Use the following code to start the run. Since you have an experiment ID, start your run using:

mlflow.start\_run().

mlflow.start\_run(experiment\_id=1027351307587712)

Alternatively, you can define the experiment name as follows, which will simply use the experiment name rather than ID:

experiment\_name = "/Demo/MLFlowDemo/"

mlflow.set\_experiment(experiment\_name)

From a logging perspective, there is the option to auto log model-specific metrics, parameters, and model artifacts. Alternatively, it is possible to define these metrics, parameters, or models by adding the following commands to the notebook code as desired:

Numerical Metrics: mlflow.log\_metric("accuracy", 0.9)

Training Parameters: mlflow.log\_param("learning\_rate", 0.001)

Models: mlflow.sklearn.log\_model(model, "myModel")

Other artifacts(files): mlflow.log\_artifact("/tmp/my-file", "myArtifactPath")

The following code imports a dataset from scikit-learn and creates the training and test datasets. This code is meant to demonstrate a sample ML model. Add this to a code block in the Databricks Notebook.

from sklearn.model\_selection import train\_test\_split

from sklearn.datasets import load\_diabetes

db = load\_diabetes()

X = db.data

y = db.target

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y)

This next block of code will import MLflow and sklearn, start the experiment, and log details of the execution run. Simply add this code to your Databricks notebook and run it.

import mlflow

import mlflow.sklearn

from sklearn.ensemble import RandomForestRegressor

from sklearn.metrics import mean\_squared\_error

# In this run, neither the experiment\_id nor the experiment\_name parameter is provided. MLflow automatically creates a notebook experiment and logs runs to it.

# Access these runs using the Experiment sidebar. Click Experiment at the upper right of this screen.

with mlflow.start\_run(experiment\_id=1027351307587712):

n\_estimators = 100

max\_depth = 6

max\_features = 3

# Create and train model

rf = RandomForestRegressor(n\_estimators = n\_estimators, max\_depth = max\_depth, max\_features = max\_features)

rf.fit(X\_train, y\_train)

# Make predictions

predictions = rf.predict(X\_test)

# Log parameters

mlflow.log\_param("num\_trees", n\_estimators)

mlflow.log\_param("maxdepth", max\_depth)

mlflow.log\_param("max\_feat", max\_features)

# Log model

mlflow.sklearn.log\_model(rf, "random-forest-model")

# Create metrics

mse = mean\_squared\_error(y\_test, predictions)

# Log metrics

mlflow.log\_metric("mse", mse)

After testing the defined logging option, also test auto logging. You can do that using the following variation of the code shown in Figure 23-13, which basically replaces all of the defined metrics, parameters, and so forth with a call to mlflow.sklearn.autolog().

import mlflow

import mlflow.sklearn

from sklearn.ensemble import RandomForestRegressor

from sklearn.metrics import mean\_squared\_error

mlflow.sklearn.autolog()

# In this run, neither the experiment\_id nor the experiment\_name parameter is provided. MLflow automatically creates a notebook experiment and logs runs to it.

# Access these runs using the Experiment sidebar. Click Experiment at the upper right of this screen.

with mlflow.start\_run():

n\_estimators = 100

max\_depth = 6

max\_features = 3

# Create and train model

rf = RandomForestRegressor(n\_estimators = n\_estimators, max\_depth = max\_depth, max\_features = max\_features)

rf.fit(X\_train, y\_train)

# Make predictions

predictions = rf.predict(X\_test)