Machine Learning Model

CAPSTONE PROJECT | ASSIGNMENT 6

In this part of the capstone project we will build a machine learning model that uses the metrics we designed to produce risk of failure predictions for our lightbulbs. To do so, we will use Jupyter Notebook within C3 Tools.

Machine Learning in the C3 Platform offers up several major benefits to end users:

- It brings ML and App Development closer together by accessing the same underlying datasets and processing frameworks.
- It speeds up ML development work and reduces the need to understand underlying databases, write SQL / CQL queries, or worry about underlying processing infrastructure.
- It accelerates ML model deployment into production enterprise applications and reduces maintenance / speeds up upgrade cycles.

Let’s jump right in!

1. STORING LIGHT BULB FAILURE PREDICTIONS

Smart Bulb Prediction

In order to be able to use the machine learning that we are about to create, we need to create a type, `SmartBulbPrediction`, where we will store our light bulb failure predictions.

We are planning to create failure predictions for every timestamp, thus creating a significantly large dataset. Let us therefore store the data in Cassandra:

```
/**
 * A prediction made for a single {@link SmartBulb}.
 */
@db(compactType=true,
   datastore='cassandra',
   partitionKeyField='smartBulb',
   persistenceOrder='timestamp',
   persistDuplicates=false,
   shortId=true,
   shortIdReservationRange=100000)
entity type SmartBulbPrediction schema name "SMRT_BLB_PRDCTN" {
   /**
    * The calculated risk score for this SmartBulb
    */
   prediction: double

   /**
    * The {@link SmartBulb} for which this prediction was made.
    */
   smartBulb: SmartBulb

   /**
    * The time at which this prediction was made.
    */
   timestamp: datetime
}
```

Smart Bulb

We want to be able to access the predictions on our `SmartBulb` type.

For that purpose, we need to add the following two fields to `SmartBulb`:

```
/**
 * This bulb's historical predictions.
 */
@db(order='descending(timestamp)')
bulbPredictions: [SmartBulbPrediction](smartBulb)
```

```
/**
 * This bulb's latest prediction.
 */
currentPrediction: SmartBulbPrediction stored calc "bulbPredictions[0]"
```

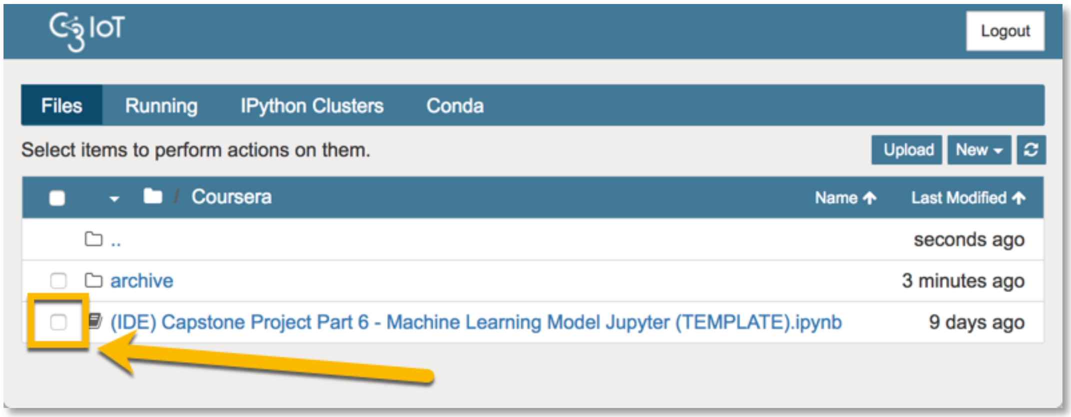
2. WORKING WITH DATA FROM C3 IOT

Now that we have created a type to store our light bulb failure predictions and are able to access these predictions via the `SmartBulb` type, we can start working on our Machine Learning model.

Note: The following 5 steps use an existing Jupyter Notebook file as a template. You may have to create your own in your Notebook in your environment.

Create your own Jupyter Notebook workspace by completing the following steps:

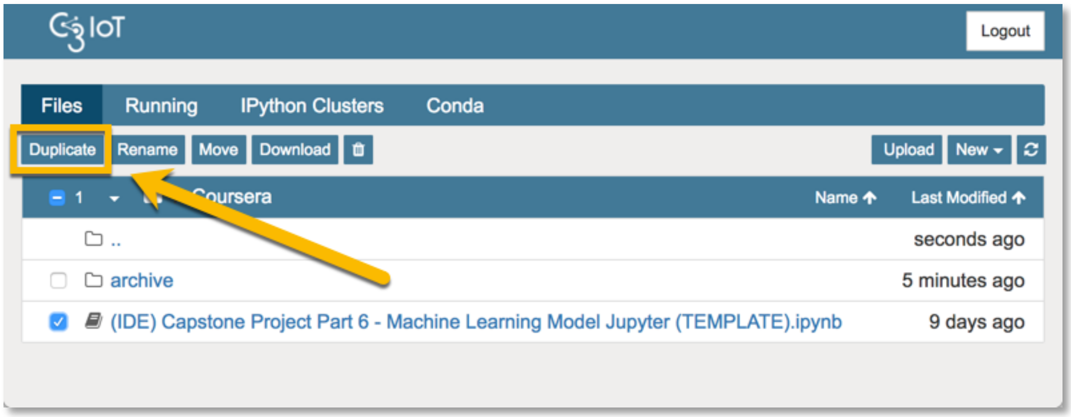
1. Visit your Juptyer Notebook
2. Select the check box to the left of the file: “Capstone Project Part 6 – Machine Learning Model Jupyter (TEMPLATE).ipynb”.



Select items to perform actions on them.

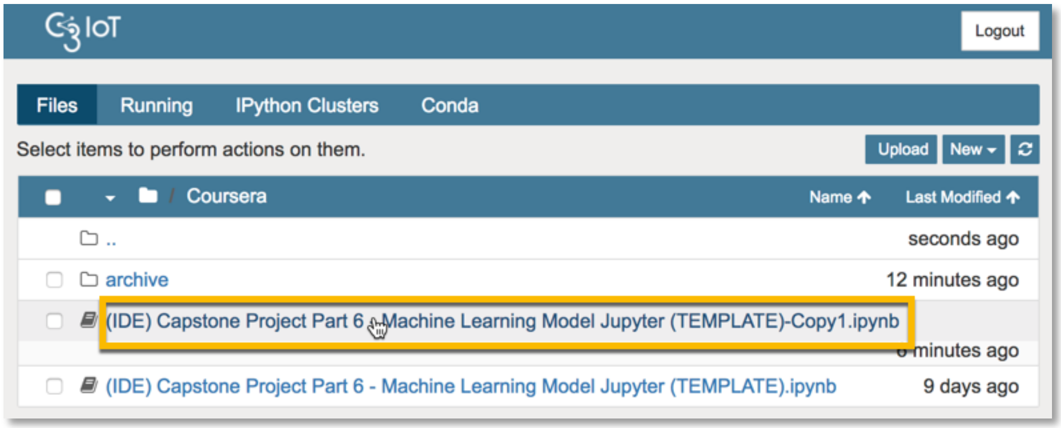
	Coursera	
	..	seconds ago
□	archive	3 minutes ago
□	(IDE) Capstone Project Part 6 - Machine Learning Model Jupyter (TEMPLATE).ipynb	9 days ago

3. Click the Duplicate button at the top, then click Duplicate in the next modal to create a copy



1	Coursera	
	..	seconds ago
□	archive	5 minutes ago
☑	(IDE) Capstone Project Part 6 - Machine Learning Model Jupyter (TEMPLATE).ipynb	9 days ago

4. Click the newly created notebook title to enter

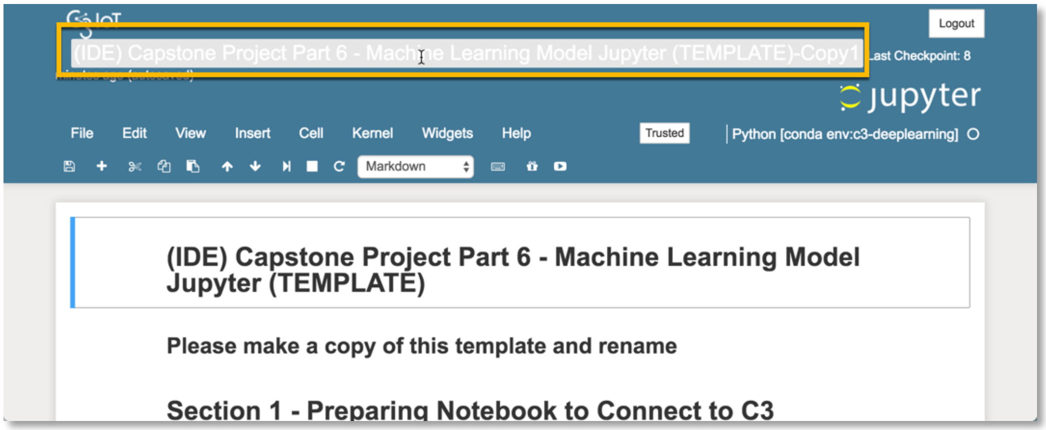


Select items to perform actions on them.

Coursera

..	seconds ago
archive	12 minutes ago
(IDE) Capstone Project Part 6 - Machine Learning Model Jupyter (TEMPLATE)-Copy1.ipynb	6 minutes ago
(IDE) Capstone Project Part 6 - Machine Learning Model Jupyter (TEMPLATE).ipynb	9 days ago

5. Rename your new copy by clicking on the title of the notebook and save it



(IDE) Capstone Project Part 6 - Machine Learning Model Jupyter (TEMPLATE)-Copy1 Last Checkpoint: 8 minutes ago (autosaved)

Python [conda env:c3-deeplearning]

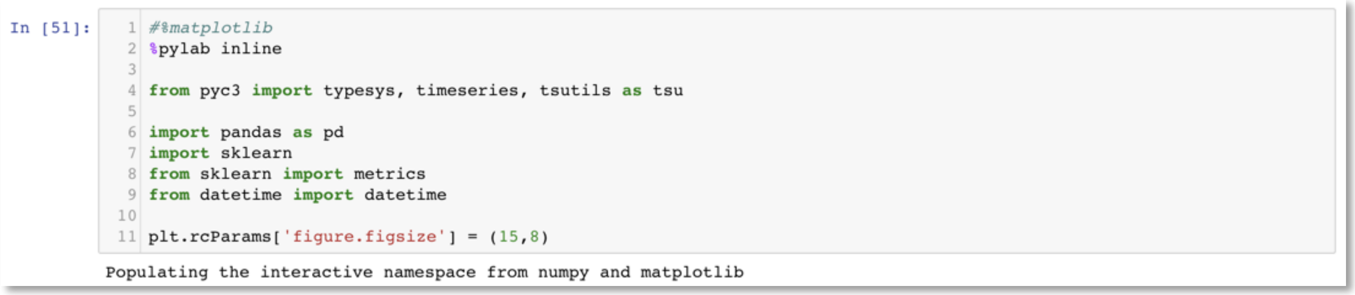
(IDE) Capstone Project Part 6 - Machine Learning Model Jupyter (TEMPLATE)

Please make a copy of this template and rename

Section 1 - Preparing Notebook to Connect to C3

Now that you are in your workspace, we need to install C3 IoT libraries that will enable us to connect to the C3 Platform, access the type system, and operate on time series data (static / interactive plotting).

To install the libraries, enter the following code in the first empty cell of your template, and click **Shift + Enter** to execute the code:



In [51]:

```
#%matplotlib
%pylab inline

from pyc3 import typesys, timeseries, tsutils as tsu

import pandas as pd
import sklearn
from sklearn import metrics
from datetime import datetime

plt.rcParams['figure.figsize'] = (15,8)
```

```
Populating the interactive namespace from numpy and matplotlib
```

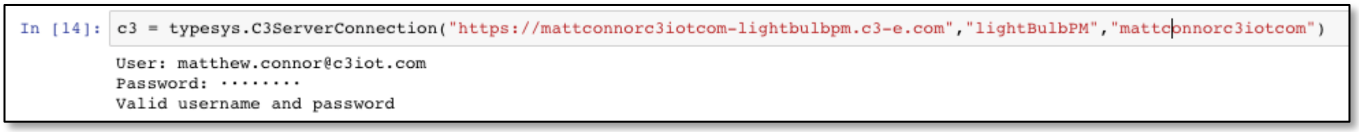
With our notebook configured, we can begin by connecting to our environment using `C3ServerConnection` as part of the `typesys` module imported from the `pyc3` library.

C3ServerConnection is the central object of Python typesystem. It is:

- A handle to a given C3 environment, able to emit REST calls and parse the answer
- A container for all type definitions
- A collection of helper functions designed to make life easier

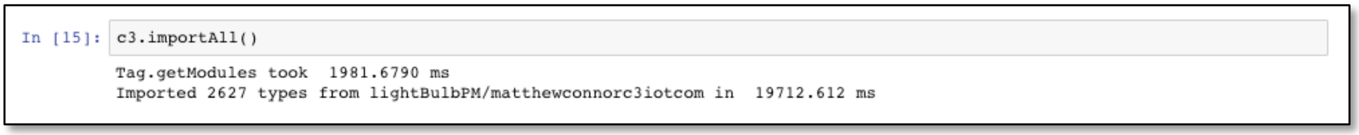
To connect to an environment, we will execute the following command in the next empty cell. An example of what the command would look like is shown in the image below.

```
c3 = typesys.C3ServerConnection("<Environment_URL>","<Tenant>","<Tag>")
```



```
In [14]: c3 = typesys.C3ServerConnection("https://mattconnorc3iotcom-lightbulbpm.c3-e.com","lightBulbPM","mattconnorc3iotcom")

         User: matthew.connor@c3iot.com
         Password: ········
         Valid username and password
```

Once you input your username and password and a connection has been created, import your types from the C3 environment with the c3ImportAll() command.



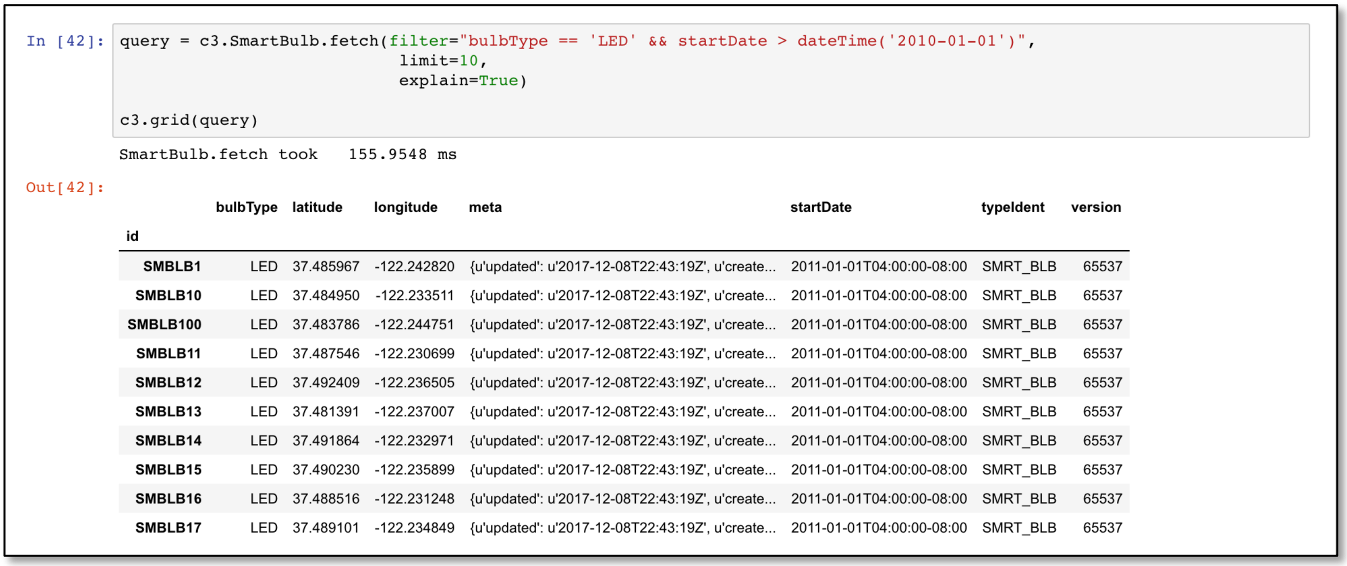
```
In [15]: c3.importAll()

         Tag.getModules took  1981.6790 ms
         Imported 2627 types from lightBulbPM/matthewconnorc3iotcom in  19712.612 ms
```

To build our ML model, we will need to fetch data from the C3 Type System and display it in a Pandas data frame.

Let’s say we wish to fetch data and see a grid view of smart bulbs. We additionally wish to filter the results such that we only fetch data for Smartbulbs where the bulb type is LED and the bulb was installed after January 1, 2010.

To do so, we invoke the C3 connection and specify the type (SmartBulb) and action (fetch()). We then provide a filter for both bulbType and startDate, and finally we use the function c3.grid() to visualize the results.

An example of the command is shown below:



```
In [42]: query = c3.SmartBulb.fetch(filter="bulbType == 'LED' && startDate > dateTime('2010-01-01')",
                                    limit=10,
                                    explain=True)

         c3.grid(query)

         SmartBulb.fetch took   155.9548 ms
Out[42]:
```

	bulbType	latitude	longitude	meta	startDate	typeIdent	version
id							
SMBLB1	LED	37.485967	-122.242820	{u'updated': u'2017-12-08T22:43:19Z', u'create...	2011-01-01T04:00:00-08:00	SMRT_BLB	65537
SMBLB10	LED	37.484950	-122.233511	{u'updated': u'2017-12-08T22:43:19Z', u'create...	2011-01-01T04:00:00-08:00	SMRT_BLB	65537
SMBLB100	LED	37.483786	-122.244751	{u'updated': u'2017-12-08T22:43:19Z', u'create...	2011-01-01T04:00:00-08:00	SMRT_BLB	65537
SMBLB11	LED	37.487546	-122.230699	{u'updated': u'2017-12-08T22:43:19Z', u'create...	2011-01-01T04:00:00-08:00	SMRT_BLB	65537
SMBLB12	LED	37.492409	-122.236505	{u'updated': u'2017-12-08T22:43:19Z', u'create...	2011-01-01T04:00:00-08:00	SMRT_BLB	65537
SMBLB13	LED	37.481391	-122.237007	{u'updated': u'2017-12-08T22:43:19Z', u'create...	2011-01-01T04:00:00-08:00	SMRT_BLB	65537
SMBLB14	LED	37.491864	-122.232971	{u'updated': u'2017-12-08T22:43:19Z', u'create...	2011-01-01T04:00:00-08:00	SMRT_BLB	65537
SMBLB15	LED	37.490230	-122.235899	{u'updated': u'2017-12-08T22:43:19Z', u'create...	2011-01-01T04:00:00-08:00	SMRT_BLB	65537
SMBLB16	LED	37.488516	-122.231248	{u'updated': u'2017-12-08T22:43:19Z', u'create...	2011-01-01T04:00:00-08:00	SMRT_BLB	65537
SMBLB17	LED	37.489101	-122.234849	{u'updated': u'2017-12-08T22:43:19Z', u'create...	2011-01-01T04:00:00-08:00	SMRT_BLB	65537

To build our machine learning model we need to select the metrics we will be using. To do so, we first create an `evalMetrics` spec. We must provide:

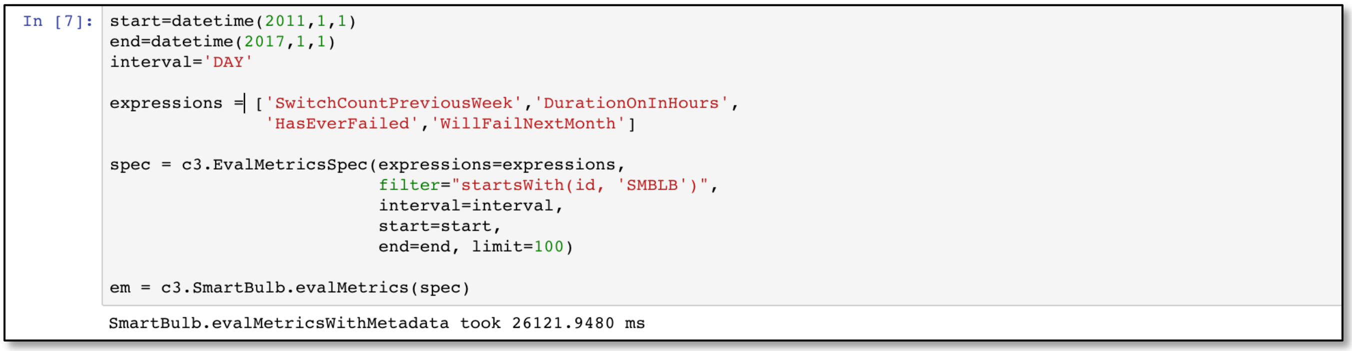
1. start: (1/1/2011)
2. end: (1/1/2017)
3. interval: (day)
4. list of metrics:
    a. `DurationOnInHours`
    b. `SwitchCountPreviousWeek`
    c. `HasEverFailed`
    d. `WillFailNextMonth`
5. (Optional) filter: (any bulb that has an id that starts with ‘SMBLB’)
6. (Optional) limit: (only return for 100 bulbs)

The first two metrics, `DurationOnInHours` and `SwitchCountPreviousWeek`, are the ones we hypothesize are correlated with light bulb failure.

The third metric, `HasEverFailed`, will be used to filter for light bulbs that have never failed.

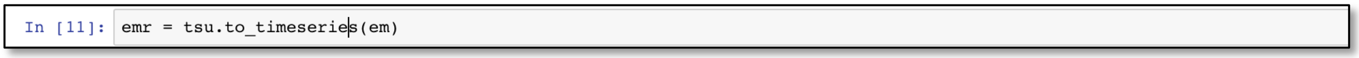
The last metric, `WillFailNextMonth`, will tell us whether a light bulb will fail in the next month. This is our dependent variable.

Execute the spec shown in the following image:



```
In [7]: start=datetime(2011,1,1)
        end=datetime(2017,1,1)
        interval='DAY'

        expressions = ['SwitchCountPreviousWeek','DurationOnInHours',
                       'HasEverFailed','WillFailNextMonth']

        spec = c3.EvalMetricsSpec(expressions=expressions,
                                  filter="startsWith(id, 'SMBLB')",
                                  interval=interval,
                                  start=start,
                                  end=end, limit=100)

        em = c3.SmartBulb.evalMetrics(spec)

        SmartBulb.evalMetricsWithMetadata took 26121.9480 ms
```

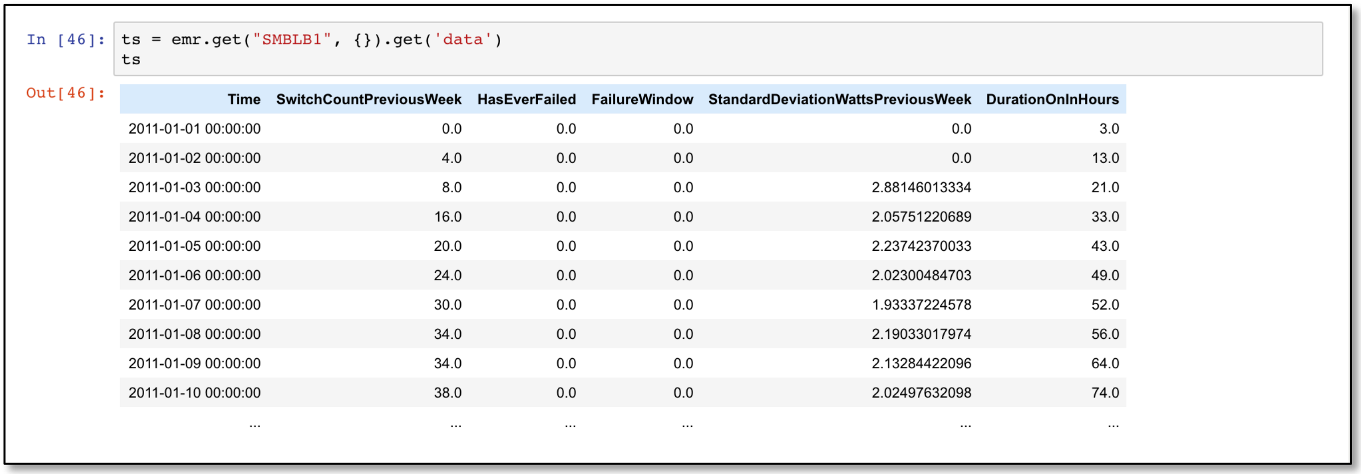
Having retrieved our time series data, we will now make use of `to_timeseries`, a convenience function defined on `C3ServerConnection` to convert `EvalMetricsResult` to a dictionary of `pyc3.TimeSeries`. Execute the following code:



```
In [11]: emr = tsu.to_timeseries(em)
```

We can now visually show this timeseries in a variety of ways:

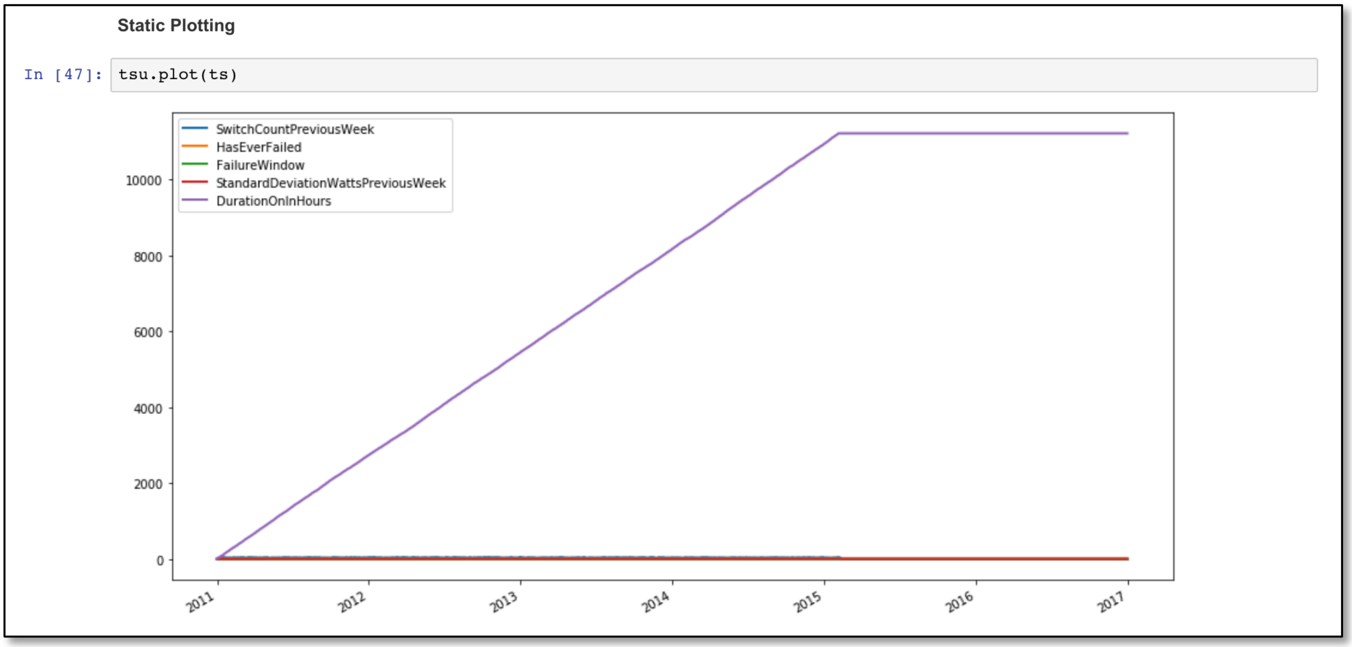
1. We can view a subset of the data in a grid view using the following command:



```
In [46]: ts = emr.get("SMBLB1", {}).get('data')
         ts

Out[46]:
```

Time	SwitchCountPreviousWeek	HasEverFailed	FailureWindow	StandardDeviationWattsPreviousWeek	DurationOnInHours
2011-01-01 00:00:00	0.0	0.0	0.0	0.0	3.0
2011-01-02 00:00:00	4.0	0.0	0.0	0.0	13.0
2011-01-03 00:00:00	8.0	0.0	0.0	2.88146013334	21.0
2011-01-04 00:00:00	16.0	0.0	0.0	2.05751220689	33.0
2011-01-05 00:00:00	20.0	0.0	0.0	2.23742370033	43.0
2011-01-06 00:00:00	24.0	0.0	0.0	2.02300484703	49.0
2011-01-07 00:00:00	30.0	0.0	0.0	1.93337224578	52.0
2011-01-08 00:00:00	34.0	0.0	0.0	2.19033017974	56.0
2011-01-09 00:00:00	34.0	0.0	0.0	2.13284422096	64.0
2011-01-10 00:00:00	38.0	0.0	0.0	2.02497632098	74.0
...	...	...	...	...	...

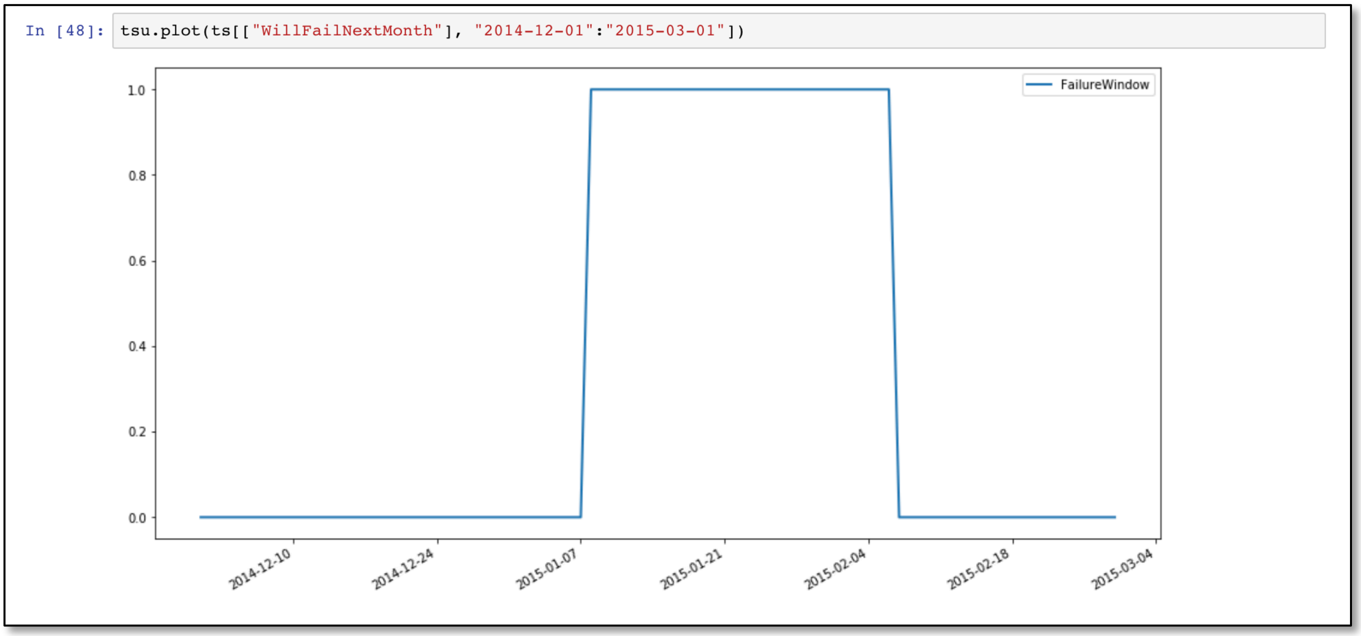
2. We can also view the timeseries as a static chart with the `tsu.plot()` command:



Static Plotting

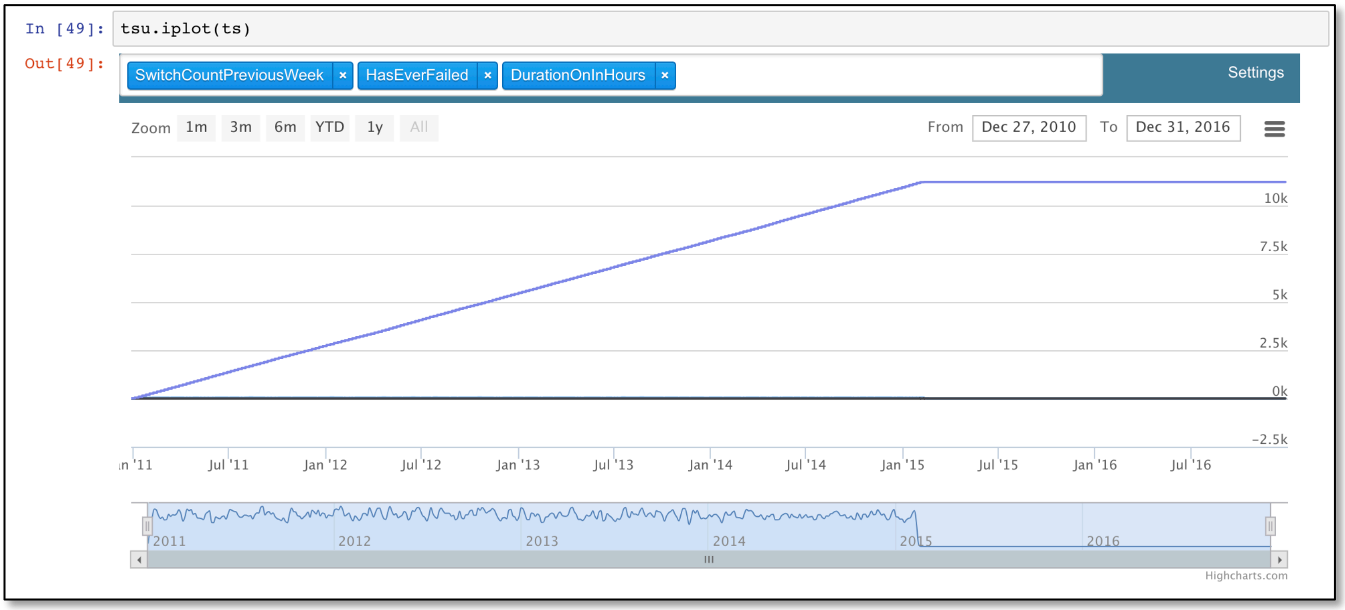
```
In [47]: tsu.plot(ts)
```

3. We can now slice the static chart of the time series by both metric and time by adding specifications to our timeseries, `ts`:



```
In [48]: tsu.plot(ts[["WillFailNextMonth"], "2014-12-01":"2015-03-01"])
```

4. Finally, we can create an interactive plot of the timeseries with the `tsu.iplot()` command:

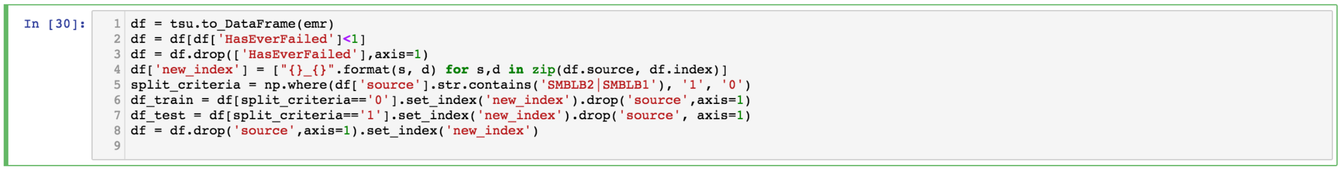


```
In [49]: tsu.iplot(ts)
```

Now that we have defined out timeseries, we need to write a custom function to split our dataset into a training set and test set, while removing the data where the bulb has died (denoted by the metric `HasEverFailed`).

To do this, we must filter our data sets to keep only the light bulbs that have never failed in our analysis window. This way we prevent the machine learning model running its algorithm on light bulbs that are already defective.

To filter the timeseries, execute the following code into your Jupyter Notebook:



```
In [30]:
df = tsu.to_DataFrame(emr)
df = df[df['HasEverFailed']<1]
df = df.drop(['HasEverFailed'],axis=1)
df['new_index'] = ["{}_{}".format(s, d) for s,d in zip(df.source, df.index)]
split_criteria = np.where(df['source'].str.contains('SMBLB2|SMBLB1'), '1', '0')
df_train = df[split_criteria=='0'].set_index('new_index').drop('source',axis=1)
df_test = df[split_criteria=='1'].set_index('new_index').drop('source', axis=1)
df = df.drop('source',axis=1).set_index('new_index')
```

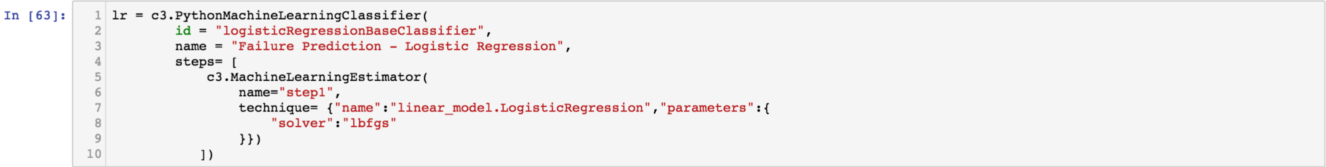
3. MACHINE LEARNING

With our timeseries filtered and split into two sets, we can define our pipeline model to train. The model will make use of the C3 Type `PythonMachineLearningClassifier`, and will consist of just one steps: Logistic Regression. Users can choose to specify additional steps, for example a Standard Scaler—a common requirement for many machine learning estimators. It involves removing the mean and scaling to unit variance.

1. Logistic regression is a special type of regression where a binary response is related to a set of explanatory variables. In our case, the binary response is `WillFailNextMonth`

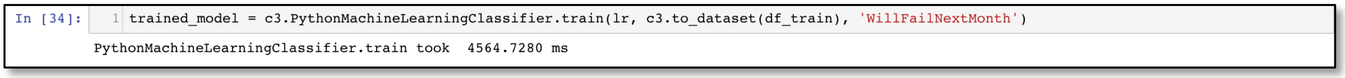
and the explanatory variables are DurationOnInHours and SwitchCountPreviousWeek, because we hypothesize that these variables are good indicators of whether or not the bulb will fail in the next month.

The final step is simply to provide an id and name for the model, and then to execute the code as shown below:



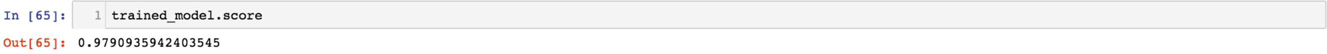
```
In [63]: lr = c3.PythonMachineLearningClassifier(
                 id = "logisticRegressionBaseClassifier",
                 name = "Failure Prediction - Logistic Regression",
                 steps= [
                     c3.MachineLearningEstimator(
                         name="step1",
                         technique= {"name":"linear_model.LogisticRegression","parameters":{
                             "solver":"lbfgs"
                         }})
                    ])
```

To train your model, call the train method on the PythonMachineLearningClassifier type and specify the training dataset and the target label:



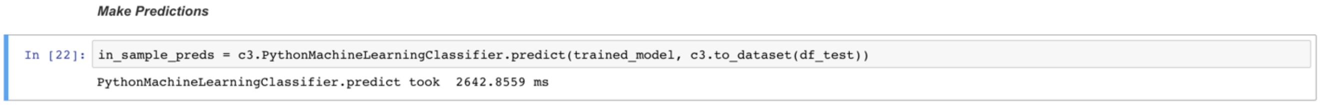
```
In [34]: trained_model = c3.PythonMachineLearningClassifier.train(lr, c3.to_dataset(df_train), 'WillFailNextMonth')

         PythonMachineLearningClassifier.train took  4564.7280 ms
```

We can check the accuracy / score of the model using the score method:



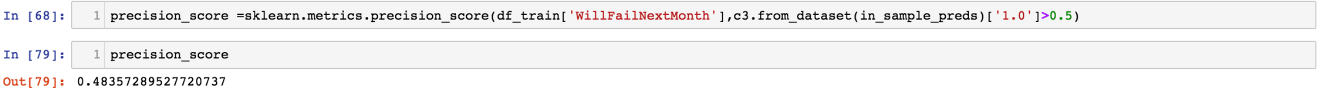
```
In [65]: trained_model.score
Out[65]: 0.9790935942403545
```

Predictions are made using the predict method.



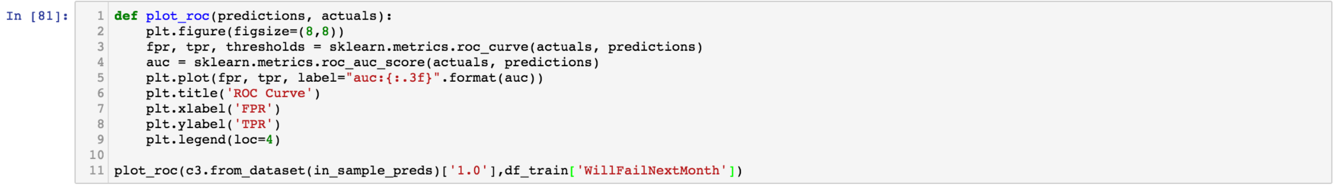
```
Make Predictions

In [22]: in_sample_preds = c3.PythonMachineLearningClassifier.predict(trained_model, c3.to_dataset(df_test))
         PythonMachineLearningClassifier.predict took  2642.8559 ms
```

We can check the precision of our trained model with these in sample predictions (using 0.5 as our classification threshold):

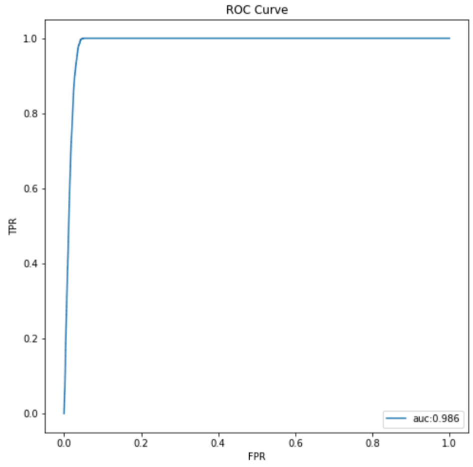


```
In [68]: precision_score =sklearn.metrics.precision_score(df_train['WillFailNextMonth'],c3.from_dataset(in_sample_preds)['1.0']>0.5)

In [79]: precision_score
Out[79]: 0.48357289527720737
```

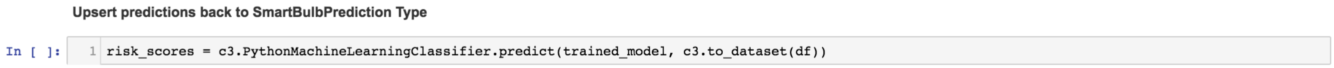
Below, we make predictions and plot the ROC curve and print the AUC. As you can see the model has done a remarkable job predicting bulb failure!



In [81]:
```
def plot_roc(predictions, actuals):
    plt.figure(figsize=(8,8))
    fpr, tpr, thresholds = sklearn.metrics.roc_curve(actuals, predictions)
    auc = sklearn.metrics.roc_auc_score(actuals, predictions)
    plt.plot(fpr, tpr, label="auc:{:.3f}".format(auc))
    plt.title('ROC Curve')
    plt.xlabel('FPR')
    plt.ylabel('TPR')
    plt.legend(loc=4)

plot_roc(c3.from_dataset(in_sample_preds)['1.0'],df_train['WillFailNextMonth'])
```



We ultimately wish to upload risk scores for all bulbs – whether they are in the train set or test set, so before proceeding to upsert we pass the entire dataframe (both train and test sets) to the predict api call.



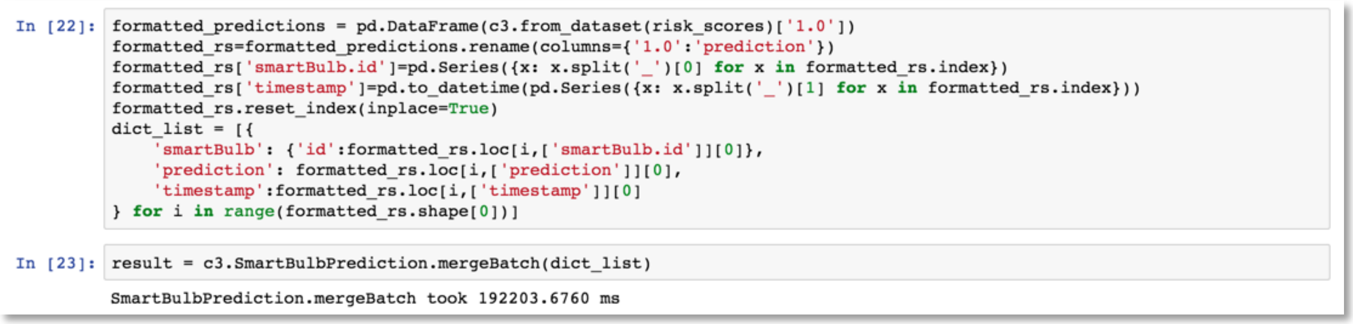
Upsert predictions back to SmartBulbPrediction Type

In [ ]:
```
risk_scores = c3.PythonMachineLearningClassifier.predict(trained_model, c3.to_dataset(df))
```

We now both have a trained ML model as well as a light bulb dataset that has predictive risk scores. The final step is to upsert these risk scores back to the C3 IoT Platform so that they can be used in further analysis or visualization.

To do so, we need to create a list of dictionaries with mapping:

a. `Column: id           to   Field Path: smartBulb.id`
b. `Column Prediction    to   Field Path: prediction`
c. `Column timestamp     to   Field Path: timestamp`



```
In [22]: formatted_predictions = pd.DataFrame(c3.from_dataset(risk_scores)['1.0'])
         formatted_rs=formatted_predictions.rename(columns={'1.0':'prediction'})
         formatted_rs['smartBulb.id']=pd.Series({x: x.split('_')[0] for x in formatted_rs.index})
         formatted_rs['timestamp']=pd.to_datetime(pd.Series({x: x.split('_')[1] for x in formatted_rs.index}))
         formatted_rs.reset_index(inplace=True)
         dict_list = [{
             'smartBulb': {'id':formatted_rs.loc[i,['smartBulb.id']][0]},
             'prediction': formatted_rs.loc[i,['prediction']][0],
             'timestamp':formatted_rs.loc[i,['timestamp']][0]
         } for i in range(formatted_rs.shape[0])]

In [23]: result = c3.SmartBulbPrediction.mergeBatch(dict_list)

         SmartBulbPrediction.mergeBatch took 192203.6760 ms
```

Congratulations! You have just made your first ML model in Jupyter Notebook!