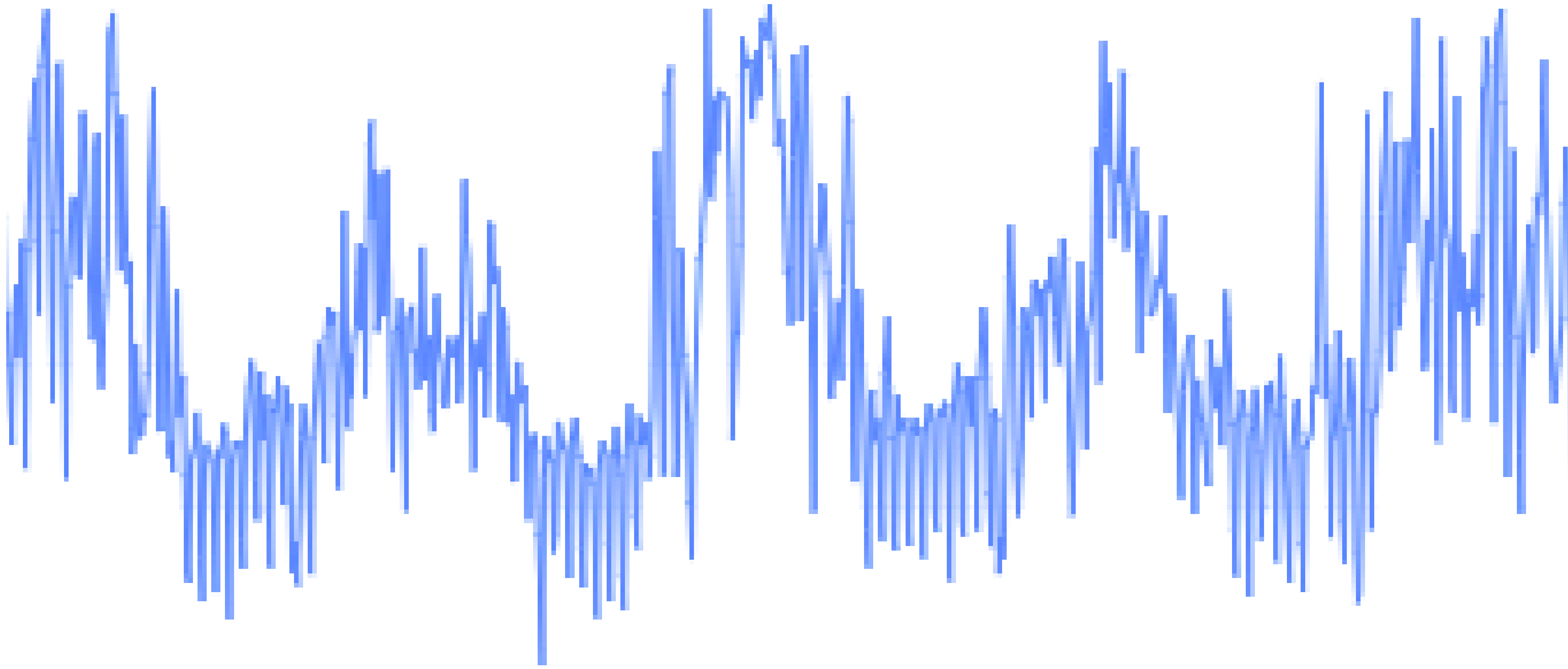


Time Series Forecasting App

AUTHOR

Fani Sentinella-Jerbić

DATE

February 7th, 2026

Agenda

1. TASK OVERVIEW
2. SOLUTION OVERVIEW
3. METHODOLOGICAL APPROACH
4. TECH STACK
5. CODE ARCHITECTURE
6. FUTURE ENHANCEMENTS

Task Overview

MAIN AIM

Develop an application to forecast future consumption based on historical data using statistical models like ARIMA, MA, or Prophet.

FUNCTIONAL REQUIREMENTS

DATA & UI

- ✓ CSV Upload (Date, Consumption format)
- ✓ Visualize historical vs. predicted values
- ✓ User-configurable forecast horizon

MODELING

- Implement time series forecasting model
- Evaluation metrics

EVALUATION CRITERIA

UI Quality
Usability, clarity, and visual appeal

Documentation
Thoroughness and clarity of explanation

Creativity
Problem solving and feature extension

Model Implementation
Correctness and efficiency

Solution Overview

POSITIONING

A general-purpose, educational time series forecasting tool designed to explore across domains, not limited to consumption data.

THREE MODELING PARADIGMS

ARIMA
Classical Stats

Prophet
Business/Trend

N-BEATS
Deep Learning

+

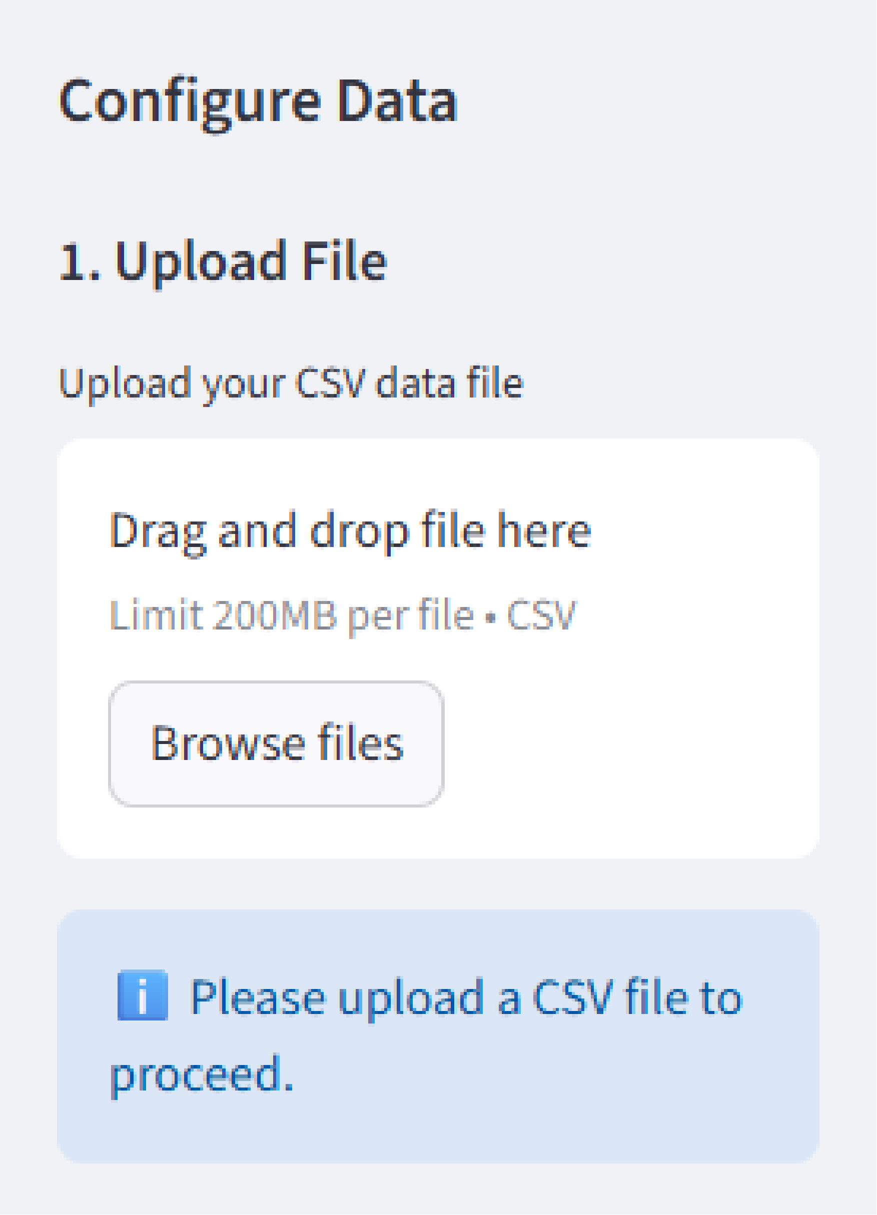
Easy Model Plugin
Any domain

USAGE FLOW

Load → Preprocess → Forecast → Evaluate → Compare

Key features

DATA UPLOAD & PREPROCESSING



Configure Data

1. Upload File

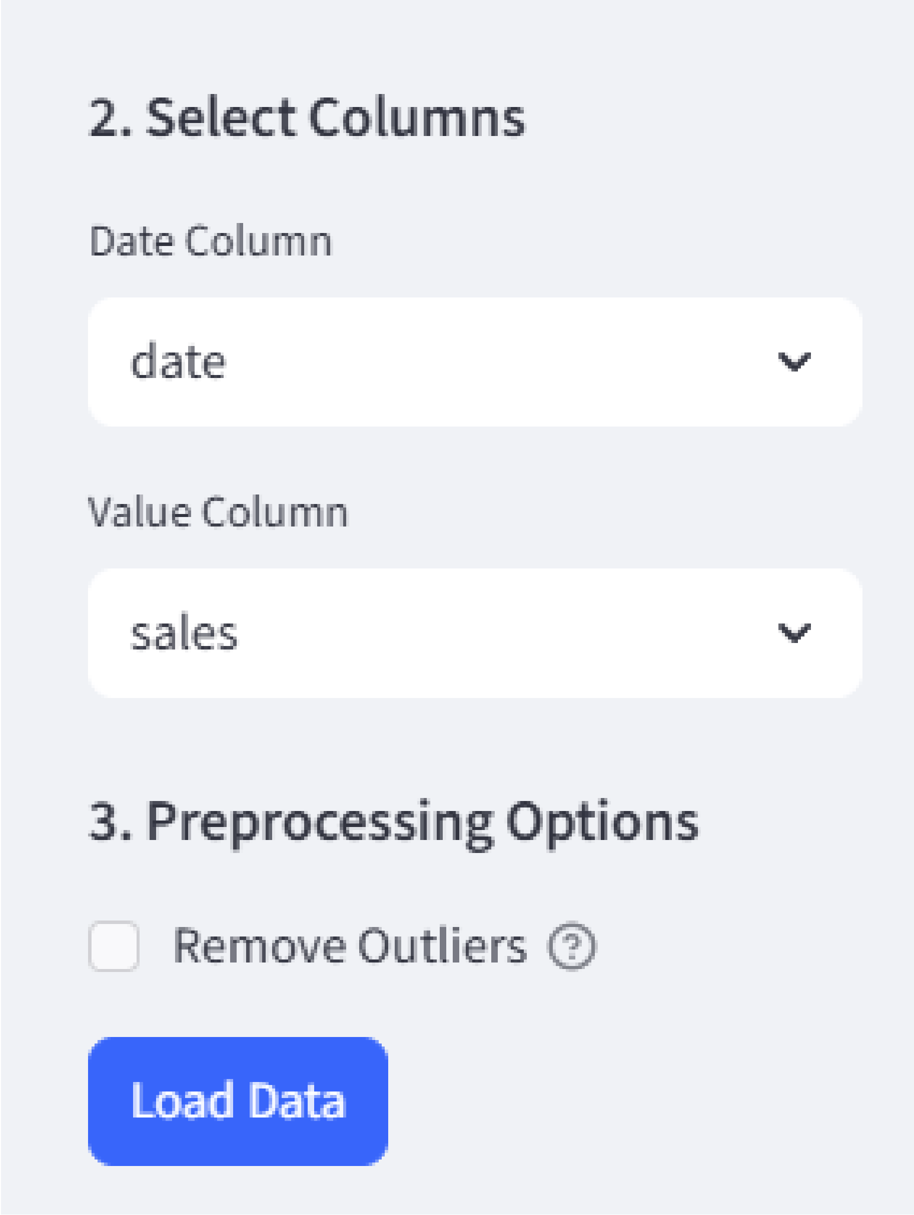
Upload your CSV data file

Drag and drop file here

Limit 200MB per file • CSV

ℹ️ Please upload a CSV file to proceed.

→



2. Select Columns

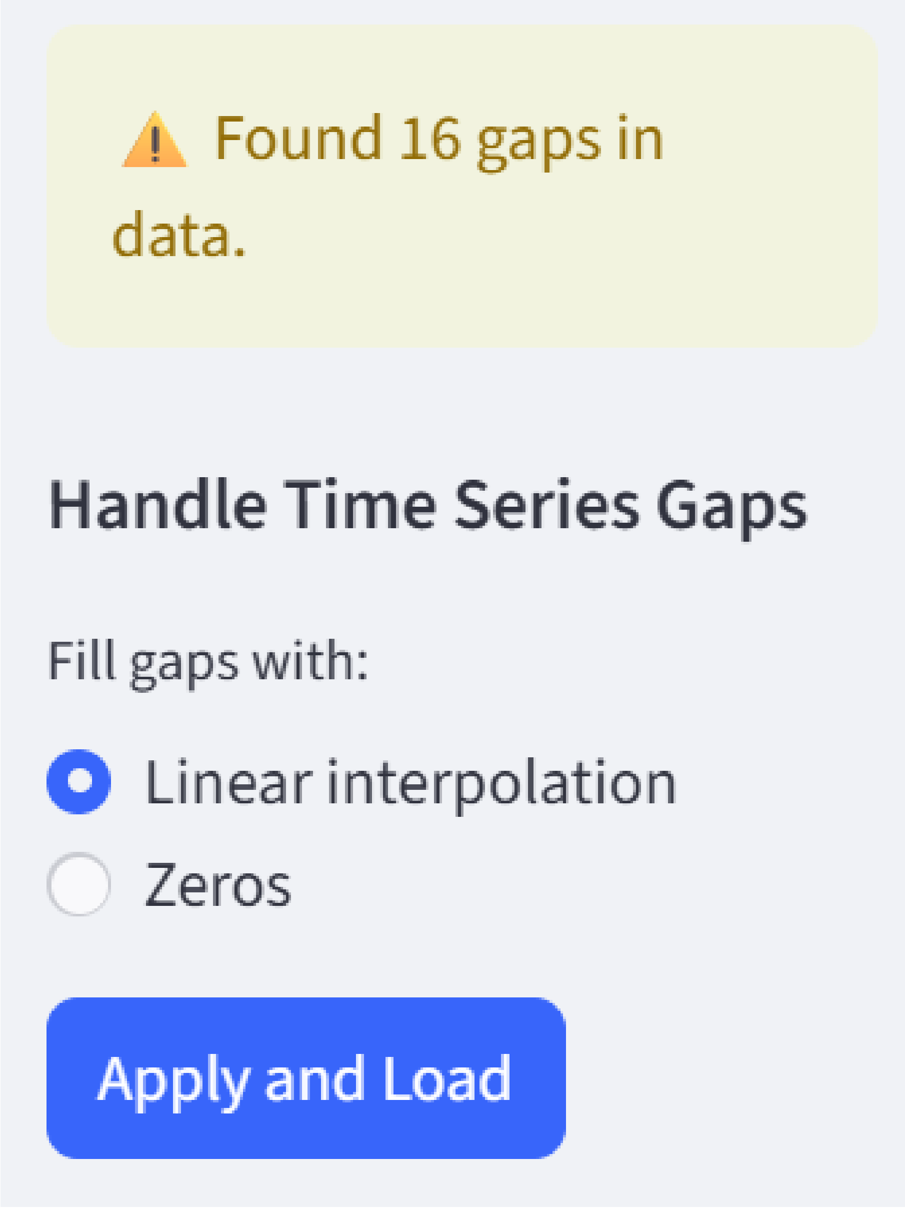
Date Column: date

Value Column: sales

3. Preprocessing Options

□ Remove Outliers

→



⚠️ Found 16 gaps in data.

Handle Time Series Gaps

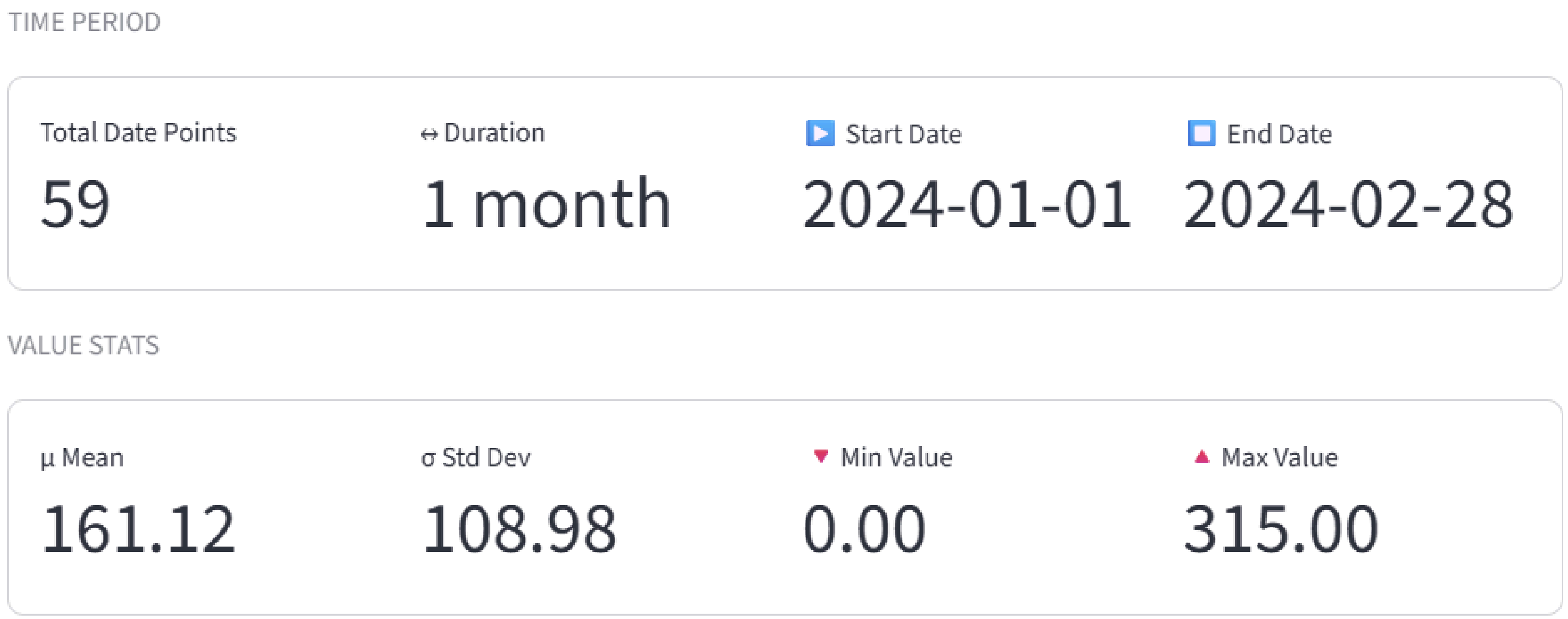
Fill gaps with:

● Linear interpolation

○ Zeros

Key features

HISTORICAL DATA PREVIEW

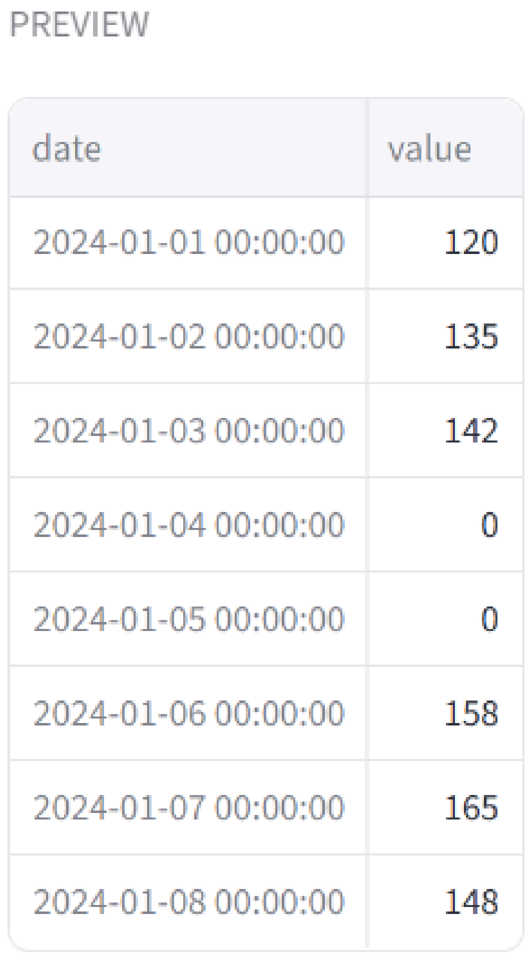


TIME PERIOD

Total Date Points	↔ Duration	▶ Start Date	□ End Date
59	1 month	2024-01-01	2024-02-28

VALUE STATS

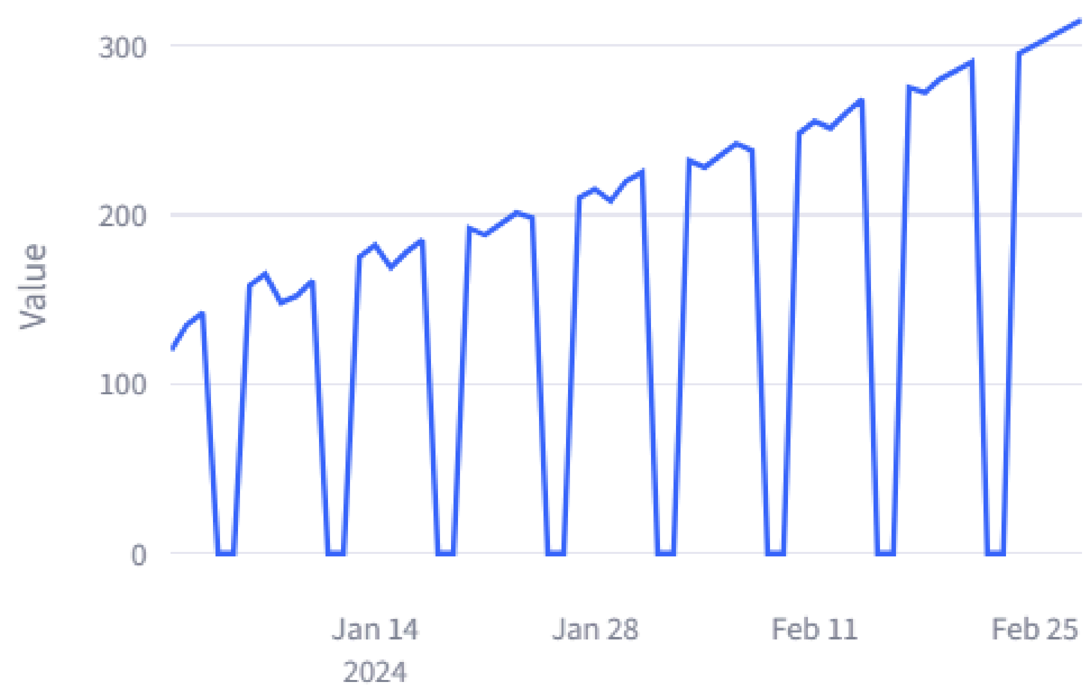
µ Mean	σ Std Dev	▼ Min Value	▲ Max Value
161.12	108.98	0.00	315.00



PREVIEW

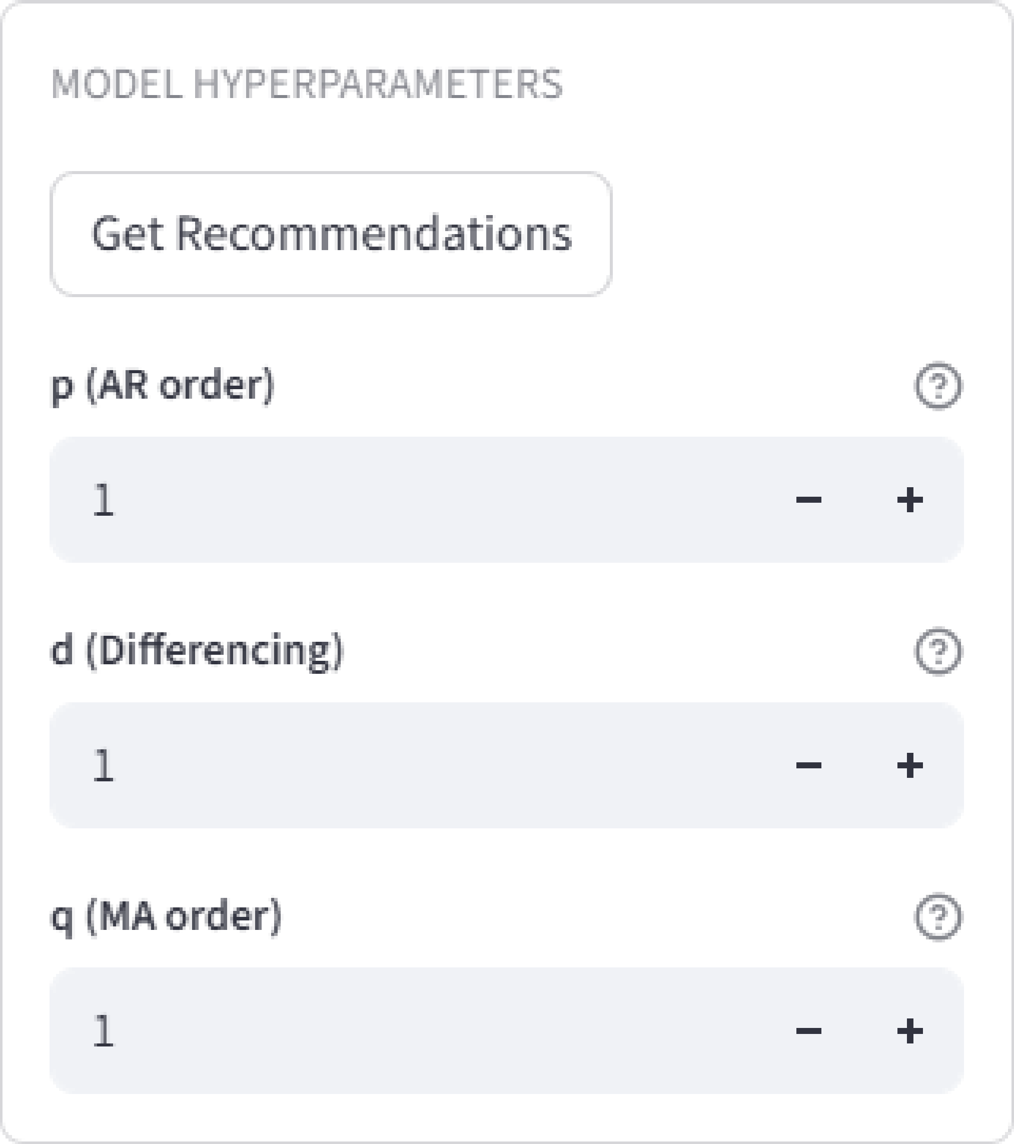
date	value
2024-01-01 00:00:00	120
2024-01-02 00:00:00	135
2024-01-03 00:00:00	142
2024-01-04 00:00:00	0
2024-01-05 00:00:00	0
2024-01-06 00:00:00	158
2024-01-07 00:00:00	165
2024-01-08 00:00:00	148

VISUALIZATION



Key features

MODEL SETUP

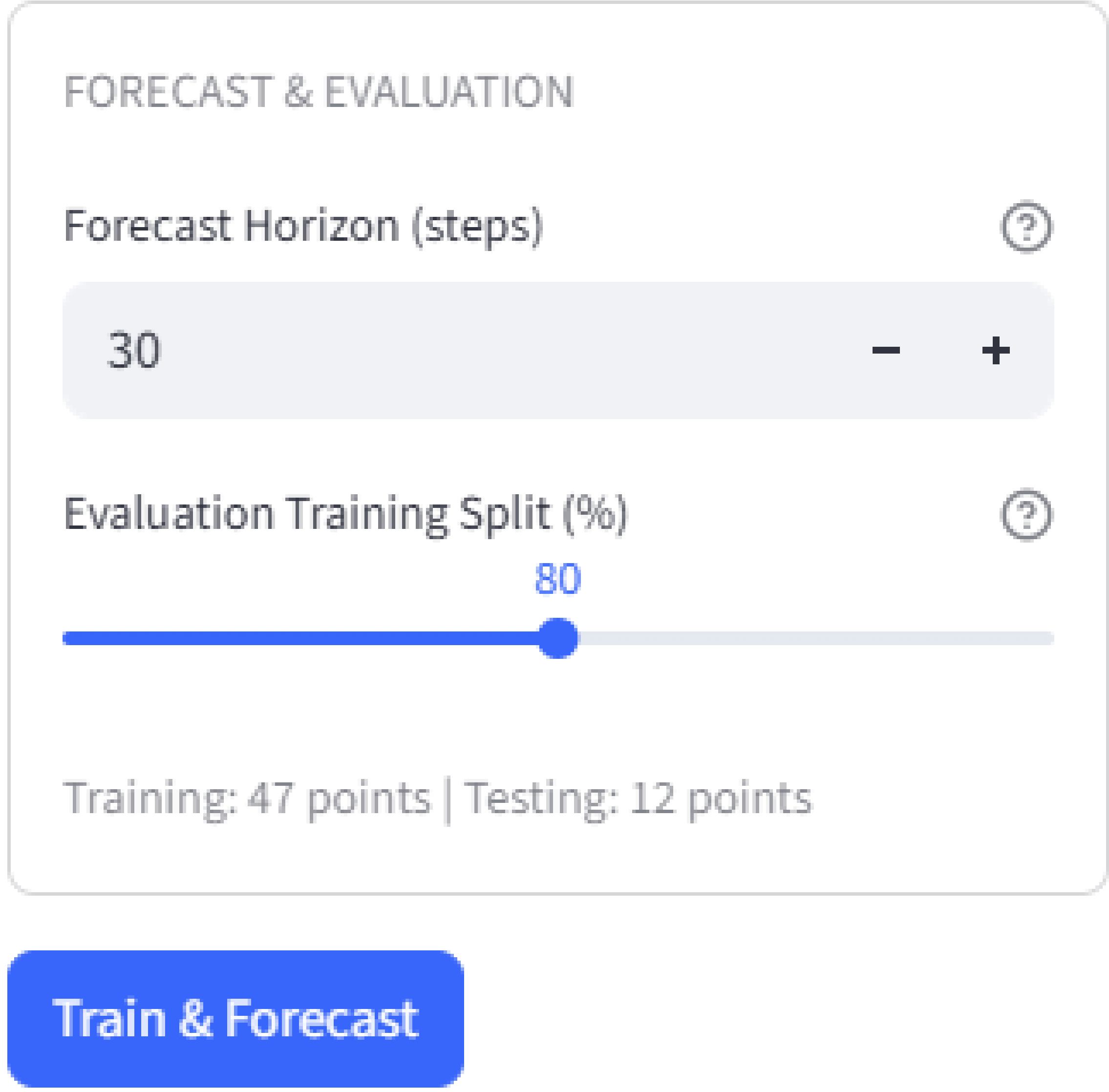


MODEL HYPERPARAMETERS

p (AR order): 1

d (Differencing): 1

q (MA order): 1

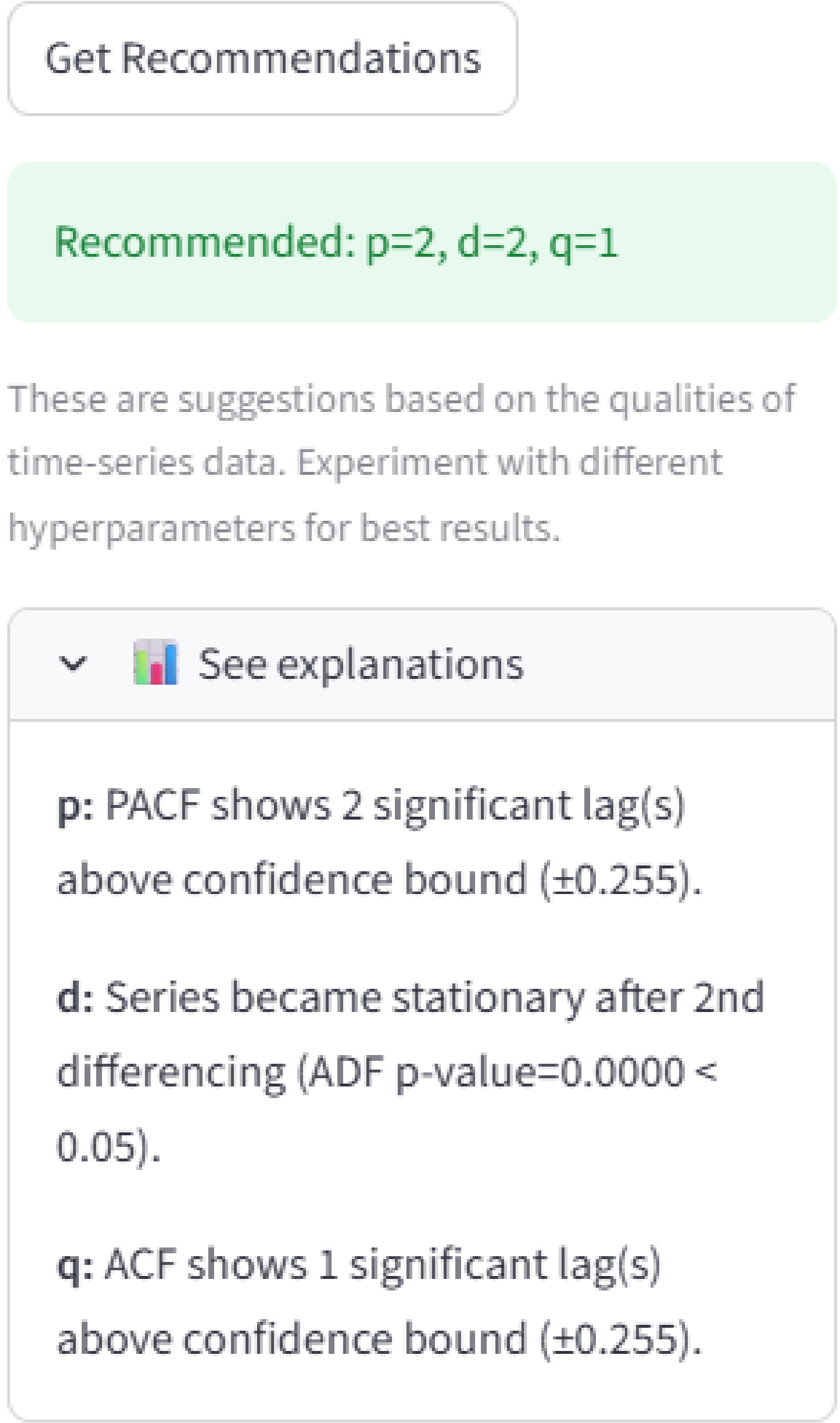


FORECAST & EVALUATION

Forecast Horizon (steps): 30

Evaluation Training Split (%): 80

Training: 47 points | Testing: 12 points



Recommended: p=2, d=2, q=1

These are suggestions based on the qualities of time-series data. Experiment with different hyperparameters for best results.

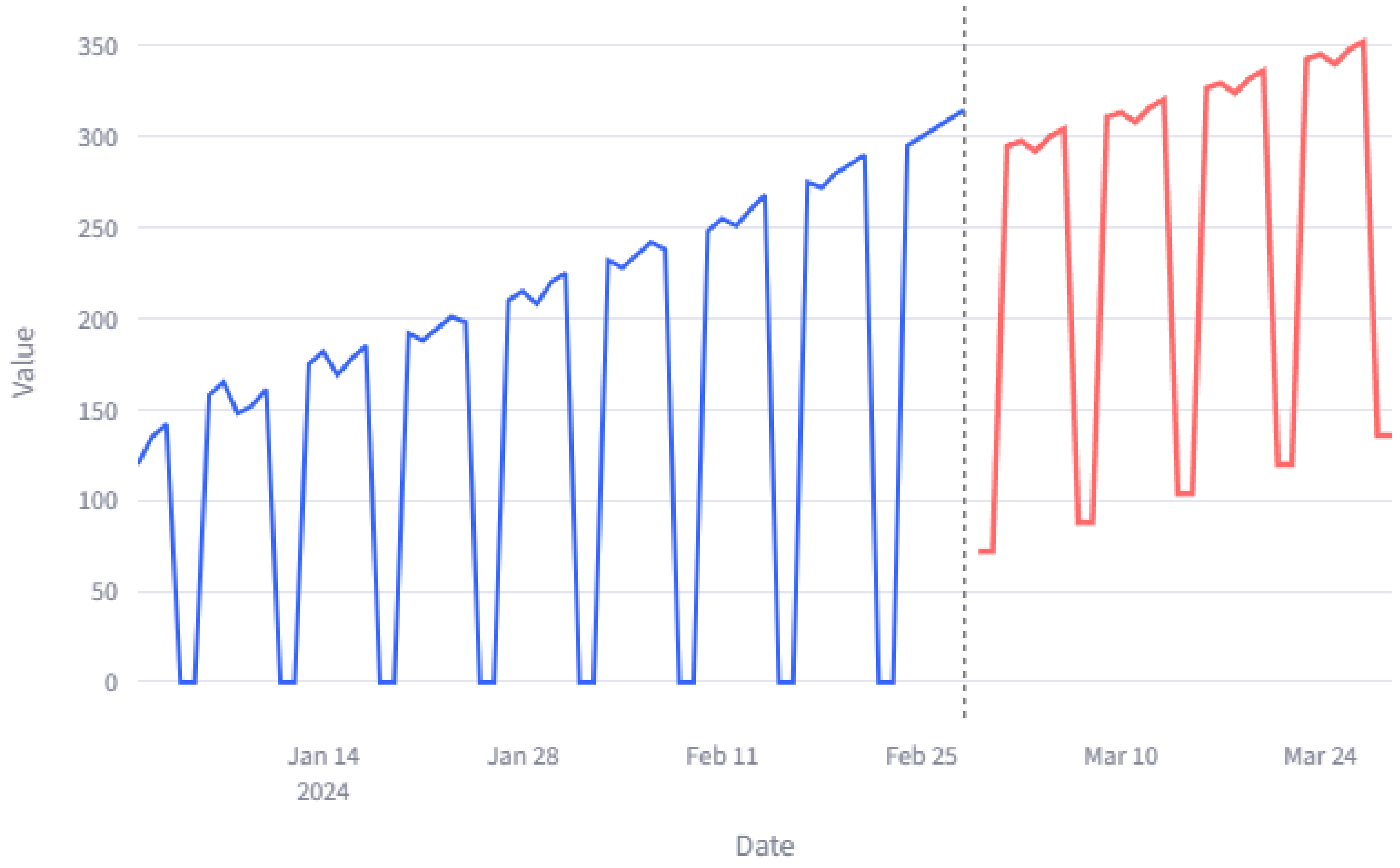
p: PACF shows 2 significant lag(s) above confidence bound (±0.255).

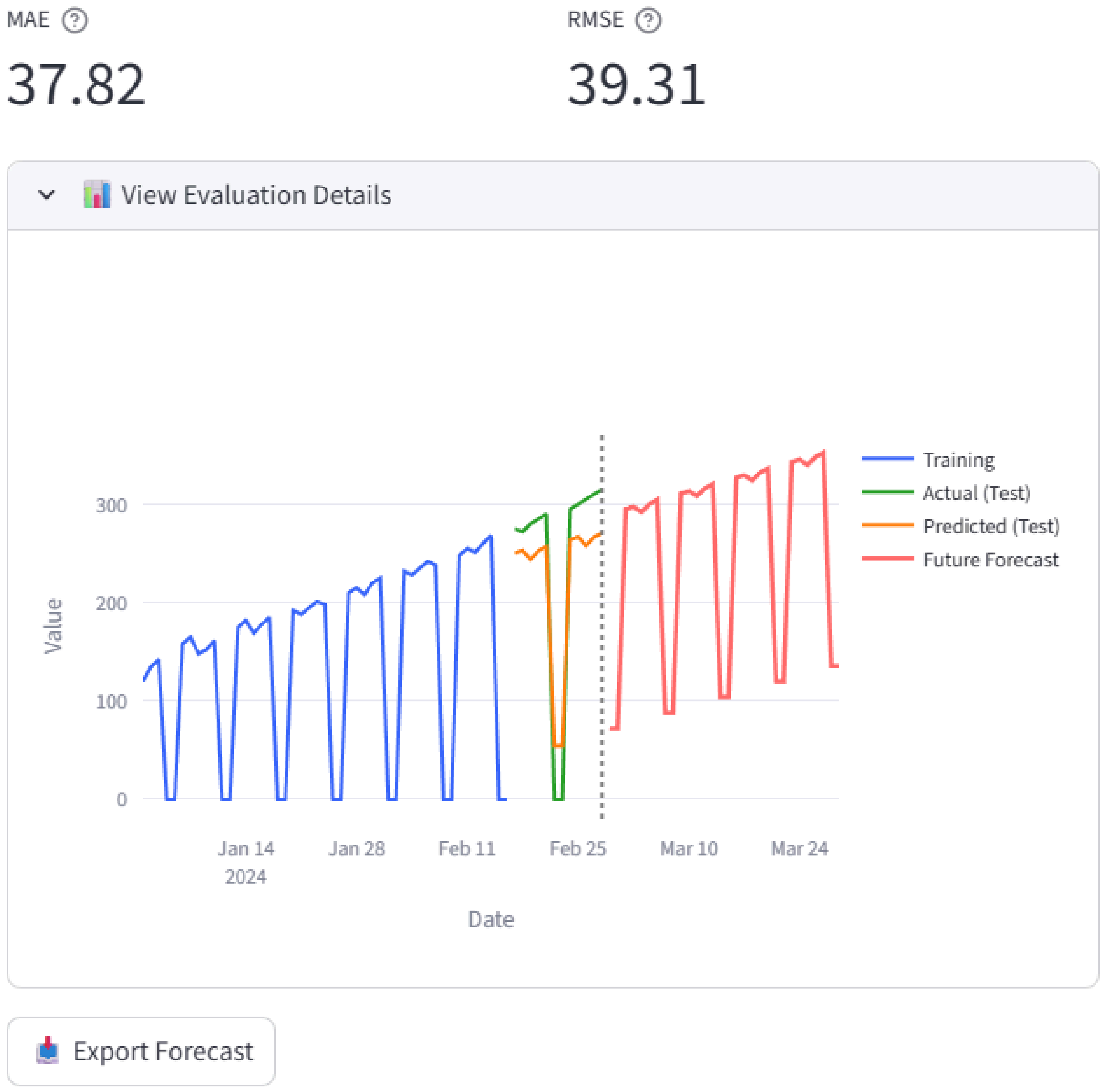
d: Series became stationary after 2nd differencing (ADF p-value=0.0000 < 0.05).

q: ACF shows 1 significant lag(s) above confidence bound (±0.255).

Key features

FORECAST & EVALUATION




MAE ?

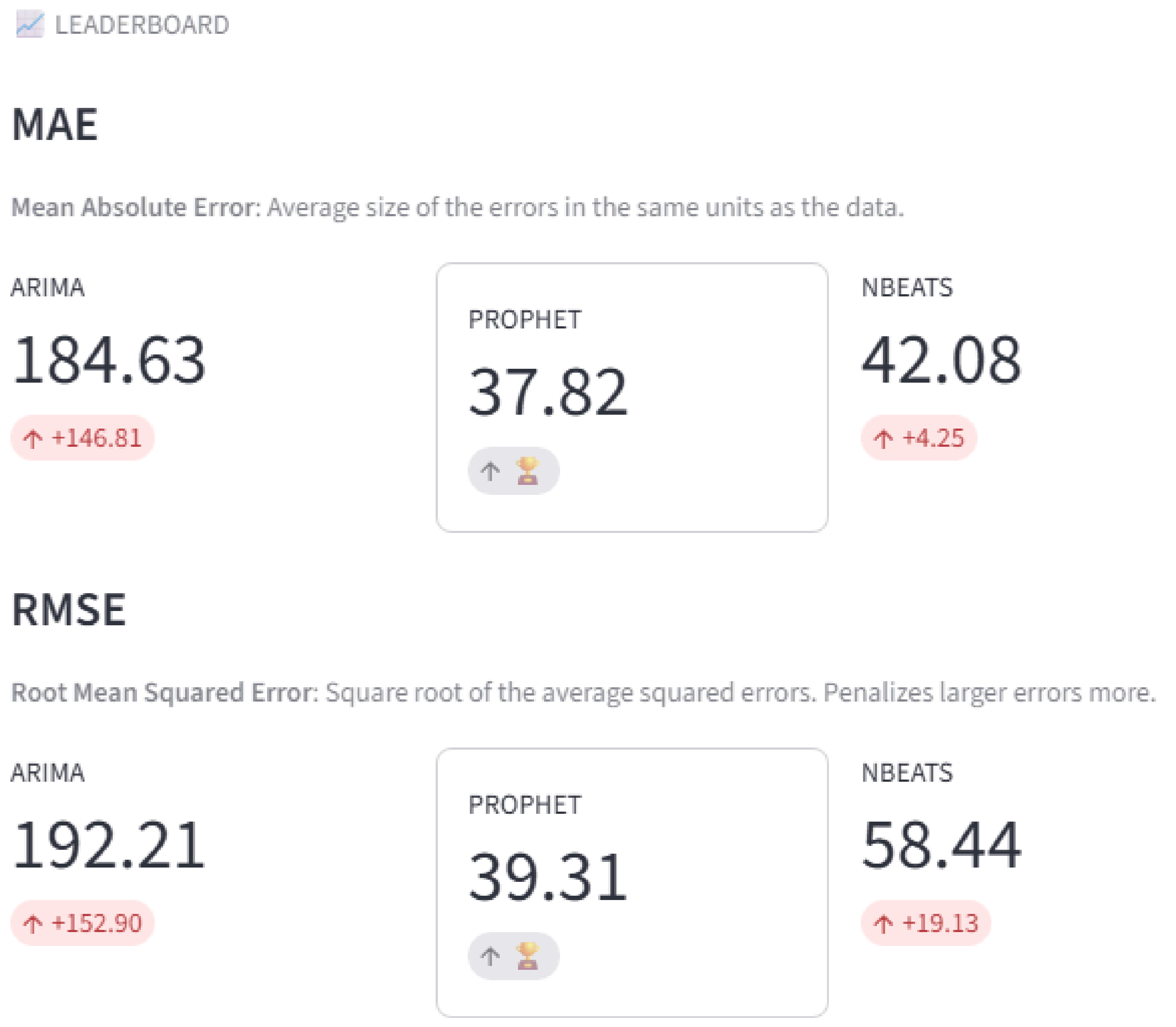
37.82

RMSE ?

39.31

Key features

MODEL COMPARISON



📈 LEADERBOARD

MAE

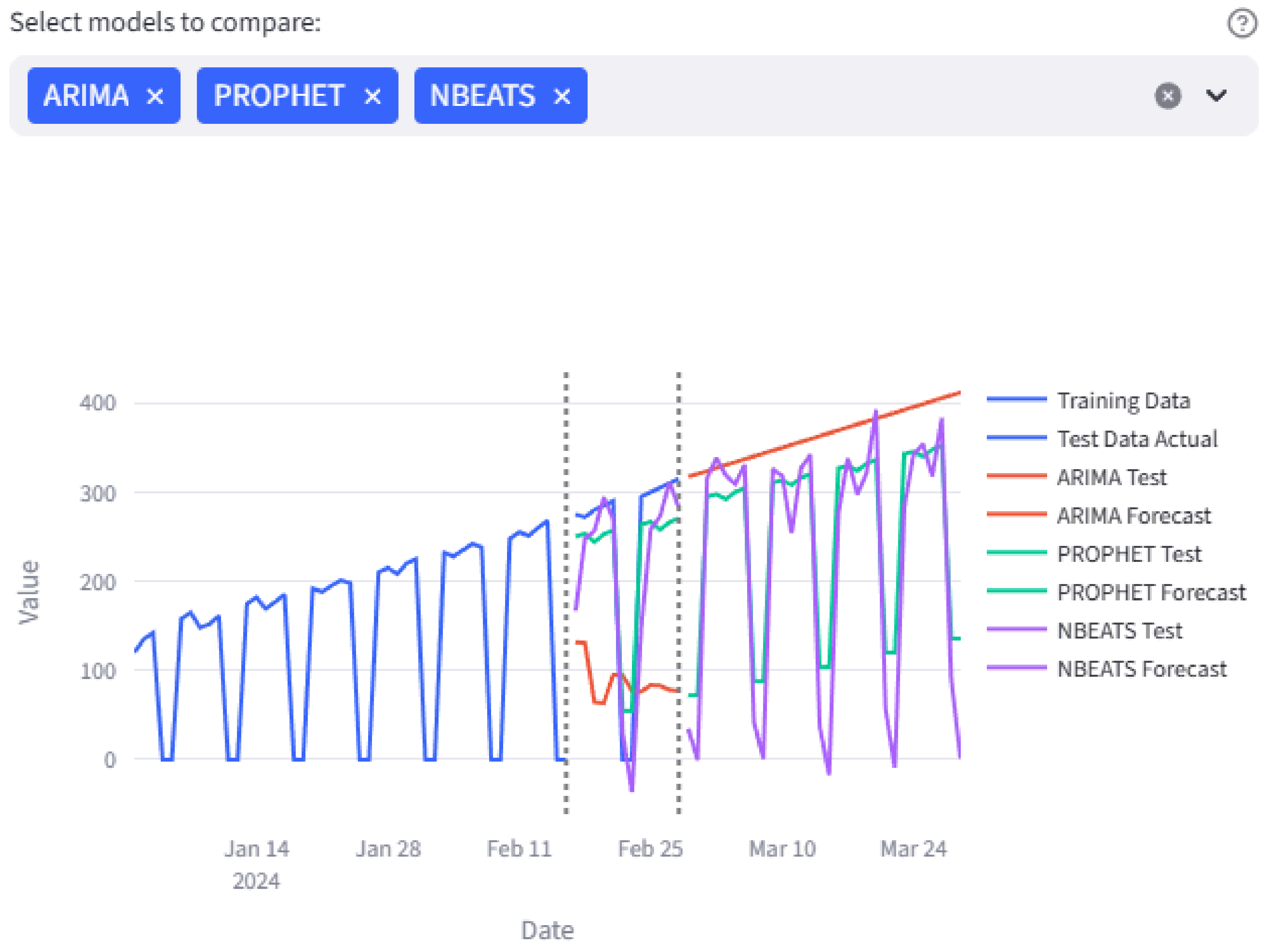
Mean Absolute Error: Average size of the errors in the same units as the data.

ARIMA	PROPHET	NBEATS
184.63	37.82	42.08
↑ +146.81	↑ 🏆	↑ +4.25

RMSE

Root Mean Squared Error: Square root of the average squared errors. Penalizes larger errors more.

ARIMA	PROPHET	NBEATS
192.21	39.31	58.44
↑ +152.90	↑ 🏆	↑ +19.13



Select models to compare:

ARIMA × PROPHET × NBEATS ×

Methodological Approach

CORE PHILOSOPHY

Prototype first

Validate tech choices early with minimal prototypes before full commitment to architecture.

Prioritize features

Focus on delivering core value first.

STRATEGIC TRADE-OFFS

- ✓ Working prototype at every stage
- ✗ Minor architectural inconsistencies

IMPLEMENTATION TIMELINE

1. **Core Functionalities**

   Data loading, basic model training, forecasting setup.

2. **Advanced Features**

   Adding hyperparameter recommendations, model comparison and data preprocessing functionalities.

3. **Architecture Refinement**

   Breaking megalithic script into UI, state, services and model layers.

4. **Deployment & Documentation**

   Finalizing docs, deployment and slide preparation.

Tech stack

SELECTED STACK

Python + Streamlit

Data Science + Rapid UI

TRADE-OFFS

- ✓ Rapid Iteration
- ✓ Native ML Support
- ✓ Minimal Frontend Code

- ✗ Layout customization limited
- ✗ State management is simple but rigid

ALTERNATIVES ASSESMENT

Python + Streamlit ★★★★★

Fast prototyping. Simple and familiar syntax.

Python + Dash ★★★☆☆

Steeper learning curve. Unnecessary complexity for assignment.

R + Shiny ★★★☆☆

Less familiar syntax. Python ecosystem superior for deep learning.

Java + Weka ☆☆☆☆☆

Pure engineering curiosity.

Code Architecture

BASIC STRUCTURE

Presentation Layer
Streamlit UI & State Management

Application Layer
Data and Model Services

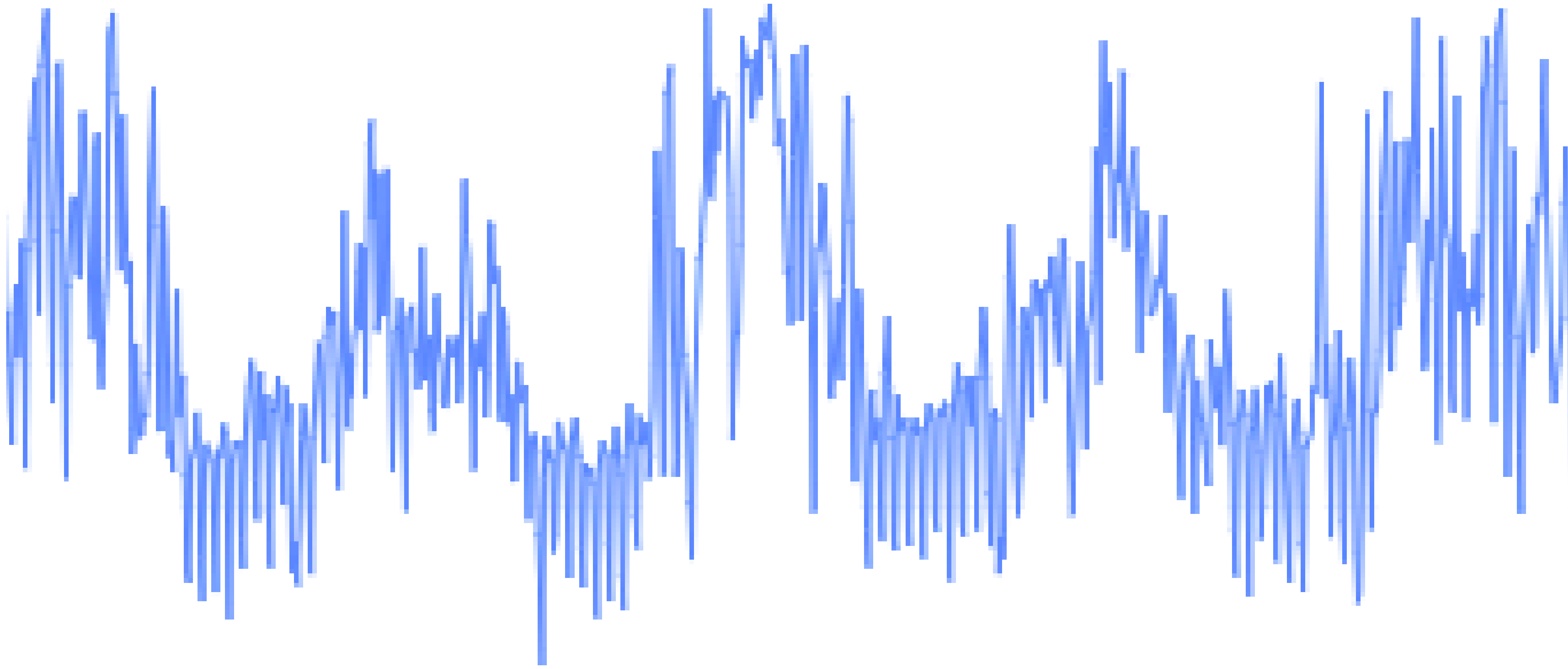
Model Layer
ARIMA, Prophet, N-BEATS

RATIONALE

Separation of Concerns
UI code never manipulates data or models directly, delegates to Service layer.

Testability & Reusability
Business logic in the Service and Model layers could easily power a different UI or serve REST API.

Extensibility
New models are added by simply implementing the `ForecastModel` base class. The UI automatically registers and renders tabs for any registered model without manual UI updates.



Code Deep Dive <>

https://github.com/fsentin/forecast

File structure

```
forecast/
├──── app.py      # Starting point
├──── ui/         # Streamlit UI
├──── state/      # Session management
├──── services/   # Data and model orchestration
├──── models/     # ARIMA, Prophet, N-BEATS
└──── utils/      # Reusable utilities
```

Presentation layer

FILE STRUCTURE

```
├──── app.py
├──── ui/
│  └──── components/
│       ├──── historical_tab.py
│       ├──── model_tab.py
│       ├──── comparison_tab.py
│       └──── sidebar.py
└──── state/
   └──── app_state.py
```

INTERFACE

```
class AppState:

initialize()
set_preprocessing_state(pending,
                        data,
                        count)
get_preprocessing_state()
get_upload_key()
has_data()
set_data(historical_data)
get_data()
```

Presentation layer

FILE STRUCTURE

```
├──── app.py
├──── ui/
│  └──── components/
│     ├──── historical_tab.py
│     ├──── model_tab.py
│     ├──── comparison_tab.py
│     └──── sidebar.py
└──── state/
  └──── app_state.py
```

INTERFACE

```
class AppState: (continued)

get_model_config(model_name)
set_hyperparameter(model_name,
                   param_name,
                   value)
get_hyperparameter(model_name,
                   param_name,
                   default)
set_model_results(model_name,
                  metrics,
                  predictions,
                  test_predictions,
                  train_pct,
                  horizon)
get_model_results(model_name)
```

Presentation layer

FILE STRUCTURE

```
├──── app.py
├──── ui/
│   └──── components/
│         ├──── historical_tab.py
│         ├──── model_tab.py
│         ├──── comparison_tab.py
│         └──── sidebar.py
└──── state/
   └──── app_state.py
```

INTERFACE

```
class AppState: (continued)

clear_preprocessing_state()
get_all_trained_models()
clear_all_models()
reset_all()
```

Application layer

FILE STRUCTURE

```
├──── services/
│   ├──── data_service.py
│   └──── model_service.py
```

INTERFACE

```
class DataService:
  prepare_dataframe(data,
                    date_column,
                    value_column)
  check_gaps(data)
  detect_and_remove_outliers(data)


class ModelService:
  train_and_evaluate(model_class,
    data, train_pct, horizon,
    upload_key, hyperparams)
```

Model Layer

FILE STRUCTURE

```
├──── models/
│    ├──── base.py
│    ├──── arima.py
│    ├──── prophet.py
│    └──── nbeats.py
└──── splitters/
      ├──── base.py
      └──── holdoutpct.py
```

INTERFACE

```
class ForecastModel(ABC):
    fit(data)
    predict(horizon)
    evaluate(data,
             splitter,
             horizon,
             metric_functions)
    has_recommendations()
    get_recommendations(data)
    get_hyperparameters()

class TimeSeriesSplitter(ABC):
    split(data)
```

Wild card: utils

FILE STRUCTURE

```
├──── utils/
    ├──── input_validation.py   # validation helpers
    ├──── model_evaluation.py   # metrics def & calculation
    ├──── plotting.py           # plotly graphs
    └──── timeseries.py         # general domain specific
```

Reminder:

CORE PHILOSOPHY

Prototype first ✓ Working prototype at every stage

Prioritize features ✗ Minor architectural inconsistencies

Bad practice!

mix of different layers

Further refactoring

```
forecast/
├───── app.py
├───── ui/
│    ├───── components/
│    └───── plotting.py       # move from utils
├───── state/
├───── services/
│    ├───── data_service.py   # move all validation logic here
│    └───── model_service.py  # move all evaluation logic here
├───── models/
│    ├───── arima.py
│    ├───── prophet.py
│    └───── nbeats.py
└───── utils/
    ├───── timeseries.py        # pure domain utilities
    └───── metrics.py           # pure metric calculations
```

Future enhancements

Diagnostic Tools

Add residual diagnostics plots and error distribution analysis.

Code Quality

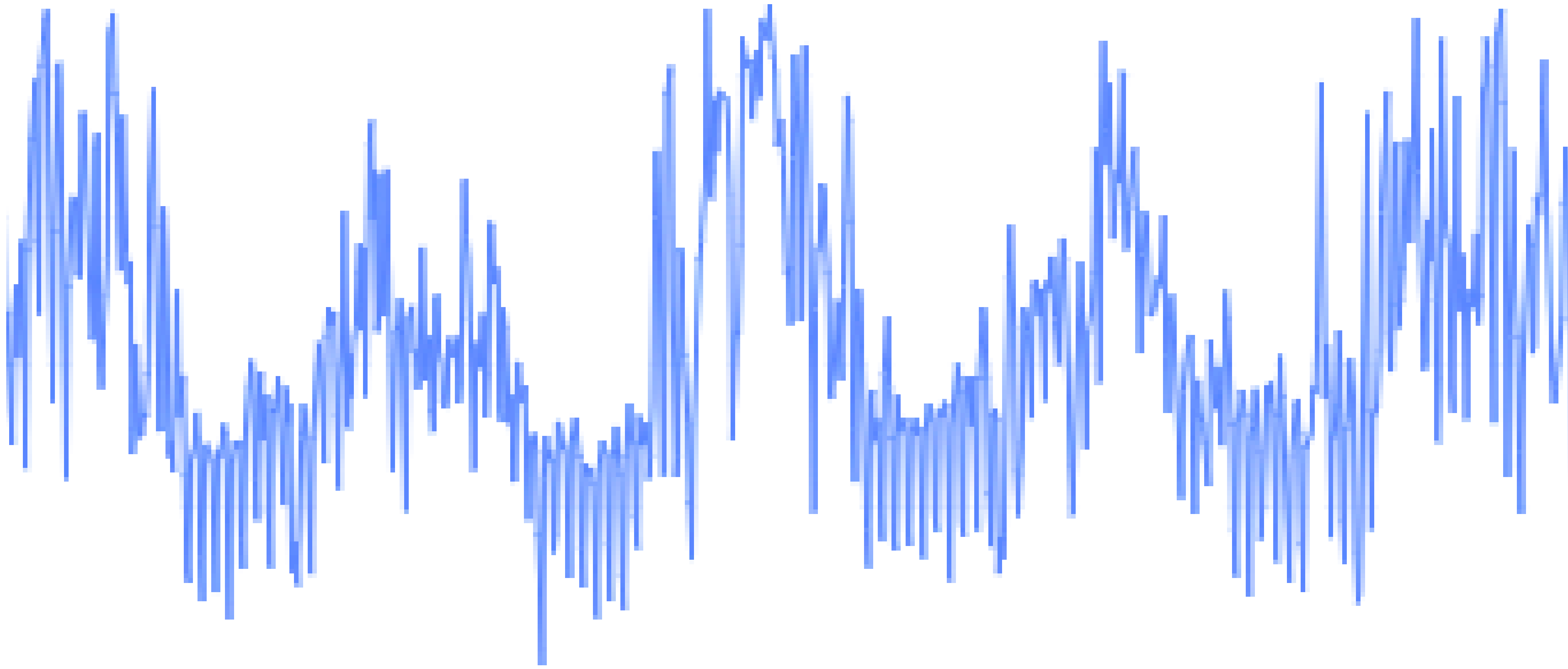
Refactor complete separation of concerns and add comprehensive unit tests.

Advanced Evaluation

Implement alternative split streategies.

Auto-Optimization

Add hyperparameter search for auto selection.



Live Demo

https://forecast-tool.streamlit.app