

# MLflow



## Introduction

MLflow is a versatile, expandable, open-source platform for managing workflows and artifacts across the machine learning lifecycle. It has built-in integrations with many popular ML libraries, but can be used with any library, algorithm, or deployment tool.

MLflow has five components:

- **MLflow Tracking:** An API for logging parameters, code versions, metrics, model environment dependencies, and model artifacts when running your machine learning code.
- **MLflow Models:** A model packaging format and suite of tools to easily deploy a trained model for batch or real-time inference on platforms such as Docker, Apache Spark, Databricks, Azure ML and AWS SageMaker.
- **MLflow Model Registry:** A centralized model store, set of APIs, and UI focused on the approval, quality assurance, and deployment of an MLflow Model.
- **MLflow Projects:** A standard format for packaging reusable data science code that can be run with different parameters to train models, visualize data, or perform any other data science task.
- **MLflow Recipes:** Predefined templates for developing high-quality models for a variety of common tasks, including classification and regression.

## MLflow Tracking

The MLflow Tracking component is an API and UI for logging parameters, code versions, metrics, and output files when running your machine learning code and for later visualizing the results.

It is organized around the concept of “**runs**”, which are executions of a piece of code. Each **run** records the following information:

- **Start Time:** start time of the run
- **Duration:** duration of the run
- **Source:** Name of the file used to launch the run, or the project name and entry point for the run
- **Parameters:** Key-value input parameters of your choice
- **Metrics:** Key-value metrics, where the value is numeric.
- **Artifacts:** Output files in any format. For example, you can record images (e.g. PNGs), models (e.g. a pickled scikit-learn model), and data files (e.g. a Parquet file) as artifacts.

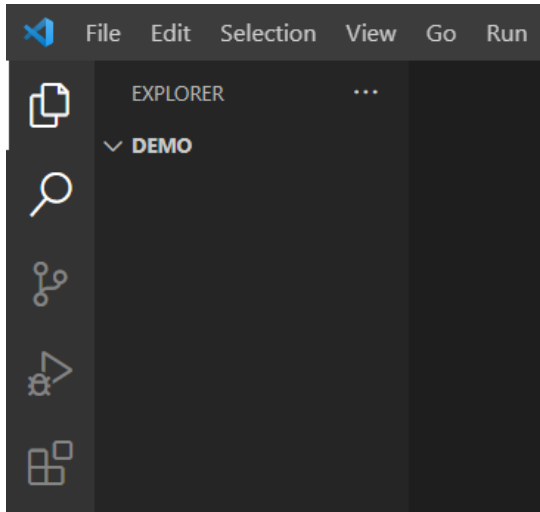
*Runs* can be recorded using MLflow Python APIs from anywhere you run your code. For example, you can record them in a standalone program, on a remote cloud machine, or in an interactive notebook.

*Runs* can also be organized into “**experiments**”, which group together runs for a specific task.

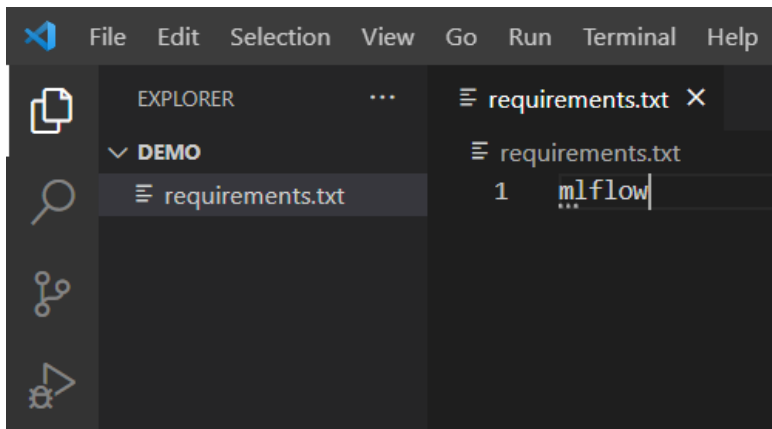
## Steps to setup MLflow Tracking for a simple Python program

Here, you will learn how to use the mlflow library to track the experiment for the Diabetes dataset.

1. Create a new folder on your system, and open it in VS-Code



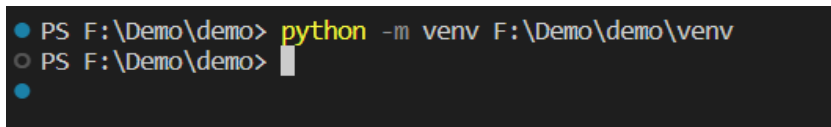
2. Inside the folder, create a new file, requirements.txt, and add 'mlflow' to it.



3. Create a virtual environment, and activate it.

Go to Terminal > New Terminal, and run below command.

```
python -m venv <your-path>\venv
```



Close this terminal, and open the command palette using Ctrl+Shift+P. Select Python Interpreter, choose the recommended venv.

Open New terminal, virtual environment should be activated.

```
PS F:\Demo\demo> & f:/Demo/demo/venv/Scripts/Activate.ps1
o (venv) PS F:\Demo\demo>
```

4. Install requirements using pip.

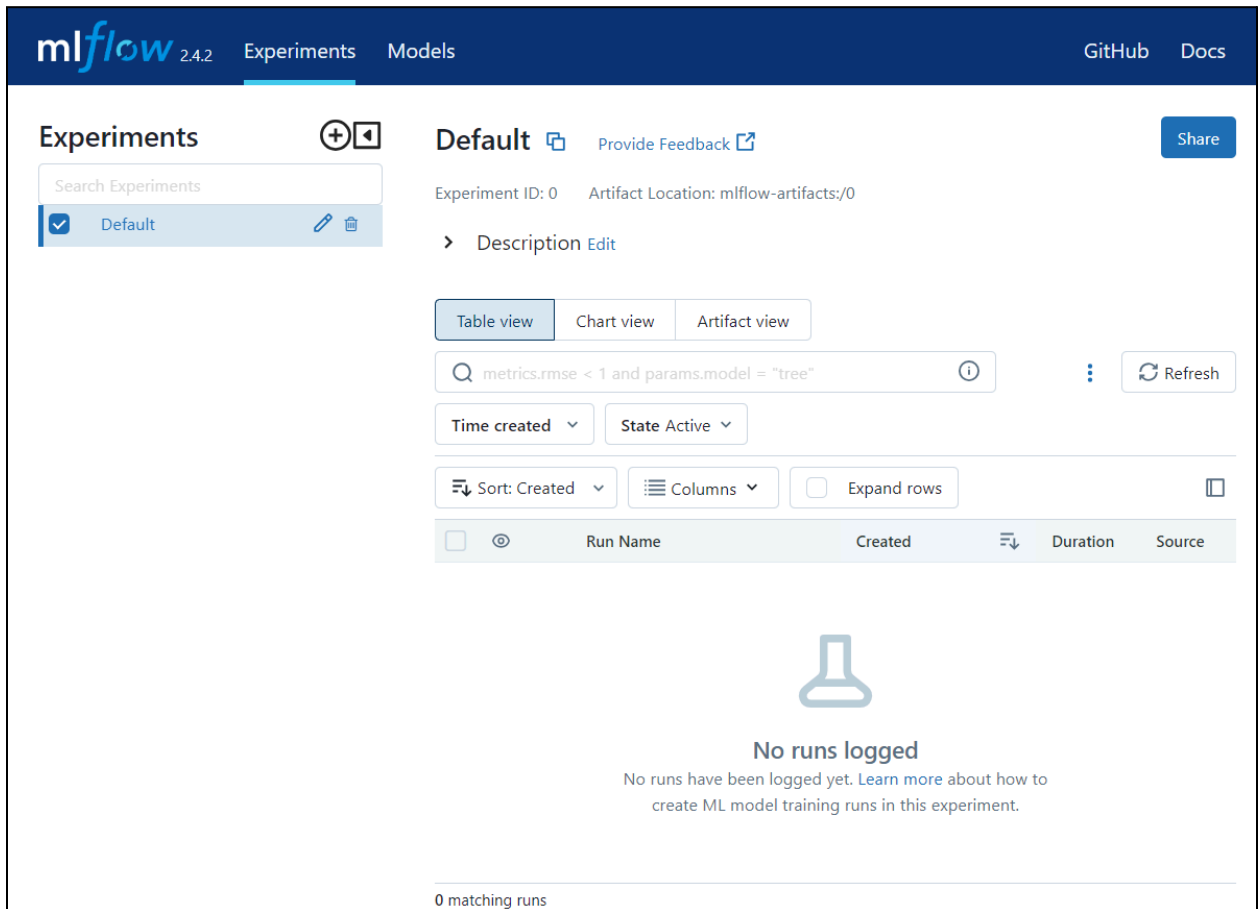
```
o (venv) PS F:\Demo\demo> pip install -r requirements.txt
Collecting mlflow
  Downloading mlflow-2.4.2-py3-none-any.whl (18.1 MB)
    0.9/18.1 MB 757.6 kB/s eta 0:00:23
```

5. Once installed, execute below command in the terminal.

`mlflow ui` OR `mlflow ui --port 5000`

```
o (venv) PS F:\Demo\demo> mlflow ui
INFO:waitress:erving on http://127.0.0.1:5000
```

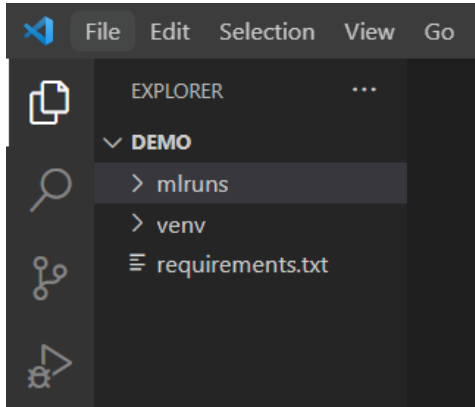
Navigate to <http://localhost:5000> in your browser. You will see a page similar to below:



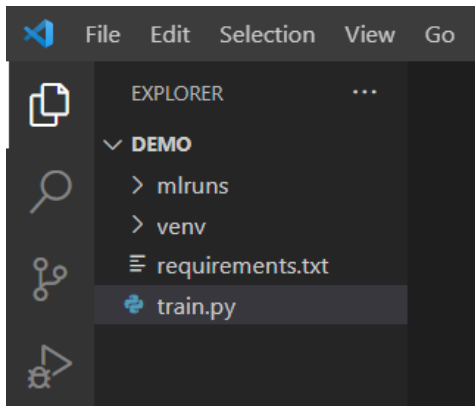
The screenshot displays the mlflow 2.4.2 web interface. The top navigation bar includes the mlflow logo, version 2.4.2, and tabs for 'Experiments' and 'Models'. On the right, there are links for 'GitHub' and 'Docs'. The 'Experiments' sidebar on the left shows a search bar and a list of experiments, with 'Default' selected. The main content area for the 'Default' experiment shows the 'Experiment ID: 0' and 'Artifact Location: mlflow-artifacts/0'. Below this, there are tabs for 'Table view', 'Chart view', and 'Artifact view'. A search bar contains the query 'metrics.rmse < 1 and params.model = "tree"'. There are also filters for 'Time created' and 'State Active'. A table with columns 'Run Name', 'Created', 'Duration', and 'Source' is shown, but it is empty. A message states 'No runs logged' and 'No runs have been logged yet. Learn more about how to create ML model training runs in this experiment.' At the bottom, it says '0 matching runs'.

You are in the Default experiment, which doesn't contain any tracking data yet.

- Note that MLflow runs can be recorded to local files, to a SQLAlchemy-compatible database, or remotely to a tracking server. By default, the MLflow Python API logs runs locally to files in an “*mlruns*” directory wherever you ran your program.



- Now, download the requirements.txt, and train.py files shared along with this document, and place them in the above working folder.



requirements.txt:

```
mlflow
scikit-learn==1.3.0
```

train.py:

```
import mlflow

from sklearn.model_selection import train_test_split
from sklearn.datasets import load_diabetes
from sklearn.ensemble import RandomForestRegressor
```

```

from sklearn.metrics import mean_squared_error, r2_score

# Parameters
N_ESTIMATORS = 100
MAX_DEPTH = 6
MAX_FEATURES = 3
# Run name
RUN_NAME = "run-01"

# Set an experiment name, unique and case-sensitive
# It will create a new experiment if the experiment with given doesn't exist
exp = mlflow.set_experiment(experiment_name = "Diabetes Experiments")

# Start RUN
mlflow.start_run(run_name= RUN_NAME,                # specify name of the run
                 experiment_id= exp.experiment_id)  # experiment id under which
to create the current run

# Log parameters
mlflow.log_param("n_estimators", N_ESTIMATORS)
mlflow.log_param("max_depth", MAX_DEPTH)
mlflow.log_param("max_features", MAX_FEATURES)

# Load dataset
db = load_diabetes()

# Split into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(db.data, db.target)

# Create and train model
rf = RandomForestRegressor(n_estimators= N_ESTIMATORS, max_depth= MAX_DEPTH,
max_features= MAX_FEATURES)
rf.fit(X_train, y_train)

# Use the model to make predictions on the test set
predictions = rf.predict(X_test)

# Log performance metrics
mlflow.log_metric("training_r2_score", r2_score(rf.predict(X_train), y_train))
mlflow.log_metric("testing_r2_score", r2_score(predictions, y_test))
mlflow.log_metric("training_mse", mean_squared_error(rf.predict(X_train), y_train))

```

```
mlflow.log_metric("testing_mse", mean_squared_error(predictions, y_test))

# Log trained model
mlflow.sklearn.log_model(sk_model = rf, artifact_path= "trained_model")

# End an active MLflow run
mlflow.end_run()
```

8. Within train.py file, along with data loading, model building and training steps, following steps are included:

- **mlflow.set\_experiment():** to set an experiment name (unique and case-sensitive)  
It will create a new experiment if the experiment with the given name doesn't exist.

```
exp = mlflow.set_experiment(experiment_name = "Diabetes Experiments")
```

- **mlflow.start\_run():** to start a run with the given name, under the given experiment id

```
mlflow.start_run(run_name= RUN_NAME, experiment_id= exp.experiment_id)
```

- **mlflow.log\_param():** to log a parameter under the current run

```
mlflow.log_param("n_estimators", N_ESTIMATORS)
```

- **mlflow.log\_metric():** to log a metric under the current run

```
mlflow.log_metric("testing_r2_score", r2_score(predictions, y_test))
```

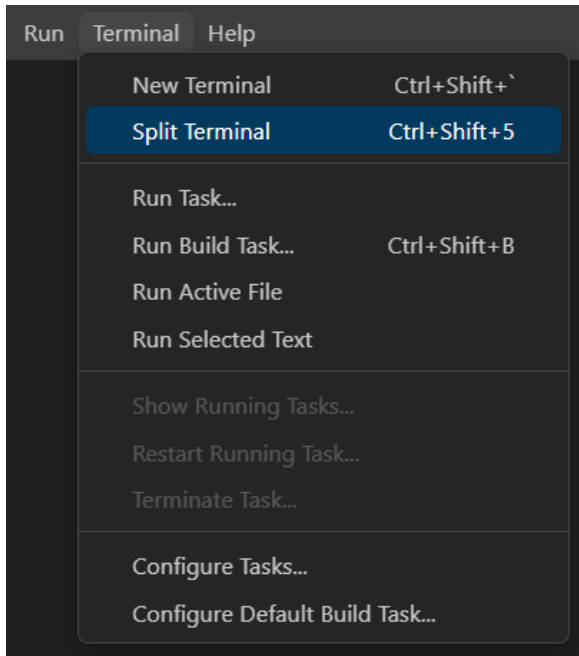
- **mlflow.sklearn.log\_model():** to log a scikit-learn model as an MLflow artifact for the current run

```
mlflow.sklearn.log_model(sk_model = rf, artifact_path= "trained_model")
```

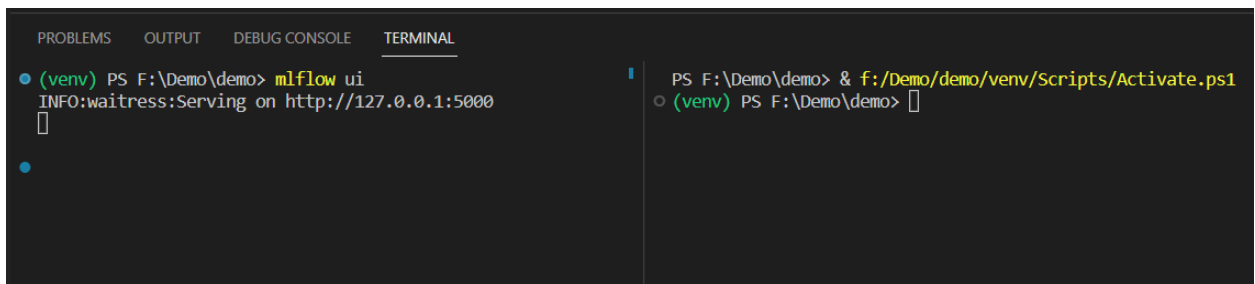
- **mlflow.end\_run():** to end an active MLflow run

```
mlflow.end_run()
```

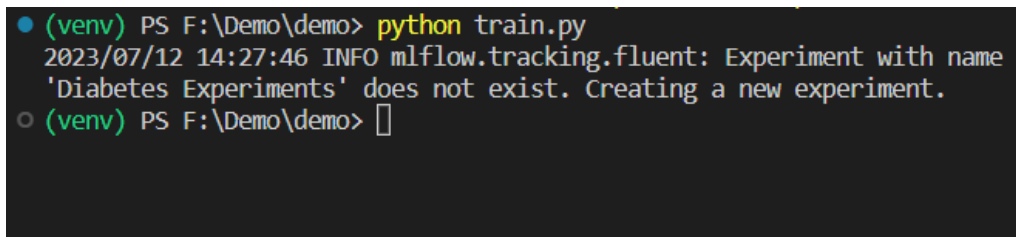
9. Now, on VS-Code, you already have mlflow ui running on one terminal. Go to *Terminal > Split Terminal*.



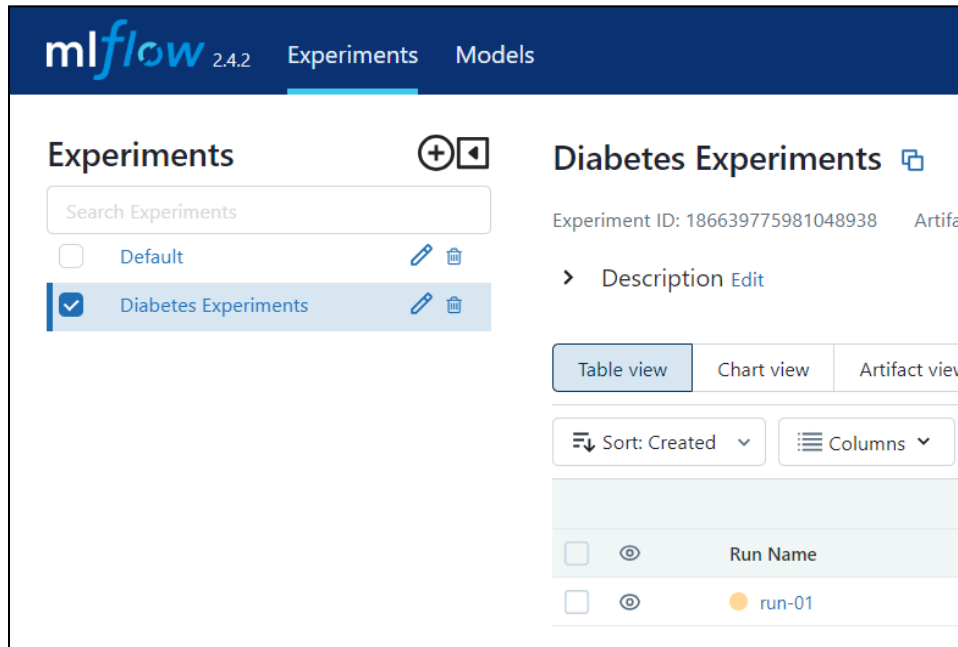
You will see another terminal enabled.



10. Use the second terminal to install the dependencies, and run the train.py file.



- Once trained, visit the mlflow UI and refresh the page.  
Select your newly created experiment.



mlflow 2.4.2 Experiments Models

### Experiments

Search Experiments

- ☐ Default
- ☒ Diabetes Experiments

### Diabetes Experiments

Experiment ID: 186639775981048938

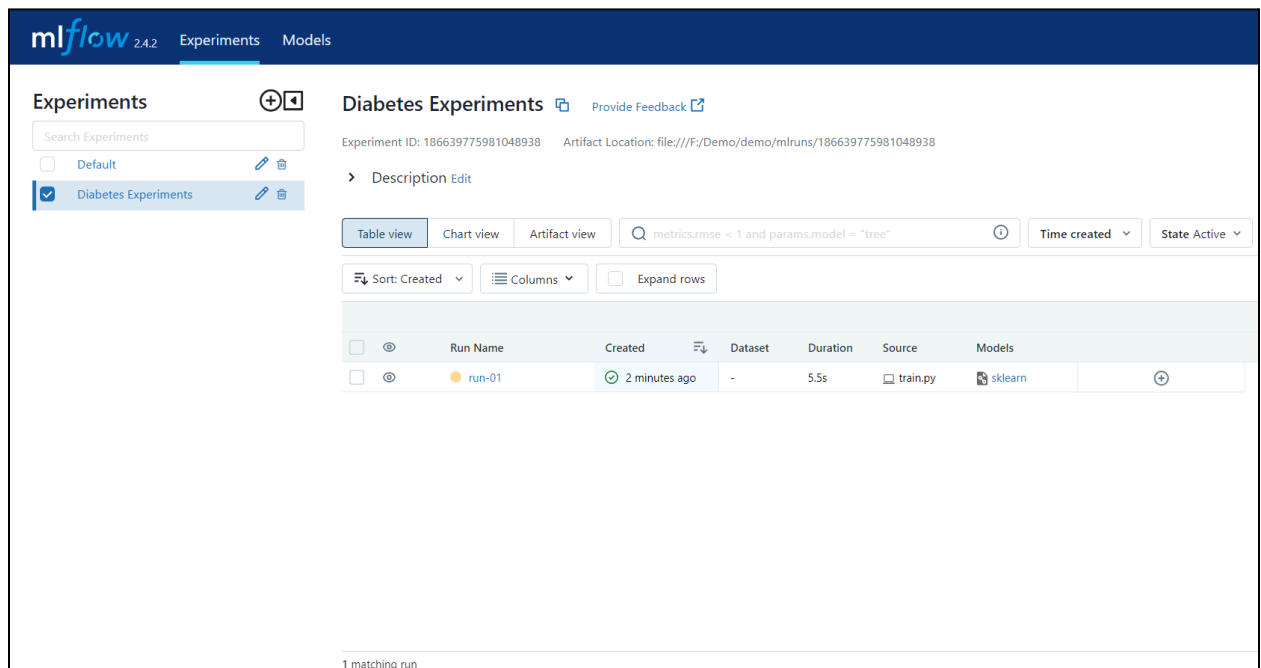
Description Edit

Table view Chart view Artifact view

Sort: Created Columns

Run Name
run-01

Runs under the experiment should also be listed, along with the details.



mlflow 2.4.2 Experiments Models

### Experiments

Search Experiments

- ☐ Default
- ☒ Diabetes Experiments

### Diabetes Experiments

Experiment ID: 186639775981048938

Description Edit

Table view Chart view Artifact view

Sort: Created Columns Expand rows

Run Name	Created	Dataset	Duration	Source	Models
run-01	2 minutes ago	-	5.5s	train.py	sklearn

1 matching run



## Diabetes Experiments [Provide Feedback](#)

Experiment ID: 186639775981048938    Artifact Location: file:///F:/Demo/demo/mlruns/186639775981048938

> Description [Edit](#)

Table view
Chart view
Artifact view

Time created
State Active

Sort: Created
Columns
Expand rows

<input type="checkbox"/>	<input type="checkbox"/>	Run Name	Created	Dataset	Duration	Source	Models	
<input type="checkbox"/>	<input type="checkbox"/>	run-01	2 minutes ago	-	5.5s	train.py	sklearn	<a href="#">+</a>

The main portion of the window shows a table of *runs*, with each row representing a single *run*. The columns show the *run name*, how long ago it was created, its *running time*, and so forth.

If you select a *run* name, you will open details for the *run*, which shows the parameters, metrics, and artifacts of the *run*.

mlflow 2.4.2

Experiments Models

Diabetes Experiments >

### run-01

Run ID: 5580d455f467438d89cbe7666dce110c    Date: 2023-07-12 14:27:46    Source: train.py

Duration: 5.5s    Status: FINISHED    Lifecycle Stage: active

> Description [Edit](#)

> Datasets

> Parameters (3)

> Metrics (4)

> Tags

▼ Artifacts

▶ trained\_model

Full Path:file:///F:/Demo/demo/mlruns/186639775981048938/5580d455f467438d89cb

12. Now, in VS-code, change the hyperparameter values in the train.py file.

For example:

```
# Parameters
N_ESTIMATORS = 100
MAX_DEPTH = 5
MAX_FEATURES = 4
# Run name
RUN_NAME = "run-02"
```

13. Re-run the train.py file.

14. Visit the mlflow UI, and refresh. New runs should be listed.

<input type="checkbox"/>	<input type="checkbox"/>	Run Name	Created	Dataset	Duration	Source	Models
<input type="checkbox"/>	<input type="checkbox"/>	run-02	✓ 31 seconds ago	-	4.7s	train.py	sklearn
<input type="checkbox"/>	<input type="checkbox"/>	run-01	✓ 32 minutes ago	-	5.5s	train.py	sklearn

15. Repeat the steps with different hyperparameters values and run the train file.

16. Then, go to the *Chart view* tab.

Table view

Chart view

Artifact view

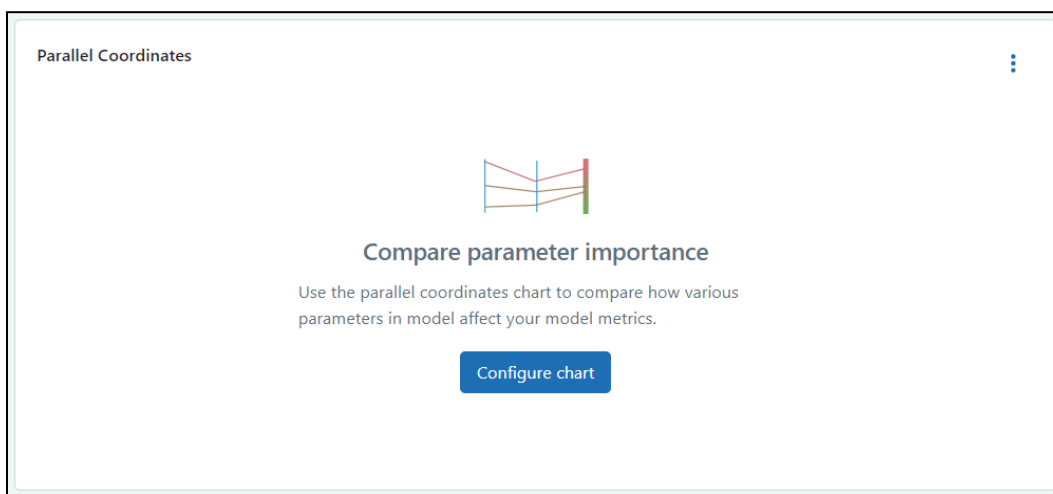
🔍 metrics.rmse < 1 and params.model = "tree" ⓘ

⌵ Sort: Created ⌵

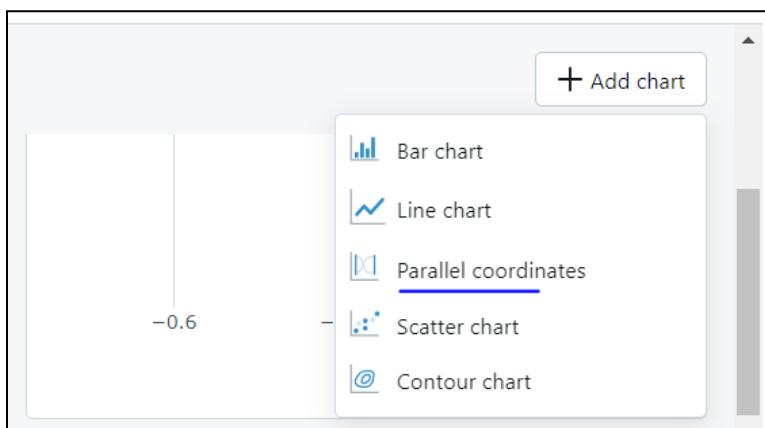
⌵ Columns ⌵

☐ Expand rows

<input type="checkbox"/>	<input type="checkbox"/>	Run Name	Created	Dataset	Duration	Source	Models
<input type="checkbox"/>	<input type="checkbox"/>	<span>●</span> run-05	<input checked="" type="checkbox"/> 12 seconds ago	-	4.8s	train.py	sklearn
<input type="checkbox"/>	<input type="checkbox"/>	<span>●</span> run-04	<input checked="" type="checkbox"/> 54 seconds ago	-	4.9s	train.py	sklearn
<input type="checkbox"/>	<input type="checkbox"/>	<span>●</span> run-03	<input checked="" type="checkbox"/> 2 minutes ago	-	5.2s	train.py	sklearn
<input type="checkbox"/>	<input type="checkbox"/>	<span>●</span> run-02	<input checked="" type="checkbox"/> 7 minutes ago	-	4.7s	train.py	sklearn
<input type="checkbox"/>	<input type="checkbox"/>	<span>●</span> run-01	<input checked="" type="checkbox"/> 39 minutes ago	-	5.5s	train.py	sklearn

17. Select *Configure chart*


OR

Select *Add chart > Parallel coordinates*.

## 18. Select at least two metrics and two params first.

### Add new chart

Type:

 Parallel coordinates ▼

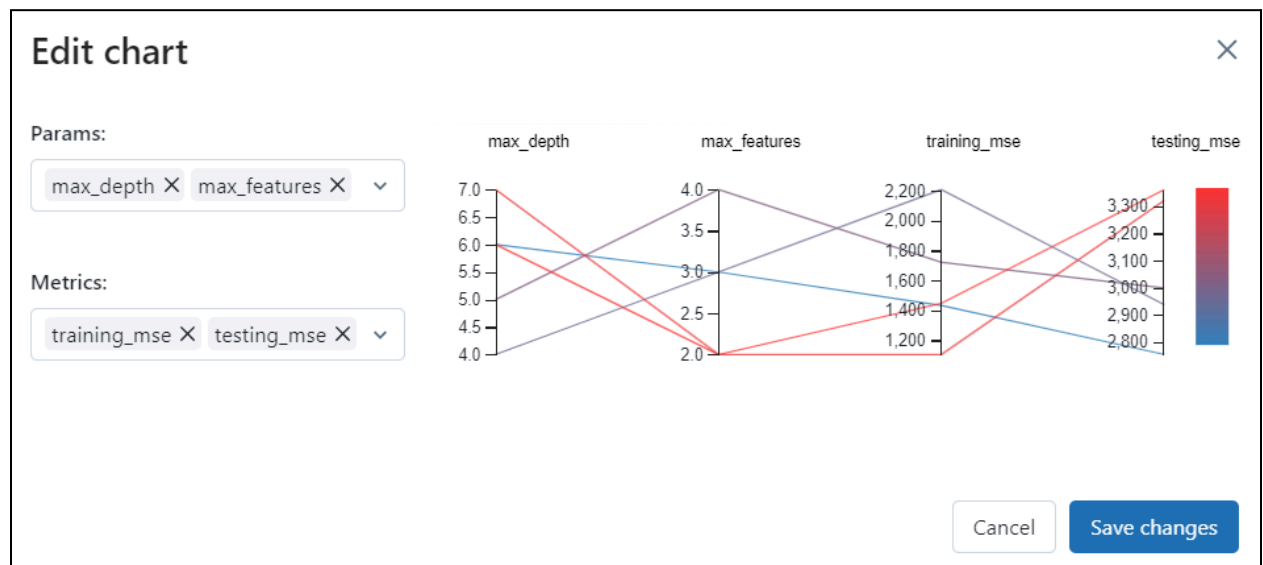
Params:

▼

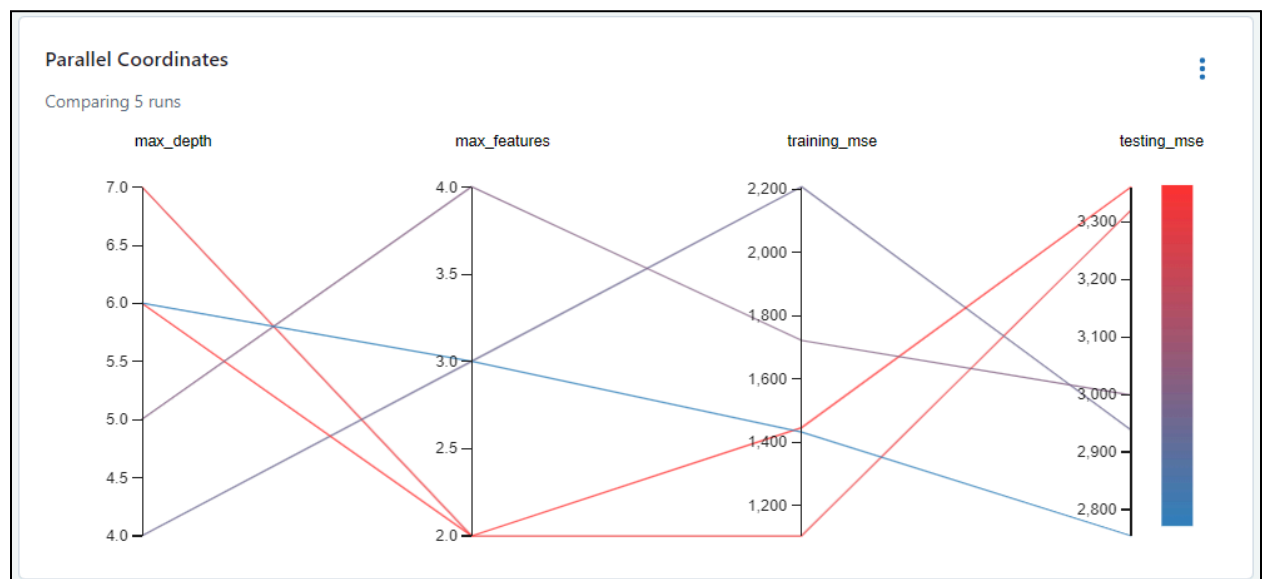
Metrics:

▼

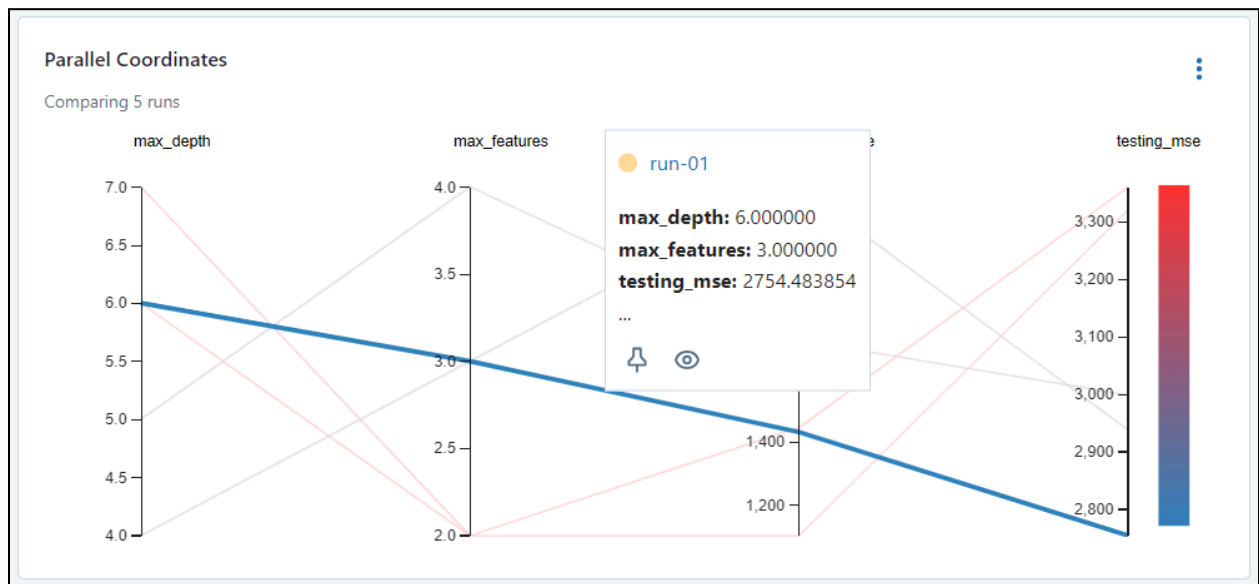
19. Save changes.



20. An interactive chart will be enabled to compare runs.

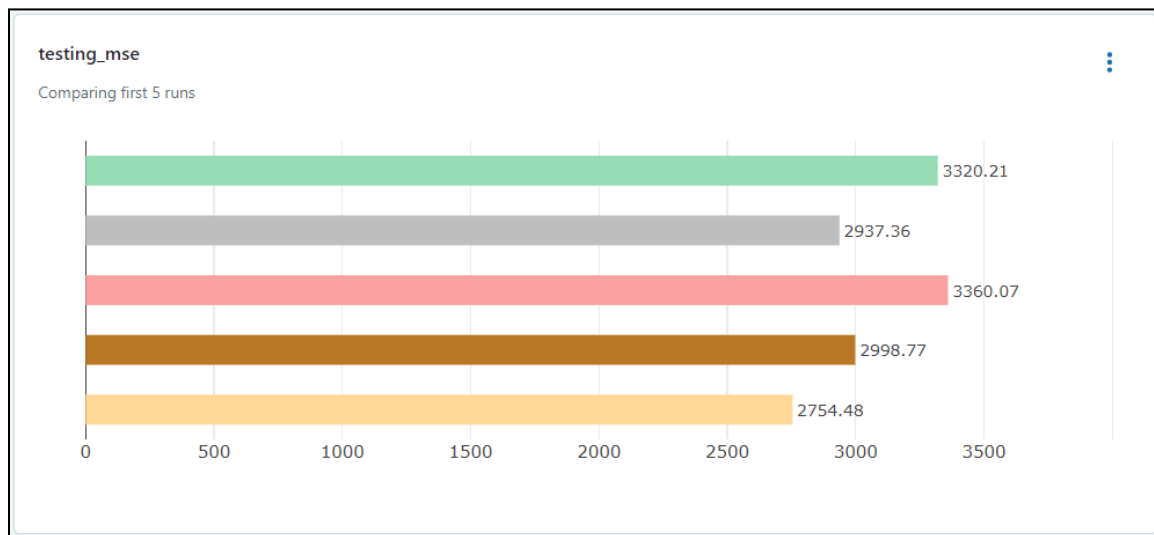


21. Hover over the lines to see with which run it is associated.



Seems like the first run has the lowest test mse value among all the runs.

22. You can configure more charts of your choice.

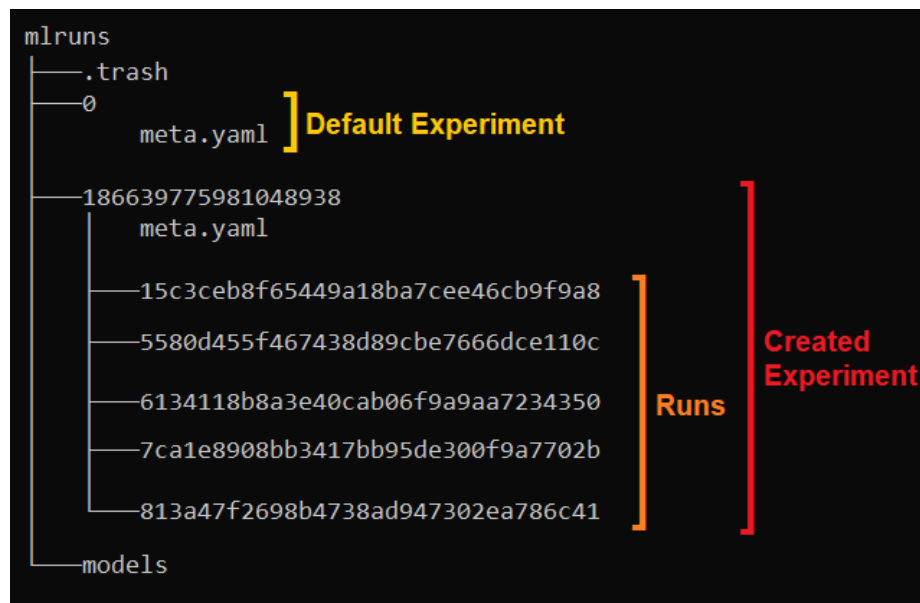


23. Once done, close the mlflow UI from the terminal by pressing Ctrl+C.

```
(venv) PS F:\Demo\demo> mlflow ui
INFO:waitress:erving on http://127.0.0.1:5000

Aborted!
(venv) PS F:\Demo\demo> 
```

24. Note that the experiment and details of the runs are still saved in the '**mlruns**' directory on your system. You can launch the mlflow UI again to visualize them.



#### References:

- [MLflow documentation](#)