

# DATA SOCIETY®

Time series day 4

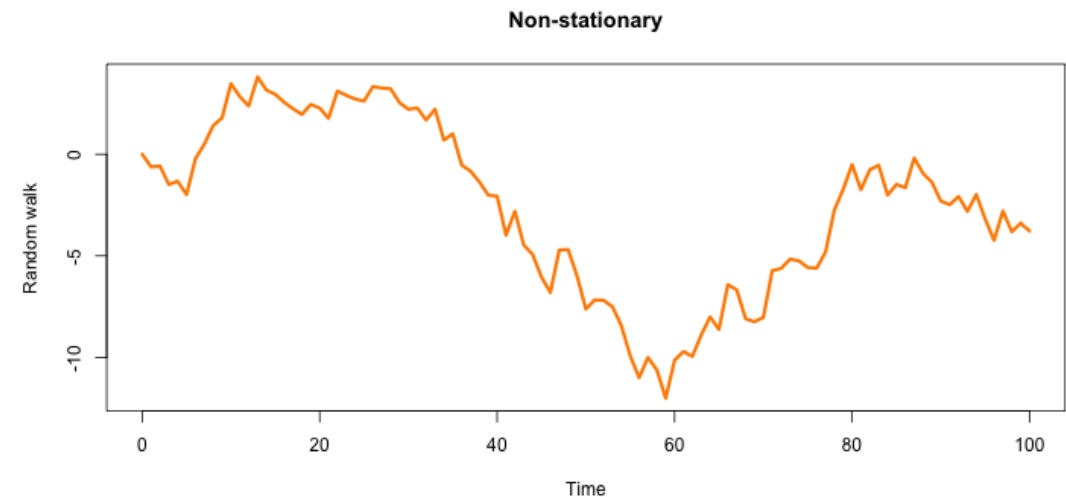
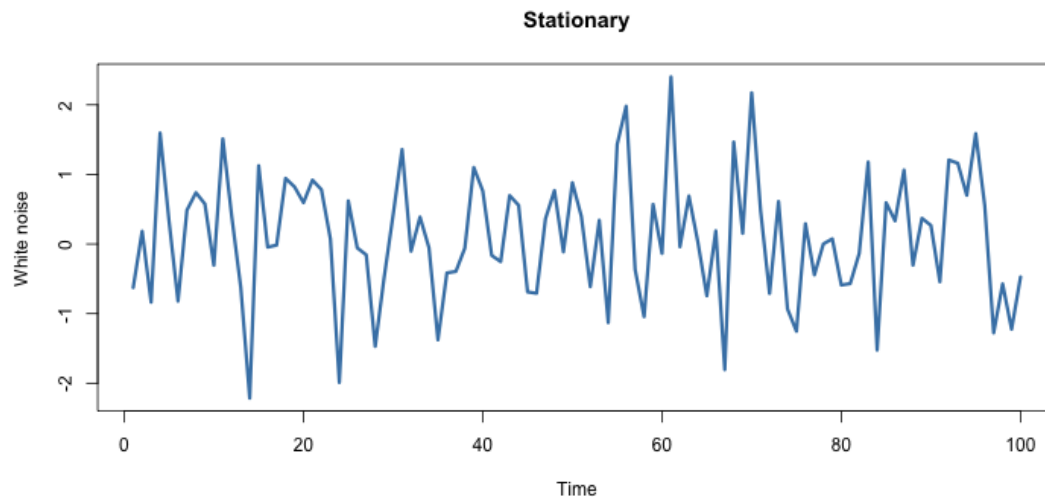
*"One should look for what is and not what he thinks should be."  
-Albert Einstein.*

# Module completion checklist

Objective	Complete
Review the concept of stationarity	
Test decomposed series for stationarity	
Describe the components of an ARIMA model	
Discuss special cases of ARIMA models	
Describe an autoregressive (AR) model	
Describe a moving average (MA) model	
Describe a non-seasonal ARIMA model	
Use non-seasonal ARIMA model to forecast using deseasonalized series	

# Recap: stationarity

- A **stationary** time series is one **whose properties do not depend on the time** at which the series is observed
- A *white noise* series is **stationary**, it does not matter when you observe it, it should look the same at any point in time
- A *random walk* series is **not stationary**, but it can be made stationary by using **differencing**



# Recap: additive model decomposition steps

- What do we do if we have a model that contains a *trend* and *seasonality*? Those two components most definitely make the series **non-stationary**
    - We can decompose the series and then detrend and deseasonalize it
    - We can use one of differencing methods
1. Compute the trend component  $T_t$  ✓
  2. Calculate the detrended series:  $Y_t - T_t$  ✓
  3. Estimate the seasonal component  $S_t$  ✓
  4. Calculate deseasonalized and detrended series (i.e. estimate the error term):  
 $\epsilon_t = Y_t - T_t - S_t$  ✓

# Recap: differencing

- **Differencing** is a method use to make non-stationary time series stationary
  - It is done by computing differences between consecutive observations
- A random walk model is stabilized and made stationary by taking first differences
  - First difference formula:  $\text{diff}(Y_t) = Y_t - Y_{t-1}$
  - It is also known as lag-1 difference

# Differencing: other types

- Sometimes the data is still non-stationary after taking first differences and it may be necessary to repeat the process again for the second time
  - This process is called **second order differencing**:  $\text{diff}^2(Y_t) = \text{diff}(Y_t) - \text{diff}(Y_{t-1})$
- To make seasonal data stationary, **seasonal differences** are taken
  - It is the difference between an observation and the previous observation from the same season:  $\text{diff}_S(Y_t) = Y_t - Y_{t-m}$ , where  $m$  is the number of seasons
  - It is also known as lag- $m$  difference
- Sometimes differencing techniques are combined (e.g. if seasonal difference doesn't make data stationary, then first difference can be applied)

# Stationarity, why bother?

- Because of *its properties*
  - A **stationary** time series has **constant mean, variance, autocorrelation**, etc. that do not change with time
  - A stationarized series is **easy to predict**: you simply predict that its statistical properties will be the same in the future as they have been in the past!
  - The **transformations** on predictions for the stationarized series **can then be reversed** to obtain predictions for the original series
- Because it is *required by forecasting models*
  - Most forecasting methods are based on the **assumption that the time series can be made stationary** through a host of mathematical transformations (e.g. differencing)
  - Finding the **sequence of transformations needed to stationarize a time series** often provides important clues **in the search for an appropriate forecasting model**
  - Stationarizing a time series through differencing (where needed) is an important part of the process of fitting an ARIMA model

# Forecasting expenditures

- Time series models are widely used to anticipate future demands and trends
- By more accurately anticipating events, we can improve efficiencies, prevent shortages, and stock supplies
- **How could these types of predictions help you?**



Economic Modelling  
Volume 28, Issue 3, May 2011, Pages 1068-1077



## Optimal military spending in the US: A time series analysis

G. d'Agostino <sup>a, b</sup>, J.P. Dunne <sup>c</sup>, L. Pieroni <sup>a, b</sup>

Show more

<https://doi.org/10.1016/j.econmod.2010.11.021>


[Get rights and content](#)

### Abstract

This paper extends previous work on the optimal size of government spending by including nested functional decompositions of military spending into consumption and investment. Post World War II US data are then used to estimate nested non-linear growth models using semi-parametric methods. As expected, investments in military and non-military expenditure are both found to be productive expenditures with respect to the private production. Moreover there is little evidence to suggest that current military spending is having a negative impact on economic growth in the US, while civilian consumption only tends to have only a weak impact. This does not imply that society will



# Module completion checklist

Objective	Complete
Review the concept of stationarity	
Test decomposed series for stationarity	
Describe the components of an ARIMA model	
Discuss special cases of ARIMA models	
Describe an autoregressive (AR) model	
Describe a moving average (MA) model	
Describe a non-seasonal ARIMA model	
Use non-seasonal ARIMA model to forecast using deseasonalized series	

# Import packages

- Let's import the libraries we will be using today

```
import os
import pandas as pd
from pandas.plotting import lag_plot
import numpy as np
import pickle
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
from statsmodels.tsa.stattools import acf
from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.tsa.arima_model import ARIMA
from sklearn.metrics import mean_squared_error
from math import sqrt
```

# Directory settings

- In order to maximize the efficiency of your workflow, you should encode your directory structure into variables
- Let the `main_dir` be the variable corresponding to your `af-werx` folder

```
# Set `main_dir` to the location of your `af-werx` folder (for Linux).  
main_dir = "/home/[username]/Desktop/af-werx"
```

```
# Set `main_dir` to the location of your `af-werx` folder (for Mac).  
main_dir = "/Users/[username]/Desktop/af-werx"
```

```
# Set `main_dir` to the location of your `af-werx` folder (for Windows).  
main_dir = "C:\\Users\\[username]\\Desktop\\af-werx"
```

```
# Make `data_dir` from the `main_dir` and  
# remainder of the path to data directory.  
data_dir = main_dir + "/data"
```

# Working directory

- Set working directory to the `data_dir` variable we set
- We do this using the `os.chdir` function, change directory
- We can then check the working directory using `.getcwd()`
- For complete documentation of the `os` package, [click here](#)

```
# Set working directory.  
os.chdir(data_dir)
```

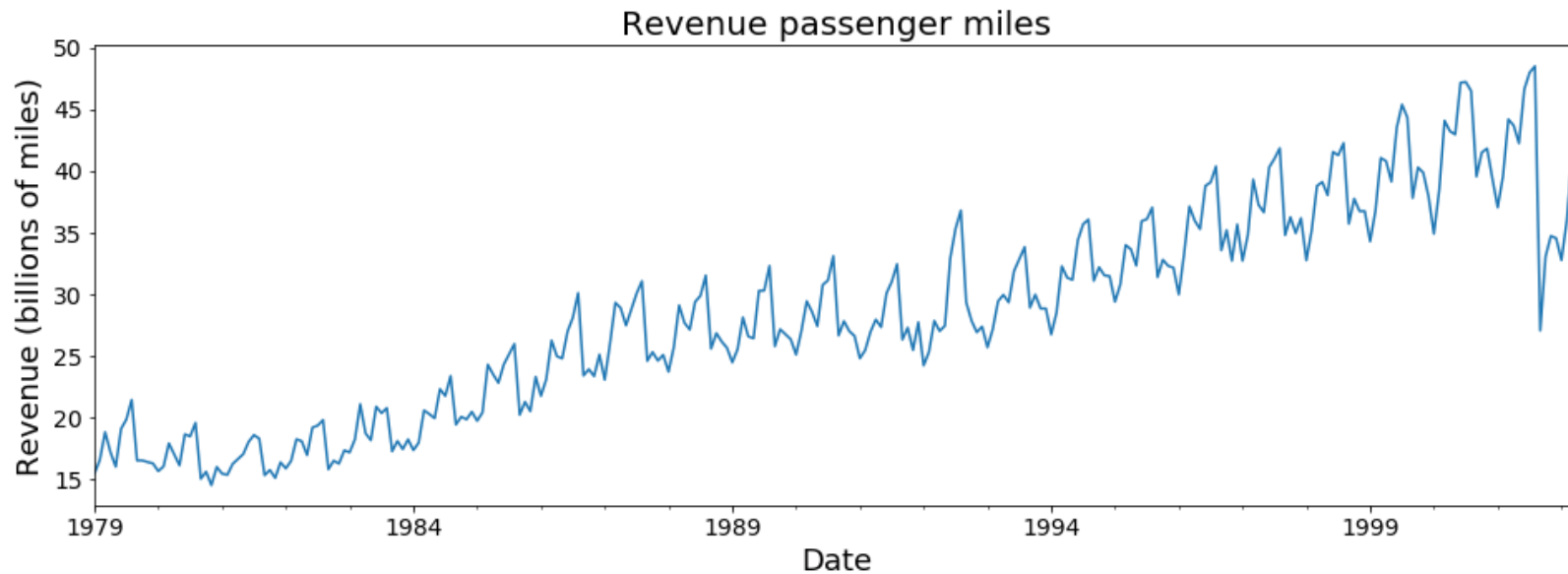
```
# Check working directory.  
print(os.getcwd())
```

```
/home/[user-name]/af-werx/data
```

# Load passenger miles dataset

```
# Read pickle file into `passenger_miles` variable.  
passenger_miles = pickle.load(open((data_dir + "/passenger_miles.sav"), "rb"))  
print(passenger_miles.head())
```

date	revenue_passenger_miles	available_seat_miles	unused_seat_miles
1979-01-01	15.50	26.64	11.15
1979-02-01	16.58	27.20	10.62
1979-03-01	18.85	27.87	9.02
1979-04-01	17.23	23.22	5.99
1979-05-01	16.04	23.27	7.23



# Recap: stationarity test

- The ADF test will test for stationarity by looking for a **unit root**, where:

$H_0$  : there is a unit root

$H_a$  : the time series is stationary

- Unit roots, at a high level, can be a signal of randomness in a model and result in spurious regressions and errant behavior

```
# Perform ADF test on original series.  
result_pm =  
adfuller(passenger_miles['revenue_passenger_miles'])
```

- The p-value is very high and we fail to reject  $H_0$
- Our original data is **not stationary**

```
print('ADF Statistic: %f' %  
      result_pm[0])
```

ADF Statistic: -1.541166

```
print('p-value: %f' % result_pm[1])
```

p-value: 0.513058

```
print('Critical Values:')
```

Critical Values:

```
for key, value in  
    result_pm[4].items():  
    print('\t%s: %.3f' % (key, value))
```

1%: -3.455  
5%: -2.873  
10%: -2.573

# Test deseasonalized series for stationarity

- What happens if we test deseasonalized series for stationarity?
- Let's decompose the series as we did in the previous module
- Recall that we can get deseasonalized series by simply subtracting the seasonal component from the original series

$$Y_t - S_t = T_t + \epsilon_t$$

```
# Decompose revenue passenger miles into its components.  
res = seasonal_decompose(passenger_miles['revenue_passenger_miles'])
```

```
# Deseasonalized series ( $Y_t - S_t = T_t + e_t$ ).  
deseasonalized = passenger_miles['revenue_passenger_miles'] - res.seasonal
```

# Test deseasonalized series for stationarity

```
# Perform ADF test on  $Y_t - T_t - S_t$  series.  
result_pm_deseason = adfuller(deseasonalized)  
  
print('ADF Statistic: %f' % result_pm_deseason[0])
```

ADF Statistic: -1.762717

```
print('p-value: %f' % result_pm_deseason[1])
```

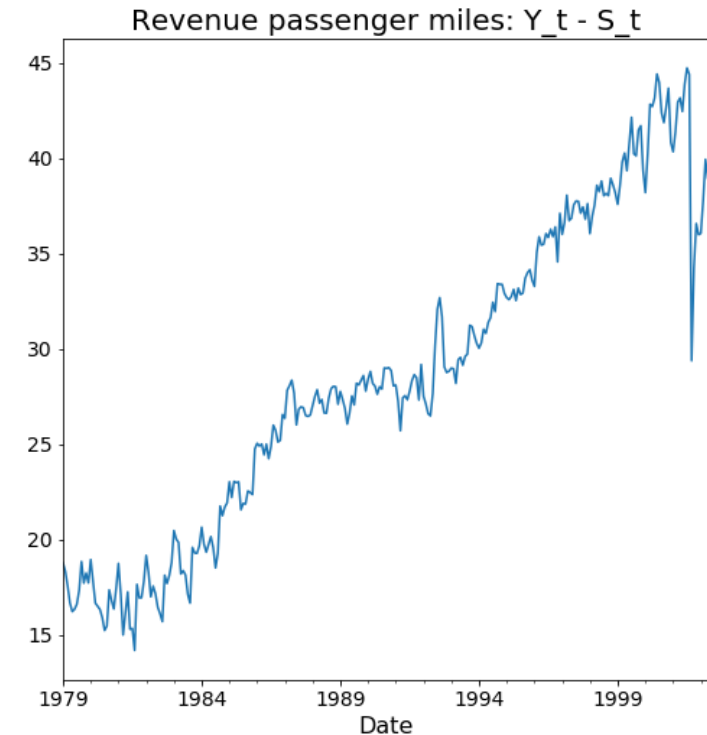
p-value: 0.399122

```
print('Critical Values:')
```

Critical Values:

```
for key, value in result_pm_deseason[4].items():  
    print('\t%s: %.3f' % (key, value))
```

1%: -3.455  
5%: -2.873  
10%: -2.573



- The plot above and the results of the ADF test show that the deseasonalized series is non-stationary.



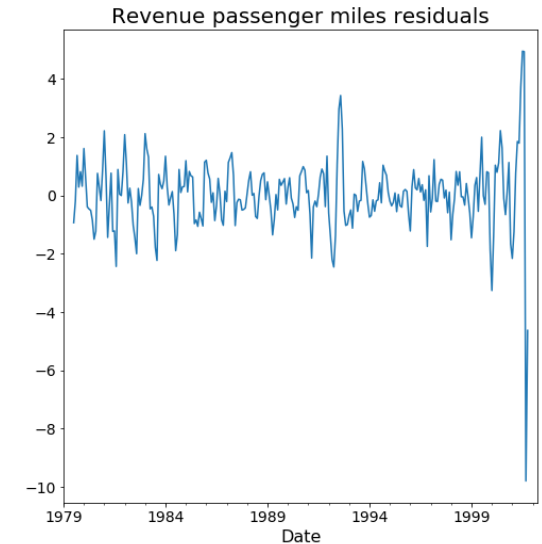
# Test decomposed series for stationarity

- Seasonal data that has a trend is definitely non-stationary
- By removing both the trend and seasonal components from the series, we ideally should have only white noise left:

$$\epsilon_t = Y_t - T_t - S_t$$

```
print(res.resid.head(10))
```

```
date
1979-01-01      NaN
1979-02-01      NaN
1979-03-01      NaN
1979-04-01      NaN
1979-05-01      NaN
1979-06-01      NaN
1979-07-01    -0.930407
1979-08-01    -0.190135
1979-09-01     1.378217
1979-10-01     0.291006
Name: revenue_passenger_miles, dtype: float64
```



# Test decomposed series for stationarity

```
# Remove NaN values
# (the test won't work otherwise).
residuals = res.resid[~ np.isnan(res.resid)]
print(residuals.head(10))
```

```
date
1979-07-01    -0.930407
1979-08-01    -0.190135
1979-09-01     1.378217
1979-10-01     0.291006
1979-11-01     0.820806
1979-12-01     0.320939
1980-01-01     1.615825
1980-02-01     0.638912
1980-03-01    -0.388095
1980-04-01    -0.458209
Name: revenue_passenger_miles, dtype: float64
```

- The ADF test statistic is way below any critical value and the p-value is nearly 0
- The deseasonalized and detrended series are indeed stationary!

```
# Perform ADF test on  $Y_t - T_t - S_t$  series.
result_pm_resid = adfuller(residuals)

print('ADF Statistic: %f' % result_pm_resid[0])
```

```
ADF Statistic: -7.069110
```

```
print('p-value: %f' % result_pm_resid[1])
```

```
p-value: 0.000000
```

```
print('Critical Values:')
```

```
Critical Values:
```

```
for key, value in result_pm_resid[4].items():
    print('\t%s: %.3f' % (key, value))
```

```
1%: -3.456
5%: -2.873
10%: -2.573
```

# Module completion checklist

Objective	Complete
Review the concept of stationarity	✓
Test decomposed series for stationarity	✓
Describe the components of an ARIMA model	
Discuss special cases of ARIMA models	
Describe an autoregressive (AR) model	
Describe a moving average (MA) model	
Describe a non-seasonal ARIMA model	
Use non-seasonal ARIMA model to forecast using deseasonalized series	

# ARIMA models: components

- When talking about time series forecasting, ARIMA models come up a lot and are a de facto standard of time series analysis and forecasting
- ARIMA models are a **generalized set of models** that include the following components:
  - *Autoregressive* (AR): lags of the stationarized series are called “autoregressive” terms
  - *Integrated* (I): a series which needs to be differenced to be made stationary is an “integrated” series
  - *Moving average* (MA): lags of the forecast errors are called “moving average” terms

# ARIMA models: a systematic set of rules

- We went through techniques and components that comprise ARIMA in earlier modules:
  - Differencing, moving averages, and lagged values of the dependent variable
- The **use of these methods** and combination of the components and parameters is **determined by a systematic set of rules** we need to follow
- We will discuss the individual ARIMA components as well as some of the rules in the following slides

# ARIMA syntax

- A non-seasonal ARIMA model can be summarized by three numbers:
  - $p$ : the number of autoregressive terms (AR)
  - $d$ : the number of nonseasonal differences (I)
  - $q$ : the number of moving-average terms (MA)
- The above components make up the ARIMA model:  $ARIMA(p, d, q)$

# Knowledge check 1



# Exercise 1





# Module completion checklist

Objective	Complete
Review the concept of stationarity	✓
Test decomposed series for stationarity	✓
Describe the components of an ARIMA model	✓
Discuss special cases of ARIMA models	
Describe an autoregressive (AR) model	
Describe a moving average (MA) model	
Describe a non-seasonal ARIMA model	
Use non-seasonal ARIMA model to forecast using deseasonalized series	

# The nature of ARIMA models

- A stationary series
  - Has a constant mean
  - Has consistent variation around that mean
  - Is a combination of **signal** and **noise**:  $Y_t = signal_t + noise_t$
- An ARIMA forecasting equation is linear for stationary series:
  - It acts as a **filter** that separates **signal** from **noise**
  - It is a weighted sum of one or more recent values of  $Y$  and/or a weighted sum of one or more recent values of the errors  $\epsilon_t$

# Special cases of ARIMA models

model	ARIMA equivalent
White noise	ARIMA(0, 0, 0)
Random walk	ARIMA(0, 1, 0) with no constant
Random walk with drift	ARIMA(0, 1, 0) with a constant
Autoregression	ARIMA(p, 0, 0)
Moving average	ARIMA(0, 0, q)

# Special cases of ARIMA models: white noise

- The **white noise** model is equivalent to  $ARIMA(0, 0, 0)$ , which translates into
  - $p = 0$  - no autoregressive components, the observations are not correlated
  - $d = 0$  - no differencing components, the time series is already stationary
  - $q = 0$  - no moving average components, the time series has a constant mean

# Special cases of ARIMA models: random walk

- The **random walk** model is equivalent to  $ARIMA(0, 1, 0)$ , which translates into
  - $p = 0$  - no autoregressive components, the observations are not correlated after the first difference has been taken
  - $d = 1$  - a differencing component equal to 1, because we need to take the first difference to make the time series stationary
  - $q = 0$  - no moving average components, the time series has a constant mean after the first difference has been taken

# Module completion checklist

Objective	Complete
Review the concept of stationarity	✓
Test decomposed series for stationarity	✓
Describe the components of an ARIMA model	✓
Discuss special cases of ARIMA models	✓
Describe an autoregressive (AR) model	
Describe a moving average (MA) model	
Describe a non-seasonal ARIMA model	
Use non-seasonal ARIMA model to forecast using deseasonalized series	

# Special cases of ARIMA models: autoregression

- An autoregressive model or autoregression (AR) is also a special case of an *ARIMA*( $p, 0, 0$ )
- It's used for time series where **signal** in our data can be explained by simply adding in the autoregressive components  $p$  to the modeling equation
- It does not need to have the other two parts of the *ARIMA*

# Multiple regression vs autoregression

- What does an autoregressive model look like?
- It looks very similar to a **multiple regression** model, where  $X_i$  is a predictor variable,  $C$  is some constant term,  $\phi_i$  is some function applied to the corresponding predictor, and  $\epsilon$  is the error term

$$Y = C + \phi_1 X_1 + \phi_2 X_2 + \dots + \phi_p X_p + \epsilon$$

- The only difference between an **autoregression** model and the above model is the **auto** nature of the regression:
  - Instead of  $X_i$  is a predictor variable, we use  $Y_i$  - past values of the  $Y$  variable itself

$$Y_t = C + \phi_1 Y_{t-1} + \phi_2 Y_{t-2} + \dots + \phi_p Y_{t-p} + \epsilon_t$$



# AR(p)

- The equation below represents an **autoregressive model of order  $p$** , because it contains  **$p$  lagged values** of  $Y$

$$Y_t = C + \phi_1 Y_{t-1} + \phi_2 Y_{t-2} + \dots + \phi_p Y_{t-p} + \epsilon_t$$

- The shorthand notation is simply  $AR(p)$
- In ARIMA terms, it would be  $ARIMA(p, 0, 0)$

# AR(p): interpretation

- The interpretation of  $AR(p)$  is fairly simple:
  - The model takes into account only  $p$  previous values of  $Y$  to predict  $Y_t$
  - $AR(1)$  simply uses a multiple of its own previous value  $Y_{t-1}$ , plus a constant to predict  $Y_t$

# AR(1): broken down

- This is an equation of  $AR(1)$  model

$$Y_t = C + \phi_1 Y_{t-1} + \epsilon_t$$

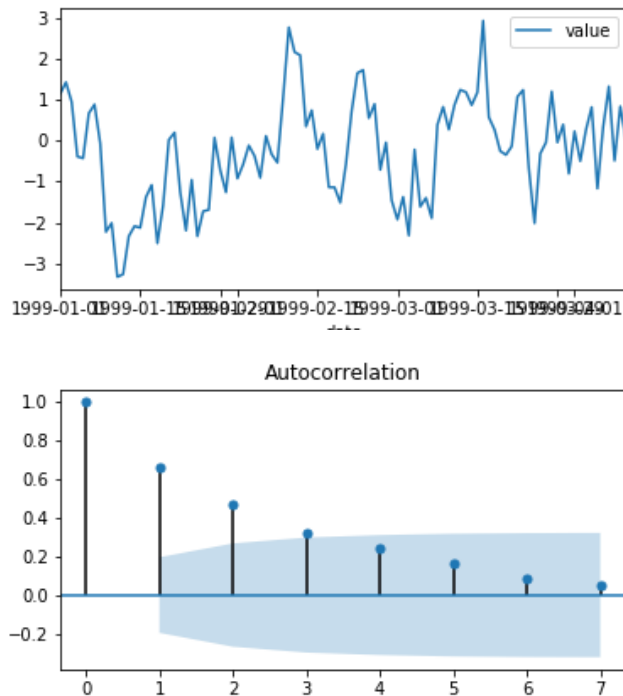
- Autoregressive models of order **1** are surprisingly flexible at handling a wide range of different time series patterns and all thanks to the  $\phi_1$ !
- Changing the parameter  $\phi_1$  results in different time series patterns
  - If  $\phi_1 = 0$ , we have our white noise model
  - If  $\phi_1 = 1$  and  $C = 0$ , we have our random walk model
  - If  $\phi_1 < 0$ ,  $Y_t$  oscillates around the mean
  - The range of  $\phi_1$  is usually restricted to be between  $-1$  and  $1$
- The error term  $\epsilon_t$  will only change the scale of the series, but not the general patterns

# AR(1): assumptions and properties

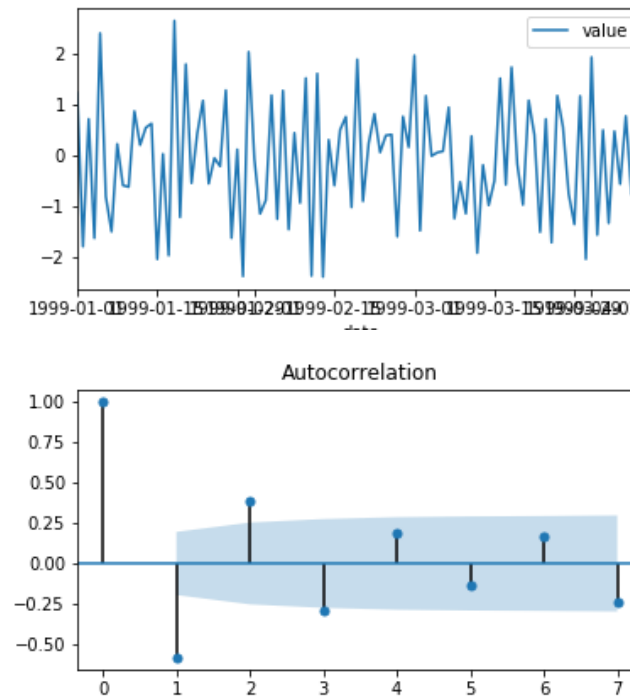
- In order to see whether this time series can be modeled with **AR(1)**, it needs to fit a set of **assumptions** (just like a linear model!)
  - The errors  $\epsilon_t$  are independent - they form a normal distribution with  $\mu = 0$  and constant  $\sigma_{\epsilon_t}^2$
  - Properties of the errors  $\epsilon_t$  are independent of  $Y_t$
  - The series is stationary and  $\phi_1$  falls within  $(-1, 1)$  range
- If the above assumptions are fulfilled, the series will have some nice **properties**
  - The slope of the **AR(1)** model is just  $\phi_1$  and it is also autocorrelation of  $\text{lag} = 1$
- The ACF plot has **distinct pattern**:
  - For a positive value of  $\phi_1$ , the ACF exponentially decreases to **0** as the lag increases
  - For negative  $\phi_1$ , the magnitude of ACF also exponentially decreases to **0** as the lag increases, but the signs alternate

# AR(1): how to recognize it?

- This series shows an exponentially decreasing ACF, which indicates that  $0 < \phi_1 < 1$



- This series shows an exponentially decreasing ACF with alternating sign ( $-1 < \phi_1 < 0$ )

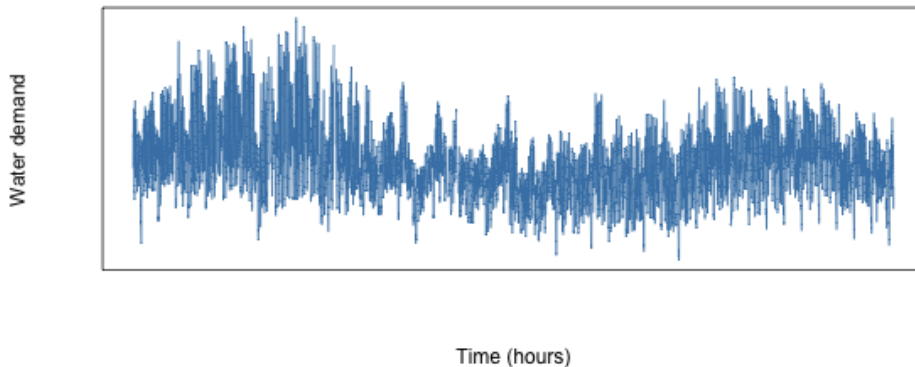


# Going from one to many

- Going *from one* AR component in the model equation *to many* is **very similar to going from a simple linear regression to multiple regression**
- The **logic of the assumptions still applies**, but the complexity increases with every added lagged value of  $Y_t$

# AR(p) model real-world applications

- How much water does a community use? What are the peak times? What's the best time to start construction to impact the smallest amount of customers?
- This study investigates the use of a double-seasonal time series model to capture daily and weekly autocorrelations to both total and regional demands [see publication](#)
- AR models can represent a very wide range of patterns, here are some more examples of applications that use AR models on time series data from very different fields of work
  - Study of volcanic tremor and long-period events, [see publication](#)
  - Representing the long-term amplitude modulations of speech/audio signals, [see publication](#)
  - Pattern discrimination of brain electrical activity mapping, [see publication](#)
  - Many many more ...



# Module completion checklist

Objective	Complete
Review the concept of stationarity	✓
Test decomposed series for stationarity	✓
Describe the components of an ARIMA model	✓
Discuss special cases of ARIMA models	✓
Describe an autoregressive (AR) model	✓
Describe a moving average (MA) model	
Describe a non-seasonal ARIMA model	
Use non-seasonal ARIMA model to forecast using deseasonalized series	



# Special cases of ARIMA models: moving average

- A moving average model (MA) is also a special case of an  $ARIMA(0, 0, q)$
- It's used for time series where the **signal** in our data can be explained by simply adding in the moving average components  $q$  to the modeling equation
- It does not need to have the other two parts of the  $ARIMA$

# AR vs MA

- How does a moving average model look then?
- It looks very similar to an **autoregressive** model, where  $Y_i$  - past values of the  $Y$  variable itself,  $C$  is some constant term,  $\phi_i$  is some function applied to the corresponding predictor, and  $\epsilon$  is the error term

$$Y_t = C + \phi_1 Y_{t-1} + \phi_2 Y_{t-2} + \dots + \phi_p Y_{t-p} + \epsilon_t$$

- The only difference between **moving average** model and the above model is the  $\epsilon_t$ :
  - Instead of past values of  $Y_i$  is predictor variables, we use past forecast errors

$$Y_t = C + \epsilon_t + \theta_1 \epsilon_{t-1} + \theta_2 \epsilon_{t-2} + \dots + \theta_q \epsilon_{t-q}$$

# MA(q)

- The equation below represents an **moving average model of order  $q$** , because it contains  **$q$  past values of  $\epsilon$**

$$Y_t = C + \epsilon_t + \theta_1 \epsilon_{t-1} + \theta_2 \epsilon_{t-2} + \dots + \theta_q \epsilon_{t-q}$$

- The shorthand notation is simply  $MA(q)$
- In ARIMA terms, it would be  $ARIMA(0, 0, q)$

# MA(q): interpretation

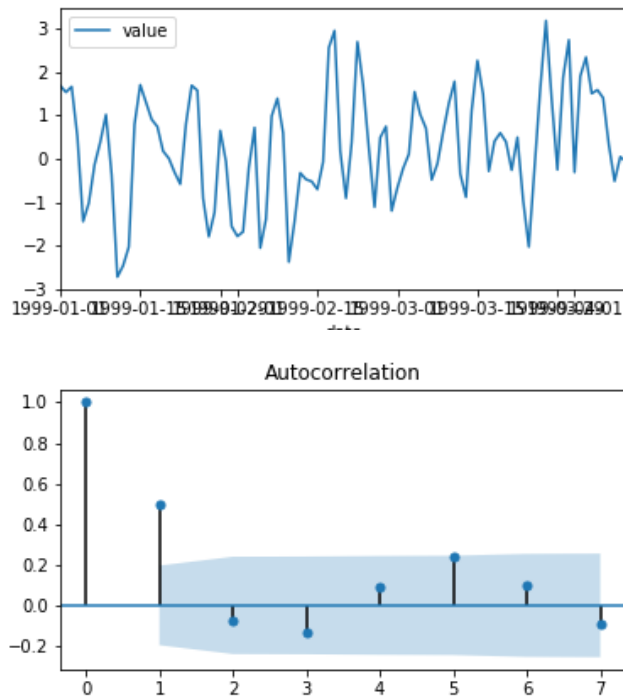
- The interpretation of  $MA(q)$  is not as intuitive as the  $AR(p)$  model interpretation, since we are using the **errors of previous forecast values for prediction** and those errors technically were not observed yet
- We look at **every value of  $Y_t$**  as **a weighted moving average of the past few forecast errors**
  - Calculate your forecast for period 1
  - Subtract the forecast value from the actual value for that period to generate the error for period 1
  - Use that error for period 1 to calculate the forecast for period 2, and so on
- *Note:  $MA$  model is NOT the same as  $MA$  smoothing, the  $MA(q)$  is used for predictions and  $MA$  smoothing is used to estimate the trend-cycle component of the series!*

# MA(1): assumptions and properties

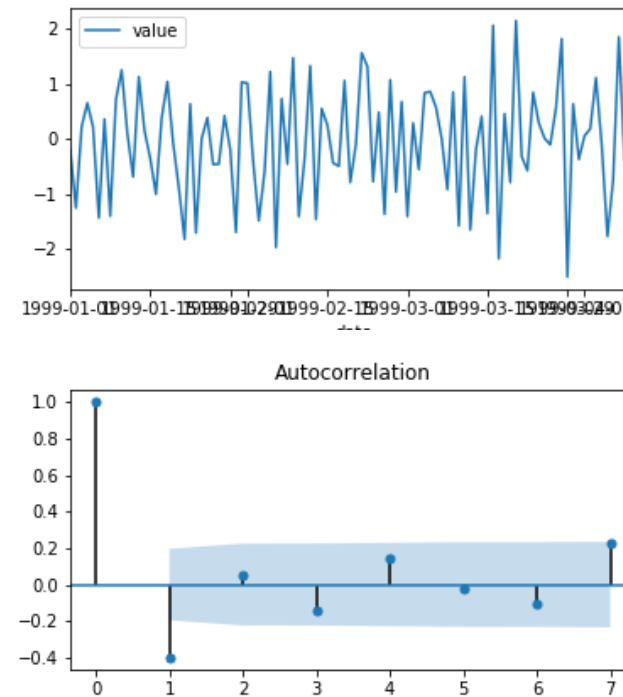
- In order to see whether a time series can be modeled with **MA(1)**, it needs to fit a set of **assumptions** (just like a linear model!)
  - The errors  $\epsilon_t$  are independent, they form a normal distribution with  $\mu = 0$  and constant  $\sigma_{\epsilon_t}^2$
  - A moving average term in a time series model is a past error multiplied by a coefficient
  - The series is stationary and  $\theta_1$  falls within  $(-1, 1)$  range
- If the above assumptions are fulfilled, the series will have some nice **properties**
  - The mean of the  $Y_t$  model is just  $C$
  - The variance of  $Y_t$  is  $\sigma^2 = \sigma_{\epsilon_t}^2(1 + \theta_1^2)$
- The ACF plot has **distinct pattern**:
  - The only nonzero value in the ACF is for lag = 1, all other autocorrelations should be 0

# MA(1): how to recognize it?

- This series shows ACF at  $\text{lag} = 1$  as significant for  $0 < \theta_1 < 1$ , while the remaining autocorrelations drop within the 95% confidence interval



- This series shows ACF at  $\text{lag} = 1$  as significant for  $-1 < \theta_1 < 0$ , while the remaining autocorrelations drop within the 95% confidence interval



# Going from one to many

- Similar to the AR model with 2+ components, going *from one* MA component in the model equation *to many* is **very similar to going from a simple linear regression to multiple regression**
- The **logic of the assumptions still applies**, but the complexity increases with every added previous forecast error term  $\epsilon_t$ 
  - The ACF for  $MA(q)$  will have significant autocorrelation values for the first  $q$  lags, and the remaining ones will be close to 0

# MA(q) model real-world applications

- MA models can represent a very wide range of patterns, here are some examples of applications that use MA models on time series data from very different fields of work
  - Forecasting Chinese consumer price index and German egg prices, *see publication*
  - Predicting the number of tourists, *see publication*
  - And more ...



# Knowledge check 2



# Module completion checklist

Objective	Complete
Review the concept of stationarity	✓
Test decomposed series for stationarity	✓
Describe the components of an ARIMA model	✓
Discuss special cases of ARIMA models	✓
Describe an autoregressive (AR) model	✓
Describe a moving average (MA) model	✓
Describe a non-seasonal ARIMA model	
Use non-seasonal ARIMA model to forecast using deseasonalized series	

# The ARIMA model equation

- As we have discussed earlier, an ARIMA model is a weighted sum of one or more recent values of  $Y_t$  and/or a weighted sum of one or more recent values of the errors  $\epsilon_t$
- More precisely, it is a sum of
  - A constant  $C$
  - A weighted sum of **AR** components that have  $p$  lagged values of  $Y_t$
  - A weighted sum of **MA** components that have  $q$  past forecast errors  $\epsilon_t$

$$Y_t = C + \sum_p AR_Y + \sum_q MA_\epsilon$$

# General steps to construct an ARIMA model

1. Test the series for stationarity and stationarize it if needed
  - i. Add the ***I*** (differencing) component to the ARIMA model
2. Inspect autocorrelations (and partial autocorrelations) to determine if lags need to be included in the forecasting equation
  - i. Add the ***AR*** and ***MA*** components to the ARIMA model
3. Fit and assess the model
4. Patterns that remain may suggest the need for additional AR or MA terms

# Constructing ARIMA model: pick differencing (I)

First, pick the *I* component with parameter *d* (the differencing component) of the *ARIMA*(*p*, *d*, *q*)

- If  $d = 0$ , then

$$d_0(Y_t) = Y_t$$

- If  $d = 1$ , then

$$d_1(Y_t) = Y_t - Y_{t-1}$$

- If  $d = 2$ , then

$$d_2(Y_t) = (Y_t - Y_{t-1}) - (Y_{t-1} - Y_{t-2}) = Y_t - 2Y_{t-1} + Y_{t-2}$$

- And so on

# Constructing ARIMA model: pick differencing (I)

- We will use deseasonalized series of revenue miles data for our ARIMA example
- Recall that we have tested our deseasonalized series for stationarity and we have concluded that **it is not stationary**
- We have also used  $d = 1$  last week and discovered that it was sufficient for our series to become stationary
- Let's stick to this **order of differencing** and set the  $I$  component to  $d = 1$

# Differencing: rules of thumb

- If the series has **positive autocorrelations out to a high number of lags**, then it probably **needs a higher order** of differencing
- If autocorrelation at  $\text{lag} = 1$  is **zero or negative**, or the autocorrelations are all **small and patternless**, then the series **does not need a higher order of differencing**
- If autocorrelation at **lag = 1 is -0.5 or more negative**, the series may be **overdifferenced!**
- The *optimal order of differencing* is often the order of differencing *at which the standard deviation is lowest*
  - For example, if you fit an  $ARIMA(0, 0, 0)$  model, an  $ARIMA(0, 1, 0)$  model, and an  $ARIMA(0, 2, 0)$  model, then the RMSE's will be equal to the standard deviations of the original series with **0**, **1**, and **2** orders of nonseasonal differencing, respectively

# General steps to construct an ARIMA model

1. Test the series for stationarity and stationarize it if needed ✓
  - i. Add the ***I*** (differencing) component to the ARIMA model ✓
2. Inspect autocorrelations (and partial autocorrelations) to determine if lags need to be included in the forecasting equation
  - i. Add the ***AR*** and ***MA*** components to the ARIMA model
3. Fit and assess the model
4. Patterns that remain may suggest the need for additional AR or MA terms



# Constructing ARIMA model: pick AR and MA

- ACF plot shows the relationship between  $Y_t$  and  $Y_{t-k}$  for different values of the lag  $k$
- Intuition tells us that adding as many **AR** terms to the ARIMA model as there are significant autocorrelations will be good, right?
- **Not exactly**
- If  $Y_t$  and  $Y_{t-1}$  are correlated, then  $Y_{t-1}$  and  $Y_{t-2}$  must also be correlated, then in turn  $Y_t$  and  $Y_{t-2}$  might also be correlated, because both share a connection to  $Y_{t-1}$ 
  - **Adding as many AR terms as there are significant ACF coefficients won't help**, since correlation in this case is only due to that connection through a shared datapoint and not through some new information that  $Y_{t-2}$  might contain to help predict  $Y_t$

# PACF mitigates the ACF shortcomings

- PACF (Partial Autocorrelation Function) plot helps with the issue
- Partial autocorrelations **measure the relationship between  $Y_t$  and  $Y_{t-k}$  AFTER removing the effects of lags  $1, 2, \dots, k-1$** 
  - The first PACF value will be the same as the first ACF value, because there is no effect to remove between  $Y_t$  and  $Y_{t-1}$
  - Every other  $k$ -th PACF coefficient is basically **equal to the estimate of the  $\phi_k$  coefficient** in an **AR( $k$ )** model. Yes, that's a lot of **AR** models to fit to look at one plot!

# ACF vs PACF: looking for patterns

- Some ACF/PACF patterns are distinct and indicative of either an  $AR(p)$  or  $MA(q)$  model
- Picking the right number of  $p$  or  $q$  components then would be easy based on either ACF or PACF plot
  - For  $AR(p)$ , we would use PACF
  - For  $MA(q)$ , we would use ACF

	<b>AR(p)</b>	<b>MA(q)</b>
<b>ACF</b>	Exponentially decaying or alternating between +/-	Close to 0 after lag q
<b>PACF</b>	Close to 0 after lag p	Exponentially decaying or alternating between +/-

# Plot ACF vs PACF

- Recall that we need to look at the *differenced* ACF/PACF in order to determine the number of *AR* and/or *MA* terms

```
# Take difference.
diff = list()
for i in range(1, len(deseasonalized)):
    value = deseasonalized[i] - deseasonalized[i - 1]
    diff.append(value)
```

- Don't forget that the list needs to be converted to a pandas series with `DatetimeIndex`
- Make sure to exclude the first observation, since differenced series for  $d = 1$  starts with the second one!

```
# Convert to series with DatetimeIndex.
deseasonalized_diff = pd.Series(diff, index = deseasonalized.index[1:])
print(deseasonalized_diff.head(3))
```

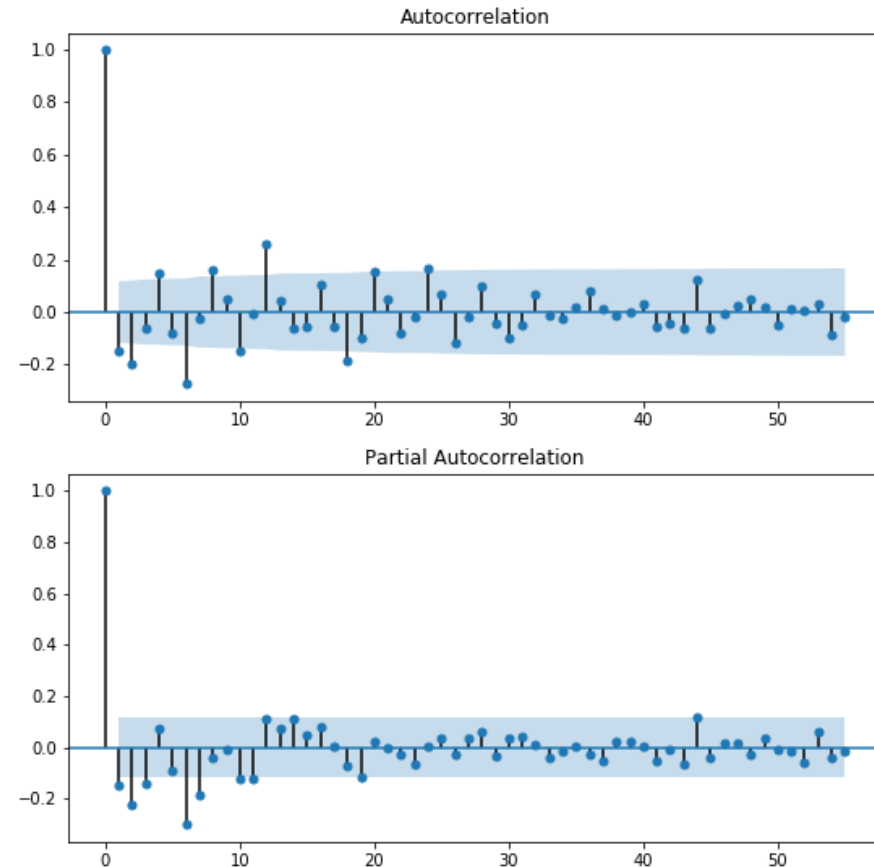
```
date
1979-02-01    -0.422330
1979-03-01    -0.767008
1979-04-01    -0.900114
dtype: float64
```

# Plot ACF vs PACF (cont'd)

```
max_k = deseasonalized_diff.shape[0]//5  
  
fig, axes = plt.subplots(nrows = 2, ncols = 1,  
                        figsize=(9, 9))  
plot_acf(deseasonalized_diff, lags = max_k,  
        ax = axes[0])  
plot_pacf(deseasonalized_diff, lags = max_k,  
        ax = axes[1])  
plt.show()
```

- There is no pattern in the ACF plot, although there are 3 positive autocorrelations above the confidence interval - lag = 12 the most prominent
- There is no pattern and no significant partial autocorrelation in PACF plot after the series has been differenced
- Although it is not strictly an **AR** model signature, we could try setting  $p = 12$ , fitting **ARIMA(12, 1, 0)**

<Figure size 1800x1800 with 2 Axes>



# Picking AR and MA components: rules of thumb

- If the  $PACF$  of the differenced series displays a sharp cutoff and/or the  $1_{lag} = 1$ , autocorrelation is positive
- If **the series appears somewhat “underdifferenced”**, then consider adding an ***AR*** term to the model
  - The lag at which the  $PACF$  drops close to 0 is the indicated number of ***AR*** terms
- If the  $ACF$  of the differenced series displays a sharp cutoff and/or the  $1_{lag} = 1$ , autocorrelation is negative
- **If the series appears somewhat “overdifferenced”**, then consider adding an ***MA*** term to the model
  - The lag at which the  $ACF$  drops close to 0 is the indicated number of ***MA*** terms

# General steps to construct an ARIMA model

1. Test the series for stationarity and stationarize it if needed ✓
  - i. Add the ***I*** (differencing) component to the ARIMA model ✓
2. Inspect autocorrelations (and partial autocorrelations) to determine if lags need to be included in the forecasting equation ✓
  - i. Add the ***AR*** and ***MA*** components to the ARIMA model ✓
3. Fit and assess the model
4. Patterns that remain may suggest the need for additional AR or MA terms

# Knowledge check 3





## Exercise 2



# Module completion checklist

Objective	Complete
Review the concept of stationarity	✓
Test decomposed series for stationarity	✓
Describe the components of an ARIMA model	✓
Discuss special cases of ARIMA models	✓
Describe an autoregressive (AR) model	✓
Describe a moving average (MA) model	✓
Describe a non-seasonal ARIMA model	✓
Use non-seasonal ARIMA model to forecast using deseasonalized series	

# Constructing ARIMA model: statsmodels ARIMA

- The most popular and robust function to set up an  $ARIMA(p, d, q)$  model is the `ARIMA` function from `statsmodels.tsa.arima_model` module

`statsmodels.tsa.arima_model.ARIMA`

```
class statsmodels.tsa.arima_model.ARIMA(endog, order, exog=None, dates=None, freq=None,  
missing='none')
```

[\[source\]](#)

Autoregressive Integrated Moving Average ARIMA(p,d,q) Model

- It takes 2 main arguments:
  - `endog`: the time series object (i.e. *endogenous* series)
  - `order`: a tuple that takes 3 values  $(p, d, q)$
- For more information about this function, take a look at the [official documentation](#)

# Constructing ARIMA model: set up model

- First, let's fit the model on the entire dataset and evaluate its performance

```
# Fit ARIMA model.  
arima = ARIMA(deseasonalized,  
              order = (12, 1, 0)) #<- add p, d, q components here
```

```
/Users/kiru/anaconda3/lib/python3.6/site-packages/statsmodels/tsa/base/tsa_model.py:171: ValueWarning:  
No frequency information was provided, so inferred frequency MS will be used.  
% freq, ValueWarning)  
/Users/kiru/anaconda3/lib/python3.6/site-packages/statsmodels/tsa/base/tsa_model.py:171: ValueWarning:  
No frequency information was provided, so inferred frequency MS will be used.  
% freq, ValueWarning)
```

```
print(arima)
```

```
<statsmodels.tsa.arima_model.ARIMA object at 0x1c288f8358>
```

- Notice the warning that tells us we have not provided the `freq` argument of the series
- Since we have provided a `pandas` series with `DatetimeIndex`, this argument is unnecessary and the model has identified the *monthly* frequency by default

# Constructing ARIMA model: fit model

- ARIMAResults object has a method `summary()` that consists of 3 tables:  
ARIMA Model Results is the first one

```
# Fit model and set `disp` to 0 to avoid  
# model debugging info print in console.  
arima_fit = arima.fit(disp = 0)
```

```
print(arima_fit.summary().tables[0])
```

ARIMA Model Results

Dep. Variable:	D.revenue_passenger_miles	No. Observations:	279
Model:	ARIMA(12, 1, 0)	Log Likelihood	-432.715
Method:	css-mle	S.D. of innovations	1.133
Date:	Thu, 19 Sep 2019	AIC	893.431
Time:	09:48:47	BIC	944.268
Sample:	02-01-1979	HQIC	913.824
	- 04-01-2002		

- The first table shows base model results with the model parameter as well as model performance metrics
- ARIMA models use *MLE*: maximum likelihood estimation to find the  $\phi$  and  $\theta$  parameters of the model (it maximizes the probability that the observed value comes from the model)
- AIC, BIC, and HQIC are called information criteria and are not used by themselves, but are only meaningful when comparing multiple models with different combinations of parameters
- The smaller these information criteria are, the better the model

# Constructing ARIMA model: results (cont'd)

```
print(arima_fit.summary().tables[1])
```

	coef	std err	z	P> z	[0.025	0.975]
const	0.0707	0.027	2.588	0.010	0.017	0.124
ar.L1.D.revenue_passenger_miles	-0.2748	0.058	-4.717	0.000	-0.389	-0.161
ar.L2.D.revenue_passenger_miles	-0.2051	0.061	-3.346	0.001	-0.325	-0.085
ar.L3.D.revenue_passenger_miles	-0.2193	0.062	-3.540	0.000	-0.341	-0.098
ar.L4.D.revenue_passenger_miles	-0.1443	0.063	-2.283	0.023	-0.268	-0.020
ar.L5.D.revenue_passenger_miles	-0.2110	0.064	-3.279	0.001	-0.337	-0.085
ar.L6.D.revenue_passenger_miles	-0.2882	0.064	-4.506	0.000	-0.414	-0.163
ar.L7.D.revenue_passenger_miles	-0.1889	0.064	-2.958	0.003	-0.314	-0.064
ar.L8.D.revenue_passenger_miles	-0.0131	0.084	-0.156	0.876	-0.177	0.151
ar.L9.D.revenue_passenger_miles	0.0083	0.085	0.098	0.922	-0.157	0.174
ar.L10.D.revenue_passenger_miles	-0.1974	0.082	-2.412	0.017	-0.358	-0.037
ar.L11.D.revenue_passenger_miles	-0.0868	0.082	-1.054	0.293	-0.248	0.075
ar.L12.D.revenue_passenger_miles	0.3085	0.079	3.887	0.000	0.153	0.464

- The second table displays performance metrics: `coef`, `stderr`, `z`, `P>|z|`, and `[0.025 0.975]` interval bounds for
  - Constant  $C$
  - Model coefficients  $\phi_i$
- The p-value for each of the coefficients `j` should ideally be under 0.05
- In our model, it seems like coefficients  $\phi_8$ ,  $\phi_9$ , and  $\phi_{11}$  are not statistically significant, which indicates a need in model parameter tuning!

# Constructing ARIMA model: results (cont'd)

```
print(arima_fit.summary().tables[2])
```

	Roots			
	Real	Imaginary	Modulus	Frequency
AR.1	-1.1013	-0.0000j	1.1013	-0.5000
AR.2	-0.9631	-0.5108j	1.0902	-0.4224
AR.3	-0.9631	+0.5108j	1.0902	0.4224
AR.4	-0.5684	-0.9096j	1.0726	-0.3389
AR.5	-0.5684	+0.9096j	1.0726	0.3389
AR.6	0.0061	-1.0189j	1.0190	-0.2490
AR.7	0.0061	+1.0189j	1.0190	0.2490
AR.8	0.5970	-0.9541j	1.1255	-0.1610
AR.9	0.5970	+0.9541j	1.1255	0.1610
AR.10	0.9197	-0.5685j	1.0812	-0.0881
AR.11	0.9197	+0.5685j	1.0812	0.0881
AR.12	1.3998	-0.0000j	1.3998	-0.0000

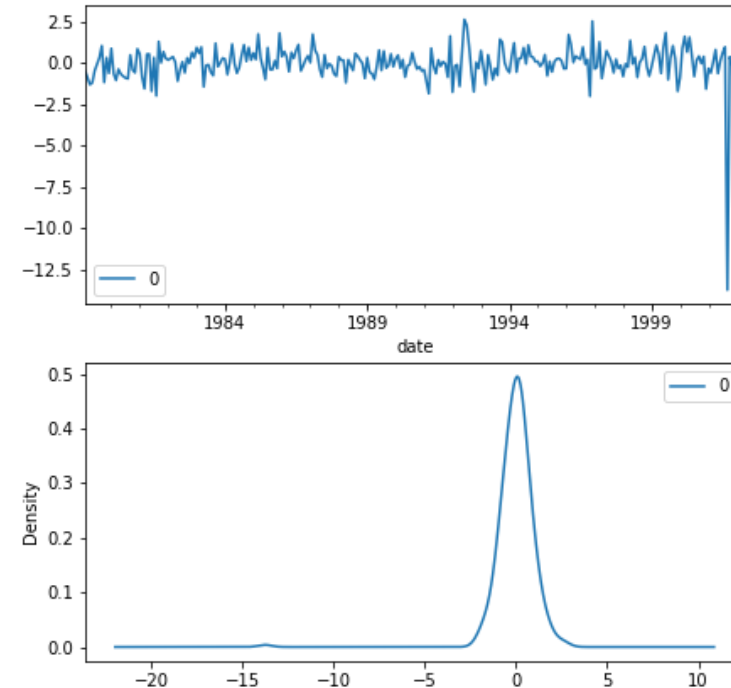
- The third table displays the Roots of the ARIMA model
- Ideally both the real and the imaginary parts of each root, which correspond to each of the model coefficients, are within the unit circle (i.e. are  $-1 < r < 1$ )
- Not all of the roots fall within the unit circle, which indicates that some of the model parameters could be adjusted

# Constructing ARIMA model: check residuals

```
# Plot residuals.  
residuals = pd.DataFrame(arima_fit.resid)  
print(residuals.describe())
```

```
count    279.000000  
mean     -0.010178  
std       1.138758  
min      -13.763552  
25%      -0.464566  
50%       0.047171  
75%       0.482207  
max       2.652529
```

```
fig, axes = plt.subplots(nrows = 2,  
                          ncols = 1,  
                          figsize = (8,8))  
residuals.plot(ax = axes[0])  
residuals.plot(ax = axes[1], kind='kde')  
plt.show()
```



- Recall: the assumptions require residuals to be normally distributed with  $\text{mean} = 0$  and  $\text{std} = 1$
- The plots and the summary statistics show that this model **meets assumptions**



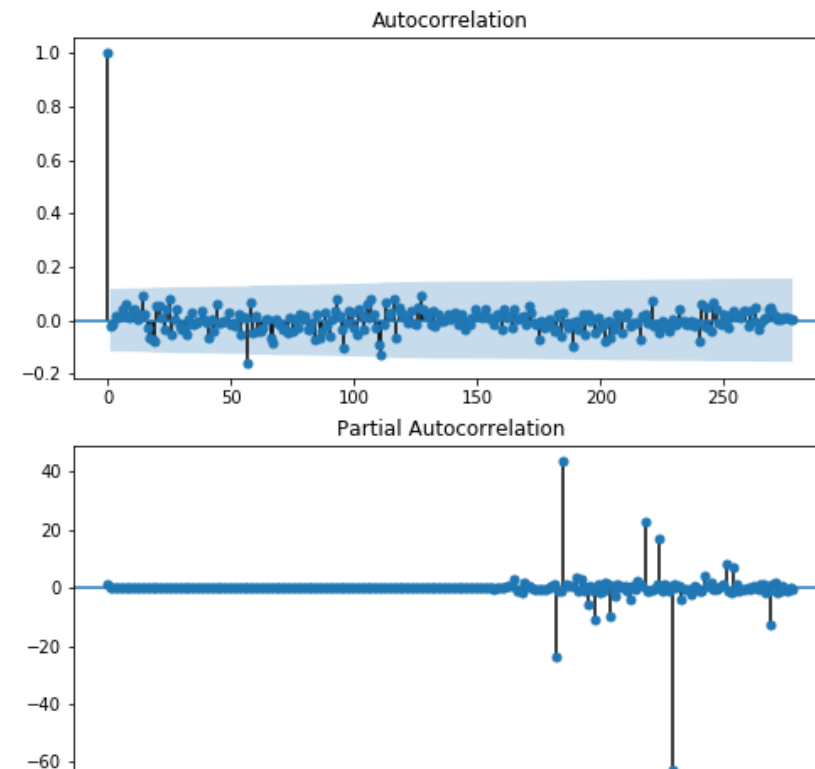
# Constructing ARIMA model: residuals (cont'd)

```
fig, axes = plt.subplots(nrows = 2,  
                          ncols = 1,  
                          figsize = (8,8))  
plot_acf(residuals, ax = axes[0])  
plot_pacf(residuals, ax = axes[1])  
plt.show()
```

<Figure size 1600x1600 with 2 Axes>

```
/Users/kiru/anaconda3/lib/python3.6/site-  
packages/statsmodels/regression/linear_model.py:1283:  
RuntimeWarning: invalid value encountered in sqrt  
return rho, np.sqrt(sigmasq)
```

- The assumptions also state that the residuals are not correlated
- If we look at both ACF and PACF plots of the residuals, we see that all of the autocorrelation coefficients are indeed insignificant
- The plots verify that the model **meets assumptions**



# Forecasting using ARIMA model: prep data

- Split data into train and test

```
# Set a cutoff point for train-test.  
cutoff = int(len(deseasonalized) * 0.90)  
# Split data into train and test.  
train, test = deseasonalized[0:cutoff], deseasonalized[cutoff:len(deseasonalized)]
```

- We will use 10% of datapoints for prediction, to demonstrate how forecasting works
- In practice, forecasts are not made too far ahead either (remember the article we discussed after the first module!)
- One-step-ahead forecasts or forecasts for short periods are common
- For longer periods, rolling forecasts are also an option, but for every new step ahead, they require an entire model recalculation

# Forecasting using ARIMA model

- The results of the `fit` object (i.e. `ARIMAResult`) has a method `forecast()` that allows us to forecast values of the time series 1 or more time steps ahead

```
statsmodels.tsa.arima_model.ARMAResults.forecast
```

method

```
ARMAResults.forecast(steps=1, exog=None, alpha=0.05)
```

[\[source\]](#)

Out-of-sample forecasts

- It requires no arguments, but if you would like to predict more values than 1, use the `steps` argument
- For more information about this function view [official documentation](#)

# Forecasting using ARIMA model (cont'd)

- Set up and fit ARIMA model to train data only

```
# Set up ARIMA model using train data only.  
arima = ARIMA(train, order = (12, 1, 0))  
# Fit ARIMA model to train data only.  
arima_fit = arima.fit(dispatch = 0)
```

- Create forecast for as many steps ahead as we have datapoints in our test data

```
# Generate ARIMA model forecast.  
output = arima_fit.forecast(steps = len(test))
```

- The output of the forecast function contains:
  - predicted values as the first element
  - standard errors as the second element
  - confidence intervals for each forecast value
- We will extract the values themselves and look at the confidence intervals within a plot

```
# Get predictions from forecast output.  
predictions = output[0]
```

# Forecasting using ARIMA model (cont'd)

- Let's print all of forecast vs actual values for each date in the test sample
- We will use `strftime` to strip the `DatetimeIndex` to just display `YYYY-MM` and omit everything else (i.e. date, hour, min, sec)

```
# Compare the raw `DatetimeIndex`.  
print(test.index[0])
```

```
2000-01-01 00:00:00
```

```
# With formatted `DatetimeIndex`.  
print(test.index[0].strftime("%Y-%m"))
```

```
2000-01
```

```
# For every datapoint in our predictions list.  
for i in range(len(predictions)):  
    # Print the date, forecast and actual value.  
    print('Date:%s forecast:%f, actual:%f'%(test.index[i].strftime("%Y-%m"), predictions[i], test[i]))
```

# Forecasting using ARIMA model (cont'd)

```
Date:2000-01 forecast:39.028022, actual:38.203325
Date:2000-02 forecast:40.058492, actual:40.240996
Date:2000-03 forecast:40.984737, actual:42.833988
Date:2000-04 forecast:40.584542, actual:42.713875
Date:2000-05 forecast:40.407356, actual:43.168988
Date:2000-06 forecast:41.740197, actual:44.419651
Date:2000-07 forecast:42.366471, actual:43.977927
Date:2000-08 forecast:41.175657, actual:42.384448
Date:2000-09 forecast:41.114959, actual:41.873633
Date:2000-10 forecast:41.983288, actual:42.691006
Date:2000-11 forecast:42.163741, actual:43.677890
Date:2000-12 forecast:40.959593, actual:40.834272
Date:2001-01 forecast:40.796452, actual:40.343325
Date:2001-02 forecast:41.652529, actual:41.380996
Date:2001-03 forecast:42.150303, actual:42.943988
Date:2001-04 forecast:41.656096, actual:43.163875
Date:2001-05 forecast:41.705496, actual:42.458988
Date:2001-06 forecast:42.662877, actual:43.889651
Date:2001-07 forecast:42.967646, actual:44.737927
Date:2001-08 forecast:42.236457, actual:44.394448
Date:2001-09 forecast:42.230956, actual:29.383633
Date:2001-10 forecast:42.864875, actual:34.251006
Date:2001-11 forecast:42.984945, actual:36.577890
Date:2001-12 forecast:42.298219, actual:35.994272
Date:2002-01 forecast:42.284912, actual:36.063325
Date:2002-02 forecast:42.947346, actual:37.780996
Date:2002-03 forecast:43.207106, actual:39.943988
Date:2002-04 forecast:42.828014, actual:38.953875
```

```
# Show mean squared error.
error = mean_squared_error(test,
                             predictions)
print('Test MSE: %.3f' % error)

# Show RMSE (i.e. metric in the same units
as our data).
```

```
Test MSE: 16.252
```

```
print('RMSE: %.3f' % sqrt(error))
```

```
RMSE: 4.031
```

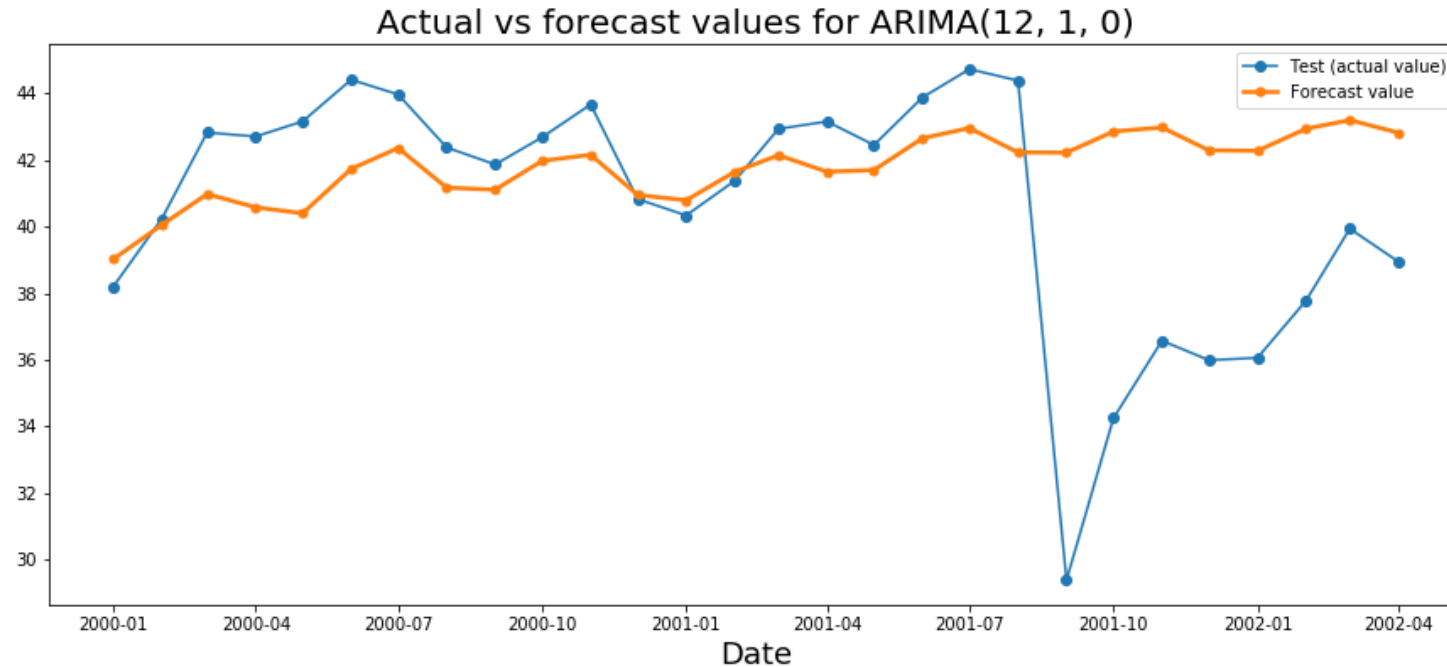
- What can you tell by looking at the results?
- Do any values jump out at you?
- The RMSE is in the same unit (billions of revenue miles) as our variable; what can you tell about it?

# View results: actual vs forecast

```
# Convert predictions list into series with DatetimeIndex.  
revenue_miles_forecast = pd.Series(predictions, index = test.index)
```

```
# Plot actual vs forecast values.  
fig, ax = plt.subplots(figsize = (15, 6))  
ax.plot(test,  
        marker = 'o',  
        linestyle = '-',  
        linewidth = 1.5,  
        label = 'Actual value')  
ax.plot(revenue_miles_forecast,  
        marker = 'o',  
        markersize = 5,  
        linewidth = 2.5,  
        linestyle = '-',  
        label = 'Forecast value')  
ax.set_xlabel('Date', fontsize = 18)  
ax.set_title("Actual vs forecast values for ARIMA(12, 1, 0)", fontsize = 20)  
ax.legend()  
plt.show()
```

# View results: actual vs forecast (cont'd)



- While the errors in predictions are fairly small for most values, the model did fail to predict a sudden dip in revenue miles in September, 2001
- The model captured overall trend and shape of some peaks and valleys, except for values from September 2001 and onward



# View forecast with confidence intervals

- Plotting the entire data + the confidence intervals of the forecast can tell just how bad the dip in revenue miles was and the values that were after it
- `ARIMAResults` object has a neat function `plot_predict` to visualize the forecast values along with the confidence intervals

```
statsmodels.tsa.arima_model.ARIMAResults.plot  
_predict
```

method

```
ARIMAResults.plot_predict(start=None, end=None, exog=None, dynamic=False, alpha=0.05,  
plot_insample=True, ax=None)
```

[\[source\]](#)

Plot forecasts

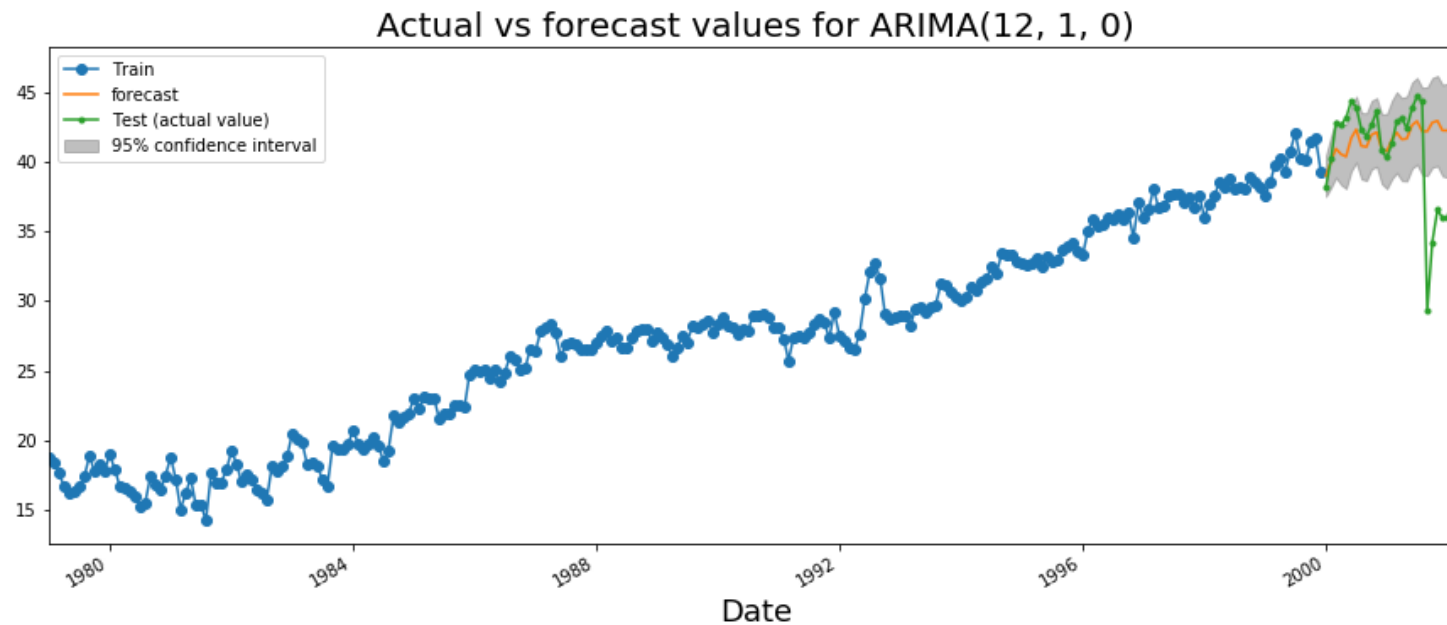
- The function requires the `start` and the `end` of the forecast period
- The default ***CI = 95*** and we will keep it (i.e. `alpha = 0.05`)
- The default for predictions is in sample: `plot_insample = True`, we will set it to `False`, because we trained our ARIMA model without the test values (out of sample)
- For more information, inspect the [official documentation](#)

# View results: the big picture

```
plt.close(fig="all") #<- closes all previous plots and figures
# Plot actual vs forecast values.
fig, ax = plt.subplots(figsize = (15, 6))
ax.plot(train,
        marker = 'o',
        linestyle = '-',
        linewidth = 1.5,
        label = 'Train')
arima_fit.plot_predict('2000-01', '2002-04', #<- date range for forecast values
                      ax = ax,
                      plot_insample = False) #<- forecast is done for out of sample values
ax.plot(test,
        marker = '.',
        linestyle = '-',
        label = 'Test (actual value)')
ax.set_xlabel('Date', fontsize = 18)
ax.set_title("Actual vs forecast values for ARIMA(12, 1, 0)", fontsize = 20)
ax.legend()
plt.show()
```

# View results: the big picture (cont'd)

- We clearly see that the forecast values did capture the overall trend and some variation
- The big dip and most of succeeding values were so different that they fall far outside of the 95% CI!
- This shows how bad forecasts (especially the long term kind) may turn out due to a sudden change caused by an external event of catastrophic proportions



# View results: forecast values of original series

- Recall that we used our non-seasonal ARIMA model on `deseasonalized` series
- We can only interpret the results correctly if we look at the actual vs forecast values in their original scale
- How do we do that?
- *Hint: we used an additive decomposition model to break the series down into its components*

# General steps to construct an ARIMA model

1. Test the series for stationarity and stationarize it if needed ✓
  - i. Add the ***I*** (differencing) component to the ARIMA model ✓
2. Inspect autocorrelations (and partial autocorrelations) to determine if lags need to be included in the forecasting equation ✓
  - i. Add the ***AR*** and ***MA*** components to the ARIMA model ✓
3. Fit and assess the model ✓
4. Patterns that remain may suggest the need for additional AR or MA terms

# The iCing on the ARIMA cake

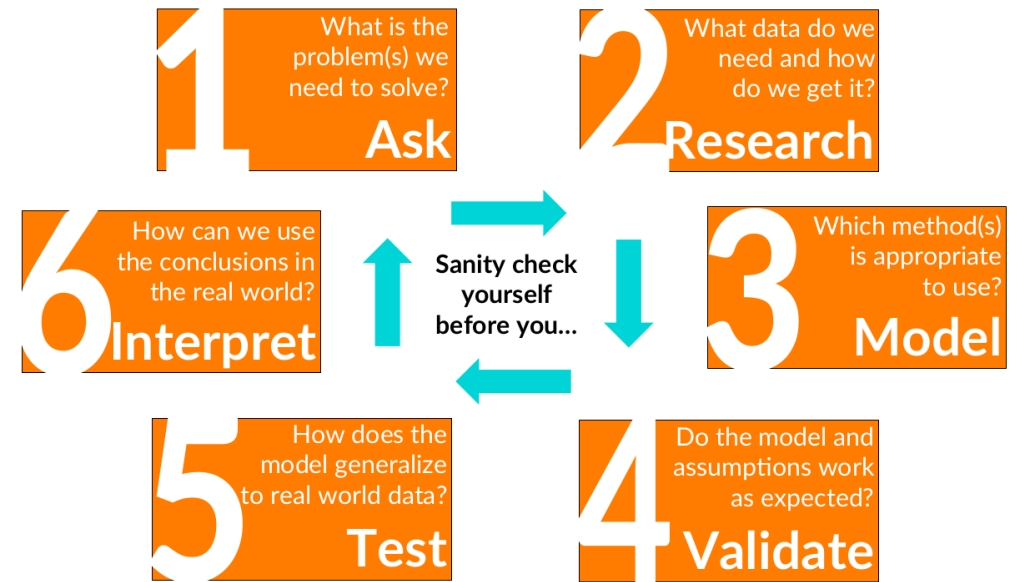
- The constant  $C$  has an important effect on the long-term forecasts obtained from the ARIMA model
  - If  $C = 0$  and  $d = 0$ , the long-term forecasts will go to zero
  - If  $C = 0$  and  $d = 1$ , the long-term forecasts will go to a nonzero constant
  - If  $C = 0$  and  $d = 2$ , the long-term forecasts will follow a straight line
  - If  $C \neq 0$  and  $d = 0$ , the long-term forecasts will go to the mean of the data
  - If  $C \neq 0$  and  $d = 1$ , the long-term forecasts will follow a straight line
  - If  $C \neq 0$  and  $d = 2$ , the long-term forecasts will follow a quadratic trend

# Tuning ARIMA model: rules of thumb

- It is possible for an *AR term and an MA term to cancel each other's effects*, so if a mixed AR-MA model seems to fit the data, also **try a model with one fewer AR term and one fewer MA term**, especially if the parameter estimates in the original model require more than 10 iterations to converge
- If the sum of the AR coefficients is almost exactly 1, reduce the number of AR terms by one and **increase the order of differencing by one**
- If the sum of the MA coefficients is almost exactly 1, reduce the number of MA terms by one and **reduce the order of differencing by one**

# What's next?

- Once adjustments are made to the original model
  - Rinse
  - Repeat
- Set up a grid search for optimal ARIMA parameters
- `statsmodels` is currently working on adding a function that lets you auto-tune the ARIMA model, so *stay tuned!*





# General steps to construct an ARIMA model

1. Test the series for stationarity and stationarize it if needed ✓
  - i. Add the ***I*** (differencing) component to the ARIMA model ✓
2. Inspect autocorrelations (and partial autocorrelations) to determine if lags need to be included in the forecasting equation ✓
  - i. Add the ***AR*** and ***MA*** components to the ARIMA model ✓
3. Fit and assess the model ✓
4. Patterns that remain may suggest the need for additional AR or MA terms ✓

# Knowledge check 4



# Exercise 3



# Module completion checklist

Objective	Complete
Review the concept of stationarity	✓
Test decomposed series for stationarity	✓
Describe the components of an ARIMA model	✓
Discuss special cases of ARIMA models	✓
Describe an autoregressive (AR) model	✓
Describe a moving average (MA) model	✓
Describe a non-seasonal ARIMA model	✓
Use non-seasonal ARIMA model to forecast using deseasonalized series	✓

# Workshop: next steps!

- Workshops are to be completed in the afternoon either with a dataset for a capstone project or with another dataset of your choosing
- Make sure to annotate and comment your code
- This is an exploratory exercise to get you comfortable with the content we discussed today

## Tasks for today:

- Test your data for stationarity and use differencing to stationarize the data if needed
- Pick **AR** and/or **MA** terms based on ACF/PACF plots
- Set up ARIMA model and evaluate its performance
- Split the data into `train` and `test` and forecast on out-of-sample `test` data, measure performance
- Discuss the results; is there a need to tune the model?
- **Bonus:** use grid search parameter testing to find the optimal  $ARIMA(p, d, q)$

# Congratulations on completing this module!

