Lab: MLflow Tracking

The purpose of this lab is to provide a quick guide to the most essential core APIs of MLflow Tracking. Specifically, those that enable the logging, registering, and loading of a model for inference.

Pre-Req:

1. Run the github erpository in the lab environment:

```
git clone https://github.com/fenago/mlops-ai-security.git
```

2. Start jupyter lab by running the following command in the terminal:

```
jupyter lab
```

Lab Solution

Complete solution for this lab is available in the `lab1_tracking_quickstart.ipynb` notebook.

What you will learn

In this lab, you will learn:

- How to **log** parameters, metrics, and a model
- The basics of the **MLflow fluent API**
- How to **register** a model during logging
- How to navigate to a model in the **MLflow UI**
- How to **load** a logged model for inference

Step 1 - Get MLflow

MLflow is available on PyPI. If you don't already have it installed on your system, you can install it with:

```
pip install mlflow
```

Step 2 - Start a Tracking Server

Run a local Tracking Server

We're going to start a local MLflow Tracking Server, which we will connect to for logging our data for this quickstart. From a terminal, run:

```
mlflow server --host 127.0.0.1 --port 8080
```

Set the Tracking Server URI (if not using a Databricks Managed MLflow Tracking Server)

Since we are using local tracking server, ensure that you set the tracking server's uri using:

- Python

```
import mlflow

mlflow.set_tracking_uri(uri="http://<host>:<port>")
```

If this is not set within your notebook or runtime environment, the runs will be logged to your local file system.

Step 3 - Train a model and prepare metadata for logging

In this section, we're going to log a model with MLflow. A quick overview of the steps are:

- Load and prepare the Iris dataset for modeling.
- Train a Logistic Regression model and evaluate its performance.
- Prepare the model hyperparameters and calculate metrics for logging.
- Python

```
import mlflow
from mlflow.models import infer_signature

import pandas as pd
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score


# Load the Iris dataset
X, y = datasets.load_iris(return_X_y=True)

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# Define the model hyperparameters
params = {
    "solver": "lbfgs",
    "max_iter": 1000,
    "multi_class": "auto",
    "random_state": 8888,
}

# Train the model
lr = LogisticRegression(**params)
lr.fit(X_train, y_train)

# Predict on the test set
y_pred = lr.predict(X_test)

# Calculate metrics
accuracy = accuracy_score(y_test, y_pred)
```

Step 4 - Log the model and its metadata to MLflow

In this next step, we're going to use the model that we trained, the hyperparameters that we specified for the model's fit, and the loss metrics that were calculated by evaluating the model's performance on the test data to log to MLflow.

The steps that we will take are:

- Initiate an MLflow **run** context to start a new run that we will log the model and metadata to.
- **Log** model **parameters** and performance **metrics**.
- **Tag** the run for easy retrieval.
- **Register** the model in the MLflow Model Registry while **logging** (saving) the model.

Note

While it can be valid to wrap the entire code within the `start_run` block, this is **not recommended**. If there as in issue with the training of the model or any other portion of code that is unrelated to MLflow-related actions, an empty or partially-logged run will be created, which will necessitate manual cleanup of the invalid run. It is best to keep the training execution outside of the run context block to ensure that the loggable content (parameters, metrics, artifacts, and the model) are fully materialized prior to logging.

- Python

```
# Set our tracking server uri for logging
mlflow.set_tracking_uri(uri="http://127.0.0.1:8080")

# Create a new MLflow Experiment
mlflow.set_experiment("MLflow Quickstart")

# Start an MLflow run
with mlflow.start_run():
    # Log the hyperparameters
    mlflow.log_params(params)

    # Log the loss metric
    mlflow.log_metric("accuracy", accuracy)

    # Set a tag that we can use to remind ourselves what this run was for
    mlflow.set_tag("Training Info", "Basic LR model for iris data")

    # Infer the model signature
    signature = infer_signature(X_train, lr.predict(X_train))

    # Log the model
    model_info = mlflow.sklearn.log_model(
        sk_model=lr,
        artifact_path="iris_model",
        signature=signature,
        input_example=X_train,
        registered_model_name="tracking-quickstart",
    )
```

Step 5 - Load the model as a Python Function (pyfunc) and use it for inference

After logging the model, we can perform inference by:

- **Loading** the model using MLflow's pyfunc flavor.
- Running **Predict** on new data using the loaded model.

Note

The iris training data that we used was a numpy array structure. However, we can submit a Pandas DataFrame as well to the predict method, as shown below.

- Python

```
# Load the model back for predictions as a generic Python Function model
loaded_model = mlflow.pyfunc.load_model(model_info.model_uri)

predictions = loaded_model.predict(X_test)

iris_feature_names = datasets.load_iris().feature_names

# Convert X_test validation feature data to a Pandas DataFrame
result = pd.DataFrame(X_test, columns=iris_feature_names)

# Add the actual classes to the DataFrame
result["actual_class"] = y_test

# Add the model predictions to the DataFrame
result["predicted_class"] = predictions

result[:4]
```

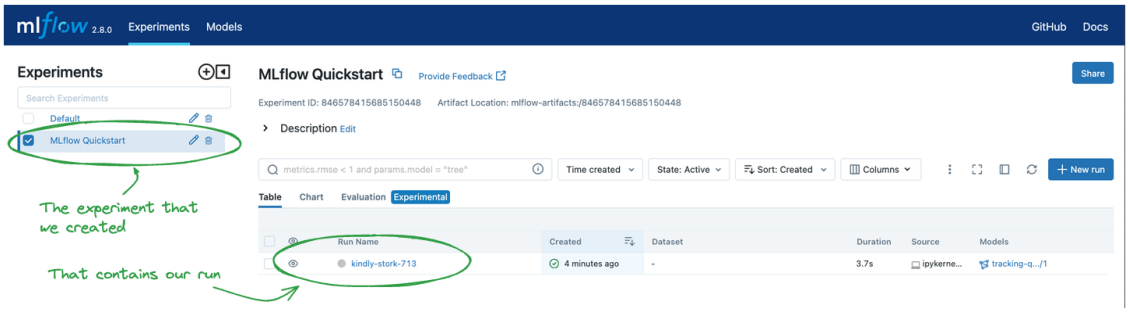
The output of this code will look something like this:

sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	actual_class	predicted_class
6.1	2.8	4.7	1.2	1	1
5.7	3.8	1.7	0.3	0	0
7.7	2.6	6.9	2.3	2	2
6.0	2.9	4.5	1.5	1	1

Step 6 - View the Run in the MLflow UI

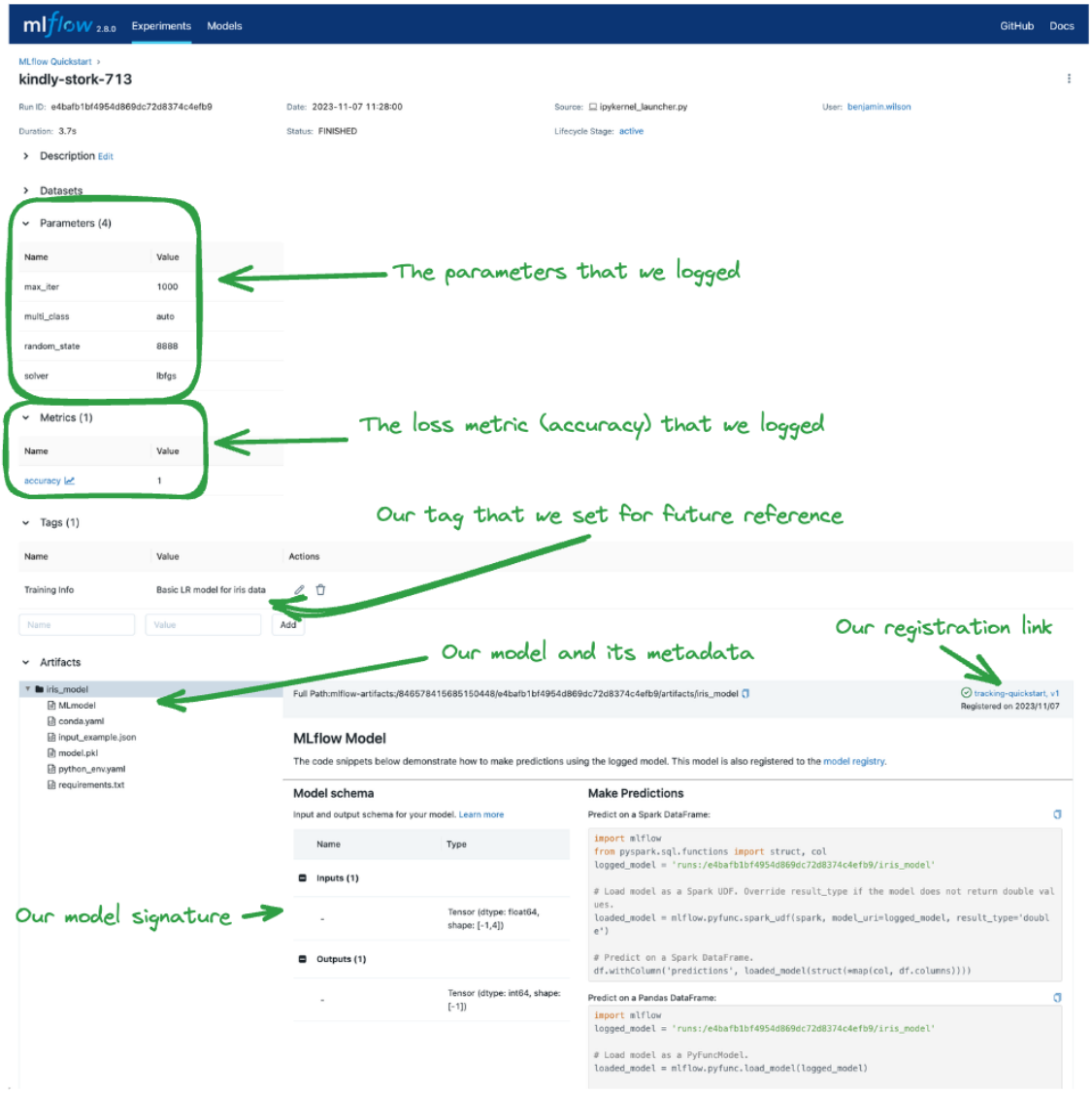
In order to see the results of our run, we can navigate to the MLflow UI. Since we have already started the Tracking Server at http://localhost:8080, we can simply navigate to that URL in our browser.

When opening the site, you will see a screen similar to the following:



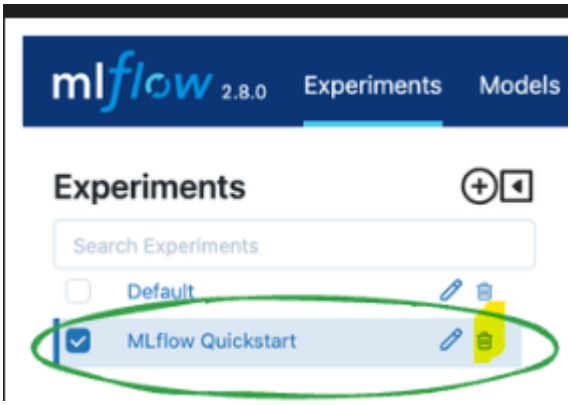
Clicking on the name of the Experiment that we created ("MLflow Quickstart") will give us a list of runs associated with the Experiment. You should see a random name that has been generated for the run and nothing else show up in the Table list view to the right.

Clicking on the name of the run will take you to the Run page, where the details of what we've logged will be shown. The elements have been highlighted below to show how and where this data is recorded within the UI.



Step 7 - Cleanup

1. Delete "MLflow Quickstart" from MLFlow UI.



2. Make sure to stop the MLFlow server in the terminal.

Conclusion

Congratulations on working through the MLflow Tracking Lab! You should now have a basic understanding of how to use the MLflow Tracking API to log models.