End-to-End System Design for Stock Price Prediction

TeamSemiColon

1. Introduction

This document outlines the end-to-end system design for deploying a stock price prediction model in a financial analysis firm. The proposed system ensures **real-time data ingestion, feature engineering, scalable model training, monitoring, and insight delivery** to analysts and brokers.

Goals of the System:

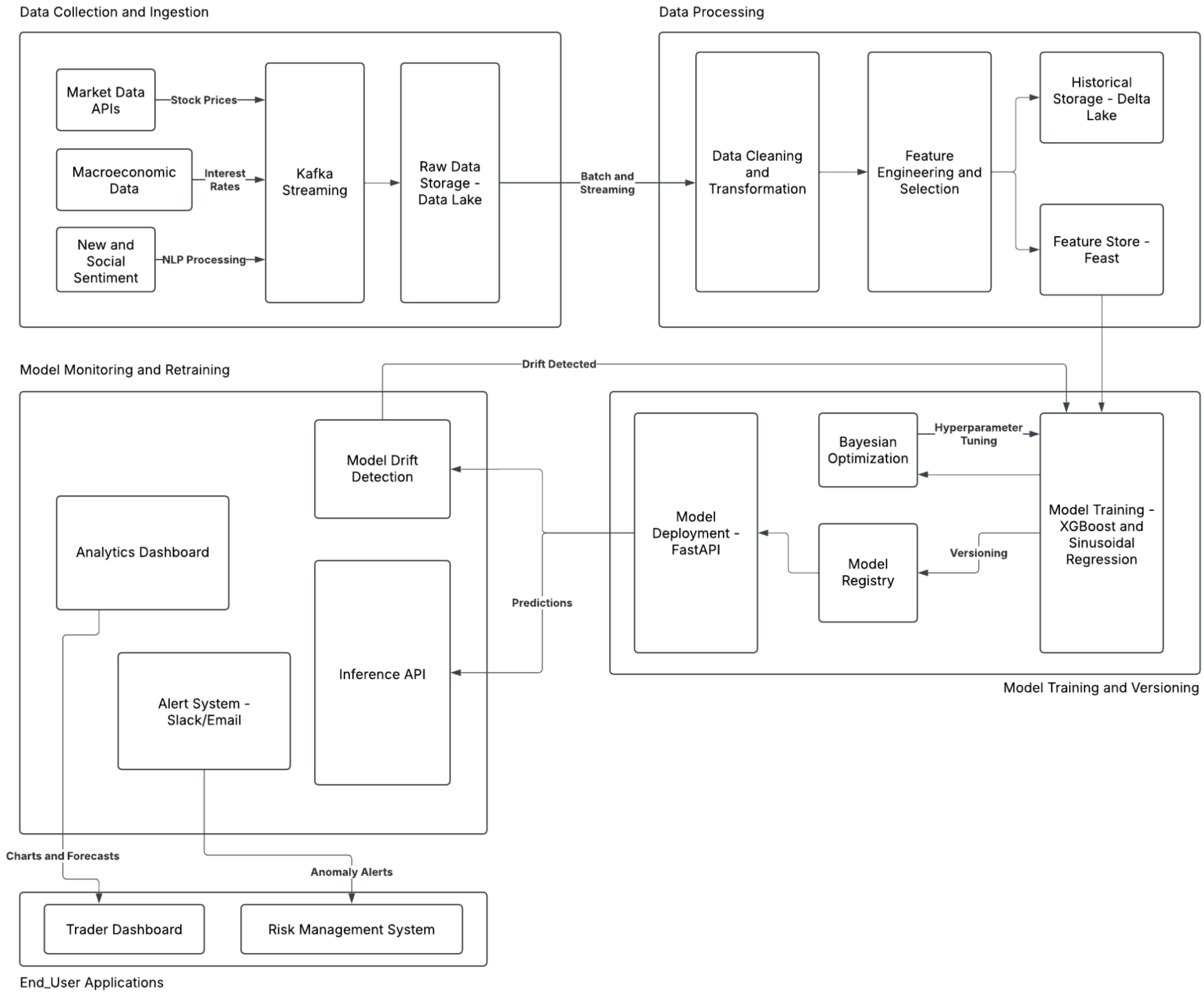
- Automate data ingestion from multiple sources.
- Engineer high-quality features and ensure consistency between training and inference.
- Deploy a **hybrid prediction model (Sinusoidal Regression + XGBoost)** with high accuracy.
- Provide **low-latency, scalable, and monitored predictions** via APIs and dashboards.
- Trigger **automatic retraining upon model drift detection**.

---

2. System Architecture

The system consists of multiple components for **data collection, processing, model training, inference, monitoring, and user insights**.

2.1 System Architecture Diagram



3. Component Justification

Component	Technology Stack	Justification	Tradeoffs
Data Streaming	Kafka + Spark	Real-time stock & macroeconomic data processing	Requires infra overhead
Data Storage	AWS S3 (raw), Redshift (structured), Delta Lake (historical)	Scalable and cost-effective for big data	Higher latency in querying
Feature Engineering	Feast (Feature Store), Pandas, NumPy	Ensures consistency between training & inference	More complex feature management
Model Training	XGBoost, Sinusoidal Regressor, Bayesian Optimization	Hybrid approach captures trends + cyclical patterns	Computationally expensive
Model Deployment	FastAPI + Kubernetes + Redis Cache	Low-latency, scalable inference	Needs load balancing
Monitoring	Evidently AI, Prometheus, Grafana	Detects drift, automates alerts & retraining	Requires tuning thresholds
User Insights	Streamlit, Plotly Dash, Webhooks	Intuitive UI & real-time alerts	Web dashboards need high availability

---

4. Advanced Data Flow Analysis

4.1 Step 1: Data Collection & Storage

- **Streaming Data:**
  - Stock prices and news sentiment are processed **in real-time via Kafka**.
  - Market indicators (inflation, GDP) fetched via APIs **hourly**.
- **Batch Processing:**
  - **Daily historical data** stored in **Delta Lake** for long-term analysis.
  - Features precomputed and stored in **Feast (Feature Store)**.

4.2 Step 2: Feature Engineering & Model Training

- **Feature Engineering:**
  - Generates **lag variables**, **seasonal indicators**, and **sentiment scores**.
  - Stores **precomputed features in Feast** for efficient inference.
- **Training Pipeline:**
  - Uses **Boosted Hybrid Model**:
    - **Sinusoidal Regression** captures **seasonal patterns**.
    - **XGBoost** models the **residuals and non-linear dependencies**.
  - **Bayesian Optimization** automatically tunes hyperparameters.

4.3 Step 3: Model Deployment & Inference

- **FastAPI exposes a REST endpoint** for traders & analysts.
- **Redis Cache** stores recent predictions to reduce latency.
- **Inference Pipeline:**
  - Retrieves latest stock price features from **Feast Feature Store**.
  - Runs prediction and **returns results in <100ms**.

4.4 Step 4: Model Monitoring & Retraining

- **Monitoring:**
  - Uses **Evidently AI & Prometheus** to detect drift.
  - **Alert system (Slack, Email, Webhooks)** triggers if predictions deviate from actuals.
- **Retraining Triggered If:**
  - **Data Drift:** Stock price behavior changes significantly.
  - **Concept Drift:** Model performance deteriorates (high RMSE, low R^2).

---

5. Key Challenges & Solutions

Challenge	Solution
Latency in Real-Time Predictions	Redis Caching for faster retrieval
Data Drift Due to Market Conditions	Evidently AI for auto-retraining
High Infrastructure Costs	Use AWS Spot Instances & Kubernetes auto-scaling
Multiple Data Sources Complexity	Centralized Feature Store (Feast) for consistency
Outlier Impact on Model Performance	Anomaly detection with Z-score filtering

---

6. Conclusion

This **detailed architecture** ensures a **scalable, robust, and high-performance stock price prediction system**. By integrating **real-time data ingestion, hybrid modeling, scalable infrastructure, and automated retraining**, the system continuously delivers value to financial analysts and traders.

With further enhancements like **alternative boosting methods, external macroeconomic indicators, and additional deep learning models**, the system can **further improve accuracy and robustness** in volatile financial markets.

Next Steps:

- Implement **CI/CD pipelines for continuous deployment**.
- Optimize **infrastructure costs** using cloud auto-scaling.
- Extend **multi-asset support** for broader financial applications.