

Financial Data Analysis

Gold Price Prediction

Extract Data from SQL Database

First, we connect to the `financial_data` database and retrieve data from tables representing different financial markets. Each table corresponds to a specific market or asset, such as indices, commodities, or currencies. The retrieved data is converted into CSV files for external storage and pandas DataFrames for in-memory processing and analysis.

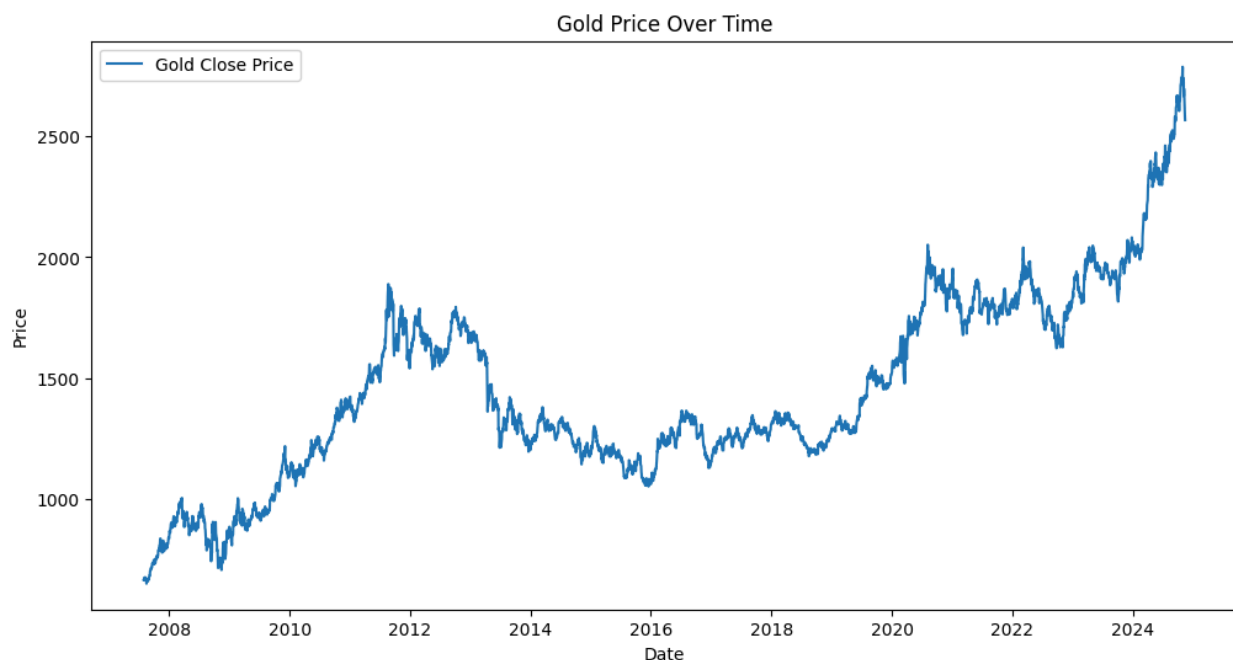
Exploratory Data Analysis (EDA)

Exploratory Data Analysis (EDA) is the process of analyzing and summarizing a dataset's main characteristics, often using visual and statistical techniques, before applying any formal modeling. EDA is a critical step in the data analysis process as it helps to:

- Understand the structure of the data.
- Detect patterns, trends, and relationships.
- Identify missing data, outliers, or anomalies.
- Validate assumptions and prepare the data for further analysis or modeling.

1. Visualize time series for gold prices

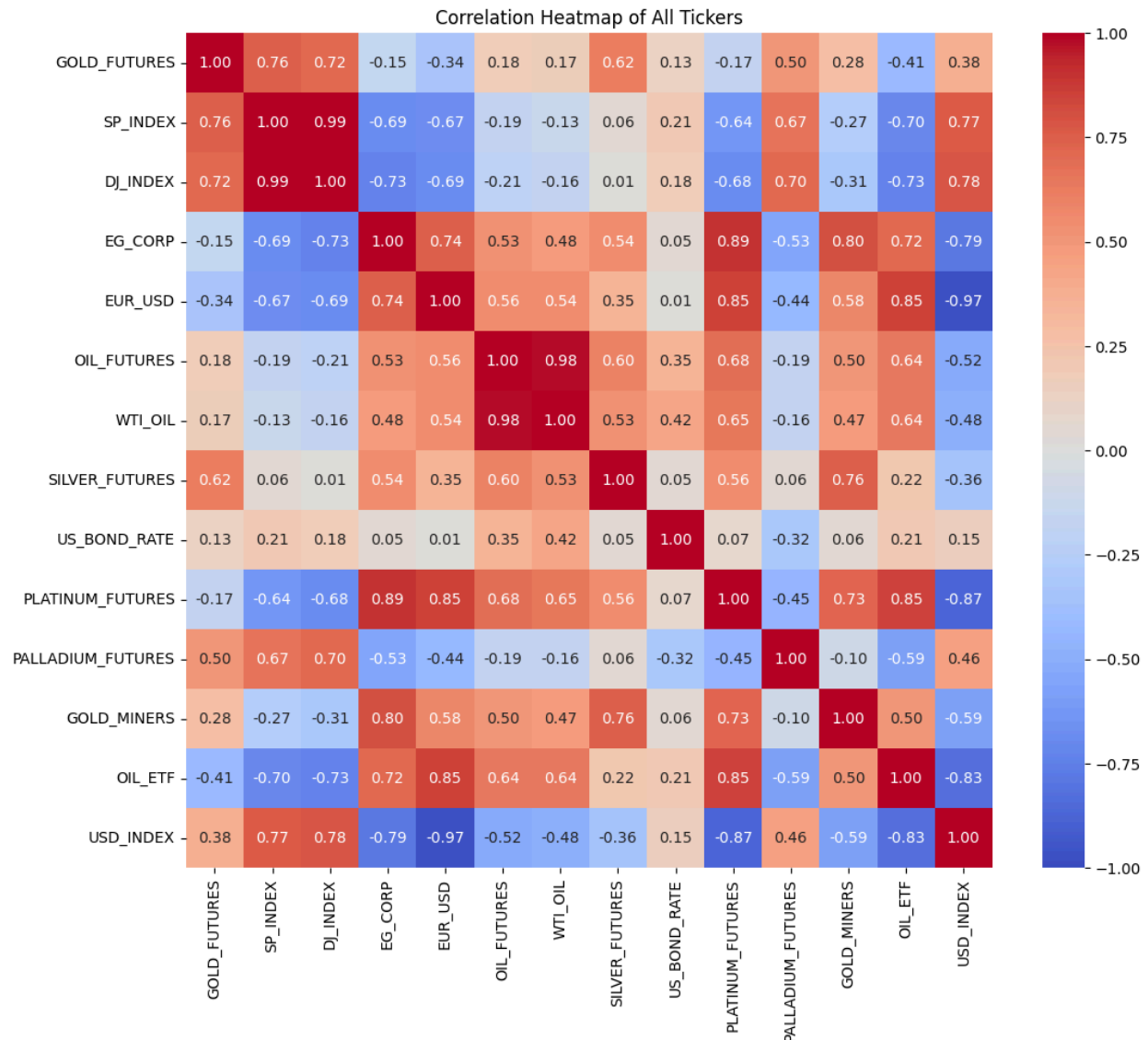
A plot of gold prices and other tickers over time reveals long-term trends, periodic patterns, and abrupt changes.



2. Compute correlations between gold prices and other tickers

Correlation coefficients quantify the strength and direction of relationships between gold prices and other tickers. Understanding whether assets move together (positive correlation) or inversely (negative correlation) aids in building diversified investment portfolios.

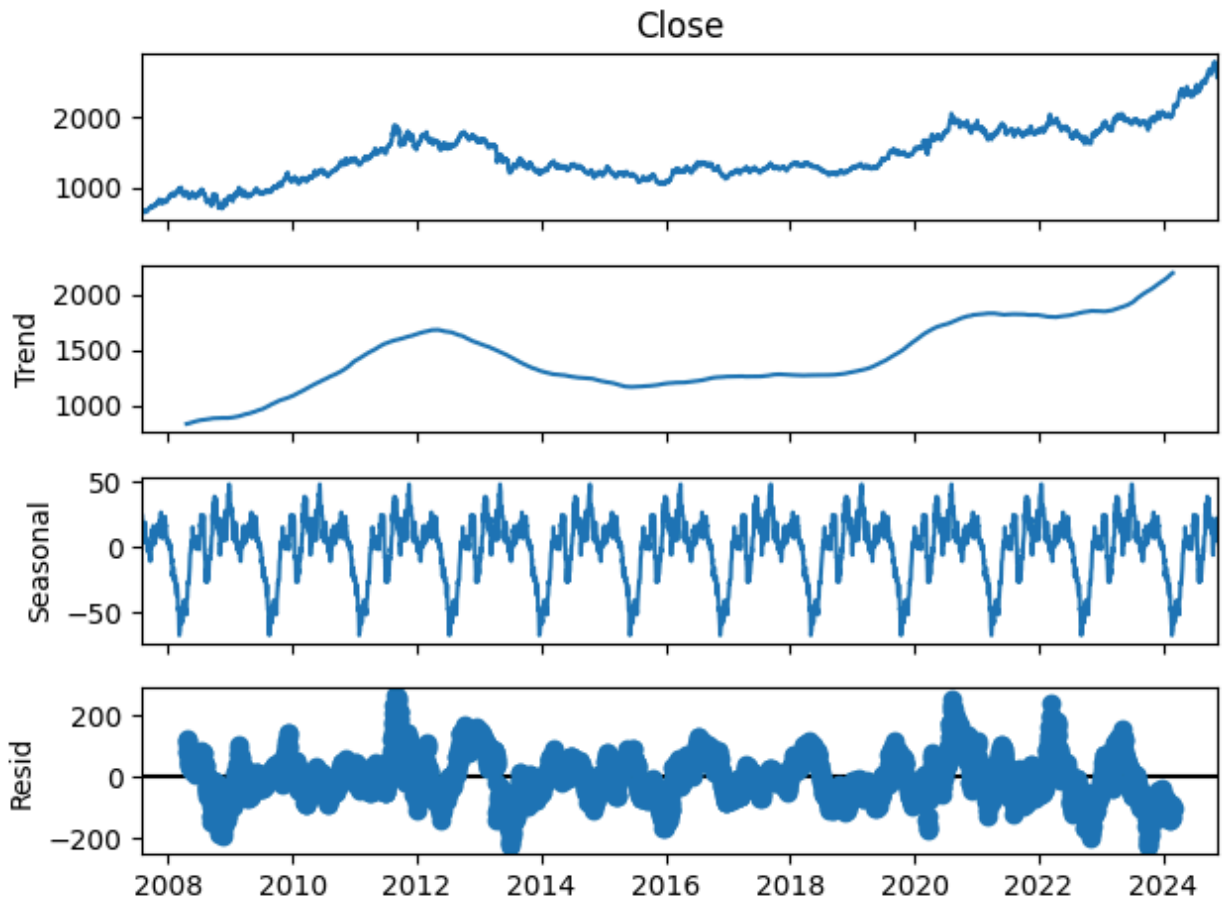
Correlation analysis can highlight potential risks if assets are strongly related and vulnerable to similar shocks.



3. Seasonality and Trend Analysis Using Decomposition

Identifies long-term movement in prices, independent of seasonal or random fluctuations. It can recognize repeating patterns (e.g., gold prices rising in December due to holiday demand or geopolitical events).

By isolating trends and seasonality, decomposition allows for better predictive modeling. Seasonality and trends provide context for macroeconomic or geopolitical events influencing asset prices.



These steps collectively build a foundational understanding of the time series data. They help uncover the **"what"** (visualization), the **"how"** (correlations), and the **"why"** (trends/seasonality) behind asset price movements.

Feature Engineering

1. Add Lagged Features

Lagged features are often used in time series models to incorporate previous values of a variable into predicting future values. For example, the closing price from the previous day can be used to predict today's price.

Lagged features are created by shifting the time series data by one or more time steps backward. For example, if we are trying to predict today's gold price, we might use the gold price from the previous day (lag-1), two days ago (lag-2), or even further back as features.

Why Use Lagged Features?

- **Capture Temporal Dependency:** Time series data often exhibits autocorrelation, meaning past values influence future values. Lagged features help capture this relationship.
- **Enhance Model Input:** Instead of only using the current state of features, lagged values provide historical context for the model to understand patterns.

2. Rolling Statistics (Moving Averages & Volatility)

Moving Average: Averages the values over a sliding window of time (e.g., past 7 days). It smooths out short-term fluctuations and highlights long-term trends.

Volatility: Measures the variation of prices within a window. It's often calculated as the rolling standard deviation or variance.

Why Use Rolling Statistics?

- Models trained with moving averages can understand market trends better, leading to more accurate predictions.
- Volatility features provide context for price changes, helping models anticipate sharp movements.
- Rolling statistics smooth out noise, making it easier for models to learn meaningful patterns.

3. Normalization/Scaling

The features (independent variables) that are used to predict the target (gold prices) can have different scales (e.g., oil prices in USD, S&P 500 index values, volatility measures, etc.).

Normalizing these features ensures that:

- All the features contribute equally to the model, preventing any single feature (like oil prices) from dominating due to its larger scale.
- It stabilizes the training process and reduces the risk of model bias towards large-scale features.

The value of the gold price is on its own scale and needs to be predicted in this same scale (e.g., dollars per ounce).

Train Test Split

We have used a train-test split that is not **random**; it preserves the temporal order of the data. The data is split **sequentially** rather than randomly. The training set contains earlier data points, while the test set contains later data points in an 80:20 ratio.

Why Temporal Order Matters:

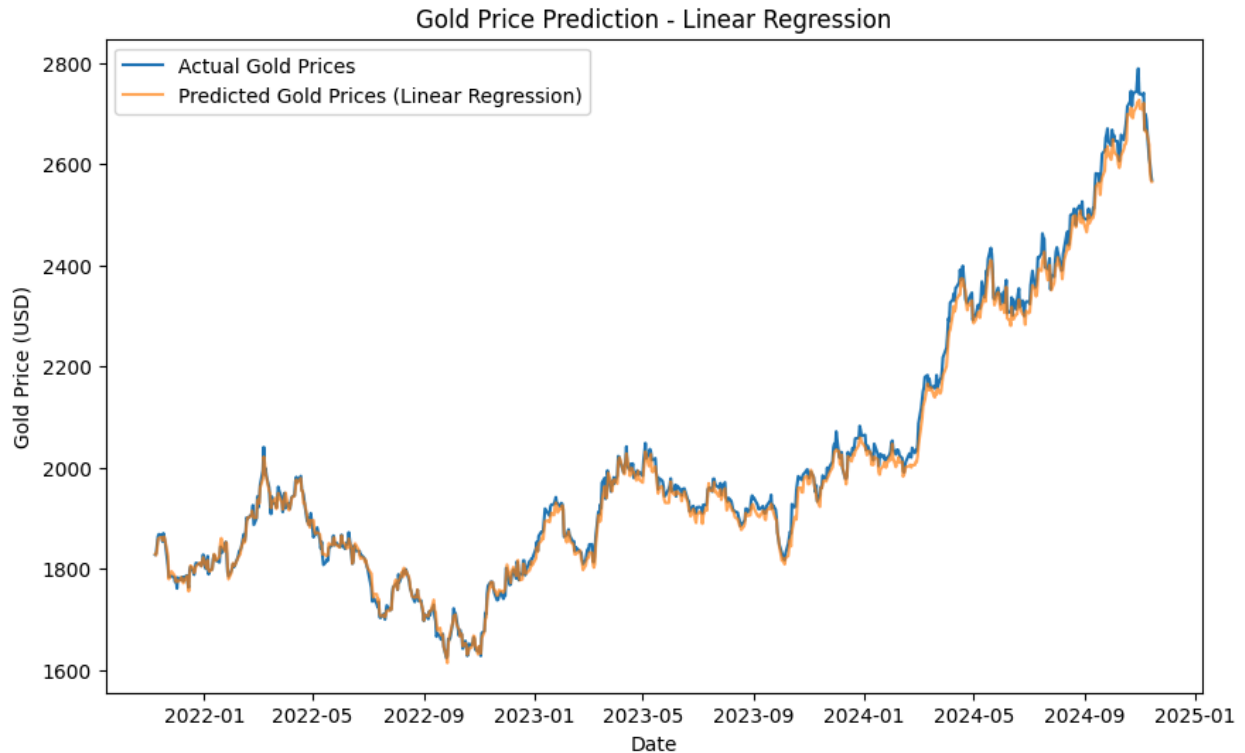
- In time series data, future values depend on past values, so randomizing the order would break the temporal relationships, making the model invalid for forecasting or sequence-based predictions.
- By maintaining temporal order, the model is trained on past data and tested on unseen future data, mimicking real-world scenarios where future data is unknown.

Model Training

1. Linear Regression

Linear regression is the simplest model to establish a baseline for prediction. It assumes a linear relationship between features and the target variable, which is often a reasonable starting assumption for financial data. It is computationally inexpensive, which is important when working with large datasets.

It works well when the relationship between features and the target (gold price) is approximately linear.



Linear Regression - R^2 : 0.996036151030439, Accuracy: 99.60%

Observation:

Linear regression performs surprisingly well for the given dataset, capturing trends and showing strong alignment with the actual gold prices. The predictions align well with the data early on but tend to deviate during periods of high volatility.

Reasons:

Linear regression is likely performing well because it works well for simpler, linear relationships. Early periods in the dataset may have consistent, linear patterns that are easier to model. The deviations in later dates might arise from increased volatility, which linear regression cannot handle effectively due to its simplicity.

Improvements:

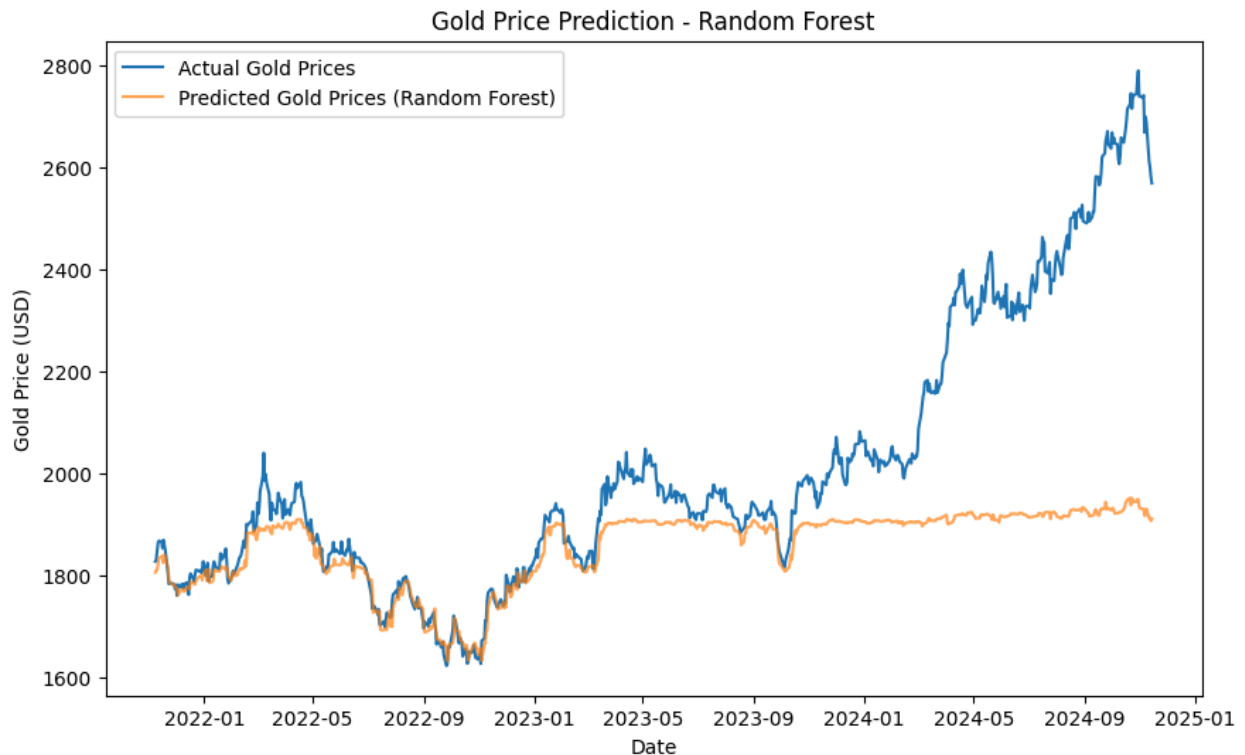
Linear regression's performance is already close to its theoretical best. For further improvement, we could:

- Transform the data using polynomial features to capture non-linear trends.
- Reduce noise in the dataset by smoothing or using feature selection methods.

2. Random Forest

Random Forest can model complex, non-linear interactions between features and the target variable. It is less prone to overfitting due to ensemble averaging and can handle noisy datasets well.

However, Random Forest is not time-series-aware, meaning it cannot inherently consider sequential dependencies in data (important for financial predictions).

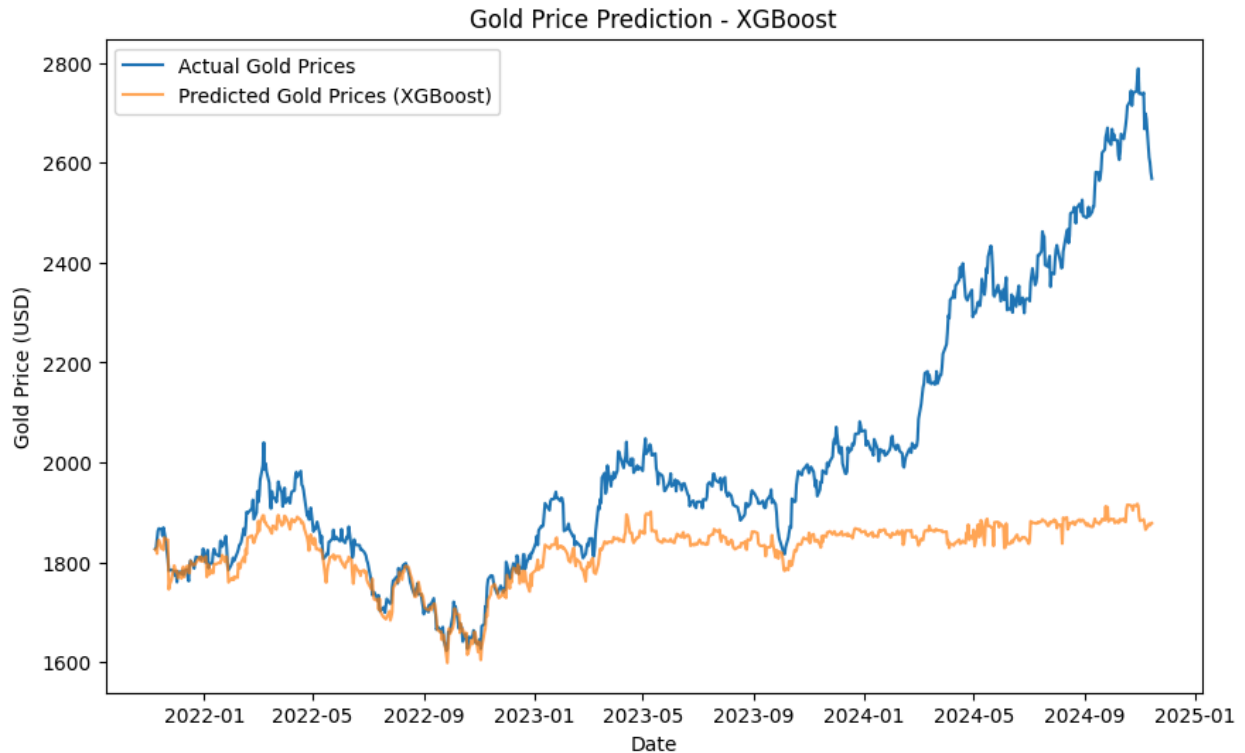


3. XGBoost

XGBoost builds trees sequentially, optimizing for prediction accuracy by minimizing the loss function at each step. This makes it highly effective at capturing complex patterns in the data.

It works well for tabular financial data where the relationship between features and the target is complex and potentially non-linear. When predicting non-sequential targets (e.g., predicting gold price changes without directly modeling time dependencies). (-> however, this is not the case in which we are predicting)

Like Random Forest, it is not inherently time-series-aware, so lag features or rolling statistics must be engineered to capture temporal dependencies.



Observation:

XGBoost fails to capture the broader trend and fluctuates around a lower range, significantly underperforming compared to the other models. It predicts relatively constant values over time, indicating it has not learned the relationship between features and the target effectively.

Reasons:

- Hyperparameter tuning: Default parameters might not be suitable for this dataset, leading to underfitting.
- Feature selection: The model may be overwhelmed by the high dimensionality (120 features) and unable to effectively distinguish signal from noise.

Improvements:

The following improvements could be made:

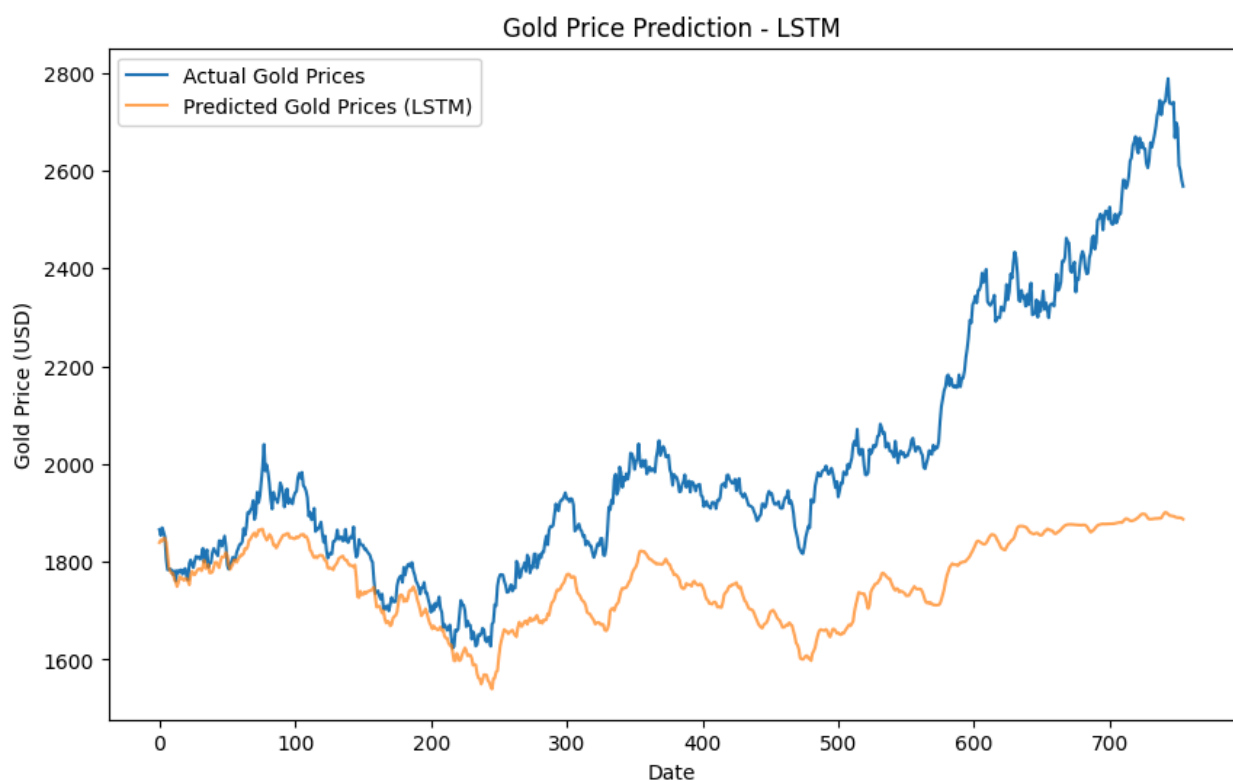
- Perform hyperparameter tuning using techniques like grid search or random search to optimize `max_depth`, `learning_rate`, `n_estimators`, and `subsample`.
- Reduce dimensionality by removing irrelevant features or using PCA.
- Experiment with different objective functions like `reg:squarederror` or `reg:gamma` for better handling of volatility.

4. Long Short Term Memory (LSTM)

LSTMs are explicitly designed to handle sequential data and capture temporal dependencies. This is critical for financial data, as past gold price trends strongly influence future prices. They can retain information from long time spans, making them ideal for modeling financial trends that depend on historical patterns.

It works well for financial time-series data where sequential dependencies (e.g., past prices, macroeconomic indicators over time) are crucial.

However, LSTMs require significant computational resources and can be slow to train. They are prone to overfitting on small datasets unless regularization techniques like dropout are applied.



Observation:

The LSTM's predictions (orange line) show a general upward trend but fail to capture the sharp fluctuations and finer details of the actual gold price (blue line). The predictions consistently lag the actual price movements and show a smoother, less dynamic curve.

Reasons:

- Limited training iterations: Training stopped at 500 iterations due to memory constraints, which might not have been enough for the model to converge fully.
- Data preparation: The fixed window size (e.g., 5 days) might not provide enough context for the LSTM to understand long-term trends.

Improvements:

- Train for more epochs if computational resources allow.
- Using a larger input window size (e.g., 30 or 60 days) to provide more historical context for the model.
- Introduce regularization techniques like dropout or L2 weight decay to prevent overfitting.
- Can consider reducing the number of features to focus on the most relevant ones.

Project Flow

1. Data Acquisition

- API Integration: A Python script retrieves data from the yahoo finance API, which generates data upon request.
- Data Preparation: The retrieved data is processed to create a format suitable for Kafka.

2. Data Streaming to Kafka

- Kafka Producer Setup: A Kafka producer is created using the `KafkaProducer` class from the `kafka` library.
- Data Sending: The prepared data is sent to a Kafka topic named `financial_dataset`

3. Data Processing with Apache Spark

- Structured Streaming: Spark consumes the data from the Kafka topic, allowing for real-time processing and analytics on the incoming data stream.

4. Containerization with Docker

- Docker Compose Configuration: A `docker-compose.yml` file defines all necessary services (elasticsearch, kibana) and their configurations.
- Service Management: All components run as Docker containers, facilitating easy deployment and management of dependencies.

6. Visualization with Kibana

- Data Indexing: The processed data can be indexed in Elasticsearch for visualization.
- Dashboard Creation: Kibana is used to create interactive dashboards that visualize the incoming user data, providing insights into trends and patterns.

7. Management with Elastic Vue

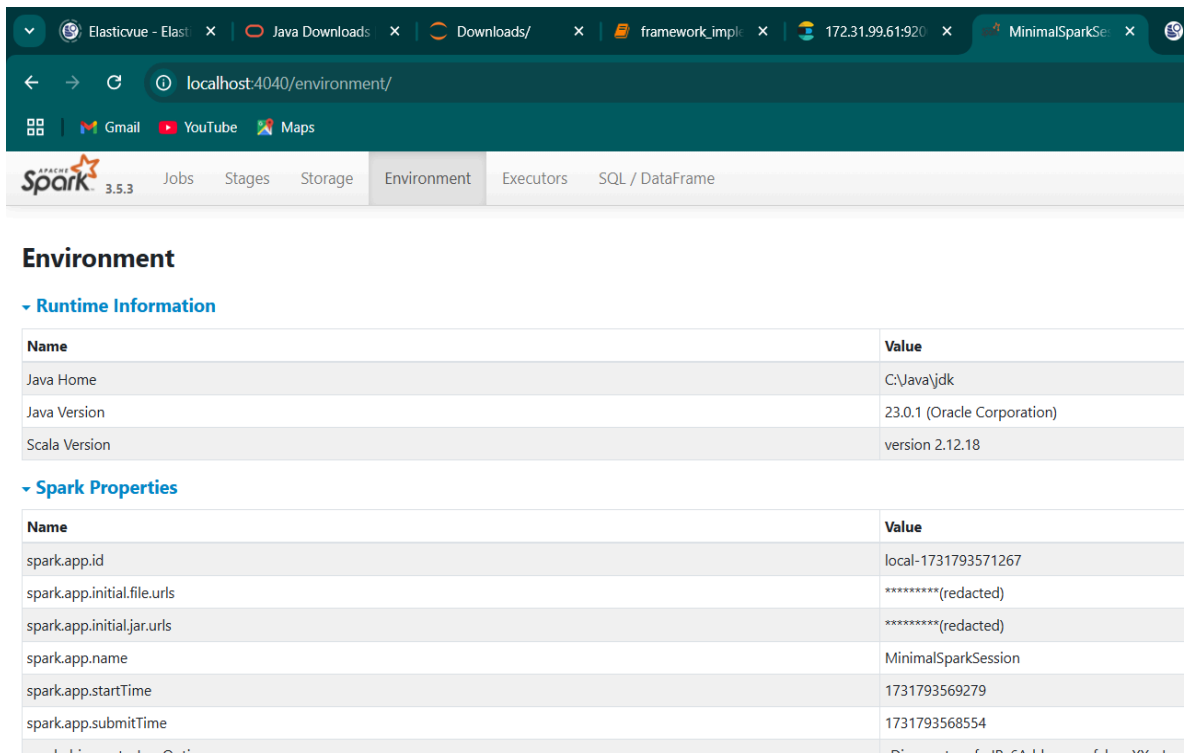
- Cluster Monitoring: Elasticvue is utilized to monitor the health of the Elasticsearch cluster and manage indices.

- **Data Queries:** Users can execute queries directly within Elastic View to explore and manage the indexed data.

Conclusion

This project serves as an illustrative example of building a robust data pipeline that integrates various technologies for real-time data processing. By following this structured approach, users can effectively manage streaming data workflows and leverage modern data engineering practices.

Spark interface:

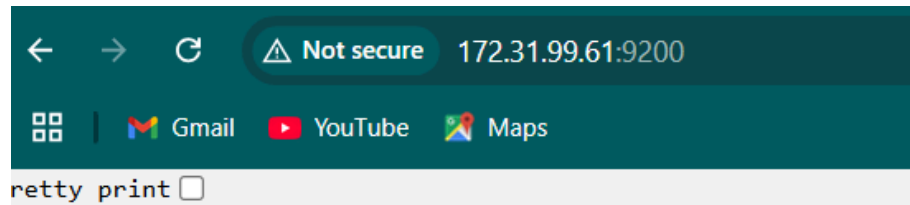


The screenshot shows the Elastic View Spark interface. The browser address bar displays 'localhost:4040/environment/'. The interface includes a navigation bar with tabs for Jobs, Stages, Storage, Environment (selected), Executors, and SQL / DataFrame. Below the navigation bar, the 'Environment' section is expanded, showing 'Runtime Information' and 'Spark Properties'.

Name	Value
Java Home	C:\Java\jdk
Java Version	23.0.1 (Oracle Corporation)
Scala Version	version 2.12.18

Name	Value
spark.app.id	local-1731793571267
spark.app.initial.file.urls	***** (redacted)
spark.app.initial.jar.urls	***** (redacted)
spark.app.name	MinimalSparkSession
spark.app.startTime	1731793569279
spark.app.submitTime	1731793568554
spark.driver.extraJavaOptions	-Djava.library.path=C:\Program Files\Java\jdk-23.0.1\bin

Elasticsearch port:



```
"name" : "es",
"cluster_name" : "es-docker-single",
"cluster_uuid" : "3xZkNIp9SqIPcUFF9VBCCQ",
"version" : {
  "number" : "7.9.0",
  "build_flavor" : "default",
  "build_type" : "docker",
  "build_hash" : "a479a2a7fce0389512d6a9361301708b92dff667",
  "build_date" : "2020-08-11T21:36:48.204330Z",
  "build_snapshot" : false,
  "lucene_version" : "8.6.0",
  "minimum_wire_compatibility_version" : "6.8.0",
  "minimum_index_compatibility_version" : "6.0.0-beta1"
},
"tagline" : "You Know, for Search"
```

GUI used for elasticsearch : Elasticvue

Health is green

Data is divided into 7 shards and all the replicas are assigned to have fault tolerance.

The screenshot shows the Elasticvue interface for a cluster named 'es-docker-single'. The top navigation bar includes links for HOME, NODES, SHARDS, INDICES, SEARCH, REST, and SNAPSHOTS. The main dashboard displays four key metrics: 1 node (1 master, 1 data), 7 shards (7 primaries, 0 replicas), 7 indices (3948 docs, 12.7 MB on disk), and a green health status. Below these metrics, there are two sections: 'Cluster Information' and 'Cluster Health'. The 'Cluster Information' section lists details such as Node Name (es), Cluster name (es-docker-single), Cluster uuid (3xZkNIp9SqIPcUFF9VBCCQ), Tagline (You Know, for Search), Version (7.9.0), Build flavor (default), Build type (docker), Build hash (a479a2a7fce0389512d6a9361301708b92dff667), and Build date (2020-08-11T21:36:48.204330Z). The 'Cluster Health' section shows the Status as green, Timed out as false, Relocating shards as 0, Initializing shards as 0, Unassigned shards as 0, Delayed unassigned shards as 0, Active shards as 7, Active shards percent as 100.00%, Pending tasks as 0, and In flight fetch as 0.

We created **financial_data** index on elasticsearch (as visible on elasticvue below)

matches 1 indices

33ms

CUSTOM SEARCH

<input type="checkbox"/>	_index	_type	_id	_score	Date	GOLD_ETF	SP	DJ	EG	EU	OF
<input type="checkbox"/>	financial_data	_doc	QUUv0ZMB9bTuaDTXNTou	1	2007-08-30	65.80000305175781	1457.6400146484375	13238.73046875	22.92889213562012	1.3642005920410156	71.900
<input type="checkbox"/>	financial_data	_doc	QkUv0ZMB9bTuaDTXNTou	1	2007-08-31	66.5199966430664	1473.98990234375	13357.740234375	23.7905216217041	1.3642005920410156	72.690
<input type="checkbox"/>	financial_data	_doc	REUv0ZMB9bTuaDTXNTou	1	2007-09-04	67.44000244140625	1489.4200439453125	13448.8603515625	24.70001602172852	1.3620080947875977	73.919
<input type="checkbox"/>	financial_data	_doc	RUUv0ZMB9bTuaDTXNTou	1	2007-09-05	67.55999755859375	1472.2900390625	13305.4697265625	24.891489028930664	1.365392804145813	74.339
<input type="checkbox"/>	financial_data	_doc	RkUv0ZMB9bTuaDTXNTou	1	2007-09-06	68.86000061035156	1478.550048828125	13363.349609375	25.848854064941406	1.3696004152297974	74.769
<input type="checkbox"/>	financial_data	_doc	R0Uv0ZMB9bTuaDTXNTou	1	2007-09-07	69.38999938964844	1453.550048828125	13113.3798828125	25.609514236450195	1.3775053024291992	75.069
<input type="checkbox"/>	financial_data	_doc	SUUv0ZMB9bTuaDTXNTou	1	2007-09-11	70.5199966430664	1471.489990234375	13308.3896484375	27.18916702270508	1.384006381034851	76.379
<input type="checkbox"/>	financial_data	_doc	SkUv0ZMB9bTuaDTXNTou	1	2007-09-12	70.45999908447266	1471.56005859375	13291.650390625	26.80622291564941	1.390801191329956	77.680

Indices

NEW INDEX [Index templates](#)

<input type="checkbox"/>	Name	Health	Status	UUID	Aliases	Shards	Segments	Docs	Storage	Created
<input type="checkbox"/>	financial_data	green	open	QPeUTdmVTiqX6cUT4QX7EQ	[]	1p 0r	1	3805	1.26 MB	17/11/2024,

Kibana visualization :

Our dashboard on kibana



1. Graph between max gold etf and date



2. Graph between average of DJ ,SP with date

