Apache spark - to process a simple Apple stock prices from yahoo finance

Meron and MD

PySpark advantages over Hadoop

- High-level programming interface
- Built on Apache Spark for performance and scalability
- Supports a wide range of data sources and formats

Advantages of PySpark

- High-level, Pythonic API for data processing:
- DataFrame API for powerful and easy-to-use interface
- Built-in functions and transformations for data processing
- Designed for distributed computing and can scale to handle large datasets:
- Includes MLlib for machine learning, GraphX for graph processing, and Spark Streaming for real-time data processing

Dataset Overview

- The dataset used in this project is the daily historical stock price data for Apple Inc. (AAPL) from Yahoo Finance.
- The dataset contains daily stock prices for AAPL from 1980 to the present.
- The dataset is provided in CSV format and has a file size of approximately 250 MB.
- The dataset is used in this project to demonstrate how to fetch and process data using PySpark and Python, and to perform basic data analysis and visualization on the results.

Code overview

```
In [29]: import yfinance as yf
```

```
In [5]: import pyspark
```

```
In [6]: from pyspark.sql import SparkSession
```

```
In [7]: spark=SparkSession.builder.appName("Gold").getOrCreate()
```

Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).

23/03/03 21:01:34 WARN NativeCodeLoader: Unable to load native-hadoop library for your platfo
rm... using builtin-java classes where applicable

```
In [26]: spark
```

Out[26]: **SparkSession - in-memory**

SparkContext

[Spark UI](#)

Version

`v3.3.2`

Master

`local[*]`

AppName

`Gold`

```
In [27]:  from pyspark.sql.functions import col, stddev, mean
          from pyspark.sql.window import Window
```

```
In [57]:  # Fetch data from yfinance
          data = yf.download("AAPL", start="1980-01-01", end="2022-03-01", interval="1d")


          [***********************100%***********************]  1 of 1 completed
```

```
In [51]:  # Convert the pandas DataFrame to a PySpark DataFrame
          df = spark.createDataFrame(data.reset_index()).select(['Date','Open','Close'])
          df.count()
```

Out[51]: 5575

```
In [54]:  # Calculate the rolling mean and standard deviation of the closing price
          window = Window.orderBy(col("Date")).rowsBetween(-29, 0)
          df = df.withColumn("rolling_mean", mean(col("Close")).over(window)) \
                 .withColumn("rolling_stddev", stddev(col("Close")).over(window)).na.drop()
```

```python
In [27]:  from pyspark.sql.functions import col, stddev, mean
          from pyspark.sql.window import Window
```

```python
In [57]:  # Fetch data from yfinance
          data = yf.download("AAPL", start="1980-01-01", end="2022-03-01", interval="1d")

          [***********************100%***********************]  1 of 1 completed
```

```python
In [51]:  # Convert the pandas DataFrame to a PySpark DataFrame
          df = spark.createDataFrame(data.reset_index()).select(['Date','Open','Close'])
          df.count()
```

```
Out[51]:  5575
```

```python
In [54]:  # Calculate the rolling mean and standard deviation of the closing price
          window = Window.orderBy(col("Date")).rowsBetween(-29, 0)
          df = df.withColumn("rolling_mean", mean(col("Close")).over(window)) \
                  .withColumn("rolling_stddev", stddev(col("Close")).over(window)).na.drop()
```