Microsoft logo white text version

Zzz

Une image contenant bleu, brouillard, capture d’écran, bleu vert

Description générée automatiquement

Responsible AI Workshop

Leveraging Responsible AI Tooling for your non-Generative AI-powered solutions

A starter guide for data engineers, data scientists, AI developers, and other AI practitioners to help putting Responsible AI into practice

Version 1.0c – August 2022 (Updated: June 2024)

This page is intentionally left blank.

Table of contents

[Notice 4](#_Toc170393918)

[About this guide and the learning objectives 5](#_Toc170393919)

[Objectives of this guide 5](#_Toc170393920)

[Non-objectives of this guide 6](#_Toc170393921)

[Guide elements 6](#_Toc170393922)

[Guide elements 6](#_Toc170393923)

[Guide prerequisites 7](#_Toc170393924)

[Cloning the tutorials’ Jupyter notebooks 8](#_Toc170393925)

[Module 1: Better understanding your data and the behavior of ML algorithms 10](#_Toc170393926)

[Fairlearn 11](#_Toc170393927)

[InterpretML 18](#_Toc170393928)

[Error-Analysis 24](#_Toc170393929)

[Responsible AI Dashboard 29](#_Toc170393930)

[Module 2: Protecting your AI systems and your data assets 35](#_Toc170393931)

[Counterfit 36](#_Toc170393932)

[Presidio 41](#_Toc170393933)

[SmartNoise 44](#_Toc170393934)

[As a conclusion 51](#_Toc170393935)

[To go beyond 52](#_Toc170393936)

# Notice

Information in this document, including URL and other Internet Web site references, is subject to change without notice. Unless otherwise noted, the example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, e-mail address, logo, person, place or event is intended or should be inferred. Complying with all applicable copyright laws is the responsibility of the user.

A drawing of a face

Description automatically generated

Attribution 4.0 International (CC BY 4.0)

Microsoft and any contributors grant you a license to this document under the [Creative Commons Attribution 4.0 International Public License](https://creativecommons.org/licenses/by/4.0/legalcode), see the [LICENSE](https://github.com/microsoft/responsible-ai-workshop/blob/main/LICENSE) file, and grant you a license to any code in the repository under the [MIT License](https://opensource.org/licenses/MIT), see the [LICENSE-CODE](https://github.com/microsoft/responsible-ai-workshop/blob/main/LICENSE-CODE) file.

Microsoft, Windows, Microsoft Azure and/or other Microsoft products and services referenced in the document may be either trademarks or registered trademarks of Microsoft in the United States and/or other countries. The license for this document does not grant you rights to use any Microsoft names, logos, or trademarks. Microsoft's general trademark guidelines can be found at <http://go.microsoft.com/fwlink/?LinkID=254653>.

Privacy information can be found at <https://privacy.microsoft.com/en-us/>

Microsoft and any contributors reserve all other rights, whether under their respective copyrights, patents, or trademarks, whether by implication, estoppel or otherwise.

# About this guide and the learning objectives

Welcome to this starter guide Leveraging Responsible AI Tooling for your non-Generative AI-powered solutions.

The tech industry is being called upon to develop and deploy Artificial Intelligence (AI) technologies and Machine Learning (ML)-powered systems (products, applications, or services) and/or features more responsibly – we will further refer to these as (non-Generative) AI systems -. Yet many organizations implementing such AI systems report being unprepared to address AI risks and failures.

To meet these challenges, Microsoft is striving to adopt a human-centered approach to AI, designing and building technologies that benefit people and society while also mitigating potential harms. This includes understanding human needs and using these insights to drive development decisions from beginning to end.

This guide consists of a series of modules and tutorials for data engineers, data scientists, AI developers and other AI practitioners, as well as potentially anyone interested considering the wide range of socio-technical aspects involved in the subject.

## Objectives of this guide

As its name indicates, this guide is part of the Responsible AI Workshop. As such, it is more particularly intended to give you an overview of the most prominent RAI tools we open sourced as (standalone) libraries and dashboards, or integrated into [Azure Machine Learning (Azure ML)](https://azure.microsoft.com/en-us/services/machine-learning/), its [responsible machine learning capabilities](https://azure.microsoft.com/en-us/services/machine-learning/responsibleml/#overview), its [MLOps capabilities](https://azure.microsoft.com/en-us/services/machine-learning/), and where to leverage them in your own product development lifecycle.

As far as the latter is concerned, this guide will more particularly explore two categories of these practical tools that help putting RAI principles to work, i.e., into practice:

1. Tooling to (better) understand the behavior of non-Generative AI systems,
2. Tooling to help protecting non-Generative AI systems’ models and the data assets,

As well as widgets which group these tools under a single roof and allow access to them through a unified set of dashboards.

By the end of the guide and the outlined tutorials, you will be able, as part of the workshop and in terms of learning objective to more specifically:

* Assess and mitigate your AI system's unfairness issues using Fairlearn.
* Understand your model's global behavior or understand the reasons behind individual predictions using the unified InterpretML API.
* Use the Responsible AI Toolboxinterpretability, error analysis and fairness dashboards to develop and monitor your own AI systems, and related solutions more responsibly.
* Assess the security of AI systems using **Counterfit**, a command-line tool and generic automation layer for such activities and AI/ML thread modeling.
* Identify and anonymize Personally Identifiable Information (PII) in your data using Presidio data protection and anonymization software development kit (SDK).
* Protect personal data for your ML-powered solutions and applications using Differential Privacy (DP) through the SmartNoise system.

## Non-objectives of this guide

This guide isn’t aimed at introducing the building blocks of Responsible AI. For an introduction to RAI, and notably through Microsoft’s ongoing journey in the field, please refer to the guide [Establishing your own Responsible AI journey for your (non-Generative vs. Generative) AI-powered solutions](https://github.com/microsoft/responsible-ai-workshop/blob/main/responsible-ai-journey/docs/establishing-your-own-responsible-ai-journey.docx).

Nor does this guide cover Generative AI as such. For guidance and tutorials for your Generative AI-powered solutions, please refer to the guide [Building and using Generative AI responsibly with Azure and beyond](https://github.com/microsoft/responsible-ai-workshop/blob/main/gen-ai-tooling-tutorials/docs/buildling-and-using-gen-ai-responsibly.docx).

These two guides are also part of this Responsible AI Workshop, which is available on GitHub at <https://github.com/microsoft/responsible-ai-workshop>.

**Note** For a complete overview of Microsoft’s resources designed to help you responsibly implement (non-Generative vs. Generative) AI systems, please refer to the [Microsoft Responsible AI resources page](https://aka.ms/rairesources).

## Guide elements

The document is organized as follows.

Both Module 1 and Module 2 each explore a series of RAI tools that can be used to understand and protect AI systems and the data used for training or inference, respectively. For each Responsible AI tool, we will provide a description of the concept behind that tool, a description of how it works and a hands-on tutorial to walk you through each outlined tool.

The goal is indeed to allow you to jump straight into using each of these techniques by providing you with minimal example code to get you started as well as pointers towards more comprehensive resources if you wish to expand your knowledge of a particular technique.

## Guide elements

For each Responsible AI tool explored in this guide in Modules 1 and 2, we provide the following elements:

* Description of the tool and the underlying concept.
* Hands-on tutorialcontaining the most important steps and their outputs. These are meant to show you the core elements and do not contain for example code for data loading and processing which we preferred to omit here for conciseness, please refer to the Jupyter notebooks for a comprehensive run through each tutorial.
* Tutorials’ interactive Jupyter notebookswhich you can access by downloading or cloning the following GitHub repository: <https://github.com/microsoft/responsible-ai-workshop>.

## Guide prerequisites

To successfully leverage the code in the Jupyter notebooks accompanying Module 2 and Module 3 of this guide, you have multiple options with the four main ones being:

1. Running the tutorials’ Jupyter notebooks in JupyterLab. [JupyterLab](https://jupyter.org/) is a web-based interactive development environment for creating such [Jupyter notebooks](https://jupyterlab.readthedocs.io/en/stable/user/notebook.html).

JupyterLab provides flexible building blocks and an [interface](https://jupyterlab.readthedocs.io/en/stable/user/interface.html) for interactive, exploratory computing with, at its core, the ability to configure and arrange the user interface (UI), manipulate data with [Python](https://www.python.org/) and display inline graphs, e.g., diagrams and/or dashboards from the results, etc. to support a wide range of data science and ML workflows.

Official instructions on how to install JupyterLab are provided [here](https://jupyter.org/install.html). For Windows 10 users, JupyterLab can be installed for example with the [Windows Subsystem for Linux (WSL) 2](https://docs.microsoft.com/en-us/windows/wsl/install-win10) or through the [Anaconda open-source distribution](https://www.anaconda.com/products/individual#Downloads).

1. Running the tutorials’ Jupyter notebooks in Visual Studio Code. [Visual Studio Code (VS Code)](https://code.visualstudio.com/) is a free code editor and development platform that you can use locally on your Linux, MacOS or Windows environments, or connected to remote compute.

Combined with the [Jupyter extension](https://marketplace.visualstudio.com/items?itemName=ms-toolsai.jupyter) (and the [Python extension](https://marketplace.visualstudio.com/items?itemName=ms-python.python)), it offers a full environment for Jupyter development that can be enhanced with additional language extensions. If you want a best-in-class, free Jupyter experience with the ability to leverage your compute of choice, this is a great option. Using VS Code, you can develop and run notebooks on remotes and containers.

Instructions on how to get started with notebooks in VS code (optionally using container) are provided [here](https://code.visualstudio.com/docs/datascience/jupyter-notebooks).

-or as an alternative -

1. Running the tutorials’ Jupyter notebooks in GitHub CodeSpaces. [GitHub CodeSpaces](https://docs.github.com/en/codespaces) currently in beta provide you with cloud-hosted environments where you can edit your notebooks using VS Code or your web browser and store them on GitHub.

GitHub CodeSpaces offers the same great Jupyter experience as VS Code above, but without needing to install anything on your local environment. GitHub CodeSpaces also allows you to use your cloud compute of choice. If you don’t want to set up a local environment and prefer a cloud-backed solution, then creating a codespace is a great option.

Instructions on how to get started with notebooks in your own codespace are provided [here](https://code.visualstudio.com/docs/datascience/jupyter-notebooks).

1. Running the tutorials’ Jupyter notebooks in Azure Machine Learning. [Azure Machine Learning (Azure ML)](https://azure.microsoft.com/en-us/services/machine-learning/) provides an end-to-end ML platform to enable you to build and deploy models faster on Azure.

Azure ML allows you to run notebooks on a virtual machine (VM) or a shared cluster computing environment. If you need a cloud-based solution for your ML workload with experiment tracking, dataset management, and more, we recommend Azure ML.

To run the Jupyter notebooks in your Machine Learning workspace, please note that if you choose this option, there are two prerequisites:

* 1. Having an Azure subscription. If you don't have an Azure subscription yet, you can create a free account [here](https://azure.microsoft.com/free/).
  2. Having a Machine Learning workspace. You will find instructions on how to setup one using the Azure portal [here](https://docs.microsoft.com/en-us/azure/machine-learning/how-to-manage-workspace?tabs=python).

Instructions on how to get started with Azure ML are provided [here](https://aka.ms/aznb-aml).

Other options to run Jupyter notebooks using products and services from Microsoft and GitHub exist. They can be found [here](https://notebooks.azure.com/).

Please note that whatever the option you choose, and provided you are using Python 3.7 or above, the code within the Jupyter notebooks is self-sufficient. You just need to uncomment the cells for libraries installation using the pip package manager when necessary and you should be good to go! Let’s shortly see how to clone them.

## Cloning the tutorials’ Jupyter notebooks

The Responsible AI Workshop repo on GitHubcontains a series of tutorials’ Jupyter notebooks.

Depending on your choice about the above prerequisites, you have various options available to clone this repo. For the sake of this starter guide, we assume here that you have a Windows 10 local environment with possibly WSL 2 installed with an Ubuntu distribution. (This also *de facto* covers both Linux and MacOS environment). Let’s consider them in order.

Let’s start with Git for Windows.

### Cloning the repo using Git for Windows

To clone the repo on your Windows 10 local machine, perform the following steps:

1. Download the [Git for Windows](https://github.com/git-for-windows/git/releases/download/v2.25.0.windows.1/Git-2.25.0-64-bit.exe) and run it.
2. In the pop-up window, click Install. Follow the instructions.
3. Open a PowerShell console, and run the following command:

PS C:\> md rai-workshop

PS C:\> cd C:\rai-workshop\

PS C:\> git clone https://github.com/microsoft/responsible-ai-workshop.git

### Cloning the repo using Git on Ubuntu

Likewise, to clone the repo on your Ubuntu environment (WSL2), from a Bash terminal console, run the following commands:

$ cd $home

$ mkdir rai-workshop

$ cd rai-workshop

$ git clone https://github.com/microsoft/responsible-ai-workshop.git

You can instead install GitHub for Desktop for Windows.

### Cloning the repo using GitHub Desktop

To clone the repo on your Windows 10 local machine, perform the following steps:

1. Download the [GitHub Desktop installer](https://central.github.com/deployments/desktop/desktop/latest/win32) and run it.
2. In the pop-up window, click Install. Follow the instructions.
3. Open a browser session and navigate on GitHub to the main page of the Responsible AI Workshop repo at <https://github.com/microsoft/responsible-ai-workshop>.
4. Click Clone or download.
5. Click Open in Desktop to clone the repository. GitHub Desktop opens up and a dialog shows up.
6. Click **Choose...** and, using Windows Explorer, navigate to a local path where you want to clone that repo. Throughout this starter guide, we will use the rai-workshop folder as an illustration.
7. Click **Clone**.

See [Cloning a repository from GitHub to GitHub Desktop](https://docs.github.com/en/desktop/contributing-and-collaborating-using-github-desktop/adding-and-cloning-repositories/cloning-a-repository-from-github-to-github-desktop).

So, at this stage, you’re all set! It’s high time to move to the first module.

# Module 1: Better understanding your data and the behavior of ML algorithms

The goal of this module is to explore some of the Responsible AI tools we provide for a better understanding of the behavior of ML models, and more particularly Fairlearn, InterpretML and Error Analysis.

But before we investigate each individual tool separately, it is important to put them in context with respect to the RAIL lifecycle depicted in the previous section and to comprehend how these tools can work together to achieve a better understanding of ML algorithms, as opposed to each being used independently.

All three tools we explore in this module fall under the Design, Build, and Document phase of the RAIL lifecycle (see the What about a Responsible AI Lifecycle? section above) and act directly on ML models and data they are fed to achieve two main goals:

1. Model fairness by ensuring ML algorithms avoid treating similarly situated groups of people in different ways because of sensitive attributes such as race, gender, age, or disability status. Fairness issues are generally due to bias already existing in training data, which is then enhanced by the models, but it might be the case that models induce their own bias into the system.

For instance, a system used in the hiring process of a company which is fed hiring decisions made by a manager as labels tends to replicate or enhance the bias existing in the manager’s decisions when selecting applicants rather than taking full consideration of the capability of a job applicant (most of time this capability is unobserved for people who are rejected). This can lead to a candidate being unrightfully discarded, which is obviously a big fairness problem and shows the extent to which a tool for assessing and mitigating this unfairness is needed.

1. Model transparency by leveraging interpretability techniques to understand models’ global trends, predictions for selected cohorts of the data as well as explaining individual predictions. Model transparency is indeed a subject of utmost importance because executives and stakeholders need to be able to grasp the value and accuracy of data scientists' results, hence data scientists must be able to explain their models to them. Moreover, some of these predictions can be life changing like credit risk modeling or hiring applicants for a job, and decision makers owe it for the subjects to explain their decision process, which is something a blackbox model can’t do.

This is the reason why there are multiple instances of sensitive settings where people call for black-box models to be barred from making predictions because of their lack of interpretability. One of the most recent such developments involves an ML model used for [diagnosing COVID-19 from chest X-rays](https://www.nature.com/articles/s42256-021-00338-7). Researchers at the University of Washington in Seattle found that the model used relies on confounding factors rather than medical pathology to diagnose COVID-19, creating a delicate situation that could have been avoided had the model been explainable in the first place.

But beyond being used independently, the three tools explored in this section are part of an overall scheme illustrated in Figure 10 below, which is comprised of three steps : i) identifying issues for example in terms of model performance, fairness metrics, etc., ii) diagnosing the problem by pinpointing what exactly is wrong and finally iii) mitigating the problem with appropriate mitigation techniques.

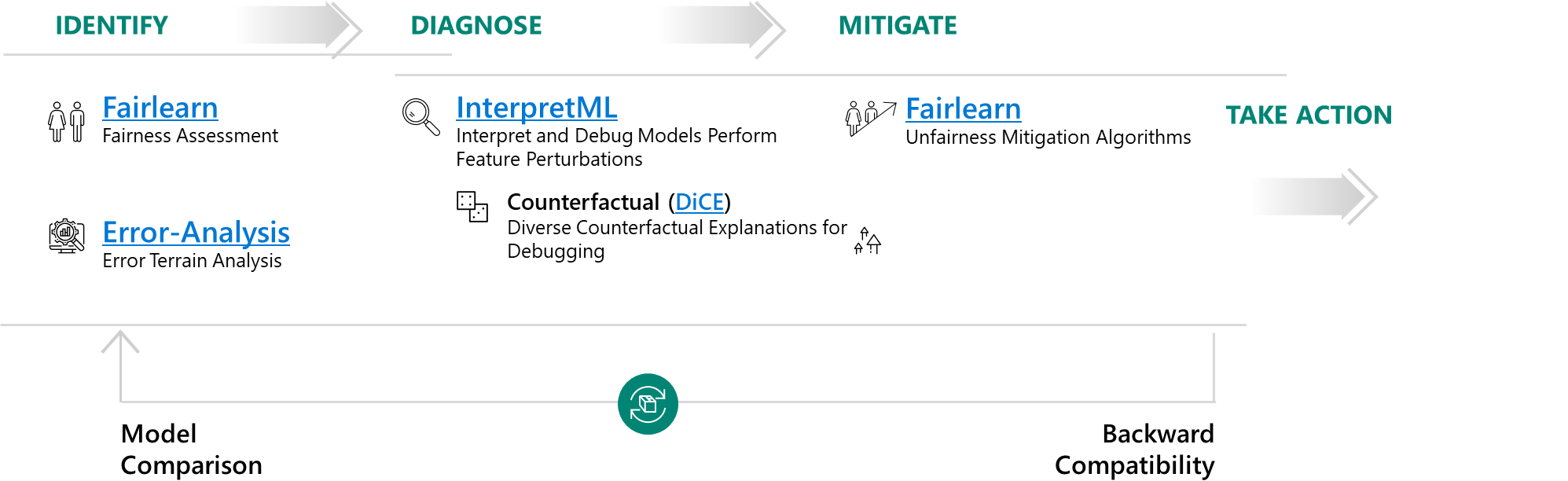


Figure . The 4 phased scheme for better understanding of ML models and the associated tools for each phase

Please note that the scheme presented above is by no means perfect and we are engaged in an iterative process of refining the tools. For example, dashboards previously made accessible through each tool independently (e.g., Fairness dashboards, interpretability dashboards and Error Analysis dashboards) are now being grouped together under one roof: the Responsible-AI-Widgets library which will be introduced at the end of this module as part of the Error Analysis section. This novelty is driven by the fact that these three tools are generally used together as we explained in the previous paragraph, so having all the dashboards in the same place comes in very handy.

Now that we have a good overview of the tooling for better understanding of ML models behavior and how these tools can be combined, we are ready to start investigating each of the tools we chose to focus on individually.

## Fairlearn

As suggested above, the first challenge encountered in understanding AI systems today is the inability to assess and mitigate unfairness in the ML models. Because, yes, models can be unfair, and this unfairness can come either from the training data which includes some bias, or models inducing their own bias into the system.

To address this challenge, we’ve recently open-sourced a toolkit called [Fairlearn](https://fairlearn.org/). Fairlearn is an [open-source Python package](https://github.com/fairlearn/fairlearn) that empowers AI practitioners to assess their algorithms fairness and mitigate any observed unfairness issues. It provides state-of-the-art fairness metrics to evaluate your model’s fairness along with algorithms for mitigating those fairness issues.

Using Fairlearn, developers and data scientists can leverage specialized algorithms to ensure fairer outcomes for everyone. While Fairlearn can be accessed through the built-in visualizations in Azure ML, the Fairlearn toolkit contains fairness metrics, mitigation algorithms as well as a Jupyter widget for model assessment.

Fairlearn focuses on models’ negative impacts on groups of people, such as those defined in terms of race, gender, age, or disability status. For example, a voice recognition system might fail to work as well for women as it does for men, or a system for screening loan or job applications might be much better at picking good candidates among white men than among other groups. The goal of Fairlearn is to detect and help mitigate such biases.

### Fairlearn components

There are two components to Fairlearn. The first is metrics and an assessment dashboard for assessing which groups are negatively impacted. The second is a set of algorithms for mitigating fairness issues in a variety of AI tasks and along a variety of fairness definitions!

1. Assessing unfairness. For assessing unfairness, the Fairlearn package provides an interactive dashboard for evaluating the overall performance of an existing model, and any disparities in the model evaluation metrics such as a disparity in model performance (e.g., accuracy rate, error rate, precision, recall, etc.) or a disparity in selection rate (e.g., loan approval rate) across different groups (e.g., different genders). This enables users to easily detect if there is unfairness against any groups in the existing model, regardless of whether the sensitive attributes have been included during the model training or not.
2. Mitigating unfairness. It’s perhaps worth calling out that simply removing known sensitive attributes from the model training dataset usually cannot effectively eliminate unfairness in the resulting model, as there are often other features correlated with the removed attributes in the training dataset that would result in unfairness in the model anyway.

So simply removing sensitive attributes like gender and race does not work as these can usually be induced from other correlated features. Thus, for mitigating fairness-related harms, the Fairlearn package provides more clever implementations of mitigation methods such as Threshold Optimization and the “Reductions” approach as depicted hereafter:

* The Threshold Optimization is a post-processing technique that takes as input an existing classifier and the sensitive feature and derives a transformation of the classifier's prediction (e.g., adjusting the threshold for predicted probabilities) to enforce the specified parity constraints. In short, modifying the decision boundary achieving better results in terms of fairness metrics, while potentially altering performance a bit as there exists a tradeoff between Fairness and performance metrics.
* The “Reductions” approach takes a standard ML estimator (e.g., a LightGBM model) as a black box and generates a set of retrained models using a sequence of reweighted training datasets.

For example, applicants of a certain gender might be upweighted or down weighted to retrain models to reduce disparities across different gender groups. Users can then pick a model that provides the best trade-off between accuracy (or other performance metrics) and disparity, which is quantified by fairness metrics defined in the previous point.

One key advantage of the “Reductions” approach is that it only requires access to the sensitive features during model training, not when the model is deployed for inferencing. This is because the sensitive features are only used for training data reweighting, and not need to be part of the input features of the actual model itself. This is extremely useful for many applications that do not have access to sensitive attributes for model prediction in production.

For additional information on Fairlearn, you can consider the following resources:

* Project landing page <https://FairLearn.org>.
* GitHub repo <https://github.com/fairlearn/fairlearn>.
* AI Show video [Building fairer AI Systems with Fairlearn](https://channel9.msdn.com/Shows/AI-Show/Building-fairer-AI-Systems-with-Fairlearn).
* White paper [Fairlearn: A toolkit for assessing and improving fairness in AI\*](Fairlearn:%20A%20toolkit%20for%20assessing%20and%20improving%20fairness%20in%20AI*).
* And more specifically for the Azure ML integration, these two articles:
  + Concept: [Machine learning fairness (preview)](https://docs.microsoft.com/en-us/azure/machine-learning/concept-fairness-ml).
  + How-to: [Use Azure Machine Learning with the Fairlearn open-source package to assess the fairness of ML models (preview)](https://docs.microsoft.com/en-us/azure/machine-learning/how-to-machine-learning-fairness-aml)”.

With that, let’s now illustrate the use of this package with a first hands-on tutorial.

### Hands-on tutorial: Fair binary classification of credit card default use-case

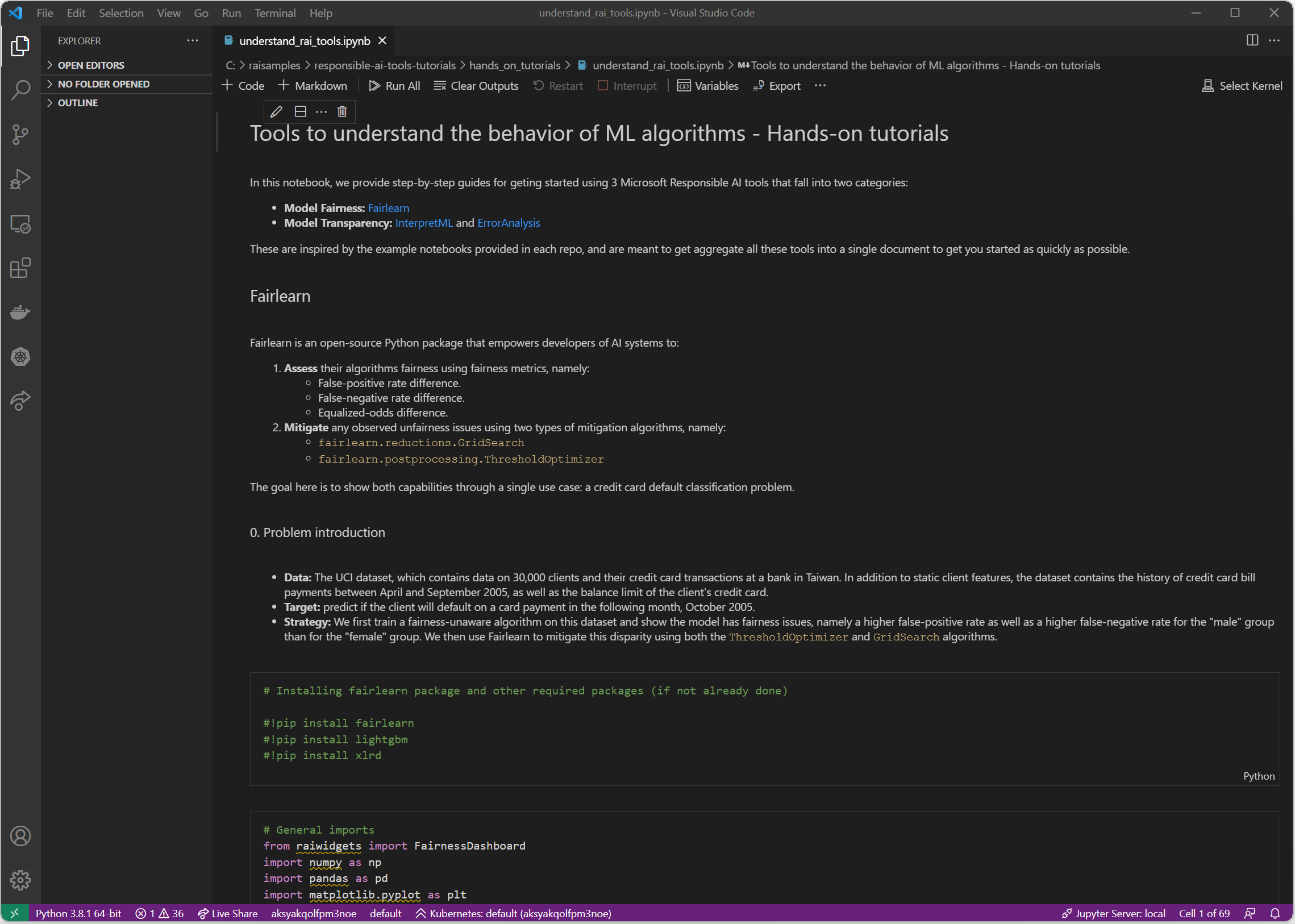
The Jupyter notebook for this hands-on tutorial is understand\_rai\_tools.ipynb located in the tooling-tutorials\hands\_on\_tutorials sub-directory underneath the folder where you have cloned the workshop repo, for example the folder rai-workshop in our illustration, please refer to the section Cloning the tutorials’ Jupyter notebooks above.

To follow this hands-on tutorial, and the same goes for the other hands-on tutorials in this starter guide, you need to open up this file in the Jupyter environment of your choice, please refer to the section Guide prerequisites above.

The goal here is to train a fairness-unaware binary classification algorithm on a credit card default dataset and show the model has a higher false-positive rate as well as a higher false-negative rate for the "male" group than for the "female" group. We then use Fairlearn to mitigate this disparity using both the Threshold Optimization approach (ThresholdOptimizer) and the Reductions approach (GridSearch).

The notebook emulates the problem presented in a [white paper](https://www.microsoft.com/en-us/research/uploads/prod/2020/09/Fairlearn-EY_WhitePaper-2020-09-22.pdf) that we developed in collaboration with EY: Assessing and mitigating unfairness in credit models with the Fairlearn toolkit. Due to data privacy, we obviously do not use the dataset from the white paper itself, but instead, we use the UCI Credit-card default dataset, a toy dataset reflecting credit-card defaults in Taiwan, as a substitute dataset to replicate the desired workflow.

Let’s open up the notebook with the environment of your choice, for example VS Code as an illustration.



#### Installing the required libraries and data loading

The first step is of course to install the required packages, Fairlearn obviously but also the lightgbm package which is the model we use in this tutorial.

!pip install fairlearn

!pip install lightgbm

Then we can load the data and have a first look at it

# Load the data

data\_url = "http://archive.ics.uci.edu/ml/machine-learning-databases/00350/default%20of%20credit%20card%20clients.xls"

dataset = pd.read\_excel(io=data\_url, header=1).drop(columns=['ID']).rename(columns={'PAY\_0':'PAY\_1'})

dataset.head()

A picture containing calendar

Description automatically generated

The data consists of static client features, the dataset contains the history of credit card bill payments between April and September 2005, as well as the balance limit of the client's credit card. The target is to predict if a client will default on a card payment in the following month, October 2005.

We then perform some preprocessing where we introduce a bias synthetic feature to create some artificial unfairness for our purposes, this is well detailed in the notebook, but we chose to omit that here.

It is also worth noting that we also completely get rid of the ‘sex’ feature, but this doesn’t help us in any way solve our unfairness issues as you’ll see below.

#### Assessing unfairness

We first fit a fairness unaware LightGBM model to our data:

lgb\_params = {

'objective' : 'binary',

'metric' : 'auc',

'learning\_rate': 0.03,

'num\_leaves' : 10,

'max\_depth' : 3

}

model = lgb.LGBMClassifier(\*\*lgb\_params)

model.fit(df\_train, Y\_train)

# Scores on test set

test\_scores = model.predict\_proba(df\_test)[:, 1]

# Predictions (0 or 1) on test set

test\_preds = (test\_scores >= np.mean(Y\_train)) \* 1

Now we can use Fairlearn dashboard from ResponsibleAI widgets to show some fairness metrics:

mf = MetricFrame({

'FPR': false\_positive\_rate,

'FNR': false\_negative\_rate},

Y\_test, test\_preds, sensitive\_features=A\_str\_test)

mf.by\_group

This prints the following metrics:

Chart, bar chart

Description automatically generated

Figure . Fairness dashboard showing both False positive rate (FPR) and False negative rate (FNR) for the “male” group (1) and the “female” group (2)

We clearly see that the “male” group suffers from much higher FNR but also FPR than the female group, so even though the overall model performance is quite good at 85% balanced accuracy, males still suffer from much lower accuracy than males, and there is clearly an unfairness problem detected by Fairlearn.

##### Threshold Optimization approach: Mitigating equalized odds difference with postprocessing

We attempt to mitigate the disparities in the LightGBM predictions using the Fairlearn postprocessing algorithm ThresholdOptimizer. This algorithm finds a suitable threshold for the scores (class probabilities) produced by the Lightgbm model by optimizing the accuracy rate under the constraint that the equalized odds difference (on training data) is zero.

postprocess\_est = ThresholdOptimizer(

estimator=model,

constraints="equalized\_odds",

prefit=True)

Since our goal is to optimize balanced accuracy, we resample the training data to have the same number of positive and negative examples.

# Balanced data set is obtained by sampling the same number of points from the majority class (Y=0)

# as there are points in the minority class (Y=1)

balanced\_idx1 = df\_train[Y\_train==1].index

pp\_train\_idx = balanced\_idx1.union(Y\_train[Y\_train==0].sample(n=balanced\_idx1.size, random\_state=1234).index)

df\_train\_balanced = df\_train.loc[pp\_train\_idx, :]

Y\_train\_balanced = Y\_train.loc[pp\_train\_idx]

A\_train\_balanced = A\_train.loc[pp\_train\_idx]

Now we can fit our ThresholdOptimizer and get the predictions. We only show the results at the end of next section.

postprocess\_est.fit(df\_train\_balanced, Y\_train\_balanced, sensitive\_features=A\_train\_balanced)

postprocess\_preds = postprocess\_est.predict(df\_test, sensitive\_features=A\_test)

##### “Reductions” approach: Mitigating equalized odds difference with GridSearch

We now attempt to mitigate disparities using the GridSearch algorithm.

# Train GridSearch

sweep = GridSearch(model,

constraints=EqualizedOdds(),

grid\_size=50,

grid\_limit=3)

sweep.fit(df\_train\_balanced, Y\_train\_balanced, sensitive\_features=A\_train\_balanced)

sweep\_scores = [predictor.predict\_proba(df\_test)[:, 1] for predictor in sweep.predictors\_]

sweep\_preds = [predictor.predict(df\_test) for predictor in sweep.predictors\_]

The Figure 11 below shows the results of the mitigation techniques we used.

Chart, scatter chart

Description automatically generated

Figure . Accuracy vs fairness for Unmitigated and mitigated models

The overall performance measure we consider is the area under ROC curve (AUC), which is suited to classification problems with a large imbalance between positive and negative examples. For binary classifiers, this is the same as balanced accuracy.

As the fairness metric we use equalized odds difference, which quantifies the disparity in accuracy experienced by different demographics. Our goal is to assure that neither of the two groups ("male" vs. "female") has substantially larger false-positive rates or false-negative rates than the other group. The equalized odds difference is equal to the larger of the following two numbers:

1. The difference between false-positive rates of the two groups
2. The difference between false-negative rates of the two groups.

The closer to zero the Equalized odds difference is, the better.

We see that ThresholdOptimizer greatly reduced the disparity in performance across multiple fairness metrics. However, the overall accuracy for the ThresholdOptimizer model were much worse than the fairness-unaware model. With the GridSearch algorithm, we have a better trade-off by mitigating unfairness with very low Equalized odds difference while maintaining the same order of performance as the unmitigated initial model.

In conclusion, the data scientist should choose and deploy the model that balances the performance-fairness trade-off in a way that meets the needs of the business.

## InterpretML

Another important aspect of understanding a ML model is the ability to interpret, or explain, its results. Interpretability is essential for:

* Helping detect fairness issues: *Does my model discriminate?*
* ML model debugging: *Why did my model make this mistake?*
* Feature engineering: *How can I improve my model?*
* Cooperation in Human-AI experience: *How can I understand and trust the model's decisions?*
* Regulatory compliance: *Does my model satisfy legal requirements?*
* High-risks AI systems in regulated industries and elsewhere.

So, to make it short, interpretability is needed to ensure there is optimal transparency ML within models to assess, and reason through the predictions it generates or the recommendations it creates. This is where [InterpretML](https://interpret.ml/docs/intro.html), an [open-source Python package](https://github.com/interpretml/interpret) (the *interpret* package), comes into play.

As its name suggests, it helps AI practitioners interpret models and predictions made by these models, and thus understand which features contribute to their ML model’s predictions. As such, it helps them understand their model's global behavior and/or the reasons behind individual predictions on local basis.

For that purpose, InterpretML incorporates state-of-the-art ML interpretability techniques under one roof, with a unified API and a built-in visualization platform.

### Supported types of interpretabilities

InterpretML exposes two types of interpretabilities:

1. Blackbox explainability, which consists of techniques for explaining existing ML models that are well known among data scientists and AI practitioners more broadly, be them global explainability techniques like Partial Dependency Plots or local techniques like LIME or SHAP.
2. Glassbox interpretability, which are ML models designed for interpretability (for example linear models, rule lists, generalized additive models).

As far as the former is concerned, we won’t get into explaining each of these techniques as this is thoroughly done elsewhere, but we will rather focus on the innovation of the InterpretML package, which is aggregating the results of all these techniques under a unified dashboard as we will show below.

Regarding the latter, InterpretML includes the first implementation of the Explainable Boosting Machine (EBM), a powerful, interpretable, glassbox model, designed to have accuracy comparable to state-of-the-art machine learning methods like Random Forest and Boosted Trees, while being highly intelligible and explainable.

This makes EBMs as accurate as state-of-the-art techniques like random forests and gradient boosted trees while still producing exact explanations and being editable by domain experts.

EBM is a generalized additive model (GAM) of the form:

A picture containing diagram

Description automatically generated

Here g is the link function that adapts the GAM to different settings such as regression or classification. Without going any deep into mathematical explanations, the goal behind showing you this formula is for you to understand that the GAM learns a separate function f*j* for each feature *j*, which means the contribution of each feature to the final prediction can be visualized and understood by plotting f*j* making GAMs highly intelligible.

The particularity of EBMs compared to traditional GAMs is that EBM learns each feature function fj using modern ML techniques such as bagging and gradient boosting and round-robin cycles through features to mitigate the effects of co-linearity.

These two types will be further investigated below through a dedicated hands-on for each, see sections below:

1. Hands-on tutorial: Blackbox explainability.
2. Hands-on tutorial: training a Glassbox model with Explainable Boosting Machine.

Figure 12 below provides a good overview of the interpretML package API for Blackbox models explainability and Glassbox models interpretability.

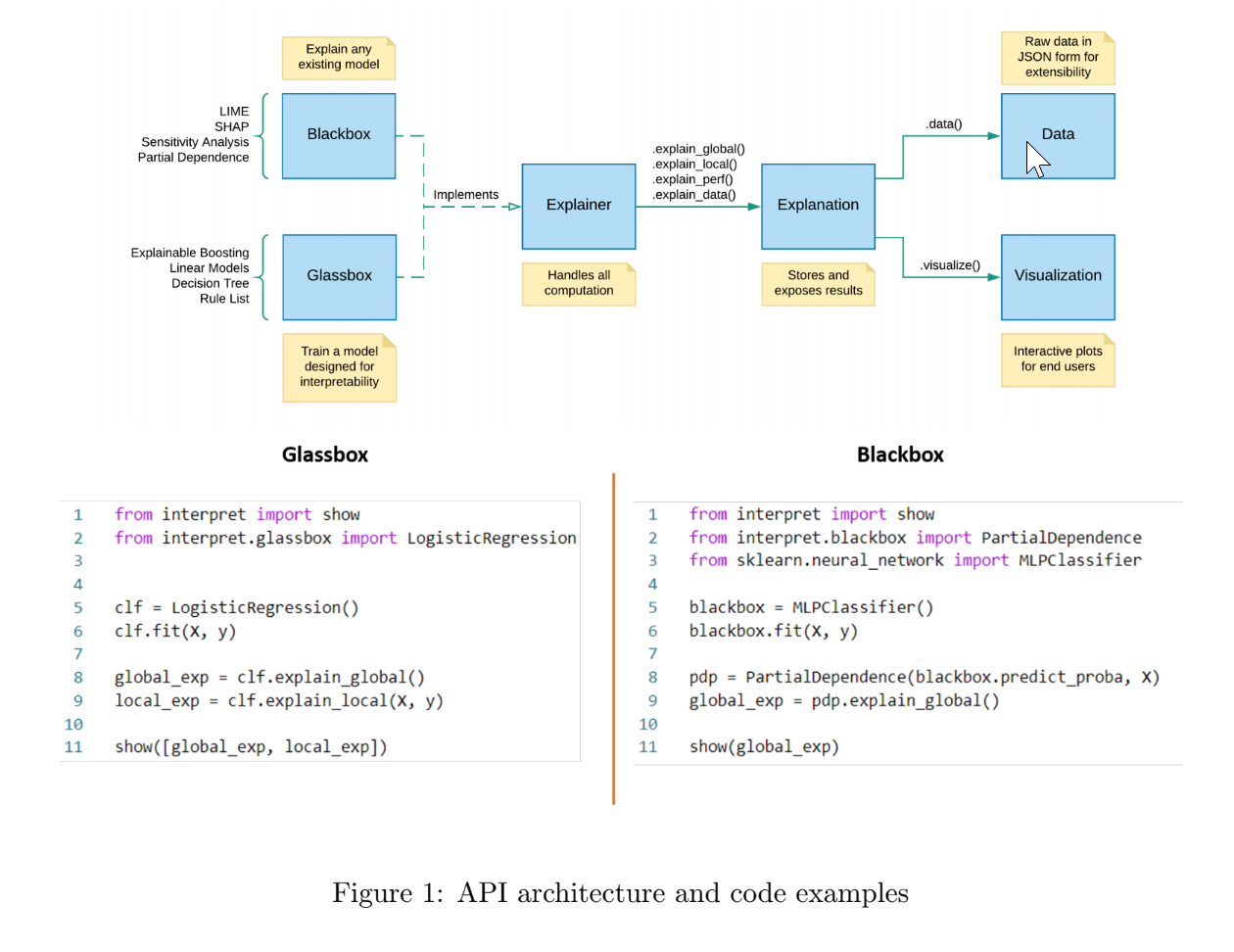


Figure . InterpretML package API architecture and code examples

Like Fairlearn, InterpretML is also integrated into Azure ML[[1]](#footnote-2).

With InterpretML and Fairlearn, you can gain a very good understanding of your models. Both are available integrated directly in Azure ML.

For additional information on InterpretML, you can consider the following resources:

* Project landing page at <https://interpret.ml/>.
* GitHub repo at <https://github.com/interpretml>.
* AI Show (video) [How to Explain Models with IntepretML Deep Dive](https://channel9.msdn.com/Shows/AI-Show/How-to-Explain-Models-with-IntepretML-Deep-Dive)
* And more specifically for the Azure ML integration, these two articles:
  + Concept: [Model interpretability in Azure Machine Learning (preview)](https://docs.microsoft.com/en-us/azure/machine-learning/how-to-machine-learning-interpretability)
  + How-to: [Use the interpretability package to explain ML models & predictions in Python (preview)](https://docs.microsoft.com/en-us/azure/machine-learning/how-to-machine-learning-interpretability-aml)

So, it’ time to see InterpretML in actions with a hands-on tutorial.

### Hands-on tutorial: Blackbox explainability

In this hands-on tutorial, we will start with the section InterpretML of the notebook understand\_rai\_tools.ipynb used so far to illustrate Fairlearn. You can scroll down below the section Explaining Blackbox Classifiers.

The purpose here is to show how blackbox model explainability works using a simple binary classification example which consists of predicting income (more or less than 50k) using a Random Forest (RF) classifier.

First, make sure you install the *interpret* Python package using pip:

pip install interpret

Here is a peak at the dataset we use:

A picture containing graphical user interface

Description automatically generated

Figure . Subset of the dataset used for income prediction, the target column is “Income”

We first train a RF classifier after some dimensionality reduction with PCA.

from sklearn.ensemble import RandomForestClassifier

from sklearn.decomposition import PCA

from sklearn.pipeline import Pipeline

#Blackbox system can include preprocessing, not just a classifier!

pca = PCA()

rf = RandomForestClassifier(n\_estimators=100, n\_jobs=-1)

blackbox\_model = Pipeline([('pca', pca), ('rf', rf)])

blackbox\_model.fit(X\_train, y\_train)

Then we use InterpretML to look at:

1. The Model performance.

from interpret import show

from interpret.perf import ROC

blackbox\_perf = ROC(blackbox\_model.predict\_proba).explain\_perf(X\_test, y\_test, name='Blackbox')

1. The local explanations with LIME and SHAP.

from interpret.blackbox import LimeTabular

from interpret import show

#Blackbox explainers need a predict function, and optionally a dataset

lime = LimeTabular(predict\_fn=blackbox\_model.predict\_proba, data=X\_train, random\_state=1)

#Pick the instances to explain, optionally pass in labels if you have them

lime\_local = lime.explain\_local(X\_test[:5], y\_test[:5], name='LIME')

from interpret.blackbox import ShapKernel

import numpy as np

background\_val = np.median(X\_train, axis=0).reshape(1, -1)

shap = ShapKernel(predict\_fn=blackbox\_model.predict\_proba, data=background\_val, feature\_names=feature\_names)

shap\_local = shap.explain\_local(X\_test[:5], y\_test[:5], name='SHAP')

1. The global explanations with Morris Sensitivity and Partial Dependence.

from interpret.blackbox import MorrisSensitivity

sensitivity = MorrisSensitivity(predict\_fn=blackbox\_model.predict\_proba, data=X\_train)

sensitivity\_global = sensitivity.explain\_global(name="Global Sensitivity")

from interpret.blackbox import PartialDependence

pdp = PartialDependence(predict\_fn=blackbox\_model.predict\_proba, data=X\_train)

pdp\_global = pdp.explain\_global(name='Partial Dependence')

And then we show them all in a single dashboard.

# Show them all in one dashboard

show([blackbox\_perf, lime\_local, shap\_local, sensitivity\_global, pdp\_global])

This results in a multi-tab interactive dashboard where we can select to visualize any of the explanations we defined above. We show for example the performance and Morris sensitivity tabs in figure 9 below, please refer to the associated notebook to explore the other tabs.

Chart, line chart

Description automatically generated

Chart

Description automatically generated

Figure . Performance and Morris sensitivity tabs from the interpret dashboard

Let’s now consider the ‘glassbox side’ of InterpretML.

### Hands-on tutorial: training a Glassbox model with Explainable Boosting Machine

You can continue with the notebook understand\_rai\_tools.ipynb. You can scroll down below the section Interpretable glassbox classification using EBM.

For the sake of simplicity, we use the same dataset used in the previous blackbox explainability hands-on tutorial.

We fit an Explainable Boosting Machine (EBM) to the data:

from interpret.glassbox import ExplainableBoostingClassifier

ebm = ExplainableBoostingClassifier() ebm.fit(X\_train, y\_train)

# or substitute with LogisticRegression, DecisionTreeClassifier or any other glassbox model

Now we can show the global behavior of the model (how the score varies with the age here):

from interpret import show

ebm\_global = ebm.explain\_global()

show(ebm\_global)

Chart

Description automatically generated

Figure . EBMs are interpretable by design, here we show how the score achieved by the model varies with respect to the age feature

And we can also explain individual predictions:

ebm\_local = ebm.explain\_local(X\_test, y\_test)

show(ebm\_local)

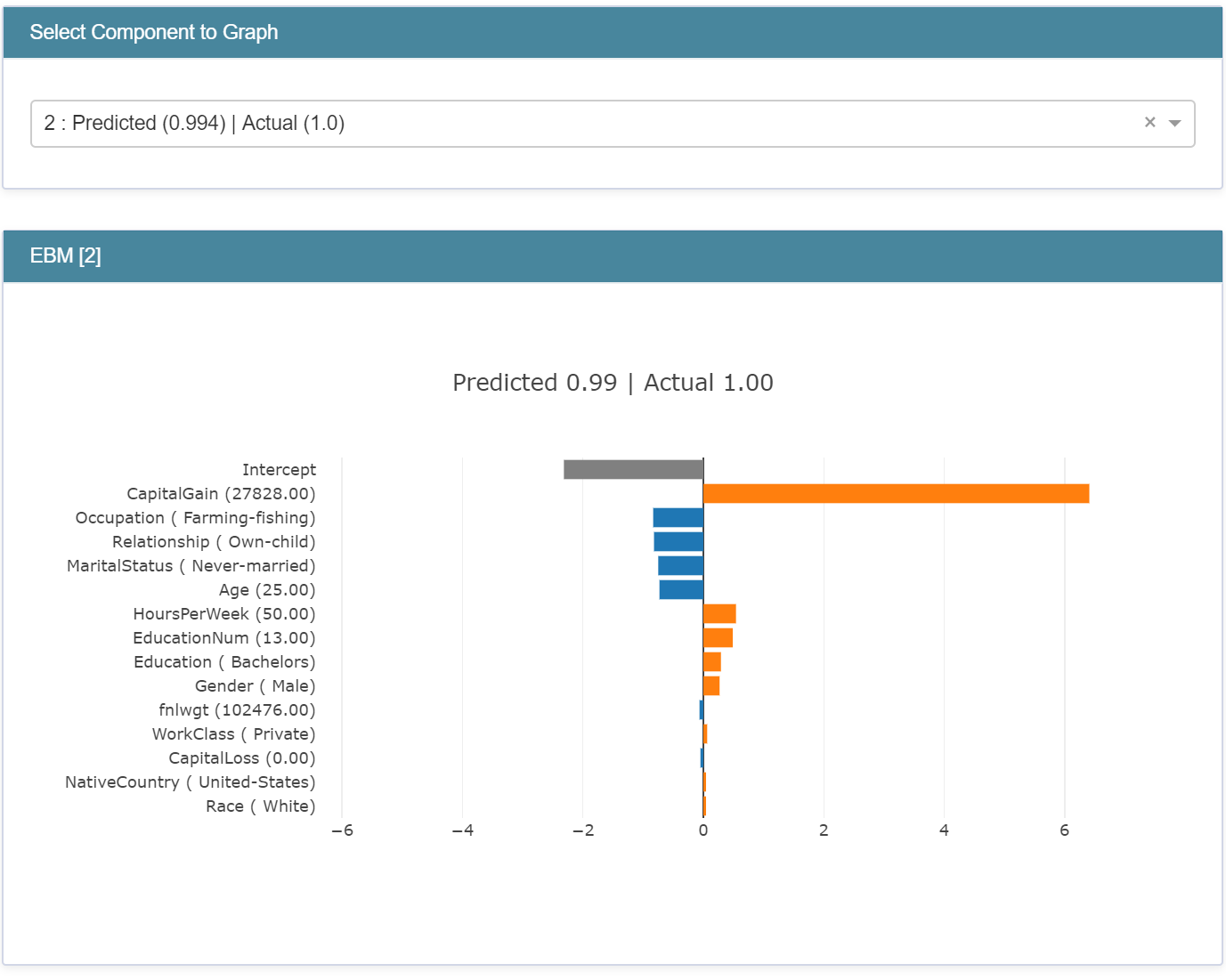


Figure . Built-in local explainability for EBMs

It is also worth mentioning that EBM overall performance is much better than other glassbox models like logistic regression or decision trees, as shown in the associated notebook, while still being interpretable by design.

## Error-Analysis

[Error-Analysis](https://erroranalysis.ai/) was initially created for our Analysis Platform, and its usage there has shown and validated this tool to be an effective and valuable approach for debugging models.

This internal toolkit has then been released in the open (in partnership with MSR). Error-Analysis is now part of the [open-source](https://github.com/microsoft/responsible-ai-widgets/) Responsible-AI-Widgets package that enables you to get a deeper understanding of ML model errors. When evaluating a ML model, a single score like the aggregate accuracy is not sufficient to understand where the model went wrong and may hide important conditions of inaccuracies between cohorts of your data.

This is where Error Analysis comes into play, it allows you to:

* Identify cohorts with higher error rates versus the overall benchmark and visualize how the error rate is distributed.
* Diagnose the root causes behind these errors.

Let’s cover these in order.

### Identification of errors

Error Analysis identifies cohorts of data with higher error rate than the overall benchmark. These discrepancies might occur when the system or model underperforms for specific demographic groups or infrequently observed input conditions in the training data. Two different methods are proposed by the Error Analysis toolkit:

* Decision Tree, which allows you to discover cohorts with higher error rates across multiple features using an intuitive binary tree visualization. Investigating indicators such as error rate, error coverage, and data representation for each discovered cohort in the decision tree allows you to form hypothesis on what features induce the most failure of your model.

Chart, bubble chart

Description automatically generated with medium confidence

Figure . Decision tree showing error rates for each cohort

* Error Heatmap: once you form hypotheses of the most impactful features for failure, use the Error Heatmap to further investigate how one or two input features impact the error rate across cohorts.

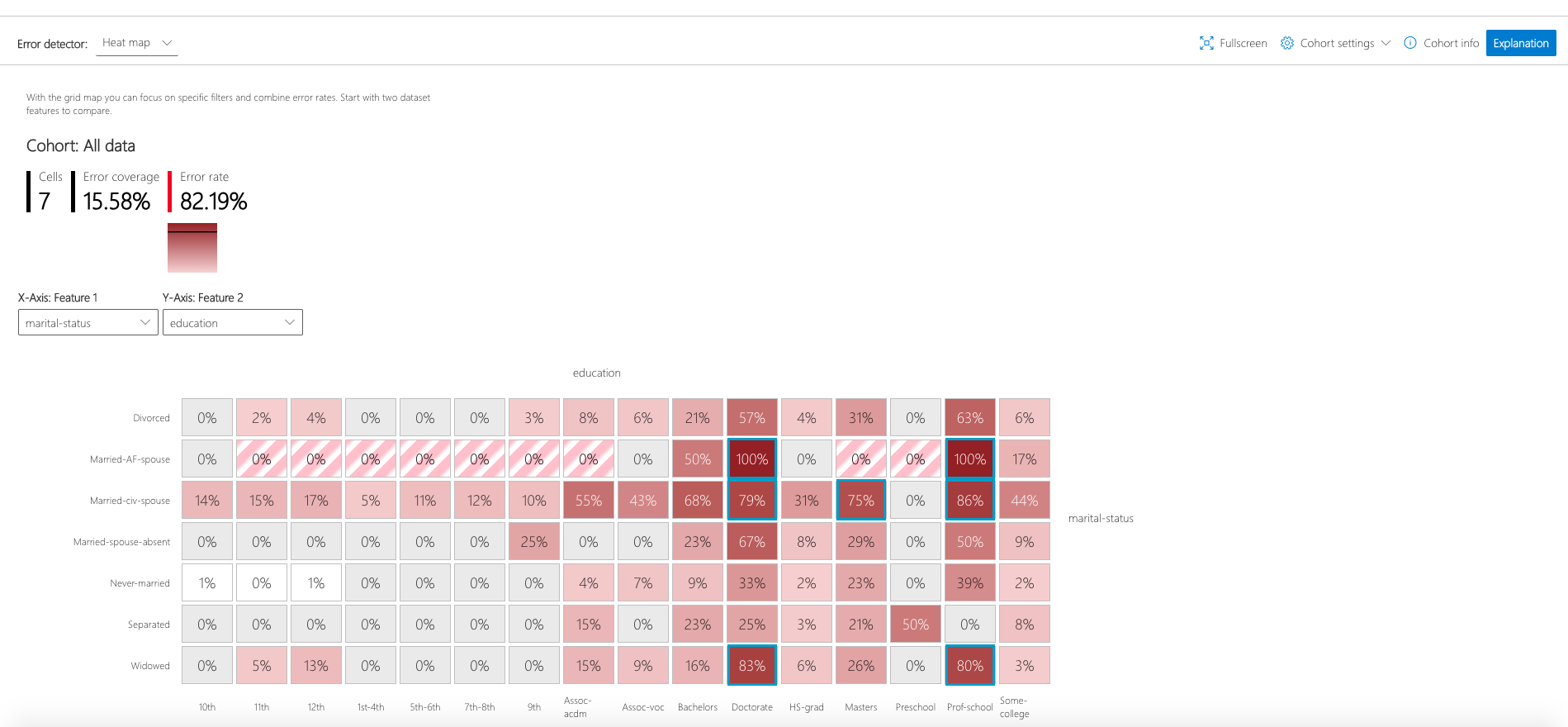


Figure . Error heatmap according to two features: education and marital status

### Diagnosis of errors

Identifying cohorts with higher error rates is only the first step of the error analysis process, the second step is to debug the errors in these cohorts further through exploratory data analysis and model explainability. The Error Analysis toolkit allows you to perform:

* Data Exploration, which compares cohort data statistics and feature distributions with other cohorts or to benchmark data. This allows us you to investigate whether certain cohorts are underrepresented or if their feature distribution is significantly different from the overall data.
* Global Explanation, which explores the top K important features that impact the overall model global explanation for a selected cohort of data and compares these features with those from other cohorts or benchmark.
* Local Explanation, which enables you to understand individual data points from the cohort have correct or incorrect prediction and compare this local explanation with other cohorts. This allows visual identification of any missing features or label noise that could lead to issues.
* What-if analysis (Perturbation Exploration), which applies changes to feature values of selected data point and observe resulting changes to the prediction.

While InterpretML allows for global and local interpretability, Error Analysis complements the picture by allowing you to detect inaccuracies between cohorts of your data, that is to say cohorts with higher error rate than the overall benchmark this is error identification, and then use model explanation for that cohort to understand what drives the error rates up so you can take corrective actions.

For additional information on Error Analysis, you can consider the following resources:

* Project landing page at <https://https://erroranalysis.ai>
* GitHub repo at <https://github.com/microsoft/responsible-ai-widgets/#getting-started>

### Hands-on tutorial

In this hands-on tutorial, we follow the section Error Analysis of the same notebook understand\_rai\_tools.ipynb used so far.

The goal is to show how to visualize model errors as well as global and local explanations using the Responsible AI Widget's Error Analysis visualization dashboard. To get there we build a model that classifies types of wine using scikit-learn, and then we analyze model errors and explanations using the Error Analysis dashboard.

Step 1:Import required packages and load the wine data from scikit-learn.

from sklearn.datasets import load\_wine

from sklearn import svm

from interpret.ext.blackbox import MimicExplainer

from interpret.ext.glassbox import LGBMExplainableModel

wine = load\_wine()

X = wine['data']

y = wine['target']

classes = wine['target\_names']

feature\_names = wine['feature\_names']

Step 2:Train a SVM classification model.

from sklearn.linear\_model import LogisticRegression

clf = svm.SVC(gamma=0.001, C=100., probability=True)

model = clf.fit(x\_train, y\_train)

print("number of errors on test dataset: " + str(sum(model.predict(x\_test) != y\_test)))

This prints the following: “number of errors on test dataset: 25”

We notice that the model makes a fair number of errors, but we have no idea why. This is where the Error Analysis Dashboard is useful.

Step 3:Identification of errors using decision trees and error heatmaps from the ErrorAnalysis dashboard (without explanations – see next step)

from raiwidgets import ErrorAnalysisDashboard

predictions = model.predict(x\_test)

ErrorAnalysisDashboard(dataset=x\_test, true\_y=y\_test, features=feature\_names, pred\_y=predictions)

The decision Tree tab of the dashboard generated by the previous command shows the following:

Diagram

Description automatically generated

Figure . Error heatmap for wine classification

We clearly see from this decision tree that the biggest error rate (19 errors out of 36 predictions – more than half) occurs when the feature malic\_acid is higher than 2.26. This is clearly a problem, and we need to investigate this cohort more punctually, which we will do in the next step.

Step 4:Running the Interpret-Community's 'explain\_model' globally and locally to generate model explanations.

from raiwidgets import ErrorAnalysisDashboard

predictions = model.predict(x\_test)

ErrorAnalysisDashboard(dataset=x\_test, true\_y=y\_test, features=feature\_names, pred\_y=predictions)

Step 5:Analyze model errors and explanations using Error Analysis dashboard by feeding model explanations generated in previous Step 4.

from raiwidgets import ErrorAnalysisDashboard

ErrorAnalysisDashboard(global\_explanation, model, dataset=X\_test, true\_y=y\_test)

This way we are able to investigate the specific cohort where we identified a larger error rate in the previous step against the entire dataset, for example by comparing feature importance like in the figure below.

Chart, bar chart

Description automatically generated

Figure . Feature importance for the problematic cohort (in orange) against the entire data (in blue).

## Responsible AI Dashboard

Until now, we have used the Error Analysis and Fair Learning dashboards independently. However, there is now another way to do this. Responsible AI dashboard is a single pane of glass designed to help practitioners accelerate their engineering processes by bringing together existing tools for responsible AI. These tools can help users evaluate and improve their machine learning models in terms of interpretability, fairness, accuracy, causality and counter factuality.

The dashboard is available on the GitHub repository [responsible-ai-toolbox](https://github.com/microsoft/responsible-ai-toolbox) and in a form of a Python library named [raiwidgets](https://pypi.org/project/raiwidgets/). It is mainly divided into two parts:

1. The Model Debugging part that provides tools to create fluid debugging experiences using interactive visualizations that identify errors, inspect the data, generate global and local explanations for models, and potentially inspect problematic examples.
2. The Responsible Decision-Making part that helps business stakeholders providing tools to explore causal relationships in the data and take informed decisions in the real world.

Diagram

Description automatically generated

Figure . Global structure of the Responsible AI dashboard ([source](https://github.com/microsoft/responsible-ai-toolbox))

While the Responsible AI dashboard was originally designed to be deployed locally, in a Jupyter notebook for example, it was recently fully integrated into Azure Machine Learning (Azure ML) platform along with a new feature : the Responsible AI Scorecards.

In the following, we will first see what tools are available in the Model Debugging part and in the Responsible Decision-Making part, see what their uses are, then we will see a practical case of creating a Responsible AI dashboard in local before dealing with the latest news related to Azure ML native integration of Responsible AI Dashboard.

### Dashboard Tools

#### Model Debugging

Model debugging is critical for Responsible AI in order to assess and determine how and why AI systems behave the way they do. Data scientists can use this knowledge to improve the fairness of the model and its performance. Conceptually, model debugging consists of three stages that we will identify below and associate with tools available in the dashboard.

1. The first stage is to identify errors in order to understand and recognize them. The dashboard provides the following tools:
   * Error Analysis: identify cohorts with high error rate.
   * Model Statistics: aggregate a variety of model assessment metrics, showing model prediction distributions.
2. The second stage is to diagnose the reasons behind the identified errors thanks to:
   * Model Interpretability: interpret and debug model.
   * Counterfactual Analysis (and What-If style questions): generate diverse counterfactual explanations for debugging.
   * Exploratory Data Analysis: understand dataset characteristics.
3. The third and final stage is to use the identification and diagnosis insights from previous stages to take targeted mitigation steps:
   * Unfairness mitigation: mitigate fairness issues (Fairlearn).
   * Data Enhancements: enhance the dataset and retrain the model.

#### Responsible Decision-Making

Responsible Decision-Making is an important feature of the field of artificial intelligence (AI) that is increasingly put into practice nowadays since we expect AI to help us make business decisions. The dashboard provides tools to inform decision-making processes based on both data-driven insights and model-driven insights. Some examples below:

* Data-driven insights w/ causal Inference: this tool provides data-driven insights allowing to understand heterogeneous treatment effects on an outcome. It allows the ability to answer questions such as “*how would a medicine impact a patient’s blood pressure?*”
* Model-driven insights w/ Counterfactual Analysis and What-If: this tool provides model-driven insights allowing to answer end-user questions using model outputs.

### Dashboard Implementation

#### Local implementation

Generating a Responsible AI dashboard is quite simple. The instructions below will help you understand how it works.

Step 1:As in the first part, users will need to load a dataset, spit it into train and test datasets, and train a model on the training dataset.

Step 2:Then, it is necessary to initialize a RAIInsights object upon which the different components can be loaded. task\_type holds the string 'regression' or 'classification' depending on the developer's purpose.

Users can also specify categorical features via the categorical\_features parameter.

If needed, it’s possible to define a sklearn pipeline which you’ll be able to pass to the RAIInsights constructor.

# If a pipeline is needed:

dashboard\_pipeline = Pipeline(steps=[('preprocess', feat\_pipe), ('model', model)])

rai\_insights = RAIInsights(dashboard\_pipeline, train\_data, test\_data, target\_feature, task\_type)

# If no pipeline is needed:

rai\_insights = RAIInsights(model, train\_data, test\_data, target\_feature, task\_type)

Step 3:We select the analyses we would like to perform, here *Interpretability* and *Error Analysis*

# Interpretability

rai\_insights.explainer.add()

# Error Analysis

rai\_insights.error\_analysis.add()

Step 4:Finally, we can compute and display the insights

rai\_insights.compute()

ResponsibleAIDashboard(rai\_insights)

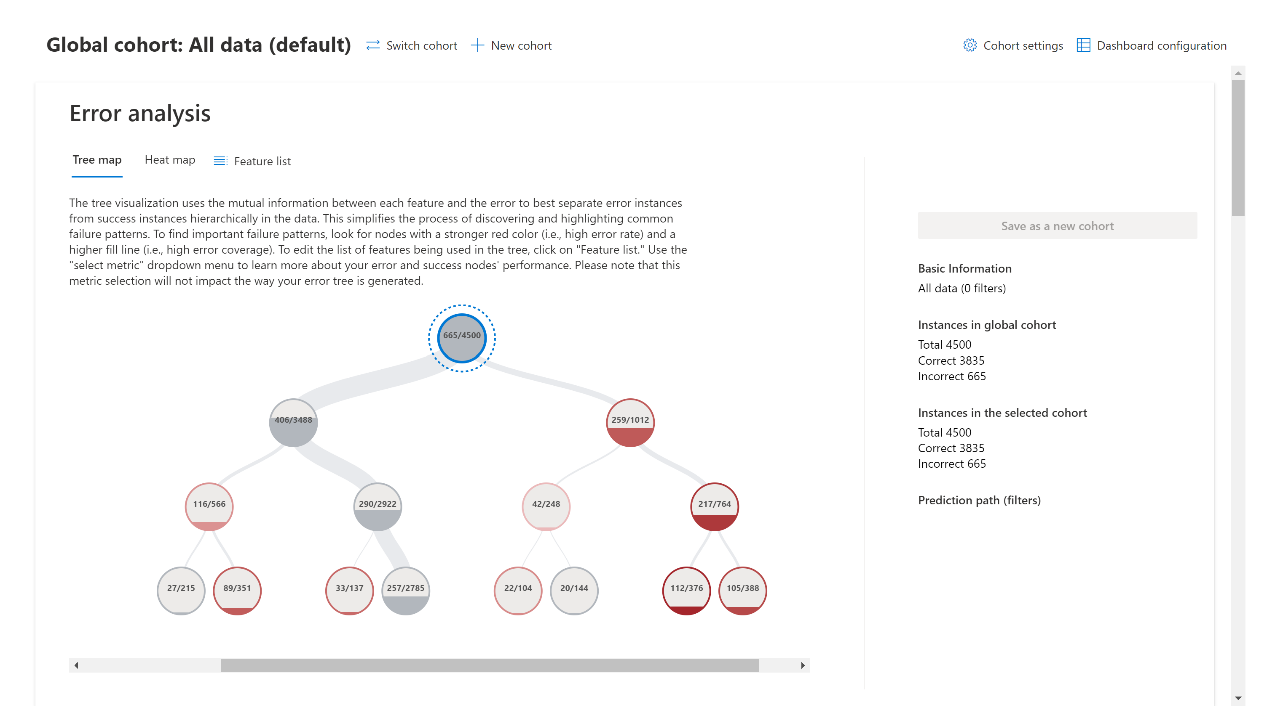


Figure : Screenshot of the “Error analysis” part of a Responsible AI dashboard in operation

#### Native implementation of Responsible AI dashboards & scorecards in Azure Machine Learning

Azure ML now natively integrates the Responsible AI dashboard for each compatible model (MLFlow type model) registered on the platform. The dashboard works the same way as the previous version except that it can now be generated in several different ways.

* Generate Responsible AI dashboard with [YAML and Python](https://docs.microsoft.com/en-us/azure/machine-learning/how-to-responsible-ai-dashboard-sdk-cli?tabs=yaml).
* Generate Responsible AI dashboard with the [no-code guided UI wizard in Azure ML studio](https://docs.microsoft.com/en-us/azure/machine-learning/how-to-responsible-ai-dashboard-ui).

thumbnail image 2 of blog post titled 
 
 
  
 
 
 
    
  
   
    
      
       Responsible AI Dashboard and Scorecard in Azure Machine Learning
       
      
     
   
  
 
   
 
 
 
 
 


Figure : Responsible AI dashboard creation using the no-code guided UI wizard in Azure ML ([source](https://techcommunity.microsoft.com/t5/ai-machine-learning-blog/responsible-ai-dashboard-and-scorecard-in-azure-machine-learning/ba-p/3391068))

Another feature of the native dashboard integration in Azure ML is that it is possible to attach multiple dashboards to a single model. Simply select a model to access the list of dashboards that have been generated.

Graphical user interface, text, application, email

Description automatically generated

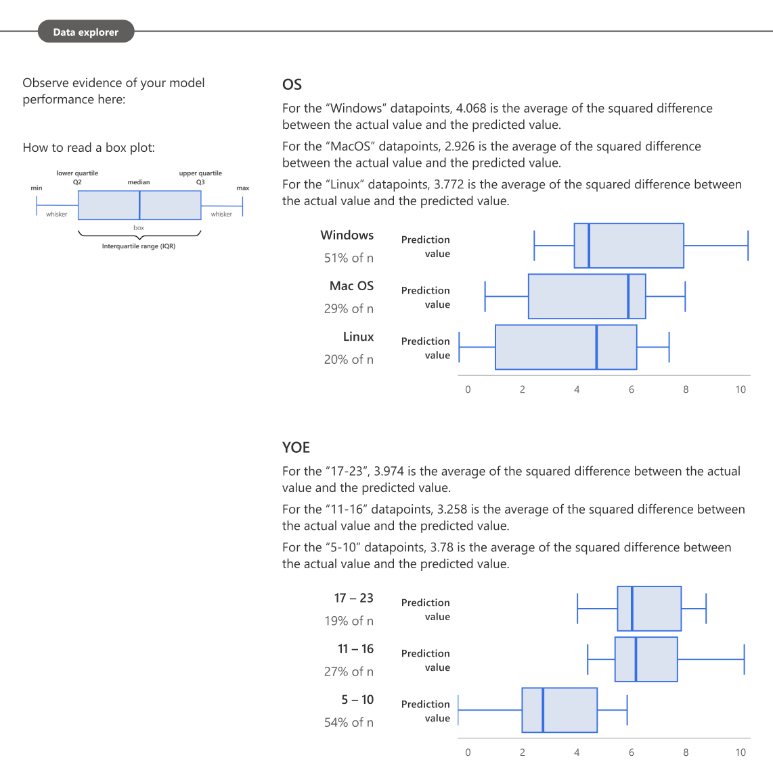
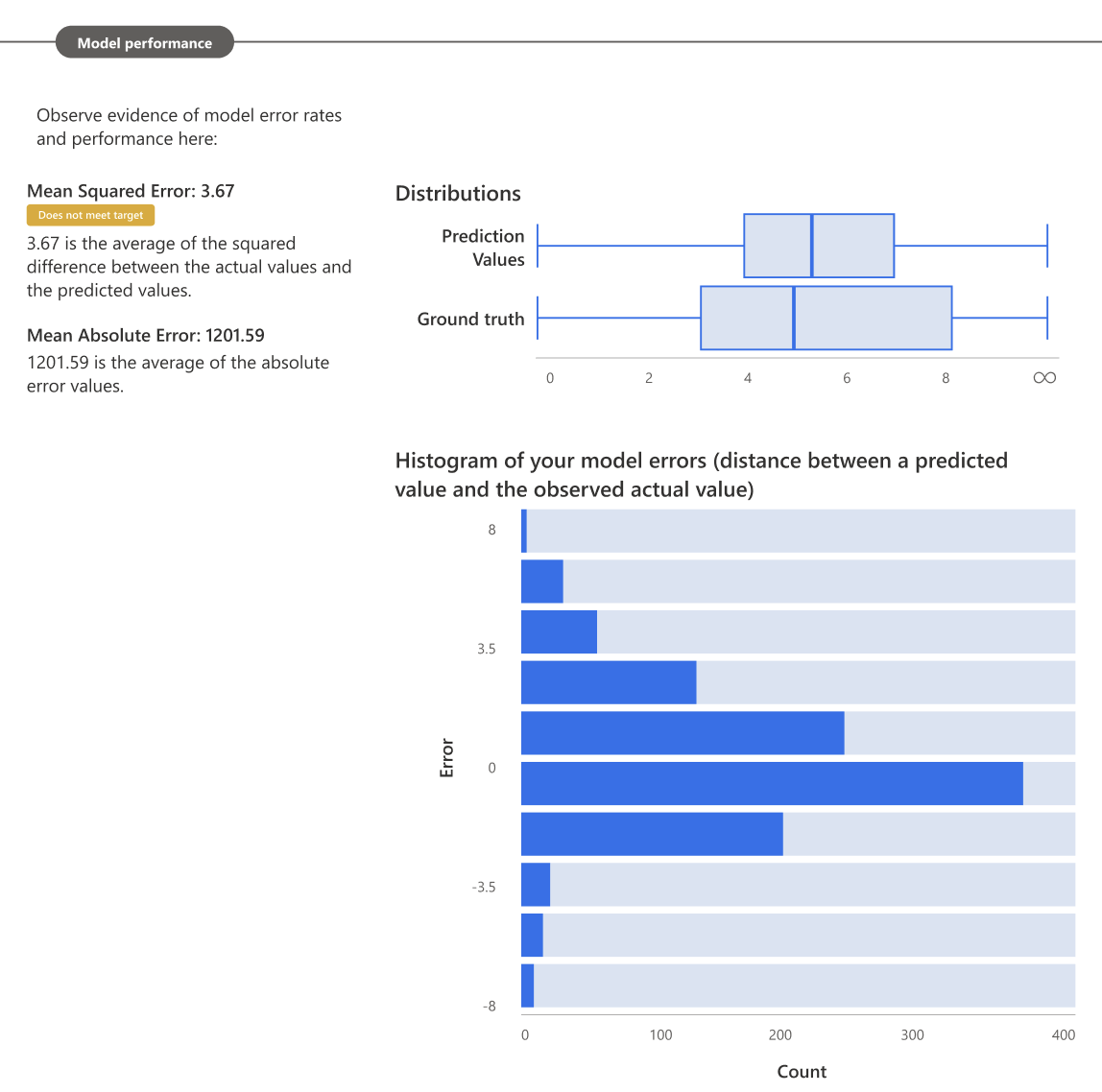
Figure : An easy way to generate multiple dashboards for a single model in Azure ML

This version of the dashboard, unlike the previous one, also offers the generation of Responsible AI scorecards. These are .pdf reports that can be customized, downloaded and shared with stakeholders to inform them about the health and compliance of a machine learning model with the goal of building stakeholder trust.

To [generate a scorecard](Creating%20a%20dashboard%20using%20the%20no-code%20guided%20UI%20wizard%20in%20Azure%20ML), you need to use your domain expertise around the problem to define your desired target values on the performance and equity metrics of the model. You have to select the inputs of the Responsible AI scorecard component among several categories such as: model choice, evaluation metrics, feature importance and fairness. At the end, the generated scorecard is accessible in the Responsible AI (preview) tab.

[thumbnail image 23 of blog post titled 
 
 
  
 
 
 
    
  
   
    
      
       Responsible AI Dashboard and Scorecard in Azure Machine Learning
       
      
     
   
  
 
   
 
 
 
 
 
](https://techcommunity.microsoft.com/t5/ai-machine-learning-blog/responsible-ai-dashboard-and-scorecard-in-azure-machine-learning/ba-p/3391068)

Figure : List of all generated Responsible AI scorecards ([source](https://docs.microsoft.com/en-us/azure/machine-learning/how-to-responsible-ai-scorecard#how-to-read-your-responsible-ai-scorecard))

Graphical user interface, text, application, email

Description automatically generatedAnd the resulting scorecard might look like this (these are only some parts of the entire scorecard):

Figure : Screenshots from the generated scorecard ([source](https://docs.microsoft.com/en-us/azure/machine-learning/how-to-responsible-ai-scorecard#how-to-read-your-responsible-ai-scorecard))

This concludes this module about Responsible AI tools to understand the behavior of AI systems, off to the next module which explores tools to protect these AI systems and/or their data(set(s)).

# Module 2: Protecting your AI systems and your data assets

Let’s now look discuss how to protect both your AI systems and your data assets, i.e., against any adversarial attack, any potential misuse used to train the models and/or during inference.

Similarly, this comprises a set of practices and techniques to articulate as well. With adversarial attacks on both the ML algorithms and data that keep increasing, and as these ML-powered features and/or systems become more pervasive, the need to understand how they fail, whether by the hand of an adversary or due to the inherent design of a system, will only become more pressing to leverage the suitable techniques as part of the design, the development, the deployment, along with the monitoring of these features and/or systems.

Regarding the failure modes, they range:

* From *intentional failures* wherein the failure is caused by an active adversary attempting to subvert the system to attain her goals – either to misclassify the result, infer private training data, or to steal the underlying algorithm.
* To *unintentional failures* wherein the failure is because an ML feature or system produces a formally correct but completely unsafe outcome.

Discussing all the related implications would lead us to a number of considerations to articulate and is definitely outside the scope of this starter guide. For more information, see guide [Framing a (more) Trustworthy AI Lifecycle for your AI-powered solutions](https://github.com/microsoft/responsible-ai-workshop/blob/main/nongen-ai-trustworthy-ai-lifecycle/docs/framing-trustworthy-ai-lifecycle.docx), also part of this Responsible AI Workshop.

But, at this stage, this doesn’t prevent in any way to introduce some entry-level considerations. This will be the purpose of the next section, where we will consider both a technique to assess the security of AI systems, and further discuss the [threat modeling](https://strikecommunity.azurewebsites.net/articles/1941/course-threat-modeling-101.html) of such systems, and its implications.

This indeed also supposes beyond such a state-of-the-art understanding, modeling, and assessment, to consider a number of techniques, would it be in terms of privacy-preserving machine learning (PPML) techniques that applies for the development, to name a few Homomorphic Encryption, Secure Multiparty Computing, and Differential Privacy, or other techniques for the deployment of the considered AI systems, the [confidential inference](https://github.com/microsoft/onnx-server-openenclave) being an illustration with the capabilities we provide with [Azure Confidential Computing](https://azure.microsoft.com/en-us/solutions/confidential-compute/), See for example the [”Data in Use Protection Compass“ Workshop](https://aka.ms/DataInUseProtectionWS).

With this understanding, and for the rest of this module, we will then more specifically focus on the PPML techniques, and specifically consider two techniques : anonymization and differential privacy.

So, let’s start with Counterfit and threat modeling activities.

## Counterfit

### Using Microsoft Security Development Lifecycle (SDL) practices to ensure minimal protection of an AI based system

The exponential growth in the development of AI allows for technological breakthroughs in many other areas that would have been unimaginable until now.

This rapid development of Machine Learning and data science in general can be explained by several factors such as the improvement of hardware components that now allow for highly efficient distributed computing. This is also due to the ever-increasing amount of qualitative data that allows to train more efficiently Machine Learning models that are deeper and deeper and therefore more capable of abstraction.

However, this ultra-fast development cannot be done without considering the cybersecurity aspect of the technology. Indeed, the development of machine learning models can lead developers to work with sensitive or even confidential data. Some Machine Learning models can carry heavy financial, ecological, and even human responsibilities.

ENISA's December 15, 2020 report entitled “[Artificial Intelligence Cybersecurity Challenges](https://www.enisa.europa.eu/publications/artificial-intelligence-cybersecurity-challenges)” presents an expanded taxonomy of all potential threats to AI systems. Among the different threats, we find adversarial attacks that allow an attacker to confuse a ML model in its decision-making process.

The attacker may have several objectives in mind:

* **Confidence reduction** when the goal is to make the model less confident in its predictions.
* **Misclassification** when the goal is to make the model no longer able to correctly classify an input which has been previously slightly modified.
* **Targeted misclassification** when the goal is to force the model to misclassify an input, which has been previously slightly modified, with a different but selected target.

To be able to execute these attacks it is necessary to determine the accesses that we have to the model to choose an adapted type of attack.

* **White-box attack**. The term "white-box" refers to the fact that the attacker has a full access to the model and its parameters to prepare the attack.
* **Black-box attack** . The term "black-box" refers to the fact that the attacker tries to compromise the model blindly without knowing how the model is built.

The Jupyter notebook for this hands-on tutorial is adversarial\_attacks\_counterfit.ipynb located in the tooling-tutorials\hands\_on\_tutorials\adversarial\_attacks\_counterfit sub-directory underneath the folder where you have cloned the workshop repo, for example the folder rai-workshop in our illustration, please refer to the section Cloning the tutorials’ Jupyter notebooks above. This notebook is accompanied by a series of complementary files for the sake of this hands-on tutorial.

With this hands-on tutorial, the related notebook and the companion files, you are introduced with an execution of a black-box attack and a white-box attack on a vanilla model using the above-mentioned [Counterfit](https://github.com/Azure/counterfit), a dedicated tool for evaluating the security of AI systems.

As before, you need to open up this file in the Jupyter environment of your choice, please refer to the section Guide prerequisites above. Please also note that this notebook can be run according to your preference ; either in the cloud on Azure as shown before, or locally. If you don't have a machine suitable for training Machine Learning models, you can simply load the already trained model fashion\_mnist\_model.h5 that is in the same folder as the notebook.

### Hands-on tutorial

#### Training a vanilla model on the Fashion MNIST dataset

The model we are going to attack is shown below. It is a very simple image classification model trained on the Fashion MNIST dataset.

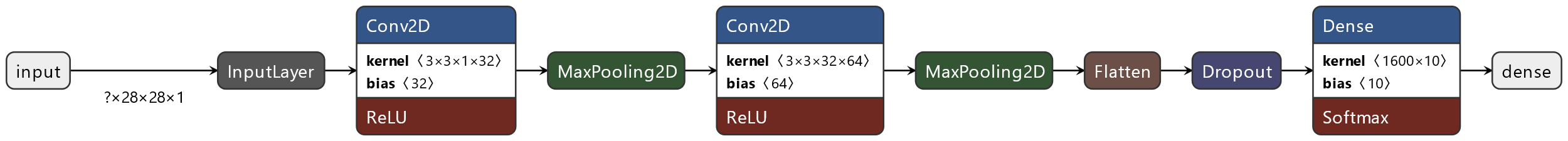


Figure . Model architecture

This dataset is a more complex version of the classic MNIST dataset but has the advantage of being composed of images having the same shape as those composing the classic version. The Fashion MNIST dataset is composed of 70,000 grayscale images in a resolution of 28 pixels by 28 pixels. The train set is composed of 60,000 images and the test set of 10,000 images of clothing articles as shown in Fig. 29 below.

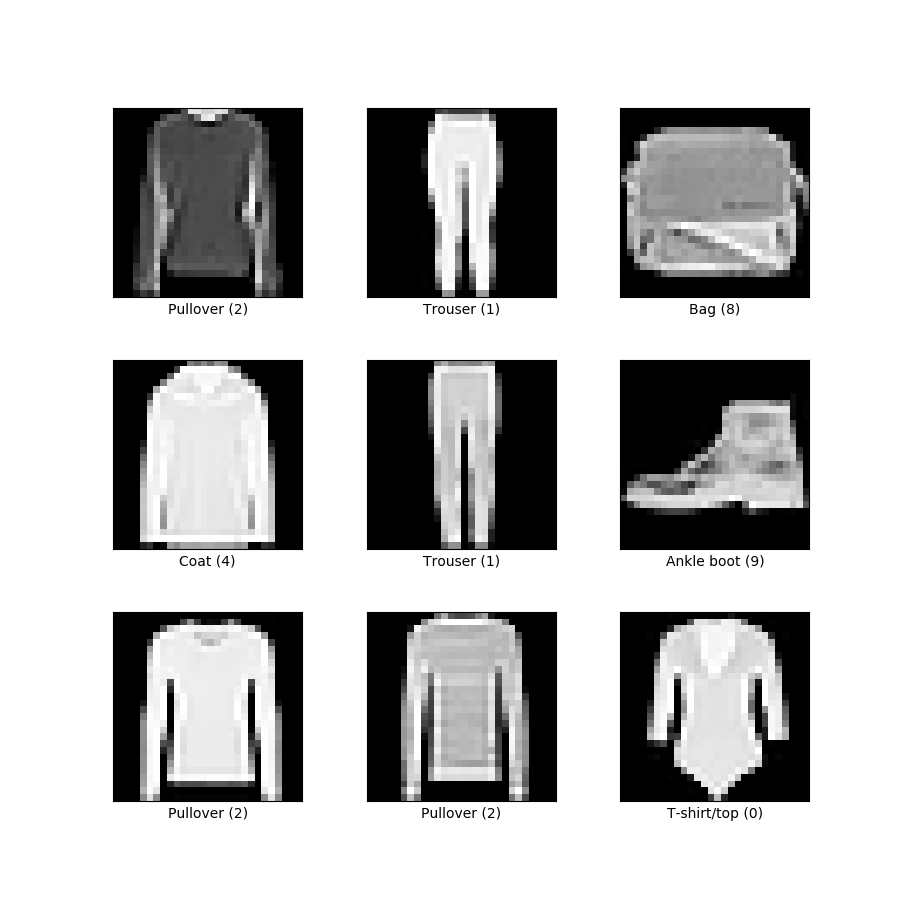


Figure . Representation of some images with their classes from the Fashion MNIST dataset

To load the model rather than training it from scratch, simply run the following cell:

model = keras.models.load\_model("fashion\_mnist\_model.h5")

#### Performing the adversarial attacks

A picture containing text, display

Description automatically generatedIn the following, we will perform the attacks using the possibilities provided by Counterfit. The objective will be to perform a misclassification attack on the image below.

**True class : 4**

**Predicted as : 4**

Figure . Image indexed at position 10 in the Fashion MNIST dataset.

To perform the attacks, we must first choose which algorithm to use depending on whether we want to implement a black-box or white-box attack. Fortunately, Counterfit can handle both cases.

* For the white-box scenario, we will use the [Carlini L Inf](https://arxiv.org/abs/1608.04644)attack
* For the black-box scenario, we will use the [HopSkipJump](https://arxiv.org/abs/1904.02144)attack

The notebook precisely describes the steps to perform the attacks if you ever want to reproduce them. Please follow them in order.

In the following, we will analyze the results obtained.A picture containing text, clock

Description automatically generatedA picture containing text, display

Description automatically generated

True class : 4

**Predicted as : 6**

True class : 4

**Predicted as : 6**

Figure . Result from the Carlini L Inf attack Figure . Result from the HopSkipJump attack

The results obtained show that our two attacks went well. The two images generated are both very similar to the original image and yet the model does not classify them correctly.

The image resulting from the white-box attack is practically indistinguishable from the original if we do not take the time to analyze both of them carefully. As for the image resulting from the black-box attack, it is also very similar to the original image, but it can be observed that it is a bit noisier.

Although this notebook is not a formal proof, we can still hypothesize that white-box attacks can achieve better results than black-box attacks. And this can be explained quite simply by the fact that in black-box attacks we have much less information to create our custom input than in white-box attacks.

A picture containing qr code

Description automatically generatedUne image contenant texte, main, clipart

Description générée automatiquementWithout going into the details of how the algorithms we have chosen work, we can display the difference between the generated image and the original image in order to understand how the original image has been modified to fool the model.

Figure . Difference image from the Carlini L Inf attack

Figure . Difference image from the HopSkipJump attack

#### Protecting the system against adversarial attacks

The consequences of adversarial attacks on Machine Learning models can be dramatic depending on the level of criticality in the decisions made. If the model has important responsibilities in terms of economic, societal, or even human stakes, it is absolutely necessary to protect the model from these threats.

One possibility to protect a system based on an AI/ML model is to follow the instructions given by the [Microsoft Security Development Lifecycle (SDL)](https://www.microsoft.com/en-us/securityengineering/sdl) or at least some of the best practices that can give interesting leads.

[Practice #4](https://www.microsoft.com/en-us/securityengineering/sdl/threatmodeling) consists in performing a Threat Modeling. This allows us to make a structured analysis of the potential risks on our system.

A picture containing chart

Description automatically generated

Figure . Threat Modeling

The image above shows the result of threat modeling on a very simplified version of our system where we would deploy our model in the cloud as an API. This API would expose endpoints that would be callable by IoT devices like connected cameras.

We see that there are several important information points to consider ensuring a minimum level of security in our system. If we want to avoid adversary attacks, we can focus our attention on these lines:

* *An adversary can inject malicious inputs into an API and affect downstream processes. (1)*
* *If proper authentication is not in place, an adversary can spoof a source process or an external entity and gain unauthorized access to the web application. (2)*

The threat modeling tool shows us that we can run into security problems if we don't control user input (1) and run the service without authentication (2).

An adversarial attack involves the misuse of the inference part of a Machine Learning model. The first step to prevent this is to detect these abusive uses. To do this, we can imagine several solutions, but the simplest is to observe when the number of requests to the model exceeds a certain threshold to be determined beforehand. If the number of requests exceeds the threshold, we can assume that the model is under attack and that we must act accordingly.

In this type of attack, it is important to quickly restrict or even cut the connection between the attacker and the model to preserve the system. However, we do not want to cut off access to the model to all users but only to the attacker. Hence the interest in implementing an authentication system to allow case-by-case decisions.

That said, it appears to be a good idea to restrict access to the model itself by authenticating each user and setting a limit on the number of requests to the model to limit the possibility of adversarial attacks.

**As already stated, the above only constitutes both an illustration and an introduction of the subject.**

To better understand how to build a secure AI starting from the [threat modeling](https://strikecommunity.azurewebsites.net/articles/1941/course-threat-modeling-101.html), we recommend considering the following articles to pursue your investments in this area:

* [Securing the future of AI and machine learning at Microsoft](https://www.microsoft.com/security/blog/2019/02/07/securing-the-future-of-ai-and-machine-learning-at-microsoft/)
* [Failure Modes in Machine Learning](https://docs.microsoft.com/en-us/security/engineering/failure-modes-in-machine-learning)
* [Threat Modeling AI/ML Systems and Dependencies](https://docs.microsoft.com/en-us/security/engineering/threat-modeling-aiml)
* [AI/ML Pivots to the Security Development Lifecycle Bug Bar](https://docs.microsoft.com/en-us/security/engineering/bug-bar-aiml)

As well as the video [AI Security Engineering—Modeling/Detecting/Mitigating New Vulnerabilities](https://www.youtube.com/watch?v=SiACfPJblAs).

## Presidio

Anonymization is the process of obscuring Personally Identifiable Information (PII) in a manner that prevents it from uniquely identifying an individual. Anonymization reduces the risk of accidental disclosure of PII data, and if a data breach does occur, the stolen information will be of no use to attackers in trying to identify individuals.

Let’s shortly share an example to set the context and illustrate the limitations.

Back in 2006, Netflix announced a [$1 million prize](https://en.wikipedia.org/wiki/Netflix_Prize) for improving their movie recommendation service, releasing a movie database which was “carefully” anonymized by deleting personal details and substituting names with random numbers. Less than 18 months later, Arvind Narayanan et al. published the [paper](https://www.semanticscholar.org/paper/How-To-Break-Anonymity-of-the-Netflix-Prize-Dataset-Narayanan-Shmatikov/c40e5c8b4957074644acdaf1f9f4332e63b5846b#matched) entitled Robust “How To Break Anonymity of the Netflix Prize Dataset” where they “successfully identified the Netflix records of known users, uncovering their apparent political preferences and other potentially sensitive information”.

If we had to take a single takeaway from the Netflix data re-identification story, it is that “anonymized data isn’t”, as has rightfully said Cynthia Dwork from Microsoft Research (MSR) and one of the pioneers of research in model security and privacy, See [Privacy and accuracy: How Cynthia Dwork is making data analysis better](https://blogs.microsoft.com/ai/privacy-and-accuracy-how-cynthia-dwork-is-making-data-analysis-better/).

The above will lead us to explore two tools for the sake of this starter guide.

The first tool called Presidio is a personally identifiable information (PII) identifier and anonymizer, which cannot be used standalone to give any privacy guarantees taking into account the previous but can be combined with the second one, the SmartNoise system, which provides a concrete implementation of Cynthia’s revolutionary Differential Privacy (DP) concept, See [Differential Privacy: A Primer for the Perplexed](https://unece.org/fileadmin/DAM/stats/documents/ece/ces/ge.46/2011/26_Dwork-Smith.pdf).

So, it’s high time to introduce Presidio.

[Presidio](https://microsoft.github.io/presidio/) is an [open-source](https://github.com/Microsoft/presidio) data protection and anonymization SDK for text and images providing fast identification and anonymization of private entities in text such as credit card numbers, names, locations, social security numbers, bitcoin wallets, US phone numbers, financial data and more.

Presidio's modules include:

1. The Presidio analyzer for custom or predefined PII detection in text, leveraging Named Entity Recognition, regular expressions, rule-based logic, and checksum with relevant context in multiple languages.
2. The Presidio anonymizer for allowing anonymization of the detected PII entities using different operators.
3. The Presidio image redactorfor redacting PII text in images.

Text

Description automatically generated

Figure . Inside Presidio identification and anonymization modules

\*NER: Named entity recognition.

With that in mind, let’s see how to use Presidio for simple PII analysis and anonymization on text and how to customize the above Presidio PII analyzer to detect new types of PII entities.

### Hands-on tutorial

The Jupyter notebook for this hands-on tutorial is protect\_rai\_tools.ipynb located in the tooling-tutorials\hands\_on\_tutorials sub-directory underneath the folder where you have cloned the workshop repo, for example the folder rai-workshop in our illustration, please refer to the section Cloning the tutorials’ Jupyter notebooks above.

You can scroll down the section Presidio.

Step 1: Installing the presidio\_analyzer and presidio\_anonymizer libraries using pip along with the spaCy English language model needed by the analyzer.

pip install presidio\_analyzer

pip install presidio\_anonymizer

# Presidio analyzer requires a spaCy language model.

Python -m spacy download en\_core\_web\_lg

Step 2:Once the presidio-analyzer package is installed, run this simple analysis script.

From presidio\_analyzer import AnalyzerEngine

text\_to\_anonymize = “His name is Mr. Jones and his phone number is 212-555-5555”

analyzer = AnalyzerEngine()

analyzer\_results = analyzer.analyze(text=text\_to\_anonymize, entities=[“PHONE\_NUMBER”], language=’en’)

print(analyzer\_results)

This will print the result of the PII analysis, in this case the detected phone numbers in the provided text.

Step 3:Creating Custom PII Entity Recognizers.

From presidio\_analyzer import PatternRecognizer

text\_to\_anonymize = “His name is Mr. Jones and his phone number is 212-555-5555”

titles\_recognizer = PatternRecognizer(supported\_entity=”TITLE”,

deny\_list=[“Mr.”,”Mrs.”,”Miss”])

pronoun\_recognizer = PatternRecognizer(supported\_entity=”PRONOUN”,

deny\_list=[“he”, “his”, “she”, “hers”])

analyzer.registry.add\_recognizer(titles\_recognizer)

analyzer.registry.add\_recognizer(pronoun\_recognizer)

analyzer\_results = analyzer.analyze(text=text\_to\_anonymize,

entities=[“TITLE”, “PRONOUN”],

language=”en”)

print(analyzer\_results)

The previous code snippet:

1. Creates custom titles and pronouns recognizers.
2. Adds the new custom recognizers to the analyzer.
3. Calls analyzer to get results from the new recognizers.

It prints the titles and pronouns detected in the provided text.

Step 4:Anonymizing the identified PII entities.

From presidio\_anonymizer import AnonymizerEngine

from presidio\_anonymizer.entities.engine import OperatorConfig

anonymizer = AnonymizerEngine()

anonymized\_results = anonymizer.anonymize(

text=text\_to\_anonymize,

analyzer\_results=analyzer\_results,

operators={"DEFAULT": OperatorConfig("replace", {"new\_value": "<ANONYMIZED>"}),

"PHONE\_NUMBER”: OperatorConfig("mask", {"type": "mask", "masking\_char" : "\*", "chars\_to\_mask" : 12, "from\_endv : True}),

"TITLE": OperatorConfig(“redact”, {})}

)

anonymized\_results.to\_json()

The previous code snippet:

1. Sets up the anonymizer engine.
2. Creates an anonymizer request – text to anonymize, list of anonymizers to apply and the results from the analyzer request.
3. Anonymizes the text.

It prints the anonymized text along with a list of the detected PII entities like this:

Text: "His name is <ANONYMIZED> and <ANONYMIZED> phone number is \*\*\*\*\*\*\*\*\*\*\*\*".

- Items:

- {"start": 59, "end": 71, "entity\_type": "PHONE\_NUMBER", "text": "\*\*\*\*\*\*\*\*\*\*\*\*", "operator": "mask"},

- {"start": 30, "end": 42, "entity\_type": "PRONOUN", "text": "<ANONYMIZED>", "operator": "replace"},

- {"start": 13, "end": 25, "entity\_type": "PERSON", "text": "<ANONYMIZED>", "operator": "replace"},

- {"start": 12, "end": 12, "entity\_type": "TITLE", "text": "", "operator": "redact"}]}

This ends our investigation of Presidio, let’s now jump into the realm of Differential Privacy with the SmartNoise system.

## SmartNoise

### You said Differential Privacy, *what do you mean?*

#### Considering membership inference attacks

Sensitive and confidential information about individuals is extensively used and shared between companies, government entities, research organizations, and other parties to train ML models. Using only black-box access to such an ML model, an adversary can determine if a sample was a member of the training set used to build this model, this is called a membership inference attack. Inadequate usage of this kind of information can result in significant consequences, such as harm to an individual’s reputation, employability, creditworthiness, and insurability.

Invented by Microsoft Research (MSR) and associates, Differential Privacy (DP) is considered the gold standard for protecting individuals’ data against membership inference attacks. It provides a mathematically measurable privacy guarantee to individual data subjects and offers significantly higher privacy levels than commonly used disclosure limitation practices like data anonymization. The latter increasingly shows vulnerability to re-identification attacks – especially as more data about individuals become publicly available.

#### The issue with traditional data anonymization approaches

The crucial problem with anonymized data is that the released records often include unique combinations of variables (digital fingerprints) that someone might link to other publicly available information to re-identify specific people. For instance, research has shown that 87% of Americans can be uniquely identified with only three pieces of data: Gender, birthday, and ZIP code.

A typical goal for today’s data disclosure practices is to achieve a standard known as *k-anonymity*.This is achieved by minimizing the number of attributes that are particularly vulnerable to re-identification or reducing details by grouping values into brackets (e.g., age brackets). For example, a released dataset satisfies 5-anonymity if at least five records exist for each combination of gender, age, and ZIP code. While this approach likely reduces the hit rate that an attacker can achieve, it is far from solving the problem and fails to provide any reliable privacy guarantee to individuals.

#### The Differential Privacy (DP) concept

Differential Privacy (DP) requires that any analytical results on a dataset A including an individual’s record are identical (or at least very close) to the analytical results on a dataset B where the individual’s record has been removed. Differential Privacy aims to mask the contribution of the individuals record by adding a precisely tuned amount of random noiseto the data.

The amount of noise that is introduced to the computation must be chosen carefully. On the one hand, higher quantities of noise increase the level of privacy. On the other side, it is more difficult to derive reliable statistical results when the noise level is too high. There is a tunable parameter available to adjust the amount of noise in the trade-off between privacy and utility. This is known as the privacy parameter epsilon. It is also called the privacy budget.

### The SmartNoise system

[SmartNoise](https://smartnoise.org/) is a joint project by Microsoft and Harvard’s Institute for Quantitative Social Science (IQSS) and the School of Engineering and Applied Sciences (SEAS) as part of the [OpenDP](https://opendp.org/) initiative. It aims to make Differential Privacy broadly accessible.

The SmartNoise tools primarily focus on the “global model” of Differential Privacy where a trusted data collector is presumed to have access to unprotected data and wishes to protect public releases of aggregate information. For example, a hospital having access to patients’ information and wishing to release aggregated statistics about these patients without affecting their privacy.

SmartNoise is an open-source project that contains different components for building global differentially private systems. SmartNoise is made up of the following top-level components (only the core library is explored here):

* [SmartNoise Core library](https://github.com/opendp/smartnoise-core)
* [SmartNoise SDK library](https://github.com/opendp/smartnoise-sdk)

The SmartNoise core library includes the following privacy mechanisms for implementing a differentially private system:

* Analysis: A graph description of arbitrary computations to perform on the data. These can include statistics like count and mean, or utilities like filtering and imputation.
* Validator: A Rust library that contains a set of tools for checking and deriving the necessary conditions for an analysis to be differentially private. Support for Python is also available.
* Runtime**:** The medium to execute the analysis. The reference runtime is written in Rust but runtimes can be written using any computation framework such as SQL and Spark depending on your data needs.
* Bindings**:** Language bindings and helper libraries to build analyses. SmartNoise currently provides [Python bindings.](https://github.com/opendp/smartnoise-core-python#more-about-smartnoise-core-python-bindings)

### Hands-on tutorial: Protecting statistics against reconstruction attacks

In this hands-on tutorial, we will explore how data can be protected against reidentification attacks using Differential Privacy and the SmartNoise system by following [this notebook](https://github.com/opendp/smartnoise-samples/blob/master/whitepaper-demos/2-reidentification-attack.ipynb) from the SmartNoise samples repository. Please be aware that only the most important parts of the code are shown here, so please follow the notebook for completeness.

The goal is to show how an attacker can leverage basic demographic information like age and zip codes to reidentify individuals even when the sensitive data is published in an anonymized format. Then we show how Differential Privacy can help prevent such an attack.

Step 1:Importing the data

We will use anonymized medical dataset and some sample demographic information dataset for performing the identification attack. The code below prints samples of each of these two datasets.

import reident\_tools as reident

from opendp.smartnoise.synthesizers.mwem import MWEMSynthesizer

df\_medical = pd.read\_csv(‘data/data\_medical.csv’, sep=”,”, encoding=”utf-8”).infer\_objects()

print(‘Anonymized dataset including sensitive medical information:’)

display(df\_medical.iloc[:,1:].sample(8))

df\_demographic = pd.read\_csv(‘data/data\_demographic.csv’, sep=”,”, encoding=”utf-8”).infer\_objects()

print(‘Attacker`s data collection with basic demographic information:’)

Output:

Table

Description automatically generated

Table

Description automatically generated

We observe that some effort was deployed to anonymize the original dataset containing sensitive medical information, by deleting the name of the patient, providing an age bracket instead of the exact age and making the last two digits of the zip code. We assume that the attacker on the other hand has access to this basic demographic data through his own data sources.

Step 2: Reidentification attack

Now, we perform the reidentification attack using the try\_reidentification function. As input, we use the data sets generated above (anonymized medical and the attacker’s demographic data).

Reident\_attack = reident.try\_reidentification(df\_demographic, df\_medical, logger)

print(f’Sample of re-identified patients:’)

reident\_attack[reident\_attack[“ID\_Match”]==True][[‘Name’, ‘Gender’, ‘Age’, ‘Zip’, ‘Diagnosis’, ‘Treatment’, ‘Outcome’, ‘ID\_Match’]].sample(10)

This prints a sample of individuals we were able to identify along with their diagnosis data:

Table

Description automatically generated

A total of 9124 individuals were re-identified!

Step 3: Protecting the medical dataset with differential privacy using the MWEM synthesizer from SmartNoise.

Let’s look at the data before encoding and synthesizing:

df\_reident\_synth = df\_medical[[‘Gender’, ‘Age’, ‘Zip’, ‘Diagnosis’, ‘Treatment’, ‘Outcome’]].copy()

df\_reident\_synth[‘Zip’] = df\_demographic[‘Zip’].copy()

df\_reident\_synth[‘Age’] = df\_demographic[‘Age’].copy()

# Have a quick glance at the data

df\_reident\_synth.head()

Table

Description automatically generated

Now we encode the data using the do\_encode-function to make it compatible with the MWEM synthesizer:

# Encode the data set and display it

df\_reident\_encoded = reident.do\_encode(df\_reident\_synth, [‘Gender’, ‘Age’, ‘Zip’, ‘Diagnosis’], reident.diseases)

df\_reident\_encoded.head()

Table

Description automatically generated

Finally, we synthesize new data using the SmartNoise MWEM synthetizer (by adding noise to the original data:

# Apply the synthesizer to the dataset

synthetic\_data = MWEMSynthesizer(Q\_count = 400, epsilon = 3.00, iterations = 60, mult\_weights\_iterations = 40, splits = [], split\_factor = 1)

synthetic\_data.fit(df\_reident\_encoded.to\_numpy())

df\_synthesized = pd.DataFrame(synthetic\_data.sample(int(df\_reident\_encoded.shape[0])), columns=df\_reident\_encoded.columns)

# Compare original and synthetic data

reident.create\_histogram(df\_reident\_encoded, df\_synthesized, ‘Diagnosis\_encoded’, reident.diseases)

Below, is a histogram comparing the diagnoses distribution of both the original and the synthesized datasets. We see that the distributions are pretty similar, which means that not much information is lost during the synthetization process, even though the individual entries are completely different in this new synthetic dataset.

Chart, bar chart

Description automatically generated

Figure . Histogram showing diagnosis distribution for both the original and synthetic datasets

Step 4:Reidentification attack on synthesized data

Finally, we try the re-identification attack on the synthesized data using the try\_reidentification\_noise-function. As explained, the synthesized data set has new combinations of demographic data, so we do not deal with the raw/real data anymore which drastically reduces the risk of a potential reidentification match.

Here are how the original and synthesized datasets look like:

print('Medical Dataset:')

display(df\_medical\_synth.sample(5))

print('\nSynthesized Demographic Dataset:')

display(df\_synthesized.sample(5))

Table

Description automatically generated

Table

Description automatically generated

Now we try to perform the reidentification attack:

reident\_attack\_2 = reident.try\_reidentification\_noise(df\_synthesized, df\_medical\_synth, logger)

print(f'Found {len(reident\_attack\_2)} potential matches!')

This prints “Found 0 potential matches!” as expected.

All-in-all, this hands-on tutorial demonstrates the magic behind Differential Privacy, it allows you to protect data against reidentification attacks by masking individual contributions and providing mathematical guarantees of privacy, while still preserving the distribution and thus summary statistics of the data.

To continue your exploration on how you can protect personal data against privacy attacks for your ML models, you can read the white paper [Microsoft SmartNoise Differential Privacy Machine Learning Case Studies](https://azure.microsoft.com/en-us/resources/microsoft-smartnoisedifferential-privacy-machine-learning-case-studies).

This white paper provides practical guidance on how personal data can be rigorously protected for applications like statistics, machine learning, and deep learning using Differential Privacy. Interestingly enough, it also provides you with a number for interactive demo scenarios:

* Protecting statistics against reconstruction attacks, *sounds familiar*.
* Protecting sensitive data against re-identification attacks.
* Privacy-preserving statistical analysis.
* Machine learning using a differentially private classifier.
* Generating a synthetic dataset for privacy-preserving machine learning.
* Detect pneumonia in X-Ray images while protecting patients' privacy.

The related Jupyter notebooks are located here: <https://github.com/opendp/smartnoise-samples/tree/master/whitepaper-demos>.

You are all set to go through them.

This concludes this module about tools to protect AI systems data.

# As a conclusion

This concludes this guide, part of the Responsible AI Workshop. We hope you have enjoyed this guided tour on (some) of the RAI tooling available to help you put Responsible AI to work.

From holistically transforming industries to addressing critical issues facing humanity, AI is already solving some of our most complex challenges and redefining how humans and technology interact.

As part of this guided tour and its various hands-on tutorials, we have outlined some of the steps we are taking to prioritize Responsible AI along with some of the tooling we use within our company and make available outside in hopes that our experience can help other people and organizations like yours. But we only scratched the surface, and we are only at the beginning of the journey towards putting Responsible AI into practice.

# To go beyond

To continue learning about the passionate subject of Responsible AI, you can follow the other tutorials and walkthroughs available in this workshop.

Une image contenant texte, motif, point

Description générée automatiquementYou can also scan this code or visit <https://aka.ms/RAIresources> where you can access the entirety of already available tools, guidelines, and other additional resources that will help you create your next AI solution in a (more) responsible manner.

Une image contenant texte, capture d’écran, Site web, Page web

Description générée automatiquement

Une image contenant bleu, brouillard, capture d’écran, bleu vert

Description générée automatiquement

1. We updated Azure ML Studio’s visualization dashboard with the revamped [version 2.0 dashboard](https://github.com/interpretml/interpret-community#visualizations) of InterpretML currently available in open source. Additionally, we modularized the dashboard so that users can call model performance, dataset explorer, and aggregate/individual feature importance and what-if tabs through separate API calls. [↑](#footnote-ref-2)