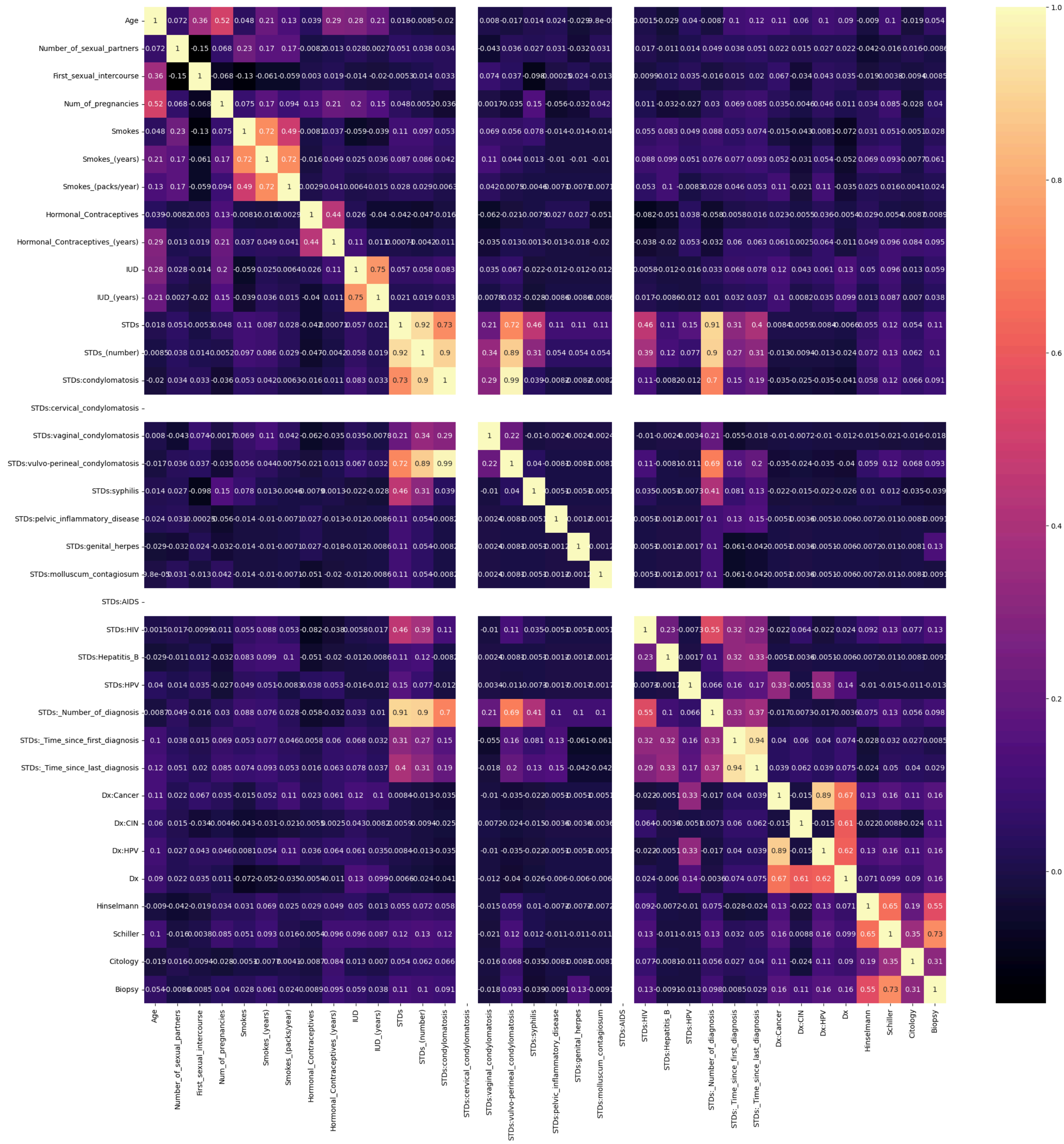
<Axes: >



Logistic Regresson

```
1 import pandas as pd
2 import numpy as np
3 import seaborn as sns
4 import matplotlib.pyplot as plt
5 %matplotlib inline
```

Types of Variable

```
cercan_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 835 entries, 0 to 857
Data columns (total 36 columns):
 #   Column                              Non-Null Count  Dtype
---  ------                              --------------  -----
 0   Age                                 835 non-null    int64
 1   Number_of_sexual_partners           835 non-null    float64
 2   First_sexual_intercourse            835 non-null    float64
 3   Num_of_pregnancies                  835 non-null    float64
 4   Smokes                              835 non-null    float64
 5   Smokes_(years)                      835 non-null    float64
 6   Smokes_(packs/year)                 835 non-null    float64
 7   Hormonal_Contraceptives             835 non-null    float64
 8   Hormonal_Contraceptives_(years)     835 non-null    float64
 9   IUD                                 835 non-null    float64
 10  IUD_(years)                         835 non-null    float64
 11  STDs                                835 non-null    float64
 12  STDs_(number)                       835 non-null    float64
 13  STDs:condylomatosis                 835 non-null    float64
 14  STDs:cervical_condylomatosis        835 non-null    float64
 15  STDs:vaginal_condylomatosis         835 non-null    float64
 16  STDs:vulvo-perineal_condylomatosis  835 non-null    float64
 17  STDs:syphilis                       835 non-null    float64
 18  STDs:pelvic_inflammatory_disease    835 non-null    float64
 19  STDs:genital_herpes                 835 non-null    float64
 20  STDs:molluscum_contagiosum          835 non-null    float64
 21  STDs:AIDS                           835 non-null    float64
 22  STDs:HIV                            835 non-null    float64
 23  STDs:Hepatitis_B                    835 non-null    float64
 24  STDs:HPV                            835 non-null    float64
 25  STDs:_Number_of_diagnosis           835 non-null    int64
 26  STDs:_Time_since_first_diagnosis    835 non-null    float64
 27  STDs:_Time_since_last_diagnosis     835 non-null    float64
 28  Dx:Cancer                           835 non-null    int64
 29  Dx:CIN                              835 non-null    int64
 30  Dx:HPV                              835 non-null    int64
 31  Dx                                  835 non-null    int64
 32  Hinselmann                          835 non-null    int64
 33  Schiller                            835 non-null    int64
 34  Citology                            835 non-null    int64
 35  Biopsy                              835 non-null    int64
dtypes: float64(26), int64(10)
memory usage: 241.4 KB
```

```
# find the categorical variables
cat = [i for i in cercan_df.columns if cercan_df[i].dtype=='O']
print(f'There are {len(cat)} categorical variables\n')
print('The categorical variables are: ', cat)
```

```
There are 0 categorical variables

The categorical variables are:  []
```

Explore Numerical Variables

```
# find the numerical variables
num = [i for i in cercan_df.columns if cercan_df[i].dtype != 'O']
print(f'There are {len(num)} numerical variables\n')
print('The numerical variables are:', num)
```

```
There are 36 numerical variables

The numerical variables are: ['Age', 'Number_of_sexual_partners', 'First_sexual_intercourse', 'Num_of_pregnancies', 'Smokes', 'Smoke
```

```
# checking if the length of the numerical is the same length as the dataframe
col = [i for i in cercan_df.columns]
len(col)
```

```
36
```

Outliers in numerical values

```
# creating a function to check all of the columns that has outliers
def outliers(data):
```

```
    outliers=[]
    for i in data.columns:
        low_bound = data[i].quantile(0.25) - (1.5 * (data[i].quantile(0.75) - data[i].quantile(0.25)))
        upper_bound = data[i].quantile(0.75) + (1.5 * (data[i].quantile(0.75) - data[i].quantile(0.25)))
        if ((data[i] < low_bound) |  (data[i] > upper_bound)).any():
            outliers.append(i)
    return outliers
```

```
# checking all the columns in the dataframe
outliers(cercan_df)
```

```
['Age',
 'Number_of_sexual_partners',
 'First_sexual_intercourse',
 'Num_of_pregnancies',
 'Smokes',
 'Smokes_(years)',
 'Smokes_(packs/year)',
 'Hormonal_Contraceptives_(years)',
 'IUD',
 'IUD_(years)',
 'STDs',
 'STDs_(number)',
 'STDs:condylomatosis',
 'STDs:vaginal_condylomatosis',
 'STDs:vulvo-perineal_condylomatosis',
 'STDs:syphilis',
 'STDs:pelvic_inflammatory_disease',
 'STDs:genital_herpes',
 'STDs:molluscum_contagiosum',
 'STDs:HIV',
 'STDs:Hepatitis_B',
 'STDs:HPV',
 'STDs:_Number_of_diagnosis',
 'STDs:_Time_since_first_diagnosis',
 'STDs:_Time_since_last_diagnosis',
 'Dx:Cancer',
 'Dx:CIN',
 'Dx:HPV',
 'Dx',
 'Hinselmann',
 'Schiller',
 'Citology',
 'Biopsy']
```

```
# checking to see which ones are booleans as to not include in the boxplot
cercan_df.describe(), 2
```

```
min     13.000000                  1.000000                10.000000
25%     21.000000                  2.000000                15.000000
50%     26.000000                  2.000000                17.000000
75%     32.000000                  3.000000                18.000000
max     84.000000                 28.000000                32.000000

       Num_of_pregnancies      Smokes  Smokes_(years)  Smokes_(packs/year)  \
count          835.000000  835.000000      835.000000           835.000000
mean             2.283832    0.147305        1.234329             0.458571
std              1.408152    0.354623        4.111264             2.239363
min              0.000000    0.000000        0.000000             0.000000
25%              1.000000    0.000000        0.000000             0.000000
50%              2.000000    0.000000        0.000000             0.000000
75%              3.000000    0.000000        0.000000             0.000000
max             11.000000    1.000000       37.000000            37.000000

       Hormonal_Contraceptives  Hormonal_Contraceptives_(years)         IUD  \
count               835.000000                       835.000000  835.000000
mean                  0.651639                         2.080520    0.099401
std                   0.446366                         3.601364    0.299379
min                   0.000000                         0.000000    0.000000
25%                   0.000000                         0.000000    0.000000
50%                   1.000000                         0.500000    0.000000
75%                   1.000000                         3.000000    0.000000
max                   1.000000                        30.000000    1.000000

       ...  STDs:_Time_since_first_diagnosis  STDs:_Time_since_last_diagnosis  \
count  ...                        835.000000                       835.000000
mean   ...                          4.182036                         3.239521
std    ...                          1.809358                         1.843420
min    ...                          1.000000                         1.000000
25%    ...                          4.000000                         3.000000
50%    ...                          4.000000                         3.000000
75%    ...                          4.000000                         3.000000
max    ...                         22.000000                        22.000000
```

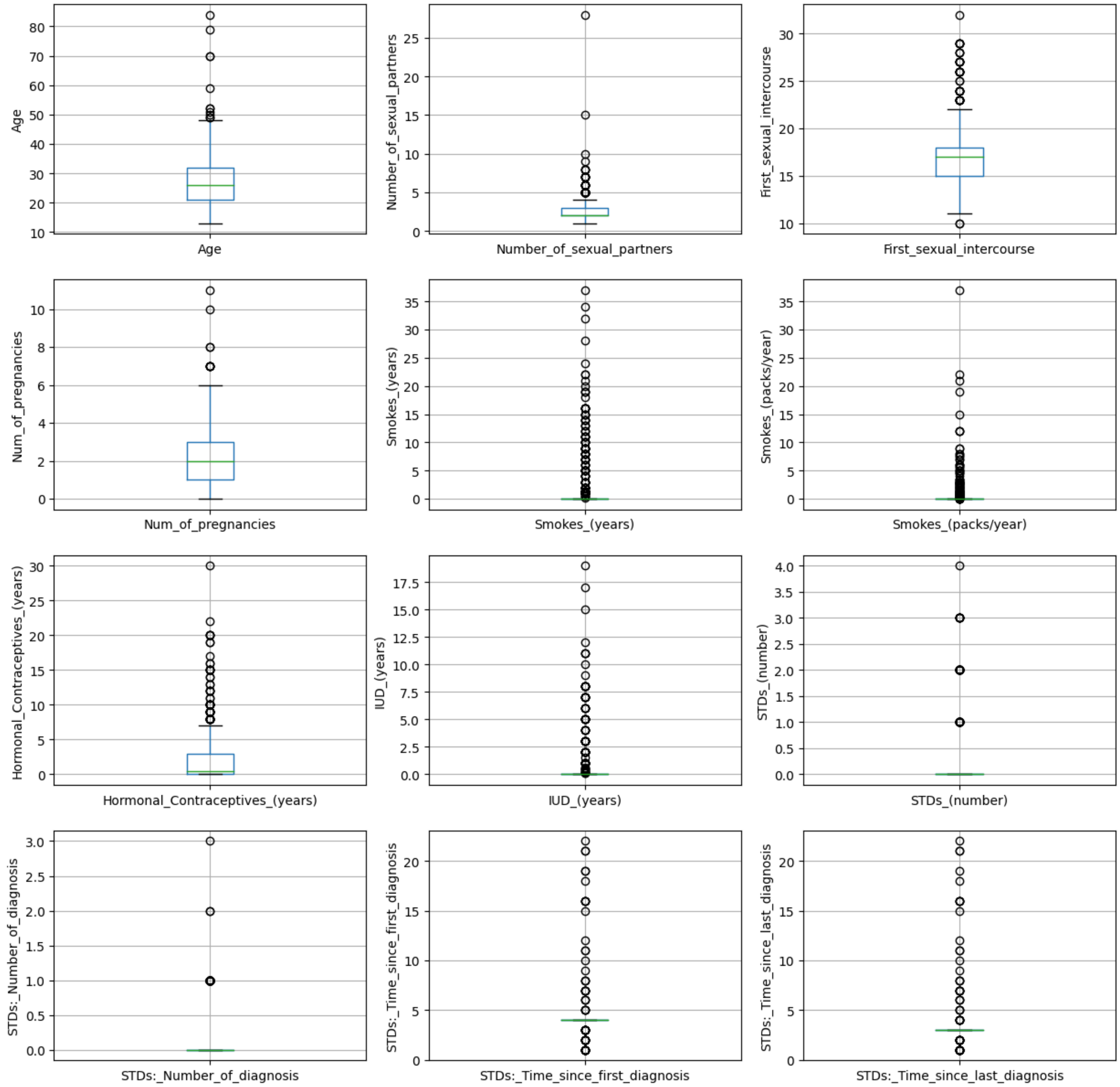
```
std      0.145319    0.103320    0.145319    0.167182    0.200518    0.282626
min      0.000000    0.000000    0.000000    0.000000    0.000000    0.000000
25%      0.000000    0.000000    0.000000    0.000000    0.000000    0.000000
50%      0.000000    0.000000    0.000000    0.000000    0.000000    0.000000
75%      0.000000    0.000000    0.000000    0.000000    0.000000    0.000000
max      1.000000    1.000000    1.000000    1.000000    1.000000    1.000000

         Citology      Biopsy
count  835.000000  835.000000
mean     0.051497    0.064671
std      0.221142    0.246091
min      0.000000    0.000000
25%      0.000000    0.000000
50%      0.000000    0.000000
75%      0.000000    0.000000
max      1.000000    1.000000

[8 rows x 36 columns],
2)
```

```
1 cercan_df['Smokes_(years)'].value_counts()
```

```
Smokes_(years)
0.000000     712
1.266973      15
5.000000       9
9.000000       9
1.000000       8
3.000000       7
2.000000       7
16.000000      6
7.000000       6
8.000000       6
11.000000      5
4.000000       5
10.000000      5
14.000000      4
15.000000      4
6.000000       4
13.000000      3
0.500000       3
19.000000      3
12.000000      3
22.000000      2
32.000000      1
20.000000      1
28.000000      1
24.000000      1
18.000000      1
34.000000      1
21.000000      1
37.000000      1
0.160000       1
Name: count, dtype: int64
```

```
# creating a boxplot for each column that is not boolean and has an outlier
plt.figure(figsize=(15,15))

plt.subplot(4, 3, 1)
fig = cercan_df.boxplot(column='Age')
fig.set_title('')
fig.set_ylabel('Age')

plt.subplot(4, 3, 2)
fig = cercan_df.boxplot(column='Number_of_sexual_partners')
fig.set_title('')
fig.set_ylabel('Number_of_sexual_partners')

plt.subplot(4, 3, 3)
fig = cercan_df.boxplot(column='First_sexual_intercourse')
fig.set_title('')
fig.set_ylabel('First_sexual_intercourse')

plt.subplot(4, 3, 4)
fig = cercan_df.boxplot(column='Num_of_pregnancies')
fig.set_title('')
fig.set_ylabel('Num_of_pregnancies')

plt.subplot(4, 3, 5)
fig = cercan_df.boxplot(column='Smokes_(years)')
fig.set_title('')
fig.set_ylabel('Smokes_(years)')

plt.subplot(4, 3, 6)
fig = cercan_df.boxplot(column='Smokes_(packs/year)')
fig.set_title('')
fig.set_ylabel('Smokes_(packs/year)')

plt.subplot(4, 3, 7)
fig = cercan_df.boxplot(column='Hormonal_Contraceptives_(years)')
fig.set_title('')
fig.set_ylabel('Hormonal_Contraceptives_(years)')

plt.subplot(4, 3, 8)
fig = cercan_df.boxplot(column='IUD_(years)')
fig.set_title('')
fig.set_ylabel('IUD_(years)')

plt.subplot(4, 3, 9)
fig = cercan_df.boxplot(column='STDs_(number)')
fig.set_title('')
fig.set_ylabel('STDs_(number)')

plt.subplot(4, 3, 10)
fig = cercan_df.boxplot(column='STDs:_Number_of_diagnosis')
fig.set_title('')
fig.set_ylabel('STDs:_Number_of_diagnosis')

plt.subplot(4, 3, 11)
fig = cercan_df.boxplot(column='STDs:_Time_since_first_diagnosis')
fig.set_title('')
fig.set_ylabel('STDs:_Time_since_first_diagnosis')

plt.subplot(4, 3, 12)
fig = cercan_df.boxplot(column='STDs:_Time_since_last_diagnosis')
fig.set_title('')
fig.set_ylabel('STDs:_Time_since_last_diagnosis')
```

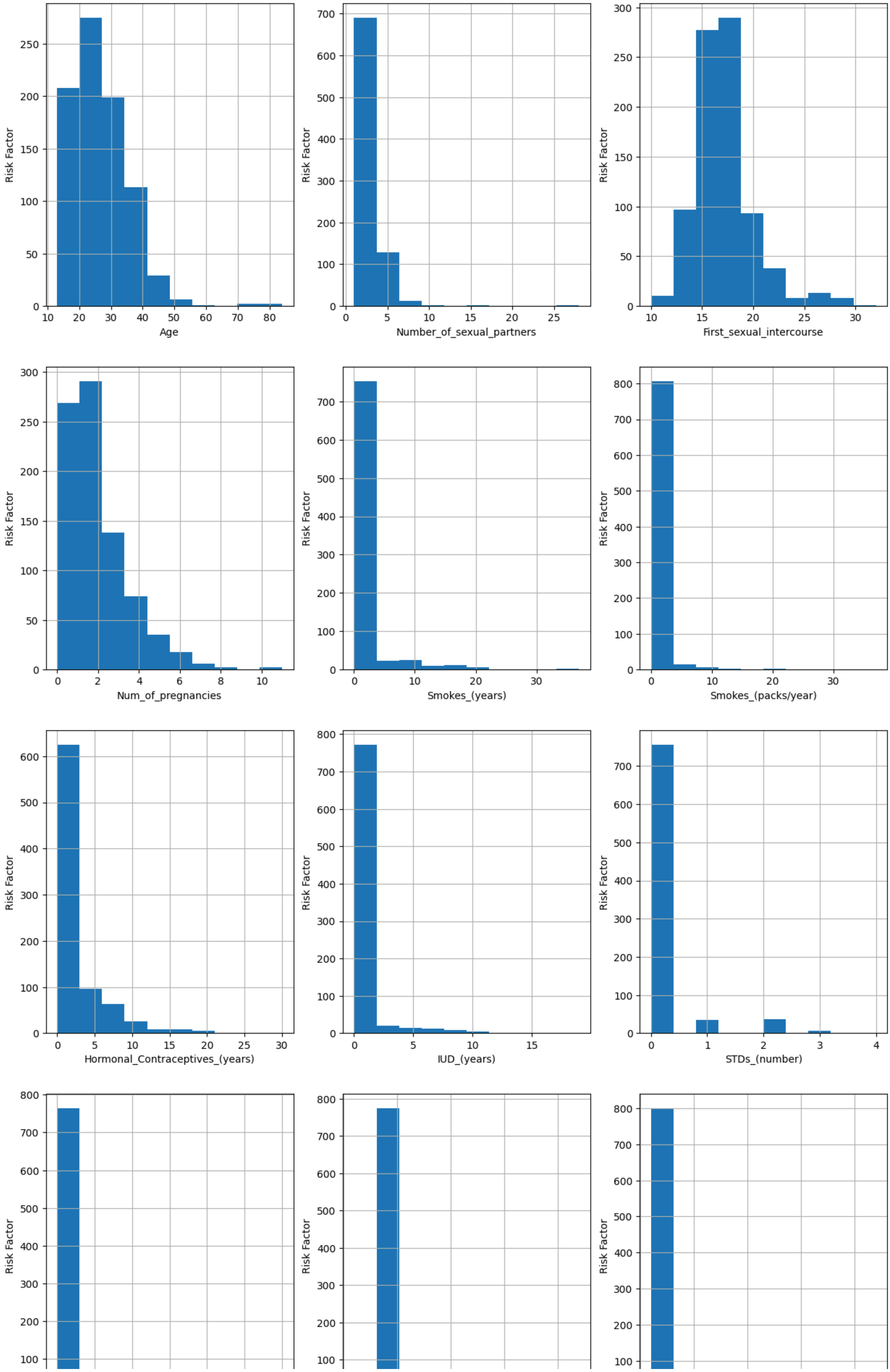
```
Text(0, 0.5, 'STDs:_Time_since_last_diagnosis')
```

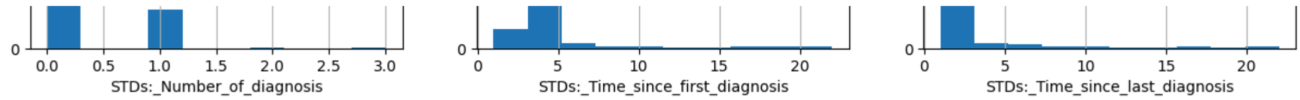


Check the distribution of variables

```
# creating a boxplot for each column that is not boolean and has an outlier
# plotting the histogram to check the outliers
plt.figure(figsize=(15,25))

plt.subplot(4, 3, 1)
fig = cercan_df['Age'].hist(bins=10)
fig.set_xlabel('Age')
fig.set_ylabel('Risk Factor')

plt.subplot(4, 3, 2)
fig = cercan_df['Number_of_sexual_partners'].hist(bins=10)
fig.set_xlabel('Number_of_sexual_partners')
fig.set_ylabel('Risk Factor')

plt.subplot(4, 3, 3)
fig = cercan_df['First_sexual_intercourse'].hist(bins=10)
fig.set_xlabel('First_sexual_intercourse')
fig.set_ylabel('Risk Factor')

plt.subplot(4, 3, 4)
fig = cercan_df['Num_of_pregnancies'].hist(bins=10)
fig.set_xlabel('Num_of_pregnancies')
fig.set_ylabel('Risk Factor')


plt.subplot(4, 3, 5)
fig = cercan_df['Smokes_(years)'].hist(bins=10)
fig.set_xlabel('Smokes_(years)')
fig.set_ylabel('Risk Factor')

plt.subplot(4, 3, 6)
fig = cercan_df['Smokes_(packs/year)'].hist(bins=10)
fig.set_xlabel('Smokes_(packs/year)')
fig.set_ylabel('Risk Factor')

plt.subplot(4, 3, 7)
fig = cercan_df['Hormonal_Contraceptives_(years)'].hist(bins=10)
fig.set_xlabel('Hormonal_Contraceptives_(years)')
fig.set_ylabel('Risk Factor')

plt.subplot(4, 3, 8)
fig = cercan_df['IUD_(years)'].hist(bins=10)
fig.set_xlabel('IUD_(years)')
fig.set_ylabel('Risk Factor')

plt.subplot(4, 3, 9)
fig = cercan_df['STDs_(number)'].hist(bins=10)
fig.set_xlabel('STDs_(number)')
fig.set_ylabel('Risk Factor')

plt.subplot(4, 3, 10)
fig = cercan_df['STDs:_Number_of_diagnosis'].hist(bins=10)
fig.set_xlabel('STDs:_Number_of_diagnosis')
fig.set_ylabel('Risk Factor')

plt.subplot(4, 3, 11)
fig = cercan_df['STDs:_Time_since_first_diagnosis'].hist(bins=10)
fig.set_xlabel('STDs:_Time_since_first_diagnosis')
fig.set_ylabel('Risk Factor')

plt.subplot(4, 3, 12)
fig = cercan_df['STDs:_Time_since_last_diagnosis'].hist(bins=10)
fig.set_xlabel('STDs:_Time_since_last_diagnosis')
fig.set_ylabel('Risk Factor')
```

```
Text(0, 0.5, 'Risk Factor')
```





Since all outliers are scewed, interquartile range would be used to find the outliers.

```
low_bound = cercan_df['Age'].quantile(0.25) - (1.5 * (cercan_df['Age'].quantile(0.75) - cercan_df['Age'].quantile(0.25)))
upper_bound = cercan_df['Age'].quantile(0.75) + (1.5 * (cercan_df['Age'].quantile(0.75) - cercan_df['Age'].quantile(0.25)))
print(f'Age outliers are value < {low_bound} or > {upper_bound}')
```

```
Age outliers are value < 4.5 or > 48.5
```

```
low_bound = cercan_df['Number_of_sexual_partners'].quantile(0.25) - (1.5 * (cercan_df['Number_of_sexual_partners'].quantile(0.75) - c
upper_bound = cercan_df['Number_of_sexual_partners'].quantile(0.75) + (1.5 * (cercan_df['Number_of_sexual_partners'].quantile(0.75) -
print(f'Number_of_sexual_partners outliers are value < {low_bound} or > {upper_bound}')
```

```
Number_of_sexual_partners outliers are value < 0.5 or > 4.5
```

```
low_bound = cercan_df['First_sexual_intercourse'].quantile(0.25) - (1.5 * (cercan_df['First_sexual_intercourse'].quantile(0.75) - cer
upper_bound = cercan_df['First_sexual_intercourse'].quantile(0.75) + (1.5 * (cercan_df['First_sexual_intercourse'].quantile(0.75) - c
print(f'First_sexual_intercourse outliers are value < {low_bound} or > {upper_bound}')
```

```
First_sexual_intercourse outliers are value < 10.5 or > 22.5
```

```
low_bound = cercan_df['Num_of_pregnancies'].quantile(0.25) - (1.5 * (cercan_df['Num_of_pregnancies'].quantile(0.75) - cercan_df['Num_
upper_bound = cercan_df['Num_of_pregnancies'].quantile(0.75) + (1.5 * (cercan_df['Num_of_pregnancies'].quantile(0.75) - cercan_df['Nu
print(f'Num_of_pregnancies outliers are value < {low_bound} or > {upper_bound}')
```

```
Num_of_pregnancies outliers are value < -2.0 or > 6.0
```

```
low_bound = cercan_df['Smokes_(years)'].quantile(0.25) - (1.5 * (cercan_df['Smokes_(years)'].quantile(0.75) - cercan_df['Smokes_(year
upper_bound = cercan_df['Smokes_(years)'].quantile(0.75) + (1.5 * (cercan_df['Smokes_(years)'].quantile(0.75) - cercan_df['Smokes_(ye
print(f'Smokes_(years) outliers are value < {low_bound} or > {upper_bound}')
```

```
Smokes_(years) outliers are value < 0.0 or > 0.0
```

```
cercan_df['Smokes_(years)'].value_counts()
```

```
Smokes_(years)
0.000000     712
1.266973      15
5.000000       9
9.000000       9
1.000000       8
3.000000       7
2.000000       7
16.000000      6
7.000000       6
8.000000       6
11.000000      5
4.000000       5
10.000000      5
14.000000      4
15.000000      4
6.000000       4
13.000000      3
0.500000       3
19.000000      3
12.000000      3
22.000000      2
32.000000      1
20.000000      1
28.000000      1
24.000000      1
```

```
18.000000      1
34.000000      1
21.000000      1
37.000000      1
0.160000       1
Name: count, dtype: int64
```

```
low_bound = cercan_df['Smokes_(packs/year)'].quantile(0.25) - (1.5 * (cercan_df['Smokes_(packs/year)'].quantile(0.75) - cercan_df['Sm
upper_bound = cercan_df['Smokes_(packs/year)'].quantile(0.75) + (1.5 * (cercan_df['Smokes_(packs/year)'].quantile(0.75) - cercan_df[
print(f'Smokes_(packs/year) outliers are value < {low_bound} or > {upper_bound}')
```

```
Smokes_(packs/year) outliers are value < 0.0 or > 0.0
```

```
cercan_df['Smokes_(packs/year)'].value_counts()
```

```
Smokes_(packs/year)
0.000000     712
0.513202      18
1.000000       6
3.000000       5
2.000000       4
            ...
7.500000       1
37.000000      1
2.250000       1
0.003000       1
0.300000       1
Name: count, Length: 62, dtype: int64
```

```
low_bound = cercan_df['Hormonal_Contraceptives_(years)'].quantile(0.25) - (1.5 * (cercan_df['Hormonal_Contraceptives_(years)'].quantil
upper_bound = cercan_df['Hormonal_Contraceptives_(years)'].quantile(0.75) + (1.5 * (cercan_df['Hormonal_Contraceptives_(years)'].quant
print(f'Hormonal_Contraceptives_(years) outliers are value < {low_bound} or > {upper_bound}')
```

```
Hormonal_Contraceptives_(years) outliers are value < -4.5 or > 7.5
```

```
low_bound = cercan_df['IUD_(years)'].quantile(0.25) - (1.5 * (cercan_df['IUD_(years)'].quantile(0.75) - cercan_df['IUD_(years)'].quant
upper_bound = cercan_df['IUD_(years)'].quantile(0.75) + (1.5 * (cercan_df['IUD_(years)'].quantile(0.75) - cercan_df['IUD_(years)'].qua
print(f'IUD_(years) outliers are value < {low_bound} or > {upper_bound}')
```

```
IUD_(years) outliers are value < 0.0 or > 0.0
```

```
cercan_df['IUD_(years)'].value_counts()
```

```
IUD_(years)
0.00     752
3.00      11
2.00      10
5.00       9
1.00       8
8.00       7
7.00       7
6.00       5
4.00       5
11.00      3
0.08       2
0.50       2
0.33       1
9.00       1
0.41       1
0.16       1
0.91       1
1.50       1
10.00      1
12.00      1
15.00      1
0.25       1
17.00      1
19.00      1
0.58       1
0.17       1
Name: count, dtype: int64
```

```
low_bound = cercan_df['STDs_(number)'].quantile(0.25) - (1.5 * (cercan_df['STDs_(number)'].quantile(0.75) - cercan_df['STDs_(number)
upper_bound = cercan_df['STDs_(number)'].quantile(0.75) + (1.5 * (cercan_df['STDs_(number)'].quantile(0.75) - cercan_df['STDs_(number
print(f'STDs_(number) outliers are value < {low_bound} or > {upper_bound}')
```

```
STDs_(number) outliers are value < 0.0 or > 0.0
```

```
cercan_df['STDs_(number)'].value_counts()
```

```
STDs_(number)
0.0    756
```

```
2.0     37
1.0     34
3.0      7
4.0      1
Name: count, dtype: int64
```

```
low_bound = cercan_df['STDs:_Number_of_diagnosis'].quantile(0.25) - (1.5 * (cercan_df['STDs:_Number_of_diagnosis'].quantile(0.75) - (
upper_bound = cercan_df['STDs:_Number_of_diagnosis'].quantile(0.75) + (1.5 * (cercan_df['STDs:_Number_of_diagnosis'].quantile(0.75) -
print(f'STDs:_Number_of_diagnosis outliers are value < {low_bound} or > {upper_bound}')
```

```
STDs:_Number_of_diagnosis outliers are value < 0.0 or > 0.0
```

```
cercan_df['STDs:_Number_of_diagnosis'].value_counts()
```

```
STDs:_Number_of_diagnosis
0    764
1     68
2      2
3      1
Name: count, dtype: int64
```

```
low_bound = cercan_df['STDs:_Time_since_first_diagnosis'].quantile(0.25) - (1.5 * (cercan_df['STDs:_Time_since_first_diagnosis'].quan
upper_bound = cercan_df['STDs:_Time_since_first_diagnosis'].quantile(0.75) + (1.5 * (cercan_df['STDs:_Time_since_first_diagnosis'].qu
print(f'STDs:_Time_since_first_diagnosis outliers are value < {low_bound} or > {upper_bound}')
```

```
STDs:_Time_since_first_diagnosis outliers are value < 4.0 or > 4.0
```

```
low_bound = cercan_df['STDs:_Time_since_last_diagnosis'].quantile(0.25) - (1.5 * (cercan_df['STDs:_Time_since_last_diagnosis'].quant:
upper_bound = cercan_df['STDs:_Time_since_last_diagnosis'].quantile(0.75) + (1.5 * (cercan_df['STDs:_Time_since_last_diagnosis'].quan
print(f'STDs:_Time_since_last_diagnosis outliers are value < {low_bound} or > {upper_bound}')
```

```
STDs:_Time_since_last_diagnosis outliers are value < 3.0 or > 3.0
```

Declare feature vector and target variable

```
# since y does not have a dataframe then there is not target variable
X = cercan_df.drop(['Biopsy'], axis=1)
y = cercan_df['Biopsy']
```

Split data into separate training and test set

```
# split X and y into training and testing sets
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
```

```
# check the shape of X_train and X_test
X_train.shape, X_test.shape
```

```
((668, 35), (167, 35))
```

Engineering outliers in numerical values

```
def max_value(df3, variable, top):
    return np.where(df3[variable]>top, top, df3[variable])

for df3 in [X_train, X_test]:
    df3['Age'] = max_value(df3, 'Age', 48.5)
    df3['Number_of_sexual_partners'] = max_value(df3, 'Number_of_sexual_partners', 4.5)
    df3['First_sexual_intercourse'] = max_value(df3, 'First_sexual_intercourse', 22.5)
    df3['Num_of_pregnancies'] = max_value(df3, 'Num_of_pregnancies', 6.0)
    df3['Hormonal_Contraceptives_(years)'] = max_value(df3, 'Hormonal_Contraceptives_(years)', 7.5)
    df3['STDs:_Time_since_first_diagnosis'] = max_value(df3, 'STDs:_Time_since_first_diagnosis', 4.0)
    df3['STDs:_Time_since_last_diagnosis'] = max_value(df3, 'STDs:_Time_since_last_diagnosis', 3.0)
```

```
X_train['Age'].max(), X_test['Age'].max()
```

```
(48.5, 48.5)
```

```
X_train['Number_of_sexual_partners'].max(), X_test['Number_of_sexual_partners'].max()
```

```
(4.5, 4.5)
```

```
X_train['First_sexual_intercourse'].max(), X_test['First_sexual_intercourse'].max()
```

```
(22.5, 22.5)
```

```
X_train['Num_of_pregnancies'].max(), X_test['Num_of_pregnancies'].max()
```

```
(6.0, 6.0)
```

```
X_train['Hormonal_Contraceptives_(years)'].max(), X_test['Hormonal_Contraceptives_(years)'].max()
```

```
(7.5, 7.5)
```

```
X_train['STDs:_Time_since_first_diagnosis'].max(), X_test['STDs:_Time_since_first_diagnosis'].max()
```

```
(4.0, 4.0)
```

```
X_train['STDs:_Time_since_last_diagnosis'].max(), X_test['STDs:_Time_since_last_diagnosis'].max()
```

```
(3.0, 3.0)
```

Feature Scaling

```
X_train.describe()
```

	Age	Number_of_sexual_partners	First_sexual_intercourse	Num_of_pregnancies	Smokes	Smokes_(years)	Smokes_(packs
count	668.000000	668.000000	668.000000	668.000000	668.000000	668.000000	668
mean	26.738024	2.439371	16.855539	2.232036	0.157186	1.267709	0
std	7.613179	1.096057	2.289344	1.282470	0.364248	4.021316	2
min	13.000000	1.000000	10.000000	0.000000	0.000000	0.000000	0
25%	21.000000	2.000000	15.000000	1.000000	0.000000	0.000000	0
50%	26.000000	2.000000	17.000000	2.000000	0.000000	0.000000	0
75%	32.000000	3.000000	18.000000	3.000000	0.000000	0.000000	0
max	48.500000	4.500000	22.500000	6.000000	1.000000	37.000000	37

8 rows × 35 columns

```
cols = X_train.columns
```

```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.fit_transform(X_test)
```

```
X_train = pd.DataFrame(X_train, columns=[cols])
```

```
X_test = pd.DataFrame(X_test, columns=[cols])
```

```
X_train.describe()
```

	Age	Number_of_sexual_partners	First_sexual_intercourse	Num_of_pregnancies	Smokes	Smokes_(years)	Smokes_(pack
count	668.000000	668.000000	668.000000	668.000000	668.000000	668.000000	668
mean	0.386987	0.411249	0.548443	0.372006	0.157186	0.034262	0
std	0.214456	0.313159	0.183147	0.213745	0.364248	0.108684	0
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0
25%	0.225352	0.285714	0.400000	0.166667	0.000000	0.000000	0
50%	0.366197	0.285714	0.560000	0.333333	0.000000	0.000000	0
75%	0.535211	0.571429	0.640000	0.500000	0.000000	0.000000	0
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1

8 rows × 35 columns

Model Training

```
# train a Logistic regression model on the training set
from sklearn.linear_model import LogisticRegression

# instantiate the model
logreg = LogisticRegression(solver='liblinear', random_state=0)

# fit the model
logreg.fit(X_train, y_train)
```

```
▾ LogisticRegression
LogisticRegression(random_state=0, solver='liblinear')
```

Predict Results

```
y_pred_test = logreg.predict(X_test)
y_pred_test
```

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

predict_proba method

predict_proba gives the probabilities for the target variable(0 and 1) in this case, in array form.

```
logreg.predict_proba(X_test)[:,0]
```

```
array([0.9318181 , 0.98652533, 0.99130247, 0.98709881, 0.97337435,
       0.98413044, 0.76004681, 0.98998675, 0.98483541, 0.98113619,
       0.9624972 , 0.9866606 , 0.99314353, 0.98595472, 0.46235823,
       0.98124161, 0.99011633, 0.98309391, 0.98426304, 0.98100949,
       0.99061539, 0.98285467, 0.93654504, 0.98448176, 0.98529191,
       0.95787997, 0.98420974, 0.98416174, 0.99570361, 0.32386843,
       0.93985344, 0.98433208, 0.69838836, 0.98504967, 0.97806979,
       0.98342582, 0.98714347, 0.98531745, 0.99012043, 0.98423749,
       0.98185636, 0.98413755, 0.98210074, 0.98476949, 0.99141842,
       0.98434966, 0.66625298, 0.98348171, 0.98668871, 0.98701919,
       0.98311955, 0.98468404, 0.98303297, 0.98402773, 0.9943138 ,
       0.93420265, 0.98265801, 0.98430784, 0.98595543, 0.98210833,
       0.9878019 , 0.97685329, 0.98682516, 0.98181535, 0.98219048,
       0.99061445, 0.97967217, 0.96883172, 0.98484432, 0.98316253,
       0.4313646 , 0.98572152, 0.97981937, 0.98093667, 0.94619307,
       0.98570094, 0.98415083, 0.5395997 , 0.98514335, 0.98676751,
       0.98469929, 0.51093354, 0.98627927, 0.98624998, 0.99228935,
       0.97930775, 0.98577665, 0.98356476, 0.98273924, 0.98231977,
       0.99606892, 0.98051242, 0.98454714, 0.98733305, 0.99291951,
       0.98490378, 0.98612827, 0.98558747, 0.99076695, 0.98652352,
       0.20734488, 0.97973866, 0.98043925, 0.98430691, 0.98365371,
       0.98169312, 0.40676658, 0.9850432 , 0.98307627, 0.9808708 ,
       0.98692098, 0.98931307, 0.9775294 , 0.98168244, 0.24084681,
```

```
       0.98666841, 0.34842113, 0.97963817, 0.98476142, 0.98869858,
       0.99007095, 0.9963137 , 0.98373624, 0.9831398 , 0.98558592,
       0.98315002, 0.9846668 , 0.98453749, 0.9881759 , 0.98304431,
       0.98605588, 0.3558348 , 0.99112175, 0.9837479 , 0.98837483,
       0.98249845, 0.21737484, 0.98556725, 0.99022947, 0.98437556,
       0.98384328, 0.98510396, 0.98872053, 0.99450419, 0.99048227,
       0.98461929, 0.9855683 , 0.98945731, 0.98848021, 0.98693139,
       0.95526428, 0.93871452, 0.98777501, 0.99079094, 0.98600817,
       0.99070472, 0.98448459, 0.5043718 , 0.98978009, 0.74300502,
       0.98703476, 0.98187904, 0.98392365, 0.98340128, 0.97905508,
       0.98499232, 0.98789245])
```

```
logreg.predict_proba(X_test)[:,1]
```

```
array([0.0681819 , 0.01347467, 0.00869753, 0.01290119, 0.02662565,
       0.01586956, 0.23995319, 0.01001325, 0.01516459, 0.01886381,
       0.0375028 , 0.0133394 , 0.00685647, 0.01404528, 0.53764177,
       0.01875839, 0.00988367, 0.01690609, 0.01573696, 0.01899051,
       0.00938461, 0.01714533, 0.06345496, 0.01551824, 0.01470809,
       0.04212003, 0.01579026, 0.01583826, 0.00429639, 0.67613157,
       0.06014656, 0.01566792, 0.30161164, 0.01495033, 0.02193021,
       0.01657418, 0.01285653, 0.01468255, 0.00987957, 0.01576251,
       0.01814364, 0.01586245, 0.01789926, 0.01523051, 0.00858158,
       0.01565034, 0.33374702, 0.01651829, 0.01331129, 0.01298081,
       0.01688045, 0.01531596, 0.01696703, 0.01597227, 0.0056862 ,
       0.06579735, 0.01734199, 0.01569216, 0.01404457, 0.01789167,
       0.0121981 , 0.02314671, 0.01317484, 0.01818465, 0.01780952,
       0.00938555, 0.02032783, 0.03116828, 0.01515568, 0.01683747,
       0.5686354 , 0.01427848, 0.02018063, 0.01906333, 0.05380693,
       0.01429906, 0.01584917, 0.4604003 , 0.01485665, 0.01323249,
       0.01530071, 0.48906646, 0.01372073, 0.01375002, 0.00771065,
       0.02069225, 0.01422335, 0.01643524, 0.01726076, 0.01768023,
       0.00393108, 0.01948758, 0.01545286, 0.01266695, 0.00708049,
       0.01509622, 0.01387173, 0.01441253, 0.00923305, 0.01347648,
       0.79265512, 0.02026134, 0.01956075, 0.01569309, 0.01634629,
       0.01830688, 0.59323342, 0.0149568 , 0.01692373, 0.0191292 ,
       0.01307902, 0.01068693, 0.0224706 , 0.01831756, 0.75915319,
       0.01333159, 0.65157887, 0.02036183, 0.01523858, 0.01130142,
       0.00992905, 0.0036863 , 0.01626376, 0.0168602 , 0.01441408,
       0.01684998, 0.0153332 , 0.01546251, 0.0118241 , 0.01695569,
       0.01394412, 0.6441652 , 0.00887825, 0.0162521 , 0.01162517,
       0.01750155, 0.78262516, 0.01443275, 0.00977053, 0.01562444,
       0.01615672, 0.01489604, 0.01127947, 0.00549581, 0.00951773,
       0.01538071, 0.0144317 , 0.01054269, 0.01151979, 0.01306861,
       0.04473572, 0.06128548, 0.01222499, 0.00920906, 0.01399183,
       0.00929528, 0.01551541, 0.4956282 , 0.01021991, 0.25699498,
       0.01296524, 0.01812096, 0.01607635, 0.01659872, 0.02094492,
       0.01500768, 0.01210755])
```

Check accuracy score

```
from sklearn.metrics import accuracy_score

print('Model accuracy score: {0:0.4f}'.format(accuracy_score(y_test, y_pred_test)))
```

```
Model accuracy score: 0.9461
```

Compare the train-set and test-set accuracy

```
y_pred_train = logreg.predict(X_train)
y_pred_train
```

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

```
print('Training-set accuracy score: {0:0.4f}'.format(accuracy_score(y_train, y_pred_train)))
```

```
Training-set accuracy score: 0.9671
```

Check for overfitting and underfitting

```
# print the scores on training and test set
train_score = logreg.score(X_train, y_train)
test_score = logreg.score(X_test, y_test)
print(f'Training set score: {train_score}')
print(f'Test set score: {test_score}')
```

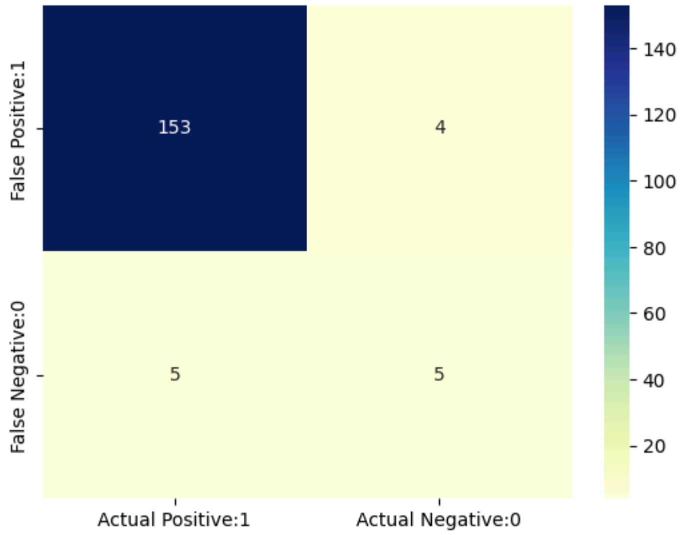
```
Training set score: 0.9670658682634731
Test set score: 0.9461077844311377
```

Confusion Matrix

```
# import the necessary library
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred_test)
```

```
# visualize confusion matrix with seaborn heatmap
cm_matrix = pd.DataFrame(data=cm, columns=['Actual Positive:1', 'Actual Negative:0'],
                         index=['False Positive:1', 'False Negative:0'])

sns.heatmap(cm_matrix, annot=True, fmt='d', cmap='YlGnBu')
```

```
<Axes: >
```



Classification metrices

```
from sklearn.metrics import classification_report

print(classification_report(y_test, y_pred_test))
```

```
              precision    recall  f1-score   support

           0       0.97      0.97      0.97       157
           1       0.56      0.50      0.53        10

    accuracy                           0.95       167
```

```
   macro avg       0.76      0.74      0.75       167
weighted avg       0.94      0.95      0.94       167
```

Classification accuracy

```
TP = cm[0,0]
TN = cm[1,1]
FP = cm[0,1]
FN = cm[1,0]
```

```
# print classification accuracy
classification_accuracy = (TP + TN) / float(TP + TN + FP + FN)
print('Classification accuracy:{0:0.4f}'.format(classification_accuracy))
```

```
Classification accuracy:0.9461
```

Classification error

```
# print classification error
classification_error = (FP + FN) / float(TP + TN + FP + FN)
print('Classification error:{0:0.4f}'.format(classification_error))
```

```
Classification error:0.0539
```

Precision

```
# print precision score
precision = TP / float(TP + FP)
print('Precision:{0:0.4f}'.format(precision))
```

```
Precision:0.9745
```

Recall

```
recall = TP / float(TP + FN)
print('Recall or Sensitivity: {0:0.4f}'.format(recall))
```

```
Recall or Sensitivity: 0.9684
```

True Positive Rate

```
tpr = TP / float(TP + FN)
print('True Positive Rate: {0:0.4f}'.format(tpr))
```

```
True Positive Rate: 0.9684
```

False Positive Rate

```
fpr = FP / float(FP + FN)
print('False Positive Rate: {0:0.4f}'.format(fpr))
```

```
False Positive Rate: 0.4444
```

Specificity

```
specificity = TN / (TN + FP)
print('Specificity: {0:0.4f}'.format(specificity))
```

```
Specificity: 0.5556
```

Adjusting the treshold level

```
# print the first 10 predicted probabilities of two classes- 0 and 1
y_pred_prob = logreg.predict_proba(X_test)[0:10]
y_pred_prob
```

```
array([[0.9318181 , 0.0681819 ],
       [0.98652533, 0.01347467],
       [0.99130247, 0.00869753],
       [0.98709881, 0.01290119],
       [0.97337435, 0.02662565],
       [0.98413044, 0.01586956],
       [0.76004681, 0.23995319],
       [0.98998675, 0.01001325],
       [0.98483541, 0.01516459],
       [0.98113619, 0.01886381]])
```

```
# store the probabilities in dataframe
y_pred_prob = pd.DataFrame(data=y_pred_prob, columns=['Prob of - No biopsy (0)', 'Prob of - Biopsy (1)'])
y_pred_prob
```

	Prob of - No biopsy (0)	Prob of - Biopsy (1)
0	0.931818	0.068182
1	0.986525	0.013475
2	0.991302	0.008698
3	0.987099	0.012901
4	0.973374	0.026626
5	0.984130	0.015870
6	0.760047	0.239953
7	0.989987	0.010013
8	0.984835	0.015165
9	0.981136	0.018864

Next steps: View recommended plots

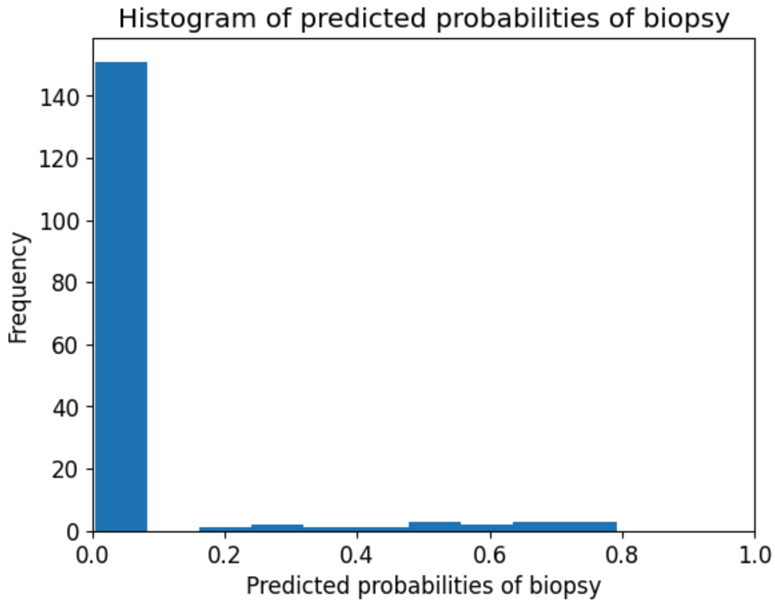
```
# print the first 10 predicted probabilities for class 1 - Probability of biopsy
logreg.predict_proba(X_test)[0:10, 1]
```

```
array([0.0681819 , 0.01347467, 0.00869753, 0.01290119, 0.02662565,
       0.01586956, 0.23995319, 0.01001325, 0.01516459, 0.01886381])
```

```
y_pred1 = logreg.predict_proba(X_test)[:,1]
```

```
# plot histogram of predicted probabilities
# adjust the font size
plt.rcParams['font.size'] = 12

# plot histogram with 10 bins
plt.hist(y_pred1, bins=10)

# set the title of predicted probabilities
plt.title('Histogram of predicted probabilities of biopsy')

# set the x-axis limit
plt.xlim(0,1)

# set the title
plt.xlabel('Predicted probabilities of biopsy')
plt.ylabel('Frequency')
```

```
Text(0, 0.5, 'Frequency')
```



Observations:

- The histogram is highly positive skewed
- The first column tells us that there are approximately 160 observation with probability between 0.0 and 0.1
- The small number of observation predict the possibility of undergoing biopsy.
- Majority of observation predicts the possibility of not undergoing biopsy.

Lower the treshold

```
from sklearn.preprocessing import binarize

for i in range(1,5):
    cm1=0
    y_pred1 = logreg.predict_proba(X_test)[:,1]
    y_pred1= y_pred1.reshape(-1,1)
    y_pred2= binarize(y_pred1, threshold=i/10)
    y_pred2 = np.where(y_pred2 == 1, 1, 0)
    cm1 = confusion_matrix(y_test, y_pred2)
    print ('With',i/10, 'threshold the Confusion Matrix is ','\n\n', cm1, '\n\n',
           'with', cm1[0,0]+cm1[1,1], 'correct predictions, ', '\n\n',
           cm1[0,1], 'Type I errors( False Positives), ','\n\n',
           cm1[1,0], 'Type II errors( False Negatives), ','\n\n',
           'Accuracy score: ', (accuracy_score(y_test, y_pred2)), '\n\n',
           'Sensitivity: ',cm1[1,1]/(float(cm1[1,1]+cm1[1,0])), '\n\n',
           'Specificity: ',cm1[0,0]/(float(cm1[0,0]+cm1[0,1])), '\n\n',
           '====================================================', '\n\n')
```

```
With 0.1 threshold the Confusion Matrix is

 [[149   8]
 [  2   8]]

 with 157 correct predictions,

 8 Type I errors( False Positives),

 2 Type II errors( False Negatives),

 Accuracy score:  0.9401197604790419

 Sensitivity:  0.8

 Specificity:  0.9490445859872612

 ====================================================


With 0.2 threshold the Confusion Matrix is

 [[149   8]
 [  2   8]]
```

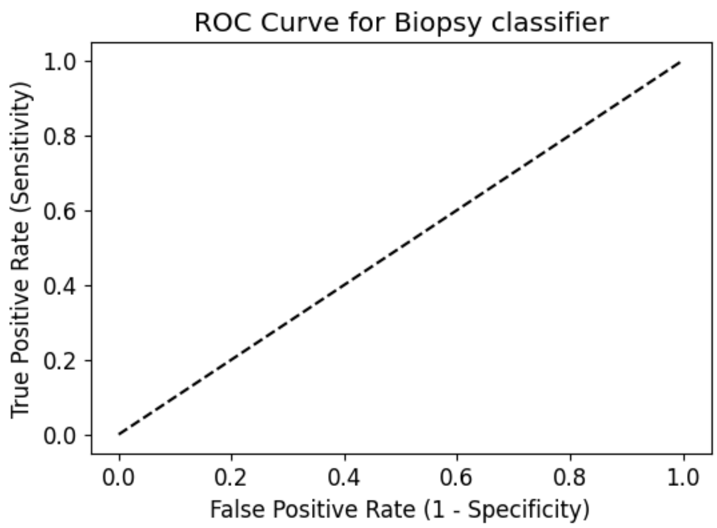
```
with 157 correct predictions,

8 Type I errors( False Positives),

2 Type II errors( False Negatives),

Accuracy score:  0.9401197604790419

Sensitivity:  0.8

Specificity:  0.9490445859872612

=====================================================


With 0.3 threshold the Confusion Matrix is

[[150   7]
 [  3   7]]

with 157 correct predictions,

7 Type I errors( False Positives),

3 Type II errors( False Negatives),

Accuracy score:  0.9401197604790419

Sensitivity:  0.7

Specificity:  0.9554140127388535

=====================================================
```

Comments

ROC - AUC

```
from sklearn.metrics import roc_curve

fpr, tpr, thresholds = roc_curve(y_test, y_pred1, pos_label = 'Yes')

plt.figure(figsize=(6,4))

plt.plot(fpr,tpr, linewidth=2)
plt.plot([0,1], [0,1], 'k--')
plt.rcParams['font.size'] =12
plt.title('ROC Curve for Biopsy classifier')
plt.xlabel('False Positive Rate (1 - Specificity)')
plt.ylabel('True Positive Rate (Sensitivity)')
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_ranking.py:1029: UndefinedMe
  warnings.warn(
Text(0, 0.5, 'True Positive Rate (Sensitivity)')
```



```
# compute ROC AUC
from sklearn.metrics import roc_auc_score

ROC_AUC = roc_auc_score(y_test, y_pred1)
print('ROC AUC : {:.4f}'.format(ROC_AUC))
```

```
ROC AUC : 0.9350
```

```
# calculate cross-validated ROC AUC
from sklearn.model_selection import cross_val_score

Cross_validated_ROC_AUC = cross_val_score(logreg, X_train, y_train, cv=5, scoring='roc_auc').mean()
print('Cross validated ROC AUC : {:.4f}'.format(Cross_validated_ROC_AUC))
```

```
Cross validated ROC AUC : 0.9405
```

k-Fold Cross Validation

```
# Applying 5-Fold Cross Validation
from sklearn.model_selection import cross_val_score

scores = cross_val_score(logreg, X_train, y_train, cv = 5, scoring='accuracy')
print('Cross-validation scores:{}'.format(scores))
```

```
Cross-validation scores:[0.96268657 0.96268657 0.94029851 0.96240602 0.94736842]
```

```
# compute Average cross-validation score
print('Average cross-validation score: {:.4f}'. format(scores.mean()))
```

```
Average cross-validation score: 0.9551
```

Hyperparameter Optimization using GridSearch CV

```
# Hyperparameter Optimization using GridSearchCV
from sklearn.model_selection import GridSearchCV

parameters = [{'penalty': ['l1', 'l2']},
              {'C':[1, 10, 100, 100]}]

grid_search = GridSearchCV(estimator = logreg,
                           param_grid = parameters,
                           scoring = 'accuracy',
                           cv = 5,
                           verbose=0)

grid_search.fit(X_train, y_train)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_validation.py:378: FitFailedWarning:
10 fits failed out of a total of 30.
The score on these train-test partitions for these parameters will be set to nan.
If these failures are not expected, you can try to debug them by setting error_score='raise'.

Below are more details about the failures:
--------------------------------------------------------------------------------
5 fits failed with the following error:
Traceback (most recent call last):
  File "/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_validation.py", line 686, in _fit_and_score
```

```
# examine the best model
# best score achieved during the GridSearchCV
print('GridSearch CV best score : {:.4f}\n\n'.format(grid_search.best_score_))

# print parameters that give the best results
print('Parameters that give the best results :', '\n\n', (grid_search.best_params_))

# print estimator that was chosen by the GridSearch
print('\n\nEstimator that was chosen by the search :', '\n\n', (grid_search.best_estimator_))
```

```
GridSearch CV best score : 0.9581


Parameters that give the best results :

 {'C': 10}


Estimator that was chosen by the search :

 LogisticRegression(C=10, random_state=0, solver='liblinear')
/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_search.py:952: UserWarning: One or more of the test scores are non
```

```
# calculate Gridsearch CV score on test set
print('Gridsearch CV score on test set: {0:0.4f}'.format(grid_search.score(X_test, y_test)))
```

```
Gridsearch CV score on test set: 0.9281
```

Results and Conclusion

1. The logistic regression model accuracy score is 0.9461. So, the model does a very good job in predicting whether a patient having the aforementioned illness would undergo biopsy.
2. Small number of observations predict that patients would undergo biopsy. Majority of observations predict that patients would not undergo biopsy.
3. The model shows no signs of overfitting.
4. ROC AUC of our model approaches towards 1. So, we can conclude that our classifier does a good job in predicting whether a patient having the aforementioned illness would undergo biopsy.
5. Our original model test accuracy is 0.9461 while GridSearch CV accuracy is 0.9281. We can see that GridSearch CV decreased the performance for this particular model.