Classification using Logistic Regression

Marquez, Keith Leigh Zhen R.

Objective(s):

This activity aims to demonstrate how to apply simple linear regression analysis to solve regression problem

```
pip install ucimlrepo
```

```
Requirement already satisfied: ucimlrepo in /usr/local/lib/python3.10/dist-packages (0.0.6)
```

```
from ucimlrepo import fetch_ucirepo

# fetch dataset
cervical_cancer_risk_factors = fetch_ucirepo(id=383)

# data (as pandas dataframes)
X = cervical_cancer_risk_factors.data.features
y = cervical_cancer_risk_factors.data.targets

# metadata
print(cervical_cancer_risk_factors.metadata)

# variable information
print(cervical_cancer_risk_factors.variables)
```

```
     14        STDs:cervical condylomatosis  Feature  Continuous        None
     15         STDs:vaginal condylomatosis  Feature  Continuous        None
     16  STDs:vulvo-perineal condylomatosis  Feature  Continuous        None
     17                       STDs:syphilis  Feature  Continuous        None
     18    STDs:pelvic inflammatory disease  Feature  Continuous        None
     19                 STDs:genital herpes  Feature  Continuous        None
     20          STDs:molluscum contagiosum  Feature  Continuous        None
     21                            STDs:AIDS  Feature  Continuous        None
     22                             STDs:HIV  Feature  Continuous        None
     23                     STDs:Hepatitis B  Feature  Continuous        None
     24                             STDs:HPV  Feature  Continuous        None
     25           STDs: Number of diagnosis  Feature     Integer        None
     26    STDs: Time since first diagnosis  Feature  Continuous        None
     27     STDs: Time since last diagnosis  Feature  Continuous        None
     28                            Dx:Cancer  Feature     Integer        None
     29                               Dx:CIN  Feature     Integer        None
     30                               Dx:HPV  Feature     Integer        None
     31                                   Dx  Feature     Integer        None
     32                           Hinselmann  Feature     Integer        None
     33                             Schiller  Feature     Integer        None
     34                             Citology  Feature     Integer        None
     35                               Biopsy  Feature     Integer        None

        description units missing_values
     0         None  None             no
     1         None  None            yes
     2         None  None            yes
     3         None  None            yes
     4         None  None            yes
     5         None  None            yes
     6         None  None            yes
     7         None  None            yes
```

```
25        None  None            no
26        None  None           yes
27        None  None           yes
28        None  None            no
29        None  None            no
30        None  None            no
31        None  None            no
32        None  None            no
```

Data Wrangling

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
cc = pd.concat([X,y], axis=1)
cc
```

	Age	Number of sexual partners	First sexual intercourse	Num of pregnancies	Smokes	Smokes (years)	Smokes (packs/year)	Hormonal Contraceptives	Hormonal Contraceptives (years)	IUD	...	STDs: Time since first diagnosis	S s diagn
0	18	4.0	15.0	1.0	0.0	0.0	0.0	0.0	0.00	0.0	...	NaN	
1	15	1.0	14.0	1.0	0.0	0.0	0.0	0.0	0.00	0.0	...	NaN	
2	34	1.0	NaN	1.0	0.0	0.0	0.0	0.0	0.00	0.0	...	NaN	
3	52	5.0	16.0	4.0	1.0	37.0	37.0	1.0	3.00	0.0	...	NaN	
4	46	3.0	21.0	4.0	0.0	0.0	0.0	1.0	15.00	0.0	...	NaN	
...	...	...	...	...	...	...	...	...	...	...	...	...	
853	34	3.0	18.0	0.0	0.0	0.0	0.0	0.0	0.00	0.0	...	NaN	
854	32	2.0	19.0	1.0	0.0	0.0	0.0	1.0	8.00	0.0	...	NaN	
855	25	2.0	17.0	0.0	0.0	0.0	0.0	1.0	0.08	0.0	...	NaN	
856	33	2.0	24.0	2.0	0.0	0.0	0.0	1.0	0.08	0.0	...	NaN	
857	29	2.0	20.0	1.0	0.0	0.0	0.0	1.0	0.50	0.0	...	NaN	

858 rows × 36 columns

```
cc.dtypes
```

```
Age                                 int64
Number of sexual partners         float64
First sexual intercourse          float64
Num of pregnancies                float64
Smokes                            float64
Smokes (years)                    float64
Smokes (packs/year)               float64
Hormonal Contraceptives           float64
Hormonal Contraceptives (years)   float64
IUD                               float64
IUD (years)                       float64
STDs                              float64
STDs (number)                     float64
STDs:condylomatosis               float64
STDs:cervical condylomatosis      float64
STDs:vaginal condylomatosis       float64
STDs:vulvo-perineal condylomatosis float64
STDs:syphilis                     float64
STDs:pelvic inflammatory disease  float64
STDs:genital herpes               float64
STDs:molluscum contagiosum        float64
STDs:AIDS                         float64
STDs:HIV                          float64
STDs:Hepatitis B                  float64
STDs:HPV                          float64
STDs: Number of diagnosis           int64
STDs: Time since first diagnosis  float64
STDs: Time since last diagnosis   float64
```

```
Dx:Cancer                           int64
Dx:CIN                              int64
Dx:HPV                              int64
Dx                                  int64
Hinselmann                          int64
Schiller                            int64
Citology                            int64
Biopsy                              int64
dtype: object
```

```
cc_null=cc.isnull().sum()
cc_null
```

```
Age                                  0
Number of sexual partners           26
First sexual intercourse             7
Num of pregnancies                  56
Smokes                              13
Smokes (years)                      13
Smokes (packs/year)                 13
Hormonal Contraceptives            108
Hormonal Contraceptives (years)    108
IUD                                117
IUD (years)                        117
STDs                               105
STDs (number)                      105
STDs:condylomatosis                105
STDs:cervical condylomatosis       105
STDs:vaginal condylomatosis        105
STDs:vulvo-perineal condylomatosis  105
STDs:syphilis                      105
STDs:pelvic inflammatory disease   105
STDs:genital herpes                105
STDs:molluscum contagiosum         105
STDs:AIDS                          105
STDs:HIV                           105
STDs:Hepatitis B                   105
STDs:HPV                           105
STDs: Number of diagnosis            0
STDs: Time since first diagnosis   787
STDs: Time since last diagnosis    787
Dx:Cancer                            0
Dx:CIN                               0
Dx:HPV                               0
Dx                                   0
Hinselmann                           0
Schiller                             0
Citology                             0
Biopsy                               0
dtype: int64
```

```
for column in cc.columns:
    cc[column] = cc[column].fillna(cc[column].mode()[0])
cc.isnull().sum()
```

```
Age                                 0
Number of sexual partners           0
First sexual intercourse            0
Num of pregnancies                  0
Smokes                              0
Smokes (years)                      0
Smokes (packs/year)                 0
Hormonal Contraceptives             0
Hormonal Contraceptives (years)     0
IUD                                 0
IUD (years)                         0
STDs                                0
STDs (number)                       0
STDs:condylomatosis                 0
STDs:cervical condylomatosis        0
STDs:vaginal condylomatosis         0
STDs:vulvo-perineal condylomatosis  0
STDs:syphilis                       0
STDs:pelvic inflammatory disease    0
STDs:genital herpes                 0
STDs:molluscum contagiosum          0
STDs:AIDS                           0
STDs:HIV                            0
STDs:Hepatitis B                    0
STDs:HPV                            0
STDs: Number of diagnosis           0
```

```
STDs: Time since first diagnosis      0
STDs: Time since last diagnosis       0
Dx:Cancer                             0
Dx:CIN                                0
Dx:HPV                                0
Dx                                    0
Hinselmann                            0
Schiller                              0
Citology                              0
Biopsy                                0
dtype: int64
```

cc

	Age	Number of sexual partners	First sexual intercourse	Num of pregnancies	Smokes	Smokes (years)	Smokes (packs/year)	Hormonal Contraceptives	Hormonal Contraceptives (years)	IUD	...	STDs: Time since first diagnosis	S s diagn
0	18	4.0	15.0	1.0	0.0	0.0	0.0	0.0	0.00	0.0	...	1.0	
1	15	1.0	14.0	1.0	0.0	0.0	0.0	0.0	0.00	0.0	...	1.0	
2	34	1.0	15.0	1.0	0.0	0.0	0.0	0.0	0.00	0.0	...	1.0	
3	52	5.0	16.0	4.0	1.0	37.0	37.0	1.0	3.00	0.0	...	1.0	
4	46	3.0	21.0	4.0	0.0	0.0	0.0	1.0	15.00	0.0	...	1.0	
...	...	...	...	...	...	...	...	...	...	...	...	...	
853	34	3.0	18.0	0.0	0.0	0.0	0.0	0.0	0.00	0.0	...	1.0	
854	32	2.0	19.0	1.0	0.0	0.0	0.0	1.0	8.00	0.0	...	1.0	
855	25	2.0	17.0	0.0	0.0	0.0	0.0	1.0	0.08	0.0	...	1.0	
856	33	2.0	24.0	2.0	0.0	0.0	0.0	1.0	0.08	0.0	...	1.0	
857	29	2.0	20.0	1.0	0.0	0.0	0.0	1.0	0.50	0.0	...	1.0	

858 rows × 36 columns

Remove duplicates

```
# Check for duplicates
duplicate_rows = cc.duplicated()

# Count the number of duplicate rows
num_duplicates = duplicate_rows.sum()
print("Number of duplicate rows:", num_duplicates)
```

```
Number of duplicate rows: 28
```

```
cc.drop_duplicates(inplace=True)
cc
```

	Age	Number of sexual partners	First sexual intercourse	Num of pregnancies	Smokes	Smokes (years)	Smokes (packs/year)	Hormonal Contraceptives	Hormonal Contraceptives (years)	IUD	...	STDs: Time since first diagnosis	S s diagn
0	18	4.0	15.0	1.0	0.0	0.0	0.0	0.0	0.00	0.0	...	1.0	
1	15	1.0	14.0	1.0	0.0	0.0	0.0	0.0	0.00	0.0	...	1.0	
2	34	1.0	15.0	1.0	0.0	0.0	0.0	0.0	0.00	0.0	...	1.0	
3	52	5.0	16.0	4.0	1.0	37.0	37.0	1.0	3.00	0.0	...	1.0	
4	46	3.0	21.0	4.0	0.0	0.0	0.0	1.0	15.00	0.0	...	1.0	
...	...	...	...	...	...	...	...	...	...	...	...	...	
853	34	3.0	18.0	0.0	0.0	0.0	0.0	0.0	0.00	0.0	...	1.0	
854	32	2.0	19.0	1.0	0.0	0.0	0.0	1.0	8.00	0.0	...	1.0	
855	25	2.0	17.0	0.0	0.0	0.0	0.0	1.0	0.08	0.0	...	1.0	
856	33	2.0	24.0	2.0	0.0	0.0	0.0	1.0	0.08	0.0	...	1.0	
857	29	2.0	20.0	1.0	0.0	0.0	0.0	1.0	0.50	0.0	...	1.0	

830 rows × 36 columns

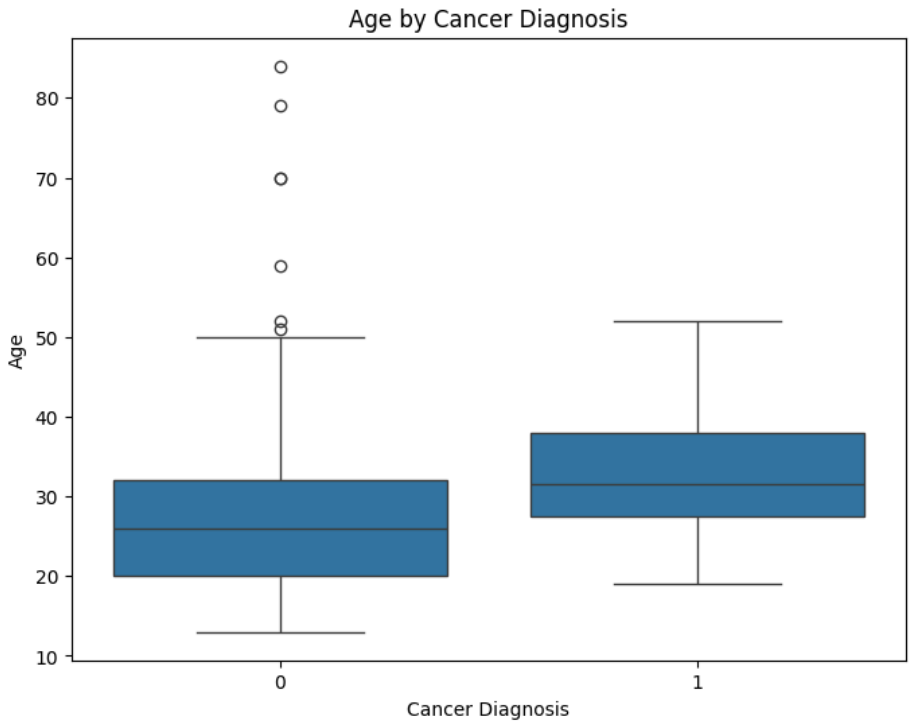
```
cc.describe()
```

	Age	Number of sexual partners	First sexual intercourse	Num of pregnancies	Smokes	Smokes (years)	Smokes (packs/year)	Hormonal Contraceptives	Hormonal Contraceptives (years)	IUD
count	830.000000	830.000000	830.000000	830.000000	830.000000	830.000000	830.000000	830.000000	830.000000	830.000000
mean	27.075904	2.540964	17.010843	2.222892	0.148193	1.241765	0.461333	0.696386	2.031005	0.100000
std	8.479609	1.656987	2.817510	1.445331	0.355505	4.122522	2.245822	0.460095	3.639632	0.300181
min	13.000000	1.000000	10.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	21.000000	2.000000	15.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
50%	26.000000	2.000000	17.000000	2.000000	0.000000	0.000000	0.000000	1.000000	0.250000	0.000000
75%	32.000000	3.000000	18.000000	3.000000	0.000000	0.000000	0.000000	1.000000	3.000000	0.000000
max	84.000000	28.000000	32.000000	11.000000	1.000000	37.000000	37.000000	1.000000	30.000000	1.000000

8 rows × 36 columns

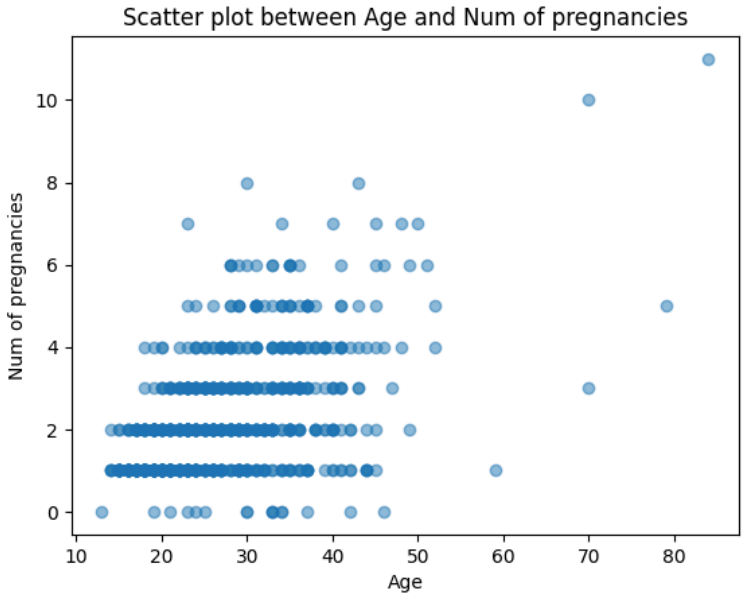
EDA

```
plt.figure(figsize=(8, 6))
sns.boxplot(x='Dx:Cancer', y='Age', data=cc)
plt.xlabel('Cancer Diagnosis')
plt.ylabel('Age')
plt.title('Age by Cancer Diagnosis')
plt.show()
```



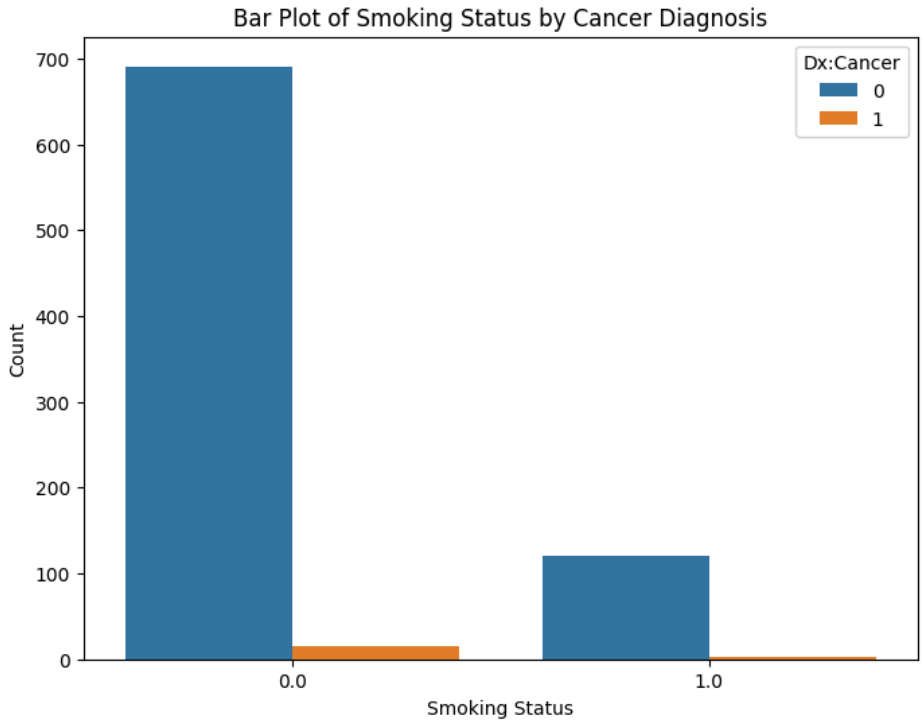
This compares the age distribution of individuals with and without a cancer diagnosis. Both groups have similar median ages around 40, but the non-diagnosed group has a wider range of ages due to outliers.

```
plt.scatter(cc['Age'], cc['Num of pregnancies'], alpha=0.5)
plt.xlabel('Age')
plt.ylabel('Num of pregnancies')
plt.title('Scatter plot between Age and Num of pregnancies')
plt.show()
```



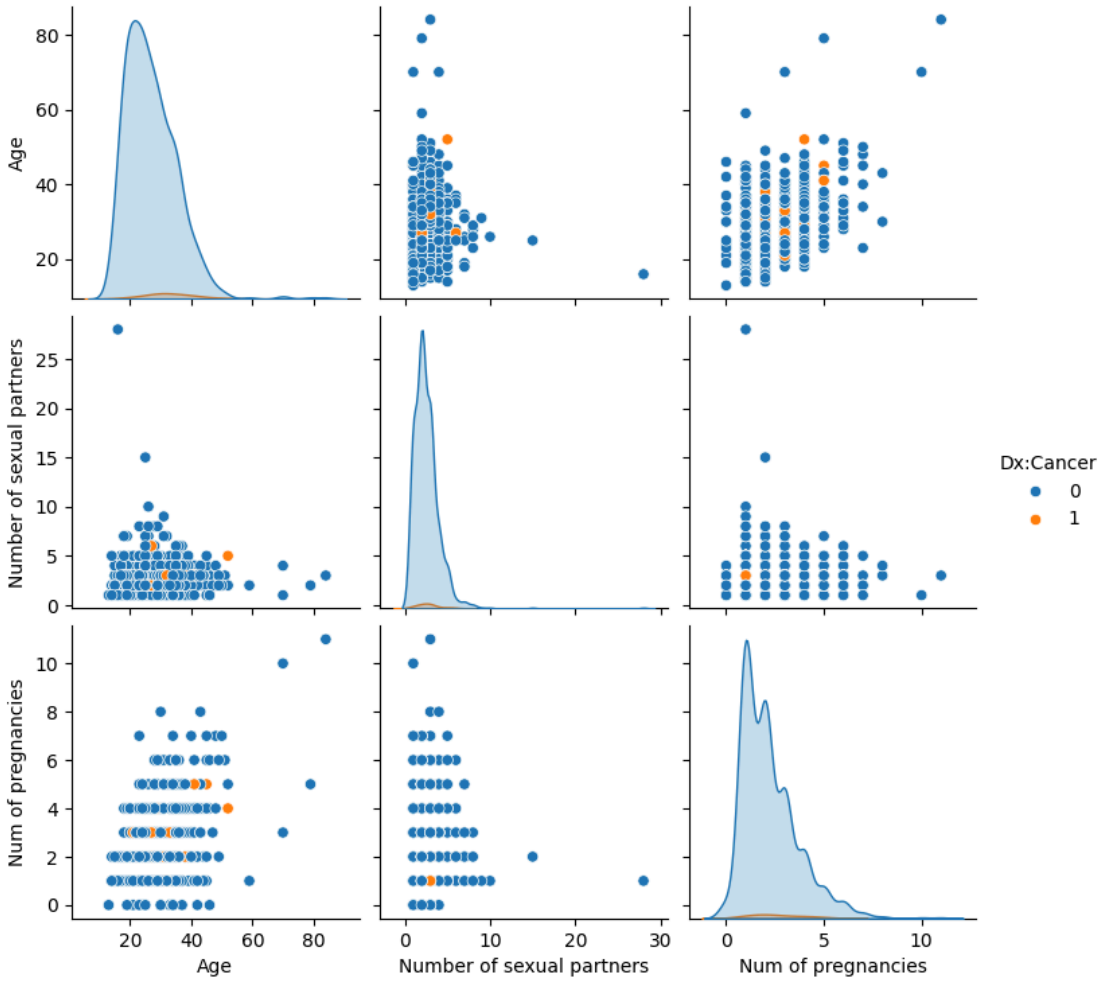
Most pregnancies occur between ages 20 and 50, with fewer occurrences in individuals below 20 or above 50. Age significantly influences pregnancy likelihood, with peak reproductive years around 20 to 50.

```
plt.figure(figsize=(8, 6))
sns.countplot(x='Smokes', hue='Dx:Cancer', data=cc)
plt.xlabel('Smoking Status')
plt.ylabel('Count')
plt.title('Bar Plot of Smoking Status by Cancer Diagnosis')
plt.show()
```



Non-smokers (smoking status 0.0) are less likely to be diagnosed with cancer, while smokers (smoking status 1.0) have a higher proportion of cancer diagnoses. The data suggests a potential link between smoking and cancer risk.

```
# Pairplot for selected numerical columns
sns.pairplot(cc[['Age', 'Number of sexual partners', 'Num of pregnancies', 'Dx:Cancer']], hue='Dx:Cancer')
plt.show()
```



This indicates that most individuals are young adults with fewer sexual partners and pregnancies. A small proportion has been diagnosed with cancer, but there is no clear pattern suggesting a direct association between these factors and cancer risk from the graphs alone.

Simple Linear Regression and Demonstration of Logistic Regression

Split data into separate training and test set

```
X = cc.drop('Biopsy', axis=1)
y = cc['Biopsy']
```

```
from sklearn.model_selection import train_test_split

# Split x and y into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Check the shape of train and test sets
print("Shape of X_train:", X_train.shape)
print("Shape of X_test:", X_test.shape)
```

```
Shape of X_train: (664, 35)
Shape of X_test: (166, 35)
```

```
X_train.dtypes
```

```
Age                                   int64
Number of sexual partners           float64
First sexual intercourse            float64
Num of pregnancies                  float64
Smokes                              float64
```

```
Smokes (years)                      float64
Smokes (packs/year)                 float64
Hormonal Contraceptives             float64
Hormonal Contraceptives (years)     float64
IUD                                 float64
IUD (years)                         float64
STDs                                float64
STDs (number)                       float64
STDs:condylomatosis                 float64
STDs:cervical condylomatosis        float64
STDs:vaginal condylomatosis         float64
STDs:vulvo-perineal condylomatosis  float64
STDs:syphilis                       float64
STDs:pelvic inflammatory disease    float64
STDs:genital herpes                 float64
STDs:molluscum contagiosum          float64
STDs:AIDS                           float64
STDs:HIV                            float64
STDs:Hepatitis B                    float64
STDs:HPV                            float64
STDs: Number of diagnosis             int64
STDs: Time since first diagnosis    float64
STDs: Time since last diagnosis     float64
Dx:Cancer                             int64
Dx:CIN                                int64
Dx:HPV                                int64
Dx                                    int64
Hinselmann                            int64
Schiller                              int64
Citology                              int64
dtype: object
```

```
#display numerical variables
numerical = [col for col in X_train.columns if X_train[col].dtypes != "0"]
numerical
```

```
['Age',
 'Number of sexual partners',
 'First sexual intercourse',
 'Num of pregnancies',
 'Smokes',
 'Smokes (years)',
 'Smokes (packs/year)',
 'Hormonal Contraceptives',
 'Hormonal Contraceptives (years)',
 'IUD',
 'IUD (years)',
 'STDs',
 'STDs (number)',
 'STDs:condylomatosis',
 'STDs:cervical condylomatosis',
 'STDs:vaginal condylomatosis',
 'STDs:vulvo-perineal condylomatosis',
 'STDs:syphilis',
 'STDs:pelvic inflammatory disease',
 'STDs:genital herpes',
 'STDs:molluscum contagiosum',
 'STDs:AIDS',
 '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']
```

Engineering outliears in numerical variables

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

numerical_columns = [
    'Age', 'Number of sexual partners', 'First sexual intercourse', 'Num of pregnancies',
    'Smokes', 'Smokes (years)', 'Smokes (packs/year)', 'Hormonal Contraceptives',
    'Hormonal Contraceptives (years)', 'IUD', 'IUD (years)', 'STDs', 'STDs (number)',
    'STDs:condylomatosis', 'STDs:cervical condylomatosis', 'STDs:vaginal condylomatosis',
    'STDs:vulvo-perineal condylomatosis', 'STDs:syphilis', 'STDs:pelvic inflammatory disease',
    'STDs:genital herpes', 'STDs:molluscum contagiosum', 'STDs:AIDS', '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'
]


# Apply max_value function to each numerical column
for df in [X_train, X_test]:
    df['Age'] = max_value(df, 'Age', 100)  # Assuming 100 is the maximum allowable age
    df['Number of sexual partners'] = max_value(df, 'Number of sexual partners', 10)  # Example threshold
    # Repeat this process for other numerical columns
```

Maximum values after transformation:

```
# Check the maximum values after transformation
print("X_train:", X_train[numerical_columns].max())
```

```
X_train: Age                                 70.0
Number of sexual partners            10.0
First sexual intercourse             32.0
Num of pregnancies                    8.0
Smokes                                1.0
Smokes (years)                       37.0
Smokes (packs/year)                  37.0
Hormonal Contraceptives               1.0
Hormonal Contraceptives (years)      30.0
IUD                                   1.0
IUD (years)                          19.0
STDs                                  1.0
STDs (number)                         4.0
STDs:condylomatosis                   1.0
STDs:cervical condylomatosis          0.0
STDs:vaginal condylomatosis           1.0
STDs:vulvo-perineal condylomatosis    1.0
STDs:syphilis                         1.0
STDs:pelvic inflammatory disease      1.0
STDs:genital herpes                   1.0
STDs:molluscum contagiosum            1.0
STDs:AIDS                             0.0
STDs:HIV                              1.0
STDs:Hepatitis B                      1.0
STDs:HPV                              1.0
STDs: Number of diagnosis             3.0
STDs: Time since first diagnosis     22.0
STDs: Time since last diagnosis      22.0
Dx:Cancer                             1.0
Dx:CIN                                1.0
Dx:HPV                                1.0
Dx                                    1.0
Hinselmann                            1.0
Schiller                              1.0
Citology                              1.0
dtype: float64
```

```
print("X_test:", X_test[numerical_columns].max())
```

```
X_test: Age                                  84.0
Number of sexual partners            10.0
First sexual intercourse             28.0
Num of pregnancies                   11.0
Smokes                                1.0
Smokes (years)                       24.0
Smokes (packs/year)                  21.0
Hormonal Contraceptives               1.0
Hormonal Contraceptives (years)      22.0
IUD                                   1.0
IUD (years)                          10.0
STDs                                  1.0
STDs (number)                         2.0
```

```
STDs:condylomatosis                 1.0
STDs:cervical condylomatosis        0.0
STDs:vaginal condylomatosis         0.0
STDs:vulvo-perineal condylomatosis  1.0
STDs:syphilis                       1.0
STDs:pelvic inflammatory disease    0.0
STDs:genital herpes                 0.0
STDs:molluscum contagiosum          0.0
STDs:AIDS                           0.0
STDs:HIV                            1.0
STDs:Hepatitis B                    0.0
STDs:HPV                            0.0
STDs: Number of diagnosis           1.0
STDs: Time since first diagnosis   12.0
STDs: Time since last diagnosis    12.0
Dx:Cancer                           1.0
Dx:CIN                              1.0
Dx:HPV                              1.0
Dx                                  1.0
Hinselmann                          1.0
Schiller                            1.0
Citology                            1.0
dtype: float64
```

```
X_train[numerical_columns].describe()
```

	Smokes (packs/year)	Hormonal Contraceptives	Hormonal Contraceptives (years)	IUD	...	STDs: Number of diagnosis	STDs: Time since first diagnosis	STDs: Time since last diagnosis	Dx:Cancer	Dx:CIN	Dx:HPV	
	664.000000	664.000000	664.000000	664.000000	...	664.000000	664.000000	664.000000	664.000000	664.000000	664.000000	664
	0.475850	0.691265	1.916573	0.096386	...	0.096386	1.487952	1.453313	0.019578	0.012048	0.022590	0
	2.332357	0.462320	3.499791	0.295342	...	0.319859	2.414731	2.312192	0.138650	0.109183	0.148706	0
	0.000000	0.000000	0.000000	0.000000	...	0.000000	1.000000	1.000000	0.000000	0.000000	0.000000	0
	0.000000	0.000000	0.000000	0.000000	...	0.000000	1.000000	1.000000	0.000000	0.000000	0.000000	0
	0.000000	1.000000	0.250000	0.000000	...	0.000000	1.000000	1.000000	0.000000	0.000000	0.000000	0
	0.000000	1.000000	2.000000	0.000000	...	0.000000	1.000000	1.000000	0.000000	0.000000	0.000000	0
	37.000000	1.000000	30.000000	1.000000	...	3.000000	22.000000	22.000000	1.000000	1.000000	1.000000	1

Feature Scaling

```
X_train.describe()
```

	Age	Number of sexual partners	First sexual intercourse	Num of pregnancies	Smokes	Smokes (years)	Smokes (packs/year)	Hormonal Contraceptives	Hormonal Contraceptives (years)	IUD
count	664.000000	664.000000	664.000000	664.00000	664.000000	664.000000	664.000000	664.000000	664.000000	664.000000
mean	27.060241	2.484940	17.025602	2.25000	0.146084	1.258027	0.475850	0.691265	1.916573	0.096386
std	7.905894	1.318078	2.782884	1.39299	0.353457	4.218211	2.332357	0.462320	3.499791	0.295342
min	14.000000	1.000000	10.000000	0.00000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	21.000000	2.000000	15.000000	1.00000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
50%	26.000000	2.000000	17.000000	2.00000	0.000000	0.000000	0.000000	1.000000	0.250000	0.000000
75%	32.000000	3.000000	18.000000	3.00000	0.000000	0.000000	0.000000	1.000000	2.000000	0.000000
max	70.000000	10.000000	32.000000	8.00000	1.000000	37.000000	37.000000	1.000000	30.000000	1.000000

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.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 (packs/year)	Hormonal Contraceptives	Hormonal Contraceptives (years)	IUD
count	664.000000	664.000000	664.000000	664.000000	664.000000	664.000000	664.000000	664.000000	664.000000	664.000000
mean	0.233219	0.164993	0.319346	0.281250	0.146084	0.034001	0.012861	0.691265	0.063886	0.096386
std	0.141177	0.146453	0.126495	0.174124	0.353457	0.114006	0.063037	0.462320	0.116660	0.295342
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.125000	0.111111	0.227273	0.125000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
50%	0.214286	0.111111	0.318182	0.250000	0.000000	0.000000	0.000000	1.000000	0.008333	0.000000
75%	0.321429	0.222222	0.363636	0.375000	0.000000	0.000000	0.000000	1.000000	0.066667	0.000000
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

8 rows × 35 columns

Model Training

```
from sklearn.linear_model import LogisticRegression

# Instantiate the logistic regression model
logreg = LogisticRegression(solver='liblinear', random_state=42)  # Adjust solver and random_state as needed

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

```
LogisticRegression(random_state=42, 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, 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, 1, 0, 0, 0, 0, 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, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 1, 0, 0, 1, 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, 1, 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])
```

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

```
array([0.97167814, 0.98621904, 0.97548734, 0.97972786, 0.98367934,
       0.97696824, 0.98450894, 0.98362969, 0.98431934, 0.97699586,
```

```
       0.98030213, 0.98187561, 0.97602653, 0.40791458, 0.98165476,
       0.95004821, 0.97979568, 0.97847942, 0.97929069, 0.93925603,
       0.98250452, 0.9709148 , 0.98038387, 0.69087378, 0.98028688,
       0.97448159, 0.9803807 , 0.98292581, 0.97990914, 0.98114977,
       0.97404099, 0.98294775, 0.96048621, 0.98351494, 0.98593851,
       0.97606419, 0.9565455 , 0.98057819, 0.97549488, 0.98791546,
       0.98474134, 0.97960026, 0.35920079, 0.9704203 , 0.94747769,
       0.97948509, 0.97877902, 0.40118625, 0.96721231, 0.98209548,
       0.98650188, 0.97285332, 0.98067339, 0.97352784, 0.98302117,
       0.98577423, 0.96692052, 0.98365639, 0.98816839, 0.48103242,
       0.98057753, 0.28532274, 0.98055511, 0.98300654, 0.98142108,
       0.97883932, 0.97356948, 0.97790029, 0.98877925, 0.95339285,
       0.98452563, 0.98644564, 0.97396277, 0.98040971, 0.98585397,
       0.97956192, 0.2697327 , 0.97700375, 0.98761238, 0.98325658,
       0.98468998, 0.98058107, 0.9841359 , 0.97569208, 0.98068851,
       0.97578687, 0.9608319 , 0.98490733, 0.64426007, 0.98393871,
       0.64400598, 0.98369889, 0.31542777, 0.97849007, 0.96344689,
       0.43346647, 0.98230825, 0.97971349, 0.98054514, 0.98185996,
       0.98005718, 0.9780225 , 0.87323158, 0.98384355, 0.98372832,
       0.97934297, 0.96756457, 0.25295199, 0.97736216, 0.97063763,
       0.98701769, 0.9792027 , 0.9850321 , 0.98460897, 0.73267532,
       0.9839528 , 0.98210336, 0.34175938, 0.96998096, 0.97882308,
       0.98211513, 0.99030635, 0.98206261, 0.98095815, 0.98053311,
       0.55083532, 0.97598307, 0.97103627, 0.17077463, 0.98301245,
       0.99086979, 0.97994397, 0.98990966, 0.98011905, 0.97113264,
       0.95499399, 0.97033208, 0.98637817, 0.97965374, 0.23696323,
       0.97727537, 0.9802864 , 0.98666697, 0.98364211, 0.4234108 ,
       0.96269302, 0.97685968, 0.9750471 , 0.9785283 , 0.98229493,
       0.9892733 , 0.95081616, 0.97757596, 0.98498087, 0.98062242,
       0.98132382, 0.97882584, 0.98487212, 0.97353172, 0.97581305,
       0.9792013 , 0.96751914, 0.98552693, 0.97980147, 0.96496642,
       0.97813663])
```

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

```
array([0.02832186, 0.01378096, 0.02451266, 0.02027214, 0.01632066,
       0.02303176, 0.01549106, 0.01637031, 0.01568066, 0.02300414,
       0.01969787, 0.01812439, 0.02397347, 0.59208542, 0.01834524,
       0.04995179, 0.02020432, 0.02152058, 0.02070931, 0.06074397,
       0.01749548, 0.0290852 , 0.01961613, 0.30912622, 0.01971312,
       0.02551841, 0.0196193 , 0.01707419, 0.02009086, 0.01885023,
       0.02595901, 0.01705225, 0.03951379, 0.01648506, 0.01406149,
       0.02393581, 0.0434545 , 0.01942181, 0.02450512, 0.01208454,
       0.01525866, 0.02039974, 0.64079921, 0.0295797 , 0.05252231,
       0.02051491, 0.02122098, 0.59881375, 0.03278769, 0.01790452,
       0.01349812, 0.02714668, 0.01932661, 0.02647216, 0.01697883,
       0.01422577, 0.03307948, 0.01634361, 0.01183161, 0.51896758,
       0.01942247, 0.71467726, 0.01944489, 0.01699346, 0.01857892,
       0.02116068, 0.02643052, 0.02209971, 0.01122075, 0.04660715,
       0.01547437, 0.01355436, 0.02603723, 0.01959029, 0.01414603,
       0.02043808, 0.7302673 , 0.02299625, 0.01238762, 0.01674342,
       0.01531002, 0.01941893, 0.0158641 , 0.02430792, 0.01931149,
       0.02421313, 0.0391681 , 0.01509267, 0.35573993, 0.01606129,
       0.35599402, 0.01630111, 0.68457223, 0.02150993, 0.03655311,
       0.56653353, 0.01769175, 0.02028651, 0.01945486, 0.01814004,
       0.01994282, 0.0219775 , 0.12676842, 0.01615645, 0.01627168,
       0.02065703, 0.03243543, 0.74704801, 0.02263784, 0.02936237,
       0.01298231, 0.0207973 , 0.0149679 , 0.01539103, 0.26732468,
       0.0160472 , 0.01789664, 0.65824062, 0.03001904, 0.02117692,
       0.01788487, 0.00969365, 0.01793739, 0.01904185, 0.01946689,
       0.44916468, 0.02401693, 0.02896373, 0.82922537, 0.01698755,
       0.00913021, 0.02005603, 0.01009034, 0.01988095, 0.02886736,
       0.04500601, 0.02966792, 0.01362183, 0.02034626, 0.76303677,
       0.02272463, 0.0197136 , 0.01333303, 0.01635789, 0.5765892 ,
       0.03730698, 0.02314032, 0.0249529 , 0.0214717 , 0.01770507,
       0.0107267 , 0.04918384, 0.02242404, 0.01501913, 0.01937758,
       0.01867618, 0.02117416, 0.01512788, 0.02646828, 0.02418695,
       0.0207987 , 0.03248086, 0.01447307, 0.02019853, 0.03503358,
       0.02186337])
```

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: 0.9639
```

Compare the train-set and test-set accuracy Now, I will compare the train-set and test-set accuracy to check for overfitting.

```
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, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0,
       0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 1, 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, 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, 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, 1, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 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, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 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, 0, 0, 0, 0, 0, 0, 1, 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, 0, 0, 0, 0, 0, 1, 1,
       0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 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, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 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, 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, 1, 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, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 1])
```

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

```
Training accuracy score: 0.9623
```

Check for overfitting and underfitting

```
#print the scores on training and test set
print('Training set score: {:.4f}'.format(logreg.score(X_train, y_train)))
print('Test set score: {:.4f}'.format(logreg.score(X_test, y_test)))
```

```
Training set score: 0.9623
Test set score: 0.9639
```

```
#fit the Logsitic Regression model with C-100
#instantiate the model
logreg100 = LogisticRegression (C=100, solver='liblinear', random_state=0)
#fit the model
logreg100.fit(X_train, y_train)
```

```
▾                    LogisticRegression
LogisticRegression(C=100, random_state=0, solver='liblinear')
```

```
#print the scores on training and test set
print('Training set score: {:.4f}'.format(logreg100.score(X_train, y_train)))
print('Test set score: {:.4f}'.format(logreg100.score(X_test, y_test)))
```

```
Training set score: 0.9654
Test set score: 0.9518
```

Compare model accuracy with null accuracy

```
#check class distribution in test set
y_test.value_counts()
```

```
Biopsy
0    155
1     11
Name: count, dtype: int64
```

```
#check null accuracy score
null_accuracy= (155/(155+11))
print('Null accuracy score: {0:0.4f}'. format(null_accuracy))
```

```
Null accuracy score: 0.9337
```

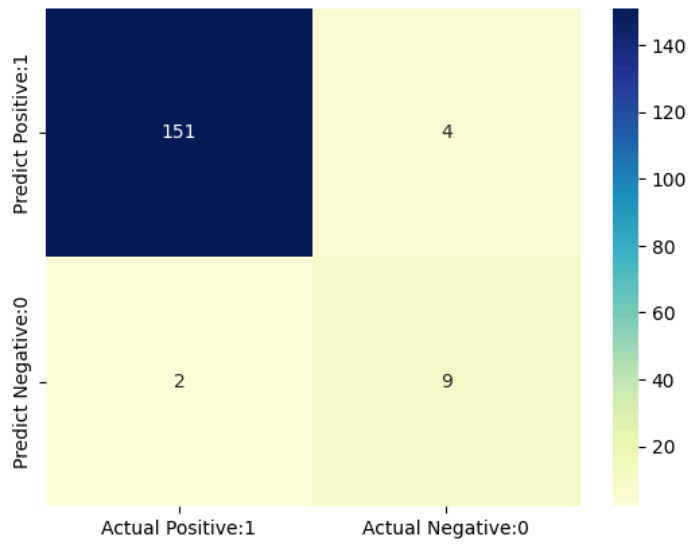
Confusion Matrix

```
#Print the Confusion Matrix and slice it into four pieces
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred_test)
print('Confusion matrix\n\n', cm)
print('\nTrue Positives (TP)=', cm[0,0])
print('\nTrue Negatives (TN)=', cm[1,1])
print('\nFalse Positives (FP)=', cm[0,1])
print('\nFalse Negatives (FN)=', cm[1,0])
```

```
Confusion matrix

 [[151   4]
 [  2   9]]

True Positives (TP)= 151

True Negatives (TN)= 9

False Positives (FP)= 4

False Negatives (FN)= 2
```

```
#visualize confusion matrix with seaborn heatmap
cm_matrix=pd.DataFrame(data=cm, columns=['Actual Positive:1', 'Actual Negative:0'],
                       index=['Predict Positive:1', 'Predict Negative:0'])
sns.heatmap(cm_matrix, annot=True, fmt='d', cmap='YlGnBu')
```

```
<Axes: >
```



The confusion matrix represents the performance of a classification model. It shows the number of true positives (correctly predicted positive instances), true negatives (correctly predicted negative instances), false positives (incorrectly predicted positive instances), and false negatives (incorrectly predicted negative instances). The model has high true positives, indicating good performance in identifying the positive class. However, there are more false positives than false negatives, suggesting potential bias. Further improvements could enhance precision and recall.

Classification Metrices

```
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred_test))
```

```
                  precision    recall  f1-score   support

               0       0.99      0.97      0.98       155
               1       0.69      0.82      0.75        11

        accuracy                           0.96       166
       macro avg       0.84      0.90      0.87       166
    weighted avg       0.97      0.96      0.97       166
```

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: 0.9639
```

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: 0.0361
```

Precision

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

```
Precision 0.9742
```

Recall

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

```
Recall or Sensitivity: 0.9869
```

True Positive Rate

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

```
True Positive Rate: 0.9869
```

False Positive Rate

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

```
False Positive Rate: 0.3077
```

Specificity

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

```
Specificity 0.6923
```

Adjusting the threshold level

```
#print the first 18 predicted probabilities of two classes 8 and 1
y_pred_prob = logreg.predict_proba(X_test)[8:10]
y_pred_prob
```

```
array([[0.98431934, 0.01568066],
       [0.97699586, 0.02300414]])
```

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

	Prob of No cancer (0)	Prob of cancer (1)
0	0.984319	0.015681
1	0.976996	0.023004

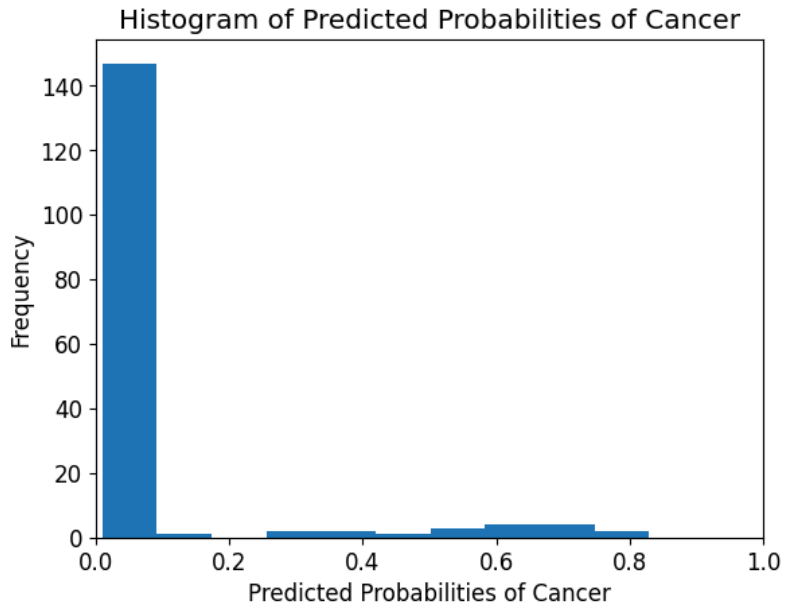
Next steps: View recommended plots

```
logreg.predict_proba (X_test) [0:10, 1]
```

```
array([0.02832186, 0.01378096, 0.02451266, 0.02027214, 0.01632066,
       0.02303176, 0.01549106, 0.01637031, 0.01568066, 0.02300414])
```

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

```
# Set font size
plt.rcParams['font.size'] = 12

# Plot histogram with 10 bins
plt.hist(y_predi, bins=10)

# Set title
plt.title('Histogram of Predicted Probabilities of Cancer')

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

# Set axis labels
plt.xlabel('Predicted Probabilities of Cancer')
plt.ylabel('Frequency')

# Show plot
plt.show()
```



The majority of predictions are concentrated around a low probability range (0-0.2), indicating that most individuals in this dataset have a low predicted risk of cancer. There are no predictions showing a moderate to high probability, which suggests that, according to this model, the studied population is at low risk for cancer.

Lower the threshold

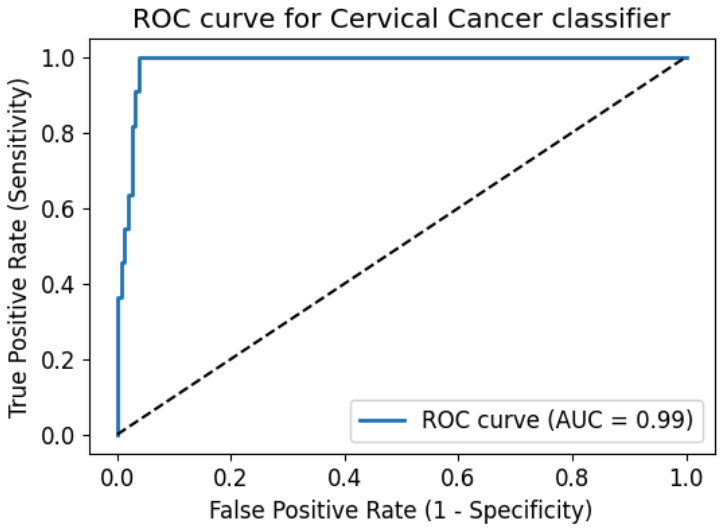
```
from sklearn.preprocessing import binarize
from sklearn.metrics import confusion_matrix, accuracy_score

for i in range(1, 5):
    cm1 = 0
    y_predi = logreg.predict_proba(X_test)[:, 1].reshape(-1, 1)
    y_pred2 = binarize(y_predi, threshold=i / 10)  # Correct usage of binarize
    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 0.2 threshold the Confusion Matrix is:

  [[148   7]
  [  0  11]]

  With 159 correct predictions
```

```
6 Type I errors (False Positives)

0 Type II errors (False Negatives)

Accuracy score: 0.963855421686747

Sensitivity: 1.0

Specificity: 0.9612903225806452

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


With 0.4 threshold the Confusion Matrix is:

[[150   5]
[  2   9]]

With 159 correct predictions

5 Type I errors (False Positives)

2 Type II errors (False Negatives)

Accuracy score: 0.9578313253012049

Sensitivity: 0.8181818181818182

Specificity: 0.967741935483871

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

ROC-AUC

```
from sklearn.metrics import roc_curve, auc

# Calculate ROC curve
fpr, tpr, thresholds = roc_curve(y_test, y_predi)
roc_auc = auc(fpr, tpr)

# Plot ROC curve
plt.figure(figsize=(6, 4))
plt.plot(fpr, tpr, linewidth=2, label='ROC curve (AUC = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], 'k--')
plt.title('ROC curve for Cervical Cancer classifier')
plt.xlabel('False Positive Rate (1 - Specificity)')
plt.ylabel('True Positive Rate (Sensitivity)')
plt.legend(loc='lower right')
plt.show()
```



The graph is a Receiver Operating Characteristic (ROC) curve for a Cervical Cancer classifier. The ROC curve is close to the top left corner, indicating high sensitivity and low false positive rate. With an Area Under Curve (AUC) of 0.99, the classifier demonstrates excellent performance, suggesting it is highly effective and reliable for predicting cervical cancer.

```
# Make predictions on the test set
y_pred = logreg.predict(X_test)

# Output predictions for each individual
for i, prediction in enumerate(y_pred):
    if prediction == 1:
        print(f"Index {i} prediction: will have cancer")
    else:
```