Download zipcodes.csv from drive

<https://drive.google.com/file/d/1RgaeZky9L8wfvHpU4x2QltE1Z7eCyc2u/view?usp=sharing>

| df = spark.read.csv("file:///home/zidane/zipcodes.csv")  df.printSchema() |
| --- |

root

|-- \_c0: string (nullable = true)

|-- \_c1: string (nullable = true)

|-- \_c2: string (nullable = true)

|-- \_c3: string (nullable = true)

|-- \_c4: string (nullable = true)

|-- \_c5: string (nullable = true)

|-- \_c6: string (nullable = true)

|-- \_c7: string (nullable = true)

|-- \_c8: string (nullable = true)

|-- \_c9: string (nullable = true)

|-- \_c10: string (nullable = true)

|-- \_c11: string (nullable = true)

|-- \_c12: string (nullable = true)

|-- \_c13: string (nullable = true)

|-- \_c14: string (nullable = true)

|-- \_c15: string (nullable = true)

|-- \_c16: string (nullable = true)

|-- \_c17: string (nullable = true)

|-- \_c18: string (nullable = true)

|-- \_c19: string (nullable = true)

| df.show() |
| --- |

### **Using Header Record For Column Names**

| df2 = spark.read.option("header",True).csv("file:///home/zidane/zipcodes.csv") |
| --- |

As mentioned earlier, PySpark reads all columns as a string (StringType) by default.

## **Options While Reading CSV File**

| df3 = spark.read.options(delimiter=',').csv("file:///home/zidane/zipcodes.csv") |
| --- |

### **InferSchema**

The default value set to this option is False when setting to true it automatically infers column types based on the data. Note that, it requires reading the data one more time to infer the schema.

| df4 = spark.read.options(inferSchema='True',delimiter=',') \  .csv("file:///home/zidane/zipcodes.csv") |
| --- |

Another way

| df4 = spark.read.option("inferSchema",True) \  .option("delimiter",",") \  .csv("file:///home/zidane/zipcodes.csv") |
| --- |

|-- \_c0: string (nullable = true)

|-- \_c1: string (nullable = true)

|-- \_c2: string (nullable = true)

|-- \_c3: string (nullable = true)

|-- \_c4: string (nullable = true)

|-- \_c5: string (nullable = true)

|-- \_c6: string (nullable = true)

|-- \_c7: string (nullable = true)

|-- \_c8: string (nullable = true)

|-- \_c9: string (nullable = true)

|-- \_c10: string (nullable = true)

|-- \_c11: string (nullable = true)

|-- \_c12: string (nullable = true)

|-- \_c13: string (nullable = true)

|-- \_c14: string (nullable = true)

|-- \_c15: string (nullable = true)

|-- \_c16: string (nullable = true)

|-- \_c17: string (nullable = true)

|-- \_c18: string (nullable = true)

|-- \_c19: string (nullable = true)

### **header**

This option is used to read the first line of the CSV file as column names. By default the value of this option is False , and all column types are assumed to be a string.

| df3 = spark.read.options(header='True', inferSchema='True', delimiter=',') \  .csv("file:///home/zidane/zipcodes.csv")  df3.printSchema() |
| --- |

|-- RecordNumber: integer (nullable = true)

|-- Zipcode: integer (nullable = true)

|-- ZipCodeType: string (nullable = true)

|-- City: string (nullable = true)

|-- State: string (nullable = true)

|-- LocationType: string (nullable = true)

|-- Lat: double (nullable = true)

|-- Long: double (nullable = true)

|-- Xaxis: double (nullable = true)

|-- Yaxis: double (nullable = true)

|-- Zaxis: double (nullable = true)

|-- WorldRegion: string (nullable = true)

|-- Country: string (nullable = true)

|-- LocationText: string (nullable = true)

|-- Location: string (nullable = true)

|-- Decommisioned: boolean (nullable = true)

|-- TaxReturnsFiled: integer (nullable = true)

|-- EstimatedPopulation: integer (nullable = true)

|-- TotalWages: integer (nullable = true)

|-- Notes: string (nullable = true)

## **Reading CSV files with a user-specified custom schema**

If you know the schema of the file ahead and do not want to use the inferSchema option for column names and types

| import pyspark  from pyspark.sql import SparkSession  from pyspark.sql.types import \*  schema = StructType() \  .add("RecordNumber",IntegerType(),True) \  .add("Zipcode",IntegerType(),True) \  .add("ZipCodeType",StringType(),True) \  .add("City",StringType(),True) \  .add("State",StringType(),True) \  .add("LocationType",StringType(),True) \  .add("Lat",DoubleType(),True) \  .add("Long",DoubleType(),True) \  .add("Xaxis",IntegerType(),True) \  .add("Yaxis",DoubleType(),True) \  .add("Zaxis",DoubleType(),True) \  .add("WorldRegion",StringType(),True) \  .add("Country",StringType(),True) \  .add("LocationText",StringType(),True) \  .add("Location",StringType(),True) \  .add("Decommisioned",BooleanType(),True) \  .add("TaxReturnsFiled",StringType(),True) \  .add("EstimatedPopulation",IntegerType(),True) \  .add("TotalWages",IntegerType(),True) \  .add("Notes",StringType(),True)    df\_with\_schema = spark.read.format("csv") \  .option("header", True) \  .schema(schema) \  .load("file:///home/zidane/zipcodes.csv") |
| --- |

### **Saving modes**

PySpark DataFrameWriter also has a method mode() to specify saving mode.

overwrite – mode is used to overwrite the existing file.

| df\_with\_schema.write.mode('overwrite').csv("file:///tmp/spark\_output/zipcodes")  //you can also use this  df\_with\_schema.write.format("csv").mode('overwrite').save("file:///tmp/spark\_output/zipcodes") |
| --- |

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* **Parquet** \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

## **Pyspark Write DataFrame to Parquet file format**

| data =[("James ","","Smith","36636","M",3000),  ("Michael ","Rose","","40288","M",4000),  ("Robert ","","Williams","42114","M",4000),  ("Maria ","Anne","Jones","39192","F",4000),  ("Jen","Mary","Brown","","F",-1)]  columns=["firstname","middlename","lastname","dob","gender","salary"]  df=spark.createDataFrame(data,columns)  df.write.parquet("file:///tmp/output/people.parquet") |
| --- |

## **Read Parquet file into DataFrame**

| parDF=spark.read.parquet("file:///tmp/output/people.parquet") |
| --- |

## **SQL queries DataFrame**

| parDF.createOrReplaceTempView("ParquetTable")  parkSQL = spark.sql("select \* from ParquetTable where salary >= 4000 ")  parkSQL.show() |
| --- |

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* **ORC** \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

| parDF.write.orc("file:///tmp/orc/data.orc") |
| --- |

| df = spark.read.orc("file:///tmp/orc/data.orc")  df.printSchema()  df.show() |
| --- |

**Run sql on Orc File**

| df.createOrReplaceTempView("ORCTable")  orcSQL = spark.sql("select firstname,dob from ORCTable where salary >= 4000 ")  orcSQL.show() |
| --- |

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***Json**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

| # Read JSON file into dataframe  parDF.write.json("file:///tmp/json/data.json")  df = spark.read.json("file:///tmp/json/data.json")  df.printSchema()  df.show() |
| --- |