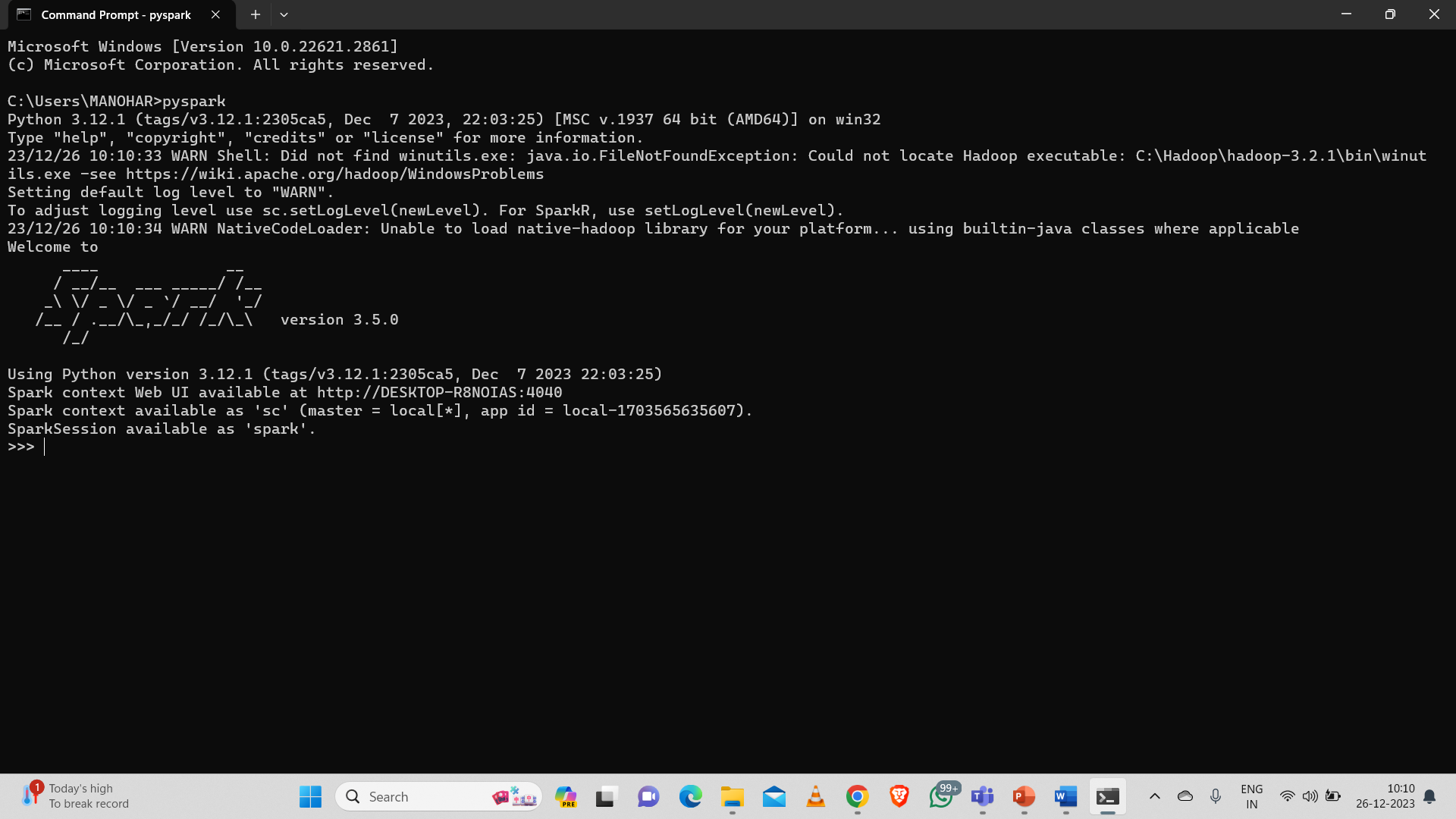
**Sunkari Manohar**

**Assessment 3 – 23/12/2023**

Open command prompt and type pyspark and it will starts the pyspark shell as below:



**Initializing SparkSession:**

A SparkSession can be used create DataFrame, register DataFrame as tables, execute SQL over tables, cache tables, and read parquet files.

from pyspark.sql import SparkSession

spark = SparkSession \

.builder \

.appName("Python Spark SQL basic example") \

.config("spark.some.config.option", "some-value") \

.getOrCreate()

**Creating DataFrames:**

**From RDDs:**

from pyspark.sql.types import \*

**Infer Schema**

sc = spark.sparkContext

lines = sc.textFile("people.txt")

parts = lines.map(lambda l: l.split(","))

people = parts.map(lambda p: Row(name=p[0],age=int(p[1])))

peopledf = spark.createDataFrame(people)

**Specify Schema**

people = parts.map(lambda p: Row(name=p[0],

age=int(p[1].strip())))

schemaString = "name age"

fields = [StructField(field\_name, StringType(), True) for field\_name in schemaString.split()]

schema = StructType(fields)

**From Spark Data Sources:**

**JSON**

df = spark.read.json("customer.json")

df.show()

**Parquet files**

df3 = spark.read.load("users.parquet")

**TXT files**

df4 = spark.read.text("people.txt")

**Inspect Data:**

df.dtypes - Return df column names and data types

df.show() - Display the content of df

df.head() - Return first n rows

df.first() - Return first row

df.take(2) - Return the first n rows

df.schema - Return the schema of df

df.describe().show() - Compute summary statistics

df.columns - Return the columns of df

df.count() - Count the number of rows in df

df.distinct().count() - Count the number of distinct rows in df

df.printSchema() - Print the schema of df

df.explain() - Print the (logical and physical) plans

**Queries in pyspark**:

from pyspark.sql import functions as F

Select df.select("firstName").show()

It shows all the data present in firstname column

df.select("firstName" , "lastName").show()

It shows all entries of firstname and lastname

df.select("firstName" , "age" ,explode("phoneNumber") .alias("contac tInfo")) .select("contactInfo.type" , "firstName" , "age") .show()

It shows all entries of firstname and age and type df.select(df["firstName"],df["age"]+ 1).show()

It shows firstname and age increased by 1

df.select(df['age'] > 24).show()

It shows all entries where age greater than 24

df.select("firstName" ,F.when(df.age >30,1) .otherwise(0)) .show()

It shows firstname and 1 if age greater than 30 otherwise 0

df[df.firstName.isin("Jane" , "Boris")].collect()

It shows all firstname if it is in given options

Like

df.select("firstName" ,df.lastName.like("Smith")) .show()

Shows firstname whose last name is smith Startswith - Endswith df.select("firstName" ,df.lastName .startswith("Sm")) .show()

It shows firstname whose last name starts with “sm” df.select(df.lastName.endswith("th")).show()

It shows all entries whose last name ends with “th” Substring df.select(df.firstName.substr(1, 3) .alias("name")).collect()

It selects first 3 letters of firstname as substring under name column Between

df.select(df.age.between(22, 24)) .show()

It shows all entries whose age is between 22 and 24

**Adding Columns:**

we use below syntax to add new column to the table Here we added city, postalcode,state,street address, number to the address table

df = df.withColumn('city',df.address.city)\ .withColumn('postalCode',df.address.postalCode) \ .withColumn('state',df.address.state) \ .withColumn('streetAddress',df.address.streetAddress) \ .withColumn('telePhoneNumber',explode(df.phoneNumber.number)) \ .withColumn('telePhoneType',explode(df.phoneNumber.type))

**Updating columns**:

We use following syntax to update columns here we are updated telePhoneNumber column to phone Number

df = df.withColumnRenamed('telePhoneNumber', 'phoneNumber')

**Removing columns:**

We have two different syntax to drop the column from table Here we are removing column named phoneNumber from adddress

df = df.drop("address", "phoneNumber") df = df.drop(df.address).drop(df.phoneNumber)

**Group By:**

The below query shows the number of entries in the table group by age

df.groupBy("age").count().show()

**Filter**:

It shows the records whose age is greater than 24

df.filter(df["age"]>24).show()

**Sort:**

peopledf.sort(peopledf.age.desc()).collect()

It sorts the age of people dataframes in descending order

df.sort("age", ascending=False).collect() df.orderBy(["age","city"],ascending=[0,1]).collect()

It sorts the age in descending order and city in alphabetical order 0,1 in above syntax represents 0 for false and 1 for true

**Missing and Replacing**:

df.na.fill(50).show()

It returns new df by relacing all null values with 50

df.na.drop().show()

It returns new dataframes by removing all the null values

df.na.replace(10,20).show()

It returns an other df by replacing 10 with 20

DataStructures

rdd1=df.rdd

It converts dataframes into resilent distrubuted dataset

df.toJSON().first()

Converts df into a RDD of string

df.toPandas()

Returns contents of df as pandas data structure

**Broadcast**:

Inorder to join two tables in pyspark we use broadcast from pyspark.sql.functions import broadcast cases=cases.join(broadcast(regions),[‘province’,’city’],how=’left’)

To limit the values to show the output we use limit() function and convert into pandas

Cases.limit(10).toPandas()

It returns only 10 row

**Write and Save Files**:

df.select("firstName", "city").write.save("nameAndCity.parquet")

It writes the data into the file named nameAndCity.perquet and save into it df.select("firstName","age").write.save("namesAndAges.json",format=" json")

It saves the file in json format

**Stoping Spark Session**:

We use below command to stop the spark session

Spark.stop()