Hands-on Activity 11.2 Classification using Logistic Regression

Name: John Rome A. Belocora
Section: CPE22S3
Date: 04/28/2024
Teacher: Engr. Roman Richard

```
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
```

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

```
df = pd.concat([X,y])
df
```

	Age	Number of sexual partners	First sexual intercourse	Num of pregnancies	Smokes	Smokes (years)	Smokes (packs/year)	Hormo Contracepti
0	18	4.0	15.0	1.0	0.0	0.0	0.0	
1	15	1.0	14.0	1.0	0.0	0.0	0.0	
2	34	1.0	NaN	1.0	0.0	0.0	0.0	
3	52	5.0	16.0	4.0	1.0	37.0	37.0	
4	46	3.0	21.0	4.0	0.0	0.0	0.0	
...	...	...	...	...	...	...	...	
853	34	3.0	18.0	0.0	0.0	0.0	0.0	
854	32	2.0	19.0	1.0	0.0	0.0	0.0	
855	25	2.0	17.0	0.0	0.0	0.0	0.0	
856	33	2.0	24.0	2.0	0.0	0.0	0.0	
857	29	2.0	20.0	1.0	0.0	0.0	0.0	

Data Wrangling

```
# Identifying the Unique values of each columns
for columns in df:
  print(columns)
  print(df[columns].unique())
  print('  ')
```

```
Age
[18 15 34 52 46 42 51 26 45 44 27 43 40 41 39 37 38 36 35 33 31 32 30 23
 28 29 20 25 21 24 22 48 19 17 16 14 59 79 84 47 13 70 50 49]

Number of sexual partners
[ 4.  1.  5.  3.  2.  6. nan  7. 15.  8. 10. 28.  9.]
```

```
First sexual intercourse
[15. 14. nan 16. 21. 23. 17. 26. 20. 25. 18. 27. 19. 24. 32. 13. 29. 11.
 12. 22. 28. 10.]

Num of pregnancies
[ 1.  4.  2.  6.  3.  5. nan  8.  7.  0. 11. 10.]

Smokes
[ 0.  1. nan]

Smokes (years)
[ 0.         37.         34.          1.26697291  3.         12.
         nan 18.          7.         19.         21.         15.
 13.         16.          8.          4.         10.         22.
 14.          0.5        11.          9.          2.          5.
  6.          1.         32.         24.         28.         20.
  0.16      ]

Smokes (packs/year)
[0.00000000e+00 3.70000000e+01 3.40000000e+00 2.80000000e+00
 4.00000000e-02 5.13202128e-01 2.40000000e+00 6.00000000e+00
            nan 9.00000000e+00 1.60000000e+00 1.90000000e+01
 2.10000000e+01 3.20000000e-01 2.60000000e+00 8.00000000e-01
 1.50000000e+01 2.00000000e+00 5.70000000e+00 1.00000000e+00
 3.30000000e+00 3.50000000e+00 1.20000000e+01 2.50000000e-02
 2.75000000e+00 2.00000000e-01 1.40000000e+00 5.00000000e+00
 2.10000000e+00 7.00000000e-01 1.20000000e+00 7.50000000e+00
 1.25000000e+00 3.00000000e+00 7.50000000e-01 1.00000000e-01
 8.00000000e+00 2.25000000e+00 3.00000000e-03 7.00000000e+00
 4.50000000e-01 1.50000000e-01 5.00000000e-02 2.50000000e-01
 4.80000000e+00 4.50000000e+00 4.00000000e-01 3.70000000e-01
 2.20000000e+00 1.60000000e-01 9.00000000e-01 2.20000000e+01
 1.35000000e+00 5.00000000e-01 2.50000000e+00 4.00000000e+00
 1.30000000e+00 1.65000000e+00 2.70000000e+00 1.00000000e-03
 7.60000000e+00 5.50000000e+00 3.00000000e-01]

Hormonal Contraceptives
[ 0.  1. nan]

Hormonal Contraceptives (years)
[ 0.          3.         15.          2.          8.         10.
  5.          0.25        7.         22.         19.          0.5
  1.          0.58        9.         13.         11.          4.
 12.         16.          0.33              nan  0.16       14.
  0.08        2.28220052  0.66        6.          1.5         0.42
  0.67        0.75        2.5         4.5         6.5         0.17
 20.          3.5         0.41       30.         17.        ]

IUD
[ 0.  1. nan]
```

As we can see above there are some columns which only consist values of 0, 1 and nan, this means that 0 stands for "False" or "No", 1 stands for "True" or "Yes" and nan means that there are missing values. While on the other hand there are some columns that consists several unique values. We can also notice the column "STDs:AIDS" and "STDs:cervical condylomatosis" which consist values of only "0" and "nan" this means that they dont have any value contribution to the dataset so we can drop those columns.

Dropping Columns

```
# Dropping the STDs:AIDS and STDs:cervical condylomatosis column
df = df.drop(columns=['STDs:AIDS','STDs:cervical condylomatosis'])
df
```

	Age	Number of sexual partners	First sexual intercourse	Num of pregnancies	Smokes	Smokes (years)	Smokes (packs/year)	Hormo Contracepti
0	18	4.0	15.0	1.0	0.0	0.0	0.0	
1	15	1.0	14.0	1.0	0.0	0.0	0.0	
2	34	1.0	NaN	1.0	0.0	0.0	0.0	
3	52	5.0	16.0	4.0	1.0	37.0	37.0	
4	46	3.0	21.0	4.0	0.0	0.0	0.0	
...	...	...	...	...	...	...	...	
853	34	3.0	18.0	0.0	0.0	0.0	0.0	
854	32	2.0	19.0	1.0	0.0	0.0	0.0	
855	25	2.0	17.0	0.0	0.0	0.0	0.0	
856	33	2.0	24.0	2.0	0.0	0.0	0.0	
857	29	2.0	20.0	1.0	0.0	0.0	0.0	

Identifying numerical datatypes

```
# Checking for columns that are not object datatype
numerical = [var for var in df.columns if df[var].dtype!='O']
print('There are {} numerical variables\n'.format(len(numerical)))
print('The numerical variables are :', numerical)
```

```
There are 34 numerical variables

The numerical variables are : ['Age', 'Number of sexual partners', 'First sexual intercourse', 'Num of pregnancies', 'Smokes', 'Smokes (
```

Identifying Outliers in Numerical Variables

```
sample = df.copy()
```

```
# View summary statistics in numerical variables
print(round(sample[numerical].describe()),2)
```

```
             Age  Number of sexual partners  First sexual intercourse  \
count  858.0                      832.0                     851.0
mean    27.0                        3.0                      17.0
std      8.0                        2.0                       3.0
min     13.0                        1.0                      10.0
25%     20.0                        2.0                      15.0
50%     25.0                        2.0                      17.0
75%     32.0                        3.0                      18.0
max     84.0                       28.0                      32.0

       Num of pregnancies  Smokes  Smokes (years)  Smokes (packs/year)  \
count               802.0   845.0           845.0                845.0
mean                  2.0     0.0             1.0                  0.0
std                   1.0     0.0             4.0                  2.0
min                   0.0     0.0             0.0                  0.0
25%                   1.0     0.0             0.0                  0.0
50%                   2.0     0.0             0.0                  0.0
75%                   3.0     0.0             0.0                  0.0
max                  11.0     1.0            37.0                 37.0

       Hormonal Contraceptives  Hormonal Contraceptives (years)    IUD  ...  \
count                    750.0                            750.0  741.0  ...
mean                       1.0                              2.0    0.0  ...
std                        0.0                              4.0    0.0  ...
min                        0.0                              0.0    0.0  ...
25%                        0.0                              0.0    0.0  ...
50%                        1.0                              0.0    0.0  ...
75%                        1.0                              3.0    0.0  ...
max                        1.0                             30.0    1.0  ...
```

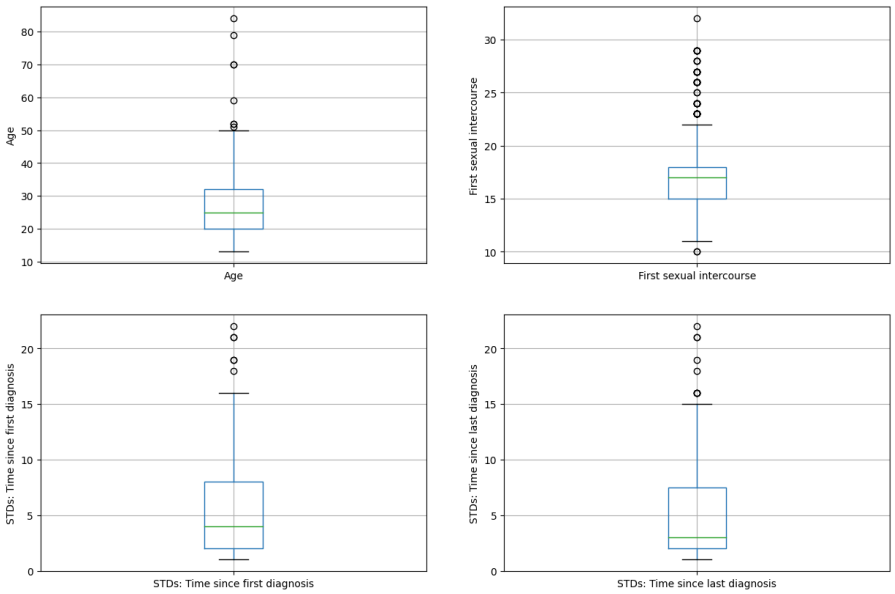
```
       STDs: Time since first diagnosis  STDs: Time since last diagnosis  \
count                              71.0                             71.0
mean                                6.0                              6.0
std                                 6.0                              6.0
min                                 1.0                              1.0
25%                                 2.0                              2.0
50%                                 4.0                              3.0
75%                                 8.0                              8.0
max                                22.0                             22.0

       Dx:Cancer  Dx:CIN  Dx:HPV     Dx  Hinselmann  Schiller  Citology  \
count      858.0   858.0   858.0  858.0       858.0     858.0     858.0
mean         0.0     0.0     0.0    0.0         0.0       0.0       0.0
std          0.0     0.0     0.0    0.0         0.0       0.0       0.0
min          0.0     0.0     0.0    0.0         0.0       0.0       0.0
25%          0.0     0.0     0.0    0.0         0.0       0.0       0.0
50%          0.0     0.0     0.0    0.0         0.0       0.0       0.0
75%          0.0     0.0     0.0    0.0         0.0       0.0       0.0
max          1.0     1.0     1.0    1.0         1.0       1.0       1.0

       Biopsy
count   858.0
mean      0.0
std       0.0
min       0.0
25%       0.0
50%       0.0
75%       0.0
```

As we can see in the results above, I think the 'Age', 'First sexual intercourse', 'STDs: Time since first diagnosis', and 'STDs: Time since last diagnosis' columns may contain outliers because of their shown result. To confirm this, we can use boxplots to visualize outliers

Using Boxplots to visualize outliers

```
#Using Boxplots to visualize outliers
plt.figure(figsize=(15,10))

plt.subplot(2, 2, 1)
fig = sample.boxplot(column='Age')
fig.set_title('')
fig.set_ylabel('Age')

plt.subplot(2, 2, 2)
fig = sample.boxplot(column='First sexual intercourse')
fig.set_title('')
fig.set_ylabel('First sexual intercourse')

plt.subplot(2, 2, 3)
fig = sample.boxplot(column='STDs: Time since first diagnosis')
fig.set_title('')
fig.set_ylabel('STDs: Time since first diagnosis')

plt.subplot(2, 2, 4)
fig = sample.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')
```

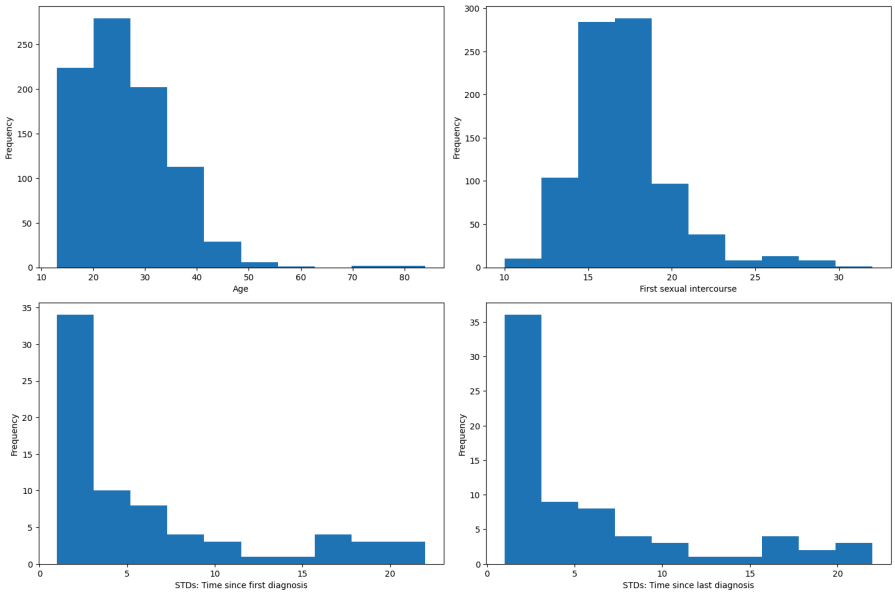


As we can notice in the boxplot results, we can confirm that there really are outliers in these variables.

Checking the Distribution of Variables

We can use histograms to check the distributions of variables and identify if they are normal or skewed.

```
#Using plot histogram to check distribution
plt.figure(figsize=(15,10))

plt.subplot(2, 2, 1)
plt.hist(sample['Age'])
plt.xlabel('Age')
plt.ylabel('Frequency')

plt.subplot(2, 2, 2)
plt.hist(sample['First sexual intercourse'], bins=10)
plt.xlabel('First sexual intercourse')
plt.ylabel('Frequency')

plt.subplot(2, 2, 3)
plt.hist(sample['STDs: Time since first diagnosis'], bins=10)
plt.xlabel('STDs: Time since first diagnosis')
plt.ylabel('Frequency')

plt.subplot(2, 2, 4)
plt.hist(sample['STDs: Time since last diagnosis'], bins=10)
plt.xlabel('STDs: Time since last diagnosis')
plt.ylabel('Frequency')

plt.tight_layout()
plt.show()
```



Base on the result of our histogram, we can notice that all the four variables are skewed. Therefore we can use interquartile range to find outliers.

First Sexual Intercourse Outliers

```
# Finding outliers for First sexual intercourse variable
IQR = sample['First sexual intercourse'].quantile(0.75) - sample['First sexual intercourse'].quantile(0.25)
Lower_fence = sample['First sexual intercourse'].quantile(0.25) - (IQR * 3)
Upper_fence = sample['First sexual intercourse'].quantile(0.25) + (IQR * 3)
print('First sexual intercourse outliers are values < {lowerboundary} or > {upperboundary}'.format(lowerboundary=Lower_fence, upperboundary=
```

```
First sexual intercourse outliers are values < 6.0 or > 24.0
```

Age Outliers

```
# Finding outliers for Age variable
IQR = sample['Age'].quantile(0.75) - sample['Age'].quantile(0.25)
Lower_fence = sample['Age'].quantile(0.25) - (IQR * 3)
Upper_fence = sample['Age'].quantile(0.25) + (IQR * 3)
print('Age outliers are values < {lowerboundary} or > {upperboundary}'.format(lowerboundary=Lower_fence, upperboundary=Upper_fence))
```

```
Age outliers are values < -16.0 or > 56.0
```

STDs: Time since first diagnosis Outliers

```
# Finding outliers for STDs: Time since first diagnosis variable
IQR = sample['STDs: Time since first diagnosis'].quantile(0.75) - sample['STDs: Time since first diagnosis'].quantile(0.25)
Lower_fence = sample['STDs: Time since first diagnosis'].quantile(0.25) - (IQR * 3)
Upper_fence = sample['STDs: Time since first diagnosis'].quantile(0.25) + (IQR * 3)
print('STDs: Time since first diagnosis outliers are values < {lowerboundary} or > {upperboundary}'.format(lowerboundary=Lower_fence, upperb
```

```
STDs: Time since first diagnosis outliers are values < -16.0 or > 20.0
```

STDs: Time since last diagnosis Outliers

```
# Finding outliers for STDs: Time since last diagnosis variable
IQR = sample['STDs: Time since last diagnosis'].quantile(0.75) - sample['STDs: Time since last diagnosis'].quantile(0.25)
Lower_fence = sample['STDs: Time since last diagnosis'].quantile(0.25) - (IQR * 3)
Upper_fence = sample['STDs: Time since last diagnosis'].quantile(0.25) + (IQR * 3)
print('STDs: Time since last diagnosis outliers are values < {lowerboundary} or > {upperboundary}'.format(lowerboundary=Lower_fence, upperbou
```

```
STDs: Time since last diagnosis outliers are values < -14.5 or > 18.5
```

Declare feature vector and target variable

We are going to use the number of pregnancies as our target variable

```
X = sample.drop(['Num of pregnancies'], axis=1)
y = sample['Num of pregnancies']
```

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 = 0.1, random_state=0)
```

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

```
((772, 33), (86, 33))
```

```
# Check the shape of y_train and y_test
y_train.shape, y_test.shape
```

```
((772,), (86,))
```

Feature Engineering

Feature Engineering is the process of transforming raw data into useful features that help us to understand our model better and increase its predicitve power. I will carry out feature engineering on different types of variables.

```
# Displaying Numerical Variables

numerical = [col for col in X_train.columns if X_train[col].dtypes != 'O']
numerical
```

```
['Age',
 'Number of sexual partners',
 'First sexual intercourse',
 'Smokes',
 'Smokes (years)',
 'Smokes (packs/year)',
 'Hormonal Contraceptives',
 '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']
```

We can clearly see that all of our Columns are Numerical datatypes

Engineering missing values in numerical variables

```
# Checking missing values in numerical variables in X_train

X_train[numerical].isnull().sum()
```

```
Age                                   0
Number of sexual partners            26
First sexual intercourse              7
Smokes                               12
Smokes (years)                       12
Smokes (packs/year)                  12
Hormonal Contraceptives              98
Hormonal Contraceptives (years)      98
IUD                                 104
IUD (years)                         104
STDs                                 94
STDs (number)                        94
STDs:condylomatosis                  94
STDs:vaginal condylomatosis          94
STDs:vulvo-perineal condylomatosis    94
STDs:syphilis                        94
STDs:pelvic inflammatory disease     94
STDs:genital herpes                  94
STDs:molluscum contagiosum           94
STDs:HIV                             94
STDs:Hepatitis B                     94
STDs:HPV                             94
STDs: Number of diagnosis             0
STDs: Time since first diagnosis    706
STDs: Time since last diagnosis     706
Dx:Cancer                             0
Dx:CIN                                0
Dx:HPV                                0
Dx                                    0
Hinselmann                            0
Schiller                              0
Citology                              0
Biopsy                                0
dtype: int64
```

```
# Print percentage of missing values in the numerical variables in training set

for col in numerical:
    if X_train[col].isnull().mean()>0:
        print(col,':', round(X_train[col].isnull().mean(),4),'%')
```

```
Number of sexual partners : 0.0337 %
First sexual intercourse : 0.0091 %
Smokes : 0.0155 %
Smokes (years) : 0.0155 %
Smokes (packs/year) : 0.0155 %
Hormonal Contraceptives : 0.1269 %
Hormonal Contraceptives (years) : 0.1269 %
IUD : 0.1347 %
IUD (years) : 0.1347 %
STDs : 0.1218 %
STDs (number) : 0.1218 %
STDs:condylomatosis : 0.1218 %
STDs:vaginal condylomatosis : 0.1218 %
STDs:vulvo-perineal condylomatosis : 0.1218 %
STDs:syphilis : 0.1218 %
STDs:pelvic inflammatory disease : 0.1218 %
STDs:genital herpes : 0.1218 %
STDs:molluscum contagiosum : 0.1218 %
STDs:HIV : 0.1218 %
STDs:Hepatitis B : 0.1218 %
STDs:HPV : 0.1218 %
STDs: Time since first diagnosis : 0.9145 %
STDs: Time since last diagnosis : 0.9145 %
```

We can use Mode imputation to fill missing values that we have in our columns

```
# Impute missing values in X_train and X_test with respective column mode in X_train
for sample in [X_train, X_test]:
    for col in numerical:
        col_mode = X_train[col].mode()[0]  # Extract mode value
        sample[col].fillna(col_mode, inplace=True)

# Check again missing values in numerical variables in X_train
X_train[numerical].isnull().sum()
```

```
Age                                 0
Number of sexual partners           0
First sexual intercourse            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: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: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
```

Engineering Outliers in Numerical Variables

Since we have seen in our previous identifying outliers that "Age", "First sexual intercourse", "STDs: Time since first diagnosis", and "STDs: Time since last diagnosis" columns contain outliers. We can use top coding approach to cap maximum values and remove outliers from the above variables

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

for sample in [X_train, X_test]:
    sample['Age'] = max_value(sample, 'Age', 56)
    sample['First sexual intercourse'] = max_value(sample, 'First sexual intercourse', 24)
    sample['STDs: Time since first diagnosis'] = max_value(sample, 'STDs: Time since first diagnosis', 20)
    sample['STDs: Time since last diagnosis'] = max_value(sample, 'STDs: Time since last diagnosis', 18.5)
```

Maximum cap value for the Age

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

```
(56, 56)
```

Maximum cap value for the First sexual intercourse

```
X_train['First sexual intercourse'].max(), X_test['First sexual intercourse'].max()
```

```
(24.0, 24.0)
```

Maximum cap value for the STDs: Time since first diagnosis

```
X_train['STDs: Time since first diagnosis'].max(), X_test['STDs: Time since first diagnosis'].max()
```

```
(20.0, 11.0)
```

Maximum cap value for the STDs: Time since last diagnosis

```
X_train['STDs: Time since last diagnosis'].max(), X_test['STDs: Time since last diagnosis'].max()
```

```
(18.5, 11.0)
```

Now we can see that the outliers in Age, First sexual intercourse, STDs: Time since first diagnosis, and STDs: Time since last diagnosis are now capped.

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

	Age	Number of sexual partners	First sexual intercourse	Smokes	Smokes (years)	Smokes (packs/year)	Hormonal Contraceptives	Hormonal Contraceptives (years)	IUD	IUD (years)	.
count	772.000000	772.000000	772.000000	772.000000	772.000000	772.000000	772.000000	772.000000	772.000000	772.000000	
mean	26.699482	2.522021	16.847150	0.148964	1.233676	0.473064	0.687824	1.963163	0.091969	0.427642	
std	7.990289	1.681757	2.486269	0.356284	4.046712	2.304001	0.463682	3.496010	0.289169	1.816494	
min	14.000000	1.000000	10.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	20.000000	2.000000	15.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
50%	25.500000	2.000000	17.000000	0.000000	0.000000	0.000000	1.000000	0.250000	0.000000	0.000000	
75%	32.000000	3.000000	18.000000	0.000000	0.000000	0.000000	1.000000	2.000000	0.000000	0.000000	
max	56.000000	28.000000	24.000000	1.000000	37.000000	37.000000	1.000000	22.000000	1.000000	19.000000	

8 rows × 33 columns

Feature Scaling

```
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_train, columns=[cols])


X_train.describe()
```

	Age	Number of sexual partners	First sexual intercourse	Smokes	Smokes (years)	Smokes (packs/year)	Hormonal Contraceptives	Hormonal Contraceptives (years)	IUD	IUD (years)	.
count	772.000000	772.000000	772.000000	772.000000	772.000000	772.000000	772.000000	772.000000	772.000000	772.000000	
mean	0.302369	0.056371	0.489082	0.148964	0.033343	0.012786	0.687824	0.089235	0.091969	0.022507	
std	0.190245	0.062287	0.177591	0.356284	0.109371	0.062270	0.463682	0.158910	0.289169	0.095605	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	0.142857	0.037037	0.357143	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
50%	0.273810	0.037037	0.500000	0.000000	0.000000	0.000000	1.000000	0.011364	0.000000	0.000000	
75%	0.428571	0.074074	0.571429	0.000000	0.000000	0.000000	1.000000	0.090909	0.000000	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 × 33 columns

Now we can use X_train dataset to be fed into the Logistic Regression classifier.

Model Training

```
from sklearn.linear_model import LogisticRegression

# Handle missing values in y_train
y_train_mode = y_train.mode()[0]
y_train.fillna(y_train_mode, inplace=True)

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

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

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

Predict Results

```
y_pred_test = logreg.predict(X_test)

y_pred_test
```

```
array([1., 1., 1., 1., 1., 3., 1., 1., 3., 1., 2., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 2., 2., 1., 1., 1., 1., 2., 1., 3.,
       1., 1., 1., 1., 1., 1., 2., 1., 1., 2., 1., 2., 1., 1., 1., 1., 1.,
       1., 3., 1., 3., 1., 1., 1., 1., 1., 1., 1., 2., 1., 1., 1., 1.,
```

```
       1., 1., 2., 3., 1., 1., 1., 1., 2., 3., 1., 2., 1., 2., 2., 1., 3.,
       1., 1., 1., 3., 3., 1., 1., 1., 1., 2., 1., 2., 1., 1., 2., 2., 1.,
       1., 3., 1., 2., 1., 1., 1., 2., 2., 1., 1., 1., 1., 1., 2., 1., 1.,
       2., 1., 1., 3., 2., 1., 3., 1., 1., 1., 2., 1., 3., 1., 1., 1., 1.,
       1., 3., 3., 4., 1., 3., 1., 1., 4., 2., 1., 1., 3., 1., 1., 1., 2.,
       3., 2., 3., 1., 2., 1., 1., 1., 1., 1., 1., 2., 2., 2., 1., 1., 1.,
       2., 2., 1., 1., 4., 1., 3., 1., 1., 1., 1., 1., 1., 2., 3., 1., 1.,
       1., 1., 6., 1., 1., 1., 2., 2., 1., 4., 3., 1., 1., 1., 1., 3., 3.,
       2., 3., 4., 2., 1., 1., 3., 1., 1., 2., 3., 1., 3., 3., 2., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 2., 1., 1., 1., 2., 3., 1., 1.,
       1., 1., 1., 1., 1., 1., 2., 2., 1., 2., 1., 1., 1., 1., 1., 2., 1.,
       1., 3., 1., 1., 1., 2., 1., 1., 1., 1., 1., 3., 1., 1., 1., 1., 1.,
       1., 1., 1., 2., 1., 1., 1., 1., 1., 1., 3., 1., 1., 4., 1., 1., 1.,
       1., 1., 4., 1., 2., 1., 3., 1., 1., 1., 1., 1., 1., 2., 1., 1., 2.,
       1., 1., 3., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 2., 2., 1.,
       1., 3., 2., 1., 4., 1., 2., 1., 1., 1., 1., 1., 2., 1., 1., 1., 2.,
       1., 4., 4., 2., 1., 1., 1., 1., 2., 2., 3., 1., 2., 1., 1., 1., 3.,
       1., 1., 1., 1., 1., 1., 2., 2., 1., 1., 3., 1., 2., 1., 1., 1., 2.,
       1., 1., 1., 2., 1., 2., 2., 4., 1., 3., 3., 3., 1., 1., 1., 1., 3.,
       1., 1., 1., 1., 1., 1., 1., 1., 2., 1., 1., 1., 1., 1., 1., 1., 1.,
       2., 3., 2., 1., 3., 2., 2., 3., 1., 1., 2., 3., 1., 1., 1., 1., 1.,
       1., 1., 2., 1., 1., 1., 1., 1., 1., 1., 2., 1., 2., 2., 1., 1., 2.,
       4., 1., 3., 1., 1., 2., 1., 1., 1., 1., 1., 2., 1., 1., 3., 2., 1.,
       2., 1., 1., 1., 1., 1., 3., 1., 1., 1., 2., 2., 1., 1., 2., 3., 1.,
       1., 1., 2., 2., 3., 1., 1., 2., 2., 1., 1., 1., 3., 1., 1., 1., 2.,
       1., 1., 2., 1., 4., 1., 3., 3., 1., 1., 1., 2., 1., 1., 1., 1., 1.,
       1., 2., 3., 1., 2., 3., 2., 1., 2., 1., 2., 1., 2., 4., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 3., 1., 1., 3., 2., 3., 1., 1., 1., 1.,
       1., 1., 1., 1., 2., 1., 1., 3., 1., 1., 1., 1., 2., 1., 2., 2., 1.,
       3., 1., 3., 1., 1., 1., 3., 1., 1., 2., 3., 1., 3., 3., 1., 3., 2.,
       1., 1., 1., 1., 2., 1., 1., 1., 1., 1., 1., 2., 1., 1., 3., 2., 1.,
       1., 1., 2., 2., 3., 2., 1., 2., 1., 1., 3., 2., 1., 1., 2., 1., 1.,
       1., 2., 1., 1., 1., 1., 1., 3., 2., 1., 1., 3., 1., 1., 2., 1., 2.,
       1., 1., 2., 1., 3., 1., 1., 2., 1., 2., 3., 2., 2., 1., 1., 1., 2.,
       2., 1., 2., 2., 1., 1., 1., 2., 1., 3., 3., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 2., 1., 1., 1., 2., 1., 3.,
       1., 1., 1., 3., 3., 1., 2., 2., 2., 4., 1., 1., 1., 1., 3., 2., 1.,
       3., 2., 1., 1., 1., 1., 1., 1., 3., 1., 1., 1., 1., 2., 2., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 3., 1., 1., 2., 1., 1., 1., 1., 1., 1.,
       1., 4., 1., 3., 1., 1., 2., 1., 1., 2., 1., 3., 1., 2., 1., 4., 1.,
       1., 2., 2., 1., 2., 1., 1., 1., 3., 2., 1., 1., 1., 1., 1., 3., 2.,
       1., 2., 1., 1., 2., 1., 4.])
```

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

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

```
array([], shape=(0, 10), dtype=float64)
```

Comparing the train-set and test-set accuracy

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

```
y_pred_train
```

```
array([1., 1., 1., 1., 1., 3., 1., 1., 3., 1., 2., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 2., 2., 1., 1., 1., 1., 2., 1., 3.,
       1., 1., 1., 1., 1., 1., 2., 1., 1., 2., 1., 2., 1., 1., 1., 1., 1.,
       1., 3., 1., 3., 1., 1., 1., 1., 1., 1., 1., 1., 2., 1., 1., 1., 1.,
       1., 1., 2., 3., 1., 1., 1., 1., 2., 3., 1., 2., 1., 2., 2., 1., 3.,
       1., 1., 1., 3., 3., 1., 1., 1., 1., 2., 1., 2., 1., 1., 2., 2., 1.,
       1., 3., 1., 2., 1., 1., 1., 2., 2., 1., 1., 1., 1., 1., 2., 1., 1.,
       2., 1., 1., 3., 2., 1., 3., 1., 1., 1., 2., 1., 3., 1., 1., 1., 1.,
       1., 3., 3., 4., 1., 3., 1., 1., 4., 2., 1., 1., 3., 1., 1., 1., 2.,
       3., 2., 3., 1., 2., 1., 1., 1., 1., 1., 1., 2., 2., 2., 1., 1., 1.,
       2., 2., 1., 1., 4., 1., 3., 1., 1., 1., 1., 1., 1., 2., 3., 1., 1.,
       1., 1., 6., 1., 1., 1., 2., 2., 1., 4., 3., 1., 1., 1., 1., 3., 3.,
       2., 3., 4., 2., 1., 1., 3., 1., 1., 2., 3., 1., 3., 3., 2., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 2., 1., 1., 1., 2., 3., 1., 1.,
       1., 1., 1., 1., 1., 1., 2., 2., 1., 2., 1., 1., 1., 1., 1., 2., 1.,
       1., 3., 1., 1., 1., 2., 1., 1., 1., 1., 1., 3., 1., 1., 1., 1., 1.,
       1., 1., 1., 2., 1., 1., 1., 1., 1., 1., 3., 1., 1., 4., 1., 1., 1.,
       1., 1., 4., 1., 2., 1., 3., 1., 1., 1., 1., 1., 1., 2., 1., 1., 2.,
       1., 1., 3., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 2., 2., 1.,
       1., 3., 2., 1., 4., 1., 2., 1., 1., 1., 1., 1., 2., 1., 1., 1., 2.,
       1., 4., 4., 2., 1., 1., 1., 1., 2., 2., 3., 1., 2., 1., 1., 1., 3.,
       1., 1., 1., 1., 1., 1., 2., 2., 1., 1., 3., 1., 2., 1., 1., 1., 2.,
       1., 1., 1., 2., 1., 2., 2., 4., 1., 3., 3., 3., 1., 1., 1., 1., 3.,
       1., 1., 1., 1., 1., 1., 1., 1., 2., 1., 1., 1., 1., 1., 1., 1., 1.,
       2., 3., 2., 1., 3., 2., 2., 3., 1., 1., 2., 3., 1., 1., 1., 1., 1.,
```

```
1., 1., 2., 1., 1., 1., 1., 1., 1., 1., 2., 1., 2., 2., 1., 1., 2.,
4., 1., 3., 1., 1., 2., 1., 1., 1., 1., 1., 2., 1., 1., 3., 2., 1.,
2., 1., 1., 1., 1., 1., 3., 1., 1., 1., 2., 2., 1., 1., 2., 3., 1.,
1., 1., 2., 2., 3., 1., 1., 2., 2., 1., 1., 1., 3., 1., 1., 1., 2.,
1., 1., 2., 1., 4., 1., 3., 3., 1., 1., 1., 2., 1., 1., 1., 1., 1.,
1., 2., 3., 1., 2., 3., 2., 1., 2., 1., 2., 1., 2., 4., 1., 1., 1.,
1., 1., 1., 1., 1., 1., 1., 3., 1., 1., 3., 2., 3., 1., 1., 1., 1.,
1., 1., 1., 1., 2., 1., 1., 3., 1., 1., 1., 1., 2., 1., 2., 2., 1.,
3., 1., 3., 1., 1., 1., 3., 1., 1., 2., 3., 1., 3., 3., 1., 3., 2.,
1., 1., 1., 1., 2., 1., 1., 1., 1., 1., 1., 2., 1., 1., 3., 2., 1.,
1., 1., 2., 2., 3., 2., 1., 2., 1., 1., 3., 2., 1., 1., 2., 1., 1.,
1., 2., 1., 1., 1., 1., 1., 3., 2., 1., 1., 3., 1., 1., 2., 1., 2.,
1., 1., 2., 1., 3., 1., 1., 2., 1., 2., 3., 2., 2., 1., 1., 1., 2.,
2., 1., 2., 2., 1., 1., 1., 2., 1., 3., 3., 1., 1., 1., 1., 1., 1.,
1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 2., 1., 1., 1., 2., 1., 3.,
1., 1., 1., 3., 3., 1., 2., 2., 2., 4., 1., 1., 1., 1., 3., 2., 1.,
3., 2., 1., 1., 1., 1., 1., 1., 3., 1., 1., 1., 1., 2., 2., 1., 1.,
1., 1., 1., 1., 1., 1., 1., 3., 1., 1., 2., 1., 1., 1., 1., 1., 1.,
1., 4., 1., 3., 1., 1., 2., 1., 1., 2., 1., 3., 1., 2., 1., 4., 1.,
1., 2., 2., 1., 2., 1., 1., 1., 3., 2., 1., 1., 1., 1., 1., 3., 2.,
1., 2., 1., 1., 2., 1., 4.])
```

Adjusting the threshold level