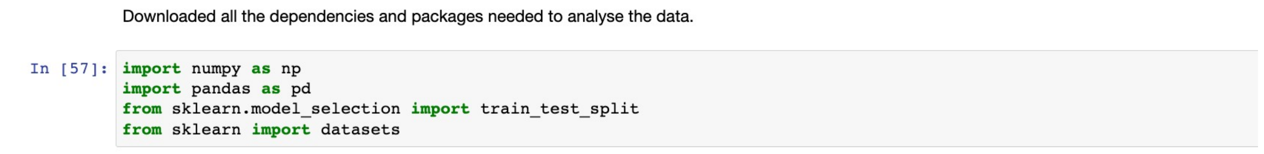
Newton Raphson Logistic Regression Report

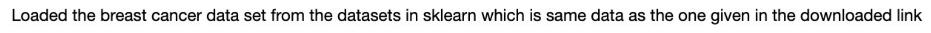
Started by importing the required dependencies:

NumPy, pandas, train_test_split , datasets



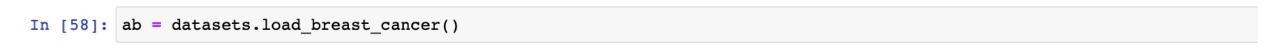
Downloaded all the dependencies and packages needed to analyse the data.

```
In [57]: import numpy as np
         import pandas as pd
         from sklearn.model_selection import train_test_split
         from sklearn import datasets
```



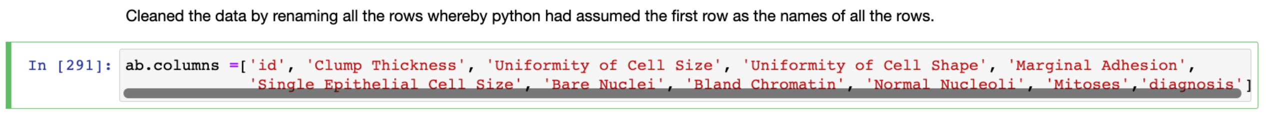
Loaded the breast cancer data set from the datasets in sklearn which is same data as the one given in the downloaded link

Loaded the data set



```
In [58]: ab = datasets.load_breast_cancer()
```

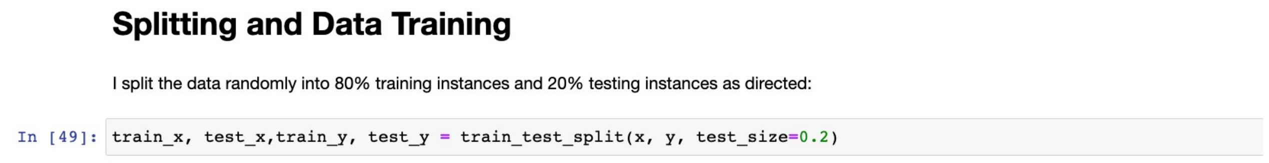
Cleaned the data



Cleaned the data by renaming all the rows whereby python had assumed the first row as the names of all the rows.

```
In [291]: ab.columns =['id', 'Clump Thickness', 'Uniformity of Cell Size', 'Uniformity of Cell Shape', 'Marginal Adhesion',
                       'Single Epithelial Cell Size', 'Bare Nuclei', 'Bland Chromatin', 'Normal Nucleoli', 'Mitoses','diagnosis']
```

Split the data into train 80% and test 20%

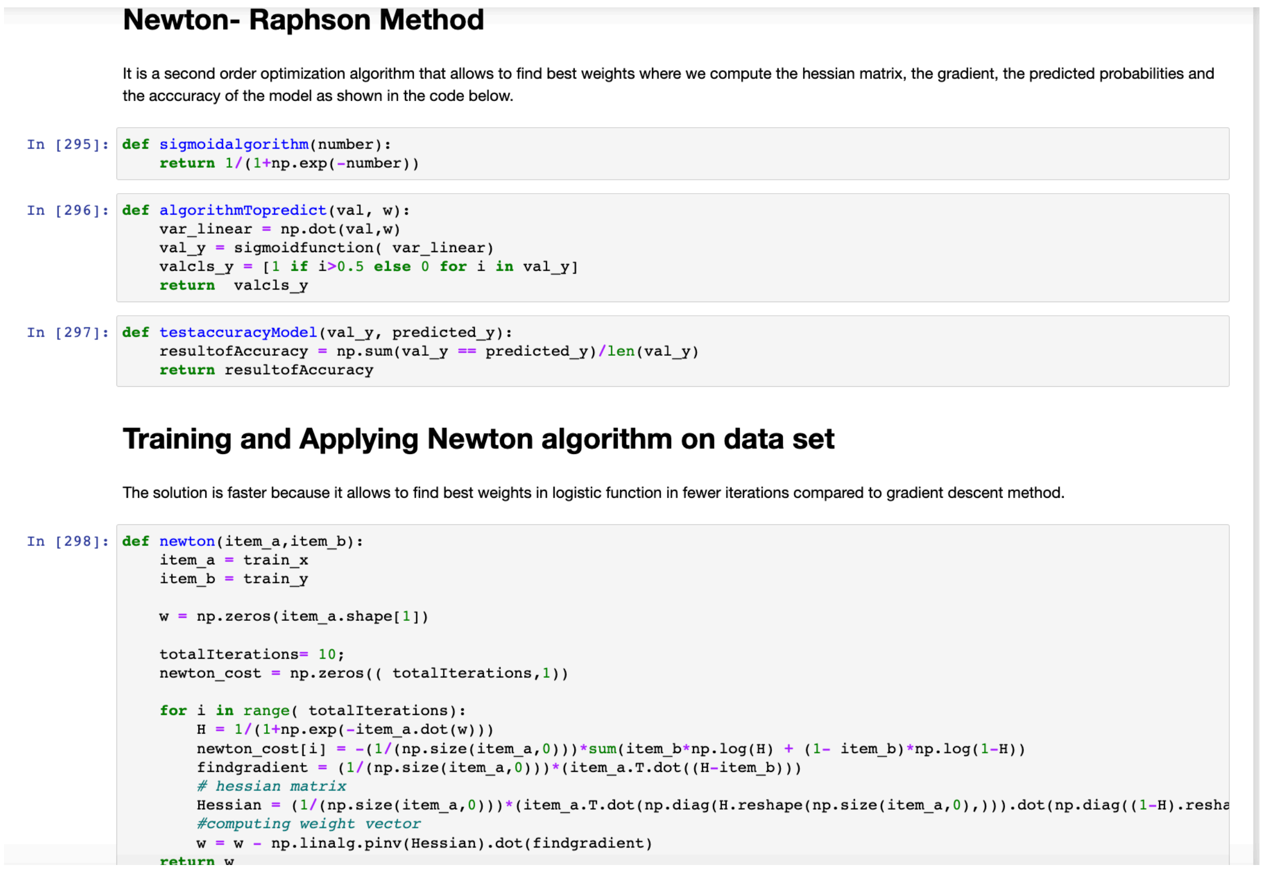


Splitting and Data Training

I split the data randomly into 80% training instances and 20% testing instances as directed:

```
In [49]: train_x, test_x,train_y, test_y = train_test_split(x, y, test_size=0.2)
```

Wrote the newton Raphson algorithm



Newton- Raphson Method

It is a second order optimization algorithm that allows to find best weights where we compute the hessian matrix, the gradient, the predicted probabilities and the acccuracy of the model as shown in the code below.

```
In [295]: def sigmoidalgorithm(number):
              return 1/(1+np.exp(-number))
```

```
In [296]: def algorithmTopredict(val, w):
              var_linear = np.dot(val,w)
              val_y = sigmoidfunction( var_linear)
              valcls_y = [1 if i>0.5 else 0 for i in val_y]
              return  valcls_y
```

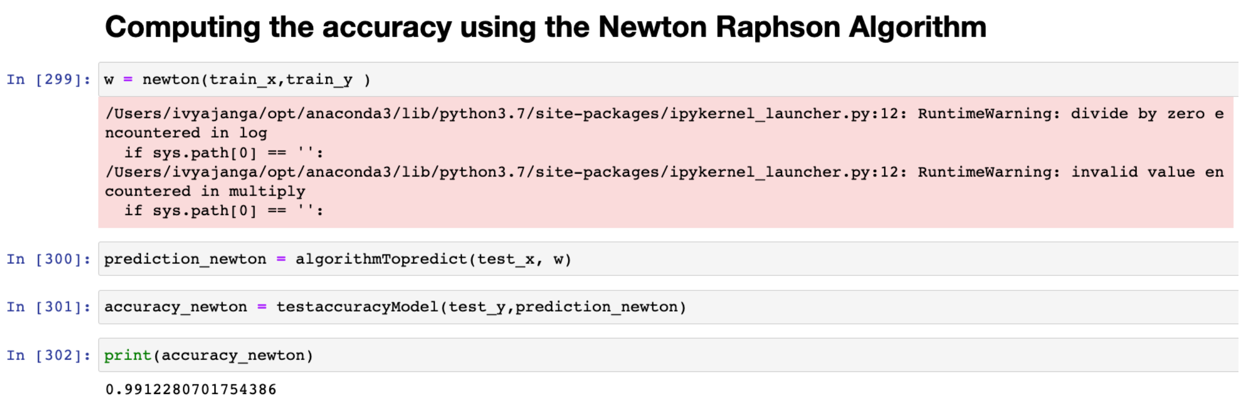
```
In [297]: def testaccuracyModel(val_y, predicted_y):
              resultofAccuracy = np.sum(val_y == predicted_y)/len(val_y)
              return resultofAccuracy
```

Training and Applying Newton algorithm on data set

The solution is faster because it allows to find best weights in logistic function in fewer iterations compared to gradient descent method.

```
In [298]: def newton(item_a,item_b):
              item_a = train_x
              item_b = train_y

              w = np.zeros(item_a.shape[1])

              totalIterations= 10;
              newton_cost = np.zeros(( totalIterations,1))

              for i in range( totalIterations):
                  H = 1/(1+np.exp(-item_a.dot(w)))
                  newton_cost[i] = -(1/(np.size(item_a,0)))*sum(item_b*np.log(H) + (1- item_b)*np.log(1-H))
                  findgradient = (1/(np.size(item_a,0)))*(item_a.T.dot((H-item_b)))
                  # hessian matrix
                  Hessian = (1/(np.size(item_a,0)))*(item_a.T.dot(np.diag(H.reshape(np.size(item_a,0),))).dot(np.diag((1-H).resha
                  #computing weight vector
                  w = w - np.linalg.pinv(Hessian).dot(findgradient)
              return w
```

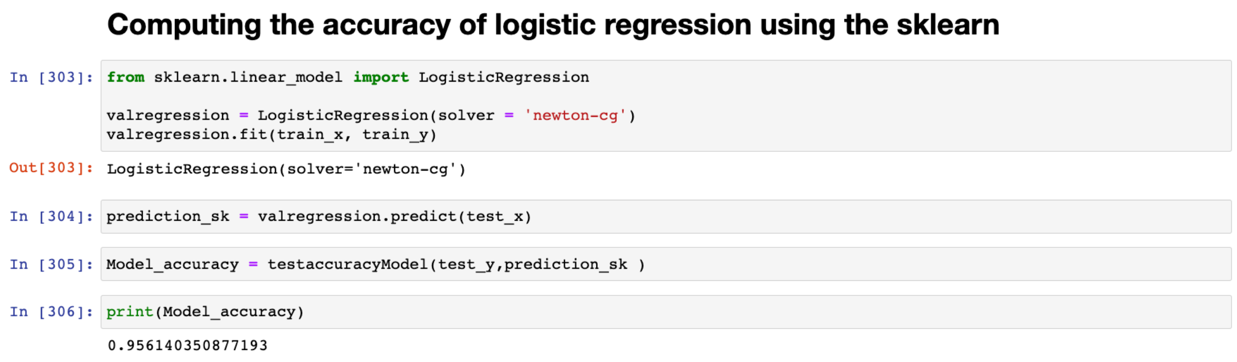
Performed the logistic regression with newton Raphson method on the data set and did 10 different trials to get an average accuracy of the model is 99.12%



Computing the accuracy using the Newton Raphson Algorithm

```
In [299]: w = newton(train_x,train_y )

          /Users/ivyajanga/opt/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:12: RuntimeWarning: divide by zero e
          ncountered in log
            if sys.path[0] == '':
          /Users/ivyajanga/opt/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:12: RuntimeWarning: invalid value en
          countered in multiply
            if sys.path[0] == '':

In [300]: prediction_newton = algorithmTopredict(test_x, w)

In [301]: accuracy_newton = testaccuracyModel(test_y,prediction_newton)

In [302]: print(accuracy_newton)

          0.9912280701754386
```

I performed the logistic regression using sklearn newton algorithm and found an accuracy of 95.61%



Computing the accuracy of logistic regression using the sklearn

```
In [303]: from sklearn.linear_model import LogisticRegression

          valregression = LogisticRegression(solver = 'newton-cg')
          valregression.fit(train_x, train_y)

Out[303]: LogisticRegression(solver='newton-cg')

In [304]: prediction_sk = valregression.predict(test_x)

In [305]: Model_accuracy = testaccuracyModel(test_y,prediction_sk )

In [306]: print(Model_accuracy)

          0.956140350877193
```

This indicates that the newton method gives a high accuracy level of detecting whether the cancerous tumor is either malignant or benign.