PREDICTIVE ANALYSIS FOR RETAIL DEMAND FORECASTING

By Swathi N, Shinjini Pal & Naveen

PROBLEM STATEMENT: RETAIL DEMAND FORECASTING

- Retail businesses face challenges in managing inventory efficiently. Overstocking leads to increased holding costs, while understocking results in lost sales and customer dissatisfaction.
- This project aims to:
  - **Predict demand** for retail products using historical sales data.
  - **Optimize inventory levels** to balance supply and demand.
  - **Incorporate external factors** (e.g., weather, seasonality, competitor pricing) for better accuracy.
  - **Enhance decision-making** in pricing, promotions, and stock replenishment.
- By leveraging **predictive analytics**, retailers can **minimize waste, maximize profits, and improve customer satisfaction**.

CHALLENGES IN RETAIL DEMAND FORECASTING

- **Data Quality:** Inaccurate or incomplete data hampers forecasting.
- **Seasonality:** Demand changes with seasons and holidays.
- Promotions: Special offers can disrupt forecast accuracy.
- **External Factors:** Economic and environmental changes are unpredictable.
- **Granularity:** Balancing forecast detail with accuracy.
- **Model Complexity:** Selecting and tuning the right forecasting model.
- **Inventory Alignment**: Matching demand with inventory levels

FORECASTING IS THE ART OF SAYING WHAT WILL HAPPEN AND THEN EXPLAINING WHY IT DIDN'T!

RETAIL STORE INVENTORY DATASET

- The dataset contains **73,100 rows and 15 columns** related to retail inventory, sales, and demand forecasting.
- Key columns include:
  - **Date, Store ID, Product ID, Category, Region**
  - **Inventory Level, Units Sold, Units Ordered, Demand Forecast**
  - **Price, Discount, Competitor Pricing**
  - **External Factors:** Weather Condition, Holiday/Promotion, and Seasonality

BASIC STATISTICS

The dataset includes numeric and categorical features related to inventory, sales, pricing, and external factors like weather and seasonality.

Summary Statistics:

	Inventory Level	Units Sold	Units Ordered	Demand Forecast	\
count	73100.000000	73100.000000	73100.000000	73100.000000	
mean	274.469877	136.464870	110.004473	141.494720	
std	129.949514	108.919406	52.277448	109.254076	
min	50.000000	0.000000	20.000000	-9.990000	
25%	162.000000	49.000000	65.000000	53.670000	
50%	273.000000	107.000000	110.000000	113.015000	
75%	387.000000	203.000000	155.000000	208.052500	
max	500.000000	499.000000	200.000000	518.550000	

	Price	Discount	Holiday/Promotion	Competitor Pricing
count	73100.000000	73100.000000	73100.000000	73100.000000
mean	55.135108	10.009508	0.497305	55.146077
std	26.021945	7.083746	0.499996	26.191408
min	10.000000	0.000000	0.000000	5.030000
25%	32.650000	5.000000	0.000000	32.680000
50%	55.050000	10.000000	0.000000	55.010000
75%	77.860000	15.000000	1.000000	77.820000
max	100.000000	20.000000	1.000000	104.940000

Dataset Overview:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 73100 entries, 0 to 73099
Data columns (total 15 columns):
```

#	Column	Non-Null Count	Dtype
0	Date	73100 non-null	object
1	Store ID	73100 non-null	object
2	Product ID	73100 non-null	object
3	Category	73100 non-null	object
4	Region	73100 non-null	object
5	Inventory Level	73100 non-null	int64
6	Units Sold	73100 non-null	int64
7	Units Ordered	73100 non-null	int64
8	Demand Forecast	73100 non-null	float64
9	Price	73100 non-null	float64
10	Discount	73100 non-null	int64
11	Weather Condition	73100 non-null	object
12	Holiday/Promotion	73100 non-null	int64
13	Competitor Pricing	73100 non-null	float64
14	Seasonality	73100 non-null	object

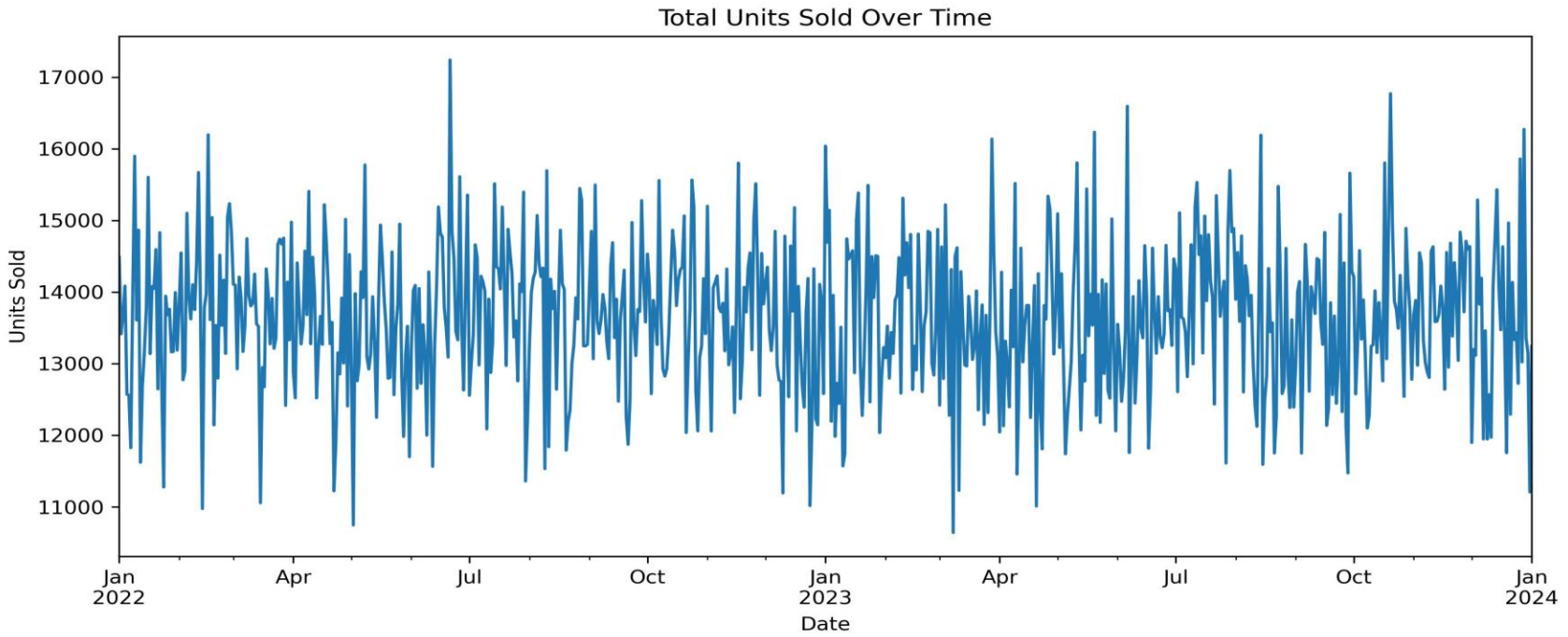
```
dtypes: float64(3), int64(5), object(7)
memory usage: 8.4+ MB
```

Missing Values:

Date	0
Store ID	0
Product ID	0
Category	0
Region	0
Inventory Level	0
Units Sold	0
Units Ordered	0
Demand Forecast	0
Price	0
Discount	0
Weather Condition	0
Holiday/Promotion	0
Competitor Pricing	0
Seasonality	0

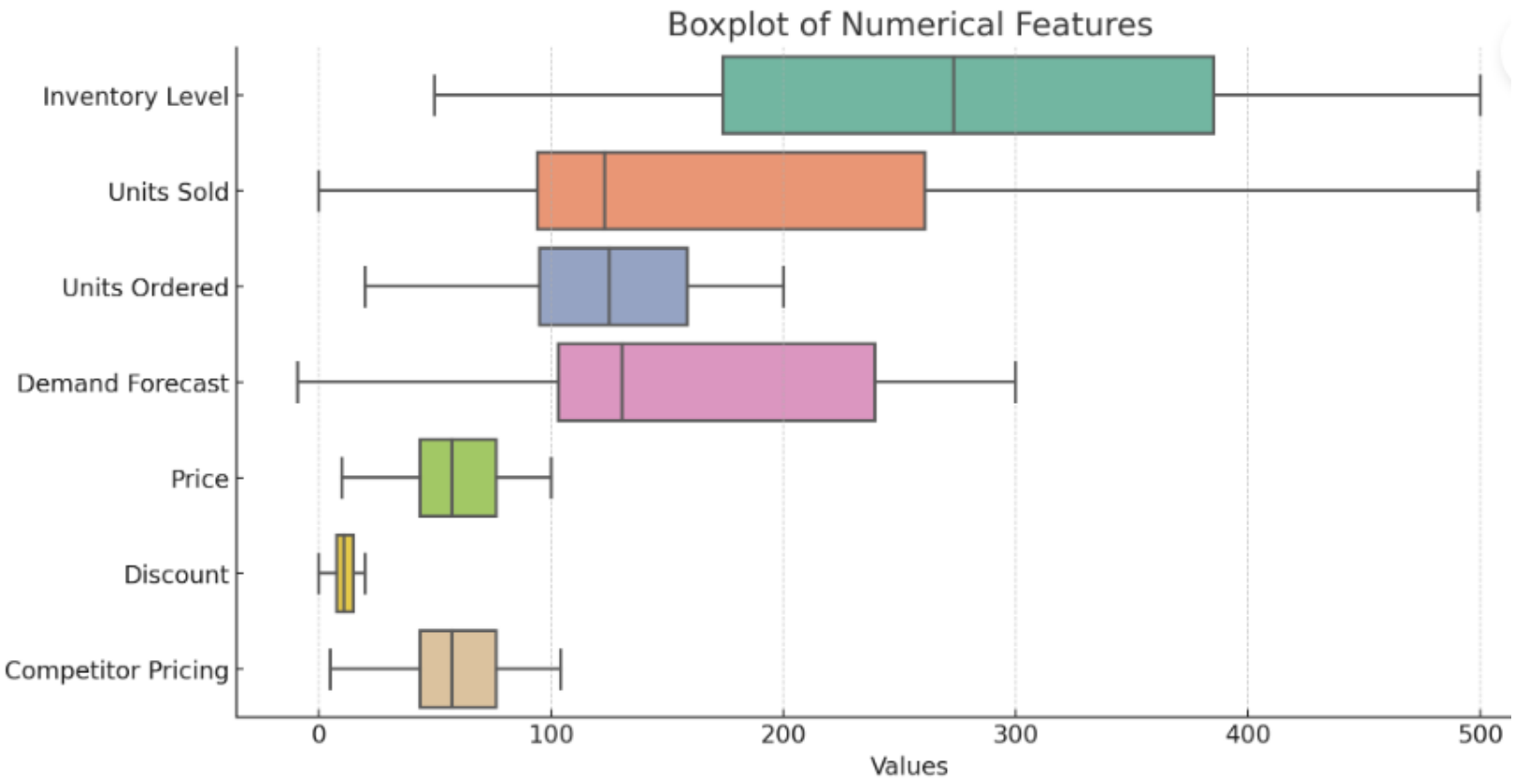
dtype: int64

EDA: EXPLORATORY DATA ANALYSIS



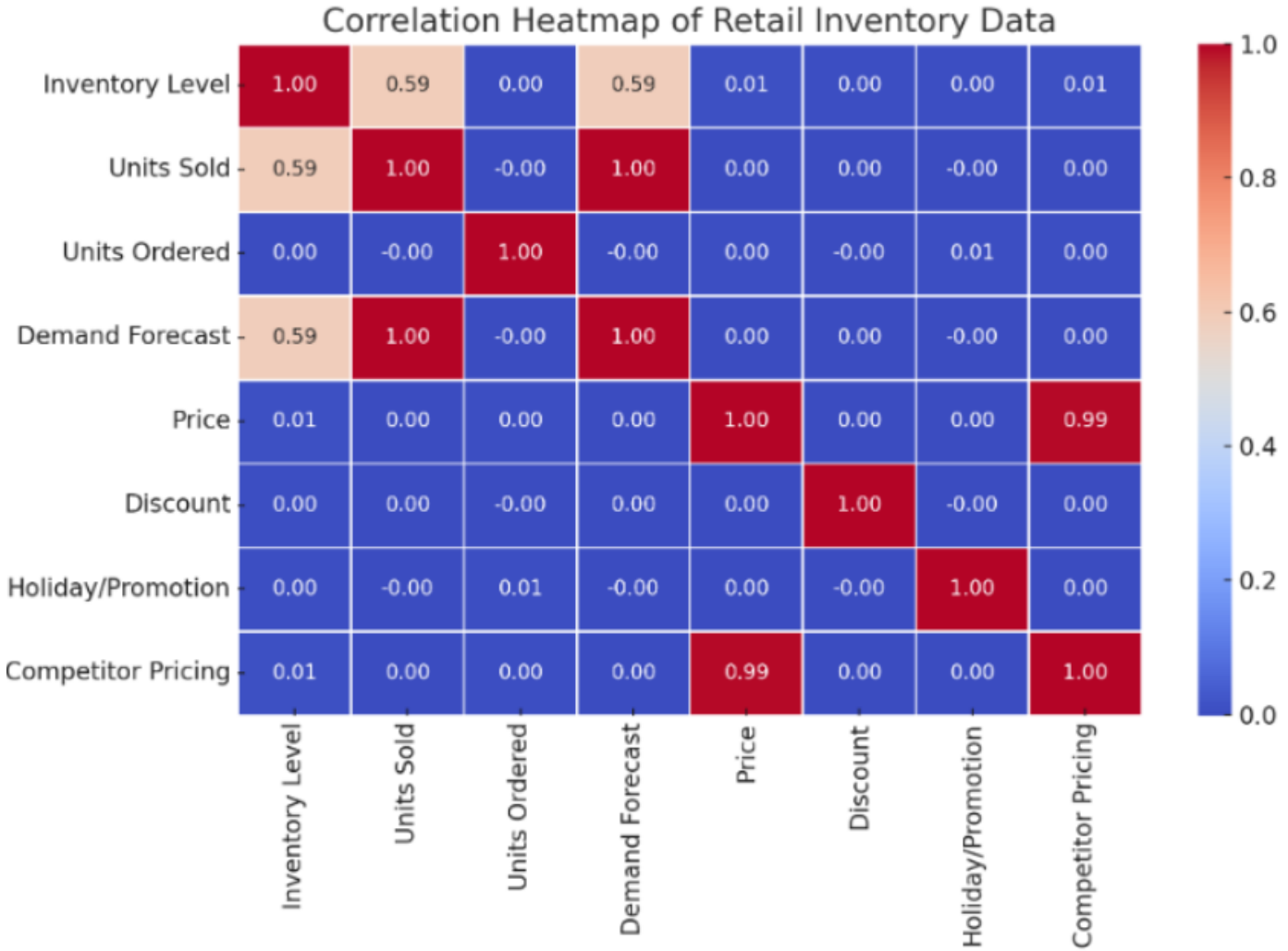
Time Series Line Plot

1. High Variability
2. No Strong Trend
3. Potential Seasonality
4. Sales Range



Box Plot:

- **Sales Fluctuations:** High variability; negative forecasts may indicate data issues.
- **Pricing Control:** Prices range **10-100**, discounts stay within **0-20**.
- **Market Competition:** Competitor prices vary widely, affecting sales.
- **Inventory Stability:** Well-managed supply minimizes stockouts.



HeatMap

- **Correlation Insights** – The heatmap reveals relationships between key retail metrics like inventory levels, demand forecasts, and competitor pricing.
- **Demand & Sales Link** – Strong correlation between **Units Sold** and **Demand Forecast**, confirming model accuracy.
- **Pricing Impact** – **Price** and **Competitor Pricing** show notable influence on sales, affecting demand fluctuations.
- **Promotions & Discounts** – **Holiday/Promotion** and **Discounts** moderately impact **Units Sold**, highlighting seasonal demand shifts.

DATA PRE-PROCESSING

Handling Missing Values: Imputation methods.

Feature Engineering: Creating new features like demand trends.

Normalization & Encoding: Scaling numerical features, one-hot encoding categorical features.

MODEL EVALUATION

Model Selection

- **Why Random Forest?**
  - Handles non-linearity and feature interactions.
  - Robust against overfitting with ensemble learning.
- **Training & Evaluation**
  - **Train/Test Split:** 80/20 ratio.
  - **Performance Metrics:** MAE, RMSE, R^2 Score.

```
# Train model
model = RandomForestRegressor(n_estimators=100, random_state=42)
model.fit(X_train, y_train)
```

```
RandomForestRegressor(random_state=42)
```

```
# Evaluate model
y_pred = model.predict(X_test)
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print(f"MAE: {mae}, MSE: {mse}, R2: {r2}")
```

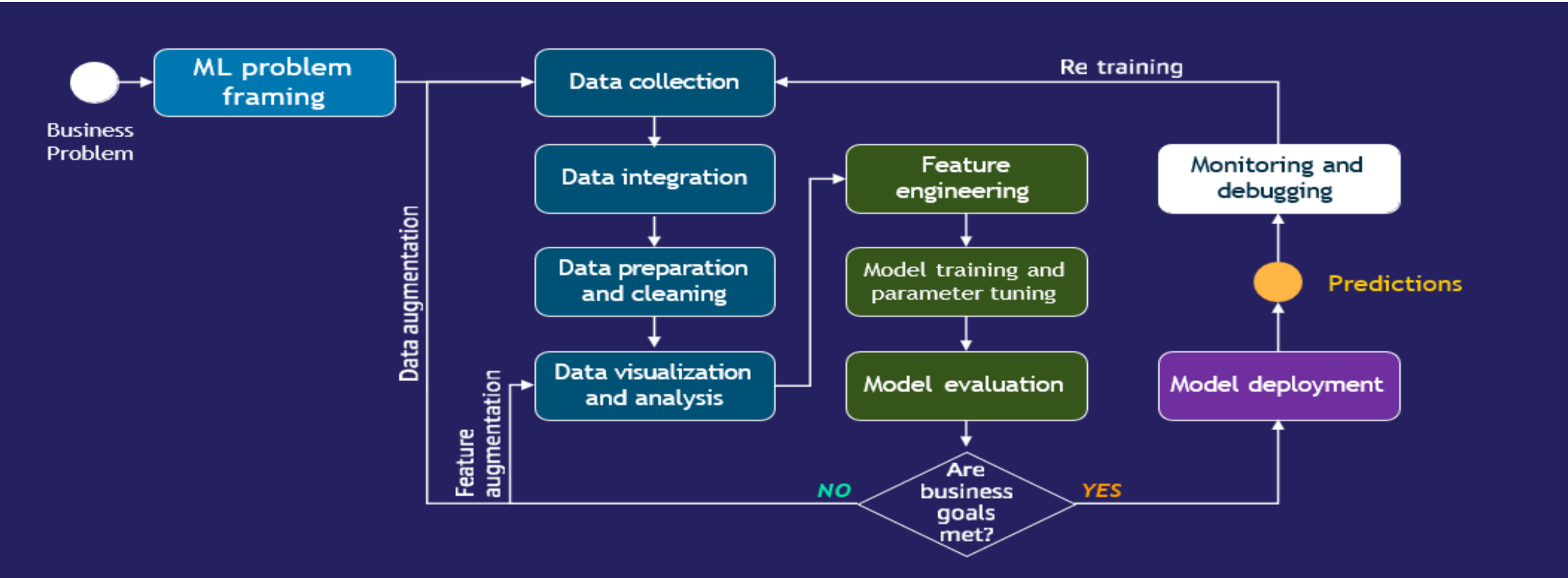
```
MAE: 7.579618194254446, MSE: 78.42278824347744, R2: 0.9934211120068258
```

- **Overall → The model is highly accurate, with minimal error and strong predictive power.**

ML PROCESS

Deployment Strategy

- **Containerization:** Deploying the model with Docker.
- **API Development:** Flask/FastAPI for model inference.
- **CI/CD Pipeline:** Automating model updates.



JENKINS CI/CD WITH TEST STAGE

CI/CD in Jenkins: Automates code integration, testing, and deployment.

Test Stage Importance:

Detects bugs early, prevents faulty deployments, ensures software quality.

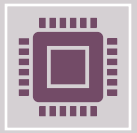
Includes Unit Tests, Integration Tests, Functional Tests.

Benefits: Reduces deployment risks, ensures stable releases, and enhances efficiency.

JENKINS CI/CD WITH TEST STAGE

Jenkins Setup Overview:	Installed Jenkins on Ubuntu Terminal
	JDK 17 required
	Jenkins accessible at localhost:8080
	Update Dockerhub credentials in Jenkins > dashboard > Manage Jenkins > Credentials > Global. [docker pull deswalcool/mlops:project10mlopsnaveenswathishinjini]
	Deploy Kubernetes authentication and add Kubernetes-cd.hpi plugin
	Updates the details of **kubectl config view — flatten** in Jenkins Credentials
	Create Pipeline Script with different stages
	Configure Github URL and Branch specifier in the pipeline option
	Configure the Minikube cluster deployment yaml file in the dockerhub URL in the Jenkins Pipeline stages.
	Test the Deployement.

DEPLOYMENT ON MINIKUBE CLUSTER



What is Minikube?

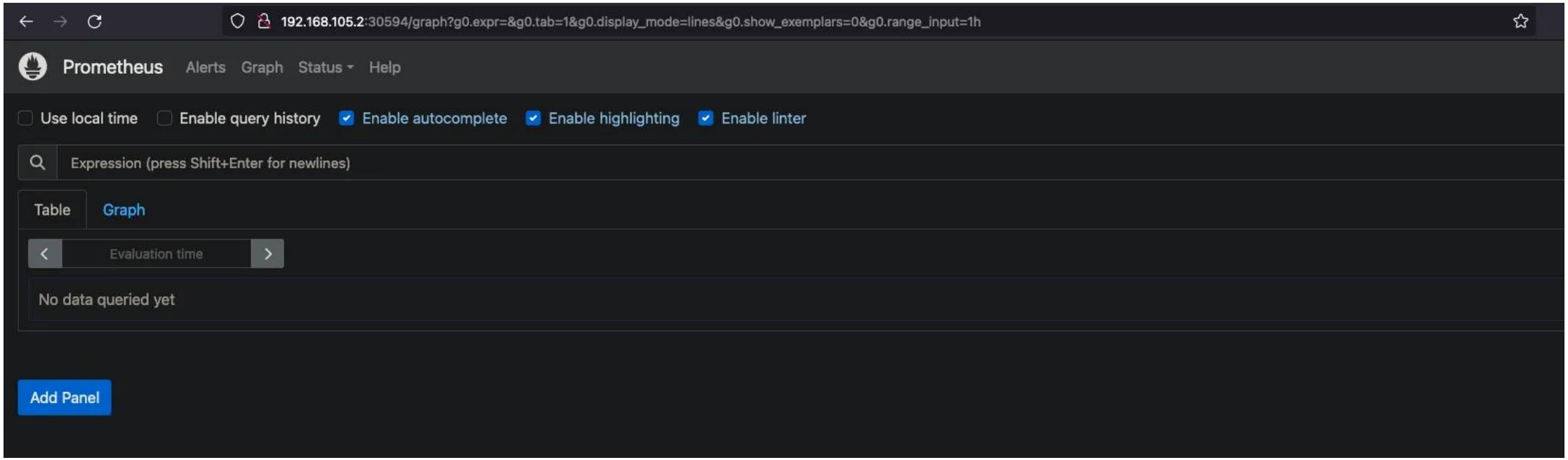
A lightweight Kubernetes tool for local development and testing.
Runs a single-node Kubernetes cluster on a local machine.

Deployment Process:

Start Minikube to initialize the Kubernetes cluster.
Create a Deployment to manage application instances.
Apply Deployment using kubectl to schedule pods.
Expose the Service to make the application accessible.
Access the Application via the Minikube service URL.

Benefits of Minikube:

Simulates a **real Kubernetes environment** for testing.
Supports **containerized application development**.
Works well with **CI/CD pipelines for automation**.



CAPTURING METRICES WITH PROMETHEUS

`http://192.168.105.2:30594` **(Default Port : 9090)**

Prometheus:

- Open-source monitoring and alerting toolkit.
- Collects and stores time-series data (metrics).
- Queries data using PromQL.

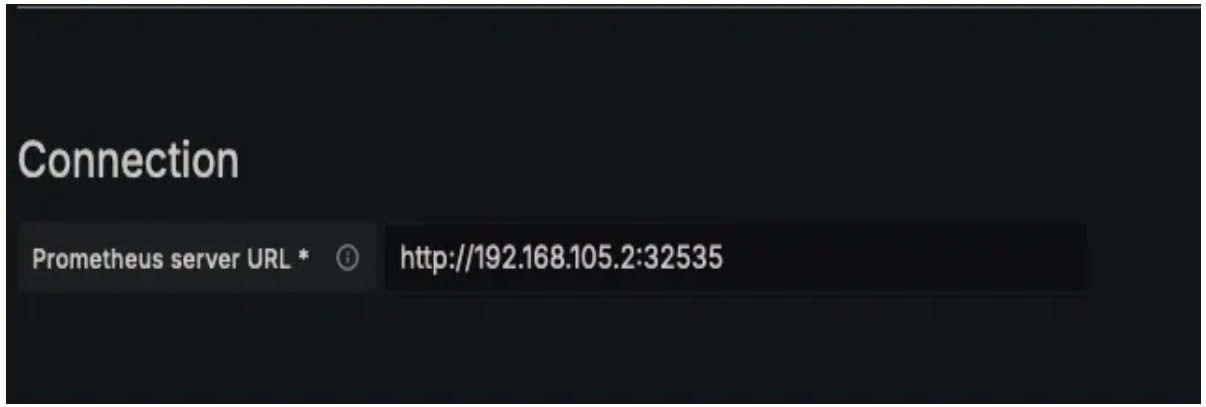
MONITORING WITH GRAFANA

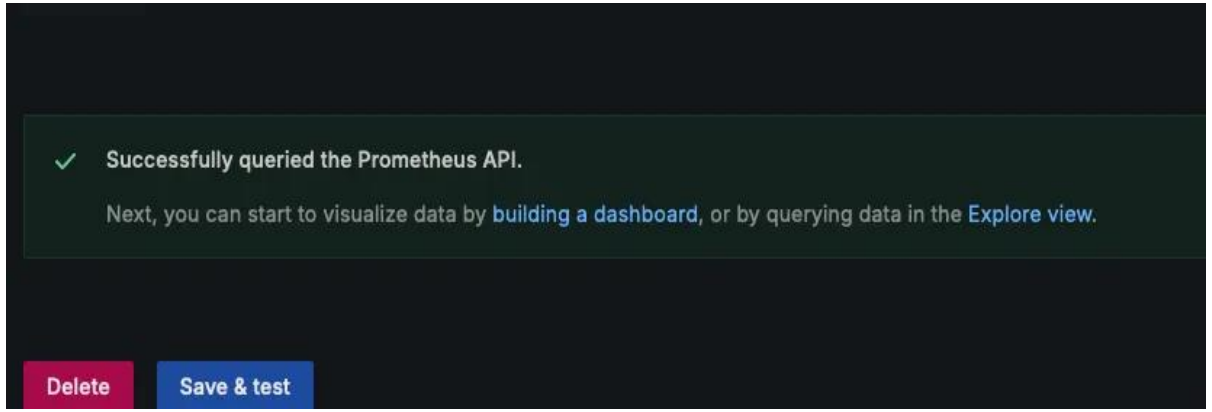
`http://192.168.105.2:31173` (Default Port : 3000)

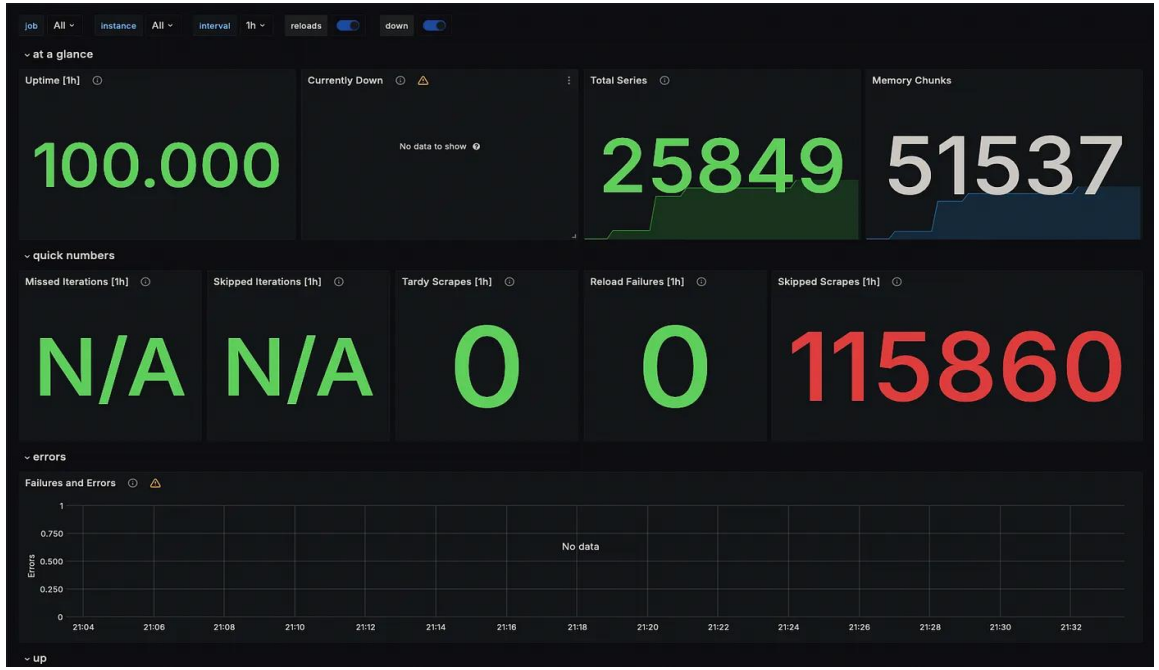
- Displays Prometheus metrics in interactive dashboards.
- Offers customizable visualizations and alert integrations.

Configure Data Source

Navigate to **Configuration > Data Sources, s**elect and configure the data source (e.g., Prometheus).







THANK YOU