

# Data Science documentation in Microsoft Fabric

Learn more about the Data Science experience in Microsoft Fabric

## About Data Science



### OVERVIEW

[What is Data Science?](#)

## Get started



### GET STARTED

[Introduction to Data science tutorials](#)

[Data science roles and permissions](#)

[Machine learning models](#)

## Tutorials



### TUTORIAL

[How to use end-to-end AI samples](#)

[Train a retail recommendation model](#)

[Fraud detection](#)

[Forecasting](#)

[Text classification](#)

[Healthcare causal impact of treatments](#)

## Preparing Data



### HOW-TO GUIDE

[How to accelerate data prep with Data Wrangler](#)

[How to read and write data with Pandas](#)

## Training Models

### OVERVIEW

[Training Overview](#)

### HOW-TO GUIDE

[Train with Spark MLlib](#)

[Train models with Scikit-learn](#)

[Train models with PyTorch](#)

## Tracking models and experiments

### HOW-TO GUIDE

[Machine learning experiments](#)

[Fabric autologging](#)

## Using SynapseML

### GET STARTED

[Your First SynapseML Model](#)

### HOW-TO GUIDE

[Use Cognitive Services with SynapseML](#)

[Deep learning with ONNX and SynapseML](#)

[Perform multivariate anomaly detection with SynapseML](#)

[Tune hyperparameters with SynapseML](#)

# Using R

---

## OVERVIEW

[R overview](#)

---

## HOW-TO GUIDE

[How to use SparkR](#)

[How to use sparklyr](#)

[R library management](#)

[R visualization](#)

# What is Data Science in Microsoft Fabric?

Article • 05/23/2023

## ⓘ Important

Microsoft Fabric is currently in PREVIEW. This information relates to a prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, expressed or implied, with respect to the information provided here.

Microsoft Fabric offers Data Science experiences to empower users to complete end-to-end data science workflows for the purpose of data enrichment and business insights. You can complete a wide range of activities across the entire data science process, all the way from data exploration, preparation and cleansing to experimentation, modeling, model scoring and serving of predictive insights to BI reports.

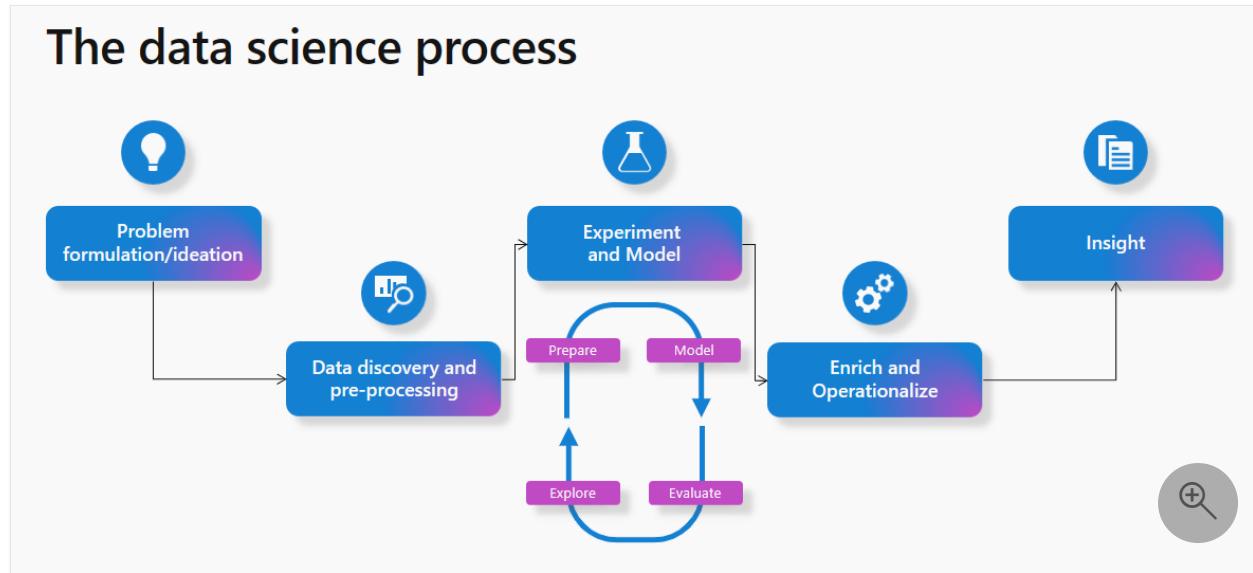
Microsoft Fabric users can access a Data Science Home page. From there, they can discover and access various relevant resources. For example, they can create machine learning Experiments, Models and Notebooks. They can also import existing Notebooks on the Data Science Home page.

The screenshot shows the Microsoft Fabric Data Science Home page. At the top, under the 'New' section, there are five buttons: 'Model (Preview)', 'Experiment (Preview)', 'Notebook (Preview)', 'Import notebook', and 'Use a sample'. Below this is a 'Recommended' section with four cards, each labeled 'You frequently open this': 'fraud-detection-multirun', 'Data Science WS', 'customer-churn', and 'ML Notebook'. At the bottom is a 'Quick access' section with buttons for 'Recent', 'Favorites', and 'Shared with me', and columns for 'Name', 'Type', 'Opened', 'Owner', 'Endorsement', and 'Sensitivity'.

You might know how a typical data science process works. As a well-known process, most machine learning projects follow it.

At a high level, the process involves these steps:

- Problem formulation and ideation
- Data discovery and pre-processing
- Experimentation and modeling
- Enrich and operationalize
- Gain insights



This article describes the Microsoft Fabric Data Science capabilities from a data science process perspective. For each step in the data science process, this article summarizes the Microsoft Fabric capabilities that can help.

## Problem formulation and ideation

Data Science users in Microsoft Fabric work on the same platform as business users and analysts. Data sharing and collaboration becomes more seamless across different roles as a result. Analysts can easily share Power BI reports and datasets with data science practitioners. The ease of collaboration across roles in Microsoft Fabric makes hand-offs during the problem formulation phase much easier.

## Data Discovery and pre-processing

Microsoft Fabric users can interact with data in OneLake using the Lakehouse item. Lakehouse easily attaches to a Notebook to browse and interact with data.

Users can easily read data from a Lakehouse directly into a Pandas dataframe. For exploration, this makes seamless data reads from One Lake possible.

There's a powerful set of tools available for data ingestion and data orchestration pipelines with data integration pipelines - a natively integrated part of Microsoft Fabric. Easy-to-build data pipelines can access and transform the data into a format that machine learning can consume.

## Data exploration

An important part of the machine learning process is to understand data through exploration and visualization.

Depending on the data storage location, Microsoft Fabric offers a set of different tools to explore and prepare the data for analytics and machine learning. Notebooks become one of the quickest ways to get started with data exploration.

## Apache Spark and Python for data preparation

Microsoft Fabric offers capabilities to transform, prepare, and explore your data at scale. With Spark, users can leverage PySpark/Python, Scala, and SparkR/SparklyR tools for data pre-processing at scale. Powerful open-source visualization libraries can enhance the data exploration experience to help better understand the data.

## Data Wrangler for seamless data cleansing

The Microsoft Fabric Notebook experience added a feature to use Data Wrangler, a code tool that prepares data and generates Python code. This experience makes it easy to accelerate tedious and mundane tasks - for example, data cleansing, and build repeatability and automation through generated code. Learn more about Data Wrangler in the Data Wrangler section of this document.

## Experimentation and ML modeling

With tools like PySpark/Python, SparklyR/R, notebooks can handle machine learning model training.

ML algorithms and libraries can help train machine learning models. Library management tools can install these libraries and algorithms. Users have therefore the option to leverage a large variety of popular machine learning libraries to complete their ML model training in Microsoft Fabric.

Additionally, popular libraries like Scikit Learn can also develop models.

MLflow experiments and runs can track the ML model training. Microsoft Fabric offers a built-in MIFlow experience with which users can interact, to log experiments and models. Learn more about how to use MLflow to track experiments and manage models in Microsoft Fabric.

## SynapseML

The SynapseML (previously known as MMLSpark) open-source library, that Microsoft owns and maintains, simplifies massively scalable machine learning pipeline creation. As a tool ecosystem, it expands the Apache Spark framework in several new directions. SynapseML unifies several existing machine learning frameworks and new Microsoft algorithms into a single, scalable API. The open-source SynapseML library includes a rich ecosystem of ML tools for development of predictive models, as well as leveraging pre-trained AI models from Azure Cognitive Services. Learn more about [SynapseML](#).

## Enrich and operationalize

Notebooks can handle machine learning model batch scoring with open-source libraries for prediction, or the Microsoft Fabric scalable universal Spark Predict function, which supports mlflow packaged models in the Microsoft Fabric model registry.

## Gain insights

In Microsoft Fabric, Predicted values can easily be written to OneLake, and seamlessly consumed from Power BI reports, with the Power BI Direct Lake mode. This makes it very easy for data science practitioners to share results from their work with stakeholders and it also simplifies operationalization.

Notebooks that contain batch scoring can be scheduled to run using the Notebook scheduling capabilities. Batch scoring can also be scheduled as part of data pipeline activities or Spark jobs. Power BI automatically gets the latest predictions without need for loading or refresh of the data, thanks to the Direct lake mode in Microsoft Fabric.

## Next steps

- Get started with end-to-end data science samples, see [Data Science Tutorials](#)
- Learn more about data preparation and cleansing with Data Wrangler, see [Data Wrangler](#)
- Learn more about tracking experiments, see [Machine learning experiment](#)
- Learn more about managing models, see [Machine learning model](#)

- Learn more about batch scoring with Predict, see [Score models with PREDICT](#)
- Serve predictions from Lakehouse to Power BI with [Direct lake Mode](#)

# Data science end-to-end scenario: introduction and architecture

Article • 05/23/2023

In this set of tutorials, we demonstrate a sample end-to-end scenario in the Fabric data science experience by implementing each step from data ingestion, cleansing, and preparation, to training machine learning models and generating insights, and then consuming those insights using visualization tools like Power BI. If you're new to Microsoft Fabric, see [What is Microsoft Fabric?](#).

## Important

Microsoft Fabric is currently in PREVIEW. This information relates to a prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, expressed or implied, with respect to the information provided here.

## Introduction

The lifecycle of a Data science project typically includes (often, iteratively) the following steps:

- Business understanding
- Data acquisition
- Data exploration, cleansing, preparation, and visualization
- Model training and experiment tracking
- Model scoring and generating insights.

The goals and success criteria of each stage depend on collaboration, data sharing and documentation. The Fabric Data science experience consists of multiple native-built features that enable collaboration, data acquisition, sharing, and consumption in a seamless way.

In this tutorial, you take the role of a data scientist who has been given the task to explore, clean, and transform a dataset containing taxicab trip data. You then build a machine learning model to predict trip duration at scale on a large dataset.

In this tutorial, you learn to perform the following activities:

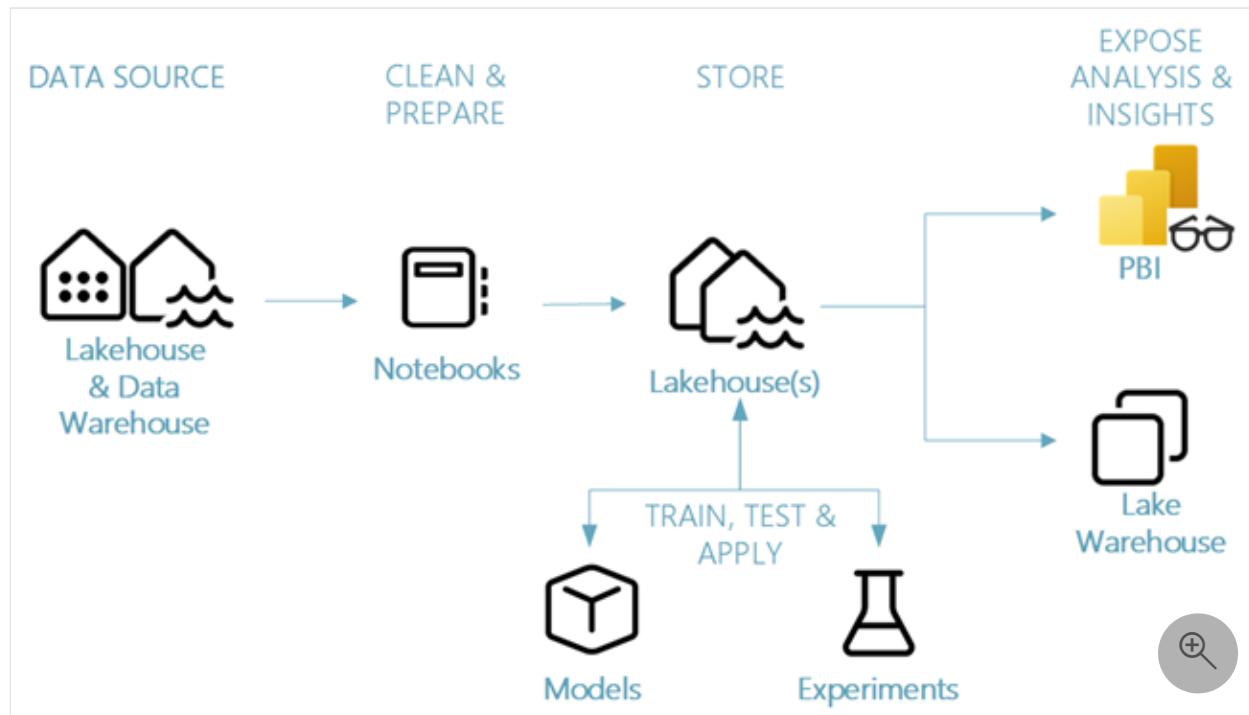
1. Use the Fabric notebooks for data science scenarios.

2. Ingest data into a Fabric lakehouse using Apache Spark.
3. Load existing data from the lakehouse delta tables.
4. Clean and transform Data using Apache Spark.
5. Create experiments and runs to Train a machine learning model.
6. Register and track trained models using MLflow and the Fabric UI.
7. Run scoring at scale and save predictions and inference results to the lakehouse.
8. Visualize predictions in Power BI using DirectLake.

## Architecture

In this tutorial, we showcase a simplified end-to-end data science scenario that involves:

1. Ingesting data from an external data source.
2. Data exploration and visualization.
3. Data cleansing, preparation, and feature engineering.
4. Model training and evaluation.
5. Model batch scoring and saving predictions for consumption.
6. Visualizing prediction results.



## Different components of the Data science scenario

**Data sources** - Fabric makes it easy and quick to connect to Azure Data Services, other cloud platforms, and on-premises data sources to ingest data from. Using Fabric Notebooks you can ingest data from the built in Lakehouse, Data Warehouse, Power BI Datasets and various Apache Spark and Python supported custom data sources. This tutorial focuses on ingesting and loading data from a lakehouse.

**Explore, clean, and prepare** - The Data science experience on Fabric supports data cleansing, transformation, exploration and featurization by using built-in experiences on Spark as well as Python based tools like Data Wrangler and SemPy Library. This tutorial will showcase data exploration using python library seaborn and data cleansing and preparation using Apache Spark.

**Models and experiments** - Fabric enables you to train, evaluate and score machine learning models by using built-in Experiment and Model items with seamless integration with\*\* [MLflow](#) for experiment tracking and model registration/deployment. Fabric also features capabilities for model prediction at scale (PREDICT) to gain and share business insights.

**Storage** - Fabric standardizes on [Delta Lake](#), which means all the engines of Fabric can interact with the same dataset stored in a lakehouse. This storage layer allows you to store both structured and unstructured data that support both file-based storage and tabular format. The datasets and files stored can be easily accessed via all Fabric experience items like notebooks and pipelines.

**Expose analysis and insights** - Data from a lakehouse can be consumed by Power BI, industry leading business intelligence tool, for reporting and visualization. Data persisted in the lakehouse can also be visualized in notebooks using Spark or Python native visualization libraries like matplotlib, seaborn, plotly, and more. Data can also be visualized using the SemPy library that supports built-in rich, task-specific visualizations for the semantic data model, for dependencies and their violations, and for classification and regression use cases.

## Next steps

- [Prepare your system for the Data science tutorial](#)

# Prepare your system for the Data science tutorial

Article • 05/23/2023

Before you begin the Data science end-to-end tutorial modules, learn about prerequisites, the sample dataset, which notebooks to import, and how to attach a lakehouse to those notebooks.

## ⓘ Important

Microsoft Fabric is currently in PREVIEW. This information relates to a prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, expressed or implied, with respect to the information provided here.

## Prerequisites

1. Power BI Premium subscription. For more information, see [How to purchase Power BI Premium](#).
2. A Power BI Workspace with assigned premium capacity.
3. An existing Microsoft Fabric lakehouse. Create a lakehouse by following the steps in [Create a lakehouse in Microsoft Fabric](#).

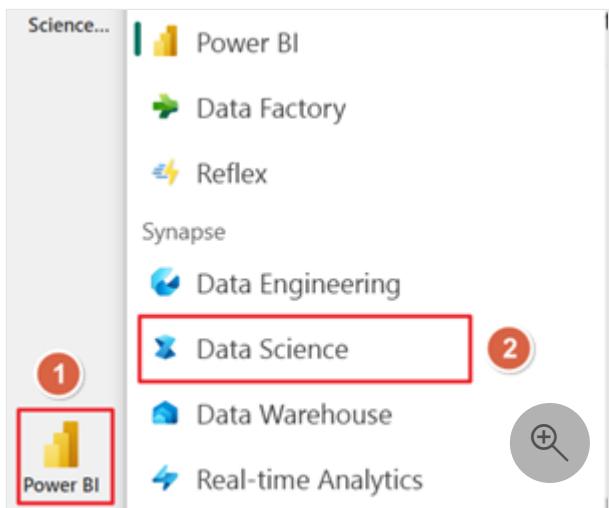
## Sample dataset

In this tutorial, we use the [NYC Taxi and Limousine yellow dataset](#), which is a large-scale dataset containing taxi trips in the city from 2009 to 2018. The dataset includes various features such as pick-up and drop-off dates, times, locations, fares, payment types, and passenger counts. The dataset can be used for various purposes such as analyzing traffic patterns, demand trends, pricing strategies, and driver behavior.

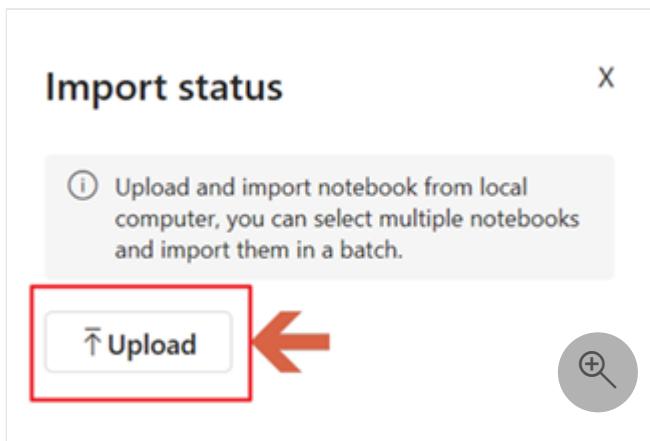
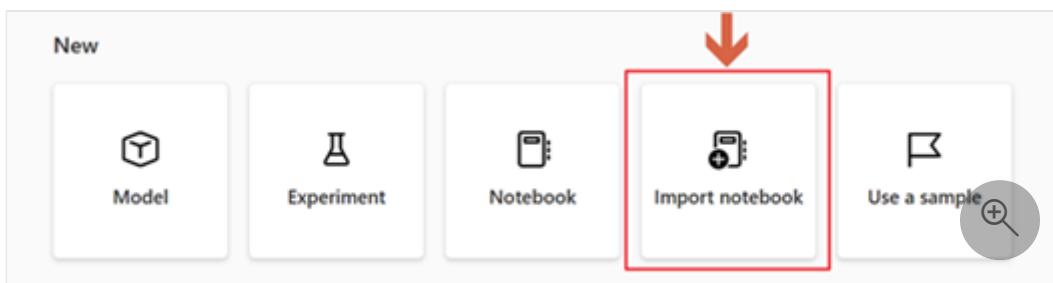
## Import tutorial notebooks

We utilize the notebook item in the Data Science experience to demonstrate various Fabric capabilities. The notebooks are available as Jupyter notebook files that can be imported to your Fabric-enabled workspace.

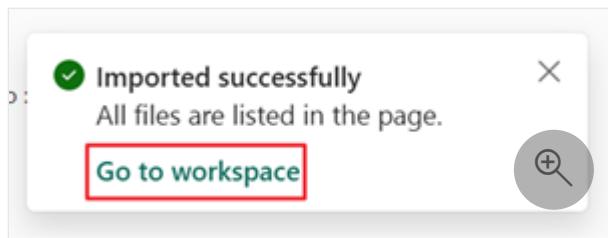
1. Download the notebooks(.ipynb) files for this tutorial from the parent folder [Data Science Tutorial Source Code](#).
2. Switch to the Data Science experience using the experience switcher icon at the left corner of your homepage.



3. On the Data science experience homepage, select **Import notebook** and upload the notebook files for modules 1- 5 that you downloaded in step 1.



4. Once the notebooks are imported, select **Go to workspace** in the import dialog box.



5. The imported notebooks are now available in your workspace for use.

	Name	Type	Owner
01	01 - Ingest data into Trident lakehouse using Apache Spark	Notebook	
02	02 - Explore and Visualize Data using Notebooks	Notebook	
03	03 - Perform Data Cleansing and preparation using Python	Notebook	
04	04 - Train and track machine learning models	Notebook	
05	05 - Perform batch scoring and save predictions to a database	Notebook	

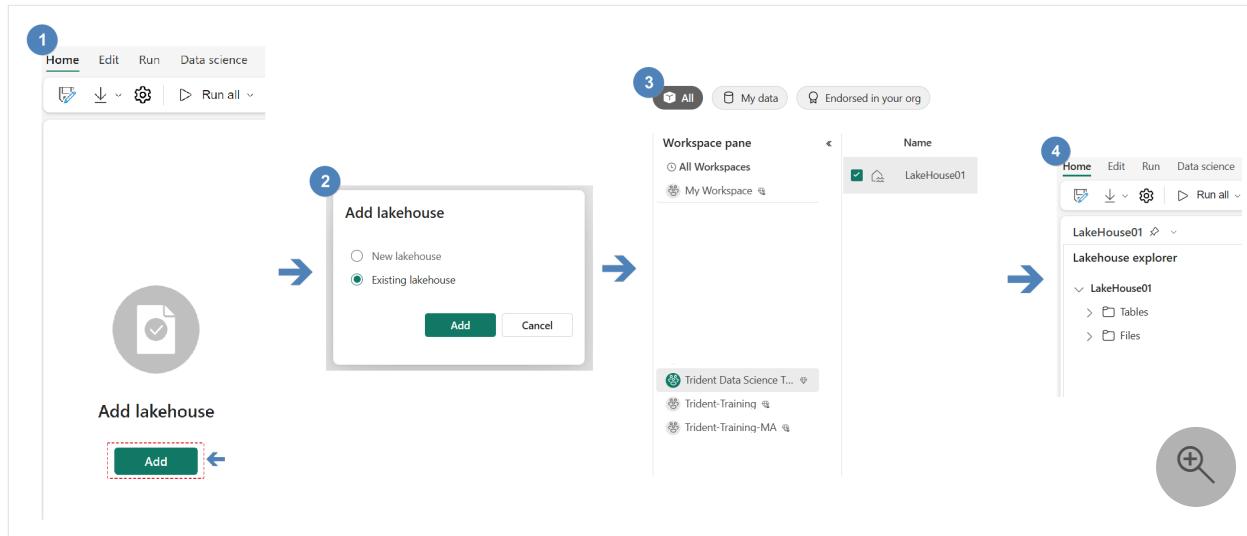
## Attach a lakehouse to the notebooks

To demonstrate Fabric lakehouse features, the first five modules in this tutorial require attaching a default lakehouse to the notebooks. The following steps show how to add an existing lakehouse to a notebook in a Fabric-enabled workspace.

1. Open the notebook for the first module **01 Ingest data into Lakehouse using Apache Spark** in the workspace.
2. Select **Add lakehouse** in the left pane and select **Existing lakehouse** to open the **Data hub** dialog box.
3. Select the workspace and the lakehouse you intend to use with these tutorials and select **Add**.
4. Once a lakehouse is added, it's visible in the lakehouse pane in the notebook UI where tables and files stored in the lakehouse can be viewed.

### ⓘ Note

Before executing all notebooks, you need to perform these steps for each notebook in this tutorial.



## Next steps

- [Module 1: Ingest data into Fabric lakehouse using Apache Spark](#)

# Module 1: Ingest data into a Microsoft Fabric lakehouse using Apache Spark

Article • 05/23/2023

In this module, we ingest the [NYC Taxi & Limousine Commission - yellow taxi trip dataset](#) to demonstrate data ingestion into Fabric lakehouses in delta lake format.

## Important

Microsoft Fabric is currently in PREVIEW. This information relates to a prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, expressed or implied, with respect to the information provided here.

**Lakehouse:** A lakehouse is a collection of files, folders, and tables that represents a database over a data lake used by the Spark engine and SQL engine for big data processing, and that includes enhanced capabilities for ACID transactions when using the open-source Delta formatted tables.

**Delta Lake:** Delta Lake is an open-source storage layer that brings ACID transactions, scalable metadata management, and batch and streaming data processing to Apache Spark. A Delta Lake table is a data table format that extends Parquet data files with a file-based transaction log for ACID transactions and scalable metadata management.

In the following steps, you use the Apache spark to read data from Azure Open Datasets containers and write data into a Fabric lakehouse delta table. [Azure Open Datasets](#) are curated public datasets that you can use to add scenario-specific features to machine learning solutions for more accurate models. Open Datasets are in the cloud on Microsoft Azure Storage and can be accessed by various methods including Apache Spark, REST API, Data factory, and other tools.

## Follow along in notebook

The python commands/script used in each step of this tutorial can be found in the accompanying notebook: [01-ingest-data-into-fabric-lakehouse-using-apache-spark.ipynb ↗](#). Be sure to [attach a lakehouse to the notebook](#) before executing it.

## Ingest the data

1. In the first step of this module, we read data from "azureopendatastorage" storage container using anonymous since the container has public access. We load [yellow cab data](#) by specifying the directory and filter the data by year (puYear) and month (puMonth). In this tutorial, we try to minimize the amount of data ingested and processed to speed up the execution. To learn more about the data, see [NYC Taxi & Limousine Commission - yellow taxi trip dataset](#).

Python

```
# Azure storage access info for open datasets yellow cab
storage_account = "azureopendatastorage"
container = "nyctlc"

sas_token = r"" # Blank since container is Anonymous access

# Set Spark config to access blob storage
spark.conf.set("fs.azure.sas.%s.%s.blob.core.windows.net" % (container,
storage_account),sas_token)

dir = "yellow"
year = 2016
months = "1,2,3,4"
wasbs_path =
f"wasbs://{{container}}@{{storage_account}}.blob.core.windows.net/{{dir}}"
df = spark.read.parquet(wasbs_path)

# Filter data by year and months
filtered_df = df.filter(f"puYear = {year} AND puMonth IN ({months})")
```

2. Next, we set spark configurations to enable VOrder engine and Optimize delta writes.

- **VOrder** - Fabric includes Microsoft's VOrder engine. VOrder writer optimizes the Delta Lake parquet files resulting in 3x-4x compression improvement and up to 10x performance acceleration over Delta Lake files not optimized using VOrder while still maintaining full Delta Lake and PARQUET format compliance.
- **Optimize write** - Spark in Microsoft Fabric includes an Optimize write feature that reduces the number of files written and targets to increase individual file size of the written data. It dynamically optimizes files during write operations generating files with a default 128-MB size. The target file size may be changed per workload requirements using configurations.

These configs can be applied at a session level (as spark.conf.set in a notebook cell) as demonstrated in the following code cell, or at workspace

level, which is applied automatically to all spark sessions created in the workspace. In this tutorial, we set these configurations using the code cell.

Python

```
spark.conf.set("sprk.sql.parquet.vorder.enabled", "true") # Enable  
VOrder write  
spark.conf.set("spark.microsoft.delta.optimizeWrite.enabled",  
"true") # Enable automatic delta optimized write
```

### 💡 Tip

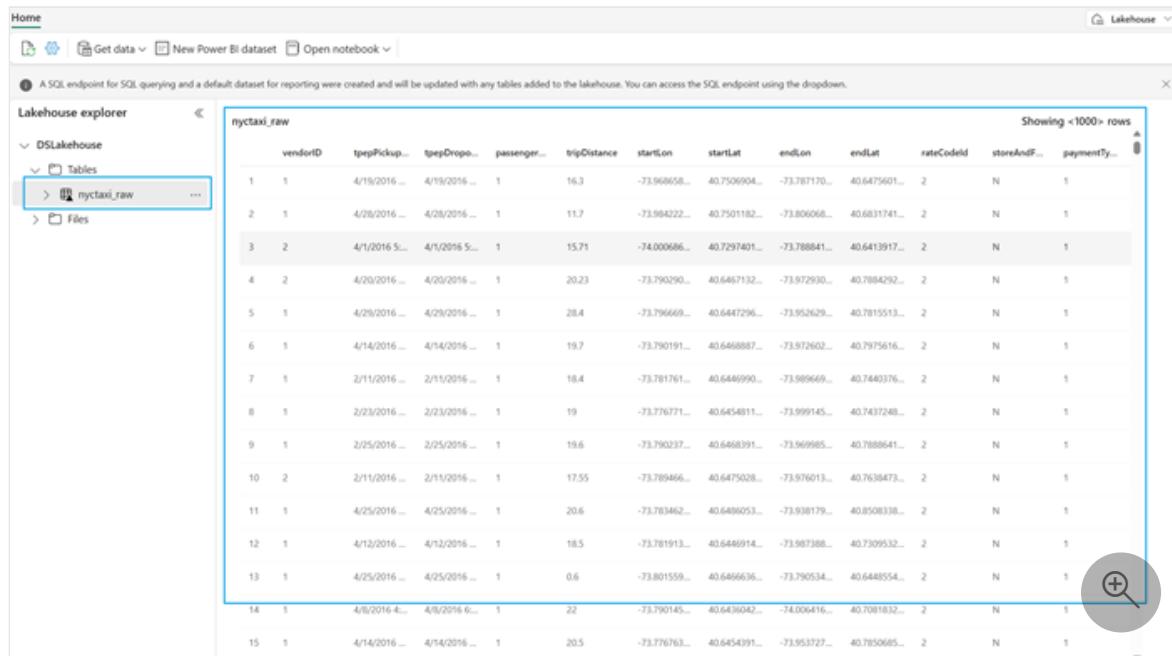
The workspace level Apache Spark configurations can be set at:  
**Workspace settings, Data Engineering/Science, Spark Compute, Spark Properties, Add.**

3. In the next step, perform a spark dataframe write operation to save data into a lakehouse table named *nyctaxi\_raw*.

Python

```
table_name = "nyctaxi_raw"  
filtered_df.write.mode("overwrite").format("delta").save(f"Tables/{table_name}")  
print(f"Spark dataframe saved to delta table: {table_name}")
```

Once the dataframe has been saved, you can navigate to the attached lakehouse item in your workspace and open the lakehouse UI to preview data in the *nyctaxi\_raw* table created in the previous steps.



The screenshot shows the Databricks Lakehouse UI interface. On the left, there's a sidebar titled 'Lakehouse explorer' with a tree view showing 'DSLakehouse' and 'Tables' (with 'nyctaxi\_raw' selected). The main area is a data preview table for the 'nyctaxi\_raw' table, showing 1000 rows of data with columns: vendorID, tpepPickup, tpepDropoff, passenger, tripDistance, startLon, startLat, endLon, endLat, rateCodeId, storeAndFwd, and paymentType. The data includes various taxi pickup and dropoff locations in New York City. A search bar with a magnifying glass icon is visible in the bottom right corner of the preview area.

# Next steps

- Module 2: Explore and visualize data using notebooks

# Module 2: Explore and visualize data using Microsoft Fabric notebooks

Article • 05/23/2023

The python runtime environment in Fabric notebooks comes with various preinstalled open-source libraries for building visualizations like matplotlib, seaborn, Plotly and more.

## Important

Microsoft Fabric is currently in PREVIEW. This information relates to a prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, expressed or implied, with respect to the information provided here.

In this module, we use seaborn, which is a Python data visualization library that provides a high-level interface for building visuals on dataframes and arrays. For more information about seaborn, see [seaborn: statistical data visualization](#).

In this tutorial you learn to perform the following actions:

1. Read data stored from a delta table in the lakehouse.
2. Generate a random sample of the dataframe.
3. Convert a Spark dataframe to Pandas dataframe, which python visualization libraries support.
4. Perform exploratory data analysis using seaborn on the New York taxi yellow cab dataset. We do this by visualizing a trip duration variable against other categorical and numeric variables.

## Follow along in notebook

The python commands/script used in each step of this tutorial can be found in the accompanying notebook; [02-explore-and-visualize-data-using-notebooks.ipynb](#). Be sure to [attach a lakehouse to the notebook](#) before executing it.

## Visualize and analyze

1. To get started, let's read the delta table (saved in module 1) from the lakehouse and create a pandas dataframe on a random sample of the data.

```
Python
```

```
data = spark.read.format("delta").load("Tables/nyctaxi_raw")
SEED = 1234
sampled_df = data.sample(True, 0.001, seed=SEED).toPandas()
```

### ⓘ Note

To minimize execution time, we are using a 1/1000 sample to explore and visualize ingested data.

2. Import required libraries and function required for visualizations, and set seaborn theme parameters that control aesthetics of the output visuals like style, color palette and size of the visual.

```
Python
```

```
import seaborn as sns
import matplotlib.pyplot as plt
import matplotlib.ticker as mticker
import numpy as np
sns.set_theme(style="whitegrid", palette="tab10", rc =
{'figure.figsize':(9,6)})
```

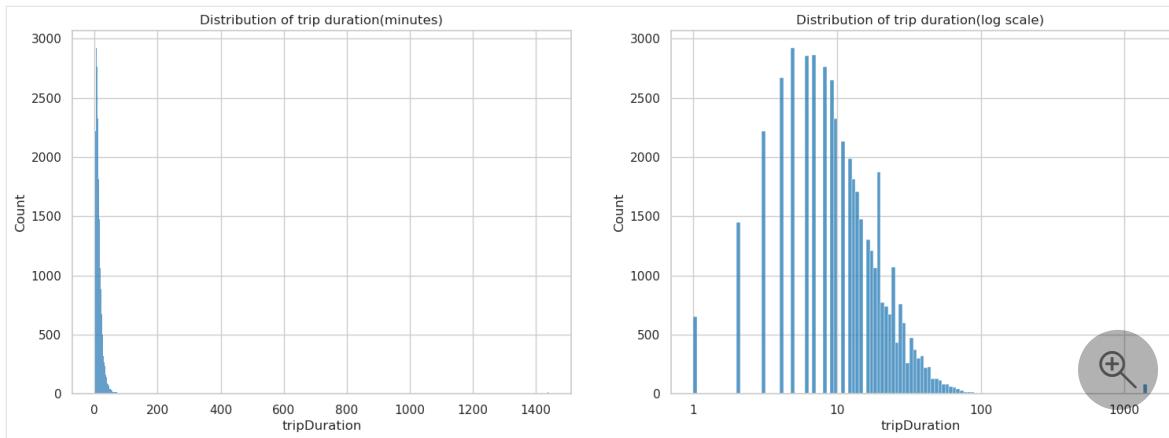
3. Visualize distribution of trip duration(minutes) on linear and logarithmic scale by running the following set of commands.

```
Python
```

```
## Compute trip duration(in minutes) on the sample using pandas
sampled_df['tripDuration'] = (sampled_df['tpepDropoffDateTime'] -
sampled_df['tpepPickupDateTime']).astype('timedelta64[m]')
sampled_df = sampled_df[sampled_df["tripDuration"] > 0]

fig, axes = plt.subplots(1, 2, figsize=(18, 6))
sns.histplot(ax=axes[0], data=sampled_df,
             x="tripDuration",
             stat="count",
             discrete=True).set(title='Distribution of trip
duration(minutes)')
sns.histplot(ax=axes[1], data=sampled_df,
             x="tripDuration",
             stat="count",
             log_scale= True).set(title='Distribution of trip
```

```
duration(log scale)')
axes[1].xaxis.set_major_formatter(mticker.ScalarFormatter())
```

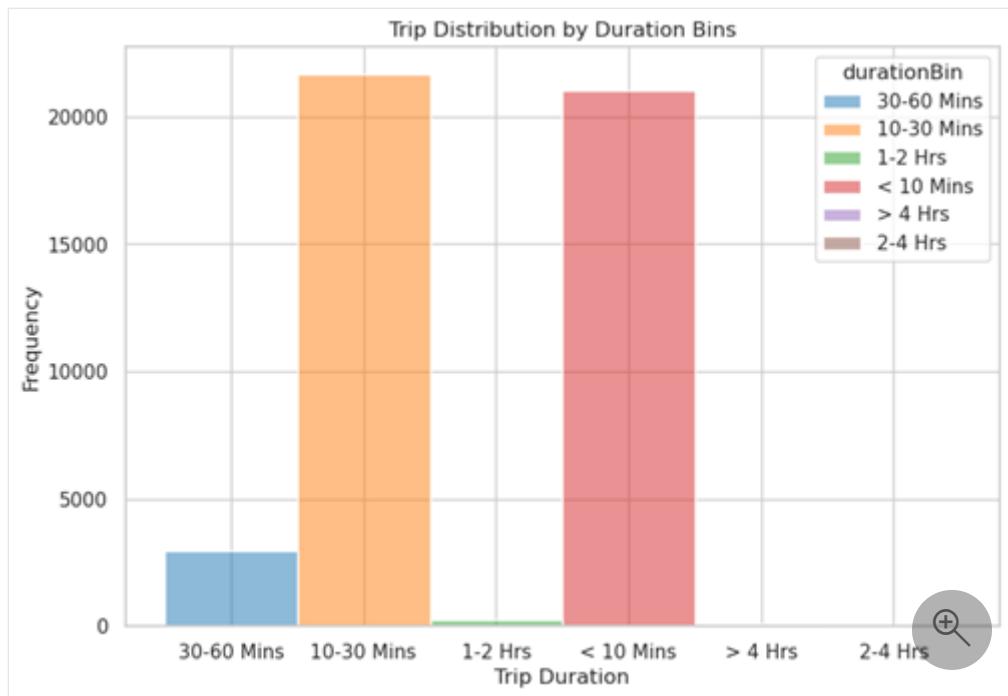


4. Create bins to segregate and understand distribution of tripDuration better. To do this, create a durationBin column using pandas operations to classify trip durations into buckets of **<10 Mins**, **10-30 Mins**, **30-60 Mins**, **1-2 Hrs**, **2-4 Hrs**, and **>4 Hrs**. Visualize the binned column using seaborn histogram plot.

Python

```
## Create bins for tripDuration column
sampled_df.loc[sampled_df['tripDuration'].between(0, 10, 'both'),
'durationBin'] = '< 10 Mins'
sampled_df.loc[sampled_df['tripDuration'].between(10, 30, 'both'),
'durationBin'] = '10-30 Mins'
sampled_df.loc[sampled_df['tripDuration'].between(30, 60, 'both'),
'durationBin'] = '30-60 Mins'
sampled_df.loc[sampled_df['tripDuration'].between(60, 120, 'right'),
'durationBin'] = '1-2 Hrs'
sampled_df.loc[sampled_df['tripDuration'].between(120, 240, 'right'),
'durationBin'] = '2-4 Hrs'
sampled_df.loc[sampled_df['tripDuration'] > 240, 'durationBin'] = '> 4 Hrs'

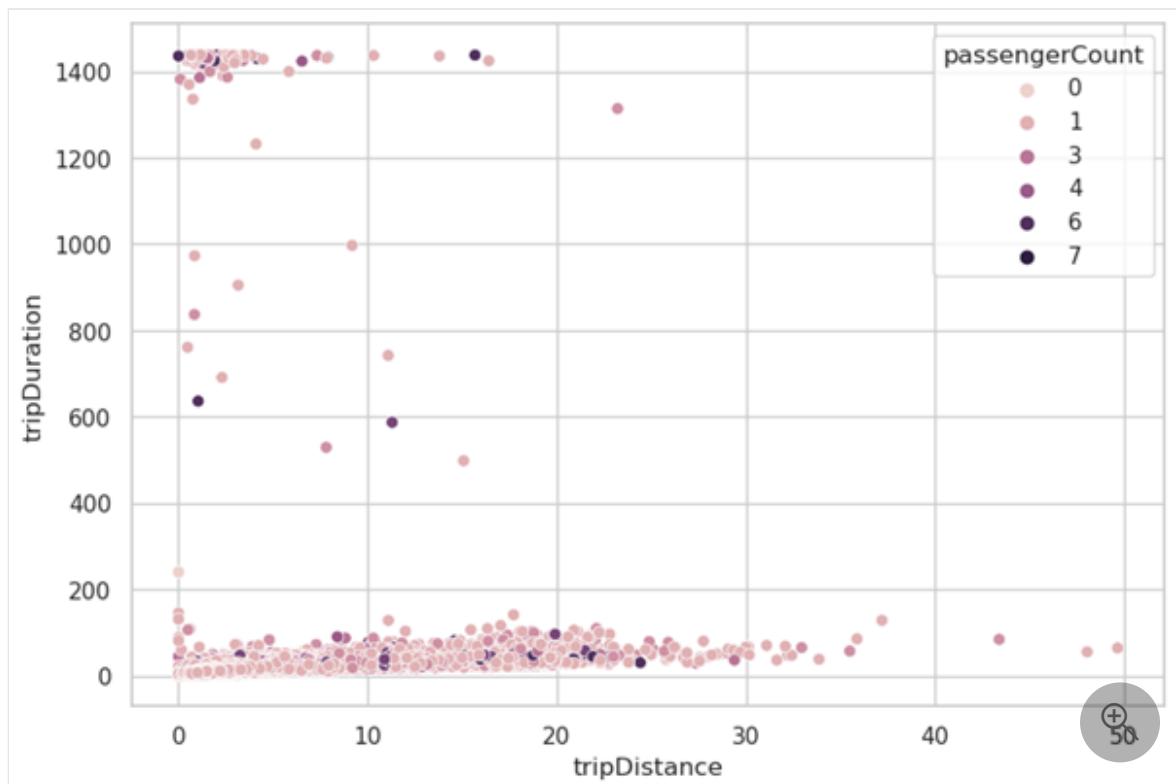
# Plot histogram using the binned column
sns.histplot(data=sampled_df, x="durationBin", stat="count",
discrete=True, hue = "durationBin")
plt.title("Trip Distribution by Duration Bins")
plt.xlabel('Trip Duration')
plt.ylabel('Frequency')
```



5. Visualize the distribution of `tripDuration` and `tripDistance` and classify by `passengerCount` using seaborn scatterplot by running below commands.

Python

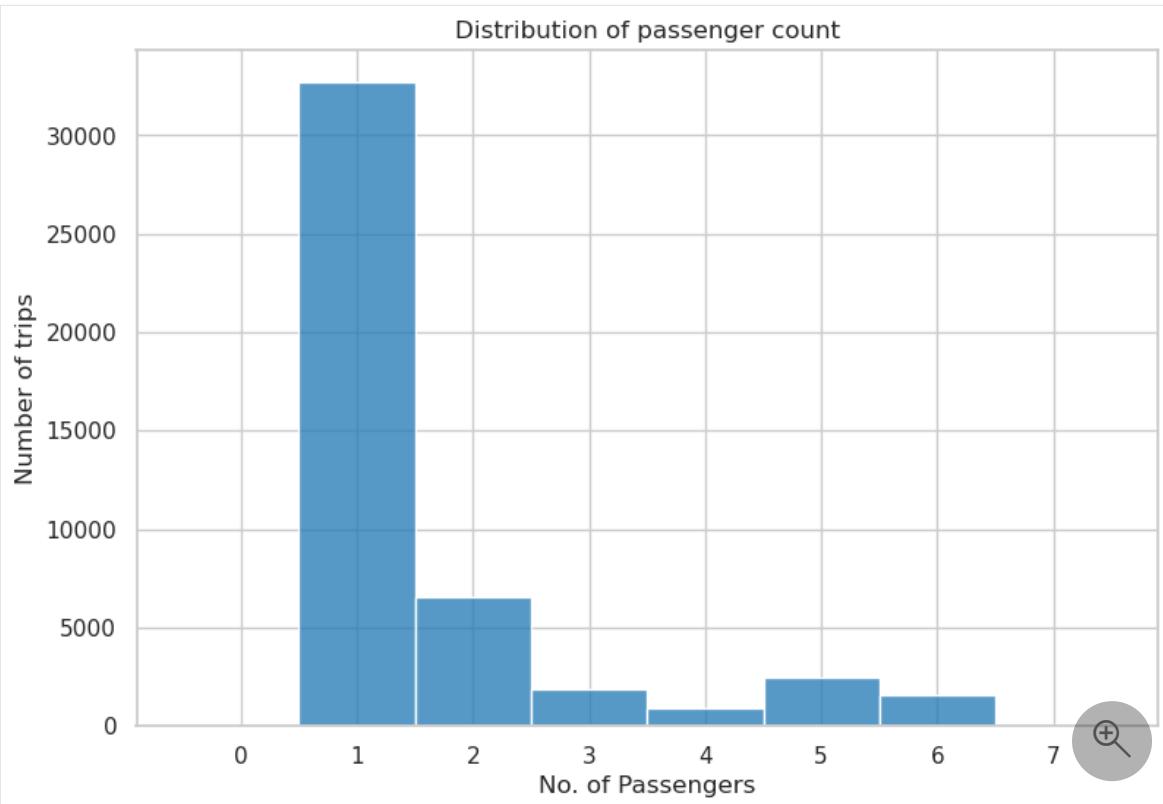
```
sns.scatterplot(data=sampled_df, x="tripDistance", y="tripDuration",
hue="passengerCount")
```



6. Visualize the overall distribution of `passengerCount` column to understand the most common `passengerCount` instances in the trips.

Python

```
sns.histplot(data=sampled_df, x="passengerCount", stat="count",
discrete=True)
plt.title("Distribution of passenger count")
plt.xlabel('No. of Passengers')
plt.ylabel('Number of trips')
```



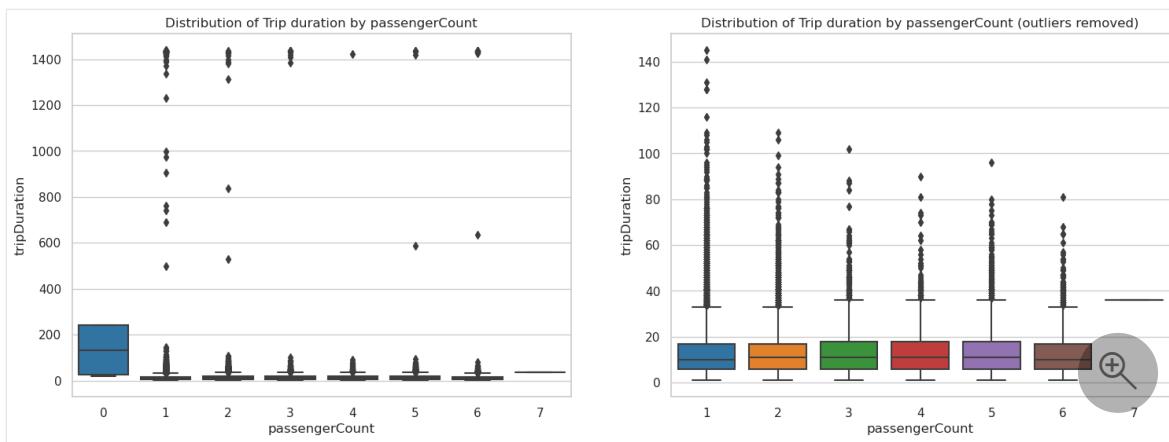
7. Create boxplots to visualize the distribution of *tripDuration* by passenger count. A boxplot is a useful tool to understand the variability, symmetry, and outliers of the data.

Python

```
# The threshold was calculated by evaluating mean trip duration (~15
# minutes) + 3 standard deviations (58 minutes), i.e. roughly 3 hours.
fig, axes = plt.subplots(1, 2, figsize=(18, 6))
sns.boxplot(ax=axes[0], data=sampled_df, x="passengerCount",
y="tripDuration").set(title='Distribution of Trip duration by
passengerCount')
sampleddf_clean = sampled_df[(sampled_df["passengerCount"] > 0) &
(sampled_df["tripDuration"] < 189)]
sns.boxplot(ax=axes[1], data=sampleddf_clean, x="passengerCount",
y="tripDuration").set(title='Distribution of Trip duration by
passengerCount (outliers removed)')
```

In the first figure, we visualize tripDuration without removing any outliers whereas in the second figure we're removing trips with duration greater than 3 hours and

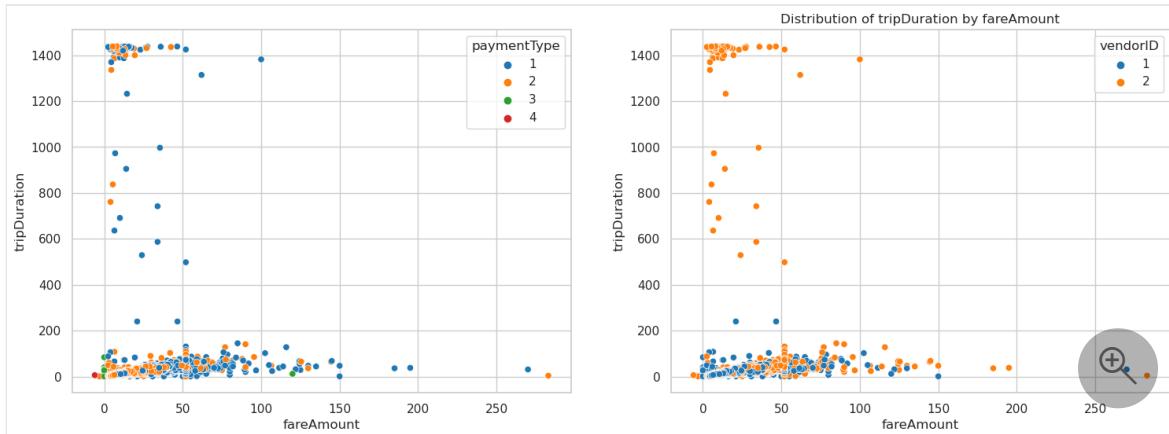
zero passengers.



8. Analyze the relationship of *tripDuration* and *fareAmount* classified by *paymentType* and *VendorID* using seaborn scatterplots.

Python

```
f, axes = plt.subplots(1, 2, figsize=(18, 6))
sns.scatterplot(ax =axes[0], data=sampled_df, x="fareAmount",
y="tripDuration", hue="paymentType")
sns.scatterplot(ax =axes[1], data=sampled_df, x="fareAmount",
y="tripDuration", hue="vendorID")
plt.title("Distribution of tripDuration by fareAmount")
plt.show()
```



9. Analyze the frequency of the taxi trips by hour of the day using a histogram of trip counts.

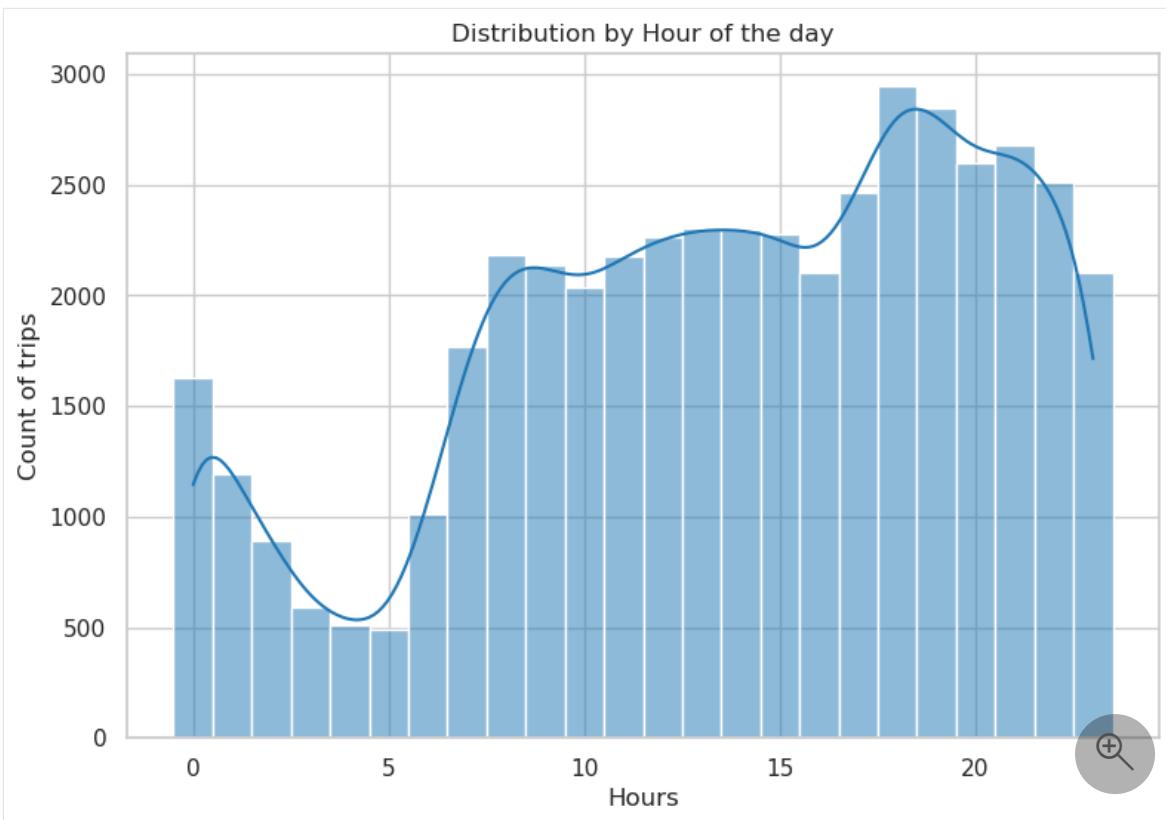
Python

```
sampled_df['hour'] = sampled_df['tpepPickupDateTime'].dt.hour
sampled_df['dayofweek'] =
sampled_df['tpepDropoffDateTime'].dt.dayofweek
sampled_df['dayname'] = sampled_df['tpepDropoffDateTime'].dt.day_name()
sns.histplot(data=sampled_df, x="hour", stat="count", discrete=True,
kde=True)
```

```

plt.title("Distribution by Hour of the day")
plt.xlabel('Hours')
plt.ylabel('Count of trips')
plt.show()

```



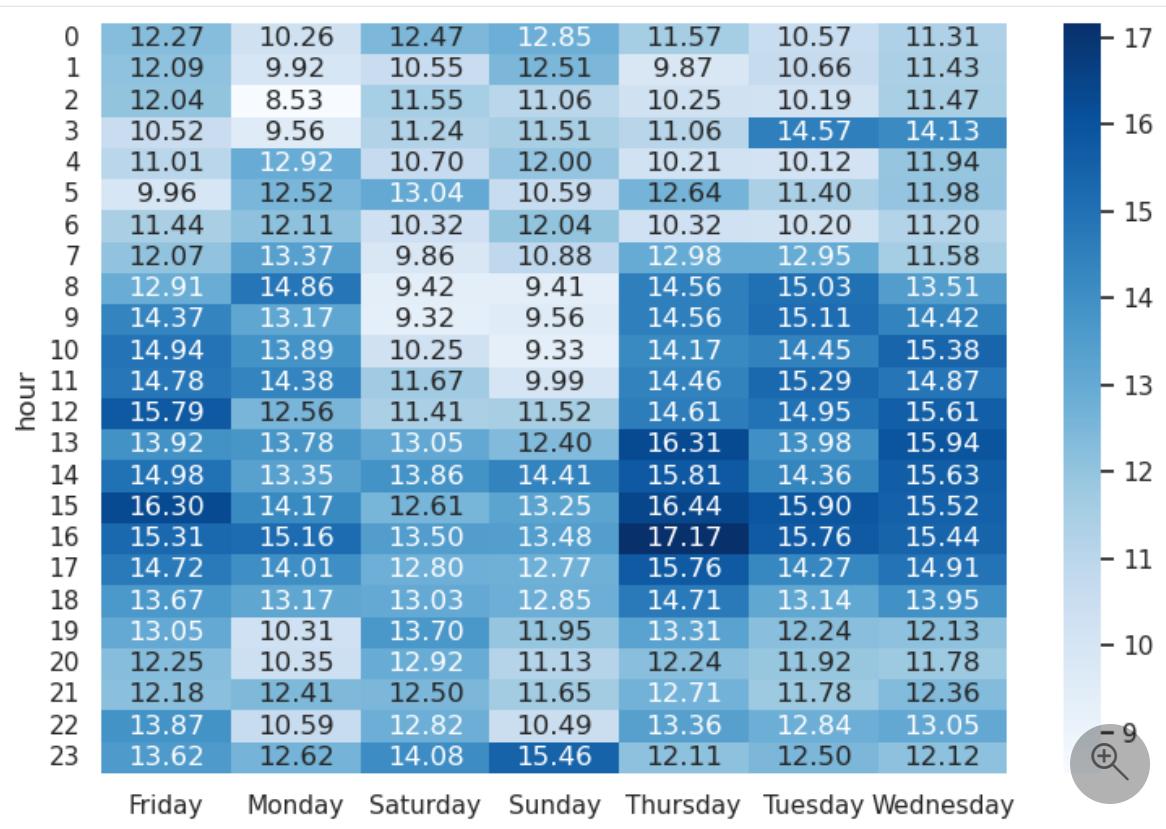
10. Analyze average taxi trip duration by hour and day together by using a seaborn heatmap. The below cell creates a pandas pivot table by grouping the trips by hour and *dayName* columns and getting a mean of the tripDuration values. This pivot table is used to create a heatmap using seaborn as shown in the following example.

Python

```

pv_df = sampled_df[sampled_df["tripDuration"]<180]\n    .groupby(["hour", "dayname"]).mean("tripDuration")\n    .reset_index().pivot("hour", "dayname", "tripDuration")\nsns.heatmap(pv_df, annot=True, fmt=' .2f', cmap="Blues").set(xlabel=None)

```



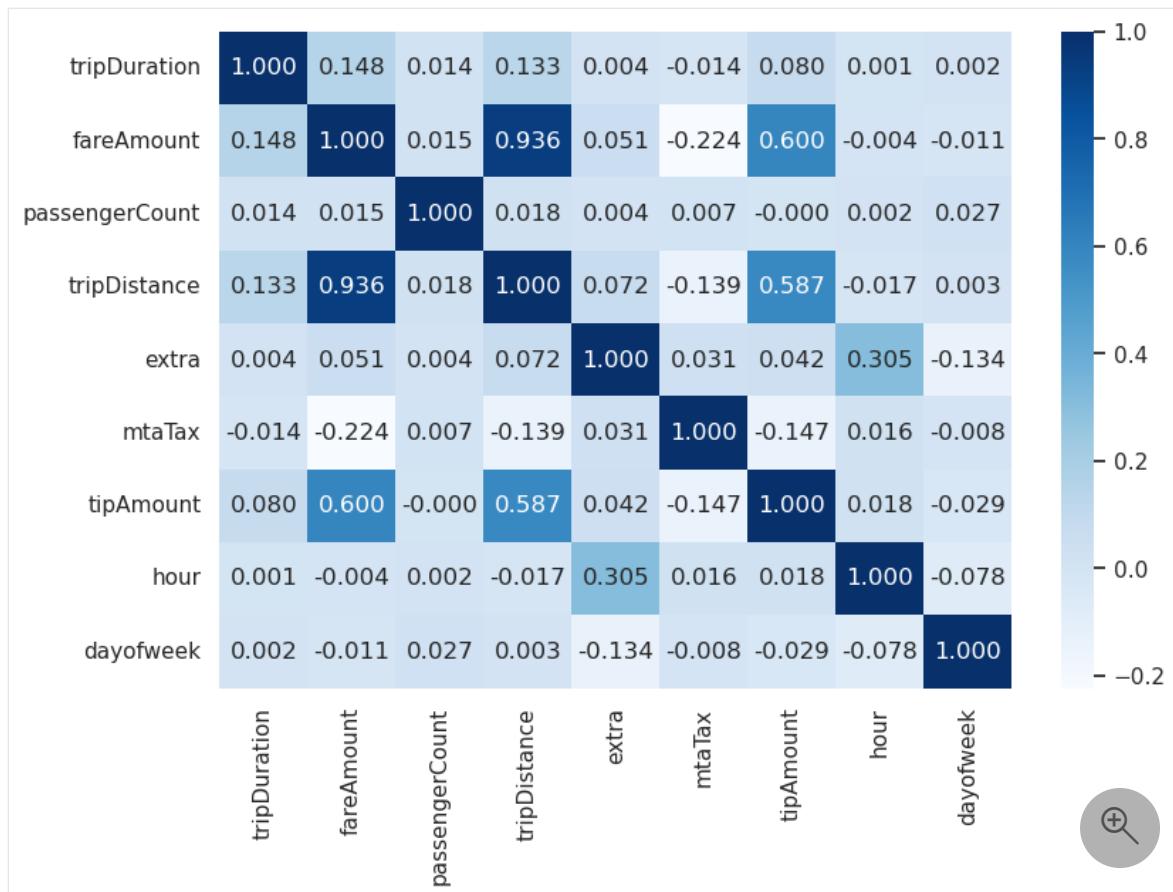
Friday   Monday   Saturday   Sunday   Thursday   Tuesday   Wednesday



11. Finally let's create a correlation plot, which is a useful tool for exploring the relationships among numerical variables in a dataset. It displays the data points for each pair of variables as a scatterplot and calculates the correlation coefficient for each pair. The correlation coefficient indicates how strongly and in what direction the variables are related. A positive correlation means that the variables tend to increase or decrease together, while a negative correlation means that they tend to move in opposite directions. In this example, we're generating correlation plot for a subset of columns in the dataframe for analysis.

Python

```
cols_to_corr = ['tripDuration', 'fareAmount', 'passengerCount',
                 'tripDistance', 'extra', 'mtaTax',
                 'improvementSurcharge', 'tipAmount', 'hour', "dayofweek"]
sns.heatmap(data =
sampled_df[cols_to_corr].corr(), annot=True, fmt='.3f', cmap="Blues")
```



## Observations from exploration data analysis

- Some trips in the sample data have a passenger count of 0 but most trips have a passenger count between 1-6.
- tripDuration column has outliers with a comparatively small number of trips having ***tripDuration*** of greater than 3 hours.
- The outliers for TripDuration are specifically present for vendorID 2.
- Some trips have zero trip distance and hence they can be canceled and filtered out from any modeling.
- A small number of trips have no passengers(0) and hence can be filtered out.
- fareAmount column contains negative outliers, which can be removed from model training.
- The number of trips starts rising around 16:00 hours and peaks between 18:00 - 19:00 hours.

## Next steps

- Module 3: Perform data cleansing and preparation using Apache Spark

# Module 3: Perform data cleansing and preparation using Apache Spark

Article • 05/23/2023

The [NYC Yellow Taxi dataset](#) contains over 1.5 Billion trip records with each month of trip data running into millions of records, which makes processing these records computationally expensive and often not feasible with nondistributed processing engines.

## Important

Microsoft Fabric is currently in PREVIEW. This information relates to a prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, expressed or implied, with respect to the information provided here.

In this tutorial, we demonstrate how to use Apache Spark notebooks to clean and prepare the taxi trips dataset. Spark's optimized distribution engine makes it ideal for processing large volumes of data.

## Tip

For datasets of relatively small size, use the Data Wrangler UI, which is a notebook-based graphical user interface tool that provides interactive exploration and a data cleansing experience for users working with pandas dataframes on Microsoft Fabric notebooks.

In the following steps, you read the raw NYC Taxi data from a lakehouse delta lake table (saved in module 1), and perform various operations to clean and transform that data to prepare it for training machine learning models.

## Follow along in notebook

The python commands/script used in each step of this tutorial can be found in the accompanying notebook: [03-perform-data-cleansing-and-preparation-using-apache-spark.ipynb](#). Be sure to [attach a lakehouse to the notebook](#) before executing it.

## Cleanse and prepare

1. Load NYC yellow taxi Data from lakehouse delta table *nytaxi\_raw* using the `spark.read` command.

```
Python
```

```
nytaxi_df = spark.read.format("delta").load("Tables/nytaxi_raw")
```

2. To aid the data cleansing process, next we use Apache Spark's built-in summary feature that generates summary statistics, which are numerical measures that describe aspects of a column in the dataframe. These measures include count, mean, standard deviation, min, and max. Use the following command to view the summary statistics of all columns in the *taxis* dataset.

```
Python
```

```
display(nytaxi_df.summary())
```

### ⚠ Note

Generating summary statistics is a computationally expensive process and can take considerable amount of execution time based on the size of the dataframe. In this tutorial, the step takes between two and three minutes.

Table	Chart	Export results			
Index	summary	vendorID	passengerCount	tripDistance	puLocationId
1	count	46429618	46429618	46429618	1942
2	mean	1.5317228326108563	1.6610720553419156	4.962012819920075	149.70957775489185
3	stddev	0.49899265250498853	1.3139169916343387	4204.500118805805	67.92532418822987
4	min	1	0	-3390583.8	10
5	25%	1.0	1	1.0	100.0
6	50%	2.0	1	1.7	142.0
7	75%	2.0	2	3.1	229.0
8	max	2	9	1.90726288E7	97

3. In this step, we clean the *nytaxi\_df* dataframe and add more columns derived from the values of existing columns.

The following is the set of operations performed in this step:

- a. Add derived Columns

- pickupDate - convert datetime to date for visualizations and reporting
- weekDay - day number of the week

- weekDayName - day names abbreviated
- dayofMonth - day number of month
- pickupHour - hour of pickup time
- tripDuration - representing duration in minutes of the trip
- timeBins - Binned time of the day

## b. Filter Conditions

- fareAmount is between and 100.
- tripDistance greater than 0.
- tripDuration is less than 3 hours (180 minutes).
- passengerCount is between 1 and 8.
- startLat, startLon, endLat, endLon aren't NULL.
- Remove outstation trips(outliers) tripDistance>100.

Python

```
from pyspark.sql.functions import col,when, dayofweek, date_format,
hour,unix_timestamp, round, dayofmonth, lit
nytaxidf_prep = nytaxi_df.withColumn('pickupDate',
col('tpepPickupDateTime').cast('date'))\
    .withColumn("weekDay",
dayofweek(col("tpepPickupDateTime")))\n        .withColumn("weekDayName",
date_format(col("tpepPickupDateTime"), "EEEE"))\
            .withColumn("dayofMonth",
dayofweek(col("tpepPickupDateTime")))\n                .withColumn("pickupHour",
hour(col("tpepPickupDateTime")))\n                    .withColumn("tripDuration",
(unix_timestamp(col("tpepDropoffDateTime")) -
unix_timestamp(col("tpepPickupDateTime")))/60)\n                        .withColumn("timeBins",
when((col("pickupHour") >=7) & (col("pickupHour")<=10) , "MorningRush")\
                            .when((col("pickupHour") >=11) &
(col("pickupHour")<=15) , "Afternoon")\
                                .when((col("pickupHour") >=16) &
(col("pickupHour")<=19) , "EveningRush")\
                                    .when((col("pickupHour") <=6) | \
(col("pickupHour")>=20) , "Night"))\
                                        .filter("""fareAmount > 0 AND fareAmount <
100 AND tripDistance > 0 AND tripDistance < 100
AND tripDuration > 0 AND
tripDuration <= 189
AND passengerCount > 0 AND
passengerCount <= 8
AND startLat IS NOT NULL AND
startLon IS NOT NULL AND endLat IS NOT NULL AND endLon IS NOT NULL""")
```

### (!) Note

Apache Spark uses Lazy evaluation paradigm which delays the execution of transformations until an action is triggered. This allows Spark to optimize the execution plan and avoid unnecessary computations. In this step, the definitions of the transformations and filters are created. The actual cleansing and transformation will be triggered once data is written (an action) in the next step.

- Once we've defined the cleaning steps and assigned them to a dataframe named *nytaxidf\_prep*, we write the cleansed and prepared data to a new delta table (*nytaxi\_prep*) in the attached lakehouse, using the following set of commands.

Python

```
table_name = "nytaxi_prep"
nytaxidf_prep.write.mode("overwrite").format("delta").save(f"Tables/{table_name}")
print(f"Spark dataframe saved to delta table: {table_name}")
```

The cleansed and prepared data produced in this module is now available in the lakehouse as a delta table and can be used for further processing and generating insights.

## Next steps

- [Module 4: Train and register machine learning models in Microsoft Fabric](#)

# Module 4: Train and register machine learning models in Microsoft Fabric

Article • 05/23/2023

In this module, you learn to train machine learning models to predict the total ride duration (`tripDuration`) of yellow taxi trips in New York City based on various factors, such as pickup and drop-off locations, distance, date, time, number of passengers, and rate code. Once a model is trained, you register the trained models, and log hyperparameters used and evaluation metrics using Fabric's native integration with the MLflow framework.

## Important

Microsoft Fabric is currently in PREVIEW. This information relates to a prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, expressed or implied, with respect to the information provided here.

## Tip

MLflow is an open-source platform for managing the end-to-end machine learning lifecycle with features for tracking experiments, packaging ML models and items, and model registry. For more information, see [MLflow](#).

## Follow along in notebook

The python commands/script used in each step of this tutorial can be found in the accompanying notebook: [04-train-and-track-machine-learning-models.ipynb](#). Be sure to [attach a lakehouse to the notebook](#) before executing it.

In the following steps, you load cleansed and prepared data from lakehouse delta table and use it to train a regression model to predict `tripDuration` variable. You also use the Fabric MLflow integration to create and track experiments and register the trained model, model hyperparameters and metrics.

## Train and register models

1. In the first step, we import the MLflow library and create an experiment named `nyctaxi_tripduration` to log the runs and models produced as part of the training

process.

Python

```
# Create Experiment to Track and register model with mlflow
import mlflow
print(f"mlflow library version: {mlflow.__version__}")
EXPERIMENT_NAME = "nyctaxi_tripduration"
mlflow.set_experiment(EXPERIMENT_NAME)
```

2. Read cleansed and prepared data from the lakehouse delta table *nyctaxi\_prep*, and create a fractional random sample from the data (to reduce computation time in this tutorial).

Python

```
SEED = 1234
# note: From the perspective of the tutorial, we are sampling training
# data to speed up the execution.
training_df =
spark.read.format("delta").load("Tables/nyctaxi_prep").sample(fraction
= 0.5, seed = SEED)
```

### 💡 Tip

A seed in machine learning is a value that determines the initial state of a pseudo-random number generator. A seed is used to ensure that the results of machine learning experiments are reproducible. By using the same seed, you can get the same sequence of numbers and thus the same outcomes for data splitting, model training, and other tasks that involve randomness.

3. Perform a random split to get train and test datasets and define categorical and numeric features by executing the following set of commands. We also cache the train and test dataframes to improve the speed of downstream processes.

Python

```
TRAIN_TEST_SPLIT = [0.75, 0.25]
train_df, test_df = training_df.randomSplit(TRAIN_TEST_SPLIT,
seed=SEED)

# Cache the dataframes to improve the speed of repeatable reads
train_df.cache()
test_df.cache()

print(f"train set count:{train_df.count()}")
```

```
print(f"test set count:{test_df.count()}")  
  
categorical_features =  
["storeAndFwdFlag", "timeBins", "vendorID", "weekDayName", "pickupHour", "rateCodeId", "paymentType"]  
numeric_features = ['passengerCount', "tripDistance"]
```

### 💡 Tip

Apache Spark caching is a feature that allows you to store intermediate data in memory or disk and reuse it for multiple queries or operations. Caching can improve the performance and efficiency of your Spark applications by avoiding reprocessing of data that is frequently accessed. You can use different methods and storage levels to cache your data, depending on your needs and resources. Caching is especially useful for iterative algorithms or interactive analysis that require repeated access to the same data.

4. In this step, we define the steps to perform more feature engineering and train the model using [Spark ML](#) pipelines and Microsoft [SynapseML](#) library. The algorithm we use for this tutorial, LightGBM, is a fast, distributed, high performance gradient-boosting framework based on decision-tree algorithms. It's an open-source project developed by Microsoft and supports regression, classification, and many other machine learning scenarios. Its main advantages are faster training speed, lower memory usage, better accuracy, and support for distributed learning.

#### Python

```
from pyspark.ml.feature import OneHotEncoder, VectorAssembler,  
StringIndexer  
from pyspark.ml import Pipeline  
from synapse.ml.core.platform import *  
from synapse.ml.lightgbm import LightGBMRegressor  
  
# Define a pipeline steps for training a LightGBMRegressor regressor  
model  
def lgbm_pipeline(categorical_features, numeric_features,  
hyperparameters):  
    # String indexer  
    stri = StringIndexer(inputCols=categorical_features,  
                         outputCols=[f"{feat}Idx" for feat in  
categorical_features]).setHandleInvalid("keep")  
    # encode categorical/indexed columns  
    ohe = OneHotEncoder(inputCols=stri.getOutputCols(),  
                         outputCols=[f"{feat}Enc" for feat in  
categorical_features])  
  
    # convert all feature columns into a vector
```

```

        featurizer = VectorAssembler(inputCols=ohe.getOutputCols() +
numerics_features, outputCol="features")

        # Define the LightGBM regressor
        lgr = LightGBMRegressor(
            objective = hyperparameters["objective"],
            alpha = hyperparameters["alpha"],
            learningRate = hyperparameters["learning_rate"],
            numLeaves = hyperparameters["num_leaves"],
            labelCol="tripDuration",
            numIterations = hyperparameters["iterations"],
        )
        # Define the steps and sequence of the SPark ML pipeline
        ml_pipeline = Pipeline(stages=[stri, ohe, featurizer, lgr])
        return ml_pipeline

```

5. Define training Hyperparameters as a python dictionary for the initial run of the lightgbm model by executing the below cell.

Python

```

# Default hyperparameters for LightGBM Model
LGBM_PARAMS = {"objective":"regression",
               "alpha":0.9,
               "learning_rate":0.1,
               "num_leaves":31,
               "iterations":100}

```

### Tip

Hyperparameters are the parameters that you can change to control how a machine learning model is trained. Hyperparameters can affect the speed, quality and accuracy of the model. Some common methods to find the best hyperparameters are by testing different values, using a grid or random search, or using a more advanced optimization technique. The hyperparameters for the LightGBM model in this tutorial have been pre-tuned using a distributed grid search (not covered as part of this tutorial) run using the [hyperopt](#) library.

6. Next, we create a new run in the defined experiment using MLflow and fit the defined pipeline on the training dataframe. We then generate predictions on the test dataset, using the following set of commands.

Python

```

if mlflow.active_run() is None:
    mlflow.start_run()
run = mlflow.active_run()
print(f"Active experiment run_id: {run.info.run_id}")
lg_pipeline =
lgbm_pipeline(categorical_features, numeric_features, LGBM_PARAMS)
lg_model = lg_pipeline.fit(train_df)

# Get Predictions
lg_predictions = lg_model.transform(test_df)
## Caching predictions to run model evaluation faster
lg_predictions.cache()
print(f"Prediction run for {lg_predictions.count()} samples")

```

7. Once a model is trained and predictions generated on the test set, we can compute model statistics for evaluating performance of the trained LightGBMRegressor model by using SynapseML library utility ***ComputeModelStatistics***, which helps evaluate various types of models based on the algorithm. Once the metrics are generated, we also convert them into a python dictionary object for logging purposes. The metrics on which a regression model is evaluated are MSE(Mean Square Error), RMSE(Root Mean Square Error), R^2 and MAE(Mean Absolute Error).

Python

```

from synapse.ml.train import ComputeModelStatistics
lg_metrics = ComputeModelStatistics(
    evaluationMetric="regression", labelCol="tripDuration",
    scoresCol="prediction"
).transform(lg_predictions)
display(lg_metrics)

```

The output values are similar to the following table:

mean_squared_error	root_mean_squared_error	r2	mean_absolute_error
25.721591239516997	5.071645811718026	0.7613537489434428	3.25946u0228763087

8. Next, we define a general function to register the trained LightGBMRegressor model with default hyperparameters under the created experiment using MLflow. We also log associated hyperparameters used and metrics for model evaluation in the experiment run and terminate the run in the MLflow experiment in the end.

Python

```

from mlflow.models.signature import ModelSignature
from mlflow.types.utils import _infer_schema

```

```

# Define a function to register a spark model
def register_spark_model(run, model, model_name, signature, metrics,
hyperparameters):
    # log the model, parameters and metrics
    mlflow.spark.log_model(model, artifact_path = model_name,
signature=signature, registered_model_name = model_name,
dfs_tmpdir="Files/tmp/mlflow")
    mlflow.log_params(hyperparameters)
    mlflow.log_metrics(metrics)
    model_uri = f"runs:{run.info.run_id}/{model_name}"
    print(f"Model saved in run{run.info.run_id}")
    print(f"Model URI: {model_uri}")
    return model_uri

# Define Signature object
sig =
ModelSignature(inputs=_infer_schema(train_df.select(categorical_features + numeric_features)),

outputs=_infer_schema(train_df.select("tripDuration")))

ALGORITHM = "lightgbm"
model_name = f"{EXPERIMENT_NAME}_{ALGORITHM}"

# Call model register function
model_uri = register_spark_model(run = run,
                                    model = lg_model,
                                    model_name = model_name,
                                    signature = sig,
                                    metrics = lg_metrics_dict,
                                    hyperparameters = LGBM_PARAMS)
mlflow.end_run()

```

9. Once the default model is trained and registered, we define tuned hyperparameters (tuning hyperparameters not covered in this tutorial) and remove the *paymentType* categorical feature. Because *paymentType* is usually selected at the end of a trip, we hypothesize that it shouldn't be useful to predict trip duration.

Python

```

# Tuned hyperparameters for LightGBM Model
TUNED_LGBM_PARAMS = {"objective":"regression",
"alpha":0.08373361416254149,
"learning_rate":0.0801709918703746,
"num_leaves":92,
"iterations":200}

# Remove paymentType
categorical_features.remove("paymentType")

```

10. After defining new hyperparameters and updating feature list, we fit the LightGBM pipeline with tuned hyperparameters on the training dataframe and generate predictions on the test dataset.

Python

```
if mlflow.active_run() is None:
    mlflow.start_run()
run = mlflow.active_run()
print(f"Active experiment run_id: {run.info.run_id}")
lg_pipeline_tn =
lgbm_pipeline(categorical_features, numeric_features, TUNED_LGBM_PARAMS)
lg_model_tn = lg_pipeline_tn.fit(train_df)

# Get Predictions
lg_predictions_tn = lg_model_tn.transform(test_df)
## Caching predictions to run model evaluation faster
lg_predictions_tn.cache()
print(f"Prediction run for {lg_predictions_tn.count()} samples")
```

11. Generate model evaluation metrics for the new LightGBM regression model with optimized hyperparameters and updated features.

Python

```
lg_metrics_tn = ComputeModelStatistics(
    evaluationMetric="regression", labelCol="tripDuration",
    scoresCol="prediction"
).transform(lg_predictions_tn)
display(lg_metrics_tn)
```

12. In the final step, we register the second LightGBM regression model, and log the metrics and hyperparameters to the MLflow experiment and end the run.

Python

```
# Define Signature object
sig_tn =
ModelSignature(inputs=_infer_schema(train_df.select(categorical_features + numeric_features)),

outputs=_infer_schema(train_df.select("tripDuration")))
model_uri = register_spark_model(run = run,
                                  model = lg_model_tn,
                                  model_name = model_name,
                                  signature = sig_tn,
                                  metrics = lg_metricstn_dict,
                                  hyperparameters = TUNED_LGBM_PARAMS)

mlflow.end_run()
```

The output values are similar to the following table:

mean_squared_error	root_mean_squared_error	r2	mean_absolute_error
25.444472646953216	5.0442514456511	0.7637293020097541	3.241663446115354

At the end of the module, we have two runs of the lightgbm regression model trained and registered in the MLflow model registry, and the model is also available in the workspace as a Fabric model item.

### (!) Note

If you do not see your model item in the list, refresh your browser.

In order to view the model in the UI:

- Navigate to your currently active Fabric workspace.
- Select the model item named *nyctaxi\_tripduration\_lightgbm* to open the model UI.
- On the model UI, you can view the properties and metrics of a given run, compare performance of various runs, and download various file items associated with the trained model.

The following image shows the layout of the various features within the model UI in a Fabric workspace.

The screenshot displays the Fabric Model UI interface. At the top, there are navigation links: Home, View, Apply Model to Dataset, and Download Model Files. Below these are buttons for Apply this version, Download model version, Delete version, and Sensitivity.

On the left, a sidebar shows 'Registered Model Versions' with entries for 'nyctaxi\_tripduration\_lig...' (Version 2, 156:46 AM) and 'Version 1' (149:08 AM).

The main content area is divided into several sections:

- Properties:** Shows Version name (Version 2), Experiment name (nyctaxi\_tripduration), Run name (funny\_papaya\_zwsq8pp7), Created time (4/18/2023 1:56 AM), Last modified, and Created by.
- Latest Model Run:** A button labeled 'Apply this version' with a description: 'Apply this model version to generate predictions from your data. Learn more'.
- Experiment:** A table showing Model version metrics (4) and Model Parameters (5).
  - Model Metrics:** mean\_absolute\_error: 3.2577370362776645, mean\_squared\_error: 25.63647085036277, R^2: 0.7619464570609955, root\_mean\_squared\_error: 5.063247065901759.
  - Model Hyperparameters:** objective: regression, alpha: 0.08373361416254149, learning\_rate: 0.0801709918703746, num\_leaves: 92, iterations: 200.
- Input schema (8):** A section showing the input schema for the model.

## Next steps

- Module 5: Perform batch scoring and save predictions to a lakehouse

# Module 5: Perform batch scoring and save predictions to a lakehouse

Article • 05/23/2023

In this module, you learn to import a trained and registered LightGBMRegressor model from the Microsoft Fabric MLflow model registry, and perform batch predictions on a test dataset loaded from a lakehouse.

## ⓘ Important

Microsoft Fabric is currently in PREVIEW. This information relates to a prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, expressed or implied, with respect to the information provided here.

## Follow along in notebook

The python commands/script used in each step of this tutorial can be found in the accompanying notebook: [05-perform-batch-scoring-and-save-predictions-to-lakehouse.ipynb](#). Be sure to [attach a lakehouse to the notebook](#) before executing it.

## Perform batch scoring and save predictions

1. Read a random sample of cleansed data from lakehouse table *nyctaxi\_prep* filtered for puYear=2016 and puMonth=3.

Python

```
SEED = 1234 # Random seed
input_df = spark.read.format("delta").load("Tables/nyctaxi_prep")\
    .filter("puYear = 2016 AND puMonth = 3")\
    .sample(True, 0.01, seed=SEED) ## Sampling data to reduce
execution time for this tutorial
```

2. Import the required pyspark.ml and synapse.ml libraries and load the trained and registered LightGBMRegressor model using the *run\_uri* copied from the final step of [Module 4: Train and register machine learning models](#).

Python

```

import mlflow
from pyspark.ml.feature import OneHotEncoder, VectorAssembler,
StringIndexer
from pyspark.ml import Pipeline
from synapse.ml.core.platform import *
from synapse.ml.lightgbm import LightGBMRegressor

## Define run_uri to fetch the model
run_uri = "<enter the run_uri from module 04 here>"
loaded_model = mlflow.spark.load_model(run_uri,
dfs_tmpdir="Files/tmp/mlflow")

```

- Run model transform on the input dataframe to generate predictions and remove unnecessary vector features created for model training using the following commands.

Python

```

# Generate predictions by applying model transform on the input
dataframe
predictions = loaded_model.transform(input_df)
cols_toremove = ['storeAndFwdFlagIdx', 'timeBinsIdx', 'vendorIDIdx',
'paymentTypeIdx', 'vendorIDEnc',
'rateCodeIdEnc', 'paymentTypeEnc', 'weekDayEnc', 'pickupHourEnc',
'storeAndFwdFlagEnc', 'timeBinsEnc', 'features','weekDayNameIdx',
'pickupHourIdx', 'rateCodeIdIdx', 'weekDayNameEnc']
output_df = predictions.withColumnRenamed("prediction",
"predictedtripDuration").drop(*cols_toremove)

```

- Save predictions to lakehouse delta table **nyctaxi\_pred** for downstream consumption and analysis.

Python

```

table_name = "nyctaxi_pred"
output_df.write.mode("overwrite").format("delta").save(f"Tables/{table_
name}")
print(f"Output Predictions saved to delta table: {table_name}")

```

- Preview the final predicted data by various methods including SparkSQL queries that can be executed using the %%sql magics command, which tells the notebook engine that the cell is a SparkSQL script.

Python

```

%%sql
SELECT * FROM nyctaxi_pred LIMIT 20

```

Table Chart ↗ Export results ▾

weekDayName	dayofMonth	pickupHour	tripDuration	timeBins	predictedtripDuration
Friday	6	16	59.78333333333333	EveningRush	61.58005626051534
Monday	2	11	35.81666666666667	Afternoon	42.417611111723566
Monday	2	10	0.2166666666666667	MorningRush	33.3357211596686
Wednesday	4	13	51.45	Afternoon	43.7793609451249
Sunday	1	20	54.21666666666667	Night	46.82853219745318
Friday	6	7	62.15	MorningRush	61.88564862320572
Friday	6	13	45.5	Afternoon	50.32821615322531
Monday	2	13	48.7	Afternoon	48.23349522219192
Saturday	7	18	57.4	EveningRush	47.96713768906313
Monday	2	22	36.36666666666667	Night	34.03186812788601
Wednesday	4	12	69.6	Afternoon	51.387055239584086
Saturday	7	15	39.13333333333333	Afternoon	43.54207101473043
Friday	6	13	25.1	Afternoon	35.83451867516566
Friday	6	15	39.35	Afternoon	59.92015611554354

6. The **nyctaxi\_pred** delta table containing predictions can also be viewed from the lakehouse UI by navigating to the lakehouse item in the active Fabric workspace.

Home

Lakehouse

Get data New Power BI dataset Open notebook

A SQL endpoint for SQL querying and a default dataset for reporting were created and will be updated with any tables added to the lakehouse. You can access the SQL endpoint using the dropdown.

Lakehouse explorer

nyctaxi\_pred

Showing <1000> rows

vendorID	tpepPickup...	tpepDro... ...	passenger... ...	tripDistance	startLon	startLat	endLon	endLat	rateCodeId	storeAndF...	paymentTy...	
1	1	3/18/2016 ...	3/18/2016 ...	1	16.1	-73.971786...	40.7513732...	-73.99054...	40.6468086...	2	N	1
2	1	3/21/2016 ...	3/21/2016 ...	1	17.4	-73.983001...	40.7618598...	-73.781791...	40.6444511...	2	N	1
3	1	3/7/2016 1...	3/7/2016 1...	1	11.5	-74.005256...	40.7237205...	-74.005249...	40.7237167...	2	N	1
4	2	3/16/2016 ...	3/16/2016 ...	1	16.08	0	0	0	0	2	N	1
5	1	3/20/2016 ...	3/20/2016 ...	1	28.2	-73.951118...	40.7744331...	-73.790390...	40.6467666...	2	Y	1
6	1	3/11/2016 ...	3/11/2016 ...	1	21.1	-73.980346...	40.7739067...	-73.785751...	40.6433143...	2	N	1
7	1	3/25/2016 ...	3/25/2016 ...	1	18	-73.982315...	40.7569236...	-73.776206...	40.6454315...	2	N	1
8	1	3/28/2016 ...	3/28/2016 ...	1	20.7	-73.790802...	40.6466712...	-73.974700...	40.7567443...	2	N	1
9	2	3/26/2016 ...	3/26/2016 ...	1	19.3	-73.780288...	40.6454315...	-73.979141...	40.7640800...	2	N	1
10	2	3/28/2016 ...	3/28/2016 ...	1	18.46	-73.788063...	40.6415863...	-73.988159...	40.7336654...	2	N	1
11	2	3/9/2016 1...	3/9/2016 1...	1	19.31	-73.776687...	40.6453170...	-73.985176...	40.7583732...	2	N	1
12	2	3/19/2016 ...	3/19/2016 ...	1	16.63	-73.785949...	40.6517143...	-73.981430...	40.7560806...	2	N	1
13	1	3/25/2016 ...	3/25/2016 ...	1	16.3	-73.992431...	40.7600288...	-74.182769...	40.6878433...	3	N	1
14	2	3/25/2016 ...	3/25/2016 ...	1	21.56	-73.979431...	40.7658500...	-74.177230...	40.6951293...	3	N	1
15	2	3/3/2016 2...	3/3/2016 3...	1	19.83	-73.980583...	40.7630462...	-74.177452...	40.6906814...	3	N	1

## Next steps

- Module 6: Create a Power BI report to visualize predictions

# Module 6: Create a Power BI report to visualize predictions

Article • 05/23/2023

In this module, we use the Microsoft Fabric DirectLake feature, which enables direct connectivity from Power BI datasets to lakehouse tables in direct query mode with automatic data refresh. For the following steps, use the prediction data produced in [Module 5: Perform batch scoring and save predictions to a lakehouse](#).

## ⓘ Important

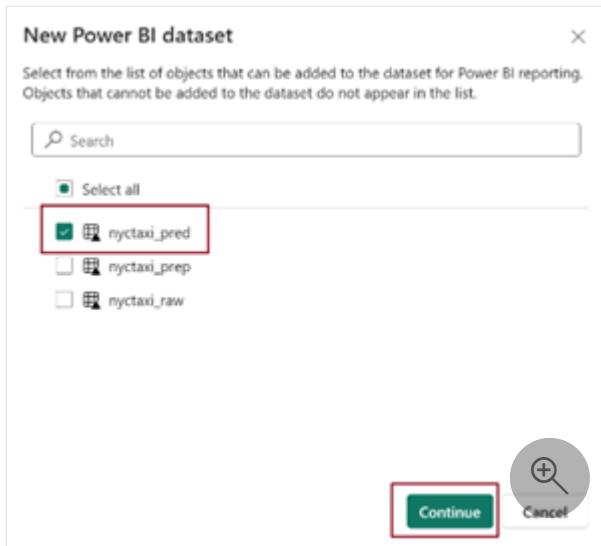
Microsoft Fabric is currently in PREVIEW. This information relates to a prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, expressed or implied, with respect to the information provided here.

## Prepare for creating reports

1. Navigate to the default lakehouse item in the workspace that you used as part of the previous modules and open the lakehouse UI.
2. Select **New Power BI dataset** on the top ribbon and select **nyctaxi\_pred**, then select **Continue** to create a new Power BI dataset linked to the predictions data you produced in module 5.

The screenshot shows the Microsoft Fabric Lakehouse explorer interface. The top navigation bar includes 'Home', 'Get data', 'New Power BI dataset' (which is highlighted with a red box), and 'Open notebook'. A status message indicates a SQL endpoint was created. The left sidebar shows a tree view of 'DSLakehouse' with 'Tables' expanded, showing 'nyctaxi\_pred' selected (also highlighted with a red box). The right pane displays the 'nyctaxi\_pred' table with columns: vendorID, tpepPickupLocation, tpepDropoffLocation, and tripDistance. The first four rows of data are visible. A magnifying glass icon with a plus sign is located in the bottom right corner of the table preview area.

vendorID	tpepPickupLocation	tpepDropoffLocation	tripDistance
1	3/18/2016 ...	3/18/2016 ...	1.00
2	3/21/2016 ...	3/21/2016 ...	1.00
3	3/7/2016 1:15:00 AM	3/7/2016 1:15:00 AM	1.00
4	3/16/2016 ...	3/16/2016 ...	1.00



- Once the UI for the new dataset loads, rename the dataset by clicking on the dropdown at top left corner of the dataset page and entering the more user-friendly name **nyctaxi\_predictions**. Click outside the drop down to apply the name change.

DSLakehouse (1) ▾

Name: nyctaxi\_predictions

Location: Review DS Tutorials

Owner: Abid Nazir Guroo

Refreshed: 4/18/23, 12:33 AM

Visualize this data

Create an interactive report, or a table, to discover and share business insights. [Learn more](#)

+ Create from scratch

- On the dataset pane in the section titled **Visualize this data**, select **Create from scratch** and then select **Start from Scratch** to open the Power BI report authoring page.

The screenshot shows the 'Details for nyctaxi\_predictions' page in Azure Data Studio. At the top, there's a 'Location' section with 'Review DS Tutorials' and a refresh icon, and a timestamp '4/18/23, 12:33:52 AM'. Below this is a red-bordered box containing a 'Visualize this data' section with a pie chart icon, a 'Create from scratch' button, and a dropdown menu for 'Auto-create' (with 'Start from scratch' and 'Paginated report' options). To the right is a 'Share this data' section with a user icon and a 'Share dataset' button. A search bar and a 'Filter' dropdown are at the top right. The main table lists two items: 'DSLakehouse' (SQL endpoint, Upstream, Refreshed, Endorsement, Sensitivity) and another 'DSLakehouse' entry (Lakehouse, Upstream, Refreshed, Endorsement, Sensitivity). A magnifying glass icon is at the bottom right of the table.

Name	Type	Relation	Location	Refreshed	Endorsement	Sensitivity
DSLakehouse	SQL endpoint	Upstream	Review DS Tutorials	—	—	Confidential/Micro... ⓘ
DSLakehouse	Lakehouse	Upstream	Review DS Tutorials	—	—	Confidential/Micro... ⓘ

You can now create various visuals to generate insights from the prediction dataset.

## Sample visuals to analyze predictedTripDuration

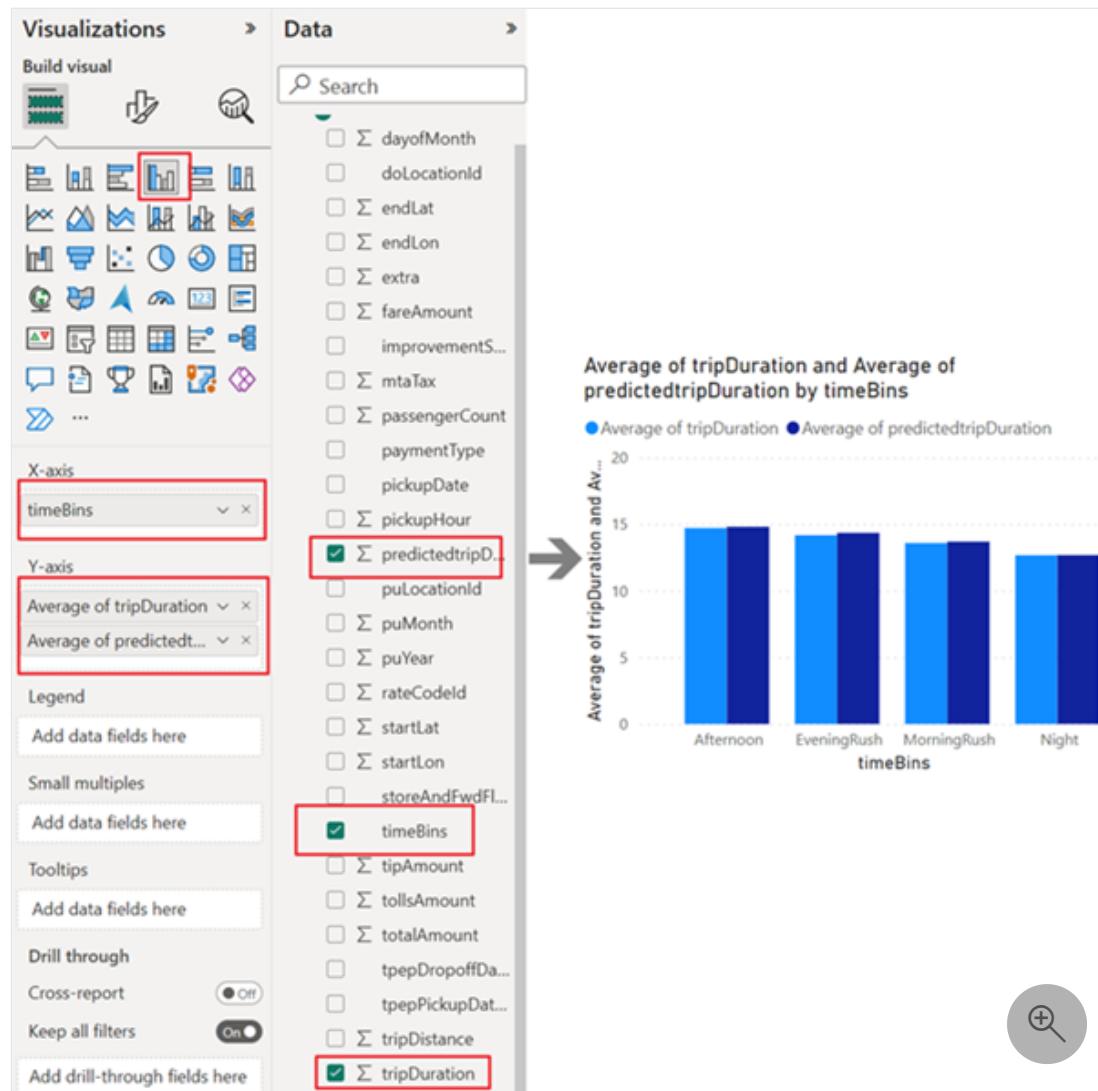
1. Create a Slicer visualization for pickupDate.

- Select the slicer option from the visualizations pane and select *pickupDate* from the data pane and drop it on the created slicer visualization field of the date slider visual.

The screenshot shows the Power BI Data view. On the left, there's a visualizations pane with various chart icons, and a 'Build visual' section. Below that is a 'Field' section where 'pickupDate' is selected. A red box highlights both the 'Field' dropdown and the 'pickupDate' entry. To the right, under 'nyctaxi\_pred', several fields are listed with checkboxes next to them. One checkbox for 'pickupDate' is checked and highlighted with a red box. Below this, there's a date range filter for 'pickupDate' set from '2/29/2016' to '3/31/2016'. A large grey arrow points from the Data view towards the bottom of the page.

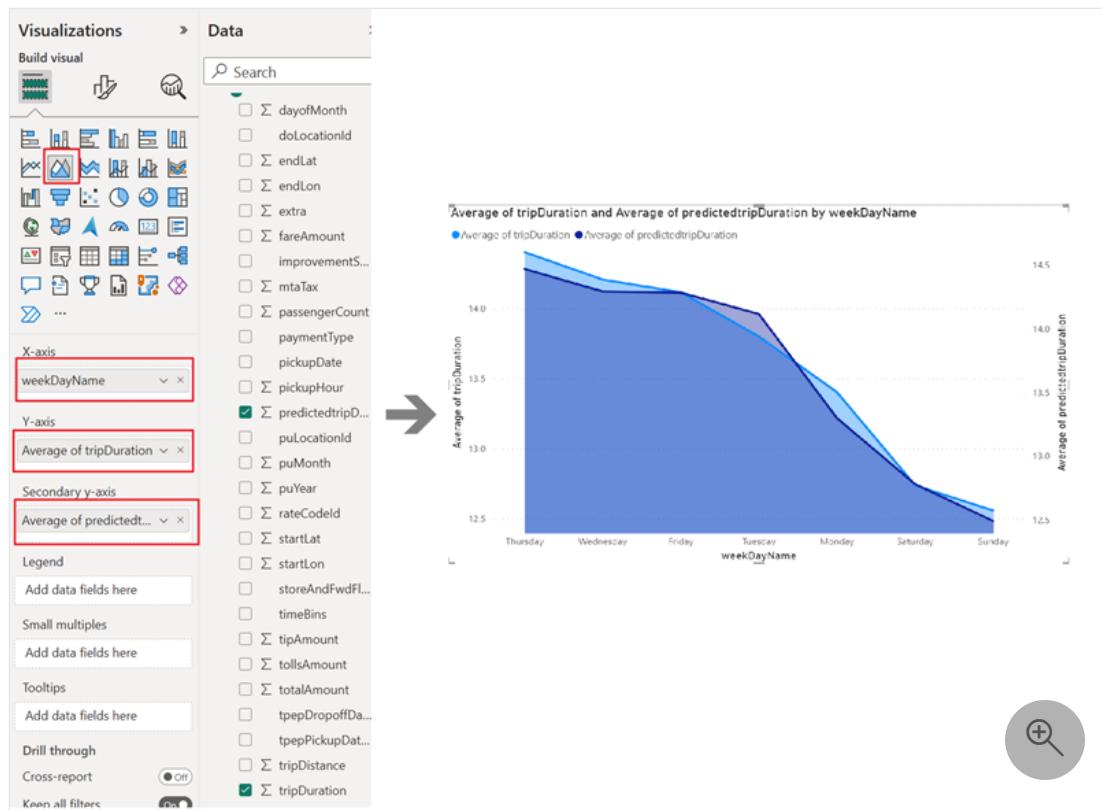
2. Visualize Average tripDuration and predictedTripDuration by timeBins using a clustered column chart.

- Add a clustered column chart, add *timeBins* to the X-axis, *tripDuration* and *predictedTripDuration* to the Y-axis and change the aggregation method to Average.



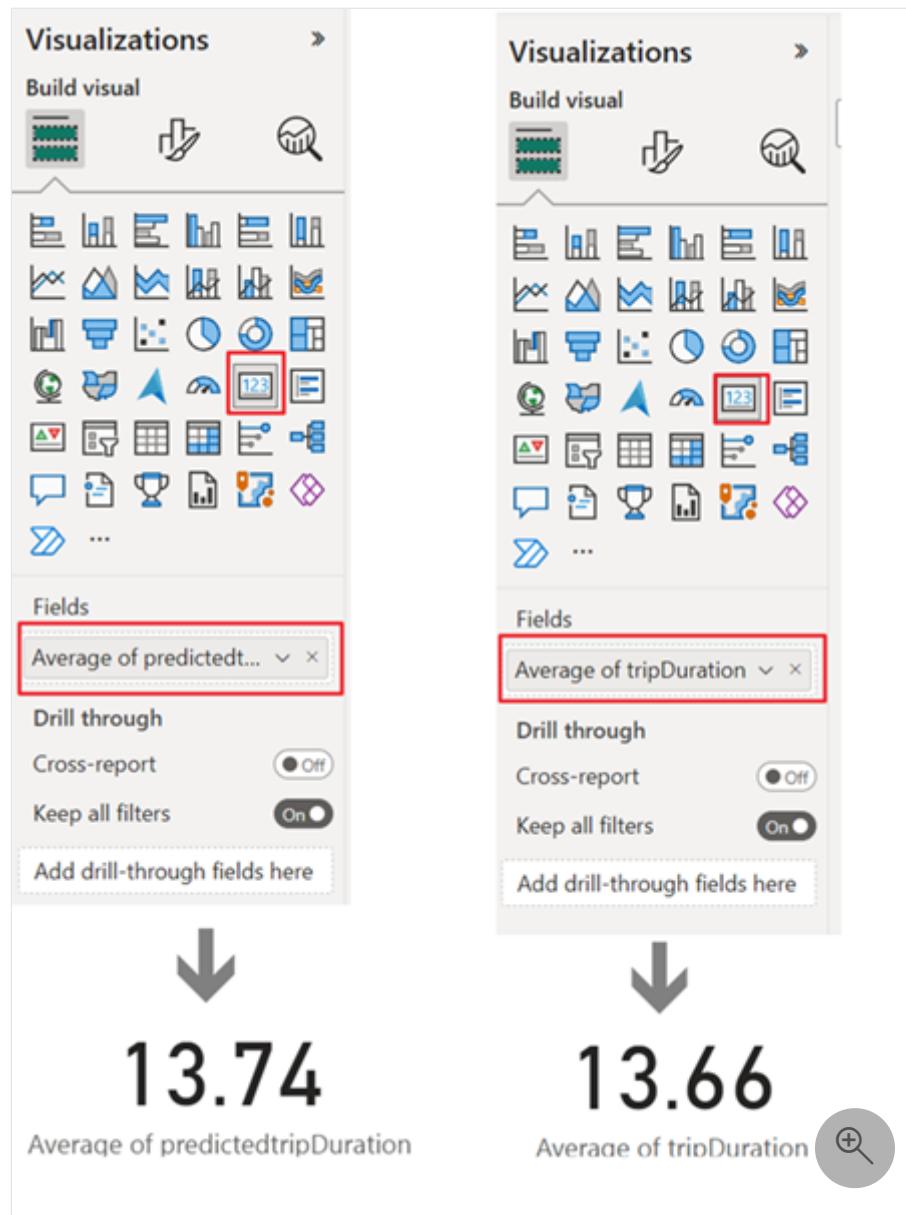
### 3. Visualize Average tripDuration and predictedTripDuration by weekDayName.

- Add an area chart visual and add *weekDayName* onto X-axis, *tripDuration* to Y-axis and *predictedTripDuration* to secondary Y-axis. Switch aggregation method to Average for both Y-axes.



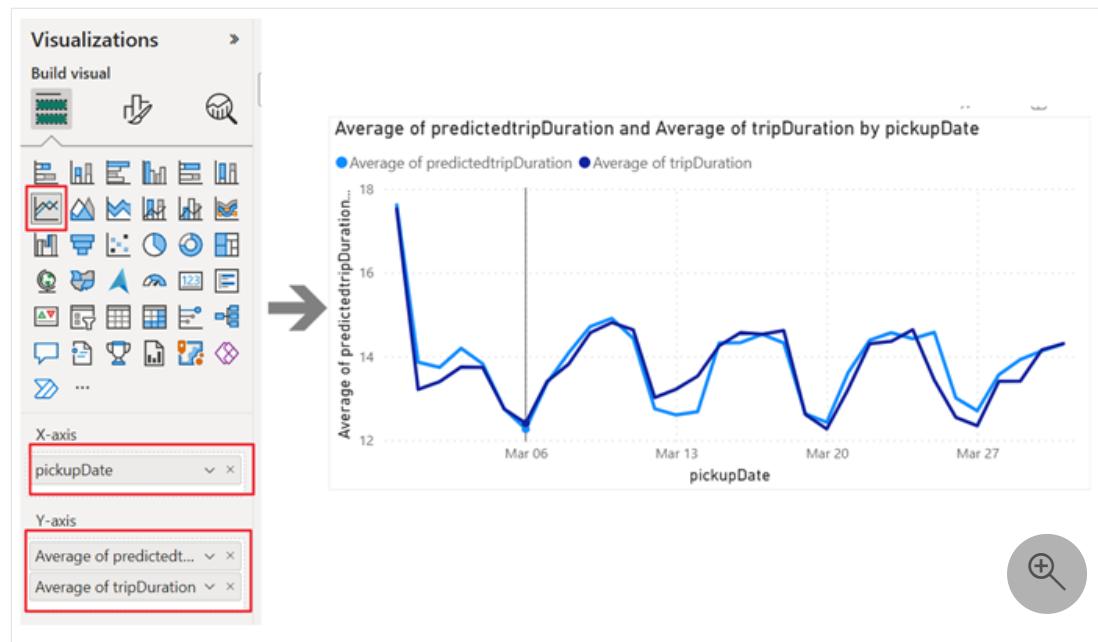
#### 4. Add Card visuals for overall predictedTripDuration and tripDuration.

- Add a Card Visual and add predictedTripDuration to the fields and switch aggregation method to Average.
- Add a Card Visual and add TripDuration to the fields and switch aggregation method to Average.



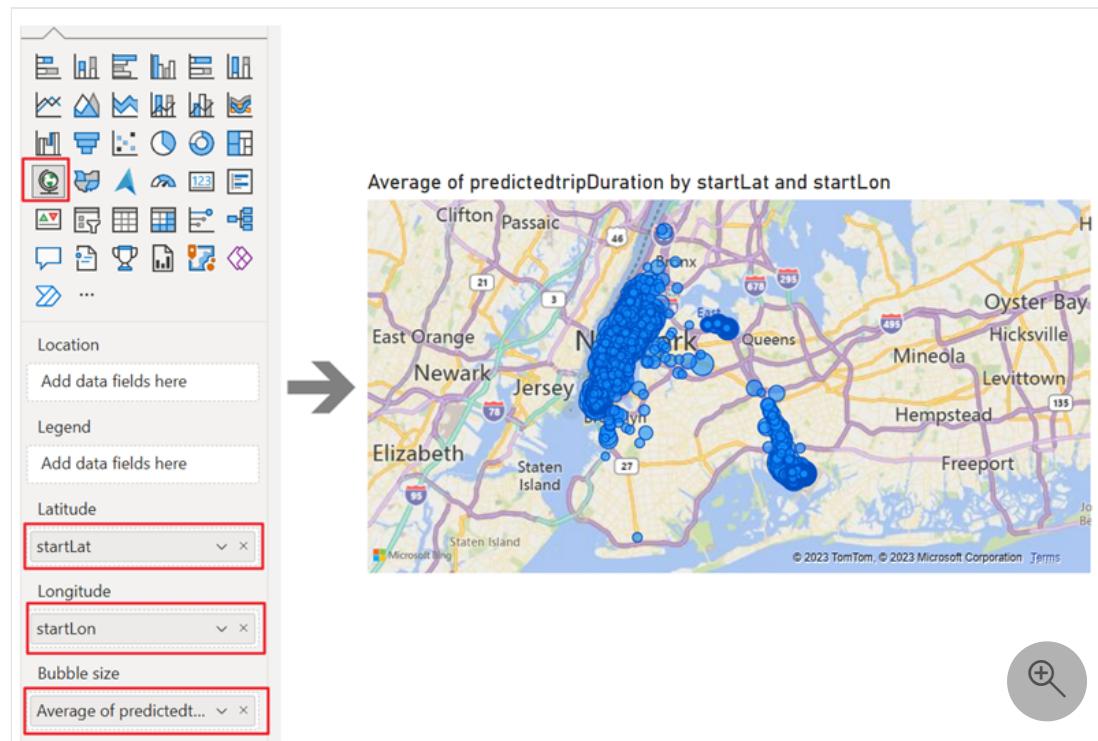
5. Visualize Average tripDuration and predictedTripDuration by pickupDate using line chart.

- Add a line chart visual and add *pickupDate* onto X-axis, *tripDuration* and *predictedTripDuration* to Y-axis and switch aggregation method to Average for both fields.

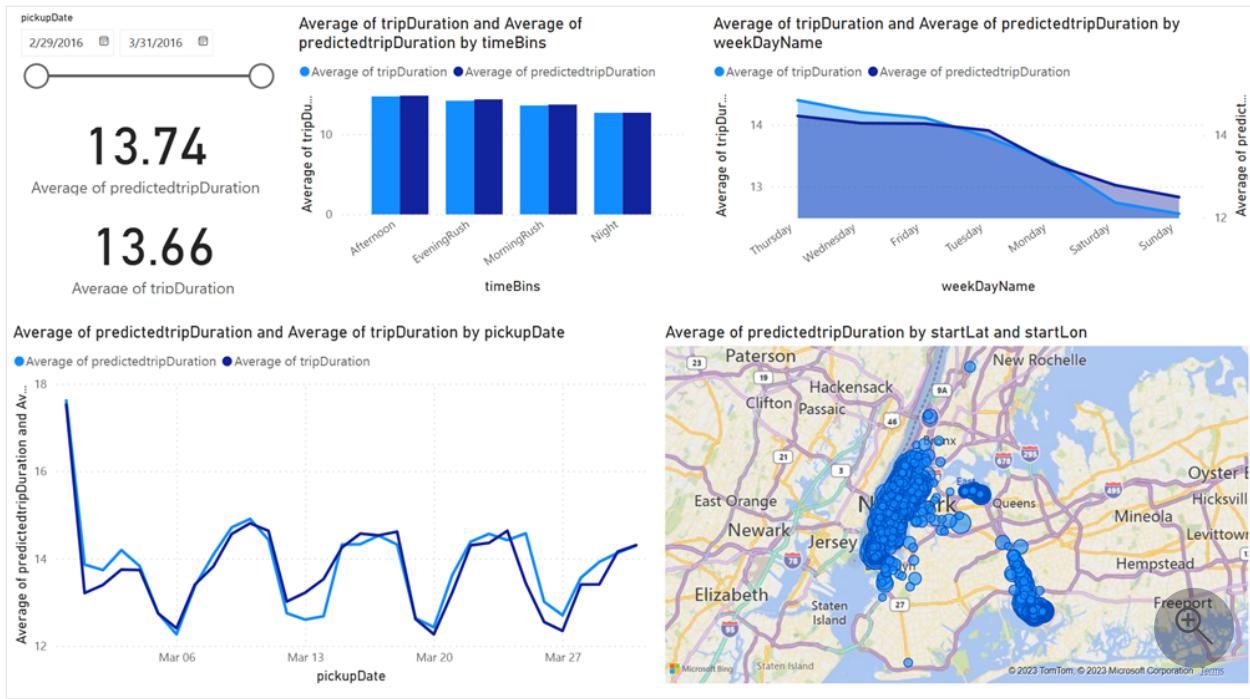


## 6. Visualize Average predictedTripDuration using a map visual.

- Add a map chart visual, and add *startLat* to the Latitude field and *startLon* to the Longitude field.
- Add *predictedTripDuration* to bubble size field and switch the aggregation method of *predictedTripDuration* to Average.



Once all the visuals are added, you can reshape the visuals and realign the layout based on your preferences.



## Next steps

- How to use end-to-end AI samples in Microsoft Fabric

# How to use Microsoft Fabric notebooks

Article • 05/23/2023

Microsoft Fabric notebook is a primary code item for developing Apache Spark jobs and machine learning experiments, it's a web-based interactive surface used by data scientists and data engineers to write code benefiting from rich visualizations and Markdown text. Data engineers write code for data ingestion, data preparation, and data transformation. Data scientists also use notebooks to build machine learning solutions, including creating experiments and models, model tracking, and deployment.

## Important

Microsoft Fabric is currently in PREVIEW. This information relates to a prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, expressed or implied, with respect to the information provided here.

With a Microsoft Fabric notebook, you can:

- Get started with zero setup effort.
- Easily explore and process data with intuitive low-code experience.
- Keep data secure with built-in enterprise security features.
- Analyze data across raw formats (CSV, txt, JSON, etc.), processed file formats (parquet, Delta Lake, etc.), leveraging powerful Spark capabilities.
- Be productive with enhanced authoring capabilities and built-in data visualization.

This article describes how to use notebooks in data science and data engineering experiences.

## Create notebooks

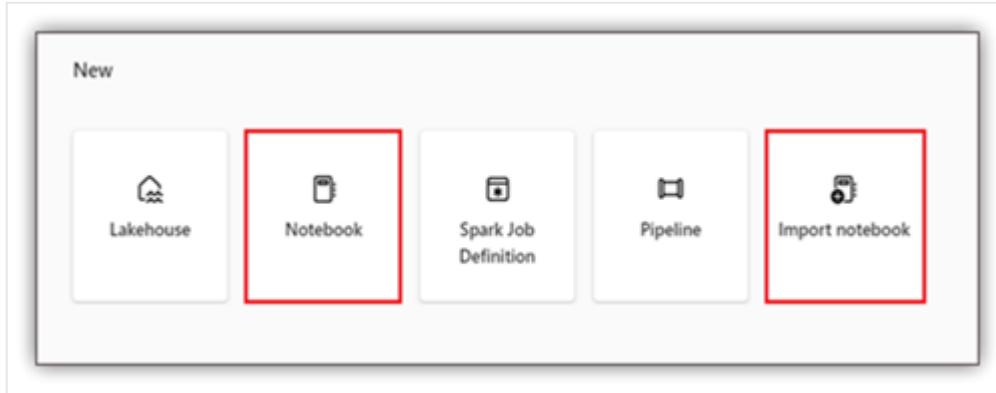
You can either create a new notebook or import an existing notebook.

### Create a new notebook

Similar with other standard Microsoft Fabric item creation, you can easily create a new notebook from the Microsoft Fabric **Data Engineering** homepage, the workspace **New** button, or the **Create Hub**.

### Import existing notebooks

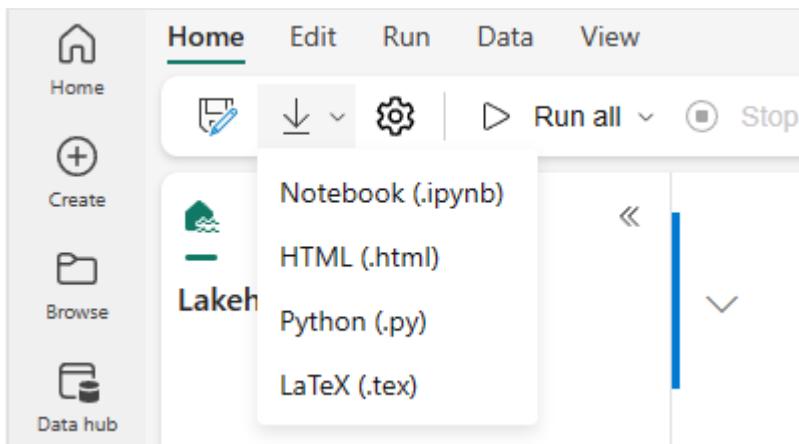
You can import one or more existing notebooks from your local computer to a Microsoft Fabric workspace from the **Data Engineering** or the **Data Science** homepage. Microsoft Fabric notebooks can recognize the standard Jupyter Notebook .ipynb files, and source files like .py, .scala, and .sql, and create new notebook items accordingly.



## Export a notebook

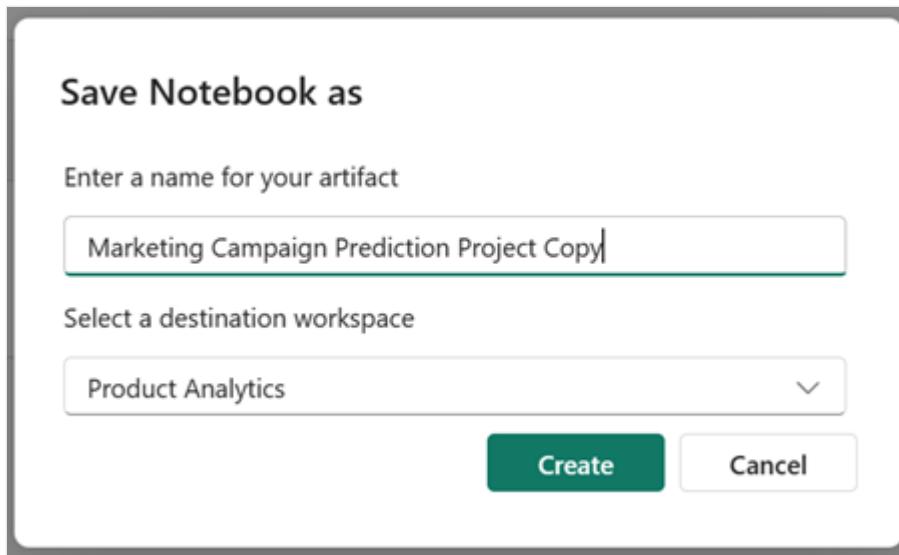
You can Export your notebook to other standard formats. Synapse notebook supports to be exported into:

- Standard Notebook file(.ipynb) that is usually used for Jupyter notebooks.
- HTML file(.html) that can be opened from browser directly.
- Python file(.py).
- Latex file(.tex).

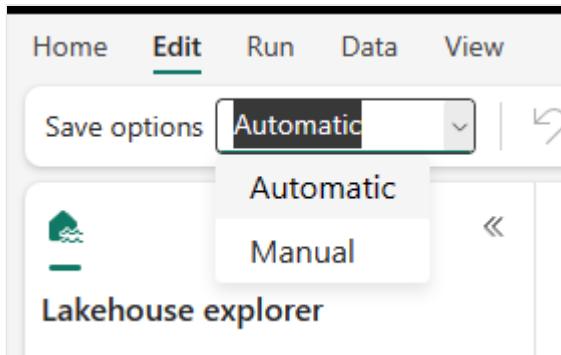


## Save a notebook

In Microsoft Fabric, a notebook will by default save automatically after you open and edit it; you don't need to worry about losing code changes. You can also use **Save a copy** to clone another copy in the current workspace or to another workspace.



If you prefer to save a notebook manually, you can also switch to "Manual save" mode to have a "local branch" of your notebook item, and use **Save** or **CTRL+s** to save your changes.

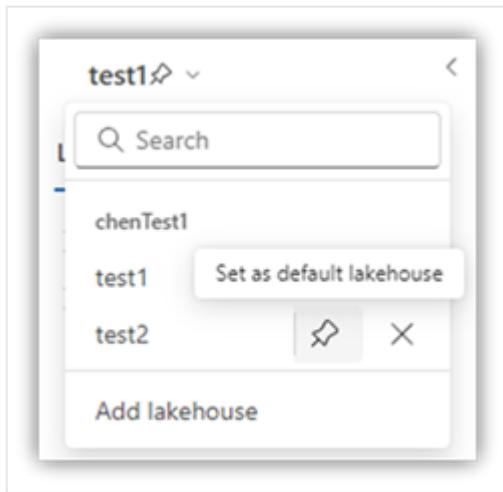


You can also switch to manual save mode from **Edit->Save options->Manual**. To turn on a local branch of your notebook then save it manually by clicking **Save** button or through "Ctrl" + "s" keybinding.

## Connect lakehouses and notebooks

Microsoft Fabric notebook now supports interacting with lakehouses closely; you can easily add a new or existing lakehouse from the lakehouse explorer.

You can navigate to different lakehouses in the lakehouse explorer and set one lakehouse as the default by pinning it. It will then be mounted to the runtime working directory and you can read or write to the default lakehouse using a local path.



### ⓘ Note

You need to restart the session after pinning a new lakehouse or renaming the default lakehouse.

## Add or remove a lakehouse

Selecting the X icon beside a lakehouse name removes it from the notebook tab, but the lakehouse item still exists in the workspace.

Select **Add lakehouse** to add more lakehouses to the notebook, either by adding an existing one or creating a new lakehouse.

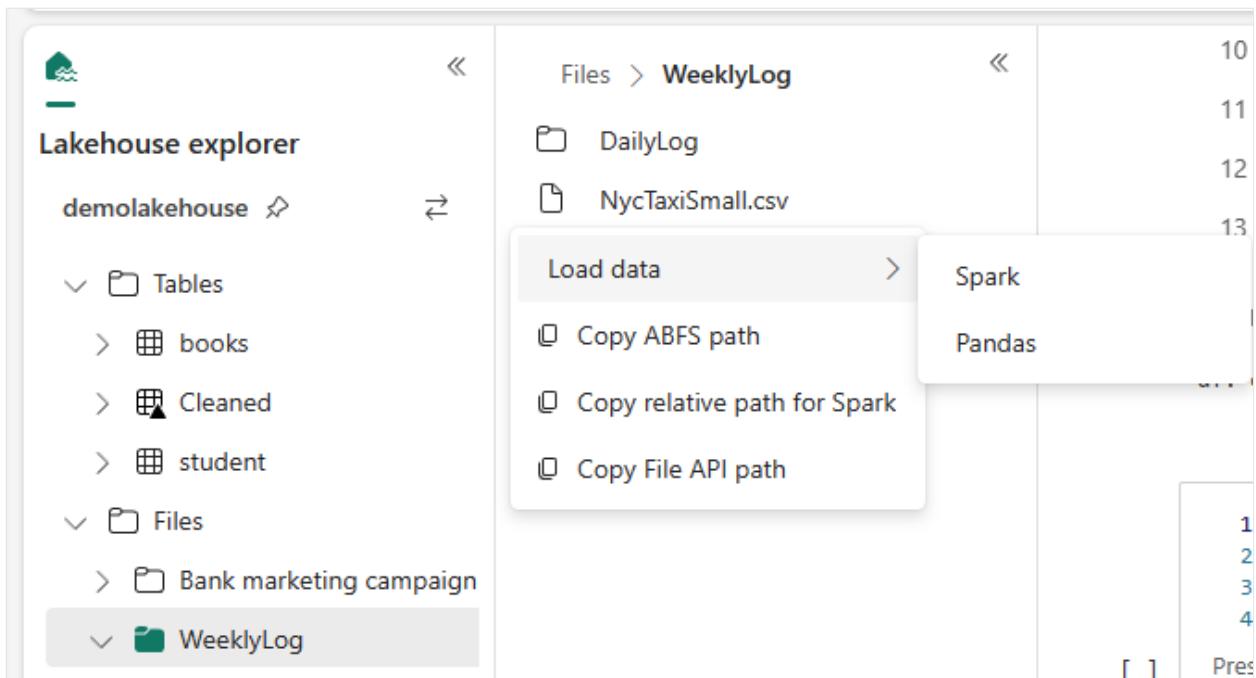
## Explore a lakehouse file

The subfolder and files under the **Tables** and **Files** section of the **Lake** view appear in a content area between the lakehouse list and the notebook content. Select different folders in the **Tables** and **Files** section to refresh the content area.

## Folder and File operations

If you select a file(.csv, .parquet, .txt, .jpg, .png, etc) with a right mouse click, both Spark and Pandas API are supported to load the data. A new code cell is generated and inserted to below of the focus cell.

You can easily copy path with different format of the select file or folder and use the corresponding path in your code.



## Collaborate in a notebook

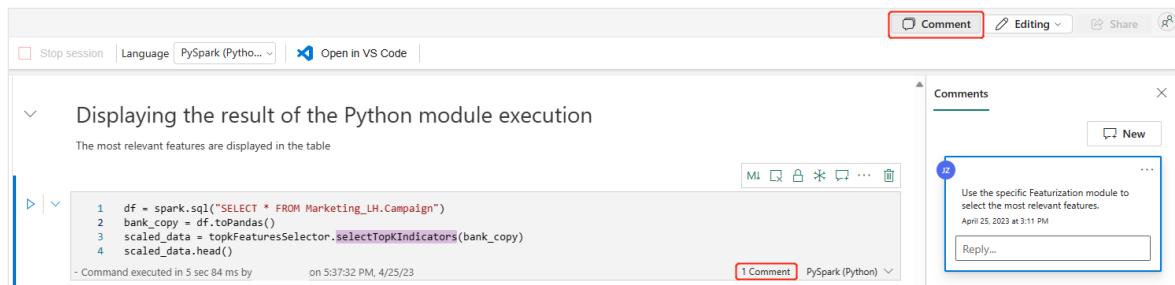
The Microsoft Fabric notebook is a collaborative item that supports multiple users editing the same notebook.

When you open a notebook, you enter the co-editing mode by default, and every notebook edit will be auto-saved. If your colleagues open the same notebook at the same time, you see their profile, run output, cursor indicator, selection indicator and editing trace. By leveraging the collaborating features, you can easily accomplish pair programming, remote debugging, and tutoring scenarios.

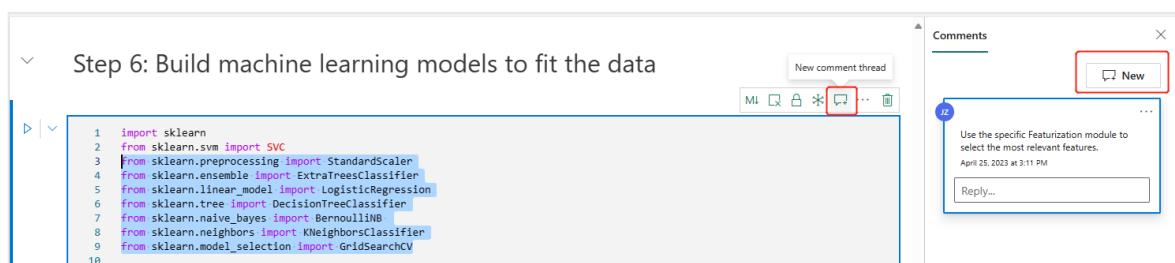
## Comment a code cell

Commenting is another useful feature during collaborative scenarios. Currently, we support adding cell-level comments.

1. Select the **Comments** button on the notebook toolbar or cell comment indicator to open the **Comments** pane.



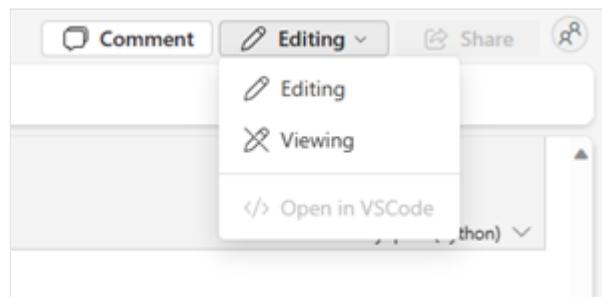
2. Select code in the code cell, select **New** in the **Comments** pane, add comments, and then select the post comment button to save.



3. You could perform **Edit comment**, **Resolve thread**, or **Delete thread** by selecting the More button besides your comment.

## Switch Notebook mode

Fabric notebook support two modes for different scenarios, you can easily switch between **Editing** mode and **Viewing** mode.



- **Editing mode:** You can edit and run the cells and collaborate with others on the notebook.
- **Viewing mode:** You can only view the cell content, output, and comments of the notebook, all the operations that can lead to change the notebook will be

disabled.

## Next steps

- [Author and execute notebooks](#)

# Develop, execute, and manage Microsoft Fabric notebooks

Article • 05/23/2023

Microsoft Fabric notebook is a primary code item for developing Apache Spark jobs and machine learning experiments. It's a web-based interactive surface used by data scientists and data engineers to write code benefiting from rich visualizations and Markdown text. This article explains how to develop notebooks with code cell operations and run them.

## Important

Microsoft Fabric is currently in PREVIEW. This information relates to a prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, expressed or implied, with respect to the information provided here.

## Develop notebooks

Notebooks consist of cells, which are individual blocks of code or text that can be run independently or as a group.

We provide rich operations to develop notebooks:

- [Add a cell](#)
- [Set a primary language](#)
- [Use multiple languages](#)
- [IDE-style IntelliSense](#)
- [Code snippets](#)
- [Drag and drop to insert snippets](#)
- [Drag and drop to insert images](#)
- [Format text cell with toolbar buttons](#)
- [Undo or redo cell operation](#)
- [Move a cell](#)
- [Delete a cell](#)
- [Collapse a cell input](#)
- [Collapse a cell output](#)
- [Lock or freeze a cell](#)
- [Notebook contents](#)
- [Markdown folding](#)
- [Find and replace](#)

## Add a cell

There are multiple ways to add a new cell to your notebook.

1. Hover over the space between two cells and select **Code** or **Markdown**.
2. Use [Shortcut keys under command mode](#). Press **A** to insert a cell above the current cell.  
Press **B** to insert a cell below the current cell.

## Set a primary language

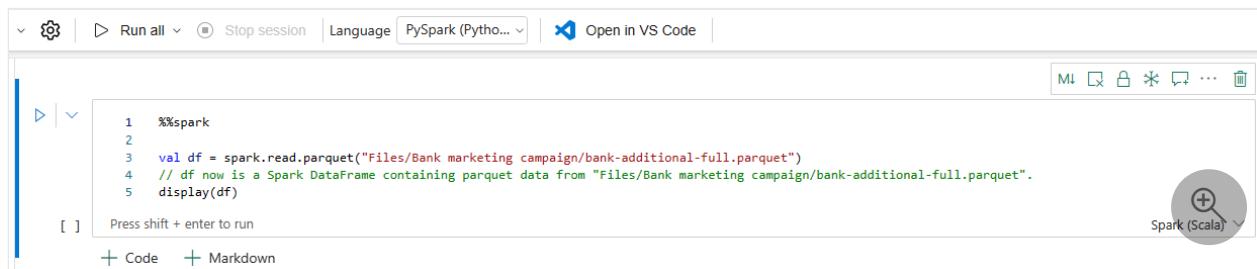
Microsoft Fabric notebooks currently support four Apache Spark languages:

- PySpark (Python)
- Spark (Scala)
- Spark SQL
- SparkR

You can set the primary language for new added cells from the dropdown list in the top command bar.

## Use multiple languages

You can use multiple languages in a notebook by specifying the language magic command at the beginning of a cell, you can also switch the cell language from the language picker. The following table lists the magic commands to switch cell languages.



```
1 %%spark
2
3 val df = spark.read.parquet("Files/Bank marketing campaign/bank-additional-full.parquet")
4 // df now is a Spark DataFrame containing parquet data from "Files/Bank marketing campaign/bank-additional-full.parquet".
5 display(df)
```

Press shift + enter to run

+ Code + Markdown

Magic command	Language	Description
%%pyspark	Python	Execute a <b>Python</b> query against Spark Context.
%%spark	Scala	Execute a <b>Scala</b> query against Spark Context.
%%sql	SparkSQL	Execute a <b>SparkSQL</b> query against Spark Context.
%%html	Html	Execute a <b>HTML</b> query against Spark Context.
%%sparkr	R	Execute a <b>R</b> query against Spark Context.

The following image is an example of how you can write a PySpark query using the **%%pyspark** magic command in a **Spark(Scala)** notebook. Notice that the primary language

for the notebook is set to PySpark.

## IDE-style IntelliSense

Microsoft Fabric notebooks are integrated with the Monaco editor to bring IDE-style IntelliSense to the cell editor. Syntax highlight, error marker, and automatic code completions help you to write code and identify issues quicker.

The IntelliSense features are at different levels of maturity for different languages. The following table shows what's supported:

Languages	Syntax Highlight	Syntax Error Marker	Syntax Code Completion	Variable Code Completion	System Function Code Completion	User Function Code Completion	Smart Indent	Code Folding
PySpark (Python)	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Spark (Scala)	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
SparkSQL	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes
SparkR	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

### ⓘ Note

An active Spark session is required to make use of the IntelliSense code completion.

## Code snippets

Microsoft Fabric notebooks provide code snippets that help you write commonly used code patterns easily, like:

- Reading data as a Spark DataFrame, or
- Drawing charts with Matplotlib.

Snippets appear in [Shortcut keys of IDE style IntelliSense](#) mixed with other suggestions. The code snippets contents align with the code cell language. You can see available snippets by typing **Snippet** or any keywords appear in the snippet title in the code cell editor. For example, by typing **read** you can see the list of snippets to read data from various data sources.

The screenshot shows a Jupyter Notebook interface with a toolbar at the top. Below the toolbar, there's a code cell containing the following Python code:

```
1 # This is an empty notebook
2 # Enter some code here!
3
4 ✓ - Starting Apache Spark session
```

Below the code cell, there are two sections labeled "part1" and "part2". "part1" contains a circular icon with a checkmark and the text "Add lakehouse". "part2" contains a button labeled "Add". At the bottom of the interface, there's a snippet of code: "1 print("11111")".

## Drag and drop to insert snippets

You can use drag and drop to read data from Lakehouse explorer conveniently. Multiple file types are supported here, you can operate on text files, tables, images, etc. You can either drop to an existing cell or to a new cell. The notebook generates the code snippet accordingly to preview the data.

The screenshot shows a Jupyter Notebook interface with a sidebar on the left labeled "TestLH". The sidebar has tabs for "Lake view" and "Table view", with "Lake view" selected. Under "Tables", there is a "Tables" tab and a "Files" tab, which is currently active. The "Files" tab shows a list of CSV files: "aisles.csv", "order\_products\_\_train.csv", "orders.csv", "products.csv", and "test".

The main area of the interface displays a notebook cell titled "Predict NYC Taxi Tips using Spark ML and Azure Open Datasets". The cell contains the following text:

Predict NYC Taxi Tips using Spark ML and Azure Open Datasets

The notebook ingests, visualizes, prepares and then trains a model based on an Open Dataset that tracks NYC Yellow Taxi trips and v for a given trip whether there will be a tip or not.

Test Drag & Drop

1 print("code cell #1")  
Press shift + enter to run

1 print("code cell #2")  
Press shift + enter to run

+ Code + Markdown

1 import matplotlib.pyplot as plt  
2  
3 from pyspark.sql.functions import unix\_timestamp

## Drag and drop to insert images

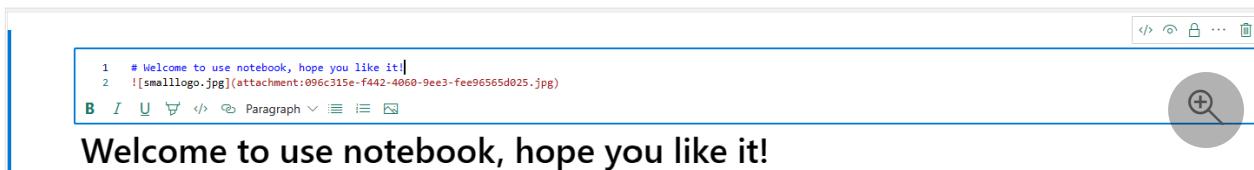
You can use drag and drop to insert images from your browser or local computer to a markdown cell conveniently.

The screenshot shows a Jupyter Notebook interface with a code cell containing the following Python code:

```
1 import random
2 import datetime
3 import pathlib
4 import os
5 import numpy as np
6 import pandas as pd
7
8 # less than 50M
9 # [0, 5], add 0.1% float/string, 10% missing value
10 # date, 1 months, 0.1 incorrect date
11 # UID, report
12 # PID, product id
13
14
15 def convert(line, month, day):
```

# Format text cell with toolbar buttons

You can use the format buttons in the text cells toolbar to do common markdown actions.



## Undo or redo cell operation

Select the **Undo** or **Redo** button, or press **Z** or **Shift+Z** to revoke the most recent cell operations. You can undo or redo up to the latest 10 historical cell operations.



Supported undo cell operations:

- Insert or delete cell: You could revoke the delete operations by selecting **Undo**, the text content is kept along with the cell.
- Reorder cell.
- Toggle parameter.
- Convert between code cell and Markdown cell.

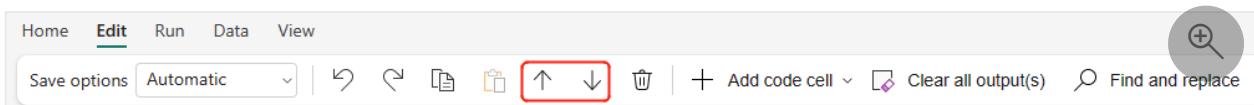
### (!) Note

In-cell text operations and code cell commenting operations can't be undone. You can undo or redo up to the latest 10 historical cell operations.

## Move a cell

You can drag from the empty part of a cell and drop it to the desired position.

You can also move the selected cell using **Move up** and **Move down** on the ribbon.



## Delete a cell

To delete a cell, select the delete button at the right hand of the cell.

You can also use **shortcut keys under command mode**. Press **Shift+D** to delete the current cell.

## Collapse a cell input

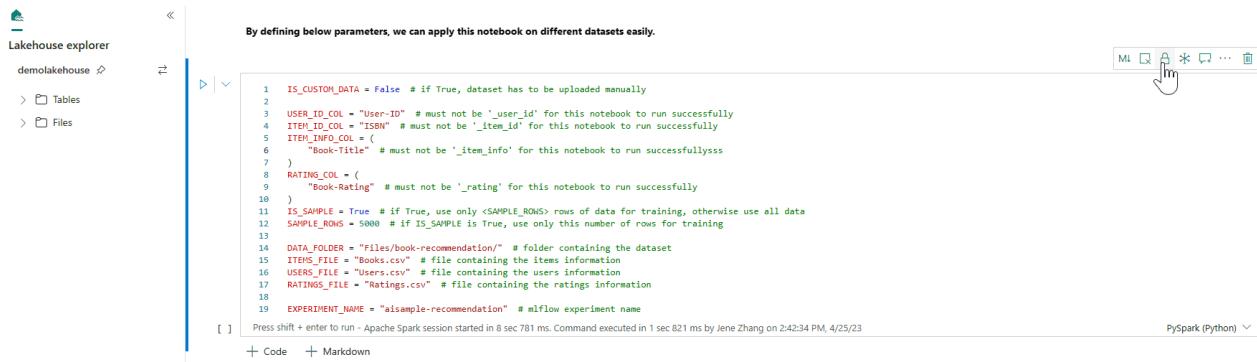
Select the **More commands** ellipses (...) on the cell toolbar and **Hide input** to collapse current cell's input. To expand it, Select the **Show input** while the cell is collapsed.

## Collapse a cell output

Select the **More commands** ellipses (...) on the cell toolbar and **Hide output** to collapse current cell's output. To expand it, select the **Show output** while the cell's output is hidden.

## Lock or freeze a cell

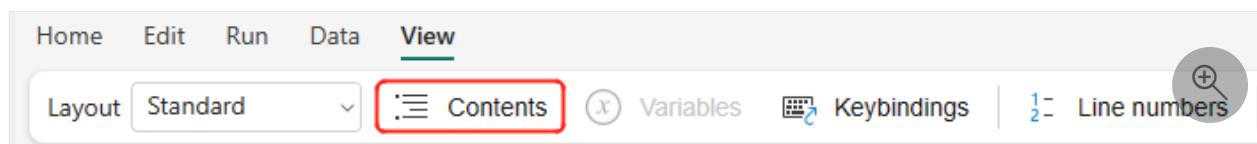
Lock and freeze cell operations allow you to make cells read-only or stop code cells from being run on an individual cell basis.



The screenshot shows a Jupyter Notebook interface with several code cells. The first cell contains Python code related to a recommendation system, including variables like `IS_CUSTOM_DATA`, `USER_ID_COL`, `ITEM_ID_COL`, `ITEM_INFO_COL`, `RATING_COL`, `IS_SAMPLE`, `SAMPLE_ROWS`, `DATA_FOLDER`, `ITEMS_FILE`, `USERS_FILE`, `RATINGS_FILE`, and `EXPERIMENT_NAME`. The second cell is a blank code cell. The third cell is a blank markdown cell. The fourth cell contains the same Python code as the first, with a note at the bottom: "Press shift + enter to run - Apache Spark session started in 1 sec 821 ms. Command executed in 1 sec 821 ms by Jene Zhang on 2:42:34 PM, 4/25/23". The status bar at the bottom right indicates "PySpark (Python)".

## Notebook contents

The Outlines or Table of Contents presents the first markdown header of any markdown cell in a sidebar window for quick navigation. The Outlines sidebar is resizable and collapsible to fit the screen in the best ways possible. You can select the **Contents** button on the notebook command bar to open or hide the sidebar.



## Markdown folding

The markdown folding allows you to hide cells under a markdown cell that contains a heading. The markdown cell and its hidden cells are treated the same as a set of contiguous

multi-selected cells when performing cell operations.

The screenshot shows a Jupyter Notebook interface. The title bar reads "Creating, Evaluating, and Deploying a Recommendation System". The left sidebar, titled "Lakehouse explorer", lists several steps: "Introduction", "Step 1: Load the Data", "Step 2: Exploratory Data Analysis", "Step 3: Model development and deploy", "Step 4: Save Prediction Results", and "Offline Recommendation". The main content area displays the "Introduction" section, which includes a flowchart titled "Recommendation System". The flowchart starts with "Recommendation System" at the top, branching into "Content Based Filtering" (red box) and "Collaborative Filtering" (orange box). "Content Based Filtering" leads to "Memory Based", which further branches into "User-based filtering" and "Item-based filtering". "Collaborative Filtering" leads to "Model Based", which further branches into "Matrix Factorization" (green box) and "Hybrid" (blue box). Below the flowchart, there are two buttons: "+ Code" and "+ Markdown". The status bar at the bottom indicates "Ready".

## Find and replace

Find and replace can help you easily match and locate the keywords or expression within your notebook content, and you can replace the target string with a new string.

The screenshot shows a Jupyter Notebook interface with the "Edit" tab selected. The left sidebar, titled "Lakehouse explorer", has a "Find" section open, showing a search input field with "sample" and a "Replace" section with a "Replace with..." input field. Below it, a "Replace" button is highlighted in green. The main content area displays a notebook cell with the title "Getting started with your Notebook". The cell contains the following code:

```
1 # Use the 2 magic commands below to reload the modules if your module has updates during the cu
2 # %load_ext autoreload
```

Below this, another cell contains:

```
1 # %autoreload 2
2
3 import builtin.Code.SampleModule as SampleModule
4 # Now use the exported members from this module with identifier: `SampleModule`
5 dir(SampleModule)
6
```

At the bottom of the notebook, there is a search icon with a magnifying glass.

## Run notebooks

You can run the code cells in your notebook individually or all at once. The status and progress of each cell is represented in the notebook.

## Run a cell

There are several ways to run the code in a cell.

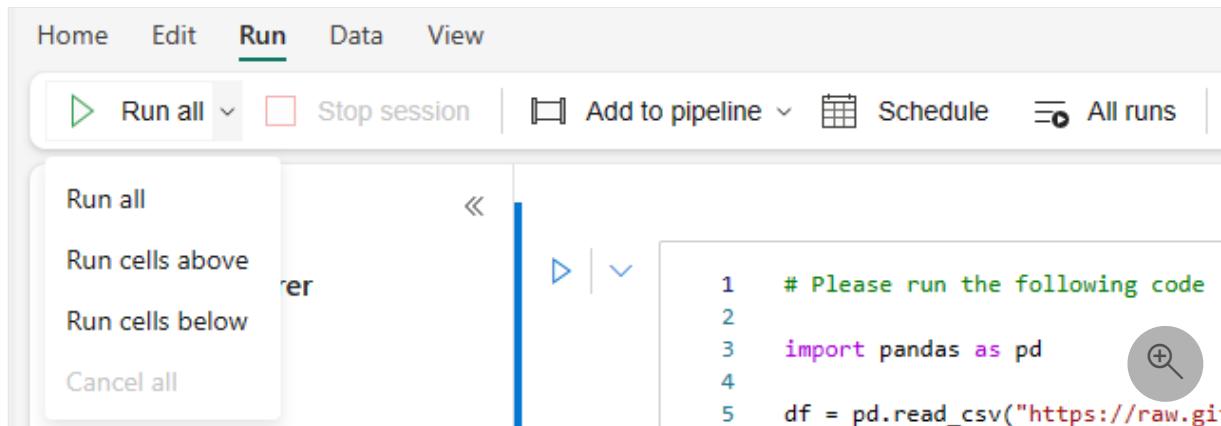
1. Hover on the cell you want to run and select the Run Cell button or press **Ctrl+Enter**.
2. Use [Shortcut keys under command mode](#). Press **Shift+Enter** to run the current cell and select the next cell. Press **Alt+Enter** to run the current cell and insert a new cell.

## Run all cells

Select the **Run All** button to run all the cells in the current notebook in sequence.

## Run all cells above or below

Expand the dropdown list from **Run all** button, then select **Run cells above** to run all the cells above the current in sequence. Select **Run cells below** to run the current cell and all the cells below the current in sequence.

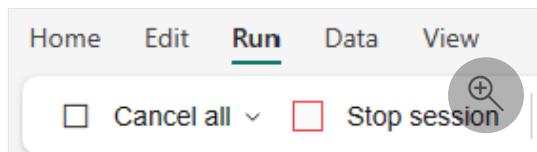


## Cancel all running cells

Select the **Cancel All** button to cancel the running cells or cells waiting in the queue.

## Stop session

**Stop session** cancels the running and waiting cells and stops the current session, you can restart a brand new session if you click the run button again.



## Notebook reference run

Besides using [mssparkutils reference run API](#) You can also use `%run <notebook name>` magic command to reference another notebook within current notebook's context. All the variables defined in the reference notebook are available in the current notebook. `%run` magic

command supports nested calls but not support recursive calls. You'll receive an exception if the statement depth is larger than **five**.

Example: `%run Notebook1 { "parameterInt": 1, "parameterFloat": 2.5, "parameterBool": true, "parameterString": "abc" }`.

Notebook reference works in both interactive mode and pipeline.

### (!) Note

- `%run` command currently only supports reference notebooks that in the same workspace with current notebook.
- `%run` command currently only supports to 4 parameter value types: `int`, `float`, `bool`, `string`, variable replacement operation is not supported.
- `%run` command do not support nested reference that depth is larger than **five**.

## Variable explorer

Microsoft Fabric notebook provides a built-in variables explorer for you to see the list of the variables name, type, length, and value in the current Spark session for PySpark (Python) cells. More variables show up automatically as they're defined in the code cells. Clicking on each column header sorts the variables in the table.

You can select the **Variables** button on the notebook ribbon "View" tab to open or hide the variable explorer.

The screenshot shows the Microsoft Fabric notebook interface with the "Variables" button highlighted in the ribbon's "View" tab. The "Variables" section of the Lakehouse explorer displays a list of variables:

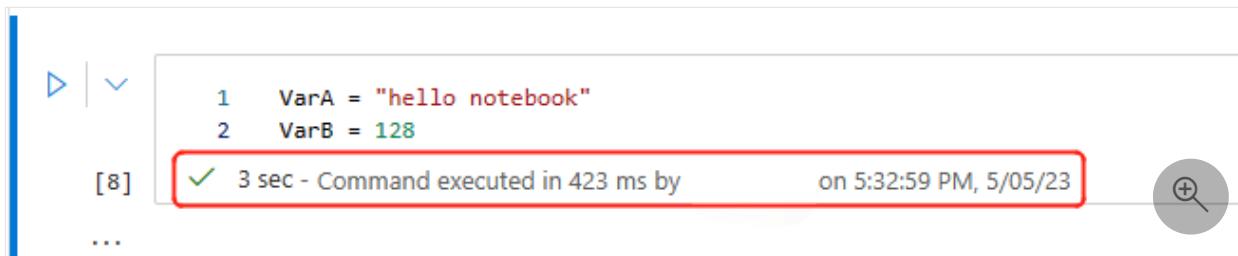
Name ↑	Type	Length	Value
a	int	10	
displayHTML	DisplayHTML		
HiveContext	type		
sc	SparkContext		
spark	SparkSession		
sqlContext	SQLContext		
StreamingContext	type		
VarA	str	14	'hello notebook'
VarB	int		128

## ① Note

Variable explorer only supports python.

## Cell status indicator

A step-by-step cell execution status is displayed beneath the cell to help you see its current progress. Once the cell run is complete, an execution summary with the total duration and end time is shown and stored there for future reference.



## Inline spark job indicator

The Microsoft Fabric notebook is Spark based. Code cells are executed on the spark cluster remotely. A Spark job progress indicator is provided with a real-time progress bar appears to help you understand the job execution status. The number of tasks per each job or stage help you to identify the parallel level of your spark job. You can also drill deeper to the Spark UI of a specific job (or stage) via selecting the link on the job (or stage) name.

You can also find the **Cell level real-time log** next to the progress indicator, and **Diagnostics** can provide you with useful suggestions to help refine and debug the code.

ID	Description	Status	Stages	Tasks	Duration
Job 7	load at NativeMethodAccessorImpl.java:0	Succeeded	1/1	1/1 succeeded	1 sec
Job 8	getRowsInJsonString at Display.scala:403	Succeeded	1/1	1/1 succeeded	< 1 ms

Spark jobs (2 of 2 succeeded) Log

Diagnostics 1

May include corrupted records within your file(s)

In **More actions**, you can easily navigate to the **Spark application details** page and **Spark web UI** page.

ID	Description	Status	Stages	Tasks	Duration	Processes
Job 9	load at NativeMethodAccessorImpl.java:0	Succeeded	1/1	1/1 succeeded	< 1 ms	1 row

Spark jobs (2 of 2 succeeded) Log

More actions

Spark application details

Spark web UI

## Secret redaction

To prevent the credentials being accidentally leaked when running notebooks, Fabric notebook support **Secret redaction** to replace the secret values that are displayed in the cell output with [REDACTED]. Secret redaction is applicable for **Python, Scala and R**.

- ✓ User-oriented interface to get secret.

```
1 %%pyspark
2 mssparkutils.credentials.getSecret("https://kvtest.vault.azure.net/", "test")
```

[6] ✓ - Command executed in 404 ms by on 4:21:07 PM, 4/25/23  
[REDACTED]

PySpark (Python) ▾

- ✓ User-oriented interface to get token.

```
1 %%pyspark
2 print(mssparkutils.credentials.getToken("https://kusto.kusto.windows.net"))
3 print(mssparkutils.credentials.getToken("pb1"))
```

[7] ✓ - Command executed in 2 sec 612 ms by on 4:21:21 PM, 4/25/23  
[REDACTED]  
[REDACTED]

PySpark (Python) ▾

## Magic commands in notebook

### Built-in magics

You can use familiar Ipython magic commands in Fabric notebooks. Review the following list as the current available magic commands.

#### ⓘ Note

Only following magic commands are supported in Fabric pipeline : %%pyspark, %%spark, %%csharp, %%sql.

Available line magics: [%lsmagic](#), [%time](#), [%timeit](#), [%history](#), [%run](#), [%load](#), [%alias](#), [%alias\\_magic](#), [%autoawait](#), [%autocall](#), [%automagic](#), [%bookmark](#), [%cd](#), [%colors](#), [%dhist](#), [%dirs](#), [%doctest\\_mode](#), [%killbgscripts](#), [%load\\_ext](#), [%logoff](#), [%logon](#), [%logstart](#), [%logstate](#), [%logstop](#), [%magic](#), [%matplotlib](#), [%page](#), [%pastebin](#), [%pdef](#), [%pfile](#), [%pinfo](#), [%pinfo2](#), [%popd](#), [%pprint](#), [%precision](#), [%prun](#), [%psearch](#), [%psource](#), [%pushd](#), [%pwd](#), [%pycat](#), [%quickref](#), [%rehashx](#), [%reload\\_ext](#), [%reset](#), [%reset\\_selective](#), [%sx](#), [%system](#), [%tb](#), [%unalias](#), [%unload\\_ext](#), [%who](#), [%who\\_ls](#), [%whos](#), [%xdel](#), [%xmode](#).

Fabric notebook also supports improved library management commands [%pip](#), [%conda](#), check [Manage Apache Spark libraries in Microsoft Fabric](#) for the usage.

Available cell magics: [%%time](#), [%%timeit](#), [%%capture](#), [%%writefile](#), [%%sql](#), [%%pyspark](#), [%%spark](#), [%%csharp](#), [%%html](#), [%%bash](#), [%%markdown](#), [%%perl](#), [%%script](#), [%%sh](#).

## Custom magics

You can also build out more custom magic commands to meet your specific needs as the below example shows.

1. Create a notebook with name "MyLakehouseModule".

Register My magic command with IPython in Notebook: MyLakehouseModule

```
1 @magics_class
2 class MyLakeHouseMagic(Magics):
3     @line_magic
4     def list_lh(self):
5         "list all my lakehouse account"
6         # do operations
7         return ['LH1', 'LH2']
8
9     @line_magic
10    def create_lh(self, name):
11        "create lakehouse with name"
12        # do operations
13        return "create Lakehouse with name: " + name
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
```

Press shift + enter to run



2. In another notebook reference the "MyLakehouseModule" and its magic commands, by this way you can organize your project with notebooks that using different languages conveniently.

```
1 %run MyLakehouseModule
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
229
230
231
232
233
234
235
236
237
238
239
239
240
241
242
243
244
245
246
247
248
249
249
250
251
252
253
254
255
256
257
258
259
259
260
261
262
263
264
265
266
267
268
269
269
270
271
272
273
274
275
276
277
278
279
279
280
281
282
283
284
285
286
287
287
288
289
289
290
291
292
293
294
295
295
296
297
297
298
299
299
300
301
302
303
303
304
305
305
306
306
307
307
308
308
309
309
310
310
311
311
312
312
313
313
314
314
315
315
316
316
317
317
318
318
319
319
320
320
321
321
322
322
323
323
324
324
325
325
326
326
327
327
328
328
329
329
330
330
331
331
332
332
333
333
334
334
335
335
336
336
337
337
338
338
339
339
340
340
341
341
342
342
343
343
344
344
345
345
346
346
347
347
348
348
349
349
350
350
351
351
352
352
353
353
354
354
355
355
356
356
357
357
358
358
359
359
360
360
361
361
362
362
363
363
364
364
365
365
366
366
367
367
368
368
369
369
370
370
371
371
372
372
373
373
374
374
375
375
376
376
377
377
378
378
379
379
380
380
381
381
382
382
383
383
384
384
385
385
386
386
387
387
388
388
389
389
390
390
391
391
392
392
393
393
394
394
395
395
396
396
397
397
398
398
399
399
400
400
```

Use it everywhere, in every language

```
1 %create_lh "mylakehouse"
2 ✓ 20 sec - Command executed in 9 sec 973 ms by mgr nbs on 4:32:09 PM, 6/23/22
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
59
60
61
62
63
64
65
66
67
68
69
69
70
71
72
73
74
75
76
77
78
79
79
80
81
82
83
84
85
86
87
88
89
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
109
110
111
112
113
114
115
116
117
118
119
119
120
121
122
123
124
125
126
127
128
129
129
130
131
132
133
134
135
136
137
138
139
139
140
141
142
143
144
145
146
147
148
149
149
150
151
152
153
154
155
156
157
158
159
159
160
161
162
163
164
165
166
167
168
169
169
170
171
172
173
174
175
176
177
178
178
179
180
181
182
183
184
185
186
187
188
189
189
190
191
192
193
194
195
196
197
198
199
199
200
201
202
203
204
205
206
207
208
209
209
210
211
212
213
214
215
216
217
218
219
219
220
221
222
223
224
225
226
227
228
229
229
230
231
232
233
234
235
236
237
238
239
239
240
241
242
243
244
245
246
247
248
249
249
250
251
252
253
254
255
256
257
258
259
259
260
261
262
263
264
265
266
267
268
269
269
270
271
272
273
274
275
276
277
278
279
279
280
281
282
283
284
285
286
287
288
288
289
289
290
291
292
293
294
295
296
297
297
298
299
299
300
301
302
303
304
305
306
307
308
309
309
310
311
312
313
314
315
316
317
318
319
319
320
321
322
323
324
325
326
327
328
329
329
330
331
332
333
334
335
336
337
338
339
339
340
341
342
343
344
345
346
347
348
349
349
350
351
352
353
354
355
356
357
358
359
359
360
361
362
363
364
365
366
367
368
369
369
370
371
372
373
374
375
376
377
378
379
379
380
381
382
383
384
385
386
387
388
389
389
390
391
392
393
394
395
396
397
398
399
399
400
400
```



## IPython Widgets

IPython Widgets are eventful python objects that have a representation in the browser. You can use IPython Widgets as low-code controls (for example, slider, text box) in your notebook just like the Jupyter notebook, currently it only works in Python context.

### To use IPython Widget

1. You need to import *ipywidgets* module first to use the Jupyter Widget framework.

Python

```
import ipywidgets as widgets
```

2. You can use top-level *display* function to render a widget, or leave an expression of *widget* type at the last line of code cell.

Python

```
slider = widgets.IntSlider()
display(slider)
```

3. Run the cell, the widget displays in the output area.

Python

```
slider = widgets.IntSlider()
display(slider)
```

```
1 import ipywidgets as widgets
2 slider = widgets.IntSlider()
3 display(slider)
```

✓ 3 sec - Command executed in 353 ms by

6:13:08 PM, 5/05/23

PySpark (Python) ✓



4. You can use multiple *display()* calls to render the same widget instance multiple times, but they remain in sync with each other.

Python

```
slider = widgets.IntSlider()
display(slider)
display(slider)
```

```
1 slider = widgets.IntSlider()
2 display(slider)
3 display(slider)
```

✓ 4 sec - Command executed in 358 ms by

on 6:14:17 PM, 5/05/23

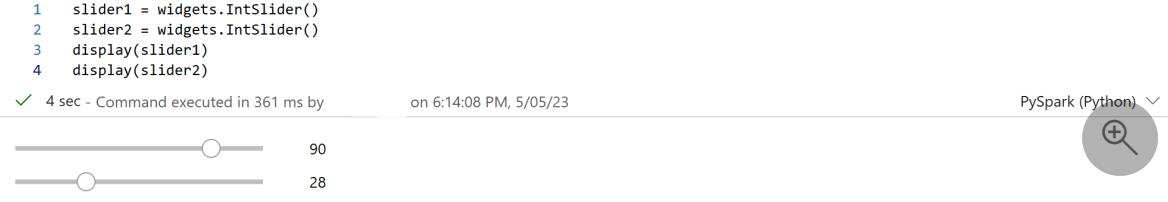
PySpark (Python) ✓



5. To render two widgets independent of each other, create two widget instances:

Python

```
slider1 = widgets.IntSlider()
slider2 = widgets.IntSlider()
display(slider1)
display(slider2)
```



```

1 slider1 = widgets.IntSlider()
2 slider2 = widgets.IntSlider()
3 display(slider1)
4 display(slider2)

```

4 sec - Command executed in 361 ms by on 6:14:08 PM, 5/05/23

PySpark (Python) ▾

## Supported widgets

Widgets type	Widgets
Numeric widgets	IntSlider, FloatSlider, FloatLogSlider, IntRangeSlider, FloatRangeSlider, IntProgress, FloatProgress, BoundedIntText, BoundedFloatText, IntText, FloatText
Boolean widgets	ToggleButton, Checkbox, Valid
Selection widgets	Dropdown, RadioButtons, Select, SelectionSlider, SelectionRangeSlider, ToggleButtons, SelectMultiple
String Widgets	Text, Text area, Combobox, Password, Label, HTML, HTML Math, Image, Button
Play (Animation) widgets	Date picker, Color picker, Controller
Container or Layout widgets	Box, HBox, VBox, GridBox, Accordion, Tabs, Stacked

## Known limitations

1. The following widgets aren't supported yet, you could follow the corresponding workaround as follows:

Functionality	Workaround
Output widget	You can use <code>print()</code> function instead to write text into stdout.
<code>widgets.jslink()</code>	You can use <code>widgets.link()</code> function to link two similar widgets.
<code>FileUpload</code> widget	Not supported yet.

2. Global `display` function provided by Microsoft Fabric doesn't support displaying multiple widgets in one call (i.e. `display(a, b)`), which is different from IPython `display` function.
3. If you close a notebook that contains IPython Widget, you can't see or interact with it until you execute the corresponding cell again.

## Python logging in Notebook

You can find Python logs and set different log levels and format like the sample code shown here:

Python

```
import logging

# Customize the logging format for all loggers
FORMAT = "%(asctime)s - %(name)s - %(levelname)s - %(message)s"
formatter = logging.Formatter(fmt=FORMAT)
for handler in logging.getLogger().handlers:
    handler.setFormatter(formatter)

# Customize log level for all loggers
logging.getLogger().setLevel(logging.INFO)

# Customize the log level for a specific logger
customizedLogger = logging.getLogger('customized')
customizedLogger.setLevel(logging.WARNING)

# logger that use the default global log level
defaultLogger = logging.getLogger('default')
defaultLogger.debug("default debug message")
defaultLogger.info("default info message")
defaultLogger.warning("default warning message")
defaultLogger.error("default error message")
defaultLogger.critical("default critical message")

# logger that use the customized log level
customizedLogger.debug("customized debug message")
customizedLogger.info("customized info message")
customizedLogger.warning("customized warning message")
customizedLogger.error("customized error message")
customizedLogger.critical("customized critical message")
```

## Integrate a notebook

### Designate a parameters cell

To parameterize your notebook, select the ellipses (...) to access the **more commands** at the cell toolbar. Then select **Toggle parameter cell** to designate the cell as the parameters cell.

The screenshot shows a Microsoft Fabric Notebook interface. At the top, there's a toolbar with icons for file operations like 'New', 'Open', 'Save', etc. Below the toolbar, a code cell contains the following Python code:

```

1 SourceData = "path 1"
2 Destination = "Path 2"

```

Below the code cell, it says 'Press shift + enter to run'. To the right of the code cell is a dropdown menu with various options: 'Move cell up', 'Move cell down', 'Hide input', 'Hide output', 'Toggle parameter cell' (which is highlighted with a red border), 'Merge with previous cell', 'Merge with next cell', and 'Split cell'. There's also a magnifying glass icon.

Below the code cell, a section titled 'Welcome to use New Fabric Notebook' is expanded. It contains a single line of code:

```

1

```

Below this line, it says 'Press shift + enter to run'.

The parameter cell is useful for integrating notebook in pipeline, pipeline activity looks for the parameters cell and treats this cell as defaults for the parameters passed in at execution time. The execution engine adds a new cell beneath the parameters cell with input parameters in order to overwrite the default values.

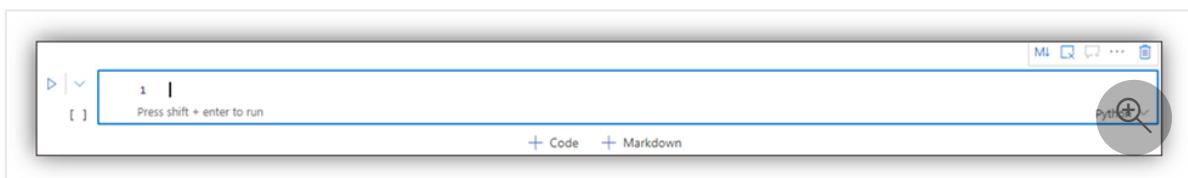
## Shortcut keys

Similar to Jupyter Notebooks, Microsoft Fabric notebooks have a modal user interface. The keyboard does different things depending on which mode the notebook cell is in. Microsoft Fabric notebooks support the following two modes for a given code cell: command mode and edit mode.

1. A cell is in command mode when there's no text cursor prompting you to type. When a cell is in Command mode, you can edit the notebook as a whole but not type into individual cells. Enter command mode by pressing ESC or using the mouse to select outside of a cell's editor area.



2. Edit mode can be indicated from a text cursor that prompting you to type in the editor area. When a cell is in edit mode, you can type into the cell. Enter edit mode by pressing Enter or using the mouse to select on a cell's editor area.



## Shortcut keys under command mode

Action	Notebook shortcuts
Run the current cell and select below	Shift+Enter

Action	Notebook shortcuts
Run the current cell and insert below	Alt+Enter
Run current cell	Ctrl+Enter
Select cell above	Up
Select cell below	Down
Select previous cell	K
Select next cell	J
Insert cell above	A
Insert cell below	B
Delete selected cells	Shift + D
Switch to edit mode	Enter

## Shortcut keys under edit mode

Using the following keystroke shortcuts, you can more easily navigate and run code in Microsoft Fabric notebooks when in Edit mode.

Action	Notebook shortcuts
Move cursor up	Up
Move cursor down	Down
Undo	Ctrl + Z
Redo	Ctrl + Y
Comment or Uncomment	Ctrl + /
Delete word before	Ctrl + Backspace
Delete word after	Ctrl + Delete
Go to cell start	Ctrl + Home
Go to cell end	Ctrl + End
Go one word left	Ctrl + Left
Go one word right	Ctrl + Right
Select all	Ctrl + A
Indent	Ctrl + ]

Action	Notebook shortcuts
Dedent	Ctrl + [
Switch to command mode	Esc

You can easily find all shortcut keys from notebook ribbon *View -> Keybindings*.

## Next steps

- [Notebook visualization](#)
- [Introduction of Fabric MSSparkUtils](#)

# Notebook visualization in Microsoft Fabric

Article • 05/23/2023

Microsoft Fabric is an integrated analytics service that accelerates time to insight, across data warehouses and big data analytics systems. Data visualization in notebook is a key component in being able to gain insight into your data. It helps make big and small data easier for humans to understand. It also makes it easier to detect patterns, trends, and outliers in groups of data.

## ⓘ Important

Microsoft Fabric is currently in PREVIEW. This information relates to a prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, expressed or implied, with respect to the information provided here.

When you use Apache Spark in Microsoft Fabric, there are various built-in options to help you visualize your data, including Microsoft Fabric notebook chart options, and access to popular open-source libraries.

## Notebook chart options

When using a Microsoft Fabric notebook, you can turn your tabular results view into a customized chart using chart options. Here, you can visualize your data without having to write any code.

### display(df) function

The *display* function allows you to turn SQL query result and Apache Spark dataframes into rich data visualizations. The *display* function can be used on dataframes created in PySpark and Scala.

To access the chart options:

1. The output of `%%sql` magic commands appear in the rendered table view by default. You can also call `display(df)` on Spark DataFrames or Resilient Distributed Datasets (RDD) function to produce the rendered table view.
2. Once you have a rendered table view, switch to the **Chart** view.

```

1 df = spark.read.parquet("Files/SampleData.parquet")
2
3 display(df)

```

[1] ✓ 15 sec - Apache Spark session started in 11 sec 452 ms. Command executed in 3 sec 665 ms by PySpark (Python) ...

> Spark Jobs (2 of 2 succeeded) Log ...

Table Chart I→ Export results

Index	age	job	marital	education	default	housing
1	56	housemaid	married	basic.4y	no	no
2	57	services	married	high.school	unknown	no
3	37	services	married	high.school	no	yes
4	40	admin.	married	basic.6y	no	no
5	56	services	married	high.school	no	no
6	45	services	married	basic.9y	unknown	no
7	59	admin.	married	professional.course	no	no
8	41	blue-collar	married	unknown	unknown	no
9	24	technician	single	professional.course	no	yes
10	25	services	single	high.school	no	yes
11	41	blue-collar	married	unknown	unknown	no
12	25	services	single	high.school	no	yes
13	29	blue-collar	single	high.school	no	no
14	57	housemaid	divorced	basic.4y	no	yes

3. You can now customize your visualization by specifying the following values:

Configuration	Description
Chart Type	The display function supports a wide range of chart types, including bar charts, scatter plots, line graphs, and more.
Key	Specify the range of values for the x-axis.
Value	Specify the range of values for the y-axis values.
Series Group	Used to determine the groups for the aggregation.
Aggregation	Method to aggregate data in your visualization.

### ⚠ Note

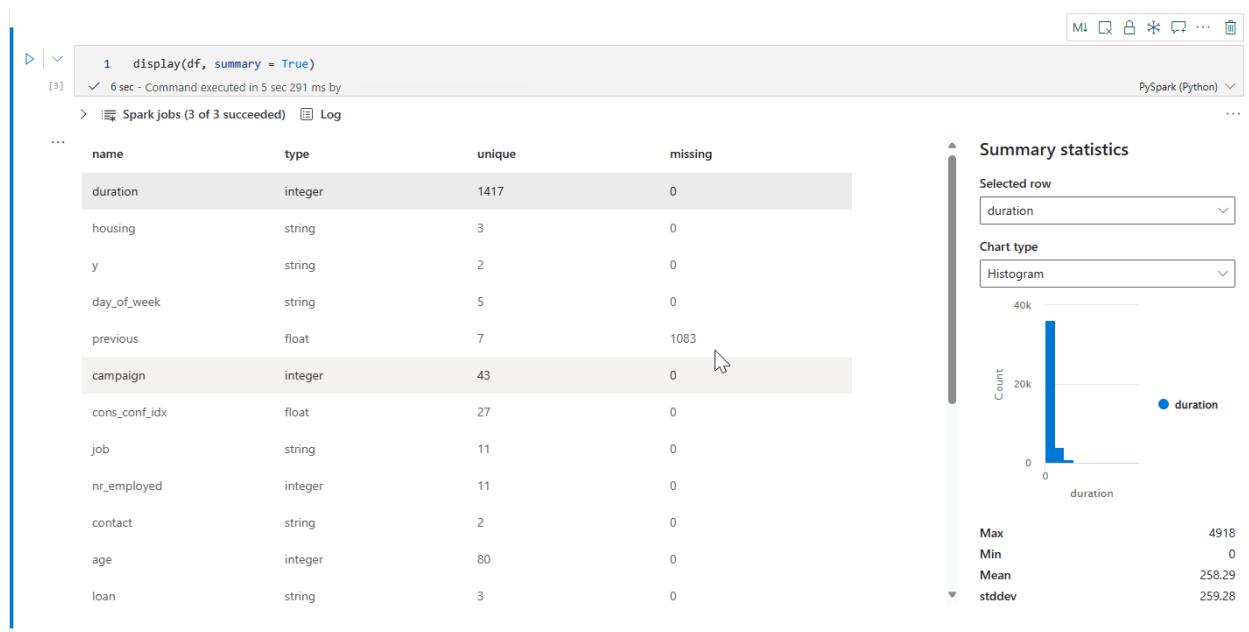
By default the `display(df)` function will only take the first 1000 rows of the data to render the charts. Select **Aggregation over all results** and then select **Apply** to apply the chart generation from the whole dataset. A Spark job will be triggered when the chart setting changes. Please note that it may take several minutes to complete the calculation and render the chart.

4. Once done, you can view and interact with your final visualization!

## display(df) summary view

You can use `display(df, summary = true)` to check the statistics summary of a given Apache Spark DataFrame that includes the column name, column type, unique values,

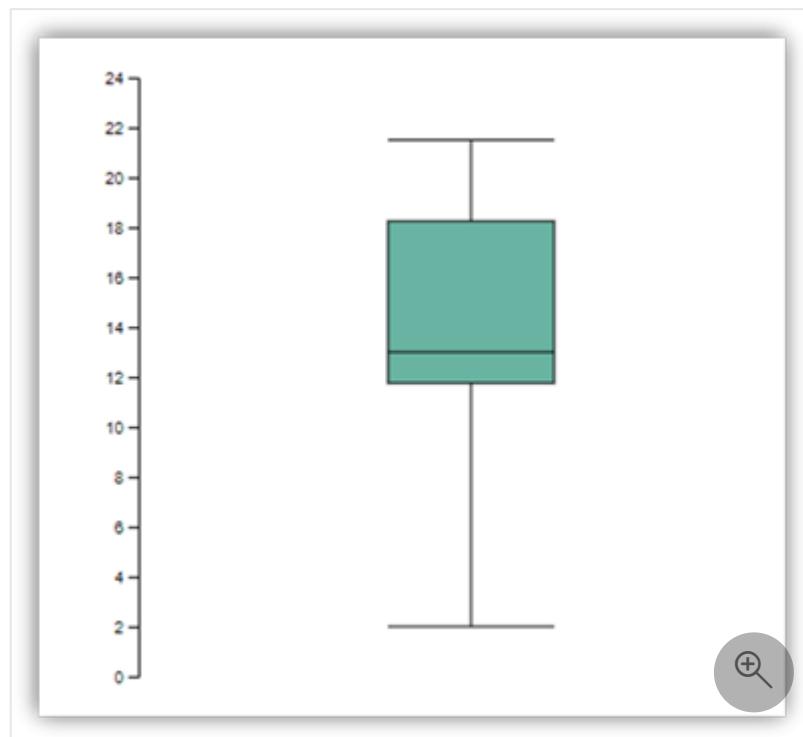
and missing values for each column. You can also select a specific column to see its minimum value, maximum value, mean value, and standard deviation.



## displayHTML() option

Microsoft Fabric notebooks support HTML graphics using the `displayHTML` function.

The following image is an example of creating visualizations using [D3.js](#).



Run the following code to create this visualization.

Python

```
displayHTML("""<!DOCTYPE html>
<meta charset="utf-8">

<!-- Load d3.js -->
<script src="https://d3js.org/d3.v4.js"></script>

<!-- Create a div where the graph will take place -->
<div id="my_dataviz"></div>
<script>

// set the dimensions and margins of the graph
var margin = {top: 10, right: 30, bottom: 30, left: 40},
    width = 400 - margin.left - margin.right,
    height = 400 - margin.top - margin.bottom;

// append the svg object to the body of the page
var svg = d3.select("#my_dataviz")
.append("svg")
    .attr("width", width + margin.left + margin.right)
    .attr("height", height + margin.top + margin.bottom)
.append("g")
    .attr("transform",
        "translate(" + margin.left + "," + margin.top + ")");
    .attr("transform",
        "translate(" + margin.left + "," + margin.top + ")");

// Create Data
var data = [12,19,11,13,12,22,13,4,15,16,18,19,20,12,11,9]

// Compute summary statistics used for the box:
var data_sorted = data.sort(d3.ascending)
var q1 = d3.quantile(data_sorted, .25)
var median = d3.quantile(data_sorted, .5)
var q3 = d3.quantile(data_sorted, .75)
var interQuantileRange = q3 - q1
var min = q1 - 1.5 * interQuantileRange
var max = q1 + 1.5 * interQuantileRange

// Show the Y scale
var y = d3.scaleLinear()
    .domain([0,24])
    .range([height, 0]);
svg.call(d3.axisLeft(y))

// a few features for the box
var center = 200
var width = 100

// Show the main vertical line
svg
.append("line")
    .attr("x1", center)
    .attr("x2", center)
    .attr("y1", y(min) )
    .attr("y2", y(max) )
    .attr("stroke", "black")
```

```

// Show the box
svg
.append("rect")
.attr("x", center - width/2)
.attr("y", y(q3) )
.attr("height", (y(q1)-y(q3)) )
.attr("width", width )
.attr("stroke", "black")
.style("fill", "#69b3a2")

// show median, min and max horizontal lines
svg
.selectAll("toto")
.data([min, median, max])
.enter()
.append("line")
.attr("x1", center-width/2)
.attr("x2", center+width/2)
.attr("y1", function(d){ return(y(d))} )
.attr("y2", function(d){ return(y(d))} )
.attr("stroke", "black")
</script>

"""
)

```

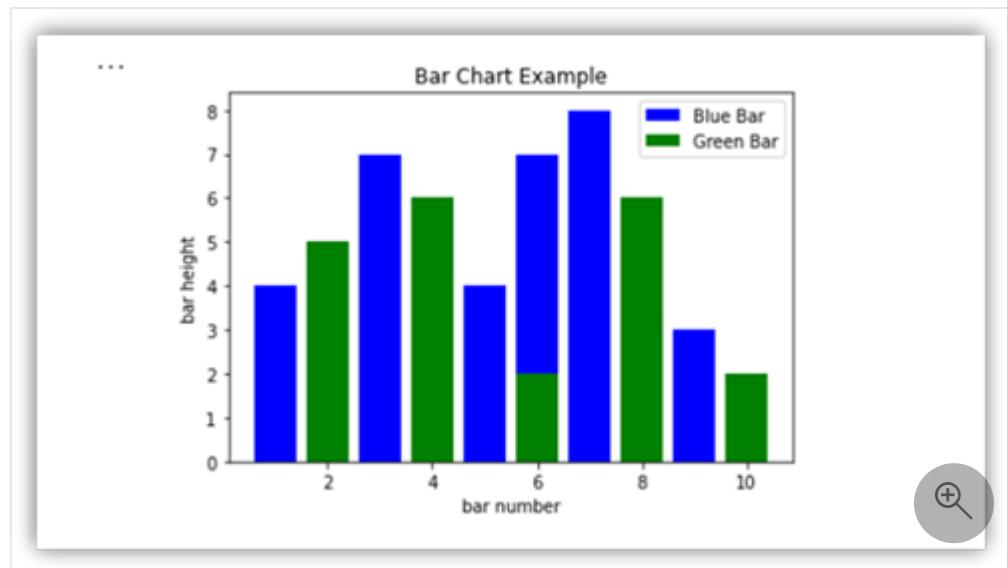
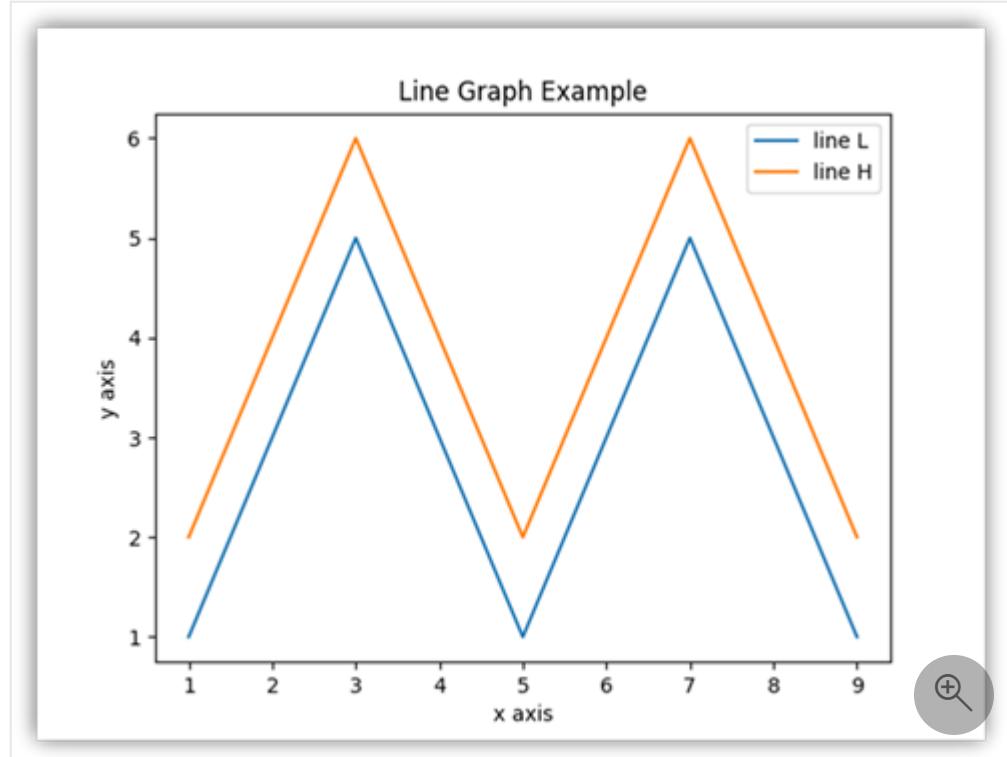
## Popular libraries

When it comes to data visualization, Python offers multiple graphing libraries that come packed with many different features. By default, every Apache Spark pool in Microsoft Fabric contains a set of curated and popular open-source libraries.

### Matplotlib

You can render standard plotting libraries, like Matplotlib, using the built-in rendering functions for each library.

The following image is an example of creating a bar chart using **Matplotlib**.



Run the following sample code to draw this bar chart.

Python

```
# Bar chart

import matplotlib.pyplot as plt

x1 = [1, 3, 4, 5, 6, 7, 9]
y1 = [4, 7, 2, 4, 7, 8, 3]

x2 = [2, 4, 6, 8, 10]
y2 = [5, 6, 2, 6, 2]

plt.bar(x1, y1, label="Blue Bar", color='b')
plt.bar(x2, y2, label="Green Bar", color='g')
```

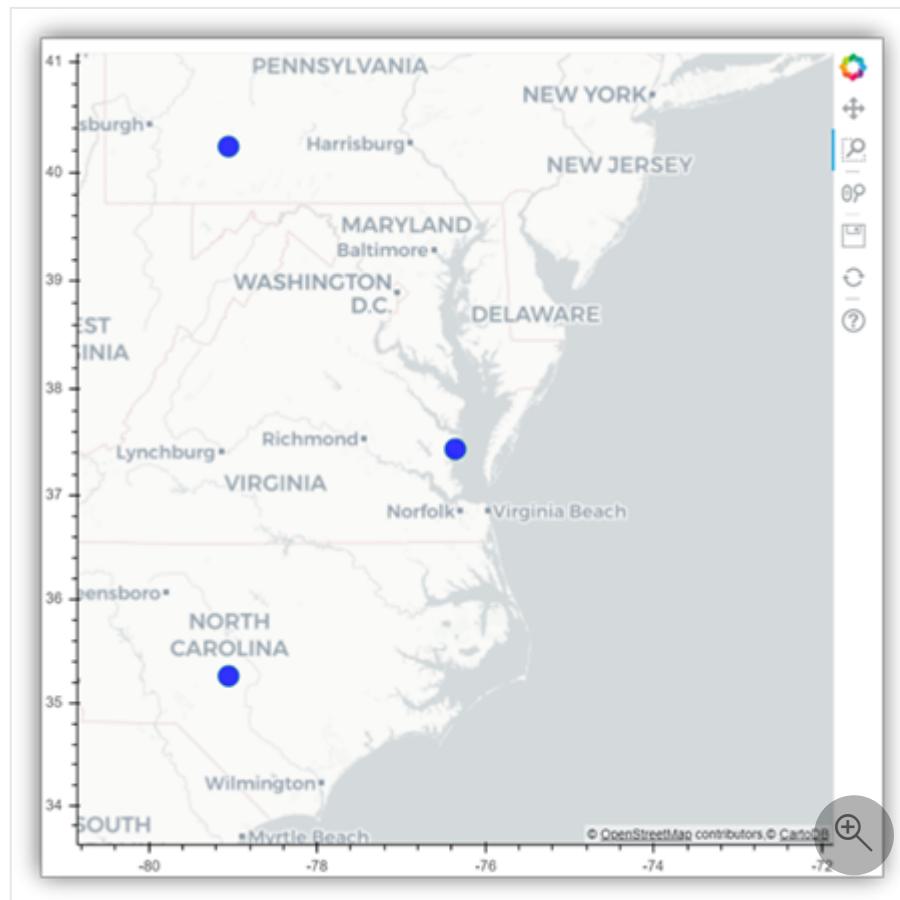
```
plt.plot()

plt.xlabel("bar number")
plt.ylabel("bar height")
plt.title("Bar Chart Example")
plt.legend()
plt.show()
```

## Bokeh

You can render HTML or interactive libraries, like **bokeh**, using the `displayHTML(df)`.

The following image is an example of plotting glyphs over a map using **bokeh**.



Run the following sample code to draw this image.

Python

```
from bokeh.plotting import figure, output_file
from bokeh.tile_providers import get_provider, Vendors
from bokeh.embed import file_html
from bokeh.resources import CDN
from bokeh.models import ColumnDataSource

tile_provider = get_provider(Vendors.CARTODBPOSITRON)

# range bounds supplied in web mercator coordinates
```

```

p = figure(x_range=(-9000000, -8000000), y_range=(4000000, 5000000),
            x_axis_type="mercator", y_axis_type="mercator")
p.add_tile(tile_provider)

# plot datapoints on the map
source = ColumnDataSource(
    data=dict(x=[ -8800000, -8500000 , -8800000],
              y=[4200000, 4500000, 4900000])
)

p.circle(x="x", y="y", size=15, fill_color="blue", fill_alpha=0.8,
         source=source)

# create an html document that embeds the Bokeh plot
html = file_html(p, CDN, "my plot1")

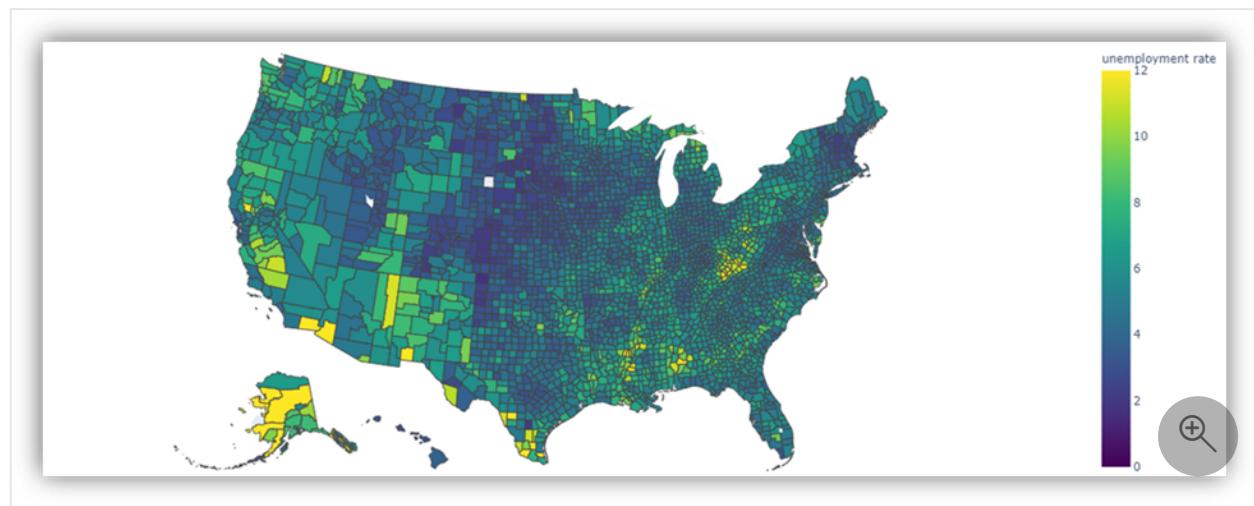
# display this html
displayHTML(html)

```

## Plotly

You can render HTML or interactive libraries like **Plotly**, using the `displayHTML()`.

Run the following sample code to draw this image:



Python

```

from urllib.request import urlopen
import json
with
urlopen('https://raw.githubusercontent.com/plotly/datasets/master/geojson-
counties-fips.json') as response:
    counties = json.load(response)

import pandas as pd
df =
pd.read_csv("https://raw.githubusercontent.com/plotly/datasets/master/fips-

```

```

unemp-16.csv",
      dtype={"fips": str})

import plotly
import plotly.express as px

fig = px.choropleth(df, geojson=counties, locations='fips', color='unemp',
                     color_continuous_scale="Viridis",
                     range_color=(0, 12),
                     scope="usa",
                     labels={'unemp':'unemployment rate'}
)
fig.update_layout(margin={"r":0,"t":0,"l":0,"b":0})

# create an html document that embeds the Plotly plot
h = plotly.offline.plot(fig, output_type='div')

# display this html
displayHTML(h)

```

## Pandas

You can view html output of pandas dataframe as the default output, notebook automatically shows the styled html content.

Model:	Decision Tree		Regression		Random	
Predicted:	Tumour	Non-Tumour	Tumour	Non-Tumour	Tumour	Non-Tumour
Actual Label:						
Tumour (Positive)	38.0	2.0	18.0	22.0	21	NaN
Non-Tumour (Negative)	19.0	439.0	6.0	452.0	226	232.0

## Python

```

import pandas as pd
import numpy as np

df = pd.DataFrame([[38.0, 2.0, 18.0, 22.0, 21, np.nan],[19, 439, 6, 452,
226,232]],

                   index=pd.Index(['Tumour (Positive)', 'Non-Tumour
(Negative)'], name='Actual Label:'),

                   columns=pd.MultiIndex.from_product([['Decision Tree',
'Regression', 'Random'], ['Tumour', 'Non-Tumour']]), names=['Model:',


```

```
'Predicted:']]
```

```
df
```

## Next steps

- Use a notebook with lakehouse to explore your data

# Explore the data in your lakehouse with a notebook

Article • 05/23/2023

In this tutorial, learn how to explore the data in your lakehouse with a notebook.

## ⓘ Important

Microsoft Fabric is currently in PREVIEW. This information relates to a prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, expressed or implied, with respect to the information provided here.

## Prerequisites

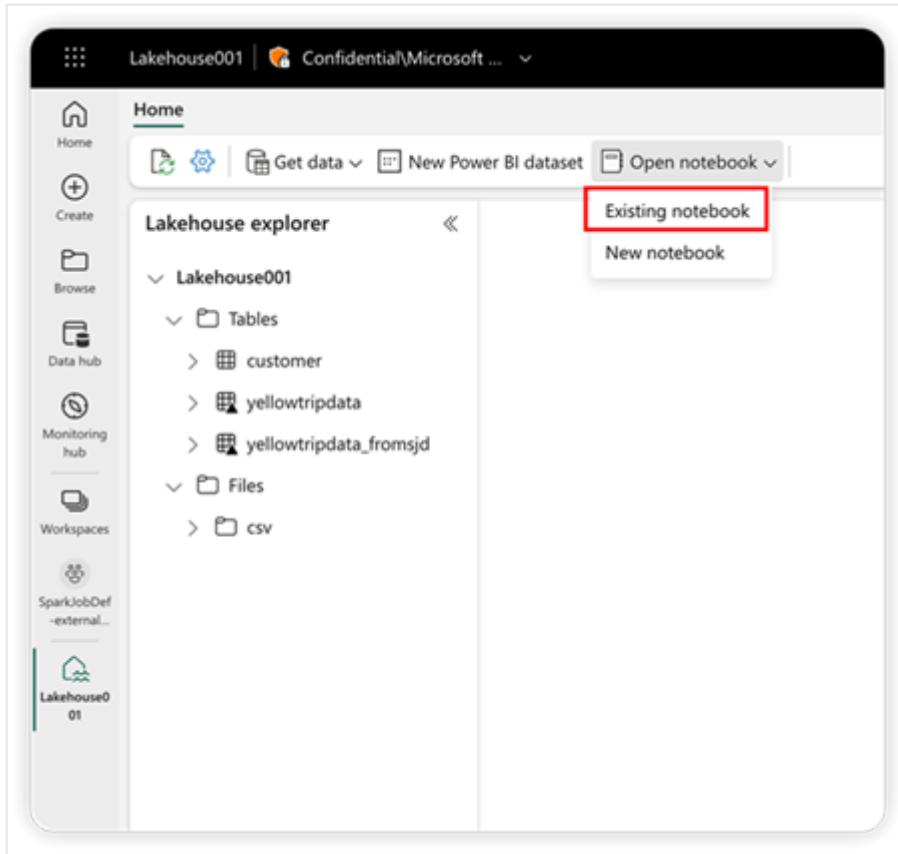
To get started, you need the following prerequisites:

- A Microsoft Fabric tenant account with an active subscription. [Create an account for free](#).
- Read the [Lakehouse overview](#).

## Open or create a notebook from a lakehouse

To explore your lakehouse data, you can add the lakehouse to an existing notebook or create a new notebook from the lakehouse.

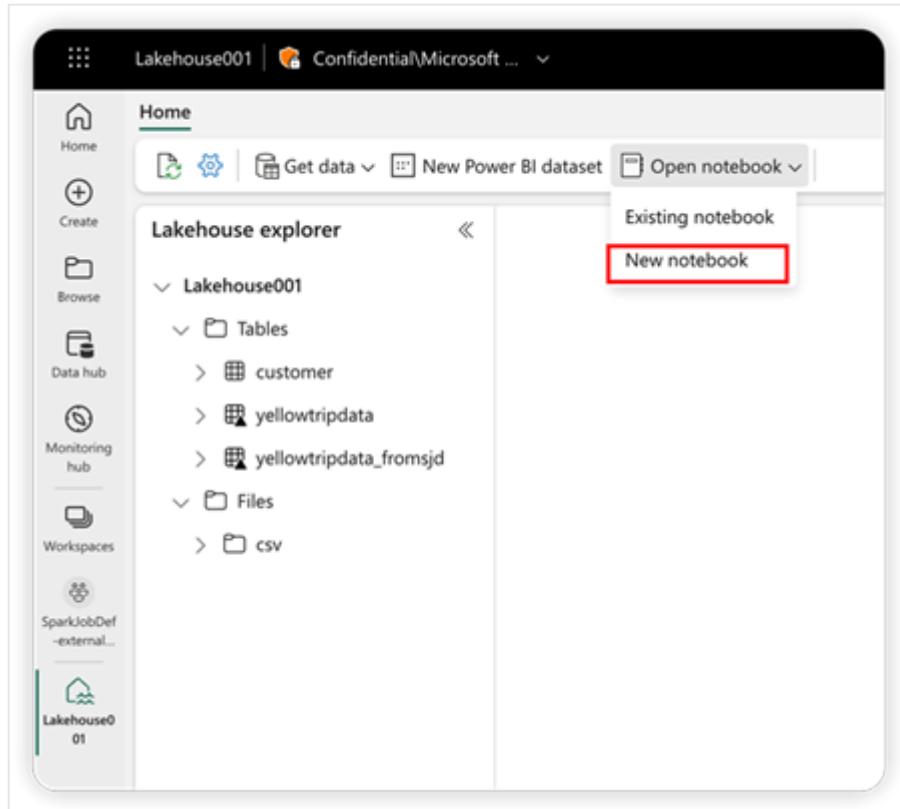
### Open a lakehouse from an existing notebook



Select the notebook from the notebook list and then select **Add**. The notebook opens with your current lakehouse added to the notebook.

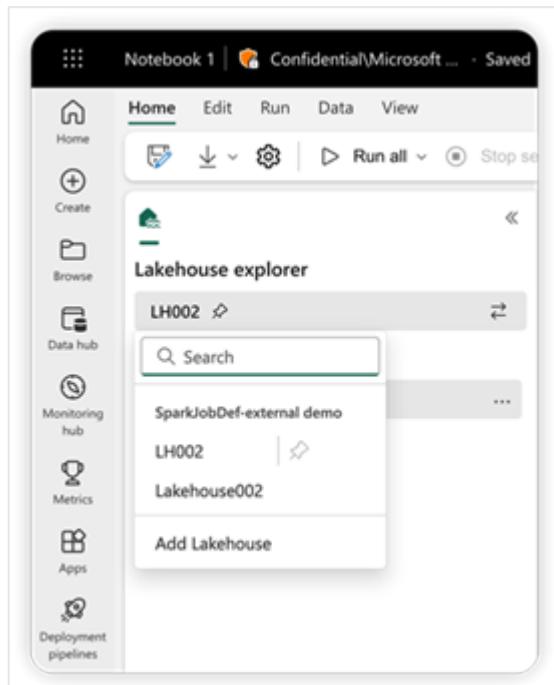
## Open a lakehouse from a new notebook

You can create a new notebook in the same workspace and the current lakehouse appears in that notebook.



## Switch lakehouses and set a default

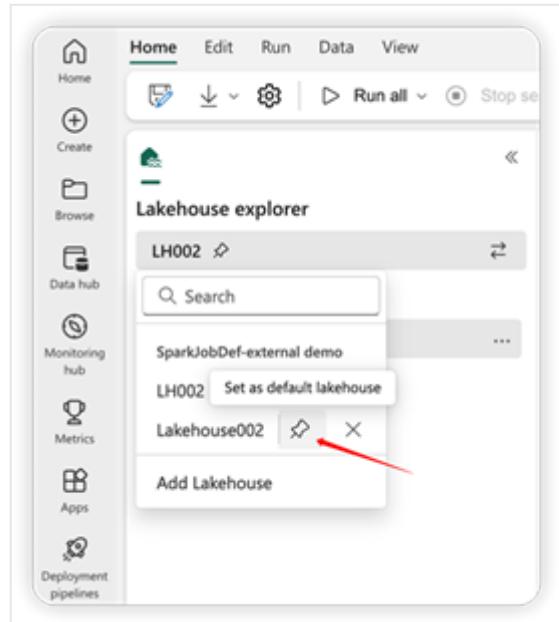
You can add multiple lakehouses to the same notebook. By switching the available lakehouse in the left panel, you can explore the structure and the data from different lakehouses.



In the lakehouse list, the pin icon next to the name of a lakehouse indicates that it's the default lakehouse in your current notebook. In the notebook code, if only a relative path

is provided to access the data from the Microsoft Fabric OneLake, then the default lakehouse is served as the root folder at run time.

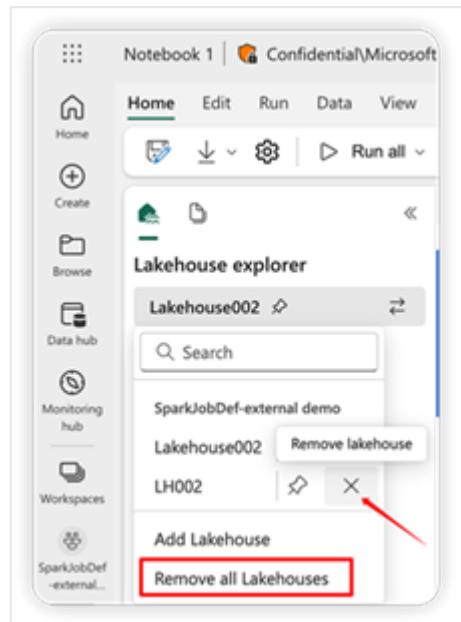
To switch to a different default lakehouse, move the pin icon.



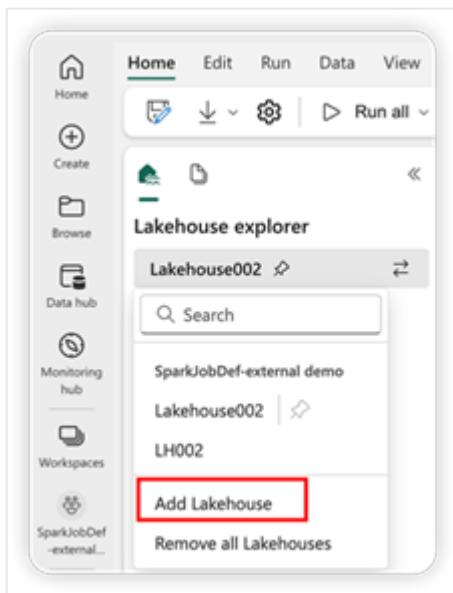
## Add or remove a lakehouse

Selecting the X icon next to a lakehouse name removes it from the notebook, but the lakehouse item still exists in the workspace.

To remove all the lakehouses from the notebook, click "Remove all Lakehouses" in the lakehouse list.

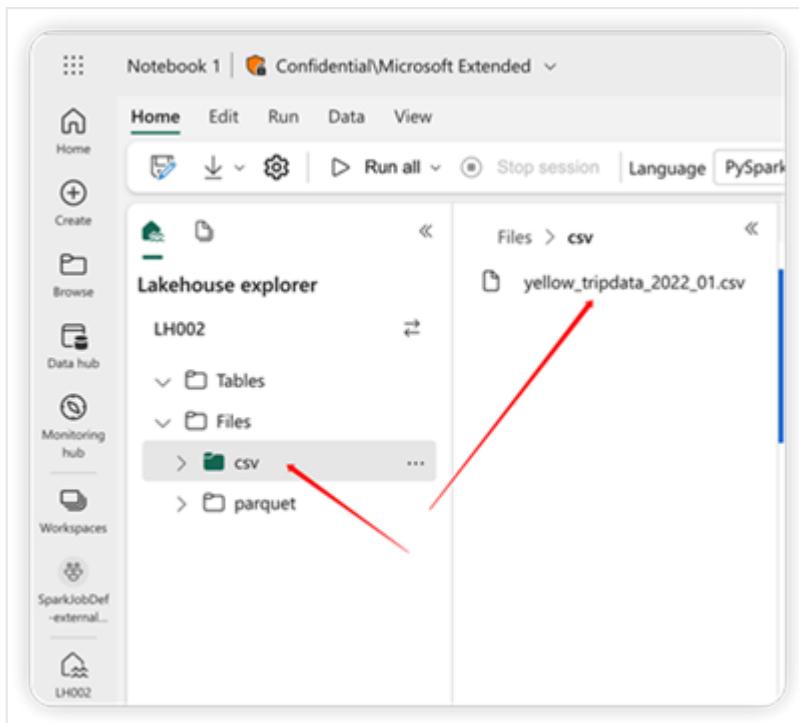


Select **Add lakehouse** to add more lakehouses to the notebook. You can either add an existing one or create a new one.



## Explore the lakehouse data

The structure of the Lakehouse shown in the Notebook is the same as the one in the Lakehouse view. For the detail please check [Lakehouse overview](#). When you select a file or folder, the content area shows the details of the selected item.



### ⓘ Note

The notebook will be created under your current workspace.

## Next steps

- How to use a notebook to load data into your lakehouse

# Use a notebook to load data into your Lakehouse

Article • 05/23/2023

In this tutorial, learn how to read/write data into your lakehouse with a notebook. Spark API and Pandas API are supported to achieve this goal.

## Important

Microsoft Fabric is currently in PREVIEW. This information relates to a prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, expressed or implied, with respect to the information provided here.

## Load data with an Apache Spark API

In the code cell of the notebook, use the following code example to read data from the source and load it into **Files**, **Tables**, or both sections of your lakehouse.

To specify the location to read from, you can use the relative path if the data is from the default lakehouse of current notebook, or you can use the absolute ABFS path if the data is from other lakehouse. you can copy this path from the context menu of the data

The screenshot shows the Azure Data Lake Studio interface. On the left, there's a sidebar with icons for Home, Create, Browse, Data hub, Monitoring hub, Metrics, Apps, Deployment pipelines, and a recent items section. The main area has a top navigation bar with Home, Edit, Run, Data, View, and a language dropdown set to PySpark (Python). Below the navigation is a toolbar with icons for New, Save, Run, Stop session, and Language. The main workspace is divided into two panes. The left pane, titled 'Lakehouse explorer', shows a tree view of a 'Marketing\_LH' lakehouse with 'Tables' and 'Files' sections. The right pane, titled 'Files > Raw', shows a list of files under 'bank-additional-full.parquet'. A context menu is open over one of the files, with options 'Load data', 'Copy ABFS path', 'Copy relative path for Spark', and 'Copy File API path'. The 'Copy relative path for Spark' option is highlighted with a red box.

**Copy ABFS path** : this return the absolute path of the file

**Copy relative path for Spark** : this return the relative path of the file in the default lakehouse

```
Python

df = spark.read.parquet("location to read from")

# Keep it if you want to save dataframe as CSV files to Files section of the
# default Lakehouse

df.write.mode("overwrite").format("csv").save("Files/ " + csv_table_name)

# Keep it if you want to save dataframe as Parquet files to Files section of
# the default Lakehouse

df.write.mode("overwrite").format("parquet").save("Files/" +
parquet_table_name)

# Keep it if you want to save dataframe as a delta lake, parquet table to
# Tables section of the default Lakehouse

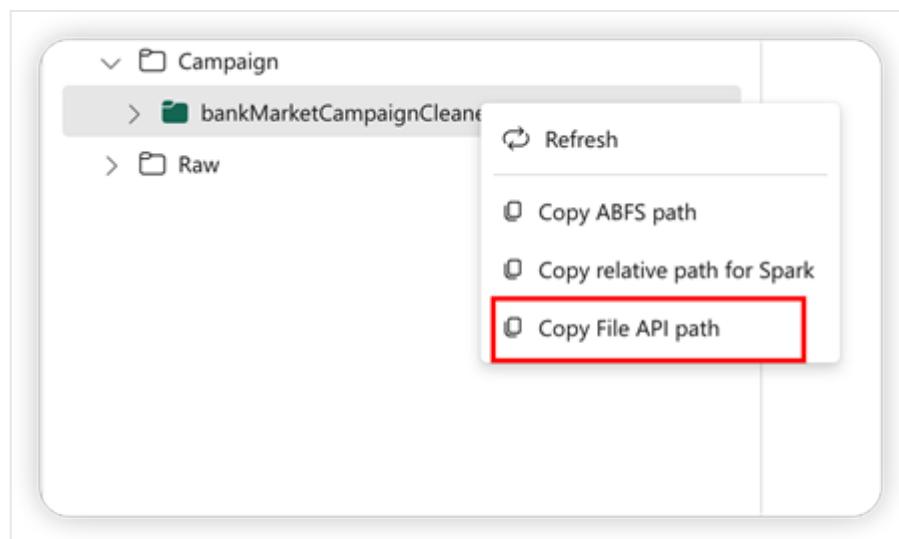
df.write.mode("overwrite").format("delta").saveAsTable(delta_table_name)

# Keep it if you want to save the dataframe as a delta lake, appending the
# data to an existing table
```

```
df.write.mode("append").format("delta").saveAsTable(delta_table_name)
```

## Load data with a Pandas API

To support Pandas API, the default Lakehouse will be automatically mounted to the notebook. The mount point is '/lakehouse/default/'. You can use this mount point to read/write data from/to the default lakehouse. The "Copy File API Path" option from the context menu will return the File API path from that mount point. The path returned from the option **Copy ABFS path** also works for Pandas API.



**Copy File API Path :**This return the path under the mount point of the default lakehouse

Python

```
# Keep it if you want to read parquet file with Pandas from the default
# lakehouse mount point

import pandas as pd
df = pd.read_parquet("/lakehouse/default/Files/sample.parquet")

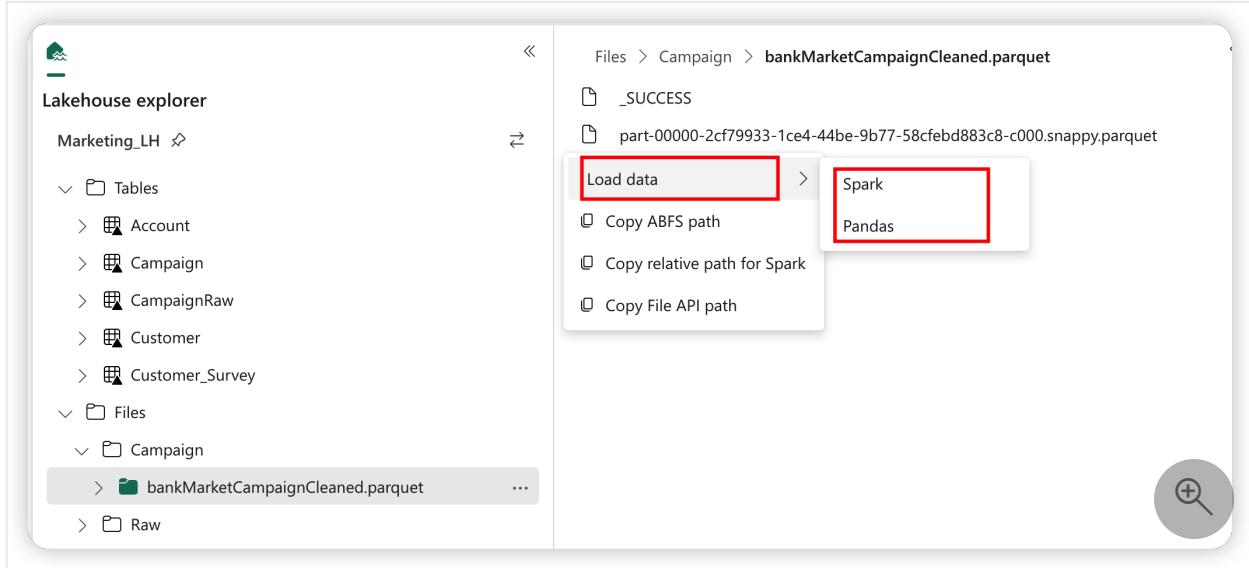
# Keep it if you want to read parquet file with Pandas from the absolute
# abfss path

import pandas as pd
df = pd.read_parquet("abfss://DevExpBuildDemo@msit-
onelake.dfs.fabric.microsoft.com/Marketing_LH.Lakehouse/Files/sample.parquet")
```

Tip

For Spark API, please use the option of **Copy ABFS path** or **Copy relative path for Spark** to get the path of the file. For Pandas API, please use the option of **Copy ABFS path** or **Copy File API path** to get the path of the file.

The quickest way to have the code to work with Spark API or Pandas API is to use the option of **Load data** and select the API you want to use. The code will be automatically generated in a new code cell of the notebook.



## Next steps

- Explore the data in your lakehouse with a notebook

# How-to use end-to-end AI samples in Microsoft Fabric

Article • 05/23/2023

In providing the Synapse Data Science in Microsoft Fabric SaaS experience we want to enable ML professionals to easily and frictionlessly build, deploy and operationalize their machine learning models, in a single analytics platform, while collaborating with other key roles. Begin here to understand the various capabilities the Synapse Data Science experience has to offer and examples of how ML models can address your common business problems.

## ⓘ Important

Microsoft Fabric is currently in PREVIEW. This information relates to a prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, expressed or implied, with respect to the information provided here.

## Recommender

An online bookstore is looking to increase sales by providing customized recommendations. Using customer book rating data in this sample you'll see how to clean, explore the data leading to developing and deploying a recommendation to provide predictions.

Follow along in the [Train a retail recommendation model](#) tutorial.

## Fraud detection

As unauthorized transactions increase, detecting credit card fraud in real time will support financial institutions to provide their customers faster turnaround time on resolution. This end to end sample will include preprocessing, training, model storage and inferencing. The training section will review implementing multiple models and methods that address challenges like imbalanced examples and trade-offs between false positives and false negatives.

Follow along in the [Fraud detection](#) tutorial.

## Forecasting

Using historical New York City Property Sales data and Facebook Prophet in this sample, we'll build a time series model with the trend, seasonality and holiday information to forecast what sales will look like in future cycles.

Follow along in the [Forecasting](#) tutorial.

## Text classification

In this sample, we'll predict whether a book in the British Library is fiction or non-fiction based on book metadata. This will be accomplished by applying text classification with word2vec and linear-regression model on Spark.

Follow along in the [Text classification](#) tutorial.

## Uplift model

In this sample, we'll estimate the causal impact of certain treatments on an individual's behavior by using an Uplift model. We'll walk through step by step how to create, train and evaluate the model touching on four core learnings:

- Data-processing module: extracts features, treatments, and labels.
- Training module: targets to predict the difference between an individual's behavior when there's a treatment and when there's no treatment, using a classical machine learning model like lightGBM.
- Prediction module: calls the uplift model to predict on test data.
- Evaluation module: evaluates the effect of the uplift model on test data.

Follow along in the [Healthcare causal impact of treatments](#) tutorial.

## Next steps

- [How to use Microsoft Fabric notebooks](#)
- [Machine learning model in Microsoft Fabric](#)

# Creating, evaluating, and deploying a recommendation system in Microsoft Fabric

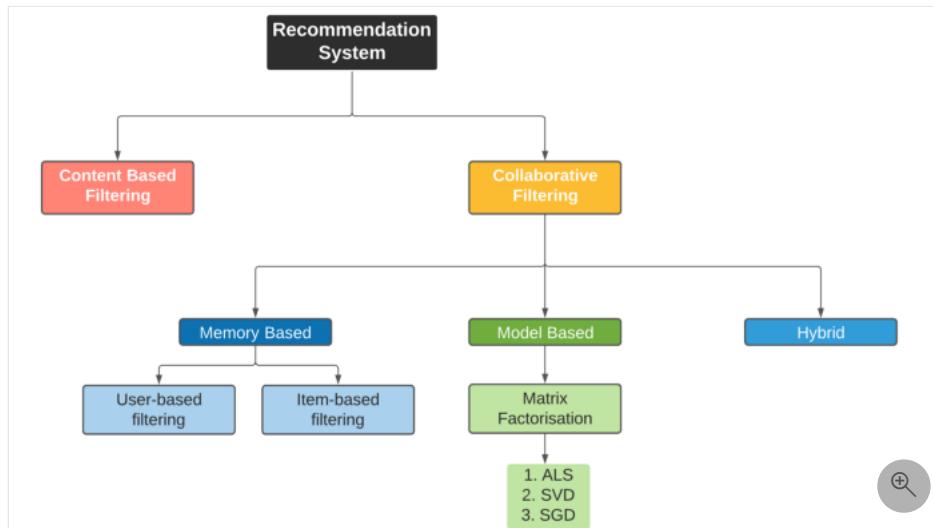
Article • 05/23/2023

In this article, we'll demonstrate data engineering and data science workflow with an e2e sample. The scenario is to build a recommender for online book recommendation.

## ⓘ Important

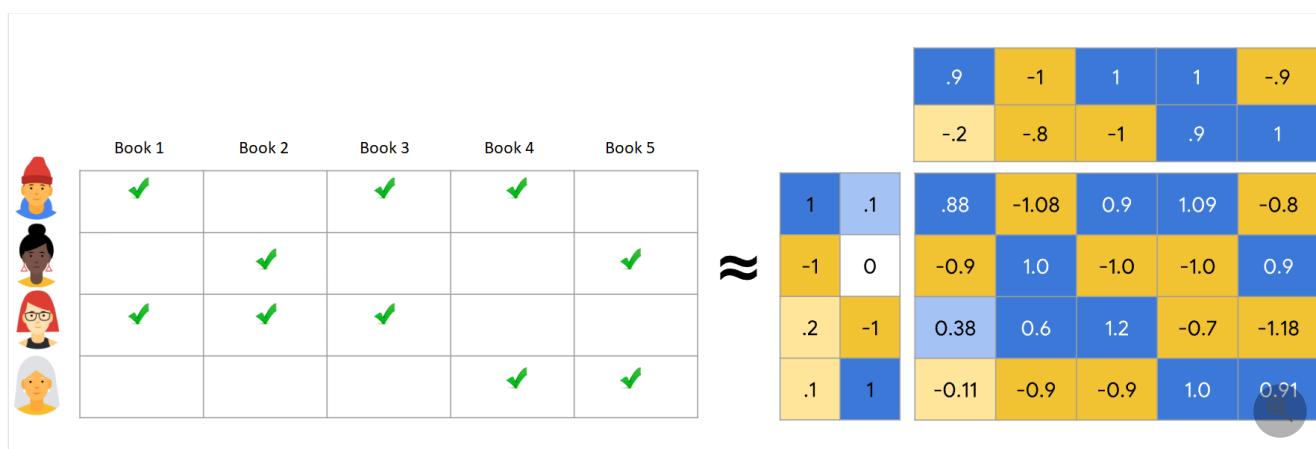
Microsoft Fabric is currently in PREVIEW. This information relates to a prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, expressed or implied, with respect to the information provided here.

There are different types of recommendation algorithms, we'll use a model based collaborative filtering algorithm named Alternating Least Squares (ALS) matrix factorization.



ALS attempts to estimate the ratings matrix  $R$  as the product of two lower-rank matrices,  $X$  and  $Y$ ,  $X * Y^T = R$ . Typically these approximations are called 'factor' matrices.

The general approach is iterative. During each iteration, one of the factor matrices is held constant, while the other is solved for using least squares. The newly solved factor matrix is then held constant while solving for the other factor matrix.



## Prerequisites

- An Azure subscription - [Create one for free ↗](#)
- Create [a new notebook](#) if you want to copy/paste code into cells.
- [Add a Lakehouse to your notebook](#). Attach your notebook to a lakehouse. On the left side, select Add to add an existing lakehouse or create a lakehouse.

## Step 1: Load the data

```
+--- Book-Recommendation-Dataset
|   +--- Books.csv
|   +--- Ratings.csv
|   +--- Users.csv
```

- Books.csv

ISBN	Book-Title	Book-Author	Year-Of-Publication	Publisher	Image-URL-S	Image-URL-M
0195153448	Classical Mythology	Mark P. O. Morford	2002	Oxford University Press	<a href="http://images.amazon.com/images/P/0195153448.01.THUMBZZZ.jpg">http://images.amazon.com/images/P/0195153448.01.THUMBZZZ.jpg</a>	<a href="http://images.amazon.com/images/P/0195153448.01.THUMBZZZ.jpg">http://images.amazon.com/images/P/0195153448.01.THUMBZZZ.jpg</a>
0002005018	Clara Callan	Richard Bruce Wright	2001	HarperFlamingo Canada	<a href="http://images.amazon.com/images/P/0002005018.01.THUMBZZZ.jpg">http://images.amazon.com/images/P/0002005018.01.THUMBZZZ.jpg</a>	<a href="http://images.amazon.com/images/P/0002005018.01.THUMBZZZ.jpg">http://images.amazon.com/images/P/0002005018.01.THUMBZZZ.jpg</a>

- Ratings.csv

User-ID	ISBN	Book-Rating
276725	034545104X	0
276726	0155061224	5

- Users.csv

User-ID	Location	Age
1	"nyc new york usa"	
2	"stockton california usa"	18.0

By defining below parameters, we can apply this notebook on different datasets easily.

Python

```
IS_CUSTOM_DATA = False # if True, dataset has to be uploaded manually

USER_ID_COL = "User-ID" # must not be '_user_id' for this notebook to run successfully
ITEM_ID_COL = "ISBN" # must not be '_item_id' for this notebook to run successfully
ITEM_INFO_COL = (
    "Book-Title" # must not be '_item_info' for this notebook to run successfully
)
RATING_COL = (
    "Book-Rating" # must not be '_rating' for this notebook to run successfully
)
IS_SAMPLE = True # if True, use only <SAMPLE_ROWS> rows of data for training, otherwise use all data
SAMPLE_ROWS = 5000 # if IS_SAMPLE is True, use only this number of rows for training

DATA_FOLDER = "Files/book-recommendation/" # folder containing the dataset
ITEMS_FILE = "Books.csv" # file containing the items information
USERS_FILE = "Users.csv" # file containing the users information
RATINGS_FILE = "Ratings.csv" # file containing the ratings information

EXPERIMENT_NAME = "aisample-recommendation" # mlflow experiment name
```

## Download dataset and upload to Lakehouse

Please add a Lakehouse to the notebook before running it.

Python

```
if not IS_CUSTOM_DATA:
    # Download demo data files into lakehouse if not exist
    import os, requests

    remote_url = "https://synapseaisolutionsa.blob.core.windows.net/public/Book-Recommendation-Dataset"
    file_list = ["Books.csv", "Ratings.csv", "Users.csv"]
    download_path = f"/lakehouse/default/{DATA_FOLDER}/raw"
```

```

if not os.path.exists("/lakehouse/default"):
    raise FileNotFoundError(
        "Default lakehouse not found, please add a lakehouse and restart the session."
    )
os.makedirs(download_path, exist_ok=True)
for fname in file_list:
    if not os.path.exists(f"{download_path}/{fname}"):
        r = requests.get(f"{remote_url}/{fname}", timeout=30)
        with open(f"{download_path}/{fname}", "wb") as f:
            f.write(r.content)
print("Downloaded demo data files into lakehouse.")

```

Python

```

# to record the notebook running time
import time

ts = time.time()

```

## Read data from Lakehouse

Python

```

df_items = (
    spark.read.option("header", True)
    .option("inferSchema", True)
    .csv(f"{DATA_FOLDER}/raw/{ITEMS_FILE}")
    .cache()
)

df_ratings = (
    spark.read.option("header", True)
    .option("inferSchema", True)
    .csv(f"{DATA_FOLDER}/raw/{RATINGS_FILE}")
    .cache()
)

df_users = (
    spark.read.option("header", True)
    .option("inferSchema", True)
    .csv(f"{DATA_FOLDER}/raw/{USERS_FILE}")
    .cache()
)

```

## Step 2. Exploratory data analysis

### Display raw data

We can explore the raw data with `display`, do some basic statistics or even show chart views.

Python

```

import pyspark.sql.functions as F
from pyspark.ml.feature import StringIndexer

```

Python

```
display(df_items, summary=True)
```

Add `_item_id` column for later usage. `_item_id` must be integer for recommendation models. Here we leverage `StringIndexer` to transform `ITEM_ID_COL` to indices.

Python

```

df_items = (
    StringIndexer(inputCol=ITEM_ID_COL, outputCol="_item_id")
    .setHandleInvalid("skip")
    .fit(df_items)
    .transform(df_items)
)

```

```
.withColumn("_item_id", F.col("_item_id").cast("int"))
)
```

Display and check if the `_item_id` increases monotonically and successively as we expected.

Python

```
display(df_items.sort(F.col("_item_id").desc()))
```

Python

```
display(df_users, summary=True)
```

There's a missing value in `User-ID`, we'll drop the row with missing value. It doesn't matter if customized dataset doesn't have missing value.

Python

```
df_users = df_users.dropna(subset=(USER_ID_COL))
```

Python

```
display(df_users, summary=True)
```

Add `_user_id` column for later usage. `_user_id` must be integer for recommendation models. Here we leverage `StringIndexer` to transform `USER_ID_COL` to indices.

In this book dataset, we already have `User-ID` column, which is integer. But we still add `_user_id` column for compatibility to different datasets, making this notebook more robust.

Python

```
df_users = (
    StringIndexer(inputCol=USER_ID_COL, outputCol="_user_id")
    .setHandleInvalid("skip")
    .fit(df_users)
    .transform(df_users)
    .withColumn("_user_id", F.col("_user_id").cast("int"))
)
```

Python

```
display(df_users.sort(F.col("_user_id").desc()))
```

Python

```
display(df_ratings, summary=True)
```

Get the distinct ratings and save them to a list `ratings` for later use.

Python

```
ratings = [i[0] for i in df_ratings.select(RATING_COL).distinct().collect()]
print(ratings)
```

## Merge data

Merge raw dataframes into one dataframe for more comprehensive analysis.

Python

```
df_all = df_ratings.join(df_users, USER_ID_COL, "inner").join(
    df_items, ITEM_ID_COL, "inner"
)
df_all_columns = [
    c for c in df_all.columns if c not in ["_user_id", "_item_id", RATING_COL]
```

```

]

# With this step, we can reorder the columns to make sure _user_id, _item_id and RATING_COL are the first three columns
df_all = (
    df_all.select(["_user_id", "_item_id", RATING_COL] + df_all.columns)
    .withColumn("id", F.monotonically_increasing_id())
    .cache()
)

display(df_all)

```

Python

```

print(f"Total Users: {df_users.select('_user_id').distinct().count()}")
print(f"Total Items: {df_items.select('_item_id').distinct().count()}")
print(f"Total User-Item Interactions: {df_all.count()}")

```

## Compute and plot most popular items

Python

```

# import libs

import pandas as pd  # dataframes
import matplotlib.pyplot as plt  # plotting
import seaborn as sns  # plotting

color = sns.color_palette()  # adjusting plotting style

```

Python

```

# compute top popular products
df_top_items = (
    df_all.groupby(["_item_id"])
    .count()
    .join(df_items, "_item_id", "inner")
    .sort(["count"], ascending=[0])
)

```

Python

```

# find top <topn> popular items
topn = 10
pd_top_items = df_top_items.limit(topn).toPandas()
pd_top_items.head()

```

Top `<topn>` popular items, which can be used for recommendation section "Popular" or "Top purchased".

Python

```

# Plot top <topn> items
f, ax = plt.subplots(figsize=(12, 10))
plt.xticks(rotation="vertical")
sns.barplot(x=ITEM_INFO_COL, y="count", data=pd_top_items)
plt.ylabel("Number of Ratings for the Item")
plt.xlabel("Item Name")
plt.show()

```

## Step 3. Model development and deploy

So far, we've explored the dataset, added unique IDs to our users and items, and plotted top items. Next, we'll train an Alternating Least Squares (ALS) recommender to give users personalized recommendations

### Prepare training and testing data

Python

```

if IS_SAMPLE:
    # Need to sort by '_user_id' before limit, so as to make sure ALS work normally.

```

```

# If train and test dataset have no common _user_id, ALS will fail
df_all = df_all.sort("_user_id").limit(SAMPLE_ROWS)

# cast column into the correct types
df_all = df_all.withColumn(RATING_COL, F.col(RATING_COL).cast("float"))

# By using fraction between 0 to 1, it returns the approximate number of the fraction of the dataset.
# fraction = 0.8 means 80% of the dataset.
# Note that rating = 0 means the user didn't rate the item, so we can't use it for training.
# With below steps, we'll select 80% the dataset with rating > 0 as training dataset.
fractions_train = {0: 0}
fractions_test = {0: 0}
for i in ratings:
    if i == 0:
        continue
    fractions_train[i] = 0.8
    fractions_test[i] = 1
train = df_all.sampleBy(RATING_COL, fractions=fractions_train)

# Join with leftanti means not in, thus below step will select all rows from df_all
# with rating > 0 and not in train dataset, i.e., the left 20% of the dataset as test dataset.
test = df_all.join(train, on="id", how="leftanti").sampleBy(
    RATING_COL, fractions=fractions_test
)

```

Python

```

# Compute the sparsity of the dataset
def get_mat_sparsity(ratings):
    # Count the total number of ratings in the dataset
    count_nonzero = ratings.select(RATING_COL).count()
    print(f"Number of rows: {count_nonzero}")

    # Count the number of distinct user_id and distinct product_id
    total_elements = (
        ratings.select("_user_id").distinct().count()
        * ratings.select("_item_id").distinct().count()
    )

    # Divide the numerator by the denominator
    sparsity = (1.0 - (count_nonzero * 1.0) / total_elements) * 100
    print("The ratings dataframe is ", "%4f" % sparsity + "% sparse.")

get_mat_sparsity(df_all)

```

Python

```

# Check the ID range
# ALS only supports values in Integer range
print(f"max user_id: {df_all.agg({'_user_id': 'max'}).collect()[0][0]}")
print(f"max item_id: {df_all.agg({'_item_id': 'max'}).collect()[0][0]}")

```

## Define the model

With our data in place, we can now define the recommendation model. We'll apply Alternating Least Squares (ALS) model in this notebook.

Spark ML provides a convenient API in building the model. However, the model isn't good enough at handling problems like data sparsity and cold start. We'll combine cross validation and auto hyperparameter tuning to improve the performance of the model.

Python

```

# Specify training parameters
num_epochs = 1
rank_size_list = [64, 128]
reg_param_list = [0.01, 0.1]
model_tuning_method = "TrainValidationSplit" # TrainValidationSplit or CrossValidator

```

Python

```

from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.ml.recommendation import ALS
from pyspark.ml.tuning import ParamGridBuilder, CrossValidator, TrainValidationSplit

# Build the recommendation model using ALS on the training data

```

```

# Note we set cold start strategy to 'drop' to ensure we don't get NaN evaluation metrics
als = ALS(
    maxIter=num_epochs,
    userCol="_user_id",
    itemCol="_item_id",
    ratingCol=RATING_COL,
    coldStartStrategy="drop",
    implicitPrefs=False,
    nonnegative=True,
)

```

## Model training and hyper-tunning

Python

```

# Define tuning parameters
param_grid = (
    ParamGridBuilder()
    .addGrid(als.rank, rank_size_list)
    .addGrid(als.regParam, reg_param_list)
    .build()
)

print("Number of models to be tested: ", len(param_grid))

```

Python

```

# Define evaluator, set rmse as loss
evaluator = RegressionEvaluator(
    metricName="rmse", labelCol=RATING_COL, predictionCol="prediction"
)

```

Python

```

# Build cross validation using CrossValidator and TrainValidationSplit
if model_tuning_method == "CrossValidator":
    tuner = CrossValidator(
        estimator=als, estimatorParamMaps=param_grid, evaluator=evaluator, numFolds=5
    )
elif model_tuning_method == "TrainValidationSplit":
    tuner = TrainValidationSplit(
        estimator=als,
        estimatorParamMaps=param_grid,
        evaluator=evaluator,
        # 80% of the data will be used for training, 20% for validation.
        trainRatio=0.8,
    )
else:
    raise ValueError(f"Unknown model_tuning_method: {model_tuning_method}")

```

Python

```

import numpy as np
import pandas as pd

# Train and Extract best model
models = tuner.fit(train)
model = models.bestModel

if model_tuning_method == "CrossValidator":
    metrics = models.avgMetrics
elif model_tuning_method == "TrainValidationSplit":
    metrics = models.validationMetrics
else:
    raise ValueError(f"Unknown model_tuning_method: {model_tuning_method}")

param_maps = models.getEstimatorParamMaps()
best_params = param_maps[np.argmin(metrics)]
pd_metrics = pd.DataFrame(data={"Metric": metrics})

print("** Best Model **")
for k in best_params:
    print(f"{k.name}: {best_params[k]}")

# Collect metrics
param_strings = []

```

```

for param_map in param_maps:
    # Use split to remove the prefix 'ALS__' in param name
    param_strings.append(
        " ".join(f"{str(k).split('_')[-1]}={v}" for (k, v) in param_map.items())
    )
pd_metrics["Params"] = param_strings

```

Python

```

# Plot metrics of different submodels
f, ax = plt.subplots(figsize=(12, 5))
sns.lineplot(x=pd_metrics["Params"], y=pd_metrics["Metric"])
plt.ylabel("Loss: RMSE")
plt.xlabel("Params")
plt.title("Loss of SubModels")
plt.show()

```

## Model evaluation

We now have the best model, then we can do more evaluations on the test data.

If we trained the model well, it should have high metrics on both train and test datasets.

If we see only good metrics on train, then the model is overfitted, we may need to increase training data size.

If we see bad metrics on both datasets, then the model isn't defined well, we may need to change model architecture or at least fine tune hyper parameters.

Python

```

def evaluate(model, data):
    """
    Evaluate the model by computing rmse, mae, r2 and var over the data.
    """

    predictions = model.transform(data).withColumn(
        "prediction", F.col("prediction").cast("double")
    )

    # Show 10 predictions
    predictions.select("_user_id", "_item_id", RATING_COL, "prediction").limit(
        10
    ).show()

    # Initialize the regression evaluator
    evaluator = RegressionEvaluator(predictionCol="prediction", labelCol=RATING_COL)

    _evaluator = lambda metric: evaluator.setMetricName(metric).evaluate(predictions)
    rmse = _evaluator("rmse")
    mae = _evaluator("mae")
    r2 = _evaluator("r2")
    var = _evaluator("var")

    print(f"RMSE score = {rmse}")
    print(f"MAE score = {mae}")
    print(f"R2 score = {r2}")
    print(f"Explained variance = {var}")

    return predictions, (rmse, mae, r2, var)

```

Evaluation on training data

Python

```
_ = evaluate(model, train)
```

Evaluation on test data.

If R2 is negative, it means the trained model is worse than a horizontal straight line.

Python

```
_ , (rmse, mae, r2, var) = evaluate(model, test)
```

## Log and load model with MLflow

Now we get a good model, we can save it for later use. Here we use MLflow to log metrics/models, and load models back for prediction.

Python

```
# Setup mlflow
import mlflow

mlflow.set_experiment(EXPERIMENT_NAME)
```

Python

```
# Log model, metrics and params
with mlflow.start_run() as run:
    print("log model:")
    mlflow.spark.log_model(
        model,
        f"{EXPERIMENT_NAME}-alsmodel",
        registered_model_name=f"{EXPERIMENT_NAME}-alsmodel",
        dfs_tmpdir="Files/spark",
    )

    print("log metrics:")
    mlflow.log_metrics({"RMSE": rmse, "MAE": mae, "R2": r2, "Explained variance": var})

    print("log parameters:")
    mlflow.log_params(
        {
            "num_epochs": num_epochs,
            "rank_size_list": rank_size_list,
            "reg_param_list": reg_param_list,
            "model_tuning_method": model_tuning_method,
            "DATA_FOLDER": DATA_FOLDER,
        }
    )

model_uri = f"runs:{run.info.run_id}/{EXPERIMENT_NAME}-alsmodel"
print("Model saved in run %s" % run.info.run_id)
print(f"Model URI: {model_uri}")
```

Python

```
# Load model back
# mlflow will use PipelineModel to wrapper the original model, thus here we extract the original ALSModel from the stages.
loaded_model = mlflow.spark.load_model(model_uri, dfs_tmpdir="Files/spark").stages[-1]
```

## Step 4. Save prediction results

### Model deploy and prediction

Offline recommendation: Recommend 10 items for each user.

Save offline recommendation results:

Python

```
# Generate top 10 product recommendations for each user
userRecs = loaded_model.recommendForAllUsers(10)
```

Python

```
# Convert recommendations into interpretable format
userRecs = (
    userRecs.withColumn("rec_exp", F.explode("recommendations"))
    .select("_user_id", F.col("rec_exp._item_id"), F.col("rec_exp.rating"))
    .join(df_items.select(["_item_id", "Book-Title"]), on="_item_id")
)
userRecs.limit(10).show()
```

Python

```
# Code for saving userRecs into lakehouse
userRecs.write.format("delta").mode("overwrite").save(
    f"{DATA_FOLDER}/predictions/userRecs"
)
```

Python

```
print(f"Full run cost {int(time.time() - ts)} seconds.")
```

## Next steps

- Training and evaluating a text classification model
- Machine learning model in Microsoft Fabric
- Train machine learning models
- Machine learning experiments in Microsoft Fabric

# Create, evaluate, and deploy a fraud detection model in Microsoft Fabric

Article • 05/23/2023

In this tutorial, we'll demonstrate data engineering and data science workflows with an end-to-end example that builds a model for detecting fraudulent credit card transactions. The steps you'll take are:

- ✓ Upload the data into a Lakehouse
- ✓ Perform exploratory data analysis on the data
- ✓ Prepare the data by handling class imbalance
- ✓ Train a model and log it with MLflow
- ✓ Deploy the model and save prediction results

## ⓘ Important

Microsoft Fabric is currently in PREVIEW. This information relates to a prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, expressed or implied, with respect to the information provided here.

## Prerequisites

- A Power BI Premium subscription. If you don't have one, see [How to purchase Power BI Premium](#).
- A Power BI workspace with assigned Premium capacity. If you don't have a workspace, use the steps in [Create a workspace](#) to create one and assign it to a Premium capacity.
- Sign in to [Microsoft Fabric](#).
- Go to the Data Science experience in Microsoft Fabric.
- Open the sample notebook or create a new notebook.
  - [Create a new notebook](#) if you want to copy/paste code into cells.
  - Or, Select **Use a sample > Fraud detection** to open the sample notebook.
- [Add a Lakehouse to your notebook](#).

## Step 1: Load the data

The dataset contains credit card transactions made by European cardholders in September 2013 over the course of two days. Out of 284,807 transactions, 492 are fraudulent. The positive class (fraud) accounts for a mere 0.172% of the data, thereby making the dataset highly unbalanced.

## Input and response variables

The dataset contains only numerical input variables, which are the result of a Principal Component Analysis (PCA) transformation on the original features. To protect confidentiality, we can't provide the original features or more background information about the data. The only features that haven't been transformed with PCA are "Time" and "Amount".

- Features "V1, V2, ... V28" are the principal components obtained with PCA.
- "Time" contains the seconds elapsed between each transaction and the first transaction in the dataset.
- "Amount" is the transaction amount. This feature can be used for example-dependent cost-sensitive learning.
- "Class" is the response variable, and it takes the value `1` for fraud and `0` otherwise.

Given the class imbalance ratio, we recommend measuring the accuracy using the Area Under the Precision-Recall Curve (AUPRC). Using a confusion matrix to evaluate accuracy isn't meaningful for unbalanced classification.

The following snippet shows a portion of the *creditcard.csv* data.

"Time"	"V1"	"V2"	"V3"	"V4"	"V5"	"V6"	"V7"
0	-1.3598071336738	-0.0727811733098497	2.53634673796914	1.37815522427443	-0.338320769942518	0.46238777762292	0.239598554061
0	1.19185711131486	0.26615071205963	0.16648011335321	0.448154078460911	0.0600176492822243	-0.0823608088155687	-0.07880298333

## Install libraries

For this tutorial, we need to install the `imblearn` library. The PySpark kernel will be restarted after running `%pip install`, thus we need to install the library before we run any other cells.

```
shell

# install imblearn for SMOTE
%pip install imblearn
```

By defining the following parameters, we can apply the notebook on different datasets easily.

```
Python

IS_CUSTOM_DATA = False # if True, dataset has to be uploaded manually

TARGET_COL = "Class" # target column name
IS_SAMPLE = False # if True, use only <SAMPLE_ROWS> rows of data for training, otherwise use all data
SAMPLE_ROWS = 5000 # if IS_SAMPLE is True, use only this number of rows for training

DATA_FOLDER = "Files/fraud-detection/" # folder with data files
DATA_FILE = "creditcard.csv" # data file name

EXPERIMENT_NAME = "aisample-fraud" # mlflow experiment name
```

## Download the dataset and upload to a Lakehouse

Before running the notebook, you must add a Lakehouse to it. The Lakehouse is used to store the data for this example. To add a Lakehouse, see [Add a Lakehouse to your notebook](#).

```
Python

if not IS_CUSTOM_DATA:
    # Download demo data files into lakehouse if not exist
    import os, requests

    remote_url = "https://synapseaisolutionsa.blob.core.windows.net/public/Credit_Card_Fraud_Detection"
    fname = "creditcard.csv"
    download_path = f"/lakehouse/default/{DATA_FOLDER}/raw"

    if not os.path.exists("/lakehouse/default"):
        raise FileNotFoundError(
            "Default lakehouse not found, please add a lakehouse and restart the session."
        )
    os.makedirs(download_path, exist_ok=True)
    if not os.path.exists(f"{download_path}/{fname}"):
        r = requests.get(f"{remote_url}/{fname}", timeout=30)
        with open(f"{download_path}/{fname}", "wb") as f:
            f.write(r.content)
    print("Downloaded demo data files into lakehouse.")
```

```
Python

# to record the notebook running time
import time

ts = time.time()
```

## Read data from the Lakehouse

```
Python

df = (
    spark.read.format("csv")
    .option("header", "true")
    .option("inferSchema", True)
    .load(f"{DATA_FOLDER}/raw/{DATA_FILE}")
    .cache()
)
```

## Step 2. Perform exploratory data analysis

In this section, we'll explore the data, check its schema, reorder its columns, and cast the columns into the correct data types.

### Display raw data

We can use `display` to explore the raw data, calculate some basic statistics, or even show chart views.

```
Python
```

```
display(df)
```

Print some information about the data, such as the schema.

```
Python
```

```
# print dataset basic info
print("records read: " + str(df.count()))
print("Schema: ")
df.printSchema()
```

### Cast columns into the correct types

```
Python
```

```
import pyspark.sql.functions as F

df_columns = df.columns
df_columns.remove(TARGET_COL)

# to make sure the TARGET_COL is the last column
df = df.select(df_columns + [TARGET_COL]).withColumn(
    TARGET_COL, F.col(TARGET_COL).cast("int")
)

if IS_SAMPLE:
    df = df.limit(SAMPLE_ROWS)
```

## Step 3. Develop and deploy a model

In this section, we'll train a LightGBM model to classify fraudulent transactions.

### Prepare training and testing data

Begin by splitting the data into training and testing sets.

```
Python
```

```
# Split the dataset into train and test
train, test = df.randomSplit([0.85, 0.15], seed=42)
```

```
Python
```

```
# Merge Columns
from pyspark.ml.feature import VectorAssembler

feature_cols = df.columns[:-1]
featurizer = VectorAssembler(inputCols=feature_cols, outputCol="features")
train_data = featurizer.transform(train)[TARGET_COL, "features"]
test_data = featurizer.transform(test)[TARGET_COL, "features"]
```

Check the data volume and imbalance in the training set.

```
Python
```

```
display(train_data.groupBy(TARGET_COL).count())
```

## Handle imbalanced data

As often happens with real-world data, this data has a class-imbalance problem, since the positive class (fraudulent transactions) accounts for only 0.172% of all transactions. We'll apply [SMOTE](#) (Synthetic Minority Over-sampling Technique) to automatically handle class imbalance in the data. The SMOTE method oversamples the minority class and undersamples the majority class for improved classifier performance.

Let's apply SMOTE to the training data:

### ⓘ Note

`imblearn` only works for pandas DataFrames, not PySpark DataFrames.

#### Python

```
from pyspark.ml.functions import vector_to_array, array_to_vector
import numpy as np
from collections import Counter
from imblearn.over_sampling import SMOTE

train_data_array = train_data.withColumn("features", vector_to_array("features"))

train_data_pd = train_data_array.toPandas()

X = train_data_pd["features"].to_numpy()
y = train_data_pd[TARGET_COL].to_numpy()
print("Original dataset shape %s" % Counter(y))

X = np.array([np.array(x) for x in X])

sm = SMOTE(random_state=42)
X_res, y_res = sm.fit_resample(X, y)
print("Resampled dataset shape %s" % Counter(y_res))

new_train_data = tuple(zip(X_res.tolist(), y_res.tolist()))
dataColumns = ["features", TARGET_COL]
new_train_data = spark.createDataFrame(data=new_train_data, schema=dataColumns)
new_train_data = new_train_data.withColumn("features", array_to_vector("features"))
```

## Define the model

With our data in place, we can now define the model. We'll use a LightGBM classifier and use SynapseML to implement the model with a few lines of code.

#### Python

```
from synapse.ml.lightgbm import LightGBMClassifier

model = LightGBMClassifier(
    objective="binary", featuresCol="features", labelCol=TARGET_COL, isUnbalance=True
)
smote_model = LightGBMClassifier(
    objective="binary", featuresCol="features", labelCol=TARGET_COL, isUnbalance=False
)
```

## Train the model

#### Python

```
model = model.fit(train_data)
smote_model = smote_model.fit(new_train_data)
```

## Explain the model

Here we can show the importance that the model assigns to each feature in the training data.

#### Python

```

import pandas as pd
import matplotlib.pyplot as plt

feature_importances = model.getFeatureImportances()
fi = pd.Series(feature_importances, index=feature_cols)
fi = fi.sort_values(ascending=True)
f_index = fi.index
f_values = fi.values

# print feature importances
print("f_index:", f_index)
print("f_values:", f_values)

# plot
x_index = list(range(len(fi)))
x_index = [x / len(fi) for x in x_index]
plt.rcParams["figure.figsize"] = (20, 20)
plt.barh(
    x_index, f_values, height=0.028, align="center", color="tan", tick_label=f_index
)
plt.xlabel("importances")
plt.ylabel("features")
plt.show()

```

## Evaluate the model

Generate model predictions:

```

Python

predictions = model.transform(test_data)
predictions.limit(10).toPandas()

```

Display model metrics:

```

Python

from synapse.ml.train import ComputeModelStatistics

metrics = ComputeModelStatistics(
    evaluationMetric="classification", labelCol=TARGET_COL, scoredLabelsCol="prediction"
).transform(predictions)
display(metrics)

```

Create a confusion matrix:

```

Python

# collect confusion matrix value
cm = metrics.select("confusion_matrix").collect()[0][0].toArray()
print(cm)

```

Plot the confusion matrix:

```

Python

# plot confusion matrix
import seaborn as sns

sns.set(rc={"figure.figsize": (6, 4.5)})
ax = sns.heatmap(cm, annot=True, fmt=".20g")
ax.set_title("Confusion Matrix")
ax.set_xlabel("Predicted label")
ax.set_ylabel("True label")

```

Define a function to evaluate the model:

```

Python

from pyspark.ml.evaluation import BinaryClassificationEvaluator

```

```

def evaluate(predictions):
    """
    Evaluate the model by computing AUROC and AUPRC with the predictions.
    """

    # initialize the binary evaluator
    evaluator = BinaryClassificationEvaluator(
        rawPredictionCol="prediction", labelCol=TARGET_COL
    )

    _evaluator = lambda metric: evaluator.setMetricName(metric).evaluate(predictions)

    # calculate AUROC, baseline 0.5
    auroc = _evaluator("areaUnderROC")
    print(f"AUROC: {auroc:.4f}")

    # calculate AUPRC, baseline positive rate (0.172% in the demo data)
    auprc = _evaluator("areaUnderPR")
    print(f"AUPRC: {auprc:.4f}")

    return auroc, auprc

```

Evaluate the original model:

Python

```

# evaluate the original model
auroc, auprc = evaluate(predictions)

```

Evaluate the SMOTE model:

Python

```

# evaluate the SMOTE model
new_predictions = smote_model.transform(test_data)
new_auroc, new_auprc = evaluate(new_predictions)

```

Python

```

if new_auprc > auprc:
    # Using model trained on SMOTE data if it has higher AUPRC
    model = smote_model
    auprc = new_auprc
    auroc = new_auroc

```

## Log and load the model with MLflow

Now that we have a decent working model, we can save it for later use. Here we use MLflow to log metrics and models, and load the models back for prediction.

Set up MLflow:

Python

```

# setup mlflow
import mlflow

mlflow.set_experiment(EXPERIMENT_NAME)

```

Log model, metrics, and parameters:

Python

```

# log model, metrics and params
with mlflow.start_run() as run:
    print("log model:")
    mlflow.spark.log_model(
        model,
        f"{EXPERIMENT_NAME}-lightgbm",
        registered_model_name=f"{EXPERIMENT_NAME}-lightgbm",
        dfs_tmpdir="Files/spark",
    )

```

```
print("log metrics:")
mlflow.log_metrics({"AUPRC": auprc, "AUROC": auroc})

print("log parameters:")
mlflow.log_params({"DATA_FILE": DATA_FILE})

model_uri = f"runs:{run.info.run_id}/{EXPERIMENT_NAME}-lightgbm"
print("Model saved in run %s" % run.info.run_id)
print(f"Model URI: {model_uri}")
```

Reload the model:

Python

```
# load model back
loaded_model = mlflow.spark.load_model(model_uri, dfs_tmpdir="Files/spark")
```

## Step 4. Save prediction results

In this section, we'll deploy the model and save the prediction results.

### Model deploy and prediction

Python

```
batch_predictions = loaded_model.transform(test_data)
```

Save predictions into the Lakehouse:

Python

```
# code for saving predictions into lakehouse
batch_predictions.write.format("delta").mode("overwrite").save(
    f"{DATA_FOLDER}/predictions/batch_predictions"
)
```

Python

```
print(f"Full run cost {int(time.time() - ts)} seconds.")
```

## Next Steps

- [How to use Microsoft Fabric notebooks](#)
- [Machine learning model in Microsoft Fabric](#)
- [Train machine learning models](#)
- [Machine learning experiments in Microsoft Fabric](#)

# Training and evaluating a time series forecasting model in Microsoft Fabric

Article • 05/23/2023

In this notebook, we'll develop a program to forecast time series data that has seasonal cycles. We'll use the [NYC Property Sales dataset](#) with dates ranging from 2003 to 2015 published by NYC Department of Finance on the [NYC Open Data Portal](#).

## ⓘ Important

Microsoft Fabric is currently in PREVIEW. This information relates to a prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, expressed or implied, with respect to the information provided here.

The dataset is a record of every building sold in New York City property market during 13-year period. Refer to [Glossary of Terms for Property Sales Files](#) for definition of columns in the spreadsheet. The dataset looks like the following table:

borough	neighborhood	building_class_category	tax_class	block	lot	easement	building_class_at_present	address	apartment_number	zip
Manhattan	ALPHABET CITY	07 RENTALS - WALKUP APARTMENTS	0.0	384.0	17.0		C4	225 EAST 2ND STREET		1000000000000000000
Manhattan	ALPHABET CITY	07 RENTALS - WALKUP APARTMENTS	2.0	405.0	12.0		C7	508 EAST 12TH STREET		1000000000000000000

We'll build up a model to forecast monthly volume of property trade based on history data. In order to forecast, we'll use [Facebook Prophet](#), which provides fast and automated forecast procedure and handles seasonality well.

## Install Prophet

Let's first install [Facebook Prophet](#). Facebook Prophet (Prophet) is an open source time-series forecasting library developed by Facebook. It uses a decomposable time series model that consists of three main components: trend, seasonality, and holidays.

For the trend component, Prophet assumes piece-wise constant rate of growth with automatic change point selection.

For seasonality component, Prophet models weekly and yearly seasonality using Fourier Series. Since we're using monthly data, so we won't have weekly seasonality and won't be considering holidays.

```
shell
!pip install prophet
```

## Step 1: Load the data

### Download dataset and upload to a Data Lakehouse

A Data Lakehouse (lakehouse) is a data architecture that provides a central repository for your data. There are 15 csv files containing property sales records from five boroughs in New York since 2003 to 2015. For your convenience, these files are compressed in `nyc_property_sales.tar` and are available in a public blob storage.

```
Python
```

```
URL = "https://synapseaisolutionsa.blob.core.windows.net/public/NYC_Property_Sales_Dataset/"
TAR_FILE_NAME = "nyc_property_sales.tar"
DATA_FOLDER = "Files/NYC_Property_Sales_Dataset"
TAR_FILE_PATH = f"/lakehouse/default/{DATA_FOLDER}/tar/"
CSV_FILE_PATH = f"/lakehouse/default/{DATA_FOLDER}/csv/"
```

```
Python
```

```

import os

if not os.path.exists("/lakehouse/default"):
    # ask user to add a lakehouse if no default lakehouse added to the notebook.
    # a new notebook will not link to any lakehouse by default.
    raise FileNotFoundError(
        "Default lakehouse not found, please add a lakehouse for the notebook."
    )
else:
    # check if the needed files are already in the lakehouse, try to download and unzip if not.
    if not os.path.exists(f"{TAR_FILE_PATH}{TAR_FILE_NAME}"):
        os.makedirs(TAR_FILE_PATH, exist_ok=True)
        os.system(f"wget {URL}{TAR_FILE_NAME} -O {TAR_FILE_PATH}{TAR_FILE_NAME}")

    os.makedirs(CSV_FILE_PATH, exist_ok=True)
    os.system(f"tar -zxvf {TAR_FILE_PATH}{TAR_FILE_NAME} -C {CSV_FILE_PATH}")

```

## Create dataframe from Lakehouse

The `display` function print the dataframe and automatically gives chart views.

```

Python

df = (
    spark.read.format("csv")
    .option("header", "true")
    .load("Files/NYC_Property_Sales_Dataset/csv")
)
display(df)

```

## Step 2: Data preprocessing

### Type conversion and filtering

Lets do some necessary type conversion and filtering.

- Need convert sale prices to integers.
- Need exclude irregular sales data. For example, a \$0 sale indicates ownership transfer without cash consideration.
- Exclude building types other than A class.

The reason to choose only market of A class building for analysis is that seasonal effect is ineligible coefficient for A class building. The model we are using outperforms many others in including seasonality, which is common needs in time series analysis.

```

Python

# import libs
import pyspark.sql.functions as F
from pyspark.sql.types import *

```

```

Python

df = df.withColumn(
    "sale_price", F.regexp_replace("sale_price", "[\$,]", "").cast(IntegerType())
)
df = df.select("*").where(
    'sale_price > 0 and total_units > 0 and gross_square_feet > 0 and building_class_at_time_of_sale like "A%"'
)

```

```

Python

monthly_sale_df = df.select(
    "sale_price",
    "total_units",
    "gross_square_feet",
    F.date_format("sale_date", "yyyy-MM").alias("month"),
)

```

```

Python

```

```
display(df)
```

Python

```
summary_df = (
    monthly_sale_df.groupBy("month")
    .agg(
        F.sum("sale_price").alias("total_sales"),
        F.sum("total_units").alias("units"),
        F.sum("gross_square_feet").alias("square_feet"),
    )
    .orderBy("month")
)
```

Python

```
display(summary_df)
```

## Visualization

Now take a look at the trend of property trade trend at NYC. The yearly seasonality is clear on the chosen building class. The peak buying seasons are usually spring and fall.

Python

```
df_pandas = summary_df.toPandas()
```

Python

```
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

f, (ax1, ax2) = plt.subplots(2, 1, figsize=(35, 10))
plt.sca(ax1)
plt.xticks(np.arange(0, 15 * 12, step=12))
plt.ticklabel_format(style="plain", axis="y")
sns.lineplot(x="month", y="total_sales", data=df_pandas)
plt.ylabel("Total Sales")
plt.xlabel("Time")
plt.title("Total Property Sales by Month")

plt.sca(ax2)
plt.xticks(np.arange(0, 15 * 12, step=12))
plt.ticklabel_format(style="plain", axis="y")
sns.lineplot(x="month", y="square_feet", data=df_pandas)
plt.ylabel("Total Square Feet")
plt.xlabel("Time")
plt.title("Total Property Square Feet Sold by Month")
plt.show()
```

## Step 3: Model training and evaluation

### Model fitting

To do model fitting, rename the time axis to 'ds' and value axis to 'y'.

Python

```
import pandas as pd

df_pandas["ds"] = pd.to_datetime(df_pandas["month"])
df_pandas["y"] = df_pandas["total_sales"]
```

Now let's fit the model. We'll choose to use 'multiplicative' seasonality, it means seasonality is no longer a constant additive factor like default assumed by Prophet. As shown in a previous cell, we printed the total property sale data per month, and the vibration amplitude isn't consistent. It means using simple additive seasonality won't fit the data well. In addition, we'll use Markov Chain Monte Carlo (MCMC)

that gives mean of posteriori distribution. By default, Prophet uses Stan's L-BFGS to fit the model, which finds a maximum a posteriori probability(MAP) estimate.

Python

```
from prophet import Prophet
from prophet.plot import add_changepoints_to_plot

m = Prophet(
    seasonality_mode="multiplicative", weekly_seasonality=False, mcmc_samples=1000
)
m.fit(df_pandas)
```

Let's use built-in functions in Prophet to show the model fitting results. The black dots are data points used to train the model. The blue line is the prediction and the light blue area shows uncertainty intervals.

Python

```
future = m.make_future_dataframe(periods=12, freq="M")
forecast = m.predict(future)
fig = m.plot(forecast)
```

Prophet assumes piece-wise constant growth, thus you can plot the change points of the trained model.

Python

```
fig = m.plot(forecast)
a = add_changepoints_to_plot(fig.gca(), m, forecast)
```

Visualize trend and yearly seasonality. The light blue area reflects uncertainty.

Python

```
fig2 = m.plot_components(forecast)
```

## Cross validation

We can use Prophet's built-in cross validation functionality to measure the forecast error on historical data. The following parameters mean we should start with 11 years of training data, then make predictions every 30 days within a one year horizon.

Python

```
from prophet.diagnostics import cross_validation
from prophet.diagnostics import performance_metrics

df_cv = cross_validation(m, initial="11 Y", period="30 days", horizon="365 days")
df_p = performance_metrics(df_cv, monthly=True)
```

Python

```
display(df_p)
```

## Step 4: Log and load model with MLflow

We can now store the trained model for later use.

Python

```
# setup mlflow
import mlflow

EXPERIMENT_NAME = "aisample-timeseries"
mlflow.set_experiment(EXPERIMENT_NAME)
```

Python

```
# log the model and parameters
model_name = f"{EXPERIMENT_NAME}-prophet"
with mlflow.start_run() as run:
    mlflow.prophet.log_model(m, model_name, registered_model_name=model_name)
    mlflow.log_params({"seasonality_mode": "multiplicative", "mcmc_samples": 1000})
    model_uri = f"runs:{run.info.run_id}/{model_name}"
    print("Model saved in run %s" % run.info.run_id)
    print(f"Model URI: {model_uri}")
```

Python

```
# load the model back
loaded_model = mlflow.prophet.load_model(model_uri)
```

# Training and evaluating a text classification model in Microsoft Fabric

Article • 05/23/2023

In this notebook, we demonstrate how to solve a text classification task with word2vec + linear-regression model on Spark.

## ⓘ Important

Microsoft Fabric is currently in PREVIEW. This information relates to a prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, expressed or implied, with respect to the information provided here.

The sample dataset consists of metadata relating to books digitized by the British Library in partnership with Microsoft. It includes human generated labels for classifying a book as 'fiction' or 'non-fiction'. We use this dataset to train a model for genre classification that predicts whether a book is 'fiction' or 'non-fiction' based on its title.

BL record ID	Type of resource	Name	Dates associated with name	Type of name	Role	All names	Title	Variant titles	Series title	Number within series	Country of publication	Place of publication	Publisher
014602826	Monograph	Yearsley, Ann	1753-1806	person		More, Hannah, 1745-1833 [person]; Yearsley, Ann, 1753-1806 [person]	Poems on several occasions [With a prefatory letter by Hannah More.]				England	London	
014602830	Monograph	A, T.		person		Oldham, John, 1653-1683 [person]; A, T. [person]	A Satyr against Vertue. (A poem: supposed to be spoken by a Town-Hector [By John Oldham. The preface signed: T. A.])				England	London	

## Step 1: Load the data

### Notebook configurations

#### Install libraries

In this notebook, we `wordcloud`, which first needs to be installed. The PySpark kernel will be restarted after `%pip install`, thus we need to install it before we run any other cells. Word Cloud is a package allowing us to use a data visualization technique for representing text data to indicate its frequency in a given piece of text.

```
shell  
  
# install wordcloud for text visualization  
%pip install wordcloud
```

By defining the following parameters, we can apply this notebook on different datasets easily.

Python

```
IS_CUSTOM_DATA = False # if True, dataset has to be uploaded manually by user
DATA_FOLDER = "Files/title-genre-classification"
DATA_FILE = "blbooksgenre.csv"

# data schema
TEXT_COL = "Title"
LABEL_COL = "annotator_genre"
LABELS = ["Fiction", "Non-fiction"]

EXPERIMENT_NAME = "aisample-textclassification" # mlflow experiment name
```

We also define some hyper-parameters for model training. (DON'T modify these parameters unless you are aware of the meaning).

Python

```
# hyper-params
word2vec_size = 128
min_word_count = 3
max_iter = 10
k_folds = 3
```

## Import dependencies

Python

```
import numpy as np
from itertools import chain

from wordcloud import WordCloud
import matplotlib.pyplot as plt
import seaborn as sns

import pyspark.sql.functions as F

from pyspark.ml import Pipeline
from pyspark.ml.feature import *
from pyspark.ml.tuning import CrossValidator, ParamGridBuilder
from pyspark.ml.classification import LogisticRegression
from pyspark.ml.evaluation import (
    BinaryClassificationEvaluator,
    MulticlassClassificationEvaluator,
)

from synapse.ml.stages import ClassBalancer
from synapse.ml.train import ComputeModelStatistics

import mlflow
```

## Download dataset and upload to Lakehouse

Please add a Lakehouse to the notebook before running it.

Python

```
if not IS_CUSTOM_DATA:
    # Download demo data files into lakehouse if not exist
    import os, requests

    remote_url = "https://synapseaisolutionsa.blob.core.windows.net/public/Title_Genre_Classification"
    fname = "blbooksgenre.csv"
    download_path = f"/lakehouse/default/{DATA_FOLDER}/raw"

    if not os.path.exists("/lakehouse/default"):
        raise FileNotFoundError(
            "Default lakehouse not found, please add a lakehouse and restart the session."
        )
    os.makedirs(download_path, exist_ok=True)
    if not os.path.exists(f"{download_path}/{fname}"):
        r = requests.get(f"{remote_url}/{fname}", timeout=30)
        with open(f"{download_path}/{fname}", "wb") as f:
            f.write(r.content)
    print("Downloaded demo data files into lakehouse.")
```

## Read data from Lakehouse

Python

```
raw_df = spark.read.csv(f"{{DATA_FOLDER}}/raw/{{DATA_FILE}}", header=True, inferSchema=True)

display(raw_df.limit(20))
```

## Step 2: Preprocess data

### Data clean

Python

```
df = (
    raw_df.select([TEXT_COL, LABEL_COL])
    .where(F.col(LABEL_COL).isin(LABELS))
    .dropDuplicates([TEXT_COL])
    .cache()
)

display(df.limit(20))
```

### Deal with unbalanced data

Python

```
cb = ClassBalancer().setInputCol(LABEL_COL)

df = cb.fit(df).transform(df)
display(df.limit(20))
```

### Tokenize

Python

```
## text transformer
tokenizer = Tokenizer(inputCol=TEXT_COL, outputCol="tokens")
stopwords_remover = StopWordsRemover(inputCol="tokens", outputCol="filtered_tokens")

## build the pipeline
pipeline = Pipeline(stages=[tokenizer, stopwords_remover])

token_df = pipeline.fit(df).transform(df)

display(token_df.limit(20))
```

## Visualization

Display a Wordcloud for each class.

Python

```
# WordCloud
for label in LABELS:
    tokens = (
        token_df.where(F.col(LABEL_COL) == label)
        .select(F.explode("filtered_tokens").alias("token"))
        .where(F.col("token").rlike(r"^\w+$"))
    )

    top50_tokens = (
        tokens.groupBy("token").count().orderBy(F.desc("count")).limit(50).collect()
    )

    # Generate a word cloud image
    wordcloud = WordCloud(
        scale=10,
        background_color="white",
```

```

    random_state=42, # Make sure the output is always the same for the same input
).generate_from_frequencies(dict(top50_tokens))

# Display the generated image the matplotlib way:
plt.figure(figsize=(10, 10))
plt.title(label, fontsize=20)
plt.axis("off")
plt.imshow(wordcloud, interpolation="bilinear")

```

## Vectorize

We use word2vec to vectorize text.

Python

```

## label transformer
label_indexer = StringIndexer(inputCol=LABEL_COL, outputCol="labelIdx")
vectorizer = Word2Vec(
    vectorSize=word2vec_size,
    minCount=min_word_count,
    inputCol="filtered_tokens",
    outputCol="features",
)

## build the pipeline
pipeline = Pipeline(stages=[label_indexer, vectorizer])
vec_df = (
    pipeline.fit(token_df)
    .transform(token_df)
    .select([TEXT_COL, LABEL_COL, "features", "labelIdx", "weight"])
)
display(vec_df.limit(20))

```

## Step 3: Model training and evaluation

We've cleaned the dataset, dealt with unbalanced data, tokenized the text, displayed word cloud and vectorized the text.

Next, we train a linear regression model to classify the vectorized text.

### Split dataset into train and test

Python

```
(train_df, test_df) = vec_df.randomSplit((0.8, 0.2), seed=42)
```

### Create the model

Python

```

lr = (
    LogisticRegression()
    .setMaxIter(max_iter)
    .setFeaturesCol("features")
    .setLabelCol("labelIdx")
    .setWeightCol("weight")
)

```

### Train model with cross validation

Python

```

param_grid = (
    ParamGridBuilder()
    .addGrid(lr.regParam, [0.03, 0.1, 0.3])
    .addGrid(lr.elasticNetParam, [0.0, 0.1, 0.2])
    .build()
)

```

```

if len(LABELS) > 2:
    evaluator_cls = MulticlassClassificationEvaluator
    evaluator_metrics = ["f1", "accuracy"]
else:
    evaluator_cls = BinaryClassificationEvaluator
    evaluator_metrics = ["areaUnderROC", "areaUnderPR"]
evaluator = evaluator_cls(labelCol="labelIdx", weightCol="weight")

crossval = CrossValidator(
    estimator=lr, estimatorParamMaps=param_grid, evaluator=evaluator, numFolds=k_folds
)

model = crossval.fit(train_df)

```

## Evaluate the model

Python

```

predictions = model.transform(test_df)

display(predictions)

```

Python

```

log_metrics = {}
for metric in evaluator_metrics:
    value = evaluator.evaluate(predictions, {evaluator.metricName: metric})
    log_metrics[metric] = value
    print(f"{metric}: {value:.4f}")

```

Python

```

metrics = ComputeModelStatistics(
    evaluationMetric="classification", labelCol="labelIdx", scoredLabelsCol="prediction"
).transform(predictions)
display(metrics)

```

Python

```

# collect confusion matrix value
cm = metrics.select("confusion_matrix").collect()[0][0].toArray()
print(cm)

# plot confusion matrix
sns.set(rc={"figure.figsize": (6, 4.5)})
ax = sns.heatmap(cm, annot=True, fmt=".20g")
ax.set_title("Confusion Matrix")
ax.set_xlabel("Predicted label")
ax.set_ylabel("True label")

```

## Log and Load Model with MLflow

Now that we have a model we're satisfied with, we can save it for later use. Here we use MLflow to log metrics/models, and load models back for prediction.

Python

```

# setup mlflow
mlflow.set_experiment(EXPERIMENT_NAME)

```

Python

```

# log model, metrics and params
with mlflow.start_run() as run:
    print("log model:")
    mlflow.spark.log_model(
        model,
        f"{EXPERIMENT_NAME}-lrm",
        registered_model_name=f"{EXPERIMENT_NAME}-lrm",
        dfs_tmpdir="Files/spark",
    )

```

```
print("log metrics:")
mlflow.log_metrics(log_metrics)

print("log parameters:")
mlflow.log_params(
    {
        "word2vec_size": word2vec_size,
        "min_word_count": min_word_count,
        "max_iter": max_iter,
        "k_folds": k_folds,
        "DATAFILE": DATA_FILE,
    }
)

model_uri = f"runs:{run.info.run_id}/{EXPERIMENT_NAME}-lrmrmodel"
print("Model saved in run %s" % run.info.run_id)
print(f"Model URI: {model_uri}")
```

Python

```
# load model back
loaded_model = mlflow.spark.load_model(model_uri, dfs_tmpdir="Files/spark")

# verify loaded model
predictions = loaded_model.transform(test_df)
display(predictions)
```

# Creating, training, and evaluating uplift models in Microsoft Fabric

Article • 05/23/2023

In this article, learn how to create, train and evaluate uplift models and apply uplift modeling technique.

## Important

Microsoft Fabric is currently in PREVIEW. This information relates to a prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, expressed or implied, with respect to the information provided here.

- What is uplift modeling?

It's a family of causal inference technology that uses machine learning models to estimate the causal effect of some treatment on an individual's behavior.

- **Persuadables** only respond positive to the treatment
- **Sleeping-dogs** have a strong negative response to the treatment
- **Lost causes** never reach the outcome even with the treatment
- **Sure things** always reach the outcome with or without the treatment

The goal of uplift modeling is to identify the "persuadables", not waste efforts on "sure things" and "lost causes", and avoid bothering "sleeping dogs"

- How does uplift modeling work?
  - **Meta Learner**: predicts the difference between an individual's behavior when there's a treatment and when there's no treatment
  - **Uplift Tree**: a tree-based algorithm where the splitting criterion is based on differences in uplift
  - **NN-based Model** : a neural network model that usually works with observational data
- Where can uplift modeling work?
  - Marketing: help to identify persuadables to apply a treatment such as a coupon or an online advertisement
  - Medical Treatment: help to understand how a treatment can affect certain groups differently

# Prerequisites

- A familiarity with [How to use Microsoft Fabric notebooks](#).
- A Lakehouse. The Lakehouse is used to store data for this example. For more information, see [Add a Lakehouse to your notebook](#).

## Step 1: Load the data

### Tip

The following examples assume that you are running the code from cells in a notebook. For information on creating and using notebooks, see [How to use notebooks](#).

## Notebook configurations

By defining below parameters, you can apply this example to different datasets.

### Python

```
IS_CUSTOM_DATA = False # if True, dataset has to be uploaded manually by user
DATA_FOLDER = "Files/uplift-modelling"
DATA_FILE = "criteo-research-uplift-v2.1.csv"

# data schema
FEATURE_COLUMNS = [f"f{i}" for i in range(12)]
TREATMENT_COLUMN = "treatment"
LABEL_COLUMN = "visit"

EXPERIMENT_NAME = "aisample-upliftmodelling" # mlflow experiment name
```

## Import dependencies

### Python

```
import pyspark.sql.functions as F
from pyspark.sql.window import Window
from pyspark.sql.types import *

import numpy as np
import pandas as pd

import matplotlib as mpl
```

```
import matplotlib.pyplot as plt
import matplotlib.style as style
import seaborn as sns

%matplotlib inline

from synapse.ml.featurize import Featurize
from synapse.ml.core.spark import FluentAPI
from synapse.ml.lightgbm import *
from synapse.ml.train import ComputeModelStatistics

import os
import gzip

import mlflow
```

## Download dataset and upload to Lakehouse

### ⓘ Important

Add a Lakehouse to your notebook before running it.

- Dataset description: The Criteo AI Lab created this dataset. The dataset consists of 13M rows, each one representing a user with 12 features, a treatment indicator and 2 binary labels (visits and conversions).
  - f0, f1, f2, f3, f4, f5, f6, f7, f8, f9, f10, f11: feature values (dense, float)
  - treatment: treatment group (1 = treated, 0 = control) which indicates if a customer was targeted by advertising randomly
  - conversion: whether a conversion occurred for this user (binary, label)
  - visit: whether a visit occurred for this user (binary, label)
- Dataset homepage: <https://ailab.criteo.com/criteo-uplift-prediction-dataset/>
- Citation:

```
@inproceedings{Diemert2018,
author = {{Diemert Eustache, Betlei Artem} and Renaudin, Christophe and Massih-Reza, Amini},
title={A Large Scale Benchmark for Uplift Modeling},
publisher = {ACM},
booktitle = {Proceedings of the AdKDD and TargetAd Workshop, KDD, London, United Kingdom, August, 20, 2018},
year = {2018}
}
```

Python

```
if not IS_CUSTOM_DATA:
    # Download demo data files into lakehouse if not exist
    import os, requests

    remote_url = "http://go.criteo.net/criteo-research-uplift-v2.1.csv.gz"
    download_file = "criteo-research-uplift-v2.1.csv.gz"
    download_path = f"/lakehouse/default/{DATA_FOLDER}/raw"

    if not os.path.exists("/lakehouse/default"):
        raise FileNotFoundError(
            "Default lakehouse not found, please add a lakehouse and
            restart the session."
        )
    os.makedirs(download_path, exist_ok=True)
    if not os.path.exists(f"{download_path}/{DATA_FILE}"):
        r = requests.get(f"{remote_url}", timeout=30)
        with open(f"{download_path}/{download_file}", "wb") as f:
            f.write(r.content)
        with gzip.open(f"{download_path}/{download_file}", "rb") as fin:
            with open(f"{download_path}/{DATA_FILE}", "wb") as fout:
                fout.write(fin.read())
    print("Downloaded demo data files into lakehouse.")
```

## Read data from Lakehouse

Python

```
raw_df = spark.read.csv(
    f"{DATA_FOLDER}/raw/{DATA_FILE}", header=True, inferSchema=True
).cache()

display(raw_df.limit(20))
```

## Step 2: Prepare the dataset

### Data exploration

- The overall rate of users that visit/convert

Python

```
raw_df.select(  
    F.mean("visit").alias("Percentage of users that visit"),  
    F.mean("conversion").alias("Percentage of users that convert"),  
    (F.sum("conversion") / F.sum("visit")).alias("Percentage of  
    visitors that convert"),  
).show()
```

- The overall average treatment effect on visit

Python

```
raw_df.groupby("treatment").agg(  
    F.mean("visit").alias("Mean of visit"),  
    F.sum("visit").alias("Sum of visit"),  
    F.count("visit").alias("Count"),  
).show()
```

- The overall average treatment effect on conversion

Python

```
raw_df.groupby("treatment").agg(  
    F.mean("conversion").alias("Mean of conversion"),  
    F.sum("conversion").alias("Sum of conversion"),  
    F.count("conversion").alias("Count"),  
).show()
```

## Split train-test dataset

Python

```
transformer = (  
  
    Featurize().setOutputCol("features").setInputCols(FEATURE_COLUMNS).fit(raw_d  
f)  
)  
  
df = transformer.transform(raw_df)
```

Python

```
train_df, test_df = df.randomSplit([0.8, 0.2], seed=42)  
  
print("Size of train dataset: %d" % train_df.count())  
print("Size of test dataset: %d" % test_df.count())  
  
train_df.groupby(TREATMENT_COLUMN).count().show()
```

## Split treatment-control dataset

Python

```
treatment_train_df = train_df.where(f"{TREATMENT_COLUMN} > 0")
control_train_df = train_df.where(f"{TREATMENT_COLUMN} = 0")
```

## Step 3: Model training and evaluation

### Uplift modeling: T-Learner with LightGBM

Python

```
classifier = (
    LightGBMClassifier()
    .setFeaturesCol("features")
    .setNumLeaves(10)
    .setNumIterations(100)
    .setObjective("binary")
    .setLabelCol(LABEL_COLUMN)
)

treatment_model = classifier.fit(treatment_train_df)
control_model = classifier.fit(control_train_df)
```

## Predict on test dataset

Python

```
getPred = F.udf(lambda v: float(v[1]), FloatType())

test_pred_df = (
    test_df.mlTransform(treatment_model)
    .withColumn("treatment_pred", getPred("probability"))
    .drop("rawPrediction", "probability", "prediction")
    .mlTransform(control_model)
    .withColumn("control_pred", getPred("probability"))
    .drop("rawPrediction", "probability", "prediction")
    .withColumn("pred_uplift", F.col("treatment_pred") -
    F.col("control_pred"))
    .select(
        TREATMENT_COLUMN, LABEL_COLUMN, "treatment_pred", "control_pred",
        "pred_uplift"
    )
)
```

```
.cache()  
)  
  
display(test_pred_df.limit(20))
```

## Model evaluation

Since actual uplift can't be observed for each individual, measure the uplift over a group of customers.

- **Uplift Curve:** plots the real cumulative uplift across the population

First, rank the test dataframe order by the predict uplift.

Python

```
test_ranked_df = test_pred_df.withColumn(  
    "percent_rank",  
    F.percent_rank().over(Window.orderBy(F.desc("pred_uplift"))))  
)  
  
display(test_ranked_df.limit(20))
```

Next, calculate the cumulative percentage of visits in each group (treatment or control).

Python

```
C = test_ranked_df.where(F"{TREATMENT_COLUMN} == 0").count()  
T = test_ranked_df.where(F"{TREATMENT_COLUMN} != 0").count()  
  
test_ranked_df = (  
    test_ranked_df.withColumn(  
        "control_label",  
        F.when(F.col(TREATMENT_COLUMN) == 0,  
F.col(LABEL_COLUMN)).otherwise(0),  
    )  
    .withColumn(  
        "treatment_label",  
        F.when(F.col(TREATMENT_COLUMN) != 0,  
F.col(LABEL_COLUMN)).otherwise(0),  
    )  
    .withColumn(  
        "control_cumsum",  
        F.sum("control_label").over(Window.orderBy("percent_rank")) / C,  
    )  
    .withColumn(  
        "treatment_cumsum",  
        F.sum("treatment_label").over(Window.orderBy("percent_rank")) / T,  
    )  
)
```

```
display(test_ranked_df.limit(20))
```

Finally, calculate the group's uplift at each percentage.

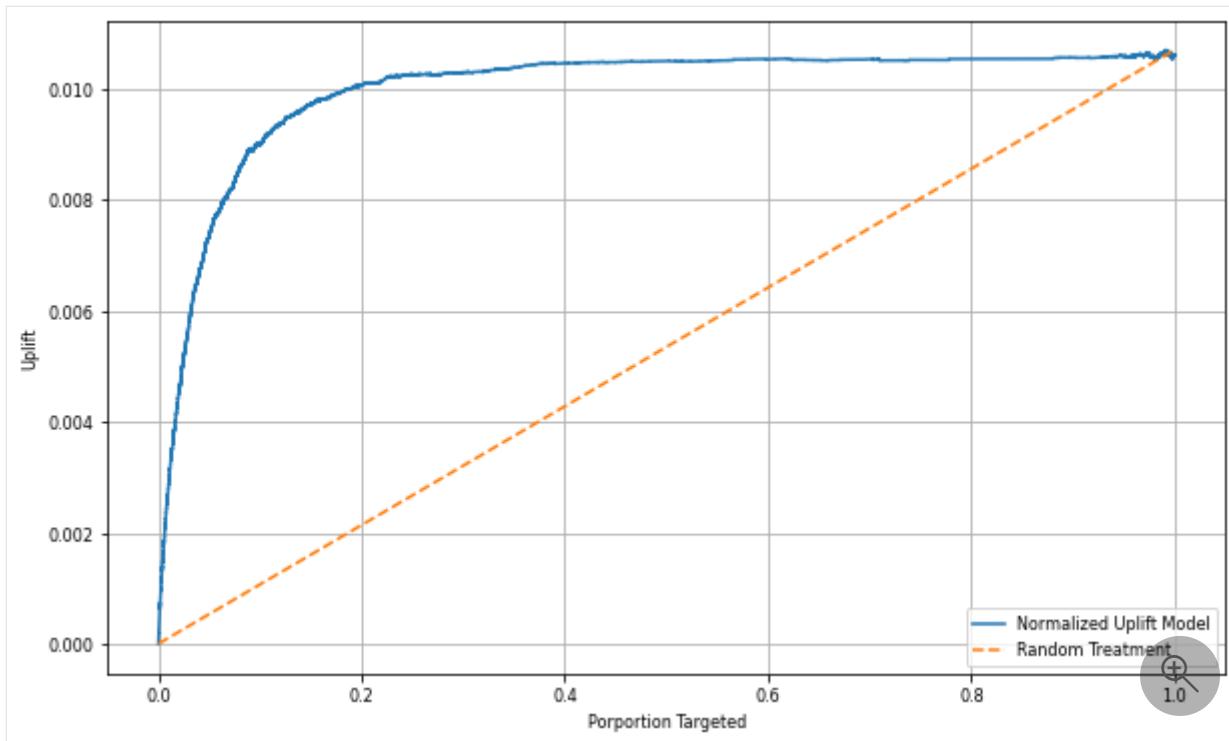
Python

```
test_ranked_df = test_ranked_df.withColumn(  
    "group_uplift", F.col("treatment_cumsum") - F.col("control_cumsum"))  
    .cache()  
  
display(test_ranked_df.limit(20))
```

Now you can plot the uplift curve on the prediction of the test dataset. You need to convert the pyspark dataframe to pandas dataframe before plotting.

Python

```
def uplift_plot(uplift_df):  
    """  
    Plot the uplift curve  
    """  
  
    gain_x = uplift_df.percent_rank  
    gain_y = uplift_df.group_uplift  
    # plot the data  
    plt.figure(figsize=(10, 6))  
    mpl.rcParams["font.size"] = 8  
  
    ax = plt.plot(gain_x, gain_y, color="#2077B4", label="Normalized Uplift  
Model")  
  
    plt.plot(  
        [0, gain_x.max()],  
        [0, gain_y.max()],  
        "--",  
        color="tab:orange",  
        label="Random Treatment",  
    )  
    plt.legend()  
    plt.xlabel("Porportion Targeted")  
    plt.ylabel("Uplift")  
    plt.grid(b=True, which="major")  
  
    return ax  
  
  
test_ranked_pd_df = test_ranked_df.select(  
    ["pred_uplift", "percent_rank", "group_uplift"]  
).toPandas()  
uplift_plot(test_ranked_pd_df)
```



From the uplift curve in the previous example, notice that the top 20% population ranked by your prediction have a large gain if they were given the treatment, which means they're the **persuadables**. Therefore, you can print the cutoff score at 20% percentage to identify the target customers.

Python

```
cutoff_percentage = 0.2
cutoff_score = test_ranked_pd_df.iloc[int(len(test_ranked_pd_df) * 
cutoff_percentage)][
    "pred_uplift"]
]

print("Uplift score higher than {:.4f} are
Persuadables".format(cutoff_score))
```

## Log and load model with MLflow

Now that you have a trained model, save it for later use. In the following example, MLflow is used to log metrics and models. You can also use this API to load models for prediction.

Python

```
# setup mlflow
mlflow.set_experiment(EXPERIMENT_NAME)
```

### Python

```
# log model, metrics and params
with mlflow.start_run() as run:
    print("log model:")
    mlflow.spark.log_model(
        treatment_model,
        f"{EXPERIMENT_NAME}-treatmentmodel",
        registered_model_name=f"{EXPERIMENT_NAME}-treatmentmodel",
        dfs_tmpdir="Files/spark",
    )

    mlflow.spark.log_model(
        control_model,
        f"{EXPERIMENT_NAME}-controlmodel",
        registered_model_name=f"{EXPERIMENT_NAME}-controlmodel",
        dfs_tmpdir="Files/spark",
    )

model_uri = f"runs:{run.info.run_id}/{EXPERIMENT_NAME}"
print("Model saved in run %s" % run.info.run_id)
print(f"Model URI: {model_uri}-treatmentmodel")
print(f"Model URI: {model_uri}-controlmodel")
```

### Python

```
# load model back
loaded_treatmentmodel = mlflow.spark.load_model(
    f"{model_uri}-treatmentmodel", dfs_tmpdir="Files/spark"
)
```

# Data science roles and permissions

Article • 05/23/2023

This article describes machine learning model and experiment permissions in Microsoft Fabric and how these permissions are acquired by users.

## ⓘ Important

Microsoft Fabric is currently in PREVIEW. This information relates to a prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, expressed or implied, with respect to the information provided here.

## ⓘ Note

After you create a workspace in Microsoft Fabric, or if you have an admin role in a workspace, you can give others access to it by assigning them a different role. To learn more about workspaces and how to grant users access to your workspace, review the articles [Workspaces](#) and [Giving users access to workspaces](#).

## Permissions for machine learning experiments

The table below describes the levels of permission that control access to machine learning experiments in Microsoft Fabric.

Permission	Description
Read	<p>Allows user to read machine learning experiments.</p> <p>Allows user to view runs within machine learning experiments.</p> <p>Allows user to view run metrics and parameters.</p> <p>Allows user to view and download run files.</p>
Write	<p>Allows user to create machine learning experiments.</p> <p>Allows user to modify or delete machine learning experiments.</p> <p>Allows user to add runs to machine learning experiments.</p> <p>Allows user to save an experiment run as a model.</p>

## Permissions for machine learning models

The table below describes the levels of permission that control access to machine learning models in Microsoft Fabric.

Permission	Description
Read	Allows user to read machine learning models. Allows user to view versions within machine learning models. Allows user to view model version metrics and parameters. Allows user to view and download model version files.
Write	Allows user to create machine learning models. Allows user to modify or delete machine learning models. Allows user to add model versions to machine learning models.

## Permissions acquired by workspace role

A user's role in a workspace implicitly grants them permissions on the datasets in the workspace, as described in the following table.

	Admin	Member	Contributor	Viewer
<b>Read</b>	✓	✓	✓	✓
<b>Write</b>	✓	✓	✓	✗

### ⓘ Note

You can either assign roles to individuals or to security groups, Microsoft 365 groups, and distribution lists. To learn more about workspace roles in Microsoft Fabric, see [Roles in workspaces](#)

## Next steps

- Learn about roles in workspaces: [Roles in Microsoft Fabric workspaces](#)
- Give users access to workspaces: [Granting access to users](#)

# Lineage for models and experiments

Article • 05/23/2023

In modern business intelligence (BI) projects, understanding the flow of data from the data source to its destination can be a challenge. The challenge is even bigger if you've built advanced analytical projects spanning multiple data sources, items, and dependencies. Questions like "What happens if I change this data?" or "Why isn't this report up to date?" can be hard to answer. They might require a team of experts or deep investigation to understand. The Fabric lineage view helps you answer these questions.

## Important

Microsoft Fabric is currently in PREVIEW. This information relates to a prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, expressed or implied, with respect to the information provided here.

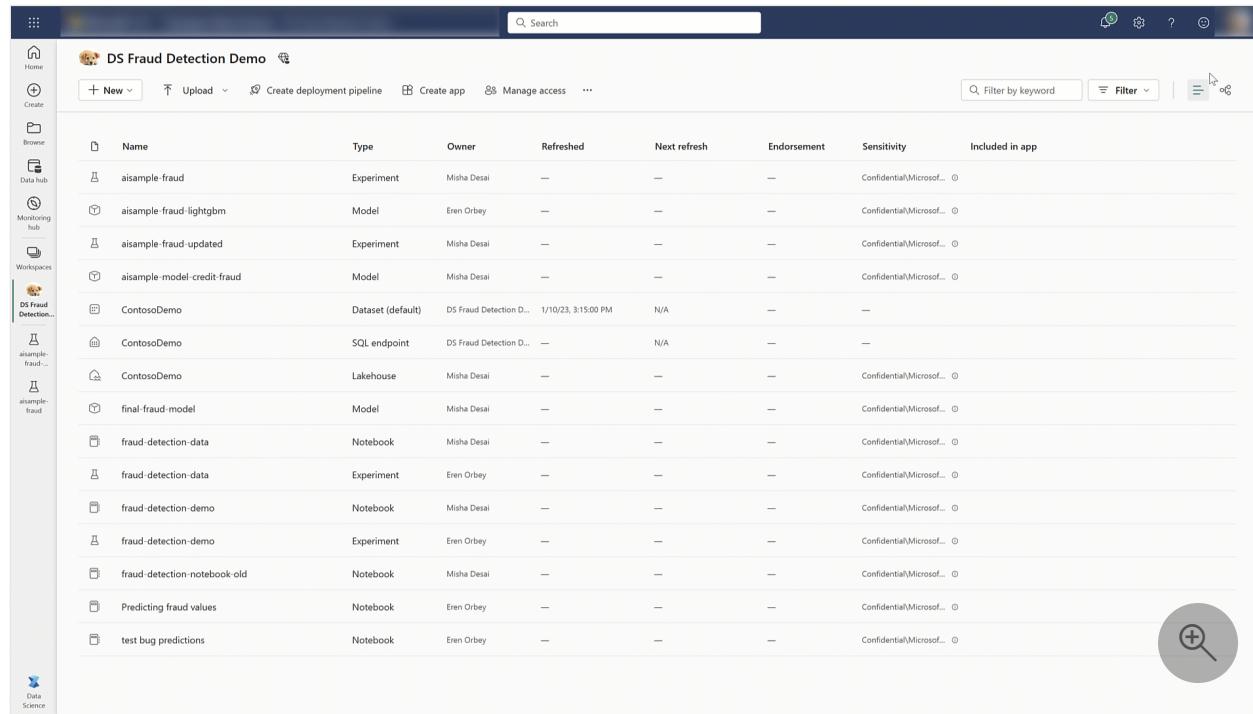
## Lineage and machine learning

There are several reasons why lineage is important in your machine learning workflow:

- **Reproducibility:** Knowing the lineage of a model makes it easier to reproduce the model and its results. If someone else wants to replicate the model, they can follow the same steps that were used to create it, and use the same data and parameters.
- **Transparency:** Understanding the lineage of a model helps to increase its transparency. This means that stakeholders, such as regulators or users, can understand how the model was created, and how it works. This can be important for ensuring fairness, accountability, and ethical considerations.
- **Debugging:** If a model is not performing as expected, knowing its lineage can help to identify the source of the problem. By examining the training data, parameters, and decisions that were made during the training process, it may be possible to identify issues that are affecting the model's performance.
- **Improvement:** Knowing the lineage of a model can also help to improve it. By understanding how the model was created and trained, it may be possible to make changes to the training data, parameters, or process that can improve the model's accuracy or other performance metrics.

## Data science item types

In Fabric, machine learning models and experiments are integrated into a unified platform. As part of this, users can browse the relationship between Fabric Data Science items and other Fabric items.



The screenshot shows the Microsoft Fabric Data Science workspace interface. The left sidebar includes links for Home, Create, Browse, Data Hub, Monitoring hub, Workspaces, DS Fraud Detection, and Data Science. The main area displays a table of items under the heading 'DS Fraud Detection Demo'. The columns are: Name, Type, Owner, Refreshed, Next refresh, Endorsement, Sensitivity, and Included in app. The table lists various items such as 'aisample-fraud' (Experiment), 'aisample-fraud-lightgbm' (Model), 'aisample-fraud-updated' (Experiment), 'aisample-model-credit-fraud' (Model), 'ContosoDemo' (Dataset), 'ContosoDemo' (SQL endpoint), 'ContosoDemo' (Lakehouse), 'final-fraud-model' (Model), 'fraud-detection-data' (Notebook), 'fraud-detection-data' (Experiment), 'fraud-detection-demo' (Notebook), 'fraud-detection-demo' (Experiment), 'fraud-detection-notebook-old' (Notebook), 'Predicting fraud values' (Notebook), and 'test bug predictions' (Notebook). A search bar at the top right contains the placeholder 'Search'.

## Machine learning models

In Fabric, users can create and manage machine learning models. A machine learning model item represents a versioned list of models, allowing the user to browse the various iterations of the model.

In the lineage view, users can browse the relationship between a machine learning model and other Fabric items to answer the following questions:

- What is the relationship between machine learning models and experiments within my workspace?
- Which machine learning models exist in my workspace?
- How can I trace back the lineage to see which Lakehouse items were related to this model?

## Machine learning experiments

A machine learning *experiment* is the primary unit of organization and control for all related machine learning runs.

In the lineage view, users can browse the relationship between a machine learning experiment and other Fabric items to answer the following questions:

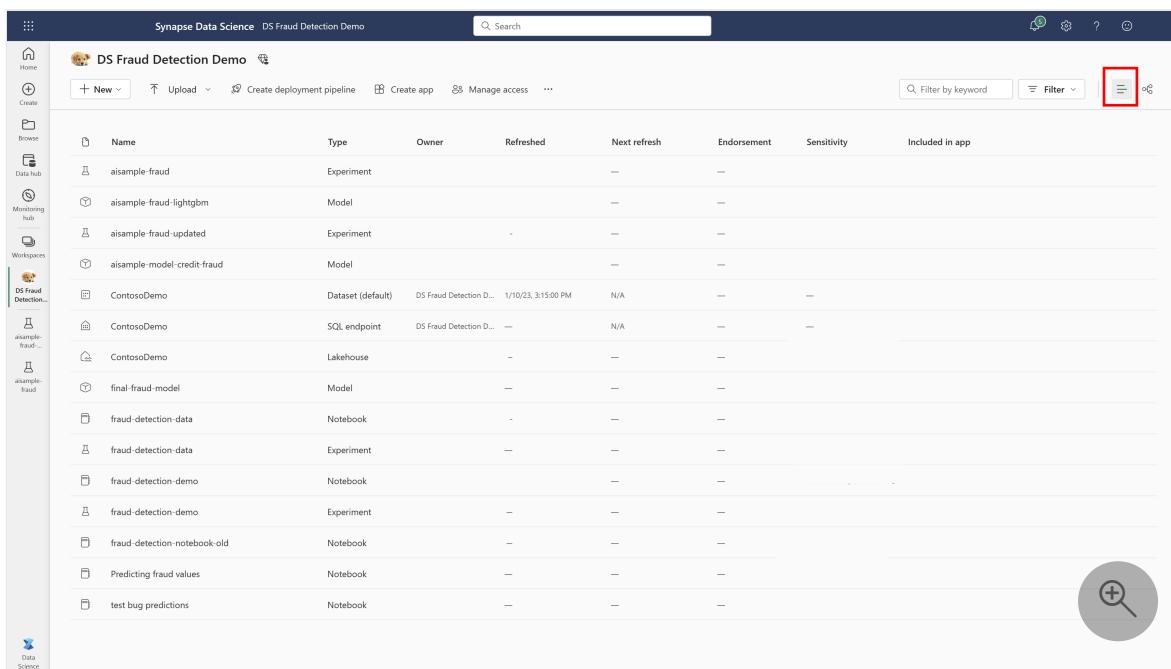
- What is the relationship between machine learning experiments and code items (e.g. notebooks and Spark Job Definitions) within my workspace?
- Which machine learning experiments exist in my workspace?
- How can I trace back the lineage to see which Lakehouse items were related to this experiment?

## Explore lineage view

Every Fabric workspace automatically has a built-in lineage view. To access this view, you must have at least the **Contributor** role within the workspace. To learn more about permissions in Fabric, you can visit the documentation on [permissions for models and experiments](#).

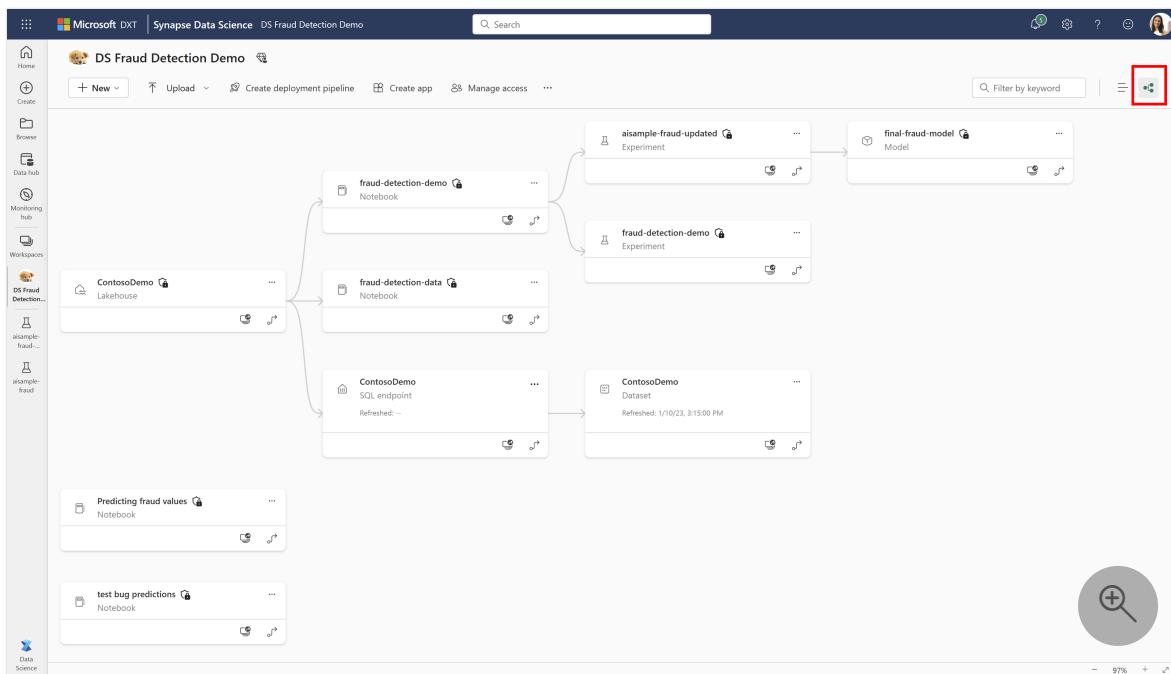
To access the lineage view:

1. Select your Fabric workspace and then navigate to the workspace list.



The screenshot shows the Synapse Data Science interface with the 'DS Fraud Detection Demo' workspace selected. The left sidebar shows navigation options like Home, Create, Browse, Data hub, Monitoring hub, Workspaces, and DS Fraud Detection. The main area displays a table of workspace items with columns: Name, Type, Owner, Refreshed, Next refresh, Endorsement, Sensitivity, and Included in app. Items listed include 'aisample-fraud', 'aisample-fraud-lightgbm', 'aisample-fraud-updated', 'aisample-model-credit-fraud', 'ContosoDemo' (Dataset), 'ContosoDemo' (SQL endpoint), 'ContosoDemo' (Lakehouse), 'final-fraud-model', 'fraud-detection-data', 'fraud-detection-data', 'fraud-detection-demo', 'fraud-detection-demo', 'fraud-detection-notebook-old', 'Predicting fraud values', and 'test bug predictions'. The top right of the interface has a toolbar with icons for New, Upload, Create deployment pipeline, Create app, Manage access, and more. A search bar and filter options are also present. A large circular button with a magnifying glass icon is located in the bottom right corner of the main area.

2. Switch from the workspace **List** view to the Workspace **Lineage** view.



3. You can also navigate to **Lineage** view for a specific item by opening the related actions.

The screenshot shows the Microsoft DXT Synapse Data Science workspace. On the left, a sidebar lists various items under the 'DS Fraud Detection...' category, including 'aisample-fraud' and 'aisample-fraud-updated'. The main area displays a table of items with columns: Name, Type, Owner, Refreshed, Next refresh, Endorsement, Sensitivity, and Included in app. One row for 'aisample-fraud' is selected, and its details are shown in a modal. The 'View lineage' action is highlighted with a red box. The top right corner of the workspace interface has a red box highlighting the 'Lineage' icon.

Name	Type	Owner	Refreshed	Next refresh	Endorsement	Sensitivity	Included in app
aisample-fraud	Experiment		—	—	—	—	
aisample-fraud-lightgbm	Open		—	—	—	—	
aisample-fraud-updated	Delete		—	—	—	—	
aisample-model-credit-fraud	Settings		—	—	—	—	
ContosoDemo	Add to Favorites		—	—	—	—	
ContosoDemo	View lineage		—	—	—	—	
ContosoDemo	Lakehouse		—	—	—	—	
final-fraud-model	Model		—	—	—	—	
fraud-detection-data	Notebook		—	—	—	—	
fraud-detection-demo	Experiment		—	—	—	—	
fraud-detection-notebook-old	Notebook		—	—	—	—	
Predicting fraud values	Notebook		—	—	—	—	
test bug predictions	Notebook		—	—	—	—	

## Next steps

- Learn about machine learning models: [Machine learning models](#)
- Learn about machine learning experiments: [Machine learning experiments](#)

# How to read and write data with Pandas in Microsoft Fabric

Article • 05/23/2023

Microsoft Fabric notebooks support seamless interaction with Lakehouse data using Pandas, the most popular Python library for data exploration and processing. Within a notebook, users can quickly read data from—and write data back to—their Lakehouses in a variety of file formats. This guide provides code samples to help you get started in your own notebook.

## Important

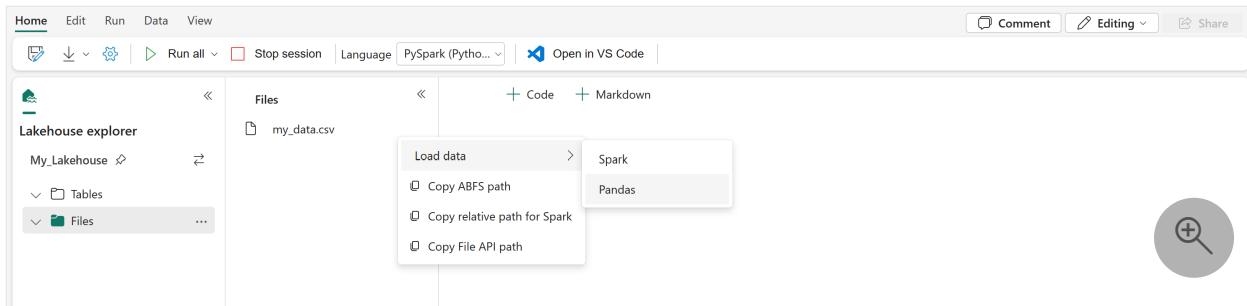
Microsoft Fabric is currently in PREVIEW. This information relates to a prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, expressed or implied, with respect to the information provided here.

## Prerequisites

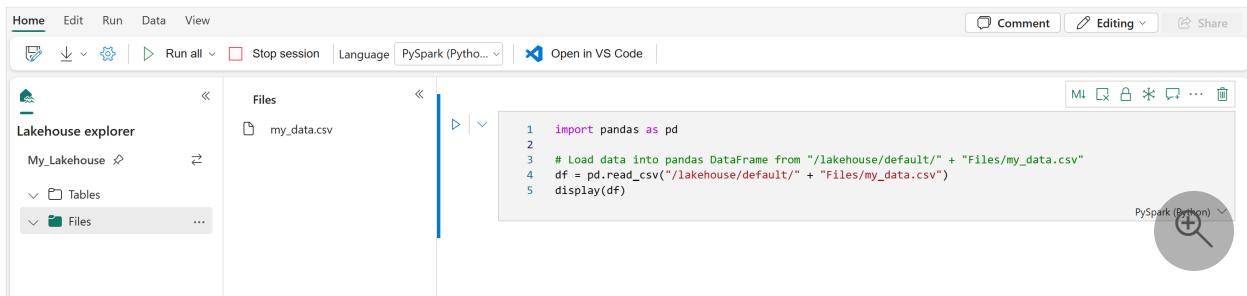
- A Power BI Premium subscription. If you don't have one, see [How to purchase Power BI Premium](#).
- A Power BI workspace with assigned Premium capacity. If you don't have a workspace, use the steps in [Create a workspace](#) to create one and assign it to a Premium capacity.
- Sign in to [Microsoft Fabric](#).

## Load Lakehouse data into a notebook

Once you attach a Lakehouse to your Microsoft Fabric notebook, you can explore stored data without leaving the page and read it into your notebook in a matter of clicks. Selecting any Lakehouse file surfaces options to "Load data" into a Spark or a Pandas DataFrame. (You can also copy the file's full ABFS path or a friendly relative path.)



Clicking on one of the "Load data" prompts will generate a code cell to load that file into a DataFrame in your notebook.



## Converting a Spark DataFrame into a Pandas DataFrame

For reference, the following command shows how to convert a Spark DataFrame into a Pandas DataFrame.

```
Python  
  
# Replace "spark_df" with the name of your own Spark DataFrame  
pandas_df = spark_df.toPandas()
```

## Reading and writing various file formats

The code samples below document the Pandas operations for reading and writing various file formats.

### ! Note

You must replace the file paths in the following samples. Pandas supports both relative paths, as shown here, and full ABFS paths. Either can be retrieved and copied from the interface according to the previous step.

## Read data from a CSV file

Python

```
import pandas as pd

# Read a CSV file from your Lakehouse into a Pandas DataFrame
# Replace LAKEHOUSE_PATH and FILENAME with your own values
df = pd.read_csv("/LAKEHOUSE_PATH/Files/Filename.csv")
display(df)
```

## Write data as a CSV file

Python

```
import pandas as pd

# Write a Pandas DataFrame into a CSV file in your Lakehouse
# Replace LAKEHOUSE_PATH and FILENAME with your own values
df.to_csv("/LAKEHOUSE_PATH/Files/Filename.csv")
```

## Read data from a Parquet file

Python

```
import pandas as pd

# Read a Parquet file from your Lakehouse into a Pandas DataFrame
# Replace LAKEHOUSE_PATH and FILENAME with your own values
df = pandas.read_parquet("/LAKEHOUSE_PATH/Files/Filename.parquet")
display(df)
```

## Write data as a Parquet file

Python

```
import pandas as pd

# Write a Pandas DataFrame into a Parquet file in your Lakehouse
# Replace LAKEHOUSE_PATH and FILENAME with your own values
df.to_parquet("/LAKEHOUSE_PATH/Files/Filename.parquet")
```

## Read data from an Excel file

Python

```
import pandas as pd

# Read an Excel file from your Lakehouse into a Pandas DataFrame
# Replace LAKEHOUSE_PATH and FILENAME with your own values
df = pandas.read_excel("/LAKEHOUSE_PATH/Files/Filename.xlsx")
display(df)
```

## Write data as an Excel file

Python

```
import pandas as pd

# Write a Pandas DataFrame into an Excel file in your Lakehouse
# Replace LAKEHOUSE_PATH and FILENAME with your own values
df.to_excel("/LAKEHOUSE_PATH/Files/Filename.xlsx")
```

## Read data from a JSON file

Python

```
import pandas as pd

# Read a JSON file from your Lakehouse into a Pandas DataFrame
# Replace LAKEHOUSE_PATH and FILENAME with your own values
df = pandas.read_json("/LAKEHOUSE_PATH/Files/Filename.json")
display(df)
```

## Write data as a JSON file

Python

```
import pandas as pd

# Write a Pandas DataFrame into a JSON file in your Lakehouse
# Replace LAKEHOUSE_PATH and FILENAME with your own values
df.to_json("/LAKEHOUSE_PATH/Files/Filename.json")
```

## Next steps

- Use Data Wrangler to clean and prepare your data
- Start training ML models

# How to accelerate data prep with Data Wrangler in Microsoft Fabric

Article • 05/23/2023

Data Wrangler is a notebook-based tool that provides users with an immersive experience to conduct exploratory data analysis. The feature combines a grid-like data display with dynamic summary statistics, built-in visualizations, and a library of common data-cleaning operations. Each operation can be applied in a matter of clicks, updating the data display in real time and generating code that can be saved back to the notebook as a reusable function.

## ⓘ Important

Microsoft Fabric is currently in PREVIEW. This information relates to a prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, expressed or implied, with respect to the information provided here.

## Prerequisites

- A Power BI Premium subscription. If you don't have one, see [How to purchase Power BI Premium](#).
- A Power BI workspace with assigned Premium capacity. If you don't have a workspace, use the steps in [Create a workspace](#) to create one and assign it to a Premium capacity.
- Sign in to [Microsoft Fabric](#).

## Limitations

- Data Wrangler currently supports *only* Pandas DataFrames. Support for Spark DataFrames is in progress.
- Data Wrangler's display works better on large monitors, although different portions of the interface can be minimized or hidden to accommodate smaller screens.

## Launch Data Wrangler

Users can launch Data Wrangler directly from a Microsoft Fabric notebook to explore and transform any Pandas DataFrame. This code snippet shows how to read sample data into a Pandas DataFrame:

```
Python

import pandas as pd

# Read a CSV into a Pandas DataFrame from e.g. a public blob store
df =
pd.read_csv("https://raw.githubusercontent.com/plotly/datasets/master/titanic.csv")
```

Under the notebook ribbon “Data” tab, use the Data Wrangler dropdown prompt to browse the active Pandas DataFrames available for editing. Select the one you wish to open in Data Wrangler.

### Tip

Data Wrangler cannot be opened while the notebook kernel is busy. An executing cell must finish its execution before Data Wrangler can be launched.

The screenshot shows the Microsoft Fabric Data Wrangler interface. On the left, the Data Hub sidebar is visible with options like Home, Create, Browse, Data Hub, Monitoring Hub, Workspaces, Data Wrangler, and Titanic Data Cleaning. The main area has a ribbon with Home, Edit, Run, Data, and View tabs. The Data tab is selected, and a dropdown menu is open under 'Launch Data Wrangler' with the option 'Select Pandas DataFrame to transform'. A search bar below it shows 'Search' and 'df'. To the right, a code editor window displays Python code for reading a CSV file into a Pandas DataFrame:

```
1 import pandas as pd
2 # Load data into pandas DataFrame from "/lakehouse/default/" + "Files/titanic.csv"
3 df = pd.read_csv("/lakehouse/default/" + "Files/titanic.csv")
4 display(df)
```

Below the code, a log message indicates: "4 sec - Apache Spark session started in 342 ms. Command executed in 1 sec 387 ms by Eren Orbey on 1:18:53 PM, 4/21/23". The PySpark (Python) tab is selected. To the right of the code editor is a data preview table for the 'titanic' DataFrame. The table has columns: Index, PassengerId, Survived, Pclass, Name, Age, SibSp, and Parch. The first 14 rows of data are listed:

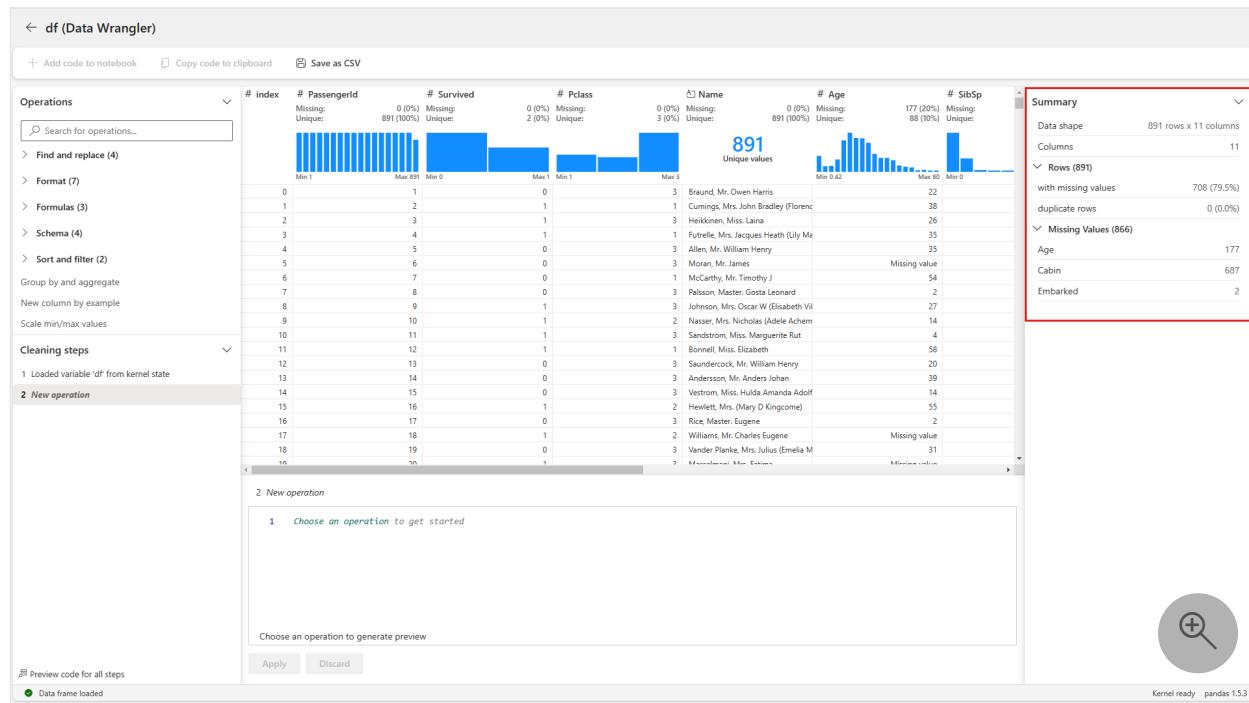
Index	PassengerId	Survived	Pclass	Name	Age	SibSp	Parch
1	1	0	3	Braund, Mr. Owen Harris	22.0	1	0
2	2	1	1	Cumings, Mrs. John Bradley (Fl... Heikkinen, Miss. Laina	38.0 26.0	1 0	0 0
3	3	1	3	Futrelle, Mrs. Jacques Heath (L... Allen, Mr. William Henry	35.0 35.0	1 0	0 0
4	4	1	1	Moran, Mr. James	NULL	0	0
5	5	0	3	McCarthy, Mr. Timothy J	54.0	0	0
6	6	0	3	Palsson, Master. Gosta Leonard	2.0	3	1
7	7	0	1	Johnson, Mrs. Oscar W (Elisab... Nasser, Mrs. Nicholas (Adèle A... Sandstrom, Miss. Marguerite R...	27.0 14.0 4.0	0 1 1	2 0 1
8	8	0	3	Bonnell, Miss. Elizabeth	58.0	0	0
9	9	1	3	Saundercock, Mr. William Henry	20.0	0	0
10	10	1	2	Andersson, Mr. Anders Joha...	39.0	1	5
11	11	1	3				
12	12	1	1				
13	13	0	3				
14	14	0	3				

## Viewing summary statistics

When Data Wrangler launches, it generates a descriptive overview of the displayed DataFrame in the Summary panel. This overview includes information about the DataFrame's dimensions, missing values, and more. Selecting any column in the Data Wrangler grid prompts the Summary panel to update and display descriptive statistics about that specific column. Quick insights about every column are also available in its header.

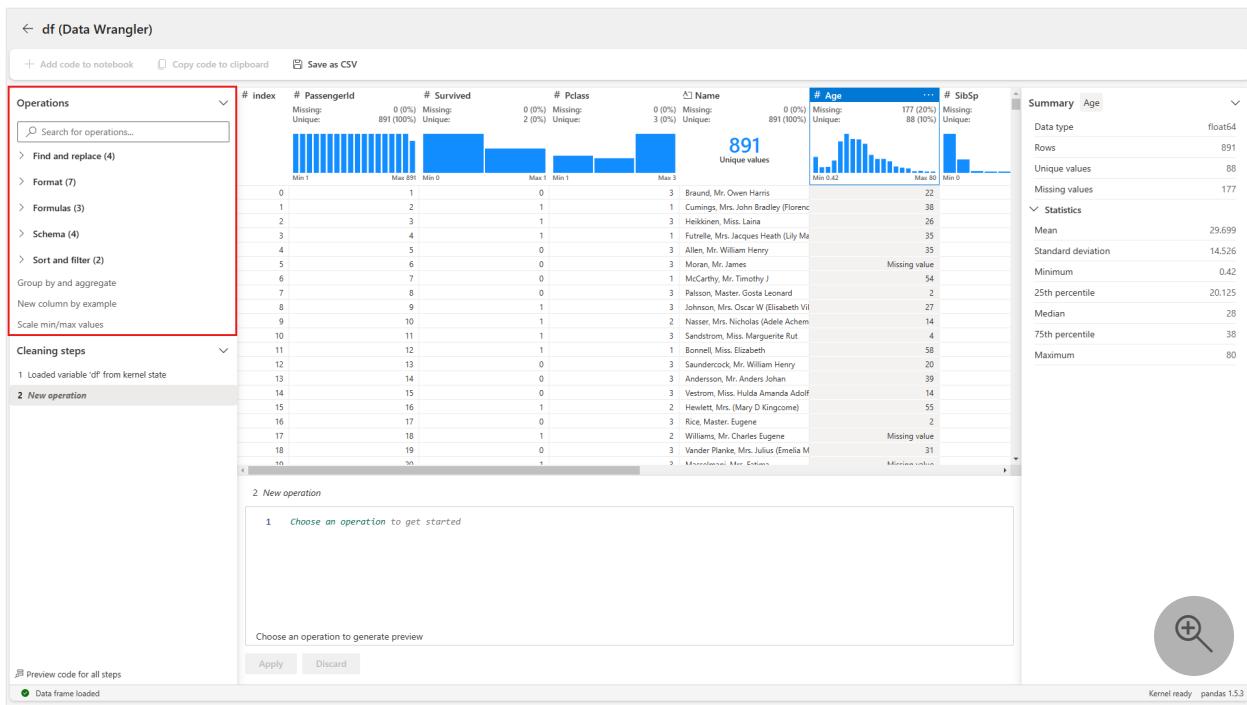
### 💡 Tip

Column-specific statistics and visuals (in both the Summary panel and in the column headers) depend on the column datatype. For instance, a binned histogram of a numeric column will appear in the column header only if the column is cast as a numeric type. Use the Operations panel to recast column types for the most accurate display.



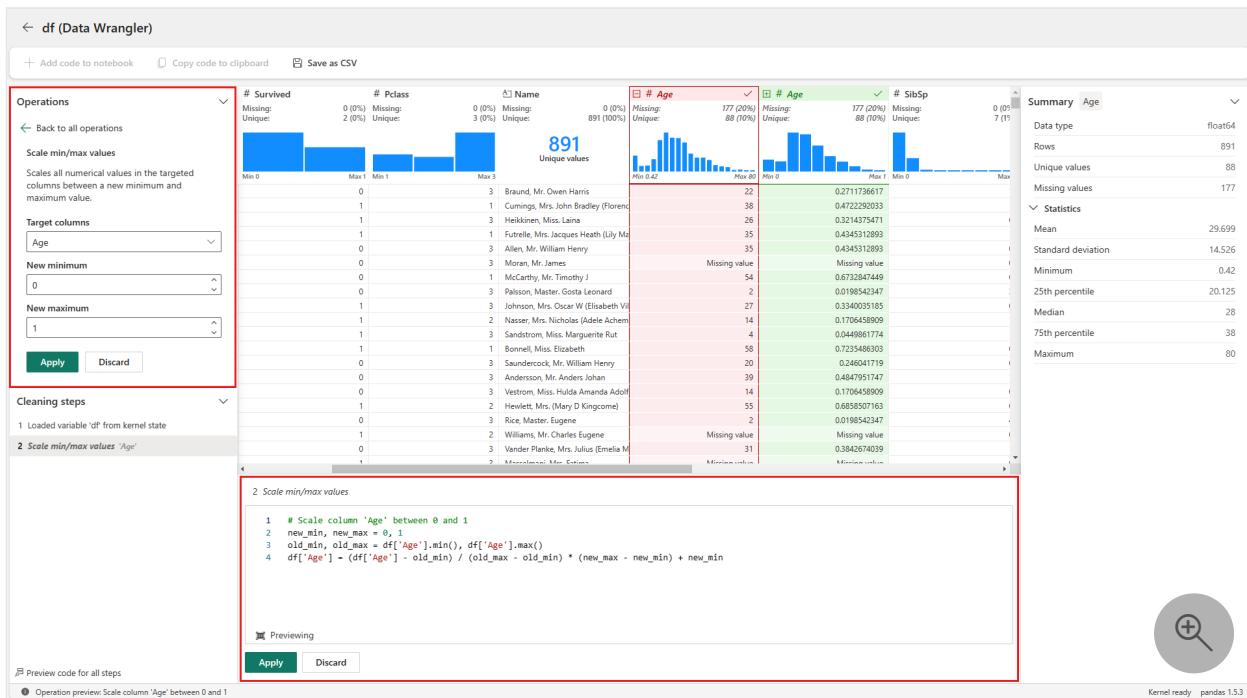
## Browsing data-cleaning operations

A searchable list of data-cleaning steps can be found in the Operations panel. (You can also access a smaller selection of the same operations in the contextual menu of each column.) From the Operations panel, selecting a data-cleaning step prompts you to select a target column or columns, along with any necessary parameters to complete the step. For example, the prompt for scaling a column numerically requires a new range of values.

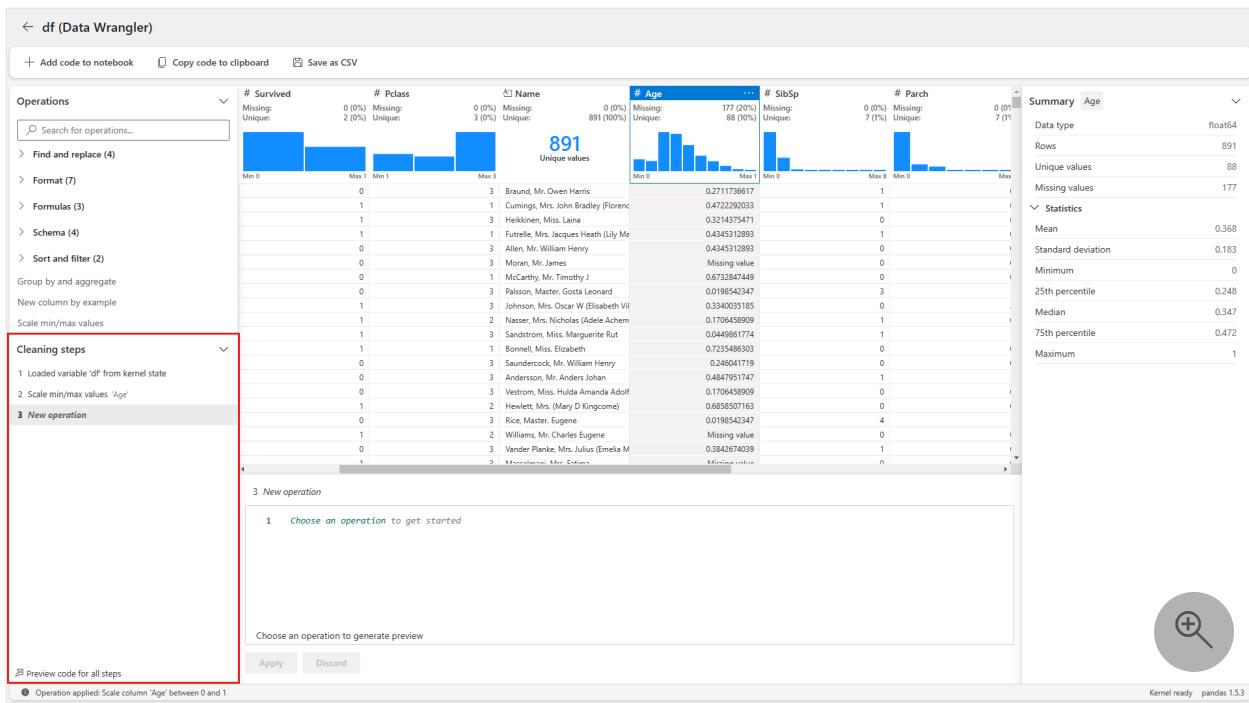


## Previewing and applying operations

The results of a selected operation will be previewed automatically in the Data Wrangler display grid, and the corresponding code will automatically appear in the panel below the grid. To commit the previewed code, select “Apply” in either place. To get rid of the previewed code and try a new operation, select “Discard.”

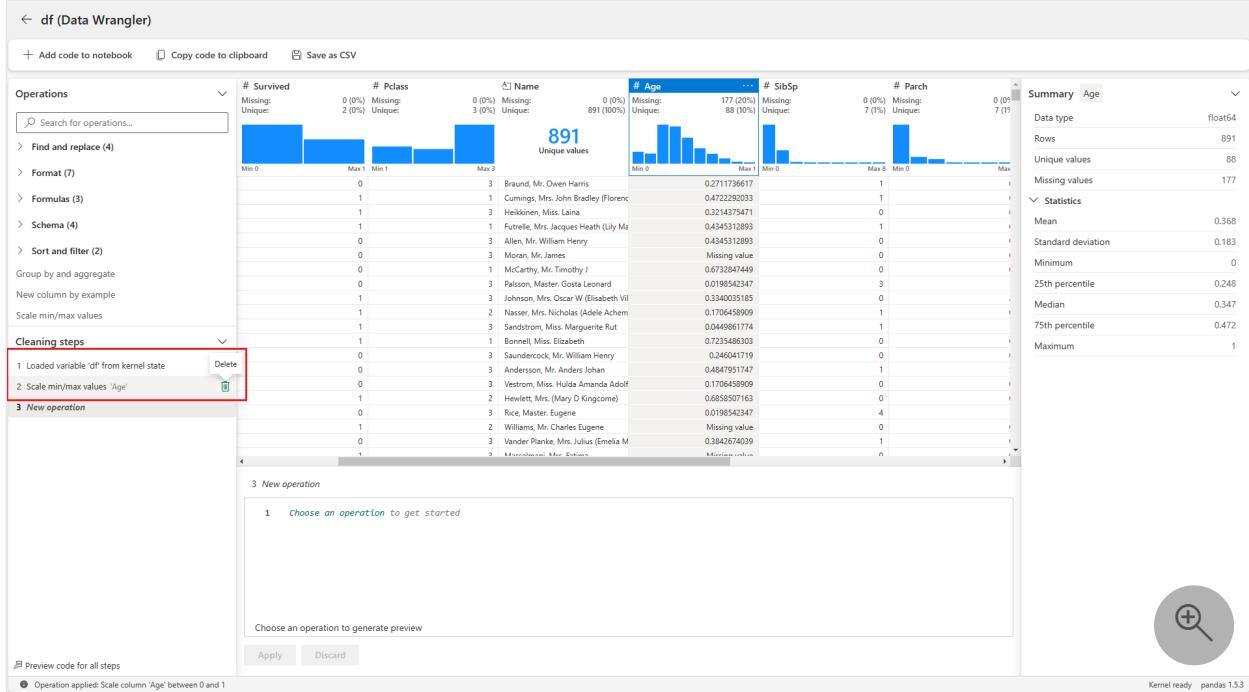


Once an operation is applied, the Data Wrangler display grid and summary statistics update to reflect the results. The previewed code appears in the running list of committed operations, located in the Cleaning steps panel.



## Tip

You can always undo the most recently applied step with the trash icon beside it, which appears if you hover your cursor over that step in the Cleaning steps panel.



The following table summarizes the operations that Data Wrangler currently supports:

Operation	Description
Sort	Sort a column in ascending or descending order
Filter	Filter rows based on one or more conditions

Operation	Description
<b>One-hot encode</b>	Create new columns for each unique value in an existing column, indicating the presence or absence of those values per row
<b>One-hot encode with delimiter</b>	Split and one-hot encode categorical data using a delimiter
<b>Change column type</b>	Change the data type of a column
<b>Drop column</b>	Delete one or more columns
<b>Select column</b>	Choose one or more columns to keep, and delete the rest
<b>Rename column</b>	Rename a column
<b>Drop missing values</b>	Remove rows with missing values
<b>Drop duplicate rows</b>	Drop all rows that have duplicate values in one or more columns
<b>Fill missing values</b>	Replace cells with missing values with a new value
<b>Find and replace</b>	Replace cells with an exact matching pattern
<b>Group by column and aggregate</b>	Group by column values and aggregate results
<b>Strip whitespace</b>	Remove whitespace from the beginning and end of text
<b>Split text</b>	Split a column into several columns based on a user-defined delimiter
<b>Convert text to lowercase</b>	Convert text to lowercase
<b>Convert text to uppercase</b>	Convert text to UPPERCASE
<b>Scale min/max values</b>	Scale a numerical column between a minimum and maximum value
<b>Flash Fill</b>	Automatically create a new column based on examples derived from an existing column

## Saving and exporting code

The toolbar above the Data Wrangler display grid provides options to save the code that the tool generates. You can copy the code to the clipboard or export it to the

notebook as a function. Exporting the code closes Data Wrangler and adds the new function to a code cell in the notebook. You can also download the cleaned DataFrame, reflected in the updated Data Wrangler display grid, as a csv file.

## Tip

The code generated by Data Wrangler won't be applied until you manually run the new cell, and it will not overwrite your original DataFrame.

**Operations**

- Find and replace (4)
- Format (7)
- Formulas (3)
- Schema (4)
- Sort and filter (2)
- Group by and aggregate
- New column by example
- Scale min/max values

**Cleaning steps**

- Loaded variable 'df' from kernel state
- Scale min/max values 'Age'
- New operation**

**Summary**

Age	Data type	float64
Rows	891	
Unique values	88	
Missing values	177	
Mean	0.368	
Standard deviation	0.163	
Minimum	0	
25th percentile	0.248	
Median	0.347	
75th percentile	0.472	
Maximum	1	

```

1 def clean_data(df):
2     # Scale column 'Age' between 0 and 1
3     new_min, new_max = 0, 1
4     old_min, old_max = df['Age'].min(), df['Age'].max()
5     df['Age'] = (df['Age'] - old_min) / (old_max - old_min) * (new_max - new_min) + new_min
6     return df
7
8 df_clean = clean_data(df.copy())
9 df_clean.head()

```

2 set - Command executed in 360 ms by Eren Orbey on 1:02:25 PM, 4/21/23

PassengerId	Survived	Pclass	Name	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	
0	1	0	3	Braund, Mr. Owen Harris	0.271174	1	0	A/5 21171	7.2500	Nan	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th... <td>0.472299</td> <td>1</td> <td>0</td> <td>PC 17599</td> <td>71.2833</td> <td>C85</td> <td>C</td>	0.472299	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	0.321438	0	0	STON/O2.3101282	7.9250	Nan	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	0.434531	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	0.434531	0	0	373450	8.0500	Nan	S

## Next steps

- To try out Data Wrangler in VS Code, see [Data Wrangler in VS Code](#).

# Train machine learning models

Article • 05/23/2023

## ⓘ Important

Microsoft Fabric is currently in PREVIEW. This information relates to a prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, expressed or implied, with respect to the information provided here.

Apache Spark in Microsoft Fabric enables machine learning with big data, providing the ability to obtain valuable insight from large amounts of structured, unstructured, and fast-moving data. There are several options when training machine learning models using Apache Spark in Microsoft Fabric: Apache Spark MLLib, SynapseML, and various other open-source libraries.

## Apache SparkML and MLLib

Apache Spark in Microsoft Fabric provides a unified, open-source, parallel data processing framework supporting in-memory processing to boost big data analytics. The Spark processing engine is built for speed, ease of use, and sophisticated analytics. Spark's in-memory distributed computation capabilities make it a good choice for the iterative algorithms used in machine learning and graph computations.

There are two scalable machine learning libraries that bring algorithmic modeling capabilities to this distributed environment: MLLib and SparkML. MLLib contains the original API built on top of RDDs. SparkML is a newer package that provides a higher-level API built on top of DataFrames for constructing ML pipelines. SparkML doesn't yet support all of the features of MLLib, but is replacing MLLib as Spark's standard machine learning library.

## ⓘ Note

You can learn more about creating a SparkML model in the article [Train models with Apache Spark MLLib](#).

## Popular libraries

The Microsoft Fabric runtime for Apache Spark includes several popular, open-source packages for training machine learning models. These libraries provide reusable code that you may want to include in your programs or projects. Some of the relevant machine learning libraries that are included by default include:

- [Scikit-learn](#) is one of the most popular single-node machine learning libraries for classical ML algorithms. Scikit-learn supports most of the supervised and unsupervised learning algorithms and can also be used for data-mining and data-analysis.
- [XGBoost](#) is a popular machine learning library that contains optimized algorithms for training decision trees and random forests.
- [PyTorch](#) & [Tensorflow](#) are powerful Python deep learning libraries. You can use these libraries to build single-machine models by setting the number of executors on your pool to zero. Even though Apache Spark is not functional under this configuration, it is a simple and cost-effective way to create single-machine models.

## SynapseML

[SynapseML](#) (previously known as MMLSpark), is an open-source library that simplifies the creation of massively scalable machine learning (ML) pipelines. This library is designed to make data scientists more productive on Spark, increase the rate of experimentation, and leverage cutting-edge machine learning techniques, including deep learning, on large datasets.

SynapseML provides a layer on top of SparkML's low-level APIs when building scalable ML models, such as indexing strings, coercing data into a layout expected by machine learning algorithms, and assembling feature vectors. The SynapseML library simplifies these and other common tasks for building models in PySpark.

## Next steps

This article provides an overview of the various options to train machine learning models within Apache Spark in Microsoft Fabric. You can learn more about model training by following the tutorial below:

- Use AI samples to build machine learning models: [Use AI samples](#)
- Track machine learning runs using Experiments: [Machine learning experiments](#)

# Build a machine learning model with Apache Spark MLlib

Article • 05/23/2023

## ⓘ Important

Microsoft Fabric is currently in PREVIEW. This information relates to a prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, expressed or implied, with respect to the information provided here.

In this article, you'll learn how to use Apache Spark [MLlib](#) to create a machine learning application that does simple predictive analysis on an Azure open dataset. Spark provides built-in machine learning libraries. This example uses *classification* through logistic regression.

SparkML and MLlib are core Spark libraries that provide many utilities that are useful for machine learning tasks, including utilities that are suitable for:

- Classification
- Regression
- Clustering
- Topic modeling
- Singular value decomposition (SVD) and principal component analysis (PCA)
- Hypothesis testing and calculating sample statistics

## Understand classification and logistic regression

*Classification*, a popular machine learning task, is the process of sorting input data into categories. It's the job of a classification algorithm to figure out how to assign *labels* to input data that you provide. For example, you can think of a machine learning algorithm that accepts stock information as input and divide the stock into two categories: stocks that you should sell and stocks that you should keep.

*Logistic regression* is an algorithm that you can use for classification. Spark's logistic regression API is useful for *binary classification*, or classifying input data into one of two groups. For more information about logistic regression, see [Wikipedia](#).

In summary, the process of logistic regression produces a *logistic function* that you can use to predict the probability that an input vector belongs in one group or the other.

## Predictive analysis example on NYC taxi data

To get started, install `azureml-opendatasets`. The data is available through [Azure Open Datasets](#). This subset of the dataset contains information about yellow taxi trips, including the start and end time and locations, the cost, and other attributes.

Python

```
%pip install azureml-opendatasets
```

In the rest of this article, we'll use Apache Spark to perform some analysis on the NYC taxi-trip tip data and then develop a model to predict whether a particular trip includes a tip or not.

## Create an Apache Spark machine learning model

1. Create a PySpark notebook. For instructions, see [Create a notebook](#).
2. Import the types required for this notebook.

Python

```
import matplotlib.pyplot as plt
from datetime import datetime
from dateutil import parser
from pyspark.sql.functions import unix_timestamp, date_format, col,
when
from pyspark.ml import Pipeline
from pyspark.ml import PipelineModel
from pyspark.ml.feature import RFormula
from pyspark.ml.feature import OneHotEncoder, StringIndexer,
VectorIndexer
from pyspark.ml.classification import LogisticRegression
from pyspark.mllib.evaluation import BinaryClassificationMetrics
from pyspark.ml.evaluation import BinaryClassificationEvaluator
```

3. We will use [MLflow](#) to track our machine learning experiments and corresponding runs. If Microsoft Fabric Autologging is enabled, the corresponding metrics and parameters are automatically captured.

```
Python
```

```
import mlflow
```

## Construct the input DataFrame

In this example, we will load the data into a Pandas dataframe and then convert it into an Apache Spark dataframe. Using this format, we can apply other Apache Spark operations to clean and filter the dataset.

1. Run the following lines to create a Spark DataFrame by pasting the code into a new cell. This step retrieves the data via the Open Datasets API. We can filter this data down to look at a specific window of data. The following code example uses `start_date` and `end_date` to apply a filter that returns a single month of data.

```
Python
```

```
from azureml.opendatasets import NycTlcYellow

end_date = parser.parse('2018-06-06')
start_date = parser.parse('2018-05-01')
nyc_tlc = NycTlcYellow(start_date=start_date, end_date=end_date)
nyc_tlc_pd = nyc_tlc.to_pandas_dataframe()

nyc_tlc_df = spark.createDataFrame(nyc_tlc_pd).repartition(20)
```

2. The following code reduces the dataset to about 10,000 rows. To speed up the development and training, we will sample down our dataset for now.

```
Python
```

```
# To make development easier, faster, and less expensive, sample down
# for now
sampled_taxi_df = nyc_tlc_df.sample(True, 0.001, seed=1234)
```

3. Next, we want to take a look at our data using the built-in `display()` command. This allows us to easily view a sample of the data or explore the trends in the data graphically.

```
Python
```

```
#sampled_taxi_df.show(10)
display(sampled_taxi_df.limit(10))
```

# Prepare the data

Data preparation is a crucial step in the machine learning process. It involves cleaning, transforming, and organizing raw data to make it suitable for analysis and modeling. In the following code, you perform several data preparation steps:

- Remove outliers and incorrect values by filtering the dataset
- Remove columns which are not needed for model training
- Create new columns from the raw data
- Generate a label to determine if there will be a tip or not for the given Taxi trip

Python

```
taxi_df = sampled_taxi_df.select('totalAmount', 'fareAmount', 'tipAmount',
'paymentType', 'rateCodeId', 'passengerCount' \
, 'tripDistance', 'tpepPickupDateTime',
'tpepDropoffDateTime' \
, date_format('tpepPickupDateTime',
'hh').alias('pickupHour') \
, date_format('tpepPickupDateTime',
'EEEE').alias('weekdayString') \
, (unix_timestamp(col('tpepDropoffDateTime')) - \
unix_timestamp(col('tpepPickupDateTime'))).alias('tripTimeSecs') \
, (when(col('tipAmount') > 0,
1).otherwise(0)).alias('tipped') \
) \
.filter((sampled_taxi_df.passengerCount > 0) &
(sampled_taxi_df.passengerCount < 8) \
& (sampled_taxi_df.tipAmount >= 0) &
(sampled_taxi_df.tipAmount <= 25) \
& (sampled_taxi_df.fareAmount >= 1) &
(sampled_taxi_df.fareAmount <= 250) \
& (sampled_taxi_df.tipAmount <
sampled_taxi_df.fareAmount) \
& (sampled_taxi_df.tripDistance > 0) &
(sampled_taxi_df.tripDistance <= 100) \
& (sampled_taxi_df.rateCodeId <= 5) \
& (sampled_taxi_df.paymentType.isin({"1", "2"})) \
)
```

We will then make a second pass over the data to add the final features.

Python

```
taxi_featurised_df = taxi_df.select('totalAmount', 'fareAmount',
'tipAmount', 'paymentType', 'passengerCount' \
, 'tripDistance',
'weekdayString', 'pickupHour','tripTimeSecs','tipped' \
, when((taxi_df.pickupHour \
<= 6) | (taxi_df.pickupHour >= 20),"Night")\
```

```

    .when((taxi_df.pickupHour >=
7) & (taxi_df.pickupHour <= 10), "AMRush")\
        .when((taxi_df.pickupHour >=
11) & (taxi_df.pickupHour <= 15), "Afternoon")\
            .when((taxi_df.pickupHour >=
16) & (taxi_df.pickupHour <= 19), "PMRush")\
                .otherwise(0).alias('trafficTimeBins')
            )\

.filter((taxi_df.tripTimeSecs >= 30)
& (taxi_df.tripTimeSecs <= 7200))

```

## Create a logistic regression model

The final task is to convert the labeled data into a format that can be analyzed through logistic regression. The input to a logistic regression algorithm needs to be a set of *label/feature vector pairs*, where the *feature vector* is a vector of numbers that represent the input point.

So, you need to convert the categorical columns into numbers. Specifically, you need to convert the `trafficTimeBins` and `weekdayString` columns into integer representations. There are multiple approaches to performing the conversion. The following example takes the `OneHotEncoder` approach.

Python

```

# Because the sample uses an algorithm that works only with numeric
features, convert them so they can be consumed
si1 = StringIndexer(inputCol="trafficTimeBins",
outputCol="trafficTimeBinsIndex")
en1 = OneHotEncoder(dropLast=False, inputCol="trafficTimeBinsIndex",
outputCol="trafficTimeBinsVec")
si2 = StringIndexer(inputCol="weekdayString", outputCol="weekdayIndex")
en2 = OneHotEncoder(dropLast=False, inputCol="weekdayIndex",
outputCol="weekdayVec")

# Create a new DataFrame that has had the encodings applied
encoded_final_df = Pipeline(stages=[si1, en1, si2,
en2]).fit(taxi_featurised_df).transform(taxi_featurised_df)

```

This action results in a new DataFrame with all columns in the right format to train a model.

## Train a logistic regression model

The first task is to split the dataset into a training set and a testing or validation set.

Python

```
# Decide on the split between training and testing data from the DataFrame
trainingFraction = 0.7
testingFraction = (1-trainingFraction)
seed = 1234

# Split the DataFrame into test and training DataFrames
train_data_df, test_data_df =
    encoded_final_df.randomSplit([trainingFraction, testingFraction], seed=seed)
```

Now that there are two DataFrames, the next task is to create the model formula and run it against the training DataFrame. Then you can validate against the testing DataFrame. Experiment with different versions of the model formula to see the impact of different combinations.

Python

```
## Create a new logistic regression object for the model
logReg = LogisticRegression(maxIter=10, regParam=0.3, labelCol = 'tipped')

## The formula for the model
classFormula = RFormula(formula="tipped ~ pickupHour + weekdayVec +
passengerCount + tripTimeSecs + tripDistance + fareAmount + paymentType+
trafficTimeBinsVec")

## Undertake training and create a logistic regression model
lrModel = Pipeline(stages=[classFormula, logReg]).fit(train_data_df)

## Predict tip 1/0 (yes/no) on the test dataset; evaluation using area under
ROC
predictions = lrModel.transform(test_data_df)
predictionAndLabels = predictions.select("label","prediction").rdd
metrics = BinaryClassificationMetrics(predictionAndLabels)
print("Area under ROC = %s" % metrics.areaUnderROC)
```

The output from this cell is:

shell

```
Area under ROC = 0.9749430523917996
```

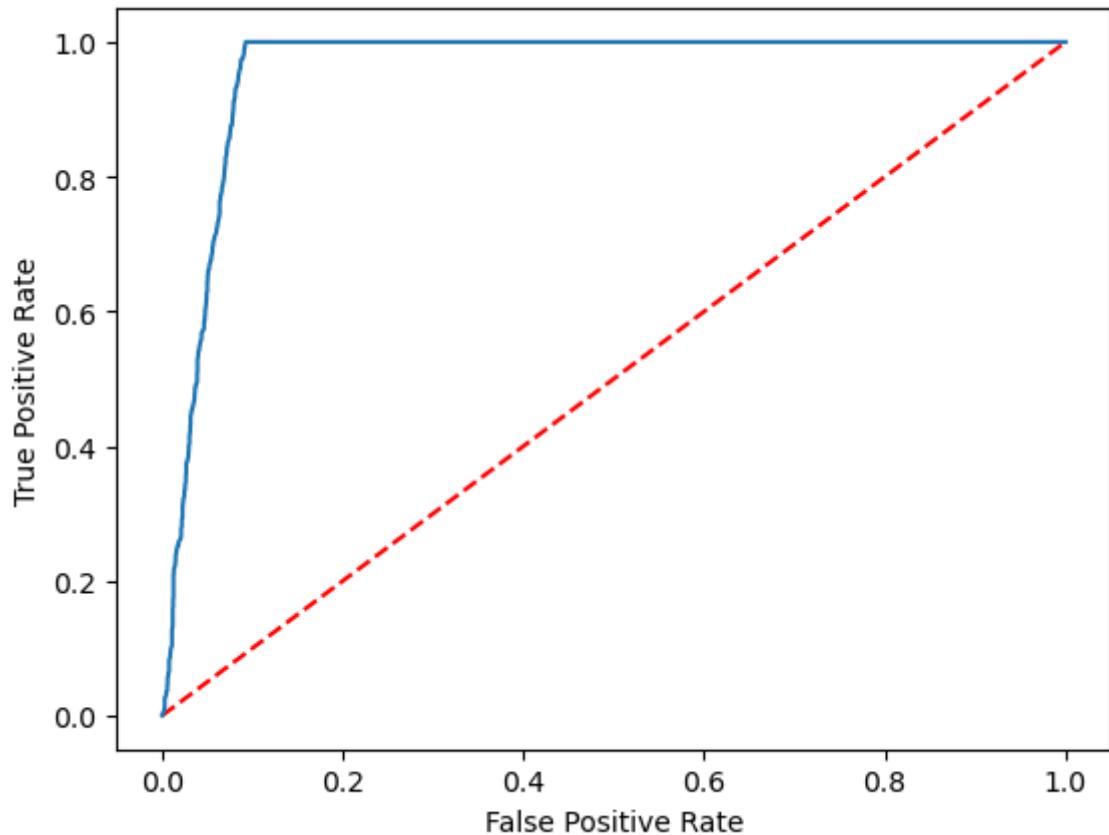
## Create a visual representation of the prediction

You can now construct a final visualization to interpret the model results. A [ROC curve](#) is one way to review the result.

Python

```
## Plot the ROC curve; no need for pandas, because this uses the
modelSummary object
modelSummary = lrModel.stages[-1].summary

plt.plot([0, 1], [0, 1], 'r--')
plt.plot(modelSummary.roc.select('FPR').collect(),
         modelSummary.roc.select('TPR').collect())
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.show()
```



## Next steps

- Use AI samples to build machine learning models: [Use AI samples](#)
- Track machine learning runs using Experiments: [Machine learning experiments](#)

# How to train models with scikit-learn in Microsoft Fabric

Article • 05/23/2023

Scikit-learn ([scikit-learn.org](https://scikit-learn.org)) is a popular, open-source machine learning framework. It's frequently used for supervised and unsupervised learning. It also provides various tools for model fitting, data preprocessing, model selection, model evaluation, and more.

## Important

Microsoft Fabric is currently in PREVIEW. This information relates to a prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, expressed or implied, with respect to the information provided here.

In this section, we'll go through an example of how you can train and track the iterations of your Scikit-Learn model.

## Install scikit-learn

To get started with scikit-learn, you must ensure that it's installed within your notebook. You can install or upgrade the version of scikit-learn on your environment using the following command:

shell

```
%pip install scikit-learn
```

Next, we'll create a machine learning experiment using the MLflow API. The MLflow `set_experiment()` API will create a new machine learning experiment if it doesn't already exist.

Python

```
import mlflow

mlflow.set_experiment("sample-sklearn")
```

# Train a scikit-learn model

After the experiment has been created, we'll create a sample dataset and create a logistic regression model. We'll also start a MLflow run and track the metrics, parameters, and final logistic regression model. Once we've generated the final model, we'll also save the resulting model for additional tracking.

Python

```
import mlflow.sklearn
import numpy as np
from sklearn.linear_model import LogisticRegression
from mlflow.models.signature import infer_signature

with mlflow.start_run() as run:

    lr = LogisticRegression()
    X = np.array([-2, -1, 0, 1, 2, 1]).reshape(-1, 1)
    y = np.array([0, 0, 1, 1, 1, 0])
    lr.fit(X, y)
    score = lr.score(X, y)
    signature = infer_signature(X, y)

    print("log_metric.")
    mlflow.log_metric("score", score)

    print("log_params.")
    mlflow.log_param("alpha", "alpha")

    print("log_model.")
    mlflow.sklearn.log_model(lr, "sklearn-model", signature=signature)
    print("Model saved in run_id=%s" % run.info.run_id)

    print("register_model.")
    mlflow.register_model(
        "runs:/{}{}".format(run.info.run_id), "sample-sklearn"
    )
    print("All done")
```

## Load and evaluate the model on a sample dataset

Once the model is saved, it can also be loaded for inferencing. To do this, we'll load the model and run the inference on a sample dataset.

Python

```
# Inference with loading the logged model
from synapse.ml.predict import MLflowTransformer

spark.conf.set("spark.synapse.ml.predict.enabled", "true")

model = MLflowTransformer(
    inputCols=["x"],
    outputCol="prediction",
    modelName="sample-sklearn",
    modelVersion=1,
)

test_spark = spark.createDataFrame(
    data=np.array([-2, -1, 0, 1, 2, 1]).reshape(-1, 1).tolist(), schema=
    ["x"]
)

batch_predictions = model.transform(test_spark)

batch_predictions.show()
```

## Next steps

- Learn about [machine learning models](#).
- Learn about [machine learning experiments](#).

# How to train models with SynapseML

Article • 05/23/2023

## ⓘ Important

Microsoft Fabric is currently in PREVIEW. This information relates to a prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, expressed or implied, with respect to the information provided here.

[SynapseML](#) is an ecosystem of tools aimed towards expanding the distributed computing framework Apache Spark in several new directions. SynapseML adds many deep learning and data science tools to the Spark ecosystem, including seamless integration of Spark Machine Learning pipelines with Microsoft Cognitive Toolkit (CNTK), LightGBM and OpenCV. These tools enable powerful and highly scalable predictive and analytical models for many types of datasources.

In this section, we'll go through an example of how you can train your SynapseML model.

## Importing Packages

Start by importing numpy and pandas as they'll be used in the sample.

Python

```
import numpy as np  
import pandas as pd
```

## Reading in Data

In a typical Spark application, you'll likely work with huge datasets stored on a distributed file system, such as HDFS. However, to keep this tutorial simple and quick, we'll copy over a small dataset from a URL. We then read this data into memory using Pandas CSV reader, and distribute the data as a Spark DataFrame. Finally, we show the first 5 rows of the dataset.

Python

```
dataFile = "AdultCensusIncome.csv"  
import os, urllib
```

```
if not os.path.isfile(dataFile):
    urllib.request.urlretrieve("https://mmlspark.azureedge.net/datasets/" + dataFile, dataFile)
data = spark.createDataFrame(pd.read_csv(dataFile, dtype={"hours-per-week": np.float64}))
data.show(5)
```

## Selecting Features and Splitting Data to Train and Test Sets

Next, select some features to use in our model. You can try out different features, but you should include "income" as it is the label column the model is trying to predict. We then split the data into a `train` and `test` sets.

Python

```
data = data.select(["education", "marital-status", "hours-per-week", "income"])
train, test = data.randomSplit([0.75, 0.25], seed=123)
```

## Training a Model

To train the classifier model, we use the `synapse.ml.TrainClassifier` class. It takes in training data and a base SparkML classifier, maps the data into the format expected by the base classifier algorithm, and fits a model.

Python

```
from synapse.ml.train import TrainClassifier
from pyspark.ml.classification import LogisticRegression
model = TrainClassifier(model=LogisticRegression(), labelCol="income").fit(train)
```

`TrainClassifier` implicitly handles string-valued columns and binarizes the label column.

## Scoring and Evaluating the Model

Finally, let's score the model against the test set, and use the `synapse.ml.ComputeModelStatistics` class to compute metrics such as accuracy, AUC, precision, and recall from the scored data.

Python

```
from synapse.ml.train import ComputeModelStatistics
prediction = model.transform(test)
metrics = ComputeModelStatistics().transform(prediction)
metrics.select('accuracy').show()
```

And that's it! you've build your first machine learning model using the SynapseML package. For help on SynapseML classes and methods, you can use Python's help() function.

Python

```
help(synapse.ml.train.TrainClassifier)
```

# Machine learning experiments in Microsoft Fabric

Article • 05/23/2023

A machine learning *experiment* is the primary unit of organization and control for all related machine learning runs. A *run* corresponds to a single execution of model code. In [MLflow](#), tracking is based on experiments and runs.

## ⓘ Important

Microsoft Fabric is currently in PREVIEW. This information relates to a prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, expressed or implied, with respect to the information provided here.

Machine learning experiments allow data scientists to log parameters, code versions, metrics, and output files when running their machine learning code. Experiments also let you visualize, search for, and compare runs, as well as download run files and metadata for analysis in other tools.

In this article, you'll learn more about how data scientists can interact with and use machine learning experiments to organize their development process and to track multiple runs.

## Prerequisites

- A Power BI Premium subscription. If you don't have one, see [How to purchase Power BI Premium](#).
- A Power BI Workspace with assigned premium capacity.

## Create an experiment

You can create a machine learning experiment directly from the Data science home page in the Power BI user interface (UI) or by writing code that uses the MLflow API.

### Create an experiment using the UI

To create a machine learning experiment from the UI:

1. Create a new data science workspace or select an existing one.

2. Select **Experiment** from the "New" section.
3. Provide an experiment name and select **Create**. This action creates an empty experiment within your workspace.



Once you've created the experiment, you can start adding runs to track run metrics and parameters.

## Create an experiment using the MLflow API

You can also create a machine learning experiment directly from your authoring experience using the `mlflow.create_experiment()` or `mlflow.set_experiment()` APIs. In the following code, replace `<EXPERIMENT_NAME>` with your experiment's name.

Python

```
import mlflow

# This will create a new experiment with the provided name.
mlflow.create_experiment("<EXPERIMENT_NAME>")

# This will set the given experiment as the active experiment.
# If an experiment with this name does not exist, a new experiment with this
# name is created.
mlflow.set_experiment("<EXPERIMENT_NAME>")
```

## Manage runs within an experiment

A machine learning experiment contains a collection of runs for simplified tracking and comparison. Within an experiment, a data scientist can navigate across various runs and explore the underlying parameters and metrics. Data scientists can also compare runs within a machine learning experiment to identify which subset of parameters yield a desired model performance.

### Track runs

A machine learning run corresponds to a single execution of model code.

The screenshot shows the Azure Machine Learning studio interface. At the top, there's a navigation bar with 'Home' selected. Below it, a sidebar lists 'aisample-recommendation' and 'sample\_run'. The main area has a 'Properties' section with fields like Run name (sample\_run), Start date (10/19/2022 11:41 PM), Duration (—), Status (Queued), Run ID (370414...), Created by (U Jiang), Source (bug-dash-aisample...), Experiment name (aisample-recommendation-ais...), and Registered version (aisample-recommendation-ais... Version 1). To the right are two cards: 'Save as model' (Save button) and 'Compare runs' (View run list button). A large table below shows detailed metrics and hyperparameters. A search icon is in the bottom right.

Name	Value
Run metrics (4)	
Explained variance	3.598376076102218
RMSE	0.6542808703458795
R2	0.5629590403149667
MAE	0.10599706055688077
Run hyperparameters (5)	
num_epochs	1
rank_size_list	[64, 128]
reg_param_list	[0.01, 0.1]
model_tuning_method	TrainValidationSplit
DATA_FOLDER	Files/book-recommendation/
Model training and test size (0)	
Input schema (0)	
Output schema (0)	

Each run includes the following information:

- **Source:** Name of the notebook that created the run.
- **Registered Version:** Indicates if the run was saved as a machine learning model.
- **Start date:** Start time of the run.
- **Status:** Progress of the run.
- **Hyperparameters:** Hyperparameters saved as key-value pairs. Both keys and values are strings.
- **Metrics:** Run metrics saved as key-value pairs. The value is numeric.
- **Output files:** Output files in any format. For example, you can record images, environment, models, and data files.

## Compare and filter runs

To compare and evaluate the quality of your machine learning runs, you can compare the parameters, metrics, and metadata between selected runs within an experiment.

### Visually compare runs

You can visually compare and filter runs within an existing experiment. This allows you to easily navigate between multiple runs and sort across them.

The screenshot shows the MLflow interface with the 'Run list' view selected. The main area displays a table of 10 runs, with the first two highlighted: 'run\_diabetes\_n21' and 'run\_diabetes\_n20'. The table includes columns for 'Properties' (Run name, Start time, Duration, Status, Created by), 'Model version metrics' (Model type, Accuracy, F-1 score, Training time), and 'Model hyperparameters' (Alpha, Max tree depth, n\_estimators). Below the table, there's a 'Metric comparison' section showing a bar chart for 'Accuracy' between the two selected runs.

To compare runs:

1. Select an existing machine learning experiment that contains multiple runs.
2. Select the **View** tab and then go to the **Run list** view. Alternatively, you could select the option to **View run list** directly from the **Run details** view.
3. Customize the columns within the table by expanding the **Customize columns** pane. Here, you can select the properties, metrics, and hyperparameters that you would like to see.
4. Expand the **Filter** pane to narrow your results based on certain selected criteria.
5. Select multiple runs to compare their results in the metrics comparison pane. From this pane, you can customize the charts by changing the chart title, visualization type, X-axis, Y-axis, and more.

## Compare runs using the MLflow API

Data scientists can also use MLflow to query and search among runs within an experiment. You can explore more MLflow APIs for searching, filtering, and comparing runs by visiting the [MLflow documentation](#).

## Get all runs

You can use the MLflow search API `mlflow.search_runs()` to get all runs in an experiment by replacing `<EXPERIMENT_NAME>` with your experiment name or `<EXPERIMENT_ID>` with your experiment ID in the following code:

Python

```
import mlflow

# Get runs by experiment name:
mlflow.search_runs(experiment_names=[ "<EXPERIMENT_NAME>" ])

# Get runs by experiment ID:
mlflow.search_runs(experiment_ids=[ "<EXPERIMENT_ID>" ])
```

### 💡 Tip

You can search across multiple experiments by providing a list of experiment IDs to the `experiment_ids` parameter. Similarly, providing a list of experiment names to the `experiment_names` parameter will allow MLflow to search across multiple experiments. This can be useful if you want to compare across runs within different experiments.

## Order and limit runs

Use the `max_results` parameter from `search_runs` to limit the number of runs returned. The `order_by` parameter allows you to list the columns to order by and can contain an optional `DESC` or `ASC` value. For instance, the following example returns the last run of an experiment.

Python

```
mlflow.search_runs(experiment_ids=[ "1234-5678-90AB-CDEFG" ], max_results=1,
order_by=[ "start_time DESC" ])
```

## Save run as a model

Once a run yields the desired result, you can save the run as a model for enhanced model tracking and for model deployment.

The screenshot shows the MLflow Experiment UI for the experiment named 'experiment\_diabetes'. The left sidebar lists 25 runs, each with a name, timestamp, and status. The right side displays properties for the selected run ('Run dia... n21') and provides options to save to the model registry or compare runs. A modal window titled 'Create model' is open, showing two selection methods: 'Create a new model' (selected) and 'Select an existing model', with a 'Create' button at the bottom.

## Next Steps

- Learn about MLflow Experiment APIs ↗
- Track and manage machine learning models

# Machine learning model in Microsoft Fabric

Article • 05/23/2023

A machine learning model is a file trained to recognize certain types of patterns. You train a model over a set of data, and you provide it with an algorithm that uses to reason over and learn from that data set. After you train the model, you can use it to reason over data that it never saw before, and make predictions about that data.

## Important

Microsoft Fabric is currently in PREVIEW. This information relates to a prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, expressed or implied, with respect to the information provided here.

In [MLflow](#), a machine learning model can include multiple model versions. Here, each version can represent a model iteration. In this article, you learn how to interact with machine learning models to track and compare model versions.

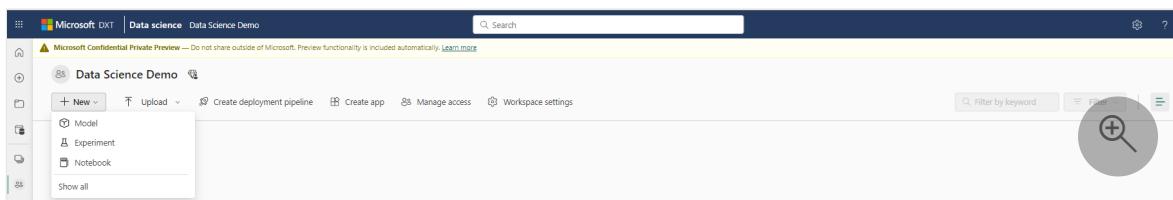
## Create a model

In MLflow, machine learning models include a standard packaging format. This format allows use of those models in various downstream tools, including batch inferencing on Apache Spark. The format defines a convention to save a model in different “flavors” that different downstream tools can understand.

The user experience can directly create a machine learning model from the user experience. The MLflow API can also directly create a machine learning model.

To create a machine learning model from the user experience, you can:

1. Create a new data science workspace, or select an existing data science workspace.
2. From the **+ New** dropdown, select **Model**. This creates an empty model in your data science workspace.



3. After model creation, you can start adding model versions to track run metrics and parameters. Register or save experiment runs to an existing model.

You can also create a machine learning experiment directly from your authoring experience with the `mlflow.register_model()` API. If a registered model with the given name doesn't exist, the API creates it automatically.

```
Python

import mlflow

model_uri = "runs:/{}//model-uri-name".format(run.info.run_id)
mv = mlflow.register_model(model_uri, "model-name")

print("Name: {}".format(mv.name))
print("Version: {}".format(mv.version))
```

## Manage versions within a model

A machine learning model contains a collection of model versions for simplified tracking and comparison. Within a model, a data scientist can navigate across various model versions to explore the underlying parameters and metrics. Data scientists can also make comparisons across model versions to identify whether or not newer models might yield better results.

## Track models

A machine learning model version represents an individual model that has been registered for tracking.

The screenshot shows the Microsoft Model Registry interface. At the top, there's a navigation bar with 'Home' and 'View' options. Below the navigation is a search bar and a message about Microsoft Confidential Private Preview. The main content area displays a model named 'aisample-fraud-lightgbm'. It shows two versions: 'Version 2' (10/20/2022) and 'Version 1' (9/25/2022). A 'Properties' section includes fields for Version name, Created time, Last modified, and Created by. To the right are buttons for 'Apply this version' and 'Compare model versions in a list'. Below these are sections for Model version metrics (AUROC, AUPRC), Model hyperparameters (DATA\_FILE), and Model training and test size. At the bottom right is a circular button with a plus sign and a magnifying glass icon.

Each model version includes the following information:

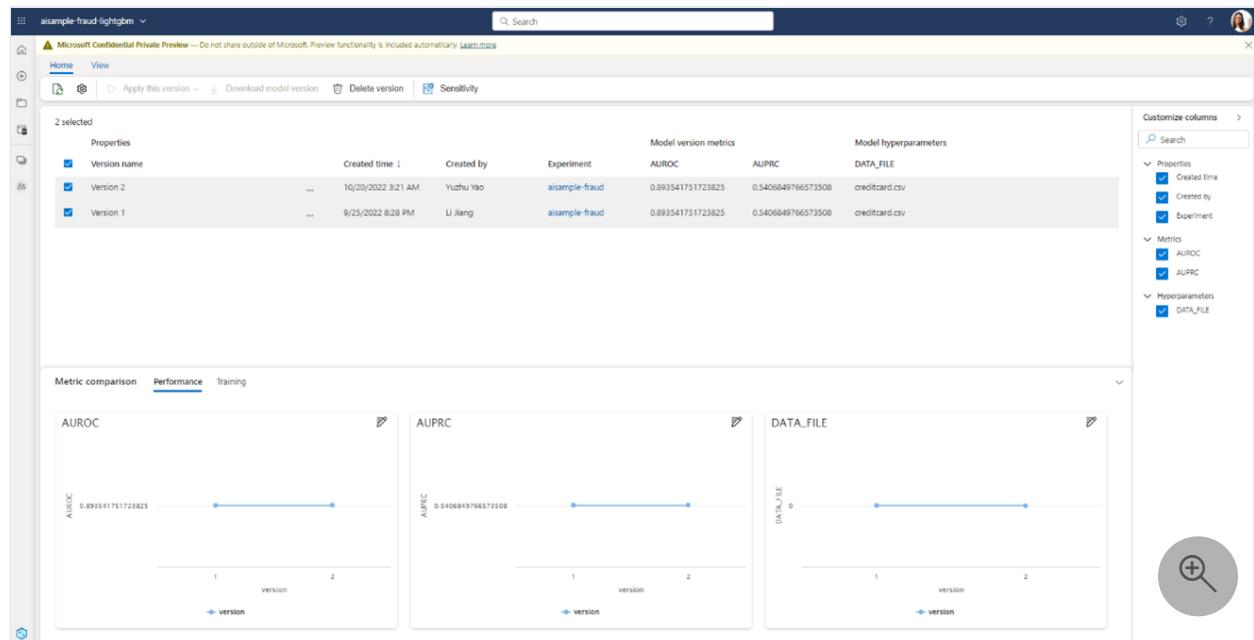
- **Time Created:** Date and time of model creation.
- **Run Name:** The identifier for the experiment runs used to create this specific model version.
- **Hyperparameters:** Hyperparameters are saved as key-value pairs. Both keys and values are strings.
- **Metrics:** Run metrics saved as key-value pairs. The value is numeric.
- **Model Schema/Signature:** A description of the model inputs and outputs.
- **Logged files:** Logged files in any format. For example, you can record images, environment, models, and data files.

## Compare and filter models

To compare and evaluate the quality of machine learning model versions, you can compare the parameters, metrics, and metadata between selected versions.

### Visually compare models

You can visually compare runs within an existing model. This allows easy navigation between, and sorts across, multiple versions.



To compare runs, you can:

1. Select an existing machine learning model that contains multiple versions.
2. Select the **View** tab, and then navigate to the **Model list** view. You can also select the option to **View model list** directly from the details view.

3. You can customize the columns within the table. Expand the **Customize columns** pane. From there, you can select the properties, metrics, and hyperparameters that you want to see.
4. Lastly, you can select multiple versions, to compare their results, in the metrics comparison pane. From this pane, you can customize the charts with changes to the chart title, visualization type, X-axis, Y-axis, and more.

## Compare models using the MLflow API

Data scientists can also use MLflow to search among multiple models saved within the workspace. Visit the [MLflow documentation](#) to explore other MLflow APIs for model interaction.

Python

```
from pprint import pprint

client = MlflowClient()
for rm in client.list_registered_models():
    pprint(dict(rm), indent=4)
```

## Apply the model

Once you train a model on a data set, you can apply that model to data it never saw to generate predictions. We call this model use technique **scoring** or **inferencing**. For more information about Microsoft Fabric model scoring, see the next section.

## Next steps

- [Learn about MLflow Experiment APIs](#)

# Autologging in Microsoft Fabric

Article • 05/23/2023

Synapse Data Science in Microsoft Fabric includes autologging, which significantly reduces the amount of code required to automatically log the parameters, metrics, and items of a machine learning model during training. This feature extends [MLflow autologging](#) capabilities and is deeply integrated into the Synapse Data Science in Microsoft Fabric experience. Using autologging, developers and data scientists can easily track and compare the performance of different models and experiments without the need for manual tracking.

## ⓘ Important

Microsoft Fabric is currently in PREVIEW. This information relates to a prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, expressed or implied, with respect to the information provided here.

## Configurations

Autologging works by automatically capturing the values of input parameters, output metrics, and output items of a machine learning model as it is being trained. This information is then logged to your Microsoft Fabric workspace, where it can be accessed and visualized using the MLflow APIs or the corresponding experiment & model items in your Microsoft Fabric workspace.

The default configuration for the notebook [mlflow.autolog\(\)](#) hook is:

Python

```
mlflow.autolog(  
    log_input_examples=False,  
    log_model_signatures=True,  
    log_models=True,  
    disable=False,  
    exclusive=True,  
    disable_for_unsupported_versions=True,  
    silent=True  
)
```

When you launch a Synapse Data Science notebook, Microsoft Fabric calls [mlflow.autolog\(\)](#) to instantly enable the tracking and load the corresponding

dependencies. As you train models in your notebook, this model information is automatically tracked with MLflow. This configuration is done automatically behind the scenes when you run `import mlflow`.

## Supported frameworks

Autologging supports a wide range of machine learning frameworks, including TensorFlow, PyTorch, Scikit-learn, and XGBoost. It can capture a variety of metrics, including accuracy, loss, and F1 score, as well as custom metrics defined by the user. To learn more about the framework specific properties that are captured, you can visit [MLflow documentation](#).

## Customize logging behavior

To customize the logging behavior, you can use the `mlflow.autolog()` configuration. This configuration provides the parameters to enable model logging, collect input samples, configure warnings, or even enable logging for user-specified content.

## Track additional content

You can update the autologging configuration to track additional metrics, parameters, files, and metadata with runs created with MLflow.

To do this:

1. Update the `mlflow.autolog()` call and set `exclusive=False`.

Python

```
mlflow.autolog(  
    log_input_examples=False,  
    log_model_signatures=True,  
    log_models=True,  
    disable=False,  
    exclusive=False, # Update this property to enable custom logging  
    disable_for_unsupported_versions=True,  
    silent=True  
)
```

2. Use the MLflow tracking APIs to log additional [parameters](#) and [metrics](#). This allows you to capture your custom metrics and parameters, while also allowing you to use autologging to capture additional properties.

For instance:

```
Python

import mlflow
mlflow.autolog(exclusive=False)

with mlflow.start_run():
    mlflow.log_param("parameter name", "example value")
    # <add model training code here>
    mlflow.log_metric("metric name", 20)
```

## Disable Microsoft Fabric autologging

Microsoft Fabric autologging can be disabled for a specific notebook session or across all notebooks using the workspace setting.

 **Note**

If autologging is disabled, users must manually log their own [parameters](#) and [metrics](#) using the MLflow APIs.

## Disable autologging for a notebook session

To disable Microsoft Fabric autologging in a notebook session, you can call `mlflow.autolog()` and set `disable=True`.

For example:

```
Python

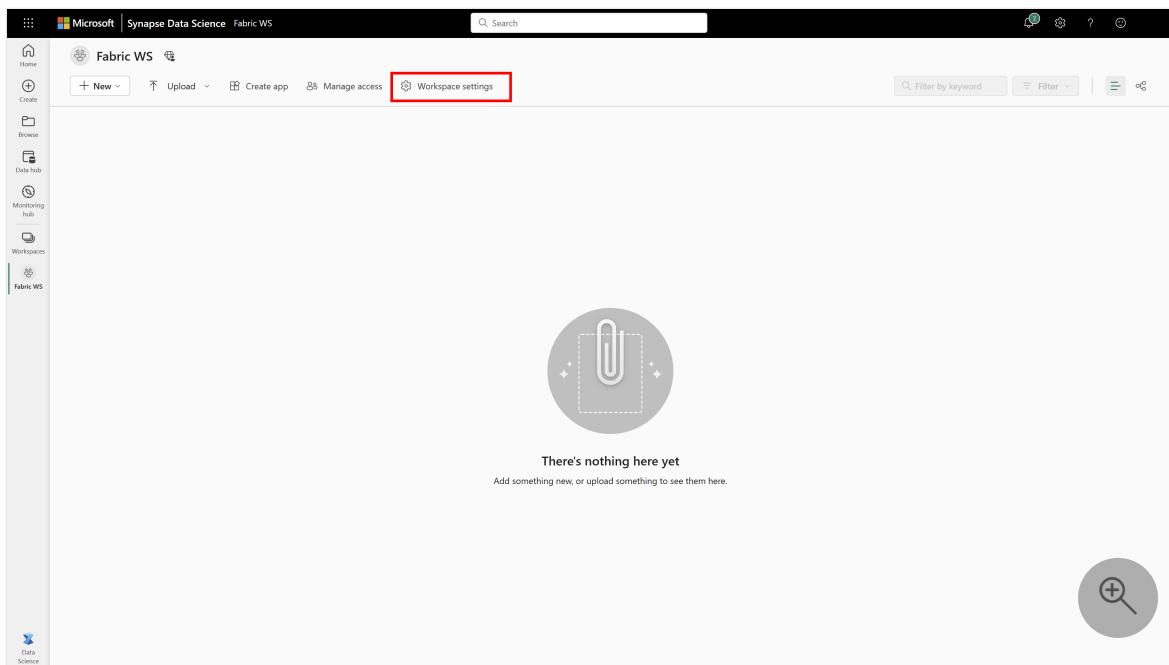
import mlflow
mlflow.autolog(disable=True)
```

## Disable autologging for the workspace

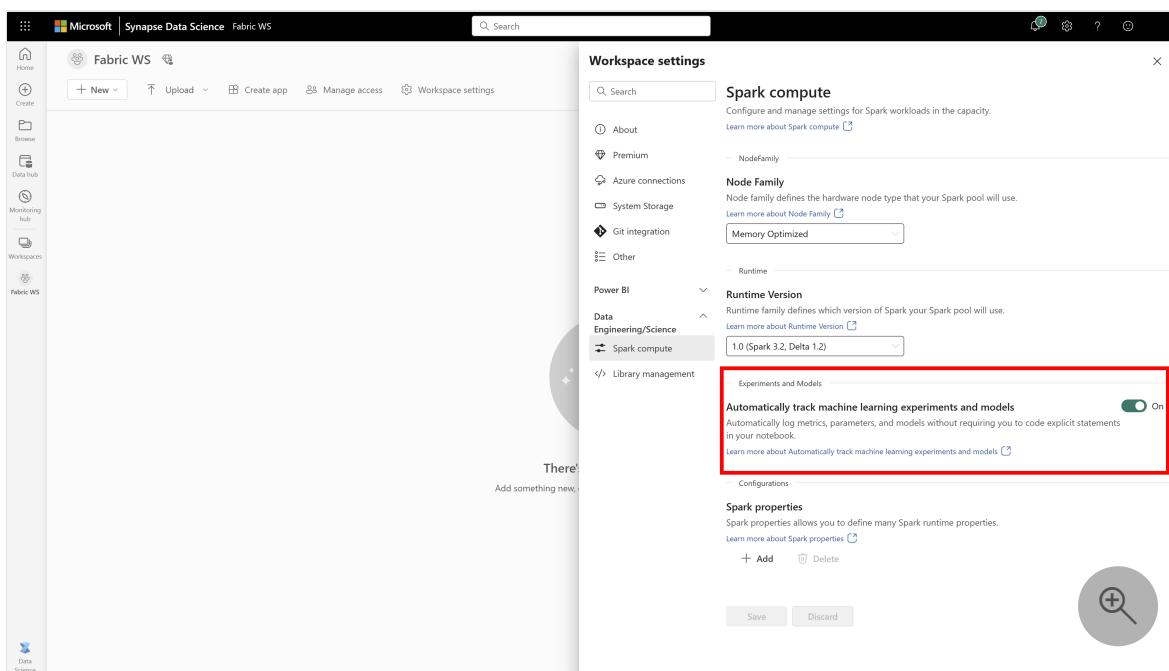
Workspace administrators can enable or disable Microsoft Fabric autologging for all sessions across their workspace.

To do this:

1. Navigate to your Synapse Data Science workspace and select **Workspace Settings**.



2. In the **Data Engineering/Science** tab, select **Spark compute**. Here, you will find the setting to enable or disable Synapse Data Science autologging.



## Next steps

- Train a Spark MLlib model with autologging: [Train with Spark MLlib](#)
- Learn about machine learning experiments in Microsoft Fabric: [Experiments](#)

# How to train models with PyTorch in Microsoft Fabric

Article • 05/23/2023

PyTorch [↗](#) is a machine learning framework based on the Torch library. It's frequently used for applications such as computer vision and natural language processing. In this article, we go through an example of how you train and track the iterations of your PyTorch model.

## Important

Microsoft Fabric is currently in PREVIEW. This information relates to a prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, expressed or implied, with respect to the information provided here.

## Install PyTorch

To get started with PyTorch, you must ensure that it's installed within your notebook. You can install or upgrade the version of PyTorch on your environment using the following command:

```
shell
```

```
%pip install torch
```

## Set up the machine learning experiment

Next, you create a machine learning experiment using the MLflow API. The MLflow `set_experiment()` API creates a new machine learning experiment if it doesn't already exist.

```
Python
```

```
import mlflow

mlflow.set_experiment("sample-pytorch")
```

# Train and evaluate a Pytorch model

After the experiment has been created, the code below loads the MNSIT dataset, generates our test and training datasets, and creates a training function.

Python

```
import os
import torch
import torch.nn as nn
from torch.autograd import Variable
import torchvision.datasets as dset
import torchvision.transforms as transforms
import torch.nn.functional as F
import torch.optim as optim

## load mnist dataset
root = "/tmp/mnist"
if not os.path.exists(root):
    os.mkdir(root)

trans = transforms.Compose(
    [transforms.ToTensor(), transforms.Normalize((0.5,), (1.0,))]
)
# if not exist, download mnist dataset
train_set = dset.MNIST(root=root, train=True, transform=trans,
download=True)
test_set = dset.MNIST(root=root, train=False, transform=trans,
download=True)

batch_size = 100

train_loader = torch.utils.data.DataLoader(
    dataset=train_set, batch_size=batch_size, shuffle=True
)
test_loader = torch.utils.data.DataLoader(
    dataset=test_set, batch_size=batch_size, shuffle=False
)

print("==>> total training batch number: {}".format(len(train_loader)))
print("==>> total testing batch number: {}".format(len(test_loader)))

## network

class LeNet(nn.Module):
    def __init__(self):
        super(LeNet, self).__init__()
        self.conv1 = nn.Conv2d(1, 20, 5, 1)
        self.conv2 = nn.Conv2d(20, 50, 5, 1)
        self.fc1 = nn.Linear(4 * 4 * 50, 500)
        self.fc2 = nn.Linear(500, 10)
```

```

def forward(self, x):
    x = F.relu(self.conv1(x))
    x = F.max_pool2d(x, 2, 2)
    x = F.relu(self.conv2(x))
    x = F.max_pool2d(x, 2, 2)
    x = x.view(-1, 4 * 4 * 50)
    x = F.relu(self.fc1(x))
    x = self.fc2(x)
    return x

def name(self):
    return "LeNet"

## training
model = LeNet()

optimizer = optim.SGD(model.parameters(), lr=0.01, momentum=0.9)

criterion = nn.CrossEntropyLoss()

for epoch in range(1):
    # trainning
    ave_loss = 0
    for batch_idx, (x, target) in enumerate(train_loader):
        optimizer.zero_grad()
        x, target = Variable(x), Variable(target)
        out = model(x)
        loss = criterion(out, target)
        ave_loss = (ave_loss * batch_idx + loss.item()) / (batch_idx + 1)
        loss.backward()
        optimizer.step()
        if (batch_idx + 1) % 100 == 0 or (batch_idx + 1) ==
len(train_loader):
            print(
                "=>> epoch: {}, batch index: {}, train loss: {:.6f}"
                .format(
                    epoch, batch_idx + 1, ave_loss
                )
            )
    # testing
    correct_cnt, total_cnt, ave_loss = 0, 0, 0
    for batch_idx, (x, target) in enumerate(test_loader):
        x, target = Variable(x, volatile=True), Variable(target,
volatile=True)
        out = model(x)
        loss = criterion(out, target)
        _, pred_label = torch.max(out.data, 1)
        total_cnt += x.data.size()[0]
        correct_cnt += (pred_label == target.data).sum()
        ave_loss = (ave_loss * batch_idx + loss.item()) / (batch_idx + 1)

        if (batch_idx + 1) % 100 == 0 or (batch_idx + 1) ==
len(test_loader):
            print(
                "=>> epoch: {}, batch index: {}, test loss: {:.6f}, acc: {:.2f}"
                .format(
                    epoch, batch_idx + 1, ave_loss, correct_cnt / total_cnt
                )
            )

```

```
{:.3f}".format(  
    epoch, batch_idx + 1, ave_loss, correct_cnt * 1.0 /  
    total_cnt  
)  
  
torch.save(model.state_dict(), model.name())
```

## Log model with MLflow

Now, you start an MLflow run and track the results within our machine learning experiment.

Python

```
with mlflow.start_run() as run:  
    print("log pytorch model:")  
    mlflow.pytorch.log_model(  
        model, "pytorch-model", registered_model_name="sample-pytorch")  
  
    model_uri = "runs:/{}//pytorch-model".format(run.info.run_id)  
    print("Model saved in run %s" % run.info.run_id)  
    print(f"Model URI: {model_uri}")
```

The code above creates a run with the specified parameters and logs the run within the sample-pytorch experiment. This snippet creates a new model called sample-pytorch.

## Load and evaluate the model

Once the model is saved, it can also be loaded for inferencing.

Python

```
# Inference with loading the logged model  
loaded_model = mlflow.pytorch.load_model(model_uri)  
print(type(loaded_model))  
  
correct_cnt, total_cnt, ave_loss = 0, 0, 0  
for batch_idx, (x, target) in enumerate(test_loader):  
    x, target = Variable(x, volatile=True), Variable(target, volatile=True)  
    out = loaded_model(x)  
    loss = criterion(out, target)  
    _, pred_label = torch.max(out.data, 1)  
    total_cnt += x.data.size()[0]  
    correct_cnt += (pred_label == target.data).sum()  
    ave_loss = (ave_loss * batch_idx + loss.item()) / (batch_idx + 1)
```

```
if (batch_idx + 1) % 100 == 0 or (batch_idx + 1) == len(test_loader):
    print(
        "==>> epoch: {}, batch index: {}, test loss: {:.6f}, acc:
{:.3f}".format(
            epoch, batch_idx + 1, ave_loss, correct_cnt * 1.0 /
total_cnt
        )
    )
```

## Next steps

- Learn about [machine learning models](#).
- Learn about [machine learning experiments](#).

# How direct lake mode works with Power BI reporting

Article • 05/23/2023

In Microsoft Fabric, when the user creates a lakehouse, the system also provisions the associated SQL endpoint and default dataset. The default dataset is a semantic model with metrics on top of lakehouse data. The dataset allows Power BI to load data for reporting.

## Important

Microsoft Fabric is currently in PREVIEW. This information relates to a prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, expressed or implied, with respect to the information provided here.

When a Power BI report shows an element that uses data, it requests it from the underlying dataset. Next the dataset accesses a lakehouse to retrieve data and return it to the Power BI report. For efficiency default dataset loads commonly requested data into the cache and refreshes it when needed.

Lakehouse applies V-order optimization to tables. This optimization enables quickly loading data into the dataset and having it ready for querying without any other sorting or transformations.

This approach gives unprecedented performance and the ability to instantly load large amounts of data for Power BI reporting.

The screenshot shows the Microsoft Fabric Data Explorer interface. On the left, there's a sidebar with navigation links like Home, Create, Browse, Data hub, Monitoring hub, Workspaces, ContosoLH, contosolh, pipeline14, and Data Engineering. The main area displays the 'Details for contosolh' section, which includes the location (ContosoLH), refresh status (Refreshed), and sensitivity (Confidential\Microsoft Exter). Below this are sections for visualizing and sharing the data, and a list of existing datasets sharing the same data source. A 'Tables' pane on the right allows selecting tables and columns from the dataset. A large circular button with a plus sign is visible in the bottom right corner.

Details for contosolh

Location: ContosoLH Refreshed: — Sensitivity: Confidential\Microsoft Exter

Visualize this data: Create an interactive report, or a table, to discover and share business insights. Learn more. + Create from scratch

Share this data: Give people access to the dataset and set their permissions to work with it. Learn more. Share dataset

See what already exists: These items use the same data source as contosolh.

Name	Type	Relation	Location	Refreshed	Endorsement	Sensitivity
contosolh	SQL endpoint	Upstream	TridentPP	—	—	Confidential\Mi...
contosolh	Lakehouse	Upstream	TridentPP	—	—	Confidential\Mi...

Filter by keyword Filter

Tables

Select a table and/or columns from this dataset to view and export the underlying data. Learn more

To select more than one table, and view summarized data, create a paginated report

Create paginated report

orders

taxi

trafficflow

## Setting permissions for report consumption

The default dataset is retrieving data from a lakehouse on demand. To make sure that data is accessible for the user that is viewing Power BI report, necessary permissions on the underlying lakehouse need to be set.

One option is to give the user *Viewer* role in the workspace and grant necessary permissions to data using SQL endpoint security. Alternatively, the user can be given *Admin*, *Member*, or *Contributor* role to have full access to the data.

## Next steps

- Default Power BI datasets in Microsoft Fabric

# Model scoring with PREDICT in Microsoft Fabric

Article • 05/23/2023

Microsoft Fabric allows users to operationalize machine learning models with a scalable function called PREDICT, which supports batch scoring in any compute engine. Users can generate batch predictions directly from a Microsoft Fabric notebook or from a given model's item page.

## Important

Microsoft Fabric is currently in PREVIEW. This information relates to a prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, expressed or implied, with respect to the information provided here.

In this article, you'll learn how to apply PREDICT both ways, whether you're more comfortable writing code yourself or using a guided UI experience to handle batch scoring for you.

## Prerequisites

- A Power BI Premium subscription. If you don't have one, see [How to purchase Power BI Premium](#).
- A Power BI workspace with assigned Premium capacity. If you don't have a workspace, use the steps in [Create a workspace](#) to create one and assign it to a Premium capacity.
- Sign in to [Microsoft Fabric](#).

## Limitations

- The PREDICT function is currently supported for a limited set of model flavors, including PyTorch, Sklearn, Spark, TensorFlow, ONNX, XGBoost, LightGBM, CatBoost, and Statsmodels.
- PREDICT requires models to be saved in the MLflow format with their signatures populated.
- PREDICT *does not* support models with multi-tensor inputs or outputs.

# Call PREDICT from a notebook

PREDICT supports MLflow-packaged models in the Microsoft Fabric registry. If you've already trained and registered a model in your workspace, you can skip to Step 2 below. If not, Step 1 provides sample code to guide you through training a sample logistic regression model. You can use this model to generate batch predictions at the end of the procedure.

1. **Train a model and register it with MLflow.** The following sample code uses the MLflow API to create a machine learning experiment and start an MLflow run for a scikit-learn logistic regression model. The model version is then stored and registered in the Microsoft Fabric registry. See [how to train models with scikit-learn](#) to learn more about training models and tracking experiments of your own.

Python

```
import mlflow
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.datasets import load_diabetes
from mlflow.models.signature import infer_signature

mlflow.set_experiment("diabetes-demo")
with mlflow.start_run() as run:
    lr = LogisticRegression()
    data = load_diabetes(as_frame=True)
    lr.fit(data.data, data.target)
    signature = infer_signature(data.data, data.target)

    mlflow.sklearn.log_model(
        lr,
        "diabetes-model",
        signature=signature,
        registered_model_name="diabetes-model"
    )
```

2. **Load in test data as a Spark DataFrame.** To generate batch predictions using the model trained in the previous step, you need test data in the form of a Spark DataFrame. You can substitute the value for the `test` variable in the following code with your own data.

Python

```
# You can substitute "test" below with your own data
test = spark.createDataFrame(data.frame.drop(['target'], axis=1))
```

3. Create an `MLFlowTransformer` object to load the model for inferencing. To create an `MLFlowTransformer` object for generating batch predictions, you must do the following:

- specify which columns from the `test` DataFrame you need as model inputs (in this case, all of them),
- choose a name for the new output column (in this case, `predictions`), and
- provide the correct model name and model version for generating those predictions.

If you're using your own model, substitute the values for the input columns, output column name, model name, and model version.

Python

```
from synapse.ml.predict import MLFlowTransformer

# You can substitute values below for your own input columns,
# output column name, model name, and model version
model = MLFlowTransformer(
    inputCols=test.columns,
    outputCol='predictions',
    modelName='diabetes-model',
    modelVersion=1
)
```

4. Generate predictions using the PREDICT function. To invoke the PREDICT function, you can use the Transformer API, the Spark SQL API, or a PySpark user-defined function (UDF). The following sections show how to generate batch predictions with the test data and model defined in the previous steps, using the different methods for invoking PREDICT.

## PREDICT with the Transformer API

The following code invokes the PREDICT function with the Transformer API. If you've been using your own model, substitute the values for the model and test data.

Python

```
# You can substitute "model" and "test" below with values
# for your own model and test data
model.transform(test).show()
```

## PREDICT with the Spark SQL API

The following code invokes the PREDICT function with the Spark SQL API. If you've been using your own model, substitute the values for `model_name`, `model_version`, and `features` with your model name, model version, and feature columns.

### ⓘ Note

Using the Spark SQL API to generate predictions still requires you to create an `MLflowTransformer` object (as in Step 3).

Python

```
from pyspark.ml.feature import SQLTransformer

# You can substitute "model_name," "model_version," and "features"
# with values for your own model name, model version, and feature columns
model_name = 'diabetes-model'
model_version = 1
features = test.columns

sqlt = SQLTransformer().setStatement(
    f"SELECT PREDICT('{model_name}/{model_version}', {', '.join(features)})"
    as predictions FROM __THIS__")

# You can substitute "test" below with your own test data
sqlt.transform(test).show()
```

## PREDICT with a user-defined function

The following code invokes the PREDICT function with a PySpark UDF. If you've been using your own model, substitute the values for the model and features.

Python

```
from pyspark.sql.functions import col, pandas_udf, udf, lit

# You can substitute "model" and "features" below with your own values
my_udf = model.to_udf()
features = test.columns

test.withColumn("PREDICT", my_udf(*[col(f) for f in features])).show()
```

## Generate PREDICT code from a model's item page

From any model's item page, you can choose either of the following options to start generating batch predictions for a specific model version with PREDICT.

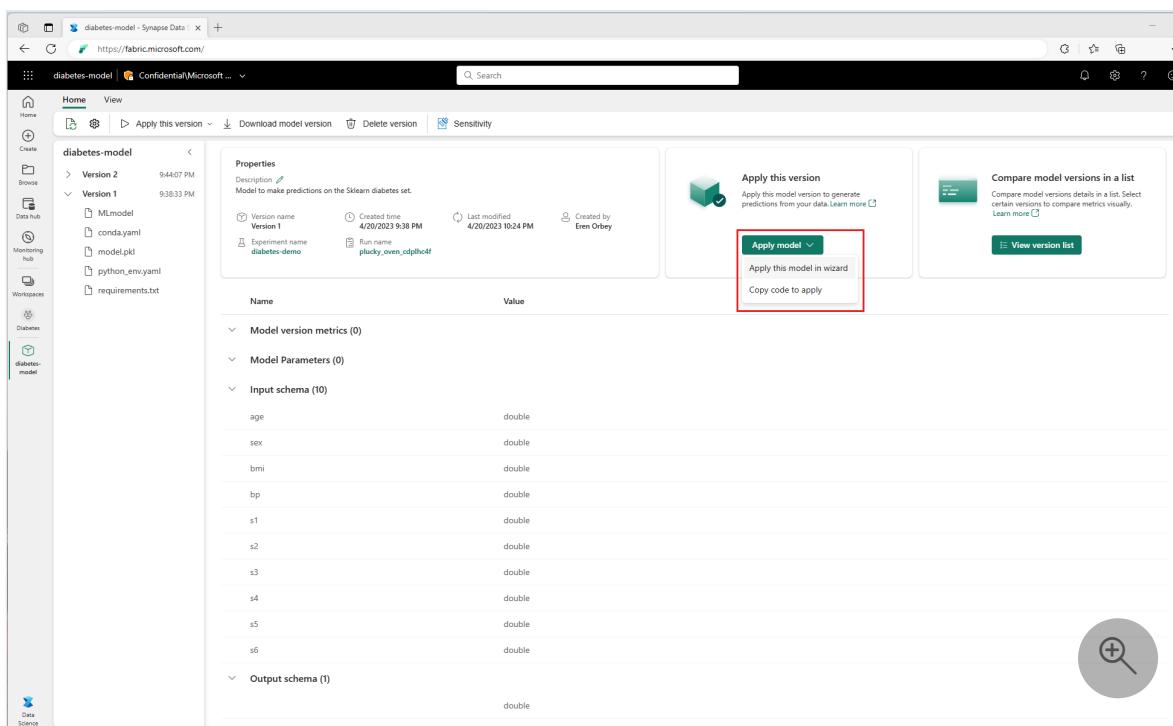
- Use a guided UI experience to generate PREDICT code for you
- Copy a code template into a notebook and customize the parameters yourself

## Use a guided UI experience

The guided UI experience walks you through steps to select source data for scoring, map the data correctly to your model's inputs, specify the destination for your model's outputs, and create a notebook that uses PREDICT to generate and store prediction results.

To use the guided experience,

1. Go to the item page for a given model version.
2. Select **Apply this model in wizard** from the **Apply model** dropdown.



The selection opens up the "Apply model predictions" window at the "Select input table" step.

3. Select an input table from one of the Lakehouses in your current workspace.

4. Select **Next** to go to the "Map input columns" step.

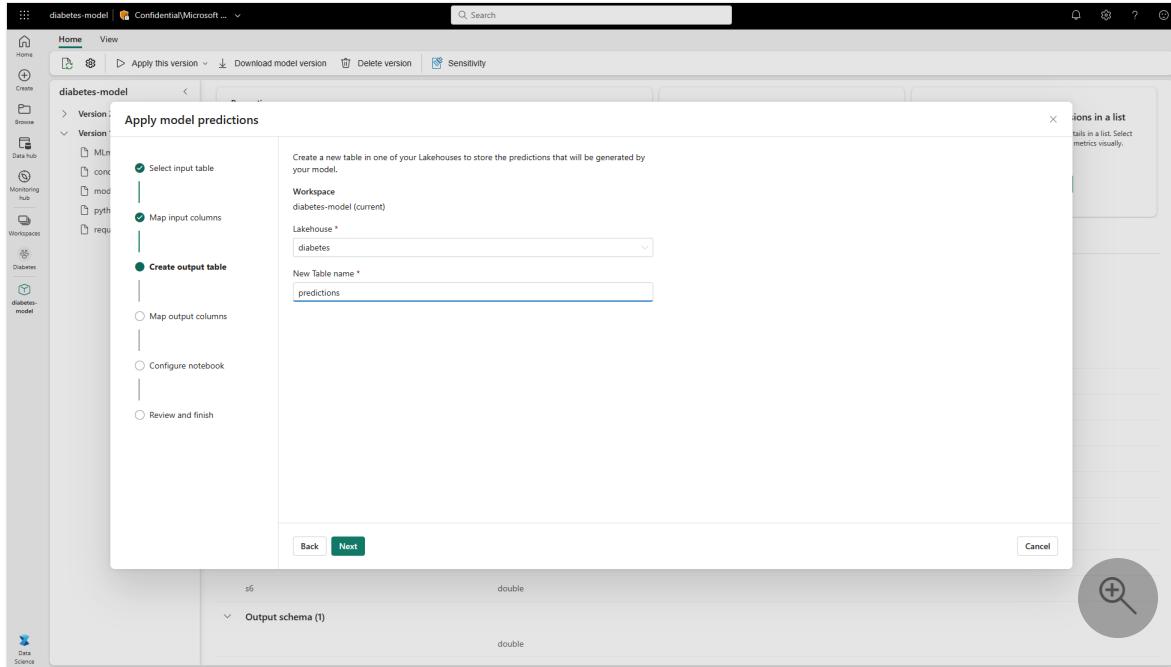
5. Map column names from the source table to the model's input fields, which have been pulled from the model's signature. You must provide an input column for all the model's required fields. Also, the data types for the source columns must match the model's expected data types.

### Tip

The wizard will prepopulate this mapping if the names of the input table's columns match the column names logged in the model signature.

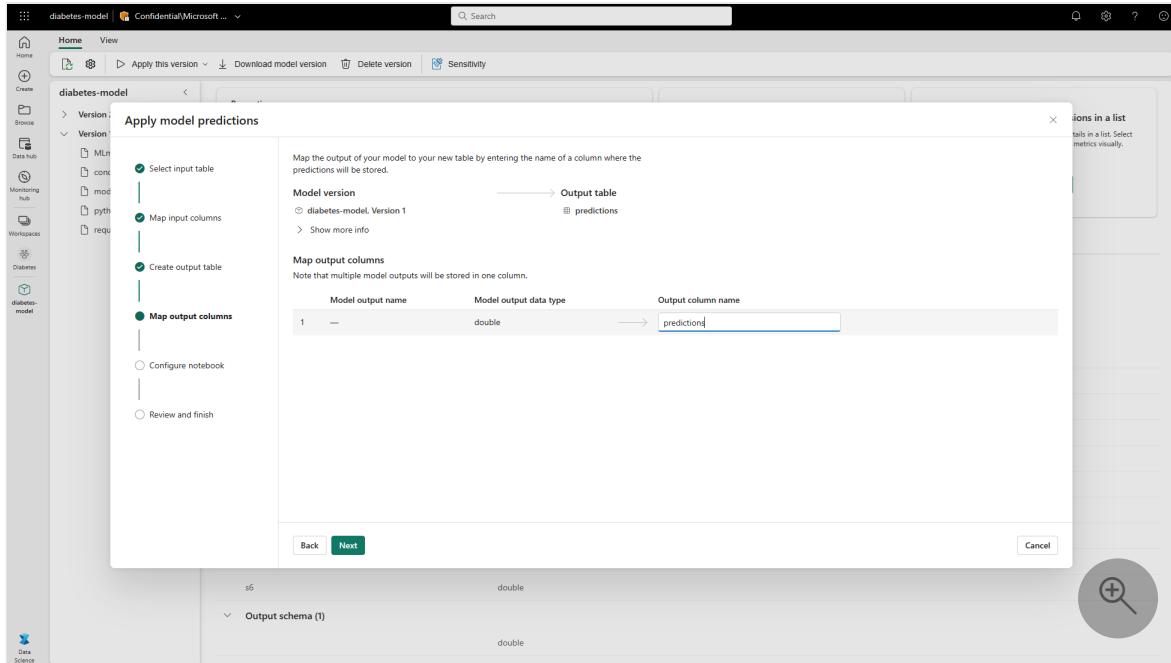
6. Select **Next** to go to the "Create output table" step.

7. Provide a name for a new table within the selected Lakehouse of your current workspace. This output table will store your model's input values with the prediction values appended. By default, the output table will be created in the same Lakehouse as the input table, but the option to change the destination Lakehouse is also available.



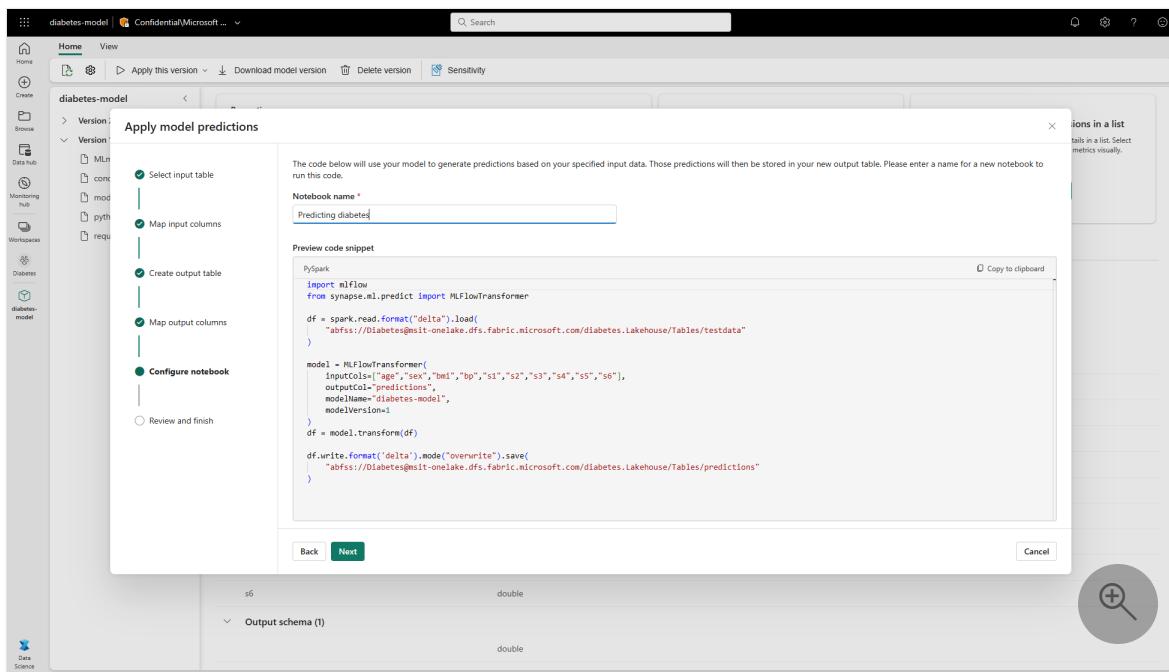
8. Select **Next** to go to the "Map output columns" step.

9. Use the provided text field(s) to name the column(s) in the output table that will store the model's predictions.



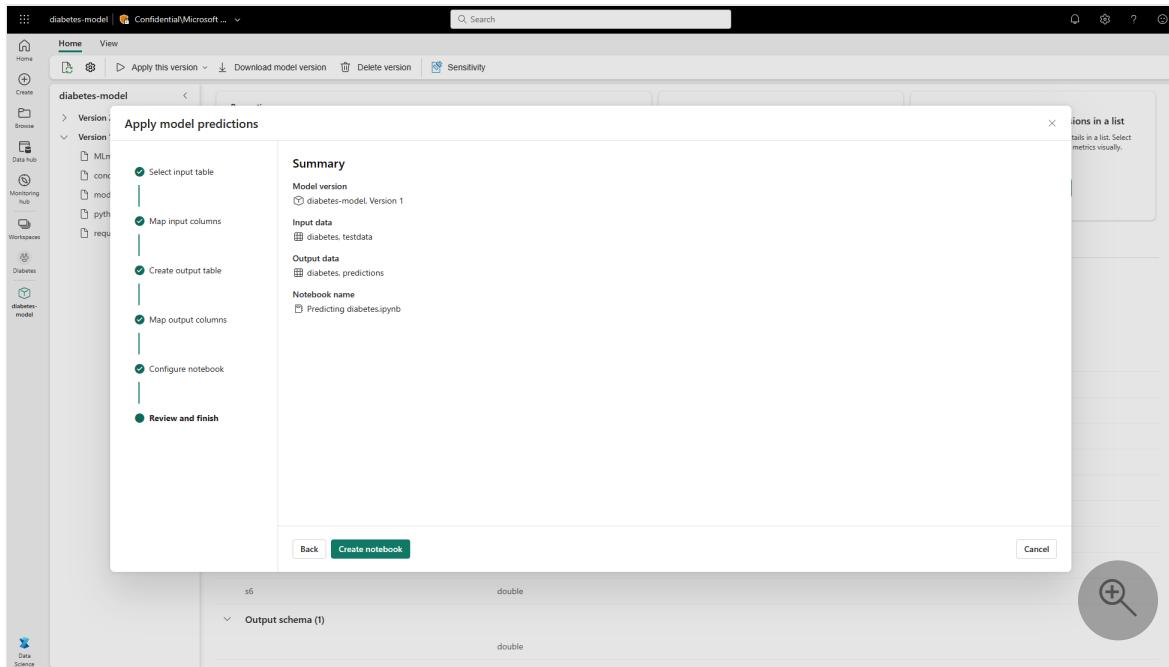
10. Select **Next** to go to the "Configure notebook" step.

11. Provide a name for a new notebook that will run the generated PREDICT code. The wizard displays a preview of the generated code at this step. You can copy the code to your clipboard and paste it into an existing notebook if you prefer.



12. Select Next to go to the "Review and finish" step.

13. Review the details on the summary page and select **Create notebook** to add the new notebook with its generated code to your workspace. You'll be taken directly to that notebook, where you can run the code to generate and store predictions.



## Use a customizable code template

To use a code template for generating batch predictions:

1. Go to the item page for a given model version.
2. Select **Copy code to apply** from the **Apply model** dropdown. The selection allows you to copy a customizable code template.

You can paste this code template into a notebook to generate batch predictions with your model. To successfully run the code template, you need to manually replace the following values:

- <INPUT\_TABLE>: The file path for the table that will provide inputs to the model
- <INPUT\_COLS>: An array of column names from the input table to feed to the model
- <OUTPUT\_COLS>: A name for a new column in the output table that will store predictions
- <MODEL\_NAME>: The name of the model to use for generating predictions
- <MODEL\_VERSION>: The version of the model to use for generating predictions
- <OUTPUT\_TABLE>: The file path for the table that will store the predictions

The screenshot shows the Azure ML studio interface. On the left, the 'diabetes-model' workspace is selected. In the center, a modal window titled 'Copy code to apply model predictions' is displayed. The modal contains a code snippet for PySpark:

```

PySpark
import mlflow
from synapse.ml.predict import MLFlowTransformer

df = spark.read.format("delta").load(
    <INPUT_TABLE>
)

model = MLFlowTransformer(
    inputCols=<INPUT_COLS>,
    outputCol=<OUTPUT_COLS>,
    modelName=<MODEL_NAME>,
    modelVersion=<MODEL_VERSION>
)

df.write.format("delta").mode("overwrite").save(
    <OUTPUT_TABLE>
)

```

At the bottom of the modal, there is a 'Copy to clipboard' button. The background of the studio shows the 'Properties' panel for Version 1 of the diabetes model, which has a description: 'Model to make predictions on Sklearn diabetes set.' and a created time of 4/20/2023.

Python

```

import mlflow
from synapse.ml.predict import MLFlowTransformer

df = spark.read.format("delta").load(
    <INPUT_TABLE>
)

model = MLFlowTransformer(
    inputCols=<INPUT_COLS>,
    outputCol=<OUTPUT_COLS>,
    modelName=<MODEL_NAME>,
    modelVersion=<MODEL_VERSION>
)

```

```
)  
df = model.transform(df)  
  
df.write.format('delta').mode("overwrite").save(  
    <OUTPUT_TABLE>  
)
```

## Next steps

- End-to-end prediction example using a fraud detection model
- How to train models with scikit-learn in Microsoft Fabric

# What is Spark compute in Microsoft Fabric?

Article • 05/23/2023

Applies to:  Data Engineering and Data Science in Microsoft Fabric

Microsoft Fabric Data Engineering and Data Science experiences operate on a fully managed Spark compute platform. This platform is designed to deliver unparalleled speed and efficiency. With starter pools, you can expect rapid spark session initialization, typically within 5 to 10 seconds. It eliminates the need for manual setup. Furthermore, you also get the flexibility to customize Spark pools according to the specific data engineering and data science requirements. It enables an optimized and tailored analytics experience.

## Important

Microsoft Fabric is currently in PREVIEW. This information relates to a prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, expressed or implied, with respect to the information provided here.

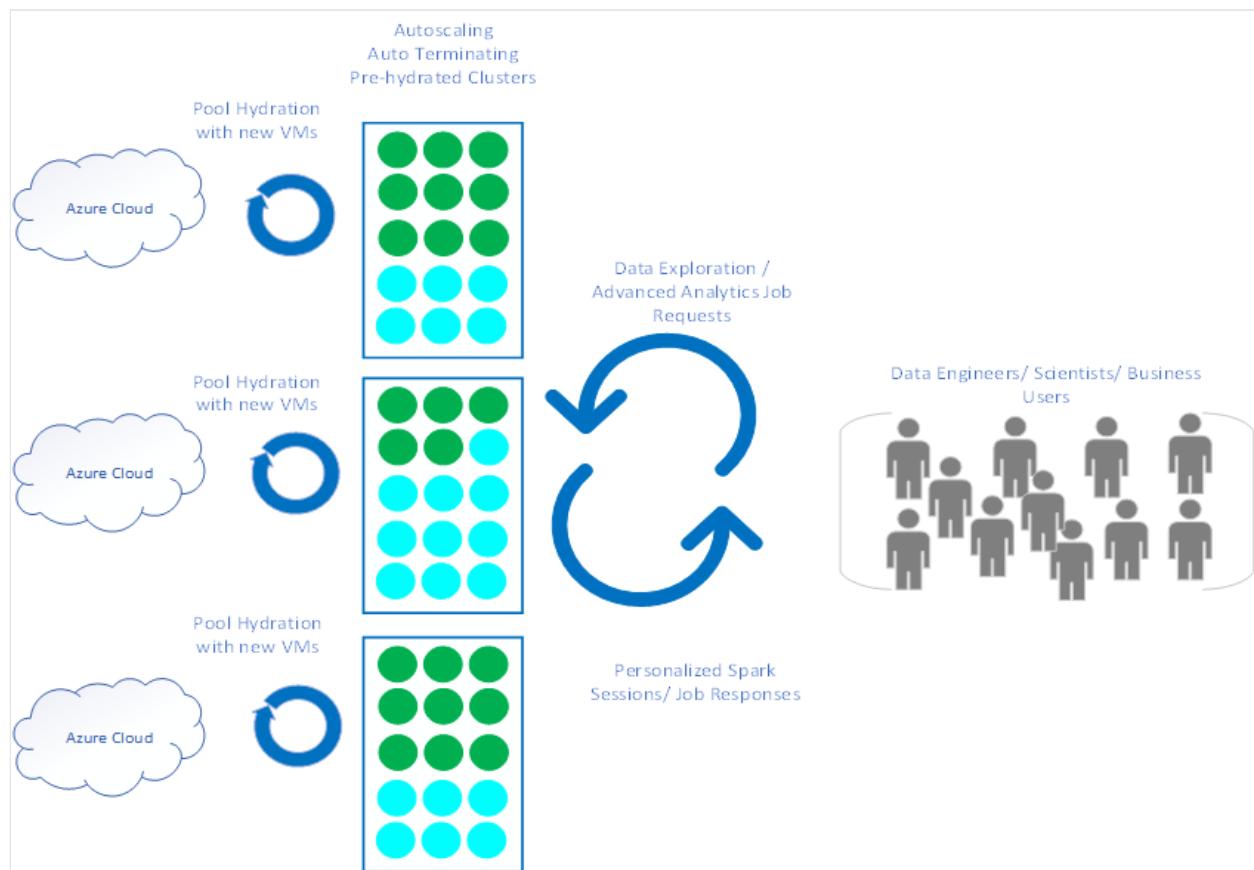
## Starter pools

Starter pools are a fast and easy way to use Spark on the Microsoft Fabric platform within seconds. You can use Spark sessions right away, instead of waiting for Spark to set up the nodes for you. This helps you do more with data and get insights quicker.

# Starter Pool Configuration

Node family	Memory optimized
Node Size	Medium
Min and Max Nodes	[1, 10]
Autoscale	On
Dynamic Allocation	On

Starter pools have Spark clusters that are always on and ready for your requests. They use medium nodes that will dynamically scale-up based on your Spark job needs.



Starter pools also have default settings that let you install libraries quickly without slowing down the session start time. However, if you want to use extra custom Spark properties or libraries from your workspace or capacity settings, it may take longer for

Spark to get the nodes for you. You only pay for starter pools when you are using Spark sessions to run queries. You don't pay for the time when Spark is keeping the nodes ready for you.

## Spark pools

A Spark pool is a way of telling Spark what kind of resources you need for your data analysis tasks. You can give your Spark pool a name, and choose how many and how big the nodes (the machines that do the work) are. You can also tell Spark how to adjust the number of nodes depending on how much work you have. Creating a Spark pool is free; you only pay when you run a Spark job on the pool, and then Spark will set up the nodes for you.

If you don't use your Spark pool for 2 minutes after your job is done, Spark will automatically delete it. This is called the "time to live" property, and you can change it if you want. If you are a workspace admin, you can also create custom Spark pools for your workspace, and make them the default option for other users. This way, you can save time and avoid setting up a new Spark pool every time you run a notebook or a Spark job. Custom Spark pools take about 3 minutes to start, because Spark has to get the nodes from Azure.

The size and number of nodes you can have in your custom Spark pool depends on how much capacity you have in your Microsoft Fabric capacity. This is a measure of how much computing power you can use in Azure. One way to think of it is that two Spark VCores (a unit of computing power for Spark) equals one capacity unit. For example, if you have a Fabric capacity SKU F64, that means you have 64 capacity units, which is equivalent to 128 Spark VCores. You can use these Spark VCores to create nodes of different sizes for your custom Spark pool, as long as the total number of Spark VCores does not exceed 128.

Possible custom pool configurations for F64 based on the above example

Fabric Capacity SKU	Capacity Units	Spark VCores	Node Size	Max Number of Nodes
F64	64	128	Small	32
F64	64	128	Medium	16
F64	64	128	Large	8
F64	64	128	X-Large	4
F64	64	128	XX-Large	2

## Note

To create custom pools, you should have the **admin** permissions for the workspace. And the Microsoft Fabric capacity admin should have granted permissions to allow workspace admins to size their custom spark pools. To learn more, see [Get Started with Custom Spark Pools in Fabric](#)

# Nodes

Apache Spark pool instance consists of one head node and two or more worker nodes with a minimum of three nodes in a Spark instance. The head node runs extra management services such as Livy, Yarn Resource Manager, Zookeeper, and the Spark driver. All nodes run services such as Node Agent and Yarn Node Manager. All worker nodes run the Spark Executor service.

## Node sizes

A Spark pool can be defined with node sizes that range from a small compute node with 4 vCore and 32 GB of memory to a large compute node with 64 vCore and 512 GB of memory per node. Node sizes can be altered after pool creation although the active session would have to be restarted.

Size	vCore	Memory
Small	4	32 GB
Medium	8	64 GB
Large	16	128 GB
X-Large	32	256 GB
XX-Large	64	512 GB

## Autoscale

Autoscale for Apache Spark pools allows automatic scale up and down of compute resources based on the amount of activity. When the autoscale feature is enabled, you set the minimum, and maximum number of nodes to scale. When the autoscale feature is disabled, the number of nodes set remains fixed. This setting can be altered after pool creation although the instance may need to be restarted.

# Dynamic allocation

Dynamic allocation allows the spark application to request more executors if the tasks exceed the load that current executors can bear. It also releases the executors when the jobs are completed and if the spark application is moving to idle state. Enterprise users often find it hard to tune the executor configurations. Because they're vastly different across different stages of a Spark Job execution process. These are also dependent on the volume of data processed which changes from time to time. Users can enable dynamic allocation of executors option as part of the pool configuration, which would enable automatic allocation of executors to the spark application based on the nodes available in the Spark pool.

When dynamic allocation option is enabled, for every spark application submitted. The system reserves executors during the job submission step based on the maximum nodes, which were specified by the user to support successful auto scale scenarios.

## Next steps

- [Get Started with Data Engineering/Science Admin Settings for your Fabric Capacity](#)
- [Get Started with Data Engineering/Science Admin Settings for your Fabric Workspace](#)

# Apache Spark Runtime in Fabric

Article • 05/23/2023

The Microsoft Fabric Runtime is an Azure-integrated platform based on Apache Spark that enables the execution and management of data engineering and data science experiences. It combines key components from both internal and open-source sources, providing customers with a comprehensive solution. For simplicity, we refer to the Microsoft Fabric Runtime powered by Apache Spark as Fabric Runtime.

## Important

Microsoft Fabric is currently in PREVIEW. This information relates to a prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, expressed or implied, with respect to the information provided here.

Major components of the Fabric Runtime:

- **Apache Spark** - a powerful open-source distributed computing library, to enable large-scale data processing and analytics tasks. Apache Spark provides a versatile and high-performance platform for data engineering and data science experiences.
- **Delta Lake** - an open-source storage layer that brings ACID transactions and other data reliability features to Apache Spark. Integrated within the Microsoft Fabric Runtime, Delta Lake enhances the data processing capabilities and ensures data consistency across multiple concurrent operations.
- **Default-level Packages for Java/Scala, Python, and R** to support diverse programming languages and environments. These packages are automatically installed and configured, allowing developers to apply their preferred programming languages for data processing tasks.
- The Microsoft Fabric Runtime is built upon a **robust open-source operating system (Ubuntu)**, ensuring compatibility with various hardware configurations and system requirements.

## Runtime 1.1

Microsoft Fabric Runtime 1.1 is the default and currently the only runtime offered within the Microsoft Fabric platform. The Runtime 1.1 major components are:

- Operating System: Ubuntu 18.04
- Java: 1.8.0\_282
- Scala: 2.12.15
- Python: 3.10
- Delta Lake: 2.2
- R: 4.2.2

## Runtime

### Runtime Version

Runtime family defines which version of Spark your Spark pool will use.

[Learn more about Runtime Version](#) ↗

1.1 (Spark 3.3, Delta 2.2) ▾

Microsoft Fabric Runtime 1.1 comes with a collection of default level packages, including a full Anaconda installation and commonly used libraries for Java/Scala, Python, and R. These libraries are automatically included when using notebooks or jobs in the Microsoft Fabric platform. Refer to the documentation for a complete list of libraries.

Microsoft Fabric periodically rolls out maintenance updates for Runtime 1.1, providing bug fixes, performance enhancements, and security patches. *Staying up to date ensures optimal performance and reliability for your data processing tasks.*

## New features and improvements

### Apache Spark 3.3.1

Following is an extended summary of key new features related to Apache Spark version 3.3.0 and 3.3.1:

- **Row-level filtering:** improve the performance of joins by prefiltering one side as long as there are no deprecation or regression impacts.oin using a Bloom filter and IN predicate generated from the values from the other side of the join ([SPARK-32268](#) ↗)
- Improve the compatibility of Spark with the SQL standard:**ANSI enhancements** ([SPARK-38860](#) ↗)
- Error Message Improvements to identify problems faster and take the necessary steps to resolve it ([SPARK-38781](#) ↗)
- Support **complex types for Parquet vectorized reader**. Previously, Parquet vectorized reader hasn't supported nested column type (struct, array, and map). The Apache Spark 3.3 contains an implementation of nested column vectorized reader for FB-ORC in our internal fork of Spark. It impacts performance improvement compared to nonvectorized reader when reading nested columns. In addition, this implementation can also help improve the non-nested column performance when reading non-nested and nested columns together in one query ([SPARK-34863](#) ↗)
- Allows users to query the metadata of the input files for all file formats, expose them as **built-in hidden columns** meaning **users can only see them when they explicitly reference them** (for example, file path and file name) ([SPARK-37273](#) ↗)
- Provide a profiler for Python/Pandas UDFs ([SPARK-37443](#) ↗)

- Previously, streaming queries with Trigger, which was loading all of the available data in a single batch. Because of this, the amount of data the queries could process was limited, or the Spark driver would be out of memory. Now, introducing **Trigger.AvailableNow** for running streaming queries like Trigger once in multiple batches ([SPARK-36533 ↗](#))
- More comprehensive DS V2 push down capabilities ([SPARK-38788 ↗](#))
- Executor Rolling in Kubernetes environment ([SPARK-37810 ↗](#))
- Support Customized Kubernetes Schedulers ([SPARK-36057 ↗](#))
- Migrating from **log4j 1** to **log4j 2** ([SPARK-37814 ↗](#)) to gain in:
  - Performance: Log4j 2 is faster than Log4j 1. Log4j 2 uses **asynchronous logging by default**, which can improve performance significantly.
  - Flexibility: Log4j 2 provides more flexibility in terms of configuration. It supports **multiple configuration formats**, including XML, JSON, and YAML.
  - Extensibility: Log4j 2 is designed to be extensible. It allows developers to **create custom plugins and appenders** to extend the functionality of the logging framework.
  - Security: Log4j 2 provides better security features than Log4j 1. It supports **encryption and secure socket layers** for secure communication between applications.
  - Simplicity: Log4j 2 is simpler to use than Log4j 1. It has a **more intuitive API** and a simpler configuration process.
- Introduce shuffle on **SinglePartition** to improve parallelism and fix performance regression for joins in Spark 3.3 vs Spark 3.2 ([SPARK-40703 ↗](#))
- Optimize **TransposeWindow** rule to extend applicable cases and optimize time complexity ([SPARK-38034 ↗](#))
- To have a parity in doing TimeTravel via SQL and Dataframe option, **support timestamp** in seconds for **TimeTravel** using Dataframe options ([SPARK-39633\] ↗](#))
- Optimize **global Sort to RepartitionByExpression** to save a local sort ([SPARK-39911 ↗](#))
- Ensure the **output partitioning** is user-specified in **AQE** ([SPARK-39915 ↗](#))
- Update Parquet V2 columnar check for nested fields ([SPARK-39951 ↗](#))
- Reading in a **parquet file partitioned on disk by a `Byte`-type column** ([SPARK-40212 ↗](#))
- Fix column pruning in CSV when `_corrupt_record` is selected ([SPARK-40468 ↗](#))

## Delta Lake 2.2

The key features in this release are as follows:

- [LIMIT](#) pushdown into Delta scan. Improve the performance of queries containing `LIMIT` clauses by pushing down the `LIMIT` into Delta scan during query planning. Delta scan uses the `LIMIT` and the file-level row counts to reduce the number of files scanned which helps the queries read far less number of files and could make `LIMIT` queries faster by 10-100x depending upon the table size.
- [Aggregate](#) pushdown into Delta scan for `SELECT COUNT(*)`. Aggregation queries such as `SELECT COUNT(*)` on Delta tables are satisfied using file-level row counts in Delta table metadata rather than counting rows in the underlying data files. This significantly reduces the query time as the query just needs to read the table metadata and could make full table count queries faster by 10-100x.
- [Support](#) for collecting file level statistics as part of the `CONVERT TO DELTA` command. These statistics potentially help speed up queries on the Delta table. By default the statistics are collected now as part of the `CONVERT TO DELTA` command. In order to disable statistics collection, specify `NO STATISTICS` clause in the command. Example: `CONVERT TO DELTA table_name NO STATISTICS`
- [Improve](#) performance of the `DELETE` command by pruning the columns to read when searching for files to rewrite.
- [Fix](#) for a bug in the DynamoDB-based `S3 multi-cluster mode` configuration. The previous version wrote an incorrect timestamp, which was used by `DynamoDB's TTL` feature to clean up expired items. This timestamp value has been fixed and the table attribute renamed from `commitTime` to `expireTime`. If you already have TTL enabled, follow the migration steps [here](#).
- [Fix](#) `nondeterministic` behavior during `MERGE` when working with sources that are nondeterministic.
- [Remove](#) the restrictions for using Delta tables with column mapping in certain Streaming + CDF cases. Earlier we used to block Streaming+CDF if the Delta table has column mapping enabled even though it doesn't contain any `RENAME` or `DROP` columns.
- [Improve](#) the monitoring of the Delta state construction queries (other queries run as part of planning) by making them visible in the Spark UI.
- [Support](#) for multiple `where()` calls in Optimize scala/python API
- [Support](#) for passing Hadoop configurations via `DeltaTable API`
- [Support](#) partition column names starting with `.` or `_` in `CONVERT TO DELTA` command.
- Improvements to metrics in table history
  - [Fix](#) a metric in `MERGE` command
  - [Source type](#) metric for `CONVERT TO DELTA`
  - [Metrics](#) for `DELETE` on partitions
  - [More](#) vacuum stats

- [Fix](#) for accidental protocol downgrades with `RESTORE` command. Until now, `RESTORE TABLE` may downgrade the protocol version of the table, which could have resulted in inconsistent reads with time travel. With this fix, the protocol version is never downgraded from the current one.
- [Fix](#) a bug in `MERGE INTO` when there are multiple `UPDATE` clauses and one of the `UPDATES` is with a schema evolution.
- [Fix](#) a bug where sometimes active `SparkSession` object isn't found when using Delta APIs
- [Fix](#) an issue where partition schema couldn't be set during the initial commit.
- [Catch](#) exceptions when writing `last_checkpoint` file fails.
- [Fix](#) an issue when restarting a streaming query with `AvailableNow` trigger on a Delta table.
- [Fix](#) an issue with CDF and Streaming where the offset isn't correctly updated when there are no data changes

Check the source and full release notes [here](#).

## Default level packages for Java/Scala libraries

Below you can find the table with listing all the default level packages for Java/Scala and their respective versions.

GroupId	ArtifactId	Version
com.aliyun	aliyun-java-sdk-core	4.5.10
com.aliyun	aliyun-java-sdk-kms	2.11.0
com.aliyun	aliyun-java-sdk-ram	3.1.0
com.aliyun	aliyun-sdk-oss	3.13.0
com.amazonaws	aws-java-sdk-bundle	1.11.1026
com.chuusai	shapeless_2.12	2.3.7
com.esotericsoftware	kryo-shaded	4.0.2
com.esotericsoftware	minlog	1.3.0
com.fasterxml.jackson	jackson-annotations-2.13.4.jar	
com.fasterxml.jackson	jackson-core	2.13.4
com.fasterxml.jackson	jackson-core-asl	1.9.13
com.fasterxml.jackson	jackson-databind	2.13.4.1
com.fasterxml.jackson	jackson-dataformat-cbor	2.13.4
com.fasterxml.jackson	jackson-mapper-asl	1.9.13

GroupId	ArtifactId	Version
com.fasterxml.jackson	jackson-module-scala_2.12	2.13.4
com.github.joshelser	dropwizard-metrics-hadoop-metrics2-reporter	0.1.2
com.github.wendykierp	JTransforms	3.1
com.google.code.findbugs	jsr305	3.0.0
com.google.code.gson	gson	2.8.6
com.google.flatbuffers	flatbuffers-java	1.12.0
com.google.guava	guava	14.0.1
com.google.protobuf	protobuf-java	2.5.0
com.googlecode.json-simple	json-simple	1.1.1
com.jcraft	jsch	0.1.54
com.jolbox	bonecp	0.8.0.RELEASE
com.linkedin.isolation-forest	isolation-forest_3.2.0_2.12	2.0.8
com.ning	compress-lzf	1.1
com.qcloud	cos_api-bundle	5.6.19
com.sun.istack	istack-commons-runtime	3.0.8
com.tdunning	json	1.8
com.thoughtworks.paranamer	paranamer	2.8
com.twitter	chill-java	0.10.0
com.twitter	chill_2.12	0.10.0
com.typesafe	config	1.3.4
com.zaxxer	HikariCP	2.5.1
commons-cli	commons-cli	1.5.0
commons-codec	commons-codec	1.15
commons-collections	commons-collections	3.2.2
commons-dbcp	commons-dbcp	1.4
commons-io	commons-io	2.11.0
commons-lang	commons-lang	2.6
commons-logging	commons-logging	1.1.3
commons-pool	commons-pool	1.5.4.jar

GroupId	ArtifactId	Version
dev.ludovic.netlib	arpack	2.2.1
dev.ludovic.netlib	blas	2.2.1
dev.ludovic.netlib	lapack	2.2.1
io.airlift	aircompressor	0.21
io.dropwizard.metrics	metrics-core	4.2.7
io.dropwizard.metrics	metrics-graphite	4.2.7
io.dropwizard.metrics	metrics-jmx	4.2.7
io.dropwizard.metrics	metrics-json	4.2.7
io.dropwizard.metrics	metrics-jvm	4.2.7
io.netty	netty-all	4.1.74.Final
io.netty	netty-buffer	4.1.74.Final
io.netty	netty-codec	4.1.74.Final
io.netty	netty-common	4.1.74.Final
io.netty	netty-handler	4.1.74.Final
io.netty	netty-resolver	4.1.74.Final
io.netty	netty-tcnative-classes	2.0.48.Final
io.netty	netty-transport	4.1.74.Final
io.netty	netty-transport-classes-epoll	4.1.74.Final
io.netty	netty-transport-classes-kqueue	4.1.74.Final
io.netty	netty-transport-native-epoll	4.1.74.Final-linux-aarch_64
io.netty	netty-transport-native-epoll	4.1.74.Final-linux-x86_64
io.netty	netty-transport-native-kqueue	4.1.74.Final-osx-aarch_64
io.netty	netty-transport-native-kqueue	4.1.74.Final-osx-x86_64
io.netty	netty-transport-native-unix-common	4.1.74.Final
io.opentracing	opentracing-api	0.33.0
io.opentracing	opentracing-noop	0.33.0
io.opentracing	opentracing-util	0.33.0
jakarta.annotation	jakarta.annotation-api	1.3.5
jakarta.inject	jakarta.inject	2.6.1

GroupId	ArtifactId	Version
jakarta.servlet	jakarta.servlet-api	4.0.3
jakarta.validation-api	2.0.2	
jakarta.ws.rs	jakarta.ws.rs-api	2.1.6
jakarta.xml.bind	jakarta.xml.bind-api	2.3.2
javax.activation	activation	1.1.1
javax.jdo	jdo-api	3.0.1
javax.transaction	jta	1.1
javax.xml.bind	jaxb-api	2.2.11
javolution	javolution	5.5.1
jline	jline	2.14.6
joda-time	joda-time	2.10.13
net.razorvine	pickle	1.2
net.sf.jpam	jpam	1.1
net.sf.opencsv	opencsv	2.3
net.sf.py4j	py4j	0.10.9.5
net.sourceforge.f2j	arpack_combined_all	0.1
org.antlr	ST4	4.0.4
org.antlr	antlr-runtime	3.5.2
org.antlr	antlr4-runtime	4.8
org.apache.arrow	arrow-format	7.0.0
org.apache.arrow	arrow-memory-core	7.0.0
org.apache.arrow	arrow-memory-netty	7.0.0
org.apache.arrow	arrow-vector	7.0.0
org.apache.avro	avro	1.11.0
org.apache.avro	avro-ipc	1.11.0
org.apache.avro	avro-mapred	1.11.0
org.apache.commons	commons-collections4	4.4
org.apache.commons	commons-compress	1.21
org.apache.commons	commons-crypto	1.1.0
org.apache.commons	commons-lang3	3.12.0

GroupId	ArtifactId	Version
org.apache.commons	commons-math3	3.6.1
org.apache.commons	commons-pool2	2.11.1
org.apache.commons	commons-text	1.10.0
org.apache.curator	curator-client	2.13.0
org.apache.curator	curator-framework	2.13.0
org.apache.curator	curator-recipes	2.13.0
org.apache.derby	derby	10.14.2.0
org.apache.hadoop	hadoop-aliyun	3.3.3.5.2-90111858
org.apache.hadoop	hadoop-annotations	3.3.3.5.2-90111858
org.apache.hadoop	hadoop-aws	3.3.3.5.2-90111858
org.apache.hadoop	hadoop-azure	3.3.3.5.2-90111858
org.apache.hadoop	hadoop-azure-datalake	3.3.3.5.2-90111858
org.apache.hadoop	hadoop-client-api	3.3.3.5.2-90111858
org.apache.hadoop	hadoop-client-runtime	3.3.3.5.2-90111858
org.apache.hadoop	hadoop-cloud-storage	3.3.3.5.2-90111858
org.apache.hadoop	hadoop-cos	3.3.3.5.2-90111858
org.apache.hadoop	hadoop-openstack	3.3.3.5.2-90111858
org.apache.hadoop	hadoop-shaded-guava	1.1.1
org.apache.hadoop	hadoop-yarn-server-web-proxy	3.3.3.5.2-90111858
org.apache.hive	hive-beeline	2.3.9
org.apache.hive	hive-cli	2.3.9
org.apache.hive	hive-common	2.3.9
org.apache.hive	hive-exec	2.3.9
org.apache.hive	hive-jdbc	2.3.9
org.apache.hive	hive-llap-common	2.3.9
org.apache.hive	hive-metastore	2.3.9
org.apache.hive	hive-serde	2.3.9
org.apache.hive	hive-service-rpc	3.1.2
org.apache.hive	hive-shims-0.23	2.3.9
org.apache.hive	hive-shims	2.3.9

GroupId	ArtifactId	Version
org.apache.hive	hive-shims-common	2.3.9
org.apache.hive	hive-shims-scheduler	2.3.9
org.apache.hive	hive-storage-api	2.7.2
org.apache.hive	hive-vector-code-gen	2.3.9
org.apache.httpcomponents	httpclient	4.5.13
org.apache.httpcomponents	httpcore	4.4.14
org.apache.httpcomponents	httpmime	4.5.13
org.apache.httpcomponents.client5	httpclient5	5.1.3
org.apache.ivy	ivy	2.5.1
org.apache.kafka	kafka-clients	2.8.1
org.apache.logging.log4j	log4j-1.2-api	2.17.2
org.apache.logging.log4j	log4j-api	2.17.2
org.apache.logging.log4j	log4j-core	2.17.2
org.apache.logging.log4j	log4j-slf4j-impl	2.17.2
org.apache.orc	orc-core	1.7.6
org.apache.orc	orc-mapreduce	1.7.6
org.apache.orc	orc-shims	1.7.6
org.apache.parquet	parquet-column	1.12.3
org.apache.parquet	parquet-common	1.12.3
org.apache.parquet	parquet-encoding	1.12.3
org.apache.parquet	parquet-format-structures	1.12.3
org.apache.parquet	parquet-hadoop	1.12.3
org.apache.parquet	parquet-jackson	1.12.3
org.apache.qpid	proton-j	0.33.8
org.apache.thrift	libfb303	0.9.3
org.apache.thrift	libthrift	0.12.0
org.apache.yetus	audience-annotations	0.5.0
org.apiguardian	apiguardian-api	1.1.0
org.codehaus.janino	commons-compiler	3.0.16
org.codehaus.janino	janino	3.0.16

GroupId	ArtifactId	Version
org.codehaus.jettison	jettison	1.1
org.datanucleus	datanucleus-api-jdo	4.2.4
org.datanucleus	datanucleus-core	4.1.17
org.datanucleus	datanucleus-rdbms	4.1.19
org.datanucleusjavax.jdo	3.2.0-m3	
org.eclipse.jdt	core	1.1.2
org.eclipse.jetty	jetty-util	9.4.48.v20220622
org.eclipse.jetty	jetty-util-ajax	9.4.48.v20220622
org.fusesource.leveldbjni	leveldbjni-all	1.8
org.glassfish.hk2	hk2-api	2.6.1
org.glassfish.hk2	hk2-locator	2.6.1
org.glassfish.hk2	hk2-utils	2.6.1
org.glassfish.hk2	osgi-resource-locator	1.0.3
org.glassfish.hk2.external	aopalliance-repackaged	2.6.1
org.glassfish.jaxb	jaxb-runtime	2.3.2
org.glassfish.jersey.containers	jersey-container-servlet	2.36
org.glassfish.jersey.containers	jersey-container-servlet-core	2.36
org.glassfish.jersey.core	jersey-client	2.36
org.glassfish.jersey.core	jersey-common	2.36
org.glassfish.jersey.core	jersey-server	2.36
org.glassfish.jersey.inject	jersey-hk2	2.36
org.ini4j	ini4j	0.5.4
org.javassist	javassist	3.25.0-GA
org.javatuples	javatuples	1.2
org.jdom	jdom2	2.0.6
org.jetbrains	annotations	17.0.0
org.jodd	jodd-core	3.5.2
org.json4s	json4s-ast_2.12	3.7.0-M11
org.json4s	json4s-core_2.12	3.7.0-M11
org.json4s	json4s-jackson_2.12	3.7.0-M11

GroupId	ArtifactId	Version
org.json4s	json4s-scalap_2.12	3.7.0-M11
org.junit.jupiter	junit-jupiter	5.5.2
org.junit.jupiter	junit-jupiter-api	5.5.2
org.junit.jupiter	junit-jupiter-engine	5.5.2
org.junit.jupiter	junit-jupiter-params	5.5.2
org.junit.platform	junit-platform-commons	1.5.2
org.junit.platform	junit-platform-engine	1.5.2
org.lz4	lz4-java	1.8.0
org.objenesis	objenesis	3.2
org.openpnp	opencv	3.2.0-1
org.opentest4j	opentest4j	1.2.0
org.postgresql	postgresql	42.2.9
org.roaringbitmap	RoaringBitmap	0.9.25
org.roaringbitmap	shims	0.9.25
org.rocksdb	rocksdbjni	6.20.3
org.scala-lang	scala-compiler	2.12.15
org.scala-lang	scala-library	2.12.15
org.scala-lang	scala-reflect	2.12.15
org.scala-lang.modules	scala-collection-compat_2.12	2.1.1
org.scala-lang.modules	scala-java8-compat_2.12	0.9.0
org.scala-lang.modules	scala-parser-combinators_2.12	1.1.2
org.scala-lang.modules	scala-xml_2.12	1.2.0
org.scalactic	scalactic_2.12	3.2.14
org.scalanlp	breeze-macros_2.12	1.2
org.scalanlp	breeze_2.12	1.2
org.slf4j	jcl-over-slf4j	1.7.32
org.slf4j	jul-to-slf4j	1.7.32
org.slf4j	slf4j-api	1.7.32
org.typelevel	algebra_2.12	2.0.1
org.typelevel	cats-kernel_2.12	2.1.1

GroupId	ArtifactId	Version
org.typelevel	spire-macros_2.12	0.17.0
org.typelevel	spire-platform_2.12	0.17.0
org.typelevel	spire-util_2.12	0.17.0
org.xerial.snappy	snappy-java	1.1.8.4
oro	oro	2.0.8
pl.edu.icm	JLargeArrays	1.5

## Default level packages for Python libraries

Below you can find the table with listing all the default level packages for Python and their respective versions.

Library	Version	Library	Version	Library	Version
_libgcc_mutex	0.1	ipykernel	6.22.0	pickleshare	0.7.5
_openmp_mutex	4.5	ipython	8.9.0	pillow	9.4.0
_py-xgboost-mutex	2.0	ipywidgets	8.0.4	pip	23.0.1
absl-py	1.4.0	isodate	0.6.1	pixman	0.40.0
adal	1.2.7	itsdangerous	2.1.2	pkginfo	1.9.6
adlfs	2023.1.0	jack	1.9.22	pkgutil-resolve-name	1.3.10
aiohttp	3.8.4	jedi	0.18.2	platformdirs	3.2.0
aiosignal	1.3.1	jeepney	0.8.0	plotly	5.13.0
alsa-lib	1.2.8	jinja2	3.1.2	ply	3.11
anyio	3.6.2	jmespath	1.0.1	pooch	1.7.0
argcomplete	2.1.2	joblib	1.2.0	portalocker	2.7.0
argon2-cffi	21.3.0	jpeg	9e	pox	0.3.2
argon2-cffi-bindings	21.2.0	jsonpickle	2.2.0	ppft	1.7.6.6
arrow-cpp	11.0.0	jsonschema	4.17.3	prettytable	3.6.0
asttokens	2.2.1	jupyter_client	8.1.0	prometheus_client	0.16.0
astunparse	1.6.3	jupyter_core	5.3.0	prompt-toolkit	3.0.38
async-timeout	4.0.2	jupyter_events	0.6.3	protobuf	4.21.12
atk-1.0	2.38.0	jupyter_server	2.2.1	psutil	5.9.4

Library	Version	Library	Version	Library	Version
attr	2.5.1	jupyter_server_terminals	0.4.4	pthread-stubs	0.4
attrs	22.2.0	jupyterlab_pygments	0.2.2	ptyprocess	0.7.0
aws-c-auth	0.6.24	jupyterlab_widgets	3.0.7	pulseaudio	16.1
aws-c-cal	0.5.20	keras	2.11.0	pulseaudio-client	16.1
aws-c-common	0.8.11	keras-preprocessing	1.1.2	pulseaudio-daemon	16.1
aws-c-compression	0.2.16	keyutils	1.6.1	pure_eval	0.2.2
aws-c-event-stream	0.2.18	kiwisolver	1.4.4	py-xgboost	1.7.1
aws-c-http	0.7.4	knack	0.10.1	py4j	0.10.9.5
aws-c-io	0.13.17	krb5	1.20.1	pyarrow	11.0.0
aws-c-mqtt	0.8.6	lame	3.100	pyasn1	0.4.8
aws-c-s3	0.2.4	lcms2	2.15	pyasn1-modules	0.2.7
aws-sdkutils	0.1.7	ld_impl_linux-64	2.40	pycosat	0.6.4
aws-checksums	0.1.14	lerc	4.0.0	pycparser	2.21
aws-crt-cpp	0.19.7	liac-arff	2.5.0	pygments	2.14.0
aws-sdk-cpp	1.10.57	libabseil	20220623.0	pyjwt	2.6.0
azure-common	1.1.28	libaec	1.0.6	pynacl	1.5.0
azure-core	1.26.4	libarrow	11.0.0	pyodbc	4.0.35
azure-datalake-store	0.0.51	libblas	3.9.0	pyopenssl	23.1.1
azure-graphrbac	0.61.1	libbrotlicommon	1.0.9	pyparsing	3.0.9
azure-identity	1.12.0	libbrotlidec	1.0.9	pyperclip	1.8.2
azure-mgmt-authorization	3.0.0	libbrotlienc	1.0.9	pyqt	5.15.7
azure-mgmt-containerregistry	10.1.0	libcap	2.67	pyqt5-sip	12.11.0
azure-mgmt-core	1.4.0	libcblas	3.9.0	pyrsistent	0.19.3
azure-mgmt-keyvault	10.2.1	libclang	15.0.7	pysocks	1.7.1
azure-mgmt-resource	21.2.1	libclang13	15.0.7	pyspark	3.3.1
azure-mgmt-storage	20.1.0	libcrc32c	1.1.2	python	3.10.10
azure-storage-blob	12.15.0	libcurl	2.3.3	python_abi	3.10
azure-storage-file-datalake	12.9.1	libcurl	7.88.1	python-dateutil	2.8.2

Library	Version	Library	Version	Library	Version
azureml-core	1.49.0	libdb	6.2.32	python-fastjsonschema	2.16.3
backcall	0.2.0	libdeflate	1.17	python-flatbuffers	23.1.21
backports	1.0	libebm	0.3.1	python-graphviz	0.20.1
backports-tempfile	1.0	libedit	3.1.20191231	python-json-logger	2.0.7
backports-weakref	1.0.post1	libev	4.33	pytorch	1.13.1
backports.functools_lru_cache	1.6.4	libevent	2.1.10	pytz	2022.7.1
bcrypt	3.2.2	libexpat	2.5.0	pyu2f	0.1.5
beautifulsoup4	4.11.2	libffi	3.4.2	pywin32-on-windows	0.1.0
bleach	6.0.0	libflac	1.4.2	pyyaml	6.0
blinker	1.6.1	libgcc-ng	12.2.0	pymq	25.0.2
brotli	1.0.9	libgcrypt	1.10.1	qt-main	5.15.8
brotli-bin	1.0.9	libgd	2.3.3	re2	2023.02.01
brotli-python	1.0.9	libgfortran-ng	12.2.0	readline	8.2
brotlipy	0.7.0	libgfortran5	12.2.0	regex	2022.10.31
bzip2	1.0.8	libglib	2.74.1	requests	2.28.2
c-ares	1.18.1	libgoogle-cloud	2.7.0	requests-oauthlib	1.3.1
ca-certificates	2022.12.7	libgpg-error	1.46	rfc3339-validator	0.1.4
cached_property	1.5.2	libgrpc	1.51.1	rfc3986-validator	0.1.1
cached-property	1.5.2	libhwloc	2.9.0	rsa	4.9
cachetools	5.3.0	libiconv	1.17	ruamel_yaml	0.15.80
cairo	1.16.0	liblapack	3.9.0	ruamel.yaml	0.17.21
certifi	2022.12.7	libllvm11	11.1.0	ruamel.yaml.clib	0.2.7
cffi	1.15.1	libllvm15	15.0.7	s2n	1.3.37
charset-normalizer	2.1.1	libnghttp2	1.52.0	salib	1.4.7
click	8.1.3	libnsl	2.0.0	scikit-learn	1.2.0
cloudpickle	2.2.1	libogg	1.3.4	scipy	1.10.1
colorama	0.4.6	libopenblas	0.3.21	seaborn	0.12.2
comm	0.1.3	libopus	1.3.1	seaborn-base	0.12.2
conda-package-handling	2.0.2	libpng	1.6.39	secretstorage	3.3.3

Library	Version	Library	Version	Library	Version
conda-package-streaming	0.7.0	libpq	15.2	send2trash	1.8.0
configparser	5.3.0	libprotobuf	3.21.12	setuptools	67.6.1
contextlib2	21.6.0	librsvg	2.54.4	shap	0.41.0
contourpy	1.0.7	libsndfile	1.2.0	sip	6.7.7
cryptography	40.0.1	libsodium	1.0.18	six	1.16.0
cycler	0.11.0	libsqLite	3.40.0	sleef	3.5.1
dash	2.9.2	libssh2	1.10.0	slicer	0.0.7
dash_cytoscape	0.2.0	libstdcxx-ng	12.2.0	smmap	3.0.5
dash-core-components	2.0.0	libsystemd0	253	snappy	1.1.10
dash-html-components	2.0.0	libthrift	0.18.0	sniffio	1.3.0
dash-table	5.0.0	libtiff	4.5.0	soupsieve	2.3.2.post1
databricks-cli	0.17.6	libtool	2.4.7	sqlalchemy	2.0.9
dbus	1.13.6	libudev1	253	sqlparse	0.4.3
debugpy	1.6.7	libutf8proc	2.8.0	stack_data	0.6.2
decorator	5.1.1	libuuid	2.38.1	statsmodels	0.13.5
defusedxml	0.7.1	libuv	1.44.2	synapseML-mlflow	1.0.14
dill	0.3.6	libvorbis	1.3.7	synapseML-utils	1.0.7
distlib	0.3.6	libwebp	1.2.4	tabulate	0.9.0
docker-py	6.0.0	libwebp-base	1.2.4	tbb	2021.8.0
entrypoints	0.4	libxcb	1.13	tenacity	8.2.2
et_xmlfile	1.1.0	libxgboost	1.7.1	tensorboard	2.11.2
executing	1.2.0	libxkbcommon	1.5.0	tensorboard-data-server	0.6.1
expat	2.5.0	libxml2	2.10.3	tensorboard-plugin-wit	1.8.1
fftw	3.3.10	libxslt	1.1.37	tensorflow	2.11.0
filelock	3.11.0	libzlib	1.2.13	tensorflow-base	2.11.0
flask	2.2.3	lightgbm	3.3.3	tensorflow-estimator	2.11.0
flask-compress	1.13	lime	0.2.0.1	termcolor	2.2.0
flatbuffers	22.12.06	llvm-openmp	16.0.1	terminado	0.17.1

Library	Version	Library	Version	Library	Version
flit-core	3.8.0	llvmlite	0.39.1	threadpoolctl	3.1.0
fluent-logger	0.10.0	lxml	4.9.2	tinycc2	1.2.1
font-ttf-dejavu-sans-mono	2.37	lz4-c	1.9.4	tk	8.6.12
font-ttf-inconsolata	3.000	markdown	3.4.1	toml	0.10.2
font-ttf-source-code-pro	2.038	markupsafe	2.1.2	toolz	0.12.0
font-ttf-ubuntu	0.83	matplotlib	3.6.3	tornado	6.2
fontconfig	2.14.2	matplotlib-base	3.6.3	tqdm	4.65.0
fonts-conda-ecosystem	1	matplotlib-inline	0.1.6	traitlets	5.9.0
fonts-conda-forge	1	mistune	2.0.5	treeinterpreter	0.2.2
fonttools	4.39.3	mkl	2022.2.1	typed-ast	1.4.3
freetype	2.12.1	mlflow-skinny	2.1.1	typing_extensions	4.5.0
fribidi	1.0.10	mpg123	1.31.3	typing_extensions	4.5.0
frozenlist	1.3.3	msal	1.21.0	tzdata	2023c
fsspec	2023.4.0	msal_extensions	1.0.0	unicodedata2	15.0.0
gast	0.4.0	msgpack	1.0.5	unixodbc	2.3.10
gdk-pixbuf	2.42.10	msrest	0.7.1	urllib3	1.26.14
geographiclib	1.52	msrestazure	0.6.4	virtualenv	20.19.0
geopy	2.3.0	multidict	6.0.4	wcwidth	0.2.6
gettext	0.21.1	multiprocess	0.70.14	webencodings	0.5.1
gevent	22.10.2	munkres	1.1.4	websocket-client	1.5.1
gflags	2.2.2	mypy	0.780	werkzeug	2.2.3
giflib	5.2.1	mypy-extensions	0.4.4	wheel	0.40.0
gitdb	4.0.10	mysql-common	8.0.32	widetsnbextension	4.0.7
gitpython	3.1.31	mysql-libs	8.0.32	wrapt	1.15.0
glib	2.74.1	nbclient	0.7.3	xcb-util	0.4.0
glib-tools	2.74.1	nbconvert-core	7.3.0	xcb-util-image	0.4.0
glog	0.6.0	nbformat	5.8.0	xcb-util-keysyms	0.4.0
google-auth	2.17.2	ncurses	6.3	xcb-util-renderutil	0.3.9
google-auth-oauthlib	0.4.6	ndg-httpsclient	0.5.1	xcb-util-wm	0.4.1
google-pasta	0.2.0	nest-asyncio	1.5.6	xgboost	1.7.1

Library	Version	Library	Version	Library	Version
graphite2	1.3.13	nspr	4.35	xkeyboard-config	2.38
graphviz	2.50.0	nss	3.89	xorg-kbproto	1.0.7
greenlet	2.0.2	numba	0.56.4	xorg-libice	1.0.10
grpcio	1.51.1	numpy	1.23.5	xorg-libsm	1.2.3
gson	0.0.3	oauthlib	3.2.2	xorg-libx11	1.8.4
gst-plugins-base	1.22.0	openjpeg	2.5.0	xorg-libxau	1.0.9
gstreamer	1.22.0	openpyxl	3.1.0	xorg-libxdmcp	1.1.3
gstreamer-orc	0.4.33	openssl	3.1.0	xorg-libxext	1.3.4
gtk2	2.24.33	opt_einsum	3.3.0	xorg-libxrender	0.9.10
gts	0.7.6	orc	1.8.2	xorg-renderproto	0.11.1
h5py	3.8.0	packaging	21.3	xorg-xextproto	7.3.0
harfbuzz	6.0.0	pandas	1.5.3	xorg-xproto	7.0.31
hdf5	1.14.0	pandasql	0.7.3	xz	5.2.6
html5lib	1.1	pandocfilters	1.5.0	yaml	0.2.5
humanfriendly	10.0	pango	1.50.14	yarl	1.8.2
icu	70.1	paramiko	2.12.0	zeromq	4.3.4
idna	3.4	parquet-cpp	1.5.1	zipp	3.15.0
imageio	2.25.0	parso	0.8.3	zlib	1.2.13
importlib_metadata	5.2.0	pathos	0.3.0	zope.event	4.6
importlib_resources	5.12.0	pathspec	0.11.1	zope.interface	6.0
importlib-metadata	5.2.0	patsy	0.5.3	zstandard	0.19.0
interpret	0.3.1	pcre2	10.40	zstd	1.5.2
interpret-core	0.3.1	pexpect	4.8.0		

## Default level packages for R libraries

Below you can find the table with listing all the default level packages for R and their respective versions.

Library	Version	Library	Version	Library	Version
askpass	1.1	highcharter	0.9.4	readr	2.1.3
assertthat	0.2.1	highr	0.9	readxl	1.4.1

Library	Version	Library	Version	Library	Version
backports	1.4.1	hms	1.1.2	recipes	1.0.3
base64enc	0.1-3	htmltools	0.5.3	rematch	1.0.1
bit	4.0.5	htmlwidgets	1.5.4	rematch2	2.1.2
bit64	4.0.5	httpcode	0.3.0	remotes	2.4.2
blob	1.2.3	httpuv	1.6.6	reprex	2.0.2
brew	1.0-8	httr	1.4.4	reshape2	1.4.4
brio	1.1.3	ids	1.0.1	rjson	0.2.21
broom	1.0.1	igraph	1.3.5	rlang	1.0.6
bslib	0.4.1	infer	1.0.3	rlist	0.4.6.2
cachem	1.0.6	ini	0.3.1	rmarkdown	2.18
callr	3.7.3	ipred	0.9-13	RODBC	1.3-19
caret	6.0-93	isoband	0.2.6	roxygen2	7.2.2
cellranger	1.1.0	iterators	1.0.14	rprojroot	2.0.3
cli	3.4.1	jquerylib	0.1.4	rsample	1.1.0
clipr	0.8.0	jsonlite	1.8.3	rstudioapi	0.14
clock	0.6.1	knitr	1.41	rversions	2.1.2
colorspace	2.0-3	labeling	0.4.2	rvest	1.0.3
commonmark	1.8.1	later	1.3.0	sass	0.4.4
config	0.3.1	lava	1.7.0	scales	1.2.1
conflicted	1.1.0	lazyeval	0.2.2	selectr	0.4-2
coro	1.0.3	lhs	1.1.5	sessioninfo	1.2.2
cpp11	0.4.3	lifecycle	1.0.3	shiny	1.7.3
crayon	1.5.2	lightgbm	3.3.3	slider	0.3.0
credentials	1.3.2	listenv	0.8.0	sourcetools	0.1.7
crosstalk	1.2.0	lobstr	1.1.2	sparklyr	1.7.8
curl	1.3	lubridate	1.9.0	SQUAREM	2021.1
curl	4.3.3	magrittr	2.0.3	stringi	1.7.8
data.table	1.14.6	maps	3.4.1	stringr	1.4.1
DBI	1.1.3	memoise	2.0.1	sys	3.4.1
dbplyr	2.2.1	mime	0.12	systemfonts	1.0.4

Library	Version	Library	Version	Library	Version
desc	1.4.2	miniUI	0.1.1.1	testthat	3.1.5
devtools	2.4.5	modeldata	1.0.1	textshaping	0.3.6
dials	1.1.0	modelenv	0.1.0	tibble	3.1.8
DiceDesign	1.9	ModelMetrics	1.2.2.2	tidymodels	1.0.0
diffobj	0.3.5	modelr	0.1.10	tidyverse	1.2.1
digest	0.6.30	munsell	0.5.0	tidyselect	1.2.0
downlit	0.4.2	numDeriv	2016.8-1.1	tidyverse	1.3.2
dplyr	1.0.10	openssl	2.0.4	timechange	0.1.1
dtplyr	1.2.2	parallelly	1.32.1	timeDate	4021.106
e1071	1.7-12	parsnip	1.0.3	tinytex	0.42
ellipsis	0.3.2	patchwork	1.1.2	torch	0.9.0
evaluate	0.18	pillar	1.8.1	triebeard	0.3.0
fansi	1.0.3	pkgbuild	1.4.0	TTR	0.24.3
farver	2.1.1	pkgconfig	2.0.3	tune	1.0.1
fastmap	1.1.0	pkgdown	2.0.6	tzdb	0.3.0
fontawesome	0.4.0	pkgload	1.3.2	urlchecker	1.0.1
forcats	0.5.2	plotly	4.10.1	urllibs	1.7.3
foreach	1.5.2	plyr	1.8.8	usethis	2.1.6
forge	0.2.0	praise	1.0.0	utf8	1.2.2
fs	1.5.2	prettyunits	1.1.1	uuid	1.1-0
furrr	0.3.1	pROC	1.18.0	vctrs	0.5.1
future	1.29.0	processx	3.8.0	viridisLite	0.4.1
future.apply	1.10.0	prodlm	2019.11.13	vroom	1.6.0
gargle	1.2.1	profvis	0.3.7	waldo	0.4.0
generics	0.1.3	progress	1.2.2	warp	0.2.0
gert	1.9.1	progressr	0.11.0	whisker	0.4
ggplot2	3.4.0	promises	1.2.0.1	withr	2.5.0
gh	1.3.1	proxy	0.4-27	workflows	1.1.2
gistr	0.9.0	pryr	0.1.5	workflowsets	1.0.0
gitcreds	0.1.2	ps	1.7.2	xfun	0.35

Library	Version	Library	Version	Library	Version
globals	0.16.2	purrr	0.3.5	xgboost	1.6.0.1
glue	1.6.2	quantmod	0.4.20	XML	3.99-0.12
googledrive	2.0.0	r2d3	0.2.6	xml2	1.3.3
googlesheets4	1.0.1	R6	2.5.1	xopen	1.0.0
gower	1.0.0	ragg	1.2.4	xtable	1.8-4
GPfit	1.0-8	rappdirs	0.3.3	xts	0.12.2
gttable	0.3.1	rbokeh	0.5.2	yaml	2.3.6
hardhat	1.2.0	rcmdcheck	1.4.0	yardstick	1.1.0
haven	2.5.1	RColorBrewer	1.1-3	zip	2.2.2
hexbin	1.28.2	Rcpp	1.0.9	zoo	1.8-11

## Migration between different Apache Spark Versions

Migrating your workloads to Fabric Runtime 1.1 (Apache Spark 3.3) from an older version of Apache Spark involves a series of steps to ensure a smooth migration. This guide outlines the necessary steps to help you migrate efficiently and effectively.

1. Review Fabric Runtime 1.1 release notes, including checking the components and default-level packages included into the runtime, to understand the new features, improvements.
2. Check compatibility of your current setup and all related libraries, including dependencies and integrations. Review the migration guides to identify potential breaking changes:
  - [Review Spark Core migration guide ↗](#)
  - [Review SQL, Datasets and DataFrame migration guide ↗](#)
  - If your solution is Apache Spark Structure Streaming related, [review Structured Streaming migration guide ↗](#)
  - If you use PySpark, [review Pyspark migration guide ↗](#)
  - If you migrate code from Koalas to PySpark, [review Koalas to pandas API on Spark migration guide ↗](#)
3. Move your workloads to Fabric and ensure that you have backups of your data and configuration files in case you need to revert to the previous version.
4. Update any dependencies that may be impacted by the new version of Apache Spark or other Fabric Runtime 1.1 related components. This could include third-party libraries or connectors. Make sure to test the updated dependencies in a staging environment before deploying to production
5. Update Apache Spark Configuration on your workload. This could include updating configuration settings, adjusting memory allocations, and modifying any deprecated

configurations.

6. Modify your Apache Spark applications (notebooks and Apache Spark Jobs Definitions) to use the new APIs and features introduced in Fabric Runtime 1.1 and Apache Spark 3.3. This may involve updating your code to accommodate any deprecated or removed APIs, and refactoring your applications to take advantage of performance improvements and new functionalities.
7. Thoroughly test your updated applications in a staging environment to ensure compatibility and stability with Apache Spark 3.3. Perform performance testing, functional testing, and regression testing to identify and resolve any issues that may arise during the migration process.
8. After validating your applications in a staging environment, deploy the updated applications to your production environment. Monitor the performance and stability of your applications after the migration to identify any issues that need to be addressed.
9. Update your internal documentation and training materials to reflect the changes introduced in Fabric Runtime 1.1. Ensure that your team members are familiar with the new features and improvements to maximize the benefits of the migration.

# How to create custom Spark pools in Microsoft Fabric

Article • 05/23/2023

In this document, we'll explain how to create custom Apache Spark pools in Microsoft Fabric for your analytics workloads. Apache Spark pools enable users to create tailored compute environments based on their specific requirements, ensuring optimal performance and resource utilization.

## Important

Microsoft Fabric is currently in PREVIEW. This information relates to a prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, expressed or implied, with respect to the information provided here.

Users specify the minimum and maximum nodes for autoscaling. Based on which, the system dynamically acquires and retires nodes as the job's compute requirements change. It results in efficient scaling and improved performance. Furthermore, the dynamic allocation of executors in Spark pools alleviates the need for manual executor configuration. Instead, the system adjusts the number of executors depending on the data volume and job-level compute needs. This way, it enables you to focus on your workloads without worrying about performance optimization and resource management.

## Note

To create a custom spark pool, you must have admin access to the workspace. The capacity admin should have enabled the **Customized workspace pools** option in the **Spark Compute** section of **Capacity Admin settings**. To learn more, see [Spark Compute Settings for Fabric Capacities](#).

## Create custom Spark pools

To create or manage the Spark Pool associated with your workspace:

1. Go to your workspace and choose the **Workspace settings**:

2. Then, select the **Data Engineering/Science** option to expand the menu. Navigate to the **Spark Compute** option from the left-hand menu:

3. Select the **New Pool** option. From the **Create Pool** menu, name your Spark pool. Select the **Node family**, and **Node size** from the available sizes Small, Medium, Large, X-Large and XX-Large based on compute requirements for your workloads.



## Create pool

Spark pool name \*

Node family

Node size

Small

Medium

Large

X-Large

XX-Large

Enable autoscale



4. You can also set the minimum node configuration for your custom pools to 1. Because the Fabric Spark provides restorable availability for clusters with single node, you do not have to worry about job failures, loss of session during failures, or over paying on compute for smaller spark jobs.
5. You can also enable or disable autoscaling for your custom Spark pools. When autoscaling is enabled, the pool will dynamically acquire new nodes up to the maximum node limit specified by the user, and then retire them after job execution. This feature ensures better performance by adjusting resources based on the job requirements. You are allowed to size the nodes, which fit within the capacity units purchased as part of the Fabric capacity SKU.

## Edit pool

Spark pool name \*

customlargepool

Node family

Memory optimized

Node size

Large

Autoscale

If enabled, your Apache Spark pool will automatically scale up and down based on the amount of activity.

Enable autoscale



Dynamically allocate executors

Enable allocate



6. You can also choose to enable dynamic executor allocation for your Spark pool, which automatically determines the optimal number of executors within the user-specified maximum bound. This feature adjusts the number of executors based on data volume, resulting in improved performance and resource utilization.

7. These custom pools have a default auto-pause duration of 2 minutes. Once the auto-pause duration is reached, the session expires and the clusters are unallocated. You are charged based on the number of nodes and the duration for which the custom spark pools are used.

## Next steps

- Learn more from the Apache Spark [public documentation ↗](#).
- Get Started with Data Engineering/Science Admin Settings for your Fabric Workspace

# What is autotune for Apache Spark configurations in Fabric and how to enable and disable it?

Article • 05/23/2023

Autotune automatically tunes Apache Spark configurations to minimize workload execution time and optimizes workloads. It empowers you to achieve more with less. This feature reduces execution time and surpasses the gains accomplished by manually tuned workloads by experts, which necessitate considerable effort and experimentation.

It leverages historical data execution from your workloads to iteratively learn the optimal configurations for a given workload and its execution time.

## Important

Microsoft Fabric is currently in PREVIEW. This information relates to a prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, expressed or implied, with respect to the information provided here.

## Query tuning

Currently, autotune configures three query-level of Apache Spark configurations:

- `spark.sql.shuffle.partitions` - configures the number of partitions to use when shuffling data for joins or aggregations. Default is 200.
- `spark.sql.autoBroadcastJoinThreshold` - configures the maximum size in bytes for a table that will be broadcasted to all worker nodes when performing a join. Default is 10 MB.
- `spark.sql.files.maxPartitionBytes` - the maximum number of bytes to pack into a single partition when reading files. Works for Parquet, JSON and ORC file-based sources. Default is 128 MB.

Since there's no historical data available during the first run of autotune, configurations will be set based on a baseline model. This model relies on heuristics related to the content and structure of the workload itself. However, as the same query or workload is run repeatedly, we'll observe increasingly significant improvements from autotune. As the results of previous runs are used to fine-tune the model and tailor it to a specific workspace or workload.

### Note

As the algorithm explores various configurations, you may notice minor differences in results. This is expected, as autotune operates iteratively and improves with each repetition of the same query.

## Configuration tuning algorithm overview

For the first run of the query, upon submission, a machine learning (ML) model initially trained using standard open-source benchmark queries (e.g., TPC-DS) will guide the search around the neighbors of the current setting (starting from the default). Among the neighbor candidates, the ML model selects the best configuration with the shortest predicted execution time. In this run, the "centroid" is the default config, around which the autotune generates new candidates.

Based on the performance of the second run per suggested configuration, we retrain the ML model by adding the new observation from this query, and update the centroid by comparing the performance of the last two runs. If the previous run is better, the centroid will be updated in the inverse direction of the previous update (similar to the momentum approach in DNN training); if the new run is better, the latest configuration setting becomes the new centroid. Iteratively, the algorithm will gradually search in the direction with better performance.

## Enable or disable autotune

Autotune is disabled by default and it's controlled by Apache Spark Configuration Settings. Easily enable Autotune within a session by running the following code in your notebook or adding it in your spark job definition code:

```
Spark SQL
SQL
%%sql
SET spark.ms.autotune.queryTuning.enabled=TRUE
```

To verify and confirm its activation, use the following commands:

```
Spark SQL
```

SQL

```
%%sql  
GET spark.ms.autotune.queryTuning.enabled
```

To disable Autotune, execute the following commands:

Spark SQL

SQL

```
%%sql  
SET spark.ms.autotune.queryTuning.enabled=FALSE
```

## Transparency note

Microsoft follows Responsible AI Standard and this transparency note aims to provide clear documentation defining the intended uses of Autotune and the evidence that the feature is fit for purpose before the service becomes externally available. We understand the importance of transparency and ensuring that our customers have the necessary information to make informed decisions when using our services.

## Intended uses of the Autotune

The primary goal of Autotune is to optimize the performance of Apache Spark workloads by automating the process of Apache Spark configuration tuning. The system is designed to be used by data engineers, data scientists, and other professionals who are involved in the development and deployment of Apache Spark workloads. The intended uses of the Autotune include:

- Automatic tuning of Apache Spark configurations to minimize workload execution time to accelerate development process
- Reducing the manual effort required for Apache Spark configuration tuning
- Leveraging historical data execution from workloads to iteratively learn optimal configurations

## Evidence that the Autotune is fit for purpose

To ensure that Autotune meets the desired performance standards and is fit for its intended use, we have conducted rigorous testing and validation. The evidence includes:

1. Thorough internal testing and validation using various Apache Spark workloads and datasets to confirm the effectiveness of the autotuning algorithms
2. Comparisons with alternative Apache Spark configuration optimization techniques, demonstrating the performance improvements and efficiency gains achieved by Autotune
3. Customer case studies and testimonials showcasing successful applications of Autotune in real-world projects
4. Compliance with industry-standard security and privacy requirements, ensuring the protection of customer data and intellectual property

We want to assure that we prioritize data privacy and security. Your data will only be used to train the model that serves your specific workload. We take stringent measures to ensure that no sensitive information is used in our storage or training processes.

# Concurrency limits and queueing in Microsoft Fabric Spark

Article • 05/23/2023

Applies to:  Data Engineering and Data Science in Microsoft Fabric

Microsoft Fabric allows allocation of compute units through capacity, which is a dedicated set of resources that is available at a given time to be used. Capacity defines the ability of a resource to perform an activity or to produce output. Different items consume different capacity at a certain time. Microsoft Fabric offers capacity through the Fabric SKUs and trials. For more information, see [What is capacity?](#)

## Important

Microsoft Fabric is currently in PREVIEW. This information relates to a prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, expressed or implied, with respect to the information provided here.

When users create a Microsoft Fabric capacity on Azure, they get to choose a capacity size based on their analytics workload size. In Spark, users get two spark VCores for every capacity unit they get reserved as part of their SKU.

*One Capacity Unit = Two Spark VCores*

Once the capacity is purchased, admins can create workspaces within the capacity in Microsoft Fabric. The Spark VCores associated with the capacity is shared among all the Spark-based items like notebooks, spark job definitions, and the lakehouse created in these workspaces.

## Concurrency throttling and queueing

The following section lists various numerical limits for Spark workloads based on Microsoft Fabric capacity SKUs:

Capacity SKU	Equivalent Power BI SKU	Capacity Units	Equivalent Spark VCores	Max Concurrent Jobs	Queue Limit
F2	-	2	4	1	4
F4	-	4	8	1	4

Capacity SKU	Equivalent Power BI SKU	Capacity Units	Equivalent Spark Vcores	Max Concurrent Jobs	Queue Limit
F8	-	8	16	2	8
F16	-	16	32	5	20
F32	-	32	64	10	40
F64	P1	64	128	20	80
Fabric Trial	P1	64	128	5	-
F128	P2	128	256	40	160
F256	P3	256	512	80	320
F512	P4	512	1024	160	640

The queueing mechanism is a simple FIFO-based queue, which checks for available job slots and automatically retries the jobs once the capacity has become available. As there are different items like notebooks, spark job definition, and lakehouse which users could use in any workspace. As the usage varies across different enterprise teams, users could run into starvation scenarios where there is dependency on only type of item, such as a spark job definition. This could result in users sharing the capacity from running a notebook-based job or any lakehouse based operation like load to table.

To avoid these blocking scenarios, Microsoft Fabric applies a **Dynamic reserve based throttling** for jobs from these items. Notebook and lakehouse based jobs being more interactive and real-time are classified as **interactive**. Whereas Spark job definition is classified as **batch**. As part of this dynamic reserve, minimum and maximum reserve bounds are maintained for these job types. The reserves are mainly to address use cases where an enterprise team could experience peak usage scenarios having their entire capacity consumed through batch jobs. During those peak hours, users are blocked from using interactive items like notebooks or lakehouse. With this approach, every capacity gets a minimum reserve of 30% of the total jobs allocated for interactive jobs (5% for lakehouse and 25% for notebooks) and a minimum reserve of 10% for batch jobs.

Job Type	Item	Min %	Max %
Batch	Spark Job Definition	10	70
Interactive	Interactive Min and Max	30	90

Job Type	Item	Min %	Max %
	Notebook	25	85
	Lakehouse	5	65

When they exceed these reserves and when the capacity is at its maximum utilization, interactive jobs like notebooks and lakehouse are throttled with the message *HTTP Response code 430: Unable to submit this request because all the available capacity is currently being used. Cancel a currently running job, increase your available capacity, or try again later.*

With queueing enabled, batch jobs like Spark Job Definitions get added to the queue and are automatically retried when the capacity is freed up.

 **Note**

The jobs have a queue expiration period of 24 hours, after which they are cancelled and users would have to resubmit them for job execution.

## Next steps

- [Get Started with Data Engineering/Science Admin Settings for your Fabric Workspace](#)
- [Learn about the Spark Compute for Fabric Data Engineering/Science experiences](#)

# What is an Apache Spark job definition?

Article • 05/23/2023

An Apache Spark Job Definition is a Microsoft Fabric code item that allows you to submit batch/streaming job to Spark cluster. By uploading the binary files from compilation output of different languages, jar from Java for example, you can apply different transformation logic to the data hosted on lakehouse. Besides the binary file, you can further customize the behavior of the job by uploading additional libraries and command line arguments.

## Important

Microsoft Fabric is currently in PREVIEW. This information relates to a prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, expressed or implied, with respect to the information provided here.

To run a Spark job definition, you must have at least one lakehouse associated with it. This default lakehouse context serves as the default file system for Spark runtime. For any Spark code using relative path to read/write data, the data is served from the default lakehouse.

## Tip

To run the Spark job definition item, main definition file and default lakehouse context are required. If you don't have a lakehouse, you can create one by following the steps in [Create a lakehouse](#).

## Important

The Spark job definition item is currently in PREVIEW.

## Next steps

In this overview, you get a basic understanding of a Spark job definition. Advance to the next article to learn how to create and get started with your own Spark job definition:

- To get started with Microsoft Fabric, see [Creating an Apache Spark job definition](#).

# How to create an Apache Spark job definition in Fabric

Article • 05/23/2023

In this tutorial, learn how to create a Spark job definition in Microsoft Fabric.

## ⓘ Important

Microsoft Fabric is currently in PREVIEW. This information relates to a prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, expressed or implied, with respect to the information provided here.

## Prerequisites

To get started, you need the following prerequisites:

- A Microsoft Fabric tenant account with an active subscription. [Create an account for free](#).

## 💡 Tip

To run the Spark job definition item, main definition file and default lakehouse context are required. If you don't have a lakehouse, you can create one by following the steps in [Create a lakehouse](#).

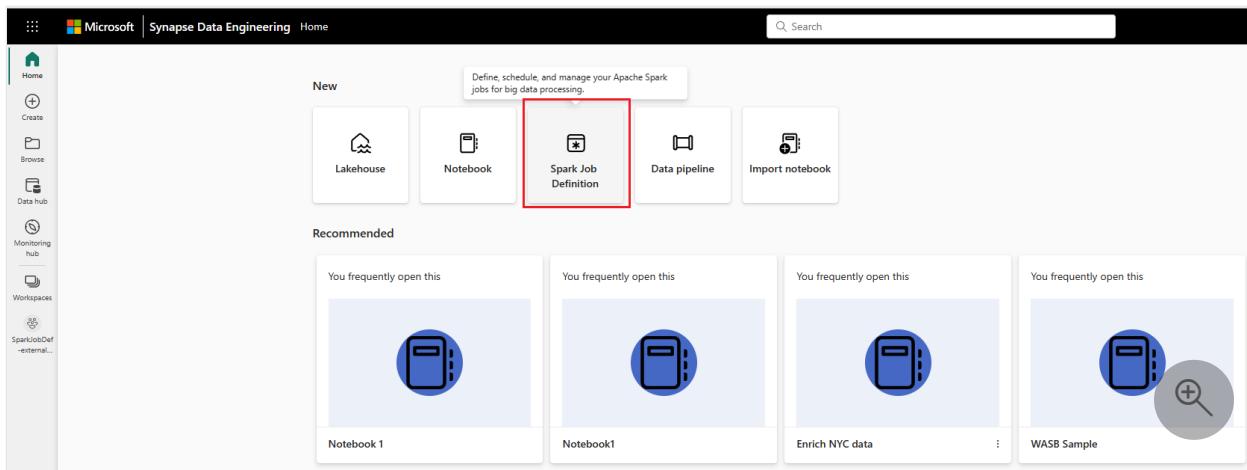
## Create a Spark job definition

The Spark job definition creation process is quick and simple and there are several ways to get started.

## Options to create a Spark job definition

There are a few ways you can get started with the creation process:

- **Data engineering homepage:** You can easily create a Spark job definition through the **Spark job definition** card under the **New** section in the homepage.



- **Workspace view:** You can also create a Spark job definition through the **Workspace** view when you are on the **Data Engineering** experience by using the **New** dropdown.

	Owner	Refreshed	Next refresh	Endorsement	Sensitivity	Included in app
Dataset (default)	SparkJobDef-external...	1/9/23, 2:22:01 PM	N/A	—	—	Confidential\Microsoft...
Lakehouse001	SQL endpoint	SparkJobDef-external...	—	N/A	—	—
Lakehouse001	Lakehouse	Jessie Irwin	—	—	—	Confidential\Microsoft...

- **Create Hub:** Another entry point to create a Spark job definition is in the **Create Hub** page under **Data Engineering**.

A name would be required to create a Spark job definition. The name must be unique within the current workspace. The newly created Spark Job definition will be created under the current workspace you are in.

## Create a Spark job definition for PySpark (Python)

To create a Spark job definition for PySpark, follow these steps:

1. Create a new Spark job definition.
2. Select **PySpark (Python)** from the **Language** dropdown.
3. Upload the main definition file as **.py** file. The main definition file is the file that contains the application logic of this *job*. *Main* definition file is mandatory to run a Spark job. For each Spark Job Definition, you can only upload one main definition file.  
Beside uploading from local desktop, you can also upload from existing Azure Data Lake Storage Gen2 by providing the full abfss path of the file. For example, abfss://your-storage-account-name.dfs.core.windows.net/your-file-path.
4. Upload Reference files as **.py** file. the Reference files are the python modules that are imported by the main definition file. Similar as uploading main definition file, you can also upload from existing Azure Data Lake Storage Gen2 by providing the full abfss path of the file. Multiple reference files are supported.

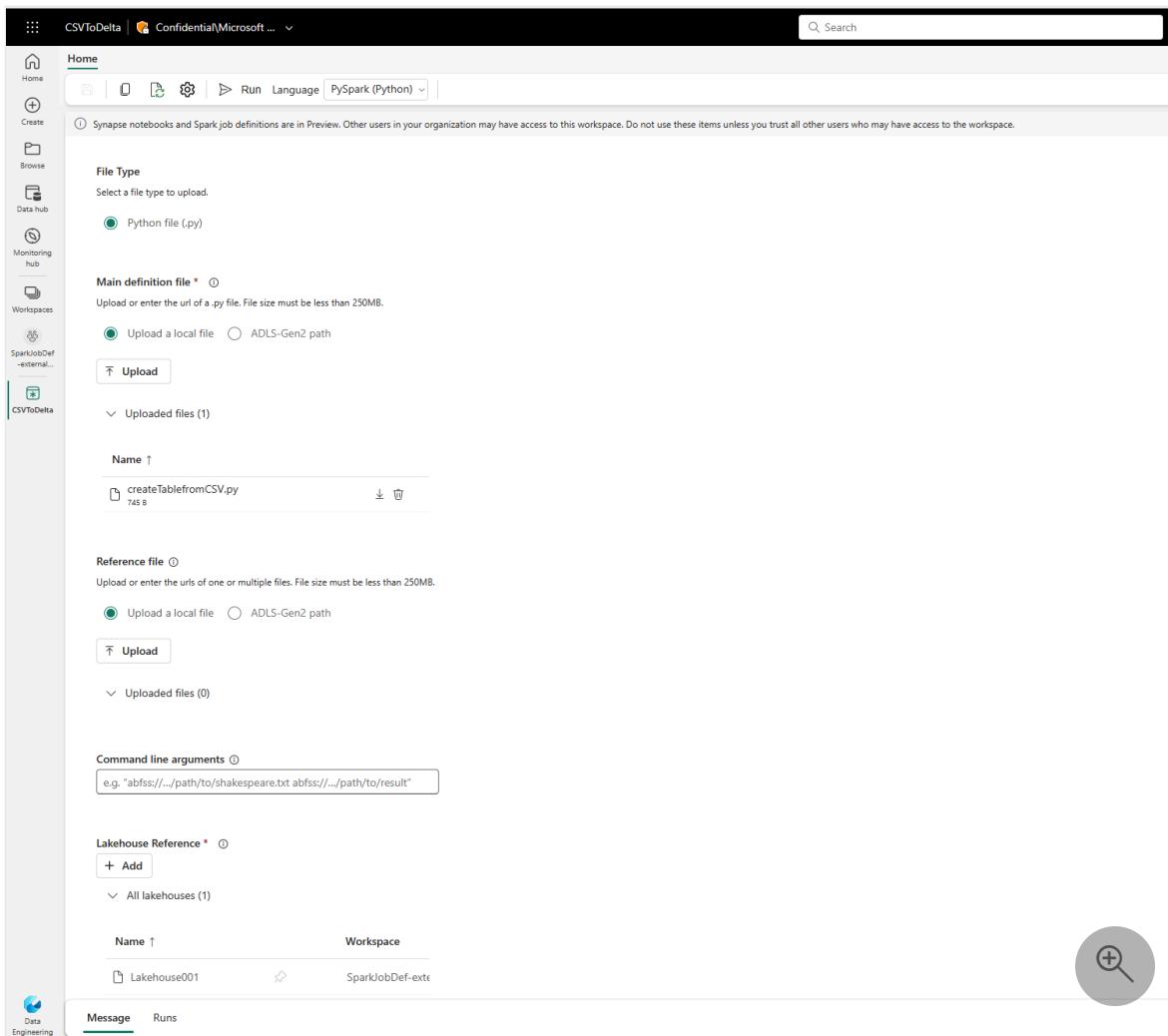
#### Tip

If ADLS-gen2 path is used, to make sure the file is accessible, The user account which is used to run the job should be assigned with proper permission to the storage account. There are two suggested way to do this:

- Assign the user account as Contributor role to the storage account.
- Grant Read and Execution permission to the user account on the file via Azure Data Lake Storage Gen2 Access Control List (ACL)

For manually run, the account of current login user would be used to run the job

5. Provide command line arguments to the job if needed. please use space as splitter to separate the arguments.
6. Add the lakehouse reference to the job. You must have at least one lakehouse reference added to the job. This lakehouse is the default lakehouse context for the job. Multiple lakehouse references are supported. For the non-default Lakehouse, you can find its name and full OneLake URL in the Spark Settings page.



In this example, we've done the following:

- Created a Spark job definition named **CSVToDelta** for PySpark
- Uploaded the *createTablefromCSV.py* file as the main definition file
- Added the lakehouse references *LH001* and *LH002* to the job
- Made *LH001* the default lakehouse context

## Create a Spark job definition for Scala/Java

To create a Spark job definition for Scala/Java, follow these steps:

1. Select **Spark(Scala/Java)** from the **Language** dropdown.
2. Upload the main definition file as .jar file. The main definition file is the file that contains the application logic of this job. A main definition file is mandatory to run a Spark Job. Provide the Main class name.
3. Upload Reference files as .jar file. the Reference files are the files that are referenced/imported by the main definition file.
4. Provides command line arguments to the job if needed.

5. Add the lakehouse reference to the job. You must have at least one lakehouse reference added to the job. This lakehouse is the default lakehouse context for the job.

## Create a Spark job definition for R

To create a Spark job definition for SparkR(R), follow these steps:

1. Select **SparkR(R)** from the **Language** dropdown.
2. Upload the main definition file as .R file. The main definition file is the file that contains the application logic of this job. A main definition file is mandatory to run a Spark Job.
3. Upload Reference files as .R file. the Reference files are the files that are referenced/imported by the main definition file.
4. Provides command line arguments to the job if needed.
5. Add the lakehouse reference to the job. You must have at least one lakehouse reference added to the job. This lakehouse is the default lakehouse context for the job.

 **Note**

The Spark job definition will be created under the current workspace you are in.

## Options to customize Spark job definition

There are a few options to further customize the execution of Spark job definition

- **Spark Compute:** Within the **Spark Compute** tab, you can see the Runtime Version which is the version of Spark that will be used to run the job. You can also see the Spark configuration settings that will be used to run the job. You can customize the Spark configuration settings by clicking on the **Add** button.

The screenshot shows the 'CSVToDelta' Spark Job Definition page. On the left, there's a sidebar with tabs: About, Sensitivity label, Endorsement, Schedule, Spark compute (which is selected and highlighted in grey), and Optimization. A search bar is at the top left. The main content area has two sections: 'Runtime Version' and 'Spark properties'. Under 'Runtime Version', it says 'Runtime family defines which version of Spark your Spark pool will use.' with a link 'Learn more about Runtime Version'. A dropdown menu shows '1.0 (Spark 3.2, Delta 1.2)'. Under 'Spark properties', it says 'Spark properties allows you to define many Spark runtime properties.' with a link 'Learn more about Spark properties'. There are buttons for '+ Add' and 'Delete'. At the bottom right of the main area is a 'Save' button and a circular icon with a plus sign.

- **Optimization:** Within the **Optimization** tab, you can enable and set up the Retry Policy for the job. When enabled, the job will be retried if it fails. You can also set the maximum number of retries and the interval between retries. For each attempt of retry, the job will be restarted, please make sure the job is idempotent.

This screenshot shows the 'Retry Policy' configuration within the 'Optimization' tab. It includes:

- A toggle switch labeled 'Off' for enabling the retry policy.
- A section for 'Maximum retry attempts' with a slider set to '1' and a checkbox for 'Allow unlimited attempts'.
- A section for 'Time between each attempt' with a slider for '0' and a dropdown for 'seconds'.
- A 'Apply' button at the bottom.

A circular icon with a plus sign is located at the bottom right of the configuration area.

## Next steps

- Run an Apache Spark job definition

# Schedule and run an Apache Spark job definition

Article • 05/23/2023

In this tutorial, learn how to run a Microsoft Fabric Spark job definition item and monitor the job.

## ⓘ Important

Microsoft Fabric is currently in PREVIEW. This information relates to a prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, expressed or implied, with respect to the information provided here.

## Prerequisites

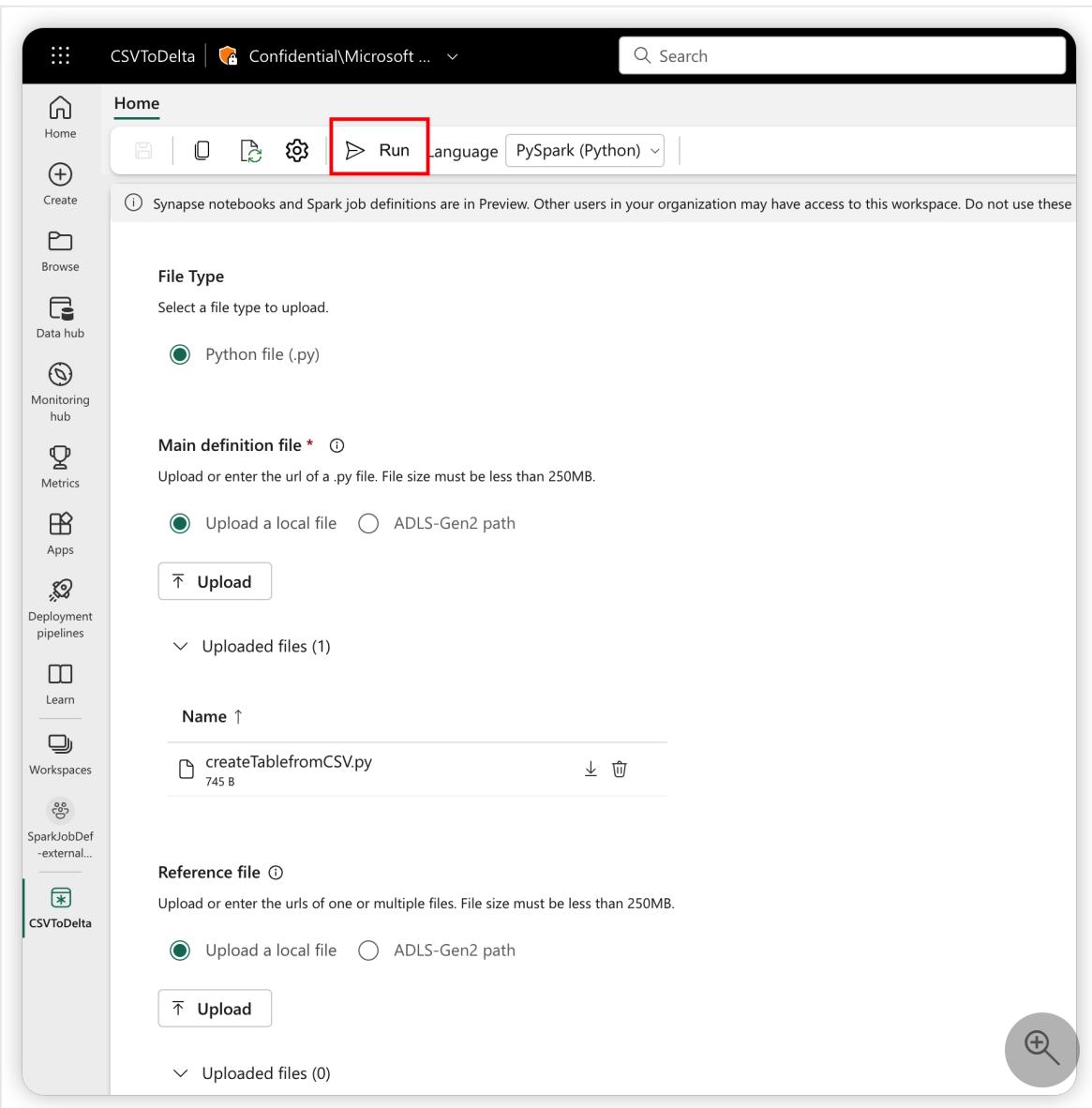
To get started, you must have the following prerequisites:

- A Microsoft Fabric tenant account with an active subscription. [Create an account for free](#).
- Understand the Spark job definition: [What is an Apache Spark job definition?](#)
- Create a Spark job definition: [How to create an Apache Spark job definition](#).

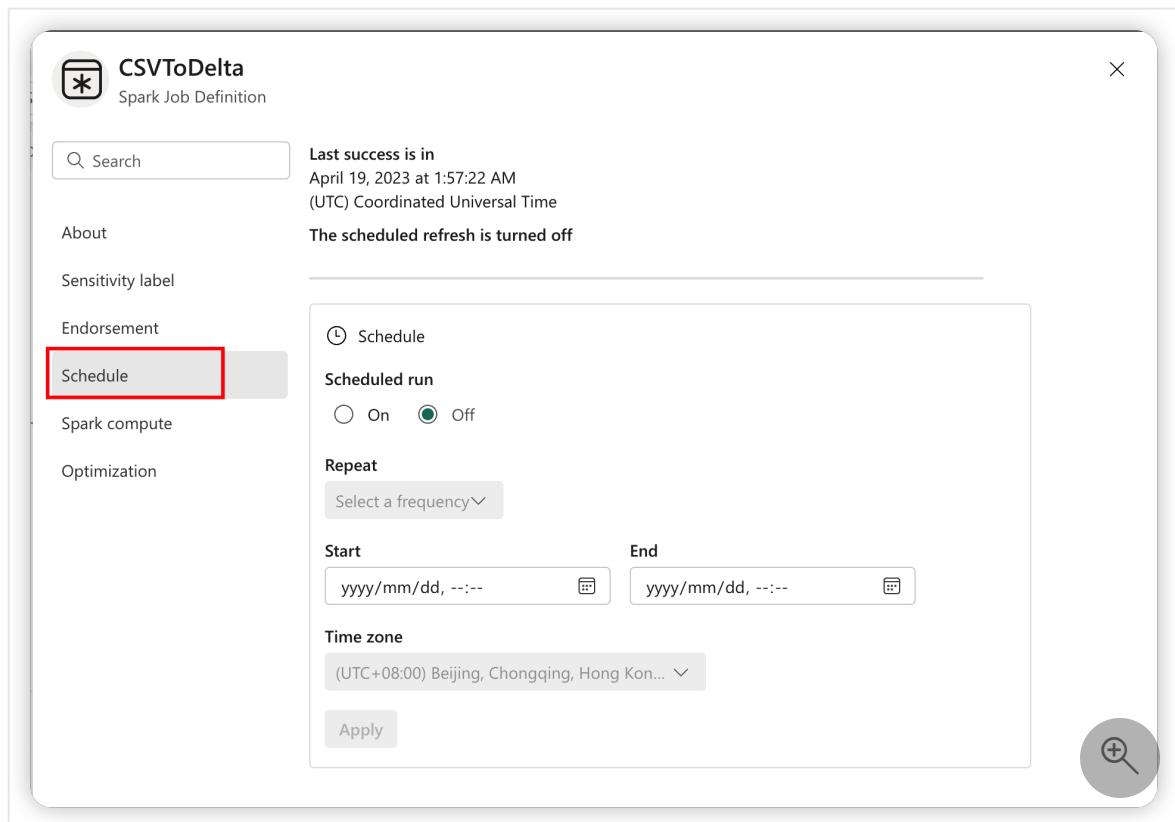
## How to run a Spark job definition

There are two ways a user could run a Spark job definition:

- Run a Spark job definition item manually by clicking the **Run** button on the Spark job definition item.



- Schedule a Spark job definition item by setting up the schedule plan under the **Settings** tab. Select **Settings** on the toolbar, then select the **Schedule** tab.



## ⓘ Important

To run a Spark job definition, it must have the main definition file and the default lakehouse context.

## 💡 Tip

For the run triggered by the "Run" button, the account of current login user will be used to submit the job. For the run triggered by the schedule plan, the account of the user who setup the schedule plan will be used to submit the job.

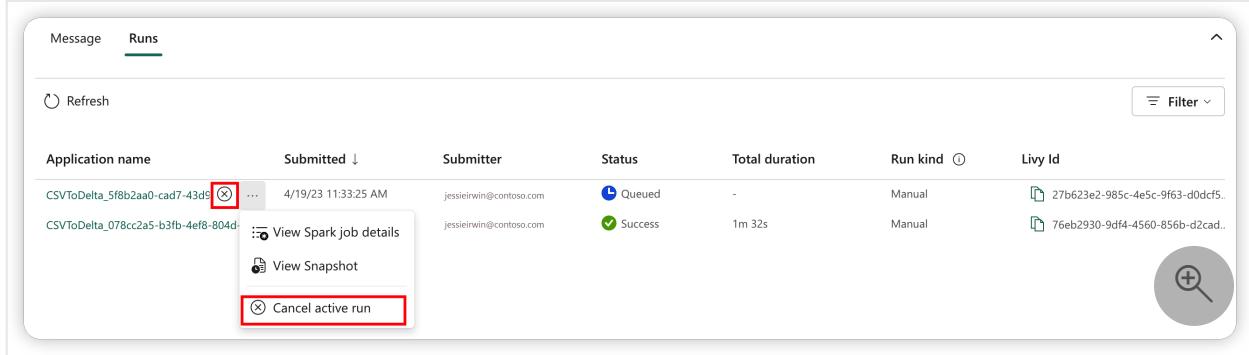
Once you've submitted the run, after three to five seconds, a new row appears under the **Runs** tab. The row shows details about your new run. The **Status** column shows the near real-time status of the job and the **Run Kind** column shows if the job is manual or scheduled.

Application name	Submitted ↓	Submitter	Status	Total duration	Run kind ⓘ	Livy Id
CSVToDelta_078cc2a5-b3fb-4ef8-804d-...	4/19/23 9:57:25 AM	jessieirwin@contoso.com	✓ Success	1m 32s	Manual	76eb2930-9df4-4560-856b-d2cad...

For the detail of how to monitor the job, see [Monitor a Spark job](#).

## How to cancel a running job

Once the job is submitted, you can cancel the job by clicking the **Cancel** button on the Spark job definition item from the job list



Application name	Submitted ↓	Submitter	Status	Total duration	Run kind ⓘ	Livy Id
CSVToDelta_5f8b2aa0-cad7-43d5...	4/19/23 11:32:5 AM	jessieirwin@contoso.com	queued	-	Manual	27b623e2-985c-4e5c-9f63-d0dcf5...
CSVToDelta_078cc2a5-b3fb-4ef8-804d...	4/19/23 11:32:5 AM	jessieirwin@contoso.com	success	1m 32s	Manual	76eb2930-9df4-4560-856b-d2cad...

## Next steps

- Advanced capabilities: Microsoft Apache Spark utilities

# Introduction of Fabric MSSparkUtils

Article • 05/23/2023

Microsoft Spark Utilities (MSSparkUtils) is a built-in package to help you easily perform common tasks. You can use MSSparkUtils to work with file systems, to get environment variables, to chain notebooks together, and to work with secrets. MSSparkUtils are available in PySpark (Python) Scala, SparkR notebooks and Microsoft Fabric pipelines.

## ⓘ Important

Microsoft Fabric is currently in PREVIEW. This information relates to a prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, expressed or implied, with respect to the information provided here.

## File system utilities

`mssparkutils.fs` provides utilities for working with various file systems, including Azure Data Lake Storage Gen2 (ADLS Gen2) and Azure Blob Storage. Make sure you configure access to [Azure Data Lake Storage Gen2](#) and [Azure Blob Storage](#) appropriately.

Run the following commands for an overview of the available methods:

Python

```
from notebookutils import mssparkutils
mssparkutils.fs.help()
```

## Output

Console

`mssparkutils.fs` provides utilities for working with various FileSystems.

Below is overview about the available methods:

```
cp(from: String, to: String, recurse: Boolean = false): Boolean -> Copies a file or directory, possibly across FileSystems
mv(from: String, to: String, recurse: Boolean = false): Boolean -> Moves a file or directory, possibly across FileSystems
ls(dir: String): Array -> Lists the contents of a directory
mkdirs(dir: String): Boolean -> Creates the given directory if it does not exist, also creating any necessary parent
directories
put(file: String, contents: String, overwrite: Boolean = false): Boolean -> Writes the given String out to a file, encoded
in UTF-8
head(file: String, maxBytes: int = 1024 * 100): String -> Returns up to the first 'maxBytes' bytes of the given file as a
String encoded in UTF-8
append(file: String, content: String, createFileIfNotExists: Boolean): Boolean -> Append the content to a file
rm(dir: String, recurse: Boolean = false): Boolean -> Removes a file or directory
exists(file: String): Boolean -> Check if a file or directory exists
mount(source: String, mountPoint: String, extraConfigs: Map[String, Any]): Boolean -> Mounts the given remote storage
directory at the given mount point
unmount(mountPoint: String): Boolean -> Deletes a mount point
mounts(): Array[MountPointInfo] -> Show information about what is mounted
getMountPath(mountPoint: String, scope: String = ""): String -> Gets the local path of the mount point
```

Use `mssparkutils.fs.help("methodName")` for more info about a method.

`mssparkutils` works with the file system in the same way as Spark APIs. Take `mssparkutils.fs.mkdirs()` and Fabric Lakehouse usage for example:

Usage	Relative path from HDFS root	Absolute path for ABFS file system
Nondefault lakehouse	Not supported	<code>mssparkutils.fs.mkdirs("abfss://&lt;container_name&gt;@&lt;storage_account_name&gt;.dfs.core.windows.net/&lt;new_dir&gt;"</code>
Default lakehouse	Directory under "Files" or "Tables": <code>mssparkutils.fs.mkdirs("Files/&lt;new_dir&gt;")</code>	<code>mssparkutils.fs.mkdirs("abfss://&lt;container_name&gt;@&lt;storage_account_name&gt;.dfs.core.windows.net/&lt;new_dir&gt;"</code>

## List files

List the content of a directory, use `mssparkutils.fs.ls('Your directory path')`, for example:

Python

```
mssparkutils.fs.ls("Files/tmp") # works with the default lakehouse files using relative path  
mssparkutils.fs.ls("abfss://<container_name>@<storage_account_name>.dfs.core.windows.net/<path>") # based on ABFS file system  
mssparkutils.fs.ls("file:/tmp") # based on local file system of driver node
```

## View file properties

Returns file properties including file name, file path, file size, and whether it's a directory and a file.

Python

```
files = mssparkutils.fs.ls('Your directory path')  
for file in files:  
    print(file.name, file.isDir, file.isFile, file.path, file.size)
```

## Create new directory

Creates the given directory if it doesn't exist and any necessary parent directories.

Python

```
mssparkutils.fs.mkdirs('new directory name')  
mssparkutils.fs.mkdirs("Files/<new_dir>") # works with the default lakehouse files using relative path  
mssparkutils.fs.ls("abfss://<container_name>@<storage_account_name>.dfs.core.windows.net/<new_dir>") # based on ABFS file system  
mssparkutils.fs.ls("file:/<new_dir>") # based on local file system of driver node
```

## Copy file

Copies a file or directory. Supports copy across file systems.

Python

```
mssparkutils.fs.cp('source file or directory', 'destination file or directory', True) # Set the third parameter as True to copy all files and directories recursively
```

## Preview file content

Returns up to the first 'maxBytes' bytes of the given file as a String encoded in UTF-8.

Python

```
mssparkutils.fs.head('file path', maxBytes to read)
```

## Move file

Moves a file or directory. Supports move across file systems.

Python

```
mssparkutils.fs.mv('source file or directory', 'destination directory', True) # Set the last parameter as True to firstly create the parent directory if it does not exist
```

## Write file

Writes the given string out to a file, encoded in UTF-8. Writes the given string out to a file, encoded in UTF-8.

Python

```
mssparkutils.fs.put("file path", "content to write", True) # Set the last parameter as True to overwrite the file if it existed already
```

## Append content to a file

Appends the given string to a file, encoded in UTF-8.

Python

```
mssparkutils.fs.append("file path", "content to append", True) # Set the last parameter as True to create the file if it does not exist
```

## Delete file or directory

Removes a file or directory.

Python

```
mssparkutils.fs.rm('file path', True) # Set the last parameter as True to remove all files and directories recursively
```

## Mount/unmount directory

You can find the detailed usage in [File mount and unmount](#).

## Notebook utilities

Use the MSSparkUtils Notebook Utilities to run a notebook or exit a notebook with a value. Run the following command to get an overview of the available methods:

Python

```
mssparkutils.notebook.help()
```

Output:

Console

```
exit(value: String): void -> This method lets you exit a notebook with a value.  
run(path: String, timeoutSeconds: int, arguments: Map): String -> This method runs a notebook and returns its exit value.
```

## Reference a notebook

Reference a notebook and returns its exit value. You can run nesting function calls in a notebook interactively or in a pipeline. The notebook being referenced runs on the Spark pool of which notebook calls this function.

Python

```
mssparkutils.notebook.run("notebook name", <timeoutSeconds>, <parameterMap>)
```

For example:

Python

```
mssparkutils.notebook.run("Sample1", 90, {"input": 20 })
```

You can open the snapshot link of reference run in the cell output, the snapshot captures the code run results and allows you to easily debug a reference run.

← Snapshot(s): Notebook 2

Save as copy

```

1 mssparkutils.notebook.run("Notebook 2", 90, {"input":50})
2
[3] ✓ 11 sec - Command executed in 10 sec 568 ms by on 9:43:21 PM, 5/05/23
> Log
... View notebook run: Notebook 2
'Hello, Notebook 2 executed successfully'
+ Code + Markdown

```

Details

Snapshot ID	
Livy ID	
Job end time	5/5/23 9:43:19 PM
Duration	9 sec
Submitter	
Default lakehouse	

### ! Note

Currently Fabric notebook only supports referencing notebooks within a workspace.

## Exit a notebook

Exits a notebook with a value. You can run nesting function calls in a notebook interactively or in a pipeline.

- When you call an `exit()` function from a notebook interactively, Fabric notebook throws an exception, skip running subsequence cells, and keep the Spark session alive.
- When you orchestrate a notebook in pipeline that calls an `exit()` function, the Notebook activity will return with an exit value, complete the pipeline run and stop the Spark session.
- When you call an `exit()` function in a notebook that is being referenced, Fabric Spark will stop the further execution of the referenced notebook, and continue to run next cells in the main notebook that calls the `run()` function. For example: Notebook1 has three cells and calls an `exit()` function in the second cell. Notebook2 has five cells and calls `run(notebook1)` in the third cell. When you run Notebook2, Notebook1 stops at the second cell when hitting the `exit()` function. Notebook2 continues to run its fourth cell and fifth cell.

Python

```
mssparkutils.notebook.exit("value string")
```

For example:

Sample1 notebook with following two cells:

- Cell 1 defines an `input` parameter with default value set to 10.
- Cell 2 exits the notebook with `input` as exit value.

```
1   input = "10"
[ ] Press shift + enter to run
```

```
1   mssparkutils.notebook.exit("Notebook executed successfully with exit value"+ str(input))
[ ] Press shift + enter to run
```

+ Code + Markdown 

You can run the **Sample1** in another notebook with default values:

Python

```
exitVal = mssparkutils.notebook.run("Sample1")
print (exitVal)
```

Output:

Console

```
Notebook executed successfully with exit value 10
```

You can run the **Sample1** in another notebook and set the **input** value as 20:

Python

```
exitVal = mssparkutils.notebook.run("Sample1", 90, {"input": 20 })
print (exitVal)
```

Output:

Console

```
Notebook executed successfully with exit value 20
```

## Session management

### Stop an interactive session

Instead of manually selecting the stop button, sometimes it's more convenient to stop an interactive session by calling an API in the code. For such cases, we provide an API `mssparkutils.session.stop()` to support stopping the interactive session via code, it's available for Scala and Python.

Python

```
mssparkutils.session.stop()
```

`mssparkutils.session.stop()` API stops the current interactive session asynchronously in the background, it stops the Spark session and release resources occupied by the session so they're available to other sessions in the same pool.

#### Note

We don't recommend calling language built-in APIs like `sys.exit` in Scala or `sys.exit()` in Python in your code, because such APIs just kill the interpreter process, leaving the Spark session alive and the resources not released.

## Credentials utilities

You can use the MSSparkUtils Credentials Utilities to get the access tokens and manage secrets in Azure Key Vault.

Run the following command to get an overview of the available methods:

Python

```
mssparkutils.credentials.help()
```

Output:

Console

```
getToken(audience, name): returns AAD token for a given audience, name (optional)
getSecret(akvName, secret): returns AKV secret for a given akvName, secret key
```

## Get token

Returns Azure AD token for a given audience, name (optional). The list below shows currently available audience keys:

- Storage Audience Resource: "storage"
- Power BI Resource: "pbi"
- Azure Key Vault Resource: "keyvault"
- Synapse RTA KQL DB Resource: "kusto"

Run the following command to get the token:

Python

```
mssparkutils.credentials.getToken('audience Key')
```

## Get secret using user credentials

Returns Azure Key Vault secret for a given Azure Key Vault name, secret name, and linked service name using user credentials.

Python

```
mssparkutils.credentials.getSecret('azure key vault name','secret name')
```

## File mount and unmount

The Microsoft Fabric support mount scenarios in the Microsoft Spark Utilities package. You can use *mount*, *unmount*, *getMountPath()* and *mounts()* APIs to attach remote storage (Azure Data Lake Storage Gen2) to all working nodes (driver node and worker nodes). After the storage mount point is in place, use the local file API to access data as if it's stored in the local file system.

### How to mount an ADLS Gen2 account

This section illustrates how to mount Azure Data Lake Storage Gen2 step by step as an example. Mounting Blob Storage works similarly.

The example assumes that you have one Data Lake Storage Gen2 account named *storegen2*. The account has one container named *mycontainer* that you want to mount to */test* into your notebook spark session.

To mount the container called *mycontainer*, *mssparkutils* first needs to check whether you have the permission to access the container. Currently, Microsoft Fabric supports two authentication methods for the trigger mount operation: *accountKey* and *sastoken*.

## Mount via shared access signature token or account key

*Mssparkutils* supports explicitly passing an account key or [Shared access signature \(SAS\)](#) token as a parameter to mount the target.

For security reasons, we recommend that you store account keys or SAS tokens in Azure Key Vault (as the following example screenshot shows). You can then retrieve them by using the *mssparkutils.credentials.getSecret* API. For the usage of Azure Key Vault, refer to [About Azure Key Vault managed storage account keys](#).

Here's the sample code of using *accountKey* method:

Python

```
from notebookutils import mssparkutils
# get access token for keyvault resource
# you can also use full audience here like https://vault.azure.net
accountKey = mssparkutils.credentials.getSecret("<vaultURI>", "<secretName>")
mssparkutils.fs.mount(
    "abfss://mycontainer@<accountname>.dfs.core.windows.net",
    "/test",
    {"accountKey":accountKey}
)
```

For *sastoken*, reference the following sample code:

Python

```
from notebookutils import mssparkutils
# get access token for keyvault resource
```

```
# you can also use full audience here like https://vault.azure.net
sasToken = mssparkutils.credentials.getSecret("<vaultURI>", "<secretName>")
mssparkutils.fs.mount(
    "abfss://mycontainer@<accountname>.dfs.core.windows.net",
    "/test",
    {"sasToken":sasToken}
)
```

### ① Note

For security reasons, it's not recommended to store credentials in code. To further protect your credentials, we will redact your secret in notebook output, for more details please check [Secret redaction](#).

## How to mount a lakehouse

Here's the sample code of mounting a lakehouse to `/test`.

Python

```
from notebookutils import mssparkutils
mssparkutils.fs.mount(
    "abfss://<workspace_id>@msit-onelake.dfs.fabric.microsoft.com/<lakehouse_id>",
    "/test"
)
```

## Access files under the mount point by using the `mssparkutils fs` API

The main purpose of the mount operation is to let customers access the data stored in a remote storage account by using a local file system API. You can also access the data by using the `mssparkutils fs` API with a mounted path as a parameter. The path format used here's a little different.

Assume that you mounted the Data Lake Storage Gen2 container `mycontainer` to `/test` by using the mount API. When you access the data by using a local file system API, the path format is like this:

Python

```
/synfs/notebook/{sessionId}/test/{filename}
```

When you want to access the data by using the `mssparkutils fs` API, we recommend using a `getMountPath()` to get the accurate path:

Python

```
path = mssparkutils.fs.getMountPath("/test")
```

- List directories:

Python

```
mssparkutils.fs.ls(f"file://{mssparkutils.fs.getMountPath('/test')}")
```

- Read file content:

Python

```
mssparkutils.fs.head(f"file://{mssparkutils.fs.getMountPath('/test')}/myFile.txt")
```

- Create a directory:

Python

```
mssparkutils.fs.mkdirs(f"file://{mssparkutils.fs.getMountPath('/test')}/newdir")
```

## Access files under the mount point via local path

You can easily read and write the files in mount point using standard file system way, use Python as an example:

```
Python

#File read
with open(mssparkutils.fs.getMountPath('/test2') + "/myFile.txt", "r") as f:
    print(f.read())
#File write
with open(mssparkutils.fs.getMountPath('/test2') + "/myFile.txt", "w") as f:
    print(f.write("dummy data"))
```

## How to check existing mount points

You can use `mssparkutils.fs.mounts()` API to check all existing mount point info:

```
Python

mssparkutils.fs.mounts()
```

## How to unmount the mount point

Use the following code to unmount your mount point (`/test` in this example):

```
Python

mssparkutils.fs.unmount("/test")
```

## Known limitations

- The current mount is a job level configuration, we recommend to use `mounts` API to check if mount point exists or not available.
- The unmount mechanism isn't automatic. When the application run finishes, to unmount the mount point to release the disk space, you need to explicitly call an unmount API in your code. Otherwise, the mount point will still exist in the node after the application run finishes.
- Mounting an ADLS Gen1 storage account isn't supported.

## Next steps

- [Library management](#)

# Manage Apache Spark libraries in Microsoft Fabric

Article • 05/23/2023

**Libraries** provide reusable code that Apache Spark developers may want to include in their Spark application.

Each workspace comes with a pre-installed set of libraries available in the Spark runtime and available to be used immediately in the notebook or Spark job definition. We refer to these as built-in libraries.

## ⓘ Important

Microsoft Fabric is currently in PREVIEW. This information relates to a prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, expressed or implied, with respect to the information provided here.

Based on the user scenarios and specific needs, you can include other libraries. There are two types of libraries you may want to include:

- **Feed library:** Feed libraries are the ones that come from public sources or repositories. You can install Python feed libraries from PyPI and Conda by specifying the source in the Library Management portals. You can also use a Conda environment specification *.yml* file to install libraries.
- **Custom library:** Custom libraries are the code built by you or your organization. *.whl*, *.jar* and *.tar.gz* can be managed through Library Management portals. Note that *.tar.gz* is only supported for R language, please use *.whl* for Python custom libraries.

## Summary of library management and best practices

You can manage all the previously mentioned types of libraries via two different entry points: library management in workspace settings and in-line installation.

1. **Workspace library management:** Workspace library settings define the working environment for the entire Workspace. The libraries installed on a Workspace level are available for all Notebooks and Spark job definitions under that workspace.

Update the workspace libraries when you want to set up the shared environment for all items in a workspace.

**ⓘ Important**

Workspace library management is restricted to workspace admin only. Workspace member, contributor and viewer can view the libraries installed by the administrator.

2. **In-line installation:** With in-line installation, you can install libraries for your Notebook session without affecting the global environment. This is a convenient option when you want a temporary and fast solution. For instance, you might want to try out a local package or use some additional packages for a single session. Currently, Python packages and R packages can be managed in-line.

**ⓘ Important**

In-line installation is session-specific and does not persist across sessions.

The Python interpreter will be restarted to apply the changes of library, any variables defined before running the command cell will be lost. Therefore, we strongly recommend you to put all the commands for adding, deleting, or updating Python packages at the beginning of your Notebook.

Summarizing all library management behaviors currently available in Microsoft Fabric:

Library name	Workspace update	In-line installation
Python Feed (PyPI & Conda)	Supported	Supported
Python Custom (.whl)	Supported	Supported
R Feed (CRAN)	Not Supported	Supported
R custom (.tar.gz)	Supported	Supported
Jar	Supported	Not Supported

**ⓘ Important**

We currently have limitations of *.jar* library.

- If you upload a *.jar* file with different version of built-in library, it will not be effective on the Starter pool, since all *.jar* files are pre-imported in the Starter pool. Only the new *.jar* will be effective on the Starter pools. The custom spark pools has no such constraints, the custom *.jar* files uploaded with different version will override the built-in ones, and the new ones are also effective.
- *%% configure* magic commands are not fully supported on Fabric at this moment. Please don't use it to bring *.jar* file to your Notebook session.

## Library Management in Workspace setting

Under the **Workspace settings**, you find the Workspace level library management portal: **Workspace setting > Data engineering > Library management**.

### Manage feed library in Workspace setting

In this section, we explain how to manage feed libraries from PyPI or Conda using the Workspace library management portal.

- **View and search feed library:** You can see the installed libraries and their name, version, and dependencies on the **library management portal**. You can also use the filter box on the upper right corner to find an installed library quickly.
- **Add new feed library:** The default source for installing Python feed libraries is PyPI. You can also select "Conda" from the drop-down button next to the add button. To add a new library, click on the + button and enter the library name and version in the new row.

Alternatively, you can upload a *.yml* file to install multiple feed libraries at once.

- **Remove existing feed library:** To remove a library, click on the Trash button on its row.
- **Update the version of existing feed library:** To change the version of a library, select a different one from the drop-down box on its row.
- **Review and apply changes:** You can review your changes in the "Pending changes" panel. You can remove a change by clicking on the X button, or discard all changes by clicking on the **Discard** button at the bottom of the page. When you are satisfied with your changes, click on **Apply** to make these changes effective.

# Manage custom libraries in Workspace setting

In this section, we explain how to manage your custom packages, such as *.jar*, using the Workspace library management portal.

- **Upload new custom library:** You can upload your custom codes as packages to the Microsoft Fabric runtime through the portal. The library management module will help you resolve potential conflicts and download dependencies in your custom libraries.

To upload a package, click on the **Upload** button under the **Custom libraries** panel and select a local directory.

- **Remove existing custom library:** You can remove a custom library from the Spark runtime by clicking on the trash button under the **Custom libraries** panel.
- **Review and apply changes:** As with feed libraries, you can review your changes in the **Pending changes** panel and apply them to your Microsoft Fabric Spark environment of the Workspace.

## ⓘ Note

For *.whl* packages, the library installation process will download the dependencies from public sources automatically. However, this feature is not available for *.tar.gz* packages. You need to upload the dependent packages of the main *.tar.gz* package manually if there are any.

## Cancel update

The library update process may take some time to complete. You have the option to cancel the process and continue editing while it is updating. The **Cancel** button will appear during the process.

## Troubleshooting

If the library update process fails, you will receive a notification. You can click on the **View log** button to see the log details and troubleshoot the problem. If you encounter a system error, you can copy the root activity ID and report it to the support team.

## In-line installation

If you want to use some additional packages for a quick test in an interactive Notebook run, in-line installation is the most convenient option.

### ⓘ Important

`%pip` is recommended instead of `!pip`. `!pip` is a IPython built-in shell command which has following limitations:

- `!pip` will only install package on driver node without executor nodes.
- Packages that install through `!pip` will not effect when conflicts with built-in packages or when it's already imported in Notebook.

However, `%pip` will handle all above mentioned scenarios. Libraries installed through `%pip` will be available on both driver & executor nodes and will be still effective even it's already imported.

### ⓘ Tip

- The `%conda install` command usually takes longer than the `%pip install` command to install new Python libraries, because it checks the full dependencies and resolves conflicts. You may want to use `%conda install` for more reliability and stability. You can use `%pip install` if you are sure that the library you want to install does not conflict with the pre-installed libraries in the runtime environment.
- All available Python in-line commands and its clarifications can be found:  
[%pip commands](#) and [%conda commands](#)

## Manage Python feed libraries through in-line installation

In this example, we show you how to use in-line commands to manage libraries. Suppose you want to use `altair`, a powerful visualization library for Python, for a one-time data exploration. And suppose the library is not installed on Workspace. In the following example, we use conda commands to illustrate the steps.

You can use in-line commands to enable `altair` on your Notebook session without affecting other sessions of the Notebook or other items.

1. Run the following commands in a Notebook code cell to install the `altair` library and `vega_datasets`, which contains dataset you can use to visualize:

Python

```
%conda install altair          # install latest version through conda command  
%conda install vega_datasets    # install latest version through conda command
```

The log in the cell output indicates the result of installation.

2. Import the package and dataset by running the following codes in another Notebook cell:

Python

```
import altair as alt  
from vega_datasets import data
```

3. Now you can play around with the session-scoped *altair* library:

Python

```
# load a simple dataset as a pandas DataFrame  
cars = data.cars()  
alt.Chart(cars).mark_point().encode(  
    x='Horsepower',  
    y='Miles_per_Gallon',  
    color='Origin',  
).interactive()
```

## Manage Python custom libraries through in-line installation

You can upload your Python custom libraries to the **File** folder of the lakehouse attached to your notebook. Go to your lakehouse, click on the ... icon on the **File** folder, and upload the custom library.

After uploading, you can use the following command to install the custom library to your Notebook session:

Python

```
# install the .whl through pip command  
%pip install /lakehouse/default/Files/wheel_file_name.whl
```

# Manage R feed libraries through in-line installation

Microsoft Fabric supports `install.packages()`, `remove.packages()` and `devtools::` commands to manage R libraries.

## Tip

All available R in-line commands and its clarifications can be found:

[install.packages command ↗](#), [remove.package command ↗](#) and [devtools commands ↗](#).

Follow this example to walk through the steps of installing an R feed library:

1. Switch working language to “SparkR(R)” in Notebook ribbon.
2. Run the following command in a Notebook cell to install `caesar` library:

Python

```
install.packages("caesar")
```

3. Now you can play around with the session-scoped `caesar` library with Spark job

Python

```
library(SparkR)
sparkR.session()

hello <- function(x) {
  library(caesar)
  caesar(x)
}
spark.lapply(c("hello world", "good morning", "good evening"), hello)
```

## Next steps

- [Apache Spark workspace administration settings](#)

# Capacity administration settings for Data Engineering and Data Science

Article • 05/23/2023

Applies to: Data Engineering and Data Science in Microsoft Fabric

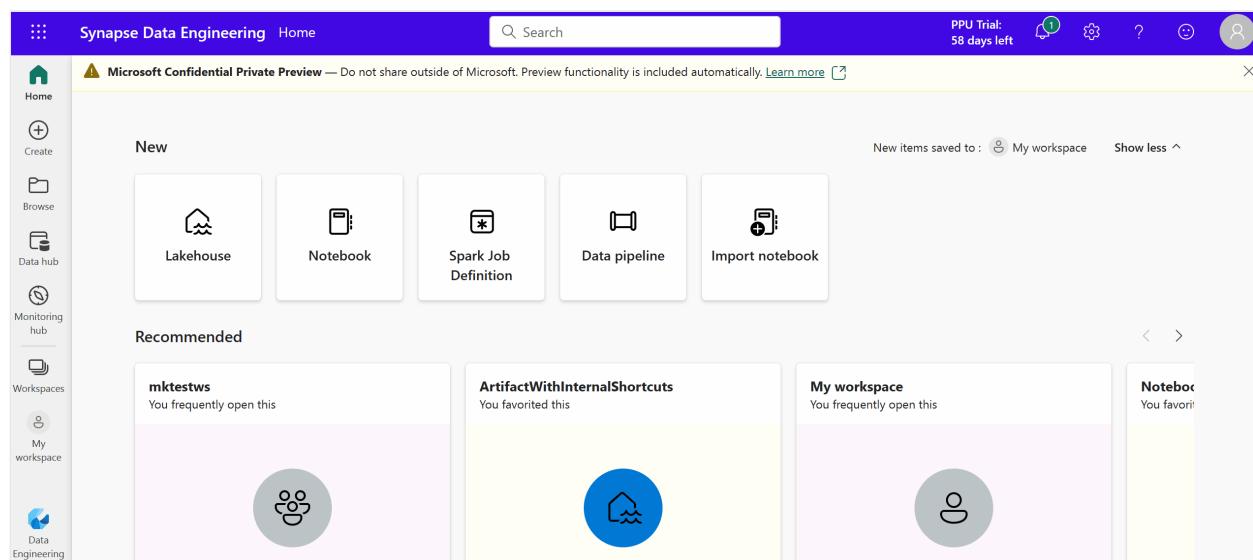
Admins purchase Microsoft Fabric capacities based on the compute and scale requirements of their enterprise's analytics needs. Admins are responsible to manage the capacity and governance. They must govern and manage the compute properties for data engineering and science analytics applications.

## Important

Microsoft Fabric is currently in PREVIEW. This information relates to a prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, expressed or implied, with respect to the information provided here.

Microsoft Fabric capacity admins can now manage and govern their Data Engineering and Data Science settings from the admin settings portal. Admins can configure Spark environment for their users by enabling workspace level compute, choose a default runtime, and also create or manage spark properties for their capacities.

From the Admin portal, navigate to the **Data Engineering/Science Settings** section and select a specific capacity as shown in the following animation:



## Next steps

- Get Started with Data Engineering/Science Admin Settings for your Fabric Capacity

# Configure and manage data engineering and data science settings for Fabric capacities

Article • 05/23/2023

Applies to:  Data Engineering and Data Science in Microsoft Fabric

When you create Microsoft Fabric from the Azure portal, it is automatically added to the Fabric tenant that's associated with the subscription used to create the capacity. With the simplified setup in Microsoft Fabric, there's no need to link the capacity to the Fabric tenant. Because the newly created capacity will be listed in the admin settings pane. This configuration provides a faster experience for admins to start setting up the capacity for their enterprise analytics teams.

## Important

Microsoft Fabric is currently in PREVIEW. This information relates to a prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, expressed or implied, with respect to the information provided here.

To make changes to the Data Engineering/Science settings in a capacity, you must have admin role for that capacity. To learn more about the roles that you can assign to users in a capacity, see [Roles in capacities](#).

Use the following steps to manage the Data Engineering/Science settings for Microsoft Fabric capacity:

1. Select the **Settings** option to open the setting pane for your Fabric account. Select [Admin portal](#) under Governance and insights section

The screenshot shows the Microsoft Fabric Admin portal. In the top navigation bar, there are icons for Microsoft, Microsoft Fabric, a search bar, and settings. Below the header, the 'Admin portal' is selected in the sidebar. The main content area is titled 'Fabric capacity > enterprisefabriccapacity'. It includes a 'Fabric capacity' section with a description and a 'Learn more' link. Below this are several expandable sections: 'Capacity usage report', 'Notifications', 'Contributor permissions' (disabled for the entire organization), 'Power BI workloads', 'Data Engineering/Science Settings' (with a 'Open Spark compute' button), and 'Workspaces assigned to this capacity'. On the right side, a 'Settings' menu is open, showing options like Preferences, General, Notifications, Item settings, Developer settings, Resources and extensions, Manage personal storage, Power BI settings, Manage connections and gateways, Manage embed codes, Azure Analysis Services migrations, and Governance and insights. The 'Admin portal' link under Governance and insights is highlighted with a blue box.

2. Choose the **Capacity settings** option to expand the menu and select **Fabric capacity** tab. Here you should see the capacities that you have created in your tenant. Choose the capacity that you want to configure.

The screenshot shows the 'Capacity settings' page with the 'Fabric capacity' tab selected. The left sidebar has 'Capacity settings' and 'Refresh summary'. The main area shows 'PREMIUM CAPACITIES' with a table. The table columns are: CAPACITY NAME, CAPACITY ADMINS, ACTIONS, CAPACITY SKU, CAPACITY UNITS, REGION, and STATUS. One row is visible: 'enterprisefabric...', 'Dakota Sanchez', with a magnifying glass icon in the 'Actions' column, 'F16', '16', 'West Central US', and 'Active'. At the bottom of the table is a link 'Set up a new capacity in Azure'.

3. You are navigated to the capacities detail pane, where you can view the usage and other admin controls for your capacity. Navigate to the **Data Engineering/Science Settings** section and select **Open Spark Compute**. Configure the following parameters:

- **Customized workspace pools:** You can restrict or democratize compute customization to workspace admins by enabling or disabling this option. Enabling this option allows workspace admins to create, update, or delete workspace level custom spark pools. Additionally, it allows you to resize them based on the compute requirements within the maximum cores limit of a capacity.
- **Runtime version:** As a capacity admin, you can select a default runtime version for the entire capacity. All the new workspaces created in that capacity inherit the selected runtime version. Workspace admins can override the default runtime version inherited and choose a different runtime version based on their workspace level requirements.

- **Spark properties:** Capacity admins can configure spark properties and their values, which are inherited to all the workspaces in the capacity. Like the spark runtime version, workspace admins can override these properties for their individual workspaces.

## Spark compute

Customized workspace pools

Permit workspace admins to size their custom Spark pools based on workspace compute requirements. [Learn more](#)

On

> Pool lists

Runtime

Runtime version

Runtime family defines which version of Spark your Spark pool will use. [Learn more](#)

1.1 (Spark 3.3, Delta 2.2)

Configuration

Spark properties

Spark properties allow you to define many Spark runtime properties. [Learn more](#)

Property lists

+ Add     Delete

<input type="checkbox"/>	Property	Value	
<input type="checkbox"/>	Text	Text	
<input type="checkbox"/>	Text	Text	
<input type="checkbox"/>	Text	Text	

**Apply**  **Discard**

4. After configuring, select **Apply**

## Next steps

- [Get Started with Data Engineering/Science Admin Settings for your Fabric Workspace](#)
- [Learn about the Spark Compute for Fabric Data Engineering/Science experiences](#)

# Spark workspace administration settings in Microsoft Fabric

Article • 05/23/2023

Applies to: Data Engineering and Data Science in Microsoft Fabric

When you create a workspace in Microsoft Fabric, a [Starter Pool](#) that is associated with that workspace is automatically created. With the simplified setup in Microsoft Fabric, there's no need to choose the node or machine sizes, as this is handled for you behind the scenes. This configuration provides a faster (5-10 seconds) spark session start experience for users to get started and run your Spark jobs in many common scenarios without having to worry about setting up the compute. For advanced scenarios with specific compute requirements, users can create a custom spark pool and size the nodes based on their performance needs.

## Important

Microsoft Fabric is currently in PREVIEW. This information relates to a prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, expressed or implied, with respect to the information provided here.

To make changes to the Spark settings in a workspace, you should have the admin role for that workspace. To learn more, see [Roles in workspaces](#).

To manage the Spark settings for the pool associated with your workspace:

1. Go to the **Workspace settings** in your workspace and choose the **Data Engineering/Science** option to expand the menu:

The screenshot shows the Microsoft Fabric workspace settings interface. On the left, there's a list of resources: HollyTest (Lakehouse, Owner Remy Morris), Notebook 1, Notebook 2, Notebook 3, and SampleLakeHouse (Lakehouse, Owner Remy Morris). On the right, the 'Workspace settings' sidebar is open, showing options like About, Premium, Azure connections, System Storage, Other, Power BI, Extension API playground, and Data Engineering. The 'Data Engineering' section is currently selected, indicated by a dropdown arrow.

2. You see the **Spark Compute** option in your left-hand menu:
3. Configure the four setting options you can change on this page: **Default pool for workspace**, **Runtime version**, **Automatically track machine learning experiments and models** and **Spark properties**.

 **Note**

If you change the default pool to a custom spark pool you may see longer session start (~3 minutes) in this case.

## Default pool for workspace

There are two options:

- **Starter Pool:** Prehydrated live clusters automatically created for your faster experience. These are medium size. Currently, a starter pool with 10 nodes is provided for evaluation purposes.
- **Custom Spark Pool:** You can size the nodes, autoscale, and dynamically allocate executors based on your spark job requirements. To create a custom spark pool, the capacity admin should enable the **Customized workspace pools** option in the **Spark Compute** section of **Capacity Admin** settings. To learn more, see [Spark Compute Settings for Fabric Capacities](#).

Admins can create custom spark pools based on their compute requirements by selecting the **New Pool** option.



## Create pool

Spark pool name \*

Node family

Node size

Small

Medium

Large

X-Large

XX-Large

Enable allocate

1



9

down based on the amount of activity.

Microsoft Fabric spark supports single node clusters, which allows users to select a minimum node configuration of 1 and a maximum of 2. Thereby offering high-availability and better job reliability for smaller compute requirements. You can also enable or disable autoscaling option for your custom spark pools. When enabled with autoscale, the pool would acquire new nodes within the max node limit specified by the user and retire them after the job execution for better performance.

You can also select the option to dynamically allocate executors to pool automatically optimal number of executors within the max bound specified based on the data volume for better performance.

## Edit pool

Spark pool name \*

Node family

Node size

Autoscale

If enabled, your Apache Spark pool will automatically scale up and down based on the amount of activity.

 Enable autoscale

Dynamically allocate executors

 Enable allocate

Learn more about [Spark Compute for Fabric](#).

## Runtime version

You may choose which version of Spark you'd like to use for the workspace. Currently, Spark 3.2 version is available.

— Runtime

### Runtime Version

Runtime family defines which version of Spark your Spark pool will use.

[Learn more about Runtime Version](#)



## Autologging for Machine Learning models and experiments

Admins can now enable autologging for their machine learning models and experiments. This option will automatically capture the values of input parameters, output metrics, and output items of a machine learning model as it is being trained.

[Learn more about autologging ↗](#)

## Spark properties

Apache Spark has many settings you can provide to optimize the experience for your scenarios. You may set those properties through the UI by selecting the **Add** option. Select an item from the dropdown menu, and enter the value.

**Spark properties**

Spark properties allows you to define many Spark runtime properties. [Learn more](#)

+ Add     Delete

<input type="checkbox"/>	Property	Value	
<input type="checkbox"/>	Property name	No default value	

You can delete items by selecting the item(s) and then select the **Delete** button. Or select the delete icon next to each item you wish you to delete.

**Spark properties**

Spark properties allows you to define many Spark runtime properties. [Learn more](#)

+ Add     Delete

<input checked="" type="checkbox"/>	Property	Value	
<input checked="" type="checkbox"/>	spark.blacklist.enabled	True	

## Next steps

- Learn more from the Apache Spark [public documentation ↗](#).

# Apache Spark workspace administration settings FAQ

FAQ

This article lists answers to frequently asked questions about Apache Spark workspace administration settings.

## Important

Microsoft Fabric is currently in PREVIEW. This information relates to a prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, expressed or implied, with respect to the information provided here.

## How do I use the RBAC roles to configure my Spark workspace settings?

Use the **Manage Access** menu to add **Admin** permissions for specific users, distribution groups, or security groups. You can also use this menu to make changes to the workspace and to grant access to add, modify, or delete the Spark workspace settings.

## Are the changes made to the Spark properties at the workspace level apply to the active notebook sessions or scheduled Spark jobs?

When you make a configuration change at the workspace level, it's not applied to active Spark sessions. This includes batch or notebook based sessions. You must start a new notebook or a batch session after saving the new configuration settings for the settings to take effect.

# **Can I configure the node family, Spark runtime, and Spark properties at a capacity level?**

Yes, you can change the runtime, or manage the spark properties using the Data Engineering/Science settings as part of the capacity admin settings page. You need have the capacity admin access to view and change these capacity settings.

# **Can I choose different node families for different notebooks and Spark job definitions in my workspace?**

Currently, you can only select Memory Optimized based node family for the entire workspace.

# **What versions of Spark are supported?**

Currently, Spark version 3.3 is the only supported version. Additional versions will be available in the upcoming release.

# **Can I configure these settings at a notebook level?**

Currently, the Spark administration settings are only available at the workspace and capacity level

# **Can I configure the minimum and maximum number of nodes for the selected node family?**

Currently, node limit configuration isn't available. This capability will be enabled in future releases.

# **Can I enable Autoscaling for the Spark Pools in a memory optimized or hardware accelerated GPU based node family?**

Autoscaling isn't currently available. This capability will be enabled in future releases.

# **Is Intelligent Caching for the Spark Pools supported or enabled by default for a workspace?**

Intelligent Caching is enabled by default for the Spark pools for all workspaces.

# Microsoft Fabric Apache Spark monitoring overview

Article • 05/23/2023

Microsoft Fabric Spark monitoring is designed to offer a web-UI based experience with built-in rich capabilities for monitoring the progress and status of Spark applications in progress, browsing past Spark activities, analyzing and optimizing performance, and facilitating troubleshooting of failures. Multiple entry points are available for browsing, monitoring, and viewing Spark application details.

## ⓘ Important

Microsoft Fabric is currently in PREVIEW. This information relates to a prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, expressed or implied, with respect to the information provided here.

## Monitoring hub

The Monitoring Hub serves as a centralized portal for browsing Spark activities across items. At a glance, you can view in-progress Spark applications triggered from Notebooks, Spark Job Definitions, and Pipelines. You can also search and filter Spark applications based on different criteria and drill down to view more Spark execution details of a Spark application.

## Item recent runs

When working on specific items, the item Recent Runs feature allows you to browse the item's current and recent activities and gain insights on the submitter, status, duration, and other information for activities submitted by you or others.

## Notebook contextual monitoring

Notebook Contextual Monitoring offers the capability of authoring, monitoring, and debugging Spark jobs within a single place. You can monitor Spark job progress, view Spark execution tasks and executors, and access Spark logs within a Notebook at the Notebook cell level. The Spark advisor is also built into Notebook to offer real-time advice on code and cell Spark execution and perform error analysis.

# Spark job definition inline monitoring

The Spark job definition Inline Monitoring feature allows you to view Spark job definition submission and run status in real-time, as well as view the Spark job definition's past runs and configurations. You can navigate to the Spark application detail page to view more details.

# Pipeline Spark activity inline monitoring

For Pipeline Spark Activity Inline Monitoring, deep links have been built into the Notebook and Spark job definition activities within the Pipeline. You can view Spark application execution details, the respective Notebook and Spark job definition snapshot, and access Spark logs for troubleshooting. If the Spark activities fail, the inline error message is also available within Pipeline Spark activities.

## Next steps

- [Workspace item recent runs](#)
- [Notebook contextual monitoring and debugging](#)
- [Run an Apache Spark job definition](#)
- [Apache Spark application detail monitoring](#)

# Apache Spark advisor for real-time advice on notebooks

Article • 05/23/2023

## ⓘ Important

Microsoft Fabric is currently in PREVIEW. This information relates to a prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, expressed or implied, with respect to the information provided here.

The Apache Spark advisor analyzes commands and code run by Apache Spark and displays real-time advice for Notebook runs. The Apache Spark advisor has built-in patterns to help users avoid common mistakes. It offers recommendations for code optimization, performs error analysis, and locates the root cause of failures.

## Built-in advice

The Spark advisor, a tool integrated with Impulse, provides built-in patterns for detecting and resolving issues in Apache Spark applications. This article explains some of the patterns included in the tool.

You can open the **Recent runs** pane based on the type of advice you need.

## May return inconsistent results when using 'randomSplit'

Inconsistent or inaccurate results may be returned when working with the *randomSplit* method. Use Apache Spark (RDD) caching before using the randomSplit() method.

Method randomSplit() is equivalent to performing sample() on your data frame multiple times. Where each sample refetches, partitions, and sorts your data frame within partitions. The data distribution across partitions and sorting order is important for both randomSplit() and sample(). If either changes upon data refetch, there may be duplicates or missing values across splits. And the same sample using the same seed may produce different results.

These inconsistencies may not happen on every run, but to eliminate them completely, cache your data frame, repartition on a column(s), or apply aggregate functions such as *groupBy*.

## Table/view name is already in use

A view already exists with the same name as the created table, or a table already exists with the same name as the created view. When this name is used in queries or applications, only the view will be returned no matter which one created first. To avoid conflicts, rename either the table or the view.

## Unable to recognize a hint

Scala

```
spark.sql("SELECT /*+ unknownHint */ * FROM t1")
```

## Unable to find a specified relation name(s)

Unable to find the relation(s) specified in the hint. Verify that the relation(s) are spelled correctly and accessible within the scope of the hint.

Scala

```
spark.sql("SELECT /*+ BROADCAST(unknownTable) */ * FROM t1 INNER JOIN t2 ON  
t1.str = t2.str")
```

## A hint in the query prevents another hint from being applied

The selected query contains a hint that prevents another hint from being applied.

Scala

```
spark.sql("SELECT /*+ BROADCAST(t1), MERGE(t1, t2) */ * FROM t1 INNER JOIN  
t2 ON t1.str = t2.str")
```

## Enable 'spark.advise.divisionExprConvertRule.enable' to reduce rounding error propagation

This query contains the expression with Double type. We recommend that you enable the configuration 'spark.advise.divisionExprConvertRule.enable', which can help reduce the division expressions and to reduce the rounding error propagation.

## Console

```
"t.a/t.b/t.c" convert into "t.a/(t.b * t.c)"
```

## Enable 'spark.advise.nonEqJoinConvertRule.enable' to improve query performance

This query contains time consuming join due to "Or" condition within query. We recommend that you enable the configuration

'spark.advise.nonEqJoinConvertRule.enable', which can help to convert the join triggered by "Or" condition to SMJ or BHJ to accelerate this query.

## User experience

The Apache Spark advisor displays the advice, including info, warnings, and errors, at Notebook cell output in real-time.

- Info

A screenshot of a Jupyter Notebook cell. The cell contains the following Scala code:

```
1 val rdd = sc.parallelize(1 to 1000000)
2 val rdd2 = rdd.repartition(64)
3 val Array(train, test) = rdd2.randomSplit(Array(70, 30), 1)
4 train.takeOrdered(10)
```

Below the code, a message indicates the command was executed in 5 seconds:

✓ 7 sec - Command executed in 5 sec 367 ms by Dakota Sanchez on 2:30:22 PM, 2/28/23

The cell also shows a 'Diagnostics' section with a warning message:

> ⚠️ May return inconsistent results when using 'randomSplit'

... (truncated)

The code output shows the creation of RDDs and their partitions:

```
rdd: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[44] at parallelize at <console>:31
rdd2: org.apache.spark.rdd.RDD[Int] = MapPartitionsRDD[48] at repartition at <console>:31
train: org.apache.spark.rdd.RDD[Int] = MapPartitionsRDD[49] at randomSplit at <console>:32
test: org.apache.spark.rdd.RDD[Int] = MapPartitionsRDD[50] at randomSplit at <console>:32
res0: Array[Int] = Array(1, 3, 4, 5, 6, 7, 8, 10, 11, 13)
```

- Warning

```

1  %%spark
2  import org.apache.spark.SparkContext
3  import java.util.Random
4  def testDataSkew(sc: SparkContext): Unit = {
5    val numMappers = 400
6    val numKVPairs = 100000
7    val valSize = 256
8    val numReducers = 200
9    val biasPct = 0.4
10   val biasCount = numKVPairs * biasPct
11   for (i < 1 to 2) {
12     val query = sc.parallelize(0 until numMappers, numMappers).flatMap { p =>
13       val ranGen = new Random
14       val arr1 = new Array[Int, Array[Byte]](numKVPairs)
15       for (i < 0 until numKVPairs) {
16         val byteArr = new Array[Byte](valSize)
17         ranGen.nextBytes(byteArr)
18         var key = ranGen.nextInt(Int.MaxValue)
19         if(i <= biasCount) {
20           key = 1
21         }
22         arr1(i) = (key, byteArr)
23       }
24     }
25     arr1
26   }.groupByKey(numReducers)
27   println(query.count())
28 }
29 testDataSkew(sc)

```

✓ 3 min 52 sec - 🐾 Dakota Sanchez is running the cell.

Spark (Scala) ▾

✓ Spark jobs (2 of 2 succeeded)

ID	Description	Status	Stages	Tasks	Duration	Rows	Data read	Data written
Job 7	count at <console>:53	✓ Succeeded	2/2	600/600 succeeded	56 sec	80000000	9.83 GB	9.83 GB
Job 8	count at <console>:53	✓ Succeeded	2/2	600/600 succeeded	2 min 38 sec	80000000	9.83 GB	9.83 GB

✓ Diagnostics ▲ 3

- > ⚠ Data skew for job 7
- > ⚠ Data skew for job 8
- > ⚠ Time skew for job 8

```

23865871
23866411
import org.apache.spark.SparkContext
import java.util.Random
testDataSkew: (sc: org.apache.spark.SparkContext)Unit

```

- Error

```

1  # display(pandas.DataFrame) sample:
2  import numpy as np
3  import pandas as pd
4  from pyspark.sql import *
5  d = {'col1': [1, 2, 3, 4, 5, 6, 7, 8], 'col12': [3, 4, 5, 6, 8, 3, 16, 20]}
6  pdf = pd.DataFrame(data=d)
7
8  display(pdf)
9

```

✗ 1 sec - 🐾 Dakota Sanchez is running the cell.

Spark (Scala) ▾

✗ Diagnostics ✗ 1

- > ✗ Spark\_User\_AutoClassification\_pandas\_DataFrame

```

<console>:1: error: illegal start of definition
# display(pandas.DataFrame) sample:
^

```

# Next steps

- Monitor Apache Spark jobs within notebooks
- Monitor Apache Spark job definition
- Monitor Apache Spark application details

# Browse the Apache Spark applications in the Fabric monitoring hub

Article • 05/23/2023

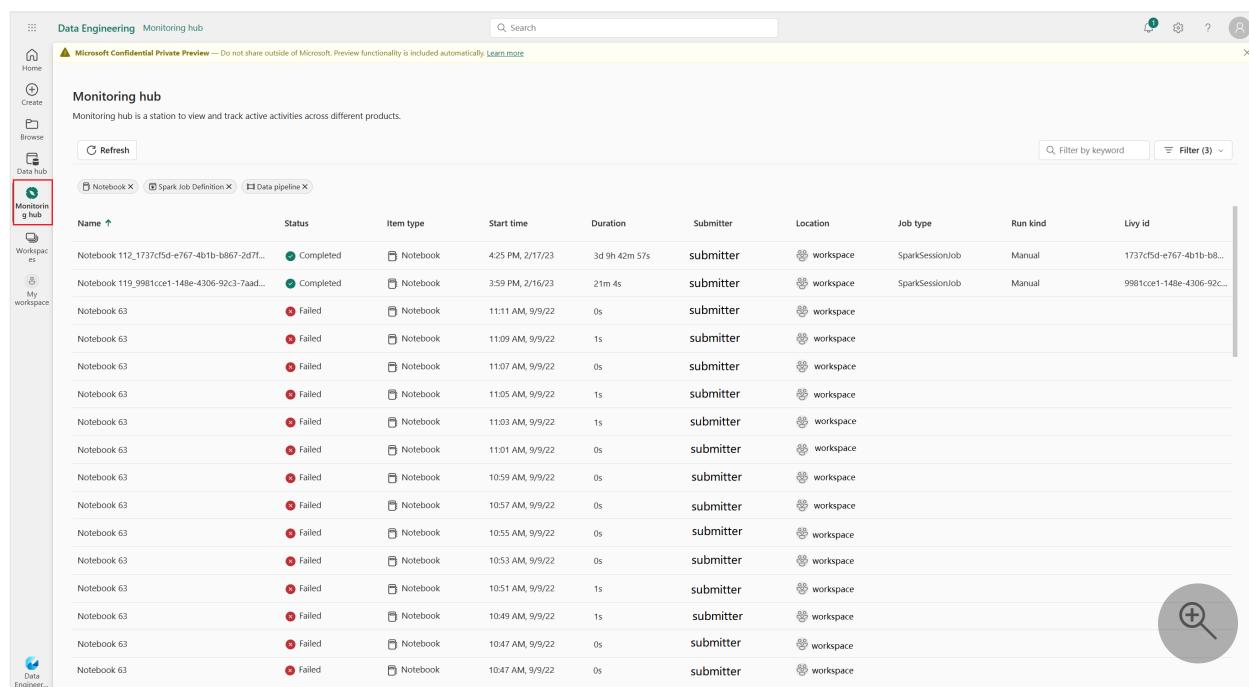
The Monitoring hub serves as a centralized portal for browsing Apache Spark activities across items. When you are in the Data Engineering or Data Science experience, you can view in-progress Apache Spark applications triggered from Notebooks, Apache Spark job definitions, and Pipelines. You can also search and filter Apache Spark applications based on different criteria. Additionally, you can cancel your in-progress Apache Spark applications and drill down to view more execution details of an Apache Spark application.

## ⓘ Important

Microsoft Fabric is currently in PREVIEW. This information relates to a prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, expressed or implied, with respect to the information provided here.

## Access the monitoring hub

You can access the Monitoring hub to view various Apache Spark activities by selecting **Monitoring hub** in the left-side navigation links.



The screenshot shows the Microsoft Fabric Data Engineering interface with the 'Monitoring hub' selected in the left sidebar. The main area displays a table of Apache Spark activities, specifically Notebooks. The columns include Name, Status, Item type, Start time, Duration, Submitter, Location, Job type, Run kind, and Livy id. Most entries show a 'Completed' status, while one is 'Failed'. A search bar and filter options are at the top right. A magnifying glass icon is in the bottom right corner.

Name	Status	Item type	Start time	Duration	Submitter	Location	Job type	Run kind	Livy id
Notebook 112_1737cf5d-e767-4b1b-b867-2d7f...	Completed	Notebook	4:25 PM, 2/17/23	3d 9h 42m 57s	submitter	workspace	SparkSessionJob	Manual	1737cf5d-e767-4b1b-b8...
Notebook 119_9981cce1-148e-430e-92c3-7aad...	Completed	Notebook	3:59 PM, 2/16/23	21m 4s	submitter	workspace	SparkSessionJob	Manual	9981cce1-148e-430e-92c...
Notebook 63	Failed	Notebook	11:11 AM, 9/9/22	0s	submitter	workspace			
Notebook 63	Failed	Notebook	11:09 AM, 9/9/22	1s	submitter	workspace			
Notebook 63	Failed	Notebook	11:07 AM, 9/9/22	0s	submitter	workspace			
Notebook 63	Failed	Notebook	11:05 AM, 9/9/22	1s	submitter	workspace			
Notebook 63	Failed	Notebook	11:03 AM, 9/9/22	1s	submitter	workspace			
Notebook 63	Failed	Notebook	11:01 AM, 9/9/22	0s	submitter	workspace			
Notebook 63	Failed	Notebook	10:59 AM, 9/9/22	0s	submitter	workspace			
Notebook 63	Failed	Notebook	10:57 AM, 9/9/22	0s	submitter	workspace			
Notebook 63	Failed	Notebook	10:55 AM, 9/9/22	0s	submitter	workspace			
Notebook 63	Failed	Notebook	10:53 AM, 9/9/22	0s	submitter	workspace			
Notebook 63	Failed	Notebook	10:51 AM, 9/9/22	1s	submitter	workspace			
Notebook 63	Failed	Notebook	10:49 AM, 9/9/22	1s	submitter	workspace			
Notebook 63	Failed	Notebook	10:47 AM, 9/9/22	0s	submitter	workspace			
Notebook 63	Failed	Notebook	10:47 AM, 9/9/22	0s	submitter	workspace			

# Sort, search and filter Apache Spark applications

For better usability and discoverability, you can sort the Apache Spark applications by selecting different columns in the UI. You can also filter the applications based on different columns and search for specific applications.

## Sort Apache Spark applications

To sort Apache Spark applications, you can select on each column header, such as **Name**, **Status**, **Item Type**, **Start Time**, **Location**, and so on.

Name	Status	Item type	Start time	Duration	Submitter	Location	Job type	Run kind	Livy id
Notebook 112_173cf5d-e767-4b1b-b867-2d7f...	Completed	Notebook	4:25 PM, 2/17/23	3d 9h 42m 57s	submitter	workspace	SparkSessionJob	Manual	173cf5d-e767-4b1b-b8...
Notebook 119_9981cce1-148e-4306-92c3-7aad...	Completed	Notebook	3:59 PM, 2/16/23	21m 4s	submitter	workspace	SparkSessionJob	Manual	9981cce1-148e-4306-92c...
Notebook 63	Failed	Notebook	11:11 AM, 9/9/22	0s	submitter	workspace			
Notebook 63	Failed	Notebook	11:09 AM, 9/9/22	1s	submitter	workspace			
Notebook 63	Failed	Notebook	11:07 AM, 9/9/22	0s	submitter	workspace			
Notebook 63	Failed	Notebook	11:05 AM, 9/9/22	1s	submitter	workspace			
Notebook 63	Failed	Notebook	11:03 AM, 9/9/22	1s	submitter	workspace			
Notebook 63	Failed	Notebook	11:01 AM, 9/9/22	0s	submitter	workspace			
Notebook 63	Failed	Notebook	10:59 AM, 9/9/22	0s	submitter	workspace			
Notebook 63	Failed	Notebook	10:57 AM, 9/9/22	0s	submitter	workspace			
Notebook 63	Failed	Notebook	10:55 AM, 9/9/22	0s	submitter	workspace			
Notebook 63	Failed	Notebook	10:53 AM, 9/9/22	0s	submitter	workspace			
Notebook 63	Failed	Notebook	10:51 AM, 9/9/22	1s	submitter	workspace			
Notebook 63	Failed	Notebook	10:49 AM, 9/9/22	1s	submitter	workspace			
Notebook 63	Failed	Notebook	10:47 AM, 9/9/22	0s	submitter	workspace			
Notebook 63	Failed	Notebook	10:47 AM, 9/9/22	0s	submitter	workspace			

## Filter Apache Spark applications

You can filter Apache Spark applications by **Status**, **Item Type**, **Start Time**, **Submitter**, and **Location** using the Filter pane in the upper-right corner.

Name	Status	Item type	Start time	Duration	Submitter	Location
Notebook	Completed	Notebook	5:02 PM, 5/9/23	21m 24s	submitter	workspace
Notebook	Completed	Notebook	4:30 PM, 5/9/23	2m 53s	submitter	workspace
Notebook	Completed	Notebook	4:29 PM, 5/9/23	1m 34s	submitter	workspace
Notebook	Completed	Notebook	4:23 PM, 5/9/23	31m 57s	submitter	workspace
Notebook	Failed	Notebook	4:04 PM, 5/9/23	1m 43s	submitter	workspace
Notebook	Completed	Notebook	4:01 PM, 5/9/23	21m 40s	submitter	workspace
Notebook	Failed	Notebook	3:59 PM, 5/9/23	1m 47s	submitter	workspace
Notebook 11	Failed	Notebook	3:54 PM, 5/9/23	1m 5s	submitter	workspace
Notebook 11	Failed	Notebook	3:54 PM, 5/9/23	1m 35s	submitter	workspace

## Search Apache Spark applications

To search for specific Apache Spark applications, you can enter certain keywords in the search box located in the upper-right corner.

Name	Status	Item type	Start time	Duration	Submitter	Location	Job type	Run kind	Livy id
Notebook-112_1737cf5d-e767-4b1b-b867-2d7f...	Completed	Notebook	4:25 PM, 2/17/23	3d 9h 42m 57s	submitter	workspace	SparkSessionJob	Manual	1737cf5d-e767-4b1b-b8...
Notebook-119_9981cce1-148e-430e-92c3-7aaad...	Completed	Notebook	3:59 PM, 2/16/23	21m 4s	submitter	workspace	SparkSessionJob	Manual	9981cce1-148e-430e-92c...
Notebook 63	Failed	Notebook	11:11 AM, 9/9/22	0s	submitter	workspace			
Notebook 63	Failed	Notebook	11:09 AM, 9/9/22	1s	submitter	workspace			
Notebook 63	Failed	Notebook	11:07 AM, 9/9/22	0s	submitter	workspace			
Notebook 63	Failed	Notebook	11:05 AM, 9/9/22	1s	submitter	workspace			
Notebook 63	Failed	Notebook	11:03 AM, 9/9/22	1s	submitter	workspace			
Notebook 63	Failed	Notebook	11:01 AM, 9/9/22	0s	submitter	workspace			
Notebook 63	Failed	Notebook	10:59 AM, 9/9/22	0s	submitter	workspace			
Notebook 63	Failed	Notebook	10:57 AM, 9/9/22	0s	submitter	workspace			
Notebook 63	Failed	Notebook	10:55 AM, 9/9/22	0s	submitter	workspace			
Notebook 63	Failed	Notebook	10:53 AM, 9/9/22	0s	submitter	workspace			
Notebook 63	Failed	Notebook	10:51 AM, 9/9/22	1s	submitter	workspace			
Notebook 63	Failed	Notebook	10:49 AM, 9/9/22	1s	submitter	workspace			
Notebook 63	Failed	Notebook	10:47 AM, 9/9/22	0s	submitter	workspace			
Notebook 63	Failed	Notebook	10:45 AM, 9/9/22	1s	submitter	workspace			

## Manage an Apache Spark application

When you hover over an Apache Spark application row, you can see various row-level actions that enable you to manage a particular Apache Spark application.

## View Apache Spark application detail pane

You can hover over an Apache Spark application row and click the **View details** icon to open the **Detail** pane and view more details about an Apache Spark application.

The screenshot shows the Azure Data Engineering Monitoring hub. On the left, there's a sidebar with various workspace options like Home, Monitoring hub, Workspaces, and TestChang05. The main area is titled 'Monitoring hub' and contains a table of Apache Spark applications. One row is highlighted with a red box around its 'View details' icon. A red arrow points from this icon to a detailed pane on the right. The pane is titled 'Details' and shows specific information for a 'Spark Application': Status (StoppedSessionTimedOut), Application name, Run kind (Manual), Livy Id (a long string of X's), Job Instance Id (another long string of X's), Submitted (5/9/23 5:02:39 PM), Submitter, Total duration (21m 21s), Running Duration (empty), and Queued Duration (empty). There's also a magnifying glass icon at the bottom right of the pane.

## Cancel an Apache Spark application

If you need to cancel an in-progress Apache Spark application, hover over its row and click the **Cancel** icon.

This screenshot shows the same Monitoring hub interface. A row for 'Notebook1' is highlighted with a red box around its 'Cancel' icon. The 'Cancel' icon is a red circle with a white minus sign. The rest of the table rows show 'Notebook2' and 'Notebook3' with statuses 'Failed' and 'Failed' respectively. The right side of the screen shows the 'Details' pane for the selected application, which is currently in progress.

## Navigate to Apache Spark application detail view

If you need more information about Apache Spark execution statistics, access Apache Spark logs, or check input and output data, you can click on the name of an Apache Spark application to navigate to its corresponding Apache Spark application detail page.

## Next steps

- [Apache Spark monitoring overview](#)
- [Browse item's recent runs](#)

- Monitor Apache Spark jobs within notebooks
- Monitor Apache Spark job definition
- Monitor Apache Spark application details

# Workspace item recent runs

Article • 05/23/2023

With Microsoft Fabric, you can use Apache Spark to run notebooks, Apache Spark job definitions, jobs, and other types of applications in your workspace. This article explains how to view your running Apache Spark applications, making it easier to keep an eye on the latest running status.

## Important

Microsoft Fabric is currently in PREVIEW. This information relates to a prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, expressed or implied, with respect to the information provided here.

## View the recent runs pane

We can open **Recent runs** pane with the following steps:

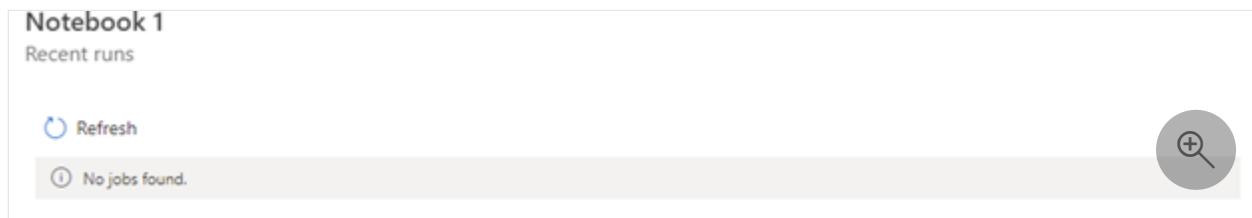
1. Open the Microsoft Fabric homepage and select a workspace where you want to run the job.
2. Selecting **Spark job definition** or **notebook item context menu** shows the recent run option.
3. Select **Recent runs**.

The screenshot illustrates the process of viewing recent runs for a specific notebook. In the main workspace, 'Notebook 1' is selected, and its context menu is open. The 'Recent runs' option is highlighted with a red box and a downward arrow pointing to the expanded pane below. The expanded pane, also framed in red, displays a list of recent runs for 'Notebook 1'. Each run entry includes the application name ('sparksession'), submission time, submitter, status, total duration, run kind, and Livy ID. The first three runs are in a 'Stopped (session)' state, while the last one is 'Cancelled'.

Application name	Submitted	Submitter	Status	Total dura...	Run kind	Livy Id
sparksession	7/29/22 11:50:54 AM	Submitter	Stopped (session)	10m 30s	-	0ec4ce2e-98dd-...
sparksession	7/29/22 11:20:09 AM	Submitter	Stopped (session)	11m 1s	-	3ec23d0c-9f82-4...
sparksession	7/29/22 11:03:04 AM	Submitter	Cancelled	5m 6s	-	f6a127d2-43ca-4...
sparksession	7/29/22 11:02:36 AM	Submitter	Cancelled	7m 37s	-	33ecbc82-a892...

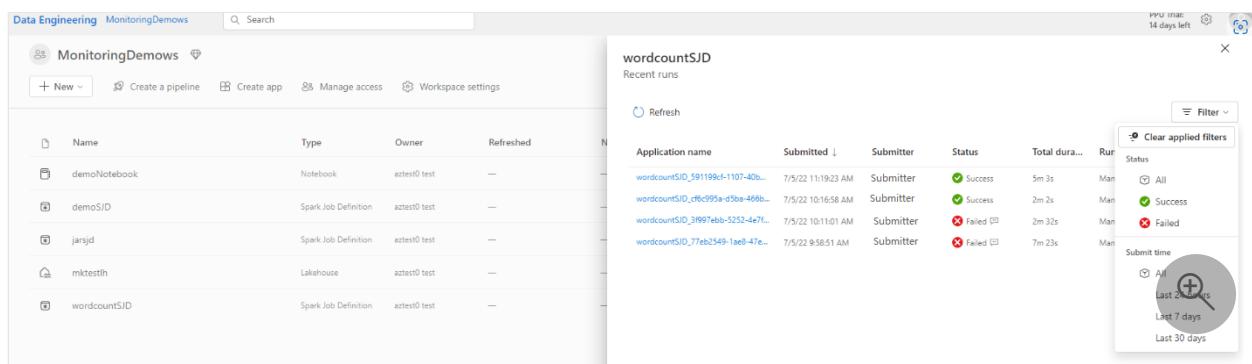
# Detail for recent run pane

If the notebook or Spark job definition doesn't have any run operations, the **Recent runs** page shows **No jobs found**.



The screenshot shows a 'Recent runs' pane with a 'Refresh' button and a message 'No jobs found.'

In the **Recent runs** pane, you can view a list of applications, including **Application name**, **Submitted time**, **Submitter**, **Status**, **Total duration**, **Run kind**, and **Livy Id**. You can filter applications by their status and submission time, which makes it easier for you to view applications.



The screenshot shows a 'Recent runs' pane with a table of applications and a filter sidebar.

Name	Type	Owner	Refreshed
demoNotebook	Notebook	aztest0 test	—
demoSID	Spark Job Definition	aztest0 test	—
jarsjd	Spark Job Definition	aztest0 test	—
mktestlh	Lakehouse	aztest0 test	—
wordcounSID	Spark Job Definition	aztest0 test	—

Filter sidebar:

- Clear applied filters
- Status:
  - All
  - Success
  - Failed
- Submit time:
  - All
  - Last 24 hours
  - Last 7 days
  - Last 30 days

Selecting the application name link navigates to spark application details where we can get to see the logs, data and skew details for the Spark run.

## Next steps

The next step after viewing the list of running Apache Spark applications is to view the application details. You can refer to:

- [Apache Spark application detail monitoring](#)

# Notebook contextual monitoring & debugging

Article • 05/23/2023

## ⓘ Important

Microsoft Fabric is currently in PREVIEW. This information relates to a prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, expressed or implied, with respect to the information provided here.

Notebook is purely Apache Spark based. Code cells are executed on the serverless remotely. A Spark job progress indicator is provided with a real-time progress bar appears to help you understand the job execution status.

## Monitor Job progress

Run the following sample code and validate the Spark job progress indicator below the Notebook cell.

Python

```
%%spark
import org.apache.spark.sql.functions.{ countDistinct, col, count, when }
val df = spark.range(0, 100000000)
df.select(df.columns.map(c => countDistinct(col(c)).alias(c)): _*).collect
```

The screenshot shows a Jupyter Notebook interface with a Python code cell and its execution output. The code performs a count distinct operation on a large dataset (100 million rows) and collects the results. The output indicates a successful execution with a duration of approximately 9 seconds. Below the code cell, a 'Spark jobs Succeeded' section displays a table of completed tasks. The table includes columns for ID, Description, Status, Stages, Tasks, and Duration. The tasks are categorized into two stages: Stage 0 and Stage 1, each with multiple stages (Job 0, Job 1, Stage 2). All tasks are marked as succeeded. A magnifying glass icon in the bottom right corner of the interface suggests a search or filter function.

ID	Description	Status	Stages	Tasks	Duration
Job 0	collect at <console>:31	SUCCEEDED	1/1	██████████	4 sec
Stage 0	collect at <console>:31	SUCCEEDED	-	██████████	4 sec
Job 1	collect at <console>:31	SUCCEEDED	1/1	██████████	< 1 ms
Stage 1	collect at <console>:31	SKIPPED	-		-
Stage 2	collect at <console>:31	SUCCEEDED	-	██████████	< 1 ms

```
1 %%spark
2 import org.apache.spark.sql.functions.{ countDistinct, col, count, when }
3 val df = spark.range(0, 100000000)
4 df.select(df.columns.map(c => countDistinct(col(c)).alias(c)): _*).collect
5
[1]: ✓ 19 sec - Apache Spark session started in 10 sec 336 ms. Command executed in 9 sec 378 ms
2:29:17 PM, 7/22/22
Scala
```

# Spark Advisor recommendation

Also supports viewing Spark Advisor info advice, after applying the advice, you would have chance to improve your execution performance, decrease cost and fix the execution failures. Run the sample code below and validate the Spark advisor info message below the Notebook cell.

Python

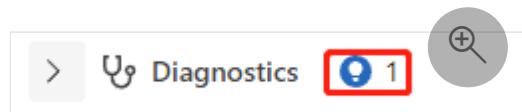
```
%%spark
val rdd = sc.parallelize(1 to 1000000)
val rdd2 = rdd.repartition(64)
val Array(train, test) = rdd2.randomSplit(Array(70, 30), 1)
train.takeOrdered(10)
```



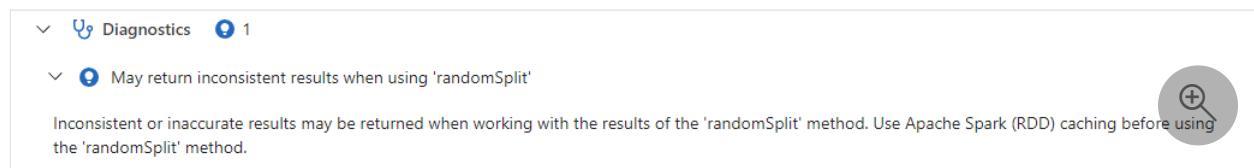
```
1 %%spark
2 val rdd = sc.parallelize(1 to 1000000)
3 val rdd2 = rdd.repartition(64)
4 val Array(train, test) = rdd2.randomSplit(Array(70, 30), 1)
5 train.takeOrdered(10)
6
[1] ✓ 19 sec - Apache Spark session started in 12 sec 281 ms. Command executed in 7 sec 529 ms
2:46:47 PM, 7/22/22
Scala
Spark Jobs Succeeded
Job runs
Progress indicator is out of sync. Reason: TypeError: Cannot read properties of null (reading 'jobs')
rdd: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[0] at parallelize at <console>:29
rdd2: org.apache.spark.rdd.RDD[Int] = MapPartitionsRDD[4] at repartition at <console>:29
train: org.apache.spark.rdd.RDD[Int] = MapPartitionsRDD[5] at randomSplit at <console>:29
test: org.apache.spark.rdd.RDD[Int] = MapPartitionsRDD[6] at randomSplit at <console>:29
res34: Array[Int] = Array(1, 2, 3, 4, 5, 7, 9, 10, 12, 14)
```

## View advice

Icon with blue light bulbs indicate suggestions are available for commands. The box shows the number of different suggestions.



Click the light bulb to expand and view the advice. One or more pieces of advice will become visible.



## Spark Advisor skew detection

It also supports viewing Spark Advisor skew detection. Run the sample code below and view the Data skew + time skew warning message below the Notebook cell.

Python

```
%%spark
import org.apache.spark.SparkContext
import java.util.Random
def testDataSkew(sc: SparkContext): Unit = {
    val numMappers = 400
    val numKVPairs = 100000
    val valSize = 256
    val numReducers = 200
    val biasPct = 0.4
    val biasCount = numKVPairs * biasPct
    for (i <- 1 to 2) {
        val query = sc.parallelize(0 until numMappers, numMappers).flatMap { p =>
            val ranGen = new Random
            val arr1 = new Array[(Int, Array[Byte])](numKVPairs)
            for (i <- 0 until numKVPairs) {
                val byteArr = new Array[Byte](valSize)
                ranGen.nextBytes(byteArr)
                var key = ranGen.nextInt(Int.MaxValue)
                if(i <= biasCount) {
                    key = 1
                }
                arr1(i) = (key, byteArr)
            }
            arr1
        }.groupByKey(numReducers)
        println(query.count())
    }
}
testDataSkew(sc)
```

The screenshot shows a Jupyter Notebook interface with two error messages displayed:

- Data skew for job 0**:  
Data Skew Analysis table:

Name	Max Task Dat...	Mean Task Da...	Task Data Read Skewness
Stage 1	4003.12 MB	50.31 MB	0.21
- Data skew for job 1**:  
Data Skew Analysis table:

Name	Max Task Dat...	Mean Task Da...	Task Data Read Skewness
Stage 3	4003.22 MB	50.31 MB	0.21

A red box highlights the "Data skew for job 0" message. A search icon is visible in the bottom right corner of the notebook area.

## Real-time logs

The log item is shown in cell output, which will display the real-time log. You can search keywords in searchbox or check filter errors and warnings to filter the logs you need.

```

[1] 25 testDataSkew(Sc)
    ✓ 1 min 57 sec - Apache Spark session started in 17 sec 766 ms. Command executed in 1 min 38 sec 560 ms by submitter on 4:49:03 PM, 4/06/23
    Spark (Scala) ▾
    ▾ Spark jobs (2 of 2 succeeded) Log
    Search Filter errors and warnings
    2023-04-06 08:48:54,089 INFO TaskSetManager [task-result-getter-2]: Finished task 193.0 in stage 13.0 (TID 1298) in 646 ms on vm-3cc46093 (exec
    2023-04-06 08:48:54,925 INFO TaskSetManager [task-result-getter-0]: Finished task 194.0 in stage 13.0 (TID 1299) in 229 ms on vm-3cc46093 (exec
    2023-04-06 08:48:54,988 INFO TaskSetManager [task-result-getter-3]: Finished task 195.0 in stage 13.0 (TID 1300) in 173 ms on vm-de560261 (exec
    2023-04-06 08:48:55,015 INFO TaskSetManager [task-result-getter-1]: Finished task 198.0 in stage 13.0 (TID 1303) in 152 ms on vm-3cc46093 (exec
    2023-04-06 08:48:55,018 INFO TaskSetManager [task-result-getter-2]: Finished task 199.0 in stage 13.0 (TID 1304) in 134 ms on vm-3cc46093 (exec
    2023-04-06 08:48:55,028 INFO TaskSetManager [task-result-getter-0]: Finished task 196.0 in stage 13.0 (TID 1301) in 195 ms on vm-de560261 (exec
    2023-04-06 08:48:55,035 INFO TaskSetManager [task-result-getter-3]: Finished task 197.0 in stage 13.0 (TID 1302) in 194 ms on vm-de560261 (exec
    ✓ 2023-04-06 08:49:02,830 INFO TaskSetManager [task-result-getter-1]: Finished task 1.0 in stage 13.0 (TID 1106) in 19350 ms on vm-de560261 (exec
    2023-04-06 08:49:02,830 INFO YarnClusterScheduler [task-result-getter-1]: Removed TaskSet 13.0, whose tasks have all completed, from pool
    2023-04-06 08:49:02,830 INFO DAGScheduler [dag-scheduler-event-loop]: ResultStage 13 (count at <console>:53) finished in 19.358 s
    2023-04-06 08:49:02,830 INFO DAGScheduler [dag-scheduler-event-loop]: Job 8 is finished. Cancelling potential speculative or zombie tasks for t
    2023-04-06 08:49:02,830 INFO YarnClusterScheduler [dag-scheduler-event-loop]: Killing all running tasks in stage 13: Stage finished
    ✓ 2023-04-06 08:49:02,831 INFO DAGScheduler [Thread-38]: Job 8 finished: count at <console>:53, took 44.148951 s
    23866440
    23866306
    2023-04-06 08:49:02,835 INFO KustoHandler [spark-listener-group-shared]: Logging DataSkew with appId: application_1680770743740_0001 to Kusto:
    ✓ 2023-04-06 08:49:02,837 INFO CreateAdviseEventHandler [spark-listener-group-shared]: Sending DataSkew to Advise Hub: Map(_source -> user, _jobId
  
```

## Access Spark UI and monitoring detail page

The number of tasks per each job or stage helps you to identify the parallel level of your spark job. You can also drill deeper to the Spark UI of a specific job (or stage) via selecting the link on the job (or stage) name. And you can also drill down to a specific job (or stage) by selecting the Spark Web UI link in the expand menu to open the Spark History Server.

ID	Description	Status	Stages	Tasks	Duration
Job 0	collect at <console>:31	<span>✓ Succeeded</span>	1/1	<div style="width: 100%;"> </div>	4 sec
Stage 0	collect at <console>:31	<span>✓ Succeeded</span>	-	<div style="width: 100%;"> </div>	4 sec
Job 1	collect at <console>:31	<span>✓ Succeeded</span>	1/1	<div style="width: 100%;"> </div>	< 1 ms
Stage 1	collect at <console>:31	<span>⌚ Skipped</span>	-	<div style="width: 100%;"> </div>	-
Stage 2	collect at <console>:31	<span>✓ Succeeded</span>	-	<div style="width: 100%;"> </div>	< 1 ms

```

import org.apache.spark.sql.functions.{countDistinct, col, count, when}
df: org.apache.spark.sql.Dataset[Long] = [id: bigint]
res34: Array[org.apache.spark.sql.Row] = Array([100000000])
  
```

## Next steps

- [Spark advisor](#)
- [Apache Spark application detail monitoring](#)

# Monitor your Apache Spark job definition

Article • 05/23/2023

Using the Spark job definition item's inline monitoring, you can track the following:

- Monitor the progress and status of a running Spark job definition.
- View the status and duration of previous Spark job definition runs.

## ⓘ Important

Microsoft Fabric is currently in PREVIEW. This information relates to a prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, expressed or implied, with respect to the information provided here.

You can get this information from the **Recent Runs** contextual menu in the workspace or by browsing the Spark job definition activities in the monitoring hub.

## Spark job definition inline monitoring

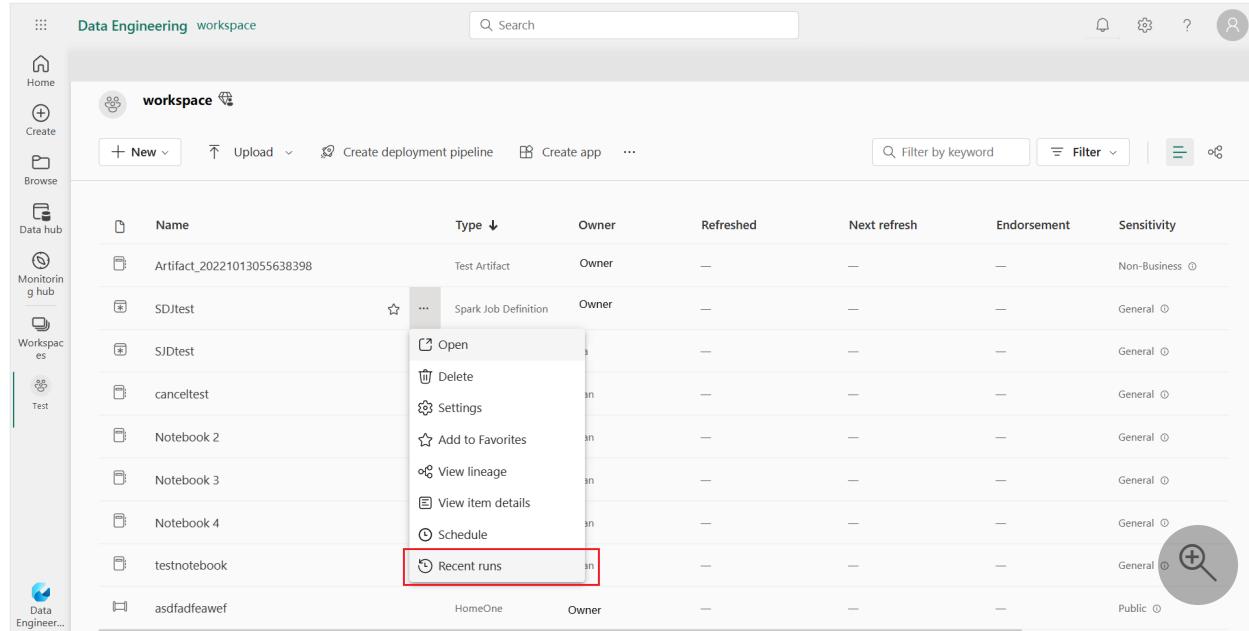
The Spark job definition inline monitoring feature allows you to view Spark job definition submission and run status in real-time. You can also view the Spark job definition's past runs and configurations and navigate to the **Spark application detail** page to view more details.

The screenshot shows the Microsoft Fabric monitoring hub. At the top, there is a navigation bar with links for Home, Notebooks, Datasets, Settings, Run, Language (set to PySpark (Python)), and a preview notice about Synapse notebooks and Spark job definitions. Below the navigation bar, a message states that uploaded files are in preview. A sidebar on the left shows a single uploaded file named 'Constant.py'. The main area is titled 'Runs' and contains a table of previous runs. The table has columns for Application name, Submitted, Submitter, Status, Total duration, Run kind, and Livy Id. The table is sorted by Name. The last row in the table is highlighted with a red border. The table includes a 'Refresh' button and a 'Filter' dropdown. The entire interface is framed by a red border.

Application name	Submitted	Submitter	Status	Total duration	Run kind	Livy Id
GuorongSJD1_35292d9c-cd8c-4568-a	5/5/23 12:13:56 PM	Submitter	✖ Failed	1m 59s	Manual	<a href="#">732064c8-c389-4be0-b55c-1f7...</a>
GuorongSJD1_35292d9c-cd8c-4568-a	5/5/23 12:13:54 PM	Submitter	✖ Failed	1m 5s	Manual	<a href="#">35292d9c-cd8c-4568-a104-672...</a>
GuorongSJD1_6ecb8bed-2d74-4134-9	5/5/23 12:13:40 PM	Submitter	✓ Success	1m 6s	Manual	<a href="#">1ce211b4-2e6c-4a66-b2eb-b7c...</a>
GuorongSJD1_d8295ffc-04a7-462e-a5	5/5/23 12:08:07 PM	Submitter	⌚ Cancelled	59s	Manual	<a href="#">58531dff-e4ad-42dc-a125-c306...</a>
GuorongSJD1_973c2f43-cf47-4fd4-99c	5/5/23 12:04:44 PM	Submitter	⌚ Cancelled	54s	Manual	<a href="#">ec15caa0-dfff-4481-86f0-e043...</a>
GuorongSJD1_960f0ffd-5142-4a0d-96	5/4/23 4:52:31 PM	Submitter	✓ Success	1m 6s	Manual	<a href="#">613b9735-c7cb-48ef-81a2-3f95...</a>
GuorongSJD1_fdc224e1-aa54-42d3-ac	5/4/23 4:43:04 PM	Submitter	✖ Failed	5m 7s	Manual	<a href="#">30955b3e-f9c6-4fbf-87a7-a5a3...</a>
GuorongSJD1_f844dd4-a903-4bdf-84	5/4/23 4:39:59 PM	Submitter	✓ Success	1m 6s	Manual	<a href="#">b2906ac3-7bd3-46ed-8d6c-8d6c...</a>

# Spark job definition item view in workspace

You can access the job runs associated with specific Spark job definition items by using the **Recent run** contextual menu on the workspace homepage.

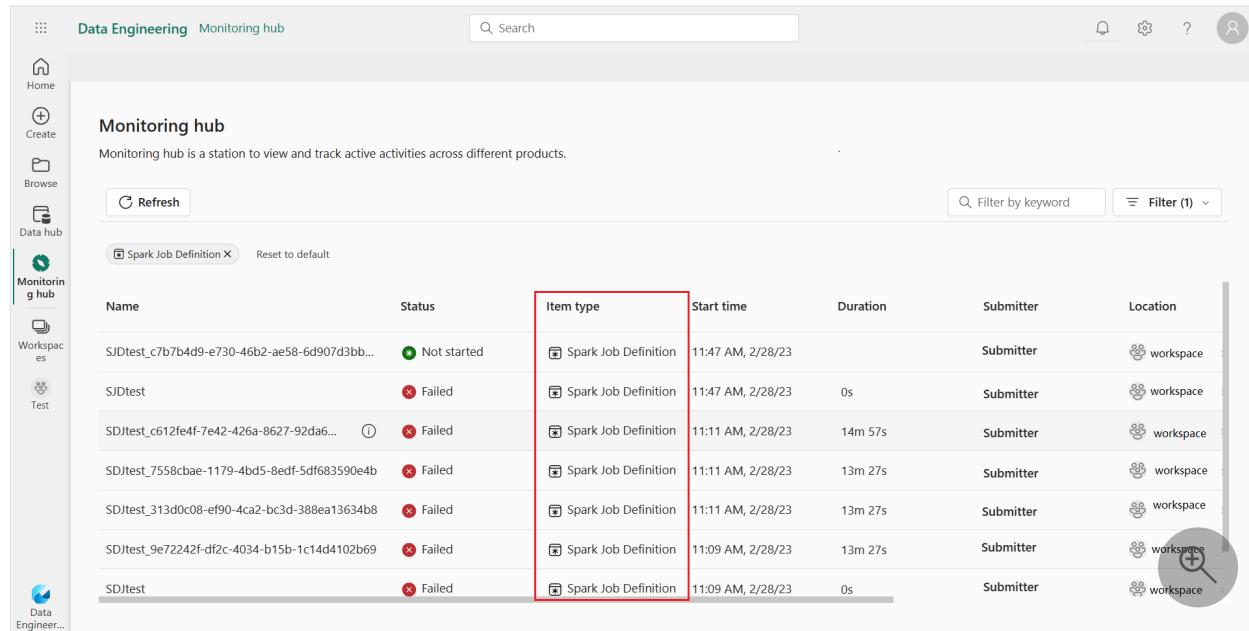


The screenshot shows the Data Engineering workspace homepage. On the left is a sidebar with icons for Home, Create, Browse, Data hub, Monitoring hub, Workspaces, Test, and Data Engineer... The main area displays a table of items. One item, 'testnotebook', has a context menu open over it. The menu options are: Open, Delete, Settings, Add to Favorites, View lineage, View item details, Schedule, and Recent runs. The 'Recent runs' option is highlighted and has a red box drawn around it. The table columns include Name, Type, Owner, Refreshed, Next refresh, Endorsement, and Sensitivity. The 'Type' column shows 'Spark Job Definition' for the 'testnotebook' item.

Name	Type	Owner	Refreshed	Next refresh	Endorsement	Sensitivity
Artifact_2022101305563898	Test Artifact	Owner	—	—	—	Non-Business ⓘ
SDJtest	Spark Job Definition	Owner	—	—	—	General ⓘ
SJDtest	Open	Owner	—	—	—	General ⓘ
canceltest	Delete	Owner	—	—	—	General ⓘ
Notebook 2	Settings	Owner	—	—	—	General ⓘ
Notebook 3	Add to Favorites	Owner	—	—	—	General ⓘ
Notebook 4	View lineage	Owner	—	—	—	General ⓘ
testnotebook	View item details	Owner	—	—	—	General ⓘ
testnotebook	Schedule	Owner	—	—	—	General ⓘ
testnotebook	Recent runs	Owner	—	—	—	General ⓘ
asdfadfeawef	HomeOne	Owner	—	—	—	Public ⓘ

# Spark job definition runs in the Monitoring hub

To view all the Spark applications related to a Spark job definition, go to the **Monitoring hub**. Sort or filter the **Item Type** column to view all the run activities associated with the Spark job definitions.



The screenshot shows the Data Engineering Monitoring hub. On the left is a sidebar with icons for Home, Create, Browse, Data hub, Monitoring hub, Workspaces, Test, and Data Engineer... The main area displays a table of Spark Job Definition runs. The 'Item type' column is highlighted and has a red box drawn around it. The table columns include Name, Status, Item type, Start time, Duration, Submitter, and Location. All runs listed are of type 'Spark Job Definition'.

Name	Status	Item type	Start time	Duration	Submitter	Location
SDJtest_c7b7b4d9-e730-46b2-ae58-6d907d3bb...	Not started	Spark Job Definition	11:47 AM, 2/28/23		Submitter	workspace
SJDtest	Failed	Spark Job Definition	11:47 AM, 2/28/23	0s	Submitter	workspace
SJDtest_c612fe4f-7e42-426a-8627-92da6...	Failed	Spark Job Definition	11:11 AM, 2/28/23	14m 57s	Submitter	workspace
SJDtest_7558cbea-1179-4bd5-8edf-5df683590e4b	Failed	Spark Job Definition	11:11 AM, 2/28/23	13m 27s	Submitter	workspace
SJDtest_313d0c08-ef90-4ca2-bc3d-388ea13634b8	Failed	Spark Job Definition	11:11 AM, 2/28/23	13m 27s	Submitter	workspace
SJDtest_9e72242f-df2c-4034-b15b-1c14d4102b69	Failed	Spark Job Definition	11:09 AM, 2/28/23	13m 27s	Submitter	workspace
SJDtest	Failed	Spark Job Definition	11:09 AM, 2/28/23	0s	Submitter	workspace

# Next steps

The next step after viewing the details of an Apache Spark application is to view Spark job progress below the Notebook cell. You can refer to

- [Spark application detail monitoring](#)

# Apache Spark application detail monitoring

Article • 05/23/2023

## ⓘ Important

Microsoft Fabric is currently in PREVIEW. This information relates to a prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, expressed or implied, with respect to the information provided here.

With Microsoft Fabric, you can use Apache Spark to run notebooks, jobs, and other kinds of applications in your workspace. This article explains how to monitor your Apache Spark application, allowing you to keep an eye on the recent run status, issues, and progress of your jobs.

## View Apache Spark applications

You can view all Apache Spark applications from **Spark job definition**, or **notebook item context** menu shows the recent run option -> **Recent runs**.

Application name	Submitted ↓	Submitter	Status	Total dura...	Run kind	Livy Id
sparksession	7/22/22 1:29:00 PM	Submitter	queued	-	-	833545fc-b0e2-4...
sparksession	7/22/22 12:05:08 PM	Submitter	Stopped (sessior	13m 20s	-	315f834c-ddc2-4...
sparksession	7/22/22 11:31:36 AM	Submitter	Stopped (sessior	13m 43s	-	16ea5191-1fb6-4...
sparksession	7/22/22 10:53:51 AM	Submitter	Stopped (sessior	11m 49s	-	8175f0ec-b857-4...
sparksession	7/21/22 7:35:08 PM	Submitter	Stopped (sessior	14m 39s	-	a29726dc-df3...
sparksession	7/21/22 1:03:50 PM	Submitter	Stopped (sessior	11m 33s	-	05ae3edc-d4e1-...
sparksession	7/21/22 12:59:03 PM	Submitter	Stopped (sessior	11m 24s	-	c5bc0149-ec1c-4...
sparksession	7/20/22 9:04:02 PM	Submitter	Stopped (sessior	10m 44s	-	0630e3cc-1778...
sparksession	7/15/22 8:35:42 AM	Submitter	Cancelled	3m 59s	-	90253fc9-d129...
sparksession	7/15/22 7:56:06 AM	Submitter	queued	-	-	6eae228a-51a4-...
-sArtifacts-s032ab742-de0da-d49fb...	7/14/22 7:40:35 AM	Submitter	Failed (...	15m 39s	-	d71040d7-73d1-...
-sArtifacts-s032ab742-de0da-d49fb...	7/14/22 7:08:36 AM	Submitter	Failed (...	15m 39s	-	e95196f8-0e9...
-sArtifacts-s032ab742-de0da-d49fb...	7/13/22 2:41:12 PM	Submitter	Failed (...	15m 40s	-	80e0f918-58fd-4...

You can select the name of the application you want to view in the application list, in the application details page you can view the application details.

## Monitor Apache Spark application status

Open the **Recent runs** page of the notebook or Spark job definition, you can view the status of the Apache application.

- Success

Application name	Submitted ↓	Submitter	Status	Total dura...	Run kind	Livy Id	
sparksession	7/29/22 11:02:36 AM	Submitter	Succeeded	7m 37s	-	33ecbc82-a892-...	

- Queued

Application name	Submitted ↓	Submitter	Status	Total dura...	Run kind	Livy Id	
sparksession	7/29/22 11:20:09 AM	Submitter	Queued	-	-	1213536sdva..	

- Stopped

Application name	Submitted ↓	Submitter	Status	Total dura...	Run kind	Livy Id	
sparksession	7/29/22 11:50:54 AM	Submitter	Stopped (session)	10m 30s	-	0ec4ce2e-98dd-...	

- Canceled

Application name	Submitted ↓	Submitter	Status	Total dura...	Run kind	Livy Id	
sparksession	7/29/22 11:02:36 AM	Submitter	Cancelled	7m 37s	-	33ecbc82-a892-...	

- Failed

Application name	Submitted ↓	Submitter	Status	Total dura...	Run kind	Livy Id	
sparksession	7/29/22 11:02:36 AM	Submitter	Failed	7m 37s	-	33ecbc82-a892-...	

## Jobs

Open an Apache Spark application job from the **Spark job definition** or **notebook** item context menu shows the **Recent run** option -> **Recent runs** -> select a job in recent runs page.

In the Apache Spark application monitoring details page, the job runs list is displayed in the **Jobs** tab, you can view the details of each job here, including **Job ID**, **Description**, **Status**, **Stages**, **Tasks**, **Duration**, **Processed**, **Data read**, **Data written** and **Code snippet**.

- Clicking on Job ID can expand/collapse the job.
- Click on the job description, you can jump to job or stage page in spark UI.
- Click on the job Code snippet, you can check and copy the code related to this job.

Notebook 2-1_5e080055-ec0c-401a-85c2-4fa2effb4bb4										
Actions		Jobs								
	Jobs	Logs	Data	Related items						
>	ID ↓	Description	Status	Stages	Tasks	Duration	Processed	Data read	Data written	Code snippet
>	Job 0	save at <console>:31	SUCCEEDED	1/1	3/3 succeeded	12 sec	3 rows	0 B	3.88 KB	<code>copy</code>
>	Job 1		SUCCEEDED	0/0	0/0 succeeded	< 1 ms	0 rows	0 B	0 B	<code>copy</code>
>	Job 2	\$anonfun\$recordDeltaOperation\$5 at SynapseLoggingShim.scala:95	SUCCEEDED	1/1	1/1 succeeded	9 sec	12 rows	2.14 KB	2.33 KB	<code>copy</code>
>	Job 3	\$anonfun\$recordDeltaOperation\$5 at SynapseLoggingShim.scala:95	SUCCEEDED	1/1	50/50 succeeded	4 sec	56 rows	2.33 KB	4.24 KB	<code>copy</code>
>	Job 4	\$anonfun\$recordDeltaOperation\$5 at SynapseLoggingShim.scala:95	SUCCEEDED	1/1	1/1 succeeded	< 1 ms	50 rows	4.24 KB	0 B	<code>copy</code>
>	Job 5	\$anonfun\$recordDeltaOperation\$5 at SynapseLoggingShim.scala:95	SUCCEEDED	1/1	50/50 succeeded	1 sec	5 rows	2.29 KB	0 B	<code>copy</code>
>	Job 6	takeAsList at <console>:36	SUCCEEDED	1/1	1/1 succeeded	1 sec	1 row	2.17 KB	0 B	<code>copy</code>

## Summary panel

In the Apache Spark application monitoring page, click the **Properties** button to open/collapse the summary panel. You can view the details for this application in **Details**.

- Status for this spark application.
- This Spark application's ID.
- Total duration.
- Running duration for this spark application.
- Queued duration for this spark application.
- Livy ID
- Submitter for this spark application.
- Submit time for this spark application.
- Number of executors.

	
<h2>Details</h2>	
Status	 Stopped (session timed out)
Application ID	application_1111111111111111
Total duration	22m 0s
Running duration	0
Queued duration	0
Livy ID	11111111111111111111111111111111111111
Submitter	Submitter
Submit time	3/31/23 4:21:39 PM
Number of executors	2 

## Logs

For the **Logs** tab, you can view the full log of **Livy**, **Prelaunch**, **Driver** log with different options selected in the left panel. And you can directly retrieve the required log information by searching keywords and view the logs by filtering the log status. Click Download Log to download the log information to the local.

Sometimes no logs are available, such as the status of the job is queued and cluster creation failed.

Live logs are only available when app submission fails, and driver logs are also provided.

Logs Log list. Select one << to check content

Driver (stderr) - stderr-active

Length: 229789 (102400 loaded)

↑ Load older log Load older logs

```

2023-05-05 04:14:38,014 INFO OutputCommitCoordinator$OutputCommitCoordinatorEndpoint [dispatcher-event-loop-0]: Successfully stopped SparkContext
2023-05-05 04:14:38,043 INFO SparkContext [shutdown-hook-0]: Unregistering ApplicationMaster
2023-05-05 04:14:38,059 INFO AMRMClientImpl [shutdown-hook-0]: Waiting for application to be succeeded
2023-05-05 04:14:38,160 INFO AMRMClientImpl [shutdown-hook-0]: Waiting for application to be succeeded
2023-05-05 04:14:38,262 INFO AMRMClientImpl [shutdown-hook-0]: Waiting for application to be succeeded
2023-05-05 04:14:38,383 WARN AzureFileSystemThreadPoolExecutor [shutdown-hook-0]: Deleting staging directory wasbs://1c50ae07-d66e-4822-b4c5-9184b563c61
2023-05-05 04:14:38,396 INFO AzureFileSystemThreadPoolExecutor [shutdown-hook-0]: Disabling threads for Delete operation as thread count
2023-05-05 04:14:38,416 INFO RpcAppSparkContextServer [shutdown-hook-0]: Time taken for Delete operation is: 13 ms with threads
2023-05-05 04:14:38,419 INFO ShutdownHookManager [shutdown-hook-0]: Closing remote SparkContext service at 10.1.128.5:18083, remote
2023-05-05 04:14:38,420 INFO ShutdownHookManager [shutdown-hook-0]: Shutting down hook called
2023-05-05 04:14:38,422 INFO ShutdownHookManager [shutdown-hook-0]: Deleting directory /mnt/var/hadoop/tmp/nm-secondary-local-dir/userca
2023-05-05 04:14:38,423 INFO ShutdownHookManager [shutdown-hook-0]: Deleting directory /mnt/var/hadoop/tmp/nm-secondary-local-dir/userca
2023-05-05 04:14:38,427 INFO RpcAppSender [shutdown-hook-0]: Closing RPC app sender
2023-05-05 04:14:38,428 INFO RpcAppSender [shutdown-hook-0]: Sending remaining events
2023-05-05 04:14:38,436 INFO RpcAppSender [shutdown-hook-0]: Sent RPC app end event of application_1683259812003_0001/1
2023-05-05 04:14:38,447 INFO RpcAppSender [shutdown-hook-0]: RPC app sender closed
2023-05-05 04:14:38,448 INFO RpcAppEventQueue [shutdown-hook-0]: Max number of events in queue: 160.
2023-05-05 04:14:38,516 INFO MetricsSystemImpl [shutdown-hook-0]: Stopping azure-file-system metrics system...
2023-05-05 04:14:38,516 INFO MetricsSystemImpl [shutdown-hook-0]: azure-file-system metrics system stopped.
2023-05-05 04:14:38,516 INFO MetricsSystemImpl [shutdown-hook-0]: azure-file-system metrics system shutdown complete.

```

End of LogType:stderr-active

\*\*\*\*\*

## Data

For the **Data** tab, you can copy the data list on clipboard, download the data list and single data, and check the properties for each data.

- The left panel can be expanded or collapse.
- The name, read format, size, source and path of the input and output files will be displayed in this list.
- The files in input and output can be downloaded, copy path and view properties.

relatedItems > Notebook > **Notebook\_ID**

Refresh Cancel Spark history server

Jobs Logs Data Related items

**Data** << Showing 1 - 1 of 1 items

	File name	Read for...	Size	Source
Input	DummyDeltaTable	delta	Failed to load	Blob storage
Output	00000000000000000000.json	delta	Failed to load	Blob storage
	part-00002-8345602c-731d...	Download	Failed to load	Blob storage
		Copy path	Failed to load	Blob storage
		Properties		

**Properties**

File name  
00000000000000000000.json

File Path  
wasbs://84cd7bd4-4071-41c6-95f9-ca...

Read format  
delta

Size  
Fail to load

Modified  
Fail to load

Group  
Fail to load

Owner  
Fail to load

Permissions  
Fail to load

Close

## Related items

The **Related items** tab allows you to browse and view items associated with the Apache Spark application, including Notebooks, Spark job definition, and/or Pipelines. The

related items page displays the snapshot of the code and parameter values at the time of execution for Notebooks. It also shows the snapshot of all settings and parameters at the time of submission for Spark job definitions. If the Apache Spark application is associated with a pipeline, the related item page also presents the corresponding pipeline and the Spark activity.

In the Related Items screen, you can:

- Browse and navigate the related items in the hierarchical tree.
- Click the 'A list of more actions' ellipse icon for each item to take different actions.
- Click the snapshot item to view its content.
- View the Breadcrumb to see the path from the selected item to the root.

The screenshot shows the 'Related items' screen with the following details:

- Breadcrumbs:** relatedItems > Notebook Root > **Notebook Root\_ID**
- Toolbar:** Refresh, Cancel, Spark history server, Help
- Navigation:** Jobs, Logs, Data, Related items (selected)
- Related items tree:** Notebook Root > Notebook 2 > Notebook 2-1
- Context menu for 'Notebook 2-1':** A list of more actions (highlighted with a red box). Other options include Save as notebook, Open this notebook, and Restore.
- Code pane:** print('Notebook 2-1')
- Job pane:** mssparkutils.notebook.run('Notebook 2-1-1')
- Details panel:** Snapshot ID, Snapshot ID (link), Livy ID, Livy ID (link), Job end time (3/22/23 11:15:43 AM), Duration (1 min 4 sec), Submitter, Default lakehouse, and a search icon.

## Diagnostics

The diagnostic panel provides users with real-time recommendations and error analysis, which are generated by the Spark Advisor through an analysis of the user's code. With built-in patterns, the Apache Spark Advisor helps users avoid common mistakes and analyzes failures to identify their root cause.

The Diagnostic panel interface includes:

- Header:** Click to check advisors, Drag to change the height of the panel, Expand/collapse diagnostic panel
- Counters:** Diagnostics (2), Clicks (3)
- Content:** Three advisor items:
  - Enable 'spark.advisor.divisionExprConvertRule.enable' to reduce rounding error propagation. This query contains the expression 'CAST(id AS DOUBLE) / CAST(id AS DOUBLE) / CAST(id AS DOUBLE)' with Double type. We recommend that you enable the configuration 'spark.advisor.divisionExprConvertRule.enable' which can help convert 'CAST(id AS DOUBLE) / CAST(id AS DOUBLE)' to 'CAST(id AS DOUBLE) / (CAST(id AS DOUBLE) \* 3.0D)' to reduce the rounding error propagation.
  - Enable 'spark.advisor.divisionExprConvertRule.enable' to reduce rounding error propagation. This query contains the expression '(CAST(id AS DOUBLE) + 1.0D) / (CAST(id AS DOUBLE) + 2.0D) / (CAST(id AS DOUBLE) \* 3.0D)' with Double type. We recommend that you enable the configuration 'spark.advisor.divisionExprConvertRule.enable' which can help convert '(CAST(id AS DOUBLE) + 1.0D) / (CAST(id AS DOUBLE) + 2.0D) / (CAST(id AS DOUBLE) \* 3.0D)' to '(CAST(id AS DOUBLE) + 1.0D) / (CAST(id AS DOUBLE) + 2.0D) \* (CAST(id AS DOUBLE) \* 3.0D)' to reduce the rounding error propagation.
  - Enable 'spark.advisor.divisionExprConvertRule.enable' to reduce rounding error propagation.
- Buttons:** Expand/collapse to check advice content (with a search icon)

## Next steps

The next step after viewing the details of an Apache Spark application is to view **Spark job progress** below the Notebook cell. You can refer to:

- [Notebook contextual monitoring and debugging](#)

# Use Python for Apache Spark

Article • 05/23/2023

## ⓘ Important

Microsoft Fabric is currently in PREVIEW. This information relates to a prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, expressed or implied, with respect to the information provided here.

Microsoft Fabric provides built-in Python support for Apache Spark. This includes support for [PySpark](#), which allows users to interact with Spark using familiar Spark or Python interfaces. You can analyze data using Python through Spark batch job definitions or with interactive Fabric notebooks. This document provides an overview of developing Spark applications in Synapse using the Python language.

## Create and run notebook sessions

Microsoft Fabric notebook is a web interface for you to create files that contain live code, visualizations, and narrative text. Notebooks are a good place to validate ideas and use quick experiments to get insights from your data. Notebooks are also widely used in data preparation, data visualization, machine learning, and other big data scenarios.

To get started with Python in Microsoft Fabric notebooks, change the primary **language** at the top of your notebook by setting the language option to *PySpark (Python)*.

In addition, you can use multiple languages in one notebook by specifying the language magic command at the beginning of a cell.

```
Python
```

```
%%pyspark  
# Enter your Python code here
```

To learn more about notebooks within Microsoft Fabric Analytics, see [How to use notebooks](#).

## Install packages

Libraries provide reusable code that you might want to include in your programs or projects. To make third party or locally built code available to your applications, you can install a library onto one of your workspace or notebook session.

To learn more about how to manage Python libraries, see [Python library management](#).

## Notebook utilities

Microsoft Spark Utilities (MSSparkUtils) is a built-in package to help you easily perform common tasks. You can use MSSparkUtils to work with file systems, to get environment variables, to chain notebooks together, and to work with secrets. MSSparkUtils is supported for PySpark notebooks.

To get started, you can run the following commands:

Python

```
from notebookutils import mssparkutils
mssparkutils.notebook.help()
```

Learn more about the supported MSSparkUtils commands at [Use Microsoft Spark Utilities](#).

## Use Pandas on Spark

The [Pandas API on Spark](#) allows you to scale your Pandas workload to any size by running it distributed across multiple nodes. If you are already familiar with pandas and want to leverage Spark for big data, pandas API on Spark makes you immediately productive and lets you migrate your applications without modifying the code. You can have a single codebase that works both with pandas (tests, smaller datasets) and with Spark (production, distributed datasets) and you can switch between the pandas API and the Pandas API on Spark easily and without overhead.

## Python runtime

The Microsoft Fabric Runtime is a curated environment optimized for data science and machine learning. The Microsoft Fabric runtime offers a range of popular, Python open-source libraries, including libraries like Pandas, PyTorch, Scikit-Learn, XGBoost, and more.

# Python visualization

The Python ecosystem offers multiple graphing libraries that come packed with many different features. By default, every Spark instance in Microsoft Fabric contains a set of curated and popular open-source libraries. You can also add or manage extra libraries or versions by using the Microsoft Fabric [library management capabilities](#).

Learn more about how to create Python visualizations by visiting [Python visualization](#).

## Next steps

- Learn how to use the Pandas API on Apache Spark: [Pandas API on Apache Spark](#)
- Manage Python libraries: [Python library management](#)
- Visualize data in Python: [Visualize data in Python](#)

# Manage Python libraries in Microsoft Fabric

Article • 05/23/2023

Libraries provide reusable code that you might want to include in your programs or projects. Each workspace comes with a pre-installed set of libraries available in the Spark run-time and available to be used immediately in the notebook or Spark job definition. We refer to these as built-in libraries. However, you might find that you need to include additional libraries for your machine learning scenario. This document describes how you can use Microsoft Fabric to install Python libraries for your data science workflows.

## Python libraries in Microsoft Fabric

Within Fabric, there are 2 methods to add additional Python libraries.

- **Feed library:** Feed libraries refer to the ones residing in public sources or repositories. We currently support Python feed libraries from PyPI and Conda, one can specify the source in Library Management portals.
- **Custom library:** Custom libraries are the code built by you or your organization. *.whl* and *.jar* can be managed through Library Management portals.

You can learn more about feed and custom libraries by going to the [manage libraries in Fabric documentation](#).

## Install workspace libraries

Workspace level libraries allow data scientists to standardize the sets of libraries and versions across all users in their workspace. Workspace library settings define the working environment for the entire workspace. The libraries installed on a workspace level are available for all notebooks and Spark job definitions under that workspace. Because these libraries are made available across sessions, it is best to use workspace libraries when you want to set up a shared environment for all sessions in a workspace.

### Important

Only Workspace admin has access to update the Workspace level settings.

You can use the workspace settings to install both Python feed and custom libraries. To learn more, you can visit [manage libraries in Fabric](#).

## Use workspace settings to manage feed libraries

In some cases, you may want to pre-install certain Python feed libraries from PyPI or Conda across all your notebook sessions. To do this, you can navigate to your workspace and manage these libraries through the [Python workspace settings](#).

From the Workspace setting, you can do the following:

- **View and search feed library:** The installed library list appears when you open the **library management panel**. From this view, you can see the library name, version, and related dependencies. You can also search to quickly find a library from this list.
- **Add new feed library:** You can add a new Python feed library from PyPI or Conda. Once the installation source is selected, you can select the + button and a new line appears. To add a library, you need to provide the library name and, optionally, specify the version in the next line. To upload a list of libraries at the same time, you can upload a `.yml` file containing the required dependencies.

## Use workspace settings to manage custom libraries

Using the Workspace setting, you can also make custom Python `.whl` files available for all notebooks in your workspace. Once the changes are saved, Fabric will install your custom libraries and their related dependencies.

## In-line installation

When developing a machine learning model or doing ad-hoc data analysis, you may need to quickly install a library for your Apache Spark session. To do this, you can use the in-line installation capabilities to quickly get started with new libraries.

### Note

In-line installation affects the current notebook session only. This means a new session will not include the packages installed in previous sessions.

We recommend placing all the in-line commands that add, delete, or update the Python packages in the first cell of your Notebook. The change of Python packages

will be effective after you restart the Python interpreter. The variables defined before running the command cell will be lost.

## Install Python feed libraries within a notebook

The `%pip` command in Microsoft Fabric is equivalent to the commonly used [pip](#) command in many data science workflows. The following section show examples of how you can use `%pip` commands to install feed libraries directly into your notebook.

1. Run the following commands in a Notebook code cell to install the *altair* library and *vega\_datasets*:

Python

```
%conda install altair      # install latest version through conda  
command  
%conda install vega_datasets  # install latest version through conda  
command
```

The log in the cell output indicates the result of installation.

2. Import the package and dataset by running the following codes in another Notebook cell:

Python

```
import altair as alt  
from vega_datasets import data
```

### ① Note

When installing new Python libraries, the `%conda install` command normally takes more time than `%pip install` since it will check the full dependencies to detect conflicts. You may want to use `%conda install` when you want to avoid potential issues. Use `%pip install` when you are certain about the library you are trying to install has no conflict with the pre-installed libraries in runtime environment.

### 💡 Tip

All available Python in-line commands and its clarifications can be found: [%pip commands](#) and [%conda commands](#)

# Manage custom Python libraries through in-line installation

In some cases, you may have a custom library that you want to quickly install for a notebook session. To do this, you can upload your custom Python library into your notebook-attached Lakehouse **File** folder.

To do this:

1. Navigate to your Lakehouse and select ... on the **File** folder.
2. Then, upload your custom Python `jar` or `wheel` library.
3. After uploading the file, you can use the following command to install the custom library to your notebook session:

Python

```
# install the .whl through pip command
%pip install /lakehouse/default/Files/wheel_file_name.whl
```

## Next steps

- Manage workspace settings: [Apache Spark workspace administration settings](#)
- Manage libraries in Fabric: [Manage libraries in Fabric documentation](#)

# Analyze data with Apache Spark and Python

Article • 05/23/2023

## ⓘ Important

Microsoft Fabric is currently in PREVIEW. This information relates to a prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, expressed or implied, with respect to the information provided here.

In this tutorial, you'll learn how to perform exploratory data analysis by using Azure Open Datasets and Apache Spark.

In particular, we'll analyze the [New York City \(NYC\) Taxi](#) dataset. The data is available through Azure Open Datasets. This subset of the dataset contains information about yellow taxi trips: information about each trip, the start and end time and locations, the cost, and other interesting attributes.

## Prerequisites

## ⓘ Important

Microsoft Fabric is currently in PREVIEW. This information relates to a prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, expressed or implied, with respect to the information provided here.

## Download and prepare the data

1. Create a notebook by using PySpark. For instructions, see [Create a notebook](#).

## ⓘ Note

Because of the PySpark kernel, you don't need to create any contexts explicitly. The Spark context is automatically created for you when you run the first code cell.

2. In this tutorial, we'll use several different libraries to help us visualize the dataset.

To do this analysis, import the following libraries:

Python

```
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
```

3. Because the raw data is in a Parquet format, you can use the Spark context to pull the file into memory as a DataFrame directly. Create a Spark DataFrame by retrieving the data via the Open Datasets API. Here, we use the Spark DataFrame *schema on read* properties to infer the datatypes and schema.

Python

```
from azureml.opendatasets import NycTlcYellow

end_date = parser.parse('2018-06-06')
start_date = parser.parse('2018-05-01')
nyc_tlc = NycTlcYellow(start_date=start_date, end_date=end_date)
nyc_tlc_pd = nyc_tlc.to_pandas_dataframe()

df = spark.createDataFrame(nyc_tlc_pd)
```

4. After the data is read, we'll want to do some initial filtering to clean the dataset.

We might remove unneeded columns and add columns that extract important information. In addition, we'll filter out anomalies within the dataset.

Python

```
# Filter the dataset
from pyspark.sql.functions import *

filtered_df = df.select('vendorID', 'passengerCount',
    'tripDistance', 'paymentType', 'fareAmount', 'tipAmount' \
        , date_format('tpepPickupDateTime', \
    'hh').alias('hour_of_day')\ \
        , \
    dayofweek('tpepPickupDateTime').alias('day_of_week')\ \
        , \
    dayofmonth(col('tpepPickupDateTime')).alias('day_of_month'))\ \
        .filter((df.passengerCount > 0) \& (df.tipAmount >= 0) \& (df.fareAmount >= 1) & (df.fareAmount \
    <= 250) \& (df.tripDistance > 0) & (df.tripDistance <= 200))
```

```
filtered_df.createOrReplaceTempView("taxi_dataset")
```

## Analyze data

As a data analyst, you have a wide range of tools available to help you extract insights from the data. In this part of the tutorial, we'll walk through a few useful tools available within Microsoft Fabric notebooks. In this analysis, we want to understand the factors that yield higher taxi tips for our selected period.

## Apache Spark SQL Magic

First, we'll perform exploratory data analysis by Apache Spark SQL and magic commands with the Microsoft Fabric notebook. After we have our query, we'll visualize the results by using the built-in `chart options` capability.

1. Within your notebook, create a new cell and copy the following code. By using this query, we want to understand how the average tip amounts have changed over the period we've selected. This query will also help us identify other useful insights, including the minimum/maximum tip amount per day and the average fare amount.

SQL

```
%%sql
SELECT
    day_of_month
    , MIN(tipAmount) AS minTipAmount
    , MAX(tipAmount) AS maxTipAmount
    , AVG(tipAmount) AS avgTipAmount
    , AVG(fareAmount) as fareAmount
FROM taxi_dataset
GROUP BY day_of_month
ORDER BY day_of_month ASC
```

2. After our query finishes running, we can visualize the results by switching to the chart view. This example creates a line chart by specifying the `day_of_month` field as the key and `avgTipAmount` as the value. After you've made the selections, select `Apply` to refresh your chart.

## Visualize data

In addition to the built-in notebook charting options, you can use popular open-source libraries to create your own visualizations. In the following examples, we'll use Seaborn and Matplotlib. These are commonly used Python libraries for data visualization.

1. To make development easier and less expensive, we'll downsample the dataset.

We'll use the built-in Apache Spark sampling capability. In addition, both Seaborn and Matplotlib require a Pandas DataFrame or NumPy array. To get a Pandas DataFrame, use the `toPandas()` command to convert the DataFrame.

```
Python
```

```
# To make development easier, faster, and less expensive, downsample
# for now
sampled_taxi_df = filtered_df.sample(True, 0.001, seed=1234)

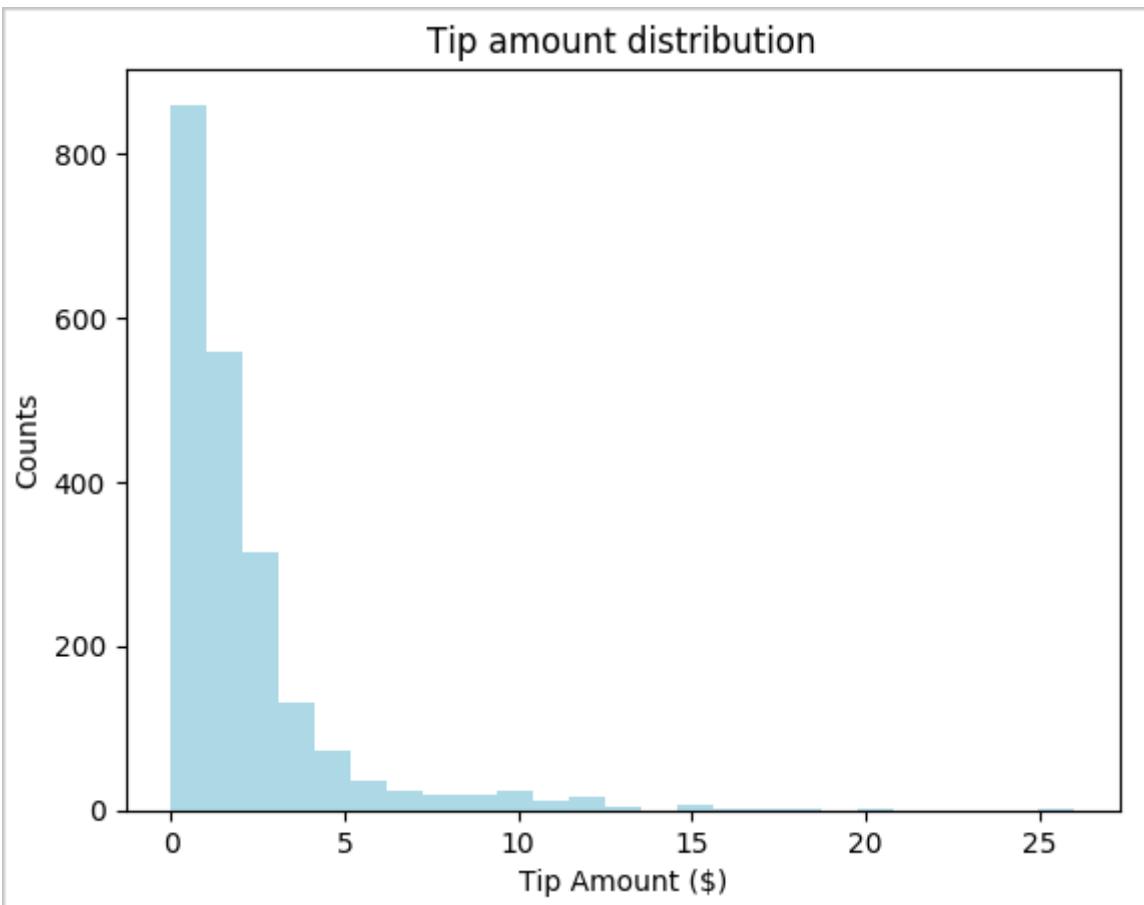
# The charting package needs a Pandas DataFrame or NumPy array to do
# the conversion
sampled_taxi_pd_df = sampled_taxi_df.toPandas()
```

2. We want to understand the distribution of tips in our dataset. We'll use Matplotlib to create a histogram that shows the distribution of tip amount and count. Based on the distribution, we can see that tips are skewed toward amounts less than or equal to \$10.

```
Python
```

```
# Look at a histogram of tips by count by using Matplotlib

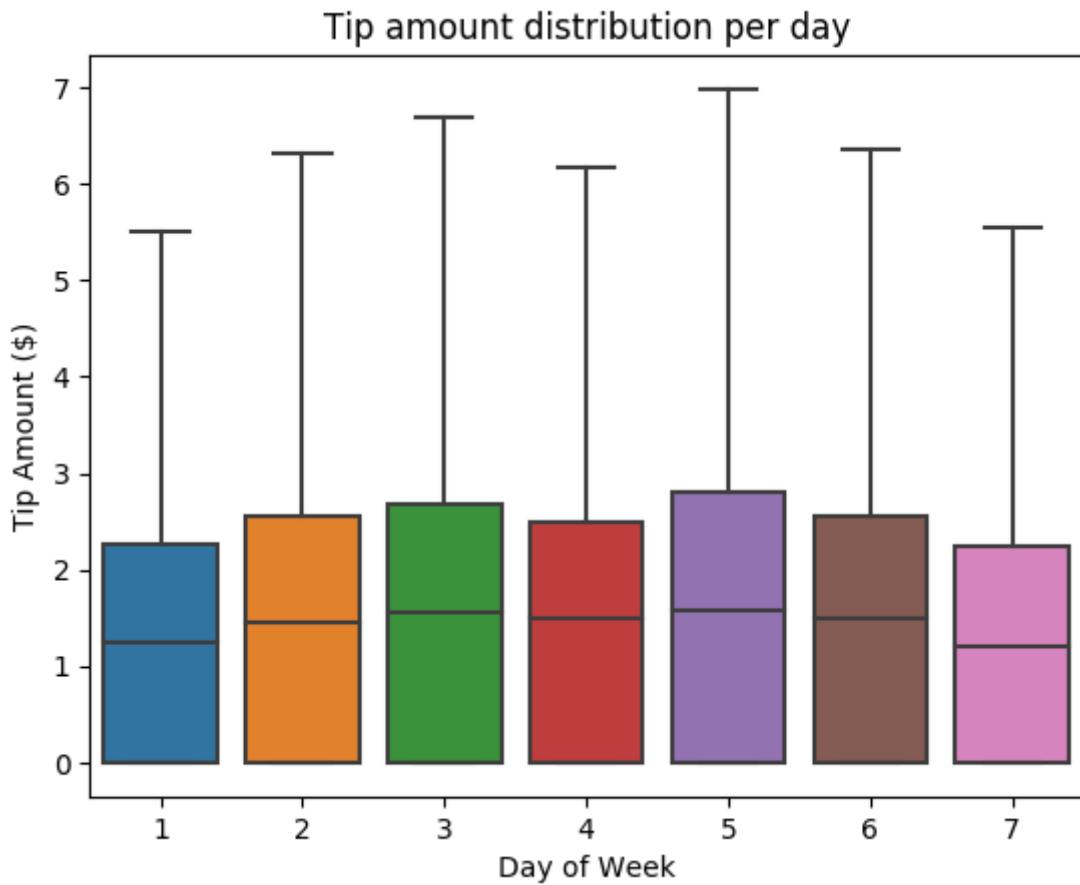
ax1 = sampled_taxi_pd_df['tipAmount'].plot(kind='hist', bins=25,
facecolor='lightblue')
ax1.set_title('Tip amount distribution')
ax1.set_xlabel('Tip Amount ($)')
ax1.set_ylabel('Counts')
plt.suptitle('')
plt.show()
```



3. Next, we want to understand the relationship between the tips for a given trip and the day of the week. Use Seaborn to create a box plot that summarizes the trends for each day of the week.

Python

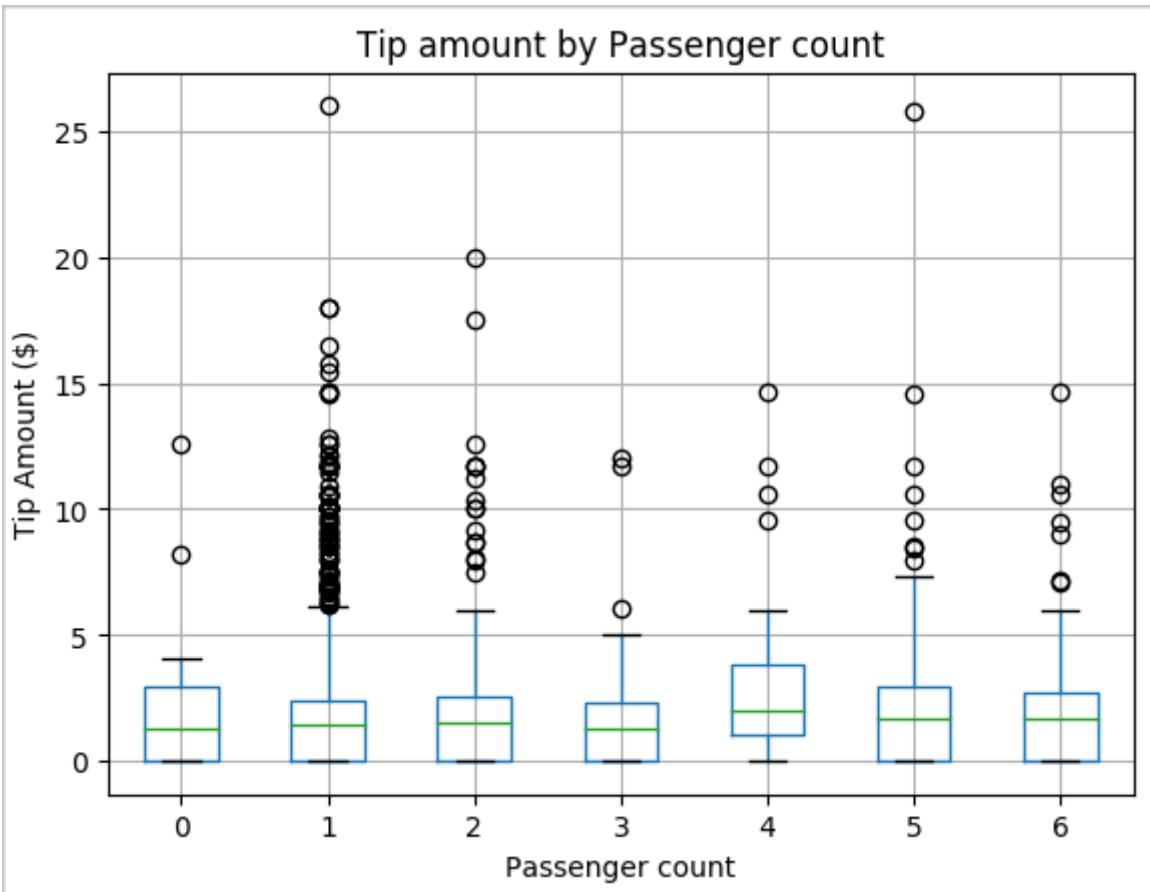
```
# View the distribution of tips by day of week using Seaborn
ax = sns.boxplot(x="day_of_week",
y="tipAmount",data=sampled_taxi_pd_df, showfliers = False)
ax.set_title('Tip amount distribution per day')
ax.set_xlabel('Day of Week')
ax.set_ylabel('Tip Amount ($)')
plt.show()
```



4. Another hypothesis of ours might be that there's a positive relationship between the number of passengers and the total taxi tip amount. To verify this relationship, run the following code to generate a box plot that illustrates the distribution of tips for each passenger count.

Python

```
# How many passengers tipped by various amounts
ax2 = sampled_taxi_pd_df.boxplot(column=['tipAmount'], by=['passengerCount'])
ax2.set_title('Tip amount by Passenger count')
ax2.set_xlabel('Passenger count')
ax2.set_ylabel('Tip Amount ($)')
ax2.set_ylim(0,30)
plt.suptitle('')
plt.show()
```

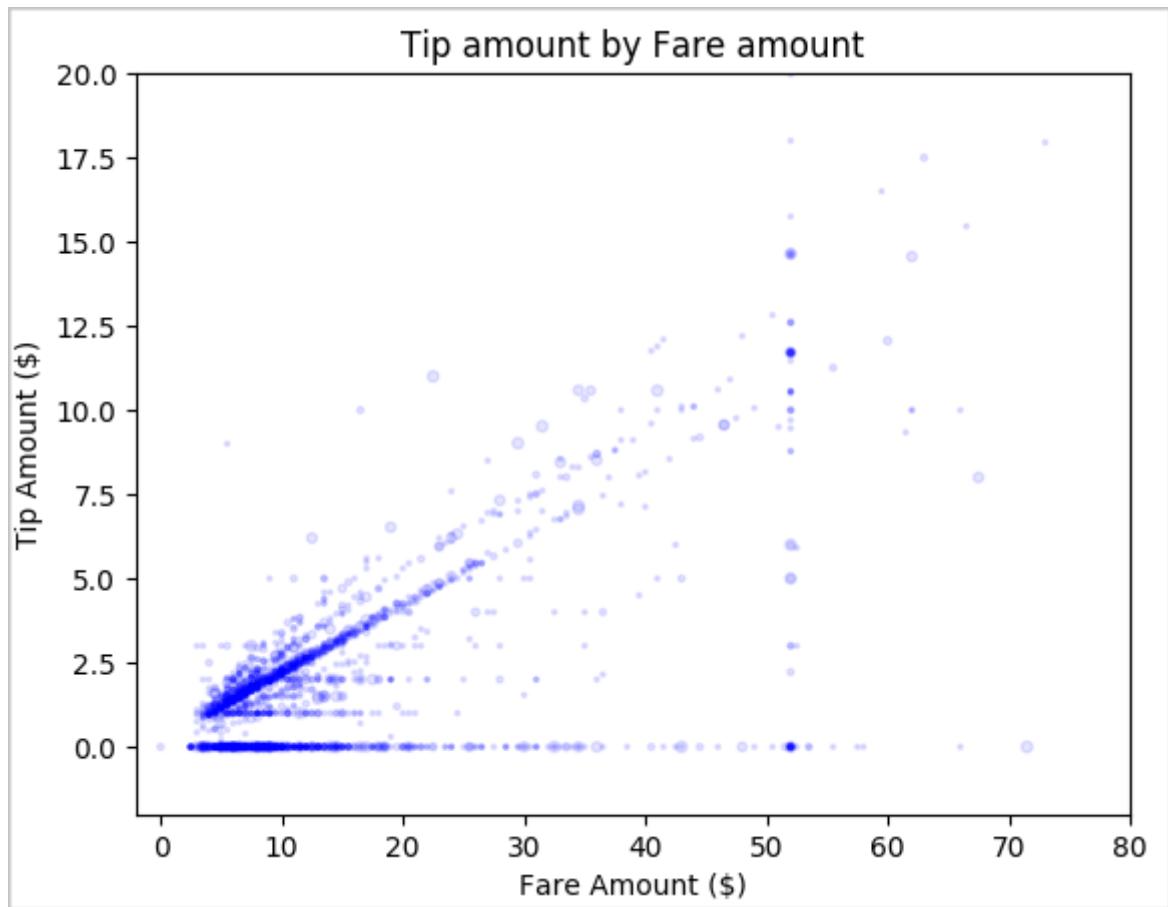


5. Last, we want to understand the relationship between the fare amount and the tip amount. Based on the results, we can see that there are several observations where people don't tip. However, we also see a positive relationship between the overall fare and tip amounts.

Python

```
# Look at the relationship between fare and tip amounts

ax = sampled_taxi_pd.plot(kind='scatter', x= 'fareAmount', y =
'tipAmount', c='blue', alpha = 0.10, s=2.5*
(sampled_taxi_pd[ 'passengerCount']))
ax.set_title('Tip amount by Fare amount')
ax.set_xlabel('Fare Amount ($)')
ax.set_ylabel('Tip Amount ($)')
plt.axis([-2, 80, -2, 20])
plt.suptitle('')
plt.show()
```



## Next steps

- Learn how to use the Pandas API on Apache Spark: [Pandas API on Apache Spark ↗](#)
- Manage Python libraries: [Python library management](#)

# Use R for Apache Spark

Article • 05/23/2023

Microsoft Fabric provides built-in R support for Apache Spark. This includes support for [SparkR](#) and [sparklyr](#), which allows users to interact with Spark using familiar Spark or R interfaces. You can analyze data using R through Spark batch job definitions or with interactive Microsoft Fabric notebooks.

## ⓘ Important

Microsoft Fabric is currently in PREVIEW. This information relates to a prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, expressed or implied, with respect to the information provided here.

This document provides an overview of developing Spark applications in Synapse using the R language.

## Prerequisites

- A Power BI Premium subscription. If you don't have one, see [How to purchase Power BI Premium](#).
- A Power BI workspace with assigned Premium capacity. If you don't have a workspace, use the steps in [Create a workspace](#) to create one and assign it to a Premium capacity.
- Sign in to [Microsoft Fabric](#).

## Create and run notebook sessions

Microsoft Fabric notebook is a web interface for you to create files that contain live code, visualizations, and narrative text. Notebooks are a good place to validate ideas and use quick experiments to get insights from your data. Notebooks are also widely used in data preparation, data visualization, machine learning, and other big data scenarios.

To get started with R in Microsoft Fabric notebooks, change the primary **language** at the top of your notebook by setting the language option to *SparkR (R)*.

In addition, you can use multiple languages in one notebook by specifying the language magic command at the beginning of a cell.

```
R  
%%sparkr  
# Enter your R code here
```

To learn more about notebooks within Microsoft Fabric Analytics, see [How to use notebooks](#).

## Install packages

Libraries provide reusable code that you might want to include in your programs or projects. To make third party or locally built code available to your applications, you can install a library onto one of your workspace or notebook session.

To learn more about how to manage R libraries, see [R library management](#).

## Notebook utilities

Microsoft Spark Utilities (MSSparkUtils) is a built-in package to help you easily perform common tasks. You can use MSSparkUtils to work with file systems, to get environment variables, to chain notebooks together, and to work with secrets. MSSparkUtils is supported for R notebooks.

To get started, you can run the following commands:

```
SparkR  
  
library(notebookutils)  
mssparkutils.fs.help()
```

Learn more about the supported MSSparkUtils commands at [Use Microsoft Spark Utilities](#).

## Use SparkR

[SparkR](#) is an R package that provides a light-weight frontend to use Apache Spark from R. SparkR provides a distributed data frame implementation that supports operations like selection, filtering, aggregation etc. SparkR also supports distributed machine learning using MLlib.

You can learn more about how to use SparkR by visiting [How to use SparkR](#).

## Use sparklyr

[sparklyr](#) is an R interface to Apache Spark. It provides a mechanism to interact with Spark using familiar R interfaces. You can use sparklyr through Spark batch job definitions or with interactive Microsoft Fabric notebooks.

To learn more about how to use sparklyr, visit [How to use sparklyr](#).

 Note

Using SparkR and sparklyr in the same notebook session isn't supported yet.

## Use Tidyverse

[Tidyverse](#) is a collection of R packages that data scientists commonly use in everyday data analyses. It includes packages for data import (`readr`), data visualization (`ggplot2`), data manipulation (`dplyr`, `tidyr`), functional programming (`purrr`), and model building (`tidymodels`) etc. The packages in `tidyverse` are designed to work together seamlessly and follow a consistent set of design principles. Microsoft Fabric distributes the latest stable version of `tidyverse` with every runtime release.

To learn more about how to use Tidyverse, visit [How to use Tidyverse](#).

## R visualization

The R ecosystem offers multiple graphing libraries that come packed with many different features. By default, every Spark instance in Microsoft Fabric contains a set of curated and popular open-source libraries. You can also add or manage extra libraries or versions by using the Microsoft Fabric [library management capabilities](#).

Learn more about how to create R visualizations by visiting [R visualization](#).

## Next steps

- [How to use SparkR](#)
- [How to use sparklyr](#)
- [How to use Tidyverse](#)

- R library management
- Visualize data in R
- Tutorial: avocado price prediction
- Tutorial: flight delay prediction

# Tutorial: Avocado price prediction with R

Article • 05/23/2023

This article shows an end to end example of using R to analyze, visualize the avocado prices in the US, and predict future avocado prices.

## ⓘ Important

Microsoft Fabric is currently in PREVIEW. This information relates to a prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, expressed or implied, with respect to the information provided here.



## Prerequisites

- A Power BI Premium subscription. If you don't have one, see [How to purchase Power BI Premium](#).
- A Power BI workspace with assigned Premium capacity. If you don't have a workspace, use the steps in [Create a workspace](#) to create one and assign it to a Premium capacity.
- Sign in to [Microsoft Fabric](#).

- Open or create a notebook. To learn how, see [How to use Microsoft Fabric notebooks](#).
- Change the primary language by setting the **language option** to **SparkR (R)**.
- Attach your notebook to a lakehouse. On the left side, select **Add** to add an existing lakehouse or create a lakehouse.

## Load libraries

Use libraries from the default R runtime.

R

```
library(tidyverse)
library(lubridate)
library(hms)
```

## Load the data

Read the avocado prices from a .csv file on the internet.

R

```
df <- 
read.csv('https://synapseaisolutionsa.blob.core.windows.net/public/AvocadoPrice/avocado.csv', header = TRUE)
head(df,5)
```

## Manipulate the data

First, rename the data to have friendlier names.

R

```
# to lower case
names(df) <- tolower(names(df))

# to snake case
avocado <- df %>%
  rename("av_index" = "x",
         "average_price" = "averageprice",
         "total_volume" = "total.volume",
         "total_bags" = "total.bags",
         "amount_from_small_bags" = "small.bags",
```

```

  "amount_from_large_bags" = "large.bags",
  "amount_from_xlarge_bags" = "xlarge.bags")

# Rename codes
avocado2 <- avocado %>%
  rename("PLU4046" = "x4046",
        "PLU4225" = "x4225",
        "PLU4770" = "x4770")

head(avocado2,5)

```

Change the data types, remove unwanted columns, and add total consumption.

R

```

# Convert data
avocado2$year = as.factor(avocado2$year)
avocado2$date = as.Date(avocado2$date)
avocado2$month = factor(months(avocado2$date), levels = month.name)
avocado2$average_price = as.numeric(avocado2$average_price)
avocado2$PLU4046 = as.double(avocado2$PLU4046)
avocado2$PLU4225 = as.double(avocado2$PLU4225)
avocado2$PLU4770 = as.double(avocado2$PLU4770)
avocado2$amount_from_small_bags =
  as.numeric(avocado2$amount_from_small_bags)
avocado2$amount_from_large_bags =
  as.numeric(avocado2$amount_from_large_bags)
avocado2$amount_from_xlarge_bags =
  as.numeric(avocado2$amount_from_xlarge_bags)

# # Remove unwanted columns
avocado2 <- avocado2 %>%
  select(-av_index, -total_volume, -total_bags)

# Calculate total consumption
avocado2 <- avocado2 %>%
  mutate(total_consumption = PLU4046 + PLU4225 + PLU4770 +
    amount_from_small_bags + amount_from_large_bags + amount_from_xlarge_bags)

```

## Install new package

Use the inline package installation to add new packages to the session.

R

```
install.packages(c("repr", "gridExtra", "fpp2"))
```

Load the libraries to use.

R

```
library(tidyverse)
library(knitr)
library(repr)
library(gridExtra)
library(data.table)
```

## Analyze and visualize the data

Compare conventional (nonorganic) avocado prices by region.

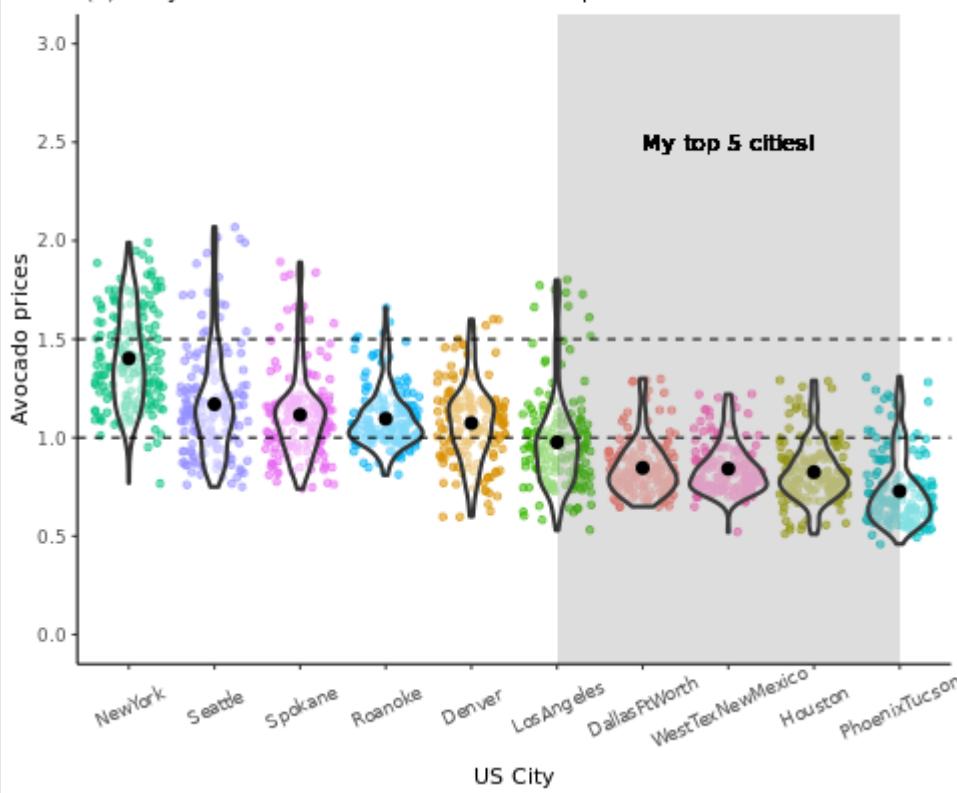
R

```
options(repr.plot.width = 10, repr.plot.height =10)
# filter(mydata, gear %in% c(4,5))
avocado2 %>%
  filter(region %in%
c("PhoenixTucson", "Houston", "WestTexNewMexico", "DallasFtWorth", "LosAngeles",
"Denver", "Roanoke", "Seattle", "Spokane", "NewYork")) %>%
  filter(type == "conventional") %>%
  select(date, region, average_price) %>%
  ggplot(aes(x = reorder(region, -average_price, na.rm = T), y =
average_price)) +
  geom_jitter(aes(colour = region, alpha = 0.5)) +
  geom_violin(outlier.shape = NA, alpha = 0.5, size = 1) +
  geom_hline(yintercept = 1.5, linetype = 2) +
  geom_hline(yintercept = 1, linetype = 2) +
  annotate("rect", xmin = "LosAngeles", xmax = "PhoenixTucson", ymin = -Inf,
ymax = Inf, alpha = 0.2) +
  geom_text(x = "WestTexNewMexico", y = 2.5, label = "My top 5 cities!",
hjust = 0.5) +
  stat_summary(fun = "mean") +
  labs(x = "US City",
y = "Avocado prices",
title = "Figure 1. Violin plot of Non-organic Avocado Prices",
subtitle = "Visual aids: \n(1) Black dots are average price of
individual avocado by city \n      between Jan-2015 to Mar-2018, \n (2) the
plot has been ordered descendingly,\n (3) Body of violin become fatter when
data points increase.") +
  theme_classic() +
  theme(legend.position = "none",
        axis.text.x = element_text(angle = 25, vjust = 0.65),
        plot.title = element_text(face = "bold", size = 15)) +
  scale_y_continuous(lim = c(0, 3), breaks = seq(0, 3, 0.5))
```

## Figure 1. Violin plot of Non-organic Avocado Prices

Visual aids:

- (1) Black dots are average price of individual avocado by city between Jan-2015 to Mar-2018,
- (2) the plot has been ordered descendingly,
- (3) Body of violin become fatter when data points increase.



Look specifically at the Houston region.

R

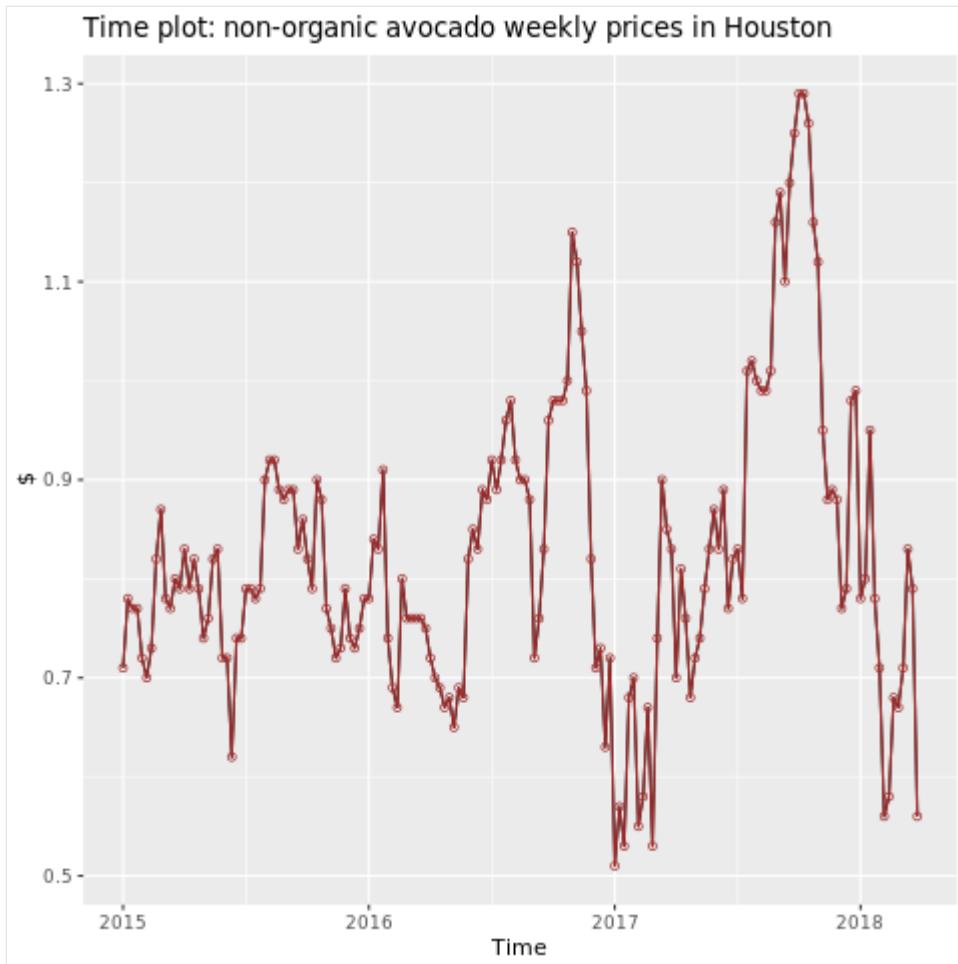
```
library(fpp2)
conv_houston <- avocado2 %>%
  filter(region == "Houston",
        type == "conventional") %>%
  group_by(date) %>%
  summarise(average_price = mean(average_price))

# set up ts

conv_houston_ts <- ts(conv_houston$average_price,
                       start = c(2015, 1),
                       frequency = 52)

# plot

autoplot(conv_houston_ts) +
  labs(title = "Time plot: non-organic avocado weekly prices in Houston",
       y = "$") +
  geom_point(colour = "brown", shape = 21) +
  geom_path(colour = "brown")
```



## Train a machine learning model

Build a price prediction model for the Houston area.

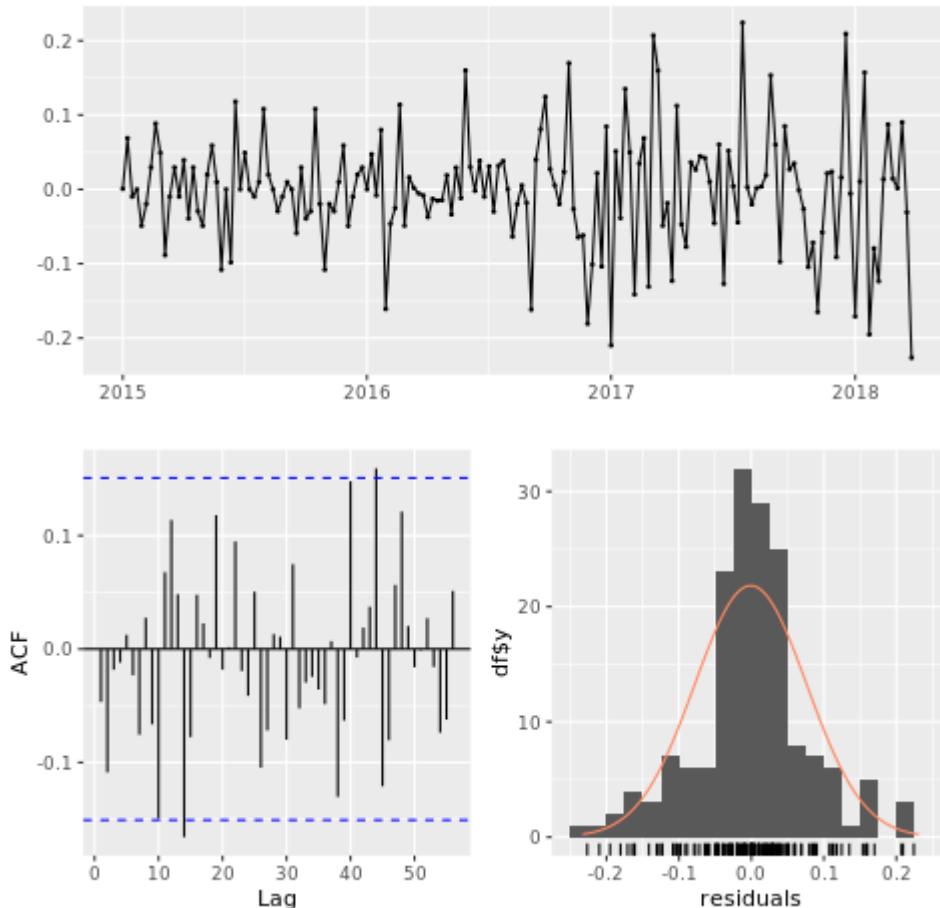
R

```
conv_houston_ts_arima <- auto.arima(conv_houston_ts,
                                         d = 1,
                                         approximation = F,
                                         stepwise = F,
                                         trace = T)
```

R

```
checkresiduals(conv_houston_ts_arima)
```

### Residuals from ARIMA(0,1,0)(0,0,1)[52]

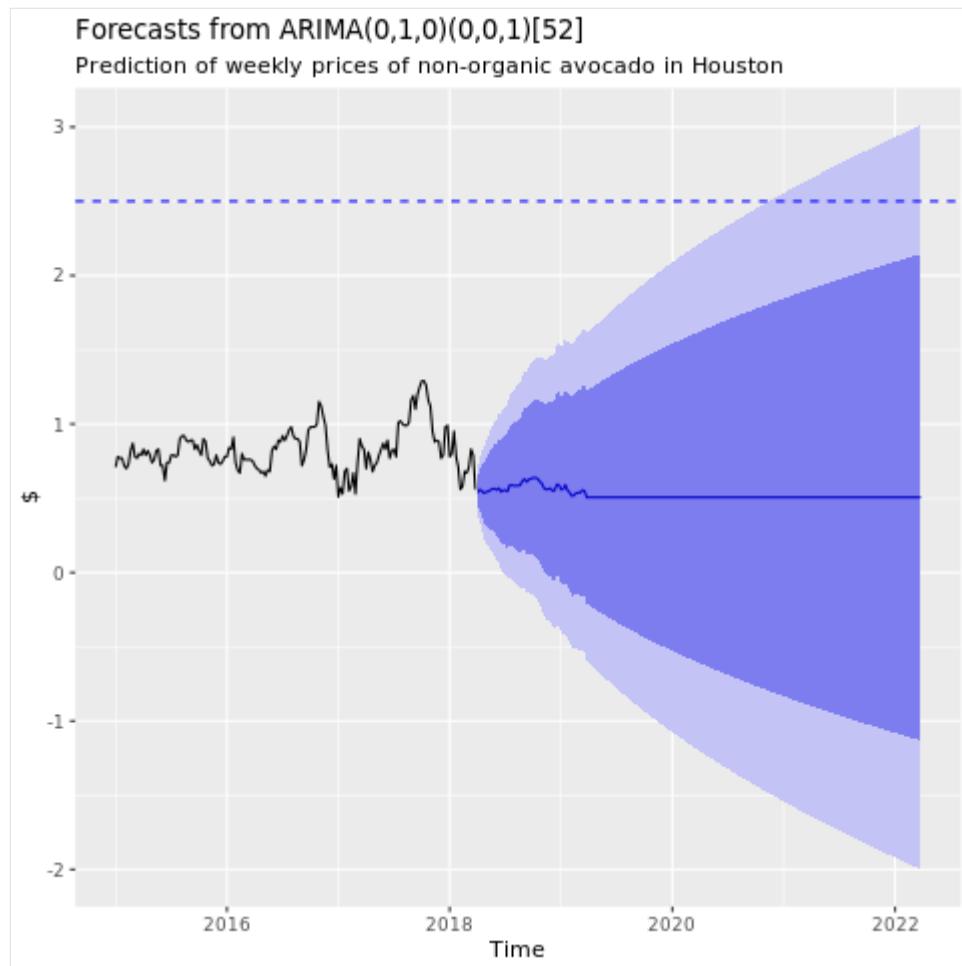


Show a graph of forecasts from the Houston model.

R

```
conv_houston_ts_arima_fc <- forecast(conv_houston_ts_arima, h = 208)

autoplot(conv_houston_ts_arima_fc) + labs(subtitle = "Prediction of weekly
prices of non-organic avocado in Houston",
y = "$") +
geom_hline(yintercept = 2.5, linetype = 2, colour = "blue")
```



## Next steps

- How to use SparkR
- How to use sparklyr
- How to use Tidyverse
- R library management
- Visualize data in R
- Tutorial: Flight delay prediction

# Flight delay prediction

Article • 05/23/2023

This article uses the [nycflights13](#) data to predict whether a plane arrives more than 30 minutes late. We then use the prediction results to build an interactive Power BI dashboard.

## ⓘ Important

Microsoft Fabric is currently in PREVIEW. This information relates to a prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, expressed or implied, with respect to the information provided here.

In this tutorial, you learn how to:

- Use [tidymodels](#) packages, such as [recipes](#), [parsnip](#), [rsample](#), [workflows](#) to process data and train a machine learning model.
- Write the output data to lakehouse as delta table.
- Build a Power BI visual report via See Through mode directly access data on your lakehouse.

## Prerequisites

- A Power BI Premium subscription. If you don't have one, see [How to purchase Power BI Premium](#).
- A Power BI workspace with assigned Premium capacity. If you don't have a workspace, use the steps in [Create a workspace](#) to create one and assign it to a Premium capacity.
- Sign in to [Microsoft Fabric](#).
- Open or create a notebook. To learn how, see [How to use Microsoft Fabric notebooks](#).
- Change the primary language by setting the **language option** to **SparkR (R)**.
- Attach your notebook to a lakehouse. On the left side, select **Add** to add an existing lakehouse or create a lakehouse.
- Attach your notebook to a lakehouse. On the left side, select **Add** to add an existing lakehouse or create a lakehouse.

# Install package

To use code in this article, install the `nycflights13` package.

```
R
```

```
install.packages("nycflights13")
```

```
R
```

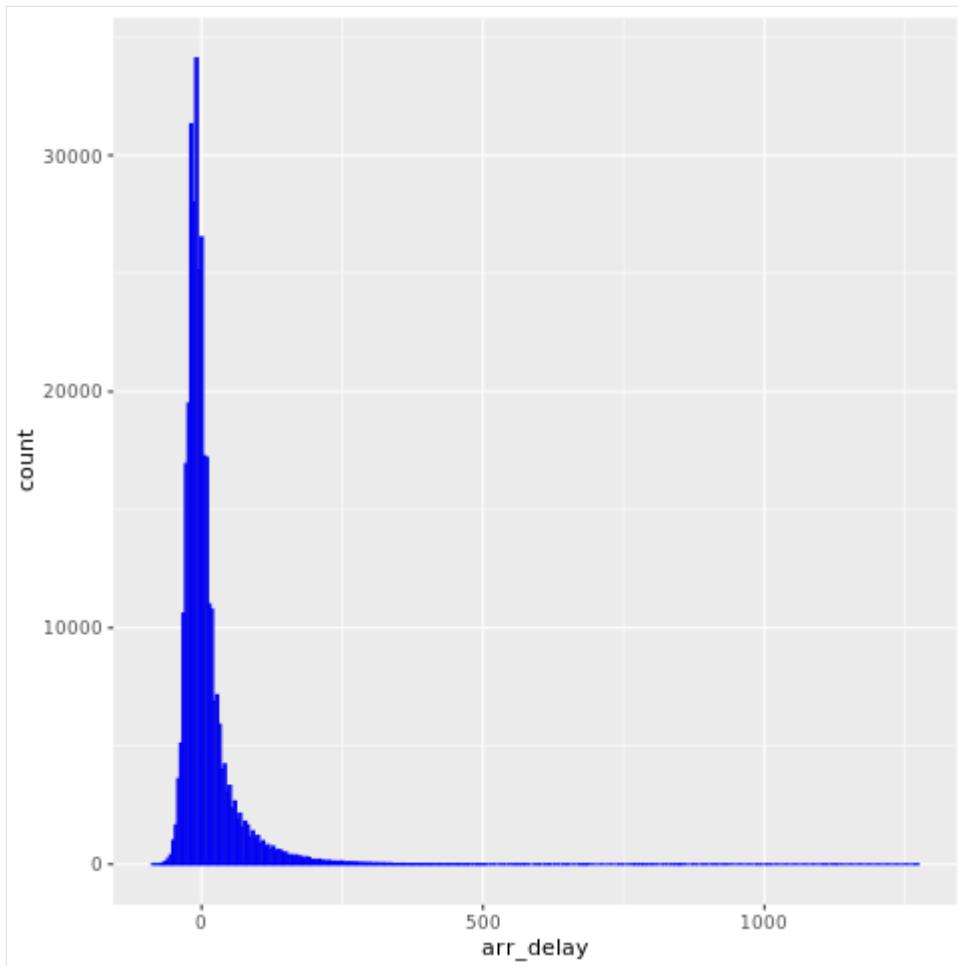
```
# load the packages
library(tidymodels)      # for tidymodels packages
library(nycflights13)     # for flight data
```

# Data exploration

The `nycflights13` data contains information on 325,819 flights departing near New York City in 2013. Let's first take a look at the flight delay distribution. The figure below shows that the distribution of the arrive delay is right skewed, it has long tail in the high values.

```
R
```

```
ggplot(flights, aes(arr_delay)) + geom_histogram(color="blue", bins = 300)
```



Load the data and make a few changes to the variables:

R

```
set.seed(123)

flight_data <-
  flights %>%
  mutate(
    # convert the arrival delay to a factor
    arr_delay = ifelse(arr_delay >= 30, "late", "on_time"),
    arr_delay = factor(arr_delay),
    # we will use the date (not date-time) in the recipe below
    date = lubridate::as_date(time_hour)
  ) %>%
  # include the weather data
  inner_join(weather, by = c("origin", "time_hour")) %>%
  # only retain the specific columns we will use
  select(dep_time, flight, origin, dest, air_time, distance,
         carrier, date, arr_delay, time_hour) %>%
  # exclude missing data
  na.omit() %>%
  # for creating models, it is better to have qualitative columns
  # encoded as factors (instead of character strings)
  mutate_if(is.character, as.factor)
```

Before you start building up your model, let's take a quick look at a few specific variables that are important for both preprocessing and modeling.

Notice that the variable called `arr_delay` is a factor variable. It's important that your outcome variable for training a logistic regression model is a factor.

```
R
```

```
glimpse(flight_data)
```

You see that about 16% of the flights in this data set arrived more than 30 minutes late.

```
R
```

```
flight_data %>%
  count(arr_delay) %>%
  mutate(prop = n/sum(n))
```

There are 104 flight destinations contained in `dest`.

```
R
```

```
unique(flight_data$dest)
```

There are 16 distinct carriers.

```
R
```

```
unique(flight_data$carrier)
```

## Data splitting

To get started, split this single dataset into two: a *training* set and a *testing* set. Keep most of the rows in the original dataset (subset chosen randomly) in the training set. The *training* data is used to fit the model, and the *testing* set is used to measure model performance.

Use the `rsample` package to create an object that contains the information on how to split the data, and then two more `rsample` functions to create data frames for the training and testing sets:

```
R
```

```
set.seed(123)
# keep most of the data into the training set
data_split <- initial_split(flight_data, prop = 0.75)

# create data frames for the two sets:
train_data <- training(data_split)
test_data <- testing(data_split)
```

## Create recipe and roles

Create a recipe for a simple logistic regression model. Before training the model, use a recipe to create a few new predictors and conduct some preprocessing required by the model.

Use the `update_role()` function to let recipes know that `flight` and `time_hour` are variables with a custom role called `ID` (a role can have any character value). The formula includes all variables in the training set other than `arr_delay` as predictors. The recipe keeps these two ID variables but doesn't use them as either outcomes or predictors.

R

```
flights_rec <-
  recipe(arr_delay ~ ., data = train_data) %>%
  update_role(flight, time_hour, new_role = "ID")
```

To view the current set of variables and roles, use the `summary()` function:

R

```
summary(flights_rec)
```

## Create features

Do some feature engineering to improve your model. Perhaps it's reasonable for the date of the flight to have an effect on the likelihood of a late arrival.

R

```
flight_data %>%
  distinct(date) %>%
  mutate(numeric_date = as.numeric(date))
```

It might be better to add model terms derived from the date that have a better potential to be important to the model. Derive the following meaningful features from the single date variable:

- Day of the week
- Month
- Whether or not the date corresponds to a holiday.

Do all three by adding steps to your recipe:

```
R

flights_rec <-
  recipe(arr_delay ~ ., data = train_data) %>%
  update_role(flight, time_hour, new_role = "ID") %>%
  step_date(date, features = c("dow", "month")) %>%
  step_holiday(date,
    holidays = timeDate::listHolidays("US"),
    keep_original_cols = FALSE) %>%
  step_dummy(all_nominal_predictors()) %>%
  step_zv(all_predictors())
```

## Fit a model with a recipe

Use logistic regression to model the flight data. Start by building model specification using the `parsnip` package:

```
R

lr_mod <-
  logistic_reg() %>%
  set_engine("glm")
```

Then use the `workflows` package to bundle your `parsnip` model (`lr_mod`) with your recipe (`flights_rec`).

```
R

flights_wf <-
  workflow() %>%
  add_model(lr_mod) %>%
  add_recipe(flights_rec)

flights_wf
```

# Train the model

Here's a single function that can be used to prepare the recipe and train the model from the resulting predictors:

```
R  
  
flights_fit <-  
  flights_wflow %>%  
  fit(data = train_data)
```

Use the helper functions `xtract_fit_parsnip()` and `extract_recipe()` to extract the model or recipe objects from the workflow. For example, here you pull the fitted model object then use the `broom::tidy()` function to get a tidy tibble of model coefficients:

```
R  
  
flights_fit %>%  
  extract_fit_parsnip() %>%  
  tidy()
```

# Predict results

Now use the trained workflow (`flights_fit`) to predict with the unseen test data, which you do with a single call to `predict()`. The `predict()` method applies the recipe to the new data, then passes them to the fitted model.

```
R  
  
predict(flights_fit, test_data)
```

Now get the output from `predict()` return the predicted class: `late` versus `on_time`. If you want the predicted class probabilities for each flight instead, use `augment()` with the model plus test data to save them together:

```
R  
  
flights_aug <-  
  augment(flights_fit, test_data)
```

The data looks like:

```
R
```

```
glimpse(flights_aug)
```

## Evaluate the model

Now you have a tibble with your predicted class probabilities. From these first few rows, you see that your model predicted 5 on time flights correctly (values of `.pred_on_time` are  $p > 0.50$ ). But you also know that we have 81,455 rows total to predict.

You want a metric that tells how well your model predicted late arrivals, compared to the true status of your outcome variable, `arr_delay`.

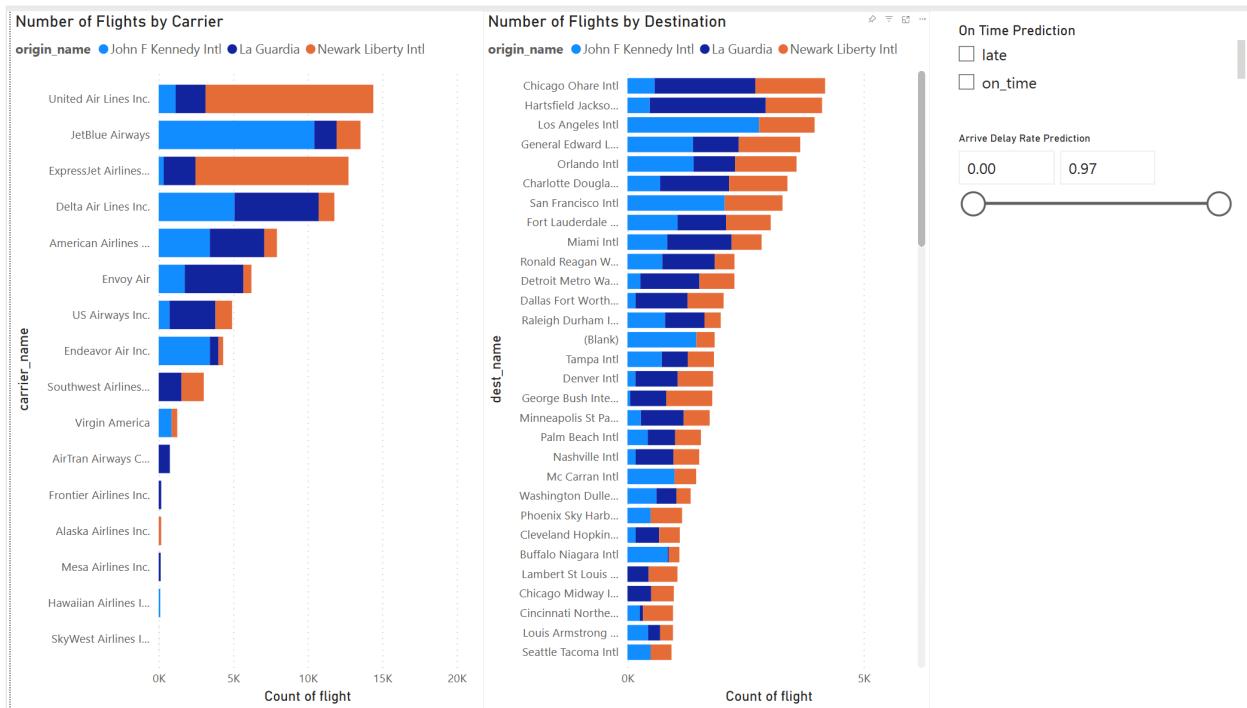
Use the area under the ROC curve as our metric, computed using `roc_curve()` and `roc_auc()` from the `yardstick` package.

```
R
```

```
flights_aug %>%  
  roc_curve(truth = arr_delay, .pred_late) %>%  
  autoplot()
```

## Build a Power BI report

The model result isn't too bad! Use the flight delay prediction results to build an interactive Power BI dashboard, showing the number of flights by carrier and number of flights by destination. The dashboard is also able to filter by the delay prediction results.



First include the carrier name and airport name to the prediction result dataset.

R

```
flights_clean <- flights_aug %>%
  # include the airline data
  left_join(airlines, c("carrier"="carrier"))%>%
  rename("carrier_name"="name") %>%
  # include the airports data for origin
  left_join(airports, c("origin"="faa")) %>%
  rename("origin_name"="name") %>%
  # include the airports data for destination
  left_join(airports, c("dest"="faa")) %>%
  rename("dest_name"="name") %>%
  # only retain the specific columns we will use
  select(flight, origin, origin_name, dest, dest_name, air_time, distance,
         carrier, carrier_name, date, arr_delay, time_hour, .pred_class, .pred_late,
         .pred_on_time)
```

The data looks like:

R

```
glimpse(flights_clean)
```

Convert the data to Spark dataframe:

R

```
sparkdf <- as.DataFrame(flights_clean)
display(sparkdf)
```

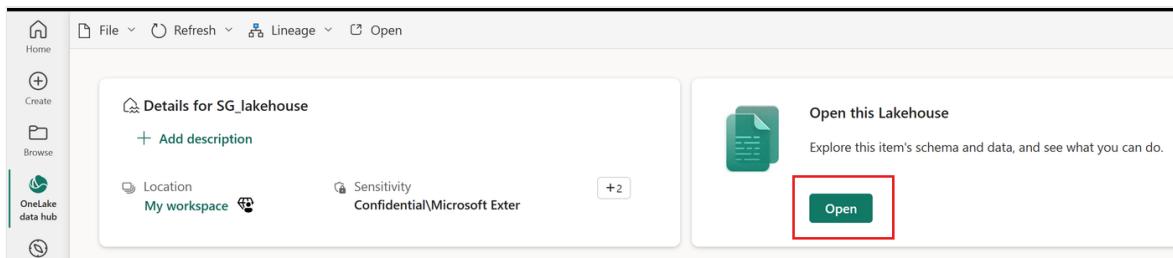
Write the data into a delta table on your lakehouse:

R

```
# write data into delta table
temp_delta<- "Tables/nycflight13"
write.df(sparkdf, temp_delta ,source="delta", mode = "overwrite", header =
"true")
```

You can now use this table to create a Power BI dataset.

1. On the left, select **OneLake data hub**.
2. Select the Lakehouse you attached to your notebook.
3. On the top right, select **Open**.



4. On the top, select **New Power BI dataset**.
5. Select **nycflight13** for your new dataset, then select **Confirm**.
6. Your Power BI dataset is created. At the top, select **New report**.
7. Select or drag fields from the data and visualizations panes onto the report canvas to build your report.

The screenshot shows the Power BI interface with the 'Visualizations' pane open. The pane includes sections for 'Filters', 'Build visual', 'Values', and 'Data'. The 'Data' section lists fields such as \_pred\_class, \_pred\_late, \_pred\_on\_time, air\_time, arr\_delay, carrier, carrier\_name, date, dest, dest\_name, distance, flight, origin, origin\_name, and time\_hour.

To create the report shown at the beginning of this section, use the following visualizations and data:

1. Stacked barchart with:
  - a. Y-axis: **carrier\_name**.
  - b. X-axis: **flight**. Select **Count** for the aggregation.
  - c. Legend: **origin\_name**
2. Stacked barchart with:
  - a. Y-axis: **dest\_name**.
  - b. X-axis: **flight**. Select **Count** for the aggregation.
  - c. Legend: **origin\_name**.
3. Slicer with:
  - a. Field: **\_pred\_class**
4. Slicer with:
  - a. Field: **\_pred\_late**

## Next steps

- [How to use SparkR](#)

- How to use sparklyr
- How to use Tidyverse
- R library management
- Visualize data in R
- Tutorial: Avocado price prediction

# R library management

Article • 05/23/2023

Libraries provide reusable code that you might want to include in your programs or projects for Microsoft Fabric Spark.

## Important

Microsoft Fabric is currently in PREVIEW. This information relates to a prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, expressed or implied, with respect to the information provided here.

Microsoft Fabric supports an R runtime with many popular open-source R packages, including TidyVerse, preinstalled. When a Spark instance starts, these libraries are included automatically and available to be used immediately in notebooks or Spark job definitions.

You might need to update your R libraries for various reasons. For example, one of your core dependencies released a new version, or your team has built a custom package that you need available in your Spark clusters.

There are two types of libraries you may want to include based on your scenario:

- **Feed libraries** refer to the ones residing in public sources or repositories, such as [CRAN](#) or GitHub.
- **Custom libraries** are the code built by you or your organization, .tar.gz can be managed through Library Management portals.

There are two levels of packages installed on Microsoft Fabric:

- **Workspace**: Workspace-level installation defines the working environment for the entire workspace. The libraries installed at the workspace level are available for all Notebooks and SJDs under this workspace. Update the workspace libraries when you want to set up the shared environment for all items in a workspace.
- **Session** : A session-level installation creates an environment for a specific notebook session. The change of session-level libraries isn't persisted between sessions.

Summarizing the current available R library management behaviors:

Library Type	Workspace-level installation	Session-level installation
R Feed (CRAN)	Not Supported	Supported
R Custom	Supported	Supported

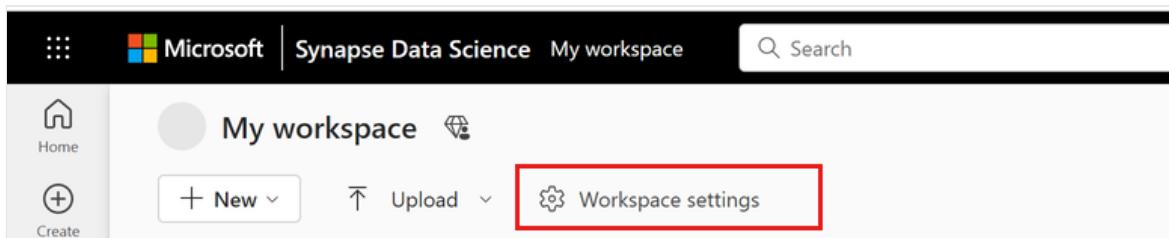
## Prerequisites

- A Power BI Premium subscription. If you don't have one, see [How to purchase Power BI Premium](#).
- A Power BI workspace with assigned Premium capacity. If you don't have a workspace, use the steps in [Create a workspace](#) to create one and assign it to a Premium capacity.
- Sign in to [Microsoft Fabric](#).

## Workspace-level R library management

Manage your custom libraries at the workspace-level in workspace settings. Currently R supports only custom libraries in the workspace settings.

1. Select your workspace.
2. Select **Workspace settings** at the top of the page.



3. Select **Data Engineering/Science** → **Library management**.
4. Select the tab for **Custom libraries**.

### ⓘ Note

- Only the workspace admin has permission to update the workspace level settings.
- Managing R feed libraries in workspace settings is currently not supported.

Install and manage your custom R packages, that is, `.tar.gz` in the **Custom libraries** section.

- **Upload:** Select the **Upload** button and select your package from your local directory. Then select **Apply** to add the package to the workspace. The library management module helps you handle potential conflicts and required dependencies in your custom libraries.
- **Remove:** If a custom library is no longer useful for your Spark applications, use the trash button to remove it.
- **Review and apply changes:** When changes are pending, go to the **Pending changes** panel to review them or cancel a change.

## Session-level R libraries

When doing interactive data analysis or machine learning, you might try newer packages or you might need packages that are currently unavailable on your workspace. Instead of updating the workspace settings, you can use session-scoped packages to add, manage, and update session dependencies.

- When you install session-scoped libraries, only the current notebook has access to the specified libraries.
- These libraries don't impact other sessions or jobs using the same Spark pool.
- These libraries are installed on top of the base runtime and pool level libraries.
- Notebook libraries take the highest precedence.
- Session-scoped R libraries don't persist across sessions. These libraries are installed at the start of each session when the related installation commands are executed.
- Session-scoped R libraries are automatically installed across both the driver and worker nodes.

### ⓘ Note

The commands of managing R libraries are disabled when running pipeline jobs. If you want to install a package within a pipeline, you must use the library management capabilities at the workspace level.

## Install R packages from CRAN

You can easily install an R library from [CRAN ↗](#).

R

```
# install a package from CRAN
install.packages(c("nycflights13", "Lahman"))
```

You can also use CRAN snapshots as the repository to ensure to download the same package version each time.

R

```
# install a package from CRAN snapshot
install.packages("highcharter", repos =
  "https://cran.microsoft.com/snapshot/2021-07-16/")
```

## Install R packages using devtools

The `devtools` library simplifies package development to expedite common tasks. This library is installed within the default Microsoft Fabric runtime.

You can use `devtools` to specify a specific version of a library to install. These libraries are installed across all nodes within the cluster.

R

```
# Install a specific version.
install_version("caesar", version = "1.0.0")
```

Similarly, you can install a library directly from GitHub.

R

```
# Install a GitHub library.

install_github("jtilly/matchingR")
```

Currently, the following `devtools` functions are supported within Microsoft Fabric:

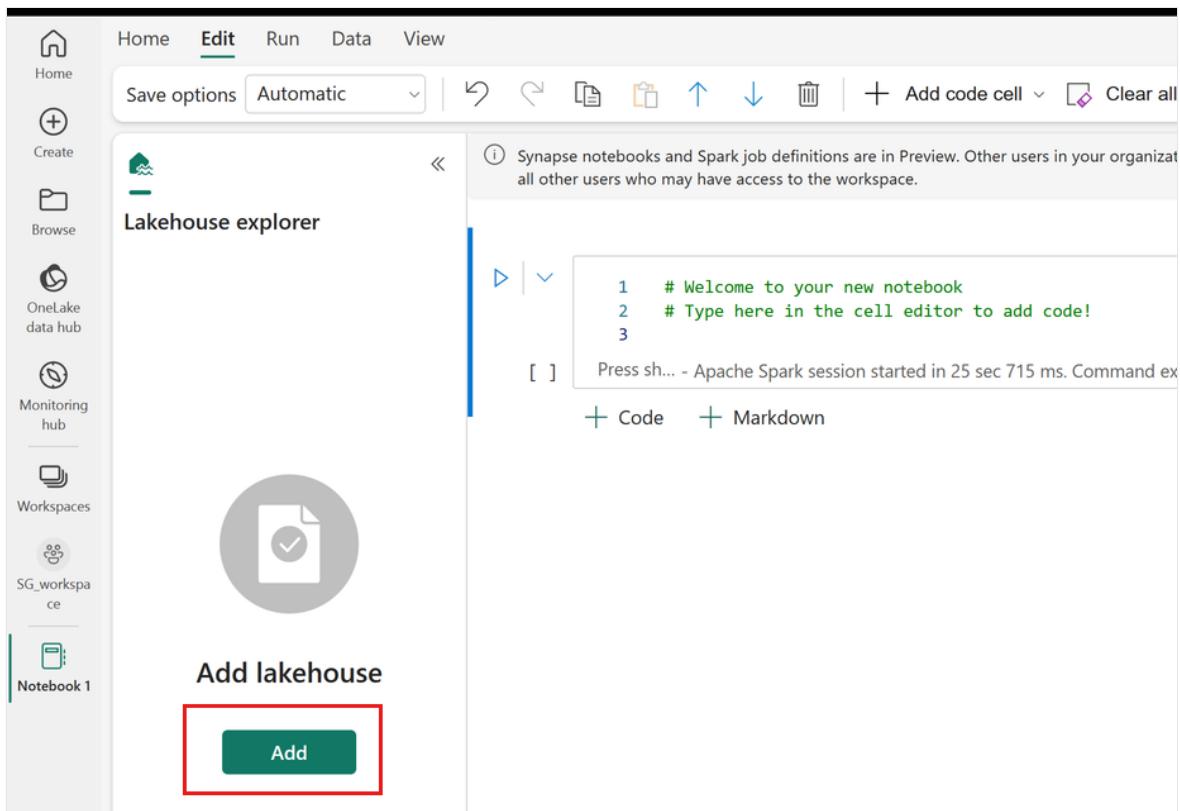
Command	Description
install_github()	Installs an R package from GitHub
install_gitlab()	Installs an R package from GitLab
install_bitbucket()	Installs an R package from BitBucket

Command	Description
install_url()	Installs an R package from an arbitrary URL
install_git()	Installs from an arbitrary git repository
install_local()	Installs from a local file on disk
install_version()	Installs from a specific version on CRAN

## Install R custom libraries

To use a session-level custom library, you must first upload it to an attached Lakehouse.

1. On the left side, select **Add** to add an existing lakehouse or create a lakehouse.



2. To add files to this lakehouse, select your workspace and then select the lakehouse.

The screenshot shows the Power BI workspace interface. On the left, there's a sidebar with icons for Home, Create, Browse, OneLake data hub, Monitoring hub, Workspaces, and Notebook 1. The 'Workspaces' section has a red box around it, and the 'SG\_workspace' icon is also highlighted with a red box. The main content area is titled 'SG\_workspace' and shows a table with one item: 'SG\_lakehouse' (Type: Lakehouse). There are buttons for '+ New', 'Upload', and '...' at the top, and a search bar and filter button below.

3. Right click or select the "..." next to **Files** to upload your *.tar.gz* file.

The screenshot shows the Power BI workspace interface with the 'Home' tab selected. The sidebar is identical to the previous screenshot. In the main area, the 'Explorer' pane shows a tree view with 'SG\_lakehouse' expanded, showing 'Tables' and 'Files'. A red box highlights the 'Files' folder. A context menu is open over the 'Files' folder, with options: 'New shortcut', 'New subfolder', 'Upload' (with 'Upload files' highlighted with a red box), 'Properties', and 'Refresh'. A message at the top says: 'A SQL endpoint for SQL querying and a default dataset for reporting were created and will be updated with the latest data from the source.'

4. After uploading, go back to your notebook. Use the following command to install the custom library to your session:

R

```
install.packages("filepath/filename.tar.gz", repos = NULL, type =  
"source")
```

## View installed libraries

Query all the libraries installed within your session using the `library` command.

R

```
# query all the libraries installed in current session  
library()
```

Use the `packageVersion` function to check the version of the library:

R

```
# check the package version  
packageVersion("caesar")
```

## Remove an R package from a session

You can use the `detach` function to remove a library from the namespace. These libraries stay on disk until they're loaded again.

R

```
# detach a library  
  
detach("package: caesar")
```

To remove a session-scoped package from a notebook, use the `remove.packages()` command. This library change has no impact on other sessions on the same cluster. Users can't uninstall or remove built-in libraries of the default Microsoft Fabric runtime.

### ⓘ Note

You can't remove core packages like SparkR, SparklyR, or R.

R

```
remove.packages("caesar")
```

## Session-scoped R libraries and SparkR

Notebook-scoped libraries are available on SparkR workers.

```
R
```

```
install.packages("stringr")
library(SparkR)

str_length_function <- function(x) {
  library(stringr)
  str_length(x)
}

docs <- c("Wow, I really like the new light sabers!",
         "That book was excellent.",
         "R is a fantastic language.",
         "The service in this restaurant was miserable.",
         "This is neither positive or negative.")

spark.lapply(docs, str_length_function)
```

## Session-scoped R libraries and sparklyr

With `spark_apply()` in sparklyr, you can use any R packages inside Spark. By default, in `sparklyr::spark_apply()`, the `packages` argument sets to FALSE. This copies libraries in the current libPaths to the workers, allowing you to import and use them on workers. For example, you can run the following to generate a caesar-encrypted message with `sparklyr::spark_apply()`:

```
R
```

```
install.packages("caesar", repos =
"https://cran.microsoft.com/snapshot/2021-07-16/")

spark_version <- sparkR.version()
config <- spark_config()
sc <- spark_connect(master = "yarn", version = spark_version, spark_home =
"/opt/spark", config = config)

apply_cases <- function(x) {
  library(caesar)
  caesar("hello world")
}
```

```
sdf_len(sc, 5) %>%  
  spark_apply(apply_cases, packages=FALSE)
```

## Next steps

Learn more about the R functionalities:

- [How to use SparkR](#)
- [How to use sparklyr](#)
- [How to use Tidyverse](#)
- [Create R visualization](#)

# Use SparkR

Article • 05/23/2023

## ⓘ Important

Microsoft Fabric is currently in PREVIEW. This information relates to a prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, expressed or implied, with respect to the information provided here.

[SparkR](#) is an R package that provides a light-weight frontend to use Apache Spark from R. SparkR provides a distributed data frame implementation that supports operations like selection, filtering, aggregation etc. SparkR also supports distributed machine learning using MLlib.

## ⓘ Important

Microsoft Fabric is currently in PREVIEW. This information relates to a prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, expressed or implied, with respect to the information provided here.

Use SparkR through Spark batch job definitions or with interactive Microsoft Fabric notebooks.

R support is only available in Spark3.1 or above. R in Spark 2.4 is not supported.

## Prerequisites

- A Power BI Premium subscription. If you don't have one, see [How to purchase Power BI Premium](#).
- A Power BI workspace with assigned Premium capacity. If you don't have a workspace, use the steps in [Create a workspace](#) to create one and assign it to a Premium capacity.
- Sign in to [Microsoft Fabric](#).
- Open or create a notebook. To learn how, see [How to use Microsoft Fabric notebooks](#).
- Change the primary language by setting the **language option** to **SparkR (R)**.

- Attach your notebook to a lakehouse. On the left side, select **Add** to add an existing lakehouse or create a lakehouse.

## Read and write SparkR DataFrames

### Read a SparkR DataFrame from a local R data.frame

The simplest way to create a DataFrame is to convert a local R data.frame into a Spark DataFrame.

R

```
# load SparkR pacakge
library(SparkR)

# read a SparkR DataFrame from a local R data.frame
df <- createDataFrame(faithful)

# displays the content of the DataFrame
display(df)
```

### Read and write SparkR DataFrame from Lakehouse

Data can be stored on the local filesystem of cluster nodes. The general methods to read and write a SparkR DataFrame from Lakehouse is `read.df` and `write.df`. These methods take the path for the file to load and the type of data source. SparkR supports reading CSV, JSON, text, and Parquet files natively.

To read and write to a Lakehouse, first add it to your session. On the left side of the notebook, select **Add** to add an existing Lakehouse or create a Lakehouse.

#### ⓘ Note

To access Lakehouse files using Spark packages, such as `read.df` or `write.df`, use its *ADFS path* or *relative path for Spark*. In the Lakehouse explorer, right click on the files or folder you want to access and copy its *ADFS path* or *relative path for Spark* from the contextual menu.

R

```
# write data in CSV using relative path for Spark
temp_csv_spark<-"Files/data/faithful.csv"
```

```
write.df(df, temp_csv_spark, source="csv", mode = "overwrite", header = "true")

# read data in CSV using relative path for Spark
faithfulDF_csv <- read.df(temp_csv_spark, source= "csv", header = "true",
inferSchema = "true")

# displays the content of the DataFrame
display(faithfulDF_csv)
```

R

```
# write data in parquet using ADFS path
temp_parquet_spark<- "abfss://xxx/xxx/data/faithful.parquet"
write.df(df, temp_parquet_spark, source="parquet", mode = "overwrite",
header = "true")

# read data in parquet using ADFS path
faithfulDF_pq <- read.df(temp_parquet_spark, source= "parquet", header =
"true", inferSchema = "true")

# displays the content of the DataFrame
display(faithfulDF_pq)
```

Microsoft Fabric has `tidyverse` preinstalled. You can access Lakehouse files in your familiar R packages, such as reading and writing Lakehouse files using `readr::read_csv()` and `readr::write_csv()`.

### ⓘ Note

To access Lakehouse files using R packages, you need to use the *File API path*. In the Lakehouse explorer, right click on the file or folder that you want to access and copy its *File API path* from the contextual menu.

R

```
# read data in CSV using API path
# To find the path, navigate to the csv file, right click, and Copy File
# API path.
temp_csv_api<- '/lakehouse/default/Files/data/faithful.csv/part-00000-
d8e09a34-bd63-41bd-8cf8-f4ed2ef90e6c-c000.csv'
faithfulDF_API <- readr::read_csv(temp_csv_api)

# display the content of the R data.frame
head(faithfulDF_API)
```

You can also read a SparkR Dataframe on your Lakehouse using SparkSQL queries.

R

```
# Register earlier df as temp view
createOrReplaceTempView(df, "eruptions")

# Create a df using a SparkSQL query
waiting <- sql("SELECT * FROM eruptions")

head(waiting)
```

## Read and write SQL tables through RODBC

Use RODBC to connect to SQL based databases through an ODBC interface. For example, you can connect to a Synapse dedicated SQL pool as shown in the following example code. Substitute your own connection details for `<database>`, `<uid>`, `<password>`, and `<table>`.

R

```
# load RODBC package
library(RODBC)

# config connection string

DriverVersion <- substr(system("apt list --installed *msodbc*", intern=TRUE,
ignore.stderr=TRUE)[2],10,11)
ServerName <- "your-server-name"
DatabaseName <- "your-database-name"
Uid <- "your-user-id-list"
Password <- "your-password"

ConnectionString = sprintf("Driver={ODBC Driver %s for SQL Server};
Server=%s;
Database=%s;
Uid=%s;
Pwd=%s;
Encrypt=yes;
TrustServerCertificate=yes;
Connection Timeout=30;",DriverVersion,ServerName,DatabaseName,Uid,Password)
print(ConnectionString)

# connect to driver
channel <- odbcDriverConnect(ConnectionString)

# query from existing tables
Rdf <- sqlQuery(channel, "select * from <table>")
class(Rdf)
```

```
# use SparkR::as.DataFrame to convert R date.frame to SparkR DataFrame.  
spark_df <- as.DataFrame(Rdf)  
class(spark_df)  
head(spark_df)
```

## DataFrame operations

SparkR DataFrames support many functions to do structured data processing. Here are some basic examples. A complete list can be found in the [SparkR API docs](#).

### Select rows and columns

R

```
# Select only the "waiting" column  
head(select(df, df$waiting))
```

R

```
# Pass in column name as strings  
head(select(df, "waiting"))
```

R

```
# Filter to only retain rows with waiting times longer than 70 mins  
head(filter(df, df$waiting > 70))
```

### Grouping and aggregation

SparkR data frames support many commonly used functions to aggregate data after grouping. For example, we can compute a histogram of the waiting time in the faithful dataset as shown below

R

```
# we use the `n` operator to count the number of times each waiting time  
appears  
head(summarize(groupBy(df, df$waiting), count = n(df$waiting)))
```

R

```
# we can also sort the output from the aggregation to get the most common
# waiting times
waiting_counts <- summarize(groupBy(df, df$waiting), count = n(df$waiting))
head(arrange(waiting_counts, desc(waiting_counts$count)))
```

## Column operations

SparkR provides many functions that can be directly applied to columns for data processing and aggregation. The following example shows the use of basic arithmetic functions.

R

```
# convert waiting time from hours to seconds.
# you can assign this to a new column in the same DataFrame
df$waiting_secs <- df$waiting * 60
head(df)
```

## Apply user-defined function

SparkR supports several kinds of user-defined functions:

### Run a function on a large dataset with `dapply` or `dapplyCollect`

#### `dapply`

Apply a function to each partition of a `SparkDataFrame`. The function to be applied to each partition of the `SparkDataFrame` and should have only one parameter, to which a `data.frame` corresponds to each partition will be passed. The output of function should be a `data.frame`. Schema specifies the row format of the resulting a `SparkDataFrame`. It must match to [data types](#) of returned value.

R

```
# convert waiting time from hours to seconds
df <- createDataFrame(faithful)
schema <- structType(structField("eruptions", "double"),
structField("waiting", "double"),
structField("waiting_secs", "double"))

# apply UDF to DataFrame
df1 <- dapply(df, function(x) { x <- cbind(x, x$waiting * 60) }, schema)
head(collect(df1))
```

## dapplyCollect

Like dapply, apply a function to each partition of a `SparkDataFrame` and collect the result back. The output of the function should be a `data.frame`. But, this time, schema isn't required to be passed. Note that `dapplyCollect` can fail if the outputs of the function run on all the partition can't be pulled to the driver and fit in driver memory.

R

```
# convert waiting time from hours to seconds
# apply UDF to DataFrame and return a R's data.frame
ldf <- dapplyCollect(
  df,
  function(x) {
    x <- cbind(x, "waiting_secs" = x$waiting * 60)
  })
head(ldf, 3)
```

## Run a function on a large dataset grouping by input column(s) with gapply or gapplyCollect

### gapply

Apply a function to each group of a `SparkDataFrame`. The function is to be applied to each group of the `SparkDataFrame` and should have only two parameters: grouping key and R `data.frame` corresponding to that key. The groups are chosen from `SparkDataFrames` column(s). The output of the function should be a `data.frame`. Schema specifies the row format of the resulting `SparkDataFrame`. It must represent R function's output schema from Spark [data types](#). The column names of the returned `data.frame` are set by user.

R

```
# determine six waiting times with the largest eruption time in minutes.
schema <- structType(structField("waiting", "double"),
structField("max_eruption", "double"))
result <- gapply(
  df,
  "waiting",
  function(key, x) {
    y <- data.frame(key, max(x$eruptions))
  },
  
```

```
    schema)
head(collect(arrange(result, "max_eruption", decreasing = TRUE)))
```

## gapplyCollect

Like `gapply`, applies a function to each group of a `SparkDataFrame` and collect the result back to R `data.frame`. The output of the function should be a `data.frame`. But, the schema isn't required to be passed. Note that `gapplyCollect` can fail if the outputs of the function run on all the partition can't be pulled to the driver and fit in driver memory.

R

```
# determine six waiting times with the largest eruption time in minutes.
result <- gapplyCollect(
  df,
  "waiting",
  function(key, x) {
    y <- data.frame(key, max(x$eruptions))
    colnames(y) <- c("waiting", "max_eruption")
    y
  })
head(result[order(result$max_eruption, decreasing = TRUE), ])
```

## Run local R functions distributed with spark.lapply

### spark.lapply

Similar to `lapply` in native R, `spark.lapply` runs a function over a list of elements and distributes the computations with Spark. Applies a function in a manner that is similar to `doParallel` or `lapply` to elements of a list. The results of all the computations should fit in a single machine. If that is not the case, they can do something like `df <- createDataFrame(list)` and then use `dapply`.

R

```
# perform distributed training of multiple models with spark.lapply. Here,
# we pass
# a read-only list of arguments which specifies family the generalized
# linear model should be.
families <- c("gaussian", "poisson")
train <- function(family) {
  model <- glm(Sepal.Length ~ Sepal.Width + Species, iris, family = family)
  summary(model)
```

```
}

# return a list of model's summaries
model.summaries <- spark.lapply(families, train)

# print the summary of each model
print(model.summaries)
```

## Run SQL queries from SparkR

A SparkR DataFrame can also be registered as a temporary view that allows you to run SQL queries over its data. The `sql` function enables applications to run SQL queries programmatically and returns the result as a SparkR DataFrame.

R

```
# Register earlier df as temp view
createOrReplaceTempView(df, "eruptions")

# Create a df using a SparkSQL query
waiting <- sql("SELECT waiting FROM eruptions where waiting>70 ")

head(waiting)
```

## Machine learning

SparkR exposes most of MLlib algorithms. Under the hood, SparkR uses MLlib to train the model.

The following example shows how to build a Gaussian GLM model using SparkR. To run linear regression, set family to `"gaussian"`. To run logistic regression, set family to `"binomial"`. When using SparkML `GLM` SparkR automatically performs one-hot encoding of categorical features so that it doesn't need to be done manually. Beyond String and Double type features, it's also possible to fit over MLlib Vector features, for compatibility with other MLlib components.

To learn more about which machine learning algorithms are supported, you can visit the documentation for [SparkR](#) and [MLlib](#).

R

```
# create the DataFrame
cars <- cbind(model = rownames(mtcars), mtcars)
carsDF <- createDataFrame(cars)

# fit a linear model over the dataset.
```

```
model <- spark.glm(carsDF, mpg ~ wt + cyl, family = "gaussian")  
  
# model coefficients are returned in a similar format to R's native glm().  
summary(model)
```

## Next steps

- How to use sparklyr
- How to use Tidyverse
- R library management
- Create R visualization
- Tutorial: avocado price prediction
- Tutorial: flight delay prediction

# Use sparklyr

Article • 05/23/2023

[sparklyr](#) is an R interface to Apache Spark. It provides a mechanism to interact with Spark using familiar R interfaces. You can use sparklyr through Spark batch job definitions or with interactive Microsoft Fabric notebooks.

## ⓘ Important

Microsoft Fabric is currently in PREVIEW. This information relates to a prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, expressed or implied, with respect to the information provided here.

`sparklyr` is used along with other [tidyverse](#) packages such as [dplyr](#). Microsoft Fabric distributes the latest stable version of sparklyr and tidyverse with every runtime release. You can import them and start using the API.

## Prerequisites

- A Power BI Premium subscription. If you don't have one, see [How to purchase Power BI Premium](#).
- A Power BI workspace with assigned Premium capacity. If you don't have a workspace, use the steps in [Create a workspace](#) to create one and assign it to a Premium capacity.
- Sign in to [Microsoft Fabric](#).
- Open or create a notebook. To learn how, see [How to use Microsoft Fabric notebooks](#).
- Change the primary language by setting the `language` option to `SparkR (R)`.
- Attach your notebook to a lakehouse. On the left side, select **Add** to add an existing lakehouse or create a lakehouse.

## Connect sparklyr to Synapse Spark cluster

Use the following connection method in `spark_connect()` to establish a `sparklyr` connection.

R

```
# connect sparklyr to your spark cluster
spark_version <- sparkR.version()
config <- spark_config()
sc <- spark_connect(master = "yarn", version = spark_version, spark_home =
"/opt/spark", config = config)
```

## Use sparklyr to read data

A new Spark session contains no data. The first step is to either load data into your Spark session's memory, or point Spark to the location of the data so it can access the data on-demand.

R

```
# load the sparklyr package
library(sparklyr)

# copy data from R environment to the Spark session's memory
mtcars_tbl <- copy_to(sc, mtcars, "spark_mtcars", overwrite = TRUE)

head(mtcars_tbl)
```

Using `sparklyr`, you can also `write` and `read` data from a Lakehouse file using ABFS path. To read and write to a Lakehouse, first add it to your session. On the left side of the notebook, select **Add** to add an existing Lakehouse or create a Lakehouse.

To find your ABFS path, right click on the **Files** folder in your Lakehouse, then select **Copy ABFS path**. Paste your path to replace

`abfss://xxxx@onelake.dfs.fabric.microsoft.com/xxxx/Files` in this code:

R

```
temp_csv =
"abfss://xxxx@onelake.dfs.fabric.microsoft.com/xxxx/Files/data/mtcars.csv"

# write the table to your lakehouse using the ABFS path
spark_write_csv(mtcars_tbl, temp_csv, header = TRUE, mode = 'overwrite')

# read the data as CSV from lakehouse using the ABFS path
mtcarsDF <- spark_read_csv(sc, temp_csv)
head(mtcarsDF)
```

# Use sparklyr to manipulate data

`sparklyr` provides multiple methods to process data inside Spark using:

- `dplyr` commands
- SparkSQL
- Spark's feature transformers

## Use `dplyr`

You can use familiar `dplyr` commands to prepare data inside Spark. The commands run inside Spark, so there are no unnecessary data transfers between R and Spark.

Click the [Manipulating Data with dplyr](#) to see extra documentation on using `dplyr` with Spark.

R

```
# count cars by the number of cylinders the engine contains (cyl), order the
# results descendingly
library(dplyr)

cargroup <- group_by(mtcars_tbl, cyl) %>%
  count() %>%
  arrange(desc(n))

cargroup
```

`sparklyr` and `dplyr` translate the R commands into Spark SQL for us. To see the resulting query use `show_query()`:

R

```
# show the dplyr commands that are to run against the Spark connection
dplyr::show_query(cargroup)
```

## Use SQL

It's also possible to execute SQL queries directly against tables within a Spark cluster. The `spark_connection()` object implements a [DBI](#) interface for Spark, so you can use `dbGetQuery()` to execute SQL and return the result as an R data frame:

R

```
library(DBI)
dbGetQuery(sc, "select cyl, count(*) as n from spark_mtcars
GROUP BY cyl
ORDER BY n DESC")
```

## Use Feature Transformers

Both of the previous methods rely on SQL statements. Spark provides commands that make some data transformation more convenient, and without the use of SQL.

For example, the `ft_binarizer()` command simplifies the creation of a new column that indicates if the value of another column is above a certain threshold.

You can find the full list of the Spark Feature Transformers available through `sparklyr` from [Reference -FT ↗](#).

R

```
mtcars_tbl %>%
  ft_binarizer("mpg", "over_20", threshold = 20) %>%
  select(mpg, over_20) %>%
  head(5)
```

## Machine learning

Here's an example where we use `ml_linear_regression()` to fit a linear regression model. We use the built-in `mtcars` dataset, and see if we can predict a car's fuel consumption (`mpg`) based on its weight (`wt`), and the number of cylinders the engine contains (`cyl`). We assume in each case that the relationship between `mpg` and each of our features is linear.

## Generate testing and training data sets

Use a simple split, 70% for training and 30% for testing the model. Playing with this ratio may result in different models.

R

```
# split the dataframe into test and training dataframes

partitions <- mtcars_tbl %>%
```

```
select(mpg, wt, cyl) %>%  
sdf_random_split(training = 0.7, test = 0.3, seed = 2023)
```

## Train the model

Train the Logistic Regression model.

R

```
fit <- partitions$training %>%  
ml_linear_regression(mpg ~ .)  
  
fit
```

Now use `summary()` to learn a bit more about the quality of our model, and the statistical significance of each of our predictors.

R

```
summary(fit)
```

## Use the model

You can apply the model on the testing dataset by calling `ml_predict()`.

R

```
pred <- ml_predict(fit, partitions$test)  
  
head(pred)
```

For a list of Spark ML models available through sparklyr visit [Reference - ML ↗](#)

## Disconnect from Spark cluster

You can call `spark_disconnect()` to or select the **Stop session** button on top of the notebook ribbon end your Spark session.

R

```
spark_disconnect(sc)
```

# Next steps

Learn more about the R functionalities:

- [How to use SparkR](#)
- [How to use Tidyverse](#)
- [R library management](#)
- [Create R visualization](#)
- [Tutorial: avocado price prediction](#)
- [Tutorial: flight delay prediction](#)

# Use Tidyverse

Article • 05/23/2023

Tidyverse [↗](#) is a collection of R packages that data scientists commonly use in everyday data analyses. It includes packages for data import (`readr`), data visualization (`ggplot2`), data manipulation (`dplyr`, `tidyr`), functional programming (`purrr`), and model building (`tidymodels`) etc. The packages in `tidyverse` are designed to work together seamlessly and follow a consistent set of design principles.

## ⓘ Important

Microsoft Fabric is currently in PREVIEW. This information relates to a prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, expressed or implied, with respect to the information provided here.

Microsoft Fabric distributes the latest stable version of `tidyverse` with every runtime release. Import and start using your familiar R packages.

## Prerequisites

- A Power BI Premium subscription. If you don't have one, see [How to purchase Power BI Premium](#).
- A Power BI workspace with assigned Premium capacity. If you don't have a workspace, use the steps in [Create a workspace](#) to create one and assign it to a Premium capacity.
- Sign in to [Microsoft Fabric](#) ↗.
- Open or create a notebook. To learn how, see [How to use Microsoft Fabric notebooks](#).
- Change the primary language by setting the `language` option to `SparkR (R)`.
- Attach your notebook to a lakehouse. On the left side, select **Add** to add an existing lakehouse or create a lakehouse.

## Load `tidyverse`

R

```
# load tidyverse  
library(tidyverse)
```

## Data import

`readr` is an R package that provides tools for reading rectangular data files such as CSV, TSV, and fixed-width files. `readr` provides a fast and friendly way to read rectangular data files such as providing functions `read_csv()` and `read_tsv()` for reading CSV and TSV files respectively.

Let's first create an R data.frame, write it to lakehouse using `readr::write_csv()` and read it back with `readr::read_csv()`.

### ⓘ Note

To access Lakehouse files using `readr`, you need to use the *File API path*. In the Lakehouse explorer, right click on the file or folder that you want to access and copy its *File API path* from the contextual menu.

R

```
# create an R data frame  
set.seed(1)  
stocks <- data.frame(  
  time = as.Date('2009-01-01') + 0:9,  
  X = rnorm(10, 20, 1),  
  Y = rnorm(10, 20, 2),  
  Z = rnorm(10, 20, 4)  
)  
stocks
```

Then let's write the data to lakehouse using the *File API path*.

R

```
# write data to lakehouse using the File API path  
temp_csv_api <- "/lakehouse/default/Files/stocks.csv"  
readr::write_csv(stocks,temp_csv_api)
```

Read the data from lakehouse.

R

```
# read data from lakehouse using the File API path  
stocks_readr <- readr::read_csv(temp_csv_api)  
  
# show the content of the R data.frame  
head(stocks_readr)
```

## Data tidying

`tidyR` is an R package that provides tools for working with messy data. The main functions in `tidyR` are designed to help you reshape data into a tidy format. Tidy data has a specific structure where each variable is a column and each observation is a row, which makes it easier to work with data in R and other tools.

For example, the `gather()` function in `tidyR` can be used to convert wide data into long data. Here's an example:

R

```
# convert the stock data into longer data  
library(tidyR)  
stocksL <- gather(data = stocks, key = stock, value = price, X, Y, Z)  
stocksL
```

## Functional programming

`purrr` is an R package that enhances R's functional programming toolkit by providing a complete and consistent set of tools for working with functions and vectors. The best place to start with `purrr` is the family of `map()` functions that allow you to replace many for loops with code that is both more succinct and easier to read. Here's an example of using `map()` to apply a function to each element of a list:

R

```
# double the stock values using purrr  
library(purrr)  
stocks_double = map(stocks %>% select_if(is.numeric), ~.x*2)  
stocks_double
```

## Data manipulation

`dplyr` is an R package that provides a consistent set of verbs that help you solve the most common data manipulation problems, such as selecting variables based on the names, picking cases based on the values, reducing multiple values down to a single summary, and changing the ordering of the rows etc. Here are some examples:

R

```
# pick variables based on their names using select()
stocks_value <- stocks %>% select(X:Z)
stocks_value
```

R

```
# pick cases based on their values using filter()
filter(stocks_value, X >20)
```

R

```
# add new variables that are functions of existing variables using mutate()
library(lubridate)

stocks_wday <- stocks %>%
  select(time:Z) %>%
  mutate(
    weekday = wday(time)
  )

stocks_wday
```

R

```
# change the ordering of the rows using arrange()
arrange(stocks_wday, weekday)
```

R

```
# reduce multiple values down to a single summary using summarise()
stocks_wday %>%
  group_by(weekday) %>%
  summarize(meanX = mean(X), n= n())
```

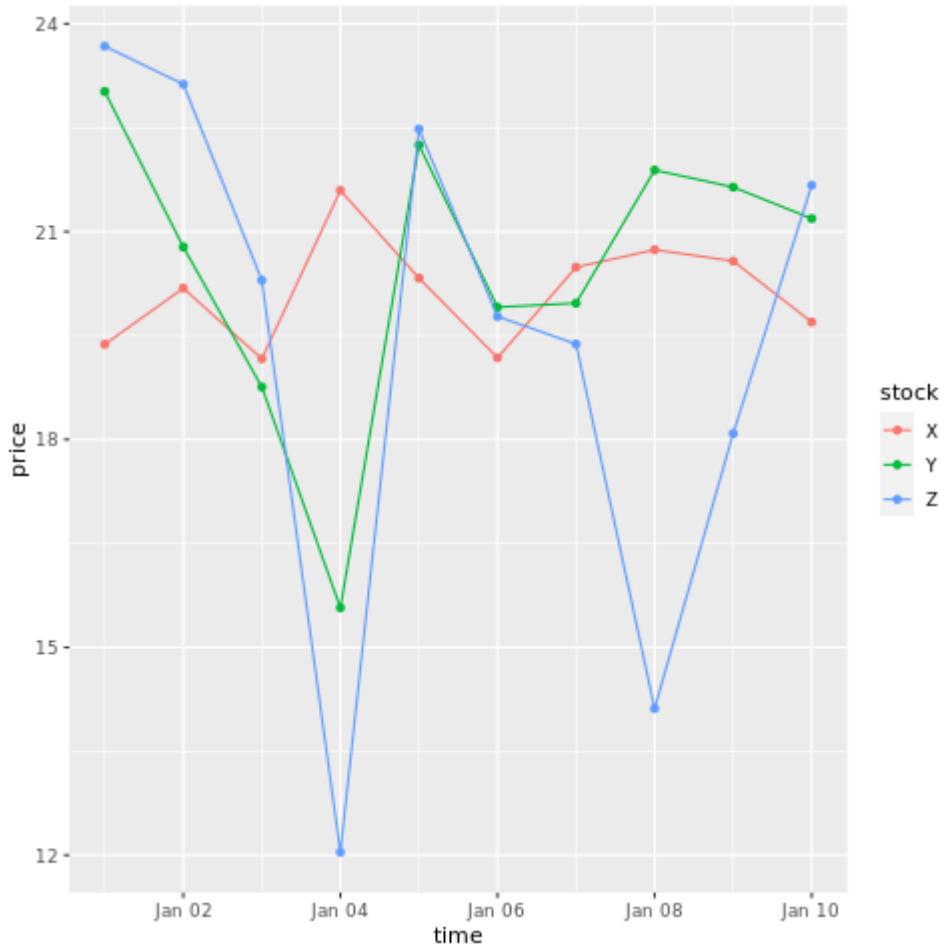
## Data visualization

`ggplot2` is an R package for declaratively creating graphics, based on The Grammar of Graphics. You provide the data, tell `ggplot2` how to map variables to aesthetics, what graphical primitives to use, and it takes care of the details. Here are some examples:

R

```
# draw a chart with points and lines all in one

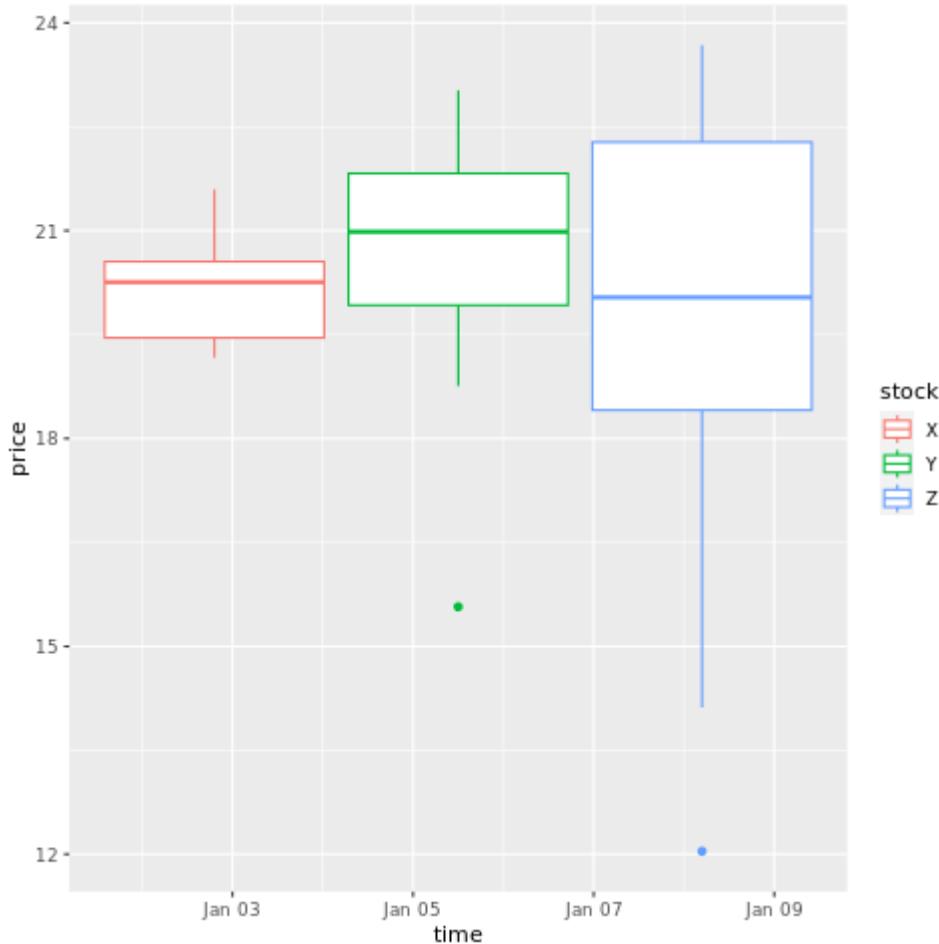
ggplot(stocksL, aes(x=time, y=price, colour = stock)) +
  geom_point()+
  geom_line()
```



R

```
# draw a boxplot

ggplot(stocksL, aes(x=time, y=price, colour = stock)) +
  geom_boxplot()
```

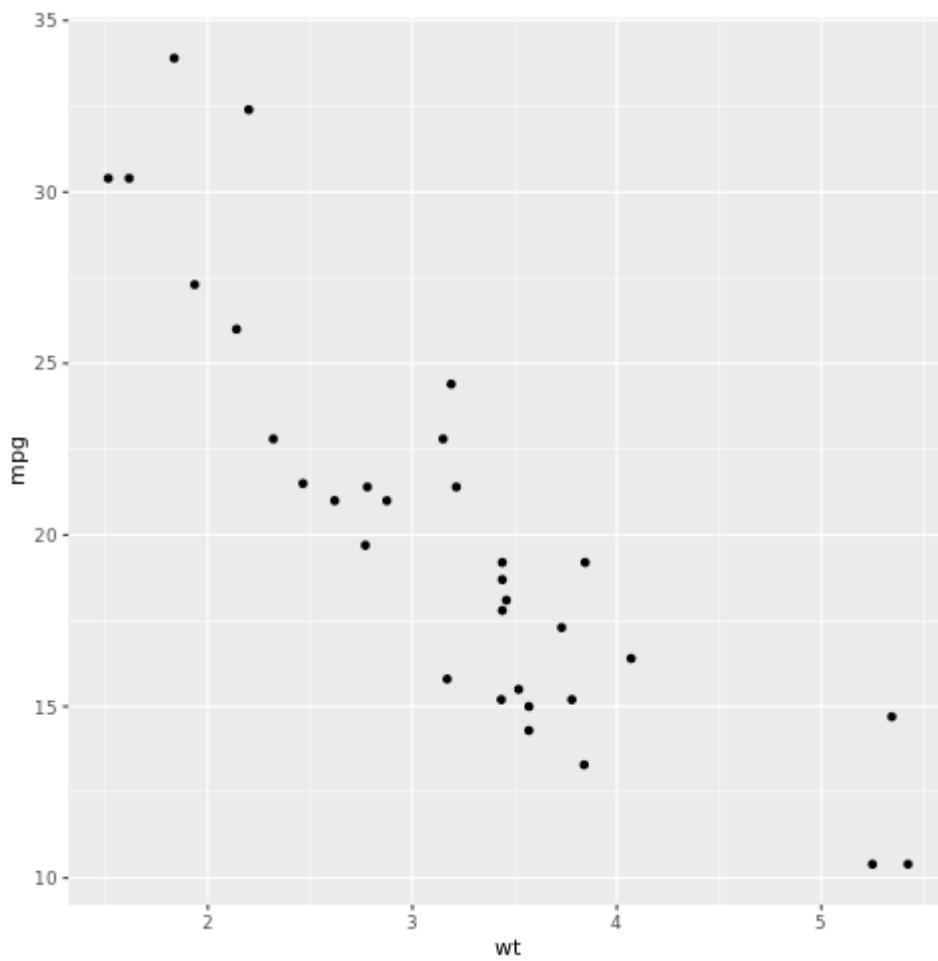


## Model building

The `tidymodels` framework is a collection of packages for modeling and machine learning using `tidyverse` principles. It covers a list of core packages for a wide variety of model building tasks, such as `rsample` for train/test dataset sample splitting, `parsnip` for model specification, `recipes` for data preprocessing, `workflows` for modeling workflows, `tune` for hyperparameters tuning, `yardstick` for model evaluation, `broom` for tiding model outputs, and `dials` for managing tuning parameters. You can learn more about the packages by visiting [tidymodels website](#). Here's an example of building a linear regression model to predict the miles per gallon (mpg) of a car based on its weight (wt):

R

```
# look at the relationship between the miles per gallon (mpg) of a car and
# its weight (wt)
ggplot(mtcars, aes(wt, mpg))+
  geom_point()
```



From the scatterplot, the relationship looks approximately linear and the variance looks constant. Let's try to model this using linear regression.

R

```
library(tidymodels)

# split test and training dataset
set.seed(123)
split <- initial_split(mtcars, prop = 0.7, strata = "cyl")
train <- training(split)
test <- testing(split)

# config the linear regression model
lm_spec <- linear_reg() %>%
  set_engine("lm") %>%
  set_mode("regression")

# build the model
lm_fit <- lm_spec %>%
  fit(mpg ~ wt, data = train)

tidy(lm_fit)
```

Apply the linear regression model to predict on test dataset.

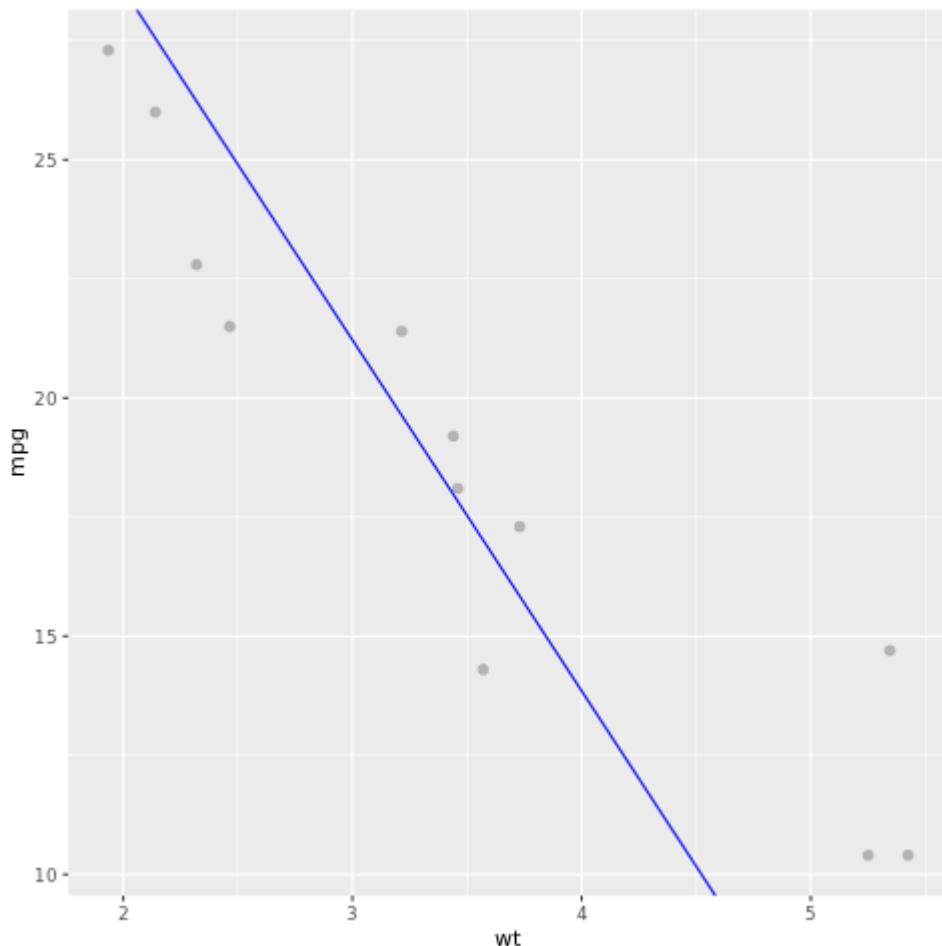
```
R
```

```
# using the lm model to predict on test dataset
predictions <- predict(lm_fit, test)
predictions
```

Let's take a look at the model result. We can draw the model as a line chart and the test ground truth data as points on the same chart. The model looks good.

```
R
```

```
# draw the model as a line chart and the test data groundtruth as points
lm_aug <- augment(lm_fit, test)
ggplot(lm_aug, aes(x = wt, y = mpg)) +
  geom_point(size=2,color="grey70") +
  geom_abline(intercept = lm_fit$fit$coefficients[1], slope =
  lm_fit$fit$coefficients[2], color = "blue")
```



## Next steps

- How to use SparkR
- How to use sparklyr

- R library management
- Visualize data in R
- Tutorial: avocado price prediction
- Tutorial: flight delay prediction

# Visualize data in R

Article • 05/23/2023

The R ecosystem offers multiple graphing libraries that come packed with many different features. By default, every Apache Spark Pool in Microsoft Fabric contains a set of curated and popular open-source libraries. Add or manage extra libraries or versions by using the Microsoft Fabric [library management capabilities](#).

## Important

Microsoft Fabric is currently in PREVIEW. This information relates to a prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, expressed or implied, with respect to the information provided here.

## Prerequisites

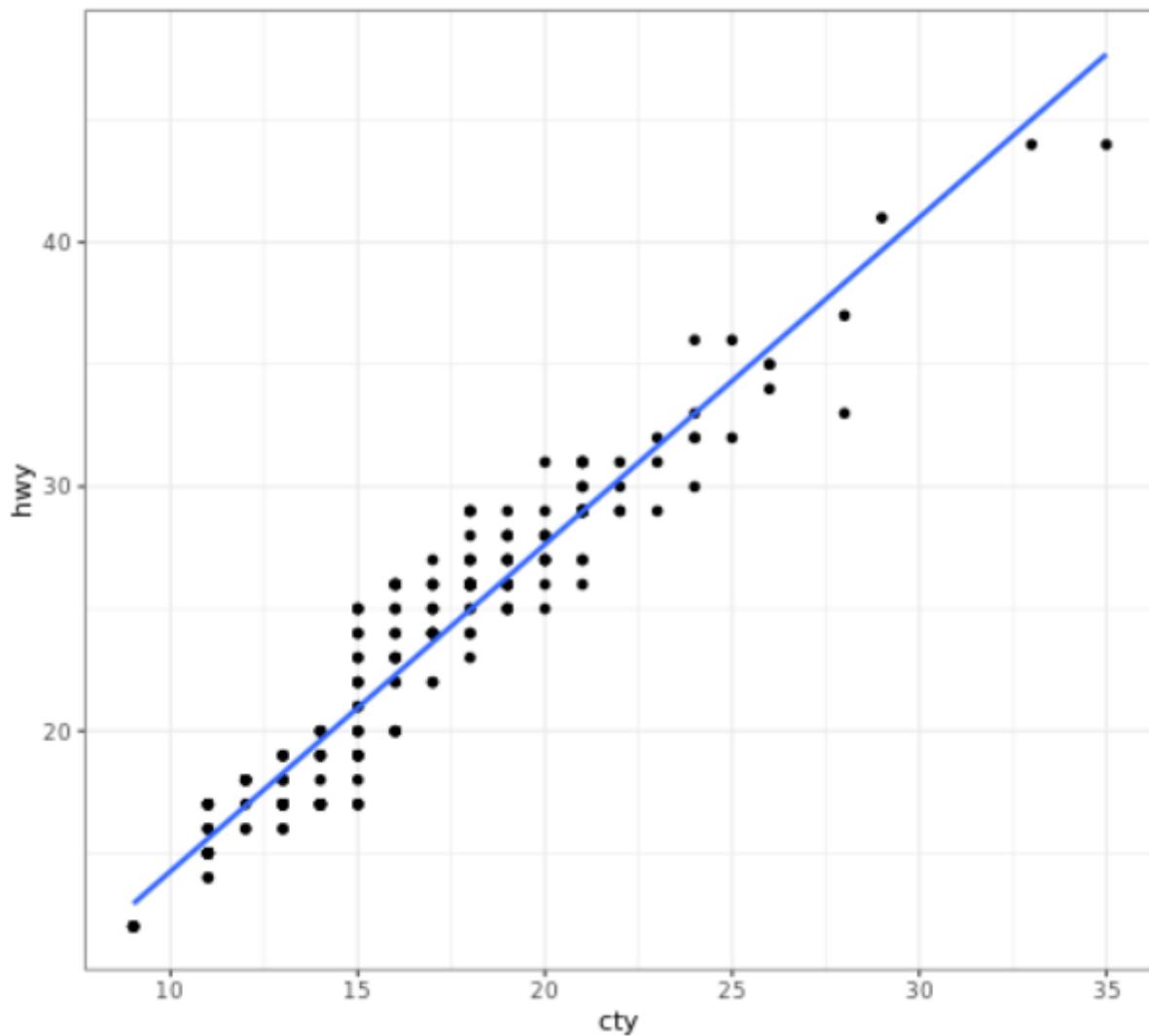
- A Power BI Premium subscription. If you don't have one, see [How to purchase Power BI Premium](#).
- A Power BI workspace with assigned Premium capacity. If you don't have a workspace, use the steps in [Create a workspace](#) to create one and assign it to a Premium capacity.
- Sign in to [Microsoft Fabric](#).
- Open or create a notebook. To learn how, see [How to use Microsoft Fabric notebooks](#).
- Change the primary language by setting the [language option](#) to **SparkR (R)**.
- Attach your notebook to a lakehouse. On the left side, select **Add** to add an existing lakehouse or create a lakehouse.

## ggplot2

The [ggplot2](#) library is popular for data visualization and exploratory data analysis.

## Scatterplot with overlapping points

mpg: city vs highway mileage



Source: midwest

R

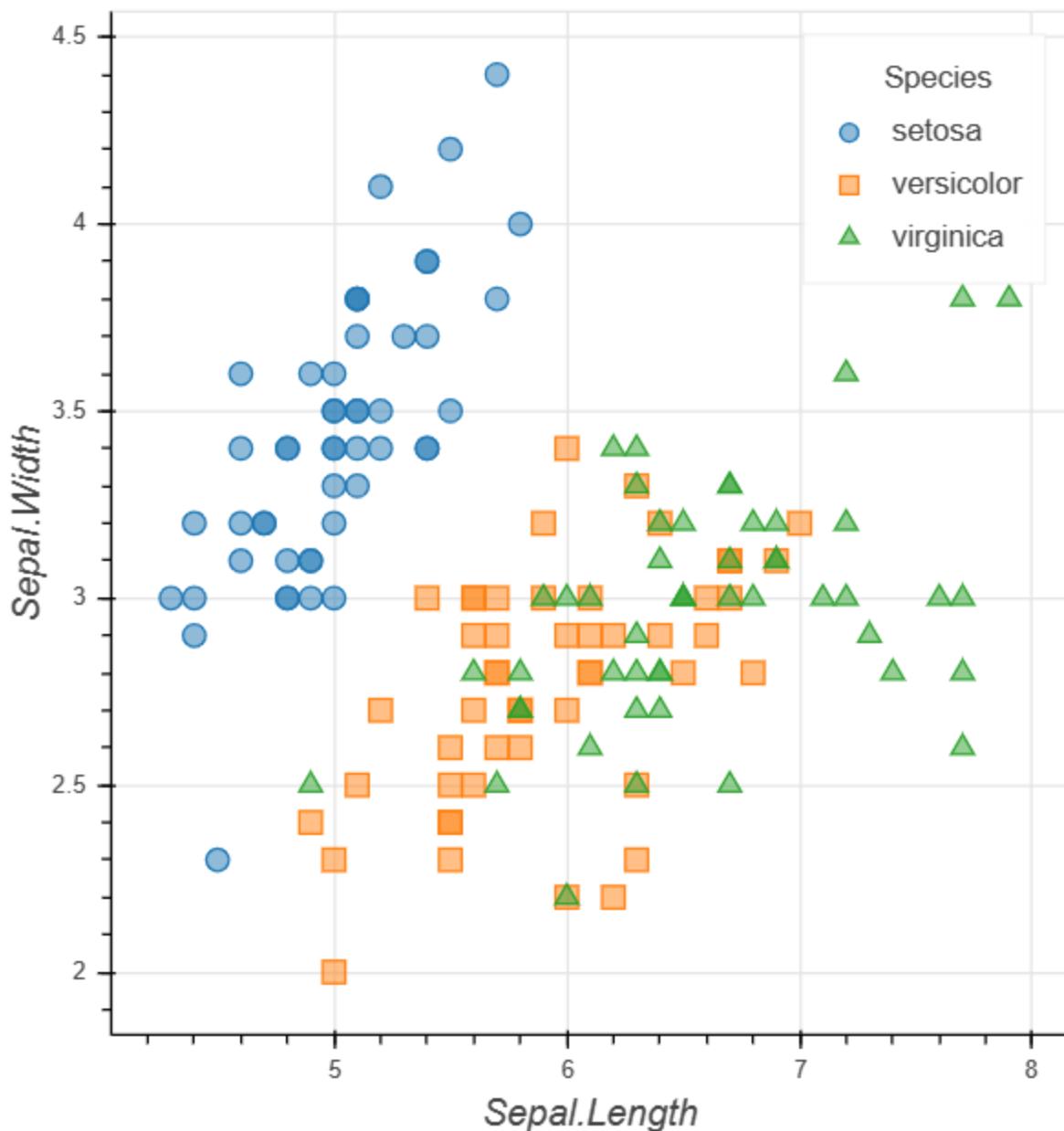
```
%%sparkr
library(ggplot2)
data(mpg, package="ggplot2")
theme_set(theme_bw())

g <- ggplot(mpg, aes(cty, hwy))

# Scatterplot
g + geom_point() +
  geom_smooth(method="lm", se=F) +
  labs(subtitle="mpg: city vs highway mileage",
       y="hwy",
       x="cty",
       title="Scatterplot with overlapping points",
       caption="Source: midwest")
```

rbokeh

rbokeh [↗](#) is a native R plotting library for creating interactive graphics.

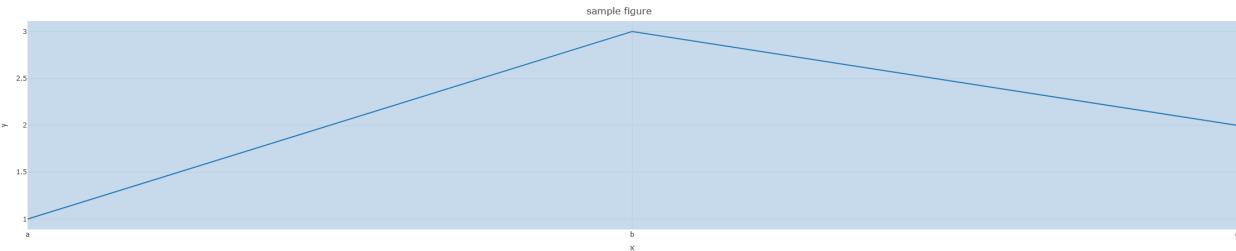


R

```
library(rbokeh)
p <- figure() %>%
  ly_points(Sepal.Length, Sepal.Width, data = iris,
            color = Species, glyph = Species,
            hover = list(Sepal.Length, Sepal.Width))
p
```

## R Plotly

Plotly [↗](#) is an R graphing library that makes interactive, publication-quality graphs.



```
R

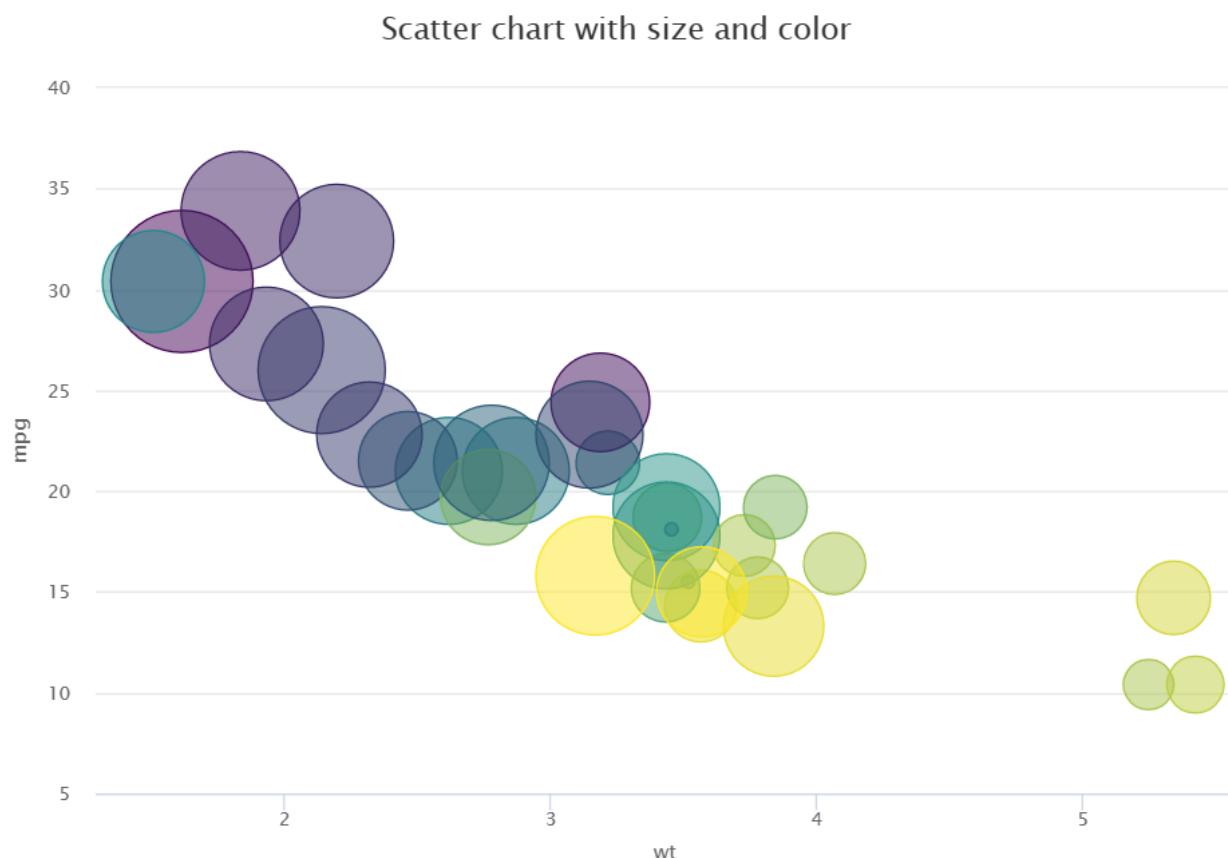
library(plotly)

fig <- plot_ly() %>%
  add_lines(x = c("a", "b", "c"), y = c(1,3,2))%>%
  layout(title="sample figure", xaxis = list(title = 'x'), yaxis =
list(title = 'y'), plot_bgcolor = "#c7daec")

fig
```

## Highcharter

[Highcharter](#) is an R wrapper for Highcharts JavaScript library and its modules.



```
R

library(magrittr)
library(highcharter)
```

```
hchart(mtcars, "scatter", hc_aes(wt, mpg, z = drat, color = hp)) %>%
  hc_title(text = "Scatter chart with size and color")
```

## Next steps

- [How to use SparkR](#)
- [How to use sparklyr](#)
- [How to use Tidyverse](#)
- [R library management](#)
- [Tutorial: avocado price prediction](#)
- [Tutorial: flight delay prediction](#)

# Your First SynapseML Model

Article • 05/23/2023

This tutorial provides a brief introduction to building your first machine learning model using SynapseML, demonstrating how SynapseML makes it easy to do complex machine learning tasks. We use SynapseML to create a small ML training pipeline with a featurization stage and LightGBM regression stage to predict ratings based on review text from a dataset containing book reviews from Amazon. Finally we showcase how SynapseML makes it easy to use prebuilt models to solve problems without having to re-solve them yourself.

## Prerequisites

- Attach your notebook to a lakehouse. On the left side, select **Add** to add an existing lakehouse or create a lakehouse.
- Cognitive Services Key. To obtain a Cognitive Services key, follow the [Quickstart](#).

## Set up your Environment

Import SynapseML libraries and initialize your spark session.

Python

```
from pyspark.sql import SparkSession
from synapse.ml.core.platform import *

spark = SparkSession.builder.getOrCreate()
```

## Load a Dataset

Load your dataset and split it into train and test sets.

Python

```
train, test = (
    spark.read.parquet(
        "wasbs://publicwasb@mmlspark.blob.core.windows.net/BookReviewsFromAmazon10K.
        parquet"
    )
```

```
.limit(1000)
.cache()
.randomSplit([0.8, 0.2])
)

display(train)
```

## Make our Model

Create a simple pipeline to featurize the data using the `TextFeaturizer` from `synapse.ml.featurize.text` and derive a rating from the `LightGBMRegressor`.

Python

```
from pyspark.ml import Pipeline
from synapse.ml.featurize.text import TextFeaturizer
from synapse.ml.lightgbm import LightGBMRegressor

model = Pipeline(
    stages=[
        TextFeaturizer(inputCol="text", outputCol="features"),
        LightGBMRegressor(featuresCol="features", labelCol="rating"),
    ]
).fit(train)
```

## Predict

call the `transform` function on the model to predict and display the output dataframe!

Python

```
display(model.transform(test))
```

## Alternate route - Let the Cognitive Services handle it!

For tasks like this that have a prebuilt solution, try using SynapseML's integration with Cognitive Services to transform your data in one step.

Python

```
from synapse.ml.cognitive import TextSentiment
from synapse.ml.core.platform import find_secret
```

```
model = TextSentiment(  
    textCol="text",  
    outputCol="sentiment",  
    subscriptionKey=find_secret("cognitive-api-key"), # Replace it with your  
cognitive service key, check prerequisites for more details  
).setLocation("eastus")  
  
display(model.transform(test))
```

## Next steps

- [How to use LightGBM with SynapseML](#)
- [How to use Cognitive Services with SynapseML](#)
- [How to perform the same classification task with and without SynapseML](#)

# LightGBM

Article • 05/23/2023

[LightGBM](#) is an open-source, distributed, high-performance gradient boosting (GBDT, GBRT, GBM, or MART) framework. This framework specializes in creating high-quality and GPU enabled decision tree algorithms for ranking, classification, and many other machine learning tasks. LightGBM is part of Microsoft's [DMTK](#) project.

## Advantages of LightGBM

- **Composability:** LightGBM models can be incorporated into existing SparkML Pipelines, and used for batch, streaming, and serving workloads.
- **Performance:** LightGBM on Spark is 10-30% faster than SparkML on the Higgs dataset, and achieves a 15% increase in AUC. [Parallel experiments](#) have verified that LightGBM can achieve a linear speed-up by using multiple machines for training in specific settings.
- **Functionality:** LightGBM offers a wide array of [tunable parameters](#), that one can use to customize their decision tree system. LightGBM on Spark also supports new types of problems such as quantile regression.
- **Cross platform** LightGBM on Spark is available on Spark, PySpark, and SparklyR

## LightGBM Usage:

- LightGBMClassifier: used for building classification models. For example, to predict whether a company bankrupts or not, we could build a binary classification model with LightGBMClassifier.
- LightGBMRegressor: used for building regression models. For example, to predict the house price, we could build a regression model with LightGBMRegressor.
- LightGBMRanker: used for building ranking models. For example, to predict website searching result relevance, we could build a ranking model with LightGBMRanker.

## Prerequisites

- Attach your notebook to a lakehouse. On the left side, select **Add** to add an existing lakehouse or create a lakehouse.

## Bankruptcy Prediction with LightGBM Classifier

In this example, we use LightGBM to build a classification model in order to predict bankruptcy.

## Read dataset

Python

```
from pyspark.sql import SparkSession

# Bootstrap Spark Session
spark = SparkSession.builder.getOrCreate()

from synapse.ml.core.platform import *
```

Python

```
df = (
    spark.read.format("csv")
    .option("header", True)
    .option("inferSchema", True)
    .load(
        "wasbs://publicwasb@mmlspark.blob.core.windows.net/company_bankruptcy_prediction_data.csv"
    )
)
# print dataset size
print("records read: " + str(df.count()))
print("Schema: ")
df.printSchema()
```

Python

```
display(df)
```

## Split the dataset into train and test

Python

```
train, test = df.randomSplit([0.85, 0.15], seed=1)
```

## Add featurizer to convert features to vector

Python

```
from pyspark.ml.feature import VectorAssembler

feature_cols = df.columns[1:]
featurizer = VectorAssembler(inputCols=feature_cols, outputCol="features")
train_data = featurizer.transform(train)[ "Bankrupt?", "features"]
test_data = featurizer.transform(test)[ "Bankrupt?", "features"]
```

## Check if the data is unbalanced

Python

```
display(train_data.groupBy("Bankrupt?").count())
```

## Model Training

Python

```
from synapse.ml.lightgbm import LightGBMClassifier

model = LightGBMClassifier(
    objective="binary", featuresCol="features", labelCol="Bankrupt?",
    isUnbalance=True
)
```

Python

```
model = model.fit(train_data)
```

## Feature Importances Visualization

Python

```
import pandas as pd
import matplotlib.pyplot as plt

feature_importances = model.getFeatureImportances()
fi = pd.Series(feature_importances, index=feature_cols)
fi = fi.sort_values(ascending=True)
f_index = fi.index
f_values = fi.values

# print feature importances
print("f_index:", f_index)
```

```
print("f_values:", f_values)

# plot
x_index = list(range(len(fi)))
x_index = [x / len(fi) for x in x_index]
plt.rcParams["figure.figsize"] = (20, 20)
plt.barh(
    x_index, f_values, height=0.028, align="center", color="tan",
    tick_label=f_index
)
plt.xlabel("importances")
plt.ylabel("features")
plt.show()
```

## Model Prediction

Python

```
predictions = model.transform(test_data)
predictions.limit(10).toPandas()
```

Python

```
from synapse.ml.train import ComputeModelStatistics

metrics = ComputeModelStatistics(
    evaluationMetric="classification",
    labelCol="Bankrupt?",
    scoredLabelsCol="prediction",
).transform(predictions)
display(metrics)
```

## Quantile Regression for Drug Discovery with LightGBMRegressor

In this example, we show how to use LightGBM to build a regression model.

### Read dataset

Python

```
triazines = spark.read.format("libsvm").load(
    "wasbs://publicwasb@mmlspark.blob.core.windows.net/triazines.scale.svmlight"
)
```

Python

```
# print some basic info
print("records read: " + str(triazines.count()))
print("Schema: ")
triazines.printSchema()
display(triazines.limit(10))
```

## Split dataset into train and test

Python

```
train, test = triazines.randomSplit([0.85, 0.15], seed=1)
```

## Model Training

Python

```
from synapse.ml.lightgbm import LightGBMRegressor

model = LightGBMRegressor(
    objective="quantile", alpha=0.2, learningRate=0.3, numLeaves=31
).fit(train)
```

Python

```
print(model.getFeatureImportances())
```

## Model Prediction

Python

```
scoredData = model.transform(test)
display(scoredData)
```

Python

```
from synapse.ml.train import ComputeModelStatistics

metrics = ComputeModelStatistics(
    evaluationMetric="regression", labelCol="label", scoresCol="prediction"
```

```
).transform(scoredData)
display(metrics)
```

# LightGBM Ranker

## Read dataset

Python

```
df = spark.read.format("parquet").load(
    "wasbs://publicwasb@mmlspark.blob.core.windows.net/lightGBMRanker_train.parquet"
)
# print some basic info
print("records read: " + str(df.count()))
print("Schema: ")
df.printSchema()
display(df.limit(10))
```

## Model Training

Python

```
from synapse.ml.lightgbm import LightGBMRanker

features_col = "features"
query_col = "query"
label_col = "labels"
lgbm_ranker = LightGBMRanker(
    labelCol=label_col,
    featuresCol=features_col,
    groupCol=query_col,
    predictionCol="preds",
    leafPredictionCol="leafPreds",
    featuresShapCol="importances",
    repartitionByGroupingColumn=True,
    numLeaves=32,
    numIterations=200,
    evalAt=[1, 3, 5],
    metric="ndcg",
)
```

Python

```
lgbm_ranker_model = lgbm_ranker.fit(df)
```

## Model Prediction

Python

```
dt = spark.read.format("parquet").load(  
    "wasbs://publicwasb@mmlspark.blob.core.windows.net/lightGBMRanker_test.parquet"  
)  
predictions = lgbm_ranker_model.transform(dt)  
predictions.limit(10).toPandas()
```

## Next steps

- How to use Cognitive Services with SynapseML
- How to perform the same classification task with and without SynapseML
- How to use knn model with SynapseML

# Cognitive Services

Article • 05/23/2023

Azure Cognitive Services [↗](#) are a suite of APIs, SDKs, and services available to help developers build intelligent applications without having direct AI or data science skills or knowledge by enabling developers to easily add cognitive features into their applications. The goal of Azure Cognitive Services is to help developers create applications that can see, hear, speak, understand, and even begin to reason. The catalog of services within Azure Cognitive Services can be categorized into five main pillars - Vision, Speech, Language, Web Search, and Decision.

## Usage

### Vision

#### [Computer Vision](#) [↗](#)

- Describe: provides description of an image in human readable language ([Scala](#) [↗](#), [Python](#) [↗](#))
- Analyze (color, image type, face, adult/racy content): analyzes visual features of an image ([Scala](#) [↗](#), [Python](#) [↗](#))
- OCR: reads text from an image ([Scala](#) [↗](#), [Python](#) [↗](#))
- Recognize Text: reads text from an image ([Scala](#) [↗](#), [Python](#) [↗](#))
- Thumbnail: generates a thumbnail of user-specified size from the image ([Scala](#) [↗](#), [Python](#) [↗](#))
- Recognize domain-specific content: recognizes domain-specific content (celebrity, landmark) ([Scala](#) [↗](#), [Python](#) [↗](#))
- Tag: identifies list of words that are relevant to the input image ([Scala](#) [↗](#), [Python](#) [↗](#))

#### [Face](#) [↗](#)

- Detect: detects human faces in an image ([Scala](#) [↗](#), [Python](#) [↗](#))
- Verify: verifies whether two faces belong to a same person, or a face belongs to a person ([Scala](#) [↗](#), [Python](#) [↗](#))
- Identify: finds the closest matches of the specific query person face from a person group ([Scala](#) [↗](#), [Python](#) [↗](#))
- Find similar: finds similar faces to the query face in a face list ([Scala](#) [↗](#), [Python](#) [↗](#))
- Group: divides a group of faces into disjoint groups based on similarity ([Scala](#) [↗](#), [Python](#) [↗](#))

# Speech

## Speech Services ↗

- Speech-to-text: transcribes audio streams ([Scala ↗](#), [Python ↗](#))
- Conversation Transcription: transcribes audio streams into live transcripts with identified speakers. ([Scala ↗](#), [Python ↗](#))
- Text to Speech: Converts text to realistic audio ([Scala ↗](#), [Python ↗](#))

# Language

## Text Analytics ↗

- Language detection: detects language of the input text ([Scala ↗](#), [Python ↗](#))
- Key phrase extraction: identifies the key talking points in the input text ([Scala ↗](#), [Python ↗](#))
- Named entity recognition: identifies known entities and general named entities in the input text ([Scala ↗](#), [Python ↗](#))
- Sentiment analysis: returns a score between 0 and 1 indicating the sentiment in the input text ([Scala ↗](#), [Python ↗](#))
- Healthcare Entity Extraction: Extracts medical entities and relationships from text. ([Scala ↗](#), [Python ↗](#))

# Translation

## Translator ↗

- Translate: Translates text. ([Scala ↗](#), [Python ↗](#))
- Transliterate: Converts text in one language from one script to another script. ([Scala ↗](#), [Python ↗](#))
- Detect: Identifies the language of a piece of text. ([Scala ↗](#), [Python ↗](#))
- BreakSentence: Identifies the positioning of sentence boundaries in a piece of text. ([Scala ↗](#), [Python ↗](#))
- Dictionary Lookup: Provides alternative translations for a word and a small number of idiomatic phrases. ([Scala ↗](#), [Python ↗](#))
- Dictionary Examples: Provides examples that show how terms in the dictionary are used in context. ([Scala ↗](#), [Python ↗](#))
- Document Translation: Translates documents across all supported languages and dialects while preserving document structure and data format. ([Scala ↗](#), [Python ↗](#))

# Form Recognizer

## Form Recognizer

- Analyze Layout: Extract text and layout information from a given document. ([Scala](#) , [Python](#) )
- Analyze Receipts: Detects and extracts data from receipts using optical character recognition (OCR) and our receipt model, enabling you to easily extract structured data from receipts such as merchant name, merchant phone number, transaction date, transaction total, and more. ([Scala](#) , [Python](#) )
- Analyze Business Cards: Detects and extracts data from business cards using optical character recognition (OCR) and our business card model, enabling you to easily extract structured data from business cards such as contact names, company names, phone numbers, emails, and more. ([Scala](#) , [Python](#) )
- Analyze Invoices: Detects and extracts data from invoices using optical character recognition (OCR) and our invoice understanding deep learning models, enabling you to easily extract structured data from invoices such as customer, vendor, invoice ID, invoice due date, total, invoice amount due, tax amount, ship to, bill to, line items and more. ([Scala](#) , [Python](#) )
- Analyze ID Documents: Detects and extracts data from identification documents using optical character recognition (OCR) and our ID document model, enabling you to easily extract structured data from ID documents such as first name, last name, date of birth, document number, and more. ([Scala](#) , [Python](#) )
- Analyze Custom Form: Extracts information from forms (PDFs and images) into structured data based on a model created from a set of representative training forms. ([Scala](#) , [Python](#) )
- Get Custom Model: Get detailed information about a custom model. ([Scala](#) , [Python](#) )
- List Custom Models: Get information about all custom models. ([Scala](#) , [Python](#) )

## Decision

### Anomaly Detector

- Anomaly status of latest point: generates a model using preceding points and determines whether the latest point is anomalous ([Scala](#) , [Python](#) )
- Find anomalies: generates a model using an entire series and finds anomalies in the series ([Scala](#) , [Python](#) )

## Search

- Bing Image search  ([Scala](#) , [Python](#) )
- Azure Cognitive search ([Scala](#) , [Python](#) )

# Prerequisites

- Attach your notebook to a lakehouse. On the left side, select **Add** to add an existing lakehouse or create a lakehouse.
- Cognitive Services Key. To obtain a Cognitive Services key, follow the [Quickstart](#).

## Shared code

To get started, we'll need to add this code to the project:

Python

```
from pyspark.sql.functions import udf, col
from synapse.ml.io.http import HTTPTransformer, http_udf
from requests import Request
from pyspark.sql.functions import lit
from pyspark.ml import PipelineModel
from pyspark.sql.functions import col
import os
```

Python

```
from pyspark.sql import SparkSession
from synapse.ml.core.platform import *

# Bootstrap Spark Session
spark = SparkSession.builder.getOrCreate()
```

Python

```
from synapse.ml.cognitive import *

# A general Cognitive Services key for Text Analytics, Computer Vision and
# Form Recognizer (or use separate keys that belong to each service)
service_key = find_secret("cognitive-api-key") # Replace it with your
# cognitive service key, check prerequisites for more details
service_loc = "eastus"

# A Bing Search v7 subscription key
bing_search_key = find_secret("bing-search-key") # Replace it with your
# cognitive service key, check prerequisites for more details

# An Anomaly Detector subscription key
anomaly_key = find_secret("anomaly-api-key") # Replace it with your
# cognitive service key, check prerequisites for more details
anomaly_loc = "westus2"
```

```

# A Translator subscription key
translator_key = find_secret("translator-key") # Replace it with your
cognitive service key, check prerequisites for more details
translator_loc = "eastus"

# An Azure search key
search_key = find_secret("azure-search-key") # Replace it with your
cognitive service key, check prerequisites for more details

```

## Text Analytics sample

The [Text Analytics](#) service provides several algorithms for extracting intelligent insights from text. For example, we can find the sentiment of given input text. The service will return a score between 0.0 and 1.0 where low scores indicate negative sentiment and high score indicates positive sentiment. This sample uses three simple sentences and returns the sentiment for each.

Python

```

# Create a dataframe that's tied to it's column names
df = spark.createDataFrame(
    [
        ("I am so happy today, its sunny!", "en-US"),
        ("I am frustrated by this rush hour traffic", "en-US"),
        ("The cognitive services on spark aint bad", "en-US"),
    ],
    ["text", "language"],
)

# Run the Text Analytics service with options
sentiment = (
    TextSentiment()
    .setTextCol("text")
    .setLocation(service_loc)
    .setSubscriptionKey(service_key)
    .setOutputCol("sentiment")
    .setErrorCol("error")
    .setLanguageCol("language")
)

# Show the results of your text query in a table format
display(
    sentiment.transform(df).select(
        "text", col("sentiment.document.sentiment").alias("sentiment")
    )
)

```

# Text Analytics for Health Sample

The [Text Analytics for Health Service](#) extracts and labels relevant medical information from unstructured texts such as doctor's notes, discharge summaries, clinical documents, and electronic health records.

Python

```
df = spark.createDataFrame(  
    [  
        ("20mg of ibuprofen twice a day",),  
        ("1tsp of Tylenol every 4 hours",),  
        ("6-drops of Vitamin B-12 every evening",),  
    ],  
    ["text"],  
)  
  
healthcare = (  
    AnalyzeHealthText()  
    .setSubscriptionKey(service_key)  
    .setLocation(service_loc)  
    .setLanguage("en")  
    .setOutputCol("response")  
)  
  
display(healthcare.transform(df))
```

# Translator sample

[Translator](#) is a cloud-based machine translation service and is part of the Azure Cognitive Services family of cognitive APIs used to build intelligent apps. Translator is easy to integrate in your applications, websites, tools, and solutions. It allows you to add multi-language user experiences in 90 languages and dialects and can be used for text translation with any operating system. In this sample, we do a simple text translation by providing the sentences you want to translate and target languages you want to translate to.

Python

```
from pyspark.sql.functions import col, flatten  
  
# Create a dataframe including sentences you want to translate  
df = spark.createDataFrame(  
    [[[ "Hello, what is your name?", "Bye" ], ]],  
    [  
        "text",  
    ],
```

```

)
# Run the Translator service with options
translate = (
    Translate()
    .setSubscriptionKey(translator_key)
    .setLocation(translator_loc)
    .setTextCol("text")
    .setToLanguage(["zh-Hans"])
    .setOutputCol("translation")
)
# Show the results of the translation.
display(
    translate.transform(df)
    .withColumn("translation", flatten(col("translation.translations")))
    .withColumn("translation", col("translation.text"))
    .select("translation")
)

```

## Form Recognizer sample

Form Recognizer [↗](#) is a part of Azure Applied AI Services that lets you build automated data processing software using machine learning technology. Identify and extract text, key/value pairs, selection marks, tables, and structure from your documents. The service outputs structured data that includes the relationships in the original file, bounding boxes, confidence and more. In this sample, we analyze a business card image and extract its information into structured data.

Python

```

from pyspark.sql.functions import col, explode

# Create a dataframe containing the source files
imageDf = spark.createDataFrame(
    [
        (
            "https://mmlspark.blob.core.windows.net/datasets/FormRecognizer/business_card.jpg",
            )
        ],
        [
            "source",
        ],
)

# Run the Form Recognizer service
analyzeBusinessCards = (
    AnalyzeBusinessCards()

```

```

.setSubscriptionKey(service_key)
.setLocation(service_loc)
.setImageUrlCol("source")
.setOutputCol("businessCards")
)

# Show the results of recognition.
display(
    analyzeBusinessCards.transform(imageDf)
    .withColumn(
        "documents",
    explode(col("businessCards.analyzeResult.documentResults.fields"))
    )
    .select("source", "documents")
)

```

## Computer Vision sample

Computer Vision  analyzes images to identify structure such as faces, objects, and natural-language descriptions. In this sample, we tag a list of images. Tags are one-word descriptions of things in the image like recognizable objects, people, scenery, and actions.

Python

```

# Create a dataframe with the image URLs
base_url = "https://raw.githubusercontent.com/Azure-Samples/cognitive-
services-sample-data-files/master/ComputerVision/Images/"
df = spark.createDataFrame(
    [
        (base_url + "objects.jpg",),
        (base_url + "dog.jpg",),
        (base_url + "house.jpg",),
    ],
    [
        "image",
    ],
)

# Run the Computer Vision service. Analyze Image extracts information
# from/about the images.
analysis = (
    AnalyzeImage()
    .setLocation(service_loc)
    .setSubscriptionKey(service_key)
    .setVisualFeatures(
        ["Categories", "Color", "Description", "Faces", "Objects", "Tags"]
    )
    .setOutputCol("analysis_results")
    .setImageUrlCol("image")
)

```

```

    .setErrorCol("error")
)

# Show the results of what you wanted to pull out of the images.
display(analysis.transform(df).select("image",
"analysis_results.description.tags"))

```

## Bing Image Search sample

[Bing Image Search](#) searches the web to retrieve images related to a user's natural language query. In this sample, we use a text query that looks for images with quotes. It returns a list of image URLs that contain photos related to our query.

Python

```

# Number of images Bing will return per query
imgsPerBatch = 10
# A list of offsets, used to page into the search results
offsets = [(i * imgsPerBatch,) for i in range(100)]
# Since web content is our data, we create a dataframe with options on that
# data: offsets
bingParameters = spark.createDataFrame(offsets, ["offset"])

# Run the Bing Image Search service with our text query
bingSearch = (
    BingImageSearch()
    .setSubscriptionKey(bing_search_key)
    .setOffsetCol("offset")
    .setQuery("Martin Luther King Jr. quotes")
    .setCount(imgsPerBatch)
    .setOutputCol("images")
)

# Transformer that extracts and flattens the richly structured output of
# Bing Image Search into a simple URL column
getUrls = BingImageSearch.getUrlTransformer("images", "url")

# This displays the full results returned, uncomment to use
# display(bingSearch.transform(bingParameters))

# Since we have two services, they are put into a pipeline
pipeline = PipelineModel(stages=[bingSearch, getUrls])

# Show the results of your search: image URLs
display(pipeline.transform(bingParameters))

```

## Speech-to-Text sample

The [Speech-to-text](#) service converts streams or files of spoken audio to text. In this sample, we transcribe one audio file.

Python

```
# Create a dataframe with our audio URLs, tied to the column called "url"
df = spark.createDataFrame(
    [{"url": "https://mmlspark.blob.core.windows.net/datasets/Speech/audio2.wav"}],
    ["url"]
)

# Run the Speech-to-text service to translate the audio into text
speech_to_text = (
    SpeechToTextSDK()
    .setSubscriptionKey(service_key)
    .setLocation(service_loc)
    .setOutputCol("text")
    .setAudioDataCol("url")
    .setLanguage("en-US")
    .setProfanity("Masked")
)

# Show the results of the translation
display(speech_to_text.transform(df).select("url", "text.DisplayText"))
```

## Text-to-Speech sample

[Text to speech](#) is a service that allows one to build apps and services that speak naturally, choosing from more than 270 neural voices across 119 languages and variants.

Python

```
from synapse.ml.cognitive import TextToSpeech

fs = ""
if running_on_databricks():
    fs = "dbfs:"
elif running_on_synapse_internal():
    fs = "Files"

# Create a dataframe with text and an output file location
df = spark.createDataFrame([
    (
        "Reading out loud is fun! Check out aka.ms/spark for more
        information",
        fs + "/output.mp3",
    )
])
```

```

        ],
        ["text", "output_file"],
    )

tts = (
    TextToSpeech()
    .setSubscriptionKey(service_key)
    .setTextCol("text")
    .setLocation(service_loc)
    .setVoiceName("en-US-JennyNeural")
    .setOutputFileCol("output_file")
)

# Check to make sure there were no errors during audio creation
display(tts.transform(df))

```

## Anomaly Detector sample

Anomaly Detector [↗](#) is great for detecting irregularities in your time series data. In this sample, we use the service to find anomalies in the entire time series.

Python

```

# Create a dataframe with the point data that Anomaly Detector requires
df = spark.createDataFrame(
    [
        ("1972-01-01T00:00:00Z", 826.0),
        ("1972-02-01T00:00:00Z", 799.0),
        ("1972-03-01T00:00:00Z", 890.0),
        ("1972-04-01T00:00:00Z", 900.0),
        ("1972-05-01T00:00:00Z", 766.0),
        ("1972-06-01T00:00:00Z", 805.0),
        ("1972-07-01T00:00:00Z", 821.0),
        ("1972-08-01T00:00:00Z", 20000.0),
        ("1972-09-01T00:00:00Z", 883.0),
        ("1972-10-01T00:00:00Z", 898.0),
        ("1972-11-01T00:00:00Z", 957.0),
        ("1972-12-01T00:00:00Z", 924.0),
        ("1973-01-01T00:00:00Z", 881.0),
        ("1973-02-01T00:00:00Z", 837.0),
        ("1973-03-01T00:00:00Z", 9000.0),
    ],
    ["timestamp", "value"],
).withColumn("group", lit("series1"))

# Run the Anomaly Detector service to look for irregular data
anomaly_detector = (
    SimpleDetectAnomalies()
    .setSubscriptionKey(anomaly_key)
    .setLocation(anomaly_loc)
    .setTimestampCol("timestamp")
)

```

```

    .setValueCol("value")
    .setOutputCol("anomalies")
    .setGroupbyCol("group")
    .setGranularity("monthly")
)

# Show the full results of the analysis with the anomalies marked as "True"
display(
    anamoly_detector.transform(df).select("timestamp", "value",
    "anomalies.isAnomaly")
)

```

## Arbitrary web APIs

With HTTP on Spark, any web service can be used in your big data pipeline. In this example, we use the [World Bank API](#) to get information about various countries around the world.

Python

```

# Use any requests from the python requests library

def world_bank_request(country):
    return Request(
        "GET", "http://api.worldbank.org/v2/country/{}?
format=json".format(country)
    )

# Create a dataframe with specifies which countries we want data on
df = spark.createDataFrame([('br',), ('usa',)], ['country']).withColumn(
    "request", http_udf(world_bank_request)(col("country")))
)

# Much faster for big data because of the concurrency :)
client = (
    HTTPTransformer().setConcurrency(3).setInputCol("request").setOutputCol("res
ponse")
)

# Get the body of the response

def get_response_body(resp):
    return resp.entity.content.decode()

# Show the details of the country data returned
display(

```

```
    client.transform(df).select(
        "country", udf(get_response_body)(col("response")).alias("response")
    )
}
```

## Next steps

- How to perform the same classification task with and without SynapseML
- How to use knn model with SynapseML
- How to use ONNX with SynapseML - Deep Learning

# Classification - Before and After SynapseML

Article • 05/23/2023

## Prerequisites

- Attach your notebook to a lakehouse. On the left side, select **Add** to add an existing lakehouse or create a lakehouse.

## Introduction

In this tutorial, we perform the same classification task in two different ways: once using plain `pyspark` and once using the `synapseml` library. The two methods yield the same performance, but one of the two libraries is drastically simpler to use and iterate on (can you guess which one?).

The task is simple: Predict whether a user's review of a book sold on Amazon is good (rating > 3) or bad based on the text of the review. We accomplish it by training LogisticRegression learners with different hyperparameters and choosing the best model.

Python

```
from pyspark.sql import SparkSession

# Bootstrap Spark Session
spark = SparkSession.builder.getOrCreate()
```

## Read the data

We download and read in the data.

Python

```
rawData = spark.read.parquet(
    "wasbs://publicwasb@mmlspark.blob.core.windows.net/BookReviewsFromAmazon10K.
    parquet"
)
rawData.show(5)
```

# Extract more features and process data

Real data however is more complex than the above dataset. It's common for a dataset to have features of multiple types: text, numeric, categorical. To illustrate how difficult it is to work with these datasets, we add two numerical features to the dataset: the **word count** of the review and the **mean word length**.

Python

```
from pyspark.sql.functions import udf
from pyspark.sql.types import *

def wordCount(s):
    return len(s.split())

def wordLength(s):
    import numpy as np

    ss = [len(w) for w in s.split()]
    return round(float(np.mean(ss)), 2)

wordLengthUDF = udf(wordLength, DoubleType())
wordCountUDF = udf(wordCount, IntegerType())
```

Python

```
from synapse.ml.stages import UDFTransformer

wordLength = "wordLength"
wordCount = "wordCount"
wordLengthTransformer = UDFTransformer(
    inputCol="text", outputCol=wordLength, udf=wordLengthUDF
)
wordCountTransformer = UDFTransformer(
    inputCol="text", outputCol=wordCount, udf=wordCountUDF
)
```

Python

```
from pyspark.ml import Pipeline

data = (
    Pipeline(stages=[wordLengthTransformer, wordCountTransformer])
    .fit(rawData)
    .transform(rawData)
    .withColumn("label", rawData["rating"] > 3)
```

```
.drop("rating")
)
```

Python

```
data.show(5)
```

## Classify using pyspark

To choose the best LogisticRegression classifier using the `pyspark` library, need to *explicitly* perform the following steps:

1. Process the features:

- Tokenize the text column
- Hash the tokenized column into a vector using hashing
- Merge the numeric features with the vector

2. Process the label column: cast it into the proper type.

3. Train multiple LogisticRegression algorithms on the `train` dataset with different hyperparameters

4. Compute the area under the ROC curve for each of the trained models and select the model with the highest metric as computed on the `test` dataset

5. Evaluate the best model on the `validation` set

As you can see, there's numerous work involved and many steps where something can go wrong!

Python

```
from pyspark.ml.feature import Tokenizer, HashingTF
from pyspark.ml.feature import VectorAssembler

# Featurize text column
tokenizer = Tokenizer(inputCol="text", outputCol="tokenizedText")
numFeatures = 10000
hashingScheme = HashingTF(
    inputCol="tokenizedText", outputCol="TextFeatures",
    numFeatures=numFeatures
)
tokenizedData = tokenizer.transform(data)
featurizedData = hashingScheme.transform(tokenizedData)

# Merge text and numeric features in one feature column
featureColumnsArray = ["TextFeatures", "wordCount", "wordLength"]
assembler = VectorAssembler(inputCols=featureColumnsArray,
```

```

outputCol="features")
assembledData = assembler.transform(featurizedData)

# Select only columns of interest
# Convert rating column from boolean to int
processedData = assembledData.select("label", "features").withColumn(
    "label", assembledData.label.cast(IntegerType()))
)

```

Python

```

from pyspark.ml.evaluation import BinaryClassificationEvaluator
from pyspark.ml.classification import LogisticRegression

# Prepare data for learning
train, test, validation = processedData.randomSplit([0.60, 0.20, 0.20],
seed=123)

# Train the models on the 'train' data
lrHyperParams = [0.05, 0.1, 0.2, 0.4]
logisticRegressions = [
    LogisticRegression(regParam=hyperParam) for hyperParam in lrHyperParams
]
evaluator = BinaryClassificationEvaluator(
    rawPredictionCol="rawPrediction", metricName="areaUnderROC"
)
metrics = []
models = []

# Select the best model
for learner in logisticRegressions:
    model = learner.fit(train)
    models.append(model)
    scoredData = model.transform(test)
    metrics.append(evaluator.evaluate(scoredData))
bestMetric = max(metrics)
bestModel = models[metrics.index(bestMetric)]

# Get AUC on the validation dataset
scoredVal = bestModel.transform(validation)
print(evaluator.evaluate(scoredVal))

```

## Classify using synapseml

Life is a lot simpler when using `synapseml`!

1. The `TrainClassifier` Estimator featurizes the data internally, as long as the columns selected in the `train`, `test`, `validation` dataset represent the features

2. The `FindBestModel` Estimator finds the best model from a pool of trained models by finding the model that performs best on the `test` dataset given the specified metric
3. The `ComputeModelStatistics` Transformer computes the different metrics on a scored dataset (in our case, the `validation` dataset) at the same time

Python

```
from synapse.ml.train import TrainClassifier, ComputeModelStatistics
from synapse.ml.automl import FindBestModel

# Prepare data for learning
train, test, validation = data.randomSplit([0.60, 0.20, 0.20], seed=123)

# Train the models on the 'train' data
lrHyperParams = [0.05, 0.1, 0.2, 0.4]
logisticRegressions = [
    LogisticRegression(regParam=hyperParam) for hyperParam in lrHyperParams
]
lrmmodels = [
    TrainClassifier(model=lrm, labelCol="label",
numFeatures=10000).fit(train)
    for lrm in logisticRegressions
]

# Select the best model
bestModel = FindBestModel(evaluationMetric="AUC", models=lrmmodels).fit(test)

# Get AUC on the validation dataset
predictions = bestModel.transform(validation)
metrics = ComputeModelStatistics().transform(predictions)
print(
    "Best model's AUC on validation set = "
    + "{0:.2f}%".format(metrics.first()["AUC"] * 100)
)
```

## Next steps

- How to use knn model with SynapseML
- How to use ONNX with SynapseML - Deep Learning
- How to use Kernel SHAP to explain a tabular classification model

# Exploring Art across Culture and Medium with Fast, Conditional, k-Nearest Neighbors

Article • 05/23/2023

This notebook serves as a guideline for match-finding via k-nearest-neighbors. We set up code that allows queries involving cultures and mediums of art amassed from the Metropolitan Museum of Art in NYC and the Rijksmuseum in Amsterdam.

## Prerequisites

- Attach your notebook to a lakehouse. On the left side, select **Add** to add an existing lakehouse or create a lakehouse.

## Overview of the BallTree

The structure functioning behind the kNN model is a BallTree, which is a recursive binary tree where each node (or "ball") contains a partition of the points of data to be queried. Building a BallTree involves assigning data points to the "ball" whose center they're closest to (with respect to a certain specified feature), resulting in a structure that allows binary-tree-like traversal and lends itself to finding k-nearest neighbors at a BallTree leaf.

## Setup

Import necessary Python libraries and prepare dataset.

Python

```
from synapse.ml.core.platform import *

if running_on_binder():
    from IPython import get_ipython
```

Python

```
from pyspark.sql.types import BooleanType
from pyspark.sql.types import *
from pyspark.ml.feature import Normalizer
from pyspark.sql.functions import lit, array, array_contains, udf, col,
```

```

struct
from synapse.ml.nn import ConditionalKNN, ConditionalKNNModel
from PIL import Image
from io import BytesIO

import requests
import numpy as np
import matplotlib.pyplot as plt
from pyspark.sql import SparkSession

# Bootstrap Spark Session
spark = SparkSession.builder.getOrCreate()

```

Our dataset comes from a table containing artwork information from both the Met and Rijks museums. The schema is as follows:

- **id**: A unique identifier for a piece of art
  - Sample Met id: 388395
  - Sample Rijks id: SK-A-2344
- **Title**: Art piece title, as written in the museum's database
- **Artist**: Art piece artist, as written in the museum's database
- **Thumbnail\_Url**: Location of a JPEG thumbnail of the art piece
- **Image\_Url**: Location of an image of the art piece hosted on the Met/Rijks website
- **Culture**: Category of culture that the art piece falls under
  - Sample culture categories: *latin american, egyptian*, etc.
- **Classification**: Category of medium that the art piece falls under
  - Sample medium categories: *woodwork, paintings*, etc.
- **Museum\_Page**: Link to the work of art on the Met/Rijks website
- **Norm\_Features**: Embedding of the art piece image
- **Museum**: Specifies which museum the piece originated from

Python

```

# loads the dataset and the two trained CKNN models for querying by medium
# and culture
df = spark.read.parquet(
    "wasbs://publicwasb@mmlspark.blob.core.windows.net/met_and rijks.parquet"
)
display(df.drop("Norm_Features"))

```

## Define categories to be queried on

We're using two kNN models: one for culture, and one for medium.

Python

```
# mediums = ['prints', 'drawings', 'ceramics', 'textiles', 'paintings',
"musical instruments", "glass", 'accessories', 'photographs', "metalwork",
#           "sculptures", "weapons", "stone", "precious", "paper",
"woodwork", "leatherwork", "uncategorized"]

mediums = ["paintings", "glass", "ceramics"]

# cultures = ['african (general)', 'american', 'ancient american', 'ancient
#             asian', 'ancient european', 'ancient middle-eastern', 'asian (general)',
#             'austrian', 'belgian', 'british', 'chinese', 'czech', 'dutch',
#             'egyptian']#, 'european (general)', 'french', 'german', 'greek',
#             'iranian', 'italian', 'japanese', 'latin american', 'middle
#             eastern', 'roman', 'russian', 'south asian', 'southeast asian',
#             'spanish', 'swiss', 'various']

cultures = ["japanese", "american", "african (general)"]

# Uncomment the above for more robust and large scale searches!

classes = cultures + mediums

medium_set = set(mediums)
culture_set = set(cultures)
selected_ids = {"AK-RBK-17525-2", "AK-MAK-1204", "AK-RAK-2015-2-9"}

small_df = df.where(
    udf(
        lambda medium, culture, id_val: (medium in medium_set)
        or (culture in culture_set)
        or (id_val in selected_ids),
        BooleanType(),
    )("Classification", "Culture", "id")
)

small_df.count()
```

## Define and fit ConditionalKNN models

We create ConditionalKNN models for both the medium and culture columns; each model takes in an output column, features column (feature vector), values column (cell values under the output column), and label column (the quality that the respective KNN is conditioned on).

Python

```
medium_cknn = (
    ConditionalKNN()
    .setOutputCol("Matches")
```

```
.setFeaturesCol("Norm_Features")
.setValuesCol("Thumbnail_Url")
.setLabelCol("Classification")
.fit(small_df)
)
```

Python

```
culture_cknn = (
    ConditionalKNN()
    .setOutputCol("Matches")
    .setFeaturesCol("Norm_Features")
    .setValuesCol("Thumbnail_Url")
    .setLabelCol("Culture")
    .fit(small_df)
)
```

## Define matching and visualizing methods

After the initial dataset and category setup, we prepare methods that will query and visualize the conditional kNN's results.

`addMatches()` creates a Dataframe with a handful of matches per category.

Python

```
def add_matches(classes, cknn, df):
    results = df
    for label in classes:
        results = cknn.transform(
            results.withColumn("conditioner", array(lit(label)))
            .withColumnRenamed("Matches", "Matches_{}".format(label)))
    return results
```

`plot_urls()` calls `plot_img` to visualize top matches for each category into a grid.

Python

```
def plot_img(axis, url, title):
    try:
        response = requests.get(url)
        img = Image.open(BytesIO(response.content)).convert("RGB")
        axis.imshow(img, aspect="equal")
    except:
        pass
    if title is not None:
        axis.set_title(title, fontsize=4)
    axis.axis("off")
```

```

def plot_urls(url_arr, titles, filename):
    nx, ny = url_arr.shape

    plt.figure(figsize=(nx * 5, ny * 5), dpi=1600)
    fig, axes = plt.subplots(ny, nx)

    # reshape required in the case of 1 image query
    if len(axes.shape) == 1:
        axes = axes.reshape(1, -1)

    for i in range(nx):
        for j in range(ny):
            if j == 0:
                plot_img(axes[j, i], url_arr[i, j], titles[i])
            else:
                plot_img(axes[j, i], url_arr[i, j], None)

    plt.savefig(filename, dpi=1600) # saves the results as a PNG

    display(plt.show())

```

## Putting it all together

We define `test_all()` to take in the data, CKNN models, the art id values to query on, and the file path to save the output visualization to. The medium and culture models were previously trained and loaded.

Python

```

# main method to test a particular dataset with two CKNN models and a set of
# art IDs, saving the result to filename.png

def test_all(data, cknn_medium, cknn_culture, test_ids, root):
    is_nice_obj = udf(lambda obj: obj in test_ids, BooleanType())
    test_df = data.where(is_nice_obj("id"))

    results_df_medium = add_matches(mediums, cknn_medium, test_df)
    results_df_culture = add_matches(cultures, cknn_culture,
    results_df_medium)

    results = results_df_culture.collect()

    original_urls = [row["Thumbnail_Url"] for row in results]

    culture_urls = [
        [row["Matches_{}".format(label)][0]["value"] for row in results]
        for label in cultures
    ]

```

```

culture_url_arr = np.array([original_urls] + culture_urls)[:, :]
plot_urls(culture_url_arr, ["Original"] + cultures, root +
"matches_by_culture.png")

medium_urls = [
    [row["Matches_{}".format(label)][0]["value"] for row in results]
    for label in mediums
]
medium_url_arr = np.array([original_urls] + medium_urls)[:, :]
plot_urls(medium_url_arr, ["Original"] + mediums, root +
"matches_by_medium.png")

return results_df_culture

```

## Demo

The following cell performs batched queries given desired image IDs and a filename to save the visualization.

Python

```
# sample query
result_df = test_all(small_df, medium_cknn, culture_cknn, selected_ids,
root=".")
```

## Next steps

- How to use ONNX with SynapseML - Deep Learning
- How to use Kernel SHAP to explain a tabular classification model
- How to use SynapseML for multivariate anomaly detection

# ONNX Inference on Spark

Article • 05/23/2023

In this example, we train a LightGBM model, convert the model to ONNX format and use the converted model to infer some testing data on Spark.

Python dependencies:

- onnxmltools==1.7.0
- lightgbm==3.2.1

## Prerequisites

- Attach your notebook to a lakehouse. On the left side, select **Add** to add an existing lakehouse or create a lakehouse.
- You may need to install onnxmltools by `!pip install onnxmltools==1.7.0`

Load training data

Python

```
from pyspark.sql import SparkSession

# Bootstrap Spark Session
spark = SparkSession.builder.getOrCreate()

from synapse.ml.core.platform import *
```

Python

```
df = (
    spark.read.format("csv")
    .option("header", True)
    .option("inferSchema", True)
    .load(
        "wasbs://publicwasb@mmlspark.blob.core.windows.net/company_bankruptcy_prediction_data.csv"
    )
)

display(df)
```

## Use LightGBM to train a model

Python

```
from pyspark.ml.feature import VectorAssembler
from synapse.ml.lightgbm import LightGBMClassifier

feature_cols = df.columns[1:]
featurizer = VectorAssembler(inputCols=feature_cols, outputCol="features")

train_data = featurizer.transform(df)[ "Bankrupt?", "features"]

model = (
    LightGBMClassifier(featuresCol="features", labelCol="Bankrupt?")
    .setEarlyStoppingRound(300)
    .setLambdaL1(0.5)
    .setNumIterations(1000)
    .setNumThreads(-1)
    .setMaxDeltaStep(0.5)
    .setNumLeaves(31)
    .setMaxDepth(-1)
    .setBaggingFraction(0.7)
    .setFeatureFraction(0.7)
    .setBaggingFreq(2)
    .setObjective("binary")
    .setIsUnbalance(True)
    .setMinSumHessianInLeaf(20)
    .setMinGainToSplit(0.01)
)
model = model.fit(train_data)
```

Export the trained model to a LightGBM booster, convert it to ONNX format.

Python

```
import lightgbm as lgb
from lightgbm import Booster, LGBMClassifier

def convertModel(lgbm_model: LGBMClassifier or Booster, input_size: int) ->
bytes:
    from onnxxmltools.convert import convert_lightgbm
    from onnxconverter_common.data_types import FloatTensorType

    initial_types = [("input", FloatTensorType([-1, input_size]))]
    onnx_model = convert_lightgbm(
        lgbm_model, initial_types=initial_types, target_opset=9
    )
    return onnx_model.SerializeToString()
```

```
booster_model_str = model.getLightGBMBooster().modelStr().get()
booster = lgb.Booster(model_str=booster_model_str)
model_payload_ml = convertModel(booster, len(feature_cols))
```

Load the ONNX payload into an `ONNXModel`, and inspect the model inputs and outputs.

Python

```
from synapse.ml.onnx import ONNXModel

onnx_ml = ONNXModel().setModelPayload(model_payload_ml)

print("Model inputs:" + str(onnx_ml.getModelInputs()))
print("Model outputs:" + str(onnx_ml.getModelOutputs()))
```

Map the model input to the input dataframe's column name (FeedDict), and map the output dataframe's column names to the model outputs (FetchDict).

Python

```
onnx_ml = (
    onnx_ml.setDeviceType("CPU")
    .setFeedDict({"input": "features"})
    .setFetchDict({"probability": "probabilities", "prediction": "label"})
    .setMiniBatchSize(5000)
)
```

Create some testing data and transform the data through the ONNX model.

Python

```
from pyspark.ml.feature import VectorAssembler
import pandas as pd
import numpy as np

n = 1000 * 1000
m = 95
test = np.random.rand(n, m)
testPd = pd.DataFrame(test)
cols = list(map(str, testPd.columns))
testDf = spark.createDataFrame(testPd)
testDf = testDf.union(testDf).repartition(200)
testDf = (
    VectorAssembler()
    .setInputCols(cols)
    .setOutputCol("features")
    .transform(testDf)
    .drop(*cols)
    .cache()
)
```

```
display(onnx_ml.transform(testDf))
```

## Next steps

- How to use Kernel SHAP to explain a tabular classification model
- How to use SynapseML for multivariate anomaly detection
- How to Build a Search Engine with SynapseML

# Interpretability - Tabular SHAP explainer

Article • 05/23/2023

In this example, we use Kernel SHAP to explain a tabular classification model built from the Adults Census dataset.

First we import the packages and define some UDFs we need later.

Python

```
import pyspark
from synapse.ml.explainers import *
from pyspark.ml import Pipeline
from pyspark.ml.classification import LogisticRegression
from pyspark.ml.feature import StringIndexer, OneHotEncoder, VectorAssembler
from pyspark.sql.types import *
from pyspark.sql.functions import *
import pandas as pd
from pyspark.sql import SparkSession

# Bootstrap Spark Session
spark = SparkSession.builder.getOrCreate()

from synapse.ml.core.platform import *

vec_access = udf(lambda v, i: float(v[i]), FloatType())
vec2array = udf(lambda vec: vec.toArray().tolist(), ArrayType(FloatType()))
```

Now let's read the data and train a binary classification model.

Python

```
df = spark.read.parquet(
    "wasbs://publicwasb@mmlspark.blob.core.windows.net/AdultCensusIncome.parquet"
)

labelIndexer = StringIndexer(
    inputCol="income", outputCol="label", stringOrderType="alphabetAsc"
).fit(df)
print("Label index assigment: " + str(set(zip(labelIndexer.labels, [0, 1]))))

training = labelIndexer.transform(df).cache()
display(training)
categorical_features = [
```

```

    "workclass",
    "education",
    "marital-status",
    "occupation",
    "relationship",
    "race",
    "sex",
    "native-country",
]
categorical_features_idx = [col + "_idx" for col in categorical_features]
categorical_features_enc = [col + "_enc" for col in categorical_features]
numeric_features = [
    "age",
    "education-num",
    "capital-gain",
    "capital-loss",
    "hours-per-week",
]
strIndexer = StringIndexer(
    inputCols=categorical_features, outputCols=categorical_features_idx
)
onehotEnc = OneHotEncoder(
    inputCols=categorical_features_idx, outputCols=categorical_features_enc
)
vectAssem = VectorAssembler(
    inputCols=categorical_features_enc + numeric_features,
    outputCol="features"
)
lr = LogisticRegression(featuresCol="features", labelCol="label",
weightCol="fnlwgt")
pipeline = Pipeline(stages=[strIndexer, onehotEnc, vectAssem, lr])
model = pipeline.fit(training)

```

After the model is trained, we randomly select some observations to be explained.

Python

```

explain_instances = (
    model.transform(training).orderBy(rand()).limit(5).repartition(200).cache()
)
display(explain_instances)

```

We create a TabularSHAP explainer, set the input columns to all the features the model takes, specify the model and the target output column we're trying to explain. In this case, we're trying to explain the "probability" output, which is a vector of length 2, and we're only looking at class 1 probability. Specify targetClasses to `[0, 1]` if you want to explain class 0 and 1 probability at the same time. Finally we sample 100 rows from the

training data for background data, which is used for integrating out features in Kernel SHAP.

Python

```
shap = TabularSHAP(  
    inputCols=categorical_features + numeric_features,  
    outputCol="shapValues",  
    numSamples=5000,  
    model=model,  
    targetCol="probability",  
    targetClasses=[1],  
    backgroundData=broadcast(training.orderBy(rand()).limit(100).cache()),  
)  
  
shap_df = shap.transform(explain_instances)
```

Once we have the resulting dataframe, we extract the class 1 probability of the model output, the SHAP values for the target class, the original features and the true label. Then we convert it to a pandas dataframe for visualization. For each observation, the first element in the SHAP values vector is the base value (the mean output of the background dataset), and each of the following element is the SHAP values for each feature.

Python

```
shaps = (  
    shap_df.withColumn("probability", vec_access(col("probability"),  
lit(1)))  
    .withColumn("shapValues", vec2array(col("shapValues").getItem(0)))  
    .select(  
        ["shapValues", "probability", "label"] + categorical_features +  
        numeric_features  
    )  
)  
  
shaps_local = shaps.toPandas()  
shaps_local.sort_values("probability", ascending=False, inplace=True,  
ignore_index=True)  
pd.set_option("display.max_colwidth", None)  
shaps_local
```

We use plotly subplot to visualize the SHAP values.

Python

```
from plotly.subplots import make_subplots  
import plotly.graph_objects as go  
import pandas as pd
```

```

features = categorical_features + numeric_features
features_with_base = ["Base"] + features

rows = shaps_local.shape[0]

fig = make_subplots(
    rows=rows,
    cols=1,
    subplot_titles="Probability: "
    + shaps_local["probability"].apply("{:.2%}".format)
    + "; Label: "
    + shaps_local["label"].astype(str),
)

for index, row in shaps_local.iterrows():
    feature_values = [0] + [row[feature] for feature in features]
    shap_values = row["shapValues"]
    list_of_tuples = list(zip(features_with_base, feature_values,
shap_values))
    shap_pdf = pd.DataFrame(list_of_tuples, columns=["name", "value",
"shap"])
    fig.add_trace(
        go.Bar(
            x=shap_pdf["name"],
            y=shap_pdf["shap"],
            hovertext="value: " + shap_pdf["value"].astype(str),
        ),
        row=index + 1,
        col=1,
    )

fig.update_yaxes(range=[-1, 1], fixedrange=True, zerolinecolor="black")
fig.update_xaxes(type="category", tickangle=45, fixedrange=True)
fig.update_layout(height=400 * rows, title_text="SHAP explanations")
fig.show()

```

## Next steps

- How to use Kernel SHAP to explain a tabular classification model
- How to use SynapseML for multivariate anomaly detection
- How to Build a Search Engine with SynapseML

# Recipe: Multivariate Anomaly Detection with Isolation Forest

Article • 05/23/2023

This recipe shows how you can use SynapseML on Apache Spark for multivariate anomaly detection. Multivariate anomaly detection allows for the detection of anomalies among many variables or timeseries, taking into account all the inter-correlations and dependencies between the different variables. In this scenario, we use SynapseML to train an Isolation Forest model for multivariate anomaly detection, and we then use to the trained model to infer multivariate anomalies within a dataset containing synthetic measurements from three IoT sensors.

To learn more about the Isolation Forest model, refer to the original paper by [Liu et al.](#).

## Prerequisites

- Attach your notebook to a lakehouse. On the left side, select **Add** to add an existing lakehouse or create a lakehouse.

## Library imports

Python

```
from IPython import get_ipython
from IPython.terminal.interactiveshell import TerminalInteractiveShell
import uuid
import mlflow

from pyspark.sql import functions as F
from pyspark.ml.feature import VectorAssembler
from pyspark.sql.types import *
from pyspark.ml import Pipeline

from synapse.ml.isolationforest import *
from synapse.ml.explainers import *
```

Python

```
%matplotlib inline
```

Python

```
from pyspark.sql import SparkSession

# Bootstrap Spark Session
spark = SparkSession.builder.getOrCreate()

from synapse.ml.core.platform import *

if running_on_synapse():
    shell = TerminalInteractiveShell.instance()
    shell.define_macro("foo", """a,b=10,20""")
```

## Input data

Python

```
# Table inputs
timestampColumn = "timestamp" # str: the name of the timestamp column in
the table
inputCols = [
    "sensor_1",
    "sensor_2",
    "sensor_3",
] # list(str): the names of the input variables

# Training Start time, and number of days to use for training:
trainingStartTime = (
    "2022-02-24T06:00:00Z" # datetime: datetime for when to start the
training
)
trainingEndTime = (
    "2022-03-08T23:55:00Z" # datetime: datetime for when to end the
training
)
inferenceStartTime = (
    "2022-03-09T09:30:00Z" # datetime: datetime for when to start the
training
)
inferenceEndTime = (
    "2022-03-20T23:55:00Z" # datetime: datetime for when to end the
training
)

# Isolation Forest parameters
contamination = 0.021
num_estimators = 100
max_samples = 256
max_features = 1.0
```

# Read data

Python

```
df = (
    spark.read.format("csv")
    .option("header", "true")
    .load(
        "wasbs://publicwasb@mmlspark.blob.core.windows.net/generated_sample_mvad_dat
a.csv"
    )
)
```

cast columns to appropriate data types

Python

```
df = (
    df.orderBy(timestampColumn)
    .withColumn("timestamp", F.date_format(timestampColumn, "yyyy-MM-
dd'T'HH:mm:ss'Z'"))
    .withColumn("sensor_1", F.col("sensor_1").cast(DoubleType()))
    .withColumn("sensor_2", F.col("sensor_2").cast(DoubleType()))
    .withColumn("sensor_3", F.col("sensor_3").cast(DoubleType()))
    .drop("_c5")
)

display(df)
```

# Training data preparation

Python

```
# filter to data with timestamps within the training window
df_train = df.filter(
    (F.col(timestampColumn) >= trainingStartTime)
    & (F.col(timestampColumn) <= trainingEndTime)
)
display(df_train)
```

# Test data preparation

Python

```
# filter to data with timestamps within the inference window
df_test = df.filter(
    (F.col(timestampColumn) >= inferenceStartTime)
    & (F.col(timestampColumn) <= inferenceEndTime)
)
display(df_test)
```

## Train Isolation Forest model

Python

```
isolationForest = (
    IsolationForest()
    .setNumEstimators(num_estimators)
    .setBootstrap(False)
    .setMaxSamples(max_samples)
    .setMaxFeatures(max_features)
    .setFeaturesCol("features")
    .setPredictionCol("predictedLabel")
    .setScoreCol("outlierScore")
    .setContamination(contamination)
    .setContaminationError(0.01 * contamination)
    .setRandomSeed(1)
)
```

Next, we create an ML pipeline to train the Isolation Forest model. We also demonstrate how to create an MLflow experiment and register the trained model.

MLflow model registration is strictly only required if accessing the trained model at a later time. For training the model, and performing inferencing in the same notebook, the model object model is sufficient.

Python

```
va = VectorAssembler(inputCols=inputCols, outputCol="features")
pipeline = Pipeline(stages=[va, isolationForest])
model = pipeline.fit(df_train)
```

## Perform inferencing

Load the trained Isolation Forest Model

Perform inferencing

Python

```
df_test_pred = model.transform(df_test)
display(df_test_pred)
```

## Next steps

- How to Build a Search Engine with SynapseML
- How to use SynapseML and Cognitive Services for multivariate anomaly detection - Analyze time series
- How to use SynapseML to tune hyperparameters

# Tutorial: Create a custom search engine and question-answering system

Article • 05/23/2023

In this tutorial, learn how to index and query large data loaded from a Spark cluster. You set up a Jupyter Notebook that performs the following actions:

- Load various forms (invoices) into a data frame in an Apache Spark session
- Analyze them to determine their features
- Assemble the resulting output into a tabular data structure
- Write the output to a search index hosted in Azure Cognitive Search
- Explore and query over the content you created

## 1 - Set up dependencies

We start by importing packages and connecting to the Azure resources used in this workflow.

Python

```
import os
from pyspark.sql import SparkSession
from synapse.ml.core.platform import running_on_synapse, find_secret

# Bootstrap Spark Session
spark = SparkSession.builder.getOrCreate()

cognitive_key = find_secret("cognitive-api-key") # replace with your
cognitive api key
cognitive_location = "eastus"

translator_key = find_secret("translator-key") # replace with your cognitive
api key
translator_location = "eastus"

search_key = find_secret("azure-search-key") # replace with your cognitive
api key
search_service = "mmlspark-azure-search"
search_index = "form-demo-index-5"

openai_key = find_secret("openai-api-key") # replace with your open ai api
key
openai_service_name = "synapseml-openai"
openai_deployment_name = "gpt-35-turbo"
openai_url = f"https://{{openai_service_name}}.openai.azure.com/"
```

## 2 - Load data into Spark

This code loads a few external files from an Azure storage account that's used for demo purposes. The files are various invoices, and they're read into a data frame.

Python

```
from pyspark.sql.functions import udf
from pyspark.sql.types import StringType

def blob_to_url(blob):
    [prefix, postfix] = blob.split("@")
    container = prefix.split("/")[-1]
    split_postfix = postfix.split("/")
    account = split_postfix[0]
    filepath = "/".join(split_postfix[1:])
    return "https://{}.{}/{}".format(account, container, filepath)

df2 = (
    spark.read.format("binaryFile")

    .load("wasbs://ignite2021@mmlsparkdemo.blob.core.windows.net/form_subset/*")
    .select("path")
    .limit(10)
    .select(udf(blob_to_url, StringType())("path").alias("url"))
    .cache()
)
display(df2)
```

## 3 - Apply form recognition

This code loads the [AnalyzeInvoices transformer](#) and passes a reference to the data frame containing the invoices. It calls the pre-built invoice model of Azure Forms Analyzer.

Python

```
from synapse.ml.cognitive import AnalyzeInvoices

analyzed_df = (
    AnalyzeInvoices()
    .setSubscriptionKey(cognitive_key)
    .setLocation(cognitive_location)
    .setImageUrlCol("url")
    .setOutputCol("invoices")
    .setErrorCol("errors")
```

```
.setConcurrency(5)
.transform(df2)
.cache()
)

display(analyzed_df)
```

## 4 - Simplify form recognition output

This code uses the [FormOntologyLearner](#), a transformer that analyzes the output of Form Recognizer transformers and infers a tabular data structure. The output of AnalyzeInvoices is dynamic and varies based on the features detected in your content.

FormOntologyLearner extends the utility of the AnalyzeInvoices transformer by looking for patterns that can be used to create a tabular data structure. Organizing the output into multiple columns and rows makes for simpler downstream analysis.

Python

```
from synapse.ml.cognitive import FormOntologyLearner

organized_df = (
    FormOntologyLearner()
    .setInputCol("invoices")
    .setOutputCol("extracted")
    .fit(analyzed_df)
    .transform(analyzed_df)
    .select("url", "extracted.*")
    .cache()
)

display(organized_df)
```

With our nice tabular dataframe, we can flatten the nested tables found in the forms with some SparkSQL

Python

```
from pyspark.sql.functions import explode, col

itemized_df = (
    organized_df.select("*", explode(col("Items")).alias("Item"))
    .drop("Items")
    .select("Item.*", "*")
    .drop("Item")
)

display(itemized_df)
```

## 5 - Add translations

This code loads [Translate](#), a transformer that calls the Azure Translator service in Cognitive Services. The original text, which is in English in the "Description" column, is machine-translated into various languages. All of the output is consolidated into "output.translations" array.

Python

```
from synapse.ml.cognitive import Translate

translated_df = (
    Translate()
    .setSubscriptionKey(translator_key)
    .setLocation(translator_location)
    .setTextCol("Description")
    .setErrorCol("TranslationError")
    .setOutputCol("output")
    .setToLanguage(["zh-Hans", "fr", "ru", "cy"])
    .setConcurrency(5)
    .transform(itemized_df)
    .withColumn("Translations", col("output.translations")[0])
    .drop("output", "TranslationError")
    .cache()
)

display(translated_df)
```

## 6 - Translate products to emojis with OpenAI ☐

Python

```
from synapse.ml.cognitive.openai import OpenAIPrompt
from pyspark.sql.functions import trim, split

emoji_template = """
    Your job is to translate item names into emoji. Do not add anything but
    the emoji and end the translation with a comma

    Two Ducks: 🦆🦆,
    Light Bulb: 💡,
    Three Peaches: 🍑🍑🍑,
    Two kitchen stoves: 🔥🔥,
    A red car: 🚗,
    A person and a cat: 🧑🐱,
    A {Description}: """
```

```
prompter = (
    OpenAIPrompt()
    .setSubscriptionKey(openai_key)
    .setDeploymentName(openai_deployment_name)
    .setUrl(openai_url)
    .setMaxTokens(5)
    .setPromptTemplate(emoji_template)
    .setErrorCol("error")
    .setOutputCol("Emoji")
)

emoji_df = (
    prompter.transform(translated_df)
    .withColumn("Emoji", trim(split(col("Emoji"), ",").getItem(0)))
    .drop("error", "prompt")
    .cache()
)
```

Python

```
display(emoji_df.select("Description", "Emoji"))
```

## 7 - Infer vendor address continent with OpenAI

Python

```
continent_template = """
Which continent does the following address belong to?

Pick one value from Europe, Australia, North America, South America, Asia,
Africa, Antarctica.
```

Dont respond with anything but one of the above. If you don't know the answer or cannot figure it out from the text, return None. End your answer with a comma.

```
Address: "6693 Ryan Rd, North Whales",
Continent: Europe,
Address: "6693 Ryan Rd",
Continent: None,
Address: "{VendorAddress}",
Continent:"""
```

```
continent_df = (
    prompter.setOutputCol("Continent")
    .setPromptTemplate(continent_template)
    .transform(emoji_df)
    .withColumn("Continent", trim(split(col("Continent"), ",").getItem(0)))
    .drop("error", "prompt")
```

```
.cache()  
)
```

Python

```
display(continent_df.select("VendorAddress", "Continent"))
```

## 8 - Create an Azure Search Index for the Forms

Python

```
from synapse.ml.cognitive import *  
from pyspark.sql.functions import monotonically_increasing_id, lit  
  
(  
    continent_df.withColumn("DocID",  
    monotonically_increasing_id().cast("string"))  
    .withColumn("SearchAction", lit("upload"))  
    .writeToAzureSearch(  
        subscriptionKey=search_key,  
        actionCol="SearchAction",  
        serviceName=search_service,  
        indexName=search_index,  
        keyCol="DocID",  
    )  
)
```

## 9 - Try out a search query

Python

```
import requests  
  
search_url = "https://{}.search.windows.net/indexes/{}/docs/search?api-  
version=2019-05-06".format(  
    search_service, search_index  
)  
requests.post(  
    search_url, json={"search": "door"}, headers={"api-key": search_key}  
).json()
```

## 10 - Build a chatbot that can use Azure Search as a tool ☰

Python

```
import json
import openai

openai.api_type = "azure"
openai.api_base = openai_url
openai.api_key = openai_key
openai.api_version = "2023-03-15-preview"

chat_context_prompt = f"""
You are a chatbot designed to answer questions with the help of a search
engine that has the following information:

{continent_df.columns}

If you dont know the answer to a question say "I dont know". Do not lie or
hallucinate information. Be brief. If you need to use the search engine to
solve the please output a json in the form of {"query": "example_query"}}
"""

def search_query_prompt(question):
    return f"""
Given the search engine above, what would you search for to answer the
following question?

Question: "{question}"

Please output a json in the form of {"query": "example_query"}}
"""

def search_result_prompt(query):
    search_results = requests.post(
        search_url, json={"search": query}, headers={"api-key": search_key}
    ).json()
    return f"""

You previously ran a search for "{query}" which returned the following
results:

{search_results}

You should use the results to help you answer questions. If you dont know
the answer to a question say "I dont know". Do not lie or hallucinate
information. Be Brief and mention which query you used to solve the problem.
"""

def prompt_gpt(messages):
    response = openai.ChatCompletion.create(
        engine=openai_deployment_name, messages=messages, max_tokens=None,
        top_p=0.95
```

```

    )
    return response["choices"][0]["message"]["content"]

def custom_chatbot(question):
    while True:
        try:
            query = json.loads(
                prompt_gpt(
                    [
                        {"role": "system", "content": chat_context_prompt},
                        {"role": "user", "content": search_query_prompt(question)},
                    ]
                )
            )["query"]

            return prompt_gpt(
                [
                    {"role": "system", "content": chat_context_prompt},
                    {"role": "system", "content": search_result_prompt(query)},
                    {"role": "user", "content": question},
                ]
            )
        except Exception as e:
            raise e

```

## 11 - Asking our chatbot a question

Python

```
custom_chatbot("What did Luke Diaz buy?")
```

## 12 - A quick double check

Python

```

display(
    continent_df.where(col("CustomerName") == "Luke Diaz")
    .select("Description")
    .distinct()
)

```

## Next steps

- How to use LightGBM with SynapseML
- How to use SynapseML and Cognitive Services for multivariate anomaly detection - Analyze time series
- How to use SynapseML to tune hyperparameters

# Recipe: Cognitive Services - Multivariate Anomaly Detection

Article • 05/23/2023

This recipe shows how you can use SynapseML and Azure Cognitive Services on Apache Spark for multivariate anomaly detection. Multivariate anomaly detection allows for the detection of anomalies among many variables or timeseries, taking into account all the inter-correlations and dependencies between the different variables. In this scenario, we use SynapseML to train a model for multivariate anomaly detection using the Azure Cognitive Services, and we then use the model to infer multivariate anomalies within a dataset containing synthetic measurements from three IoT sensors.

To learn more about the Anomaly Detector Cognitive Service, refer to [this documentation page](#).

## Prerequisites

- An Azure subscription - [Create one for free ↗](#)
- Attach your notebook to a lakehouse. On the left side, select **Add** to add an existing lakehouse or create a lakehouse.

## Setup

### Create an Anomaly Detector resource

Follow the instructions to create an `Anomaly Detector` resource using the Azure portal or alternatively, you can also use the Azure CLI to create this resource.

- In the Azure portal, click `Create` in your resource group, and then type `Anomaly Detector`. Click on the Anomaly Detector resource.
- Give the resource a name, and ideally use the same region as the rest of your resource group. Use the default options for the rest, and then click `Review + Create` and then `Create`.
- Once the Anomaly Detector resource is created, open it and click on the `Keys and Endpoints` panel on the left. Copy the key for the Anomaly Detector resource into the `ANOMALY_API_KEY` environment variable, or store it in the `anomalyKey` variable.

# Create a Storage Account resource

In order to save intermediate data, you need to create an Azure Blob Storage Account. Within that storage account, create a container for storing the intermediate data. Make note of the container name, and copy the connection string to that container. You need it later to populate the `containerName` variable and the `BLOB_CONNECTION_STRING` environment variable.

## Enter your service keys

Let's start by setting up the environment variables for our service keys. The next cell sets the `ANOMALY_API_KEY` and the `BLOB_CONNECTION_STRING` environment variables based on the values stored in our Azure Key Vault. If you're running this tutorial in your own environment, make sure you set these environment variables before you proceed.

Python

```
import os
from pyspark.sql import SparkSession
from synapse.ml.core.platform import find_secret

# Bootstrap Spark Session
spark = SparkSession.builder.getOrCreate()
```

Now, lets read the `ANOMALY_API_KEY` and `BLOB_CONNECTION_STRING` environment variables and set the `containerName` and `location` variables.

Python

```
# An Anomaly Dectector subscription key
anomalyKey = find_secret("anomaly-api-key") # use your own anomaly api key
# Your storage account name
storageName = "anomalydetectiontest" # use your own storage account name
# A connection string to your blob storage account
storageKey = find_secret("madtest-storage-key") # use your own storage key
# A place to save intermediate MVAD results
intermediateSaveDir = (
    "wasbs://madtest@anomalydetectiontest.blob.core.windows.net/intermediateData"
)
# The location of the anomaly detector resource that you created
location = "westus2"
```

First we connect to our storage account so that anomaly detector can save intermediate results there:

Python

```
spark.sparkContext._jsc.hadoopConfiguration().set(
    "fs.azure.account.key.{storageName}.blob.core.windows.net", storageKey
)
```

Let's import all the necessary modules.

Python

```
import numpy as np
import pandas as pd

import pyspark
from pyspark.sql.functions import col
from pyspark.sql.functions import lit
from pyspark.sql.types import DoubleType
import matplotlib.pyplot as plt

import synapse.ml
from synapse.ml.cognitive import *
```

Now, let's read our sample data into a Spark DataFrame.

Python

```
df = (
    spark.read.format("csv")
    .option("header", "true")

    .load("wasbs://publicwasb@mmlspark.blob.core.windows.net/MVAD/sample.csv")
)

df = (
    df.withColumn("sensor_1", col("sensor_1").cast(DoubleType()))
    .withColumn("sensor_2", col("sensor_2").cast(DoubleType()))
    .withColumn("sensor_3", col("sensor_3").cast(DoubleType()))
)

# Let's inspect the dataframe:
df.show(5)
```

We can now create an `estimator` object, which is used to train our model. We specify the start and end times for the training data. We also specify the input columns to use, and the name of the column that contains the timestamps. Finally, we specify the number of data points to use in the anomaly detection sliding window, and we set the connection string to the Azure Blob Storage Account.

Python

```
trainingStartTime = "2020-06-01T12:00:00Z"
trainingEndTime = "2020-07-02T17:55:00Z"
timestampColumn = "timestamp"
inputColumns = ["sensor_1", "sensor_2", "sensor_3"]

estimator = (
    FitMultivariateAnomaly()
    .setSubscriptionKey(anomalyKey)
    .setLocation(location)
    .setStartTime(trainingStartTime)
    .setEndTime(trainingEndTime)
    .setIntermediateSaveDir(intermediateSaveDir)
    .setTimestampCol(timestampColumn)
    .setInputCols(inputColumns)
    .setSlidingWindow(200)
)
```

Now that we have created the `estimator`, let's fit it to the data:

Python

```
model = estimator.fit(df)
```

Once the training is done, we can now use the model for inference. The code in the next cell specifies the start and end times for the data we would like to detect the anomalies in.

Python

```
inferenceStartTime = "2020-07-02T18:00:00Z"
inferenceEndTime = "2020-07-06T05:15:00Z"

result = (
    model.setStartTime(inferenceStartTime)
    .setEndTime(inferenceEndTime)
    .setOutputCol("results")
    .setErrorCol("errors")
    .setInputCols(inputColumns)
    .setTimestampCol(timestampColumn)
    .transform(df)
)

result.show(5)
```

When we called `.show(5)` in the previous cell, it showed us the first five rows in the dataframe. The results were all `null` because they weren't inside the inference window.

To show the results only for the inferred data, lets select the columns we need. We can then order the rows in the dataframe by ascending order, and filter the result to only show the rows that are in the range of the inference window. In our case `inferenceEndTime` is the same as the last row in the dataframe, so can ignore that.

Finally, to be able to better plot the results, lets convert the Spark dataframe to a Pandas dataframe.

Python

```
rdf = (
    result.select(
        "timestamp",
        *inputColumns,
        "results.contributors",
        "results.isAnomaly",
        "results.severity"
    )
    .orderBy("timestamp", ascending=True)
    .filter(col("timestamp") >= lit(inferenceStartTime))
    .toPandas()
)

rdf
```

Let's now formatted the `contributors` column that stores the contribution score from each sensor to the detected anomalies. The next cell formats this data, and splits the contribution score of each sensor into its own column.

Python

```
def parse(x):
    if type(x) is list:
        return dict([item[::-1] for item in x])
    else:
        return {"series_0": 0, "series_1": 0, "series_2": 0}

rdf["contributors"] = rdf["contributors"].apply(parse)
rdf = pd.concat([
    rdf.drop(["contributors"], axis=1),
    pd.json_normalize(rdf["contributors"]),
], axis=1)
```

Great! We now have the contribution scores of sensors 1, 2, and 3 in the `series_0`, `series_1`, and `series_2` columns respectively.

Let's run the next cell to plot the results. The `minSeverity` parameter in the first line specifies the minimum severity of the anomalies to be plotted.

Python

```
minSeverity = 0.1

##### Main Figure #####
plt.figure(figsize=(23, 8))
plt.plot(
    rdf["timestamp"],
    rdf["sensor_1"],
    color="tab:orange",
    linestyle="solid",
    linewidth=2,
    label="sensor_1",
)
plt.plot(
    rdf["timestamp"],
    rdf["sensor_2"],
    color="tab:green",
    linestyle="solid",
    linewidth=2,
    label="sensor_2",
)
plt.plot(
    rdf["timestamp"],
    rdf["sensor_3"],
    color="tab:blue",
    linestyle="solid",
    linewidth=2,
    label="sensor_3",
)
plt.grid(axis="y")
plt.tick_params(axis="x", which="both", bottom=False, labelbottom=False)
plt.legend()

anoms = list(rdf["severity"] >= minSeverity)
_, _, ymin, ymax = plt.axis()
plt.vlines(np.where(anoms), ymin=ymin, ymax=ymax, color="r", alpha=0.8)

plt.legend()
plt.title(
    "A plot of the values from the three sensors with the detected anomalies highlighted in red."
)
plt.show()

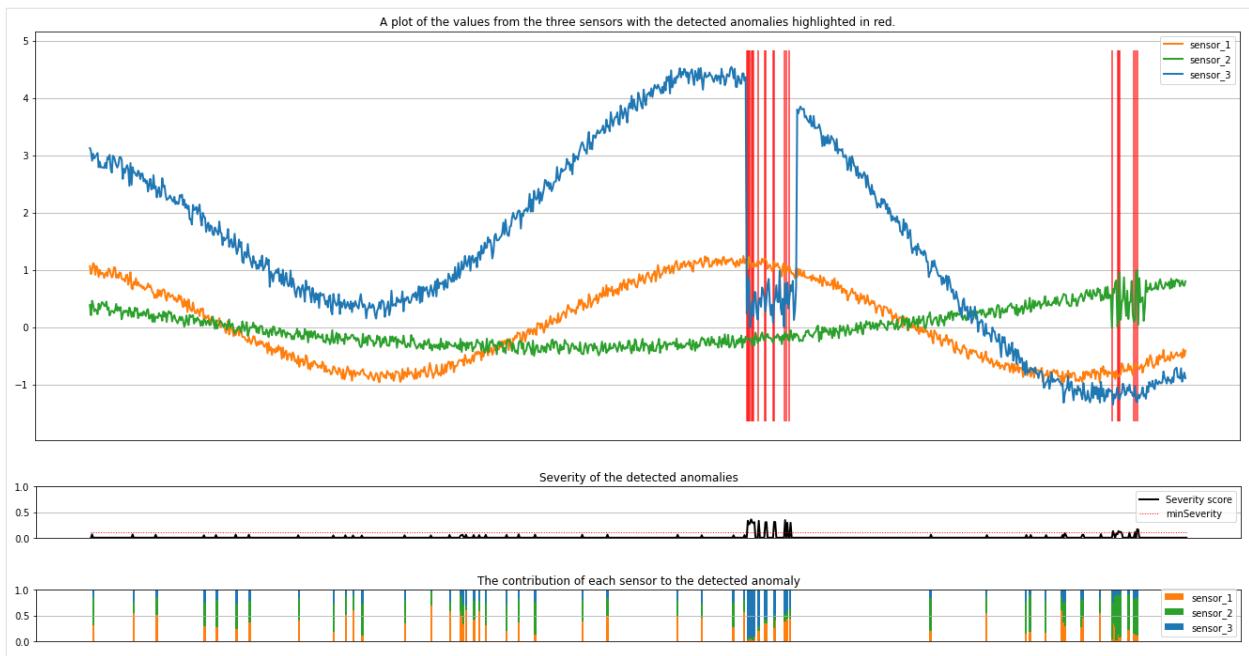
##### Severity Figure #####
plt.figure(figsize=(23, 1))
plt.tick_params(axis="x", which="both", bottom=False, labelbottom=False)
plt.plot(
    rdf["timestamp"],
```

```

        rdf["severity"],
        color="black",
        linestyle="solid",
        linewidth=2,
        label="Severity score",
    )
plt.plot(
    rdf["timestamp"],
    [minSeverity] * len(rdf["severity"]),
    color="red",
    linestyle="dotted",
    linewidth=1,
    label="minSeverity",
)
plt.grid(axis="y")
plt.legend()
plt.ylim([0, 1])
plt.title("Severity of the detected anomalies")
plt.show()

##### Contributors Figure #####
plt.figure(figsize=(23, 1))
plt.tick_params(axis="x", which="both", bottom=False, labelbottom=False)
plt.bar(
    rdf["timestamp"], rdf["series_0"], width=2, color="tab:orange",
    label="sensor_1"
)
plt.bar(
    rdf["timestamp"],
    rdf["series_1"],
    width=2,
    color="tab:green",
    label="sensor_2",
    bottom=rdf["series_0"],
)
plt.bar(
    rdf["timestamp"],
    rdf["series_2"],
    width=2,
    color="tab:blue",
    label="sensor_3",
    bottom=rdf["series_0"] + rdf["series_1"],
)
plt.grid(axis="y")
plt.legend()
plt.ylim([0, 1])
plt.title("The contribution of each sensor to the detected anomaly")
plt.show()

```



The plots show the raw data from the sensors (inside the inference window) in orange, green, and blue. The red vertical lines in the first figure show the detected anomalies that have a severity greater than or equal to `minSeverity`.

The second plot shows the severity score of all the detected anomalies, with the `minSeverity` threshold shown in the dotted red line.

Finally, the last plot shows the contribution of the data from each sensor to the detected anomalies. It helps us diagnose and understand the most likely cause of each anomaly.

## Next steps

- How to use LightGBM with SynapseML
- How to use Cognitive Services with SynapseML
- How to use SynapseML to tune hyperparameters

# HyperParameterTuning - Fighting Breast Cancer

Article • 05/23/2023

This tutorial shows how SynapseML can be used to identify the best combination of hyperparameters for your chosen classifiers, ultimately resulting in more accurate and reliable models. In order to demonstrate this, we'll show how to perform distributed randomized grid search hyperparameter tuning to build a model to identify breast cancer.

## 1 - Set up dependencies

Start by importing pandas and setting up our Spark session.

Python

```
import pandas as pd
from pyspark.sql import SparkSession

# Bootstrap Spark Session
spark = SparkSession.builder.getOrCreate()
```

Next, read the data and split it into tuning and test sets.

Python

```
data = spark.read.parquet(
    "wasbs://publicwasb@mmlspark.blob.core.windows.net/BreastCancer.parquet"
).cache()
tune, test = data.randomSplit([0.80, 0.20])
tune.limit(10).toPandas()
```

Define the models to be used.

Python

```
from synapse.ml.automl import TuneHyperparameters
from synapse.ml.train import TrainClassifier
from pyspark.ml.classification import (
    LogisticRegression,
    RandomForestClassifier,
    GBTClassifier,
)
```

```
logReg = LogisticRegression()
randForest = RandomForestClassifier()
gbt = GBTClassifier()
smlmodels = [logReg, randForest, gbt]
mmlmodels = [TrainClassifier(model=model, labelCol="Label") for model in
smlmodels]
```

## 2 - Find the best model using AutoML

Import SynapseML's AutoML classes from `synapse.ml.automl`. Specify the hyperparameters using the `HyperparamBuilder`. Add either `DiscreteHyperParam` or `RangeHyperParam` hyperparameters. `TuneHyperparameters` will randomly choose values from a uniform distribution:

Python

```
from synapse.ml.automl import *

paramBuilder = (
    HyperparamBuilder()
    .addHyperparam(logReg, logReg.regParam, RangeHyperParam(0.1, 0.3))
    .addHyperparam(randForest, randForest.numTrees, DiscreteHyperParam([5,
10]))
    .addHyperparam(randForest, randForest.maxDepth, DiscreteHyperParam([3,
5]))
    .addHyperparam(gbt, gbt.maxBins, RangeHyperParam(8, 16))
    .addHyperparam(gbt, gbt.maxDepth, DiscreteHyperParam([3, 5]))
)
searchSpace = paramBuilder.build()
# The search space is a list of params to tuples of estimator and hyperparam
print(searchSpace)
randomSpace = RandomSpace(searchSpace)
```

Next, run `TuneHyperparameters` to get the best model.

Python

```
bestModel = TuneHyperparameters(
    evaluationMetric="accuracy",
    models=mmlmodels,
    numFolds=2,
    numRuns=len(mmlmodels) * 2,
    parallelism=1,
    paramSpace=randomSpace.space(),
    seed=0,
).fit(tune)
```

## 3 - Evaluate the model

We can view the best model's parameters and retrieve the underlying best model pipeline

```
Python
```

```
print(bestModel.getBestModelInfo())
print(bestModel.getBestModel())
```

We can score against the test set and view metrics.

```
Python
```

```
from synapse.ml.train import ComputeModelStatistics

prediction = bestModel.transform(test)
metrics = ComputeModelStatistics().transform(prediction)
metrics.limit(10).toPandas()
```

## Next steps

- [How to use LightGBM with SynapseML](#)
- [How to use Cognitive Services with SynapseML](#)
- [How to perform the same classification task with and without SynapseML](#)