① Azure Subscription
② → Resource Group
③ → Azure ML workspace
④ → Compute Resource (Compute clusters, instances)
⑤ → Datastore (Connects to blobs, ADLS etc)
⑥ → Environment (curated / custom)
⑦ → Datasets / data assets
⑧ → Notebook / scripts
⑨ → Components / Jobs / Pipeline
⑩ → Endpoints (For deployment)

ML Problem solving steps (lifecycle)

① Define the problem (classification / Regression)
② Get the Data (extract data from source)
③ prepare the data
④ train the model
⑤ integrate the model (Deploy the model to an endpoint to generate ~~integrate~~ prediction)
⑥ monitor the model

- Azure ML provides a platform for data scientists to train, deploy and manage their ML models on Microsoft Azure platform.

- To get the Azure ML workspace, you need to Create the Azure ML Service in your Subscription. ②
- for reproducibility, the workspace stores a history of all training jobs, including logs, metrics, outputs & a snapshot of our code.

Via Python sdk

- Then this ②

```
from azure.ai.ml.entities import Workspace

workspace_name = 'mlco-ex',
ws_basic = Workspace(name = workspace_name,
                     location = "eastus",
                     display_name = "Basic ws ex",
                     description = "ex to show how ws is Created",)

ml_client.workspaces.begin_create(ws_basic)
```

② ②

First This

① from azure.identity import DefaultAzureCredential
from azure.ai.ml import MLClient

subscription_id = "abc" resource_group = "def" workspace_name = "mlt"

```
ml_client = MLClient(credential = Credential,
                     subscription_id = subscription_id,
                     resource_group = resource_group)
```

① ①

- You can give individual user's or team access to the Azure ML workspace. access is granted using RBAC.

3 generic roles! —

① Owner — Get full access
② Contributor → " but Can't ~~delete~~ grant access to others
③ Reader → Can only view the Resource.

- Resource in Azure ML refer to the infrastructure you need to run a ML workflow. ③

  The Resource in Azure ML include! —

  ① The Workspace
  ② Compute Resources
  ③ Datastores

- As data scientist u mostly work with assets in Azure ML workspace. Assets are Created & used at Various stages of a project & include

  - Models
  - Environment
  - Data
  - Components.

Create & manage models! —

ⓐ end product of training a model is the model itself
ⓑ u Can train ML model with various framework (Pytorch, Scikit-learn) etc
ⓒ Common way to store model as python pickle file (.pkl)

Create & manage Environments! —

ⓐ environment specify software packages, environment variables, & software settings to run scripts.
ⓑ An env is stored as an image in the Azure Container registry Created with workspace when used for the first time.
ⓒ whenever you want to run a script you Can specify the environment that needs to be used by Compute target The env install all necessary req on the Compute before executing the script making your Code robust & reusable across Compute target.

Create & manage data

ⓐ datastores Contain the Connection information to Azure data storage services, data assets refer to a specific file or folder.

ⓑ we Can use data assets to easily access data every time, without having to provide ~~Can~~ authentication every time you want to access it

ⓒ when we Create a data assets in the workspace, we specify the path to the file or folder & name & version.

Create & manage Components

ⓐ To make easier to share the code, you Can Create the Component in a workspace. To Create, you have to specify name, version, code & env needed to run code.

ⓑ you ~~Can~~ Create Can use Components when Creating Pipelines.

ⓒ Component often represent a step in a pipeline for ex to normalize data → to train regression model → or to test trained model

Train models in the Workspace:-

- to train models with Azure ML workspace we have several options:-
  - ⓐ use Automated ML
  - ⓑ Run a jupyter notebook
  - ⓒ Run a script as a job.

There are different types of jobs depending on how you want to execute a workload:—

Command → Execute a single script.

Sweep → perform hyperparameter tuning when executing a single script.

Pipeline → Run a pipeline Consisting of multiple scripts or Components.

Explore developer tools for workspace interaction: – (5)

- Azure ML provides data scientist with several resources & assets to build & manage ML models.
- we can create & manage resources & assets by using various tools that interact with Azure ML workspace.
  - Azure ML studio
  - Python SDK
  - Azure CLI

Explore studio: —

(a) easiest & most intuitive way to interact with Azure ML workspace is using the studio.

The menu shows what you can do in studio: —

Author → Create new jobs to train & track a ML model
Assets → Create and review assets you use when training models.
Manage → Create & manage resources you need to train models

Explore Python SDK

(a) Currently 2 versions of Python SDK (V1 & V2)

Pip install azure-ai-ml → To install Python SDK within your Python env

```
from azure.ai.ml import MLClient
from azure.identity import DefaultAzureCredential

ml_client = MLClient (
        DefaultAzureCredential (), subscription_id,
        resource_group, workspace )
```

```
from azure.ai.ml import Command   (small C)

job = Command (
    code = "./src",
    command = "python train.py",
    environments = "azure ML-sklearn-0.24-ubuntu18.04-Py37-
                    CPU@latest",
    compute = "aml-cluster",
    experiment_name = "train-model" )

Connect to ws & submit job! —
returned_job = ml_client.create_or_update (job)
```

Explore the CLI

- As a data scientist u may not work with the CLI.

Make Data available in Azure ML ②

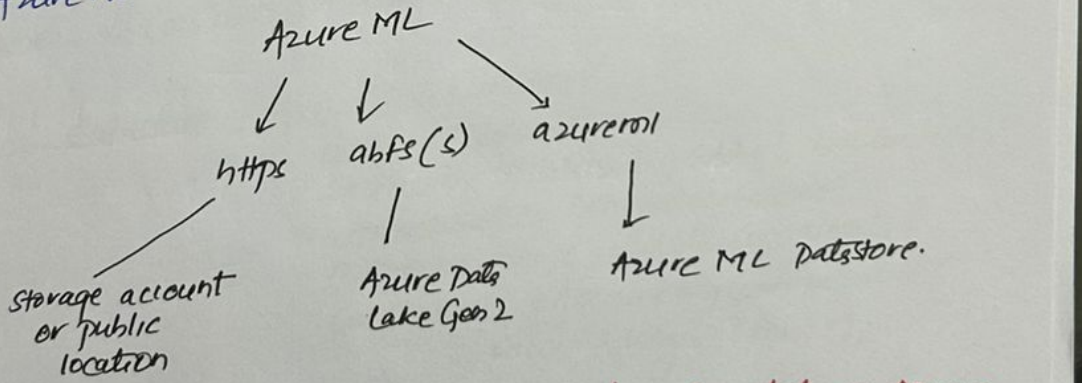
- Data is fundamental element in any ML workload. you need to train a model and you create data when using a model to generate prediction.

- understand URIs

ⓐ u Can store data on your local device or somewhere in the cloud.
ⓑ To find and access data in Azure ML, u Can use Uniform Resource Identifier (URIs).

ⓒ A URI references the location of your data.
ⓓ for Azure ML to Connect to your data, you need to prefix the URI with the appropriate protocol.

ⓔ There are 3 Common protocol when working with data in Context of Azure ML.



- hTTP(s) → use for data stores publicly or privately in Azure blob storage or publicly available hTTP(s) location.

- abfs(s) → use for data store in Azure Data Lake Storage Gen 2.

- azureml: use for data stored in a datastore.

Make Data available in Azure ML ②

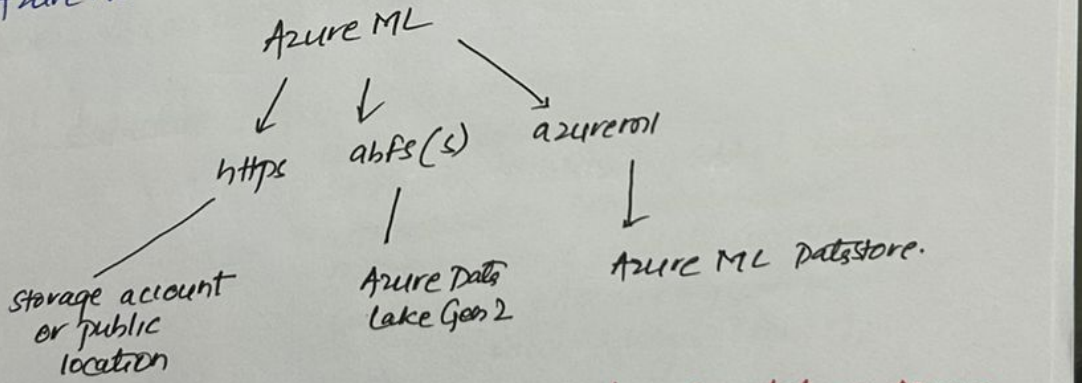
- Data is fundamental element in any ML workload. you need to train a model and you create data when using a model to generate prediction.

- understand URIs

ⓐ u Can store data on your local device or somewhere in the cloud.
ⓑ To find and access data in Azure ML, u Can use Uniform Resource Identifier (URIs).

ⓒ A URI references the location of your data.
ⓓ for Azure ML to Connect to your data, you need to prefix the URI with the appropriate protocol.

ⓔ There are 3 Common protocol when working with data in Context of Azure ML.



- hTTP(s) → use for data stores publicly or privately in Azure blob storage or publicly available hTTP(s) location.

- abfs(s) → use for data store in Azure Data Lake Storage Gen 2.

- azureml: use for data stored in a datastore.

Create a data store!

(a) A datastore in Azure ML is like a connection or pointer to your data storage in Azure.

(b) It lets securely access and manage your data stored in places like Azure blob storage or Azure Data lake without having to write connection details (like account key) in your code every time.

- Data stores – Manages the connection & credential to your storage.
- you can create a datastore through the graphical user interface, the CLI, & the python sdk
- Depending on the storage service you want to connect to there are different options for azure ML to authenticate.

for ex to create a datastore to connect to an Azure blob storage container u can use an account key! –

```
blob_datastore = AzureBlobDatastore(
                    name= "blob-ex",
                    description= "Datastore pointing to blob container",
                    account_name= "mytestblobstore"
                    container_name = "data_container",
                    credentials = AccountKeyConfiguration(
                                account_key= "xxxx"),
                    )

ml_client.create_or_update (blob_datastore)
```

also u can use sas token to authenticate

```
Credential= SasTokenConfiguration(sas_token=" ____ "),)
ml_client.create_or_update (blob_datastore).
```

Create a data asset! —

- In azure ML data assets are references to where data is stored, how to get access and any other relevant metadata.
- You can create data assets to get access to data in datastores, Azure Storage service, public url or data stored in local device.

Benefits of Datastore! —

- U can share & reuse data with other members of the team such that they don't have to remember file locations.
- U can seamlessly access data during model training without worrying about connection strings or data paths.
- U can version the metadata of the data assets.

3 main types of data assets u can use: —

- uri file → points to a specific file.
- uri folder → Points to a folder
- ML Table → Points to a folder or file & includes a schema to read as tabular data.

- Data assets are most useful when executing ML Tasks as Azure ML Jobs.

Create a URI file data asset! —

- A uri file data asset points to a specific file. The supported paths you can use when creating a URI file data asset are! —
  - Local (./<path>)
  - Azure Blob storage: wasbs://<account_name>.blob.core.windows.net/<container_name>/<folder>/<file>
  - Azure data lake storage (Gen 2): abfss://<file_system>@<account_name>.dfs.core.windows.net/<folder>/<file>
  - Datastore: azureml://datastores/<datastore_name>/paths/<folder>/<file>

To Create URI file data asset


```
From azure.ai.ml.entities import Data
From azure.ai.ml.constants import AssetTypes.

my_path = '<supported-path>'

my-path = Data(
            path = my-path,
            type = AssetTypes.URI_file
            description = "<description>",
             name = '(name)',
             version = "<version>")
ml_client.data.create_or_update(my_data)
```

URI_folder
MLTABLE

Work with Compute targets in Azure ML —

(a) As data scientist we can train ML model in local device for large scale projects it not possible

(b) In Azure ML u can use various types of managed cloud Computes.

(c) In Azure ML Compute targets are physical or virtual Computers on which jobs are run.

Compute Instance → ideal for experimentation (virtual machine)

Compute Clusters → virtual machine, Multi node cluster (Scale up or down to meet req)

Kubernetes clusters

Attached Compute → Allow u to attach existing Compute

Serverless Compute → A fully managed on demand Compute u can use for training jobs.

Compute Instance (experimentation) → Compute cluster (removing to production) when

Compute target for deployment (The selection depends upon whether u want batch or real time prediction)

→ for batch (Compute clusters & Azure ML serverless compute are ideal for pipeline jobs as they are on demand & scalable.

→ when real-time prediction

- u need a type of Compute that u running continously
- real time deployment therfore benefit from more lightweight (cost effective) Compute.
- Containers are ideal for real time deployment.

Create & use a Compute instance - (using python SDK)

otherwise easily using AzureML

```
from azure.ai.ml.entities import ComputeInstance
ci_basic_name = "basic-ci-12345"
ci_basic = ComputeInstance(
        name=ci_basic_name,
        size="Standard_DS3_v2"
        )
ml_client.begin_create_or_update(ci_basic).result()
```

Create and use a Compute Cluster

```
from azure.ai.ml.entities import AmlCompute
cluster_basic = AmlCompute(
        name='cpu-cluster',
        type='amlcompute',
        size='STandard_DS3_v2',
        location='westus',
        min_instances=0,
        max_instances=2,
        idle_time_before_scale_down=120,
        tier='low_priority',
        )
ml_client.begin_create_or_update(cluster_basic).result()
```

Size of vm type of each node within the Compute cluster.

Vm are low priority or dedicated.

Use a Compute cluster! —

There are 3 main scenario in which you can use a Compute cluster! —

1. Running a pipeline job you built in the Designer.
2. Running an automated ML job.
3. Running a script as a job.

Ex Command job (with Compute cluster)

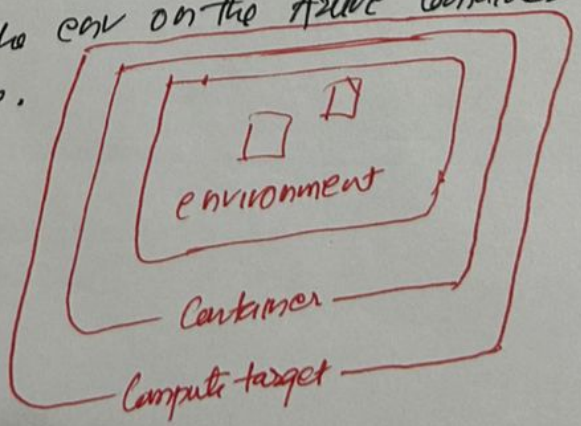
```
from azure.ai.ml import command
job = command(
        code = "./src",
        command = "python diabetes-train.py",
        environment = "AzureML-sklearn-0.24-ubuntu18.04-Py37-cpu
                                                    @latest",
      compute = "cpu-cluster",
      display_name = "train_with_cluster"
      experiment_name = "diabetes-training"
        )

returned_job = ml_client.create_or_update(job)
aml_url = returned_job.studio_url
print("Monitor your job at", aml_url)
```

Work with Environment in Azure ML :—

- As a data scientist we want to create a code that works in any development environment
- In the enterprise ML ops where experiments may be run in various compute contexts, it is important to be aware of the environment in which your experiment code is Running.
- you can use Azure ML environments to create environment & specify runtime configuration for an experiment.
- when Azure ML workspace is created, curated env are automatically created & made available.
- you can create & manage your own custom environment & register them in the workspace.

- Python code runs in the context of a virtual environment that defines the version of the python runtime to be used as well as the installed packages available to the code. Comonly managed using in environment using conda or pip.
- To improve portability you usually create environment in Docker containers that are in turn hosted on compute targets, such as development compute, VM or clusters in the cloud.
- Azure ML builds the env on the Azure Container registry associated with the workspace.



To list all the env using python SDK

(B)

```
envs = ml_client.environments.list()
for env in envs:
    print(env.name)
```

To review details of specific env

```
env = ml_client.environments.get(name="my-env", version="1")
print(env)
```

Explore & use Curated Environments :-

(a) Curated env are prebuilt environments for the most common ML workloads available in your workspace by default

(b) Curated env use the prefect AzureML — and are designed to provide for scripts that uses popular ML framework & tooling.

```
env = ml_client.environments.get("AzureML-sklearn-0.24-ubuntu-04-Py37-Cpu", version=44)
print(env.description, env.tags)
```

Use a Created Environment :-

```
from azure.ai.ml import command
job = command(
    code="./src",
    command="python train.py",
    environment="AzureML-sklearn-0.24-ubuntu18.04-Py37-Cpu@latest",
    compute="aml-cluster",
    display_name="train-with-curated-env",
    experiment_name="train with curated env"
    )
returned_job = ml_client.create_or_update(job)
```

- As Curated environment allows for faster deployment time. It is best practice to first explore whether one of the preCreated environment can be used to run the code.

- If your env doesn't include all necessary packages to run your code, the job fails.

- When job fails u can review the detailed error logs in the [output + log] — tab of your job in Azure ML studio.

- A common error message that indicates ur env is incomplete is ModuleNotFoundError.
  the module that can't found is listed in the error msg.
  you can update the env to include the libraries.

Create & use Custom environment :—

- when u need to Create your own environment in Azure ML to list all necessary packages, libraries and dependencies to run your scripts, you Can Create Custom env.
- you Can define an env from a ① Docker image
  ② Docker build Context
  ③ Conda Specification with Docker image.

Create a Custom env with Docker image :—

ⓐ The easiest approach is likely to be Create an env from a Docker image. Docker image Can be hosted in a public registry like Docker hub.

To Create an env from a docker image u Can use Python SDK :—

```
from azure.ai.ml.entities import Environment
env_docker_image = Environment(
        image = "pytorch/pytorch:latest",
        name = "public-Docker-image-example",
        description = "Env Created from public image",)
ml_client.environments.Create_or_update(env_docker_image)
```

u Can also use Azure ML base images to Create an images (which is similar to image used by Curated env).

```
from azure.ai.ml.entities import Environment
env_docker_image = Environment
        image = "mcr.microsoft.com/azureml/openmpi3.1.2-
                                        ubuntu18.04",
        name = "aml-docker-image-example",
        description = "env Created from Azure ml docker image",
)
ml_client.environments.Create_or_update (env_docker_image)
```

Create a Custom env with a Conda Specification file!—

(a) Though docker images contains all neccessary packages when working with a specific framework, it may be that you need to include other packages to run ur code.
ex!— u want pytorch to train & mlflow to track model

(b) when u need to include other packages or libraries in your environment, u can add conda specification file to a docker image when creating the env.

(c) A conda specification file is a YAML file which list the packages needs to be installed using conda or PIP.

yml
```
name: basic-env-cpu
channels:
  - conda-forge
dependencies
  - python 3.7
  - pandas
  - numpy
```

To create the env from a docker image & a conda specification file —

```
from azure.ai.ml.entities import Environment
env_docker_conda = Environment(
    image = "mcr.microsoft.com/azureml/openmpi3.1.2-ubuntu18.04",
    conda_file = "./conda-env.yml",
    name = "docker-image-plus-conda-ex",
    description = "env created from docker image & conda env", )
ml_client.environment.create_or_update(env_docker_conda)
```

use an environment! —

- most commonly u use an env when you want to run a script as a command job.
- to specify which env u want to use to run your script — u reference an env using the
  <curated-env-name>:<version or
  <curated-env-name>@latest

```
from azure.ai.ml. import command
job= command (
        code= "./src",
        command= "python train.py",
        environment= "docker-image-plus-conda-ex":1",
        compute = "aml-cluster",
        display_name= "train-custom-env",
        experiment_name= "train-custom-env")
#submit job: -
returned_job= ml_client.create_or_update (job)
```

* when u submit the job env is built.
* first time u use an env it can take 10-15 min to build.
* u can review the logs of the env build in the logs of the job.
* when Azure ML builds an env it is added in the list of custom env in the workspace.
* The image of the env is hosted in the Azure container registry associated with the WS.
* whenever u use the same env for another job or script the env is ready to go and doesn't need to be build again.

Experiment with Azure ML:—

Find the Best classification model with AutoML:—

- Automl allows you to try multiple preprocessing transformation and algorithms with your data to find best ML model.
- u Can Create Automl experiment using the Virtual interface of Azure ML studio, the Azure CLI, Python sdk.
- u Can use AutoML for tasks like regression, forecasting, image classification & NLP.
- Prepare ur data to use Automl for classification → Configure & run Automl experiment → Evaluate & Compare models

Preprocess data & Configure featurization:—

(1) before u run auto ml experiment u need to prepare ur data.

(2) once the data is Collected → u need to Create a data asset in Azure ML → In order to understand how to read the data, u need to Create a MLTable data asset that includes schema of the data.

(3) you Can Create MLTable data asset when your data is stored in a folder together with a MLTable file. when you have Created the data asset u Can Specify it as input with the below Code:—

Experiment with Azure ML :—

Find the Best classification model with AutoML :—

- AutoML allows you to try multiple preprocessing transformation and algorithms with your data to find best ML model.
- u Can Create AutoML experiment using the Virtual interface of Azure ML studio, the Azure CLI, Python SDK.
- u Can use AutoML for tasks like regression, forecasting, image classification & NLP.
- Prepare ur data to use AutoML for classification → Configure & run AutoML experiment → Evaluate & Compare models

Preprocess data & Configure featurization :—

(1) before u run auto ml experiment u need to prepare ur data.

(2) once the data is Collected ⇒ u need to Create a data asset in Azure ML → In order to understand how to read the data, u need to Create a MLTable data assets that includes schema of the data.

(3) you Can Create MLTable data asset when your data is stored in a folder together with a MLTable file. when you have Created the data asset u Can Specify it as input with the below Code :—

```
from azure.ai.ml.constants import AssetTypes
from azure.ai.ml import Input
my_training_data_input = Input (type = AssetTypes.MLTABLE, path =
                                "azureml:input-data-automl:1")
```

- once u have created the data asset u can configure the Automl experiment.

- Before AutoML trains a model, preprocessing transformations can be applied to your data.

understand Scaling and Normalization! —

(1) Automl applies automatically scaling & normalization to numeric data.

(2) Configure optional featurization! —

u can choose automl apply preprocessing transformations such as—

① missing value imputation
② Categorical encoding
③ Dropping high Cardinality features (such as ID)
④ feature engineering (Date fe, Day, months etc)

- By default Automl will perform featurization on your data. you can disable it if u don't want the data to be transformed.

Run an automated ML Experiment! —

⇒ To run an automl experiment u Can Configure & submit the job with Python SDK v2.

Logistic Regression, DT, RF, NB, SVM, XGBOOST etc.
LGBM

⇒ By default Automl randomly selects from the full range of algorithm fo specified tasks

⇒ u Can block individual algorithm. (if u know that won't perform)

⇒ Configure the automl experiment! —

• Python SDK v2 to Configure automl exp & job.

```
from azure.ai.ml import automl
classification_job = automl.classification(
    compute = "aml-cluster",
    experiment_name = "auto-ml-class-dev",
    training_data = my_training_data_input,
    target_column_name = "Diabetic",
    primary_metric = "accuracy",
    n_cross_validations = 5
    enable_model_explainability = True )
```

Automl needs MLTable data asset as input.

Specify the primary metric! —

• primary_metric is important setting.
• it's a target performance metric for which the optimal model will be determined.

To retrieve the list of metrics available :

→ from azure.ai.ml.automl import ClassificationPrimaryMetrics
list(ClassificationPrimaryMetrics)

Set the limits

- train ML model will cost compute
- To minimise the cost & time spent on training u can set limits to an automl experiment or job by set_limits()

Several options to set limit

(1) timeout_minutes → No of min after which the complete AutoML exp is terminated.

(2) trial_timeout_minutes → max^m no of minutes one trial can take.

(3) max_trials → max no of trials, no models that will be trained.

(4) enable_early_termination → whether to end the exp if score isn't improving in short term.

```
classification_job.set_limits(
        timeout_minutes=60,
        trial_timeout_minutes=20,
        max_trials=5,
        enable_early_termination=True, )
```

(5) To save time u can also run multiple trials in parallel.

(6) when use compute cluster you can have as many parallel trials as you have nodes.

(7) the max^m no of parallel trials is therefore related to max^m no of nodes your computer has.

(8) if u have to set max^m no of parallel trials to be less than max^m no of nodes, use [max_concurrent_trials]

Submit the automl experiment! —

- u can submit an automL job with the below code! —

```
returned_job = ml_client.jobs.create_or_update(
                    classification_job)
```

⇒ u can monitor Automl exp job runs in the Azure ML studio. u can get a direct link to the AutoML job by running following codes.

```
aml_url = returned_job.studio_url
print ("Monitor your job at", aml_url)
```

Evaluate & Compare models

- when an automl exp is completed, u will review the model that have been trained & performed the best.
- u select automl exp in Azure ML studio & explore.
- overview | Data guardrails | Models (all model trained (tab)) | output + logs | child jobs

Explore preprocessing steps! —

- when u enabled featurization for ur automl exp data guardrails will automatically be applied too. The 3 guardrails that are supported by classification model are! —
  - (a) class imbalance detection / balancing detection
  - (b) missing feature value imputation
  - (c) high Cardinality feature detection

each of these guardrails will show 1 of 3 possible state:—

Passed → no problem, no action required
Done → changes were applied to your data
Alerted → An issue detected but Couldn't be fixed. you should review the data to fix the issue.

Track model Training in jupyter Notebook with mlflow! —

Configure MLFlow for model tracking in notebooks! —

- MLFlow is an open source library for tracking & managing ml experiment.
- MLFlow tracking is a component of MLFlow that logs everything about the model you are training such as parameters, metrics, artifacts.
- To use mlflow in Azure ML ws u need to install the necessary libraries & set Azure ML as the tracking store.
  - Pip show mlflow
  - Pip show azureml-mlflow — contains integration code of Azure ML with MLFlow.

use mlflow on a local Device! —

- when u prefer working in notebooks on a local device, u can also use MLFlow

1. install mlflow & azureml-mlflow package.
   - pip install mlflow.
   - pip install azureml-mlflow.
2. Navigate to ML studio
3. select the name of ws
4. select view all properties in Azure portal
5. Copy the value of MLFlow tracking URI
6. use your local notebook to Configure MLFlow to point in AZ ML ws & set the workspace tracking URI.

```
mlflow.set_tracking_uri = "MLFLOW-TRACKING-URI"
```

Train & Track models in notebooks! —

(1) To group model training results, u will use experiment.

Create an MLFlow experiment! —

(1) u can create a MLFlow experiment which allows you to group runs.

(2) If u won't create experiment, MLflow will assume the default experiment with name Default.

To create an experiment run the following command in notebook! —

```
import mlflow
mlflow.set_experiment(experiment_name="heart-condition-classifier")
```

log results with MLflow! —

⇒ To start a run tracked by MLFlow, u will use start_run()

u can! —

(1) enable autologging
(2) use Custom logging

Enable autologging

↳ (a) MLFlow Supports automatic logging for popular ML libraries
(b) If u are using those mlflow tells the framework u are using to log all the metrics, parameters, artefacts & models that framework Consider relevant.

- u can turn autologging by using (autolog) method.

ex! — to enable autologging for xgboost model u can use [mlflow.xgboost.autolog()]

```
from xgboost import XGBClassifier

with mlflow.start_run():
    mlflow.xgboost.autolog()
    model = XGBClassifier(use_label_encoder=False, eval_metric="logloss")
    model.fit(x_train, y_train, eval_set=[(x_test, y_test)],
              verbose=False)
```

- As soon as mlflow.xgboost.autolog is called, MLFlow will start a run within an experiment in Azure ML to start tracking the experiment run.
- When the job has completed, u can review all logged metrics in the Studio.

Use Custom logging

(1) manually logging models is helpful when you want to log supplementary or custom information that isn't logged through autologging

(2) u can choose to only use custom logging, or use custom logging in combination with autologging.

Common fun^n used with custom logging are:—

(1) mlflow.log_param(): — logs a single key-value parameter. use this fun^n for an input parameter you want to log.

(2) mlflow.log_metric() → logs a single key-value metric. value must be a number use this fun^n for any output you want to store with the Run.

(3) mlflow.log_artifact() →, logs a file.
- use this fun^n for any plot u want to log, save as image file first.

(4) mlflow.log_model() → logs a model, uses this function to create an MLflow model, which may include a custom signature, env & input example.

```
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score

with mlflow.start_run():
    model = XGBClassifier(use_label_encoder=False, eval_metric="logloss")
    model.fit(X_train, y_train, eval_set=[(X_test, y_test)], verbose=False)

    y_pred = model.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    mlflow.log_metric("accuracy", accuracy)
```

- Custom logging gives u more flexibility
- When ur job is completed u can review all logged metrics in the studio.

Optimize model Training with Azure ML: —

• Run a training script as a Command job in Azure ML:—

(1) notebook are ideal for Experimentation & development.
(2) scripts are better for production scenarios & workloads.
(3) In Azure ML we can run scripts as a Command job.
(4) when we submit Command job u can configure various parameters like input data, Compute env.

Convert notebook to a script!—

- To create production ready script u have to:—
  (1) Remove non essential code. — Print, describe etc.
  (2) Refactor ur code into functions
  (3) Test your script in the terminal. — Python train.py

Run a script as a Command job:—

- when u have a script that train ML model, u can run it as a Command job in Azure ML:—

→ Configure & Submit Command job! —

- To run script as a Command job, u will need to configure & submit the job.

```
from azure.ai.ml import Command
job = Command(
    code="./src",                                  # folder that include the script to run.
    command="python train.py",                     # Command:- specifies which file to run
    environment="AzureML-sklearn-0.24-ubuntu18.04-py37-cpu@latest",   # necessary package to be installed on the compute before running the Command.
    compute="aml-cluster",                         # Compute to use to run the Command
    display_name="train-model",                    # name of individual job
    experiment_name="train-classification-model")  # name of exp the job belongs to
```

returned_jobs = ml_client.create_or_update(job)

(1) U can monitor and review the jobs in Azure ML studio.
(2) All jobs with the same experiment name will be grouped under same experiment.
(3) U can find the individual jobs using the specified display name.

use parameter in Command job:—

(1) U can increase the flexibility of your script by using parameters
(2) To use parameter in script, U must use argparse to read arguments passed to the script & assign them to variables.
(3) Any parameter you expect should be defined in the script
(4) Python train.py --training_data diabetes.csv

also

In a script like this Command = "python train.py --training_data diabetes.csv"

Track model training with mlflow in jobs:

- Scripts are ideal when u want to run ML workloads in production env.
- When u use mlflow together with Azure ML, u can run training scripts locally or in the cloud.
- u can review model metrics & artefacts in Azure ML workspace to compare runs & decide on next steps.

- When u train model with scripts include MLflow to track parameters, metrics & artefacts
- When u run scripts as a job in Azure ML u will be able to review input parameters & output for each run.

There are 2 options to track ML jobs with MLFlow:—

(1) enable autologging using mlflow.autolog()
(2) use logging fun^{n} to track custom metrics using mlflow.log_*

Include MLflow in the environment:—

(1) To use mlflow during training job the mlflow & azureml-mlflow PIP packages needs to be installed on the Compute executing the script.
(2) So u have to include these 2 packages in the env
(3) env by YAML file describe the Conda env.

```
name: mlflow-env
channels:
  - conda-forge
dependencies:
  - python=3.8
  - pip
    - mlflow
    - azureml-mlflow
```

→ once env is defined & registered make sure to refer to it when submitting a job

Enable autologging! –

- for common libraries in ML we can enable autologging in MLflow.
- autologging logs parameter, metrics & model artifacts without anyone needing to specify what needs to be logged.

scikit-learn
Pytorch
Spark
~~fastai~~
etc
} — supported

To enable autologging add following code in training script! —

```
import mlflow
mlflow.autolog()
```

Log metrics with mlflow! —

mlflow.log_param() → log single (K,V) pair — use this fun^n for an input parameter (parameter) u want to log.

mlflow.log_metric() ⇒ • log single (K,V) pair metric
• value must be a number
• use this fun^n to store any op you want to store with the run.

mlflow.log_artifact() → • logs a file.
• use this fun^n for any plot u want to log
save as image file first.

```
import mlflow
reg_rate=0.1
mlflow.log_param("Regularization rate", reg_rate)
```
} To add mlflow to an existing script.

2. Submit the job

(1) Submit the training script as a job in Azure ML. when mlflow is used in script & run it as a job all tracked parameters, metrics & artefacts are stored with the job run

(2) u have to make sure the env u refer to job include the necessary package & script describes which metric u want to log.

View metrics & evaluate model' ——

- After u have trained and tracked models with mlflow in Azure ML u can explore the metrics & evaluate your models.

(1) Review metrics in Azure ML studio
(2) Retrieve Runs and metrics with mlflow.

(a) find ur exp (b) in Details tab all logged parameters are shown under Params.
(c) Select Metrics tab & select metrics u want to explore.
(d) plots logged as artifacts can found under Images.
(e) model assets that can be used to register & deploy the model are stored in models under Outputs+logs

In Note books

search all experiments

(1)
```
exps = mlflow.search_experiments (max_results=2)
for exp in exps:
    print (exp.name)
```

To retrieve archived experiments

(2)
```
from mlflow.entities import ViewType
exps = mlflow.search.experiments (viewtype = ViewType.ALL)
for exp in exps:
    Print (exp.name)
```

To retrieve Specific experiment

(3)
```
exp = mlflow.get_experiment_by_name (experiment_name)
print (exp)
```

Retrieve Runs

- mlflow allows u to ~~run~~ search for runs inside any exp · u need experiment id or its name.

  mlflow.search_runs(exp.experiment_id)

- u can use search_all_experiments=True) if u want to search across all experiments in ur.

- exp are queried ordered by descending
  ~~to look for~~ Specific combination in ur
- mlflow.search_runs(
  exp.experiment_id, filter_string = "params.num_boost_round='100'", max_results=2)

Manage & review models with in Azure ML:—

Register an MLFlow model in Azure ML:—

(1) After training u have to deploy ML model in order to integrate the model with an application.

(2) It can be easily done in Azure ML ie. deployment of batch or online endpoint when u register the model with mlflow.

(3) freshdata → retained → better performing model

(4) how to register the model with mlflow in Azure ML → to prepare the model for deployment.

(a) log model with MLFlow:—

(1) mlflow is an open source framework that streamline ml deployment regardless of type of model you trained & framework you used.

(a) why use MLFlow:—

(1) when u train ML model in Azure ML u can use MLflow to register your model.

(2) it standardized the packaging of models which meant that an MLFlow can be easily be imported or exported across different workflows

(3) when u train & log a model you store all relevant artefacts in a directory.

(4) when you register the model an ML model file is created in that directory. the ML Model file contain model's metadata which allows for model traceability.

⑤ u Can register model with MLFlow by enabling autologging or by using Custom logging.

use autologging to log a model:—

- include mlflow.autolog() → To enable autologging
- Ihey automatically logs parameters, metrics, artifacts & the model you train.
- model logged when the .fit() method is Called.

```
mlflow.<flavor>.autolog()
mlflow.keras.autolog()
mlflow.pytorch.autolog()
    etc.
```

- when autologging is used an op folder is created which includes all necessary model artifacts including MLModel file that references these files and include the model's metadata.

Manually log a Model:

- when u have want to have more Control over how the model is logged u Can use autolog (for parameters, metrics & other artifacts) and set log_models = False.
- when u set false mlflow doesn't automatically log the model & u Can add it manually.

* The Schemas of the expected inputs & outputs are defined as the signature in the MLmodel file.

Customize the Signature! —

(1) The model signature defines the schema of the model's input & output.

(2) The signature is stored in JSON format in the ML Model file together with other metadata of the model.

(3) model signature can be inferred from datasets or created manually by hand.

(4) To log a model with a signature that is inferred from your training dataset & model predictions you can use infer_signature.

```
from mlflow.models.signature import infer_signature
signature = infer_signature (iris_train, RFC.predict (iris_train))
mlflow.sklearn.log_model (RFC, "iris_rf", signature = signature)
```

Creating Signature manually! —

- ```
  from mlflow.models.signature import ModelSignature
  from mlflow.types.schema import Schema, ColSpec

  input_schema = Schema ([
      ColSpec ("double", "Sepal length (cm)"),
      ...
  ])

  Define schema for output
  output_schema = Schema ([ColSpec ("long")])
  Create Signature
  signature = ModelSignature (inputs = input_schema, outputs =
                              output_schema)
  ```
  ```

Understand the MLflow model format! —

(1) mlflow uses an MLModel format to store all relevant model assets in a folder or directory

(2) One essential file in the directory is MLmodel file.

(3) This file is the single source of truth about how model should be loaded & used.

Ex:- MLmodel file created for CV model trained with fastai

```
artifact_path : classifier
flavors:
  fastai:
    data : model.fastai
    fastai_version: 2.4.1
  python_function:
    data: model.fastai
    env: conda.yaml
    loader_module: mlflow.fastai
    python_version: 3.6.12
model_uuid: e6493- - - -
run_id: - - - - -
signature :
  inputs: '[{"type": "tensor",
            "tensor_spec":
              {"dtype": "uint8", "shape": [-1, 300, 300, 3]}
          }]'
  outputs: '[{"type": "tensor",
            "tensor_spec":
              {"dtype": "float32", "shape": [-1, 2]}
            }]'
```

most imp thing to set is flavor & signature

- A flavor of the ML library with which the model was created.

- Configure the Signature

- Flavor & signature are part of MLmodel file.
- Signature serves as data contract b/w the model & the server running your model.

Two types of Signature! —

(1) Column based — used for tabular data with Pandas Dataframe as inputs
(2) tensor-based — used for n-dimensional arrays or tensors with numpy.ndarrays as inputs.

- MLmodel file is created when you register the model, the signature also is created when u register the model.
- If u want the signature to be different u need to manually log the model.
- The signature input & output are important when deploying your model.
- when u send data to deployed MLflow model that expected input & output need to match the schema as defined in signature.

Register an MLFlow model

- Model Registry makes it easy to organize and keep track of your trained model when you register a model, u store & version your model in workspace.

- 3 types of models u can register! —

(1) mlflow → model trained & tracked with MLFlow, Recommended for most scenarios.
(2) Custom → Model type with Custom standard not supported currently
(3) Triton → Model type for DL workloads Currently used for TF & Pytorch model deployment.

- A flavor of the ML library with which the model was created.

- Configure the Signature

- Flavor & signature are part of MLmodel file.
- Signature serves as data contract b/w the model & the server running your model.

Two types of Signature! —

(1) Column based — used for tabular data with Pandas Dataframe as inputs
(2) tensor-based — used for n-dimensional arrays or tensors with numpy.ndarrays as inputs.

- MLmodel file is created when you register the model, the signature also is created when u register the model.
- If u want the signature to be different u need to manually log the model.
- The signature input & output are important when deploying your model.
- when u send data to deployed MLflow model that expected input & output need to match the schema as defined in signature.

Register an MLFlow model

- Model Registry makes it easy to organize and keep track of your trained model when you register a model, u store & version your model in workspace.

- 3 types of models u can register! —

(1) mlflow → model trained & tracked with MLFlow, Recommended for most scenarios.
(2) Custom → Model type with Custom standard not supported currently
(3) Triton → Model type for DL workloads Currently used for TF & Pytorch model deployment.

Register an MLFlow Model: —

(1) To register an MLFlow model u can use Studio CLI, Python SDK

To train the model submit training script as Command job: —

```
from azure.ai.ml import Command
job = Command (
        Code = "./src,
        Command = "python train-model-Ignature.py --training-
                                                    data diabetes.csv",
        environment = ' '
        Compute = ' '
        display_name = " "
        Experiment_name = "      " )

returned_job = ml_client.Create_or_update (job)
aml_url = returned_job.studio_url
Print ("Monitor ur job", aml_url)
```

once the job is Completed the model is trained, use the job run & register the model from its output —

```
from azure.ai.ml.entities import Model
from azure.ai.ml.Constants import AssetTypes
job_name = returned_job.name
run_model = Model (
        path = f"azureml://jobs/{job_name}/outputs/ artifacts /paths/model",
        name = ' '
        description = "Model Created from Run",
        type = AssetTypes.MLFlow_Model, )

ml_client.models.create_or_update (run_model)
```

- All registered models are listed in the Models page of Azure ML.
- The registered model includes model output directory.
- When u register & log an MLflow model u can find the ~~MLflow~~ MLModel file in the artifacts of registered Model.

Create & Explore the Responsible AI dashboard for a Model in Azure ML:—

- To help you with implementing responsible AI, Azure ML offers the Responsible AI Dashboard.
- you can create & customize the Responsible AI dashboard to explore your data & model.
- whatever u use a model for your should consider the Responsible AI Principles.
- 5 principle of RAI:—
  (1) fairness & inclusiveness → treat everyone fairly
  (2) Reliability & safety → reliable, safe & consistent
  (3) privacy & security → treat data with care to ensure an individual's privacy.
  (4) Transparency → transparent in the decisions (model)
  (5) Accountability → Take accountability for decisions.

Create the Responsible AI Dashboard!—

- To help u implement RAI principles in Azure ML u can create RAI dashboard
- Azure ML has built-in components that can generate RAI insights for you.

Create RAI dashboard

- To create RAI Dashboard u need to create a pipeline by using the built in components the pipeline should!—
  (1) Starts with RAI Insights dashboard Constructor.
  (2) Include one RAI tool Component
  (3) End with Gather RAI Insights dashboard to collect all insights into one dashboard.
  (4) optionally u can also add the Gather RAI Insights Score Card at the end of your pipeline.

Explore the Responsible AI Components!

The available tools Components and Insights u Can use are! —

(1) Add explanation to RAI Insight dashboard → Interpret models by generating expln. explanation shows how much feature influence the prediction.

(2) Add Causal to RAI Insights dashboard. → use historical data to view the Causal effects of features on outcomes.

(3) Add Counterfactuals to RAI Insights dashboard. → Explore how a change in input would change the model's output

(4) Add Error Analysis to RAI Insights dashboard. → Explains the distribution of your data & identify erroneous subgroup of data.

Build & Run the pipeline to Create the Responsible AI dashboard! —

- To Create the RAI dashboard, you build a pipeline with the Component you selected.
- when you run the pipeline a RAI dashboard u generated & associated with your model.
- After you trained & registered your model in Azure ML workspace u Can Create RAI Dashboard in 3 ways! —

  ① CLI ② Python sdk ③ using Azure ML Studio no code.

using Python sdk to build & run the pipeline! —

- To generate RAI dashboard u need to! —

(1) Register the training & test datasets as MLTable data assets.
(2) Register the model.
(3) Retrieve the built in Component u want to use.
(4) Build the pipeline.
(5) Run the pipeline.

- If you want to build the pipeline using the python sdk, you first have to retrieve the Components you want to use.
- You should start the pipeline with the RAI Insights dashboard Constructor component.
- Then u can add any of the available insights, like the explanations, by retrieving the Add Explanation to RAI Insights dashboard Component.
- And finally your pipeline should end with a Gather RAI Insights dashboard Component.
- After building the pipeline u need to run it to generate the RAI dashboard.
- When the pipeline successfully completed you can select to view the RAI dashboard from pipeline overview.

Evaluate the RAI dashboard:

- When your RAI dashboard is generated, u can explore its contents in Azure ML Studio to evaluate your model.
- The tabs of each component u added to the pipeline is reflected in the dashboard. depending on the components you selected u can find following insights in ur RAI dashboard!

① Error Analysis ② Explanations ③ Counterfactuals

④ Causal Analysis

Explore Error Analysis!

- A model is expected to make mistakes (fake predictions) or error.
- With the error Analysis feature in RAI Dashboard u can review & understand how errors are distributed in your datasets.
- for ex are there specific subgroups or cohorts in your datasets for which the model makes more false prediction.
- when u include error Analysis, there are 2 types of visuals u can explore in RAI Dashboards.

(1) Error tree map! — Allows u to explore which combination of subgroups results in the model making more false predictions.

(2) Error heat map! — presents a grid overview of a model's error over the scale of one or two feature.

Explore Explanations

- when u use a model for decision making you want to understand how the model reaches a certain prediction
- In other words u want to understand how each input feature influences the model's prediction.
- various statistical technique u can use as a model explainers
- u can explain 2 types of feature importance! —

(1) Aggregate feature Importance.

(2) Individual feature "

Explore Counterfactuals

- To explore how the model's output would change based on a change in the input you can use Counterfactuals.
- what if

=

Explore Causal Analysis

- This analysis uses statistical technique to estimate the average effect of a feature on a desired prediction
- It analyses how certain interventions or treatments may result in better outcome across a population or for specific individual.

There are 3 available tabs in the RAI dashboard when including Causal Analysis' —

Aggregate Causal effects → the feature u want to change to optimize the model's prediction.

Individual Causal effects → shows individual data points and allows you to change the treatment feature to explor their influence on prediction.

Treatment policy → shows which parts of your data points benefit most from a treatment.

Deploy your Model

- After u Create an endpoint in Azure ML you Can deploy a model to that endpoint.
- To deploy your model you need to managed online endpoints, you need to Specify 4 things:

① Model assets → like the model pickle file or a registered model in Azure ML workspace.

② Scoring script → that loads the model.

③ Environment → which list all necessary packages that need to be installed on the Compute & the endpoint.

④ Compute Configuration → including the needed Compute size & scale settings to ensure u Can handle the amount & request the endpoint will receive.

All these element defined in the ~~endpoint~~ deployment. (online)

* when you deploy mlflow model to an endpoint u don't need a scoring script & environment are both automatically generated.

Blue Green Deployment

- Blue/green deployment allows for multiple model to be deployed to an endpoint. u Can decide how much traffic to forward to each deployed model.
- This way u Can switch to new version of the model without interrupting service to the Consumer.

Deploy & Consume models with Azure ML! —

- Deploy a model to an endpoint.
- When you deploy a model u can get real time or batch predictions by calling the endpoint.

Deploy a model to a managed online endpoint! —

- To consume the model you need to deploy it.
- One way to deploy it to integrate it with service that allows apps to request instant or real time predictions for individual or small set of data points.
- To make ML model available for other applications u can deploy the model to a managed online endpoint.
- To get real time prediction u can deploy a model to an endpoint
- An endpoint is an HTTPS endpoint to which you can send data and which will return a response immediately (almost)
- Any data you send to the endpoint will be served as the input for the scoring script hosted on the endpoint.

Managed online Endpoint! ——

(1) Managed online Endpoints → Azure ML manages all the underlying infrastructure.

(2) Kubernetes online Endpoints! — users manage the Kubernetes cluster which provides the necessary infrastructure.

Create an endpoint! —

```
from azure.ai.ml.entities import ManagedOnlineEndpoint
endpoint = ManagedOnlineEndpoint(
    name="endpoint-ex",
    description="online endpoint",
    auth_mode="key",
)
ml_client.begin_create_or_update(endpoint).result()
```

name & endpoint

use key for key based authentication also aml-token for token based

Deploy your MLFlow model to a Managed Online Endpoint :-

- easiest way to deploy a model to an endpoint is to use an MLFlow model & deploy it to a managed online endpoint.
- it generates scoring script & env automatically
- To deploy a MLFlow model to an endpoint
  - u need must have model files stored on a local path or with a registered model
  - In this ex we are taking model files from a local path. files are stored in local folder called Model
  - The folder must include the MLmodel file which describes how the model can be loaded & used.

To deploy (and automatically register) the model following code:—

```
from azure.ai.ml.entities import Model, ManagedOnlineDeployment
from azure.ai.ml.constants import AssetTypes.

model = Model(
        path = "./model",
        type = AssetTypes.MLFlowModel,
        description = "my sample mlflow model",
        )

blue_deployment = ManagedOnlineDeployment(
                  name = "blue",
                  endpoint_name = "endpoint-example",
                  model = model
                  instance_type = 'Standard_F4s_v2",
                  instance_count = 1,
                )

ml_client.online_deployment.begin_create_or_update(blue_deployment).
                                                    result()
```

- Since 1 model is deployed to the endpoint, it would the traffic to be 100%.
- When multiple model deployed to same endpoint it can distribute the traffic among the deployed models.

- To route traffic to a specific deployment below code.

```
endpoint.traffic = {"blue": 100}
ml_client.begin_create_or_update(endpoint).result()
```

To delete the endpoint

```
ml_client.online_endpoints.begin_delete(name = 'endpoint_example")
```

Deploy a model to a managed online endpoint:— (32)

- To deploy a model to managed online endpoint without using MLFlow model format

- To deploy a model u need
  (1) Model files stored on a local path or registered model.
  (2) A scoring script
  (3) An execution environment

The model file can be ~~store~~ logged & stored when you train a model.

Create a scoring script

- The scoring script includes 2 Function
  (1) init() — Called when service is initialized.
  (2) run() — Called when new data is submitted to the service.

- init() → is called when the deployment is created or updated to load and cache the model from the model registry.

- run() - This fun" is called for every time the endpoint is invoked to generate prediction from the input data —

```
import json
import joblib
import numpy as np
import os

# Called when deployment is created or updated
def init():
    global model
    # get the path to the registered model file & load it
    model_path = os.path.join(os.getenv('AZUREML_MODEL_DIR'),
                              'model.PKL')
    model = joblib.load(model_path)

# Called when req is received
def run(raw_data):
    # get the I/p data as numpy array
    data = np.array(json.loads(raw_data)['data'])
    # get the prediction from model
    predictions = model.predict(data)
    # return the prediction as any json serializable format.
    return predictions.tolist()
```

Create an Environment! —

- your deployment requires an execution environment in which to run the scoring script.
- you create an environment with a Docker image with Conda dependencies or with Dockerfile
- Conda.yml file

```
name: basic-env-cpu
channels:
  - conda-forge
dependencies:
  - python=3.7
  - scikit-learn
  - numpy
```

- Then to create the env run following code! —

```
from azure.ai.ml.entities import Environment
env = Environment(
    image="mcr.microsoft.com/azureml/openmpi3.1.2-ubuntu18.04",
    conda_file="./src/conda.yml",
    name="deployment-environment",
    description="Env created from docker image + Conda env")
ml_client.environments.create_or_update(env)
```

Create the Deployment

- when u have model files, scoring script & env u can create the deployment.
- To deploy a model to an endpoint u can specify the compute configuration with 2 parameters

  instance_type → VM size to use

  instance_count → No of instance to use

To deploy the model use the ManagedOnlineDeployment class & run the below code' —

```
from azure.ai.ml.entities import ManagedOnlineDeployment, CodeConfiguration

model = Model(path="./model",
blue_deployment = ManagedOnlineDeployment(
        name="blue",
        endpoint_name="endpoint_ex",
        model=model,
        environment="deployment_environment",
        code_configuration = CodeConfiguration(
                code="./src", scoring_script="score.py"),
        instance_type="Standard_DS2_V2",
        instance_count=1,
        )
ml_client.online_deployments.begin_create_or_update(blue_deployment).
                result()
```

- u Can deploy multiple model to an endpoint. / Similar to discussed above like MLFlow.
- Also delete the endpoint

Test managed Online Endpoints!

- After deploying a real time service. u Can Consume it from client applications to predict labels for new data Cases.
- u Can list all endpoints in the Azure ML studio by navigating to Endpoint page. In Real time endpoint tab all endpoint are showes.
- also u Can use studio to test endpoint

Use the Azure ML Python SDK! —

- for testing u Can use Azure ML Python SDK to invoke an endpoint
- typically used data to deployed model in json format — ..

json

```
{ "data": [
    [01, 0.3, 8.4, 6.2],
    [06, 0.9, 8.1, 78],
  ]}
```

The Response from the deployed model is a JSON Collection with a prediction for each case that was submitted in the data.

The below code invokes an endpoint & displays the Response—

```
response = ml_client.online_endpoints.invoke(
    endpoint_name= online_endpoint_name,
    deployment_name= "blue"
    request_file= "sample-data.json",
    )

if response[1]=='1':
    Print("yes")
else:
    print("NO")
```

Deploy a model to a batch endpoint

- long running tasks that deal with large amounts of data are performed as batch operations.
- In ML batch Inferencing is used to asynchronously apply a predictive model to multiple cases & write the results to a file or database.

Understand and Create batch endpoints —

- To get a model to generate batch predictions, u can deploy the model to a batch endpoint.
- To get batch predictions you can deploy a model to an Endpoint.
- whenever the endpoint is invoked a batch scoring job is submitted to the Azure ML workspace

Create a batch endpoint: —

- To create batch endpoint u will use BatchEndpoint class.

```
endpoint = BatchEndpoint(
        name= "endpoint-ex",
        description = "A batch endpoint",
        )

mlClient.batch_endpoints.begin_create_or_update(endpoint)
```

Deploy a model to a batch Endpoint: —

Use Compute clusters for batch deployments: —

To Can Create Compute cluster -- u Can use AmlCompute class

```
from azure.ai.ml.entities import AmlCompute
cpu_cluster= AmlCompute (
        name= "aml-cluster",
        type = 'amlcompute'
        size = "STANDARD_DS11_V2",
        min_instances =0,
        max_instances= 4,
        idle_time_before_scale_down= 120
        tier= "Dedicated",
    )
cpu_cluster= ml_client.compute.begin_create_or_update (cpu_cluster)
```

Deploy your MLFlow model to a batch Endpoint! —

- To deploy an MLFlow model u need to have created an endpoint.

Register an MLFlow model

- To register an MLFlow model u will use Model class, while specifying the model type to be MLFLOW_MODEL.
- To register the model with the Python SDK! — following code.

```
from azure.ai.ml.entities import Model
from azure.ai.ml.constants import AssetTypes
model_name= 'mlflow-model'
model= ml_client.models.create_or_update (
        Model (name= model_name, path='./model', type =
                AssetTypes.MLFLOW_MODEL)
    )
```

- In this we are taking the model files from local path. The files are all stored in a local folder called model. The folder must include the MLmodel file, which describes how the model can be loaded and used.

Deploy an MLFlow model to an Endpoint! ——

- When u deploy a model, u need to specify how you want the batch scoring job to behave.

To deploy an MLFlow model to a batch endpoint! ——

```
from azure.ai.ml.entities import BatchDeployment, BatchRetrySettings
from azure.ai.ml.constants import BatchDeploymentOutputAction

deployment = BatchDeployment(
    name="forecast-mlflow",
    description="A sales forecaster",
    endpoint_name=endpoint.name,
    model=model,
    compute='aml-cluster',
    instance_count=2,
    max_concurrency_per_instance=2,
    mini_batch_size=2,
    output_action=BatchDeploymentOutputAction.APPEND_ROW,
    output_file_name="prediction.csv",
    retry_settings=BatchRetrySettings(max_retries=3, timeout=300),
    logging_level="info",
    )
ml_client.batch_deployments.begin_create_or_update(deployment)
```

- Deploy a Custom model to a batch endpoint
  - Create the Scoring Script: → init()
                               → run()
  - Create an Environment
  - Configure & Create the Deployment! ——

```
from azure.ai.ml.entities import BatchDeployment, BatchRetrySettings
from azure.ai.ml.constants import BatchDeploymentOutputAction.

deployment = BatchDeployment(
    name= "forecast-mlflow",
    description= "A sales forecaster",
    endpoint_name= endpoint.name,
    model = model,
    compute = "aml-cluster",
    code_path = "./code",
    scoring_script = "score.py",
    environment=env,
    instance_count=2
    max_concurrency_per_instance=2,
    mini_batch_size=2,
    output_action = BatchDeploymentOutputAction.APPEND_ROW,
    output_file_name= 'prediction.csv',
    retry_settings = BatchRetrySettings(error_retries=3, timeout=300),
    logging_level = "info",
)
ml_client.batch_deployments.begin_create_or_update(deployment)
```

Invoke and Troubleshoot batch endpoints! —

- when u invoke a batch endpoint u trigger an Azure ML pipeline job.

Trigger the batch scoring job! —

- To prepare data for batch predictions u can register a folder as a data asset in Azure ML workspace.
- u can then use the registered data asset as input when invoking the batch endpoint with Python sdk.

```
from azure.ai.ml import Input
from azure.ai.ml.constants import AssetTypes

input = Input(type=AssetTypes.URI_FOLDER, path="azureml:new-
                                                data:1")

job = ml_client.batch_endpoints.invoke(
    endpoint_name=endpoint.name,
    input=input)
```

- The predictions will be stored in the default datastore.

Troubleshoot a batch scoring job:—

- The batch scoring job runs as a pipeline job.
- If u want to troubleshoot the pipeline job you can review the detail & output and logs of the pipeline job itself.