·DATAXDAYS

azureml : a python SDK to train, package and deploy ML models

Another story of MLOps…

publicis sapient BY ENGINEERS *at* PUBLICIS SAPIENT

To get value, algorithms must live outside the laptop !

Microsoft AI MVP since 2018
Meetup organizer and speaker
Formed as a Data Miner
(an old buzzword…)
Senior Data Consultant @AZEO

AZEO talents & technology

What side are you on ? DATA SCIENTIST or DATA ENGINEER ?

Remember the wall between devs and ops. Should we build a new one between the data scientists and the data engineers?

We understand each other better when we use common tools...

NEVERMIND ! LET'S MERGE !

https://www.linkedin.com/in/paul-peton-datascience

https://github.com/methodidacte/azureml

AGENDA

Paul Péton
Tuesday June 2 at 17:30

Azureml : Python SDK to train, package and deploy models (another story of MLOps)

Intro : **Why and how to industrialize ML**

Partie 01 : **Azure Machine Learning “new” Studio**

Partie 02 : **The MLOps approach**

Partie 03 : **The azureml python SDK**

Partie 04 : **DEMO - 3 workflows, gridsearch and pipelines**

Partie 05 : **… if we have enough time !**

WHY INDUSTRIALIZE ?

BECAUSE IT IS RETURN ON INVESTISSEMENT !

Check versioning

Have backups

Plan jobs execution

Monitor services

Integrate organizational security

Deploy on light terminals (edge)

WHAT DO WE NEED FOR PRODUCTION ?

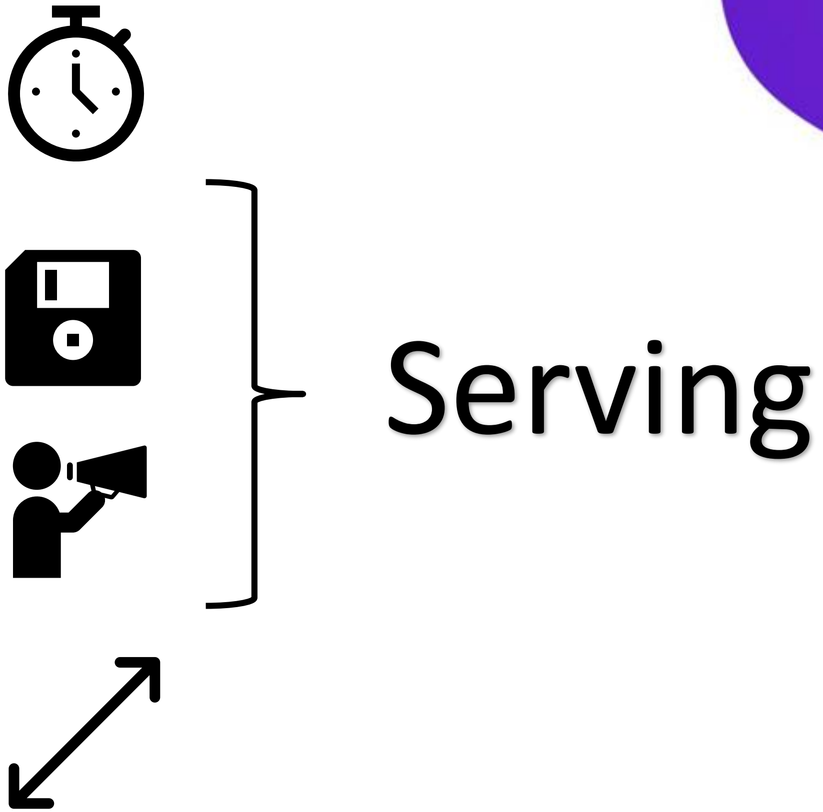
MY WISH LIST (try to do yours)

A scheduler to plan
Data cleansing
Training / re-training of the model
The forecast calculation (in batch mode)

A storage system to archive models
Per algorithm, per version, per training dataset
In a serialized (not proprietary) binary format

A tool for exposing the model
Via REST API
Secure access (key, token, LDAP…)

Resources that can be deployed at scale
With the help of the containers
In the (public) Cloud



WHAT IS AZURE MACHINE LEARNING ?

Set of Azure Cloud Services

+

Python SDK & R

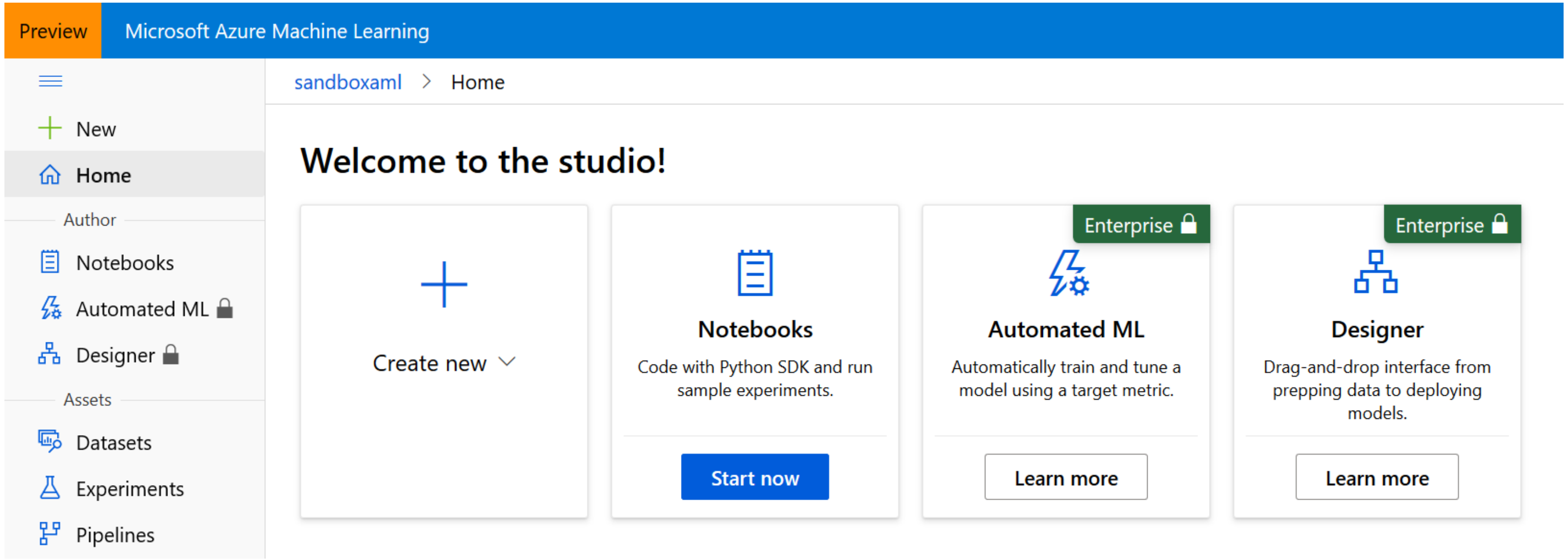
That enables you to:

- ✓ Prepare Data
- ✓ Build Models
- ✓ Train Models
- ✓ Manage Models
- ✓ Track Experiments
- ✓ Deploy Models

AZURE MACHINE LEARNING « new studio »

The studio is the User Interface on the URL : https://ml.azure.com/

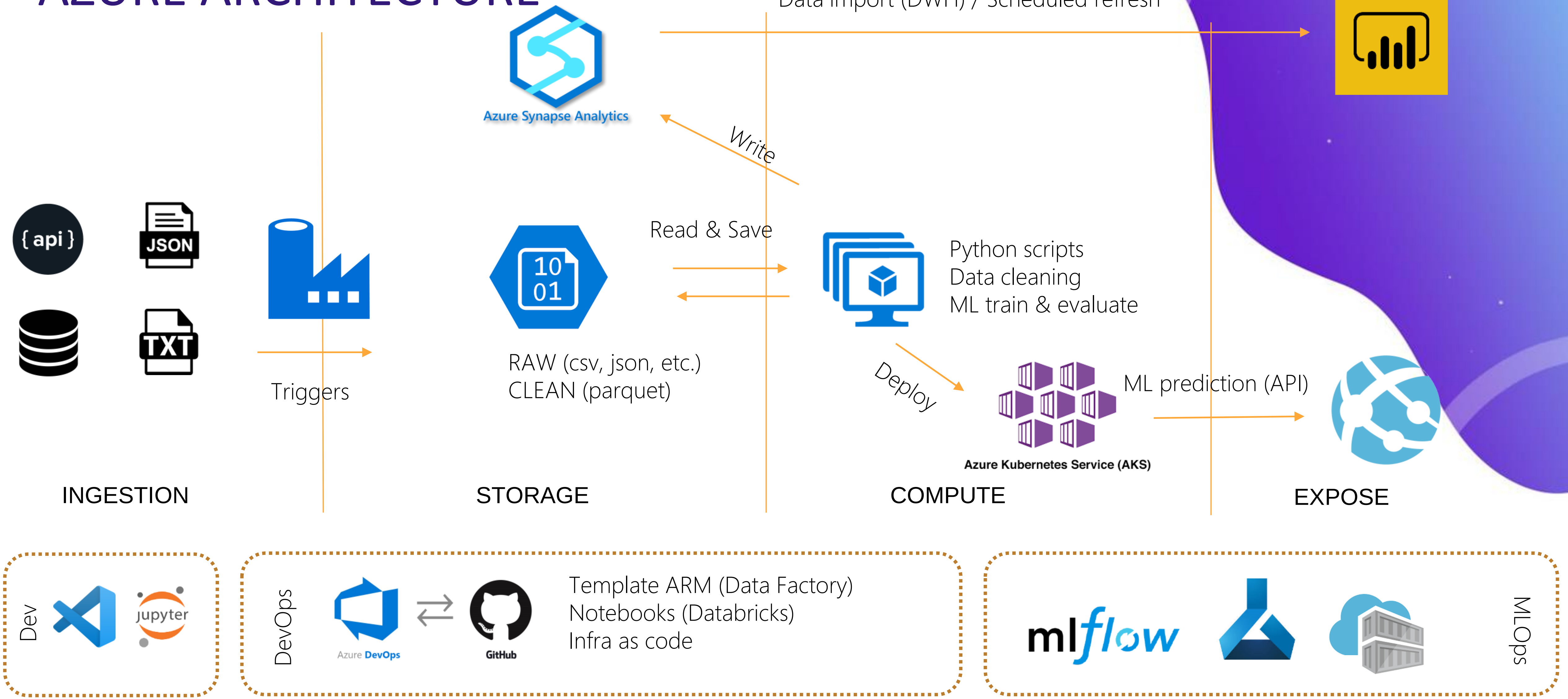
But we can do better with some code !



Licensing : standard or enterprise (more UI features)

Tarification : pay as long as the virtual machines are running.

AZURE ARCHITECTURE



MLOPS : quick and concrete definition

DevOps

Code reproducibility

Code testing

App deployment

MLOps

Model reproducibility

Model validation

Model deployment

Model retraining

DEMO : DIABETES USE CASE

Modeling Y value by the others (regression)

Authors of dataset : Bradley Efron, Trevor Hastie, Iain Johnstone and Robert Tibshirani

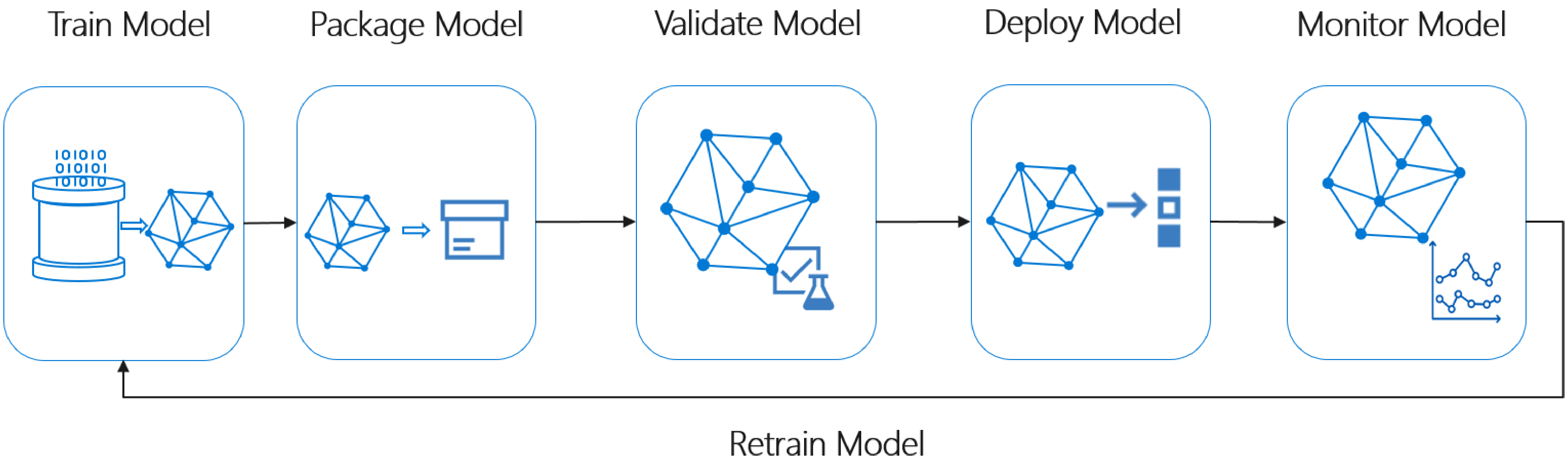
	AGE	SEX	BMI	BP	S1	S2	S3	S4	S5	S6	Y
0	59	2	32.1	101.0	157	93.2	38.0	4.0	4.8598	87	151
1	48	1	21.6	87.0	183	103.2	70.0	3.0	3.8918	69	75
2	72	2	30.5	93.0	156	93.6	41.0	4.0	4.6728	85	141
3	24	1	25.3	84.0	198	131.4	40.0	5.0	4.8903	89	206
4	50	1	23.0	101.0	192	125.4	52.0	4.0	4.2905	80	135

Ten baseline variables, age, sex, body mass index, average blood pressure, and six blood serum measurements were obtained for each of **n = 442** diabetes patients, as well as the response of interest, a quantitative measure of disease progression one year after baseline.

THE DATA SCIENCE WORKFLOW

You know the theory, let's apply !

Each part of the process must be code.



Train : could be done with Automated ML or hyperparameter tuning

The Python SDK : azureml

R version available (not tested)

WHAT IS THE AZUREML PYTHON SDK ?

A set of libraries that facilitate access to :

- **Azure** components (Virtual Machine, Cluster, Image…)
- Runtime components (ServiceBus using HTTP, Batch, Monitor…)

Official GitHub repository :
https://github.com/Azure/azure-sdk-for-python

Full list of available packages and their latest version :
https://docs.microsoft.com/fr-fr/python/api/overview/azure/?view=azure-python

Installation :

```
!pip install --upgrade azureml-sdk
```

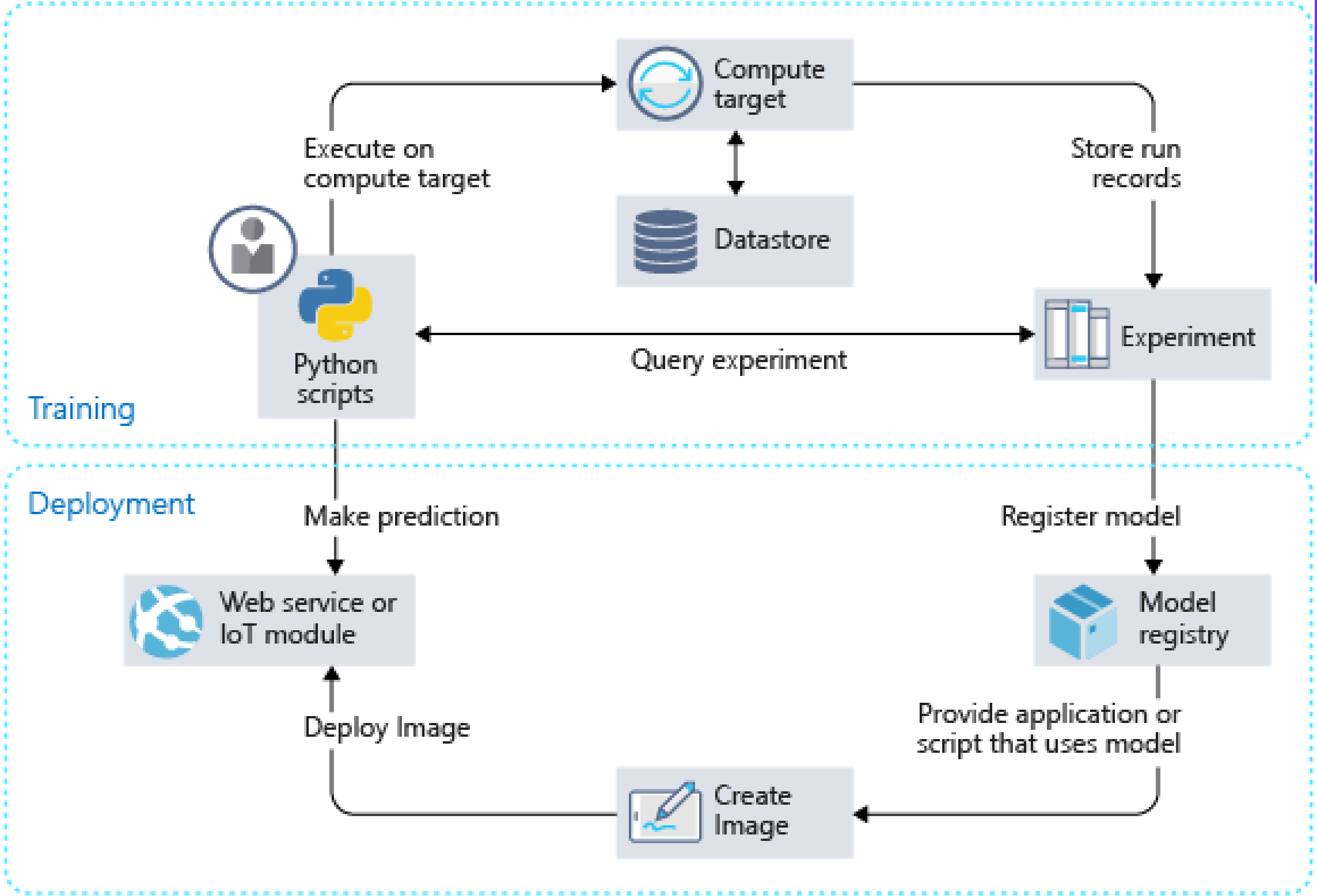
Or clone th GtiHub reporsitory :

```
git clone git://github.com/Azure/azure-sdk-for-python.git
cd azure-sdk-for-python
python setup.py install
```

THE MAIN OBJECTS

Inside the workspace (your Azure resource)

- ❑ Datastore & Dataset
- ❑ Compute target
- ❑ Experiment
  - ❑ Pipeline
  - ❑ Run
- ❑ Model
  - ❑ Environment
  - ❑ Estimator
- ❑ Inference
- ❑ Endpoint



DATASTORE : supported storages

Azure file systems or managed databases

- ❑ Azure Blob Container
- ❑ Azure File Share
- ❑ Azure Data Lake
- ❑ Azure Data Lake Gen2

- ❑ Azure SQL Database
- ❑ Azure Database for PostgreSQL
- ❑ Azure Database for MySQL

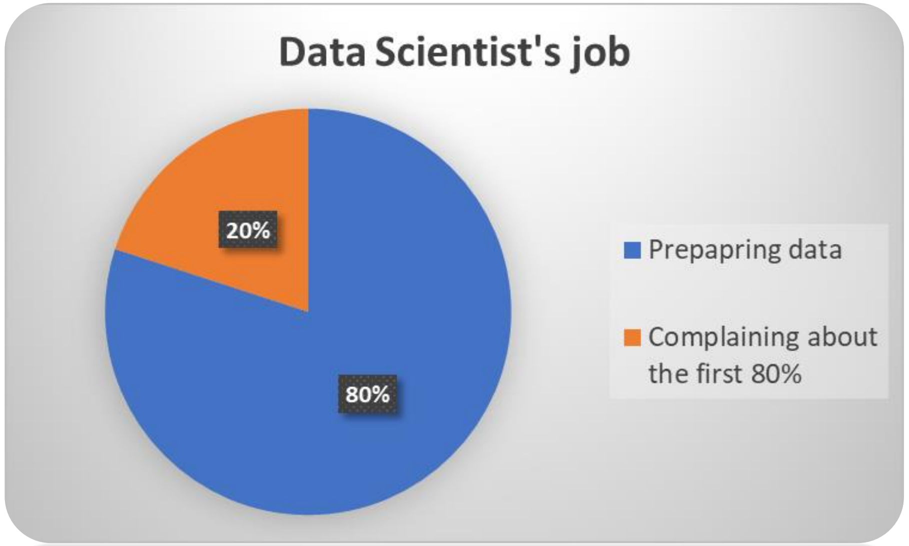
- ❑ Databricks File System

```
from azureml.core import Dataset

dataset = Dataset.Tabular.from_delimited_files(
          path = [(datastore, 'tabular/iris.csv')])

# Convert the Dataset object to a (in-memory) pandas dataframe
df = dataset.to_pandas_dataframe()
```

FIND THE BEST MODEL WITH BRUT FORCE

AutomatedML *vs* Data Scientist, who's best ?

- ❑ Parallel compute
- ❑ Stop criteria
- ❑ *It's up to you to prepare the datas !*




```
from azureml.train.automl import AutoMLConfig
automl_settings = {
    "iteration_timeout_minutes" : 10,
    "iterations" : 2,
    "primary_metric" : 'spearman_correlation',
    "preprocess" : True,
    "verbosity" : logging.INFO,
    "n_cross_validations": 5
}

automl_config = AutoMLConfig(task = 'regression',
          debug_log = 'automated_ml_errors.log',
          path = train_model_folder,
          compute_target = aml_compute,
          run_configuration = aml_run_config,
          data_script = train_model_folder+"/get_data.py",
          **automl_settings)
```

WHERE TO DEPLOY IN AZURE ?

Everywhere we can use a Docker image

- ❑ Azure Web App
- ❑ Azure Function : serverless, but 5 minutes maximum
- ❑ Azure Container Instance : DEV scenario, authentication by key or token
- ❑ Azure Kubernetes Services : PROD scenario, 12 cores minimum, AD authentication

THREE COMPLETE WORKFLOWS

Notebooks available (github.com/methodidacte/azureml)

FIRST AZUREML WORKFLOW

Default model

Initialize workspace

```
[2]: # Load workspace configuration from the config.json file in the current folder.
ws = Workspace.from_config()
```

Create an experiment

```
experiment_name = 'diabetes_exp'

from azureml.core import Experiment
exp = Experiment(workspace=ws, name=experiment_name)

exp
```

Upload dataset

```
from azureml.core import Dataset

dataset = Dataset.get_by_name(ws, name='diabetes')
```

```
diabetes = dataset.to_pandas_dataframe().drop("Path", axis=1)
```

Intialize the workspace

- To interact with Azure ML Service

Create an experiment

- Which will « run »

Create a datastore / upload a dataset

- Or upload an Open Dataset

Train the model

- Best model can be found with Automated ML

Register the model

- With **SCIKITLEARN, ONNX or TensorFlow framework**

Deploy the model

- Create a standard service

[18]:
```python
print(service.state)
```

Healthy

[19]:
```python
print(service.scoring_uri)
```

http://1859b69e-d0fb-4aee-a65f-46b0e2452e4a.westus2.azurecontainer.io/score

[20]:
```python
print(service.swagger_uri)
```

http://1859b69e-d0fb-4aee-a65f-46b0e2452e4a.westus2.azurecontainer.io/swagger.json

Test the service

[21]:
```python
import json

input_payload = json.dumps({
    'data': [
        [59, 2, 32.1, 101.0, 157, 93.2, 38.0, 4.0, 4.8598, 87],
        [69, 2, 32.1, 101.0, 157, 93.2, 38.0, 4.0, 4.8598, 87]
    ],
    'method': 'predict'  # If you have a classification model, you can get probabilities by changing this to 'predict_p
})

output = service.run(input_payload)

print(output)
```

{'predict': [[210.74209144739257], [212.1189695974167]]}

CLASSICAL WORKFLOW

Custom model

…

Train the model

- Best model can be found with Automated ML

Register the model

- With CUSTOM framework

Define a scoring script

- Called score.py

Define the environment

- Packages needed

Create an inference configuration

- Profile the model if needed

Deploy the model

- Locally
- ACI
- AKS

Define the (inference) environement

```
from azureml.core import Environment
from azureml.core.conda_dependencies import CondaDependencies


environment = Environment('diabetes-deployment-env')
environment.python.conda_dependencies = CondaDependencies.create(pip_packages=[
    'azureml-defaults',
    'inference-schema[numpy-support]',
    'joblib',
    'numpy',
    'sklearn'
])
```

Define a inference configuration

```
from azureml.core.model import InferenceConfig


inference_config = InferenceConfig(entry_script='score.py', environment
```

```
from azureml.core import Webservice
from azureml.core.webservice import AciWebservice
from azureml.exceptions import WebserviceException


service_name = 'diabetes-custom-service'

# Remove any existing service under the same name.
try:
    Webservice(ws, service_name).delete()
except WebserviceException:
    pass

aci_config = AciWebservice.deploy_configuration(cpu_cores=1, memory_gb=1, auth_enabled=True)

service = Model.deploy(workspace=ws,
                       name=service_name,
                       models=[model],
                       inference_config=inference_config,
                       deployment_config=aci_config)

service.wait_for_deployment(show_output=True)
```

```
Running..................................
Succeeded
ACI service creation operation finished, operation "Succeeded"
```

PROFILING OF THE INFERENCE CONFIGURATION

Find the best values for vCPU and RAM

Give a sample of data

```
recommended_memory=0.5,
recommended_cpu=0.5
```

It takes (compute) time but you will optimize your inference compute.

```python
from datetime import datetime

# if cpu and memory_in_gb parameters are not provided
# the model will be profiled on default configuration of
# 1CPU and 0,5GB memory
profile = Model.profile(ws,
            'sklearn-%s' % datetime.now().strftime('%m%d%Y-%H%M%S'),
            [model],
            inference_config,
            input_dataset=sample_request_data,
            cpu=1.0,
            memory_in_gb=0.5)

profile.wait_for_completion(True)
details = profile.get_details()
```

```
Runnin
g...........................................................................................................
..........
Succeeded
```

```python
profile
```


```
ModelProfile(workspace=Workspace.create(name='eacbmlservicews', subscription_id='f80606e5-788f-4dc3-a9ea-2eb9a783608
2', resource_group='adlsgen2'), name=sklearn-06012020-071100, create_operation_id=1cba6f63-a6b0-47fc-9344-6e20473e6e1
1, image_id=None, description=None, created_time=2020-06-01 07:11:01.326865+00:00, id=621904fc-c1b3-4433-8693-28c51bf0
6095, requested_cpu=1.0, requested_memory_in_gb=0.5, requested_queries_per_second=0, input_dataset_id=7b76435d-c354-46
29-8141-0680b8f6d4b3, state=Succeeded, model_ids=['diabetes_regression_model:6'], max_utilized_memory=0.03806481066666
6664, max_utilized_cpu=0.003, measured_queries_per_second=218.9621195533173, environment={'name': 'my-sklearn-environm
ent', 'version': 'Autosave_2020-06-01T07:11:02Z_8611b891', 'python': {'interpreterPath': 'python', 'userManagedDepende
ncies': False, 'condaDependencies': {'channels': ['anaconda', 'conda-forge'], 'dependencies': ['python=3.6.2', {'pip':
['azureml-defaults', 'inference-schema[numpy-support]', 'joblib', 'numpy', 'scikit-learn']}], 'name': 'azureml_0bc8cb4
f185b37c328a427a1e38cbb9b'}, 'baseCondaEnvironment': None}, 'environmentVariables': {'EXAMPLE_ENV_VAR': 'EXAMPLE_VALU
E'}, 'docker': {'baseImage': 'mcr.microsoft.com/azureml/base:intelmpi2018.3-ubuntu16.04', 'baseDockerfile': None, 'bas
eImageRegistry': {'address': None, 'username': None, 'password': None}, 'enabled': False, 'arguments': []}, 'spark':
{'repositories': [], 'packages': [], 'precachePackages': True}, 'inferencingStackVersion': None}, error=None, error_lo
gs_url=None, total_queries=100.0, success_queries=100.0, success_rate=100.0, average_latency_in_ms=4.566999999999999,
latency_percentile_50_in_ms=3.39, latency_percentile_90_in_ms=5.35, latency_percentile_95_in_ms=9.07, latency_percenti
le_99_in_ms=18.06, latency_percentile_999_in_ms=18.06, recommended_memory=0.5, recommended_cpu=0.5)
```

JUPYTER WIDGET

Allows to follow the run of the experiment in the notebook

Jupyter widget

Watch the progress of the run with a Jupyter widget. Like the run submission, the widget is asynchronous and provides live updates every 10-15 seconds until the job completes.

Entrée [13]:

```
from azureml.widgets import RunDetails
RunDetails(run).show()
```

Run Properties

Status	Preparing
Start Time	14/04/2020 07:56:04
Duration	0:00:42
Run Id	sklearn-mnist_1586843745_876530d6
Arguments	N/A

Output Logs azureml-logs/20_image_build_log.txt ☑ Auto-switch

```
 ---> 06556a281e2a
Step 3/15 : RUN mkdir -p $HOME/.cache
 ---> Running in 7a8a53c6937a
Removing intermediate container 7a8a53c6937a
 ---> 38a6055afb2e
Step 4/15 : WORKDIR /
 ---> Running in d7fcad727a99
Removing intermediate container d7fcad727a99
 ---> 779b4dbd7050
Step 5/15 : COPY azureml-environment-setup/99brokenproxy /etc/apt/apt.conf.d/
```

Click here to see the run in Azure Machine Learning studio

BIG DATA WORKFLOW

Remote train

Intialize the workspace
- To interact with Azure ML Service

Create an experiment
- Which will « run »

Create a datastore / upload a dataset
- Or upload an Open Dataset

Create a compute target
- Cluster of VM

Train the model
- Best model can be found with Automated ML

Define the compute target environment
- Create a YAML file

Create an estimator
- Configuration of the run

Monitor the run

…

```
[25]: # Set up the (compute target) environnement

from azureml.core import Environment
from azureml.core.conda_dependencies import CondaDependencies


env = Environment("diabetes_remote_env")

env.docker.enabled = True
env.python.conda_dependencies = CondaDependencies.create(conda_packages=['scikit-learn',
                                                                          'pandas',
                                                                          'numpy',
                                                                          'joblib',
                                                                          'matplotlib'
                                                                         ])
env.python.conda_dependencies.add_pip_package("inference-schema[numpy-support]")

env.python.conda_dependencies.save_to_file(".", "diabetes_env.yml")
```

```
[25]: 'diabetes_env.yml'
```

```
[26]: from azureml.train.estimator import Estimator

script_params = {
    '--regularization': 0.5
}

estimator = Estimator(source_directory=script_folder,
                      script_params=script_params,
                      compute_target=cpu_cluster_name,
                      environment_definition=env,
                      entry_script='train.py')
```

```
[27]: run = exp.submit(config=estimator)
run
```

MONITOR THE RUN

Run 7 Completed

Refresh Resubmit Cancel

Details | **Metrics** | Images | Child runs | Outputs + logs | Snapshot | Raw JSON | Explanations (preview)

Select a metric to see a visualization or table of the data.

- [x] score
- [x] regularization strength

View as: (•) Chart ( ) Table

0.4523830... score

0.5 regularization str...

```
# Output the Mean Squared Error to the notebook and to the run
print('Mean Squared Error is', mean_squared_error(data['test']['y'], preds))
run.log('mse', mean_squared_error(data['test']['y'], preds))

# Save the model to the outputs directory for capture
model_file_name = 'outputs/model.pkl'

joblib.dump(value = regression_model, filename = model_file_name)

# upload the model file explicitly into artifacts
run.upload_file(name = model_file_name, path_or_stream = model_file_name)
```

IF YOU LIKE IT, USE mlflow

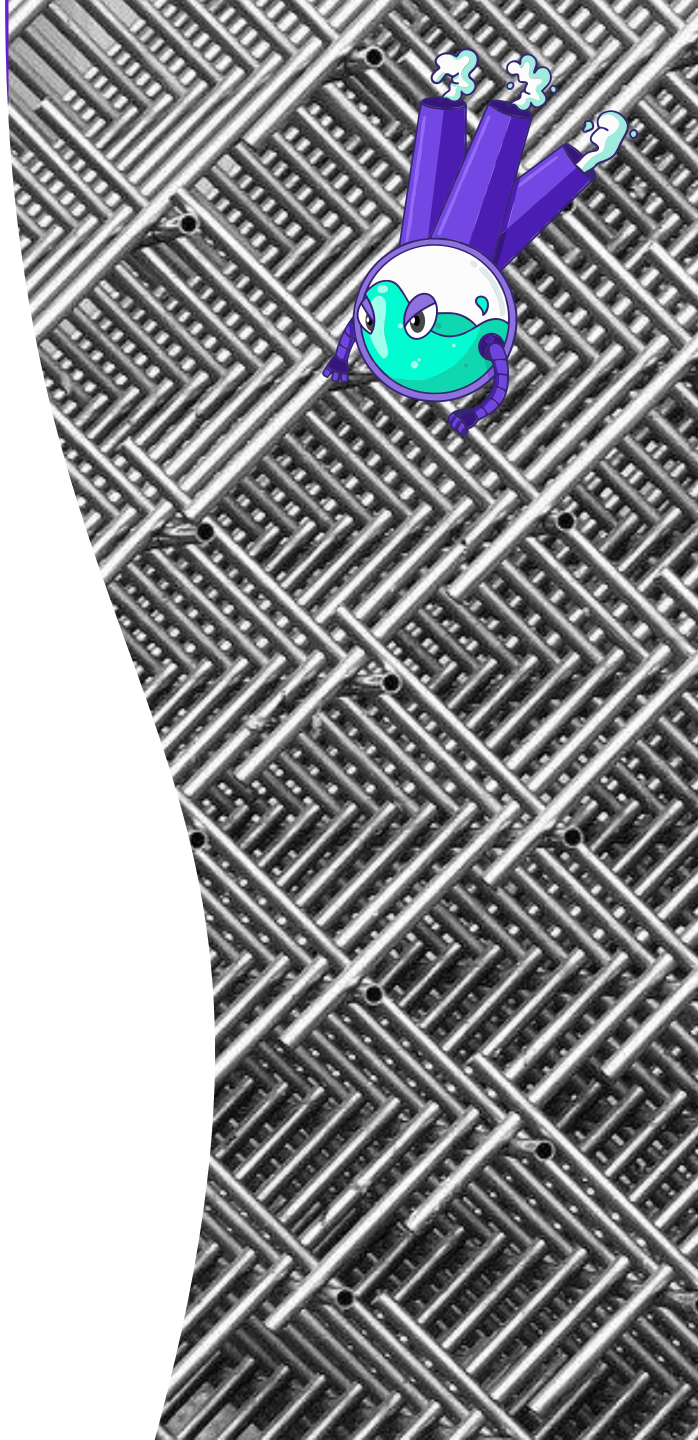
```
import mlflow
from azureml.core import Workspace

ws = Workspace.from_config()

mlflow.set_tracking_uri(ws.get_mlflow_tracking_uri())
```

HYPERDRIVE : GRIDSEARCH in azureml

Tune hyperparameters for your model

Define the parameter search space
Specify a primary metric to optimize
Specify early termination criteria for poorly performing runs
Allocate resources for hyperparameter tuning
Launch an experiment with the above configuration
Visualize the training runs
Select the best performing configuration for your model

```python
from azureml.train.hyperdrive import *

# define hyperparameter sampling space
ps = RandomParameterSampling(
     {
         '--regularization': choice(0.01, 0.1, 0.5, 1.0),
     }
)

# define early termination policy
early_termination_policy = BanditPolicy(slack_factor = 0.15, evaluation_interval=10)

# configure the run
hyperdrive_run_config = HyperDriveConfig(estimator = estimator,
                                         hyperparameter_sampling = ps,
                                         policy = early_termination_policy,
                                         primary_metric_name = "score",
                                         primary_metric_goal = PrimaryMetricGoal.MAXIMIZE,
                                         max_total_runs = 20,
                                         max_concurrent_runs = 2)
```



jupyter diabetes_regression_compute_hyperdrive Dernière Sauvegarde : il y a 20 minutes (auto-sauvegardé)

File Edit View Insert Cell Kernel Widgets Help Fiable Python 3.6 - AzureML

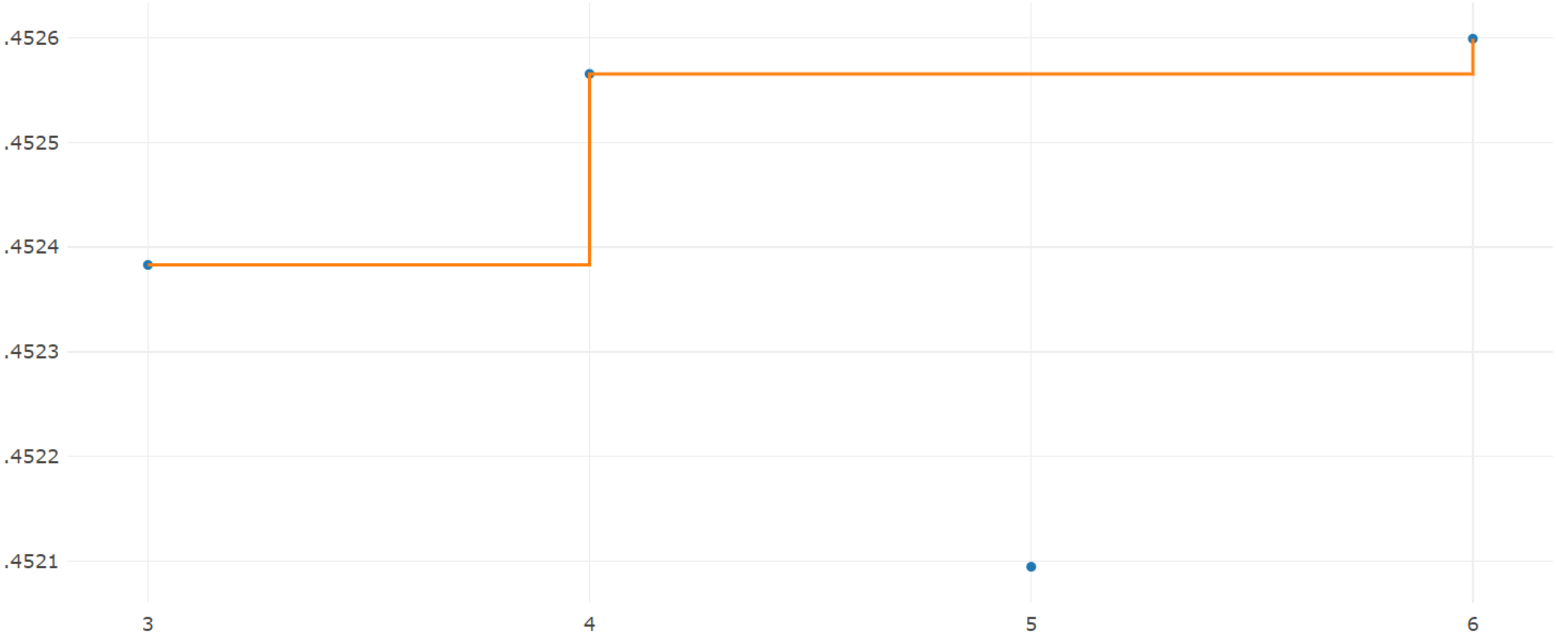
Exécuter Code nbdiff

Completed (4)

Run	Best Metric*	Status	Started	Duration	Run Id
6	0.45259922	Completed	Jun 1, 2020 8:59 AM	0:02:02	HD_c19b511d-c899-4ef2-9f08-e3c6f5f6e7dd_3
4	0.4525656	Completed	Jun 1, 2020 8:56 AM	0:01:54	HD_c19b511d-c899-4ef2-9f08-e3c6f5f6e7dd_1
3	0.45238303	Completed	Jun 1, 2020 8:57 AM	0:02:41	HD_c19b511d-c899-4ef2-9f08-e3c6f5f6e7dd_0
5	0.45209455	Completed	Jun 1, 2020 8:58 AM	0:01:22	HD_c19b511d-c899-4ef2-9f08-e3c6f5f6e7dd_2

* The best metric field is obtained from the min/max of primary metric achieved by a run

HyperDrive Run Primary Metric : score



2D Scatter Chart

```python
best_run = hd_run.get_best_run_by_primary_metric()
best_run_metrics = best_run.get_metrics()
parameter_values = best_run.get_details()
parameter_values['runDefinition']
```

•••

```python
parameter_values = best_run.get_details()['runDefinition']['arguments']
parameter_values
```

```
['--regularization', '0.01']
```

```python
print('Best Run Id: ', best_run.id)
print('\n Score:', best_run_metrics['score'])
print('\n regularization: ',parameter_values[1])
```

```
Best Run Id:  HD_be49cad5-8bf0-40e8-9f73-b27a14510b9f_1

 Score: 0.45259921776197887

 regularization:  0.01
```

Register the BEST model

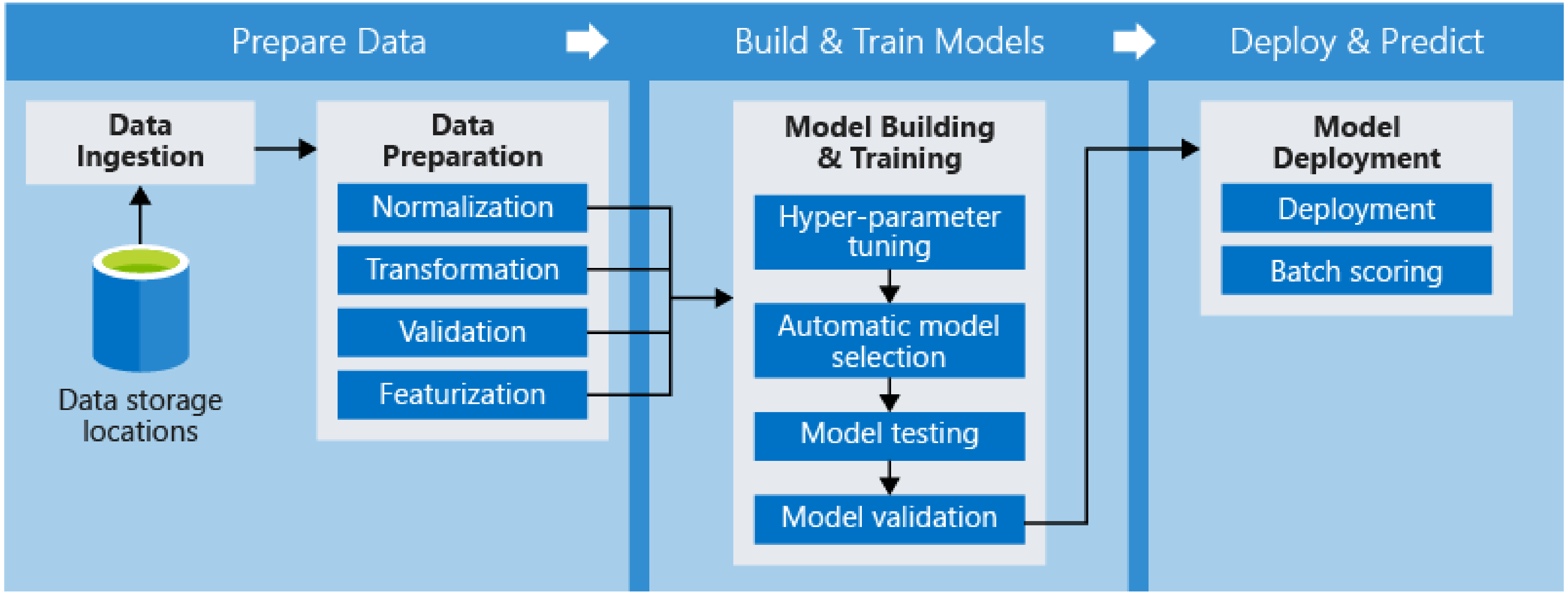
```python
print(best_run.get_file_names())
```

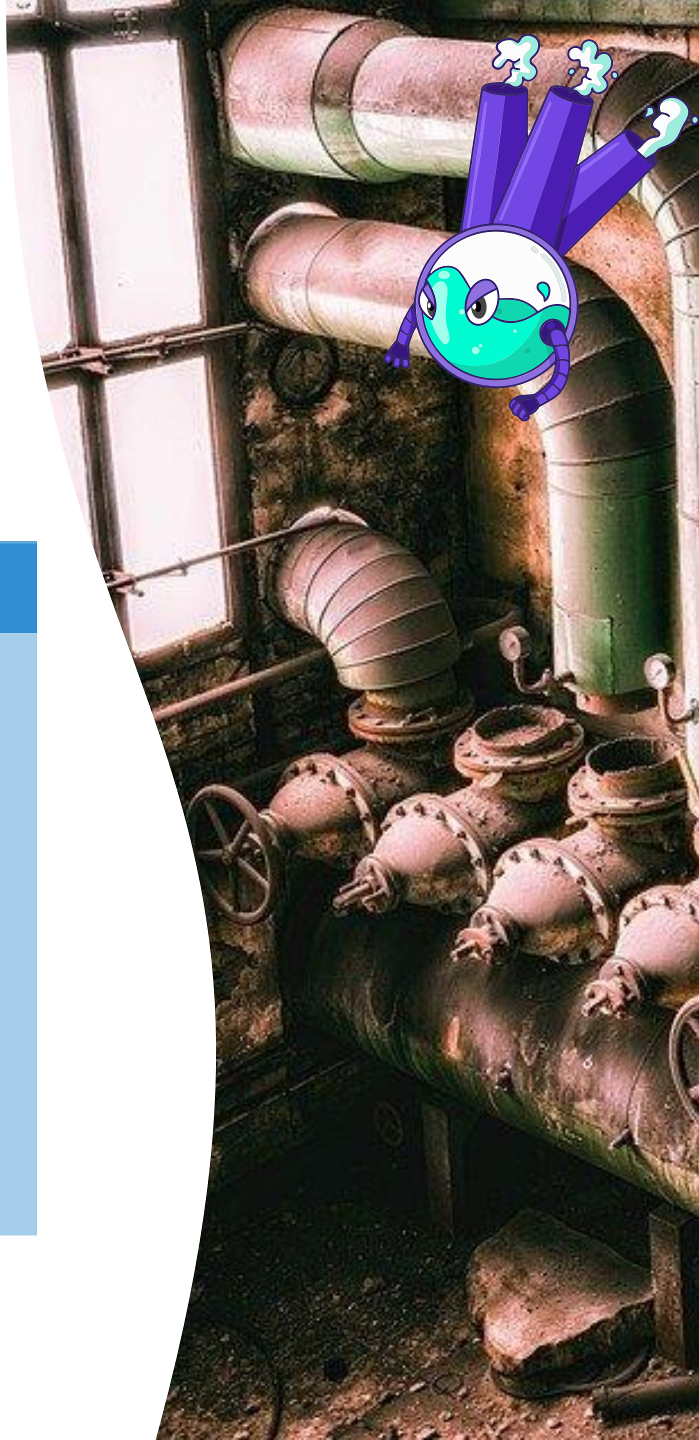
```
['azureml-logs/55_azureml-execution-tvmps_09d7d0d9b617f765ff09ce467e41ed0dc9207fb726d1c4feddce8efadb0ae365_d.txt', 'azureml-logs/65_job_pre
p-tvmps_09d7d0d9b617f765ff09ce467e41ed0dc9207fb726d1c4feddce8efadb0ae365_d.txt', 'azureml-logs/70_driver_log.txt', 'azureml-logs/75_job_pos
t-tvmps_09d7d0d9b617f765ff09ce467e41ed0dc9207fb726d1c4feddce8efadb0ae365_d.txt', 'azureml-logs/process_info.json', 'azureml-logs/process_st
atus.json', 'logs/azureml/109_azureml.log', 'logs/azureml/job_prep_azureml.log', 'logs/azureml/job_release_azureml.log', 'outputs/diabetes_
HD_remote_model.pkl']
```

```python
# register model
model = best_run.register_model(model_name='diabetes_HD_best_model', model_path='outputs/diabetes_HD_remote_model.pkl')
```

WORK WITH PIPELINES

Several ways to do DS but the same sequence of steps





A PIPELINE EXAMPLE

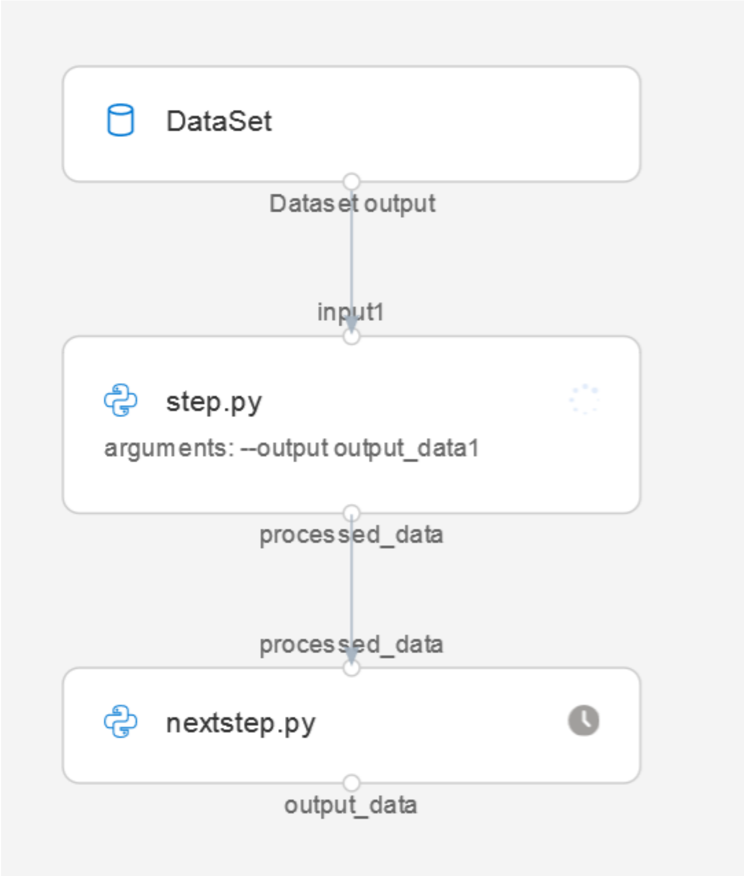
Define the steps and sequence them

+ use arguments to specify the process

Inputs and outputs can have different formats

```
[60]: from azureml.pipeline.steps import PythonScriptStep
      from azureml.pipeline.core.graph import PipelineParameter

      step = PythonScriptStep(
          script_name="step.py",
          arguments=["--output", processed_data],
          inputs=[input_named],
          outputs=[processed_data],
          compute_target=compute_target,
          source_directory="scripts"
          ,allow_reuse=True
      )

      nextstep = PythonScriptStep(
          script_name="nextstep.py",
          arguments=["--output_data", output_data], #"--input_data", processed_data,
          inputs=[processed_data.parse_delimited_files(file_extension=None)],
          outputs=[output_data],
          compute_target=compute_target,
          source_directory="scripts"
          ,allow_reuse=True
      )

      steps = [ step, nextstep ]
```

```
[31]: pipeline = Pipeline(workspace=ws, steps=[step])

      pipeline_run = experiment.submit(pipeline)
      pipeline_run.wait_for_completion()
```



SCHEDULE A PIPELINE

Pipeline is the only available for scheduling

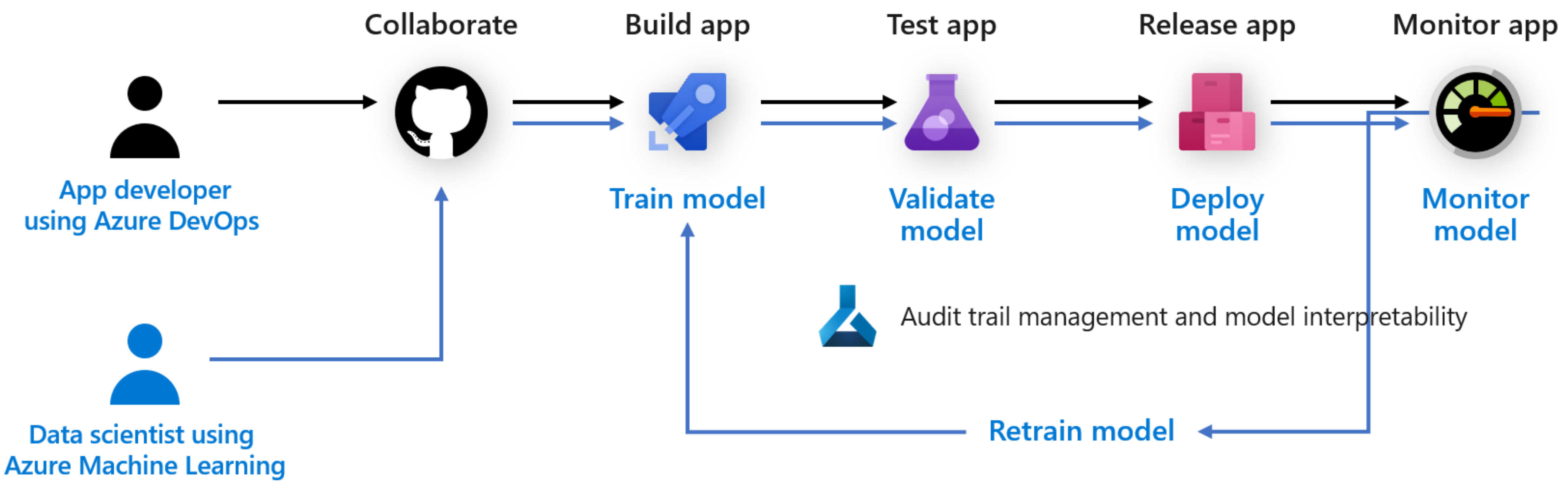
+ API endpoint to launch the pipeline

```
from azureml.pipeline.core import Pipeline, PublishedPipeline
published_pipelines = PublishedPipeline.list(ws)
for published_pipeline in  published_pipelines:
    print(f"{published_pipeline.name},'{published_pipeline.id}'")

from azureml.pipeline.core import Schedule, ScheduleRecurrence
recurrence = ScheduleRecurrence(frequency="Minute", interval=15)
recurring_schedule = Schedule.create(ws, name="MyRecurringSchedule",
                            description="Based on time",
                            pipeline_id=pipeline_id,
                            experiment_name=experiment_name,
                            recurrence=recurrence)

pipeline = PublishedPipeline.get(ws, id=pipeline_id)
pipeline.disable()
```

THE MLOPS PIPELINE

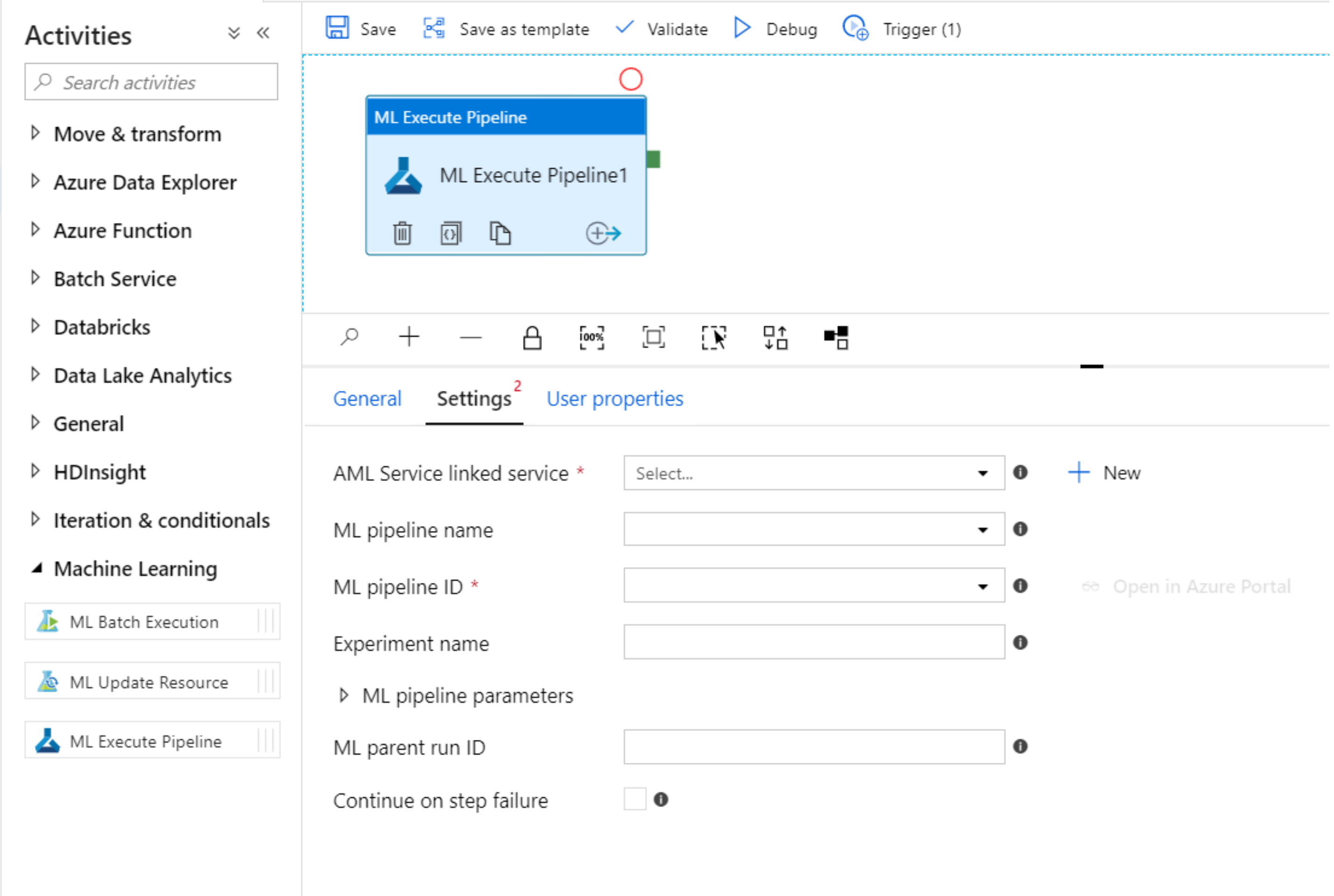
~~Sky~~ Cloud is the limit !

Let’s imaginate a **A/B testing : retrain and deploy if metrics are better**



LAUNCH FROM AZURE DATA FACTORY

Choose the ML Execute Pipeline activity and give the ID of the pipeline



TIME TO CONCLUDE

This is my opinion. Try and make yours !

STRENGHTS AND WEAKNESSES OF AZURE MACHINE LEARNING

What I love

Code / UI duality
Jupyter and RStudio in one click
Split storage and compute resources but easy to link the services together

When notebooks templates are ready, we save a lot of time !

What I need to work better

Other connectors (JDBC, ODBC ?)
Data preparation UI (does it make sense ?)
Monitor parallel compute (something like Spark UI ?)
Integrate a feedback loop

The good scenario :

- Data in Azure storage
- Data Scientists know Python
- Azure resources management to contain cost

NOW, WE CAN SPEND TIME ON…

The most interesting part of our job !

INTERPRET THE BLACK BOX

The more efficient the model is, the more complex it is to interpret.

eXplainable AI is an approach that aims to understand what happens inside the black box.

It is essential to convince people of the interest of using algorithms and to know **how to influence a forecast**.
Predictive becomes **prescriptive.**

It helps to answer the following questions :
- Model debugging - Why did my model make this mistake?
- Detecting fairness issues - Does my model discriminate?
- Human-AI cooperation - How can I understand and trust the model's decisions?
- Regulatory compliance - Does my model satisfy legal requirements?

Microsoft Research works on the **Interpret Community SDK**
https://github.com/interpretml/interpret-community

Interpretability Technique
Explainable Boosting
Decision Tree
Decision Rule List
Linear/Logistic Regression
SHAP Kernel Explainer
SHAP Tree Explainer
LIME
Morris Sensitivity Analysis
Partial Dependence

```python
from interpret.ext.blackbox import TabularExplainer

# "features" and "classes" fields are optional
explainer = TabularExplainer(model,
                             X_train,
                             features=np.array(['age', 'sex', 'bmi', 'bp', 's1', 's2', 's3', 's4', 's5', 's6'], dtype='<U23')
                             #,classes=['Y']
                            )

global_explanation = explainer.explain_global(X_test)
```

```python
# Corresponding feature names
print('ranked global importance names: {}'.format(global_explanation.get_ranked_global_names()))

# Sorted SHAP values
print('ranked global importance values: {}'.format(global_explanation.get_ranked_global_values()))

# Feature ranks (based on original order of features)
print('global importance rank: {}'.format(global_explanation.global_importance_rank))
```

```
ranked global importance names: ['s1', 's5', 'bmi', 's2', 'bp', 'sex', 's4', 's3', 's6', 'age']
ranked global importance values: [34.175703187057636, 29.19342523864581, 22.56742133583082, 19.725798776315216, 13.066054697621201, 11.47827
4591665915, 10.136964208294199, 6.62116297286142, 1.7154433363372315, 1.6745392477467216]
global importance rank: [4, 8, 2, 5, 3, 1, 7, 6, 9, 0]
```

Shapley values : based on game theory

It allows you to order features according to their contribution

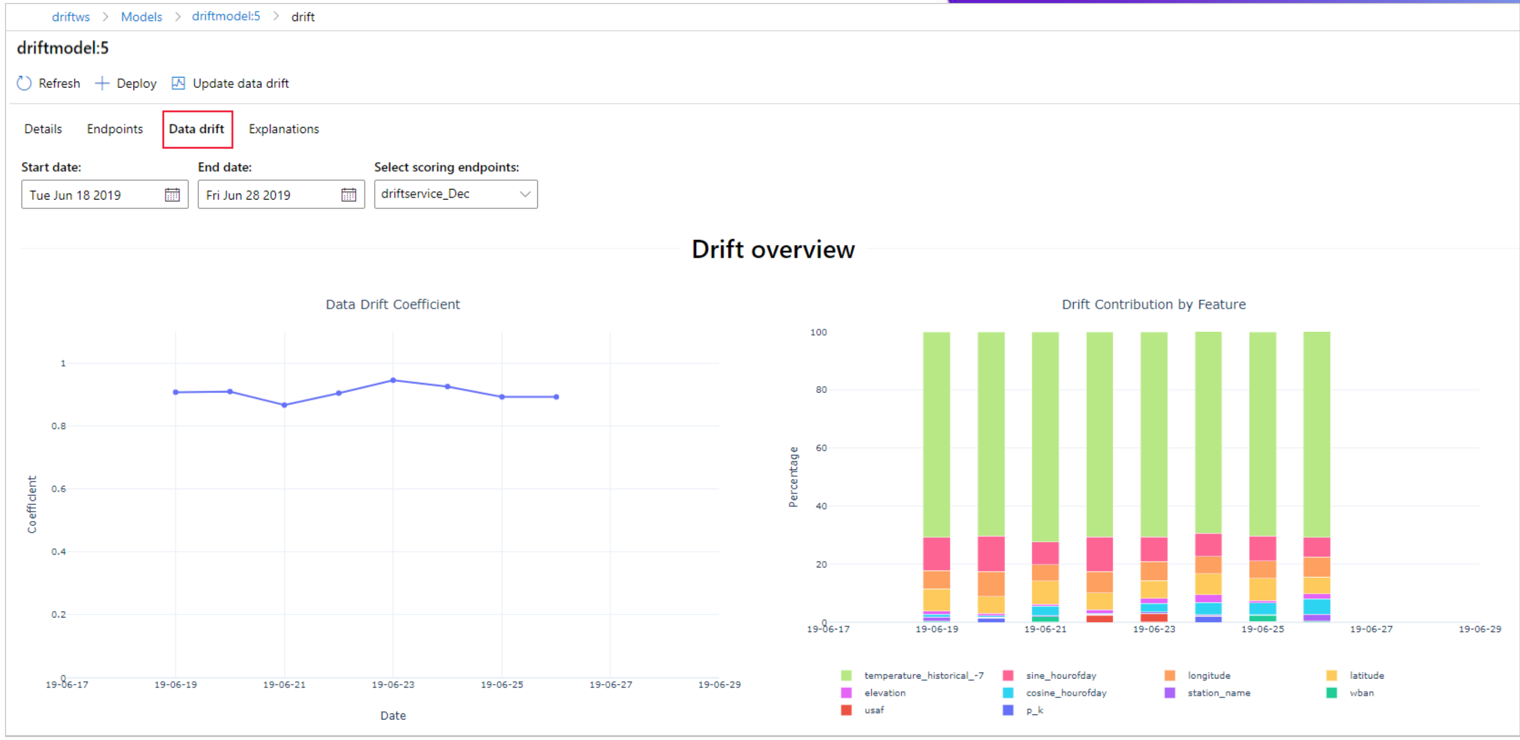
THE SILENT FAILURE OF THE MODEL

Data drift (enterprise preview feature)

Change in new input data that leads to model performance degradation

Only on Azure Kubernetes Service (AKS) deployment

Drift coefficient :
based on Matthews correlation coefficient
(equivalent to Pearson's phi coefficient)



Monitor :

- Measures the **magnitude** of data drift
- Measures the data drift **contribution** by feature, indicating which features caused data drift
- Send alerts to data drift by **email**

TIME FOR QUESTION & ANSWER

Q&A

DATAXDAYS

Programme du mardi 2 juin

publicis sapient
ENGINEERING • PRODUCT • AI & DATA • STRATEGY • EXPERIENCE

13h - 13h45
Data Security

Johan Jublan & Giulia Bianchi :
Dévoiler les secrets d'un modèle de machine learning : une menace crédible ?

Niveau 3 :
Securmax

17h30 - 18h15
DataScience en prod.

Paul Péton :
Azureml : Python SDK to train, package and deploy models (another story of MLOps)

Niveau 3 :
SciProdank

18h30 - 19h15
Data Architecture

Florent Ramière :
Retour aux fondamentaux : des bases de données aux plateformes à base d'évènement

Niveau 1 :
Prototys

THANK YOU

CONTACT

Paul PETON

paul.peton@azeo.com
M: +33 6 111 022 01