Homework I - Introduction to Machine Learning

Edwing Alexis Casillas Valencia
January 28, 2026

Abstract

In this assignment, the goal is the implementation of the linear regression, building the model from scratch in order to understand the whole process and calculation needed for it. Once the we have the basic implementation, we need to make use of the concepts of ”training data” and ”test data” on a binary classification.

PROBLEM 1

Given the insurance.csv please do the following:

(a) Extract information into a Dataframe, as pandas, to manipulate the csv data.
(b) Generate a simple Linear Regression (Which needs to be programmed from scratch using Jax https://github.com/jax-ml/jax) to generate an estimator of insurance cost.
(c) Explain this estimator.

Answer

During the development of the Linear Regression model, two files were generated, a python notebook for tests to see the behavior of each part and to understand what I was doing, and a functional python script where I can insert validation data related to the insurance.csv.

(a) I import the dataset as following

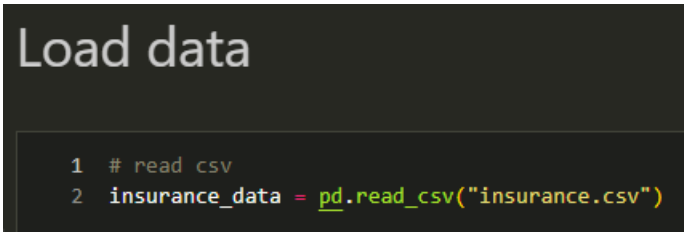

Fig. 1: Insurance data as data frame

Doing this statement directly on .CSV files allow us to manipulate the data to create the linear regression model, for example, apply binning to the data called ”sex” and ”smoker” to assign binary values instead the gender and yes/no. For this, I used 0 for female and no smoker, and 1 for male and smoker. Also, I used dummies to map the regions, using True or False.

```
   1  print(insurance_data)
 ✓  0.0s
```

```
      age  sex     bmi  children  smoker      charges  region_northeast  \
0      19    0  27.900         0       1  16884.92400             False
1      18    1  33.770         1       0   1725.55230             False
2      28    1  33.000         3       0   4449.46200             False
3      33    1  22.705         0       0  21984.47061             False
4      32    1  28.880         0       0   3866.85520             False
...   ...  ...     ...       ...     ...          ...               ...
1333   50    1  30.970         3       0  10600.54830             False
1334   18    0  31.920         0       0   2205.98080              True
1335   18    0  36.850         0       0   1629.83350             False
1336   21    0  25.800         0       0   2007.94500             False
1337   61    0  29.070         0       1  29141.36030             False

      region_northwest  region_southeast  region_southwest
0                False             False              True
1                False              True             False
2                False              True             False
3                 True             False             False
4                 True             False             False
...                ...               ...               ...
1333              True             False             False
1334             False             False             False
1335             False              True             False
1336             False             False              True
1337              True             False             False

[1338 rows x 10 columns]
```

Fig. 2: Insurance dataset

By doing this actions, now we can start to develop the linear model, because it requires numerical data instead of labels for the calculations in order to generate the estimation $\hat{y}$.

(b) Once we have the data prepared, we need to separate the target values from it, because it is our goal to estimate a value close to them, and also, we're going to use it to calculate the β vector.

Fig. 3: Insurance estimator

(c) From the previous image we can see that the implementation of the model is complete, the last line corresponds to the following equation

$$f(x) = \beta_0 + \sum_{j=1}^{p} X_j \beta_j \tag{1}$$

At first sight we notice that β_0 is not on represented on the script, but in fact, it is. As we saw, we can add it to the characteristic vector (augmented vector) to facilitate the calculations, starting from the first cost value on the hyperplane. Then we have a reduced formula as the following one

$$f(x) = \sum_{j=1}^{p} X_j^{aug} \beta_j \tag{2}$$

Which left us just to do the dot product of those vectors. @ is the equivalent of jnp.dot(), is just a newer notation. Doing that, we predict the insurance cost for each person on the dataset. Some of the data predicted is the following

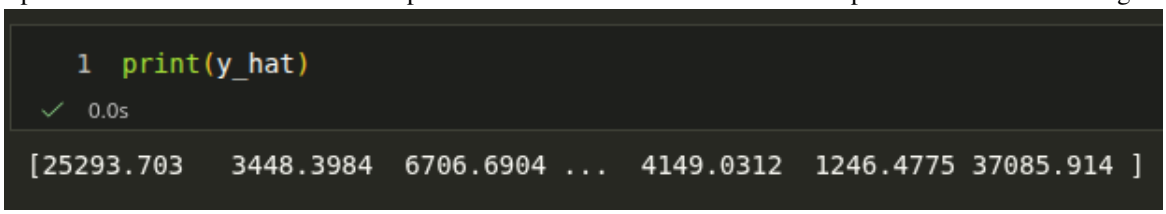

Fig. 4: Data predicted

How far my prediction is from the target value? We can calculate that difference for each prediction, but is going to take some time in it, then, we can calculate an average for the total data. As well, we can calculate the RMSE to se how big my error is in some of the values.

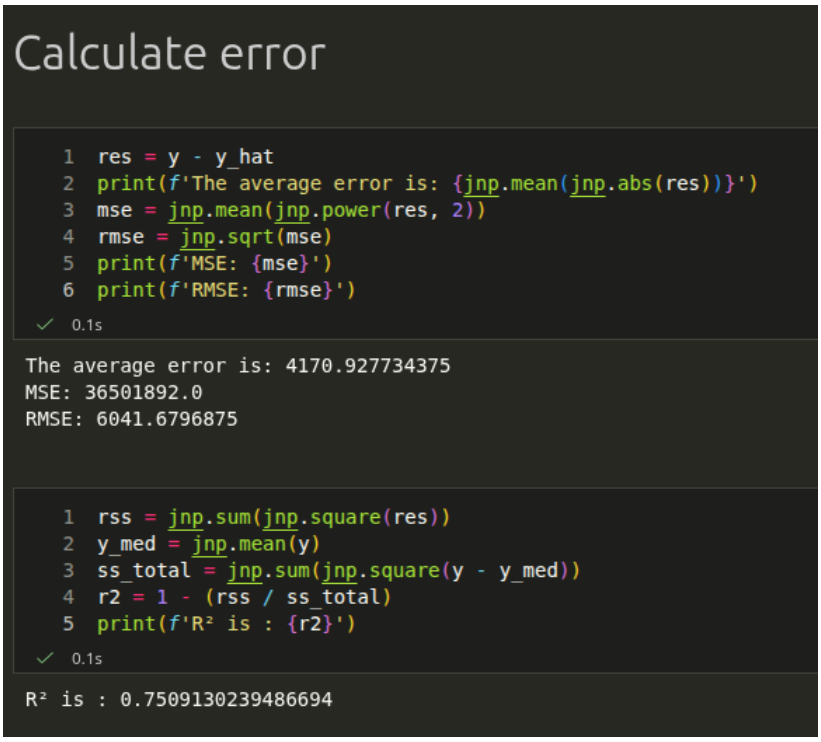

Fig. 5: Error and precision

We can notice that the average error is around 4200 dollars and the Root Square Error is around 6450 dollars, this is a pretty high error for an insurance estimator. We can see as well that we have a precision of 75%, in general, it could be considered good be if we talk about money, that extra dollars calculated on the prediction are not good.

We can improve these results adding another feature that is not on the main data. By now with the data we have, were assuming that the efect of being a smoker is the same for someone with high bmi as for someone else with low bmi, what does this means? it means that now we're considering a health risk on the insurance. In some cases, adding new variables based on our dataset could reduce the error, improving the predictions in general.

To do that, we need to add this line on the script at the moment to create the features of the data frame.

```
1       insurance_data['smoker_bmi'] = insurance_data['smoker'] * insurance_data['bmi']
2
```

```
1  print(y_hat)
```
✓ 0.0s

```
[22057.387   2122.1445  5773.035  ...  2178.7246  2688.2275 35491.77  ]
```

Calculate error

```
1  res = y - y_hat
2  print(f'The average error is: {jnp.mean(jnp.abs(res))}')
3  mse = jnp.mean(jnp.power(res, 2))
4  rmse = jnp.sqrt(mse)
5  print(f'MSE: {mse}')
6  print(f'RMSE: {rmse}')
```
✓ 0.0s

```
The average error is: 2898.849853515625
MSE: 23312318.0
RMSE: 4828.283203125
```

```
1  rss = jnp.sum(jnp.square(res))
2  y_med = jnp.mean(y)
3  ss_total = jnp.sum(jnp.square(y - y_med))
4  r2 = 1 - (rss / ss_total)
5  print(f'R² is : {r2}')
```
✓ 0.0s

```
R² is : 0.8409180045127869
```

Fig. 6: Prediction improved and error reduced

As we can see, the prediction of the insurance costs were reduced, the average error was reduced by 2000 dollars approximately, and the R^2 was improved from 75% almost 85%.

PROBLEM 2

Now using the Jax implementation, please use the data from

```
from sklearn.datasets import load_breast_cancer
data = load_breast_cancer()
```

to get the data for breast cancer with labels ”malign” and ”benign”. Then, split the data in the following percentages

- 90% training
- 10% testing

Then, report the precision and accuracy benchmarks on the testing dataset.

Answer

To make possible the use of the dataset, firs we have to convert it into a Dataframe as we did on the first part. For this, we need to import that data into the script, and then we can create our data frame with it. On the following image we see that part.

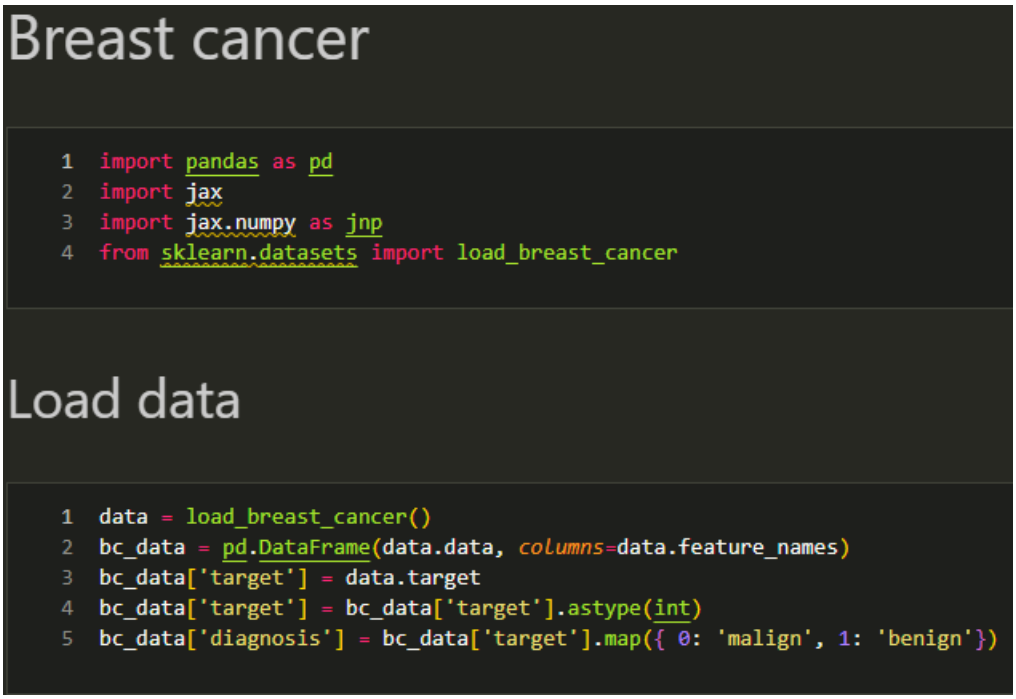

Fig. 7: Breast cancer data converted in our data frame

Notice that in this part I already applied binning to convert the 0's and 1's from the target into labels. It´s not necessary to do that heer, because we only need the numerical data for the calculations, and is in the final part where we convert the numerical data into labels. In the following images we can see the separation of the target data from the main dataframe.

```
X_wo_diagnosis = bc_data.drop(['target', 'diagnosis'], axis=1).astype(float).values
y_diagnosis = bc_data['target'].astype(float).values
```

Fig. 8: Target values for breast cancer

Before we split the data, we need to add a seed for randomness, because we want that every time we run the code, we got the same random values, instead using the RTC from the computer, letting randomness be deterministic, this is because we use a PRNG generator for it.

I'm using 73 as the seed for it's unique mathematical properties, but we can choose any number, like 42, which is the answer to the sense of life. Those properties, known as the Sheldon Cooper theorem, are the following:

- It is the 21° prime number
- Its reverse, 37, is the 12° prime number
- The product of it's digits (7 * 3) equals 21, another prime number.

Once we have the seed, we shuffle the data and calculate the 90% of it to use it as training data, and the rest for test. We can see the process on the following image.

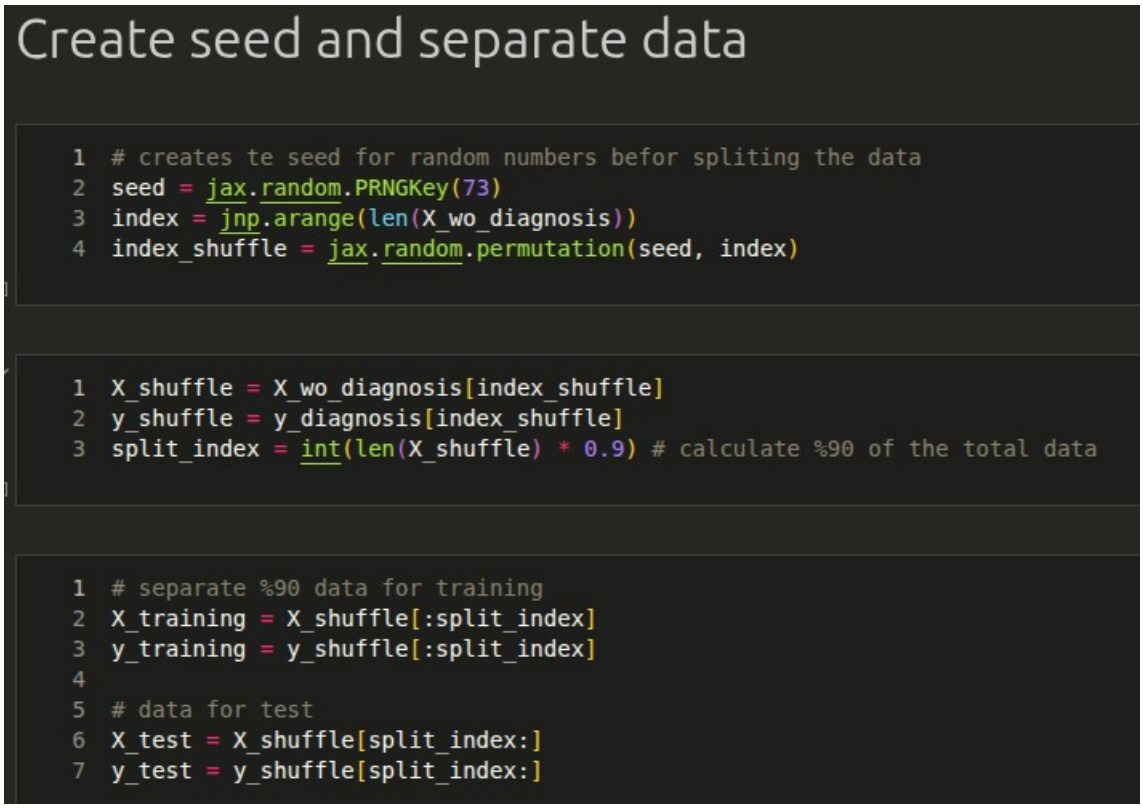

Fig. 9: Seed creation and split data

With this part of the data, we can now proceed to calculate β and $\hat{y}$ the same way we done on the insurance estimator, but using X training and y training for it. Once we have those calculations, we can estimate the output, but linear regression give us numerical data as output instead of categorical, so we need to apply some criteria on it, converting it into a binary classification, where 0 is malign and 1 is benign.

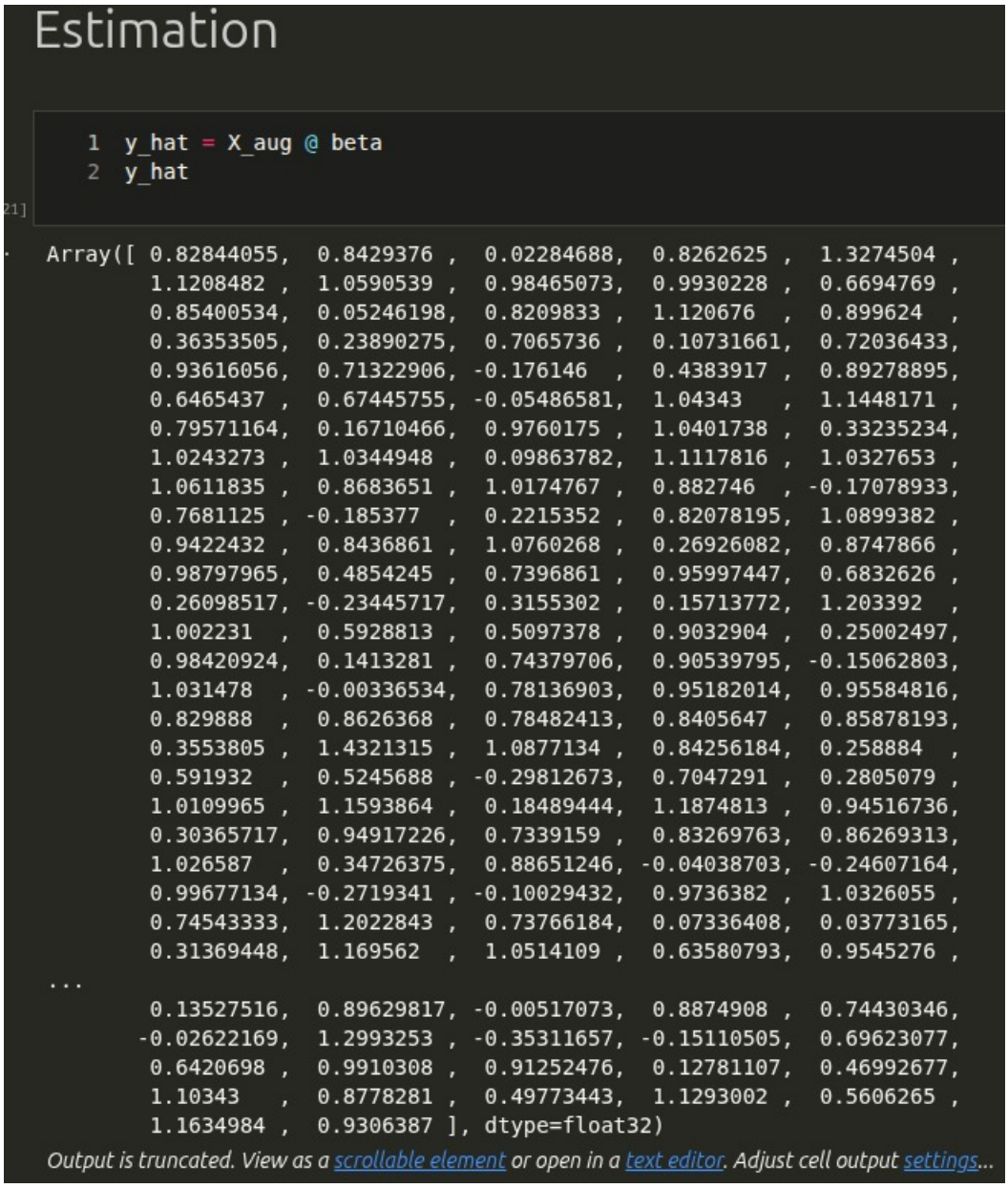

Fig. 10: Numerical estimations

```
1  y_labeled = ["benign" if val > 0.5 else "malign" for val in y_hat]
2  y_labeled
```

```
['benign',
 'benign',
 'malign',
 'benign',
 'benign',
 'benign',
 'benign',
 'benign',
 'benign',
 'benign',
 'benign',
 'malign',
 'benign',
 'benign',
 'benign',
 'malign',
 'malign',
 'benign',
 'malign',
 'benign',
 'benign',
 'benign',
 'malign',
 'malign',
 'benign',
...
 'malign',
 'benign',
 'benign',
 'benign',
 'benign']
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

Fig. 11: Label classification

To get a global overview of the results, we can calculate the average error, MSE, RMSE and R^2 the same way we done before.

```
The average error is: 0.17772424221038818
MSE: 0.05146179720759392
RMSE: 0.25877130031585693
R² is : 0.7769997119903564
```

Fig. 12: Training error

Now, we use the test data to calculate the prediction and see if the behavior is similar to the training data.

```
1  y_hat_test = X_aug_test @ beta
2  y_hat_test
```

```
Array([ 0.42507637,  0.4547841 , -0.17375398,  0.911738  ,  0.41667664,
        0.92332476,  0.86908555,  0.99157953,  0.04441762,  0.94362414,
       -0.04768378,  0.9862835 ,  0.14328593,  0.98563373,  0.8104745 ,
        0.8740094 ,  0.7020477 ,  0.56655097,  0.26217085,  0.06126428,
        1.0105889 ,  0.7317765 ,  0.6840583 ,  0.6268207 ,  0.7645969 ,
       -0.00717929,  0.84736395,  0.72186697, -0.4452382 ,  0.57377005,
        0.22694808,  0.3697394 ,  0.65685666,  0.8383722 ,  0.05384582,
       -0.04821756,  1.0093751 ,  0.87104404,  0.92159486,  0.14291012,
        0.02481177, -0.19077939,  1.1473175 ,  0.8956164 ,  0.15830249,
        0.42663294,  0.7364408 ,  0.83262134,  0.3565853 ,  0.7697538 ,
       -0.20547923,  0.19650167,  0.9753064 ,  1.0817051 ,  1.111849  ,
        1.3021945 ,  0.3014489 ], dtype=float32)
```

```
1  y_labeled_test = ["benign" if val > 0.5 else "malign" for val in y_hat_test]
2  y_labeled_test
```

```
['malign',
 'malign',
 'malign',
 'benign',
 'malign',
 'benign',
 'benign',
 'benign',
 'malign',
 'benign',
 'malign',
 'benign',
 'malign',
 'benign',
 'benign',
 'benign',
 'benign',
 'benign',
 'malign',
 'malign',
 'benign',
 'benign',
 'benign',
 'benign',
```

Fig. 13: Test results

Test error

```
1  res_test = y_test - y_hat_test
2  print(f'The average error is: {jnp.mean(jnp.abs(res))}')
3  mse_test = jnp.mean(jnp.power(res, 2))
4  rmse_test = jnp.sqrt(mse_test)
5  print(f'MSE: {mse_test}')
6  print(f'RMSE: {rmse_test}')
7  rss_test = jnp.sum(jnp.square(res_test))
8  y_med_test = jnp.mean(y_test)
9  ss_total_test = jnp.sum(jnp.square(y_test - y_med_test))
10 r2_test = 1 - (rss_test / ss_total_test)
11 print(f'R² is : {r2}')
```

```
The average error is: 0.17772424221038818
MSE: 0.05146179720759392
RMSE: 0.2268519252538681
R² is : 0.7769997119903564
```

Fig. 14: Test errors

As we can see, the results are almost the same for the estimations, to complete the assignment and get a total comprehension on the topic, we need to calculate the precision and accuracy. What are they?, the accuracy tell us how good is an estimator in general, and the precision is the ratio of correct positive predictions to the overall number of positive predictions. One way to do that is using the confusion matrix, which summarize the True and False negatives and True and False positives of our results. Instead of calculate the whole matrix, we can just calculate the information we need directly, as it is on the image bellow.

Confusion matrix

```
1  # aplying binning, where 1 is benign and 0 malign
2  y_pred = jnp.where(y_hat_test > 0.5, 1, 0)
3
4  # true positive
5  tp = jnp.sum((y_pred == 1) & (y_test == 1))
6
7  #true negative
8  tn = jnp.sum((y_pred == 0) & (y_test == 0))
9
10 #false positive
11 fp = jnp.sum((y_pred == 1) & (y_test == 0))
12
13 #false negative
14 fn = jnp.sum((y_pred == 0) & (y_test == 1))
```

✓ 0.0s

Fig. 15: Confusion matrix data

The formulas I used for precision and accuracy are the following.

$$precision = TP/(TP + FP) \tag{3}$$

$$accuracy = (TP + TN)/(TP + TF + FP + FN) \tag{4}$$

```
Precision: 0.9090908765792847
Accuracy: 0.9473684430122375
```

Fig. 16: Precision and Accuracy

The precision and accuracy are 90% and 94%, which is pretty good, and it means that almost all the time, we're going to have an estimation almost equal to the target. Surprisingly, the linear regression model worked really good for a non linear problem.