Understanding Logistic Regression with Python: Practical Guide 1

Author, Mayank Tripathi

A Data Science Foundation White Paper

April 2020

---

www.datascience.foundation

Data Science Foundation
Data Science Foundation, Atlantic Business Centre, Atlantic Street, Altrincham, WA14 5NQ
Tel: 0161 926 3641 Email: admin@datascience.foundation Web: www.datascience.foundation
Registered in England and Wales 4th June 2015, Registered Number 9624670

Introduction

Classification techniques are an essential part of machine learning and data mining applications. Approximately 60 to 80 per-cent of the challenges facing a Data Scientist are classification problems.

"In machine learning and statistics, classification is the problem of identifying to which of a set of categories (sub-populations) a new observation belongs, on the basis of a training set of data containing observations (or instances) whose category membership is known."

https://en.wikipedia.org/wiki/Statistical_classification

There are lots of classification solutions available, but logistic regression is a common and is a useful regression method for solving binary classification problems. It is easy to implement and can be used as the baseline for any binary classification problem. Let's start to understand logistic regression with Python with the aid of an example.

Don't confuse **Logistic Regression** with **Linear Regression**, which we will come to later.

"In linear regression, the outcome (dependent variable) is continuous. It can have any one of an infinite number of possible values. In logistic regression, the outcome (dependent variable) has only a limited number of possible values. Logistic regression is used when the response variable is categorical in nature."

https://stackoverflow.com/questions/12146914/what-is-the-difference-between-linear-regression-and-logistic-regression

In general, binary logistic regression describes the relationship between the dependent binary variable and one or more independent variables.

In statistics, the logistic model (or logit model) is used to model the probability of a certain class or event existing such as pass/fail, win/lose, alive/dead or healthy/sick. This can be extended to model several classes of events such as determining whether an image contains a cat, dog, lion, etc. Each object being detected in the image would be assigned a probability

between 0 and 1 and the sum adding to one. Taken from Wikipedia.

For further reading on this, here are two sites that I often refer to:

https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html
https://www.statisticssolutions.com/what-is-logistic-regression/

I find this YouTube channel very informative, **3Blue1Brown.**They have a talent for explaining Statistics clearly.

Back to the topic in hand, Logistic Regression, here where the target or dependent variable has two possible outcomes as shown below, which are binary in nature. Thus this type of classification is known as Binary Logistic Regression.

'1' for True / Success / Yes

or

'0' for False / Failure / No

You might be wondering why we started with Logistic Regression and then started taking about Binary Logistic Regression. So, let's investigate this point.

Types of Logistic Regression

Let's see how many types of Logistic Regression there are:

1. **Logistic Regression**

   The categorical response has only two 2 possible outcomes.

   Example:

   - If your Email is Spam or Non-Spam
   - In-case if you have worked on Datasets such as Titanic, then Survival status as Yes or No.
   - Whether the user will click on a given advertisement link or not.
   - Some other dataset like Diabetes prediction; cancer detection.
   - If a given customer will purchase a product (specific or a particular type of product) or not.
   - If a given customer will retain or they will churn / leave.
2. **Multinomial Logistic Regression**

Three or more categories without ordering.

Example:

- Predicting which food is preferred (Veg, Non-Veg, Vegan)
- IRIS dataset a very famous example of multi-class classification.
- Entering high school students make program choices among general programs, vocational programs and academic programs.
- Other examples are classifying written files; article / blog / document.

3. **Ordinal Logistic Regression**

   Three or more categories with ordering.

   Example:

   - Movie rating from 1 to 5

Binary Logistic Regression.

Let's apply logistic regression in Python using two practical examples. The first is a simple introduction and the second using a Kaggle dataset

Note: Here that the intention is to understand Logistic Regression, so I will not spend time on data cleaning or accuracy score.

These examples will incorporate the following steps: (steps may vary from person to person or example to example).

Step 1: Gather the data / dataset.

Step 2: Import the required Python packages (as we are using Python here).

Step 3: Build a data frame

Step 4: Create the Model in Python (In this example Logistic Regression)

Step 5: Predict using Test Dataset and Check the score

Step 6: Prediction with a New Set of Data and evaluate the accuracy

Practical 1

Step 1: Gather your data / dataset.

The task in our first practical example is to build a simple logistic regression model to determine whether a candidate's application will get accepted and approved for admission into Harvard university or if it should be rejected.

So, the two possible outcomes are:

Application Accepted and Approved for Admission, which can be represented by the value of '1'

and

Application Rejected, which can be represented by the value of '0'.

The input data may be as shown below with Feature or Independent Variables; GMAT score, GPA and Years of work experience, and the Target or Dependent Variable to represent the application status of a candidate.

GMAT Score	GPA	Years of Experience	STATUS
640	3	1	Rejected
650	3.7	6	Approved
580	2.7	2	Rejected
620	3.3	2	Approved
…	…	…	…

Please refer to the link provided at the end of this article to obtain the Jupyter notebook for reference.

Note: This dataset has been generated manually for the purpose of this example. As such this dataset contains just a few observations, in actual practice, we would need a larger sample size to get more accurate results.

```
import pandas as pd
application = {'APPLICATION_NBR': ['oJCvIoTB', 'q4ICdH0f', '34EK9wXH', 'T83EGu9A',
'tpGQZJDm', 'zHbdj4hS', 'AodXvUsc', 'QE3wAy9T', 'gBa6psw3', 'X7H38FHe', 'iwVeBGu3',
'AMc5dykm', 'cAJ0MbEh', 'sXoTwUHx', 'Fsv0RQA3', 'mJXu1o6R', 'xL3EDeGa', 'c0Ed7hbm',
'LInGqJ59', 'lSKXvwGK', 'xTlGMUG2', 'DvZ2TcBB', 'm67QrLB0', '2YuxgSLF', 'jUHD7lQg',
'Df38sD1D', 'uBewE4el', 'yQduYqfd', 'xs1TTyDO', 'HNHSkiLV', 'DSvTzs8J', 'plboghkI',
'wL2gVzRP', 'gaR3nmER', 'kfZWQwyh', 'kcDoPfu1', 'GHDbDtEE', 'sZv1NmY5', '68lmOgBL',
'jGjMFKRV', 'GHDbDtFE', 'sZ31NmY5', '682mOgbL', 'jGjMFLRV'],
        'GMAT_SCORE':[640, 650, 580, 620, 780, 750, 690, 710, 680, 730, 690, 720, 740,
690, 610, 690, 710, 680, 770, 610, 580, 650, 540, 590, 620, 600, 550, 550, 570, 670, 660, 580,
650, 660, 640, 620, 660, 660, 680, 650, 670, 580, 590, 690],
        'GPA': [3, 3.7, 2.7, 3.3, 4, 3.9, 3.3, 3.7, 3.9, 3.7, 2.3, 3.3, 3.3, 1.7, 2.7, 3.7, 3.7, 3.3, 3.3,
3, 2.7, 3.7, 2.7, 2.3, 3.3, 2, 2.3, 2.7, 3, 3.3, 3.7, 2.3, 3.7, 3.3, 3, 2.7, 4, 3.3, 3.3, 2.3, 2.7, 3.3,
1.7, 3.7],
        'EXPERIENCE': [1, 6, 2, 2, 3, 4, 3, 5, 4, 6, 1, 4, 5, 1, 3, 5, 6, 4, 3, 1, 4, 6, 2, 3, 2, 1, 4, 1, 2,
6, 4, 2, 6, 5, 1, 2, 4, 6, 5, 1, 2, 1, 4, 5],
        'STATUS': ['Rejected', 'Approved', 'Rejected', 'Approved', 'Approved', 'Approved',
'Rejected', 'Approved', 'Rejected', 'Approved', 'Rejected', 'Approved', 'Approved', 'Rejected',
'Rejected', 'Approved', 'Approved', 'Rejected', 'Approved', 'Rejected', 'Rejected', 'Approved',
'Rejected', 'Rejected', 'Approved', 'Rejected', 'Rejected', 'Rejected', 'Rejected', 'Approved',
'Approved', 'Rejected', 'Approved', 'Approved', 'Rejected', 'Rejected', 'Approved', 'Approved',
'Approved', 'Rejected', 'Rejected', 'Rejected', 'Rejected', 'Approved']
        }
```

Step 2: Import the required Python packages

Here will install all the required packages. Also note we have already imported the pandas' package, still I am importing it again just to have all imports together.

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
import seaborn as sn
import matplotlib.pyplot as plt
```

Step 3: Build a data frame

In this step we need to convert the dataset into Python Data-Frame.

Alternatively, we could import the data into Python from an external file, using read_csv; read_json etc.

```
df = pd.DataFrame(application,columns= ['APPLICATION_NBR', 'GMAT_SCORE',
'GPA','EXPERIENCE','STATUS'])

df.head()
```

	APPLICATION_NBR	GMAT_SCORE	GPA	EXPERIENCE	STATUS
0	oJCvIoTB	640	3.0	1	Rejected
1	q4ICdH0f	650	3.7	6	Approved
2	34EK9wXH	580	2.7	2	Rejected
3	T83EGu9A	620	3.3	2	Approved
4	tpGQZJDm	780	4.0	3	Approved

When looking at the result, you will see that the Target column “STATUS” is a String value as Approved / Reject, we must set it into a Binary Format such as 1 / 0. This I made intentionally just to show you how we can change the value. In a real world dataset we will not have straight forward data.

```
df["STATUS"].replace({"Rejected":"0", "Approved":"1"}, inplace=True)
df.head()
```

```
df.head()
```

	APPLICATION_NBR	GMAT_SCORE	GPA	EXPERIENCE	STATUS
0	oJCvIoTB	640	3.0	1	0
1	q4ICdH0f	650	3.7	6	1
2	34EK9wXH	580	2.7	2	0
3	T83EGu9A	620	3.3	2	1
4	tpGQZJDm	780	4.0	3	1

In the above screenshot check the red-encircled values which have now been changed.

Step 4: Create the Model in Python

Now, to work on the model, first we must select / choose the independent variables or features and represent them as X (it’s not necessary to call it X, but it’s a general recommendation) and represent the dependent variable or target as y.

If you have noticed, I have used X (in upper case) and y (in lower case).

Also, an Application Number is not used, as it has no importance in deciding if a candidate

should be selected or rejected. Thus, it will be ignored. As the dataset is small, we are able to identify who the candidates are. Whereas in a real-world scenario we would have multiple columns, there we would use Correlation, and based on this would select the necessary features.

```
X = df[['GMAT_SCORE', 'GPA','EXPERIENCE']]
y = df['STATUS']
display(X.shape, y.shape)
```

```
[11] X = df[['GMAT_SCORE', 'GPA','EXPERIENCE']]
     y = df['STATUS']
     display(X.shape, y.shape)

     (44, 3)
     (44,)
```

With the shape method we can see the count of rows and columns in the dataset. So now in X we have 44 rows with 3 columns, and in y we have 44 rows with the self-column.

Once we have the features and target, will split the data into train and test using train_test_split.

For example, you can set the test size to 0.25, and therefore the model testing will be based on 25% of the dataset, while the model training will be based on 75% of the dataset.

```
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.25,random_state=0)
display(X_train.shape, y_train.shape, X_test.shape, y_test.shape)
```

```
display(X_train.shape, y_train.shape, X_test.shape, y_test.shape)
```

```
(33, 3)
(33,)
(11, 3)
(11,)
```

Now to apply the Logistic Regression. If you remember we have already imported all the required libraries, and the one below is for Logistic Regression.

```
from sklearn.linear_model import LogisticRegression
```

We will first create an instance of Logistic Regression as lr. And then use the fit method i.e. Fit the model according to the given training data. LogisticRegression() has various parameters and various methods to use. For now, will focus on the generic one.

```
lr= LogisticRegression()
lr.fit(X_train,y_train)
```

Step 5: Predict using Test Dataset and Check the score.

```
y_pred=lr.predict(X_test)
```

Once we have fit the training mode, it's time to Predict based on test dataset which we have in X test and then will match with the y_test to which we already have the answers. You remember we have taken only ~75% of data i.e. 33 records out of 44 for training our model and left ~25% i.e. 11 records out of 44 for test our model.

```
[17] y_pred=lr.predict(X_test)
```

```
[18] y_pred
```

```
array(['1', '1', '0', '1', '0', '0', '0', '1', '1', '0', '1'],
      dtype=object)
```

Here is the result of the prediction which our model has generated. As the dataset is small, we can manually compare the results with the **y_test** which we have. The results.

```
[20] y_test
```

```
30    1
37    1
27    0
4     1
10    0
25    0
28    0
11    1
38    1
31    0
29    1
Name: STATUS, dtype: object
```

We can get the score using the Confusion Matrix as below. This tells us how accurate our model is. It is in fact 100% accurate.

```
cnf_matrix = metrics.confusion_matrix(y_test, y_pred)
cnf_matrix
```

Let's see the same Metrix data in a Graphical view using heatmap.

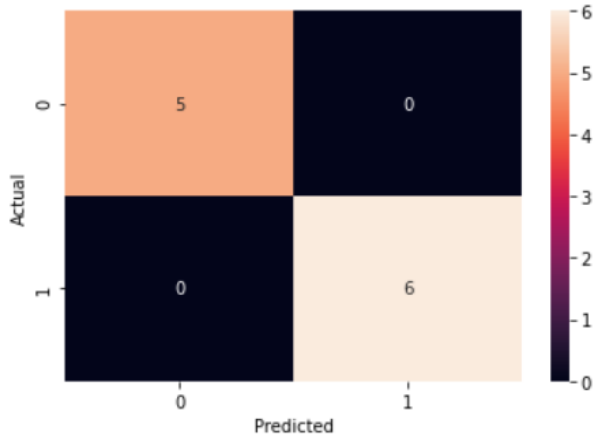
```
cnf_matrix = pd.crosstab(y_test, y_pred, rownames=['Actual'], colnames=['Predicted'])
sn.heatmap(cnf_matrix, annot=True)
```

```
[24] cnf_matrix = metrics.confusion_matrix(y_test, y_pred)
     cnf_matrix

     array([[5, 0],
            [0, 6]])
```

```
[25] cnf_matrix = pd.crosstab(y_test, y_pred, rownames=['Actual'], colnames=['Predicted'])
     sn.heatmap(cnf_matrix, annot=True)

     <matplotlib.axes._subplots.AxesSubplot at 0x7fcf0591b240>
```



As can be observed from the above matrix

TP = True Positives = 5

TN = True Negatives = 6

FP = False Positives = 0 (in Black Block)

FN = False Negatives = 0 (in Black Block)

You can then get the Accuracy using:

Accuracy = (TP+TN)/Total = (5+6)/11 = 1

The accuracy is therefore 100% for the test set.

Another way to check the accuracy is shown below, using accuracy_score() method.

```
accuracyScore = metrics.accuracy_score(y_test, y_pred)
print('Accuracy Score : ',accuracyScore)
print('Accuracy In Percentage : ', int(accuracyScore*100), '%')
```

```
[29] accuracyScore = metrics.accuracy_score(y_test, y_pred)
     print('Accuracy Score : ',accuracyScore)
     print('Accuracy In Percentage : ', int(accuracyScore*100), '%')

     Accuracy Score :  1.0
     Accuracy In Percentage :  100 %
```

Now the final step, is to test with unseen data.

Step 6: Prediction with a New Set of Data

For this step let's imagine that we have a GMAT Score as 680 with 6 years of experience and a GPA of 3.3. What do you say, will the candidate be approved or rejected?

I believe they should be approved. Let's find out.

```
new_application = {'GMAT_SCORE': [680],
                   'GPA': [3.3],
                   'EXPERIENCE': [6]
                   }

df2 = pd.DataFrame(new_application,columns= ['GMAT_SCORE', 'GPA','EXPERIENCE'])
y_pred=logistic_regression.predict(df2)
print(y_pred)
```

```
[30] new_application = {'GMAT_SCORE': [680],
                        'GPA': [3.3],
                        'EXPERIENCE': [6]
                        }

     df2 = pd.DataFrame(new_application,columns= ['GMAT_SCORE', 'GPA','EXPERIENCE'])
     y_pred=logistic_regression.predict(df2)
     print(y_pred)

     ['1']
```

Yes, the y_pred value generated is 1, meaning that the candidate is travelling to Cambridge.

The model shown can be applied to new or unseen data but remember to have the new dataset in same format on which the model was trained.

Congrats!!!

With this you have completed your first Logistic Regression.

Practical 2

Here, we are going to predict diabetes using Logistic Regression Classifier.

Let's first load the required Pima Indian Diabetes dataset using the pandas' read CSV function. You can download data from the following link: https://www.kaggle.com/uciml/pima-indians-diabetes-database

With this I can say that we have successfully completed Step 1 which is to Gather the data / dataset.

Here I am using the Kaggle site to write the code, if you wish would continue at Kaggle or you could download the dataset on to your local computer.

Moving on to Step 2, Import the required Python packages.

Import Libaries

[n [8]:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
import seaborn as sn
import matplotlib.pyplot as plt
```

Step 3: Build a data frame

Load Data

In [16]:

```
col_names = ['pregnant', 'glucose', 'bp', 'skin', 'insulin', 'bmi', 'pedigree', 'age', 'label']

# load dataset
pima = pd.read_csv("/kaggle/input/pima-indians-diabetes-database/diabetes.csv", names=col_name
s)
```


Data Science Foundation, Atlantic Business Centre, Atlantic Street, Altrincham, WA14 5NQ
Tel: 0161 926 3641 Email: admin@datascience.foundation Web: www.datascience.foundation
Registered in England and Wales 4th June 2015, Registered Number 9624670

In [17]:
```
pima.head()
```

Out[17]:

	pregnant	glucose	bp	skin	insulin	bmi	pedigree	age	label
0	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
1	6	148	72	35	0	33.6	0.627	50	1
2	1	85	66	29	0	26.6	0.351	31	0
3	8	183	64	0	0	23.3	0.672	32	1
4	1	89	66	23	94	28.1	0.167	21	0

When you load the data and view it using the head() method, we notice that the dataset has an additional row (row with index as 0). We have to skip this, and to do that we need to use the skiprows parameter in read_csv() as shown below.

In [18]:
```
# load dataset
pima = pd.read_csv("/kaggle/input/pima-indians-diabetes-database/diabetes.csv", skiprows=1, nam
es=col_names)
```

In [19]:
```
pima.head()
```

Out[19]:

	pregnant	glucose	bp	skin	insulin	bmi	pedigree	age	label
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

Now data looks good, but it needs to be checked and cleaned.

- Review the data to see how many rows and columns the dataset has.
- Does it have any null values?
- Do any of the columns have incorrect data formats etc. All of these validations are

required.

This data is clean, so we will move on.

Once we have all our data, it's time to select the right features for model.

Here, we need to pide the given columns into two types of Features (X) and Target (y) remember this from above.

As we have X and y, we can split the dataset into Train and Test. This is to evaluate model performance, piding the dataset into a training set and a test set is a good strategy.

Let's split dataset by using function train_test_split(). You need to pass 3 parameters features, target, and test_set size. Additionally, you can use random_state to select records randomly.

Next comes training the Model. Here will use the same method we used in practical 1.

Predict the model using X_test, and will validate it using y_test.

Predict with Test Dataset

In [27]:
```
y_pred=lr.predict(X_test)
```

In [28]:
```
y_pred
```

Out[28]:
```
array([1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0,
       0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1,
       1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1,
       1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1,
       0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
       0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0])
```

Model Evaluation using Confusion Matrix

A confusion matrix is a table that is used to evaluate the performance of a classification model. You can also visualize the performance of an algorithm. The fundamentals of a confusion matrix are the number of correct and incorrect predictions summed up class-wise.

Model Evaluation

Confusion Matrix

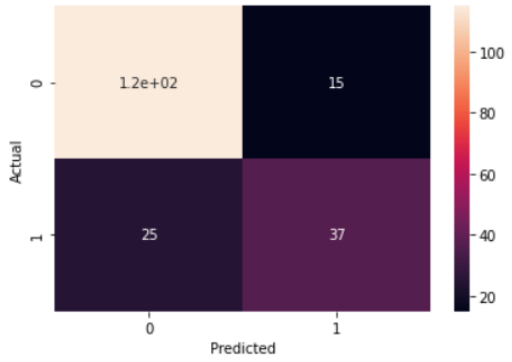
In [29]:
```
cnf_matrix = metrics.confusion_matrix(y_test, y_pred)
cnf_matrix
```

Out[29]:
```
array([[115,  15],
       [ 25,  37]])
```


Data Science Foundation, Atlantic Business Centre, Atlantic Street, Altrincham, WA14 5NQ
Tel: 0161 926 3641 Email: admin@datascience.foundation Web: www.datascience.foundation
Registered in England and Wales 4th June 2015, Registered Number 9624670

In [31]:
```
cnf_matrix = pd.crosstab(y_test, y_pred, rownames=['Actual'], colnames=['Predicted'])
sn.heatmap(cnf_matrix, annot=True)
```

Out[31]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f0664595c50>
```



Here, you can see the confusion matrix in the form of an array object or graphical view.

Diagonal values represent accurate predictions, while non-diagonal elements are inaccurate predictions.

In the output, 115 and 37 are actual predictions, and 25 and 15 are incorrect predictions.

TP = True Positives = 115

TN = True Negatives = 37

FP = False Positives = 25 (in Black/Dark Gray Block)

FN = False Negatives = 15 (in Black/Dark Gray Block)

You can then get the Accuracy using:

Accuracy = (TP+TN)/Total = (115+37)/192 = 0.79:

The accuracy is therefore **79%** for the test set.

The same result can also be achieved using the accuracy_score() method shown below.

Accuracy Score

In [32]:
```
accuracyScore = metrics.accuracy_score(y_test, y_pred)
print('Accuracy Score : ',accuracyScore)
print('Accuracy In Percentage : ', int(accuracyScore*100), '%')
```

```
Accuracy Score :  0.7916666666666666
Accuracy In Percentage :  79 %
```

Congrats again!!!

You are now a champion of Logistic Regression.

Please note, that this guide has been created as a first introduction to the subject of Logistic Regression and does not consider the many various challenges a data scientist would be expected to deal with. It is in my opinion a good kick start to the topic, and once started you can keep on digging deeper and deeper. So please don't stop here go and learn more about the subject of Logistic Regression and how it can be put into practice.

As promised, you can download the full code using these links

Practical 1 : https://colab.research.google.com/drive/1VC7kP7AeiLt4sA9WFijq758QCe7MkGiF

Practical 2 : https://www.kaggle.com/dskagglemt/pima-indians-diabetes-database-version1

References:

- https://en.wikipedia.org/wiki/Logistic_regression
- https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html
- https://www.statisticssolutions.com/what-is-logistic-regression/
- https://pandas.pydata.org/pandas-docs/version/0.17.0/generated/pandas.crosstab.html

About the Data Science Foundation

The Data Science Foundation is a professional body representing the interests of the Data Science Industry. Its membership consists of suppliers who offer a range of big data analytical and technical services and companies and individuals with an interest in the commercial advantages that can be gained from big data. The organisation aims to raise the profile of this developing industry, to educate people about the benefits of knowledge based decision making and to encourage firms to start using big data techniques.

Contact Data Science Foundation

Email:admin@datascience.foundation
Telephone: 0161 926 3641
Atlantic Business Centre
Atlantic Street
Altrincham
WA14 5NQ
web: www.datascience.foundation