Capstone 2project Health Care (1)

May 16, 2023

[ ]:

```

```

[1]:

```
#import the libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import warnings
import seaborn as sns
warnings.filterwarnings(action = "ignore", category = FutureWarning)
```

[2]:

```
data= pd.read_csv("health care diabetes.csv")
data
```

[2]:

```
     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
..           …        …              …            …       …   …
763           10      101             76             48      180  32.9
764            2      122             70             27        0  36.8
765            5      121             72             23      112  26.2
766            1      126             60              0        0  30.1
767            1       93             70             31        0  30.4

     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
..                        …   …       …
763                     0.171   63        0
764                     0.340   27        0
765                     0.245   30        0
```

```
766                       0.349   47         1
767                       0.315   23         0

[768 rows x 9 columns]
```

0.0.1 1. Perform descriptive analysis. It is very important to understand the variables and corresponding values. We need to think through - Can minimum value of below listed columns be zero (0)? On these columns, a value of zero does not make sense and thus indicates missing value.Glucose, BloodPressure, SkinThickness, Insuline, BMI.How will you treat these values?

[3]:
```python
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
```

[4]:
```python
data.head(15)
```

[4]:
```
    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
```

10	4	110	92	0	0	37.6
11	10	168	74	0	0	38.0
12	10	139	80	0	0	27.1
13	1	189	60	23	846	30.1
14	5	166	72	19	175	25.8

	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
10	0.191	30	0
11	0.537	34	1
12	1.441	57	0
13	0.398	59	1
14	0.587	51	1

0.0.2 we can see there are 0 in the columns of bloodpressure, bmi, insulin etc.. so this value doesnot make any sense. so replacing all the 0 value with median.

[5]:
```
data['Glucose']=data['Glucose'].replace(0,data['Glucose'].median())
data['BloodPressure']=data['BloodPressure'].replace(0,data['BloodPressure'].
 ↪median())
data['SkinThickness']=data['SkinThickness'].replace(0,data['SkinThickness'].
 ↪median())
data['Insulin']=data['Insulin'].replace(0,data['Insulin'].median())
data['BMI']=data['BMI'].replace(0,data['BMI'].median())
data.describe()
```

[5]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin \
count	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	121.656250	72.386719	27.334635	94.652344
std	3.369578	30.438286	12.096642	9.229014	105.547598
min	0.000000	44.000000	24.000000	7.000000	14.000000
25%	1.000000	99.750000	64.000000	23.000000	30.500000
50%	3.000000	117.000000	72.000000	23.000000	31.250000
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    32.450911                  0.471876   33.240885    0.348958
std      6.875366                  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%     32.000000                  0.372500   29.000000    0.000000
75%     36.600000                  0.626250   41.000000    1.000000
max     67.100000                  2.420000   81.000000    1.000000
```

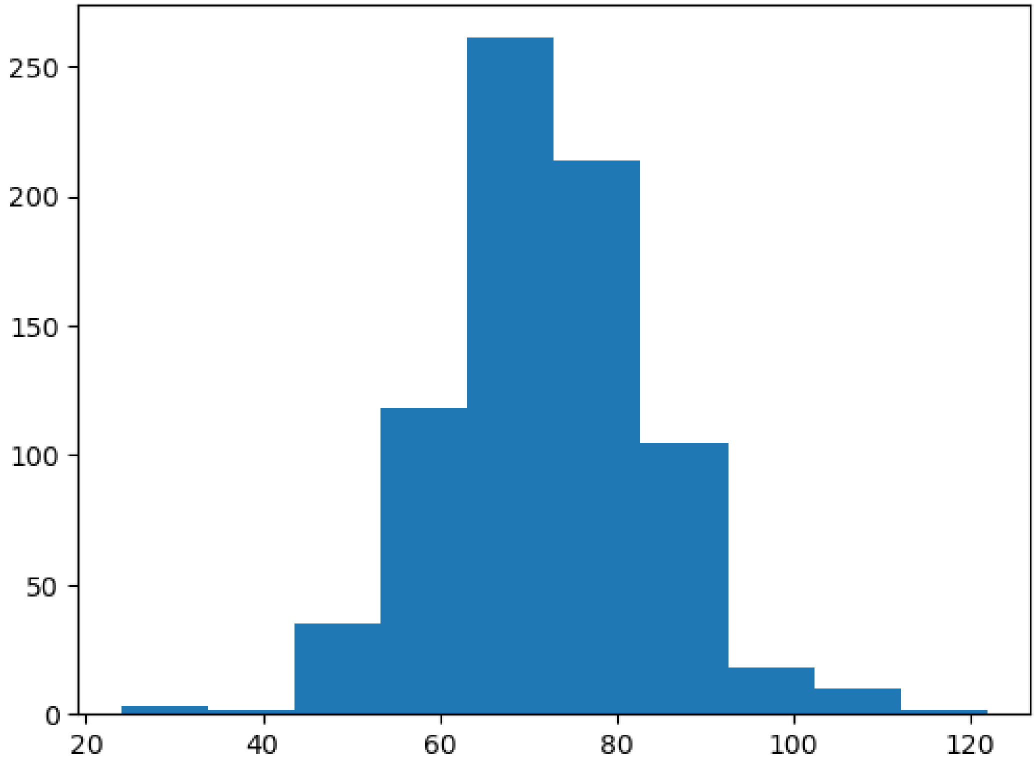
0.0.3 2 Visually explore these variables using histograms. Treat the missing values accordingly.

[6]:
```python
data.isna().any()
```

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

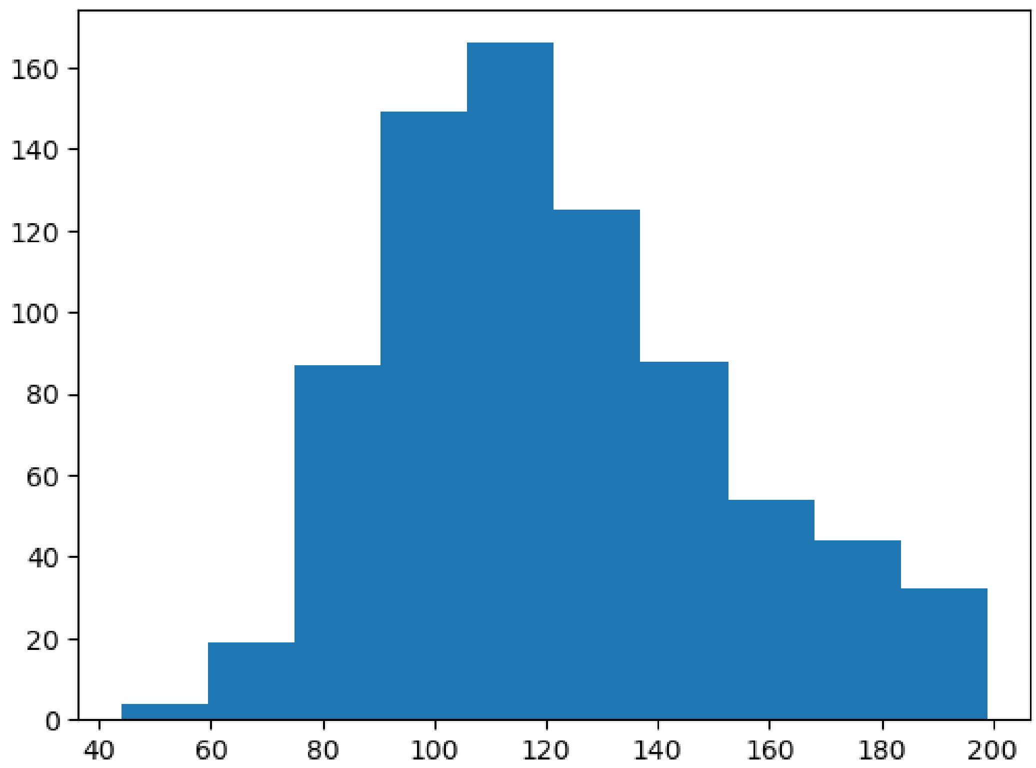
[7]:
```python
plt.hist("BloodPressure", data= data)
```

[7]:
```
(array([  3.,   2.,  35., 118., 261., 214., 105.,  18.,  10.,   2.]),
 array([ 24. ,  33.8,  43.6,  53.4,  63.2,  73. ,  82.8,  92.6, 102.4,
        112.2, 122. ]),
 <BarContainer object of 10 artists>)
```



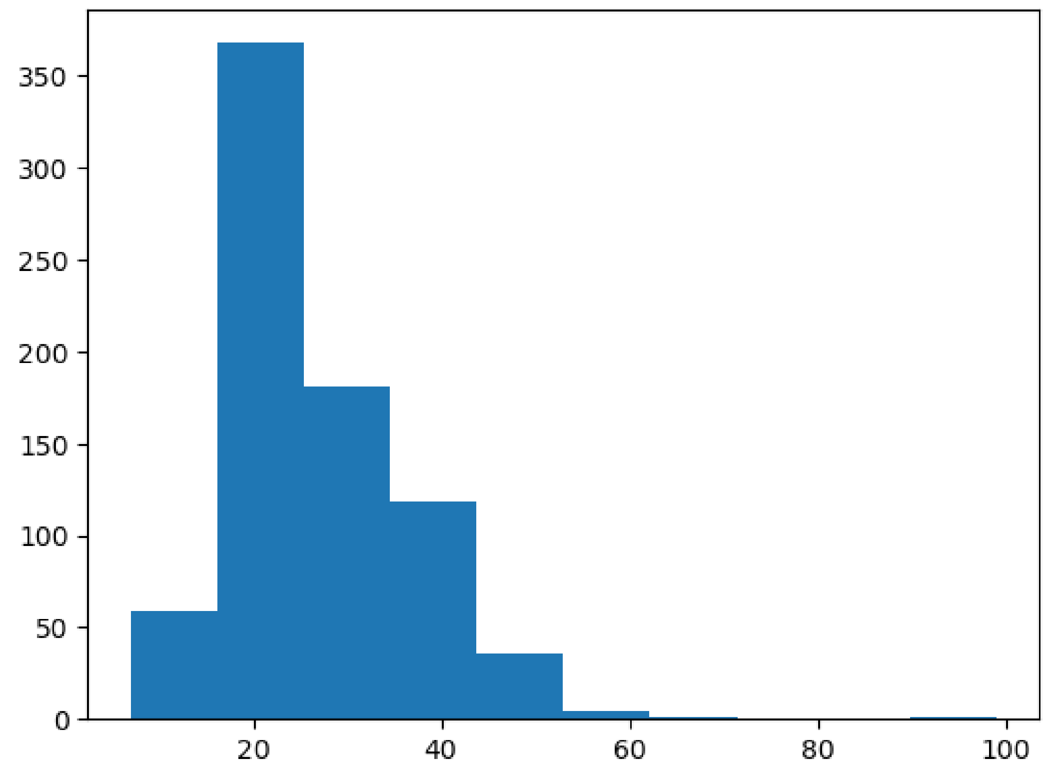
[8]:
```
plt.hist("Glucose", data= data)
```

[8]:
```
(array([  4.,  19.,  87., 149., 166., 125.,  88.,  54.,  44.,  32.]),
 array([ 44. ,  59.5,  75. ,  90.5, 106. , 121.5, 137. , 152.5, 168. ,
        183.5, 199. ]),
 <BarContainer object of 10 artists>)
```



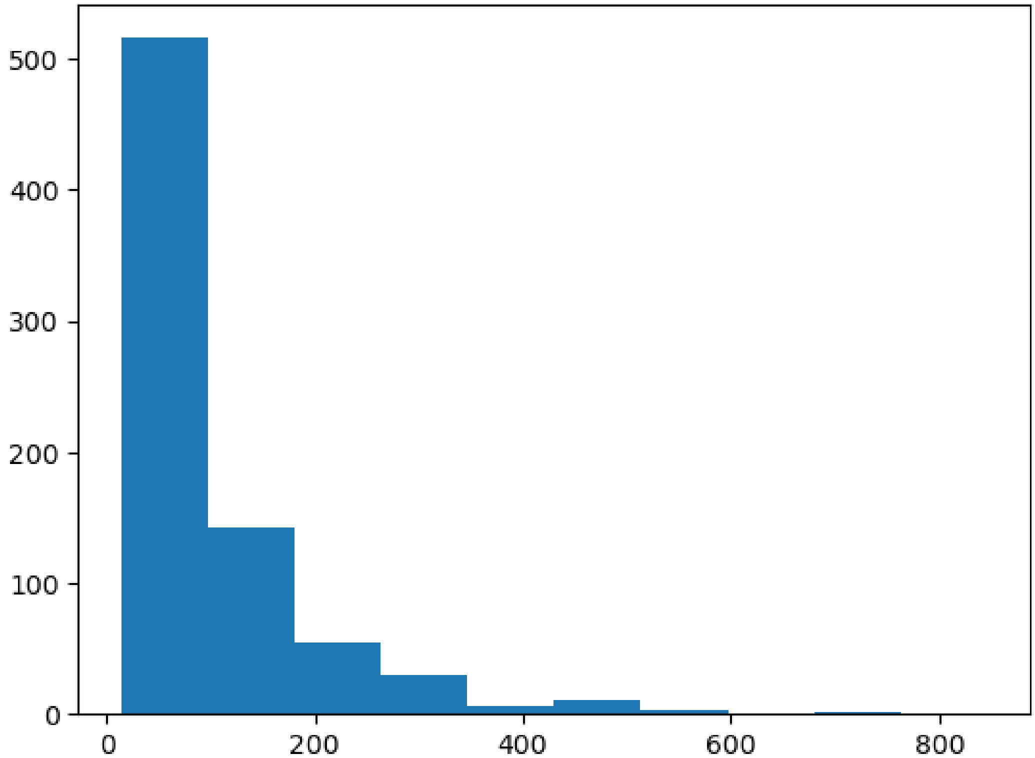
[9]:
```
plt.hist("SkinThickness", data= data)
```

[9]:
```
(array([ 59., 368., 181., 118.,  36.,   4.,   1.,   0.,   0.,   1.]),
 array([ 7. , 16.2, 25.4, 34.6, 43.8, 53. , 62.2, 71.4, 80.6, 89.8, 99. ]),
 <BarContainer object of 10 artists>)
```



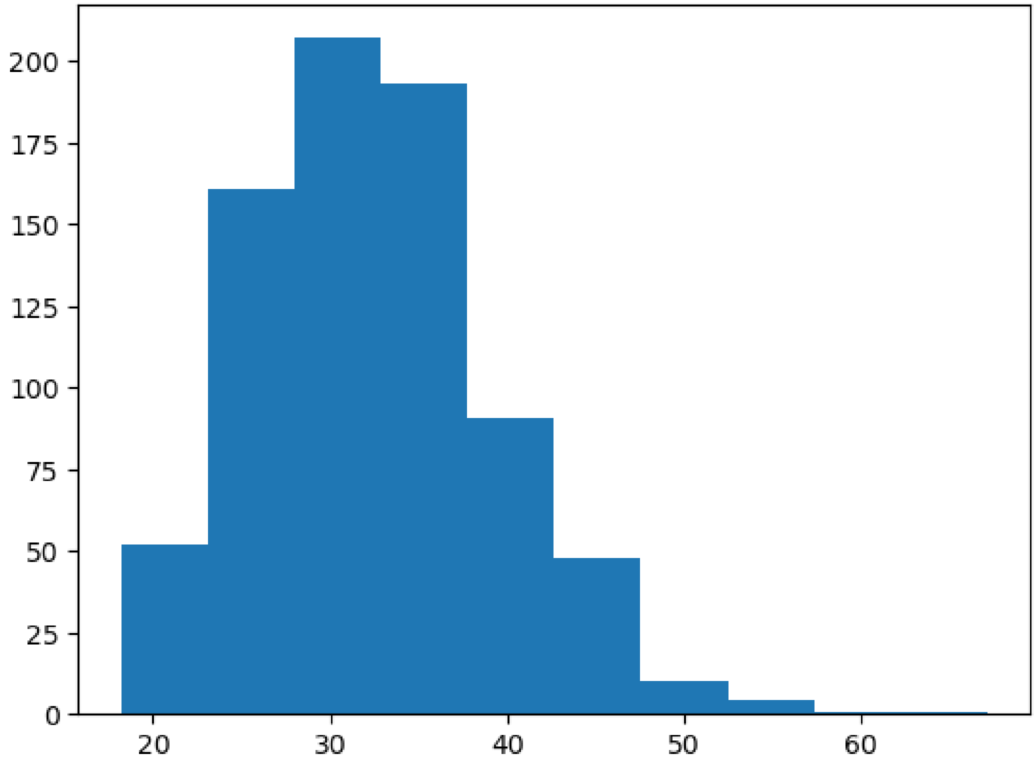
[10]:
```
plt.hist("Insulin", data= data)
```

[10]:
```
(array([516., 143.,  55.,  29.,   7.,  10.,   4.,   1.,   2.,   1.]),
 array([ 14. ,  97.2, 180.4, 263.6, 346.8, 430. , 513.2, 596.4, 679.6,
        762.8, 846. ]),
 <BarContainer object of 10 artists>)
```



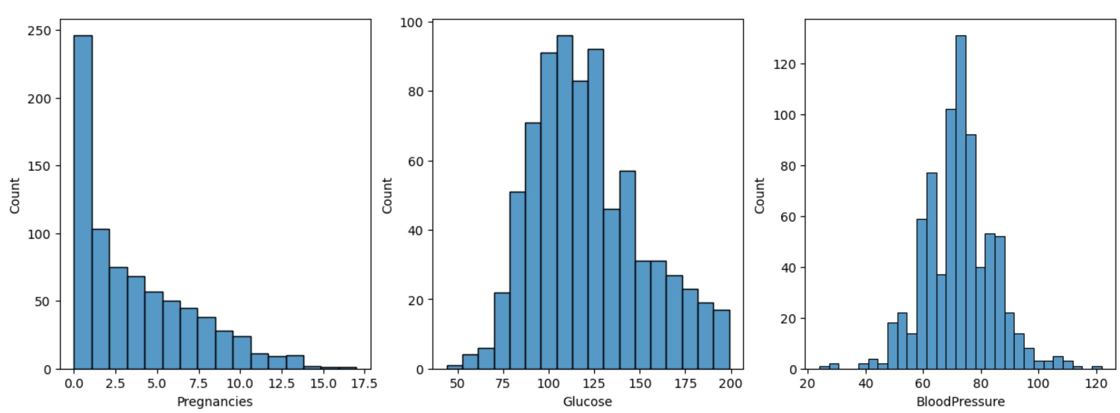
[11]:
```
plt.hist("BMI", data= data)
```

[11]:
```
(array([ 52., 161., 207., 193.,  91.,  48.,  10.,   4.,   1.,   1.]),
 array([18.2 , 23.09, 27.98, 32.87, 37.76, 42.65, 47.54, 52.43, 57.32,
        62.21, 67.1 ]),
 <BarContainer object of 10 artists>)
```



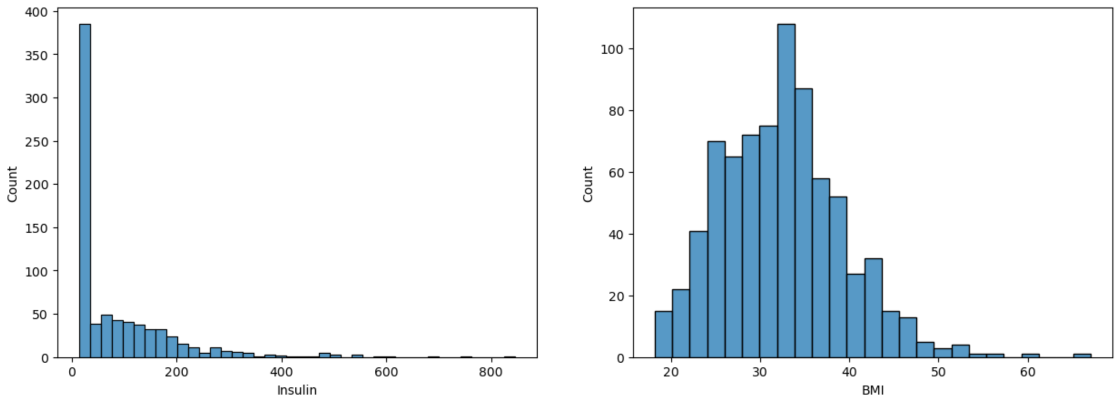
[12]:
```
fig, ax= plt.subplots(ncols= 3, figsize=(15,5))

sns.histplot(x= "Pregnancies", data= data, ax=ax[0])
sns.histplot(x= "Glucose", data= data, ax= ax[1])
sns.histplot(x= "BloodPressure", data= data, ax= ax[2])
```

[12]: <AxesSubplot:xlabel='BloodPressure', ylabel='Count'>



[13]:
```
fig, ax= plt.subplots(ncols=2 , figsize=(15,5))
sns.histplot(x= "Insulin", data= data, ax= ax[0])
sns.histplot(x= "BMI", data= data, ax= ax[1])
```

[13]: <AxesSubplot:xlabel='BMI', ylabel='Count'>

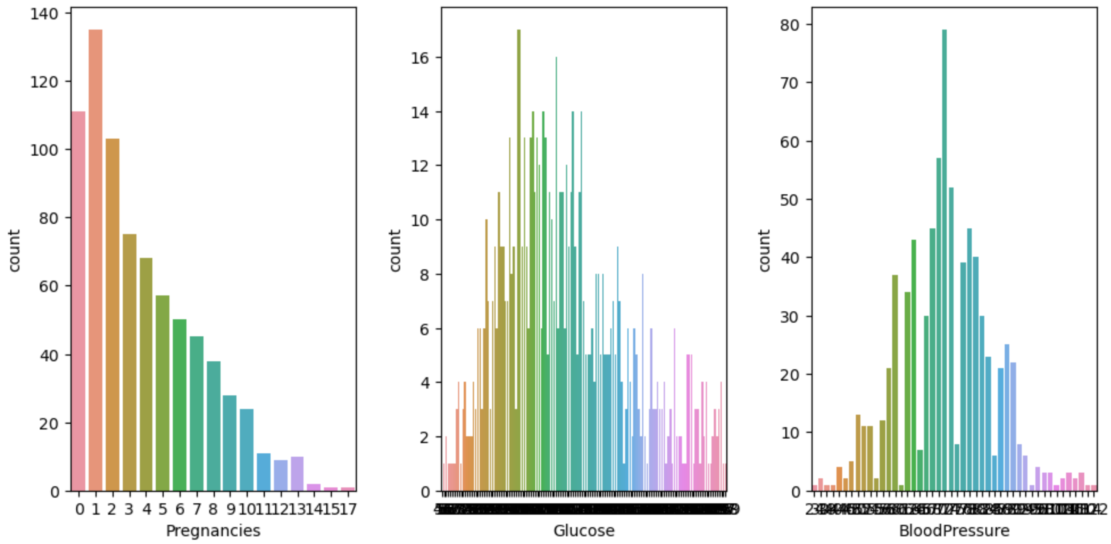


0.0.4 3. There are integer and float data type variables in this dataset. Create a count (frequency) plot describing the data types and the count of variables.

[14]:
```
data.dtypes
```

[14]:
```
Pregnancies                   int64
Glucose                       int64
BloodPressure                 int64
SkinThickness                 int64
Insulin                     float64
BMI                         float64
DiabetesPedigreeFunction    float64
Age                           int64
Outcome                       int64
dtype: object
```

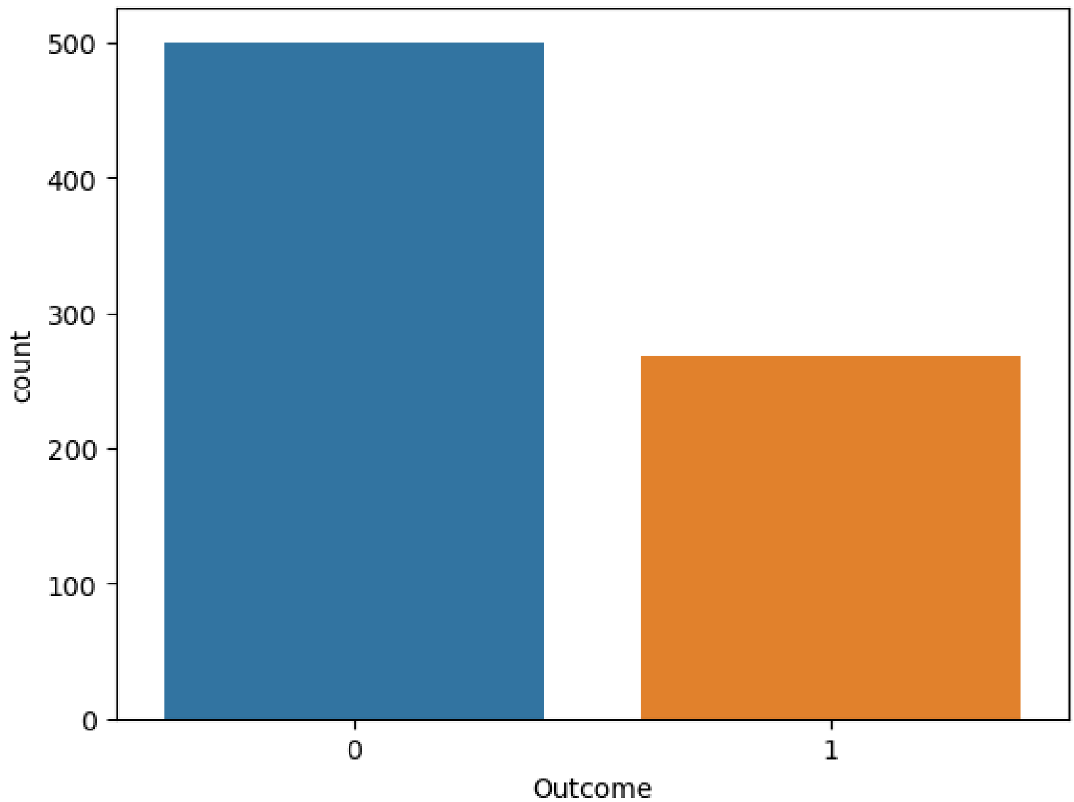
[15]:
```
fig, ax= plt.subplots(ncols=3, figsize= (10,5))
sns.countplot(x= "Pregnancies", data= data, ax= ax[0])
sns.countplot(x= "Glucose", data= data, ax= ax[1])
sns.countplot(x= "BloodPressure", data= data, ax= ax[2])
plt.tight_layout()
```



0.0.5 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.

[16]:
```
sns.countplot(x= "Outcome", data= data)
```

[16]:
```
<AxesSubplot:xlabel='Outcome', ylabel='count'>
```



[17]:
```
print("value of \n", data["Outcome"].value_counts())
```

```
value of
 0    500
1    268
Name: Outcome, dtype: int64
```

[18]:
```
data["Outcome"].value_counts()/len(data)
```

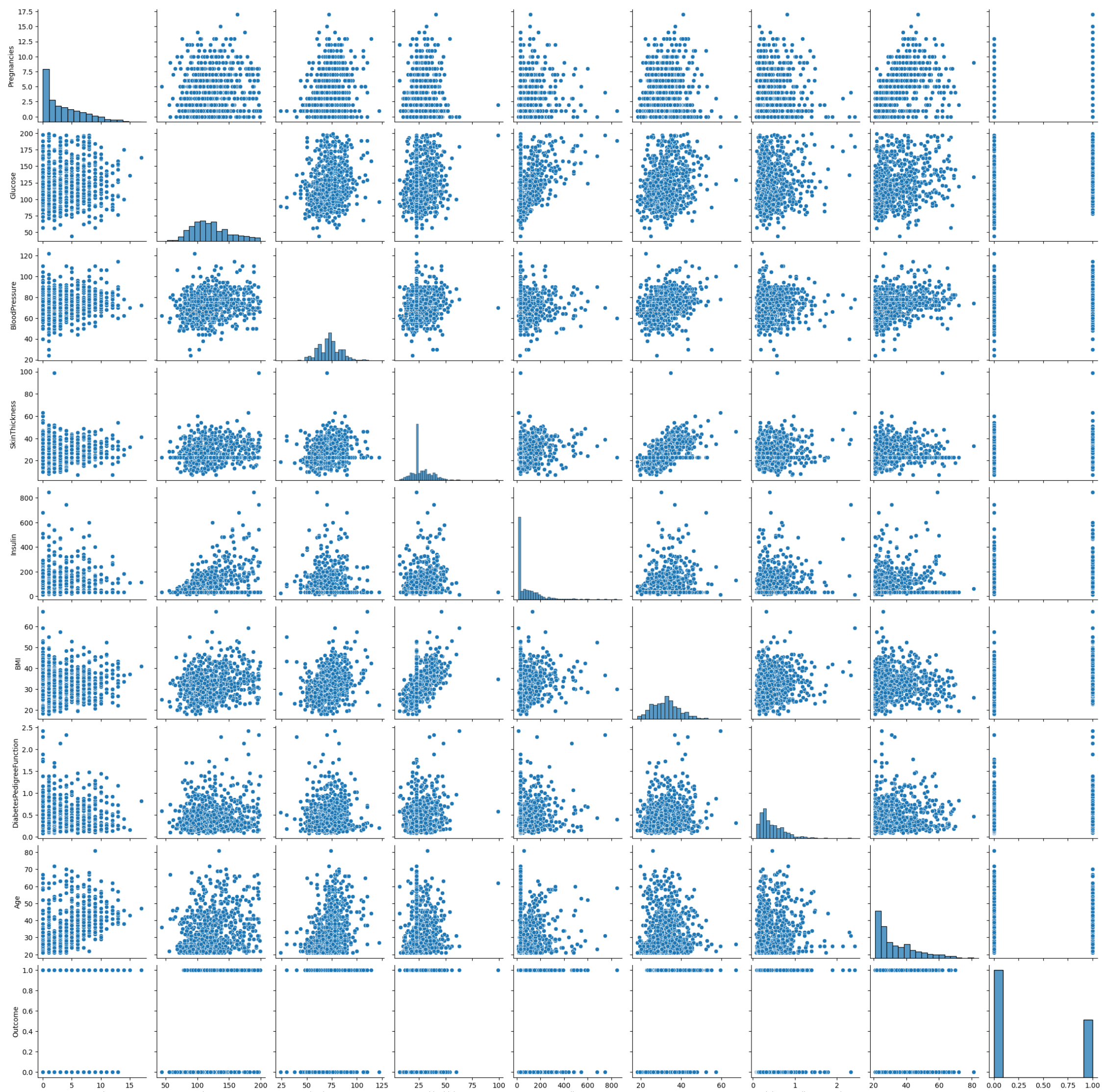
[18]:
```
0    0.651042
1    0.348958
Name: Outcome, dtype: float64
```

0.0.6 The data set is balanced.

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

[19]:
```
sns.pairplot(data)
```

[19]: <seaborn.axisgrid.PairGrid at 0x2ae616e1550>



0.0.8 with this visualisation. we can see positive correlation between BMI and Skin Thickness ; and also between age and pregnancy,

[20]:
```
data.corr()
```

[20]:

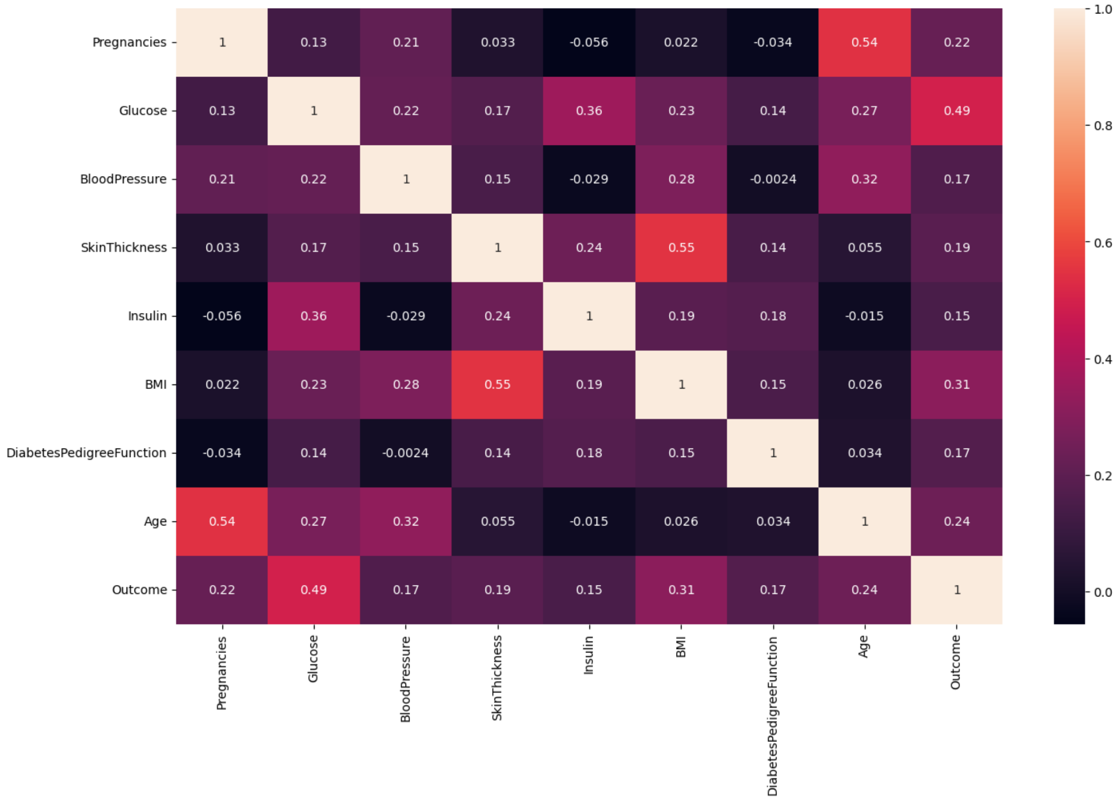
	Pregnancies	Glucose	BloodPressure	SkinThickness
Pregnancies	1.000000	0.128213	0.208615	0.032568
Glucose	0.128213	1.000000	0.218937	0.172143
BloodPressure	0.208615	0.218937	1.000000	0.147809
SkinThickness	0.032568	0.172143	0.147809	1.000000
Insulin	-0.055697	0.357573	-0.028721	0.238188
BMI	0.021546	0.231400	0.281132	0.546951
DiabetesPedigreeFunction	-0.033523	0.137327	-0.002378	0.142977
Age	0.544341	0.266909	0.324915	0.054514
Outcome	0.221898	0.492782	0.165723	0.189065

	Insulin	BMI	DiabetesPedigreeFunction
Pregnancies	-0.055697	0.021546	-0.033523
Glucose	0.357573	0.231400	0.137327
BloodPressure	-0.028721	0.281132	-0.002378
SkinThickness	0.238188	0.546951	0.142977
Insulin	1.000000	0.189022	0.178029
BMI	0.189022	1.000000	0.153506
DiabetesPedigreeFunction	0.178029	0.153506	1.000000
Age	-0.015413	0.025744	0.033561
Outcome	0.148457	0.312249	0.173844

	Age	Outcome
Pregnancies	0.544341	0.221898
Glucose	0.266909	0.492782
BloodPressure	0.324915	0.165723
SkinThickness	0.054514	0.189065
Insulin	-0.015413	0.148457
BMI	0.025744	0.312249
DiabetesPedigreeFunction	0.033561	0.173844
Age	1.000000	0.238356
Outcome	0.238356	1.000000

[21]:
```
plt.subplots(figsize=(15,9))
sns.heatmap(data.corr(), annot= True)
```

[21]: <AxesSubplot:>



1 Project Task: Week 2

1.1 Data Modeling:

1.1.1 1. Devise strategies for model building. It is important to decide the right validation framework. Express your thought process.

Apply an appropriate classification algorithm to build a model.

Compare various models with the results from KNN algorithm.

Create a classification report by analyzing sensitivity, specificity, AUC (ROC curve), etc.

Please be descriptive to explain what values of these parameter you have used.

1.1.2 The logistic regression will suit best as our dependent value is categorical data and independent variable are continuos.

[22]:
```
#training and test data
X= data[["Pregnancies","Glucose", 'BloodPressure', 'SkinThickness', 'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age']]
y= data[["Outcome"]]

#Importing "train_test-split" function to test the model
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_state=42)
```

[23]:
```
print("shape of train data is", X_train.shape)
print("shape of test data is", X_test.shape)
```

```
shape of train data is (537, 8)
shape of test data is (231, 8)
```

[24]:
```
#importing Logistic Regression
from sklearn.linear_model import LogisticRegression

lr = LogisticRegression()

#Fit the model in train and test data
lr.fit(X_train,y_train).score(X_train,y_train)
```

```
C:\Users\03man\anaconda3\lib\site-packages\sklearn\utils\validation.py:993:
DataConversionWarning: A column-vector y was passed when a 1d array was
expected. Please change the shape of y to (n_samples, ), for example using
ravel().
  y = column_or_1d(y, warn=True)
C:\Users\03man\anaconda3\lib\site-
packages\sklearn\linear_model\_logistic.py:814: 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
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

[24]: 0.7877094972067039

[25]:
```python
#Now fitting the model in test set
prediction=lr.predict(X_test)
```

[26]:
```python
#Printing first 5 rows after fitting the model in test set
print (X_test.head())
```

```
     Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
668            6       98             58             33    190.0  34.0
324            2      112             75             32     30.5  35.7
624            2      108             64             23     30.5  30.8
690            8      107             80             23     30.5  24.6
473            7      136             90             23     30.5  29.9

     DiabetesPedigreeFunction  Age
668                     0.430   43
324                     0.148   21
624                     0.158   21
690                     0.856   34
473                     0.210   50
```

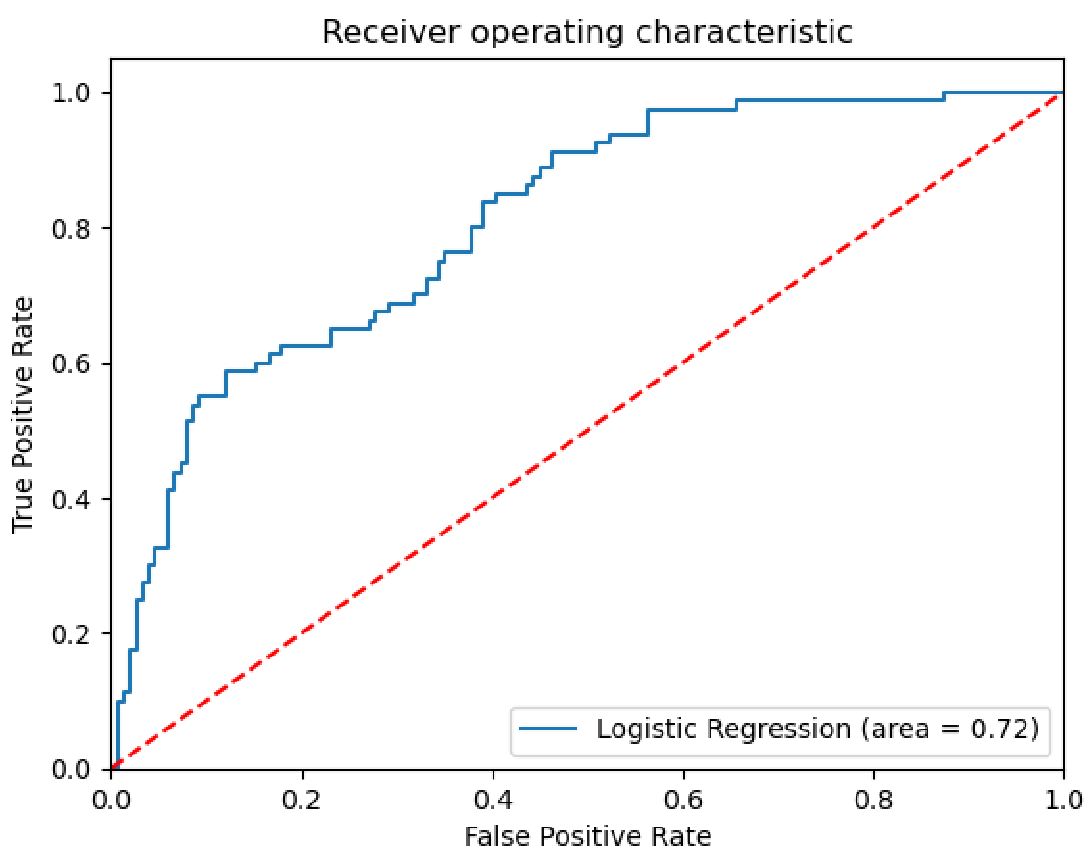
[27]:
```python
from sklearn import metrics
cm = metrics.confusion_matrix(y_test, prediction)
print('\n', "Confusion metrics is ",'\n', cm, '\n')
accuracy = metrics.accuracy_score(y_test, prediction)
print( '\n', "Accuracy score of logistic regression is :",accuracy, '\n')

print ( "classification score is",'\n',   metrics.classification_report(y_test,␣
↪prediction))
```

```
 Confusion metrics is
 [[125  26]
 [ 31  49]]


 Accuracy score of logistic regression is : 0.7532467532467533

classification score is
               precision    recall  f1-score   support

           0       0.80      0.83      0.81       151
           1       0.65      0.61      0.63        80

    accuracy                           0.75       231
   macro avg       0.73      0.72      0.72       231
weighted avg       0.75      0.75      0.75       231
```

1.1.3 Logistic Regression gives 75% acccuracy

[28]:
```
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
lr_roc_auc = roc_auc_score(y_test, lr.predict(X_test))
fpr, tpr, thresholds = roc_curve(y_test, lr.predict_proba(X_test)[:,1])
plt.figure()
plt.plot(fpr, tpr, label='Logistic Regression (area = %0.2f)' % lr_roc_auc)
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
plt.savefig('Log_ROC')
print('AUC: %.3f' % lr_roc_auc)
plt.show()
```

```
AUC: 0.720
```



2 SVM Model

[29]:
```
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
```

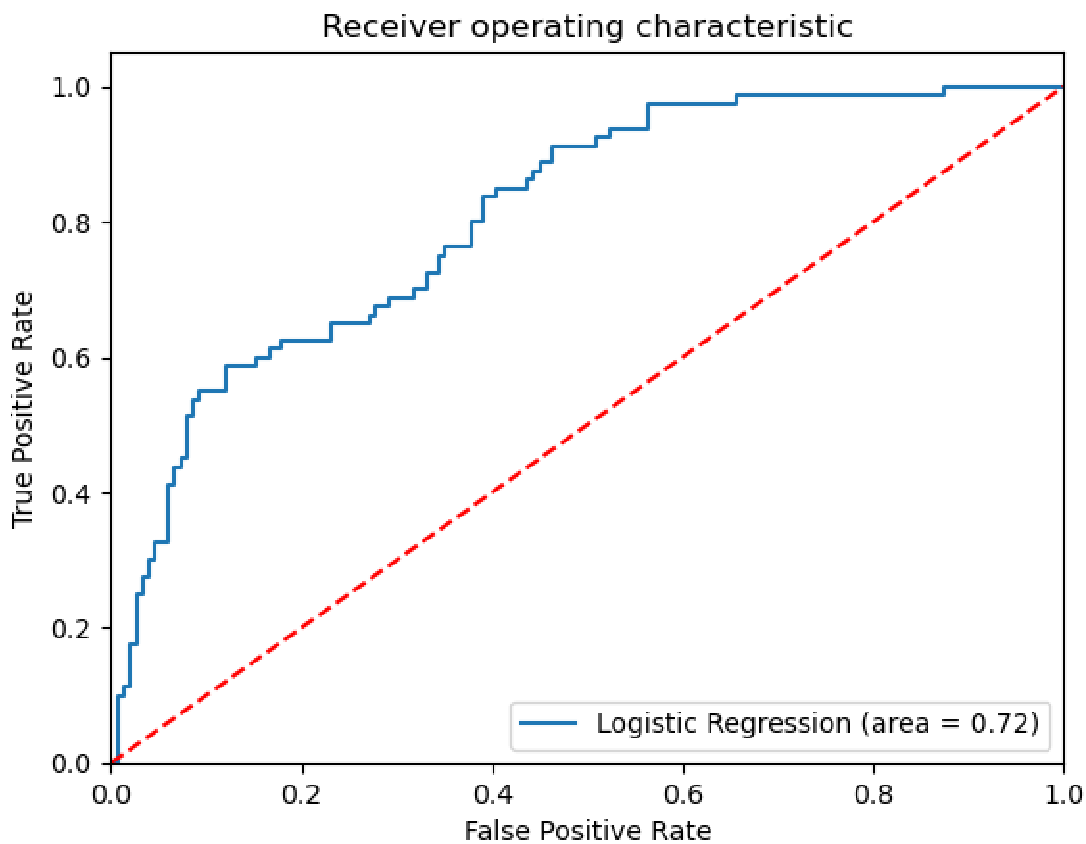
[30]:
```
model=SVC()
model.fit(X_train, y_train)
#creating the predictions on the test data i.e. X_test
y_pred =model.predict(X_test)
accuracy= accuracy_score(y_test, y_pred)
print("Accuracy for the SVM Classifier: {:.3f}".format(accuracy)) #.3f is upto␣
↪3 places decimal
```

```
Accuracy for the SVM Classifier: 0.740
```

```
C:\Users\03man\anaconda3\lib\site-packages\sklearn\utils\validation.py:993:
DataConversionWarning: A column-vector y was passed when a 1d array was
expected. Please change the shape of y to (n_samples, ), for example using
ravel().
  y = column_or_1d(y, warn=True)
```

[31]:
```
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
lr_roc_auc = roc_auc_score(y_test, lr.predict(X_test))
fpr, tpr, thresholds = roc_curve(y_test, lr.predict_proba(X_test)[:,1])
plt.figure()
plt.plot(fpr, tpr, label='Logistic Regression (area = %0.2f)' % lr_roc_auc)
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
plt.savefig('Log_ROC')
print('AUC: %.3f' % lr_roc_auc)
plt.show()
```

```
AUC: 0.720
```



3 KNN Model

[32]:
```
#feature Scaling
from sklearn.preprocessing import StandardScaler
st_X= StandardScaler()
X_train= st_X.fit_transform(X_train)
X_test= st_X.transform(X_test)
```

[33]:
```
#Fitting K-NN classifier to the training set
from sklearn.neighbors import KNeighborsClassifier
classifier= KNeighborsClassifier(n_neighbors=5, metric='minkowski', p=2 )
classifier.fit(X_train, y_train)
```

```
C:\Users\03man\anaconda3\lib\site-
packages\sklearn\neighbors\_classification.py:198: DataConversionWarning: A
column-vector y was passed when a 1d array was expected. Please change the shape
of y to (n_samples,), for example using ravel().
  return self._fit(X, y)
```

[33]: KNeighborsClassifier()

[34]:
```
#Predicting the test set result
y_pred= classifier.predict(X_test)
```

[35]:
```
#Creating the Confusion matrix
from sklearn.metrics import confusion_matrix , classification_report
cm= confusion_matrix(y_test, y_pred)
cm
```

[35]:
```
array([[120,  31],
       [ 34,  46]], dtype=int64)
```

[36]:
```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.78      0.79      0.79       151
           1       0.60      0.57      0.59        80

    accuracy                           0.72       231
   macro avg       0.69      0.68      0.69       231
weighted avg       0.72      0.72      0.72       231
```

[37]:
```
accuracy= accuracy_score(y_test, y_pred)
print("Accuracy for KNN model is: {:.2f}".format(accuracy)) #.2f is upto 2␣
↪places decimal
```

```
Accuracy for KNN model is: 0.72
```

4 Random Forest

[38]:
```
#fitting model
from sklearn.ensemble import RandomForestClassifier
clf= RandomForestClassifier(criterion="gini", #gini is entropy , y we use? we␣
↪see how much gini index reducing for this
                            max_depth= 7,
                            n_estimators= 200,
                            random_state= 5)

#the min. no. of trees in the forest should be atleast 40-50 , so n_estimator␣
↪=200 or nything more then 50 u can take
# the gini index heps us to understand the highest IG or the lowest entropy of␣
↪each feature. So essentiaully
# it will help us select the most imp variablles in our model.
```

```
# max depth allow is 3, 5,7)
```

[39]:
```
#fitting the training data
clf.fit(X_train, y_train)
```

```
C:\Users\03man\AppData\Local\Temp\ipykernel_30556\3181438177.py:2:
DataConversionWarning: A column-vector y was passed when a 1d array was
expected. Please change the shape of y to (n_samples,), for example using
ravel().
  clf.fit(X_train, y_train)
```

[39]:
```
RandomForestClassifier(max_depth=7, n_estimators=200, random_state=5)
```

[40]:
```
clf.feature_importances_
```

[40]:
```
array([0.07197571, 0.32375323, 0.06285006, 0.06634026, 0.06758    ,
       0.16750927, 0.09165808, 0.14833338])
```

[41]:
```
data.columns
```

[41]:
```
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
      dtype='object')
```

5 Glucose is very important as the value is max

[42]:
```
y_pred= clf.predict(X_test)
```

[43]:
```
from sklearn.metrics import confusion_matrix
confusion_matrix(y_test, y_pred)
```

[43]:
```
array([[125,  26],
       [ 31,  49]], dtype=int64)
```

[44]:
```
#find accuracy score

from sklearn.metrics import accuracy_score
print("Accuracy of Random Forest is ", accuracy_score(y_test, y_pred))
```

```
Accuracy of Random Forest is  0.7532467532467533
```

6 decision Tree

[45]:
```
#Fitting a decision tree clasifier
from sklearn.tree import DecisionTreeClassifier
dtree = DecisionTreeClassifier()
dtree.fit(X_train,y_train)
```

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

[46]:
```
#test the accuracy of the decision tree
predictions=dtree.predict(X_test)
from sklearn.metrics import classification_report,confusion_matrix
print(classification_report(y_test,predictions))
```

```
              precision    recall  f1-score   support

           0       0.83      0.72      0.77       151
           1       0.57      0.72      0.64        80

    accuracy                           0.72       231
   macro avg       0.70      0.72      0.70       231
weighted avg       0.74      0.72      0.72       231
```

[47]:
```
from sklearn.metrics import accuracy_score
print("Accuracy of Decision Tree is ", accuracy_score(y_test, predictions))
```

```
Accuracy of Decision Tree is  0.7186147186147186
```