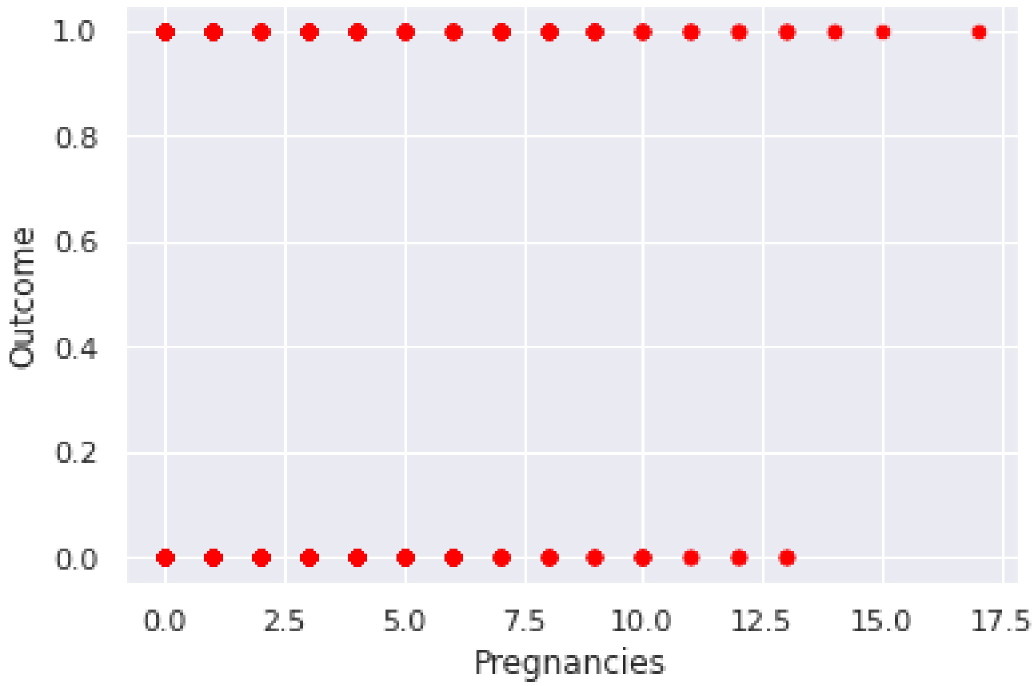
[25]: <AxesSubplot:title={'center':'Insulin'}>



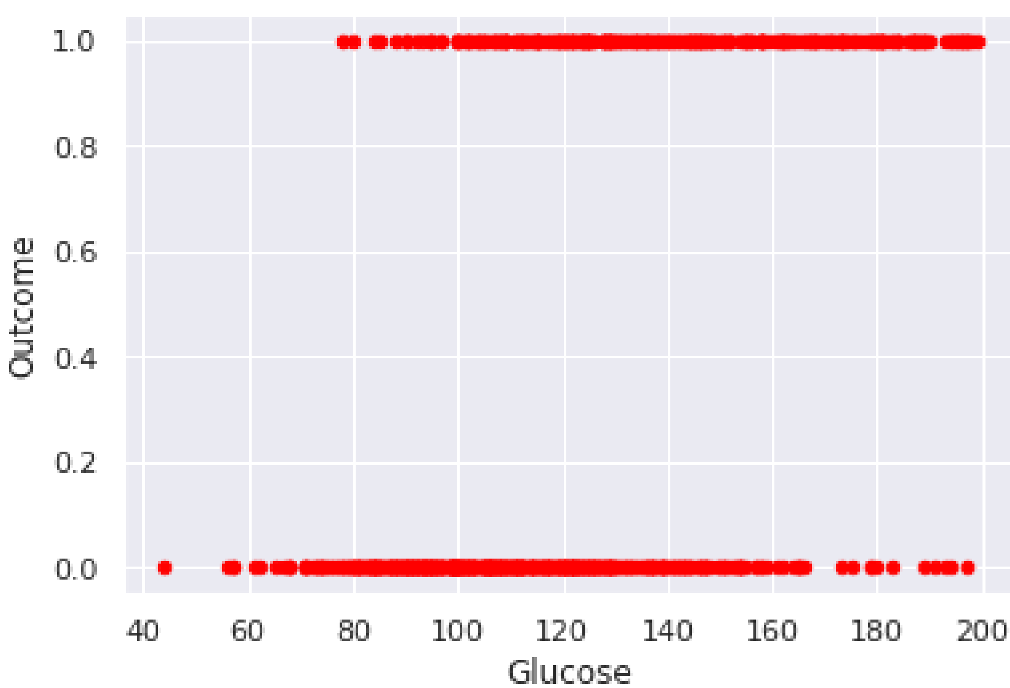
0.5 5. Create scatter charts between the pair of variables to understand the relationships. Describe your findings.

[26]:
```
var_list=['Pregnancies','Glucose','BloodPressure','SkinThickness','Insulin','BMI','Age','Diabe
for i in var_list:
    plt.figure()
    df1.plot(x=i, y='Outcome',kind='scatter',color='red')
    plt.show()
```

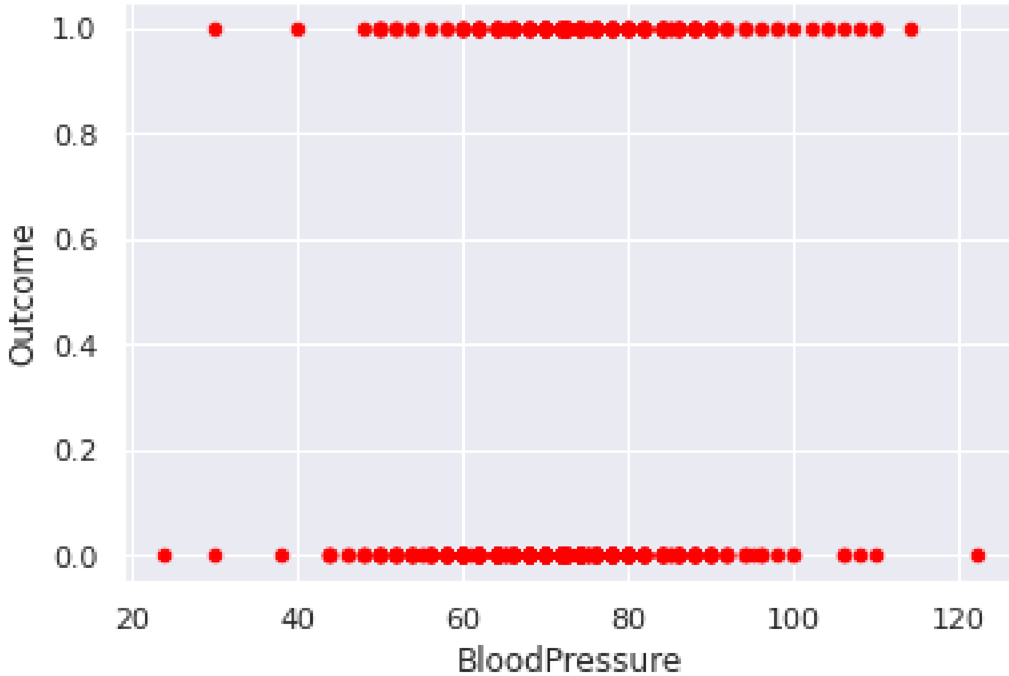
```
<Figure size 432x288 with 0 Axes>
```



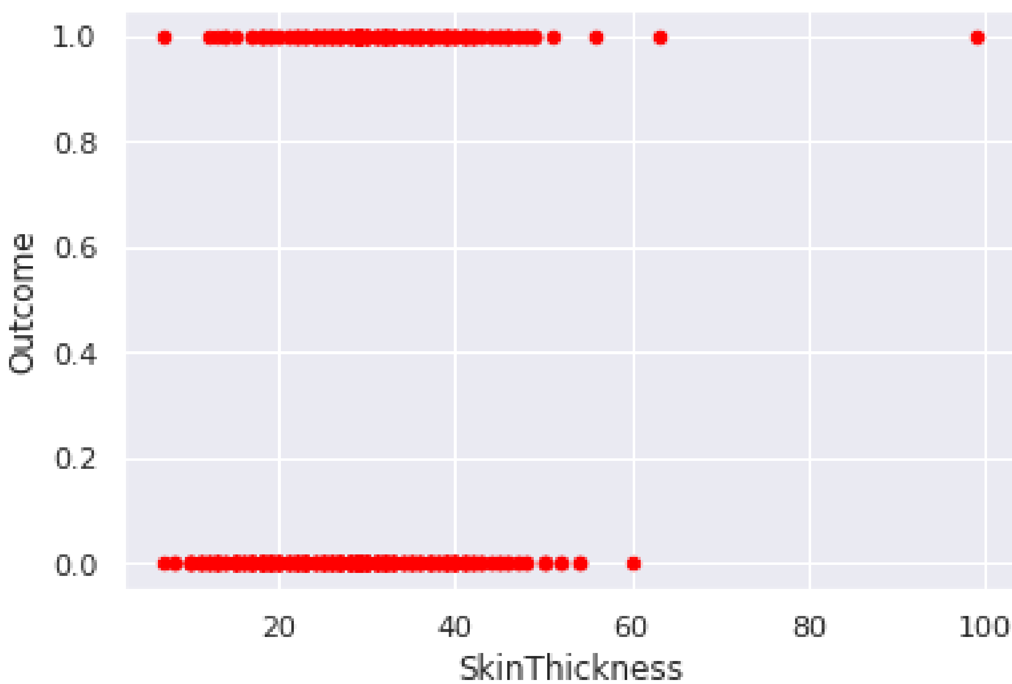
<Figure size 432x288 with 0 Axes>



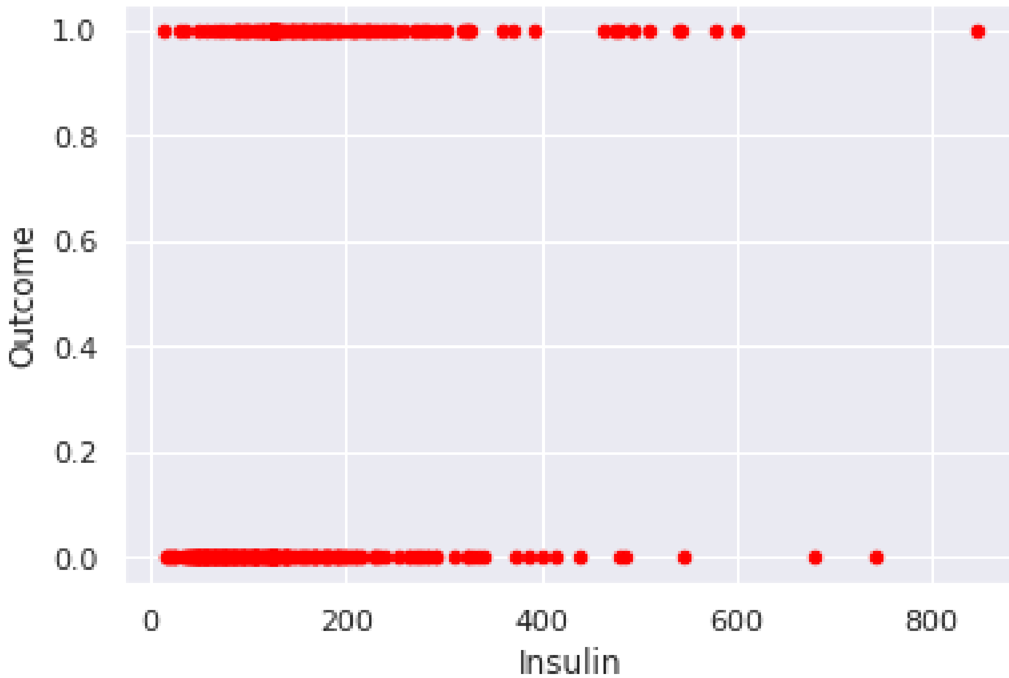
<Figure size 432x288 with 0 Axes>



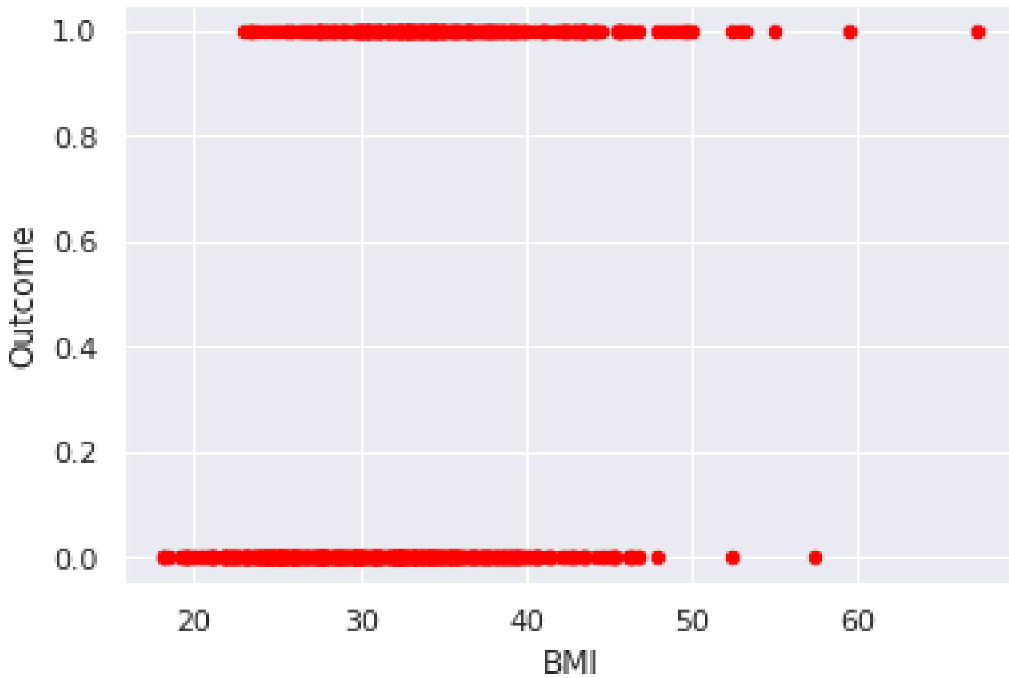
<Figure size 432x288 with 0 Axes>



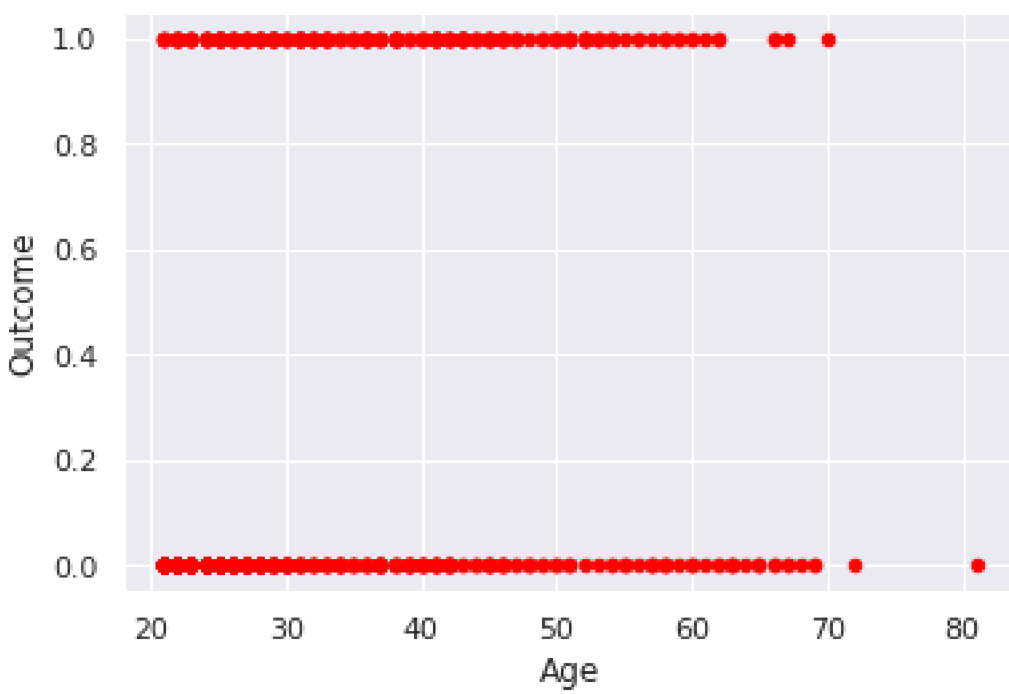
<Figure size 432x288 with 0 Axes>



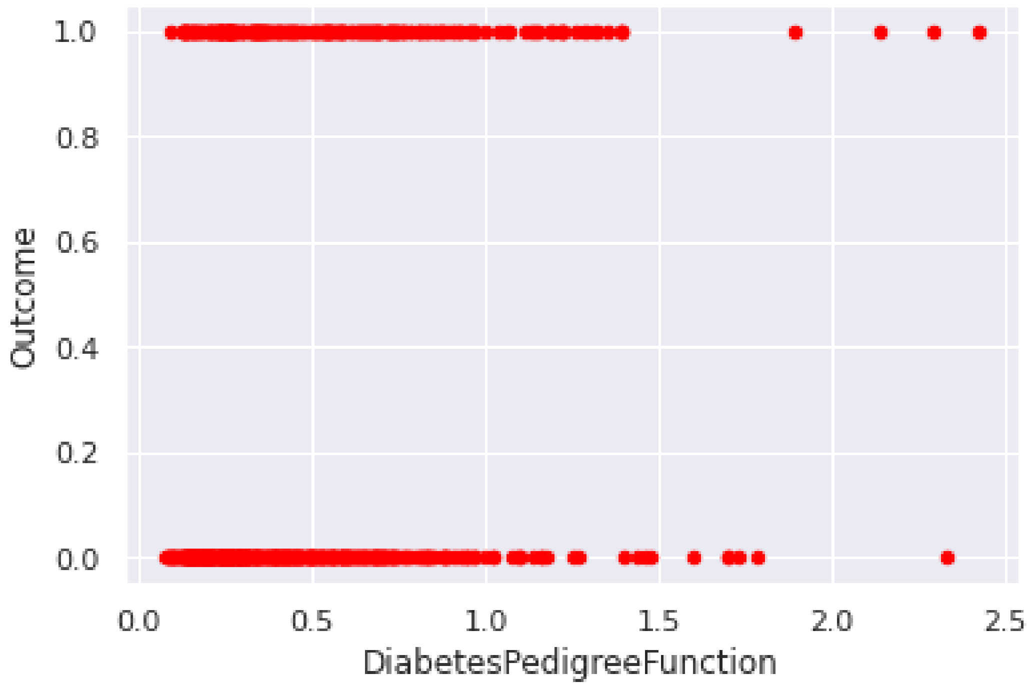
<Figure size 432x288 with 0 Axes>



<Figure size 432x288 with 0 Axes>



<Figure size 432x288 with 0 Axes>



Number of Pregnancies doesnt have major influence on the outcome People with higher Glucose levels may be diabetic BloodPressure is again not a significant influencing factor Higher Insulin levels are prone to be diabetic Higher BMI are prone to be diabetic Age and DiabetesPedigreeFunction are not influencing factors

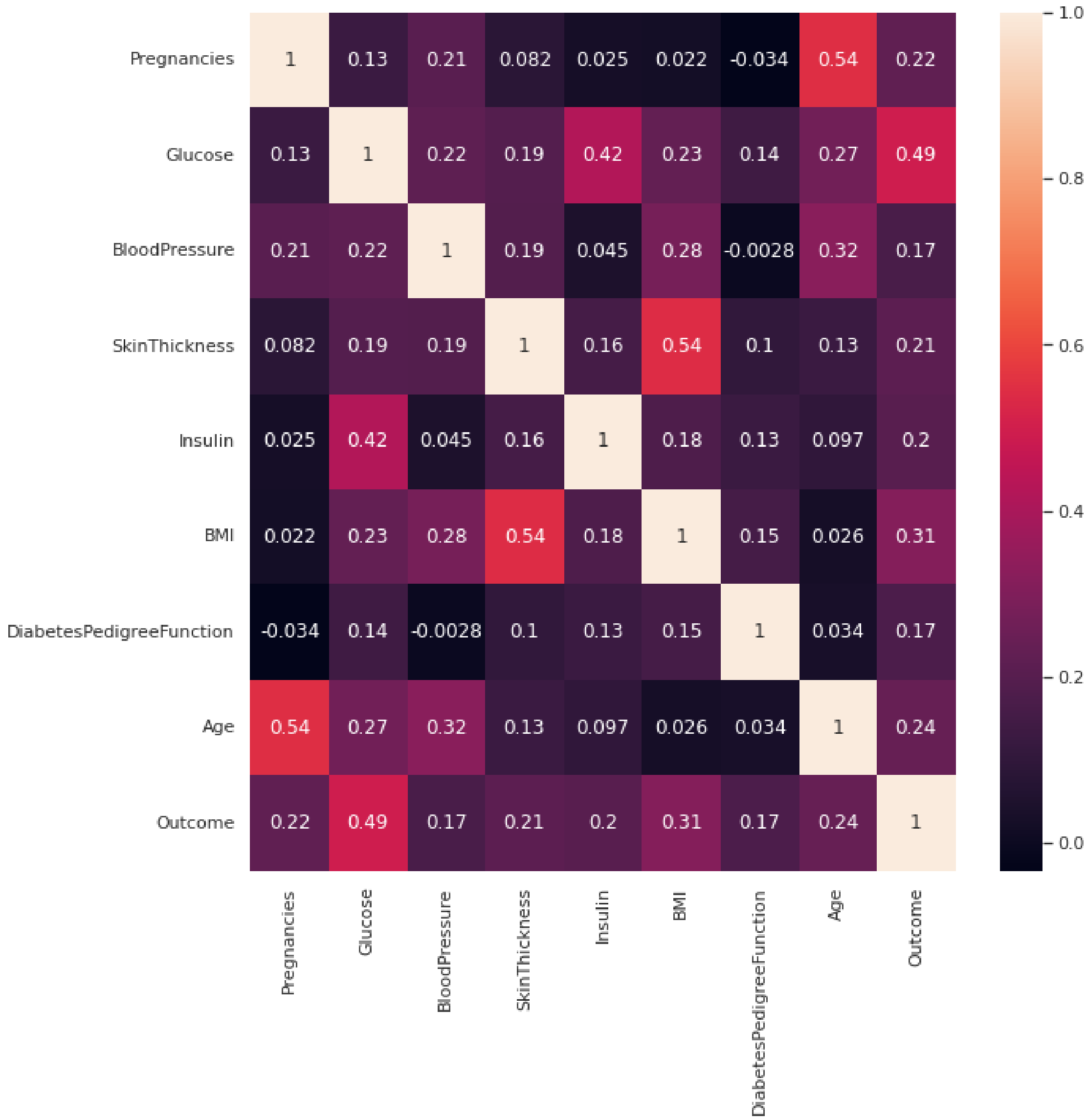
0.6 6. Perform correlation analysis. Visually explore it using a heat map

[27]:
```
corr = df1.corr()
print(corr)
plt.figure(figsize=(10,10))
sns.heatmap(corr,
         xticklabels=corr.columns,
         yticklabels=corr.columns,annot=True)
```

```
                          Pregnancies   Glucose  BloodPressure  SkinThickness  \
Pregnancies                  1.000000  0.127911       0.208522       0.081770
Glucose                      0.127911  1.000000       0.218367       0.192686
BloodPressure                0.208522  0.218367       1.000000       0.191853
SkinThickness                0.081770  0.192686       0.191853       1.000000
Insulin                      0.025047  0.419064       0.045087       0.155610
BMI                          0.021559  0.231128       0.281199       0.543205
```

```
DiabetesPedigreeFunction    -0.033523  0.137060      -0.002763       0.102188
Age                          0.544341  0.266534       0.324595       0.126107
Outcome                      0.221898  0.492928       0.166074       0.214873

                           Insulin       BMI  DiabetesPedigreeFunction  \
Pregnancies               0.025047  0.021559                 -0.033523
Glucose                   0.419064  0.231128                  0.137060
BloodPressure             0.045087  0.281199                 -0.002763
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                   0.266534  0.492928
BloodPressure             0.324595  0.166074
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
```

[27]: <AxesSubplot:>



0.7 Project Task: Week 2

Data Modeling:

1. Devise strategies for model building. It is important to decide the right validation framework. Express your thought process.
2. Apply an appropriate classification algorithm to build a model.
3. Compare various models with the results from KNN algorithm.
4. Create a classification report by analyzing sensitivity, specificity, AUC (ROC curve), etc.

[28]:
```
df1.head()
```

[28]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	\
0	6	148.0	72.0	35.0	125.0	33.6	
1	1	85.0	66.0	29.0	125.0	26.6	
2	8	183.0	64.0	29.0	125.0	23.3	
3	1	89.0	66.0	23.0	94.0	28.1	
4	0	137.0	40.0	35.0	168.0	43.1	

	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

[29]:
```
df1.describe()
```

[29]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	\
count	768.000000	768.000000	768.000000	768.000000	768.000000	
mean	3.845052	121.686763	72.405184	29.108073	140.671875	
std	3.369578	30.435949	12.096346	8.791221	86.383060	
min	0.000000	44.000000	24.000000	7.000000	14.000000	
25%	1.000000	99.750000	64.000000	25.000000	121.500000	
50%	3.000000	117.000000	72.202592	29.000000	125.000000	
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	32.455208	0.471876	33.240885	0.348958
std	6.875177	0.331329	11.760232	0.476951
min	18.200000	0.078000	21.000000	0.000000
25%	27.500000	0.243750	24.000000	0.000000
50%	32.300000	0.372500	29.000000	0.000000
75%	36.600000	0.626250	41.000000	1.000000
max	67.100000	2.420000	81.000000	1.000000

[30]:
```
X_feature = df1.drop('Outcome',axis=1)
```

[31]:
```
Y_target = df[['Outcome']]
```

[32]:
```
print(X_feature.shape)
print(Y_target.shape)
```

```
(768, 8)
(768, 1)
```

[33]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test =␣
↪train_test_split(X_feature,Y_target,random_state=1)
```

0.8 Logistic Regression

[34]:
```
from sklearn.linear_model import LogisticRegression
logReg = LogisticRegression(solver='lbfgs', max_iter=1000)
logReg.fit(X_train,Y_train.values.ravel())
```

[34]:
```
LogisticRegression(max_iter=1000)
```

[35]:
```
y_pred = logReg.predict(X_test)
```

[36]:
```
y_pred
```

[36]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0,
       1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0,
       0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0,
       1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0,
       0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0,
       0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0,
       0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0,
       1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0,
       1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0])
```

[37]:
```
from sklearn import metrics
print(metrics.accuracy_score(Y_test,y_pred))
logreg_acc = metrics.accuracy_score(Y_test,y_pred)
print("Accuracy = ", logreg_acc )
```

```
0.78125
Accuracy =  0.78125
```

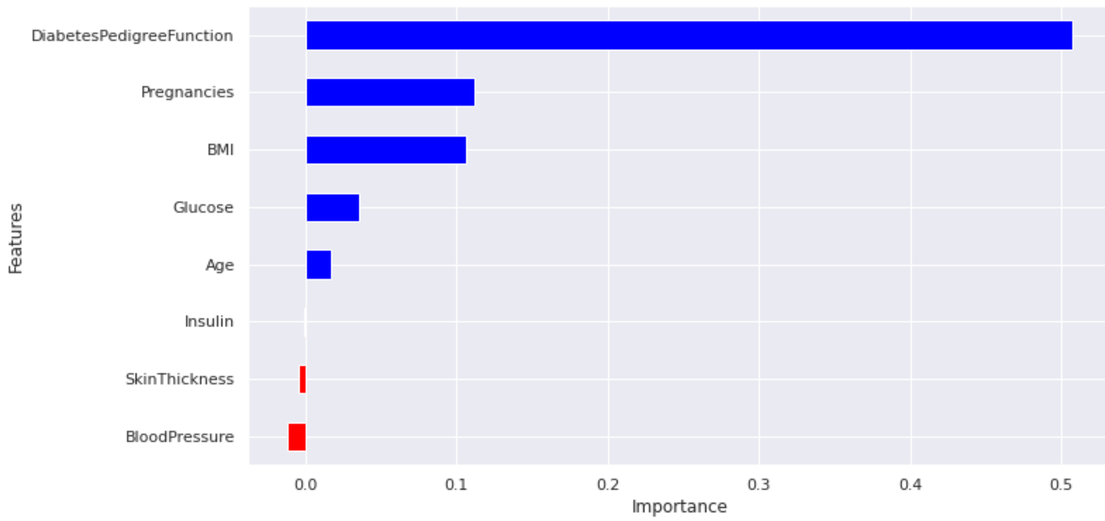
[38]:
```
print('Actual    :', Y_test.values[0:30].ravel())
print('Predicted:', y_pred[0:30])
```

```
Actual    : [0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 1 0 0 0 1 1 1 1 0 0 0 1 0 1]
Predicted: [0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 0 0 0 0 0 0 1 0 0 0 0 1 0 0]
```

[39]:
```
coeff = list(logReg.coef_[0])
labels = list(X_train.columns)
features = pd.DataFrame()
features['Features'] = labels
features['importance'] = coeff
features.sort_values(by=['importance'], ascending=True, inplace=True)
```

```
features['positive'] = features['importance'] > 0
features.set_index('Features', inplace=True)
features.importance.plot(kind='barh', figsize=(11, 6),color = features.positive.
 ↪map({True: 'blue', False: 'red'}))
plt.xlabel('Importance')
```

[39]: Text(0.5, 0, 'Importance')



a)The following features have significant influence on the Model : DiabetesPedigreeFunction, Pregnancies, BMI, Glucose b)The following have negative influence on the Model : Insulin, SkinThickness and BloodPressure

[40]:
```
import joblib
joblib.dump([logReg], 'diabeteseModel.pkl') # saving model for future use
```

[40]: ['diabeteseModel.pkl']

0.9 Random Forest

[41]:
```
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier(n_estimators=200)
rfc.fit(X_train, Y_train.values.ravel())
```

[41]: RandomForestClassifier(n_estimators=200)

[42]:
```
rfc_train = rfc.predict(X_train)
print("Accuracy_Score =", format(metrics.accuracy_score(Y_train, rfc_train)))
```

```
Accuracy_Score = 1.0
```

Training Dataset is overfitted using Random Forest

[43]:
```
predictions = rfc.predict(X_test)
rfc_acc = metrics.accuracy_score(Y_test, predictions)
print("Accuracy_Score =", format(rfc_acc))
```

```
Accuracy_Score = 0.7916666666666666
```

Classification report and confusion matrix of random forest model

[44]:
```
from sklearn.metrics import classification_report, confusion_matrix

print(confusion_matrix(Y_test, predictions))
print(classification_report(Y_test,predictions))
```

```
[[107  16]
 [ 24  45]]
              precision    recall  f1-score   support

           0       0.82      0.87      0.84       123
           1       0.74      0.65      0.69        69

    accuracy                           0.79       192
   macro avg       0.78      0.76      0.77       192
weighted avg       0.79      0.79      0.79       192
```

0.10 Decision Tree

[45]:
```
from sklearn.tree import DecisionTreeClassifier
dtree = DecisionTreeClassifier()
dtree.fit(X_train, Y_train.values.ravel())
```

[45]:
```
DecisionTreeClassifier()
```

[46]:
```
predictions = dtree.predict(X_test)
dt_acc = metrics.accuracy_score(Y_test,predictions)
print("Accuracy Score =", format(dt_acc))
```

```
Accuracy Score = 0.6614583333333334
```

Classification report and confusion matrix of the decision tree model

[47]:
```
print(confusion_matrix(Y_test, predictions))
print(classification_report(Y_test,predictions))
```

```
[[91 32]
 [33 36]]
              precision    recall  f1-score   support
```

```
           0       0.73      0.74      0.74       123
           1       0.53      0.52      0.53        69

    accuracy                           0.66       192
   macro avg       0.63      0.63      0.63       192
weighted avg       0.66      0.66      0.66       192
```

0.11 XgBoost Classifier

Installed using conda install -c anaconda py-xgboost

[48]:
```python
from xgboost import XGBClassifier

xgb_model = XGBClassifier(gamma=0,use_label_encoder=False)
xgb_model.fit(X_train, Y_train.values.ravel())
```

[48]:
```
XGBClassifier(base_score=0.5, booster=None, colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
              importance_type='gain', interaction_constraints=None,
              learning_rate=0.300000012, max_delta_step=0, max_depth=6,
              min_child_weight=1, missing=nan, monotone_constraints=None,
              n_estimators=100, n_jobs=0, num_parallel_tree=1, random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
              tree_method=None, use_label_encoder=False,
              validate_parameters=False, verbosity=None)
```

[49]:
```python
xgb_pred = xgb_model.predict(X_test)
xgb_acc = metrics.accuracy_score(Y_test, xgb_pred)
print("Accuracy Score =", format(xgb_acc))
```

```
Accuracy Score = 0.765625
```

[50]:
```python
print(confusion_matrix(Y_test, xgb_pred))
print(classification_report(Y_test,xgb_pred))
```

```
[[100  23]
 [ 22  47]]
              precision    recall  f1-score   support

           0       0.82      0.81      0.82       123
           1       0.67      0.68      0.68        69

    accuracy                           0.77       192
   macro avg       0.75      0.75      0.75       192
```

```
weighted avg       0.77      0.77      0.77       192
```

0.12 Support Vector Machine (SVM)

[51]:
```python
from sklearn.svm import SVC

svc_model = SVC()
svc_model.fit(X_train, Y_train.values.ravel())
```

[51]:
```
SVC()
```

[52]:
```python
svc_pred = svc_model.predict(X_test)
```

[53]:
```python
svm_acc = metrics.accuracy_score(Y_test, svc_pred)
print("Accuracy Score =", format(svm_acc))
```

```
Accuracy Score = 0.7708333333333334
```

[54]:
```python
print(confusion_matrix(Y_test, svc_pred))
print(classification_report(Y_test,svc_pred))
```

```
[[113  10]
 [ 34  35]]
              precision    recall  f1-score   support

           0       0.77      0.92      0.84       123
           1       0.78      0.51      0.61        69

    accuracy                           0.77       192
   macro avg       0.77      0.71      0.73       192
weighted avg       0.77      0.77      0.76       192
```

0.13 Knn Algortithm

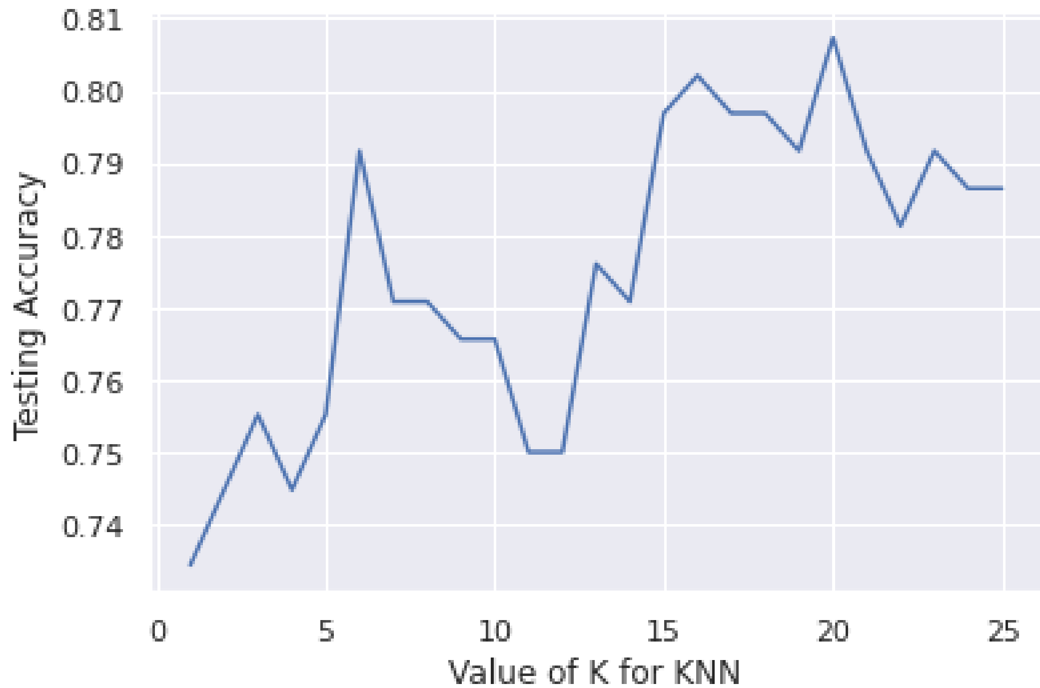
[55]:
```python
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, Y_train.values.ravel())
y_pred = knn.predict(X_test)
knn_acc = metrics.accuracy_score(Y_test, y_pred)
print("Accuracy: ",knn_acc)
```

```
Accuracy:  0.7552083333333334
```

[56]:
```python
# Checking K=1 through K=25 and testing accuracy
k_range = range(1, 26)


scores = []


# We append the scores in the dictionary
for k in k_range:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, Y_train.values.ravel())
    y_pred = knn.predict(X_test)
    scores.append(metrics.accuracy_score(Y_test, y_pred))

print(scores)

plt.plot(k_range, scores)
plt.xlabel('Value of K for KNN')
plt.ylabel('Testing Accuracy')
```

```
[0.734375, 0.7447916666666666, 0.7552083333333334, 0.7447916666666666,
0.7552083333333334, 0.7916666666666666, 0.7708333333333334, 0.7708333333333334,
0.765625, 0.765625, 0.75, 0.75, 0.7760416666666666, 0.7708333333333334,
0.796875, 0.8020833333333334, 0.796875, 0.796875, 0.7916666666666666,
0.8072916666666666, 0.7916666666666666, 0.78125, 0.7916666666666666,
0.7864583333333334, 0.7864583333333334]
```

[56]:
```
Text(0, 0.5, 'Testing Accuracy')
```



[57]:
```
scores_knn = np.array(scores)
print("For K value of {} the accuracy is maximum at {}".format(scores_knn.
    ↪argmax(),scores_knn.max()))
knn_acc = scores_knn.max()
```

```
For K value of 19 the accuracy is maximum at 0.8072916666666666
```

0.14 Conclusion

[58]:
```
alg=np.array(['Support Vector Machine','XgBoost Classifier','Decision␣
    ↪Tree','Random Forest','Logistic Regression','KNN Algorithm'])
acc_score=np.array([svm_acc,xgb_acc,dt_acc,rfc_acc,logreg_acc,knn_acc])
```

[59]:
```
df_accuracy=pd.DataFrame({'Algorithm':alg,'Accuracy':acc_score})
df_accuracy
```

[59]:

	Algorithm	Accuracy
0	Support Vector Machine	0.770833
1	XgBoost Classifier	0.765625
2	Decision Tree	0.661458
3	Random Forest	0.791667
4	Logistic Regression	0.781250
5	KNN Algorithm	0.807292

[60]:
```
max_acc = df_accuracy['Accuracy'].max()
print("{} provides maximum accuracy {}".format(df_accuracy.
 ↪loc[df_accuracy['Accuracy'] == max_acc,'Algorithm'].iloc[0],max_acc))
```

```
KNN Algorithm provides maximum accuracy 0.8072916666666666
```

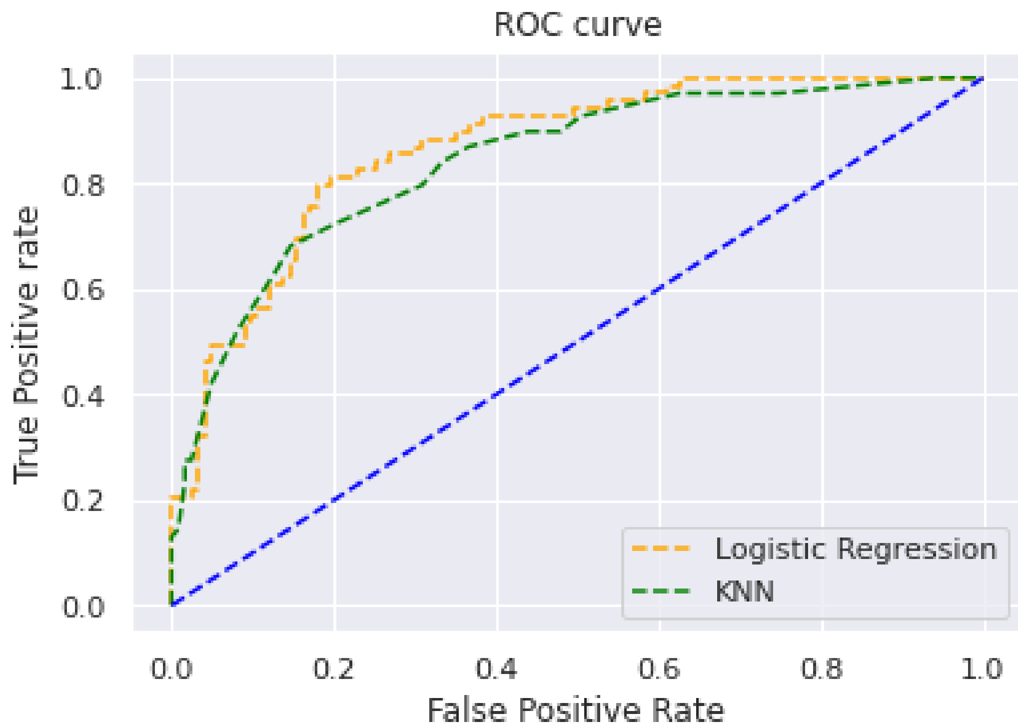
[61]:
```
# predict probabilities
pred_prob1 = logReg.predict_proba(X_test)
pred_prob2 = knn.predict_proba(X_test)
```

[62]:
```
from sklearn.metrics import roc_curve

# roc curve for models
fpr1, tpr1, thresh1 = roc_curve(Y_test, pred_prob1[:,1], pos_label=1)
fpr2, tpr2, thresh2 = roc_curve(Y_test, pred_prob2[:,1], pos_label=1)

# roc curve for tpr = fpr
random_probs = [0 for i in range(len(Y_test))]
p_fpr, p_tpr, _ = roc_curve(Y_test, random_probs, pos_label=1)
```

[63]:
```
from sklearn.metrics import roc_auc_score

# auc scores
auc_score1 = roc_auc_score(Y_test, pred_prob1[:,1])
auc_score2 = roc_auc_score(Y_test, pred_prob2[:,1])

print(auc_score1, auc_score2)
```

```
0.8668551902910333 0.8427595145516673
```

[64]:
```
plt.plot(fpr1, tpr1, linestyle='--',color='orange', label='Logistic Regression')
plt.plot(fpr2, tpr2, linestyle='--',color='green', label='KNN')
plt.plot(p_fpr, p_tpr, linestyle='--', color='blue')
# title
plt.title('ROC curve')
# x label
plt.xlabel('False Positive Rate')
# y label
plt.ylabel('True Positive rate')

plt.legend(loc='best')
plt.savefig('ROC',dpi=300)
plt.show();
```



It is evident from the plot that the AUC for the Logistic Regression ROC curve is higher than that for the KNN ROC curve. Therefore, we can say that logistic regression did a better job of classifying the positive class in the dataset.

```
[ ]:
```