Lectures 2 & 3: Time Series, ARIMA Models

This lesson is based on material by Robert Nau, Duke University

Refresher:

Using Statsmodels to implement OLS

```
import statsmodels.api as sm
import patsy as pt

# NOTE: Make the file a single line again!
data = pd.read_csv("https://github.com/dustywhite7/Econ8310/blob/master/
        DataSets/pollutionBeijing.csv?raw=true")
y, x = pt.dmatrices("Q('pm2.5') ~ TEMP", data=data)
model = sm.ols(exog=x, endog=y)
reg = model.fit() # Fit the model using standard params
```

Time Series Data

A time series consists of repeated observations of a single variable, y, at various times, t.

$$\mathbf{y} = \{y_1, y_2, y_3, \ldots, y_t\}$$

We seek to predict y_{t+1} using the information from previous observations $\mathbf{y}$.

Time Series Data

In order to estimate y_{t+1}, we need to find the effect of previous observations of y on the upcoming period. We might write this model as

$$y_{t+1} = \alpha + \sum_{s=1}^{t} \beta_s \cdot y_s + \epsilon$$

Time Series Data

If we choose to base our model solely on the previous period, then 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

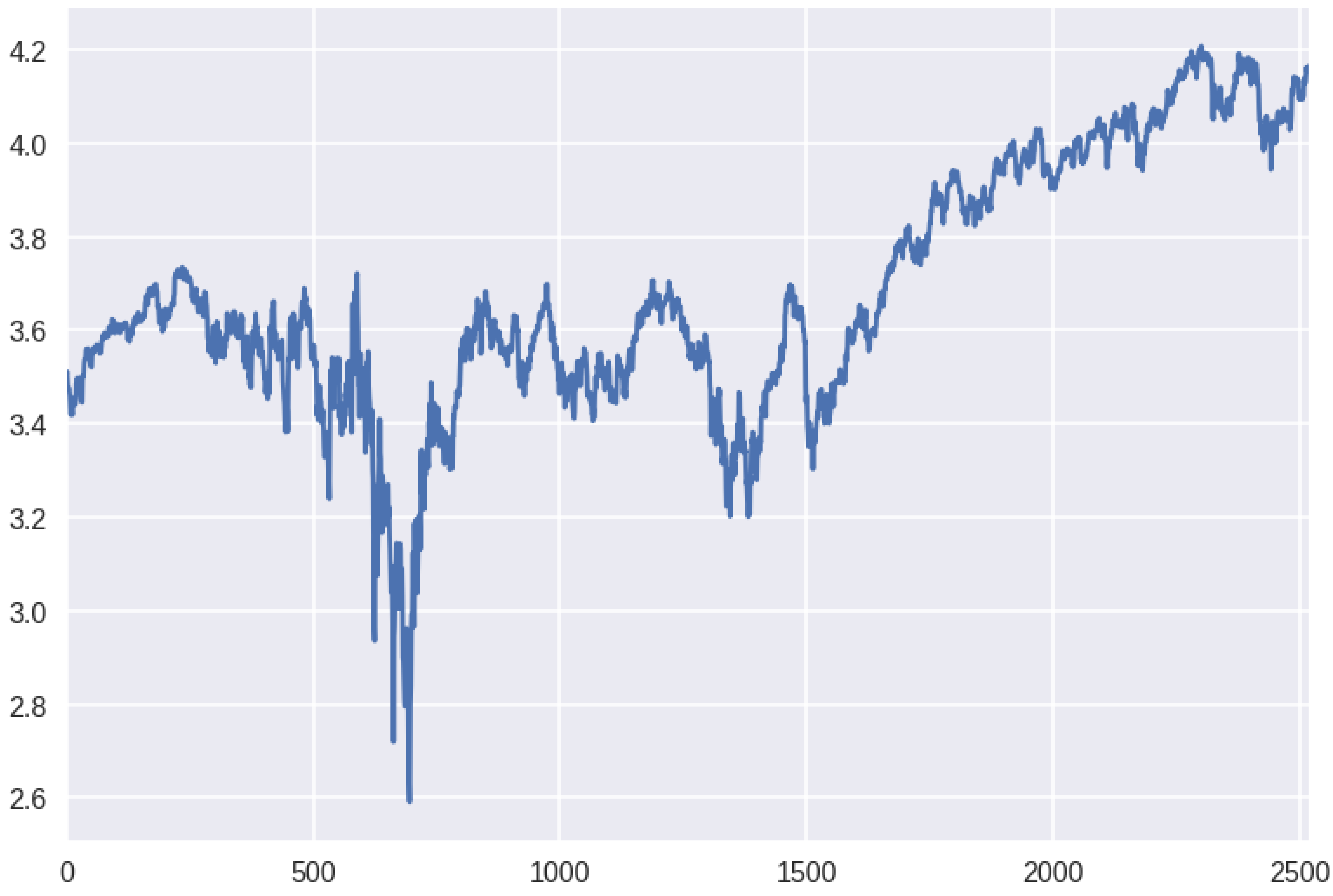
One of the primary assumptions of the OLS model is that

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

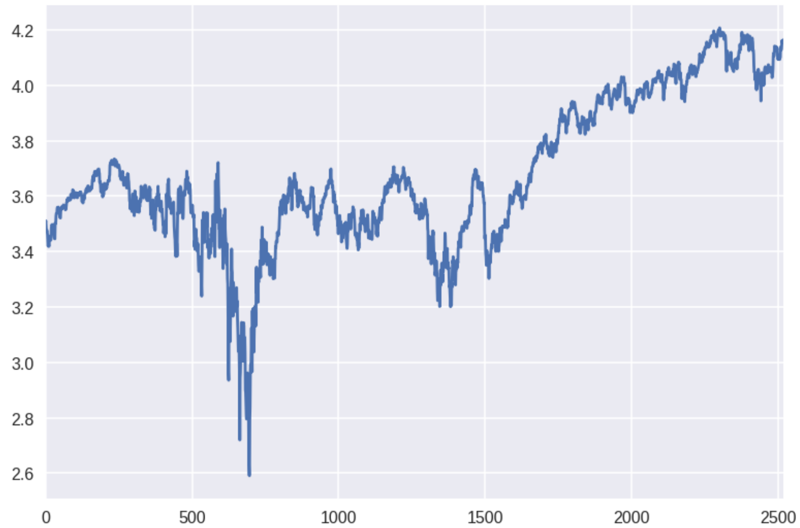
This assumption is clearly **not** valid in many time series.

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(·) model accounts for previous values of the dependent variable, MA(·) 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.

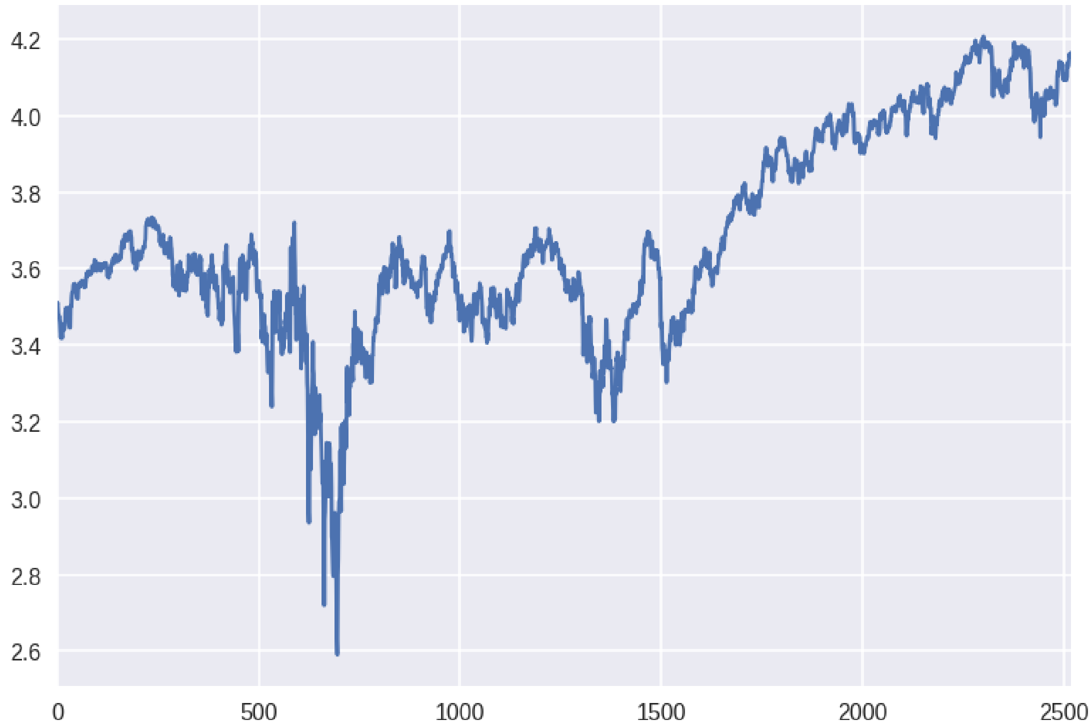
- Current value depends on how far away from the the underlying trend previous periods fell
- The larger θ becomes, the more persistent those error terms are

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.

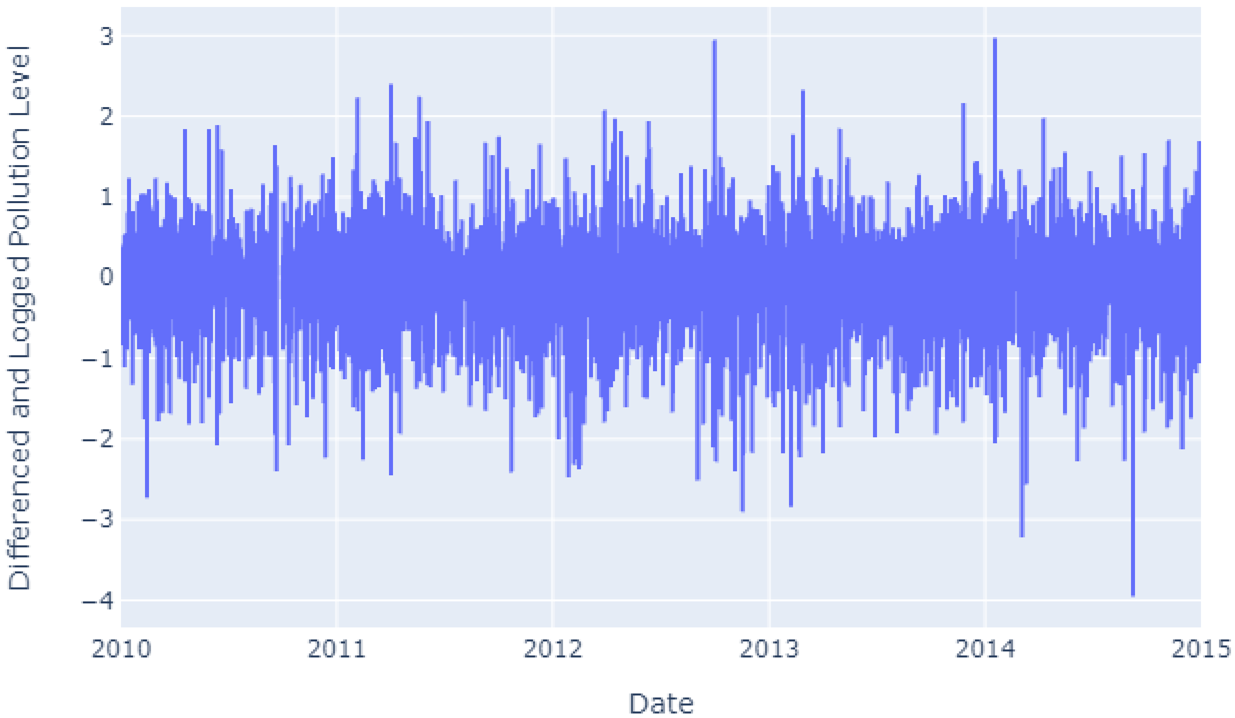
- This is typically done through differencing

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

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

The ARIMA Model

ARIMA models are often referred to as ARIMA(p, d, q) models, where p, d, and q are the parameters denoting the order of the autoregressive terms, integration terms, and moving average terms, respectively.

- It is often a matter of guessing and checking to find the correct specification for a model

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

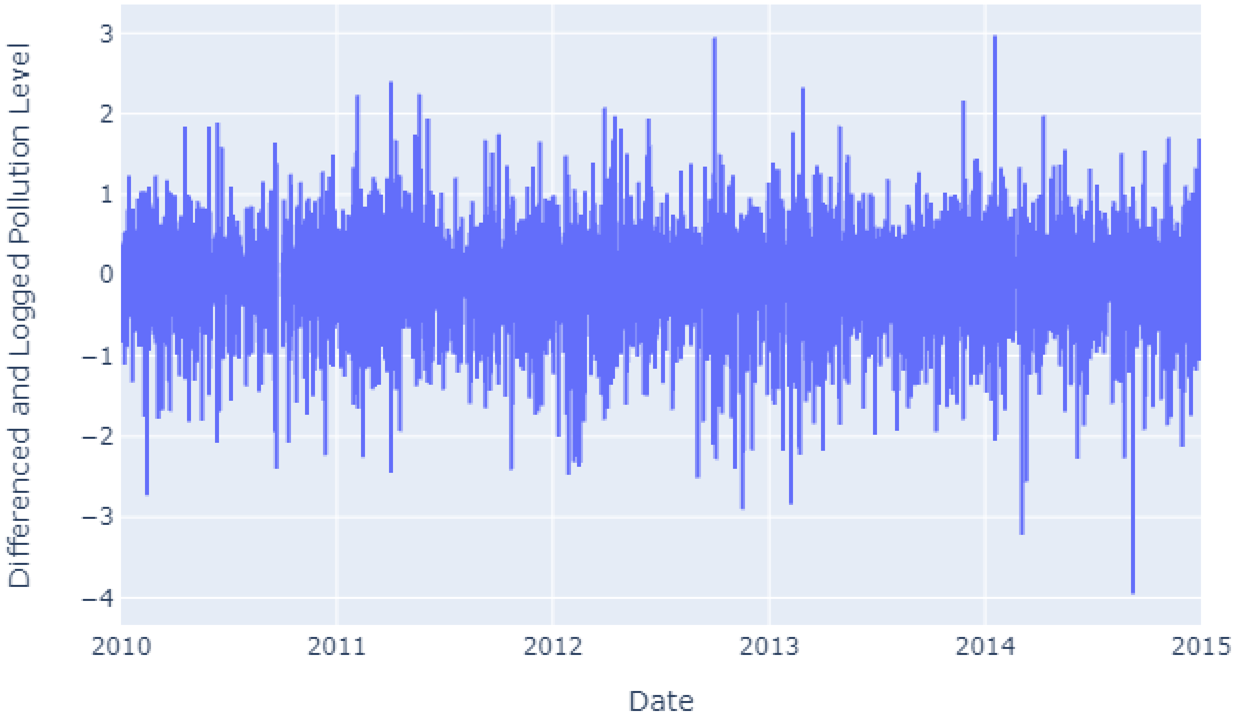
```
# Plot the data
px.line(data, x='datetime', y='logpm',
        labels = {
            'datetime' : 'Date',
            'logpm' : 'Logged Pollution Level'
        })
```

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%': -3.457105309726321,
  '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(np.log(data["pm2.5"]), (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
```

Once we fit the ARIMA model using our selected specification, we can then explore the goodness of fit of the model using our model residuals (forecast errors). We will focus on this next week.

Part 2 (Lecture 3) - Choosing your model

Finding the Right Fit

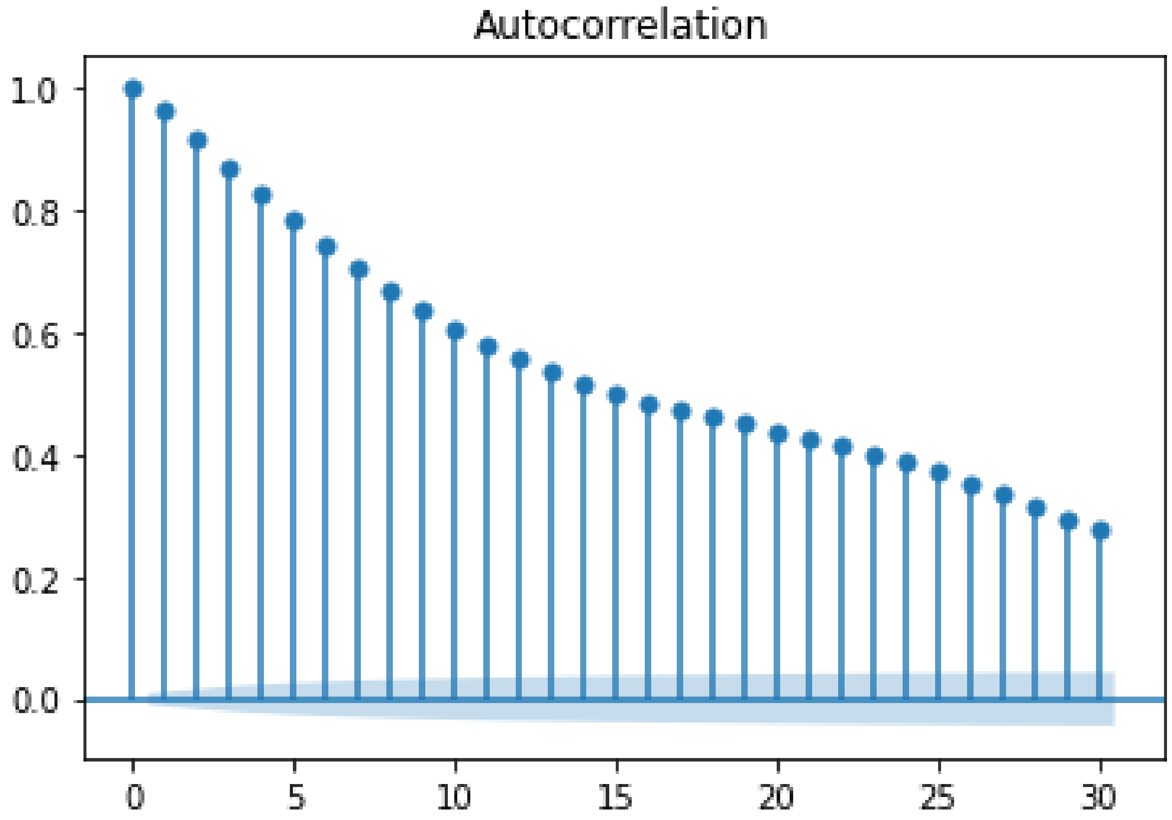
- Time series models are unique in Econometrics: we will typically **visually** diagnose the proper specifications for our model
  - This takes practice
  - This takes repetition and iteration for any given model

The Autocorrelation Function (ACF)

The ACF illustrates the correlation between a dependent variable and its lags.

- Choose how many lags to explore (based on nature of data)
- **Reminder**: correlations will vary between -1 and 1, with 1 being perfect correlation, and -1 being perfect inverse correlation
- Correlation can be cyclical!

The Autocorrelation Function (ACF)

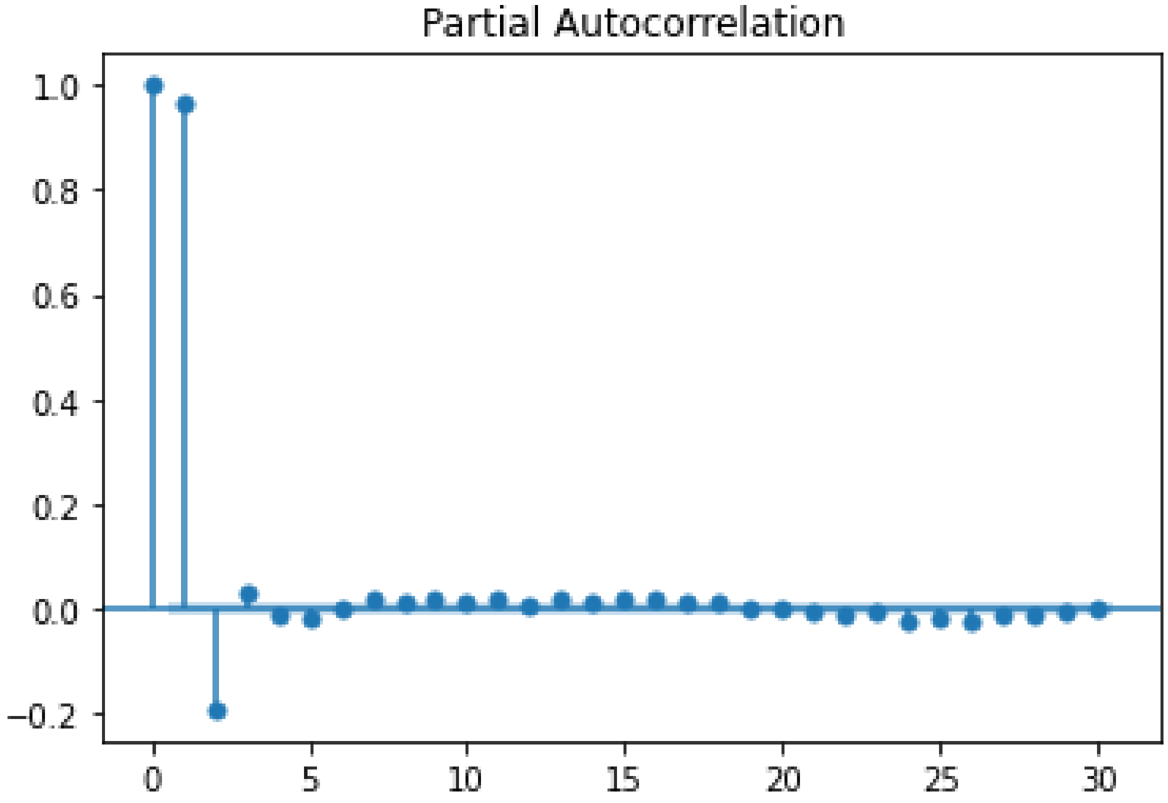


The Partial Autocorrelation Function

The PACF illustrates the correlation between a dependent variable and its lags, **after controlling for lower-order lags**.

- Choose how many lags to explore (based on nature of data)

The Partial Autocorrelation Function (PACF)



Building the Model

1. Make the series **stationary**
    - When the ACF falls "quickly" to zero at higher lags, the series is stationary
    - Can also use a **unit root test** to check for stationarity

Building the Model

1. Make the series **stationary**
2. Use ACF and PACF plots to decide if you should include **AR** or **MA** terms in your model
   - Remember that we typically do not use both in the same model

Signatures of AR and MA models:

AR Model: ACF dies out gradually, and the PACF cuts off sharply after a few lags

MA Model: ACF cuts off sharply, and PACF dies off more gradually (remember that **MA** models are based on previous *errors*)

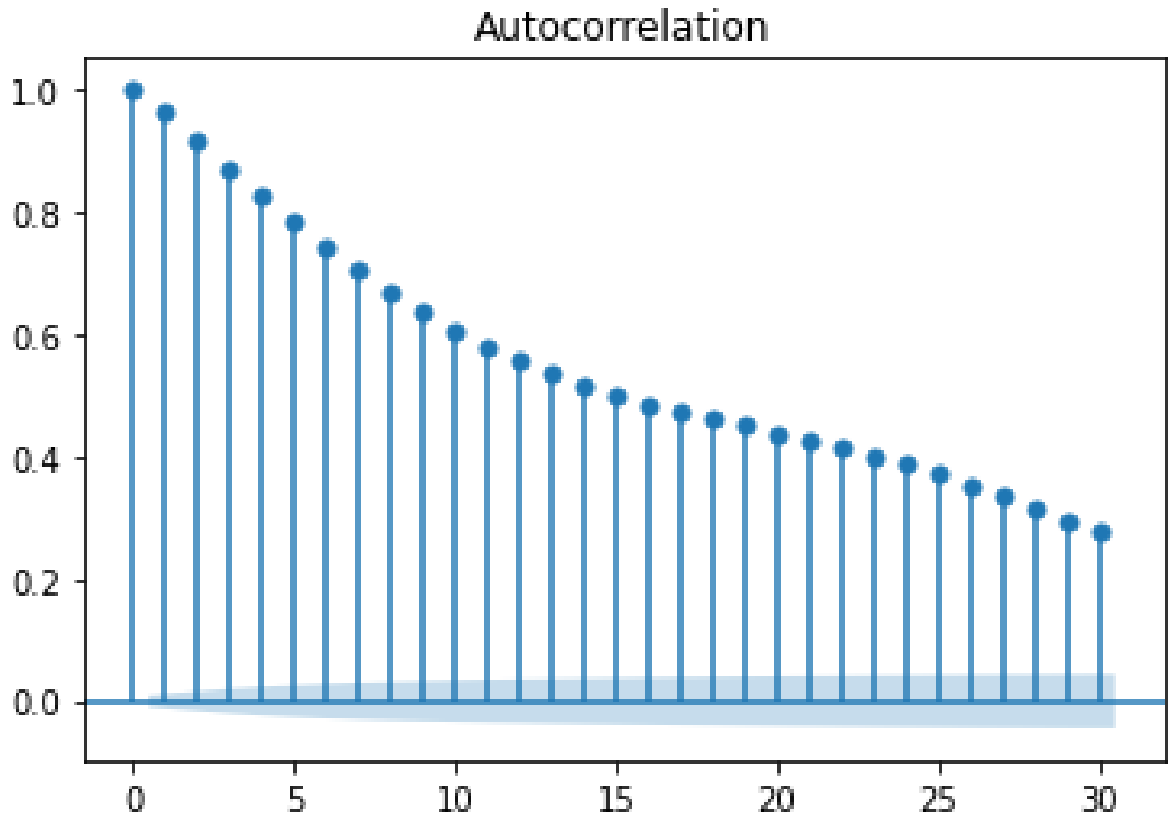
Building the Model

1. Make the series **stationary**
2. Use ACF and PACF plots to decide if you should include **AR** or **MA** terms in your model
3. Fit the model, and check residual ACF and PACF for lingering signals
4. If there are significant terms in residual ACF or PACF, add **AR** or **MA** terms, and try again

ARIMA Diagnostics - ACF

```python
# Generate plot from ACF

from statsmodels.graphics.tsaplots import plot_acf

fig = plot_acf(data['pm2.5'], lags=10)
fig.show()
```

ACF Plot

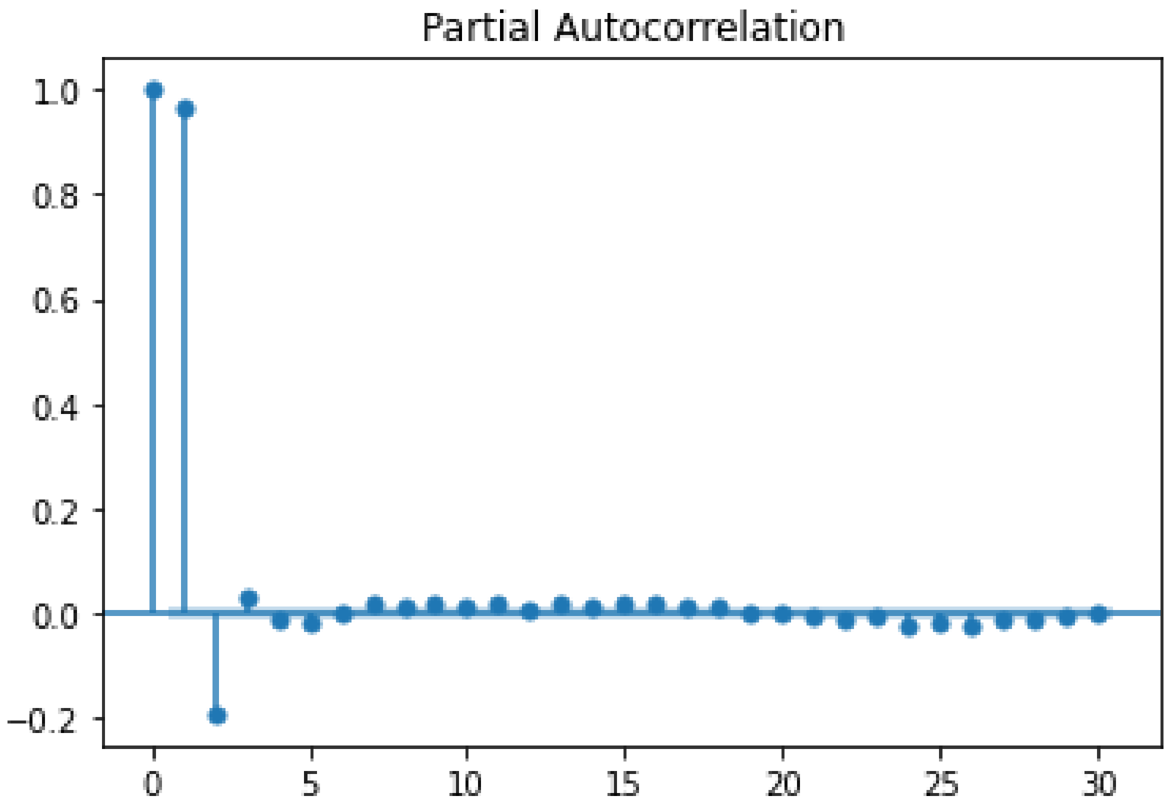


This is a clear indication that we do NOT have stationary data (yet)

ARIMA Diagnostics - PACF

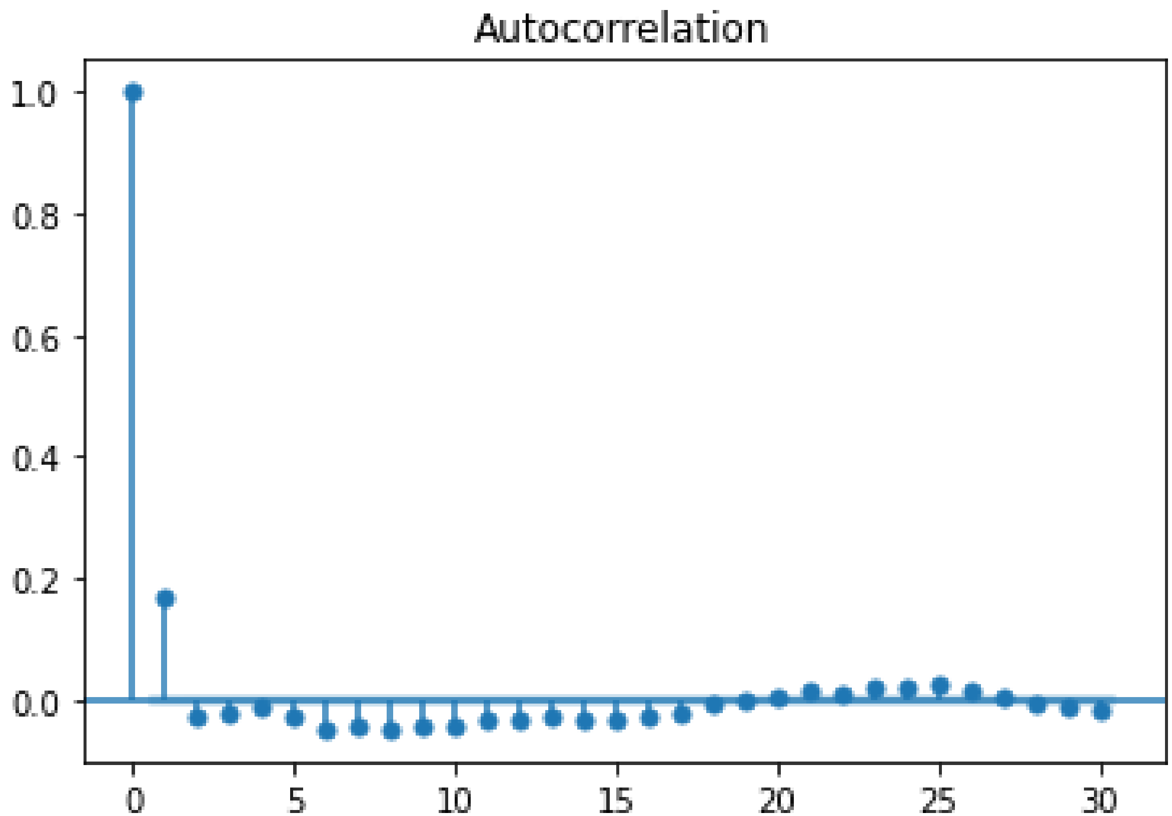
```
# Generate plot from PACF

fig = plot_pacf(data['pm2.5'], lags=10)
fig.show()
```

PACF Plot



Differenced ACF Plot

```
plt = sm.graphics.tsa.plot_acf(data['pm2.5'].diff().dropna(), lags=30)
plt.show()
```



Differencing our data can reduce the amount of structure that remains in the ACF.

Time to Model!

Once we have

- Utilized our ACF and PACF plots to diagnose our model
- Discovered the amount of differencing required by our data (to make our data stationary)

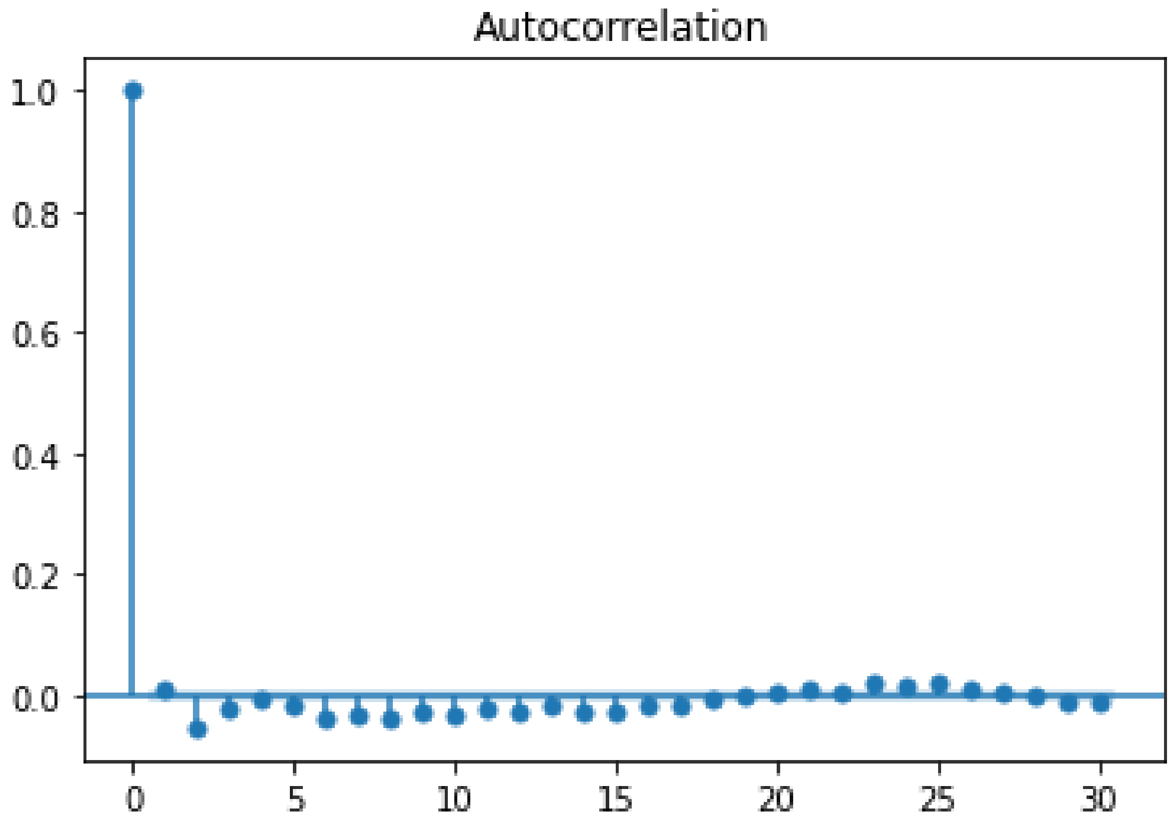
It is time to fit our model using the `arima` command we learned last week.

We can then validate our model by examining the residual ACF and PACF plots.

Residual ACF

```
import statsmodels.api as sm

arima = sm.tsa.ARIMA(data['pm2.5'], order=(1, 0, 0)).fit()

fig = plot_acf(arima.resid, lags=10)
fig.show()
```

Residual ACF

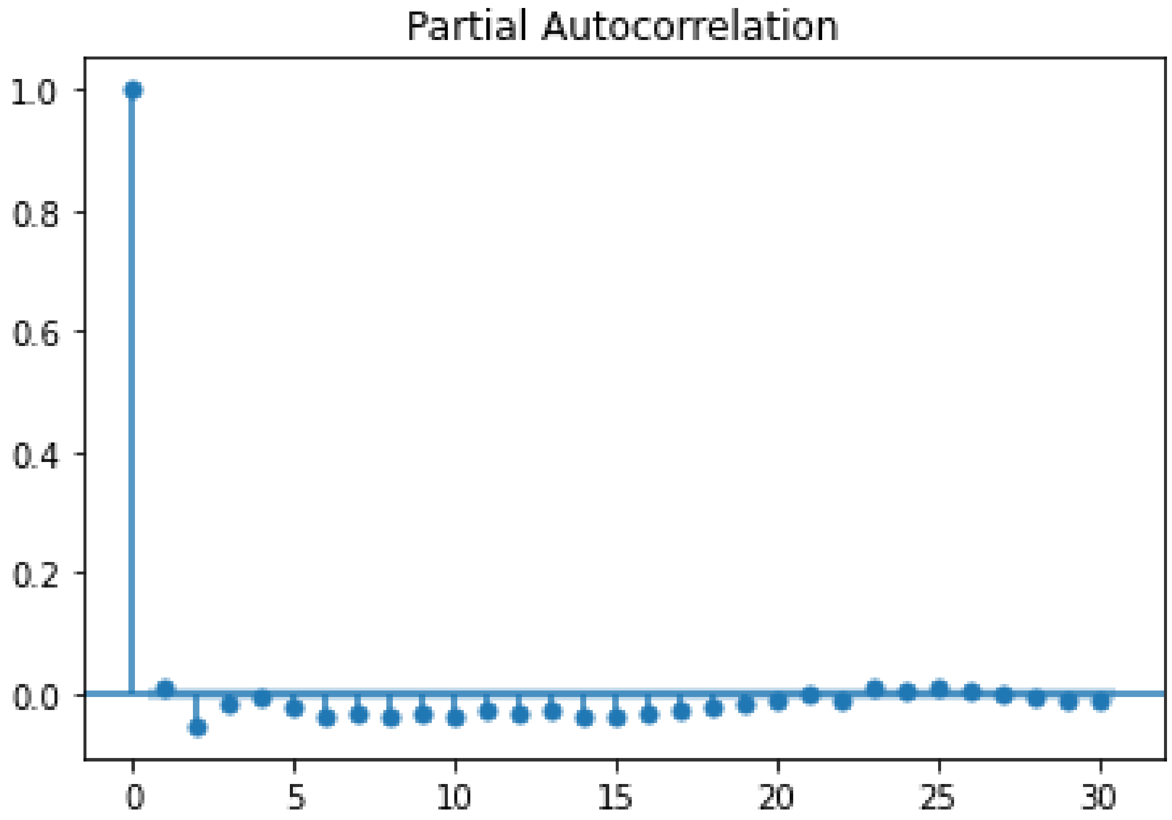


Residual ACF

```
fig = plot_pacf(arima.resid, lags=10)
fig.show()
```

Residual PACF

Nearly identical to the ACF plot (and is very small, cyclical)



Looking Ahead

Now that we have a fitted model, we can start to make predictions

```
fcst = reg.forecast(steps=10) # Generate forecast
```

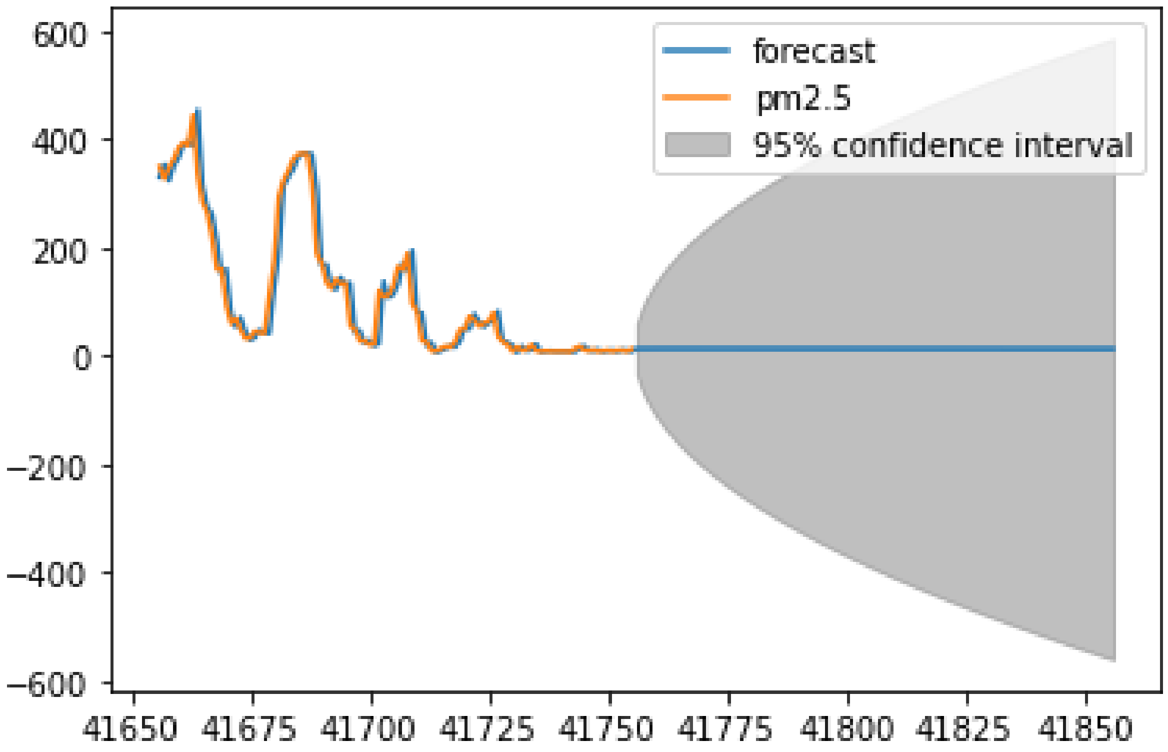
We make our out-of-sample forecast, and store it as an object. It contains three arrays:

1. The forecast
2. The standard errors
3. Upper and lower confidence intervals

Looking Ahead - Plotting

```
fig = reg.plot_predict(start=len(data)-100, end=len(data)+100)
fig.show()
```

Looking Ahead - Plotting



ARIMAX Models and Seasonal ARIMA models (SARIMAX)

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

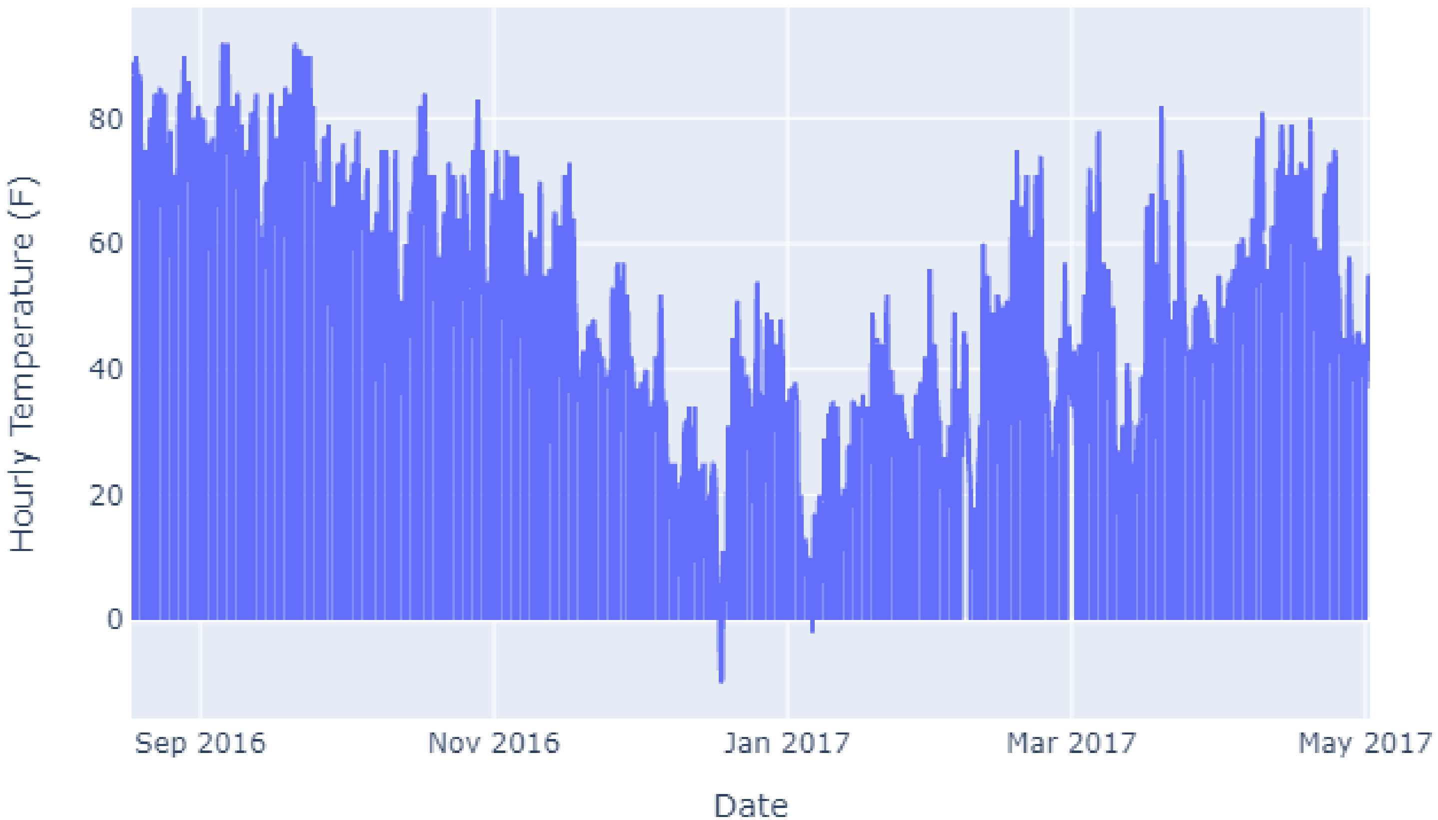
Let's use some weather data to get started:

```
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



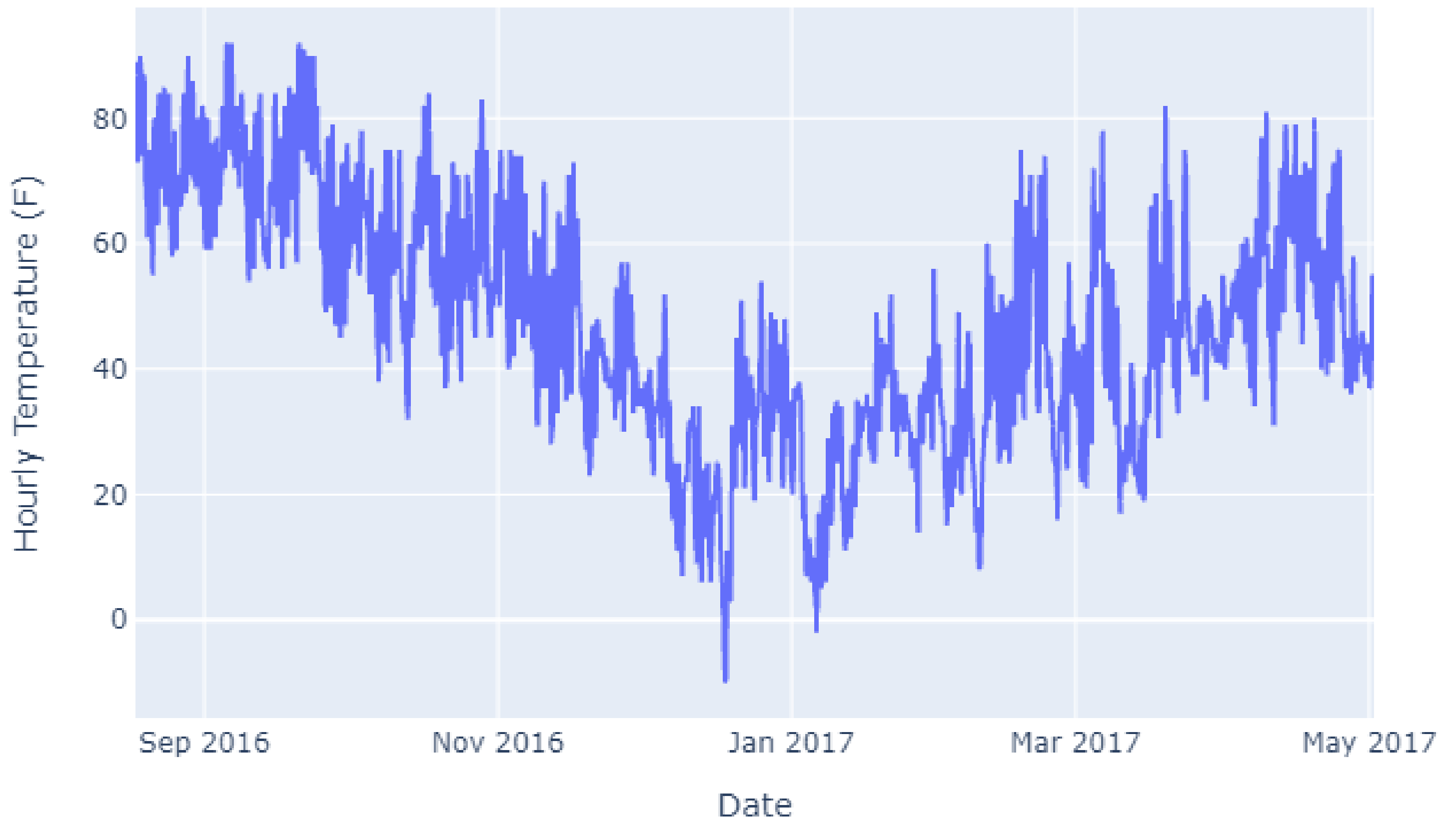
Hourly Temperature (F)
80
60
40
20
0
Sep 2016
Nov 2016
Jan 2017
Mar 2017
May 2017
Date

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(y, order=(1,1,0), exog=x)
reg = model.fit(trend='nc', method='mle',
                maxiter=500, solver='nm')
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.SARIMAX(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
```

Review

- We can use diagnostic plots to determine the order of our model, and to determine the processes involved (AR vs MA, etc.)
- ARIMAX allows for the use of exogenous variables to help explain our model
- SARIMAX adds seasonality to the model, allowing us to better account for cyclicality in our data.