Predicting Car Battery Failure

Your goal in this notebook is to **predict how much time a car battery has left until it is expected to fail**. You are provided training data that includes telemetry from different vehicles, as well as the expected battery life that remains. From this you will train a model that given just the vehicle telemetry predicts the expected battery life.

You will use compute resources provided by Azure Machine Learning (AML) to **remotely** train a **set** of models using **Automated Machine Learning**, evaluate performance of each model and pick the best performing model to deploy as a web service hosted by **Azure Kubernetes Service**.

Because you will be using the Azure Machine Learning SDK, you will be able to provision all your required Azure resources directly from this notebook, without having to use the Azure Portal to create any resources.

Setup

To begin, you will need to provide the following information about your Azure Subscription.

In the following cell, be sure to set the values for `subscription_id`, `resource_group`, `workspace_name` and `workspace_region` as directed by the comments (*these values can be acquired from the Azure Portal*). Also provide the values for the pre-created AKS cluster (`aks_resource_group_name` and `aks_cluster_name`).

Execute the following cell by selecting the `&gt;|Run` button in the command bar above.

In [1]:
```
#Provide the Subscription ID of your existing Azure subscription
subscription_id = "30fc406c-c745-44f0-be2d-63b1c860cde0"

#Provide values for the new Resource Group and Workspace that will be created
resource_group = "aml-workspace-tech-immersion"
workspace_name = "aml-workspace"

#Provide values for the pre-created AKS cluster
aks_resource_group_name = "ti-aks"
aks_cluster_name = 'my-aks-cluster'

#Optionally, set the Azure Region in which to deploy your Azure Machine Learni
ng Workspace
workspace_region = "westcentralus" # other options include eastus, westcentral
us, southeastasia, australiaeast, westeurope
```

In [39]:
```
# constants, you can leave these values as they are or experiment with changin
g them after you have completed the notebook once
experiment_name = 'automl-regression'
project_folder = './automl-regression'

# this is the URL to the CSV file containing the training data
data_url = "https://databricksdemostore.blob.core.windows.net/data/connected-c
ar/training-formatted.csv"

# this is the URL to the CSV file containing a small set of test data
test_data_url = "https://databricksdemostore.blob.core.windows.net/data/connec
ted-car/fleet-formatted.csv"

cluster_name = "cpucluster"
aks_service_name ='contoso-service'
```

Import required packages

The Azure Machine Learning SDK provides a comprehensive set of a capabilities that you can use directly within a notebook including:

- Creating a **Workspace** that acts as the root object to organize all artifacts and resources used by Azure Machine Learning.
- Creating **Experiments** in your Workspace that capture versions of the trained model along with any desired model performance telemetry. Each time you train a model and evaluate its results, you can capture that run (model and telemetry) within an Experiment.
- Creating **Compute** resources that can be used to scale out model training, so that while your notebook may be running in a lightweight container in Azure Notebooks, your model training can actually occur on a powerful cluster that can provide large amounts of memory, CPU or GPU.
- Using **Automated Machine Learning (AutoML)** to automatically train multiple versions of a model using a mix of different ways to prepare the data and different algorithms and hyperparameters (algorithm settings) in search of the model that performs best according to a performance metric that you specify.
- Packaging a Docker **Image** that contains everything your trained model needs for scoring (prediction) in order to run as a web service.
- Deploying your Image to either Azure Kubernetes or Azure Container Instances, effectively hosting the **Web Service**.

In Azure Notebooks, all of the libraries needed for Azure Machine Learning are pre-installed. To use them, you just need to import them. Run the following cell to do so:

In [3]:
```
import logging
import os
import random
import re

from matplotlib import pyplot as plt
from matplotlib.pyplot import imshow
import numpy as np
import pandas as pd
from sklearn import datasets

import azureml.core
from azureml.core.experiment import Experiment
from azureml.core.workspace import Workspace
from azureml.core.compute import AksCompute, ComputeTarget
from azureml.core.webservice import Webservice, AksWebservice
from azureml.core.image import Image
from azureml.core.model import Model
from azureml.train.automl import AutoMLConfig
from azureml.train.automl.run import AutoMLRun
from azureml.core import Workspace
```

```
/home/nbuser/anaconda3_501/lib/python3.6/site-packages/matplotlib/font_manage
r.py:229: UserWarning: Matplotlib is building the font cache using fc-list. T
his may take a moment.
  'Matplotlib is building the font cache using fc-list. '
```

Create and connect to an Azure Machine Learning Workspace

Run the following cell to create a new Azure Machine Learning **Workspace** and save the configuration to disk (next to the Jupyter notebook).

Important Note: You will be prompted to login in the text that is output below the cell. Be sure to navigate to the URL displayed and enter the code that is provided. Once you have entered the code, return to this notebook and wait for the output to read `Library configuration succeeded`.

In [4]:
```
# By using the exist_ok param, if the worskpace already exists you get a refer
ence to the existing workspace
# allowing you to re-run this cell multiple times as desired (which is fairly
 common in notebooks).
ws = Workspace.create(
    name = workspace_name,
    subscription_id = subscription_id,
    resource_group = resource_group,
    location = workspace_region,
    exist_ok = True)

ws.write_config()
print('Library configuration succeeded')
```

```
Warning: Falling back to use azure cli login credentials.
If you run your code in unattended mode, i.e., where you can't give a user in
put, then we recommend to use ServicePrincipalAuthentication or MsiAuthentica
tion.
Please refer to aka.ms/aml-notebook-auth for different authentication mechani
sms in azureml-sdk.

Performing interactive authentication. Please follow the instructions on the
terminal.
To sign in, use a web browser to open the page https://microsoft.com/devicelo
gin and enter the code F97AKE725 to authenticate.
Interactive authentication successfully completed.
Wrote the config file config.json to: /home/nbuser/library/aml_config/config.
json
Library configuration succeeded
```

Create a Workspace Experiment

Notice in the first line of the cell below, we can re-load the config we saved previously and then display a summary of the environment.

In [5]:
```
ws = Workspace.from_config()

# Display a summary of the current environment
output = {}
output['SDK version'] = azureml.core.VERSION
output['Subscription ID'] = ws.subscription_id
output['Workspace'] = ws.name
output['Resource Group'] = ws.resource_group
output['Location'] = ws.location
output['Project Directory'] = project_folder
pd.set_option('display.max_colwidth', -1)
pd.DataFrame(data=output, index=['']).T
```

```
Found the config file in: /home/nbuser/library/aml_config/config.json
```

Out[5]:

Location	westcentralus
Project Directory	./automl-regression
Resource Group	aml-workspace-tech-immersion
SDK version	1.0.17
Subscription ID	30fc406c-c745-44f0-be2d-63b1c860cde0
Workspace	aml-workspace

Next, create a new Experiment.

In [6]:
```
experiment = Experiment(ws, experiment_name)
```

Get and explore the Vehicle Telemetry Data

Run the following cell to download and examine the vehicle telemetry data. The model you will build will try to predict how many days until the battery has a freeze event. Which features (columns) do you think will be useful?

In [7]:
```
data = pd.read_csv(data_url)
data
```

Out[7]:

	Survival_In_Days	Province	Region	Trip_Length_Mean	Trip
0	1283	Bretagne	West	18.10	6.0
1	1427	Occitanie	South	14.64	4.8
2	1436	Auvergne_Rhone_Alpes	South	14.51	4.8
3	894	Martinique	West	20.85	6.9
4	1539	Reunion	South	11.58	3.8
5	1872	Marseille	South	14.07	4.6
6	151	Ile_de_France	MidWest	13.39	4.4
7	1975	Normandie	MidWest	16.72	5.5
8	1957	Paris	MidWest	12.28	4.0
9	1150	Corse	South	19.62	6.5
10	149	Bretagne	West	24.92	8.3
11	1746	Occitanie	South	12.85	4.2
12	1264	Provence_Alpes_Cote_d_Azure	South	14.43	4.8
13	1239	Martinique	West	24.37	8.1
14	746	Reunion	South	12.36	4.1
15	2169	Marseille	South	16.55	5.5
16	364	Ile_de_France	MidWest	11.73	3.9
17	1077	Normandie	MidWest	19.48	6.4
18	1671	Paris	MidWest	12.26	4.0
19	1517	Hauts_de_France	Northeast	10.62	3.5
20	903	Bretagne	West	22.01	7.3
21	1525	Occitanie	South	16.30	5.4
22	1679	Provence_Alpes_Cote_d_Azure	South	16.96	5.6
23	1292	Martinique	West	20.75	6.9
24	1949	Reunion	South	10.82	3.6
25	1441	Marseille	South	17.21	5.7
26	458	Mayotte	South	15.31	5.1
27	1583	Normandie	MidWest	17.17	5.7
28	1257	Paris	MidWest	13.38	4.4
29	1873	Hauts_de_France	Northeast	12.36	4.1
...	...	...	...	...	...
9970	1277	Normandie	MidWest	18.30	6.1

	Survival_In_Days	Province	Region	Trip_Length_Mean	Tri
9971	886	Guadeloupe	West	14.40	4.8
9972	1077	Corse	South	21.54	7.1
9973	1520	Bretagne	West	19.91	6.6
9974	1348	French_Guiana	South	17.96	5.9
9975	1848	Auvergne_Rhone_Alpes	South	18.79	6.2
9976	1176	Martinique	West	24.30	8.1
9977	638	Reunion	South	13.30	4.4
9978	1685	Grand_Est	Northeast	14.86	4.9
9979	320	Ile_de_France	MidWest	12.02	4.0
9980	1129	Normandie	MidWest	16.14	5.3
9981	1642	Paris	MidWest	13.28	4.4
9982	1672	Corse	South	19.14	6.3
9983	1403	Bretagne	West	17.34	5.7
9984	1658	Occitanie	South	15.44	5.1
9985	1356	Auvergne_Rhone_Alpes	South	16.42	5.4
9986	1085	Martinique	West	21.44	7.1
9987	1492	Reunion	South	9.86	3.2
9988	1927	Marseille	South	15.01	5.0
9989	1812	Ile_de_France	MidWest	14.17	4.7
9990	1497	Normandie	MidWest	14.13	4.7
9991	1602	Paris	MidWest	14.69	4.9
9992	1410	Corse	South	18.88	6.2
9993	1273	Bretagne	West	19.92	6.6
9994	1478	Occitanie	South	13.64	4.5
9995	1774	Provence_Alpes_Cote_d_Azure	South	19.72	6.5
9996	1078	Martinique	West	22.00	7.3
9997	778	Reunion	South	11.61	3.8
9998	1772	Marseille	South	13.12	4.3
9999	455	Ile_de_France	MidWest	14.83	4.9

10000 rows × 74 columns

Remotely train multiple models using Auto ML and Azure ML Compute

In the following cells, you will *not* train the model against the data you just downloaded using the resources provided by Azure Notebooks. Instead, you will deploy an Azure ML Compute cluster that will download the data and use Auto ML to train multiple models, evaluate the performance and allow you to retrieve the best model that was trained. In other words, all of the training will be performed remotely with respect to this notebook.

As you will see this is almost entirely done thru configuration, with very little code required.

Create the data loading script for remote compute

The Azure Machine Learning Compute cluster needs to know how to get the data to train against. You can package this logic in a script that will be executed by the compute when it starts executing the training.

Run the following cells to locally create the **get_data.py** script that will be deployed to remote compute. You will also use this script when you want train the model locally.

Observe that the get_data method returns the features (`X`) and the labels (`Y`) in an object. This structure is expected later when you will configure Auto ML.

In [8]:
```
# create project folder
if not os.path.exists(project_folder):
    os.makedirs(project_folder)
```

In [9]:
```
%%writefile $project_folder/get_data.py

import pandas as pd
import numpy as np

def get_data():

    data = pd.read_csv("https://databricksdemostore.blob.core.windows.net/dat
a/connected-car/training-formatted.csv")

    X = data.iloc[:,1:73]
    Y = data.iloc[:,0].values.flatten()

    return { "X" : X, "y" : Y }
```

```
Overwriting ./automl-regression/get_data.py
```

Create AML Compute Cluster

Now you are ready to create the compute cluster. Run the following cell to create a new compute cluster (or retrieve the existing cluster if it already exists). The code below will create a *CPU based* cluster where each node in the cluster is of the size `STANDARD_D12_V2`, and the cluster will have at most *4* such nodes.

In [10]:
```
### Create AML CPU based Compute Cluster
from azureml.core.compute import ComputeTarget, AmlCompute
from azureml.core.compute_target import ComputeTargetException

try:
    compute_target = ComputeTarget(workspace=ws, name=cluster_name)
    print('Found existing compute target.')
except ComputeTargetException:
    print('Creating a new compute target...')
    compute_config = AmlCompute.provisioning_configuration(vm_size='STANDARD_D
12_V2',
                                                           max_nodes=4)

    # create the cluster
    compute_target = ComputeTarget.create(ws, cluster_name, compute_config)

    compute_target.wait_for_completion(show_output=True)

# Use the 'status' property to get a detailed status for the current AmlComput
e.
print(compute_target.status.serialize())
```

```
Found existing compute target.
{'allocationState': 'Steady', 'allocationStateTransitionTime': '2019-03-17T1
9:52:56.295000+00:00', 'creationTime': '2019-03-17T19:18:49.583535+00:00', 'c
urrentNodeCount': 0, 'errors': None, 'modifiedTime': '2019-03-17T19:19:23.811
310+00:00', 'nodeStateCounts': {'idleNodeCount': 0, 'leavingNodeCount': 0, 'p
reemptedNodeCount': 0, 'preparingNodeCount': 0, 'runningNodeCount': 0, 'unusa
bleNodeCount': 0}, 'provisioningState': 'Succeeded', 'provisioningStateTransi
tionTime': None, 'scaleSettings': {'minNodeCount': 0, 'maxNodeCount': 4, 'nod
eIdleTimeBeforeScaleDown': 'PT120S'}, 'targetNodeCount': 0, 'vmPriority': 'De
dicated', 'vmSize': 'STANDARD_D12_V2'}
```

Instantiate an Automated ML Config

Run the following cell to configure the Auto ML run. In short what you are configuring here is the training of a regressor model that will attempt to predict the value of the first feature (`Survival_in_days`) based on all the other features in the data set. The run is configured to try at most 3 iterations where no iteration can run longer that 2 minutes.

Additionally, the data will be automatically pre-processed in different ways as a part of the automated model training (as indicated by the `preprocess` attribute having a value of `True`. This is a very powerful feature of Auto ML as it tries many best practices approaches for you, and saves you a lot of time and effort in the process.

The goal of Auto ML in this case is to find the best models that result, as measure by the normalized root mean squared error metric (as indicated by the `primary_metric` attribute). The error is basically a measure of what the model predicts versus what was provided as the "answer" in the training data. In short, AutoML will try to get the error as low as possible when trying its combination of approaches.

The local path to the script you created to retrieve the data is supplied to the AutoMLConfig, ensuring the file is made available to the remote cluster. The actual execution of this training will occur on the compute cluster you created previously.

In general, the AutoMLConfig is very flexible, allowing you to specify all of the following:

- Task type (classification, regression, forecasting)
- Number of algorithm iterations and maximum time per iteration
- Accuracy metric to optimize
- Algorithms to blacklist (skip)/whitelist (include)
- Number of cross-validations
- Compute targets
- Training data

Run the following cell to create the configuration.

In [11]:
```
automl_config = AutoMLConfig(task = 'regression',
                             iterations = 3,
                             iteration_timeout_minutes = 2,
                             max_cores_per_iteration = 10,
                             preprocess= True,
                             primary_metric='normalized_root_mean_squared_erro
r',
                             n_cross_validations = 5,
                             debug_log = 'automl.log',
                             verbosity = logging.DEBUG,
                             data_script = project_folder + "/get_data.py",
                             path = project_folder)
```

Run locally

You can run AutomML locally, that is it will use the resource provided by your Azure Notebook environment.

Run the following cell to run the experiment locally. Note this will take **a few minutes**.

In [13]:
```
local_run = experiment.submit(automl_config, show_output=True)
local_run
```

```
Running on local machine
Parent Run ID: AutoML_0c2ba27e-481f-48fd-804d-8b9e9012a7ee
*****************************************************************************
**************************************
ITERATION: The iteration being evaluated.
PIPELINE: A summary description of the pipeline being evaluated.
SAMPLING %: Percent of the training data to sample.
DURATION: Time taken for the current iteration.
METRIC: The result of computing score on the fitted pipeline.
BEST: The best observed score thus far.
*****************************************************************************
**************************************

 ITERATION   PIPELINE                                      SAMPLING %  DURAT
ION      METRIC      BEST
         0                                                 100.0000    0:02:
23          nan       nan
ERROR: Fit operation exceeded provided timeout, terminating and moving onto t
he next iteration.
         1                                                 100.0000    0:02:
12          nan       nan
ERROR: Fit operation exceeded provided timeout, terminating and moving onto t
he next iteration.
         2                                                 100.0000    0:00:
34          nan       nan
ERROR: Run 'AutoML_0c2ba27e-481f-48fd-804d-8b9e9012a7ee_2' failed. Check the
log for more details.
```

Out[13]:

Experiment	Id	Type	Status	Details Page	Docs Page
automl-regression	AutoML_0c2ba27e-481f-48fd-804d-8b9e9012a7ee	automl	Completed	Link to Azure Portal	Link to Documentation

Run our Experiment on AML Compute

Let's increase the performance by performing the training the AML Compute cluster. This will remotely train multiple models, evaluate them and allow you review the performance characteristics of each one, as well as to pick the *best model* that was trained and download it.

We will alter the configuration slightly to perform more iterations. Run the following cell to execute the experiment on the remote compute cluster.

In [12]:
```
automl_config = AutoMLConfig(task = 'regression',
                             iterations = 4,
                             iteration_timeout_minutes = 10,
                             max_cores_per_iteration = 10,
                             preprocess= True,
                             primary_metric='normalized_root_mean_squared_erro
r',
                             n_cross_validations = 5,
                             debug_log = 'automl.log',
                             verbosity = logging.DEBUG,
                             data_script = project_folder + "/get_data.py",
                             compute_target = compute_target,
                             path = project_folder)
remote_run = experiment.submit(automl_config, show_output=False)
remote_run
```

Out[12]:

Experiment	Id	Type	Status	Details Page	Docs Page
automl-regression	AutoML_27486c43-de93-440b-b614-d8d63bb7da8d	automl	Preparing	Link to Azure Portal	Link to Documentation

Once the above cell completes, the run is starting but will likely have a status of `Preparing` for you. To wait for the run to complete before continuing (and to view the training status updates as they happen), run the following cell (this will take **a few minutes** to complete, output will be streamed as the tasks progress):

In [13]:
```
remote_run.wait_for_completion(show_output=True)
```

```
*****************************************************************************
**************************************
ITERATION: The iteration being evaluated.
PIPELINE: A summary description of the pipeline being evaluated.
SAMPLING %: Percent of the training data to sample.
DURATION: Time taken for the current iteration.
METRIC: The result of computing score on the fitted pipeline.
BEST: The best observed score thus far.
*****************************************************************************
**************************************

 ITERATION   PIPELINE                                       SAMPLING %  DURAT
ION      METRIC      BEST
         0   StandardScalerWrapper ElasticNet               100.0000    0:01:
24       0.0950    0.0950
         1   StandardScalerWrapper ElasticNet               100.0000    0:01:
00       0.0946    0.0946
         2   StandardScalerWrapper ElasticNet               100.0000    0:01:
07       0.1190    0.0946
         3    Ensemble                                      100.0000    0:00:
53       0.0968    0.0946

Execution Summary
=================
RunId: AutoML_27486c43-de93-440b-b614-d8d63bb7da8d
```

```
Out[13]: {'runId': 'AutoML_27486c43-de93-440b-b614-d8d63bb7da8d',
          'target': 'cpucluster',
          'status': 'Completed',
          'startTimeUtc': '2019-03-17T22:19:47.894349Z',
          'endTimeUtc': '2019-03-17T22:24:27.222564Z',
          'properties': {'num_iterations': '4',
           'training_type': 'TrainFull',
           'acquisition_function': 'EI',
           'primary_metric': 'normalized_root_mean_squared_error',
           'train_split': '0',
           'MaxTimeSeconds': '600',
           'acquisition_parameter': '0',
           'num_cross_validation': '5',
           'target': 'cpucluster',
           'RawAMLSettingsString': "{'name': 'automl-regression', 'path': './automl-re
         gression', 'subscription_id': '30fc406c-c745-44f0-be2d-63b1c860cde0', 'resour
         ce_group': 'aml-workspace-tech-immersion', 'workspace_name': 'aml-workspace',
         'iterations': 4, 'primary_metric': 'normalized_root_mean_squared_error', 'dat
         a_script': './automl-regression/get_data.py', 'compute_target': 'cpucluster',
         'task_type': 'regression', 'validation_size': 0.0, 'n_cross_validations': 5,
         'y_min': None, 'y_max': None, 'num_classes': None, 'preprocess': True, 'lag_l
         ength': 0, 'is_timeseries': False, 'max_cores_per_iteration': 10, 'max_concur
         rent_iterations': 1, 'iteration_timeout_minutes': 10, 'mem_in_mb': None, 'enf
         orce_time_on_windows': False, 'experiment_timeout_minutes': None, 'experiment
         _exit_score': None, 'whitelist_models': None, 'blacklist_algos': ['XGBoostReg
         ressor', 'XGBoostRegressor', 'KNeighborsClassifier', 'KNeighborsRegressor',
         'SVCWrapper', 'kNN', 'kNN regressor', 'SVM'], 'auto_blacklist': True, 'blackl
         ist_samples_reached': True, 'exclude_nan_labels': True, 'verbosity': 10, 'deb
         ug_log': 'automl.log', 'show_warnings': False, 'model_explainability': False,
         'service_url': None, 'sdk_url': None, 'sdk_packages': None, 'telemetry_verbos
         ity': 'INFO', 'send_telemetry': True, 'spark_service': None, 'metrics': None,
         'enable_ensembling': True, 'ensemble_iterations': 4, 'enable_tf': False, 'ena
         ble_cache': True, 'enable_subsampling': False, 'subsample_seed': None, 'cost_
         mode': 0, 'metric_operation': 'minimize'}",
           'AMLSettingsJsonString': '{\n  "name": "automl-regression",\n  "path": "./a
         utoml-regression",\n  "subscription_id": "30fc406c-c745-44f0-be2d-63b1c860cde
         0",\n  "resource_group": "aml-workspace-tech-immersion",\n  "workspace_name":
         "aml-workspace",\n  "iterations": 4,\n  "primary_metric": "normalized_root_me
         an_squared_error",\n  "data_script": "./automl-regression/get_data.py",\n  "c
         ompute_target": "cpucluster",\n  "task_type": "regression",\n  "validation_si
         ze": 0.0,\n  "n_cross_validations": 5,\n  "y_min": null,\n  "y_max": null,\n
         "num_classes": null,\n  "preprocess": true,\n  "lag_length": 0,\n  "is_timese
         ries": false,\n  "max_cores_per_iteration": 10,\n  "max_concurrent_iteration
         s": 1,\n  "iteration_timeout_minutes": 10,\n  "mem_in_mb": null,\n  "enforce_
         time_on_windows": false,\n  "experiment_timeout_minutes": null,\n  "experimen
         t_exit_score": null,\n  "whitelist_models": null,\n  "blacklist_algos": [\n
         "XGBoostRegressor",\n    "XGBoostRegressor",\n    "KNeighborsClassifier",\n
         "KNeighborsRegressor",\n    "SVCWrapper",\n    "kNN",\n    "kNN regressor",\n
         "SVM"\n  ],\n  "auto_blacklist": true,\n  "blacklist_samples_reached": tru
         e,\n  "exclude_nan_labels": true,\n  "verbosity": 10,\n  "debug_log": "autom
         l.log",\n  "show_warnings": false,\n  "model_explainability": false,\n  "serv
         ice_url": null,\n  "sdk_url": null,\n  "sdk_packages": null,\n  "telemetry_ve
         rbosity": "INFO",\n  "send_telemetry": true,\n  "spark_service": null,\n  "me
         trics": null,\n  "enable_ensembling": true,\n  "ensemble_iterations": 4,\n
         "enable_tf": false,\n  "enable_cache": true,\n  "enable_subsampling": fals
         e,\n  "subsample_seed": null,\n  "cost_mode": 0,\n  "metric_operation": "mini
         mize"\n}',
           'DataPrepJsonString': None,
           'EnableSubsampling': 'False',
           'runTemplate': 'AutoML',
           'azureml.runsource': 'automl',
           'dependencies_versions': '{"azureml-widgets": "1.0.17", "azureml-train":
         "1.0.17", "azureml-train-restclients-hyperdrive": "1.0.17", "azureml-train-co
         re": "1.0.17", "azureml-train-automl": "1.0.17.1", "azureml-telemetry": "1.0.
         17", "azureml-sdk": "1.0.17", "azureml-pipeline": "1.0.17", "azureml-pipeline
         -steps": "1.0.17", "azureml-pipeline-core": "1.0.17", "azureml-explain-mode
         l": "1.0.17", "azureml-dataprep": "1.0.16", "azureml-dataprep-native": "11.2.
         2", "azureml-core": "1.0.17.1", "azureml-contrib-notebook": "1.0.17"}',
           'ContentSnapshotId': '7bfb34fc-d4a3-4f14-9973-0ce044d3081b',
           'snapshotId': '7bfb34fc-d4a3-4f14-9973-0ce044d3081b',
           'SetupRunId': 'AutoML_27486c43-de93-440b-b614-d8d63bb7da8d_setup',
           'ProblemInfoJsonString': '{"dataset_num_categorical": 0, "dataset_classes":
         2243, "dataset_features": 125, "dataset_samples": 10000, "is_sparse": true,
         "subsampling": false}'},
          'logFiles': {}}
```

List the Experiments from your Workspace

Using the Azure Machine Learning SDK, you can retrieve any of the experiments in your Workspace and drill into the details of any runs the experiment contains. Run the following cell to explore the number of runs by experiment name.

In [16]:
```
ws = Workspace.from_config()
experiment_list = Experiment.list(workspace=ws)

summary_df = pd.DataFrame(index = ['No of Runs'])
pattern = re.compile('^AutoML_[^_]*$')
for experiment in experiment_list:
    all_runs = list(experiment.get_runs())
    automl_runs = []
    for run in all_runs:
        if(pattern.match(run.id)):
            automl_runs.append(run)
    summary_df[experiment.name] = [len(automl_runs)]

pd.set_option('display.max_colwidth', -1)
summary_df.T
```

```
Found the config file in: /home/nbuser/library/aml_config/config.json
```

Out[16]:

	No of Runs
automl-regression	8

List the Automated ML Runs for the Experiment

Similarly, you can view all of the runs that ran supporting Auto ML:

In [17]:
```
proj = ws.experiments[experiment_name]
summary_df = pd.DataFrame(index = ['Type', 'Status', 'Primary Metric', 'Iterat
ions', 'Compute', 'Name'])
pattern = re.compile('^AutoML_[^_]*$')
all_runs = list(proj.get_runs(properties={'azureml.runsource': 'automl'}))
for run in all_runs:
    if(pattern.match(run.id)):
        properties = run.get_properties()
        tags = run.get_tags()
        amlsettings = eval(properties['RawAMLSettingsString'])
        if 'iterations' in tags:
            iterations = tags['iterations']
        else:
            iterations = properties['num_iterations']
        summary_df[run.id] = [amlsettings['task_type'], run.get_details()['sta
tus'], properties['primary_metric'], iterations, properties['target'], amlsett
ings['name']]

from IPython.display import HTML
projname_html = HTML("<h3>{}</h3>".format(proj.name))

from IPython.display import display
display(projname_html)
display(summary_df.T)
```

automl-regression

	Type	Status	Primary Metric	Iterations
AutoML_0d4daa95-edc9-48ce-95c7-506e00c40d86	regression	Completed	normalized_root_mean_squared_error	4
AutoML_0c2ba27e-481f-48fd-804d-8b9e9012a7ee	regression	Completed	normalized_root_mean_squared_error	3
AutoML_c48adbe5-e51d-4564-a99e-ebb6fc1c9929	regression	Completed	normalized_root_mean_squared_error	10
AutoML_2b399222-b0d2-4662-90fa-eb0f15384180	regression	Completed	normalized_root_mean_squared_error	3
AutoML_4fde9917-479a-4531-ab0e-b27dea36ea90	regression	Canceled	normalized_root_mean_squared_error	10
AutoML_c68492de-f8c0-4cd0-aafc-0d4acf106943	regression	Completed	normalized_root_mean_squared_error	10
AutoML_d7ceb6dc-2572-4be3-b284-1712fffd47c2	regression	Completed	normalized_root_mean_squared_error	10
AutoML_f85a80f5-dd65-4a53-881c-f2fad4a2ae22	regression	Completed	normalized_root_mean_squared_error	10

Display Automated ML Run Details

For a particular run, you can display the details of how th run performed against the performance metric. The Azure Machine Learning SDK includes a built-in widget that graphically summarizes the run.

Execute the following cell to see it.

In [14]:
```
run_id = remote_run.id

from azureml.widgets import RunDetails

experiment = Experiment(ws, experiment_name)
ml_run = AutoMLRun(experiment=experiment, run_id=run_id)

RunDetails(ml_run).show()
```

```
_AutoMLWidget(widget_settings={'childWidgetDisplay': 'popup', 'send_telemetr
y': False, 'log_level': 'INFO', 's…
```

Get the best run and the trained model

At this point you have multiple runs, each with a different trained models. How can you get the model that performed the best? Run the following cells to learn how.

In [15]:
```
best_run, fitted_model = remote_run.get_output()
print(best_run)
print(fitted_model)
```

```
Run(Experiment: automl-regression,
Id: AutoML_27486c43-de93-440b-b614-d8d63bb7da8d_1,
Type: azureml.scriptrun,
Status: Completed)
Pipeline(memory=None,
     steps=[('datatransformer', DataTransformer(logger=None, task=None)), ('s
tandardscalerwrapper', <automl.client.core.common.model_wrappers.StandardScal
erWrapper object at 0x7f82f93d1828>), ('elasticnet', ElasticNet(alpha=0.15873
684210526315, copy_X=True, fit_intercept=True,
      l1_ratio=0.6873684210526316, max_iter=1000, normalize=False,
      positive=False, precompute=False, random_state=None,
      selection='cyclic', tol=0.0001, warm_start=False))])
```

You can query for the best run when evaluated using a specific metric.

```
In [16]: # show run and model by a specific metric
         lookup_metric = "root_mean_squared_error"
         best_run, fitted_model = remote_run.get_output(metric = lookup_metric)
         print(best_run)
         print(fitted_model)
```

```
Run(Experiment: automl-regression,
Id: AutoML_27486c43-de93-440b-b614-d8d63bb7da8d_1,
Type: azureml.scriptrun,
Status: Completed)
Pipeline(memory=None,
     steps=[('datatransformer', DataTransformer(logger=None, task=None)), ('s
tandardscalerwrapper', <automl.client.core.common.model_wrappers.StandardScal
erWrapper object at 0x7f82f934fa20>), ('elasticnet', ElasticNet(alpha=0.15873
684210526315, copy_X=True, fit_intercept=True,
      l1_ratio=0.6873684210526316, max_iter=1000, normalize=False,
      positive=False, precompute=False, random_state=None,
      selection='cyclic', tol=0.0001, warm_start=False))])
```

You can retrieve a specific iteration from a run.

```
In [17]: # show run and model from iteration 3
         iteration = 3
         third_run, third_model = remote_run.get_output(iteration=iteration)
         print(third_run)
         print(third_model)
```

```
Run(Experiment: automl-regression,
Id: AutoML_27486c43-de93-440b-b614-d8d63bb7da8d_3,
Type: azureml.scriptrun,
Status: Completed)
Pipeline(memory=None,
     steps=[('datatransformer', DataTransformer(logger=None, task=None)), ('p
refittedsoftvotingregressor', PreFittedSoftVotingRegressor(estimators=[('Elas
ticNet', Pipeline(memory=None,
     steps=[('standardscalerwrapper', <automl.client.core.common.model_wrappe
rs.StandardScalerWrapper object at 0x7f82f9..., tol=0.0001, warm_start=Fals
e))]))],
               flatten_transform=None, weights=[0.25, 0.75]))])
```

At this point you now have a model you could use for predicting the time until battery failure. You would typically use this model in one of two ways:

- Use the model file within other notebooks to batch score predictions.
- Deploy the model file as a web service that applications can call.

In the following, you will explore the latter option to deploy the best model as a web service.

Download the best model

With a run object in hand, it is trivial to download the model.

In [18]:
```
# fetch the best model
best_run.download_file("outputs/model.pkl",
                       output_file_path = "./model.pkl")
```

Deploy the Model as a Web Service

Azure Machine Learning provides a Model Registry that acts like a version controlled repository for each of your trained models. To version a model, you use the SDK as follows. Run the following cell to register the best model with Azure Machine Learning.

In [19]:
```
# register the model for deployment
model = Model.register(model_path = "model.pkl",
                       model_name = "model.pkl",
                       tags = {'area': "auto", 'type': "regression"},
                       description = "Contoso Auto model to predict battery fa
ilure",
                       workspace = ws)

print(model.name, model.description, model.version)
```

```
Registering model model.pkl
model.pkl Contoso Auto model to predict battery failure 5
```

Once you have a model added to the registry in this way, you can deploy web services that pull their model directly from this repository when they first start up.

Create Scoring File

Azure Machine Learning SDK gives you control over the logic of the web service, so that you can define how it retrieves the model and how the model is used for scoring. This is an important bit of flexibility. For example, you often have to prepare any input data before sending it to your model for scoring. You can define this data preparation logic (as well as the model loading approach) in the scoring file.

Run the following cell to create a scoring file that will be included in the Docker Image that contains your deployed web service.

```
In [20]: %%writefile score.py
         import pickle
         import json
         import numpy
         import pandas as pd
         import azureml.train.automl
         from sklearn.externals import joblib
         from azureml.core.model import Model

         def init():
             global model
             model_path = Model.get_model_path('model.pkl') # this name is model.id of
          model that we want to deploy
             # deserialize the model file back into a sklearn model
             model = joblib.load(model_path)

         def run(rawdata):
             try:
                 data = pd.read_json(rawdata,orient="split")
                 result = model.predict(data)
             except Exception as e:
                 result = str(e)
                 return json.dumps({"error": result})
             return json.dumps({"result":result.tolist()})
```

```
Overwriting score.py
```

Create Environment Dependency File

When you deploy a model as web service to either Azure Container Instance or Azure Kubernetes Service, you are deploying a Docker container. The first steps towards deploying involve defining the contents of that container. In the following cell, you create Conda Dependencies YAML file that describes what Python packages need to be installed in the container- in this case you specify scikit-learn, numpy, pandas and the Azure ML SDK.

Execute the following cell. The output will show the conda dependencies file content.

```
In [21]: from azureml.core.conda_dependencies import CondaDependencies

         myenv = CondaDependencies.create(conda_packages=['numpy','pandas','scikit-lear
         n'],pip_packages=['azureml-sdk[notebooks,automl]'])
         print(myenv.serialize_to_string())

         with open("myenv.yml","w") as f:
             f.write(myenv.serialize_to_string())
```

```
# Conda environment specification. The dependencies defined in this file will
# be automatically provisioned for runs with userManagedDependencies=False.

# Details about the Conda environment file format:
# https://conda.io/docs/user-guide/tasks/manage-environments.html#create-env-
file-manually

name: project_environment
dependencies:
  # The python interpreter version.
  # Currently Azure ML only supports 3.5.2 and later.
- python=3.6.2

- pip:
  - azureml-sdk[notebooks,automl]==1.0.17
- numpy
- pandas
- scikit-learn
```

Create Container Image

To create a Container Image, you need three things: the scoring script file, the runtime configuration (defining whether Python or PySpark should be used) and the Conda Dependencies file. Calling `ContainerImage.image_configuration` will capture all of the container image configuration in a single object. This Image will be stored in an instance of Azure Container Registry that is associated with your Azure Machine Learning Workspace.

Execute the following cell (this step will take **a few minutes** as your container image is created remotely).

```
In [22]: from azureml.core.image import ContainerImage

         image_config = ContainerImage.image_configuration(execution_script = "score.p
         y",
                                                           runtime = "python",
                                                           conda_file = "myenv.yml",
                                                           description = "Image with re
         gression model",
                                                           tags = {'area': "auto", 'typ
         e': "regression"}
                                                          )

         image = ContainerImage.create(name = "contosoimage",
                                       # this is the model object
                                       models = [model],
                                       image_config = image_config,
                                       workspace = ws)

         image.wait_for_creation(show_output = True)
```

```
Creating image
Running...............................................................
SucceededImage creation operation finished for image contosoimage:4, operatio
n "Succeeded"
```

Attach to AKS Compute Cluster

With the Container Image configuration in hand, you are almost ready to deploy to AKS. You need to first either provision an AKS cluster or attach to an existing one.

You can provision an AKS cluster from a notebook using the Pyton SDK. As this step will take about 15-20 minutes to create a new cluster, one has already been provided for you. The following code is for your reference when you need to create a cluster.

Use the default configuration (can also provide parameters to customize)

prov_config = AksCompute.provisioning_configuration(location='westus2')

Create the cluster

aks_target = ComputeTarget.create(workspace = ws, name = aks_cluster_name, provisioning_configuration = prov_config)

aks_target.wait_for_completion(True)

print("state:" + aks_target.provisioning_state)

print(aks_target.provisioning_errors)

Execute the following cell to attach to the existing AKS cluster.

```
In [42]: attach_config = AksCompute.attach_configuration(resource_group=aks_resource_gr
         oup_name, cluster_name=aks_cluster_name)
         aks_target = ComputeTarget.attach(workspace = ws, name = aks_cluster_name, att
         ach_configuration = attach_config)
         print("Cluster status is: '{status}'".format(status=aks_target.get_status()))
```

```
Cluster status is: 'Creating'
```

Deploy AKS Hosted Web Service

Now you are ready to deploy your web service to the AKS cluster. To deploy the container that operationalizes your model as a webservice, you can use `Webservice.deploy_from_image` which will use your registered Docker Image, pulling it from the Container Registry, and run the created container in AKS.

Notice in the creation of aks_config, `collect_model_data` and `enable_app_insights` are both enabled. Model data can be collected for the purposes of monitoring the model in production- the inputs to the model and the predictions that result are logged to Azure Storage Blobs. The model data can analyzed using Power BI or Azure Databricks, or any other tool that can process CSV files. This is important for you to be able to keep tabs on the model in production and spot signals that might mean you need to retrain the model. Application Insights will collect diagnostic telemetry such as request rates, response times, failure rates and errors. This telemetry can be monitored and explored using the Application Insights instance accessed within the Azure Portal.

Modify the `aks_service_name` value in the following cell to replace `YOUR-UNIQUE-IDENTIFIER` with your assigned value.

Execute the following cell to deploy your webservice to AKS. This step will take **2-3 minutes** to complete.

In [44]:
```
%%time
aks_service_name ='aks-automl-service-YOUR-UNIQUE-IDENTIFIER'

aks_config = AksWebservice.deploy_configuration(collect_model_data=True, enabl
e_app_insights=True)
aks_service = Webservice.deploy_from_image(workspace = ws,
                                           name = aks_service_name,
                                           image = image,
                                           deployment_config = aks_config,
                                           deployment_target = aks_target
                                           )
aks_service.wait_for_deployment(show_output = True)
print(aks_service.state)
```

```
Creating service
Running................
SucceededAKS service creation operation finished, operation "Succeeded"
Healthy
CPU times: user 1.47 s, sys: 94.3 ms, total: 1.56 s
Wall time: 1min 40s
```

Test the deployed web service

With the deployed web service ready, you are now ready to test calling the service with some car telemetry to see the scored results. There are three ways to approach this:

1. You could use the `aks_service` object that you acquired in the previous cell to call the service directly.
2. You could use the `Webservice` class to get a reference to a deployed web service by name.
3. You could use any client capable of making a REST call.

In this notebook, we will take the second approach. Run the following cells to retrieve the web service by name and then to invoke it using some sample car telemetry.

The output of this cell will be an array of numbers, where each number represents the expected battery lifetime in days for the corresponding row of vehicle data.

In [45]:
```
%%time
# connect to the deployed webservice
aks_service = Webservice(ws,aks_service_name)

# load some test vehicle data that the model has not seen
test_data = pd.read_csv(test_data_url)

# prepare the data and select five vehicles
test_data = test_data.drop(columns=["Car_ID", "Battery_Age"])
test_data.rename(columns={'Twelve_hourly_temperature_forecast_for_next_31_days
_reversed': 'Twelve_hourly_temperature_history_for_last_31_days_before_death_l
ast_recording_first'}, inplace=True)
test_data_json = test_data.iloc[:5].to_json(orient="split")
prediction = aks_service.run(input_data = test_data_json)
print(prediction)
```

```
{"result": [1549.9914837800693, 1357.4299701332193, 1479.2195258541233, 1709.
3394216265456, 1738.7507897341916]}
CPU times: user 243 ms, sys: 17.4 ms, total: 260 ms
Wall time: 14.3 s
```