ML Platform Notebooks

The ML platform sample notebooks are intended to provide users with an on-ramp to using Azure Machine Learning. Currently our collection of notebooks was written initially with the purpose of demonstrating individual features and at some point, also to test and validate these features. As a result, the way in which the notebooks are delivered to the users represents more of our internal organizational structure rather than a cohesive view of what user can accomplish with our platform and how they can do this.

In contrast the AWS SageMaker notebook repository is structured with a customer first approach, inviting the customer to learn about machine learning and what it does, using SageMaker almost as a coincidental tool that can conveniently accomplish the task. This collegial and welcoming and again, customer first, approach makes SageMaker feel much more trustworthy as they *primarily* discuss solutions and problems, and *secondarily* how to use their products.

Simply put, as notebooks are the primary interface with which users will adopt Azure Machine Learning, we need to change our approach to be more of the same.

This means that we need to redesign how our notebooks are structured and laid out, and how they are written as well. While there is still room for notebooks that focus on the “how” aspect of specific features, these should be in the minority and relegated to “advanced topics”. Mainstream features should always be demonstrated in the scope of a scenario that demonstrates the benefits of machine learning.

Developing such notebooks will take time and effort and will be an ongoing task for essentially as long as we build and maintain the ML Platform. This plan is designed to allow us to quickly move forward with notebook development.

**Contents**

Organizational Approach 2

The README 2

Notebooks 3

File Organization 3

Notebook Content 3

Prose Style and Length 4

Code Descriptions 4

Notebook Outline 5

Embedded Code 6

Monitoring and Metrics 7

Execution Plan 7

# Organizational Approach

The approach for organizing the repository of notebooks will be to organize in high-level customer focused folders rather than low-level feature focused folders. Each folder will have a narrative README.md that describe where the user should go from that point. At the root level of the hierarchy, the README may direct the user to interesting topics and samples, but at more local levels, each significant item in the folder will be described.

The folders in the root of the repository will have customer focused names that should be self-explanatory to the user. For example, these names could include:

* introduction-to-machine-learning
* applying-machine-learning-algorithms
* how-to-use-the-AML-sdk
* automated-machine-learning
* deploying-and-monitoring-machine-learning-models
* advanced-features
* Etc.

*Note that these are not suggested names, rather samples of the types of names to use that show customer focus rather than product feature focus.*

The root readme describes the purpose of each folder with direct links to subfolders that contain interesting samples. Note that the folder names are customer focused and not explicitly numbered. If there is a preferred ordering for a customer to execute samples, this is indicated in the README and not by assuming a particular directory sort order by name – e.g. numbering.

Due to Azure Machine Learning’s ability to run in any notebook environment, the root folder also contains the configuration notebook required to ease connection to the Azure Machine Learning workspace. This is described in the README and in each notebook.

In each topic folder there is a collection of scenario folders and a readme. Each scenario folder is named with a customer focused name describing the purpose of the scenario – note that this does not need to be over-thought, as the README will describe the scenario in narrative format. A “scenario” is a solution that the customer can follow that consists of one or more notebooks. Departing from previous guidance, it is allowed for notebooks to be dependent upon other notebooks in the same folder. Additional steps are required to indicate the correct ordering of notebook execution for automation, which will be covered below.

# The README

Since folders in github are presented with contents first and only commit comments for information, all folders with more than a single notebook is required to have a README.md file. The readme needs to start with a title, a short description of the purpose of the folder, and links to relevant content with meaningful descriptions. If any order is required, this is laid out clearly in the readme.

For example,

If the README is in a folder containing subfolders – e.g. a topic folder, links are directed to the folder containing the sample. If the README is in a folder containing notebooks – e.g. a multi-part sample – link directly to the ipnyb files.

Advanced Samples

This folder contains examples of advanced Azure Machine Learning features

* Distributed Hyperparameter Tuning shows how to execute the Keras MNIST sample across multiple machines in a cluster
* Spark Streaming shows how to inference in real time on Spark.

The top level README will highlight samples from throughout the hierarchy and may not contain links to all samples.

# Notebooks

It goes without saying that all Azure Machine Learning notebook scenarios will use Azure Machine Learning. It may be necessary to have setup notebooks that allocate non-AML resources that are useful in conjunction with Azure Machine Learning – e.g. HDI, Databricks, etc – but these notebooks should only be present alongside notebooks that demonstrate how to use these resources in the context of AML.

Notebooks will be named with a meaningful name based on their purpose. In general, notebooks should have the same name as their parent folder except in the case of multi-step examples requiring multiple notebooks. Notebook names should be lower case with - separators, e.g. this-is-my-notebook.ipynb.

## File Organization

Notebooks and all related files should be in the same folder unless explicitly required for the example. For example, if a notebook train\_model.ipnyb submits a train.py file both of these files should be in the same directory. The notebook **should not** create subdirectories to copy files to submit, unless, of course, explicitly required for the sample execution. By keeping the notebook in the same folder as the rest of the experiment, the notebook is captured in Run History, so the user can rerun the experiment in a meaningful way.

In rare cases where the notebook directory has large files that a user reasonably wouldn’t want to snapshot – e.g. data files that are included in the notebook directory for convenience, or documentation files included for the same reason. The notebook author should include an **.amlignore** file listing files and directories not to be snapshotted.

## Notebook Content

The purpose of the notebook is to educate the user and encouraging them to continue using AML. By adopting a common format and style for notebook creation we can promote these goals by helping the user orient themselves and encourage sample completion and continuation.

## Prose Style and Length

First, all notebook authors need to read and internalize the [Microsoft Style Guide](https://aka.ms/style) – this has been vetted over the year’s and contains a wealth of condensed knowledge.

Notebook authors should understand that users reading our notebooks are not familiar with our product. Concepts that are clear to us are not clear to our users and need extra explanation with an inviting tone. Many of our API’s require several parameters and it is unclear to a novice user how to make the right choices to achieve the results they desire. Therefore, notebooks should have an approximately 4:1 ratio of prose to code as measured by screen real estate. That is, for every 1 inch of code, there should be 4 inches of prose.

## Code Descriptions

Markdown sections prefacing AML SDK API’s need to be very descriptive on the specific API’s that are being called. Each parameter should be explicitly called out, with details on what the parameter means and why the user would choose any particular value. Exceptions are for API’s that are standard python – e.g. collections, and API’s that are used multiple times and should be described in a dedicated notebook – e.g. run.wait\_for\_completion or Workspace.load\_from\_config.

It’s a good idea to add a link to the SDK Reference wherever possible. If not appropriate in the context, it is always appropriate to drop referral links in the “Next Steps” section of the notebook.

For example:

Let’s create our Azure Machine Learning services workspace. To create a workspace you need to either be an owner of the subscription, or the owner of the Azure resource group in which you are creating the workspace. You can learn more about Azure resource groups here.

You need several pieces of information to create the workspace:

* **name** – this is the workspace name. Pick a name that is easy to remember and is relevant to the organization, e.g. based on a person or department’s name. The workspace name is unique within the subscription and can contain letters, numbers, and the underscore character (\_). The maximum length of the name is 30 characters.
* **subscription\_id** – this is the id of the subscription to contain the workspace and represents the billing for all costs associated with the workspace. We find our subscription id in the Azure Portal**.**
* **resource\_group** – This is the name of the resource group that you own that will contain the workspace and all related Azure objects that are created. See below on how to automatically create a resource group.
* **location** – this is the location in the world where our workspace will be located. We find the list of Azure regions here. If possible, it’s best to choose a region that is close that contains our data, but we may also choose specific regions due to availability of other resources such as GPU machines.
* **create\_resource\_group** – set this to *True* if we are the subscription owner and want the create method to also create a resource group for the workspace using the name given above. This is simply a shortcut so we don’t have to explicitly create the resource group elsewhere.
* **exist\_ok** – set this to *True* if we don’t want create to throw an exception if the workspace already exists. Create will still fail if the existing workspace has a different location or resource group.

In this example, we start off introducing the why of the following code snippet by inviting the user with the pronoun “*we*”. All prose is in an “active” tense using direct language – minimizing words such as *would, could* or *should*, stating “*You need”* instead of “*are needed*”*.* Additionally avoid superlatives such as “simple”, “easy”, “fast” etc – if the process is simple, easy, or fast it will speak for itself. The prose describing the function call provides direct information that assists the user in making valid choices for all parameters – e.g. how do I pick a name for a workspace? Why would I pick a particular location? Additionally, the length of the prose is much longer than the code it describes.

# import the Workspace class and check the azureml SDK version  
from azureml.core import Workspace

ws = Workspace.create(name = workspace\_name,  
 subscription\_id = subscription\_id,  
 resource\_group = resource\_group,   
 location = workspace\_region,  
 create\_resource\_group = True,  
 exist\_ok = True)

# Write the configuration to a local file for other notebooks to use  
ws.write\_config()

## Static Images

Static images in your notebook, if any, can be either directly in the notebook folder or in an ./img subfolder. Any static images require alt-text for accessibility.

## Notebook Outline

Each notebook will have these required sections:

* Copyright notice
  + Copyright (c) Microsoft Corporation. All rights reserved.  
    Licensed under the MIT License.
* Title
  + Markdown: # Title
  + *Note there is a space between the # and the first word*
* Subtitle
  + Markdown: \_\*\*Subtitle\*\*\_
  + *Note there is* ***no space*** *between the \* and the text*
* Two horizontal lines
  + Horizontal line markdown: carriage return, ---, carriage return
  + Each subsequent top level section separated by a single horizontal line
* Table of contents
  + Start each line with markdown “1.” Markdown will automatically increment numbers
  + Indent for subsections
  + Use ‘%20’ for spaces in the links
  + Each top level section are linked – subsection linking is optional
    - Markdown: 1. [Section Name](#Section%20Name)  
       1. [Next Section Name](#Next%20Section%20Name)  
       1. [Sub Section Name]
* Introduction
  + Descriptive prose describing what the user will learn and accomplish by going through the notebook. If specific technologies algorithms are used, describe or link to meaningful descriptions.
* Setup
  + Indicate the user should run the configuration notebook if not done and provide link
  + Indicate any required notebooks that need to be run prior to this notebook and provide links
  + Importing the aml sdk and comparing the version to that used to create the notebook.
  + Access the AML workspace
  + Specify variables that could be configured through environment variables
  + Perform any additional notebook specific setup
* Optional Sections separated by horizontal lines (see below)
* Next Steps

Optional sections obviously depend on the nature of the notebook, but where possible need to have consistent headings. Common optional sections are:

* Data
* Train
* Deploy

Sample header markdown: (nb: this is imperfect with respect to carriage returns due to Word formatting)

Copyright (c) Microsoft Corporation. All rights reserved.

Licensed under the MIT License.

# My Title

\_\*\*My Subtitle\*\*\_

---

---

## Contents

1. [Introduction](#Introduction)  
1. [Setup](#Setup)  
 1. Read Environment Values  
 1. Connect to Workspace  
1. [Data](#Data)  
1. [Train](#Train)  
1. [Deploy](#Deploy)  
1. [Next Steps](#Next%20Steps)

---

## Embedded Code

We will not have any code in any notebooks that write static code or files to disk. If a notebook requires a static code or other file, this file will be provided with the notebook. If it is important for the user to read the contents of the file, the file will be linked in the prose describing the operation where it is used. If it is important for the user to *understand* portions or all the file, this is also be described in the prose of the notebook and not presented in a cell as a writefile operation.

Using writefile magics in notebooks is limited to the rare case where a file needs to be customized for the particular use case of the notebook. In this instance we should consider improving our SDK API to avoid magic files.

## Environment Variables

If there are external variables that are required for the notebook to run, the preferred mechanism is to use environment variables. This is sometimes necessary to enable testing of the notebook in our testing infrastructure and can also be useful in instructing users how to automate their notebook processes.

If your notebook requires environment variables, create a “Read environment variables” section underneath the “Setup” section. This section should describe the variables that are being read and how the user should set values. All required environment variables are read in this section and set to variables. To read environment variables use os.getenv() and make sure to use the syntax “default=” to make it clear that the second parameter is the value that will be used.

For example:

import os

subscription\_id = os.getenv(“SUBSCRIPTION\_ID”, default=“<your-subscription-id>”)  
resource\_group = os.getenv("RESOURCE\_GROUP", default=“<your-resource-group>”)  
workspace\_name = os.getenv("WORKSPACE\_NAME", default=“<your-workspace-name>”)  
workspace\_region = os.getenv("WORKSPACE\_REGION", default=“<your-region>”)

**Setup**

**Read Environment Variables**

We use environment variables to set some notebook variables for potential automation. In most cases we set these manually using values relevant to our usage. If the environment variable isn’t set, the python variable is set to the default value in the cell. If you are not using environment variables, replace the default values in the following cell with the values that will allow the notebook to work in your specific case.

subscription\_id – this is the ID of your Azure subscription  
resource\_group – the name of the resource group in which to create the workspace  
workspace\_name – the name to provide to create your workspace  
workspace\_region – the location in which to create the workspace

# Monitoring and Metrics

Given the nature of notebooks and where they are executed, it is generally difficult to track if and how users are successful with individual notebooks. However, we can track subscriptions that contain uniquely named objects and use the presence of these object as a proxy for notebook usage.

Notebooks that create object should use statistically unique names. A statistically unique name does not need to be “weird”, rather it needs to be descriptive and non-generic. For example, “Fraud Analysis with Keras” instead of “My Experiment.” Objects that should have unique and meaningful names include: Experiments, Images, Models, etc.

We will add to our metric tracking the number of subscriptions that contain each of these object names – a registry of names that we look for will be provided.

Success will be measured by

* Growth in usage of a notebook – “usage of this notebook grew 20%”
* Retention of users using a notebook – “customers using this notebook have 2d, 10d, 2w, 4w trends of X vs overall trends of Y”
* Clustering of notebooks – e.g. how many other notebooks are also used by users of a target notebook – “customers using notebook X also used notebooks A,B,C,D”

# Execution Plan

Changing from where we are to where we need to be will be a multi-phase effort. Once we have some example notebooks converted to the desired format and content we can distribute the remaining work. Although this work is urgent in that our only organic doorway to customers is through the notebooks we provide, it is also incremental in that as we improve we will receive value at that time.

Phase 1 should be completed prior to GA. Phase 2 will be an ongoing effort and needs to be planned for individual notebooks

Phase 1

* Rename all notebooks starting with index numbers
* Create the high level folder structure and place notebooks inside
* Rewrite 1-2 notebooks using the new style guide
* Create top level README guiding the user to notable notebooks

Phase 2

* Prioritize and assign notebooks to be rewritten
* Create README’s for each significant subfolder