MnV tool - User Guide

Notebook:	Guides		
Created:	8/20/2018 9:15 AM	**Updated:**	9/5/2018 10:37 AM
Author:	ckoshnick@ucdavis.edu		
URL:	http://localhost:8889/notebooks/Box%20Sync/UCD_ECO_coding/test%20project/Project%20MnV/MnV%20PES%20p...		

Latest MnV version - 1.5
This guide last updated - Aug 29, 2018

Getting Anaconda

It is recommended that anyone using the ECO tools gets the Anaconda distribution of python.
https://www.anaconda.com/download/
This tool is written to be Python 2.7 and Python 3.6 compatible, but python 3.6 is recommended. All default installation parameters should be used.

What is a notebook

Anaconda is nice since it bundles with it iPython, Jupyter Notebook, and Spyder. The tool is designed to be mainly run from a Jupyer Notebook. What is a Jupyter Notebook, found out here. http://jupyter.org/
In general it is a convenient way to execute small snippets of code within discrete cells so that data can be manipulated, visualized, and manipulated again without re-running an entire python script.

Python Basics website

If you're interested in learning some python basics to make looking at the notebook a little more intuitive have a look at http://learnpython.org/
If you can complete topics within "Learn the basics" and "Data Science Tutorials" that would be splendid!

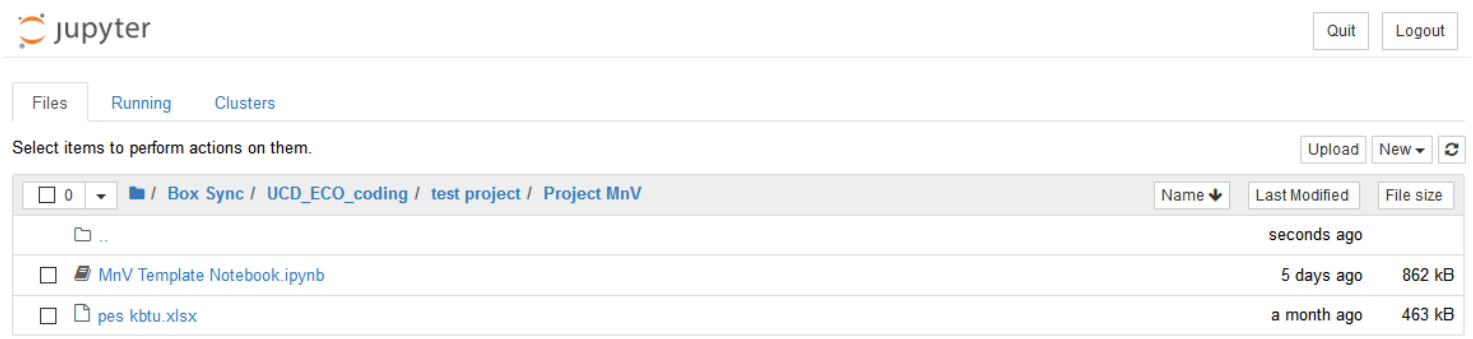
This guide will be a section by section explanation to give the user enough information to manipulate their data, and models properly and to within the limits of the tool. For a complete reference to the MnV class please see MnV tool - Detailed Guide.

Getting the tool via GitHub

- Easy way -- Visit https://github.com/Ckoshnick/MnV-Tool and click the "Clone or download" button, choose Download ZIP. Save and unzip it in your desired location. (don't forgot where it is!)
- Medium -- Get the github desktop app https://desktop.github.com/. use the repository link to add the repositories https://github.com/Ckoshnick/MnV-Tool.git
- Hard way -- Learn how to use normal .git

Launching the Tool

Place **A COPY** of the "MnV Template Notebook.ipynb" (found in the MnV-tool folder) into the project's MnV folder then navigate there within Jupyter. From Anaconda Navigator, open Jupyter Notebook by clicking "Launch". You should see the Jupyter Home page that looks something like the browser window below. Navigate to the project MnV folder and launch the template notebook. Once inside give it a new name.



Using the tool - a step by step guide

Section 0 - Imports

Make sure the **toolPath** directory is correct for your computer. Then run this cell (native hotkey Ctrl + Enter), if it was successful you should see the version number printed below. It may be convenient to save a new copy of the template for yourself so this variable stays correct.

In [3]:
```
## Make sure this tool Path Matches the path for tool on YOUR computer
toolPath = r'/Users/koshnick/Box Sync/UCD_ECO_coding/MnV-Tool'

import sys
import pandas as pd
import matplotlib.pyplot as plt
from IPython.core.display import display, HTML
display(HTML("<style>.container { width:90% !important; }</style>"))
sys.path.append(toolPath)
import mnv

pd.set_option('display.max_columns', 10)
pd.set_option('display.width', 500)

print(mnv.version)
```

```
Version 1.5
```

Section 1 - Data Loading

Local Data

Change filePath appropriately. If loading data from PI skip this section (or comment it all of the lines by selecting them and pressing CTRL + /)

Local Data Loading

In [5]:
```
filePath = 'pes kbtu.xlsx'
data = pd.read_excel(filePath, header=0, index_col=0, parse_dates=True, infer_datetime_format=True)
print(data.columns)
```

```
Index(['PES_ChilledWater_Demand_kBtu', 'PES_Electricity_Demand_kBtu', 'PES_Steam_Demand_kBtu'], dtype='object')
```

PI data

Use the pi_client search function to find tags of interest. If loading data locally you can skip this section (or comment it out by using CTRL + /).

PI Data Loading

In [ ]:
```
from PI_client import pi_client
pi = pi_client()

tags = pi.search_by_point('*PES*kbtu')[0]
# tags += pi.search_by_point('*shields*kbtu*')[0]

print(tags)
```

When satisfied run the next cell to pull the tags. Specify the date ranges, and the interval. Demand data should be taken as "calculated". If loading data locally you can skip this section (or comment it out by using CTRL + /).

Pull tags

In [ ]:
```
startDate = '2017-01-01'
endDate = '2018-01-01'
interval = '1 hour' #Can be "minute" "hour" "day"
calculation = 'calculated' # Redundant?

data = pi.get_stream_by_point(tags, start=startDate, end=endDate, interval=interval, calculation=calculation)
```

Section 2 - Data Cleaning

The data is loaded into a class known as the "data_keeper" where it will be modified by the user inputs before being passed onto the "model" class.

Input Section

Data Section

In [ ]:
```
dataParams = {'column': 2,
              'IQRmult' : 3.0,
              'IQR' : True,
              'floor': 0,
              'ceiling': 40000,
              'resampleRate' : 'D', #'D' for daily 'H' for hourly
              'OATsource' : 'file', #'self' or 'file'
              'OATname' : 'OAT', #Name of OAT column if OATsource is 'self'} #only needed with sliceType : 'ranges'
              'sliceType' : 'ranges', #half, middate, ranges
              'midDate' : '2017-01-01', #only needed with sliceType : 'middate'
              'dateRanges' : ['2016-06-01','2017-06-01','2017-08-01','2017-12-01'], #only needed with sliceType : 'ranges'
             }

dk = mnv.data_keeper(data, dataParams)
dk.default_clean()

# Plots
dk._outlier_plot()
dk._resampled_plot()
dk._pre_post_plot()
```

Variable explanation for dataParams

'column': 2 - column index in the data that is specified as the "data of interest for modeling" Following the example above 'PES_Electricity_Demand_kBtu' would be 'column': 1 [since it is indexed from 0]
IQR : True or False - specifies if you want to use the Inner Quartile Range outlier selection method.
IQRmulti = 3 - value that is multiplied by the IQR to determine the final outlier range above the 75th and below the 25th percentiles
'floor': 0 - the minimum value the data should ever take on (for outlier detection. This may be modified UPWARD only if the IQR function demands)

'ceiling': 40000 - the maximum value the data should ever take on (for outlier detection. This may be modified DOWNWARD if the IQR function demands)
'resamplerate': 'H' - force a sampling rate on the raw data. A Default '1H' sampling with be applied after outlier detection (necessary for HDH and CDH calculations)
'OATsource': 'file' - Determines whether the OAT data will be pulled from the Master 'file' or is supplied along with the data fed into the data_keeper 'self'
'OATname': 'OAT' - The exact name of the column that contains OAT data ONLY needed if the OAT is supplied as 'self'
sliceType: 'ranges' - Determines how the data is sliced into 'pre' and 'post' periods.

- half: split the data 50/50
- middate = split the data at one particular date [start:mid] and [mid:end] -- specified under 'midDate'
- ranges = split the data into specific ranges pre = [s1:e1] post = [s2:e2] -- specified under 'dateRanges'

Functions

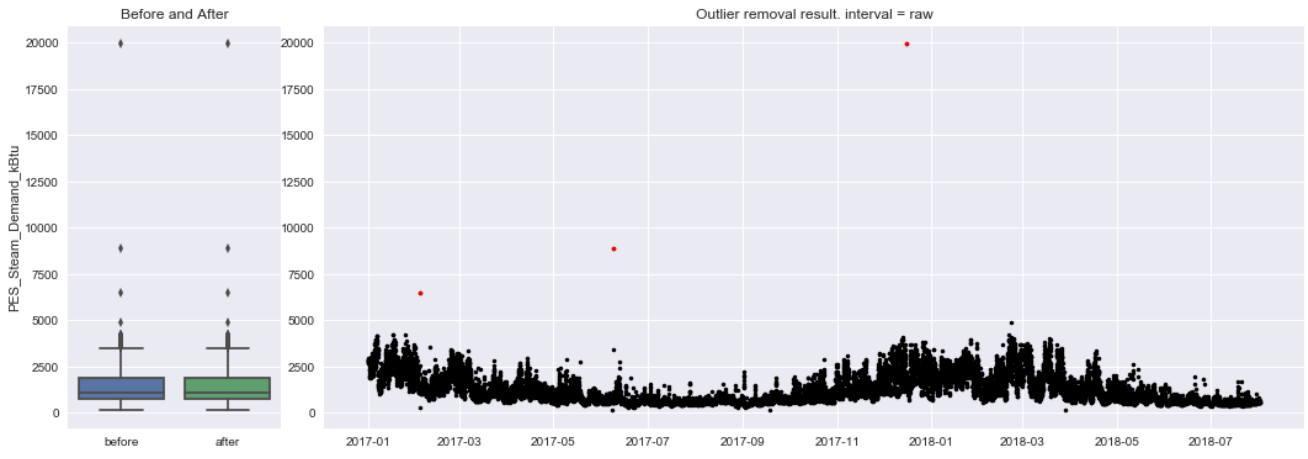
dk = mnv.data_keeper(data, dataParams) -- passes the data, and parameters into the data_keeper class
dk.default_clean() -- runs a pre-baked set of functions on the data to remove outliers, clean, resample, add HDH/CDH, and slice the data into pre/post

Typical Output

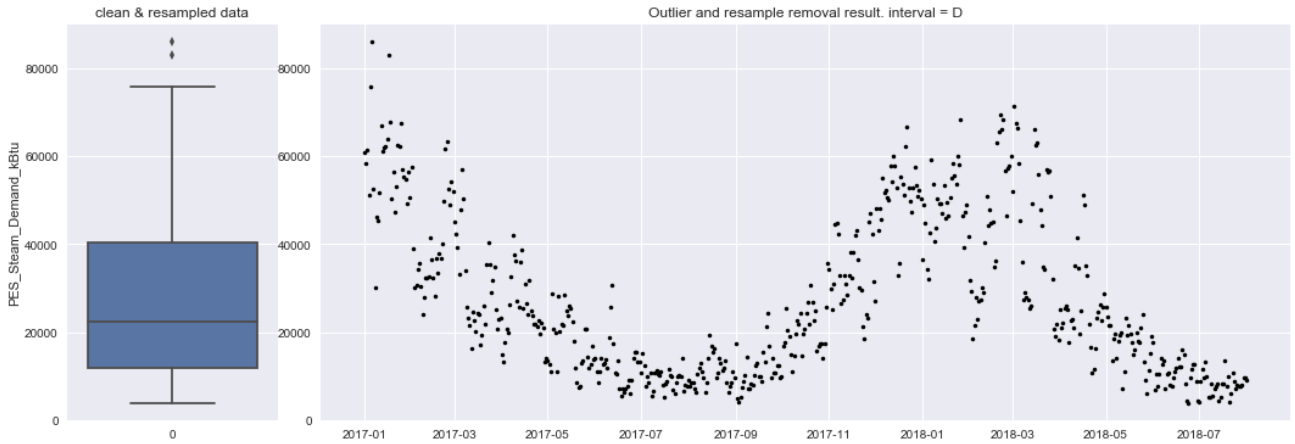
```
Q(75%): 1857.14 Q(25%): 739.21
IQR value is 1117.93
IQRupper 5210.95 ; IQRlower -2614.6
Ceiling adjusted by IQR - Now 5210.95
```

Q(75%) and Q(25%) are the percentiles in the raw data.
IQR - is the calculated inner quartile range
IQRupper and IQRlower are the calculated boundaries, if they are within floor/ceiling then those values will be adjusted.

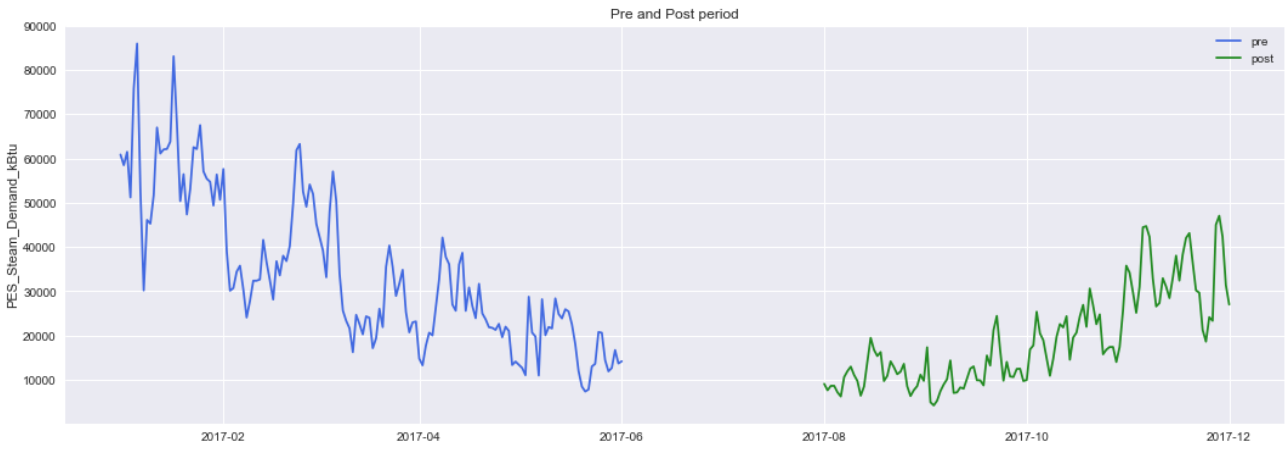
The below outlier plots show which data points (colored in red) that are being removed ...



... and the data after re-sampling.



The next plot shows the data in the 'pre' and 'post' periods. In this pretend example there was a 2 month gap between June and Aug when the project was being implemented.



Now that the data has been processed, it will be passed onto the 'model' sections.

Section 3 - Many Linear Models

Many Linear Models

In [ ]:
```
modelParams = {'var': ['CDH', 'HDH', 'month', 'hour', 'weekday'],
               'testTrainSplit': 'random',
               'randomState': 4291990,  #Good idea to use a seed value so folds are identical
               'testSize': 0.20,
               'commodityRate': 0.0651,
               'varPermuteList': ['', 'C(month)', 'C(weekday)']
              }

allmod = mnv.many_ols(dk.pre, dk.post, modelParams)
allmod.run_all_linear()

print(allmod.statsPool[0:5])  # Display top stats
allmod.plot_pool(2)  # Visualize top models
```

Variable explanation for modelParams

'var': ['CDH', 'HDH','month','hour','weekday'] - A collection of variables that will be used to construct the linear model below (may be modified in this section based on the 'Many Models' result

'testTrainSplit': 'random', - Determines how the train/test data will be split up. 'random' pulls the data apart in random splits, where 'simple' does an simple [start:end] split based on the testSize
'randomState': None, - When using random train/test this variable can be supplied as a seed number for the random number generator
'testSize': 0.2, - The fraction of data that will be included in the test set [0 < testSize < 1]
'commodityRate': 0.0651, - The USD rate for the commodity being modeled, be careful to mind the units
'paramPermuteList': ['', 'C(weekday)', 'C(month)', 'C(hour)'] - The collection of variables that will be randomized to create the 'best model' (ranked by AIC)

Functions

allmod = mnv.many_ols(dk.pre, dk.post, modelParams)) -- passes the data, and parameters into the many_ols class
allmod.run_all_linear() -- runs all of the models using the variables specified above

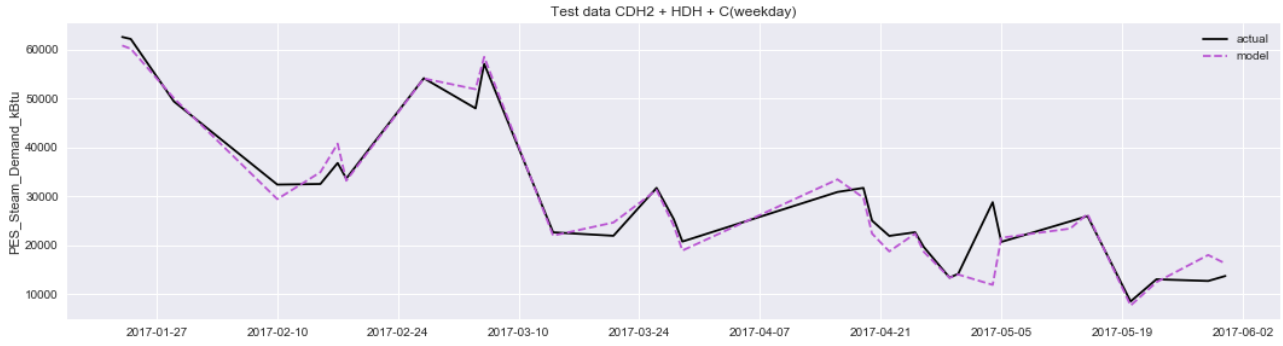
Viewing the results

print(allmod.statsPool[0:5]) - Shows the stats for the top <5> models
allmod.plot_pool(5) - Shows the plots for the top <5> models

The following table are some of the statistical results of the top 5 models (ranked by AIC).

```
Entering run_all_linear()...
run_all_linear() complete
        AIC       AR2        R2     cvrmse postDiff                                 var
14  2277.26  0.972092  0.972789  0.0825551   115467              CDH2 + HDH + C(weekday)
27  2283.31  0.969156  0.970955  0.0867123   623702           HDH + C(month) + C(weekday)
2   2289.56  0.966432  0.967271  0.0894101   219631               CDH + HDH + C(weekday)
15  2295.95   0.96967  0.971692  0.0882544   539736  CDH2 + HDH + C(month) + C(weekday)
26  2302.49  0.963664   0.96427  0.0955616   142065                     HDH + C(weekday)
```

Model Plot - A visualization of how closely the model matches the data in the TEST period. (this is the period in which the R2 etc... statistics are derived). In a perfect model actual and model line would be exact matches. The model displayed here is an example of a very good match.



Stats Plots - from left to right: QQ-plot. Actual vs Model scatter, Residuals vs Actual scatter.

QQ-plot

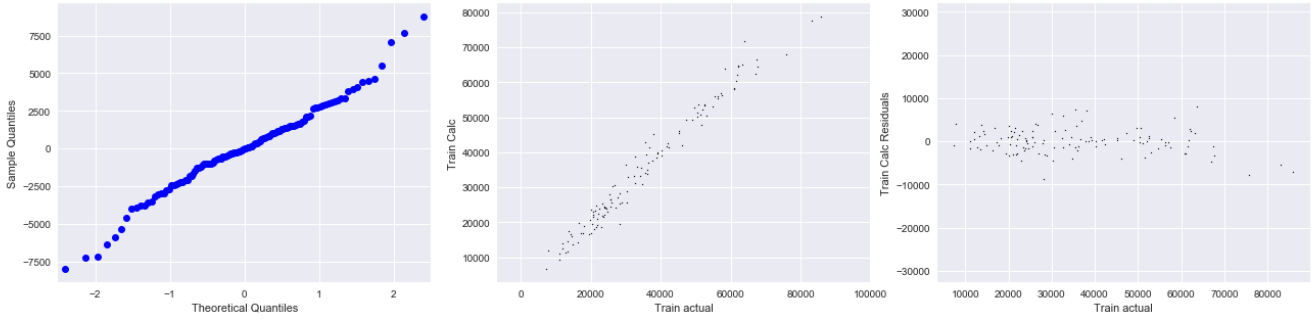
- Checks to see if prediction quality is consistent accross the entire range of the data. Sometimes the ends tend to bend indicating that the prediction quality deviates at the extreme ends. A straight line is ideal.

Train Calc vs Train Actual

- Shows the spread in values between actual and predicted. A straight line with slope 1 is ideal here.

Train Calc Residuals vs Train Actual

- The residual values from the above plot. A flat line is ideal here.



Section 4 - Single Linear Model

Single Linear Model

In [ ]:
```
# Set the modelParam "var" to the 'best' model from run_all_linear()
modelParams['var'] = allmod.statsPool.iloc[0]['var']

mod = mnv.ols_model(dk.pre, dk.post, modelParams)
mod.calculate_kfold()
mod.calculate_vif()

print(mod.vif[mod.vif['VIF'] > 5])

print('\nKfold')
print(mod.kfoldStats)

mod.model_plot()
plt.show() # Show plot before Stats summary
mod.Fit.summary()
```

In this section, a model is being created using the params determined to be the best from the above "many_ols" class. The user can modify the modelParams dictionary to substitute in their own parameters if desired.

Sets the params for the SINGLE model to the top parameters based on the many_ols model results. Comment out the following row to remove this behavior.
modelParams['params'] = allmod.statsPool.iloc[0]['params']

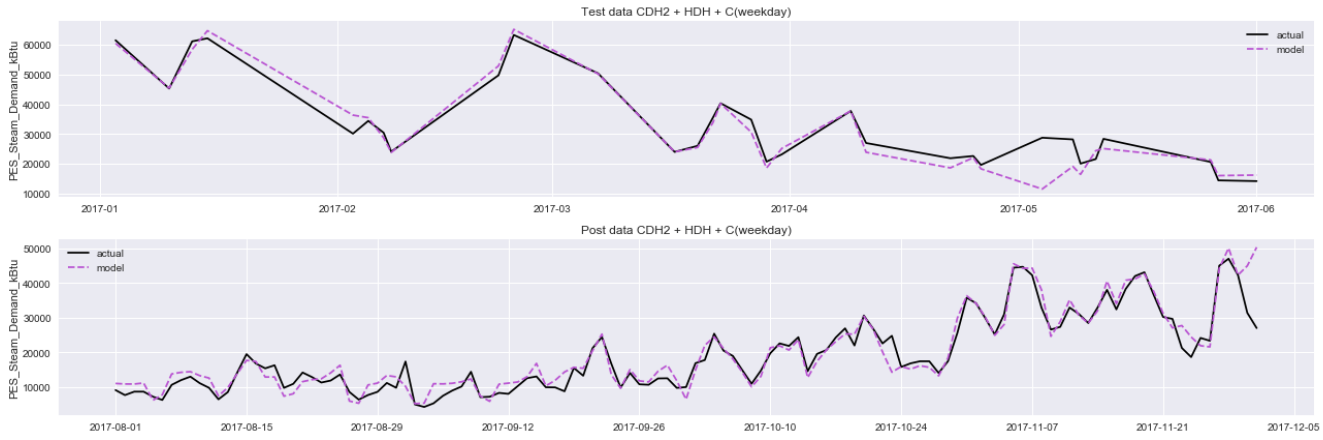
Next we calculate two more sets of values to ensure the model that is chosen passes these test.

The first is the Variance Inflation Factor (VIF) and it helps to ensure that the predictor variables are not too highly correlated with one another. The VIF table is filtered for results > 10. Variables with too high of a value are considered unacceptable because they are linearly correlated with one another.
K-folding is a process by which other samplings of the test/train data is made to determine if there is a strong bias in the model based on the training data. The number of folds is based on 1/testSize, so a testSize of 0.1 will give 10 folds, where each 10% of the data is used as the test set once.
To pass the K-folding the mean squared error (mse) of each version of the model should not deviate when changing the test set. The relative errors are plotted in the table to see the % spread around the mean mse. Values within +/ 10% of the mean are acceptable. __TODO__ what is acceptable?

```
              VIF
const    6.888057

Kfold
           R2        AR2           mse
0    0.976270   0.975662  7.583778e+06
2    0.967192   0.966358  8.799817e+06
1    0.963548   0.962614  9.455887e+06
4    0.968208   0.967399  1.042848e+07
3    0.965983   0.965118  1.050064e+07
0               81.0
2               94.0
1              101.0
4              111.0
3              112.0
<mse>     9353721.0
Name: rel. pct., dtype: float64
```

This is another version of the model plot where the model is also compared to the POST data in the second plot. This is the preliminary way to assess the results of the project.



The data table below shows all of the statistical quantities calculated for the linear model along with the fitting values for the parameters.

Out[12]:

OLS Regression Results

Dep. Variable:	PES_Steam_Demand_kBtu	**R-squared:**	0.976
Model:	OLS	**Adj. R-squared:**	0.976
Method:	Least Squares	**F-statistic:**	1605.
Date:	Fri, 24 Aug 2018	**Prob (F-statistic):**	7.73e-95
Time:	15:18:31	**Log-Likelihood:**	-1128.1
No. Observations:	121	**AIC:**	2264.
Df Residuals:	117	**BIC:**	2275.
Df Model:	3		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	6581.2539	631.915	10.415	0.000	5329.779	7832.729
C(weekday)[T.1]	5077.6803	539.659	9.409	0.000	4008.913	6146.447
CDH2	-0.0050	0.009	-0.535	0.594	-0.023	0.013
HDH	122.4287	2.002	61.152	0.000	118.464	126.394

Omnibus:	4.628	**Durbin-Watson:**	1.457
Prob(Omnibus):	0.099	**Jarque-Bera (JB):**	4.510
Skew:	-0.291	**Prob(JB):**	0.105
Kurtosis:	3.746	**Cond. No.**	9.50e+04

Section 5 - Savings

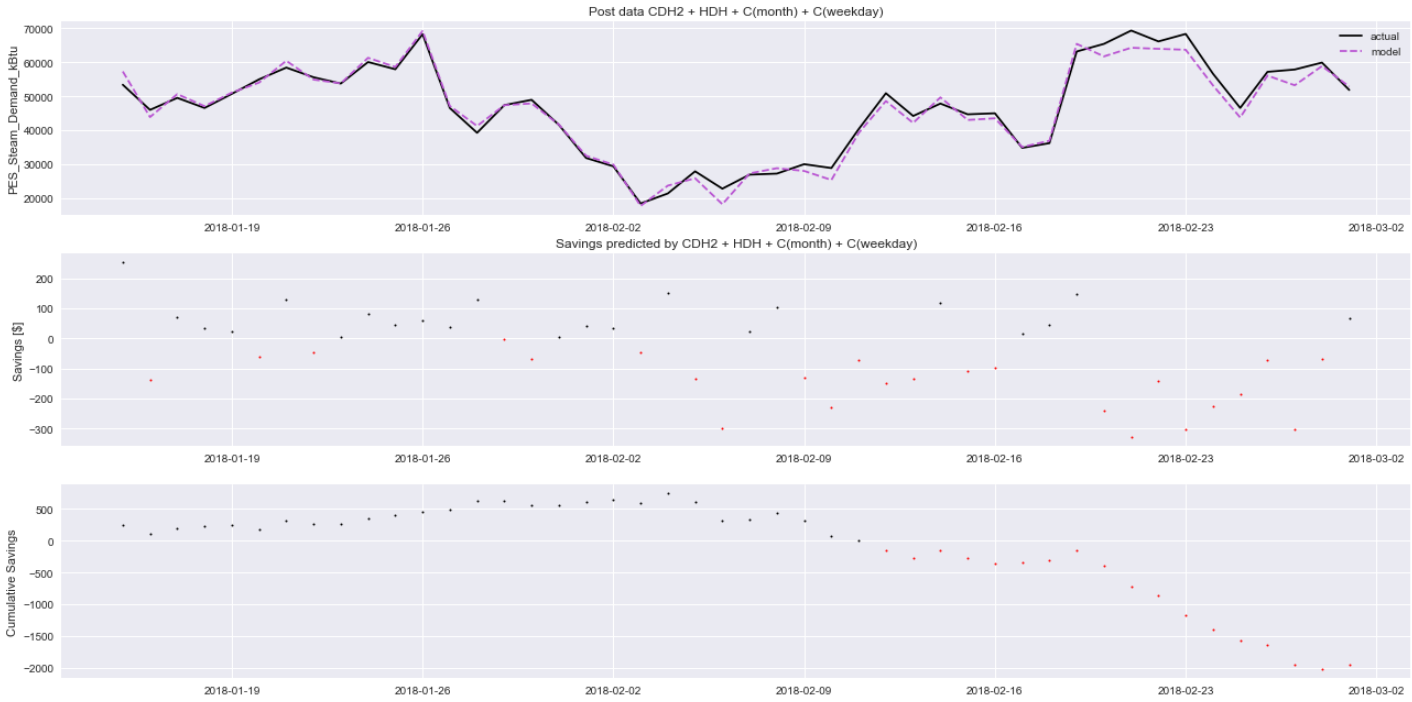
The savings section shows the difference between the model and the actual data.
Again we have the model vs actual POST data.
In the middle plot the individual savings/spending for each data point is shown.

Savings

In [ ]:
```
print("Savings = $" +str(round(mod.postDiffSum * mod.params.commodityRate, 1)))
mod.savings_plot(yaxis='dollars')
```

Output shows the sum of the ACTUAL **difference** in energy or dollars over the post period. and some plots showing independent 1 day difference, and the cumulative sum.

```
Savings = $-1948.4
```



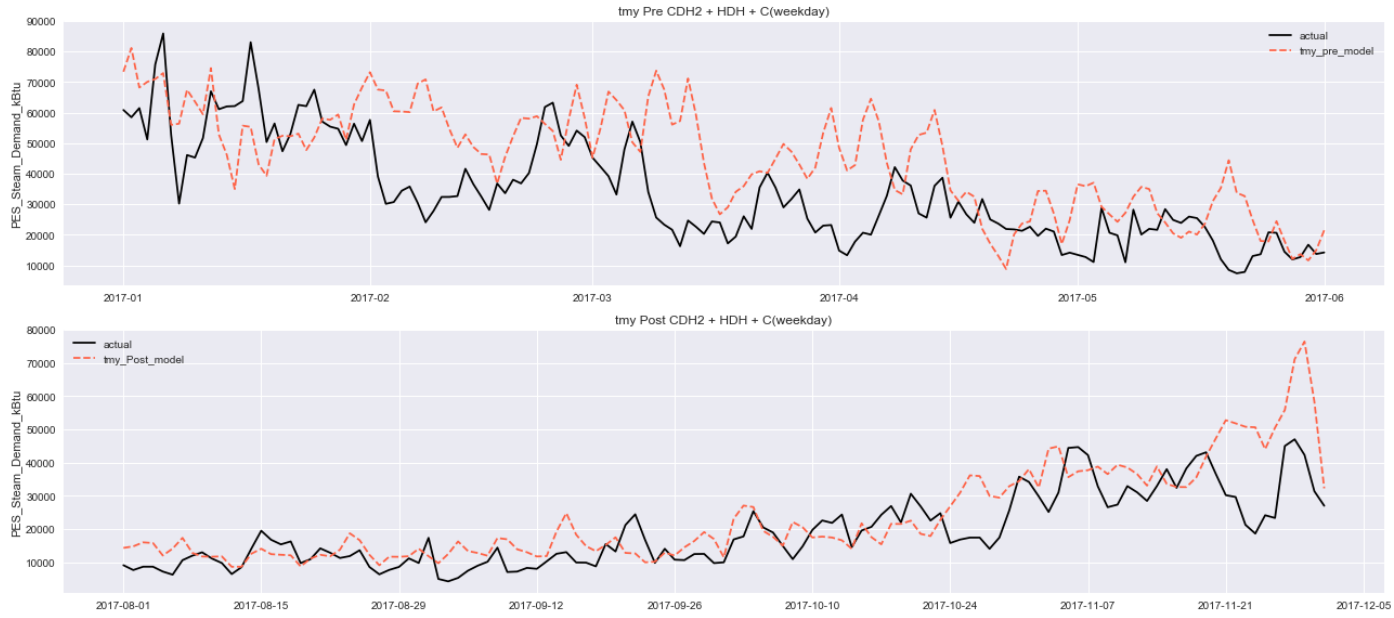
Section 6 - TMY predictions

The TMY section is to provide a ROUGH estimation of the future difference in performance a project. The difference values that are reported are 3 time segments combined in two ways.
The Actual accounts for the difference up until now, the TMY predicted difference from the end of the actual data to the end of the next fiscal year, and the TMY predicted difference from the end of the post period to 1 year after the start of the post period.
These values are reported in the savings summary

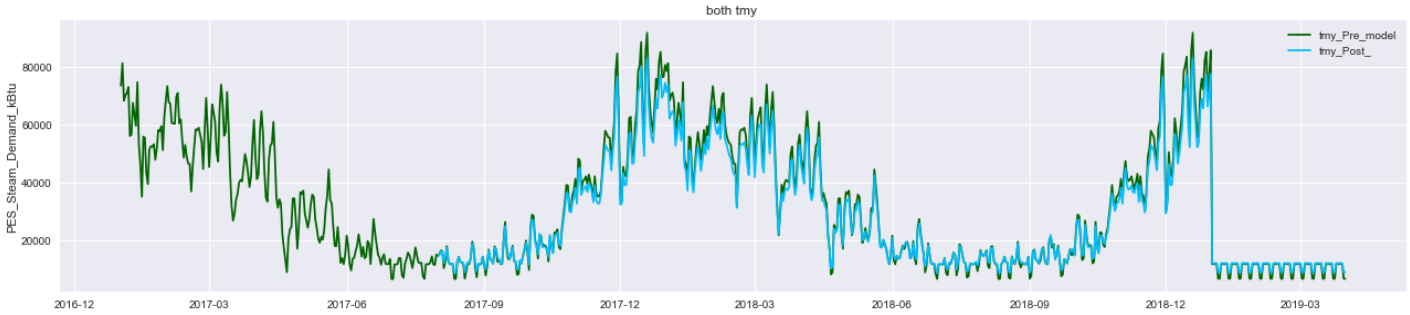
TMY

In [ ]:
```
mod.calculate_tmy_models()
mod.plot_tmy_comparison()

print("Savings = $" +str(round(mod.data['tmyDiff'].cumsum()[-1]*mod.params.commodityRate, 1)))
mod.compile_savings()
```



more plots



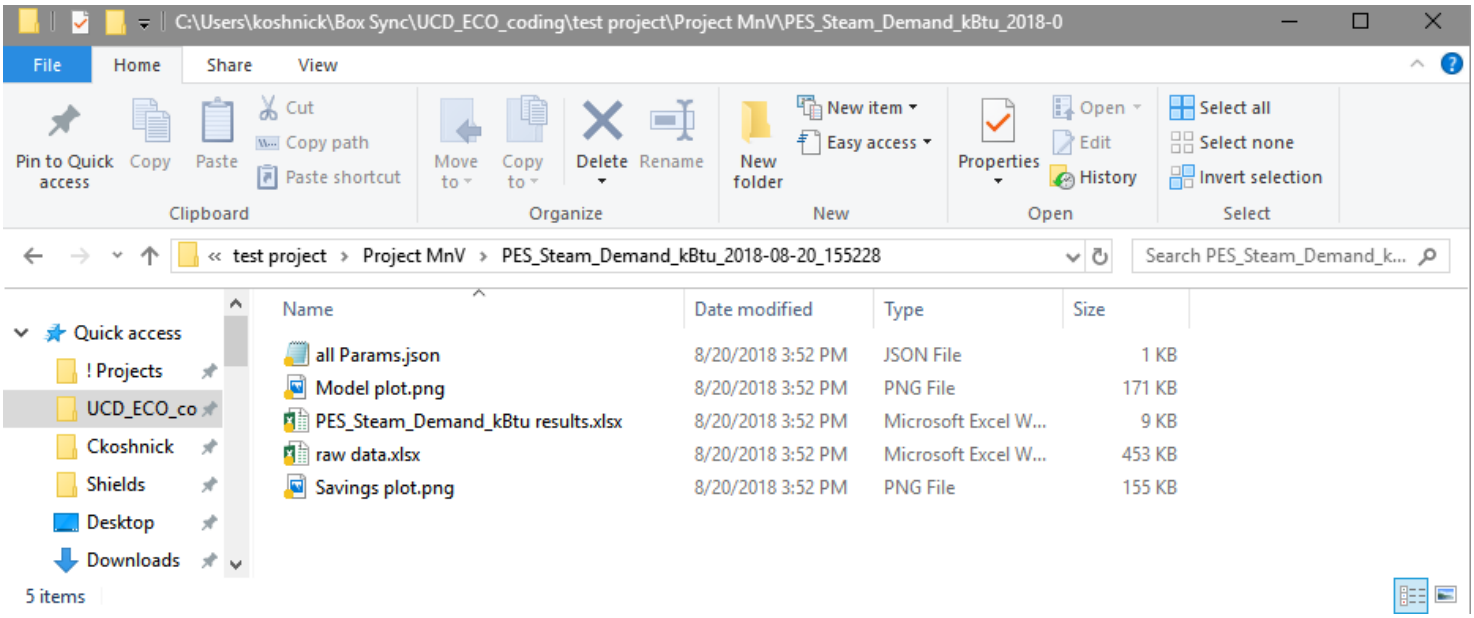
Section 7 - Archive

Creating the archive allows the modeling work that was done to be saved an accessed later. Another notebook has been developed to open a saved model, pull new data from PI and assess the continued performance of the project against the model that was generated. The archived work can also be used to generate a report of the projects outcome once enough post period data has been collected.

Creating an Archive

```
In [ ]: mnv.create_archive(dk, mod, saveFigs=True, copyRemodel=True)
```

Choosing the saveFigs=True option will save .png version of selected plots from the notebook.



All of the output information from the model can be viewed in the folder that is auto-generated with name <column name> <yyyy-mm-dd_hhmmss>