Time Series Analysis, Regression, and Forecasting

With tutorials in Python

The Nonlinear Least Squares (NLS) Regression Model

And a tutorial on NLS Regression in Python and SciPy

Nonlinear Least Squares (NLS) is an optimization technique that can be used to build regression models for data sets that contain nonlinear features. Models for such data sets are nonlinear in their coefficients.

PART 1: The concepts and theory underlying the NLS regression model. This section has some math in it. You will enjoy it if you like math and/or are curious about how Nonlinear Least Squares Regression works.

PART 2: Tutorial on how to build and train an NLS regression model using Python and SciPy (https://www.scipy.org/)**.** You do *not* need to read PART 1 to understand PART 2.

PART 1: The theory behind NLS regression

We'll follow these representational conventions:

The 'hat' symbol (^) will be used for values that are generated by the process of fitting the regression model on data. For e.g. **β_(hat)** is the vector of **fitted** coefficients.

y_obs is the vector of observed values of the dependent variable **y**.

Variables in plain style are scalers and those in **bold** style represent their vector or matrix equivalents. For example, y_obs_i is a scaler containing the ith observed value of the **y_obs** vector which is of size (m x 1).

We will assume that the regression matrix **X** is of size (m x n) i.e. it has m data rows and each row contains n regression variables. The y matrix is of size (m x 1) and the coefficients matrix is of size (m x 1) (or 1 x m in its transpose form)

Now let's look at three examples of the sorts of nonlinear models which can be trained using NLS.

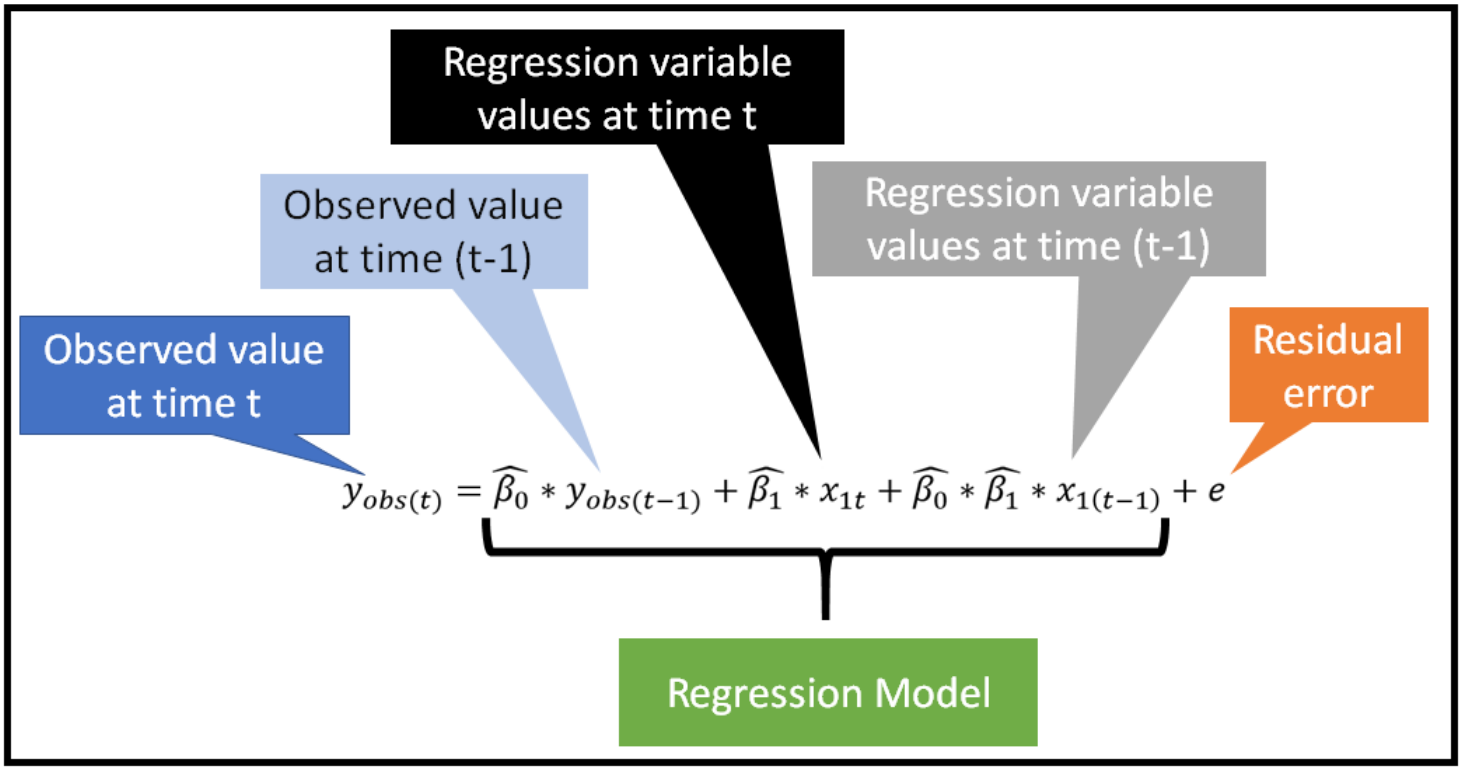
In the following model, the regression coefficients *β_1* and *β_2* are powers of two and three and thereby not linear.

$$y_{obs} = \widehat{\beta_0} + \widehat{\beta_1^2} * x_1 + \widehat{\beta_2^3} * x_2 + e$$

Equation of a regression model that is nonlinear in coefficients (Image by Author (https://www.linkedin.com/in/sachindate/))

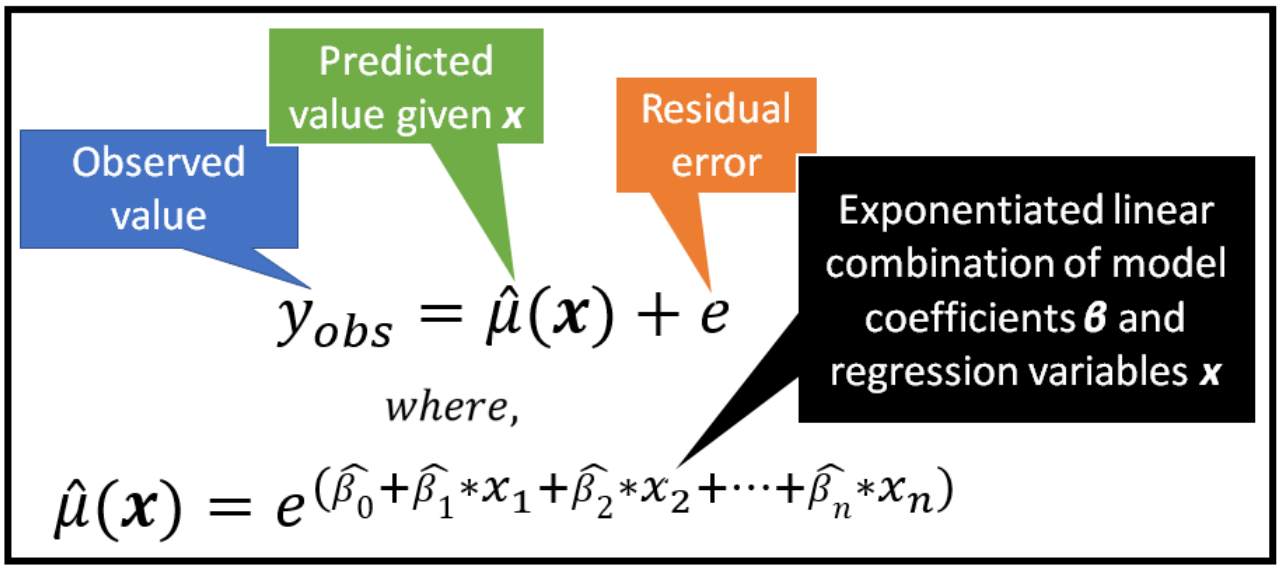
'*e*' is the residual error of the model, namely the difference between the observed *y* and the predicted value (which is the the combination of *β_0, β_1* and *β_2* and ***x*** on the R.H.S.*)*

The following model is an auto-regressive time series model containing coefficient*s β_1* and *β_2* in a multiplicative relationship, and therefore nonlinear in nature.



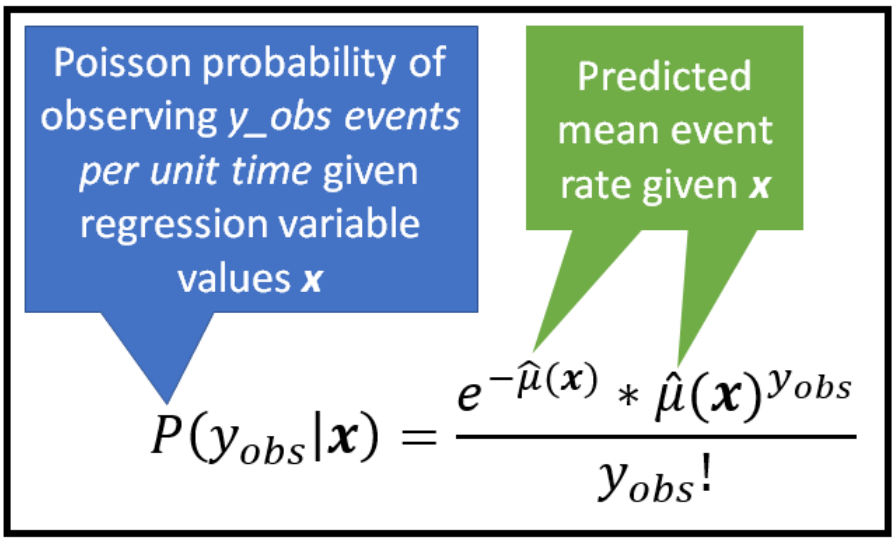
An auto-regressive time series model that is nonlinear in coefficients (Image by Author (https://www.linkedin.com/in/sachindate/))

In the following model, the predicted value is an exponential function of a linear combination of regression variables ***X***.



An exponentiated mean regression model (Image by Author (https://www.linkedin.com/in/sachindate/))

This last formulation is usually used in a Poisson regression model (https://timeseriesreasoning.com/contents/poisson-regression-model/) or its derivatives such as the Generalized Poisson (https://timeseriesreasoning.com/contents/generalized-poisson-regression-model/) or the Negative Binomial regression model (https://timeseriesreasoning.com/contents/negative-binomial-regression-model/). Specifically, the fitted mean *μ_cap* is expressed as the conditional mean of a Poisson probability distribution as follows:



Poisson probability of seeing y events per unit time given *a mean predicted rate of μ_(cap) events per unit time where μ_(cap) is a function of regression parameters* ***x.*** We've dropped the _i subscript for brevity (Image by Author (https://www.linkedin.com/in/sachindate/))

Such a Poisson regression model is used for fitting counts based data sets such as the number of people renting one of the bikes in a bike sharing program on each day.

How does NLS optimization work?

In NLS, our goal is to look for the model parameters vector $\boldsymbol{\beta}$ which would **minimize the sum of squares of residual errors**. In other words, we have to minimize the following:

$$Residual\ Sum\ of\ Squares\ (RSS) = \sum_{i}^{m} r_i = \sum_{i}^{m} \left(y_{obs_i} - \hat{\mu}_i\right)^2$$

Residual Sum of Squares of the fitted regression model (Image by Author (https://www.linkedin.com/in/sachindate/))

μ_cap_i (the model's prediction for the *ith* row in data set) is a function of model parameters vector $\boldsymbol{\beta\_cap}$ and the regression variables $\boldsymbol{x\_i}$, i.e.:

$$\hat{\mu}(\boldsymbol{x_i}) = f\left(\hat{\boldsymbol{\beta}}, \boldsymbol{x_i}\right)$$

The conditional mean predicted by the model for a given $\boldsymbol{x\_i}$ is a function of the fitted $\boldsymbol{\beta}$ ***and*** $\boldsymbol{x\_i}$ (Image by Author (https://www.linkedin.com/in/sachindate/))

Replacing *μ_i* with *f(**β_cap**, **x_i**)* in the earlier equation for RSS, we have:

$$RSS = \sum_{i}^{m} \left(y_{obs_i} - f\left(\hat{\boldsymbol{\beta}}, \boldsymbol{x_i}\right) \right)^2$$

Residual Sum of Squares of the fitted regression model (Image by Author (https://www.linkedin.com/in/sachindate/))

Once again, ***β_cap*** is the vector the ***fitted*** coefficients and ***x_i*** is the *ith* row of the regression variable matrix ***X***.

One way to minimize RSS is to differentiate RSS with respect to ***β_cap***, then set the differentiation to zero and solve for ***β_cap,*** i.e.:

$$\frac{\partial}{\partial \widehat{\beta_J}} (RSS) = 0 \;\; \forall j \in [1, n]$$

Partial derivatives of RSS w.r.t. ***β_cap***, and set to zero (Image by Author (https://www.linkedin.com/in/sachindate/))

Since ***β_cap*** is a vector of length *n* corresponding to *n* regression variables *x1, x2,…xn*, RSS needs to be partially differentiated w.r.t. each one of these *β_cap_j* coefficients and each equation set to zero. For example, the partial differentiation of RSS w.r.t. *β_cap_1* goes as follows:

$$\frac{\partial}{\partial \widehat{\beta_1}} (RSS) = 0$$

$$\implies \frac{\partial}{\partial \widehat{\beta_1}} \sum_{i}^{m} \left(y_{obs_i} - f\left(\hat{\boldsymbol{\beta}}, \boldsymbol{x_i}\right) \right)^2 = 0$$

$$\implies -2 \sum_{i}^{m} \left(y_{obs_i} - f\left(\hat{\boldsymbol{\beta}}, \boldsymbol{x_i}\right) \right) * \frac{\partial f\left(\hat{\boldsymbol{\beta}}, \boldsymbol{x_i}\right)}{\partial \widehat{\beta_1}} = 0$$

Partial differentiation of Residual Sum of Squares w.r.t. coefficient *β_1* (Image by Author (https://www.linkedin.com/in/sachindate/))

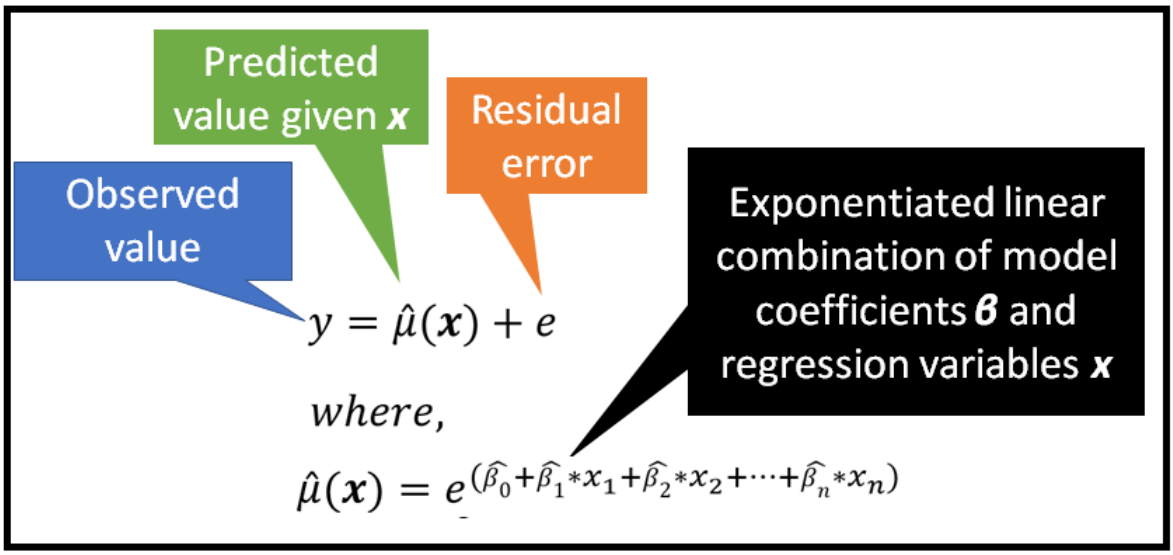
Since there are *n* coefficients *β_cap_1 to β_cap_n*, we get *n* equations of the kind shown above in *n* variables. However, as against the Ordinary Least Squares (OLS) (https://timeseriesreasoning.com/contents/assumptions-of-linear-regression/)estimation, there is no closed form solution for this system of *n* equations. So we have to use an iterative optimization technique in which at each iteration *k*, we make small adjustments to the values of *β_cap_1 to β_cap_n* as shown below, and reevaluate RSS:

$$\hat{\beta}_j^k = \hat{\beta}_j^{(k-1)} + \delta\hat{\beta}_j$$

At the *kth* iteration, increment *β_j* by *δβ_j*

Several algorithms have been devised to efficiently update the ***β_cap*** vector until an optimal set of values is reached that would minimize RSS. Chief among these are Trust Region (https://en.wikipedia.org/wiki/Trust_region) based methods (https://optimization.mccormick.northwestern.edu/index.php/Trust-region_methods) such as the Trust Region Reflective (https://epubs.siam.org/doi/10.1137/S1064827595289108) algorithm, the Levenberg–Marquardt algorithm (https://en.wikipedia.org/wiki/Levenberg%E2%80%93Marquardt_algorithm) and the imaginatively named Dogbox algorithm (https://nmayorov.wordpress.com/2015/06/19/dogbox-algorithm/). SciPy (https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.least_squares.html#scipy.optimize.least_squares) has support for all three algorithms.

Let's return to the exponentiated mean model we introduced earlier. In this model, we have:



The exponentiated mean model (Image by Author (https://www.linkedin.com/in/sachindate/))

Substituting in RSS:

$$RSS = \sum_i^m \left(y_{obs_i} - e^{\boldsymbol{x_i}\hat{\boldsymbol{\beta}}}\right)^2$$

Residual Sum of Squares of the fitted Poisson model (Image by Author (https://www.linkedin.com/in/sachindate/))

Notice that the ***x_i*β_cap*** in the exponent is a matrix multiplication of two matrices of dimensions *[1 x n]* and *[n x 1]* and therefore the result is a *[1×1]* matrix, i.e. effectively a scaler.

Differentiating the above equation w.r.t. ***β_cap*** and setting the differentiation to zero, we get the following set of equations (expressed in **vector** format) which need to be solved using one of the iterative optimization algorithms mentioned above:

$$\sum_{i}^{m} \boldsymbol{x_i} * \left(y_{obs_i} - e^{\boldsymbol{x_i}\widehat{\boldsymbol{\beta}}}\right) * e^{\boldsymbol{x_i}\widehat{\boldsymbol{\beta}}} = \mathbf{0}$$

The solution to this system of equations yields the fitted estimator ***β_cap*** for the Poisson regression model (Image by Author (https://www.linkedin.com/in/sachindate/))

PART 2: Tutorial on NLS Regression using Python and SciPy

Let's use the **Nonlinear Least Squares technique** to fit a Poisson regression model to a data set of daily usage of rental bicycles spanning two years.

The first 10 rows of the data set are as below:

instant	dteday	season	yr	mnth	holiday	weekday	workingday	weathersit	temp	atemp	hum	windspeed	casual_user_count	registered_user_count	total_user_count
1	01-01-11	1	0	1	0	6	0	2	0.344167	0.363625	0.805833	0.160446	331	654	985
2	02-01-11	1	0	1	0	0	0	2	0.363478	0.353739	0.696087	0.248539	131	670	801
3	03-01-11	1	0	1	0	1	1	1	0.196364	0.189405	0.437273	0.248309	120	1229	1349
4	04-01-11	1	0	1	0	2	1	1	0.2	0.212122	0.590435	0.160296	108	1454	1562
5	05-01-11	1	0	1	0	3	1	1	0.226957	0.22927	0.436957	0.1869	82	1518	1600
6	06-01-11	1	0	1	0	4	1	1	0.204348	0.233209	0.518261	0.0895652	88	1518	1606
7	07-01-11	1	0	1	0	5	1	2	0.196522	0.208839	0.498696	0.168726	148	1362	1510
8	08-01-11	1	0	1	0	6	0	2	0.165	0.162254	0.535833	0.266804	68	891	959
9	09-01-11	1	0	1	0	0	0	1	0.138333	0.116175	0.434167	0.36195	54	768	822
10	10-01-11	1	0	1	0	1	1	1	0.150833	0.150888	0.482917	0.223267	41	1280	1321

Rental bike usage counts (Source: UCI Machine Learning Repository (https://archive.ics.uci.edu/ml/datasets/Bike+Sharing+Dataset)) (Image by Author (https://www.linkedin.com/in/sachindate/))

You can download the data set from here (https://gist.github.com/sachinsdate/413910079ab4ef4332e7a97cae55d13a).

The NLS regression model

We'll build a regression model in which the **dependent variable** (***y***) is:

total_user_count: count of total bicycle renters

The **regression variables** matrix ***X*** will contain the following explanatory variables:

- **season**: the prevailing weather season
- **yr**: the prevailing year: 0=2011, 1=2012
- **mnth**: the prevailing month: 1 thru 12
- **holiday**: Whether the measurement was taken on a holiday (yes=1, no=0)
- **weekday**: day of the week (0 thru 6)
- **workingday**: Whether the measurement was taken on a working day (yes=1, no=0)
- **weathersit**: The weather situation on the day:
  1=Clear, Few clouds, Partly cloudy, Partly cloudy.
  2=Mist + Cloudy, Mist + Broken clouds, Mist + Few clouds, Mist.
  3=Light Snow, Light Rain + Thunderstorm + Scattered clouds, Light Rain + Scattered clouds.
  4=Heavy Rain + Ice Pellets + Thunderstorm + Mist, Snow + Fog.
- **temp**: Temperature, normalized to 39C
- **atemp**: Real feel, normalized to 50C
- **hum**: Humidity, normalized to 100
- **windspeed**: Wind speed, normalized to 67

Let's import all the required packages.

```
from scipy.optimize import least_squares
import pandas as pd
from patsy import dmatrices
import numpy as np
import statsmodels.api as sm
import statsmodels.formula.api as smf
import statsmodels.stats.stattools as st
import matplotlib.pyplot as plt
```

Read the data set into a Pandas DataFrame:

```
df = pd.read_csv('bike_sharing_dataset_daywise.csv', header=0, parse_dates=['dteday'], infer_datetime_format=True)
```

Create the training and test data sets. Since we are treating this data as a cross-sectional data (https://en.wikipedia.org/wiki/Cross-sectional_data), we will randomly select 90% of the data rows as our training data and the remaining 10% as our test data:

```
mask = np.random.rand(len(df)) < 0.9
df_train = df[mask]
df_test = df[~mask]
print('Training data set length='+str(len(df_train)))
print('Testing data set length='+str(len(df_test)))
```

Create the regression expression in Patsy (https://patsy.readthedocs.io/en/latest/quickstart.html) syntax. We are saying that **total_user_count** is the dependent variable and it depends on all the variables mentioned on the right side of the tilde (~) symbol:

Use Patsy to carve out the $\boldsymbol{y}$ and $\boldsymbol{X}$ matrices:

```
y_train, X_train = dmatrices(expr, df_train, return_type='dataframe')
y_test, X_test = dmatrices(expr, df_test, return_type='dataframe')
```

Let's define a couple a functions. The first function we will define is one that will calculate the exponentiated mean: ***mu_cap*** = *exp(**X***β_cap):*

```
def calc_exponentiated_mean(beta, x):
    lin_combi = np.matmul(np.array(x), np.array(beta))
    mean = np.exp(lin_combi)
    return mean
```

The second function we will define is one that will calculate the plain residual *r_i = (y_predicted — y_observed)* given the input matrices $\boldsymbol{\beta}$, $\boldsymbol{X}$ and ***y_obs***:

```
def calc_residual(beta, x, y_obs):
    y_pred = calc_exponentiated_mean(beta, x)
    r = np.subtract(y_pred, np.array(y_obs).flatten())
    return r
```

Initialize the coefficients $\boldsymbol{\beta}$ vector to all 1.0 values. There are regression variables in X (count them to verify!) and the regression intercept, i.e. two variables in all. So $\boldsymbol{\beta}$ is of size *(1 x 12)*. The numpy vector we will construct will be of the transpose shape (12,) which suits us as we will have to multiply the X_train with this vector and X_train is of shape (661, 12):

```
num_params = len(X_train.columns)
beta_initial = np.ones(num_params)
```

Finally, it is time to use the *least_squares()* method in SciPy to train the NLS regression model on (y_train, X_train) as follows:

```
result_nls_lm = least_squares(fun=calc_residual, x0=beta_initial, args=(X_train, y_train), method='lm', verbose=1)
```

Notice that we are using the Levenberg–Marquardt algorithm (https://en.wikipedia.org/wiki/Levenberg%E2%80%93Marquardt_algorithm) (***method**='lm'*) to perform the iterative optimization of the $\boldsymbol{\beta}$ vector.

The `result_nls_lm.x` variable contains the fitted $\boldsymbol{\beta}$ vector, in other words, ***β_cap***.

Let's organize ***β_cap*** into a Pandas DataFrame and print out the values:

```
df_beta_cap = pd.DataFrame(data=res_lsq_lm.x.reshape(1,num_params), columns=X_train.columns)

print(df_beta_cap)
```

We see the following output showing the fitted coefficient value for each regression variable and the fitted regression intercept:

Intercept	yr	season	mnth	holiday	weekday	workingday	weathersit	temp	atemp	hum	windspeed
7.655036	0.461603	0.107666	-0.004523	-0.133878	0.016588	0.025204	-0.1342	0.259541	0.844973	-0.143082	-0.451563

Fitted coefficient value for each regression variable and the fitted regression intercept

Prediction

Let's see how our model did on the test data set ***X_test*** that we had carved out earlier. We'll use the calc_exponentiated_mean() function which we had defined earlier but this time, we'll pass in the fitted ***β_cap*** vector and ***X_test***:

```
predicted_counts=calc_exponentiated_mean(beta=result_nls_lm.x, x=X_test)
```

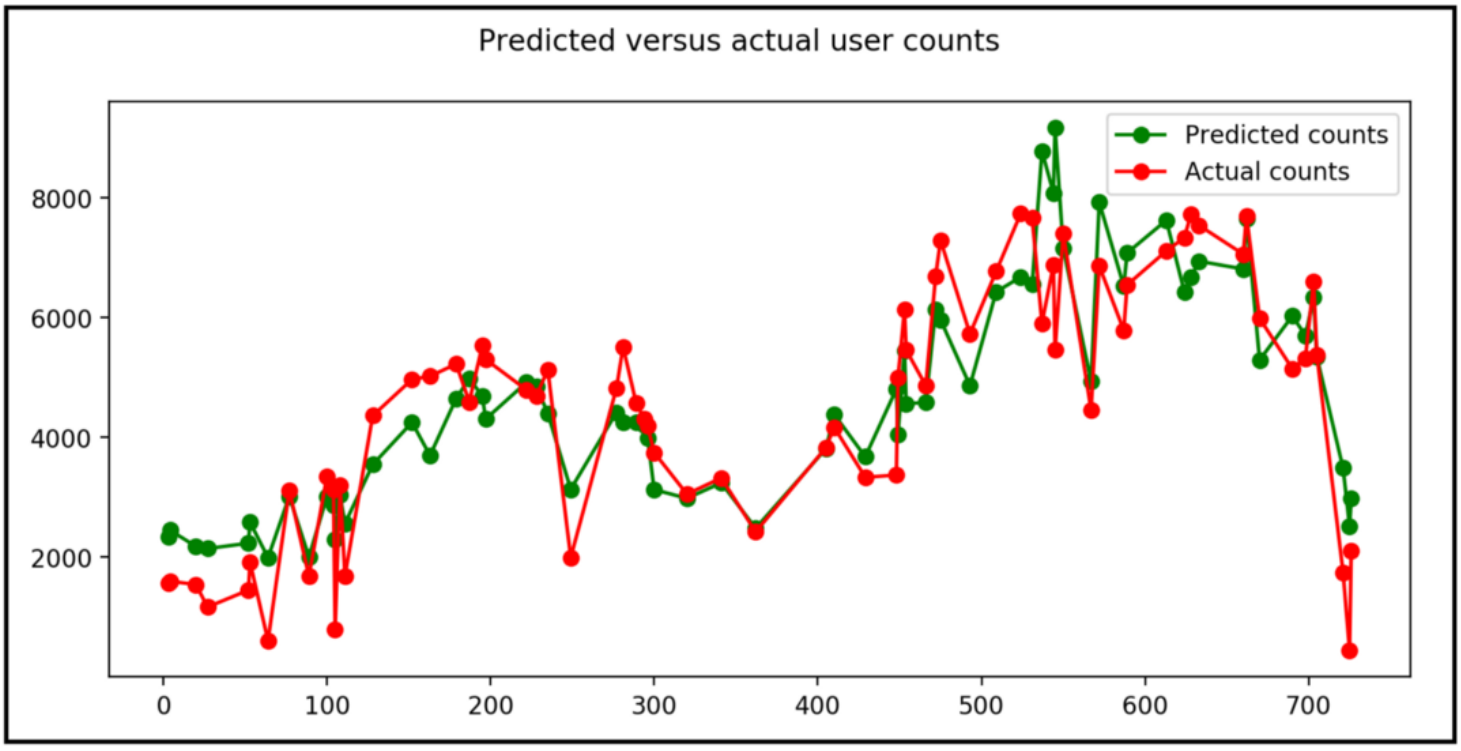
Let's plot the predicted versus the actual counts:

```
actual_counts = y_test['registered_user_count']

fig = plt.figure()
fig.suptitle('Predicted versus actual user counts')

predicted, = plt.plot(X_test.index, predicted_counts, 'go-', label='Predicted counts')
actual, = plt.plot(X_test.index, actual_counts, 'ro-', label='Actual counts')

plt.legend(handles=[predicted, actual])
plt.show()
```



Plot of predicted versus actual bicycle user counts on the test data set (Image by Author (https://www.linkedin.com/in/sachindate/))

Citations and Copyrights

Data set

The Bike sharing data set has been downloaded from UCI Machine Learning Repository (https://archive.ics.uci.edu/ml/datasets/Bike+Sharing+Dataset). **Curated data set download link** (https://gist.github.com/sachinsdate/413910079ab4ef4332e7a97cae55d13a).

Data set cited in paper: *Fanaee-T, Hadi, and Gama, Joao, "Event labeling combining ensemble detectors and background knowledge", Progress in Artificial Intelligence (2013): pp. 1–15, Springer Berlin Heidelberg, doi:10.1007/s13748–013–0040–3.*

Book

Cameron A. Colin, Trivedi Pravin K., *Regression Analysis of Count Data* (http://cameron.econ.ucdavis.edu/racd/count.html), Econometric Society Monograph №30, Cambridge University Press, 1998. ISBN: 0521635675

Images

All images are copyright Sachin Date (https://www.linkedin.com/in/sachindate/) under CC-BY-NC-SA (https://creativecommons.org/licenses/by-nc-sa/4.0/), unless a different source and copyright are mentioned underneath the image.

PREVIOUS: The Stratified Cox Proportional Hazards Model (https://timeseriesreasoning.com/contents/stratified-cox-proportional-hazards-regression/)

NEXT: Testing for Normality of Residual Errors Using Skewness and Kurtosis Tests (https://timeseriesreasoning.com/contents/skewness-and-kurtosis-measures/)

UP: Table of Contents (https://timeseriesreasoning.com/)