 **MLflow Overview:**

* MLflow is an open-source platform for managing the ML lifecycle.
* It standardizes model packaging, making models easy to import/export across workflows.
* MLflow integrates with Azure Machine Learning for seamless deployment.

 **Benefits of Using MLflow:**

* Facilitates easy model deployment and tracking with Azure ML.
* Standardizes model packaging for consistent import/export.
* Supports logging models and their metadata for traceability.

 **Model Registration with MLflow:**

* Models can be logged as artifacts or as models with additional information for direct use in pipelines or deployments.
* Registration creates an MLmodel file containing the model's metadata.

 **Autologging:**

* Use mlflow.autolog() to automatically log parameters, metrics, artifacts, and the model.
* Autologging detects the training framework and logs the model accordingly.
* Common flavors for autologging: Keras, Scikit-learn, LightGBM, XGBoost, TensorFlow, PyTorch, ONNX.

 **Manual Logging:**

* For more control, use autolog() with log\_models=False and manually log the model.
* Manually customize model's expected inputs and outputs by defining a signature.

 **Model Signature:**

* Defines the schema of the model's inputs and outputs, stored in the MLmodel file.
* Can be inferred from training datasets and model predictions using infer\_signature().

 **Azure ML Integration:**

* Easily deploy MLflow models with Azure's no-code deployment.
* Custom models can be registered if not supported by Azure ML and MLflow integration.

 **Key Points:**

* MLflow simplifies model deployment across different environments.
* Autologging and manual logging options provide flexibility in how models are tracked and deployed.
* Model signature ensures compatibility and error-free deployment.

**MLflow Model Format**

**MLModel File**

* **Essential File**: The MLmodel file is crucial for defining how the model should be loaded and used.
* **Contents of MLmodel File**:
  + **artifact\_path**: Path where the model is logged during training.
  + **flavor**: Specifies the machine learning library used to create the model.
  + **model\_uuid**: Unique identifier for the registered model.
  + **run\_id**: Unique identifier for the job run during which the model was created.
  + **signature**: Defines the schema of the model's inputs and outputs.
    - **inputs**: Valid inputs for the model (e.g., subset of the training dataset).
    - **outputs**: Valid outputs of the model (e.g., model predictions for the input dataset).

**Model Flavors**

* **Definition**: A flavor specifies the machine learning library used for model creation.
* **Customization**: Each flavor defines its own methods for persisting and loading models, optimizing performance without compromising compatibility with the MLModel standard.
* **Example**: Fastai for image classification models.
* **Default Flavor**: Python function flavor is the default interface for models created from an MLflow run. It ensures compatibility and interoperability across different workflows and deployment environments.

**Model Signature**

* **Purpose**: Serves as a data contract between the model and the server.
* **Types of Signatures**:
  + **Column-based**: For tabular data, using pandas.DataFrame as inputs.
  + **Tensor-based**: For n-dimensional arrays or tensors, using numpy.ndarray as inputs.
* **Autologging**: When using MLflow's autologging, the signature is inferred automatically.
* **Manual Logging**: If a different signature is required, it must be set manually.
* **Deployment Enforcement**: In Azure Machine Learning's no-code deployment, the inputs and outputs defined in the signature must match the data sent to the model.

**Key Points**

* **MLmodel File**: Central to model loading and usage, containing metadata and schema definitions.
* **Model Flavors**: Allow flexibility and optimization in model persistence and loading.
* **Signatures**: Define input and output schemas, crucial for model deployment and operationalization.

**Registering an MLflow Model in Azure Machine Learning**

**Key Points**

* **Model Registry**: Azure Machine Learning's model registry stores and versions models in the workspace, identified by name and version.
* **Model Types**:
  + **MLflow**: Recommended for standard use cases.
  + **Custom**: For models with a custom standard not supported by Azure ML.
  + **Triton**: For deep learning workloads, particularly TensorFlow and PyTorch.
* **Metadata Tags**: Useful for easier searching and organization of models.
* **Flexibility**: Models trained outside Azure ML can also be registered by providing the local path to the model's artifacts.

**Best Practices**

* **Integration with MLflow**: Logging and registering MLflow models simplifies model management and deployment, as the environment and scoring script are automatically created.

**Steps to Register an MLflow Model**

1. **Submit a Training Job**:
   * Use the azure.ai.ml library to configure and submit a training job.

from azure.ai.ml import command

# configure job

job = command(

code="./src",

command="python train-model-signature.py --training\_data diabetes.csv",

environment="AzureML-sklearn-0.24-ubuntu18.04-py37-cpu@latest",

compute="aml-cluster",

display\_name="diabetes-train-signature",

experiment\_name="diabetes-training"

)

# submit job

returned\_job = ml\_client.create\_or\_update(job)

aml\_url = returned\_job.studio\_url

print("Monitor your job at", aml\_url)

1. **Register the Model**:
   * Once the job is completed, use the job name to locate the job run and register the model from its outputs.

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="mlflow-diabetes",

description="Model created from run.",

type=AssetTypes.MLFLOW\_MODEL,

)

# Uncomment after adding required details above

ml\_client.models.create\_or\_update(run\_model)

1. **Verify Registration**:
   * All registered models are listed in the Models page of the Azure Machine Learning studio.
   * The registered model includes the model's output directory.
   * The MLmodel file can be found in the artifacts of the registered model.

**Additional Tips**

* **Monitoring Jobs**: Use the provided aml\_url to monitor your job's progress in the Azure ML studio.
* **MLmodel File**: Ensure that the MLmodel file is correctly logged to facilitate smooth deployment and usage.

This process ensures your MLflow model is properly registered, versioned, and ready for deployment within Azure Machine Learning, making it easier to manage and deploy across different environments.

To help you with implementing responsible AI, Azure Machine Learning offers the Responsible AI dashboard. You can create and customize the Responsible AI dashboard to explore your data and model.

Whatever you use a model for, you should consider the Responsible Artificial Intelligence (Responsible AI) principles. Depending on the use case, you may focus on specific principles. Nevertheless, it's a best practice to consider all principles to ensure you're addressing any issues the model may have.

Microsoft has listed five Responsible AI principles:

Fairness and inclusiveness: Models should treat everyone fairly and avoid different treatment for similar groups.

Reliability and safety: Models should be reliable, safe, and consistent. You want a model to operate as intended, handle unexpected situations well, and resist harmful manipulation.

Privacy and security: Be transparent about data collection, use, and storage, to empower individuals with control over their data. Treat data with care to ensure an individual's privacy.

Transparency: When models influence important decisions that affect people's lives, people need to understand how those decisions were made and how the model works.

Accountability: Take accountability for decisions that models may influence and maintain human control.

**Creating a Responsible AI Dashboard in Azure Machine Learning**

**Overview**

The Responsible AI (RAI) dashboard helps evaluate whether your model is safe, trustworthy, and ethical by generating and exploring various insights.

**Steps to Create a Responsible AI Dashboard**

1. **Pipeline Structure**:
   * Start with the **RAI Insights dashboard constructor**.
   * Include one or more **RAI tool components** (e.g., explanation, causal analysis, counterfactuals, error analysis).
   * End with **Gather RAI Insights dashboard** to collect all insights.
   * Optionally, add **Gather RAI Insights scorecard** to generate a PDF scorecard.
2. **RAI Tool Components**:
   * **Explanation**: Generates model explanations showing feature influences on predictions.
   * **Causal**: Uses historical data to view causal effects of features on outcomes.
   * **Counterfactuals**: Explores how changes in input affect model output.
   * **Error Analysis**: Identifies and explores erroneous subgroups in the data.
3. **Building the Pipeline Using the Python SDK**:

**Registering Datasets and Model**:

# Assuming train\_data, test\_data, and azureml\_model\_id are already registered and available

**Retrieving Components**:

rai\_constructor\_component = ml\_client\_registry.components.get(

name="microsoft\_azureml\_rai\_tabular\_insight\_constructor", label="latest"

)

rai\_explanation\_component = ml\_client\_registry.components.get(

name="microsoft\_azureml\_rai\_tabular\_explanation", label="latest"

)

rai\_gather\_component = ml\_client\_registry.components.get(

name="microsoft\_azureml\_rai\_tabular\_insight\_gather", label="latest"

)

**Building the Pipeline**:

from azure.ai.ml import Input, dsl

from azure.ai.ml.constants import AssetTypes

@dsl.pipeline(

compute="aml-cluster",

experiment\_name="Create RAI Dashboard",

)

def rai\_decision\_pipeline(

target\_column\_name, train\_data, test\_data

):

# Initiate the RAIInsights

create\_rai\_job = rai\_constructor\_component(

title="RAI dashboard diabetes",

task\_type="classification",

model\_info=expected\_model\_id,

model\_input=Input(type=AssetTypes.MLFLOW\_MODEL, path=azureml\_model\_id),

train\_dataset=train\_data,

test\_dataset=test\_data,

target\_column\_name="Predictions",

)

create\_rai\_job.set\_limits(timeout=30)

# Add explanations

explanation\_job = rai\_explanation\_component(

rai\_insights\_dashboard=create\_rai\_job.outputs.rai\_insights\_dashboard,

comment="add explanation",

)

explanation\_job.set\_limits(timeout=10)

# Combine everything

rai\_gather\_job = rai\_gather\_component(

constructor=create\_rai\_job.outputs.rai\_insights\_dashboard,

insight=explanation\_job.outputs.explanation,

)

rai\_gather\_job.set\_limits(timeout=10)

rai\_gather\_job.outputs.dashboard.mode = "upload"

return {

"dashboard": rai\_gather\_job.outputs.dashboard,

}

**Running the Pipeline**:

from azure.ai.ml import MLClient

from azure.identity import DefaultAzureCredential

# Authenticate and create a client

credential = DefaultAzureCredential()

ml\_client = MLClient(credential, subscription\_id, resource\_group, workspace\_name)

# Define and run the pipeline

pipeline = rai\_decision\_pipeline(

target\_column\_name="your\_target\_column",

train\_data=Input(type=AssetTypes.MLTABLE, path="path\_to\_train\_data"),

test\_data=Input(type=AssetTypes.MLTABLE, path="path\_to\_test\_data")

)

# Submit the pipeline

pipeline\_run = ml\_client.jobs.create\_or\_update(pipeline)

pipeline\_run.wait\_for\_completion(show\_output=True)

1. **Exploring the Dashboard**:
   * Once the pipeline run is complete, view the Responsible AI dashboard from the pipeline overview.
   * Alternatively, find the dashboard in the Responsible AI tab of the registered model in the Azure Machine Learning studio.

**Evaluating the Responsible AI Dashboard in Azure Machine Learning**

The Responsible AI dashboard in Azure Machine Learning studio allows you to evaluate your model using various insights to ensure it is safe, trustworthy, and ethical. Here's a detailed breakdown of what you can explore and review in the dashboard:

**Connecting to a Compute Instance**

When you open the Responsible AI dashboard, Azure Machine Learning studio tries to automatically connect it to a compute instance. This instance provides the necessary resources for interactive exploration within the dashboard.

**Insights in the Responsible AI Dashboard**

Depending on the components you selected during the pipeline creation, the following insights may be available in your Responsible AI dashboard:

1. **Error Analysis**
2. **Explanations**
3. **Counterfactuals**
4. **Causal Analysis**

**Explore Each Insight**

**1. Error Analysis**

Error analysis helps you understand how errors are distributed across your dataset. This analysis is critical for identifying specific subgroups, or cohorts, where the model might be making more false predictions.

* **Error Tree Map**: Visualizes which combinations of subgroups result in higher error rates.
* **Error Heat Map**: Presents a grid view of errors over the scale of one or two features, making it easier to spot patterns.

**2. Explanations**

Understanding how a model reaches its predictions is essential for ensuring transparency and trustworthiness. Explanations provide insights into the feature importance:

* **Aggregate Feature Importance**: Shows how each feature in the test data influences the model's predictions overall.
* **Individual Feature Importance**: Displays the impact of each feature on an individual prediction.

**3. Counterfactuals**

Counterfactual analysis allows you to explore how changes in input features could alter the model's predictions. This analysis is useful for understanding the conditions needed to achieve different outcomes.

* **What-If Counterfactuals**: Select a data point and desired prediction to see what changes in input would achieve that prediction.

**4. Causal Analysis**

Causal analysis helps estimate the average effect of a feature on a desired prediction, aiding in decision-making:

* **Aggregate Causal Effects**: Shows the average causal effects for predefined treatment features.
* **Individual Causal Effects**: Allows you to explore the influence of treatment features on individual data points.
* **Treatment Policy**: Indicates which data points benefit most from a treatment.