

# Creating Diagnostic Plots in Python

and how to interpret them

*Posted on June 4, 2018*

Making the switch to Python after having used R for several years, I noticed there was a lack of good base plots for evaluating ordinary least squares (OLS) regression models in Python. From using R, I had familiarized myself with debugging and tweaking OLS models with the built-in diagnostic plots, but after switching to Python I didn't know how to get the original plots from R that I had turned to time and time again.

So, I did what most people in my situation would do - I turned to Google for help.

After trying different queries, I eventually found this (<https://medium.com/@emredjan/emulating-r-regression-plots-in-python-43741952c034>) excellent resource that was helpful in recreating these plots in a programmatic way. This post will leverage a lot of that work and at the end will wrap it all in a function that anyone can cut and paste into their code to reproduce these plots regardless of the dataset.

## What are diagnostic plots?

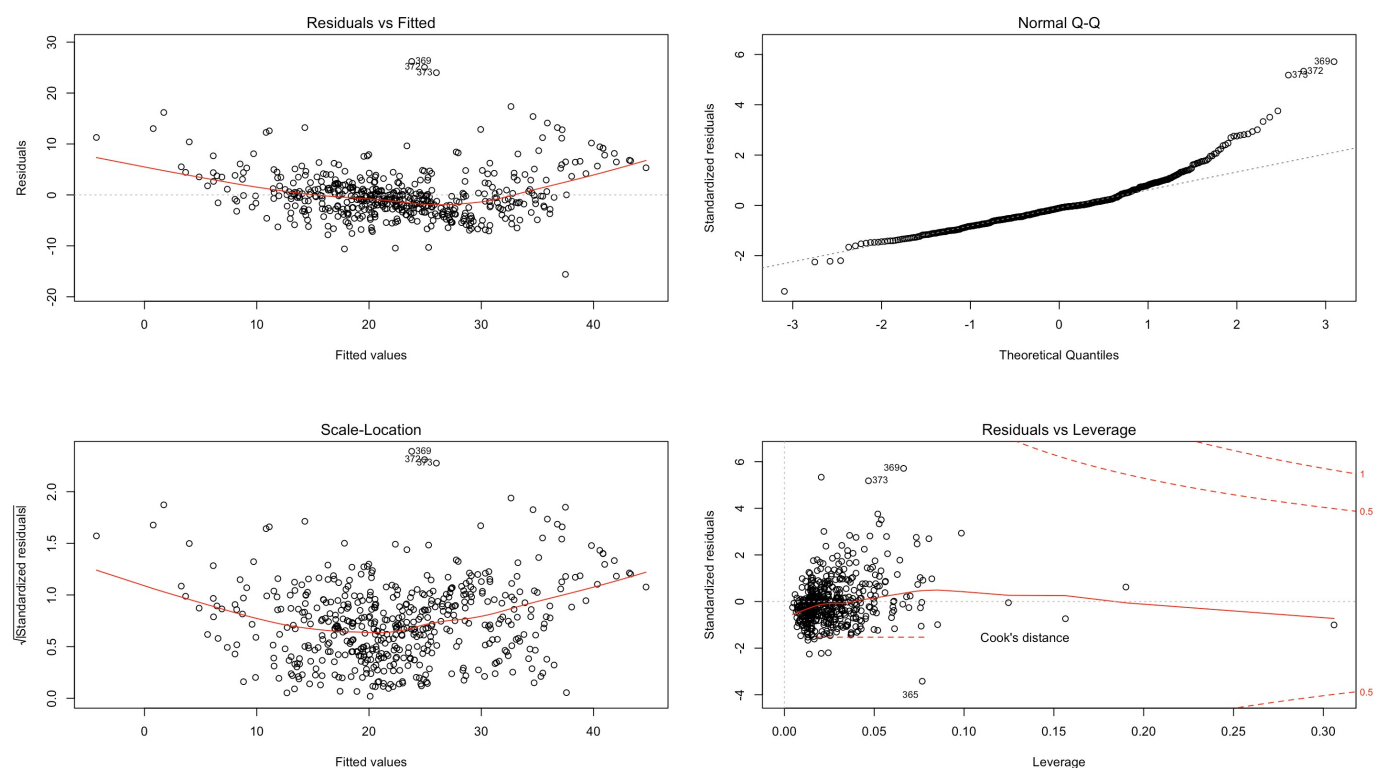
In short, diagnostic plots help us determine visually how our model is fitting the data and if any of the basic assumptions of an OLS model are being violated. We will be looking at four main plots in this post and describe how each of them can be used to diagnose issues in an OLS model. Each of these plots will focus on the **residuals** - or errors - of a model, which is mathematical jargon for the difference between the actual value and the predicted value, i.e.,  $r_i = y_i - \bar{y}_i$ .

These 4 plots examine a few different assumptions about the model and the data:

- 1) The data can be fit by a line (this includes any transformations made to the predictors, e.g.,  $x^2$  or  $\sqrt{x}$ )
- 2) Errors are normally distributed with mean zero
- 3) Errors have constant variance, i.e., homoscedasticity (<https://en.wikipedia.org/wiki/Homoscedasticity>)
- 4) There are no high leverage points (<https://newonlinecourses.science.psu.edu/stat501/node/337/>)

Let's look at an example in R, and its corresponding output, using the *Boston* housing data.

```
library(MASS)
model <- lm(medv ~ ., data=Boston)
par(mfrow=c(2,2))
plot(model)
```



Our goal is to recreate these plots using Python and provide some insight into their

usefulness using the housing dataset.

We'll begin by importing the relevant libraries necessary for building our plots and reading in the data.

## Libraries

```
import numpy as np
import pandas as pd
import seaborn as sns
import statsmodels.api as sm
import matplotlib.pyplot as plt
from statsmodels.graphics.gofplots import ProbPlot
plt.style.use('seaborn') # pretty matplotlib plots
plt.rc('font', size=14)
plt.rc('figure', titlesize=18)
plt.rc('axes', labelsiz=15)
plt.rc('axes', titlesize=18)
```

## Data

```
from sklearn.datasets import load_boston

boston = load_boston()

X = pd.DataFrame(boston.data, columns=boston.feature_names)
y = pd.DataFrame(boston.target)

# generate OLS model
model = sm.OLS(y, sm.add_constant(X))
model_fit = model.fit()

# create dataframe from X, y for easier plot handling
dataframe = pd.concat([X, y], axis=1)
```

## Residuals vs Fitted

First up is the **Residuals vs Fitted** plot. This graph shows if there are any nonlinear patterns in the residuals, and thus in the data as well. One of the mathematical assumptions in building an OLS model is that the data can be fit by a line. If this

assumption holds and our data can be fit by a linear model, then we should see a relatively flat line when looking at the residuals vs fitted.

An example of this failing would be trying to fit the function  $f(x) = x^2$  with a linear regression  $y = \beta_0 + \beta_1 x$ . Clearly, the relationship is nonlinear and thus the residuals have non-random patterns.

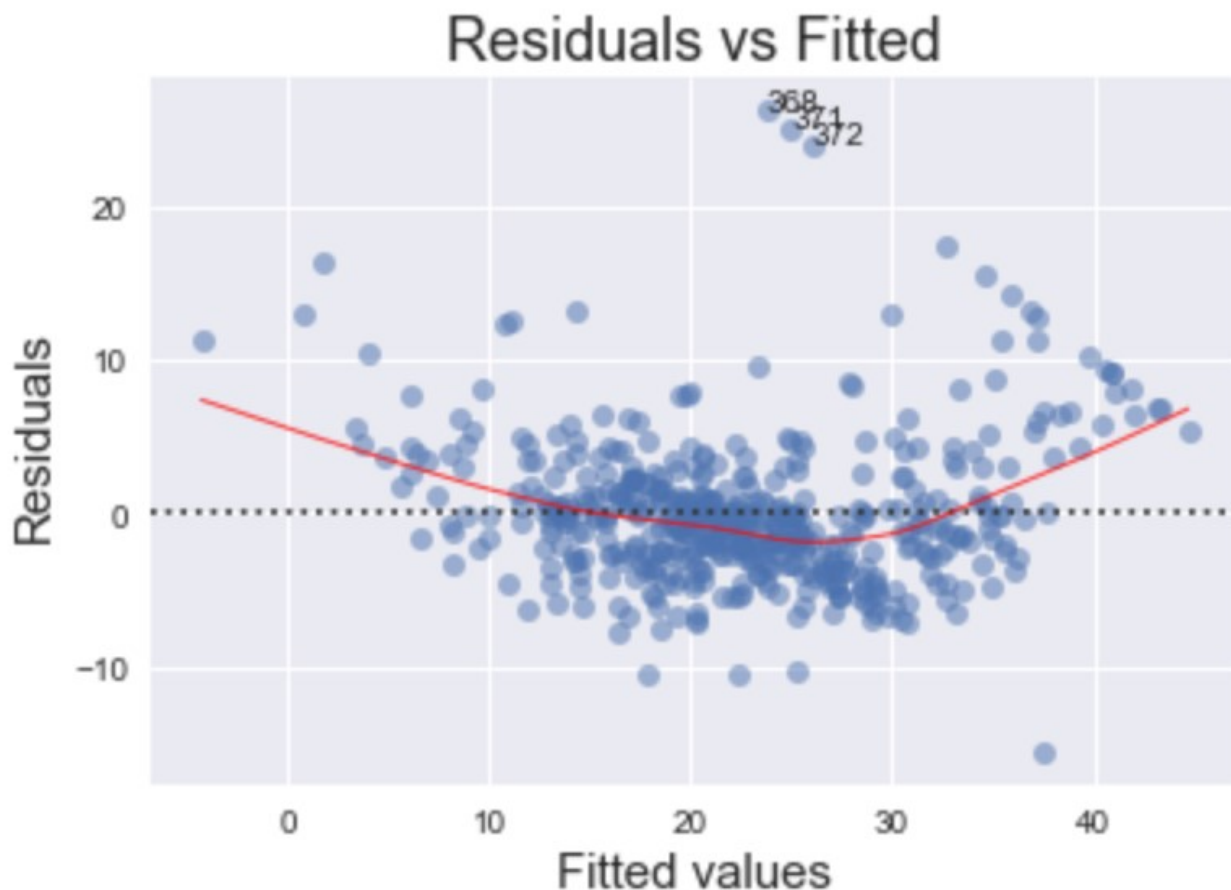
## Code

```
# model values
model_fitted_y = model_fit.fittedvalues
# model residuals
model_residuals = model_fit.resid
# normalized residuals
model_norm_residuals = model_fit.get_influence().resid_studentized_internal
# absolute squared normalized residuals
model_norm_residuals_abs_sqrt = np.sqrt(np.abs(model_norm_residuals))
# absolute residuals
model_abs_resid = np.abs(model_residuals)
# Leverage, from statsmodels internals
model_leverage = model_fit.get_influence().hat_matrix_diag
# cook's distance, from statsmodels internals
model_cooks = model_fit.get_influence().cooks_distance[0]

plot_lm_1 = plt.figure()
plot_lm_1.axes[0] = sns.residplot(model_fitted_y, dataframe.columns[-1], data=data,
                                lowess=True,
                                scatter_kws={'alpha': 0.5},
                                line_kws={'color': 'red', 'lw': 1, 'alpha': 0.8})

plot_lm_1.axes[0].set_title('Residuals vs Fitted')
plot_lm_1.axes[0].set_xlabel('Fitted values')
plot_lm_1.axes[0].set_ylabel('Residuals');
```

The code above yields the following plot



An ideal Residuals vs Fitted plot will look like random noise; there won't be any apparent patterns in the scatterplot and the red line would be horizontal.

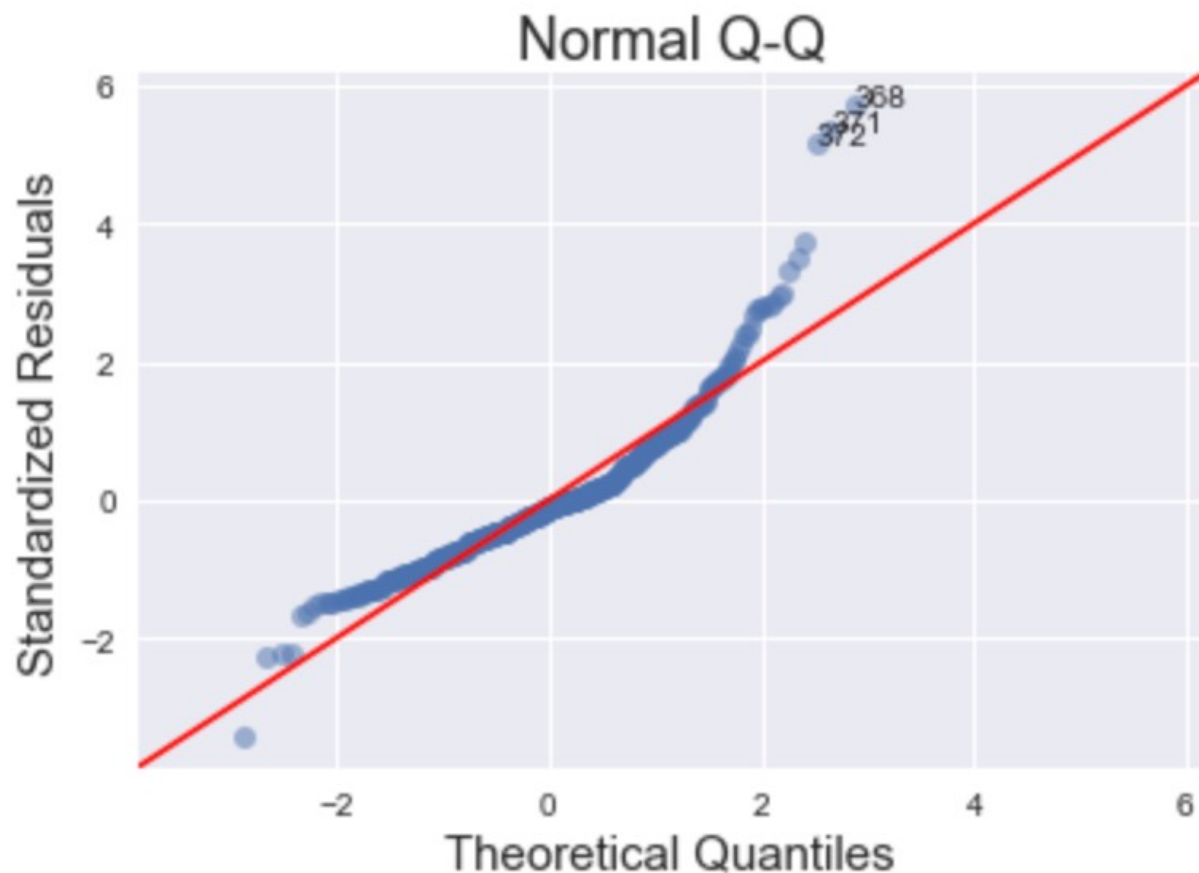
Examine the plot generated using the housing dataset. Notice the bow-shaped line in red? This is an indicator that we are failing to capture some of the non-linear features of the model. In other words, we are *underfitting* the model. Perhaps the variance in the data might be better captured using the *square* (or some other non-linear transformation) of one or more of the features. Which feature(s) specifically is beyond the scope of this post.

## Normal Q-Q Plot

This plot shows if the residuals are normally distributed. A **good** normal QQ plot has all of the residuals lying on or close to the red line.

## Code

```
QQ = ProbPlot(model_norm_residuals)
plot_lm_2 = QQ.qqplot(line='45', alpha=0.5, color='#4C72B0', lw=1)
plot_lm_2.axes[0].set_title('Normal Q-Q')
plot_lm_2.axes[0].set_xlabel('Theoretical Quantiles')
plot_lm_2.axes[0].set_ylabel('Standardized Residuals');
# annotations
abs_norm_resid = np.flip(np.argsort(np.abs(model_norm_residuals))), 0)
abs_norm_resid_top_3 = abs_norm_resid[:3]
for r, i in enumerate(abs_norm_resid_top_3):
    plot_lm_2.axes[0].annotate(i,
                               xy=(np.flip(QQ.theoretical_quantiles, 0)[r],
                                    model_norm_residuals[i]));
```



Looking at the graph above, there are several points that fall far away from the red line. This is indicative of the errors **not** being normally distributed, in fact our model suffers from “heavy tails”.

What does this say about the data? We are more likely to see extreme values than to

be expected if the data was truly normally distributed.

In general, there is plenty of wiggle room in violating these assumptions, but it is good to know what assumptions about the data we are violating.

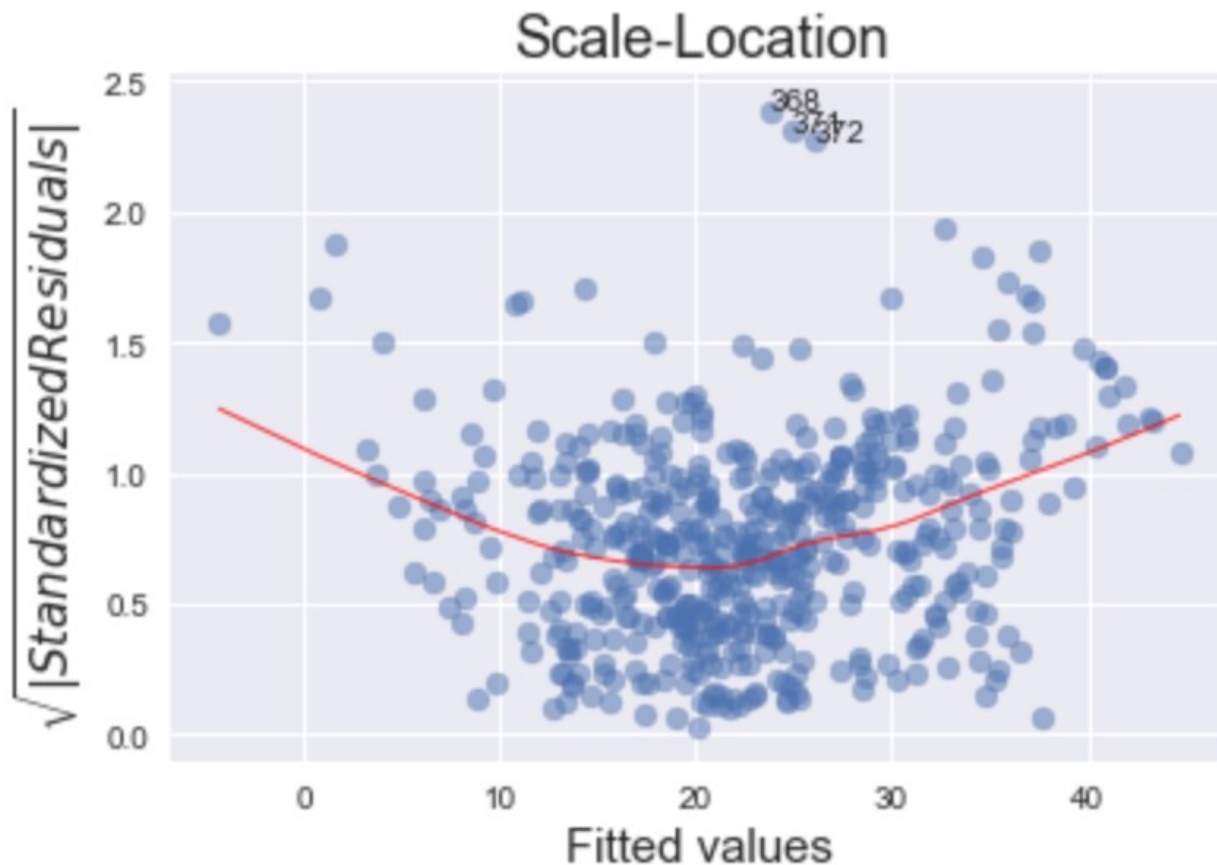
## Scale-Location

This plot is a way to check if the residuals suffer from non-constant variance, aka heteroscedasticity (<https://en.wikipedia.org/wiki/Heteroscedasticity>).

### Code

```
plot_lm_3 = plt.figure()
plt.scatter(model_fitted_y, model_norm_residuals_abs_sqrt, alpha=0.5);
sns.regplot(model_fitted_y, model_norm_residuals_abs_sqrt,
            scatter=False,
            ci=False,
            lowess=True,
            line_kws={'color': 'red', 'lw': 1, 'alpha': 0.8});
plot_lm_3.axes[0].set_title('Scale-Location')
plot_lm_3.axes[0].set_xlabel('Fitted values')
plot_lm_3.axes[0].set_ylabel('$\sqrt{|\text{Standardized Residuals}|}$');

# annotations
abs_sq_norm_resid = np.flip(np.argsort(model_norm_residuals_abs_sqrt), 0)
abs_sq_norm_resid_top_3 = abs_sq_norm_resid[:3]
for i in abs_sq_norm_resid_top_3:
    plot_lm_3.axes[0].annotate(i,
                              xy=(model_fitted_y[i],
                                  model_norm_residuals_abs_sqrt[i]));
```



This particular plot (with the housing data) is a tricky one to debug. The more horizontal the red line is, the more likely the data is homoscedastic. While a typical heteroscedastic plot has a sideways “V” shape, our graph has higher values on the left and on the right versus in the middle. This might be caused by not capturing the non-linearities in the model (see Residuals vs Fitted plot) and merits further investigation or model tweaking. The two most common methods of “fixing” heteroscedasticity is using a weighted least squares approach, or using a heteroscedastic-corrected covariance matrix (hccm). Both of these methods are beyond the scope of this post.

## Residuals vs Leverage

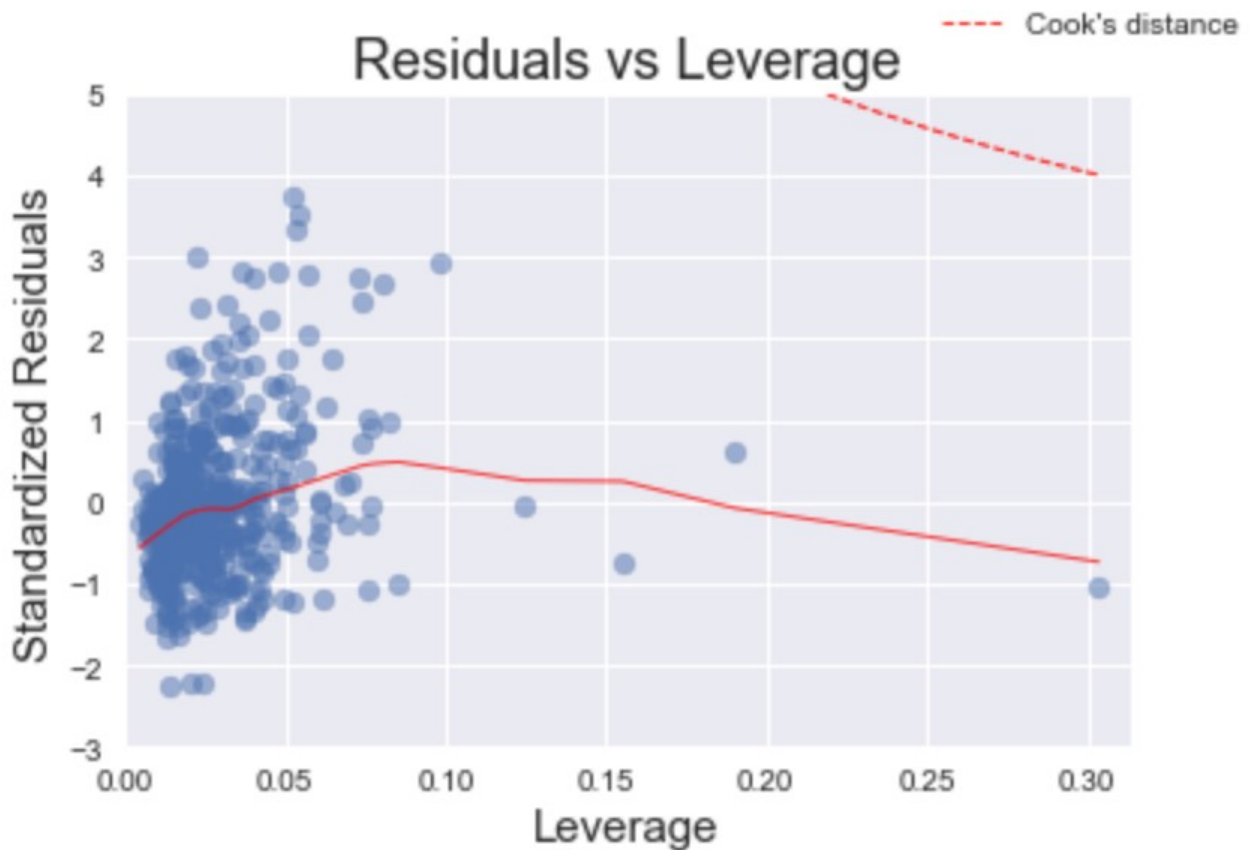
Leverage points are nasty buggers. Unlike outliers, which have an unusually large  $y$  value, leverage points have extreme  $x$  values. This may not seem so bad at face value, but it can have damaging effects on the model because the  $\beta$  coefficients are very sensitive to leverage points. The purpose of the Residuals vs Leverage plot is to identify these problematic observations.



## Code

```
plot_lm_4 = plt.figure();
plt.scatter(model_leverage, model_norm_residuals, alpha=0.5);
sns.regplot(model_leverage, model_norm_residuals,
            scatter=False,
            ci=False,
            lowess=True,
            line_kws={'color': 'red', 'lw': 1, 'alpha': 0.8});
plot_lm_4.axes[0].set_xlim(0, max(model_leverage)+0.01)
plot_lm_4.axes[0].set_ylim(-3, 5)
plot_lm_4.axes[0].set_title('Residuals vs Leverage')
plot_lm_4.axes[0].set_xlabel('Leverage')
plot_lm_4.axes[0].set_ylabel('Standardized Residuals');

# annotations
leverage_top_3 = np.flip(np.argsort(model_cooks), 0)[:3]
for i in leverage_top_3:
    plot_lm_4.axes[0].annotate(i,
                               xy=(model_leverage[i],
                                   model_norm_residuals[i]));
```



Fortunately, this is arguably one of the easiest plots to interpret. Thanks to Cook's Distance ([https://en.wikipedia.org/wiki/Cook%27s\\_distance](https://en.wikipedia.org/wiki/Cook%27s_distance)), we only need to find leverage points that have a distance greater than 0.5. In this plot, we do not have any leverage points that meet this criteria.

In practice, there may be cases where we may want to remove points with a Cook's distance of less than 0.5, especially if there are only a few observations compared to the rest of the data. I would argue that removing the point on the far right of the plot should improve the model. If the point is removed, we would re-run this analysis again and determine how much the model improved.

## Conclusion

In this post I set out to reproduce, using Python, the diagnostic plots found in the R programming language. Furthermore, I showed various ways to interpret them using a sample dataset.

Lastly, there will be readers who after seeing this post will want to reproduce these plots in a systematic way. This was something I had initially set out to do myself but did not find much success. Below, I provide the code for the function to reproduce the plots in Python.

## **Wrapping it all in a function**

```
1 def graph(formula, x_range, label=None):
2     """
3     Helper function for plotting cook's distance lines
4     """
5     x = x_range
6     y = formula(x)
7     plt.plot(x, y, label=label, lw=1, ls='--', color='red')
8
9
10 def diagnostic_plots(X, y, model_fit=None):
11     """
12     Function to reproduce the 4 base plots of an OLS model in R.
13
14     ---
15     Inputs:
16
17     X: A numpy array or pandas dataframe of the features to use in building t
18
19     y: A numpy array or pandas series/dataframe of the target variable of the
20
21     model_fit [optional]: a statsmodel.api.OLS model after regressing y on X.
22                          generated from X, y
23     """
24
25     if not model_fit:
26         model_fit = sm.OLS(y, sm.add_constant(X)).fit()
27
28     # create dataframe from X, y for easier plot handling
29     dataframe = pd.concat([X, y], axis=1)
30
31     # model values
32     model_fitted_y = model_fit.fittedvalues
33     # model residuals
34     model_residuals = model_fit.resid
35     # normalized residuals
36     model_norm_residuals = model_fit.get_influence().resid_studentized_intern
37     # absolute squared normalized residuals
38     model_norm_residuals_abs_sqrt = np.sqrt(np.abs(model_norm_residuals))
39     # absolute residuals
40     model_abs_resid = np.abs(model_residuals)
41     # leverage, from statsmodels internals
```

```
42 model_leverage = model_fit.get_influence().hat_matrix_diag
43 # cook's distance, from statsmodels internals
44 model_cooks = model_fit.get_influence().cooks_distance[0]
45
46 plot_lm_1 = plt.figure()
47 plot_lm_1.axes[0] = sns.residplot(model_fitted_y, dataframe.columns[-1],
48                                 lowess=True,
49                                 scatter_kws={'alpha': 0.5},
50                                 line_kws={'color': 'red', 'lw': 1, 'alpha': 0.8})
51
52 plot_lm_1.axes[0].set_title('Residuals vs Fitted')
53 plot_lm_1.axes[0].set_xlabel('Fitted values')
54 plot_lm_1.axes[0].set_ylabel('Residuals');
55
56 # annotations
57 abs_resid = model_abs_resid.sort_values(ascending=False)
58 abs_resid_top_3 = abs_resid[:3]
59 for i in abs_resid_top_3.index:
60     plot_lm_1.axes[0].annotate(i,
61                               xy=(model_fitted_y[i],
62                                   model_residuals[i]));
63
64 QQ = ProbPlot(model_norm_residuals)
65 plot_lm_2 = QQ.qqplot(line='45', alpha=0.5, color='#4C72B0', lw=1)
66 plot_lm_2.axes[0].set_title('Normal Q-Q')
67 plot_lm_2.axes[0].set_xlabel('Theoretical Quantiles')
68 plot_lm_2.axes[0].set_ylabel('Standardized Residuals');
69 # annotations
70 abs_norm_resid = np.flip(np.argsort(np.abs(model_norm_residuals)), 0)
71 abs_norm_resid_top_3 = abs_norm_resid[:3]
72 for r, i in enumerate(abs_norm_resid_top_3):
73     plot_lm_2.axes[0].annotate(i,
74                               xy=(np.flip(QQ.theoretical_quantiles, 0)[r],
75                                   model_norm_residuals[i]));
76
77 plot_lm_3 = plt.figure()
78 plt.scatter(model_fitted_y, model_norm_residuals_abs_sqrt, alpha=0.5);
79 sns.regplot(model_fitted_y, model_norm_residuals_abs_sqrt,
80            scatter=False,
81            ci=False,
82            lowess=True,
```

```

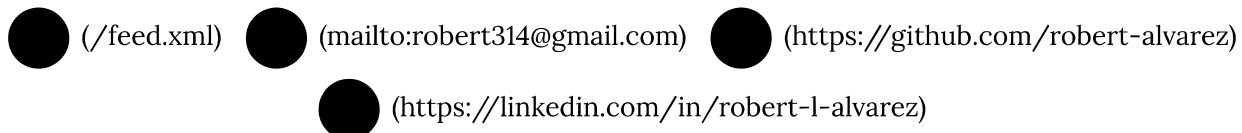
83         line_kws={'color': 'red', 'lw': 1, 'alpha': 0.8});
84     plot_lm_3.axes[0].set_title('Scale-Location')
85     plot_lm_3.axes[0].set_xlabel('Fitted values')
86     plot_lm_3.axes[0].set_ylabel('$\sqrt{|Standardized Residuals|}$');
87
88     # annotations
89     abs_sq_norm_resid = np.flip(np.argsort(model_norm_residuals_abs_sqrt), 0)
90     abs_sq_norm_resid_top_3 = abs_sq_norm_resid[:3]
91     for i in abs_sq_norm_resid_top_3:
92         plot_lm_3.axes[0].annotate(i,
93                                   xy=(model_fitted_y[i],
94                                       model_norm_residuals_abs_sqrt[i]));
95
96
97     plot_lm_4 = plt.figure();
98     plt.scatter(model_leverage, model_norm_residuals, alpha=0.5);
99     sns.regplot(model_leverage, model_norm_residuals,
100                scatter=False,
101                ci=False,
102                lowess=True,
103                line_kws={'color': 'red', 'lw': 1, 'alpha': 0.8});
104     plot_lm_4.axes[0].set_xlim(0, max(model_leverage)+0.01)
105     plot_lm_4.axes[0].set_ylim(-3, 5)
106     plot_lm_4.axes[0].set_title('Residuals vs Leverage')
107     plot_lm_4.axes[0].set_xlabel('Leverage')
108     plot_lm_4.axes[0].set_ylabel('Standardized Residuals');
109
110     # annotations
111     leverage_top_3 = np.flip(np.argsort(model_cooks), 0)[:3]
112     for i in leverage_top_3:
113         plot_lm_4.axes[0].annotate(i,
114                                   xy=(model_leverage[i],
115                                       model_norm_residuals[i]));
116
117     p = len(model_fit.params) # number of model parameters
118     graph(lambdax: np.sqrt((0.5 * p * (1 - x)) / x),
119          np.linspace(0.001, max(model_leverage), 50),
120          'Cook\'s distance') # 0.5 Line
121     graph(lambdax: np.sqrt((1 * p * (1 - x)) / x),
122          np.linspace(0.001, max(model_leverage), 50)) # 1 Line
123     plot_lm_4.legend(loc='upper right');
```

Tags: [OLS \(/tags#OLS\)](/tags#OLS) [diagnostic plots \(/tags#diagnostic plots\)](/tags#diagnostic%20plots) [python \(/tags#python\)](/tags#python) [linear regression \(/tags#linear regression\)](/tags#linear%20regression) [machine learning \(/tags#machine learning\)](/tags#machine%20learning)



← **PREVIOUS POST** (</2015-02-28-TEST-MARKDOWN/>)

**NEXT POST** → (</2019-06-07-INTRO-ANALYTICS-SYLLABUS/>)



Robert Alvarez • 2019 • [robert-alvarez.github.io](https://robert-alvarez.github.io) (<https://robert-alvarez.github.io>)

Theme by beautiful-jekyll (<http://deanattali.com/beautiful-jekyll/>)