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 [1]:
```
import yfinance as yf
```

PySpark and Initializing Spark Session

In [2]:
```
import pyspark
```

In [3]:
```
from pyspark.sql import SparkSession
```

In [4]:
```
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).

23/03/06 12:07:30 WARN NativeCodeLoader: Unable to load native-hadoop library for your platfo
rm... using builtin-java classes where applicable
```

In [5]:
```
spark
```

Out[5]:

SparkSession - in-memory

SparkContext

Spark UI

Version

`v3.3.2`

Master

`local[*]`

AppName

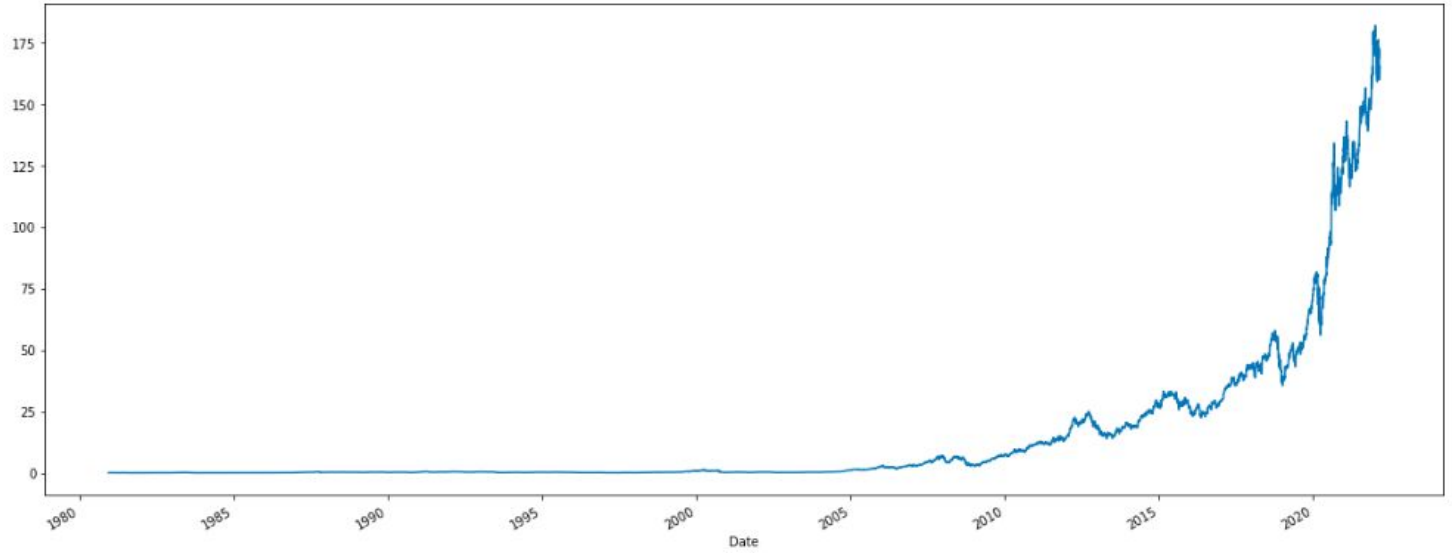
`Gold`

Fetching dataset

In [7]:
```
# Fetch data from yfinance
data = yf.download("AAPL", start="1980-01-01", end="2022-03-01", interval="1d")
data['Close'].plot(figsize = (20,8))
```

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

Out[7]: <AxesSubplot:xlabel='Date'>



Converting dataset to PySpark dataframe

In [23]:
```
# Convert the pandas DataFrame to a PySpark DataFrame
df = spark.createDataFrame(data.reset_index()).select(['Date','Open','Close'])
df.count()
```

Out[23]: 10391

Smart anaysis

In [21]:
```
# Calculate the rolling mean and standard deviation of the closing price
window = Window.orderBy(col("Date")).rowsBetween(-200, 0)
df = df.withColumn("rolling_mean", mean(col("Close")).over(window)) \
       .withColumn("rolling_stddev", stddev(col("Close")).over(window)).na.drop()
```

In [22]:
```
df.show(4)
```

```
+-------------------+-------------------+-------------------+-------------------+--------------------
--------+
|               Date|               Open|              Close|       rolling_mean|      rollin
g_stddev|
+-------------------+-------------------+-------------------+-------------------+--------------------
--------+
|1980-12-15 00:00:00|0.12221000343561172|0.12165199965238571| 0.1249999962747097|  0.00473478
22300699|
|1980-12-16 00:00:00| 0.1132809966802597|0.11272300034761429|0.12090766429901123|  0.00783904
48640804|
|1980-12-17 00:00:00|0.11551299691200256|0.11551299691200256|0.11955899745225906| 0.006945695
93164169|
|1980-12-18 00:00:00|0.11886200308799744|0.11886200308799744|0.11941959857940673|0.0060232200
03443378|
+-------------------+-------------------+-------------------+-------------------+--------------------
--------+
only showing top 4 rows
```

Simple Visualization

Converting to pandas for visualization

In [24]:
```
import matplotlib.pyplot as plt
```

In [25]:
```
pdf = df.toPandas()
```

In [26]:
```
pdf.head()
```

Out[26]:

	Date	Open	Close
0	1980-12-12	0.128348	0.128348
1	1980-12-15	0.122210	0.121652
2	1980-12-16	0.113281	0.112723
3	1980-12-17	0.115513	0.115513
4	1980-12-18	0.118862	0.118862

A graph of rolling mean

In [14]:
```
# Set the date column as the index for the Pandas DataFrame
pdf.set_index('Date', inplace=True)

# Plot the rolling mean and closing price against the index
pdf[['rolling_mean', 'Close']].plot(figsize=(25,9))

# Set the title and legend for the plot
plt.title("Rolling Mean and Closing Price of AAPL")
plt.legend(['Rolling Mean', 'Closing Price'])

# Show the plot
plt.show()
```

A graph for standard deviation

In [15]:
```
# Plot the rolling standard deviation and closing price against the index
pdf[['rolling_stddev', 'Close']].plot(figsize=(25,9))

# Set the title and legend for the plot
plt.title("Rolling Standard Deviation and Closing Price of AAPL")
plt.legend(['Rolling Standard Deviation', 'Closing Price'])

# Show the plot
plt.show()
```