ECON 8310 - Business Forecasting

Instructor:

Dustin White

Mammel Hall 332G

Office Hours:

By appointment (remote or live options available)

Contact Info:

drwhite@unomaha.edu

Quick Note

Your ability to use code to solve problems will be the basis for your grade in this course, so if you cannot commit the time to practice coding, you are not likely to pass this class.

Grade Details

Score	Grade	Score	Grade
>94%	A	72.5-77.4	C
90-93.9	A-	70-72.4	C-
87.5-89.9	B+	62.5-69.9	D
82.5-87.4	B	60-62.5	D-
80-82.4	B-	<60	F
77.5-79.9	C+		

Grade Details

Assignment	Percent of Grade
Lab Work	30%
Reading Assignments	20%
Participation	20%
Course Project	20%

My Expectations

- Plan on spending **all of our time in lab** working on homework and projects and refining your predictions
- **Take charge** of your assignments and projects; they will be open-ended!

Expectations of Me

- I will work through examples of code in class
- I will be available during lab and office hours to help you with assignments
- I will revise the course material as needed to suit your interests
    - Just added a bunch of new models last spring!

Lecture 1: Intro and OLS Review

What is Forecasting?

Forecast: "to predict or estimate (a future event or trend)" -- Google Dictionary

- Predict weather patterns
- Estimate the quantity of stock required during a certain time-span
- Generally, determine the most likely outcome of a stochastic process based on previous events
- **Learn from patterns**

Forecasting is just fancy trendlines

In this course, we want to learn how to predict outcomes based on the information that we already possess.

Forecasting

- Time Series modeling
- Neural Networks
- Predictive modeling using machine learning
- Bayesian models for complex processes
- **Choosing the best model for the job**

Remembering OLS...

- Ordinary Least Squares (OLS) is the foundation of regression analysis, and an excellent starting point for this course
- Estimates the expected outcome ($\hat{y}$) given the inputs (x)

A Little More Remembering...

- Calculating coefficient standard errors informs us about the level of noise in the data
- R^2 and Adjusted R^2 tell us how much of the total variation our model accounts for

OLS in Statsmodels

```
import statsmodels as sm
# Declare the model, and create an instance of the OLS
#  class
model = sm.OLS(endog = y, exog = x)
# Fit the model, optionally using specific parameters
modelFit = model.fit()
```

We can easily use just a couple lines to implement an Ordinary Least Squares regression model. We simply declare the model, then fit it.

We can also use the `.summary()` and `.predict()` methods on fitted models to view regression tables or to make predictions with new observations.

Transforming OLS for Time Series Data

What if we take yesterday's y value and make it our exogenous variable? The model would be written

$$y_{t+1} = \alpha + \beta_t \cdot y_t + \epsilon$$

This model violates our OLS model assumptions! We need tools to overcome this

Autocorrelation

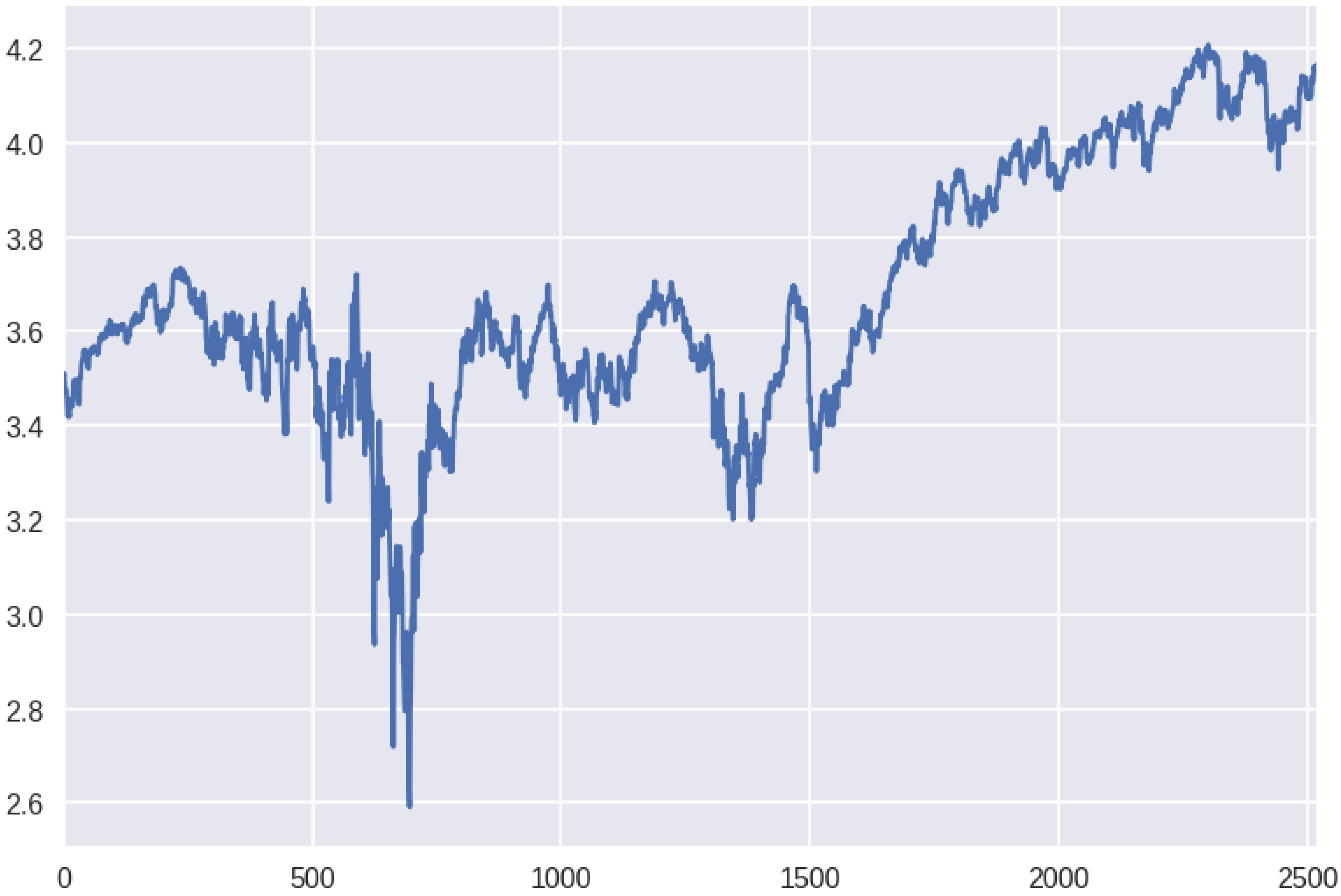
One of the primary assumptions of the OLS model is that

$$Cov(\epsilon_t, \epsilon_s) = 0, \ \forall \ t \neq s$$

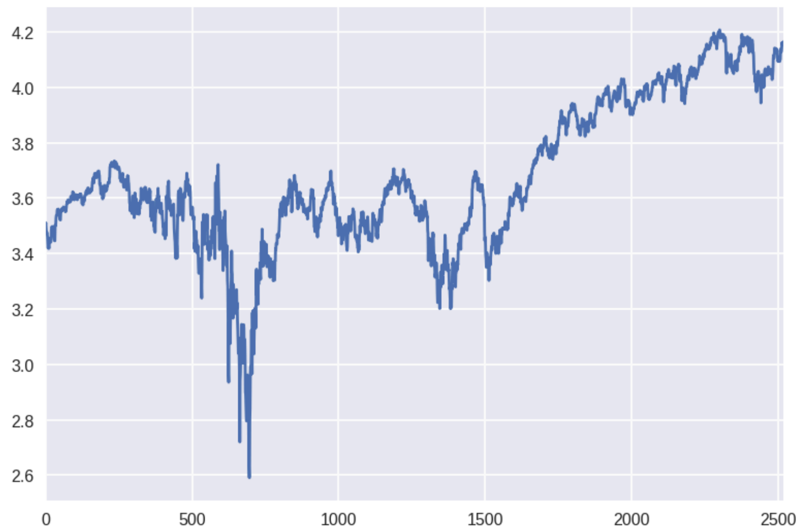
This assumption is clearly **not** valid in any time series that can be meaningfully modeled.

Let's look at some data to find out why.

Autocorrelation



Autocorrelation



We need to find a model that can eliminate the autocorrelation almost always seen in time series data.

Autoregressive Models

AR models are based on the premise that deviation from the underlying trend in the data persists in **all future observations**.

$$y_t = \alpha + \sum_{i=1}^{p} \rho_i \cdot y_{t-i} + \epsilon_t$$

Here ρ is the correlation term between periods and ϵ is an error (shock) term

AR Models

- We need to consider lagged observations of y in order to predict future outcomes
- The number of lags that we include is the **order** of our AR model
    - The model is an AR(p) Model, where p is the order of the model

AR Models

- The AR coefficients tell us how quickly a model returns to its mean
    - If the coefficients on AR variables add up to close to 1, then the model reverts to its mean **slowly**
    - If the coefficients sum to near zero, then the model reverts to its mean **quickly**

Moving Average Models

While an AR($\cdot$) model accounts for previous values of the dependent variable, MA($\cdot$) models account for previous values of the **error** terms:

$$AR(p) = \alpha + \sum_{i=1}^{p} \rho_i \cdot y_{t-i} + \epsilon_t$$

$$MA(q) = \alpha + \sum_{i=1}^{q} \theta_i \cdot \epsilon_{t-i} + \epsilon_t$$

Moving Average Models

An MA model suggests that the current value of a time-series depends linearly on previous error terms.

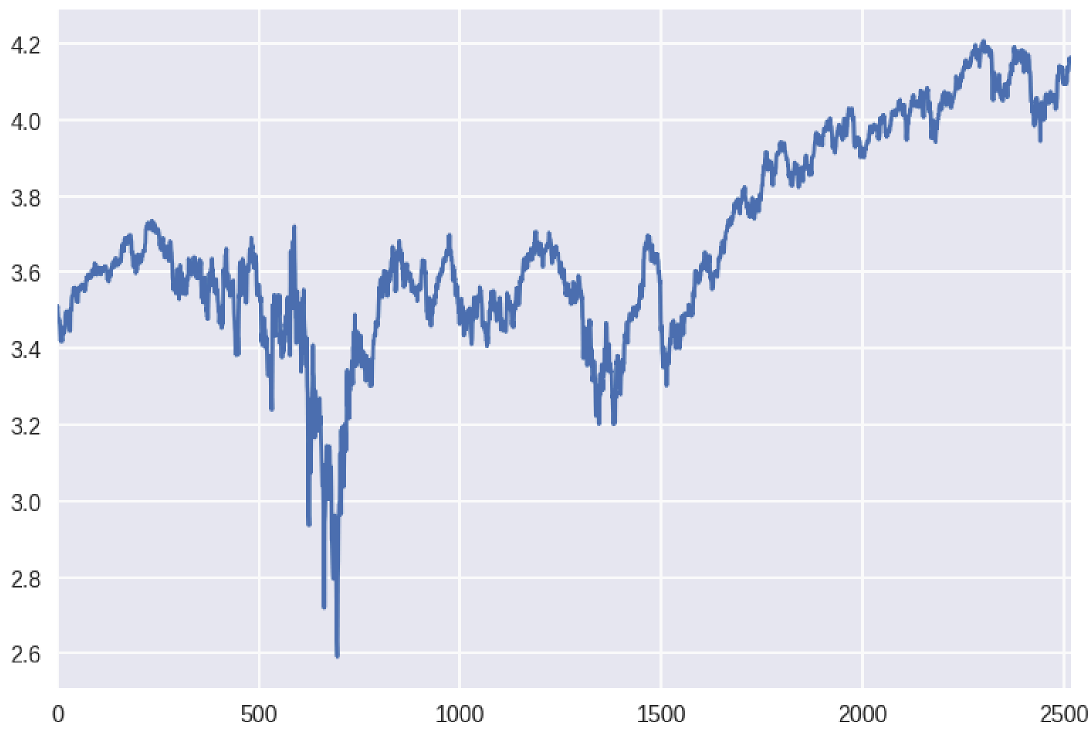
- Current value depends on how far away from the underlying trend previous periods fell
- The larger θ becomes, the more persistent those error terms are

Moving Average Models

- AR models' effects last infinitely far into the future
  - Each observation is dependent on the observation before
- In an MA model, the effect of previous periods only persist for q periods
  - Because each error is uncorrelated with previous errors

Integrated Models

Integration occurs when a process is non-stationary. A non-stationary process is one that contains a linear time trend. One example might be a long-term series of stock prices:



Integrated Models

We need to ensure that our data is stationary. To do so, we need to remove any time-trend from the data.

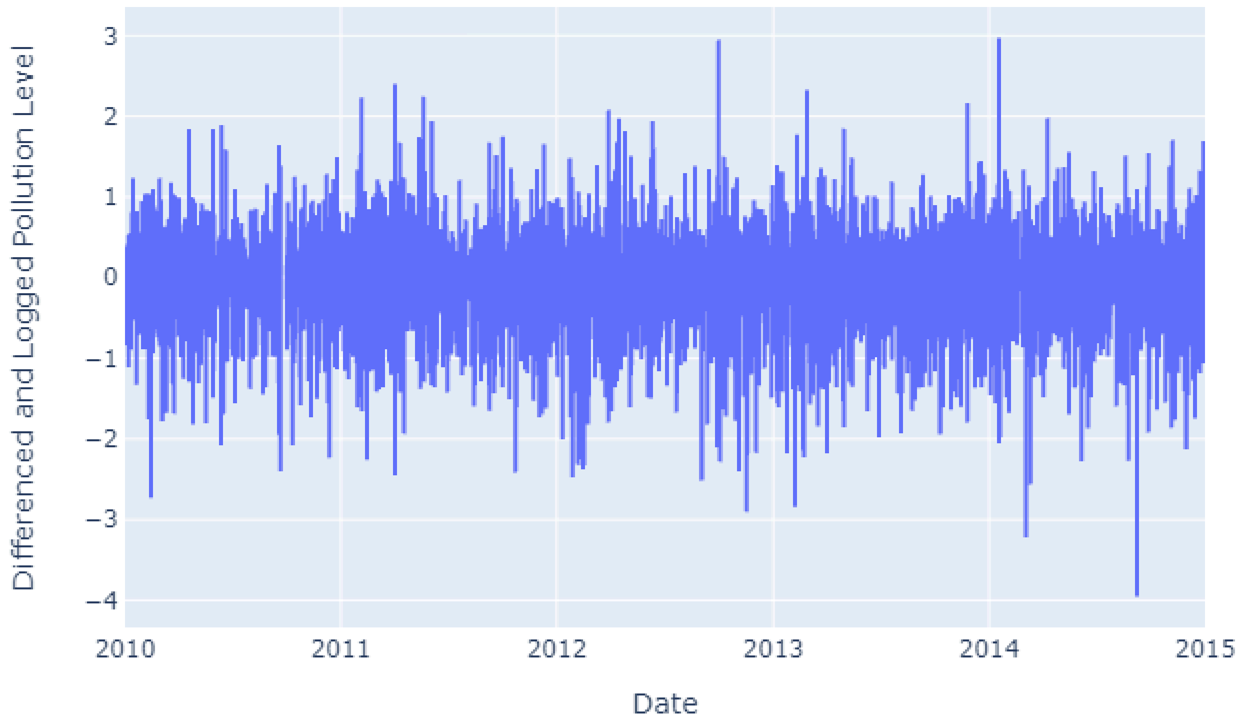
- This is typically done through differencing

$$y_i^s = y_i - y_{i-1}$$

where y_t^s is the stationary time series based on the original series y_t

Integrated Models

Here, the time trend has been differenced out of the data:



Integrated Models

The Integration term d represents the number of differencing operations performed on the data:

- I(1): $y_t^s = y_t - y_{t-1}$
- I(2): $y_t^s = (y_t - y_{t-1}) - (y_{t-1} - y_{t-2})$

Where an I(2) model is analogous to a standard difference-in-differences model applied to time-series data.

Putting it Together

In order to account for all the problems that we might encounter in time series data, we can make use of ARIMA models.

Auto**R**egressive **I**ntegrated **M**oving **A**verage models allow us to

- Include lags of the dependent variable
- Take differences to eliminate trends
- Include lagged error terms

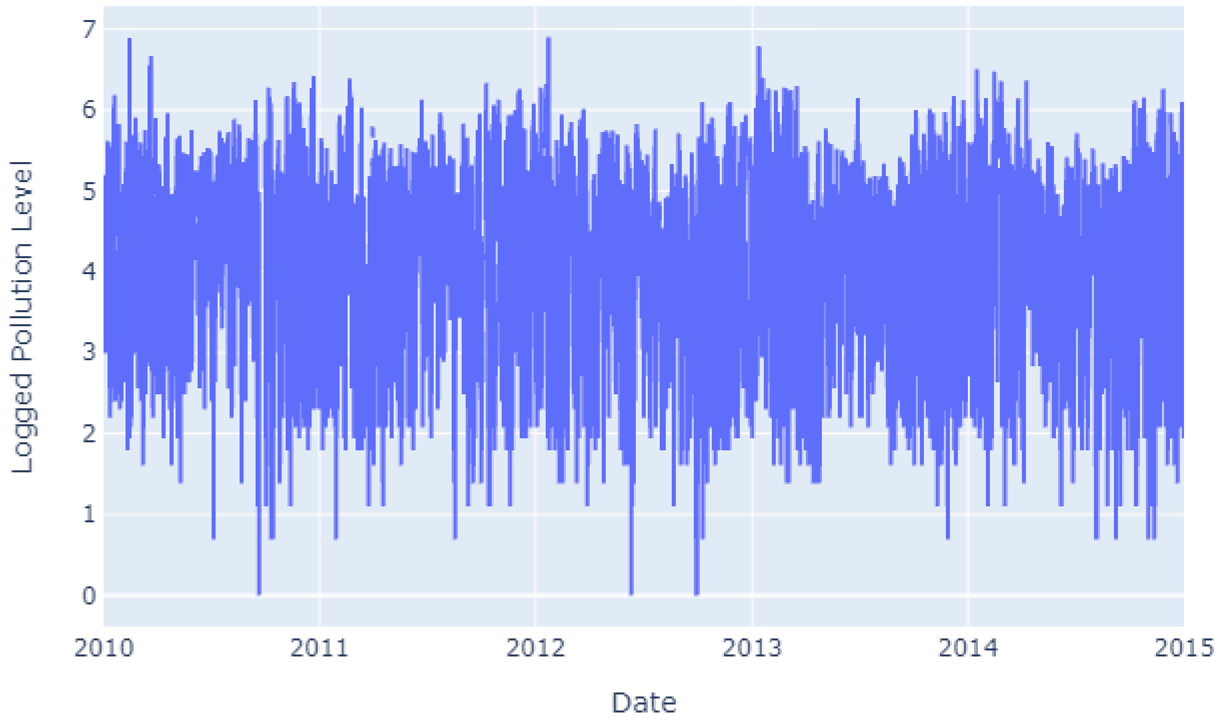
ARIMA in Python

```
# Import needed libraries
import pandas as pd
import numpy as np
import statsmodels.api as sm
import statsmodels.tsa.stattools as st
import plotly.express as px

# Read data, then set the index to be the date
# NOTE: make the file a single line!!
data = pd.read_csv("https://github.com/dustywhite7/Econ8310/blob/master/
DataSets/pollutionBeijing.csv?raw=true")

data['datetime'] = pd.to_datetime(data['datetime'],
        format='%Y-%m-%d %H:%M:%S')
data.set_index(pd.DatetimeIndex(data['datetime']),
        inplace=True)
data['logpm'] = np.log(data['pm2.5'])
```

ARIMA in Python

```
# Plot the data
px.line(data, x='datetime', y='logpm',
       labels = {
           'datetime' : 'Date',
           'logpm' : 'Logged Pollution Level'
       })
```

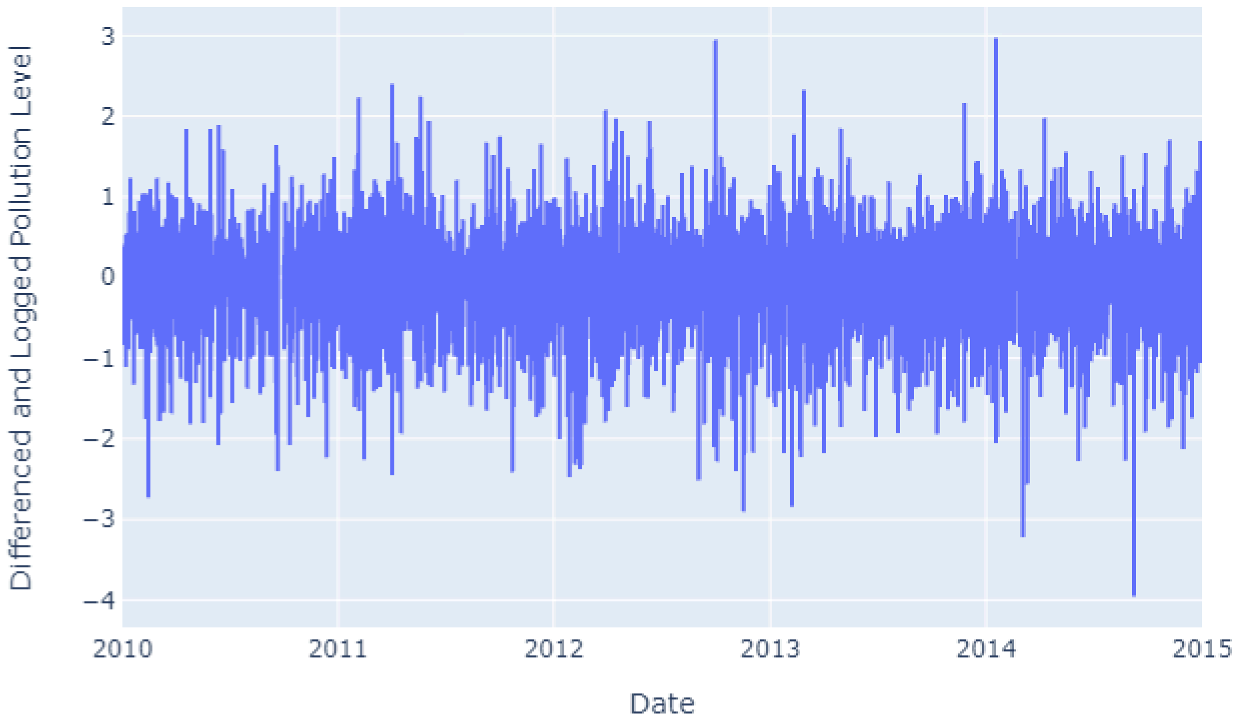
ARIMA in Python



ARIMA in Python

```
# Plot the DIFFERENCED data

data['pmdiff'] = data['logpm'].diff()


px.line(data, x='datetime', y='pmdiff',
       labels = {
           'datetime' : 'Date',
           'pmdiff' : 'Differenced and Logged Pollution Level'
       })
```

ARIMA in Python



Testing for Stationarity

We can use the **Augmented Dickey-Fuller Test** to determine whether or not our data is stationary.

- H_0: A unit root is present in our data
- H_A: The data is stationary

This can help us to determine whether or not differencing our data is required or sufficient for inducing stationarity.

Testing for Stationarity

We can use the **Augmented Dickey-Fuller Test** to determine whether or not our data is stationary.

```
>>> st.adfuller(
>>>     data['pm2.5'][-250:], maxlag=12)

(-3.1576359480752445, # The test statistic
 0.022571607041567278, # The p-value
 2, # Number of AR lags in model
 247, # Number of obvservations
 {'1%': -3.45710530972632109,
  '10%': -2.5730443824681606, # The 1%, 5%, and 10%
  '5%': -2.873313676101283},  # thresholds
 2272.5419900847974) # The model information criterion
```

In this case, we can reject the unit-root hypothesis!

Fitting the ARIMA model

```
import statsmodels.api as sm

model = sm.tsa.arima.ARIMA(endog=data['logpm'], order=(1,1,0))
                  # specifying an ARIMA(1,1,0) model
reg = model.fit() # Fit the model using standard params
res = reg.resid   # store the residuals as res
```

ARIMA + X

We can improve on the ARIMA model in many cases if we use ARIMA**X** (ARIMA with e**X**ogenous variables) models to include exogenous regressors in our estimations!

ARIMAX

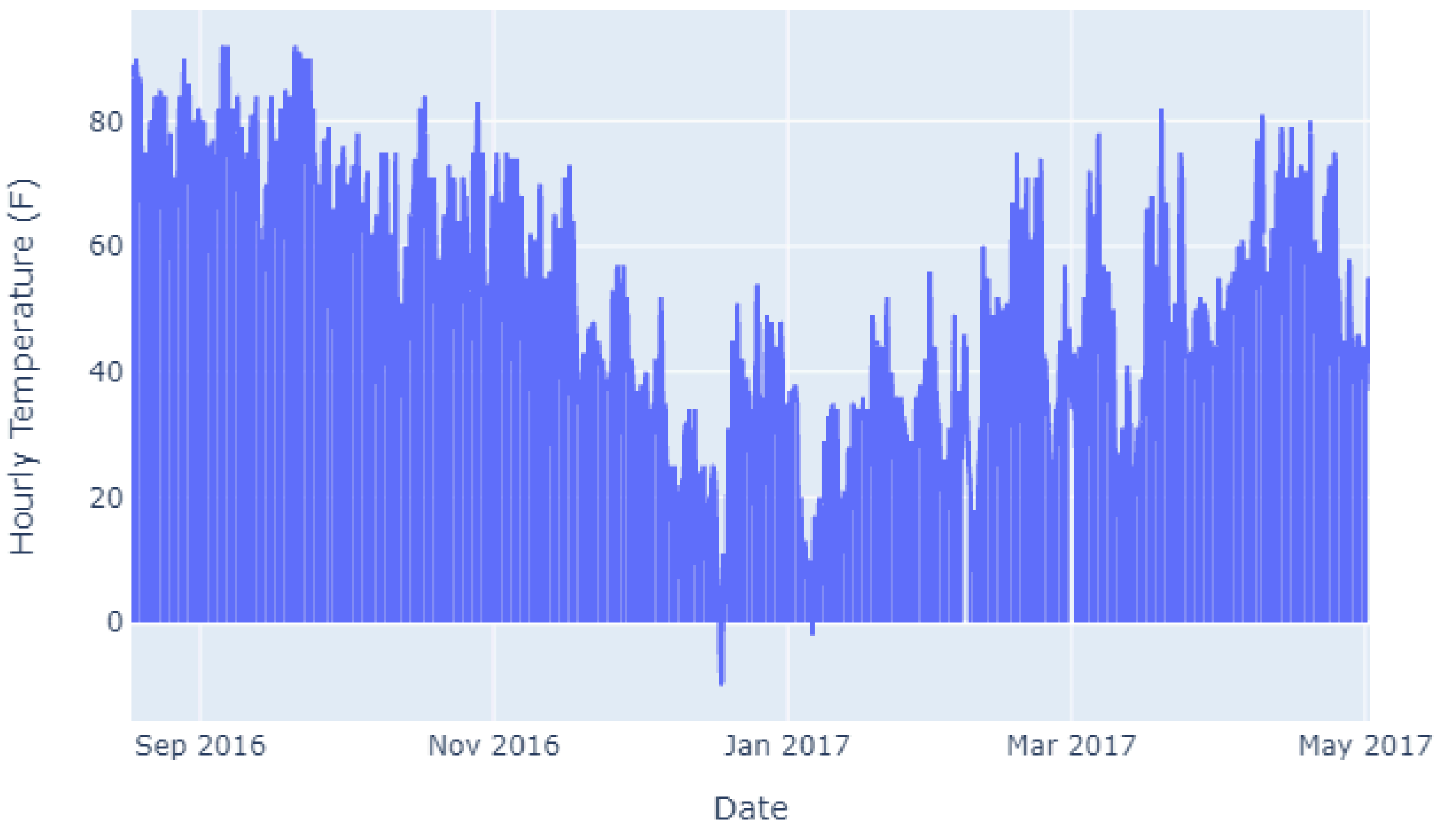
Let's use some weather data to get started:

```
import pandas as pd
import numpy as np
import patsy as pt
import statsmodels.api as sm
import statsmodels.tsa.stattools as st
import plotly.express as px

# NOTE: Make the string a single line again!
data = pd.read_csv("https://github.com/dustywhite7/Econ8310/
blob/master/DataSets/omahaNOAA.csv?raw=true")[-(365*24):]
                # We are keeping only the last 365 days
```

ARIMAX

```
px.line(data, x='DATE', y='HOURLYDRYBULBTEMPF',
        labels = {
            'DATE' : 'Date',
            'HOURLYDRYBULBTEMPF' : 'Hourly Temperature (F)'
        })
```

ARIMAX



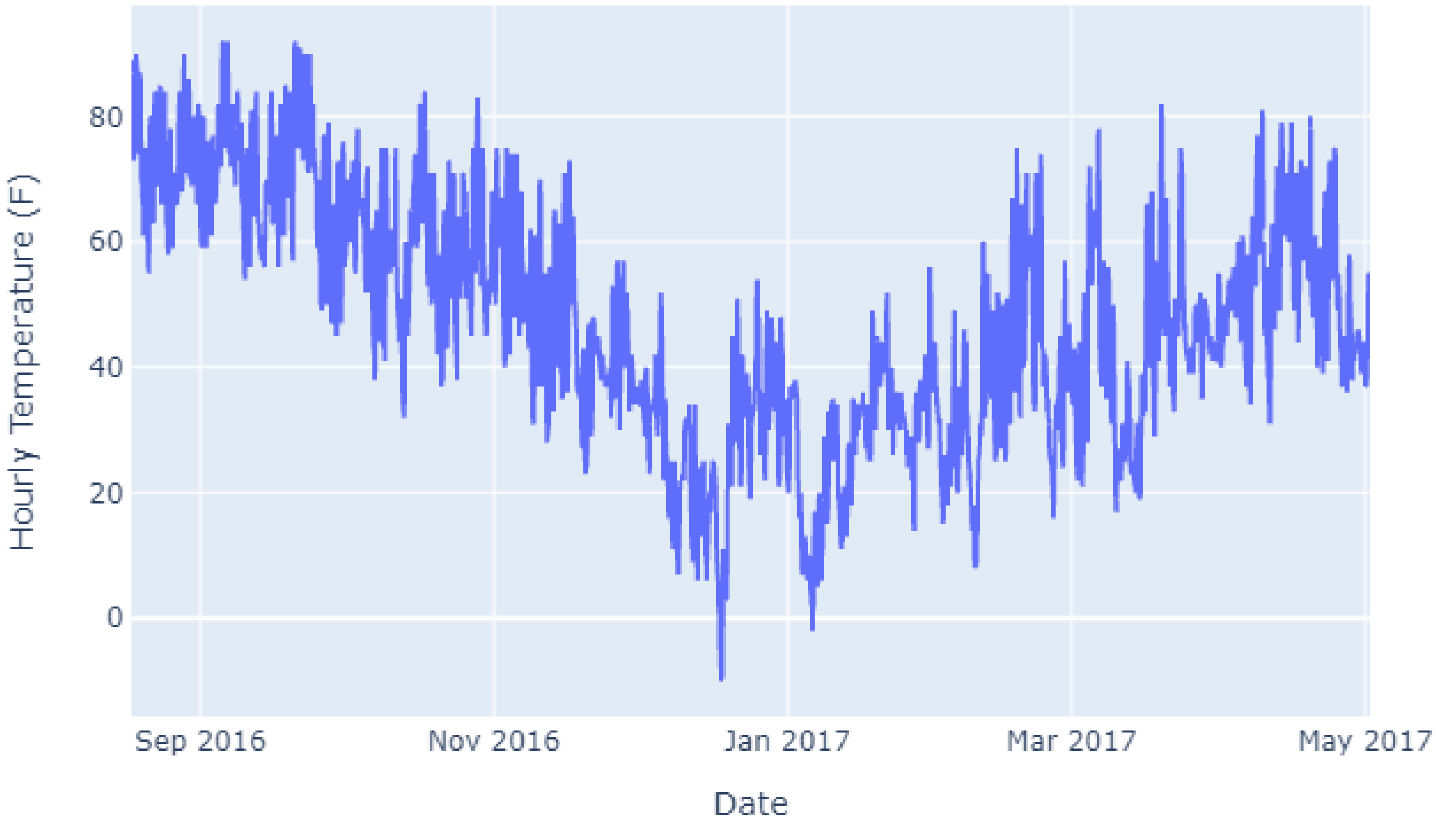
Hourly Temperature (F)
80
60
40
20
0
Sep 2016
Nov 2016
Jan 2017
Mar 2017
May 2017
Date

ARIMAX

We have a lot of erroneous entries, and they're all recorded as 0!

```
data['cleanTemp'] = data['HOURLYDRYBULBTEMPF'].replace(0, method='pad')

px.line(data, x='DATE', y='cleanTemp',
        labels = {
            'DATE' : 'Date',
            'cleanTemp' : 'Hourly Temperature (F)'
        })
```

Much Better!



Hourly Temperature (F)
80
60
40
20
0
Sep 2016
Nov 2016
Jan 2017
Mar 2017
May 2017
Date

ARIMAX

```
eqn = "HOURLYDRYBULBTEMPF ~ HOURLYWindSpeed + " +
"HOURLYStationPressure + HOURLYPrecip"

y, x = pt.dmatrices(eqn, data = data)

# The exog argument permits us to include exogenous vars
model = sm.tsa.arima.ARIMA(y, order=(1,1,0), exog=x)
reg = model.fit()
reg.summary()
```

ARIMA Model Results

Dep. Variable:	D.HOURLYDRYBULBTEMPF	**No. Observations:**	8740
Model:	ARIMA(1, 1, 0)	**Log Likelihood**	-31354.110
Method:	mle	**S.D. of innovations**	8.745
Date:	Tue, 25 Aug 2020	**AIC**	62720.220
Time:	13:26:12	**BIC**	62762.673
Sample:	1	**HQIC**	62734.687

	coef	**std err**	**z**	**P>|z|**	**[0.025**	**0.975]**
Intercept	-31.1611	0.666	-46.773	0.000	-32.467	-29.855
HOURLYWindSpeed	-0.0004	0.012	-0.035	0.972	-0.025	0.024
HOURLYStationPressure	1.1108	0.024	46.197	0.000	1.064	1.158
HOURLYPrecip	4.0063	1.937	2.069	0.039	0.210	7.802
ar.L1.D.HOURLYDRYBULBTEMPF	-0.2799	0.014	-20.474	0.000	-0.307	-0.253

Roots

	Real	**Imaginary**	**Modulus**	**Frequency**
AR.1	-3.5726	+0.0000j	3.5726	0.5000

SARIMAX

Where can we go when we have cyclical data?

- We can introduce "seasonality" into our model

The Seasonal Autoregressive Integrated Moving Average Model with Exogenous Regressors (SARIMAX) is designed to deal with this kind of data and model.

SARIMAX

We know that temperatures fluctuate daily (even though we have attempted to difference this out)

```
model = sm.tsa.arima.ARIMA(y, order=(1,1,0),
                seasonal_order=(1,1,0,24), exog=x)
# trend='c' indicates that we want to include a
#   constant/intercept term in our regression
reg = model.fit(trend='c', maxiter=500, solver='nm')
reg.summary()
```

Here, we need to include terms for our **seasonal** AR, I, and MA terms, as well as the periodicity of our data (24 observations per day).

SARIMAX Results

Dep. Variable:	y	**No. Observations:**	8741
Model:	SARIMAX(1, 1, 0)x(1, 1, 0, 24)	**Log Likelihood**	-27458.025
Date:	Tue, 25 Aug 2020	**AIC**	54930.051
Time:	13:42:18	**BIC**	54979.561
Sample:	0	**HQIC**	54946.925
	- 8741		
Covariance Type:	opg		

	coef	**std err**	**z**	**P>|z|**	**[0.025**	**0.975]**
const	2.956e-09	7.22e+04	4.1e-14	1.000	-1.41e+05	1.41e+05
x1	0.0984	0.011	8.649	0.000	0.076	0.121
x2	1.4308	0.005	291.983	0.000	1.421	1.440
x3	0.8634	1.468	0.588	0.557	-2.015	3.742
ar.L1	-0.3363	0.005	-71.300	0.000	-0.346	-0.327
ar.S.L24	-0.4756	0.004	-105.925	0.000	-0.484	-0.467
sigma2	31.8809	0.205	155.252	0.000	31.478	32.283

Ljung-Box (Q):	2260.57	**Jarque-Bera (JB):**	35120.33
Prob(Q):	0.00	**Prob(JB):**	0.00
Heteroskedasticity (H):	0.48	**Skew:**	-0.06
Prob(H) (two-sided):	0.00	**Kurtosis:**	12.83

Forecasting ARIMAX/SARIMAX

When we forecast based on models with exogenous variables, we need to include those variables as an argument to the forecast method.

```
# Generating our Forecast
fcst = reg.forecast(steps=10, exog=x[-10:])
                     # Generate forecast
```

Lab Time!