**Pyspark Coding Challenge**

**Name: PRIYESHWAR**

**Mail:** [**priyesh2664@gmail.com**](mailto:priyesh2664@gmail.com)

***Spark Initialization & Data Loading***

Initializes a Spark session and loads the loan dataset with headers and inferred data types.

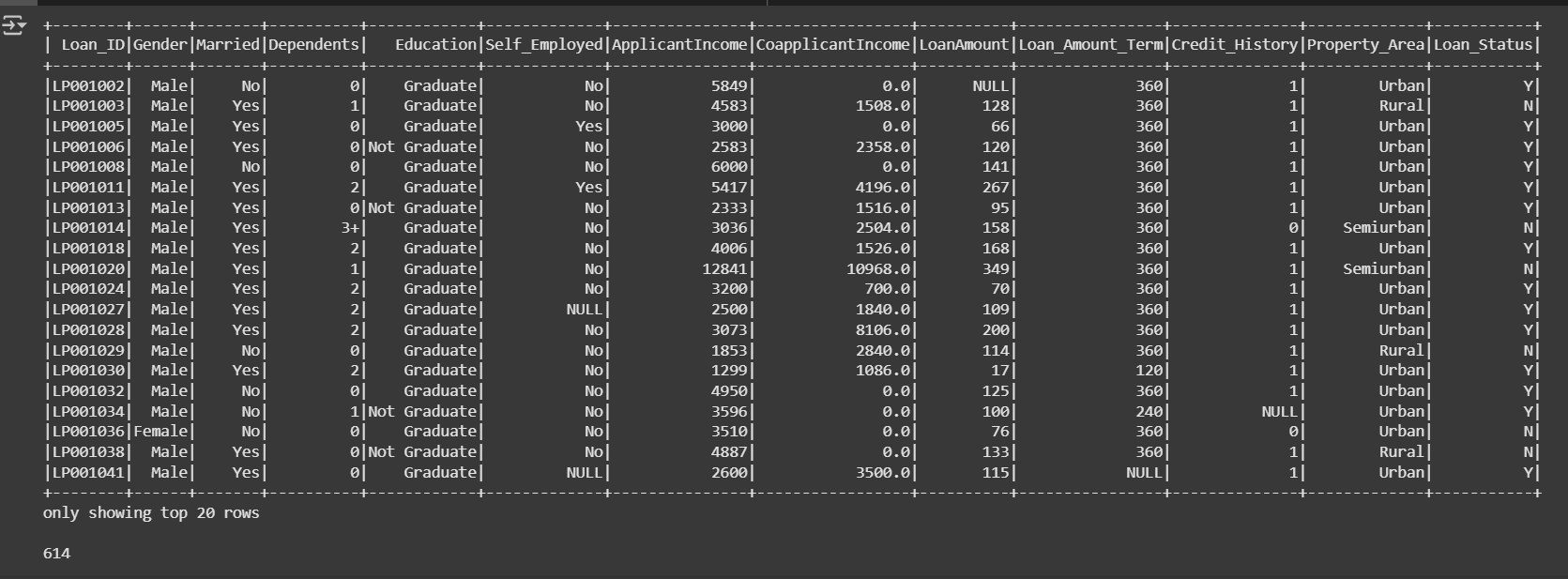
show() displays data; count() gives total number of records.

**from pyspark.sql import SparkSession**

**spark = SparkSession.builder.appName('Coding\_Challenge').getOrCreate()**

**df = spark.read.csv("/content/LoanData (1).csv", header=True, inferSchema=True)**

**df.show()**

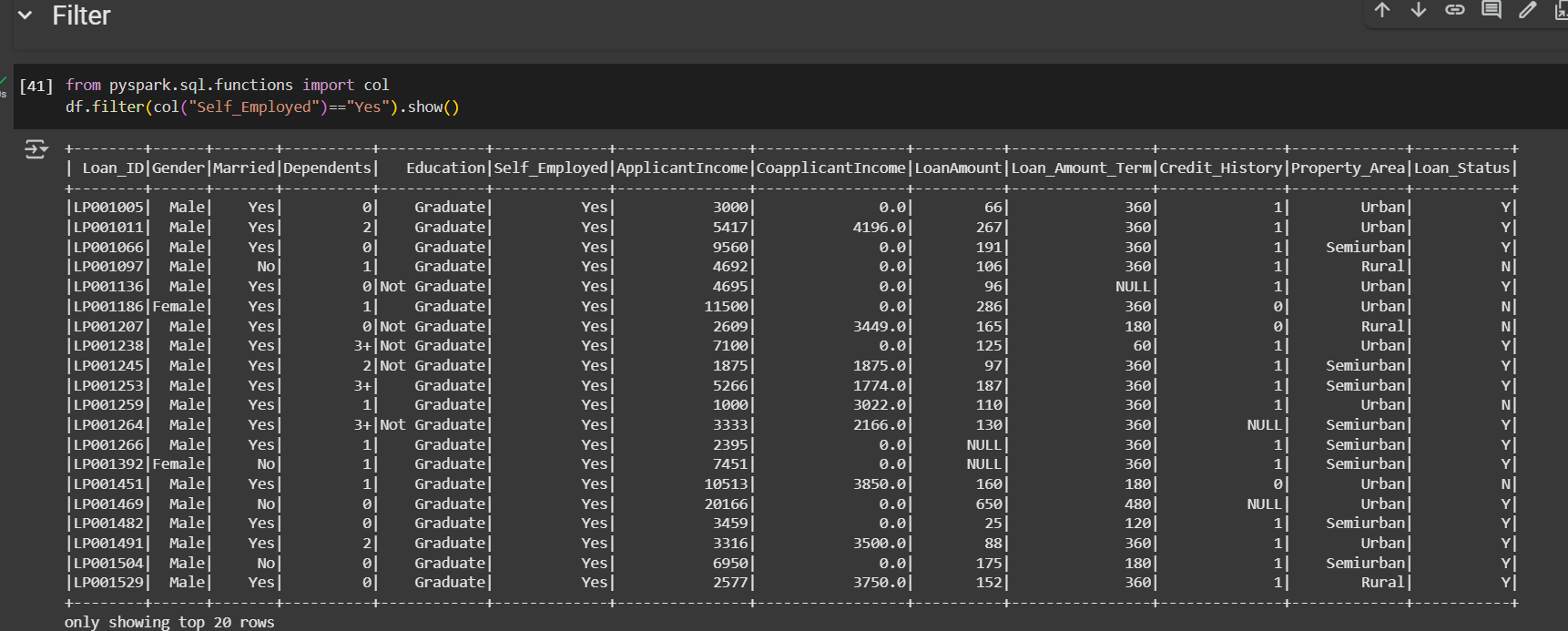
****

***Transformations***

1. ***Filter***

Filters the dataset to show only self-employed applicants.

**df.filter(col("elf\_Employed")=="Yes").show()**

****

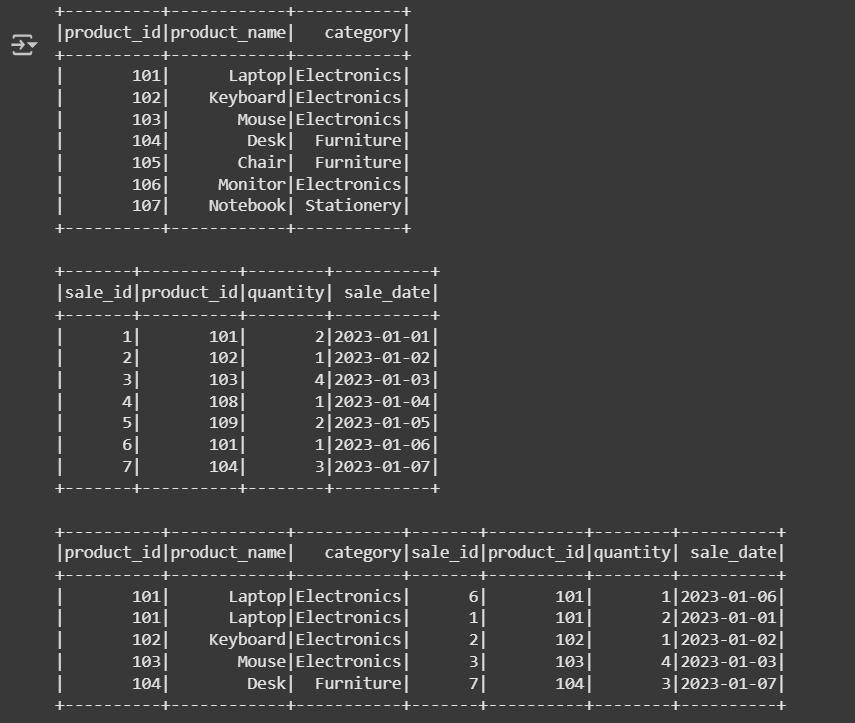
1. ***Join***

Joins product and sales datasets on product\_id to combine related records.

**ProductDf = spark.read.csv("/content/products.csv", header=True, inferSchema=True)**

**salesDf = spark.read.csv("/content/sales.csv", header=True, inferSchema=True)**

**ProductDf.join(salesDf, ProductDf.product\_id == salesDf.product\_id).show()**

****

1. ***Simple Aggregations***

Calculates total, maximum, minimum, and average applicant income.

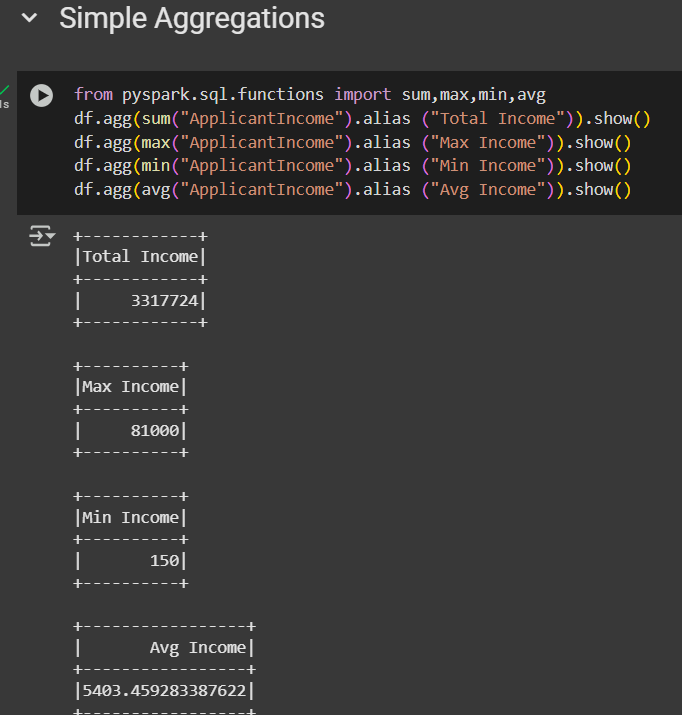
**from pyspark.sql.functions import sum, max, min, avg**

**df.agg(sum("ApplicantIncome").alias("Total Income")).show()**

**df.agg(max("ApplicantIncome").alias("Max Income")).show()**

**df.agg(min("ApplicantIncome").alias("Min Income")).show()**

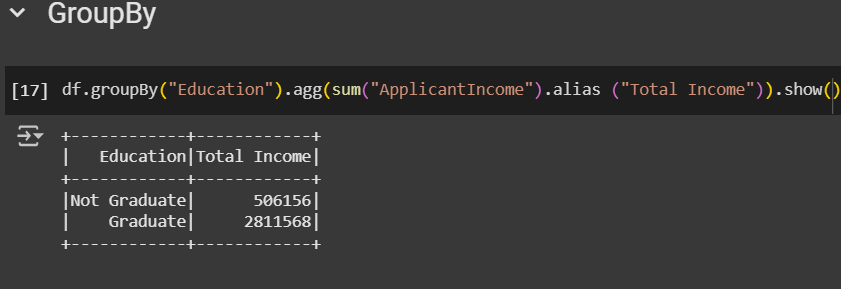
**df.agg(avg("ApplicantIncome").alias("Avg Income")).show()**

****

1. ***GroupBy Aggregation***

Groups data by education level and calculates total income per group.

**df.groupBy("Education").agg(sum("ApplicantIncome").alias("Total Income")).show()**

****

1. ***Window Function***

Assigns a rank to each applicant's income within their property area using windowing.

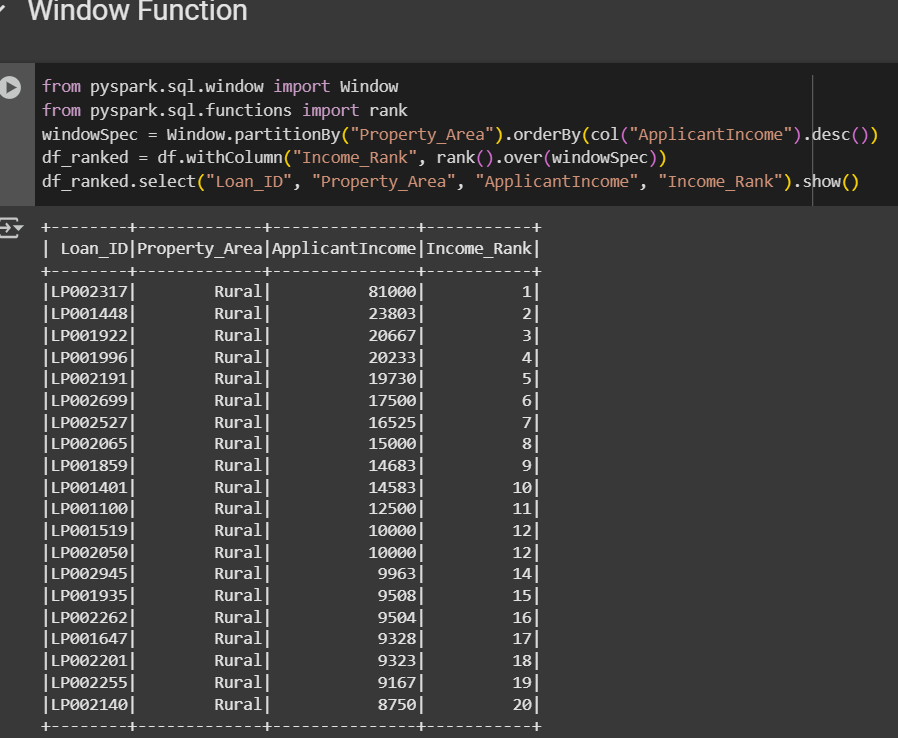
**from pyspark.sql.window import Window**

**from pyspark.sql.functions import rank**

**windowSpec = Window.partitionBy("Property\_Area").orderBy(col("ApplicantIncome").desc())**

**df\_ranked = df.withColumn("Income\_Rank", rank().over(windowSpec))**

**df\_ranked.select("Loan\_ID", "Property\_Area", "ApplicantIncome", "Income\_Rank").show()**

****

1. ***sortBy (RDD Transformation)***

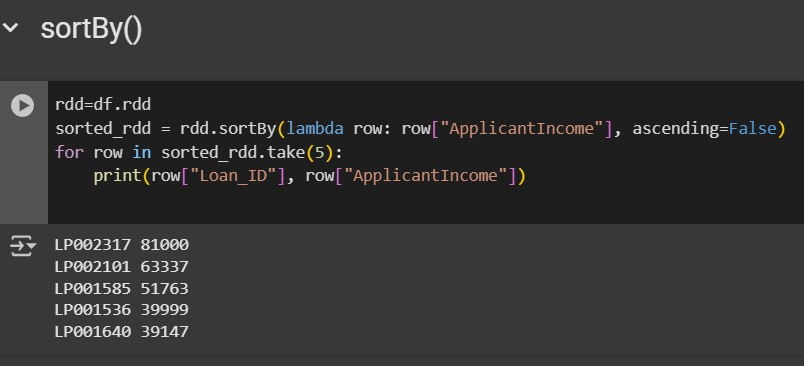
Converts DataFrame to RDD and sorts applicants by income in descending order.

**rdd = df.rdd**

**sorted\_rdd = rdd.sortBy(lambda row: row["ApplicantIncome"], ascending=False)**

**for row in sorted\_rdd.take(5):**

**print(row["Loan\_ID"], row["ApplicantIncome"])**

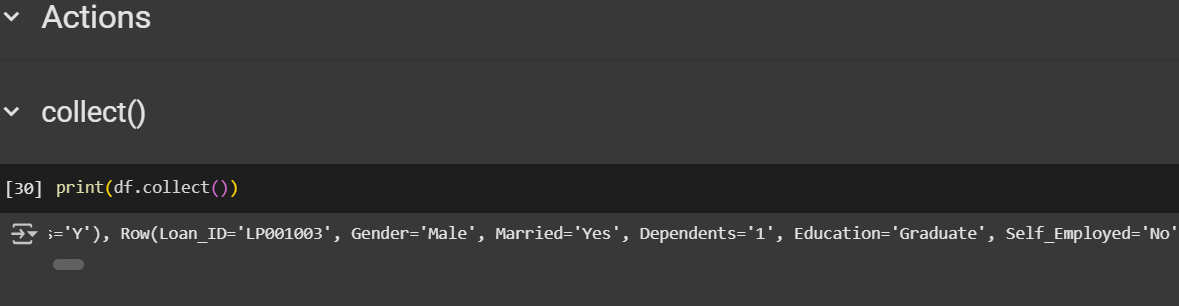
****

***Actions***

1. ***collect()***

Retrieves the entire DataFrame into driver memory (use cautiously with large data).

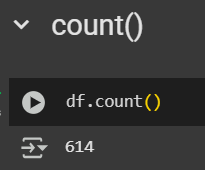
**print(df.collect())**

****

1. ***count()***

Returns the number of records in the DataFrame.

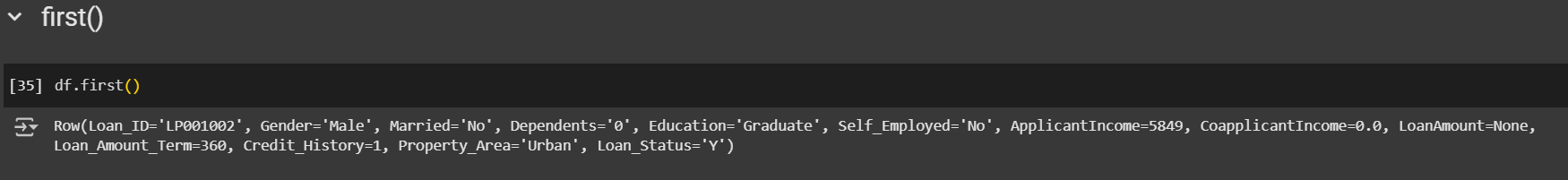
**df.count()**

****

1. ***first()***

Retrieves the first row from the DataFrame.

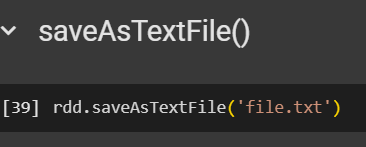
**df.first()**

****

1. ***saveAsTextFile()***

Saves the RDD content as a text file to the specified location.

**rdd.saveAsTextFile('file.txt')**

****

****