

MLOps

Anshu Pandey

Introduction



Anshu Pandey

AI & Data Consultant

10+ years of experience as Data Scientists, AI Consultant and Architect

MCT, WCT, 10+ other certifications

Introduction



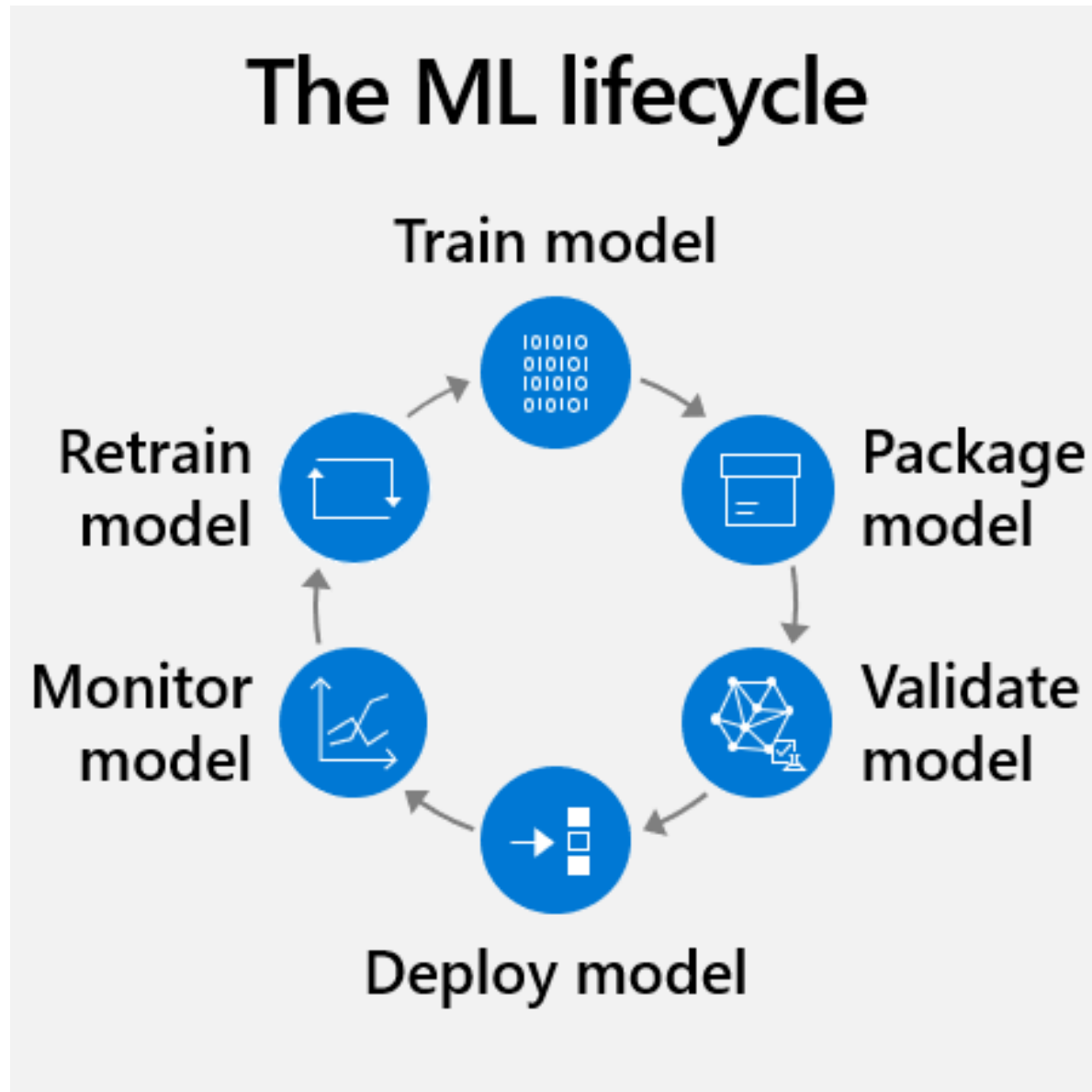
Tell more about yourself

- Your experience with
 - python, Azure cloud
 - Machine Learning
- Your expectation from this course

Common challenges with ML

- Cross-team alignment: Siloed teams impede workflow alignment and collaboration.
- Standard, repeatable processes: Without automated and repeatable processes, employees have to reinvent the wheel each time they create and deploy a new model.
- Resources: Large amounts of time and personnel are required to manage the lifecycle.
- Auditability: It can be difficult to ensure that models meet regulatory standards and performance thresholds over time.
- Explainability: Black box models make it difficult to understand how the model works.

Typical ML model lifecycle



What is MLOps?

MLOps stands for Machine Learning Operations. It is an approach to managing the lifecycle of machine learning models that emphasizes collaboration, automation, and monitoring.

MLOps aims to bridge the gap between data science and IT operations by applying principles and practices from software engineering to machine learning development and deployment.

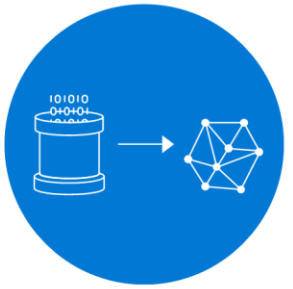
MLOps involves various tasks, such as data preparation, model training, deployment, monitoring, and maintenance, and relies on tools and technologies such as version control, continuous integration and deployment (CI/CD), containerization, and orchestration platforms.

MLOps helps organizations to build and deploy machine learning models more efficiently, reliably, and at scale, while reducing risks and costs associated with machine learning projects.

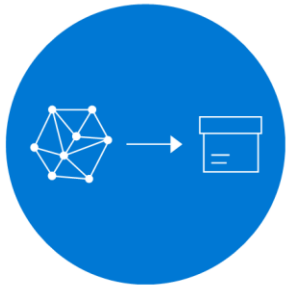
What is MLOps?

- Machine learning operations or MLOps aims to more efficiently scale from a proof of concept or pilot project to a machine learning workload in production.

1. Train model



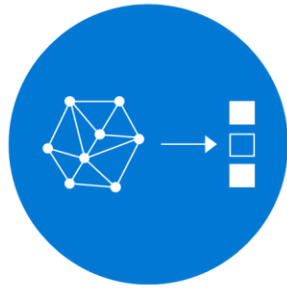
2. Package model



3. Validate model



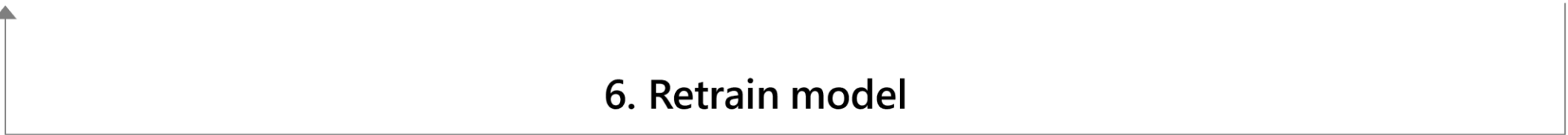
4. Deploy model



5. Monitor model



6. Retrain model



Why MLOps



Improved collaboration: MLOps promotes collaboration between data scientists, software engineers, and operations teams, enabling them to work together more effectively and efficiently



Faster time to market: By automating many of the tasks involved in model development and deployment, MLOps can help organizations bring new machine learning models to market more quickly.



Better model performance: MLOps emphasizes continuous monitoring and optimization of machine learning models, helping to ensure that they continue to operate effectively over time.



Reduced costs: By automating many of the tasks involved in model development and deployment, MLOps can help organizations reduce the costs associated with managing machine learning models.



Improved reliability: MLOps promotes the use of best practices for testing, deployment, and monitoring, helping to ensure that machine learning models operate reliably in production environments.

Principles of MLOps

Collaboration

- MLOps emphasizes collaboration between data scientists, software engineers, and operations teams. By working together, these teams can ensure that machine learning models are built and deployed effectively.

Automation

- MLOps promotes the use of automation to streamline the development and deployment of machine learning models. Automation can help to reduce the time and effort required to manage these models, and can improve their reliability and performance.

Version Control

- MLOps relies on version control to manage changes to machine learning models and related code. Version control helps to ensure that all changes are tracked and documented, and can help to facilitate collaboration among team members.

Testing

- MLOps emphasizes the importance of testing machine learning models to ensure that they are working correctly. Testing can help to identify and address issues before they become problems in production environments.

Monitoring

- MLOps promotes continuous monitoring of machine learning models in production environments. This can help to identify issues and opportunities for optimization, and can help to ensure that the models continue to operate effectively over time.

Deployment

- MLOps emphasizes the importance of automating the deployment of machine learning models to production environments. This can help to reduce the risk of errors and ensure that models are deployed consistently across different environments.

Tasks involved in MLOps

Data preprocessing

- Cleaning, transforming, and organizing data so that it can be used for training and evaluation.
- Example: Removing missing values from a dataset, normalizing numerical features, and encoding categorical variables.

Model selection

- Choosing the appropriate algorithm and hyperparameters to create models that meet specific business and technical requirements.
- Example: Deciding whether to use a linear regression or a decision tree algorithm to predict sales, and selecting appropriate values for parameters such as learning rate or regularization.

Model training

- Using data to train models, and optimizing them for specific use cases.
- Example: Using historical sales data to train a machine learning model that predicts future sales, and adjusting the model's parameters to maximize accuracy.

Tasks involved in MLOps

Model Testing

- Model explanation, Biasness Analysis, Fairness Analysis, response time, packing, environment, dependencies etc.
- Example: Performing detailed testing on model to ensure it matches business requirements

Model deployment

- Integrating models into production environments, often using containerization technologies such as Docker, and ensuring they can scale and operate reliably.
- Example: Deploying a machine learning model as a web service that can be accessed by other applications, and monitoring its performance to ensure it meets service level agreements.

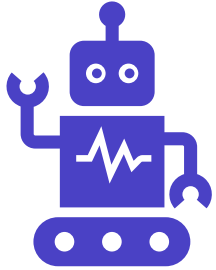
Model monitoring

- Collecting and analyzing data on model performance, identifying potential issues or problems, and responding quickly to minimize any impact on users.
- Example: Monitoring a machine learning model's predictions over time, and detecting if its accuracy starts to degrade or if it starts to produce biased results.

Model optimization

- Continuously improving models by refining algorithms, tweaking hyperparameters, and incorporating new data or feedback from users.
- Example: Incorporating customer feedback into a machine learning model that recommends products, and adjusting the model's parameters to better reflect user preferences.

Tools for MLOps



MLflow

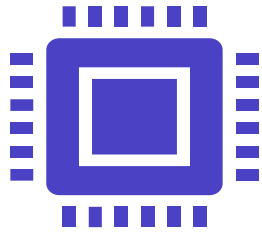
MLflow is an open-source platform for managing the lifecycle of machine learning models. It provides tools for tracking experiments, packaging code, and deploying models.



Apache Airflow

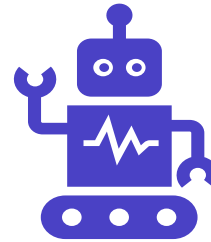
Apache Airflow is an open-source platform for creating, scheduling, and monitoring workflows. It provides a way to automate the steps involved in building and deploying machine learning models, including data ingestion, model training, and model deployment.

Tools for MLOps



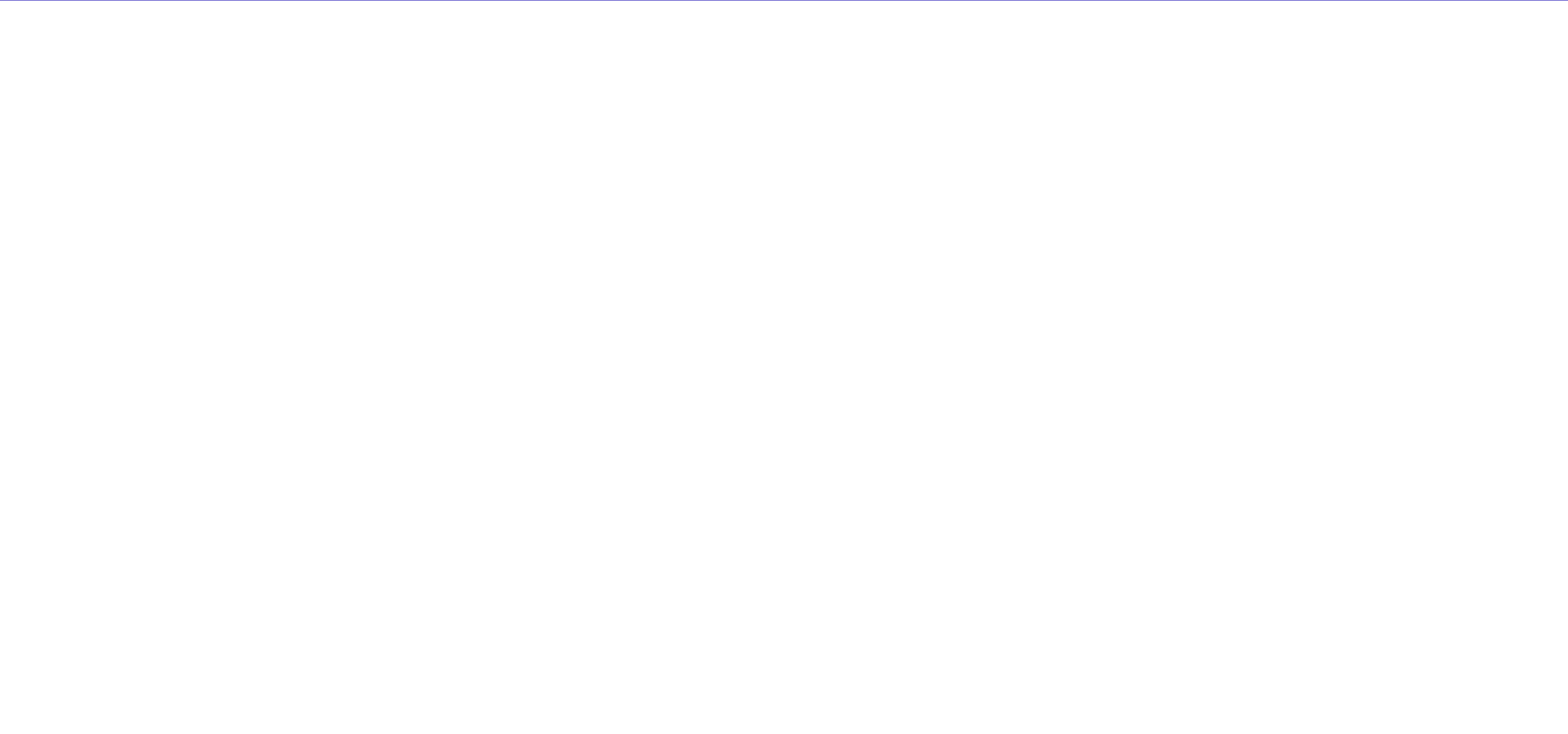
Docker

Docker is a containerization platform that is commonly used in MLOps to package and deploy machine learning models. With Docker, machine learning models can be packaged into a container image, along with all the dependencies needed to run the model. This makes it easy to deploy the model to any environment that supports Docker, such as a cloud platform or on-premise server.



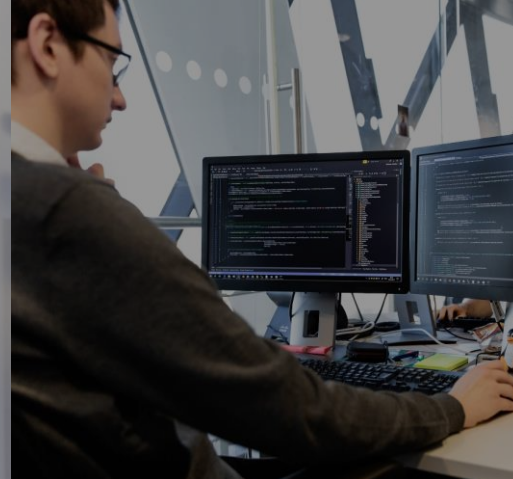
Kubeflow

Kubeflow is an open-source platform for building and deploying machine learning workflows on Kubernetes. It provides tools for model training, serving, and monitoring, as well as tools for managing data pipelines and model artifacts.



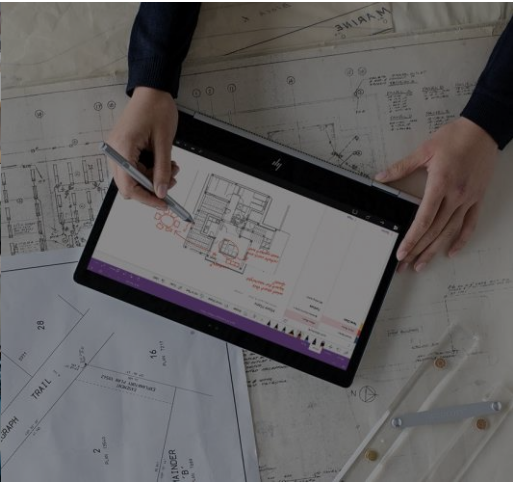
Agenda

- Introduction to Model Interpretability
- Using Model Explainers



Lesson 1

Introduction to Model Interpretability

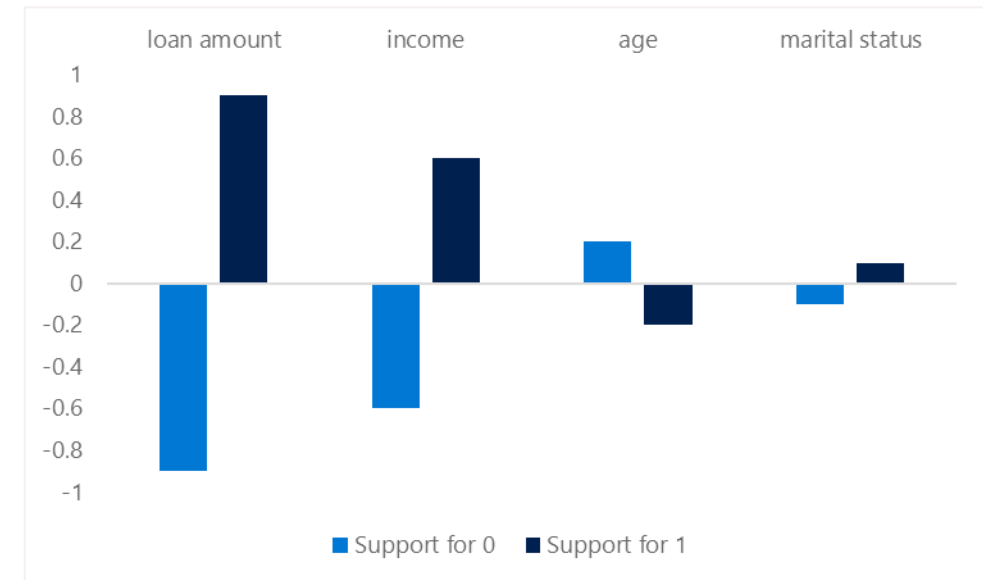
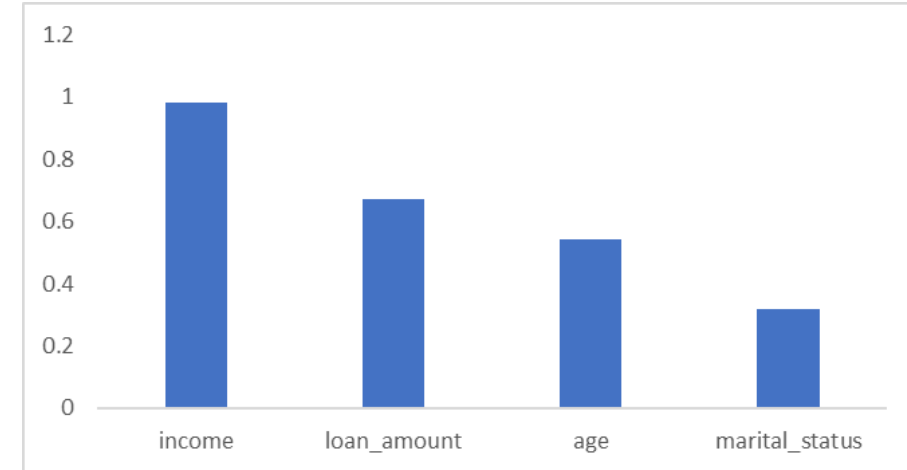


Model Interpretability in Azure Machine Learning

- Statistical explanation of feature importance
 - Quantifies the influence of each feature on prediction
 - Important to identify bias or unintended correlation in the model
- Based on the Open Source *Interpret-Community* package
 - Includes explainers based on common model interpretation algorithms like:
 - Shapely Additive Explanations (SHAP)
 - Local Interpretable Model-Agnostic Explanations (LIME)

Global and Local Feature Importance

- Global Feature Importance
 - Overall feature importance for all test data
 - Indicates the relative influence of each feature on the predicted label
- Local Feature Importance
 - Feature importance for an individual prediction
 - In classification, this shows the relative support for each possible class per feature



Automated Machine Learning Explanations

- Enable **model_explanability** parameter

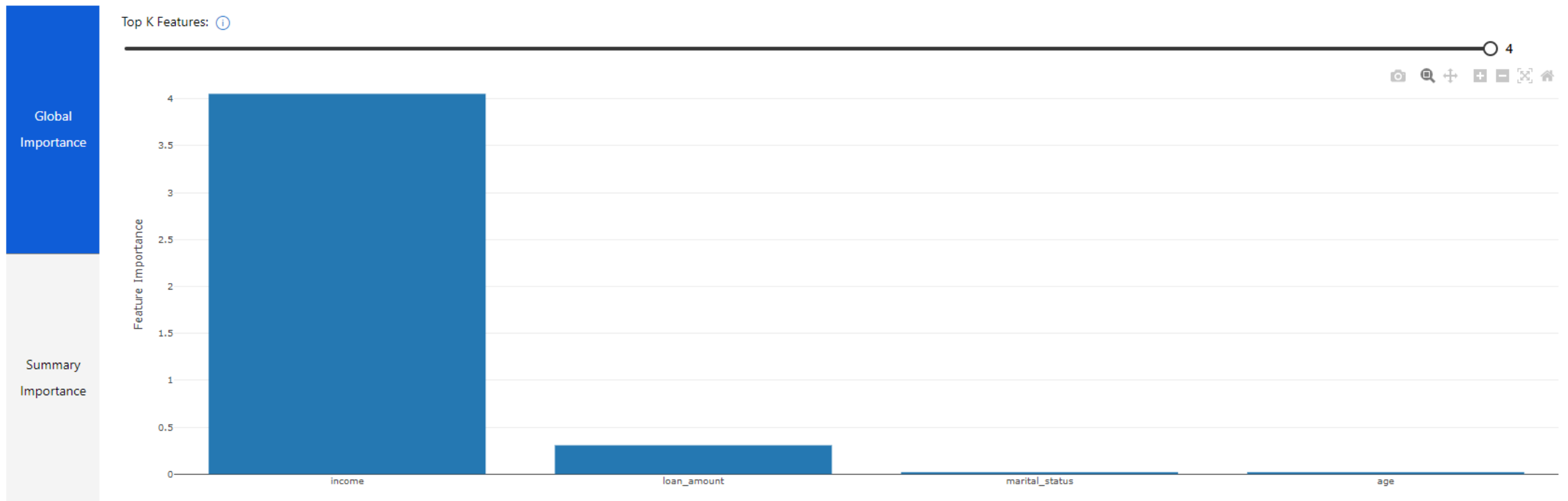
```
automl_config = AutoMLConfig(name='Automated ML Experiment',  
                             ...,  
                             model_explanability=True)
```

- View explanations:
 - Azure Machine Learning studio
 - **RunDetails** widget in Jupyter
 - SDK **ExplanationClient** class

```
best_run, fitted_model = automl_run.get_output()  
client = ExplanationClient.from_run(best_run)  
explanation = client.download_model_explanation()  
feature_importances = explanation.get_feature_importance_dict()
```

Visualizing Model Explanations

- View the **Explanations** tab for the run in Azure Machine Learning studio



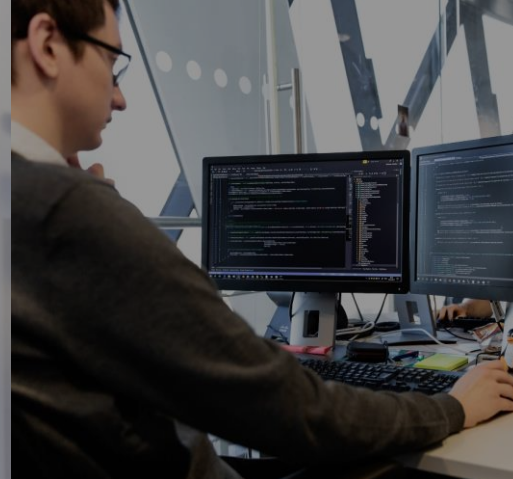
Lab 9A

Reviewing Automated Machine Learning
Explanations

<https://aka.ms/msl-dp100>

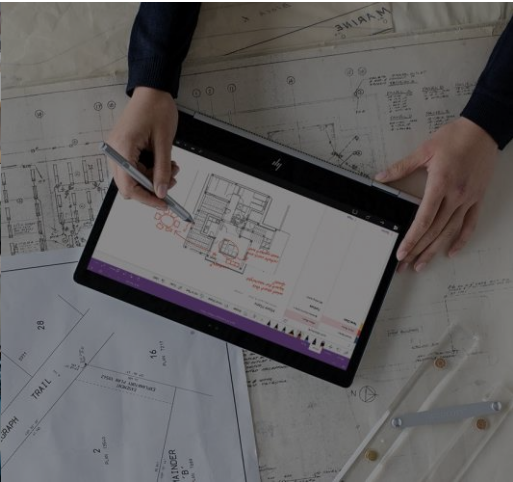
Lesson Review

Complete the Review Questions



Lesson 2

Using Model Explainers



Local Model Interpretation

- Use the **azureml-interpret** package
- Create an explainer:
 - **MimicExplainer** - global surrogate model that approximates your model
 - **TabularExplainer** - Invokes direct SHAP explainer based on model architecture
 - **PFIExplainer** - Permutation Feature Importance based on feature shuffling
- Get global or local feature explanations

```
from interpret.ext.blackbox import TabularExplainer

tab_explainer = TabularExplainer(model, X_train, features=features, classes=labels)
global_explanation = tab_explainer.explain_global(X_train)
```


Adding Interpretability to Training Experiments

- In the training script, import the **ExplanationClient** class
- Generate explanations and upload them to the run

```
explain_client = ExplanationClient.from_run(run)
explainer = MimicExplainer(model, X_train, LinearExplainableModel,
                           features=features, classes=labels)
explanation = explainer.explain_global(X_test)
explain_client.upload_model_explanation(explanation, comment='Model Explanation')
```

- Use **ExplanationClient** to download explanations

```
from azureml.contrib.interpret.explanation.explanation_client import ExplanationClient

client = ExplanationClient.from_run_id(workspace=ws,
                                       experiment_name=experiment.experiment_name,
                                       run_id=run.id)
explanation = client.download_model_explanation()
```

Interpretability During Inferencing

- Register a lightweight scoring explainer with the model

```
scoring_explainer = KernelScoringExplainer(explainer)
save(scoring_explainer, directory='dir', exist_ok=True)
Model.register(ws, model_name='model', model_path='dir/model.pkl')
Model.register(ws, model_name='explainer', model_path='dir/scoring_explainer.pkl')
```

- Use the model and the explainer in the service scoring script

```
def run(raw_data):
    data = json.loads(raw_data)['data']
    predictions = model.predict(data)
    local_importance_values = explainer.explain(data)
    return {"predictions":predictions.tolist(), "importance":local_importance_values}
```

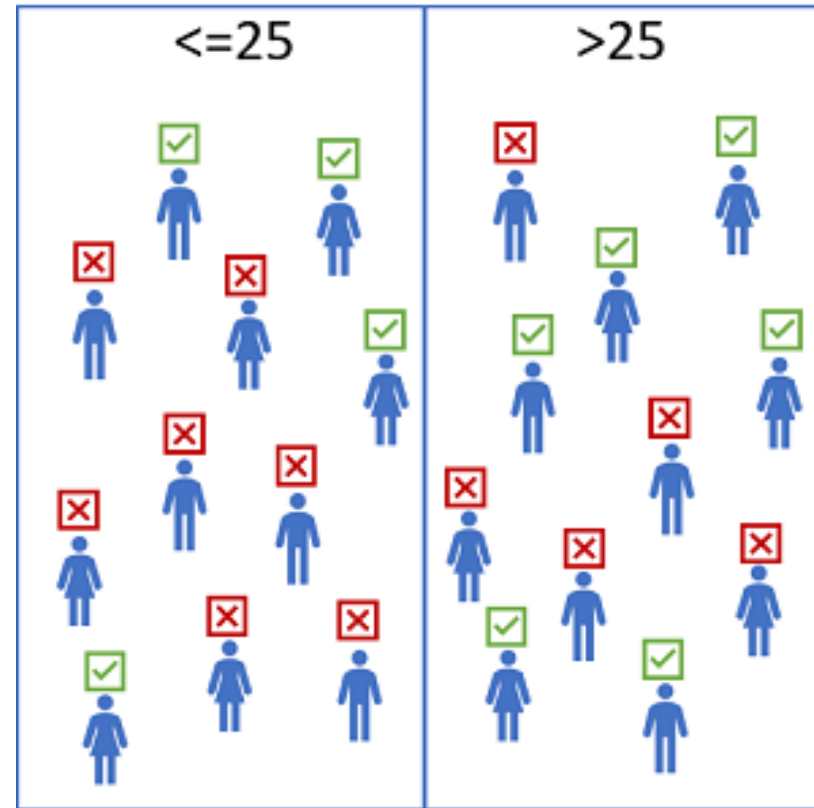
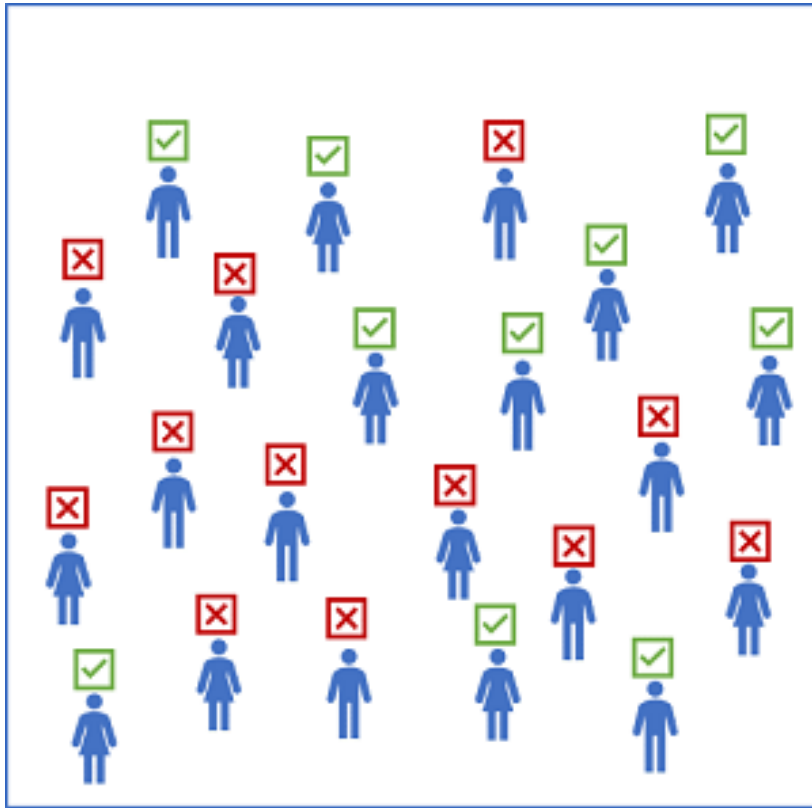
- Deploy a service with the model and explainer

```
service = Model.deploy(ws, 'classify_svc', [model, explainer], inf_config, dep_config)
```

Model fairness

- When we consider the concept of fairness concerning predictions made by machine learning models, it helps to be clear about what we mean by "fair".
- For example, suppose a classification model is used to predict the probability of successful loan repayment and therefore influences whether or not the loan is approved. The model will likely be trained using features that reflect the characteristics of the applicant, such as:
 - Age
 - Employment status
 - Income
 - Savings
 - Current debt

Example



Potential causes of disparity

- Data imbalance. Some groups may be overrepresented in the training data, or the data may be skewed so that cases within a specific group aren't representative of the overall population.
- Indirect correlation. The sensitive feature itself may not be predictive of the label, but there may be a hidden correlation between the sensitive feature and some other feature that influences the prediction. For example, there's likely a correlation between age and credit history, and there's likely a correlation between credit history and loan defaults. If the credit history feature is not included in the training data, the training algorithm may assign a predictive weight to age without accounting for credit history, which might make a difference to loan repayment probability.
- Societal biases. Subconscious biases in the data collection, preparation, or modeling process may have influenced feature selection or other aspects of model design.

Mitigating Bias

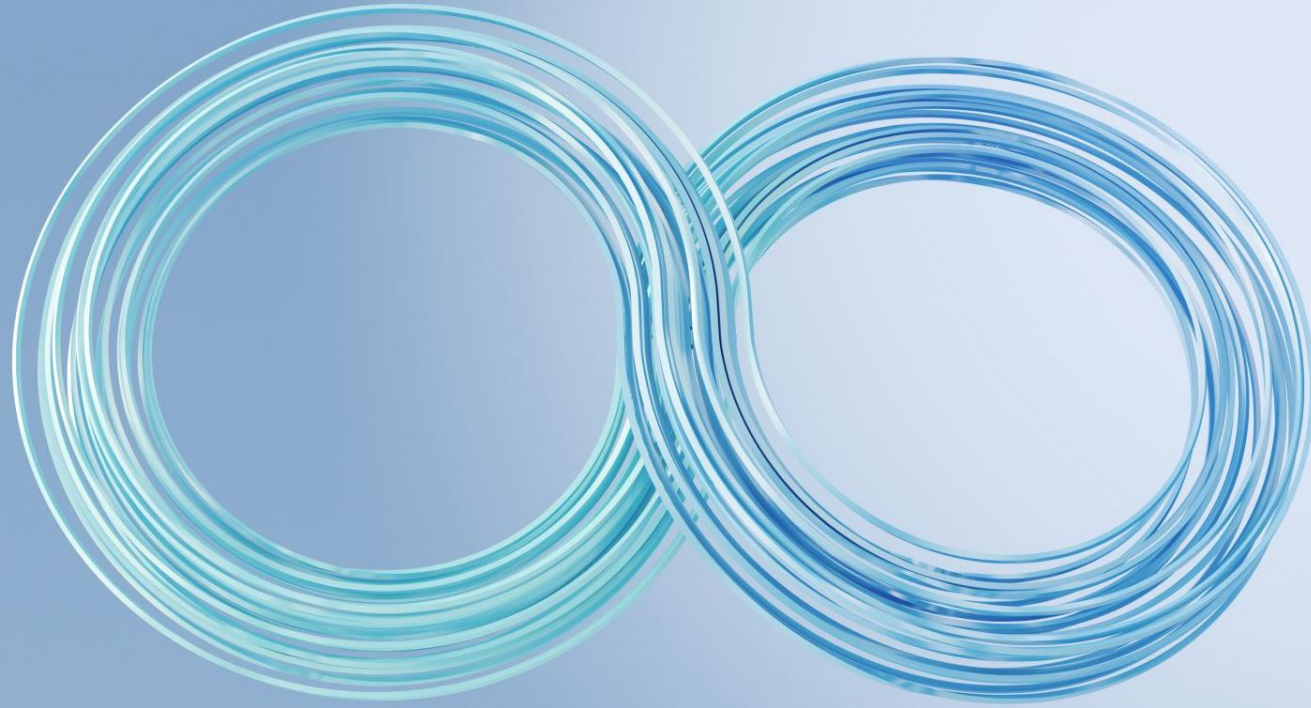
- Optimizing for fairness in a machine learning model is a sociotechnical challenge. In other words, it's not always something you can achieve purely by applying technical corrections to a training algorithm. However, there are some strategies you can adopt to mitigate bias, including:
- Balance training and validation data. You can apply over-sampling or under-sampling techniques to balance data and use stratified splitting algorithms to maintain representative proportions for training and validation.
- Perform extensive feature selection and engineering analysis. Make sure you fully explore the interconnected correlations in your data to try to differentiate features that are directly predictive from features that encapsulate more complex, nuanced relationships. You can use the model interpretability support in Azure Machine Learning to understand how individual features influence predictions.
- Evaluate models for disparity based on significant features. You can't easily address the bias in a model if you can't quantify it.
- Trade-off overall predictive performance for the lower disparity in predictive performance between sensitive feature groups. A model that is 99.5% accurate with comparable performance across all groups is often more desirable than a model that is 99.9% accurate but discriminates against a particular subset of cases.

Mitigation algorithms and parity constraints

The mitigation support in Fairlearn is based on the use of algorithms to create alternative models that apply parity constraints to produce comparable metrics across sensitive feature groups. Fairlearn supports the following mitigation techniques.

Technique	Description	Model type support
Exponentiated Gradient	A reduction technique that applies a cost-minimization approach to learning the optimal trade-off of overall predictive performance and fairness disparity	Binary classification and regression
Grid Search	A simplified version of the Exponentiated Gradient algorithm that works efficiently with small numbers of constraints	Binary classification and regression
Threshold Optimizer	A post-processing technique that applies a constraint to an existing classifier, transforming the prediction as appropriate	Binary classification

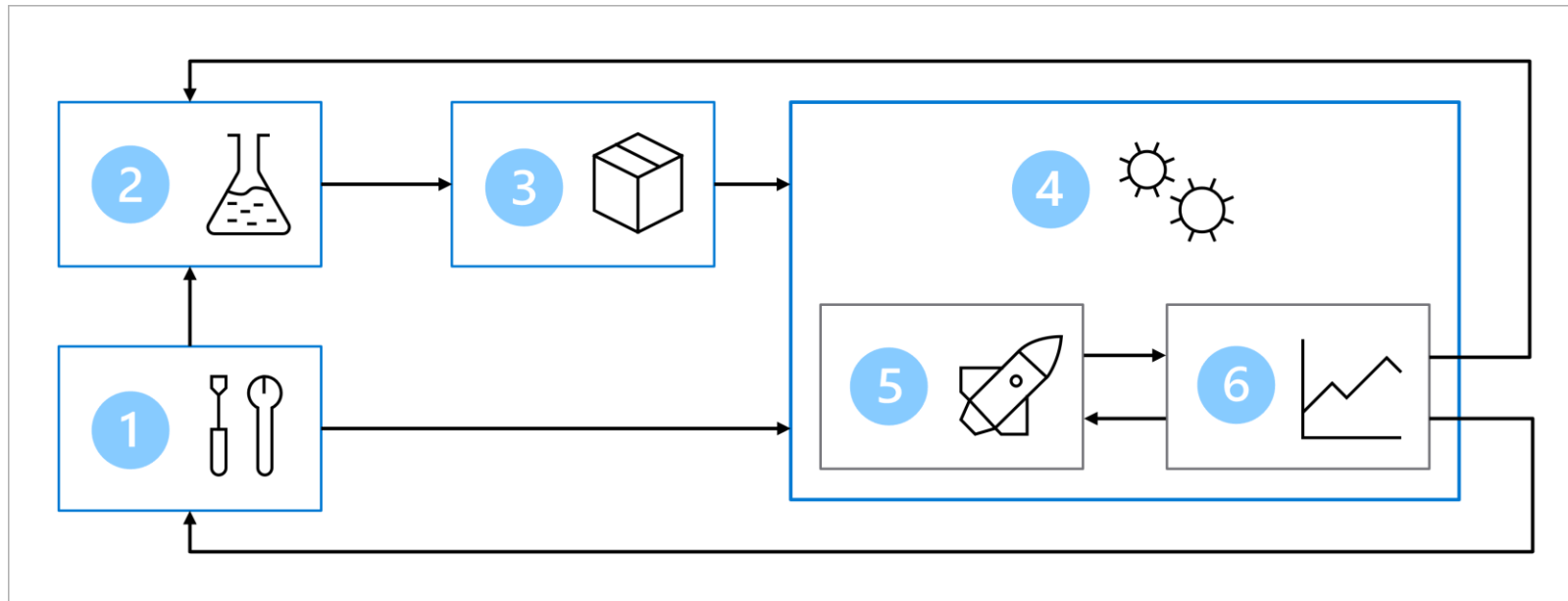




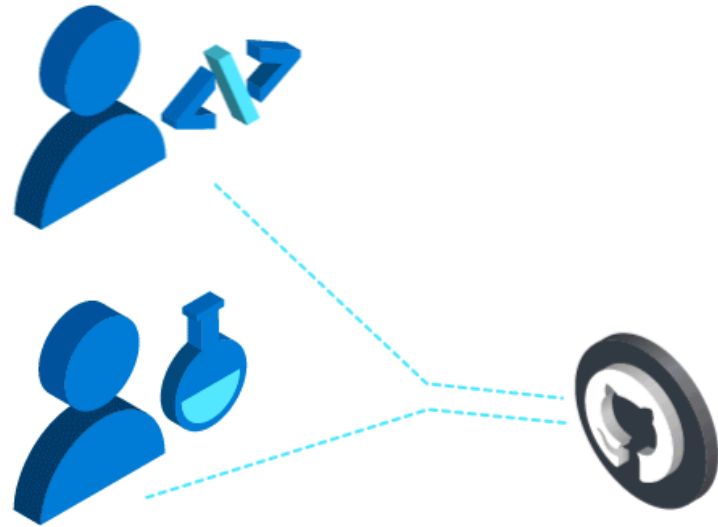
CI/CD Pipeline with GitHub Actions <-> Azure ML

MLOps Architecture

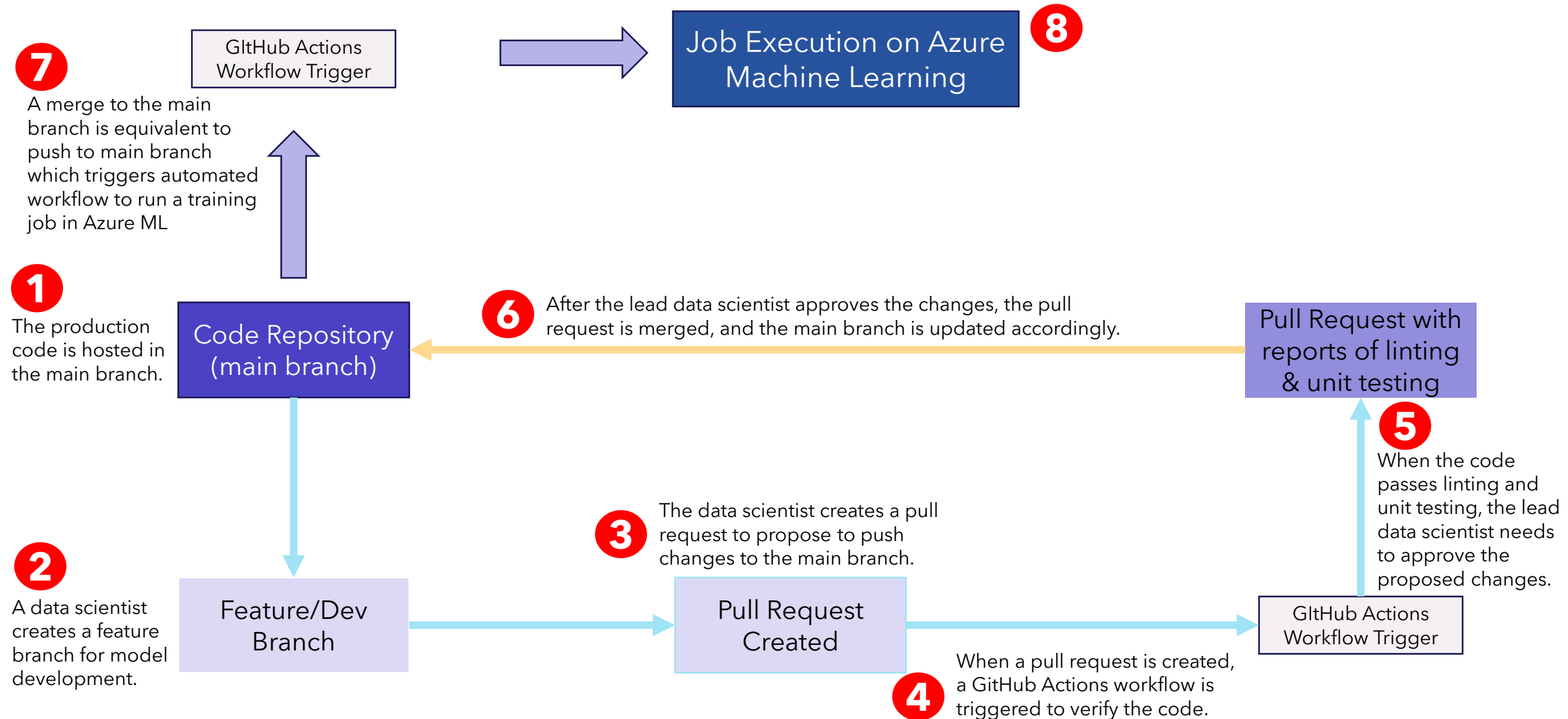
- 1. Setup:** Create all necessary Azure resources for the solution.
- 2. Model development (inner loop):** Explore and process the data to train and evaluate the model.
- 3. Continuous integration:** Package and register the model.
- 4. Model deployment (outer loop):** Deploy the model.
- 5. Continuous deployment:** Test the model and promote to production environment.
- 6. Monitoring:** Monitor model and endpoint performance.



MLOPs with Azure & GitHub Actions



CI pipeline



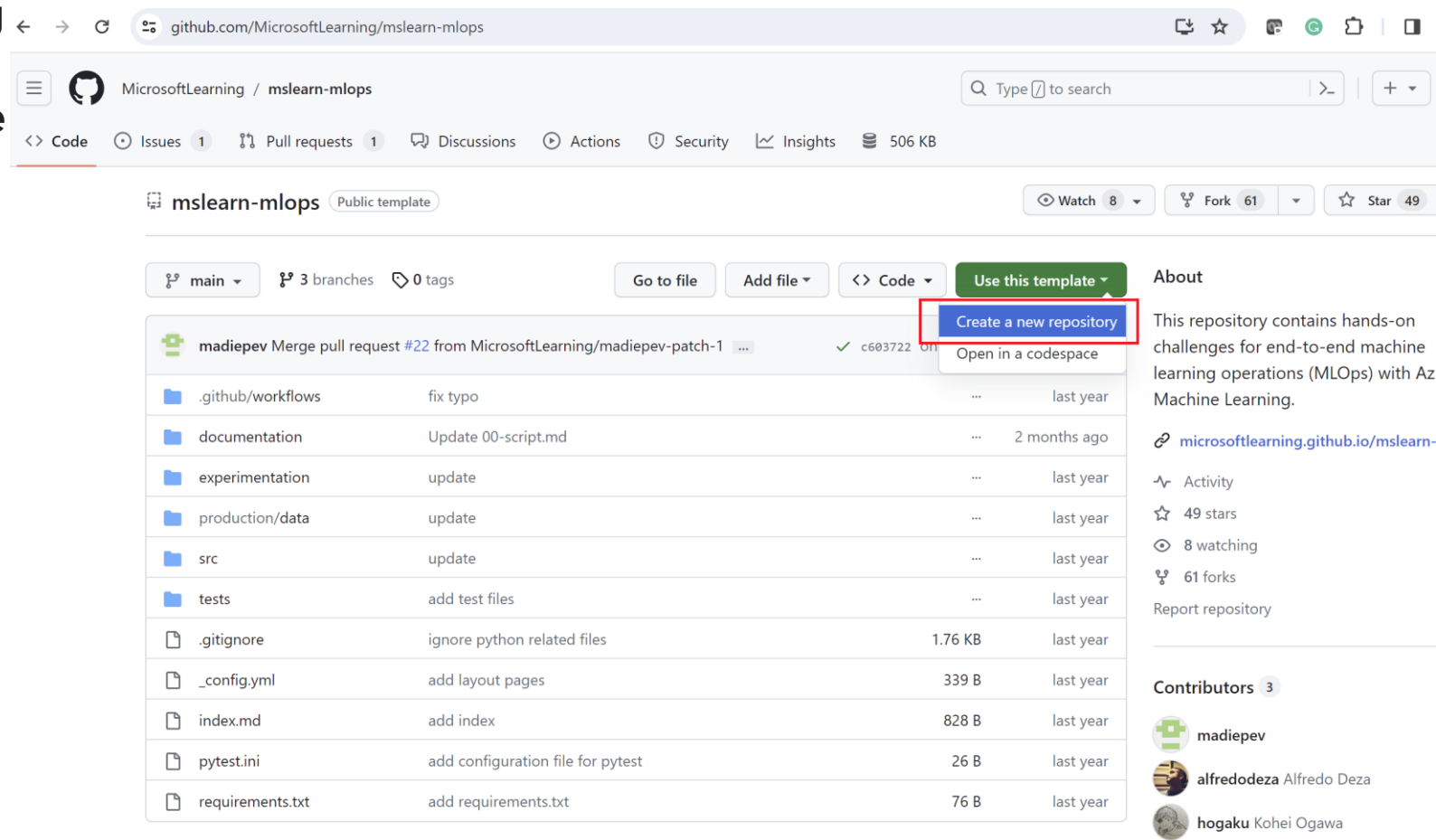
CI Pipeline Milestone 1

Objective:

1. Setting up GitHub Repository for ML project
2. Converting notebook to script
3. Testing local code execution

Clone the mlops repo

- Create a new public repo by navigating to <https://github.com/MicrosoftLearning/mslearn-mlops> and selecting the **Use this template** button to create your own repo.
- In the **experimentation** folder, you'll find a Jupyter notebook that trains a classification model. The data used by the notebook is in the **experimentation/data** folder and contains a CSV file.
- In the **src/model** folder you'll find a train.py script which already includes code converted from part of the notebook. It's up to you to complete it.



Structure your repo

- **.cloud:** contains cloud-specific code like templates to create an Azure Machine Learning workspace.
- **.ad/.github:** contains Azure DevOps or GitHub artifacts like YAML pipelines to automate workflows.
- **src:** contains any code (Python or R scripts) used for machine learning workloads like preprocessing data or model training.
- **docs:** contains any Markdown files or other documentation used to describe the project.
- **pipelines:** contains Azure Machine Learning pipelines definitions.
- **tests:** contains unit and integration tests used to detect bugs and issues in your code.
- **notebooks:** contains Jupyter notebooks, mostly used for experimentation.

Convert a notebook to a script

Scripts are ideal for testing and automation in your production environment. To create a production-ready script, you'll need to:

- Remove nonessential code.
- Refactor your code into functions.
- Test your script in the terminal.

Remove all nonessential code

- The main benefit of using notebooks is being able to quickly explore your data. For example, you can use **`print()`** and **`df.describe()`** statements to explore your data and variables. When you create a script that will be used for automation, you want to avoid including code written for exploratory purposes.
- The first thing you therefore need to do to convert your code to production code is to remove the nonessential code. Especially when you'll run the code regularly, you want to avoid executing anything nonessential to reduce cost and compute time.

Complete the code train.py

- The `split_data` function is already included in the main function. You only need to add the function itself with the required inputs and outputs underneath the comment TO DO: add function to split data.
- Add logging so that every time you run the script, all parameters and metrics are tracked. Use the autologging feature of MLflow to also ensure the necessary model files are stored with the job run to easily deploy the model in the future.

Refactor your code into functions

Python

```
def main(csv_file):
    # read data
    df = get_data(csv_file)

    # split data
    X_train, X_test, y_train, y_test = split_data(df)

    # function that reads the data
    def get_data(path):
        df = pd.read_csv(path)

        return df

    # function that splits the data
    def split_data(df):
        X, y = df[['Pregnancies', 'PlasmaGlucose', 'DiastolicBloodPressure', 'TricepsThickness',
                    'SerumInsulin', 'BMI', 'DiabetesPedigree', 'Age']].values, df['Diabetic'].values

        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=0)

        return X_train, X_test, y_train, y_test
```

Test your script

- Before using scripts in production environments, for example by integrating them with automation pipelines, you'll want to test whether the scripts work as expected.

```
python train.py
```

CI Pipeline

Milestone 2

Objective:

1. Setting up Azure ML resources
2. Setting up Azure CLI
3. Testing Azure ML job from Local Machine

Azure CLI

- Download: <https://learn.microsoft.com/en-us/cli/azure/install-azure-cli-windows?tabs=azure-cli>
- Setup
 - az login
 - az account set -s "<YOUR_SUBSCRIPTION_NAME_OR_ID>"
 - az extension add -n ml -y
 - az ml -h

Workspace creation with CLI

```
az ml workspace create --resource-group scb --name anshu-aml
```

Workspace creation with CLI

```
az ml workspace create --resource-group  
                        [--application-insights]  
                        [--container-registry]  
                        [--description]  
                        [--display-name]  
                        [--file]  
                        [--image-build-compute]  
                        [--location]  
                        [--managed-network]  
                        [--name]  
                        [--no-wait]  
                        [--primary-user-assigned-identity]  
                        [--public-network-access]  
                        [--set]  
                        [--tags]  
                        [--update-dependent-resources]
```


Workspace creation with CLI

```
az ml workspace create --resource-group  
                        [--application-insights]  
                        [--container-registry]  
                        [--description]  
                        [--display-name]  
                        [--file]  
                        [--image-build-compute]  
                        [--location]  
                        [--managed-network]  
                        [--name]  
                        [--no-wait]  
                        [--primary-user-assigned-identity]  
                        [--public-network-access]  
                        [--set]  
                        [--tags]  
                        [--update-dependent-resources]
```

Creating compute with CLI

- Create a compute cluster, *consider creating compute cluster for this lab.*
- `az ml compute create --name aml-compute --size STANDARD_DS11_V2 --max-instances 2 --type AmlCompute --workspace anshu-aml --resource-group scb`
- Create a compute instance (do not create compute instance for this activity)
- `az ml compute create --name "testdev-vm" --size STANDARD_DS11_V2 --type ComputeInstance workspace anshu-aml --resource-group scb`

Creating environment (optional)

- Not needed for this lab

```
az ml environment create --file my_env.yml --resource-group my-resource-group --workspace-name my-workspace
```

- Options: <https://learn.microsoft.com/en-us/cli/azure/ml/environment?view=azure-cli-latest#az-ml-environment-create>

Create data asset

- Use the CLI (v2) to create a registered data asset with the following configuration:
 - **Name:** *diabetes-dev-folder*
 - **Path:** The **data** folder in the **experimentation** folder which contains the CSV file to train the model. The path should point to the folder, not to the specific file.

```
az ml data create --name diabetes-dev-folder --version 1  
--path "D:\Repos\devb\SCBmlops\experimentation\data" --  
resource-group scb --workspace-name anshu-aml
```

Create data asset

- `az ml data create --name my-data --version 1 --path ./my-data.csv --resource-group my-resource-group --workspace-name my-workspace`

```
az ml data create [--datastore]
                  [--description]
                  [--file]
                  [--name]
                  [--no-wait]
                  [--path]
                  [--registry-name]
                  [--resource-group]
                  [--set]
                  [--skip-validation]
                  [--type {mltable, uri_file, uri_folder}]
                  [--version]
                  [--workspace-name]
```

Get dataset URI

Complete the job.yml file to define the Azure Machine Learning job to run the train.py script, with the registered data asset as input. **"Replace the Dataset URI in path option for training data"**
Use the CLI (v2) to run the job.

The screenshot shows the Azure ML Studio interface. The left sidebar contains navigation options: All workspaces, Home, Model catalog, Authoring (Notebooks, Automated ML, Designer, Prompt flow), Assets (Data, Jobs, Components, Pipelines, Environments, Models, Endpoints), and Manage (Compute, Monitoring). The main panel displays the details for the 'diabetes-dev-folder' dataset, version 1 (latest). The 'Attributes' section lists: Type (Folder (uri_folder)), Named asset URI (azureml:diabetes-dev-folder:1), Created by (Anshu Pandey), Current version (1), Latest version (1), Created time (Nov 3, 2023 1:54 PM), and Modified time (Nov 3, 2023 1:54 PM). The 'Data sources' section shows the Datastore as 'workspaceblobstore' and the Relative path as 'LocalUpload/a5b05a17f213e8e12e54948a8e3d1dfa/data/'. The 'Datastore URI' is highlighted with a red box and contains the text: 'azureml://subscriptions/84a5808b-5549-459a-98f2-f102e84fa1bb/resourcegroups/scb/workspaces/anshu-aml/datastores/workspaceblobstore/paths/LocalUpload/a5b05a17f213e8e12e54948a8e3d1dfa/data/'.

azureml://subscriptions/84a5808b-5549-459a-98f2-f102e84fa1bb/resourcegroups/scb/workspaces/anshu-aml/datastores/workspaceblobstore/paths/LocalUpload/a5b05a17f213e8e12e54948a8e3d1dfa/data/

Running a job

Complete the job.yml file to define the Azure Machine Learning job to run the train.py script, with the registered data asset as input. **"Replace the Dataset URI in path option for training data"**
Use the CLI (v2) to run the job.

```
$schema: https://azuremlschemas.azureedge.net/latest/commandJob.schema.json
code: model
command: >-
  python train.py
  --training_data ${inputs.training_data}
  --reg_rate ${inputs.reg_rate}
inputs:
  training_data:
    type: uri_folder
    path: azureml://subscriptions/84a5808b-5549-459a-98f2-f102e84fa1bb/resourcegroups/SCB/workspaces/aml-
\anshu/datastores/workspaceblobstore/paths/LocalUpload/386681db2f7cd59e8c96f5ee80b212db/data/
  reg_rate: 0.09
environment: azureml:AzureML-sklearn-0.24-ubuntu18.04-py37-cpu@latest
compute: aml-compute
experiment_name: diabetes-mlops
description: A demo classification model
```

Run command: **az ml job create --file job.yml --web**

CI Pipeline

Milestone 3

Objective:

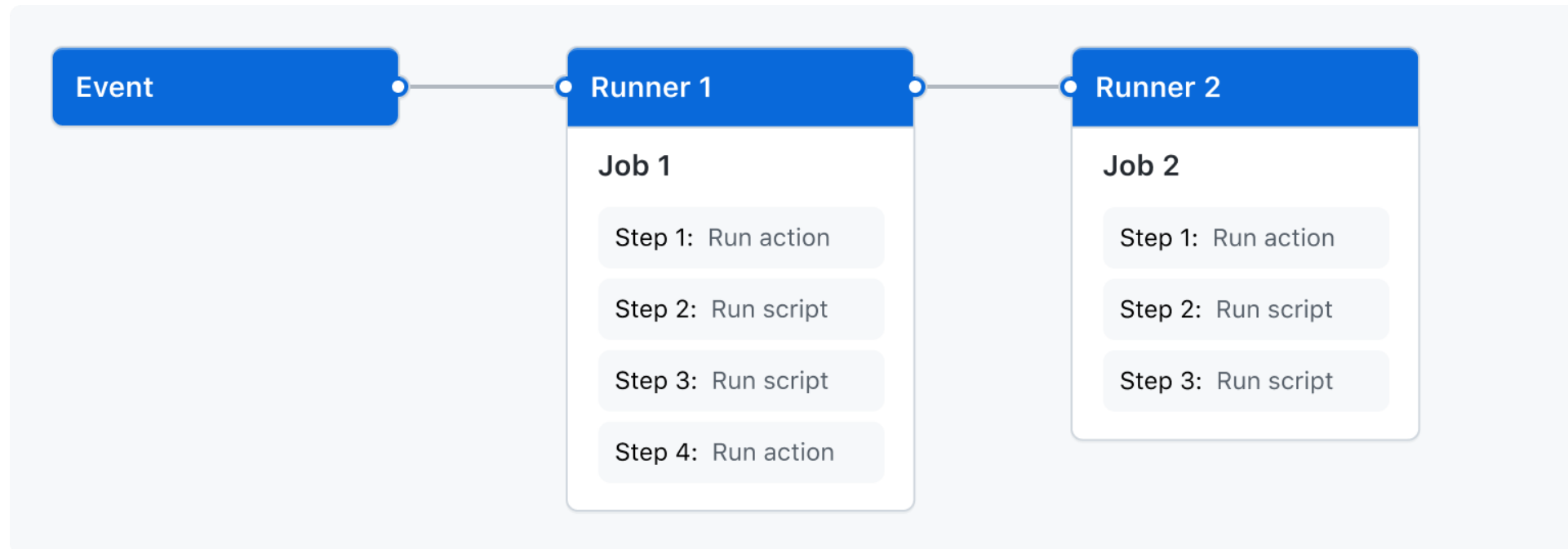
1. Creating service principal for GitHub Repo
2. Testing AzureML job trigger on push to main branch

GitHub Actions

- GitHub Actions is a continuous integration and continuous delivery (CI/CD) platform that allows you to automate your build, test, and deployment pipeline. You can create workflows that build and test every pull request to your repository, or deploy merged pull requests to production.
- GitHub Actions goes beyond just DevOps and lets you run workflows when other events happen in your repository. For example, you can run a workflow to automatically add the appropriate labels whenever someone creates a new issue in your repository.
- GitHub provides Linux, Windows, and macOS virtual machines to run your workflows, or you can host your own self-hosted runners in your own data center or cloud infrastructure.
- Detailed Tutorial: <https://docs.github.com/en/actions/guides>

GitHub Actions

- You can configure a GitHub Actions workflow to be triggered when an event occurs in your repository, such as a pull request being opened or an issue being created.
- Your workflow contains one or more jobs which can run in sequential order or in parallel.
- Each job will run inside its own virtual machine runner, or inside a container, and has one or more steps that either run a script that you define or run an action, which is a reusable extension that can simplify your workflow.



Trigger the Azure Machine Learning job with GitHub Actions

- Steps
 - Create a service principal and use it to create a GitHub secret for authentication.
 - Run the Azure Machine Learning job with GitHub Actions.
- note: GitHub is authenticated to use your Azure Machine Learning workspace with a service principal. The service principal is only allowed to submit jobs that use a compute cluster, not a compute instance.

Trigger the Azure Machine Learning job with GitHub Actions

- In the .github/workflows folder, you'll find the 02-manual-trigger.yml file. The file defines a GitHub Action which can be manually triggered. The workflow checks out the repo onto the runner, installs the Azure Machine Learning extension for the CLI (v2), and logs in to Azure using the AZURE_CREDENTIALS secret.
- Create a service principal, using the Cloud Shell in the Azure portal, which has contributor access to your resource group.
- Save the output, you'll also need it for later challenges. Update the <service-principal-name> (should be unique), <subscription-id>, and <your-resource-group-name> before using the following command:
- Code

```
az ad sp create-for-rbac --name "<service-principal-name>" --role contributor \  
    --scopes /subscriptions/<subscription-id>/resourceGroups/<your-resource-group-name> \  
    --sdk-auth
```
- Create a GitHub secret in your repository. Name it AZURE_CREDENTIALS and copy and paste the output of the service principal to the Value field of the secret.

Creating service principal

```
az ad sp create-for-rbac --name "github-actions" --role  
contributor --scopes /subscriptions/84a5808b-5549-459a-  
98f2-f102e84fa1bb/resourceGroups/scb --sdk-auth
```

Template:

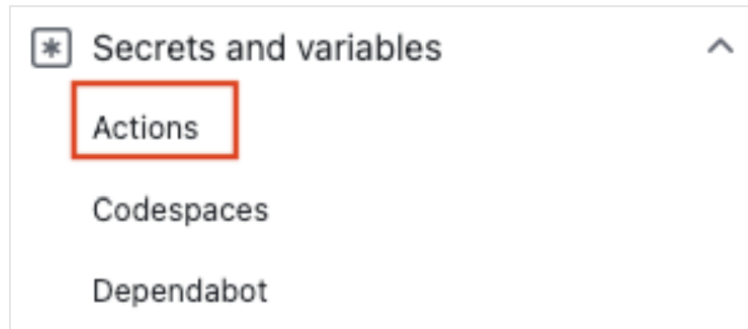
```
az ad sp create-for-rbac --name "<service-principal-  
name>" --role contributor \ --scopes  
/subscriptions/<subscription-id>/resourceGroups/<your-  
resource-group-name> \ --sdk-auth
```

GitHub Action secret format

```
{  
  "clientId": "your-client-id",  
  "clientSecret": "your-client-secret",  
  "subscriptionId": "your-subscription-id",  
  "tenantId": "your-tenant-id",  
  "activeDirectoryEndpointUrl": "https://login.microsoftonline.com",  
  "resourceManagerEndpointUrl": "https://management.azure.com/",  
  "activeDirectoryGraphResourceId": "https://graph.windows.net/",  
  "sqlManagementEndpointUrl": "https://management.core.windows.net:8443/",  
  "galleryEndpointUrl": "https://gallery.azure.com/",  
  "managementEndpointUrl": "https://management.core.windows.net/"  
}
```

Client Secret

1. In [GitHub](#), go to your repository.
2. Go to **Settings** in the navigation menu.
3. Select **Security > Secrets and variables > Actions**.



4. Select **New repository secret**.
5. Paste the entire JSON output from the Azure CLI command into the secret's value field. Give the secret the name `AZURE_CREDENTIALS`.
6. Select **Add secret**.

Prepare workflow:

Edit the 02-manual-trigger.yml workflow to trigger the Azure Machine Learning job.
Prepare job.yml, add details such as command for execution (last 2 lines), Push changes to github repo

```
name: Manually trigger an Azure Machine Learning job
on:
  workflow_dispatch:
jobs:
  train:
    runs-on: ubuntu-latest
    steps:
      - name: Check out repo
        uses: actions/checkout@main
      - name: Install az ml extension
        run: az extension add -n ml -y
      - name: Azure login
        uses: azure/login@v1
        with:
          creds: ${{secrets.AZURE_CREDENTIALS}}
      - name: Run Azure ML Job
        run: az ml job create --file src/job.yml --workspace-name anshu-aml --resource-group scb
```


Trigger the Azure Machine Learning job with GitHub Actions

Navigate to GitHub repo and manually run the workflow to trigger the azure machine learning job. Check azure machine learning job execution over AML Studio. You can observe that the job is created by service principal.

The screenshot displays the GitHub Actions interface for a repository named 'anshupandey / SCBmllops'. The 'Actions' tab is selected, showing a workflow titled 'Manually trigger an Azure Machine Learning job' (02-manual-trigger-job.yml). The workflow has 11 runs. A red box labeled '1' highlights the 'Actions' tab in the top navigation bar. Another red box labeled '2' highlights the workflow name in the left sidebar. A third red box labeled '3' highlights the 'Run workflow' button in the top right of the workflow details. A fourth red box labeled '4' highlights the 'Run workflow' button in the dropdown menu that appears after clicking the 'Run workflow' button. The interface also shows a search bar, a 'Filter workflow runs' input, and a 'Beta' badge.

More documentation: <https://docs.github.com/en/actions/using-workflows/manually-running-a-workflow>

Trigger the Azure Machine Learning job with GitHub Actions

Validate the workflow run.

Also check azure machine learning job execution over AML Studio. You can observe that the job is created by service principal.

<> Code

Issues

Pull requests

Actions

Projects

Wiki

Security

Insights

Settings

← Manually trigger an Azure Machine Learning job

✓ Manually trigger an Azure Machine Learning job #11

Re-run all jobs

...

Summary

Jobs

train

Run details

Usage

Workflow file

train

succeeded now in 55s

Search logs

Settings

> ✓ Set up job

4s

> ✓ Check out repo

1s

> ✓ Install az ml extension

27s

> ✓ Azure login

6s

> ✓ Run Azure ML Job

14s

> ✓ Post Check out repo

1s

> ✓ Complete job

0s

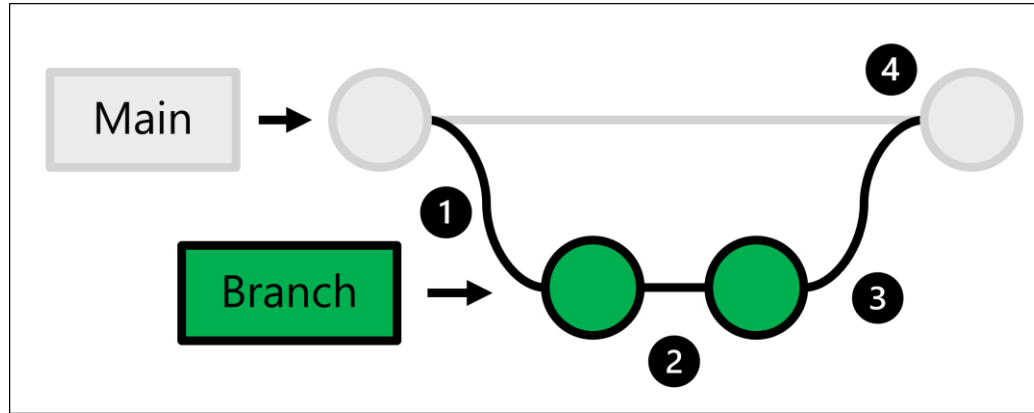
CI Pipeline

Milestone 4

Objective:

1. Setting up feature branch, creating branch protection rule
2. Setting up testing and linting process flow with GitHub Actions
3. Building complete CI Pipeline

Trunk-based development



Triggering a workflow by pushing directly to the repo is **not** considered a best practice.

The production code is hosted in the main branch. Whenever someone wants to make a change:

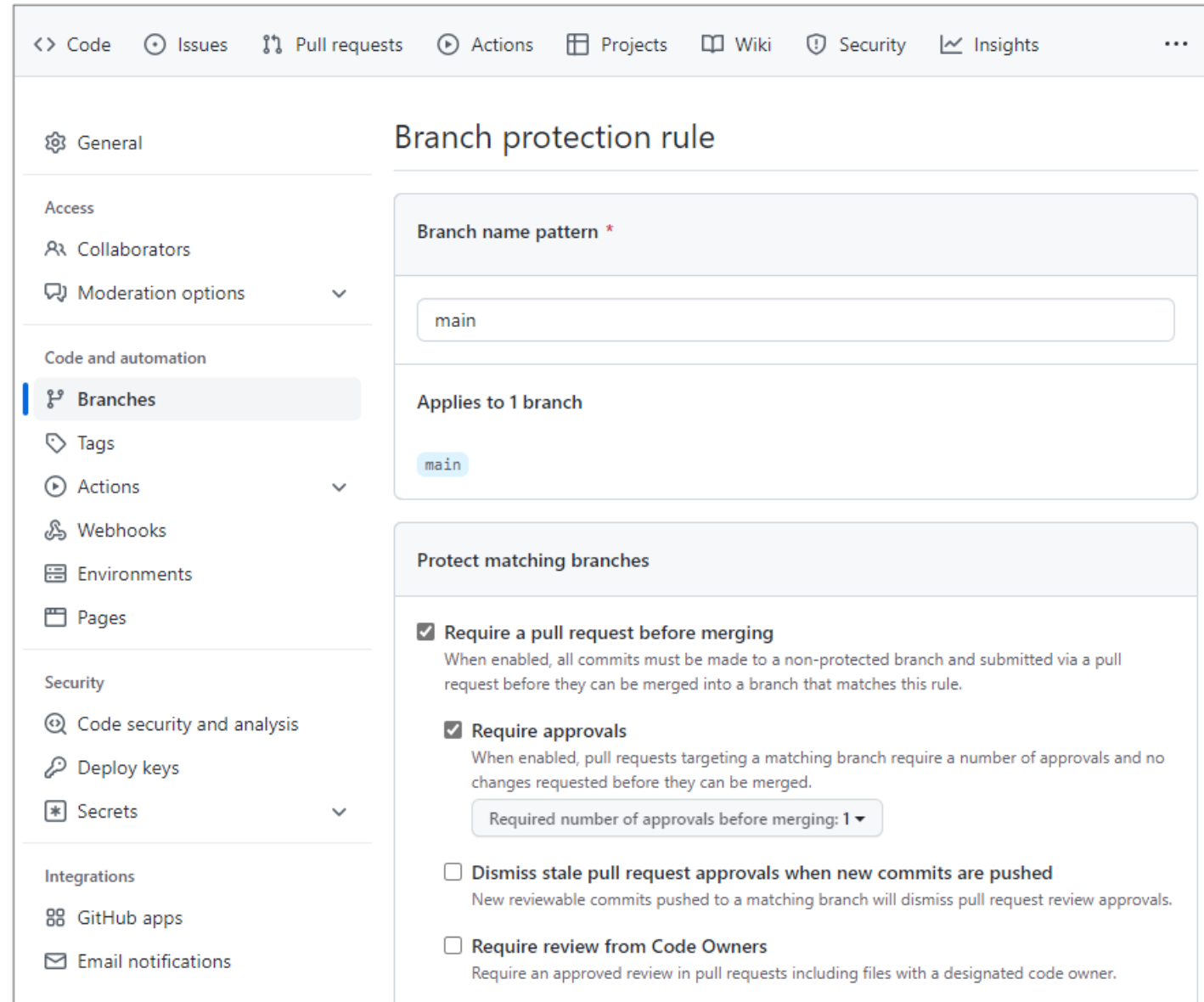
1. You create a full copy of the production code by creating a branch.
2. In the branch you created, you make any changes and test them.
3. Once the changes in your branch are ready, you can ask for someone to review the changes.
4. If the changes are approved, you merge the branch you created with the main repo and the production code will be updated to reflect your changes.

Create a branch protection rule

To protect your code, you want to **block any direct pushes to the main branch**. Blocking direct pushes means that no one will be allowed to directly push any code changes to the main branch. Instead, changes to the main branch can be made by merging pull requests.

To protect the main branch, enable a **branch protection rule** in GitHub:

1. Navigate to the **Settings** tab in your repo.
2. In the **Settings** tab, under **Code and automation**, select **Branches**.
3. Select **Add rule**.
4. Enter main under **Branch name pattern**.
5. Enable **Require a pull request before merging** and **Require approvals**.
6. Save your changes.



The screenshot shows the GitHub interface for configuring a branch protection rule. The top navigation bar includes links for Code, Issues, Pull requests, Actions, Projects, Wiki, Security, and Insights. The left sidebar shows the 'Settings' tab with a list of categories: General, Access, Moderation options, Code and automation (selected), Branches (active), Tags, Actions, Webhooks, Environments, Pages, Security, Code security and analysis, Deploy keys, Secrets, Integrations, GitHub apps, and Email notifications. The main content area is titled 'Branch protection rule' and contains the following sections:

- General**:
 - Access**: Collaborators, Moderation options.
 - Code and automation**: Branches (selected), Tags, Actions, Webhooks, Environments, Pages.
 - Security**: Code security and analysis, Deploy keys, Secrets.
 - Integrations**: GitHub apps, Email notifications.
- Branch name pattern ***: A text input field containing 'main'.
- Applies to 1 branch**: A list of branches with 'main' selected.
- Protect matching branches**:
 - ☒ **Require a pull request before merging**: When enabled, all commits must be made to a non-protected branch and submitted via a pull request before they can be merged into a branch that matches this rule.
 - ☒ **Require approvals**: When enabled, pull requests targeting a matching branch require a number of approvals and no changes requested before they can be merged. A dropdown menu shows 'Required number of approvals before merging: 1'.
 - ☐ **Dismiss stale pull request approvals when new commits are pushed**: New reviewable commits pushed to a matching branch will dismiss pull request review approvals.
 - ☐ **Require review from Code Owners**: Require an approved review in pull requests including files with a designated code owner.

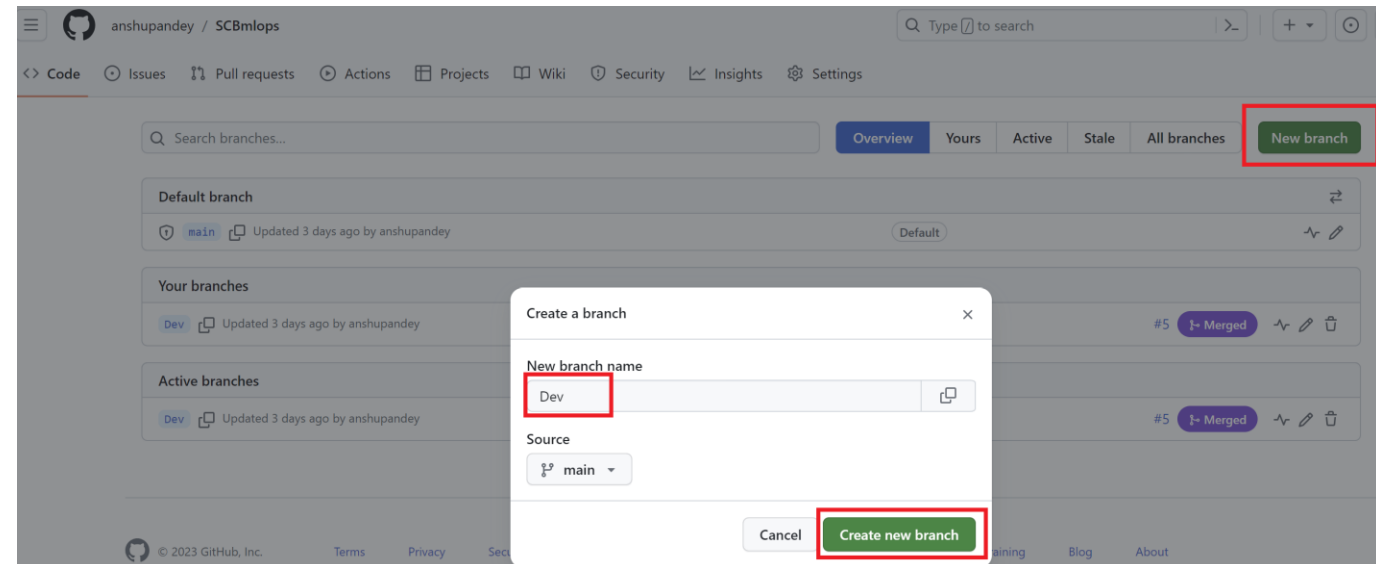
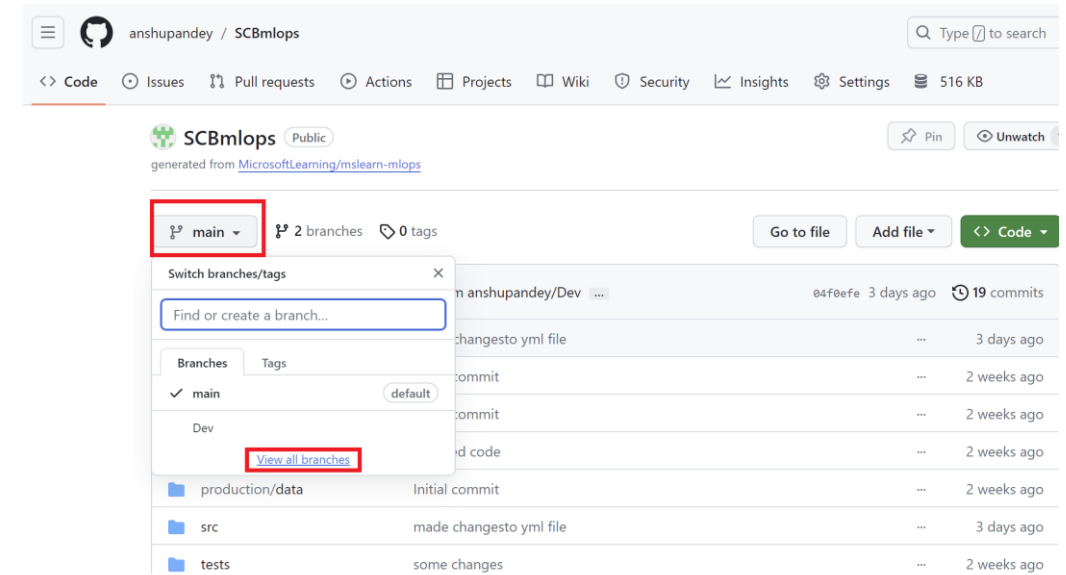
Create a branch to update the code.

Whenever you want to edit the code, you'll have to create a branch and work in there. Once you want to make your changes final, you can create a pull request to merge the feature branch with the main branch.

More reads:

<https://learn.microsoft.com/en-us/training/modules/source-control-for-machine-learning-projects/3-work-azure-repos-github-repos>

Clone the “Dev” branch locally in VS Code.



Trigger a GitHub Actions workflow

Finally, you may want to use the creation of pull requests as a trigger for GitHub Actions workflows. For example, whenever someone makes changes to the code, you'll want to run some code quality checks.

Only when the edited code has passed the quality checks and someone has verified the proposed changes, do you want to actually merge the pull request.

To trigger a GitHub Actions workflow, you can use `on: [pull_request]`. When you use this trigger, your workflow will run whenever the pull request is created.

If you want a workflow to run whenever a pull request is merged, you'll need to use another trigger. Merging a pull request is essentially a push to the main branch. So, to trigger a workflow to run when a pull request is merged, use the following trigger in the GitHub Actions workflow:

yaml

```
on:
  push:
    branches:
      - main
```

Unit Testing for dev code

To check whether the code works as expected, you can create **unit tests**. To easily test specific parts of your code, your scripts should contain **functions**. You can test functions in your scripts by creating test files. A popular tool to test Python code is **Pytest**.

Where linting verifies how you wrote the code, unit tests check how your code works. Units refer to the code you create. Unit testing is therefore also known as code testing.

As a best practice, your code should exist mostly out of functions. Whether you've created functions to prepare data, or to train a model. You can apply unit testing to, for example:

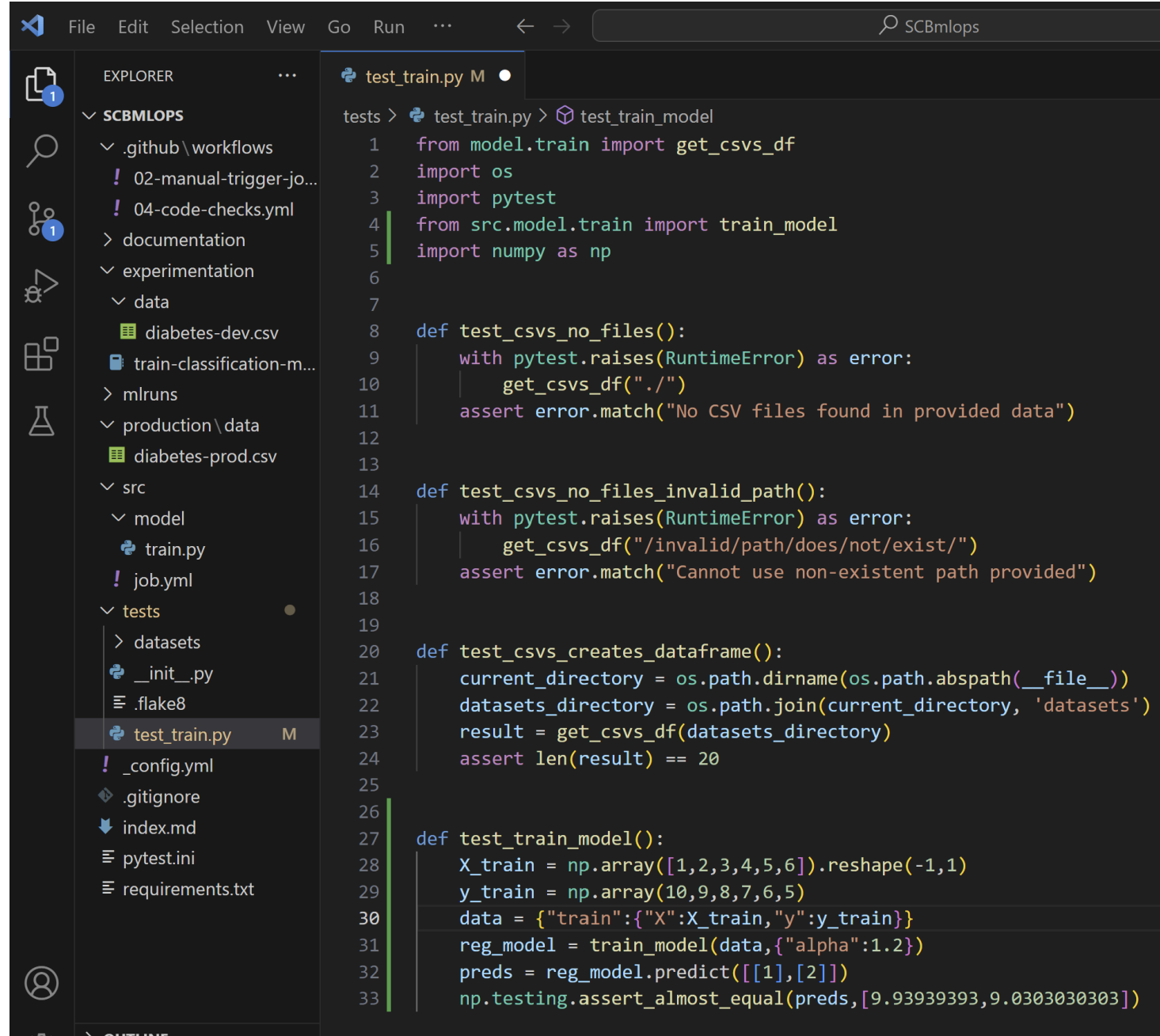
- Check that column names are right.
- Check the prediction level of model on new datasets.
- Check the distribution of prediction levels.

When you work with Python, you can use Pytest and Numpy (which uses the Pytest framework) to test your code.

Install pytest

```
pip install pytest
```

Complete test_main.py



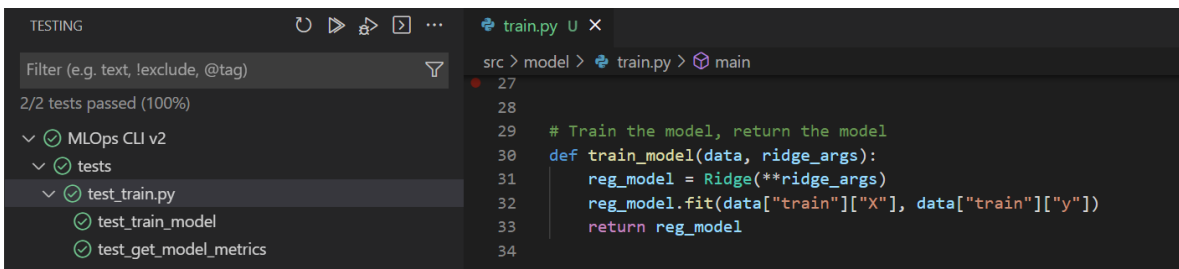
```
test_train.py M
tests > test_train.py > test_train_model
1 from model.train import get_csvs_df
2 import os
3 import pytest
4 from src.model.train import train_model
5 import numpy as np
6
7
8 def test_csvs_no_files():
9     with pytest.raises(RuntimeError) as error:
10         get_csvs_df("./")
11     assert error.match("No CSV files found in provided data")
12
13
14 def test_csvs_no_files_invalid_path():
15     with pytest.raises(RuntimeError) as error:
16         get_csvs_df("/invalid/path/does/not/exist/")
17     assert error.match("Cannot use non-existent path provided")
18
19
20 def test_csvs_creates_dataframe():
21     current_directory = os.path.dirname(os.path.abspath(__file__))
22     datasets_directory = os.path.join(current_directory, 'datasets')
23     result = get_csvs_df(datasets_directory)
24     assert len(result) == 20
25
26
27 def test_train_model():
28     X_train = np.array([1,2,3,4,5,6]).reshape(-1,1)
29     y_train = np.array(10,9,8,7,6,5)
30     data = {"train":{"X":X_train,"y":y_train}}
31     reg_model = train_model(data,{"alpha":1.2})
32     preds = reg_model.predict([[1],[2]])
33     np.testing.assert_almost_equal(preds,[9.93939393,9.03030303])
```


Unit Testing for dev code

To test your code in Visual Studio Code using the UI:

- Install all necessary libraries to run the training script.
- Ensure pytest is installed and enabled within Visual Studio Code.
- Install the Python extension for Visual Studio Code.
- Select the train.py script you want to test.
- Select the Testing tab from the left menu.
- Configure Python testing by selecting pytest and setting the test directory to your tests/ folder.
- Run all tests by selecting the play button and review the results.

<https://learn.microsoft.com/en-us/training/modules/source-control-for-machine-learning-projects/5-verify-your-code-locally>



```
from model.train import get_csvs_df
import os
import pytest
from src.model.train import train_model
import numpy as np
```

```
def test_csvs_no_files():
    with pytest.raises(RuntimeError) as error:
        get_csvs_df("./")
    assert error.match("No CSV files found in provided data")
```

```
def test_csvs_no_files_invalid_path():
    with pytest.raises(RuntimeError) as error:
        get_csvs_df("/invalid/path/does/not/exist/")
    assert error.match("Cannot use non-existent path provided")
```

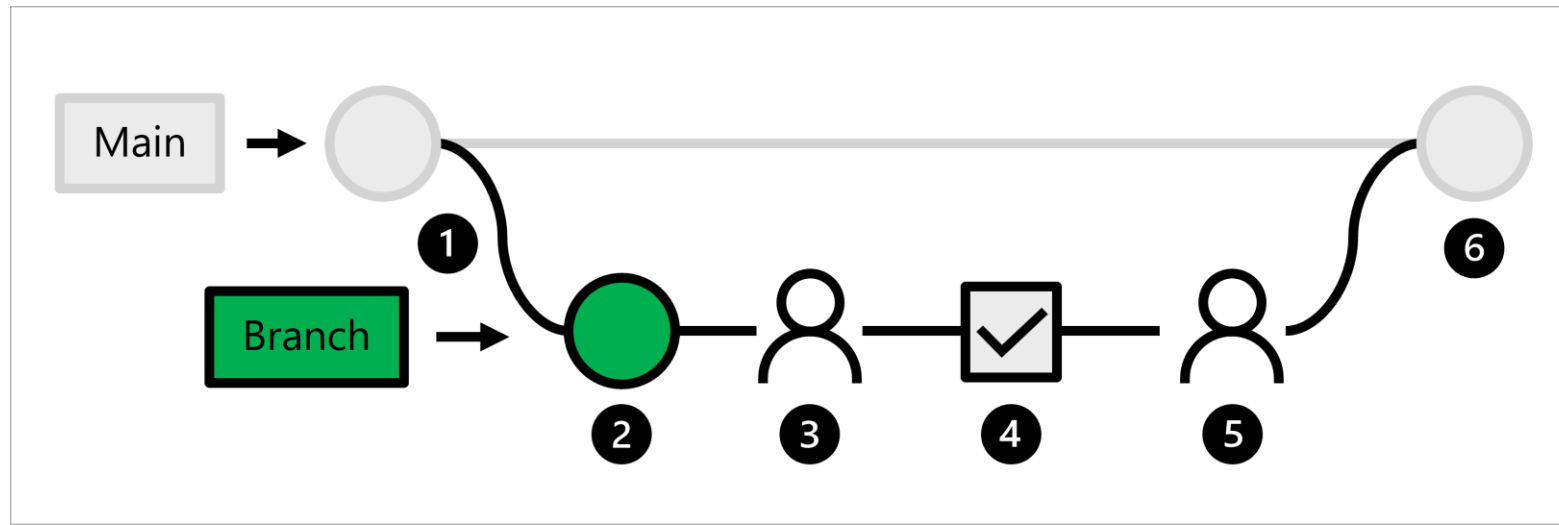
```
def test_csvs_creates_dataframe():
    current_directory = os.path.dirname(os.path.abspath(__file__))
    datasets_directory = os.path.join(current_directory, 'datasets')
    result = get_csvs_df(datasets_directory)
    assert len(result) == 20
```

```
def test_train_model():
    X_train = np.array([1,2,3,4,5,6]).reshape(-1,1)
    y_train = np.array(10,9,8,7,6,5)
    data = {"train":{"X":X_train,"y":y_train}}
    reg_model = train_model(data,{"alpha":1.2})
    preds = reg_model.predict([[1],[2]])
    np.testing.assert_almost_equal(preds,[9.93939393,9.03030303])
```

Linting and testing

1. The production code is hosted in the main branch.
2. A data scientist creates a feature branch for model development.
3. The data scientist creates a pull request to propose to push changes to the main branch.
4. When a pull request is created, a GitHub Actions workflow is triggered to verify the code.
5. When the code passes linting and unit testing, the lead data scientist needs to approve the proposed changes.
6. After the lead data scientist approves the changes, the pull request is merged, and the main branch is updated accordingly.

As a machine learning engineer, you'll need to create a GitHub Actions workflow that verifies the code by running a linter and unit tests whenever a pull request is created.



Verify your code locally

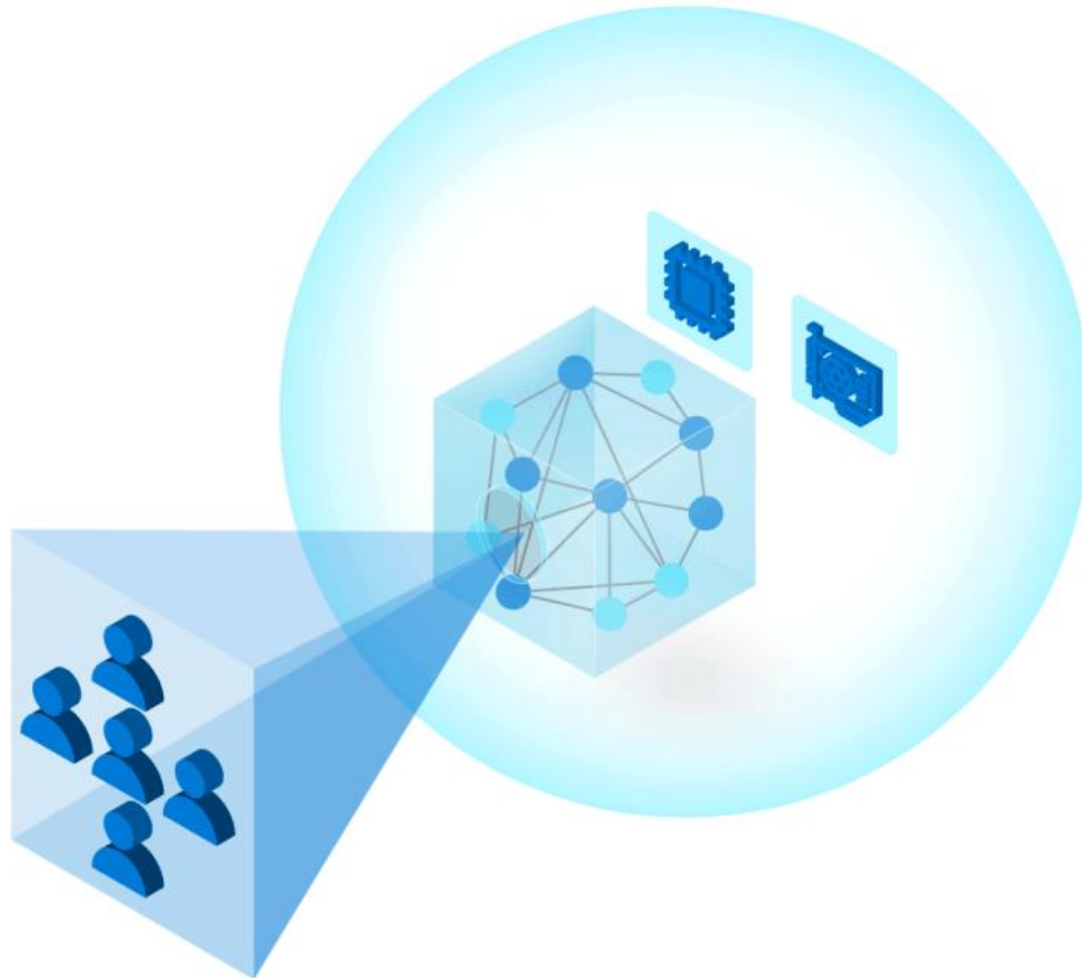
- Whenever you change any code in your machine learning project, you want to verify the code and model quality.
- During continuous integration, you create and verify assets for your application. As a data scientist, you'll probably focus on creating scripts used for data preparation and model training. The machine learning engineer uses the scripts later in pipelines to automate these processes.
- To verify your scripts, there are two common tasks:
 1. Linting: Check for programmatic or stylistic errors in Python or R scripts.
 2. Unit testing: Check the performance of the content of the scripts.

<https://learn.microsoft.com/en-us/training/modules/source-control-for-machine-learning-projects/5-verify-your-code-locally>

Modify the workflow and test

- Modify the workflow to trigger it when there is push to main branch.
- Make changes to code, in the dev branch, push to dev branch
- Create a pull request to merge the code to the main branch
- Merge the pull request, merge to main branch is equivalent to push to main branch, it should automatically trigger the workflow to execute the training processing.

Continuous Deployment



Deployment config

- \$schema:
<https://azuremlschemas.azureedge.net/latest/managedOnlineDeployment.schema.json>
- name: mlflow-deployment
- endpoint_name: churn-endpoint
- model:
 - name: mlflow-sklearn-model
 - version: 1
 - local_path: model
 - model_format: mlflow
- instance_type: Standard_F2s_v2
- instance_count: 1

Deploy model with endpoint

- `az ml online-endpoint create --name diabetes-mlflow -f ./mslearn-aml-cli/Allfiles/Labs/05/mlflow-endpoint/create-endpoint.yml`
- `az ml online-deployment create --name mlflow-deployment --endpoint diabetes-mlflow -f ./mslearn-aml-cli/Allfiles/Labs/05/mlflow-endpoint/mlflow-deployment.yaml --all-traffic`
- `az ml online-endpoint update --name churn-endpoint --traffic "blue=0 green=100"`
- `az ml online-endpoint delete --name churn-endpoint --yes --no-wait`

Resources

- <https://learn.microsoft.com/en-in/training/paths/build-first-machine-operations-workflow/>
- <https://microsoftlearning.github.io/mslearn-mlops/>
- <https://learn.microsoft.com/en-us/training/modules/deploy-azure-machine-learning-model-managed-endpoint-cli-v2/>
- <https://learn.microsoft.com/en-us/azure/machine-learning/how-to-deploy-online-endpoints?view=azureml-api-2&tabs=azure-cli#understand-the-scoring-script>