B142 Data Integration Final Project

Big Data Processing Pipeline for Stock Market Analysis Using Apache Spark

Name: Aryan Mishra **Student ID:** GH1027140

Project URLs:

To view the notebook either click on this link [https://github.com/aryanwise/Stock-Analysis/blob/main/Aryan%20Mishra%20Final%20Project%20Data%20Integration%20GH1027140.html] , if the notebook is not loading you can download it and view in your local PC.

or use the project link for databricks below

Project Link (Databricks): [https://databricks-prod-cloudfront.cloud.databricks.com/public/4027ec902e239c93eaaa8714f173bcfc/3506588037102990/2899488326664568/633278239033927/latest.html]

GitHub Repository: [https://github.com/aryanwise/Stock-Analysis]

Video Demonstration:

Introduction

This project provides an analysis of how technical indicators can be used to understand and predict stock market movements. In today's data-driven world, the ability to process vast amounts of financial data in real-time is crucial for making informed investment decisions.

My goal was to ingest historical stock data, apply a variety of technical indicators, and perform analyses to generate actionable insights. I focused on several categories of indicators:

- **Momentum Indicators:** Such as RSI, Stochastic RSI, and the Awesome Oscillator.

- **Trend Indicators:** Including Simple Moving Average (SMA), Exponential Moving Average (EMA), and MACD.
- **Volume Indicators:** Like Money Flow Index (MFI) and Force Index (FI).
- **Volatility Indicators:** Including Bollinger Bands and Average True Range (ATR).

By integrating data from multiple stocks and applying these transformations, I aimed to demonstrate a complete data pipeline that turns raw financial data into a valuable analytical asset.

System Design

I designed a data pipeline that follows a logical flow from raw data collection to final analysis, leveraging a modern big data technology stack.

- **Key Components & Technology Choices:**
    1. **Data Source:** I used historical financial data for 27 stock tickers (e.g., AAPL, TSLA, GOOGL) sourced from NASDAQ. The data was provided in CSV format. The dataset link for each ticker is provided in my github (link)
    2. **Processing Engine:** I chose **Apache Spark** (via PySpark on Databricks Community Edition) as the core processing engine due to its powerful distributed computing capabilities, which are ideal for handling large-scale datasets efficiently.
    3. **Data Storage:** The raw and processed data were stored within the **Databricks FileStore (DBFS)**, allowing for an integrated environment where data storage and computation are closely linked.
    4. **Analysis & Visualization:** For in-depth analysis and calculations, I used the **Python** `ta` library, which is built on **Pandas**. For visualization, I used **Plotly** to create interactive and comprehensive charts and dashboards.
- Pipeline Structure:
  My data pipeline follows a simple, effective ETL (Extract, Transform, Load) structure:
    1. **Data Collection:** Historical data for each stock ticker was downloaded and stored in the DBFS catalog under a dedicated folder (`hist_price`).
    2. **Data Storage & Ingestion:** Using PySpark, I ingested the raw CSV files directly from DBFS into my Databricks notebook.
    3. **Data Processing & Transformation:** The initial processing and cleaning (e.g., renaming columns, casting data types) were performed using PySpark. For advanced technical analysis, I converted the Spark DataFrames into Pandas DataFrames to utilize the `ta` library for calculating indicators.
    4. **Analysis:** The final, enriched DataFrames were used to perform various analyses, such as comparing performance across industries and identifying risk-return profiles.

Implementation

The implementation leveraged PySpark for initial data handling and Python libraries for the core analytical tasks.

1. **Data Loading and Preprocessing:** I began by loading the dataset for each of the 27 tickers from its respective CSV file in DBFS using `spark.read.csv()`. A crucial first step was data cleaning and schema normalization. I performed the following operations for each ticker's DataFrame:
    - Renamed the `Close/Last` column to `Close` for consistency.
    - Removed the `$` sign from price columns (`Open`, `High`, `Low`, `Close`) using PySpark's `regexp_replace` function.
    - Casted the price columns to the `float` data type to ensure accurate calculations.
2. **Data Transformation and Integration:** To perform technical analysis, I converted the cleaned PySpark DataFrames into Pandas DataFrames using the `.toPandas()` method. This was necessary because the `ta` library I chose is built to operate on Pandas objects. Using this library, I calculated and appended over a dozen technical indicators (e.g., `RSI`, `MACD`, `Bollinger Bands`) as new columns to each stock's DataFrame. This process enriched the dataset with valuable features for my analysis.
3. **Analysis and Workflow:** Once the data was transformed, I created a dictionary of the Pandas DataFrames, with each stock ticker as a key. This allowed me to easily access and analyze individual stocks or combine them for industry-level insights. For instance, to perform industry-wide comparisons, I added an 'Industry' column to each DataFrame and then concatenated them into a single master DataFrame for visualization.

Challenges and Solutions

During the project, I encountered a few challenges but was able to find effective solutions.

- **Challenge:** My initial plan was to code the technical indicator calculations (like RSI and MACD) from scratch. This proved to be highly complex and time-consuming, and the results were not always accurate.
- **Solution:** After some research, I discovered the Python `ta` (Technical Analysis) library. This library provided pre-built, optimized functions for all the indicators I needed, which saved significant time and ensured my calculations were correct.
- **Challenge:** This was my first time using PySpark extensively. The syntax and methods for data manipulation were different from Pandas, which I was more familiar with. Loading and cleaning the data across multiple files presented a learning curve.
- **Solution:** I referred heavily to the official Apache Spark documentation and the course materials provided in the course module Week 1. This helped me understand the core concepts of Spark DataFrames and functions like `withColumnRenamed` and

`regexp_replace`. I wrote a function to loop through each ticker, apply the cleaning steps, and store the resulting DataFrame, which made the process scalable.
- **Challenge:** Creating visualizations that were both informative and easy to understand was challenging. I wanted to display multiple layers of information (e.g., price, volume, and indicators) in a single, coherent view.
- **Solution:** I chose Plotly because of its powerful support for creating interactive charts and subplots. I was able to build complex dashboards, like the multi-timeframe analysis, by combining candlestick charts, line plots for indicators, and bar charts for volume, which provided a comprehensive analytical tool.

Results

The analysis of the processed data provided several key insights into market behavior and stock performance.

- **Industry Performance Comparison:** By grouping stocks by industry, I analyzed their average daily returns against their volatility (risk). The resulting bar chart showed which industries offered better risk-adjusted returns, providing a high-level view for portfolio allocation.
- **Risk-Return Scatter Plot:** This visualization plotted each stock based on its individual risk (standard deviation of returns) and average daily return. This made it easy to identify stocks in the desirable "top-left" quadrant—those with high returns and low risk.
- **Volatility and Correlation Heatmaps:** I created a heatmap to visualize the volatility of each stock within different industries, quickly highlighting which sectors were more stable. A second heatmap showed the price correlation between stocks, which is crucial for building a diversified portfolio by avoiding assets that move too closely together.
- **Technical Dashboard with Subplots:** For individual stocks like NVIDIA, I created a detailed dashboard featuring a price chart with moving averages, alongside subplots for RSI and MACD. This integrated view allows a user to assess trend, momentum, and potential buy/sell signals all at once.

Conclusion and Future Work

This project successfully demonstrated the design and implementation of a data integration pipeline for financial analysis. Using Apache Spark, I efficiently processed a large dataset of stock prices, and through transformations using Python libraries, I generated meaningful insights. The visualizations created provide clear, actionable information for understanding market trends and individual stock performance.

For **future work**, I have identified two primary extensions:

1. **Machine Learning Prediction:** I have already started developing a Linear Regression model to test the predictive power of the technical indicators. The next step is to build a more sophisticated **Long Short-Term Memory (LSTM)** neural network, which is well-suited for time-series forecasting, to predict future stock prices.
2. **Stock Analysis & Recommendation Dashboard:** I plan to build an interactive dashboard that not only visualizes technical indicators but also incorporates **sentiment analysis** from financial news to provide holistic stock recommendations.
3. Incorporating Streaming - Real Time Data for getting stock prices and news/social media sentiments.