

Introduction

Welcome to the EDS MLOps training. In this training you will learn the fundamentals of MLOps with a hands on example. The training consists of two parts:

1. a self paced study to satisfy the prerequisites.
2. a one day workshop to implement a Machine Learning solution.

Please take some days to complete the prerequisites and install software if you do not have those installed yet.

In the self paced section you will learn

- Git
- Python
- Working with VSCode
- Basics of DevOps
- The differences between DevOps and MLOps

In the workshop you will:

- Get hand-on experience in setting up a DevOps pipeline.
- Productionalise a proof of concept ML model, including development and deployment
- Monitor and improve the ML model.
- Detect drift and retrain the model.

The training is set up in Azure, but the focus will be on the concepts and benefits for working with a MLOps environment. Learning Azure specific code is avoided as much as possible.

Structure

To be fully prepared for the workshop you will need to study the prerequisites, set up the development environment and solve a basic assignment. A successful outcome of this assignment is key to participate in the workshop, it is the entry ticket. This is done to ensure that everything is setup correctly and we don't lose time during the workshop by setting up everything and solving individual problems.

The pre-requisites consist of 4 steps. If you have already set up conda with VSCode you study the prerequisites and skip to part 4.

1. **Study the prerequisites** [01-prerequisites.md](#)
2. **Install the required software** [02-installation.md](#)
3. **Setting up the development environment** [03-setting-up.md](#)
4. **Do assignment 0 to check if everything works** [04-assignment-0.md](#)

Please take some days to install software, because approval through OneIT can take some time.

01-prerequisites.md

Prerequisites

This is a living document, if you have any feedback or better learning sources: please contact Kyllian (869361) or Bram (869362).

Please study the tools you do not know, if you already have experience in a certain tool, just skip that section.

Git

Git is a distributed version control system. It is the most commonly used system to version control source code. It is essential to productionalise software. Because you have a record of what has been done and you can revert to specific versions. Git makes collaboration easier, allowing multiple people to work on code.

- Sources to visit and practise
 - ☐ Read the first three chapters of the [git book](#) from the official git website.
 - ☐ Optional also read 7.6 and 7.7 which are useful to correct commits
 - ☐ Practise git with an [interactive tool](#). Do the first 8 lessons all the way up to reverting changes. Stop when you encounter cherry picking. Type levels and change the levels to remote. Do all 8 levels of git remotes and the first 4 levels of advanced git remotes. This includes push arguments, but not push arguments expanded. [Click here for the lesson overview](#).
- Now you should have a basic understanding of git including the concepts: commits, local and remote branches and the commands: clone, fetch, pull, merge, push, rebase, reset, revert.

Installation

For details or help see this [detailed installation guide](#).

Python

Installation

Installation of Python on Cognizant assets is done through OneIT. Please make a request to install Anaconda. For details or help see this [detailed installation guide](#). As a IDE we are using Visual Studio Code. Please also install vscode by following the [installation guide](#).

Courses

There are many Python courses to follow. Here we listed some suggested courses. Feel free to create your own learning track based on the tasks below.

- ☐ [Python for data science beginner - through My Learning Studio](#)

- ☐ [Python for data science intermediate - through My Learning Studio](#)
- ☐ [Python for Beginners: Learn Python Programming \(Python 3\) - through Cognizant Learn - code: 666914](#)
- ☐ [Data Analysis with Pandas and Python - through Cognizant Learn - code: 932344](#)
- ☐ [The Python Bible Everything You Need to Program in Python - through Cognizant Learn - code: 903378](#)

Tasks

Essentially you should be able to perform the following tasks

- ☐ Create a conda environment based on a environment.yml file
- ☐ Launch a local jupyter lab environment to experiment with data.
- ☐ Perform basic data analysis with [Pandas](#)
 - ☐ Understand how to merge datasets (join)
 - ☐ Analyse a dataset with categorical variables (get occurrences)
 - ☐ Plot some data with Pandas
- ☐ Train a basic model with [scikit-learn](#)
 - ☐ Understand factorizing and normalising data.

General DevOps concepts

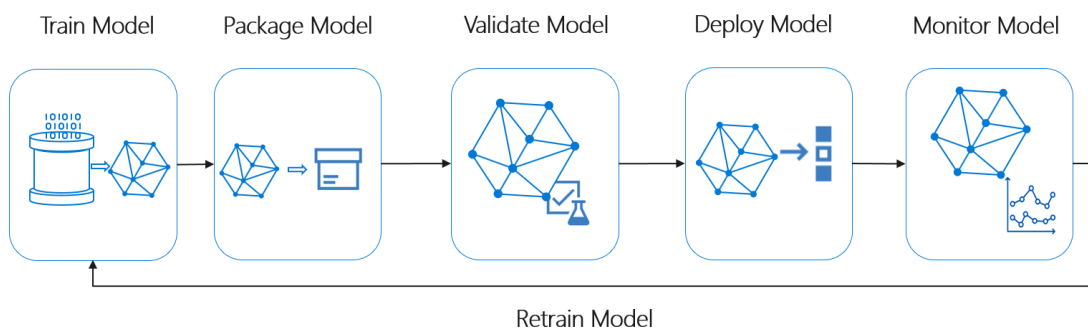
- ☐ [Read this short overview on DevOps by Gitlab](#)
- ☐ [Read this article by Microsoft on DevOps](#)

General MLOps concepts

- ☐ [Read this Google cloud background article on MLOps](#)

MLOps with Azure ML

This workshop will help you to understand how to build a Continuous Integration and Continuous Delivery pipeline for an ML/AI project. We will be using the Azure DevOps Project for build and release/deployment pipelines along with Azure ML services for model retraining pipeline, model management and operationalisation.



Architecture and Features

- ☐ Please take a close look at this Architecture Reference: [Machine learning operationalisation \(MLOps\) for Python models using Azure Machine Learning](#).

This reference architecture is the basis for this MLOps training. It shows how to implement continuous integration (CI), continuous delivery (CD), and retraining pipeline for an AI application using Azure DevOps and Azure Machine Learning. Please make sure that you understand the components of the pipeline. Pay particular attention to the difference between DevOps pipelines and ML pipelines, those are completely different things.

Other useful tools

REST APIs

A basic understanding of REST APIs is useful as the model will be deployed as a REST API.

- ☐ [Read this article on REST APIs](#)
- ☐ [Install ARC to send requests to an API \(right click Troubleshoot Compatibility\)](#)
- ☐ [Tryout the ARC on jsonplaceholder](#)

Containers and Docker

The DevOps pipeline itself uses Docker containers and the model itself is also deployed as a container. Therefore, basic understanding of containers and Docker is useful.

- ☐ [Read about the differences between Containers and VMs in this article by Backblaze](#)
- ☐ [Read this 'What is Docker?' article](#)

Next step

Go to [Setting up your development environment](#)

References (not needed to study)

- [Azure Machine Learning \(Azure ML\) Service Workspace](#)
- [Azure ML CLI](#)
- [Azure ML Samples](#)
- [Azure ML Python SDK Quickstart](#)
- [Azure DevOps](#)
- [Machine learning operationalisation \(MLOps\) for Python models using Azure Machine Learning](#)
- [MLOps with Azure ML in Python](#)

02-installation.md

Installation of required software

Visual Studio Enterprise Subscription

To get full access to the cognizant azure devops cloud environment you will need a Visual Studio Enterprise Subscription. Request this through OneIT.

Visual Studio Enterprise Edition

Version : 2019 | Manufacturer : Microsoft | License : Per User | Profile : MSDN
Details : this is MSDN Subscription

Cognizant
Owned



As a Business Justification you can write: "Required for MLOps Workshops"

They will contact you to install Visual Studio, when they do, please let them know that you only need the subscription. Not the full software installation.

Python

Installation of Python on Cognizant assets is done through OneIT. Please make a request to install Anaconda. **Please take into account that this can take some time**

If you have problems installing follow this [be.cognizant guide](#). Before connecting to the VPN make sure to log in onto OneCognizant and go to the installation page of Anaconda.

[Back to introduction](#)

Git

There are two ways to install git:

- request via OneIT, the official way to go, if you cannot find it, please search for git and its latest version number: e.g. at time of writing git 2.28.
- through Anaconda: easiest but only local installation within the conda environment (not recommended)

Request via OneIT

Go to OneIT portal in OneCognizant

Through Anaconda

When Anaconda is installed open the Anaconda prompt and type:

```
conda install -c anaconda git
```

This installs git on your machine.

[Back to introduction](#)

IDE: Visual Studio Code

To install vscode you do not need to raise a request, you can follow the following steps.

1. Download the software from the link <https://code.visualstudio.com/download> (please download system installer 64 bit file)
2. Once download is complete please move the files to C Drive Development_Avecto (If you do not find such a folder, please create one)
3. Right click on the setup file and select troubleshoot compatibility
4. Choose try recommended settings and select 'Test the program' only once and give it a minute
5. The installation wizard will open up.

[Back to introduction](#)

Once everything is installed go to [Setting up your development environment](#)

Setting up your development environment

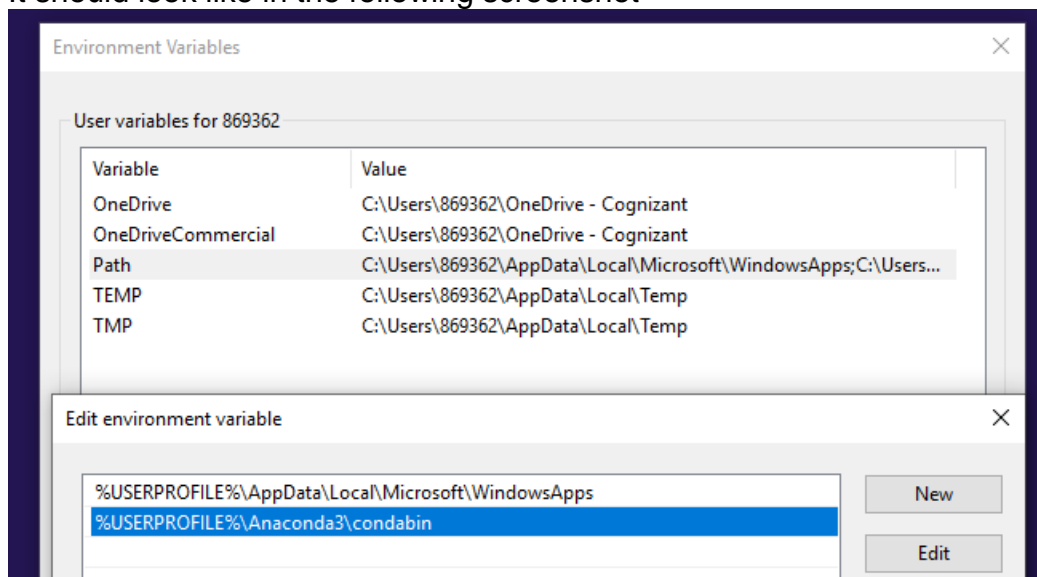
This document explains how to work with vscode, conda, the command line and jupyter notebooks.

Adding Conda to your PATH environment variable

In order to use Conda commands from your terminal within VSCode, you have to add the executables to your PATH variable. This means that when you type in a command, e.g. `conda env list` the command line interpreter will look in those locations to find the conda executable.

If you have already checked “Add Anaconda to my PATH environment variable” during Anaconda installation, skip step 1.

1. If Anaconda is installed for the current use only, add %USERPROFILE% into the environment variable PATH (the user one).
 - Type in Path in the Windows menu
 - Click on “Edit the system environment variables”
 - In this pane select “Environment Variables”
 - Select the existing **User** variable Path
 - Click Edit
 - Click new and copy %USERPROFILE%\Anaconda3\condabin in this field
 - It should look like in the following screenshot



2. Open a new Powershell terminal, e.g in VSCode (see below), and run the following command once to initialise conda

```
conda init
```

3. Alternatively it is also possible to use the git bash terminal. Bash is used on Linux and Unix (macOS) and therefore the default in most cloud based application. Once git is installed you can use `Ctrl + Shift + P` and type "select default shell", select bash. This can also be found in the terminal drop down menu. If you start a new terminal than it should automatically be bash. In order to work with conda from bash, you need to associate conda within bash. Use the following command:

```
echo ". $HOME/Anaconda3/etc/profile.d/conda.sh" >> ~/.bashrc
```

Close the terminal and start a new one. Type:

```
conda init
```

As a final step: close and start a new terminal.

For more details see: <https://stackoverflow.com/questions/44597662/conda-command-is-not-recognized-on-windows-10>

Viewing the README and assignments

The README and assignment files are written in markdown. You can open a preview of these files within VSCode.

- Open the folder within VSCode.
 - Under File select open folder
- Open README.md by double clicking
 - Single click temporarily opens a file while a double click opens it permanently
- Open a preview by selecting the preview button or by pressing `Ctrl + K V`.

– Preview button:



Terminal in VSCode

The easiest way to work with the terminal is to use the terminal in VSCode. You can display the terminal by clicking `View -> Terminal` or by pressing `Ctrl + ``

You can check your conda installation by running:

```
conda --version
# conda 4.8.4
```

Installing extensions in VSCode

Besides being open source, one of the other main reasons for its popularity are very useful extensions in VSCode. We can recommend to at least install:

- Python (ms.python.python) (required)
- Git Graph (mhutchie.git-graph) (optional, but very handy)

Managing conda environments

Within the repository is a conda environment file. This file specifies the requirements, all the packages needed for the code to run.

1. Open the `MLOpsTraining-Assignments` folder within VSCode.
2. Open a terminal within VSCode.
 - Verify that the working directory is within the root of the `MLOpsTraining-Assignments` folder.
 - Type `pwd` (powershell) or `dir` (command prompt) in the terminal to print the working directory
 - Type `ls` to list the files within the directory

3. To create a conda environment for this workshop type:

```
conda env create -f environment.yml
```

This will create a conda environment called `MLOpsTraining` from [environment.yml](#).

4. Activate the environment with:

```
conda activate MLOpsTraining
```

5. This environment does not include a Jupyter notebook server. Please go ahead and install by running:

```
conda install -c conda-forge jupyterlab
```

You have now installed all necessary packages.

Start assignment-0

Read the instructions in [assignment-0](#)

04-assignment-0.md

Assignment 0: Setting up Visual Studio Code, Linting and Formatting

This workshop uses Visual Studio Code. If you are familiar with these contents please skip the explanation and run the test script.

Running python in VSCode

If not done yet check: [installation details](#) and [setting up the development environment](#).

- Open the folder in VSCode
- Open the file `hello_world.py` in the folder `assignment-0`
 - The python interpreter should start loading
- To run the code in the correct environment you have to specify the correct Conda environment.
 - Then on the statusbar in the bottom of the screen you can select a Python interpreter. Choose the one that contains `MLOPsTraining`.
 - Another way to change the interpreter is by pressing `Ctrl + Shift + P` and type `python select interpreter`. Select the right environment that contains `MLOPsTraining`.
- Run the file `assignment-0/hello_world.py` by pressing the play button (a green triangle) in the top right of the screen. You should see an output in a terminal. If no errors appear, you set up your environment correctly.
- To run the file from the command line itself you also have to activate the conda environment from the terminal.

```
conda activate MLOPsTraining
```

- Now you can run the file by typing

```
python assignment-0\hello_world.py
```

Running a jupyter notebook

- Create a second terminal within VSCode, by pressing the + sign, and run:

```
conda activate MLOPsTraining  
jupyter lab
```

This terminal keeps running in the background, so that you can use another terminal for git commands or running python code.

- In the jupyter notebook open `notebooks\hello_world.ipynb`

- Another option is to open this notebook within VSCode
 - Make sure you select the right kernel to run your notebook. The `hello_world.ipynb` will check this.
 - At time of writing the kernel could not be started in that case take a look at: <https://github.com/microsoft/vscode-python/issues/13701> and run


```
python -m pip install 'traitlets==4.3.3' --force-reinstall
```

Congratulations you have now created a custom conda environment that contains all packages to run the scripts and launched a notebook server to experiment with Python notebooks.

Linting

Linting is a process to automatically check code quality within a python project. The linter used in this project is `flake8` which enforces the `pep8` standard.

Open the file `linting.py` in the folder `assignment-0`.

- In the terminal type

```
flake8 .
```

This will output the errors in the terminal.

To see the same errors within the ide we have to set `flake8` as the linter for python files.

- Press `Ctrl + Shift + P` and type `Python Select Linter`. And choose `flake8` from the dropdown.
- In the file `linting.py` you will now see all red and orange underlined markers that show linting errors within the file.
- All problems are also shown in the problems tab next to the terminal.
 - You can also open this with `Ctrl + Shift + M`

You can off course manually fix all these issues, but there is a smarter way to do this. We can use a formatter.

Formatting

A formatter tries to automatically format your code to adhere to the `pep8` standard. Some formatter go beyond that and also try to improve readability and make it nicer. Within the DS guild `Black` is the default formatter.

`Black` is very easy to use: from the terminal type

```
black [source_file_or_directory]
```

For example

black assignment-0\linting.py

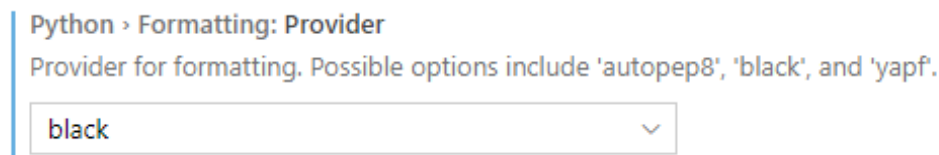
This will format the file.

Another way to format this is by setting a formatter within VSCode.

- Press **Alt + Shift + F** and select black as the default formatter.

If this does not work follow these steps

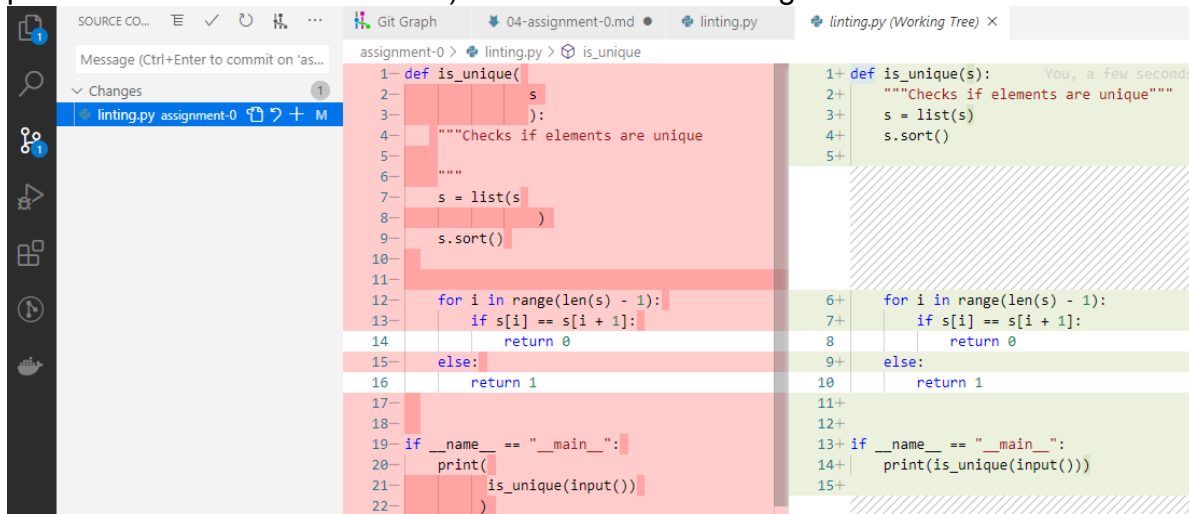
- Press **Ctrl + Shift + P** and type “format” and select “Format Document With” Now select python.
- In the VSCode settings (**Ctrl + `**) search for ‘format python’ and select black in this dialog box.



- You can now use the formatter by typing **Alt + Shift + F**
- You can also apply auto formatting on save, by going to the VSCode settings, and search for “format save” and select the checkbox “Format on Save”.

Git

Now we are going to create a new commit with the formatted file. First explore the changes made to the file by selecting the source control button on the left sidebar. (or press **Ctrl + Shift + G**) You will see something like this:

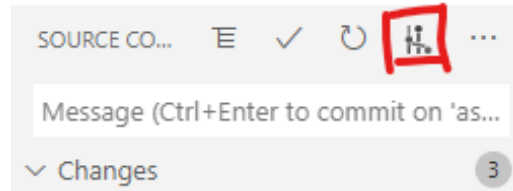


Here you can inspect the differences between the changes.

Now create a new commit with the file.

```
git add assignment-0\linting.py      # stage the file
git commit -m 'applied formatting'    # commit the file and set a
commit message
```

Now inspect your git tree by using the Git Graph extension



- press this button

Here you you can see the commit you have just made.

Conclusion

You now have all software installed and know how to apply linting and formatting within VSCode. If you have any other questions or are stuck in the process do not hesitate to contact Kyllian or Bram.

Assignment-1.md (distributed at the first day of the workshop)

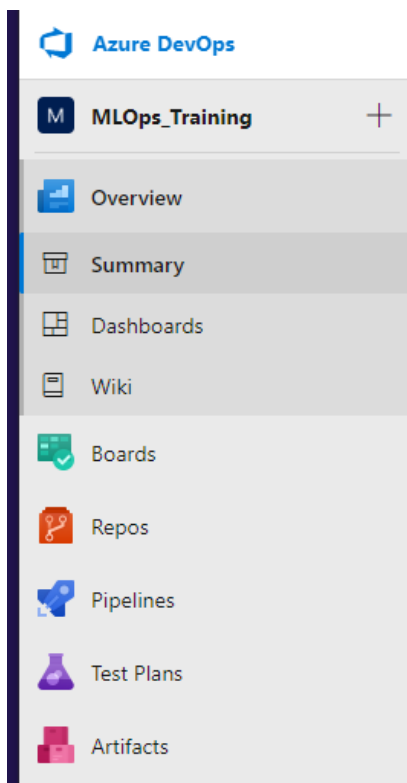
Assignment 1: Git and DevOps

Please make sure you have a working development setup and completed assignment 0.

This assignment covers the basics of Git and DevOps.

1. Getting the Git repository on your local machine

1. To clone the repository go to <https://dev.azure.com/MLOpsTraining/> and login with your cognizant credentials. Here you will see your team project. On the left side you will see the following sidebar



The most important sections are `Repos` and `Pipelines`. In the `Repos` section you will find the Git repository you will be working on as a team. In the `Pipelines` section you will see the pipelines and runs, once specified.

2. Go to the `Repos` section
3. Press the Clone button on the top right of the screen.



4. Click generate Git credentials and store these credentials somewhere on your computer. You will need these later on.
5. Copy the Git https URL and use the URL to clone the repository to a suitable directory on your computer.
 - If you have opened a folder in VSCode this is the working directory.
 - To find out your current working directory run `pwd` in the terminal.

Hint

```
git clone [URL]
```

Extra info: Git Basics getting a Git Repository

This should ask for your credentials. Fill in the credentials saved in the previous step. If not try again after setting `git config --global credential.helper manager`

The repository is now downloaded to your local computer.

Now is also a good time to set your Git identity. This name and email address will be added to every commit. Using the global tag this will be the default for every new repository. Leave out the global tag to set a different username and email for a specific repository.

```
git config --global user.name "John Doe"
git config --global user.email johndoe@cognizant.com
```

For private (corporate) repositories it is fine to use your private (or corporate) email address. For open source repositories it is best practice not to use a real email address.

6. Checkout the branch `assignment-1`

Hint

```
git checkout assignment-1
```

2. Data analysis

The main goal of this first assignment is to get familiar with Git and DevOps. We are going to do this by running some python scripts to analyse data that will be used in the next assignment when building a model.

1. Start a new jupyter lab session as explained in assignment 0.
2. Open the notebook `exploratory data analysis.ipynb` in the folder `notebooks`
3. The dataset describes secondhand car acceptability. Dataset description can be found within the notebook. We want to predict the acceptability.
4. Take a look at the variables. What do you note? Write down two to three things.

5. Take a look at the factorisation function, why do you think factorisation is done in this way? Why not one hot encoding?

In the next section we are going to automate the generation of the files in the output folder by creating a DevOps pipeline. This pipeline will generate the graphs, data description, correlation and variable description.

3. Create a pipeline in Azure DevOps

To be done by one person

1. Go to the Azure DevOps "Pipelines section" on dev.azure.com and select 'create new pipeline'.
2. On the 'Where is your code' page select 'Azure Repos Git'
3. Select the MLOpsTraining repository
4. On the configure your pipeline section choose 'Existing Azure Pipelines YAML file'

Explanation

The Azure DevOps pipelines are specified with YAML files. The other options help you to create your own YAML file. We have already created a pipeline specification for you. By creating the pipeline Azure DevOps starts tracking the file and its triggers. In the next step we are going to define a trigger.

5. In the dialog box select the branch `assignment-1` and the path `/.pipelines/assignment-1.yml`. This is the pipeline we will be triggering.
6. On the Review page select run.
7. Go back to the pipelines page, select the new pipeline and rename the newly created pipeline to `assignment-1`

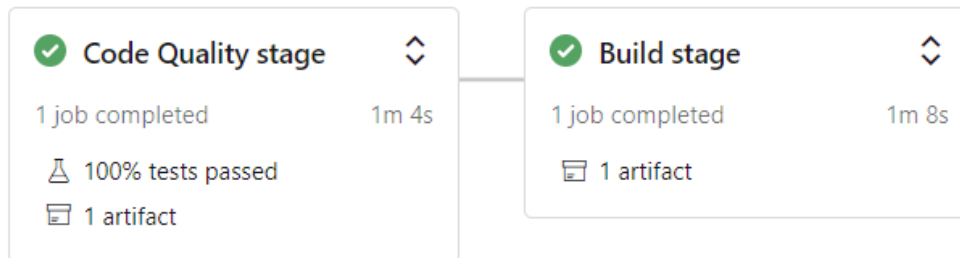
A pipeline as specified in `assignment-1.yml` file has the following structure: A pipeline consist of multiple `stages`, `stages` are major divisions in a pipeline. `Stages` consist of `jobs` and `jobs` are units of work assignable to the same machine. A `job` is a linear series of `steps`, these `steps` can be tasks, scripts, or references to external templates. For more detailed explanation see these [Microsoft docs](#)

If you look into [assignment-1.yml](#) you will see that it consists of several parts:

- `triggers`: these specify when this pipeline should trigger
- `variables`: some global variables specified for pipeline, including the VM it runs.
- `Stages (2x)`:
 - `Code Quality`
 - `Job (1x)`: Check code quality, with multiple steps
 - `Build`
 - `Job (1x)`: Build Job with multiple steps

It thus consist of a pipeline with two stages with each a single job. Visually this (a successfully run pipeline) looks like:

Stages Jobs



In the meantime the pipeline has started running and will show errors. Some scripts in the process block the pipeline because code quality checks fail. You can see the pipeline status in the pipeline section and click on the created pipeline. Here you can see multiple runs for a single pipeline and you can see that the pipeline has failed. You can click on the stages and jobs to figure out why the pipeline has failed. You can also click on the test report.

But before you are going to fix the errors in the pipeline, you will have to create triggers such that the pipeline automatically triggers on code pushes.

4. Creating triggers on the pipeline

To be done by one person

To fix the errors and experiment with code we will create user branches and adjust the pipeline to trigger on commits to all these branches. The assignment-1 branch is protected from direct pushes. This is a common practice in software development where you have to create pull requests, which others have to review, in order to update an important branch.

1. Create a new temporary branch e.g. a1-yourname

```
git checkout -b a1-yourname
```

2. On one computer open the file `/.pipelines/assignment-1.yml`.
3. Adjust the file such that the trigger corresponds with the following:

```
trigger:
  branches:
    include:
      - assignment-1
      - a1-*
```

The pipeline will now trigger on commits to the `assignment-1` branch and any branch that starts with `a1-`. The idea is that you can create branches for each team member, e.g. `a1-bram`.

4. On this computer commit and push the changes to the repository. Commit message suggestion: 'added pipeline triggers'

Hint

```
git add .pipelines/assignment-1.yml
git commit -m 'added pipeline triggers'
git push origin [branch name]
```

What you have done is staged the `assignment-1.yml` file with the `add` command. You committed the file to the current branch, which is `assignment-1` and added a commit message with `-m 'message'`. These edits are local and to push them to the DevOps environment you used the `push` command. You can also set an upstream branch, you specify that a push should automatically go to a certain remote. `git push --set-upstream origin a1-yourname`. In this case you only have to type `git push`.

5. Now create a pull request in the section "Repos - Pull requests". Give it a name, description and select all other team members as reviewers.
6. All team members can now go to the Pull requests section and review the changes that were made. If you are satisfied with the changes approve and merge the pull request.

5. Pull the updated repository to each computer and branch

Now each team member can:

1. pull the repository
2. checkout the `assignment-1` branch and
3. create their own user branch.

Hint

```
git pull # get the updates from the repository
git checkout assignment-1 #if you didn't already
git checkout -b a1-[yourname]
# this is short for
# git branch a1-[yourname] # create branch
# git checkout a1-[yourname] # checkout branch
```

Make sure your branch starts with `a1-` otherwise the triggers don't work.

Now everyone has created a branch to work in.

6. Fixing the errors

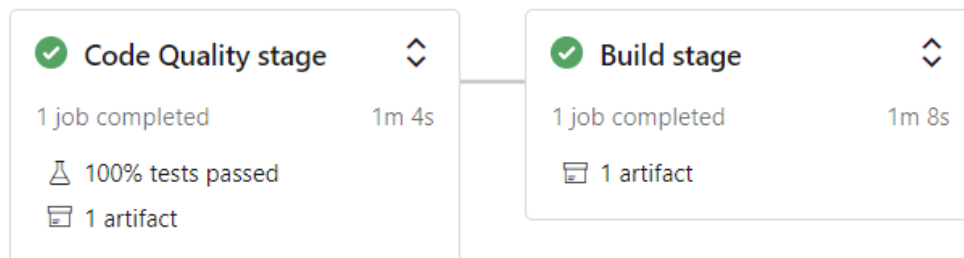
Use the newly created branch to fix the errors. You can find the error descriptions in the pipeline section of DevOps.

1. Find out what causes the errors
2. Can you replicate the error messages on your local machine?
 - What causes the errors?
 - Write down the commands that you have used to replicate the errors locally.

Hint

Look at the output of the pipeline to see where it fails. You can look in the logs as well as the test reports. Also think of the tools used in Assignment-0. You might also want to look in assignment-1.yml.
3. Fix the errors.
 - Use black to fix the errors.
 - Write down how you fixed the errors.
4. If you have fixed the errors push your local branch to the remote and see if your code passes all checks and generates artifacts.
5. Download the artifacts to your local machine.
 - You can find the artifacts in the Pipeline run section under the Build stage.

Stages Jobs



6. If you are satisfied with the outcomes create a pull request from your branch and merge it into the assignment-1 branch.

You have created a full CI/CD pipeline which checks code quality and creates artifacts. In a real case these artifacts resemble the application that you can deploy and can be anything, ranging from a static website, a full fledged application to a machine learning model. A separate pipeline can be created to deploy these artifacts. Exactly this is the topic of assignment 2.

To go to assignment 2 checkout the assignment-2 branch. The assignment is specified in the README.

Assignment 2 and 3 (distributed at second day of the workshop)

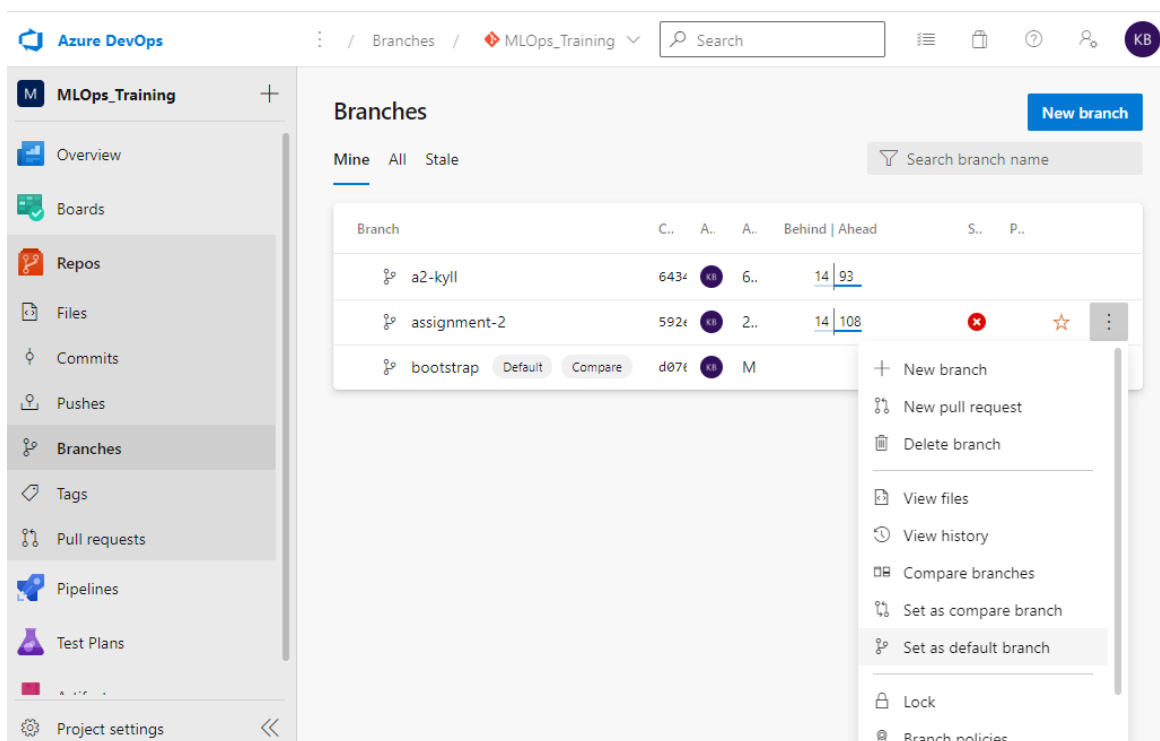
Assignment 2: MLOps, POC to production

This assignment will focus on getting a proof of concept machine learning model in a notebook to production ready code and setting up a CI/CD pipeline to get the code to production.

Hint: open the assignment README in a separate tab.

1. Set up environment

1. In Azure DevOps on the `Branches` blade of the `Repos` tab, set the `assignment-2` branch as the default. See the image below:



2. Check out the branch `assignment-2` locally. When necessary, branch to individual user branches named `a2-[name]` so that everyone works in their own branch

2. Create the CI pipeline

When working on a project we want to automate the integration part. Therefore, before we start working on the code, we will set up the CI pipeline.

This pipeline consists of two stages. The first tests the code and sets up a machine-learning pipeline inside the Azure machine learning studio. The second runs the code with the data in the machine learning studio and saves the outputs. The pipeline is specified by the yaml file `.pipelines/ci/ci.yml`.

However, some parts of the pipeline still need filling in with the correct templates. The first stage consists of one job with two steps that need to be filled in. The second stage has two jobs and one step that need to be filled. Each template file will be used once and has high-level descriptions of its use on the first line.

3. From files in the folder `.pipelines/ci/steps/`, determine which step template files should be inserted in `.pipelines/ci/ci.yml` on line 33 and 34. Replace `[template_file_name]` with the filename including extension.
4. From files the folder `.pipelines/ci/jobs/`, determine which order the job template files should be inserted in `.pipelines/ci/ci.yml` on line 41 and 42. Replace `[template_file_name]` with the filename including extension.
5. From files in the folder `.pipelines/ci/steps/`, use the remaining unused step template file for line 50 in `.pipelines/ci/ci.yml`. Replace `[template_file_name]` with the filename including extension.
6. Commit and push your code to the remote repository.
7. In the Devops environment, create a pull request from your branch into the `assignment-2` branch.
 - Review each other's pull requests and accept if satisfied.
8. Go to the `Pipelines` tab and create a new pipeline. Following the same steps as the previous assignment, build the CI pipeline. Select the `assignment-2` branch and the filled in `.pipelines/ci/ci.yml` file. Run the pipeline when set up.

Select an existing YAML file

Select an Azure Pipelines YAML file in any branch of the repository.

Branch

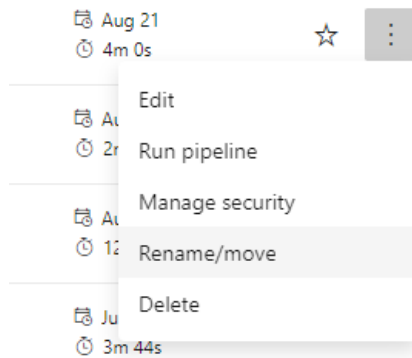
🔗 assignment-2

Path

/.pipelines/ci/ci.yml

Select a file from the dropdown or type in the path to your file

9. In the pipelines overview, rename the pipeline to `Model-Train-Register-CI` for later reference.



10. You have now created an integration pipeline that builds a model each time it has run successfully. For now it will fail, since we still need to implement the code.

If you are stuck on this task, check in with an instructor.

3. Create the CD pipeline

Once the integration pipeline completes and a new model is created, we want to deploy this model automatically. To do this we will set up a CD pipeline in the same manner as the CI pipeline.

The pipeline is specified by the yaml file `.pipelines/cd/cd.yml`.

The deploy pipeline consists of one stage with one job. In this job, three step templates need to be filled in to complete the pipeline. Again, each template file will be used once and has high-level descriptions of its use on the first line.

11. From files in the folder `.pipelines/cd/steps/`, determine which step template file should be inserted in `.pipelines/cd/cd.yml` on line 48.

This step is given three parameters as specified on lines 49-52, so the template should also accept three parameters.

Replace `[template_file_name]` with the filename including extension.

12. From files the folder `.pipelines/cd/steps/`, determine in which order the unused step template files should be inserted in `.pipelines/cd/cd.yml` on line 53 and 54.

Replace `[template_file_name]` with the filename including extension.

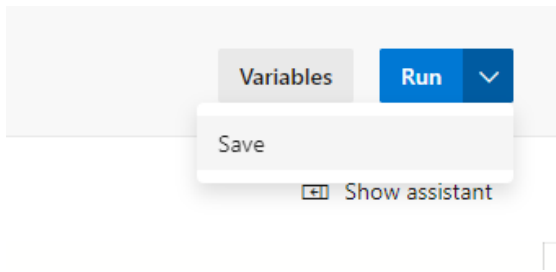
13. Commit and push your code to the remote repository.

14. In the Devops environment, create a pull request to merge your code into the `assignment-2` branch.

- Review each other's pull requests and accept if satisfied.

15. Go to the `Pipelines` tab and create a new pipeline. Following the same steps as the previous assignment, build the CI pipeline. Select the `assignment-2` branch

and the filled in `.pipelines/cd/cd.yml` file. Instead of Run, select Save pipeline from the dropdown next to the button.



16. Rename the pipeline to `Deploy-Model` for clarity from the pipeline overview. Select the “all” filter to show the newly created pipeline.
17. You have now created a deployment pipeline that deploys a container with a Web API. This container runs our model each time a new model artifact is created from the CI pipeline after completing the next assignment.

← Artifacts

Published

Name	Size
Code Coverage Report_19	
coverage_html.js	19 KB
index.html	4 KB
jquery.ba-throttle-debounce.min.js	731 B
jquery.hotkeys.js	3 KB
jquery.isonscreen.js	2 KB
jquery.min.js	94 KB
jquery.tablesorter.min.js	13 KB
keybd_closed.png	112 B
keybd_open.png	112 B
model_training_test_train_py.html	13 KB
model_training_train_py.html	24 KB
model_util__init__py.html	3 KB
model_util_model_helper_py.html	21 KB
status.json	893 B
style.css	7 KB
summary19	4 KB
model	159 B
model.json	159 B

If you are stuck on this task, check in with an instructor.

4. POC notebook to production code

Continuing from the previous assignment, a POC for a prediction model has been made using the exploratory data analyses. For this task, you will move this POC notebook code to a proper Python script and verify that it works.

18. In the folder `experimentation` you will find the Jupyter notebook `notebooks/RandomForest.ipynb`. This notebook contains the POC code for a Random Forest model on the car sales data.
 - Open the notebook using Jupyter and run the cells to explore the model and code.

We will move certain code to functions in the `model/train.py` file. To see how these functions are expected to work, unit tests have been made. This is called Test Driven Development (TDD).

19. To run the unit tests, use the following terminal command: `bash python -m pytest model` This will look for all files starting with `test_` in the `model` folder and run those.

Analyze the output, see why the tests fail and look at the functions in `model/test_train.py`.

Make sure that the `MLOpsTraining` environment is activated.

20. Now use this information to replace the `python raise NotImplementedError` lines in the file `model/train.py` and implement the code from the notebook. To divide the work, each could focus on a particular test error and the corresponding function.

21. Repeat the previous two steps until your tests pass.

(Optional) To test with real data, run the following command in the terminal:

```
python -m model.training.train
```

22. Once you fixed the test or tests, run a syntax check as specified below. Afterwards, create a pull requests to merge your changes again into the remote `assignment-2`.

- Test the file on any syntax errors using the `flake8` command in the terminal. `bash flake8` If there are any, fix them by hand or with the `black` formatter as used in the previous assignment.
- Commit and push your code to your remote branch.
- Go to the repository in the Azure Devops environment and create a pull request from your branch into the `assignment-2` branch.
- Review each other's pull requests and accept if satisfied.
- The merged pull request will trigger the pipeline. Check if it runs successful, otherwise check in the run logs what went wrong. See the following images to see where to find these in the machine learning studio: ml.azure.com.

Directory + Subscription + Workspace ✕

Current directory

Cognizant

Current subscription

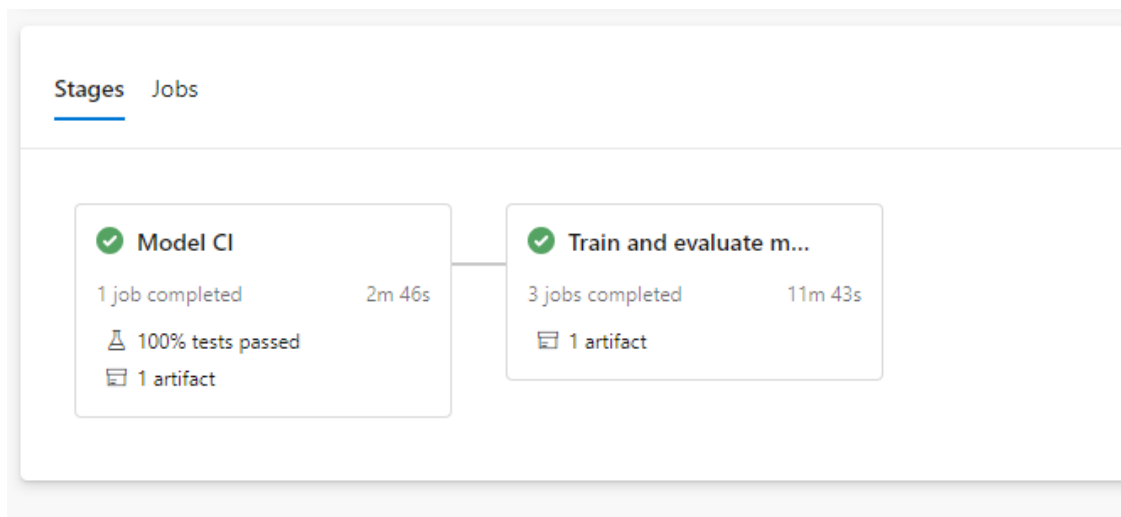
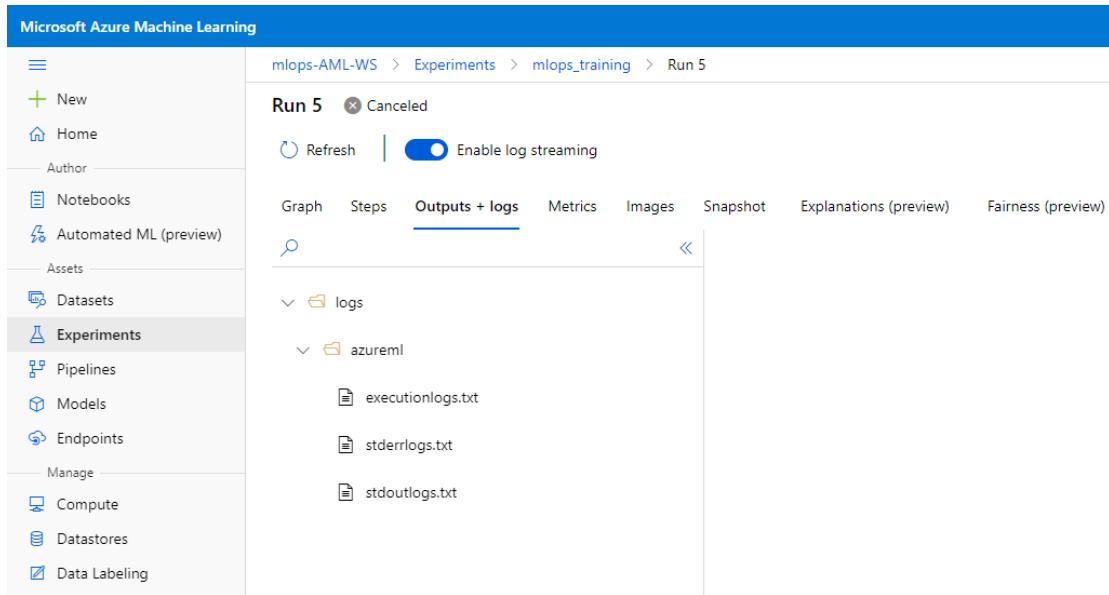
Visual Studio Enterprise(Converted ...

Current workspace

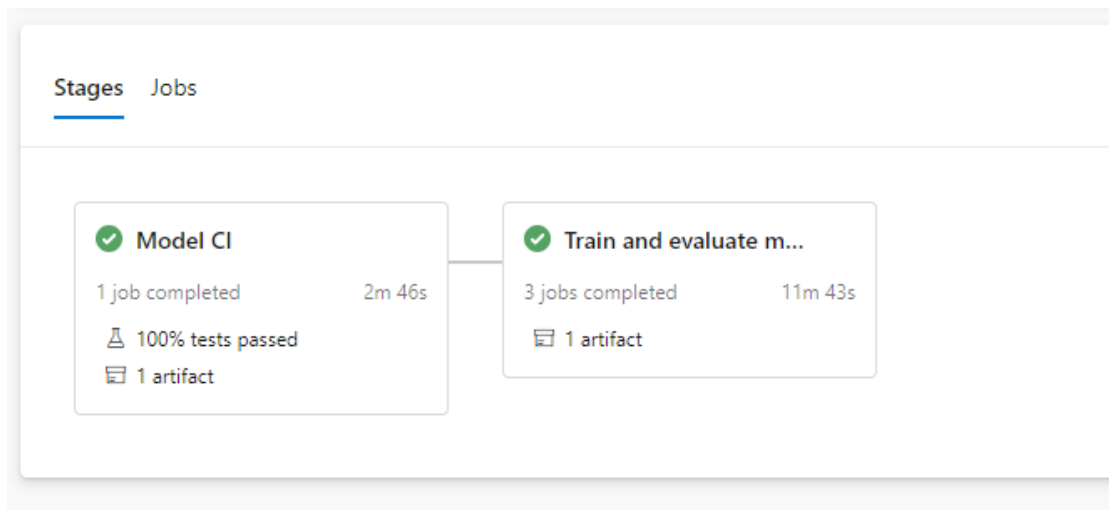
group2ws

As a workspace select your group number.

The screenshot displays the Microsoft Azure Machine Learning (ML) interface. The top navigation bar shows the path: **mlops-AML-WS > Experiments > mlops_training > Run 5**. The left sidebar contains a navigation menu with options: New, Home, Author, Notebooks, Automated ML (preview), Assets, Datasets, **Experiments** (selected), Pipelines, Models, Endpoints, Manage, Compute, Datastores, and Data Labeling. The main content area shows the details for **Run 5**, which has a status of **Canceled**. Below the status, there is a **Refresh** button. A tabbed interface at the top of the main area includes **Graph** (selected), **Steps**, **Outputs + logs**, **Metrics**, **Images**, **Snapshot**, **Explanations (preview)**, and **Fairness (preview)**. The **Graph** tab displays a workflow diagram with three steps: **Train Model** (status: Completed), **Evaluate Model** (status: Completed), and **Register Model** (status: Not started). The **Train Model** step has a command `--model_name model_name --step_output`. The **Evaluate Model** step has a command `--model_name model_name --`. The **Register Model** step has a command `--model_name model_name --step_input`. Data flow is indicated by arrows between steps, with labels like `pipeline_data...` and `_run_step...`.



- The complete pipeline might take up to 15 minutes, see the following



- If the errors are code related repeat the TDD process. If they relate to the CI or CD pipeline, check the respective task and rethink the order of the pipeline templates.

If you are stuck on this task, check in with an instructor.

23. Write down/draw the structure of the pipeline and its steps on a powerpoint. Also prepare a slide on your approach to this assignment and any remarks or questions you have.

Assignment 3: Monitoring the deployed model

This assignment will focus on monitoring your machine learning model in production. This requires assignment 2 to be completed and working.

We use a data monitor to check for drift on incoming data compared to the trained dataset. This might be a first indicator that your model might underperform as the incoming data lies outside the distributions of the trained data and is a useful metric especially when there are no ground truth labels available (yet).

1. Set up environment

First, we need to enable the deployed model to log data so that we can monitor that later.

1. The instructors will provide you with a key. Use this key and your group number for the `TODO` in `score.py`.
2. Commit and push your code to the remote repository.
3. In the Devops environment, create a pull request to merge your code into the `assignment-2` branch. Make sure the `Deploy-Model` pipeline triggers and runs successfully. When done, your deployed model will collect the inference data into a storage container.
 - Review each other's pull requests and accept if satisfied.

2. Collect inference data into a dataset

First we are going to mimic a real life situation where the endpoint is used to do inferencing. For this we need the url of model and we use a python script which will do a POST request to it. The key from the previous task will make sure the data from the request is logged to a datastore file.

4. Go to ml.azure.com. If prompted, select your the Cognizant Directory, Visual Studio Enterprise (Converted to EA) Subscription and your group's workspace. Go to the `Endpoints` tab.
5. Click the `mlops-aci` endpoint. Copy the `REST endpoint` url. This is the URL of the API of your deployed model.

Microsoft Azure Machine Learning

mlops-AML-WS > Endpoints > mlops-aci

mlops-aci

Details Consume

Basic consumption info

REST endpoint

<http://8967a98f-9acb-40b9-918d-50e1e0e044f5.centralus.azurecontainerinsights.io>

- To mimic inference request to this API, run the python script in the terminal `bash python data/post_data.py [REST_endpoint_url]` with `[REST_endpoint_url]` replaced by the copied value. This will post inference data to your Azure Container Instance endpoint.

Next we need to interpret logged data files as a datastore. Then we can use it as a dataset to monitor

- To add it as a dataset in ml.azure.com, go to the `datasets` tab and click `Create dataset > From datastore`.

Home

Author

Notebooks

Automated ML (preview)

Designer (preview)

Assets

Datasets

Experiments

Pipelines

Datasets

Registered datasets Dataset monitors (preview)

+ Create dataset Refresh Unregister

Name	Version	Created on	Modified on	Properties
car_train	1	Sep 4, 2020 9:57 AM	Sep 4, 2020 9:57 AM	Tabular

- Name it `car_inf` and leave the Dataset type on `Tabular`. Click `Next`.

Create dataset from datastore

The screenshot shows a multi-step wizard for creating a dataset from a datastore. On the left, a vertical sidebar contains five steps: 'Basic info' (selected with a blue circle), 'Datastore selection', 'Settings and preview', 'Schema', and 'Confirm details'. The main area is titled 'Basic info' and contains the following fields:

- Name ***: A text input field containing 'car_inf' with a red asterisk and a help icon (i) to its right.
- Dataset version**: A numeric input field containing '1'.
- Dataset type ***: A dropdown menu with 'Tabular' selected and a help icon (i) to its right.
- Description**: A large text area containing the placeholder text 'Dataset description'.

9. Select Create new datastore.

10. Name the datastore also `car_inf`.

- From the `Storage account` dropdown, select the storage account of your group: `group[n]sa`.
- From the `Blob container` dropdown, select `modeldata`.
- As the `Account key`, provide the key given by the instructors.

11. Click Create datastore.

Create dataset from datastore

Basic info

Datastore selection

Settings and preview

Schema

Confirm details

Datastore selection

Select or create a datastore *

☐ Currently selected datastore: None
☐ Previously created datastore
☒ Create new datastore

Datastore name * 👁

car_inf *

Datastore type *

Azure Blob Storage ▾ *

Account selection method

☒ From Azure subscription
 ☐ Enter manually

Subscription ID *

Visual Studio Enterprise(Converted to EA) (d2bda5a8-0cf7-480c-bf1f-6d7ec7b665ba) ▾

Storage account *

edsmlopstraini2567337446 (aia_datascience_training) ▾

Blob container * Creating...

azureml-blobstore-9d13d33f-bab0-4b1d-bc51-3f8d2e0bb826 ▾

Use workspace managed identity for data access in the ML studio ⓘ

No
 Yes

Authentication type * ⓘ

Account key ▾ *

Account key * ⓘ

..... *

ⓘ Note: Azure Machine Learning service does not validate whether the underlying data source exists or whether the user provided credential ... ⌵

Create datastore
 Cancel

Back

Next

12. For the Path put `d2bda5a8-0cf7-480c-bf1f-6d7ec7b665ba/aia_datascience_mlops_workshop/group[n]ws/group[n]ci/model.pkl/*/inputs/*/*/inputs.csv`. Where `[n]` should be replaced by your group number. Click Next.

Create dataset from datastore

Basic info

Datastore selection

Settings and preview

Schema

Confirm details

Datastore selection

Select or create a datastore *

☒ Currently selected datastore: car_inf (Azure Blob Storage)
☐ Previously created datastore
☐ Create new datastore

Path *

Browse

sub_id/edsmlopstrain/mlops-aml-ws/mlops-aci/model.pkl/"*/"/inputs.csv

To include files in subfolders, append "/" after the folder name like so: "(Folder)/".

☐ Skip data validation ⓘ

[Advanced settings](#)

13. Set Column headers to All files have the same headers. Click Next.

Settings and preview

These settings were automatically detected. Please verify that the selections were made correctly or update

File format

Delimited

Delimiter

Comma

Example

Field1,Field2,Field3

Encoding

UTF-8

Column headers

All files have same headers

Skip rows

None

14. Exclude all columns starting with a \$ except \$aml_dc_scoring_timestamp. Set the Properties for \$aml_dc_scoring_timestamp to Timestamp. Click Next

Schema				
Include	Column name	Properties	Type	Format settings and example
<input type="checkbox"/>	Path	Not applicable to selected type	String	
<input checked="" type="checkbox"/>	buying	Not applicable to selected type	String	low, vhigh, low
<input checked="" type="checkbox"/>	maint	Not applicable to selected type	String	high, vhigh, low
<input checked="" type="checkbox"/>	doors	Not applicable to selected type	String	3, 5more, 4
<input checked="" type="checkbox"/>	persons	Not applicable to selected type	String	more, more, more
<input checked="" type="checkbox"/>	lug_boot	Not applicable to selected type	String	big, med, big
<input checked="" type="checkbox"/>	safety	Not applicable to selected type	String	low, high, high
<input checked="" type="checkbox"/>	\$aml_dc_correlation_id	Not applicable to selected type	String	correlation_id, correlation_id, co...
<input checked="" type="checkbox"/>	\$aml_dc_scoring_timestamp	Timestamp	Date	2020-09-01 08:53:12, 2020-09-...
<input type="checkbox"/>	\$aml_dc_boundary	Not applicable to selected type	Integer	60, 60, 60
<input type="checkbox"/>	\$aml_workspace	Not applicable to selected type	String	aml_ws, aml_ws, aml_ws
<input type="checkbox"/>	\$aml_service_name	Not applicable to selected type	String	aml_service, aml_service, aml_se...
<input type="checkbox"/>	\$aml_model_name	Not applicable to selected type	String	model.pkl, model.pkl, model.pkl
<input type="checkbox"/>	\$aml_model_version	Not applicable to selected type	Integer	1, 1, 1
<input type="checkbox"/>	\$aml_request_id	Not applicable to selected type	String	request_id, request_id, request_id

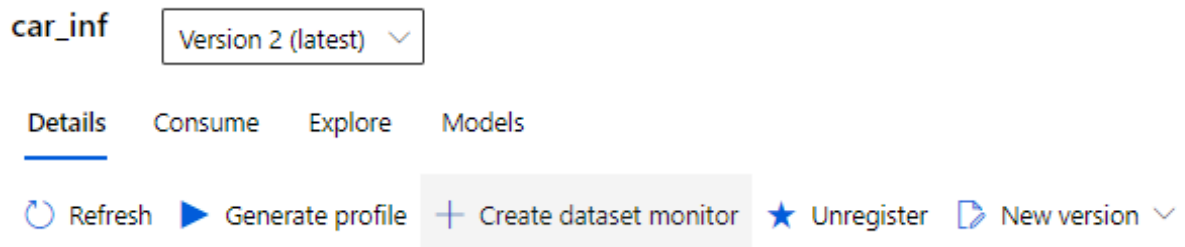
Back Next Cancel

15. Click Create to finish setting up the inference dataset.

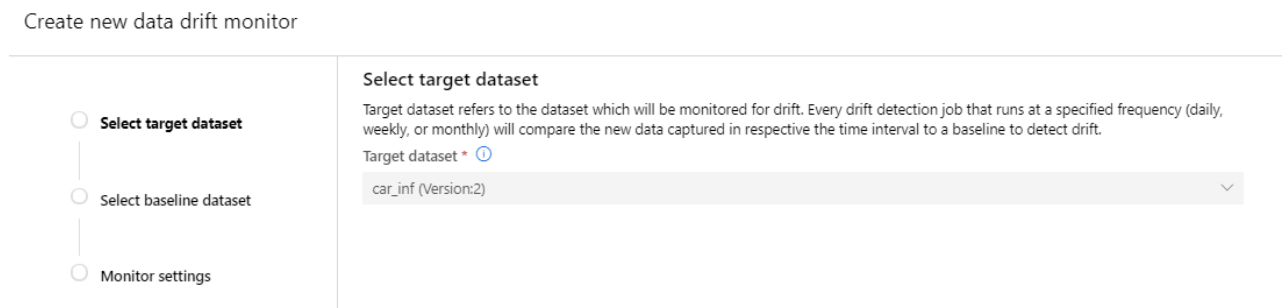
3. Set up a data monitor for data drift

With the inference dataset in place, automatically updating to new data, we can add a monitor to it. This will help us monitor data drift as a indicator if the incoming data is similar to the data the model is trained on. We want this, because we dont know how the model will perform is the data is too different.

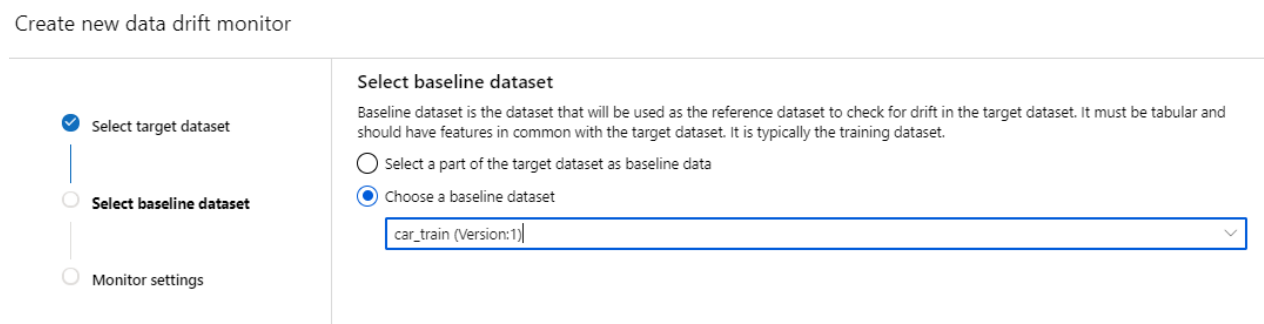
16. Click on the `car_inf` dataset to show its details. From here, click `Create dataset monitor`.



17. The inference dataset is the target dataset. Click `Next`.



18. Select `Choose a baseline dataset`.
Choose the train dataset as baseline and click `Next`.



19. Give the monitor a name and select the available compute target. Disable `Monitor enabled` and click `Create`.

Create new data drift monitor

☒ Select target dataset

☒ Select baseline dataset

☐ Monitor settings

Monitor settings

Settings for the data drift scheduled pipeline that will monitor the target dataset and send an email alert if the data drift percentage is above the set threshold.

Name * ⓘ 🔗

Features * ⓘ

Compute target * ⓘ

Enable or disable schedule monitor runs

☒ Monitor disabled

Frequency ⓘ




Latency (hrs) ⓘ

Email addresses ⓘ

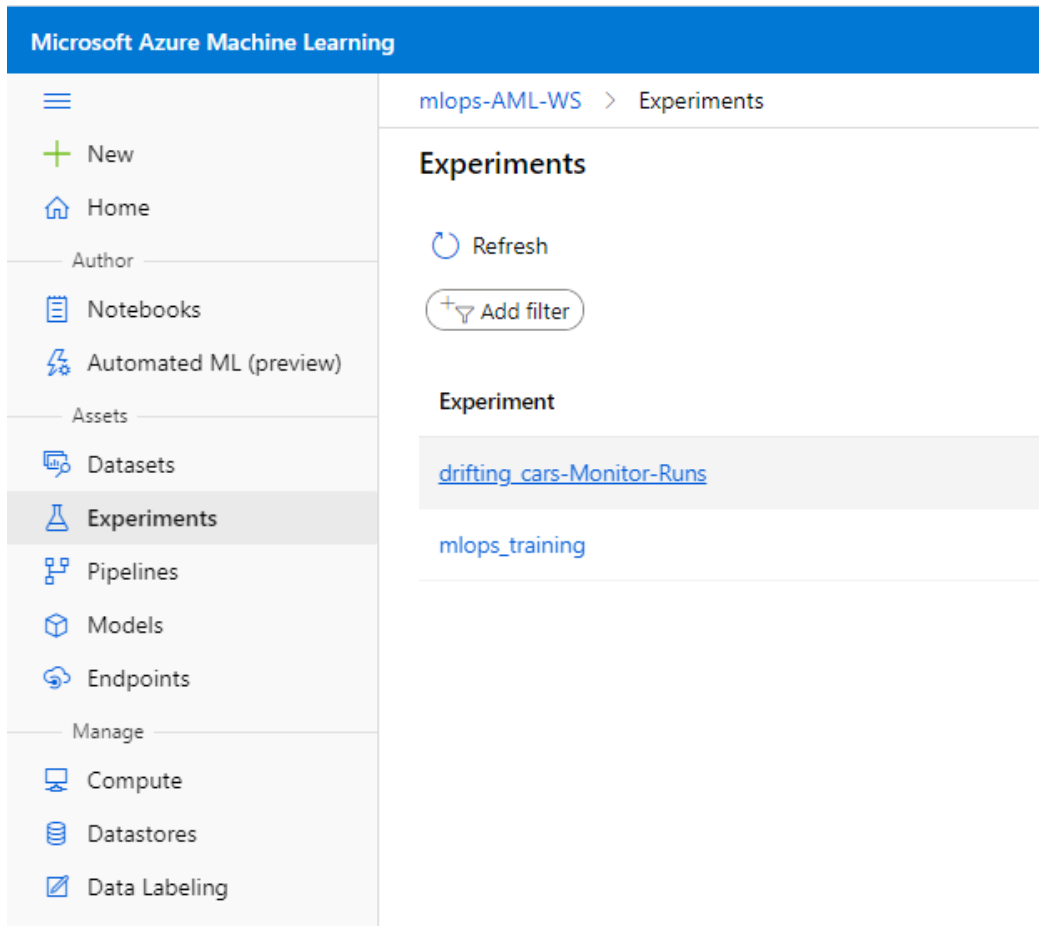
Threshold ⓘ

20%

20. Go to the `Dataset monitors (preview)` tab. Click the `drifting_cars` monitor to go in the detail view and click `Analyze existing data`. Leave the settings as they are and click `Submit`.

 Settings  Analyze existing data  Refresh

21. Wait for the monitor to finish, you can see the progress in the `Experiments` tab.



22. Go back to the Dataset monitors (preview) tab to view the analyses. See the following link: <https://docs.microsoft.com/en-us/azure/machine-learning/how-to-monitor-datasets#understand-data-drift-results>, on how to interpret the results. Create a couple of slides on the interpretation of the results and a proposal on retraining and why this might be needed.