

Applied Accelerated Artificial Intelligence
Prof. Saurav Agarwal
Department of Computer Science and Engineering
Indian Institute of Technology, Palakkad

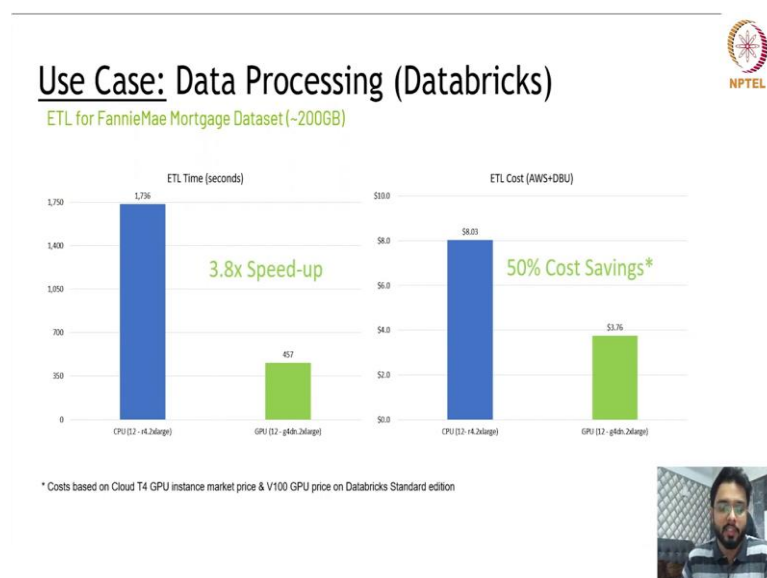
Lecture - 53
Accelerated ETL Pipeline with SPARK part 2

(Refer Slide Time: 00:14)



Ok. So, I have been telling about that there is a lot of performance improvement and all that stuff; but let us understand how much is the performance improvement.

(Refer Slide Time: 00:27)



So, we picked up a use case, where we had data processing from data we did data processing on based on the public dataset of 200GB CSV which is called the FannieMae Mortgage Dataset and you can find out, I will show you the links and all in some time; everything for notebook and all available on the internet.

So, there you will find that ok; you will be seeing that ok data in the data bricks environment, if you use 12 node cluster of CPUs with r4.2xlarge GPU type on the T4 instances versus the T4 instances of cloud where you use in data AWS theg4dn to its large environment.

So, you will see that the same ETL which is reading a CSV file, converting into parquet format and then, reading it, again doing some aggregations and transformations and then, writing it back to the parquet file is doing the same job 4 times, almost 4 times faster than the CPU counterpart. We call it is 4 times faster; the dollar cloud bill of CPU is 8 dollars and the cloud bill of a AWS of GPU is 3.76 dollars. Since we can see that we are saving 50 percent cost as well, in addition to accelerating it at 4x performance.

(Refer Slide Time: 01:56)

NVIDIA DECISION SUPPORT (NDS)



NVIDIA Decision Support (NDS) is our adaptation of an industry-standard* data science benchmark often used by Spark customers and providers.

NDS consists of the same 100+ SQL queries as the industry standard benchmark but has modified parts for dataset generation and execution scripts.

* NDS is derived from the TPC-DS benchmark and is used for internal performance testing. Results from NDS are not comparable to TPC-DS.



So, this was a simple dataset and simple used case. But when we go to enterprises, and industry; so, we do use something called a decision support. Ah. So, decision support is something which is a industry accepted benchmarks which is cross like maintained by an independent organization. So, that any kind of processing benchmarks you need to test,

we can go through this at the TPC-DS benchmark and we can compare it. Suppose if you want to compare spark with map reduce or map reduce spark 2 versus spark 3.

So, everyone uses this TPC-DS as a benchmark tool for doing that kind of testing. Hence, we thought that using this particular benchmark would be more preferable so that we can prove our point to the larger customers as well.

(Refer Slide Time: 02:51)



DATASET

- Approximately 3 TB of raw data
- Compressed using parquet about 1 TB on disk
- Data uses double values instead of decimal values
- The data set is partitioned
- Stored in the native file system for the environment (HDFS for EGX cluster, Local files for DGX-A100)



So, we use 3 terabyte of raw data; yes, we compressed the data of 3 terabyte of CSV into 1 terabyte on disk and then, we use double values instead of decimal values and the data was partitioned, stored in the native file system for the environment using the EGX cluster and the DGX A100.

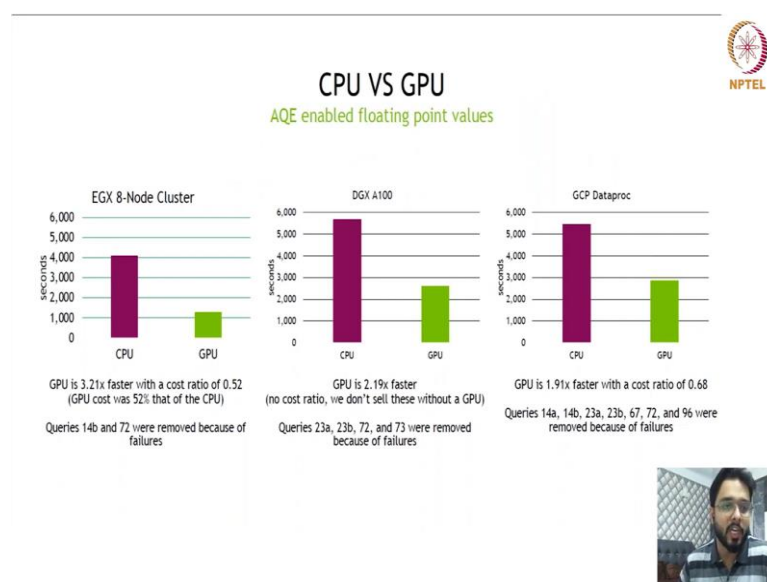
(Refer Slide Time: 03:17)



	EGX / NVIDIA Certified OEM servers	DGX A100	GCP Dataproc
Nodes	8	1	1 driver (CPU only), 8 workers
CPU	2 x AMD EPYC 7452 (64 cores/128 threads)	2 x AMD Rome 7742 (128 cores/256 threads)	n1-standard-4 (driver) 8 x n1-highmem-16 (workers)
GPU	2 x NVIDIA Ampere A100, PCIe, 250W, 40GB	8 x NVIDIA Ampere A100 40GB	1 x 16GB T4 per executor
RAM	0.5 TB	2 TB	104 GB
Storage	4 x 7.68 TB Gen4 U.2 NVMe	8 x 3.84 TB Gen4 U.2 NVMe	Google Cloud Storage
Networking	1 x Mellanox CX-6 Single Port HDR100 QSFP56	8 x Mellanox CX-6 Single Port HDR 200Gb/s InfiniBand	32 Gbps
Cost w/o GPU	~\$42,000 per w/ bulk discount	N/A	\$9.08/hour incl GCE + Dataproc
Cost w/ GPU	~\$71,000 per w/ bulk discount	Approximately \$239,000 retail	\$11.88/hour incl GCE + Dataproc
Software	HDFS (Hadoop 3.2.1) Spark 3.0.2 (stand alone)	Spark 3.0.2 (stand alone)	Dataproc Spark 3.0.1 + YARN

So, what we did is if you see this diagram, so the hardware we compared against Google Data Proc, DGX A100 which is combination of 8 A100 GPUs and finally, EGX which is combination of 8 servers with 2 GPUs each ok. So, we did this benchmarking on these three kind of hardware cloud on premise and two type of on premise; node based and specialized machine based.

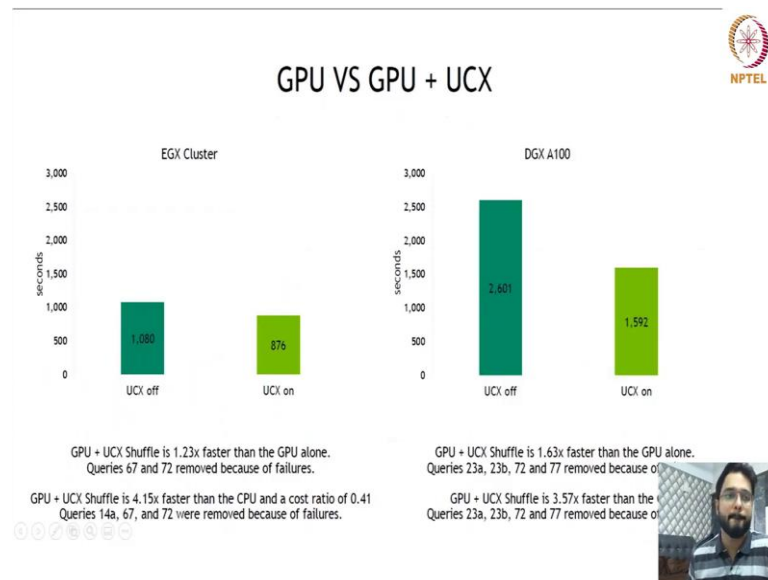
(Refer Slide Time: 03:46)



So, we found that whether it is EGX, DGX or cloud, we are definitely faster. Though, we are then fastest in the EGX kind of system, where we have 2 GPUs each in the each

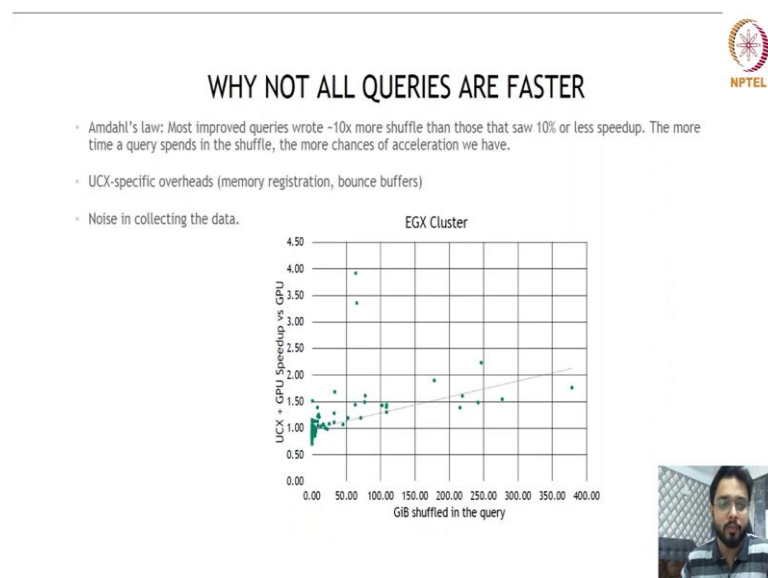
server and multiple such servers. DGX was faster, but it was around 2.19x faster and then, on cloud, it was almost 2x faster. The latest benchmark we have does not have even failures all the queries are running as usual.

(Refer Slide Time: 04:16)



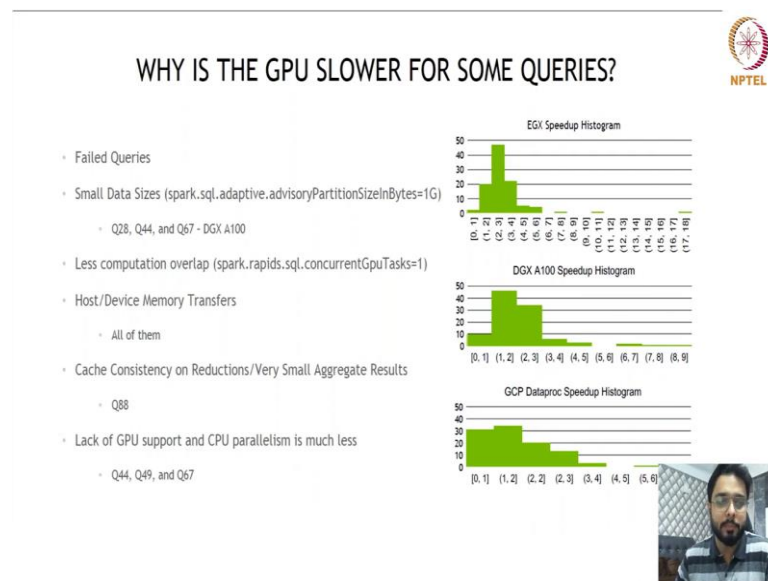
Ah So, if we add the UCX, so how much faster it is? The accelerated spark shuffle, if we add that on top of the normal GPU acceleration, we further get 1.63 X faster than GPU alone and if we compare it with CPU, we get 3.5x faster. So, hence, we see 1.5x to 2x faster for the networking acceleration as well.

(Refer Slide Time: 04:47)



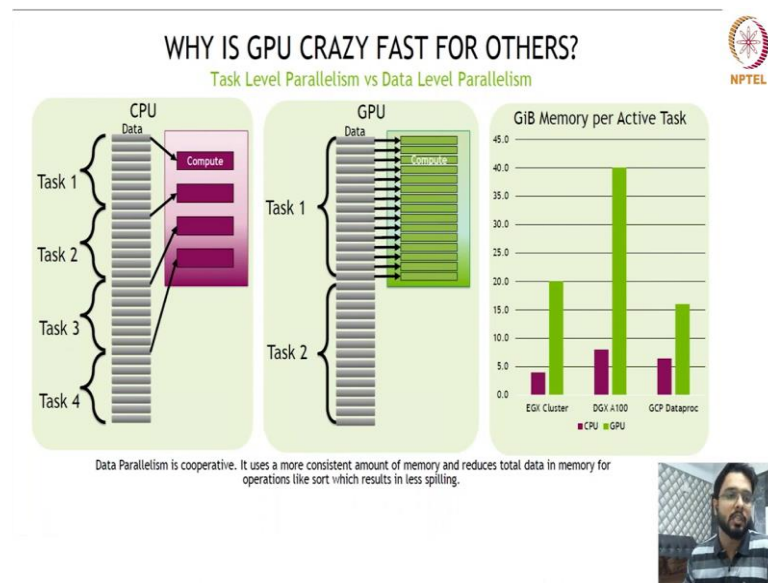
All type of queries are not faster. So, there is no one stop solution as of now that you can have to run all the queries faster. So, here if you see UCX plus GPU speed up versus GPU. So, the amount of the amount of data, we are shuffling is directly proportional to the acceleration we are getting. If we are not shuffling a lot of data, the acceleration is not that much. If we are shuffling a lot of data, then we have a lot of acceleration.

(Refer Slide Time: 05:23)



So, why GPUs are slower? There are if the data size is small, if the data queries are failing, if there is a lot of CPU usage, if there is not lack of GPU support, these kind of queries we are seeing to be slower.

(Refer Slide Time: 05:45)



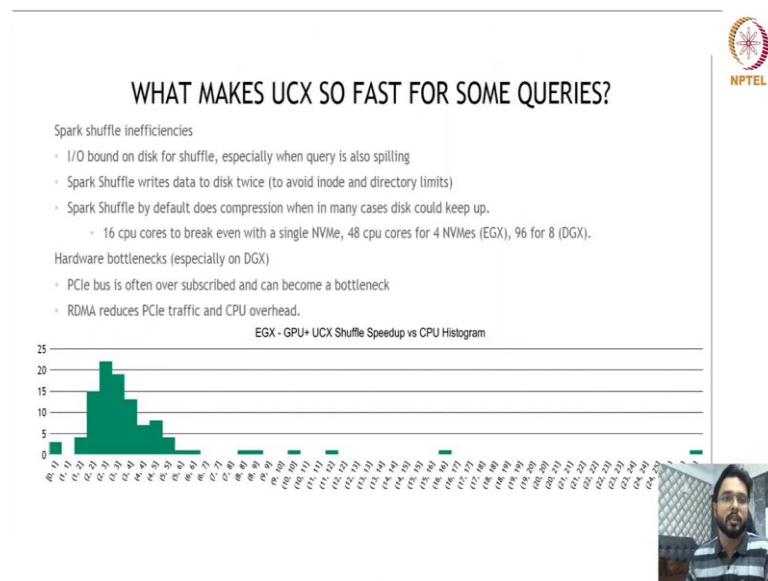
But most of the queries are faster, they are crazy fast because we are able to achieve task level parallelization. So, if you see here, the amount of data we are processing or the number of parallel tasks we are running; task 1, 2, 3, 4 amount of data we have. So, the task 1 has a very less amount of data as compared to the task 1 of GPU. So, overall throughput in addition to the parallelization is increased in processing using the GPU base system. Hence, we are seeing the fastness.

(Refer Slide Time: 06:23)



Cloud is a bit slow because it does not have networking interconnects faster; the collocation of the nodes are not there sometimes. It has shared nodes, data proc specially has external shuffle for stability, data locality is missing there. So, if you are sitting in Bangalore and if you are running cluster in US, then the data locality issue is also there. Hence it is slower.

(Refer Slide Time: 06:51)



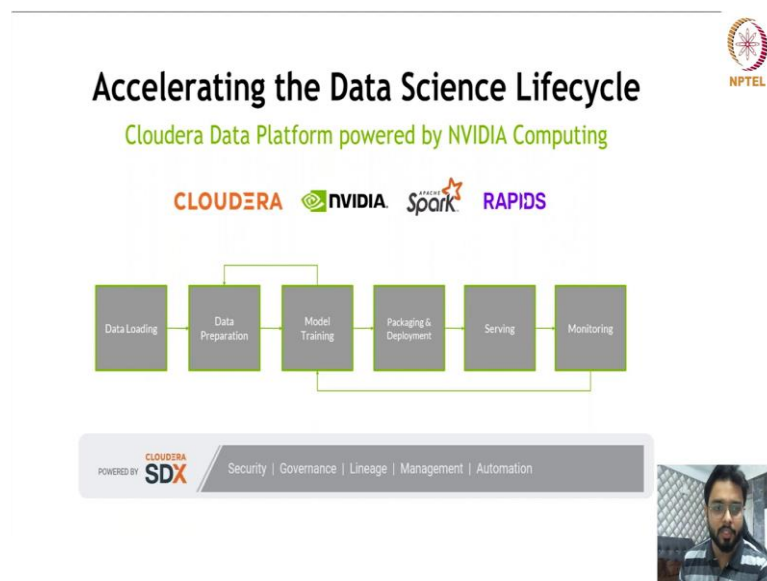
Some of the queries are crazy fast on networking interconnect because sparks of shuffle are inefficient for I/O bound or disk for shuffle which writes a lot of data to the disk and many times to the disk; sometimes there are hardware bottlenecks, where PCIe bus is involved as I was showing in the diagram. So, for all those scenarios, the thus UCX is fast and slow respectively ok.

(Refer Slide Time: 07:22)



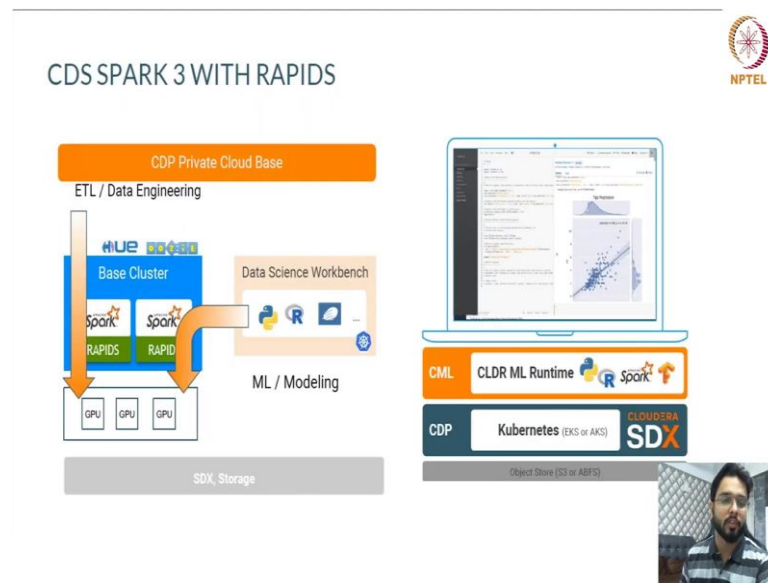
Let us move to the Cloudera section now. So, we accelerate rapids on cloudera as well.

(Refer Slide Time: 07:36)



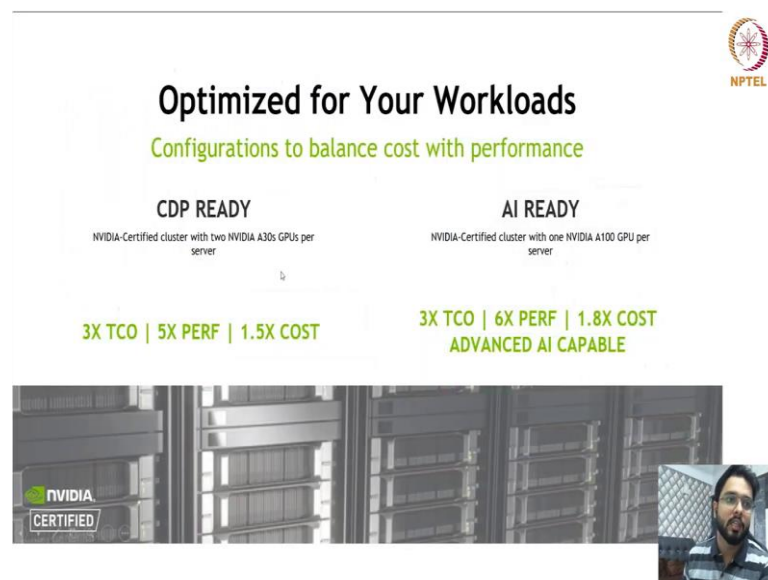
So, where we if you see a typical data science workflow which starts with data loading, data preparation, model training, packaging deployment, serving and monitoring. So, all this based on cloudera data science, data platform is made available to the users and we, at media are accelerating each part of it, if not all. ah

(Refer Slide Time: 07:59)



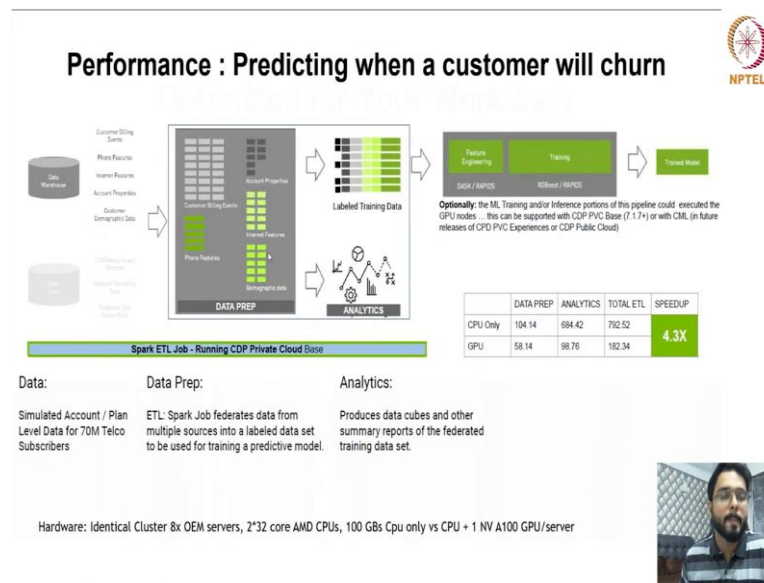
So, we know that spark, the latest version of cloudera called cloudera data platform has spark 3 enabled capability where we can enable spark 3.

(Refer Slide Time: 08:13)



And we can optimize for our workloads as well based on GPU acceleration. So, we have seen that 5X performance acceleration, where we are just saving 3X of the total cost because of the 5X acceleration. Based on the type of GPUs, it is more or less. So, if we use A30; two A30 per node, then we see that ok it is 5X faster; if we use one A100 per node, it is 6X faster.

(Refer Slide Time: 08:47)

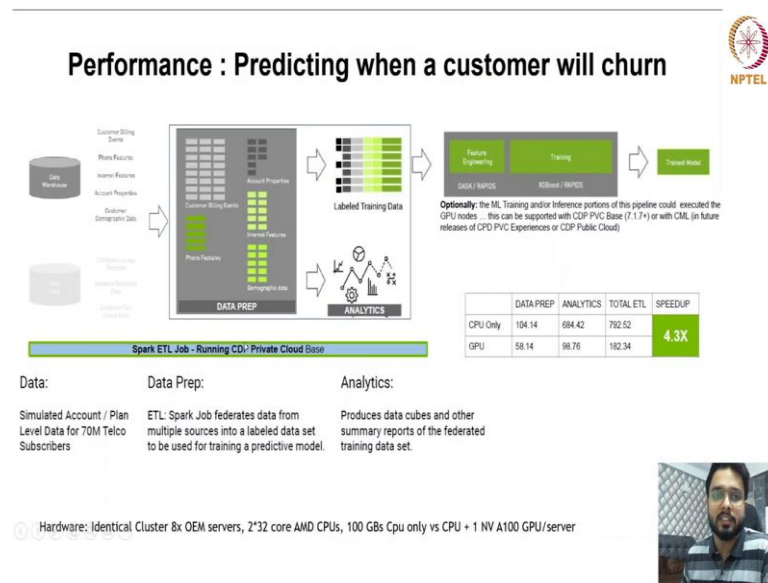


So, we did some benchmarking specifically on cloudera data platform as well, where we used identical cluster of CPU as a GPU, where we use 8 servers with 232 core AMD CPUs and 100 GBs CPU versus the same CPU plus 1 A100 GPU per server.

So, there we saw another in our data analytics pipeline, where we had data from customer billing, phone features, internet features, account properties, customer demographic data and so on and so forth. We saw that where we doing data preparation like finding out the account properties, internet features, customer billing events, phone features, demographic data and then, doing some analytics on top of it was faster.

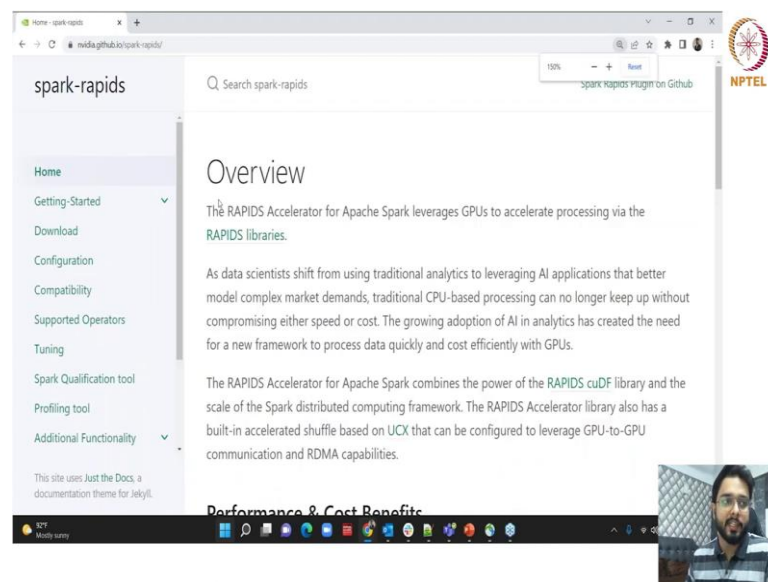
So, how much faster it was? So, it was around 5X faster; 4.3X faster. Now, this 3X faster; 4.3X faster is coming from the entire pipeline on an average, where we are combining data preparation, analytics both together. So, analytics also involve the data science portion, where we are also doing feature engineering model training and so on and so forth ok.

(Refer Slide Time: 10:11)



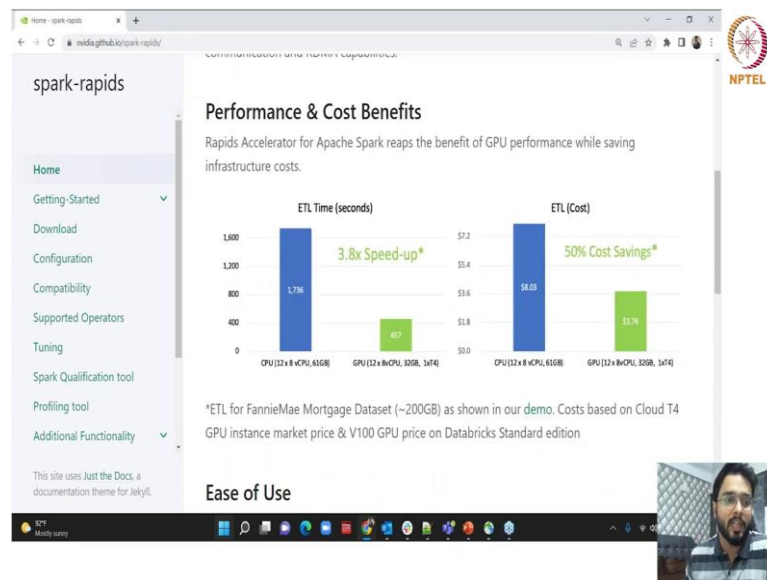
So, that is the from the theory perspective. Let us let me show you some hands on as well.

(Refer Slide Time: 10:28)

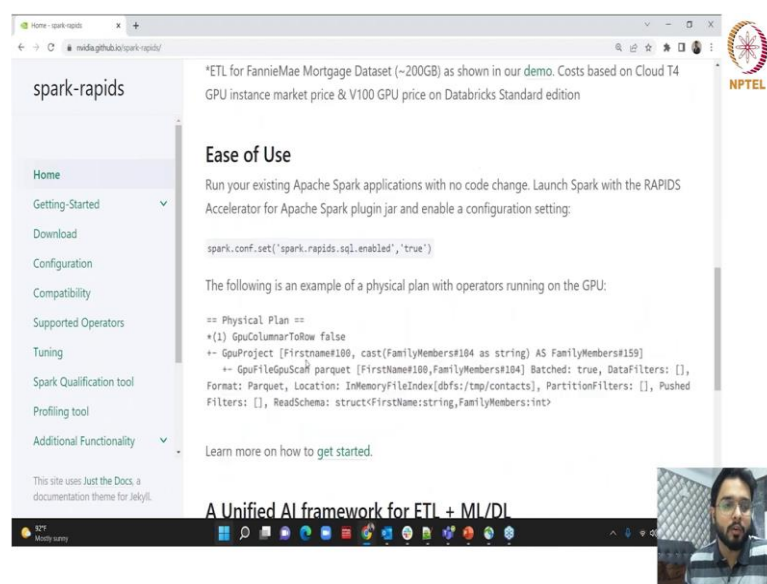


So, first of all, I will like to show this particular website, where we have the entire details about the spark on GPU acceleration. So, this is called spark nvidia.github.io/spark-rapids.

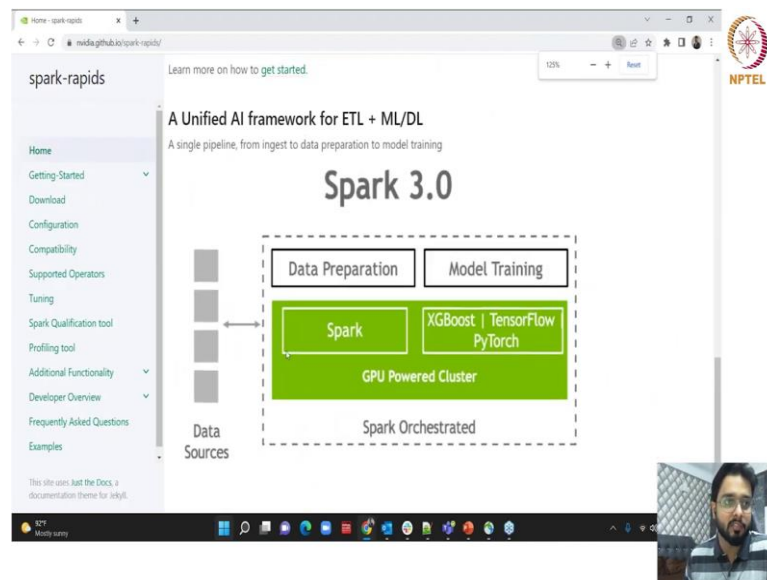
(Refer Slide Time: 10:45)



(Refer Slide Time: 10:45)

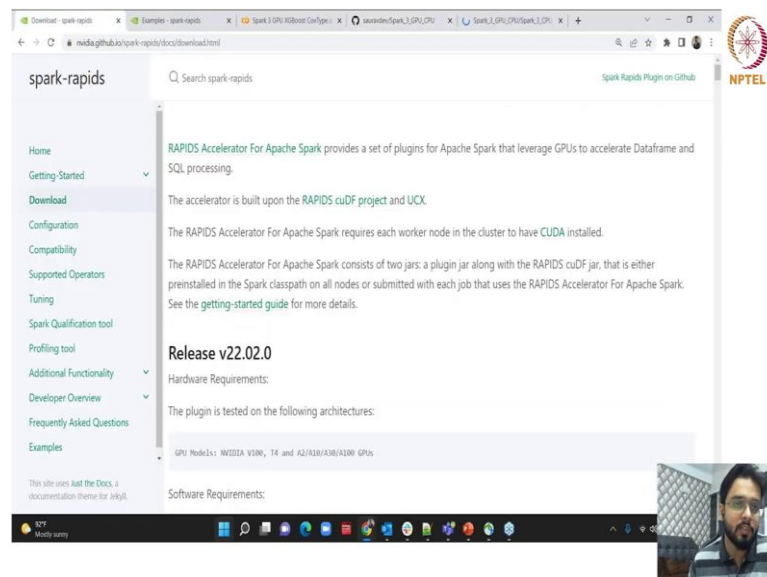


(Refer Slide Time: 10:46)



There you will see what kind of acceleration, we give how to use at a high level and all the details which I will went through today. Then, it will also give ok how to configure it On-Prem, EMR, Databricks etc etc; how to download those jars and configurate; how it to check the compatibility, what are the operative just a put it.

(Refer Slide Time: 11:00)



(Refer Slide Time: 11:06)

The screenshot shows the 'spark-rapids' documentation page. The left sidebar contains a navigation menu with items like Home, Getting-Started, Download, Configuration, Compatibility, Supported Operators (highlighted), Tuning, Spark Qualification tool, Profiling tool, Additional Functionality, Developer Overview, Frequently Asked Questions, and Examples. The main content area is titled 'General limitations' and includes a section on 'Decimal' types. It explains that the RAPIDS Accelerator for Apache Spark supports 128-bit precision for Decimal types starting from version 21.12. A table below lists operations and their resulting precision and scale.

Operation	Result Precision	Result Scale
$e1 + e2$	$\max(p1, s2) + \max(p1 - s1, p2 - s2) + 1$	$\max(s1, s2)$
$e1 - e2$	$\max(p1, s2) + \max(p1 - s1, p2 - s2) + 1$	$\max(s1, s2)$
$e1 * e2$	$p1 + p2 + 1$	$s1 + s2$
$e1 / e2$	$p1 - s1 + s2 + \max(6, s1 + p2 + 1)$	$\max(5, s1 + p2 + 1)$
$e1 \text{ \& } e2$	$\min(p1 - s1, p2 - s2) + \max(s1, s2)$	$\max(s1, s2)$
$e1 \text{ union } e2$	$\max(p1, s2) + \max(p1 - s1, p2 - s2)$	$\max(s1, s2)$

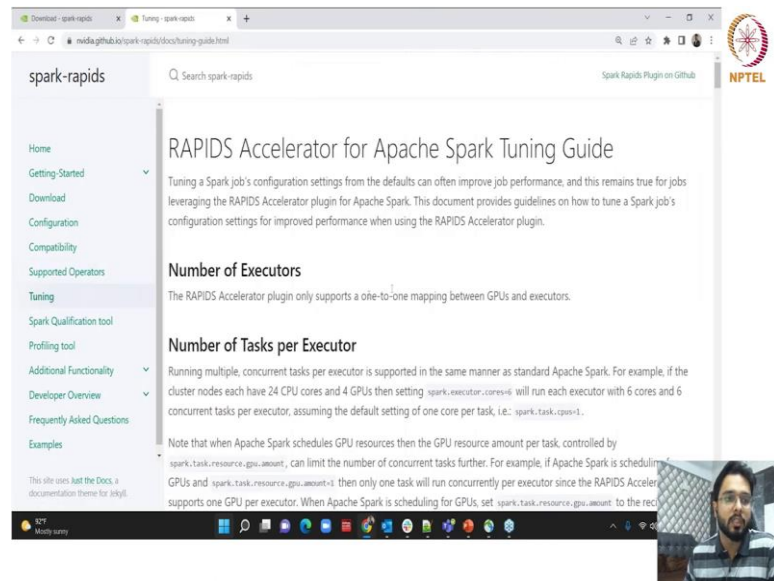
(Refer Slide Time: 11:10)

This screenshot continues the 'Decimal' section from the previous slide. It provides a detailed table of operations and their resulting precision and scale. Below the table, a paragraph explains that Spark inserts `ProratePrecision` to CAST both sides to the same type. It notes that GPU mode may fall back to CPU mode if the result's Decimal precision is within 18 digits. For example, `Decimal(6,2) * Decimal(5,1)` results in `Decimal(11,3)`, which runs on CPU because the precision is within 18 digits. It also mentions that GPU mode assumes the result is `Decimal(19,4)` and that there are extreme cases where Spark can temporarily return a Decimal value larger than what can be stored in 128-bits, then uses the `checkOverflow` operator to round it to a desired precision and scale.

Operation	Result Precision	Result Scale
$e1 + e2$	$\max(p1, s2) + \max(p1 - s1, p2 - s2) + 1$	$\max(s1, s2)$
$e1 - e2$	$\max(p1, s2) + \max(p1 - s1, p2 - s2) + 1$	$\max(s1, s2)$
$e1 * e2$	$p1 + p2 + 1$	$s1 + s2$
$e1 / e2$	$p1 - s1 + s2 + \max(6, s1 + p2 + 1)$	$\max(5, s1 + p2 + 1)$
$e1 \text{ \& } e2$	$\min(p1 - s1, p2 - s2) + \max(s1, s2)$	$\max(s1, s2)$
$e1 \text{ union } e2$	$\max(p1, s2) + \max(p1 - s1, p2 - s2)$	$\max(s1, s2)$

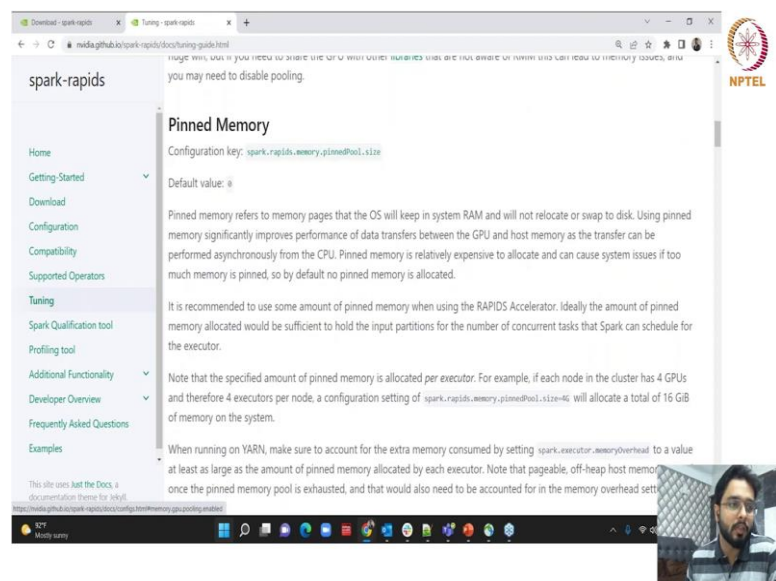
As I was saying that not everything is supported; decimal some limitation is there, timestamp calendar intervals and so on so forth.

(Refer Slide Time: 11:15)



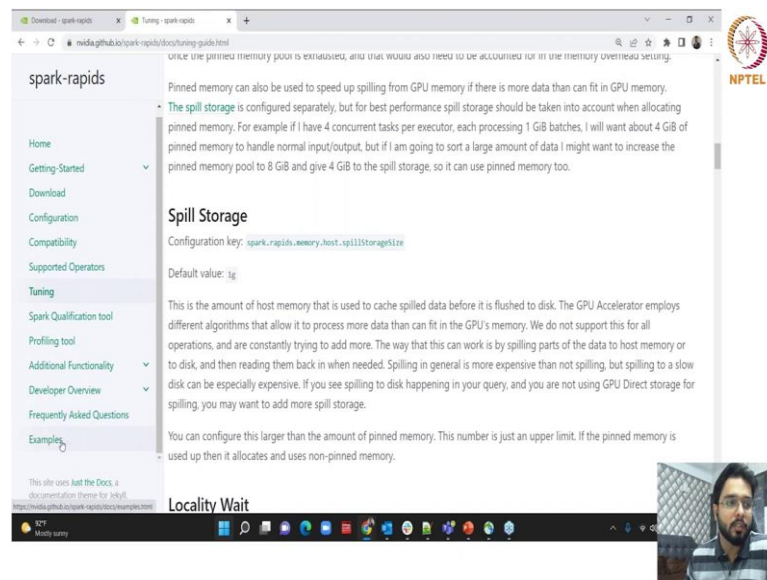
How to tune the accelerator for spark tuning?

(Refer Slide Time: 11:21)



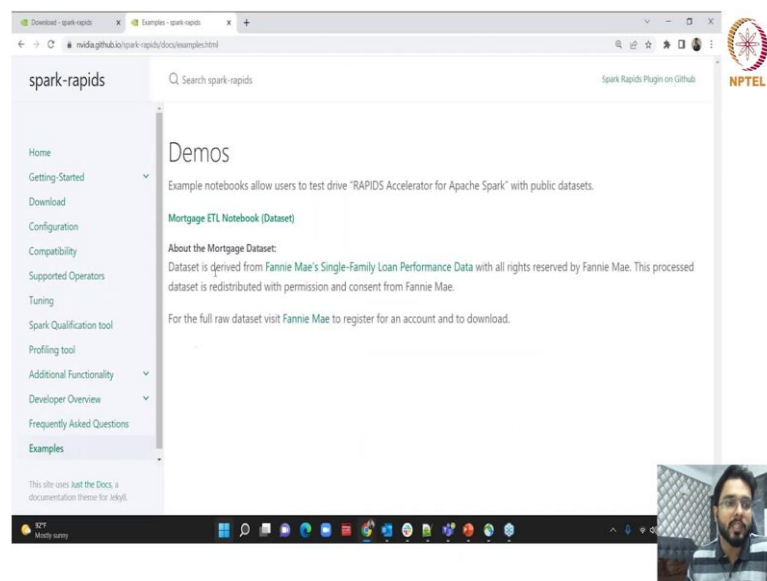
How to set the number of executor, pooled memory, print, pinned memory; all these details will find here.

(Refer Slide Time: 11:25)



The screenshot shows a web browser displaying the 'spark-rapids' documentation page. The left sidebar contains a navigation menu with items like Home, Getting-Started, Download, Configuration, Compatibility, Supported Operators, Tuning, Spark Qualification tool, Profiling tool, Additional Functionality, Developer Overview, Frequently Asked Questions, and Examples. The main content area is titled 'Spill Storage' and discusses how spilled data is cached in host memory before being flushed to disk. It includes a configuration key 'spark.rapids.memory.host.spillStorageSize' and a default value of '1g'. A small video inset in the bottom right corner shows a man speaking.

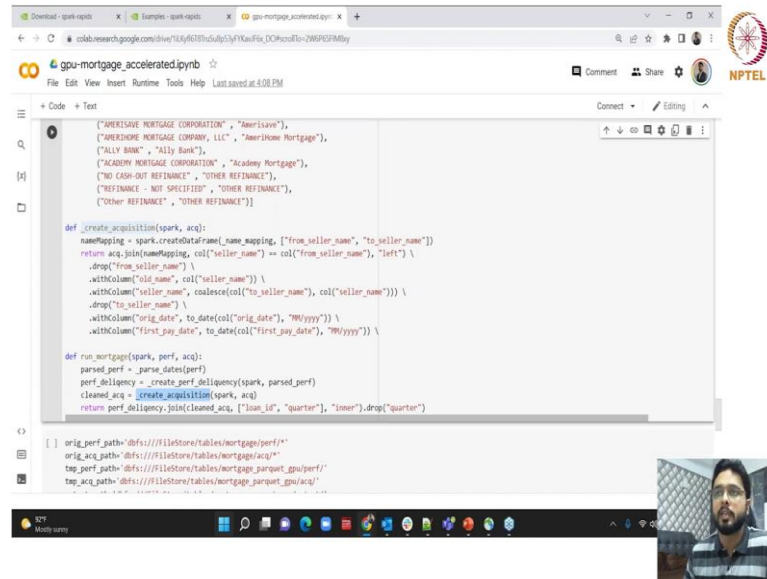
(Refer Slide Time: 11:31)



The screenshot shows the 'spark-rapids' documentation page, specifically the 'Demos' section. The left sidebar is the same as in the previous slide. The main content area is titled 'Demos' and describes example notebooks for testing the RAPIDS Accelerator for Apache Spark. It highlights the 'Mortgage ETL Notebook (Dataset)' and mentions that the dataset is derived from Fannie Mae's Single-Family Loan Performance Data. A small video inset in the bottom right corner shows the same man speaking.

And also, if you want to do your hands on some examples of the ETL notebook, you will find. So, one such notebook apart from this is the mortgage dataset we have.

(Refer Slide Time: 11:44)



The screenshot shows a Databricks notebook interface with the title 'gpu-mortgage_accelerated.ipynb'. The code is written in Python and defines two main functions: `_create_acquisition` and `run_mortgage`. The `_create_acquisition` function takes a Spark session and an acquisition ID as input, creates a DataFrame with mortgage data, and returns a Spark DataFrame. The `run_mortgage` function takes a Spark session and a performance DataFrame as input, creates a Spark DataFrame, and returns a Spark DataFrame. The notebook also includes a section for saving the results to Delta Lake.

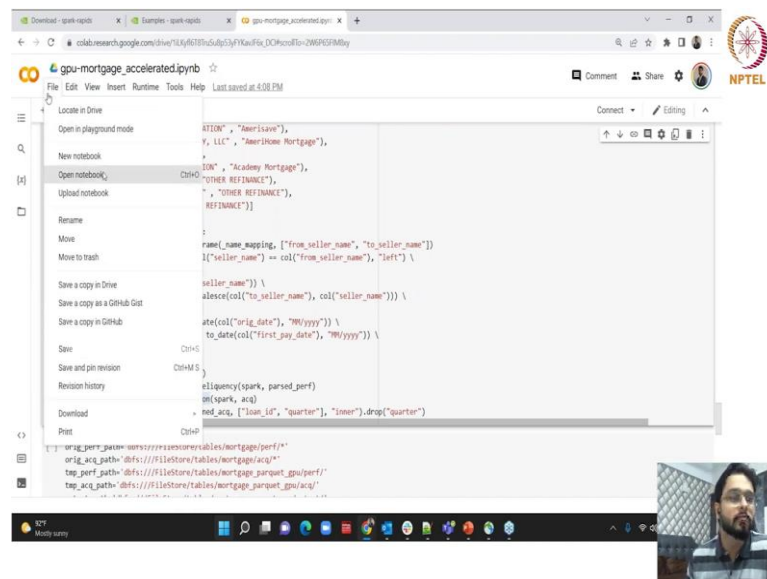
```
[{"OVERSEAS MORTGAGE CORPORATION", "Americas"}, {"AMERINDIAN MORTGAGE COMPANY, LLC", "Americas Mortgage"}, {"ALLY BANK", "Ally Bank"}, {"ACADEMY MORTGAGE CORPORATION", "Academy Mortgage"}, {"TWO CASH-OUT REFINANCE", "OTHER REFINANCE"}, {"REFINANCE - NOT SPECIFIED", "OTHER REFINANCE"}, {"OTHER REFINANCE", "OTHER REFINANCE"}]

def _create_acquisition(spark, acq):
    name_mapping = spark.createDataFrame(name_mapping, ["from_seller_name", "to_seller_name"])
    return acq.join(name_mapping, col("from_seller_name") == col("from_seller_name"), "left") \
        .drop("from_seller_name") \
        .withColumn("old_name", col("seller_name")) \
        .withColumn("seller_name", coalesce(col("to_seller_name"), col("seller_name"))) \
        .drop("to_seller_name") \
        .withColumn("orig_date", to_date(col("orig_date"), "MM/yyyy")) \
        .withColumn("first_pay_date", to_date(col("first_pay_date"), "MM/yyyy")) \
        .drop("first_pay_date")

def run_mortgage(spark, perf, acq):
    parsed_perf = parse_dates(perf)
    perf_deliqency = _create_perf_deliqency(spark, parsed_perf)
    cleaned_acq = _create_acquisition(spark, acq)
    return perf_deliqency.join(cleaned_acq, ["loan_id", "quarter"], "inner").drop("quarter")

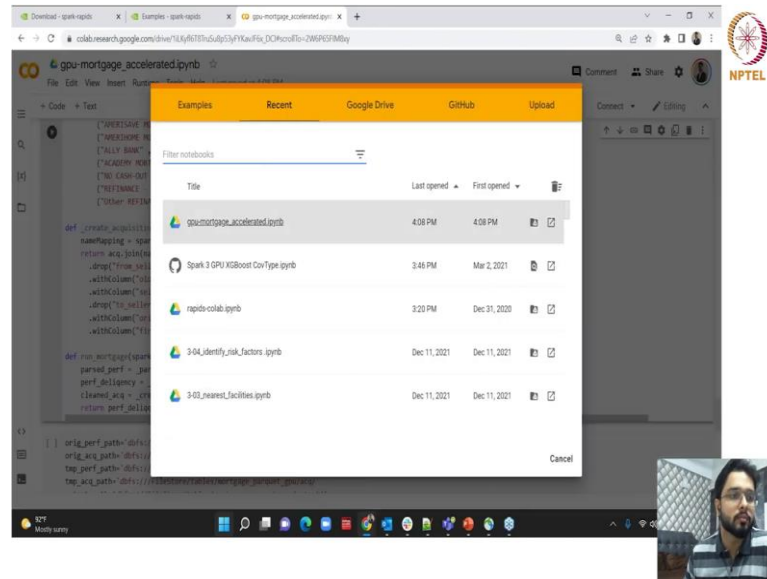
orig_perf_path = dbfs://FileStore/tables/mortgage/perf/
orig_acq_path = dbfs://FileStore/tables/mortgage/acq/
top_perf_path = dbfs://FileStore/tables/mortgage/parquet_gpu/perf/
top_acq_path = dbfs://FileStore/tables/mortgage/parquet_gpu/acq/
```

(Refer Slide Time: 11:49)



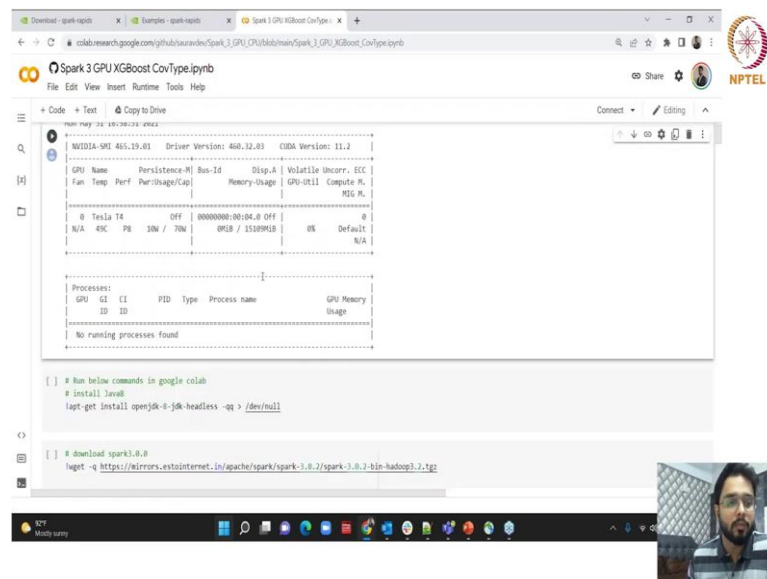
The screenshot shows the same Databricks notebook interface, but with the file explorer on the left side open. The file explorer shows a list of files and folders, including 'Locate in Drive', 'Open in playground mode', 'New notebook', 'Open notebook', 'Upload notebook', 'Rename', 'Move', 'Move to trash', 'Save a copy in Drive', 'Save a copy as a GitHub Gist', 'Save a copy in GitHub', 'Save', 'Save and pin revision', 'Revision history', 'Download', and 'Print'. The notebook content is still visible in the background.

(Refer Slide Time: 11:54)



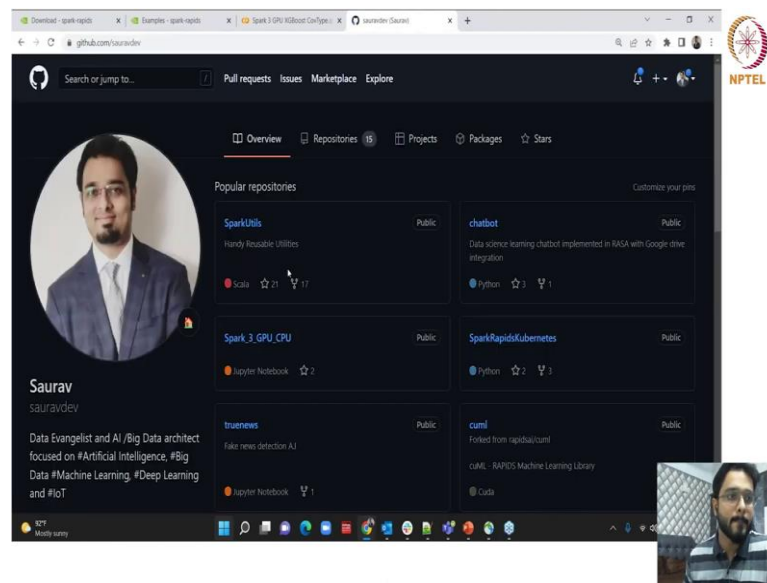
And another notebook is on the and just open that XGBooster.

(Refer Slide Time: 11:56)

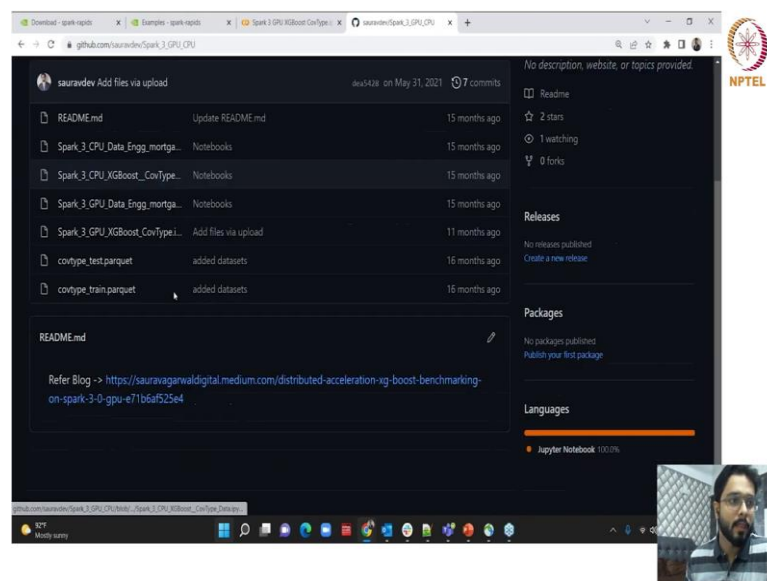


So, this is a simple use case, where I am comparing CPU versus GPU on using colab; Google colab ah. So, you can find these notebooks in my github as well which is again in public website.

(Refer Slide Time: 12:11)

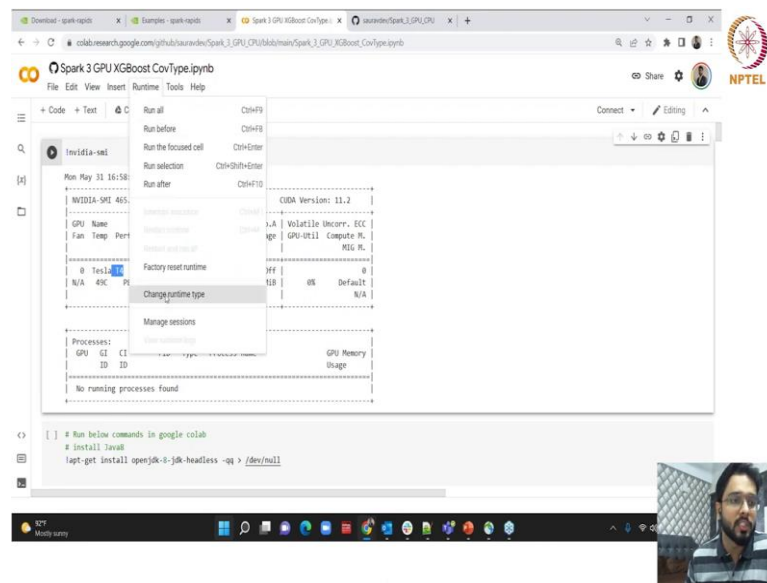


(Refer Slide Time: 12:23)



So, here the CPU versions are there and here, the GPU versions are there. So, I just use the same notebook here and import it in colab and then, I made sure that I am allocated at T4 GPU.

(Refer Slide Time: 12:38)



The screenshot shows the Google Colab interface for a notebook titled 'Spark 3 GPU XGBoost CovType.ipynb'. The 'Runtime' menu is open, and the 'Change runtime type' option is highlighted. The interface also displays the 'nvidia-smi' command output, which shows the GPU status and configuration. The output includes details about the NVIDIA-SMI 465.19.01 driver, the Tesla T4 GPU, and its memory usage. The 'Processes' section shows no running processes found.

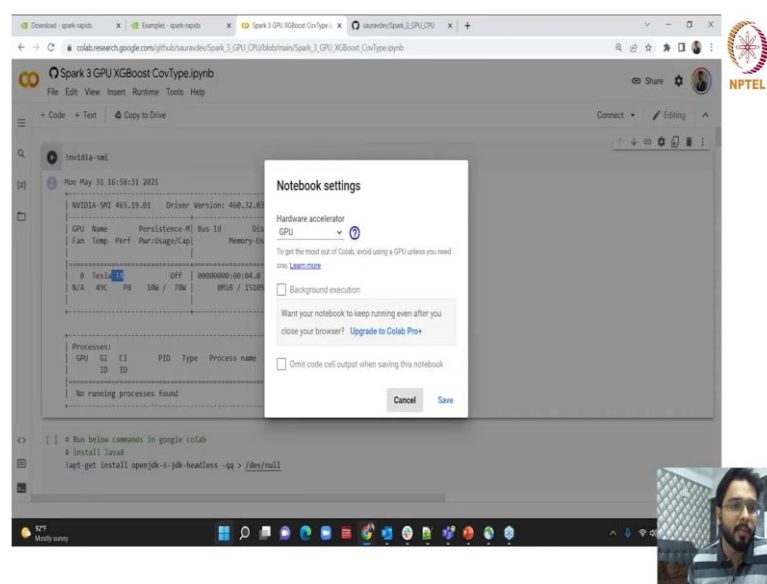
GPU	GI	CI	PID	Type	Process name
0	0	0	0	0	0

Processes:

GPU	GI	CI	PID	Type	Process name
0	0	0	0	0	0

No running processes found

(Refer Slide Time: 12:40)



The screenshot shows the Google Colab interface for the same notebook. A 'Notebook settings' dialog box is open, allowing the user to change the hardware accelerator to 'GPU'. The dialog box also includes options for 'Background execution' and 'Omit code cell output when saving this notebook'. The 'Hardware accelerator' is currently set to 'GPU', and the 'Background execution' checkbox is unchecked.

Notebook settings

Hardware accelerator: GPU

To get the most out of Colab, avoid using a GPU unless you need one. [Learn more](#)

☐ Background execution

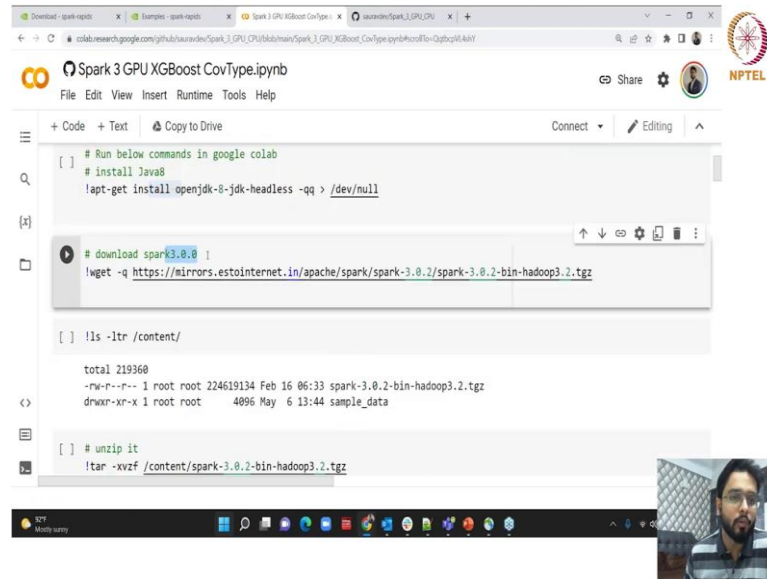
Want your notebook to keep running even after you close your browser? [Upgrade to Colab Pro](#)

☐ Omit code cell output when saving this notebook

Cancel Save

So, if you are not allocated, then you have to go to the runtime, change the runtime type to GPU and then, ok. Again, if you do not see the nvidia-smi command giving tesla T4, then you can do a factory reset and try to run it again and again. So, maybe 5 to 10 times you have to do; but you will definitely get allocated one GPU for sure.

(Refer Slide Time: 12:58)



```
# Run below commands in google colab
# install Java8
!apt-get install openjdk-8-jdk-headless -qq > /dev/null

# download spark3.0.2
!wget -q https://mirrors.estointernet.in/apache/spark/spark-3.0.2/spark-3.0.2-bin-hadoop3.2.tgz

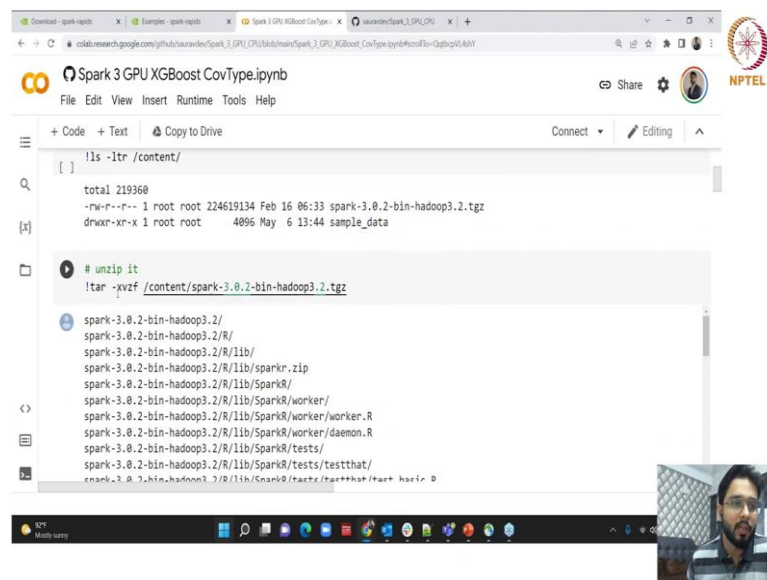
!ls -ltr /content/

total 219360
-rw-r--r-- 1 root root 224619134 Feb 16 06:33 spark-3.0.2-bin-hadoop3.2.tgz
drwxr-xr-x 1 root root 4096 May 6 13:44 sample_data

# unzip it
!tar -xvzf /content/spark-3.0.2-bin-hadoop3.2.tgz
```

Once you are allocated the GPU, you can follow the notebook. I will just magnify and explain at a high level. So, here if you see first, I am running downloading Java8 because it needs Java, then I am downloading spark 3.

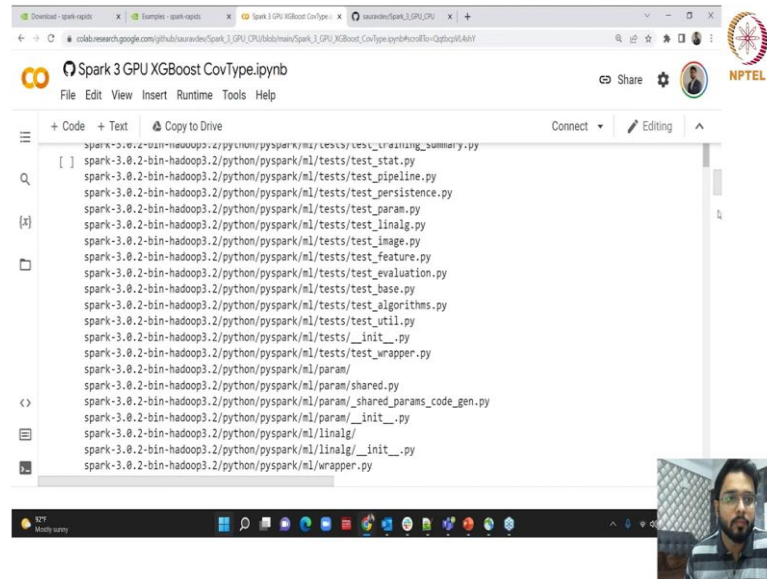
(Refer Slide Time: 13:14)



```
# unzip it
!tar -xvzf /content/spark-3.0.2-bin-hadoop3.2.tgz

spark-3.0.2-bin-hadoop3.2/
spark-3.0.2-bin-hadoop3.2/R/
spark-3.0.2-bin-hadoop3.2/R/lib/
spark-3.0.2-bin-hadoop3.2/R/lib/sparkr.zip
spark-3.0.2-bin-hadoop3.2/R/lib/SparkR/
spark-3.0.2-bin-hadoop3.2/R/lib/SparkR/worker/
spark-3.0.2-bin-hadoop3.2/R/lib/SparkR/worker/worker.R
spark-3.0.2-bin-hadoop3.2/R/lib/SparkR/worker/daemon.R
spark-3.0.2-bin-hadoop3.2/R/lib/SparkR/tests/
spark-3.0.2-bin-hadoop3.2/R/lib/SparkR/tests/testthat/
spark-3.0.2-bin-hadoop3.2/R/lib/SparkR/
```

(Refer Slide Time: 13:21)

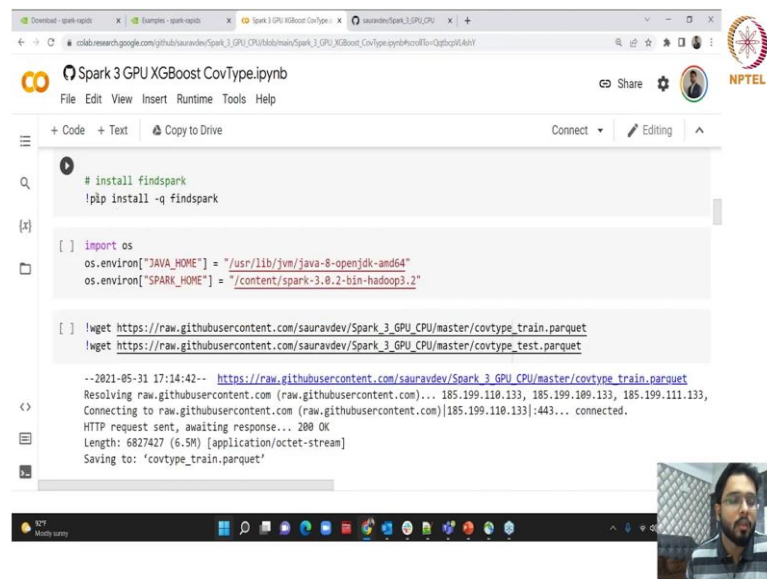


The screenshot shows a Jupyter Notebook interface with the title 'Spark 3 GPU XGBoost CovType.ipynb'. The notebook is open to a cell containing a list of files in the Spark 3 GPU XGBoost CovType directory. The files listed are:

- spark-3.0.2-bin-hadoop3.2/python/pyspark/ml/tests/test_understanding_summary.py
- spark-3.0.2-bin-hadoop3.2/python/pyspark/ml/tests/test_stat.py
- spark-3.0.2-bin-hadoop3.2/python/pyspark/ml/tests/test_pipeline.py
- spark-3.0.2-bin-hadoop3.2/python/pyspark/ml/tests/test_persistence.py
- spark-3.0.2-bin-hadoop3.2/python/pyspark/ml/tests/test_param.py
- spark-3.0.2-bin-hadoop3.2/python/pyspark/ml/tests/test_linalg.py
- spark-3.0.2-bin-hadoop3.2/python/pyspark/ml/tests/test_image.py
- spark-3.0.2-bin-hadoop3.2/python/pyspark/ml/tests/test_feature.py
- spark-3.0.2-bin-hadoop3.2/python/pyspark/ml/tests/test_evaluation.py
- spark-3.0.2-bin-hadoop3.2/python/pyspark/ml/tests/test_base.py
- spark-3.0.2-bin-hadoop3.2/python/pyspark/ml/tests/test_algorithms.py
- spark-3.0.2-bin-hadoop3.2/python/pyspark/ml/tests/test_util.py
- spark-3.0.2-bin-hadoop3.2/python/pyspark/ml/tests/__init__.py
- spark-3.0.2-bin-hadoop3.2/python/pyspark/ml/tests/test_wrapper.py
- spark-3.0.2-bin-hadoop3.2/python/pyspark/ml/param/
- spark-3.0.2-bin-hadoop3.2/python/pyspark/ml/param/shared_params.py
- spark-3.0.2-bin-hadoop3.2/python/pyspark/ml/param/shared_params_code_gen.py
- spark-3.0.2-bin-hadoop3.2/python/pyspark/ml/param/__init__.py
- spark-3.0.2-bin-hadoop3.2/python/pyspark/ml/linalg/
- spark-3.0.2-bin-hadoop3.2/python/pyspark/ml/linalg/__init__.py
- spark-3.0.2-bin-hadoop3.2/python/pyspark/ml/wrapper.py

Then, I am downloading unpacking the spark 3; open source for this.

(Refer Slide Time: 13:24)



The screenshot shows a Jupyter Notebook interface with the title 'Spark 3 GPU XGBoost CovType.ipynb'. The notebook is open to a cell containing the following code:

```
# install findspark
!pip install -q findspark

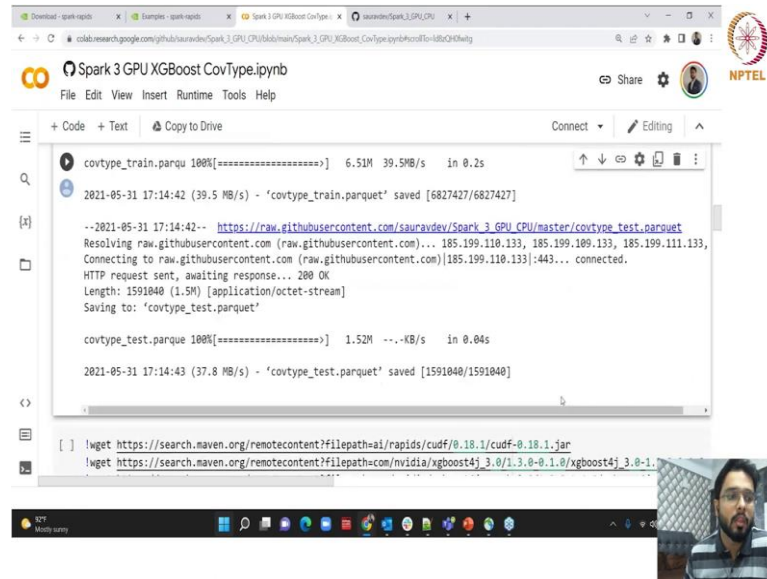
[ ] import os
os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-8-openjdk-amd64"
os.environ["SPARK_HOME"] = "/content/spark-3.0.2-bin-hadoop3.2"

[ ] !wget https://raw.githubusercontent.com/sauravdev/Spark_3_GPU_CPU/master/covtype_train.parquet
!wget https://raw.githubusercontent.com/sauravdev/Spark_3_GPU_CPU/master/covtype_test.parquet

--2021-05-31 17:14:42-- https://raw.githubusercontent.com/sauravdev/Spark_3_GPU_CPU/master/covtype_train.parquet
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.110.133, 185.199.109.133, 185.199.111.133,
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)[185.199.110.133]:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 6827427 (6.5M) [application/octet-stream]
Saving to: 'covtype_train.parquet'
```

Then, I am installing some library called find spark, where which will be able to use and you know initialize spark; setting the JAVA HOME, SPARK HOME, downloading that data the forest cover data in parquet format, then setting up the libraries.

(Refer Slide Time: 13:45)



The screenshot shows a Jupyter Notebook titled "Spark 3 GPU XGBoost CovType.ipynb". The code cell contains the following commands and output:

```
covtype_train.parqu 100%[=====] 6.51M 39.5MB/s in 0.2s
2021-05-31 17:14:42 (39.5 MB/s) - 'covtype_train.parquet' saved [6827427/6827427]

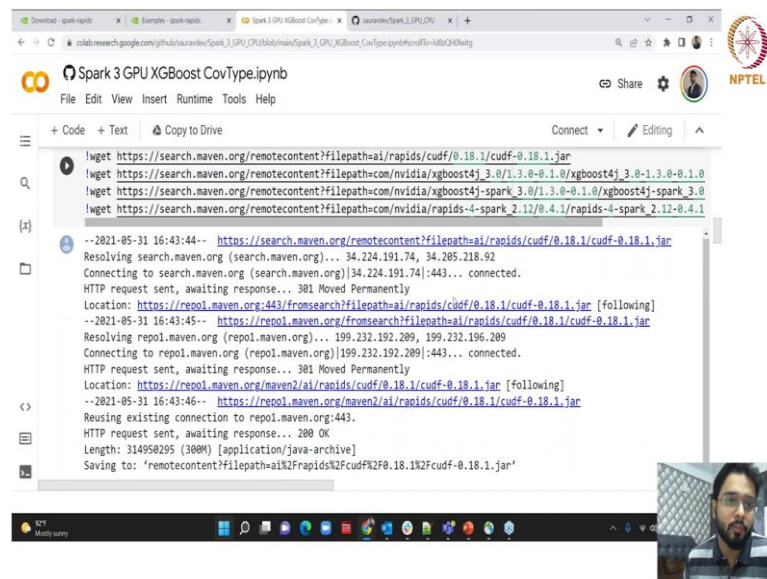
--2021-05-31 17:14:42-- https://raw.githubusercontent.com/sauravdev/Spark_3_GPU_CPU/master/covtype_test.parquet
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.110.133, 185.199.109.133, 185.199.111.133,
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)[185.199.110.133]:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1591040 (1.5M) [application/octet-stream]
Saving to: 'covtype_test.parquet'

covtype_test.parque 100%[=====] 1.52M --.-KB/s in 0.04s
2021-05-31 17:14:43 (37.8 MB/s) - 'covtype_test.parquet' saved [1591040/1591040]

[ ] !wget https://search.maven.org/remotecontent?filepath=ai/rapids/cudf/0.18.1/cudf-0.18.1.jar
!wget https://search.maven.org/remotecontent?filepath=com/nvidia/xgboost4j_3.0/1.3.0-0.1.0/xgboost4j_3.0-1.3.0-0.1.0.jar
```

The output shows the successful saving of two parquet files and the execution of two wget commands to download jars from Maven repositories.

(Refer Slide Time: 13:47)



The screenshot shows a Jupyter Notebook titled "Spark 3 GPU XGBoost CovType.ipynb". The code cell contains the following commands and output:

```
!wget https://search.maven.org/remotecontent?filepath=ai/rapids/cudf/0.18.1/cudf-0.18.1.jar
!wget https://search.maven.org/remotecontent?filepath=com/nvidia/xgboost4j_3.0/1.3.0-0.1.0/xgboost4j_3.0-1.3.0-0.1.0.jar
!wget https://search.maven.org/remotecontent?filepath=com/nvidia/xgboost4j-spark_3.0/1.3.0-0.1.0/xgboost4j-spark_3.0-1.3.0-0.1.0.jar
!wget https://search.maven.org/remotecontent?filepath=com/nvidia/rapids-4-spark_2.12/0.4.1/rapids-4-spark_2.12-0.4.1.jar

--2021-05-31 16:43:44-- https://search.maven.org/remotecontent?filepath=ai/rapids/cudf/0.18.1/cudf-0.18.1.jar
Resolving search.maven.org (search.maven.org)... 34.224.191.74, 34.205.218.92
Connecting to search.maven.org (search.maven.org)[34.224.191.74]:443... connected.
HTTP request sent, awaiting response... 301 Moved Permanently
Location: https://repo1.maven.org/443/fromsearch?filepath=ai/rapids/cudf/0.18.1/cudf-0.18.1.jar [following]
--2021-05-31 16:43:45-- https://repo1.maven.org/443/fromsearch?filepath=ai/rapids/cudf/0.18.1/cudf-0.18.1.jar
Resolving repo1.maven.org (repo1.maven.org)... 199.232.192.209, 199.232.196.209
Connecting to repo1.maven.org (repo1.maven.org)[199.232.192.209]:443... connected.
HTTP request sent, awaiting response... 301 Moved Permanently
Location: https://repo1.maven.org/maven2/ai/rapids/cudf/0.18.1/cudf-0.18.1.jar [following]
--2021-05-31 16:43:46-- https://repo1.maven.org/maven2/ai/rapids/cudf/0.18.1/cudf-0.18.1.jar
Reusing existing connection to repo1.maven.org:443.
HTTP request sent, awaiting response... 200 OK
Length: 314950295 (300M) [application/java-archive]
Saving to: 'remotecontent?filepath=ai%2Frapids%2Fcudf%2F0.18.1%2Fcudf-0.18.1.jar'
```

The output shows the successful downloading of four jars from Maven repositories using wget.

So, we need the cuDF library, we need the RAPIDS Spark library, the jars which I was talking about, we need the XGBoost because we are running XGBoost model for GPU. So, all those jars will be downloaded from the respective maven repositories into the local.

(Refer Slide Time: 14:04)



The screenshot shows a Windows desktop with a taskbar at the bottom containing various application icons. A web browser window is open, displaying the NVIDIA website. The address bar shows the URL: <https://www.nvidia.com/en-us/ai/edge-computing/>. The page content includes the NVIDIA logo, a navigation menu, and a main heading "AI Edge Computing". Below this, there is a section titled "AI Edge Computing" with a sub-heading "AI Edge Computing" and a paragraph of text. A terminal window is open in the foreground, displaying the output of a command. The command is: `curl -O https://nvidia.github.io/spark_rtd/downloads/linux/spark_3_GPU_CU113b/main/spark_3_GPU_XGBoost_CovType-synchrbl-to-lidsCvHtg`. The output shows the file being downloaded and its size (3.06 MB/s). The terminal window also shows the file being saved to a local directory.

(Refer Slide Time: 14:07)



The screenshot shows a Jupyter Notebook interface in a web browser. The browser's address bar displays the URL: https://colab.research.google.com/github/haurendev/Spark_3_GPU_XGBoost_CovType.ipynb/blob/main/Spark_3_GPU_XGBoost_CovType.ipynb?colab=1&hdp=40170476. The notebook title is "Spark 3 GPU XGBoost CovType.ipynb". The interface includes a top bar with "File", "Edit", "View", "Insert", "Runtime", "Tools", and "Help" menus. A "Share" button and a user profile icon are also present. The notebook content is divided into two main sections: "Code" and "Output". The "Code" section contains the following Python code:

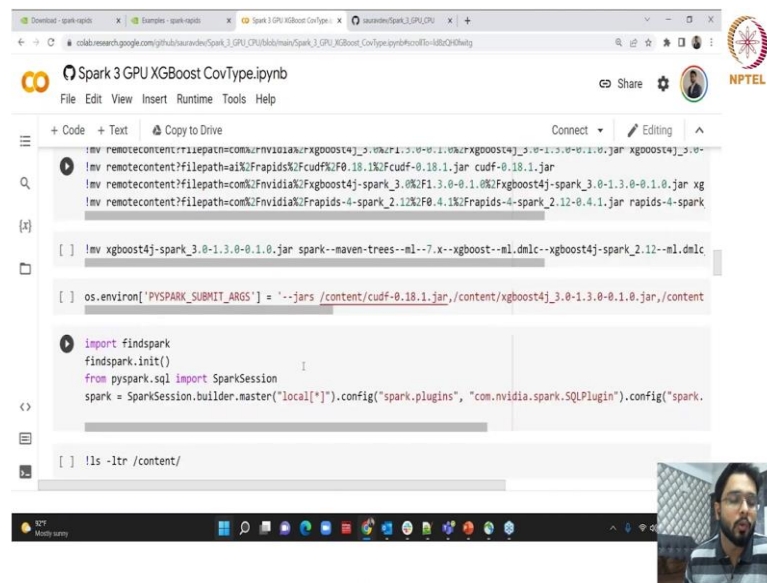
```
[ ] import findspark
findspark.init()
from pyspark.sql import SparkSession
spark = SparkSession.builder.master("local[*]").config("spark.plugins", "com.nvidia.spark.SQLPlugin").config("spark.
```

The "Output" section shows the execution of the code, displaying the Spark version and the configuration of XGBoost:

```
total 1066180
-rw-r--r-- 1 root root 534157371 Jan 15 04:58 xgboost4j_3.0-1.3.0-0.1.0.jar
-rw-r--r-- 1 root root 4979676 Jan 15 04:59 spark-maven-trees-ml-7.7.0-xgboost-ml.dmlc-xgboost4j-spark-2.12-m
dnuox-xr-x 13 1000 1000 4096 Feb 16 06:32 spark-3.0.2-bin-hadoop3.2
-rw-r--r-- 1 root root 224619134 Feb 16 06:33 spark-3.0.2-bin-hadoop3.2.tgz
-rw-r--r-- 1 root root 314950295 Mar 16 03:23 cudf-0.18.1.jar
-rw-r--r-- 1 root root 9117914 Mar 23 11:07 rapids-4-spark-2.12-0.4.1.jar
dnuox-xr-x 1 root root 4096 May 6 13:44 sample_data
-rw-r--r-- 1 root root 6827427 May 31 16:43 covtype_train.parquet
-rw-r--r-- 1 root root 1591048 May 31 16:43 covtype_test.parquet
```

The bottom of the screen shows a Windows taskbar with various application icons and a system clock indicating 12:47 on Monday, January 15, 2024. A small inset image in the bottom right corner shows a person's face, likely a video call participant.

(Refer Slide Time: 14:08)



```
!mv remotecontent:r11eapn=com.nvidia.spark.xgboost4j_3.0-1.3.0-0.1.0.jar xgboost4j_3.0-1.3.0-0.1.0.jar
!mv remotecontent?filepath=a1%Frapids%FcuF%F0.18.1%FcuF-0.18.1.jar cudf-0.18.1.jar
!mv remotecontent?filepath=com%Nvidia%Fgboost4j-spark_3.0%F1.3.0-0.1.0%Fgboost4j-spark_3.0-1.3.0-0.1.0.jar xg
!mv remotecontent?filepath=com%Nvidia%Frapids-4-spark_2.12%F0.4.1%Frapids-4-spark_2.12-0.4.1.jar rapids-4-spark

[ ] !mv xgboost4j-spark_3.0-1.3.0-0.1.0.jar spark--maven-trees--ml--7.x--xgboost--ml.dmlc--xgboost4j-spark_2.12--ml.dmlc

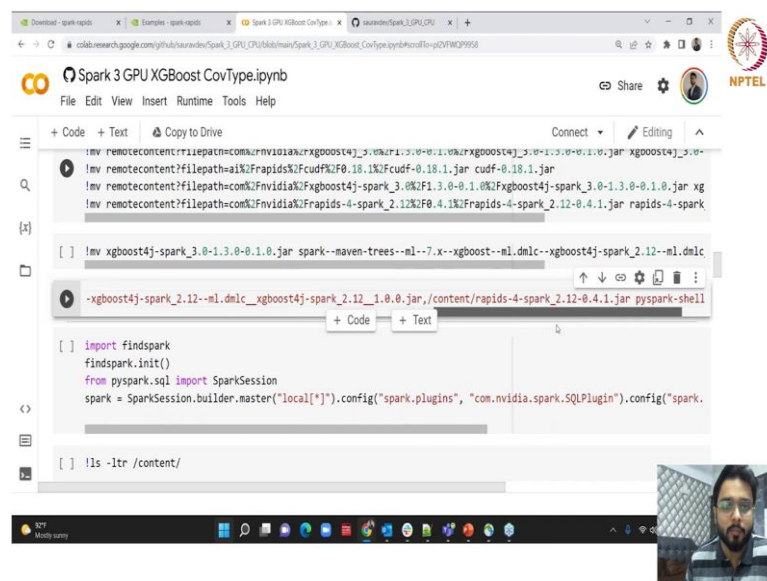
[ ] os.environ['PYSPARK_SUBMIT_ARGS'] = '--jars /content/cudf-0.18.1.jar,/content/xgboost4j-spark_3.0-1.3.0-0.1.0.jar,/content

import findspark
findspark.init()
from pyspark.sql import SparkSession
spark = SparkSession.builder.master("local[*]").config("spark.plugins", "com.nvidia.spark.SQLPlugin").config("spark.

[ ] !ls -ltr /content/
```

And then, we can move it to the respective location.

(Refer Slide Time: 14:11)



```
!mv xgboost4j-spark_3.0-1.3.0-0.1.0.jar spark--maven-trees--ml--7.x--xgboost--ml.dmlc--xgboost4j-spark_2.12--ml.dmlc

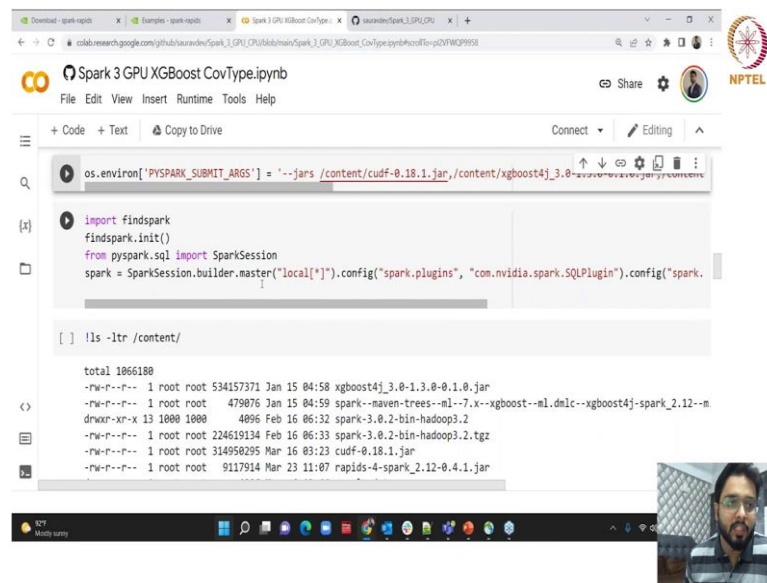
-xgboost4j-spark_2.12--ml.dmlc_xgboost4j-spark_2.12_1.0.0.jar,/content/rapids-4-spark_2.12-0.4.1.jar pyspark-shell

[ ] import findspark
findspark.init()
from pyspark.sql import SparkSession
spark = SparkSession.builder.master("local[*]").config("spark.plugins", "com.nvidia.spark.SQLPlugin").config("spark.

[ ] !ls -ltr /content/
```

And then, set up the form like set up the jars in the PYSPARK shell. So, that we are able to execute spark code now.

(Refer Slide Time: 14:23)



```
os.environ['PYSPARK_SUBMIT_ARGS'] = '--jars /content/cudf-0.18.1.jar,/content/xgboost4j_3.0-1.3.0-0.1.0.jar

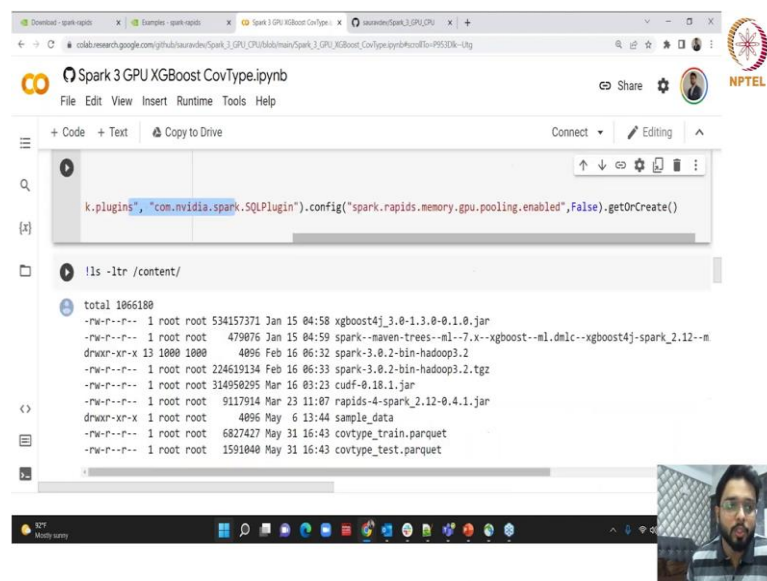
import findspark
findspark.init()
from pyspark.sql import SparkSession
spark = SparkSession.builder.master("local[*]").config("spark.plugins", "com.nvidia.spark.SQLPlugin").config("spark.

[ ] !ls -ltr /content/

total 1066180
-rw-r--r-- 1 root root 534157371 Jan 15 04:58 xgboost4j_3.0-1.3.0-0.1.0.jar
-rw-r--r-- 1 root root 479076 Jan 15 04:59 spark-maven-trees-m1-7.x-xgboost-ml.dmlc-xgboost4j-spark_2.12--m
drwxr-xr-x 13 1000 1000 4096 Feb 16 06:32 spark-3.0.2-bin-hadoop3.2
-rw-r--r-- 1 root root 224619134 Feb 16 06:33 spark-3.0.2-bin-hadoop3.2.tgz
-rw-r--r-- 1 root root 314958295 Mar 16 03:23 cudf-0.18.1.jar
-rw-r--r-- 1 root root 9117914 Mar 23 11:07 rapids-4-spark_2.12-0.4.1.jar
```

So, we can configure the local; that means, it will use the only one node cluster.

(Refer Slide Time: 14:28)



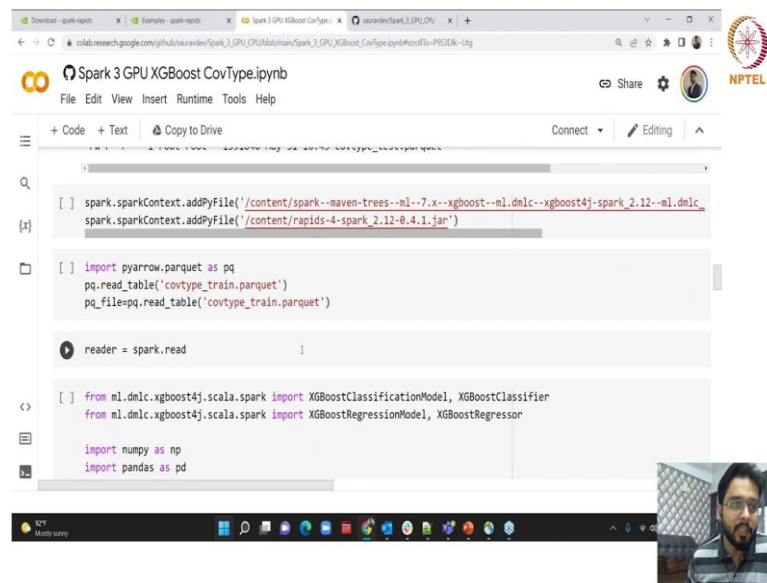
```
k.plugins", "com.nvidia.spark.SQLPlugin").config("spark.rapids.memory.gpu.pooling.enabled", False).getOrCreate()

!ls -ltr /content/

total 1066180
-rw-r--r-- 1 root root 534157371 Jan 15 04:58 xgboost4j_3.0-1.3.0-0.1.0.jar
-rw-r--r-- 1 root root 479076 Jan 15 04:59 spark-maven-trees-m1-7.x-xgboost-ml.dmlc-xgboost4j-spark_2.12--m
drwxr-xr-x 13 1000 1000 4096 Feb 16 06:32 spark-3.0.2-bin-hadoop3.2
-rw-r--r-- 1 root root 224619134 Feb 16 06:33 spark-3.0.2-bin-hadoop3.2.tgz
-rw-r--r-- 1 root root 314958295 Mar 16 03:23 cudf-0.18.1.jar
-rw-r--r-- 1 root root 9117914 Mar 23 11:07 rapids-4-spark_2.12-0.4.1.jar
drwxr-xr-x 1 root root 4096 May 6 13:44 sample_data
-rw-r--r-- 1 root root 6827427 May 31 16:43 covtype_train.parquet
-rw-r--r-- 1 root root 1591040 May 31 16:43 covtype_test.parquet
```

This is the configuration for enabling the spark rapid GPU plugin and we do not want GPU pooling because we only have single GPU of now. So, I am disabled

(Refer Slide Time: 14:40)



```
[ ] spark.sparkContext.addPyFile('/content/spark--maven-trees--ml--7.x--xgboost--ml.dmlc--xgboost4j-spark_2.12--ml.dmlc')
spark.sparkContext.addPyFile('/content/rapids-4-spark_2.12-0.4.1.jar')

[ ] import pyarrow.parquet as pq
pq.read_table('covtype_train.parquet')
pq_file=pq.read_table('covtype_train.parquet')

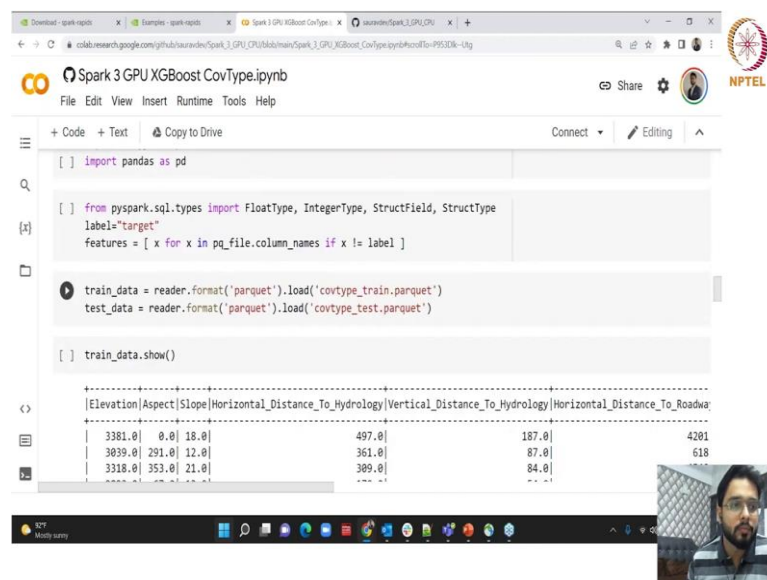
reader = spark.read

[ ] from ml.dmlc.xgboost4j.scala.spark import XGBoostClassificationModel, XGBoostClassifier
from ml.dmlc.xgboost4j.scala.spark import XGBoostRegressionModel, XGBoostRegressor

import numpy as np
import pandas as pd
```

Then, we can add all the machine learning XGBoost jars reading the data; creating data spark data frame; training and testing data.

(Refer Slide Time: 14:52)



```
[ ] import pandas as pd

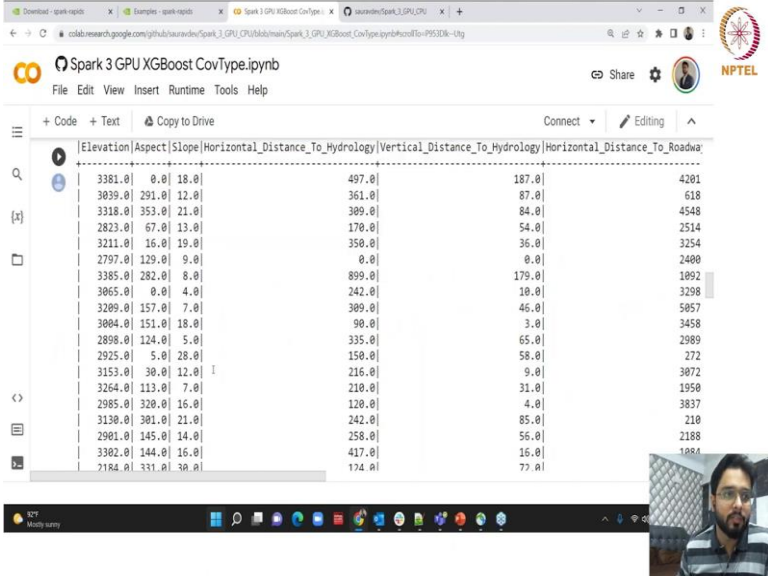
[ ] from pyspark.sql.types import FloatType, IntegerType, StructField, StructType
label="target"
features = [ x for x in pq_file.column_names if x != label ]

train_data = reader.format('parquet').load('covtype_train.parquet')
test_data = reader.format('parquet').load('covtype_test.parquet')

[ ] train_data.show()
```

Elevation	Aspect	Slope	Horizontal_Distance_To_Hydrology	Vertical_Distance_To_Hydrology	Horizontal_Distance_To_Roadway
3381.0	0.0	18.0	497.0	187.0	4281
3039.0	291.0	12.0	361.0	87.0	618
3318.0	353.0	21.0	309.0	84.0	

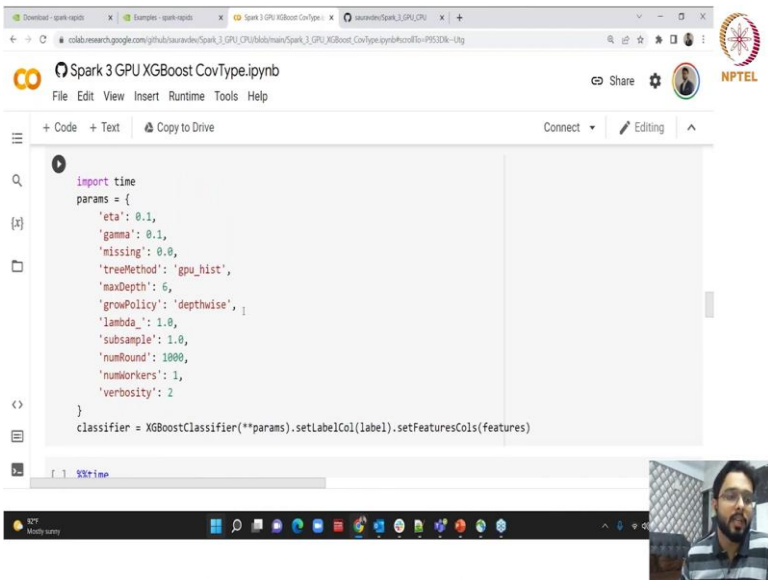
(Refer Slide Time: 14:55)



Elevation	Aspect	Slope	Horizontal_Distance_To_Hydrology	Vertical_Distance_To_Hydrology	Horizontal_Distance_To_Roadways	
3381.0	0.0	18.0	497.0	187.0		4201
3039.0	291.0	12.0	361.0	87.0		618
3318.0	353.0	21.0	309.0	84.0		4548
2823.0	67.0	13.0	170.0	54.0		2514
3211.0	16.0	19.0	350.0	36.0		3254
2797.0	129.0	9.0	0.0	0.0		2400
3385.0	282.0	8.0	899.0	179.0		1092
3065.0	0.0	4.0	242.0	10.0		3298
3209.0	157.0	7.0	309.0	46.0		5057
3004.0	151.0	18.0	90.0	3.0		3458
2898.0	124.0	5.0	335.0	65.0		2989
2925.0	5.0	28.0	150.0	58.0		272
3153.0	30.0	12.0	216.0	9.0		3072
3264.0	113.0	7.0	210.0	31.0		1950
2985.0	320.0	16.0	120.0	4.0		3837
3130.0	301.0	21.0	242.0	85.0		210
2901.0	145.0	14.0	258.0	56.0		2188
3302.0	144.0	16.0	417.0	16.0		1084
2184.0	331.0	30.0	174.0	77.0		

So, this is the data frame forest tree cover. So, where we have elevation, aspects, slope, horizontal distance, to hydrology, vertical distance, horizontal distance, to roadways and so on and so forth.

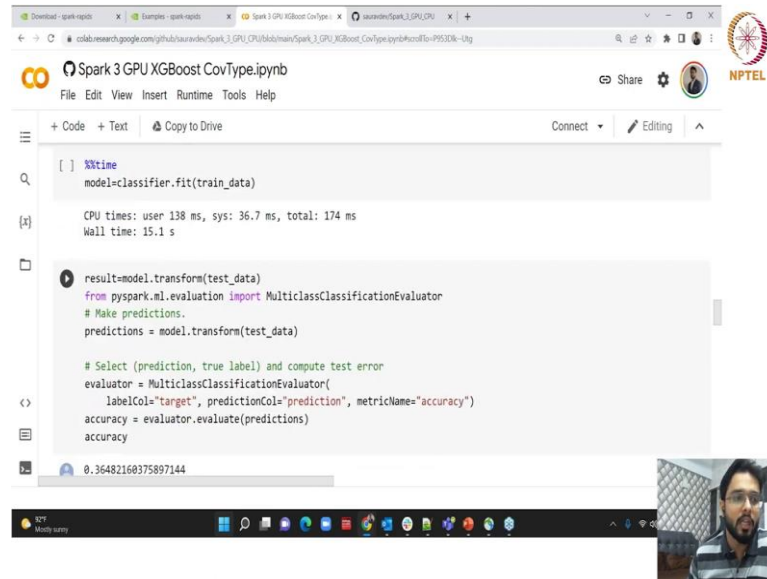
(Refer Slide Time: 15:10)



```
import time
params = {
    'eta': 0.1,
    'gamma': 0.1,
    'missing': 0.0,
    'treeMethod': 'gpu_hist',
    'maxDepth': 6,
    'growPolicy': 'depthwise',
    'lambda': 1.0,
    'subsample': 1.0,
    'numRound': 1000,
    'numWorkers': 1,
    'verbosity': 2
}
classifier = XGBoostClassifier(**params).setLabelCol(label).setFeaturesCols(features)
```

So, we are using this data.

(Refer Slide Time: 15:17)



```
model=classifier.fit(train_data)

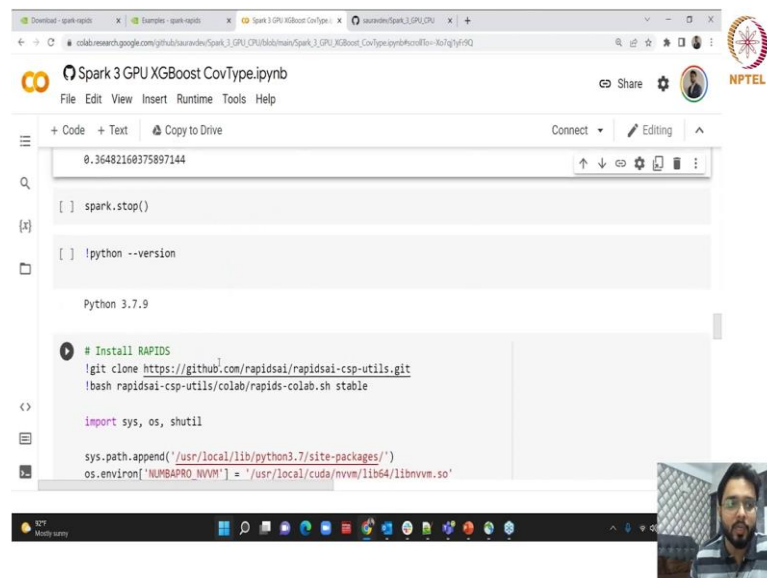
CPU times: user 138 ms, sys: 36.7 ms, total: 174 ms
Wall time: 15.1 s

result=model.transform(test_data)
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
# Make predictions.
predictions = model.transform(test_data)

# Select (prediction, true label) and compute test error
evaluator = MulticlassClassificationEvaluator(
    labelCol="target", predictionCol="prediction", metricName="accuracy")
accuracy = evaluator.evaluate(predictions)
accuracy
```

Passing the parameters, classifier, setting the label, feature columns, classifier.fit train the data and then, making the predictions.

(Refer Slide Time: 15:23)



```
spark.stop()

!python --version

Python 3.7.9

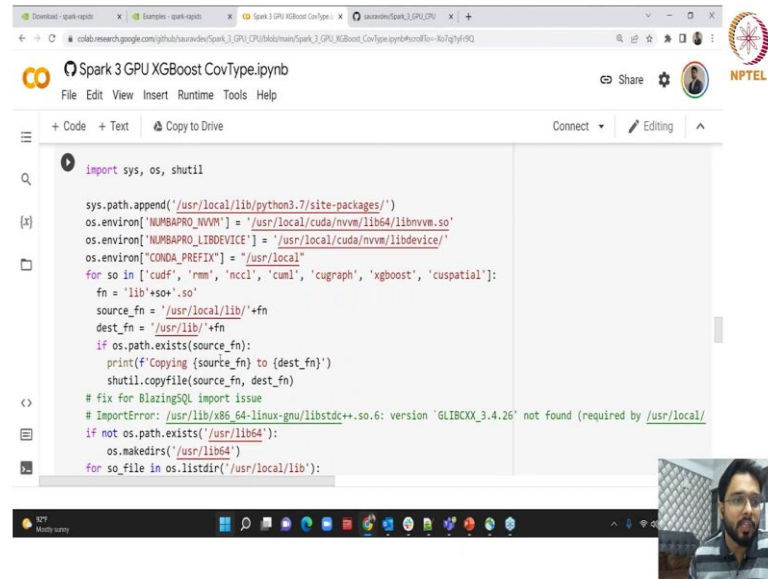
# Install RAPIDS
!git clone https://github.com/rapidsai/rapidsai-csp-utils.git
!bash rapidsai-csp-utils/colab/rapids-colab.sh stable

import sys, os, shutil

sys.path.append('/usr/local/lib/python3.7/site-packages/')
os.environ['NUMBAPRO_NVVM'] = '/usr/local/cuda/nvvm/lib64/libnvvm.so'
```

Ah Selecting the and computing the test error; so, here we are using the ml library basically multi classed evaluation for understanding the accuracy, though the accuracy is bad; but definitely, it will help you to understand the flow at least.

(Refer Slide Time: 15:42)

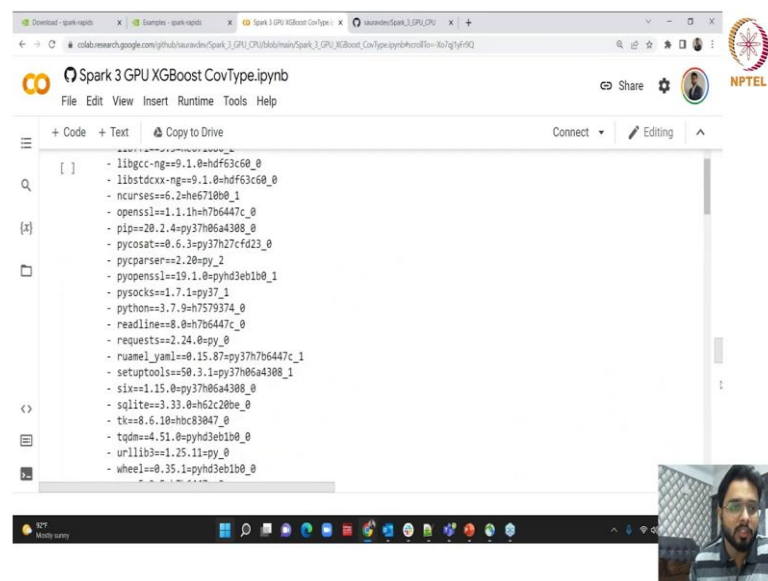


```
import sys, os, shutil

sys.path.append('/usr/local/lib/python3.7/site-packages/')
os.environ['MUMBAI_PRO_NVVM'] = '/usr/local/cuda/nvvm/lib64/libnvvm.so'
os.environ['MUMBAI_PRO_LIBDEVICE'] = '/usr/local/cuda/nvvm/libdevice/'
os.environ['CONDA_PREFIX'] = '/usr/local'
for so in ['cudf', 'rmm', 'nccl', 'cuml', 'cugraph', 'xgboost', 'cuspatial']:
    fn = 'lib'+so+'.so'
    source_fn = '/usr/local/lib/'+fn
    dest_fn = '/usr/lib/'+fn
    if os.path.exists(source_fn):
        print(f'Copying {source_fn} to {dest_fn}')
        shutil.copyfile(source_fn, dest_fn)

# Fix for BlazingSQL import issue
# ImportError: /usr/lib/x86_64-linux-gnu/libstdc++.so.6: version 'GLIBCXX_3.4.26' not found (required by /usr/local/
if not os.path.exists('/usr/lib64'):
    os.makedirs('/usr/lib64')
for so_file in os.listdir('/usr/local/lib'):
```

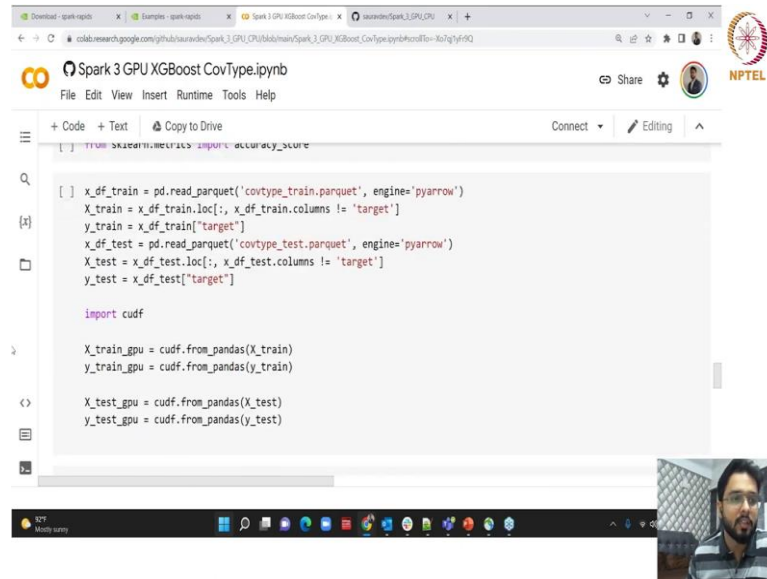
(Refer Slide Time: 15:46)



```
[ ]
- libgcc-ng==9.1.0-hdf63c60_0
- libstdc++-ng==9.1.0-hdf63c60_0
- ncurses==6.2-he671080_1
- openssl==1.1.1h7b6447c_0
- pip==20.2.4-py37h06a4308_0
- pycosat==0.6.3-py37h27cf23_0
- pycparser==2.20-py_2
- pyopenssl==19.1.0-pyhd3eb1b0_1
- pysocks==1.7.1-py37_1
- python==3.7.9h7579374_0
- readline==8.0h7b6447c_0
- requests==2.24.0-py_0
- ruamel-yaml==0.15.87-py37h7b6447c_1
- setuptools==50.3.1-py37h06a4308_1
- six==1.15.0-py37h06a4308_0
- sqlite3==3.33.0-h52c20be_0
- tk==8.6.10-hbc83847_0
- tqdm==4.51.0-pyhd3eb1b0_0
- urllib3==1.25.11-py_0
- wheel==0.35.1-pyhd3eb1b0_0
```

So, the same thing, we are doing for CPU versus GPU and we are finding ok for CPU versus GPU training and testing data, converting from pandas to GPU format, running by XGB format.

(Refer Slide Time: 15:54)



The screenshot shows a Jupyter Notebook titled "Spark 3 GPU XGBoost CovType.ipynb". The code in the notebook is as follows:

```
from sklearn.metrics import accuracy_score

[ ] x_df_train = pd.read_parquet('covtype_train.parquet', engine='pyarrow')
    X_train = x_df_train.loc[:, x_df_train.columns != 'target']
    y_train = x_df_train['target']
    x_df_test = pd.read_parquet('covtype_test.parquet', engine='pyarrow')
    X_test = x_df_test.loc[:, x_df_test.columns != 'target']
    y_test = x_df_test['target']

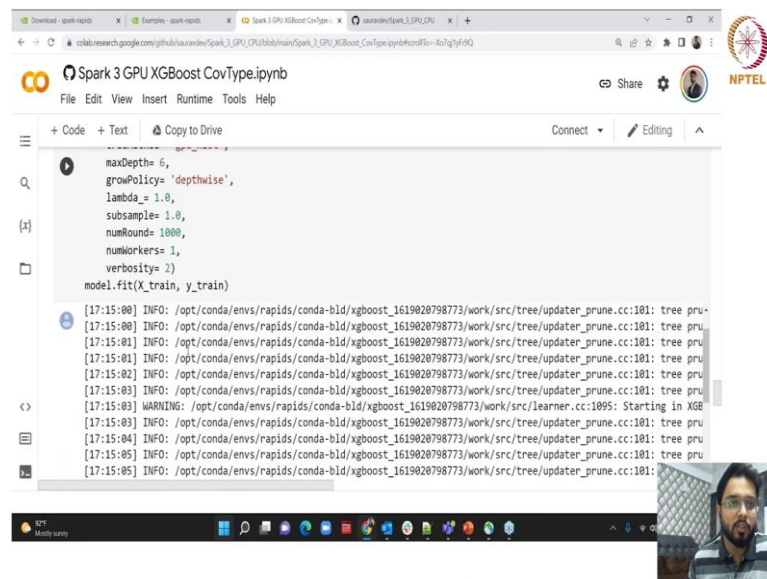
import cudf

X_train_gpu = cudf.from_pandas(X_train)
y_train_gpu = cudf.from_pandas(y_train)

X_test_gpu = cudf.from_pandas(X_test)
y_test_gpu = cudf.from_pandas(y_test)
```

The notebook interface includes a top bar with "File", "Edit", "View", "Insert", "Runtime", "Tools", and "Help" menus. A "Share" button and a user profile icon are also visible. The bottom of the notebook shows a terminal window with the command "STF" and the output "Moody sunny". A small video feed of a person is visible in the bottom right corner.

(Refer Slide Time: 16:01)



The screenshot shows the same Jupyter Notebook interface, but with different code and output. The code is:

```
maxDepth= 6,
growPolicy= 'depthwise',
lambda_ = 1.0,
subsample= 1.0,
numRound= 1000,
numWorkers= 1,
verbosity= 2)
model.fit(X_train, y_train)
```

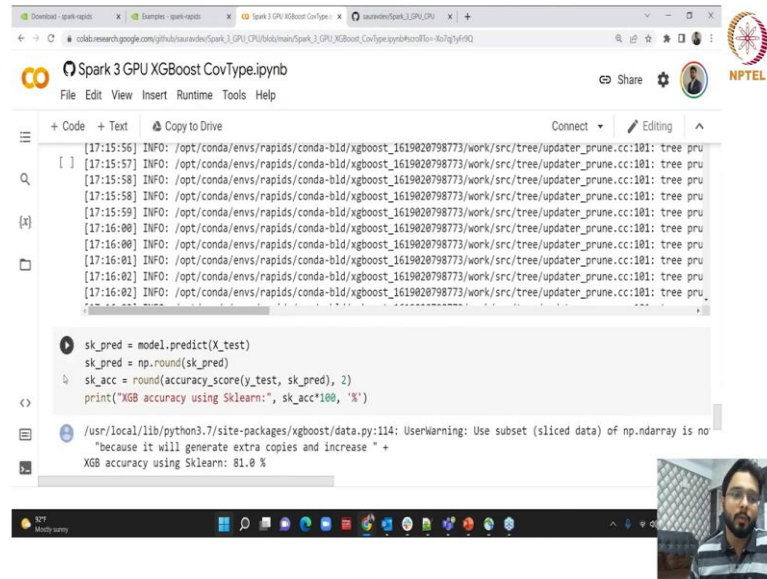
The output of the code is a series of log messages from XGBoost, including:

```
[17:15:00] INFO: /opt/conda/envs/rapids/conda-bld/xgboost_1619020798773/work/src/tree/updater_prune.cc:101: tree prun
[17:15:00] INFO: /opt/conda/envs/rapids/conda-bld/xgboost_1619020798773/work/src/tree/updater_prune.cc:101: tree prun
[17:15:01] INFO: /opt/conda/envs/rapids/conda-bld/xgboost_1619020798773/work/src/tree/updater_prune.cc:101: tree prun
[17:15:01] INFO: /opt/conda/envs/rapids/conda-bld/xgboost_1619020798773/work/src/tree/updater_prune.cc:101: tree prun
[17:15:02] INFO: /opt/conda/envs/rapids/conda-bld/xgboost_1619020798773/work/src/tree/updater_prune.cc:101: tree prun
[17:15:03] INFO: /opt/conda/envs/rapids/conda-bld/xgboost_1619020798773/work/src/tree/updater_prune.cc:101: tree prun
[17:15:03] WARNING: /opt/conda/envs/rapids/conda-bld/xgboost_1619020798773/work/src/learner.cc:1095: Starting in XGB
[17:15:04] INFO: /opt/conda/envs/rapids/conda-bld/xgboost_1619020798773/work/src/tree/updater_prune.cc:101: tree prun
[17:15:05] INFO: /opt/conda/envs/rapids/conda-bld/xgboost_1619020798773/work/src/tree/updater_prune.cc:101: tree prun
[17:15:05] INFO: /opt/conda/envs/rapids/conda-bld/xgboost_1619020798773/work/src/tree/updater_prune.cc:101: tree prun
```

The notebook interface is the same as in the previous screenshot, with the same top bar and bottom terminal window. The video feed of the person is also present in the bottom right corner.

So, we are comparing against a normal cudf as well.

(Refer Slide Time: 16:06)



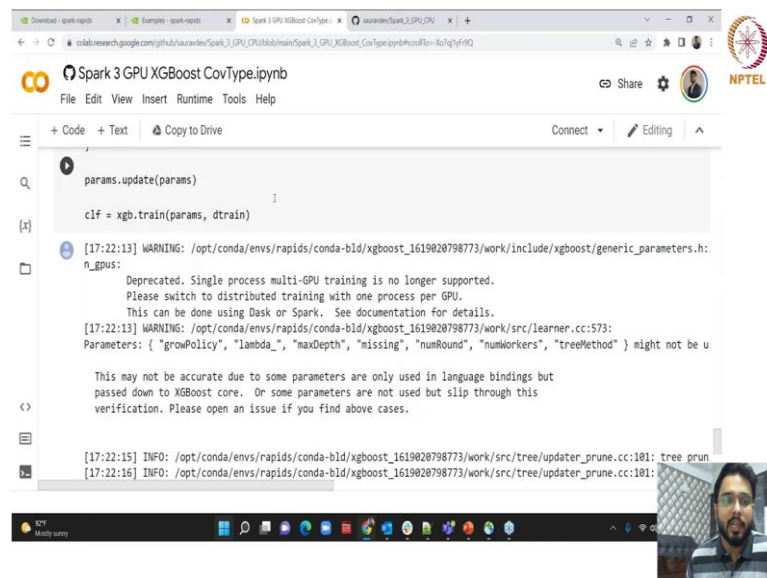
```
[17:15:56] INFO: /opt/conda/envs/rapids/contrib-xgboost_1619020798773/work/src/tree/updater_prune.cc:181: tree pruned
[17:15:57] INFO: /opt/conda/envs/rapids/contrib-xgboost_1619020798773/work/src/tree/updater_prune.cc:181: tree pruned
[17:15:58] INFO: /opt/conda/envs/rapids/contrib-xgboost_1619020798773/work/src/tree/updater_prune.cc:181: tree pruned
[17:15:59] INFO: /opt/conda/envs/rapids/contrib-xgboost_1619020798773/work/src/tree/updater_prune.cc:181: tree pruned
[17:16:00] INFO: /opt/conda/envs/rapids/contrib-xgboost_1619020798773/work/src/tree/updater_prune.cc:181: tree pruned
[17:16:01] INFO: /opt/conda/envs/rapids/contrib-xgboost_1619020798773/work/src/tree/updater_prune.cc:181: tree pruned
[17:16:02] INFO: /opt/conda/envs/rapids/contrib-xgboost_1619020798773/work/src/tree/updater_prune.cc:181: tree pruned
[17:16:02] INFO: /opt/conda/envs/rapids/contrib-xgboost_1619020798773/work/src/tree/updater_prune.cc:181: tree pruned
[17:16:02] INFO: /opt/conda/envs/rapids/contrib-xgboost_1619020798773/work/src/tree/updater_prune.cc:181: tree pruned
[17:16:02] INFO: /opt/conda/envs/rapids/contrib-xgboost_1619020798773/work/src/tree/updater_prune.cc:181: tree pruned

sk_pred = model.predict(X_test)
sk_pred = np.round(sk_pred)
sk_acc = round(accuracy_score(y_test, sk_pred), 2)
print("XGB accuracy using Sklearn:", sk_acc*100, '%')

/usr/local/lib/python3.7/site-packages/xgboost/data.py:114: UserWarning: Use subset (sliced data) of np.ndarray is not
"because it will generate extra copies and increase " +
XGB accuracy using Sklearn: 81.8 %
```

So, cuDF might have heard about that this alternative to pandas on GPU. So, even spark gets faster than cuDF.

(Refer Slide Time: 16:20)



```
params.update(params)
clf = xgb.train(params, dtrain)

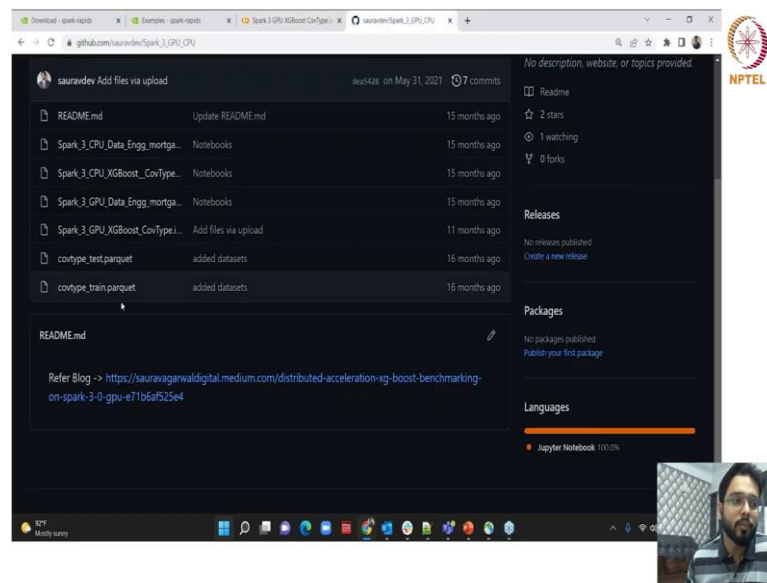
[17:22:13] WARNING: /opt/conda/envs/rapids/contrib-xgboost_1619020798773/work/include/xgboost/generic_parameters.h:
n_gpus:
Deprecated. Single process multi-GPU training is no longer supported.
Please switch to distributed training with one process per GPU.
This can be done using Dask or Spark. See documentation for details.
[17:22:13] WARNING: /opt/conda/envs/rapids/contrib-xgboost_1619020798773/work/src/learner.cc:573:
Parameters: { "growPolicy", "lambda_", "maxDepth", "missing", "numRound", "numWorkers", "treeMethod" } might not be u

This may not be accurate due to some parameters are only used in language bindings but
passed down to XGBoost core. Or some parameters are not used but slip through this
verification. Please open an issue if you find above cases.

[17:22:15] INFO: /opt/conda/envs/rapids/contrib-xgboost_1619020798773/work/src/tree/updater_prune.cc:181: tree pruned
[17:22:16] INFO: /opt/conda/envs/rapids/contrib-xgboost_1619020798773/work/src/tree/updater_prune.cc:181: tree pruned
```

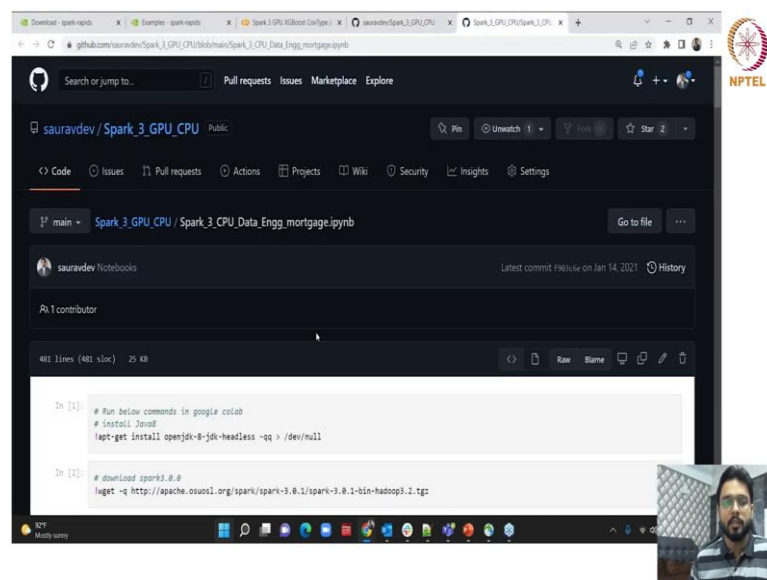
So, this is a simple classification kind of data set. ah

(Refer Slide Time: 16:25)

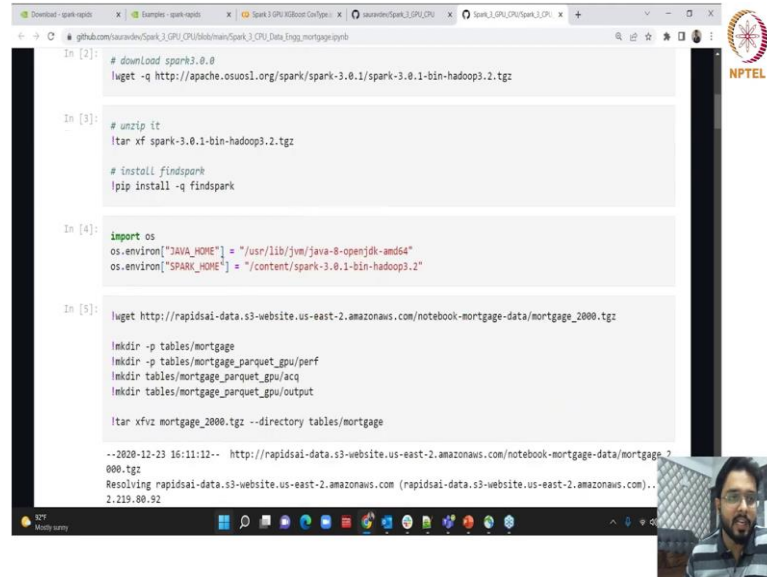


Though you can find more and more examples, where the FannieMae Mortgage Dataset, which I was talking about you can keep that as well here.

(Refer Slide Time: 16:31)



(Refer Slide Time: 16:36)



```
In [2]: # download spark3.0.0
!wget -q http://apache.osuosl.org/spark/spark-3.0.1/spark-3.0.1-bin-hadoop3.2.tgz

In [3]: # unzip it
!tar xf spark-3.0.1-bin-hadoop3.2.tgz

# install findspark
!pip install -q findspark

In [4]: import os
os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-8-openjdk-amd64"
os.environ["SPARK_HOME"] = "/content/spark-3.0.1-bin-hadoop3.2"

In [5]: !wget http://rapidsai-data.s3-website.us-east-2.amazonaws.com/notebook-mortgage-data/mortgage_2000.tgz

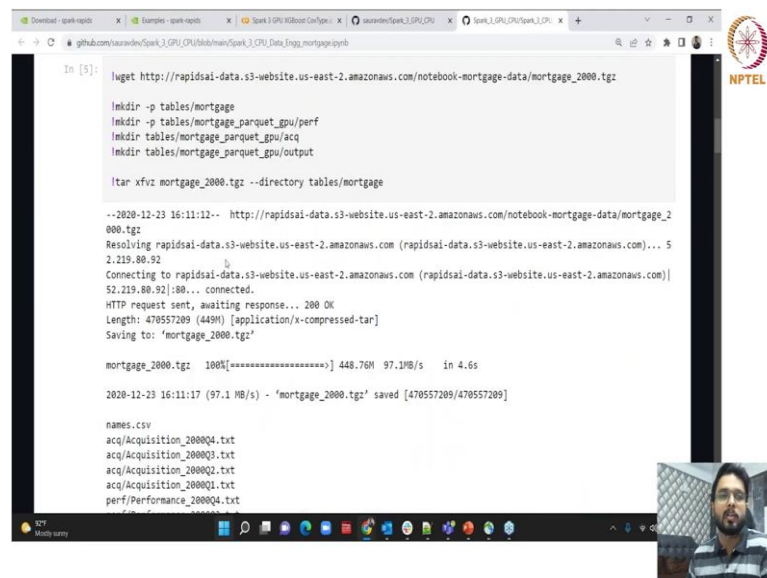
!mkdir -p tables/mortgage
!mkdir -p tables/mortgage_parquet_gpu/perf
!mkdir tables/mortgage_parquet_gpu/acq
!mkdir tables/mortgage_parquet_gpu/output

!tar xfvz mortgage_2000.tgz --directory tables/mortgage

--2020-12-23 16:11:12-- http://rapidsai-data.s3-website.us-east-2.amazonaws.com/notebook-mortgage-data/mortgage_2000.tgz
Resolving rapidsai-data.s3-website.us-east-2.amazonaws.com (rapidsai-data.s3-website.us-east-2.amazonaws.com)... 52.219.80.92
Connecting to rapidsai-data.s3-website.us-east-2.amazonaws.com (rapidsai-data.s3-website.us-east-2.amazonaws.com)|52.219.80.92|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 470557209 (449M) [application/x-compressed-tar]
Saving to: 'mortgage_2000.tgz'
```

So, if I can go through; so, here if you see the fanniemae mortgage data set, the installation part is exactly the same; just that the data download is different.

(Refer Slide Time: 16:45)



```
In [5]: !wget http://rapidsai-data.s3-website.us-east-2.amazonaws.com/notebook-mortgage-data/mortgage_2000.tgz

!mkdir -p tables/mortgage
!mkdir -p tables/mortgage_parquet_gpu/perf
!mkdir tables/mortgage_parquet_gpu/acq
!mkdir tables/mortgage_parquet_gpu/output

!tar xfvz mortgage_2000.tgz --directory tables/mortgage

--2020-12-23 16:11:12-- http://rapidsai-data.s3-website.us-east-2.amazonaws.com/notebook-mortgage-data/mortgage_2000.tgz
Resolving rapidsai-data.s3-website.us-east-2.amazonaws.com (rapidsai-data.s3-website.us-east-2.amazonaws.com)... 52.219.80.92
Connecting to rapidsai-data.s3-website.us-east-2.amazonaws.com (rapidsai-data.s3-website.us-east-2.amazonaws.com)|52.219.80.92|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 470557209 (449M) [application/x-compressed-tar]
Saving to: 'mortgage_2000.tgz'

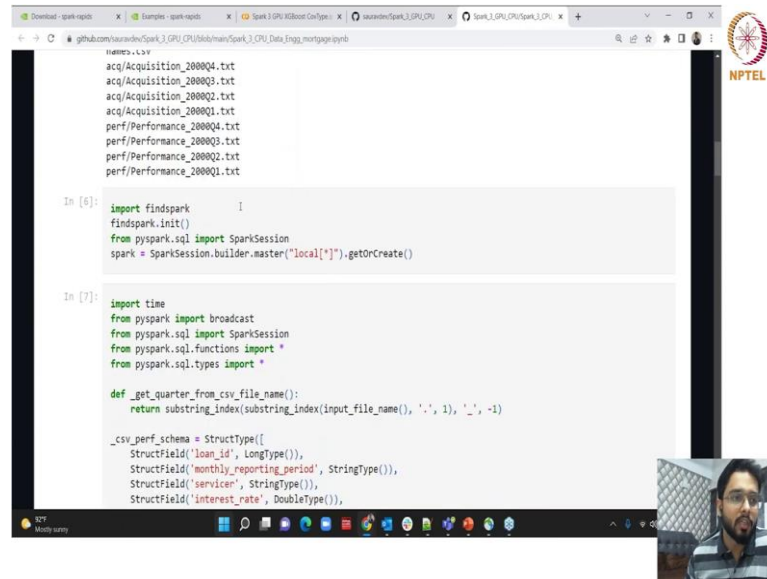
mortgage_2000.tgz 100%[=====] 448.76M 97.1MB/s in 4.6s

2020-12-23 16:11:17 (97.1 MB/s) - 'mortgage_2000.tgz' saved [470557209/470557209]

names.csv
acq/Acquisition_2000Q4.txt
acq/Acquisition_2000Q3.txt
acq/Acquisition_2000Q2.txt
acq/Acquisition_2000Q1.txt
perf/Performance_2000Q4.txt
```

Because you are downloading a new data set of mortgage data of 200 GB csv.

(Refer Slide Time: 16:54)



The screenshot shows a Jupyter Notebook interface with a file explorer at the top listing files like 'acq/Acquisition_2000Q4.txt' and 'perf/Performance_2000Q1.txt'. The code in the notebook includes:

```
import findspark
findspark.init()
from pyspark.sql import SparkSession
spark = SparkSession.builder.master("local[*]").getOrCreate()

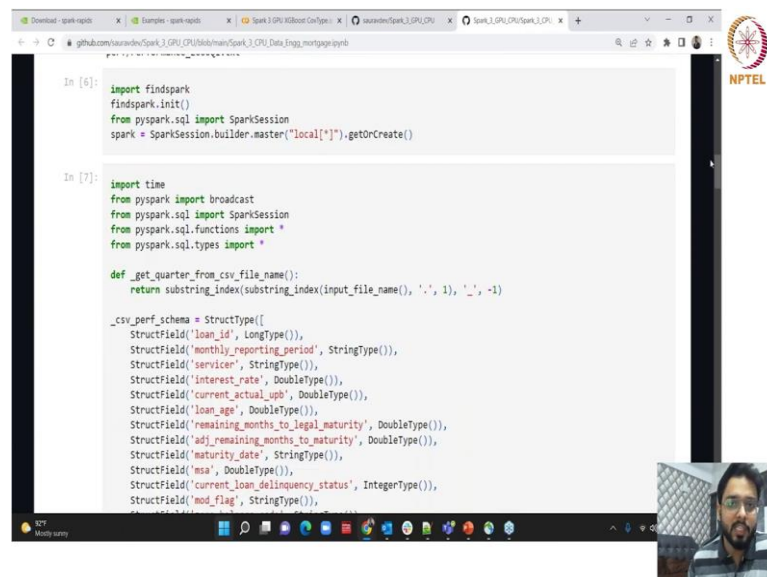
import time
from pyspark import broadcast
from pyspark.sql import SparkSession
from pyspark.sql.functions import *
from pyspark.sql.types import *

def _get_quarter_from_csv_file_name():
    return substring_index(substring_index(input_file_name(), '.', 1), '-', -1)

_csv_perf_schema = StructType([
    StructField('loan_id', LongType()),
    StructField('monthly_reporting_period', StringType()),
    StructField('servicer', StringType()),
    StructField('interest_rate', DoubleType()),
    StructField('current_actual_upb', DoubleType()),
    StructField('loan_age', DoubleType()),
    StructField('remaining_months_to_legal_maturity', DoubleType()),
    StructField('adj_remaining_months_to_maturity', DoubleType()),
    StructField('maturity_date', StringType()),
    StructField('msa', DoubleType()),
    StructField('current_loan_delinquency_status', IntegerType()),
    StructField('mod_flag', StringType())])
```

Then, we are reading the data. ah

(Refer Slide Time: 16:56)



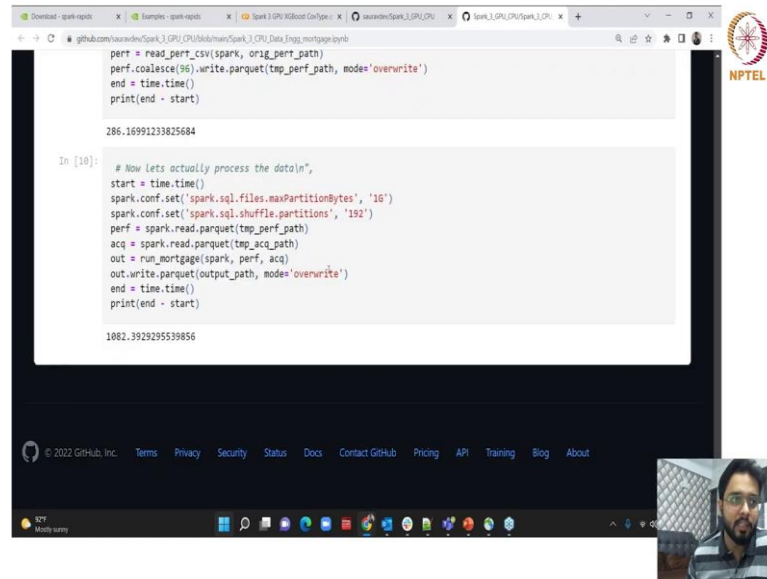
This screenshot shows the same Jupyter Notebook interface, but the schema definition in the code is more complete:

```
def _get_quarter_from_csv_file_name():
    return substring_index(substring_index(input_file_name(), '.', 1), '-', -1)

_csv_perf_schema = StructType([
    StructField('loan_id', LongType()),
    StructField('monthly_reporting_period', StringType()),
    StructField('servicer', StringType()),
    StructField('interest_rate', DoubleType()),
    StructField('current_actual_upb', DoubleType()),
    StructField('loan_age', DoubleType()),
    StructField('remaining_months_to_legal_maturity', DoubleType()),
    StructField('adj_remaining_months_to_maturity', DoubleType()),
    StructField('maturity_date', StringType()),
    StructField('msa', DoubleType()),
    StructField('current_loan_delinquency_status', IntegerType()),
    StructField('mod_flag', StringType())])
```

So, the base function, the main function is at the end, where which is calling the read
parquet.

(Refer Slide Time: 17:00)



```
perft = read_pert_csv(spark, orig_pert_path)
perft.coalesce(96).write.parquet(tmp_perft_path, mode='overwrite')
end = time.time()
print(end - start)

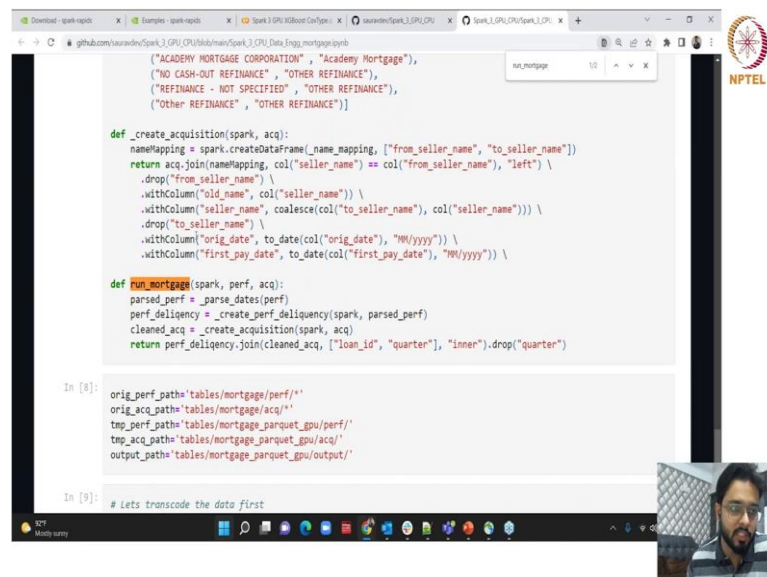
286.1699123825684

In [10]: # Now lets actually process the data\n",
start = time.time()
spark.conf.set('spark.sql.files.maxPartitionBytes', '16')
spark.conf.set('spark.sql.shuffle.partitions', '192')
perft = spark.read.parquet(tmp_perft_path)
acq = spark.read.parquet(tmp_acq_path)
out = run_mortgage(spark, perft, acq)
out.write.parquet(output_path, mode='overwrite')
end = time.time()
print(end - start)

1062.3929295539856
```

Then, it is converting that parquet into read csv and converting to parquet.

(Refer Slide Time: 17:15)



```
("ACADEMY MORTGAGE CORPORATION", "Academy Mortgage"),
("NO CASH-OUT REFINANCE", "OTHER REFINANCE"),
("REFINANCE - NOT SPECIFIED", "OTHER REFINANCE"),
("Other REFINANCE", "OTHER REFINANCE"))

def _create_acquisition(spark, acq):
    nameMapping = spark.createDataFrame(name_mapping, ["from_seller_name", "to_seller_name"])
    return acq.join(nameMapping, col("seller_name") == col("from_seller_name"), "left") \
        .drop("from_seller_name") \
        .withColumn("old_name", col("seller_name")) \
        .withColumn("seller_name", coalesce(col("to_seller_name"), col("seller_name"))) \
        .drop("to_seller_name") \
        .withColumn("orig_date", to_date(col("orig_date"), "MM/yyyy")) \
        .withColumn("first_pay_date", to_date(col("first_pay_date"), "MM/yyyy")) \

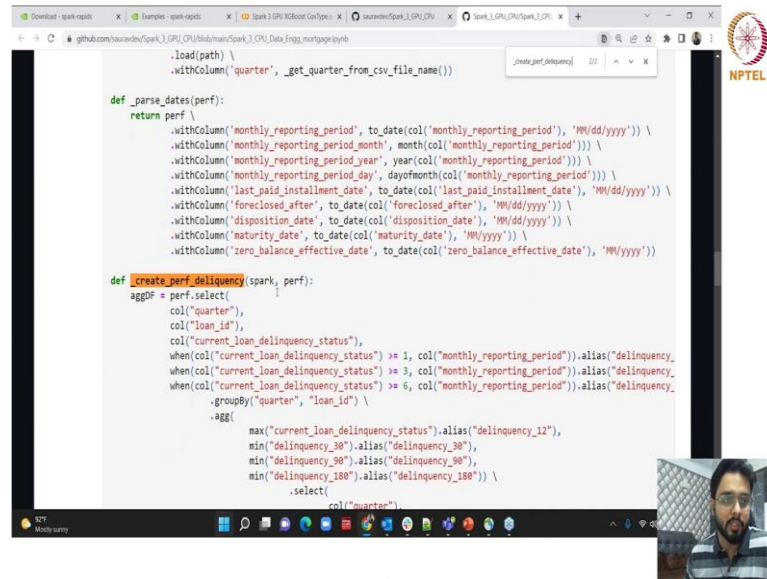
def run_mortgage(spark, perft, acq):
    parsed_perft = _parse_dates(perft)
    perft_delinquency = _create_perft_delinquency(spark, parsed_perft)
    cleaned_acq = _create_acquisition(spark, acq)
    return perft_delinquency.join(cleaned_acq, ["loan_id", "quarter"], "inner").drop("quarter")

In [8]: orig_perft_path='tables/mortgage/perft/'
orig_acq_path='tables/mortgage/acq/'
tmp_perft_path='tables/mortgage_parquet_gpu/perft/'
tmp_acq_path='tables/mortgage_parquet_gpu/acq/'
output_path='tables/mortgage_parquet_gpu/output/'

In [9]: # Lets transcode the data first
```

Then, reading the parquet and running this notebook function called run mortgage, which is basically running the two more functions called create of delinquency and create acquisition. So, create perft delinquency is one where we do the maximum transformation, where you can see we have case one statements, we have group by aggregates, where we have max, min, select, joint joining of data frames.

(Refer Slide Time: 17:27)

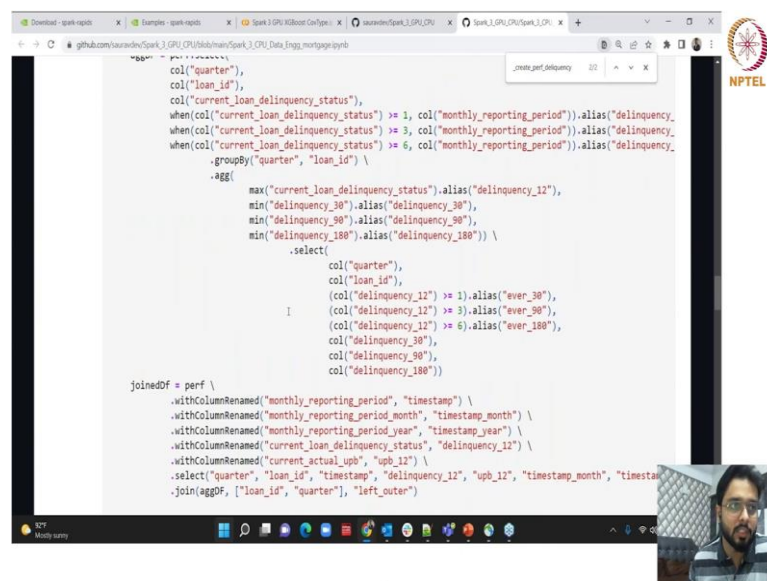


```
.load(path) \
.withColumn('quarter', _get_quarter_from_csv_file_name())

def _parse_dates(perf):
    return perf \
        .withColumn('monthly_reporting_period', to_date(col('monthly_reporting_period'), 'MM/dd/yyyy')) \
        .withColumn('monthly_reporting_period_month', month(col('monthly_reporting_period'))) \
        .withColumn('monthly_reporting_period_year', year(col('monthly_reporting_period'))) \
        .withColumn('monthly_reporting_period_day', dayofmonth(col('monthly_reporting_period'))) \
        .withColumn('last_paid_installment_date', to_date(col('last_paid_installment_date'), 'MM/dd/yyyy')) \
        .withColumn('foreclosed_after', to_date(col('foreclosed_after'), 'MM/dd/yyyy')) \
        .withColumn('disposition_date', to_date(col('disposition_date'), 'MM/dd/yyyy')) \
        .withColumn('maturity_date', to_date(col('maturity_date'), 'MM/yyyy')) \
        .withColumn('zero_balance_effective_date', to_date(col('zero_balance_effective_date'), 'MM/yyyy'))

def _create_perf_delinquency(spark, perf):
    aggDF = perf.select(
        col("quarter"),
        col("loan_id"),
        col("current_loan_delinquency_status"),
        when(col("current_loan_delinquency_status") >= 1, col("monthly_reporting_period")).alias("delinquency_12"),
        when(col("current_loan_delinquency_status") >= 3, col("monthly_reporting_period")).alias("delinquency_30"),
        when(col("current_loan_delinquency_status") >= 6, col("monthly_reporting_period")).alias("delinquency_90"),
        .groupBy("quarter", "loan_id") \
        .agg(
            max("current_loan_delinquency_status").alias("delinquency_12"),
            min("delinquency_30").alias("delinquency_30"),
            min("delinquency_90").alias("delinquency_90"),
            min("delinquency_180").alias("delinquency_180")) \
        .select(
            col("quarter"),
```

(Refer Slide Time: 17:29)

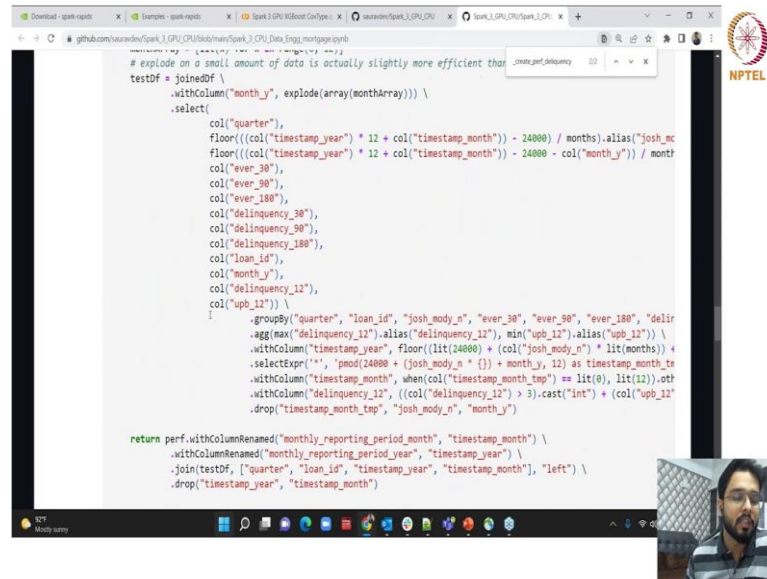


```
col("quarter"),
col("loan_id"),
col("current_loan_delinquency_status"),
when(col("current_loan_delinquency_status") >= 1, col("monthly_reporting_period")).alias("delinquency_12"),
when(col("current_loan_delinquency_status") >= 3, col("monthly_reporting_period")).alias("delinquency_30"),
when(col("current_loan_delinquency_status") >= 6, col("monthly_reporting_period")).alias("delinquency_90"),
.groupBy("quarter", "loan_id") \
.agg(
    max("current_loan_delinquency_status").alias("delinquency_12"),
    min("delinquency_30").alias("delinquency_30"),
    min("delinquency_90").alias("delinquency_90"),
    min("delinquency_180").alias("delinquency_180")) \
    .select(
        col("quarter"),
        col("loan_id"),
        (col("delinquency_12") >= 1).alias("ever_30"),
        (col("delinquency_12") >= 3).alias("ever_90"),
        (col("delinquency_12") >= 6).alias("ever_180"),
        col("delinquency_30"),
        col("delinquency_90"),
        col("delinquency_180"))

joinedDF = perf \
    .withColumnRenamed("monthly_reporting_period", "timestamp") \
    .withColumnRenamed("monthly_reporting_period_month", "timestamp_month") \
    .withColumnRenamed("monthly_reporting_period_year", "timestamp_year") \
    .withColumnRenamed("current_loan_delinquency_status", "delinquency_12") \
    .withColumnRenamed("current_loan_delinquency_status", "upb_12") \
    .select("quarter", "loan_id", "timestamp", "delinquency_12", "upb_12", "timestamp_month", "timestamp_year") \
    .join(aggDF, ["loan_id", "quarter"], "left_outer")
```

hm.

(Refer Slide Time: 17:41)



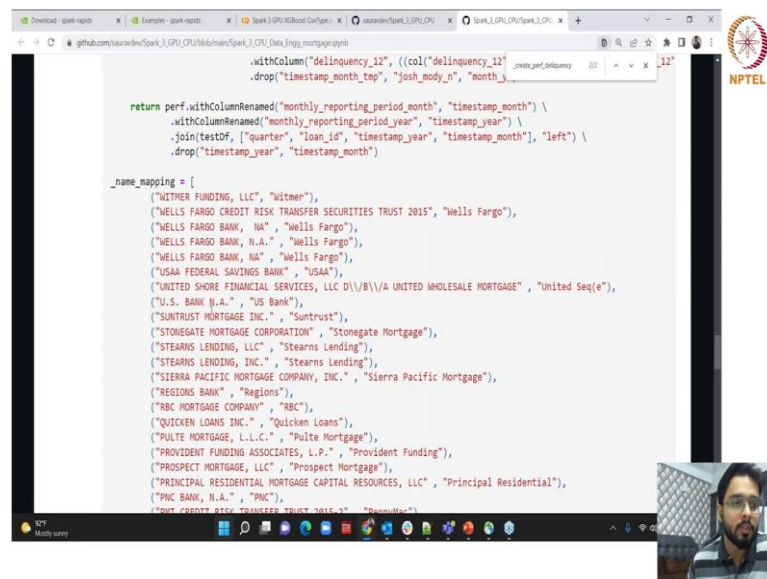
```
# explode on a small amount of data is actually slightly more efficient than
testDF = joinedDF
    .withColumn("month_y", explode(array(monthArray))) \
    .select(
        col("quarter"),
        floor((col("timestamp_year") * 12 + col("timestamp_month")) - 24000) / months).alias("josh_mc"),
        floor((col("timestamp_year") * 12 + col("timestamp_month")) - 24000 - col("month_y")) / months).alias("josh_n"),
        col("ever_30"),
        col("ever_90"),
        col("ever_180"),
        col("delinquency_30"),
        col("delinquency_90"),
        col("delinquency_180"),
        col("loan_id"),
        col("month_y"),
        col("delinquency_12"),
        col("upb_12")) \
    .groupBy("quarter", "loan_id", "josh_mc", "josh_n", "ever_30", "ever_90", "ever_180", "delinquency_30", "delinquency_90", "delinquency_180", "upb_12") \
    .agg(max("delinquency_12").alias("delinquency_12"), min("upb_12").alias("upb_12")) \
    .withColumn("timestamp_year", floor(lit(24000) + (col("josh_mc") * 12) + col("josh_n"))) \
    .selectExpr('timestamp_year', 'timestamp_month', 'timestamp_year * 12 + timestamp_month as timestamp_month_tmp', 'timestamp_year * 12 + timestamp_month_tmp as timestamp_month', 'timestamp_year * 12 + timestamp_month_tmp as timestamp_month_tmp', 'timestamp_year * 12 + timestamp_month_tmp as timestamp_month') \
    .withColumn("delinquency_12", ((col("delinquency_12") > 3).cast("int") + col("upb_12"))) \
    .drop("timestamp_month_tmp", "josh_mc", "month_y")

return perf.withColumnRenamed("monthly_reporting_period_month", "timestamp_month") \
    .withColumnRenamed("monthly_reporting_period_year", "timestamp_year") \
    .join(testDF, ["quarter", "loan_id", "timestamp_year", "timestamp_month"], "left") \
    .drop("timestamp_year", "timestamp_month")

_name_mapping = [
    ("WITHER FUNDING, LLC", "Wither"),
    ("WELLS FARGO CREDIT RISK TRANSFER SECURITIES TRUST 2015", "Wells Fargo"),
    ("WELLS FARGO BANK, NA", "Wells Fargo"),
    ("WELLS FARGO BANK, N.A.", "Wells Fargo"),
    ("WELLS FARGO BANK, NA", "Wells Fargo"),
    ("USAA FEDERAL SAVINGS BANK", "USAA"),
    ("UNITED SHORE FINANCIAL SERVICES, LLC D\I\B\A UNITED WHOLESALE MORTGAGE", "United Seq(a)"),
    ("U.S. BANK (A.A.", "US Bank"),
    ("SUNTRUST MORTGAGE INC.", "Suntrust"),
    ("STONEGATE MORTGAGE CORPORATION", "Stonegate Mortgage"),
    ("STEARNS LENDING, LLC", "Stearns Lending"),
    ("STEARNS LENDING, INC.", "Stearns Lending"),
    ("SIERRA PACIFIC MORTGAGE COMPANY, INC.", "Sierra Pacific Mortgage"),
    ("REGIONS BANK", "Regions"),
    ("RBC MORTGAGE COMPANY", "RBC"),
    ("QUICKEN LOANS INC.", "Quicken Loans"),
    ("PULTE MORTGAGE, L.L.C.", "Pulte Mortgage"),
    ("PROVIDENT FUNDING ASSOCIATES, L.P.", "Provident Funding"),
    ("PROSPECT MORTGAGE, LLC", "Prospect Mortgage"),
    ("PRINCIPAL RESIDENTIAL MORTGAGE CAPITAL RESOURCES, LLC", "Principal Residential"),
    ("PMC BANK, N.A.", "PMC"),
    ("PNC CREDIT RISK TRANSFER TRUST 2015-2", "PNC Credit Risk Transfer Trust 2015-2")]
```

And then, with column exploding selecting and so on and so forth.

(Refer Slide Time: 17:45)

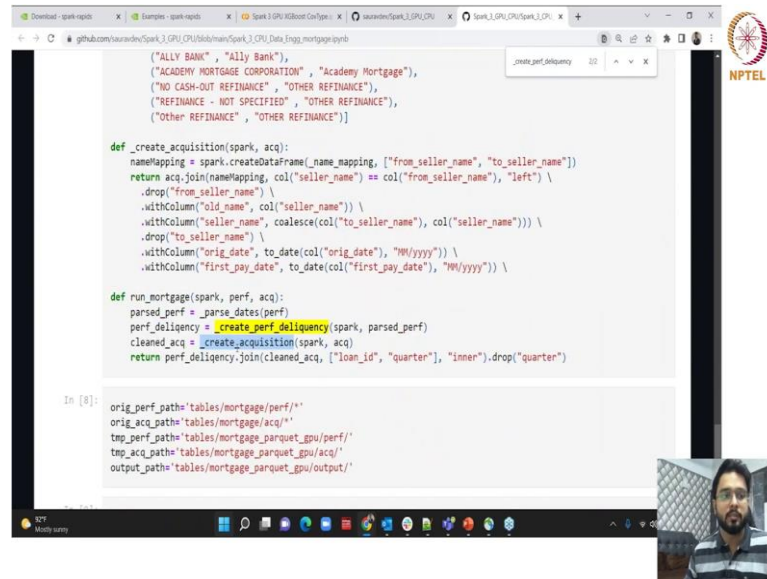


```
    .drop("timestamp_month_tmp", "josh_mc", "month_y")

return perf.withColumnRenamed("monthly_reporting_period_month", "timestamp_month") \
    .withColumnRenamed("monthly_reporting_period_year", "timestamp_year") \
    .join(testDF, ["quarter", "loan_id", "timestamp_year", "timestamp_month"], "left") \
    .drop("timestamp_year", "timestamp_month")

_name_mapping = [
    ("WITHER FUNDING, LLC", "Wither"),
    ("WELLS FARGO CREDIT RISK TRANSFER SECURITIES TRUST 2015", "Wells Fargo"),
    ("WELLS FARGO BANK, NA", "Wells Fargo"),
    ("WELLS FARGO BANK, N.A.", "Wells Fargo"),
    ("WELLS FARGO BANK, NA", "Wells Fargo"),
    ("USAA FEDERAL SAVINGS BANK", "USAA"),
    ("UNITED SHORE FINANCIAL SERVICES, LLC D\I\B\A UNITED WHOLESALE MORTGAGE", "United Seq(a)"),
    ("U.S. BANK (A.A.", "US Bank"),
    ("SUNTRUST MORTGAGE INC.", "Suntrust"),
    ("STONEGATE MORTGAGE CORPORATION", "Stonegate Mortgage"),
    ("STEARNS LENDING, LLC", "Stearns Lending"),
    ("STEARNS LENDING, INC.", "Stearns Lending"),
    ("SIERRA PACIFIC MORTGAGE COMPANY, INC.", "Sierra Pacific Mortgage"),
    ("REGIONS BANK", "Regions"),
    ("RBC MORTGAGE COMPANY", "RBC"),
    ("QUICKEN LOANS INC.", "Quicken Loans"),
    ("PULTE MORTGAGE, L.L.C.", "Pulte Mortgage"),
    ("PROVIDENT FUNDING ASSOCIATES, L.P.", "Provident Funding"),
    ("PROSPECT MORTGAGE, LLC", "Prospect Mortgage"),
    ("PRINCIPAL RESIDENTIAL MORTGAGE CAPITAL RESOURCES, LLC", "Principal Residential"),
    ("PMC BANK, N.A.", "PMC"),
    ("PNC CREDIT RISK TRANSFER TRUST 2015-2", "PNC Credit Risk Transfer Trust 2015-2")]
```

(Refer Slide Time: 17:50)



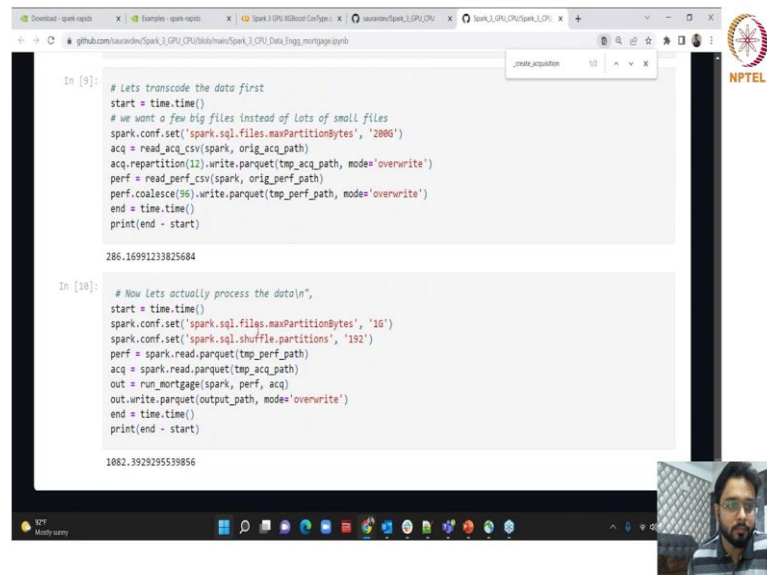
```
def _create_acquisition(spark, acq):
    nameMapping = spark.createDataFrame(name_mapping, ["from_seller_name", "to_seller_name"])
    return acq.join(nameMapping, col("seller_name") == col("from_seller_name"), "left") \
        .drop("from_seller_name") \
        .withColumn("old_name", col("seller_name")) \
        .withColumn("seller_name", coalesce(col("to_seller_name"), col("seller_name"))) \
        .drop("to_seller_name") \
        .withColumn("orig_date", to_date(col("orig_date"), "MM/yyyy")) \
        .withColumn("first_pay_date", to_date(col("first_pay_date"), "MM/yyyy")) \

def run_mortgage(spark, perf, acq):
    parsed_perf = _parse_dates(perf)
    perf_delinquency = _create_perf_delinquency(spark, parsed_perf)
    cleaned_acq = _create_acquisition(spark, acq)
    return perf_delinquency.join(cleaned_acq, ["loan_id", "quarter"], "inner").drop("quarter")

In [8]:
orig_perf_paths='tables/mortgage/perf/'
orig_acq_paths='tables/mortgage/acq/'
tmp_perf_paths='tables/mortgage_parquet_gpu/perf/'
tmp_acq_paths='tables/mortgage_parquet_gpu/acq/'
output_path='tables/mortgage_parquet_gpu/output/'
```

And then, we have creating acquisition a function, where we have creating data frame, joining the data frame, dropping, adding more columns. So, you might have if you even do not know spark, it is very similar to what you can do with pandas or similar tools. So, it is a data frame based processing, a logic which you definitely might know about.

(Refer Slide Time: 18:13)



```
In [9]:
# Lets transcode the data first
start = time.time()
# we want a few big files instead of lots of small files
spark.conf.set('spark.sql.files.maxPartitionBytes', '200G')
acq = read_acq_csv(spark, orig_acq_path)
acq.repartition(12).write.parquet(tmp_acq_path, mode='overwrite')
perf = read_perf_csv(spark, orig_perf_path)
perf.coalesce(96).write.parquet(tmp_perf_path, mode='overwrite')
end = time.time()
print(end - start)

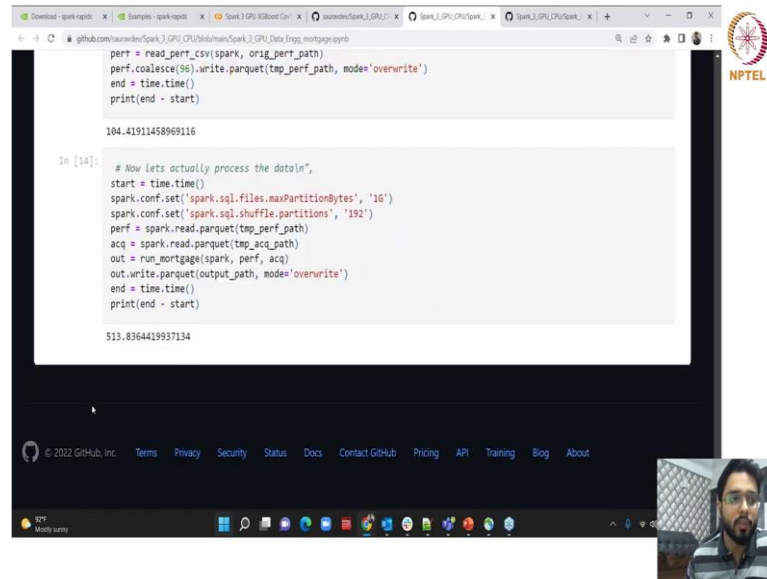
286.1699123825684

In [10]:
# Now lets actually process the data
start = time.time()
spark.conf.set('spark.sql.files.maxPartitionBytes', '1G')
spark.conf.set('spark.sql.shuffle.partitions', '192')
perf = spark.read.parquet(tmp_perf_path)
acq = spark.read.parquet(tmp_acq_path)
out = run_mortgage(spark, perf, acq)
out.write.parquet(output_path, mode='overwrite')
end = time.time()
print(end - start)

1082.3929295539856
```

So, here also you can see that there is a performance improvement for sure. So, here if you see we are running in 1082 seconds. Ah. So, if I compare it with GPU, so this was the CPU part, where we are taking 1082 seconds.

(Refer Slide Time: 18:54)



```
perft = read_perft_csv(spark, orig_perft_path)
perft.coalesce(96).write.parquet(tmp_perft_path, mode='overwrite')
end = time.time()
print(end - start)

104.41911458969116

In [14]: # Now lets actually process the data\n",
start = time.time()
spark.conf.set('spark.sql.files.maxPartitionBytes', '16')
spark.conf.set('spark.sql.shuffle.partitions', '192')
perft = spark.read.parquet(tmp_perft_path)
acq = spark.read.parquet(tmp_acq_path)
out = run_mortgage(spark, perft, acq)
out.write.parquet(output_path, mode='overwrite')
end = time.time()
print(end - start)

513.8364419937134
```

But the GPU the same thing on the GPU will just run in 513 seconds. So, you will see that ok you are even gaining 2x on the Google colab environment which is free to use as well. Because it does not have that great GPU, it is a very cheap GPU called T4. So, you will just see 2 to 2.5x acceleration. But if you use better GPUs like A30, A100, then definitely you will find more and more acceleration.