ISA 444: Business Forecasting

08: Forecasting Environment

Fadel M. Megahed, PhD

Professor
Farmer School of Business
Miami University

@FadelMegahed
fmegahed
fmegahed@miamioh.edu
Automated Scheduler for Office Hours

Spring 2025

Quick Refresher of Last Class

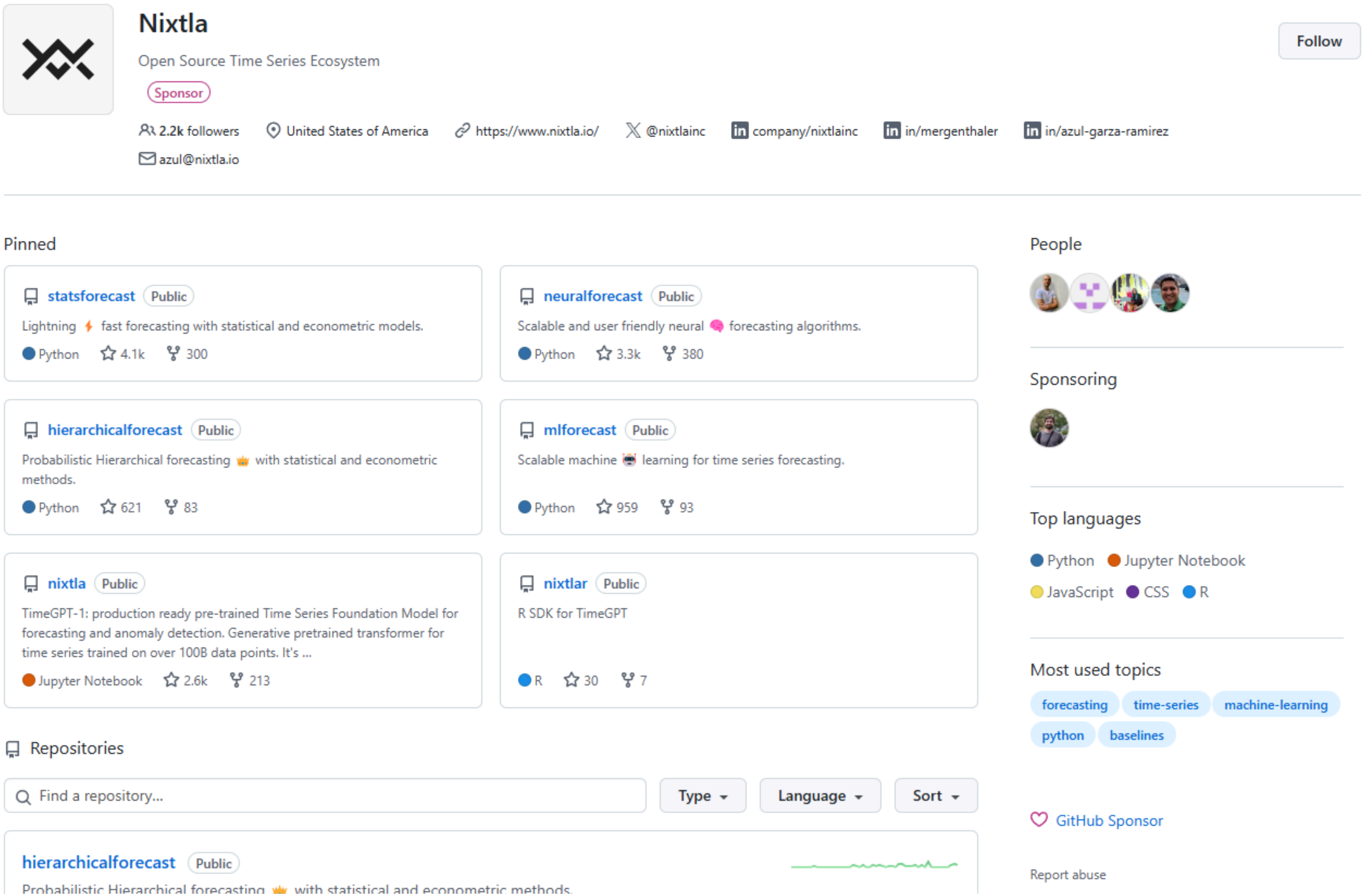
- ☑ Explain the differences between wide vs. long format
- ☑ Use seaborn to plot multiple time-series
- ☑ Convert a data set to Nixtla's long format (`unique_id`, `ds`, `y`)
- ☑ Use UtilsForecast to visualize multiple series

Learning Objectives for Today's Class

- Install and import Nixtla's libraries (StatsForecast, MLForecast, NeuralForecast, UtilsForecast, and TimeGPT) for forecasting
- Distinguish fixed window from rolling-origin
- Introduce forecast accuracy metrics (MAE, MAPE, RMSE)

The Nixtlaverse Open-Source Forecasting Libraries

Nixtla's Forecasting Libraries



Nixtla's Forecasting Libraries

Nixtla provides **several open-source Python libraries** (and a closed source **TimeGPT** tool accessible via API calls) for **scalable forecasting tasks**. These libraries are **relatively** easy to use and can be integrated into your future forecasting workflows:

- **StatsForecast** – Fast & scalable statistical models (`ARIMA`, `ETS`, etc.).
- **MLForecast** – Machine learning-based forecasting (e.g., `XGBoost`, `LightGBM`).
- **NeuralForecast** – Deep learning models for time series (e.g., `NBEATS`, `NHITS`, and `TFT`).
- **UtilsForecast** – Utility functions for plotting, evaluation, etc.
- **TimeGPT** – an AI transformer-powered forecasting API that requires minimal tuning.

These libraries enable **forecasting at scale, which you will need in practice**.

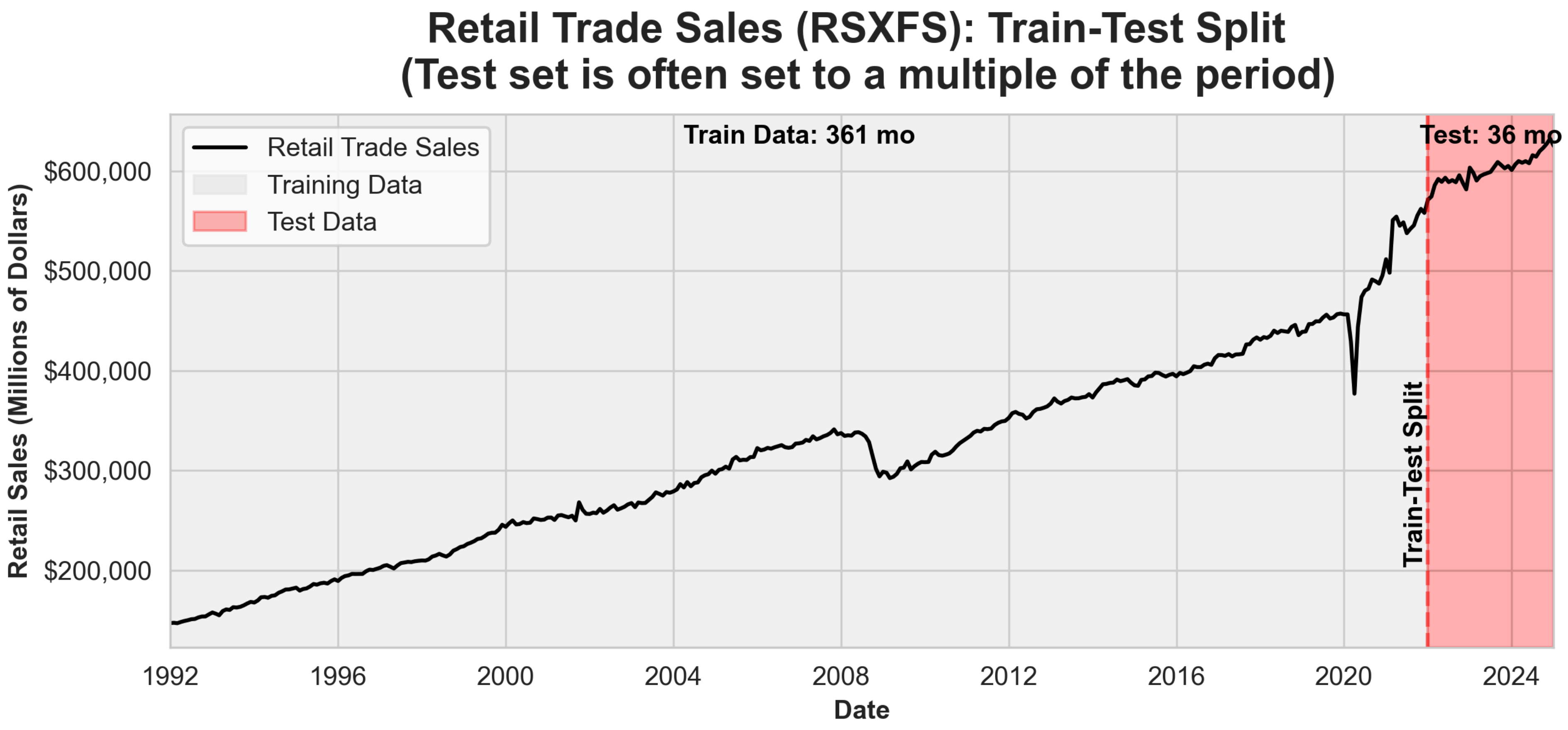
Note: These libraries can be installed via `pip` and are described in detail in nixtlaverse.nixtla.io. Use the left-hand navigation bar to explore the documentation for each library.

Fixed Window vs. Rolling-Origin

The Fixed Window Evaluation Approach

- **Fixed Window** is the simplest approach to splitting your time series data into a training and a testing/holdout set.
- The **goal** is to **train your model on the training set** and **evaluate its performance on the testing set (the last k observations in the data)**.
  - **Note that this is quite different than traditional machine learning applications for cross sectional data.**
- The evaluation on the **testing/holdout** set can serve two purposes:
  - **Model Evaluation**: Assess the model's **performance on unseen data** (since it is not used during training, and hence acts as a proxy for the model's performance on future data).
  - **Model Selection**: Compare the **performance of different models** to select the **best one**.
    - Note that using this approach for model selection is **reasonable if your models do not involve hyperparameter tuning** (otherwise, you may overfit to the testing set).

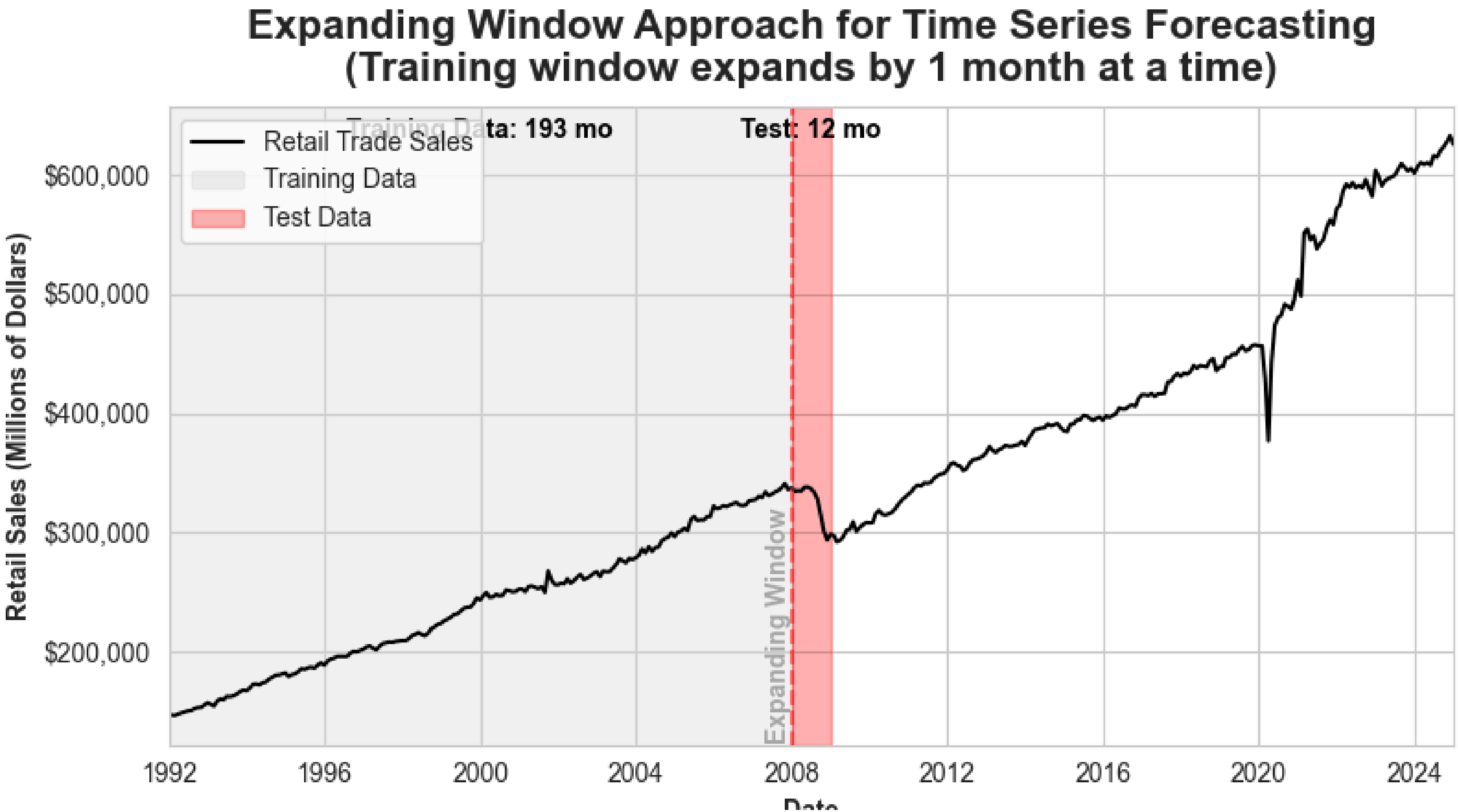
The Fixed Window Evaluation Approach



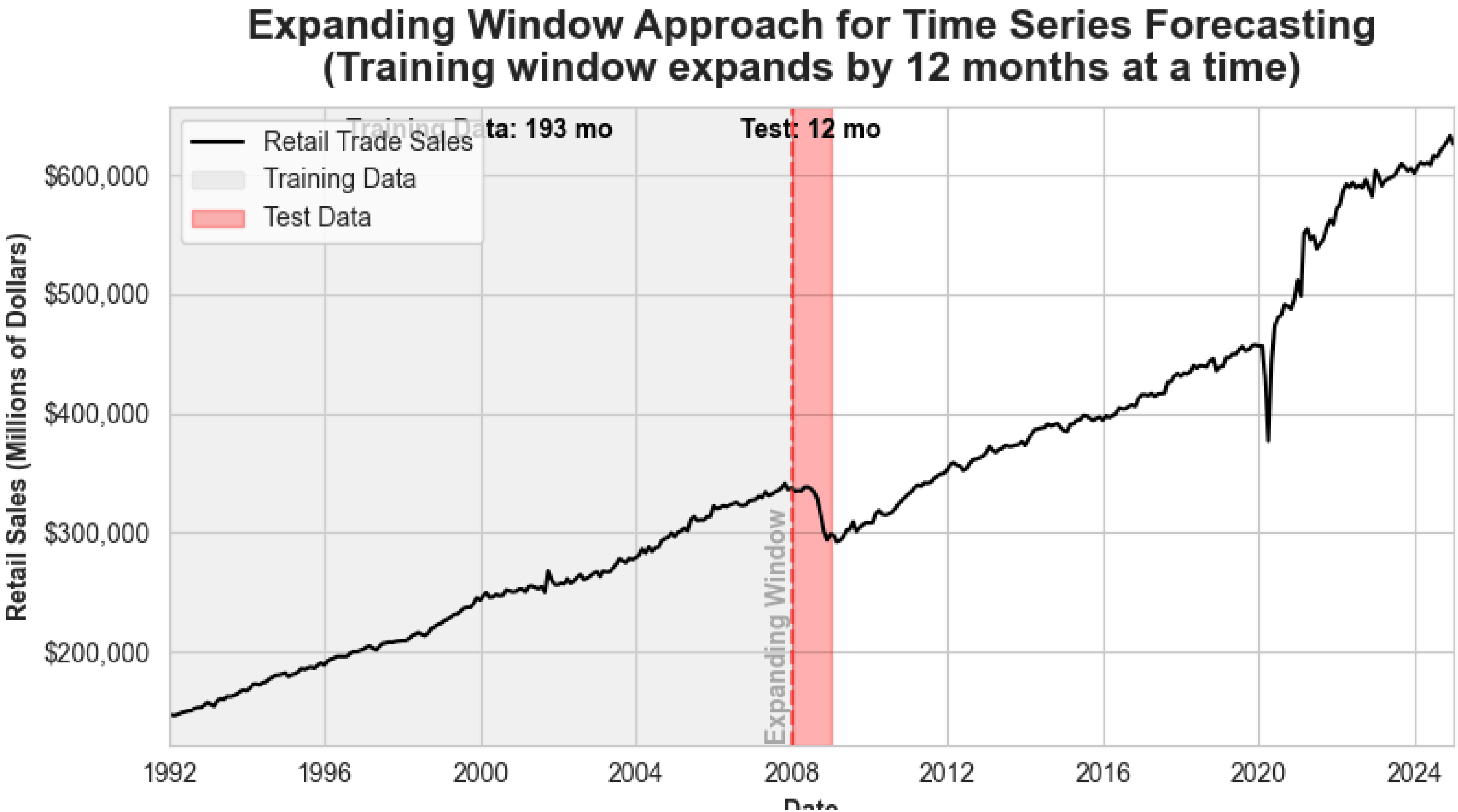
The Rolling-Origin Evaluation Approach

- **Rolling-Origin Evaluation** is a method for splitting time series data into training and testing sets where the testing sets move forward over time.
- The **goal** is to **train the model on a subset of past observations** and evaluate its **performance on a future testing set at multiple time steps**.
- The key difference from a fixed window approach is that the **testing set shifts forward, allowing for multiple evaluations**:
  - The **training set may expand (expanding window)** or **remain fixed (rolling window)**.
  - This ensures that model performance is assessed across **different points in time**.
  - It **reduces sensitivity to the initial split point** and provides a more **robust evaluation** of model performance over time.

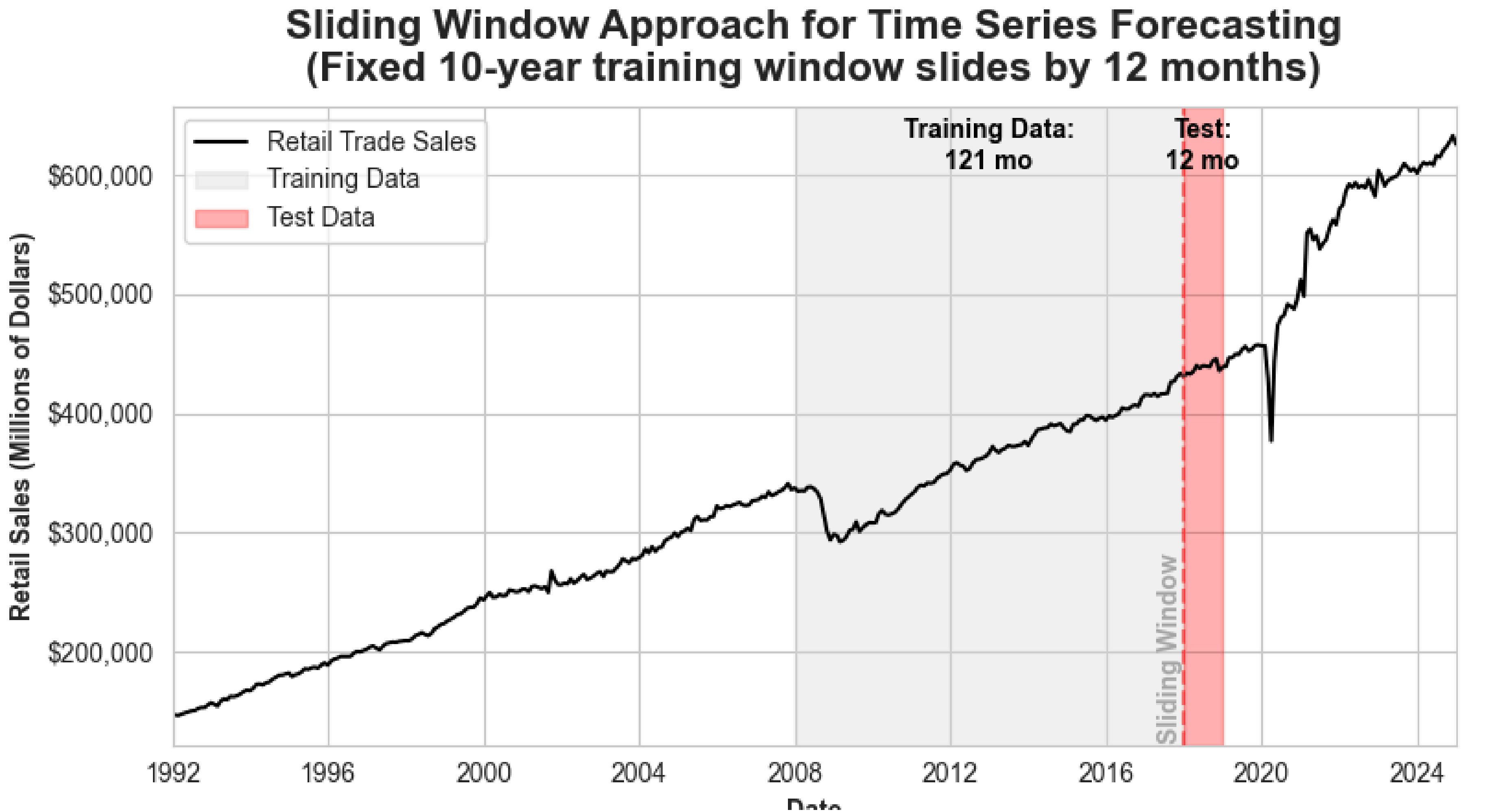
Expanding Window Evaluation (By 1 Month)



Expanding Window Evaluation (By 12 Month)



Rolling Non-Expanding Window Evaluation



Cross Validation within the Nixtlaverse

Perform time series cross-validation

Once the `StatsForecast` object has been instantiated, we can use the `cross_validation` method, which takes the following arguments:

- `df` : training data frame with `StatsForecast` format
- `h` (int): represents the h steps into the future that will be forecasted
- `step_size` (int): step size between each window, meaning how often do you want to run the forecasting process.
- `n_windows` (int): number of windows used for cross-validation, meaning the number of forecasting processes in the past you want to evaluate.

For this particular example, we'll use 3 windows of 24 hours.

```
cv_df = sf.cross_validation(
    df = df,
    h = 24,
    step_size = 24,
    n_windows = 3
  )
```

The `cv_df` object is a new data frame that includes the following columns:

- `unique_id` : series identifier
- `ds` : datestamp or temporal index
- `cutoff` : the last datestamp or temporal index for the n_windows.
- `y` : true value
- `"model"` : columns with the model's name and fitted value.

Note: The `cross_validation` method can also be applied to other NixtlaForecast objects (e.g. `MLForecast`, `NeuralForecast`) to perform cross-validation for machine learning and deep learning models. See the StatsForecast Cross Validation Tutorial to access the page shown above.

Activity: The `cross_validation` Method

ross_validation method from Nixtla implements:

0

0

d Window Evaluation

Expanding Rolling Window

Non-Expanding Rolling Window

Recap of Fixed vs. Rolling-Origin

- **Fixed Window**:
  - **Simplest approach** to splitting data into training and testing sets.
  - **Testing set is fixed** and **does not move forward** over time.
  - **Provides a single evaluation** of model performance.
- **Rolling-Origin**:
  - **Testing set moves forward** over time.
  - **Training set may expand or remain fixed**.
  - **Provides multiple evaluations** of model performance.
  - **Reduces sensitivity to the initial split point**.
  - **More robust evaluation** of model performance over time.

Recap of Fixed vs. Rolling-Origin

- In practice, the **rolling-origin approach is preferred** for time series forecasting tasks since it mimics the real-world scenario of forecasting future data points. The choice of:
  - **Expanding vs. Rolling Window**,
  - **Window Size**, and
  - **Step Size** depends on the specific forecasting task and the data at hand.

Forecast Accuracy Metrics

Model Performance Evaluation in the Nixtlaverse

```
from utilsforecast.losses import *
```

evaluate

```
evaluate (df:~AnyDFType, metrics:List[Callable],
           models:Optional[List[str]]=None,
           train_df:Optional[~AnyDFType]=None,
           level:Optional[List[int]]=None, id_col:str='unique_id',
           time_col:str='ds', target_col:str='y',
           agg_fn:Optional[str]=None)
```

Source: Nixtla's UtilsForecast Evaluation Documentation

Model Performance Evaluation in the Nixtlaverse (Cont.)

	Type	Default	Details
df	AnyDFType		Forecasts to evaluate. Must have `id_col`, `time_col`, `target_col` and models' predictions.
metrics	List		Functions with arguments `df`, `models`, `id_col`, `target_col` and optionally `train_df`.
models	Optional	None	Names of the models to evaluate. If `None` will use every column in the dataframe after removing id, time and target.
train_df	Optional	None	Training set. Used to evaluate metrics such as `mase`.
level	Optional	None	Prediction interval levels. Used to compute losses that rely on quantiles.
id_col	str	unique_id	Column that identifies each serie.
time_col	str	ds	Column that identifies each timestep, its values can be timestamps or integers.
target_col	str	y	Column that contains the target.
agg_fn	Optional	None	Statistic to compute on the scores by id to reduce them to a single number.
Returns	**AnyDFType**		**Metrics with one row per (id, metric) combination and one column per model. If `agg_fn` is not `None`, there is only one row per metric.**

Source: Nixtla's UtilsForecast Evaluation Documentation

Losses

The most important train signal is the forecast error, which is the difference between the observed value y_τ and the prediction $\hat{y}_\tau$, at time y_τ:

$$e_\tau = y_\tau - \hat{y}_\tau \qquad \tau \in \{t+1, \ldots, t+H\}$$

The train loss summarizes the forecast errors in different evaluation metrics.

Source: Nixtla's UtilsForecast Losses Documentation

Scale-Dependent Errors: `mae`

MAE measures the relative prediction accuracy by averaging the absolute deviations between forecasts and actual values.

$$\mathrm{MAE}(\mathbf{y}_\tau, \hat{\mathbf{y}}_\tau) = \frac{1}{H} \sum_{\tau=t+1}^{t+H} |y_\tau - \hat{y}_\tau|$$

- **Interpretation**: Provides a straightforward measure of forecast accuracy; lower MAE indicates better performance.
- **Characteristic:** Does not penalize larger errors more than smaller ones; treats all errors equally.

Source: Nixtla's UtilsForecast Losses Documentation

Scale-Dependent Errors: `rmse`

RMSE is the square root of the average of the squared differences between forecasts and actual values.

$$\mathrm{RMSE}(y_\tau, \hat{y}_\tau) = \sqrt{\frac{1}{H} \sum_{\tau=t+1}^{t+H} (y_\tau - \hat{y}_\tau)^2}$$

- **Interpretation**: Emphasizes larger errors due to squaring; useful when large errors are particularly undesirable.
- **Characteristic:** Penalizes large errors more than MAE (i.e., more sensitive to ourliers compared to MAE).

Source: Nixtla's UtilsForecast Losses Documentation

Percentage Errors: `mape`

MAPE calculates the average absolute error as a percentage of actual values.

$$\mathrm{MAPE}(y_\tau, \hat{y}_\tau) = \frac{1}{H} \sum_{\tau=t+1}^{t+H} \left| \frac{y_\tau - \hat{y}_\tau}{y_\tau} \right|$$

Interpretation: Expresses forecast accuracy as a percentage; lower MAPE indicates better performance.

Characteristic: Can be misleading if actual values are close to zero, leading to extremely high MAPE values.

Source: Nixtla's UtilsForecast Losses Documentation

Recap

Summary of Main Points

By now, you should be able to do the following:

- Install and import Nixtla's libraries (StatsForecast, MLForecast, NeuralForecast, UtilsForecast, and TimeGPT) for forecasting
- Distinguish fixed window from rolling-origin
- Introduce forecast accuracy metrics (MAE, MAPE, RMSE)

📝 Review and Clarification 📝

- **Class Notes**: Take some time to revisit your class notes for key insights and concepts.
- **Zoom Recording**: The recording of today's class will be made available on Canvas approximately 3-4 hours after the session ends.
- **Questions**: Please don't hesitate to ask for clarification on any topics discussed in class. It's crucial not to let questions accumulate.