CTA Algorithm 3, Python Examples

Last Updated: September 13; First Released: January 09, 2015

Author: Kevin Boyle, President, DevTreks

Version: 2.1.6

A. Algorithm 3 Introduction

The sibling reference, *Conservation Technology Assessment* (CTA), introduces the background numerical techniques for completing CTAs. This reference introduces examples of CTAs completed using Algorithm 3, Python Algorithms.

This algorithm relies on the Python programming language to run Python scripts. Python is an open source programming language that contains mathematical, statistical, and graphing libraries that support the analysis of scientific data. The following URLs demonstrate using Python statistical libraries (statsmodel, scipy) to carry out a typical statistical analysis (ordinary least squares regression analysis):

http://statsmodels.sourceforge.net/devel/example_formulas.html

http://statsmodels.sourceforge.net/devel/examples/notebooks/generated/ols.html

http://docs.scipy.org/doc/scipy/reference/tutorial/linalg.html

Although the URLs demonstrate using Python directly, this algorithm programmatically runs these types of Python scripts for Indicator.URL datasets and then summarizes the statistical results in the Indicator.MathResult property. The scripts and datasets must be developed and tested in Python by algorithm authors prior to using this algorithm. Datasets and scripts must be cleaned up and tested prior to using them with any CTA algorithm. The same versions of the statistical packages documented in **Section B** must be used.

In the case of Resource Stock and M&E calculations, this algorithm requires that paths to Python script files, with ".txt" extensions, be added to the 1st URL of the Indicator.URL property. The corresponding TEXT dataset, with ".csv" extensions, gets added to the 2nd URL of the Indicator.URL. Use a semi-colon delimiter between all URLs. This algorithm does not require that datasets follow the standard explained in the *Resource Stock and M&E Calculation* references. Instead, the dataset should follow Python conventions for csv files (i.e. a header row followed by the raw data).

The current version runs Python scripts using the following techniques.

1. **Python 3.5 Anaconda installation package (1*)**: A current Anaconda installation package containing Python and supporting packages is installed on appropriate web servers or cloud virtual machines. Scripts are run by executing pythonw.exe through a programmatic command line. The path to pythonw.exe is set in the web project's appsettings.json file. This version uses Python 3.5.2 with Anaconda 4.1.1. Example 1 in **Appendix A** demonstrates this technique.
2. **Statistical Virtual Machine (2*)**: Azure cloud computing centers have preconfigured virtual machines that include Python and other machine learning packages (i.e. Azure Data Sciences Virtual Machine or ADSVM**)**. Fees are charged only for the virtual machine, not the software on the vm. Examples 2 and 3 in **Appendix A** demonstrates using respective ASP.NET Core WebApis to run scripts using this technique.
3. **Python 3.5.2 with packages installation (3*)**: Azure App Service web apps can be configured to directly use specific versions of Python, including the latest version, 3.5.2. Implying that the Azure site can directly access Python without the need to pay for a virtual machine. Footnote 3 documents current difficulties with this approach.
4. **Azure Machine Learning (AML) web service with Python algorithms**: The sibling reference, CTA 4 AML Algorithms, has examples demonstrating how to use Python with an AML web service.

The current default configuration uses Technique 1 for web server deployments and Techniques 2 and 3, Statistical Virtual Machines, for cloud sites. Future releases will investigate using more installation techniques as well as supporting additional Python features (i.e. IPython notebooks, chained scripts and datasets, graphical displays).

The goal of most scripts will be to produce confidence intervals for an Indicator's QTM, QTL, and QTU properties. Most subalgorithms generate those properties from the last csv row of the results.

B. Subalgorithms and Examples

Appendix A has examples demonstrating each of the following subalgorithms.

subalgorithm1: Scripts: Runs generic Python scripts where Python has been installed on the same server as the web app (**3***).

The following subalgos were not tested for Version 2.1.6

subalgorithm2: WebApi Scripts, Small Jobs: Runs generic Python scripts where Python has been installed on another server where an ASP.NET Core 1 WebApi web app, DevTreksStatsApi, has been deployed. DevTreks is the client that consumes the response from the WebApi. This technique is cross platform.

subalgorithm3: WebApi Scripts, Large Jobs: [Large job example needed.]

subalgorithm4: **Custom Packages**: [Custom package example needed.]

subalgorithm5: Python Mathematical Libraries: [Python mathematical library (numpy) example needed.]

Summary

This reference demonstrates how to use Python algorithms to complete CTAs. CTAs may help people to reach decisions that improve their lives and livelihoods.

Footnotes

1. The security implications of allowing users to run generic Python scripts has not been fully investigated yet.
2. The Statistical Virtual Machine can be cross platform. Although any statistical package can be used on the virtual machine, the current source only supports R. Python, and AML. Julia has been stubbed out but not fully developed.
3. Although Python 3.5.2 can be activated, developers must install its related packages, such as scipy, manually. After experimenting with manually installing scipy and statsmodels packages, the author stopped because too much time was being wasted. The dependencies needed for successful installation require too much manual, error-prone, labor (or significant investment in learning). Essentially, packages optimized for Unix present unusual challenges for Windows. The author decided to rely on the Anaconda installation for the current release. The Azure Python reference, below, suggests improvements in package installation on Azure web app sites may be coming out at a later date.

References (URLs were last accessed September, 2016)

Python. Version 3.5.2 https://www.python.org/

Anaconda: https://www.continuum.io/downloads

Anaconda: http://conda.pydata.org/docs/test-drive.html

Azure Python: https://blogs.msdn.microsoft.com/pythonengineering/2016/08/04/upgrading-python-on-azure-app-service/

Azure Data Sciences Virtual Machine: https://azure.microsoft.com/en-us/documentation/articles/machine-learning-data-science-provision-vm/

References Note

We try to use references that are open access or that do not charge fees.

Improvements, Errors, and New Features

Please notify DevTreks (devtrekkers@gmail.com) if you find errors in these references. Also please let us know about suggested improvements or recommended new features.

A video tutorial explaining this reference can be found at:

https://www.devtreks.org/commontreks/preview/commons/resourcepack/Technology Assessment 1/1526/none

Appendix A. Algorithm 3 Examples.

These examples demonstrate using Python subalgorithms.

Example 1. Algorithm 3. SubAlgorithm1: Python Scripts

URLs:
These datasets are owned by the Natural Resource Stock club in the GreenTreks network group. If testing on localhost, switch to the Carbon Emission Reduction club. The Social Performance tutorial contains examples of more advanced Python statistics.

https://www.devtreks.org/greentreks/preview/carbon/input/Python OLS 1/2147397538/none

Python 1st Indicator.URL script URL
https://devtreks1.blob.core.windows.net/resources/network_carbon/resourcepack_1534/resource_7969/Regress1.csv

Python 2nd Indicator.URL TEXT dataset
https://devtreks1.blob.core.windows.net/resources/network_carbon/resourcepack_1534/resource_7967/PyOLSWeb1.txt
https://devtreks1.blob.core.windows.net/resources/network_carbon/resourcepack_1534/resource_7970/PyOLSCloud1.txt

Media URL graphical plots
https://devtreks1.blob.core.windows.net/resources/network_carbon/resourcepack_1534/resource_7966/Ex9Plot1.PNG

The following URL contains both Resource Stock and M&E calculators that run this example.

http://localhost:5000/greentreks/preview/carbon/input/Example 3, Python Web Regression/2147409826/none

The first example is for the same linear regression that is used in Example 1h in the CTA 1 reference. The dependent variable in the following regression is *household energy consumption* and the independent variables are *household size* and *household size squared*.

Abstract equation: y = B0 + B1x1 + B2x1^2 + e
Math Expression: I1.Q1.housesize1 + I1.Q2.housesize2

The first script uses Technique 1 to run pythonw.exe directly on web servers. The author admits to being a novice Python programmer. The last 3 rows of data are not used in the regression, but are used to carry out a small sensitivity analysis of the confidence interval for the *predicted* value of the Qx variables (see Example 1h). The final row is used to fill in QTM, QTL, and QTU. Note that the CTA 1 reference has examples of regression algorithms that generate confidence intervals for both estimated and predicted values.

```
import sys
import statsmodels.formula.api as smf
import statsmodels.api as sm
import pandas as pd
from statsmodels.sandbox.regression.predstd import wls_prediction_std

#make a panda dataframe from input dataset csv file
dataset1 = pd.read_csv(sys.argv[1])
#dataset2 leaves out the last 3 lines of data used for ci
dataset2 = dataset1.head(-3)
#dataset3 is the last 3 lines of data used for ci
dataset3 = dataset1.tail(3)
#run statistical model with dataset2
mod = smf.ols('energyuse ~ housesize1 + housesize2', data=dataset2)
results = mod.fit()
```

```
#print a summary of model results
print(results.summary());

#get predicted values for dataset3
ci = results.predict(dataset3)
#make an exog parameter from a nobs X k array
colindex = len(dataset3.columns)
X = dataset3.iloc[:,1:colindex]
#add the intercept
X = sm.add_constant(X)
#get confidence intervals for predicted value of dataset3
prstd, iv_l, iv_u = wls_prediction_std(results, exog=X, alpha=.05)
indexcount = len(dataset3.index) - 1
print(round(ci[0], 4), round(iv_l[indexcount - 2], 4), round(iv_u[indexcount - 2], 4))
print(round(ci[1], 4), round(iv_l[indexcount - 1], 4), round(iv_u[indexcount - 1], 4))
print(round(ci[2], 4), round(iv_l[indexcount], 4), round(iv_u[indexcount], 4))
```

Calculator Properties: Refer to Example 1, R Scripts, in Appendix A in the R reference to see how Calculator properties are set. Both algorithms work identically (i.e. with allowances for subalgorithm3).

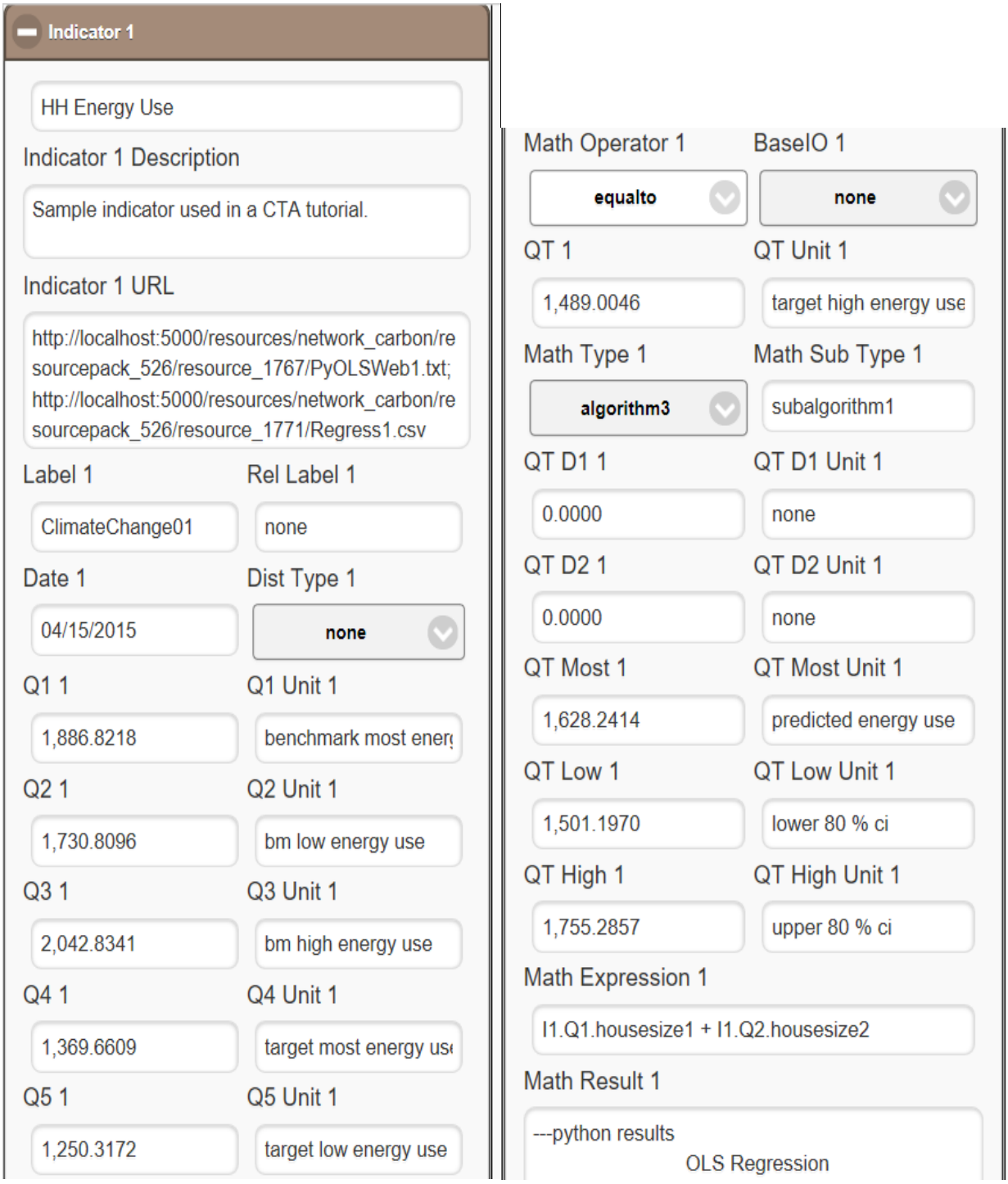
[Version 2.1.6 had to refactor this algorithm because the python executable no longer accepted the dataset URL argument passed to it. The cause is believed to be the 216 redirection to https URLs, but documentation was not found confirming this limitation. The dataset had to be saved in the file system/blob prior to be passed as an argument to py.]

The following results are similar to the results in Example 1h. The images demonstrate that numbers should be formatted correctly inside the Python script because the algorithm does not format any of the statistical results. The algorithm uses the last row to complete an Indicator's

QTM, QTL, and QTU properties (predicted energy use). The Technology Assessment 2 tutorial explains that large mathematical results should always be stored in Math Result URLs.

The image also shows that a DevTreks convention with several algorithms is to use the last 3 rows of a dataset to generate confidence intervals for 3 predicted ranges of values. Version 2.1.4 upgraded the rules for several algorithms to use these 3 predicted ranges in the same manner as introduced in Example 5 in the Social Performance tutorial -to fill in Indicator metadata properties for most likely, low, and high, predictions for 3 sets of Indicator.metadata properties. The analyst must fill in appropriate units for the 3 predictions (i.e. Example 5's benchmarks, targets, and actuals).

Web server results (run using Version 2.1.6):

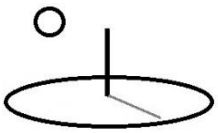

Indicator 1
HH Energy Use
Indicator 1 Description
Sample indicator used in a CTA tutorial.
Indicator 1 URL
http://localhost:5000/resources/network_carbon/resourcepack_526/resource_1767/PyOLSWeb1.txt;
http://localhost:5000/resources/network_carbon/resourcepack_526/resource_1771/Regress1.csv
Label 1
ClimateChange01
Rel Label 1
none
Date 1
04/15/2015
Dist Type 1
none
Q1 1
1,886.8218
Q1 Unit 1
benchmark most ener
Q2 1
1,730.8096
Q2 Unit 1
bm low energy use
Q3 1
2,042.8341
Q3 Unit 1
bm high energy use
Q4 1
1,369.6609
Q4 Unit 1
target most energy us
Q5 1
1,250.3172
Q5 Unit 1
target low energy use
Math Operator 1
equalto
BaseIO 1
none
QT 1
1,489.0046
QT Unit 1
target high energy use
Math Type 1
algorithm3
Math Sub Type 1
subalgorithm1
QT D1 1
0.0000
QT D1 Unit 1
none
QT D2 1
0.0000
QT D2 Unit 1
none
QT Most 1
1,628.2414
QT Most Unit 1
predicted energy use
QT Low 1
1,501.1970
QT Low Unit 1
lower 80 % ci
QT High 1
1,755.2857
QT High Unit 1
upper 80 % ci
Math Expression 1
I1.Q1.housesize1 + I1.Q2.housesize2
Math Result 1
---python results
OLS Regression

Math Result 1

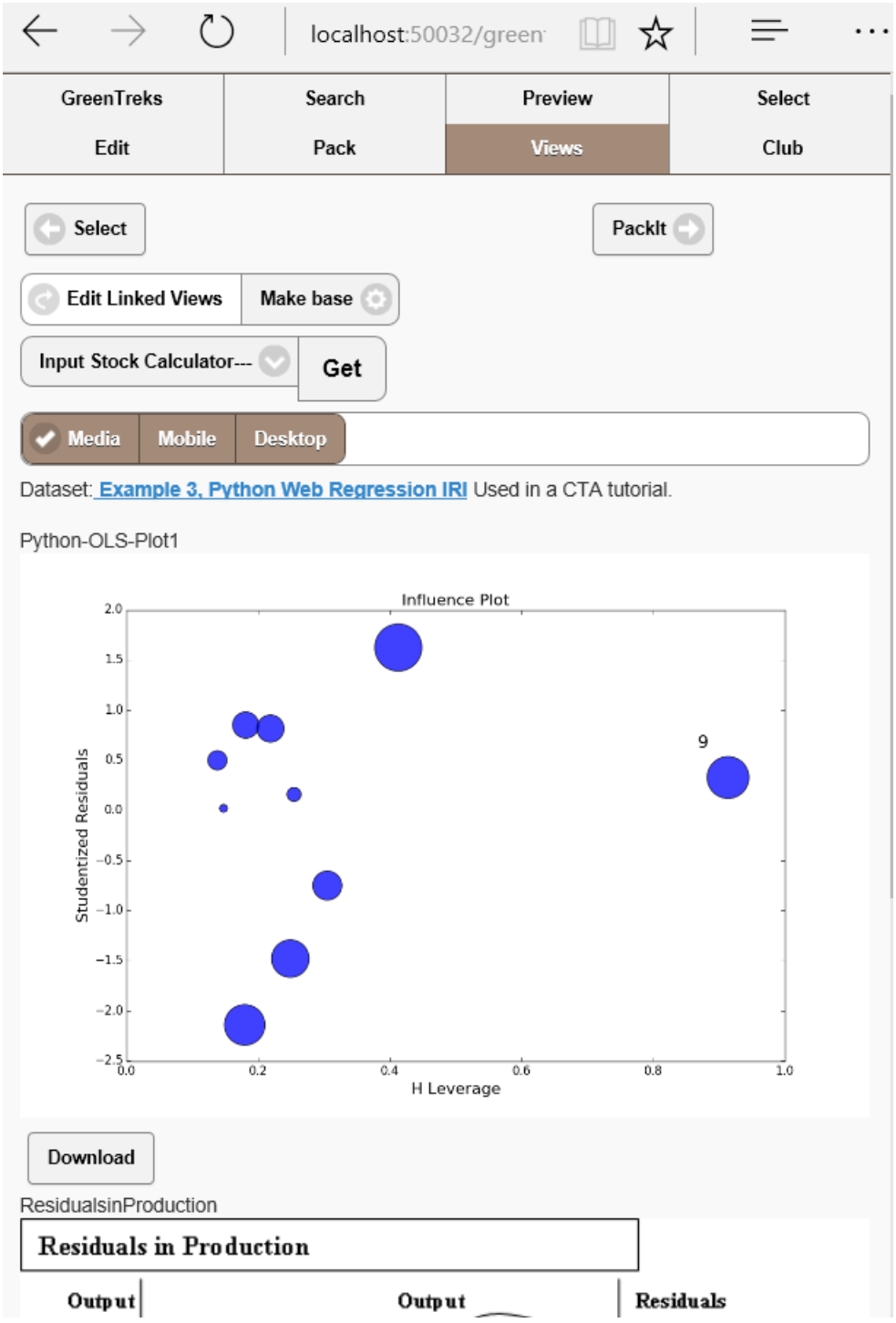
```
---python results
                            OLS Regression Results
==============================================================================
Dep. Variable:              energyuse   R-squared:                       0.982
Model:                            OLS   Adj. R-squared:                  0.977
Method:                 Least Squares   F-statistic:                     189.7
Date:                Wed, 12 Sep 2018   Prob (F-statistic):           8.00e-07
Time:                        16:34:21   Log-Likelihood:                -50.865
No. Observations:                  10   AIC:                             107.7
Df Residuals:                       7   BIC:                             108.6
Df Model:                           2
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [95.0% Conf. Int.]
------------------------------------------------------------------------------
Intercept  -1216.1439    242.806     -5.009      0.002    -1790.290  -641.998
housesize1     2.3989      0.246      9.758      0.000        1.818     2.980
housesize2    -0.0005   5.91e-05     -7.618      0.000       -0.001    -0.000
==============================================================================
Omnibus:                        1.129   Durbin-Watson:                   2.079
Prob(Omnibus):                  0.569   Jarque-Bera (JB):                0.859
Skew:                          -0.587   Prob(JB):                        0.651
Kurtosis:                       2.172   Cond. No.                     7.04e+07
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 7.04e+07. This might indicate that there are
strong multicollinearity or other numerical problems.
1628.2414 1501.197 1755.2857
1886.8218 1730.8096 2042.8341
1369.6609 1250.3172 1489.0046
```

The following image demonstrates that graphical explanations of analytic results can be produced directly in Python and then displayed using the Media View.



DevTreks –social budgeting that improves lives and livelihoods



Example 2. Algorithm 3. SubAlgorithm2: WebApi Python Scripts, Small Jobs

URLs:

The URLs from Example 1 are also used to test this algorithm. The only difference is to change the Indicator.MathType to algorithm3 and Indicator.SubMathType to subalgorithm2.

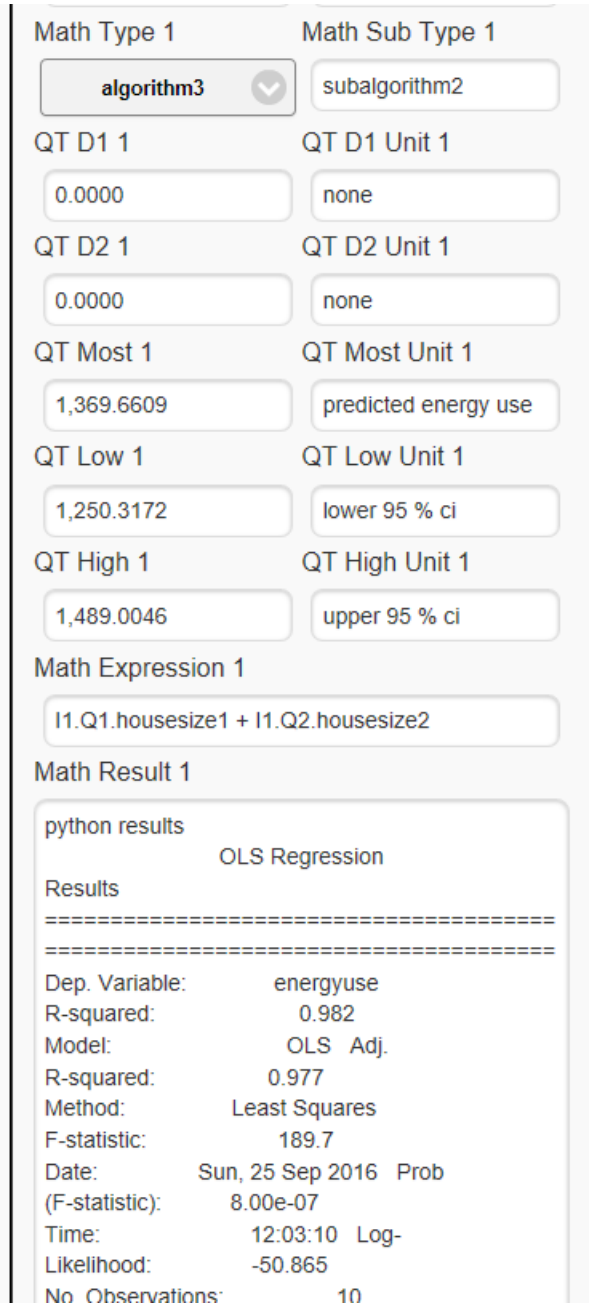
This subalgorithm is appropriate when the Python script and associated dataset run relatively fast –the results return to the client relatively quickly. The term "relatively quickly" should be defined by the developer in an exact way.

A cross-platform, ASP.Net Core 1 Web Api web app, DevTreksStatsApi, is deployed to another web server. The Web API implements a REST method that accepts http POST requests that contain a JSON object. The JSON object is serialized into a regular POCO object, StatScript, with properties that include a script file URL, a data file URL, and an output file URL. Paths to the web server's Pythonw.exe has been added to StatScript using dependency injection which has been configured in the Startup.cs class and appsettings.json.

The server processes the http POST by running the statistical scripts. The statistical results are saved in the output csv URL. The client, DevTreks, waits for a successful response from the WebApi call. When received, DevTreks deserializes the json response into a StatScript POCO object and fills in an Indicator's QT properties from DataScript.StatisticalResults. If the MathResult contains a legitimate Resource URL, the results will be saved in the URL. If not, the results will be stored in the MathResult itself. The recommended way of storing statistical results is in MathResult URLs. Python runs scripts much slower than R.

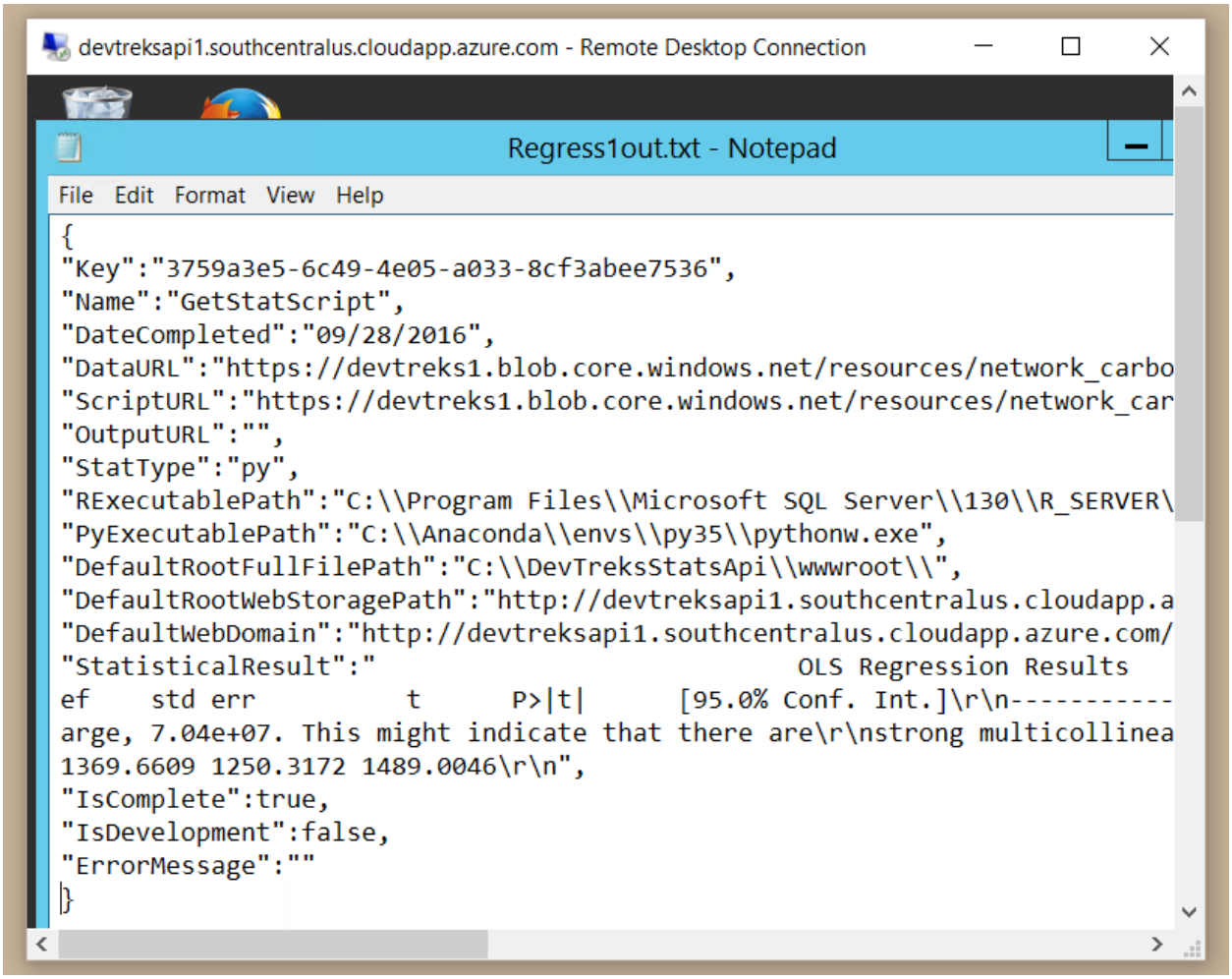
The WebApi host stores the data URL, script URL, and output URL, files in the standard "wwwroot\resources\temp" directory of the WebApi site. The WebApi folder and files must be maintained by DevTreks adminstrators by periodically deleting old directories. A future release may automate this task (i.e. see the Controller Action that runs DELETE http commands).

The following results show that this subalgorithm produces the same results as Example 1. The CTA02 reference for R discusses difficulties in debugging this using localhost sites and with Version 2.0.6 cloud sites.



Cloud server results:

The following image displays the json response generated from the WebApi deployed on a Windows Data Sciences Virtual Machine. This response is deserialized by a DevTreks client to produce the previous image's results. Linux will also be tested.



Example 3. Algorithm 3. SubAlgorithm3: WebApi Python Scripts, Large Jobs

URLs:

See the corresponding subalgorithm3 in the CTA 2 R reference as an example of what's needed.

The same proviso holds:

This example may be put on hold. It's not clear that a small ngo should be demonstrating machine learning techniques that investors are currently pouring large sums of money into. On the other hand.

Example 4. Algorithm 3. SubAlgorithm4: Python Project Custom Packages

URLs:

Example needed

Example 5. Algorithm 3. SubAlgorithm5: Python Mathematical Libraries

URLs:

SciPy and NumPy examples needed