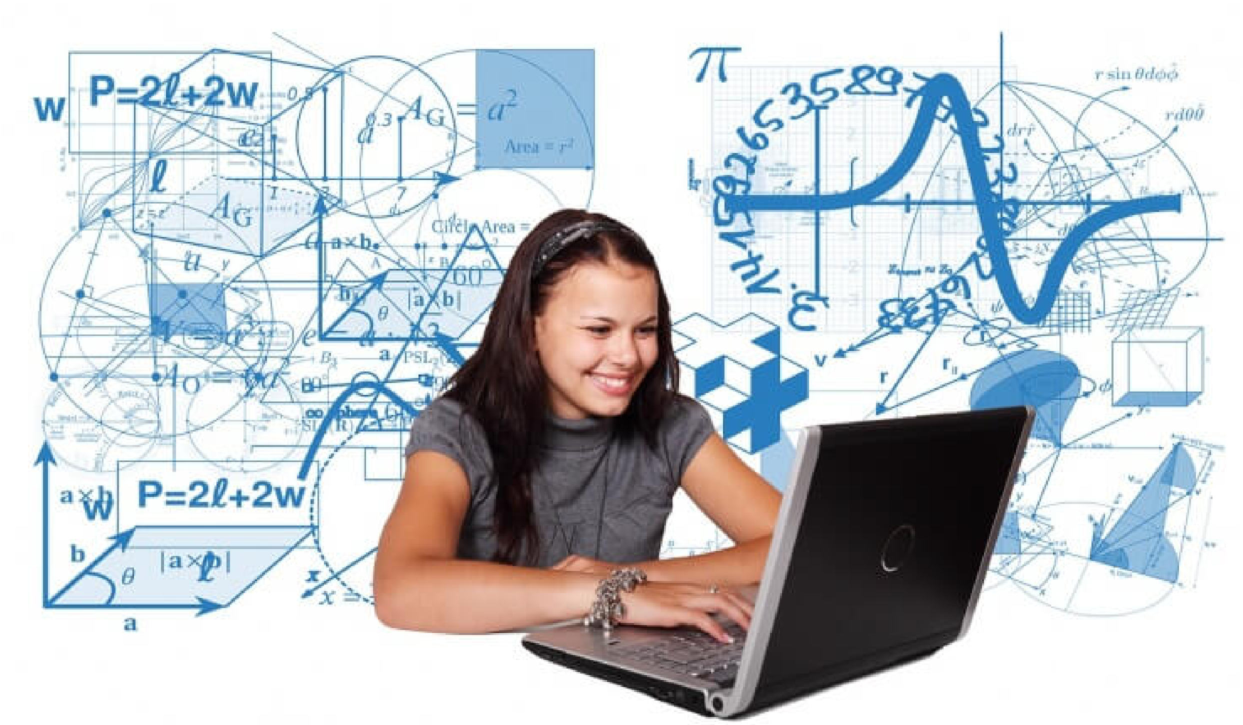
Step-by-step guide to execute Linear Regression in Python

edvancer.in/step-step-guide-to-execute-linear-regression-python



In my previous post, I explained the concept oflinear regression using R. In this post, I will explain how to implement linear regression using Python. I am going to use a Python library called Scikit Learn to execute Linear Regression.

Scikit-learn is a powerful Python module for machine learning and it comes with default data sets. I will use one such default data set called Boston Housing, the data set contains information about the housing values in suburbs of Boston.

Introduction

In my step by step guide to Python for data science article, I have explained how to install Python and the most commonly used libraries for data science. Go through this post to understand the commonly used Python libraries.

```
import numpy as np # python library for numerical functions
import matplotlib # fro plotting the graphs
import matplotlib.pyplot as  plt
from matplotlib import style # to use different styles while plotting
import pandas as pd # for making data frames
import sklearn # python library for linear and other models
import warnings # to supress future warnings ( not related to model making)
from sklearn import linear_model
from sklearn.cross_validation import train_test_split # for train-test split
warnings.simplefilter(action = "ignore", category = FutureWarning)
%matplotlib inline
```

Linear Regression using two dimensional data

First, let’s understand Linear Regression using just one dependent and independent variable.

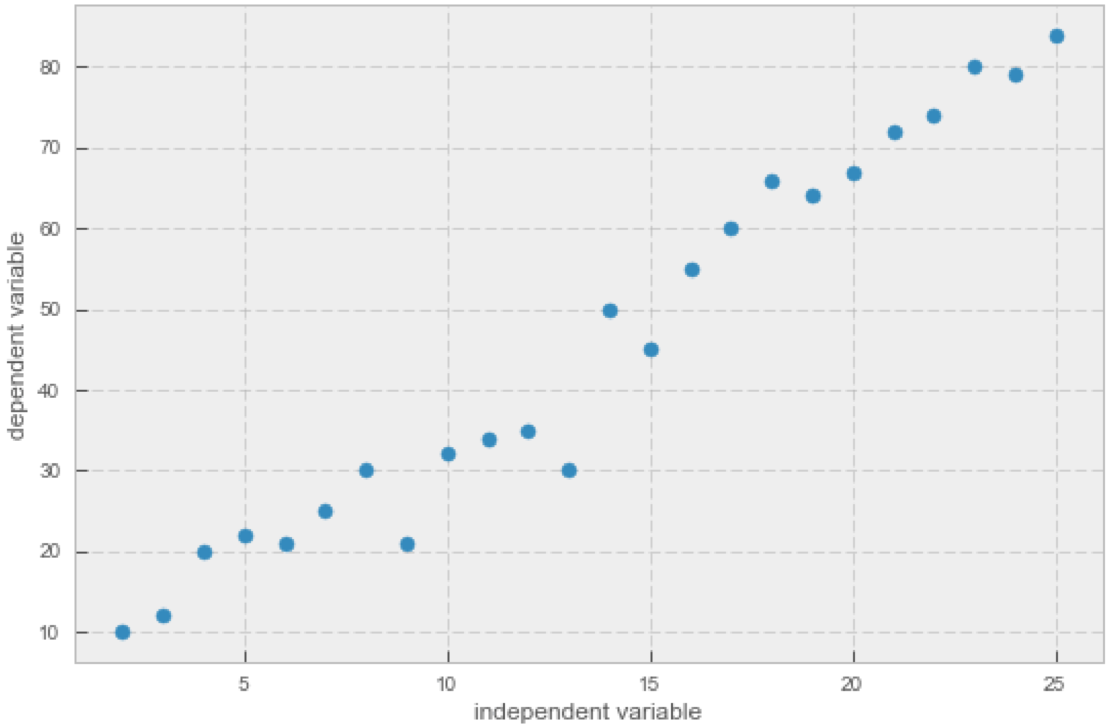
I create two lists xs and ys.

```
xs=[2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25]
ys=[10,12,20,22,21,25,30,21,32,34,35,30,50,45,55,60,66,64,67,72,74,80,79,84]
len(xs),len(ys)
```

```
Out[142]: (24, 24)
```

I plot these lists using a scatter plot. I assume xs as the independent variable and ys as the dependent variable.

```
In [143]: plt.scatter(xs,ys)
          plt.ylabel("dependent variable")
          plt.xlabel("independent variable")
          plt.show()
```



You can see that the dependent variable has a linear distribution with respect to the independent variable.

A linear regression line has the equation Y = mx+c, where m is the coefficient of independent variable and c is the intercept.

The mathematical formula to calculate slope (m) is:

(mean(x) * mean(y) – mean(x*y)) / (mean (x)^2 – mean(x^2))

The formula to calculate intercept (c) is:

mean(y) – mean(x) * m

Now, let’s write a function for intercept and slope (coefficient):

```
def slope_intercept(x_val,y_val):
    x=np.array(x_val)
    y=np.array(y_val)
    m=( ( (np.mean(x)*np.mean(y)) - np.mean(x*y) ) /
        ((np.mean(x)*np.mean(x)) - np.mean(x*x))   )
    m=round(m,2)
    b=(np.mean(y) - np.mean(x)*m)
    b=round(b,2)

    return m,b
```

To see the slope and intercept for xs and ys, we just need to call the function slope_intercept:

```
In [165]: slope_intercept(xs,ys)
Out[165]: (3.29, 0.92)
```

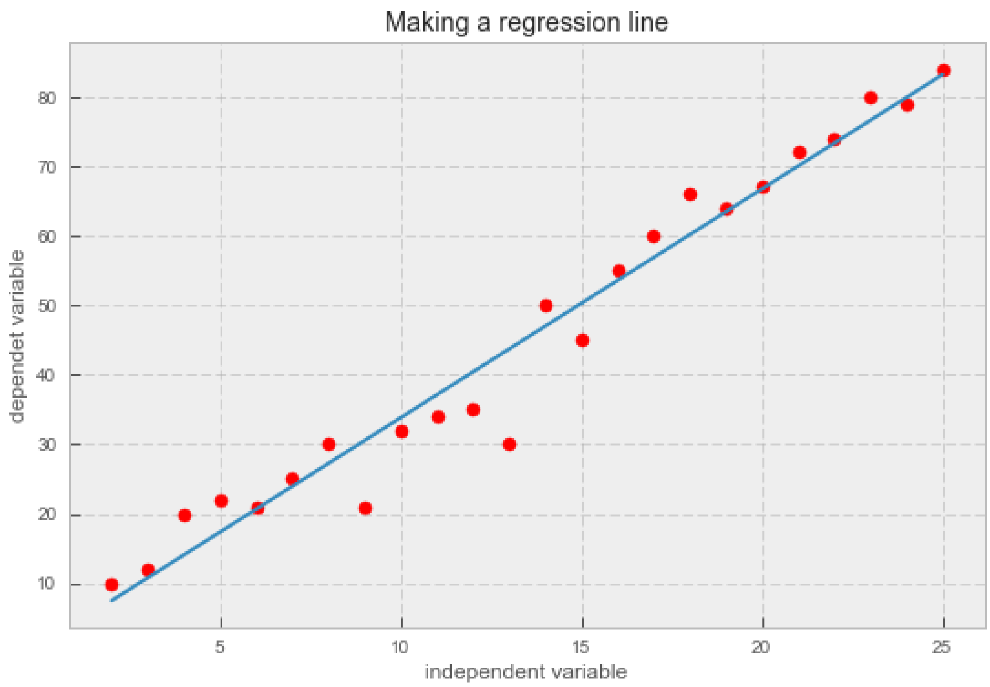
```
In [168]: m,b=slope_intercept(xs,ys)
```

reg_line is the equation of the regression line:

```
reg_line=[(m*x)+b for x in xs]
```

Now, let's plot a regression line on xs and ys:

```
In [172]: plt.scatter(xs,ys,color="red")
          plt.plot(xs,reg_line)
          plt.ylabel("dependet variable")
          plt.xlabel("independent variable")
          plt.title("Making a regression line")
          plt.show()
```



Root Mean Squared Error(RMSE)

RMSE is the standard deviation of the residuals (prediction errors). Residuals are a measure of how far from the regression line data points are, and RMSE is a measure of how spread out these residuals are.

If Yi is the actual data point and Y^i is the predicted value by the equation of line then RMSE is the square root of (Yi – Y^i)**2

Let’s define a function for RMSE:

```
def rmse(y1,y_hat):
    y_actual=np.array(y1)
    y_pred=np.array(y_hat)
    error=(y_actual-y_pred)**2
    error_mean=round(np.mean(error))
    err_sq=sqrt(error_mean)
    return err_sq
```

```
In [175]: rmse(ys,reg_line)

Out[175]: 4.58257569495584
```

Linear Regression using Scikit Learn

Now, let’s run Linear Regression on Boston housing data set to predict the housing prices using different variables.

I create a Pandas data frame for independent and dependent variables. The boston.target is the housing prices.

```
from sklearn.datasets import load_boston
```

```
In [7]: df_x=pd.DataFrame(boston.data,columns=boston.feature_names) # making a data frame for independent variables
```

```
In [8]: df_y=pd.DataFrame(boston.target) # making data frame of dependent variable or target
```

```
In [9]: df_x.head(13)
```

Out[9]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33
5	0.02985	0.0	2.18	0.0	0.458	6.430	58.7	6.0622	3.0	222.0	18.7	394.12	5.21
6	0.08829	12.5	7.87	0.0	0.524	6.012	66.6	5.5605	5.0	311.0	15.2	395.60	12.43

In [10]:
```
df_y.head(10)
```

Out[10]:

	0
0	24.0
1	21.6
2	34.7
3	33.4
4	36.2
5	28.7
6	22.9
7	27.1
8	16.5
9	18.9

In [13]:
```
df_x.shape # to know number of row and columns
```

Out[13]:
```
(506, 13)
```

In [25]:
```
names=[i for i in list(df_x)] # to get list of column names
names
```

Out[25]:
```
['CRIM',
 'ZN',
 'INDUS',
 'CHAS',
 'NOX',
 'RM',
 'AGE',
 'DIS',
 'RAD',
 'TAX',
 'PTRATIO',
 'B',
 'LSTAT']
```

Now, I am calling a linear regression model.

```
regr = linear_model.LinearRegression()
```

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. So that you train your model on training data and see how well it performed on test data.

I use 20 percentage of the total data as my test data.

```
x_train, x_test, y_train, y_test = train_test_split(df_x, df_y, test_size=0.2, random_state=4)
```

In [63]:
```
x_train.head()
```

Out[63]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
192	0.08664	45.0	3.44	0.0	0.437	7.178	26.3	6.4798	5.0	398.0	15.2	390.49	2.87
138	0.24980	0.0	21.89	0.0	0.624	5.857	98.2	1.6686	4.0	437.0	21.2	392.04	21.32
251	0.21409	22.0	5.86	0.0	0.431	6.438	8.9	7.3967	7.0	330.0	19.1	377.07	3.59
13	0.62976	0.0	8.14	0.0	0.538	5.949	61.8	4.7075	4.0	307.0	21.0	396.90	8.26
256	0.01538	90.0	3.75	0.0	0.394	7.454	34.2	6.3361	3.0	244.0	15.9	386.34	3.11

I fit the linear regression model to the training data set.

In [64]:
```
regr.fit(x_train,y_train) # making a linear regression model
```

Out[64]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
```

Let's calculate the intercept value, mean squared error, coefficients, and the variance score.

In [176]:
```
regr.intercept_
```

Out[176]:
```
array([ 35.60325757])
```

In [65]:
```
# The coefficients
print('Coefficients: \n', regr.coef_)
# The mean squared error
print("Mean squared error: %.2f"
      % np.mean((regr.predict(x_test) - y_test) ** 2))
# Explained variance score: 1 is perfect prediction
print('Variance score: %.2f' % regr.score(x_test, y_test))
```

```
('Coefficients: \n', array([[ -1.14743504e-01,   4.70875035e-02,   8.70282354e-03,
          3.23818824e+00,  -1.67240567e+01,   3.87662996e+00,
         -1.08218769e-02,  -1.54144627e+00,   2.92604151e-01,
         -1.33989537e-02,  -9.07306805e-01,   8.91271054e-03,
         -4.58747039e-01]]))
Mean squared error: 25.41
Variance score: 0.73
```

These are the coefficients of Independent variables (slope (m) of the regression line).

I attach the slopes to the respective independent variables.

```
reg.coef_[0].tolist() #
```

```
[-0.11474350352784292,
 0.04708750352305251,
 0.008702823544638971,
 3.2381882373524027,
 -16.724056662483488,
 3.876629957608199,
 -0.010821876932426422,
 -1.541446269218063,
 0.29260415086770986,
 -0.013398953732595587,
 -0.9073068048891137,
 0.008912710541206792,
 -0.45874703942843986]
```

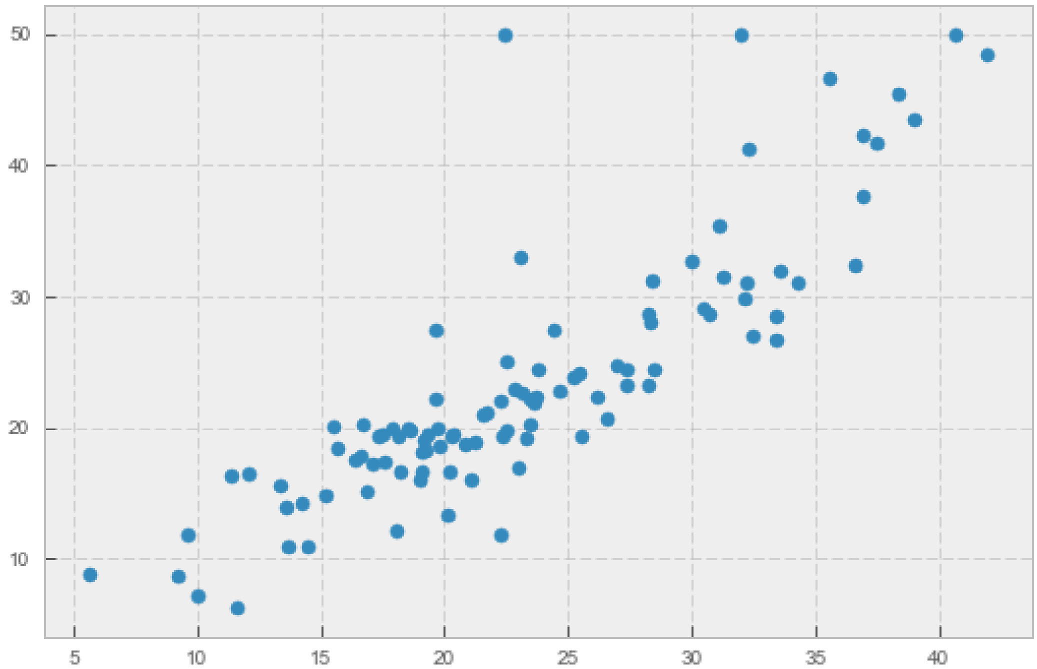
```
In [84]: pd.DataFrame(zip(names,reg.coef_[0].tolist()),columns=["names","coefficient"])
```

Out[84]:

	names	coefficient
0	CRIM	-0.114744
1	ZN	0.047088
2	INDUS	0.008703
3	CHAS	3.238188
4	NOX	-16.724057
5	RM	3.876630
6	AGE	-0.010822
7	DIS	-1.541446
8	RAD	0.292604
9	TAX	-0.013399
10	PTRATIO	-0.907307
11	B	0.008913
12	LSTAT	-0.458747

I plot the predicted x_test and y_test values.

```
style.use("bmh")
plt.scatter(regr.predict(x_test),y_test)
plt.show()
```



Select only the important variables for the model.

Scikit-learn is a good way to plot a linear regression but if we are considering linear regression for modelling purposes then we need to know the importance of variables(significance) with respect to the hypothesis.

To do this, we need to calculate the p value for each variable and if it is less than the desired cutoff(0.05 is the general cut off for 95% significance) then we can say with confidence that a variable is significant. We can calculate the p-value using another library called 'statsmodels'.

```
import statsmodels.api as sm
from statsmodels.sandbox.regression.predstd import wls_prediction_std
```

Ordinary least squares or linear least squares is a method for estimating the unknown parameters in a linear regression model. We have explained the OLS method in the first part of the tutorial.

model1=sm.OLS(y_train,x_train)

```
In [221]: result=model1.fit()
```

```
In [222]: print(result.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      0   R-squared:                       0.959
Model:                            OLS   Adj. R-squared:                  0.958
Method:                 Least Squares   F-statistic:                     711.8
Date:                Sun, 16 Apr 2017   Prob (F-statistic):          8.37e-263
Time:                        21:23:08   Log-Likelihood:                -1210.8
No. Observations:                 404   AIC:                             2448.
Df Residuals:                     391   BIC:                             2500.
Df Model:                          13
Covariance Type:            nonrobust
==============================================================================

                 coef    std err          t      P>|t|      [95.0% Conf. Int.]
------------------------------------------------------------------------------
CRIM          -0.1077      0.039     -2.779      0.006        -0.184    -0.031
ZN             0.0484      0.016      2.952      0.003         0.016     0.081
INDUS         -0.0232      0.073     -0.317      0.751        -0.167     0.121
CHAS           2.9930      1.062      2.819      0.005         0.906     5.080
NOX           -2.1626      3.662     -0.591      0.555        -9.362     5.036
RM             5.9590      0.339     17.584      0.000         5.293     6.625
AGE           -0.0169      0.015     -1.094      0.274        -0.047     0.013
DIS           -1.0273      0.220     -4.661      0.000        -1.461    -0.594
RAD            0.1669      0.075      2.240      0.026         0.020     0.313
TAX           -0.0105      0.004     -2.368      0.018        -0.019    -0.002
PTRATIO       -0.3753      0.124     -3.018      0.003        -0.620    -0.131
B              0.0143      0.003      4.733      0.000         0.008     0.020
LSTAT         -0.3463      0.057     -6.129      0.000        -0.457    -0.235
==============================================================================
Omnibus:                      151.837   Durbin-Watson:                   1.804
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              864.676
Skew:                           1.497   Prob(JB):                    1.73e-188
Kurtosis:                       9.512   Cond. No.                     8.44e+03
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 8.44e+03. This might indicate that there are
strong multicollinearity or other numerical problems.
```

We can drop few variables and select only those that have p values < 0.5 and then we can check improvement in the model.

A general approach to compare two different models is AIC(Akaike Information Criteria) and the model with minimum AIC is the best one.

```
model2=sm.OLS(y_train,x_train[['CRIM','ZN','CHAS','RM','DIS','RAD','TAX','PTRATIO','B','LSTAT']])
result2=model2.fit()
print(result2.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      0   R-squared:                       0.959
Model:                            OLS   Adj. R-squared:                  0.958
Method:                 Least Squares   F-statistic:                     926.5
Date:                Sun, 16 Apr 2017   Prob (F-statistic):          1.08e-266
Time:                        20:40:51   Log-Likelihood:                -1212.1
No. Observations:                 404   AIC:                             2444.
Df Residuals:                     394   BIC:                             2484.
Df Model:                          10
Covariance Type:            nonrobust

==============================================================================
                 coef    std err          t      P>|t|      [95.0% Conf. Int.]
------------------------------------------------------------------------------
CRIM          -0.1040      0.039     -2.695      0.007        -0.180    -0.028
ZN             0.0521      0.016      3.229      0.001         0.020     0.084
CHAS           2.7772      1.046      2.655      0.008         0.721     4.834
RM             5.7093      0.271     21.103      0.000         5.177     6.241
DIS           -0.8541      0.188     -4.542      0.000        -1.224    -0.484
RAD            0.1845      0.071      2.607      0.009         0.045     0.324
TAX           -0.0125      0.004     -3.412      0.001        -0.020    -0.005
PTRATIO       -0.3939      0.123     -3.197      0.002        -0.636    -0.152
B              0.0138      0.003      4.640      0.000         0.008     0.020
LSTAT         -0.3920      0.048     -8.168      0.000        -0.486    -0.298
==============================================================================
Omnibus:                      145.576   Durbin-Watson:                   1.802
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              764.271
Skew:                           1.454   Prob(JB):                    1.10e-166
Kurtosis:                       9.078   Cond. No.                     2.39e+03
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 2.39e+03. This might indicate that there are
strong multicollinearity or other numerical problems.
```

Dealing with multicollinearity

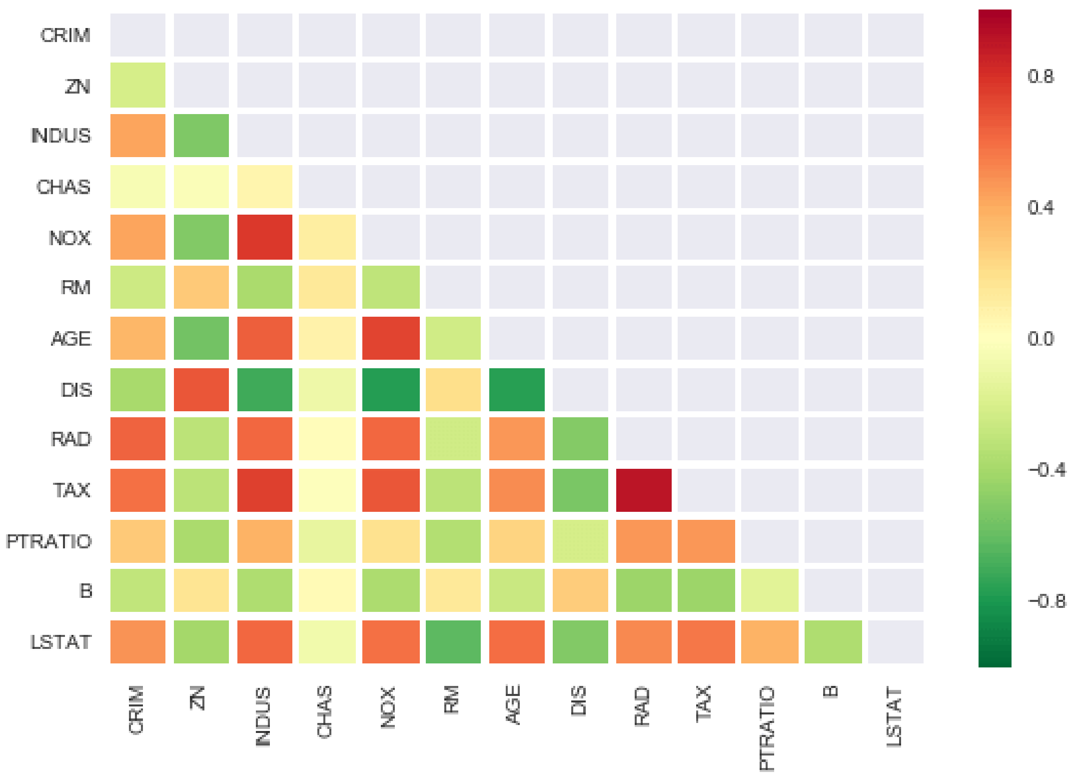
Multicollinearity is problem that you can run into when you're fitting a regression model. Simply put, multicollinearity is when two or more independent variables in a regression are highly related to one another, such that they do not provide unique or independent information to the regression.

We can check multicollinearity using this command: corr(method = “name of method”). I am going to make a correlation plot to see which parameters have multicollinearity issue.

In [228]:

```
import seaborn # a library similar to matplotlib
corr_df=x_train.corr(method='pearson')
print("--------------- CREATE A correlation plot ---------------")
# Create a mask to display only the lower triangle of the matrix (since it's mirrored around its
# top-left to bottom-right diagonal).
mask = np.zeros_like(corr_df)
mask[np.triu_indices_from(mask)] = True
# Create the heatmap using seaborn library.
# List if colormaps (parameter 'cmap') is available here: http://matplotlib.org/examples/color/colormaps_reference.html
seaborn.heatmap(corr_df, cmap='RdYlGn_r', vmax=1.0, vmin=-1.0 , mask = mask, linewidths=2.5)
# Show the plot we reorient the labels for each column and row to make them easier to read.
plt.yticks(rotation=0)
plt.xticks(rotation=90)
plt.show()
```

--------------- CREATE A correlation plot ---------------



Since this is a Pearson Coefficient, the values near to 1 or -1 have high correlation. For example, we can drop AGE and DIS and then execute a linear regression model to see if there are any improvements.

- About
- Latest Posts

Manu Jeevan

Manu Jeevan is a professional blogger, content marketer, and big data enthusiast. You can connect with him on LinkedIn, or email him at manu@bigdataexaminer.com.