```
Let's use again the Docker official image.
We assume files to be used are stored in the attached volume.
        docker run -it -v $PWD/data:/data -p 8080:8080 -p:4040:4040
--rm spark:python3 /opt/spark/bin/pyspark

# Creating new DataFrame objects from text, csv, JSON, and other
files
# can be done easily with spark.read.
# If the DataFrame schema is specified on the first line of the
file, use
# spark.read.option("header", True).
# Additionally, using spark.createDataFrame() you can create
DataFrames from
# existing Pandas DataFrames, RDDs, numpy arrays, and lists.

# Create a DataFrame from CSV file
df = spark.read.csv("/data/address.csv", header=True)
# Display the top rows of a DataFrame
df.show()
# Display the schema of a DataFrame
df.printSchema()

# From JSON to Parquet
peopleDF = spark.read.json("/data/people.json")
peopleDF.write.parquet("/data/people.parquet")
parquetFile = spark.read.parquet("/data/people.parquet")

# DataFrame and Spark SQL share the same execution engine so they
# can be interchangeably used seamlessly.
# Let's register the DataFrame as a table and run a SQL query
parquetFile.createOrReplaceTempView("parquetFile")
teenagers = spark.sql("SELECT name FROM parquetFile WHERE age >= 13
AND age <= 19")
teenagers.show()

# Load a text file as a DataFrame, convert it to an RDD, count the
# number of occurrences of each word, and sort the words by count
# in descending order.
# Return a list of tuples containing the first five (word, count)
pairs.
data = spark.read.text("data/input.txt").rdd.map(lambda r: r[0])
words = data.flatMap(lambda line: line.split(' ')) \
        .map(lambda word: (word, 1)) \
        .reduceByKey(lambda a, b: a + b)
#sort descending
wordsSorted = words.sortBy(lambda a: -a[1])
wordsSorted.take(5)

# Compute mean value using DataFrames
from pyspark.sql.functions import avg
data_df = spark.createDataFrame([("Brooke", 20), ("Denny", 31),
        ("Jules", 30), ("TD", 35), ("Brooke", 25)],
        ["name", "age"])
# Group the same names together, aggregate their ages,
```

```
# and compute an average
avg_df = data_df.groupBy("name").agg(avg("age"))
# Show the result
avg_df.show()
# Visualize the DAG from UI

# Caching a DataFrame
# Create a DataFrame with 1M records
df = spark.range(1 * 1000000).toDF("id")
df.withColumn('square', df.id * df.id)
# Cache it
df.cache()
# Materialize the cache
df.count()
# Now get the DataFrame from the cache
df.count()
# Visualize the DAG from UI
```