Lecture 4: Time Series, VAR Models

What is a VAR model?

VAR models are another way that we can model time series data.

- VAR: **V**ector **A**uto**R**egressive model
- Makes use of multiple correlated time series
- Based on SUR (Seemingly Unrelated Regressions) models

Quick Overview of SUR models

Consider j regression equations:

$$Y_j = X_j \beta_j + \epsilon_j$$

where Y_j, and ϵ_j are $N \times 1$, X_j is $N \times K$, and β_j is $K \times 1$

Quick Overview of SUR models

Consider j regression equations:

$$Y_j = X_j\beta_j + \epsilon_j$$

Imagine that the outcomes Y_{ij} are correlated such that

$$Cov(\epsilon_{ij}, \epsilon_{ik}) = \sigma_{ij}$$

and

$$Cov(\epsilon_{ij}, \epsilon_{i'k}) = 0, \;\; \forall\, i \neq i'$$

Quick Overview of SUR models

We can stack our regressions to get a single system of equations:

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix} = \begin{bmatrix} X_1 & \mathbf{0} & \ldots & \mathbf{0} \\ \mathbf{0} & X_1 & \ldots & \mathbf{0} \\ \vdots & \vdots & \ddots & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & X_1 \end{bmatrix} \begin{bmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_N \end{bmatrix} + \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \vdots \\ \epsilon_N \end{bmatrix}$$

Quick Overview of SUR models

Then the FGLS estimator of the system is

$$\hat{\beta}_{FGLS} = \left(X' \left(\hat{\Sigma} \otimes I_N\right) X\right)^{-1} X' \left(\hat{\Sigma} \otimes I_N\right) Y$$

Where $\hat{\Sigma} = [\hat{\sigma}_{ij}]$, and

$$\hat{\sigma}_{ij} = \frac{1}{N} (y_i - X_i \beta_i)' (y_j - X_j \beta_j)$$

Quick Overview of SUR models

So what does all this mean?

- SUR models relax the assumption that each regression is uncorrelated with the others
- Allows us to use some of our dependent variables in the X matrices for other regressions
    - This will in turn allow us to model simultaneous time series, where the errors across the series will certainly be correlated

VAR Models

Just a SUR model where the multiple dependent variables are time series

- We can include lags of dependent variables as part of the X matrix of covariates
- VAR models are built to capture the interactions between variables as time passes

VAR Models

We can write the VAR model

$$\mathbf{y}_t = \mu + \mathbf{\Gamma}_1 \mathbf{y}_{t-1} + \ldots + \mathbf{\Gamma}_p \mathbf{y}_{t-p} + \epsilon_t$$

Representing m equations relating lagged dependent variables to the dependent variables in time t.

Implementing a VAR Model

```
# Getting started by importing modules and data
import pandas as pd, numpy as np
from statsmodels.tsa.api import VAR
import statsmodels.tsa.stattools as st
from bokeh.plotting import figure, show
from datetime import datetime

# Collect data, set index
data = pd.read_csv("pollutionBeijing.csv")
# Difference and log dep. var.
format = '%Y-%m-%d %H:%M:%S'
data['datetime'] = pd.to_datetime(data['datetime'],
        format=format)
data.set_index(pd.DatetimeIndex(data['datetime']),
        inplace=True)
```

Implementing a VAR Model

```
# Select variables for VAR model
varData = data[['pm2.5','TEMP','PRES',
                        'Iws']].dropna()[:-50]
```

- **REMEMBER: We need ALL stationary variables**
    - `st.adfuller` is the test to use on each variable
- We also need the terminal values of each variable PRIOR to differencing (you'll see why later)

Implementing a VAR Model

```
model = VAR(varData) # define the model and data
model.select_order() # uses information criteria to
                     # select the model order
modelFit = model.fit(30)
                # order chosen based on BIC criterion
```

- Diagnostics like those from the ARIMA(p,d,q) models are not available to determine our model order
- Use information criteria to find the optimal order of the VAR model

Forecasting with a VAR Model

```
# Forecasting
pred = reg.forecast(varData['2013-01-04':].values,
                            steps = 50)
```

- When using a trained VAR model, we must include enough observations from our dataset in order to provide the expected number of lags to the model
- We have to begin our data **at least** k observations prior to our end-point, where k is the order of our model

Forecasting with a VAR Model

- Recall that our forecast is not always what we will observe in the real world
- If we have **differenced** our data, we need to undo that differencing
- THEN we apply our transformed forecasts to the most recent actual evaluation

Forecasting with a VAR Model

```
def dediff(todaysVal, forecast):
    future = forecast
    for i in range(np.shape(forecast)[0]):
        if (i==0):
            future[i] = todaysVal + forecast[0]
        else:
            future[i] = future[i-1] + forecast[i]

    return future
```

- Use a function like this one to generate predicted values that can be applied to the original series (only if your data had to be differenced)

Forecasting with a VAR Model

```
nextPer = pd.DataFrame(pred,
                pd.DatetimeIndex(
                start=datetime(2014,12,29,22),
                freq='H', periods=50),
                columns=varData.columns)
```

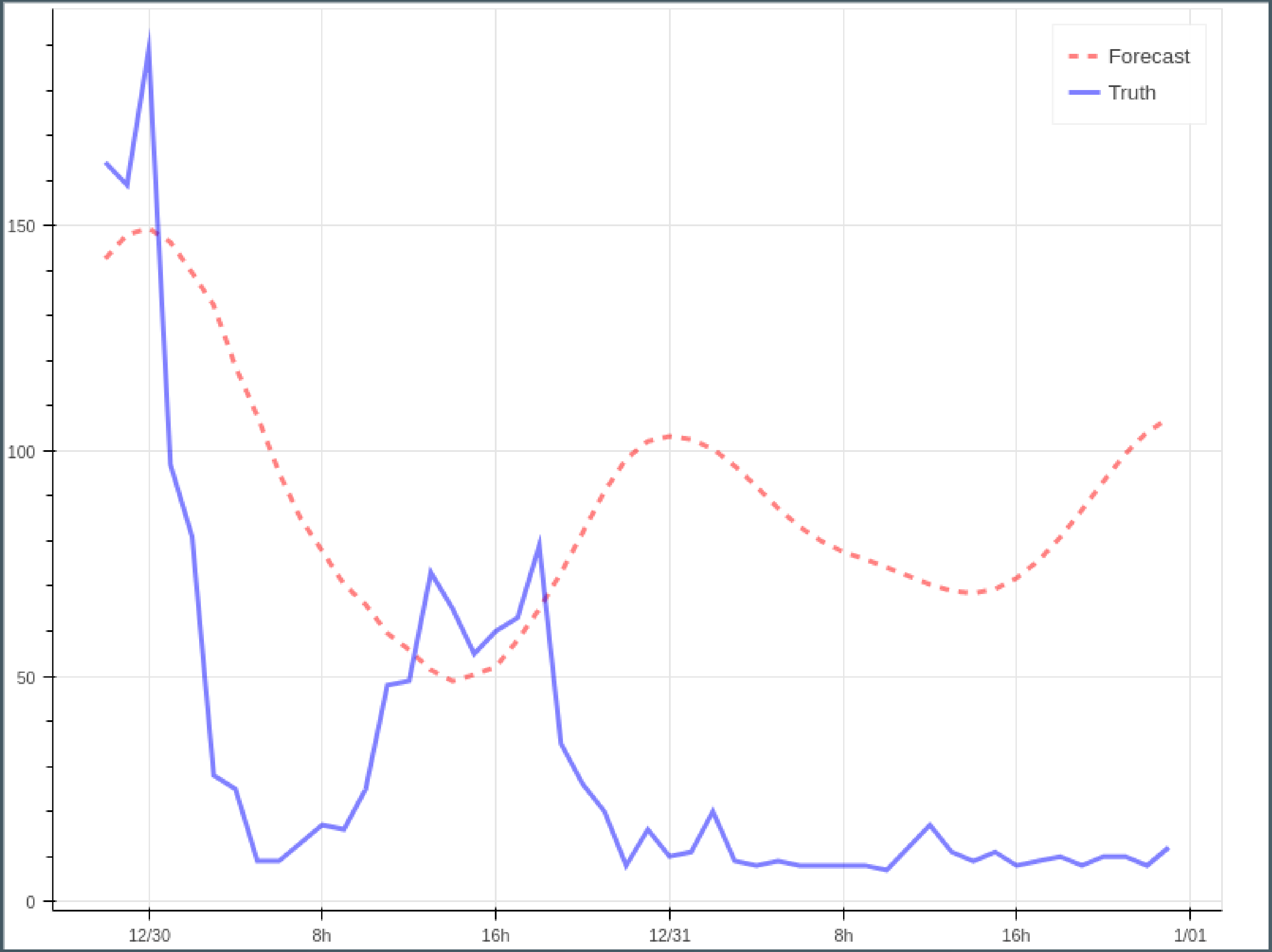
Here, we transform our predictions into datetime formatted values, so that we can more easily plot them.

Forecasting with a VAR Model

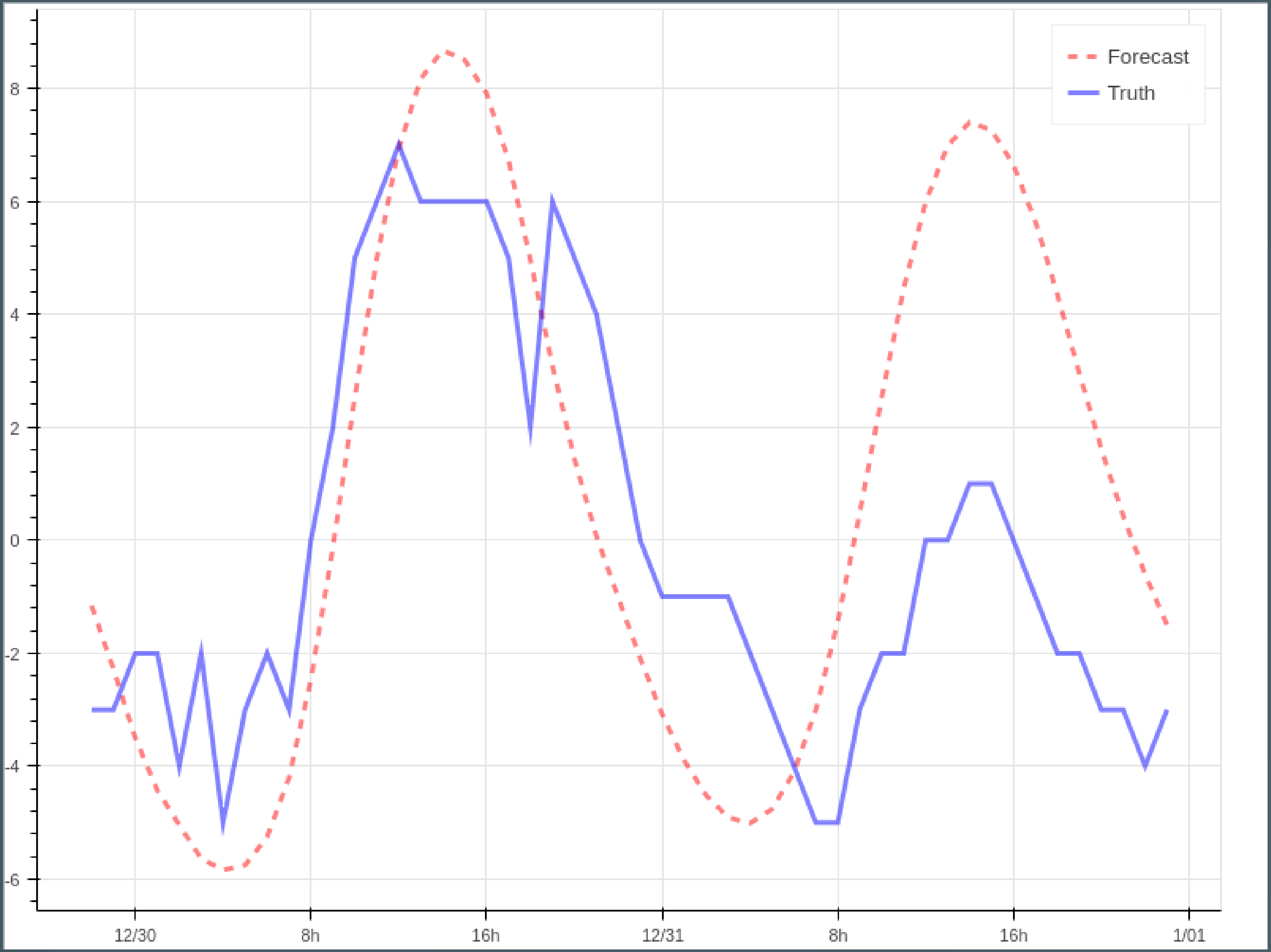
```
#Pm 2.5 Plot
p = figure(plot_width=800, plot_height=600,
        x_axis_type='datetime')
p.line(nextPer.index.values, nextPer['pm2.5'],
        color = 'red', line_width=3,
        line_dash='dashed', alpha=0.5,
        legend='Forecast')
p.line(test.index.values, test['pm2.5'],
        color = 'blue', line_width=3,
        alpha=0.5, legend='Truth')
show(p)
```

Plotting prediction vs truth

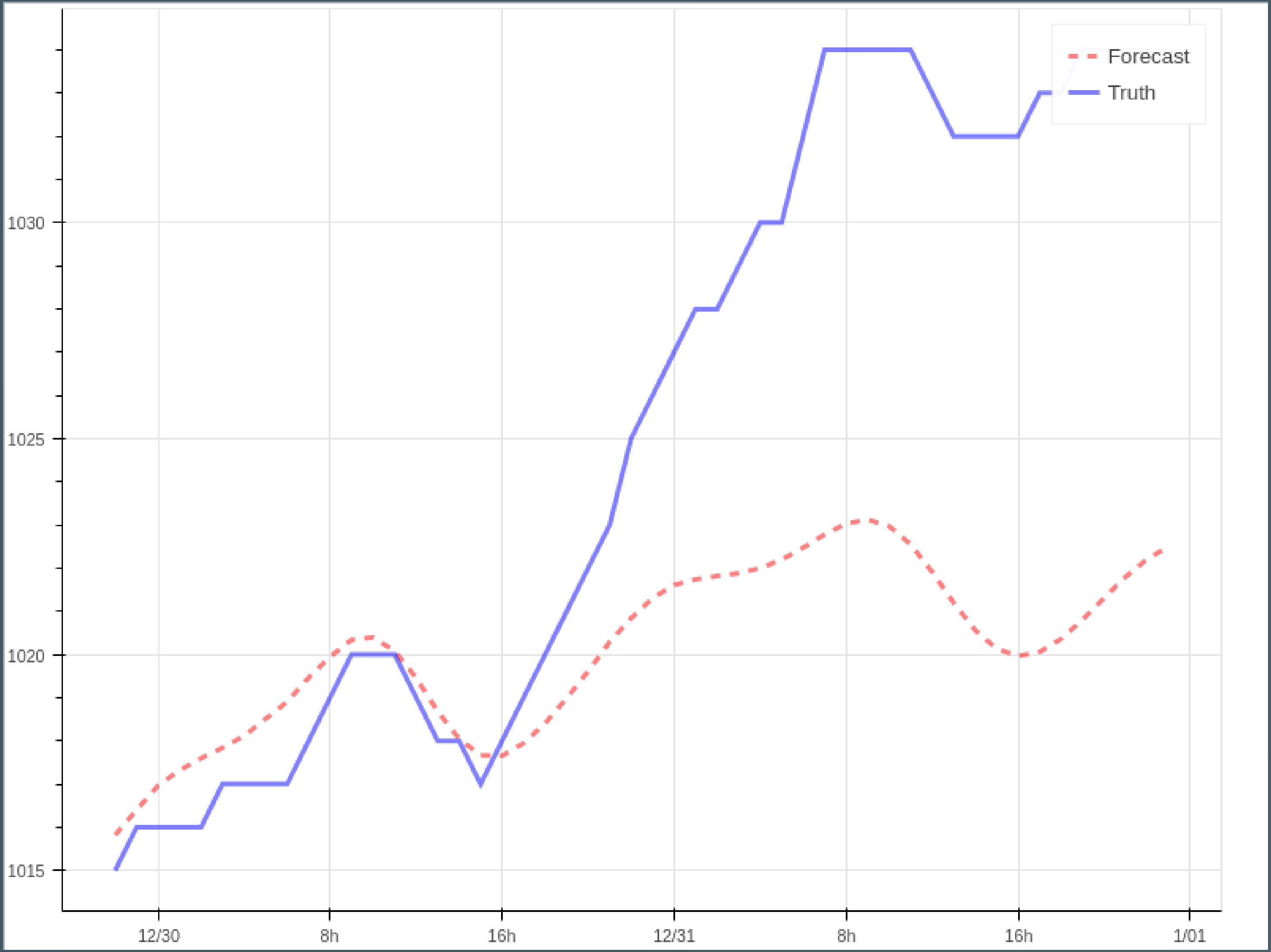
Particulate Matter



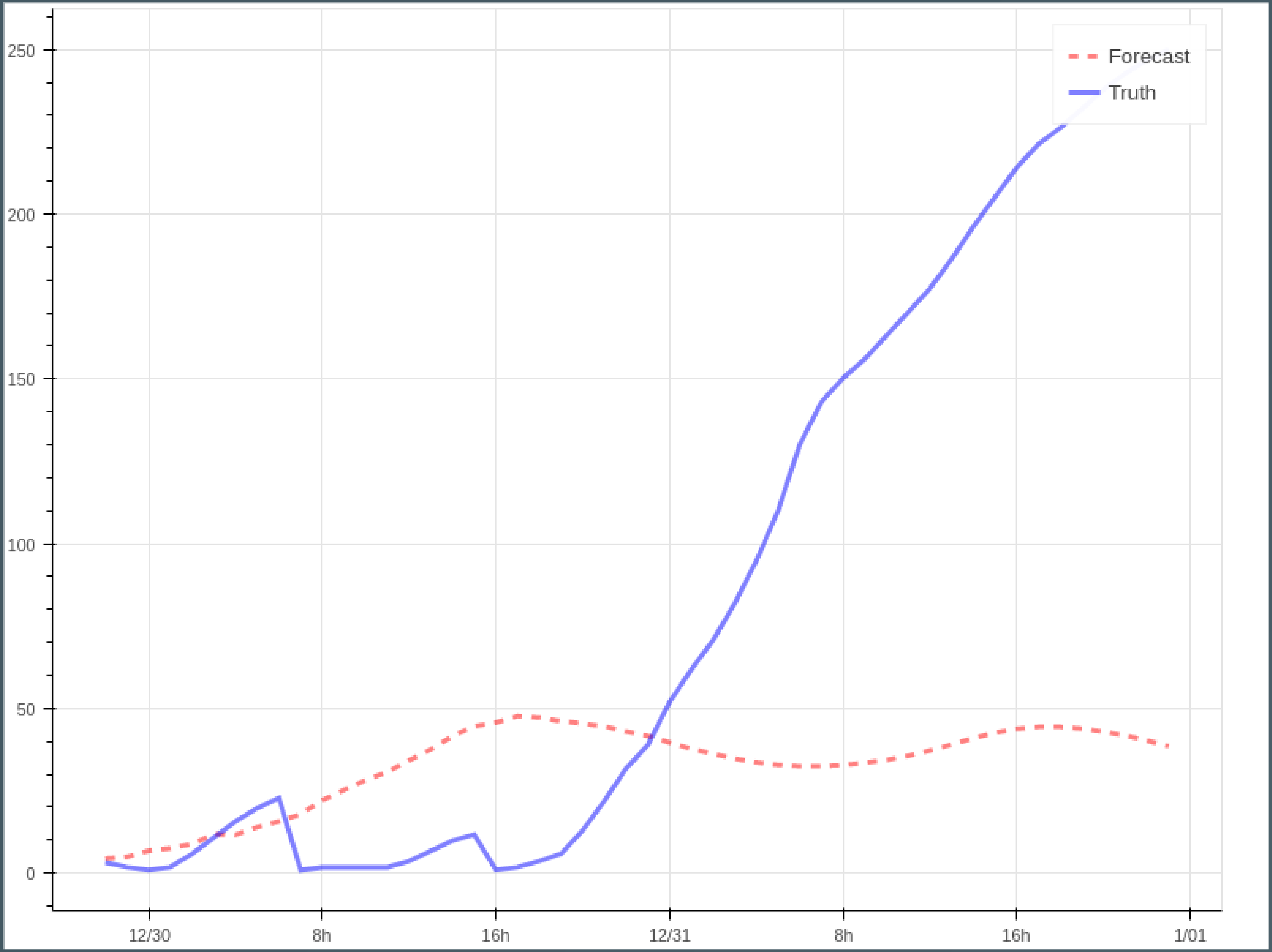
Temperature



Air Pressure



Wind Speed



Forecasting Observations

- Repeated Forecasts are needed when data is updated
- Forecasts are not accurate far into the future

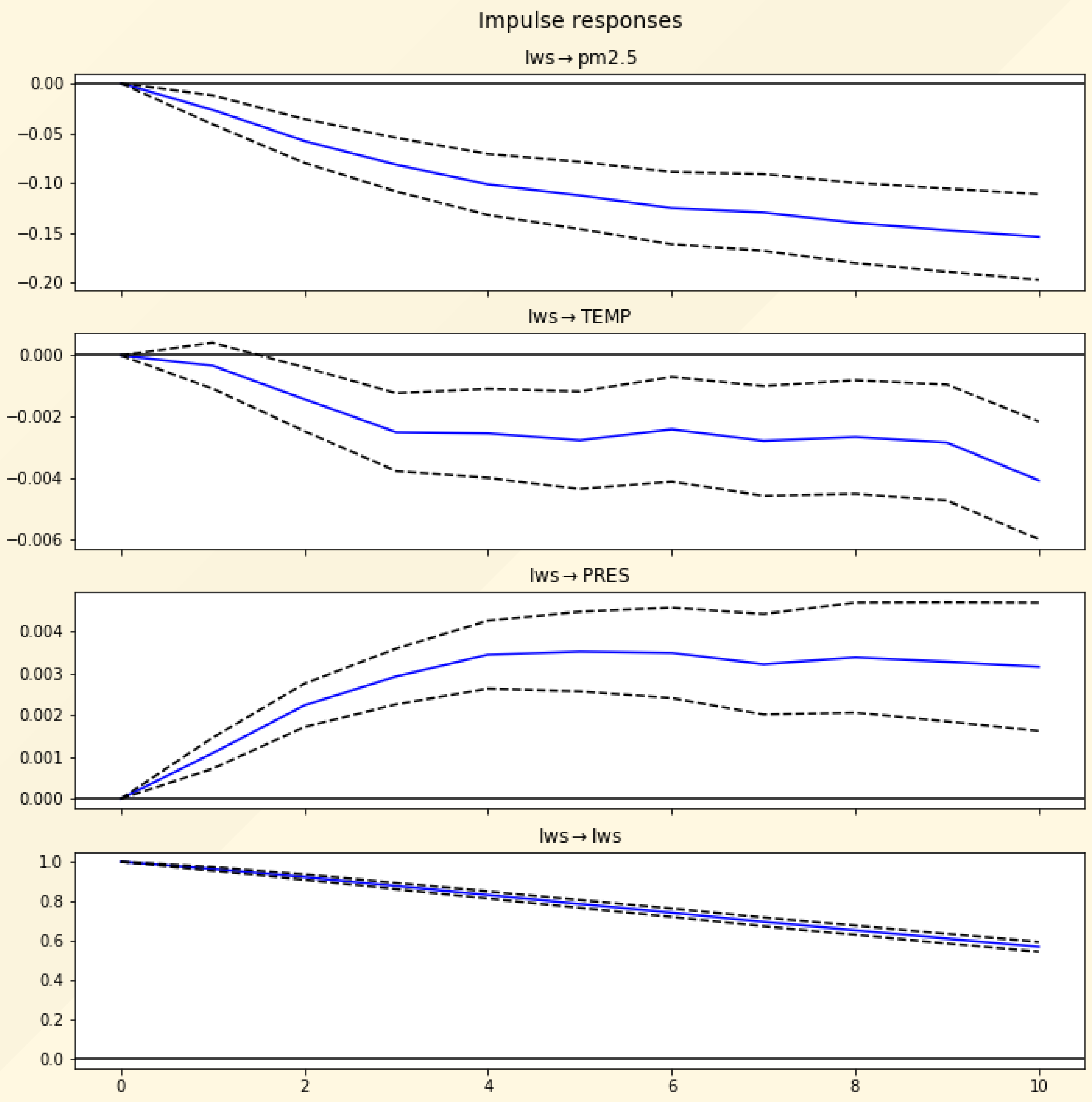
Impulse Response Functions

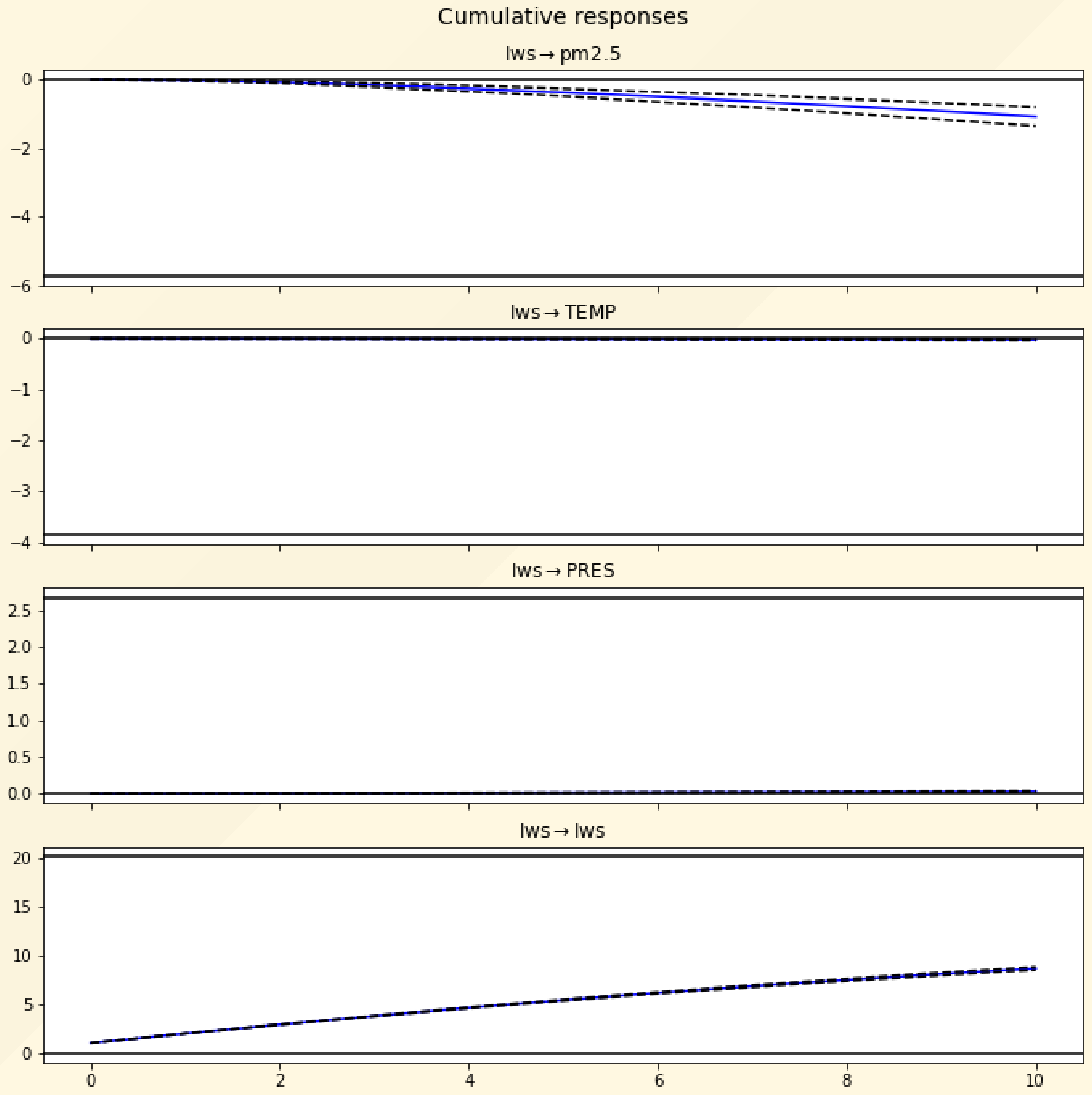
- VAR Models can show us how each variable responds to a shock in our system
- Frequently used to determine impact of policy changes or economic shocks in Macro models
- Give us insight into how our VAR model perceives the relationship between parameters over time

Impulse Response Functions

```
irf = reg.irf(10) # 10-period Impulse Response Fn
irf.plot(impulse = 'Iws') # Plot volume change impact
irf.plot_cum_effects(impulse = 'Iws') # Plot effects
```

- Generate a 10-period Impulse Response Function (IRF)
- Focus on plotting the effect of changes in trade volume on all variables (over 10 periods)
- Plot the cumulative effect over 10 periods


Impulse responses
Iws → pm2.5
Iws → TEMP
Iws → PRES
Iws → Iws
0.00
−0.05
−0.10
−0.15
−0.20
0.000
−0.002
−0.004
−0.006
0.004
0.003
0.002
0.001
0.000
1.0
0.8
0.6
0.4
0.2
0.0
0
2
4
6
8
10


Cumulative responses
lws → pm2.5
lws → TEMP
lws → PRES
lws → lws
0
−2
−4
−6
0
−1
−2
−3
−4
2.5
2.0
1.5
1.0
0.5
0.0
20
15
10
5
0
0
2
4
6
8
10

Saving Models

We can use `pickle` functions to store our models to disk, and utilize them later.

```
import cPickle as pkl

filename = '/your/directory/here' #string of file location
output = open(filename, 'wb') # allow python to write
pkl.dump(reg, output) # stores the reg object @ filename
output.close() # terminate write process
```

In this way, we can store just about any object in Python, although we have to take care with how large some objects may be.

Restoring Models

```
reg = pkl.load(open('yourfile.pkl', 'rb'))
```

When you are ready to access your model or data again, you can load your pickle back into memory.

- Forecast from same model on different days
- Share models with co-workers
- Just need to make sure to import libraries first!

For lab today:

New data!

- Let's check out our NFL scores data
- Get the data cleaned and ready
- Try shaping it and using models like VAR, ARIMA, or OLS to make predictions