In [1]:

```
import warnings #Importing few important python libraries
warnings.filterwarnings('ignore')
import pandas as pd
import numpy as np
from sklearn import preprocessing
from numpy import linalg as LA
from sklearn.model_selection import train_test_split
import math
import matplotlib.pyplot as plt
import copy
import math
%matplotlib inline
```

In [2]:

```
df = pd.read_csv('risk_factors_cervical_cancer.csv')
```

In [3]:

```
df.head()
```

Out[3]:

	Age	Number of sexual partners	First sexual intercourse	Num of pregnancies	Smokes	Smokes (years)	Smokes (packs/year)	Hormonal Contraceptives
0	18	4.0	15.0	1.0	0.0	0.0	0.0	0.0
1	15	1.0	14.0	1.0	0.0	0.0	0.0	0.0
2	34	1.0	?	1.0	0.0	0.0	0.0	0.0
3	52	5.0	16.0	4.0	1.0	37.0	37.0	1.0
4	46	3.0	21.0	4.0	0.0	0.0	0.0	1.0

5 rows × 36 columns

In [4]:

```
df = pd.read_csv('risk_factors_cervical_cancer.csv',na_values = ["?"])
```

In [5]:

```
df.shape
```

Out[5]:

```
(858, 36)
```

In [6]:

```
df.head()
```

Out[6]:

	Age	Number of sexual partners	First sexual intercourse	Num of pregnancies	Smokes	Smokes (years)	Smokes (packs/year)	Hormonal Contraceptives
0	18	4.0	15.0	1.0	0.0	0.0	0.0	0.0
1	15	1.0	14.0	1.0	0.0	0.0	0.0	0.0
2	34	1.0	NaN	1.0	0.0	0.0	0.0	0.0
3	52	5.0	16.0	4.0	1.0	37.0	37.0	1.0
4	46	3.0	21.0	4.0	0.0	0.0	0.0	1.0

5 rows × 36 columns

In [7]:

```
df.dtypes
```

Out[7]:

```
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
```

In [8]:

```
# Remove columns with all na values
all_na = df.columns[df.isna().all()]
df.drop(all_na, axis = 1, inplace = True)
```

In [9]:

```
# Separating binary and non binary Data
df_bin = df.loc[:,df.nunique() <=2]
df_nbin = df.loc[:,df.nunique() >2]
```

In [10]:

```
df_bin.shape, df_nbin.shape
```

Out[10]:

```
((858, 24), (858, 12))
```

In [11]:

```
# Check na values in binary
df_bin.isna().sum()
```

Out[11]:

```
Smokes                                 13
Hormonal Contraceptives               108
IUD                                   117
STDs                                  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
Dx:Cancer                               0
Dx:CIN                                  0
Dx:HPV                                  0
Dx                                      0
Hinselmann                              0
Schiller                                0
Citology                                0
Biopsy                                  0
dtype: int64
```

In [12]:

```
#Filling na values in binary variables
col_bin = df_bin.columns[df_bin.isna().any()]
for col in col_bin:
    df_bin[col].fillna((df_bin[col].mode()[0]), inplace = True)

df_bin.isna().sum()
```

Out[12]:

```
Smokes                                0
Hormonal Contraceptives               0
IUD                                   0
STDs                                  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
Dx:Cancer                             0
Dx:CIN                                0
Dx:HPV                                0
Dx                                    0
Hinselmann                            0
Schiller                              0
Citology                              0
Biopsy                                0
dtype: int64
```

In [13]:

```
# Check na values in nonbinary
df_nbin.isna().sum()
```

Out[13]:

```
Age                                   0
Number of sexual partners            26
First sexual intercourse              7
Num of pregnancies                   56
Smokes (years)                       13
Smokes (packs/year)                  13
Hormonal Contraceptives (years)     108
IUD (years)                         117
STDs (number)                       105
STDs: Number of diagnosis             0
STDs: Time since first diagnosis    787
STDs: Time since last diagnosis     787
dtype: int64
```

In [14]:

```
# Filling na values in non binary variables
col_nbin = df_nbin.columns[df_nbin.isna().any()]
for col in col_nbin:
    df_nbin[col].fillna((df_nbin[col].median()), inplace = True)

df_nbin.isna().sum()
```

Out[14]:

```
Age                                   0
Number of sexual partners             0
First sexual intercourse              0
Num of pregnancies                    0
Smokes (years)                        0
Smokes (packs/year)                   0
Hormonal Contraceptives (years)       0
IUD (years)                           0
STDs (number)                         0
STDs: Number of diagnosis             0
STDs: Time since first diagnosis      0
STDs: Time since last diagnosis       0
dtype: int64
```

In [15]:

```
# Removing columns with same values in all rows
df_bin = df_bin.loc[:, (df_bin!= df_bin.loc[0]).any()]
df_nbin = df_nbin.loc[:, (df_nbin!= df_nbin.loc[0]).any()]
```

In [16]:

```
# Removing Duplicate Columns
df_bin = df_bin.T.drop_duplicates().T
df_nbin = df_nbin.T.drop_duplicates().T
```

In [17]:

```
# Description of Non Binary Data
df_nbin.describe()
```

Out[17]:

	Age	Number of sexual partners	First sexual intercourse	Num of pregnancies	Smokes (years)	Smokes (packs/year)	Horn Contracep (y
count	858.000000	858.000000	858.000000	858.000000	858.000000	858.000000	858.0C
mean	26.820513	2.511655	16.995338	2.257576	1.201241	0.446278	2.03
std	8.497948	1.644759	2.791883	1.400981	4.060623	2.210351	3.56
min	13.000000	1.000000	10.000000	0.000000	0.000000	0.000000	0.0C
25%	20.000000	2.000000	15.000000	1.000000	0.000000	0.000000	0.0C
50%	25.000000	2.000000	17.000000	2.000000	0.000000	0.000000	0.5C
75%	32.000000	3.000000	18.000000	3.000000	0.000000	0.000000	2.0C
max	84.000000	28.000000	32.000000	11.000000	37.000000	37.000000	30.0C

In [18]:

```
# Scaling (Normalization) of Numerical variables
min_max_scaler = preprocessing.MinMaxScaler()
nbin = df_nbin.values
nbin_scaled = min_max_scaler.fit_transform(nbin)
df_nbin_scaled = pd.DataFrame(nbin_scaled, columns = df_nbin.columns)
```

In [19]:

```
# Joining binary and non binary DFs to form cleaned DF
df_cleaned = df_bin.join(df_nbin_scaled)
```

In [20]:

```
X = df_cleaned.drop("Dx:Cancer", axis=1) # X is a dataframe containing all the features
y = df_cleaned["Dx:Cancer"].copy() # Y is a dataframe containing only the target variabl

X_test = df_cleaned.drop("Dx:Cancer", axis=1)
y_test = df_cleaned["Dx:Cancer"].copy()
```

In [21]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=1
print("X_train.shape", X_train.shape, "y_train.shape", y_train.shape)
print("X_test.shape", X_test.shape, "y_test.shape", y_test.shape)
```

```
X_train.shape (643, 33) y_train.shape (643,)
X_test.shape (215, 33) y_test.shape (215,)
```

In [22]:

```
def sigmoid(z):
    g = 1/(1+np.exp(-z))
    return g
```

In [23]:

```
def compute_cost(X, y, theta):
    # Number of training examples
    m = len(y)

    # Compute the hypothesis function
    z = np.dot(X, theta)
    h = sigmoid(z)

    # Compute the cost function
    J = (-1 / m) * np.sum(y * np.log(h) + (1 - y) * np.log(1 - h))

    return J
```

In [24]:

```
def gradient(X, y, theta):
    # Number of training examples
    m = len(y)

    # Compute the hypothesis function
    z = np.dot(X, theta)
    h = sigmoid(z)

    # Compute the gradient
    #function computes the gradient of the cost function with respect to the parameters.
    #This is done using the formula below
    grad = (1 / m) * np.dot(X.T, (h - y))

    return grad
```

In [25]:

```
def gradient_descent(X, y, theta, alpha, num_iters):
    # Number of training examples
    m = len(y)

    # List to store the cost function values over iterations
    J_history = []

    for i in range(num_iters):
        # Compute the hypothesis function and its error
        z = np.dot(X, theta)
        h = 1 / (1 + np.exp(-z))
        error = h - y

        # Compute the gradient and update the parameters
        grad = (1 / m) * np.dot(X.T, error)
        theta -= alpha * grad

        # Compute the cost function and append it to the list
        J_history.append(compute_cost(X, y, theta))

    return theta, J_history
```

In [26]:

```
np.random.seed(0)
theta = np.random.rand(X_train.shape[1])
alpha = 0.01
num_iters = 1000
theta, J_history = gradient_descent(X_train, y_train, theta, alpha, num_iters)
compute_cost(X_train, y_train,theta)
```

Out[26]:

```
0.3341059499223987
```

In [27]:

```
def predict(X, theta):
    # Compute the hypothesis function
    # By computing the hypothesis function for each example in the input feature matrix
    # These probabilities can then be rounded to the nearest integer to get the predicte
    z = np.dot(X, theta)
    h = sigmoid(z)

    # Round to nearest integer to get predicted labels
    y_pred = np.round(h)

    return y_pred
```

In [28]:

```
# Predict on test set
y_pred = predict(X_test, theta)

# Compute accuracy
accuracy = np.mean((y_pred == y_test)*100)

print("Test set accuracy: {:.2f}%".format(accuracy))
```

```
Test set accuracy: 88.37%
```

In [29]:

```
from sklearn.preprocessing import StandardScaler
```

In [30]:

```
patient_data = np.array([[18,4,15,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

# make a prediction for the patient using your trained model
prediction = predict(patient_data,theta)
print('Prediction: %d'%(prediction))
```

```
Prediction: 0
```