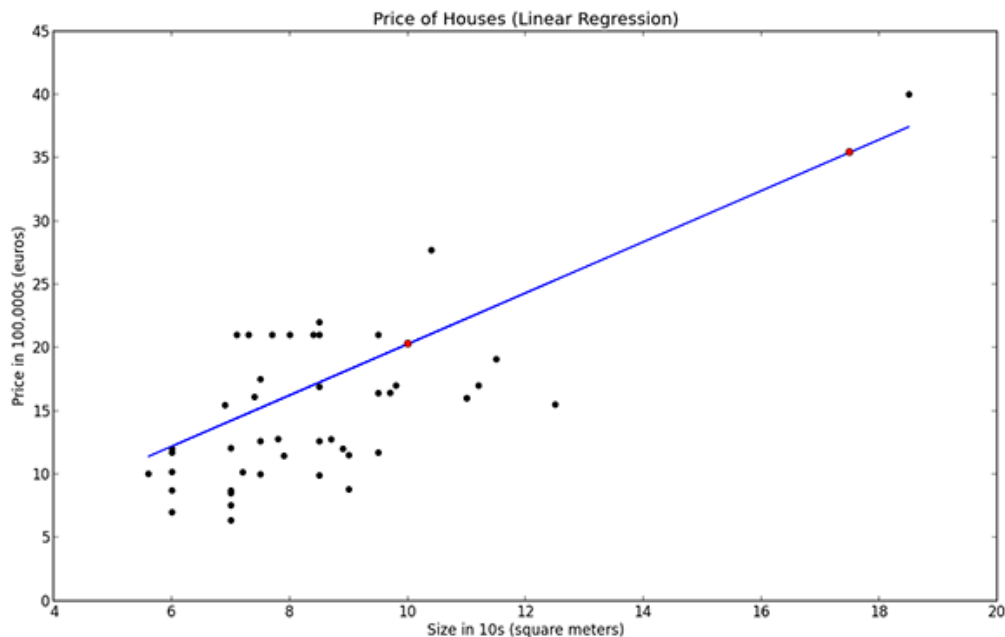


Big Data Examiner

[MENU](#)

How To Run Linear Regression In Python SciKit-Learn



You know that linear regression is a popular technique and you might as well see the mathematical equation of [linear regression](#). But do you know how to implement a linear regression in Python?? if so don't read this post because this post is all about implementing linear regression in Python. There are several ways in which you can do that, you can do linear regression using numpy, scipy, stats model and scikit learn. But in this post I am going to use scikit learn to perform linear regression.

[Scikit-learn](#) is a powerful Python module for machine learning. It contains function for regression, classification, clustering, model selection and dimensionality reduction. Today, I will explore the `sklearn.linear_model` [module](#) which contains “*methods intended for regression in which the target value is expected to be a linear combination of the input variables*”.

In this post, I will use [Boston Housing data set](#), the data set contains information about the housing values in suburbs of Boston. This dataset was originally taken from the StatLib library which is maintained at Carnegie Mellon University and is now available on the [UCI Machine Learning Repository](#). UCI machine learning repository contains many interesting data sets, I encourage you to go through it.

So come on lets have fun with linear regression,

Exploring Boston Housing Data Set

The first step is to import the required Python libraries into [Ipython Notebook](#).

```
%matplotlib inline

import numpy as np
import pandas as pd
import scipy.stats as stats
import matplotlib.pyplot as plt
import sklearn
```

This data set is available in [sklearn](#) Python module, so I will access it using *scikit learn*. I am going to import Boston data set into Ipython notebook and store it in a variable called *boston*.

```
from sklearn.datasets import load_boston
boston = load_boston()
```

The object *boston* is a dictionary, so you can explore the keys of this dictionary.

```
boston.keys()
['data', 'feature_names', 'DESCR', 'target']
```

```
boston.data.shape
(506, 13)
```

I am going to print the feature names of *boston* data set.

```
print boston.feature_names
['CRIM' 'ZN' 'INDUS' 'CHAS' 'NOX' 'RM' 'AGE' 'DIS' 'RAD' 'TAX' 'PTRATIO'
'B' 'LSTAT']
```

I will see the description of this data set to know more about it. In this data set I have 506 instances(rows) and 13 attributes or parameters(columns). The goal of this exercise is to predict the housing prices in boston region using the features given.

```
print boston.DESCR

Boston House Prices dataset

Notes
-----
Data Set Characteristics:

    :Number of Instances: 506

    :Number of Attributes: 13 numeric/categorical predictive

    :Median Value (attribute 14) is usually the target


:Attribute Information (in order):
- CRIM      per capita crime rate by town
- ZN        proportion of residential land zoned for lots over 25,000 sq.ft.
- INDUS     proportion of non-retail business acres per town
- CHAS      Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
- NOX       nitric oxides concentration (parts per 10 million)
- RM        average number of rooms per dwelling
- AGE       proportion of owner-occupied units built prior to 1940
- DIS       weighted distances to five Boston employment centres
- RAD       index of accessibility to radial highways
- TAX       full-value property-tax rate per $10,000
- PTRATIO   pupil-teacher ratio by town
- B         1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town
- LSTAT     % lower status of the population
- MEDV      Median value of owner-occupied homes in $1000's
```

I am going to convert *boston.data* into a pandas data frame.

```
bos = pd.DataFrame(boston.data)
bos.head()
```

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0.00632	18	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.98
1	0.02731	0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.14
2	0.02729	0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03
3	0.03237	0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94
4	0.06905	0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	5.33

As you can see the column names are just numbers, so I am going to replace those numbers with the feature names.

```
bos.columns = boston.feature_names
bos.head()
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
0	0.00632	18	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.98
1	0.02731	0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.14
2	0.02729	0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03
3	0.03237	0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94
4	0.06905	0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	5.33


boston.target contains the housing prices.

```
boston.target[:5]
array([ 24. ,  21.6,  34.7,  33.4,  36.2])
```

I am going to add these target prices to the *bos* data frame.

```
bos['PRICE'] = boston.target
```

	RAD	TAX	PTRATIO	B	LSTAT	PRICE
900	1	296	15.3	396.90	4.98	24.0
371	2	242	17.8	396.90	9.14	21.6
371	2	242	17.8	392.83	4.03	34.7
322	3	222	18.7	394.63	2.94	33.4
322	3	222	18.7	396.90	5.33	36.2



Scikit Learn

In this section I am going to fit a linear regression model and predict the Boston housing prices. I will use the *least squares* method as the way to estimate the coefficients.

Y = boston housing price(also called “target” data in Python)

and

X = all the other features (or independent variables)

First, I am going to import linear regression from sci-kit learn module. Then I am going to [drop](#) the price column as I want only the parameters as my X values. I am going to store linear regression object in a variable called *lm*.

```
from sklearn.linear_model import LinearRegression
X = bos.drop('PRICE', axis = 1)

# This creates a LinearRegression object
lm = LinearRegression()
lm
```

If you want to look inside the linear regression object, you can do so by typing `LinearRegression`. and the press <tab> key. This will give a list of functions available inside linear regression object.

```
LinearRegression.  
LinearRegression.decision_function  
LinearRegression.fit  
LinearRegression.get_params  
LinearRegression.mro  
LinearRegression.predict  
LinearRegression.register  
LinearRegression.score  
LinearRegression.set_params
```

Important functions to keep in mind while fitting a linear regression model are:

`lm.fit()` -> fits a linear model

`lm.predict()` -> Predict Y using the linear model with estimated coefficients

`lm.score()` -> Returns the coefficient of determination (R^2). *A measure of how well observed outcomes are replicated by the model, as the proportion of total variation of outcomes explained by the model.*

You can also explore the functions inside `lm` object by pressing `lm.<tab>`

```
lm.  
lm.copy_X  
lm.decision_function  
lm.fit  
lm.fit_intercept  
lm.get_params  
lm.normalize  
lm.predict  
lm.score  
lm.set params
```

`.coef_` gives the coefficients and `.intercept_` gives the estimated intercepts.

Output	Description
<code>lm.coef_</code>	Estimated coefficients
<code>lm.intercept_</code>	Estimated intercept

Fitting a Linear Model

I am going to use all 13 parameters to fit a linear regression model. Two other parameters that you can pass to linear regression object are [fit_intercept and normalize](#).

```
In [20]: lm.fit(X, bos.PRICE)
```

```
Out[20]: LinearRegression(copy_X=True, fit_intercept=True, normalize=False)
```

I am going to print the intercept and number of coefficients.

```
In [26]: print 'Estimated intercept coefficient:', lm.intercept_  
Estimated intercept coefficient: 36.4911032804  
  
In [28]: print 'Number of coefficients:', len(lm.coef_)  
Number of coefficients: 13
```

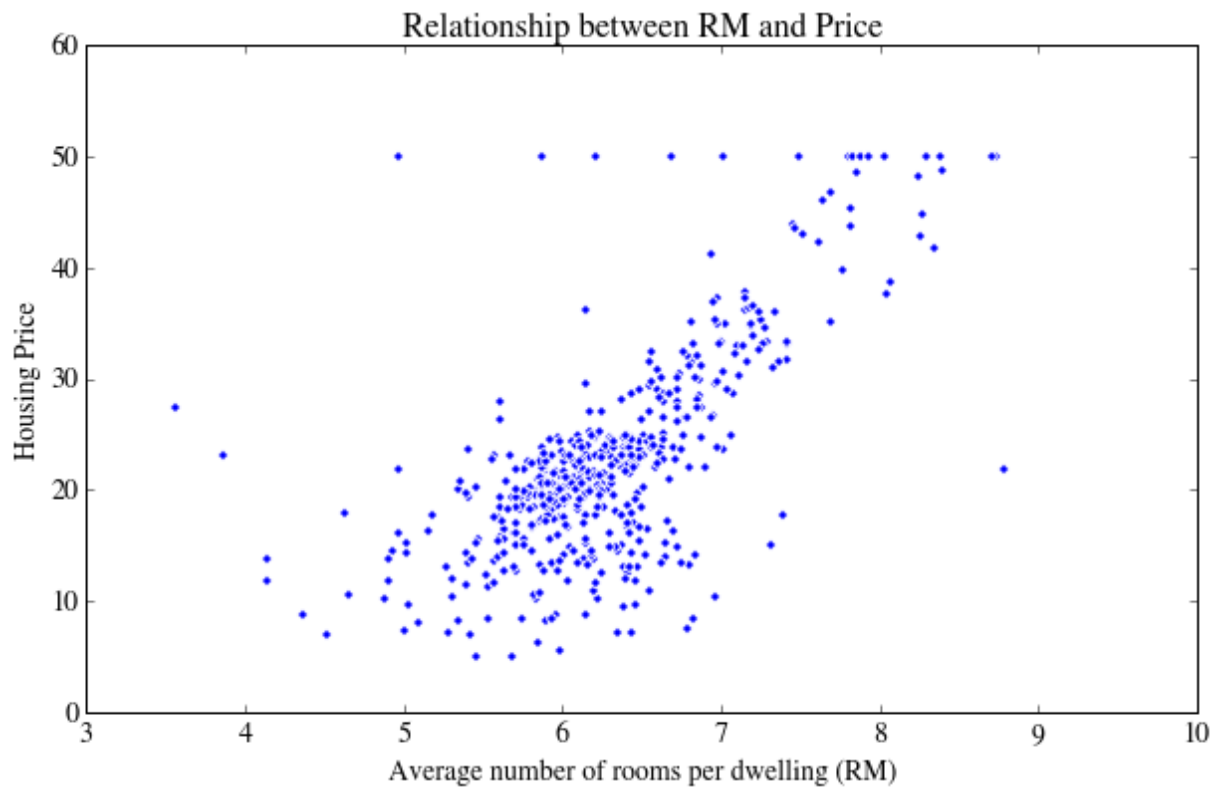
I then construct a data frame that contains features and estimated coefficients.

```
pd.DataFrame(zip(X.columns, lm.coef_), columns = ['features', 'estimatedCoefficients'])
```

	features	estimatedCoefficients
0	CRIM	-0.107171
1	ZN	0.046395
2	INDUS	0.020860
3	CHAS	2.688561
4	NOX	-17.795759
5	RM	3.804752
6	AGE	0.000751
7	DIS	-1.475759
8	RAD	0.305655

As you can see from the data frame that there is a high correlation between RM and prices. Lets plot a scatter plot between True housing prices and True RM.

```
In [29]: plt.scatter(bos.RM, bos.PRICE)  
plt.xlabel("Average number of rooms per dwelling (RM)")  
plt.ylabel("Housing Price")  
plt.title("Relationship between RM and Price")  
plt.show()
```



As you can see that there is a positive correlation between RM and housing prices.

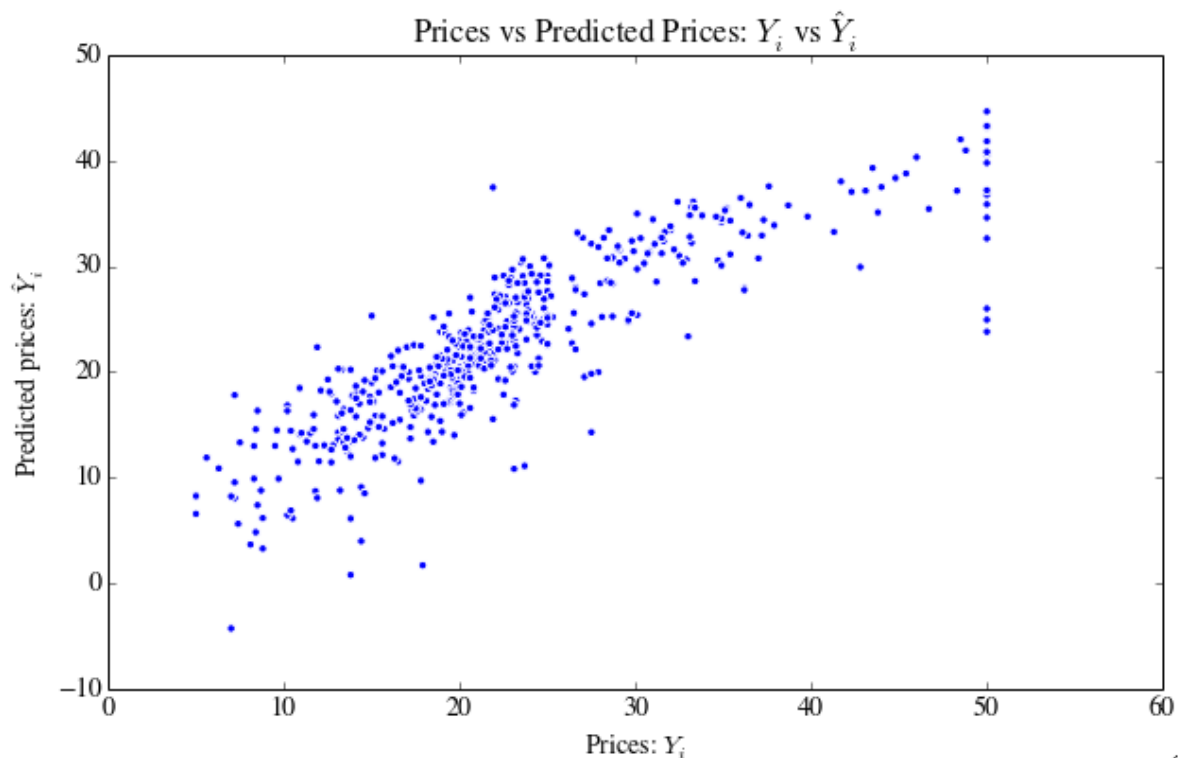
Predicting Prices

I am going to calculate the predicted prices (\hat{Y}_i) using *lm.predict*. Then I display the first 5 housing prices. These are my predicted housing prices.

```
In [24]: lm.predict(X)[0:5]
Out[24]: array([ 30.00821269,  25.0298606 ,  30.5702317 ,  28.60814055,  27.94288232])
```

Then I plot a scatter plot to compare true prices and the predicted prices.

```
plt.scatter(bos.PRICE, lm.predict(X))
plt.xlabel("Prices:  $Y_i$ ")
plt.ylabel("Predicted prices:  $\hat{Y}_i$ ")
plt.title("Prices vs Predicted Prices:  $Y_i$  vs  $\hat{Y}_i$ ")
```

You can notice that there is some error in the prediction as the housing prices increase.

Lets calculate the [mean squared error](#).

```
mseFull = np.mean((bos.PRICE - lm.predict(X)) ** 2)
print mseFull
```

21.8977792177

But if you fit linear regression for one feature the error will be very high. Lets take the feature 'PTRATIO' and calculate the mean squared error.

```
lm = LinearRegression()
lm.fit(X[['PTRATIO']], bos.PRICE)
```

LinearRegression(copy_X=True, fit_intercept=True, normalize=False)

```
msePTRATIO = np.mean((bos.PRICE - lm.predict(X[['PTRATIO']])) ** 2)
print msePTRATIO
```

62.6522000138

The mean squared error has increased. So this shows that a single feature is not a good predictor of housing prices.

Training and Validation Data sets

In practice you won't implement linear regression on the entire data set, you will have to split the data sets into [training and test](#) data sets. So that you train your model on training data and see how well it performed on test data.

How not to do train-test split:

```
X_train = X[:-50]
X_test = X[-50:]
Y_train = bos.PRICE[:-50]
Y_test = bos.PRICE[-50:]
print X_train.shape
print X_test.shape
print Y_train.shape
print Y_test.shape
```

(456, 13)
(50, 13)
(456,)
(50,)

You can create training and test data sets manually, but this is not the right way to do, because you may be training your model on less expensive houses and testing on expensive houses.

How to do train-test split:

You have to divide your data sets randomly. *Scikit* learn provides a function called *train_test_split* to do this.

```
X_train, X_test, Y_train, Y_test = sklearn.cross_validation.train_test_split(
    X, bos.PRICE, test_size=0.33, random_state = 5)
print X_train.shape
print X_test.shape
print Y_train.shape
print Y_test.shape
```

(339, 13)
(167, 13)
(339,)
(167,)

I am going to build a linear regression model using my train-test data sets.

```
lm = LinearRegression()
lm.fit(X_train, Y_train)
pred_train = lm.predict(X_train)
pred_test = lm.predict(X_test)
```

Then I calculate the mean squared error for training and test data.

Input:

```
print "Fit a model X_train, and calculate MSE with Y_train:", np.mean((Y_train - lm.predict(X_train))
** 2)

print "Fit a model X_train, and calculate MSE with X_test, Y_test:", np.mean((Y_test -
lm.predict(X_test)) ** 2)
```

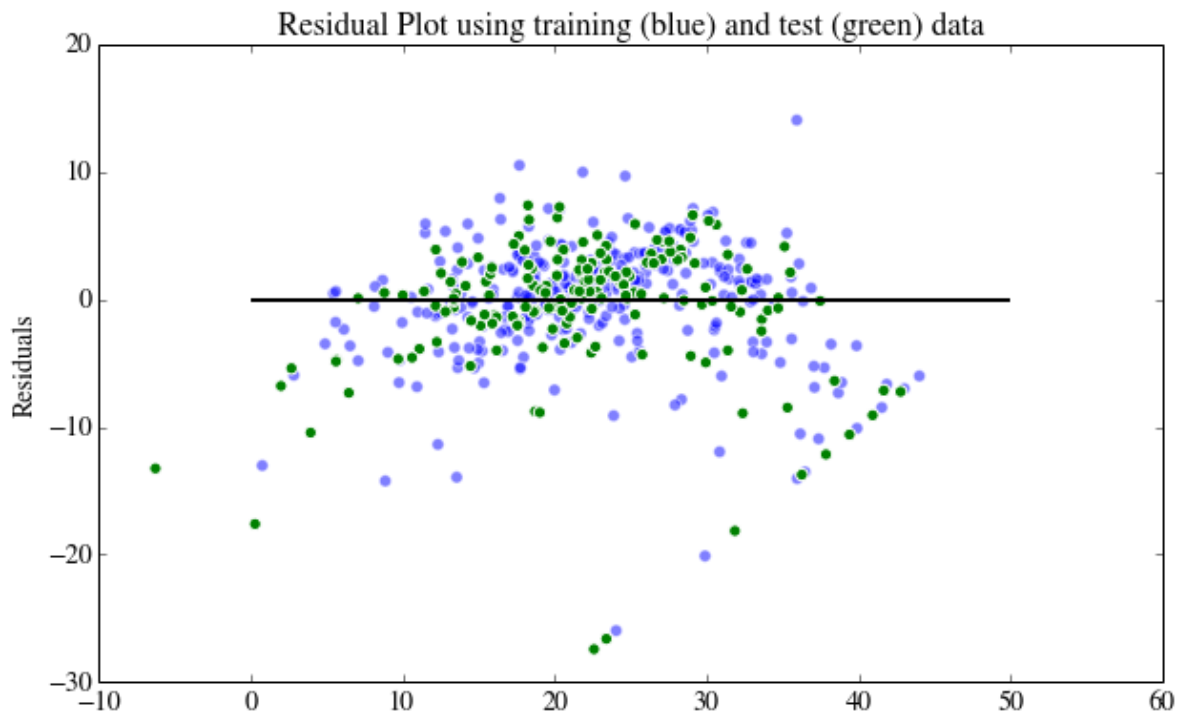
output:

```
Fit a model X_train, and calculate MSE with Y_train: 19.5467584735 Fit a model X_train, and
calculate MSE with X_test, Y_test: 28.5413672756
```

Residual plots

Residual plots are a good way to visualize the errors in your data. If you have done a good job then your data should be randomly scattered around line zero. If you see structure in your data, that means your model is not capturing some thing. Maybe there is a interaction between 2 variables that you are not considering, or maybe you are measuring time dependent data. If you get some structure in your data, you should go back to your model and check whether you are doing a good job with your parameters.

```
plt.scatter(lm.predict(X_train), lm.predict(X_train) - Y_train, c='b', s=40, alpha=0.5)
plt.scatter(lm.predict(X_test), lm.predict(X_test) - Y_test, c='g', s=40)
plt.hlines(y = 0, xmin=0, xmax = 50)
plt.title('Residual Plot using training (blue) and test (green) data')
plt.ylabel('Residuals')
```



Conclusion

To recap what I have done till now,

1. I explored the boston data set and then renamed its column names.
2. I explored the boston data set using `.DESCR`, my goal was to predict the housing prices using the given features.
3. I used Scikit learn to fit linear regression to the entire data set and calculated the mean squared error.
4. I made a train-test split and calculated the mean squared error for my training data and test data.
5. I then plotted the residuals for my training and test datasets.



admin / January 16, 2015 / Uncategorized

2 thoughts on “How To Run Linear Regression In Python SciKit-Learn”



April 17, 2015 at 6:59 pm

Hi Manu,

It is very nice presentation. I am wondering how to do the significant test for the coefficients of the predictor, how to perform anova, model selection etc. Those things are quite easy in R.

Thanks

Chris

[Reply](#)



Manu Jeevan 

May 3, 2015 at 3:43 pm

Hi Chris,

I do agree that these things are easy in r. But Scikit learn is mainly used for machine learning purpose.

Thanks, Manu

[Reply](#)

Leave a Reply

Your email address will not be published. Required fields are marked *

NAME *

EMAIL *

WEBSITE

COMMENT

PREVIOUS

How To Implement These 5 Powerful Probability Distributions In Python

NEXT

8 Best Python Data Science Books

RECENT POSTS

- [Interview With Digital Analytics Association Chairman – Jim Sterne](#)
- [The Minimalist Guide To Google Analytics](#)
- [How To Pass Google Analytics Exam – 2015](#)
- [Dealing with Unbalanced Classes ,Svm, Random Forests And Decision Trees In Python](#)
- [I Thought Of Sharing These 7 Machine Learning Concepts With You](#)

RECENT COMMENTS

- [reconquistar ex](#) on [Dealing with Unbalanced Classes ,Svm, Random Forests And Decision Trees In Python](#)
- [buy bulk car notes for sale](#) on [Dealing with Unbalanced Classes ,Svm, Random Forests And Decision Trees In Python](#)
- [Triki Judi](#) on [Dealing with Unbalanced Classes ,Svm, Random Forests And Decision Trees In Python](#)
- [The Hackathon Practice Guide by Analytics Vidhya](#) on [Dealing with Unbalanced Classes ,Svm, Random Forests And Decision Trees In Python](#)
- [5 Amazingly powerful Python libraries for data science | Big Data Made Simple – A gateway to Big Data environment](#) on [How I chose the right programming language for data science](#)

ARCHIVES

- [April 2015](#)
 - [February 2015](#)
 - [January 2015](#)
 - [December 2014](#)
 - [November 2014](#)
 - [October 2014](#)
-

CATEGORIES

- [Data Science](#)
 - [Uncategorized](#)
-

META

- [Log in](#)
- [Entries RSS](#)
- [Comments RSS](#)
- [WordPress.org](#)

Blog

About

Contact

Big Data Examiner / Proudly powered by WordPress