NYU FRE 7773 - Week 6

Machine Learning in Financial Engineering

Ethan Rosenthal

Time Series Machine Learning

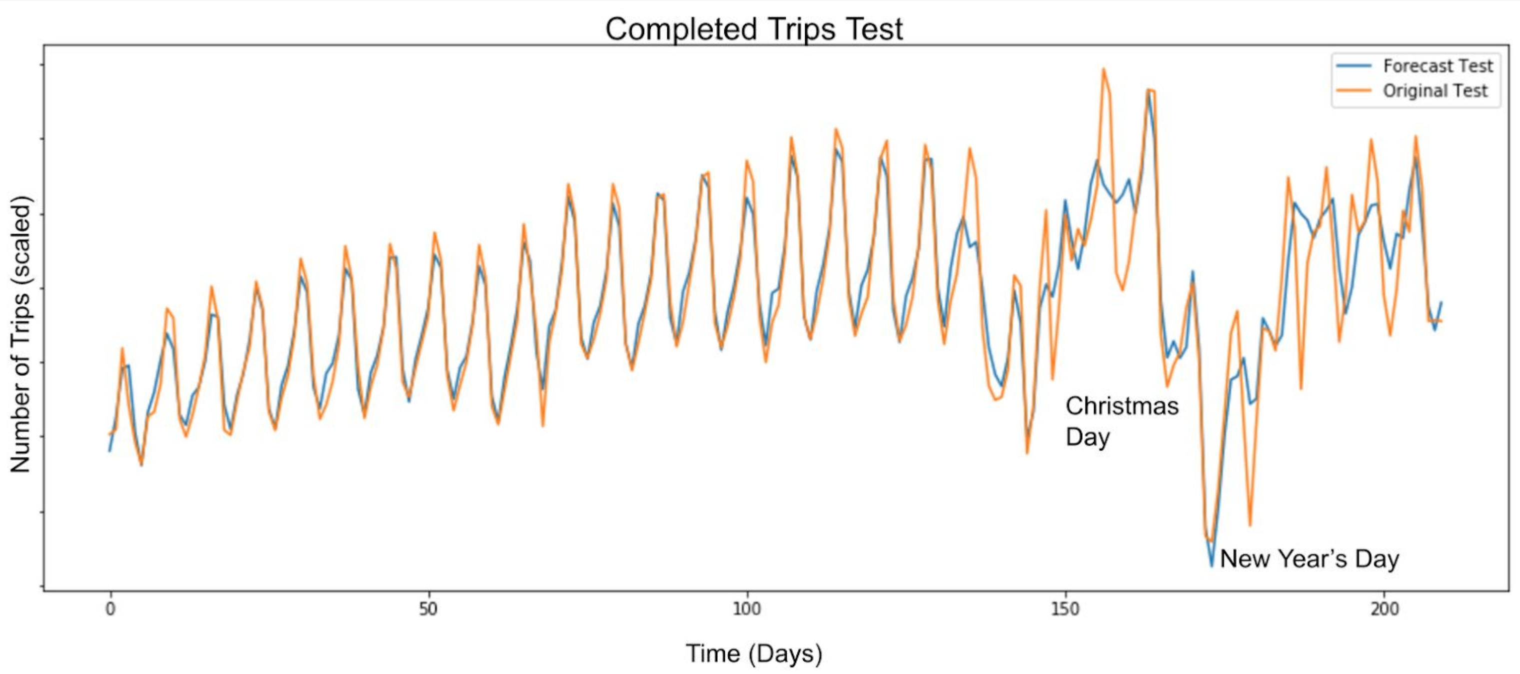
Machine Learning in Financial Engineering

Ethan Rosenthal

Time

https://commons.wikimedia.org/wiki/File:S_and_P_500_daily_linear_chart_1950_to_2016.png

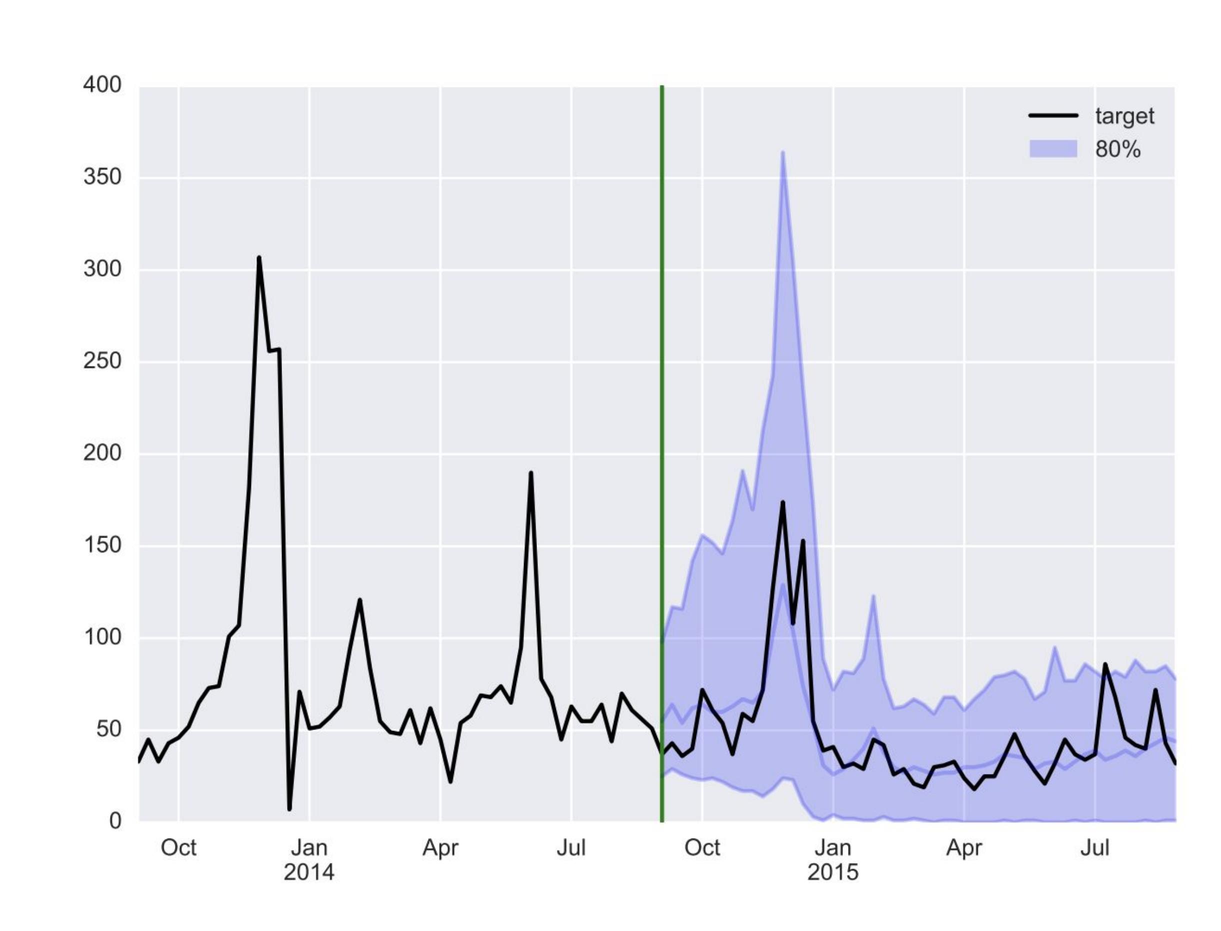
Uber trips
https://eng.uber.com/neural-networks/



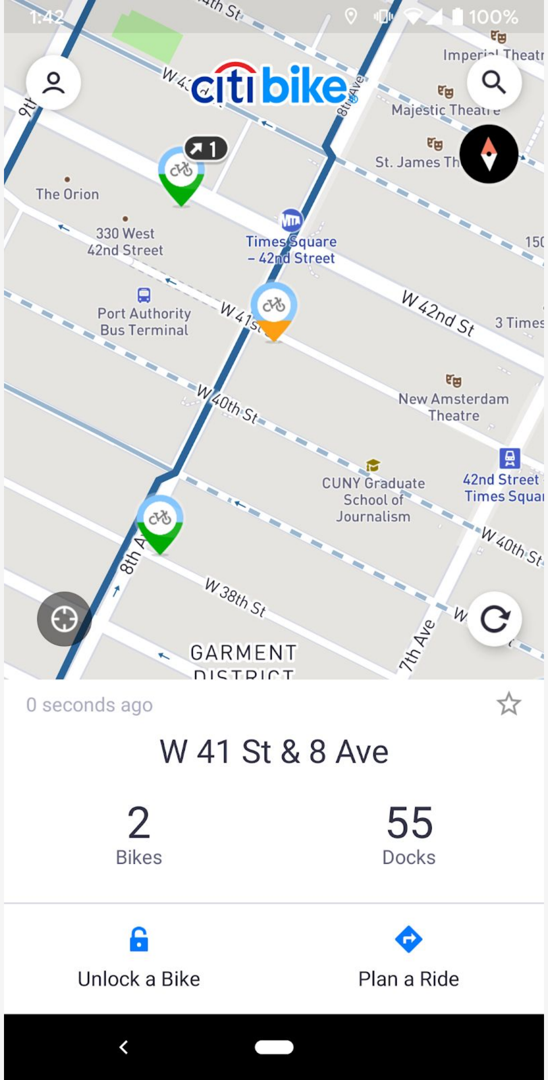
Amazon weekly item sales

DeepAR: Probabilistic Forecasting with Autoregressive Recurrent Networks

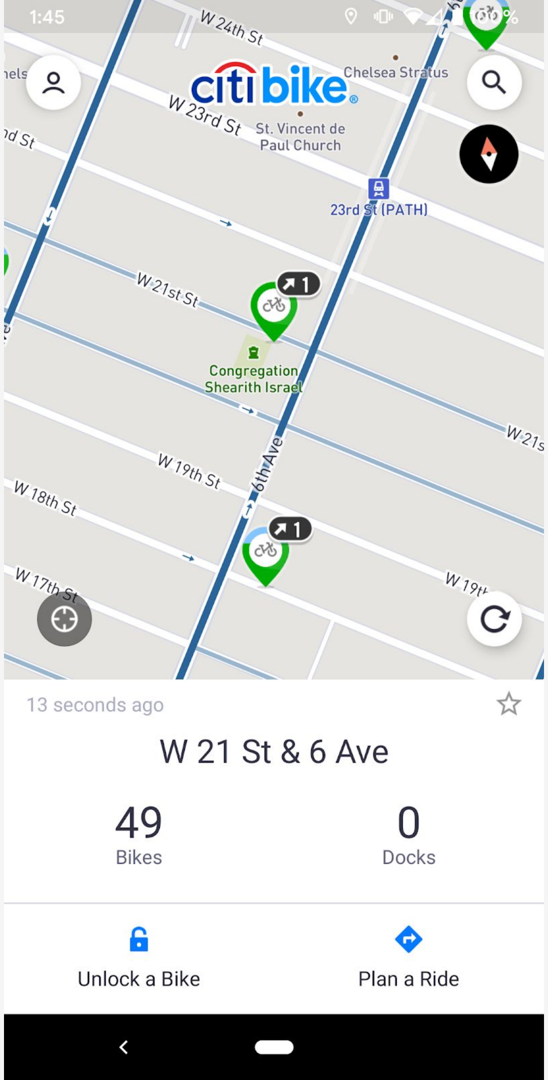
https://arxiv.org/abs/1704.04110

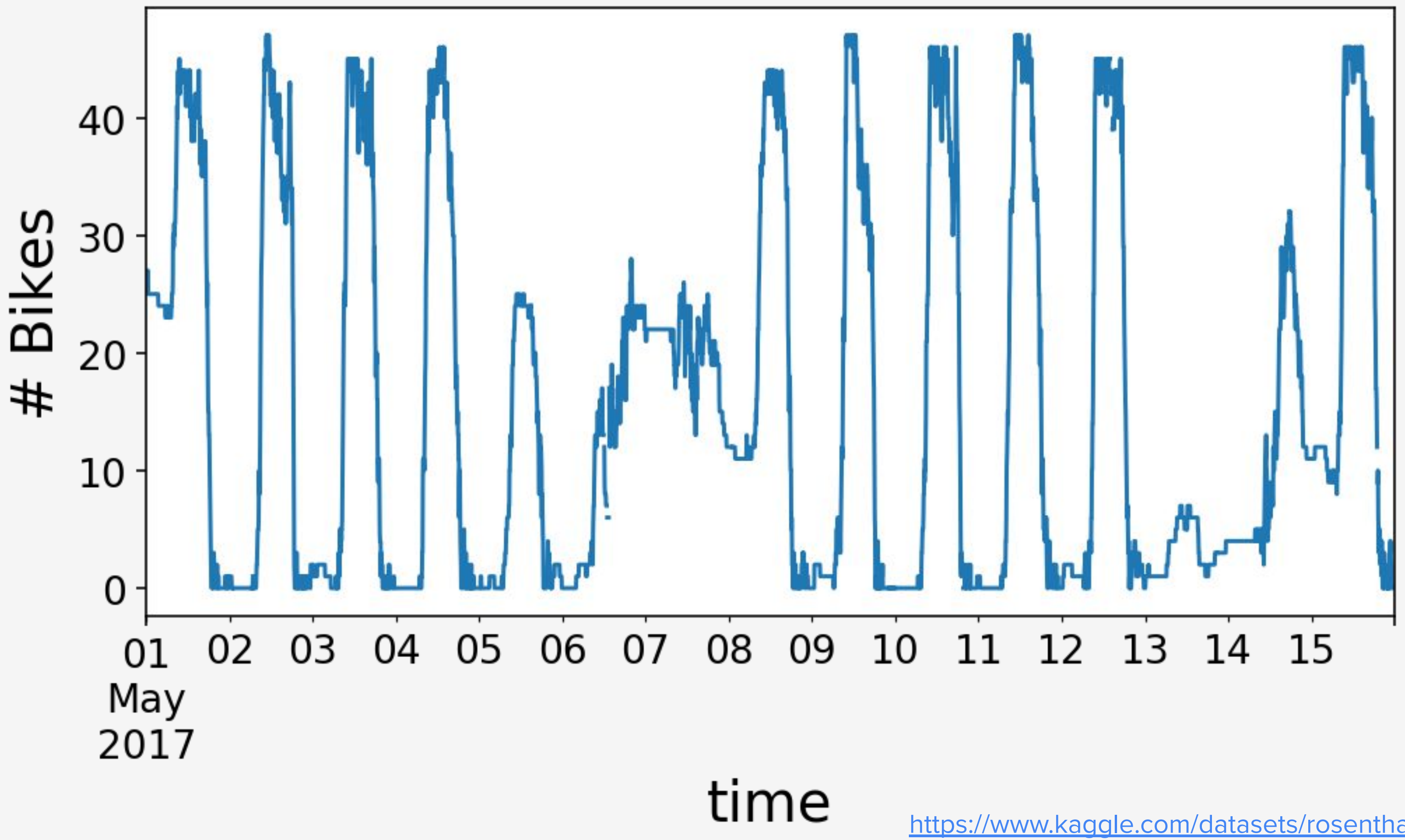


No Citi Bikes

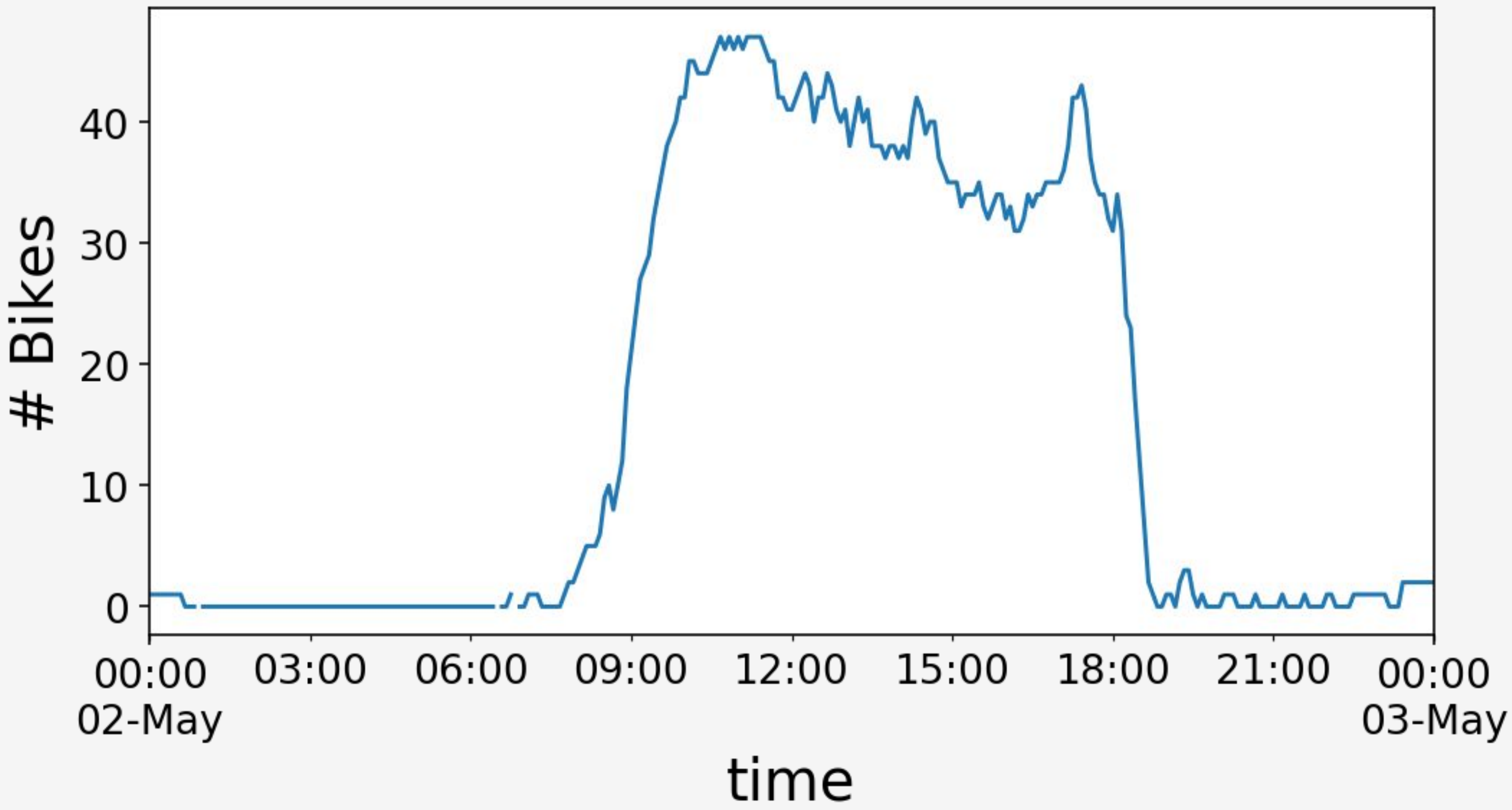


No Docks





https://www.kaggle.com/datasets/rosenthal/citi-bike-stations



https://www.kaggle.com/datasets/rosenthal/citi-bike-stations

“Classical” Time Series Modeling

State Space Models

Nonlinear PDE’s

ARIMA

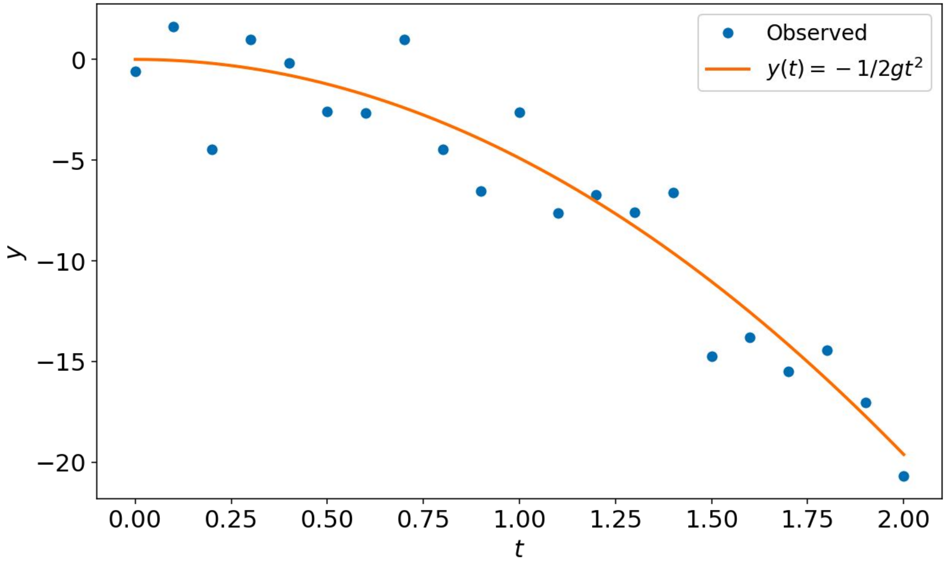
Gauss-Markov Theorem

$$\begin{array}{ccccc}
(1 - \phi_1 B - \cdots - \phi_p B^p) & (1 - B)^d y_t & = & c + (1 + \theta_1 B + \cdots + \theta_q B^q)\varepsilon_t \\
\uparrow & \uparrow & & \uparrow \\
\text{AR}(p) & d \text{ differences} & & \text{MA}(q)
\end{array}$$

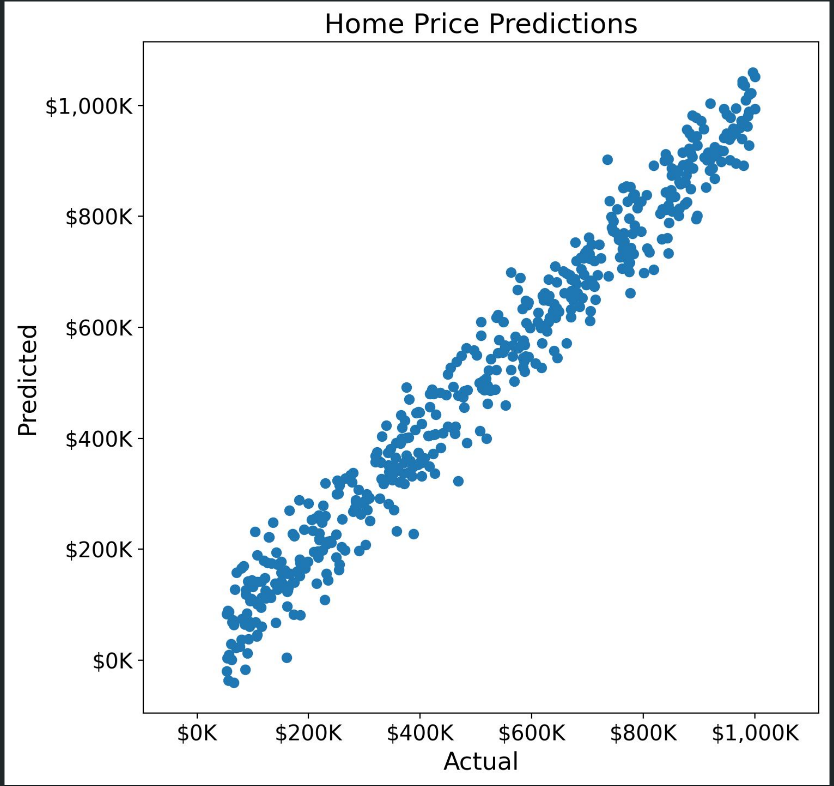
https://otexts.com/fpp2/non-seasonal-arima.html

Can you skip all of this?

Inference



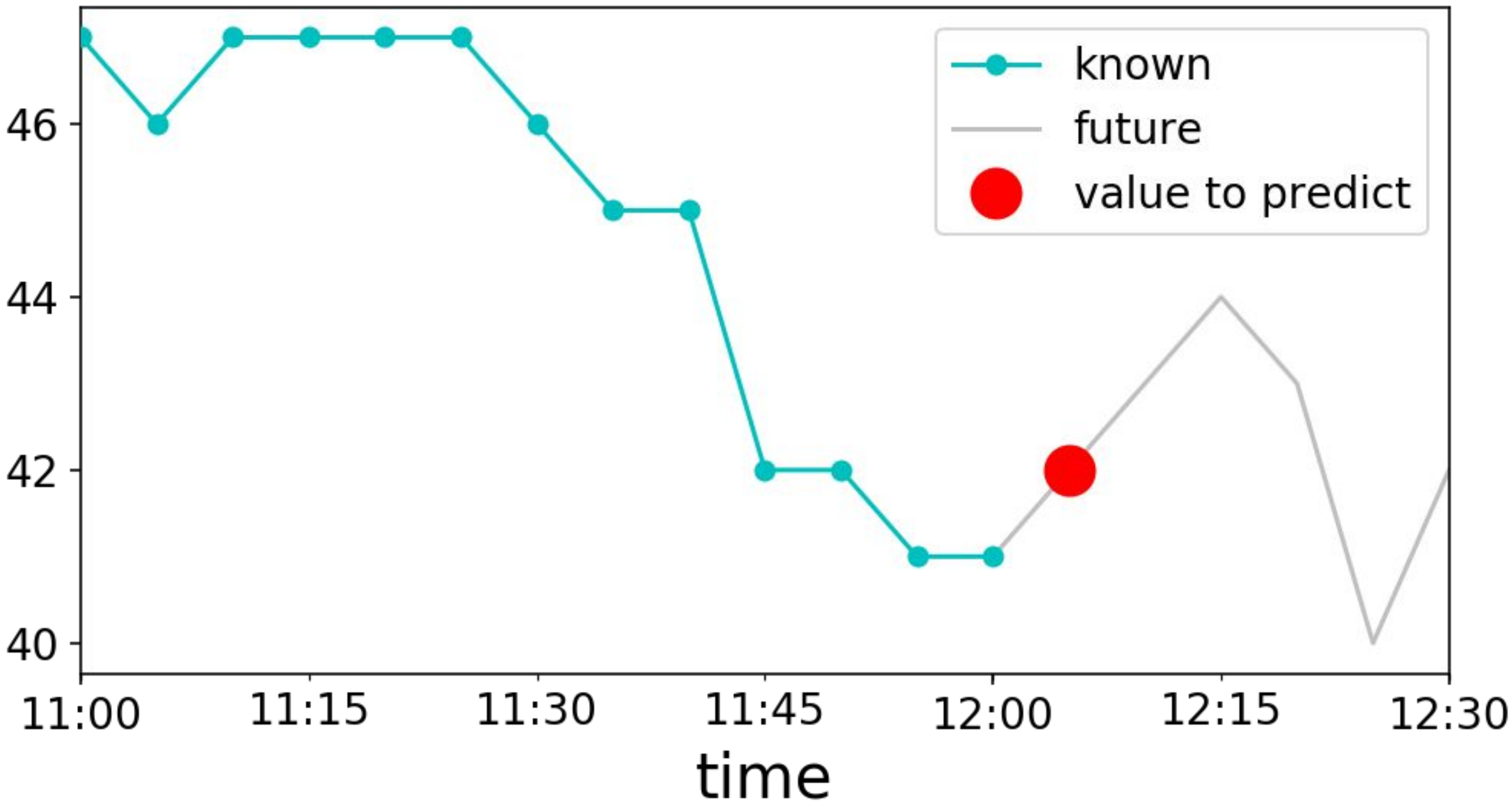
Prediction



Where’s the X Matrix?

```
f.fit(X, y)
y_new = f.predict(X_new)
```

In the beginning, there was ***y***



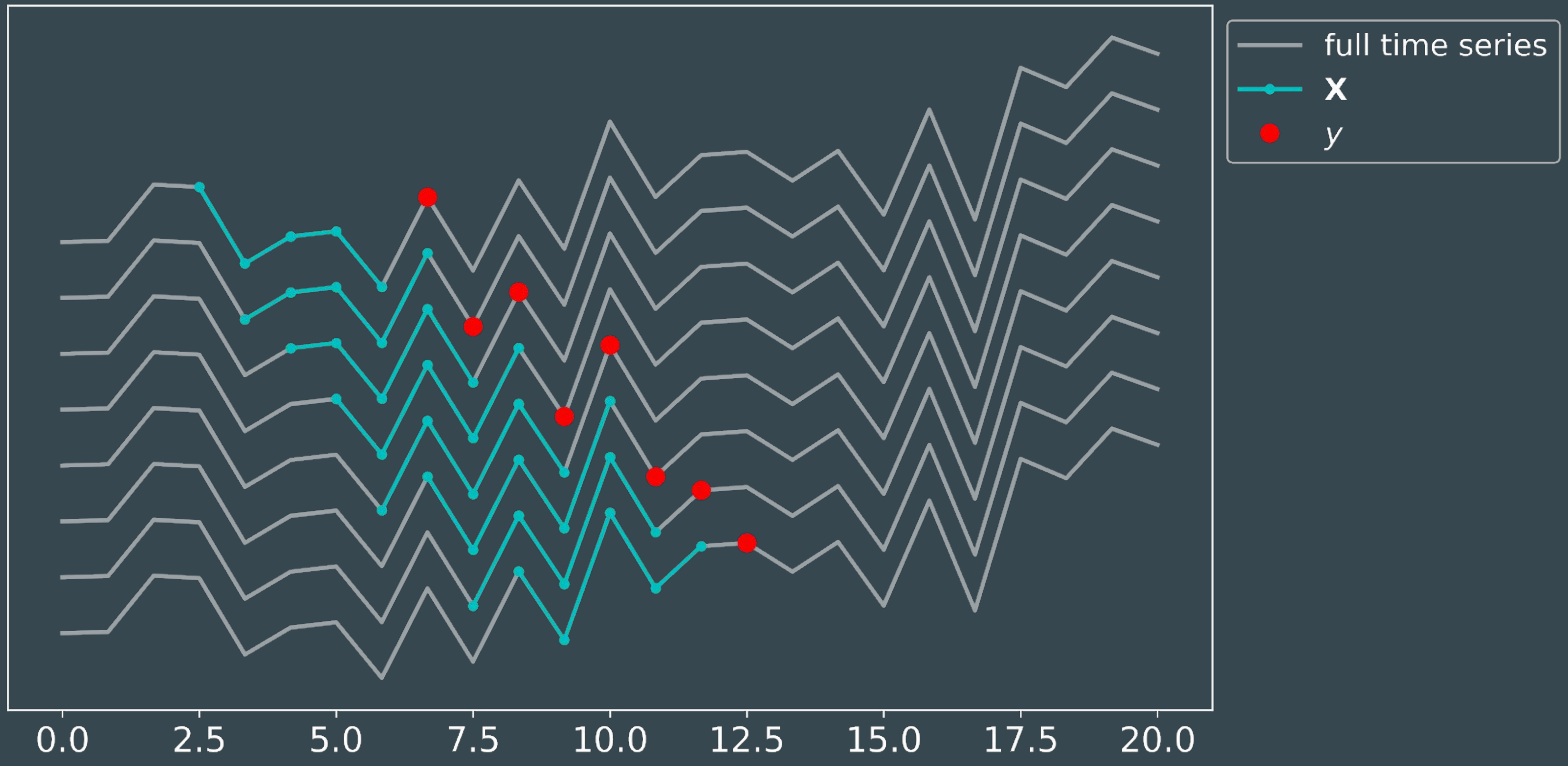
known
future
value to predict
46
44
42
40
11:00
11:15
11:30
11:45
12:00
12:15
12:30
time

Autoregressive / Lag Features

```python
X = []
for idx in range(len(y) - window):
    X.append(y[idx:idx + window])

X = np.array(X)
```

```
y: array([0,
          1,
          2,
          3,
          4,
X: array([[0, 1, 2, 3, 4],   ──→  5,
          [1, 2, 3, 4, 5],   ──→  6,
          [2, 3, 4, 5, 6],   ──→  7,
          [3, 4, 5, 6, 7],   ──→  8,
          [4, 5, 6, 7, 8],   ──→  9,
          [5, 6, 7, 8, 9]])  ──→  10])
```



full time series
X
y
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0

$$\mathbf{X}\beta = \hat{\mathbf{y}}$$

$$\begin{bmatrix} y_0 & y_1 & y_2 & \cdots & y_{w-1} \\ y_1 & y_2 & y_3 & \cdots & y_w \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ y_{t-2-w} & y_{t-1-w} & y_{t-w} & \cdots & y_{t-2} \\ y_{t-1-w} & y_{t-w} & y_{t-w+1} & \cdots & y_{t-1} \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_{w-2} \\ \beta_{w-1} \end{bmatrix} = \begin{bmatrix} \hat{y}_w \\ \hat{y}_{w+1} \\ \vdots \\ \hat{y}_{t-1} \\ \hat{y}_t \end{bmatrix}$$

Modeling

- Not limited to Linear Regression. Use trees, neural nets, whatever you want!
- Not limited to regression. Classification, quantile regression, etc...

Preprocessing and Feature Engineering

- Differencing
- Rolling Mean
- Filters
- Fourier components
- Seasonal lags
- etc...

Adding “Exogenous” Features

```
X = []
for idx in range(len(y) - window):
    X.append(y[idx:idx + window])

X = np.array(X)
X = np.hstack((X, X_features))
```

```
                                                                y: array([0,
                                                                          1,
                                                                          2,
                          Lag Features      Extra Features                3,
                                                                          4,
X: array([[0, 1, 2, 3, 4,   .5, -.1],                      →              5,
          [1, 2, 3, 4, 5, 2.3,   .2],                      →              6,
          [2, 3, 4, 5, 6, -.2,   .4],                      →              7,
          [3, 4, 5, 6, 7,   .9, 1.1],                      →              8,
          [4, 5, 6, 7, 8, 1.2,   .5],                      →              9,
          [5, 6, 7, 8, 9, -.7, -.2]])                      →              10])
```

Adding “Exogenous” Features

```python
X = []
for idx in range(len(y) - window):
    X.append(y[idx:idx + window])

X = np.array(X)
X = np.hstack((X, X_features))
```

Complicated to construct!

Lag Features   Extra Features

```
X: array([[0, 1, 2, 3, 4,  .5, -.1],        5,
          [1, 2, 3, 4, 5, 2.3,  .2],        6,
          [2, 3, 4, 5, 6, -.2,  .4],        7,
          [3, 4, 5, 6, 7,  .9, 1.1],        8,
          [4, 5, 6, 7, 8, 1.2,  .5],        9,
          [5, 6, 7, 8, 9, -.7, -.2]])       10])
```

```
y: array([0,
          1,
          2,
          3,
          4,
          5,
          6,
          7,
          8,
          9,
          10])
```

Recap

- Take your time series **y**.
- Treat each point in **y** as a point that you want to predict.
- Construct **X** from any data you want that comes *prior* to the point in **y** that you want to predict.

$$\mathbf{X}_t = \mathbf{y}_{t'<t}$$

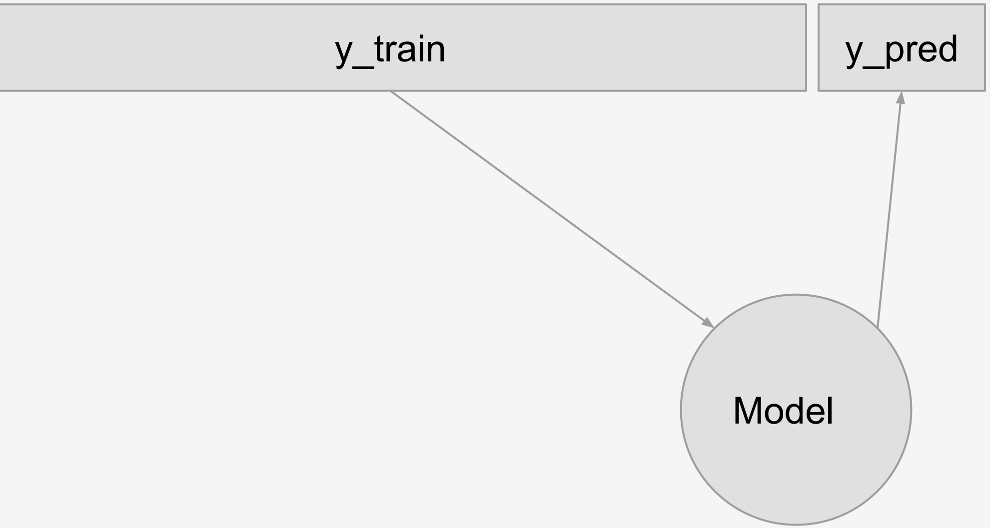
- Fit a regression model on **X** and **y**.
- For each point in **y**, use model to predict the next point.

$$\hat{y}_t = f(\mathbf{X}_t)$$

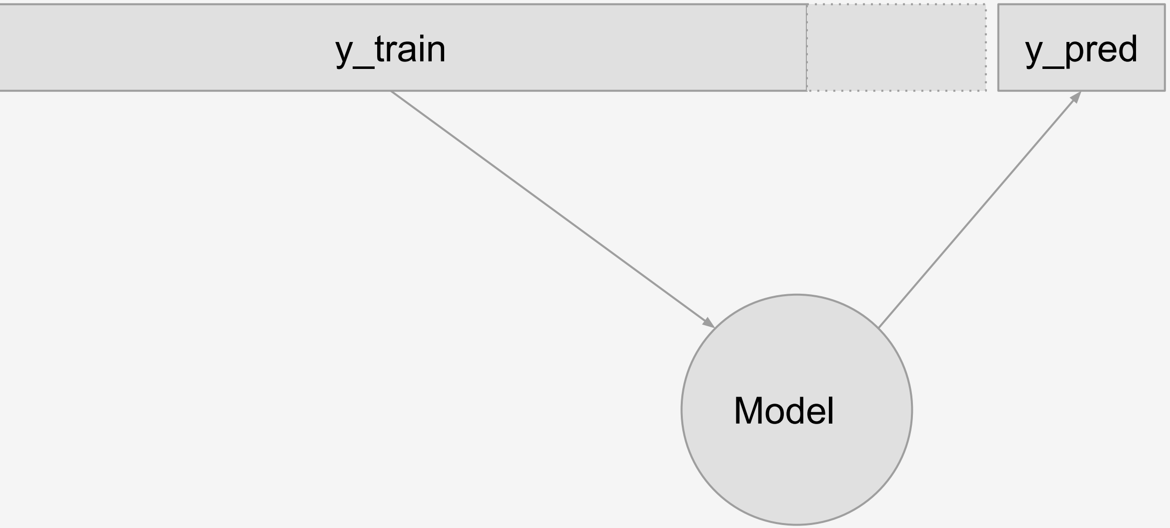
$$\hat{y}_t = f(\mathbf{y}_{t'<t} \quad)$$

Forecasting

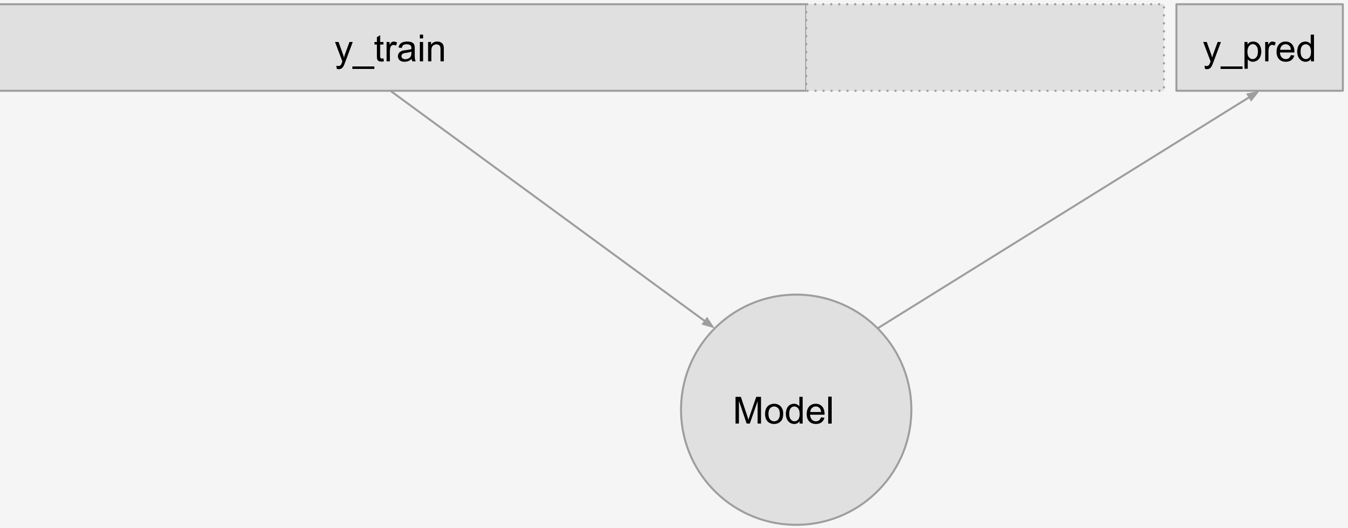
Recursive Forecasting



Recursive Forecasting



Recursive Forecasting



Recursive Forecasting

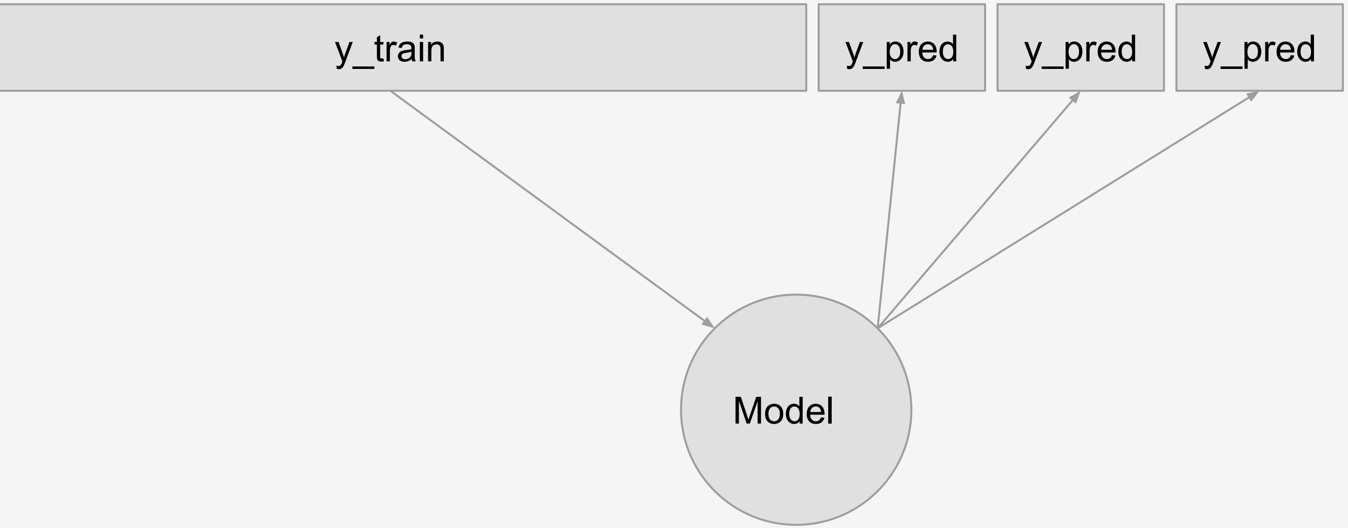
```python
def recursive_forecast(model, input_data, num_points_in_future):
    for point in range(num_points_in_future):
        prediction = model.predict(input_data)
        # Append prediction to the input data
        input_data = np.hstack((input_data, prediction))

    return prediction
```

Optimize for next step

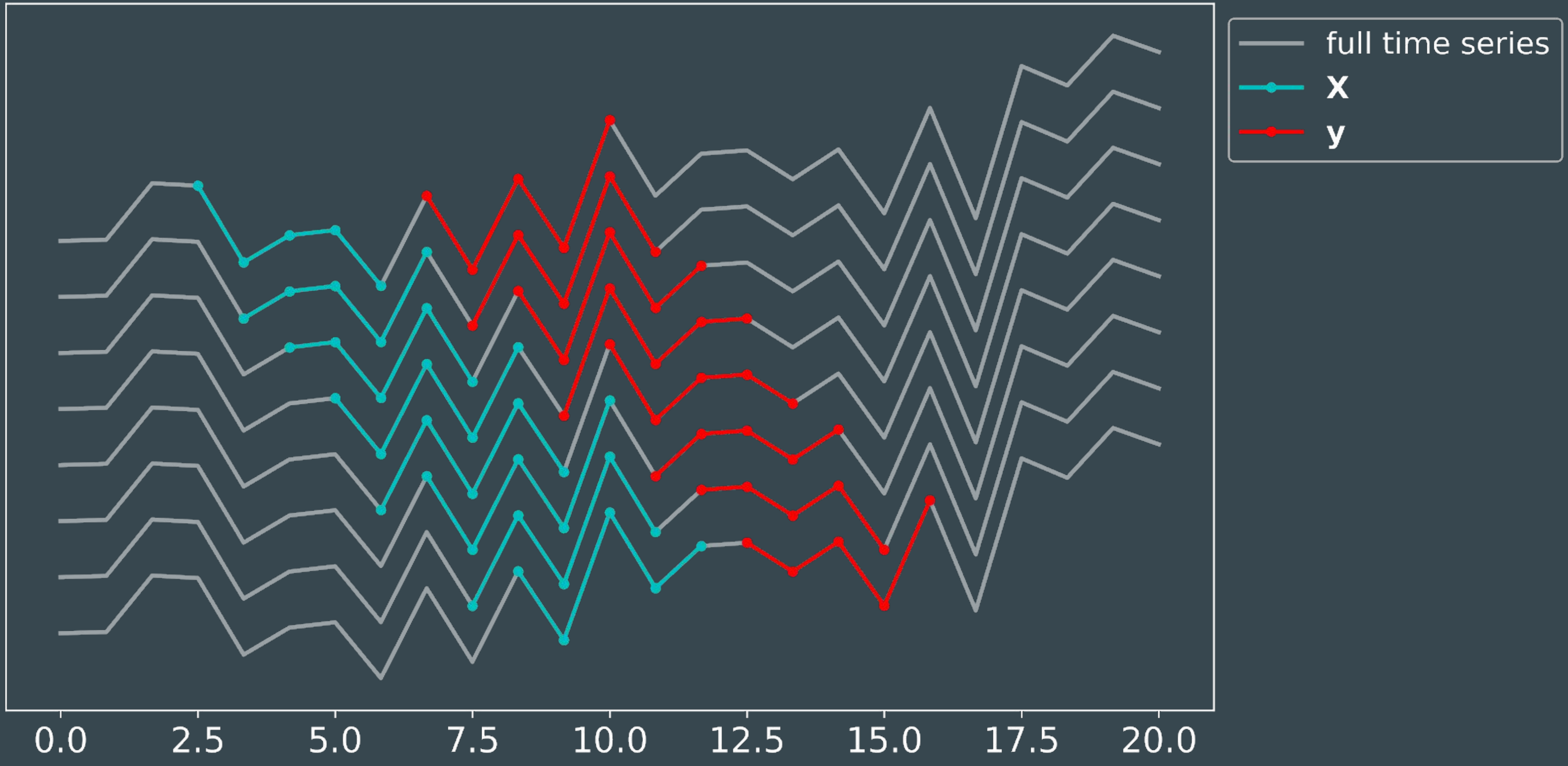
Pray recursive steps work

Horizon Forecasting



More info: “Machine learning strategies for multi-step-ahead time series forecasting”, Souhaib Ben Taieb https://souhaib-bentaieb.com/papers/2014_phd.pdf

Horizon Forecasting



```
y: array([ 0.,
           1.,
           2.,
           3.,
           4.,
           5.,
           6.,
           7.,
           8.,
           9.,
          10.])
```

```
y_horizon: array([[ 0.,  1.,  2.],
                  [ 1.,  2.,  3.],
                  [ 2.,  3.,  4.],
                  [ 3.,  4.,  5.],
                  [ 4.,  5.,  6.],
                  [ 5.,  6.,  7.],
                  [ 6.,  7.,  8.],
                  [ 7.,  8.,  9.],
                  [ 8.,  9., 10.],
                  [nan, nan, nan],
                  [nan, nan, nan]])
```

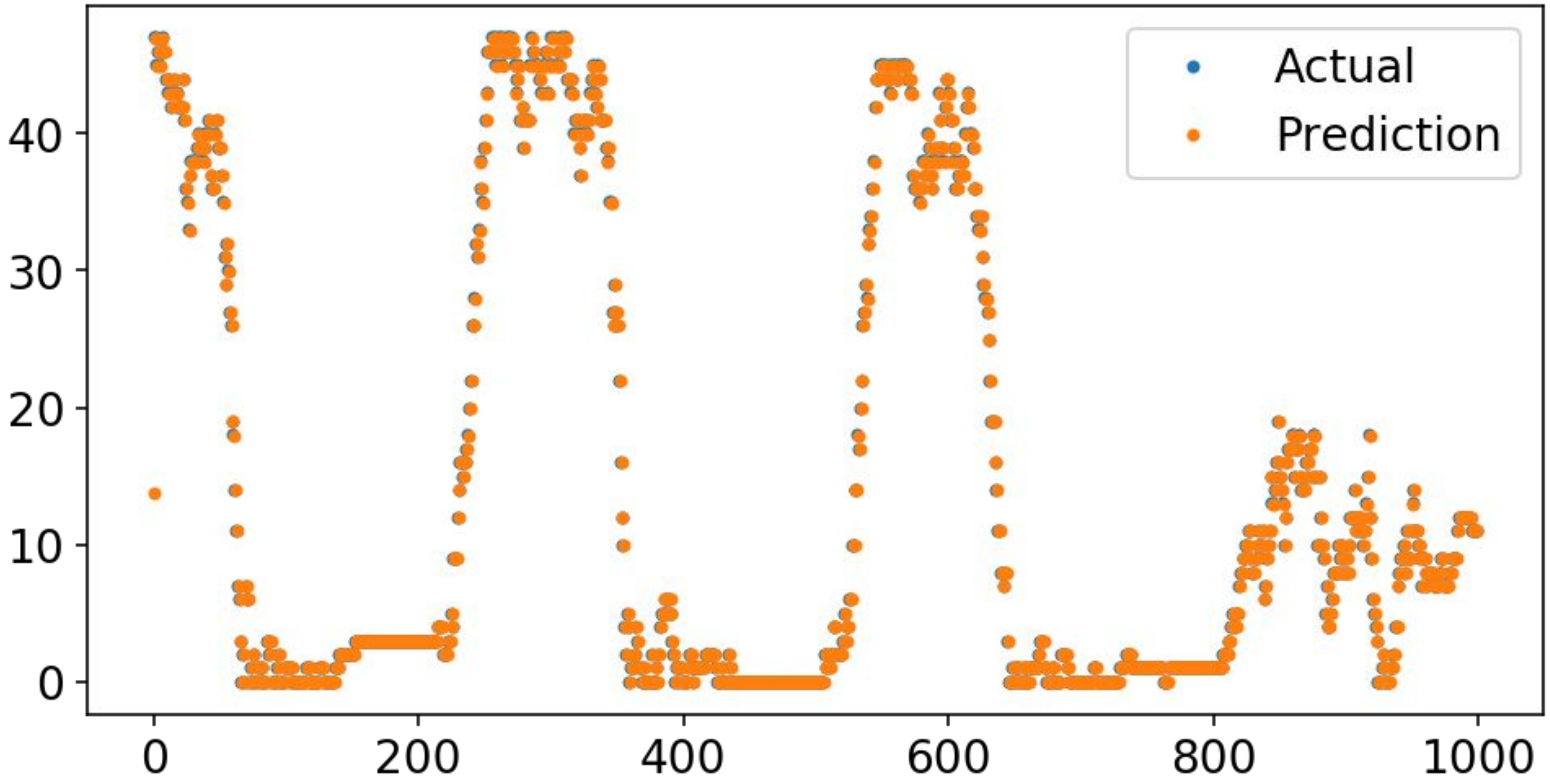
Predicting multiple targets

- sklearn.multioutput.MultiOutputRegressor
- Train an individual model for each target.
- Pros:
  - Very simple
  - Works with any model
- Cons
  - Resource intensive
  - No sharing of knowledge

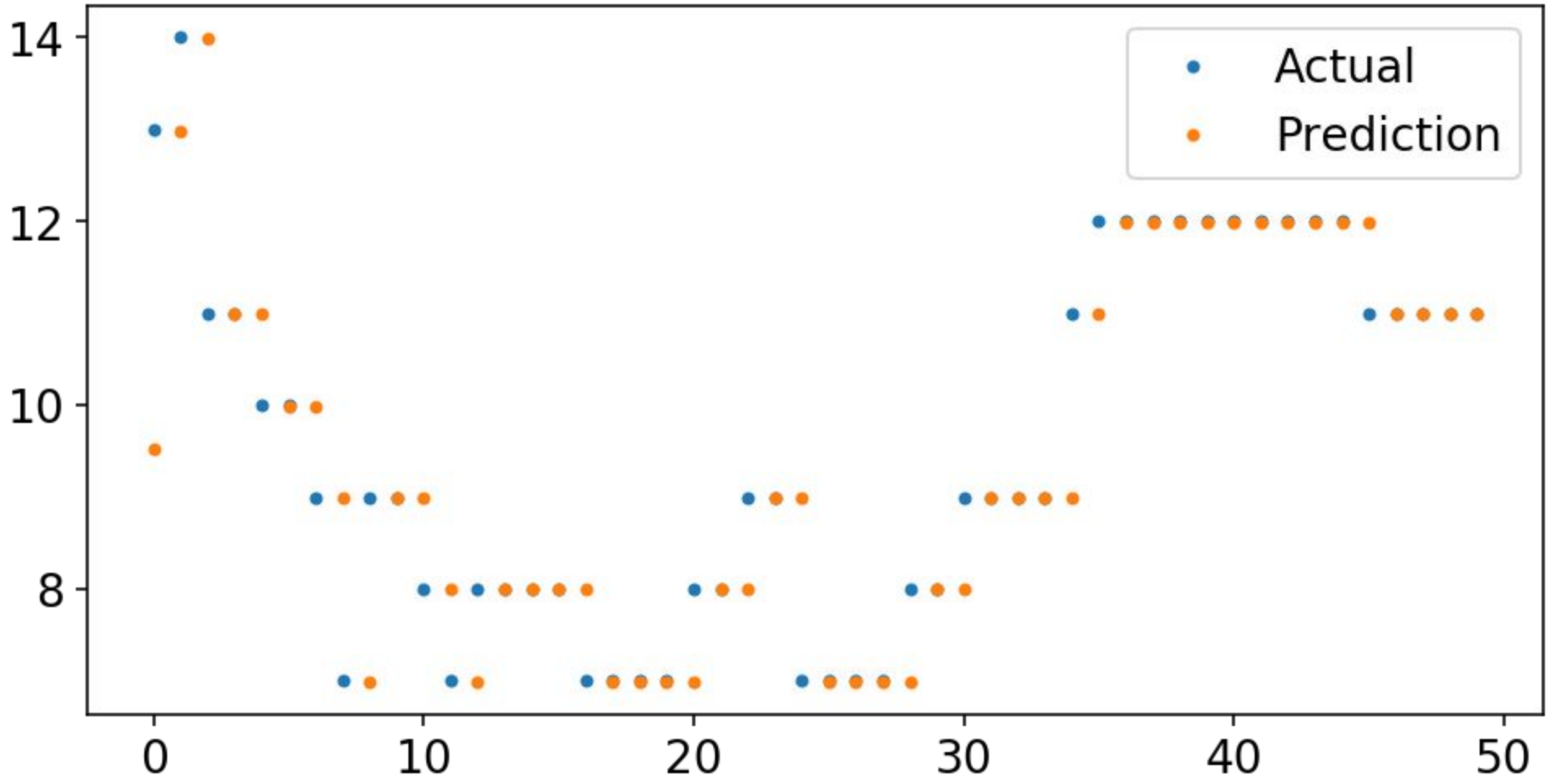
Predicting multiple targets

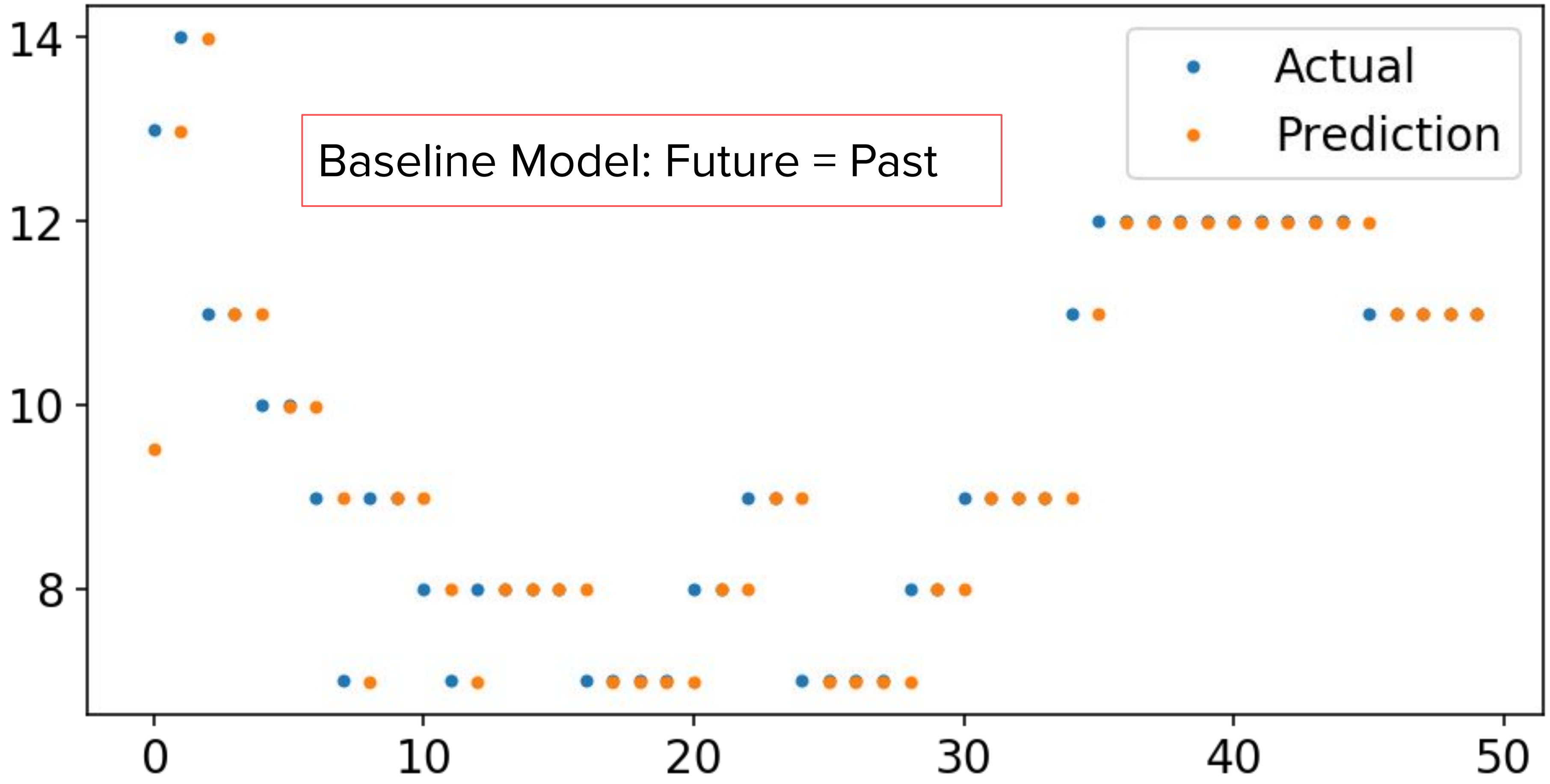
- Deep learning model with multiple outputs.
- Pros:
  - Potentially less resource intensive
  - Direct optimization
  - Sharing of knowledge
- Cons:
  - All the caveats of deep learning
  - Limited to the horizon

Evaluation

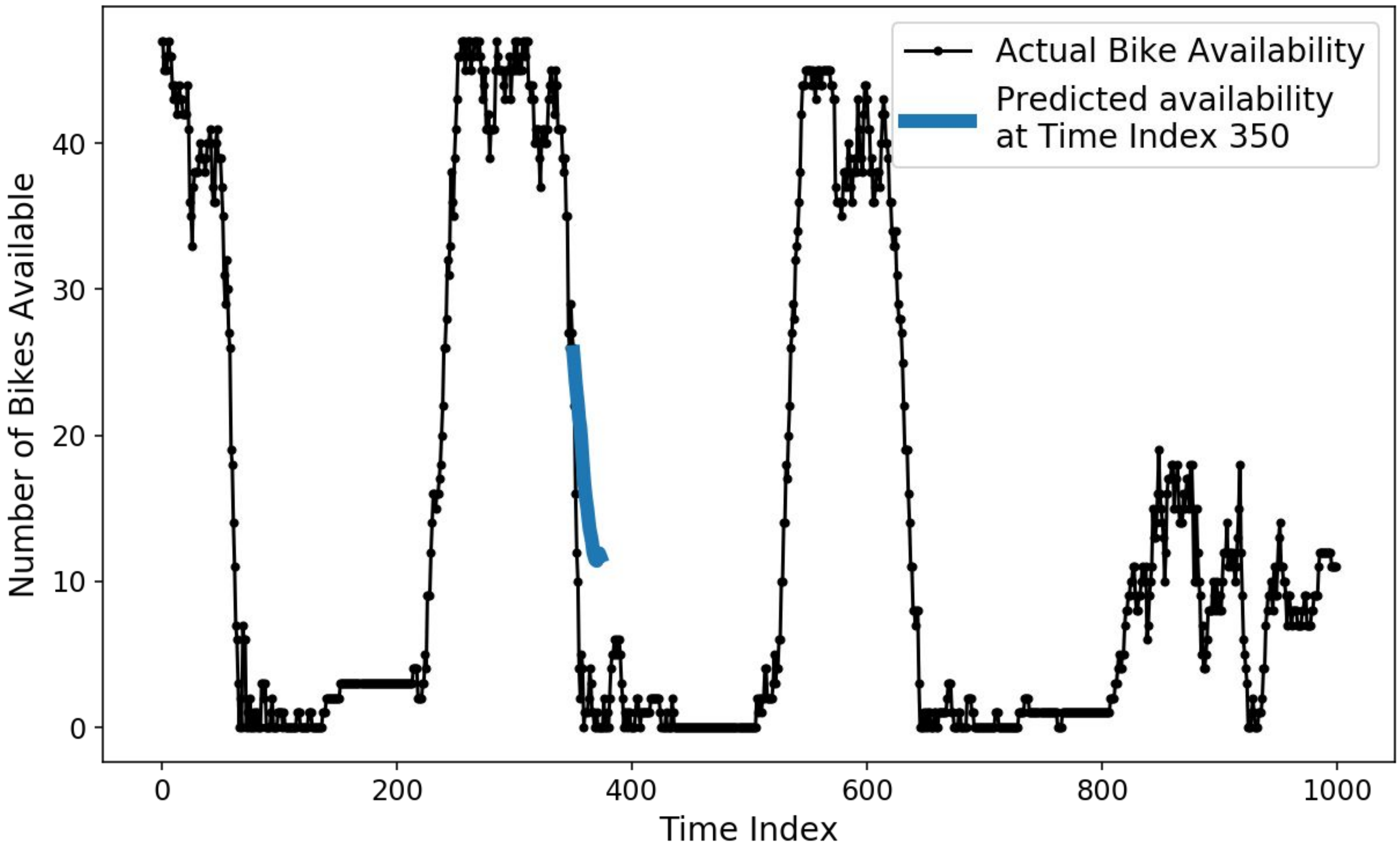


Actual
Prediction
0
10
20
30
40
0
200
400
600
800
1000

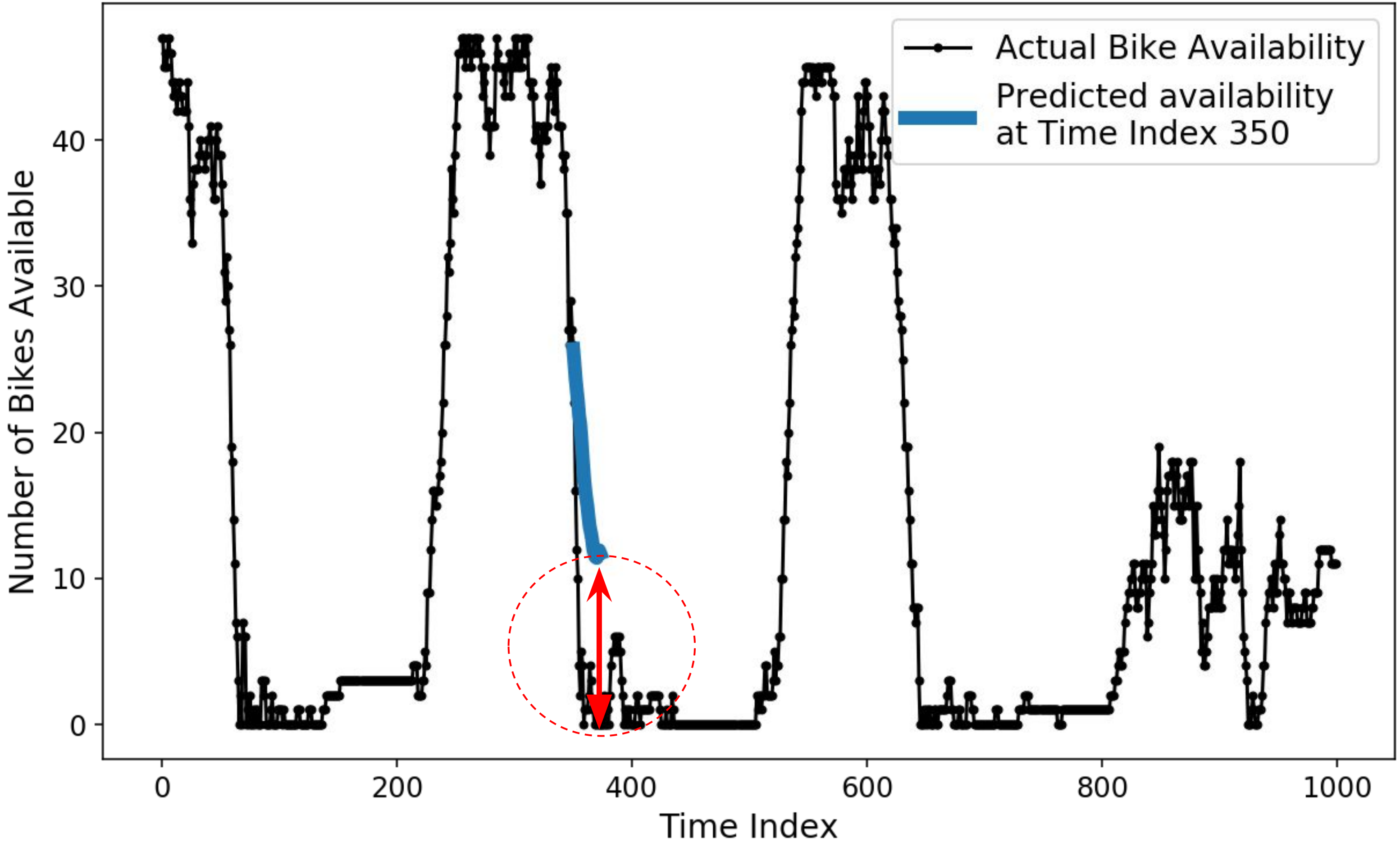

Actual
Prediction
14
12
10
8
0
10
20
30
40
50


Baseline Model: Future = Past
Actual
Prediction
14
12
10
8
0
10
20
30
40
50

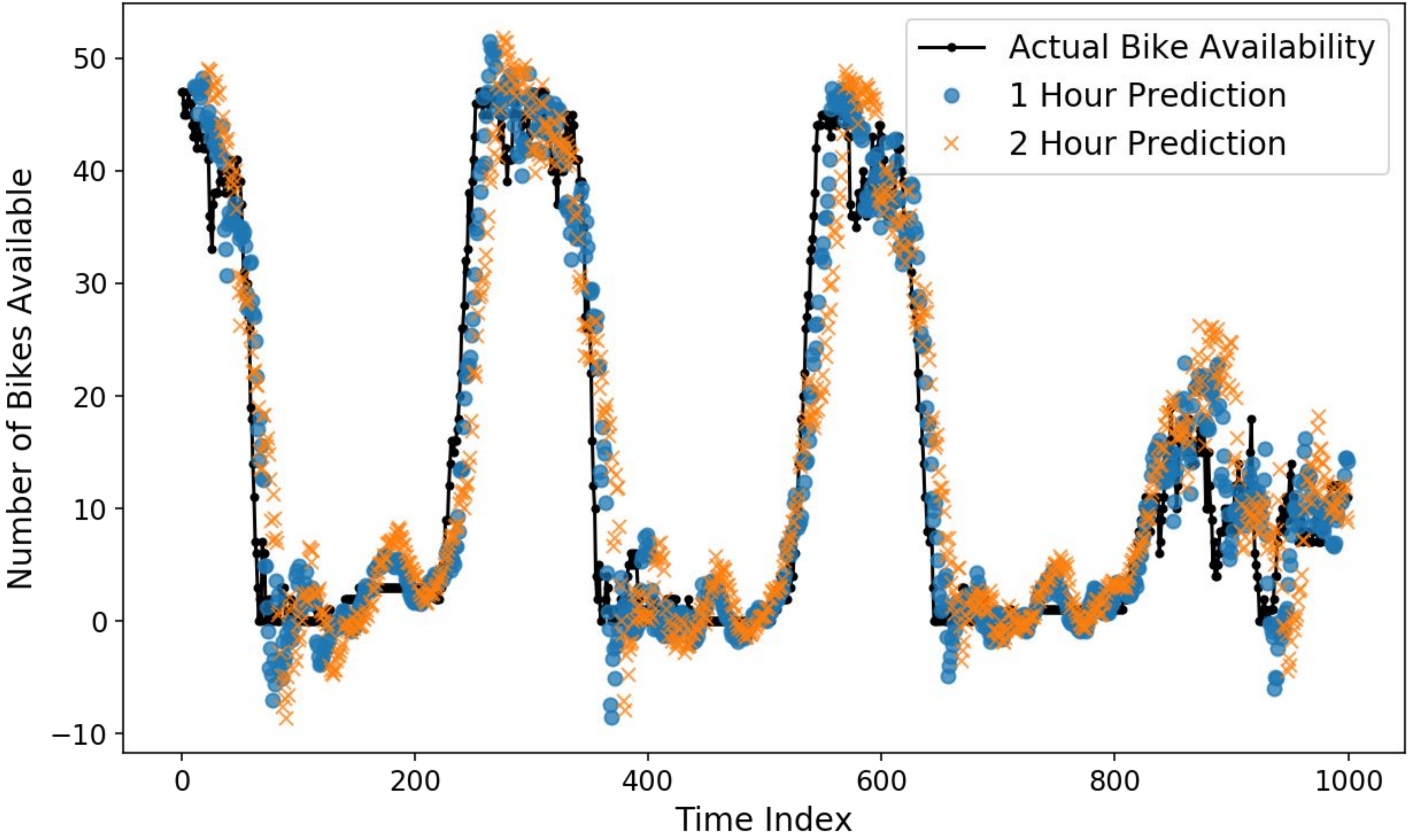
Forecasting Views



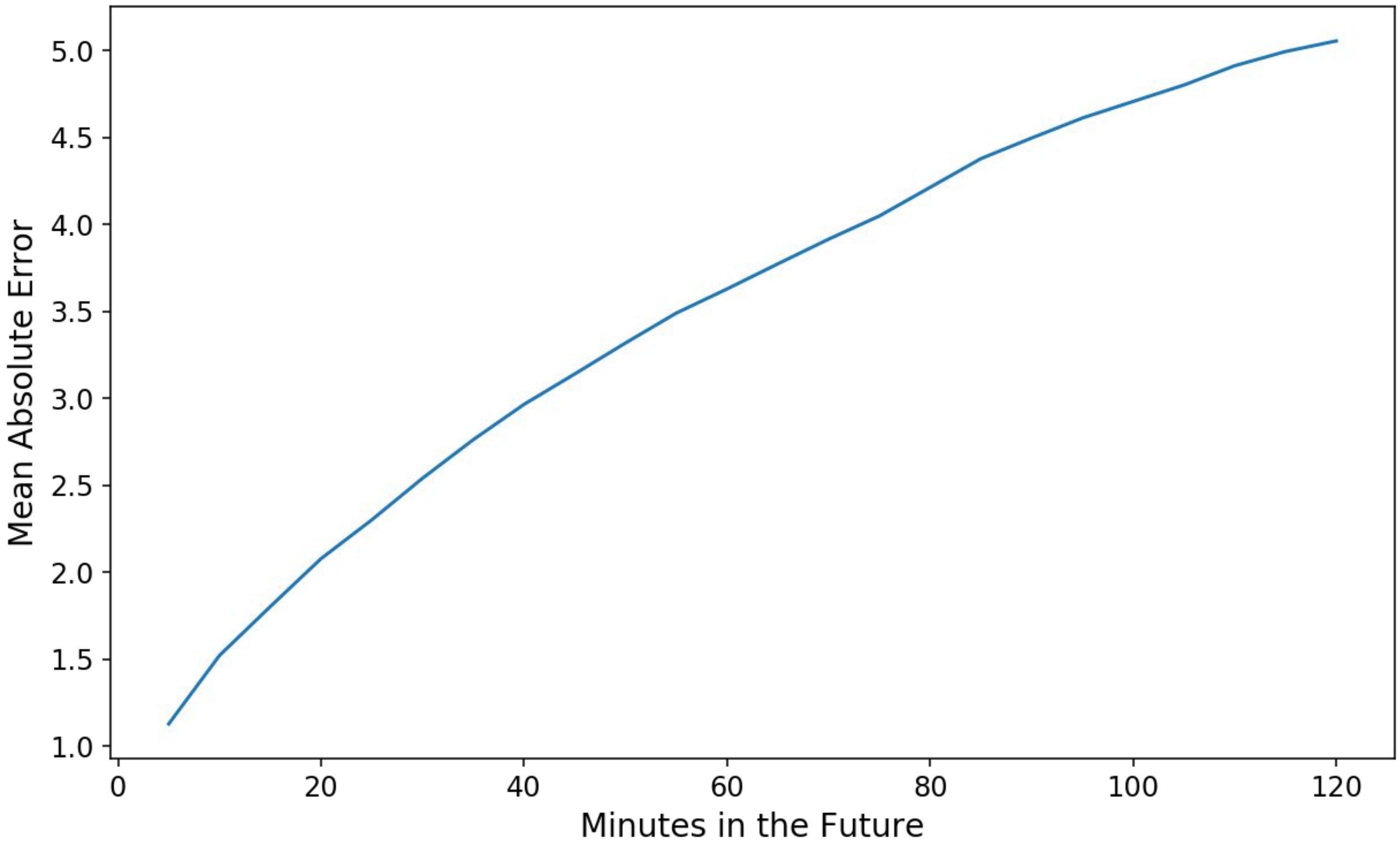
Forecasting Views



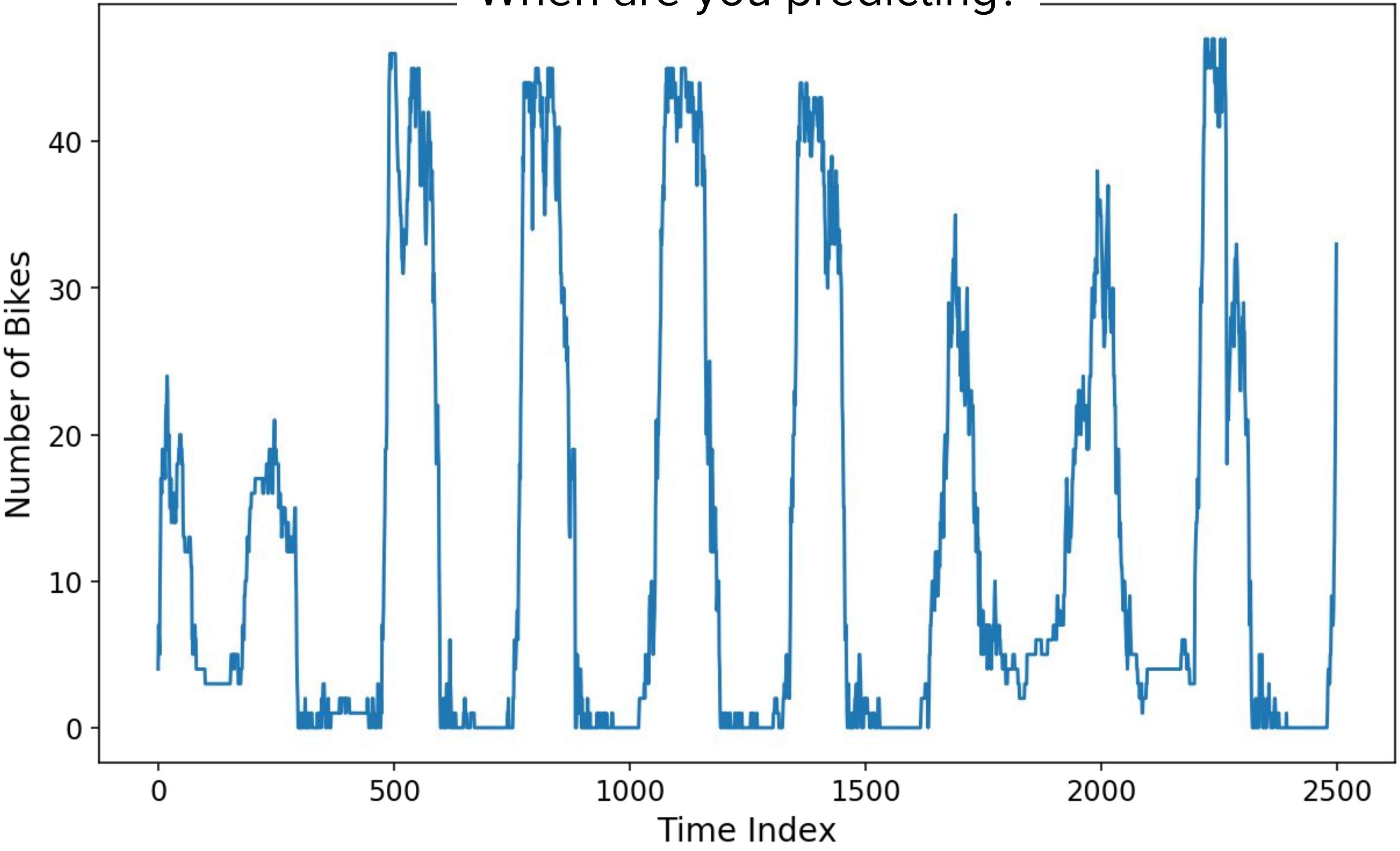
Forecasting Views



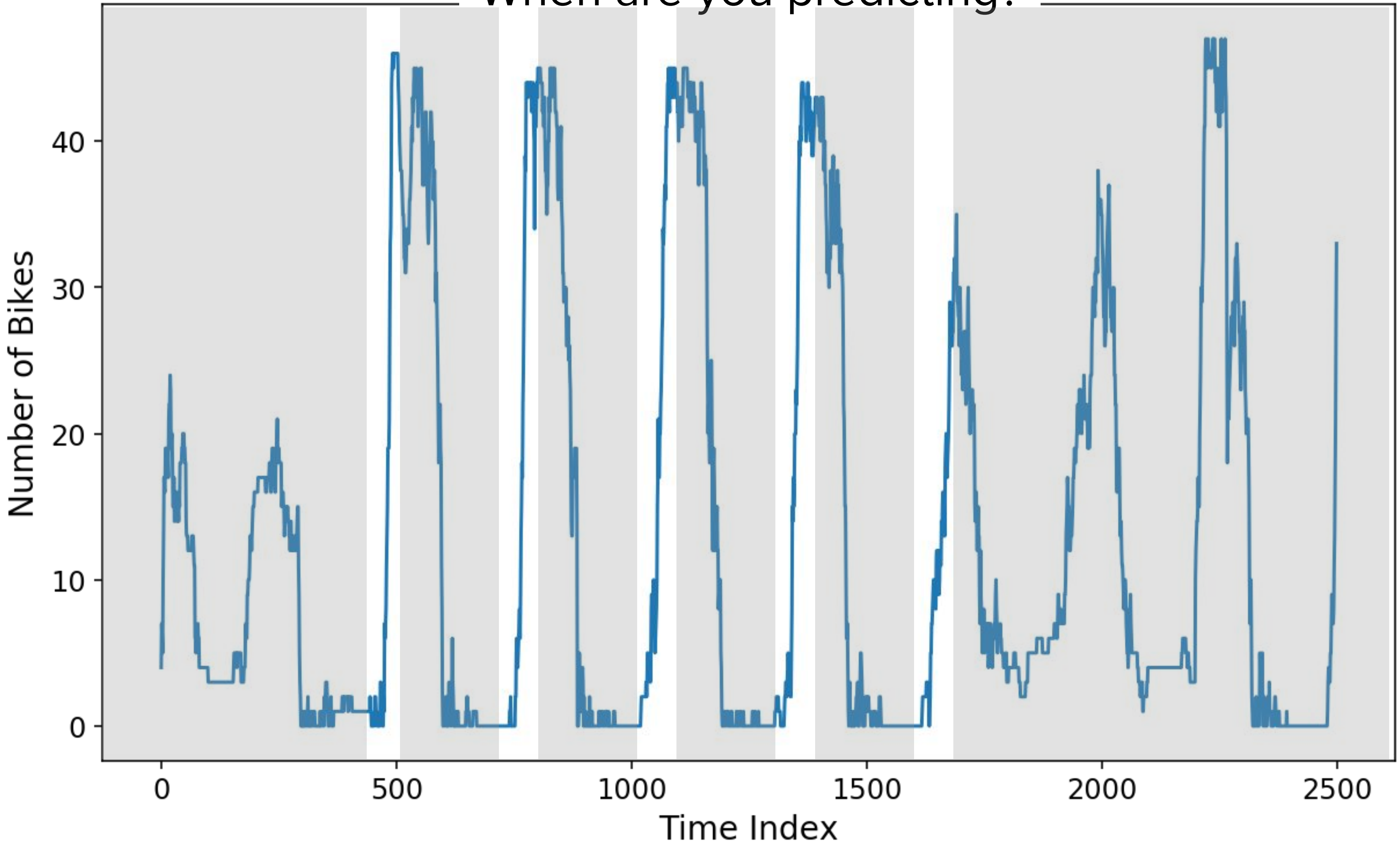
Metrics Aggregation



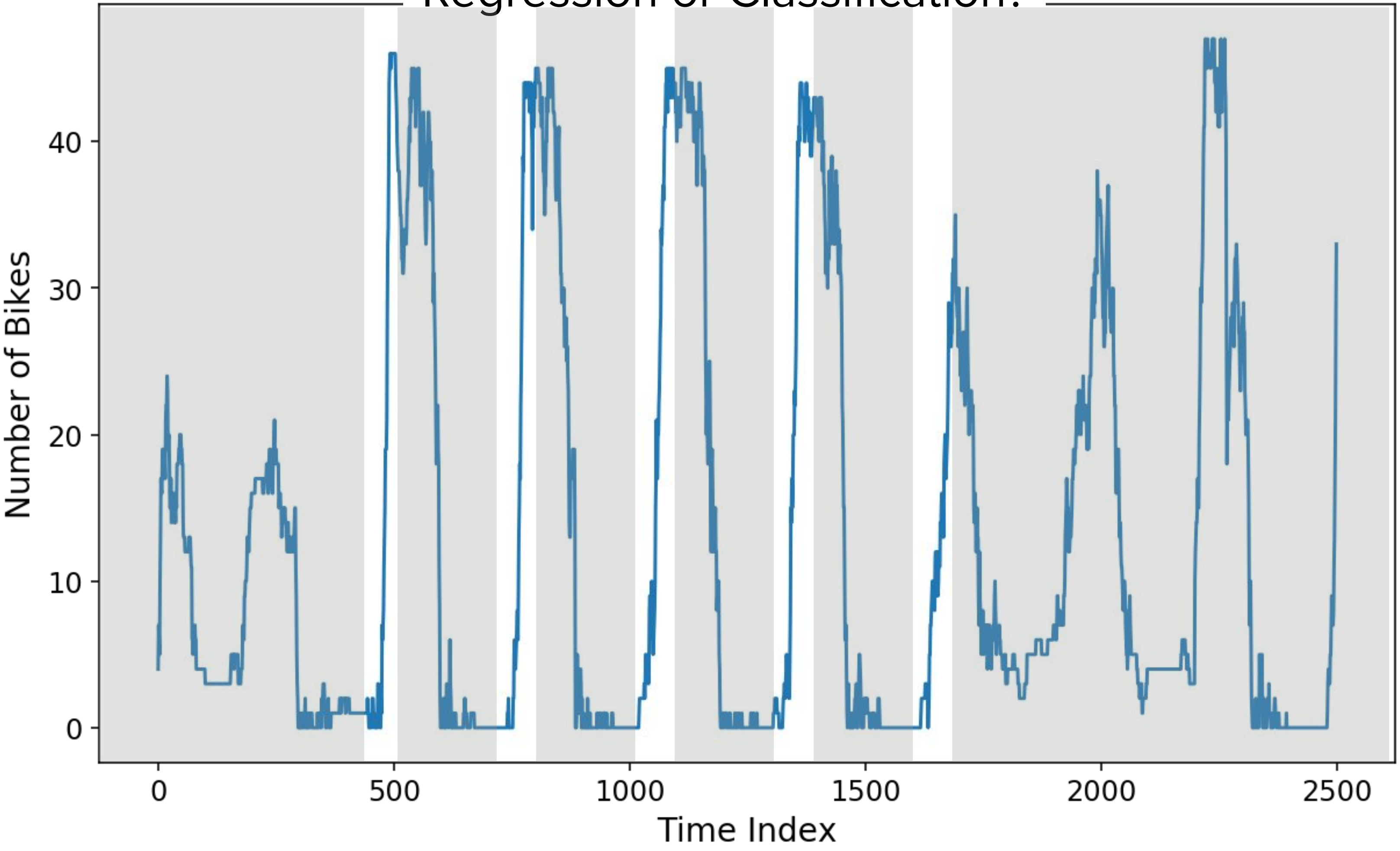
When are you predicting?



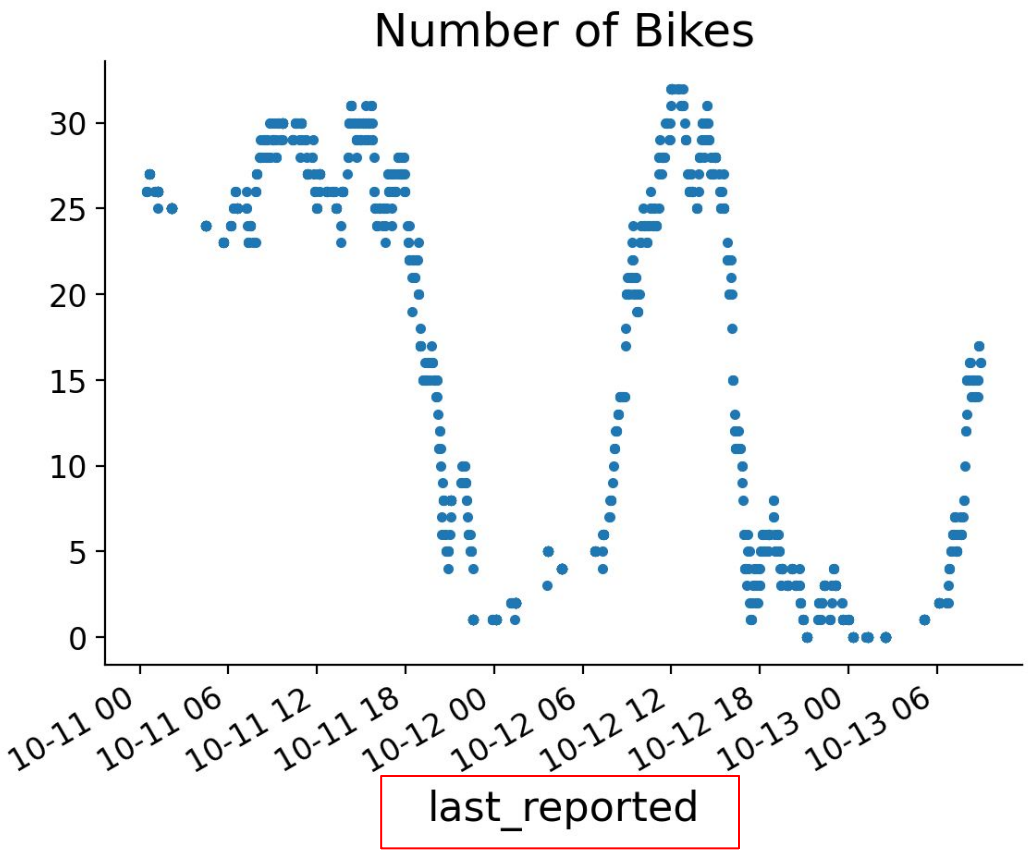
When are you predicting?



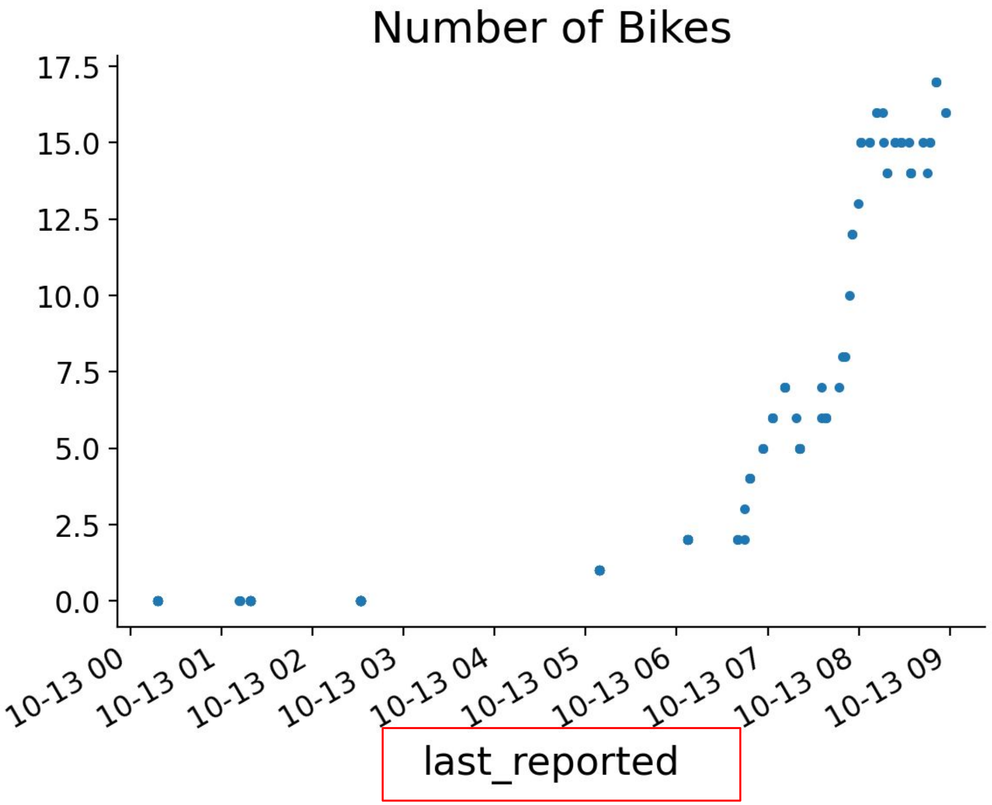
Regression or Classification?



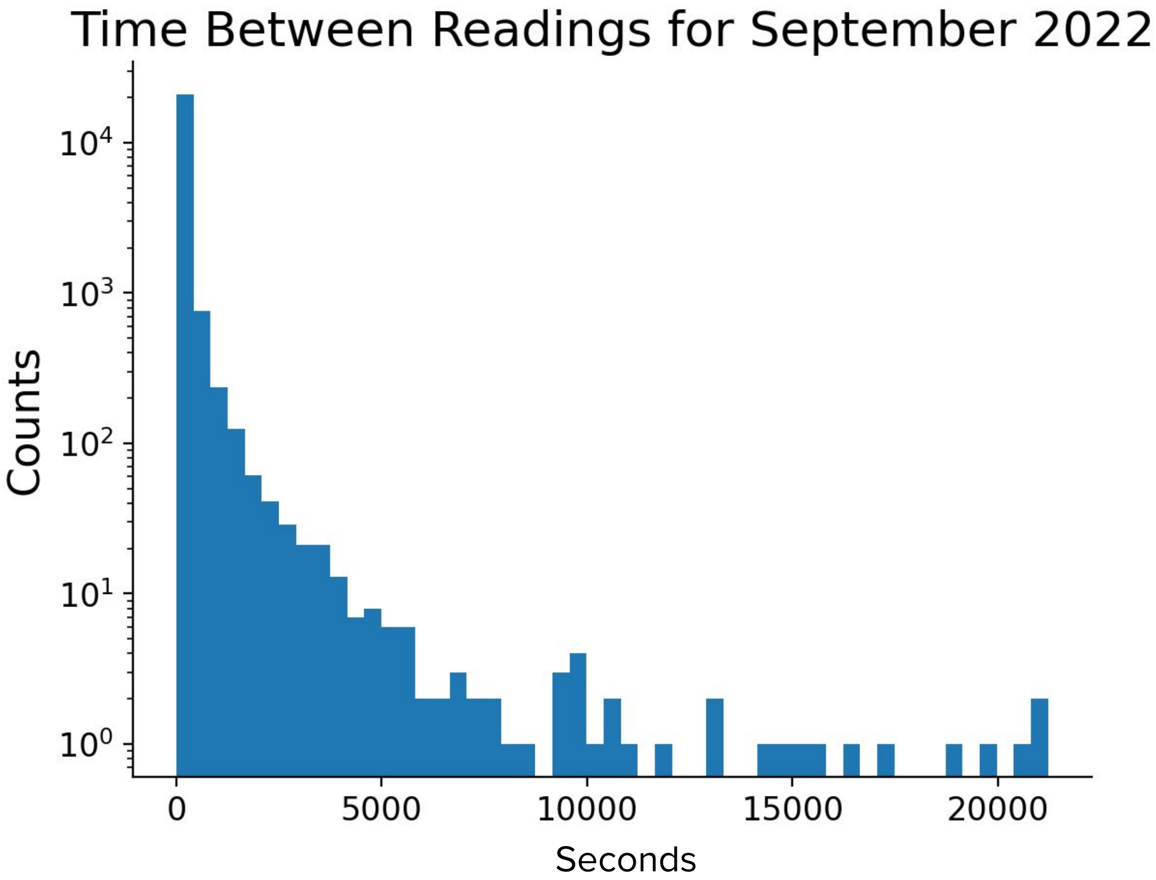
What do you know when you predict?



What do you know when you predict?

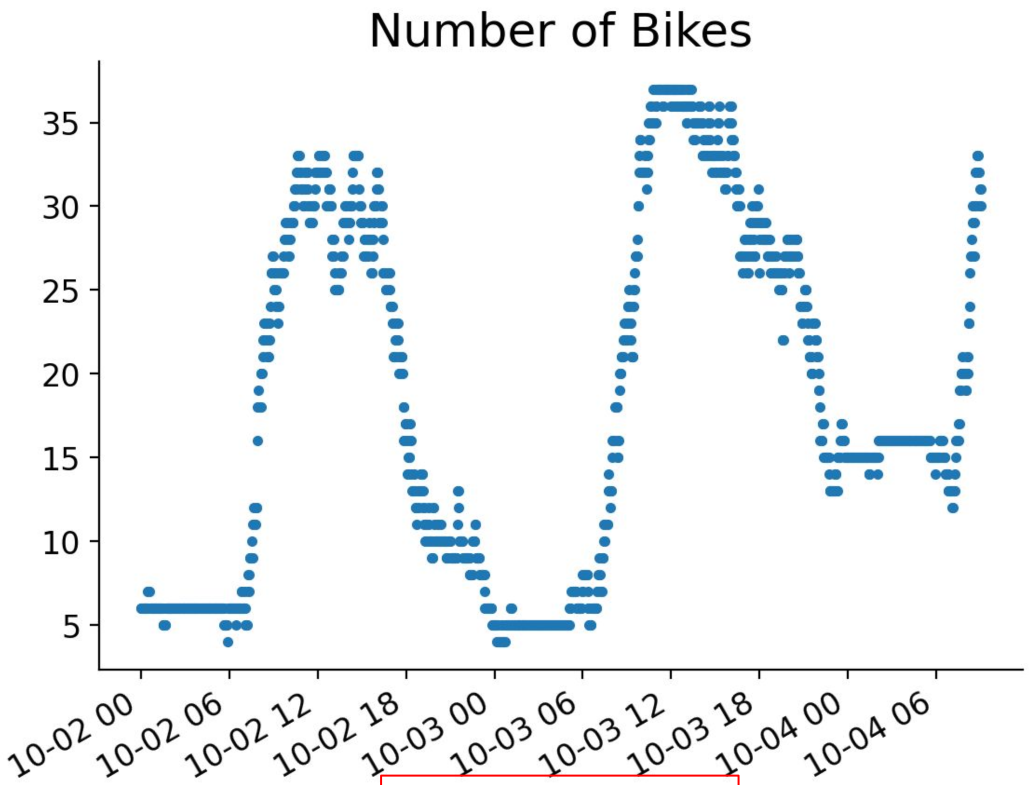


What do you know when you predict?



2023

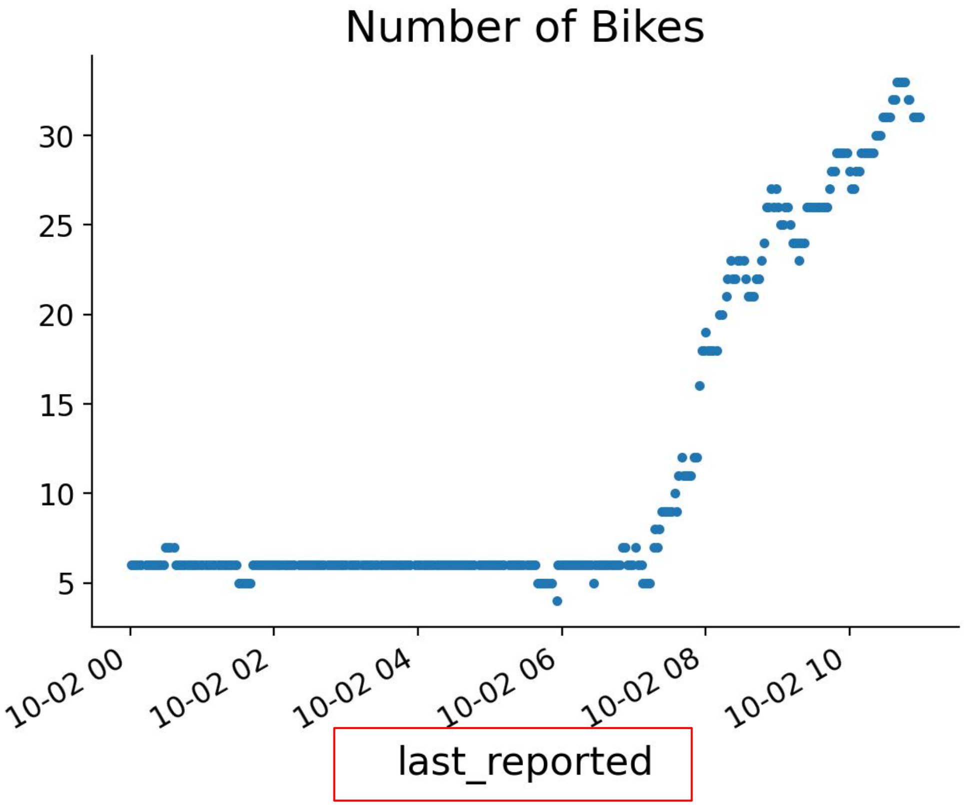
What do you know when you predict?



last_reported

2023

What do you know when you predict?



What do you know when you predict?