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
```

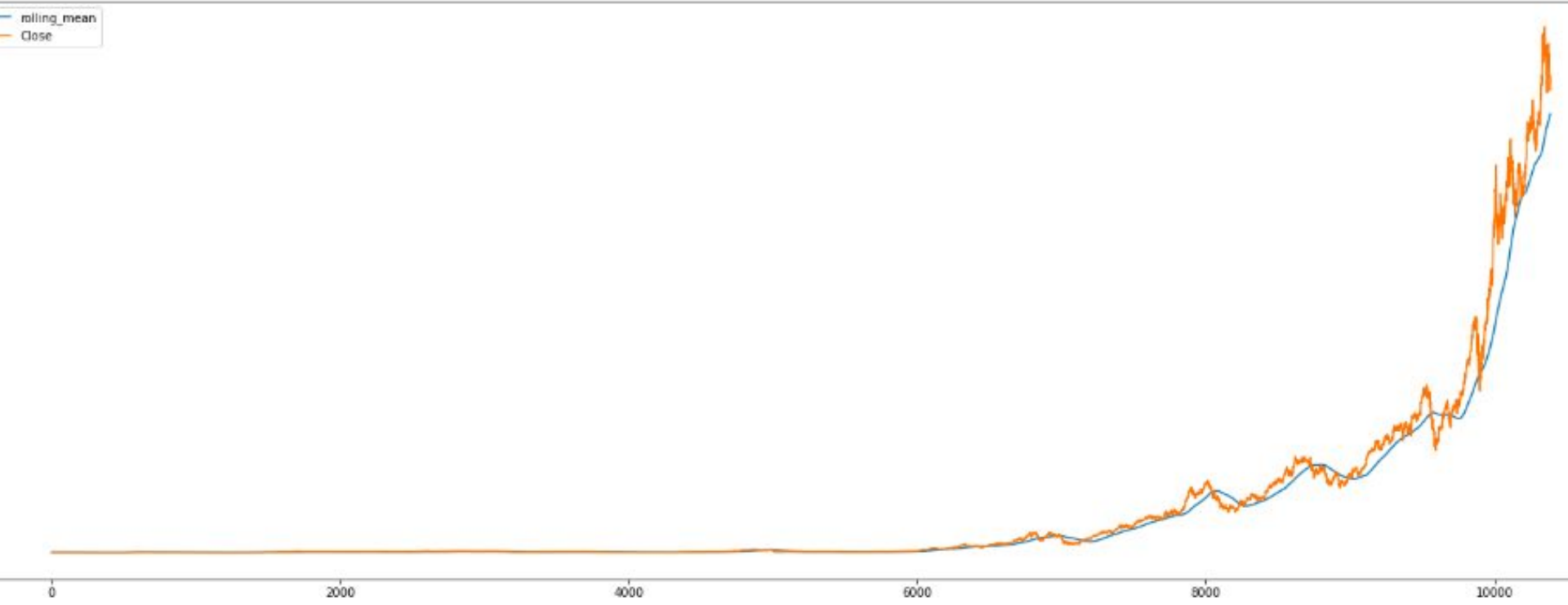
In [54]:
```
# Calculate the rolling mean and standard deviation of the closing price
window = Window.orderBy(col("Date")).rowsBetween(-29, 0)
df = df.withColumn("rolling_mean", mean(col("Close")).over(window)) \
       .withColumn("rolling_stddev", stddev(col("Close")).over(window)).na.drop()
```

```
+-------------------+-----------------+-----------------+-----------------+-------------
----+
|               Date|             Open|            Close|     rolling_mean|     rolling_st
ddev|
+-------------------+-----------------+-----------------+-----------------+-------------
----+
|2000-01-04 00:00:00| 0.966517984867096|0.9151790142059326|0.9573104977607727|0.0595829154461
5387|
|2000-01-05 00:00:00|0.9263389706611633|0.9285709857940674|0.9477306604385376|0.0452811412830
2451|
|2000-01-06 00:00:00|0.9475449919700623|0.8482139706611633| 0.922851487994194|0.0619904352029
9742|
|2000-01-07 00:00:00| 0.861607015132904| 0.888392984867096|0.9159597873687744|0.0558532741210
8275|
+-------------------+-----------------+-----------------+-----------------+-------------
----+
only showing top 4 rows
```

Simple Visualization

```
.rolling_mean.plot(figsize=(25,9))
['Close'].plot()
f.Open.plot()
.legend(['rolling_mean','Close'])
```

```
<matplotlib.legend.Legend at 0x11ff77a60>
```



In [25]:
```
# Visualizing the rolling_mean
pdf.rolling_stddev.plot(figsize=(25,9))
pdf['Close'].plot()
#pdf.Open.plot()
plt.legend(['rolling_stddev','Close'])
```

Out[25]: <matplotlib.legend.Legend at 0x12005d280>