# Spark & Hive & Hadoop – by Ricardo

## Install WSL (Windows Subsystem for Linux)

1. Install Ubuntu via WSL:

power shell

wsl --install -d Ubuntu-24.04

1. Set WSL version 2 as the default version for any new Linux distributions:

power shell

wsl --set-version Ubuntu-24.04 2

1. Selecting the distribution

power shell

wsl -d Ubuntu-24.04 2

1. Uninstall WSL Wwhen needed

power shell

wsl --unregister Ubuntu-24.04

## From now on into the Ubuntu - Update System Packages

bash

sudo apt update && sudo apt upgrade -y

sudo apt install build-essential curl git -y

sudo apt install python3 python3-pip -y

## Create Python Virtual Environment

bash

mkdir ~/pyspark-project

cd ~/pyspark-project

sudo apt install python3.12-venv -y

python3 -m venv spark\_env

source spark\_env/bin/activate

## Install PySpark & Java 11 & Other Python Libraries

With the virtual environment activated

bash

(spark\_env) ricardo@Mini-Ryzen:~/apache-hive$  
pip install pyspark pandas numpy jupyterlab tqdm && sudo apt install openjdk-11-jdk -y

## Download and install Apache Hive & Hadoop

bash

mkdir ~/apache-hive && cd ~/apache-hivewget https://archive.apache.org/dist/hive/hive-2.3.9/apache-hive-2.3.9-bin.tar.gz

tar xzf apache-hive-2.3.9-bin.tar.gz

mkdir ~/hadoop && cd ~/hadoop   
wget <https://archive.apache.org/dist/hadoop/common/hadoop-2.7.7/hadoop-2.7.7.tar.gz>

tar xzf hadoop-2.7.7.tar.gz

## Set up the environment variables by adding to your ~/.bashrc

bash

echo 'export HIVE\_HOME=~/apache-hive/apache-hive-2.3.9-bin' >> ~/.bashrc

echo 'export PATH=$PATH:$HIVE\_HOME/bin' >> ~/.bashrc

echo 'export HADOOP\_HOME=~/hadoop/hadoop-2.7.7' >> ~/.bashrc

echo 'export PATH=$PATH:$HADOOP\_HOME/bin' >> ~/.bashrc

echo 'export JAVA\_HOME=/usr/lib/jvm/java-11-openjdk-amd64' >> ~/.bashrc

source ~/.bashrc

## Verify Hive installation

bash

hive --version

## Initialize the Hive schema by running

bash

$HIVE\_HOME/bin/schematool -initSchema -dbType derby

## Remove SLF4J Library Conflict

bash

rm ~/hadoop/hadoop-2.7.7/share/hadoop/common/lib/slf4j-log4j12-1.7.10.jar

## From the inside of our Ubuntu let’s call VsCode

bash

cd ~/pyspark-project && source spark\_env/bin/activate

code .

## Let’s create a new file of type Jupyter Notebook

A screenshot of a computer

Description automatically generated

Inside of each cell there are more text than displayed, please control + a and copy and paste into a Notebook cell.



Notebook Cell 4

cleanup\_previous\_session()

Notebook Cell 3

# Verify stocks\_info table

print("\nVerifying table creation:")

tables\_df = spark.sql("SHOW TABLES IN new\_stock\_db")

tables\_list = [row.tableName for row in tables\_df.collect()]

# Loop through each table and show its description

for table in tables\_list:

    print(f"\nDescription for table: {table}")

    print("-" \* 50)

    spark.sql(f"SHOW TBLPROPERTIES new\_stock\_db.{table}").show()

    spark.sql(f"DESCRIBE TABLE new\_stock\_db.{table}").show()

    spark.sql(f"SELECT \* FROM new\_stock\_db.{table} LIMIT 5").show()

Notebook Cell 2

from pyspark.sql.functions import lit

from datetime import datetime

import pandas as pd

# Configure logging to show only errors

import logging

logging.getLogger("py4j").setLevel(logging.ERROR)

logging.getLogger("org.apache.spark").setLevel(logging.ERROR)

logging.getLogger("org.apache.hadoop").setLevel(logging.ERROR)

logging.getLogger("org.apache.hive").setLevel(logging.ERROR)

# Create database if it doesn't exist

NEW\_DB\_NAME = "new\_stock\_db"

spark.sql(f"CREATE DATABASE IF NOT EXISTS {NEW\_DB\_NAME}")

def log\_memory\_usage(phase=""):

    """Log current memory usage"""

    process = psutil.Process(os.getpid())

    memory\_info = process.memory\_info()

    print(f"\nMemory Usage {phase}:")

    print(f"RSS: {memory\_info.rss / 1024 / 1024:.2f} MB")

    print(f"VMS: {memory\_info.vms / 1024 / 1024:.2f} MB")

from pyspark.sql.types import (

    StructType,

    StructField,

    StringType,

    DoubleType,

    DateType,

    LongType

)

# Define schema for sector data

schema = StructType([

    StructField("Date", StringType(), True),

    StructField("Ticker", StringType(), True),

    StructField("Open", DoubleType(), True),

    StructField("High", DoubleType(), True),

    StructField("Low", DoubleType(), True),

    StructField("Close", DoubleType(), True),

    StructField("Volume", LongType(), True),

    StructField("Dividends", DoubleType(), True),

    StructField("Stock Splits", DoubleType(), True),

    StructField("Daily Return", DoubleType(), True),

    StructField("Cumulative Return", DoubleType(), True),

    StructField("SMA\_20", DoubleType(), True),

    StructField("EMA\_20", DoubleType(), True)

])

# File list

current\_dir = os.getcwd()

# Set up warehouse directory

warehouse\_dir = spark.conf.get("spark.sql.warehouse.dir")

NEW\_TABLE\_NAME = "new\_sectors\_historical"

table\_path = os.path.join(warehouse\_dir, f"{NEW\_DB\_NAME}.db", NEW\_TABLE\_NAME)

table\_path = os.path.abspath(os.path.expanduser(table\_path))

# Set up warehouse directory

warehouse\_dir = spark.conf.get("spark.sql.warehouse.dir")

table\_path = os.path.join(warehouse\_dir, f"{NEW\_DB\_NAME}.db", NEW\_TABLE\_NAME)

table\_path = os.path.abspath(os.path.expanduser(table\_path))

# Process files

all\_dfs = []

chunk\_size = 10000

log\_memory\_usage("Before processing files")

# Create stocks\_info table

print("\nCreating stocks\_info table...")

try:

    df\_pandas = pd.read\_csv(

        "https://raw.githubusercontent.com/ricardokazuo/notebooks/refs/heads/main/Stocks\_Info.csv"

    )

    # Clean column names

    df\_pandas.columns = [

        col.strip().replace(" ", "\_").replace(".", "\_")

            .replace("(", "").replace(")", "")

            .replace("-", "\_")

        for col in df\_pandas.columns

    ]

    # Convert to Spark DataFrame and save as table

    df\_spark\_info = spark.createDataFrame(df\_pandas)

    df\_spark\_info.write.mode("overwrite").saveAsTable(f"{NEW\_DB\_NAME}.stocks\_info")

    # Clean up

    del df\_pandas

    df\_spark\_info.unpersist()

    clean\_memory()

except Exception as e:

    print(f"Error creating stocks\_info table: {str(e)}")

Notebook Cell 1

import os

import gc

import glob

import psutil

from pyspark.sql import SparkSession

NEW\_WAREHOUSE\_PATH = os.path.expanduser("~/new\_spark\_warehouse")

def cleanup\_previous\_session():

    """Cleanup any existing Spark session and memory"""

    if 'spark' in globals():

        try:

            # Clear cache and show memory usage before stopping

            spark.catalog.clearCache()

            # Check and display memory usage

            process = psutil.Process(os.getpid())

            memory\_info = process.memory\_info()

            print("\nMemory Usage before stopping previous session:")

            print(f"RSS: {memory\_info.rss / 1024 / 1024:.2f} MB")

            print(f"VMS: {memory\_info.vms / 1024 / 1024:.2f} MB")

            # Stop the session

            spark.stop()

            print("Previous Spark session stopped successfully")

        except Exception as e:

            print(f"Error during previous session cleanup: {e}")

def clean\_lock\_files():

    """Clean up Derby lock files and Spark metastore locks"""

    # Clean Derby lock files

    derby\_locks = glob.glob("metastore\_db/\*.lck")

    service\_locks = glob.glob("metastore\_db/db.lck")

    temp\_locks = glob.glob("metastore\_db/dbex.\*")

    for lock\_file in derby\_locks + service\_locks + temp\_locks:

        try:

            os.remove(lock\_file)

            print(f"Removed lock file: {lock\_file}")

        except Exception as e:

            print(f"Error removing {lock\_file}: {e}")

    # Clean Spark warehouse locks if they exist

    warehouse\_locks = glob.glob(f"{NEW\_WAREHOUSE\_PATH}/\*\*/\*.lock", recursive=True)

    for lock\_file in warehouse\_locks:

        try:

            os.remove(lock\_file)

            print(f"Removed warehouse lock: {lock\_file}")

        except Exception as e:

            print(f"Error removing warehouse lock {lock\_file}: {e}")

def clean\_memory():

    """Force garbage collection and clear memory"""

    gc.collect()

    torch\_clean = "torch" in globals()

    if torch\_clean:

        import torch

        if torch.cuda.is\_available():

            torch.cuda.empty\_cache()

# Perform complete cleanup of previous session

cleanup\_previous\_session()

clean\_lock\_files()

clean\_memory()

# Set network configurations for Spark

os.environ['SPARK\_LOCAL\_IP'] = 'localhost'

os.environ['SPARK\_DRIVER\_HOST'] = 'localhost'

spark = (

    SparkSession.builder

        .appName("NewSectorDataCombine")

        .config("spark.driver.bindAddress", "localhost")

        .config("spark.driver.host", "localhost")

        .config("spark.driver.memory", "4g")

        .config("spark.executor.memory", "4g")

        .config("spark.sql.shuffle.partitions", "100")

        .config("spark.driver.maxResultSize", "2g")

        .config("spark.memory.fraction", "0.6")

        .config("spark.memory.storageFraction", "0.2")

        .config("spark.sql.adaptive.enabled", "true")

        .config("spark.sql.adaptive.coalescePartitions.enabled", "true")

        .config("spark.sql.adaptive.skewJoin.enabled", "true")

        .config("spark.dynamicAllocation.enabled", "true")

        .config("spark.sql.inMemoryColumnarStorage.compressed", "true")

        .config("spark.sql.inMemoryColumnarStorage.batchSize", "10000")

        .config("spark.sql.autoBroadcastJoinThreshold", "100m")

        .config("spark.sql.files.maxPartitionBytes", "64m")

        .config("spark.sql.files.maxRecordsPerFile", "5000000")

        .config("spark.memory.offHeap.enabled", "false")

        .config("spark.sql.warehouse.dir", NEW\_WAREHOUSE\_PATH) \

        .config("javax.jdo.option.ConnectionURL", "jdbc:derby:;databaseName=metastore\_db;create=true") \

        .config("javax.jdo.option.ConnectionDriverName", "org.apache.derby.jdbc.EmbeddedDriver") \

        .config("hive.metastore.schema.verification", "false")

        # Hive specific configurations

        .config("spark.sql.hive.metastore.version", "2.3.9")

        .config("spark.sql.hive.metastore.jars", "path")

        .config("spark.sql.hive.metastore.jars.path",

                os.path.expanduser("~/apache-hive/apache-hive-2.3.9-bin/lib/\*"))

        # Additional cleanup configurations

        .config("spark.cleaner.periodicGC.interval", "15min")

        .config("spark.cleaner.referenceTracking.blocking", "true")

        .config("spark.cleaner.referenceTracking.blocking.shuffle", "true")

        .config("spark.cleaner.referenceTracking.cleanCheckpoints", "true")

        .enableHiveSupport()

        .getOrCreate()

)