**Azure Machine Learning Workbench**

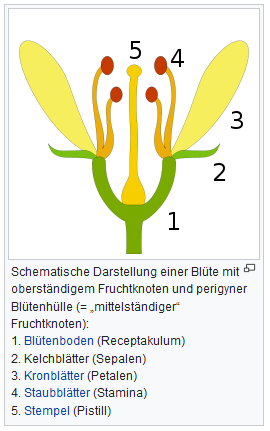
* **Install and start (up to 30 Minutes to complete)**

<https://docs.microsoft.com/de-de/azure/machine-learning/service/quickstart-installation>

(Installation of workbench, first project, first python script)

* **Iris (Schwertlilien) Dataset**

see <https://en.wikipedia.org/wiki/Iris_flower_data_set>

The Iris flower data set or Fisher's Iris data set is a multivariate data set introduced by the British statistician and biologist Ronald Fisher. The data set consists of 50 samples from each of **three species of Iris** :

* + Iris **setosa**,
  + Iris **virginica**
  + and Iris **versicolor**

**Four features** were measured from each sample:

* + the **length**
  + and the **width**
  + of the **sepals** (Kelchblatt)
  + and **petals** (Kronblatt),

in centimeters. Based on the combination of these

four features, Fisher developed **a linear discriminant**

**model** to **distinguish the species from each other**.

* **Video: How to work with the run history**

<https://www.youtube.com/watch?v=9GAiEKaNCtk&feature=youtu.be>

* **Click on the Home Page in AML Workbench and execute the commands under „Quick CLI references“ (proceed the first tutorial „Install and start“ )**

First, launch the Command Prompt or Powershell from the File menu. Then enter the following commands:

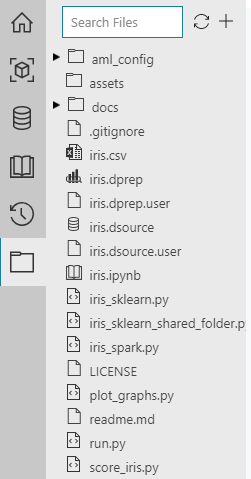
**# first let's install matplotlib locally**

**$ pip install matplotlib**

**# log in to Azure if you haven't done so**

**$ az login**

**# kick off many local runs sequentially**

**$ python run.py**

The command above will execute the following script:

**# run iris\_sklearn.py with descending regularization rates**

**# run this with just "python run.py". It will fail if you run using az ml execute.**

**import os**

**# set regularization rate as an argument**

**reg = 10**

**while reg > 0.005:**

**os.system('az ml experiment submit -c local ./iris\_sklearn.py {}'.format(reg))**

**# cut regularization rate to half**

**reg = reg / 2**

Run iris\_sklearn.py Python script **in local Python** environment. (Docker will be installed later)

**$ az ml experiment submit -c local iris\_sklearn.py**

This command will do the following steps (look into the source code):

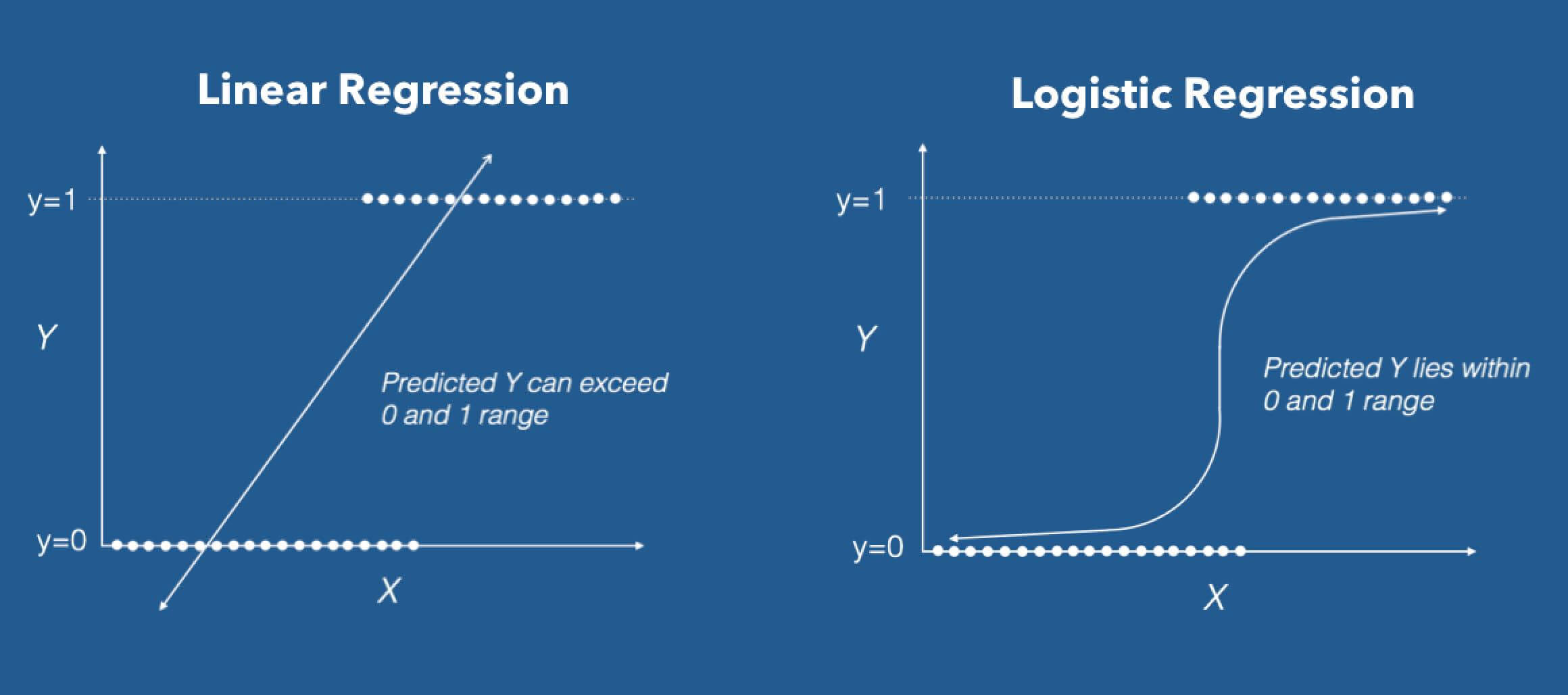
* load Iris dataset from a DataPrep package as a pandas DataFrame
* add n more random features to make the problem harder to solve
* change regularization rate and you will likely get a different accuracy
* evaluate the test set
* log accuracy which is a single numerical value
* calculate and log precision, recall, and thresholds, which are list of numerical values
* serialize the model on disk in the special 'outputs' folder
* load the model back from the 'outputs' folder into memory
* predict on a new sample
* add random features to match the training data
* score on the new sample

Check the

Runs- Page  für additional runs

* **What is the logistic regression model?**

<https://dzone.com/articles/machinex-simplifying-logistic-regression>

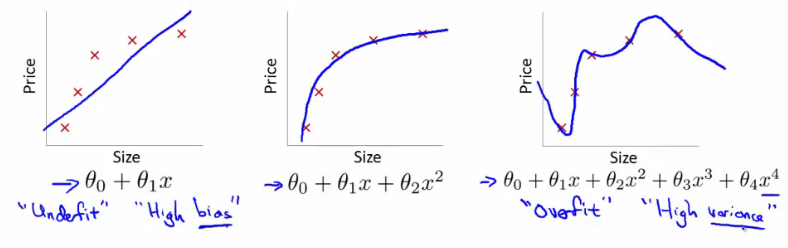


*Logistic regression is one of the most popular machine learning algorithms for binary classification. This is because it is a simple algorithm that performs very well on a wide range of problems. It is used* ***when you know that the data is linearly separable/classifiable*** *and the outcome is* binary *or* dichotomous *but it can be extended when the dependent has more than two categories.* ***Logistic regression is used when the outcome is a discrete variable****, such as trying to figure out who will win the election,* ***whether a student will pass an exam*** *or not, or* ***whether an email is a spam****. This is commonly* ***called as a classification problem*** *because we are trying to determine which class the dataset best fits.*

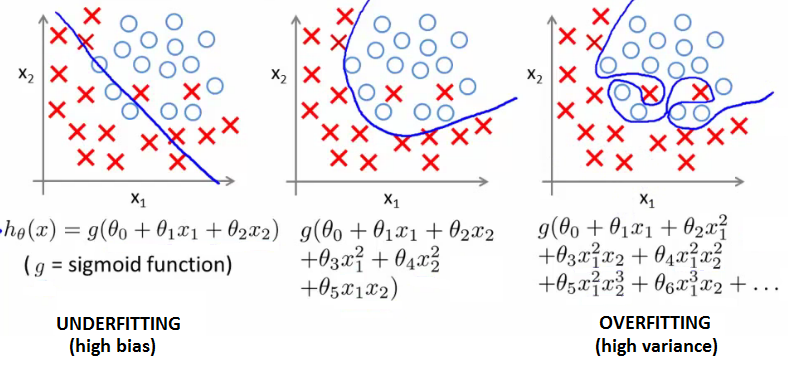
**What ist the meaning of the argument: „regularization rate“**

Visit: <http://www.holehouse.org/mlclass/07_Regularization.html>

**Underfit and Overfit**

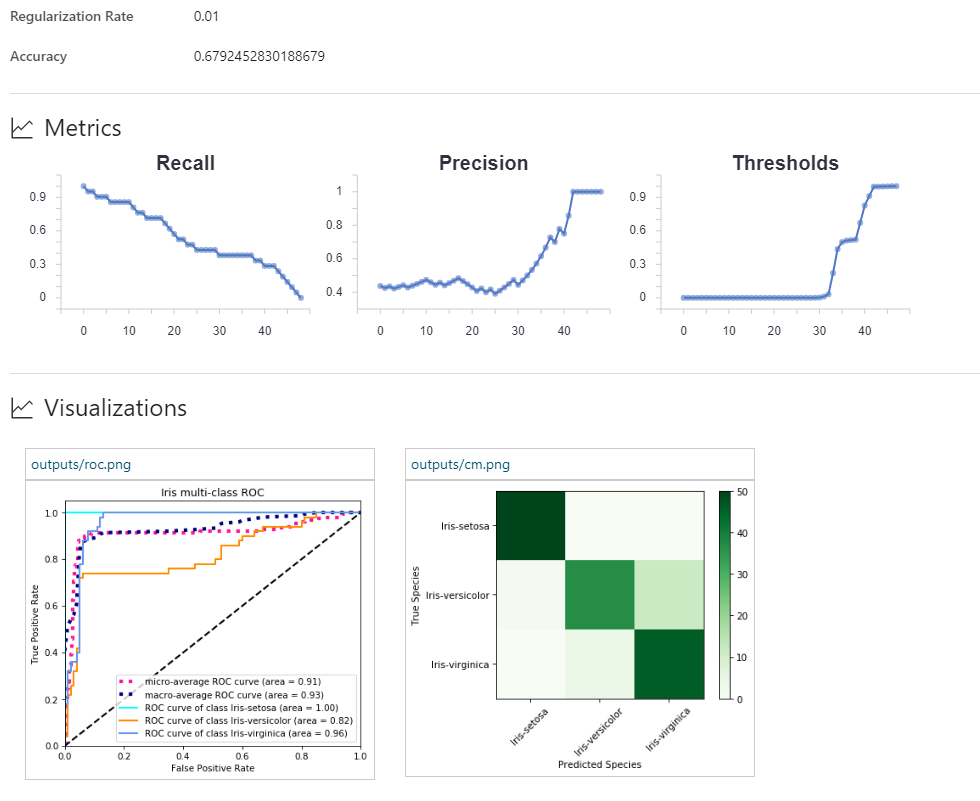


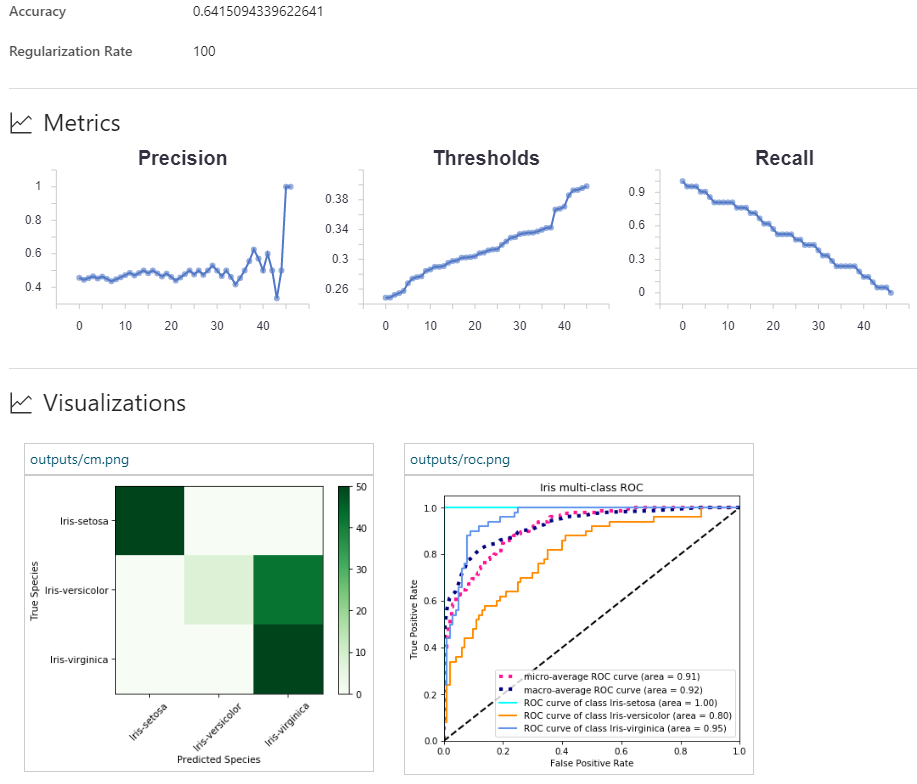
**Overfitting with logistic regression**

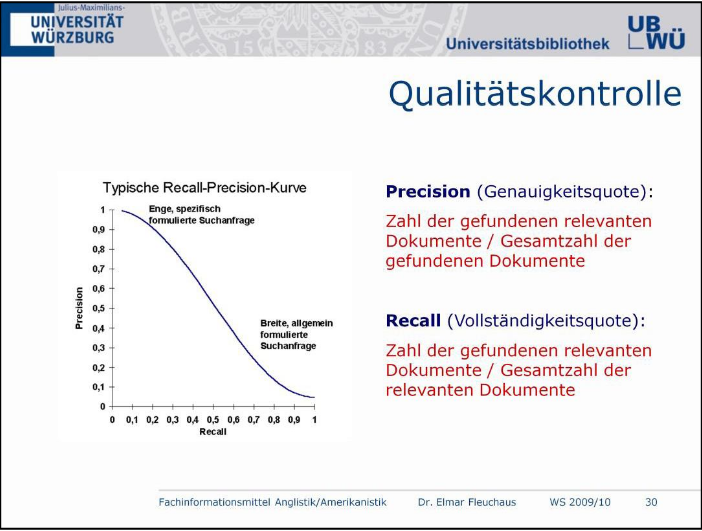


**Addressing / preventing overfitting:**

* Eiter reduce number of features
* **or work with** **regularization**:
  + This keeps all features, but reduces magnitude of parameters θ
  + It works well when we have a lot of features, each of which contributes a bit to predicting the result
* The **regularization parameter** controls ***a trade off*** between our two goals:
* Want to fit the training set well
* Want to keep parameters small
  + **Regularization parameter** should be chosen carefully - **not too big...**
* **What are Thresholds, Precision and Recall ?**

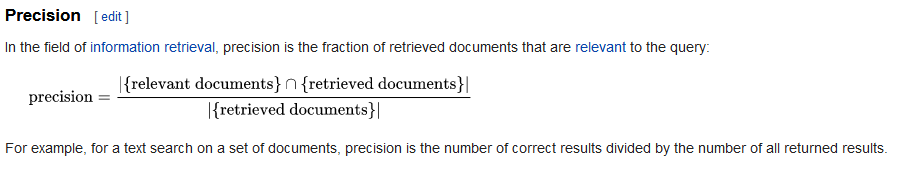


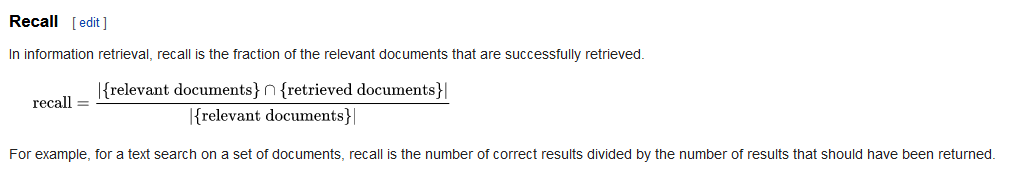




Zwischen den beiden Qualitätskriterien besteht jedoch wie das obige Schaubild zeigt ein Zielkonflikt: Mit steigender Precision sinkt i.d.R. der Recall und umgekehrt. Anders formuliert: Je mehr Wert bei der Recherche auf Genauigkeit gelegt wird, desto größer ist die Gefahr, dass relevante Dokumente nicht gefunden werden. Je mehr Wert auf Vollständigkeit gelegt wird, desto größer ist tendenziell der Ballast an irrelevanten Treffern

<https://en.wikipedia.org/wiki/Precision_and_recall#Precision>





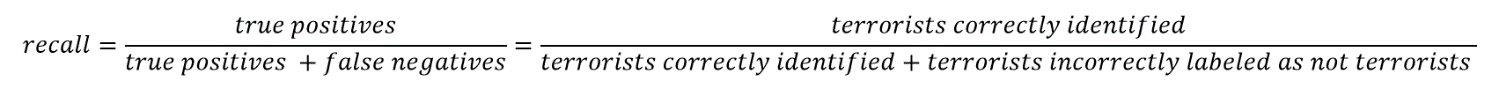
* **Beyond Accuracy: Precision and Recall**

<https://towardsdatascience.com/beyond-accuracy-precision-and-recall-3da06bea9f6c>

Would you believe someone who claimed to create a model entirely in their head to identify terrorists trying to board flights with greater than 99% accuracy? Well, here is the model: simply label every single person flying from a US airport as not a terrorist. Given the [800 million average passengers on US flights per year](https://www.rita.dot.gov/bts/press_releases/bts018_16) and the [19 (confirmed) terrorists who boarded US flights from 2000–2017](https://en.wikipedia.org/wiki/List_of_aircraft_hijackings#2000s), this model achieves an astounding accuracy of 99.9999999%! That might sound impressive, but I have a suspicion the US Department of Homeland Security will not be calling anytime soon to buy this model. While this solution has nearly-perfect accuracy, this problem is one in which accuracy is clearly not an adequate metric!

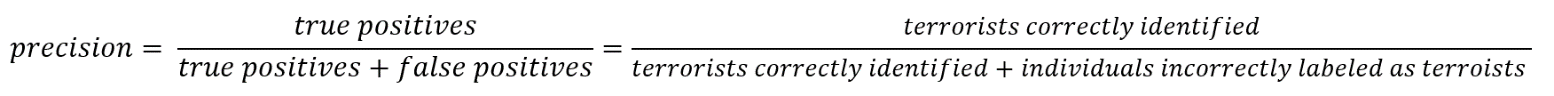
**The terrorist detection task** is an [imbalanced classification problem](https://www.analyticsvidhya.com/blog/2017/03/imbalanced-classification-problem/): we have two classes we need to identify — terrorists and not terrorists — with one category representing the overwhelming majority of the data points. Another imbalanced classification problem occurs in disease detection when the rate of the disease in the public is very low. In both these cases the positive class — disease or terrorist — is greatly outnumbered by the negative class. These types of problems are examples of the fairly common case in data science when accuracy is not a good measure for assessing model performance.

Intuitively, we know that proclaiming all data points as negative in the terrorist detection problem is not helpful and, instead, we should focus on identifying the positive cases. The metric our intuition tells us we should maximize is known in statistics as [recall](https://en.wikipedia.org/wiki/Precision_and_recall), or the ability of a model to find all the relevant cases within a dataset. The precise definition of recall is the number of true positives divided by the number of true positives plus the number of false negatives. True positives are data point classified as positive by the model that actually are positive (meaning they are correct), and false negatives are data points the model identifies as negative that actually are positive (incorrect). In the terrorism case, true positives are correctly identified terrorists, and false negatives would be individuals the model labels as not terrorists that actually were terrorists. Recall can be thought as of a model’s ability to find all the data points of interest in a dataset.

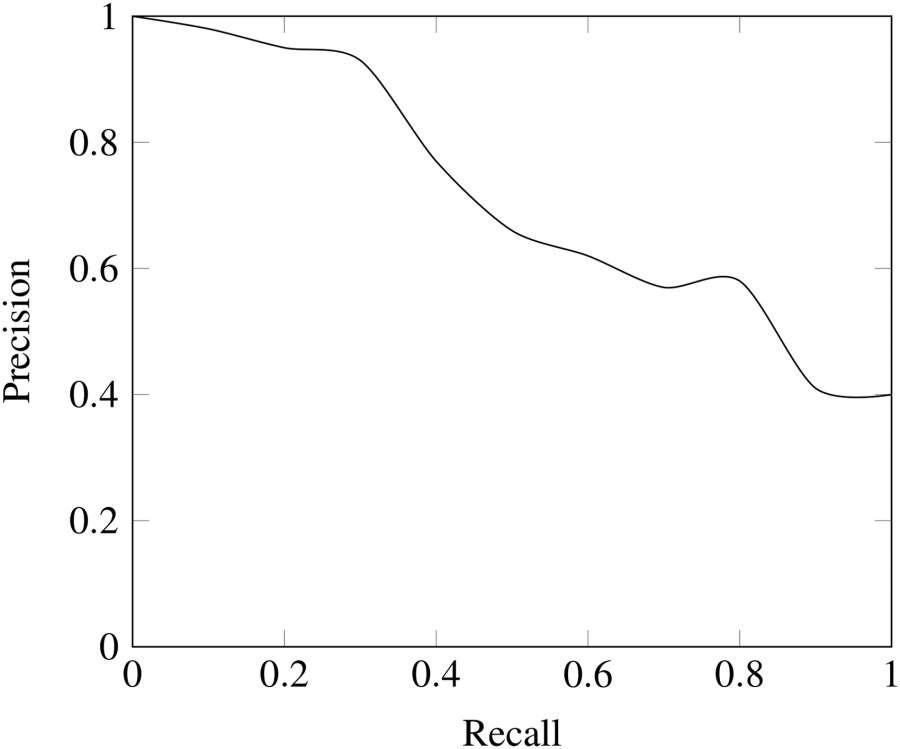


You might notice something about this equation: if we label all individuals as terrorists, then our recall goes to 1.0! We have a perfect classifier right? Well, not exactly. As with most concepts in data science, there is a trade-off in the metrics we choose to maximize. In the case of recall, when we increase the recall, we decrease the precision. Again, we intuitively know that a model that labels 100% of passengers as terrorists is probably not useful because we would then have to ban every single person from flying. Statistics provides us with the vocabulary to express our intuition: this new model would suffer from low [precision](https://en.wikipedia.org/wiki/Precision_and_recall), or the ability of a classification model to identify only the relevant data points.

Precision is defined as the number of true positives divided by the number of true positives plus the number of false positives. False positives are cases the model incorrectly labels as positive that are actually negative, or in our example, individuals the model classifies as terrorists that are not. While recall expresses the ability to find all relevant instances in a dataset, precision expresses the proportion of the data points our model says was relevant actually were relevant.



Now, we can see that our first model which labeled all individuals as not terrorists wasn’t very useful. Although it had near-perfect accuracy, it had 0 precision and 0 recall because there were no true positives! Say we modify the model slightly, and identify a single individual correctly as a terrorist. Now, our precision will be 1.0 (no false positives) but our recall will be very low because we will still have many false negatives. If we go to the other extreme and classify all passengers as terrorists, we will have a recall of 1.0 — we’ll catch every terrorist — but our precision will be very low and we’ll detain many innocent individuals. In other words, as we increase precision we decrease recall and vice-versa.



*The Precision-Recall Trade-off (*[*Source*](http://computingengineering.asmedigitalcollection.asme.org/article.aspx?articleid=2610217)*)*

**Combining Precision and Recall**

In some situations, we might know that we want to maximize either recall or precision at the expense of the other metric. For example, in preliminary disease screening of patients for follow-up examinations, we would probably want a recall near 1.0 — we want to find all patients who actually have the disease — and we can accept a low precision if the cost of the follow-up examination is not significant. However, in cases where we want to find an optimal blend of precision and recall we can combine the two metrics using what is called the [F1 score](https://en.wikipedia.org/wiki/F1_score).

The F1 score is the harmonic mean of precision and recall taking both metrics into account in the following equation:

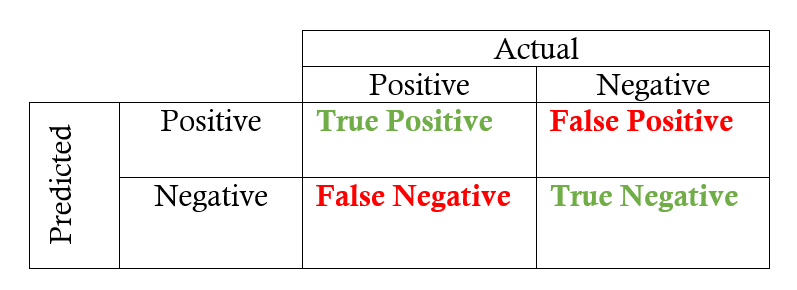


We use the [harmonic mean instead of a simple average because it punishes extreme values](https://stackoverflow.com/questions/26355942/why-is-the-f-measure-a-harmonic-mean-and-not-an-arithmetic-mean-of-the-precision). A classifier with a precision of 1.0 and a recall of 0.0 has a simple average of 0.5 but an F1 score of 0. The F1 score gives equal weight to both measures and is a specific example of the general Fβ metric where β can be adjusted to give more weight to either recall or precision. (There are other metrics for combining precision and recall, such as the [Geometric Mean of precision and recall](https://en.wikipedia.org/wiki/Fowlkes%E2%80%93Mallows_index), but the F1 score is the most commonly used.) If we want to create a balanced classification model with the optimal balance of recall and precision, then we try to maximize the F1 score.

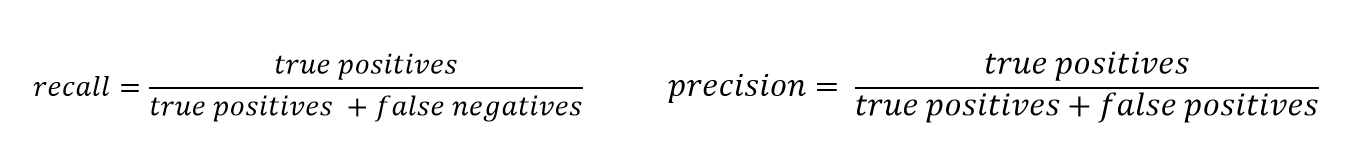
**Visualizing Precision and Recall**

I’ve thrown a couple new terms at you and we’ll walk through an example to show how they are used in practice. Before we can get there though we need to briefly talk about tw concepts used for showing precision and recall.

First up is the [confusion matrix](http://www.dataschool.io/simple-guide-to-confusion-matrix-terminology/) which is useful for quickly calculating precision and recall given the predicted labels from a model. A confusion matrix for binary classification shows the four different outcomes: true positive, false positive, true negative, and false negative. The actual values form the columns, and the predicted values (labels) form the rows. The intersection of the rows and columns show one of the four outcomes. For example, if we predict a data point is positive, but it actually is negative, this is a false positive.

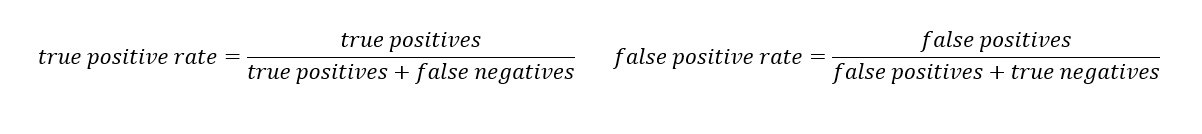


Going from the confusion matrix to the recall and precision requires finding the respective values in the matrix and applying the equations:

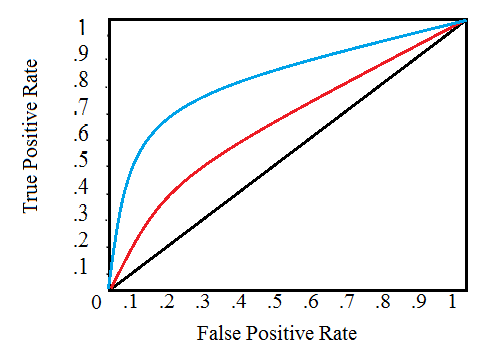


The other main visualization technique for showing the performance of a classification model is the [**Receiver Operating Characteristic (ROC) curve**](http://scikit-learn.org/stable/auto_examples/model_selection/plot_roc.html). Don’t let the complicated name scare you off! The idea is relatively simple: **the ROC curve shows how the recall vs precision relationship changes as we vary the threshold for identifying a positive in our model**. The threshold represents the value above which a data point is considered in the positive class. If we have a model for identifying a disease, our model might output a score for each patient between 0 and 1 and we can set a threshold in this range for labeling a patient as having the disease (a positive label). By altering the threshold, we can try to achieve the right precision vs recall balance.

An **ROC curve** plots the true positive rate on the y-axis versus the false positive rate on the x-axis. The true positive rate (TPR) is the recall and the false positive rate (FPR) is the probability of a false alarm. Both of these can be calculated from the confusion matrix:



A typical ROC curve is shown below:



*Receiver Operating Characteristic Curve (*[*Source*](http://www.statisticshowto.com/c-statistic/)*)*

The black diagonal line indicates a random classifier and the red and blue curves show two different classification models. For a given model, we can only stay on one curve, but we can move along the curve by adjusting our threshold for classifying a positive case. Generally, as we decrease the threshold, we move to the right and upwards along the curve. With a threshold of 1.0, we would be in the lower left of the graph because we identify no data points as positives leading to no *true positives* and no *false positives* (TPR = FPR = 0). As we decrease the threshold, we identify more data points as positive, leading to more true positives, but also more false positives (the TPR and FPR increase). Eventually, at a threshold of 0.0 we identify all data points as positive and find ourselves in the upper right corner of the ROC curve (TPR = FPR = 1.0).

Finally, we can quantify a model’s ROC curve by calculating the total [Area Under the Curve (AUC)](https://en.wikipedia.org/wiki/Receiver_operating_characteristic#Area_under_the_curve), a metric which falls between 0 and 1 with a higher number indicating better classification performance. In the graph above, the AUC for the blue curve will be greater than that for the red curve, meaning the blue model is better at achieving a blend of precision and recall. A random classifier (the black line) achieves an AUC of 0.5.

**Recap**

We’ve covered a few terms, none of which are difficult on their own, but which combined can be a little overwhelming! Let’s do a quick recap and then walk through an example to solidly the new ideas we learned.

**Four Outcomes of Binary Classification**

* **True positives :** data points labeled as positive that are actually positive
* **False positives :** data points labeled as positive that are actually negative
* **True negatives :** data points labeled as negative that are actually negative
* **False negatives :** data points labeled as negative that are actually positive

**Recall and Precision Metrics**

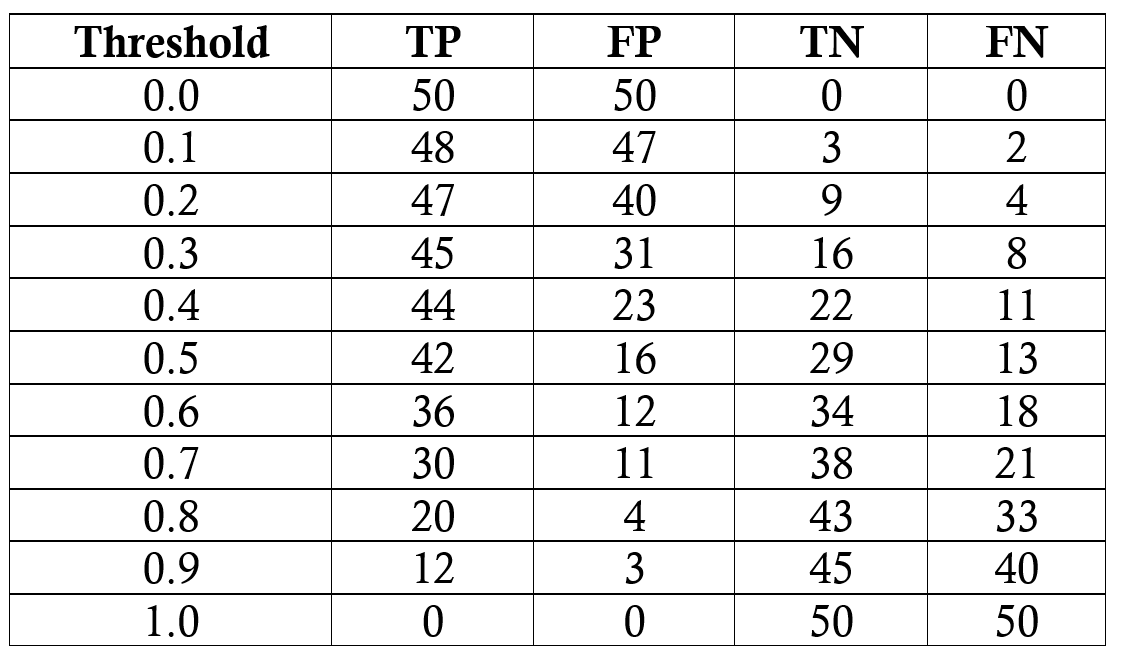
* **Recall:** ability of a classification model to identify all relevant instances
* **Precision:** ability of a classification model to return only relevant instances
* **F1 score:** single metric that combines recall and precision using the harmonic mean

**Visualizing Recall and Precision**

* **Confusion matrix:** shows the actual and predicted labels from a classification problem
* **Receiver operating characteristic (ROC) curve:** plots the true positive rate (TPR) versus the false positive rate (FPR) as a function of the model’s threshold for classifying a positive
* **Area under the curve (AUC):** metric to calculate the overall performance of a classification model based on area under the ROC curve

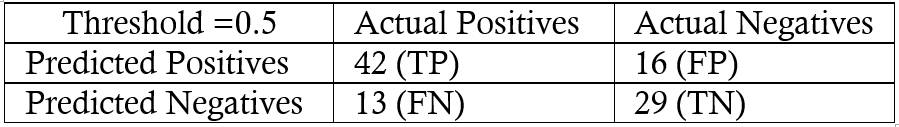
**Example Application**

Our task will be to diagnose 100 patients with a disease present in 50% of the general population. We will assume a black box model, where we put in information about patients and receive a score between 0 and 1. **We can alter the threshold for labeling a patient as positive (has the disease) to maximize the classifier performance**. **We will evaluate thresholds from 0.0 to 1.0 in increments of 0.1, at each step calculating the precision, recall, F1, and location on the ROC curve.** Following are the classification outcomes at each threshold:



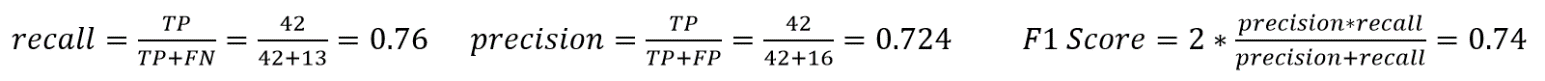
*Outcome of model at each threshold*

We’ll do one sample calculation of the recall, precision, true positive rate, and false positive rate at athreshold of 0.5. First we make the confusion matrix:

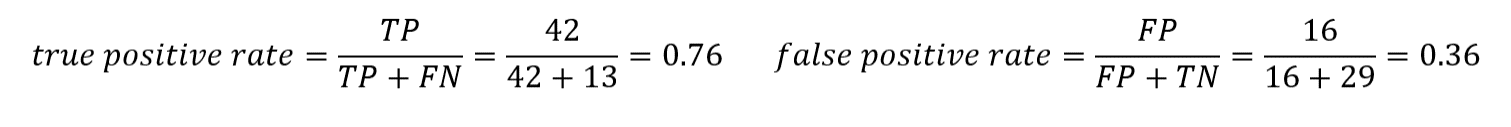


*Confusion Matrix for Threshold of 0.5*

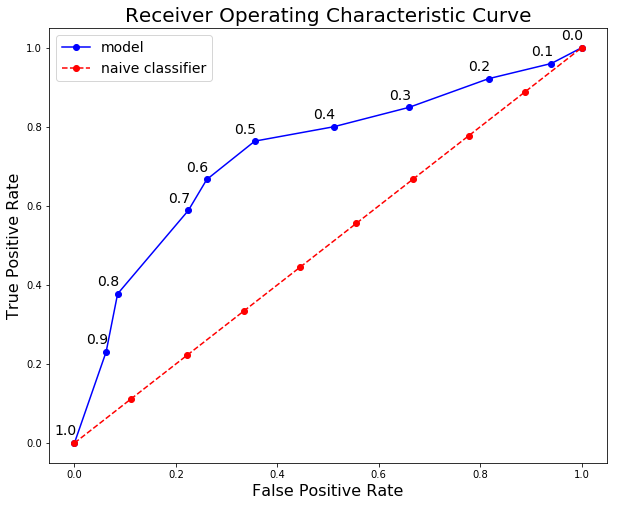
We can use the numbers in the matrix to calculate the recall, precision, and F1 score:



Then we calculate the true positive and false positive rate to find the y and x coordinates for the ROC curve.

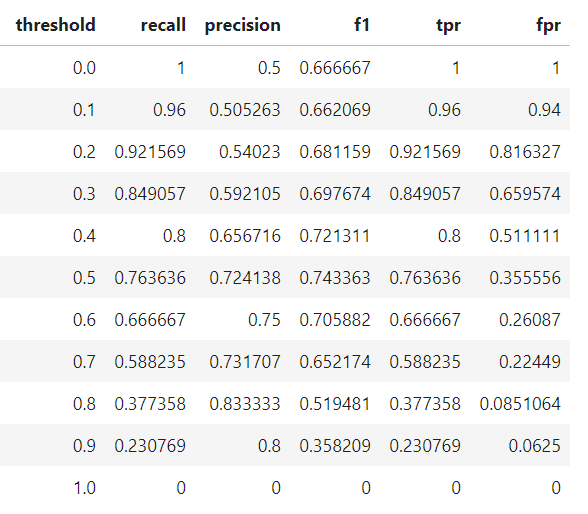


To make the entire ROC curve, we carry out this process at each threshold. As you might think, this is pretty tedious, so instead of doing it by hand, we use a language like Python to do it for us! The [Jupyter Notebook](https://github.com/WillKoehrsen/Data-Analysis/blob/master/recall_precision/recall_precision_example.ipynb) with the calculations is on GitHub for anyone to see the implementation. The final ROC curve is shown below with the thresholds above the points.



Here we can see all the concepts come together! **At a threshold of 1.0, we classify no patients** **as having the disease** and hence have a recall and precision of 0.0. **As the threshold decreases, the recall increases because we identify more patients that have the disease.** However, as our recall increases, our precision decreases because in addition to increasing the *true* positives, we increase the *false* positives. At a threshold of 0.0, our recall is perfect — we find all patients with the disease — but our precision is low because we have many false positives. We can move along the curve for a given model by changing the threshold and select the threshold that maximizes the F1 score. To shift the entire curve, we would need to build a different model.

Final model statistics at each threshold are below:



Based on the F1 score, the overall best model occurs at a threshold of 0.5. If we wanted to emphasize precision or recall to a greater extent, we could choose the corresponding model that performs best on those measures.

**Conclusions**

We tend to use accuracy because everyone has an idea of what it means rather than because it is the best tool for the task! Although better-suited metrics such as recall and precision may seem foreign, we already have an intuitive sense of why they work better for some problems such as imbalanced classification tasks. Statistics provides us with the formal definitions and the equations to calculate these measures. [Data science](https://www.datacamp.com/community/podcast/data-science-astronomy) is about knowing the right tools to use for a job, and often we need to go beyond accuracy when developing classification models. Knowing about recall, precision, F1, and the ROC curve allows us to assess classification models and should make us think skeptically about anyone touting only the accuracy of a model, especially for imbalanced problems. As we have seen, accuracy does not provide a useful assessment on several crucial problems, but now we know how to employ smarter metrics!

* **CLI (Command Line Interface) for Azure Machine Learning (Workbench)**

<https://docs.microsoft.com/en-us/azure/machine-learning/desktop-workbench/cli-for-azure-machine-learning>

* **Azure Machine learning CLI**

<https://docs.microsoft.com/en-us/azure/machine-learning/service/reference-azure-machine-learning-cli>

* **Tutorial 1: Preparing the data (up to 15 Minutes to complete)**

<https://docs.microsoft.com/en-us/azure/machine-learning/desktop-workbench/tutorial-classifying-iris-part-1>

This tutorial is part one of a three-part series. In this tutorial, you walk through the **basics of Azure Machine Learning services (preview)** and learn how to:

* Create a project in Azure Machine Learning Workbench
* Create a data preparation package
* Generate Python/PySpark code to invoke a data preparation package

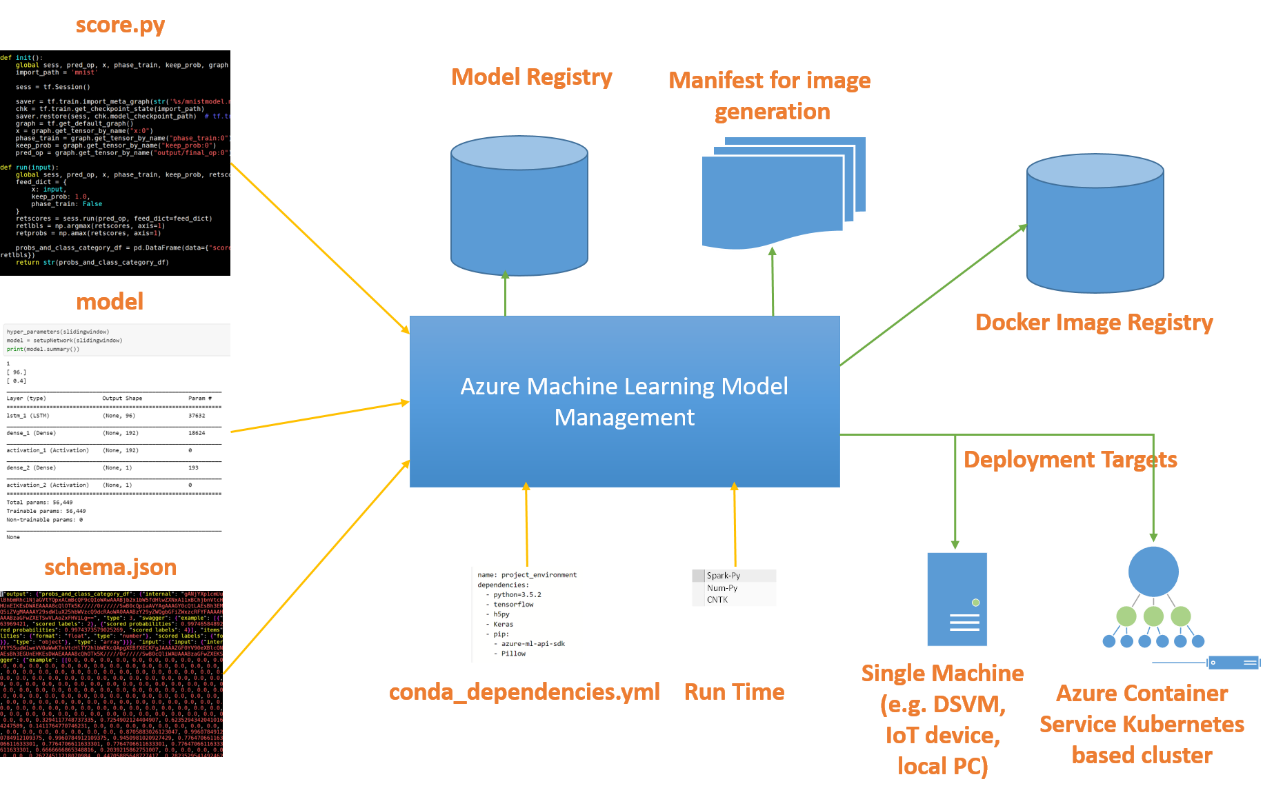
To complete this tutorial, you must have:

* an **Azure Machine Learning Experimentation account**
* an **Azure Machine Learning Workbench installed**
* **Tutorial 2: Classify Iris - Build a model**

<https://docs.microsoft.com/en-us/azure/machine-learning/desktop-workbench/tutorial-classifying-iris-part-2>

This tutorial is part two of a three-part series. In this part of the tutorial, you use Azure Machine Learning services to:

* Open scripts and review code
* Execute scripts in a local environment
* Review run histories
* Execute scripts in a **local Azure CLI window**



To complete this tutorial, you need:

* an **Azure subscription**. If you don't have an Azure subscription, create a free account before you begin.
* an **experimentation account and Azure Machine Learning Workbench installed** as described in this quickstart
* **the project and prepared Iris data from Tutorial part 1**
* **a** **Docker engine installed and running locally**. Docker's Community Edition is sufficient. Learn how to install Docker here: <https://docs.docker.com/engine/installation/>

During the execution of the steps of this tutorial, the installation of the Docker environment will be explained. See the ***Docker tutorial in this Training***.

## Run scripts in the local Docker Environment

No hints

* + - Execute scripts in a **local Docker environment (problems with virtual machines and HyperV)**

## Run scripts in the CLI Window

No hints

## Run scripts remote Docker Container

Read the following hints

* + Execute scripts in a **remote Docker environment**

*To execute your script in a Docker container* ***on a remote Linux machine****, you need to have SSH access (username and password) to that remote machine. In addition, the machine must have a Docker engine installed and running. The easiest way to obtain such a Linux machine is to create an Ubuntu-based Data Science Virtual Machine (DSVM) on Azure. Learn* [*how to create an Ubuntu DSVM to use in Azure ML Workbench*](https://docs.microsoft.com/en-us/azure/machine-learning/desktop-workbench/how-to-create-dsvm-hdi#create-an-ubuntu-dsvm-in-azure-portal)*.*

* + - **Create a DSVM to use** 
      * **Docker as remote targe and**
      * **HDI Spark cluster as compute targets**

<https://docs.microsoft.com/en-us/azure/machine-learning/desktop-workbench/how-to-create-dsvm-hdi>

## Use „Create an Ubuntu DSVM in Azure portal“

## Attach a DSVM compute target

## Run scripts in a remote Docker container

Step 1

**Important Hint: Use Powershell from Azure Machine Learning Workbench and change with command „cd“ to your project directory**

# attach the DSVM compute target

# it is a good idea to use FQDN in case the IP address changes after you deallocate the VM and restart it

$ az ml computetarget attach remotedocker --name <compute target name> --address **<ip address or FQDN>** --username <admin username> --password <admin password>

*$ az ml computetarget attach remotedocker --name* ***DSVMLnx1*** *--address* [***13.81.245.131***](https://portal.azure.com/)*--username* ***stefan*** *--password* ***my\_secret\_pw***

# prepare the Docker image on the DSVM

$ az ml experiment prepare -c <compute target name>

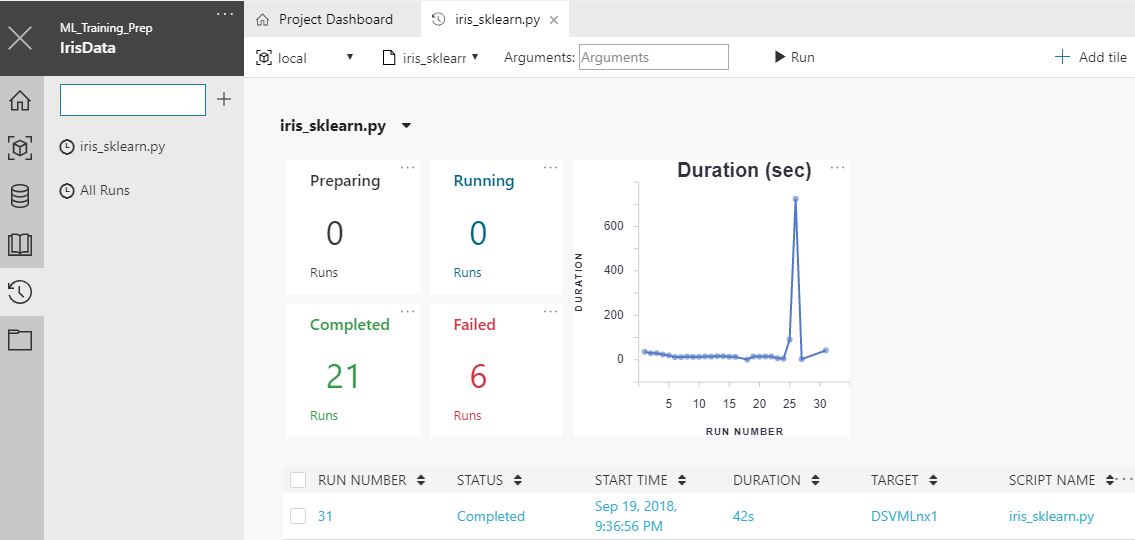
*$ az ml experiment prepare -c* ***DSVMLnx1***

This step is not mandatory: Edit the generated myvm.runconfig file (if it exists) under aml\_config and change the framework from the default value PySpark to Python:

$ az ml experiment submit -c myvm iris\_sklearn.py

*$ az ml experiment submit -c* ***DSVMLnx1*** *iris\_sklearn.py*

**The command executes as if you're in a docker-python environment, except that the execution happens on the remote Linux VM. The CLI window displays the same output information.**

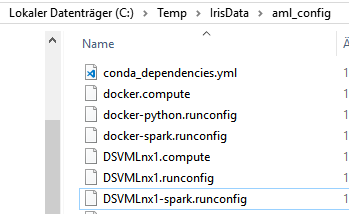


## Run scripts in a remote Docker container

Step 4

Attention: replace **myvm** by the **name of your remote DSVM**

Let's try using Spark in the container. Open File Explorer. Make a copy of the myvm.runconfig file and name it myvm-spark.runconfig. Edit the new file to change the Framework setting from Python to PySpark:



**ArgumentVector:**

**- $file**

**CondaDependenciesFile: aml\_config/conda\_dependencies.yml**

**EnvironmentVariables: null**

**Framework: PySpark**

**PrepareEnvironment: false**

**SparkDependenciesFile: aml\_config/spark\_dependencies.yml**

**Target: DSVMLnx1**

**TrackedRun: true**

**UseSampling: true**

## Run scripts in a remote Docker container

Step 5

az ml experiment submit -c myvm-spark .\iris\_spark.py

*az ml experiment submit -c* ***DSVMLnx1-spark*** *.\iris\_spark.py*

## Run scripts in HDInsights Clusters

The following step is just an option and useful for scaling in large environments

* *Execute scripts in a* ***cloud Azure HDInsight environment***

*Visit* [*https://docs.microsoft.com/en-us/azure/machine-learning/desktop-workbench/how-to-create-dsvm-hdi*](https://docs.microsoft.com/en-us/azure/machine-learning/desktop-workbench/how-to-create-dsvm-hdi) *and go to „****Create an Apache Spark for Azure HDInsight cluster in Azure portal****“*

If you have access to a Spark for Azure HDInsight cluster, generate an HDInsight run configuration command as shown here. Provide the HDInsight cluster name and your HDInsight username and password as the parameters.

Use the following command to create a compute target that points to a HDInsight cluster:

az ml computetarget attach cluster --name myhdi --address <cluster head node FQDN> --username <your-username> --password <your-password>

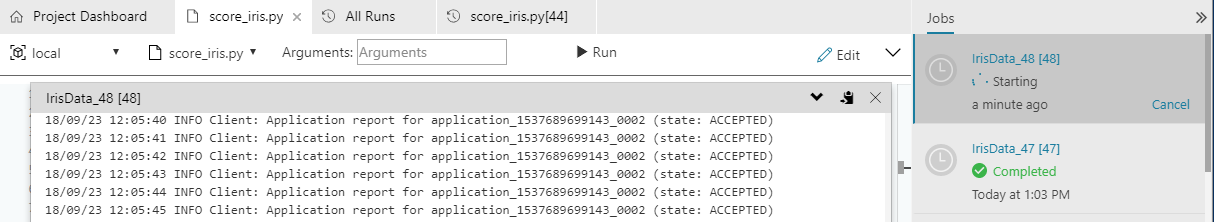
az ml computetarget attach cluster --name myhdicluster123 --address ***myhdicluster123-ssh.azurehdinsight.net*** --username ***sshuser*** --***password my\_secret\_pw***

prepare experiment

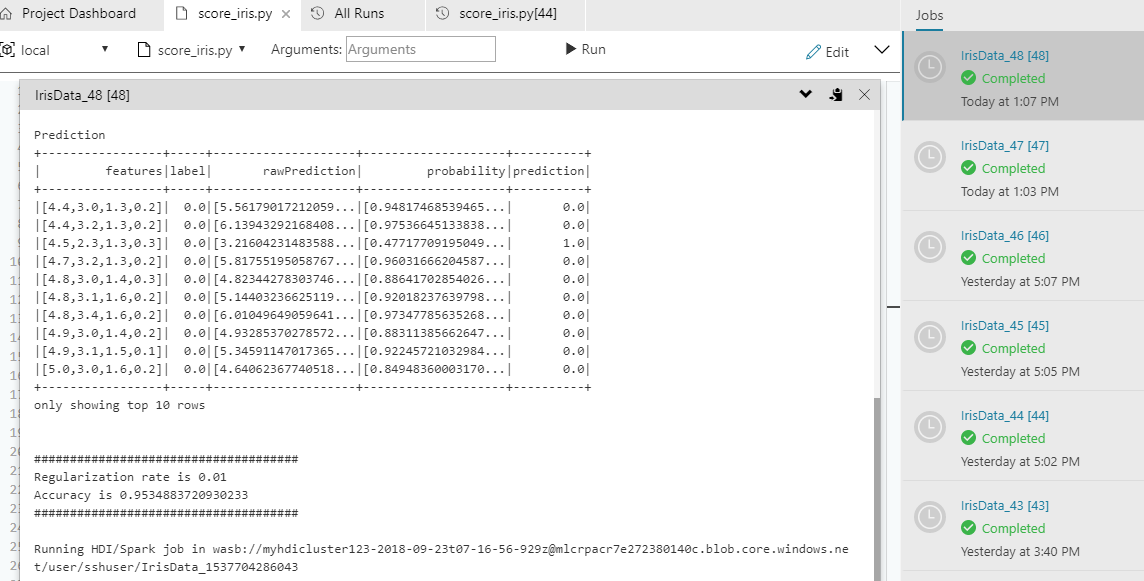
az ml experiment prepare -c ***myhdicluster123***

Run the **iris\_spark.py** script in the HDInsight cluster with this command:

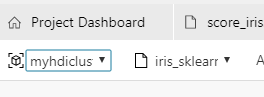
az ml experiment submit -c ***myhdicluster123*** .\iris\_spark.py

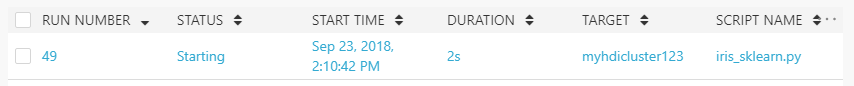
See the running activtity in the Dasboard of AML Workbench:

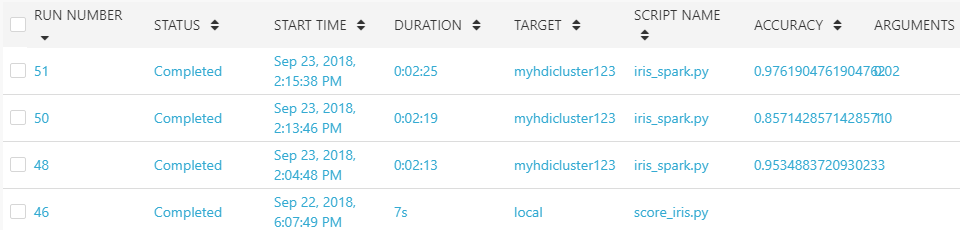
Result, when finished



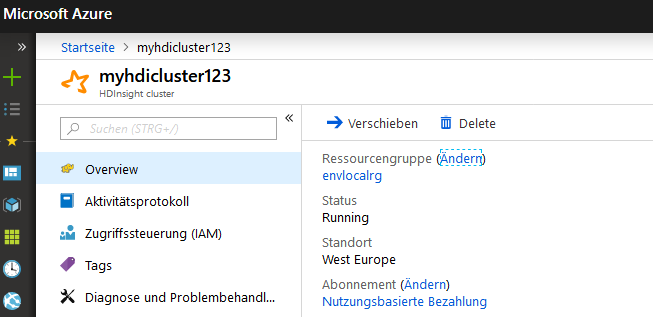
***myhdicluster123*** is also available from AML Workbench:

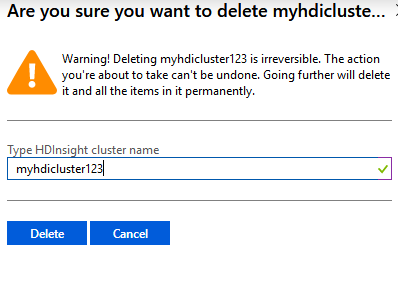
 select the **iris\_spark.py** script





## Clean up resources





* **Tutorial 3: Classify Iris: Deploy a model**

<https://docs.microsoft.com/en-us/azure/machine-learning/desktop-workbench/tutorial-classifying-iris-part-3>

This tutorial is part three of a three-part series. In this part of the tutorial, you use Machine Learning (preview) to:

* Locate the model file.
* Generate a scoring script and schema file.
* Prepare the environment.
* Create a real-time web service.
* Run the real-time web service.
* Examine the output blob data.

To complete this tutorial, you need:

An Azure subscription. If you don't have an Azure subscription, create a free account before you begin.

An experimentation account and Azure Machine Learning Workbench installed as described in this quickstart

The classification model from Tutorial part 2

A Docker engine installed and running locally

Prepare to operationalize locally [For development and testing your service]

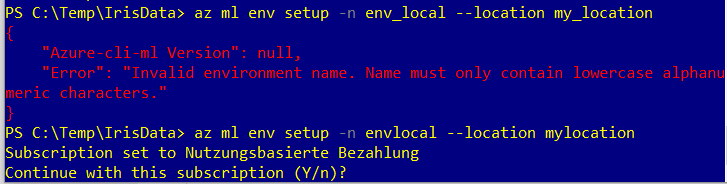
Step 2: If **Microsoft.ContainerRegistry** is not registered, you can register it by using the following command

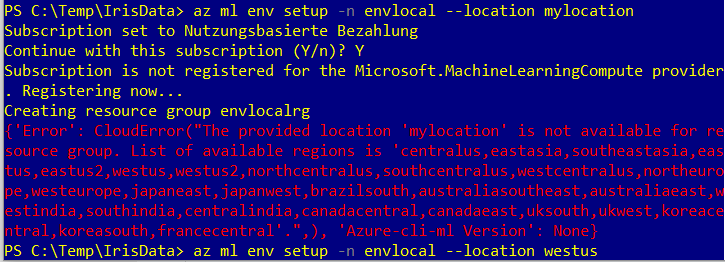
az provider register --namespace Microsoft.ContainerRegistry

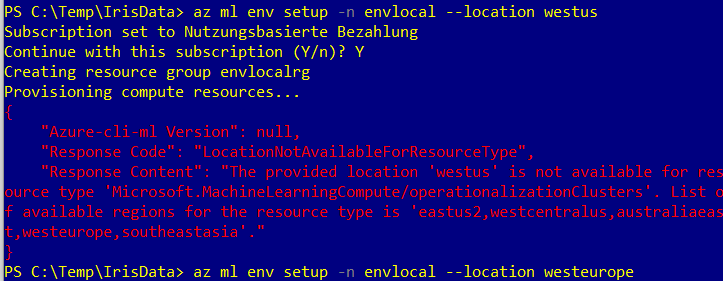
Step 3: Create the environment

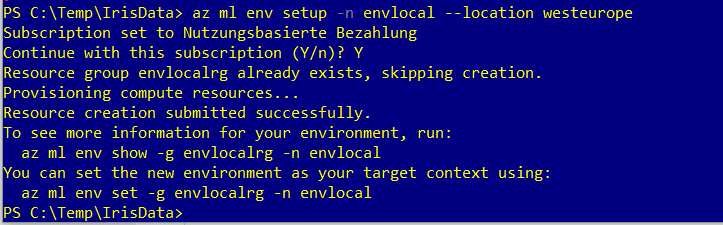
az ml env setup -n <new deployment environment name> --location <e.g. eastus2>

az ml env setup -n ***envlocal*** --location ***mylocation***







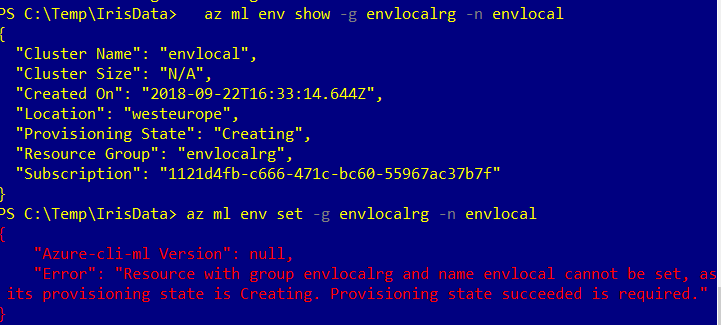


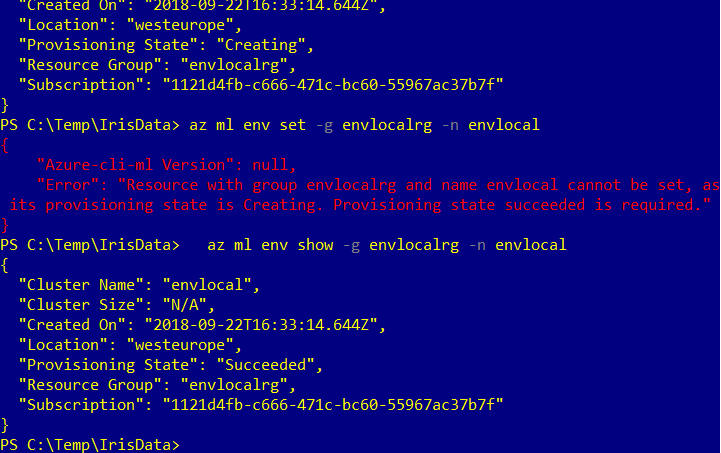
To see more information for your environment, run:

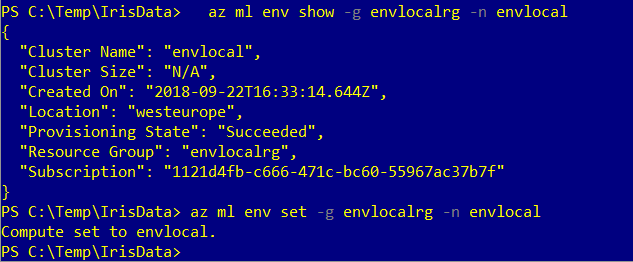
az ml env show -g ***envlocalrg*** -n ***envlocal***

You can set the new environment as your target context using:

az ml env set -g ***envlocalrg*** -n ***envlocal***







Prepare to operationalize locally [For development and testing your service]

Step 4: If you didn't create a Model Management account in previous parts of this tutorial, do so now. This is a one-time setup.

az ml account modelmanagement create --location <e.g. eastus2> -n <new model management account name> -g <existing resource group name> --sku-name S1

az ml account modelmanagement create --location ***westeurope*** -n ***mymodelacct*** -g ***sabmlexpresgrp*** --sku-name S1

Achtung! Falls mehrere Abonnements existieren, muss eine ressourcengruppe ausgewählt werden, die zum ausgewählten Abonnement gehört:



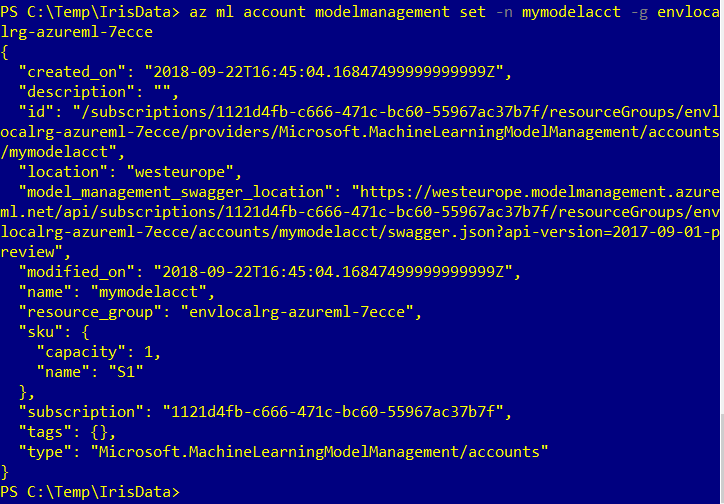
az ml account modelmanagement create --location ***westeurope*** -n ***mymodelacct*** -g ***envlocalrg-azureml-7ecce***

***p*** --sku-name S1

Step 5: Set the Model Management account.

az ml account modelmanagement set -n <youracctname> -g <yourresourcegroupname>

az ml account modelmanagement set -n ***mymodelacct*** -g ***envlocalrg-azureml-7ecce***

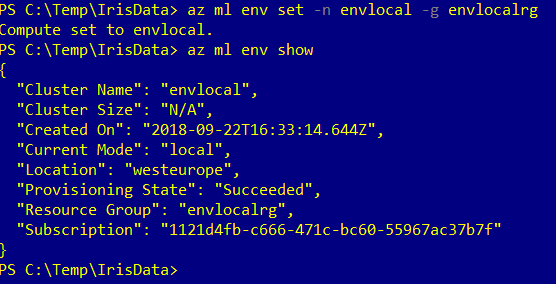


Step 6: Set the Environment

az ml env set -n <deployment environment name> -g <existing resource group name>

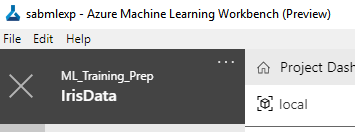
az ml env set -n ***envlocal*** -g ***envlocalrg*** (see Step 3 above for the name of the resourc egroup)

Step 7: To verify that you have properly configured your operationalized environment for local web service deployment, enter the following command:



## Create a real-time web service in one command

1. To create a real-time web service, use the following command:



az ml service create realtime -f **score\_iris.py** --model-file **model.pkl** -s **service\_schema.json** -n ***irisdata*** -r python --collect-model-data true -c aml\_config\conda\_dependencies.yml

(check wether you used this name for your project, only lowercase letters allowed)

Here: local docker container will be created

* **Tutorial 4 (optional) : Advanced Data Preparation**

<https://docs.microsoft.com/en-us/azure/machine-learning/desktop-workbench/tutorial-bikeshare-dataprep>

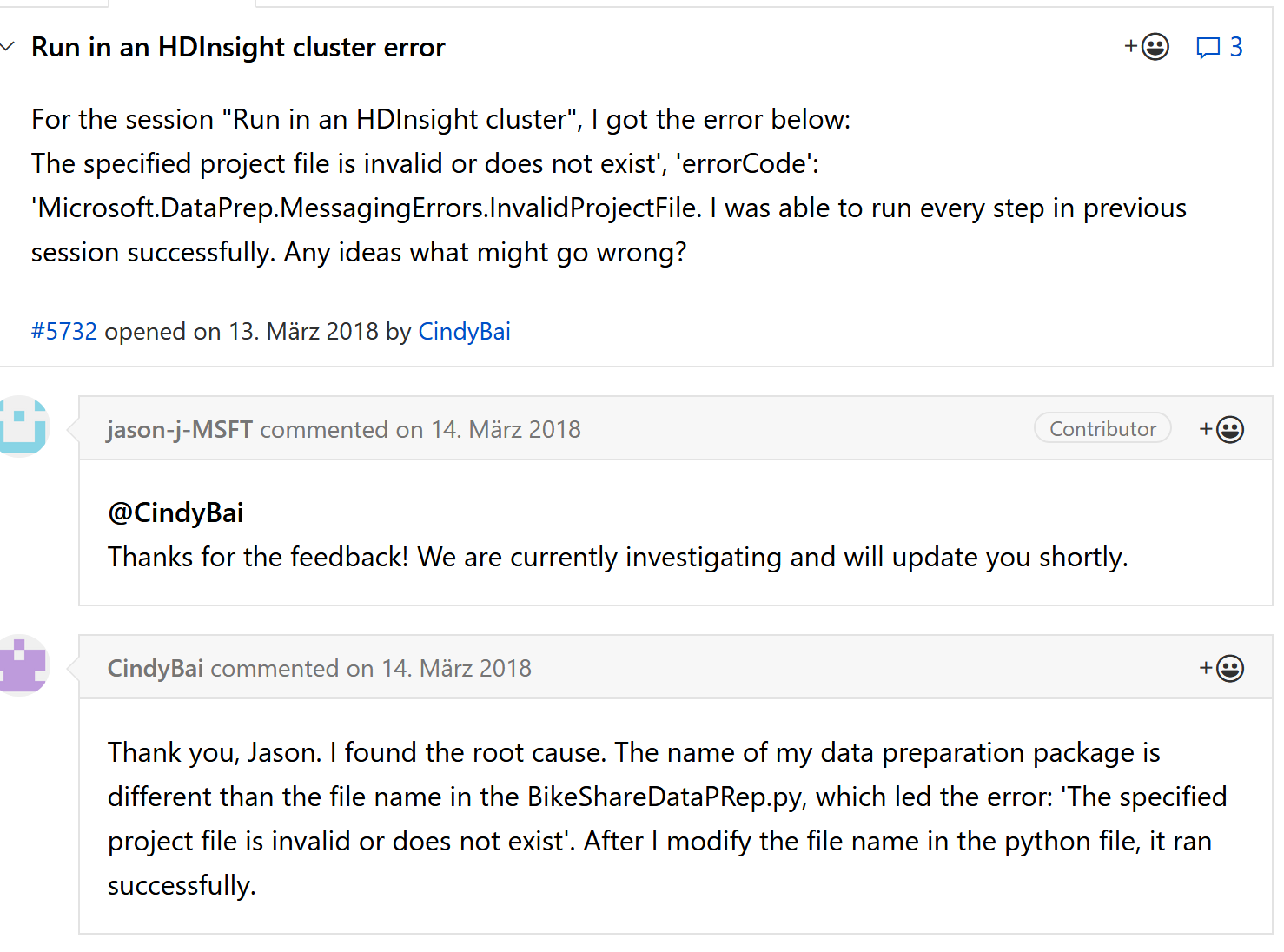
Use Azure Machine Learning Workbench for advanced data preparation (Bike share data)

Azure Machine Learning (preview) is an integrated, end-to-end data science and advanced analytics solution for professional data scientists to prepare data, develop experiments, and deploy models at cloud scale.

In this tutorial, you use Machine Learning (preview) to learn how to:

* Prepare data interactively with the Machine Learning data preparation tool.
* Import, transform, and create a test dataset.
* Generate a data preparation package.
* Run the data preparation package by using Python.
* Generate a training dataset by reusing the data preparation package for additional input files.
* Execute scripts in a local Azure CLI window.
* Execute scripts in a cloud Azure HDInsight environment.

Achtung Fehler im Tutorial oder in der Software



**Links to documentation and How-Tos**

* **CLI (Command Line Interface) for Azure Machine Learning (Workbench)**

<https://docs.microsoft.com/en-us/azure/machine-learning/desktop-workbench/cli-for-azure-machine-learning>

* **Azure Machine learning CLI**

<https://docs.microsoft.com/en-us/azure/machine-learning/service/reference-azure-machine-learning-cli>

* **Data Preparations User Guide / Documentation**

<https://docs.microsoft.com/en-us/azure/machine-learning/desktop-workbench/data-prep-user-guide>

* **Tutorial (optional): Classifying Iris using the command-line interface**

<https://docs.microsoft.com/en-us/azure/machine-learning/desktop-workbench/tutorial-iris-azure-cli>

# How to Use Run History and Model Metrics in Azure Machine Learning Workbench

<https://docs.microsoft.com/en-us/azure/machine-learning/desktop-workbench/how-to-use-run-history-model-metrics>

# Find runs with the best accuracy and lowest duration

<https://docs.microsoft.com/en-us/azure/machine-learning/desktop-workbench/how-to-find-best-accuracy-cli>

# Model management setup

<https://docs.microsoft.com/en-us/azure/machine-learning/desktop-workbench/deployment-setup-configuration>

* **Azure Machine Learning Model Management**

<https://docs.microsoft.com/en-us/azure/machine-learning/desktop-workbench/model-management-overview>

Azure Machine Learning Model Management enables you to manage and deploy machine-learning workflows and models.

Model Management provides capabilities for:

* Model versioning
* Tracking models in production
* Deploying models to production through AzureML Compute Environment with [Azure Container Service](https://azure.microsoft.com/services/container-service/) and [Kubernetes](https://docs.microsoft.com/azure/container-service/kubernetes/container-service-kubernetes-walkthrough)
* Creating Docker containers with the models and testing them locally
* Automated model retraining
* Capturing model telemetry for actionable insights.

Azure Machine Learning Model Management provides a registry of model versions. It also provides automated workflows for packaging and deploying Machine Learning containers as REST APIs. The models and their runtime dependencies are packaged in Linux-based Docker container with prediction API.

