**Real-Time Stock Prediction Pipeline with Kafka & LSTM**

**1. Environment Setup**

**Kafka Cluster**

* Install Apache Kafka locally or using Docker.
* Create necessary Kafka topics:
  + **stock-input-data-topic** : For producer messages (real-time stock data input).
  + **stock-prediction-data-topic**: For output messages (model predictions).

**Programming Language & Libraries**

* Python is used along with:
  + kafka-python: For Kafka producer and consumer.
  + tensorflow: For building the LSTM model.
  + pandas: For data processing.
  + scikit-learn: For normalization and performance evaluation.
  + matplotlib: to Plot Loss and actual/predicted values

**2. Data Preprocessing**

* Load NIFTY50\_all.csv dataset.
* Normalize Open, High, Low, Close, and Volume columns.
* Encode stock symbols as categorical data.
* Create sequences of **10 days** of stock prices for training the LSTM model.

**3. Producer Implementation**

**Data Generation**

* The Kafka producer reads real-time stock data.
* Data can be:
  + Simulated stock prices.
  + Live data from APIs.

**Message Formatting**

Each Kafka message includes:

* A unique identifier.
* The input features in JSON format.
* A timestamp.

**4. Consumer Implementation with Neural Network Prediction**

**Data Ingestion & Preprocessing**

* The Kafka consumer listens to the input-data topic.
* Messages are parsed and pre-processed to match the LSTM model's input format.

**Neural Network Model**

This **LSTM-based neural network** is designed for **time-series forecasting**, specifically predicting **stock closing prices** based on past trends.

* **Two LSTM layers** (64 & 32 units) extract patterns from historical stock data.
* **Dropout layers (20%)** prevent overfitting and improve generalization.
* **A Dense layer (16 neurons, ReLU activation)** helps refine extracted features.
* **The final Dense layer (1 neuron)** outputs the predicted closing price.

This architecture enables the model to **capture sequential dependencies**, making it ideal for stock market prediction.

**Training Phase**

* The LSTM model is trained on historical stock data (regression task for predicting next-day prices).
* The dataset is split into:
  + **Train (80%)**: Model training.
  + **Validation (10%)**: Hyperparameter tuning.
  + **Test (10%)**: Model evaluation.

history = model.fit(

X\_train, y\_train,

validation\_data=(X\_val, y\_val),

epochs=10, batch\_size=16, verbose=1

)

**Integration into Consumer**

* The trained model is loaded in the consumer.
* Each incoming message is fed into the LSTM model for prediction.
* Predictions are denormalized using scaler.pkl.
* The consumer publishes predictions to the predictions topic.

joblib.dump(scaler, "scaler.pkl")

joblib.dump(scaler\_close, "scaler\_close.pkl")

**5. Running Producer and Consumer Asynchronously**

To ensure real-time processing, both the producer and consumer run in separate threads:

import concurrent.futures

with concurrent.futures.ThreadPoolExecutor(max\_workers=2) as executor:

executor.submit(run\_producer, df\_stock)

executor.submit(start\_consumer)

**6. Testing and Evaluation**

**Performance Metrics**

* Predictions are evaluated using:
  + **Root Mean Squared Error (RMSE)** for accuracy.
  + **R-squared score** for model performance.

**Real-Time Monitoring**

* Implement logging or a dashboard to monitor:
  + Number of messages processed.
  + Prediction latency.
  + Performance metrics over time.

Model Training Loss Graph

A graph of a training loss

AI-generated content may be incorrect.

Actual vs Predicted Closing price

