Perquisites

```
!apt-get update -qq
!apt-get install -y openjdk-11-jdk-headless -qq
!pip install -q pyspark
```

```
import os
os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-11-openjdk-amd64"
os.environ["SPARK_HOME"] = "/usr/local/lib/python3.12/dist-packages/pyspark"
```

```
from pyspark.sql import SparkSession

spark = SparkSession.builder.appName("Test").getOrCreate()
print("✅ Spark version:", spark.version)
spark.stop()
```

Intro to Pyspark & Big Data Fundamentals with PySpark

```
"""
    Create a SparkSession and use it to access basic SparkContext properties.

    Tasks for the student:
      1. Create a SparkSession with an application name of your choice.
      2. Retrieve the SparkContext from the session.
      3. Print the following SparkContext attributes:
            - Spark version
            - Python version used by Spark
            - Master URL
      4. (Optional) Stop the Spark session after execution.

    No code is provided — implement all steps inside the section below.
"""

# start code here









# end of code
```

```
"""
    Create a Python list of numbers and load it into a PySpark RDD.

    Tasks for the student:
      1. Create a Python list or range containing the numbers 1 through 100.
      2. Use sc.parallelize() to convert this list into an RDD.
      3. Store the resulting RDD in a variable.

    Implement all steps below.
"""

# start code here









# end of code
```

```
"""
    Load a text file Complete_Shakespeare.txt into a PySpark RDD.

    Tasks for the student:
```

```
      1. Define a file path pointing to a local text file in Colab.
      2. Use sc.textFile() to load the file into an RDD.
      3. Store the RDD in a variable for later use.

    Implement all steps below.
"""

# start code here




# end of code
```

```
"""
    Work with a Python list and apply a transformation using map().

    Tasks for the student:
      1. Define a Python list containing several integers.
      2. Print the list.
      3. Use map() + a lambda function to square each element.
      4. Convert the map result into a list.
      5. Print the squared result.

    Implement all steps below.
"""

# start code here




# end of code
```

```
"""
    Load a CSV file loan.csv into a Spark DataFrame and create a transformed column.

    Tasks for the student:
      1. Read a CSV file (loan.csv) using spark.read.csv() with:
            - header = True
            - inferSchema = True
      2. Display the first few rows of the DataFrame.
      3. Use pyspark.sql.functions.col to select a numeric column (e.g., LoanAmount).
      4. Create a new column that contains the squared value of this numeric column.
      5. Display both the original and the new squared column.

    Implement all steps below.
"""

# start code here




# end of code
```

```
"""
    Work with text data using RDD operations.

    Tasks for the student:
      1. Load a CSV file loan.csv into an RDD using sc.textFile().
      2. Print the first few lines using take().
      3. Map each line to its character length.
      4. Compute and print the average line length using mean().

    Implement all steps below.
"""

# start code here
```

```
# end of code
```

```
"""
    Filter elements in a Python list using lambda + filter().

    Tasks for the student:
      1. Create a Python list of integers.
      2. Print the list.
      3. Use filter() with a lambda to keep numbers divisible by 10.
      4. Convert the result into a list.
      5. Print the filtered list.

    Implement all steps below.
"""

# start code here




# end of code
```

```
"""
    Create an RDD from a list of words and inspect its type.

    Tasks for the student:
      1. Create a Python list of words.
      2. Convert the list into an RDD using sc.parallelize().
      3. Print the type of the resulting RDD.

    Implement all steps below.
"""

# start code here




# end of code
```

```
"""
    Load a text file Complete_Shakespeare.txt into an RDD and inspect its type.

    Tasks for the student:
      1. Define a file path pointing to a text file in Colab.
      2. Print the file path.
      3. Load the file using sc.textFile().
      4. Print the type of the created RDD.

    Implement all steps below.
"""

# start code here




# end of code
```

```
"""
    Inspect and control the number of RDD partitions.

    Tasks for the student:
      1. Use getNumPartitions() to print the number of partitions
         in an RDD created from a text file.
      2. Reload the same file using sc.textFile() with minPartitions set
```

```
        to a custom number (e.g., 5).
      3. Print the number of partitions in the new RDD.

    Implement all steps below.
"""

# start code here




# end of code
```

```
"""
    Create an RDD and apply a map() transformation.

    Tasks for the student:
      1. Create or reuse an active SparkSession and SparkContext.
      2. Define a Python list of numbers.
      3. Convert the list into an RDD using sc.parallelize().
      4. Apply a map() transformation to cube each number.
      5. Collect the transformed results back to the driver.
      6. Print both the original list and the cubed numbers.

    Implement all steps below.
"""

# start code here




# end of code
```

```
"""
    Filter an RDD based on a keyword and inspect results.

    Tasks for the student:
      1. Use the existing fileRDD created earlier.
      2. Apply a filter() transformation to keep only lines containing a chosen keyword.
      3. Count how many lines match the filter condition.
      4. Print the first few (e.g., 4) matching lines.

    Implement all steps below.
"""

# start code here




# end of code
```

```
"""
    Create a Pair RDD and use reduceByKey().

    Tasks for the student:
      1. Create an RDD containing (key, value) tuples.
      2. Apply reduceByKey() to aggregate values for each key.
      3. Collect the result.
      4. Loop through the output and print key–value results.

    Implement all steps below.
"""

# start code here
```

```
# end of code
```

```
"""
    Sort a Pair RDD by key in descending order.

    Tasks for the student:
      1. Take the previously reduced RDD.
      2. Apply sortByKey() with descending order.
      3. Collect the sorted result.
      4. Print each key and its corresponding value.

    Implement all steps below.
"""

# start code here




# end of code
```

```
"""
    Count the occurrences of keys in a Pair RDD.

    Tasks for the student:
      1. Use countByKey() on an existing Pair RDD.
      2. Print the type of the returned object.
      3. Iterate through the dictionary and print each key with its count.

    Implement all steps below.
"""

# start code here




# end of code
```

```
"""
    Count total words using flatMap() on a text file RDD.

    Tasks for the student:
      1. Load a text file Complete_Shakespeare.txt into an RDD.
      2. Use flatMap() to split lines into individual words.
      3. Count the total number of words.
      4. Print the count.

    Implement all steps below.
"""
# start code here




# end of code
```

```
"""
    Remove stop words and count word frequencies.

    Tasks for the student:
      1. Use an existing list named stop_words.
      2. Filter the RDD to remove any word appearing in the stop_words list.
      3. Map remaining words to (word, 1) tuples.
      4. Aggregate counts using reduceByKey().

    Implement all steps below.
"""
```

```
# start code here




# end of code
```

```
"""
    Display most frequent words after filtering stop words.

    Tasks for the student:
      1. Display the first 10 elements of the word-count RDD.
      2. Swap (word, frequency) → (frequency, word) using map().
      3. Sort by key (frequency) in descending order.
      4. Display the top 10 most frequent words.

    Implement all steps below.
"""

# start code here







# end of code
```

```
"""
    Convert an RDD of tuples into a PySpark DataFrame.

    Tasks for the student:
      1. Create a Python list of (Name, Age) tuples.
      2. Convert it into an RDD using sc.parallelize().
      3. Create a DataFrame from the RDD using spark.createDataFrame().
      4. Print the type of the DataFrame.
      5. Show the DataFrame content.

    Implement all steps below.
"""

# start code here







# end of code
```

```
"""
    Load a CSV file into a PySpark DataFrame.

    Tasks for the student:
      1. Define a file path to a CSV file people.csv.
      2. Load the file into a DataFrame using spark.read.csv() with:
            - header = True
            - inferSchema = True
      3. Print the type of the resulting DataFrame.

    Implement all steps below.
"""

# start code here







# end of code
```

```
"""
    Task:
        - Display the first 10 rows of the DataFrame `people_df`.
        - Count and print the total number of rows in the DataFrame.
        - Count and print the number of columns, including their names.

    Hint:
        Use appropriate DataFrame actions such as `show()`, `count()`, and attributes like `.columns`.

    Returns:
        None
"""

# start code here



# end of code
```

```
"""
    Task:
        - Select the columns 'name', 'sex', and 'date of birth' from `people_df`
          and store them into `people_df_sub`.
        - Display the first 10 rows of this new DataFrame.
        - Remove duplicate rows and store the result in `people_df_sub_nodup`.
        - Print the number of rows before and after removing duplicates.

    Args:
        people_df : existing Spark DataFrame

    Returns:
        None
"""

# start code here



# end of code
```

```
"""
    Task:
        - Filter the DataFrame `people_df` to create:
            1. `people_df_female` containing only rows where sex == "female".
            2. `people_df_male`   containing only rows where sex == "male".
        - Count and print the number of rows in each resulting DataFrame.

    Args:
        people_df : existing Spark DataFrame

    Returns:
        None
"""

# start code here



# end of code
```

```
"""
    Task:
        - Register `people_df` as a temporary SQL table named "people".
        - Write an SQL query that selects only the 'name' column from this table.
        - Run the query and store the result into `people_df_names`.
        - Display the first 10 rows of the result.

    Returns:
        None
"""

# start code here



# end of code
```

```
"""
    Task:
        - Using Spark SQL:
            * Create a DataFrame `people_female_df` containing rows where sex == "female".
            * Create a DataFrame `people_male_df` containing rows where sex == "male".
        - Print the number of rows in each DataFrame.

    Returns:
        None
"""

# start code here



# end of code
```

```
"""
    Task:
        - Print the column names of the DataFrame `names_df`.
        - Convert `names_df` into a Pandas DataFrame.
        - Using Pandas' plotting capabilities:
            * Create a horizontal bar chart where:
                x-axis = 'Name'
                y-axis = 'Age'
        - Display the plot.

    Note:
        Matplotlib and seaborn are already imported.

    Returns:
        None
"""

# start code here



# end of code
```

```
"""
    Task:
        - Load the CSV file located at '/content/Fifa2018_dataset.csv'
          into a Spark DataFrame named `fifa_df`.
        - Print the schema of the DataFrame.
        - Display the first 10 rows.
        - Print the total number of rows in the DataFrame.

    Returns:
        None
"""

# start code here


# end of code
```

```
"""
    Task:
        - Register `fifa_df` as a temporary SQL view named 'fifa_df_table'.
        - Write an SQL query that selects the 'Age' column only for players
          whose Nationality is "Germany".
        - Store the result into `fifa_df_germany_age`.
        - Generate and display the summary statistics (describe) for the result.

    Returns:
        None
"""

# start code here



# end of code
```

```
"""
    Task:
        - Convert `fifa_df_germany_age` into a Pandas DataFrame.
        - Plot the density curve (distribution) of the 'Age' column.
        - Display the plot.

    Returns:
        None
"""

# start code here



# end of code
```

ML with PySpark

```
"""
    Task:
        - Import the following MLlib modules:
            * ALS and Rating   (from pyspark.mllib.recommendation)
            * LogisticRegressionWithLBFGS (from pyspark.mllib.classification)
            * KMeans           (from pyspark.mllib.clustering)

    Note:
        No implementation is required beyond importing the libraries.

    Returns:
        None
"""

# start code here



# end of code
```

```
"""
    Task:
        - Load the file '/content/ratings.csv' into an RDD.
        - Split each line by comma.
        - Convert each line into a Rating(user, item, rating) object.
        - Split the final RDD into training and test sets with an 80/20 ratio.

    Returns:
        training_data : RDD
        test_data     : RDD
"""

# start code here


# end of code
```

```
"""
    Task:
        - Train an ALS model using the training dataset.
        - Remove the rating values from the test dataset (keep only user, item pairs).
        - Use the trained model to generate predictions.
        - Display the first 2 predicted rows.

    Returns:
        None
"""

# start code here



# end of code
```

```
"""
    Task:
        - Prepare `ratings_final` as key-value pairs: ((user, item), rating).
        - Prepare predictions in the same key-value format.
        - Join actual ratings with predicted ratings.
        - Calculate the Mean Squared Error (MSE).
        - Print the MSE value.

    Returns:
        None
"""

# start code here



# end of code
```

```
"""
    Task:
        - Load 'spam.txt' and 'ham.txt' into separate RDDs.
        - Split each email into individual words using flatMap().
        - Print the first word from both resulting RDDs.

    Returns:
        None
"""

# start code here



# end of code
```

```
"""
    Task:
        - Use HashingTF with 200 features to map words to feature vectors.
        - Transform spam and non-spam words into hashed features.
        - Label spam as 1 and non-spam as 0 using LabeledPoint.
        - Combine (union) the two labeled datasets into one RDD named `samples`.

    Returns:
        samples : RDD of LabeledPoint
"""

# start code here



# end of code
```

```
"""
    Task:
        - Split the combined dataset (samples) into training and testing sets.
        - Train a Logistic Regression model using the training data.
        - Predict labels for the test dataset.
        - Pair true labels with predicted labels.
        - Compute and print the classification accuracy.

    Returns:
        None
"""

# start code here



# end of code
```

```
"""
    Task:
        - Load the file '5000_points.txt' into an RDD.
        - Split each line by tab ("\t").
        - Convert resulting values into integers inside a list [x, y].
        - Count and print the number of rows in the transformed RDD.
```

```
    Returns:
        None
"""

# start code here



# end of code
```

```
"""
    Task:
        - Define an error function computing squared distance from each point to its cluster center.
        - Train KMeans models for cluster counts 13 to 16.
        - For each model, compute and print WSSSE (Within Set Sum of Squared Error).
        - Train a final model using the best k (16).
        - Extract and store the resulting cluster centers.

    Returns:
        cluster_centers : list of cluster center coordinates
"""

# start code here



# end of code
```

```
"""
    Task:
        - Convert the RDD (rdd_split_int) into a Spark DataFrame, then to a Pandas DataFrame.
        - Convert KMeans cluster centers into a Pandas DataFrame.
        - Create a scatter plot of all points.
        - Overlay cluster centers on the scatter plot using a distinct marker.

    Returns:
        None
"""

# start code here



# end of code
```