Table of Contents

Sessions` Recordings

https://www.scholarnest.in/courses/take/bigdataandspark/multimedia/34984128-welcome-live-session-recording-7th-may-2022

https://www.scholarnest.in/courses/take/spark-tables-and-dataframes/multimedia/35064706-week-1-review-doubts-datalake-vs-lakehouse

https://www.scholarnest.in/courses/take/spark-tables-and-dataframes/multimedia/35356447-interview-preparation-strategy

FAQs

1. What is the difference between Data Warehouse and Data Lake?
2. ~~What is difference between Data Lake and Lakehouse?~~
3. When do you recommend Data Lake/~~Lakehouse~~ over Data Warehouse?
4. I have a few petabytes of structured data, and I want to build a new system to process this data. Do you recommend Hadoop or Teradata will do the job?
5. What is Scalability? How does Hadoop/Spark offer Scalability?
6. What is High Availability? How does Hadoop/Spark offer High Availability?
7. What is fault tolerance? How does Hadoop/Spark offer fault tolerance?
8. Can you explain Hadoop Architecture?
9. Can you explain HDFS architecture?
10. Can you explain the YARN architecture?
11. Can you explain the Map/Reduce architecture? How does M/R work?
12. How YARN runs your application?
13. What is the purpose of the master node in a Hadoop cluster?
14. Can you explain how data is stored in a Hadoop cluster?
15. What is the role of a Name node in Hadoop?
16. What happens if a master node fails in a Hadoop cluster?
17. I am collecting data from an IOT device (smart meter). Is it structured/semi-structured or unstructured data?
18. Do you recommend using Hive or Spark? Which one is better and why? Can we migrate the Hive project to spark?
19. How will you calculate the Hadoop cluster capacity available for your application?
20. What is difference between Spark and Databricks?

Emergence of Big Data

Categorising software systems accurately and crafting a complete list of all possible categories is almost impossible. However, we can still start with the following categories listed below.

Software Engineering Specializations

1. System Software
3. Programming Tools
3. Desktop Applications
4. Data Processing Application
5. Web & Mobile Applications
6. Software Platforms
7. Cloud Computing
8. Machine Learning and AI

We saw the evolution of the COBOL programming language specifically designed for business data processing. The COBOL, also known as Common Business-Oriented Language, was the first of its kind. COBOL allowed us to store data in files, create index files, and process data efficiently.

However, we saw data processing shift from COBOL to relational databases such as Oracle and Microsoft SQL Server.

Out of all the software specializations we have listed earlier, I wanted to attract your attention towards Data processing applications. You can think of COBOL as the first serious attempt towards enabling data processing. And COBOL was designed in 1959.

The Oracle database achieved the subsequent major success in enabling data processing. And Oracle was founded in 1977. So data processing has always been at the centre of the Software industry. Everything else will come and go, but data will only grow.

Data processing is an evergreen field for software engineers. Technologies may evolve, but the requirement for processing more and more data at a faster speed will keep becoming more critical.

We have used RDBMS technology for many decades. Some popular RDBMS systems are Oracle, SQL Server, PostgreSQL, MySQL, Teradata, and Exadata. These RDBMS systems offered us three main features to help us develop Data Processing applications.

1. SQL - An easy Data Query Language
2. Scripting Languages such as PL/SQL and Transact SQL
3. Interface for other programming languages such as JDBC and ODBC

So we used SQL for querying data and PL/SQL for doing things that we couldn't do using SQL. They also offered interfaces such as ODBC/JDBC so we could interact with data using the programming languages.

We could create data processing applications using these technologies.

We had the following requirements in the Big Data problem.

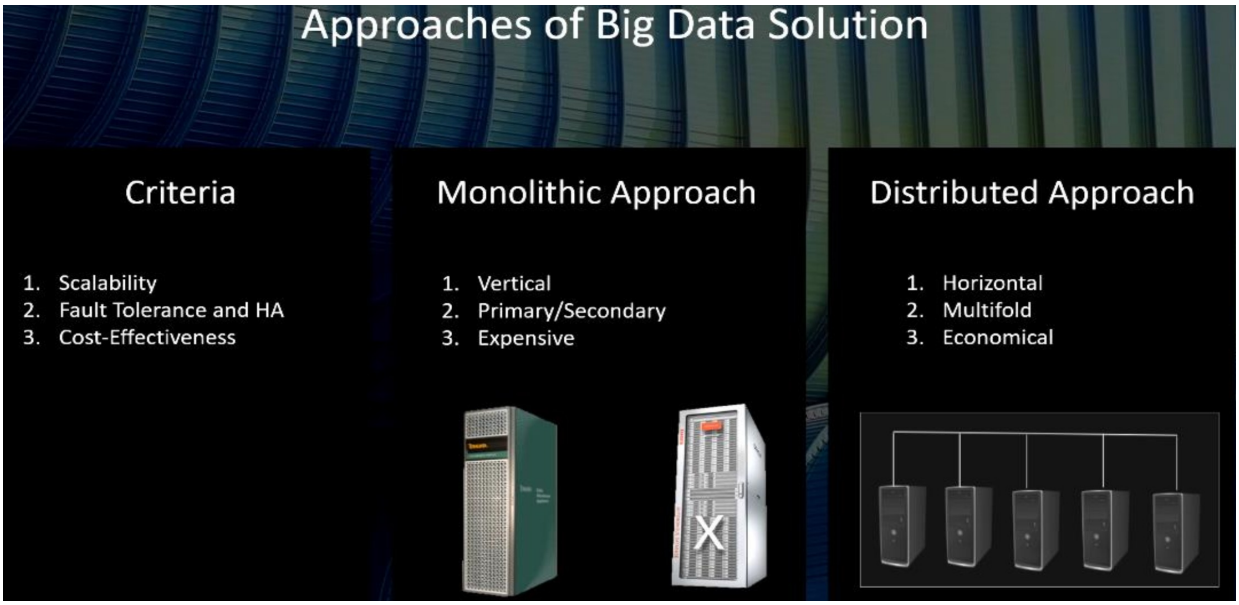
There was two approaches to the big data problem:

1. The monolithic approach designs one large and robust system

that handles all the requirements. Teradata and Exadata are examples. These two systems mainly support only structured data. So we cannot call them big data systems, but they are designed using a monolithic approach.

2. The distributed approach takes many smaller systems and bring them together to solve a bigger problem.

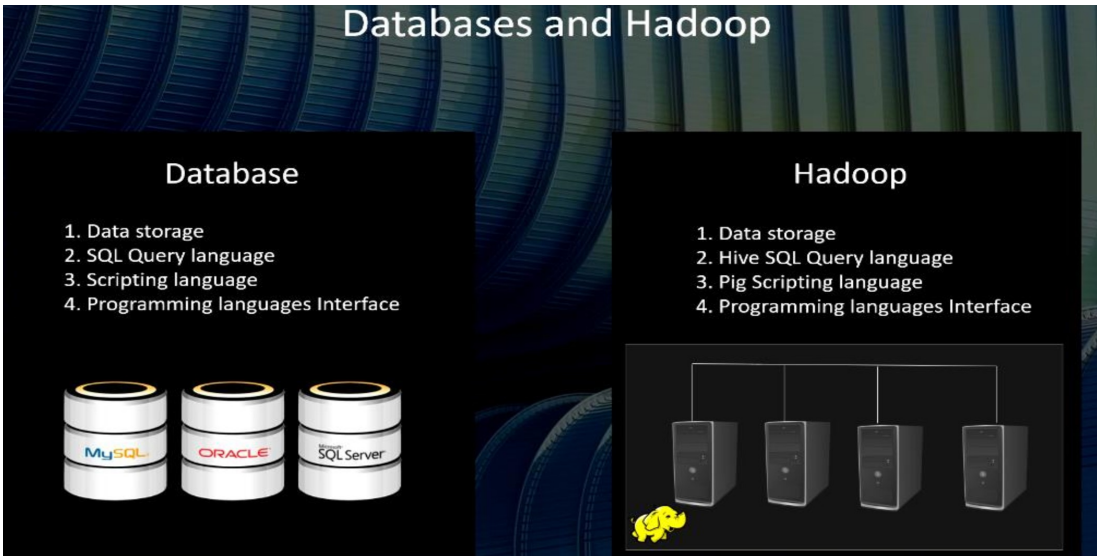
Here are the two approaches to Big Data solution.



Hadoop came up as a new data processing platform to solve Big Data problems. The Hadoop platform was designed and developed in layers. The core platform layer offered three capabilities:

1. Distributed cluster formation or Cluster Operating System
2. Data storage and retrieval on the distributed cluster or Distributed Storage
3. Distributed data processing using Java programming language or Map-Reduce Framework

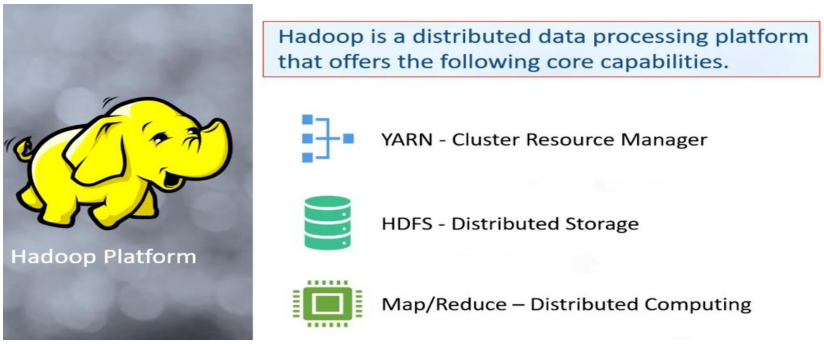
Here is a comparison between Database and Hadoop.



Hadoop Architecture – History and Evolution

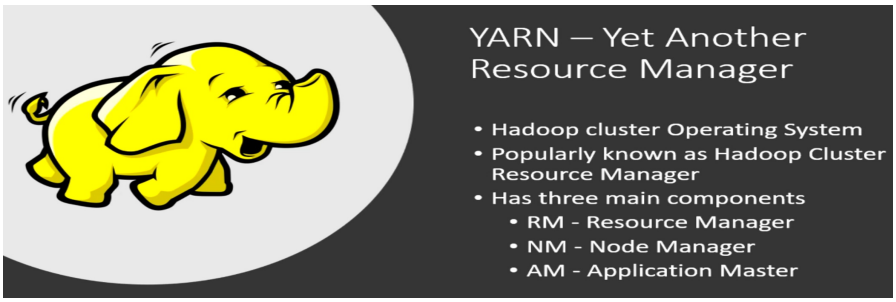
Hadoop is a distributed data processing platform that offers three core capabilities listed below:

1. YARN
2. HDFS
3. Map/Reduce



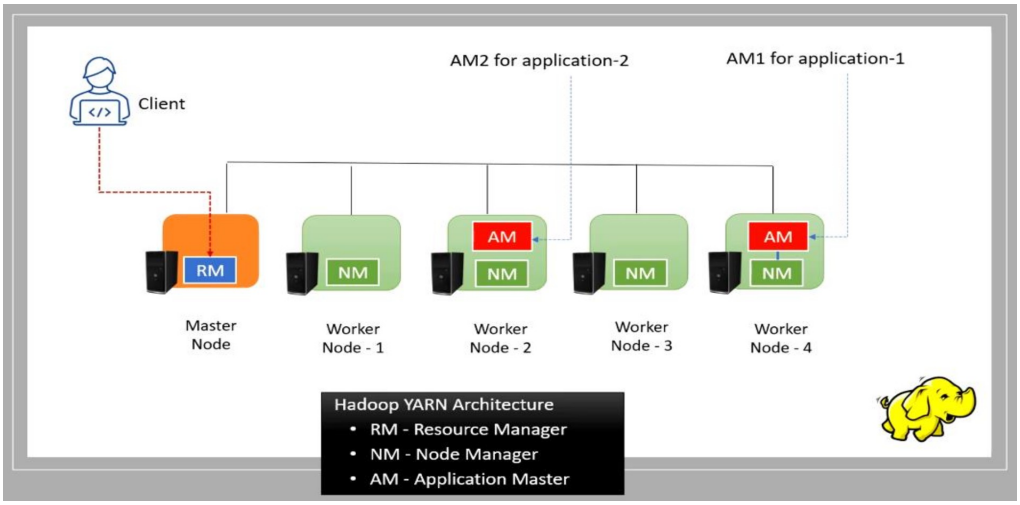
YARN:

YARN is the Hadoop cluster resource manager. It allows multiple applications to run on the Hadoop Cluster and share resources amongst the applications.



YARN:

Assume we installed Hadoop, and now these five computers form a Hadoop cluster. Hadoop uses a master-slave architecture. So one of these machines will become the master, and the remaining will act as the worker node.



YARN:

We have a five-node cluster that I showed you in the earlier slide. One node acts as a master and runs the YARN resource manager service. The other four nodes act as a Worker and run a node manager service. The node manager will regularly send the node status report to the resource manager. We created a Hadoop cluster so we can run big data applications. For

running an application on Hadoop, you must submit the application to the YARN resource manager. Assume you submitted a Java application to the YARN using a command line submit tool. Now the resource manager should run this application on the cluster.

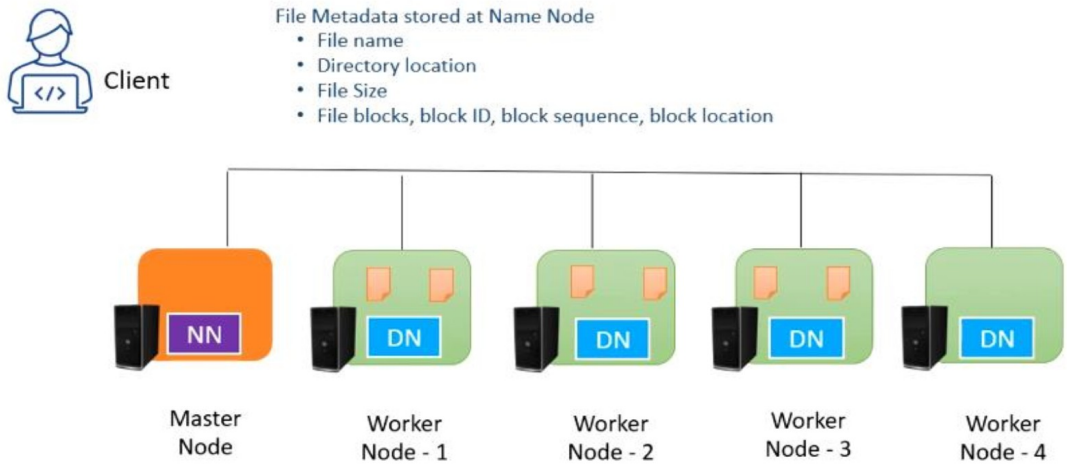
So, the resource manager will request one of the node managers to start a resource container and run an AM (application master) in the container. And your application starts running inside the Application Master container. So, we submit our application to the Resource Manager.

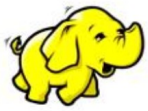
The resource manager requests the node manager for allocating an application master container and starting your application inside the AM container. Each application on YARN runs inside a different AM container. If you have ten applications running in parallel, you will see 10 AM containers on your Hadoop cluster.

HDFS:

The HDFS stands for Hadoop Distributed File system, and it allows you to save and retrieve data files in the Hadoop Cluster. The HDFS has the following components.

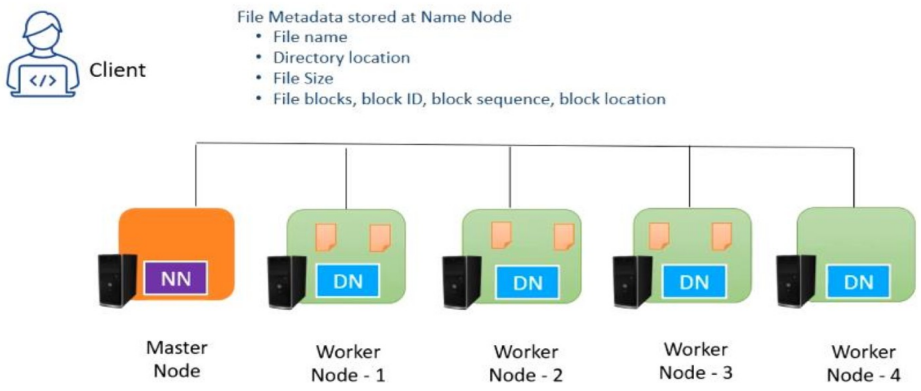
1. Name Node (NN)
2. Data Node (DN)





HDFS:

Assume we have five computers shown below. We already installed Hadoop on these computers and created a Hadoop cluster. Hadoop will install the Name Node service on the master. And each worker node runs a data node service. The name node with the data node service forms the HDFS. The primary purpose of the HDFS is to allow us to save files on this Hadoop cluster and read them whenever required.



Map/Reduce:

Map-reduce is a programming model and a framework. A programming model is a technique or a way of solving problems. The M/R framework is a set of APIs and services that allow you to apply the map-reduce programming model. Hadoop taught us the map-reduce programming modal and also offered a Map-Reduce programming framework to implement it.

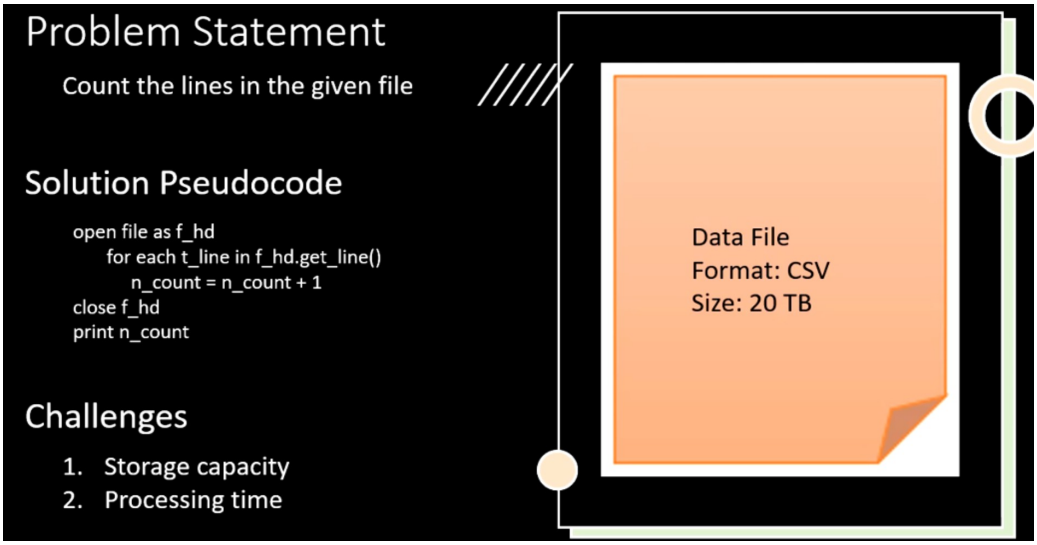
Map/Reduce:

You have to count the lines in a 20 TB CSV file. There are two challenges in this problem statement:

1. Huge file size, It is hard to find machines to store 20 TB of data. And this problem becomes more

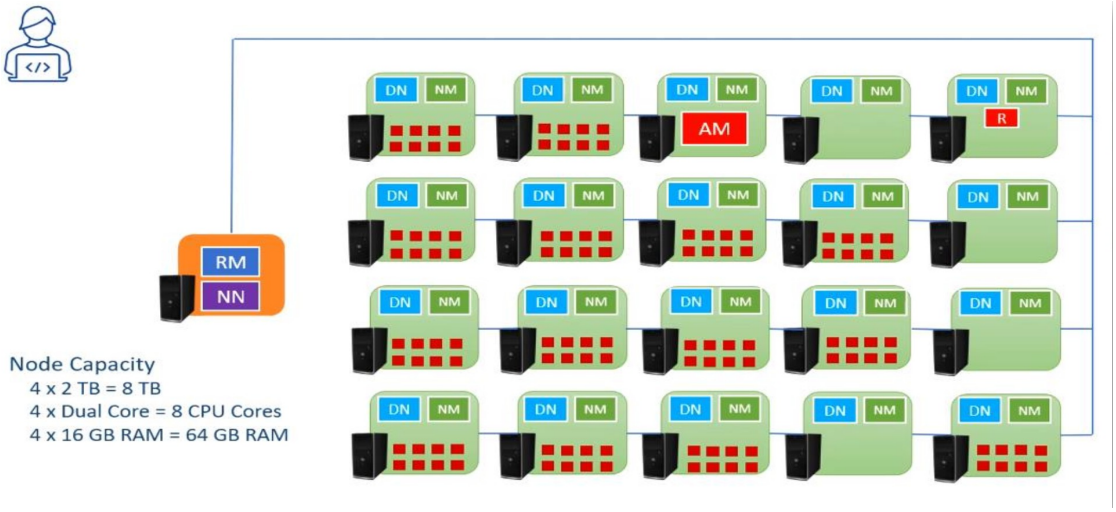
   complex if we grow the size in petabytes.

2. We also have a processing time problem. A simple line count on a 20 TB file takes hours or days.



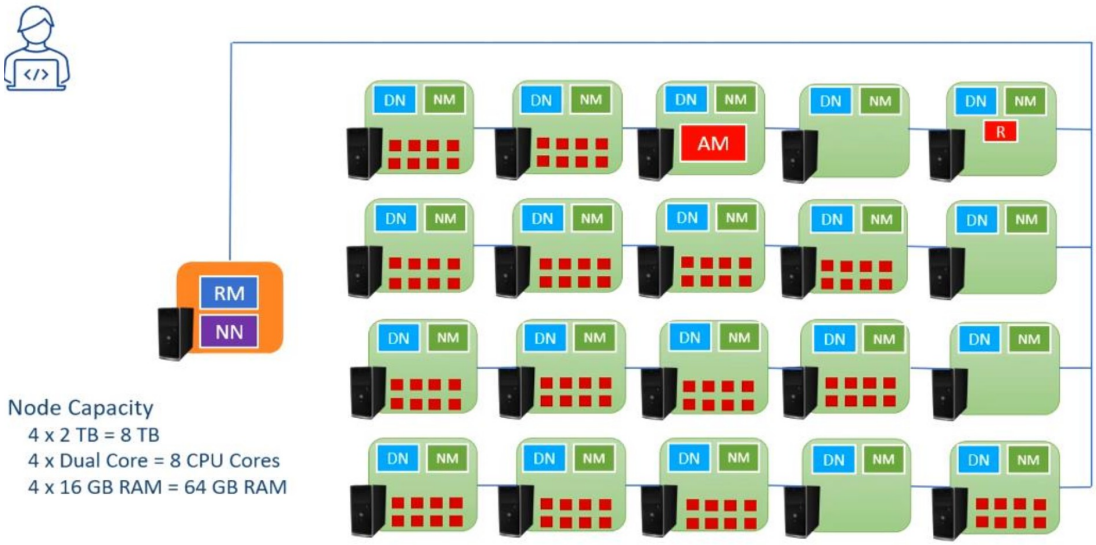
Map/Reduce:

Hadoop offered to solution to both problems we discussed in the previous slide. You can use the Hadoop cluster to store the file. Let's assume you have a 21 node Hadoop cluster. One node becomes the master, and the other 20 nodes are the workers. HDFS runs a name node in the master and a data node on the other workers. YARN runs a Resource Manager on the master, and Node Manager runs on the workers. So we have those services running on the cluster.



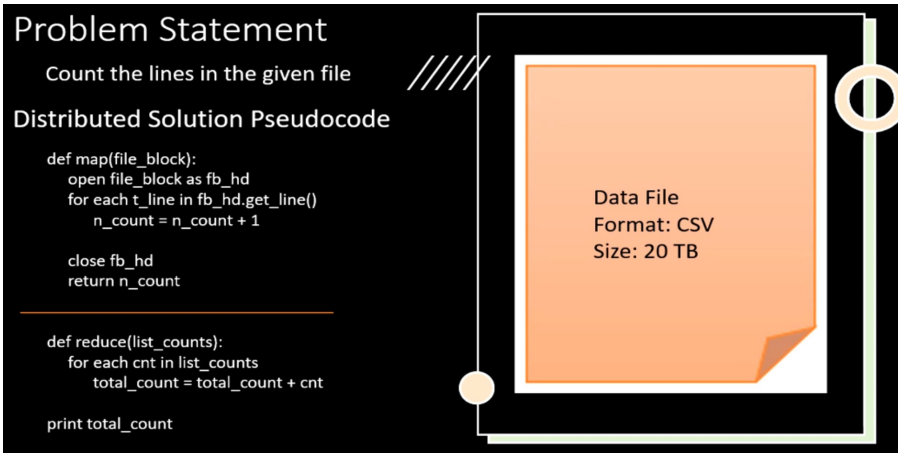
Map/Reduce:

You can use HDFS to copy your 20 TB file on this cluster. HDFS will break the file into small 128 MB blocks and spread them across the cluster. So some data nodes will sore data blocks, and altogether they can easily store your 20 TB file. Your storage problem is taken care of by the Hadoop cluster. If you need more storage, you can increase the cluster size and add more computers.



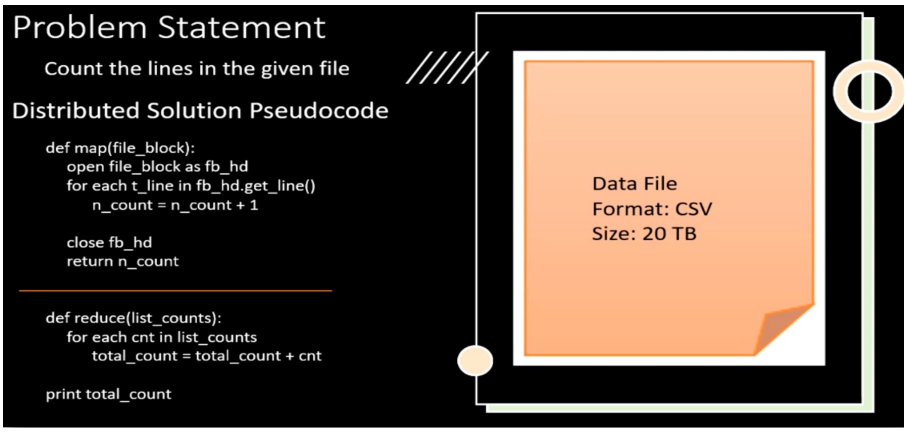
Map/Reduce:

Now let us come to the processing time problem. I have broken my logic into two part which you can see in the image below. The first part is known as the Map function. The second part is known as the Reduce function. The old logic was to open the file and count the lines. And the new logic is almost the same as old logic. But the map function opens the file block and counts the lines. And the old logic opens the file and counts the lines.



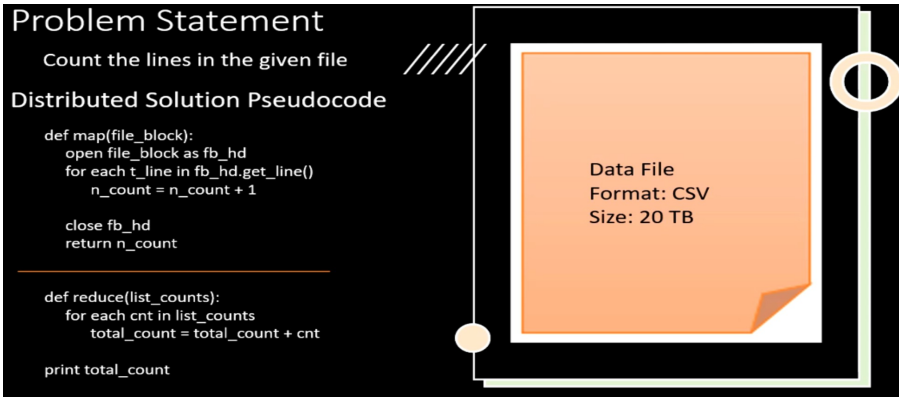
Map/Reduce:

I can run the map function on all the data nodes in parallel. This map() function will open each block on the data node and count the lines. End of the execution, I will have the number of lines in the blocks at the given data node. I am counting lines on 14 data nodes in parallel. Everything runs at the same time. And I will get the line counts in 1/14th of the time compared to doing it on a single machine. However, I will have 14 line counts. Each count represents the number of lines on their respective data node.



Map/Reduce:

Then, I will start a Reduce function at one node. All the data node will send their counts to the reduce function. The reduce function will receive 14 line counts in an array. So I will look through the array and sum up all the line counts. The reduce function will loop through the list of counts and sum it up. And the sum is the number of lines in the file.



Map/Reduce:

Here is the summarized context of Map Reduce.

Map Reduce Model

Implement logic in two functions

1. Map Function
   - Reads data block
   - Applies logic at block level
   - Map output is sent to Reduce
2. Reduce Function
   - Receives Map output
   - Consolidates the results

Hadoop M/R framework implement the map-reduce model.

- YARN manages resource allocation
- HDFS manages data blocks

The three Big Data problems: 1. High Data Volume 2. Variety of Data 3. High speed

Google was the first company to realize the big data problem. And they were also the first company to develop a viable solution and establish a commercially successful business around it.

Google had four main problems to solve which are listed below. They were creating a search engine, and the first thing they wanted to do was to discover and crawl the web pages over the Internet and capture the content and metadata. We now categorize these problems as data collection and data ingestion. Once the webpage-related data is collected, they want to store and manage it. So the next problem was to store and manage hundreds of petabytes of data. The next problem was to get the computation power for processing those massive volumes. They wanted to apply the PageRank algorithm to the received data and create an index. So, the third problem was to get the required processing power. Finally, the last problem was organizing and storing the outcome of the processing. In Google's case, the result was the index. They wanted to keep it in a random-access database to support high-speed queries by the Google Search Engine application.

Google successfully solved all the four problems, and they were generous enough to reveal their solution to the rest of the world in a series of white papers. Google published the first whitepaper in 2003, and it talked about solving the Data Storage and Management problem. They termed the solution as Google File System (GFS).

The second whitepaper was published by Google in 2004 and talked about the Data Processing and Transformation problem. They termed it as MapReduce (MR) programming model.

1. Google File System – 2003
   https://ai.google/research/pubs/pub51
2. MapReduce – 2004
   https://ai.google/research/pubs/pub62

The open-source community well appreciated these whitepapers, and they formed the basis for the design and the development of a similar open-source implementation – The Hadoop. The open-source community implemented the GFS as a Hadoop Distributed File System – HDFS. They applied Google MR as the Hadoop MapReduce programming framework.

Hadoop grabbed immense attention and popularity among organizations and professionals. Since the development of Hadoop, there have been many other solutions developed over the

Hadoop platform and open-sourced by various organizations. Some of the most widely adopted systems were Pig, Hive, and HBase.

Apache Hive was one of the most popular components of Hadoop. Hive offered the following core capabilities on the Hadoop Platform: 1. Create Databases, Tables, and Views 2. Run SQL Queries on the Hive Tables

So Hive simplified using Hadoop. Application developers struggled to solve data processing problems using Map Reduce. Hive came to the rescue. It allowed us to create databases and tables using DDL Statement. Then they also allowed us to use SQL queries on the table.

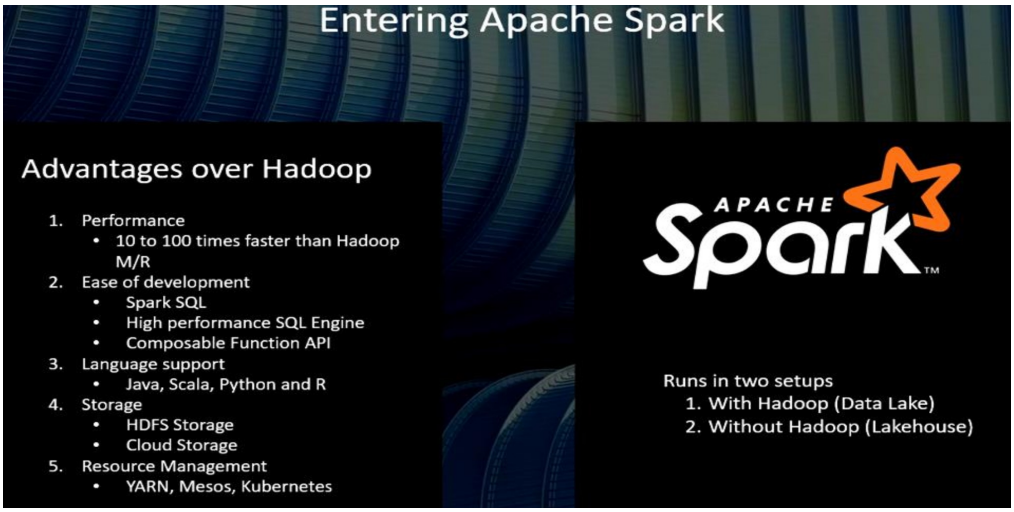
The majority of the development workforce was familiar with the RDBMS, and they already knew SQL. So using SQL was easy to adopt.

Hive SQL engine internally translated SQL queries into M/R programs. But application developers were saved from writing Map Reduce code in Java.

Hadoop as a platform and Hive as a Hadoop database became very popular. But we still had the following problems which needed improvements:

1. Performance
2. Ease of development
3. Language support
4. Storage
5. Resource Management

Apache Spark comes to rescue.



Spark exists on two kinds of platforms:

1. On Hadoop Platform - Data Lake
2. On Cloud Platform - Lakehouse

We use the Hadoop platform as the Data Lake platform, and the primary developer technology

on Hadoop Data Lake is now Apache Spark. Map/Reduce Framework is gone away forever, and Hive is also losing its place for Spark SQL.

The Cloud platforms are more popular these days. So the idea of Hadoop Data Lake is now advanced and modernized with a new name of Lakehouse on the cloud platforms. The driving force behind Cloud Lakehouse is the Databricks Spark platform. So Spark is again the primary developer technology for Lakehouse.

Data Lake – Emergence and Use

We know that the distributed computing started with Google finding a solution for their storage requirements using GFS. The open-source community created a similar solution called HDFS that allowed us to form a cluster of computers and use the combined capacity for storing our data. Then we also got the MapReduce framework which allowed us to use the combined computing power of the cluster and use it to process the enormous data volumes that we stored in HDFS.

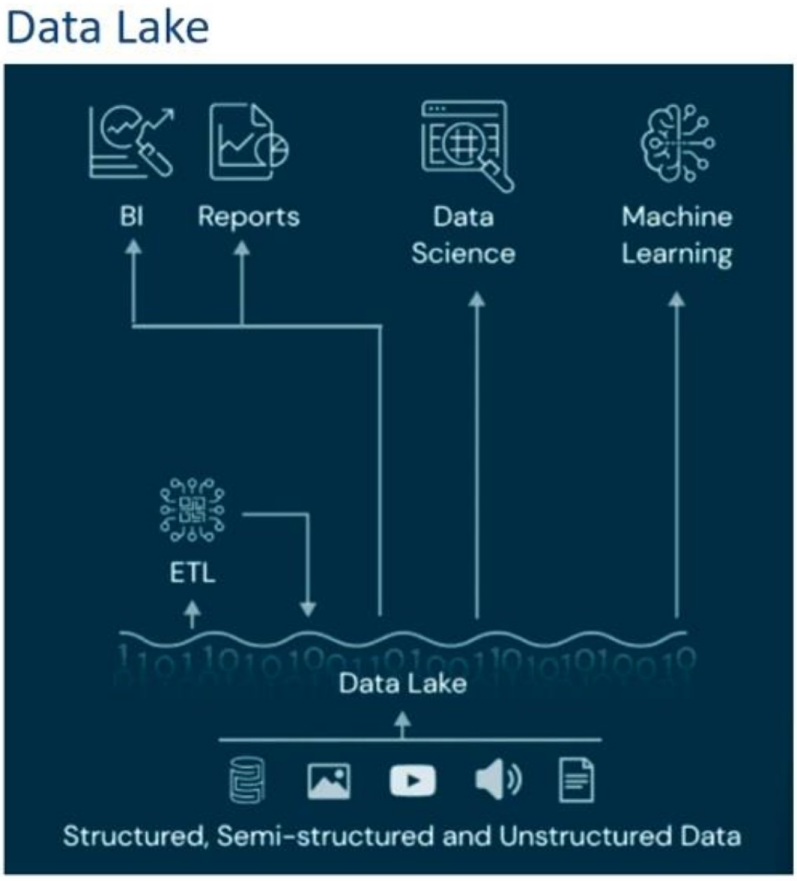
Before the HDFS and Map/Reduce came into existence we had Data Warehouses - Like Teradata and Exadata. We created pipelines to collect data from many OLTP systems and brought them into the Data Warehouse. Then we processed all that data to extract business insights and used it to make the correct business decision.

Hadoop also offered to collect data and process it to extract business insights. So the advent of HDFS and MR started challenging these Data Warehouses in three critical respects:

1. Ease of Horizontal Scalability
2. Capital Cost
3. Volume and Variety of Data

Here is a comparison between Data Warehouse and Hadoop offerings.

Data Warehouses were challenged, and they needed a new name for the Hadoop approach. And it is when the Data Lake came into existence.

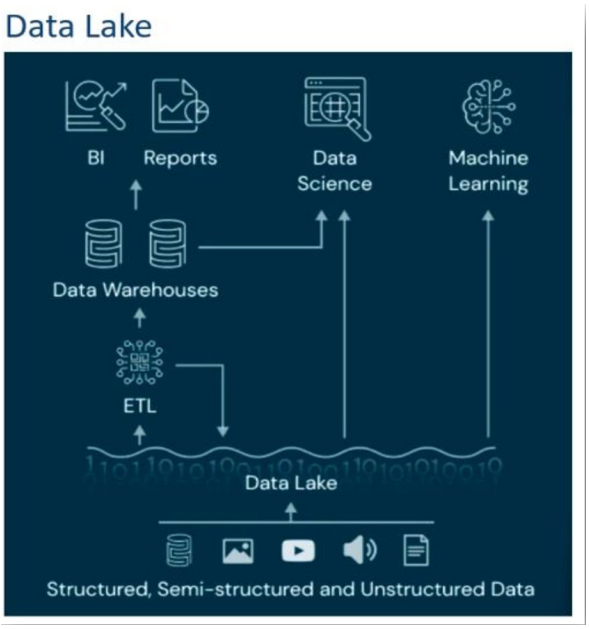


Same as Data warehouses, we collected data from different data sources and stored them in HDFS. Then we used Map Reduce and Spark to process this data and prepare new data models for generating reports and business insights. Spark took over the Map/Reduce and other tools over a period of time, so it is safe to consider that we used Spark to process and prepare data for reporting.

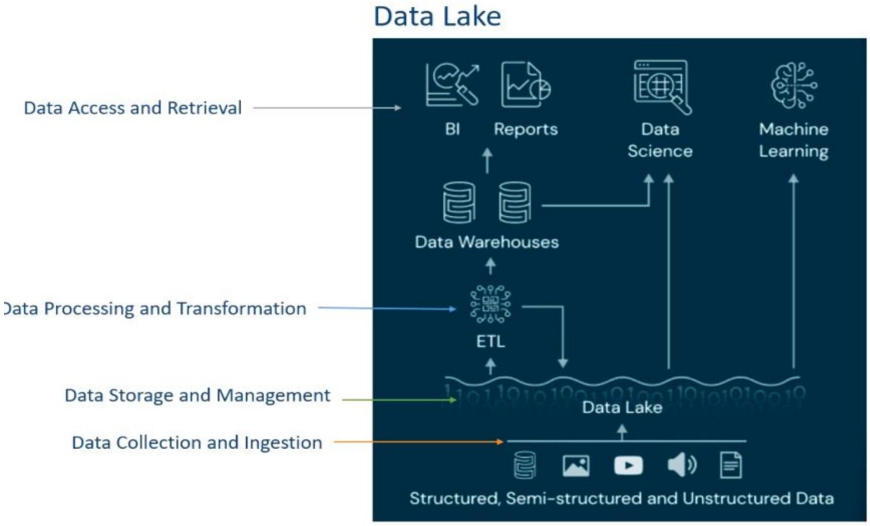
This processed data was also stored in Data Lake storage for business intelligence and reporting. Most of the popular BI and reporting tools offered a connector to access data from the data lake. So, Data Lake also allowed us to collect and process huge volumes of semi-structured and unstructured data.

However, the Data Lake technology missed two supercritical features that Data warehouses offered: 1. Transaction and Consistency 2. Reporting Performance

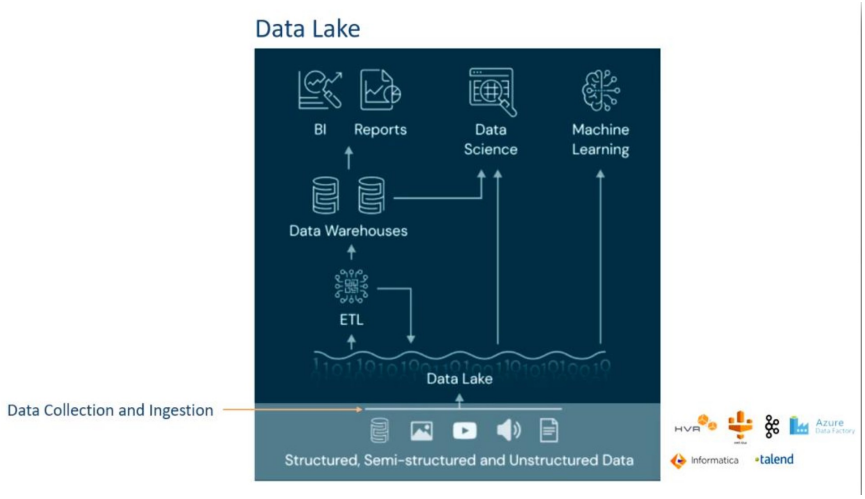
So, we adopted a different architecture for implementing Data Lakes. We collected data in Data lake storage, proceeded it using Apache Spark, and stored the result in a Data Warehouse. And finally, we connected the BI and Reporting with the Data Warehouse.



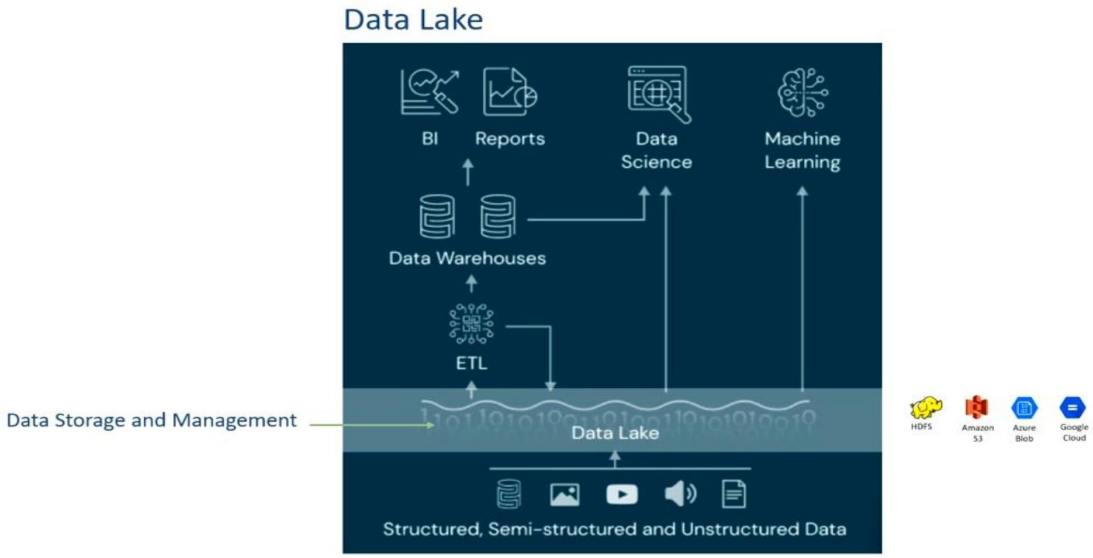
This new architecture also matured as a platform with four key capabilities. And these are nothing but the same problems that Google attacked and solved.



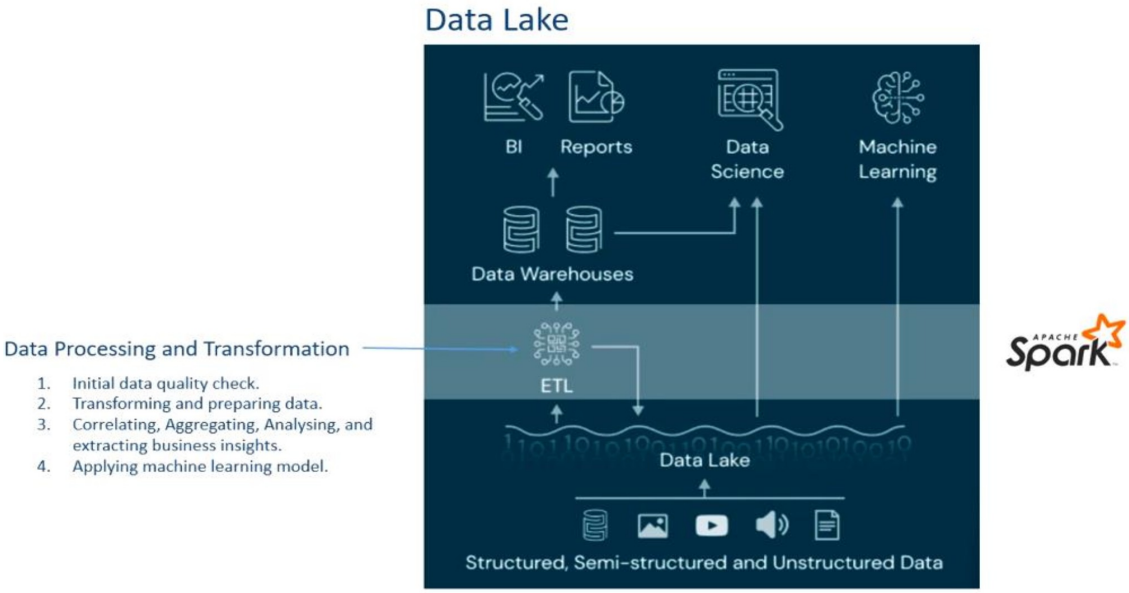
The ingest block of the data lake is all about identifying, implementing, and managing the right tools to bring data from the source systems to the data lake. And we have many vendors competing for a place in this box.



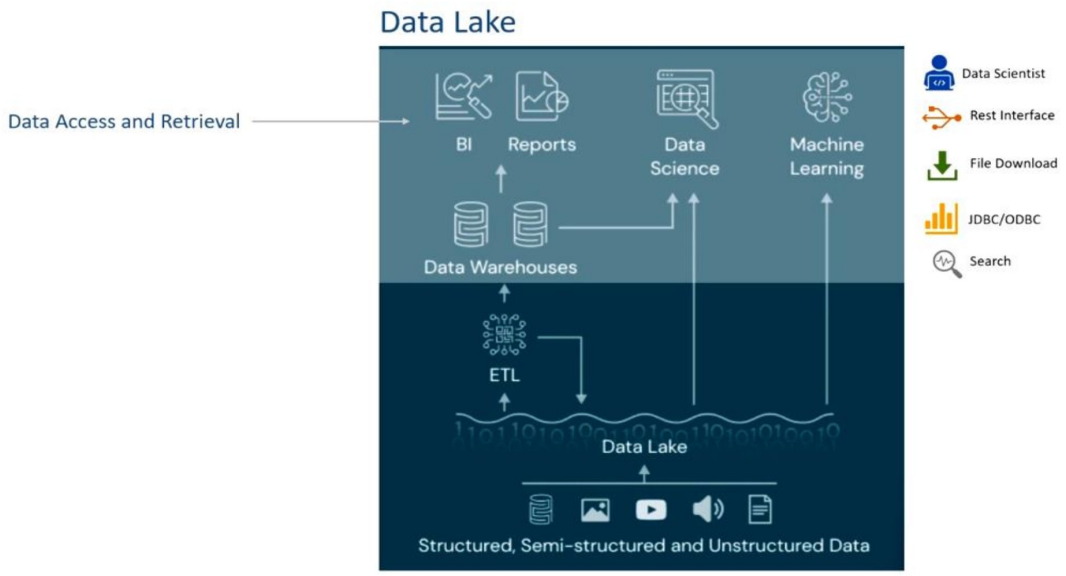
The core of the data lake platform is the storage infrastructure. In today's data lake, this could be an on-premise HDFS or Cloud Object Stores such as Amazon S3, Azure Blob, Azure Data Lake Storage, or Google Cloud Storage. Cloud storage is leading because they offer scalable and high availability access at an extremely low cost in almost no time to procure.



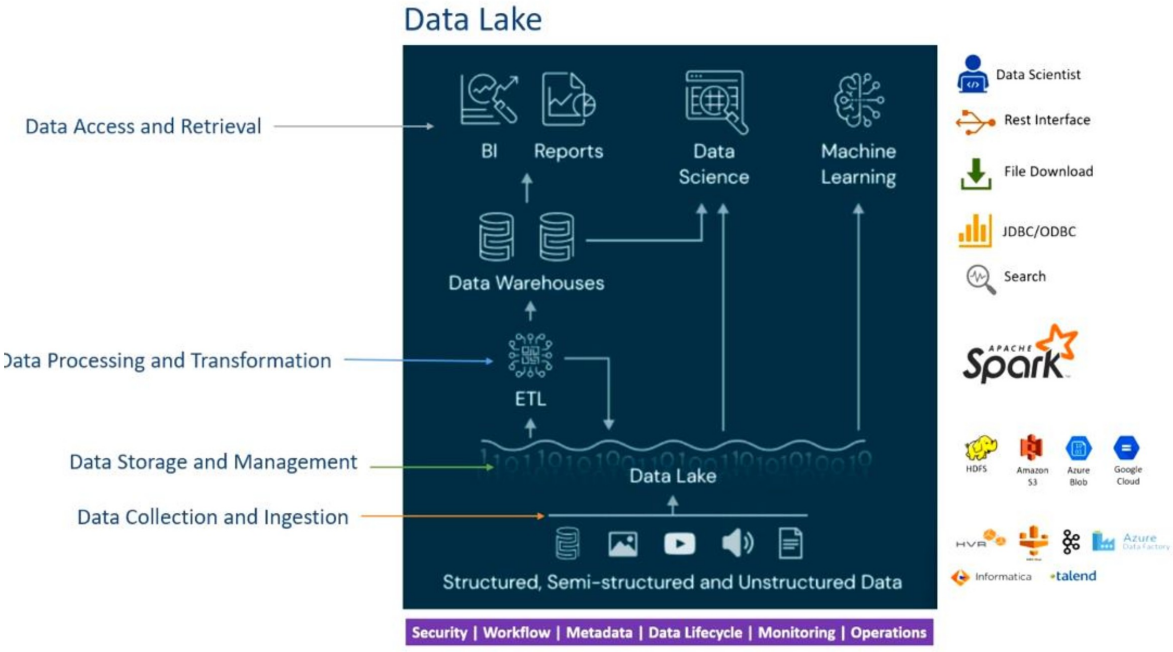
The next layer is the processing layer. This is the place where all the computation is going to happen. When I say computation, it means the following kind of work listed in the image below.



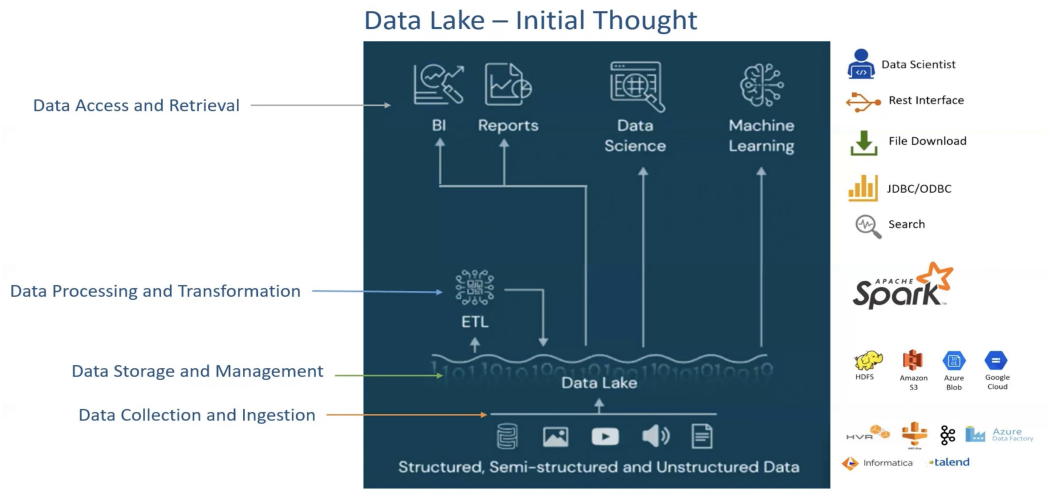
The last and most critical capability of the data lake is to allow you to consume the data from the data lake. You can think of your data lake as a repository of raw and processed data. Now the consumption is all about putting that data for real-life usage, which can be of various types.



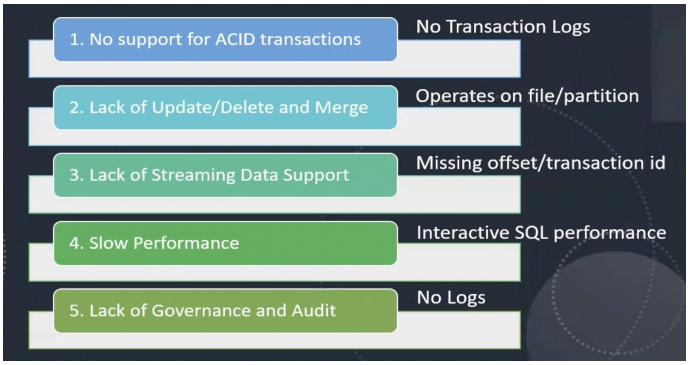
This is the complete Data Lake architecture.



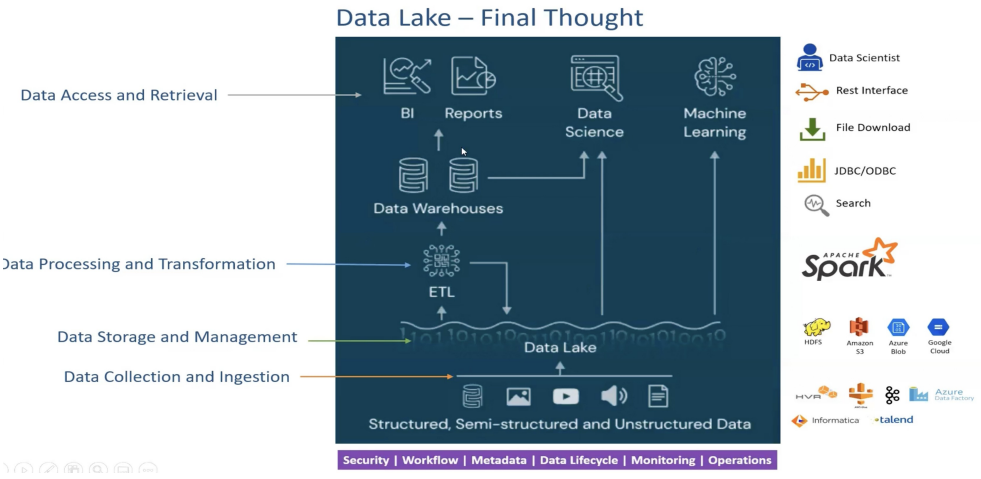
Datelake vs Lakehouse Architecture



Problems (missing features) with the above Datalake architecture (present in Data-warehouses) – from 1:06:16 to ... in this recording (https://www.scholarnest.in/courses/take/spark-tables-and-dataframes/multimedia/35064706-week-1-review-doubts-datalake-vs-lakehouse)



To solve above issues, they combined Datalkae and Warehouse and came up with the below architecture.



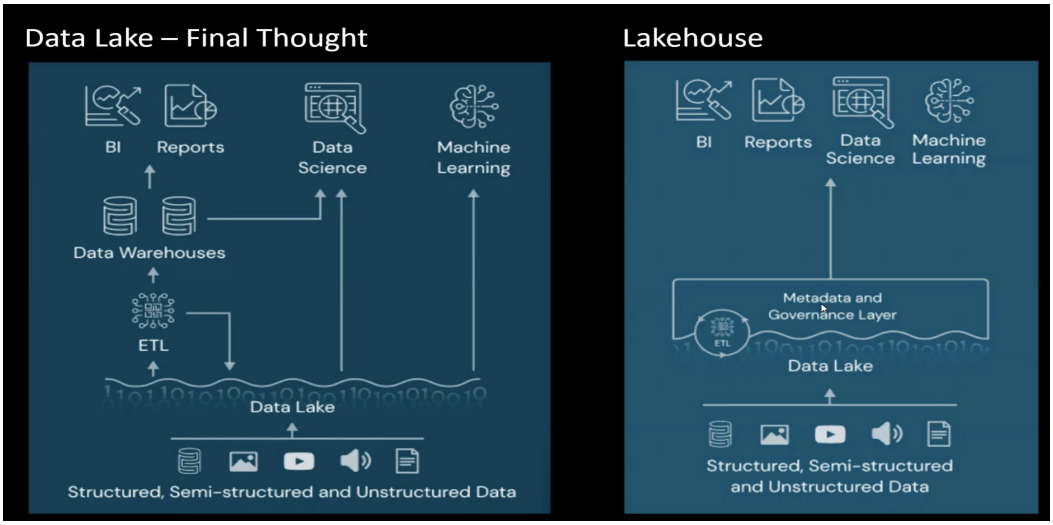
This solved above issues with the original Datalake architecture, but still had below issues – only performance issues solved by introducing DWs + DLs.

Cost & Management Problem

Duplication & Sync Problems

Consistency & Reliability Problems

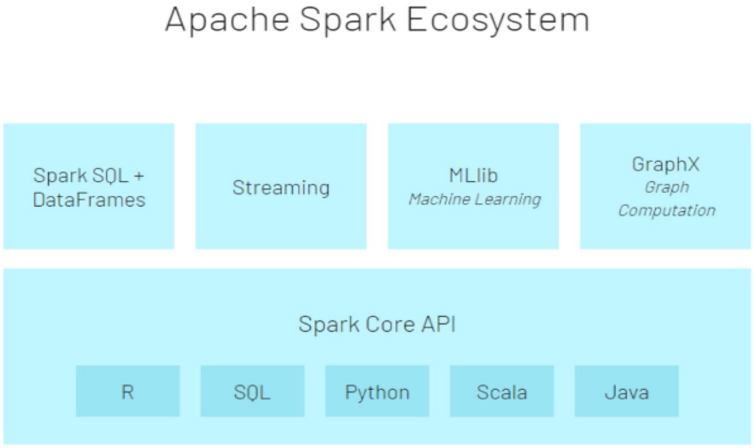
If any Datalake can support transactions, all the 4 problems out of 5 will be solved. And remaining is performance, which can be tackled separately. Databricks came up with the inovative solution of Lakehouse architecture with Metadata and Governance Layer (Delta tables / format) solving all the 4 problems except performance. SQL Performance is tackled by Databricks separately with “Photon” engine.



Apache Spark and Databricks Cloud – Introduction

Apache Spark is a distributed data processing framework, and this diagram represents the Spark Ecosystem. The Spark ecosystem is designed in two layers:

1. The top layer is a set of DSLs, libraries, and APIs.
2. The bottom layer is the Spark Core Layer.

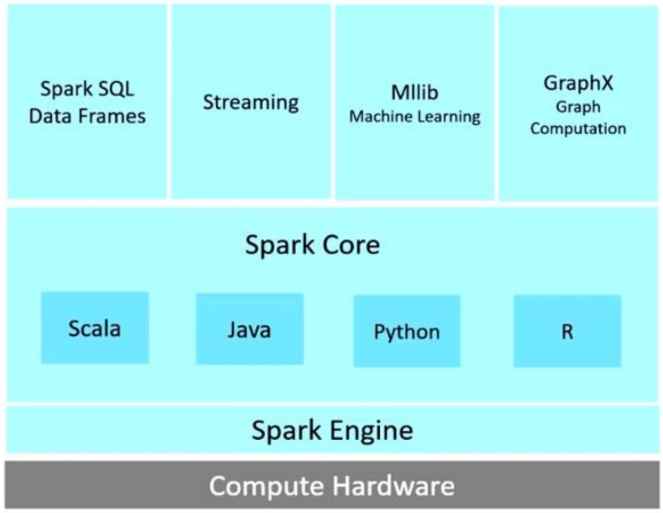


The Spark core layer itself has got two parts:

1. A distributed Computing Engine

2. A set of Core APIs

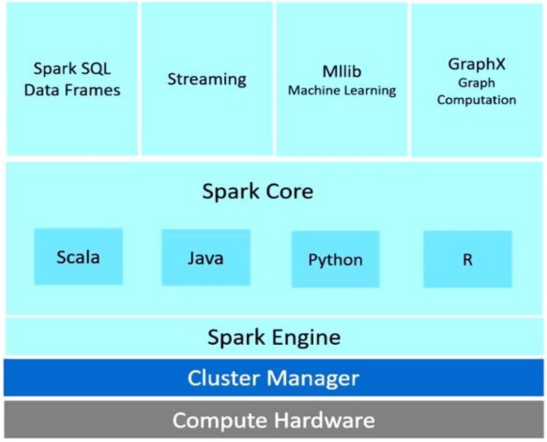
And this whole thing runs on a cluster of computers to offer you distributed data processing. However, Spark does not manage the cluster. It only gives you a data processing framework.



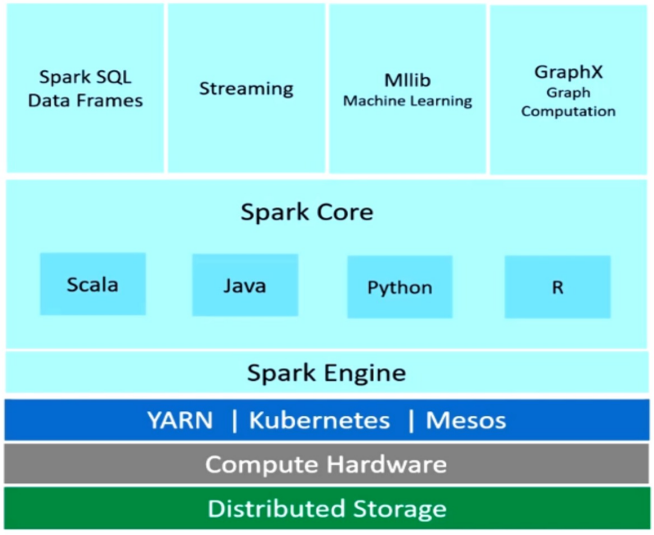
The Spark core layer itself has got two parts:

1. A distributed Computing Engine
2. A set of Core APIs

And this whole thing runs on a cluster of computers to offer you distributed data processing. However, Spark does not manage the cluster. It only gives you a data processing framework. Some examples of Cluster Manager are: YARN, Mesos, Kubernetes



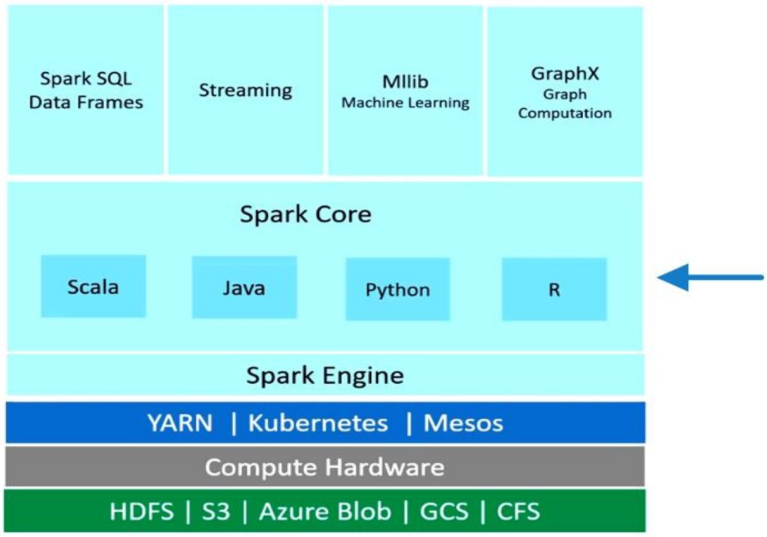
Spark also doesn't come with an in-built storage system. And it allows you to process the data, which is stored in a variety of storage systems. The most popular and commonly used storage systems are HDFS, Amazon S3, Azure Data Lase Storage, Google Cloud Storage, and the Cassandra file system.



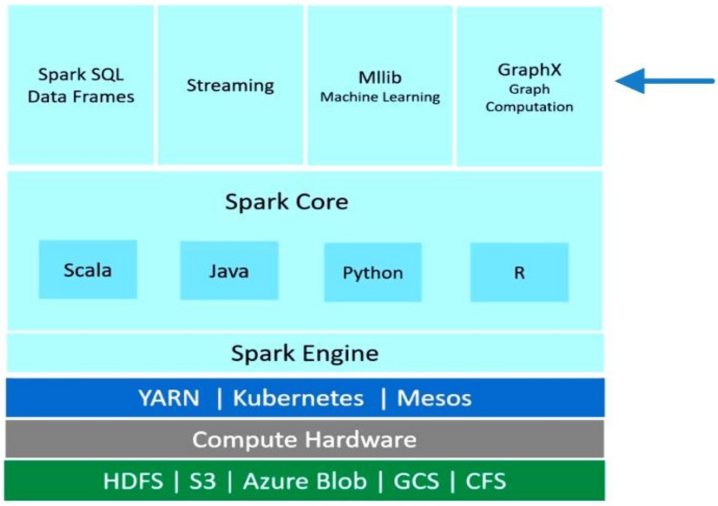
Apache Spark does not offer Cluster Management and Storage Management. All you can do with Apache Spark is to run your data processing workload.

And that part is managed by the Spark Compute Engine. So the Spark compute engine is the core that runs and manages your data processing work and provides you with a seamless experience. All you need to do is submit your data processing jobs to Spark, and the Spark core will take care of everything else.

The Core API layer is the programming interface layer that offers you the core APIs in four major languages: Scala, Java, Python, and R. These are the APIs that we used to write data processing logic during the initial days of Apache Spark. However, these APIs were based on resilient distributed datasets (RDD). These APIs are the most complicated way to work with Apache Spark and they also lack some performance optimization features. However, these APIs can offer you the highest level of flexibility to solve some complex data processing problems.



The topmost layer is the prime area of interest for most Spark developers and data scientists. It is a set of libraries, packages, APIs, and DSL. These are developed by the Spark community over and above the Core APIs. The topmost API layer is grouped into four categories to support four different data processing requirements. However, this is just a logical grouping, and there is no rigid boundary.



Let us look at the four categories of the top most API layer: 1. Spark SQL and DataFrame/Dataset APIs - Spark SQL allows you to use SQL queries to

process your data. Both of these together can help you resolve most of the structured

and semi-structured data crunching problems. 2. Spark Streaming libraries - They allow you to process a continuous and unbounded

stream of data.

3. A set of libraries specifically designed to meet your machine learning, deep learning, and AI requirements.
4. Graph Processing libraries, and they allow you to implement Graph Processing Algorithms using Apache Spark.

So the topmost layer is nothing but a set of libraries and DSLs to help you solve your data crunching problems. And all these are available in multiple languages such as Java, Scala, Python, and R.

There are three reasons why Spark Ecosystem is so popular: 1. Abstract - Spark will abstract away that you are coding to execute your

program on a cluster of computers. So, all the complexities of distributed storage, computation, and parallel programming, is abstracted away by the Spark core.

2. Unified Platform - Spark combines the capability of SQL queries, Batch Processing, Stream Processing, Structured and semi-structured data handling, Graph processing, machine learning, and deep learning. All of this in a single framework using your favourite programming language. You can mix and match them to solve many sophisticated requirements.

3. Ease of Use - Comparing it with old Hadoop and MapReduce code, Spark code is much shorter, simpler, and easy to read and understand.

Now let’s come to Databricks. Apache Spark is an open-source project. The original creators of Apache Spark donated it to Apache Foundation and made it an Open Source project. However, that same team formed a company and a commercial product around Apache Spark. The company and the product are both named Databricks. And Databricks offers the following things over and above Apache Spark:

1. Spark on Cloud Platform
2. Spark Cluster Management
3. Notebooks and Workspace
4. Administration Controls

5. Optimized Spark
6. Databases/Tables and Catalog
7. Databricks SQL Analytics
8. Delta Lake Integration
9. ML Flow
10. Industry vertical solutions and accelerators

Spark Development Environments

Spark projects are developed and deployed in two kinds of environments.

Cloud Platforms

On-premise Platform

We have two standard methods to set up your Spark development environment

Notebook

Python IDE

If your project is going in Cloud, you should prefer Notebook. And an on-premise project prefers to use Python IDE. However, it is not mandatory. Cloud platforms also allow you to develop on your local machine and deploy it in Cloud.

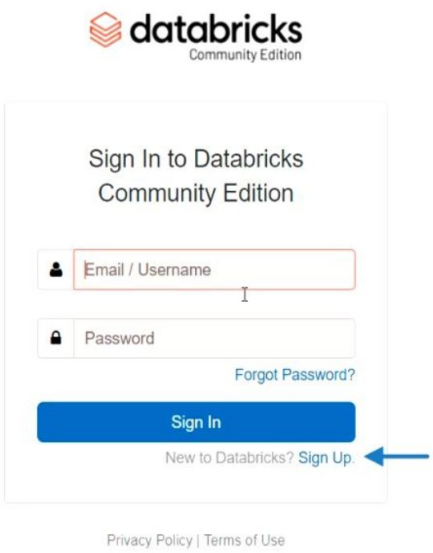
The first and the easiest method to get Spark is to use Spark Notebooks in the Databricks Cloud environment. However, you should also learn to set up a Local Spark development Environment.

Notebook

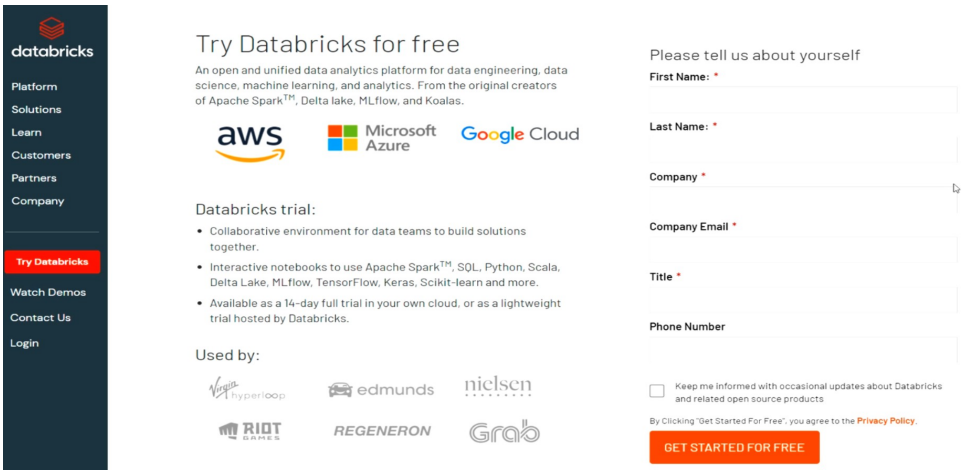
Python IDE

Databricks Community Account

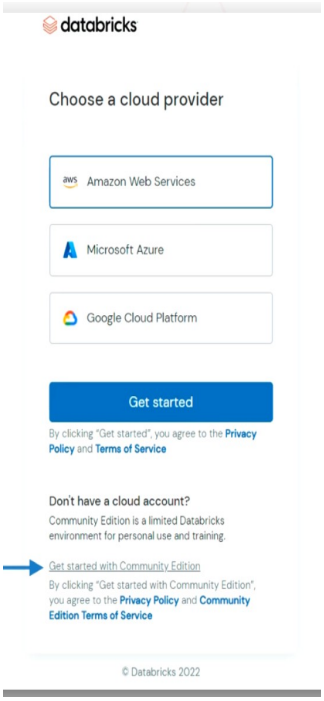
Visit https://community.cloud.databricks.com/login.html Click the Sign-Up Button



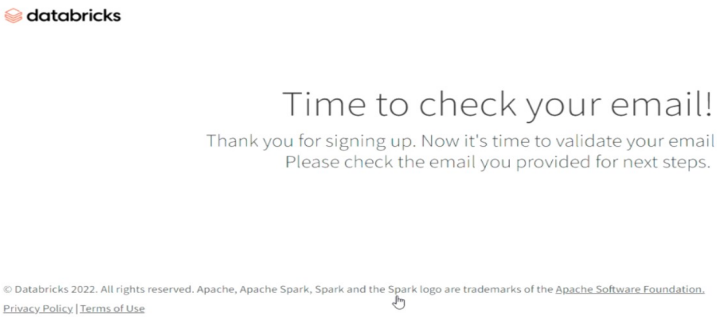
Databricks Cloud is available in AWS, Microsoft Azure, and Google cloud platforms. If you already have an account in any of these cloud platforms, you can get 14 day free trial. However, we will be using a completely free lightweight community version of the Databricks Cloud. But you should register for the account creation. Fill out the details below and click the get started for free button.



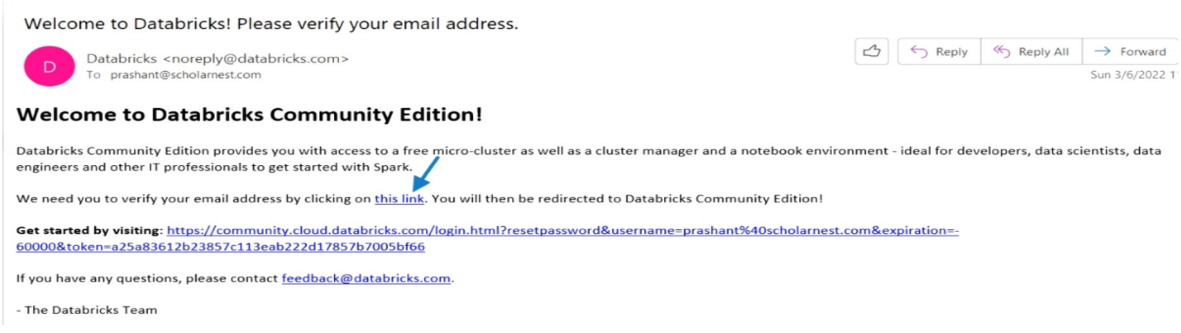
They are asking to choose the cloud environment. We will not choose any of these three cloud environments because that option is for availing of 14 day free trial. We wanted to sign up for a free community edition. So you should go down and click the community edition link.



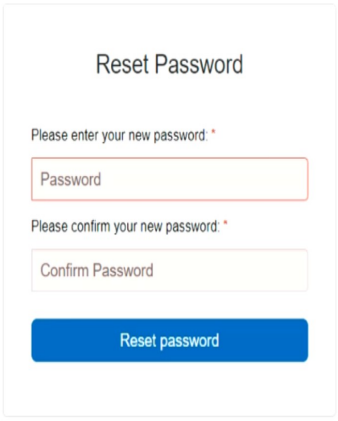
They might ask for captcha verification, so go ahead and verify that you are a human. Once your captcha verification is complete, they will send you an email.



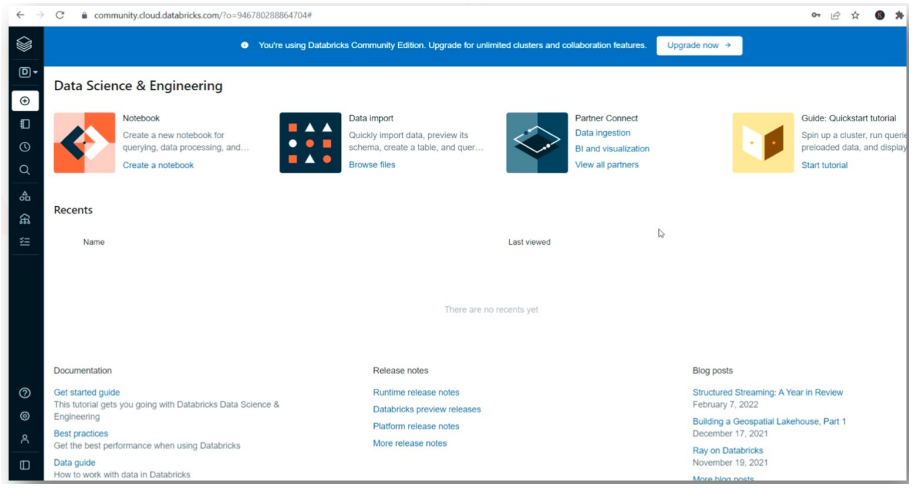
Login to your email box and check the email from Databricks. Below is an example email. You should expect a similar email from Databricks. Make sure you are checking your spam folder and found this email. Once you find the email, click the link provided in the email.



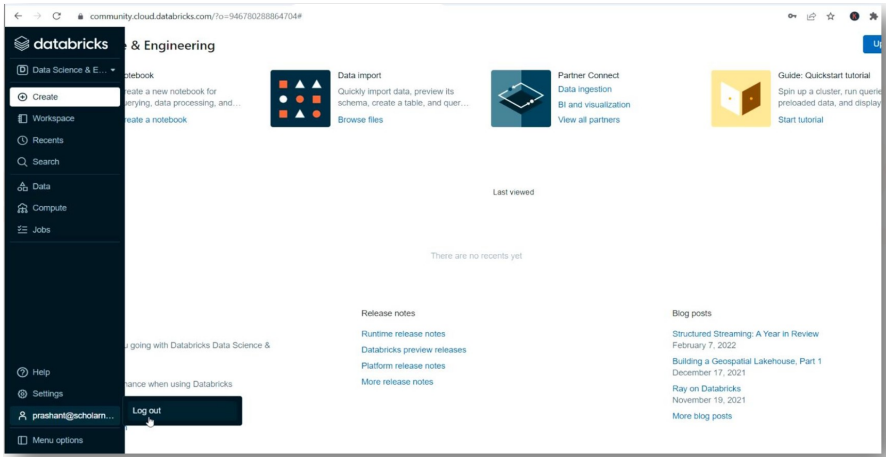
The confirmation link takes you to the password reset page. Set your password and confirm it.



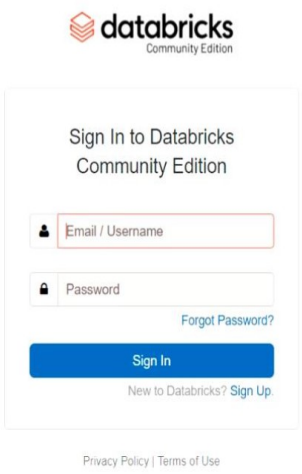
The password reset will take you inside the Databricks Cloud Workspace. Below page is the Databricks Cloud Workspace home page.



You can log out from the workspace and close everything.



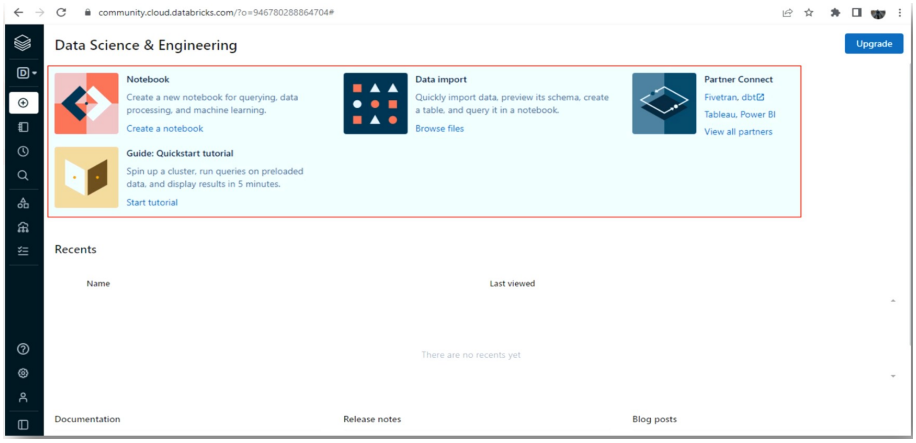
You can visit https://community.cloud.databricks.com/login.html And login using your credentials. I recommend that you bookmark this page.



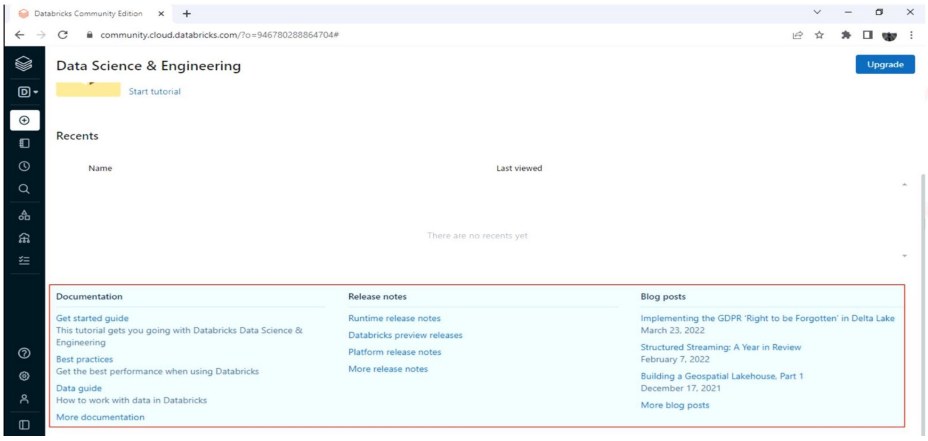
Databricks Workspace Introduction

The below screenshot shows Databricks Workspace home page. In the center of the page, you have some shortcuts for the most common tasks.

1. The first link is to create a new notebook and start writing some code.
2. The second link is to import data files.
3. The next one allows you to use some partner tool.
4. The last link is for a quick start tutorial.

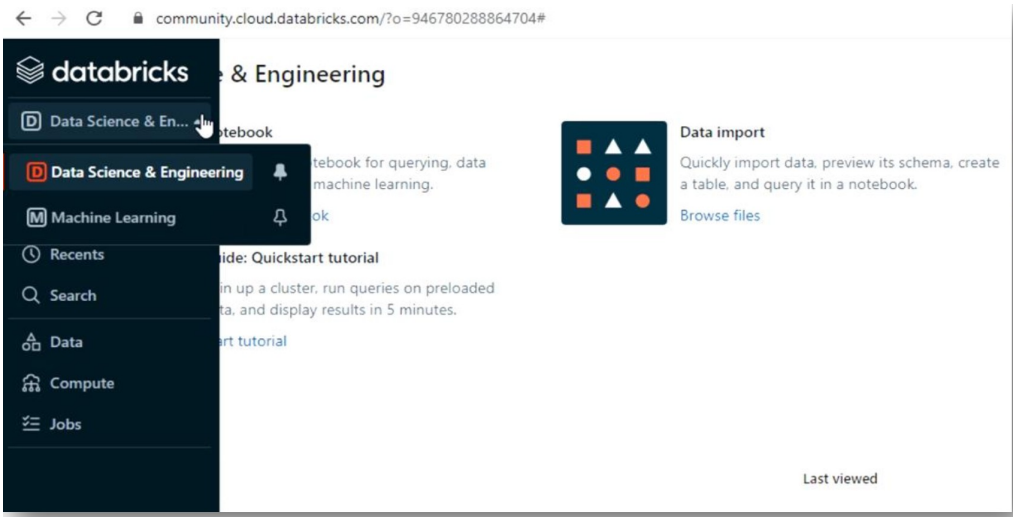


You also have some links at the bottom. These are mainly Databricks documentation and blog posts.



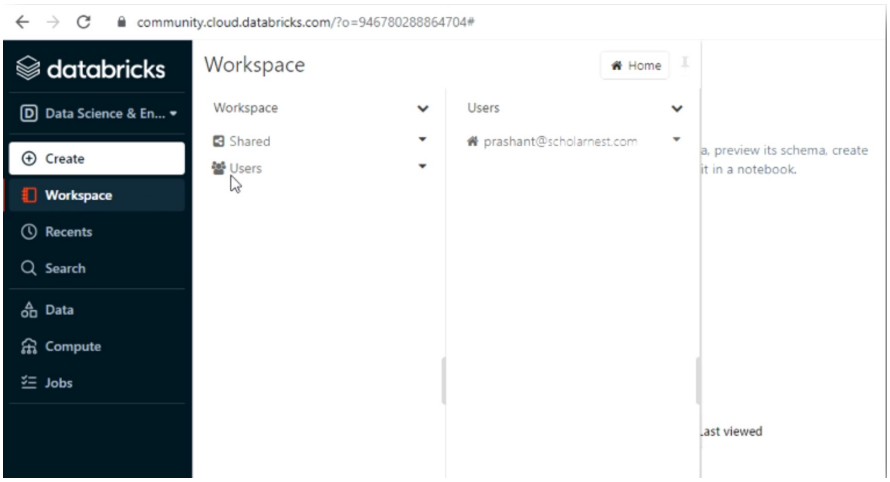
The main functionality is hidden behind the left side menu. The first item in the navigation menu allows you to choose workspace type. You can see two workspace types.

1. The first one is Data Science and engineering
2. The second workspace type is for machine learning professionals.



Up next, you will see the create button on the menu. When you click on this button, it will show you options to create a Notebook, Table, and Cluster. We create and use these three main things while working with the Databricks community.

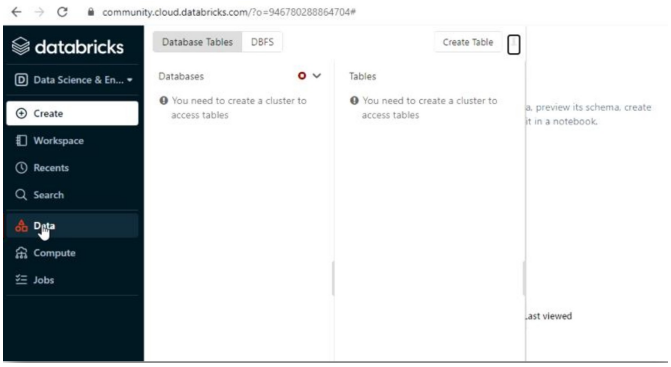
The following item in your menu option will show your workspace. So, this workspace here is a set of folders and files assigned to each user. We only have one user here, so the workspace shown below is my workspace.



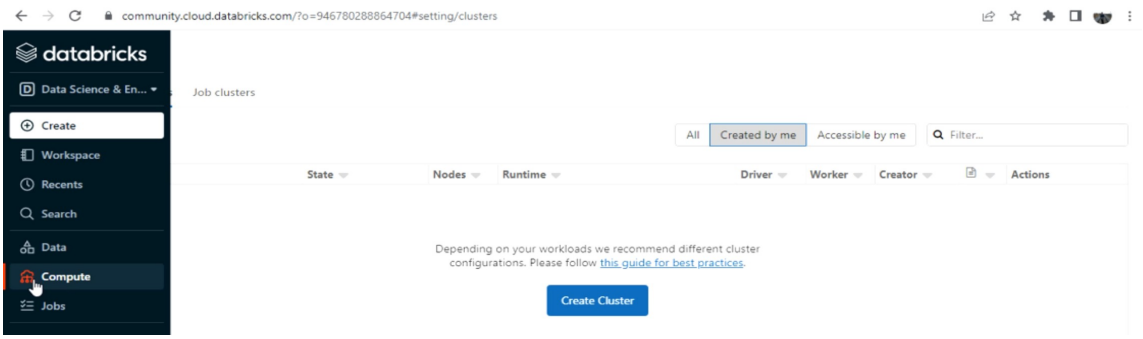
The next two items in your menu options are:

1. Recents – It shows a list of recent items used in the Databricks Workspace.
2. Search - It allows you to do a full-text search in your Databricks workspace.

The next item is for data. It allows you to create Spark Databases and Tables. Even if you create databases and tables using your program, they will appear here.

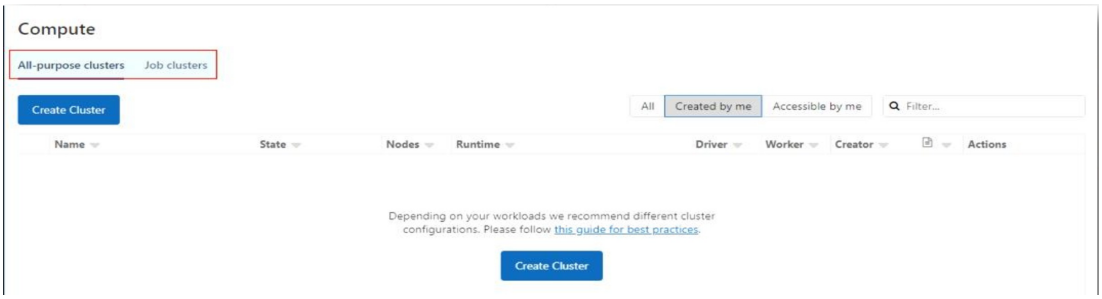


The following item is the Compute menu. This page allows you to create, launch and manage spark clusters. Creating a database and table requires some computation power. So you can create them only when you have a Spark cluster.

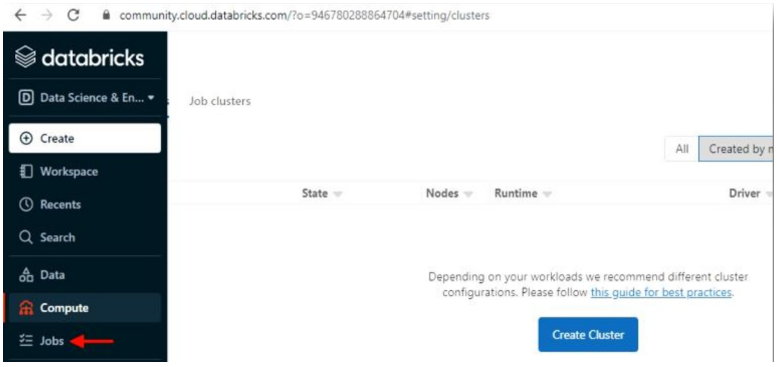


You can create two types of clusters in compute menu:

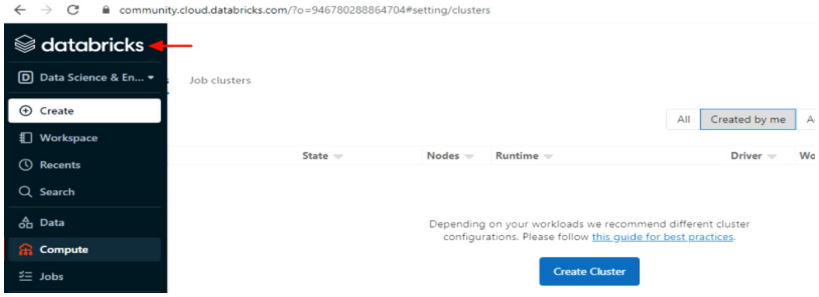
1. All purpose clusters – They are suitable for development and testing activities.
2. Job Clusters - These are short-term on-demand clusters, which launch a Spark application, run it, and automatically shuts down once your application finishes. It is not available for the community edition.



The following menu item is for creating Spark Jobs. However, the Spark Jobs feature is not available in the community edition.



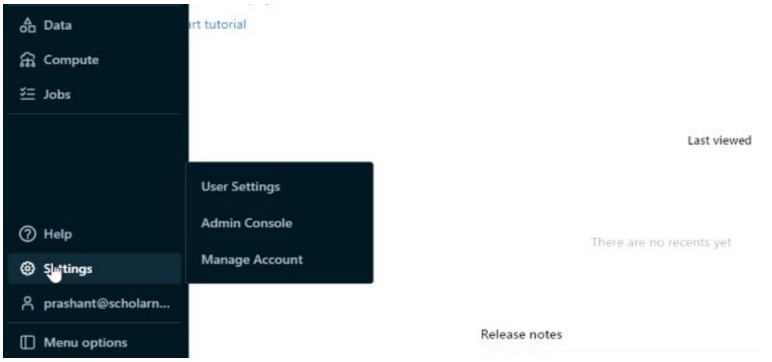
You can click on this “databricks” logo shown below to return to the workspace homepage.



You also have a few more things in the menu down below, such as Help, Settings, Logout Link, and Menu Options. If you go to the settings option, you can see 3 links:

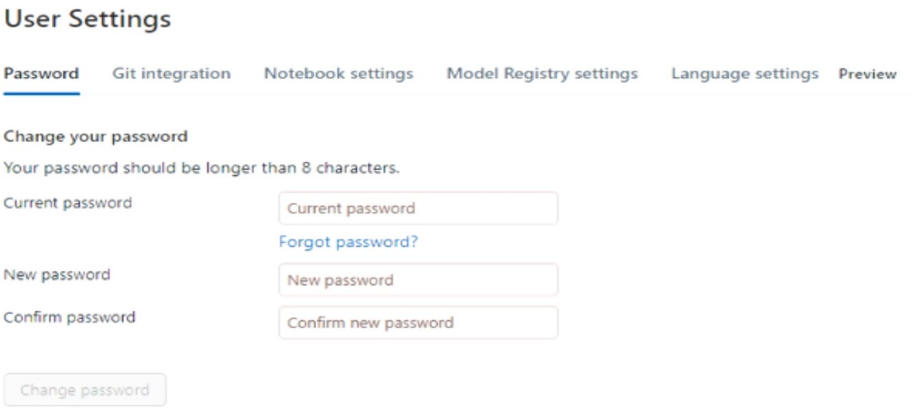
1. User Settings

2. Admin Console
3. Manage Account

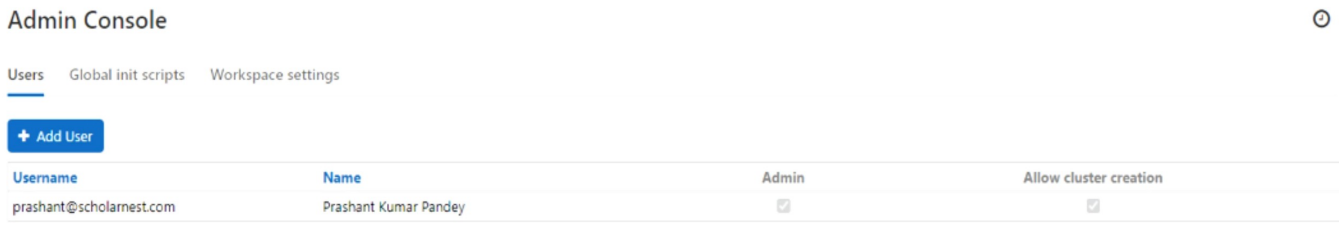


If you go inside user settings from the main settings option, you will see the following alternatives.

1. You can change your password from here.
2. You can manage your Git integration.
3. You have some notebook settings
4. There is some model registry setting
5. There are some language settings as well

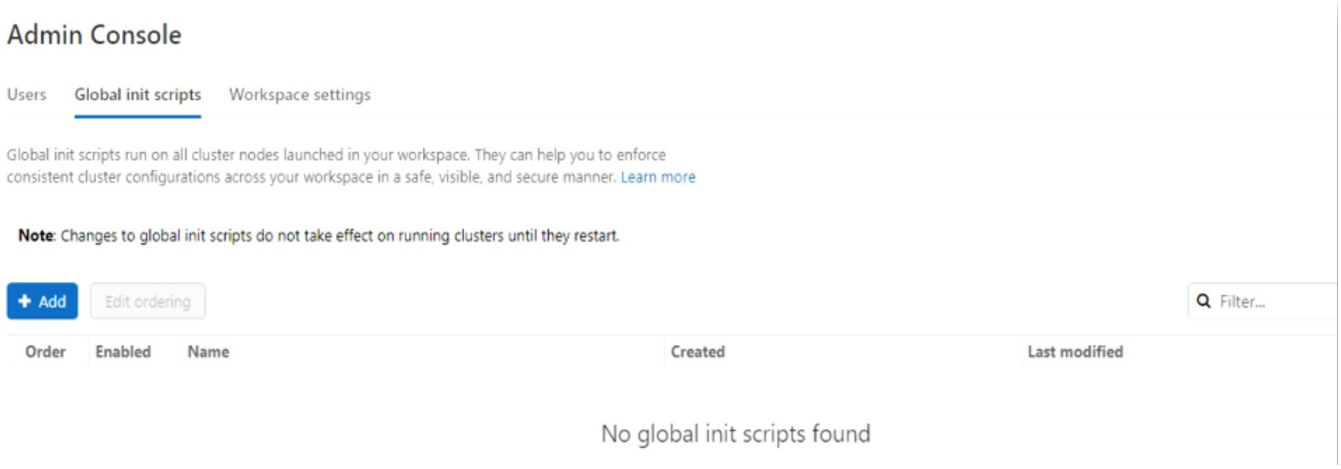


If you go inside admin console from the main settings option, you will see a screen as shown below. I created this workspace, so I became the admin for this workspace. However, you can invite other users by clicking the add user button shown below.



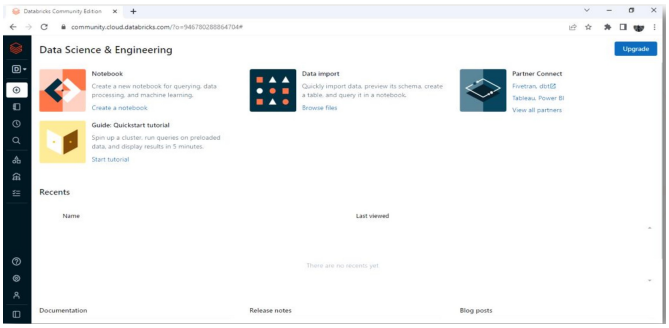
There are a couple of more options available inside the admin console.

1. Global Init Scripts – It is known as cluster node initialization scripts, which is basically a shell script that runs during the start-up of each cluster node before the Apache Spark driver or worker JVM starts. You can use these scripts to configure your cluster and install other things not included in the Databricks runtime.
2. Workspace Settings – You can use this option to customize your workspace.

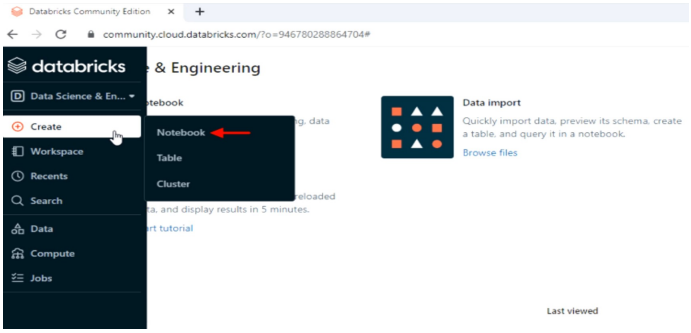


First Spark Application in Databricks Cloud

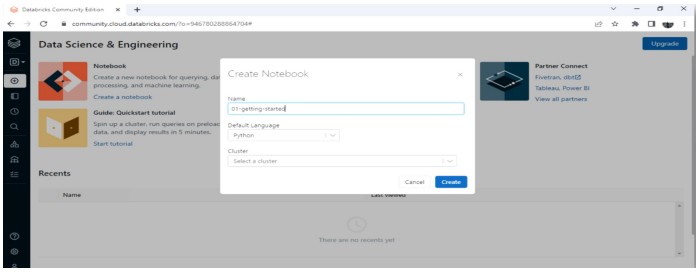
Login to your Databricks Workspace.



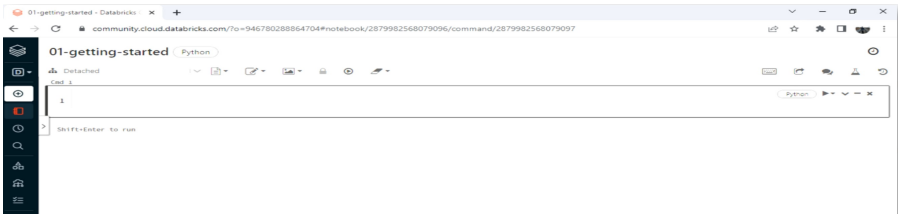
Come to the navigation menu and hit the create button. You will see three options there: Notebook, Table and Cluster. Hit the notebook option and start creating a notebook.



Give a name to your notebook and hit the create button to get started.

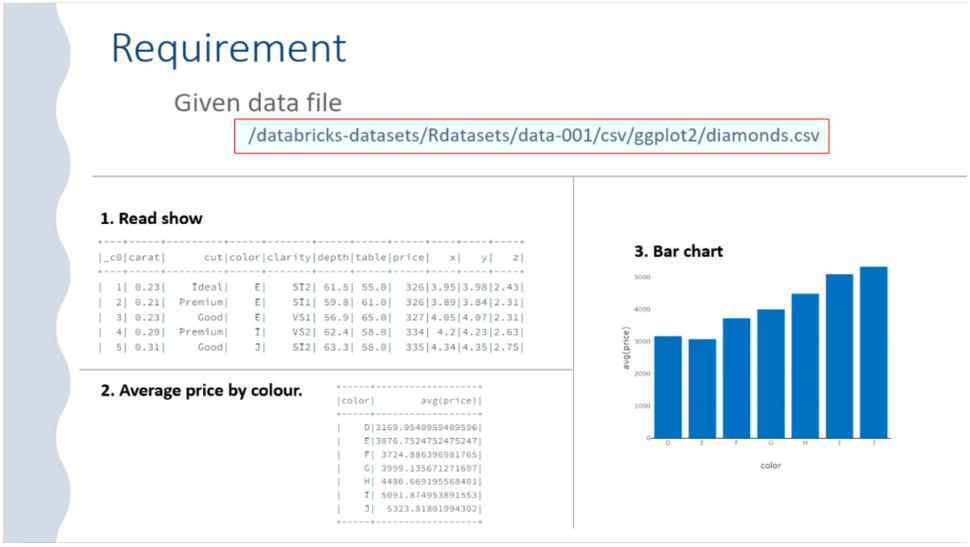


Here is your notebook.



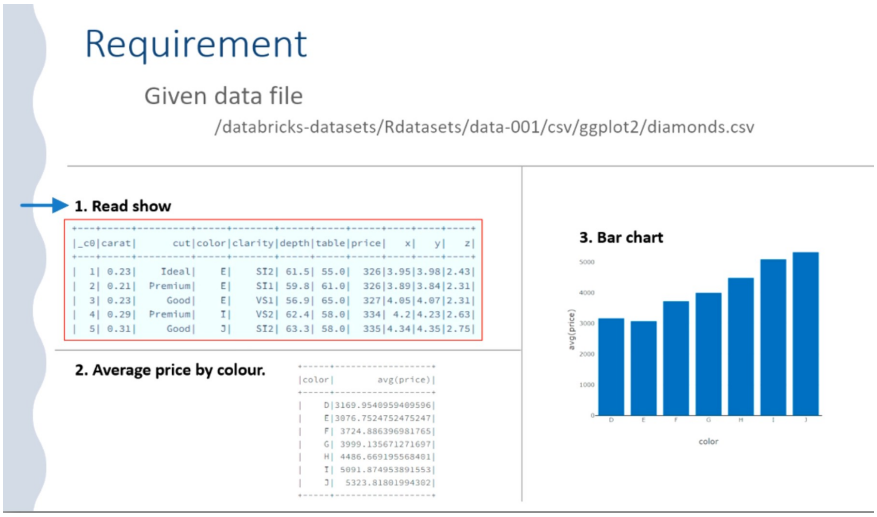
Requirements:

We have some data available in the cloud environment at the location given below. It is a CSV format data file, which contains some information about the diamonds.



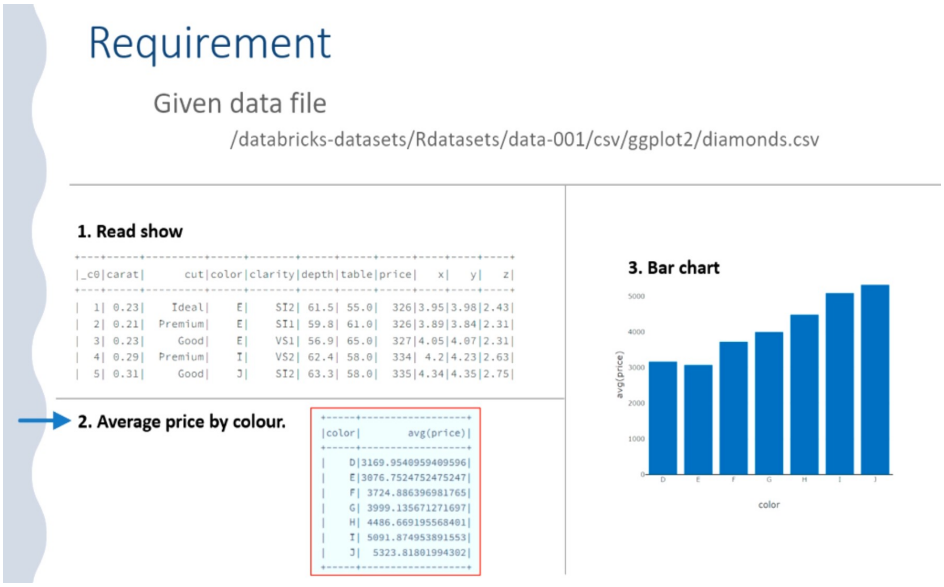
Requirements:

1. Read the data file and show some records in tabular format. And the output should look like the image highlighted below.



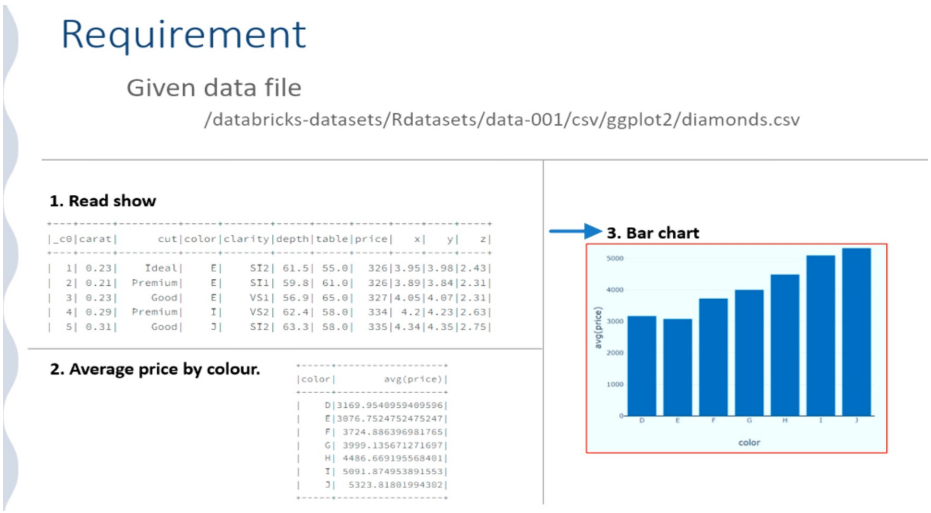
Requirements:

2. Calculate the Average price of the diamonds by color. And the result might look like the image highlighted below.

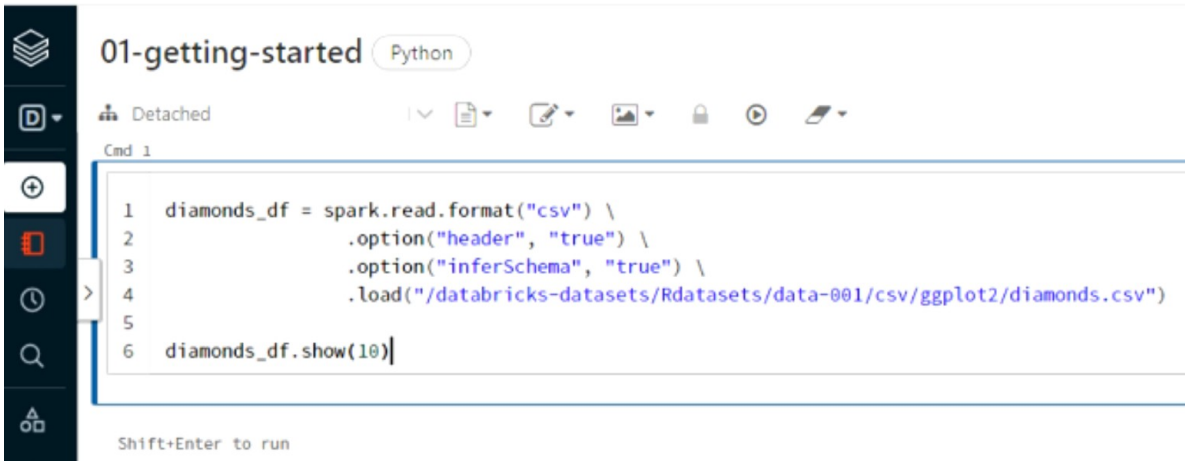


Requirements:

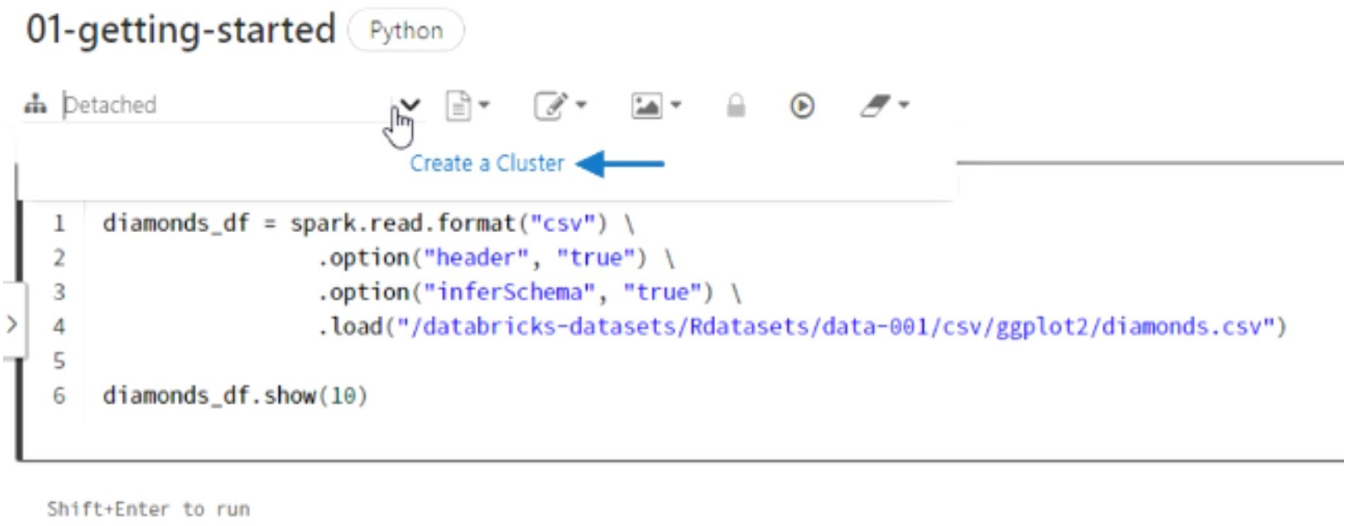
3. Draw a bar chart using this table to visualize the results. And the bar chart might look like the image highlighted below.



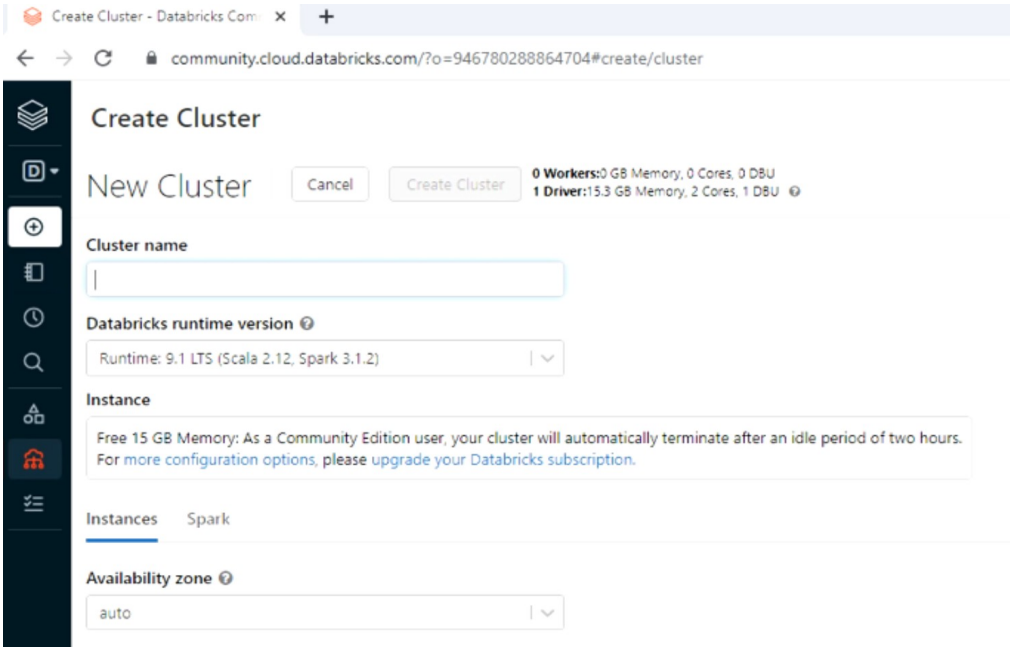
Here is the code shown below which meets your first requirement. (Reference: 01-getting-started.ipynb) It will read the data file and show some records in tabular format.



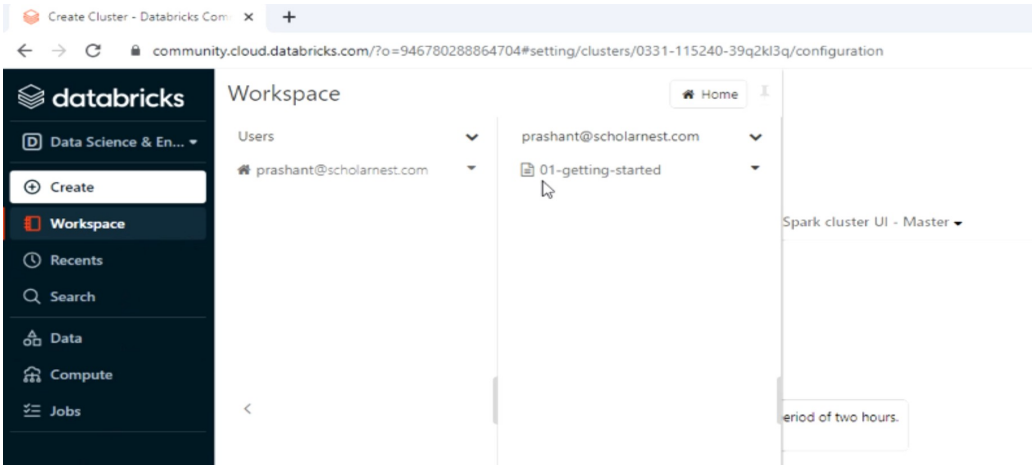
Now, we want to run this code and see the results. But running a Spark code requires a Spark cluster. As shown in the image below, click the dropdown menu, and you will see a link to create a cluster. Hit the link, and it will take you to the cluster creation page.



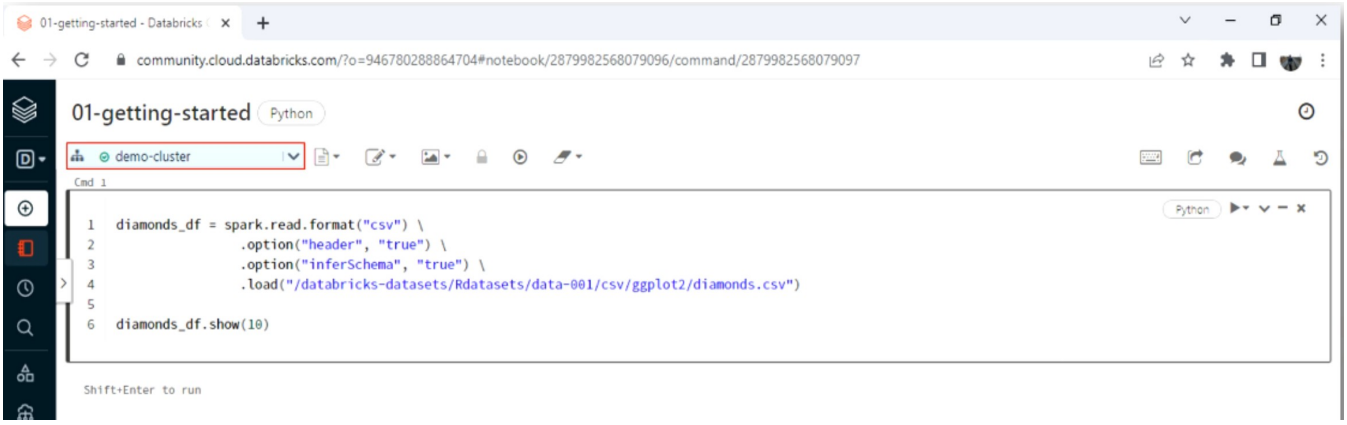
Give a name to the cluster and hit the create button, it might take a few minutes to wait for the cluster to start.



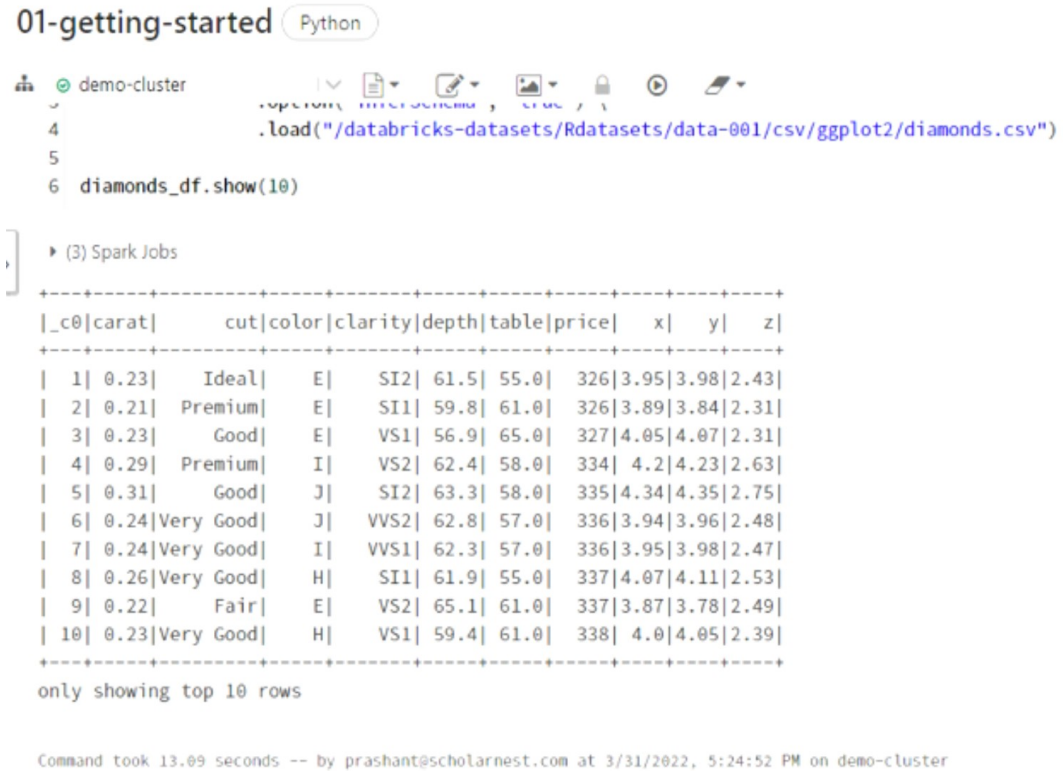
Now, go to the workspace menu option and go back to your notebook.



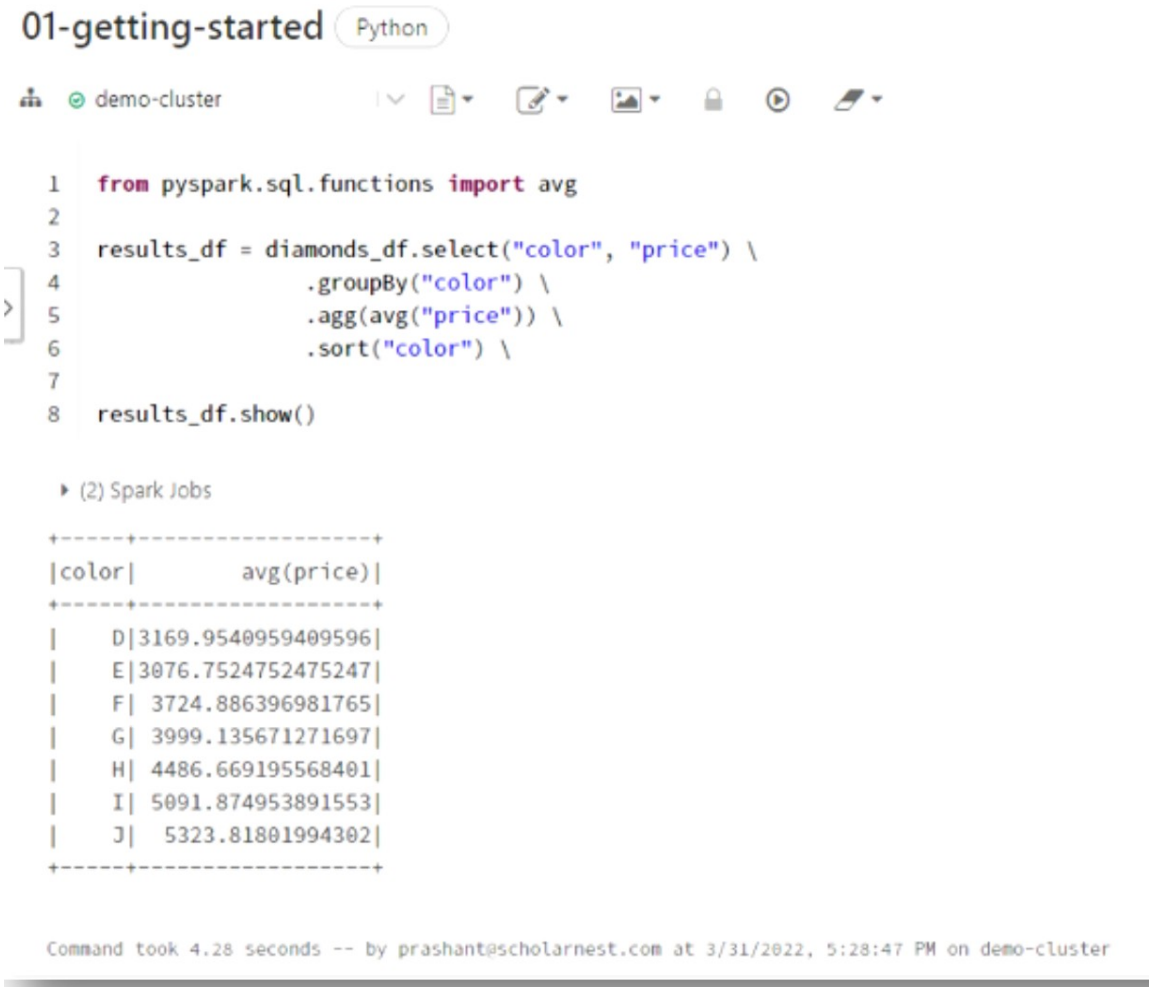
Make sure your cluster is attached to the notebook, and your code will run on this cluster. Now, click inside the code cell and press shift+enter to run the code.



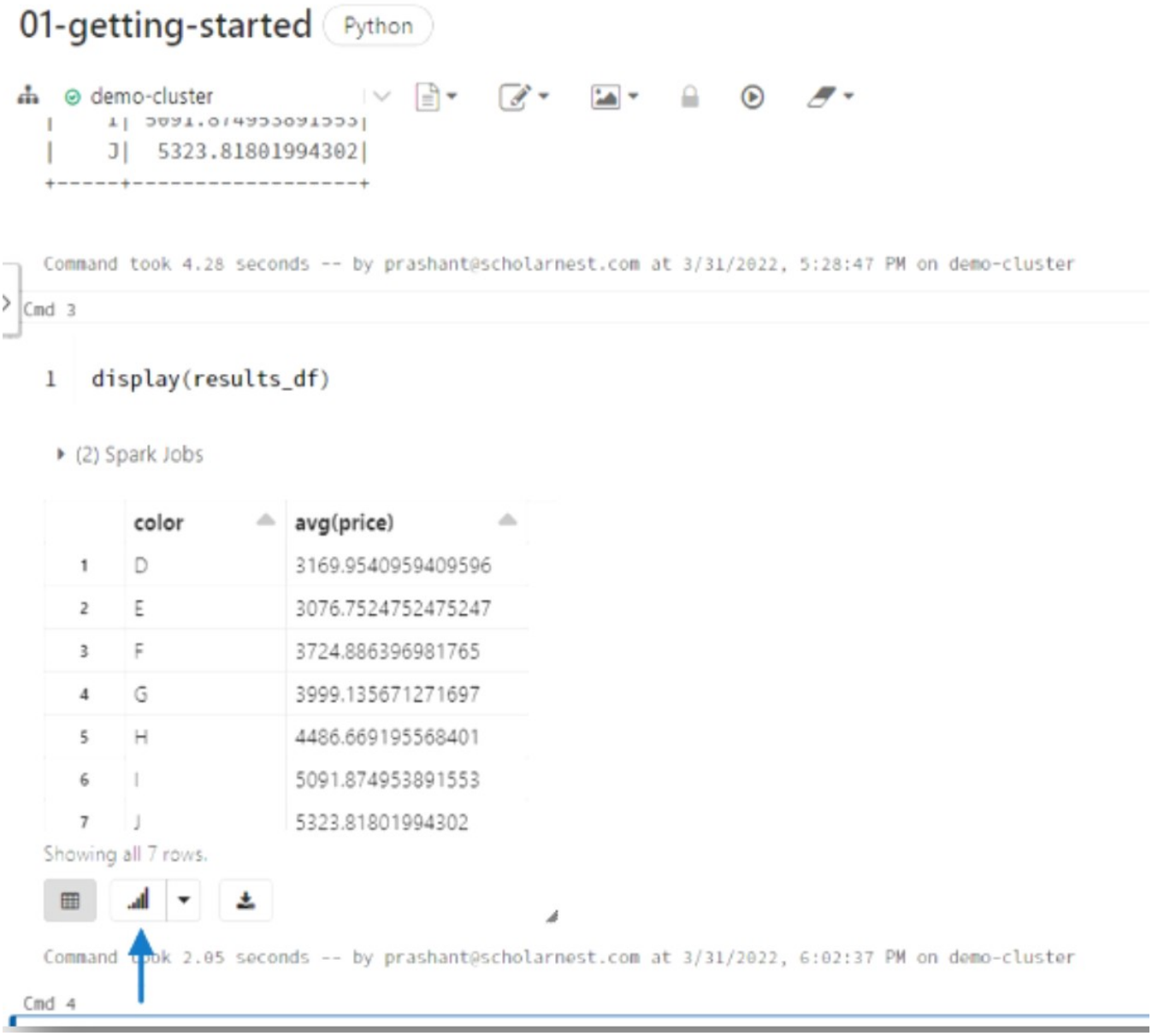
You can see the results here, and it matches our expected results. So we loaded the dimonds.csv file into Spark and showed ten records from the data file.



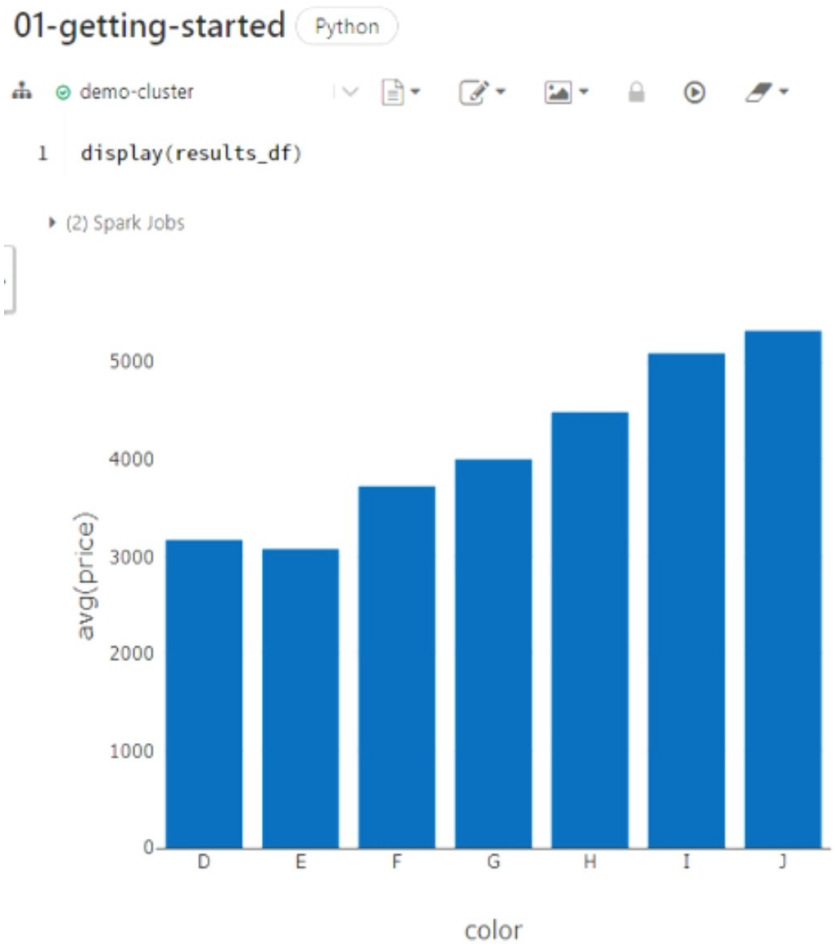
The following requirement is to process this diamond's data frame and calculate the average price by color. And here is the code for the same. So this code takes color and price columns, groups them by color, and calculates the average price. Finally, I show the results. You can also see the output shown below.



In the last requirement, we wanted to present the result obtained using a bar chart. Here is the code for the same. And you can see the output below. The display function allows you to format the results. The display function depicts the result in the tabular format but you can switch to the chart format by clicking chart button below the results.



And here is the bar chart we wanted.



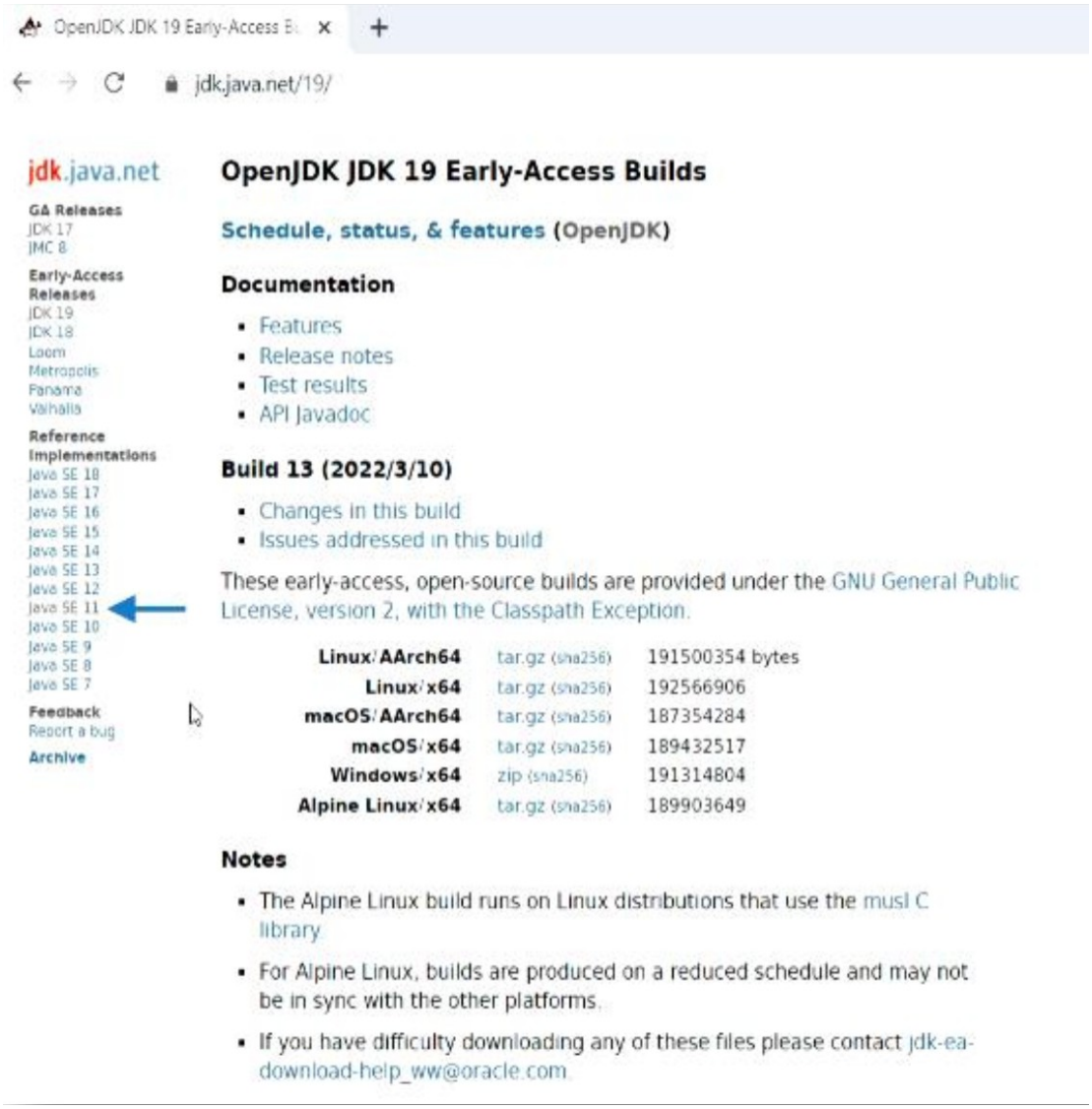
Setup your Local Development IDE

You need the following set of requirements for Spark setup on your machine. Let us go through this setup one by one.

Let's start with the JDK installation. Spark is a JVM application. So you need JDK on your local machine for installing and running your Spark application. The current most recent Java is at JDK 19. However, as of today, Spark supports JDK 8 or JDK 11. Do not use any other version of JDK because Spark is not well tested on other versions. You might face problems running Spark on other JDK versions.

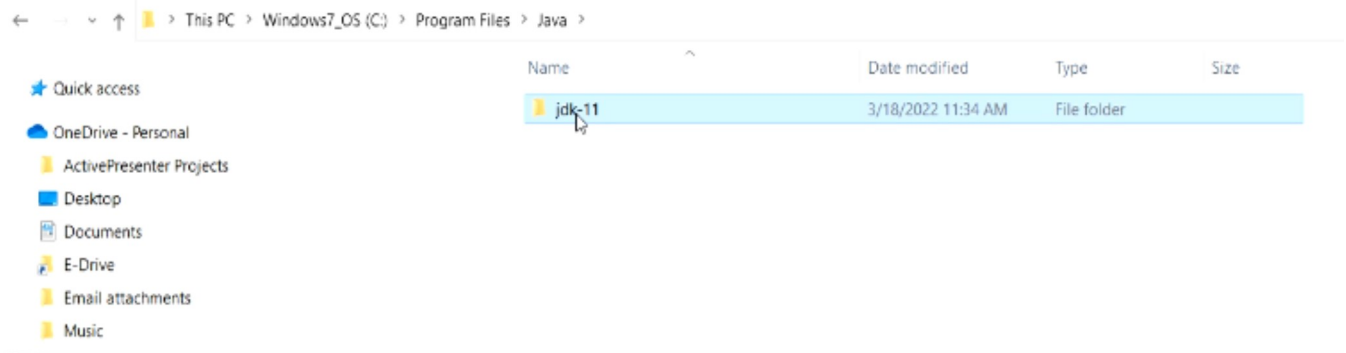
Start your browser and visit http://jdk.java.net/ You will a link for the most recent version of the JDK. Click the link to go to the downloads page.

Then choose Java SE 11 from the list of all available versions. And download the JDK 11 for your platform (Windows/Linux) in the next step.



Go to the downloaded file and extract it. You will see the JDK-11 folder. Inside the JDK-11, you will see some more folders. Copy the JDK- 11 folder and paste it at some permanent place.

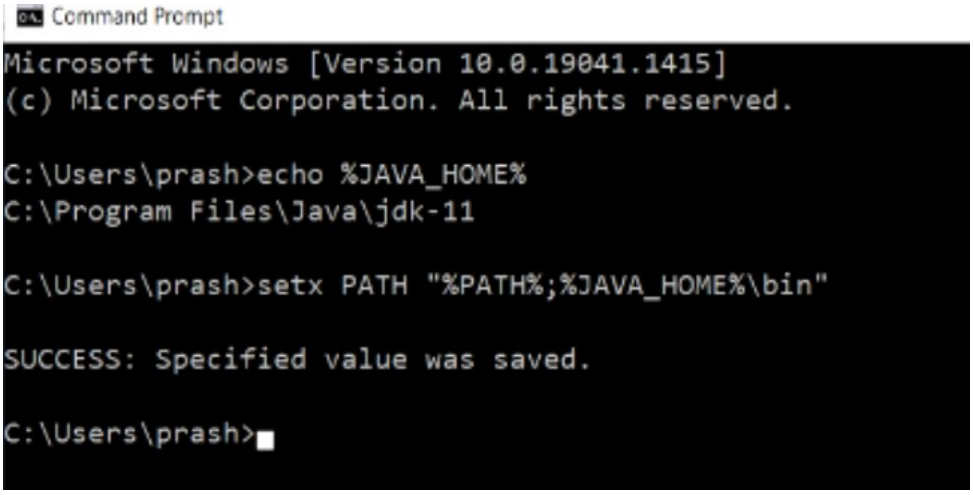
It is recommended to keep it in *C:\Program Files\Java* folder.



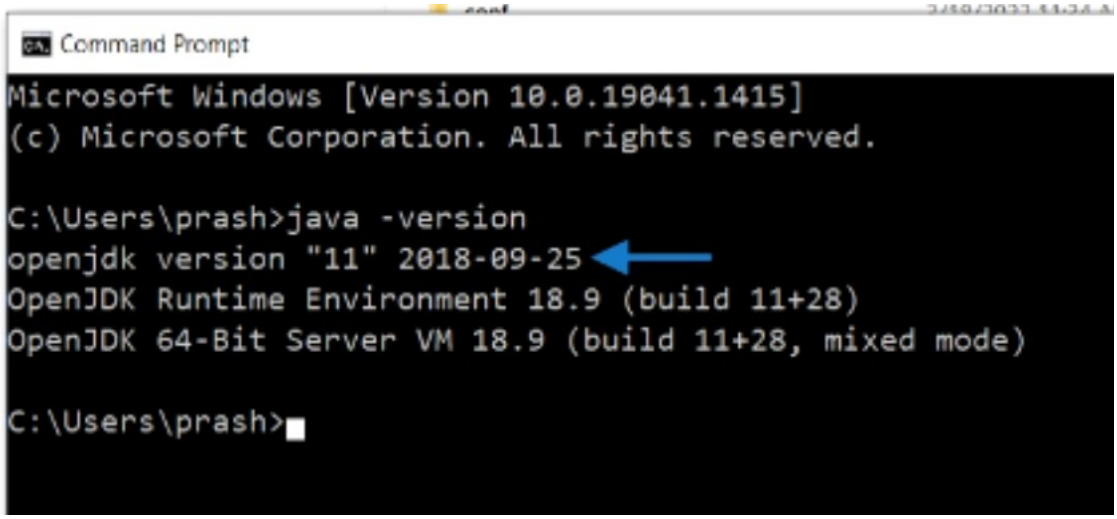
You have JDK 11 on your system. But it won't work as we need to set up two environment variables to make it work. Start windows command prompt. Now you can use the setx command as shown below to set the JAVA_HOME environment variable. The JAVA_HOME variable must point to your JDK-11 directory.

Make sure you see a success message.

If you want to check that your environment variable is set properly, you can restart the command prompt and check it using the echo command as shown below. Then the second requirement is to add the JAVA_HOME\bin to your PATH environment variable. You can use the setx command as shown in the image below. And make sure you get a success message. Ensure you include the current value of your PATH environment variable and add JAVA_MOME\bin to the same.



Finally, execute the Java -version command, and you should see the current Java version. Make sure you see Java version 11. If you already have a Java 8 or Java 11 pre-installed on your machine, you can skip the Java installation steps.

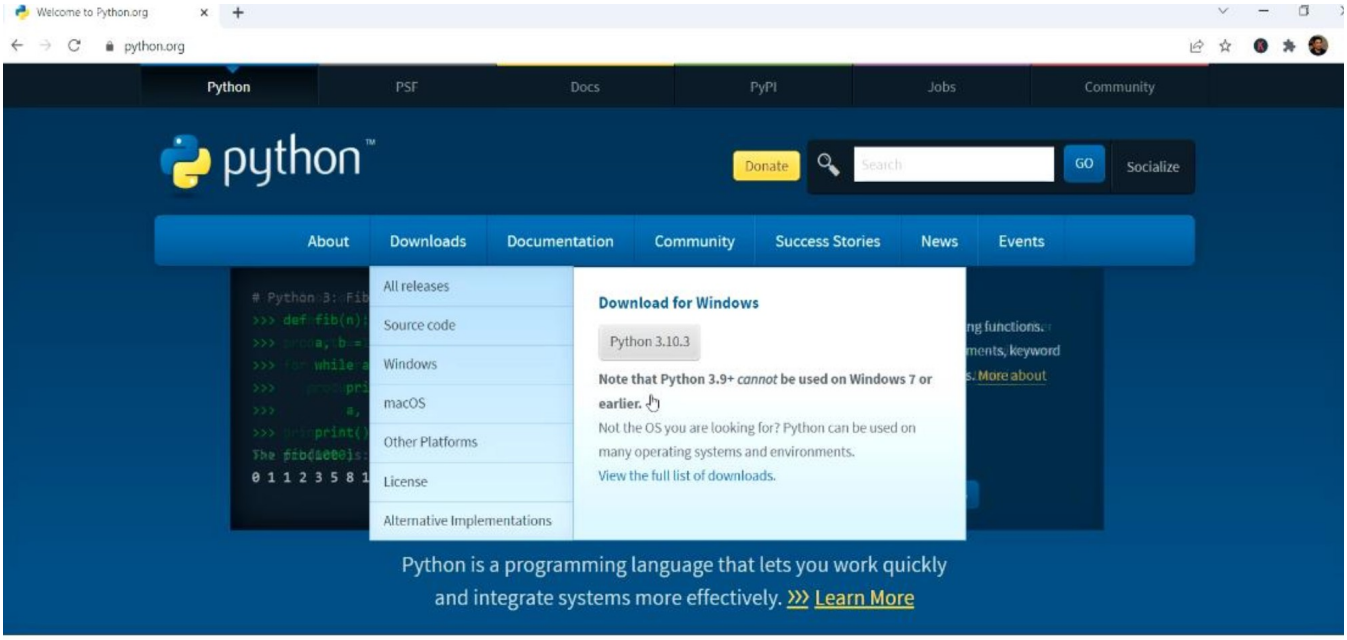


So, basically there are three steps you need to check in order to complete your JDK installation.

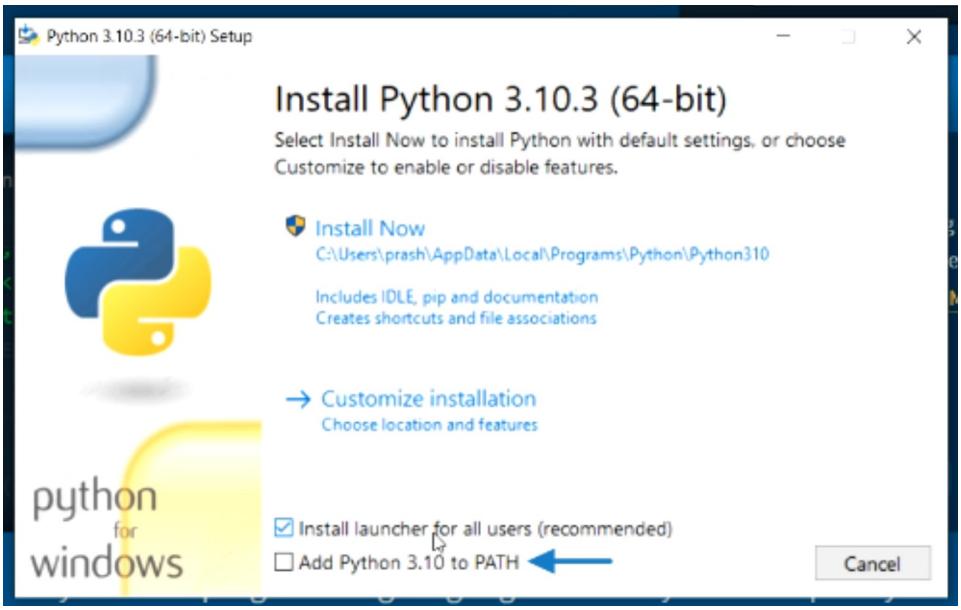
1. JAVA_HOME environment variable is set, and it is pointing to Java 8 or java 11
2. JAVA_HOME\bin is included in your PATH environment variable.
3. java -version command is showing Java 8 or Java 11

If all these three are there in place, you are done.

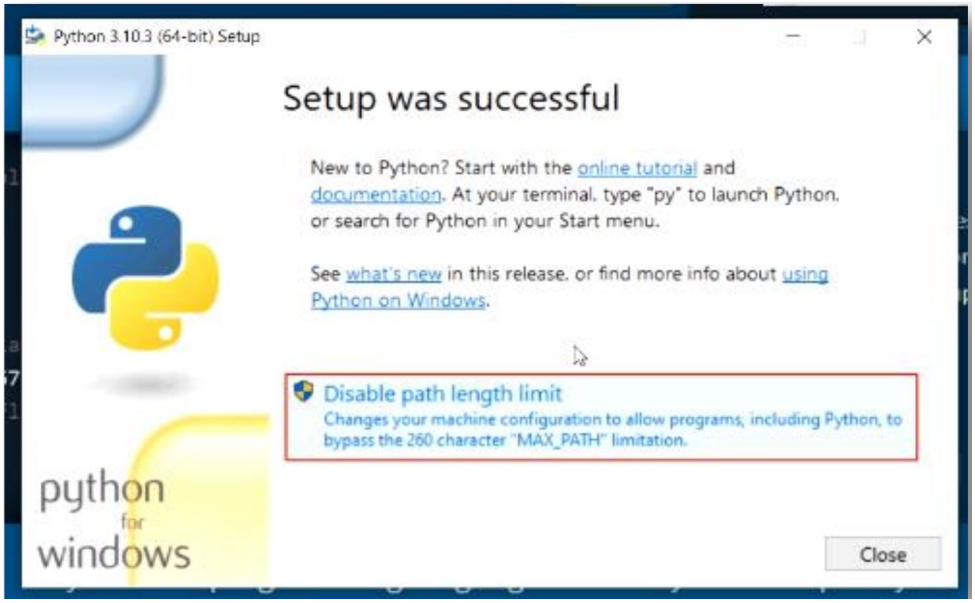
The next step to setup your Local Development IDE is to install Python. Visit https://www.python.org/ and download the latest Python version for your machine.



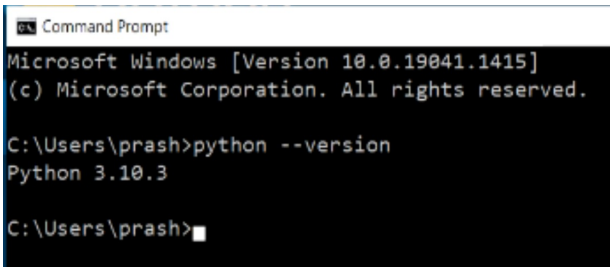
Once downloaded, click the file to run it. You should see an option to add Python to your PATH environment variable. Enable that option, so we do not have to do it manually. Then click the install now button.



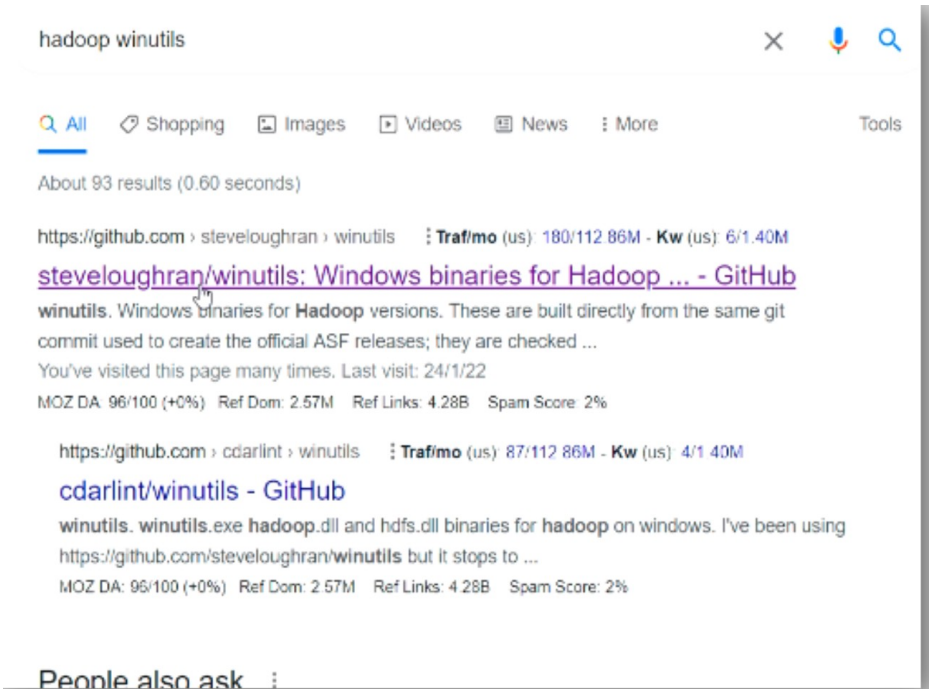
It might take a couple of minutes to complete the installation. You might see a notification to disable the path length limit. An older windows machine allows only 255 characters for any path. The max path length limit is a problem, and this limitation is removed from the newer systems. If you are using an older system, you might see this message. I recommend that you choose this option and remove the max path length.



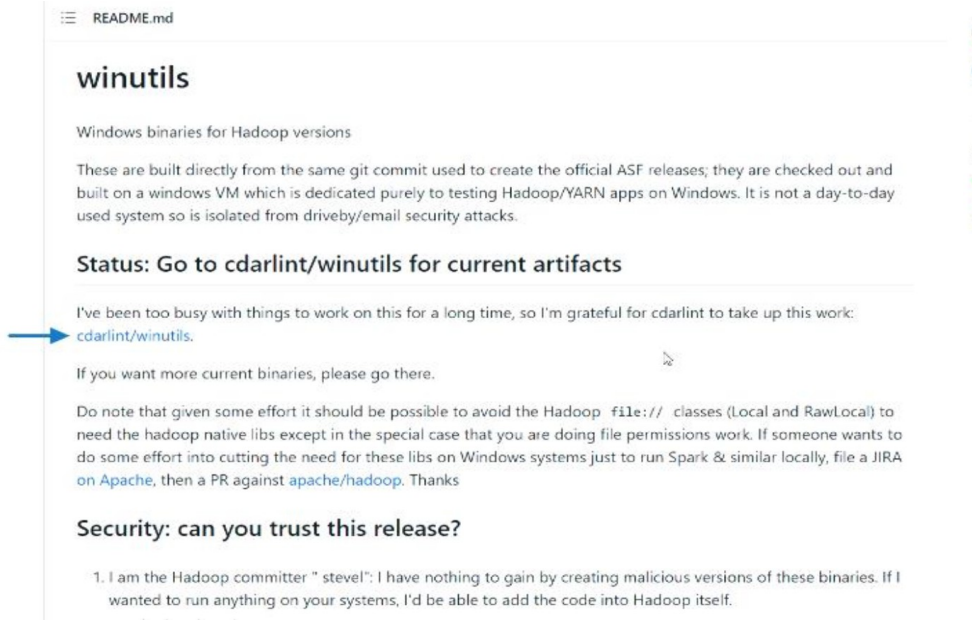
Once done with the installation, you should check it once. Start a command prompt and run *python --version* command and make sure you see the latest version that you recently installed.



The following requirement is to setup Hadoop Winutils. Google for Hadoop Winutils, and you should see a GitHub page link. This is an old repository and is not being actively managed.



After following the GitHub link, scroll down, and you will see a link for the new repository as shown in the image below. Follow the link to go to the current repository.


README.md

winutils

Windows binaries for Hadoop versions

These are built directly from the same git commit used to create the official ASF releases; they are checked out and built on a windows VM which is dedicated purely to testing Hadoop/YARN apps on Windows. It is not a day-to-day used system so is isolated from driveby/email security attacks.

Status: Go to cdarlint/winutils for current artifacts

I've been too busy with things to work on this for a long time, so I'm grateful for cdarlint to take up this work: cdarlint/winutils.

If you want more current binaries, please go there.

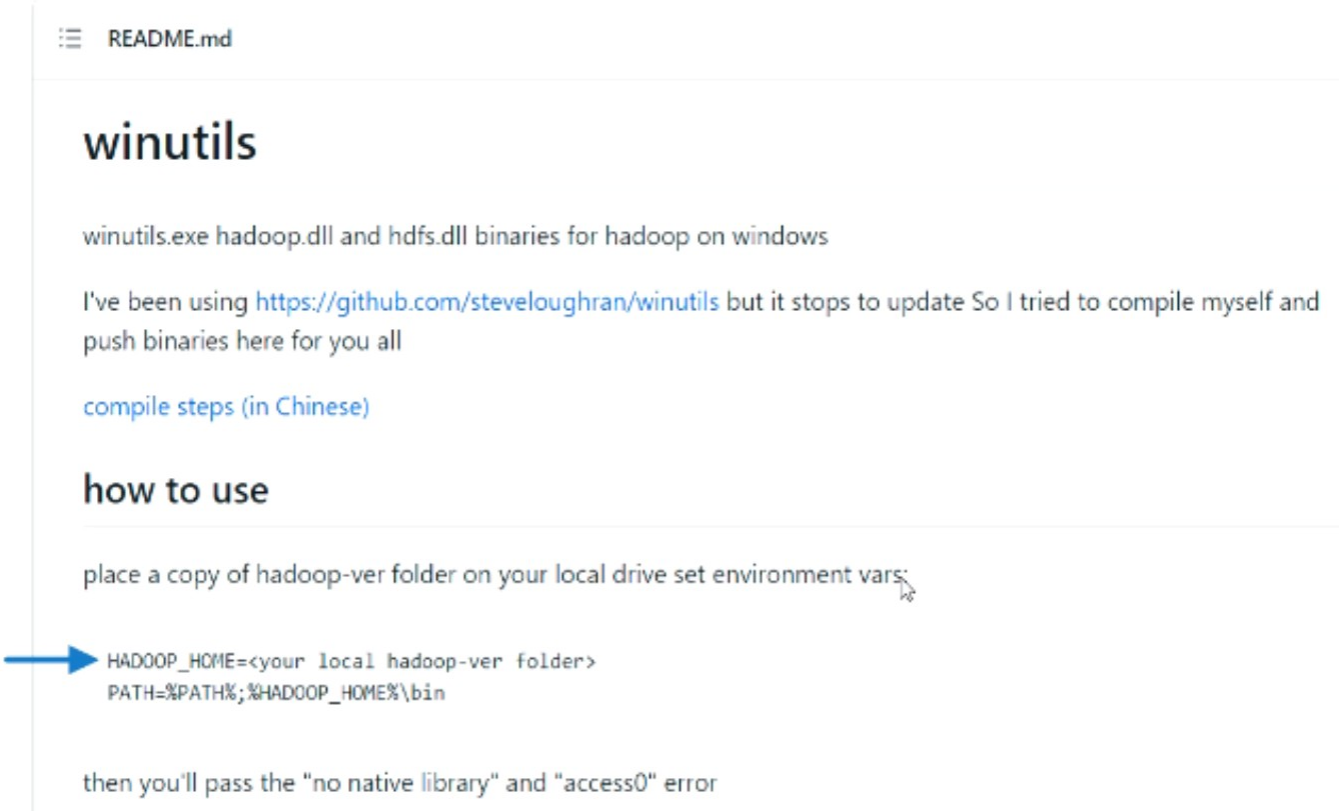
Do note that given some effort it should be possible to avoid the Hadoop `file://` classes (Local and RawLocal) to need the hadoop native libs except in the special case that you are doing file permissions work. If someone wants to do some effort into cutting the need for these libs on Windows systems just to run Spark & similar locally, file a JIRA on Apache, then a PR against apache/hadoop. Thanks

Security: can you trust this release?

1. I am the Hadoop committer " stevel": I have nothing to gain by creating malicious versions of these binaries. If I wanted to run anything on your systems, I'd be able to add the code into Hadoop itself.

Download the current repository, and if you scroll down you will see 2 steps of using this:

1. Setup your HADOOP_HOME environment variable
2. Include HADOOP_HOME\bin to your PATH environment variable.


README.md

winutils

winutils.exe hadoop.dll and hdfs.dll binaries for hadoop on windows

I've been using https://github.com/steveloughran/winutils but it stops to update So I tried to compile myself and push binaries here for you all

compile steps (in Chinese)

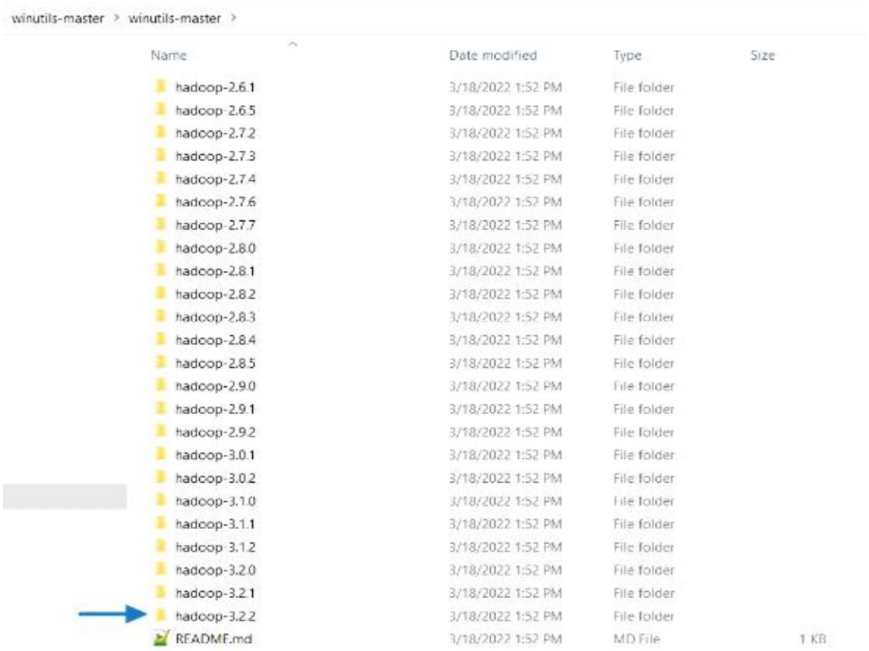
how to use

place a copy of hadoop-ver folder on your local drive set environment vars:

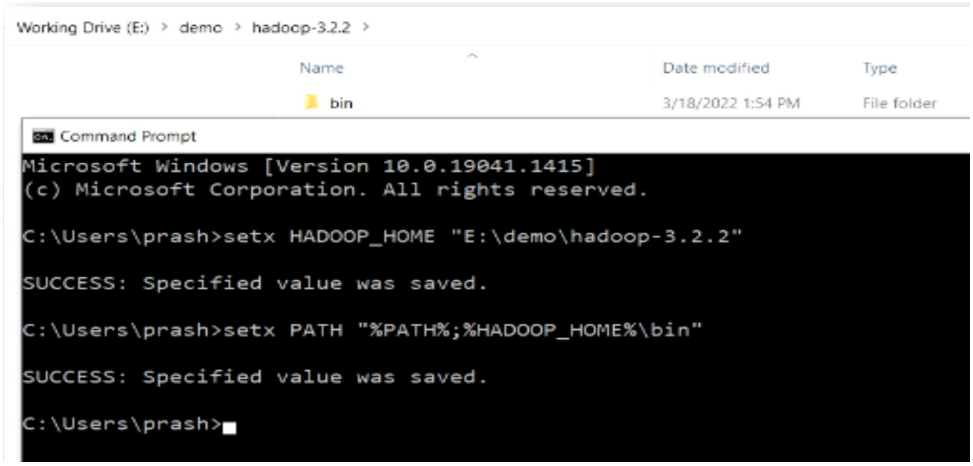
```
HADOOP_HOME=<your local hadoop-ver folder>
PATH=%PATH%;%HADOOP_HOME%\bin
```

then you'll pass the "no native library" and "access0" error

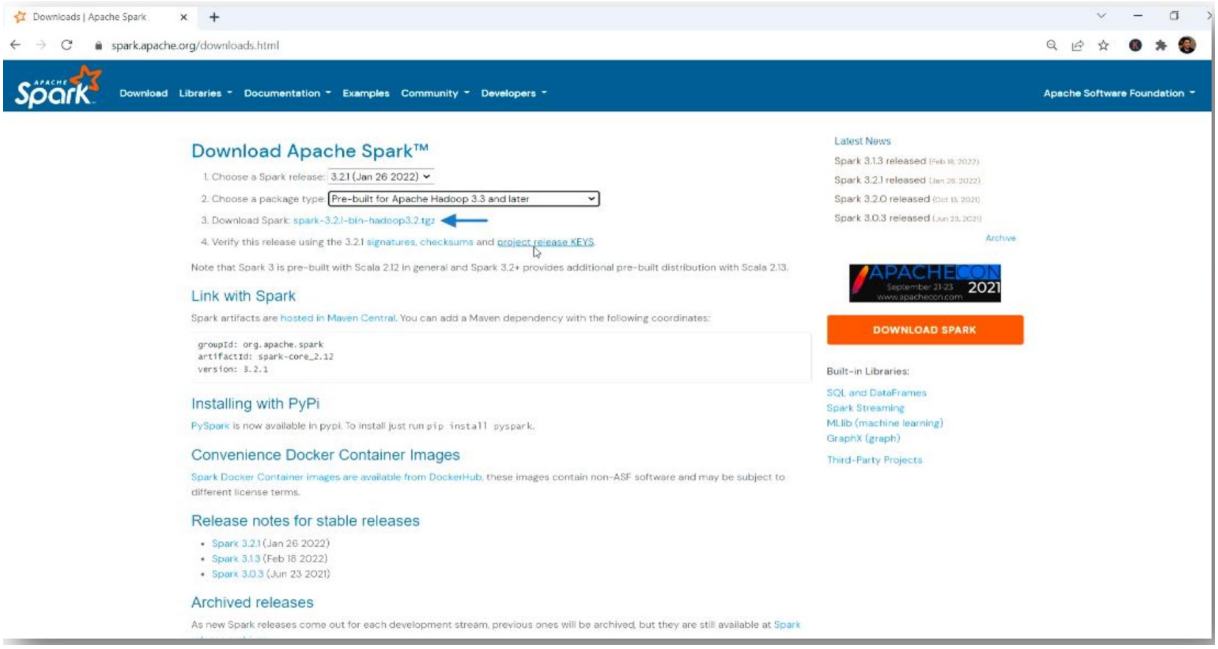
Once downloaded, go to the zip file and un-compress it. You will see the winutils-master directory. Go inside, and you will see many other folders. So I recommend copying the latest version folder and pasting it at another permanent location. If you go inside, and you will see a bin directory.



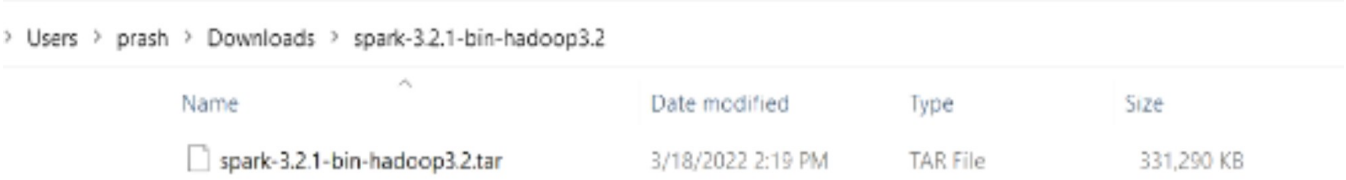
The next step is to setup your environment variable. Go the command prompt and use the setx command to set the environment variable, as shown in the image below. Make sure you see the success message. Now add it to the PATH environment variable, as shown in the image below. Make sure you see the success message again.



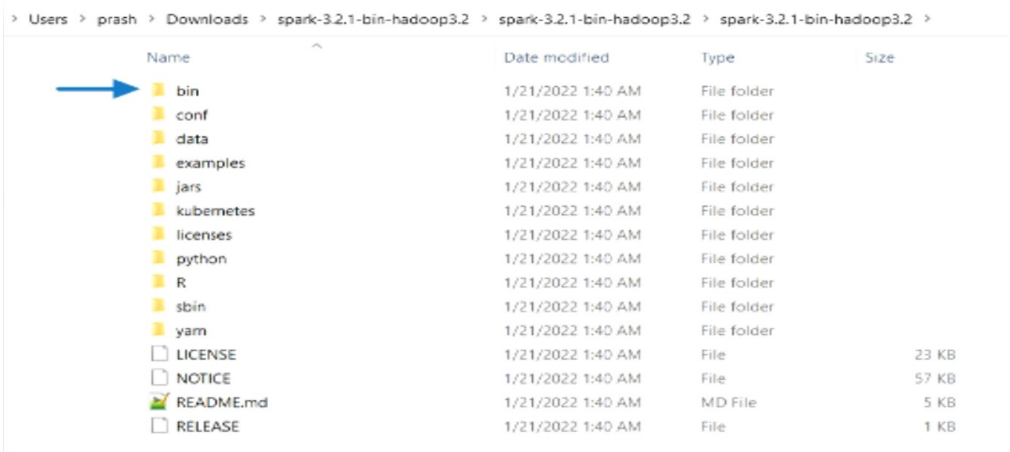
The next requirement is Spark Binaries. Visit https://spark.apache.org/ and click the download link. Then choose the latest Spark version and the Hadoop version. Now you can click the download link and start the download.



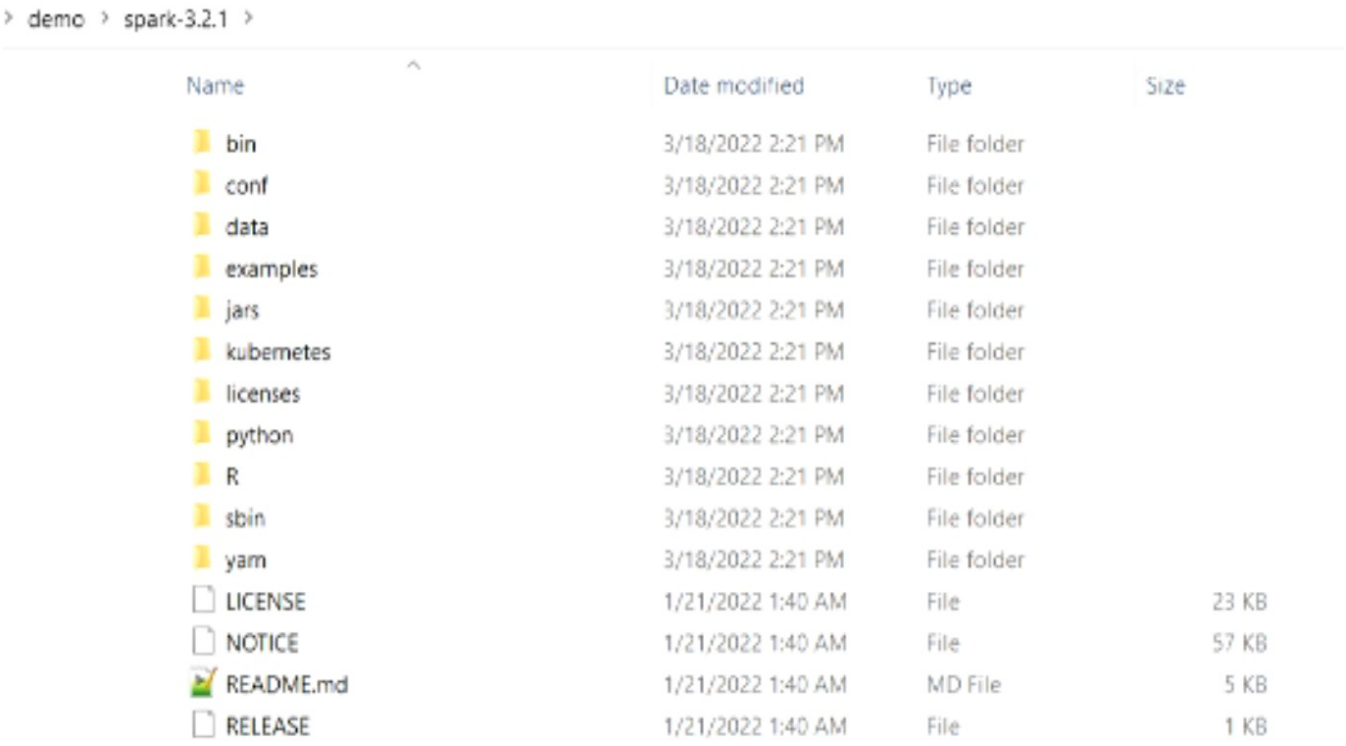
Once downloaded, go to the file and uncompress it. The Spark binary is not a zip file. It is a tgz file. So you might need 7zip for uncompressing this tgz on the windows platform. After compressing the file, go inside, and you will see a tar file. This one is also a compressed file. You can untar this file using the 7zip tool.



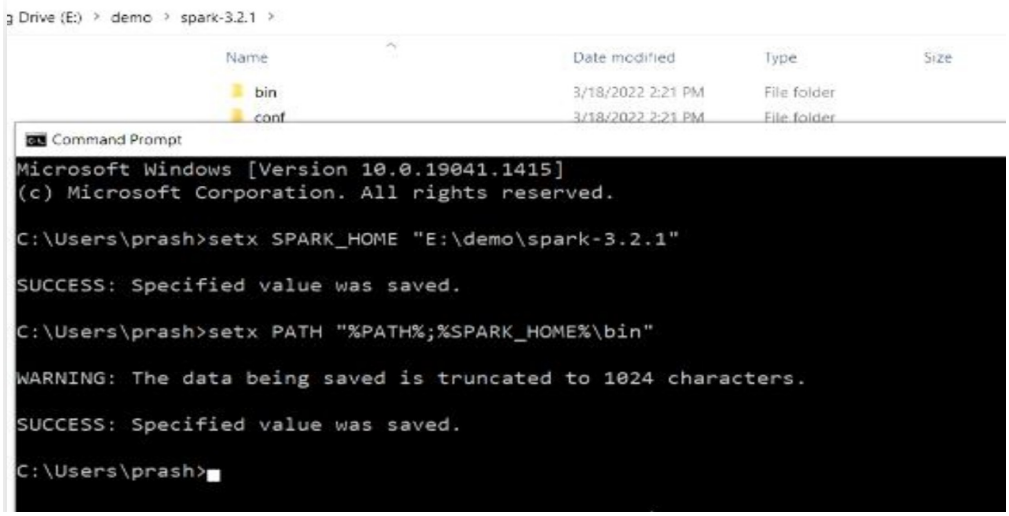
After compressing the tar file, you will see another uncompressed directory. If you go inside the folder a couple of times you will the see the bin directory shown in the image below.



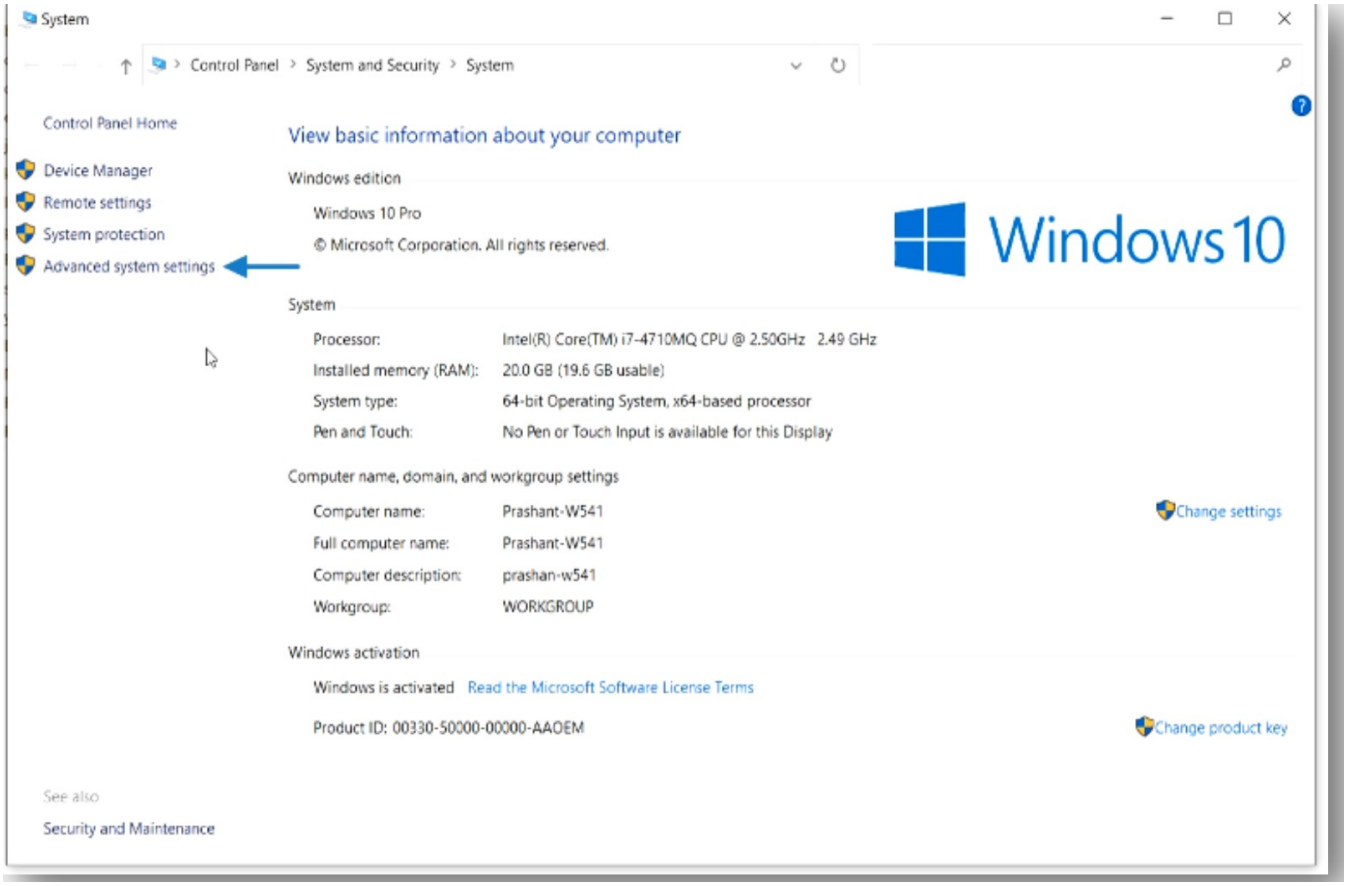
There are too much folders inside folders, so you can copy the parent folder and paste it to a permanent location. So the image shown below is your spark home. But we must set the environment variables.



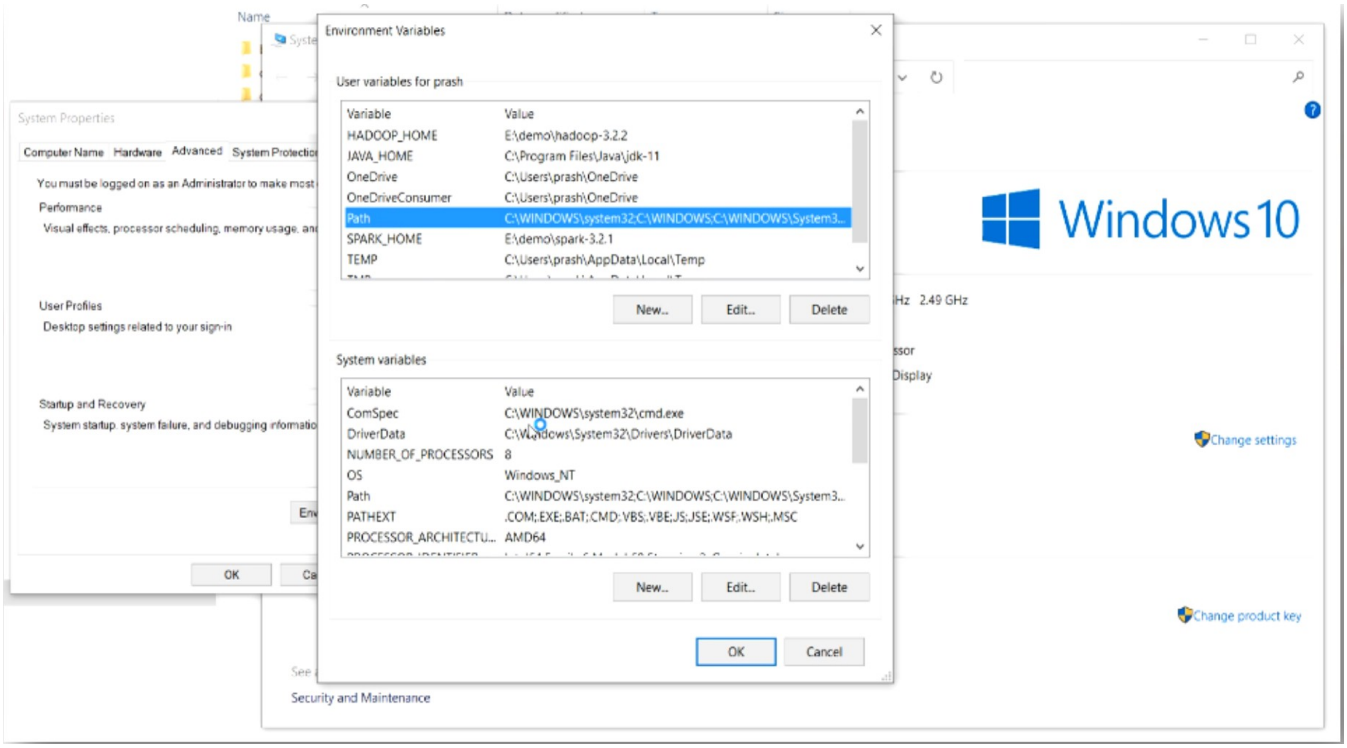
Start your command prompt and use the setx command to set the SAPRK_HOME environment variable. Make sure you see the success message. Then you should also add the SPARK_HOME\bin to our PATH environment variable. You might also see a warning that data being saved is truncated. If you see it, your PATH environment variable is not set correctly.



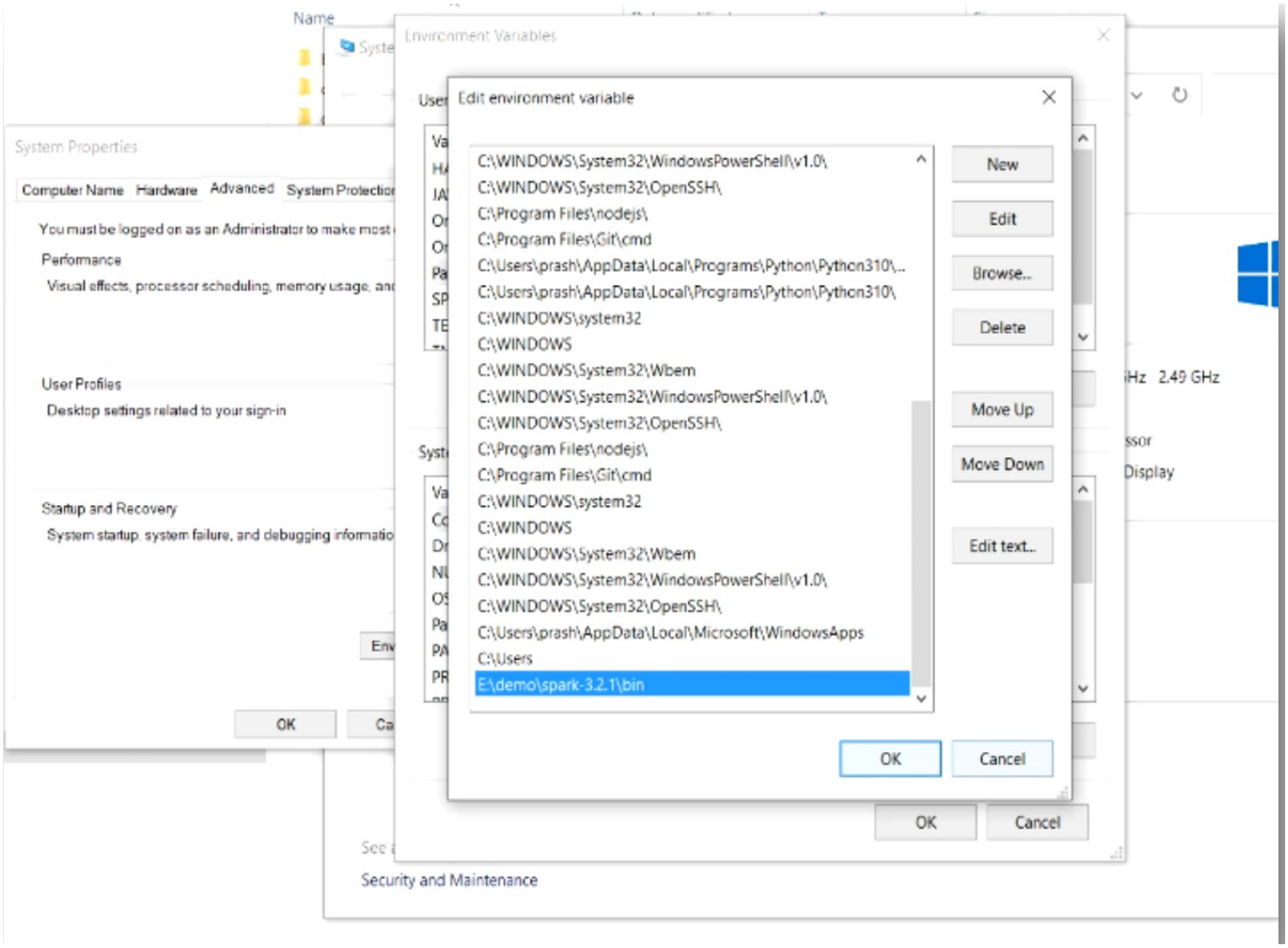
You can add a more extended PATH environment variable using the Windows UI. Right-click on This PC and choose Properties. You will see a window as shown in the image below. Then, go to advanced system settings and then follow the Environment Variable button.



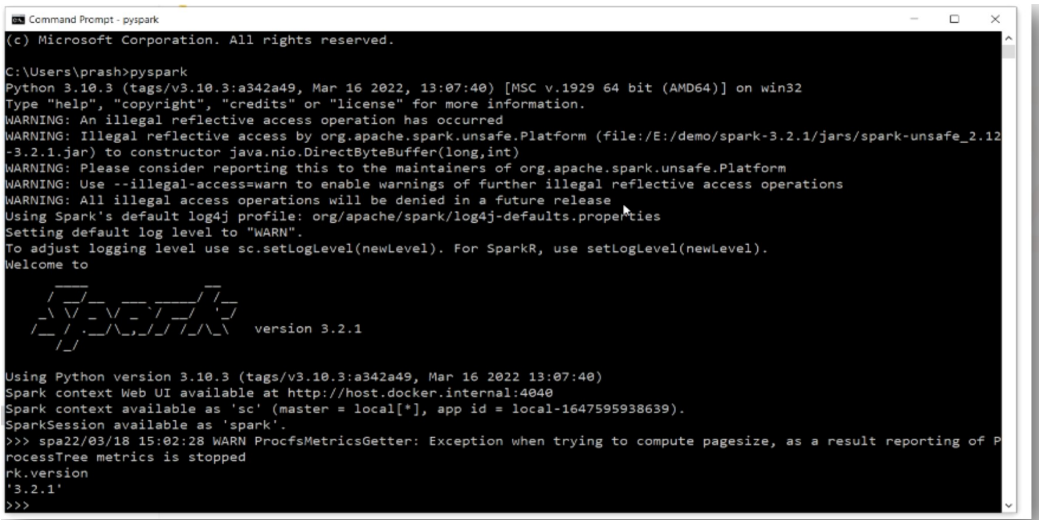
You will land on the window shown below. You will see the Path environment variable. Edit it. You can see a long list of things included in the path ENVIRONMENT Variable.



Click the new button, and paste the Spark Home directory location including the “\bin” at the end. That's all. Click Ok and close everything.



To test if Spark setup is successful, go to command prompt and type “pyspark” You should see some messages and finally a command prompt. If you see a command prompt and no error messages, you are good to go. You have Apache Spark running on your local machine. You might see a warning stating Exception when trying to compute the page side.

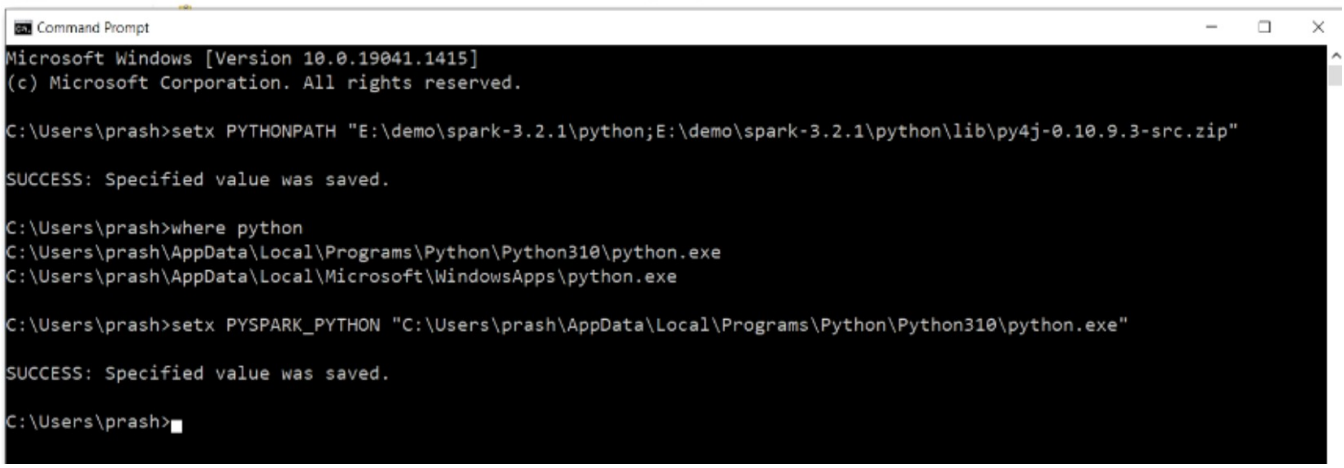


The next step is to set up the following two environment variables that can save you from seeing unnecessary errors while working with Spark on your local machine.

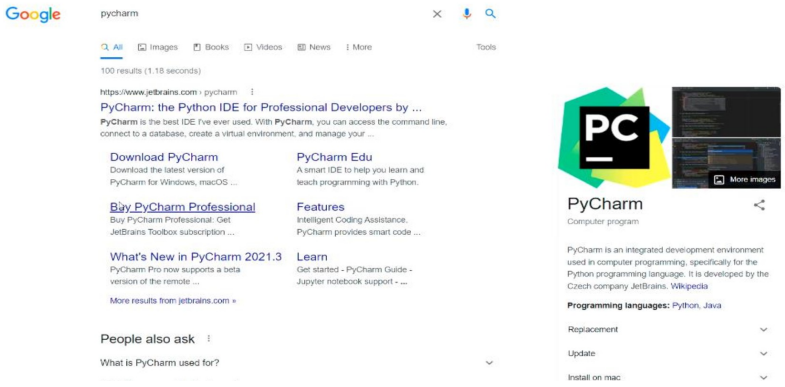
1. PYTHONPATH – If you go back to your Spark home directory, you will see a Python directory there. Go inside that python directory and copy that path. Then, Paste it into

the PYTHONPATH variable value. Make sure to take the absolute path to the py4j zip file and paste it to PYTHONPATH.

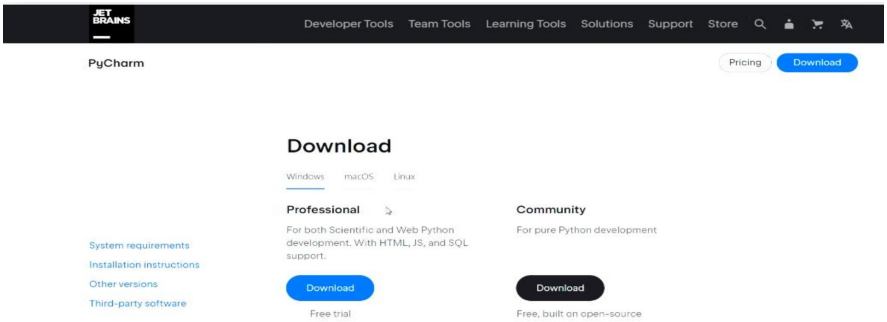
2. PYSPARK_PYTHON - You can use the where-command to find your Python installation location. And use the location to set PYSPARK_PYTHON variable.



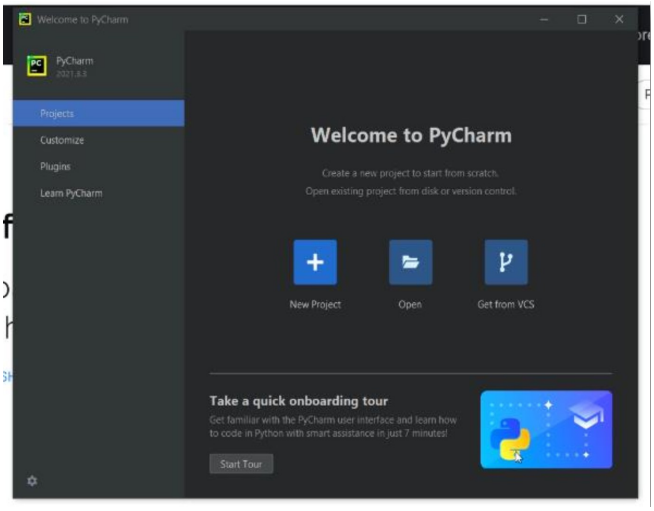
But writing code on command prompt if not a very good approach. We do not want to develop large Spark applications working on the command prompt. I need an IDE for being productive and managing large project development. PyCharm is the most popular IDE for Spark development. Start your browser and search for PyCharm. And click the link shown jetbrains.com.



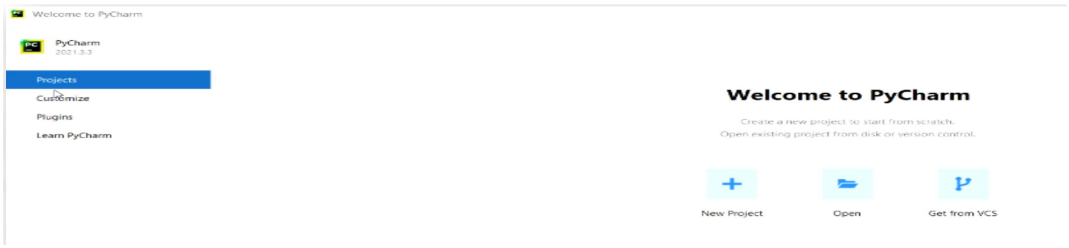
Click on the downloads button, and select the community edition. The community edition is a free opensource version of PyCharm. And the community edition is more than enough for our purpose.



Execute the downloaded installer and follow the on-screen default instructions. Once installed, you can start your PyCharm IDE. And you will see the welcome screen as shown below. You can go to customize option to set a lighter theme.

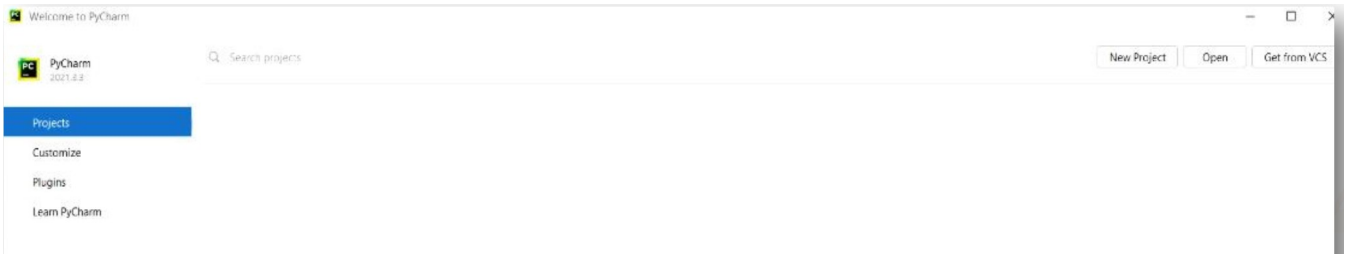


You can come to the Project menu, and you are ready to create your first Spark project.



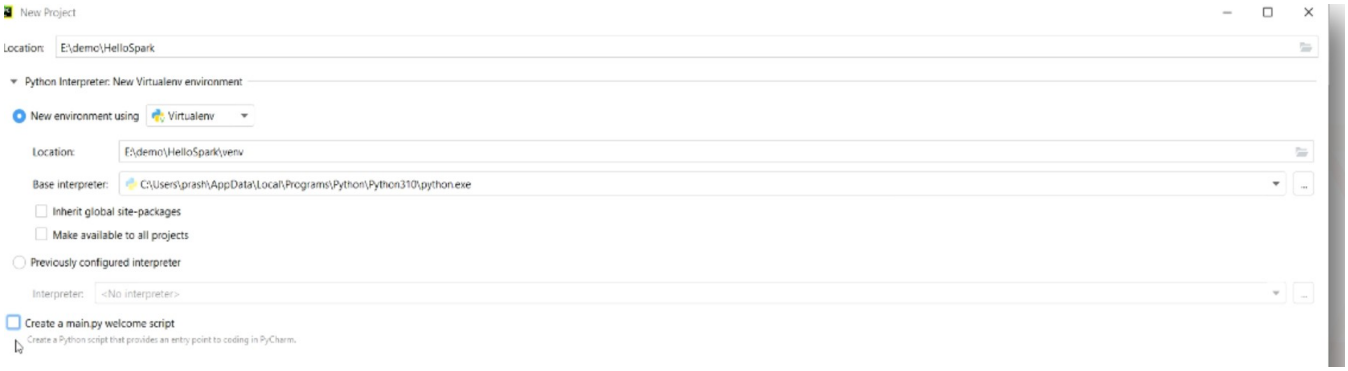
First Spark Application using IDE

Start your PyCharm IDE and create a new project.

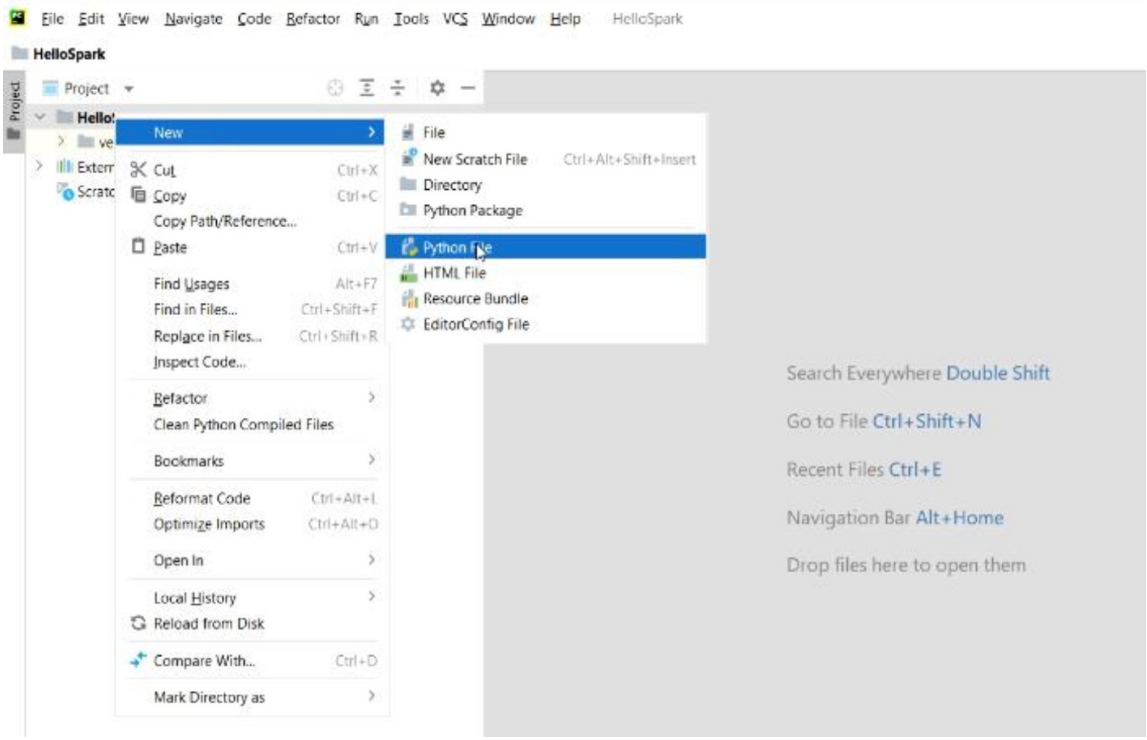


Set your new project location by hitting the browse button and choosing your working directory. Also type a new project directory location.

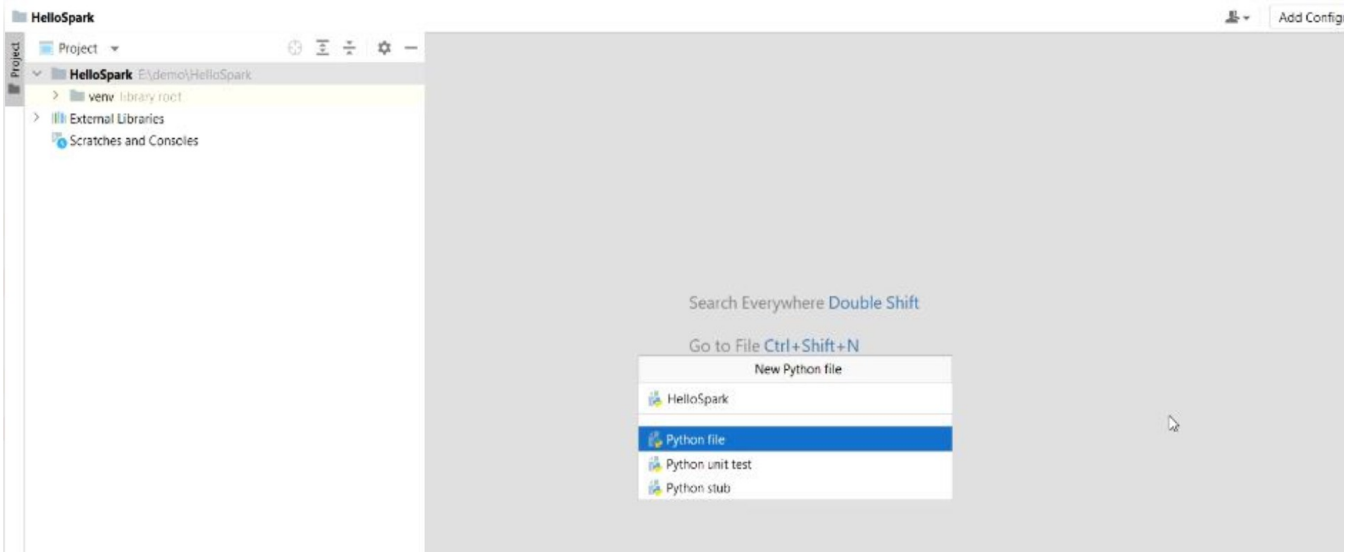
1. Ensure you have the latest version of Python selected in the Python Interpreter dropdown.
2. Uncheck all other options. We do not need anything else.
3. Use VirtualEnv for setting up our project.
4. Hit the create button.



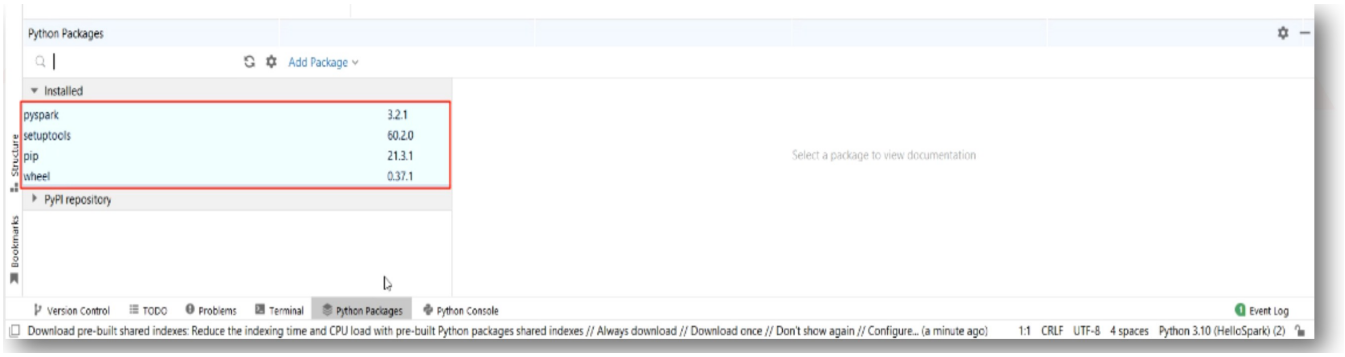
Once your blank project setup is done. You can create a new Python file in your project directory as shown in the image below



Give a name to your Python file.

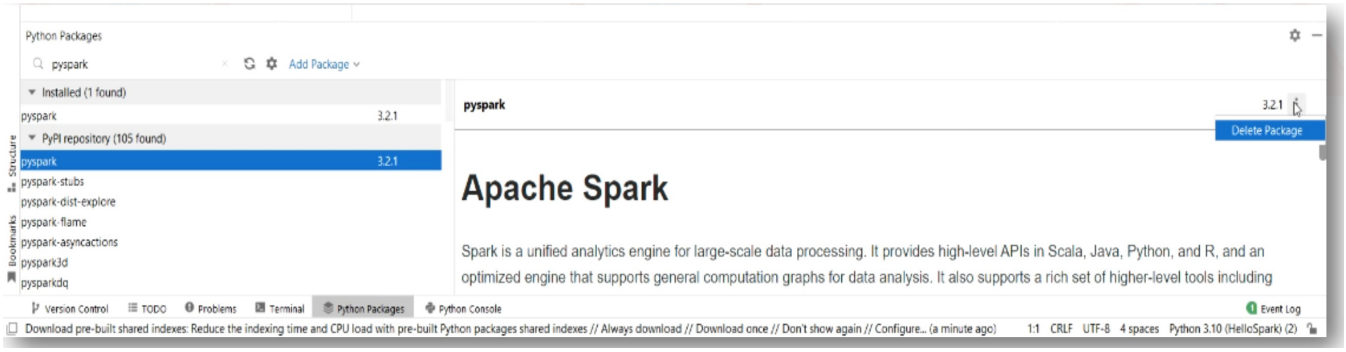


Make sure you have the PySpark package installed in your project's VirtualEnv. You check that by clicking the python packages window at the bottom of your IDE. And you should see your installed packages there as shown in the image below. I already have the pyspark package installed.

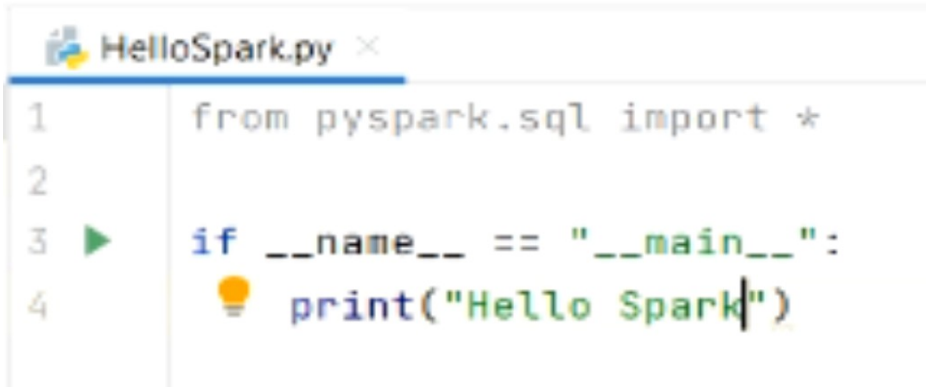


However, if you do not see PySpark in the list of your installed packages, search for it. Select the pyspark package under the PyPI repository. Come to the right side, three dots, and you should see an option to install. I see a delete option here because I have installed it already.

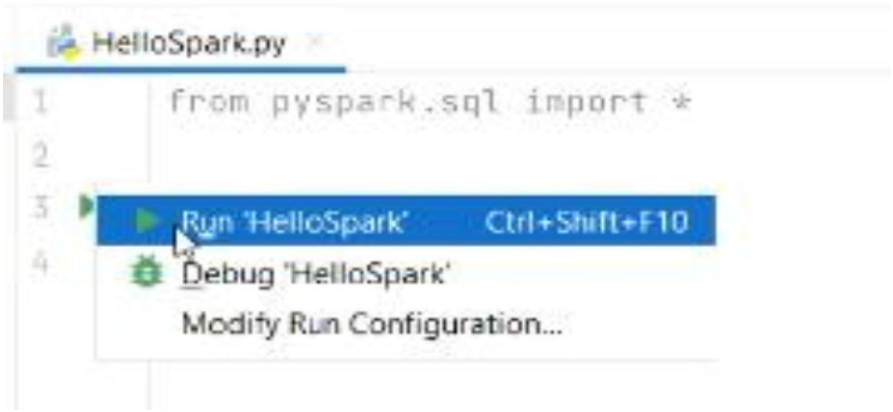
But you may see an install button to install the selected package. Install it if you do not have it already installed.



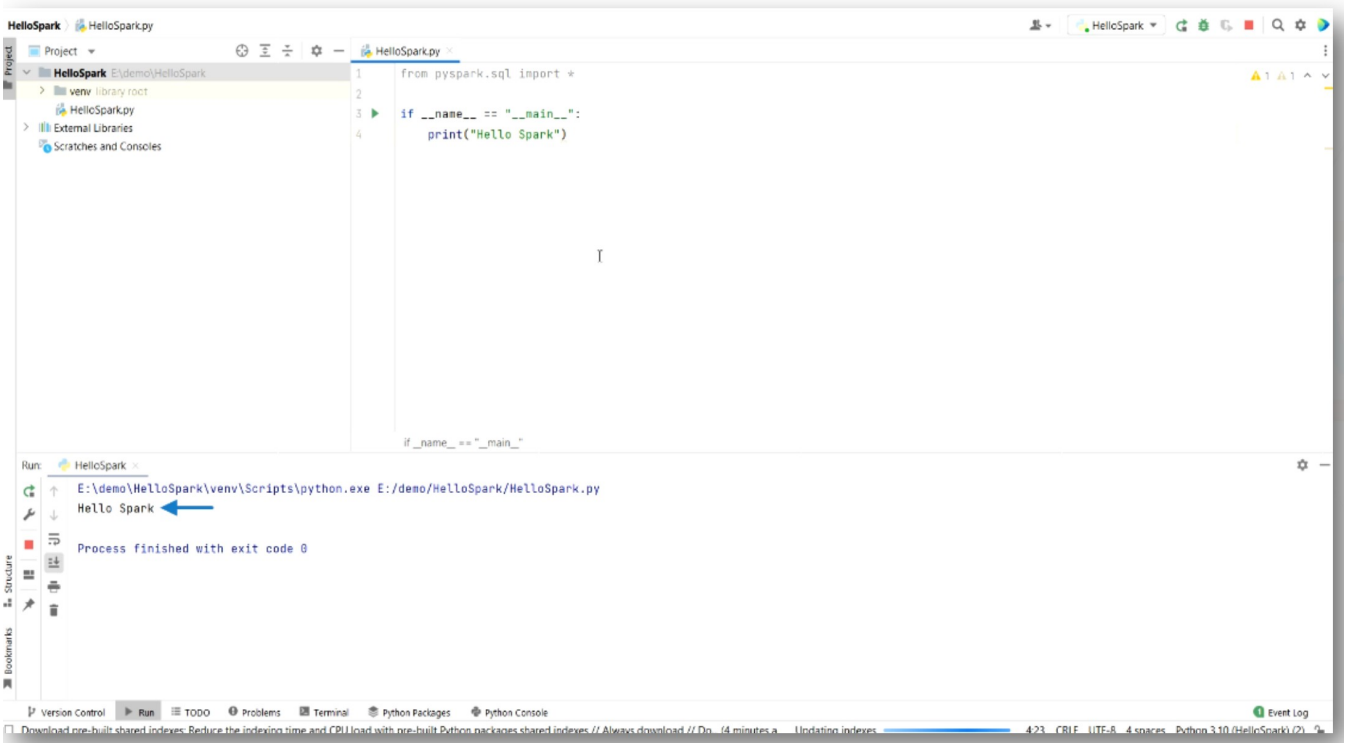
We haven't started learning Spark programming yet. So I am not expecting you to understand the code. However, I want to create a super simple Spark application and run it from the IDE. In the screenshot shown below, we have a simple Hello Spark Python code, it is not a Spark program.



Run your program as shown below.



It worked! I can see the output in the console below.

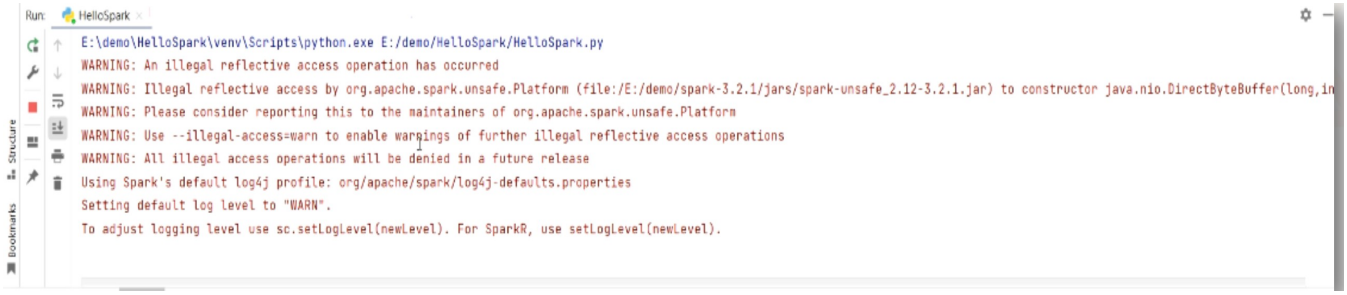


Now, let us look at a Spark code. Here is my super simple Spark application. I am creating a Spark Dataframe using a data list. I am giving some column names to list items and executing the show() method on my Dataframe. Do not stress yourself to understand the code.

HelloSpark.py

```
from pyspark.sql import *

if __name__ == "__main__":

    spark = SparkSession.builder \
            .appName("Hello Spark") \
            .master("local[2]") \
            .getOrCreate()

    data_list = [("Ravi", 28),
                 ("David", 45),
                 ("Abdul", 37)]

    df = spark.createDataFrame(data_list).toDF("Name", "Age")
    df.show()
```

You will see some warnings when you run the code. But you can ignore those warnings.



Finally, you will see Dataframe output. We managed to run a super simple Spark application on our local computer.

Micro Project – Problem Statement

We have seen a Spark example earlier which gives us the following feel:

1. What do we do in the Spark application?
2. How does a typical Spark application code looks?
3. How to write and execute your Spark application?

Apache Spark is a data processing engine. In a typical Spark application, we read data from a source, process it, and write or present the results. And this course is all about learning how to do these three things - Read, Process and Write. We have seen a sample code of the Spark application, where we learned how to write and execute Spark code.

It's time to start learning Spark Programming and develop your Spark coding skills. We will take a project-driven approach to learning things. We have a micro project problem statement shown below. I am giving you the Fire Calls-For-Service dataset.

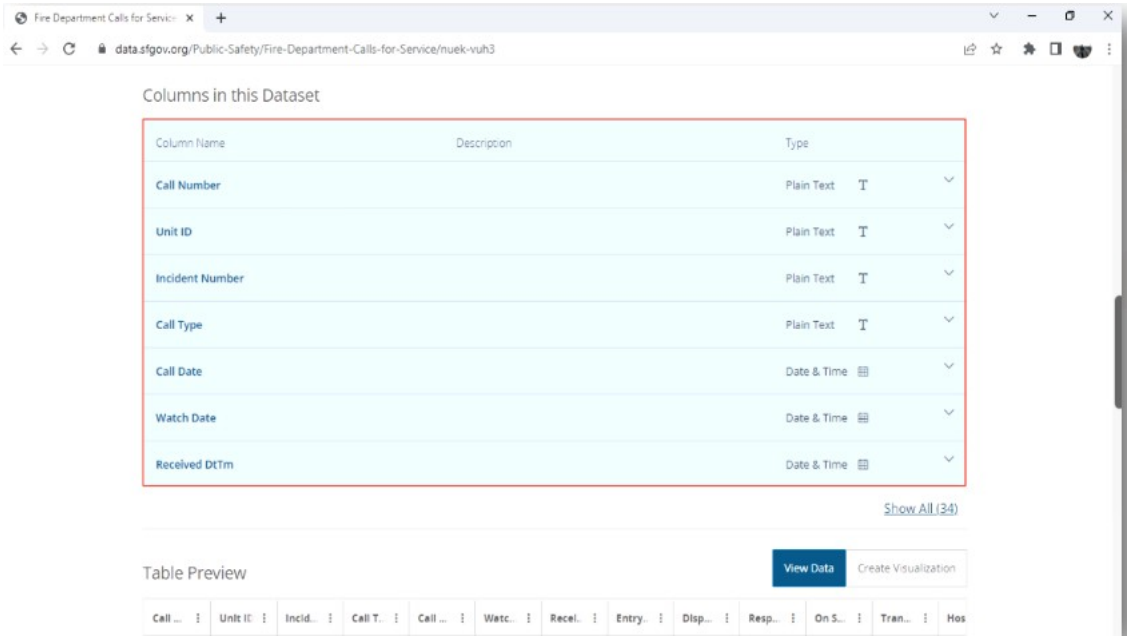
Problem Statement

Data Set: Fire Department Calls for Service
URL: https://data.sfgov.org/Public-Safety/Fire-Department-Calls-for-Service/nuek-vuh3

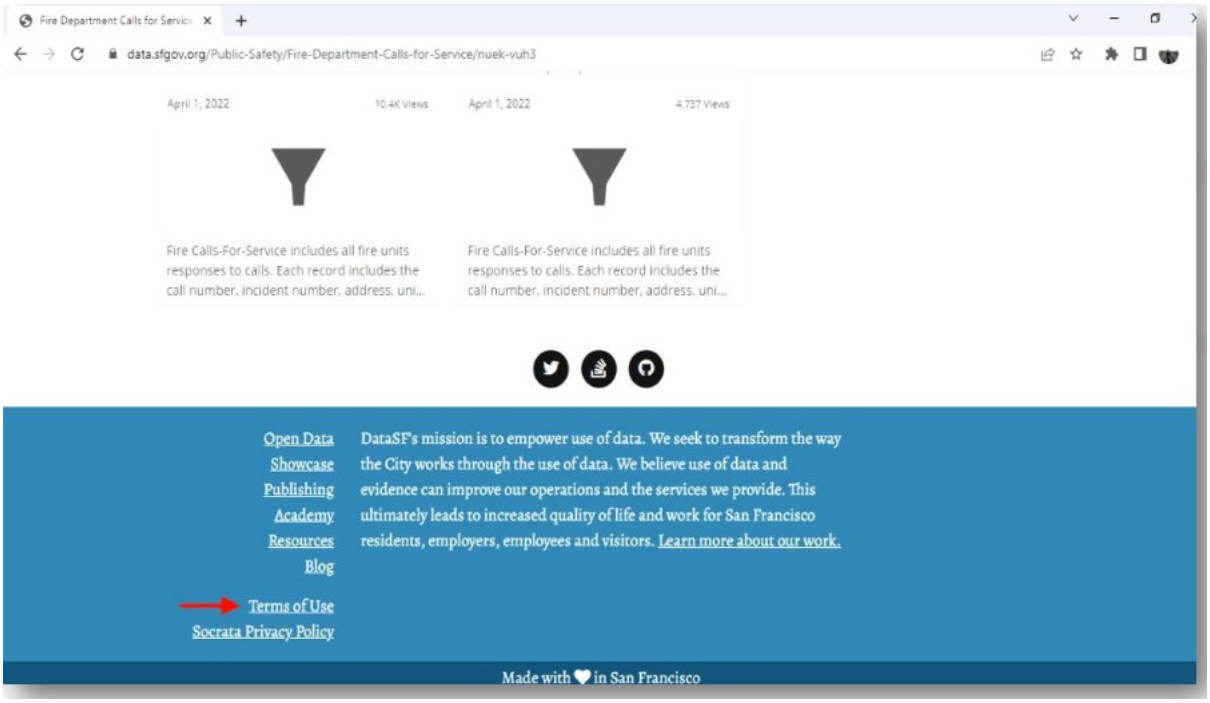
What are the requirements

1. Load the given data file and create a Spark data frame.
2. Use the Spark data frame to answer the following questions.
    1. How many distinct types of calls were made to the fire department?
    2. What are distinct types of calls made to the fire department?
    3. Find out all responses or delayed times greater than 5 mins?
    4. What were the most common call types?
    5. What zip codes accounted for the most common calls?
    6. What San Francisco neighborhoods are in the zip codes 94102 and 94103
    7. What was the sum of all calls, average, min, and max of the call response times?
    8. How many distinct years of data are in the CSV file?
    9. What week of the year in 2018 had the most fire calls?
    10. What neighborhoods in San Francisco had the worst response time in 2018?

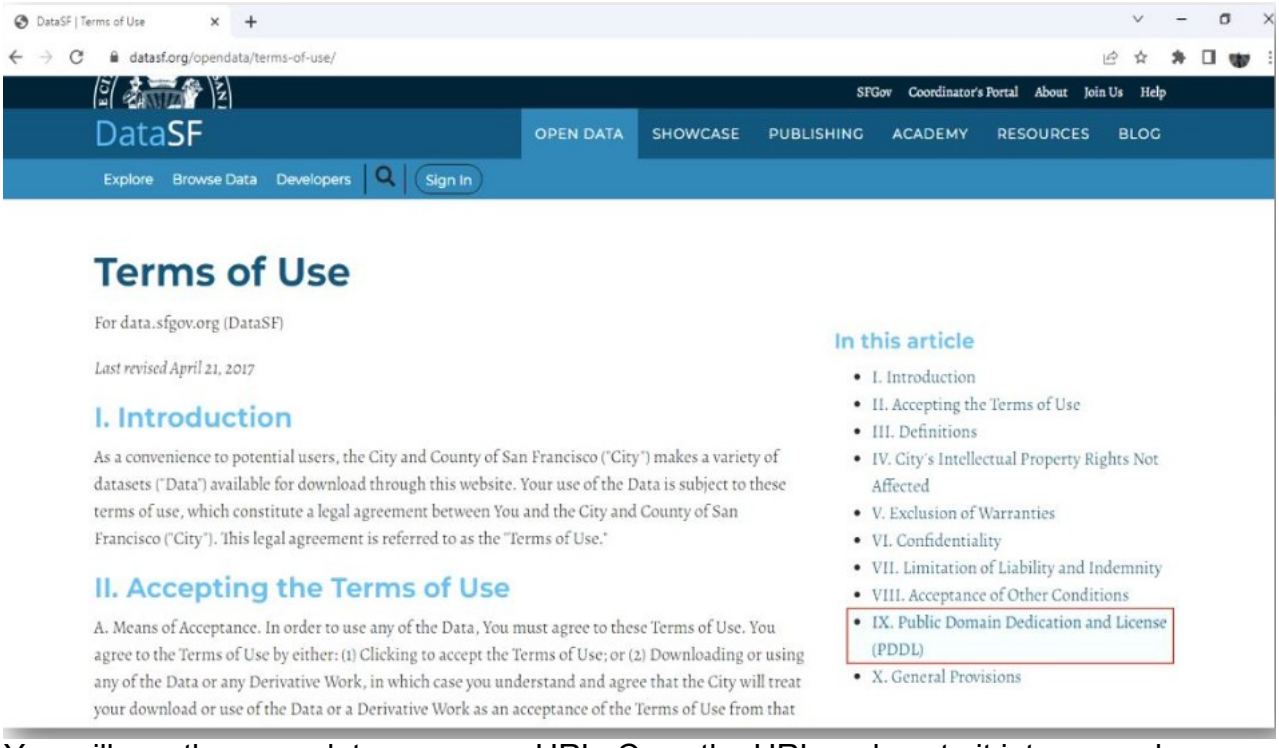
Reference: https://data.sfgov.org/Public-Safety/Fire-Department-Calls-for-Service/nuek- vuh3
This data set represents a response to calls at fire units. If we go to the above link, we can see the 34 column names available in this data set. You can also see the table preview in this page if you scroll down.



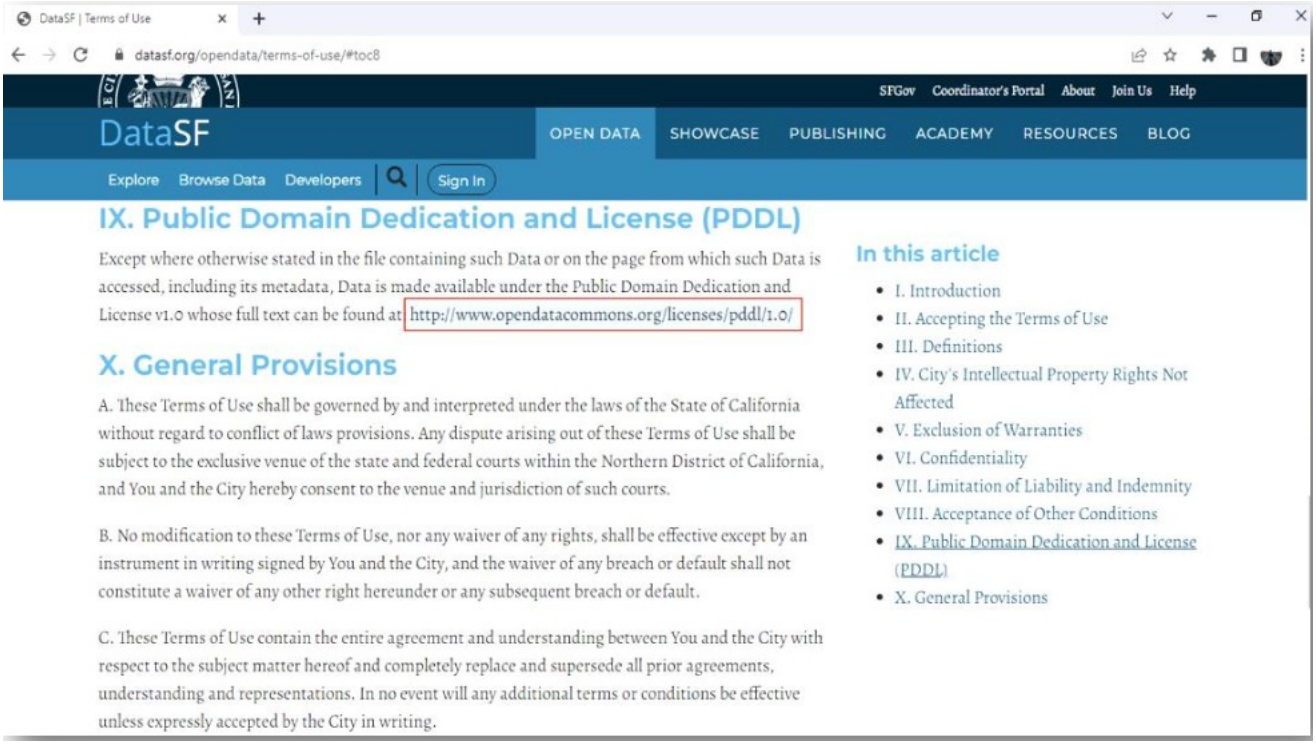
Reference: https://data.sfgov.org/Public-Safety/Fire-Department-Calls-for-Service/nuek- vuh3
Scroll to the bottom of this page, you will see a link for terms of use, click that.



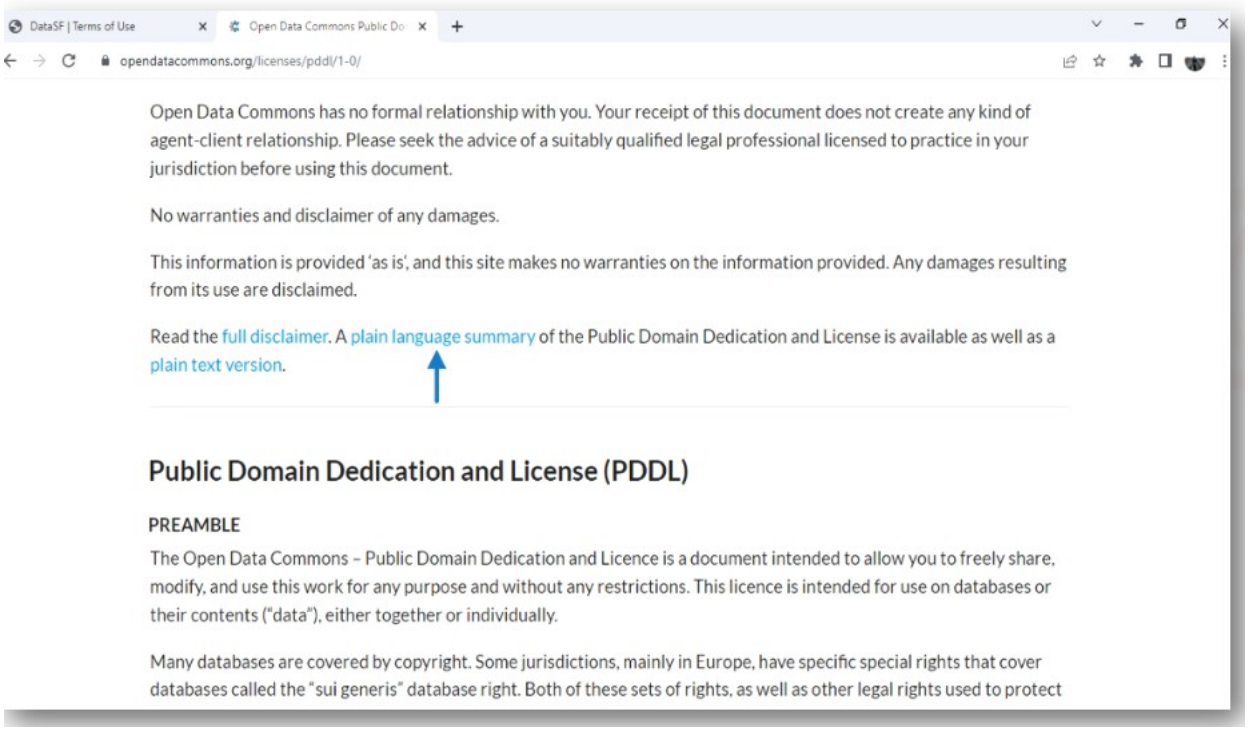
The data set is available under the PDDL terms. Click the PDDL link you see in the side panel.



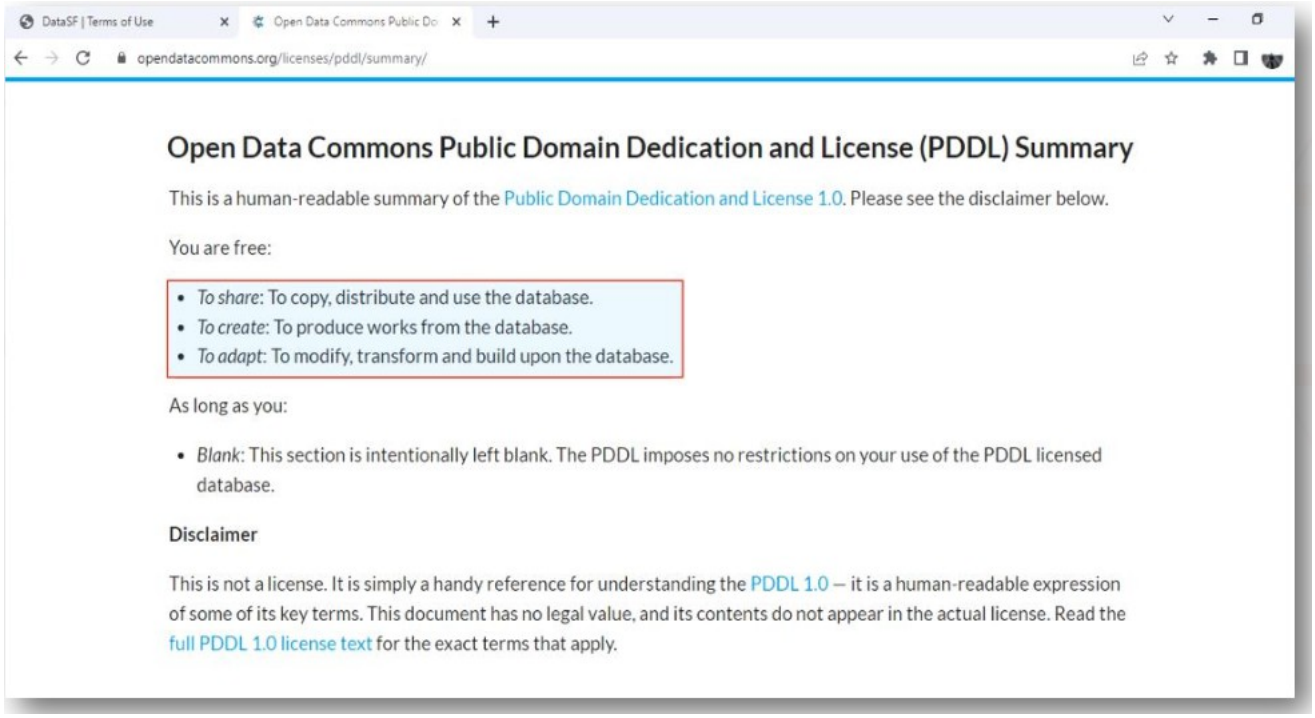
You will see the open data commons URL. Copy the URL and paste it into a new browser tab.



If you go to the open data commons URL, and scroll down, you will see a plain-language summary. Click that.



So you are free to share, create and adapt this data set. It is essential to check the terms of use for any data set that you are using for your learning and experiments. I showed you how to check the license terms for the Fire Calls-For-Service dataset.



You are given a San Francisco fire call response data set. And you are asked to fulfil the following set of requirements

Problem Statement

Data Set: Fire Department Calls for Service
URL: https://data.sfgov.org/Public-Safety/Fire-Department-Calls-for-Service/nuek-vuh3

What are the requirements

1. Load the given data file and create a Spark data frame.
2. Use the Spark data frame to answer the following questions.
    1. How many distinct types of calls were made to the fire department?
    2. What are distinct types of calls made to the fire department?
    3. Find out all responses or delayed times greater than 5 mins?
    4. What were the most common call types?
    5. What zip codes accounted for the most common calls?
    6. What San Francisco neighborhoods are in the zip codes 94102 and 94103
    7. What was the sum of all calls, average, min, and max of the call response times?
    8. How many distinct years of data are in the CSV file?
    9. What week of the year in 2018 had the most fire calls?
    10. What neighborhoods in San Francisco had the worst response time in 2018?

We want to do only two things:

1. Load the data into a Spark Data frame
2. Query it to find the answers to the ten questions shown earlier.

The main objective of the Micro Project is to learn two things:

1. How to load data into Spark
2. How to query data in Spark.

We have two approaches to do it:

1. Using Spark SQL
2. Using Spark Dataframe API

While we work on the solution of the given 10 questions, we will learn the following concepts:

1. Using Spark SQL and DDL statements

2. Spark Data Frame and Data Frame Reader API
3. Working with CSV Files
4. Data Frame Transformations and Actins
5. How to read Spark API Documentation

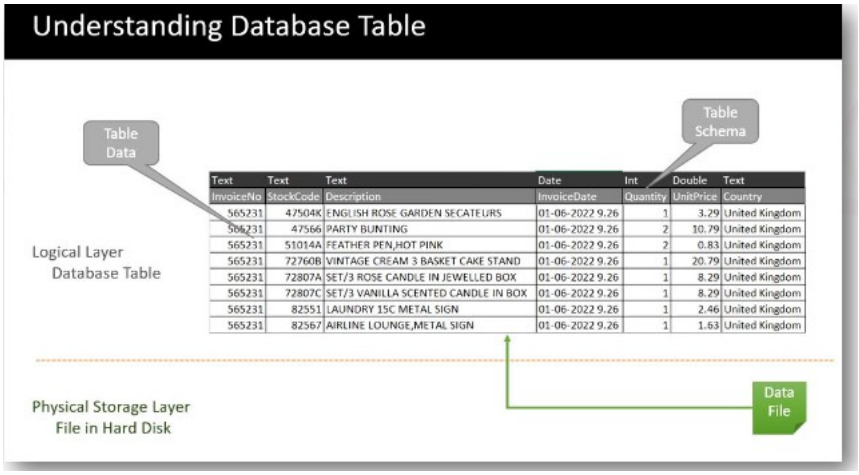
Spark Dataframes - Introduction

We learned that Spark is a data processing platform. We also know that Databases are the most popular and most widely used data processing platforms. They offer two things at a high level.

1. Tables 2. SQL

A database table allows you to load the data in the table. The data in a table is internally stored as a .dbf file, which are stored on the disk. But we don't care about this dbf file and storage layer, we always look at the table, and query the table. A table contains two things:

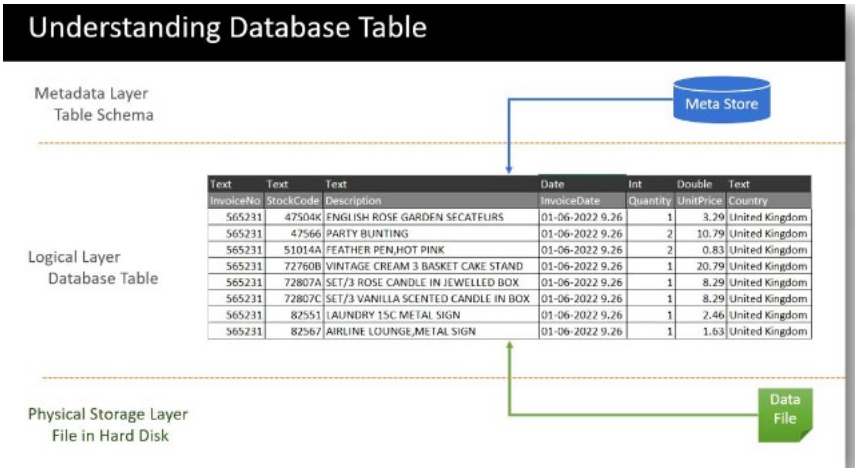
1. Table Schema – It is a list of column names and data types.
2. Table Data – It is the data stored inside the table.



Text	Text	Text	Date	Int	Double	Text
InvoiceNo	StockCode	Description	InvoiceDate	Quantity	UnitPrice	Country
565231	47504K	ENGLISH ROSE GARDEN SECATEURS	01-06-2022 9.26	1	3.29	United Kingdom
565231	47566	PARTY BUNTING	01-06-2022 9.26	2	10.79	United Kingdom
565231	51014A	FEATHER PEN,HOT PINK	01-06-2022 9.26	2	0.83	United Kingdom
565231	72760B	VINTAGE CREAM 3 BASKET CAKE STAND	01-06-2022 9.26	1	20.79	United Kingdom
565231	72807A	SET/3 ROSE CANDLE IN JEWELLED BOX	01-06-2022 9.26	1	8.29	United Kingdom
565231	72807C	SET/3 VANILLA SCENTED CANDLE IN BOX	01-06-2022 9.26	1	8.29	United Kingdom
565231	82551	LAUNDRY 15C METAL SIGN	01-06-2022 9.26	1	2.46	United Kingdom
565231	82567	AIRLINE LOUNGE,METAL SIGN	01-06-2022 9.26	1	1.63	United Kingdom

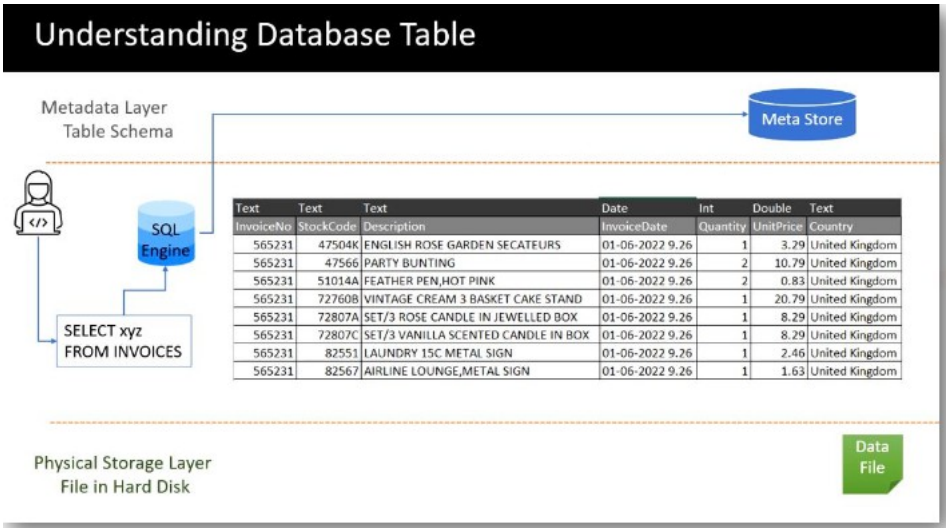
We have three layers to form a table:

1. Storage layer stores the table data in a file
2. Metadata layer stores the table schema and other important information
3. The Logical layer presents you with a database table, and you can execute SQL queries on the logical table.



Text	Text	Text	Date	Int	Double	Text
InvoiceNo	StockCode	Description	InvoiceDate	Quantity	UnitPrice	Country
565231	47504K	ENGLISH ROSE GARDEN SECATEURS	01-06-2022 9.26	1	3.29	United Kingdom
565231	47566	PARTY BUNTING	01-06-2022 9.26	2	10.79	United Kingdom
565231	51014A	FEATHER PEN,HOT PINK	01-06-2022 9.26	2	0.83	United Kingdom
565231	72760B	VINTAGE CREAM 3 BASKET CAKE STAND	01-06-2022 9.26	1	20.79	United Kingdom
565231	72807A	SET/3 ROSE CANDLE IN JEWELLED BOX	01-06-2022 9.26	1	8.29	United Kingdom
565231	72807C	SET/3 VANILLA SCENTED CANDLE IN BOX	01-06-2022 9.26	1	8.29	United Kingdom
565231	82551	LAUNDRY 15C METAL SIGN	01-06-2022 9.26	1	2.46	United Kingdom
565231	82567	AIRLINE LOUNGE,METAL SIGN	01-06-2022 9.26	1	1.63	United Kingdom

When you submit a SQL query to your database SQL Engine, the database will refer to the metadata store for parsing your SQL queries. And the database will throw a syntax error or an analysis error if you are using a column name in your SQL that does not exist in the metadata store.



Text	Text	Text	Date	Int	Double	Text
InvoiceNo	StockCode	Description	InvoiceDate	Quantity	UnitPrice	Country
565231	47504K	ENGLISH ROSE GARDEN SECATEURS	01-06-2022 9.26	1	3.29	United Kingdom
565231	47566	PARTY BUNTING	01-06-2022 9.26	2	10.79	United Kingdom
565231	51014A	FEATHER PEN,HOT PINK	01-06-2022 9.26	2	0.83	United Kingdom
565231	72760B	VINTAGE CREAM 3 BASKET CAKE STAND	01-06-2022 9.26	1	20.79	United Kingdom
565231	72807A	SET/3 ROSE CANDLE IN JEWELLED BOX	01-06-2022 9.26	1	8.29	United Kingdom
565231	72807C	SET/3 VANILLA SCENTED CANDLE IN BOX	01-06-2022 9.26	1	8.29	United Kingdom
565231	82551	LAUNDRY 15C METAL SIGN	01-06-2022 9.26	1	2.46	United Kingdom
565231	82567	AIRLINE LOUNGE,METAL SIGN	01-06-2022 9.26	1	1.63	United Kingdom

The schema is essential for the database table and for the SQL expressions to work correctly. The table data is stored in the dbf files behind the table. So if your SQL query is correct and passes all the schema validation, you will see query results. The database will read data from the "dbf" file, process it according to your SQL query, and show you the results. And that is all about the data processing in databases.

Apache Spark offers you two ways of data processing: 1. SparkDatabase and SQL 2. Spark Dataframe and Dataframe API

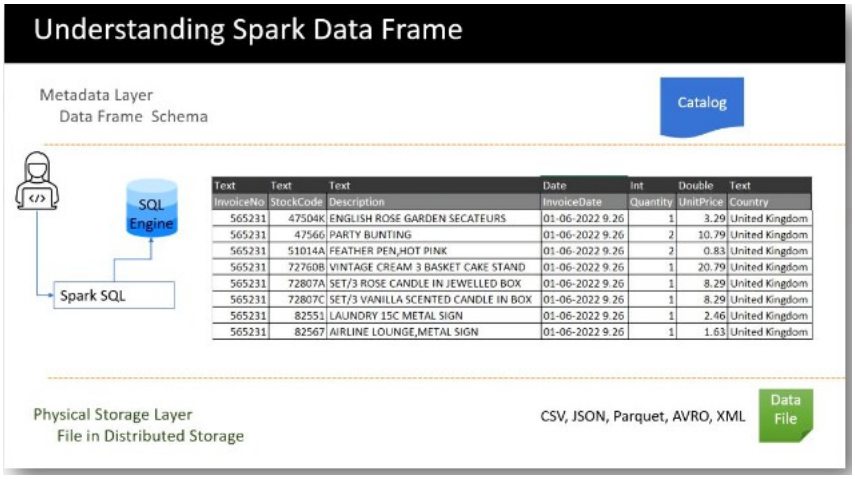
The first approach is precisely the same as a typical database. You will create table and load data into the table. Spark table data is internally stored in the data files. But these files are not dbf files like in the case of a database. Spark gives you the flexibility to choose the file format and supports many file formats such as CSV, JSON, Parquet, AVRO and XML.

The DBF file format and database engine were designed to support structured data. However, Spark supports structured, semi-structured, and unstructured data.

The spark storage layer also supports distributed storage such as HDFS and Cloud storage such as Amazon S3 and Azure ADLS. So you are not limited to disk storage capacity. You can use distributed storage and store large data files. Spark also has a metadata store for storing table schema information. So that part is similar to the databases.

Then Spark also comes with an SQL query engine and supports standard SQL syntax for processing and querying data from Spark tables.

The second approach to data processing in Apache Spark is **Spark Dataframe and Dataframe API**. Spark Dataframe is structurally the same as the table. However, it does not store any schema information in the metadata store. Instead, we have a runtime metadata catalog to store the Dataframe schema information. It is similar to the metadata store, but Spark will create it at the runtime to store schema information in the catalog.



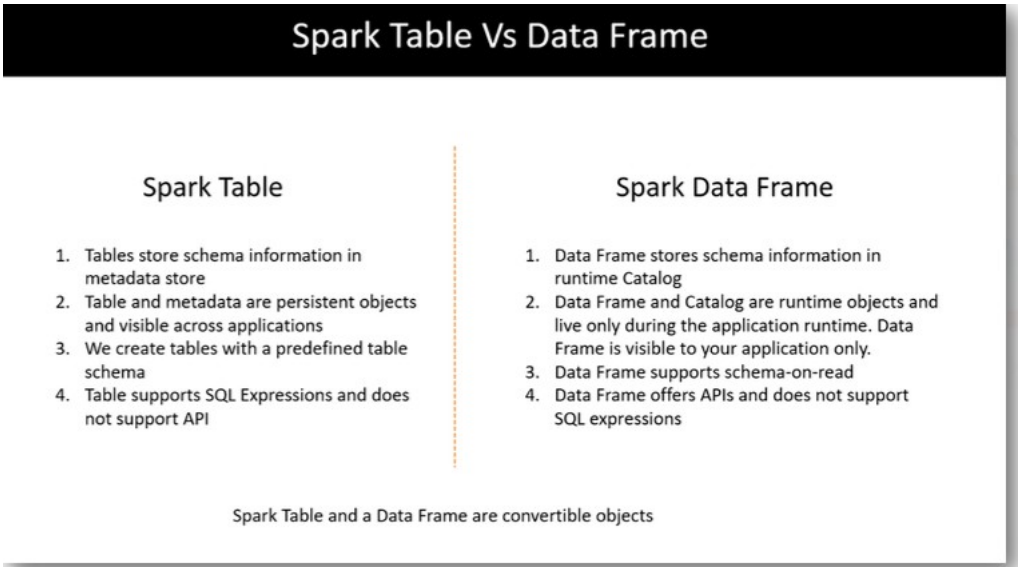
We have two reasons for storing schema information in the catalog.

1. Stark Dataframe is a runtime object – You can create a Spark Data frame

   at runtime and keep it in memory until your program terminates. Once your program terminates, your Dataframe is gone. So, it is an in-memory object.

2. Spark Dataframe supports schema-on-read – Dataframe does not have a fixed and predefined schema stored in the metadata store. We load the data into a Dataframe and tell the schema when loading the data. And Spark will read the file, apply the schema at the time of reading, create the Dataframe using the schema and load the data.

Here is a quick summary discussing the differences between Spark Tables and Dataframes.



Creating Spark Dataframe

You have already seen the requirement for creating a Dataframe using the San Francisco fire call response data set. The data set is already available in the Databricks community cloud, and it is made available for you by the Databricks for learning purposes. You can access it from the following location in your Databricks community cloud. There is no need to download the data from the source and bring it to the cloud. That part is already done for you.

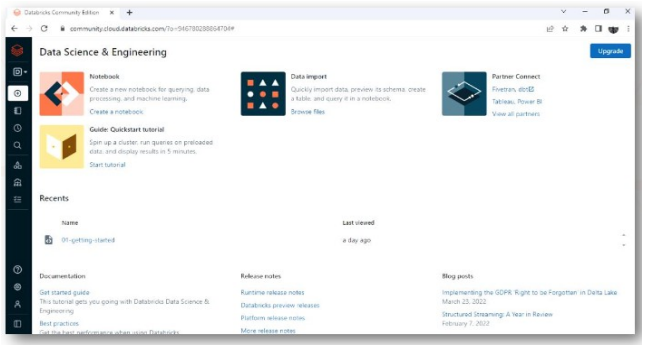
Problem Statement

Data Set: Fire Department Calls for Service
Location: /databricks-datasets/learning-spark-v2/sf-fire/sf-fire-calls.csv

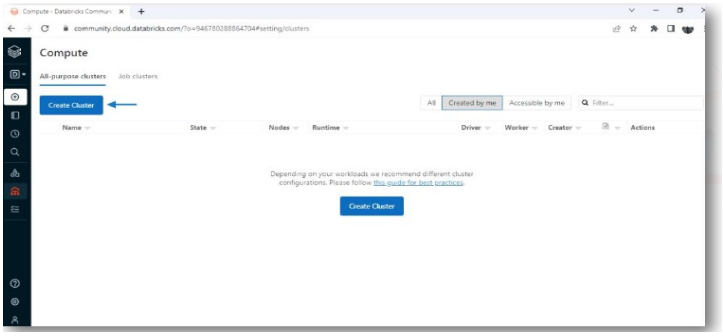
What are the requirements

1. Load the given data file and create a Spark data frame.
2. Use the Spark data frame to answer the following questions.
    1. How many distinct types of calls were made to the fire department?
    2. What are distinct types of calls made to the fire department?
    3. Find out all responses or delayed times greater than 5 mins?
    4. What were the most common call types?
    5. What zip codes accounted for the most common calls?
    6. What San Francisco neighborhoods are in the zip codes 94102 and 94103
    7. What was the sum of all calls, average, min, and max of the call response times?
    8. How many distinct years of data are in the CSV file?
    9. What week of the year in 2018 had the most fire calls?
    10. What neighborhoods in San Francisco had the worst response time in 2018?

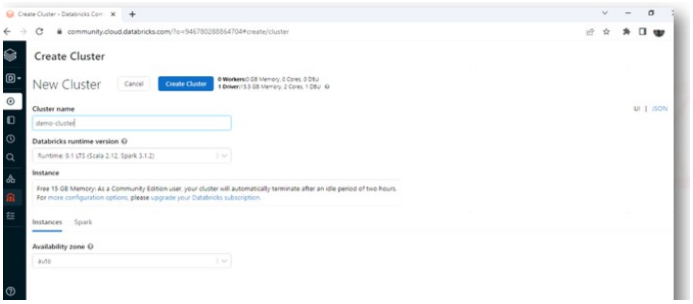
Start your Databricks Workspace.



Go to the Compute menu and create a new cluster.



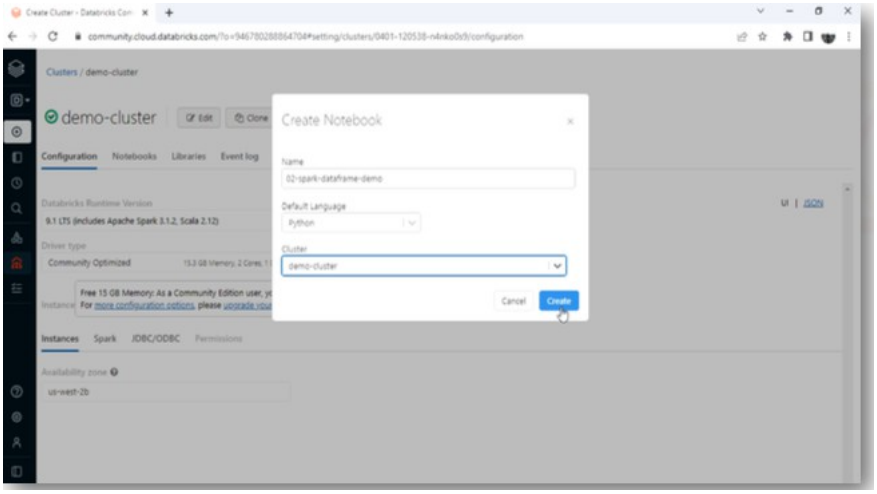
Give a name to your cluster and create it. Your cluster will start in a few minutes, and Databricks will terminate it if your cluster is idle for 2 hours. Once terminated, you cannot start it again. So if you want to work the next day, you should delete the older terminated cluster and create a new one. You can use your cluster for as long as you want and delete it when you are done.



Once your cluster is up and running, you can go ahead to the "*create*" menu options and create a new notebook. Give a name to the notebook. Every notebook should have a default

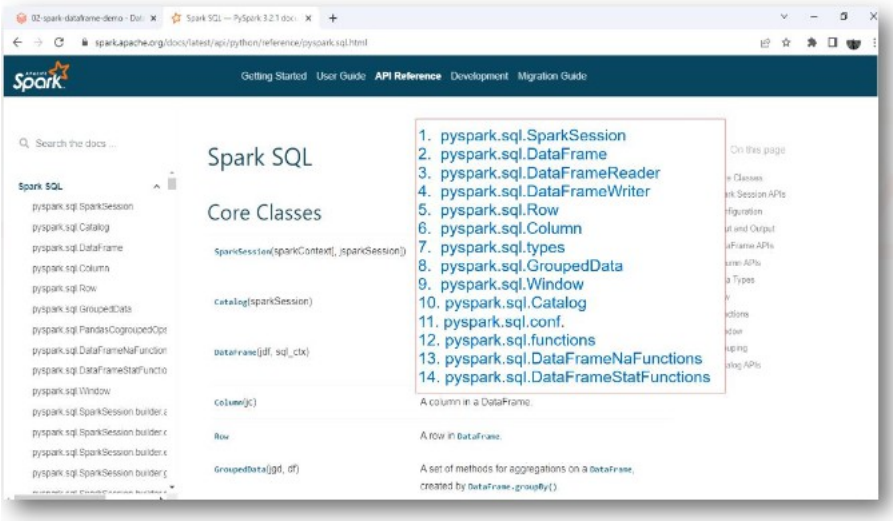
language. You have four language options: Python, Scala, SQL and R.

We want Python for Spark programming, so keep the default language Python. Your notebook will not run without a cluster. So you should also attach your cluster with the notebook. And finally, hit the create button.

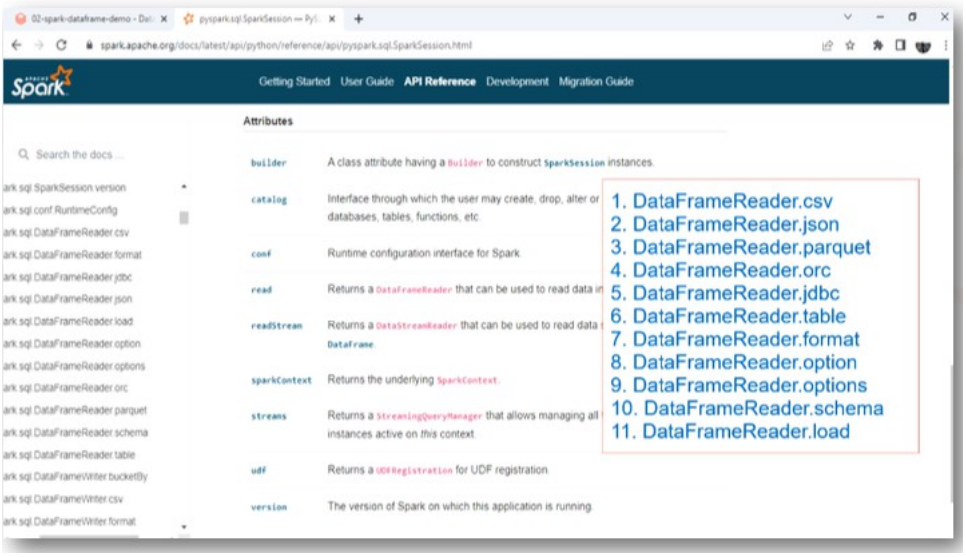


Once the notebook is created, you can write the code to create a Spark Dataframe using the San Francisco fire call response data set. And here we have the code for the same shown below. **Reference: (02-spark-dataframe-demo.ipynb)** I am using spark.read which is a Spark Session object. It is the entry point for the Spark programming APIs. The rest of the code, such as format, option, and load, are the methods of the DataframeReader. I am telling the DataframeReader that the data file is a CSV file. Then I am telling that the CSV file comes with a header row. I am also saying that the DataframeReader should infer schema from the file itself. Finally, I call the load method so the DataframeReader can load the file from the given location. The code will return a dataframe, and I am assigning the dataframe to the fire_df variable.

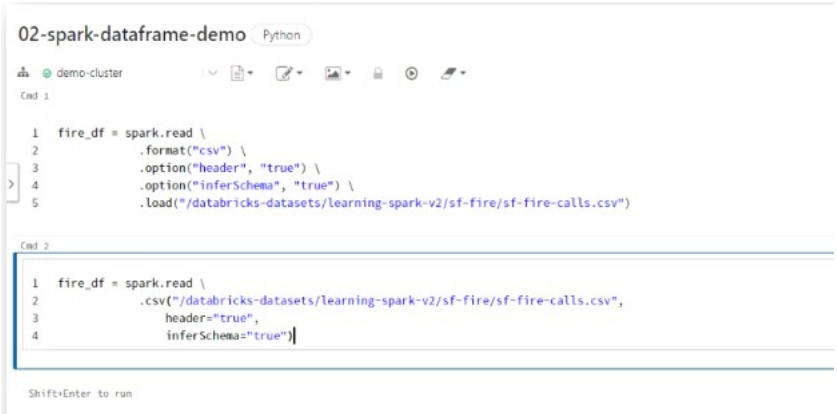
Visit https://spark.apache.org/docs/latest/api/python/reference/pyspark.sql.html to go to the Spark SQL documentation. You will see that all the things listed in this page belongs to the 14 objects in the pyspark.sql package which are listed in the image below. Each object offers some attributes and methods.



Visit https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.html to go to the DataFrameReader object under the Spark SQL documentation. The DataFrameReader offers eleven methods as shown in the screenshot below.

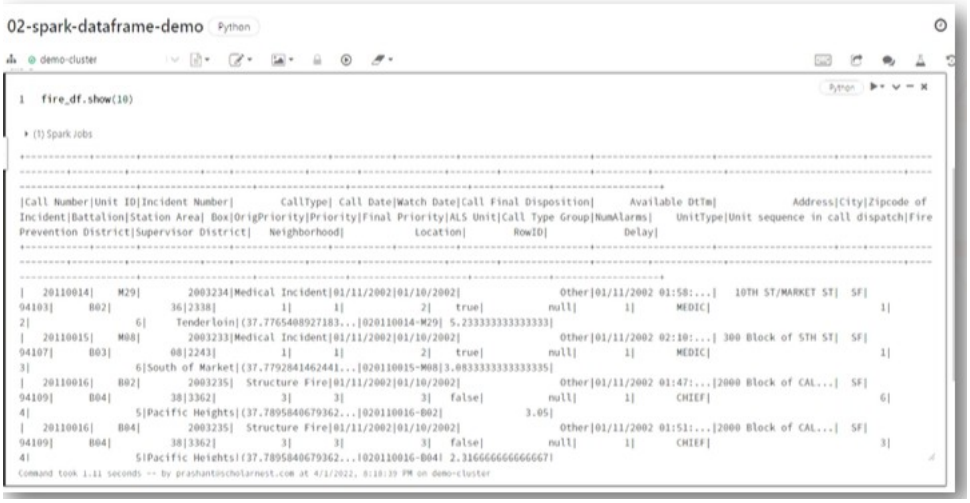


We can also create the same Dataframe using CSV method as shown in the second cell of the screenshot. In the first cell, I am using the format method to tell that I want to load a CSV file and the option method to set other configurations. In the second cell, we are using the shortcut CSV method. I am not setting the format and options. The CSV method implicitly tells that the format is CSV, and all other details will go as function arguments. So, both the code are doing the same thing, but I prefer the first one as this code is standardized for loading data from any file format.

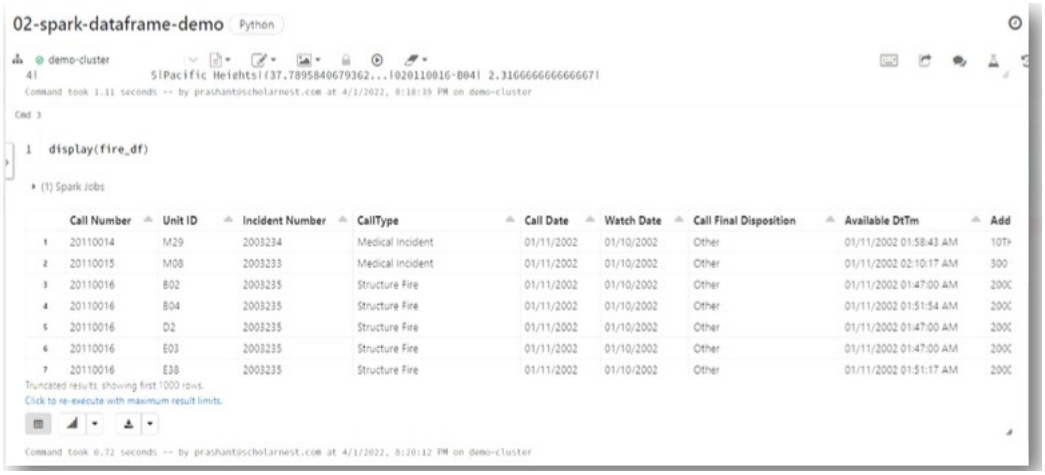


I created a DataFrame, and now I want to see some records from this dataframe. You can use

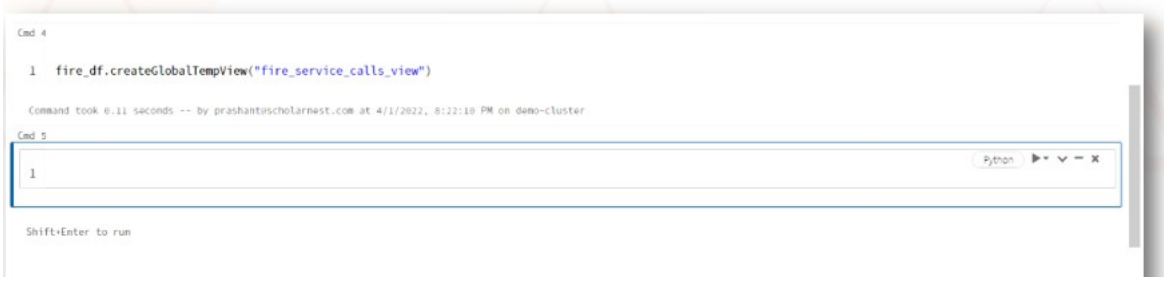
the show method. I take this data frame variable and call the Dataframe show method. I want to see ten records, so I use show(10). And you can see the output below the code cell. But it doesn't look nice and easy to read.



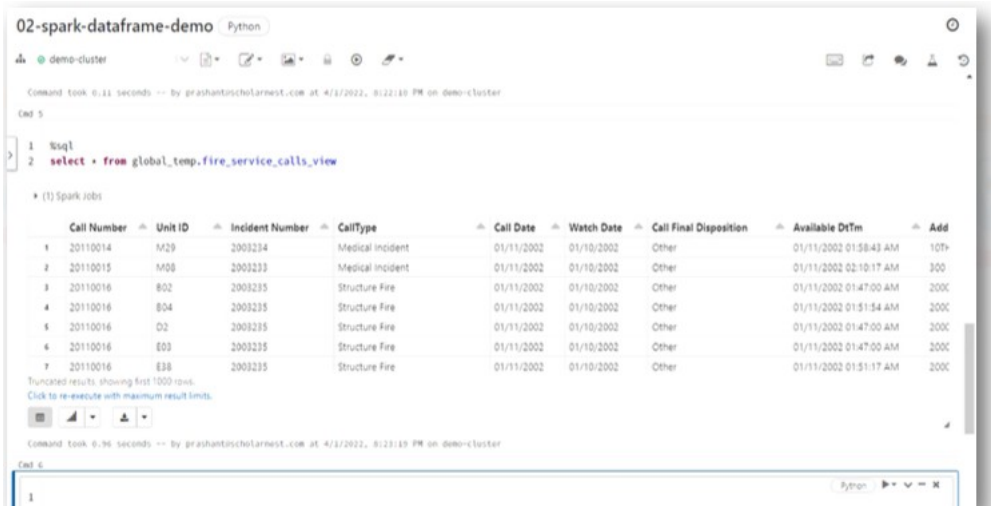
So, you can use display() function. And now the output looks much better and easy to read. The display function is not part of Apache Spark. It is an additional tool by Databricks to format the output of a DataFrame. So it takes a Dataframe and shows the content of the data frame.



In the code shown below, I am taking my dataframe and creating a global- temporary view for the dataframe. The view name is *fire_service_calls_view*. I am not converting my Dataframe to a table. Instead, I am creating a global- temporary view that is similar to the table.

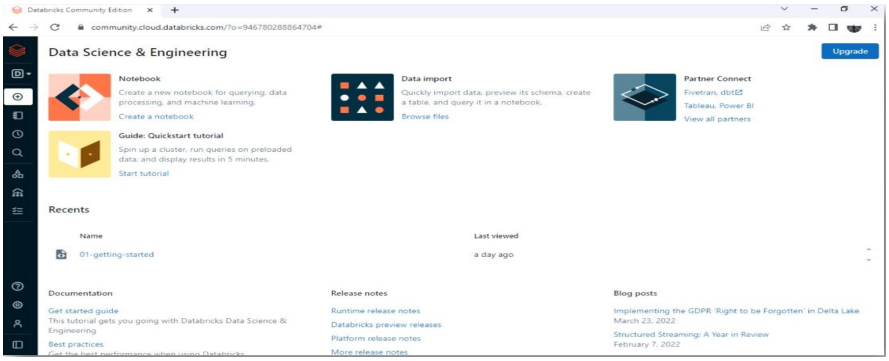


You can run SQL expression on the global-temporary view. But you need to change the cell type from Python to SQL. The global_temp is a hidden database, and all global temporary tables are created in this global_temp database. So you must use global_temp for accessing a global-temporary view.

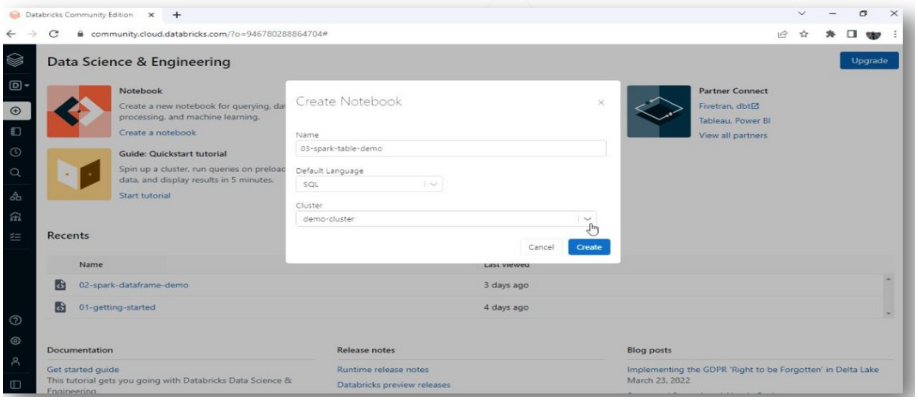


Creating Spark Tables

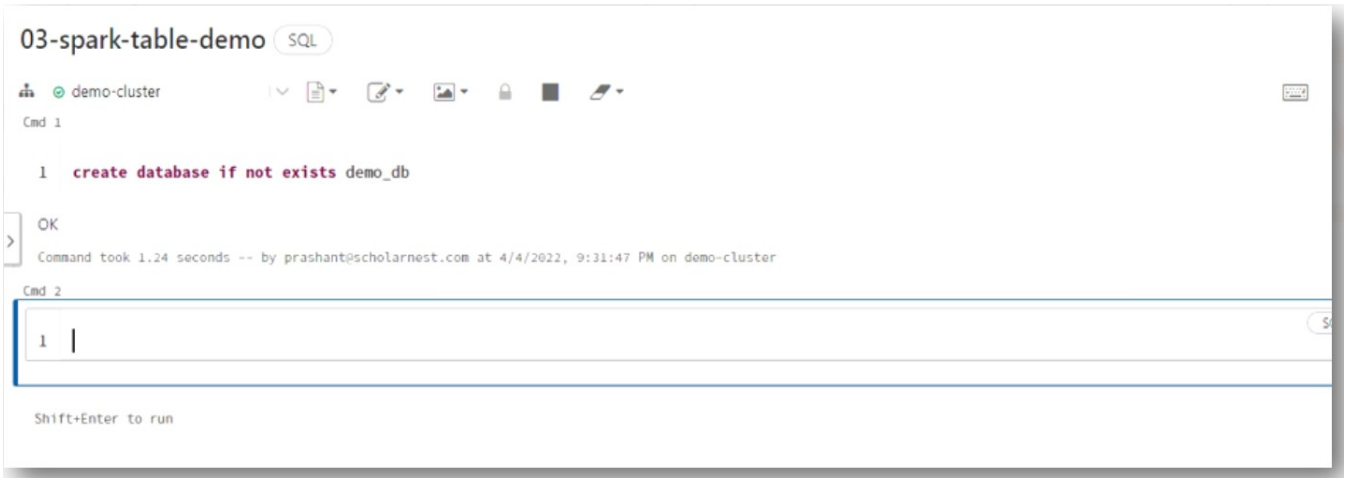
Start your Databricks Workspace.



Create a new notebook. (**Reference: 03-spark-table-demo.sql)** Give a name to your notebook. Choose your notebook's default language. This time, we want to use Spark SQL, so the default language for the notebook should be SQL. Attach a cluster to your notebook. If you do not have a running cluster, go back to the Compute menu and create a new cluster.



Create a database as shown below. The code shown here is purely a SQL DDL Statement. A database should have been created.



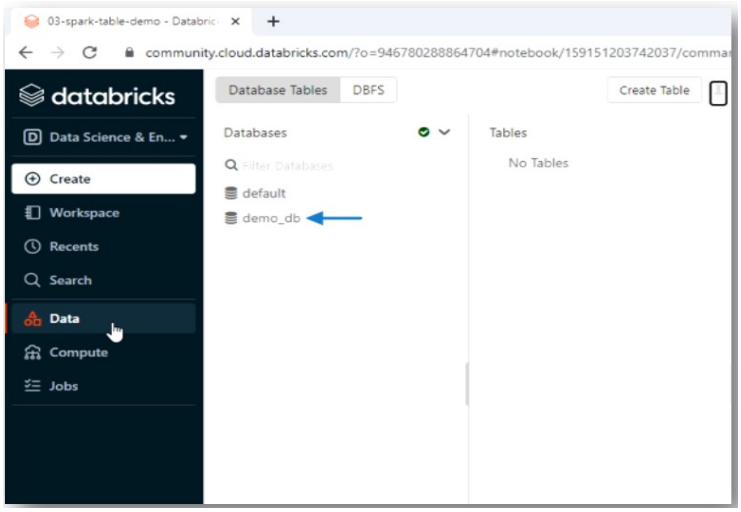
To see your database, click the Data menu item, and you will see two databases:

1. Spark created one default database for you. So if you create a table, it will be created inside the

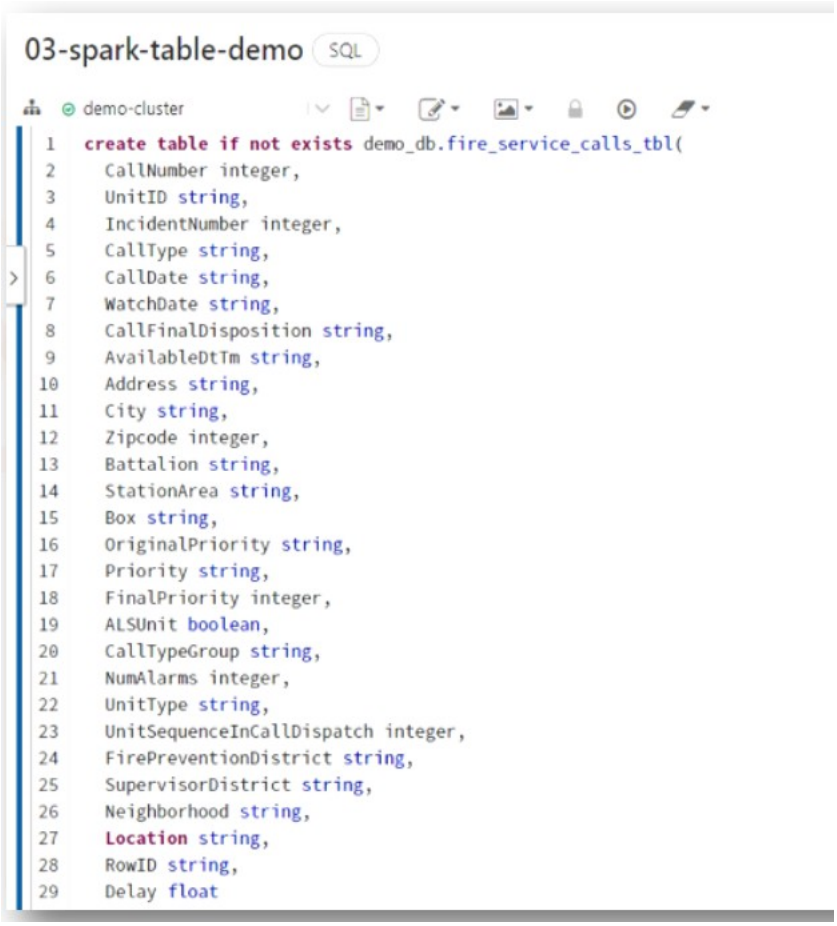
   default database. But we do not recommend using the default database.

2. Instead, you should create a database for your project and keep all your project tables inside your

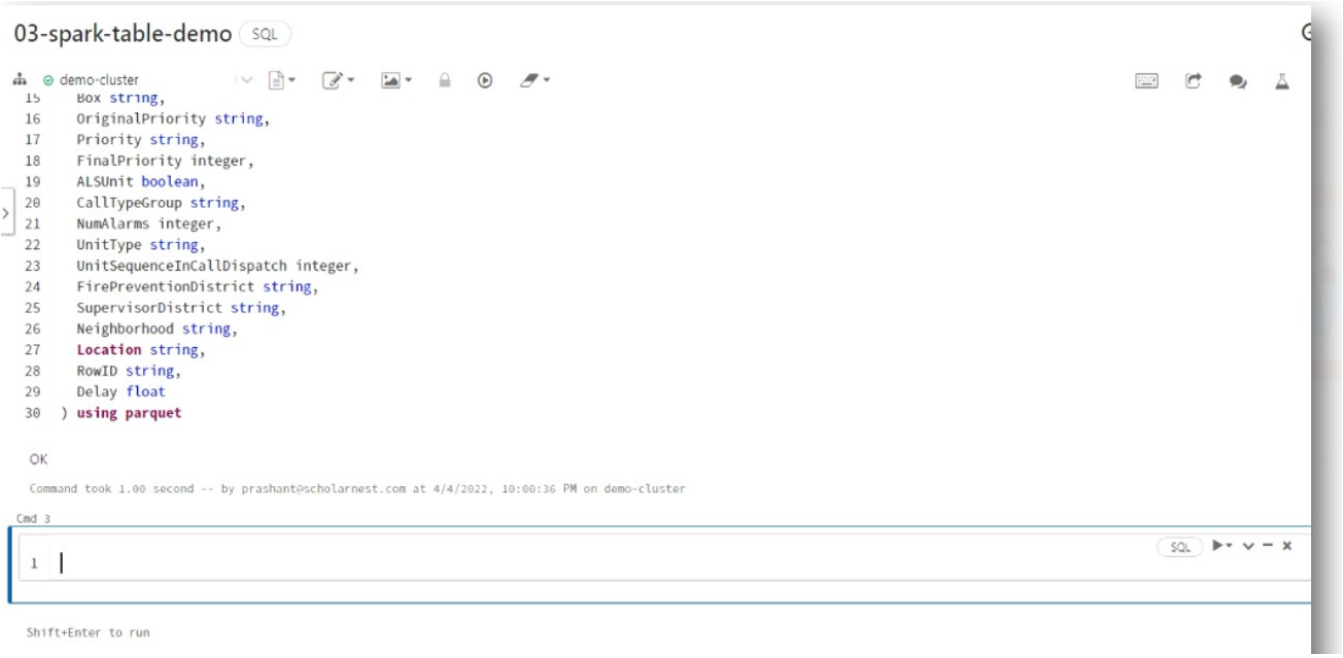
   database. So we have already created one database for ourselves.



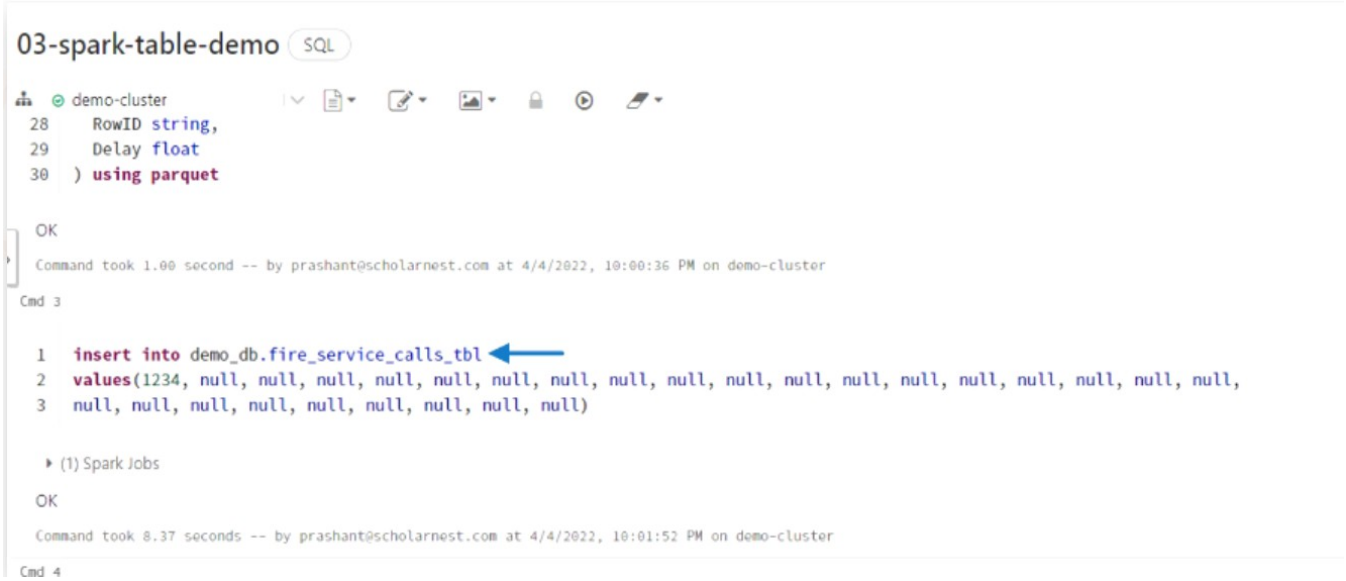
Now the next aim is to create a table. The creation of table also follows a create table DDL statement as shown below. We have 28 columns in the fire response call dataset. Most of them are string columns, but some are integer and float columns.



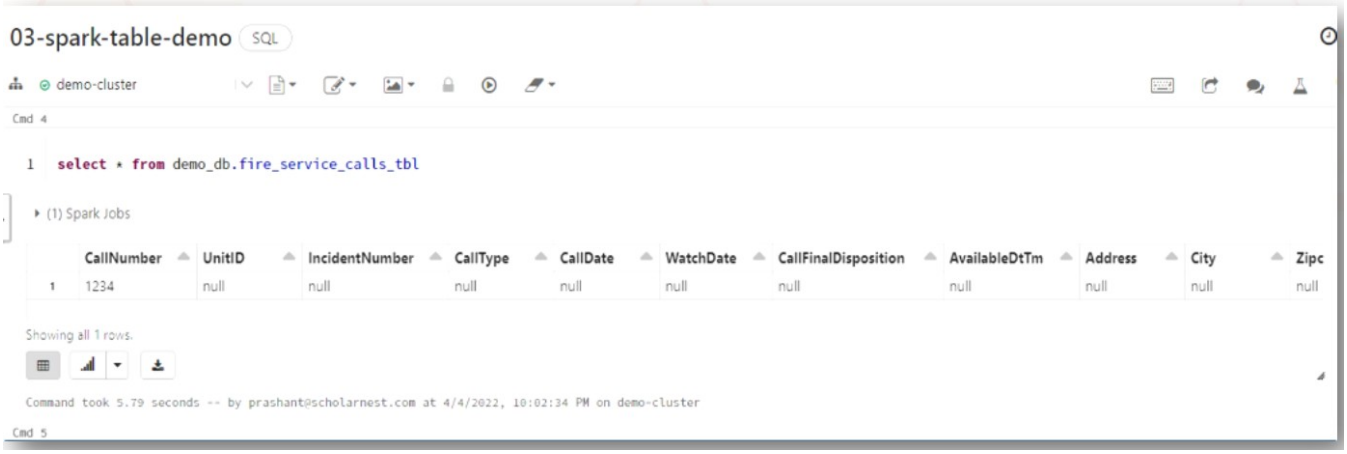
Finally, we will specify the file format for the table. I will be using parquet for the table. Every database table internally stores data in files. So for this table, I want Spark to use parquet file format. And if we run this following code, a table would be created for us.



We do not have any data in the table. So we can use the insert-into command to insert some records into the table in a traditional database table, as shown in the image below. I am inserting one row into the *demo_db.fire_service_calls_tbl*. All the column values are null except the first column.



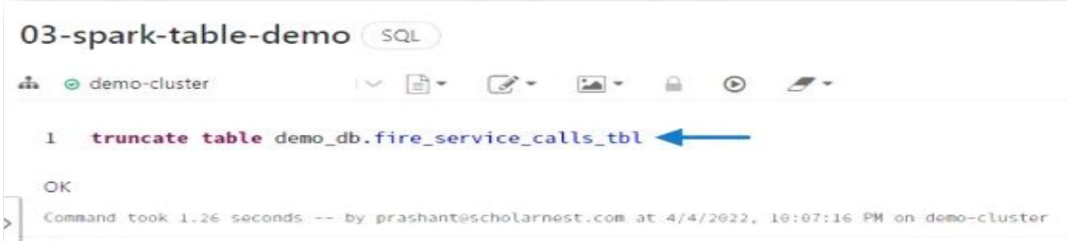
You can run a select * expression to see the content of the table. And we can see in the output that data is inserted successfully.



Spark tables are expected to hold millions and billions of records. And we can't write the insert-into values command to load those records. The insert into values expression doesn't make much sense in the big data world.

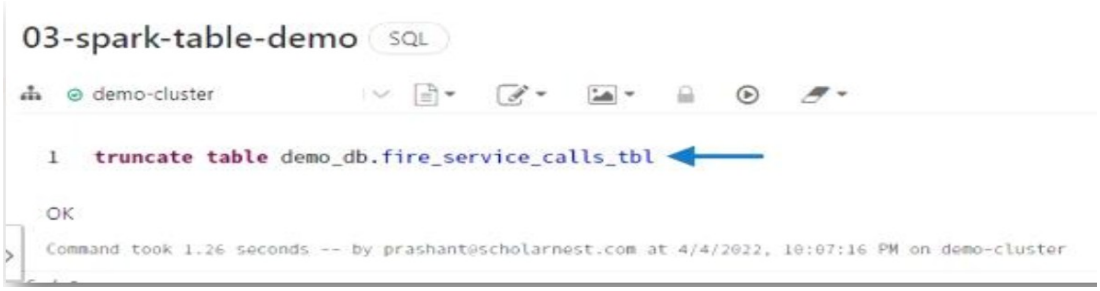
You will not find a project that uses insert into values to load data into Spark tables. There are many ways to load data into Spark tables. But for now, we will insert records into the table from another table.

Let us truncate the table, so that we clean the garbage record that we just inserted. We used the truncate statement because Spark SQL doesn't offer to delete statements.



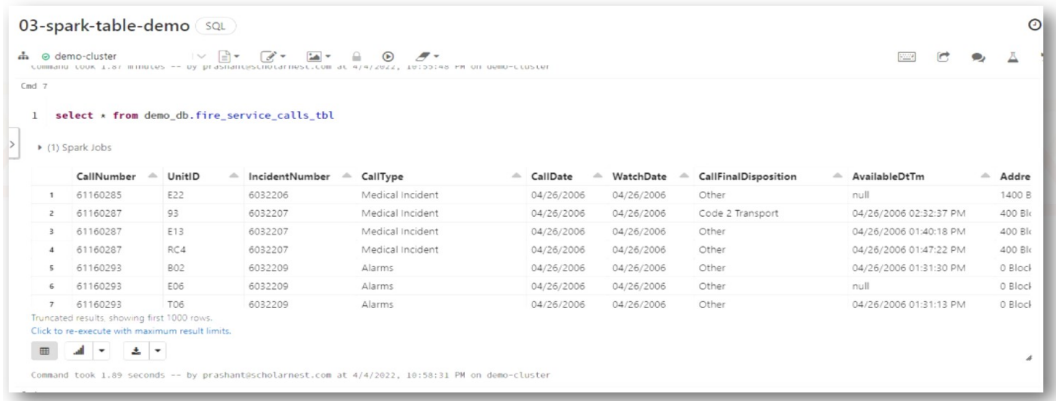
Let us truncate the table, so that we clean the garbage record that we just inserted. We used the truncate statement because Spark SQL doesn't offer to delete statements. You cannot delete data from a Spark Table using Spark SQL. Databricks does offer a delete expression

on Spark tables.



The next target is to insert data into the table after it is truncated. And we have the code for the same in the screenshot shown below. I am inserting records into the *demo_db.fire_service_calls_tbl* from *global_temp.fire_service_calls_view*.

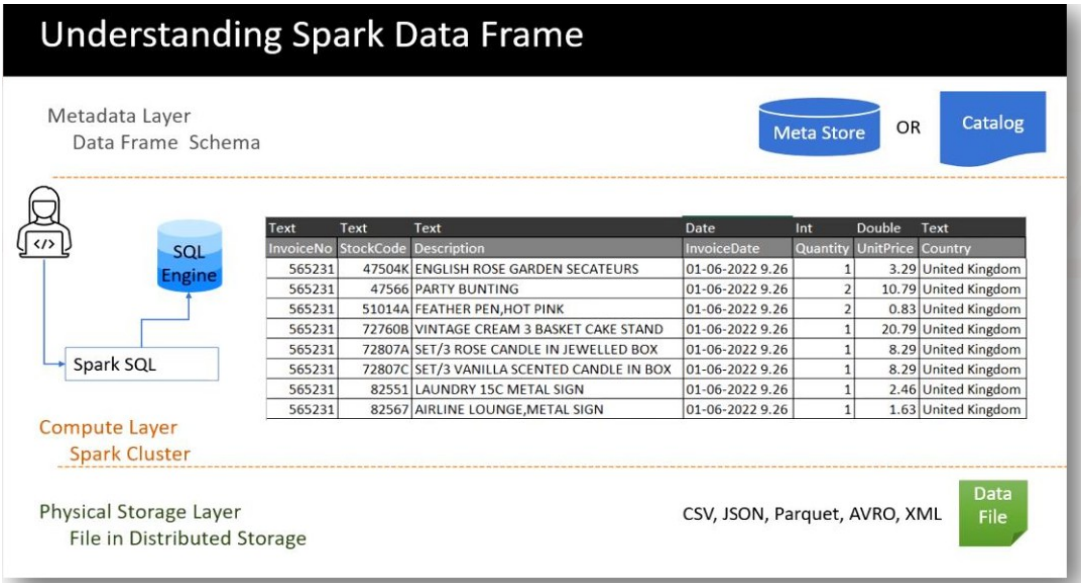
Finally, you can see the content of the table using the select expression.



Problems with Databricks Community

Before we proceed further with the micro project, let us have a look at the below diagram once again. Every Spark Table or the Spark Dataframe lives in three layers:

1. Metadata Layer – It stores Table or Dataframe definition and the schema information.
2. Compute Layer – It runs your Spark SQL engine.
3. Physical Storage Layer – It stores the spark data in a data file.



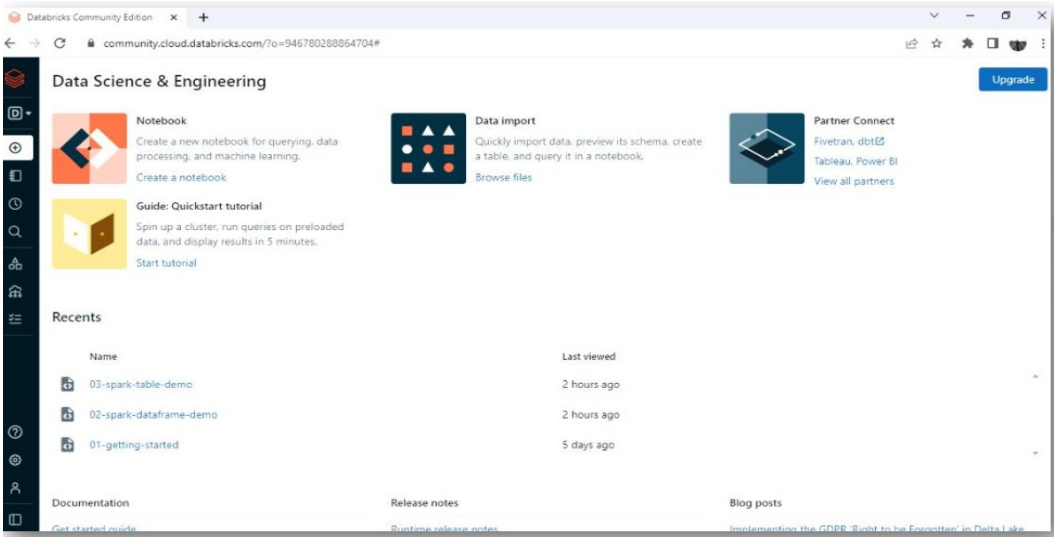
Text	Text	Text	Date	Int	Double	Text
InvoiceNo	StockCode	Description	InvoiceDate	Quantity	UnitPrice	Country
565231	47504K	ENGLISH ROSE GARDEN SECATEURS	01-06-2022 9.26	1	3.29	United Kingdom
565231	47566	PARTY BUNTING	01-06-2022 9.26	2	10.79	United Kingdom
565231	51014A	FEATHER PEN,HOT PINK	01-06-2022 9.26	2	0.83	United Kingdom
565231	72760B	VINTAGE CREAM 3 BASKET CAKE STAND	01-06-2022 9.26	1	20.79	United Kingdom
565231	72807A	SET/3 ROSE CANDLE IN JEWELLED BOX	01-06-2022 9.26	1	8.29	United Kingdom
565231	72807C	SET/3 VANILLA SCENTED CANDLE IN BOX	01-06-2022 9.26	1	8.29	United Kingdom
565231	82551	LAUNDRY 15C METAL SIGN	01-06-2022 9.26	1	2.46	United Kingdom
565231	82567	AIRLINE LOUNGE,METAL SIGN	01-06-2022 9.26	1	1.63	United Kingdom

When you execute a Spark SQL query, or you run a Dataframe code, the Spark SQL engine will refer to the metadata store. And the metadata will tell the following things:

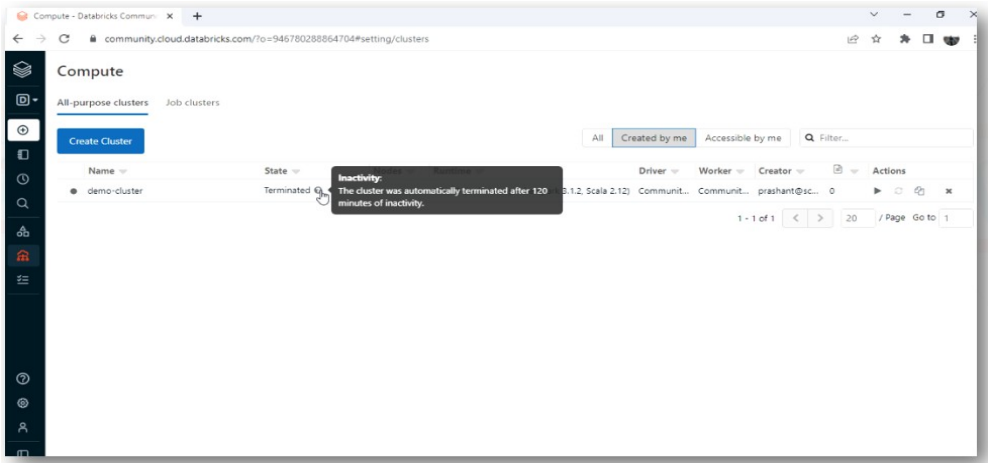
1. Where is the data for this table?
2. What is the schema of the data and table?

If the Spark SQL engine doesn't find metadata data for the table, it throws an analysis exception. If metadata is there, the SQL engine will read the data from the data file and present it to you like a table or Dataframe. So, the three(Metadata, Compute and Storage) layers of Spark are supercritical. You will face problems and see errors when any of these three layers are missing or broken.

Go to your Databricks workspace and go to the compute menu.

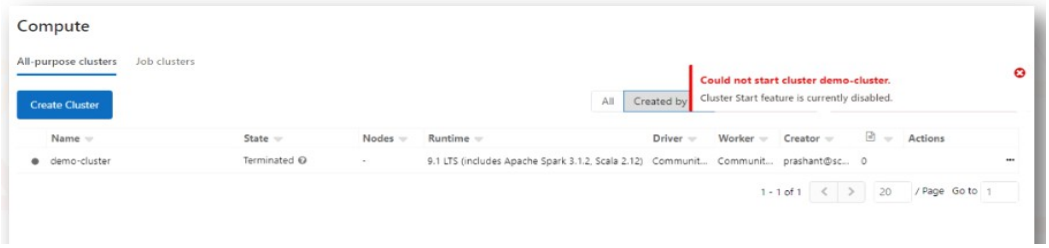


You need a running cluster for doing anything in the Databricks environment because you need a compute layer. But we have an old cluster here which is in terminated state because I left it idle for more than 2 hours, so Databricks terminated my cluster.

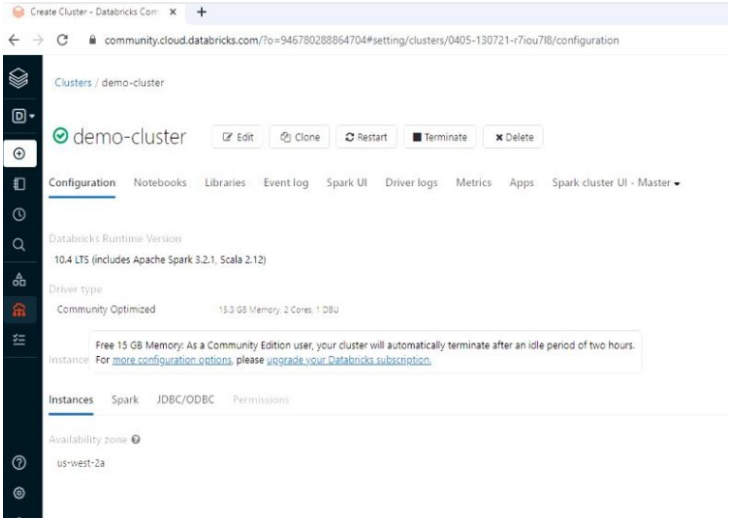


I cannot restart this cluster again. I must create a new cluster if I want to work again. Databricks community edition will terminate your cluster after 2 hours of idle time, and once your cluster is terminated, Databricks will clean up the following things:

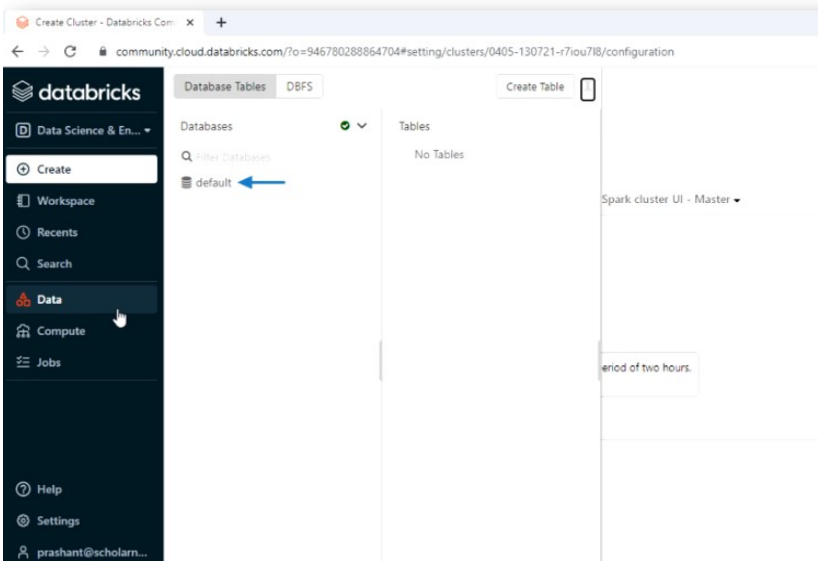
1. Cluster VM (Compute Layer) – You cannot restart your cluster again.
2. Spark Metadata Store (Metadata Layer) – You will lose your databases and tables.



You can delete the old cluster, and create a new one with same name. Creating a cluster takes about 5-10 minutes.



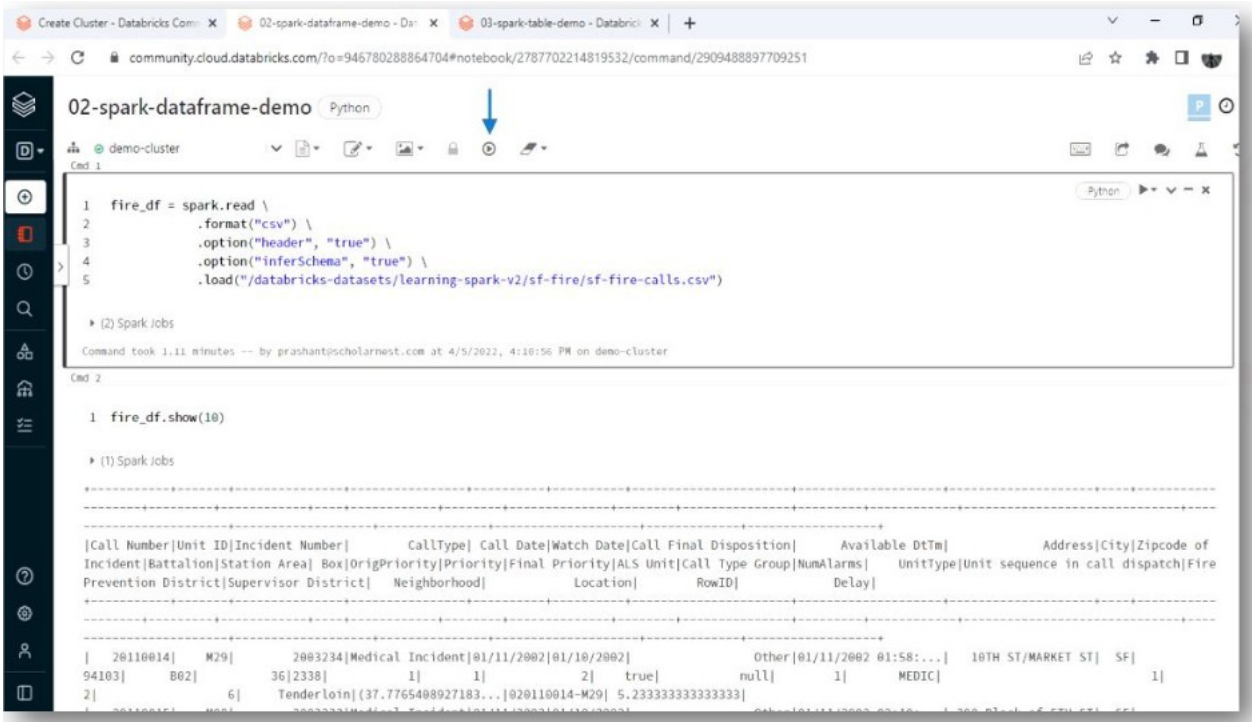
You can click the data menu, and you can see the database and tables. We only have a default database the *demo_db* database that we created yesterday is gone. Because Databricks cleaned my Spark metadata store as my cluster was idle for more than 2 hours. So I lost my database and table definition. This problem does not exist for the full licensed version of the Databricks Cloud.



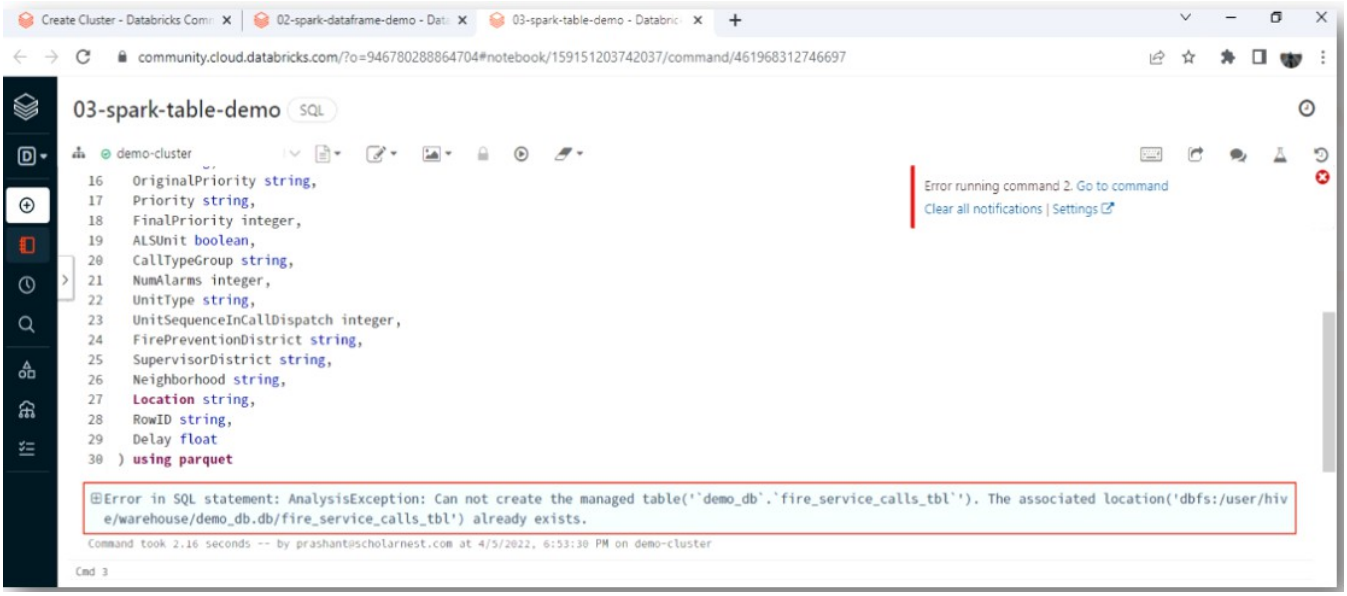
The community edition will clean up the VM and the metadata store. But it doesn't clean up the directories and files. So you do not lose your storage layer. Your data files remain in place even if your compute and metadata layer is gone.

Directories are created in an external distributed storage, and files are stored in these directories on external storage.

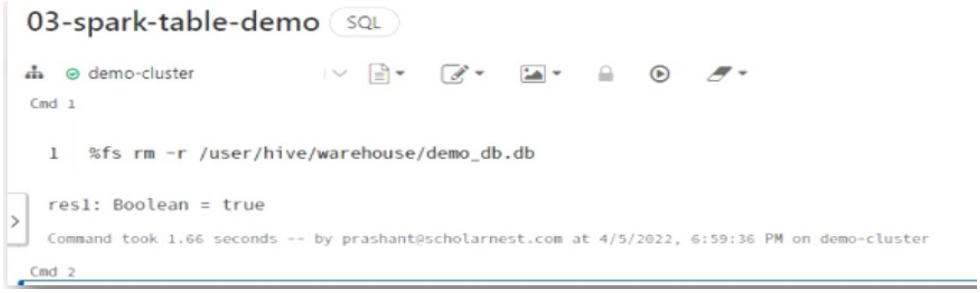
You can go the notebooks you created earlier (02-spark-dataframe-demo.ipynb and 03- spark-table-demo.sql) from the workspace menu option, and click the run all option shown below to run the entire notebook at once. Make sure your cluster is attached to both the notebooks.



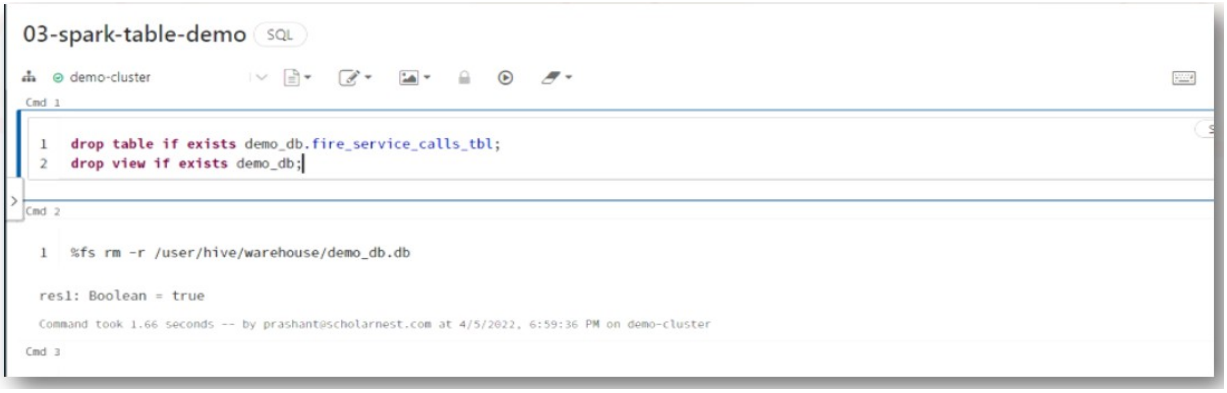
We ran the *02-spark-dataframe-demo.ipynb* successfully but encountered an error in the *03-spark- table-demo.sql* notebook. The error says that “can not create the managed table. The associated location already exists.” We didn't face any errors yesterday when we created this table. But we are getting an error today because Databricks didn't clean the directory and data files. My storage layer is still there. So my data directory is not cleaned.



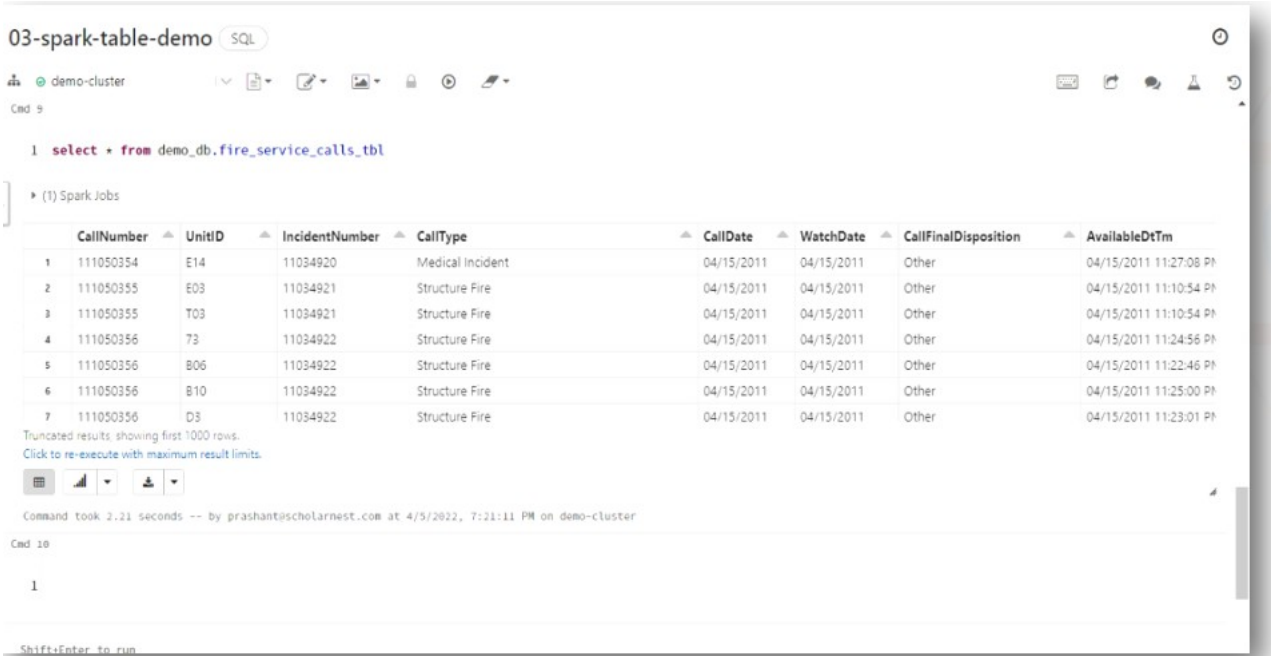
In order to fix the Analysis Exception, we can add a new cell at the top of the notebook and type the filesystem rm command to clean the directory. Copy the directory location from the error message.



I also recommend adding a drop table and drop database command at the top. You can add another cell at the top of the notebook and run two DDL statements to drop your table and database. Do not forget to add a semicolon at the end of the SQL expression.

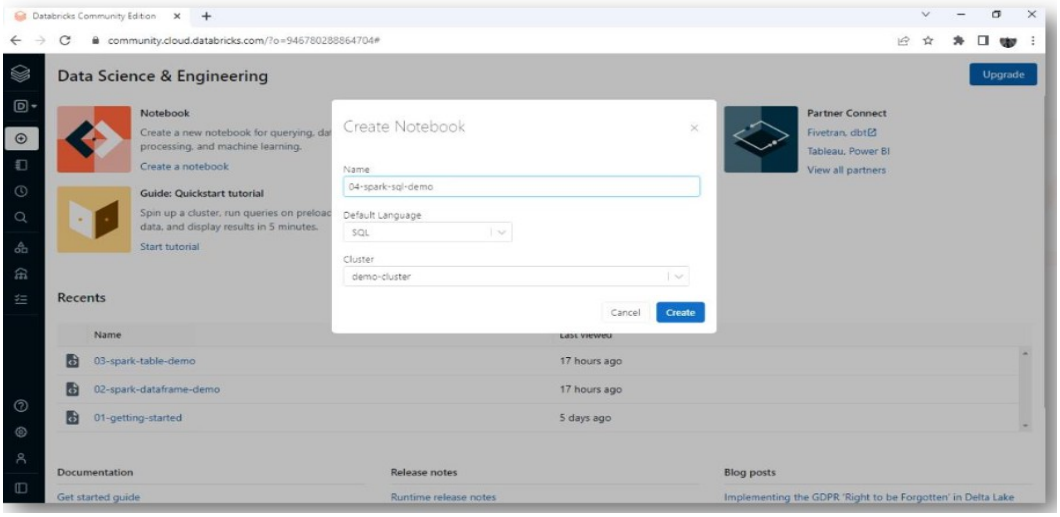


Now, you can go ahead and click the run all button for the *03-spark- table-demo.sql* notebook and it should work perfectly fine without giving any errors.

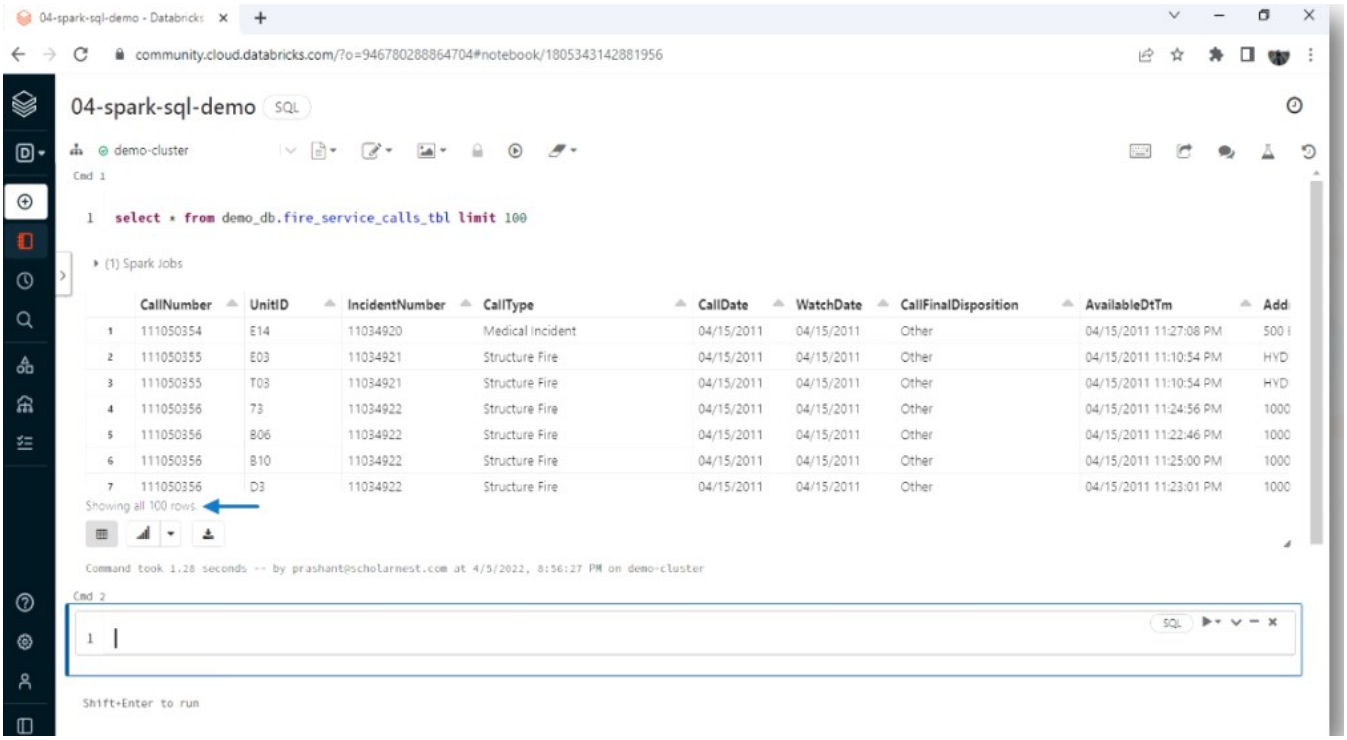


Data Caching Essentials

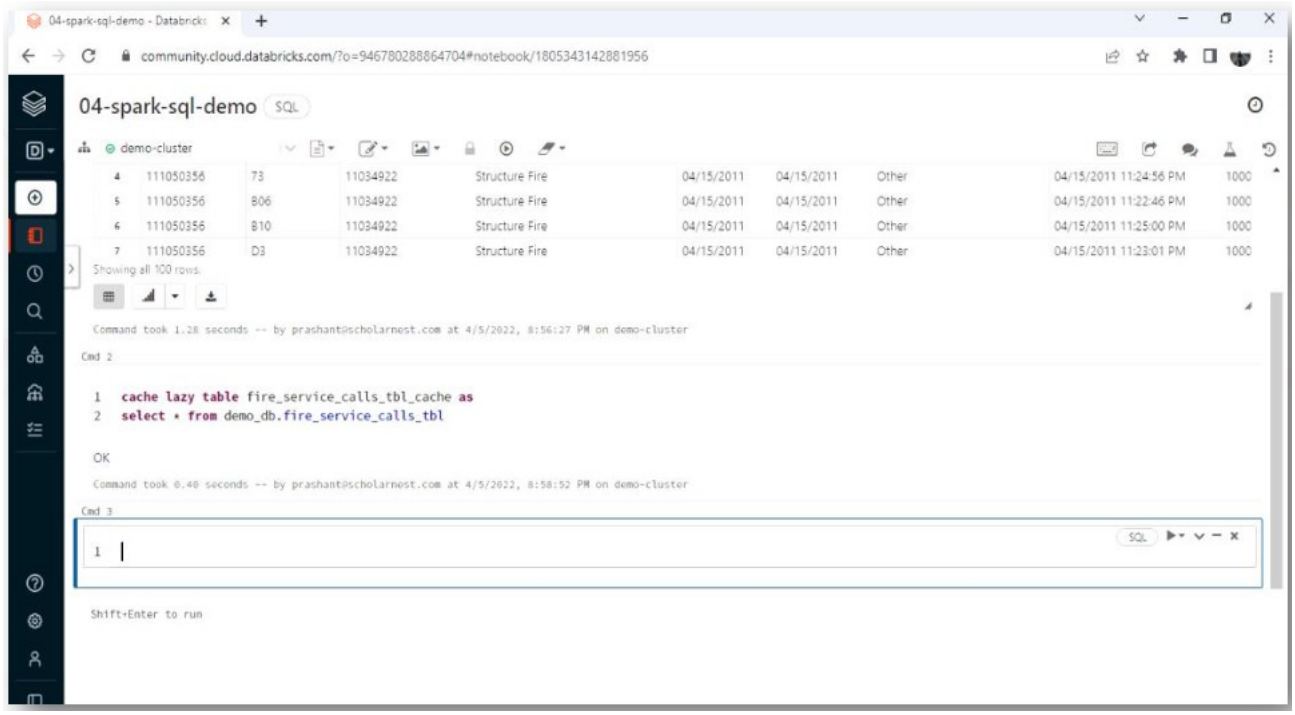
Go to your Databricks workspace. Create a notebook. **(Reference: 04-spark-sql-demo.sql)** We want to use Spark SQL for data analysis. So choose SQL as the default language of your notebook. Make sure to attach a cluster to your notebook.



We created a table for the San Fransisco fire call response dataset. Now let me type the code to select hundred records from the table.



I want to cache this table into memory to speed up my queries. I want to run ten queries on the same table. So it makes sense that I cache this table into the memory and avoid reading data from the disk for every query. And we have the code for the same in the snapshot below.



You must learn the following things about the Spark SQL cache table. 1. How does the Spark cache table works?

The cache-table-as command will create a temporary view using the Select expression. So you already have a table. But when you cache it, you will create a temporary view on the table. Once your table is cached as a temporary view, you can use the view or the table. Once cached, Spark will always bring data from the cache.

2. How to make it re-executable? The cache-table-as command creates a temporary view, so you cannot rerun it. And if you try re-running the cache command again, you will see a

“temporary view already exists” error. You can fix this issue by dropping the temporary view before rerunning the cache-table-as command.

3. What is Lazy? The lazy is to make sure we do not cache the table until it is used. It is optional. You can also remove it, but it is recommended to keep it as the execution will be very fast because it simply marks the table to be cached on its first use.

Caching the table consumes a lot of memory. So you should make use of this feature carefully. We have some guidelines:

1. Do not cache tiny tables because tiny table will anyway work

   faster.

2. Do not cache huge tables that cannot fit in the Spark memory

   because there is no point in caching a table into memory if it cannot fit in the memory.

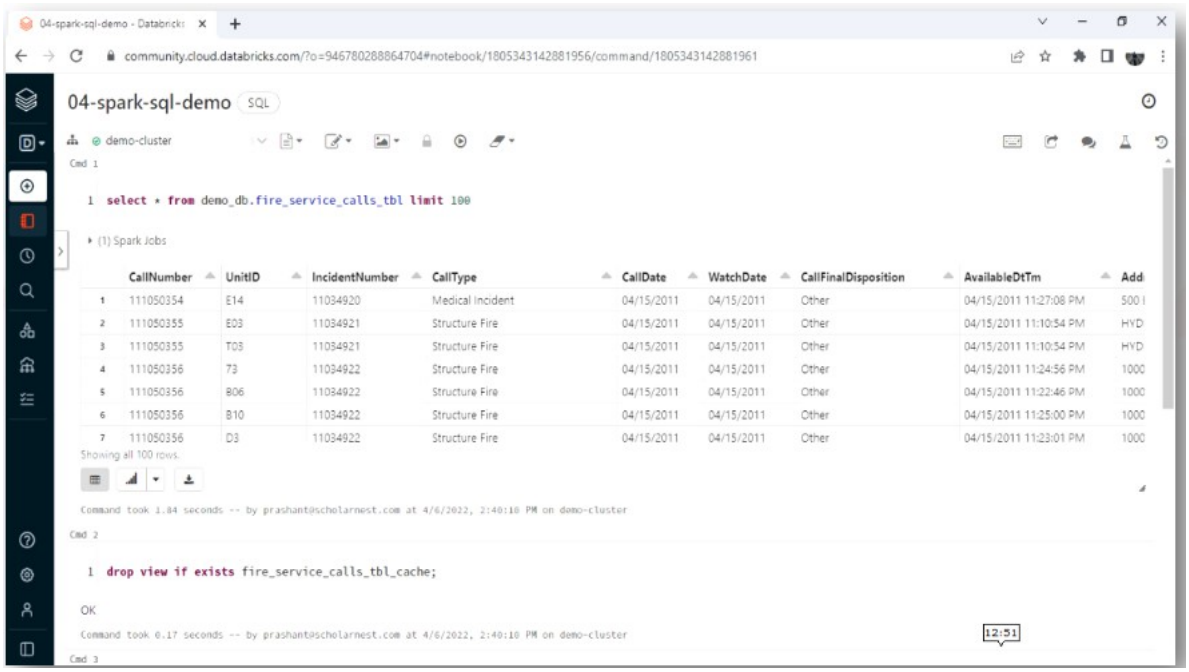
It is recommended to cache medium-sized frequently used tables. So, you should cache a table if you meet the following criteria:

1. Table data is a reasonable size and fits into the memory. 2. Table is frequently used. You should not cache a table

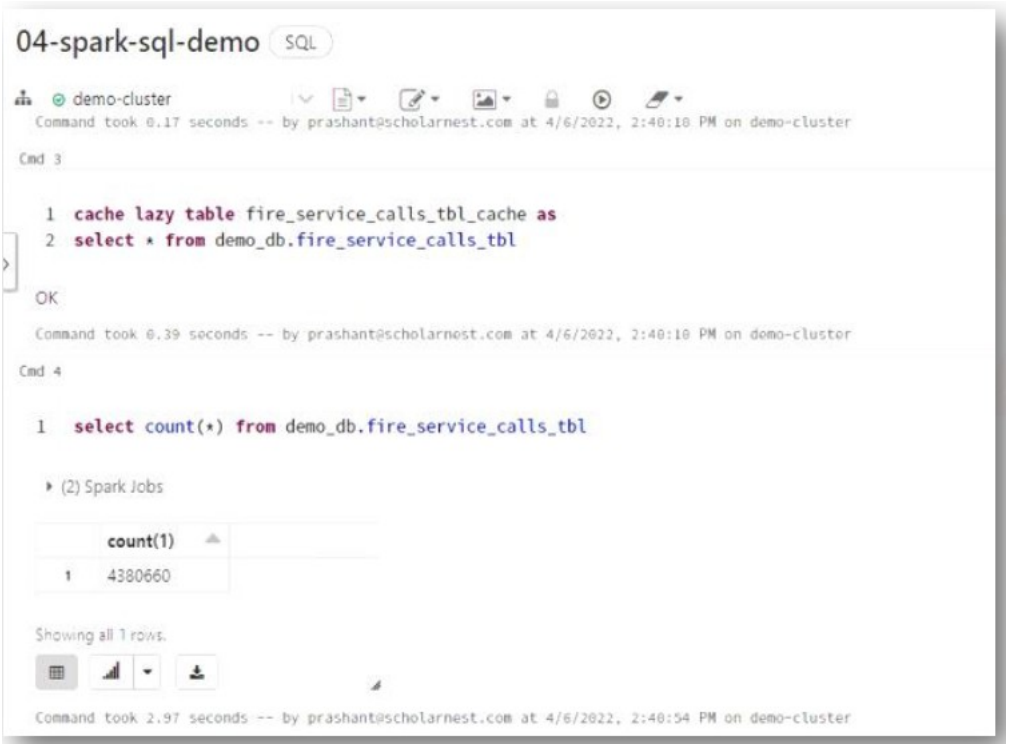
that you do not want to use again and again.

Working with Spark SQL

Go back to your notebook you created in the previous lecture. **(Reference: 04- spark-sql-demo.sql)** We created a table and also cached it into the memory. Now we are ready to start writing some SQL expressions to answer the 10 micro project questions.



Let us start by taking a simple count from the table. We can use the select count(*) query. If we run it, we can see there are more than 4.3 million records in the table.



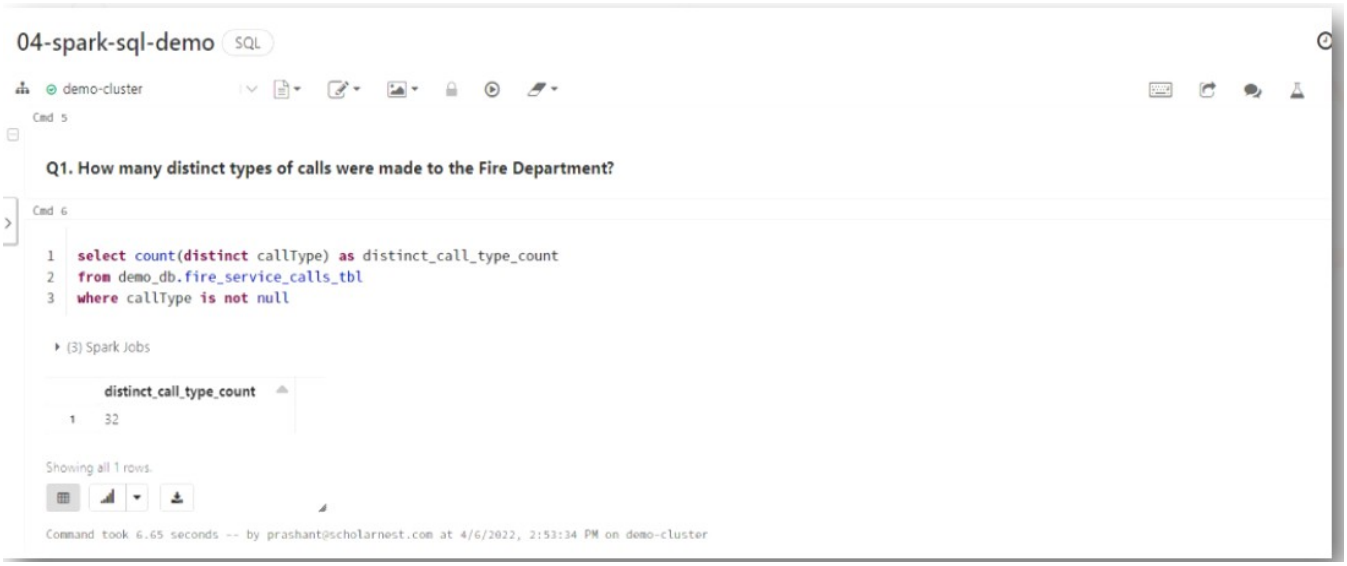
Here are ten questions we want to answer:

1. How many distinct types of calls were made to the fire department?
2. What are distinct types of calls made to the fire department?
3. Find out all responses or delayed times greater than 5 mins?
4. What were the most common call types?
5. What zip codes accounted for the most common calls?
6. What San Francisco neighbourhoods are in the zip codes 94102 and 94103
7. What was the sum of all calls, average, min, and max of the call response times?
8. How many distinct years of data are in the CSV file?
9. What week of the year in 2018 had the most fire calls?
10. What neighbourhoods in San Francisco had the worst response time in 2018?

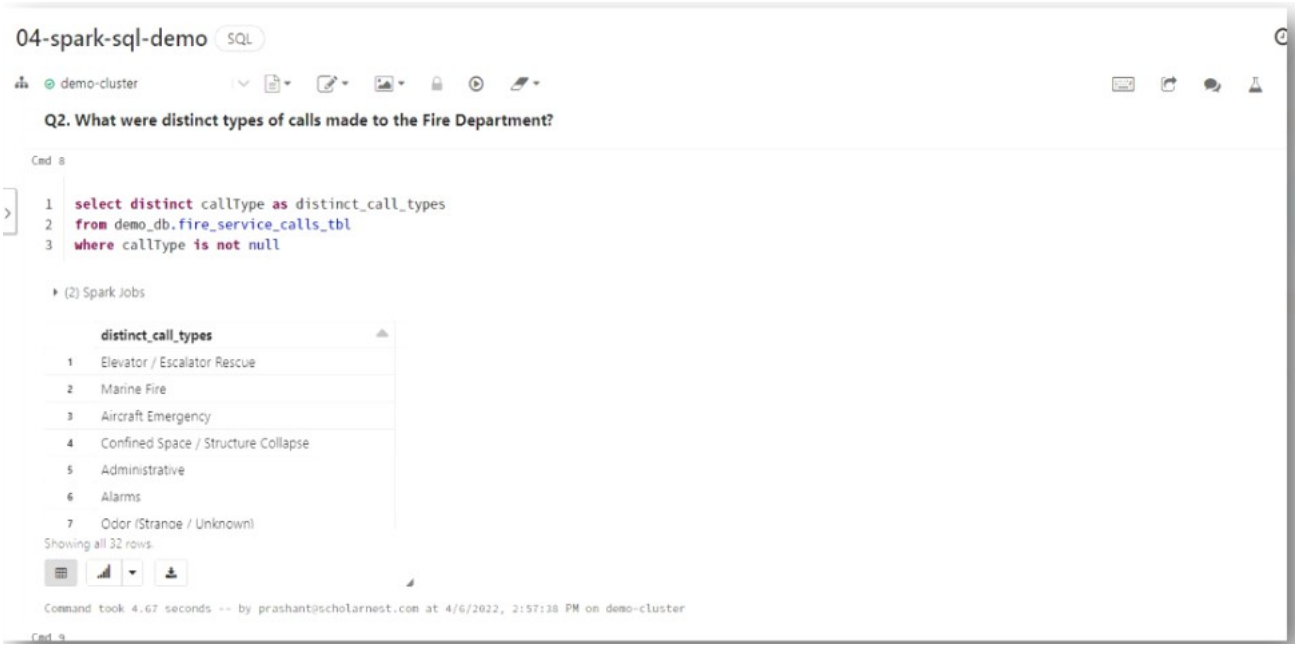
Databricks Notebook allows us to document things using markdown language. You can use the %md command to create a documentation cell, as shown in the image given below.

Let us look at the first question. They are asking for how many distinct types of calls were

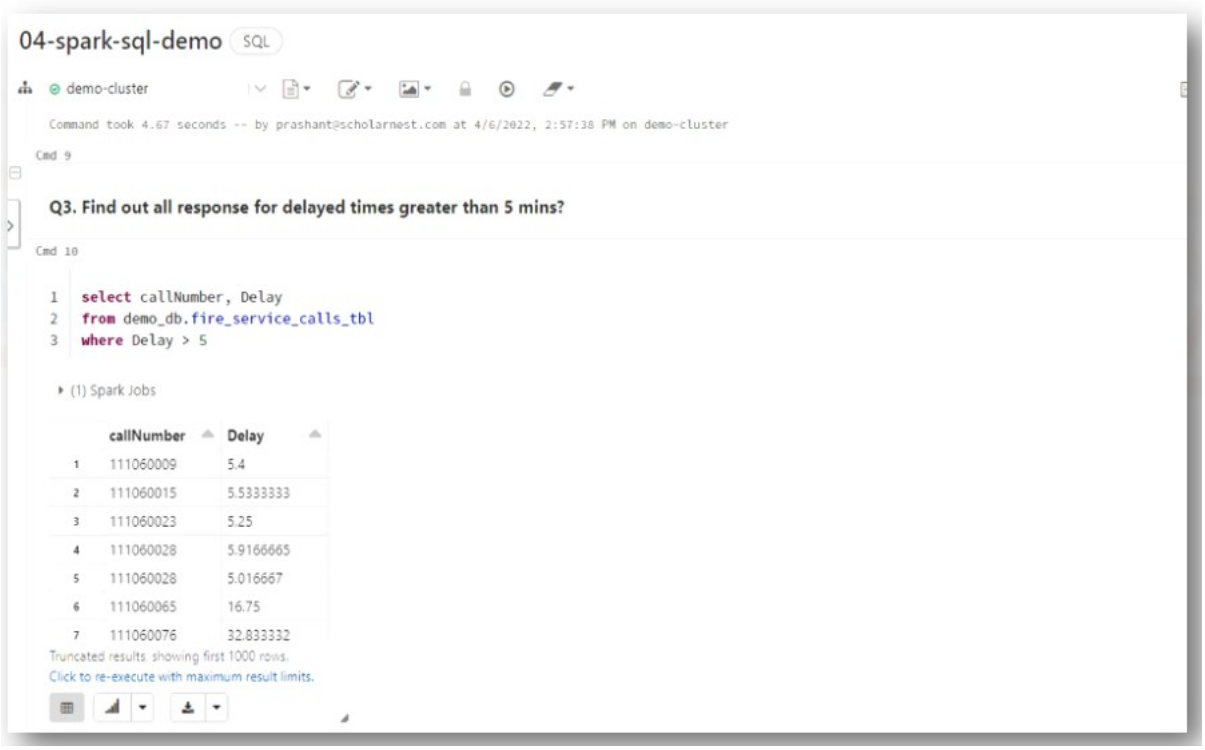
made. So the solution is to take count of distinct call types. And we have the SQL code which says *select distinct call type* and then take a *count*. I added one extra condition to eliminate null values in the cell type. And we got the expected output below the cell. We have 32 unique call types made to the San Francisco Fire station.



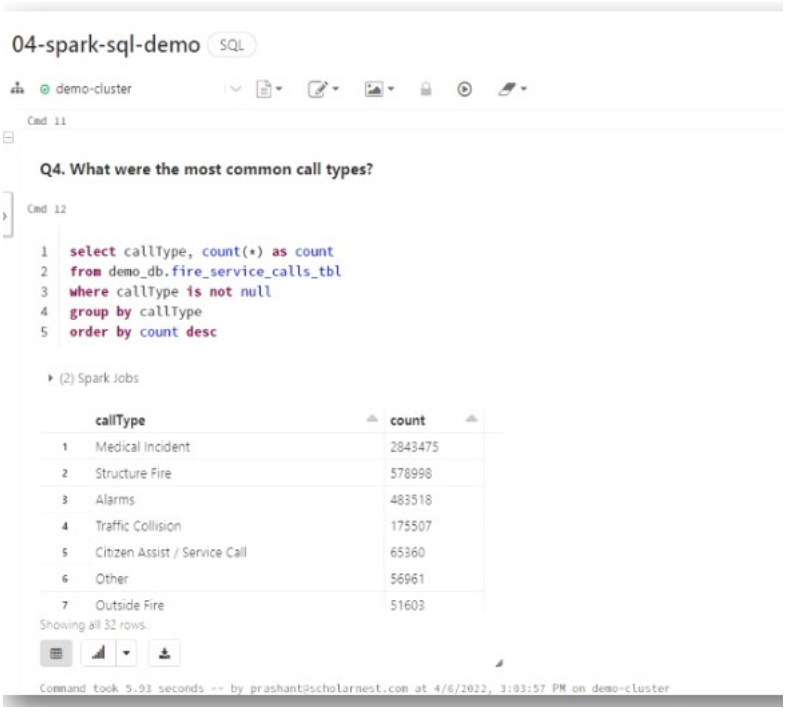
Let us look at the second question. So the question is almost the same as the first one, but this time, we need a list of distinct call types. Earlier, we wanted to count distinct call types, but now we want the list. We have the code for the same shown below. I removed the count function from the select expression we used in the first question, and we can see the expected results.



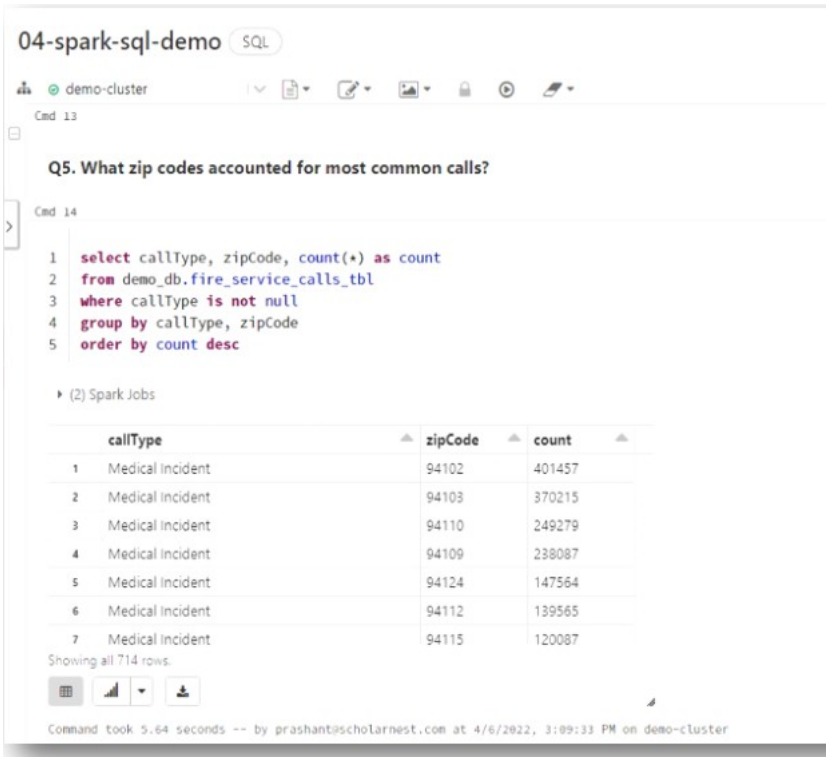
Let us look at the third question. We want to list the call numbers where the delay is greater than 5 minutes. We have the code and desired results shown below.



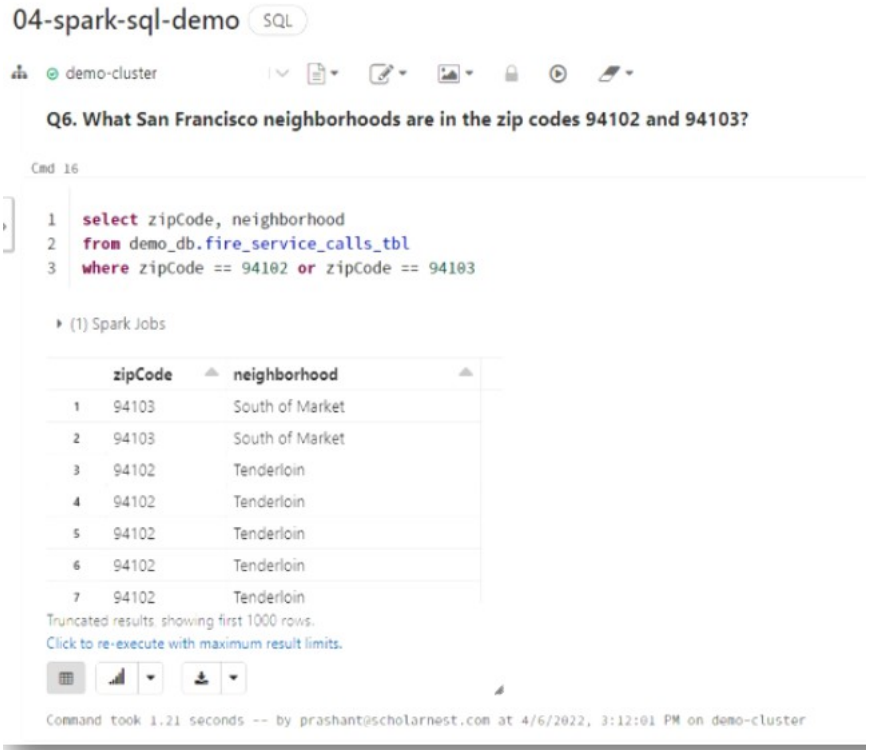
Let us look at the fourth question. We have to find the most common call types. So, we need to take a count on the call type and sort the result in descending order, and we will get the most common calls at the top. I have written the code for the same and you can see the most common types in sorted order.



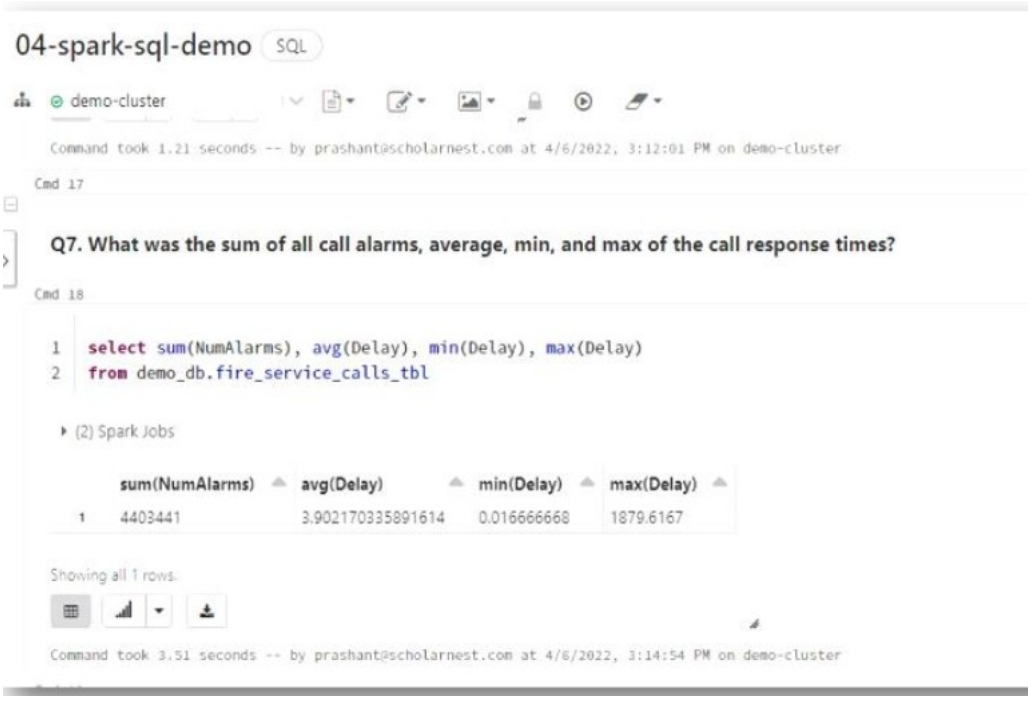
Let us look at the fifth question. Here we want to find the most common call types, but we also want to see the zip code for the most common call types. I can do that by adding a zip code column in my group by clause.



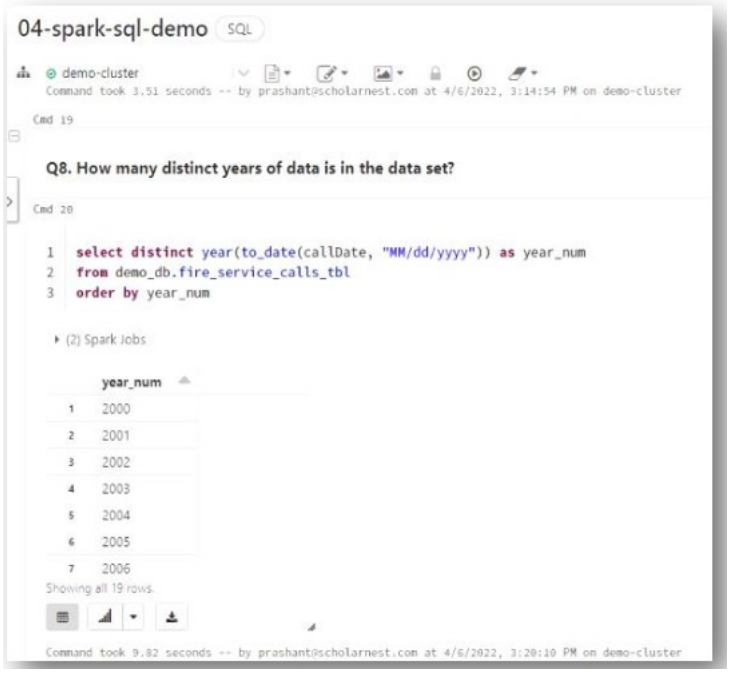
Let us look at the sixth question. This question wants us to filter the records for two zip codes - 94102 and 94103. Then we want to see the names of unique neighbourhoods in these zip codes.



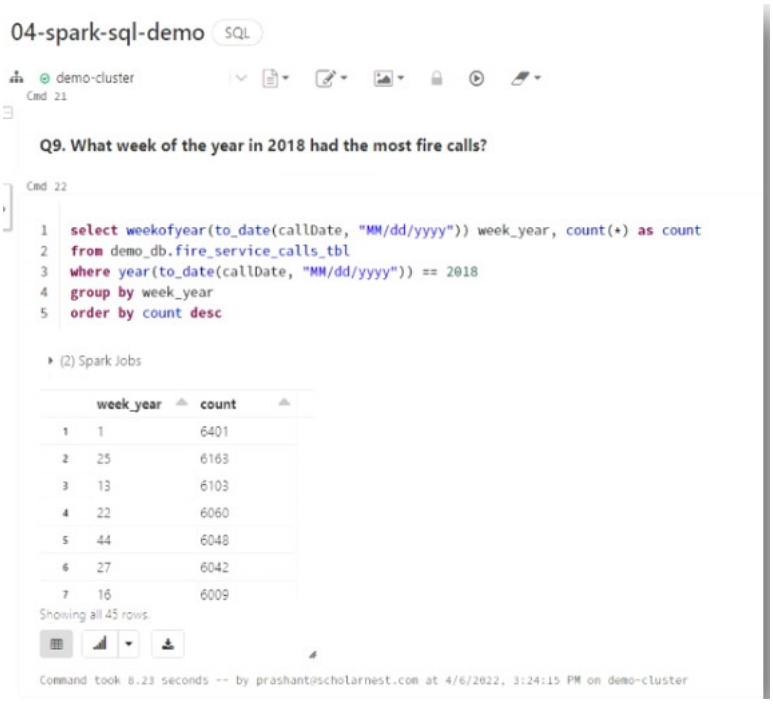
Let us look at the seventh question. The first part of the question wants us to find the sum of *NumAlarms*. The second part of the question wants us to find avg, min, and max response time delay. And we want to find it for all the calls. So no filter and no grouping.



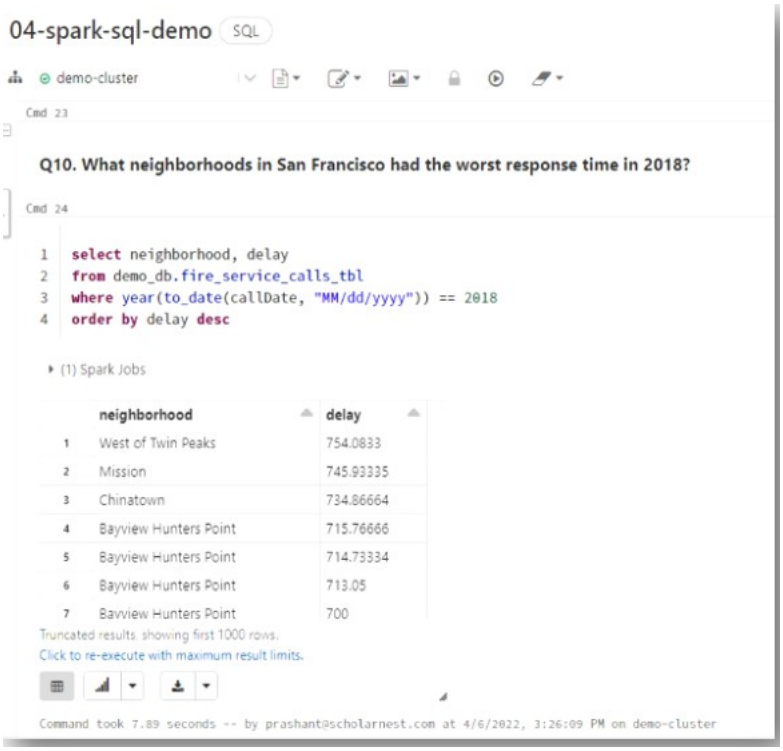
Let us look at the eighth question. They want us to find the number of distinct years in the data set. We have a Call Date field, and we can take out the year from that field. However, the Call Date is a string field. But we can convert it to a date field and take out the year. Once we have the year number, we can apply a distinct clause to find unique years.



Let us look at the ninth question. They want us to filter the table for the year 2018. We can convert the CallDate into a date filed, take out the year and filter it for 2018. Then we can take out the week number from the CallDate and count it by week number. Once counted, you can sort it in descending order to find the most calls.

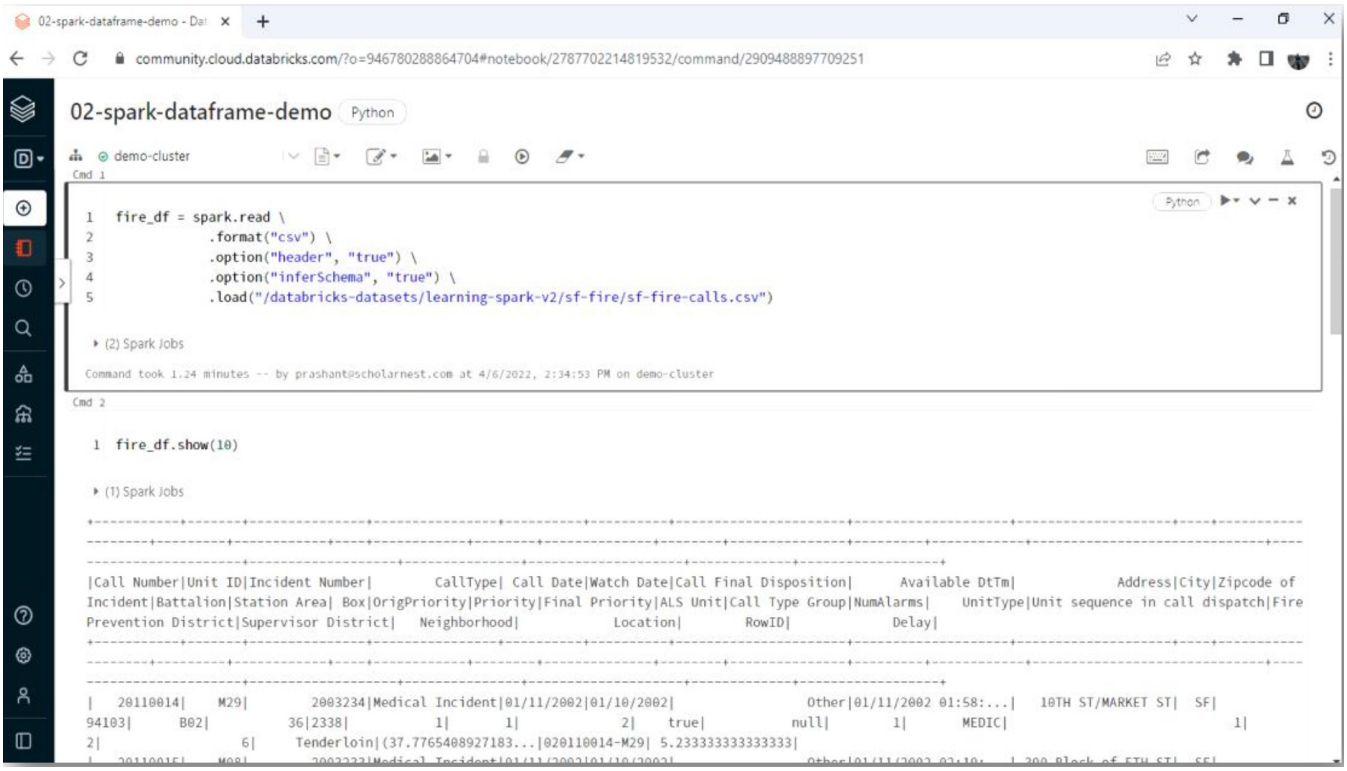


Let us look at the tenth question. We just need to filter records for the year 2018. Then look for the neighbourhoods and delay columns. And finally, sort it by delay in descending order.

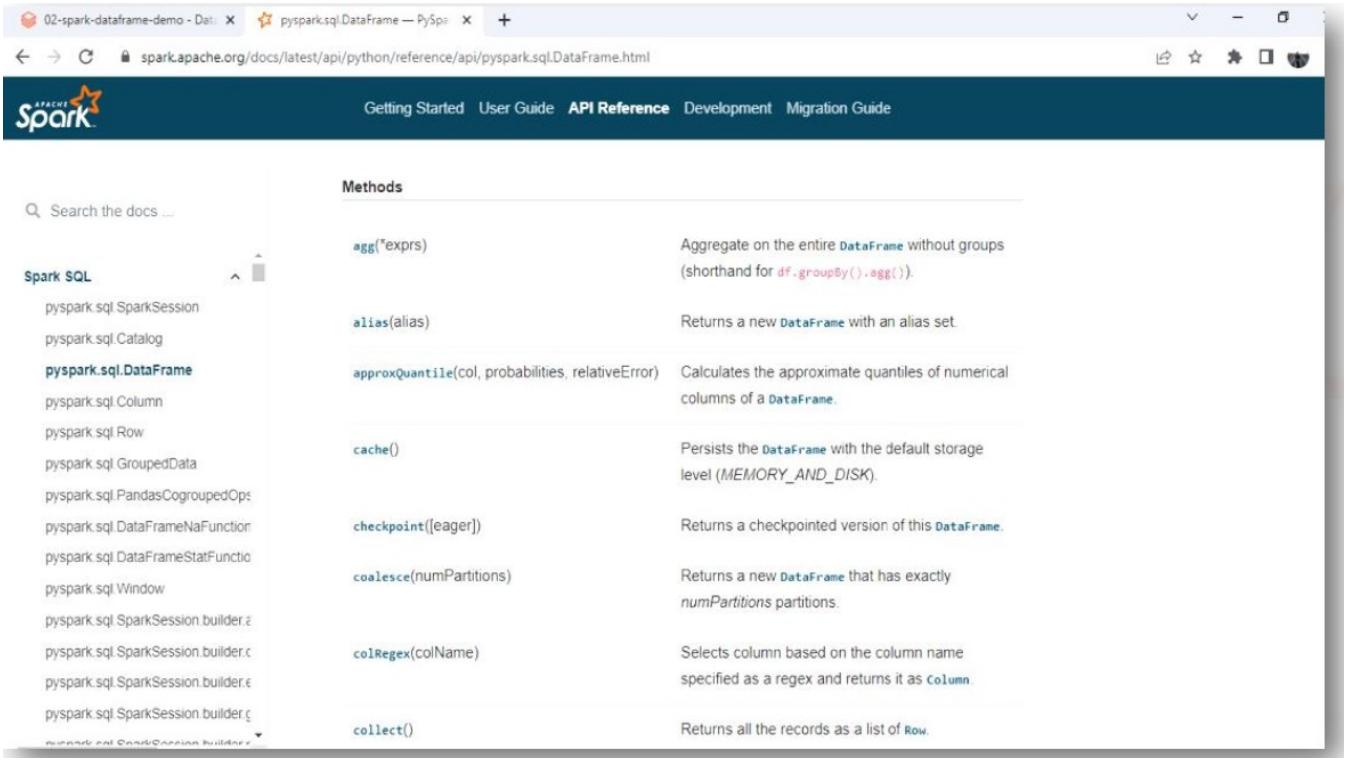


Dataframe Methods

Go to your Databricks workspace and open spark-dataframe-demo notebook. **(Reference: 02-spark-dataframe-demo.ipynb)**



Now that we have created a Dataframe, we can apply Dataframe methods to process the data. You can visit https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.sql.DataFrame.html to get the list of all the Dataframe methods.



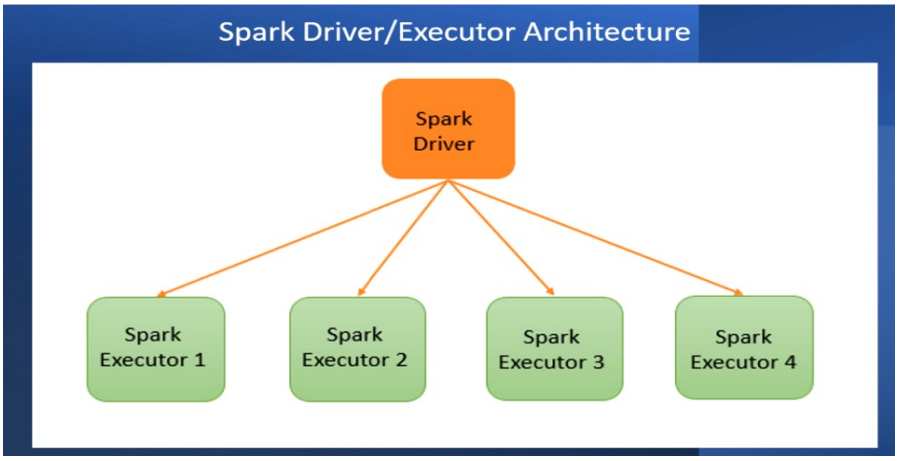
Dataframe offers you a long list of methods. However, we can classify them into three logical groups:

1. Actions 2. Transformations 3. Functions/Methods

Actions: Actions trigger a Spark Job and return to the Spark driver. Spark is a distributed

processing system. So every Spark program runs as one driver and one or more executors And Actions are Dataframe operations that performs two things:

1. Kick off a Spark Job
2. Return to the Driver



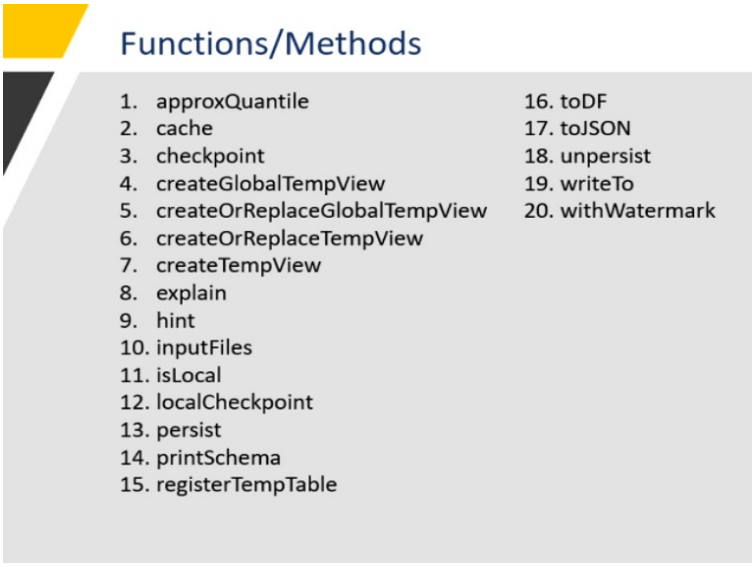
Transformations: Spark Dataframe transformation produces a newly transformed Dataframe. Transformations are different than actions. They do not kick off a Spark Job, and they do not return to the driver.

They simply create a new Dataframe and return it.

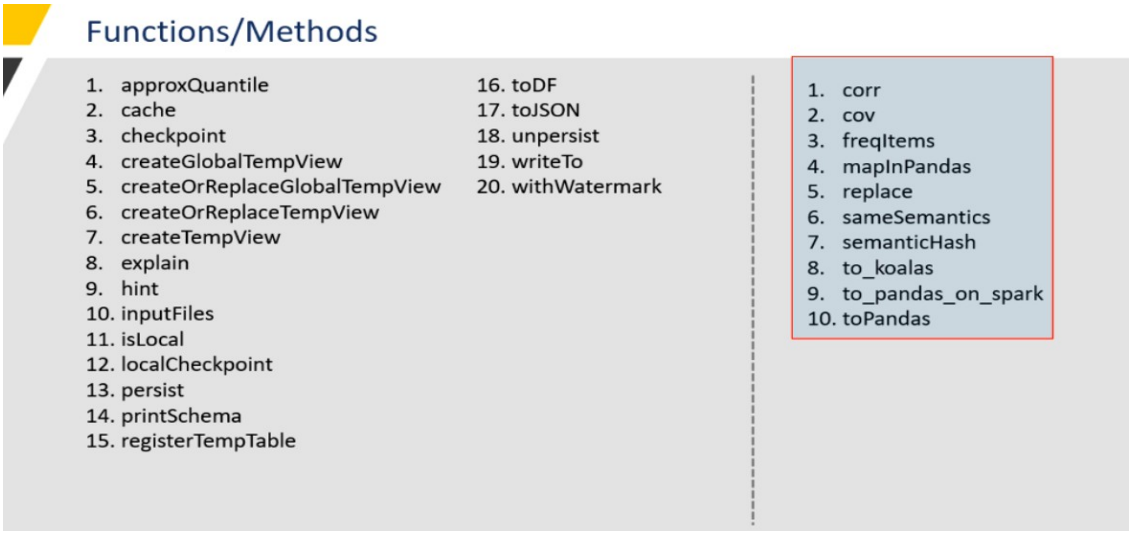
Functions and Methods : These are DataFrame methods or functions which are not categorized into Actions or Transformations. So everything other than the Actions and Transformations is a method or function.

Here is a list of Dataframe Actions and Dataframe Transformations. Spark offers you 12 actions as on date. This list may update with a newer version of Spark. We have 38 transformations.

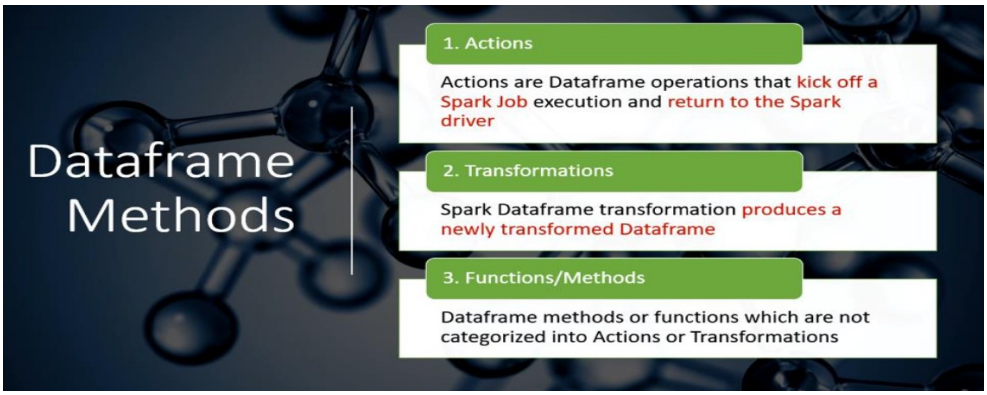
Here is a list of Dataframe Functions and Methods. We have 20 functions and methods. These are utility functions or methods.



There are ten more functions that I excluded from the list that I have highlighted in the image below. I have excluded these because those functions are only available in Python and not in other languages, and it is not frequently used.

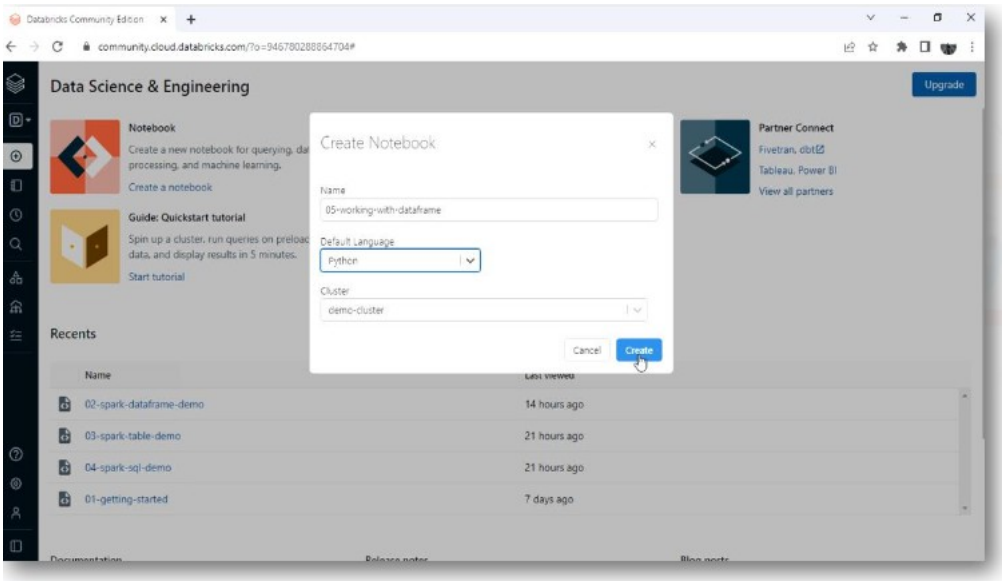


Finally, here is the definition of all the three types of Dataframe Methods.

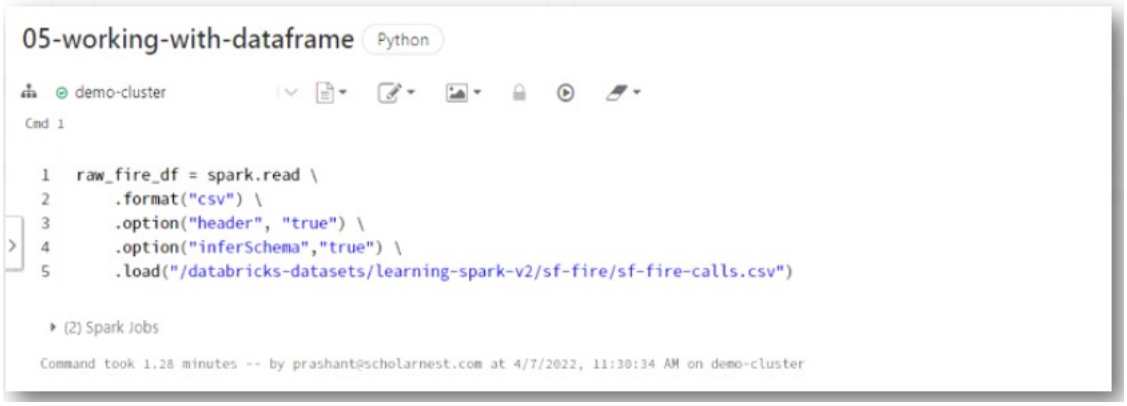


Applying Dataframe Transformations

Go to your Databricks workspace and create a new notebook as given below. **(Reference: 05-working-with-dataframe.ipynb)**

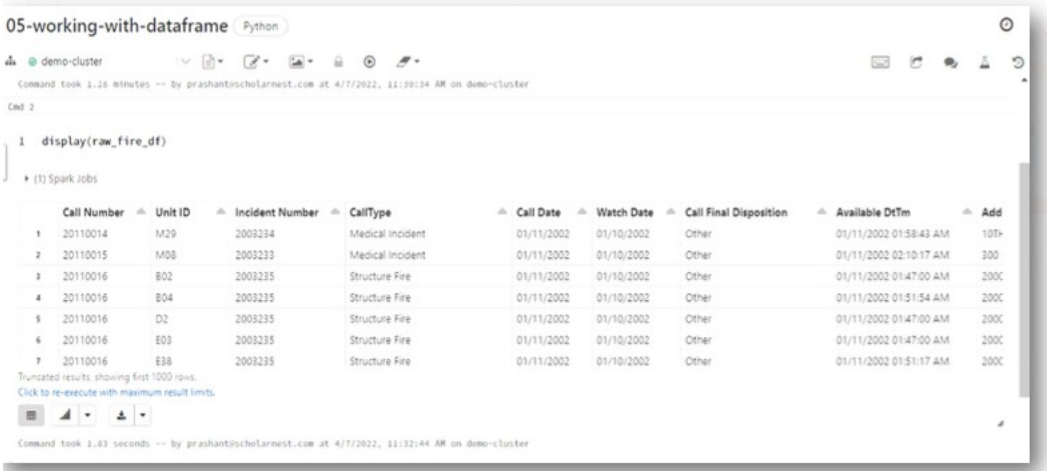


Here we the older code which we used earlier to create a data frame for the fire call response data set.



If we display the older raw_fire_df Dataframe and see the results as given below. We can see there are two problems in this Dataframe:

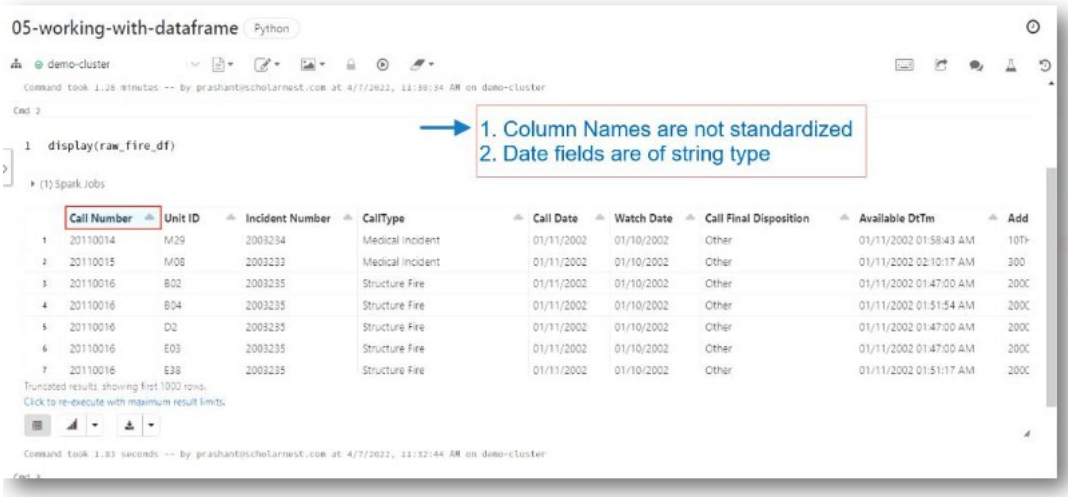
1. ColumnNames are nots tandardized 2. Date fields are of string type



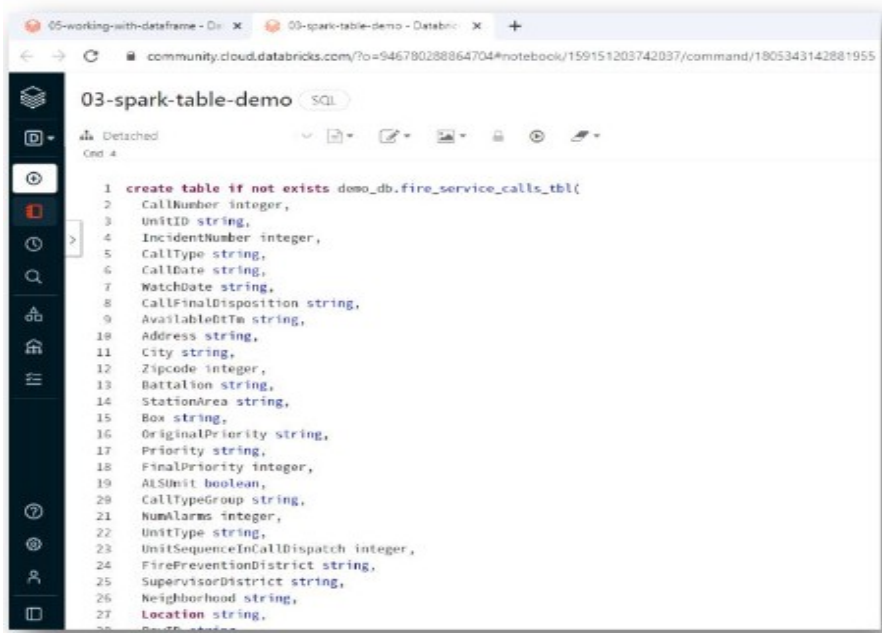
Column Names are not standardized:

You can see a blank space between call and number. That's not standard. Spark Dataframe column names are case insensitive. However, space in a column name is not a common thing.

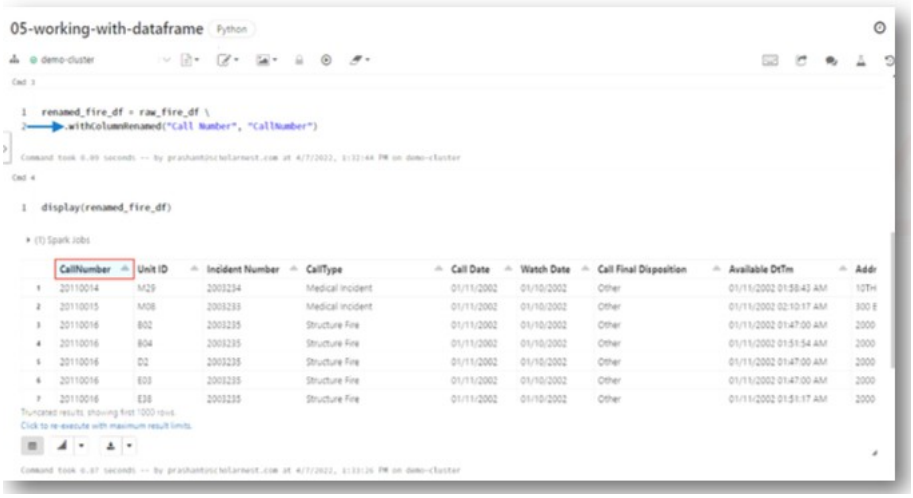
Let me fix these column names and rename them to remove spaces.



Column Names are not standardized: We fixed this problem while creating the table. **(Reference: 03-spark-table-demo.sql)** You can see in the image below that we created column names without spaces when we created the table. Then we loaded data into this table. So we already fixed the column names while creating a table.

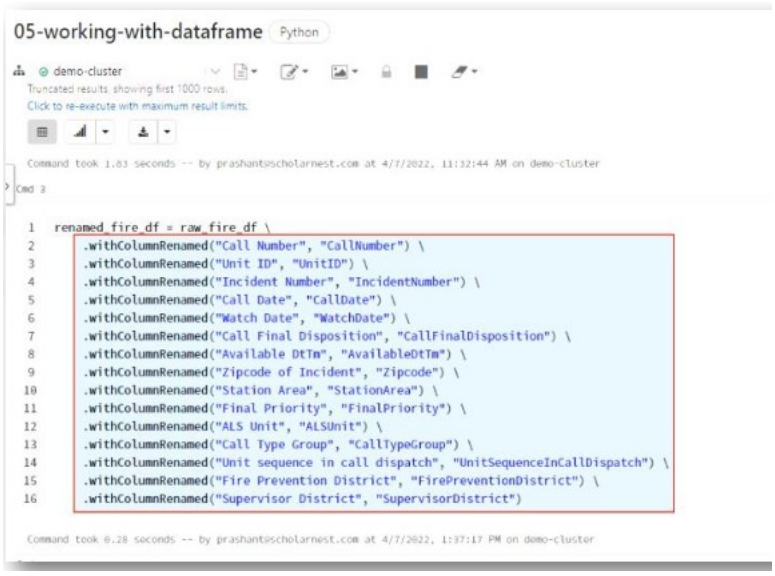


Column Names are not standardized: Here is how we can fix the column names while working on Dataframes. **(Reference: 05-working-with- dataframe.ipynb)** Spark Dataframe offers you withColumnRenamed() transformation to rename a column. You can see in the screenshot below that withColumnRenamed() takes two arguments: Old column name and New column name. And we can see the desired results when we are displaying the Dataframe below.



Column Names are not standardized:

Similarly, we can rename all the problem columns as shown in the code below. If we run this code and display the results, our first problem is resolved and our column names and standardized.



Spark Dataframe programming is all about doing two things:

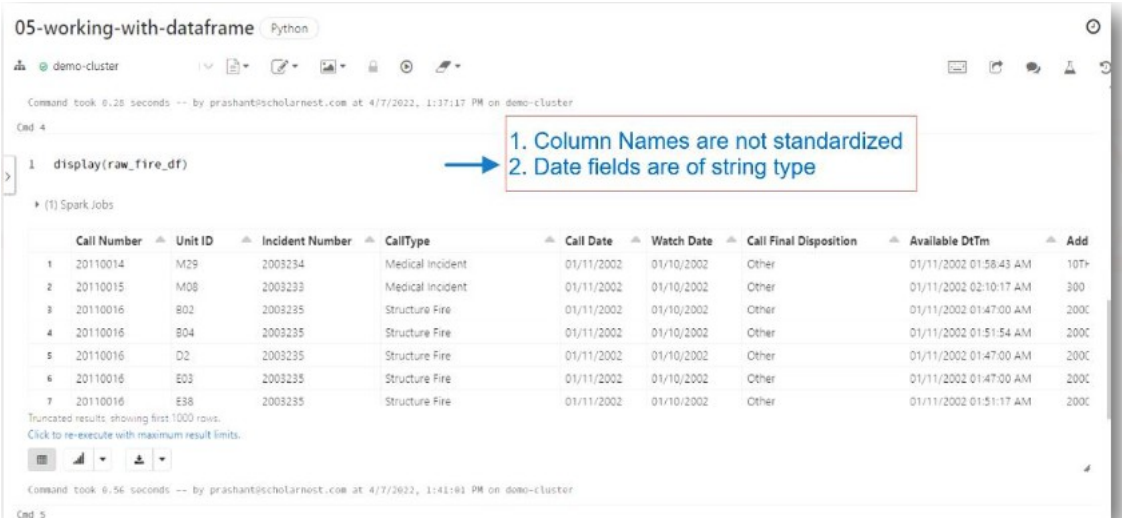
1. Read your raw data into a Dataframe. 2. Transform your raw Dataframe and create a new transformed Dataframe.

We can also narrow down 3 key observations from what we have learned till now:

1. You can create a chain of Spark transformation methods one after the other.
2. Spark transformation returns a new Dataframe after transforming the old Dataframe.
3. Spark Dataframe is immutable. We cannot change or modify an existing Dataframe. Instead, we transform the existing Dataframe and create a new Dataframe.

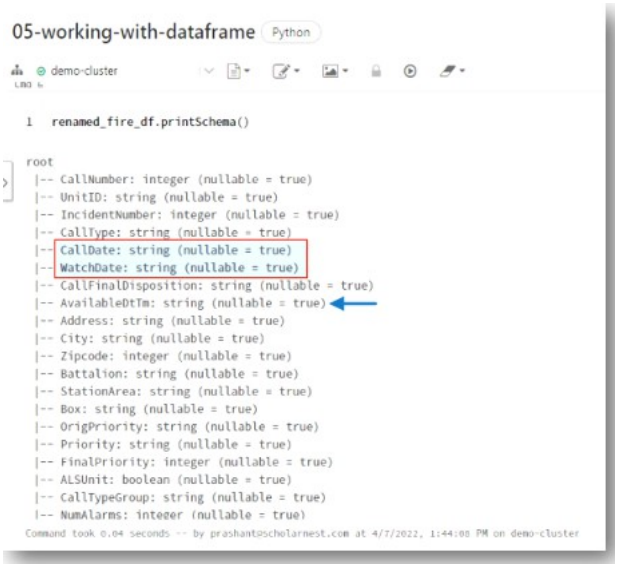
Now let us move ahead to the second problem. You can see call date, watch date, and AvailableDtTm fields in the data set. The call date and watch date represent a date. And the

AvailableDtTm represents a timestamp. However, these fields are of the type string, which is not desirable.

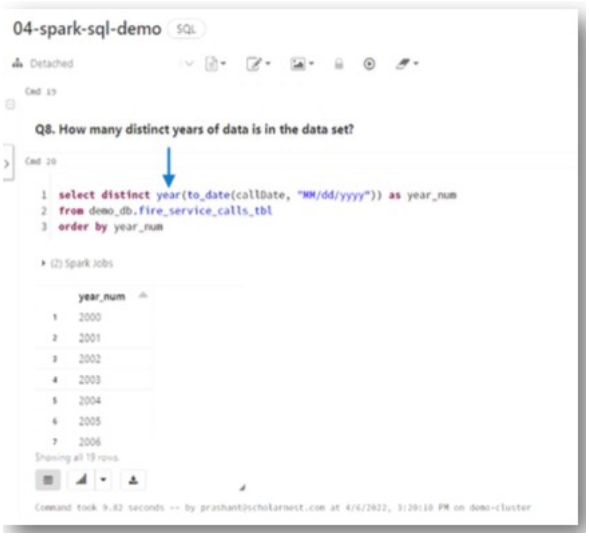


Date fields are of string type:

You can see the data type information using the printSchema() utility method. We have used this method on remaned_fire_df as we have transformed the older Dataframe into this new Dataframe by removing the problem column names. You can see that all the three fields are of the string type. Let me fix this problem.



Date fields are of string type: We had this problem while working on spark-SQL-demo notebook. **(Reference: 04-spark-sql-demo.sql)** You can see the solution of question number 8, that we converted the Call Date to a date filed. Then we applied the year() function. So, if your date and timestamp fields are stored in a string column, you may have to convert it to date on almost every use of the column.



Date fields are of string type:

You can rectify this issue by transforming the *renamed_fire_df* to a new Dataframe *fire_df* by applying the *.withColumn()* transformation. The withColumn method takes two arguments:

1. The first column is the one you want to transform. I want to transform the Call Date column, so I give the

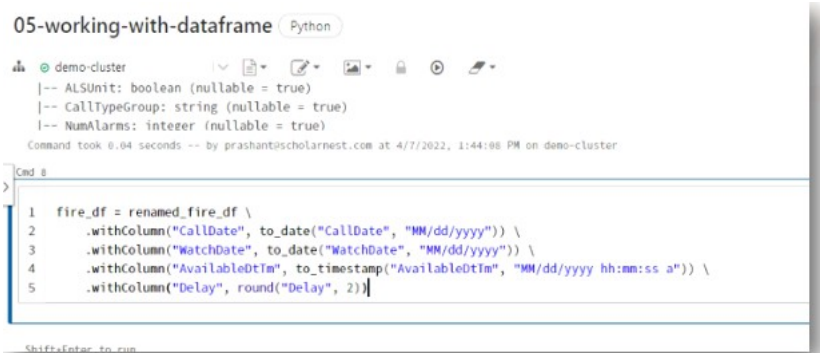
   column name here.

2. The second argument is the logic to transform the given column. We want to convert the Call Date to a

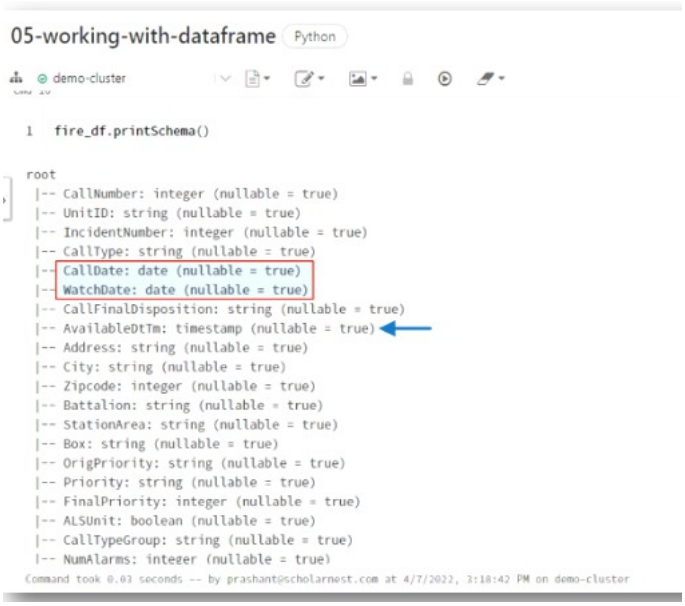
   Date. So I am using the to_date() function. And the Call Date column in the remaned_fire_df is

   represented using MM/dd/yyyy string. Similarly, we have the corresponding logic for AvailableDtTm.

Make sure you import the to_date() and to_timestamp() function from the pyspark.sql.functions package. We are also transforming the Delay column to round the Delay column for two digits.



We can print the schema of the new fire_df Dataframe and see the data types. And we see the desired results in the output, the date and timestamp transformation is successful.



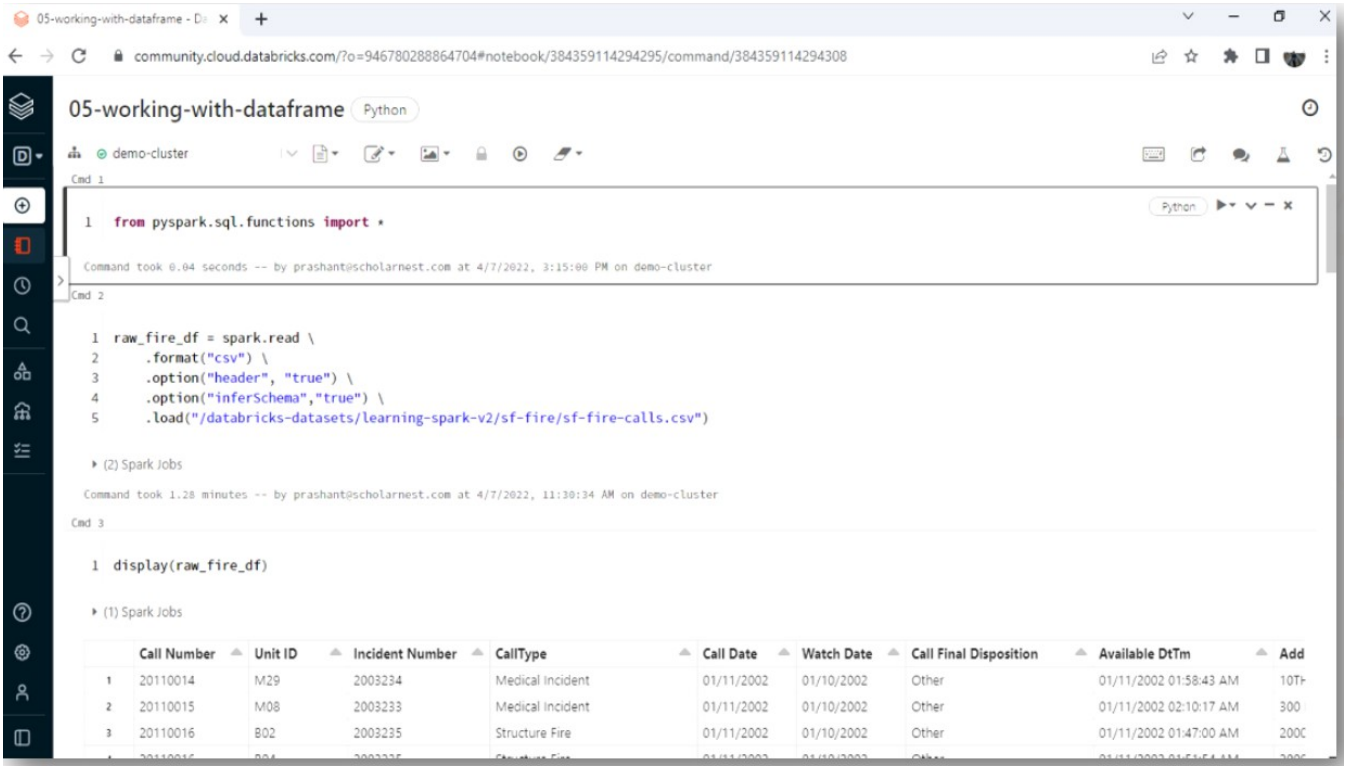
We learned about three new methods in this lecture. Column Transformation Methods:

1. withColumnRenamed 2. withColumn

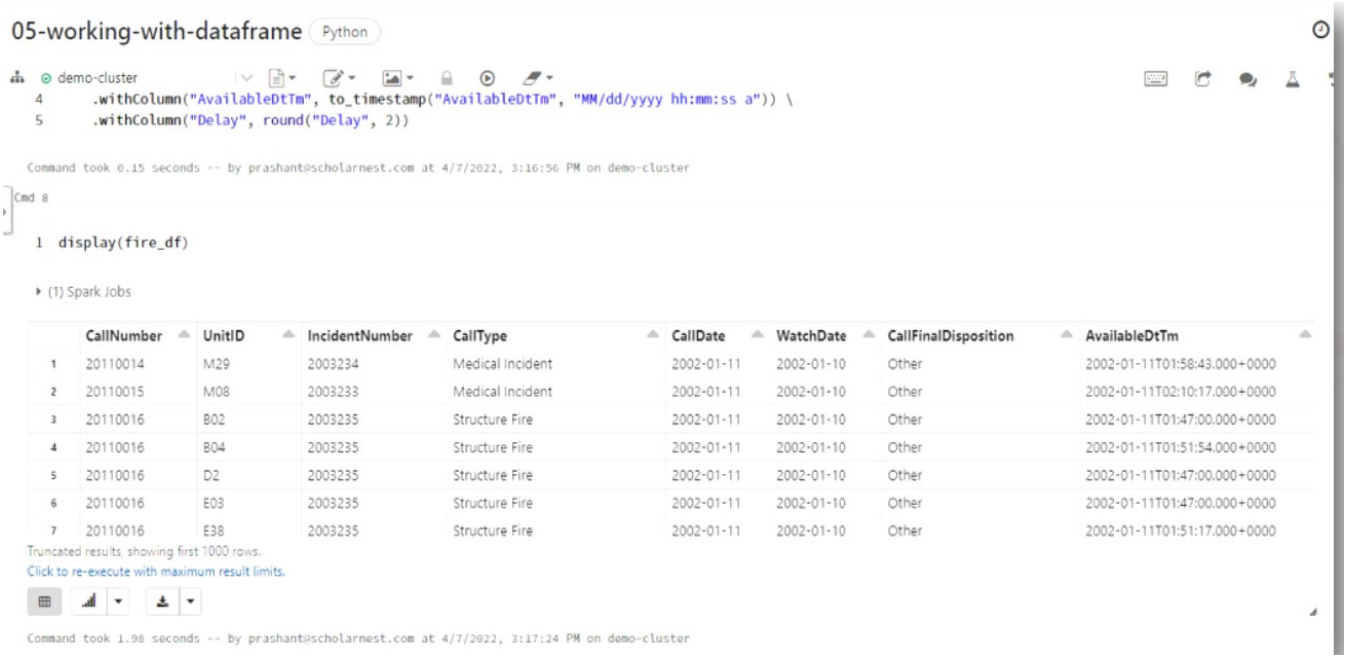
Utility Method: 1. printSchema()

Working with Spark Dataframe – Part I

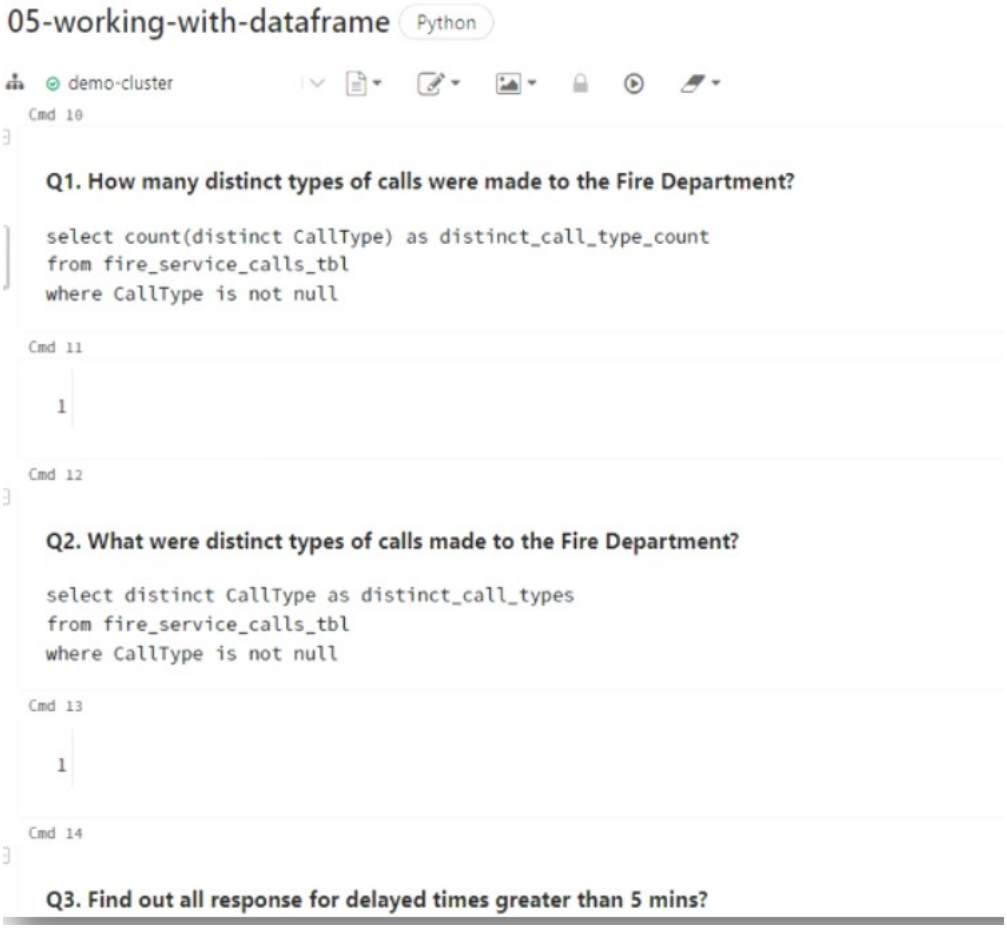
Go to your Databricks workspace and open the notebook we created in the previous lecture. **(Reference: 05-working-with-dataframe.ipynb)**



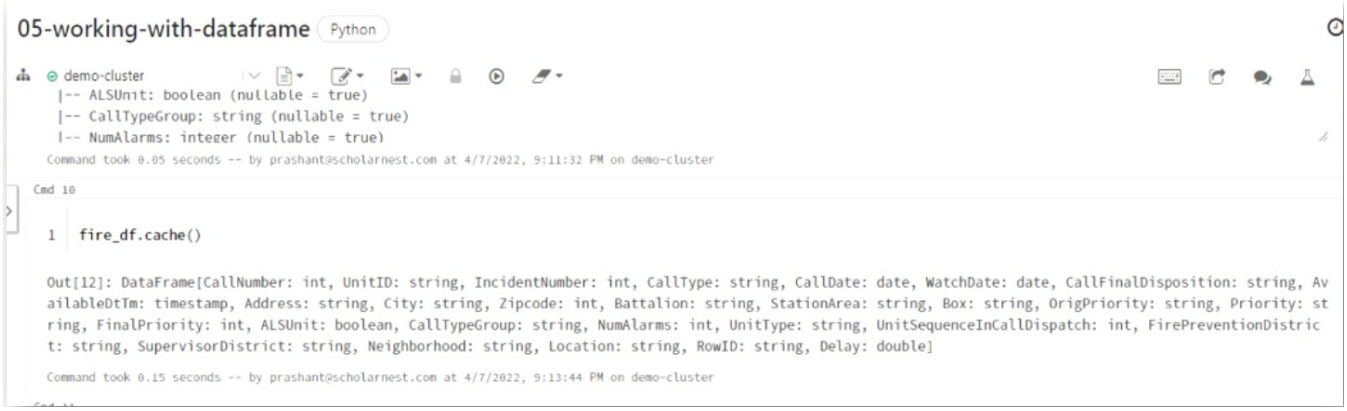
We have transformed the created a Dataframe fire_df which is ready for analysis. You can see the fire_df Dataframe here in the screenshot below.



We can document all the 10 micro project question in this notebook along with it's solution SQL expression which we have already seen in the earlier lectures.



Before I start analysis, let me cache the Dataframe. You can use the cache() utility method to cache the Dataframe into memory. Because I want to run ten analysis transformations on the same Dataframe.



We already know the logic behind the 10 question as we implemented the same using SQL. But SQL runs on a table. And we want to learn how to run those same 10 logic on a Dataframe.

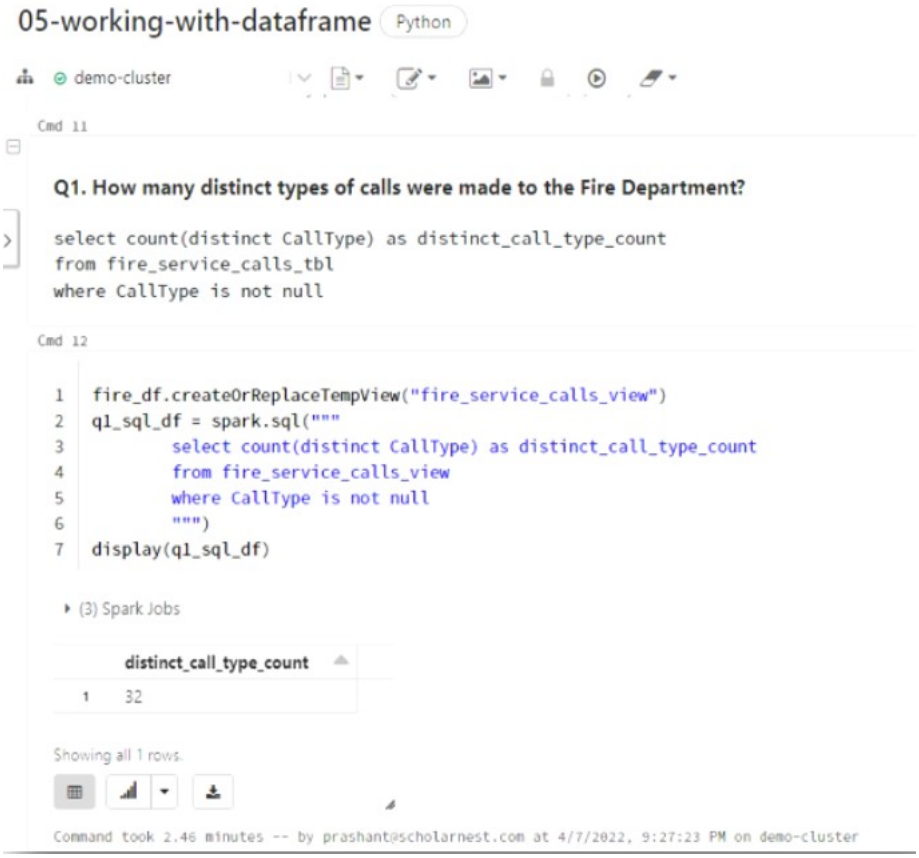
We have two approaches:

1. SQL Approach
2. Dataframe Transformation Approach

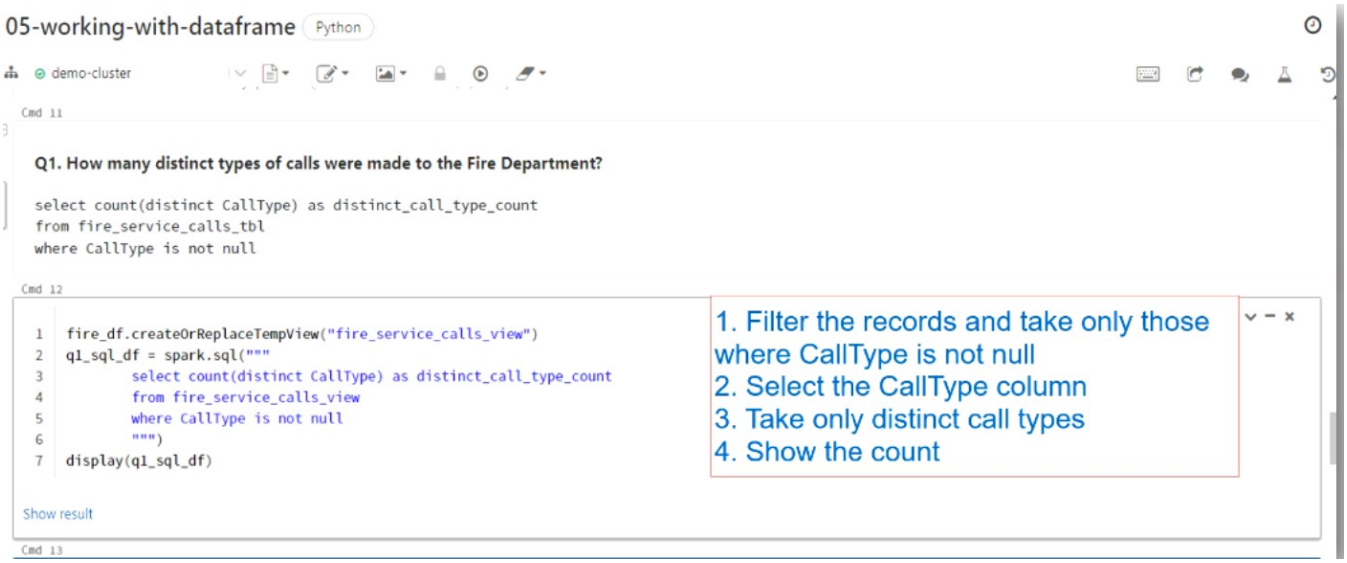
Further, the SQL approach is a two-step process:

1. Convert your Dataframe to a temporary view
2. Run your SQL on the view

Let me proceed with the SQL approach, the first thing is to convert my Dataframe to a temporary view. So I am using the same SQL that I used on the table. But I changed the table name to the view name. Now, the second step is to run SQL queries on the view. The second step will give you a Dataframe.

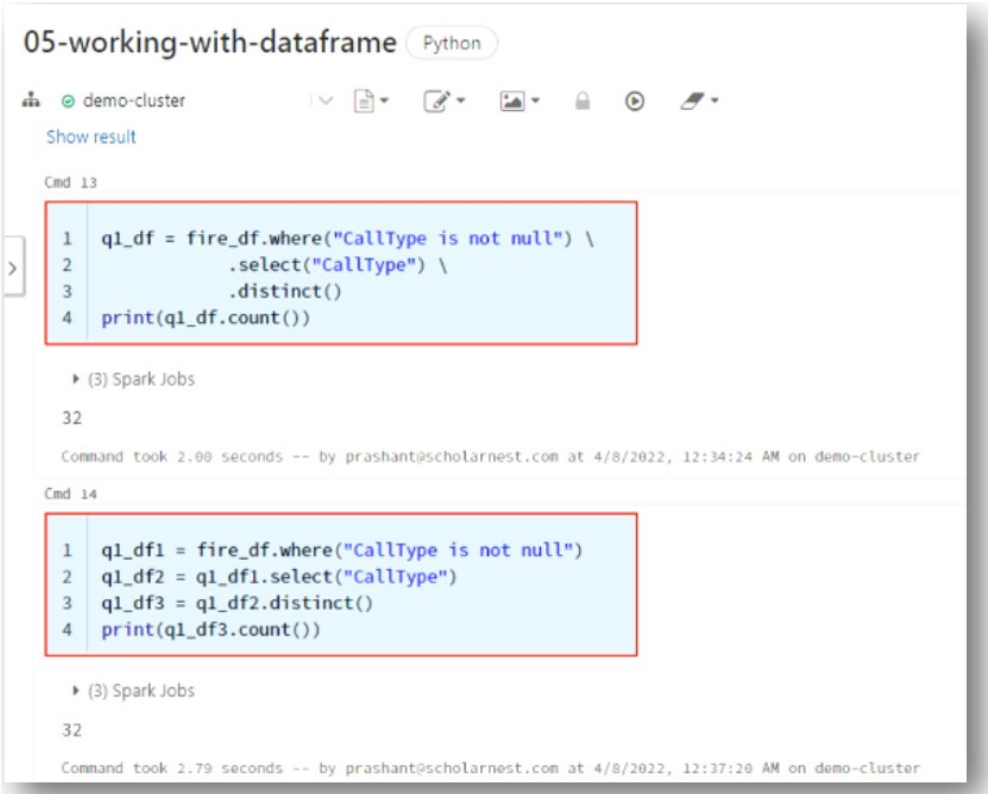


Now let us see the Dataframe Transformation approach. Here we have the logic for this approach given below. We can build the logic easily just by looking at SQL solutions.



Here is the code for the Dataframe Transformation approach. We got the same desired output with this approach as well.

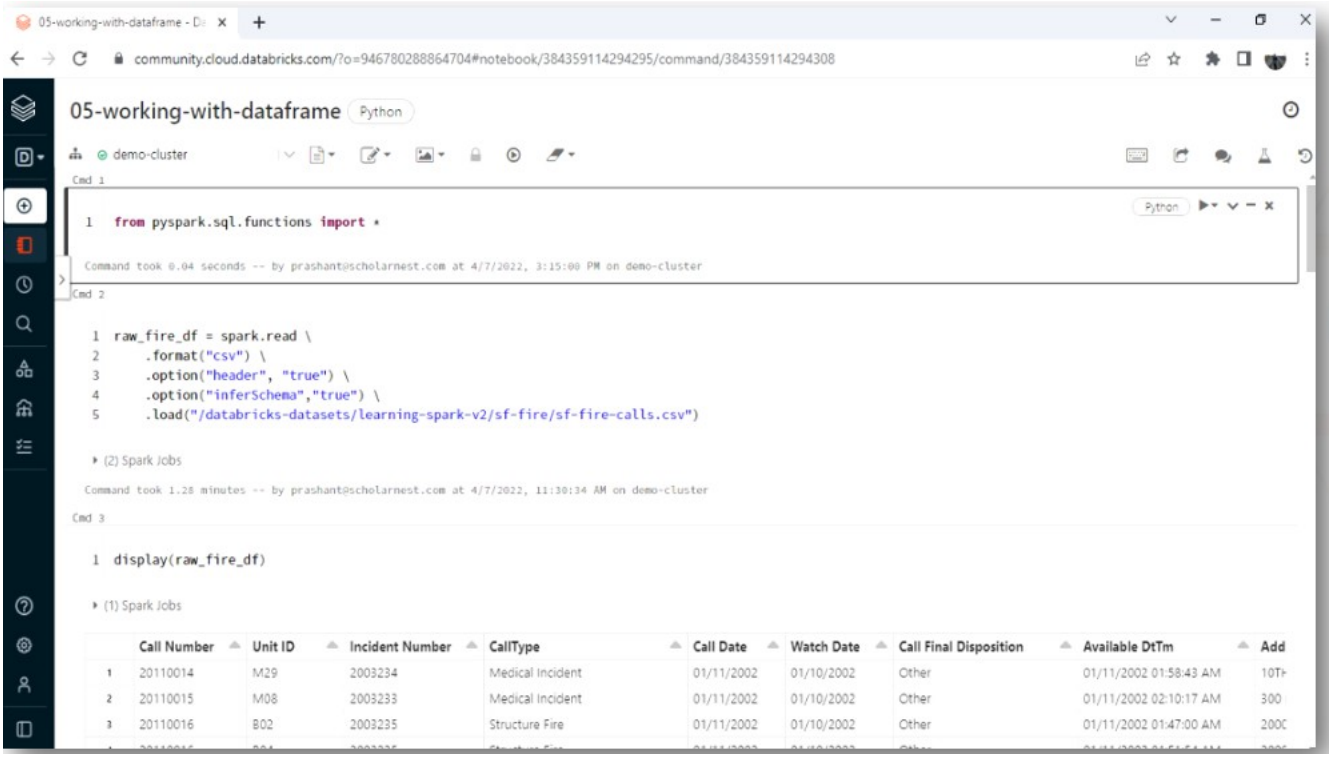
We can also write the code for the Dataframe approach in the manner shown below. Both the approach will give you the same answer, but it is recommended to follow the first approach shown in the image.



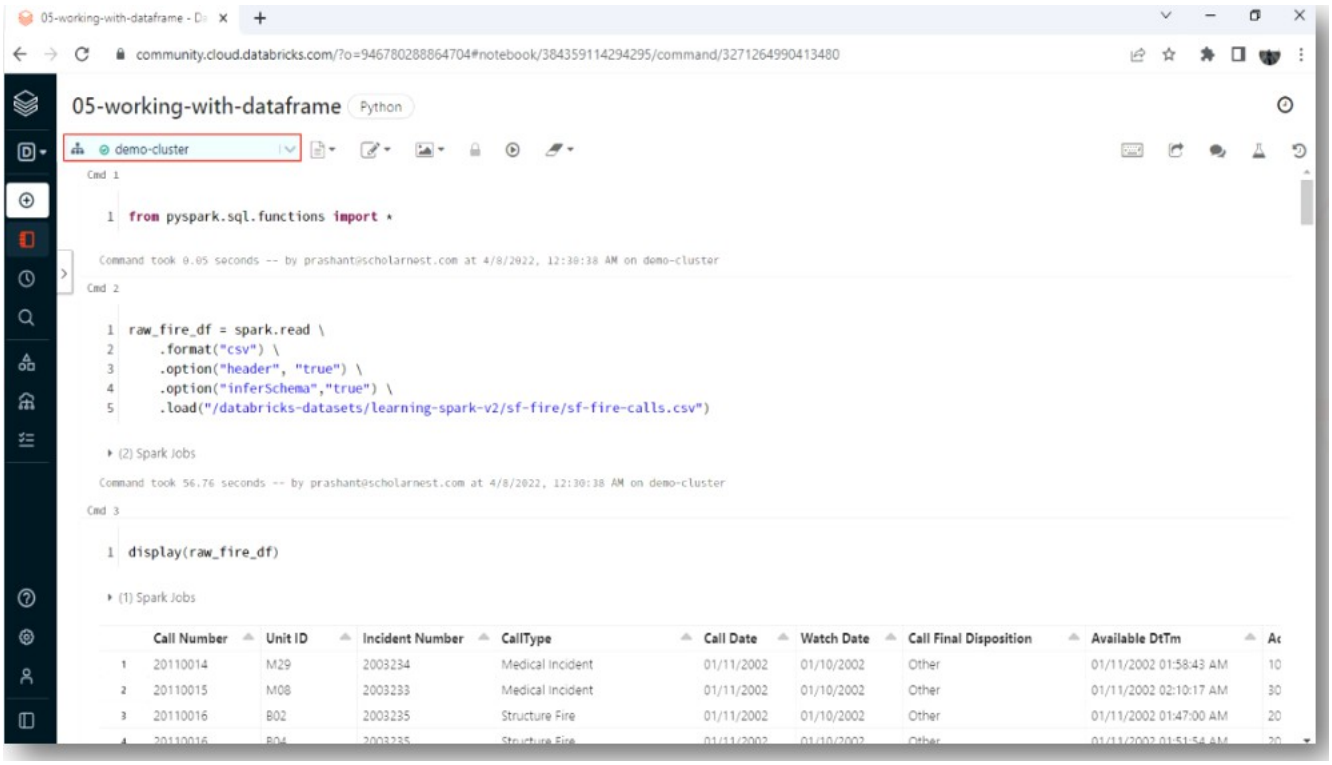
We can also write the code for the Dataframe approach in the manner shown below. Both the approach will give you the same answer, but it is recommended to follow the first approach shown in the image.

Working with Spark Dataframe – Part II

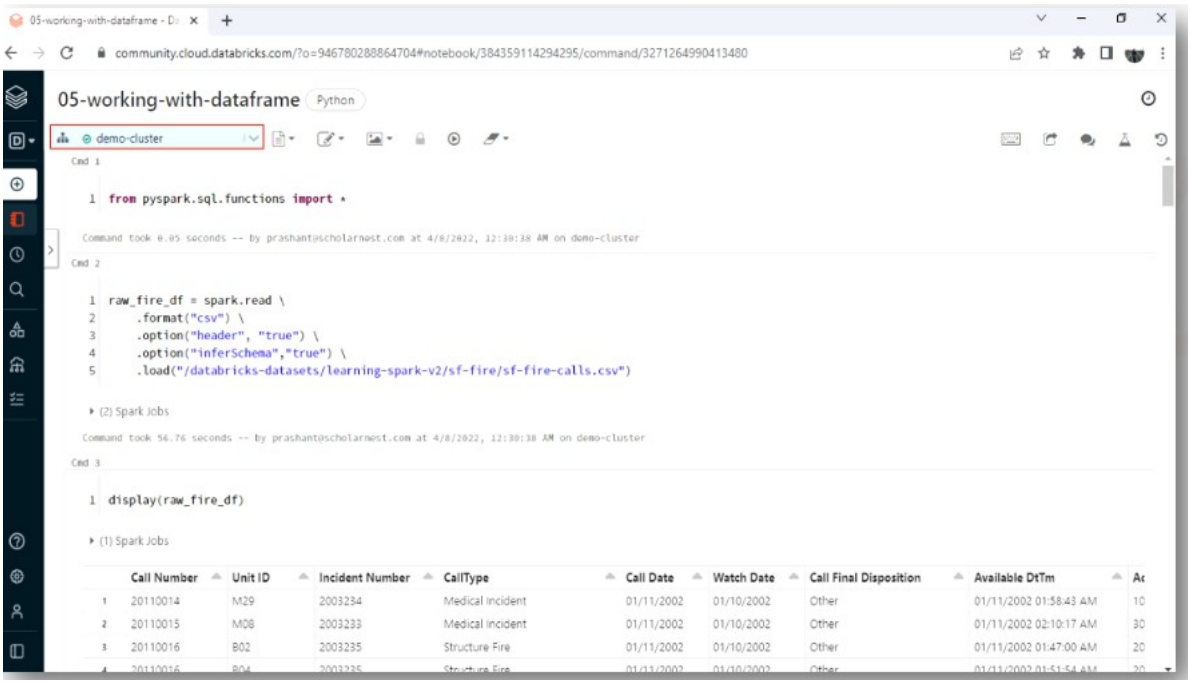
Go to your Databricks workspace and open the notebook we worked with in the previous lecture. **(Reference: 05-working-with-dataframe.ipynb)**



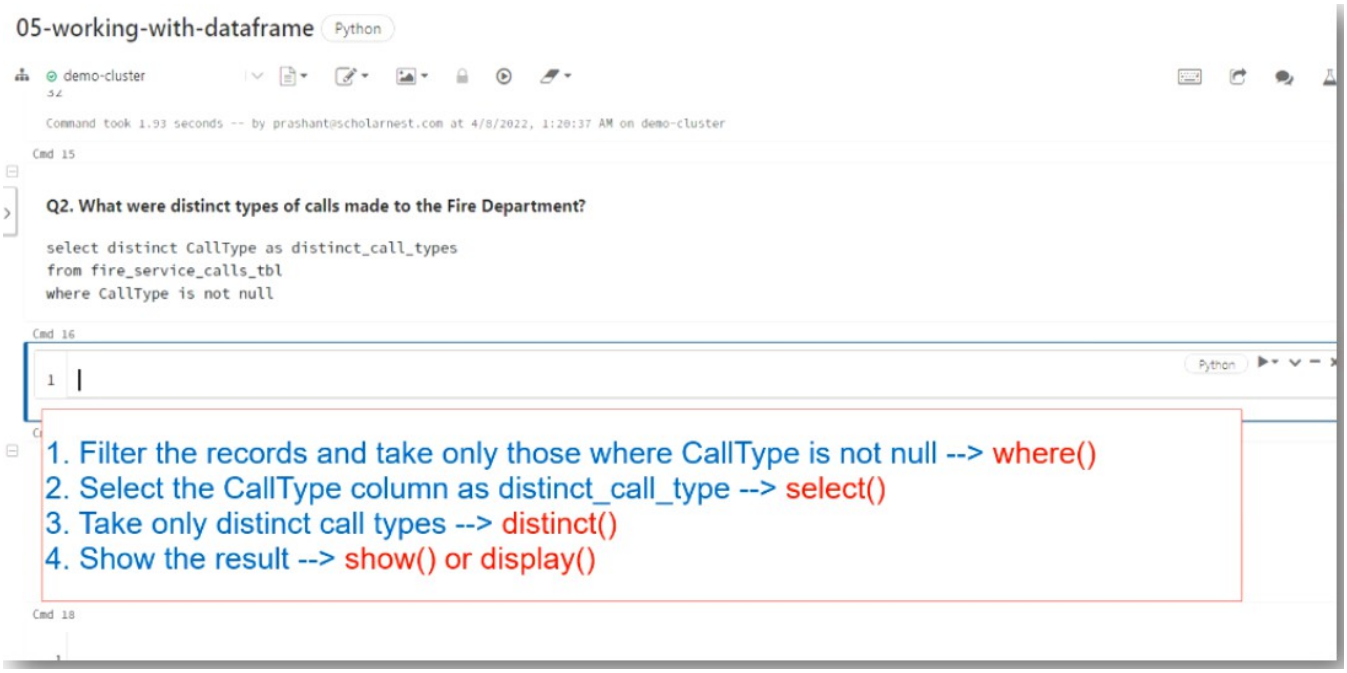
Make sure you have attached a running cluster to your notebook, and run the entire notebook once before you start working on this notebook further.



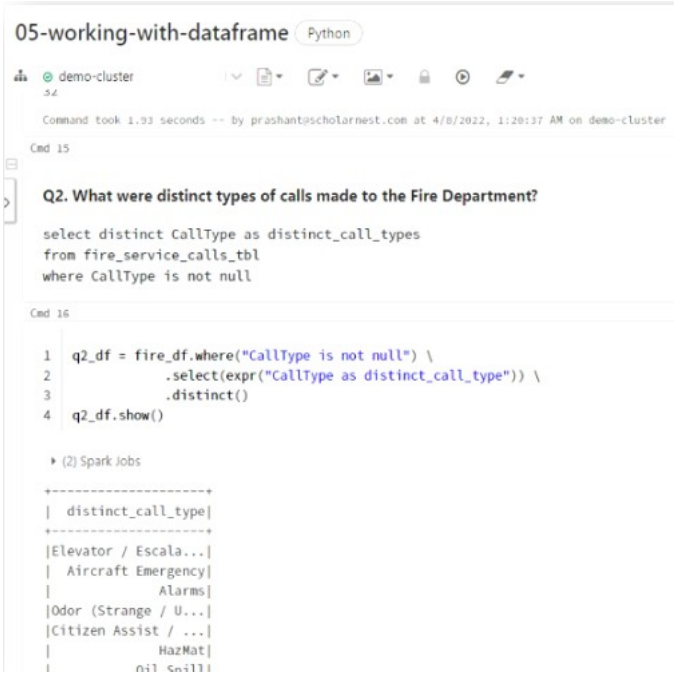
Make sure you have attached a running cluster to your notebook, and run the entire notebook once before you start working on this notebook further. Whenever you stop working on the Databricks cloud and want to return, you must start a new cluster, attach it to your notebook and recreate all the objects you want to use.



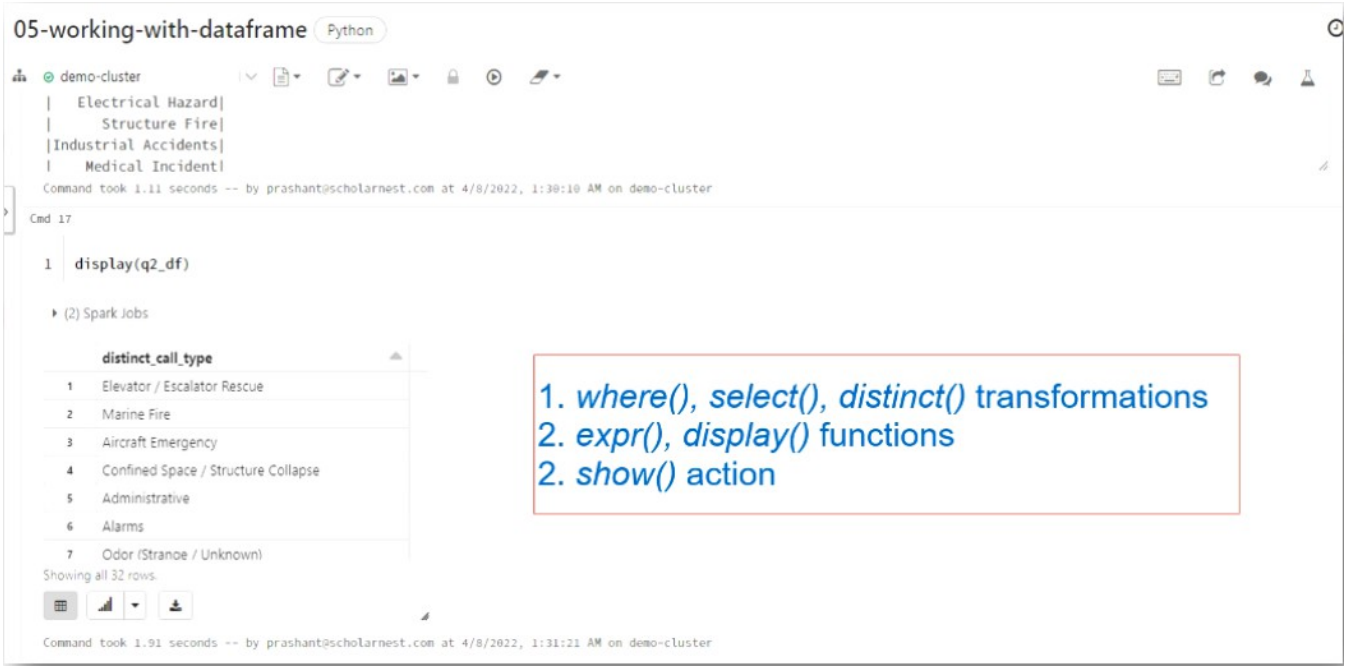
We already solved the first question in the previous lecture. Now, let us look at the second question. We have the logic framed for this question in the image below.



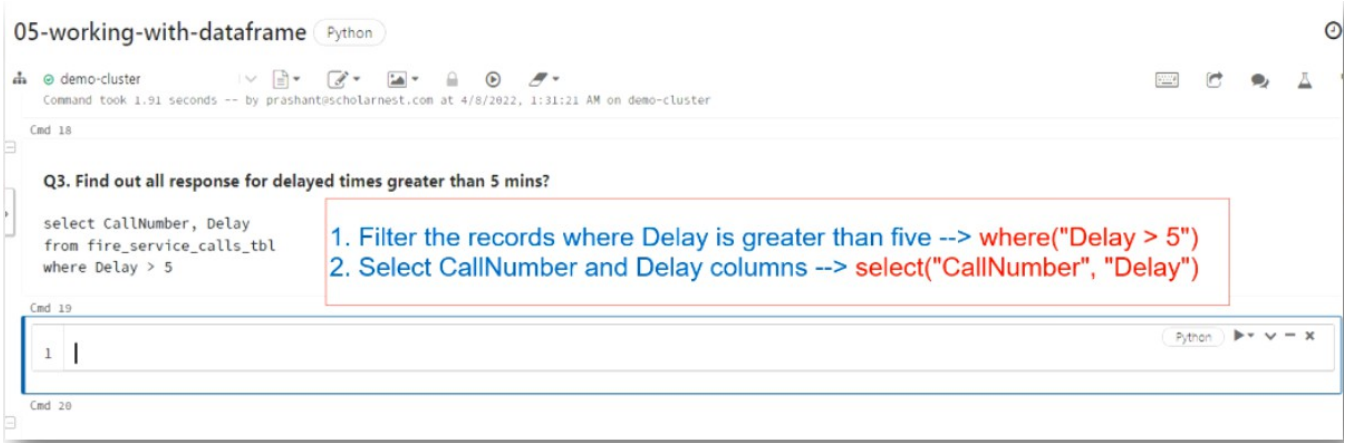
Here is the code for the logic we framed in the second question. I am using fire_df and applying where transformation. The filter condition takes only those records where CallType is not null. Then I am chaining the select transformation which takes a column name. However, I wanted to give an expression. So, I must use the expr() function to evaluate my expression. Then I am chaining the distinct() transformation. And last step is to take action and show the results.



You can also use to the display() function depicts the formatted results. The display() function is not part of Apache Spark. It is a utility function added by Databricks. And the show() method is a commonly used Spark action which shows a plain text output. So, we learned the following Dataframe methods from the second question.



Now let us look at the 3rd question. The question asks for all responses for delayed times greater than 5 mins. We have framed the logic for the same in the image shown below.



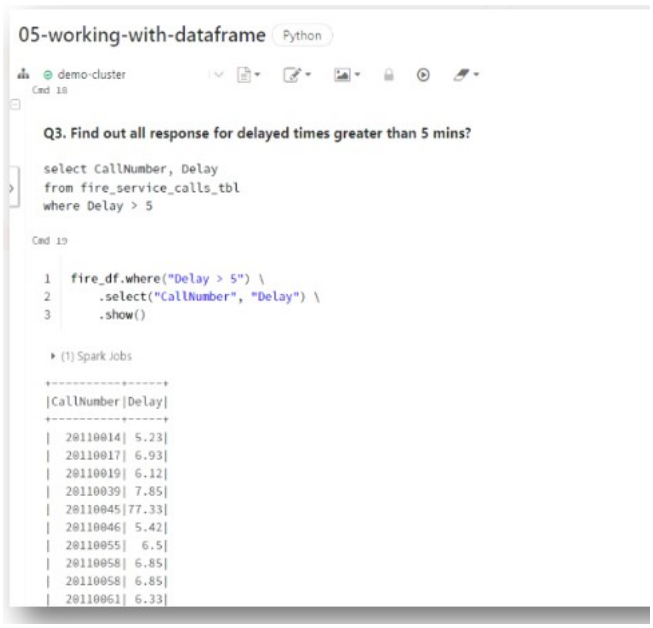
We have implemented the logic into code and we can see the desired output below. We are doing two new things in this code:

1. We are selecting two columns in the select() method. The select() method takes a comma-

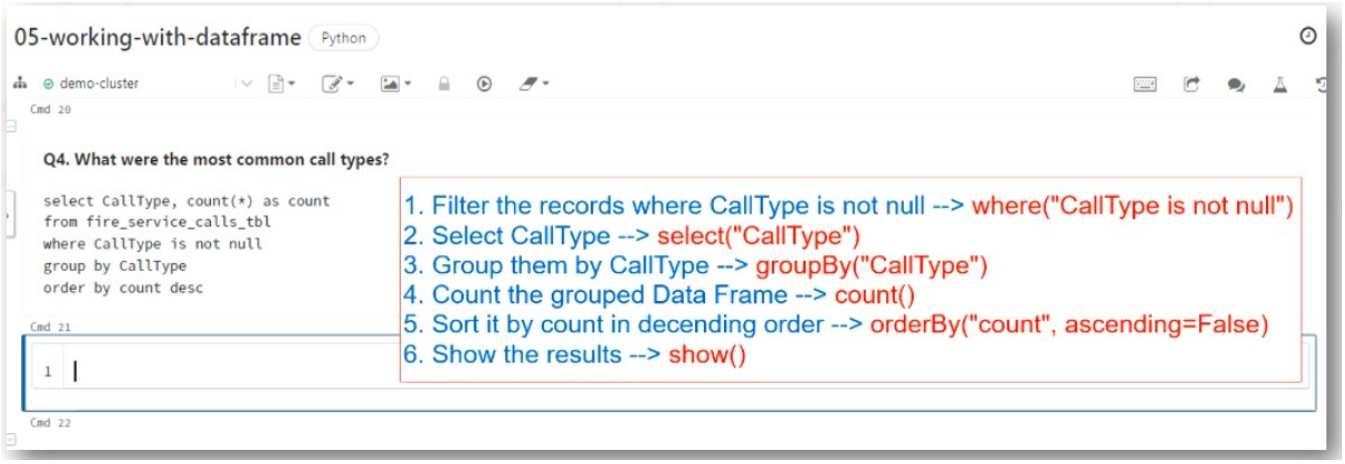
   separated list of columns.

2. We chained the show() action. Remember that you can chain one and only one Action, and that

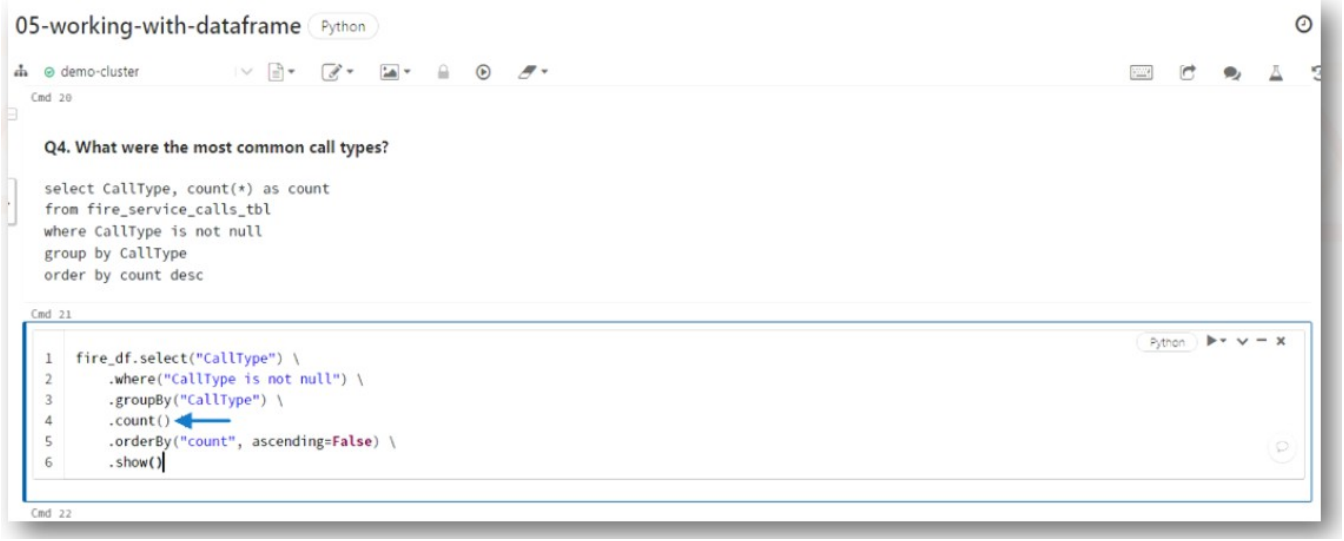
   must be the last method in the chain. So I cannot chain anything else after the show() method.



Here we have the next question and it's logic along side. We want to filter records where the call type is not null and select the CallType. But we want to find the most common CallType. So we must count the calls for each CallType. Hence, we will group by the CallType and then count for each group. The last step is to sort the record by the count in descending order to know which ones are the most common.



Here is the code for the fourth question. But I want to highlight one thing here, I am using count() transformation in this example. But we already learned that the count() is an action.



So we have two count() methods in Spark: 1. Dataframe.count() 2. GroupedData.count()

If you are using a count before the groupBy(), it will be an action, and you cannot chain any other transformation after the count () Action. However, if you use a count() method after the groupBy(), it will be a transformation, and you can chain further transformations.

We can see the desired output for the fourth question as well. The remaining 6 questions are given for practice, you can compare your answer with the solutions in the material provided.

05-working-with-dataframe Python

demo-cluster

Cmd 21

```
1  fire_df.select("CallType") \
2      .where("CallType is not null") \
3      .groupBy("CallType") \
4      .count() \
5      .orderBy("count", ascending=False) \
6      .show()
```

(2) Spark Jobs

```
+--------------------+-------+
|            CallType|  count|
+--------------------+-------+
|    Medical Incident|2843475|
|      Structure Fire| 578998|
|              Alarms| 483518|
|   Traffic Collision| 175507|
|Citizen Assist / ...|  65360|
|               Other|  56961|
|        Outside Fire|  51603|
|        Vehicle Fire|  20939|
|        Water Rescue|  20037|
|Gas Leak (Natural...|  17284|
|   Electrical Hazard|  12608|
|Elevator / Escala...|  11851|
|Odor (Strange / U...|  11680|
|Smoke Investigati...|   9796|
|          Fuel Spill|   5198|
```