**PySpark**

Hadoop 🡪 HDFS Old data processing in Hadoop 🡪 Mapreduce

Data Storage Data Processing

Batch Processing 🡪 Mapreduce

Streaming 🡪 logs, sensor, gps, tweets 🡺storm

Data analytics/warehouse 🡺 Hive

Machine Learning 🡪 Mahout

Graph processing 🡪 Giraph

In-memory 🡪 spark 🡪 unified data analytical engine for large scale

UCB 🡪 Amplab 🡪 spark

Data processing 🡪 lightening faster cluster computing

Apache spark 🡪 opensource

Databricks 🡪 spark 🡪 licensed/commercial/runs only on cloud platform

Apache spark is Polyglot 🡪 python, Scala, R, Java, C#, SQL i.e., Multiple lib

Multiple cluster 🡺 YARN, Mesos, K8s, spark standalone

Different Source and target support is here. No client server architecture. Multiple nodes support (Master slave architecture). Here we call it as Daemons 🡺 master 1 and workers 1000s. Using scala they developed this product.

Dataframe – abstraction layer on top of RDD. (so that data is visible in proper tabular form)

Data structure 🡪 building block of spark storage is RDD (Resilient Distributed Dataset)

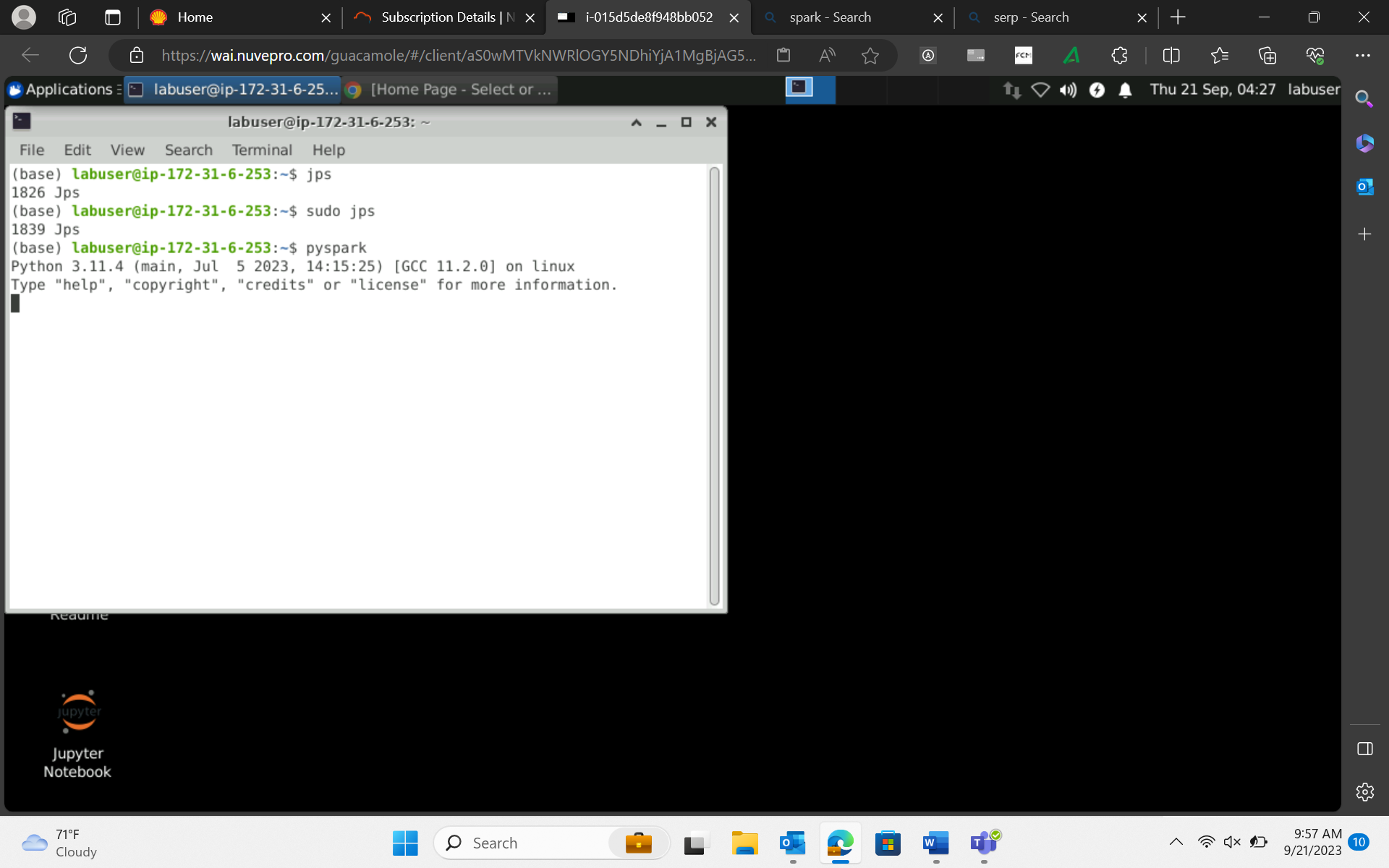
Resilient means recoverable.

Features of RDD:

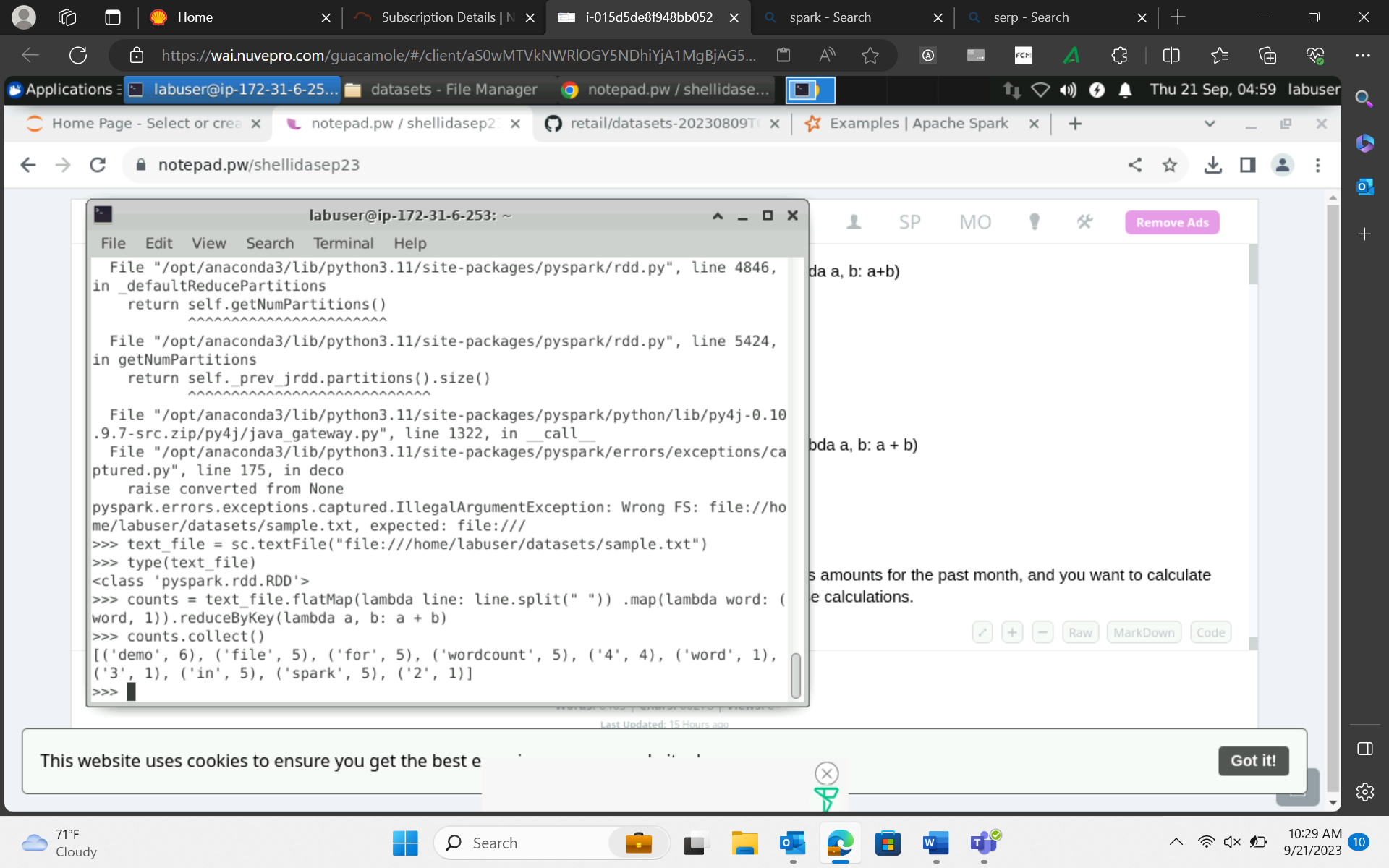
1. Immutable
2. Type inferred
3. Data is Cacheable
4. Supports lazy evaluation -
5. Fault tolerance
6. Architecture is distributed computing so data is partition/chunks/blocks/piece/shards
7. Fault tolerance
8. High availability of data
9. MPP – Massively parallel processing

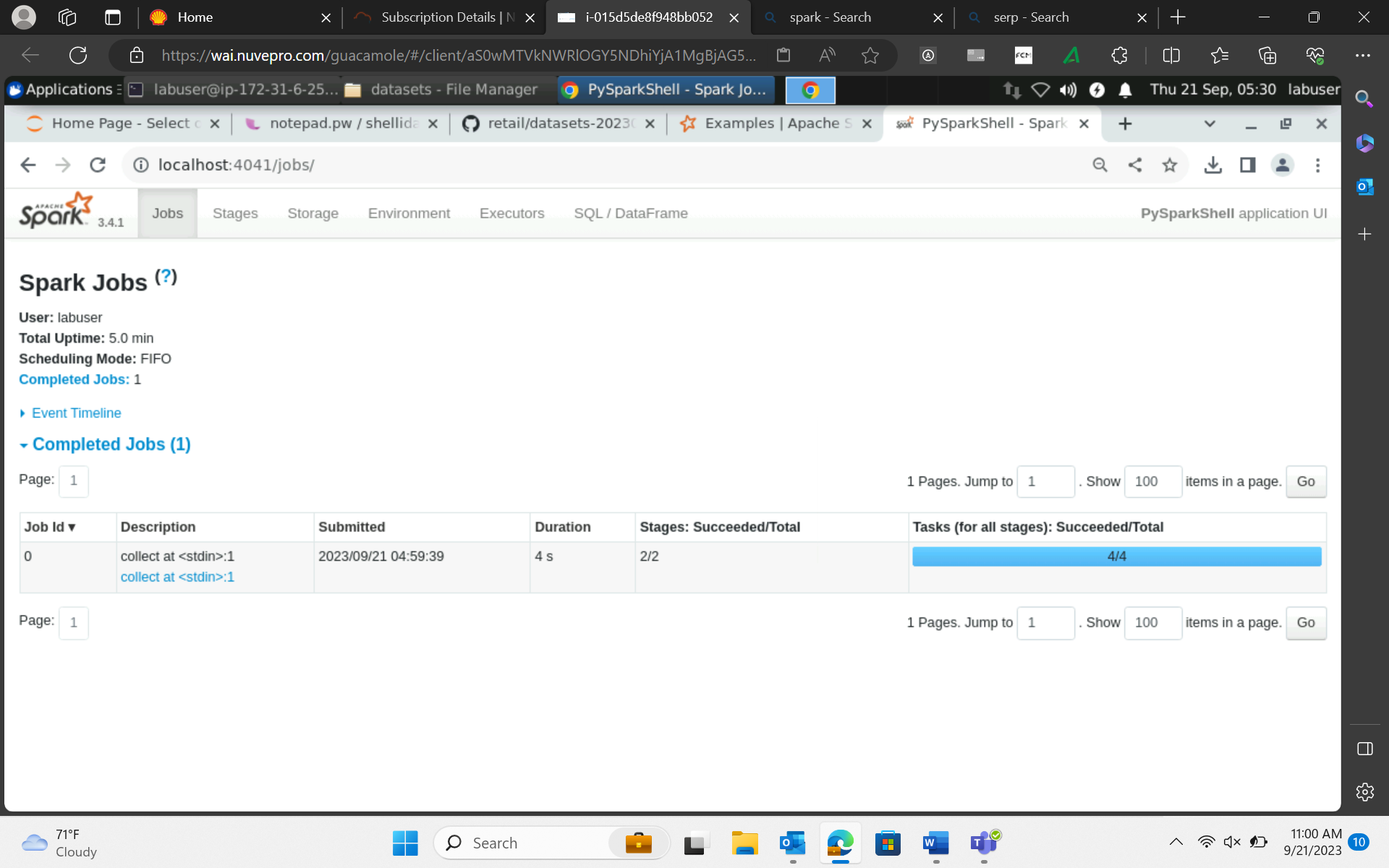
Two operations in spark 🡪 transformation and action

1. When run new Transformations 🡪 new RDD/df
2. Then apply Actions 🡪trigger spark job🡪 output



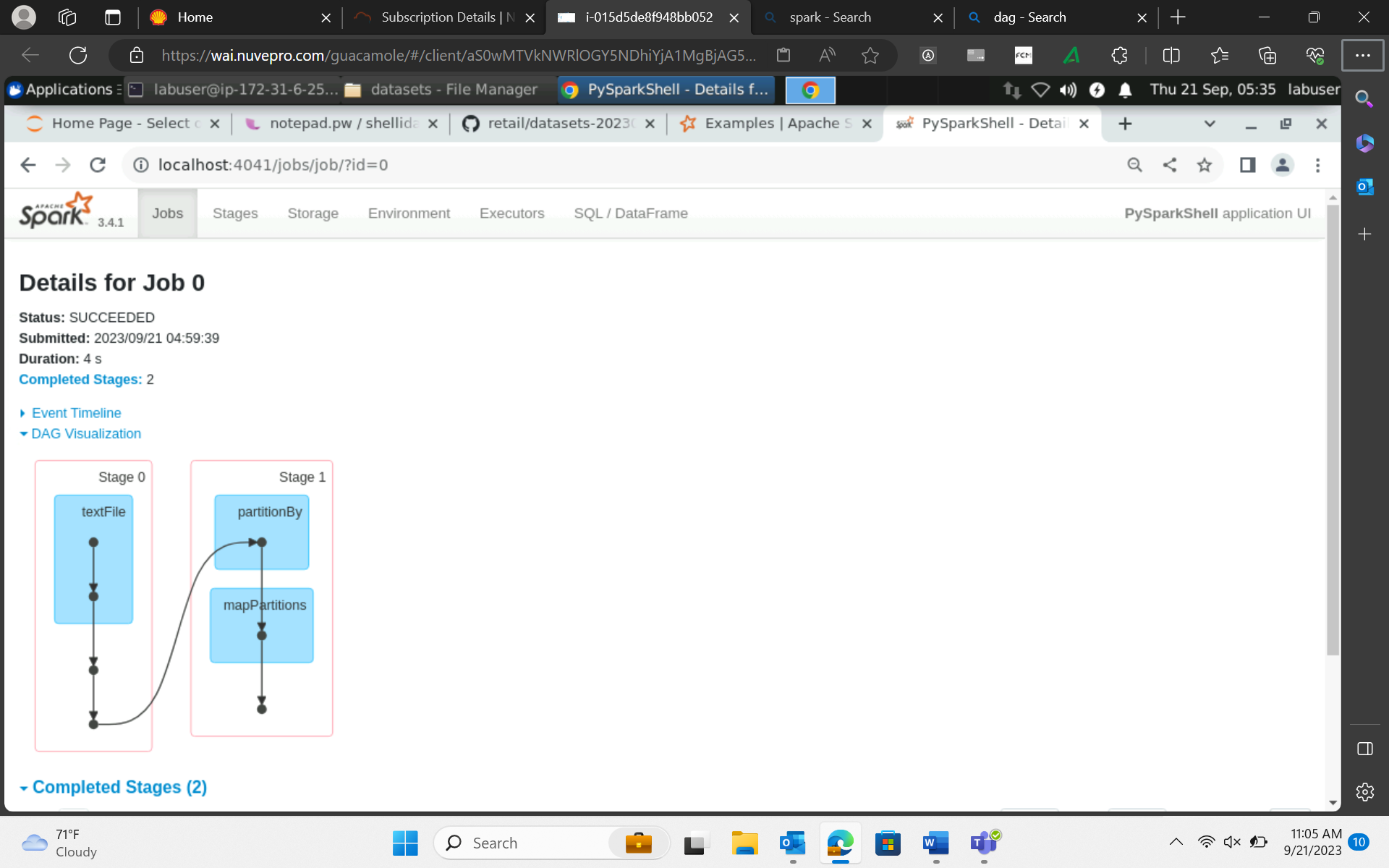
* Wordcount in spark

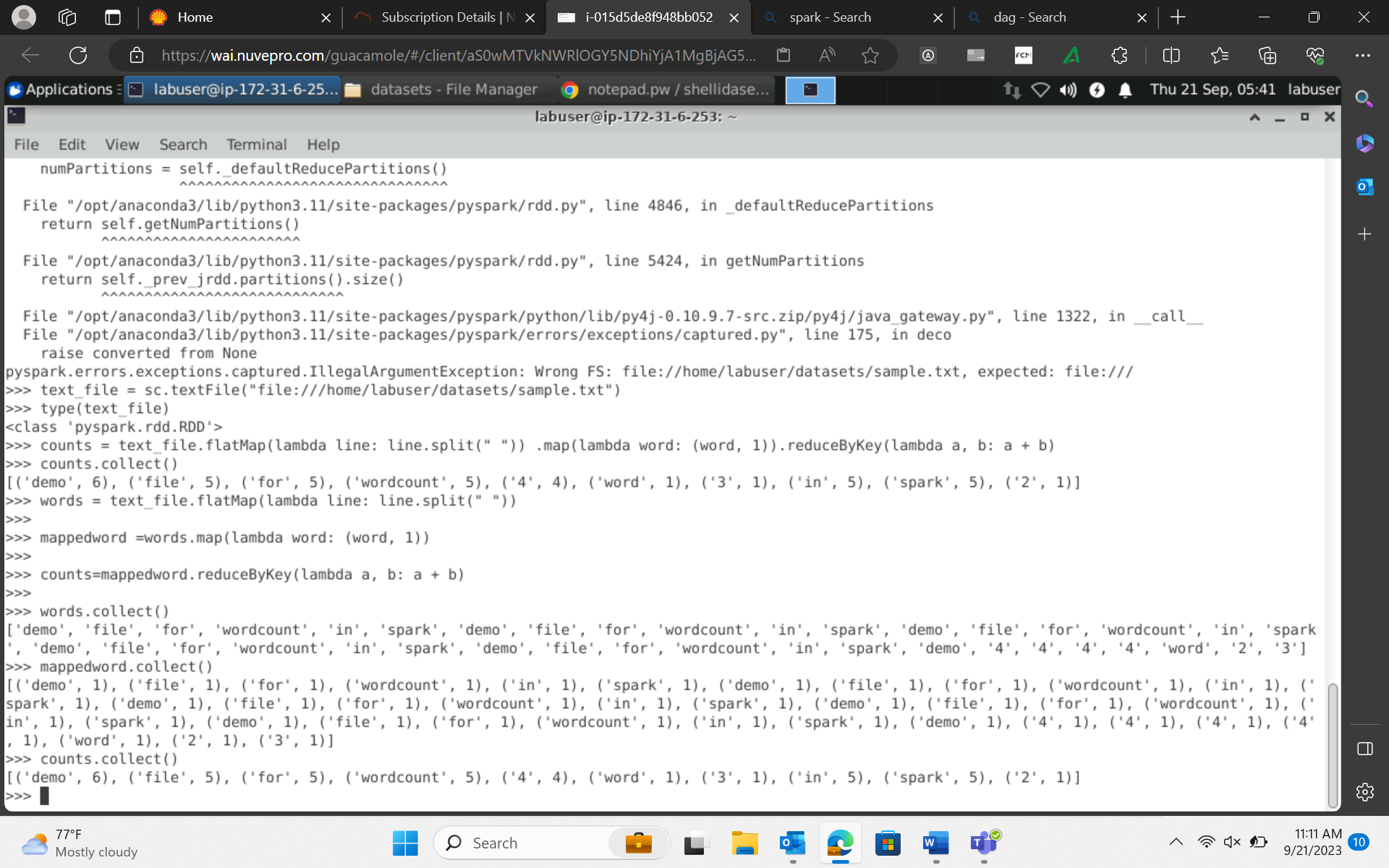




sc.uiWebUrl 🡪 to know port number

DAG 🡪 directed acyclic graph 🡪





Flatmap 🡪 converts line into words (Flattens the data, line is split into words)

Map 🡪 assigns every word with integer value 1

ReduceByKey 🡪 aggregation (count of words)

Flatmap and map do not have any shuffling.

Transformation have two types :

1. Narrow transformation – if there is no shuffle. E.g. is flatmap and map

Each partition of the parent RDD is used by at most one partition of the child RDD.

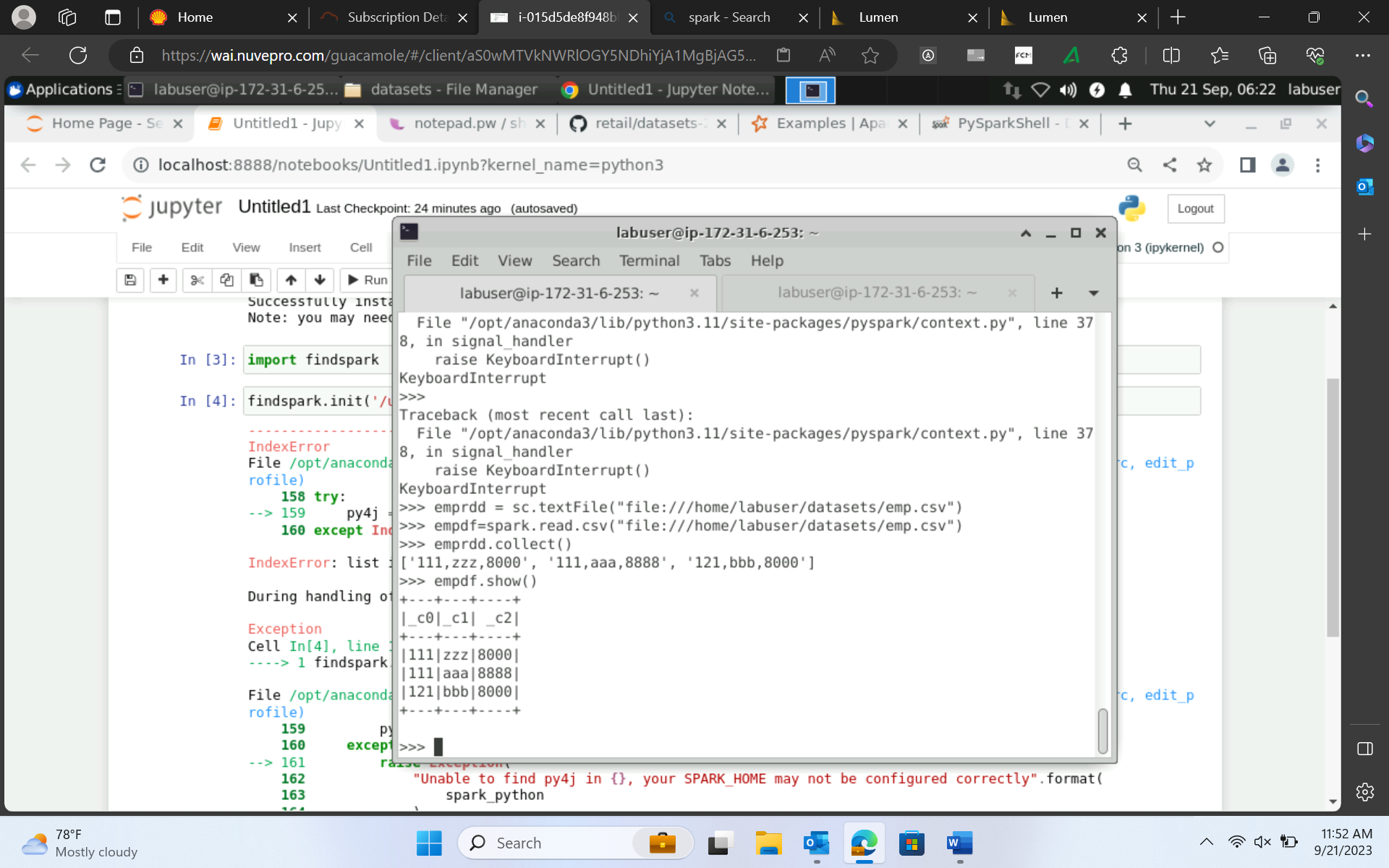
It performs in same stage

1. Wide transformation – If there is a shuffle. Eg reduceByKey

Multiple child RDD

New stage is created for the shuffle.

\*\*Whenever there is a shuffle it happens in a new stage. Therefore reducebykey have two stages and map, flatmap are done in single stage.



RDD vs Dataframe/ datasets(Java/Scala)(Not available in python/R bcoz of its dynamically typed nature)

RDD 🡪 Data 🡺 list

Dataframe 🡪 table format ( Schema + data)

One partition 🡪 one task 🡪 one executor

Components:

1. Job – a piece of code with specific business logic
2. Stages – shuffle happens in new stages
3. Partition 🡪 data is stored in collection of partitions 🡪 Tasks 🡪 stored in worker’s memory. That’s why we say spark is in-memory processing.
4. DAG
5. Executor
6. Driver
7. Master 🡪 driver program gets executed from master
8. Slave 🡪 they actually executes the task

Spark – in-memory processing framework.

Data is stored in memory and not in disk. Storing in disk is expensive. Dataset/RDD is not a processor.

SparkSession 🡪 entry point of your job 🡪 referred as spark

SparkContext 🡪 coordinate with cluster resource manager and then allocate worker to do job 🡪 referred as sc

Data Ingestion : spark.read.csv(path)

Header in spark

salesdf=spark.read.option("header","true").csv("/home/labuser/datasets/superstore.csv")

To check data type of dataframe - employeedf.printSchema()

Option(“sep”,”|”).csv

Id needs to be integer (to infer data types) - salesdf=spark.read.option("header","true").option("inferSchema","true").csv("/home/labuser/datasets/superstore.csv")

OR

employeedf=spark.read.option("header","true").csv("/home/labuser/datasets/employee.csv",inferSchema=True)

User defined schema creation 🡪 recommended in real time 🡪 using StructType and StructField

from pyspark.sql.types import StructType,StructField, StringType,IntegerType,DoubleType

EmpSchema = StructType([

StructField('Empno', IntegerType(), True),

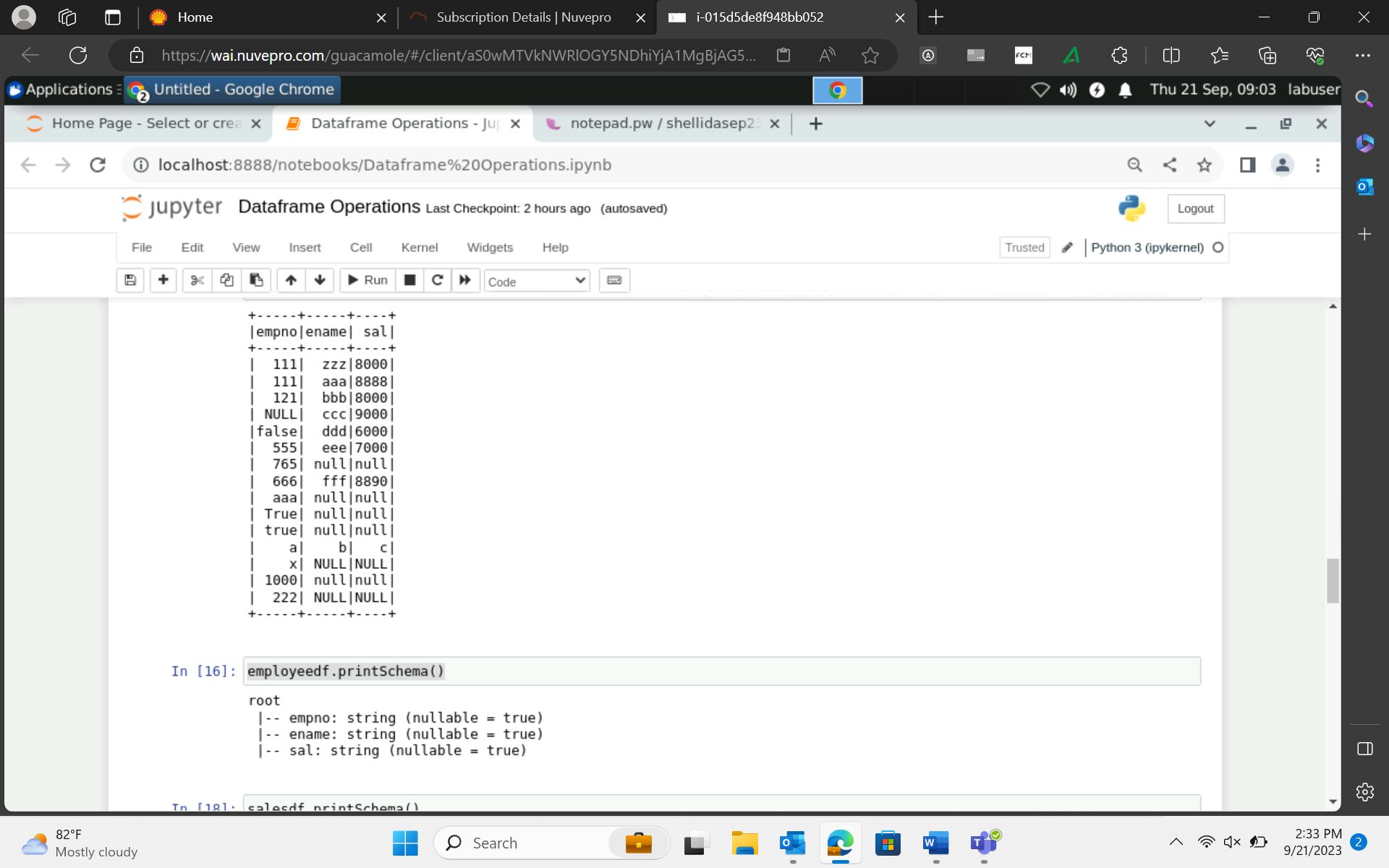
StructField('Empname', StringType(), True),

StructField('salary', DoubleType(), True)

])

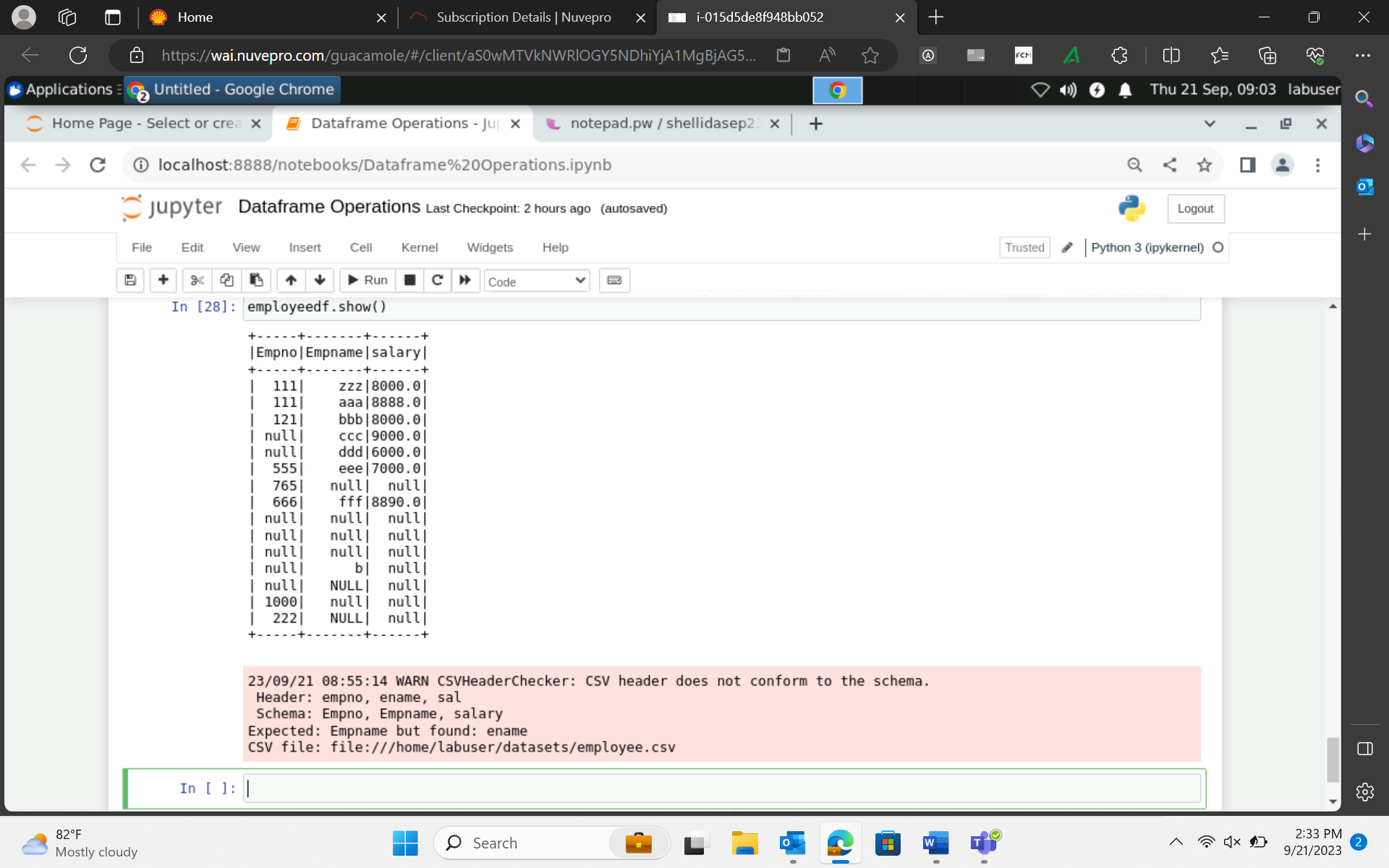
employeedf=spark.read.option("header","true").schema(EmpSchema).csv("/home/labuser/datasets/employee.csv")

Handling bad records

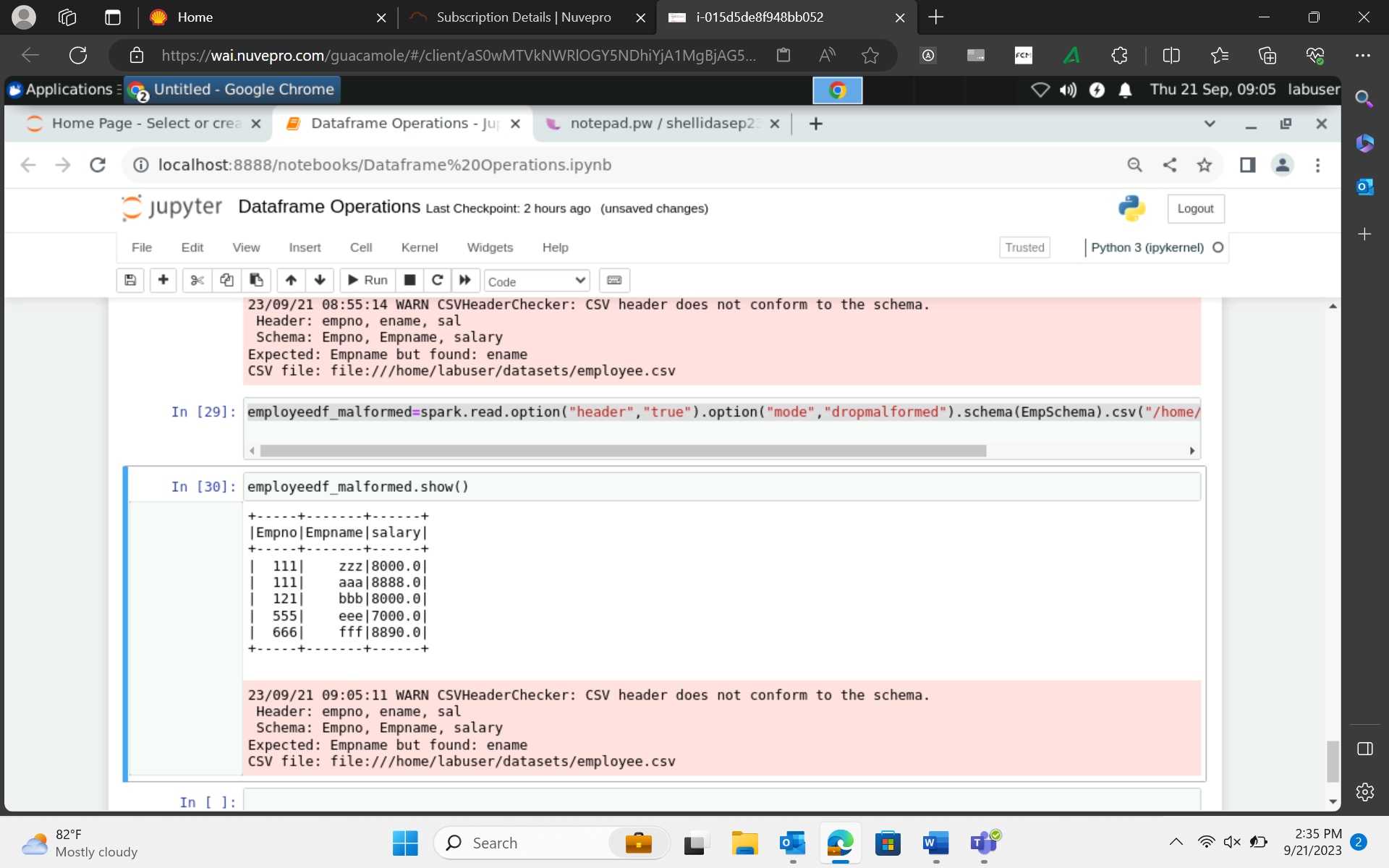


Modes :

1. Permissive – default – replace the mismatch datatype with null.



1. Dropmalformed - employeedf\_malformed=spark.read.option("header","true").option("mode","dropmalformed").schema(EmpSchema).csv("/home/labuser/datasets/employee.csv")



1. Failfast –for very sensitive data

salesdf.count()

salesdf.head()

salesdf.rdd.getNumPartitions()

salesdf.show(2)

salesdf.first()

first does not allows argument but head does. You can pass no of records needed.

salesdf.take(2)

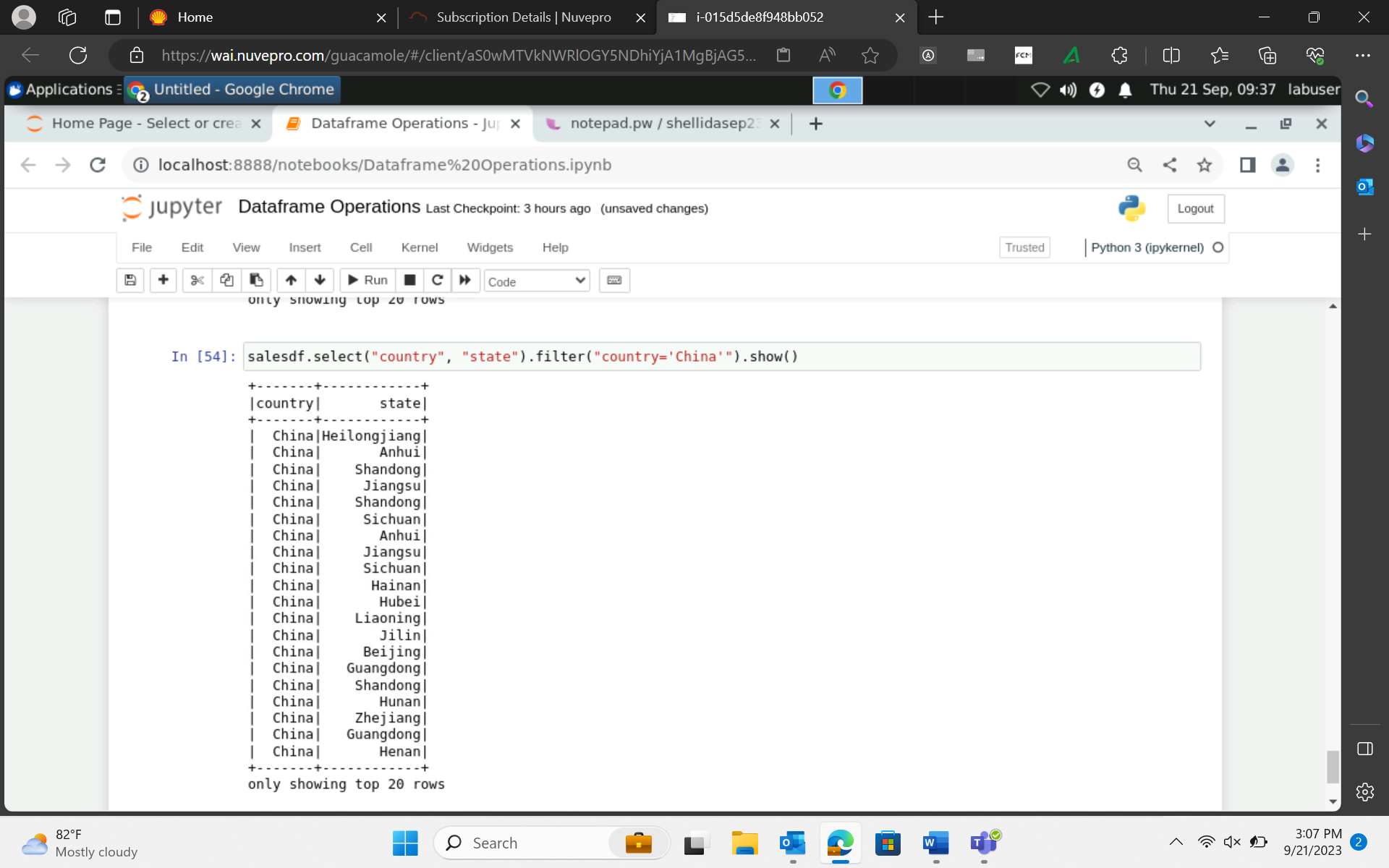
Parameter mandatory in take. Not in head.

Selecting data 🡺 salesdf.select("country", "state").show()

Select() 🡪 transformation here and not action 🡪 column filtering/column pruning

Filter() or where() 🡪 row filtering or row pruning

salesdf.select("country", "state").filter("country='India'").show()



Count() is an action.

salesdf.select("country", "state").filter("country='India'").distinct().show() 🡪 applied on entire dataframe

salesdf.select("country", "state").filter("country='India'").dropDuplicates().show() 🡪 allow arguments

Dataframe 🡪 in-persistence

Data is available only at that session. Cannot use data across other sessions.

\*\*Do not use any action eg show() when assigning value to dataframe.

spark.sparkContext.uiWebUrl

Tasks is based on no. of partitions.

Writing data back in target after the process.

sales\_india\_df.write.format("json").save("/home/labuser/spartpout/sales\_india")

spark stores data in parquet file format by default.

DAY 2 PYSPARK:

Wordcount.py (notepad code)

Spark submit

Java/scala 🡪 build 🡪

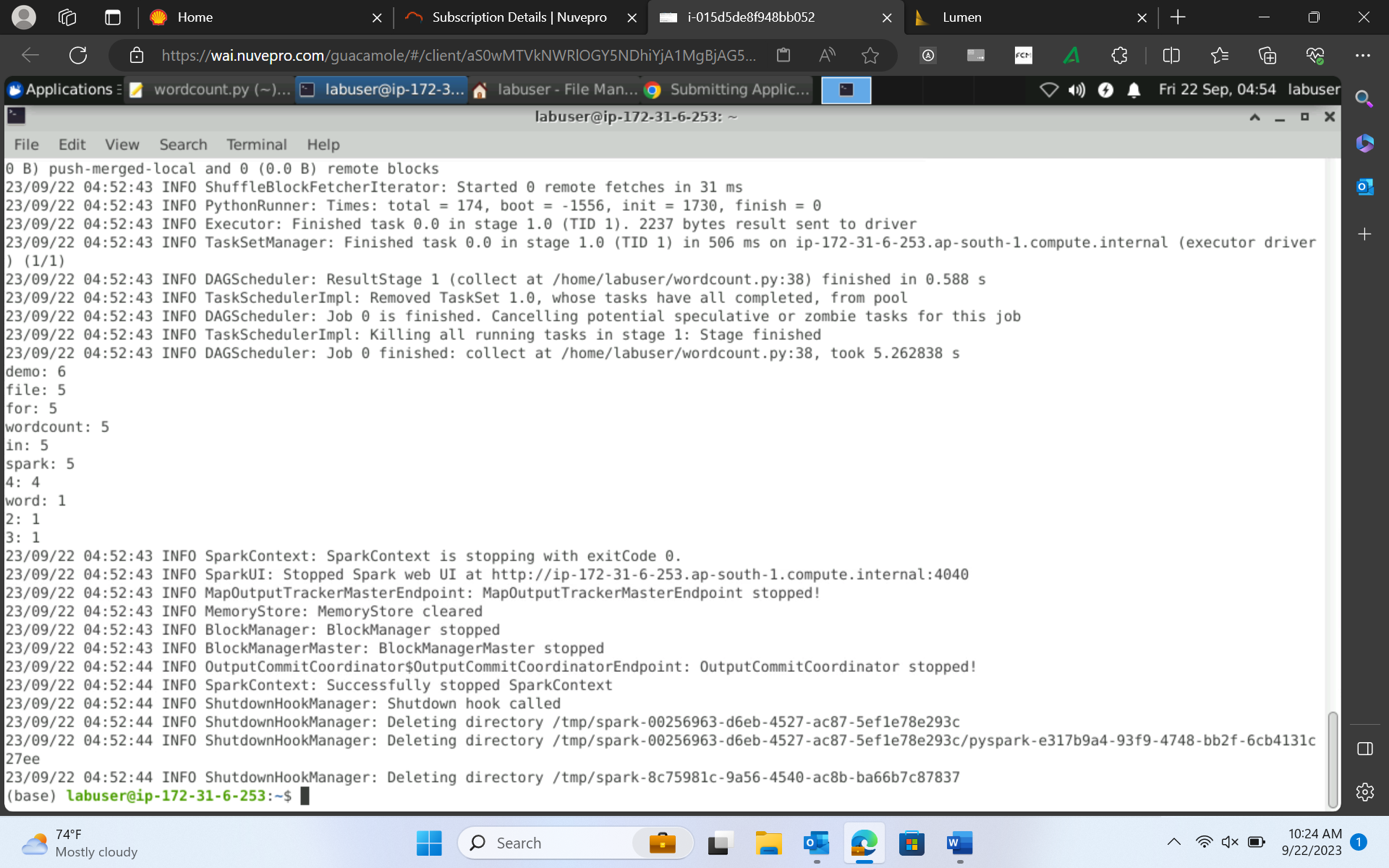
<https://spark.apache.org/docs/latest/submitting-applications.html>

Terminal 🡪 spark-submit

Launched locally from local machine then client node

If launched from the worker’s machine inside your cluster then cluster mode

spark-submit wordcount.py /home/labuser/datasets/sample.txt



spark-submit wordcount.py /home/labuser/datasets/sample.txt --executor-memory 2g --master local[2]

2 is the no of ports. If \* means all ports

\*\*\*\*\*\*\*\*\*\*\*\*\*Handling parallelism in spark

In Salesdf, Partitions 🡪 2

Records 🡪 51290

You can increase/decrease the no of partitions using Repartition(increasing and decreasing) and coalesce(Only decreasing)(cannot increase from the parent dataframe partition count)

salesdf\_repart=salesdf.repartition(5)

salesdf\_repart.write.format("parquet").save("/home/labuser/spartpout/sales\_repart")

salesdf.repartition(8).write.orc("/home/labuser/spartpout/sales\_repart8")

Repartition 🡪 Full shuffling of data happen

Coalesce🡪 no shuffling happens

TempView 🡪 A temporary table 🡪 available only for the session/user

salesdf.createOrReplaceTempView("sales")

spark.sql("show tables").show()

spark.sql("select country, state, profit from sales where country='Australia'").show()

Permanent

Hive Delta table

managed table External table

spark.sql("create database shellida")

spark.sql("show databases").show()

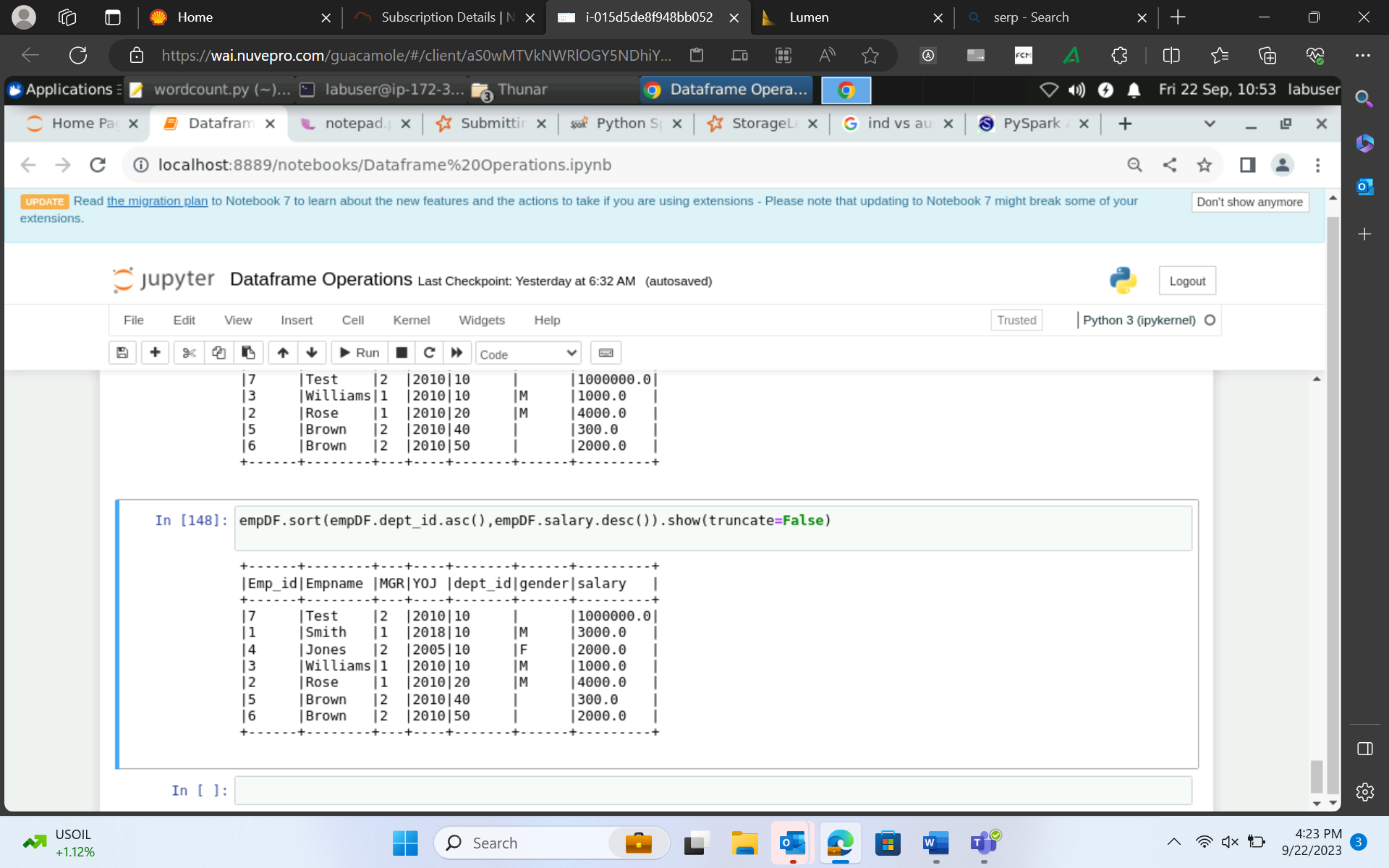
spark.sql("use shellida")

salesdf.write.saveAsTable("shellida.sales\_perm")

spark.sql("show tables").show()

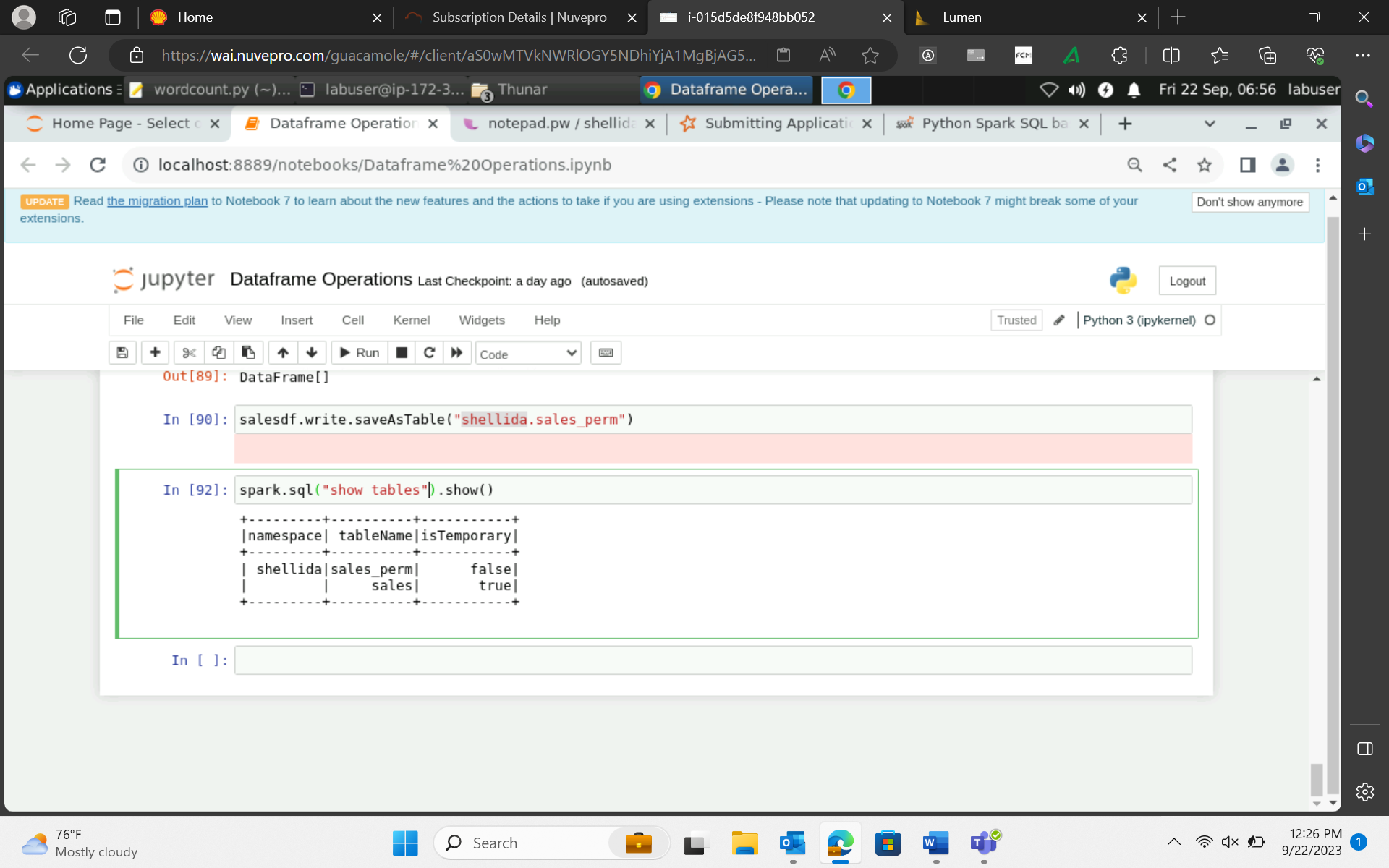
Perform sort on the employee df 🡪 deptno in asc, salary in des

empDF.sort(empDF.dept\_id.asc(),empDF.salary.desc()).show(truncate=False)



Perform aggregation on the employee

Deptname, sum of salary



Pyspark 🡪 python

Spark-sql 🡪 sql

Spark-shell 🡪 scala

Spark-submit 🡪 spark jobs

Java 🡪 IDE 🡪 Eclipse, IntelliJ, Netbeans

Dataframe cache

LRU 🡪 Least Read Used 🡪 evicted to the disk

salesdf.cache()

Any dataframe or processed data needs to be used recursively in your job, then cache the data.

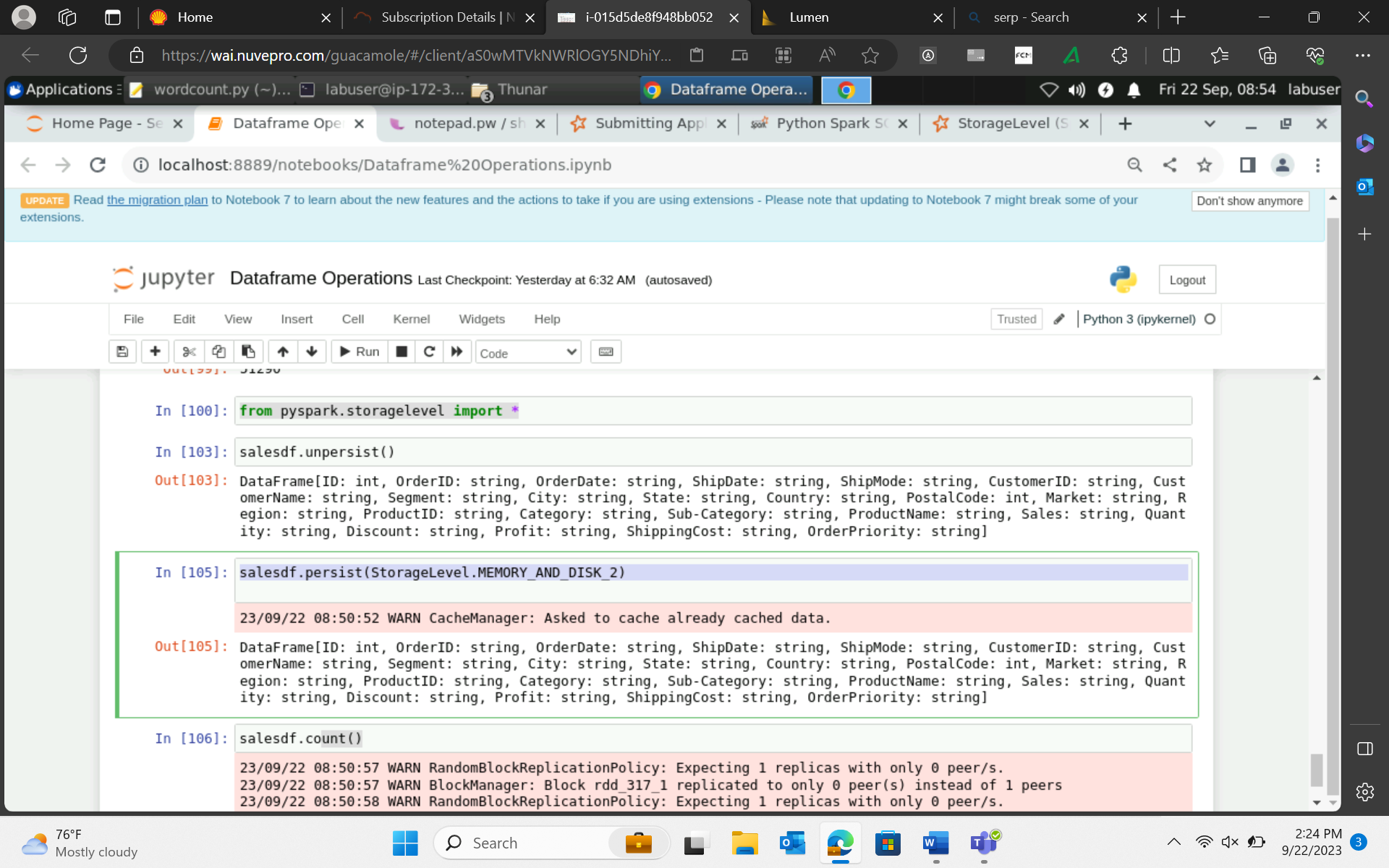
Optimal performance 🡪 cache 🡪 persist –storage level

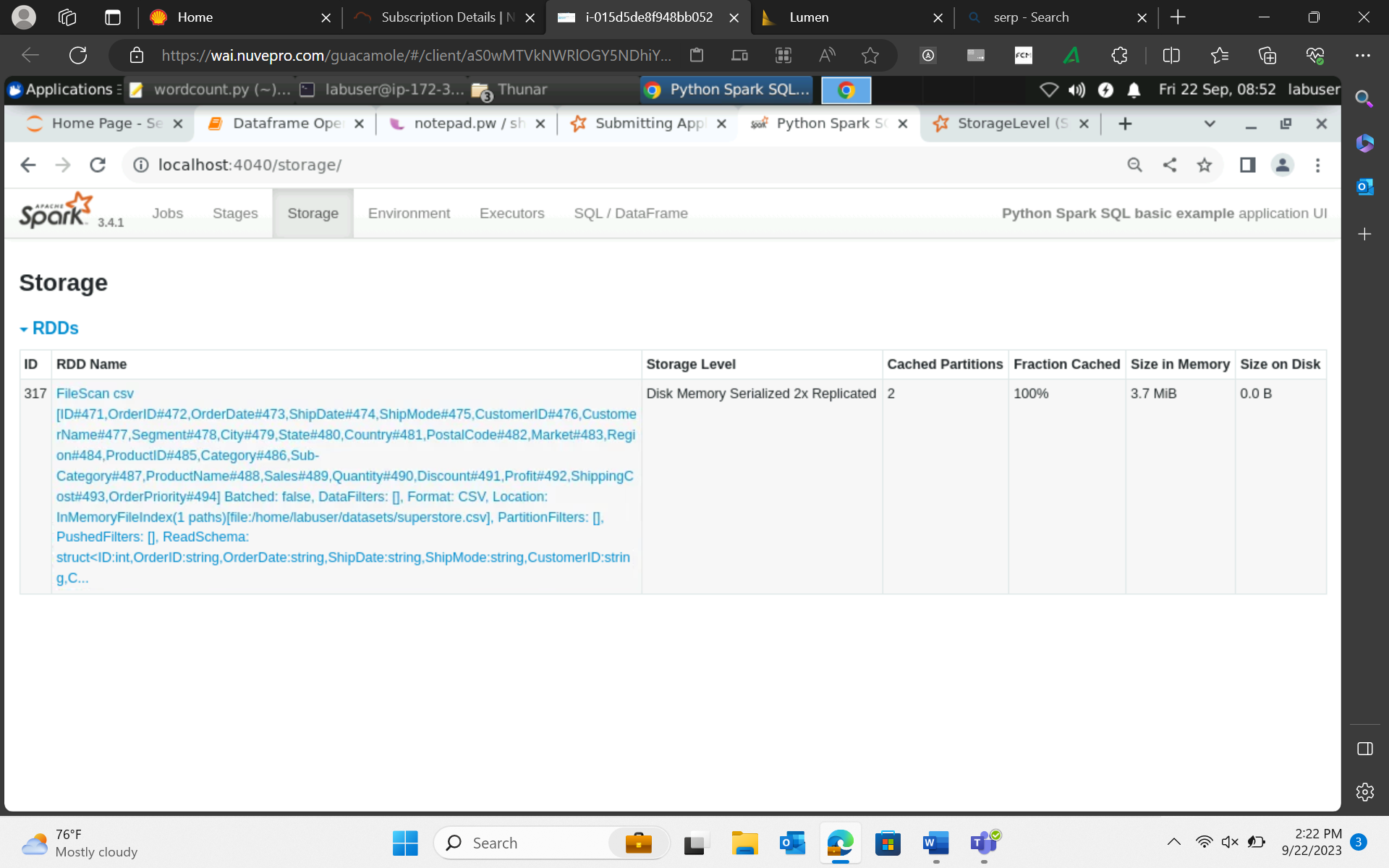
Cache, persist and unpersist

<https://spark.apache.org/docs/3.2.0/api/java/org/apache/spark/storage/StorageLevel.html>



from pyspark.storagelevel import \*

salesdf.persist(StorageLevel.MEMORY\_AND\_DISK\_2) 



emp = [(1,"Smith",1,"2018","10","M",3000.00),

(2,"Rose",1,"2010","20","M",4000.00),

(3,"Williams",1,"2010","10","M",1000.00),

(4,"Jones",2,"2005","10","F",2000.00),

(5,"Brown",2,"2010","40","",300.00),

(6,"Brown",2,"2010","50","",2000.00)

]

EmpSchema = StructType([

StructField('Emp\_id', IntegerType(), True),

StructField('Empname', StringType(), True),

StructField('MGR', IntegerType(), True),

StructField('YOJ', StringType(), True),

StructField('dept\_id', StringType(), True),

StructField('gender', StringType(), True),

StructField('salary', DoubleType(), True)

])

empDF = spark.createDataFrame(data=emp, schema = EmpSchema)

dept = [("Finance",10),

("Marketing",20),

("Sales",30),

("IT",40)

]

deptColumns = ["dept\_name","dept\_id"]

deptDF = spark.createDataFrame(data=dept, schema = deptColumns)

println("Inner join")

empDF.join(deptDF,empDF.dept\_id == deptDF.dept\_id,"inner").show()

Joins :

1. Inner
2. Outer
3. Left Outer
4. Right Outer
5. Cross/Cartesian
6. Leftanti
7. Leftsemi