OSIsoft Cloud Services Course

Enabling Enterprise-Wide Data Science

OSIsoft, LLC
1600 Alvarado Street
San Leandro, CA 94577 USA
Tel: (01) 510-297-5800
Web: http://www.osisoft.com

Published: August 31, 2020

Table of Contents

1. Introduction

1.1 Learning Objectives and Problem Statement

 Video	*Before reading this section, please refer to the following course YouTube video: https://youtu.be/DhR8S90X4nw*

OSIsoft Cloud Services (OCS) is a cloud-native platform built for historical, real-time and future/forecasted operational data. OCS complements existing on-premise PI systems and enables users to easily define, visualize, query and shape data sets required for data science. This course will show users how OCS extends PI infrastructure to support enterprise-wide advanced analytics and machine learning.

In this course, we will be predicting the performance of a wind farm's ability to generate power. We will use a predictive machine learning model to forecast power generation. This capability is essential for every wind operator as having a reliable forecast helps them manage the portfolio of power generation resources in their fleet in order to meet the forecasted demand.

We have collected performance data from 10 wind turbines. Our first step is to evaluate the turbine data stored in PI via PI System Explorer (PSE). We will then stream the necessary data to an OCS namespace via the PI to OCS Agent from the PI Data Archive.

Next, we will then view the newly streamed data in OCS, add metadata in order to contextualize the streams, trend the data, as well as create a Data View.

We will then connect to and visualize this dataset in a Jupyter notebook using Python. After cleansing and filtering our dataset, we then use this filtered data to train, test and evaluate a model using a predictive machine learning algorithm.

The final step is to operationalize our predictive model by testing it against historical and forecasted weather data, and finally sending back the forecasted values to OCS.

1.2 PI System software

The VM (virtual machine) used for this course has the following PI System software installed:

Software	Version
PI Data Archive	2018 SP3
PI Asset Framework (PI AF) server	2018 SP3
PI Asset Framework (PI AF) client (PI System Explorer)	2018 SP3
PI Analysis & PI Notifications Services	2018 SP3
PI Vision	2019

For details on PI System software, please see http://www.osisoft.com/pi-system/pi-capabilities/product-list/

2. Analyzing & Configuring PI System Data in OSIsoft Cloud Services (OCS)

2.1 Objective

The objective of this section is to review the Wind Farm Asset Framework (AF) Model in PI System Explorer, then log in to OCS to view the streamed data and explore all of OCS' features/functionalities. Finally, an OCS Data View and OCS Client are created, in order to access this wind farm dataset programmatically from an external application.

2.2 Tasks

- Explore the Wind Farm Asset Framework Data in PI System Explorer
- Log in to OCS and review OCS features/functionalities
- Analyze Wind Farm data streams in the OCS Sequential Data Store
- Visualize Wind Farm data streams in the OCS Trend page
- Add & manage metadata for Wind Farm data streams in OCS
- Configure an OCS Data View from Wind Farm data streams & metadata
- Create an OCS Client for access to OCS from external applications

2.3 Approach & Details

 Video	*Before reading this section, please refer to the following course YouTube video: https://youtu.be/PO6yNFfOltM*

2.3.1 Wind Turbine Overview

Wind has kinetic energy that can be converted into electrical energy through a wind turbine. These turbines operate by capturing wind energy to turn rotor blades that run a generator. The rotors and turbines are governed by a rule known as Betz's Law, which states that a turbine may only capture 59.3% of the energy from the wind. Betz's limit refers to the fact that if more than this amount of wind energy is captured, a "build-up" in front of the rotor creates a limiting factor for future wind capture. This fact is relevant, as we will see that after a certain limit, higher wind speeds do not translate to additional power generated.

Wind turbines are comprised of several main parts, which include the foundation, tower, nacelle, rotor, and transformer. The foundation is the base of the turbine that keeps the structure secured to the ground. The tower is the largest and most visual aspect of the structure. The tower size will vary based on the size of the turbine, required height to achieve optimal wind input, and size of the rotor. The nacelle is an important structure that connects the tower to the generator and eventually the rotor. The rotor is comprised of blades that ultimately capture the wind and run the turbine. Lastly, the transformer stores the electricity that is produced. A controller inside the base of the tower ensures all aspects of the turbine are running properly.

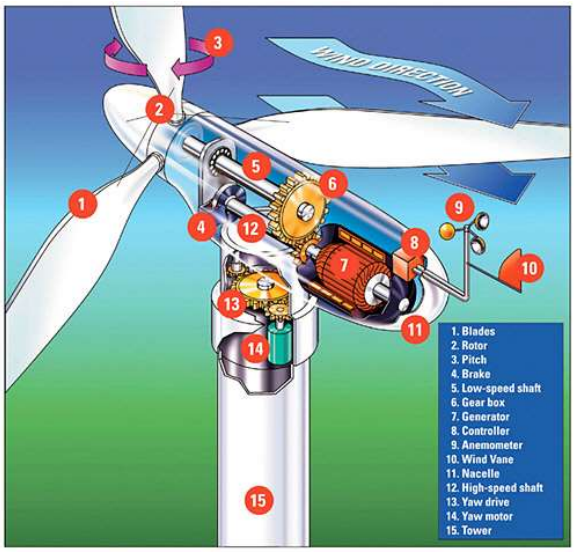


Figure 1: Wind Turbine Cross Section

Most of the inner workings of the turbine are located inside of the nacelle. The rotor is attached to the nacelle by way of the main shaft. As the wind turns the rotor, it turns the large main shaft, which is attached to a gearbox. The gearbox is used to turn the low rotations per minute (RPMs) of the rotor into a much higher number of rotations to generate more electricity. This ratio will vary depending on the size of the turbine. As an example, the gearbox in an industrial sized turbine could turn 22 RPMs into around 1500 revolutions.

The next essential part connected to the gear box is the yaw bearing. This is a bearing mounted right at the top of the tower. A large yaw wheel fits into the bearing and turns the nacelle and rotor in the direction of the wind when the yaw motor is engaged. A large generator is then attached to the gearbox. The current from the generator is sent down through the tower and into the transformer through large electrical cables. In addition to the large controller at the base of the tower, there is a smaller controller

in the nacelle that will allow the rotor to start when the anemometer reads that there is enough wind. The anemometer is a small wind meter that will read the wind speed. The wind vane attached to the top of the anemometer will read out the dominant direction, causing the yaw motor and wheel to turn accordingly. Lastly in the nacelle, there is a mechanical brake that will allow the rotor to be stopped if mechanical repairs are necessary. For more information and references, please refer to Appendix D.

A typical wind turbine power curve is shown in Figure 1, which plots Active Power (in kW) vs. Wind Speed (in meters/second):

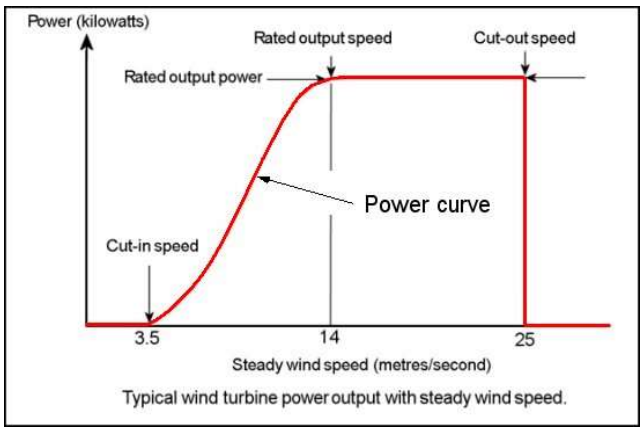


Figure 2: Typical Wind Turbine Active Power vs. Wind Speed

The Cut-in Speed is the wind speed value which causes the active power to begin rising, while the Cut-out Speed is the wind speed which causes the active power to begin dropping. The Rated Speed is the wind speed which causes the active power to plateau, while the Rated Power is the corresponding active power value at the rated speed. This graph is true for a fixed air temperature.

Changes in air temperature will cause this power curve to shift, as per Figure 2 below:

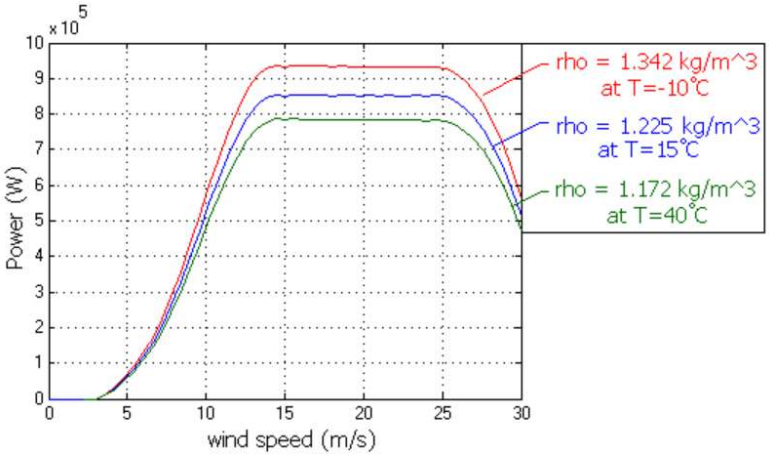


Figure 3: Effect of Air Temperature on Typical Wind Turbine Power Curve

Air temperature variations shift the power curve up at lower temperatures and shift it down at higher temperatures. Rho is the air density, which is affected by air temperature.

The wind turbine power curve will become important in Section 3, when we prepare our wind farm dataset for machine learning, in order to predict active power from a given wind speed and air temperature.

2.3.2 Review the Wind Farm Asset Framework Model in PI System Explorer

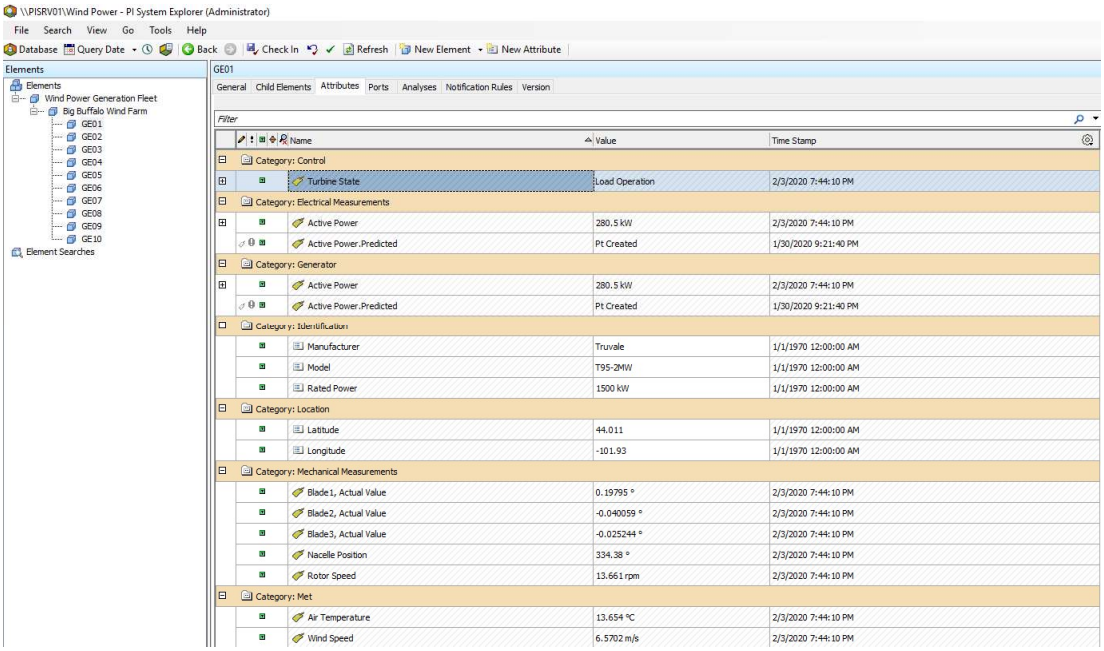
Video	*Before reading this section, please refer to the following course YouTube video: https://youtu.be/9kW4ihPqFxU*

The class will use a sample wind farm database named Wind Power which consists of a fleet level element (Wind Power Generation Fleet) with a single wind farm (Big Buffalo Wind Farm) in Amarillo, TX that has 10 turbines. Open PI System Explorer (PSE) and navigate to the Wind Power AF database.

Each of the turbines is built from the same AF element template, Wind Turbine, which can be viewed in the Library under Element Templates.

Each turbine has a variety of data associated with it, including PI Point real-time values, as well as metadata such as location and manufacturer information.

We will not focus on all the AF attributes, but will be primarily concerned with Active Power, Wind Speed, Air Temperature and Turbine State data for this course.



The AF attributes of interest and their corresponding PI tags are shown below (***XX*** refers to the Wind Turbine number, e.g. 01 for GE01, 10 for GE10)

i. Active Power : \\PISRV01\GE***XX***_P.ACT_PV
ii. Wind Speed : \\PISRV01\GE***XX***_V.WIN_PV
iii. Air Temperature : \\PISRV01\GE***XX***_T.GEN.COOL_PV
iv. Turbine State : \\PISRV01\GE***XX***_OS_PV

After exploring the data for this wind farm in PSE, we want to send these tags as separate data streams to OCS next.

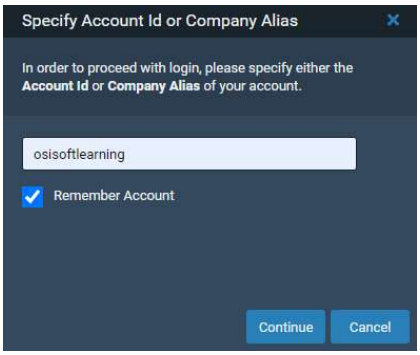
Video	*Before reading the next section, please refer to the following course YouTube video: https://youtu.be/IsQhYP7t3ho*

2.3.3 Analyzing Wind Farm Data Streamed in OCS

Video	*Before reading this section, please refer to the following course YouTube video: https://youtu.be/-r-osfjRdIA*

2.3.3.1 Logging in to OCS

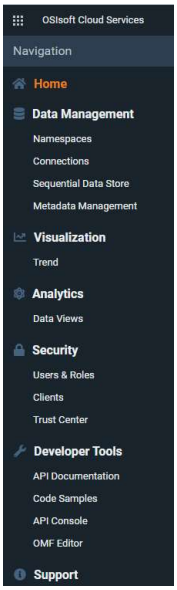
In a browser, navigate to cloud.osisoft.com. Click on the Sign In button at the top right corner, and enter osisoftlearning as the Account ID/Company Alias, check the Remember Account box, and click on Continue:



On the login page, please enter the email address and password that was given to OSIsoft for the OCS invitation receipt and click on the Sign In button.

OCS is configured to use Azure Active Directory (AAD), Microsoft Accounts (MSA) and Google as supported Identity Providers.

Once logged in to OSISoft Learning account name, click on the dotted square icon at the top left of the screen to bring up the Navigation menu in the OCS Portal.



We will begin with a brief tour of the OCS Portal, specifically the Data Management section, in order to familiarize ourselves with the features and functionalities available to subscribers.

The wind farm data streamed to OCS is housed in a *Namespace* which has a *Connection* to the PI System, and the individual tags are stored as data streams in the *Sequential Data Store* (SDS) under that Namespace.

2.3.3.2 Exploring Data Management Section in OCS

Introduction to Namespaces

The Namespace we will be using for this course is ***Enabling Enterprise-Wide Data Science,*** which is available from the list.

An OCS account (e.g. osisoftlearning) is divided into one or more namespaces, which correspond to a specific collection of infrastructure data. Each namespace corresponds to an instance of SDS, and holds its own set of SDS Types, SDS Streams, and SDS Stream Views. Namespaces serve as the destination for incoming stream data.

In the Navigation menu under Data Management, click on Namespaces**.** The Namespace screen is where a user can add, edit, remove and manage permissions for a namespace.

To view namespace details, select an existing namespace, and click Display Details. The type count, and stream count refer to the number of SDS types and SDS streams, respectively, that have been created for the selected namespace.

Introduction to Connections

The specific connection that streams the wind farm data from the PI System to OCS is ***Wind Turbine Data***, which is already created in the ***Enabling Enterprise-Wide Data Science*** namespace. This connection was done via the PI to OCS Agent, which has already been configured and running on a Master Training Cloud Environment (TCE) machine to stream this data to OCS. For

information regarding setting up this PI to OCS Agent to stream data to OCS, please refer to Appendix A of this manual.

You can establish high-throughput data connections from a PI system or any OMF-compatible data source into OCS with the Connections feature.

Introduction to Sequential Data Store (SDS)

SDS is a cloud-based streaming data storage that is optimized for storing sequential data, usually time-series, but can be anything that is indexed by an ordered sequence. SDS stores these streams of events with a specific type and provides convenient ways to find and associate events.

There are 3 elements of SDS, which are SDS Types, SDS Streams and SDS Stream Views. SDS Types and SDS Streams will be presented in more detail below. SDS Stream Views will not be discussed in this course but there are OSIsoft YouTube videos which explain their definition and configuration:

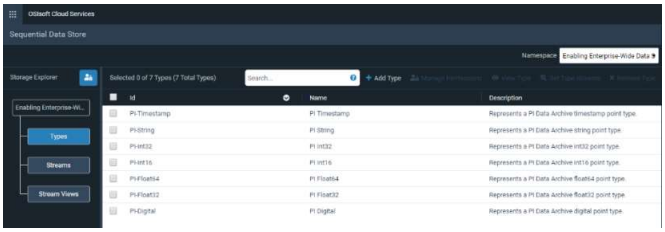
1. What is a Stream View? - https://www.youtube.com/watch?v=8iTgWyvc7eQ
2. Create a Stream View: https://www.youtube.com/watch?v=jhLqmLN0rQE

SDS Types

An SDS Type defines the shape of a single measured event and gives structure to data. SDS Types can define simple atomic types, such as integers, floats, strings, arrays, and dictionaries, or they can define complex types, which can be collections of simple types. In this course, we will create a complex SDS Type to store our forecasted data back in OCS later.

Note	*You cannot edit an existing type. In order to have a type with different properties than originally assigned, you must create a new type.*

The pre-configured PI to OCS Agent creates and uses seven (7) SDS Types, which are PI-Timestamp, PI-String, PI-Int32, PI-Int16, PI-Float64, PI-Float32 and PI-Digital.

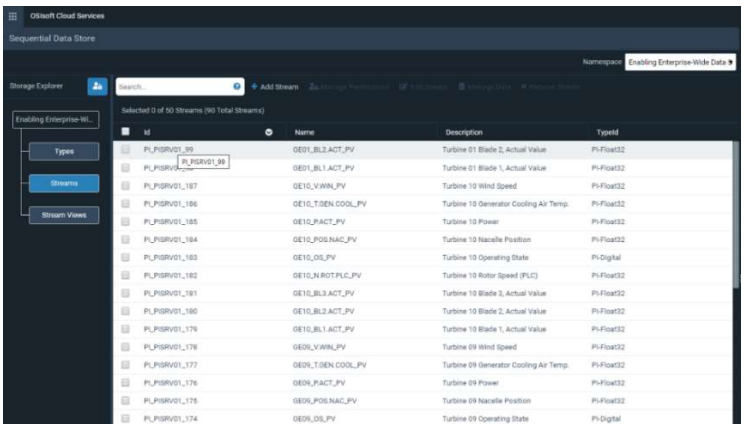


SDS Streams

Streams are collections of sequentially occurring values indexed by a single property, typically time series data. You define streams to organize incoming data from another system into OCS. To define

a stream, you must first define an SDS type, which defines the structure of the data you want to stream into a selected namespace. SDS Stream identifiers must be unique within a Namespace and must include a TypeId that references this SDS Type.

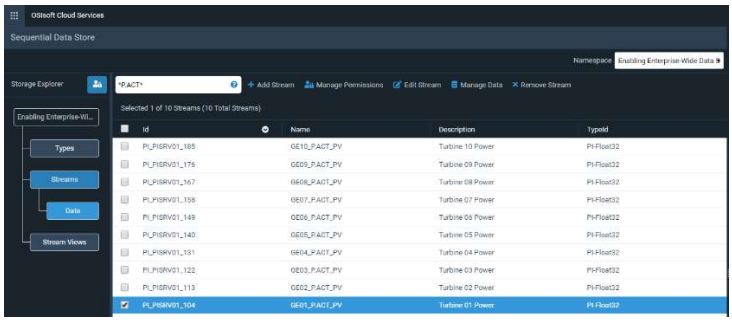
You create and write data to streams using a REST (*REpresentational State Transfer*) API (*Application Programming Interface*) or OMF (OSIsoft Message Format). The streams created can be used to store simple or complex types. In this course, we will create a new data stream and send values using the OCS Python library functions later.



We would like to now confirm and verify that all relevant data/events for the wind farm have been sent from the PI System and are present in the SDS, as well as perform some preliminary data exploration and analysis.

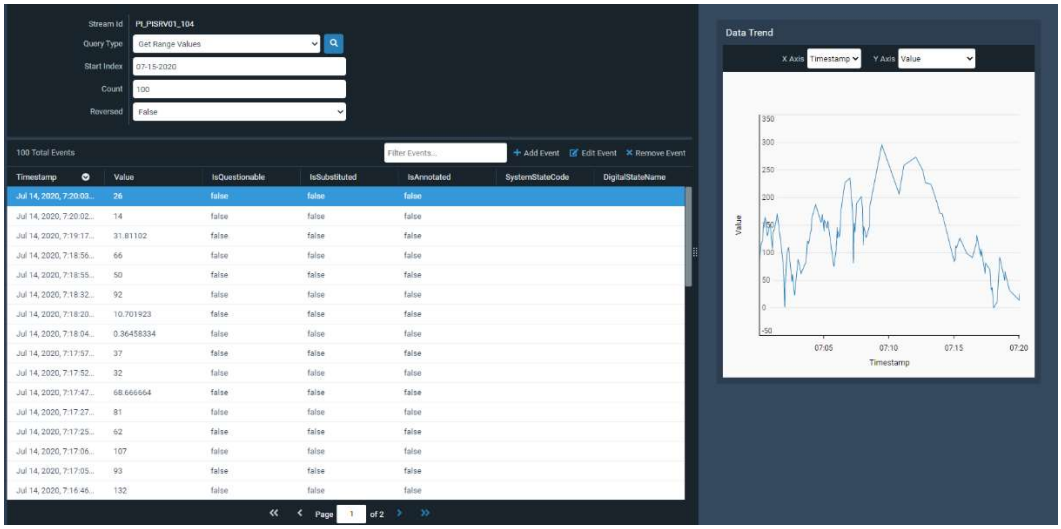
On this page, we can search for all the Active Power streams. On the Search field on the Streams tab, enter *P.ACT*, which would result in only the Active Power data streams for all the wind farm turbines being displayed.

If one of the data streams is selected such as GE01_P.ACT_PV, a Data tab appears under the Streams tab



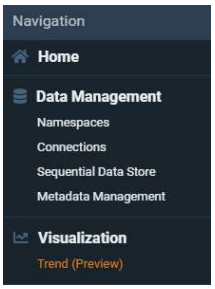
Under the Data tab, set the Query type to Get Range Values. The Get Range Values query type is similar to viewing data via PI System Management Tools under Data>Archive Editor. If the Get First Value or Get Last Value query types are selected, this would display a single data point at either the beginning or the end of the data timeframe, respectively.

Enter a Start Index of 07-15-2020 and click on the magnifying glass icon. The results for the Active Power stream for Wind Turbine GE01 will be shown in tabular form with a 100-event count (modifiable via the Count field) with a trend shown on the Details tab on the right.

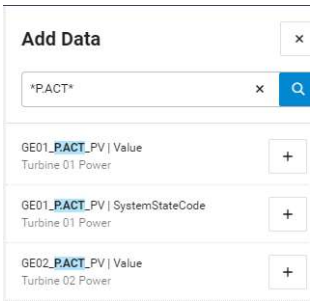


2.3.3.3 Visualizing Wind Farm Data in OCS

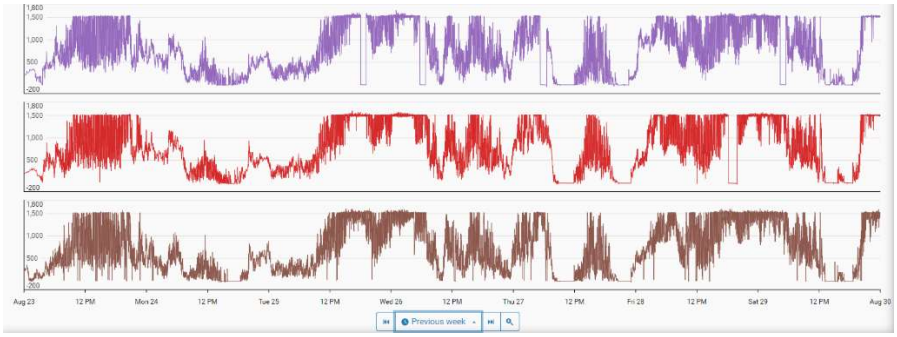
We can delve deeper into the visualization and trending of the wind farm data in the Visualization section of OCS. On the Trend page under Navigation, data streams in OCS can be trended, analyzed and overlaid for various time ranges of interest.



In this section, the Active Power for the wind turbines for the last week will be analyzed. In the Search query bar, enter *P.ACT*, and click the magnifying glass icon. This will return the Active Power for all the 10 turbines. Add the P.ACT_PV values for the first 3 turbines (GE01, GE02 & GE03) to the trend by clicking the + button to the right of the corresponding data stream.

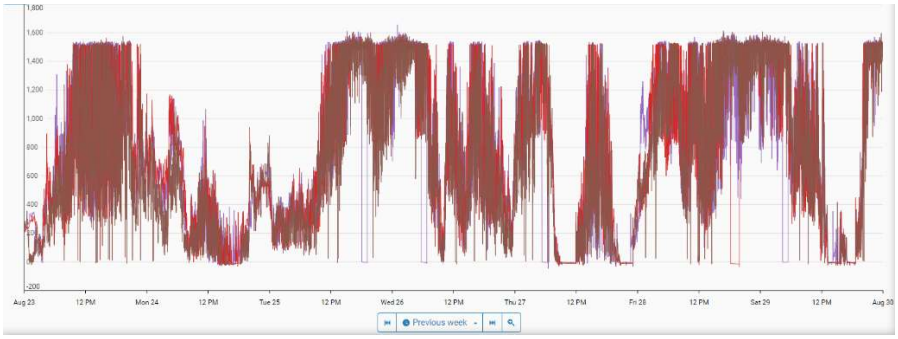


In the Time Range field, select Previous Week from Quick Ranges. The corresponding trend should be as below:

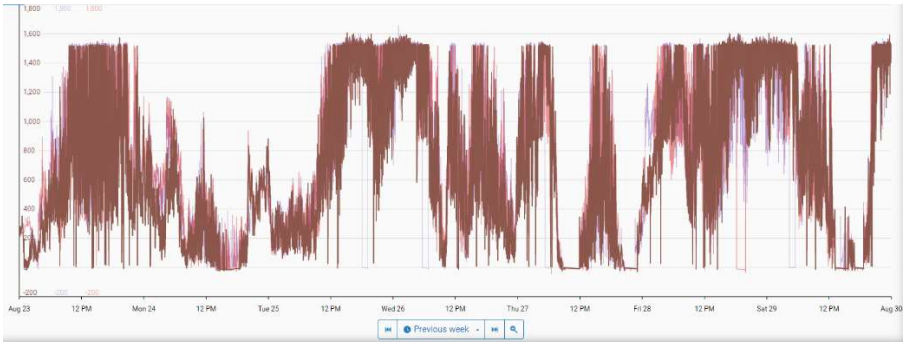


Each Active Power data stream is trended as a separate graph, but we can combine the 3 data streams into a single graph by clicking on the Layers icon at the top left corner:

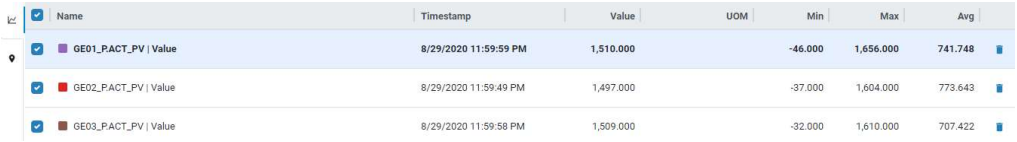
Selecting Single converts the trends into a single graph with a single y-scale as below.



Selecting Multiple converts the trend into a single graph but with multiple y-scales now:

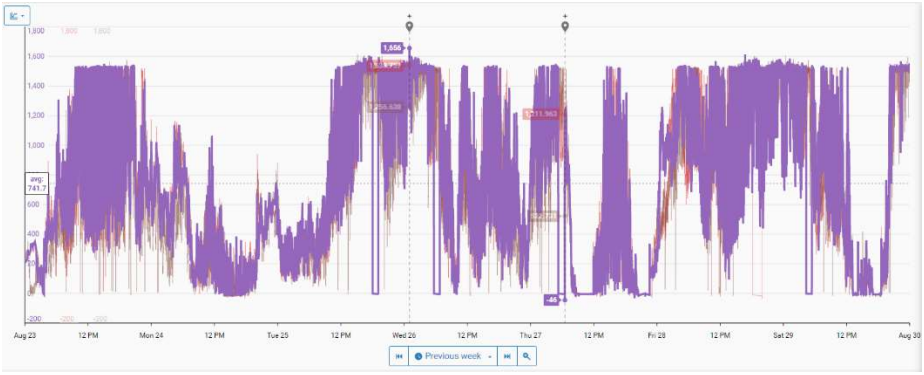


Clicking on a specific data stream in the table below the trend highlights the trend, and a trend can be added or removed by checking or unchecking the box to the left of the stream name. The last value, minimum, maximum, and average values are shown for each trace in the table, making it easy to analyze data over the entire time range in the trend.

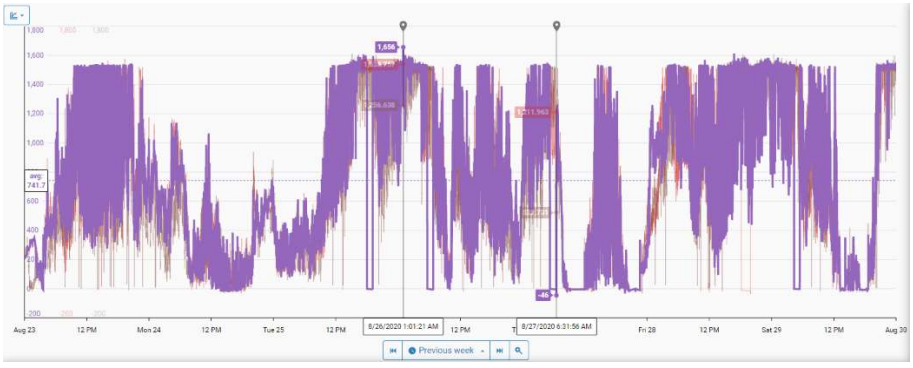


Name	Timestamp	Value	UOM	Min	Max	Avg
GE01_P.ACT_PV | Value	8/29/2020 11:59:59 PM	1,510.000		-46.000	1,656.000	741.748
GE02_P.ACT_PV | Value	8/29/2020 11:59:49 PM	1,497.000		-37.000	1,604.000	773.643
GE03_P.ACT_PV | Value	8/29/2020 11:59:58 PM	1,509.000		-32.000	1,610.000	707.422

When you select (highlight) a trace, its minimum and maximum are automatically marked as well as the average for the time range.



When multiple cursors are added to the trend, the duration, delta (difference), average, minimum, and maximums between consecutive cursors are calculated and displayed, to analyze periods in-between the cursors.



Timestamp	GE01_P.ACT_PV | Value	GE02_P.ACT_PV | Value	GE03_P.ACT_PV | Value
08/26/2020 01:01:21 AM	1,656.000	1,530.700	1,269.691
1d 5:30:35	Min: -46.000 Max: 1,656.000 Average: 839.083 Delta: -1,702.000	Min: 56.000 Max: 1,588.000 Average: 951.171 Delta: -373.307	Min: -15.000 Max: 1,596.000 Average: 824.589 Delta: -626.033
08/27/2020 06:31:56 AM	-46.000	1,157.393	643.658

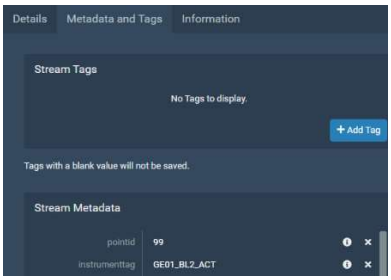
2.3.3.4 Managing Stream Metadata in OCS

Video	*Before reading this section, please refer to the following course YouTube video: https://youtu.be/wijDu4hM8TA*

Stream metadata in OCS is essential in providing context and associated information for data streams. In our case, we need to provide metadata for our wind farm data streams, so that we can identify which streams are associated with a specific turbine, and what type of measurement that stream describes.

This is needed when we create an OCS Data View in the next section for the subsequent machine learning application, and is analogous to metadata inclusion for elements in PI Asset Framework in order to be used later in a PI Integrator for Business Analytics, for instance.

Under the Sequential Data Store>Streams, there is a Metadata and Tags tab on the right:

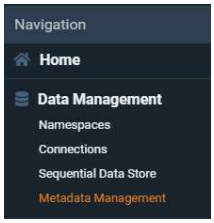


Stream Tags and Stream Metadata

Stream tags in SDS are used to categorize or denote special attributes of streams and are represented as a list of strings.

Stream metadata is represented as a dictionary of string keys and associated string values (i.e. key-value pairs). It can be used to associate additional information with a stream. When streaming data from a PI System to OCS, all configured PI Point attributes and classes such as PointID, Point Source, EngUnits, etc. are automatically included as Stream Metadata. If a wind farm stream such as GE10_P.ACT_PV is selected from the list, its Stream Metadata is shown as below, as these values were already set when first configuring these tags in the PI Data Archive.

Metadata for each stream can be added individually via the +Add Metadata button, but if metadata for multiple streams need to be added in bulk, the Metadata Management page under the Data Management section of the Navigation menu should be used instead.



Metadata Management

OCS Metadata Rules enable users to relate similar data streams for the purposes of analysis and display, and to feed into downstream applications. Metadata rules also create and assign metadata to the streams they identify in bulk. Metadata rules are created by selecting parsable metadata from stream name structures and applying the rule to identify all streams whose names match the defined pattern.

In this case, we need to create metadata rules for all the wind farm data streams in order to associate each stream with a specific turbine (i.e. GE01, GE02, etc.), as well as designate the measurement type (i.e. Active Power, Wind Speed, etc.).

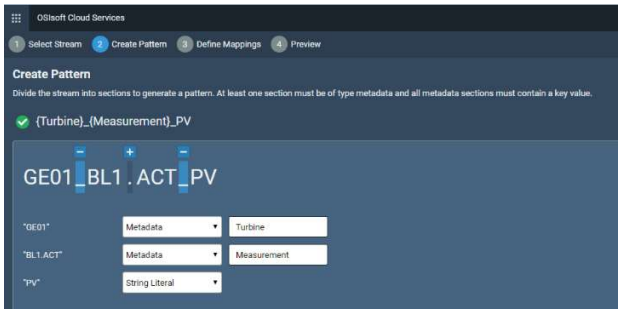
Note	*There is already a Metadata Rule created for this course called **Wind Turbine Metadata_APN_2020-08-28**. Please do not create any additional Metadata Rules because every student uses the same SDS Streams, and any new Metadata Rule created will overwrite the existing rule on the shared SDS Streams, thus creating potential errors/conflicts. This is because Metadata Rules are intended to only be used by Account Administrators when initially configuring data in Namespaces.* *You can view this existing Metadata Rule configuration by selecting the Edit Metadata Rule option. The information stated below is just for reference in case this Metadata Rule needs to be recreated from scratch.*

Under the Metadata Management section, click on Add Metadata Rule. On the Select Stream page, enter GE* in the Search bar. This will return all the tags for the wind turbines that were streamed to OCS:

The first step would be to select a known or familiar data stream that can be used as an example to define a consistent naming pattern. In this case, we are using the GE01_BL1.ACT_PV Stream Name as the template for this Metadata Rule. Click on Next to move to the next page.

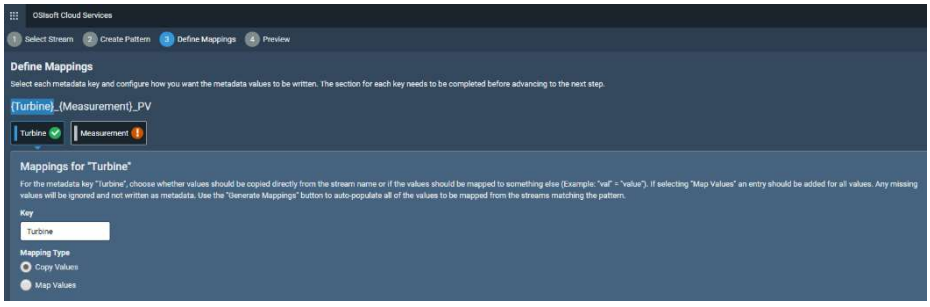
On this page, click on the + symbol above the underscore separating GE01 and BL1, as well as the + symbol above the underscore separating ACT and PV. These + symbols are automatically added based on delimiters detected in the stream name (e.g. underscores in this case) so that users can create a metadata pattern based on these delimiters.
GE01 and BL1.ACT are both valuable and interesting information that we can store as metadata for this data stream, as they correspond to the Turbine Name and Measurement, both required fields. Thus, enter the keys for GE01 and BL1.ACT_PV as Turbine and Measurement, respectively.

Each delimited section of a selected stream name can be designated as Metadata, String Literal, or Wildcard. Streams matching this pattern will be assigned the metadata pattern defined in this step.



PV is a String Literal in this case because all the relevant wind farm streams end with a fixed string “PV”, and this section of the stream name is not relevant for metadata management, thus can be ignored. If the streams ended with multiple different suffixes other than PV, that section could be defined as a Wildcard instead.

Click Next. For the metadata key "Turbine", select the Copy Values Mapping Type. This means that the metadata key values will be copied directly from the stream name section, i.e. Turbine = GE01 in this example.



For the metadata key “Measurement”, select the Map Values Mapping Type. Since each wind farm data stream has a different measurement type, the Generate Mappings button needs to be clicked to generate all the key-value metadata pairs to be mapped from the streams matching that pattern.

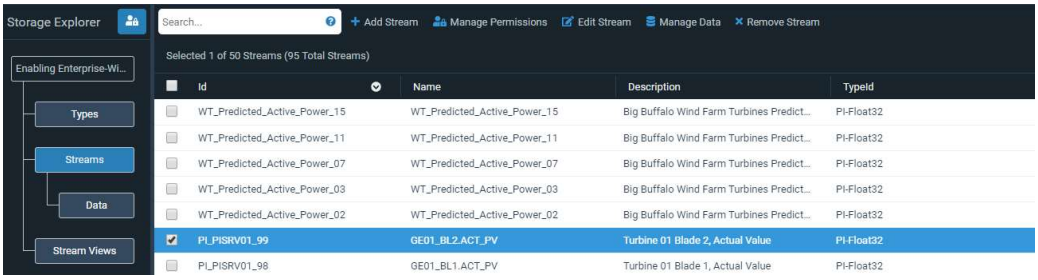
This Mapping Type means that the metadata key values will be mapped to something other than the stream name section, i.e. Blade1, Actual Value for BL1.ACT in this example. Any missing values will be ignored and not written as metadata. For each mapping, the format is below for all the wind turbine Measurement metadata:

Once these mappings are completed, click Next. Enter the name for this Metadata Rule, Wind Turbine Metadata Rule, then click on Save & Execute.

✓ Best Practice	*Stream names which follow a standard pattern are more applicable to the creation and use of metadata rules. The flexibility and complexity of metadata pattern definition will be further extended in future OCS versions.*

Tip	*The value of metadata itself lies in its capacity to enrich sequential data, and to facilitate logical segregation and contextualization. Other services and applications, such as OCS data views, leverage stream metadata to simplify finding data and to providing context.* *Please reach out to OSIsoft if there are any questions, comments or concerns regarding metadata management in OCS.*

In order to confirm that these metadata are added to the wind farm data, go to Data Management>Sequential Data Store under the Navigation menu. In the Streams section, select one of the wind turbine data streams such as GE01_BL2.ACT_PV by checking the box to the left of the Id.

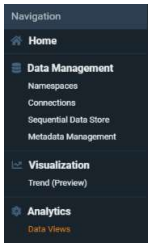


On the Metadata and Tags tab on the right, the Turbine and Measurement metadata values are displayed for the selected data stream under the Stream Metadata section.

2.3.3.5 Preparing Wind Farm Data for Machine Learning Using OCS Data Views

Video	*Before reading this section, please refer to the following course YouTube video: https://youtu.be/YRpU4xyZ69A*

Once the relevant stream metadata has been added, we now need to prepare our wind farm dataset to be used for machine learning, with the relevant associated metadata. For this purpose, we utilize the Data View feature in OCS under the Analytics section of the Navigation menu:

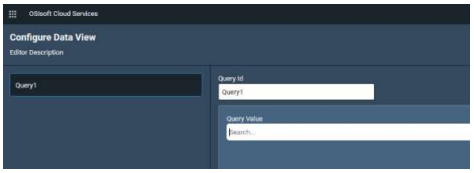


A Data View is a user-selected subset of data from one or more streams, presented in a tabular format.

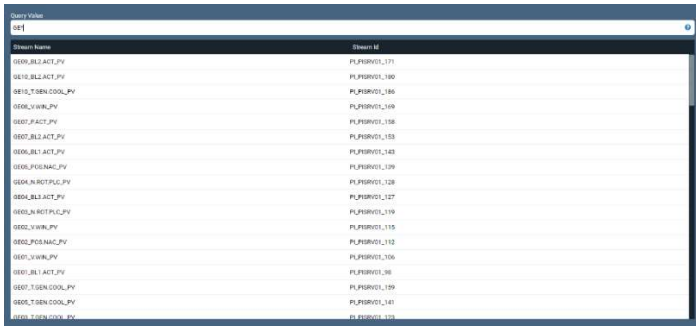
A data view is created by defining the target namespace, the source stream or streams, the desired data fields, time period, and interpolation interval. The ability to create data views in OCS meshes directly with OSIsoft's Data Science Enablement efforts, whereby users will be able to programmatically access Data View content for the purposes of advanced analytics. In this section, we will create a new Data View for the wind turbine data so that it is accessible externally in a Jupyter Notebook.

Note	*Unlike Metadata Rules, each student can create their own Data View, which will not impact the shared SDS Streams. Thus, feel free to follow the instructions below to create your own Data View in OCS.*

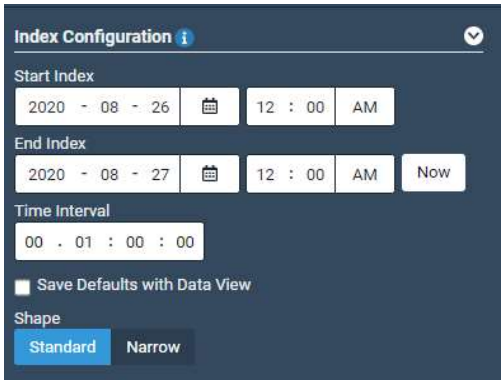
To create a new data view, click Add Data View in the Data Views pane.



In the Query Value bar, enter GE*. This will return all data streams pertaining to the 10 wind turbines.



Click Apply. On the next screen, under Index Configuration, keep the default settings for Start Index, End Index and Time Interval. These settings are not important now as they will be changed programmatically later. If the Save Defaults with Data View box is checked, the configuration set in these fields will be maintained the next time the Data View is accessed via the user interface.



When the Apply button is clicked with the default field configuration, the resulting Data View is as below:

Timestamp.0	DigitalStateName.1	IsAnnotated.2	IsQuestionable.3	IsSubstituted.4	SystemStateCode.5	Value.6	Digital
Aug 26, 2020, 12:00:00 AM		false	false	false		5.0779757	
Aug 26, 2020, 1:00:00 AM		false	false	false		8.037793	
Aug 26, 2020, 2:00:00 AM		false	false	false		3.128	
Aug 26, 2020, 3:00:00 AM		false	false	false		8.716666	
Aug 26, 2020, 4:00:00 AM		false	false	false		7.0324144	
Aug 26, 2020, 5:00:00 AM		false	false	false		0.9320645	
Aug 26, 2020, 6:00:00 AM		false	false	false		82.89608	
Aug 26, 2020, 7:00:00 AM		false	false	false		-0.14426066	
Aug 26, 2020, 8:00:00 AM		false	false	false		-0.54036814	
Aug 26, 2020, 9:00:00 AM		false	false	false		-0.4961364	
Aug 26, 2020, 10:00:00 AM		false	false	false		-0.13821733	
Aug 26, 2020, 11:00:00 AM		false	false	false		0.21970174	
Aug 26, 2020, 12:00:00 PM		false	false	false		-0.5395207	
Aug 26, 2020, 1:00:00 PM		false	false	false		-0.07352354	
Aug 26, 2020, 2:00:00 PM		false	false	false		-0.13814035	
Aug 26, 2020, 3:00:00 PM		false	false	false		0.4669166	
Aug 26, 2020, 4:00:00 PM		false	false	false		1.1318202	
Aug 26, 2020, 5:00:00 PM		false	false	false		-0.52339363	
Aug 26, 2020, 6:00:00 PM		false	false	false		-0.29750064	
Aug 26, 2020, 7:00:00 PM		false	false	false		-0.543748	
Aug 26, 2020, 8:00:00 PM		false	false	false		-0.56121814	

We need to modify the fields displayed so that the dataset is more relevant for our machine learning application. Since we added the Turbine and Measurement metadata field, we can see these fields under the Field Management section.

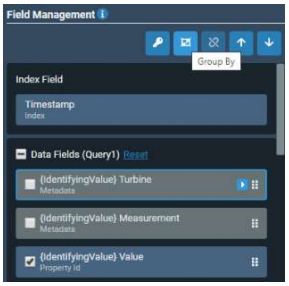
Uncheck all Fields, except for Timestamp, Turbine, Measurement and Value. Bring the checked fields up (the fields can be moved up or down by clicking on the ↑ or ↓ buttons).

Clicking the Apply button again will update the generated Data View as below:

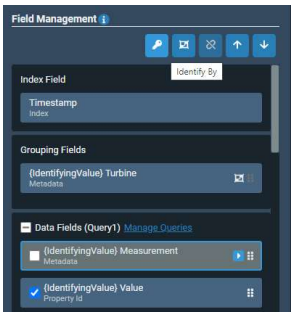
Timestamp.0	Value.1	Measurement.2	Turbine.3	Value.4	Measurement.5	Turbine.6	Value.
Aug 26, 2020, 12:00:00 AM	5.0779757	Blade1, Actual Value	GE01	4.957525	Blade2, Actual Value	GE01	5.1395
Aug 26, 2020, 1:00:00 AM	8.037793	Blade1, Actual Value	GE01	7.3965273	Blade2, Actual Value	GE01	6.9913
Aug 26, 2020, 2:00:00 AM	3.128	Blade1, Actual Value	GE01	2.85	Blade2, Actual Value	GE01	3.14
Aug 26, 2020, 3:00:00 AM	8.716666	Blade1, Actual Value	GE01	8.72	Blade2, Actual Value	GE01	8.6966
Aug 26, 2020, 4:00:00 AM	7.0324144	Blade1, Actual Value	GE01	7.2994647	Blade2, Actual Value	GE01	7.4672
Aug 26, 2020, 5:00:00 AM	0.9320645	Blade1, Actual Value	GE01	0.3976003	Blade2, Actual Value	GE01	0.9703
Aug 26, 2020, 6:00:00 AM	82.89608	Blade1, Actual Value	GE01	82.92	Blade2, Actual Value	GE01	83.435
Aug 26, 2020, 7:00:00 AM	-0.14426066	Blade1, Actual Value	GE01	0.049545456	Blade2, Actual Value	GE01	-0.254
Aug 26, 2020, 8:00:00 AM	-0.54036814	Blade1, Actual Value	GE01	-0.57023007	Blade2, Actual Value	GE01	-0.493
Aug 26, 2020, 9:00:00 AM	-0.4961364	Blade1, Actual Value	GE01	-0.4561734	Blade2, Actual Value	GE01	-0.516
Aug 26, 2020, 10:00:00 AM	-0.13821733	Blade1, Actual Value	GE01	-0.12322543	Blade2, Actual Value	GE01	-0.158
Aug 26, 2020, 11:00:00 AM	0.21970174	Blade1, Actual Value	GE01	0.20972253	Blade2, Actual Value	GE01	0.1997
Aug 26, 2020, 12:00:00 PM	-0.5395207	Blade1, Actual Value	GE01	-0.5647759	Blade2, Actual Value	GE01	-0.550
Aug 26, 2020, 1:00:00 PM	-0.07352354	Blade1, Actual Value	GE01	-0.115360186	Blade2, Actual Value	GE01	-0.116
Aug 26, 2020, 2:00:00 PM	-0.13814035	Blade1, Actual Value	GE01	-0.17232443	Blade2, Actual Value	GE01	-0.121
Aug 26, 2020, 3:00:00 PM	0.4669166	Blade1, Actual Value	GE01	0.4543899	Blade2, Actual Value	GE01	0.4650
Aug 26, 2020, 4:00:00 PM	1.1318202	Blade1, Actual Value	GE01	0.5929577	Blade2, Actual Value	GE01	0.4064
Aug 26, 2020, 5:00:00 PM	-0.52339363	Blade1, Actual Value	GE01	-0.44962445	Blade2, Actual Value	GE01	-0.516
Aug 26, 2020, 6:00:00 PM	-0.29750064	Blade1, Actual Value	GE01	-0.35086635	Blade2, Actual Value	GE01	0.3355
Aug 26, 2020, 7:00:00 PM	-0.543748	Blade1, Actual Value	GE01	-0.5085606	Blade2, Actual Value	GE01	-0.528
Aug 26, 2020, 8:00:00 PM	-0.56121814	Blade1, Actual Value	GE01	-0.61950004	Blade2, Actual Value	GE01	-0.561

We see that the Turbine and Measurement columns are repeated several times (with integer suffixes on the column names), with the Measurement column showing the Metadata key value, i.e. Blade1, Actual Value, instead of the actual measurement value, i.e. 5.0779757. Thus, we need to group these measurements and values by a specific turbine, identify the Value field by the Measurement key value, and not display the Measurement value as rows in the table.

Thus, uncheck the Measurement field, click on the Turbine field and select the Group By button. This groups the data streams by the Turbine metadata.

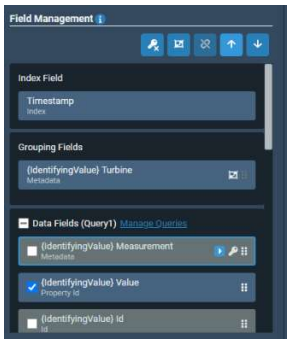


Then, click on the Measurement field and select the Identify By button. This identifies the data streams by the Measurement metadata.

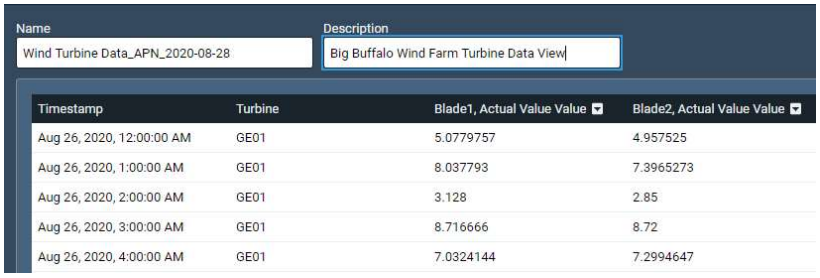


Optionally, you can also remove the {IdentifyingValue} prefix from the field names by clicking on the icon to the right of each data field and selecting the {FirstKey} value only from the menu.

However, it is recommended to keep the {IdentifyingValue} for the Value field, in order to identify which measurement a specific data stream corresponds to (e.g. if only the Value is selected, the column name will only state “Value” instead of the full Measurement key value such as Blade1, Actual Value Value” when including the {IdentifyingValue}). The resulting Field Management section should be as below:



Click the Apply button to refresh the resulting Data View, enter the name for the Data View as ***Wind Turbine Data_NNN_XXXX-XX-XX***, where NNN are your initials and XXXX-XX-XX is today’s date in four digit year, two digit month and two digit day respectively, then click Save. The resulting Data View should be as below:



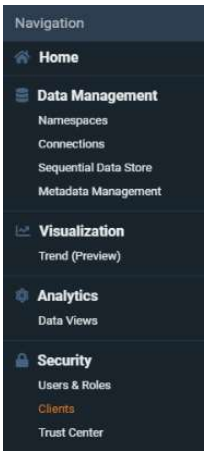
Timestamp	Turbine	Blade1, Actual Value Value	Blade2, Actual Value Value
Aug 26, 2020, 12:00:00 AM	GE01	5.0779757	4.957525
Aug 26, 2020, 1:00:00 AM	GE01	8.037793	7.3965273
Aug 26, 2020, 2:00:00 AM	GE01	3.128	2.85
Aug 26, 2020, 3:00:00 AM	GE01	8.716666	8.72
Aug 26, 2020, 4:00:00 AM	GE01	7.0324144	7.2994647

The Data View now appears to be in the correct format needed.

2.3.3.6 Connecting to OCS Data View using an OCS Client

Video	*Before reading this section, please refer to the following course YouTube video: https://youtu.be/14Si8-cnhXU*

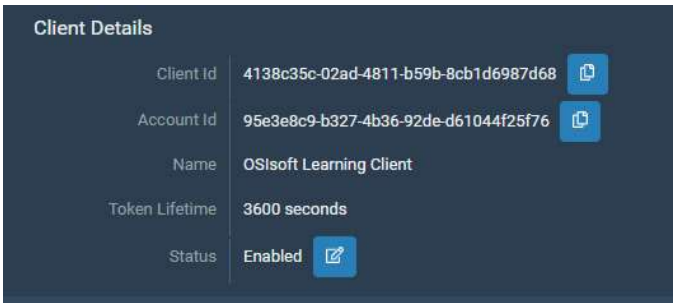
For external programs to connect to OCS, an OCS Client needs to be set up first. This Client's details will then be used to connect to OCS and access this Data View. Navigate to the Clients page under Security in the Navigation menu.



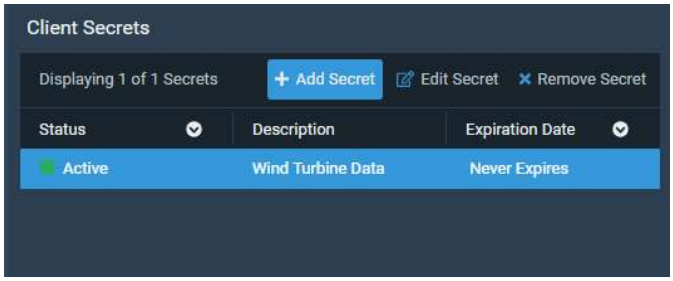
There is already a Client defined for this course, called ***OSIsoft Learning Client***. Click on this client to view Details, Code Examples and more Information. The Client Type for this case is Client Credentials.

Client-credentials type clients are used for server to server communication where no user interaction is required. The client typically must authenticate with the token endpoint using its Client Id and Client Secret, provided they are valid. The default (and maximum) token lifetime is 3600 seconds, which means that authentication must be performed again if this time period elapses. Please make note of the Client Id and Client Secret.

The Client Id can be obtained at any time for an OCS Client in the Client Details section under the Details tab:



However, the Client Secret is only available upon first creation of an OCS Client, thus needs to be copied immediately and stored in a safe place. However, if a user forgets or misplaces the Client Secret, another one can be added by clicking on +Add Secret in the Client Secrets section of the Clients page.



In this section, a user can also edit or remove an existing Client Secret. On the Edit Secret page, the Description or Expiration Date (including setting it to never expire) can be modified.

Note	*A Client Id and Client Secret are used to authenticate and retrieve a token that provides access to OCS. Any client that presents the Client Id and Client Secret will be able to authenticate and access data. A Client Id is not confidential, but a Client Secret is. Hence, it is imperative that the Client Secret be stored in a secure location that is accessible only to the client that needs to use it to authenticate and access OCS.*

For detailed information regarding adding a client-credentials type client, please refer to Appendix B.

Example code illustrating common connection syntax in different programming languages is provided on the Code Example tab on the right. These examples are important if a user does not want to use the OCS Python library but instead wants to connect to OCS via generic API calls instead.

3. Utilizing OCS Data in Machine Learning Model

3.1 Objective

The objective of this section is to connect to the wind farm OCS Data View created in the previous section, filter the data set to remove outliers and use this cleansed dataset to train, test and evaluate a machine learning model. This is all accomplished programmatically in a Jupyter Notebook via Python code.

3.2 Tasks

All the tasks below are to be done in a Jupyter Notebook with Python code:

- Connect to OCS and access the Wind Farm Data View
- Determine variables that affect Wind Farm Active Power
- Cleanse and prepare a Wind Farm dataset for machine learning
- Train & evaluate a selected machine learning algorithm
- Test trained machine learning model with sample input

3.3 Approach & Details

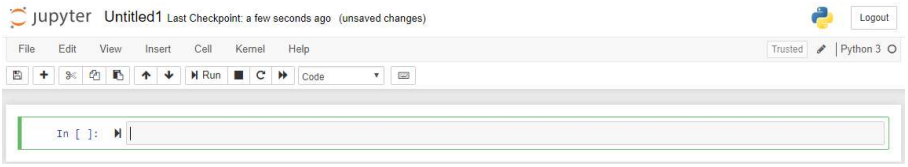
We will connect to the created OCS Data View in a Jupyter Notebook locally on your Virtual Machine.

3.3.1 Using Jupyter Notebook Installed Locally on Virtual Machine

Video	*Before reading this section, please refer to the following course YouTube video: https://youtu.be/npZYLFUvNG0*

In order to connect to the Jupyter Notebook locally on your Virtual Machine, open the Course Material folder on the Desktop, then double-click on the Start Jupyter Notebook.bat file. The screen below should appear after a few seconds:

Click on the New dropdown menu in the upper right corner, then select Notebook: Python 3 to create a new notebook with Python 3:



In the Course Material folder, the completed Jupyter Notebook is available for review titled ***Wind Turbine OCS Data_OCS Python Library***. It is suggested that you either follow along with this finished Notebook or copy and paste code sections into a new Notebook.

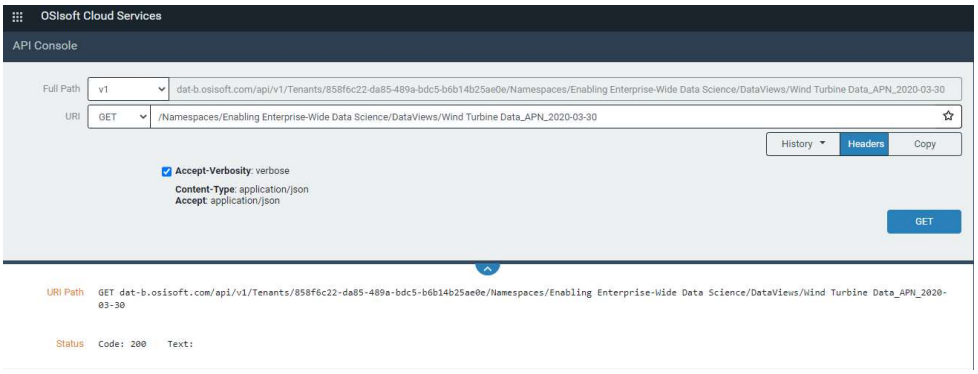
3.3.2 Connect to OCS and access the Wind Farm Data View

In this course, we are going to use the OCS Python library, which has built-in functions to perform the various OCS actions required. For documentation regarding the OCS Python library, as well as more help with Python, please refer to Appendix D.

Video	*Before reading the next section regarding the API Console, please refer to the following course YouTube videos:* 1. *https://youtu.be/BCPZOOJc16I* 2. *https://youtu.be/zBJ2Fz-EUsA* 3. *https://youtu.be/5QS_haLiu-0*

An alternative would be to access OCS via generic REST API calls. There is another Jupyter Notebook in the Course Materials folder titled ***Wind Turbine OCS Data_RESTAPI & OMF***, which shows you how to do the various OCS calls via REST API without the OCS Python library. There is an API Console under the Developer Tools section of the Navigation menu which helps users verify their generic API calls:

The API Console provides a graphical interface to utilize the SDS REST API, which enables you to read and write SDS data.



For more information regarding API calls, please refer to Appendix D.

Video	*Before reading the next section, please refer to the following course YouTube videos:* 1. https://youtu.be/uo7aCMtaaOc 2. https://youtu.be/atwGVfEDzBE

If we utilize the OCS Python library connection methodology in the Jupyter Notebook, we need to import the necessary Python libraries, modify the config.ini file, connect to OCS, then authenticate and pass the identity token.

The ***config.ini*** file is available for review in the Course Material folder. The strings referenced in the OCS Python library functions below correspond to the sections and content of this file, which needs to be modified with the correct namespace, tenant ID, Client Id, Client Secret, etc. for this course:

```
; IMPORTANT: replace these values with those provided by OSIsoft
[Configurations]
Namespace = Enabling Enterprise-Wide Data Science

[Access]
Resource = https://dat-b.osisoft.com
Tenant = 95e3e8c9-b327-4b36-92de-d61044f25f76
ApiVersion = v1

[Credentials]
ClientId = [redacted]
ClientSecret = [redacted]
```

Note the Client Id and Client Secret under the Credentials section of the config.ini file for connecting to the OCS Client we created, i.e. PI World 2020 Course Client.

Video	*Before reading the next section, please refer to the following course YouTube video:* https://youtu.be/wUDY5wpjYwg

We first import the necessary Python packages in our Jupyter Notebook and then connect to OCS, where the code for this is as below:

```
In [1]: get_ipython().magic(u'config IPCompleter.greedy=True')

In [2]: import requests
        import configparser
        import json
        import pandas as pd
        from datetime import date, timedelta
        from ocs_sample_library_preview import *

In [3]: config = configparser.ConfigParser()
        config.read('config.ini')

        ocsClient = OCSClient(config.get('Access', 'ApiVersion'), config.get('Access', 'Tenant'), config.get('Access', 'Resource'),
                              config.get('Credentials', 'ClientId'), config.get('Credentials', 'ClientSecret'))

        namespaceId = config.get('Configurations', 'Namespace')
```

The next step is to get the Data View and pass it to a pandas Data Frame. The Start Index is set to 10 days prior to today's date, the End Index is set to today's date and the Time Interval is set to be 1 minute. The count for the maximum number of rows of the Data View passed to the Data Frame needs to be increased from the default of 1,000 to 150,000, so that all data contained is retrieved. Please ensure that the dataviewId corresponds to the Data View name entered previously i.e. ***Wind Turbine Data_NNN_XXXX-XX-XX***, where NNN are your initials and XXXX-XX-XX is today's date in four digit year, two digit month and two digit day respectively (e.g. Wind Turbine Data_APN_2020-08-28 in the example code below).

```
In [5]: #dateFrom = '2020-01-12'
        #dateTo ='2020-01-23'
        dateFrom = str(date.today()-timedelta(days=10))
        dateTo = str(date.today()) #10 days of data
        timeinterval = '00:01:00' #interpolate every minute

        #dataviewId = Wind Turbine Data_NNN_XXXX-XX-XX, where NNN are your initials and XXXX-XX-XX is today's date in four
        # digit year, two digit month and two digit day respectively (Wind Turbine Data_APN_2020-03-23 in this example)
        dataviewId= "Wind Turbine Data_APN_2020-08-28"

        data, nextPage, firstPage = ocsClient.DataViews.getDataInterpolated(namespaceId, dataviewId, startIndex=dateFrom,
                                                                            endIndex=dateTo, interval=timeinterval, count=150000)

        df = pd.DataFrame(data)
        df["Timestamp"] = pd.to_datetime(df["Timestamp"])
        df
```

3.3.3 Determine variables that affect Wind Farm Active Power

Video	*Before reading this section & Section 3.3.4.1, please refer to the following course YouTube video: https://youtu.be/e_5msGD5EME*

As a data scientist creating a machine learning model, one needs to be aware of the features that affect the label prediction. This would need to be discussed with the Subject Matter Expert (SME). Thus, in order to determine the features that affect the Active Power for a wind turbine, a correlation plot is added via the matplotlib Python library. However, first, the Data Frame column names must be abbreviated, in order to be displayed clearly in the plot. The code section for this, and the corresponding plot, are below:

```
In [10]:
import matplotlib.pyplot as plt
import numpy as np
```

```
In [11]:
#Renaming DataFrame column names to abbreviations, in order to display these column names clearly in a correlation plot

df.rename(columns = {'Blade1, Actual Value Value':'BL1', 'Blade2, Actual Value Value':'BL2',
                     'Blade3, Actual Value Value':'BL3', 'Rotor Speed Value':'RS','Turbine State Value':'TS',
                     'Active Power Value':'AP', 'Nacelle Position Value':'NP', 'Air Temperature Value':'AT',
                     'Wind Speed Value':'WS'}, inplace = True)
```

```
In [12]:
#Check the correlation between Active Power and the rest of the variables

#retrieve the correlation table
df_corr = df.corr()

#increase the size of the figure
fig = plt.figure(figsize=(50,10))
ax = fig.add_subplot(111)

#set the color pallete (Red, yellow, green)
cax = ax.matshow(df_corr, cmap=plt.cm.RdYlGn)
fig.colorbar(cax)

#configure the labels
labels = [c for c in df_corr.columns]

#make sure to show all the labels
ax.set_xticks(np.arange(len(labels)))
ax.set_yticks(np.arange(len(labels)))

#Setting labels for the x and y axes of the correlation plot
ax.set_xticklabels(labels)
ax.set_yticklabels(labels)

plt.show()
```

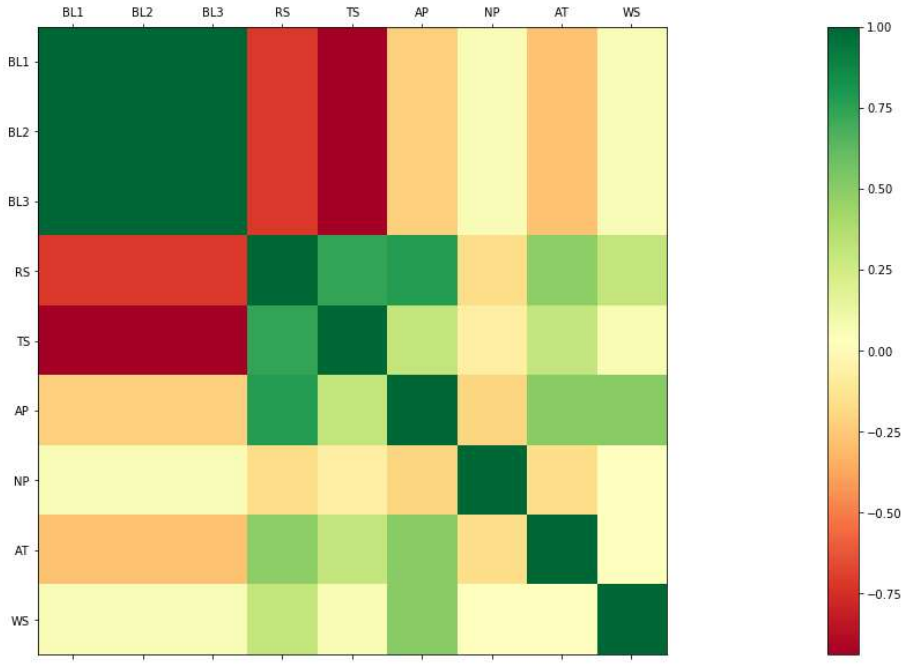


Figure 4: Correlation plot for Wind Turbine Attributes

As per the plot above, the Active Power is strongly correlated with Air Temperature, Rotor Speed, Turbine State and Wind Speed. Since Rotor Speed is directly affected by Wind Speed and the turbine only generates power when it is operating (the Turbine State), the two weather variables, Wind Speed and Air Temperature, are selected as the features of interest for our machine learning model, with Active Power as the desired label.

3.3.4 Cleanse and prepare a Wind Farm dataset for machine learning

3.3.4.1 Generating Wind Turbine Power Curve & Identifying Outlier Data Regions

In order to be consistent with the original Data View column names, we rename the Data Frame columns back to their original titles. Next, a scatter plot of Active Power vs. Wind Speed is added and visualized:

```
In [13]: #Renaming DataFrame column names from abbreviations back to their original full names

         df.rename(columns = {'BL1':'Blade1, Actual Value Value', 'BL2':'Blade2, Actual Value Value',
                              'BL3':'Blade3, Actual Value Value', 'RS':'Rotor Speed Value','TS':'Turbine State Value',
                              'AP':'Active Power Value','NP':'Nacelle Position Value', 'AT':'Air Temperature Value',
                              'WS':'Wind Speed Value'}, inplace = True)

In [14]: #Plotting Active Power versus Wind Speed

         fig = plt.figure(figsize=(10,10))
         ax = fig.add_subplot(111)
         ax.scatter(df['Wind Speed Value'], df['Active Power Value'])
         ax.set_xlabel('Wind Speed (m/s)')
         ax.set_ylabel('Active Power (kW)')
         ax.set_title('Active Power vs Wind Speed')

         plt.show()
```

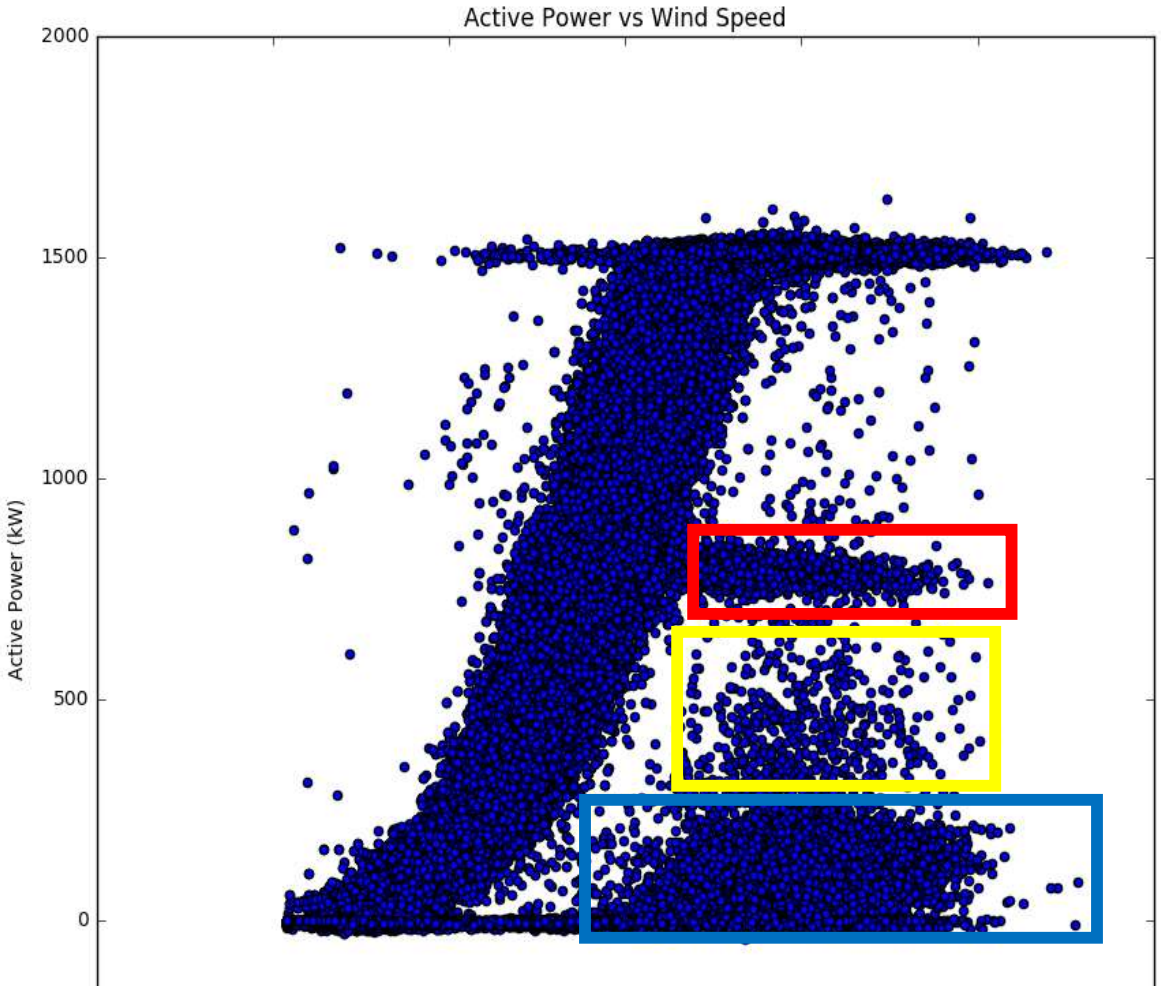


Figure 5: Plot of Wind Turbine Active Power vs. Wind Speed for Unfiltered Dataset

The data seems to describe something like the typical power curve for a turbine. This is good news for our modeling efforts. However, the data set needs to be cleaned up before using it to train a machine learning model. Upon consultation with the subject matter expert (SME) for wind turbines, 3 outlier regions are identified:

I. The red region is due to one of the turbines (GE05) having a different Rated Power than the others.

II. The blue region is due to the turbines not producing power even though the wind is blowing; this could be due to maintenance, shutdowns, or curtailments (e.g., the grid will not accept power).

III. The yellow region is due partially to some short periods of low production between stops and mostly to an unexplained period of low production during high winds.

These outliers need to be filtered out before we send the data to a machine learning model.

First, any negative values of Active Power are removed. Next, in order to exclude operating periods corresponding to high wind with lower than expected power generation (yellow box in scatter plot), a filter criterion is used when 'Wind Speed' > 10 And 'Active Power' < 600. Typically, the power is expected to be above 1000 kW for the turbines of interest at these high wind speeds.

For the red box in the scatter plot, GE05 is also excluded from the data set, because it has a much lower Rated Power than the other turbines. Also, high wind speeds (above 13 m/s) that do not alter the Active Power are also removed.

For the blue box in the scatter plot (i.e. to eliminate times of little or no production due to curtailments, maintenance, etc.), we need to know the Turbine State of the wind turbines. We have a Turbine State Value column in our Data View, but these are integer values which correspond to specific operating states. Thus, we need to go back to the OCS portal and create another Data View to understand the operating states these integers represent.

3.3.4.2 Creating a New Data View in OCS for Turbine State Values

 Video	*Before reading this section, please refer to the following course YouTube video: https://youtu.be/RH1azsfcSDE*

Under Analytics>Data Views in the Navigation menu, click on +Add Data View. In the Query Id field, enter Turbine State Codes_ ***NNN_XXXX-XX-XX***, where NNN are your initials and XXXX-XX-XX is today's date in four digit year, two digit month and two digit day, respectively. In the Query Value field, enter *_OS* to only get the data streams corresponding to the 10 turbines' operating states.

Query Value

_OS

Stream Name	Stream Id
GE01_OS_PV	PI_PISRV01_102
GE02_OS_PV	PI_PISRV01_111
GE03_OS_PV	PI_PISRV01_120
GE04_OS_PV	PI_PISRV01_129
GE05_OS_PV	PI_PISRV01_138
GE06_OS_PV	PI_PISRV01_147
GE07_OS_PV	PI_PISRV01_156
GE08_OS_PV	PI_PISRV01_165
GE09_OS_PV	PI_PISRV01_174
GE10_OS_PV	PI_PISRV01_183

Click Apply and keep the default Index Configuration. Under the Field Management section, only check the Timestamp, Turbine, Value and DigitalStateName fields, Group By the Turbine field and Identify By the Measurement field (without checking the Measurement field). Click Apply once again, and this Data View should be as below:

Timestamp	Turbine	Turbine State DigitalStateNa...	Turbine State Value
Mar 3, 2020, 12:00:00 AM	GE01	Load Operation	16
Mar 3, 2020, 1:00:00 AM	GE01	Idle Position	12
Mar 3, 2020, 2:00:00 AM	GE01	Feathering Position	11
Mar 3, 2020, 3:00:00 AM	GE01	Feathering Position	11
Mar 3, 2020, 4:00:00 AM	GE01	Feathering Position	11
Mar 3, 2020, 5:00:00 AM	GE01	Feathering Position	11
Mar 3, 2020, 6:00:00 AM	GE01	Load Operation	16
Mar 3, 2020, 7:00:00 AM	GE01	Idling Mode	15
Mar 3, 2020, 8:00:00 AM	GE01	Load Operation	16
Mar 3, 2020, 9:00:00 AM	GE01	Load Operation	16
Mar 3, 2020, 10:00:00 AM	GE01	Load Operation	16
Mar 3, 2020, 11:00:00 AM	GE01	Idling Mode	15

From this table, we can surmise that a Turbine State Value of 16 refers to the Load Operation state. This corresponds to times when the wind is blowing, and the turbine is generating, which is the desired operating state. We do not need to save this new Data View, thus click Cancel.

Open the previous Data View created by selecting Wind Turbine Data_NNN_XXXX-XX-XX and clicking Edit Data View. We can see that the Turbine State Values are already included in our previous Data View.

Turbine State Value
16
12
11
11
11
11
16
15
16
16
16
15

Thus, in the df Data Frame, we need to use a filter criterion when the ‘Turbine State Value’ = 16.

3.3.4.3 Applying Filter Criteria to Cleanse Wind Turbine Dataset

Video	*Before reading this section, please refer to the following course YouTube video: https://youtu.be/QYNGtMKqhOc*

The code for all these filter operations to cleanse our dataset is below:

```
In [15]: #Remove the GE05 turbine rows from the data frame because it has a lower rating relative to all the other turbines
filterOutGE05 = df['Turbine'] != "GE05"
df_Filter = df[filterOutGE05]

In [16]: #Filter out negative & excessive Active Power Values
filterNegativeActivePower = (df_Filter['Active Power Value'] >= 0)
df_Filter = df_Filter[filterNegativeActivePower]

In [17]: #Remove the rows where we have a high wind speed and low active power in order to keep only the normal operating conditions
filterOutLowPowerHighWindSpeedData = (~((df_Filter['Wind Speed Value'] >  10) & (df_Filter['Active Power Value'] <600)))
df_Filter = df_Filter[filterOutLowPowerHighWindSpeedData]

In [20]: #Keep only the rows which correspond to the "Load Operation" state, which means Turbine State Value=16
filterLoadOperationState = df_Filter['Turbine State Value'] == 16
df_Filter = df_Filter[filterLoadOperationState]

In [21]: #Filter out high Wind Speeds (> 13 m/s) that do not change the Active Power results
filterOutHighWind = df_Filter['Wind Speed Value'] <13
df_Filter = df_Filter[filterOutHighWind]
```

The updated scatter plot now more closely resembles a typical Power vs. Wind Speed curve for a wind turbine (minus the section above the rated speed) as per below:

```
In [22]: #Plotting Active Power versus Wind Speed - filtered data frame representing Normal Operating Conditions

fig = plt.figure(figsize=(10,10))
ax = fig.add_subplot(111)
ax.scatter(df_Filter['Wind Speed Value'], df_Filter['Active Power Value'])
ax.set_xlabel('Wind Speed (m/s)')
ax.set_ylabel('Active Power (kW)')
ax.set_title('Active Power vs Wind Speed')

plt.show()
```

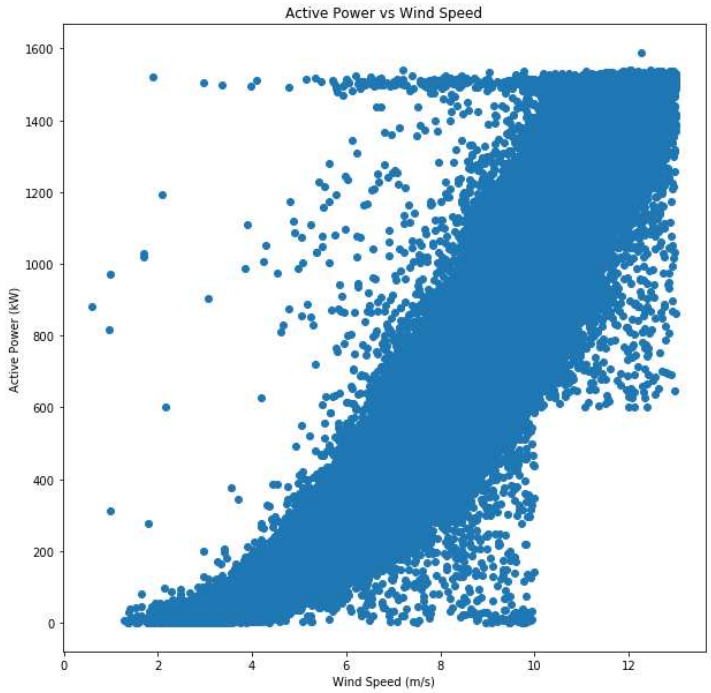


Figure 6: Plot of Wind Turbine Active Power vs. Wind Speed for Filtered Dataset

3.3.5 Train & evaluate a selected machine learning algorithm

Video	*Before reading this section, please refer to the following course YouTube videos:* 1. https://youtu.be/-Xu099ubv9w 2. https://youtu.be/jqQcfEnIzaA 3. https://youtu.be/9pwUHx60Tus 4. https://youtu.be/WkneG5Hd6cI

The next step would be to randomly split the overall data into a training set and a testing set using a function from the scikit-learn Python package. 75% of the data set is used as training data, while 25% is used as scoring/testing data. The label data used are the Active Power values while the feature data used are the Air Temperature and Wind Speed values:

```
In [110]: #Prepare the training & testing/scoring data sets, and split them randomly
          from sklearn.model_selection import train_test_split
          #define the target variable to be predicted
          y = df_Filter['Active Power Value'].values
          #split the dataset randomly into test and train sets
          X_train, X_test, y_train, y_test = train_test_split(df_Filter[['Air Temperature Value', 'Wind Speed Value']].values,
                                                              y, test_size=0.25, random_state=42)
```

The machine learning model that we are selecting to use for this wind turbine case will be the Decision Tree Regressor (DTR) model, which is available in the scikit-learn package. For more information regarding the DTR algorithm, please refer to Appendix D.

```
In [111]: #Use the Decision Tree Regression Machine Learning model from scikit-learn
          from sklearn.tree import DecisionTreeRegressor
          regr_1 = DecisionTreeRegressor(max_depth=2)
          regr_2 = DecisionTreeRegressor(max_depth=5)
          regr_1.fit(X_train, y_train)
          regr_2.fit(X_train, y_train)

          # Predict
          y_1 = regr_1.predict(X_test)
          y_2 = regr_2.predict(X_test)
```

The resulting fit to the data by the machine learning model can be plotted as per the code below:

```
In [130]: # Plot the results
          plt.figure(figsize=(10,10))
          plt.scatter(X_train[:,1], y_train, s=20, edgecolor="black", c="darkorange", label="data")
          plt.plot(X_test[:,1], y_1, color="cornflowerblue",label="max_depth=2", linewidth=2)
          plt.plot(X_test[:,1], y_2, color="yellowgreen", label="max_depth=5", linewidth=2)
          plt.xlabel("Wind Speed")
          plt.ylabel("Active Power")
          plt.title("Decision Tree Regression")
          plt.legend()
          plt.show()
```

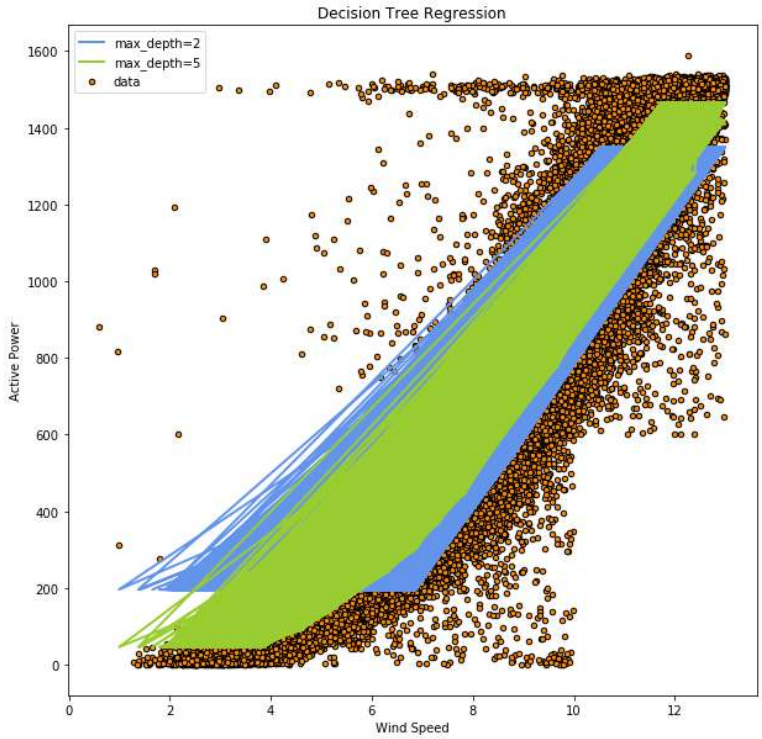


Figure 7: Wind Turbine Power Curve with DTR Algorithm Fit to Filtered Dataset

As can be seen from the plot above, the Decision Tree Regressor model with a maximum depth of 5 decision trees fits the training dataset well.

3.3.6 Test trained machine learning model with sample input

Video	*Before reading this section, please refer to the following course YouTube video: https://youtu.be/F3JhYkHKvZE*

Next, this machine learning model is saved to a file via the pickle module, so that we can access it later and feed live weather data to it. This model is then opened from the file, tested/scored and a sample prediction generated from a given set of air temperature and wind speed values, as per below:

In [26]:
```
#save the machine learning model to disk
import pickle
filename = 'WT_ActivePower_model.sav'
pickle.dump(regr_2, open(filename, 'wb'))
```

In [27]:
```
#Test the model with the scoring/testing data set
loaded_model = pickle.load(open(filename, 'rb'))
result = loaded_model.score(X_test, y_test)
#print the model score
print(result)
```

```
0.9041954554579937
```

In [28]:
```
#Sample prediction
# define input
new_input = [[45, 6.6]] #Temp=45 F, Wind Speed = 6.6 m/s
# get prediction for new input
new_output = regr_2.predict(new_input)
print(new_output)
```

```
[302.35981876]
```

The coefficient of determination (R^2) of the testing data set is approximately 0.9, which indicates that the algorithm picked can generate sufficiently accurate active power predictions from a given set of wind speed and air temperature data. The sample input of 45 F and 6.6 m/s generated an active power result of 302 kW.

Congratulations! You have successfully created a predictive machine learning model using data from OCS in a Jupyter Notebook!

4. Getting Forecasted Weather Data into ML Model

4.1 Objective

The objective of this section is to obtain live weather forecasts for Amarillo, TX (the location where the wind farm operates) and predict Active Power using the previously created Machine Learning model, given Wind Speed and Air Temperature forecast data.

4.2 Tasks

- Connect and retrieve forecasted weather data via the Open Weather API
- Utilize retrieved weather data to predict Active Power from machine learning model

4.3 Approach & Details

4.3.1 Connect and retrieve forecasted weather data via the Open Weather API

Video	*Before reading this section, please refer to the following course YouTube videos:* 1. *https://youtu.be/X50WOWhlGGw* 2. *https://youtu.be/M-rLSyngYHc*

Live weather forecast data is available via the Open Weather API: https://openweathermap.org/api. A free subscription can be obtained, which allows programmatic access via an API key to obtain weather data for many cities for 5 days into the future, updated every 3 hours, including for Amarillo, TX. This weather data can be brought into the Jupyter Notebook via the code section below, which returns the result in a JavaScript Object Notation (JSON) format.

```
In [117]: #Call the OpenWeather API to retrieve the forecasted air temperature and wind speed for Amarillo, TX for the next 5 days
          import requests
          url="https://api.openweathermap.org/data/2.5/forecast?q=Amarillo,US&APPID=5dac981ce33f41f61d8d1ea06ee89798"
          responseWeatherForecast=requests.get(url)

In [118]: responseWeatherForecast.json()

Out[118]: {'city': {'coord': {'lat': 35.222, 'lon': -101.8313},
            'country': 'US',
            'id': 5516233,
            'name': 'Amarillo',
            'population': 190695,
            'sunrise': 1581687314,
            'sunset': 1581726481,
            'timezone': -21600},
           'cnt': 40,
           'cod': '200',
           'list': [{'clouds': {'all': 99},
             'dt': 1581714000,
             'dt_txt': '2020-02-14 21:00:00',
             'main': {'feels_like': 270.84,
              'grnd_level': 890,
              'humidity': 67,
              'pressure': 1020,
              'sea_level': 1020,
              'temp': 279.54,
```

Note	*If an error is received while accessing the Open Weather API, please try again later, as this is most probably due to the limit of 60 calls per minute offered by the free subscription being exceeded.*

Video	*Before reading the next section, please refer to the following course YouTube video: https://youtu.be/sy3ofJ7sVlw*

This JSON response is stored in a pandas DataFrame, with the necessary unit and format conversions needed. For instance, the air temperature from Open Weather is sampled in degrees Kelvin, thus needs to be converted to degrees Fahrenheit. The wind speed is sampled in meters/sec, so no conversion is needed. In addition, the timestamp in the JSON response is a string, thus needs to be converted to a Date/Time format. The Timestamp, Air Temperature and Wind Speed values are all stored as NumPy arrays. The code for this section is as below:

```
In [119]: #Store the forecasted air temperature, wind speed and timestamp from the API json response in a pandas DataFrame

from decimal import Decimal
import datetime
TempArray = []
WindSpeedArray = []
TimestampArray = []

for val in responseWeatherForecast.json()["list"]:
    tempKelvin = val["main"]["temp"]
    tempF = round(((tempKelvin - 273.15) * (9/5)) + 32, 2)
    windSpeedMeterPerSec = round(val["wind"]["speed"], 2)
    np.array(TempArray.append(tempF))
    np.array(WindSpeedArray.append(windSpeedMeterPerSec))
    np.array(TimestampArray.append(datetime.datetime.strptime(val["dt_txt"], '%Y-%m-%d %H:%M:%S')))

dfWeatherForecast = pd.DataFrame({'Timestamp':TimestampArray, 'Temp (F)':TempArray, 'Wind Speed (m/s)':WindSpeedArray})

dfWeatherForecast
```

Out[119]:

	Temp (F)	Timestamp	Wind Speed (m/s)
0	43.50	2020-02-14 21:00:00	9.74
1	42.60	2020-02-15 00:00:00	9.44
2	41.81	2020-02-15 03:00:00	9.71
3	40.30	2020-02-15 06:00:00	8.90
4	40.35	2020-02-15 09:00:00	9.77
5	40.73	2020-02-15 12:00:00	9.74
6	43.81	2020-02-15 15:00:00	6.17
7	51.91	2020-02-15 18:00:00	5.49
8	55.22	2020-02-15 21:00:00	3.49
9	52.12	2020-02-16 00:00:00	2.61
10	44.78	2020-02-16 03:00:00	3.14
11	41.72	2020-02-16 06:00:00	3.46
12	40.30	2020-02-16 09:00:00	2.92

4.3.2 Utilize retrieved weather data to predict Active Power from ML model

Video	*Before reading this section, please refer to the following course YouTube video: https://youtu.be/uexAC58YiwY*

This Air Temperature and Wind Speed data from Open Weather are then passed to the machine learning model saved in a file previously, in order to predict the Active Power values.

These predicted Active Power values (in units of kW) are then stored in the same Data Frame as a new column, as per the code section below:

In [120]:
```
#Use the machine learning model developed previously to predict the Active Power and add the values to the existing Data Frame

import pickle
filename = 'WT_ActivePower_model.sav'
loaded_model = pickle.load(open(filename, 'rb'))

PredictedPowerArray=[]

for index, row in dfWeatherForecast.iterrows():
    new_input = [[row['Temp (F)'], row['Wind Speed (m/s)']]]
    result = loaded_model.predict(new_input)
    np.array(PredictedPowerArray.append(result))

dfWeatherForecast['Predicted Active Power (kW)']=pd.DataFrame(PredictedPowerArray)

dfWeatherForecast
```

Out[120]:

	Temp (F)	Timestamp	Wind Speed (m/s)	Predicted Active Power (kW)
0	43.50	2020-02-14 21:00:00	9.74	864.288650
1	42.60	2020-02-15 00:00:00	9.44	804.135001
2	41.81	2020-02-15 03:00:00	9.71	864.288650
3	40.30	2020-02-15 06:00:00	8.90	730.086759
4	40.35	2020-02-15 09:00:00	9.77	864.288650
5	40.73	2020-02-15 12:00:00	9.74	864.288650
6	43.81	2020-02-15 15:00:00	6.17	249.480099
7	51.91	2020-02-15 18:00:00	5.49	198.153096
8	55.22	2020-02-15 21:00:00	3.49	48.226675
9	52.12	2020-02-16 00:00:00	2.61	48.226675
10	44.78	2020-02-16 03:00:00	3.14	48.226675
11	41.72	2020-02-16 06:00:00	3.46	48.226675
12	40.30	2020-02-16 09:00:00	2.92	48.226675
13	38.75	2020-02-16 12:00:00	3.38	48.226675

5. Sending External Data Back to OCS

5.1 Objective

The objective of this section is to send the forecasted data back to OCS as a new stream in the Sequential Data Store (SDS) of the Namespace we are using for this course, then graph this data in the Visualization section.

5.2 Tasks

- Create a new complex SDS Type for the forecasted data
- Create a new SDS Stream
- Send forecasted data back to OCS via this new SDS Stream
- Analyze & visualize forecasted data in OCS

5.3 Approach & Details

 Video	*Before starting this section, please refer to the following course YouTube video: https://youtu.be/4pRcV4jS668*

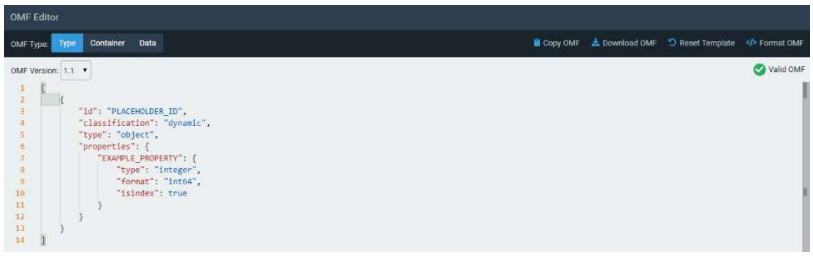
In order to send data back to OCS in this course, we utilize the OCS Python Library once again.

However, we can also do this via the OSIsoft Message Format or OMF protocol. This method is shown in the other Jupyter Notebook in the Course Materials folder titled ***Wind Turbine OCS Data_RESTAPI & OMF***.

 Video	*Before reading the next section regarding OMF, please refer to the following course YouTube videos:* 1. *https://youtu.be/5bwtHI9NS7U* 2. *https://youtu.be/L43UB8Iae7U*

OMF defines a set of message headers and message bodies, which are written in JSON and are used to generate compliant messages for data ingress. For OCS, OMF can be used to create types, create streams, and populate streams with data. There are three message formats that can be used to accomplish these tasks: type messages, container messages, and data messages, respectively. The OMF Editor in the

Developer Tools section of the Navigation menu helps build and validate OMF messages to be ingested by the OCS Data Store.



For more information regarding OMF, please refer to Appendix D.

But in order to use the OMF methodology, a new OMF Connection needs to be created in OCS first, so that we can use this channel to send data streams to the SDS. For more information on configuring a new OMF Connection, please refer to Appendix C.

5.3.1 Creating a new SDS Type

Video	*Before reading this section, please refer to the following course YouTube video: https://youtu.be/8w748p0sl54*

In order to send our forecasted data (which includes Air Temperature, Wind Speed and Active Power values) back to OCS, a new SDS Type needs to be created first. This new type will be complex, i.e. a collection of simple atomic types. This time-series indexed complex type with 3 values will be utilizing two simple types, DateTime and Double, in order to define the data shape/structure.

Note	*The alternative **Wind Turbine OCS Data_RESTAPI & OMF** Notebook in the Course Materials folder does not create a new SDS Type for the new data stream but instead uses the pre-existing simple PI-Float32 type.*

In order to create a new SDS Type, there are 2 methods:

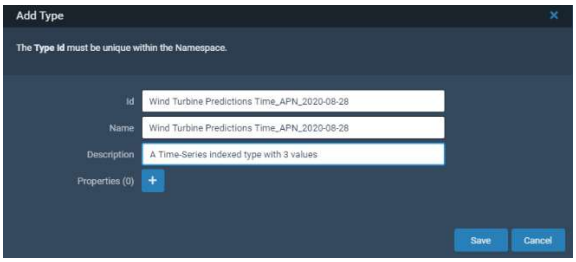
i. Create the new type via the user interface
ii. Create the new type programmatically

Note	*Unlike Metadata Rules, each student can create their own SDS Types. Thus, feel free to follow the instructions below to create your own SDS Types in OCS.*

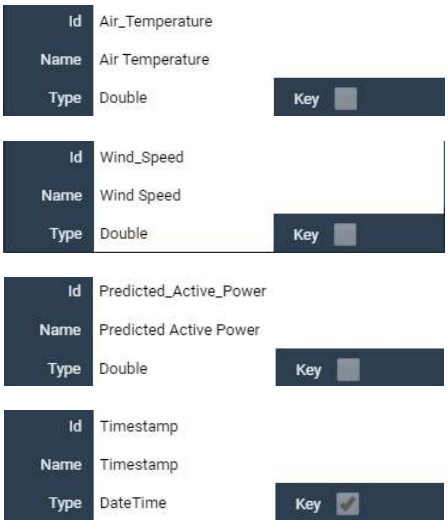
Creating New SDS Type Using the User Interface

The first step would be to navigate to the Sequential Data Store under Data Management in the Navigation menu.

The next step would be to click on Types, then Add Type. Enter ***Wind Turbine Predictions Time_NNN_XXXX-XX-XX*** for the Id and Name, where NNN are your initials and XXXX-XX-XX is today's date in four digit year, two digit month and two digit day respectively (i.e. Wind Turbine Predictions Time_APN_2020-08-28 in the example below). Under the Description field, enter "A Time-Series indexed type with 3 values".



Click the + button to the right of the Properties field, so that there are 4 total Properties. These are for Air Temperature, Wind Speed, Predicted Active Power and Timestamp. Enter the following for Id, Name and Type, then check the Key box for Timestamp:



Click Save to store this new complex SDS Type.

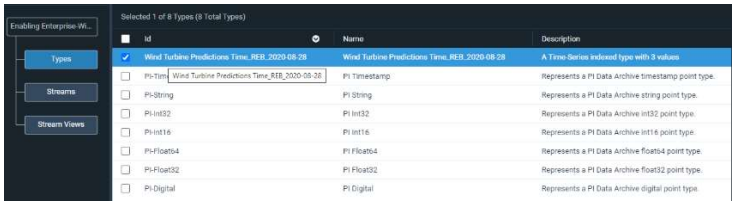
Creating New SDS Type Programmatically using OCS Python Library

In order to create the new SDS Type programmatically, we first need to reconnect to OCS and authenticate our token using the config.ini file and the OCS Python library functions, then create this new SDS type titled ***Wind Turbine Predictions Time_NNN_XXXX-XX-XX***, where NNN are your initials and XXXX-XX-XX is today's date in four digit year, two digit month and two digit day respectively ((i.e. Wind Turbine Predictions Time_REB_2020-08-28 in the example below).

In [26]:
```
config = configparser.ConfigParser()
config.read('config.ini')

ocsClient = OCSClient(config.get('Access', 'ApiVersion2'), config.get('Access', 'Tenant'), config.get('Access', 'Resource'),
                      config.get('Credentials', 'ClientId'), config.get('Credentials', 'ClientSecret'))

namespaceId = config.get('Configurations', 'Namespace')
```

In [27]:
```
#SDS Type Name = Wind Turbine Predictions Time_NNN_XXXX-XX-XX, where NNN are your initials and XXXX-XX-XX is today's date in
#four digit year, two digit month and two digit day respectively (Wind Turbine Predictions Time_REB_2020-08-28 below)

typeWindTurbinePredTimeName = "Wind Turbine Predictions Time_REB_2020-08-28"


typeWindTurbinePredTime = SdsType(id=typeWindTurbinePredTimeName, description="A Time-Series indexed type with 3 values",
        sdsTypeCode=SdsTypeCode.Object)

doubleType = SdsType()
doubleType.Id = "doubleType"
doubleType.SdsTypeCode = SdsTypeCode.Double

timeType = SdsType()
timeType.Id = "DateTimeType"
timeType.SdsTypeCode = SdsTypeCode.DateTime

airtemperature = SdsTypeProperty()
airtemperature.Id = "Air_Temperature"
airtemperature.Name =  "Air Temperature"
airtemperature.SdsType = doubleType

windspeed = SdsTypeProperty()
windspeed.Id = "Wind_Speed"
windspeed.Name =  "Wind Speed"
windspeed.SdsType = doubleType

activepower = SdsTypeProperty()
activepower.Id = "Predicted_Active_Power"
activepower.Name =  "Predicted Active Power"
activepower.SdsType = doubleType

time = SdsTypeProperty()
time.Id = "Timestamp"
time.Name = "Timestamp"
time.SdsType = timeType
time.IsKey = True

typeWindTurbinePredTime.Properties = [airtemperature, windspeed, activepower, time]

ocsClient.Types.getOrCreateType(namespaceId, typeWindTurbinePredTime)
```

If we go to the OCS portal, we should see this new SDS Type under Data Management> Sequential Data Store> Types in the Navigation menu:



5.3.2 Creating a New SDS Stream

Video	*Before reading this section, please refer to the following course YouTube video: https://youtu.be/h3lq1XL36pk*

After the creation of a new SDS Type, the forecasted air temperature, wind speed and active power values will be stored in a new SDS Stream based on this new type.

Note	*The alternative **Wind Turbine OCS Data_RESTAPI & OMF** Notebook in the Course Materials folder creates a new SDS Stream via OMF to only contain the active power values with a PI-Float32 type.*

There are also 2 methods to create a new SDS Stream:

i. Create the new stream via the user interface
ii. Create the new stream programmatically

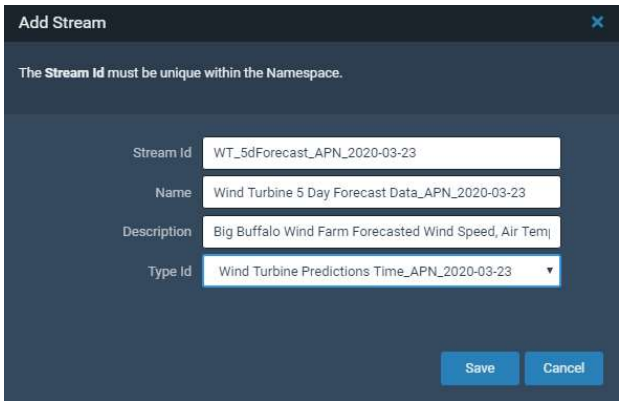
Note	*Unlike Metadata Rules, each student can create their own SDS Streams. Thus, feel free to follow the instructions below to create your own SDS Streams in OCS.*

Creating the New Stream Using the User Interface

The first step would be to navigate to the Sequential Data Store under Data Management in the Navigation menu.

The next step would be to click on Streams, then Add Stream. In the Stream Id and Name fields, enter ***WT_5dForecast_NNN_XXXX-XX-XX*** and ***Wind Turbine 5 Day Forecast Data_NNN_XXXX-XX-XX*** respectively, where NNN are your initials and XXXX-XX-XX is today's date in four digit year, two digit month and two digit day respectively.

In the Description field, enter ***Big Buffalo Wind Farm Forecasted Wind Speed, Air Temperature & Active Power for Next 5 Days***, and in the Type Id field, select the previously created type ***Wind Turbine Predictions Time_NNN_XXXX-XX-XX*** from the dropdown menu:



Creating the New Stream Programmatically using OCS Python Library

We create the new SDS Stream programmatically via the OCS Python library functions as below, assuming we already connected to OCS and authenticated our token as per the previous section. This new stream Id is ***WT_5dForecast_NNN_XXXX-XX-XX*** and the Stream Name is ***Wind Turbine 5 Day Forecast Data_NNN_XXXX-XX-XX***, where NNN are your initials and XXXX-XX-XX is today's date in four digit year, two digit month and two digit day respectively.

```
In [28]:
#SDS Stream Id = WT_5dForecast_NNN_XXXX-XX-XX, SDS Stream Name = Wind Turbine 5 Day Forecast Data_NNN_XXXX-XX-XX,
#where NNN are your initials and XXXX-XX-XX is today's date in four digit year, two digit month and two digit day
#respectively

StreamId = "WT_5dForecast_REB_2020-08-28"
StreamName = "Wind Turbine 5 Day Forecast Data_REB_2020-08-28"
StreamDescr = "Big Buffalo Wind Farm Forecasted Wind Speed, Air Temperature & Active Power for Next 5 Days"
```

```
In [29]:
stream = SdsStream()
stream.Id = StreamId
stream.Name = StreamName
stream.Description = StreamDescr
stream.TypeId = typeWindTurbinePredTime.Id

ocsClient.Streams.createOrUpdateStream(namespaceId, stream)
```

If we go to the OCS portal, we should see this new SDS Stream under Data Management> Sequential Data Store> Streams in the Navigation menu:

5.3.3 Sending Data to New SDS Stream

Video	*Before reading this section & Section 5.3.4, please refer to the following course YouTube video: https://youtu.be/Wf8z4fNBPb8*

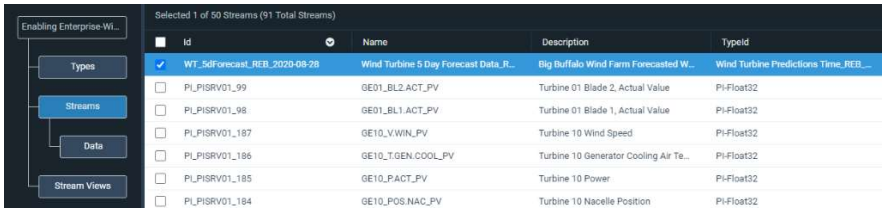
Once this new data stream is created, either using the OCS User Interface or programmatically, the data values can then be sent to this new stream. This is done by looping through the dfWeatherForecast Data Frame's rows and using an OCS Python library function as per below:

```
In [30]:
#Prepare the newly created OCS data stream values by looping through the dfWeatherForecast Data Frame's rows

values = []

for index, row in dfWeatherForecast.iterrows():
        values.append({"Air_Temperature": row["Temp (F)"], "Wind_Speed": row["Wind Speed (m/s)"],
                       "Predicted_Active_Power": row["Predicted Active Power (kW)"], "Timestamp":
                       row["Timestamp"].strftime('%Y-%m-%d %H:%M:%S') })

ocsClient.Streams.insertValues(namespaceId, stream.Id, json.dumps(values))
```

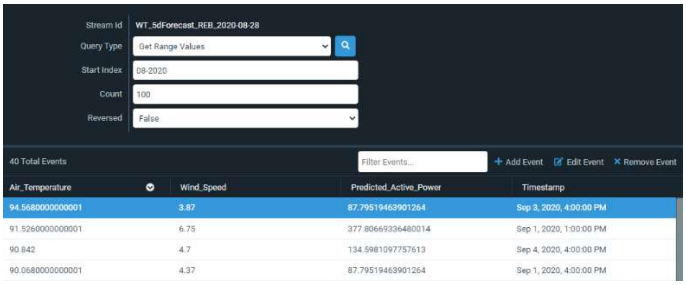
Note	*The alternative **Wind Turbine OCS Data_RESTAPI & OMF** Notebook in the Course Materials folder sends data via OMF. Since OMF needs all the fields and values as strings, the data contained in the Data Frame as DateTime and Float types need to be converted to strings and must be comma-separated.* *Then, a new values message would need to be created and Predicted Active Power data containing timestamps and values posted using the requests.post method*

5.3.4 Analyzing the New SDS Stream Data in OCS

The newly created SDS stream and data is now available to be viewed in OCS. Navigate to the Sequential Data Store under Data Management>Streams and select the newly created SDS Stream:

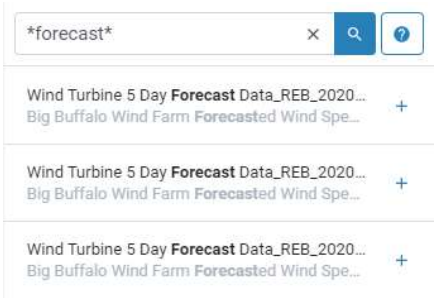


On the Data page under Streams, the values can be viewed by selecting Get Range Values under Query Type, a 08-2020 Start Index and finally clicking on the magnifying glass icon:



A graph can be viewed under the Details tab on the right, where the Y-axis values can be selected from a dropdown list of Air Temperature, Wind Speed and Predicted Active Power data:

Alternatively, this data stream can also be trended in the Visualization section of OCS by entering *Forecast* in the search field, clicking the magnifying glass icon, and selecting these forecasted Air Temperature, Wind Speed and Predicted Active Power values:

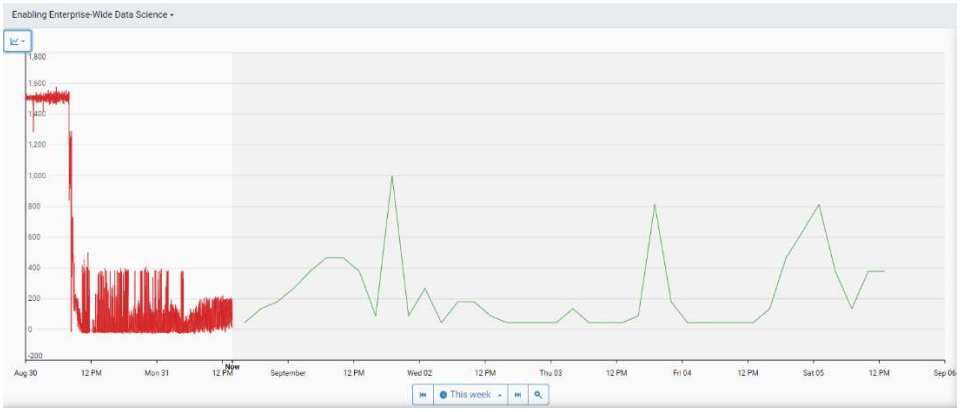


By selecting the appropriate time range, i.e. This Week, we can see trends of this forecasted data:

Name	Timestamp	Value	UOM	Min	Max	Avg
Wind Turbine 5 Day Forecast Data_REB_2020-08-28 | Air Temperature	9/5/2020 1:00:00 PM	88.376		62.924	94.568	76.516
Wind Turbine 5 Day Forecast Data_REB_2020-08-28 | Wind Speed	9/5/2020 1:00:00 PM	6.520		0.750	10.070	4.390
Wind Turbine 5 Day Forecast Data_REB_2020-08-28 | Predicted Active Power	9/5/2020 1:00:00 PM	377.807		43.682	999.445	197.533

From the data stream tables and graphs, we can confirm that the forecasted data from our machine learning model has been successfully sent and stored in OCS.

We can also plot the historical Active Power values for one of the turbines, i.e. GE02 in this case, with the forecasted Active Power values:



As the real-time Active Power data is streamed to OCS, we can later compare how the actual values match up with the forecasted data.

Congratulations! With this, you have successfully completed this course!

Video	*To conclude this course, please refer to the following course YouTube video: https://youtu.be/WBnygON2IBA*

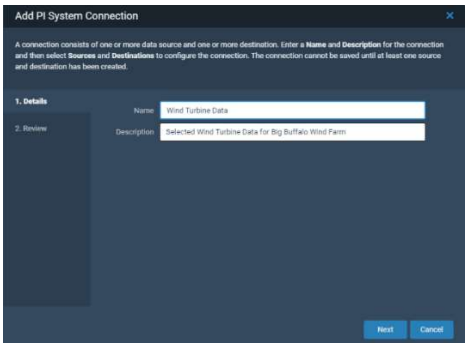
6. Appendix A: Streaming Data to OCS from PI Server

6.1 PI System Connections

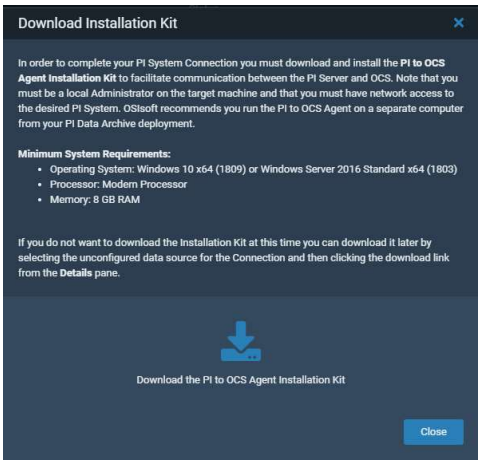
You can establish high-throughput data connections from a PI system into OCS with the Connections feature.

Select PI System from the Type drop-down list in Connections and designate a namespace as the intended destination for the data connection by following the steps below:

1. Select from the Namespace drop-down list the namespace into which you want to transfer PI system data. In this case, it is ***Enabling Enterprise-Wide Data Science***.
2. Create a PI System connection by selecting PI System from the Type drop-down list and clicking Add Connection in the Connections page.
3. In the Add PI System Connection dialog, type the name (***Wind Turbine Data***) and Description (***Selected Wind Turbine Data for Big Buffalo Wind Farm***) for the new connection into the Name and Description fields, and click Next.



4. Verify that the assigned name and namespace are correct, and then click Save. A link to download the PI to OCS Agent Installation Kit displays.



5. Click the Download Install Kit link. Refer to the Getting Started Guide link in the Details tab for installation instructions.

To edit an existing connection, complete the following:

1. Select an existing connection in the Connections page and click Edit Connection.
2. In the Edit dialog, make any changes to the name and description of the connection in the Name and Description entry fields, and then click Next.
3. In the Review panel of the Edit dialog, click Save.

To delete an existing connection, complete the following:

1. Select an existing connection in the Connections page and click Remove Connection.

   Note: To remove a connection, you must first stop any data transfers that are currently running.

6.2 Installing the PI to OCS Agent

PI to OCS enables data transfer from your on-premises PI Server(s) to OCS. The first release of PI to OCS supports the following features:

- Transfer of a selection of PI Data Archive PI Points from PI Servers to OCS Sequential Data Store (SDS) streams. Some of the PI Point attributes information is transferred as SDS stream metadata and properties.
- Simultaneous transfer of both historical and streaming data from PI Data Archive(s) to SDS for a selection of PI Points
- Configuration of a PI Points selection and data transfer management via the OCS Customer Portal.

PI to OCS has two major components:

1. An on-premises component called the “PI to OCS Agent”, which is installed on a computer used as a bridge between the source PI Server and the OCS destination. It runs as a service and performs fast, secure data transfer.
2. A cloud component called a “PI System Connection” or “PI System Connection Data Source”, residing within OCS that receives the data from the on-premises PI to OCS Agent and stores it in SDS. Note that storage in SDS is partitioned by OCS namespace.

The deployment of a PI to OCS Agent establishes a 1 to 1 connection from a on-premises source PI Server to an OCS PI System Connection.

The PI to OCS Agent must be installed by a user who has Administrative privileges on the local computer as well as in your OCS account.

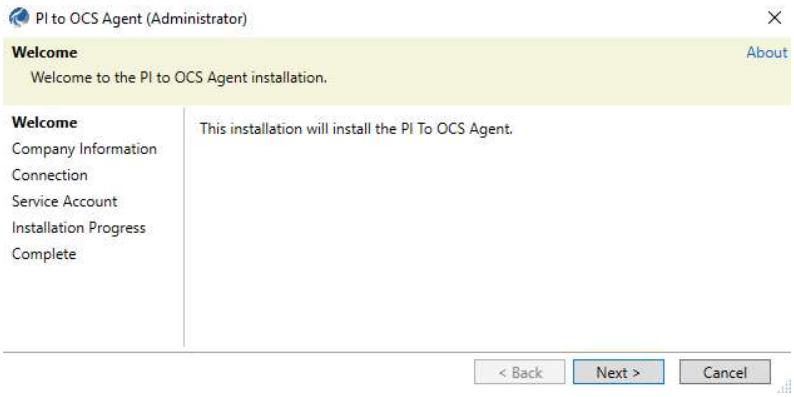
PI to OCS Agent requires the following permissions on PI Data Archive:

- Read access to Archive Data (the PIARCDATA Security Table)

- Read access to the PI Points configuration (the PIPOINT Security Table)
- Read access to all the PI Points that you are interested in transferring
- Read access to all the Data of the PI Points that you are interested in transferring

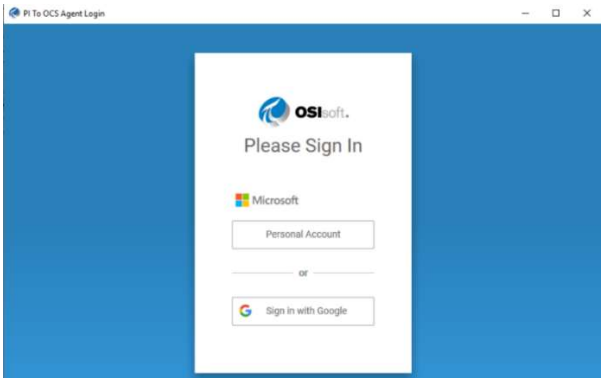
6.2.1 Step-by Step Installation Process

i. Once downloaded, double-click or "Run as Administrator the installation kit to install the PI to OCS Agent:



ii. Click Next and enter the Account Id or Company Alias. In this case, it is osisoftlearning. Click Next.

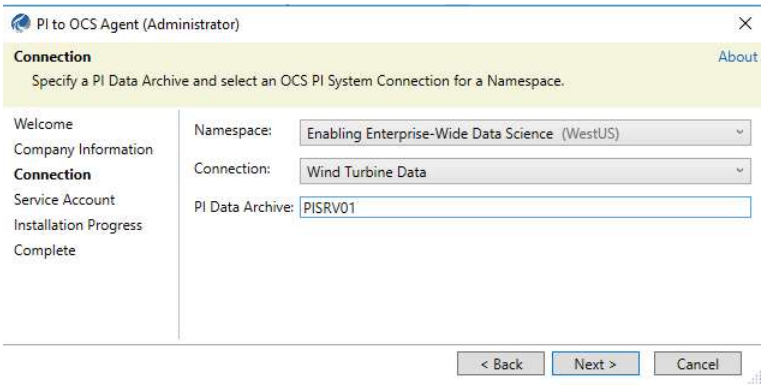
iii. A login prompt opens for you to sign in your OCS account. The user you choose must have Administrator privileges in your OCS account. The OCS account is configured to use both Microsoft Account (MSA) and Google as supported Identity Providers.



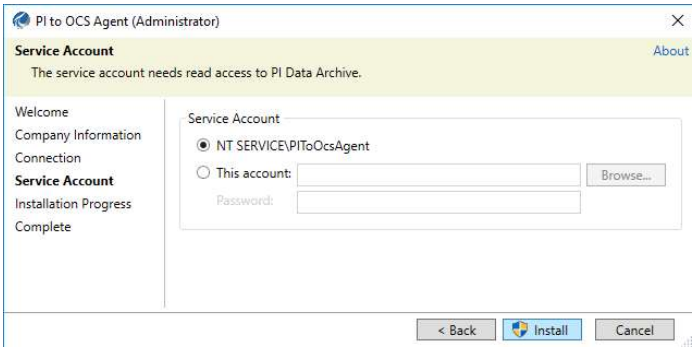
iv. Once logged in, from the top dropdown menu, select the Namespace in which you created the PI System Connection and select the corresponding Connection. In this case, the Namespace is **Enabling Enterprise-Wide Data Science** and the Connection is **Wind Turbine Data**.

The Region field next to the Namespace name indicates which region the Namespace resides in, and consequently where the data from this connection will be stored. Streaming data sent by the PI to OCS Agent will only go to the selected Namespace's region.

Next, enter the name of the PI Data Archive from which you want to transfer data from. In this case, it is **PISRV01**. Then click Next.

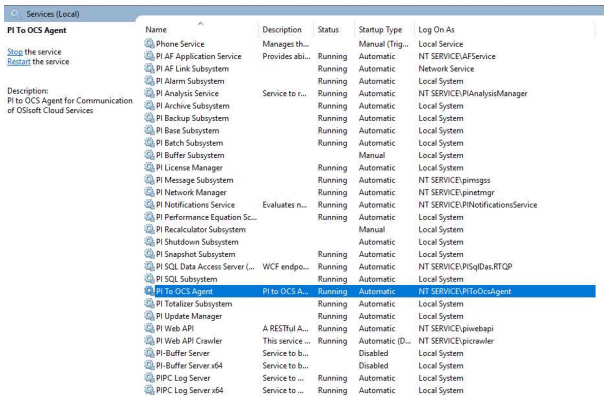


v. Enter the Service Account for the PI to OCS Agent. This account must have access to the PI Data Archive with the privileges previously mentioned. For this course, we will leave it as the default NT Service\PItoOcsAgent account. Then click on the Install button.

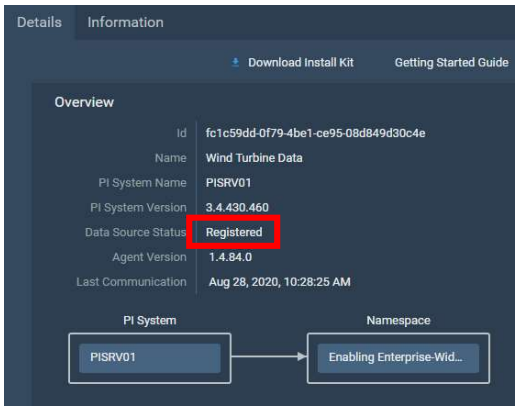


The installation should complete successfully and the PI to OCS Agent should register to the PI System Data Source in OCS. The registration process may take several minutes.

vi. Open Windows Services (services.msc) and verify that the service has been installed, started and is running with the service account provided:



vii. When you return and refresh the OCS Connections page, verify that registration has taken place by confirmation in the Details pane.



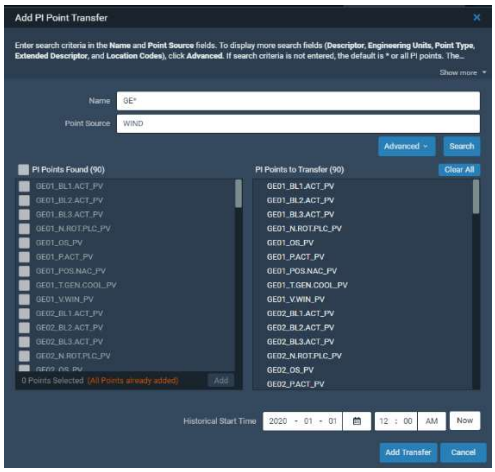
6.3 Creating a Data Transfer

Once you have installed and registered the PI to OCS Agent application, you can create a Transfer of data from the PI system to the namespace you selected when creating the connection. It can take several minutes after installing the PI to OCS Agent application for the on-premises agent to become available in the Connections page.

Note: A single instance of the PI to OCS Agent application can only support a single connection, which can only be the connection you create prior to installing the application. To create another connection (from the selected PI system to a different namespace, for example), you would have to create that connection, and then install another instance of the PI to OCS Agent application.

To transfer data with the new connection, click the Add PI Point Transfer button. The Add PI Point Transfer dialog displays. In the Add PI Point Transfer dialog, enter the Point Name Mask and Point Source Mask fields. In this case the Point Name Mask is GE* and the Point Source Mask is WIND. Click on Search. Check the PI Points Found box to select all 90 PI Points, then click the Add button.

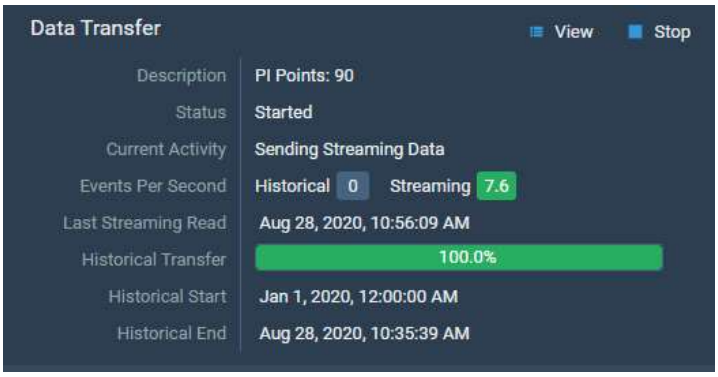
Set the Historical Start Time to 2020-01-01 at 12:00 AM. Click Add Transfer to transfer the PI Points returned in the query.



The progress of the transfer displays in the Data Transfer dialog in the Details tab.

Note: To transfer a set of PI Points different from those returned by the query, you must create a new query that groups in the desired PI Points.

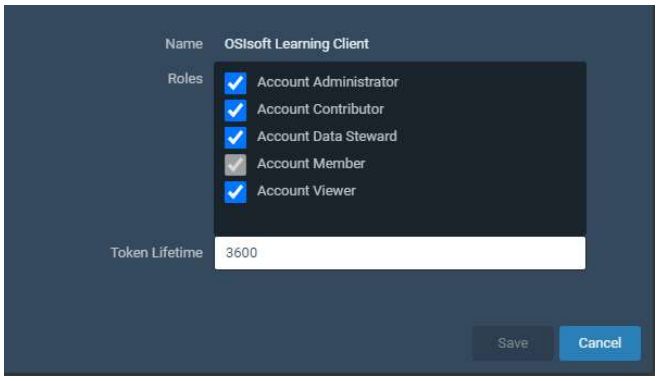
Click Start to start the Data Transfer or Stop to stop the Data Transfer. After historical data transfer has completed, current data will continue to be transferred. To create a different Data Transfer, you must first stop and remove the current Data Transfer. Note that this will not remove any data already transferred.



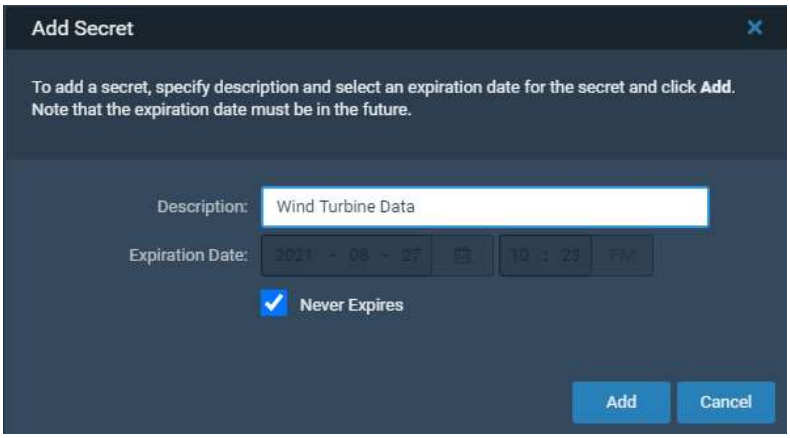
Once the historical transfer is at 100%, we can go to the Sequential Data Store under the Data Management section to view the transferred data.

7. Appendix B: Configuring a New OCS Client

On the Clients page under the Security section of the Navigation menu, click on the +Add Client button. Set the Client Type as Client-Credentials and Account. The Client Name entered is OSIsoft Learning Client, select the Account Administrator checkbox, then click Continue.



On the Add Secret window, enter the Description as below, then check the Never Expires box, then click Add:

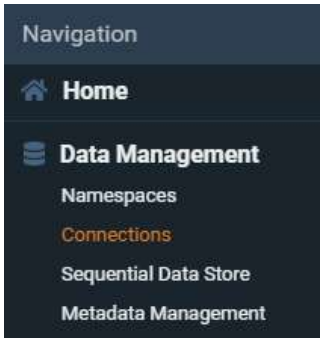


Make note of the Client Id and the Client Secret, as they are needed for accessing OCS programmatically from an external application.

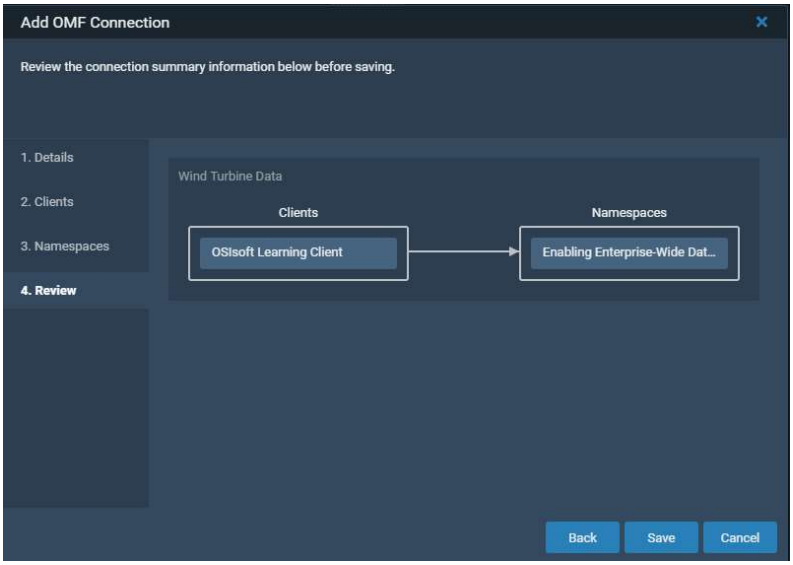
8. Appendix C: Creating an OMF Connection in OCS

Another method to send data to OCS is via the OSIsoft Message Format or OMF protocol. But first, a new OMF Connection needs to be created in OCS, so that we can use this channel to send data streams to the SDS via OMF.

Click on Connections in the Navigation menu>Data Management:



On the Connections page, select OMF as the Type and click on +Add Connection. Enter ***Wind Turbine Data*** for both the Name and Description. Click Next and select the ***OSIsoft Learning Client*** from the list. Click Next again and select the ***Enabling Enterprise-Wide Data Science*** Namespace from the list. Click Next, and the last screen should be as below:



Click Save to complete the process of creating a new OMF Connection.

For step by step details regarding creating an OMF Connection, please refer to the following OSIsoft YouTube video: https://www.youtube.com/watch?v=52lAnkGC1IM

9. Appendix D: References

For more information regarding the resources utilized in this course, please see the links below:

1. Wind Turbine Background:
    i. https://www.awea.org/wind-101/basics-of-wind-energy
    ii. Figure 1: http://www.galileoscientific.com/wind_energy.jpg 2009
    iii. Figure 2: http://www.wind-power-program.com/turbine_characteristics.htm
    iv. Figure 3: Badran, Omar & Abdulhadi, Emad & Mamlook, Rustom. (1992). Evaluation of parameters affecting wind turbine power generation.

2. OSIsoft Cloud Services: https://cloud.osisoft.com/

3. OCS API Documentation: https://ocs-docs.osisoft.com/

4. Jupyter Notebook: https://jupyter.org/

5. Python Help: https://www.python.org/

6. OCS Python Library Documentation: https://github.com/osisoft/OSI-Samples-OCS/tree/master/basic_samples/SDS/Python/SDSPy/Python3

7. Scikit-learn Python Package for Machine Learning: https://scikit-learn.org/

8. Decision-Tree Regressor Machine Learning Algorithm:
    i. https://gdcoder.com/decision-tree-regressor-explained-in-depth/
    ii. https://scikit-learn.org/stable/modules/tree.html#regression

9. OpenWeather API: https://openweathermap.org/api

10. OMF Documentation: https://omf-docs.readthedocs.io/en/v1.1/


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