Spark & Hive & Hadoop – by Ricardo

Install WSL (Windows Subsystem for Linux)

1. Install Ubuntu via WSL:

power shell

```
wsl --install -d Ubuntu-24.04
```

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

power shell

```
wsl --set-version Ubuntu-24.04 2
```

3. Selecting the distribution

power shell

```
wsl -d Ubuntu-24.04 2
```

4. 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

(spark_env) ricardo@Mini-Ryzen:~/apache-hive$
sudo apt install openjdk-11-jdk -y
```

Download and install Apache Hive & Hadoop

bash

```
mkdir ~/apache-hive
cd ~/apache-hive
wget 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
```

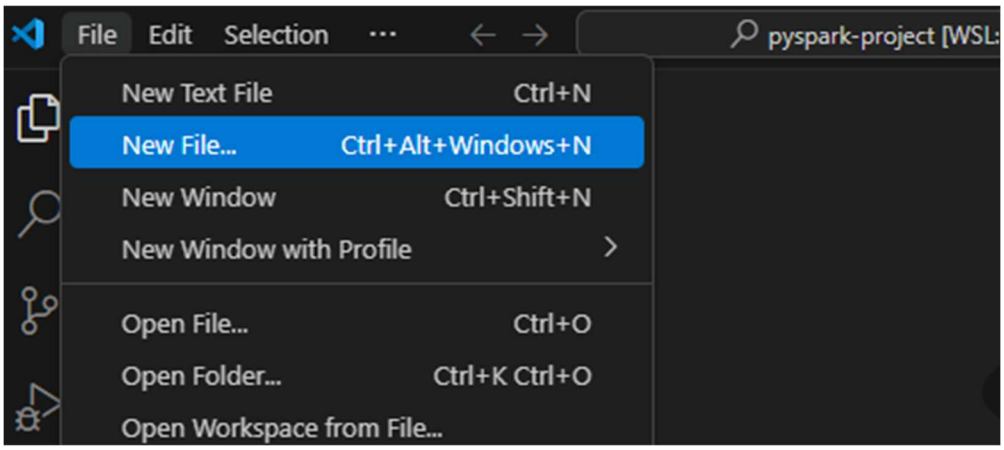
Remove SLF4J Library Conflict

```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



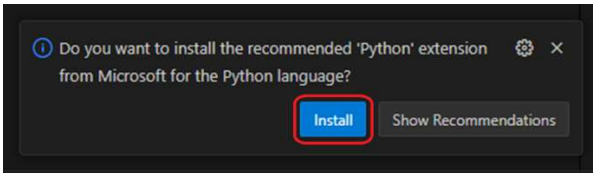
```
import os
import shutil
import pandas as pd
from pyspark.sql import SparkSession

# 1) Stop any existing Spark session just in case
try:
    spark.stop()
except:
    pass

# 2) Remove leftover directories to start from zero
paths_to_remove = [
    "metastore_db",
    "/home/ricardo/pyspark-project/~/Documents/spark-warehouse",
    "/home/ricardo/Documents/spark-warehouse",
    "/home/ricardo/pyspark-project/~/spark-warehouse",
    # If you have partial/corrupt data in the stocks_info location, remove it too:
    "/home/ricardo/Documents/spark-warehouse/stock_db.db/stocks_info"
]

for path in paths_to_remove:
    if os.path.exists(path):
        print(f"Removing {path}")
        shutil.rmtree(path)
    else:
        print(f"Path not found (skipping): {path}")

# 3) Expanded warehouse path (avoid literal "~")
warehouse_path = os.path.expanduser("~/Documents/spark-warehouse")
```

Beware of the current user path, in this case the try part deletes the existing metastore and warehouse to create from zero again.

```
# 4) Create a new Spark session with advanced configs
spark = (
   SparkSession.builder
      .appName("SectorDataCombine")
      .config("spark.driver.memory", "16g")
      .config("spark.executor.memory", "16g")
      .config("spark.sql.shuffle.partitions", "200")
      .config("spark.driver.maxResultSize", "8g")
      .config("spark.memory.fraction", "0.8")
      .config("spark.memory.storageFraction", "0.3")
      .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", "true")
      .config("spark.memory.offHeap.size", "8g")
      .config("spark.sql.warehouse.dir", warehouse_path)
      # Only add these Hive configs if you truly need Hive
      .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/*"))
      .enableHiveSupport()
      .getOrCreate()
)
```

```
# 5) Read CSV via Pandas, then convert to Spark
df_pandas = pd.read_csv(
   "https://raw.githubusercontent.com/ricardokazuo/notebooks/refs/heads/main/Stocks_Info.csv"
)
df_pandas.columns = [
   col.strip().replace(" ", "_").replace(".", "_")
     .replace("(", "").replace(")", "")
     .replace("-", "_")
   for col in df_pandas.columns
]
df_spark = spark.createDataFrame(df_pandas)

# Create a database and ensure table is dropped to avoid conflicts
spark.sql("CREATE DATABASE IF NOT EXISTS stock_db")
spark.sql("DROP TABLE IF EXISTS stock_db.stocks_info")

# 6) Write DataFrame as Hive table
df_spark.write.mode("overwrite").saveAsTable("stock_db.stocks_info")

# 7) Verify the table
print("\nVerifying table creation:")
spark.sql("SELECT * FROM stock_db.stocks_info").show()

# 8) Stop Spark session to prevent re-init conflicts
spark.stop()
```

When prompted, let's install the extension



Let's select