Below is a **step-by-step** project plan to build an **Early Crypto Rally (or Crash) Detection & Notification System** using your specified tech stack:

## **1. Project Setup & Requirements**

1. **Team Roles**:  
   * *Data Engineer*: Handles data ingestion (Kafka/Airflow) and pipeline setup.
   * *Data Scientist/ML Engineer*: Focuses on model development (time-series, anomaly detection, LLM sentiment analysis).
   * *DevOps/Deployment Engineer*: Oversees Flask API deployment, Dockerization, MLOps infrastructure, Grafana setup.
   * *BI Specialist*: Designs and configures Tableau/Power BI dashboards.
2. **Infrastructure & Tools**:  
   * **Snowflake**: Central data warehouse for storing crypto prices, social sentiment, historical data, etc.
   * **Firecrawl**: Web scraping tool for additional data (e.g., niche crypto forums or aggregator sites if no official API is available).
   * **Kafka** or **Airflow**: Chosen method for data ingestion/orchestration.
   * **Spark/Snowpark**: Large-scale data processing and feature engineering.
   * **Python ML stack** (scikit-learn, XGBoost, plus time-series libraries like Prophet or PyTorch for more advanced modeling).
   * **LLM** (Deepseek / LangChain) for sentiment or advanced NLP on social media text.
   * **MLOps** (model versioning, automated re-training) + **Grafana** for monitoring.
   * **Tableau/Power BI**: Dashboards for stakeholders/investors.
   * **Flask**: Deployment of ML inference for real-time detection.
   * **Zapier**: Optional automation to send alerts (email, SMS, Slack) when a rally or crash signal is detected.

## **2. Data Ingestion Pipeline**

### **2.1 Exchange Price Data (Real-Time or Near-Real-Time)**

1. **Select Data Source**:  
   * Official exchange APIs (e.g., Binance, Coinbase, Kraken, or aggregator APIs like CoinGecko, CoinMarketCap).
   * Decide how frequently you’ll pull data (every minute, every 5 seconds, etc.).
2. **Ingestion Method**
   * If you want near *real-time streaming*, use **Kafka**:
     1. A small Python/Node.js microservice calls the exchange API at the desired interval.
     2. Publishes the price data as messages to a **Kafka** topic.
     3. A Kafka consumer pulls these messages and writes them to **Snowflake** (possibly via Spark streaming or a Kafka-Snowflake connector).
   * Alternatively, if *batch ingestion* is enough (every minute or 5 minutes), use **Airflow**:
     1. An Airflow DAG runs an API call.
     2. Transforms the response (if needed).
     3. Inserts data into Snowflake.
3. **Data Schema**

**Snowflake** table example:  
 CREATE TABLE crypto\_price\_data (

timestamp TIMESTAMP,

symbol VARCHAR,

exchange VARCHAR,

price FLOAT,

volume FLOAT,

...

);

### **2.2 Social Media Sentiment Data**

1. **Twitter & Reddit**:  
   * Use official APIs or **Firecrawl** if you need to scrape certain pages/threads.
   * Identify relevant keywords/hashtags (#bitcoin, #eth, etc.) or subreddits (r/CryptoCurrency, r/Bitcoin).
2. **Ingestion**
   * For near real-time, connect the data to **Kafka** as well.
   * Or set up **Airflow** to run scraping tasks / API pulls every few minutes.
   * Store the raw text and metadata (user, timestamp, likes/upvotes, etc.) in **Snowflake**.

**Table Structure** CREATE TABLE social\_media\_posts (

post\_id VARCHAR,

platform VARCHAR, -- e.g., 'twitter' or 'reddit'

text VARCHAR,

timestamp TIMESTAMP,

sentiment\_score FLOAT,

...

);

## **3. Data Processing & Enrichment (Spark / Snowpark)**

1. **Data Cleaning & Normalization**
   * Convert timestamps to a unified format (UTC).
   * Handle missing fields (e.g., if volume isn’t reported by some exchanges).
2. **Feature Engineering**
   * **Price-Based Features**:
     + Moving averages (e.g., 1h, 6h, 24h).
     + Price change percentages over various time windows.
     + Volume change, order book depth (if available).
   * **Sentiment-Based Features**:
     + Calculate aggregated sentiment scores from social media. (Either use a pretrained sentiment model, or the LLM for classification. See “LLM Usage” below.)
     + Frequency of bullish/bearish keywords.
     + Weighted sentiment (factoring user credibility or follower count if relevant).
3. **Data Aggregation**
   * Use **Spark** or **Snowpark** to join price data with social sentiment data on a time-based window.
   * Create a consolidated dataset that includes: [timestamp, price, volume, sentiments, technical indicators, etc.].
4. **Storage in Snowflake**

Write the enriched dataset back to Snowflake in a curated table:  
 CREATE TABLE crypto\_features (

timestamp TIMESTAMP,

symbol VARCHAR,

price FLOAT,

volume FLOAT,

rolling\_avg\_1h FLOAT,

rolling\_avg\_24h FLOAT,

sentiment\_score FLOAT,

...

);

## **4. LLM Usage for Sentiment & Trend Analysis**

1. **Basic Sentiment**:  
   * Start with a standard sentiment model (e.g., a pretrained transformer from Hugging Face) to assign a sentiment score (+1 to -1).
   * Or use an LLM (Deepseek / LangChain) to classify posts more contextually (e.g., “bearish vs. bullish,” “neutral,” “FUD,” etc.).
2. **LangChain** or **Deepseek**
   * Construct a pipeline where the LLM receives a snippet of text (tweet or Reddit post) and outputs:
     + A classification: {“bullish”, “bearish”, “neutral”}
     + An explanation or summary of the text (optional).
   * Store these results in Snowflake for further aggregation (e.g., average sentiment per hour).
3. **Scaling**
   * If you have a large volume of posts, consider using a smaller or more efficient LLM, or do partial sampling of tweets to stay within compute/time limits.
   * For real-time streaming, you might batch requests or scale horizontally to handle the throughput.

## **5. ML Modeling for Rally/Crash Detection**

1. **Goal**: Identify significant upward (rally) or downward (crash) price movements *before* they happen or *as* they begin.
2. **Model Options**
   * **Time-Series Forecasting**: e.g., *Facebook Prophet*, *ARIMA*, or *LSTM*.
   * **Anomaly Detection**: Possibly using *XGBoost* or an unsupervised method (e.g., isolation forests) to detect unusual spikes or drops.
   * **Classification Approach**: Convert rally detection into a classification problem: *“Likely to rally in next 1 hour” vs. “Not likely to rally.”*
3. **Training Pipeline**
   * Pull the latest features from **Snowflake**.
   * Split data into training/validation sets.
   * Fit the model (e.g., XGBoost with features like sentiment, price trends).
   * Evaluate metrics (precision, recall, F1-score, or RMSE if it’s a regression forecast).
4. **Model Serving**
   * Save the trained model to a centralized location (S3, local file, or Snowflake external stage).
   * Use a **Flask** microservice for inference (details in Step 7).
5. **Automated Re-Training** (MLOps)  
   * Set up an **Airflow** DAG or a scheduled job that re-trains the model daily or weekly with the newest data.
   * Version each model artifact (MLflow or a simple versioning scheme in Snowflake).

## **6. MLOps & Monitoring (Grafana)**

1. **Pipeline Monitoring**
   * Track data ingestion rates (Kafka/Airflow tasks), latency, and error rates in real-time dashboards.
   * **Grafana** can connect to metrics from your ingestion pipeline or from Spark cluster metrics.
2. **Model Performance**
   * Log predictions to a table in Snowflake.
   * Periodically compare predictions with actual outcomes (did a rally actually happen?).
   * Display performance metrics (accuracy, F1) in **Grafana**.
   * Watch for data drift: e.g., if sentiment trends shift drastically over time.
3. **Alert Thresholds**
   * If model accuracy drops below a certain threshold, Grafana can trigger an alert.
   * Automatic re-training or human intervention can be scheduled.

## **7. Deployment & Notification Layer**

### **7.1 Flask Microservice**

1. **Inference Endpoint**:  
   * POST /detect – Accepts relevant input (crypto symbol, timeframe), returns a *rally score* or *crash likelihood*.
   * GET /health – Basic health check.
2. **Implementation**:  
   * Load the latest model version on startup.
   * Pull real-time or near real-time data from Snowflake (or in-memory) to generate predictions.
   * Return predictions + optional interpretation (e.g., “Bullish sentiment is high, 10% chance of a sudden rally in next 30 minutes”).
3. **Containerization**:  
   * Dockerize the Flask app.
   * Deploy to a low-cost cloud environment (AWS EC2, Azure VM, or a container service like ECS/AKS).

### **7.2 Notification & Automation**

1. **Zapier** (Optional):  
   * Triggered by an API call or a published event from the Flask service.
   * Send email, Slack, or SMS alerts to subscribed users if the “rally likelihood” or “crash likelihood” exceeds a preset threshold.
2. **Alert Customization**:  
   * Users can set thresholds (e.g., “Alert me if Bitcoin is predicted to rise 5% in the next hour”).

## **8. Data Visualization (Tableau / Power BI)**

1. **Real-Time Dashboards**
   * Visualize price trends, sentiment over time, and predicted rally/crash signals.
   * Combine line charts of price with bar charts of sentiment scores to see correlations.
2. **Historical Insights**
   * Analyze which signals triggered the best predictions historically.
   * Display success rates of the model’s “early warnings.”
3. **User Segmentation**
   * If you want to differentiate by coin or user preference, allow filtering for different symbols (BTC, ETH, etc.).

## **9. Detailed Timeline (Approx.)**

1. **Week 1**
   * *Infrastructure Setup*: Snowflake account, Kafka/Airflow environment, Spark cluster, initial code repositories.
   * *API Access & Scraping Setup*: Exchange APIs, Twitter/Reddit authentication, Firecrawl config.
2. **Week 2**
   * *Data Ingestion Pipeline*: Implement Kafka or Airflow DAGs, test ingestion into Snowflake.
   * *Initial Data Model in Snowflake*: Create staging & curated tables.
3. **Week 3**
   * *Spark / Snowpark Data Processing*: Feature engineering, merging price + sentiment.
   * *LLM Pipeline for Sentiment*: Basic classification on sample tweets or Reddit posts.
4. **Week 4**
   * *ML Modeling*: Start with a straightforward approach (XGBoost or Prophet).
   * *Evaluation & Tuning*: Improve accuracy, handle overfitting, gather labeled “rally events” for training.
5. **Week 5**
   * *Deployment Prep*: Build Flask microservice, containerize with Docker.
   * *MLOps & Grafana*: Configure dashboards for pipeline metrics, model performance.
   * *Integration Testing*: End-to-end tests from data ingestion → model inference → alerts.
6. **Week 6**
   * *User-Facing Dashboards*: Tableau or Power BI for historical and real-time visualization.
   * *Alerting & Automations (Zapier)*: Set up alerts for thresholds.
   * *Final Optimization*: Tweak model, confirm performance, finalize documentation.

## **10. Potential Extensions & “Wow” Factors**

1. **Multi-Coin / Multi-Market Support**: Expand beyond BTC and ETH to track altcoins.
2. **Multi-LLM Ensemble**: Combine different LLM sentiment analyses to see if aggregator sentiment is more accurate.
3. **Event Detection**: Incorporate news scraping from major crypto publications (CoinDesk, Decrypt) with **Firecrawl**, then let an LLM rank “breaking news” impact.
4. **Advanced Graph Analysis**: Analyze on-chain data (if feasible) to detect large whale movements or unusual transaction patterns.
5. **Mobile App or Web Front-End**: Provide a user-friendly dashboard or real-time notifications on mobile.

# **Putting It All Together**

By following the steps above, you’ll create a system that **continuously ingests** crypto price data and social media sentiment, **analyzes** trends in near real-time, **predicts** potential rallies or crashes, and **notifies** users through automated alerts. Throughout the pipeline, you’ll utilize:

* **Firecrawl** for scraping (if needed beyond official APIs).
* **Kafka/Airflow** for orchestrating data ingestion into **Snowflake**.
* **Spark/Snowpark** for large-scale data transformation and feature engineering.
* **LLM** to interpret and classify social sentiment.
* **Python ML** (XGBoost, Prophet, etc.) for time-series/anomaly detection.
* **MLOps** with **Grafana** to monitor pipeline health and model performance.
* **Tableau/Power BI** for visual dashboards.
* **Flask** for model serving and real-time inference.
* **Zapier** for alerting or workflow automation.

This step-wise approach ensures a **production-grade** solution that can scale, adapt over time, and continuously improve through automated re-training and monitoring.