PI World 2019 Lab

Exploring AF Analytics for Advanced Analysis and Prediction

OSIsoft, LLC
1600 Alvarado Street
San Leandro, CA 94577

Published: March 22, 2019

Table of Contents

Contents

Course Learning Objectives and Toolkit

Problem Statement

Secondary transformers deliver electric power to homes and businesses. You have probably seen one on a pole or pad in your own neighborhood. Utilities monitor energy loads on these transformers because exceeding designed capacity can cause them to fail, resulting in loss of service or liability from explosion. Although secondary transformers are not instrumented, their energy loads can be computed using AF analytics to sum up readings taken from meters placed at each service delivery point they supply. This summing up, or rollup, analysis has been completed for the sixty transformers we will be analyzing in this lab.

Since transformer load monitoring is important, our objective will be to develop a simple statistical model for predicting the power load on each of the sixty transformers operating on one circuit of our power distribution network. The predictive model will be operationalized to forecast network loading to anticipate loading problems before they occur. The model will be developed from utility metering and weather forecast data stored in the PI archive and contextualized in PI Asset Framework (AF).

Learning Objectives

AF Analytics offers distinctive advantages when preparing time-series data for advanced analytics. It can play an essential role in testing, evaluating and operationalizing developed models. In addition, the openness of the PI System supports several different data access methods, meeting the needs of data engineers or scientists.

In this lab, we'll complete three exercises demonstrating how AF Analytics can serve as an advanced analytics workbench. For comparison, we will employ different data access methods, PI Integrator for Business Analytics and PI SQL Client. (Exercises are replicated in the Appendix using PI OLEDB Enterprise and the PI Web API.) Predictive models using PI Data will be developed in Python, a readily available open source application that is widely used for statistical modelling and analysis.

The lab consists of three exercises. The same statistical model is developed in each exercise, however different features of the PI Infrastructure are used in each case.

- **Exercise 1** - Develop Predictive Model for a Single Transformer
    - Data preparation - PI Integrator for Business Analytics (BA).
    - Model testing - AF Analytics "Preview Results" – *No PI Tags required*
    - Model evaluation – Microsoft Power BI
- **Exercise 2** - Develop Predictive Model for Multiple Transformers
    - Data preparation - PI SQL Client
    - Model testing – AF Analytics Backfill – *PI Tags required*
    - Model evaluation - PI Vision Collections

- **Exercise 3** - Operationalize Forecast Model for Multiple Transformers
    - Operationalization – AF Analytics, PI Future Data and PI Vision

Additional data access methods and learning resources are described in the appendix.

- Appendix A - Example 2 using PI OLEDB Enterprise
- Appendix B - Example 2 using PI Web API
- Appendix C – Learning Resources

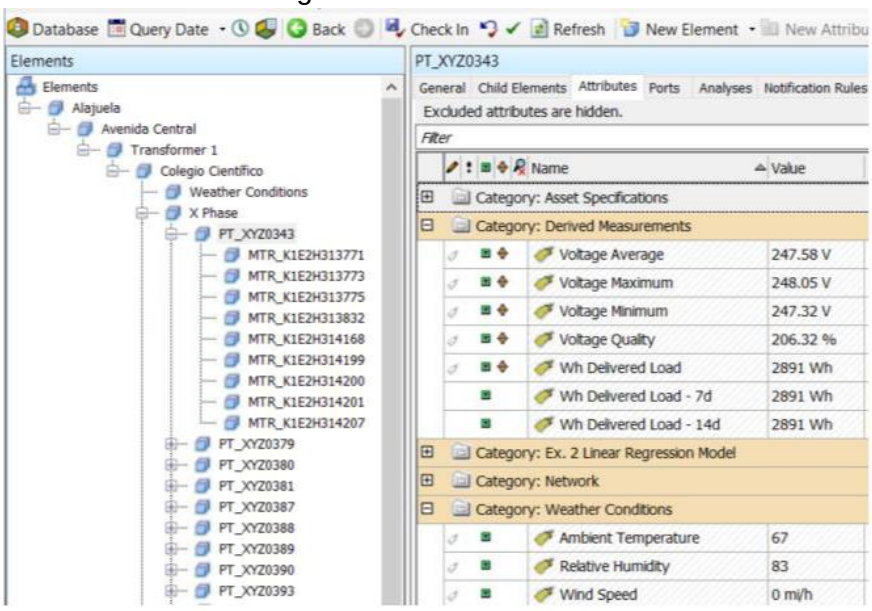
Supporting PI Infrastructure

The distribution network for the ***Avenada Central*** substation in the city of Alajuela, Costa Rica is represented in the PI AF hierarchy below. The AF model provides required context, data preparation, and operationalization support for statistical modelling projects. We will be focusing on the three-phase (X, Y, Z) ***Colegio Cientrifico*** circuit which has sixty pole transformers.

Our modelling effort requires data from the pole transformer (PT_*) level of the AF hierarchy, e.g. ***PY_XYZ20343***. Meter data (MTR_*) has been loaded into the PI Server for the three-month period, 6/1/17 to 8/31/17. An AF Rollup analytic was used to compute pole transformer power loads by summing the attribute ***Wh Delivered Load***, for the meters connected to it.

Additional AF attributes have been configured as relevant modelling features. These provide historical values for the Wh Delivered Load at exactly 7 and 14 days prior to the current time (***Wh Delivered Load – 7d*** and ***Wh Delivered Load – 14d***, respectively).

Weather conditions in Alajuela are also available in the AF Model and will be used in our analysis. For this lab, we will be assuming today is 8/31/17. The PI System contains historical weather data for ambient temperature, relative humidity and wind speed from 6/1/17 to 8/31/17, as well as, hourly weather forecast for the upcoming month of September 2017. These values have been stored in PI Future data tags.



Software Toolkit

- PI Server 2018 (with the optional RTQP Engine installed to support PI SQL Client)
- PI Integrator for Business Analytics
- PI SQL Client 2018
- PI OLEDB Enterprise 2017 R2
- PI Web API
- PI Vision
- Python and required package within Jupyter Notebook
- Microsoft Power BI

Exercise 1 - Develop Predictive Model for a Single Transformer

Learning Objectives

- Export transformer load and weather data into a text file using the PI Integrator for Business Analytics
- Examine the transformer dataset to gain a better understanding of loading characteristics
- Develop load prediction model for a single transformer using Python, in Jupyter Notebooks
- Test the model using AF Analytics "Results Preview" – *No PI tags required*.
- Assess model results in MS Power BI

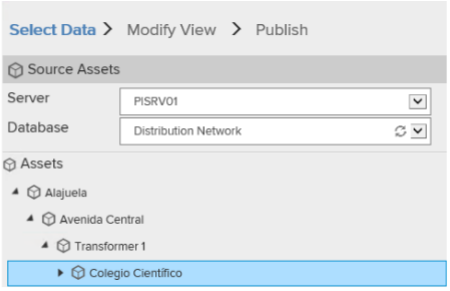
Create an Asset View – PI Integrator for BA

From the Windows taskbar, open AF Explorer and review the "Distribution Network" database. We will be using the PI Integrator for BA to create an asset view publication based on this asset model. The publication target will be a text file posted in the subdirectory, **C:\Users\student01\Python Lab** (this is the "Exploring AF for Advanced Analytics" folder shortcut on the desktop). This text file will provide data for creating the predictive model in Python.

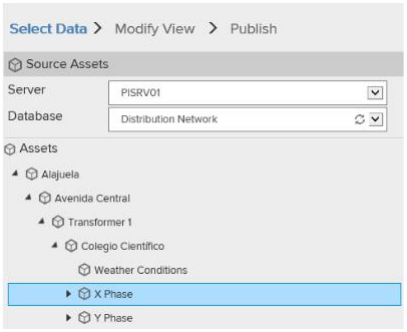
a. Open the PI Integrator for Business Analytics from the desktop shortcut.

b. Click **Create Asset View** on the top menu, name your view ***My Pole Transformer Loads*** and click **Create View**.

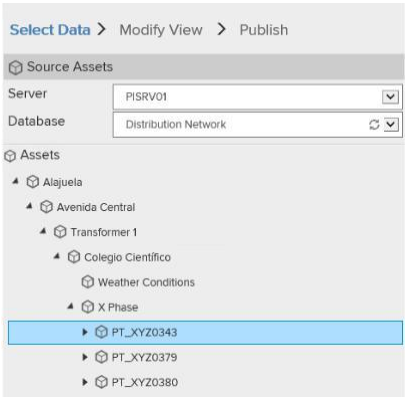
c. On the Select Data page, select the **Distribution Network** database. Expand the AF Hierarchy to the circuit level and select element **Colegio Científico**. Drag and drop this element into the Search Shape area in the middle of the screen under Asset Shape.



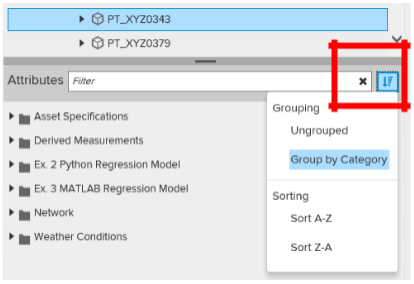
d. Expand the hierarchy to expose the phase level and select element **X Phase**. Drag and drop this element into the Search Shape area under the **Colegio Científico** element.



e. Expand the hierarchy to expose the transformer level, select element **PT_XYZ0343.** Drag and drop the element into the Search Shape area under the **X Phase** element.

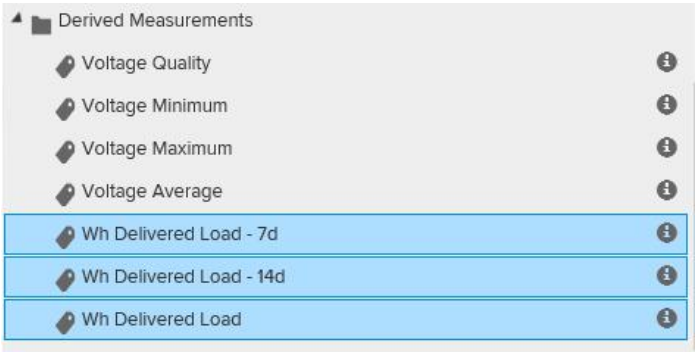


f. Use the sort button next to the Attributes filter to group the transformer attributes list by category.

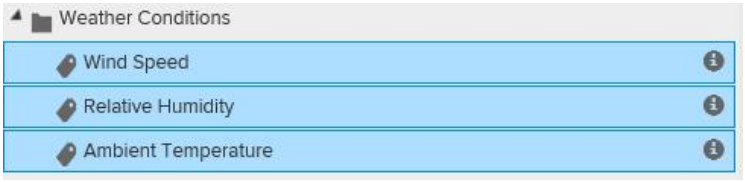


g. Click the **Asset Specifications** category and select the attributes shown below. Drag and drop the group into the Search Shape area under the element **PT_XYZ0343**.

h. Click the **Derived Measurements** category and select the attributes shown below. Drag and drop the group into the Search Shape area under the element **PT_XYZ0343**.



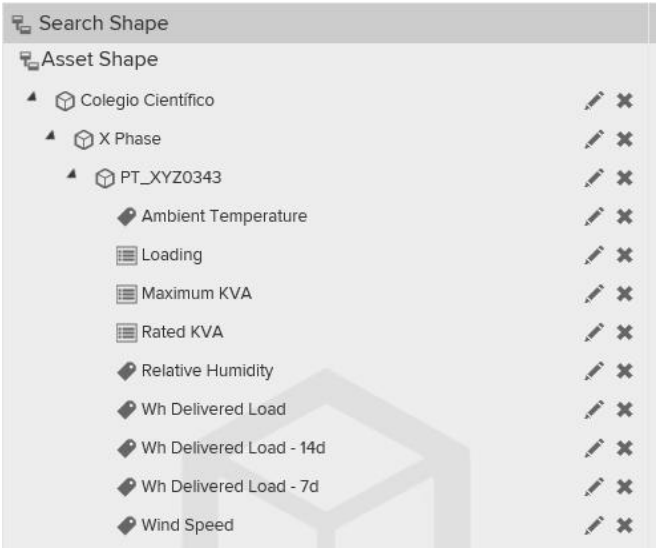
i. Click the **Weather Conditions** category and select the attributes shown below. Drag and drop the group into the Search Shape area under the element **PT_XYZ0343.**



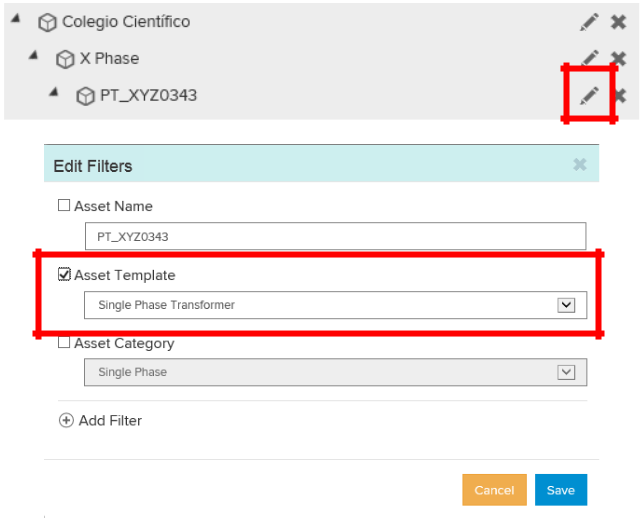
j. The steps above define an Asset Shape for a Single Phase Transformer element along with its selected attributes.

Tip

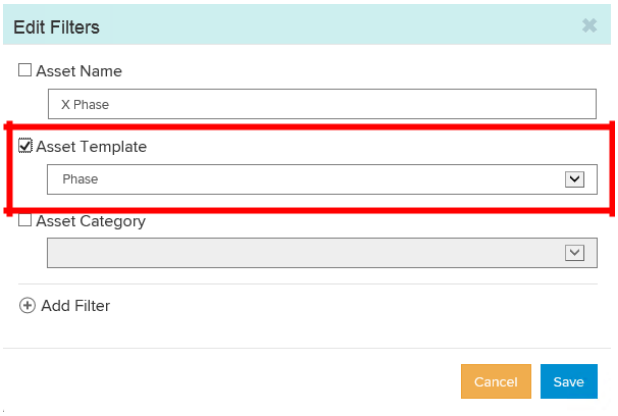
At this point, only one element is returned in the Match area because only one transformer element, PT_XYZ0343, has been specified. We need to leverage our AF Model to obtain data for all the assets we want to include in our analysis.



k. Click the edit pencil next to element **PT_XYZ0343** and modify the shape filter. Uncheck the **Asset Name** and check **Asset Template**. Select the **Single Phase Transformer** template to include all transformers under the X Phase of the circuit. The Matches area should show a list of 16 elements.



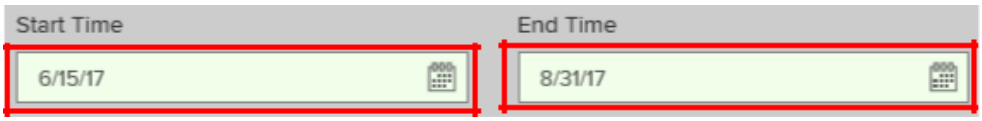
l. Click the edit pencil next to the element **X Phase** and modify the shape filter to include all phases derived from the **Phase** template. (Remember to uncheck the **Asset Name**.)



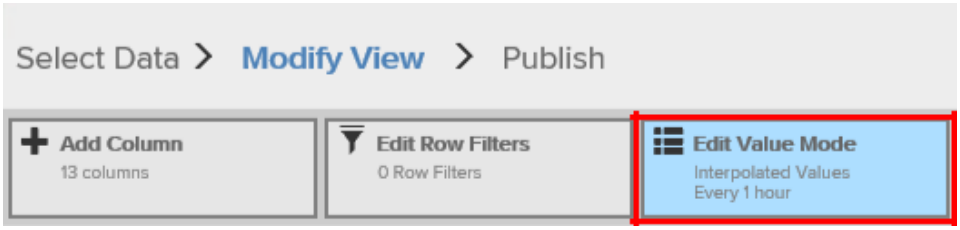
m. With the shape selected and filters set, you should see all sixty transformers listed int he Matches area.

n. Click the **Next** button in the upper right corner to proceed to the Modify View page.

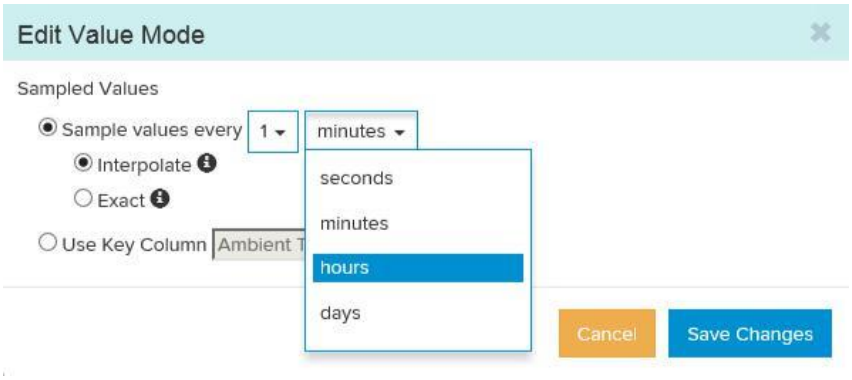
o. The Modify View page shows a preview of the first 100 rows from the first 10 matches of the asset shape. Set the time range, **6/15/17** to **8/31/17**, to match the available PI data for the lab. (Since the Wh Delivered -14d attribute returns values from two weeks ago, we must start the request on 6/15/17 instead of 6/1/17.) Click **Apply**.



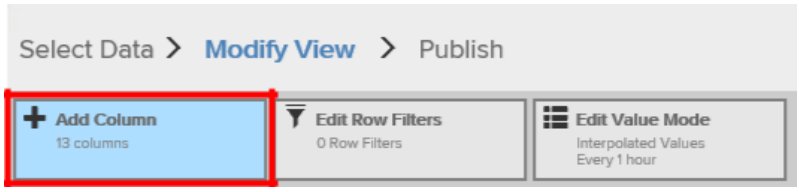
p. Click **Edit Value Mode** at the left side of the top menu.



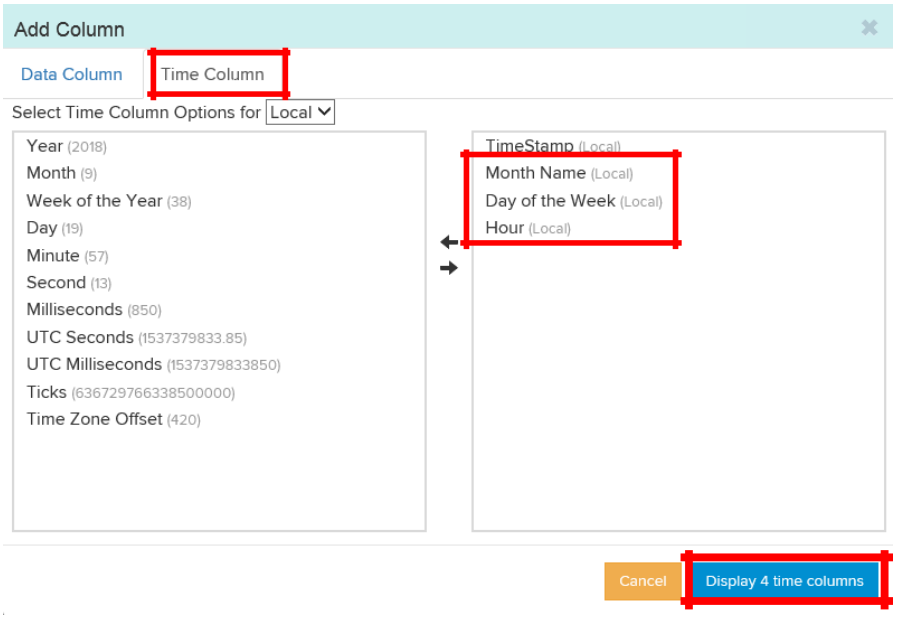
Leave the Sampled Values setting defaulted to return interpolated values. However, use the drop-down menu to select hours instead of minutes samples for each row. Click **Save Changes**.



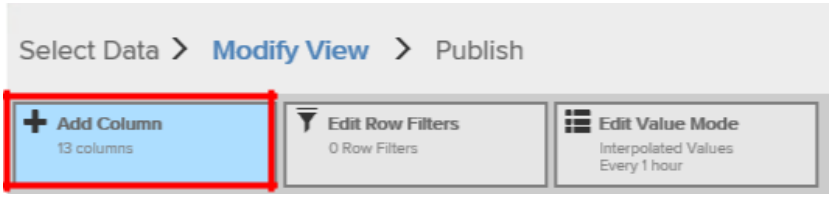
q. Click **Add Column** to add several time columns to the asset view.



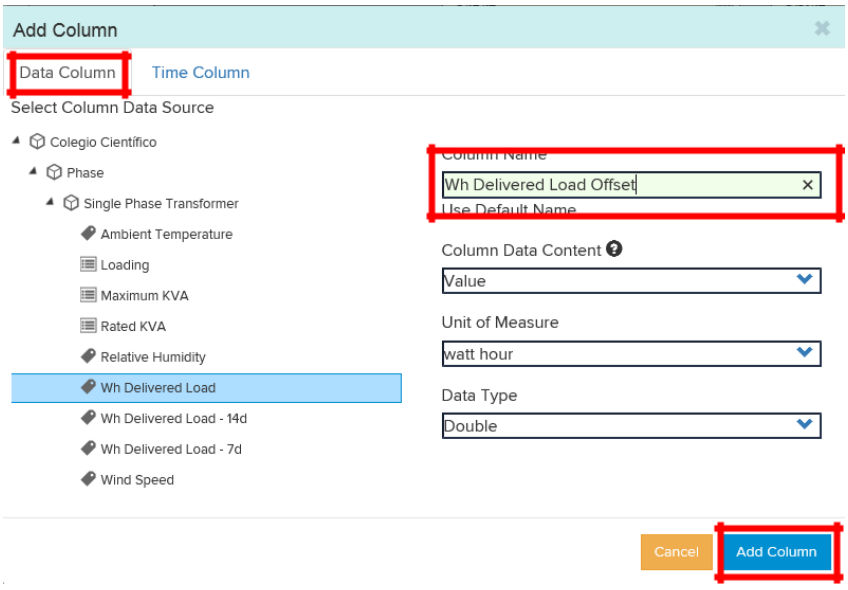
Select the **Time Column** tab and add the columns shown below (the TimeStamp column is always included by default). Click **Display 4 time columns**.



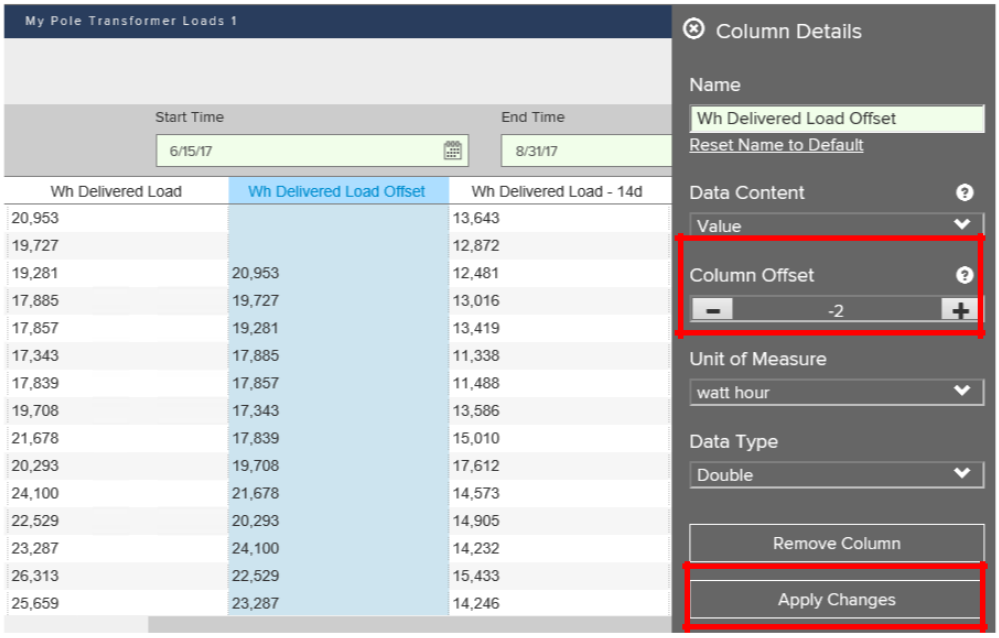
r. Click **Add Column** again, this time to illustrate a point.



Select the **Data Column** tab. Select the **Wh Delivered Load** attribute and change the Column Name to **Wh Delivered Load Offset**. Click **Add Column**.



s. Select the newly added **Wh Delivered Load Offset** column in the table, set its **Column Offset** to be minus 2 hours and click **Apply Changes**.

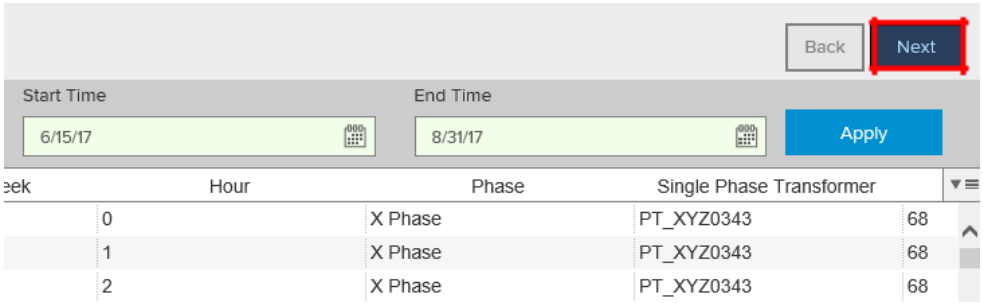


Tip

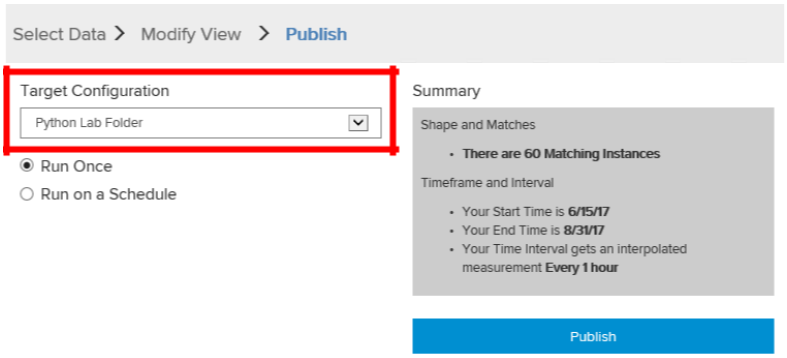
The PI Integrator for BA will return null values for requested values falling outside the shape's time range. Fortunately, we have configured AF attributes to return Wh Delivered Load values for 7 and 14 days ago.

t. Select the **Wh Delivered Load Offset** column in the table again. Click **Remove Column**, in its Column Details, we're not going to need this.

u. Click **Next** to proceed to the Publish page.



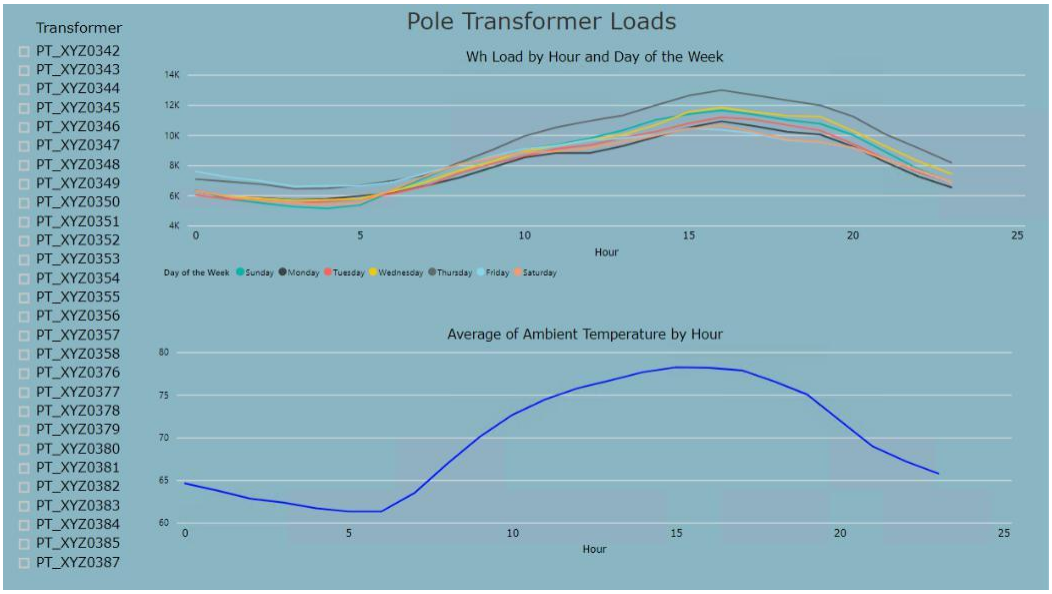
v. Select **Python Lab Folder** as the Target Configuration and click **Publish**, then **Confirm** to publish the data as a text file.



The file name will be the name of the asset view, My Pole Transformer Loads, and it will be in the subdirectory, C:\Users\student01\Python Lab (this is the "Exploring AF for Advanced Analytics" folder shortcut on the desktop). This target was configured using the Administration pages of the PI Integrator for BA.

Visual Assessment of Transformer Data

From the desktop, open the Power BI report called "Example 1 Data Analysis". The Pole Transformer Loads text file has been imported into this report to allow visual inspection of all sixty transformer load profiles. The profiles are presented for the 24 hours of the day and broken down by the days of the week. Look at transformers, 342, 358, and 416, to observe some of the differences than can exist. Obviously, we will eventually need to build separate models for each transformer.



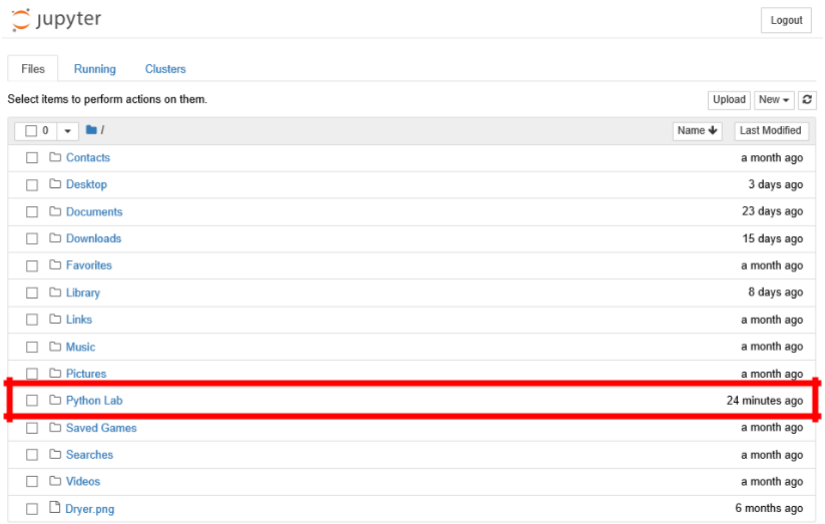
Python Development Environment

There are many open-source packages that support Python programming. In this lab, we'll use the analysis environment contained in the Anaconda distribution of Python. Also included in this distribution is Jupyter Notebooks. We will be using Jupyter Notebooks as our Integrated Development Environment (IDE) for the Python work. It is a browser-based environment which makes it easy to manage program development. This lab contains the following key components:

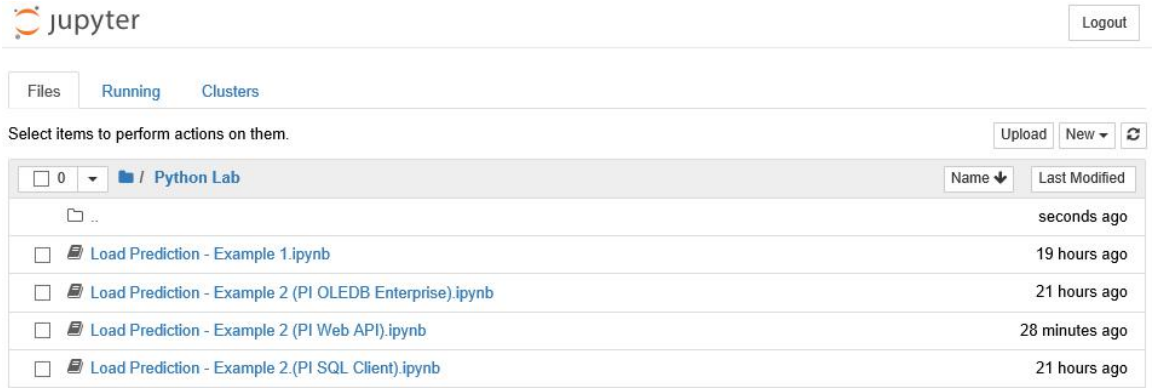
- Anaconda 5.1 distribution of Python 3.6
  - The Anaconda distribution of Python contains hundreds of commonly used packages to support python application development and data science
  - Jupyter: An interactive notebook useful for annotating python code with graphics, documentation, or tutorial instructions
- Python packages utilized in this lab - included in Anaconda 5.1
  - Numpy: Highly optimized numerical library useful for linear algebra
  - Pandas: Data management library for DataFrames, similar to R DataFrames
  - Matplotlib: Visualization library that feels similar to Matlab based plotting
  - Seaborn: Visualization library with a focus on statistics
  - Scikit-Learn: Machine Learning library
  - json, requests, urllib.parse, requests_kerberos: Support access to PI Web API
- Python packages utilized in this lab – ***not*** included in anaconda 5.1
  - adodbapi: toolkit for interacting with MS SQL and PI via the PI SQL Client and PI OLEDB

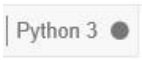
Jupyter Notebooks

Launch Jupyter Notebook by clicking once on the  icon in the Windows taskbar. You will see a command window appear - keep this open (minimized) throughout the lab. Eventually, Internet Explorer will open exposing a list of files in Jupyter's root directory. Click the link to the **Python Lab** folder.



The Python Lab folder contains the Python scripts files used the lab examples. We will be returning to it several times. Keeping the browser open will save time in accessing them during the session.



For this lab, you will simply need to know how to step through the Python scripts in a cell by cell fashion. In Jupyter, this is accomplished by hitting **Shift+Enter** while focused on a cell. This will execute the current cell and advance to the next cell. As you execute each cell in the notebook, wait for the Python's busy light to switch from busy, 

to ready 

. This indication is in the upper right-hand corner of the Jupyter Notebook. If this is your first time using Jupyter Notebooks or simply want to learn more, the Jupyter learning resources listed in Appendix C may be helpful.

Develop the Model in Python

The following code creates a multivariable linear regression model for one transformer. We will import the transformer data from the text file published by the PI Integrator for B A. Through a simple cut and paste operation, we will place the model's equation into AF Analytics and leverage the AF Analysis Service to test the model. Test results will be exported from AF Analytics and evaluated using Power BI.

Tip	**To run the code with your PI Asset View, you'll want to replace "Pole Transformer Loads" with "My Pole Transformer Loads" in the first cell of this Jupyter Notebook**

a. From Jupyter, open the Python script, **Load Prediction - Example 1**.

b. Edit the **file_path** assignment statement in the first cell to be '**My** Pole Transformer Loads.txt'

c. Import transformer data from the text file, “**My** Pole Transformer Loads.txt” published by the PI Integrator for Business Analytics. Select the first cell and type **Shift+Enter**. It may take a few seconds for the file to be loaded.

```
# EXAMPLE 1 - Create linear regression models to predict pole transformer loading.
# Read PI data from text file published by PI Integrator for Busines Analytics.

import pandas as pd             # "pandas" for managing dataframe from published .txt file.
import numpy as np              # "numpy" for statistics.
import matplotlib.pyplot as plt # "matplot.pyplot" for graphics
import seaborn as sns           # "seaborn" for statistical data visualization.

# Define path relative to curent directory.
file_path = 'My Pole Transformer Loads.txt'

# Read "Pole Transformer Load.txt" into the transformers dataframe.
poleTransformerLoads = pd.read_csv(file_path,delimiter="\t")

# Print header rov of dataframe.
print(poleTransformerLoads[0:0])
```

```
Empty DataFrame
Columns: [Id, Colegio Cientifico, TimeStamp, Month Name, Day of the Week, Hour, Phase, Single Phase Transformer, Ambi
ent Temperature, Loading, Maximum KVA, Rated KVA, Relative Humidity, Wh Delivered Load, Wh Delivered Load - 14d, Wh D
elivered Load - 7d, Wind Speed, PIIntTSTicks, PIIntShapeID]
Index: []
```

The column headers returned should match the ones configured in the PI Asset View. The columns “Id”, “PIIntTSTicks” and “PIIntShapeID” have been added by the PI Integrator for BA and are for internal use.

d. Rename columns with shorter names. Create a smaller dataframe, **modellingData**, containing only the values need for predicting transformer loads. Select the next cell and type **Shift+Enter**.

```
# Rename some columns with shorter names to make them easier to work with.
poleTransformerLoads.rename(columns = {'Single Phase Transformer':'Transformer'}, inplace = True )
poleTransformerLoads.rename(columns = {'Ambient Temperature':'Temperature'}, inplace = True )
poleTransformerLoads.rename(columns = {'Relative Humidity':'Humidity'}, inplace = True )
poleTransformerLoads.rename(columns = {'Wh Delivered Load':'Wh Load'}, inplace = True )
poleTransformerLoads.rename(columns = {'Wh Delivered Load - 14d':'Wh Load-14d'}, inplace = True )
poleTransformerLoads.rename(columns = {'Wh Delivered Load - 7d':'Wh Load-7d'}, inplace = True )
poleTransformerLoads.rename(columns = {'Wind Speed':'Wind'}, inplace = True )

# Define second dataframe with just data needed for our modelling.
modellingData = poleTransformerLoads[['Transformer', 'TimeStamp', 'Hour', 'Temperature', 'Humidity',
                    'Wind','Wh Load', 'Wh Load-7d', 'Wh Load-14d']]

# Peek at the first five rows to make sure things look right.
modellingData.head()
```

	Transformer	TimeStamp	Hour	Temperature	Humidity	Wind	Wh Load	Wh Load-7d	Wh Load-14d
0	PT_XYZ0343	6/15/2017 12:00:00 AM	0	68	93.0	7.0	6529.0	3278.0	2464.0
1	PT_XYZ0343	6/15/2017 1:00:00 AM	1	68	93.0	6.0	6413.0	3122.0	2798.0
2	PT_XYZ0343	6/15/2017 2:00:00 AM	2	68	93.0	6.0	6781.0	2658.0	2490.0
3	PT_XYZ0343	6/15/2017 3:00:00 AM	3	68	90.0	0.0	5348.0	2873.0	2722.0
4	PT_XYZ0343	6/15/2017 4:00:00 AM	4	67	93.0	0.0	6916.0	3306.0	2698.0

The new dataframe, **modellingData**, contains only data needed for building the predictive model.

e. Examine the last five rows of the dataframe using the **tails()** method. Select this cell and hit **Shift+Enter**.

```
# Peek at the last five rows to make sure we got them all.
modellingData.tail()
```

	Transformer	TimeStamp	Hour	Temperature	Humidity	Wind	Wh Load	Wh Load-7d	Wh Load-14d
110935	PT_XYZ0414	8/30/2017 8:00:00 PM	20	68	73.0	7.0	69.0	68.0	44.0
110936	PT_XYZ0414	8/30/2017 9:00:00 PM	21	64	81.0	3.0	68.0	68.0	45.0
110937	PT_XYZ0414	8/30/2017 10:00:00 PM	22	66	75.0	3.0	68.0	69.0	45.0
110938	PT_XYZ0414	8/30/2017 11:00:00 PM	23	65	78.0	3.0	67.0	68.0	46.0
110939	PT_XYZ0414	8/31/2017 12:00:00 AM	0	62	86.0	3.0	68.0	69.0	44.0

f. Gain a simple statistical perspective of the values in our dataframe. Select this cell and hit **Shift+Enter**.

```
# Gain an easy statistical perspective of the values in our dataframe.
modellingData.describe().T
```

	count	mean	std	min	25%	50%	75%	max
Hour	110940.0	11.493780	6.925509	0.0	5.0	11.0	17.00	23.0
Temperature	110940.0	69.853434	8.511516	43.0	64.0	70.0	76.00	89.0
Humidity	110940.0	69.361259	18.196455	29.0	54.0	71.0	87.00	100.0
Wind	110940.0	6.581908	4.486848	0.0	3.0	6.0	9.00	24.0
Wh Load	110940.0	8475.092098	7452.368773	0.0	1970.0	7372.0	12713.00	57814.0
Wh Load-7d	110940.0	8859.001180	7763.364070	0.0	2063.0	7732.0	13279.25	57814.0
Wh Load-14d	110940.0	8667.456263	7665.040191	0.0	1973.0	7519.0	12951.00	57814.0

g. Generate a correlation matrix of the variables. This will provide statistics on all sixty transformers as a group. Select this cell and hit **Shift+Enter**.

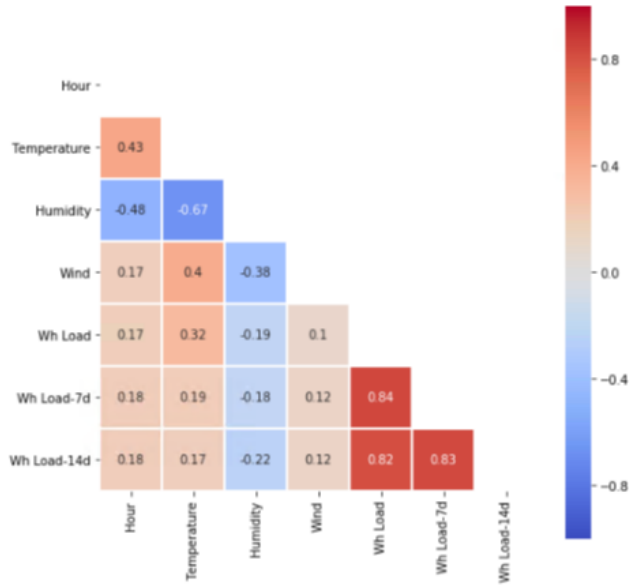
```
# Generate a correlation matrix to check for the exstance of good relatinoships for our model.
model_corr = modellingData.corr(method ='pearson')
model_corr
```

	Hour	Temperature	Humidity	Wind	Wh Load	Wh Load-7d	Wh Load-14d
Hour	1.000000	0.431996	-0.478854	0.165933	0.172268	0.177432	0.178858
Temperature	0.431996	1.000000	-0.667318	0.403062	0.318962	0.185169	0.169464
Humidity	-0.478854	-0.667318	1.000000	-0.382336	-0.194920	-0.184463	-0.220623
Wind	0.165933	0.403062	-0.382336	1.000000	0.099989	0.120011	0.121844
Wh Load	0.172268	0.318962	-0.194920	0.099989	1.000000	0.843289	0.820274
Wh Load-7d	0.177432	0.185169	-0.184463	0.120011	0.843289	1.000000	0.833011
Wh Load-14d	0.178858	0.169464	-0.220623	0.121844	0.820274	0.833011	1.000000

h. Generate a correlation chart of the variables. This will provide statistics on all sixty transformers as a group. Select this cell and hit **Shift+Enter**.

```
# Set size of chart.
fig, ax = plt.subplots(figsize=(8,8))

# Lets supress some of this info by masking it
mask = np.zeros_like(model_corr, dtype=np.bool)
mask[np.triu_indices_from(mask)] = True

# Lets take a graphical view of these correlations.
sns.heatmap(data=model_corr, mask=mask, cmap='coolwarm', vmin=-1, vmax=1, linewidths=1, square=True,
    annot=True, ax=ax)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0xe016d49978>
```



i. Change the index of **modellingData** from the “id” column to the “Transformer” column. Select this cell and hit **Shift+Enter**.

```
# In order to analyze transformers individually, we need to set the datframe's index to the "Transformer" column.
modellingData = modellingData.set_index("Transformer", drop=False)

# Take a look, see the difference?
modellingData.head()
```

	Transformer	TimeStamp	Hour	Temperature	Humidity	Wind	Wh Load	Wh Load-7d	Wh Load-14d
Transformer									
PT_XYZ0343	PT_XYZ0343	6/15/2017 12:00:00 AM	0	68	93.0	7.0	6529.0	3278.0	2464.0
PT_XYZ0343	PT_XYZ0343	6/15/2017 1:00:00 AM	1	68	93.0	6.0	6413.0	3122.0	2798.0
PT_XYZ0343	PT_XYZ0343	6/15/2017 2:00:00 AM	2	68	93.0	6.0	6781.0	2658.0	2490.0
PT_XYZ0343	PT_XYZ0343	6/15/2017 3:00:00 AM	3	68	90.0	0.0	5348.0	2873.0	2722.0
PT_XYZ0343	PT_XYZ0343	6/15/2017 4:00:00 AM	4	67	93.0	0.0	6916.0	3306.0	2698.0

With rows indexed by transformer name, we will be able to build a regression models for one transformer.

j. Based on the Power BI report, we know each transformer's load profile is unique. Let's focus on just one, PT_XYZ0358. Select this cell and hit **Shift+Enter**.

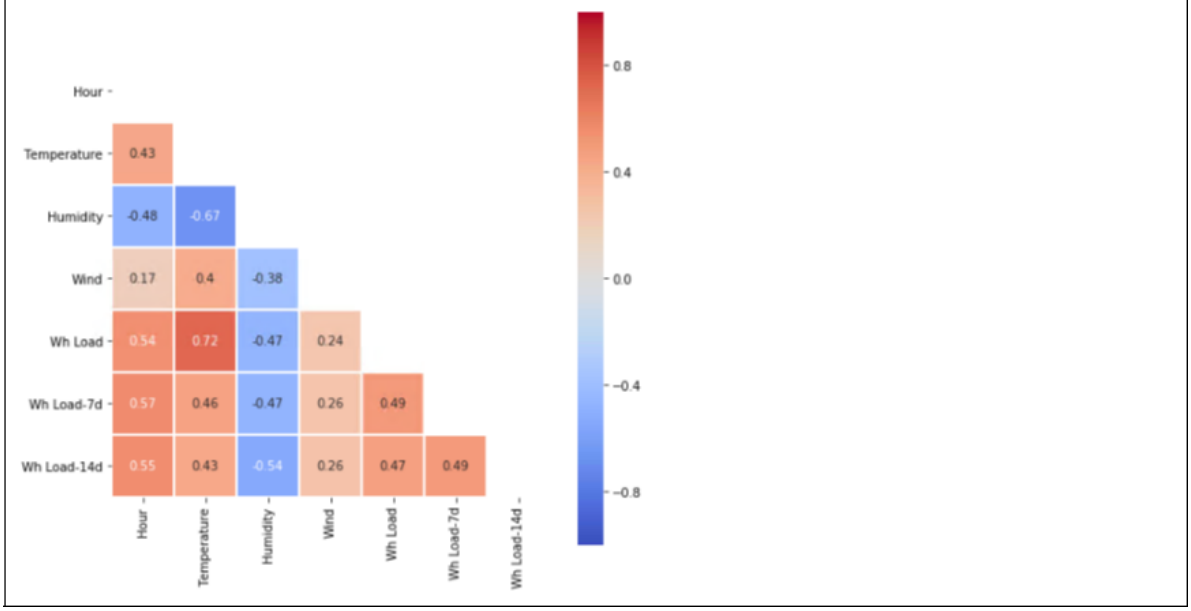
```
# Here's how we can focus on one transformer, say PT_XYZ0358.
modellingData.loc["PT_XYZ0358",:].head()
```

Transformer	Transformer	TimeStamp	Hour	Temperature	Humidity	Wind	Wh Load	Wh Load-7d	Wh Load-14d
PT_XYZ0358	PT_XYZ0358	6/15/2017 12:00:00 AM	0	68	93.0	7.0	8439.0	2519.0	3071.0
PT_XYZ0358	PT_XYZ0358	6/15/2017 1:00:00 AM	1	68	93.0	6.0	7596.0	2454.0	2468.0
PT_XYZ0358	PT_XYZ0358	6/15/2017 2:00:00 AM	2	68	93.0	6.0	7497.0	2253.0	2285.0
PT_XYZ0358	PT_XYZ0358	6/15/2017 3:00:00 AM	3	68	90.0	0.0	6089.0	2238.0	2193.0
PT_XYZ0358	PT_XYZ0358	6/15/2017 4:00:00 AM	4	67	93.0	0.0	6128.0	2530.0	2497.0

k. Generate a correlation chart for PT_XYZ_0358. Select this cell and hit **Shift+Enter**.

```
# Redo the correlatioin matrix for just PT_XYZ0358.
model_corr = modellingData.loc["PT_XYZ0358",:].corr(method ='pearson')

fig, ax = plt.subplots(figsize=(8,8))

# Lets supress some of this info by masking it
mask = np.zeros_like(model_corr, dtype=np.bool)
mask[np.triu_indices_from(mask)] = True

# Lets take a graphical view of these correlations.
sns.heatmap(data=model_corr, mask=mask, cmap='coolwarm', vmin=-1, vmax=1, linewidths=1, square=True,
    annot=True, ax=ax)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0xe017573828>
```



l. Generate linear regression model coefficients for PT_XYZ0358 and print the equation. Select this cell and hit **Shift+Enter**. Keep the browser open, we will be coming back to get this equation.

```
# Import the linear regression model from the scikit-learn package.
from sklearn.linear_model import LinearRegression

# Create linear regression object
LinReg = LinearRegression()

# Create dataset of just PT_XYZ0358 data.
transformer_0358 = modellingData.loc["PT_XYZ0358",:]

# Perform linear regression fit - four terms.
LinReg.fit(transformer_0358[["Wh Load-7d","Wh Load-14d","Temperature","Humidity"]],transformer_0358["Wh Load"])

# Print equation.
print("Eq:\n", LinReg.coef_[0], "*","'Wh Delivered Load - 7d' + ",
      LinReg.coef_[1], "*","'Wh Delivered Load - 14d' + ",
      LinReg.coef_[2], "*", "'Ambient Temperature' +",
      LinReg.coef_[3], "*","'Relative Humidity' +(",LinReg.intercept_, ")" )
```

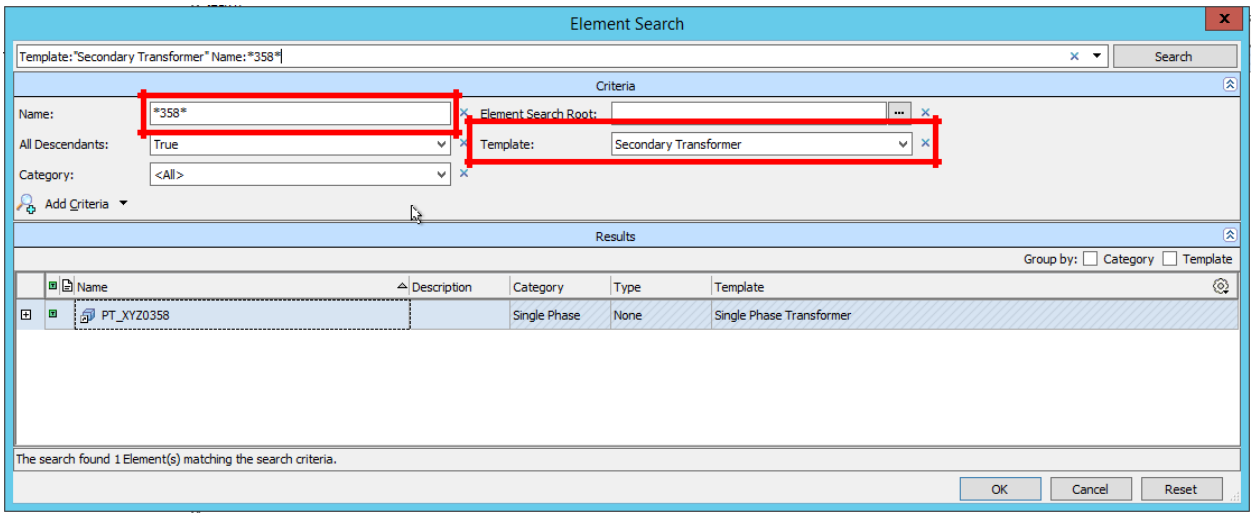
```
Eq:
 0.16826318479599217 * 'Wh Delivered Load - 7d' +  0.1781988453046811 * 'Wh Delivered Load - 14d' +  330.093697742419
75 * 'Ambient Temperature' + 37.6745301245552 * 'Relative Humidity' +( -19997.07169999319 )
```

Test Model - AF Analytics Results Preview

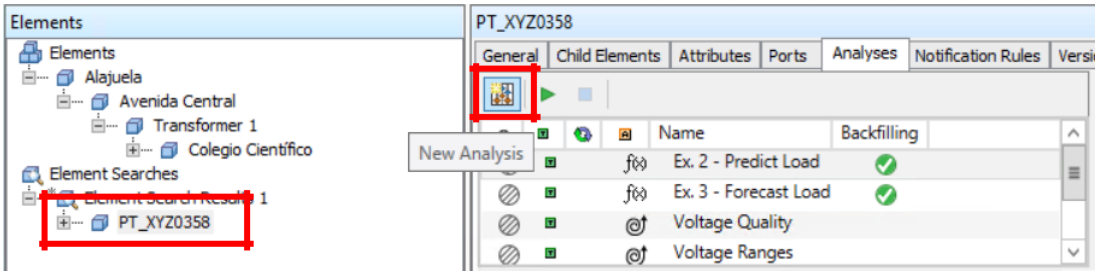
Our first Python script has produced a modelling equation for predicting the load on transformer, PT_XYZ0358. We can use AF Analytics to test the results without using PI tags.

a. Open PI System Explorer from the Windows taskbar,  .

b. In the upper left-hand corner select **Search, Element Search** to open the Element Search dialog. Fill-in the search criteria matching that shown below, **Name: *358* Template: Secondary Transformer**. Click **Ok**.



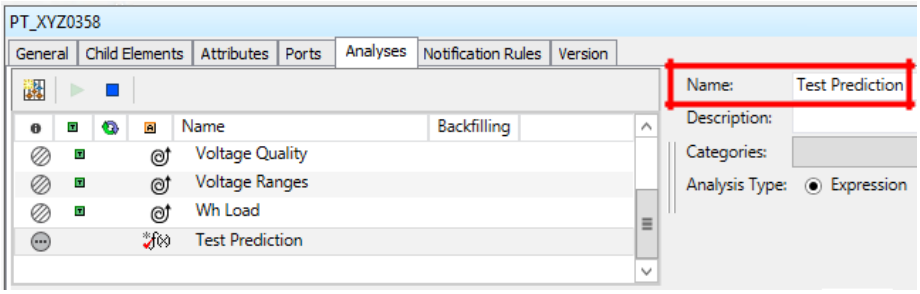
c. Under **Element Search Results** in the AF hierarchy, select element **PT_XYZ0358**. Select the Analysis tab and add a new analysis.



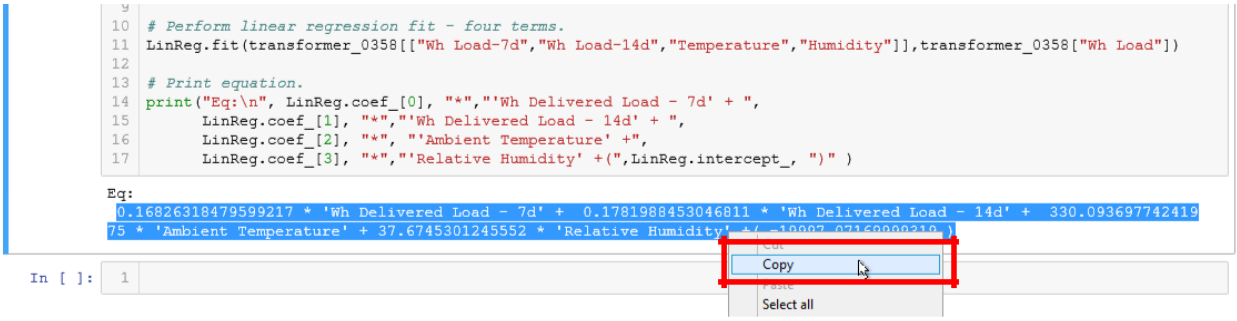
✓ Best Practice

Alternatively, we could have navigated the AF Hierarchy to get to this element but for larger AF structures searching is much easier.

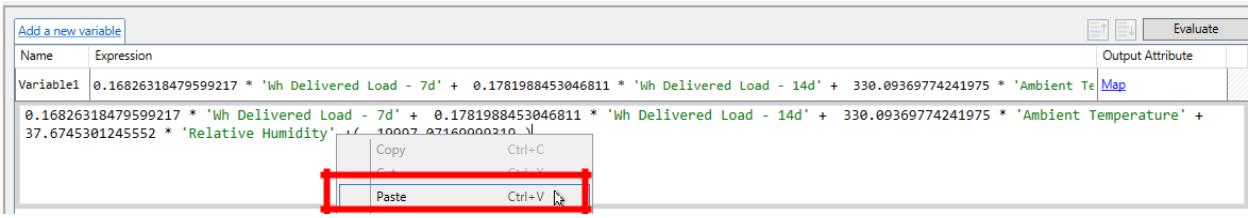
d. Select **"Analysis 1"** and change the name to "Test Prediction".



e. Return to the Python script in Jupyter Notebook and select the load prediction equation for transformer PT_XYZ0358 **Copy** it into the clipboard.



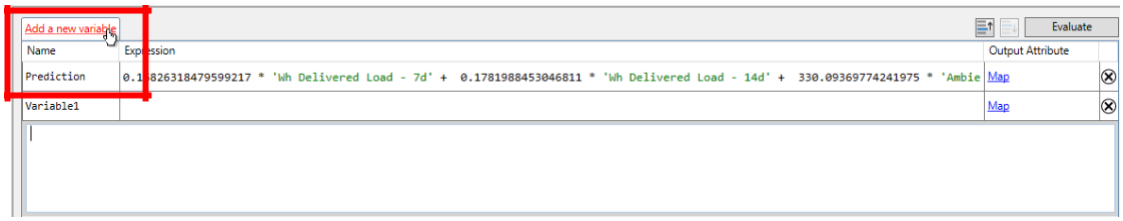
f. Go back to PI System Explorer and **Paste** the equation into the "Test Prediction" analysis.



Tip

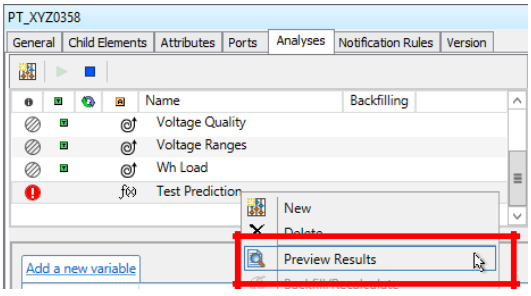
We have designed the Python print statement to align with the attribute names we have configured int AF, as indicated by green text.

g. Change the variable name from **Variable 1** to ***Prediction***. Click **Add a new variable** to add another one.

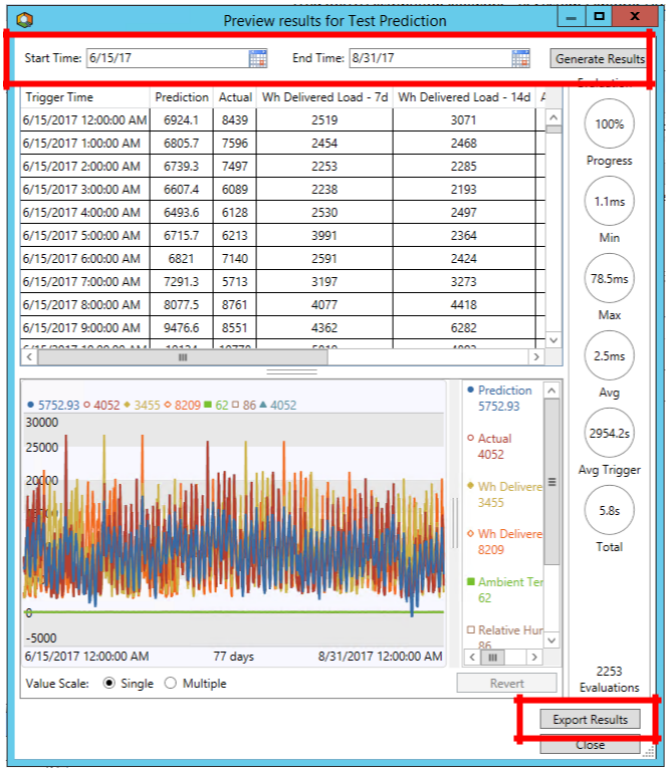


h. Change the new **Variable1** to ***Actual*** and type in the transformer load attribute **'Wh Delivered Load'.** Once you type the first single quote, AF Analytics should provide a selection list.

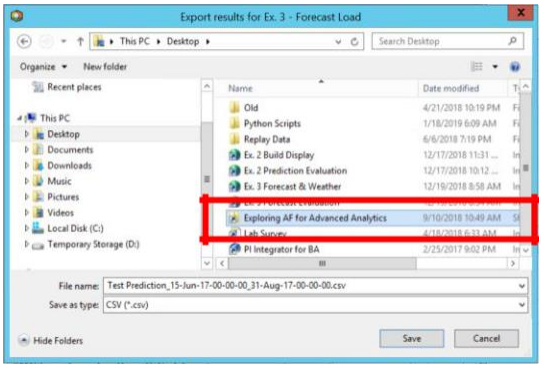
i. Right-click on the "Test Prediction" analysis and click **Preview Results**.



j. In the preview dialog, set the Start Time to **6/15/17** and the End Time to **8/31/17**. Click **Generate Results**.

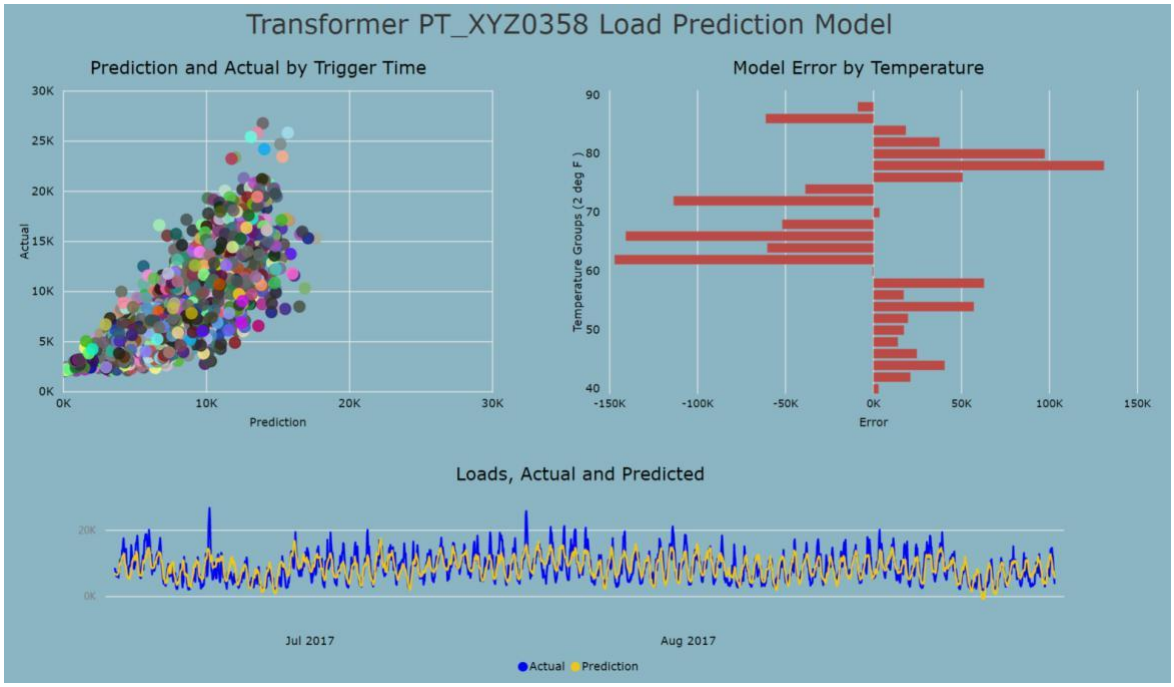


k. Once you see a completed trend of data, the PI Analysis Service has finished. Click **Export Results** and save the .csv file to the **Exploring AF for Advanced Analytics** desktop folder.



Evaluate Model – Power BI

Reopen the Power BI report, “Example 1 Data Analysis”. The “Test Prediction_15-Jun-17-00-00-00_31-Aug-17-00-00-00.csv” file has been imported into this report. Click the bottom tab **PT_XYZ0358 Model**, to see some example analysis of the model test. Looks like there may be room for improvement, but this will be good enough for this lab’s curriculum.



Exercise 2 - Develop Predictive Models for Multiple Transformers

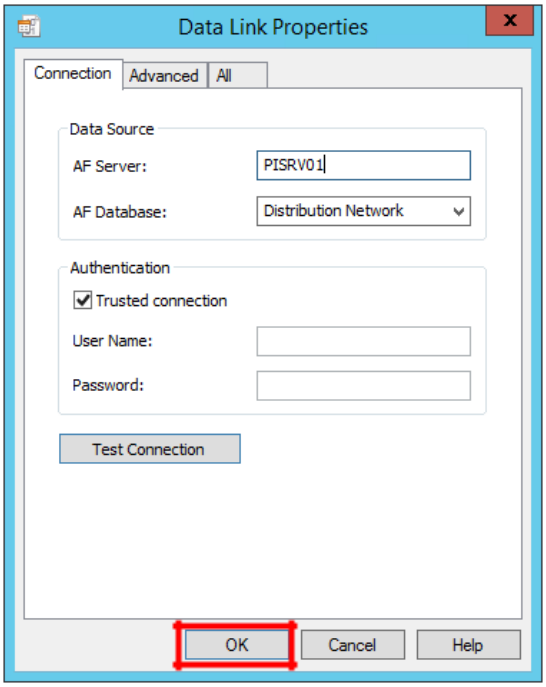
Learning Objectives

- Create a view using PI SQL Client to extract transformer load and weather data directly from the PI System.
- Develop load prediction model for multiple transformers using Python, in Jupyter Notebooks.
- Test and assess this model using AF analytics by backfilling the prediction into PI.

Create a PI View – PI SQL Client

a. Open PI SQL Commander Lite from the Windows taskbar,  .

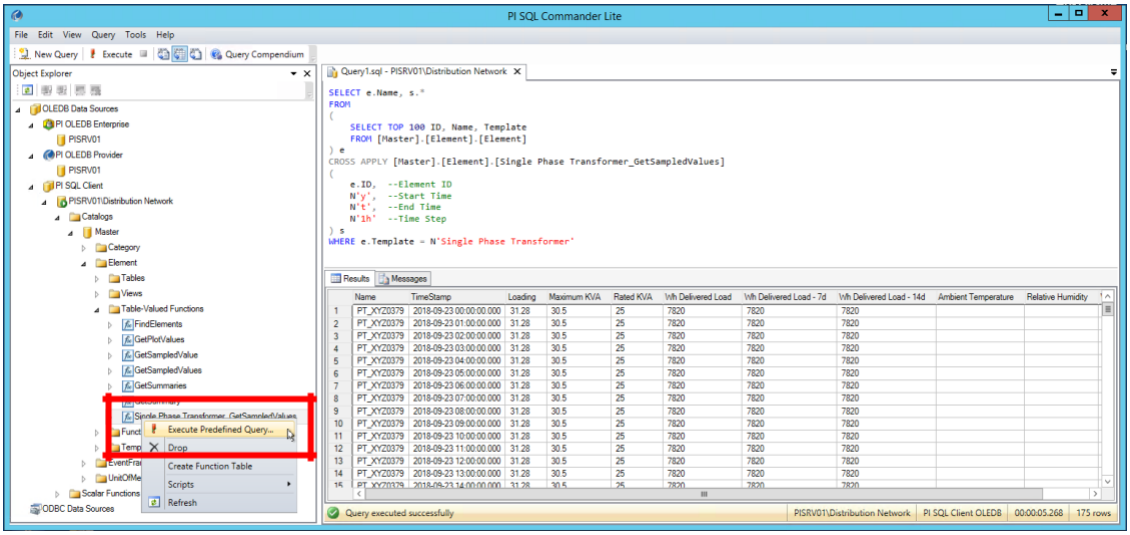
b. Under the **PI SQL Client** branch of the Object Explorer hierarchy, select the **PISRV01\Distribution Network** catalog and click **Connect.** Once the Data Link Properties dialog appears, click **Ok.**



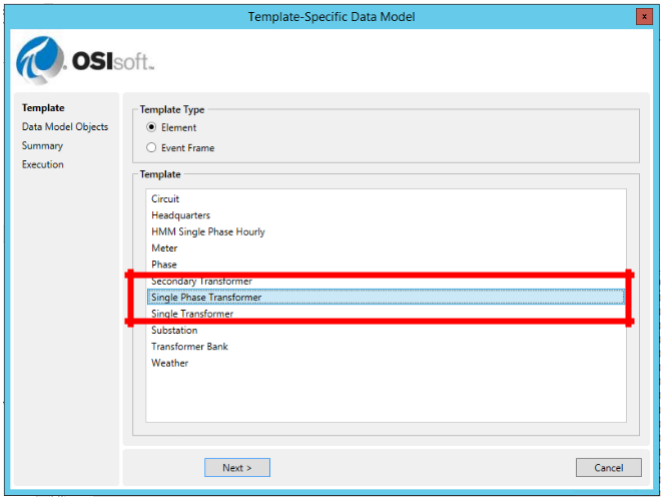
c. Expand the Object Explorer hierarchy through the **Master \ Element \ Table Valued-Functions** path.

Right-click on the **Single Phase Transformer_GetSampleValues** function and select **Execute Predefined Query…** The query's results will be returned to the grid.

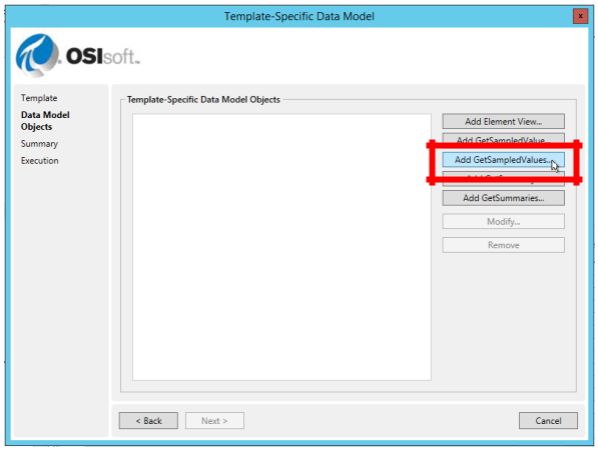
This function was created based on the Single Phase Transformer template defined in the Distribution Network AF model. Next, we will create our own view to access the data needed for our Python modelling.



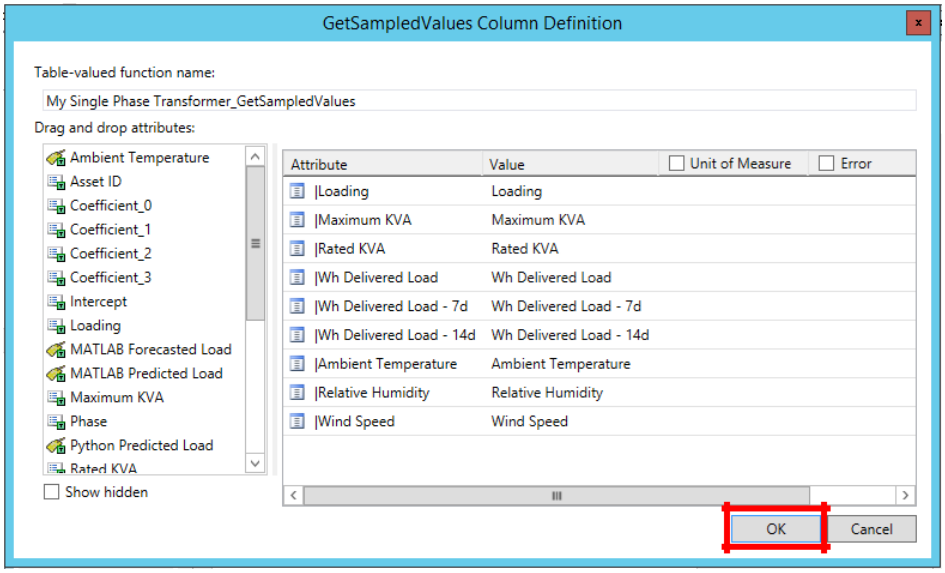
d. Make a new Table-Valued Function by right-clicking on the **Element** folder and selecting **Create Template-specific Data Model**… This will open the dialog shown below. Select the Single Phase Transformer template and click **Next**.



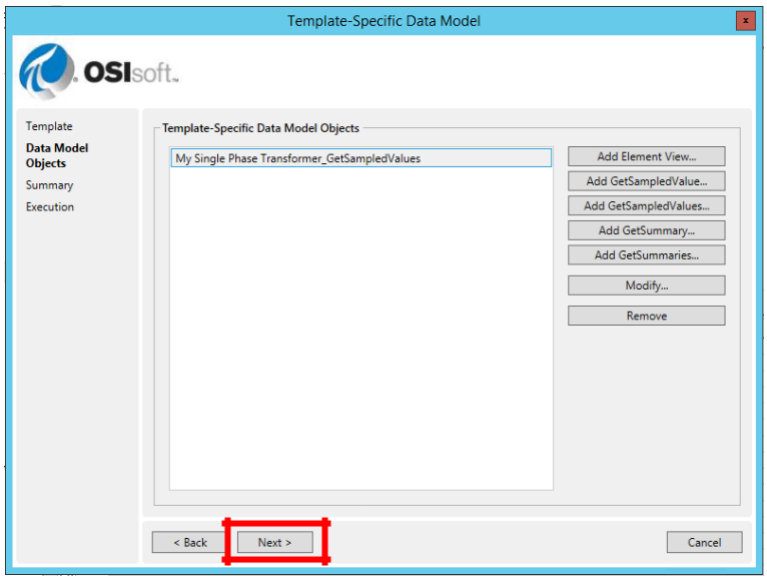
e. Next, click **Add GetSampledValues** in the Data Model dialog.



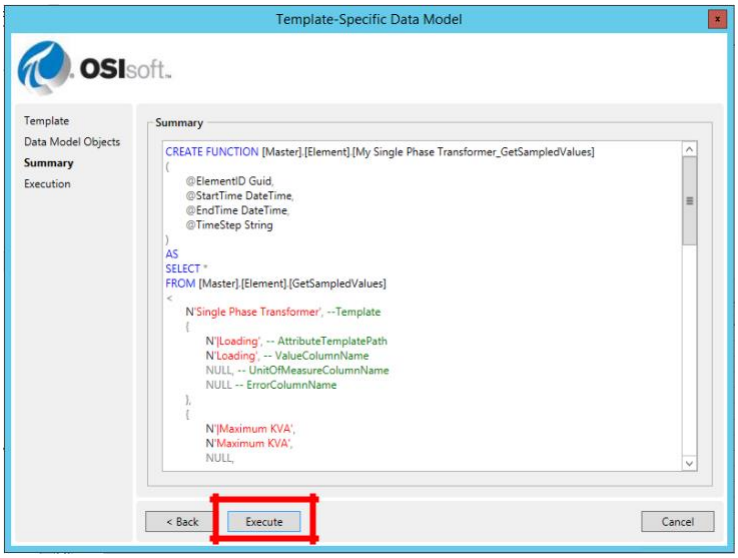
f. At the top of the Column Definition dialog, uncheck **Units of Measure** and **Error.** Change the Table-valued function name from the default, which has already been created, to **"My Single Phase Transformer_GetSampledValues".** Drag and drop the attributes shown below to specify the columns we need for analysis. Click **OK**.



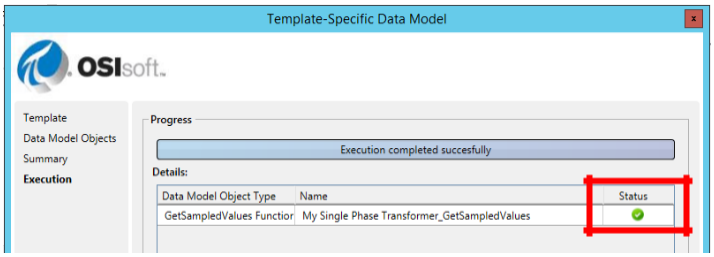
g. Returning to the Data Model dialog, with the new function configured, click **Next**.



h. The CREATE FUNCTION has been configured for you. Click **Execute** to create the function.



i. A green Status check means the function is configured. Click **Done**.



j. Return to the Table-Valued Functions folder in the Object Explorer and find your new function, **My Single Phase Transformer_GetSampledValues**. Right-click on it and select **Execute Predefined Query…** The following query should appear about the query viewer.

```
SELECT e.Name, s.*
FROM
(
    SELECT TOP 100 ID, Name, Template
    FROM [Master].[Element].[Element]
) e
CROSS APPLY [Master].[Element].[My Single Phase Transformer_GetSampledValues]
(
    e.ID,  --Element ID
    N'y',  --Start Time
    N't',  --End Time
    N'1h'  --Time Step
) s
WHERE e.Template = N'Single Phase Transformer'
```

k. The default function is limited to 100 rows and its time range is set to yesterday at one-hour intervals. We will modify this query and save it as a view, so we can get all the data needed for our statistical model. To start, copy this entire query into the clipboard.

Select the **Views** folder under Element. Right-click on **Views**, select **Scripts**, then **CREATE VIEW**, then click **New Query Editor Window**. The following query will appear. Replace <view definition> with the query saved in the clipboard.

```
CREATE VIEW [Master].[Element].[<view name>]
AS
<view definition>
```

l. The following query should now appear in the query window.

```
CREATE VIEW [Master].[Element].[<view name>]
AS
SELECT e.Name, s.*
FROM
(
    SELECT TOP 100 ID, Name, Template
    FROM [Master].[Element].[Element]
) e
CROSS APPLY [Master].[Element].[My Single Phase Transformer_GetSampledValues]
(
    e.ID,  --Element ID
    N'y',  --Start Time
    N't',  --End Time
    N'1h'  --Time Step
) s
WHERE e.Template = N'Single Phase Transformer'
```

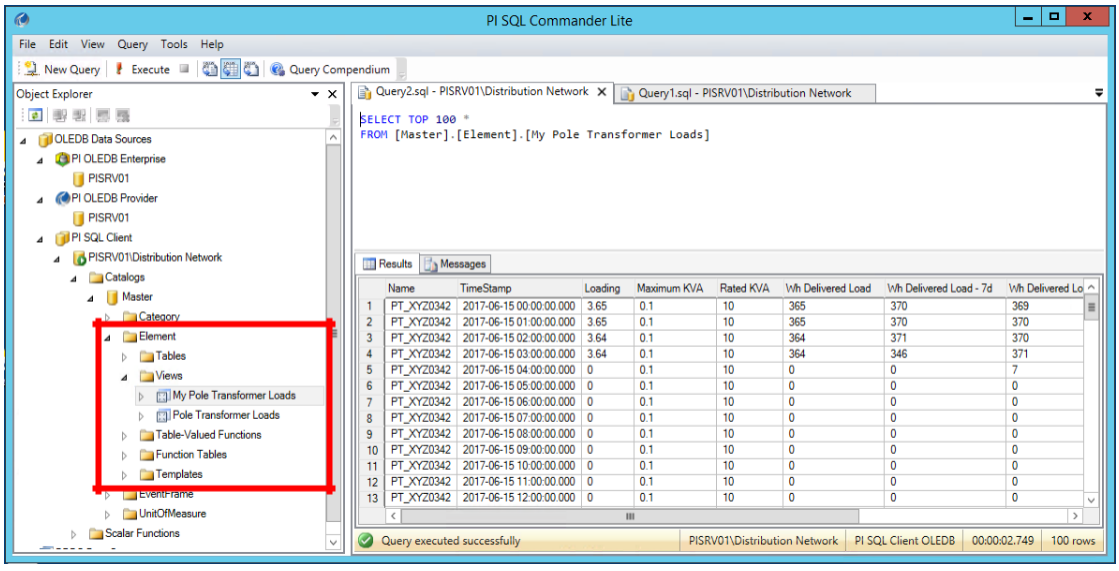
m. Modify this query by;
    a. Replacing, within the brackets, “<view name>” with “My Pole Transformer Loads”.
    b. Deleting “TOP 100” from the second SELECT statement.
    c. Change the Start Time from ‘y’ to ‘15-jun-2017'.
    d. Change the End Time from ‘t’ to '31-aug-2017'.

n. When finished with the edits, the query should look like the one below.

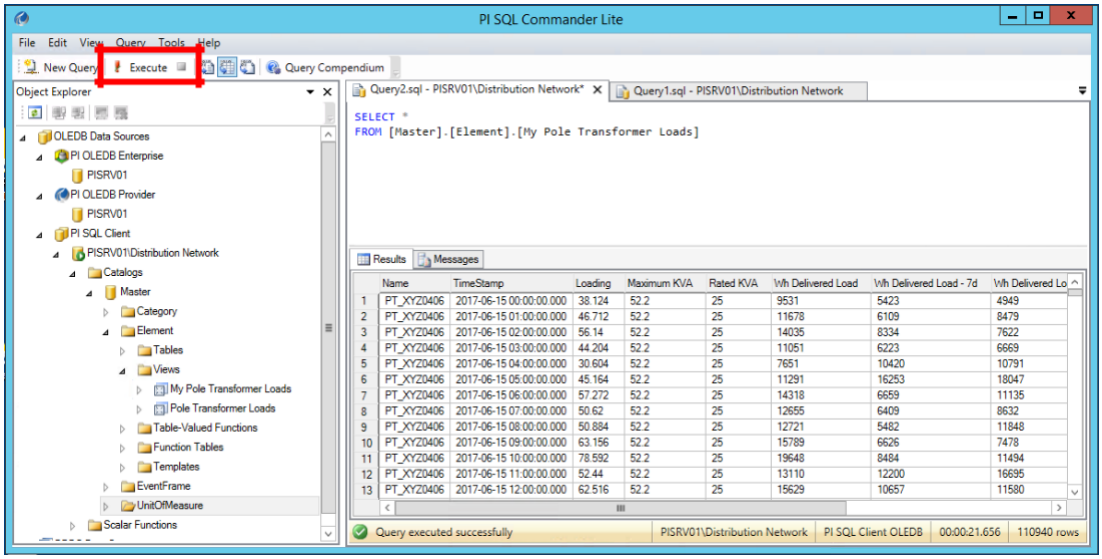
```
CREATE VIEW [Master].[Element].[My Pole Transformer Loads]
AS
SELECT e.Name, s.*
FROM
(
    SELECT ID, Name, Template
    FROM [Master].[Element].[Element]
) e
CROSS APPLY [Master].[Element].[My Single Phase Transformer_GetSampledValues]
(
    e.ID,  --Element ID
    N'15-jun-2017',  --Start Time
    N'31-aug-2017',  --End Time
    N'1h'  --Time Step
) s
WHERE e.Template = N'Single Phase Transformer'
```

o. Either right-click on the query and select **Execute** or click on the **Execute** button on the top menu bar to run this query and to create the view. You should see the following message in the Results grid and a green light in the bottom message bar of PI SQL Commander Lite.

p. Return to the Views folder in the Object Explorer hierarchy. You may need to right-click on the view folder and **Refresh** it to see your new view. Right-click on the view, **My Pole Transformer Loads**, and **Execute Predefined Query…** The query will return the first 100 rows of the view.



q. Edit the query by deleting the “TOP 100” specification in the SELECT statement. Either right-click on the query and select **Execute** or click on the **Execute** button on the top menu bar to rerun the query. This time you should get all 110,940 rows.



Develop the Model in Python

The following code creates a multivariable linear regression model for all 60 transformers. We will use the “adodbapi” package for Python to connect directly to PI through the view previously created using the PI SQL Client. The code will publish coefficients for each modelling equation to a table in MS SQL. This table will be reference by AF allowing us to easily reproduce these models through the Single Phase Transformer template.

Tip

To run the code with your PI SQL View, you’ll want to replace “Pole Transformer Loads” with “My Pole Transformer Loads” in the first cell of this Jupyter Notebook

a. Open Jupyter Notebooks from the windows Taskbar,  . Open the Python script, “Load Prediction - Example 2 (PI SQL Client)”.

b. In the first cell, change the select statement to **'SELECT * FROM [My Pole Transformer Loads]').** This will cause Python to access your PI SQL Client view and create the dataframe, **poleTransformerLoads**. Select this cell and hit **Shift+Enter**. Wait until you see the table preview before moving to the next cell.

```
# EXAMPLE 2 - Create linear regression models for all 60 pole transformers.
# Read PI data via PI SQL Client using the adodbapi package to establish an OLEDB connection.

# Import Python packages.
import adodbapi as ado                          # Support for accessing PI SQL Client - import AF/PI data.
import pandas as pd                             # Dataframe support.
from sklearn.linear_model import LinearRegression # Linear regression model from the scikit-learn package.

# Connect to "Distribution Network Lab" model in AF through the PI SQL Client.
PI_connection=ado.connect("Provider=PISQLClient;Data Source=PISRV01\Distribution Network;Integrated Security=SSPI;")

# Create a cursor object to access the data serer for the "Distribution Network Lab" database in AF.
PI_cursor = PI_connection.cursor()

# Select the entire table using the view configured in PI SQL Commander.
PI_cursor.execute('SELECT * FROM [My Pole Transformer Loads]')

# Extract first row, index=0, to get column names for use as dataframe headers
columnNames = [ x[0] for x in PI_cursor.description]
print("PI View column Names:\n\n", columnNames)

# Unpack the cursor rows into a dataframem.
poleTransformerLoads = pd.DataFrame([dict(zip(columnNames, row)) for row in PI_cursor], columns=columnNames)

# Take a look to see if everythking worked.
poleTransformerLoads.head()
```

```
PI View column Names:

 ['Name', 'TimeStamp', 'Loading', 'Maximum KVA', 'Rated KVA', 'Wh Delivered Load', 'Wh Delivered Load - 14d', 'Wh Del
ivered Load - 7d', 'Ambient Temperature', 'Relative Humidity', 'Wind Speed']
```

	Name	TimeStamp	Loading	Maximum KVA	Rated KVA	Wh Delivered Load	Wh Delivered Load - 14d	Wh Delivered Load - 7d	Ambient Temperature	Relative Humidity	Wind Speed
0	PT_XYZ0342	2017-06-15 00:00:00	3.65	0.1	10.0	365.0	369.0	370.0	68.0	93.0	7.0
1	PT_XYZ0342	2017-06-15 01:00:00	3.65	0.1	10.0	365.0	370.0	370.0	68.0	93.0	6.0
2	PT_XYZ0342	2017-06-15 02:00:00	3.64	0.1	10.0	364.0	370.0	371.0	68.0	93.0	6.0
3	PT_XYZ0342	2017-06-15 03:00:00	3.64	0.1	10.0	364.0	371.0	346.0	68.0	90.0	0.0
4	PT_XYZ0342	2017-06-15 04:00:00	0.00	0.1	10.0	0.0	7.0	0.0	67.0	93.0	0.0

c. Rename columns with shorter names. Create a smaller dataframe, **modellingData**, containing only the values need for predicting transformer loads. Select this cell and hit **Shift+Enter**.

```
# Rename some columns with shorter names to make them easier to work with.
poleTransformerLoads.rename(columns = {'Name' :'Transformer'}, inplace = True)
poleTransformerLoads.rename(columns = {'Wh Delivered Load':'Wh Load'}, inplace = True )
poleTransformerLoads.rename(columns = {'Wh Delivered Load - 14d':'Wh Load-14d'}, inplace = True )
poleTransformerLoads.rename(columns = {'Wh Delivered Load - 7d':'Wh Load-7d'}, inplace = True )
poleTransformerLoads.rename(columns = {'Ambient Temperature':'Temperature'}, inplace = True )
poleTransformerLoads.rename(columns = {'Relative Humidity':'Humidity'}, inplace = True )
poleTransformerLoads.rename(columns = {'Wind Speed':'Wind'}, inplace = True )

# Define second dataframe with just data needed for our modelling.
modellingData = poleTransformerLoads[['Transformer', 'TimeStamp', 'Temperature', 'Humidity',
                    'Wind','Wh Load', 'Wh Load-7d', 'Wh Load-14d']]

# Peek at the first five rows to make sure things look right.
modellingData.head()
```

	Transformer	TimeStamp	Temperature	Humidity	Wind	Wh Load	Wh Load-7d	Wh Load-14d
0	PT_XYZ0342	2017-06-15 00:00:00	68.0	93.0	7.0	365.0	370.0	369.0
1	PT_XYZ0342	2017-06-15 01:00:00	68.0	93.0	6.0	365.0	370.0	370.0
2	PT_XYZ0342	2017-06-15 02:00:00	68.0	93.0	6.0	364.0	371.0	370.0
3	PT_XYZ0342	2017-06-15 03:00:00	68.0	90.0	0.0	364.0	346.0	371.0
4	PT_XYZ0342	2017-06-15 04:00:00	67.0	93.0	0.0	0.0	0.0	7.0

d. Change the index of **modellingData** to be the transformer name. Select this cell and hit **Shift+Enter**.

```
# In order to analyze transformers individually, we need to set the datframe's index to the "Transformer" column.
modellingData = modellingData.set_index("Transformer", drop=False)

# Take a look, see the difference?
modellingData.head()
```

	Transformer	TimeStamp	Temperature	Humidity	Wind	Wh Load	Wh Load-7d	Wh Load-14d
Transformer								
PT_XYZ0342	PT_XYZ0342	2017-06-15 00:00:00	68.0	93.0	7.0	365.0	370.0	369.0
PT_XYZ0342	PT_XYZ0342	2017-06-15 01:00:00	68.0	93.0	6.0	365.0	370.0	370.0
PT_XYZ0342	PT_XYZ0342	2017-06-15 02:00:00	68.0	93.0	6.0	364.0	371.0	370.0
PT_XYZ0342	PT_XYZ0342	2017-06-15 03:00:00	68.0	90.0	0.0	364.0	346.0	371.0
PT_XYZ0342	PT_XYZ0342	2017-06-15 04:00:00	67.0	93.0	0.0	0.0	0.0	7.0

e. Leverage the connection to PI SQL Client and **Pole Transformer Loads** view to get the list of transformer names. Select this cell and hit **Shift+Enter**.

```
# Use our existing PI View connectiont to select just the transformer names from the view configured in
# PI SQL Commander.
PI_cursor.execute('SELECT DISTINCT Name FROM [Pole Transformer Loads]')

# Create a list for the transformer names.
transformerNames = []

# Populate the list
for row in PI_cursor:
    transformerNames.append(row[0])

# Show tranfomer names list.
print(transformerNames)
```

```
['PT_XYZ0406', 'PT_XYZ0407', 'PT_XYZ0408', 'PT_XYZ0409', 'PT_XYZ0410', 'PT_XYZ0411', 'PT_XYZ0412', 'PT_XYZ0413', 'PT_
XYZ0414', 'PT_XYZ0415', 'PT_XYZ0416', 'PT_XYZ0418', 'PT_XYZ0419', 'PT_XYZ0420', 'PT_XYZ0382', 'PT_XYZ0383', 'PT_XYZ03
84', 'PT_XYZ0385', 'PT_XYZ0387', 'PT_XYZ0388', 'PT_XYZ0389', 'PT_XYZ0390', 'PT_XYZ0391', 'PT_XYZ0392', 'PT_XYZ0393',
'PT_XYZ0394', 'PT_XYZ0395', 'PT_XYZ0396', 'PT_XYZ0397', 'PT_XYZ0398', 'PT_XYZ0399', 'PT_XYZ0400', 'PT_XYZ0401', 'PT_X
YZ0402', 'PT_XYZ0403', 'PT_XYZ0404', 'PT_XYZ0405', 'PT_XYZ0342', 'PT_XYZ0343', 'PT_XYZ0344', 'PT_XYZ0345', 'PT_XYZ034
6', 'PT_XYZ0347', 'PT_XYZ0348', 'PT_XYZ0349', 'PT_XYZ0350', 'PT_XYZ0351', 'PT_XYZ0352', 'PT_XYZ0353', 'PT_XYZ0354',
'PT_XYZ0355', 'PT_XYZ0356', 'PT_XYZ0357', 'PT_XYZ0358', 'PT_XYZ0376', 'PT_XYZ0377', 'PT_XYZ0378', 'PT_XYZ0379', 'PT_X
YZ0380', 'PT_XYZ0381']
```

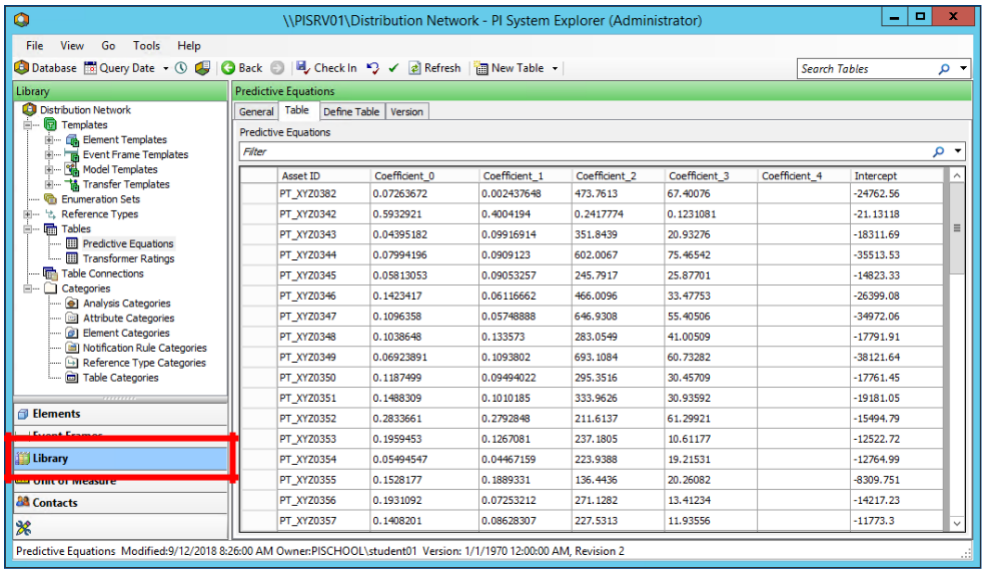
f. Generate linear regression model coefficients for each of the 60 transformers and write them to SQL. Select this cell and hit **Shift+Enter**.

```
# Using the "adodbapi" package, to connect to the "Predictive Equations" MS SQL.

# Set connection parameters.
con_string = 'DRIVER={SQL Server};SERVER=PISRV01;DATABASE=PIWorld;Trusted_Connection=Yes;'

# Connect to "Distribution Network Lab" model in AF through the PI SQL Client.
SQL_connection = ado.connect(con_string)

# Create a cursor object to access the data serer for the "Distribution Network Lab" database in AF.
SQL_cursor = SQL_connection.cursor()

# Create linear regression object from the "sklearn" package we imported earlier.
LinReg = LinearRegression()

# Looping through the trasformer list, Perform linear regression on each transformer.
for transformer in transformerNames:

    # Create dataframe for one transformer.
    transformerData = modellingData.loc[transformer,:]

    # Perform linear regression fit
    LinReg.fit(transformerData[["Wh Load-7d","Wh Load-14d","Temperature","Humidity"]],transformerData["Wh Load"])

    # Update asset ID value with the name of this transformer.
    asset_id = "'"+transformer+"'"

    # Print equation.
    print(transformer, "Eq:\n", LinReg.coef_[0], "*","'Wh Delivered Load - 7d' + ",LinReg.coef_[1],
          "*","'Wh Delivered Load - 14d' + ", LinReg.coef_[2], "*", "'Ambient Temperature' +",
          LinReg.coef_[3], "*","'Relative Humidity' +(",LinReg.intercept_, ")" )

    ## INSERT and UPDATE queries to load table for the first time or update an existing one.
    # Construct query to add this transfomer's model coefficients.
    insert_query = f'INSERT [Predictive Equations] ([Asset ID], Coefficient_0, Coefficient_1, Coefficient_2, Coeffic

    update_query = f'UPDATE [Predictive Equations] SET Coefficient_0={LinReg.coef_[0]}, Coefficient_1={LinReg.coef_[

    # Insert this record into the "Predictive Equations" table.
    SQL_cursor.execute(update_query)

# Commit the queries to write the data into SQL.
SQL_connection.commit()

# Close database connections.
SQL_connection.close()
PI_connection.close()
```

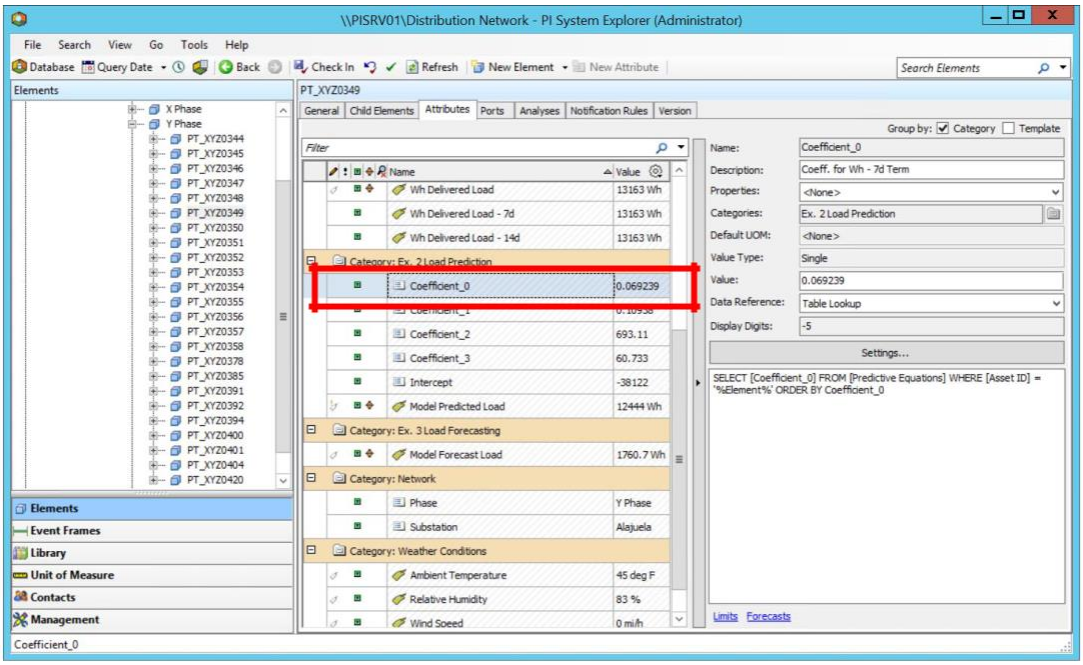
```
PT_XYZ0378 Eq:
 0.15878008013229636 * 'Wh Delivered Load - 7d' +  0.17917457192857675 * 'Wh Delivered Load - 14d' +  110.23388419266524 * 'A
mbient Temperature' + -0.2658384322606984 * 'Relative Humidity' +( -3314.933455710232 )
PT_XYZ0415 Eq:
 -0.001999771117489486 * 'Wh Delivered Load - 7d' +  0.0035564537270706955 * 'Wh Delivered Load - 14d' +  0.0514892870018438
4 * 'Ambient Temperature' + -0.0032873513352864587 * 'Relative Humidity' +( -1.0928457626402812 )
PT_XYZ0407 Eq:
 0.22133353596401245 * 'Wh Delivered Load - 7d' +  0.18940221947528418 * 'Wh Delivered Load - 14d' +  303.2482562107521 * 'Am
bient Temperature' + 7.845159067779704 * 'Relative Humidity' +( -14852.0518428231 )
PT_XYZ0385 Eq:
 0.17033499389302167 * 'Wh Delivered Load - 7d' +  -0.13442528680210902 * 'Wh Delivered Load - 14d' +  -0.001006904293367613
* 'Ambient Temperature' + 0.002808240581481683 * 'Relative Humidity' +( 10.552971591136941 )
PT_XYZ0346 Eq:
 0.14229519100681018 * 'Wh Delivered Load - 7d' +  0.06105023128834538 * 'Wh Delivered Load - 14d' +  466.02179316494465 * 'A
mbient Temperature' + 33.496691778996954 * 'Relative Humidity' +( -26399.777514147783 )
PT_XYZ0408 Eq:
 0.22918154817171893 * 'Wh Delivered Load - 7d' +  0.1723254292743076 * 'Wh Delivered Load - 14d' +  207.3195523807009 * 'Amb
ient Temperature' + 5.783909412754433 * 'Relative Humidity' +( -8273.493697771371 )
```

Test Model - AF Analytics Backfill

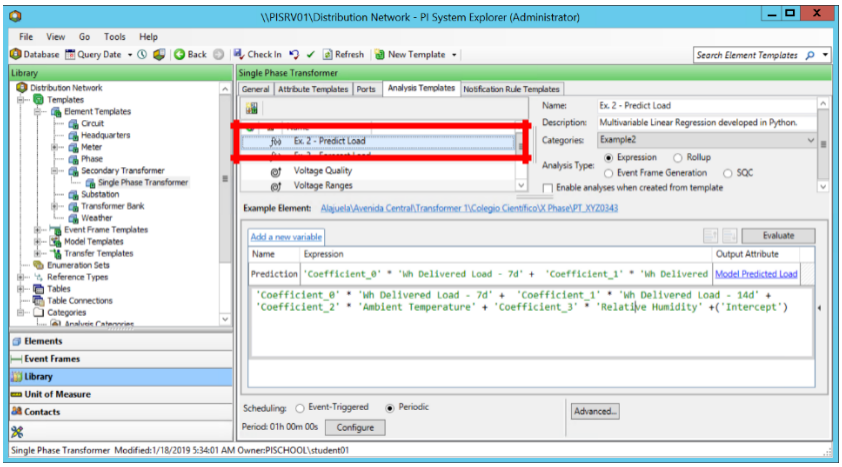
a. Open PI System Explorer and go to the **Library** view. Expand the **Tables** branch and click on the **Predictive Equations** table. Select the **Table** tab to view the model coefficients generated by the Python script. Although the script wrote the coefficients to a MS SQL table, we have imported them into the AF model for use in AF Analytics.



b. Return to the **Elements** view in PI System Explorer. Select any Single Phase Transformer element, **PT_XYZ…** Select the element's **Attributes** tab. Find the attributes under **the "Ex. 2 Load Prediction".** Select the **Coefficient_1** attribute and note the used of the Table Lookup Data Reference to make each transformer's coefficients available within the element.

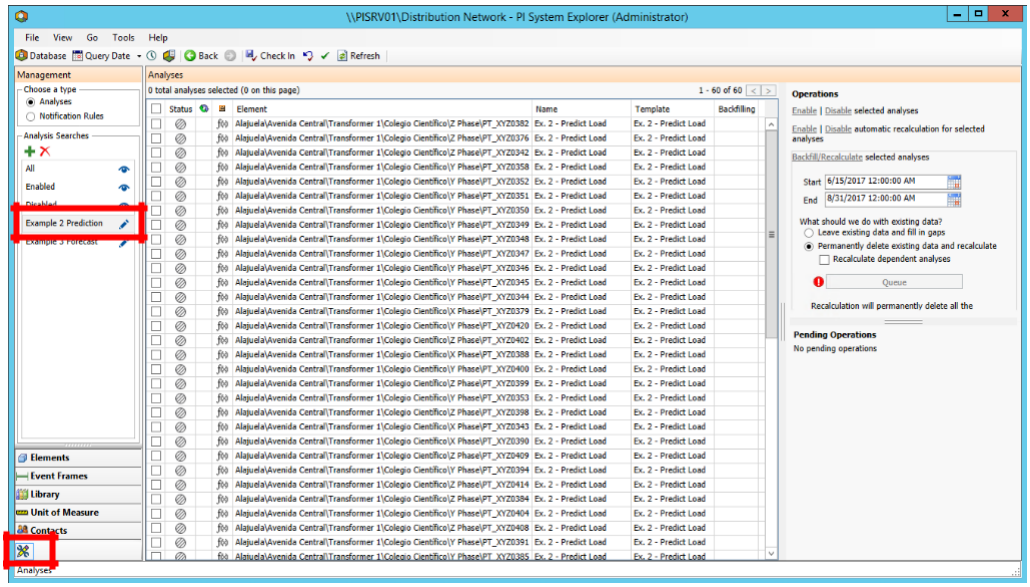


c. Select the element's **Analyses** tab. Select the **Ex. 2 – Predict Load**. Here the linear model developed in Python has been expressed in terms of AF Attributes. This expression was configured as part of the template used for all Pole (Single Phase) Transformers.

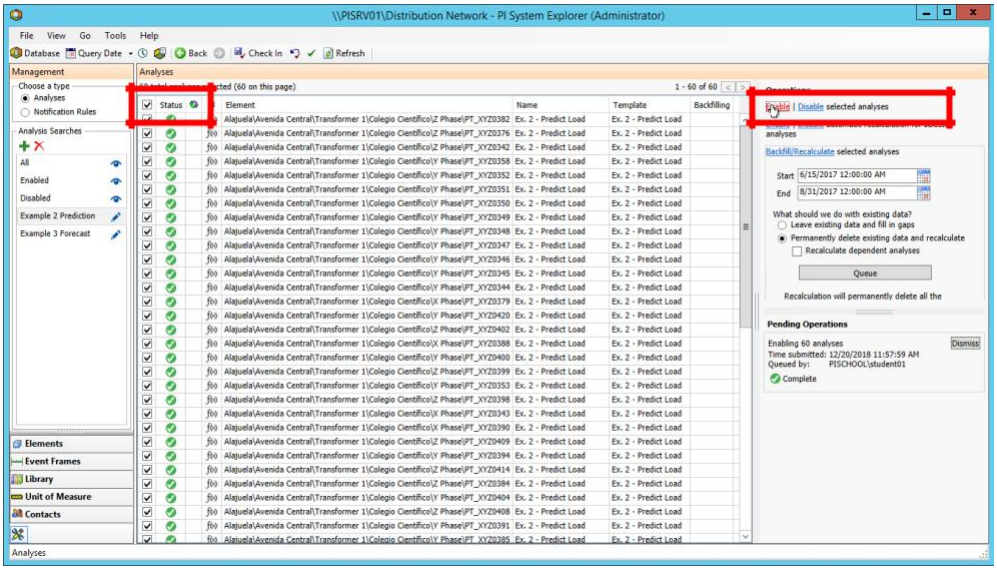


d. Next, we will use the backfilling feature of AF Analytics to populate he result of the predictive model into PI points configured for all 60 transformers.

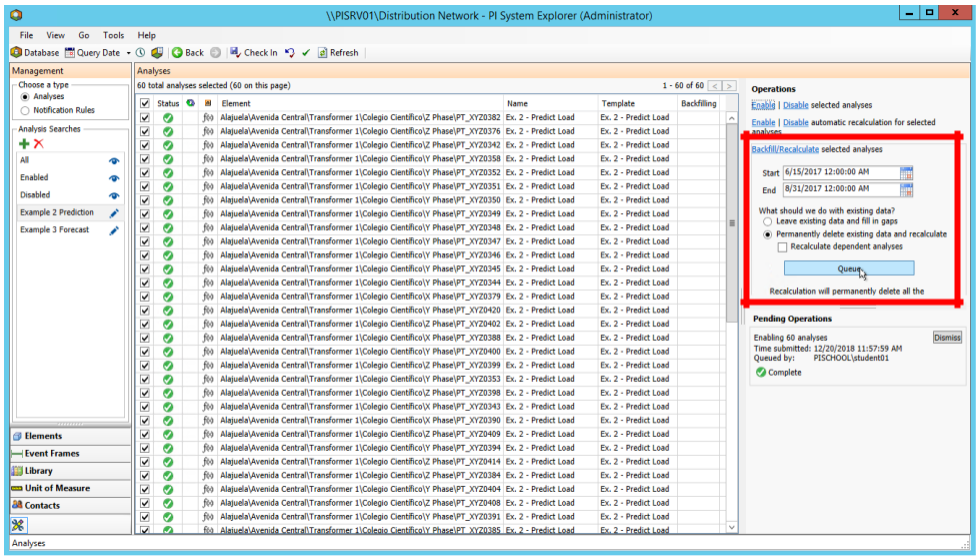
Go to the **Management** view in PI System Explorer (icon in lower left-hand corner). Select **Example 2 Prediction** under Analysis Searches. This allows us to manage just the AF Analytic configured for Example 2 of the lab.



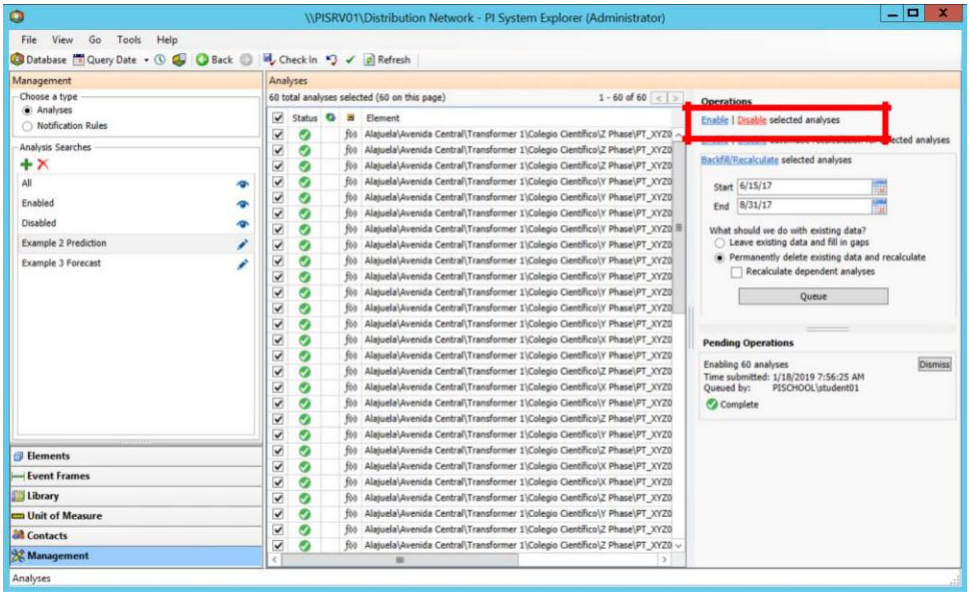
e. Check the **Status** box at the top of this column to select all transformers and click **Enable** to start this group of 60 analytics.



f. Set the Backfill Start and End times to **6/15/17** and **8/31/17**, respectively. Check **Permanently delete existing data and recalculate**. The click **Queue** button to start the Backfill operation.



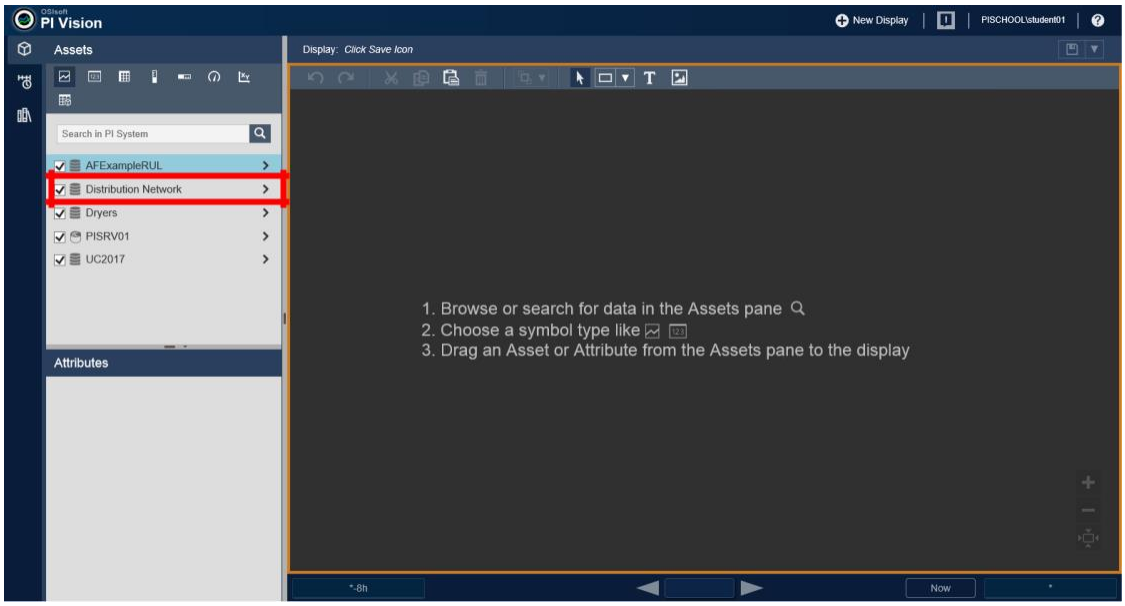
g. Once all 60 analyses have been backfilled, click **Disable** to stop them. We are only testing our new model over a controlled time range. It's not ready for general use yet.



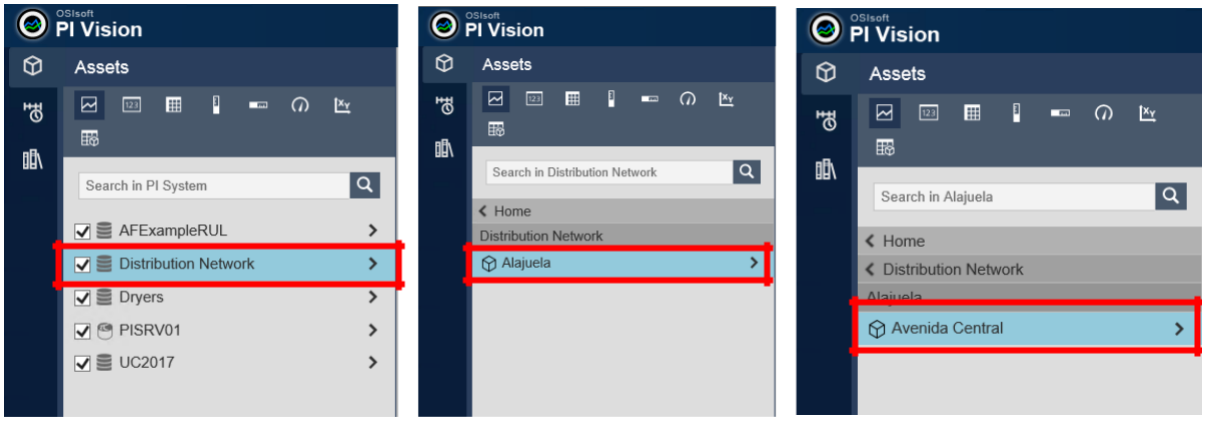
Evaluate Model – PI Vision Collection

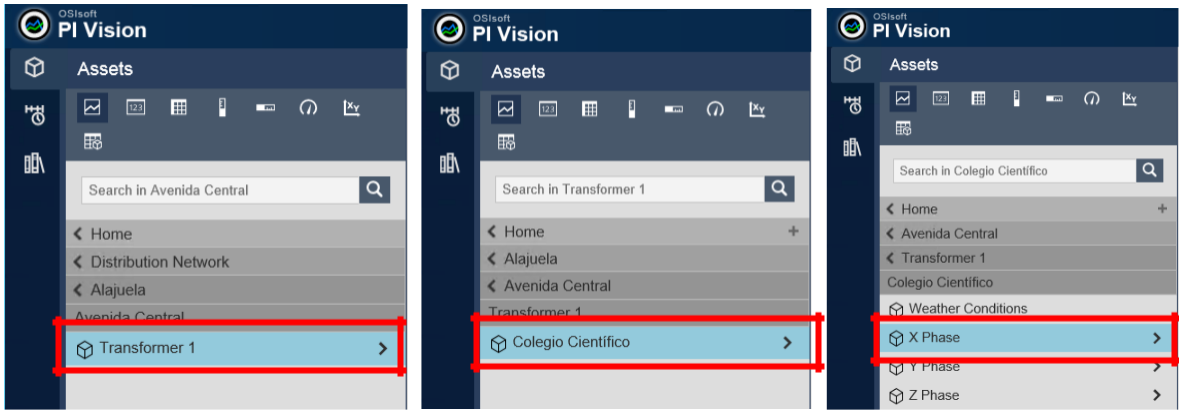
PI Vision provides a powerful feature called a “Collection” which makes it easy to group visualization objects and apply them to a set of AF Elements derived from the same AF Template. In this step, we will be creating a PI vision collection to show the backfilled results for all 60 transformers in our network.

a. Open PI Vision using the “Ex. 2 Build Display” shortcut on the desktop. This will open a new display.

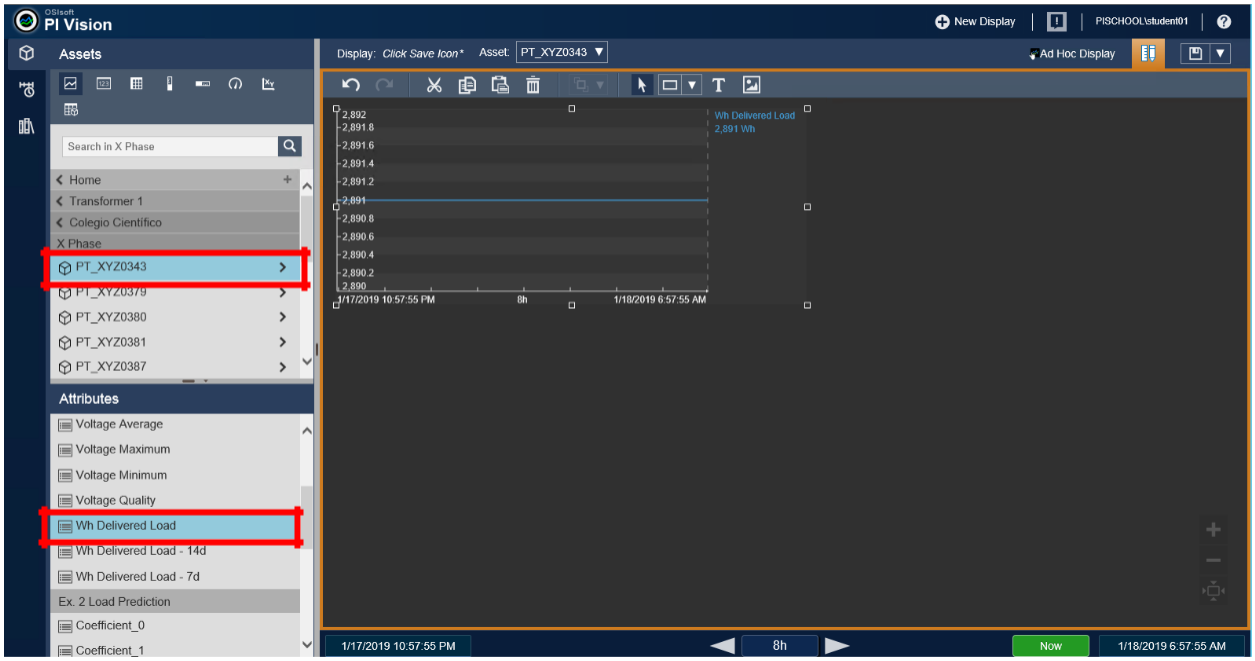


b. In the **Asset** pane on the left, select the **Distribution Network** AF model and navigate the model’s hierarchy by selecting the arrow icon, Drilldown the tree using the following path, **Distribution Network \ Alajuela \ Avenida Central \ Transformer 1 \ Colegio Cientifico \ X Phase**.

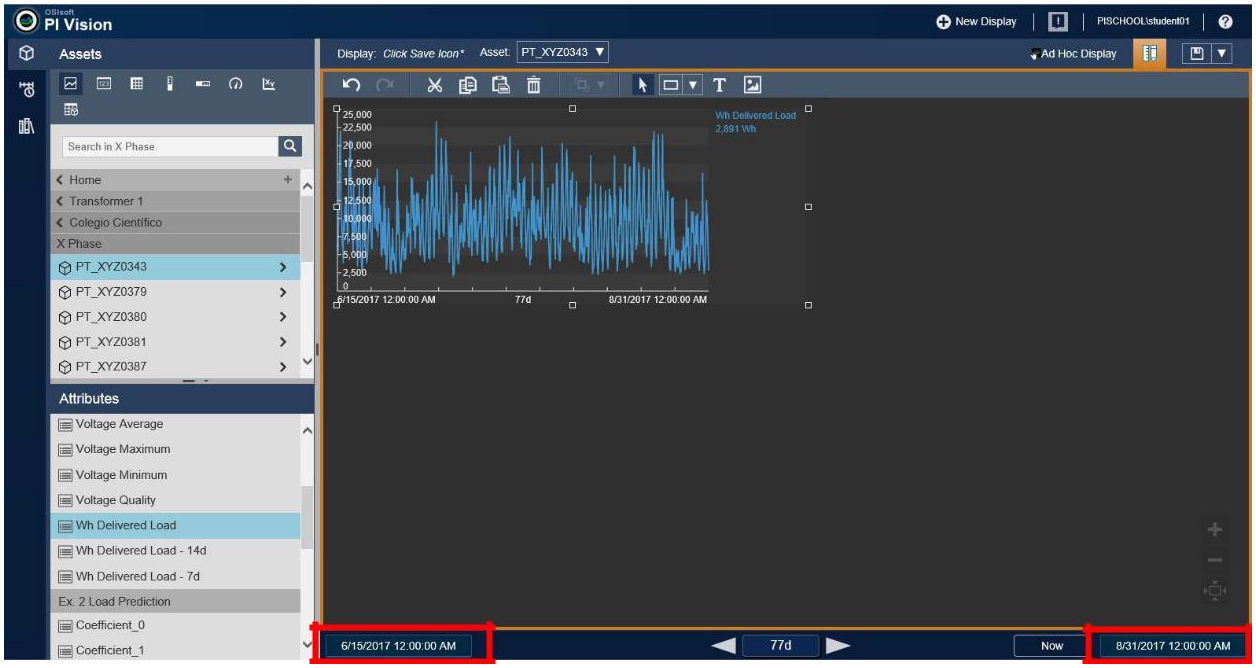




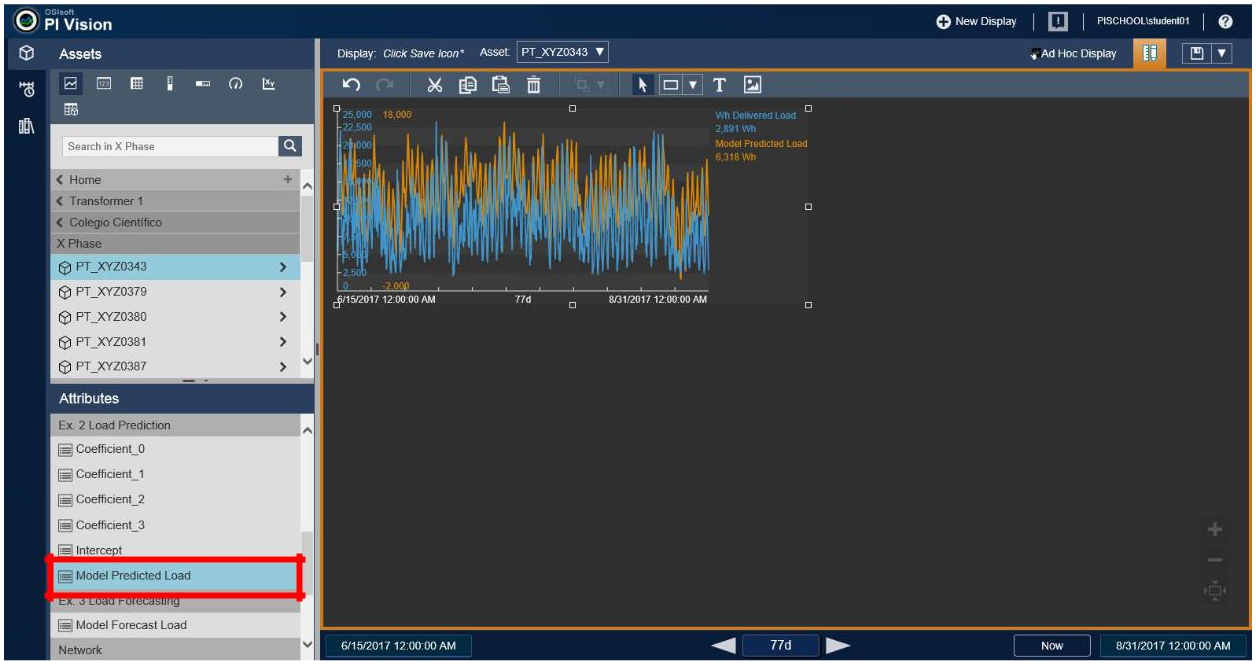
c. Select the transformer element **PT_XYZ_0343**. From the attributes list below, select and drag the **Wh Delivered Load** attribute on to the display canvas. This will add a trend of the transformer loading on the display. You won't see data until we reset the display time range to match our dataset, 6/15/17 to 8/31/17.



d. Edit the start and end times at the bottom of the PI Vision display to be **6/15/17** and **8/31/17**.

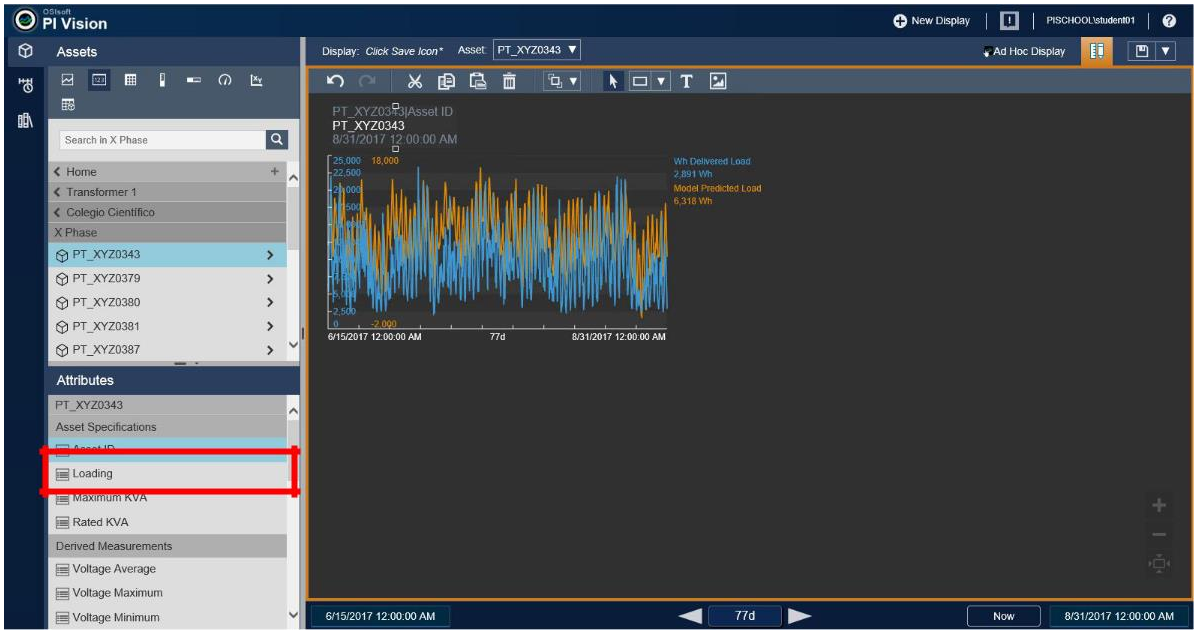


e. Scroll down the attributes list and find the attribute **Model Predicted Load**. Select and drag this attribute onto the display canvas and drop it on the trend object. You should see both traces on the trend.

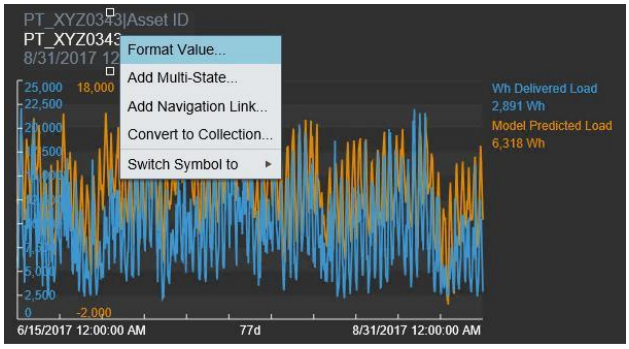


f. At the top of the **Asset** pane, change the default display object type from a Trend to a Value.

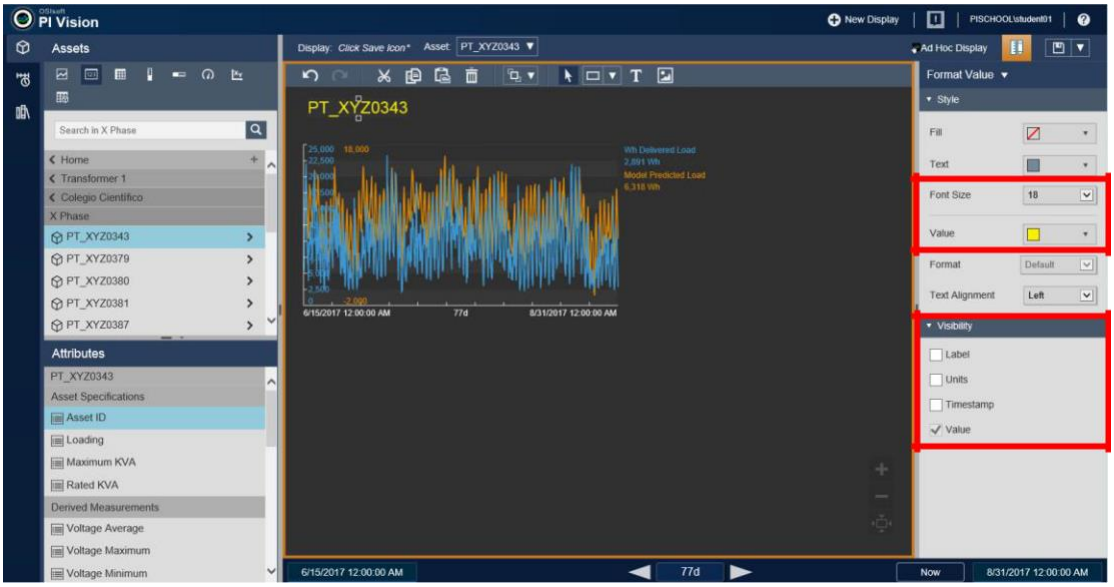
g. Scroll up the attributes list and find the attribute **Asset ID**. Select and drag this attribute onto the display canvas above the trend. The transformer's asset name should appear along with the attribute name and timestamp. (You may need to rearrange the trend and value objects to get them to look like the display below.)



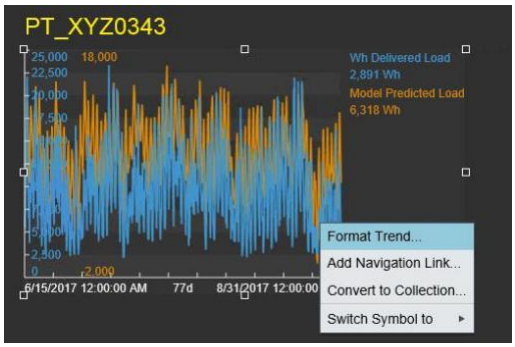
h. Select he value object, right-click on it and choose **Format Value**.



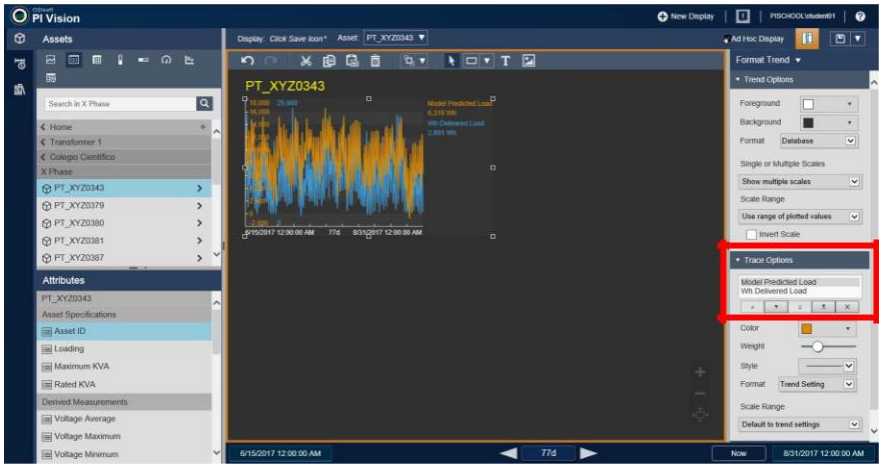
i. In the **Format Value** pane at right, expand the **Visibility** section and uncheck the Label, Units and Timestamp items. In the **Style** section, change the **Font Size** to **18** and the **Value** color to yellow.



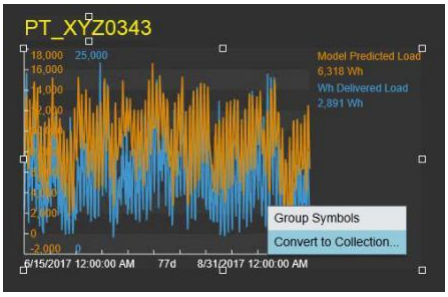
j. Select he value object, right-click on it and choose **Format Trend**.



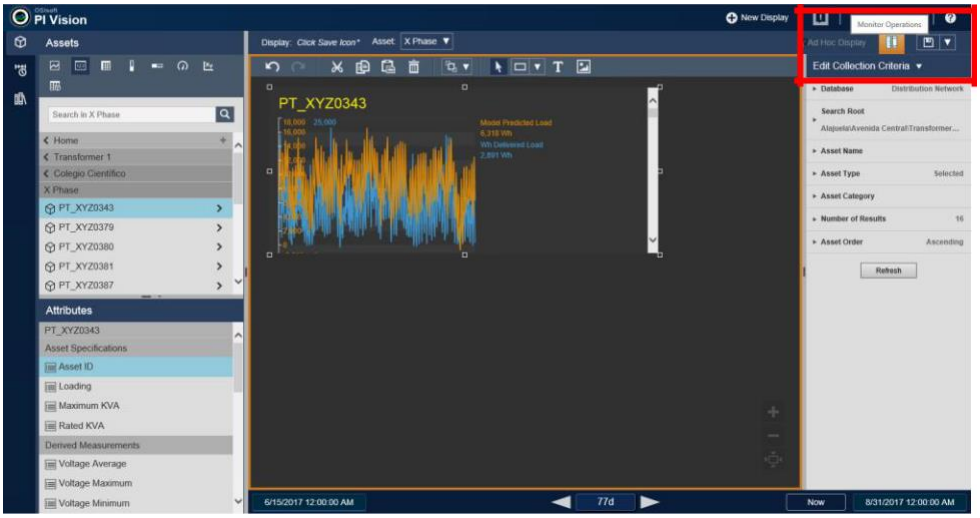
k. In the **Format Trend** pane at right, in the **Trace Options** section, move the **Model Predicted Load** attribute up to the first trace in the trend.



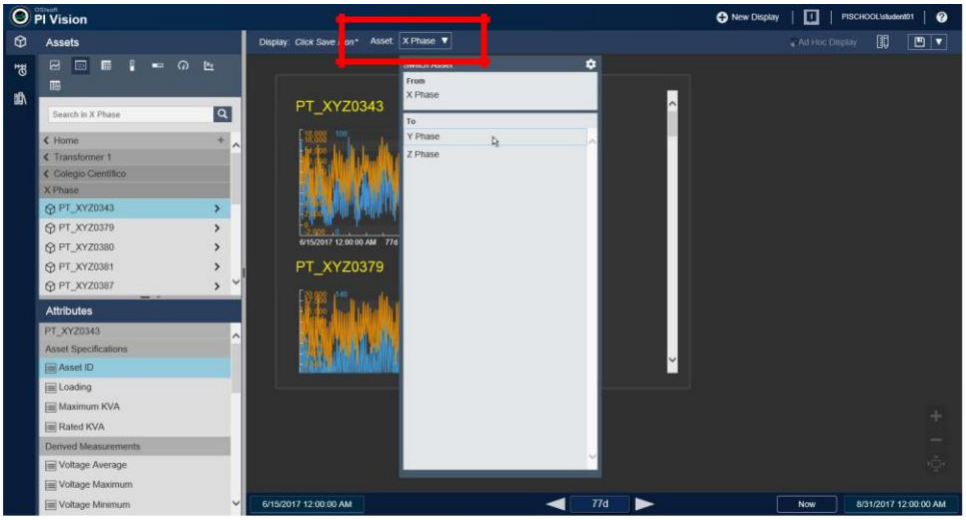
l. Holding the Ctrl key, multiselect the Value and Trend objects. Right-click and choose **Convert to Collection**.



m. Once the display updates, click on the PI Vision display mode icon in the upper right-hand corner of the bowser to switch to the **Monitor Operations** mode.



n. Using the display scrollbar, you will be able to see backfilled results and actual values for each transformer. These will be grouped by the phase, or parent, level of the Distribution Network model's hierarchy. Using the **Asset** dropdown list at the top of the display, select a different phase to different to get a different group.

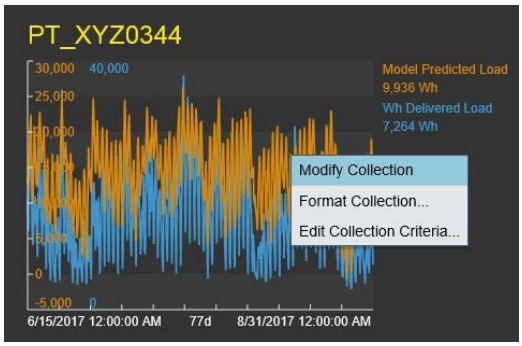


o. Adjustments can be made to make the display easier to use. Here are some suggestions;

1. Close the Asset pane by clicking on its icon. This will give you more display space.

2. Change the display back to **Modify Display** mode, select the Collection and stretch it out to show more assets.

3. Right-click on the collection and choose **Modify Collection**. This will return you to the original group containing the original Value and Trend objects. Here you can resize them and change their formatting.



Click the **Exit Modify Mode** icon in the upper right-hand corner of the object boundary to return to the full display. 

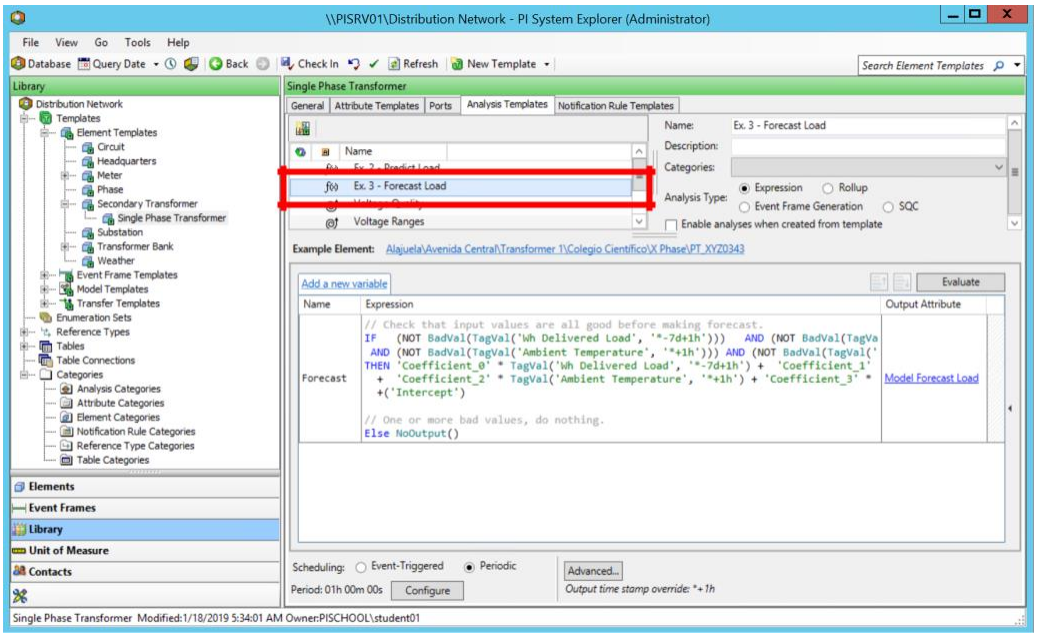
Example 3 - Operationalize Forecast Models for Multiple Transformers

Learning Objectives

- Modify the predictive equation to calculate forecasted values for transformer loads
- Employ Advanced scheduling feature of AF Analytics and PI Future data tags.
- Operationalize forecasting model (assume today is 8/31/17)
- Asses the forecast for all 60 transformers in PI Vision Collection

Create a Forecast from the Predictive Model

a. Go to the **Library** view in PI System Explorer (lower left-hand corner). Select the **Single Phase Transformer** template. Click the **Analysis Templates** tab. Choose the **"Ex. 3 Forecast Load"** analytic.



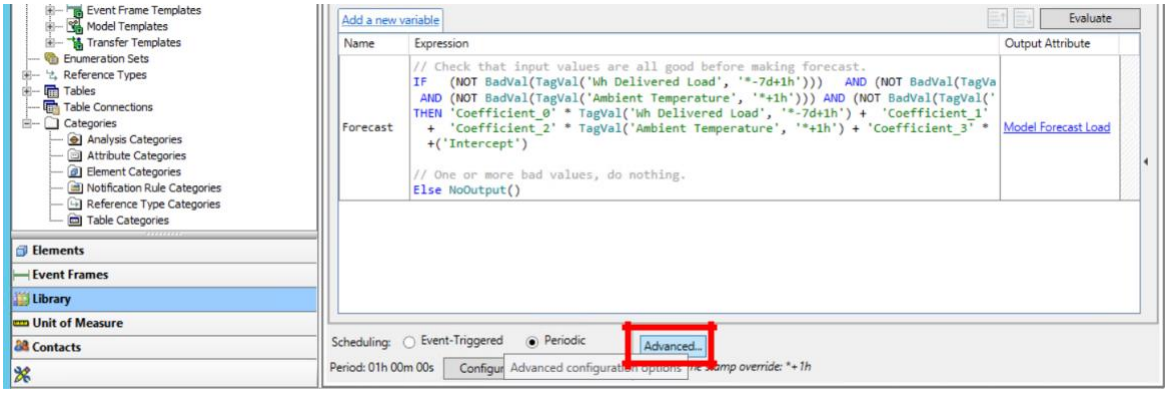
b. Take a closer look at the analytic equation below. First, we have added a conditional check to ensure all forecast-based measurements are good. This reduces the chance of posting an errant forecast value which can erode user confidence.

   Secondly, we have shifted the time references forward one hour. This change requires the terms of the modelling equation to be based on the PI Point attribute, Wh Delivered Load using the TagVal() function in AF Analytics.

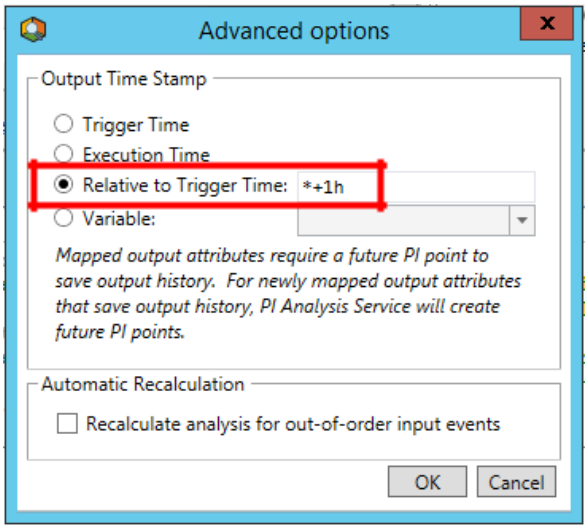
Name	Expression

Forecast	// Check that input values are all good before making forecast.
	IF (NOT BadVal(TagVal('Wh Delivered Load', '*-7d+1h'))) AND (NOT BadVal(TagVal('Wh Delivered Load', '*-14d+1h')))
	AND (NOT BadVal(TagVal('Ambient Temperature', '*+1h'))) AND (NOT BadVal(TagVal('Relative Humidity','*+1h')))
	THEN 'Coefficient_0' * TagVal('Wh Delivered Load', '*-7d+1h') + 'Coefficient_1' * TagVal('Wh Delivered Load', '*-14d+1h')
	+ 'Coefficient_2' * TagVal('Ambient Temperature', '*+1h') + 'Coefficient_3' * TagVal('Relative Humidity','*+1h')
	+('Intercept')
	
	// One or more bad values, do nothing.
	Else NoOutput()

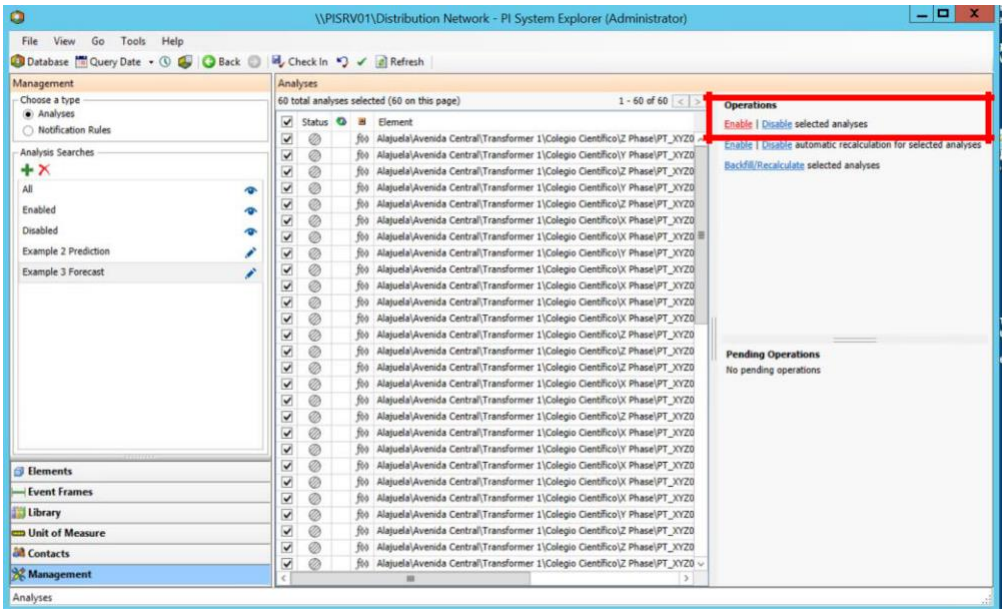
c. By default, AF Analytics posts results at the current time. This analytic is scheduled to run hourly, however, as a forecast, we need to be able to post the results, one hour into the future. Click on **Advanced** to access the dialog to change the result posting rime.



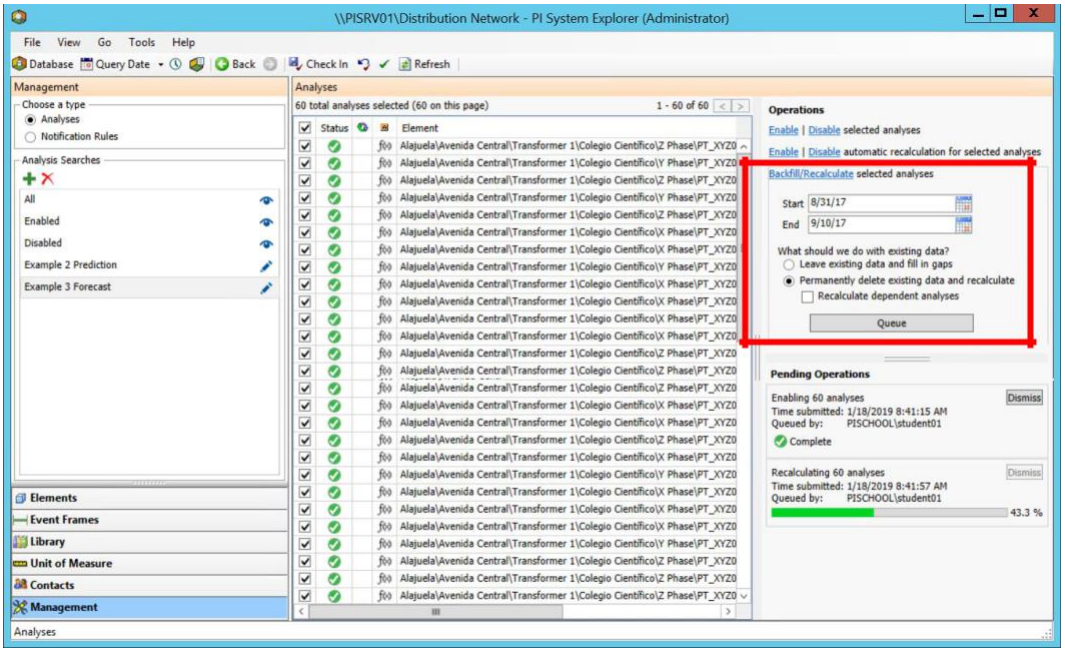
d. We have specified that the output of this analytics be posted in a future data tag one hour past the time of its calculation.



e. We are now ready to test the forecasting analyses. Go to the **Management** view in PI System Explorer (icon in lower left-hand corner). Select **Ex. 3 Load Forecast** under Analysis Searches. Click the checkbox at the top next to Status to select all 60 forecasting analytics. Start them by clicking **Enable**.



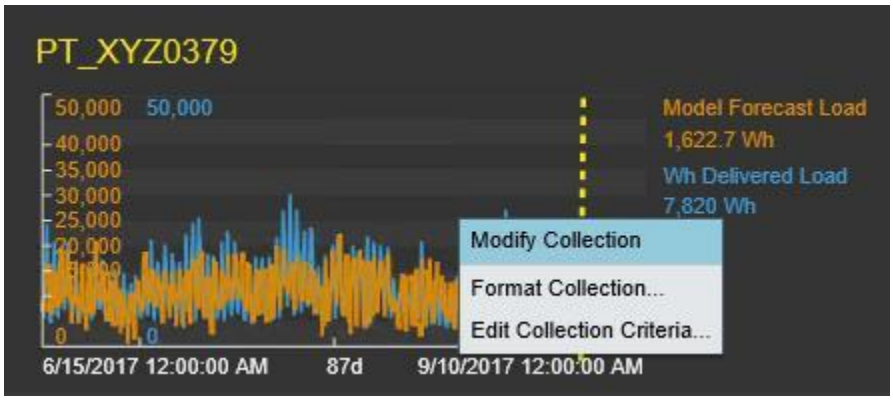
f. Once you get green icons indicating all the analytics have been started, Backfill, or in this case “Forewardfill”, the analysis from 8/31/17 to 9/10/17. (Remember, today is 8/31/17 and we have weather forecast data through the end of September 2017.)



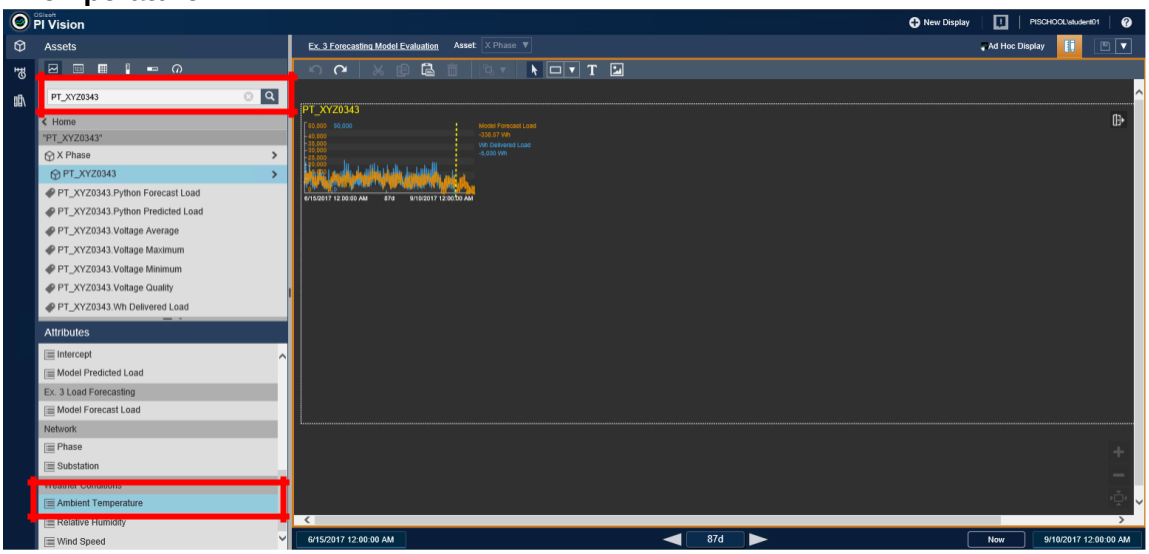
g. Once the backfill is complete, **Disable** the analyses. Return to the desktop and open PI Vision using the desktop shortcut named **"Ex. 3 Forecast Evaluation"**.

h. The dashed yellow lines have been drawn over the trends to show "today", 8/31/17. The forecasted values we have just generated are shown to the right of this line as future data. Let's see how the Model Forecast Loads are affected by the cooler ambient temperatures forecasted for September.

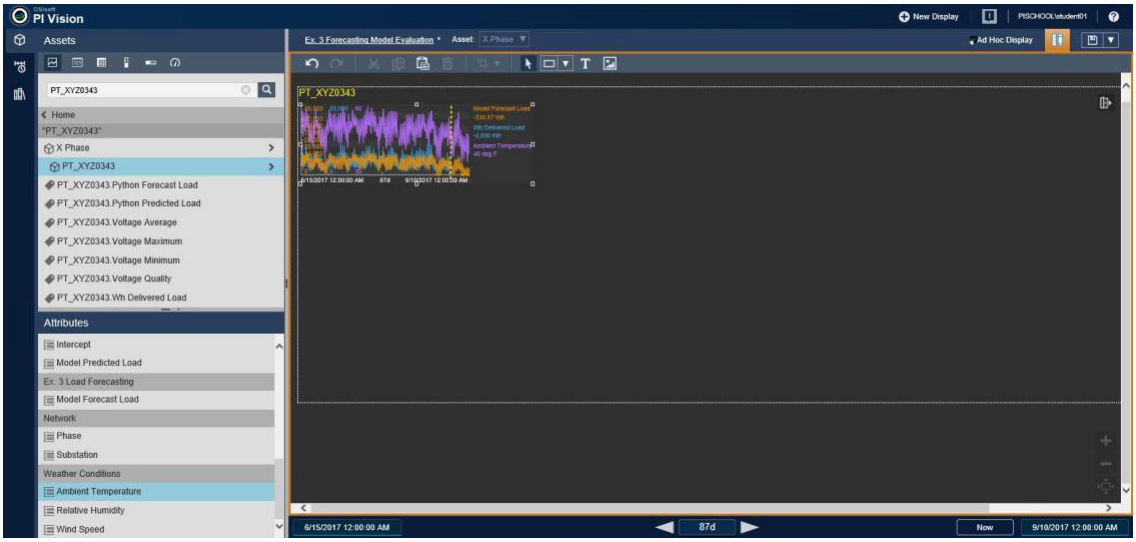
Right-click on any trend in the PI Vision Collection and select **Modify Collection**.



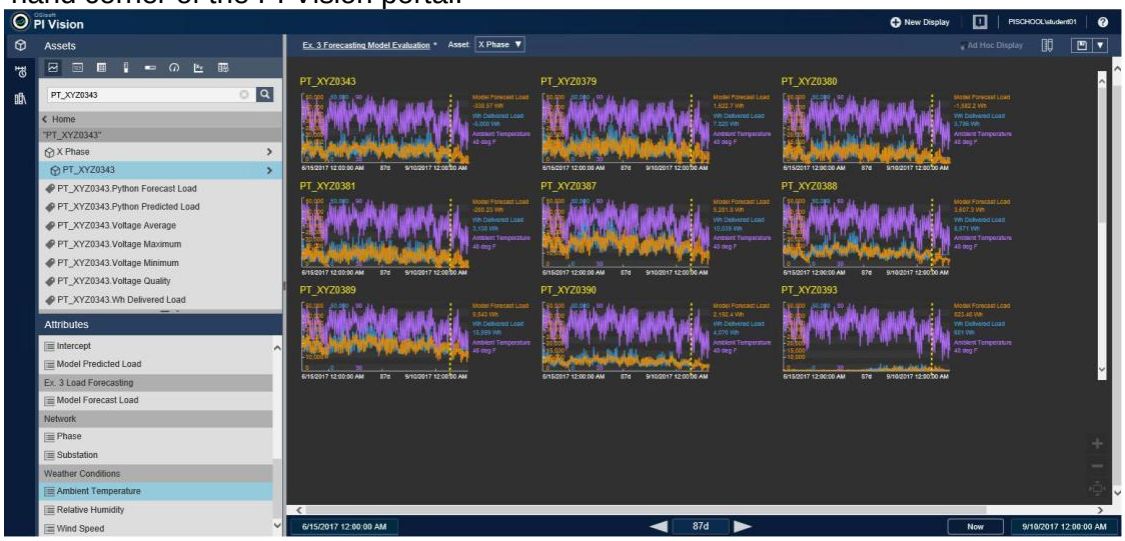
i. Expand the “Assets” pane by clicking  in the upper left-hand corner of the PI Vision portal. In the Search dialog type the name of base element used to configure the PI Vision collection, **PT_XYZ0343**. Scroll down the Attributes list and find **Ambient Temperature**.



j. Select and drag the **Ambient Temperature** attribute onto the trend object in the collection. This will add the temperature forecast data, both historical and future, to the trend.



k. Return PI Vision to “Motoring Operations” mode by clicking  in the upper fight-hand corner of the PI Vision portal.



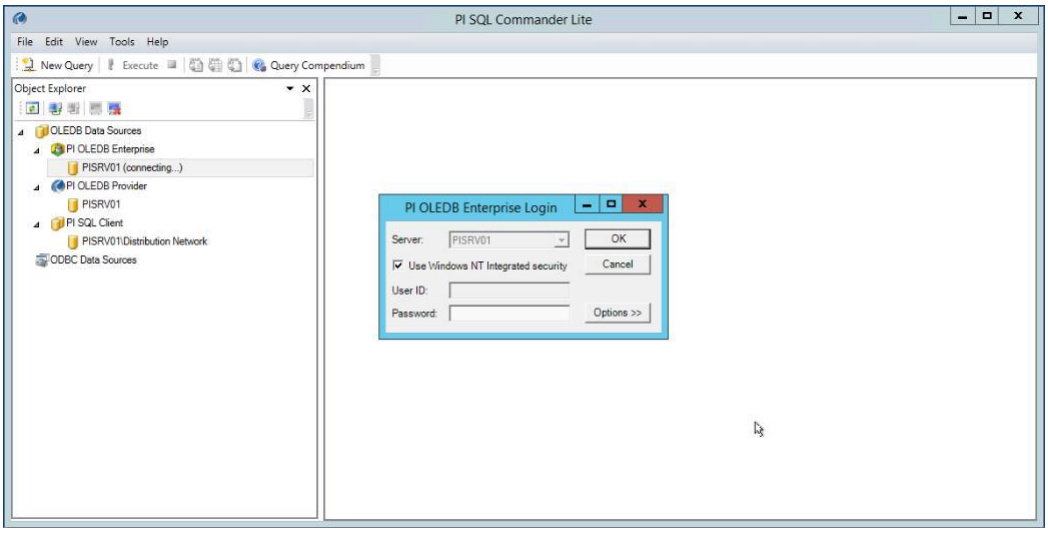
l. Collapse the “Assets” pane to give you a better look. You can double-click on any individual trend for a more detailed examination of how things look.

Appendix A - Python Access to PI using PI OLEDB Enterprise

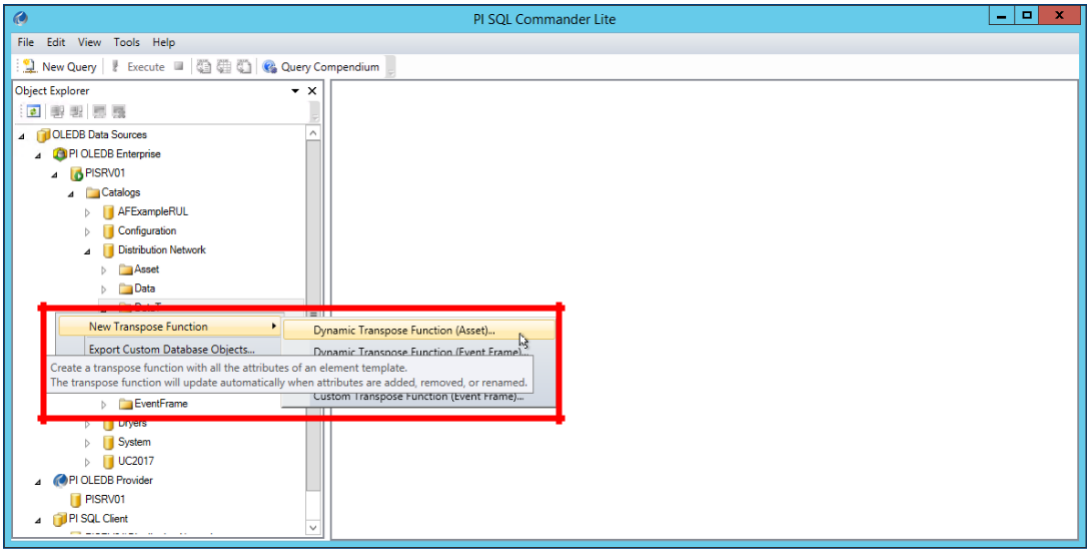
Create a PI View – PI OLEDB Enterprise

PI Data can be imported directly into Python through a view configured in PI OLEDB Enterprise. Views are configured in PI SQL Commander Lite. Here are the steps to create a view containing the data used in the lab example.

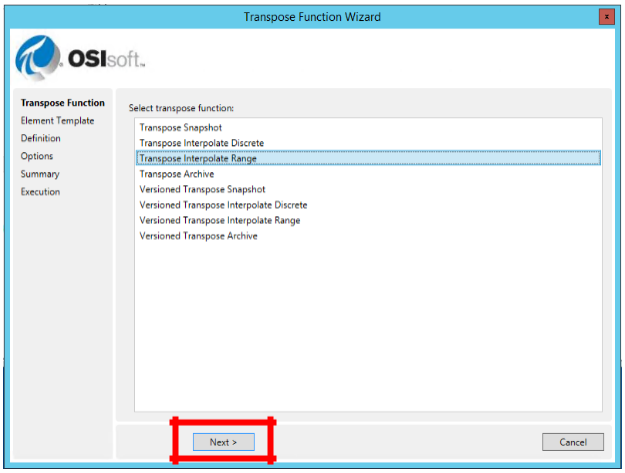
a. Open PI SQL Commander Lite from the Windows taskbar,  .
b. Under the **PI OLEDB Enterprise** branch of the Object Explorer hierarchy, select the **PISRV01,** right-click and **Connect.** Click **Ok.**



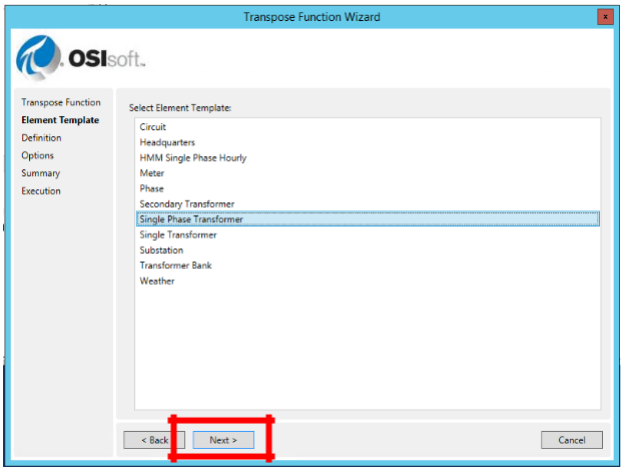
c. Expand the **Distribution Network** catalog in Object Explorer hierarchy. Expand the **Data T** folder. Then right-click on the Data T folder and slide through the menu to select **Dynamic Transpose Function (Asset)**.



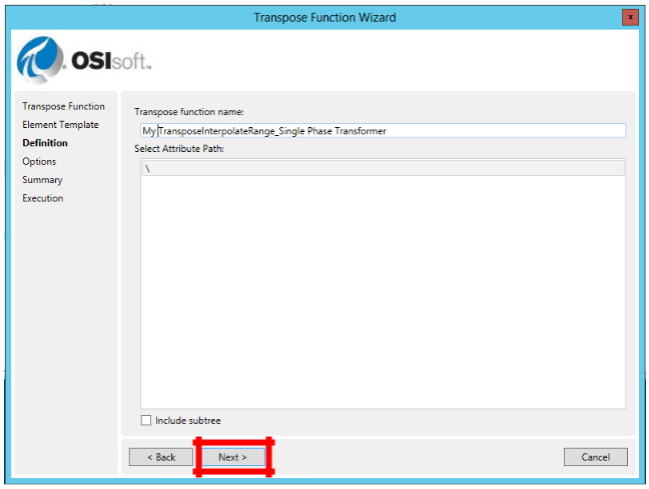
d. In the Transpose Function Wizard dialog choose **Transpose Interpolate Range**. Click **Next**.



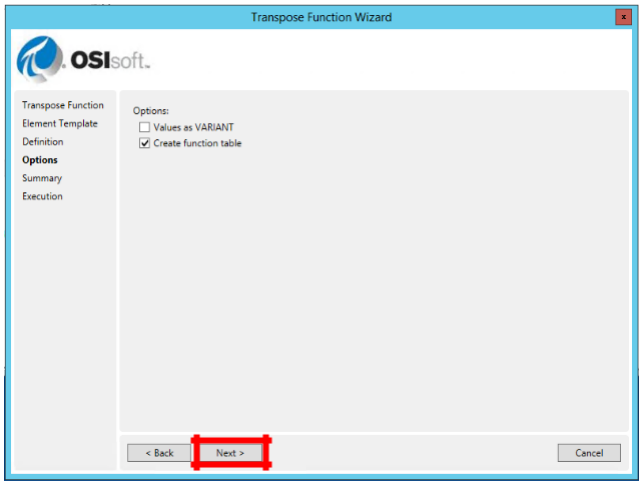
e. Choose **Single Phase Transformer**. Click **Next**.



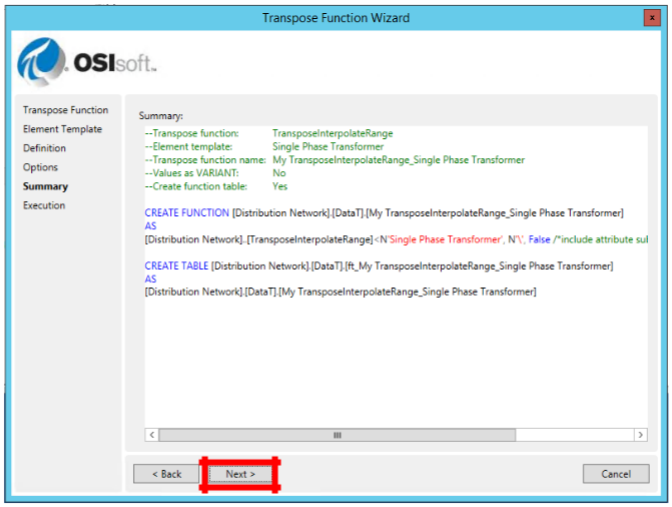
f. Change the function name to **"My TransposeInterpolateRange_Single Phase Transformer".** Click **Next**.



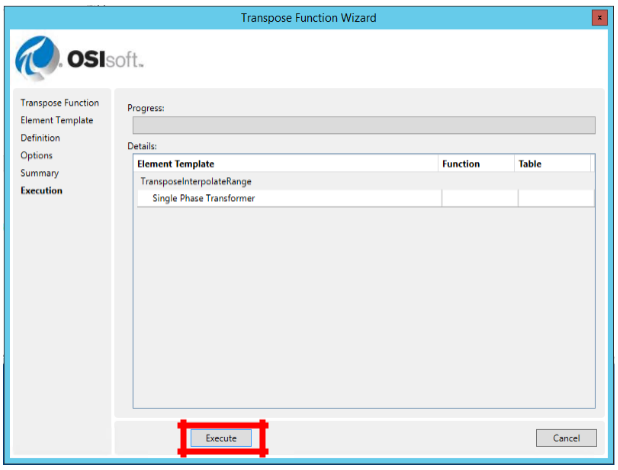
g. Click **Next**.



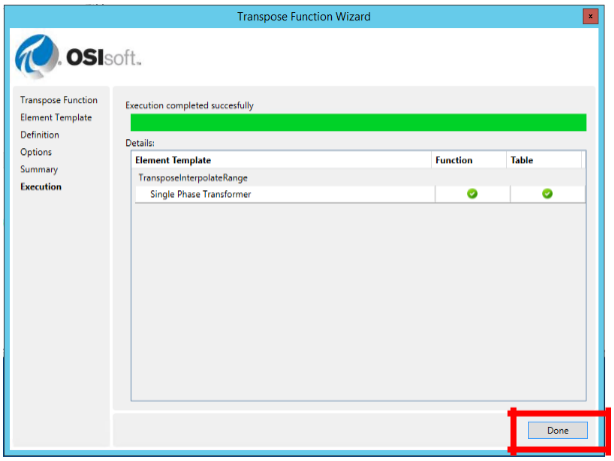
h. Click **Next**.



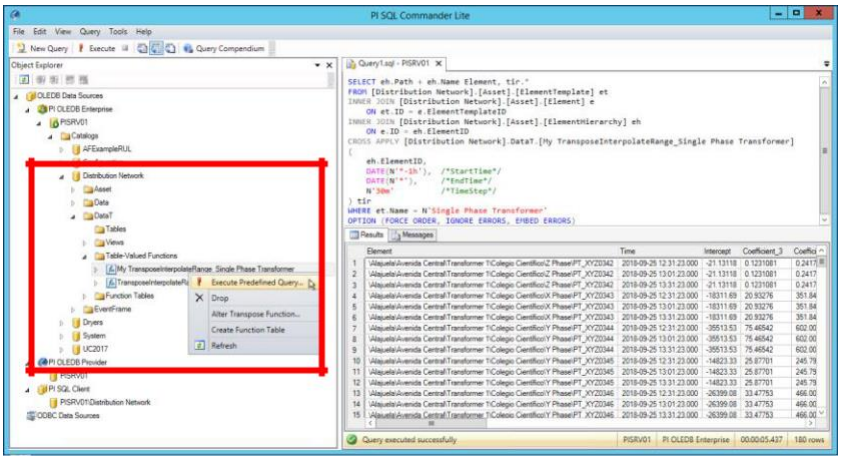
i. Click **Execute**.



j. Click **Done**.



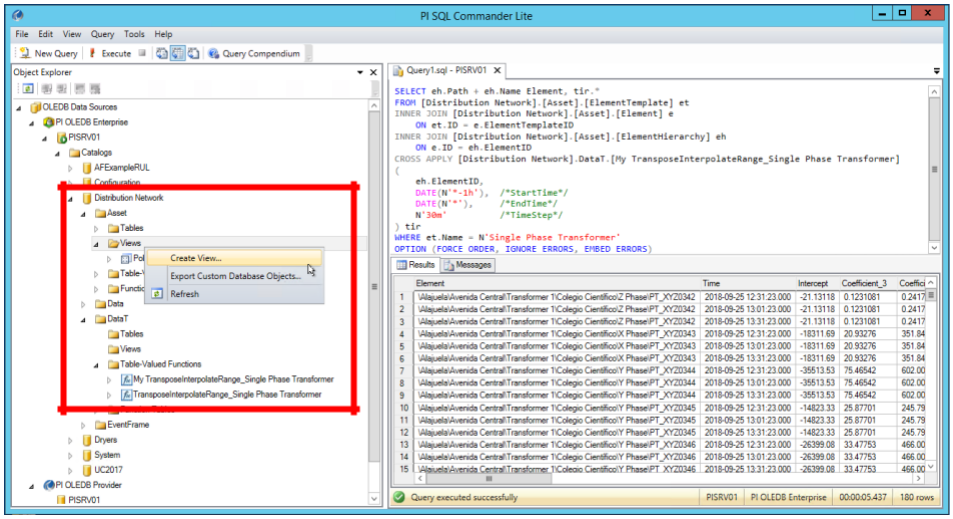
k. Expand the **Table-Valued Functions** folder to see you newly created function. Right-click on **My TransposeInterpolateRange_Single Phase Transformer** and **Execute Predefined Query…**



l. Select the entire predefined query shown in the query viewer and copy it into the clipboard.

```
SELECT eh.Path + eh.Name Element, tir.*
FROM [Distribution Network Lab].[Asset].[ElementTemplate] et
INNER JOIN [Distribution Network Lab].[Asset].[Element] e
    ON et.ID = e.ElementTemplateID
INNER JOIN [Distribution Network Lab].[Asset].[ElementHierarchy] eh
    ON e.ID = eh.ElementID
CROSS APPLY [Distribution Network Lab].DataT.[My TransposeInterpolateRange_Single Phase Transformer]
(
    eh.ElementID,
    DATE(N'*-1h'),  /*StartTime*/
    DATE(N'*'),     /*EndTime*/
    N'30m'          /*TimeStep*/
) tir
WHERE et.Name = N'Single Phase Transformer'
OPTION (FORCE ORDER, IGNORE ERRORS, EMBED ERRORS)
```

m. Under the **Asset** folder, find the **Views** folder. Right-click on **Views** select **Create View.**



n. The following query should now appear in the query window.

```
CREATE VIEW [Distribution Network].[Asset].[<view name>]
AS
<view definition>
```

o. Replace <view definition> with the query saved in the clipboard

```
CREATE VIEW [Distribution Network].[Asset].[<view name>]
AS
SELECT eh.Path + eh.Name Element, tir.*
FROM [Distribution Network].[Asset].[ElementTemplate] et
INNER JOIN [Distribution Network].[Asset].[Element] e
    ON et.ID = e.ElementTemplateID
INNER JOIN [Distribution Network].[Asset].[ElementHierarchy] eh
    ON e.ID = eh.ElementID
CROSS APPLY [Distribution Network].DataT.[My TransposeInterpolateRange_Single Phase Transformer]
(
    eh.ElementID,
    DATE(N'*-1h'),  /*StartTime*/
    DATE(N'*'),     /*EndTime*/
    N'30m'          /*TimeStep*/
) tir
WHERE et.Name = N'Single Phase Transformer'
OPTION (FORCE ORDER, IGNORE ERRORS, EMBED ERRORS)
```

p. Modify this query by;
   a. Replacing, within the brackets, “<view name>” with “My Pole Transformer Loads”.
   b. Replace the SELECT statement by cut-and-pasting the following text:

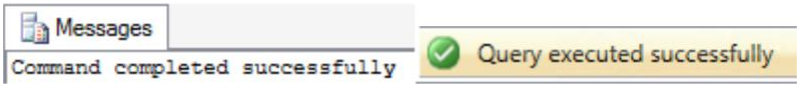
      eh.Name, tir.Time Timestamp, tir.Loading, tir.[Maximum KVA], tir.[Rated KVA], tir.[Wh Delivered Load], tir.[Wh Delivered Load - 7d], tir.[Wh Delivered Load - 14d], tir.[Ambient Temperature], tir.[Relative Humidity], tir.[Wind Speed]

   c. Change the Start Time from ‘y’ to ‘15-jun-2017'.
   d. Change the End Time from ‘t’ to '31-aug-2017'.
   e. Change the Time Step from ‘30m’ to ‘1h.

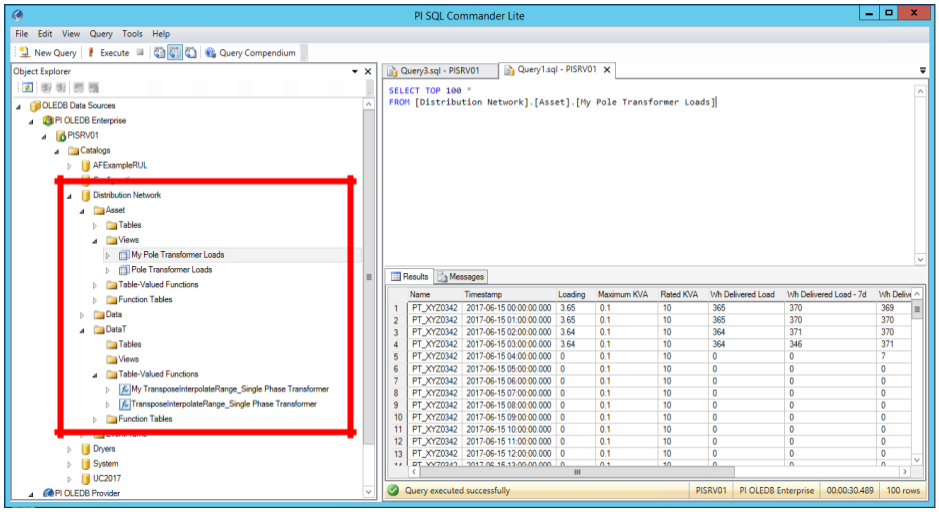
q. When finished with the edits, the query should look like the one below.

```
CREATE VIEW [Distribution Network].[Asset].[My Pole Transformer Loads]
AS
SELECT eh.Name, tir.Time Timestamp, tir.Loading, tir.[Maximum KVA], tir.[Rated KVA],
tir.[Wh Delivered Load], tir.[Wh Delivered Load - 7d], tir.[Wh Delivered Load - 114d],
tir.[Ambient Temperature], tir.[Relative Humidity], tir.[Wind Speed]

FROM [Distribution Network].[Asset].[ElementTemplate] et
INNER JOIN [Distribution Network].[Asset].[Element] e
    ON et.ID = e.ElementTemplateID
INNER JOIN [Distribution Network].[Asset].[ElementHierarchy] eh
    ON e.ID = eh.ElementID
CROSS APPLY [Distribution Network].DataT.[My TransposeInterpolateRange_Single Phase Transformer]
(
    eh.ElementID,
    DATE(N'06/15/17'),  /*StartTime*/
    DATE(N'08/31/17'),      /*EndTime*/
    N'1h'           /*TimeStep*/
) tir
WHERE et.Name = N'Single Phase Transformer'
OPTION (FORCE ORDER, IGNORE ERRORS, EMBED ERRORS)
```

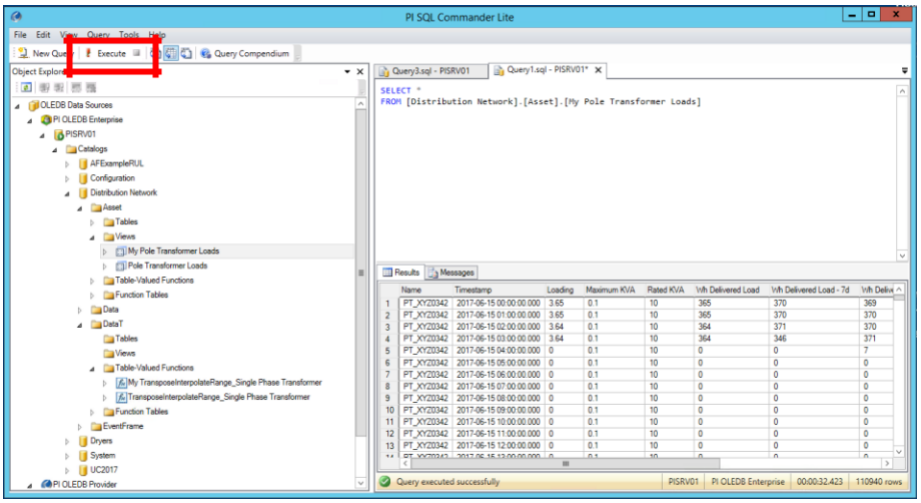
r. Either right-click on the query and select **Execute** or click on the **Execute** button on the top menu bar to run this query and to create the view. You should wee the following message in the Results grid and a green light in the bottom message bar of PI SQL Commander Lite.



s. Return to the Views folder in the Object Explorer hierarchy. You may need to right-click on the view folder and **Refresh** it to see your new view. Right-click on the view, **My Pole Transformer Loads**, and **Execute Predefined Query…**



t. Edit the query by deleting the “TOP 100” specification in the SELECT statement. Either right-click on the query and select **Execute** or click on the **Execute** button on the top menu bar to rerun the query. This time you should get all 110,940 rows.



Develop the Model in Python

The following code provides the same analysis as Example 2 except it accesses PI data using a PI OLEDB view. The Python "adodbapi" package is used to manage the connection.

> Tip
>
> **To run the code with your PI OLEDB View, you'll want to replace "Pole Transformer Loads" with "My Pole Transformer Loads" in the first cell of this Jupyter Notebook.**

a. Open Jupyter Notebooks from the windows Taskbar,  . Open the Python script, **Load Prediction - Example 2 (PI OLEDB Enterprise)**.

b. In the first cell, change the select statement to **'SELECT * FROM [Distribution Network].[Asset].[My Pole Transformer Loads]'.** This will cause Python to access your PI OLEDB Client view and create the dataframe, **poleTransformerLoads**. Select this cell and hit **Shift+Enter**. Wait until you see the table preview before moving to the next cell.

```
# EXAMPLE 2 - Create linear regression models for all 60 pole transformers.
# Read PI data via PI OLEDB Enterprise using the adodbapi package to establish an OLEDB connection.

# Import Python packages.
import adodbapi as ado                          # OLEDB support for accessing PI SQL Client - import AF/PI data.
import pandas as pd                             # Dataframe support.
from sklearn.linear_model import LinearRegression # Linear regression model from the scikit-learn package.

# Connect to "Distribution Network Lab" model in AF through the PI OLEDB Enterprise.
PI_connection = ado.connect("Provider=PIOLEDBEnt; Integrated Security=SSPI; Data Source=PISRV01;Command Timeout=0;")

# Create a cursor object to access the data serer for the "Distribution Network Lab" database in AF.
PI_cursor = PI_connection.cursor()

# Select the entire table using the view configured in PI SQL Commander.
PI_cursor.execute('SELECT * FROM [Distribution Network].[Asset].[My Pole Transformer Loads]')

# Extract first row, index=0, to get column names for use as dataframe headers
columnNames = [ x[0] for x in PI_cursor.description]
print("PI View column Names:\n\n", columnNames)

# Unpack the cursor rows into a datafraem.
poleTransformerLoads = pd.DataFrame([dict(zip(columnNames, row)) for row in PI_cursor], columns=columnNames)

# Take a look to see if everythking worked.
poleTransformerLoads.head()
```

```
PI View column Names:

 ['Name', 'Timestamp', 'Loading', 'Maximum KVA', 'Rated KVA', 'Wh Delivered Load', 'Wh Delivered Load - 7d', 'Wh Deli
vered Load - 14d', 'Ambient Temperature', 'Relative Humidity', 'Wind Speed']
```

	Name	Timestamp	Loading	Maximum KVA	Rated KVA	Wh Delivered Load	Wh Delivered Load - 7d	Wh Delivered Load - 14d	Ambient Temperature	Relative Humidity	Wind Speed
0	PT_XYZ0342	2017-06-15 00:00:00	3.65	0.1	10.0	365.0	370.0	369.0	68.0	93.0	7.0
1	PT_XYZ0342	2017-06-15 01:00:00	3.65	0.1	10.0	365.0	370.0	370.0	68.0	93.0	6.0
2	PT_XYZ0342	2017-06-15 02:00:00	3.64	0.1	10.0	364.0	371.0	370.0	68.0	93.0	6.0
3	PT_XYZ0342	2017-06-15 03:00:00	3.64	0.1	10.0	364.0	346.0	371.0	68.0	90.0	0.0
4	PT_XYZ0342	2017-06-15 04:00:00	0.00	0.1	10.0	0.0	0.0	7.0	67.0	93.0	0.0

c. Rename columns with shorter names. Create a smaller dataframe, **modellingData**, containing only the values need for predicting transformer loads. Select this cell and hit **Shift+Enter**.

```
# Rename some columns with shorter names to make them easier to work with.
poleTransformerLoads.rename(columns = {'Name' :'Transformer'}, inplace = True)
poleTransformerLoads.rename(columns = {'Wh Delivered Load':'Wh Load'}, inplace = True )
poleTransformerLoads.rename(columns = {'Wh Delivered Load - 14d':'Wh Load-14d'}, inplace = True )
poleTransformerLoads.rename(columns = {'Wh Delivered Load - 7d':'Wh Load-7d'}, inplace = True )
poleTransformerLoads.rename(columns = {'Ambient Temperature':'Temperature'}, inplace = True )
poleTransformerLoads.rename(columns = {'Relative Humidity':'Humidity'}, inplace = True )
poleTransformerLoads.rename(columns = {'Wind Speed':'Wind'}, inplace = True )

# Define second dataframe with just data needed for our modelling.
modellingData = poleTransformerLoads[['Transformer', 'Timestamp', 'Temperature', 'Humidity',
                    'Wind','Wh Load', 'Wh Load-7d', 'Wh Load-14d']]

# Peek at the first five rows to make sure things look right.
modellingData.head()
```

	Transformer	Timestamp	Temperature	Humidity	Wind	Wh Load	Wh Load-7d	Wh Load-14d
0	PT_XYZ0342	2017-06-15 00:00:00	68.0	93.0	7.0	365.0	370.0	369.0
1	PT_XYZ0342	2017-06-15 01:00:00	68.0	93.0	6.0	365.0	370.0	370.0
2	PT_XYZ0342	2017-06-15 02:00:00	68.0	93.0	6.0	364.0	371.0	370.0
3	PT_XYZ0342	2017-06-15 03:00:00	68.0	90.0	0.0	364.0	346.0	371.0
4	PT_XYZ0342	2017-06-15 04:00:00	67.0	93.0	0.0	0.0	0.0	7.0

d. Change the index of **modellingData** to be the transformer name. Select this cell and hit **Shift+Enter**.

```
# In order to analyze transformers individually, we need to set the datframe's index to the "Transformer" column.
modellingData = modellingData.set_index("Transformer", drop=False)

# Take a look, see the difference?
modellingData.head()
```

Transformer	Transformer	Timestamp	Temperature	Humidity	Wind	Wh Load	Wh Load-7d	Wh Load-14d
PT_XYZ0342	PT_XYZ0342	2017-06-15 00:00:00	68.0	93.0	7.0	365.0	370.0	369.0
PT_XYZ0342	PT_XYZ0342	2017-06-15 01:00:00	68.0	93.0	6.0	365.0	370.0	370.0
PT_XYZ0342	PT_XYZ0342	2017-06-15 02:00:00	68.0	93.0	6.0	364.0	371.0	370.0
PT_XYZ0342	PT_XYZ0342	2017-06-15 03:00:00	68.0	90.0	0.0	364.0	346.0	371.0
PT_XYZ0342	PT_XYZ0342	2017-06-15 04:00:00	67.0	93.0	0.0	0.0	0.0	7.0

e. Leverage the connection to PI OLEDB Enterprise and **My Pole Transformer Loads** view to get the list of transformer names. Select this cell and hit **Shift+Enter**.

```
# Use our existing PI View connectiont to select just the transformer names from the view configured in
# PI SQL Commander.
PI_cursor.execute('SELECT DISTINCT Name FROM [Distribution Network].[Asset].[Pole Transformer Loads]')

# Create a list for the transformer names.
transformerNames = []

# Populate the list
for row in PI_cursor:
    transformerNames.append(row[0])

# Show tranfomer names list.
print(transformerNames)
```

```
['PT_XYZ0342', 'PT_XYZ0343', 'PT_XYZ0344', 'PT_XYZ0345', 'PT_XYZ0346', 'PT_XYZ0347', 'PT_XYZ0348', 'PT_XYZ0349', 'PT_
XYZ0350', 'PT_XYZ0351', 'PT_XYZ0352', 'PT_XYZ0353', 'PT_XYZ0354', 'PT_XYZ0355', 'PT_XYZ0356', 'PT_XYZ0357', 'PT_XYZ03
58', 'PT_XYZ0376', 'PT_XYZ0377', 'PT_XYZ0378', 'PT_XYZ0379', 'PT_XYZ0380', 'PT_XYZ0381', 'PT_XYZ0382', 'PT_XYZ0383',
'PT_XYZ0384', 'PT_XYZ0385', 'PT_XYZ0387', 'PT_XYZ0388', 'PT_XYZ0389', 'PT_XYZ0390', 'PT_XYZ0391', 'PT_XYZ0392', 'PT_X
YZ0393', 'PT_XYZ0394', 'PT_XYZ0395', 'PT_XYZ0396', 'PT_XYZ0397', 'PT_XYZ0398', 'PT_XYZ0399', 'PT_XYZ0400', 'PT_XYZ040
1', 'PT_XYZ0402', 'PT_XYZ0403', 'PT_XYZ0404', 'PT_XYZ0405', 'PT_XYZ0406', 'PT_XYZ0407', 'PT_XYZ0408', 'PT_XYZ0409',
'PT_XYZ0410', 'PT_XYZ0411', 'PT_XYZ0412', 'PT_XYZ0413', 'PT_XYZ0414', 'PT_XYZ0415', 'PT_XYZ0416', 'PT_XYZ0418', 'PT_X
YZ0419', 'PT_XYZ0420']
```

f. Generate linear regression model coefficients for each of the 60 transformers and write them to SQL. Select this cell and hit **Shift+Enter**.

```
# Using the "adodbapi" package, to connect to the "Predictive Equations" MS SQL.

# Set connection parameters.
con_string = 'DRIVER={SQL Server};SERVER=PISRV01;DATABASE=PIWorld;Trusted_Connection=Yes;'

# Connect to "Distribution Network Lab" model in AF through the PI SQL Client.
SQL_connection = ado.connect(con_string)

# Create a cursor object to access the data serer for the "Distribution Network Lab" database in AF.
SQL_cursor = SQL_connection.cursor()

# Create linear regression object from the "sklearn" package we imported earlier.
LinReg = LinearRegression()

# Looping through the trasformer list, Perform linear regression on each transformer.
for transformer in transformerNames:

    # Create dataframe for one transformer.
    transformerData = modellingData.loc[transformer,:]

    # Perform linear regression fit
    LinReg.fit(transformerData[["Wh Load-7d","Wh Load-14d","Temperature","Humidity"]],transformerData["Wh Load"])

    # Update asset ID value with the name of this transformer.
    asset_id = "'"+transformer+"'"

    # Print equation.
    print(transformer, "Eq:\n", LinReg.coef_[0], "*","'Wh Delivered Load - 7d' + ",LinReg.coef_[1],
          "*","'Wh Delivered Load - 14d' + ", LinReg.coef_[2], "*", "'Ambient Temperature' +",
          LinReg.coef_[3], "*","'Relative Humidity' +(",LinReg.intercept_, ")" )

    ## INSERT and UPDATE queries to load table for the first time or update an existing one.
    # Construct query to add this transfomer's model coefficients.
    insert_query = f'INSERT [Predictive Equations] ([Asset ID], Coefficient_0, Coefficient_1, Coefficient_2, Coeffic

    update_query = f'UPDATE [Predictive Equations] SET Coefficient_0={LinReg.coef_[0]}, Coefficient_1={LinReg.coef_[

    # Insert this record into the "Predictive Equations" table.
    SQL_cursor.execute(update_query)

# Commit the queries to write the data into SQL.
SQL_connection.commit()

# Close database connections.
SQL_connection.close()
PI_connection.close()
```

```
PT_XYZ0407 Eq:
 0.22133353596401245 * 'Wh Delivered Load - 7d' +  0.18940221947528418 * 'Wh Delivered Load - 14d' +  303.2482562107521 * 'Am
bient Temperature' + 7.845159067779704 * 'Relative Humidity' +( -14852.0518428231 )
PT_XYZ0408 Eq:
 0.22918154817171893 * 'Wh Delivered Load - 7d' +  0.1723254292743076 * 'Wh Delivered Load - 14d' +  207.3195523807009 * 'Amb
ient Temperature' + 5.783909412754433 * 'Relative Humidity' +( -8273.493697771371 )
PT_XYZ0409 Eq:
 0.2706555850527234 * 'Wh Delivered Load - 7d' +  0.24808404285129917 * 'Wh Delivered Load - 14d' +  257.19175040719324 * 'Am
bient Temperature' + 37.67773251209764 * 'Relative Humidity' +( -15786.699575218732 )
PT_XYZ0410 Eq:
 0.28804834752338854 * 'Wh Delivered Load - 7d' +  0.23893391851491846 * 'Wh Delivered Load - 14d' +  472.11670533964144 * 'A
mbient Temperature' + 27.87650630785805 * 'Relative Humidity' +( -24316.385741388473 )
PT_XYZ0411 Eq:
 0.435869138307828 * 'Wh Delivered Load - 7d' +  0.41339607398963046 * 'Wh Delivered Load - 14d' +  0.3590444851840645 * 'Amb
ient Temperature' + 0.3196969465241239 * 'Relative Humidity' +( -34.91262860815391 )
PT_XYZ0412 Eq:
 0.2075887778891295 * 'Wh Delivered Load - 7d' +  0.20641741419115514 * 'Wh Delivered Load - 14d' +  321.2758186735902 * 'Amb
ient Temperature' + 36.830405127594084 * 'Relative Humidity' +( -17855.72844495021 )
```

Appendix B - Python Access to PI via PI Web API

Develop the Model in Python

The PI Web API is installed with the PI System. It provides programmatic access to the PI Archive and PI AF. The following script imports the same dataset used in Example 2 using the PI Web API.

> Tip
>
> **By default, there is a constraint placed on calls made to the PI System when using the PI Web API. Each call can return no more than 150,000 items. The type of items returned depends on the nature of the call. For example, your maximum items could consist of elements, attributes, or time series data values depending on how your call is defined.**
>
> **This restriction can be adjusted by adding or resetting the "MaxReturnedItemsPerCall" attribute value under the PI Web API branch of the Configuration database on your AF Server. More information can be found on page 3 of the PI Web API 2018 Users Guide.**

a. Import Python packages for supporting PI Web API session and managing message requests and unpacking json responses. Select this cell and hit **Shift+Enter**.

```
# EXAMPLE 2 - Create linear regression models for all 60 pole transformers.
# Read PI data via PI Web API.

# Import Python packages.
import adodbapi as ado                          # Support for accessing MS SQL to store model coefficients.
import pandas as pd                             # Dataframe support.

from sklearn.linear_model import LinearRegression # Linear regression model from the scikit-learn package.

# Import packages
import json, requests, urllib.parse, requests_kerberos # Support access to PI Web API.

# Setup Kerberos cnnection.
from requests_kerberos import HTTPKerberosAuth, OPTIONAL
clientAuth = HTTPKerberosAuth(force_preemptive=True, mutual_authentication=OPTIONAL, delegate=True)

# Disable warning messages.
requests.packages.urllib3.disable_warnings()

# Open session.
session = requests.session()
```

b. Make a client connection to PI via the PI Web API. Select this cell and hit **Shift+Enter**.

```
# Connect to PI System at highest level.
print('\nSending top level PI Web API Request\n')

# Base URL to PI System.
baseurl = "https://PISRV01/piwebapi/"

# Post URL to PI System and retrieve response.
response = session.get(baseurl, auth=clientAuth, verify=False,  timeout=30)

# Response status - did it work, i.e. 200 means success?
print('Response Received: {0} ' .format(response.status_code),'\n')

# Unpack items of interest using the "json" package.  Extracts dictionary of links.
Links = json.loads(response.text)['Links']

# Extracts base level URL's for PI Web API from the Links dictionary using the "json" package.
print('Self:', Links['Self'])
print('Asset Servers:', Links['AssetServers'])
print('Data Servers:', Links['DataServers'])
print('PI System:', Links['System'], '\n')
```

```
Sending top level PI Web API Request

Response Received: 200

Self: https://pisrv01/piwebapi/
Asset Servers: https://pisrv01/piwebapi/assetservers
Data Servers: https://pisrv01/piwebapi/dataservers
PI System: https://pisrv01/piwebapi/system
```

c. PI Web API Request for “WebId” of ***Colegio Científico*** circuit (parent) element.

```
# Get WebId for circuit element, "Colegio Clientifico".

# Root path to "circuit" level in AF heirarchy.
circuitPath = '\\\\pisrv01\\Distribution Network\\Alajuela\\Avenida Central\\Transformer 1\\Colegio Cientifico'

# Construct URL to the Get Elements controller of the PI Web API.
baseurl_elements = baseurl + '/elements'

# Set parameters list by specifying the AF path to the circuit's element.
parameters = {'path': circuitPath,}

# Format parameter list for URL.
url_parameters = urllib.parse.urlencode(parameters, quote_via=urllib.parse.quote)

# Show HTTPS request string.
print('PI Web API Request for WebId of Colegio Cientifico. Click it to see the response from PI.\n\n', \
      baseurl_elements + '?' + url_parameters, '\n')

# Post URL with path and paramters to PI System
response = session.get(baseurl_elements, auth=clientAuth, params=url_parameters, verify=False, timeout=30)

# Response status - did it work?
print('Response Received: {0} ' .format(response.status_code),'\n')

# Show element's Web_Id.
circuitWebId = json.loads(response.text)['WebId']
print("Colegio Clientifico element's WebId:\n", circuitWebId)
```

```
PI Web API Request for WebId of Colegio Cientifico. Click it to see the response from PI.

 https://PISRV01/piwebapi//elements?path=%5C%5Cpisrv01%5CDistribution%20Network%5CAlajuela%5CAvenida%20Central%5CTransformer%201%5CColegio%20Cient%C3%ADfico

Response Received: 200

Colegio Clientifico element's WebId:
 F1Em_BJ6yxOWfEu1XN8dxGow1g3a_SqpOs6BGBFQANOjYKWwUElTUlYwMVxESVNUUklCVVRJT04gTkVUV09SS1xBTEFKVUVMQVxBVkVOSURBIENFTlRSQUxcVFJBTlNGT1JNRVIgMVxDT0xFR0lPIENJRU5Uw41GSUNP
```

Tip	**For this example, we launched Jupyter Notebooks in Google Chrome. In Chrome, hyperlinks printed from Python are active and when clicked will open an adjacent browser window allowing you to see messages returned from the PI Server. This is a very handy feature for debugging your script.**

d. Clicking the hyperlink in the Python output shows the response message from the PI Web API.

e. PI Web API Request for “WebId”’s of X, Y and Z Phase (child) elements.

```
# Get names and WebId's for phase elements under the "Colegio Cientifico" circuit element.

# Define dictionary to store the names and WebId's for the three phase elements
phaseElements = {}

# Construct URL to all child elements of circuit "Colegio Cientifico" using it's WebId.
baseurl_elements = baseurl + '/elements/' + circuitWebId + '/elements'

# Request names and WebId's for all child elements derived from the "Phase" template.
parameters = {'templateName': 'Phase', 'selectedFields':'Items.Name;Items.WebId'}

# Format parameter list for URL.
url_parameters = urllib.parse.urlencode(parameters, quote_via=urllib.parse.quote)

# Show HTTPS request string.
print("PI Web API Request for WebId's of X, Y and Z Phase elements.", \
      "Click it to see the response from PI.\n\n", baseurl_elements + '?' + url_parameters, '\n')

# Post URL with path and paramters to PI System
response = session.get(baseurl_elements, auth=clientAuth, params=url_parameters, verify=False, timeout=30)

# Response status - did it work?
print('Response Received: {0} ' .format(response.status_code),'\n')

#  Loop throught he response to pupulate the phaseElements dictinary.
for item in json.loads(response.text)['Items']:

    # Store the strings.
    phaseElements.update({item['Name']:item['WebId']})

# Print the WebIds using the Name index.
print("X Phase's WebId:\n", phaseElements['X Phase'])
print("Y Phase's WebId:\n", phaseElements['Y Phase'])
print("Z Phase's WebId:\n", phaseElements['Z Phase'])
```

```
PI Web API Request for WebId's of X, Y and Z Phase elements. Click it to see the response from PI.

 https://PISRV01/piwebapi//elements/F1Em_BJ6yxOWfEu1XN8dxGow1g3a_SqpOs6BGBFQANOjYKWwUElTU1YwMVxESVNUUklCVVRJT04gTkVUV
09SS1xBTEFKVUVMQVxBVkVOSURBIENFTlRSQUxcVFJBTlNGT1JNRVIgMVxDT0xFR0lPIENJRU5Uw41GSUNP/elements?templateName=Phase&selec
tedFields=Items.Name%3BItems.WebId

Response Received: 200

X Phase's WebId:
 F1Em_BJ6yxOWfEu1XN8dxGow1g46_SqpOs6BGBFQANOjYKWwUElTU1YwMVxESVNUUklCVVRJT04gTkVUV09SS1xBTEFKVUVMQVxBVkVOSURBIENFTlRS
QUxcVFJBTlNGT1JNRVIgMVxDT0xFR0lPIENJRU5Uw41GSUNPXFggUEhBU0U
Y Phase's WebId:
 F1Em_BJ6yxOWfEu1XN8dxGow1gehQfu5Os6BGBFQANOjYKWwUElTU1YwMVxESVNUUklCVVRJT04gTkVUV09SS1xBTEFKVUVMQVxBVkVOSURBIENFTlRS
QUxcVFJBTlNGT1JNRVIgMVxDT0xFR0lPIENJRU5Uw41GSUNPXFkgUEhBU0U
Z Phase's WebId:
 F1Em_BJ6yxOWfEu1XN8dxGow1gBBkfu5Os6BGBFQANOjYKWwUElTU1YwMVxESVNUUklCVVRJT04gTkVUV09SS1xBTEFKVUVMQVxBVkVOSURBIENFTlRS
QUxcVFJBTlNGT1JNRVIgMVxDT0xFR0lPIENJRU5Uw41GSUNPXFogUEhBU0U
```

f. Define an empty dataframe named **modellingData** and list name **allTransformers** to write the transformer data and names into. Select this cell and hit **Shift+Enter**.

```
# Create dataframe of interpolated data for each transformer.

# Create empty pandas dataframe for transformer data.
columns =['Transformer', 'TimeStamp', 'Temperature', 'Humidity','Wind','Wh Load', 'Wh Load-14d', 'Wh Load-7d']
modellingData = pd.DataFrame(columns=columns)

# As we loop through heirarchy to get data, establish a list of all transformers.
allTransformers = []
```

g. We will step through each Phase, X, Y and Z. For each phase, we will loop through each transformer, appending its attribute values to the dataset. Select this cell and hit **Shift+Enter**.

```
# Fill dataframe by stepping through the phase level and appending each transformer's attribute values.

# Print URL as example on the first time through the loop.
printURL = True

# Step through phases.
for phaseName, phaseWebId in phaseElements.items():

    # Constructed URL to reference each phase element.
    baseurl_referencedelements = baseurl + '/elements' + '/' + phaseWebId + '/referencedelements'

    # Request for all child elements derived from the "Single Phase Transfrmer" template.
    parameters = {'templateName': 'Single Phase Transformer','selectedFields':'Items.Name;Items.WebId'}

    # Format parameter list for URL.
    url_parameters = urllib.parse.urlencode(parameters, quote_via=urllib.parse.quote)

    # Show HTTPS request string
    if printURL:
        print("Sample PI Web API request for Name and WebId's of transformers under",\
            phaseName, ' Click it to see the response from PI.\n\n', \
            baseurl_referencedelements + '?' + url_parameters, '\n')

    # Post URL with path and paramters to PI System
    response = session.get(baseurl_referencedelements, auth=clientAuth, params=url_parameters, verify=False,  timeou

    # Response status - did it work?
    print('Response Received: {0}' .format(response.status_code),"Transformer Name and WebID's for", phaseName, '\n

    # Define\empty dictionary to hold transformer names and WebId's.
    transformerElements = {}

    # Loop through list and put names and WebId's into dictinary.
    for item in json.loads(response.text)['Items']:

        # Store the strings.
        transformerElements.update({item['Name']:item['WebId']})
```

```
    # Loop through phases.
    for transformerName, transformerWebId in transformerElements.items():

        # Add this transformer to the list.
        allTransformers.append(transformerName)

        # Construct URL for getting selected attributes.
        baseurl_multipleattributes = baseurl + '/attributes/multiple'

        # Request WebId's for specified transformer attributes.
        parameters = [('Path', circuitPath + '\\' + phaseName + '\\' + transformerName + '|Ambient Temperature'), \
                      ('Path', circuitPath + '\\' + phaseName + '\\' + transformerName + '|Relative Humidity'), \
                      ('Path', circuitPath + '\\' + phaseName + '\\' + transformerName + '|Wh Delivered Load'), \
                      ('Path', circuitPath + '\\' + phaseName + '\\' + transformerName + '|Wh Delivered Load - 14d'),
                      ('Path', circuitPath + '\\' + phaseName + '\\' + transformerName + '|Wh Delivered Load - 7d'),
                      ('Path', circuitPath + '\\' + phaseName + '\\' + transformerName + '|Wind Speed'), \
                      ('SelectedFields','Items.Object.Name;Items.Object.WebId')]

        # Format parameter list for URL.
        url_parameters = urllib.parse.urlencode(parameters, quote_via=urllib.parse.quote)

        # Show HTTPS request string
        if printURL:
            print("Sample PI Web API request for Name an WebId's of", transformerName, "'s selected attributes.",\
                ' Click it to see the response from PI.\n\n', baseurl_multipleattributes + '?' + url_parameters, '\n

        # Post URL with path and paramters to PI System.
        response = session.get(baseurl_multipleattributes, auth=clientAuth, params=url_parameters, verify=False,  t

        # Response status - did it work?
        print('     Response Received: {0}' .format(response.status_code),"Attribute Name and WebId's for", \
              transformerName, '\n')
```

```
        # Define list for attribute Webid's.
        attributeWebId = []

        # Loop through unnamed list and extract WebId and Name into lists.
        for attributes in json.loads(response.text)['Items']:

            attribute = attributes['Object']
            attributeWebId.append(attribute['WebId'])

        # Construct URL for getting interpoated data for attributes.
        baseurl_streamsets = baseurl + '/streamsets/interpolated'

        # Request for start and end times, and interval.
        parameters = [('WebId', attributeWebId[0]),('WebId', attributeWebId[1]),\
                      ('WebId', attributeWebId[2]),('WebId', attributeWebId[3]),\
                      ('WebId', attributeWebId[4]),('WebId', attributeWebId[5]),\
                      ('startTime','15-jun-2017'),('endTime','31-aug-2017'),('interval','1h'),\
                      ('selectedFields', 'Items.Name;Items.Items.Timestamp;Items.Items.Value')]

        # Format parameter list for URL.
        url_parameters = urllib.parse.urlencode(parameters, quote_via=urllib.parse.quote)

        # Show HTTPS request string
        if printURL:
            print("Sample PI Web API request for Timestamp and Values for transformer's selected attributes.",\
                ' Click it to see the response from PI.\n\n', baseurl_streamsets + '?' + url_parameters, '\n')

        # Post URL with path and paramters to PI System
        response = session.get(baseurl_streamsets, auth=clientAuth, params=url_parameters, verify=False,  timeout=3

        # Response status - did it work?
        print('     Response Received: {0}' .format(response.status_code), "Attribute Timestamp and Values for", \
              transformerName, '\n')
```

```
        # Initialize lists to store unpacked data.
        transformer = []; timestamp = []; temperature = []; humidity= []; wind = []
        load = []; load_14d = []; load_7d = []

        # Loop through response and create pandas dataframe for this transformer.
        # For each attribute.
        for attribute in json.loads(response.text)['Items']:

            # For each teimstamp\value.
            for value in attribute['Items']:

                # Store values for "Ambient Temperature" attribute into its list.
                if attribute['Name'] == 'Ambient Temperature':
                    temperature.append(value['Value'])

                    # Fill timestamp and transformer name lists here, only need to do this once.
                    timestamp.append(value['Timestamp'])
                    transformer.append(transformerName)

                # Store values for "Relative Humidity" attribute into its list.
                if attribute['Name'] == 'Relative Humidity':
                    humidity.append(value['Value'])

                # Store values for "Wind Speed" attribute into its list.
                if attribute['Name'] == 'Wind Speed':
                    wind.append(value['Value'])

                # Store values for "Wh Delivered Load" attribute into its list.
                if attribute['Name'] == 'Wh Delivered Load':
                    load.append(value['Value'])

                # Store values for "Wh Delivered Load - 14d" attribute into its list.
                if attribute['Name'] == 'Wh Delivered Load - 14d':
                    load_14d.append(value['Value'])

                # Store values for "Wh Delivered Load - 7d" attribute into its list.
                if attribute['Name'] == 'Wh Delivered Load - 7d':
                    load_7d.append(value['Value'])

        # Put lists of data for this transformer into a temporary dataframe
        thisTransformerData = pd.DataFrame({'Transformer':transformer, 'TimeStamp':timestamp, 'Temperature':tempera
                                            'Humidity':humidity, 'Wind':wind, 'Wh Load':load, 'Wh Load-14d':load_14
                                            'Wh Load-7d':load_7d})
```

```
        # Add this transformer's data to the others.
        modellingData = pd.concat([modellingData, thisTransformerData],ignore_index=True)

        # Turn off URL printing.
        printURL = False

# Print end of dataframe to see if we got all 110,940 records(row index of 110,939).
modellingData.tail(5)
```

h. Scroll down the results window to see the phase steps and transformer loops used to build the dtataframe. At the very bottom inspect the last five records to confirm all 110, 939 have been extracted.

	Humidity	Temperature	TimeStamp	Transformer	Wh Load	Wh Load-14d	Wh Load-7d	Wind
110935	73.0	68.0	2017-08-31T03:00:00Z	PT_XYZ0414	69.0	44.0	68.0	7.0
110936	81.0	64.0	2017-08-31T04:00:00Z	PT_XYZ0414	68.0	45.0	68.0	3.0
110937	75.0	66.0	2017-08-31T05:00:00Z	PT_XYZ0414	68.0	45.0	69.0	3.0
110938	78.0	65.0	2017-08-31T06:00:00Z	PT_XYZ0414	67.0	46.0	68.0	3.0
110939	86.0	62.0	2017-08-31T07:00:00Z	PT_XYZ0414	68.0	44.0	69.0	3.0

i. Generate linear regression model coefficients for each transformer and write them to SQL. Select this cell and hit **Shift+Enter**.

```
# In order to analyze transformers individually, we need to set the datframe's index to the "Transformer" column.
modellingData = modellingData.set_index("Transformer", drop=False)

# Set connection parameters.
con_string = 'DRIVER={SQL Server};SERVER=PISRV01;DATABASE=PIWORLD;Trusted_Connection=Yes;'

# Connect to "Distribution Network Lab" model in AF through the PI SQL Client.
SQL_connection = ado.connect(con_string)

# Create a cursor object to access the data serer for the "Distribution Network Lab" database in AF.
SQL_cursor = SQL_connection.cursor()

# Create linear regression object from the "sklearn" package we imported earlier.
LinReg = LinearRegression()

# Looping through the trasformer list, Perform linear regression on each transformer.
for transformer in allTransformers:

    # Create dataframe for one transformer.
    transformerData = modellingData.loc[transformer,:]

    # Perform linear regression fit
    LinReg.fit(transformerData[["Wh Load-7d","Wh Load-14d","Temperature","Humidity"]],transformerData["Wh Load"])

    # Update asset ID value with the name of this transformer.
    asset_id = "'"+transformer+"'"

    # Print equation.
    print(transformer, "Eq:\n", LinReg.coef_[0], "*","'Wh Delivered Load - 7d' + ",LinReg.coef_[1], "*","'Wh Deliver
    LinReg.coef_[2], "*", "'Ambient Temperature' +", LinReg.coef_[3], "*","'Relative Humidity' +(",LinReg.intercept_

    ## INSERT and UPDATE queries to load table for the first time or update an existing one.
    # Construct query to add this transfomer's model coefficients.
    insert_query = f'INSERT [Predictive Equations] ([Asset ID], Coefficient_0, Coefficient_1, Coefficient_2, Coeffic

    update_query = f'UPDATE [Predictive Equations] SET Coefficient_0={LinReg.coef_[0]}, Coefficient_1={LinReg.coef_[

    # Insert this record into the "Predictive Equations" table.
    SQL_cursor.execute(update_query)

# Commit the queries to write the data into SQL.
SQL_connection.commit()

# Close database connections.
SQL_connection.close()
```

Appendix C – Learning Resources

PI Resources

PI Integrator for Business Analytics (YouTube Playlist)

Exposing PI Data with the PI SQL Framework (Online Course)

Programming in PI Web API (Online Course)

Jupyter Resources

Jupyter Notebook for Beginners: A Tutorial

Jupyter Notebook Tutorial: The Definitive Guide

Jupyter Notebook Cheat Sheet

Jupyter Notebook Keyboard Shortcuts

28 Jupyter Notebook tips, tricks, and shortcuts

Machine Learning Resources

The table below contains a collection of learning resources I have experience with **Course | Topic:** Abbreviated title and hyperlink of learning resource **Type:** Course or Book
Language: Programming language used in the course
Platform: Course provider
Complexity: (1 introductory) → (5 bleeding-edge)
Curve: Learning-curve or approachability of the material (1 requires strong background knowledge) → (5 easy to follow given the complexity)
Theoretical: Theoretical coverage of algorithm details (1 few details) → (5 in-depth explanations)
Practical: (1 difficult to apply to project) → (5 readily applicable to real-world scenarios)

Course | Topic	Type	Language	Platform	Complexity	Curve	Theoretical	Practical
Data Science Bootcamp	Course	Python	Udemy	2	5	2	4
The Analytics Edge	Course	R	edX	3	5	4	4
Statistical Learning in R	Book	R	Book	3	5	5	5
Machine Learning	Course	Matlab	Coursera	3	4	5	4
TensorFlow Deep Learning	Course	Python	Udemy	3	4	2	3
Neural Networks for Machine Learning	Course	Python	Coursera	3	3	5	3

Hands-On Machine Learning	Book	Python	Book	4	5	4	5
Deep Learning Specialization	Course	Python	Coursera	4	4	5	4
Machine Learning for Coders	Course	Python	Fast.AI	4	4	4	4
Practical Deep Learning For Coders	Course	Python	Fast.AI	4	4	3	5
Deep Learning with Python	Book	Python	Book	4	4	4	5
Deep Learning	Book	Python	Book	4	2	5	1
Cutting Edge Deep Learning For Coders	Course	Python	Fast.AI	5	4	3	5
Time Series Analysis	Course	R	edX	Unknown			
Big Data Analytics Using Spark	Course	Python	edX	Unknown			

Join Kaggle! A vibrant machine learning community with over 13,000 datasets to practice on and learn from others. Kaggle has many community-made tutorials for beginners, competitions with rewards (such as the $1.5 million Homeland Security Passenger Screening Challenge), and many examples of bleeding-edge machine learning algorithms.


Have an idea how to improve our products?
OSIsoft wants to hear from you!
https://feedback.osisoft.com/

Save the Date!

OSIsoft PI World Users Conference in Gothenburg, Sweden. September 16-19, 2019.

Register your interest now to receive updates and notification early bird registration opening.

https://pages.osisoft.com/UC-EMEA-Q3-19-PIWorldGBG-RegisterYourInterest_RegisterYourInterest-LP.html?_ga=2.20661553.86037572.1539782043-591736536.1533567354