



MLflow Tracking Quickstart

! INFO

Looking for using MLflow for LLMs/Agent development? Checkout the [MLflow for GenAI documentation](#) instead. This guide is intended for data scientists who train traditional machine learning models, such as decision trees.

💡 MLFLOW ASSISTANT

Need help setting up tracking? Try [MLflow Assistant](#) - a powerful AI assistant that can help you set up MLflow tracking for your project.

Welcome to MLflow! The purpose of this quickstart is to provide a quick guide to the most essential core APIs of MLflow Tracking. In just a few minutes of following along with this quickstart, you will learn:

- How to **log** parameters, metrics, and a model using the MLflow logging API
- How to navigate to a model in the **MLflow UI**
- How to **load** a logged model for inference

Step 1 - Set up MLflow

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

```
bash
```

```
pip install mlflow
```

! INFO

See [Secure Installs](#) to learn how to pin dependencies to known good versions using hash checking and upload-time filtering.

Then, follow the instructions in the [Set Up MLflow](#) guide to set up MLflow.

If you just want to start super quick, run the following code in a notebook cell:

 Ask AI

```
python
```

```
import mlflow

mlflow.set_experiment("MLflow Quickstart")
```

Step 2 - Prepare training data

Before training our first model, let's prepare the training data and model hyperparameters.

```
python
```

```
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,
    "random_state": 8888,
}
```

Step 3 - Train a model with MLflow Autologging

In this step, we train the model on the training data loaded in the previous step, and log the model and its metadata to MLflow. The easiest way to do this is to using MLflow's **Autologging** feature.

```
python
```

```
import mlflow

# Enable autologging for scikit-learn
mlflow.sklearn.autolog()
```



```
# Just train the model normally
lr = LogisticRegression(**params)
lr.fit(X_train, y_train)
```

With just one line of additional code `mlflow.sklearn.autolog()`, **now** you get the best of both worlds: you can focus on training the model, and MLflow will take care of the rest:

- **Saving** the trained model.
- Recording the model's performance metrics during training, such as accuracy, precision, AUC curve.
- Logging hyperparameter values used to train the model.
- Track metadata such as input data format, user, timestamp, etc.

To learn more about autologging and supported libraries, see the [Autologging](#) documentation.

Step 4 - **View** the Run in the MLflow UI

To see the results of training, you can access the MLflow UI by navigating to the URL of the Tracking Server. If you have not started one, open a new terminal and run the following command at the root of the MLflow project and access the UI at <http://localhost:5000> (or the port number you specified).

```
bash

mlflow server --port 5000
```

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

<input type="checkbox"/>	Name	Time created	Last modified	Description	Tags
<input type="checkbox"/>	MLflow Quickstart	10/17/2025, 08:33:08 AM	10/17/2025, 08:33:08 AM	-	
<input type="checkbox"/>	Default	05/21/2025, 07:48:22 PM	05/21/2025, 07:48:22 PM	-	

The "Experiments" section shows a list of (recently created) experiments. Click on the "MLflow Quickstart" experiment.



MLflow Quickstart Machine learning

Run Name	Created	Dataset	Duration	Source	Models
thundering-fly-695	2 hours ago	dataset (aa7a0237) Eval	2.0s	ipykern...	model

The training **Run** created by MLflow is listed in the table. Click the run to view the details.

thundering-fly-695

Overview | Model metrics | System metrics | Traces | Artifacts

Description
No description

Metrics (7)

Metric	Value	Models
training_score	0.975	model
training_roc_auc	0.9981258273521201	model
training_accuracy_score	0.975	model
training_precision_score	0.9767857142857144	model
training_log_loss	0.13582657756813249	model
training_f1_score	0.9743882794186532	model
training_recall_score	0.975	model

Parameters (15)

Parameter	Value
tcl	0.0001
penalty	l2
class_weight	None
random_state	8888

About this run

- Created at: 10/17/2025, 08:33:11 AM
- Created by: admin
- Experiment ID: 379551186730453731
- Status: Finished
- Run ID: 43dcfb01bc594b198f03hbchfeeed34d
- Duration: 2.0s
- Source: ipykernel_launcher.py
- Registered prompts: -

Datasets

- dataset (aa7a0237) Eval +1

Tags

- estimator_class: sklearn.linear_model._logistic...
- estimator_name: LogisticRegression

Registered models

None

The Run detail page shows an overview of the run, its recorded metrics, hyper-parameters, tags, and more. Play around with the UI to see the different views and features.

Scroll down to the "Model" section and you should see the model that was logged during training. Click on the model to view the details.

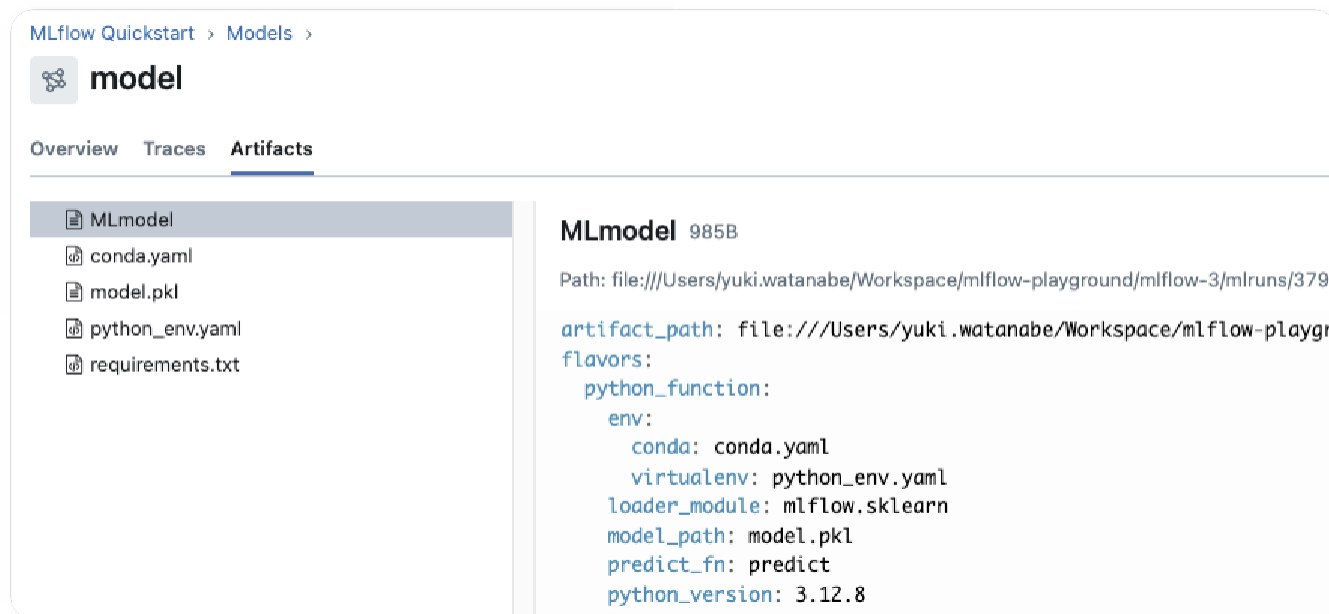
Logged models (1)

Type	Step	Model name	Status	Created	Registered models
Output	0	model	Ready	2 hours ago	-

Ask AI

The model page displays similar metadata such as performance metrics and hyper-parameters. It also includes an "Artifacts" section that lists the files that were logged during training. You can also see environment information such as the Python version and dependencies, which are stored for

reproducibility.



The screenshot shows the MLflow interface for a model named 'MLmodel'. The 'Artifacts' tab is active, displaying a list of files: MLmodel, conda.yaml, model.pkl, python_env.yaml, and requirements.txt. The details for the 'MLmodel' artifact are shown on the right, including its path, artifact path, and various flavors like python_function, env, loader_module, model_path, predict_fn, and python_version.

Step 5 - Log a model and metadata manually

Now that we have learned how to log a model training run with MLflow autologging, let's step further and learn how to log a model and metadata manually. This is useful when you want to have more control over the logging process.

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.
- Train and test the model.
- **Log** model **parameters** and performance **metrics**.
- **Tag** the run for easy retrieval.

```
python
```

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

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

    # Log the model
    model_info = mlflow.sklearn.log_model(sk_model=lr, name="iris_model")

    # Predict on the test set, compute and log the Loss metric
```



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

```
# Optional: 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")
```

Step 6 - Load the model back 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.

! INFO

To load the model as a native scikit-learn model, use

```
mlflow.sklearn.load_model(model_info.model_uri)
```

 instead of the `pyfunc` flavor.

```
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
```

```
result = pd.DataFrame(X_test, columns=iris_feature_names)
result["actual_class"] = y_test
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



Ask AI

sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	actual_class	predicted_class
7.7	2.6	6.9	2.3	2	2
6.0	2.9	4.5	1.5	1	1

Next Steps

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

- [MLflow for GenAI](#): Learn how to use MLflow for GenAI/LLM development.
- [MLflow for Deep Learning](#): Learn how to use MLflow for deep learning frameworks such as PyTorch, TensorFlow, etc.
- [MLflow Tracking](#): Learn more about the MLflow Tracking APIs.
- [Self-hosting Guide](#): Learn how to self-host the MLflow Tracking Server and set it up for team collaboration.



[Components](#) [Releases](#) [Blog](#)

[Docs](#)

[Ambassador Program](#)

