# Azure Analytics / IoT – Hands-on Labs

#### Summary

In these hands-on labs, we build out an architecture to ingest, query, store and visualize asset (equipment) performance data. We start by exploring historical run-to-failure data to understand general trends/patterns – and validate that the historical data can be used to build a machine learning model for predicting future failures (i.e. remaining useful life (RUL), mean-time to failure, etc.). After the machine learning model is built, we “operationalize” the model, along with collection of data in near real-time, to monitor performance across a fleet of assets.

The labs are based on a solution template available in the Azure Gallery (<https://gallery.azure.ai/Solution/Predictive-Maintenance-10>).

Before getting started with the hands-on-labs, take some time to explore the historical data by opening the Power BI Desktop file **PredictiveMaintenance\_R\_VisualInspection.pbix** (located in your lab documents).

## Hands-On Lab 1 – Build a Regression Model

#### Summary

In this lab, we recreate a portion of the sample Azure template experiment **Predictive Maintenance: Step 2A of 3, train and evaluate regression models.** This experiment predicts time to failure i.e. how many more cycles a given piece of equipment will run before failing.

#### Before you get started

Open the Azure Machine Learning Studio in your browser - <https://studio.azureml.net>. Use your organizational credentials to login (alternatively, you can use a Windows account to sign up free in Azure ML studio).

#### Background

This lab is based on the Azure Predictive Maintenance Template (<https://gallery.azure.ai/Collection/Predictive-Maintenance-Template-3>), a set of 7 experiments the demonstrate how to build and deploy predictive maintenance models to predict asset failures. Failure of an asset can be modeled in several different ways, including binary classification (e.g., is this equipment likely to fail, yes or no), multi-class classification (e.g., will it fail within 0-10 cycles, 11-20 cycles, or 20+ cycles), and regression (i.e. how many more cycles will it run before it fails). In this lab, we focus on regression techniques.

|  |  |
| --- | --- |
| Steps | Directions/Notes |
| 1. In Azure Machine Learning Studio, locate the **+NEW** link (located in the bottom land-hand side of the browser). Click this link, and then select **Experiment-Blank Experiment**. This will create a blank experiment with a generic name (e.g., *Experiment created on 5/10/2018*); left-click anywhere on the experiment name, and rename your experiment to something useful (e.g., *Hand On Lab1 – Train Regression Model*) 2. Using the Search experiment items textbox, enter the word **Import** to locate the **Import Data** module. Double-click, or drag and drop this module onto the experiment canvas, and in the Properties section, set the following properties -  * Data Source - **Web URL via HTTP.** * URL - <http://azuremlsamples.azureml.net/templatedata/PM_step1output_train.csv> * Data format – CSV * Check the **CSV or TSV had headers** checkbox * Check the **Use cached results** checkbox |  |
| 1. Run your experiment by selecting the **RUN-Run** link near the bottom of the page). The experiment should finish running within a few seconds. 2. Next, hover over and then click the small circle at the bottom of the **Import Data** module; select the **Visualize** menu item. This will bring up a dataset visualization window. After looking at the dataset, you can close this window.   **Note** – From within this window, you can click on a column to generate a simple set of statistics, along with a histogram. |  |
| 1. Add a **Select Columns in Dataset** module to your experiment (place it below **Import Data)**- and then connect (i.e. link) it to the **Import Data** module (see screen shot below). Linking is done by left-clicking on the small circle at the bottom of **Import Data** – and dragging/dropping the line to the small circle at the top of **Select Columns in Dataset**. 2. Select (i.e. click) the **Select Columns in Dataset** module, and then launch the column selector from the Properties Pane. Set the properties as shown in the screenshot to the right.   **Note** – Project Columns let you exclude columns that are not useful for building a model (e.g., a unique identifier). In our example, we are removing label2 and label2; they are used for classification/multi-class classification. |  |
| 1. Next, add a **Filter Based Feature Selection** module. Set the properties as follows:   **Note**: Filter Based Feature Selection isn’t a necessary step, but it can cut down on the time to run an experiment by weeding out features will little correlation – and helps us understand (at a high level) which features may have an impact on the dependent variable. |  |
| 1. Run the experiment again; select the **Filter Based Feature Selection** module, and then select the **RUN-Run Selected** item. 2. After the experiment finishes, click on the circle at the bottom right-hand side of **Filter Based Feature Selection,** and then select the Visualize menu item. You will see a result set similar to the screenshot on the right.   In this example, RUL and a4 have a correlation value of .732976. Note this is an absolute value, as these two arrays are actually negatively correlated.  Tip – To calculate the Pearson correlation value in Excel, you can use either the CORREL or PEARSON function (which takes 2 arrays as arguments). I’ve shown an example of this in the PM\_step1output\_train\_CorrelationsExample.xlsx workbook. |  |
| 1. Add another **Import Data** module to pull in the testing data (you don’t need to connect this module to the other modules; we will use this a bit later).   URL - http://azuremlsamples.azureml.net/templatedata/PM\_step1output\_test.csv |  |
| 1. Now we are ready to train a regression model. We are going to add 2 modules - 2. **Decision Forrest Regression –** we will not modify any of the property default values 3. **Train Model** – After hooking up the modules as shown to the right, push the launch column selector button, and select the RUL column. This specifies that RUL (remaining useful life) is the value to predict.   Tip: this layout may appear a bit confusing; think of Train Model as a generic component that accepts two modules as parameters – a specific model (e.g., Decision Forest Regression) and a training dataset. |  |
| 1. Now add modules to score/evaluate a model. Add a **Score Model** and **Evaluate Model** module to the experiment (as shown in the screenshot to the right)   **Note** - For the Score Model module, connect the Test Data (not the Training Data).   1. Run the experiment. After it finishes, click the bottom circle of the Score Model to Visualize the predicted results. In the Scored dataset window, find and click the RUL column; then, in the Visualizations area of the window, select **Scored Label Mean** in the **compare to** dropdown box. You will see a scatter chart that compares the actual (RUL) to the predicted value. Close this window when you are done. |  |
| 1. Click the bottom circle of the **Evaluate Model** module (and select **Visualize**) to see a set of aggregate evaluation metrics (we will discuss and look at these metrics in our next lab). |  |
| We will stop here – and switch over to the completed sample experiment (Predictive Maintenance: Step 2A of 3, train and evaluate regression models). If you haven’t already done so, import this experiment (from the Gallery) and Run it.  Tip – To import the experiment, click the **+NEW** link (located in the bottom land-hand side of the browser); search for **Predictive Maintenance** in the Search textbox. Find the Sample named **Predictive Maintenance: Step 2A of 3, train and evaluate regression**, and click the OPEN IN STUDIO button.  After running the experiment, take note of a few important points –   * The Evaluate model can be used to score/evaluate one or two models (in this experiment, there are 4 regression models, and hence 2 Evaluate Model modules). * Not all regression models return the same metrics. This experiment uses an R Script module to combine all results together (shown in the screenshot to the right). |  |

## Hands-On Lab 2 – Evaluate Regression Metrics (Results)

#### Summary

In this lab, we are going to take a brief look at an Excel workbook that will help us better understand the five metrics shared across all 4 regression algorithms (Mean Absolute Error, Root Mean Squared Error, Relative Absolute Error, Relative Squared Error, and the Coefficient of Determination (aka R², pronounced “R Squared”)

#### Before you get started

Open the workbook Demand\_Metrics\_Excel\_Template\_Tyler.xlsx in Excel.

|  |  |
| --- | --- |
| Step | Directions/Notes |
| 1. In the Demand\_Metrics\_Excel\_Template\_Tyler.xlsx workbook, navigate to the **Data** worksheet.   Note: I found this workbook from an online search. Most of the worksheets have not been changed. I did add an additional forecast (Forecast2) to the Data worksheet – along with the “Actual and Forecast” Chart - and additional cells (Accuracy Method). Some of what I added in the Accuracy Method area can also be found in the Demand Metrics worksheet – I’ve tried to simplify things a bit to match the metrics available in Azure ML. |  |
| 1. Let’s look at the first metric – Mean Absolute Error (MAE). This metric takes the average (mean) of the forecast errors (expressed as positive i.e. absolute values). A forecast “error” sometimes called a residual) is simply the difference between the actual and forecasted value. |  |
| 1. The second metric, Root Mean Squared Error, is similar to MAE, but instead it squares each forecast error, computes an average of these squared values, and takes the square root of the average. The net result is that larger errors receive a higher “weight” in the final “score”. |  |
| 1. The third metric, Relative Absolute Error, is a bit more complex. It involved the following steps:  * Absolute Deviation (from Mean Demand) – using the average of all actuals, it calculates (for each row) the absolute value of (actual – actuals average) * Cumulative Absolute Deviation – a running (cumulative) total of the prior step for each row. * Cumulative Abs Forecast. Error – a running total of the absolute value of the forecast error for each row * Relative Absolute Error - Cumulative Abs Forecast. Error / Cumulative Absolute Deviation. Calculated for each row |  |
| 1. The 4th metric, Relative Squared Error is discussed here - <http://www.saedsayad.com/model_evaluation_r.htm> |  |
| 1. The final metric, Coefficient of Determination, is also referred to a R² (R Square) or the least squared. R² is the square root of r – which is the correlation coefficient (i.e. the Pearson correlation). R² can be calculated as:   1 – ( Sum of Forecast Errors squared / Sum of Squared Deviations from the Mean Squared)  As notation….  1 – ( ² / )  For another way to understand R², see my time series forecasting article - <http://sqlmag.com/sql-server-analysis-services/understanding-time-series-forecasting-concepts>. | See the Demand Metrics worksheet for the R² calculation |
| Decent (easy to understand) discussion on common forecast evaluation methods. <http://www.forecastpro.com/Trends/forecasting101August2011.html> |  |

## Hands-On Lab 3 – Deploy Web Service with a Regression Model

#### Summary

In this lab, we are going deploy the regression model (Step 2A) as a web service

#### Before you get started

Open up the Machine Learning Studio in your browser - <https://studio.azureml.net>.

|  |  |
| --- | --- |
| Step | Directions/Notes |
| 1. In Azure ML Studio, open up the experiment Predictive Maintenance: Step 2A of 3, train and evaluate regression models; Click the **Save-Save As** button to create a copy of this experiment.  * For our Web Service, we will use the Boosted Decision Tree Regression Model – **remove the steps associated with other three models**. Your experiment should look similar to what is shown. * Run your experiment. |  |
| 1. Click the **Set Up Web Service – Predictive Web Service (recommended)** button. Click the Next buttons (several times) to see the Predictive Experiment. Note the steps Azure ML does on our behalf -  * A copy of the experiment is made (same name – ending with [Scoring Exp.]   + A Tabbed view is created (allowing you to toggle between the training and predictive experiment * The Boosted Decision Tree Regression model is saved as a Trained Model (it is given the same name as the experiment) – in the predictive   + The model now shows up under **Trained Models –**you can drag/drop into new experiments.   + Note – you can manually save a Model by right-clicking the bottom node of a model (make sure you’ve first run the experiment) and selecting the **Save as Trained Model** menu-item. With this approach, you can customize the name. You can also re-save a model if you perform further training * A Web service input is created   + Notice that the Web Service input is hooked directly to the Score Model – it therefore has the same schema as the testing data. * A Web service output is created. Output includes the original inputs and a scored value. |  |
| 1. Click the Web Services link in the browser (the Globe) – you will now see a Web Service.   You can click the Web Service to bring up additional information about the service. From the browser, you can also test the service interactively – or you can download a sample Workbook that will call the service. |  |

## Hands-On Lab 4 – Configure Event Hubs

#### Summary

In this lab, we are going create/configure Azure Event Hubs – which will be used to store events coming from a data simulator. High-level information about Event Hubs can be found in the Azure documentation - <https://docs.microsoft.com/en-us/azure/event-hubs/event-hubs-create>

#### Before you get started

Navigate and login to the Azure Portal (portal.azure.com). Note - you will need to have the appropriate permissions to create resources in the Azure subscription/resource group. If you are not an administrator of the subscription, see here for more information - <https://docs.microsoft.com/en-us/azure/role-based-access-control/role-assignments-portal> - and then work with administrator to be granted permissions.

|  |  |
| --- | --- |
| Step | Directions/Notes |
| 1. From the Azure portal, click **Create a resource** as the top left of the screen. 2. Using the *Search the Marketplace* textbox, enter **Event Hubs** and hit the Enter key. Alternatively, select **Internet of Things**, and then select Event Hubs. 3. Create your Event Hub namespace using the screenshot to the right as a guide. Note the following -  * **Name** must be unique (e.g., append your initials to the name) * Use the default **Standard** Pricing Tier * If you have already configured a Resource group (or been given permissions to a particular Resource group), select **Use existing**; otherwise, create a new Resource Group. * For Location, select East US (unless your organization and/or existing Resource Group uses a different Location) * Unselect the **Enable auto-inflate?** checkbox; our demo will not exceed the default capacity of 1 Throughput Unit.  1. Push the **Create** button to save your changes. 2. After the resource is provisioned (it could take ~1-2 minutes), locate and click the newly created namespace. |  |
| 1. In the Events Hubs Namespace window, click **Shared access policies**, and then click **RootManageSharedAccessKey** 2. Copy the Connection-string-primary key. We will need it later for the data simulator. |  |
| 1. Next, click **Event Hubs**, and then click the **+ Event Hub** lilnk. Use the name **azurehol** – don’t change any of the other settings. 2. Click the **Create** button. |  |
| 1. The Event Hub is now ready to receive data; we will test it with the data simulator. Locate the PMSimulator folder in your lab materials, and open the **Setting.json** file in an editor (Notepad will work fine). 2. Edit the **EventHubConnectionString** value – replace it with the connection string that you copied above. Replace **EventHubName** with the name of your Event Hub from above (**azurehol**). 3. Now, run PMSimulator.exe (you can do this from your local machine); click the **Start Ingestion** button. Within a few seconds, you should see events being uploaded (in the Windows application), and when you look at the Overview tab (of the Event Hub in the Azure Portal), you should see Requests being processed 4. Click the **Stop Ingestion** button; we will use the simulator again in an upcoming lab. |  |

## Hands-On Lab 5 – Configure Stream Analytics

#### Summary

In this lab, we are going create/configure Azure Stream Analytics – which will be used to query events coming from Event Hubs. Using the windowing functions available in Stream Analytics, we will push raw data and alerts to a Power BI Dashboard.

#### Before you get started

Navigate and login to the Azure Portal (portal.azure.com). Note - you will need to have the appropriate permissions to create resources in the Azure subscription/resource group. If you are not an administrator of the subscription, see here for more information - <https://docs.microsoft.com/en-us/azure/role-based-access-control/role-assignments-portal> - and then work with administrator to be granted permissions.

|  |  |
| --- | --- |
| Step | Directions/Notes |
| 1. Part of this lab will require setting up a connection to Power BI. If you haven’t already done so, work with your instructor to login to the Power BI Service. |  |
| 1. From the Azure portal, click **Create a resource** as the top left of the screen. 2. Using the *Search the Marketplace* textbox, enter **Stream Analytics** and hit the Enter key. Alternatively, select **Internet of Things**, and then select **Stream Analytics job**. 3. Create your Stream Analytics job using the screenshot to the right as a guide. Note the following -  * **Name** – sa\_azurehol * Resource group – use existing, and select the resource group used for the Event Hub * Hosting environment – use the default (Cloud) * Streaming units – use the default (1)  1. Push the **Create** button to save your changes. 2. After the resource is provisioned (it could take ~1-2 minutes), locate and click the newly created job |  |
| 1. We are now going to define Inputs and Outputs for the job; start with Inputs by clicking the **Inputs** link, and then clicking **+Add stream input – Event Hub.** Configure the input as shown in the screenshot on the right (replacing any names as needed to match your environment). Make sure you change Event serialization format to **CSV**.      1. Tip – if you need to generate sample data, you can click the **Sample data** link (from the InputHub); this requires that you have data in the Event Hub. We will use sample data (in your lab materials) that was captured from this link when we design our queries. |  |
| 1. Next, we are going to add a reference input that can be joined to the streaming data from Event Hubs. Note – the file, Engine\_Master.csv, may have already been uploaded to an Azure Blob storage account; check with your instructor. Click on **+Add stream input – Blob Storage.** Name the input **enginemaster**, and then fill out the details to match the screen shot on the right, adjusting the Storage Account and Container as needed.   **Tip** – don’t forget to change the Event serialization format to **CSV**!  Note – The reference file, Engine\_Master.csv, contains the manufacturer, Model name, and PeriodsInService indicator for each engine – as shown below.    When you finish, you will have two Inputs defined – |  |
| 1. Next, we will define an Output; click the **Outputs** link, and then click **+Add – Power BI.** Give this output the name **OutputPowerBI** – and a Dataset/Table name as shown to the right. 2. Authorize to your Power BI account, choose a Group Workspace (you can use My Workspace if desired) and then Save your changes. |  |
| 1. Now add another Power BI Output, named **HPCPressureOutputAlert**; configure as shown to the right (select the same Group workspace that was used in the prior step).   After saving your changes, you will now have two Outputs defined. |  |
| 1. Almost there! We can now define a query. Click the **Query** Link; a blank Query window will open as show to the right |  |
| 1. Replace the query template with the following two queries 2. Next we prepare to test the query by uploading sample data. In the Query Editor, click on the Ellipsis (…) next to the **enginemaster** Input. Select the **Upload sample data from file** menu item      1. Find and select the **Engine\_Master2.json** file located in your lab documents. Click OK. 2. Now, repeat the process of uploading sample data for the **InputHub** input; select the **sa\_azurehol-InputHub.json** file located in your lab documents.   When you have completed these steps, you should see a file icon located next to both Inputs. | -- Query 1. Push detailed data (aggregated at a 1-second grain) to Power BI. Join to enginemaster  SELECT  MAX(CAST(IH.processed as Datetime)) as processed,  IH.id, EM.Model,  IH.cycle,  IH.counter,  IH.endofcycle,  AVG(CAST(IH.s9 as float)) as s9,  AVG(CAST(IH.s11 as float)) as s11,  AVG(CAST(IH.s14 as float)) as s14,  AVG(CAST(IH.s15 as float)) as s15  INTO  OutputPowerBI  FROM  InputHub IH INNER JOIN enginemaster EM ON IH.id = EM.Id  GROUP BY IH.processed,  IH.id, EM.Model, IH.cycle,IH.counter,IH.endofcycle,  TumblingWindow(second,1);    -- Query 2. Count of rows where the avg s11 value exceeds the alert value. This query is run every 5 seconds  SELECT id,max(CAST(processed as Datetime)) as processed,avg(s11) as avg\_s11,count(\*) as alerts  INTO HPCPressureOutputAlert  FROM InputHub  GROUP BY id,TumblingWindow(second,5) HAVING avg\_s11 > 48.26; |
| 1. Click on the Test button to validate the queries; results will be returned for the two queries.   Note that no rows will be returned for the 2nd query, HPCPressoureOutputAlert, but you should see results for the first query (OutputPowerBI) – as shown in the screenshot to the right. |  |
| 1. Azure Stream Analytics is now successfully configured. We will go ahead and start the job so that is will pick up events flowing into Event Hubs (note – at this point, the data simulator isn’t running, we will do this in the next lab). 2. Click the **Overview** link for the stream analytic job, and click the **Start** link; keep the default start time of **Now**, and click the **Start** button. In ~1-2 minutes, the job will show a status of running. |  |

## Hands-On Lab 6 – Visualize Streaming Data with Power BI

#### Summary

In this lab, we will use Power BI to visualize streaming data (coming from Stream Analytics) – and combine it with 1) historical data, and 2) RUL predictions onto a single dashboard.

#### Before you get started

Make sure you have completed all prior labs. Your Azure Stream Analytics job should be running, and you should have already configured/tested the data simulator which pushes data into Event Hubs.

|  |  |
| --- | --- |
| Step | Directions/Notes |
| 1. Start by uploading the Power BI Desktop file (located in your lab documents), **PredictiveMaintenance\_MonitoringPrediction.pbix**, into the same PowerBI workspace where your stream analytics jobs will be outputting data. You can upload the document either directly from the Power BI Desktop (click the Publish button) or from the service itself (i.e. in a browser, sign into Power BI, navigate to the applicable workspace, and then click the **Get Data** link –and then select **Files-Local File**).   After the file is uploaded, you should see a dataset, report, and dashboard in the workspace. You should not yet see the streaming datasets in the workspace (as the data simulator isn’t running).  **Important Note** – In addition to the original historical dataset, this workbook contains a large number of predictions that were generated by…   * Persisting the streaming data into storage * Periodically calling the Azure Machine Learning Web Service (i.e. our trained regression model) to generate RUL predictions for each asset * Persisting the predictions back into storage - and then to an Azure SQL Database   In other words, this workbook is the result of the “cold path” part of the reference architecture discussed/presented at the beginning of the workshop. We didn’t implement this cold path during our hands-on labs; instead, we are using this workbook to simulate the cold path. |  |
| 1. Open the PMSimulator.exe application. Click the **Start Ingestion** pushbutton. Within a few seconds (if everything is working properly), you should see a new dataset in your workspace – **Pmdemostream.** Within a few minutes, you should also see the **HPCPressureOutputAlert** dataset. If you don’t see these datasets after a few minutes, work with your instructor to troubleshoot (a good place to begin will be the Azure Stream Analytics job). |  |
| 1. Now we can begin building out our dashboard. Let’s start by creating a simple card to show the number of events flowing from stream analytics. Back in Power BI (i.e. in your browser) click the **PMdemostream** Dataset; this will bring up a blank report. 2. In the Fields area, click on the **counter** field, and then change the visualization to a **Card** (see the screenshot to the right. 3. Format the card by turning the **Category label** Off, and then creating a **Title** for the visualization. You can get to the formatting options by clicking on the Paintbrush icon in the Visualizations section of the report (if you don’t see the paintbrush icon, make sure the Card visual is selected on the report canvas) 4. Save this report (File – Save) – name the report something descriptive (e.g., PMDemoStream). 5. Next, hover over the Card visual, and click on the Pin icon (Pin visual). Select the dashboard that was created when you uploaded the **PredictiveMaintenance\_MonitoringPrediction.pbix** file, or create a new dashboard (tip - you can rename the existing dashboard to something more descriptive if desired). |  |
| 1. Navigate to your dashboard. You should see the Card visualization, and it will update automatically approximately every 1-2 seconds. |  |
| 1. We are now (as a group), going to create the remainder of our dashboard, which will look similar to the screenshot to the right. Wait for your instructor to begin building out the remainder of the dashboard! |  |
| 1. When you are finished, make sure to stop the data simulator application, and stop the Azure Stream Analytics job. |  |