**PYSPARK ASSIGNMENT**

PySpark RDD (Resilient Distributed Dataset) is the foundational data structure in Apache Spark, a distributed computing framework for large-scale data processing. RDDs are immutable, partitioned collections of objects that can be processed in parallel across a cluster of machines. Here are some key theoretical aspects of PySpark RDD operations:

**1. Immutability:**

RDDs are immutable, meaning their contents cannot be changed once created. If you want to transform an RDD, you create a new RDD with the desired transformations.

Immutability ensures fault tolerance by allowing the recreation of lost data by recomputing transformations.

**2. Resilience:’**

The term "Resilient" in RDD stands for fault tolerance. RDDs automatically recover from node failures by recomputing lost partitions.

Lineage information is stored for each RDD, allowing Spark to reconstruct lost data by recomputing from the original data source.

**3. Parallel Processing:**

RDDs support parallel processing across a cluster of machines. Spark automatically distributes the data and computation across nodes in the cluster.

Each partition of an RDD can be processed independently, enabling parallel execution of transformations.

**4. Transformations and Actions:**

RDD operations can be broadly categorized into transformations and actions.

Transformations are operations that create a new RDD from an existing one, like map, filter, union, groupByKey, etc.

Actions are operations that return a value to the driver program or write data to an external storage system, such as collect, count, saveAsTextFile, etc.

**5. Lazy Evaluation:**

Spark uses lazy evaluation, which means that transformations on RDDs are not executed immediately. Instead, they are recorded as a sequence of transformations to be applied later.

Spark only computes the result when an action is triggered. This optimises the execution plan and reduces unnecessary computation.

**6. Narrow and Wide Transformations:**

Transformations are categorised into narrow (e.g., map, filter) and wide (e.g., groupByKey, reduceByKey) transformations.

Narrow transformations result in dependencies between a single parent partition and its child partitions, while wide transformations involve dependencies between multiple partitions.

**7. Persistence (Caching):**

RDDs can be persisted in memory or on disk to avoid recomputation, especially for iterative algorithms.

The cache or persist methods can be used to store intermediate results, improving the performance of iterative algorithms.

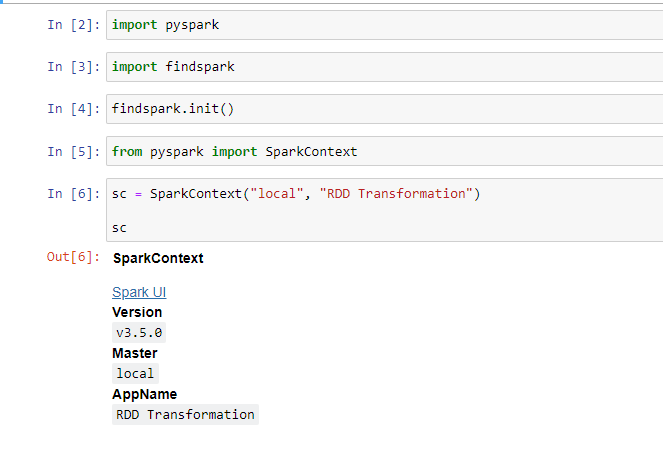
**8. Partitioning:**

RDDs are divided into partitions, which are the basic units of parallelism.

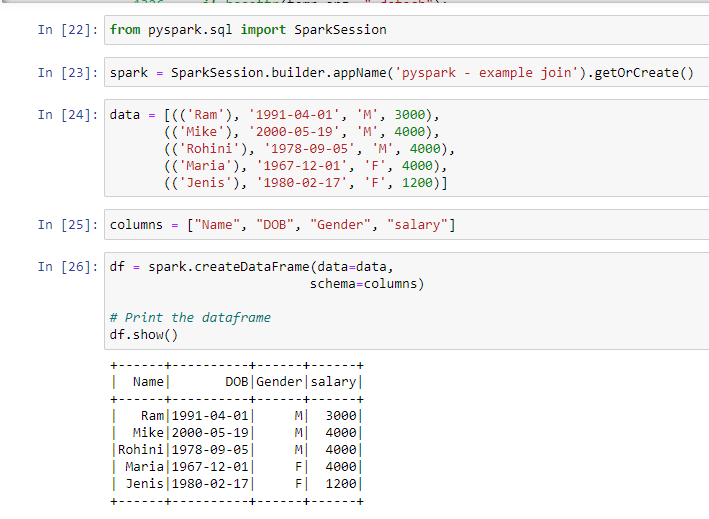
The number of partitions affects parallelism and performance. Partitioning can be controlled using the repartition and coalesce methods.

**9. Broadcast Variables:**

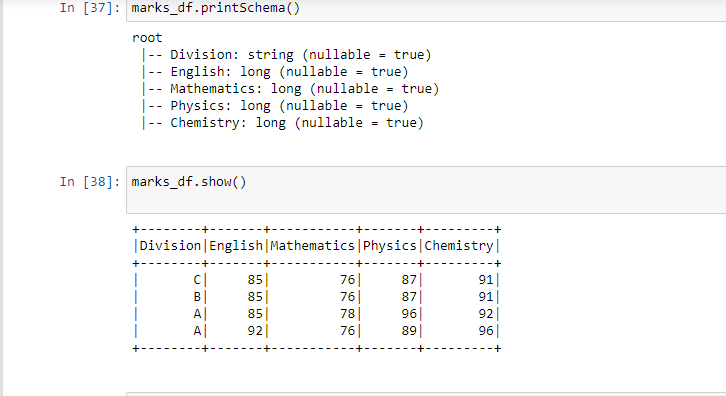
Broadcast variables are read-only variables cached on each worker node, reducing the need to transfer large read-only data sets over the network.

****

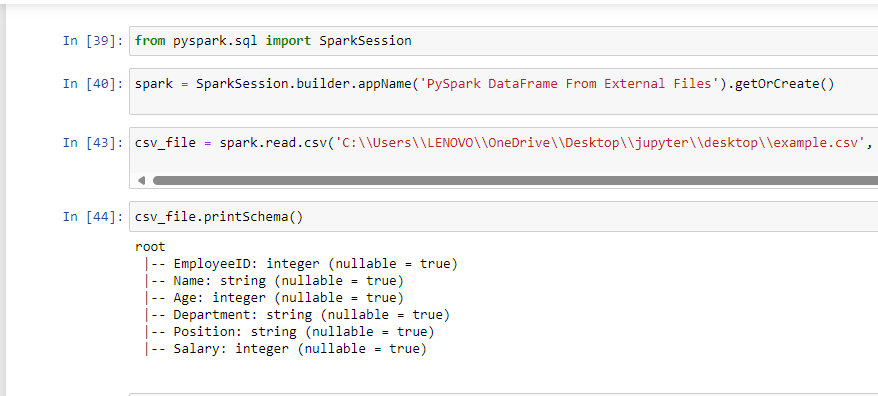
**CREATE DATAFRAME**

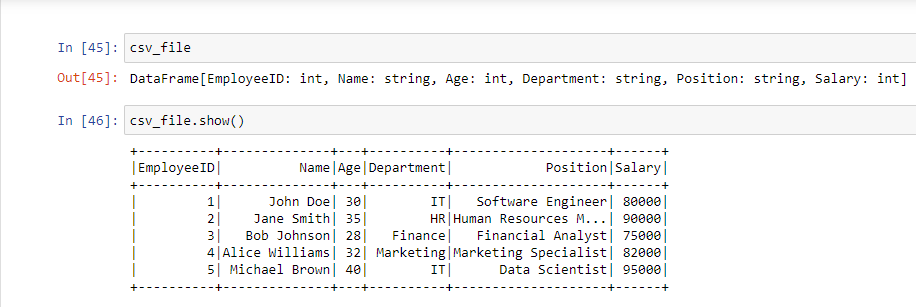
****

****

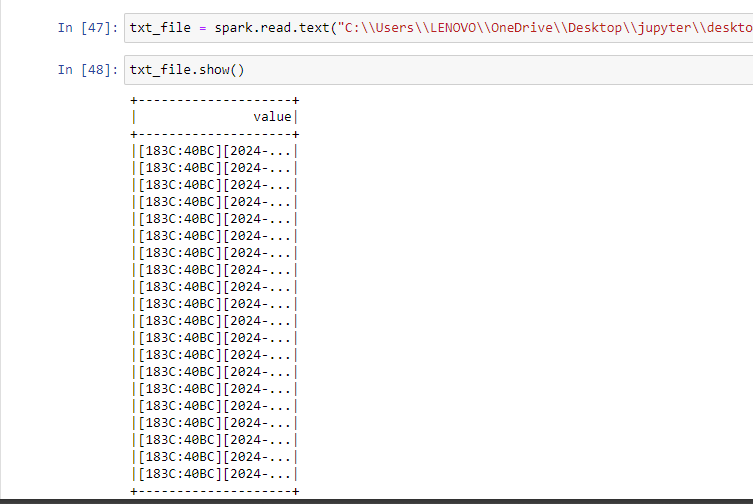
****

**READ CSV**

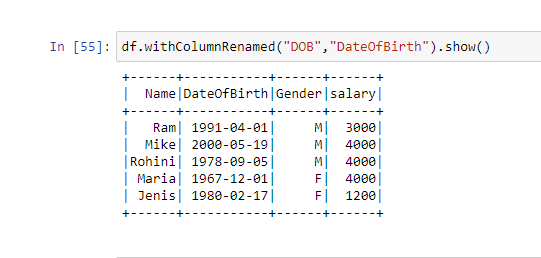
****

****

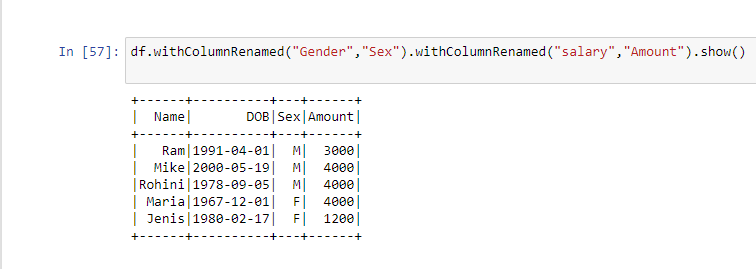
**READ TEXT FILE**

****

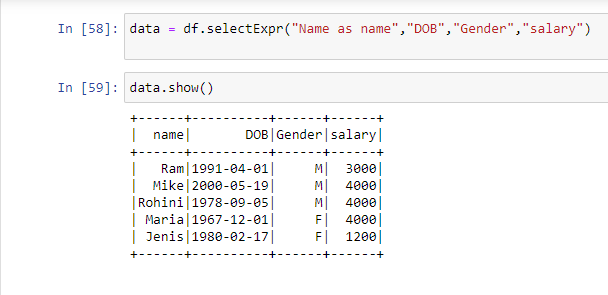
**COLUMN READ**

****

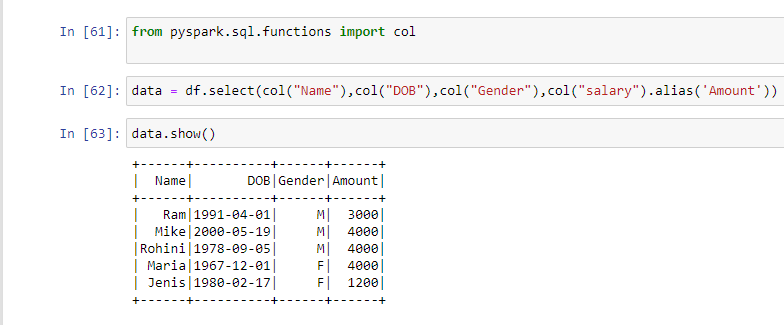
**COLUMN RENAME**

****

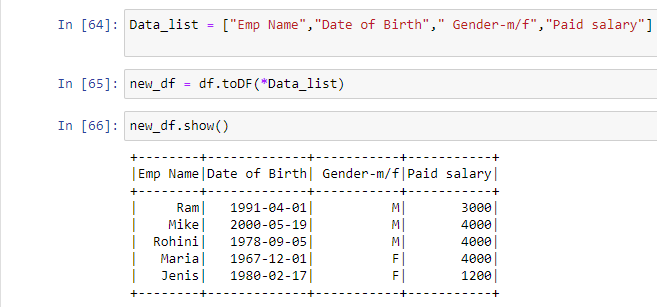
**SELECT**

****

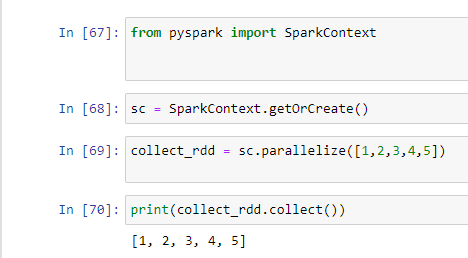
**SELECT COLUMN**

****

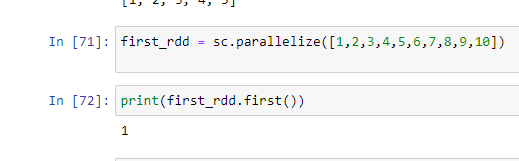
**CONVERT TO DATA FRAME**

****

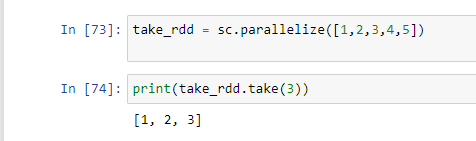
**COLLECT**

****

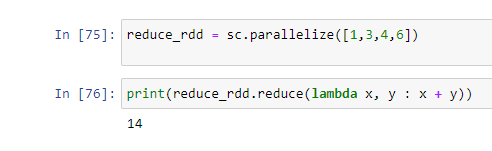
**FIRST**

****

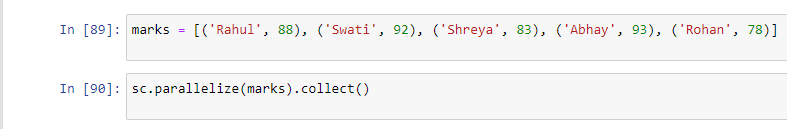
**TAKE**

****

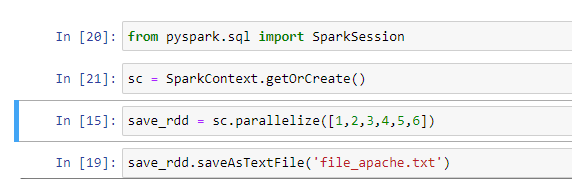
**REDUCE**

****

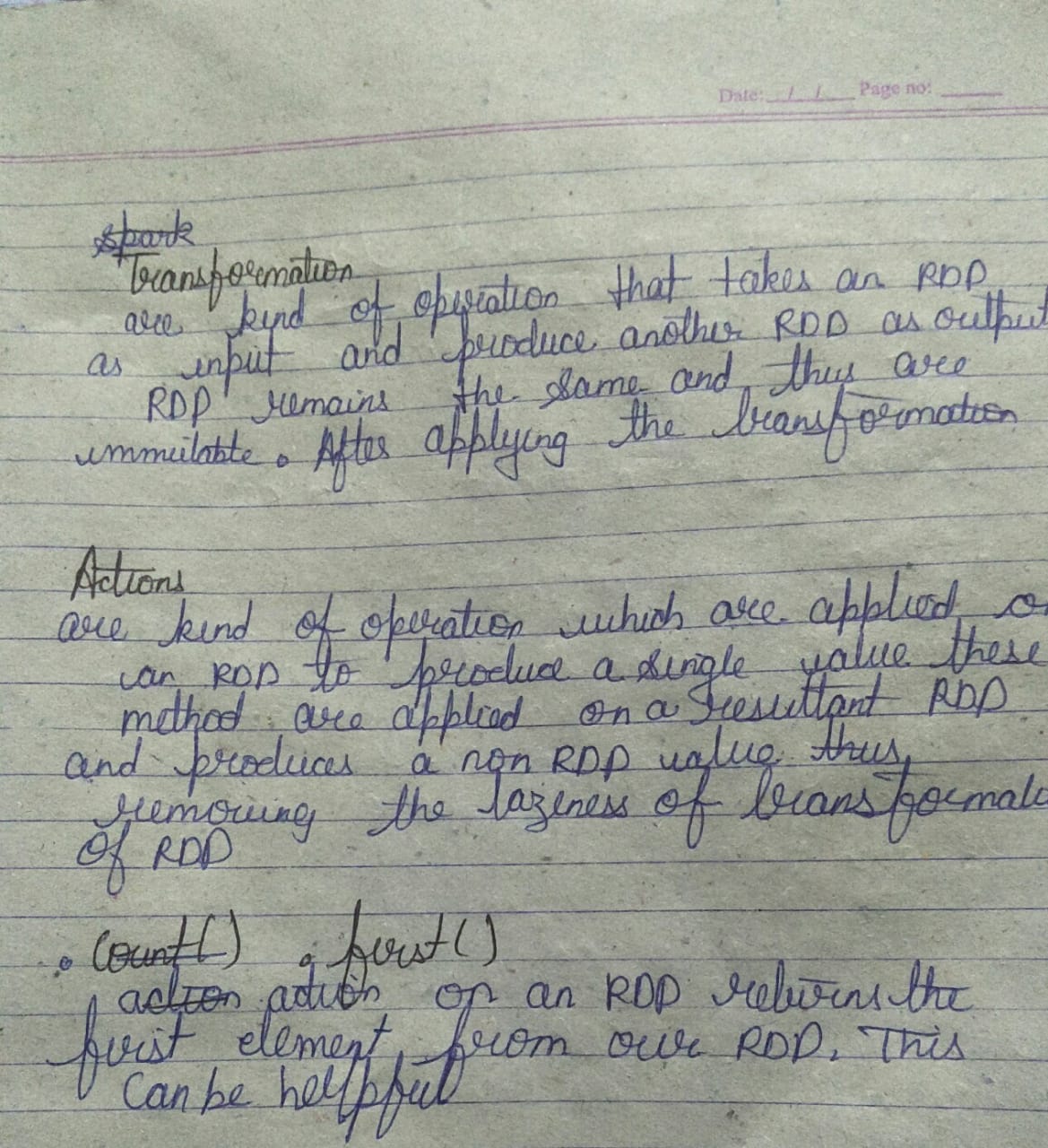
**COLLECT**

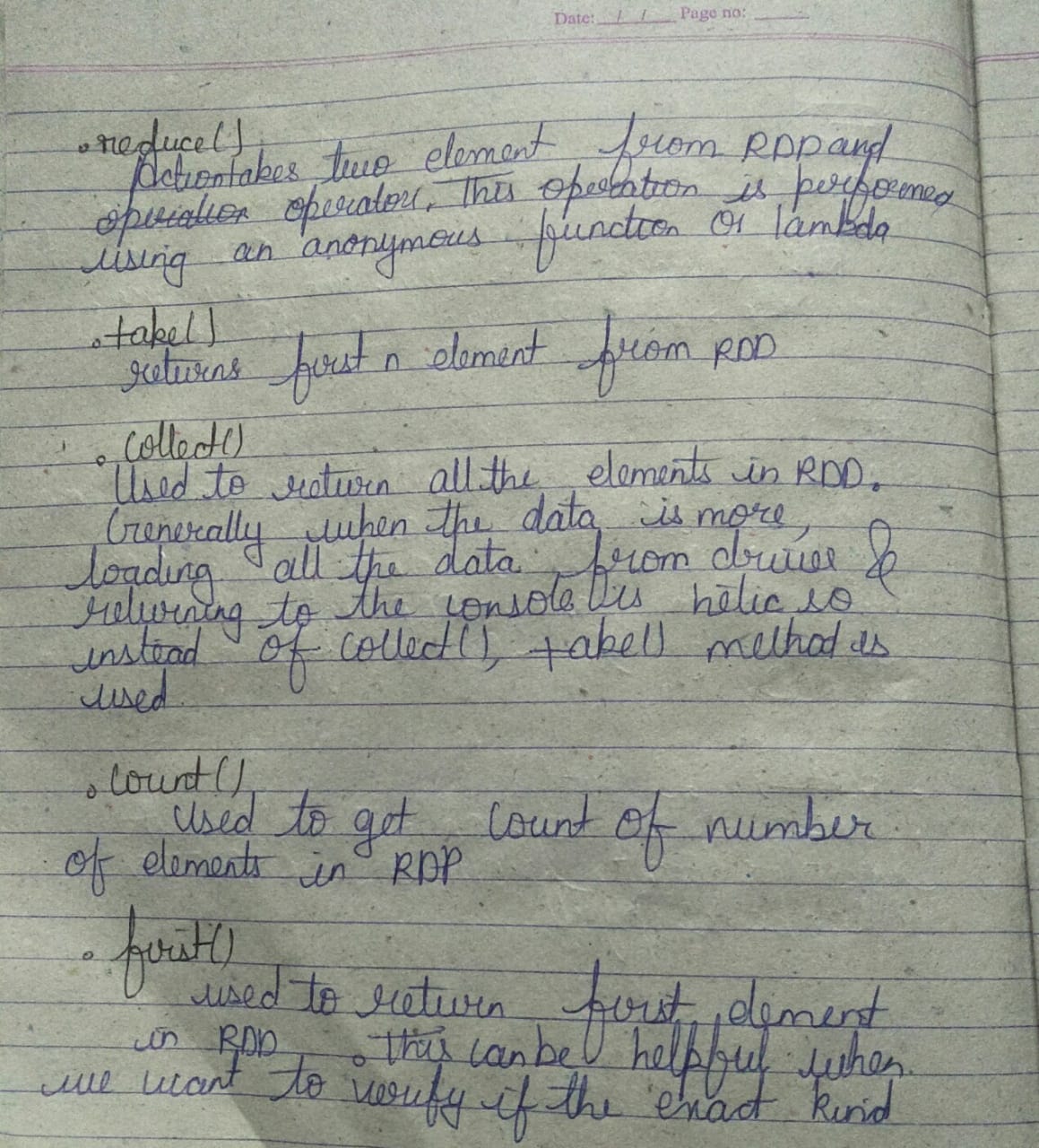
****

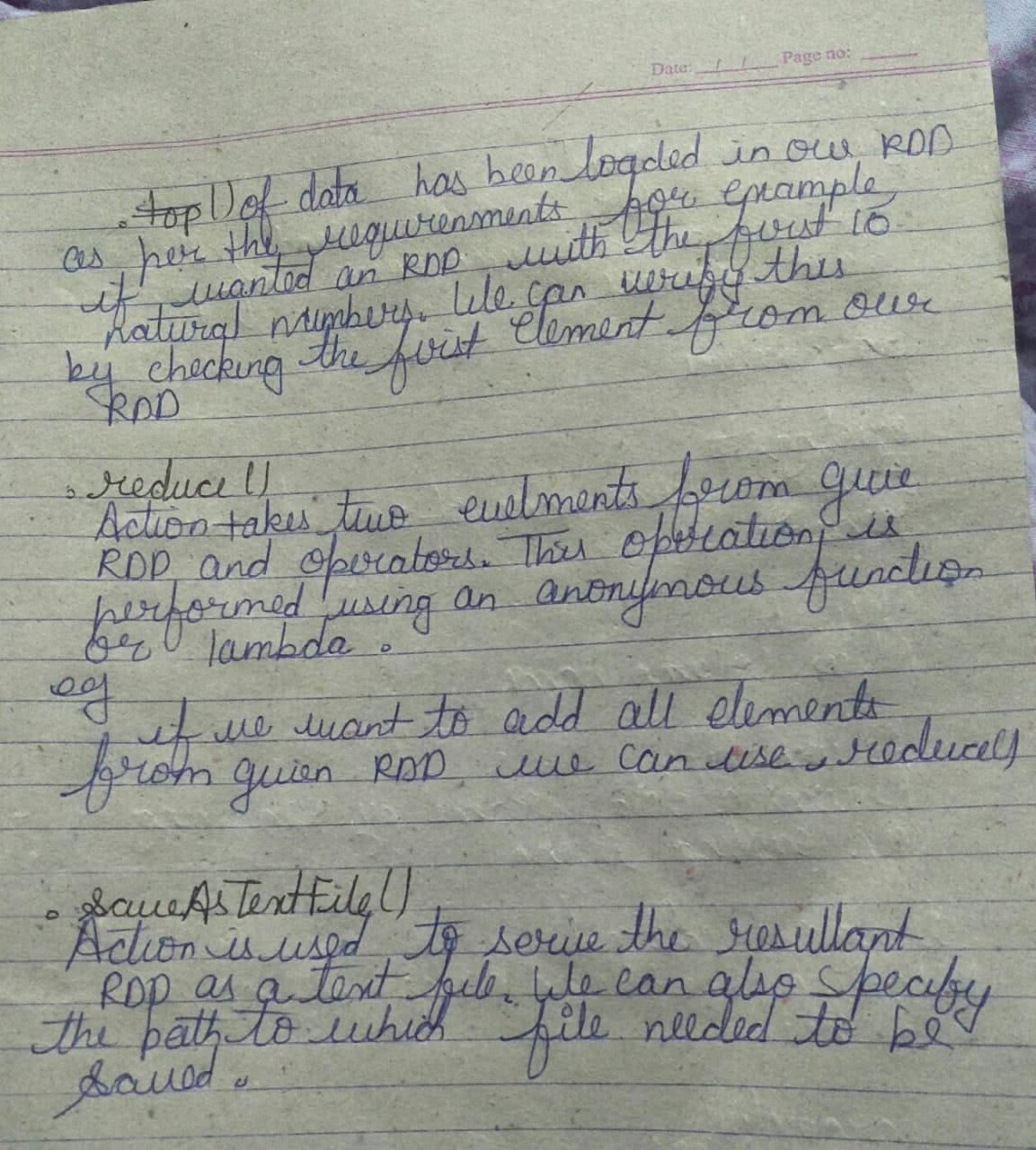
**SAVE AS TEXT FILE**

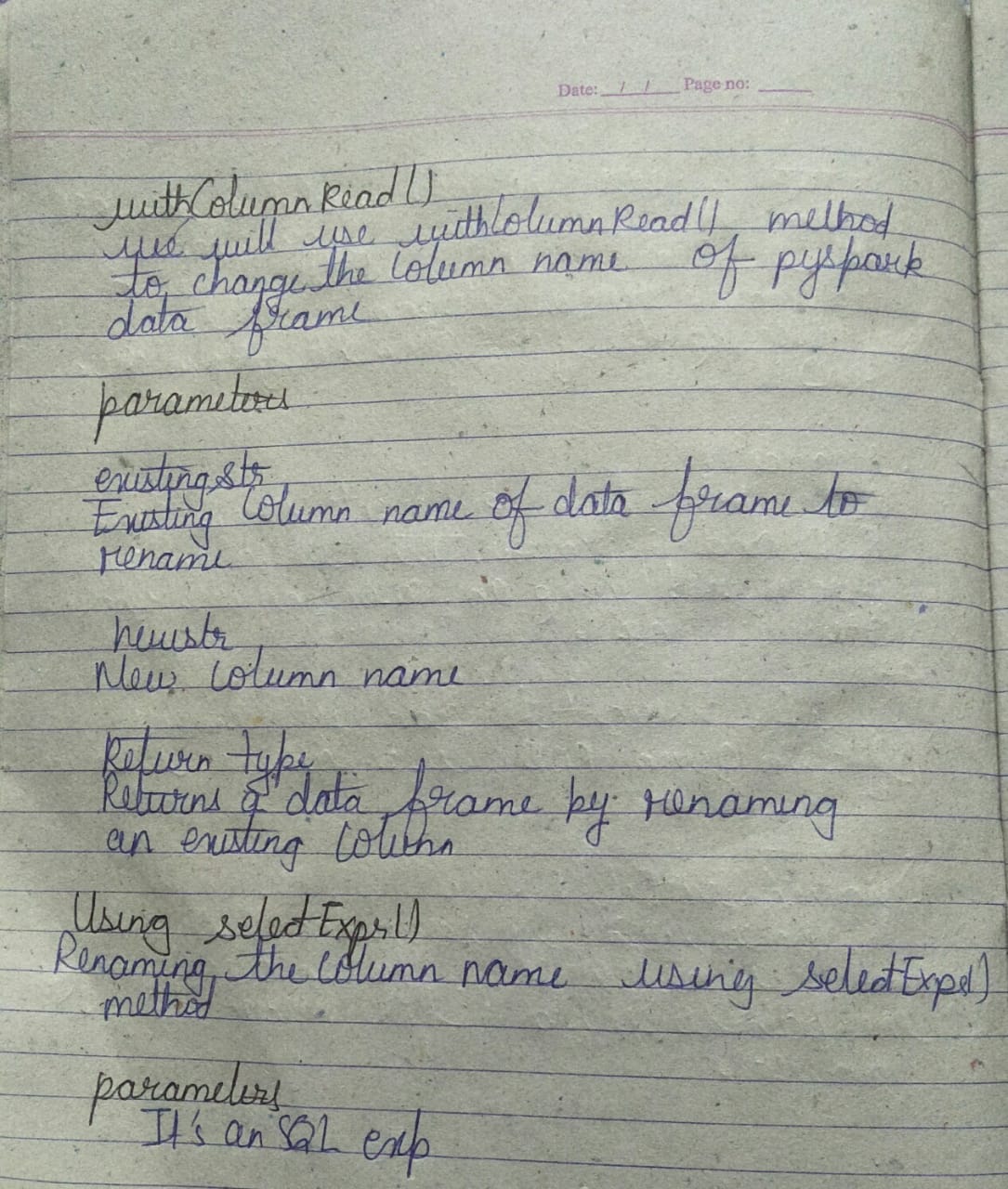
****

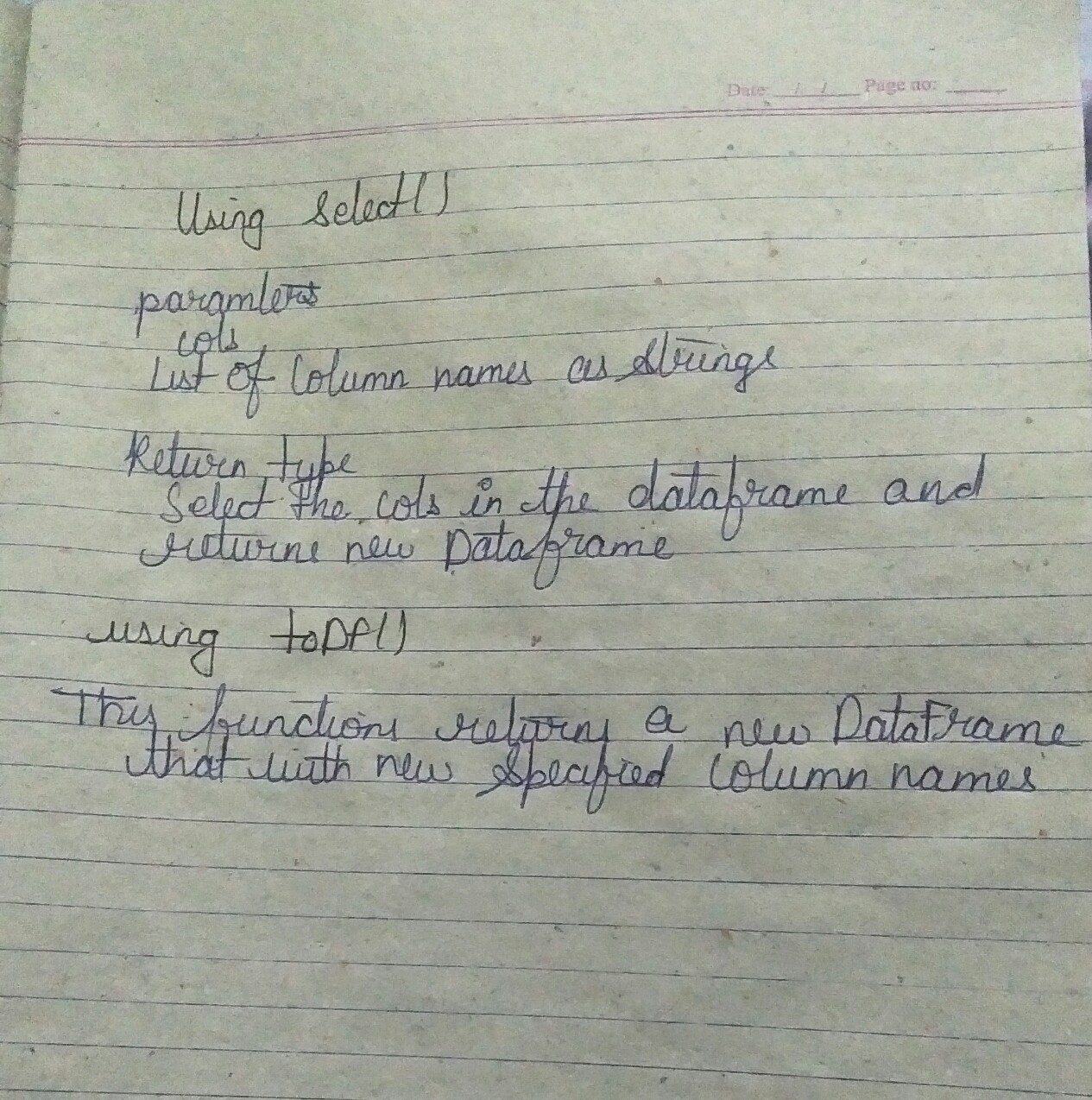
**NOTES**

****

****

****

****

****