**AZURE DATA BRICKS**

**NOTEBOOK-1:**

**[https://adb-7749372881626709.9.azuredatabricks.net/?o=7749372881626709#notebook/3176001210207604](https://adb-7749372881626709.9.azuredatabricks.net/?o=7749372881626709" \l "notebook/3176001210207604)**

**NOTEBOOK-2:**

[**https://adb-7749372881626709.9.azuredatabricks.net/?o=7749372881626709#notebook/300518369860453**](https://adb-7749372881626709.9.azuredatabricks.net/?o=7749372881626709#notebook/300518369860453)

**26/09/2023:**

[Azure DataBricks Training - Session 1.mp4](https://modakanalytics0.sharepoint.com/:v:/r/sites/ModakTraining/Shared%20Documents/Training%20Data/Training%20Videos/Azure%20DataBricks%20Training/Azure%20DataBricks%20Training%20-%20Session%201.mp4?csf=1&web=1&e=TGBbnX)

**Catalog:**

* Login to Azure Databricks page.
* Click on **Catalog.**
* We have below folders in the catalog page.

**A screenshot of a computer

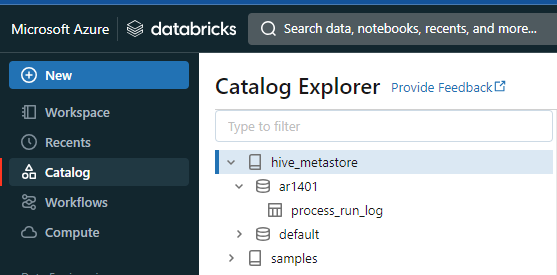
Description automatically generated**

* In each folder we have the **databases.**

**A screenshot of a computer

Description automatically generated**

* Under each database, we have **tables.**

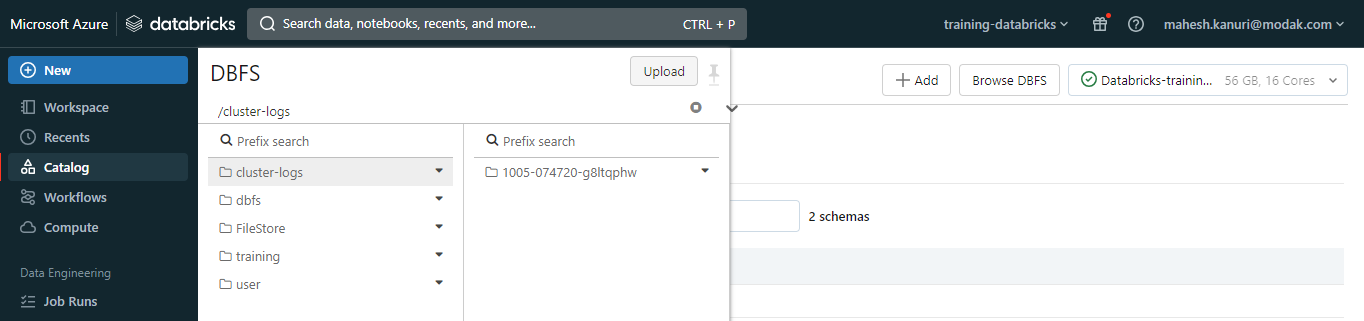


**File System in Databricks:**

* We have a file system for databricks called **DBFS (Data Bricks File System).**

****

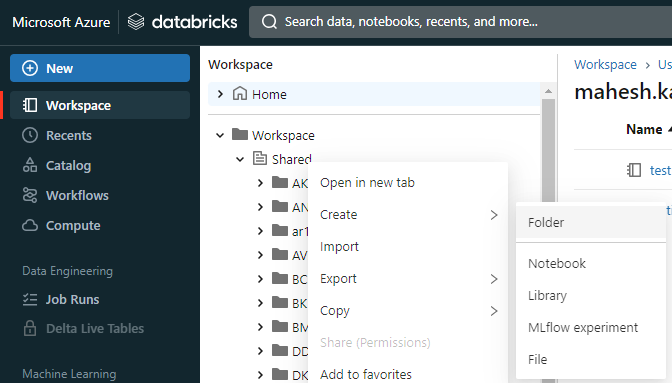
* Click on **Browse DBFS**

****

* We have some folders right, in there we will upload the files.
* If we click on **Upload,** by default it will select the **FileStore** folder to upload the files.

**Notebooks:**

* Click on **Workspace.**
* Right Click on **Shared** and create one **Folder.**

****

* Folder named as **MK2307**

**A white background with blue lines

Description automatically generated**

* Now inside the folder, create one **Notebook** named as **MY NOTEBOOK.** And then connect the available Cluster.

**A screen shot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**How to run the jobs in Notebook:**

* Let us take an example.
* Here I am loading the **CSV** file present in **DBFS** to dataframe.
* First of all, we need to find the path of the file present in **dbfs.**

**A screenshot of a computer

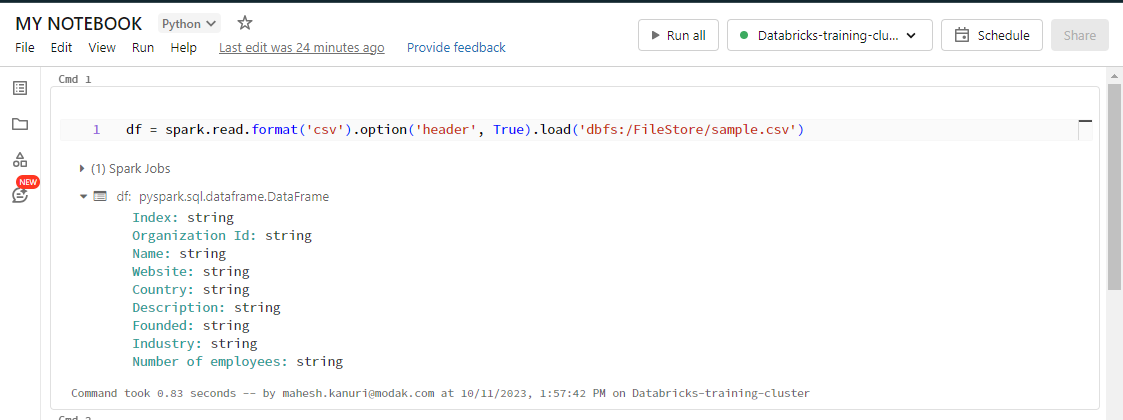
Description automatically generated**

**A screenshot of a computer

Description automatically generated**

* Path of the file: **dbfs:/FileStore/sample.csv**
* Now write the code.

df = spark.read.format('csv').option('header', True).load('dbfs:/FileStore/sample.csv')

****

* Display the Dataframe.

df.display()

**A screenshot of a computer

Description automatically generated**

* This is not a temporary table. To create a temporary table, run the below command.

df.createOrReplaceTempView('Sample')

* Now in notebook, change the language from **python** to **sql.**

****

* Run the query **select \* from Sample;** to display the content in the table.

**A screenshot of a computer

Description automatically generated**

* To display the Schema.

**A white screen with black text

Description automatically generated**

* This table is available until the session got restarted.
* Once the Cluster got restarted, the table got deleted.

**Difference between Default and other Databases in Catalog:**

* In the **hive\_metastore** catalog, we have two databases. One is **default** and another is **ar1401.**

**A screenshot of a computer

Description automatically generated**

* In **default** database, we have the following tables.

**A screenshot of a computer

Description automatically generated**

* For the tables present in **default** database, it’s not mandatory to mention the **database** name.

**A screenshot of a computer

Description automatically generated**

**colors** is the table present in default database.

* But for the tables present in other databases, it is mandatory to mention the **database** name also.
* Below one is the table present in other database **ar1401.**

**A screenshot of a computer

Description automatically generated**

* Now run the query without mentioning the **database** name.

**A screenshot of a computer program

Description automatically generated**

It shows that **table or view cannot be found.**

* So we need to mention the **database** name for the tables not present in **default database.**

A screenshot of a computer

Description automatically generated

* Same for **Catalog** also. For now we have default catalog **hive\_metastore.** If we have another metastore, we need to mention the **catalog** name also while running the queries.

Select \* from <**catalog\_name>.<database\_name>.<table\_name>;**

**27/09/2023:**

[Azure DataBricks Training - Session 2.mp4](https://modakanalytics0.sharepoint.com/:v:/r/sites/ModakTraining/Shared%20Documents/Training%20Data/Training%20Videos/Azure%20DataBricks%20Training/Azure%20DataBricks%20Training%20-%20Session%202.mp4?csf=1&web=1&e=TYFXsM)

**06/10/2023:**

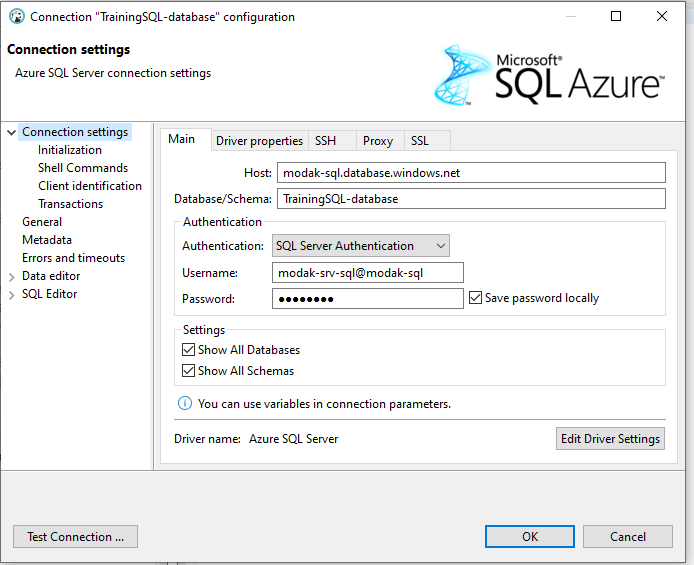
[Azure DataBricks Training - Session 3.mp4](https://modakanalytics0.sharepoint.com/:v:/r/sites/ModakTraining/Shared%20Documents/Training%20Data/Training%20Videos/Azure%20DataBricks%20Training/Azure%20DataBricks%20Training%20-%20Session%203.mp4?csf=1&web=1&e=A7O75k)

**SQL DATABASES:**

* Platform team provided the below Connection String for the **Azure SQL Database.**

**Connection String:** jdbc:sqlserver://modak-sql.database.windows.net:1433;database=TrainingSQL-database;user=modak-srv-sql@modak-sql;password=dW5n@o96;encrypt=true;trustServerCertificate=false;hostNameInCertificate=\*.database.windows.net;loginTimeout=30;

* Turn on the **Modak VPN,** using the above connection string, we will connect to the database using **DBeaver** as Client.



* If it was not connected, check the **encrypt=true, trustServerCertificate=false, hostNameInCertificate=\*.database.windows.net;loginTimeout=30** in **Driver Properties.**
* Because the above details are mentioned in the **Connection String.**

**Containers:**

* Login to [**Azure Portal**](https://portal.azure.com/#home)
* Click on **trainingnew** storage account.
* Click on **Containers.**

A computer screen shot of a computer

Description automatically generated

**A screenshot of a computer

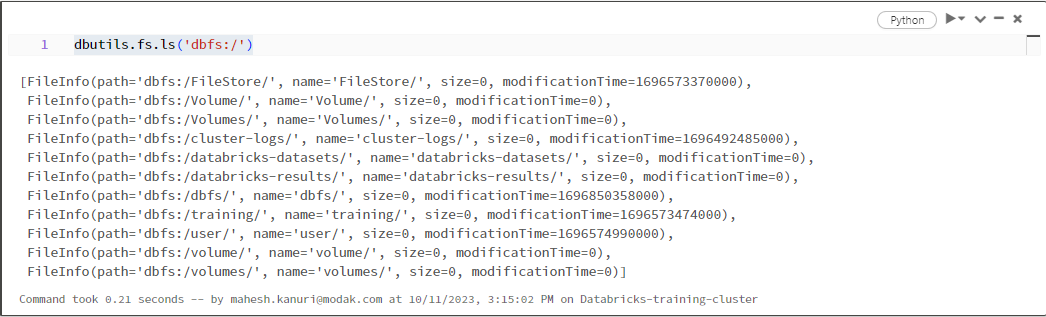
Description automatically generated**

* Open the **databricks-training** container and create directory. In this directory, we will store the files.

**Creating the directories in DBFS:**

* To display the folders present in the **DBFS,** run the below command

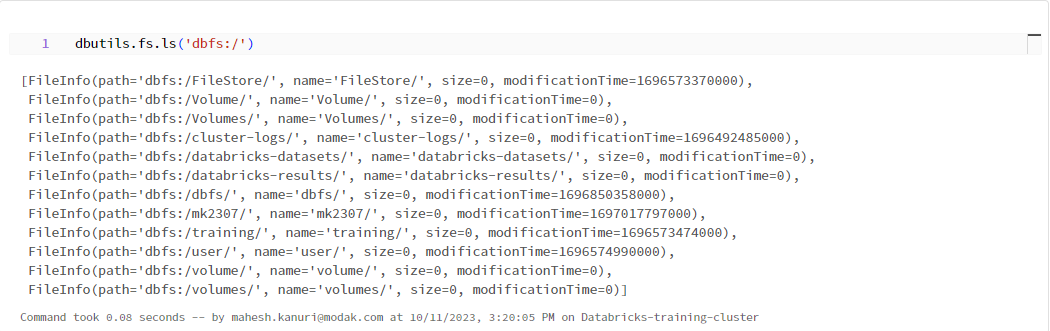
dbutils.fs.ls('dbfs:/')

****

* To create a folder in **DBFS,** run the below command

dbutils.fs.mkdirs('dbfs:/mk2307')

* Now check whether the directory was created or not by displaying all the directories.



* It was created. Now create another directory named **training** to store the files.

**Command:**

dbutils.fs.mkdirs('dbfs:/mk2307/training')

* Check whether the directory was created or not in **mk2307.**

**Command:**

dbutils.fs.ls('dbfs:/mk2307')

* Now I want to add one **CSV** file in **/mk2307/training** directory.
* But I am only able to add the files in **/FileStore** directory.
* So I have added the file in **/FileStore** directory and then selected that file and moved to **/mk2307/training** directory.

**InferSchema:**

* In Spark, **InferSchema** is a feature provided by the Spark DataFrame API, particularly when reading data from various sources like CSV, JSON, Parquet, and more.
* It's used to automatically infer the schema of the data without explicitly specifying it. Schema inference is particularly useful when you have data sources where the schema is not predefined or when you want to avoid manually defining the schema.
* Let’s take an example for **Reading Data from a file**.
* When you read data from a file using Spark (e.g., CSV, JSON), you can **enable schema inference** to **automatically detect the data types of the columns and create a schema for the DataFrame**.
* For example, when reading a CSV file with spark.read.csv("file.csv"), you can enable schema inference as follows:

**df = spark.read.option("inferSchema", "true").csv("file.csv")**

**Example to load data into Dataframe using a new file and option InferSchema:**

* Copied the file path.
* Run the below commands.

file\_path = 'dbfs:/mk2307/training/SampleCSVFile\_2kb.csv'

df = spark.read.format('csv').option('header',True).option('inferSchema', True).load(file\_path)

**A screen shot of a computer code

Description automatically generated**

* Display the data.

df.display()

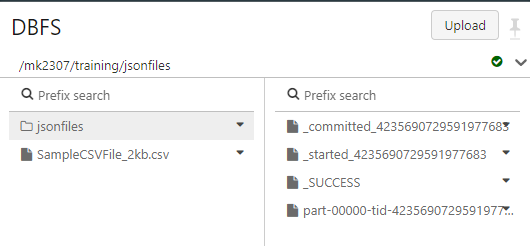
**A screenshot of a computer

Description automatically generated**

**Creating JSON files using Dataframe:**

* After loading the data into dataframe, append the data to a json file using below command

df.write.mode('append').format('json').save('dbfs:/mk2307/training/jsonfiles')



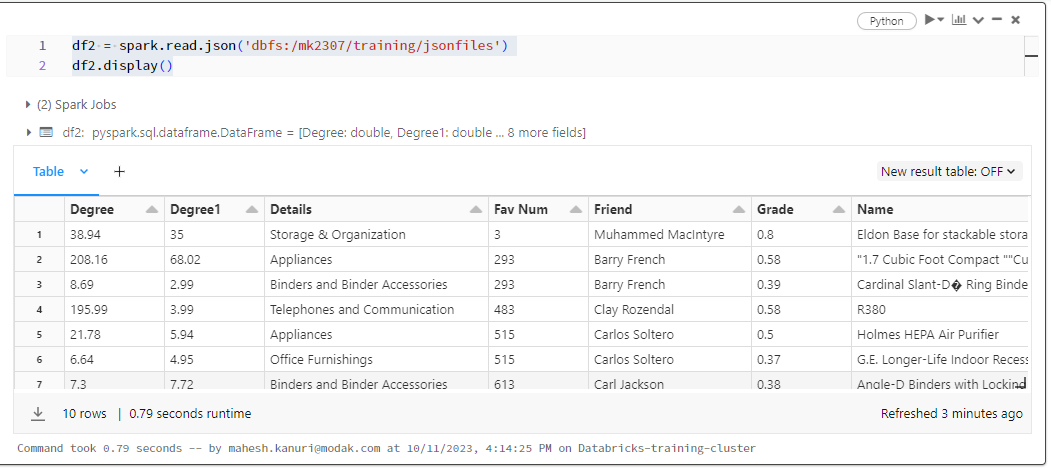
* Here one folder was created named **jsonfiles** and then the json file was created in that directory.
* We cannot give the name of the json files. After the json files was created, we are able to rename those files.
* Based on the **available cores,** the part files were created.

**Read the JSON Files:**

* For now, the JSON files have been created.
* To read all the JSON files, run the below commands.

df2 = spark.read.json('dbfs:/mk2307/training/jsonfiles')

df2.display()



**Creating/Overwriting the Tables in Catalog Databases:**

* I have taking another file.

**PATH:** dbfs:/training/st1406/color\_first.csv

* Run the below commands

#file path

file\_path = 'dbfs:/training/st1406/color\_first.csv'

#loading data into dataframe

df3 = spark.read.format('csv').option('header',True).option('inferSchema', True).load(file\_path)

#displaying data

df3.display()

#saving dataframe data into json files

df3.write.mode('overwrite').format('json').save('dbfs:/mk2307/training/jsonfiles2')

#writing data from json files to new dataframe

df4 = spark.read.format('json').option('multiLine', True).load('dbfs:/mk2307/training/jsonfiles2')

#displaying data

df4.display()

#creating temporary table. This table got deleted when the cluster got restarted.

df4.createOrReplaceTempView('jsontotable')

#creating a new table in default catalog database using the dataframe.

#Here we didn't mention any database name, so the table will got created in default database.

df4.write.mode('overwrite').saveAsTable('overwritedjsontotable')

* Here the table got created in default database.

A screenshot of a computer

Description automatically generated

**A screenshot of a computer

Description automatically generated**

**Creating Parquet files:**

* In the above process, we are saving the data into **JSON** files using below commands.

#file path

file\_path = 'dbfs:/training/st1406/color\_first.csv'

#loading data into dataframe

df3 = spark.read.format('csv').option('header',True).option('inferSchema', True).load(file\_path)

#displaying data

df3.display()

#saving dataframe data into json files

df3.write.mode('overwrite').format('json').save('dbfs:/mk2307/training/jsonfiles2')

* In the last command, just change the **json** to **parquet** and change the saving path like below.

#saving dataframe data into parquet files

df3.write.mode('overwrite').format(‘parquet’).save('dbfs:/mk2307/training/parquetfiles')

**A screenshot of a computer

Description automatically generated**

* The **parquet** files got created.

**09/10/2023:**

[Azure DataBricks Training - Session 4.mp4](https://modakanalytics0.sharepoint.com/:v:/r/sites/ModakTraining/Shared%20Documents/Training%20Data/Training%20Videos/Azure%20DataBricks%20Training/Azure%20DataBricks%20Training%20-%20Session%204.mp4?csf=1&web=1&e=X4liZe)

**Connecting to Azure SQL Database using Connection String:**

* We will connect to Azure SQL Database using Connection String.
* In Connection String, we have the below details:
  + **Hostname**
  + **Port Number**
  + **Database**
  + **Username**
  + **Password**
* The above details are mandatory. We have other details like **encrypt, trustServerCertificate, hostNameInCertificate, loginTimeout.**
* After connecting to database, try to read data from table.
* Below are the commands.

#Azure SQL Database Connection String

con\_url = 'jdbc:sqlserver://modak-sql.database.windows.net:1433;database=TrainingSQL-database;user=modak-srv-sql@modak-sql;password=dW5n@o96;encrypt=true;trustServerCertificate=false;hostNameInCertificate=\*.database.windows.net;loginTimeout=30;'

#Reading 'AdventureWorks\_Customers' table from 'dbo' schema and writing to new dataframe

adf = spark.read.jdbc(con\_url, 'dbo.AdventureWorks\_Customers')

#displaying dataframe data

adf.display()

**A screenshot of a computer

Description automatically generated**

**Appending the data present in dataframe to a table:**

* We have read the data from **dbo.AdventureWorks\_Customers** table. Now we are trying to append the data present in dataframe to another table in another schema.
* Here I am appending the data into another table named ‘**AdventureWorks\_Customers07’** in another schema named **‘dbx\_training’.**
* Below are the commands.

#Appending the data present in dataframe to table 'AdventureWorks\_Customers07' in another schema 'dbx\_training' using mode 'append'

adf.write.jdbc(con\_url, 'dbx\_training.AdventureWorks\_Customers07', mode='append')

#NOTE:

#If the table 'AdventureWorks\_Customers07' was not there in schema 'dbx\_training', it will automatically create the new table named 'AdventureWorks\_Customers07' using either 'append' or 'overwrite' mode.

**Reading the data present in the table:**

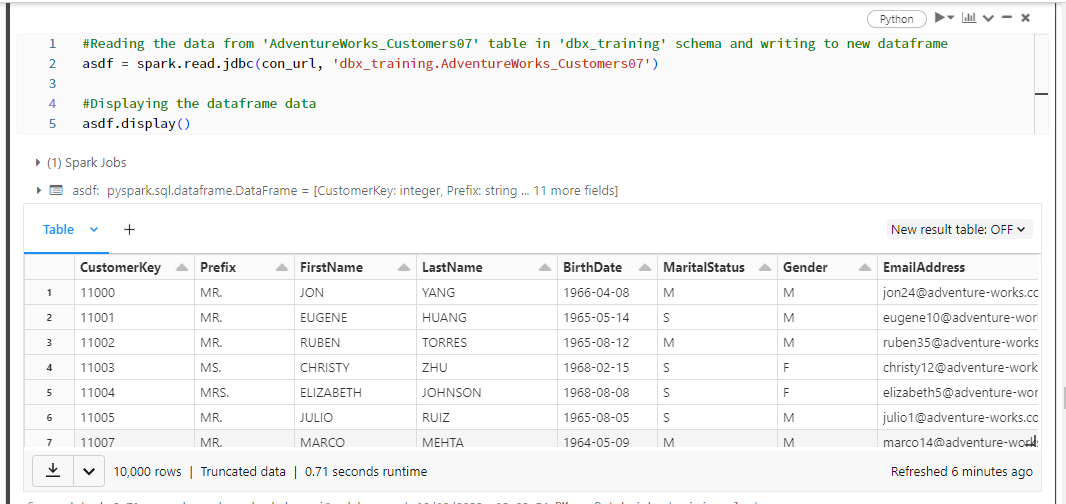
* Below are the commands to read the data present in the table and writing to a new dataframe.

#Reading the data from 'AdventureWorks\_Customers07' table in 'dbx\_training' schema and writing to new dataframe

asdf = spark.read.jdbc(con\_url, 'dbx\_training.AdventureWorks\_Customers07')

#Displaying the dataframe data

asdf.display()

****

**Displaying the selected columns data from Dataframe:**

* For now, we have table data in Dataframe. Now we are trying to display the selected columns data using **select** function in the below commands.

#Displaying the selected columns data 'CustomerKey' and 'FirstName' using select function

asdf.select('CustomerKey','FirstName').display()

**A screenshot of a computer

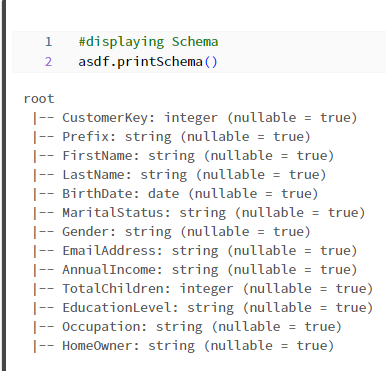
Description automatically generated**

**Filtering the selected column:**

* I am trying to filter one column. So first of all we need to find the **type** of each column.
* Below is the command to display the **Schema**

#displaying Schema

asdf.printSchema()

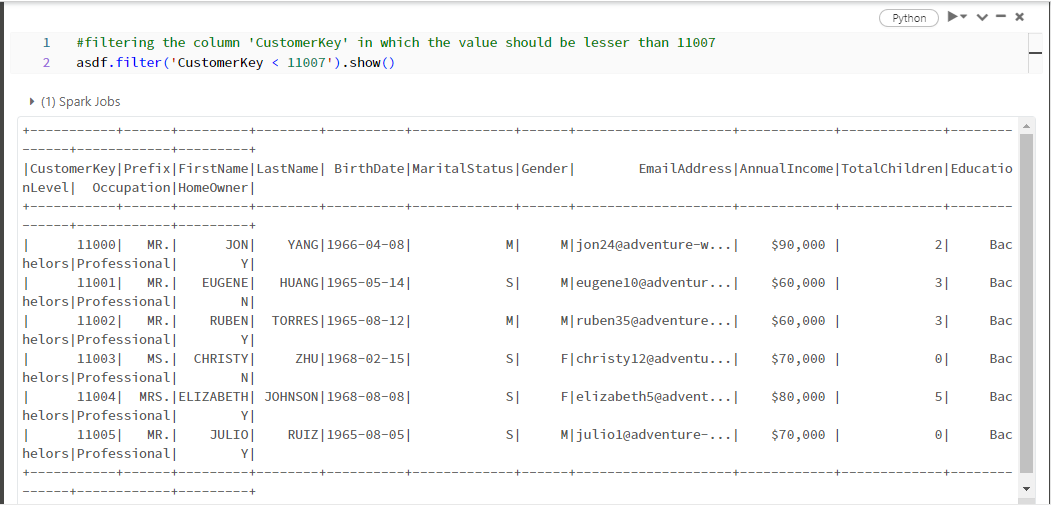
****

* We have one column **‘CustomerKey’** with type ‘**int’.**
* I am trying to filter the column **‘CustomerKey’** in which the values should be lesser than **11007.**

**Command:**

#filtering the column 'CustomerKey' in which the value should be lesser than 11007

asdf.filter('CustomerKey < 11007').show()

****

**NOTE:**

* One of the advantages of using **Azure Databricks** are
  + **Filesystem**
  + **Notebook**
  + **Widgets**
  + **Jobs**
  + **Secrets**

**DBUTILS:**

* **dbutils** in Azure Databricks is a utility provided by Databricks that allows you to interact with various aspects of your Databricks workspace and perform common data operations.
* It provides a set of functions and commands that can be used within Databricks notebooks, making it easier to work with data and perform tasks within the Databricks environment.
* We have so common uses of ‘**dbutils’** in Azure Databricks.

----------------------------------------------------------------------------------------------------------------

* **DBFS (Databricks File System):**
  + List files and directories in DBFS.
    - **dbutils.fs.ls(path)**
  + View the beginning of a file.
    - **dbutils.fs.head(file, maxBytes)**
  + Create or overwrite a file in DBFS with the given content.
    - **dbutils.fs.put(file, content, overwrite)**
* **File Operations:**
  + Copy files and directories within the Databricks file system.
    - **dbutils.fs.cp(source, destination)**
  + Remove files and directories.
    - **dbutils.fs.rm(path, recurse= True)**
  + Create directories.
    - **dbutils.fs.mkdirs(path)**
* **Widgets:**
  + Create a text widget that allows you to input values into a notebook.
    - **dbutils.widgets.text(name, defaultValue)**
  + Create a dropdown widget.
    - **dbutils.widgets.dropdown(name, values, defaultValue)**
* **Notebook Utilities:**
  + Run another notebook from the current notebook.
    - **dbutils.notebook.run(path, timeoutSeconds, parameters)**
  + Exit a notebook with a specified return value.
    - **dbutils.notebook.exit(value)**
* **Secrets Management:**
  + Retrieve a secret stored in Databricks Secrets.
    - **dbutils.secrets.get(scope, key)**
* **Spark Utilities:**
  + Run another notebook and capture its results.
    - **dbutils.notebook.runNotebook(path, timeoutSeconds, parameters)**
  + Run another notebook and capture its results with more options.
    - **dbutils.notebook.runNotebookV2(path, timeoutSeconds, notebookName, baseParameters, extraConfigs)**
  + Install Python libraries.
    - **dbutils.library.install(pypiPackage)**
  + Restart the Python runtime.
    - **dbutils.library.restartPython()**
* **Database Utilities:**
  + Mount an external storage source to a Databricks file system mount point.
    - **dbutils.fs.mount(source, mountPoint, extraConfigs)**
* **Job Management:**
  + Run a notebook in a Databricks job.
    - **dbutils.notebook.run(path)**

----------------------------------------------------------------------------------------------------------------

* **dbutils** simplifies common tasks within the Databricks environment and enables interaction with the file system, widgets, notebooks, and other resources.
* It is particularly useful when working in Databricks notebooks, as it provides a convenient way to manage and access data, perform operations, and create interactive user interfaces within your notebooks.

**DBFS (Databricks File System):**

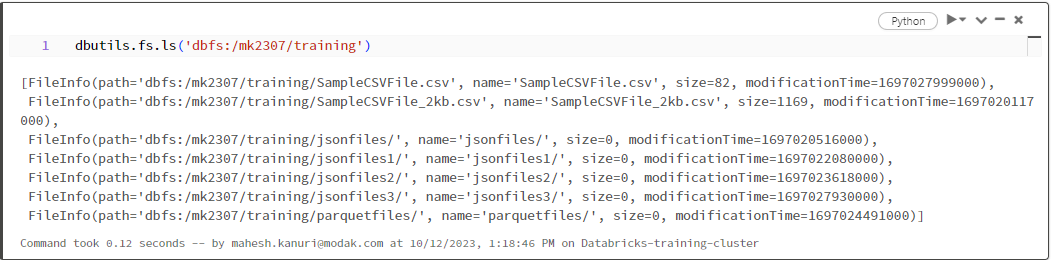
* **List files and directories in DBFS.**

**SYNTAX:**

**dbutils.fs.ls(path)**

#displaying the directories present in given dbfs path

dbutils.fs.ls('dbfs:/mk2307/training')

****

**File Operations:**

* **Copy files and directories within the Databricks file system:**

**SYNTAX:**

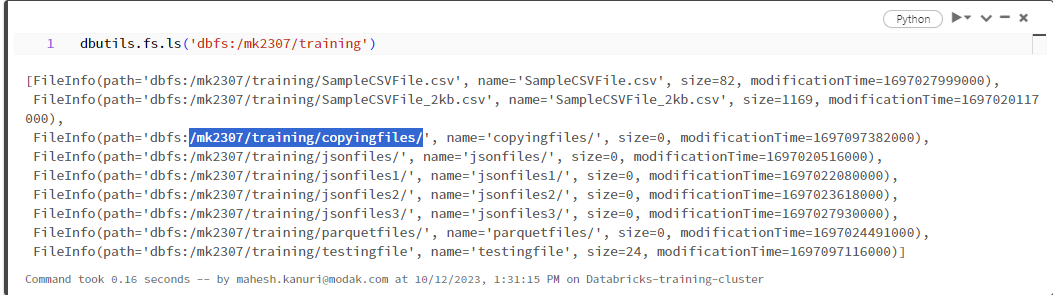
**dbutils.fs.cp(source, destination)**

#copying files from source path to destination path

dbutils.fs.cp('dbfs:/mk2307/training/testingfile', 'dbfs:/mk2307/training/copyingfiles/testingfile')

#NOTE:

#If the destination directory was not there, it will automatically create the directory and copied the file.



* **Remove files and directories:**

While removing **non-empty directories**, we need to set **recursive query parameter** to be **true.**

Otherwise, it will throw an error like below.

A screenshot of a computer code

Description automatically generated

So mention **recurse** parameter and run the command like below.

**SYNTAX:  
dbutils.fs.rm(path, recurse= True)**

#removing the mentioned directory

dbutils.fs.rm('dbfs:/mk2307/training/copyingfiles')

#NOTE:

#In the above command, the directory 'copyingfiles' was not a empty directory. So if we run the above command, it will throw an error. So to overcome this, mention the paramater 'recurse' to be True.

#removing the mentioned non-empty directory by mentioning directory path and recurse parameter to be True

dbutils.fs.rm('dbfs:/mk2307/training/copyingfiles', recurse= True)

A screenshot of a computer code

Description automatically generated

The directory got deleted along with all files present itself.

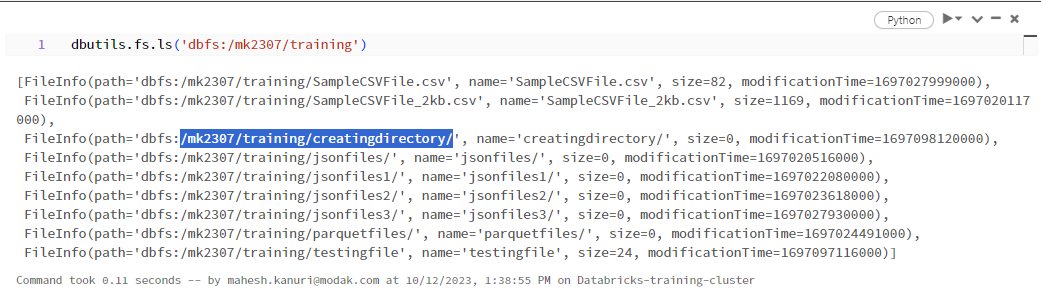
* **Create directories:**

**SYNTAX:**

**dbutils.fs.mkdirs(path**)

#creating directory

dbutils.fs.mkdirs('dbfs:/mk2307/training/creatingdirectory')

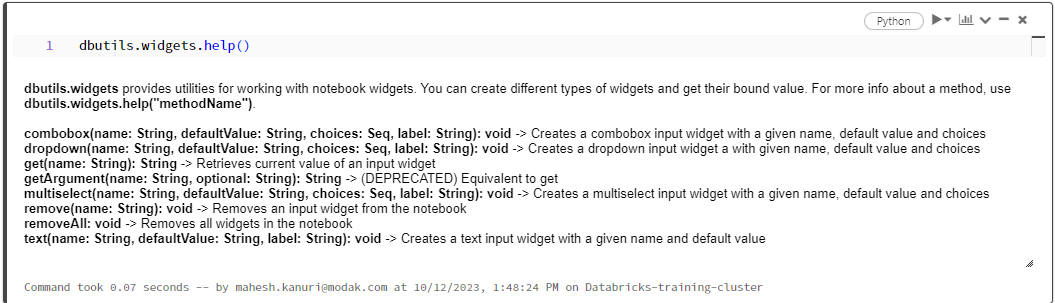


**Widgets:**

* Run the below command to see different types of **widgets.**

#displaying different types of widgets

dbutils.widgets.help()

****

* **text:**

Creates a text input widget with a given name and default value.

**SYNTAX:**

**text(name: String, defaultValue: String, label: String): void**

#creating a text input widget with a given name and default value

dbutils.widgets.text('mytext','textmk2307')

#retrieving current value of an input widget

value3=dbutils.widgets.get('mytext')

#printing value of an input widget

print(value3)

**A screenshot of a computer program

Description automatically generated**

* **combobox:**

Creates a combobox input widget with a given name, default value and choices.

**SYNTAX:**

**combobox(name: String, defaultValue: String, choices: Seq, label: String): void**

#creating a combobox input widget with given name, default value and choices

dbutils.widgets.combobox('mycombobox','combomk2307',['dhoni','virat'])

#retrieving current value of an input widget

value=dbutils.widgets.get('mycombobox')

#printing value of an input widget

print(value)

A screen shot of a computer

Description automatically generated

If we remove the value in combox, we will see the choices names **dhoni** and **virat** which are mentioned in the command. We can write different names also.

A screenshot of a computer

Description automatically generated

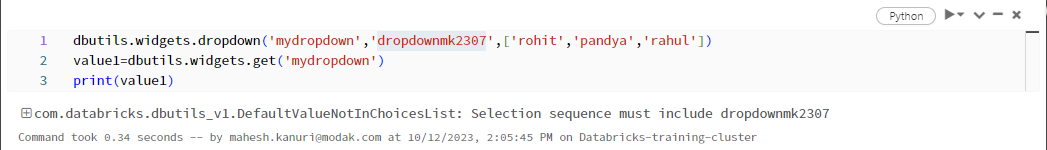
* **dropdown:**

Creates a dropdown input widget a with given name, default value and choices.

**SYNTAX:**

**dropdown(name: String, defaultValue: String, choices: Seq, label: String): void**

Here it is **mandatory** to mention the **default** value in **choices.** Otherwise it will throw the below error.



#creating a dropdown input widget a with given name, default value and choices

#NOTE:

#Here it is mandatory to mention the default value in choices. Otherwise it will throw the error.

dbutils.widgets.dropdown('mydropdown','dropdownmk2307',['dropdownmk2307','rohit','pandya','rahul'])

#retrieving current value of an input widget

value1=dbutils.widgets.get('mydropdown')

#printing value of an input widget

print(value1)

A screenshot of a computer

Description automatically generated

* **get:**

Retrieves current value of an input widget.

**SYNTAX:**

**get(name: String): String**

#retrieving current value of an input widget

value1=dbutils.widgets.get('mydropdown')

* **multiselect:**

Creates a multiselect input widget with a given name, default value and choices.

**SYNTAX:  
multiselect(name: String, defaultValue: String, choices: Seq, label: String): void**

#creating a multiselect input widget with a given name, default value and choices.

#NOTE:

#Here it is mandatory to mention the default value in choices. Otherwise it will throw the error.

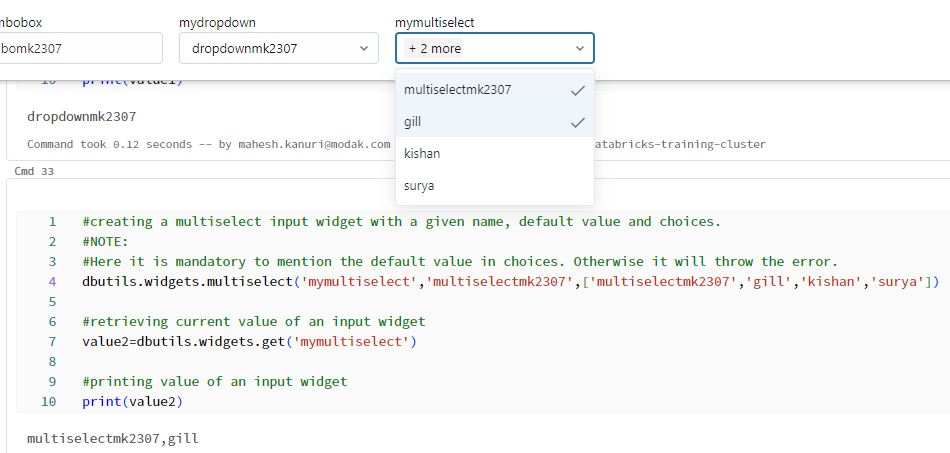
dbutils.widgets.multiselect('mymultiselect','multiselectmk2307',['multiselectmk2307','gill','kishan','surya'])

#retrieving current value of an input widget

value2=dbutils.widgets.get('mymultiselect')

#printing value of an input widget

print(value2)

****

* **remove:**

Removes an input widget from the notebook.

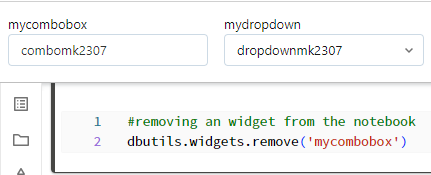
**SYNTAX:**

**remove(name: String): void**

#removing an widget from the notebook

dbutils.widgets.remove('mycombobox')

**Before removing widget:**

****

**After removing widget:**

**A screenshot of a computer

Description automatically generated**

* **removeAll:**

Removes all widgets in the notebook.

**SYNTAX:**

**removeAll: void**

#removing all widgets from the notebook

dbutils.widgets.removeAll()

**Before removing all widgets from the notebook:**

**A screenshot of a computer

Description automatically generated**

**After removing all widgets from the notebook:**

**A screenshot of a computer

Description automatically generated**

**Notebook Utilities:**

* **Run another notebook from the current notebook.**

**SYNTAX:**

**dbutils.notebook.run(path, timeoutSeconds, parameters)**

I have created another notebook like below

A screenshot of a computer

Description automatically generated

I want to run this newly created notebook **MK2307\_NOTEBOOK\_2** from the current notebook **MK2307\_NOTEBOOK\_1.**

#running another notebook from the current notebook.

dbutils.notebook.run('/Shared/MK2307/MK2307\_NOTEBOOK\_2', 120)

A screenshot of a computer

Description automatically generated

Now check whether the second notebook is executed or not by clicking on **Notebook job #783749311197772.**

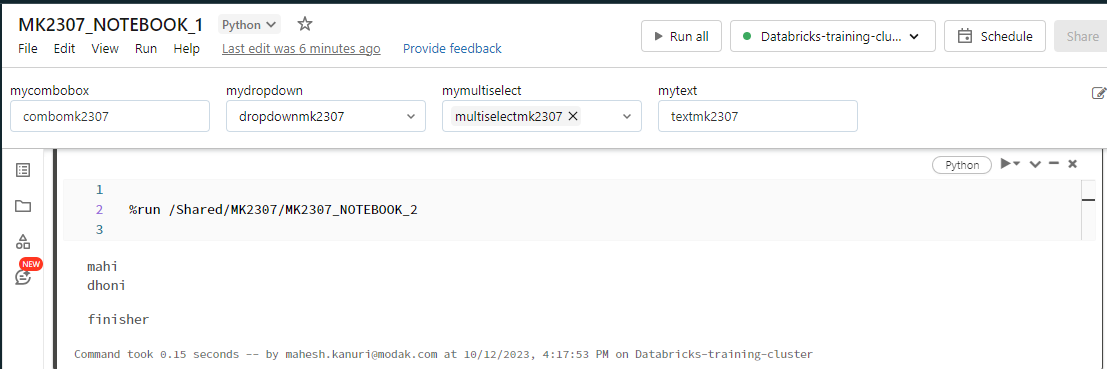
A screenshot of a computer

Description automatically generated

**The second notebook got executed.**

**%run /Shared/MK2307/MK2307\_NOTEBOOK\_2**

Another command to execute second notebook from the current notebook.



* **Exit a notebook with a specified return value:**

**SYNTAX:**

**dbutils.notebook.exit(value)**

Let us take an example.

In the above command, we are executing another notebook from the current notebook.

Same as that, we can **exit** that notebook from the current notebook.

That means when the **exit** command is executed in a second notebook, the commands which are present after the **exit** command cell will not get executed.

A screenshot of a computer

Description automatically generated

This is the second notebook. In this, **first two cells** got executed normally. But when the third cell got executed, the remaining cells will not get executed.

Now I am running this notebook from the current first notebook.

A screenshot of a computer

Description automatically generated

Now open the **Notebook job #662168914959276.**



Here, the last two cells will not be executed. It is showing **‘Command skipped’.**

Because when this runbook got executed, the **exit** command also will execute. So when the **exit** command got executed, the remaining cells will not got executed.

**10/10/2023:**

[Azure DataBricks Training - Session 5.mp4](https://modakanalytics0.sharepoint.com/:v:/r/sites/ModakTraining/Shared%20Documents/Training%20Data/Training%20Videos/Azure%20DataBricks%20Training/Azure%20DataBricks%20Training%20-%20Session%205.mp4?csf=1&web=1&e=NnFa99)

* Login to [**Azure Portal**](https://portal.azure.com/#home)**.**
* Search for **Key vaults.**
* Open your key-vault **training-hub.**

**Azure Vault:**

**------------------------------------------------------------------------------------------------**

* **Azure Key Vault** is a cloud-based service provided by Microsoft Azure that allows you to securely manage and safeguard cryptographic keys, secrets, certificates, and other sensitive information used by your applications and services.
* It serves as a centralized, highly secure repository for managing and controlling access to sensitive data.
* Here are some **key features** and purposes of Azure Key Vault:

**Secure Key Management:** Azure Key Vault provides a secure and compliant way to manage encryption keys, which are essential for data protection. It supports both hardware security modules (HSMs) and software-based key storage.

**Secrets Management:** Key Vault allows you to securely store and manage sensitive application settings, connection strings, API keys, and other secrets. This helps you avoid storing such information in code or configuration files.

**Certificate Management:** You can use Key Vault to store and manage SSL/TLS certificates for securing web applications, as well as code-signing certificates for software security.

**Access Control:** Key Vault supports fine-grained access control and permissions. You can grant specific users or services access to certain keys, secrets, or certificates, while restricting access for others.

**Auditing and Monitoring:** Key Vault provides robust auditing and monitoring capabilities, including detailed logs and metrics. You can monitor who is accessing your keys and secrets and what operations are being performed.

**Integration with Azure Services:** Azure Key Vault seamlessly integrates with various Azure services, making it easier to protect your applications and data. For example, you can use it to store keys for encrypting Azure Storage data, securing Azure Virtual Machines, or managing secrets for Azure Functions.

**Compliance and Certification:** Azure Key Vault is designed to meet various industry compliance standards, making it suitable for a wide range of applications, including those subject to regulatory requirements.

**Developer-Friendly APIs**: Azure Key Vault provides RESTful APIs for programmatic access, and it also offers client libraries and SDKs for multiple programming languages, simplifying integration into your applications.

* By using Azure Key Vault, you can enhance the security of your applications and services by centralizing the management of cryptographic keys and sensitive data, reducing the risk of exposure, and ensuring compliance with security and privacy regulations.

**------------------------------------------------------------------------------------------------**

**Secret Scopes:**

**------------------------------------------------------------------------------------------------**

* In Azure Databricks, a "**Secret Scope**" is a secure storage area within the Databricks workspace that is used to manage and store sensitive data, such as API keys, credentials, tokens, and other secrets.
* Secret Scopes are designed to help you keep your sensitive information secure and easily accessible to your notebooks and clusters when needed.

**Key characteristics of Secret Scopes** in Azure Databricks include:

**Secure Storage**: Secret Scopes provide a secure way to store sensitive information, preventing accidental exposure of secrets in your notebooks or code.

**Hierarchical Structure**: Secret Scopes can be organized hierarchically, allowing you to group related secrets together. This helps in managing and organizing secrets in a structured manner.

**Access Control**: You can configure access control to secret scopes, ensuring that only authorized users and clusters have access to the secrets within a scope.

**Scoped Secrets**: Secrets stored in a secret scope are scoped to a specific Databricks workspace, making them available for use within that workspace.

**Integration with Databricks Libraries and Notebooks**: You can reference secrets

in a Secret Scope directly in your Databricks notebooks and clusters. This integration simplifies the process of accessing secrets securely from your code.

**Key Integration with Databricks Runtime and Jobs**: Secrets can be easily integrated into Databricks jobs and runtime environments, allowing your scheduled jobs and runtime clusters to access the secrets they need.

**Secrets Rotation**: Some secrets, like database credentials, need to be rotated periodically for security. Databricks allows you to automate secrets rotation within Secret Scopes.

**Common use cases for Secret Scopes** in Azure Databricks include:

**Storing Database Credentials**: Storing database connection strings, usernames, and passwords securely, so they can be accessed by notebooks and clusters when connecting to external data sources.

**Accessing API Keys**: Storing API keys for external services and platforms, enabling secure access to APIs for data integration and processing.

**Handling Authentication Tokens**: Managing tokens required for authenticating with cloud services or other systems.

**Protecting Encryption Keys**: Securing encryption keys used for data encryption and decryption.

* To use Secret Scopes in Databricks, you typically define the secrets and their values within a secret scope and then reference these secrets in your notebooks and clusters. This approach ensures that sensitive information is kept safe and only accessible to authorized users and processes.

**------------------------------------------------------------------------------------------------**

**Creating Scopes:**

**------------------------------------------------------------------------------------------------**

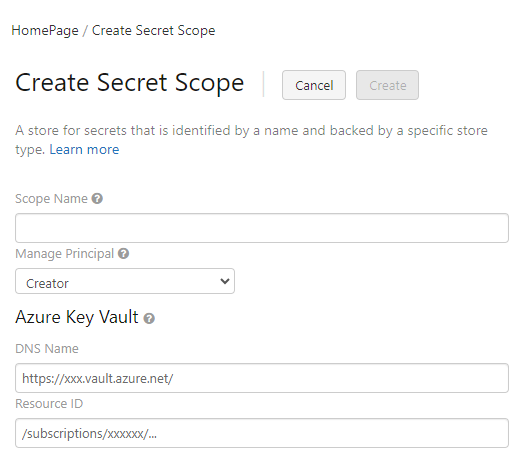
* To create a **scope,** open the **Databricks** URL.

[https://adb-7749372881626709.9.azuredatabricks.net/?o=7749372881626709#](https://adb-7749372881626709.9.azuredatabricks.net/?o=7749372881626709)

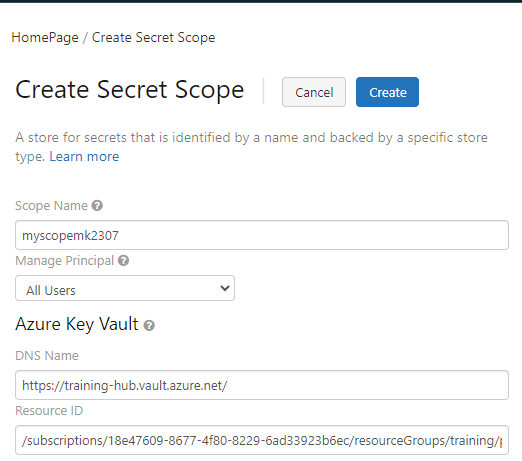
* The above one is the **URL.**
* Now add the **secrets/createScope** at the end of the URL like below

[https://adb-7749372881626709.9.azuredatabricks.net/?o=7749372881626709#secrets/createScope](https://adb-7749372881626709.9.azuredatabricks.net/?o=7749372881626709%23secrets/createScope)

* Now fill the below details



* Change the **Manage Principal** to **All Users.**
* To find the **DNS Name,** go to **Key Vaults** page.
* Click on Overview, we will find the **Vault URI.** It is nothing like **DNS Name.**
* To find the **Resource ID,** on the Overview page of **Key Vaults** page, click on **JSON View.**
* It will display the **Resource ID.**



* The scope **myscopemk2307** is created.

A screenshot of a computer

Description automatically generated

**------------------------------------------------------------------------------------------------**

* **Displaying the list of Scopes:**

#displaying the list of Scopes

dbutils.secrets.listScopes()

A screenshot of a computer code

Description automatically generated

* **Creating the Secret:**

Goto **Key-vault** page and click on **Secrets.**

Click on **Generate/Import.**

Fill the below details.

A screenshot of a computer

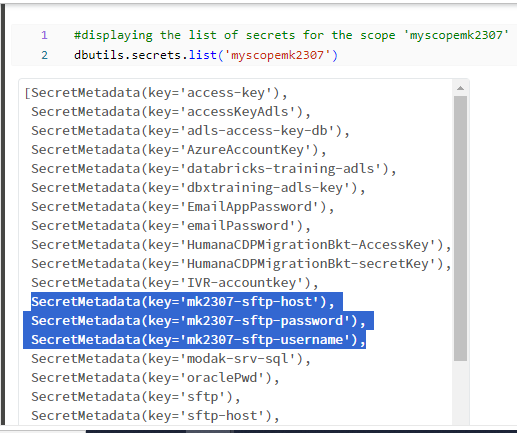
Description automatically generated

I have created three secrets **mk2307-sftp-host, mk2307-sftp-username** and **mk2307-sftp-password.**

* **Displaying the Secrets:**

#displaying the list of secrets for the scope 'myscopemk2307'

dbutils.secrets.list('myscopemk2307')

****

* **Retrieving the Secret values:**

#retreiving the sftp details from 'myscopemk2307' scope

sftp\_host = dbutils.secrets.get('myscopemk2307', 'mk2307-sftp-host')

sftp\_username = dbutils.secrets.get('myscopemk2307', 'mk2307-sftp-username')

sftp\_password = dbutils.secrets.get('myscopemk2307', 'mk2307-sftp-password')

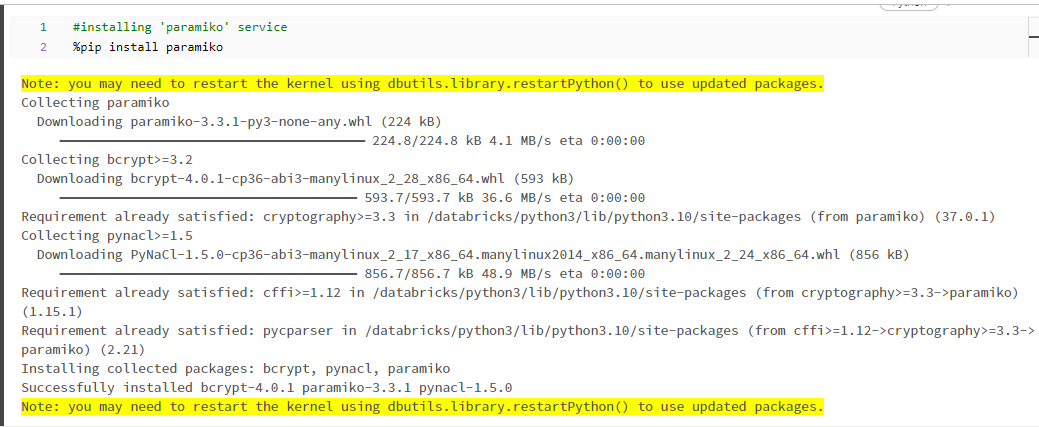
print(sftp\_host)

* **Copying files from SFTP to DBFS:**

We need to install **paramiko** service.

#installing 'paramiko' service

%pip install paramiko



Importing **Paramiko** service, retrieving **sftp hostname, username and password** from secret ‘**myscopemk2307’** and establishing the connection.

#importing paramiko service

import paramiko

#retreiving sftp hostname, username and password from secret 'myscopemk2307'

sftp\_host = dbutils.secrets.get('myscopemk2307', 'mk2307-sftp-host')

sftp\_username = dbutils.secrets.get('myscopemk2307', 'mk2307-sftp-username')

sftp\_password = dbutils.secrets.get('myscopemk2307', 'mk2307-sftp-password')

sftp\_port = 22

#establishing the connection

ssh\_client = paramiko.SSHClient()

ssh\_client.set\_missing\_host\_key\_policy(paramiko.AutoAddPolicy())

ssh\_client.connect(sftp\_host, sftp\_port, sftp\_username, sftp\_password)

print('Connection is established')

Now opening the **SFTP** connection. After that creating one directory **sftp** in **dbfs** to store the file which will be copied from **SFTP.**

Used the method **get** method.

#opening the sftp connection

sftp = ssh\_client.open\_sftp()

#creating one directory 'sftp' in dbfs to store the copied file

dbutils.fs.mkdirs('dbfs:/mk2307/training/sftp')

#sftp source file path

remote\_path = '/uploads/format1-json/format2.json'

#dbfs destination path. Here we need to mention the file name also, otherwise the file will be created with different name.

local\_path = '/dbfs/mk2307/training/sftp/format2.json'

# sftp://sftp\_srv\_account@w3.devcrawlers.modak.com/uploads/format1-json/entity.json

# sftp://sftp\_srv\_account@w3.devcrawlers.modak.com/uploads/format1-json/format2.json

#copying file from SFTP to DBFS

sftp.get(remote\_path,local\_path)

print('File copied successfully from SFTP to DBFS!!')

#checking whether file was copied or not

dbutils.fs.ls('dbfs:/mk2307/training/sftp')

* **Copying files from DBFS to SFTP:**

We used the method **put** to copy the file from **DBFS** to **SFTP.**

#dbfs source file path

local\_path = '/dbfs/mk2307/training/sftp/format2.json'

#sftp destination path

remote\_path = '/uploads/csvfiles/dbfstosftp/format2.json'

#putting file from dbfs source path to sftp destination path

sftp.put(local\_path, remote\_path)

print('File copied successfully from DBFS to SFTP!!')

#checking whether file was copied or not

sftp.stat('/uploads/csvfiles/dbfstosftp/format2.json')

**11/10/2023:**

[Azure DataBricks Training - Session 6.mp4](https://modakanalytics0.sharepoint.com/:v:/r/sites/ModakTraining/Shared%20Documents/Training%20Data/Training%20Videos/Azure%20DataBricks%20Training/Azure%20DataBricks%20Training%20-%20Session%206.mp4?csf=1&web=1&e=MzgYZa)

**ABOUT DELTA LAKE:**

[Databricks\_Delta\_Concepts.pptx](https://modakanalytics0-my.sharepoint.com/:p:/g/personal/mahesh_kanuri_modak_com/EUWCzfaSQ6tLqoUPor8D04cB_QHyN6t9doA9bNHnyaQ7pQ?e=2QBZrj)

**Creating a Delta Table using SQL:**

%sql

--using sql like syntax

-- creating a delta table using SQL

create or replace table mk2307\_delta\_table\_using\_sql(

  Name STRING,

  Emp\_ID INT,

  Department STRING,

  Gender STRING,

  Salary INT

)

USING DELTA

LOCATION "dbfs:/mk2307/training/deltatables/mk2307\_delta\_table\_using\_sql";

-- In the above query, if we didn't mentioned the location, only table is create in the 'hive\_metastore' catalog database.

-- Mentioning the location path will help to create the directory in given location path also.

-- In that directory, we have one folder called 'delta\_log'. In that directory, the JSON files will be created for each operation (Like Select, Insert)

-- select \* from mk2307\_delta\_table\_using\_sql;

**Creating a Delta Table using PySpark:**

from delta.tables import \*

#creating a delta table using PySpark

DeltaTable.createOrReplace(spark) \

  .tableName("mk2307\_delta\_table\_using\_pyspark") \

  .addColumn("Name","STRING") \

  .addColumn("Emp\_ID","INT") \

  .addColumn("Department","STRING") \

  .addColumn("Gender","STRING") \

  .addColumn("Salary","INT") \

  .location("dbfs:/mk2307/training/deltatables/mk2307\_delta\_table\_using\_pyspark") \

  .execute()

#In the above command, if we didn't mentioned the location, only table is create in the 'hive\_metastore' catalog database.

#Mentioning the location path will help to create the directory in given location path also.

#In that directory, we have one folder called 'delta\_log'. In that directory, the JSON files will be created for each operation (Like Select, Insert)

**Creating a Delta Table using DataFrame:**

#reading data from CSV file and writing to Dataframe

dataframe = spark.read.option("header","true").csv("dbfs:/mk2307/training/color\_first.csv")

#creating a table by writing the data from dataframe using mode 'append'

dataframe.write.format("delta").mode("append").option("path","dbfs:/mk2307/training/deltatables/mk2307\_delta\_table\_using\_dataframe").saveAsTable("mk2307\_delta\_table\_using\_dataframe")

**Inserting data into Delta table which is created using SQL:**

%sql

-- inserting a row  into delta table

-- inserting data into delta table which is created using 'SQL'

--  insert into mk2307\_delta\_table\_using\_sql values ("Mahesh",1,"Software","Male",1000);

--  insert into mk2307\_delta\_table\_using\_sql values ("Keerthi",2,"Development","Female",2000),

--                                 ("Bhargav",3,"Sales","Male",3000),

--                                 ("Abhilash",4,"Sales","Male",3000),

--                                 ("Sharath",5,"Development","Male",3050);

-- reading the data from delta table

select \* from mk2307\_delta\_table\_using\_sql;

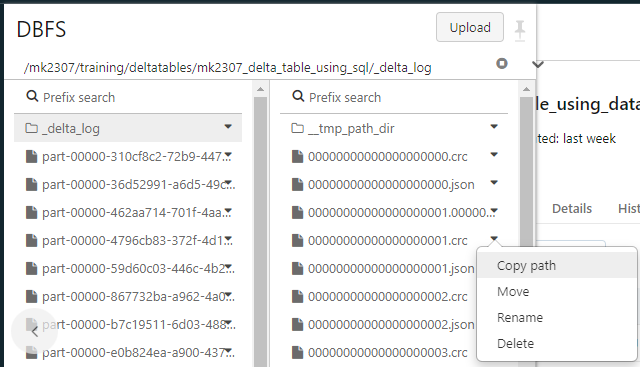
-- deleting the table

-- delete from mk2307\_delta\_table\_using\_sql;

**A screenshot of a computer

Description automatically generated**

**Reading a Transaction Log file:**

****

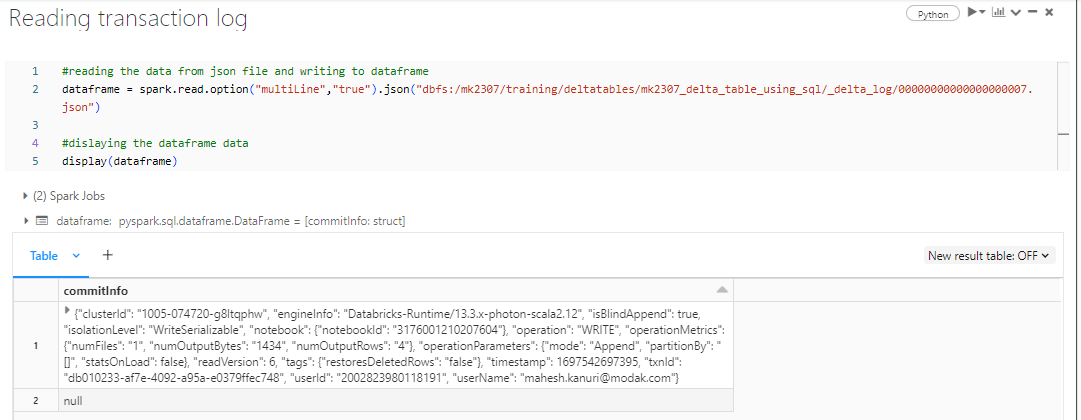
Copied the json Path file.

#reading the data from json file and writing to dataframe

dataframe = spark.read.option("multiLine","true").json("dbfs:/mk2307/training/deltatables/mk2307\_delta\_table\_using\_sql/\_delta\_log/00000000000000000007.json")

#dislaying the dataframe data

display(dataframe)

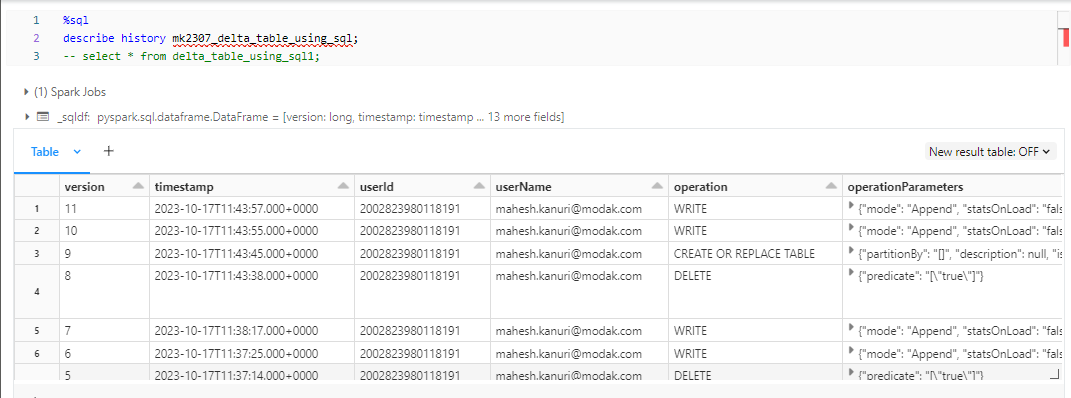


**Checking history of Delta table:**

%sql

describe history mk2307\_delta\_table\_using\_sql;

-- select \* from delta\_table\_using\_sql1;

****

In the **history,** we have two columns named **version** and **timestamp.**

Using these column data, we can display what is happened for that operation.

Like if we write any data into table, it will create one **version** and **timestamp.**

Using any of these two columns, we can find what is happened for that operation.

**Time Travel using versionAsOf() and timestampAsOf() using pyspark:**

# Time Travel using versionAsOf() in pyspark

dataframe = spark.read.format("delta").option("versionAsOf","10").table("mk2307\_delta\_table\_using\_sql")

#displaying dataframe data

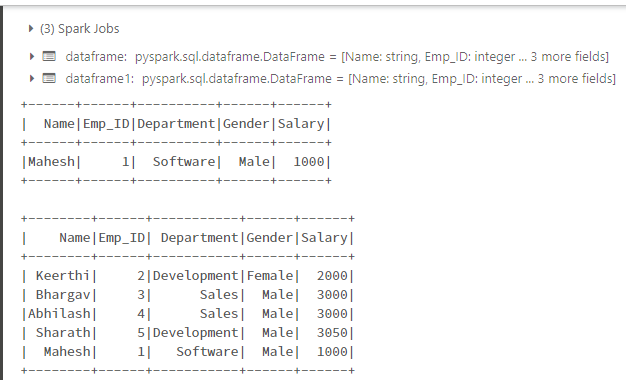
dataframe.show()

# Time Travel using timestampAsOf() in pyspark

dataframe1 = spark.read.format("delta").option("timestampAsOf","2023-10-17T11:43:57.000+0000").table("mk2307\_delta\_table\_using\_sql")

#displaying dataframe data

dataframe1.show()

****

**Time Travel using versionAsOf() and timestampAsOf() using SQL:**

%sql

-- time travel using versionAsOf()

select \* from mk2307\_delta\_table\_using\_sql version as of 6;

-- time travel using timestampAsOf()

select \* from mk2307\_delta\_table\_using\_sql timestamp as of "2023-10-17T11:36:23.000+0000" ;

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**Schema Evolution:**

* For now, we have the columns **Name, Emp\_ID, Department, Gender** and **Salary.**
* The task is to create two **dummy** columns and add the data.
* I am adding two columns **extra\_col** and **dummy\_col.**
* To do that, firstly I am creating Dataframe.
* After that, using the option **mergeSchema,** I am merging the columns.

# creating a dataframe with additional column

from pyspark.sql.types import \*

#adding new data to be appended to the table

emp\_data = [("Dhoni",183,"Cricketer","Male",10000,"dummy\_col\_data","sample")]

#schema

emp\_schema = StructType([

    StructField("Name",StringType(),True), \

    StructField("Emp\_ID",IntegerType(),False), \

    StructField("Department",StringType(),True),\

    StructField("Gender",StringType(),True), \

    StructField("Salary",IntegerType(),True), \

    StructField("extra\_col",StringType(),True), \

    StructField("dummy\_col",StringType(),True)

])

#creating dataframe using schema

df = spark.createDataFrame(data=emp\_data,schema=emp\_schema)

#displaying dataframe data

display(df)

A screenshot of a computer

Description automatically generated

* Using the below command, we are merging the data.

# writing it using the mergeSchema option

df.write.format("delta").mode("append").option("mergeSchema",True).saveAsTable("mk2307\_delta\_table\_using\_sql")

* Read the table to check whether the newly data is added or not.

%sql

-- displaying the table data

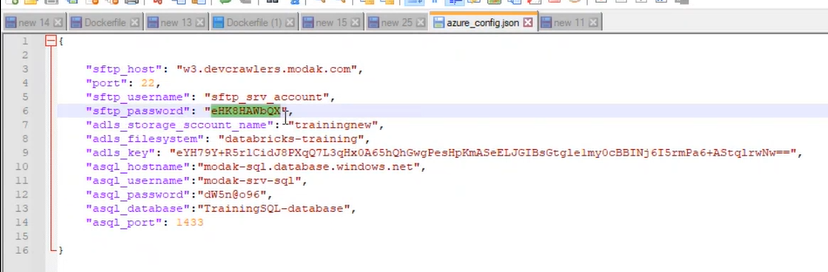
select \* from mk2307\_delta\_table\_using\_sql;

**A screenshot of a computer

Description automatically generated**

**13/10/2023:**

[Azure DataBricks Training - Session 7.mp4](https://modakanalytics0.sharepoint.com/:v:/r/sites/ModakTraining/Shared%20Documents/Training%20Data/Training%20Videos/Azure%20DataBricks%20Training/Azure%20DataBricks%20Training%20-%20Session%207.mp4?csf=1&web=1&e=YlF3P3)



**Steps to Install Python ( Version >3) in Windows:**

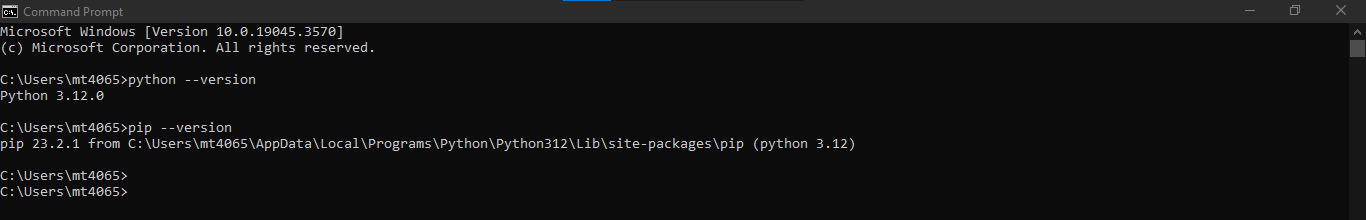
* Visit the official Python website: [Python Downloads](https://www.python.org/downloads/)
* Click on the "**Downloads**" tab.
* Choose the latest version of Python 3.x for Windows.
* Run the downloaded installer file.
* Make sure to check the box that says **"Add Python to PATH"** during installation.
* Open the **Command Prompt.**
* Type **python --version** or python -V to check if Python is installed.
* Type **pip --version** or pip -V to check if Pip is installed.

**Steps to Install Python ( Version >3) in Ubuntu:**

* Open the Terminal.
* Run the below command to Update the package list.
  + **sudo apt update**
* Run the below command to install python 3.
  + **sudo apt install python3**
* Run the below command to install pip for python 3.
  + **sudo apt install python3-pip**
* Type **python3 --version** or python3 -V to check if Python is installed.
* Type **pip3 --version** or pip3 -V to check if Pip is installed.

**Installing Databricks CLI:**

* We need to install python ( Version >3) and pip.

****

* Then install the **databricks** using below command

**Command:**

pip install databricks-cli

**A screenshot of a computer

Description automatically generated**

* Run the below command to view the different options.

**Command:**

databricks --help

**A screenshot of a computer

Description automatically generated**

**Configuring Databricks CLI:**

* We will use the command **configure** to configure host and authentication info for CLI.
* Run the below command to start the process of configuring Databricks CLI.

**Command:**

databricks configure --token

****

* After running this command, it will ask the **Databricks Host.**
* Below one is the **Databricks Host.**

****

* After giving **Databricks Host,** it will ask **Token.**
* To find the Token, Click on Mail ID >> User Settings >> Developer.

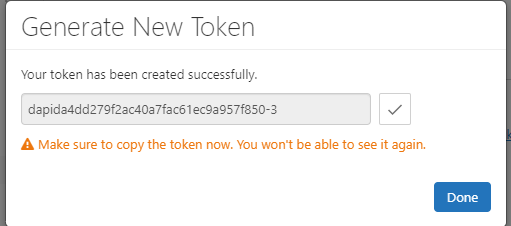
**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

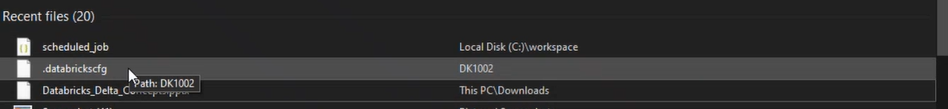
Description automatically generated**

* Click on “**Generate new token”** to create a new token.

****

****

* After entering the details, the file **.databricks.cfg** will be created in **C:\Users\mt4065**.
* In that file, we have all the details **host, token.**
* Check whether we gave correct details or not.

****

**A close up of a computer screen

Description automatically generated**

* If token was not present like in above picture, replace “[]” with our token like below

**A screenshot of a computer

Description automatically generated**

* Run the below command to view the directories present in **DBFS.**

**Command:**

databricks fs ls

**A screen shot of a computer

Description automatically generated**

**Running the Notebook using Workflows:**

* Go to **Workflows.**
* Click on **Create JOB.**
* Fill the required details.

**A screenshot of a computer

Description automatically generated**

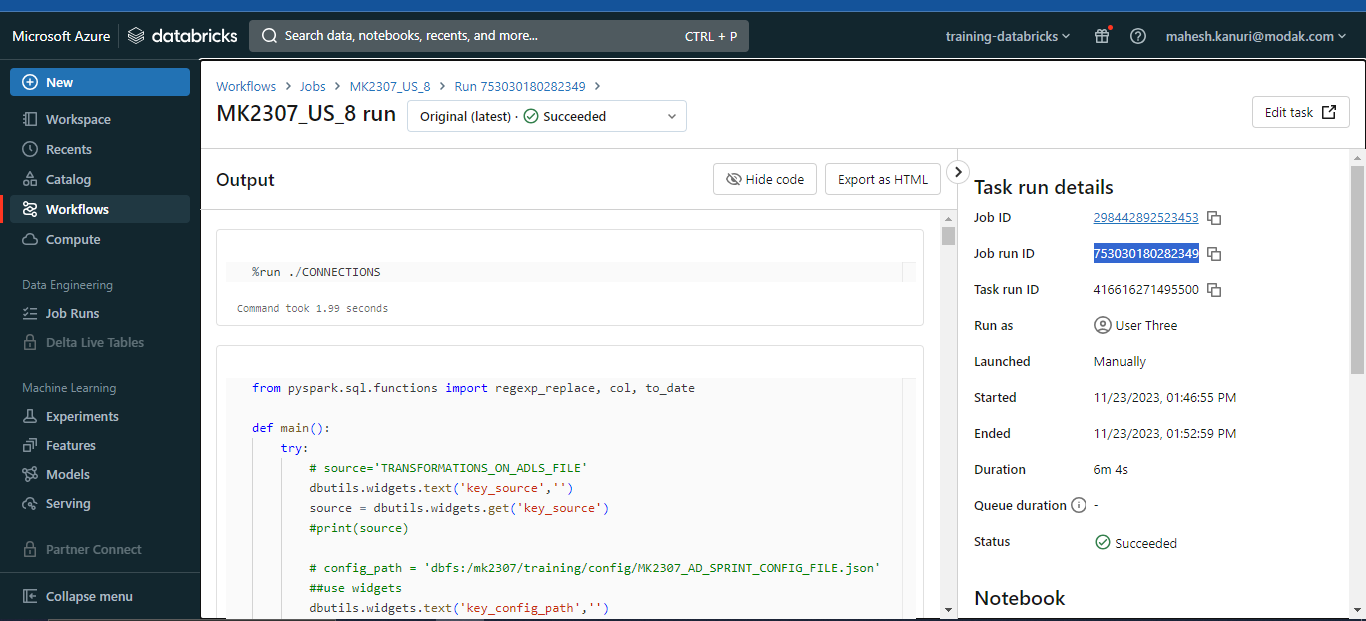
* After creating the JOB, click on **Run now** to run the job.
* Run the below command to run the job in **databricks CLI.**

**Command:**

databricks jobs run-now --job-id <<job-id>>

****

* To view the status of this run, find the **run\_id**

****

* Run the below command to display the status of that run-id.

**Command:**

databricks runs get --run-id <run-id>

