Documentation for Market Sentiment Dashboard

Eric Rusli, Syahreza Raditya Pratama, Lana Solovej

Overview

The Market Sentiment Dashboard focuses on the stock market. It is designed for individuals with some prior knowledge and experience in the stock market who are interested in making informed investment decisions. This dashboard helps users analyze market sentiment and provides valuable insights to guide their investment choices in specific stocks.

The dashboard is designed to provide data analysis for four different stock tickers:

- **AAPL**: Apple Inc.
- **AMZN**: Amazon.com Inc.
- **GOOG**: Alphabet Inc.
- **MSFT**: Microsoft Corporation

Users can select a ticker from the dropdown menu to view detailed information, including real-time stock data, Google Trends analysis, stock price predictions, news articles, sentiment analysis, and the impact of news headlines on stock prices, as analyzed by BART Large MNLI, all tailored specifically for the selected company.

Dashboard Features

1. **Real-Time Stock Data:** Provides up-to-date stock information, including opening price, high/low, volume, market capitalization and more.

2. **Google Trends Integration:** Allows users to analyze search trends for selected keywords, offering insights into market sentiment.

3. **Stock Prediction Models:** Utilizes multiple predictive models (Linear Regression, XGBoost, ARIMA, LSTM) to forecast future stock prices.

4. **News Integration:** Provides recent news articles related to the four stock tickers featured on the dashboard.

5. **Sentiment Analysis:** Employs one sentiment analysis tool (VADER) and two Large Language Models (LLMs) (Twitter RoBERTa and DistilRoberta) to analyze the sentiment of news article headlines related to the featured stock tickers.

6. **BART Large MNLI:** Applies BART Large MNLI to determine whether a news headline suggests a positive, negative, or neutral impact on a company's stock price.

Code Structure

The repository is organized into the following folders: **assets**, **google_trends**, **news_sentiment** and **stock_prediction**, which hold all Python scripts and CSV files necessary for data processing, analysis, and visualization within the Market Sentiment Dashboard. In addition, the repository contains the **app.py** file, which initializes and runs the Dash app, managing the various components of the dashboard.

assets

This folder contains CSS styles for the dashboard components.

google_trends

Contains Python scripts for fetching and processing Google Trends data. It also stores CSV files with the fetched Google Trends data for preselected keywords for each ticker featured on the dashboard.

- **main.py**

  The *plot_from_csv()* function plots Google Trends data from a CSV file using Plotly.

  The *get_trends_data()* function fetches Google Trends data for a list of keywords with a delay to avoid rate limiting.

news_sentiment

Contains Python scripts for fetching and processing news data, applying sentiment analysis (VADER, Twitter RoBERTa, and DistilRoberta), and using BART Large MNLI. Additionally, it stores CSV files with the fetched news data for every featured ticker and the results from the applied models.

- **bart_large_mnli.py**

  The *analyze_impact_on_stock_price_bart_large_mnli()* function analyzes the impact of news articles on stock prices using the BART Large MNLI model. The function loads news data (previously analyzed with Twitter RoBERTa), applies BART Large MNLI classification, calculates the impact probabilities on stock prices, and saves the results to a new CSV file.

- **distilroberta.py**

  The *analyze_news_sentiment_distilroberta()* function analyzes the sentiment of news headlines for a given stock ticker using the DistilRoberta model, which is fine-tuned for financial news sentiment analysis. The function loads the news data, applies sentiment analysis, and saves the results to a new CSV file.

- **finnhub_fetch_news.py**

  The *fetch_and_process_news_data()* function fetches and processes news articles for a given stock ticker from Finnhub over the past 160 days. The function fetches news in chunks, processes the data, and saves it to a CSV file.

  The *verify_missing_dates()* function verifies whether there are any missing dates in the news data within the last 160 days.

- **news_plots.py**

  The *calculate_news_analysis_moving_avg()* function calculates the daily average and 20-day moving average of scores generated by a specified analysis model (e.g., Twitter RoBERTa) over a dataset of news articles. This function is used within the dashboard to visualize the sentiment analysis results from VADER, Twitter RoBERTa, and DistilRoberta models, as well as the natural language inference results from the BART Large MNLI model. The resulting DataFrame serves as the input for other functions that plot these results, enabling visual representation of sentiment trends over time for the sentiment analysis models, and inferred impacts on stock prices for the BART Large MNLI model.

- **news_table.py**

  The *fetch_news_for_ticker()* function fetches news articles for a given stock ticker from Finnhub for the last day.

  The *vader_news_sentiment()* function analyzes the sentiment of news headlines using the VADER sentiment analysis tool.

The *twitter_roberta_news_sentiment()* function analyzes the sentiment of news headlines using the Twitter RoBERTa model.

The *distilroberta_news_sentiment()* function analyzes the sentiment of news headlines using the DistilRoberta model, which is fine-tuned for financial news.

The *bart_large_mnli_impact_on_stock_price()* function analyzes the impact of news articles on stock prices using the BART Large MNLI model.

The *combine_news_analysis_models()* function combines the results from the multiple news analysis models, including sentiment analysis models and BART Large MNLI, into a single DataFrame.

- **twitter_roberta.py**

  The *analyze_news_sentiment_twitter_roberta()* function analyzes the sentiment of news headlines for a given stock ticker using the Twitter RoBERTa model. The function loads the news data, applies sentiment analysis, and saves the results to a new CSV file.

stock_prediction

This contains python scripts for Linear Regression, ARIMA, LSTM and XGBoost models.

- **arima.py**

  The *test_stationarity()* function performs the Augmented Dickey-Fuller (ADF) test on a given time series to determine if it is stationary. It calculates and prints rolling statistics and returns the p-value of the test.

  The *ARIMA_model()* function etches historical stock data for a specified ticker and time period, tests the stationarity of the data, and applies necessary transformations. It then fits an ARIMA model to the data, generates a forecast for the specified number of days, and returns the historical data, forecasted prices, and confidence intervals.

  The *add_noise()* function adds Gaussian noise to the forecasted data to introduce variability. It returns the forecasted data with the added noise.

- **lr.py**

  The *test_stationarity()* function performs the Augmented Dickey-Fuller (ADF) test on a given time series to assess whether it is stationary, printing out the test statistics and p-value.

The *create_features()* function generates input features and corresponding labels for time series prediction by using a sliding window approach, creating sequences of data for training a model.

The *LR_model()* function trains a Linear Regression model on historical stock prices fetched for a specified ticker and time range. It uses the model to predict both past prices (for evaluation) and future prices, returning DataFrames containing actual, predicted, and forecasted prices.

- **lstm.py**

  The *create_dataset()* function converts a time series dataset into input-output pairs (X, Y) for training a model, where X consists of a specified number of previous time steps and Y is the next time step in the series.

  The *LSTM_model()* function builds, trains, and evaluates an LSTM (Long Short-Term Memory) model on historical stock prices for a specified ticker and time range. It returns two DataFrames: one containing the actual prices and the model's predictions for training and testing, and another with forecasted future prices.

- **xgboost.py**

  The *create_dataset()* function transforms a time series dataset into input-output pairs (X, Y) for training a model, where X consists of a specified number of previous time steps and Y is the next time step in the series.

  The *XGBoost_model()* function builds, trains, and evaluates an XGBoost model on historical stock prices for a specified ticker and time range. It returns two DataFrames: one containing the actual prices along with the model's predictions for both the training and testing sets, and another with forecasted future prices.

app.py

This is the main file of the project that initializes and runs the Dash app. It implements and manages the various components of the dashboard application. The high-level structure of the file is as follows:

1. **Imports and Dependencies**: The necessary libraries are imported at the beginning.

2. **Stock Data Retrieval**: Uses the *yfinance* library to fetch the current stock data for a given ticker. The data includes stock prices, volume, market capitalization, etc., which are displayed on the dashboard.

3. **Prediction Models**: Implements four different stock prediction models and visualizes their results:

    - **Linear Regression:** A foundational model that establishes a clear and interpretable prediction curve for short to midterm forecasting

    - **XGBoost (Extreme Gradient Boosting):** XGBoost is a machine learning powerhouse that excels in prediction tasks by optimizing model accuracy through boosting.

    - **ARIMA (AutoRegressive Integrated Moving Average) :** ARIMA is particularly effective for producing highly accurate forecasts in stable and cyclical market conditions.

    - **LSTM (Long Short-Term Memory):** LSTM model is finely tuned to capture and predict time-series data with long-range dependencies.

    These models are used to forecast future stock prices based on historical data.

4. **Google Trends Data**: Retrieves and visualizes Google Trends data for specific keywords, helping users gauge public interest in a stock over time.

5. **News:** News data is fetched in real-time from the *Finnhub Stock API* and displayed in the news table on the dashboard.

6. **Sentiment Analysis:** The dashboard employs one sentiment analysis tool (VADER) and two Large Language Models (LLMs) (Twitter RoBERTa and DistilRoberta) to analyze the sentiment of news article headlines related to the four tickers featured on the dashboard. The sentiment analysis results are both displayed and visualized on the dashboard.

    - **VADER**: VADER (Valence Aware Dictionary and sEntiment Reasoner) is a lexicon and rule-based sentiment analysis tool specifically attuned to sentiments expressed in social media, while also applicable to texts from other domains.

    - **Twitter RoBERTa**: Twitter RoBERTa is a transformer-based model fine-tuned specifically for sentiment analysis. It was trained on approximately 124 million sentiment-annotated tweets from January 2018 to December 2021.

    - **DistilRoberta**: DistilRoberta is a distilled version of the RoBERTa-base model, which is on average twice as fast. It was trained on sentences from

financial news categorized by sentiment, making it particularly suitable for sentiment analysis in financial contexts.

7. **BART Large MNLI: Impact on Stock Price:** BART Large MNLI is applied to determine whether a news headline suggests a positive, negative, or neutral impact on a company's stock price. The results of BART Large MNLI are displayed on the dashboard.

    - **BART Large MNLI**: BART Large MNLI is a versatile model used for Natural Language Inference (NLI) tasks and zero-shot classification. This model classifies text into categories without requiring task-specific training by converting the input text into a premise and comparing it to user-provided labels as hypotheses. It calculates the probabilities of each hypothesis being an entailment, contradiction, or neutral in relation to the premise. The hypothesis with the highest probability of entailment is chosen as the most likely category.

8. **Dashboard Layout**: The main layout of the dashboard is defined using Bootstrap components. It includes elements for the navigation bar, stock graphs, stock indicators, prediction and news tabs, Google Trends, news table, and visualizations of the sentiment analysis results and BART Large MNLI outcomes.

9. **Styling and Layout:** The visual appearance and layout of the dashboard are managed using styles defined in the assets/styles.py and assets/styles.css files. These files include CSS styles that are applied to various components throughout the dashboard. Styles such as the navigation bar (NAVBAR_STYLE), layout container (LAYOUT_STYLE) and others ensure that the dashboard maintains a consistent and visually appealing design. Custom styles are applied using Dash's style attribute, which enhances the user experience by ensuring that elements are well-aligned, appropriately colored, and responsive to different screen sizes.

10. **Callbacks**: Dash callbacks are used to dynamically update the dashboard based on user input. These callbacks handle updates to stock data, prediction graphs, the news table, sentiment analysis and BART Large MNLI plots, as well as Google Trends data when a new stock ticker is selected or new keywords are entered.

11. **Running the Application**: The application is set to run locally and it automatically opens in the default web browser.

Deploying the Application

The application is deployed on Google Cloud Platform, utilizing a Docker container to ensure a consistent and scalable environment. The Dockerfile sets up the working python environment, installs the necessary dependencies, and uses Gunicorn to serve the application efficiently. After creating a new project in Google Cloud, a unique project ID is assigned, which is used in the deployment process. The application is then deployed using the Google Cloud with a few terminal commands:

- gcloud config set project <project-id>
- gcloud run deploy –source .

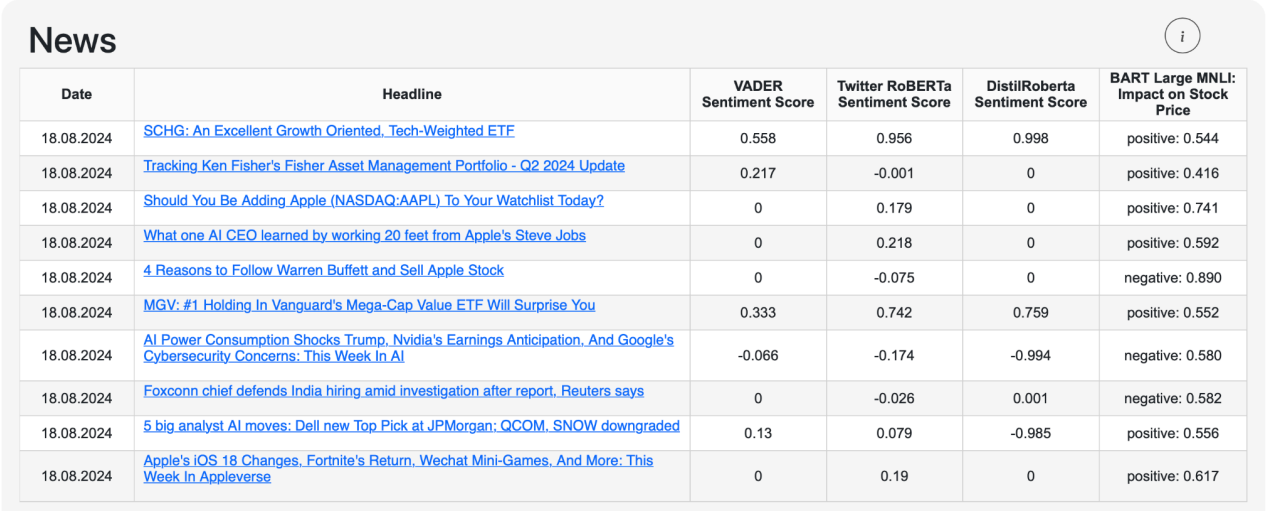
The deployment process typically takes around 30 minutes. Once the deployment is complete, the url to the website will be provided.

The deployed website can be accessed live by clicking this link:

https://appliedpredictiveanalytics-b6rsdb5r2a-lm.a.run.app.

Explanation on Errors

After deployment, we noticed that the news table is not displaying correctly on the deployed website, although the data is being loaded. No errors related to the news sentiment table were observed in the Logs Explorer on the Google Cloud Platform. The news table is supposed to look like the following:



News

Date	Headline	VADER Sentiment Score	Twitter RoBERTa Sentiment Score	DistilRoberta Sentiment Score	BART Large MNLI: Impact on Stock Price
18.08.2024	SCHG: An Excellent Growth Oriented, Tech-Weighted ETF	0.558	0.956	0.998	positive: 0.544
18.08.2024	Tracking Ken Fisher's Fisher Asset Management Portfolio - Q2 2024 Update	0.217	-0.001	0	positive: 0.416
18.08.2024	Should You Be Adding Apple (NASDAQ:AAPL) To Your Watchlist Today?	0	0.179	0	positive: 0.741
18.08.2024	What one AI CEO learned by working 20 feet from Apple's Steve Jobs	0	0.218	0	positive: 0.592
18.08.2024	4 Reasons to Follow Warren Buffett and Sell Apple Stock	0	-0.075	0	negative: 0.890
18.08.2024	MGV: #1 Holding In Vanguard's Mega-Cap Value ETF Will Surprise You	0.333	0.742	0.759	positive: 0.552
18.08.2024	AI Power Consumption Shocks Trump, Nvidia's Earnings Anticipation, And Google's Cybersecurity Concerns: This Week In AI	-0.066	-0.174	-0.994	negative: 0.580
18.08.2024	Foxconn chief defends India hiring amid investigation after report, Reuters says	0	-0.026	0.001	negative: 0.582
18.08.2024	5 big analyst AI moves: Dell new Top Pick at JPMorgan; QCOM, SNOW downgraded	0.13	0.079	-0.985	positive: 0.556
18.08.2024	Apple's iOS 18 Changes, Fortnite's Return, Wechat Mini-Games, And More: This Week In Appleverse	0	0.19	0	positive: 0.617