

DATA SOCIETY®

Time series day 3

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

Module completion checklist

Objective	Complete
Define the concept of seasonality and trend in a time series model	
Utilize <code>DatetimeIndex</code> properties to augment time series data	
Visualize time series data for different time periods using line and boxplots	
Explain the idea of moving averages and how it can be used to decompose time series	
Deseasonalize and detrend a time series model using moving averages	

Data: use case in civil aviation

- We are using the same civil aviation dataset as our previous time series work
- The **passenger miles** dataset contains time series of three variables **from January 1979 to April 2002**:
 - Revenue passenger mile is a measure of the volume of air passenger transportation; a revenue passenger-mile is equal to one paying passenger carried one mile
 - Available seat mile is the number of seats multiplied by the distance traveled per flight and can represent the overall capacity of the aircraft
 - Unused mile is the difference between the above two variables and is used to measure the airline capacity utilization



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.seasonal import seasonal_decompose
from statsmodels.graphics.tsaplots import plot_acf
```

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)
```

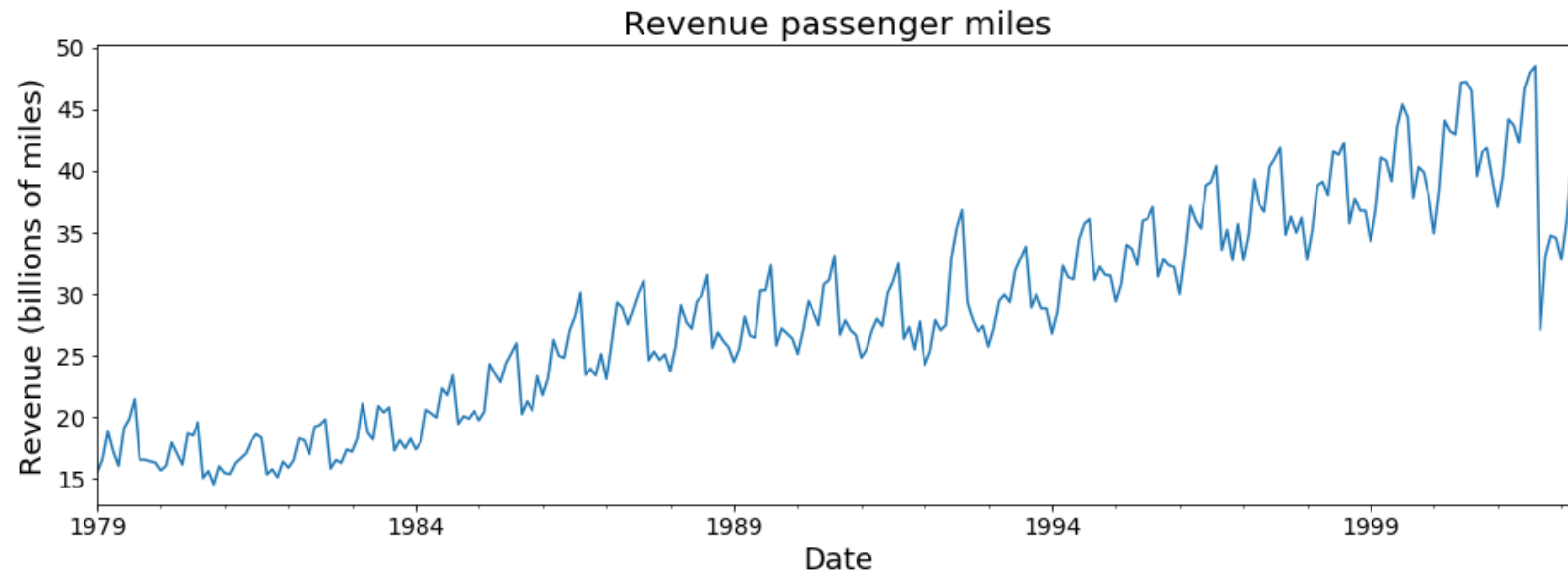
```
# Check working directory.  
print(os.getcwd())
```

```
/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



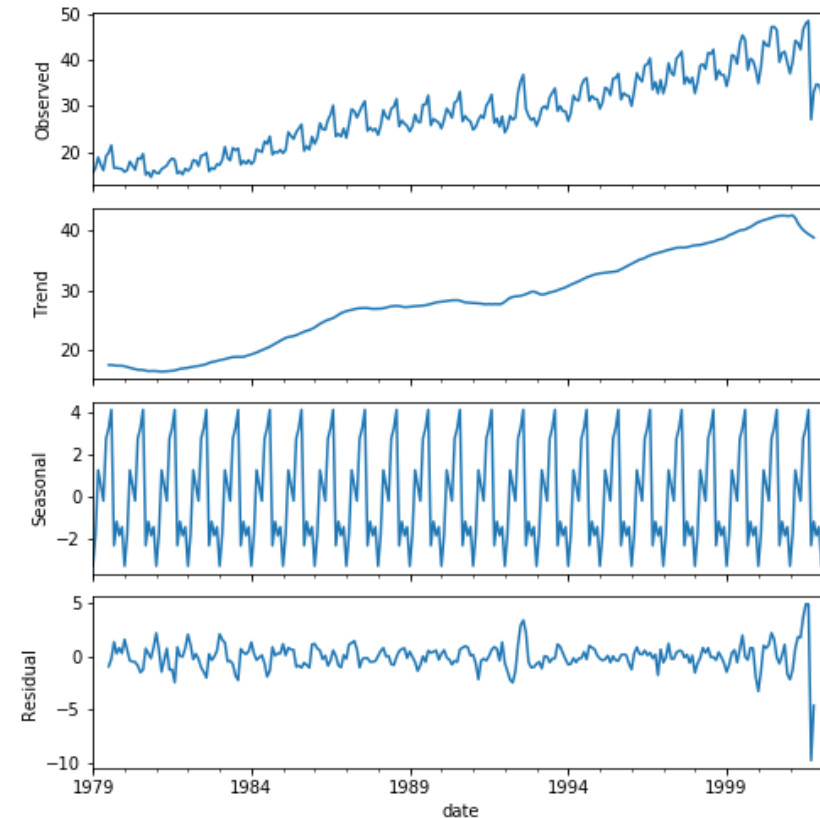
What if ...

- The time series is not random and does not come from **white noise**?
- The time series cannot be modeled by a random walk model?
- Are there **components** to each value of the series **we can extract and analyze**?
 - We can represent time series as a **combination of patterns at different scales** such as daily, weekly, seasonally, and yearly, **along with an overall trend**
 - Such representation of time series is called an **additive model** and we have already seen it in our first module: $Y_t = T_t + S_t + \epsilon_t$
- If the series *can be decomposed into such components*, it means *there are patterns* in our time series *that will help us make predictions* about the future values!

Recap: deterministic components of time series

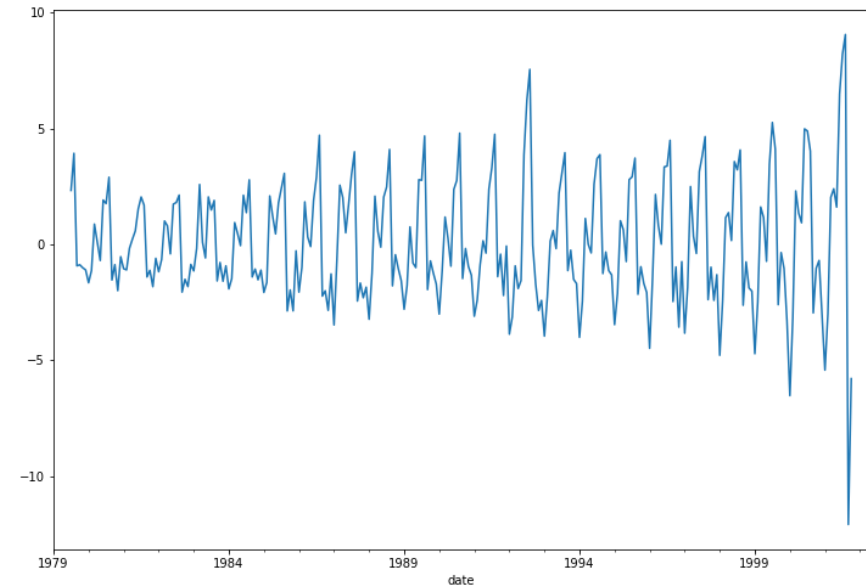
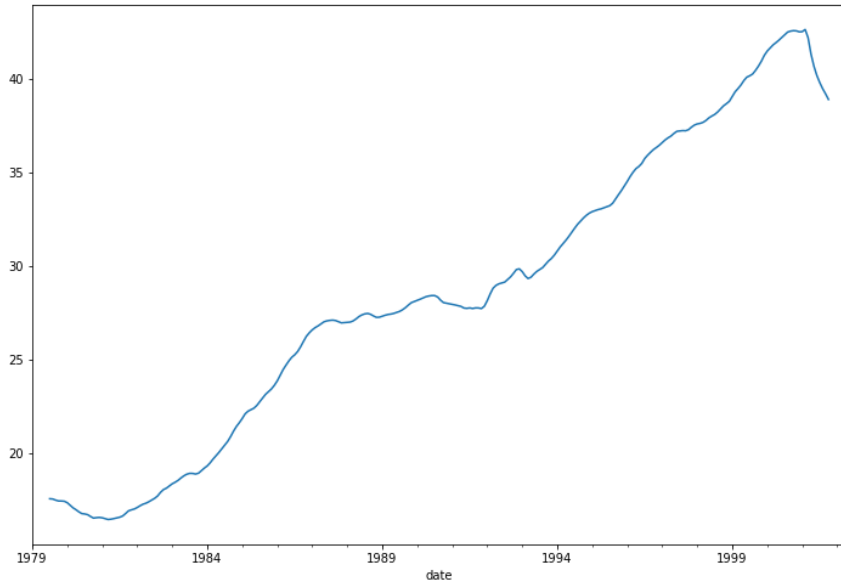
- The additive model has the following components:
 - The **trend** component (T_t)
 - The **seasonal** component (S_t)
 - The **random error** component (ϵ_t)
- When decomposed into the above components, the *additive* time series model now looks like this:

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



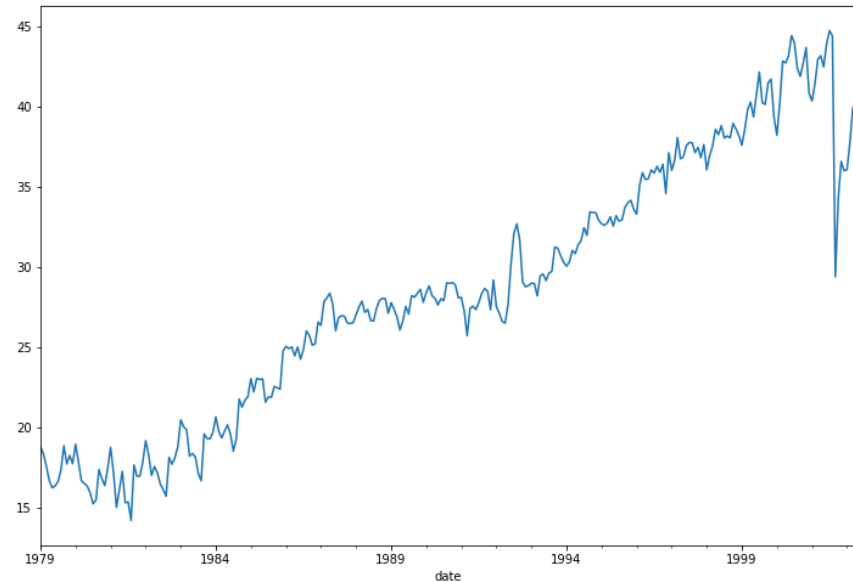
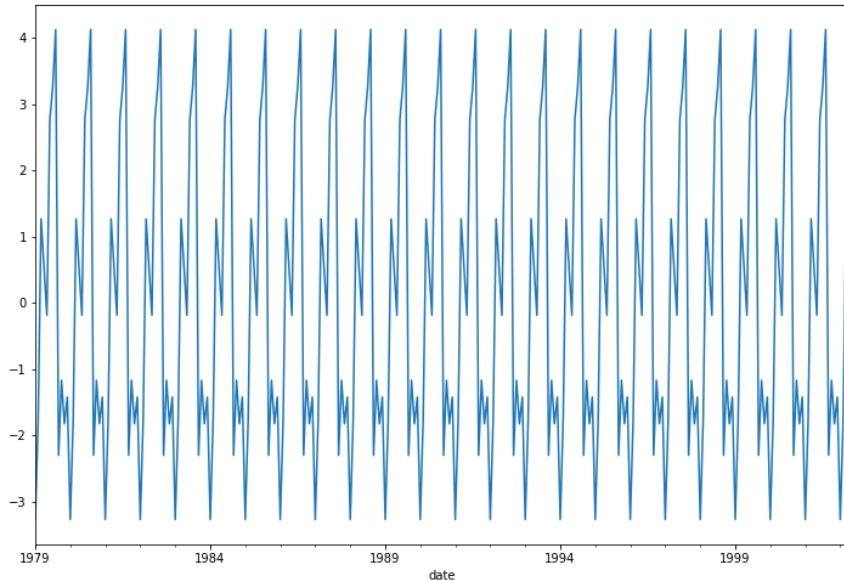
Recap: the trend component

- T_t : the **trend** component at time t
- $(Y_t - T_t)$: series without trend, or **detrended series**



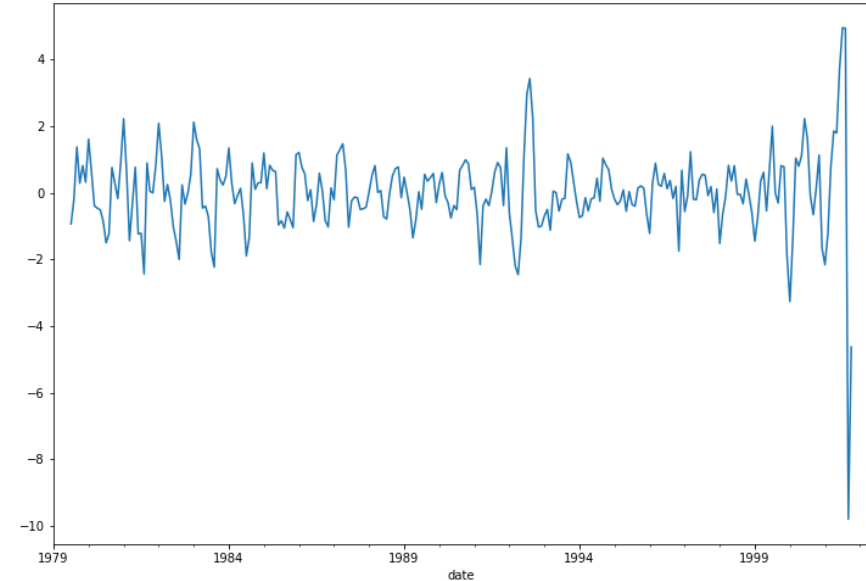
Recap: the seasonal component

- S_t : the **seasonal** component
- $(Y_t - S_t)$: series without the seasonal component, or **deseasonalized series**



Recap: detrended and deseasonalized series

- Time series with strong seasonality can often be well represented with models that decompose into seasonality and a long-term trend
- $(Y_t - S_t - T_t)$: series without the seasonal and trend components, or **deseasonalized** and **detrended** series
- Let's take out the trend and seasonal components from this equation
$$Y_t = T_t + S_t + \epsilon_t$$
- What's left?
 - **Noise**, also known as **residuals**, also known as **error**



Recap: what is all that noise?

- This quantity is our **forecast** for Y_t at time $t - 1$
- The error term ϵ_t , which is also known as the **noise** term is a simple difference between the **forecast** and the actual value of Y_t

$$\epsilon_t = Y_t - \mu_t = Y_t - (T_t + S_t) = Y_t - T_t - S_t$$

- When we remove the trend and the seasonality components of the time series, all we have left is the error $Y_t = \epsilon_t$
- Ideally, we have only **white noise** left, which means that we have captured the true nature of our data with all its variability with the trend and the seasonal components!


Additive model decomposition steps

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

Additive model decomposition steps: prework

- We can represent a time-series as a **combination of patterns at different scales** such as *daily, weekly, seasonally, and yearly*, to name a few, along with an **overall trend**
- In order for us to be able to estimate the T_t and the S_t we need to find out *what scale* to pick for our decomposition
- Let's make use of some useful pandas functions for working with `DatetimeIndex` and visualize our data for different periods first

Module completion checklist

Objective	Complete
Define the concept of seasonality and trend in a time series model	
Utilize <code>DatetimeIndex</code> properties to augment time series data	
Visualize time series data for different time periods using line and boxplots	
Explain the idea of moving averages and how it can be used to decompose time series	
Deseasonalize and detrend a time series model using moving averages	

Time based indexing

- One of the reasons to set the index to `DatetimeIndex` is so we can query our data using dates
- Notice that we are using `.loc` to *locate* the row at a given index value

```
# Time based indexing.  
print(passenger_miles.loc['2001-09-01'])
```

```
revenue_passenger_miles    27.08  
available_seat_miles       48.24  
unused_seat_miles          21.16  
Name: 2001-09-01 00:00:00, dtype: float64
```

Time based indexing for date ranges

- Not only can you get values for a single date, but also you can retrieve values for ranges of dates
- Just pass the range of dates you would like to subset

```
# Time based indexing.  
print(passenger_miles.loc['2001-09-01':'2002-04-01'])
```

date	revenue_passenger_miles	available_seat_miles	unused_seat_miles
2001-09-01	27.08	48.24	21.16
2001-10-01	33.08	51.07	17.99
2001-11-01	34.75	51.02	16.27
2001-12-01	34.57	51.54	16.96
2002-01-01	32.79	51.91	19.12
2002-02-01	36.01	53.40	17.39
2002-03-01	41.21	54.85	13.64
2002-04-01	39.50	55.54	16.04

Time based indexing for partial dates

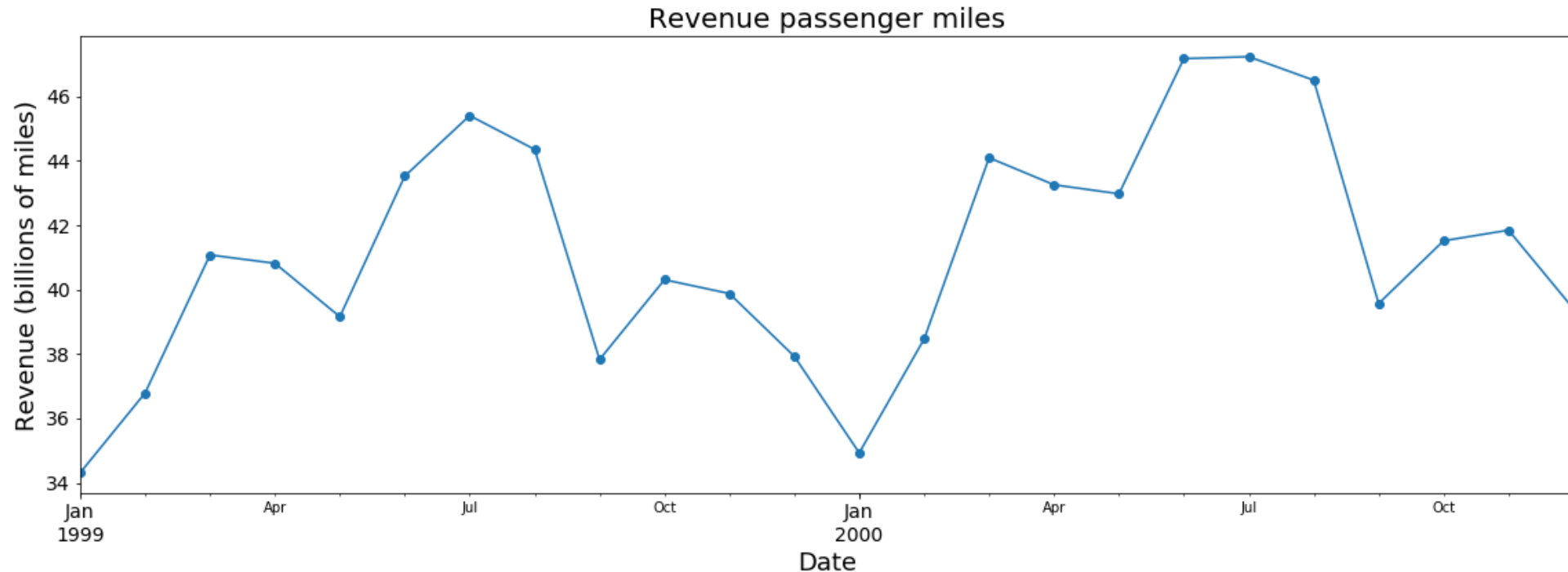
- Time based indexing works for partial dates
 - For example, if your `DatetimeIndex` contains days, but all you want to see is a range of a few months, you can just give the `year-mm`

```
# Partial indexing.  
print(passenger_miles.loc['2001-09':'2002-04'])
```

date	revenue_passenger_miles	available_seat_miles	unused_seat_miles
2001-09-01	27.08	48.24	21.16
2001-10-01	33.08	51.07	17.99
2001-11-01	34.75	51.02	16.27
2001-12-01	34.57	51.54	16.96
2002-01-01	32.79	51.91	19.12
2002-02-01	36.01	53.40	17.39
2002-03-01	41.21	54.85	13.64
2002-04-01	39.50	55.54	16.04

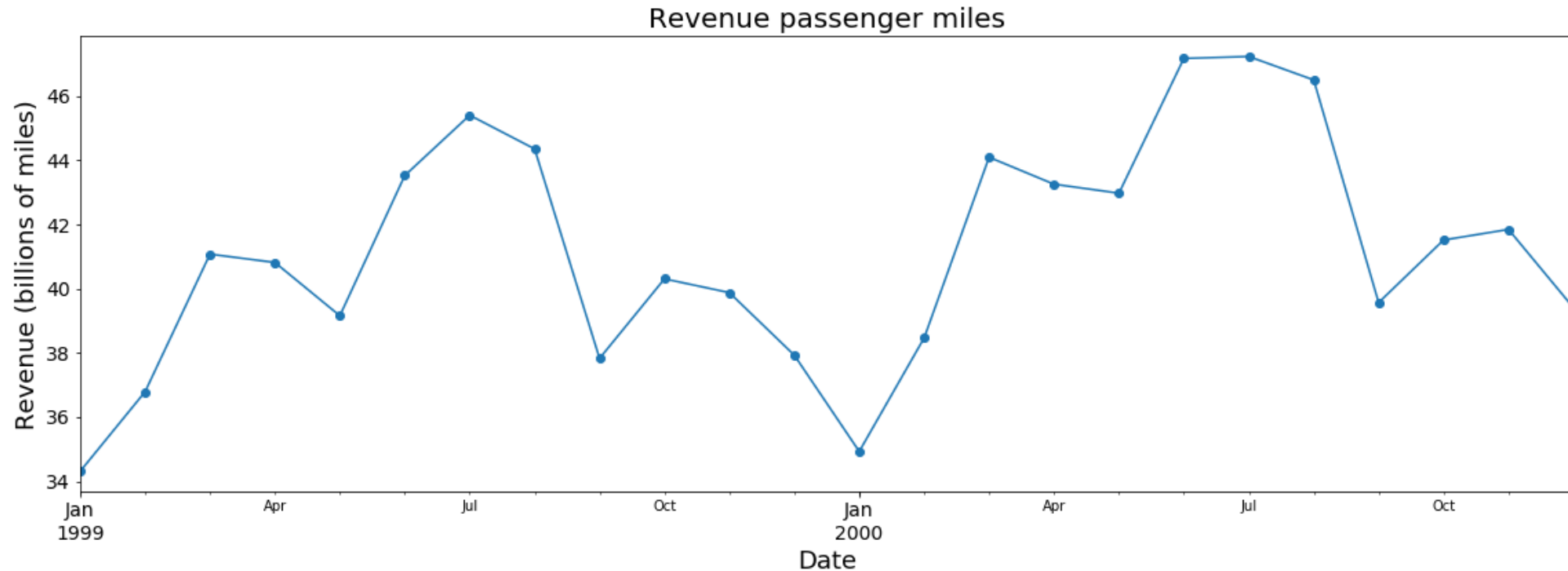
Visualize a subset of data

```
# Let's plot just the miles for 1999 and 2000.
fig, ax = plt.subplots(figsize = (16, 6))
passenger_miles.loc['1999':'2000', 'revenue_passenger_miles'].plot(marker = 'o') #<- set marker
plt.title('Revenue passenger miles', fontsize = 20)
plt.xlabel('Date', fontsize = 18)
plt.ylabel('Revenue (billions of miles)', fontsize = 18)
ax.tick_params(labelsize = 14)
plt.tight_layout() #<- allows labels to fit within plotting area
plt.show()
```



Any patterns so far?

- This plot allows us to see individual months clearly
- Aside from that, do you notice any patterns?



Augmenting time series data using DatetimeIndex

- A useful aspect of the `DatetimeIndex` is that the individual date/time components are all available as attributes such as `year`, `month`, `day`, and so on
- Let's add a few more columns to `passenger_miles` data, containing the `year`, `month`, and `quarter` information
- We will use the following attributes of the `DatetimeIndex`:
 - `dataframe.index.year`
 - `dataframe.index.month`
 - `dataframe.index.quarter`
- For a full list of attributes, inspect the official [*pandas.DatetimeIndex documentation*](#)

Augmenting time series data using DatetimeIndex

```
# Add columns with year, month, and weekday name.
passenger_miles['year'] = passenger_miles.index.year
passenger_miles['month'] = passenger_miles.index.month
passenger_miles['quarter'] = passenger_miles.index.quarter
print(passenger_miles.head())
```

date	revenue_passenger_miles	available_seat_miles	...	month	quarter
1979-01-01	15.50	26.64	...	1	1
1979-02-01	16.58	27.20	...	2	1
1979-03-01	18.85	27.87	...	3	1
1979-04-01	17.23	23.22	...	4	2
1979-05-01	16.04	23.27	...	5	2

[5 rows x 6 columns]

- The breakdown of dates into their components will help us **visualize** and **resample** our data so that we can perform seasonal decomposition!

Knowledge check 1



Exercise 1



Module completion checklist

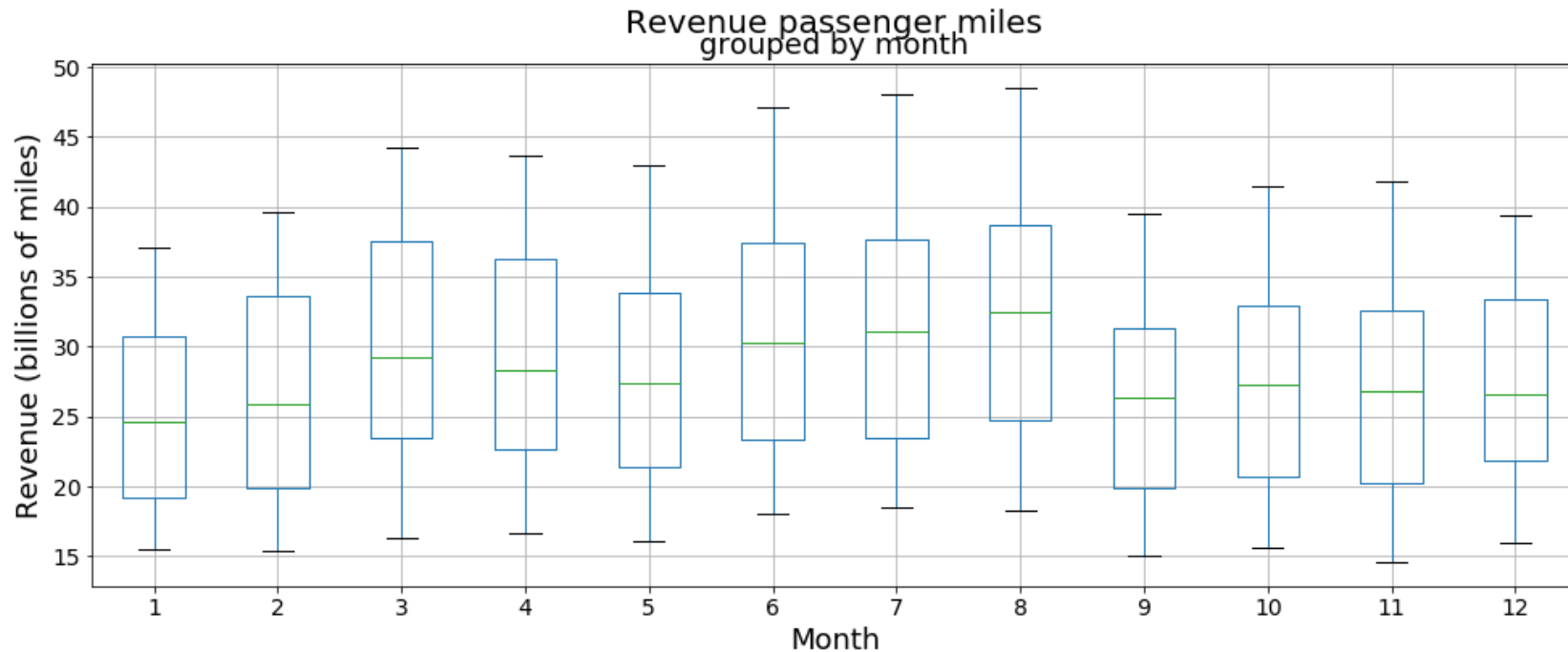
Objective	Complete
Define the concept of seasonality and trend in a time series model	✓
Utilize <code>DatetimeIndex</code> properties to augment time series data	✓
Visualize time series data for different time periods using line and boxplots	
Explain the idea of moving averages and how it can be used to decompose time series	
Deseasonalize and detrend a time series model using moving averages	

Visualize data for different periods

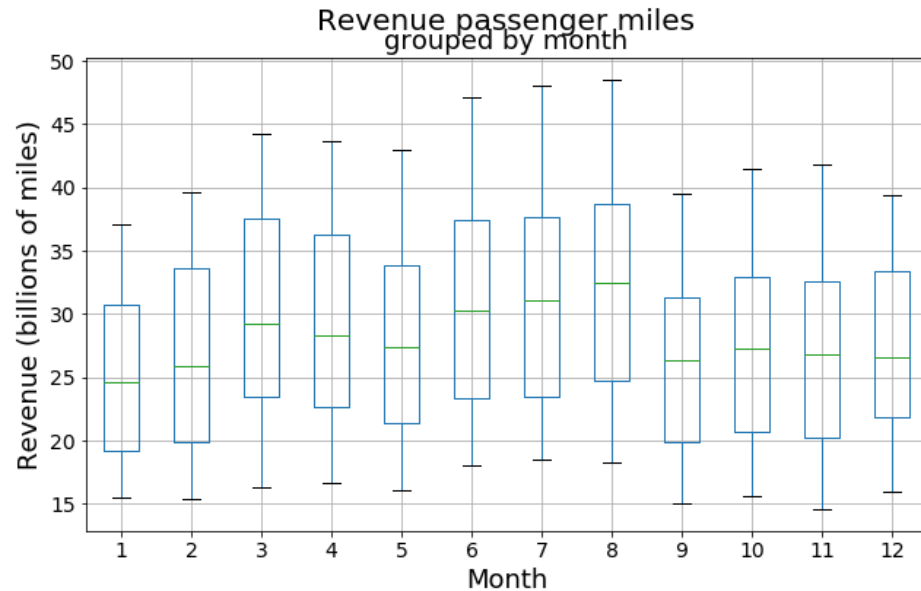
- One of the simplest and most effective ways to detect seasonal patterns is to visualize data grouped by different periods
- We are going to plot distribution of datapoints for such groups using box plots
- `dataframe.boxplot()` function from `pandas` makes it very easy to do that as it takes 2 main arguments:
 - `column`: a dataframe column we would like to visualize
 - `by`: a column from the dataframe by which we want our observations to be grouped
- For more arguments and options, please take a look at the official [*`pandas.DataFrame.boxplot` documentation*](#)

Visualize data for different periods: by month

```
passenger_miles.boxplot(column = 'revenue_passenger_miles',  
                        by = 'month',  
                        figsize = (16, 6), fontsize = 14) #<- you can adjust figure and tick fontsize  
plt.suptitle('Revenue passenger miles', fontsize = 20) #<- change default subtitle  
plt.title('grouped by month', fontsize = 18) #<- change default title  
plt.xlabel('Month', fontsize = 18)  
plt.ylabel('Revenue (billions of miles)', fontsize = 18)  
plt.show()
```



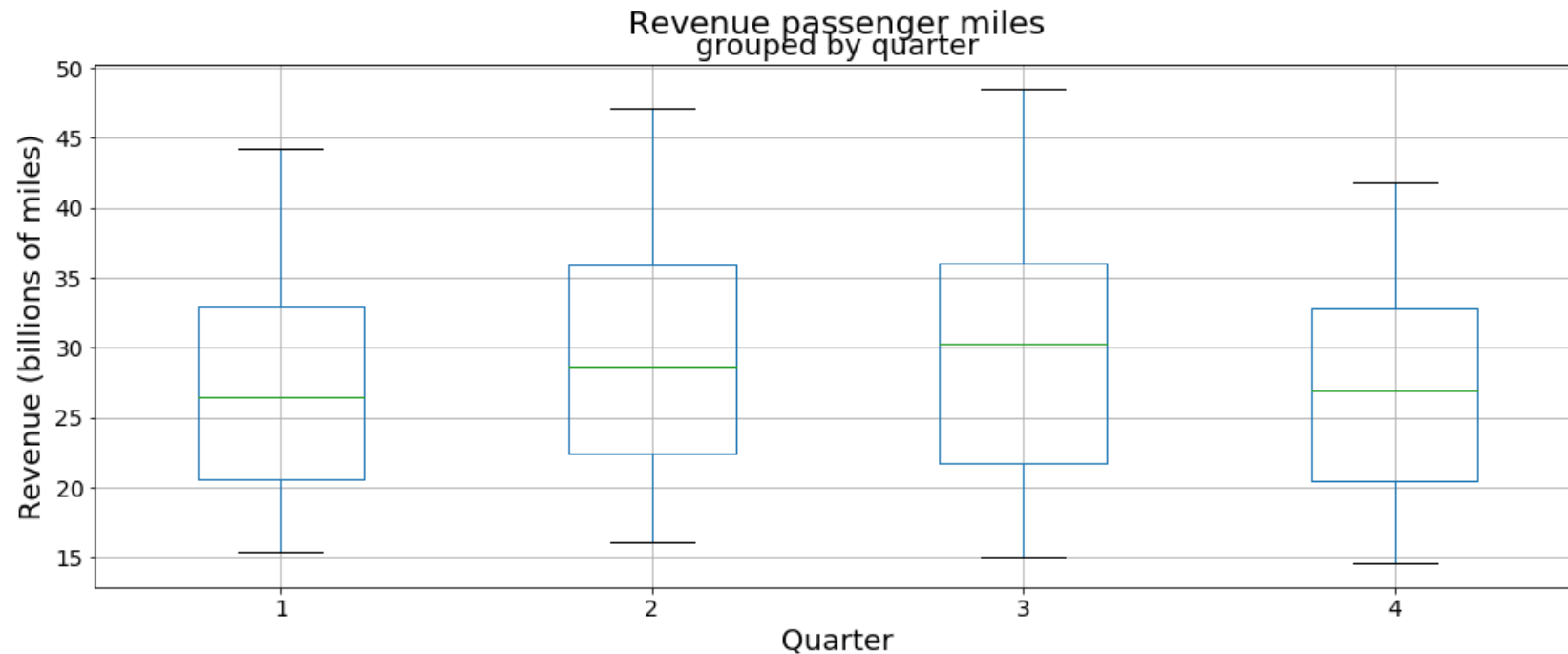
Visualize data for different periods: by month



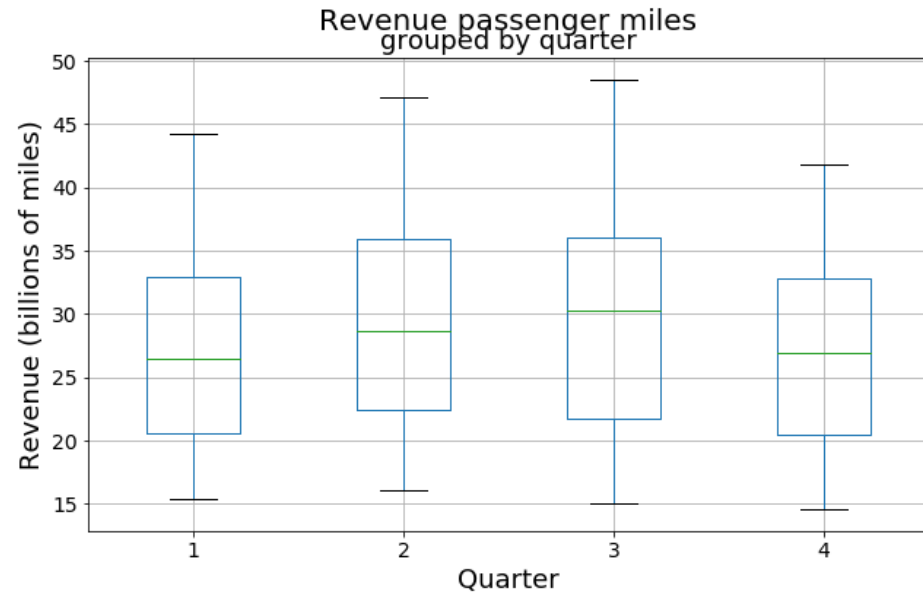
- When looking at monthly values distributions across our dataset, what can you say about this data?
 - Which months tend to have the highest median revenue miles?
 - Which ones are the lowest?
 - Why do you think that happens?

Visualize data for different periods: by quarter

```
passenger_miles.boxplot(column = 'revenue_passenger_miles',  
                        by = 'quarter',  
                        figsize = (16, 6), fontsize = 14)  
plt.suptitle('Revenue passenger miles', fontsize = 20)  
plt.title('grouped by quarter', fontsize = 18)  
plt.xlabel('Quarter', fontsize = 18)  
plt.ylabel('Revenue (billions of miles)', fontsize = 18)  
plt.show()
```



Visualize data for different periods: by quarter



- When looking at quarterly values distributions across our dataset, what can you say about this data?
 - Which quarters tend to have the highest median revenue miles?
 - Which ones are the lowest?
 - Why do you think that happens?

Quarters do not equal seasons

- Quarterly data shows that the highest median revenue miles occur in Q3, but the distribution of the values for Q3 is skewed left, with more values in the lower end of the distribution (lower revenue) and fewer in the high end of the distribution (higher revenue)
- At the same time, Q2 shows a lower median value, but is more normally distributed
- This is happening because our quarters are not aligned with the seasons:
 - Q1: Jan, Feb, Mar
 - Q2: **Apr**, **May**, **Jun**
 - Q3: **Jul**, **Aug**, **Sept**
 - Q4: Oct, Nov, Dec
- The **moderately high** revenue miles in Spring and **high values** in June got lumped together in Q2, while **high** Summer values and **lower** Fall values got mixed in Q3
- Since our data displays a more “seasonal” seasonality, let's add a `season` column to our data and see if the pattern re-aligns itself!

Adding seasons to data

```
# Apply math formula to convert months to corresponding seasons.
passenger_miles['season'] = (passenger_miles['month']%12 + 3)//3
```

```
# Make a dictionary mapping numeric values to season names.
season_dict = {1: 'winter',
               2: 'spring',
               3: 'summer',
               4: 'fall'}
```

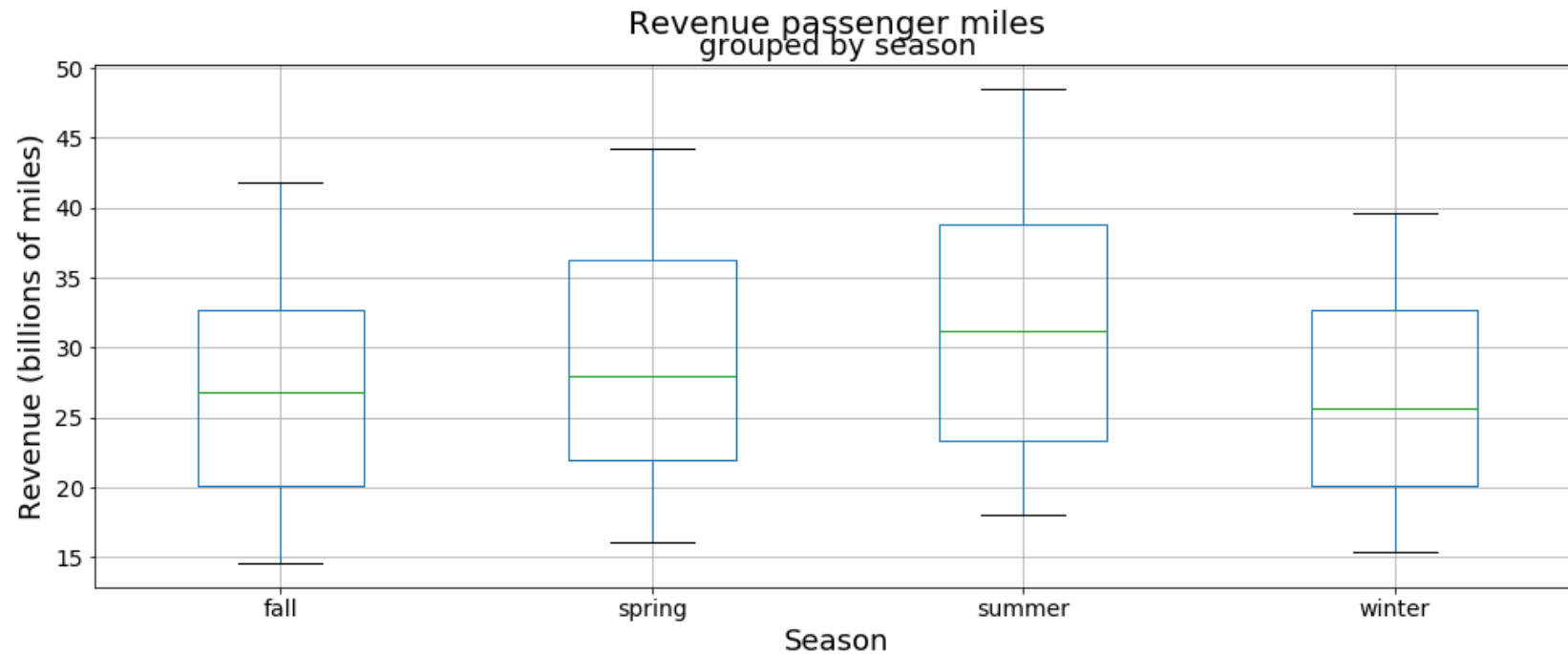
```
# Map numeric values to corresponding season names.
passenger_miles['season'] = passenger_miles['season'].map(season_dict)
print(passenger_miles.head())
```

date	revenue_passenger_miles	available_seat_miles	...	quarter	season
1979-01-01	15.50	26.64	...	1	winter
1979-02-01	16.58	27.20	...	1	winter
1979-03-01	18.85	27.87	...	1	spring
1979-04-01	17.23	23.22	...	2	spring
1979-05-01	16.04	23.27	...	2	spring

```
[5 rows x 7 columns]
```

Seasonal distributions

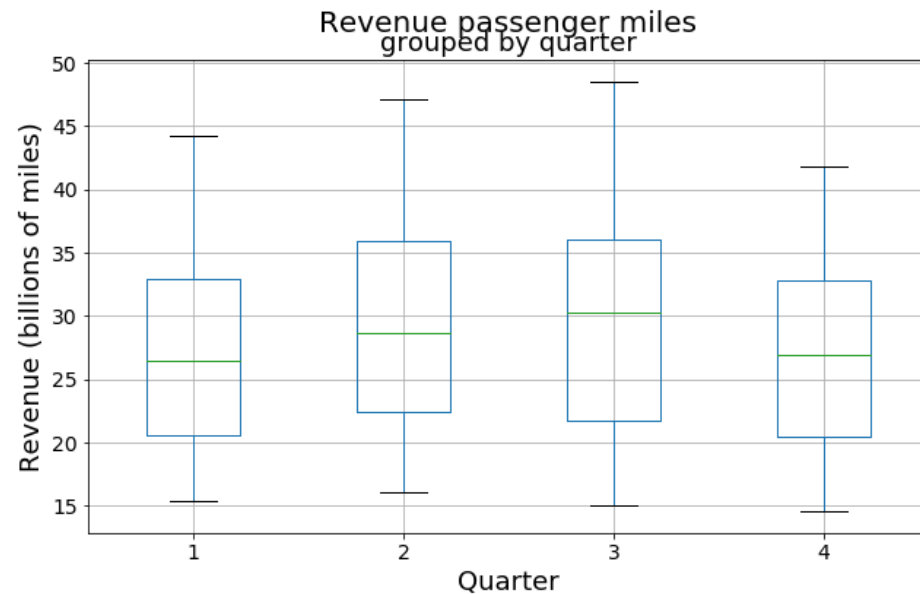
```
passenger_miles.boxplot(column = 'revenue_passenger_miles',  
                        by = 'season',  
                        figsize = (16, 6), fontsize = 14)  
plt.suptitle('Revenue passenger miles', fontsize = 20)  
plt.title('grouped by season', fontsize = 18)  
plt.xlabel('Season', fontsize = 18)  
plt.ylabel('Revenue (billions of miles)', fontsize = 18)  
plt.show()
```



Quarters vs seasons

Quarters

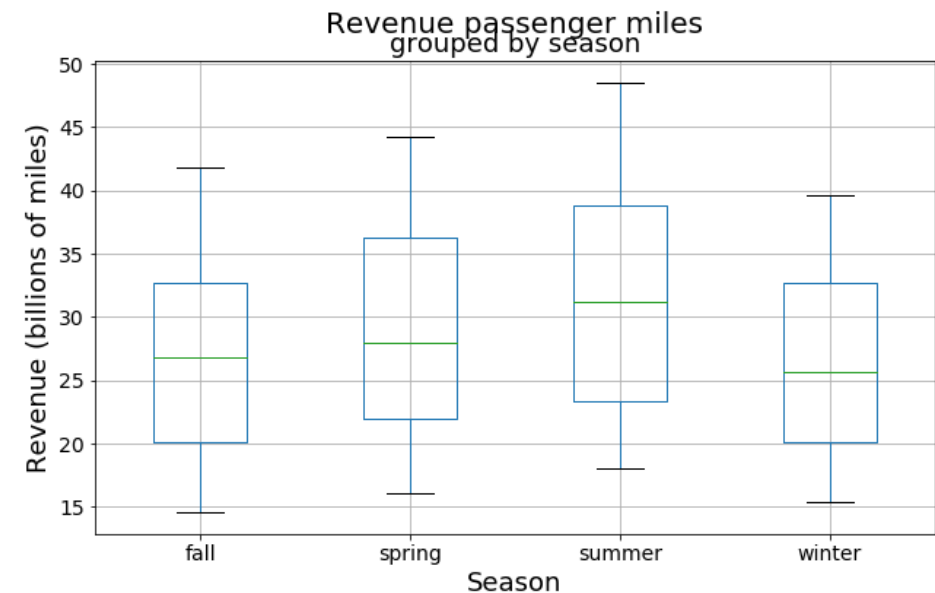
- Both Q2 and Q3 show high median revenue miles, but neither goes over 30 billion



- Seasonal pattern is not as pronounced

Seasons

- Q3 most definitely displays the highest median revenue miles, which is about 32 billion



- Seasonal pattern is definitely present

Resampling data

- It is often useful to **resample** our time series data to a lower or higher frequency to look at a bigger picture or an emerging pattern in time series
- Resampling to a **lower frequency (downsampling)** usually involves an aggregation operation — for example, computing monthly sales totals from daily data
 - The data we're working with in this module appears to have *monthly frequency*
 - We will **downsample** it **to quarterly and seasonal frequency** to see if any patterns in passenger mobility appear

Resampling data (cont'd)

- Resampling to a **higher frequency (upsampling)** is less common and often involves interpolation or other data filling method — for example, interpolating hourly weather data to 10 minute intervals for input to a scientific model
- We will use another `pandas` function designed for time series data `series.resample()`, where `series` is any time series of our choice
- For more information about this function, see the [official documentation](#)

Resample: compute quarterly and seasonal means

- To get **conventional quarterly** resampling, use `.resample(Q)`, which returns a conventional quarter with **quarter end in December**

```
# Resample to quarterly frequency, aggregating with mean.  
revenue_miles_quarterly_mean = passenger_miles['revenue_passenger_miles'].resample('Q').mean()
```

- To get **seasonal quarterly** resampling, use `.resample(QS-DEC)`, which returns an offset quarter with **quarter start in December**

```
# Resample to seasonal frequency, aggregating with mean.  
revenue_miles_seasonal_mean = passenger_miles['revenue_passenger_miles'].resample('QS-DEC').mean()
```

Compare: quarterly vs seasonal

- Values are more homogeneous for classic quarterly data

```
print(revenue_miles_quarterly_mean.head(10))
```

```
date
1979-03-31    16.976667
1979-06-30    17.460000
1979-09-30    19.300000
1979-12-31    16.426667
1980-03-31    16.563333
1980-06-30    17.290000
1980-09-30    17.720000
1980-12-31    15.400000
1981-03-31    15.710000
1981-06-30    17.270000
Freq: Q-DEC, Name: revenue_passenger_miles,
dtype: float64
```

- Values vary more for quarterly data offset to match the seasons

```
print(revenue_miles_seasonal_mean.head(10))
```

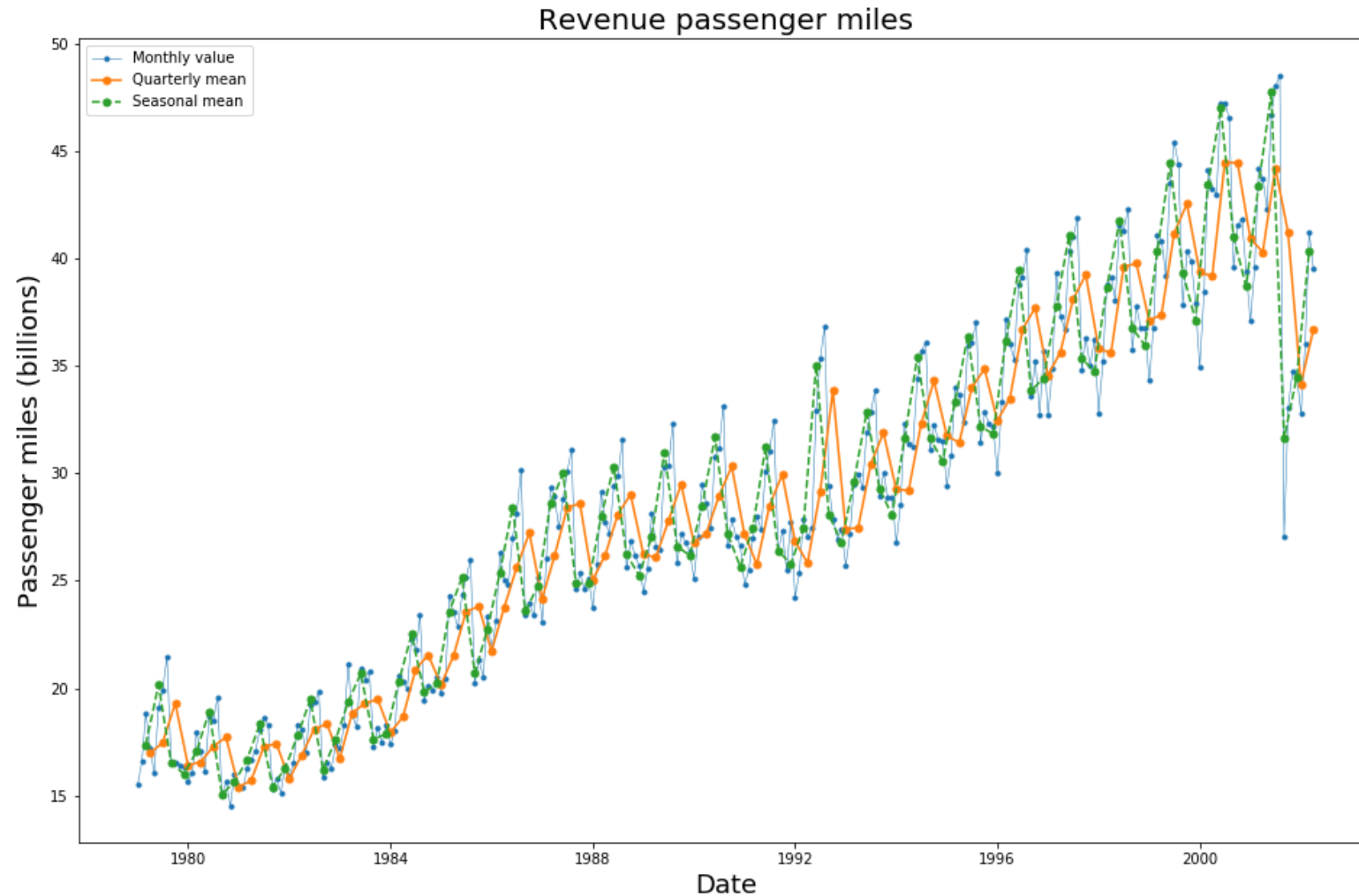
```
date
1978-12-01    16.040000
1979-03-01    17.373333
1979-06-01    20.153333
1979-09-01    16.506667
1979-12-01    16.020000
1980-03-01    17.046667
1980-06-01    18.923333
1980-09-01    15.080000
1980-12-01    15.626667
1981-03-01    16.673333
Freq: QS-DEC, Name: revenue_passenger_miles,
dtype: float64
```

Visualize data for different periods: line graph

```
# Name the start and end of the date range to extract.
start, end = '1979-01', '2002-04'

# Plot monthly and quarterly resampled time series together.
fig, ax = plt.subplots(figsize = (16, 10))
ax.plot(passenger_miles['revenue_passenger_miles'].loc[start:end, ],
        marker = 'T.',
        linestyle = '-',
        linewidth = 0.5,
        label = 'Monthly value')
ax.plot(revenue_miles_quarterly_mean.loc[start:end, ],
        marker = 'o',
        markersize = 5,
        linestyle = '-',
        label = 'Quarterly mean')
ax.plot(revenue_miles_seasonal_mean.loc[start:end, ],
        marker = 'o',
        markersize = 5,
        linestyle = '--',
        label = 'Seasonal mean')
ax.set_ylabel("Passenger miles (billions)", fontsize = 18)
ax.set_xlabel('Date', fontsize = 18)
ax.set_title('Revenue passenger miles', fontsize = 20)
ax.legend()
plt.show()
```


Visualize data for different periods: line graph

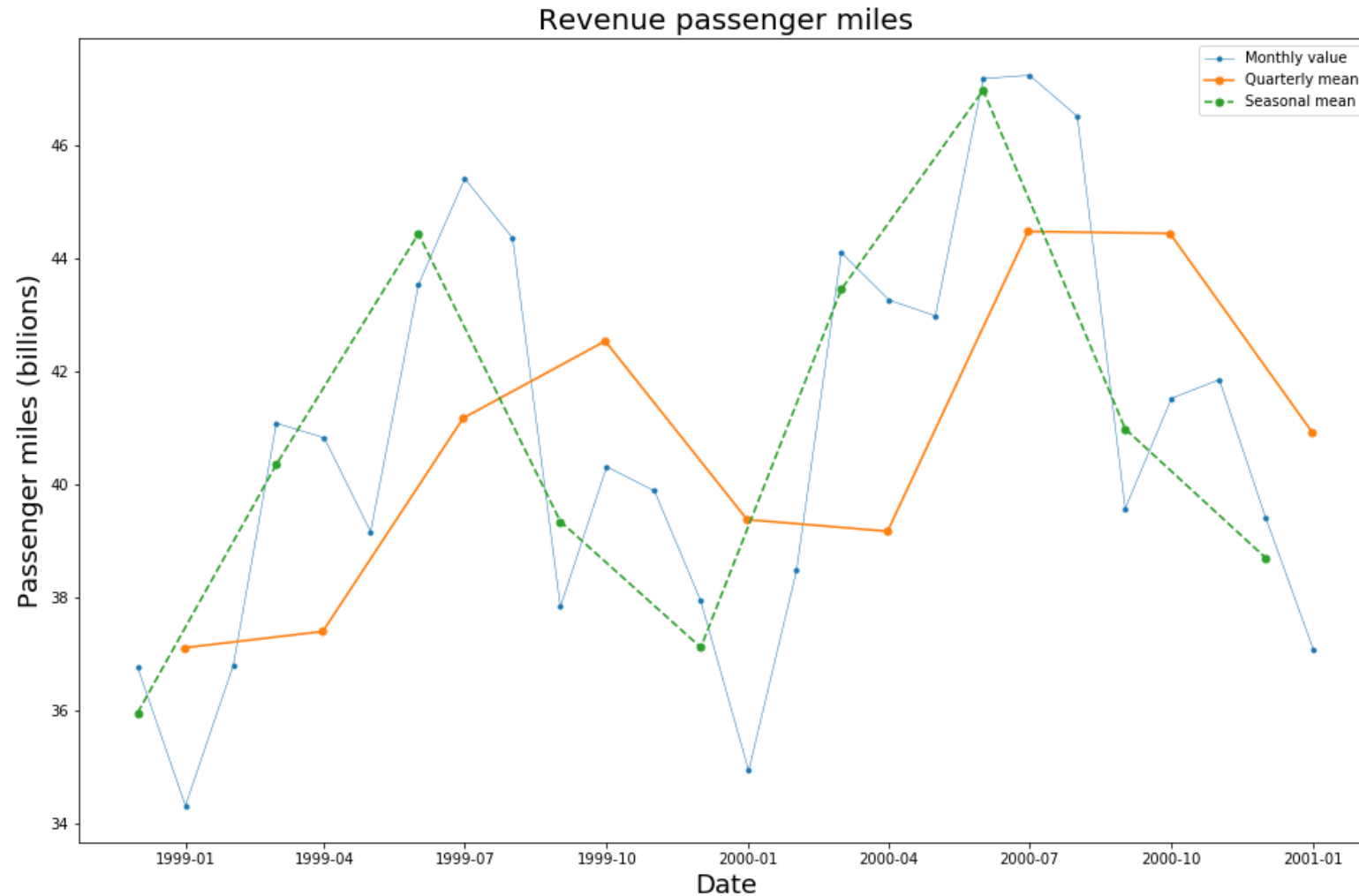


Visualize data for different periods: line graph

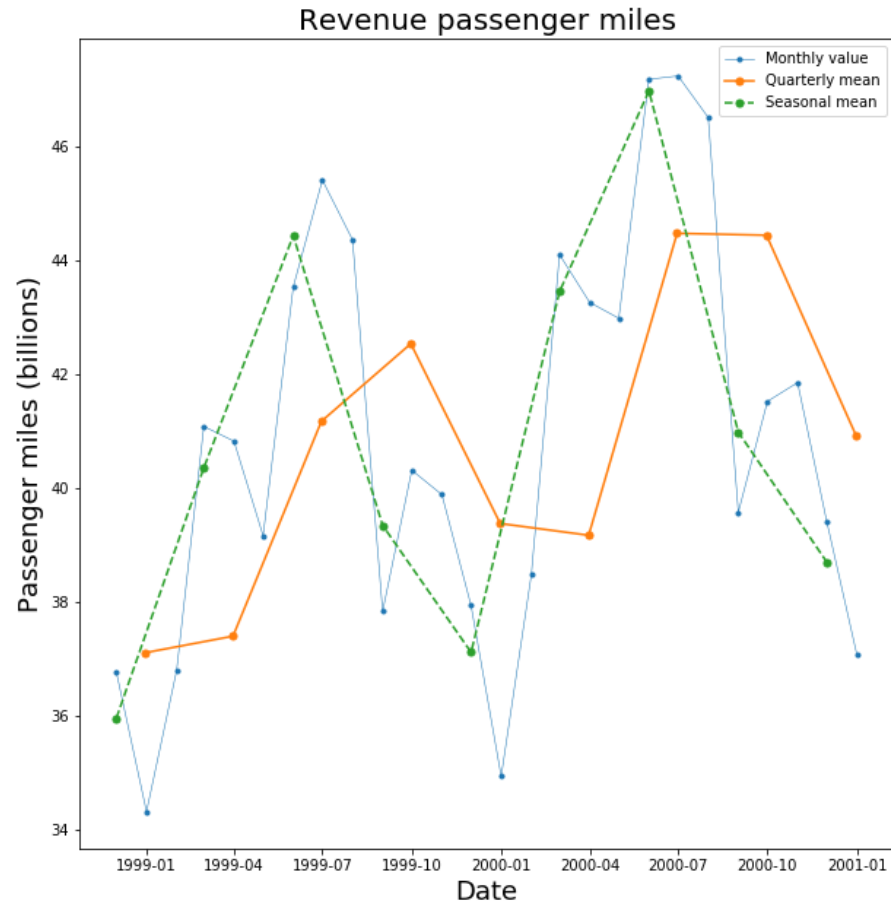
```
# Label the start and end of the date range to extract.
start, end = '1998-12', '2001-01'

# Plot monthly and quarterly resampled time series together.
fig, ax = plt.subplots(figsize = (16, 10))
ax.plot(passenger_miles['revenue_passenger_miles'].loc[start:end, ],
        marker = 'T.',
        linestyle = '-',
        linewidth = 0.5,
        label = 'Monthly value')
ax.plot(revenue_miles_quarterly_mean.loc[start:end, ],
        marker = 'o',
        markersize = 5,
        linestyle = '-',
        label = 'Quarterly mean')
ax.plot(revenue_miles_seasonal_mean.loc[start:end, ],
        marker = 'o',
        markersize = 5,
        linestyle = '--',
        label = 'Seasonal mean')
ax.set_ylabel("Passenger miles (billions)", fontsize = 18)
ax.set_xlabel('Date', fontsize = 18)
ax.set_title('Revenue passenger miles', fontsize = 20)
ax.legend()
plt.show()
```

Visualize data for different periods: line graph



What's the verdict: quarters or seasons?



- When looking at this graph, what could you say about this plot?
- Does the quarterly mean capture seasonal variations better than the seasonal mean?
- Which one should be used in our seasonal decomposition? Why?

Knowledge check 2



Exercise 2



Module completion checklist

Objective	Complete
Define the concept of seasonality and trend in a time series model	✓
Utilize <code>DatetimeIndex</code> properties to augment time series data	✓
Visualize time series data for different time periods using line and boxplots	✓
Explain the idea of moving averages and how it can be used to decompose time series	
Deseasonalize and detrend a time series model using moving averages	

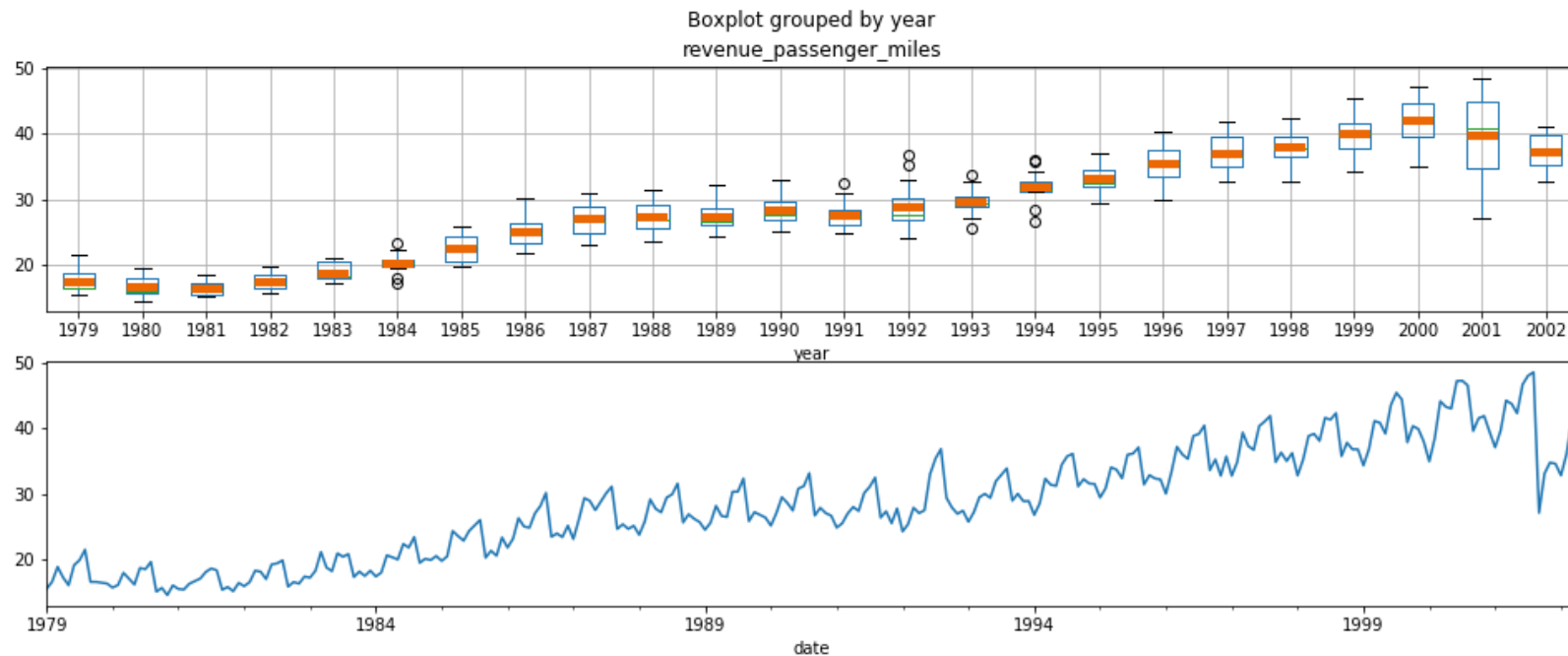
Moving averages and the idea of smoothing

- **Recall the seasonal (additive model) decomposition steps:**

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):
$$\epsilon_t = Y_t - T_t - S_t$$
- To compute the trend component T_t we need to **smooth the series**
 - One way to do it is to average the values like we did when we computed quarterly and seasonal means
 - A more robust way is to create a new series where those averages come from overlapping time windows

Boxplots: yearly means

- Follow the **mean** lines in each box plot: each line represents a **yearly average**
- Compare the curve that the **mean** lines form to the curve of the original series and describe what you see



Moving averages and the idea of smoothing

Computing simple (trailing) moving averages

- The formula to compute m_t is simple: it's an average of the values within the given window w :

$$m_t = \frac{1}{w}(Y_t + Y_{t-1} + \dots + Y_{t-w+1})$$

- Rolled into a summation, it looks like this:

$$\frac{1}{w} \sum_{j=t-w+1}^t Y_j$$

Computing simple (trailing) moving averages

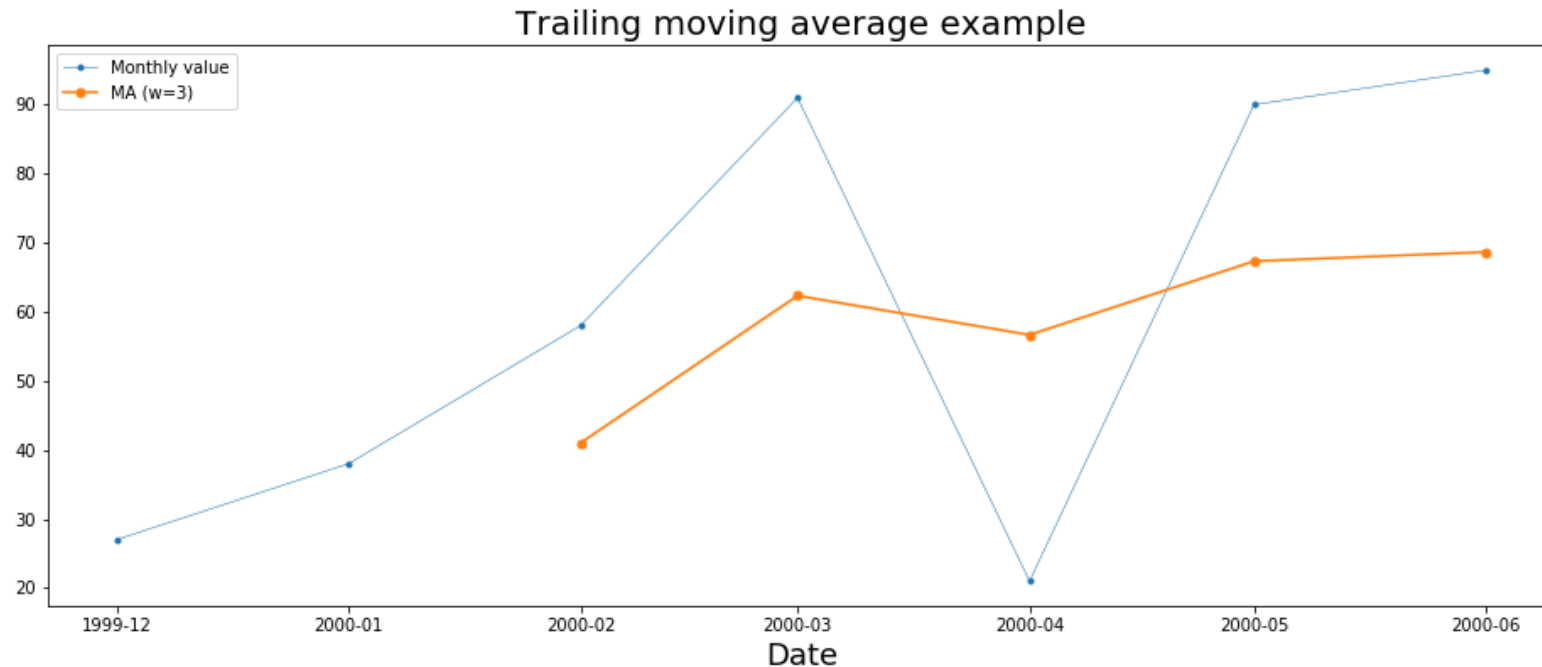
Computing simple (trailing) moving averages

- Below is a worked example of the simple trailing moving averages for $w = 3$

month	value	ma_formula	ma_value
Dec	27	NA	NA
Jan	38	NA	NA
Feb	58	$m_1 = 1/3(\text{Dec} + \text{Jan} + \text{Feb})$	41.00000
Mar	91	$m_2 = 1/3(\text{Jan} + \text{Feb} + \text{Mar})$	62.33333
Apr	21	$m_3 = 1/3(\text{Feb} + \text{Mar} + \text{Apr})$	56.66667
May	90	$m_4 = 1/3(\text{Mar} + \text{Apr} + \text{May})$	67.33333
Jun	95	$m_5 = 1/3(\text{Apr} + \text{May} + \text{Jun})$	68.66667

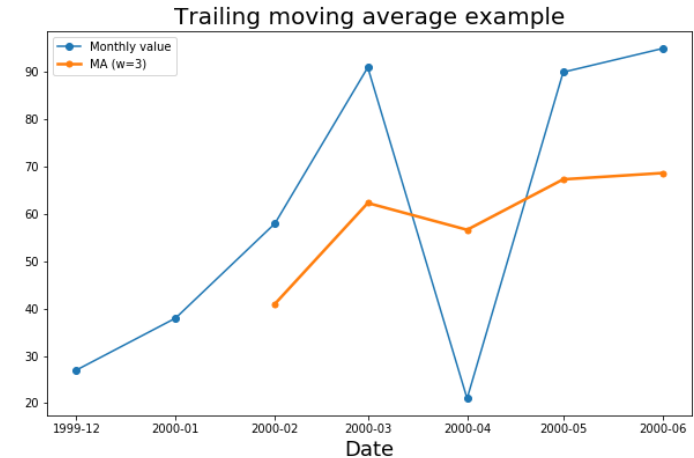
Computing simple (trailing) moving averages

- Take a look at this plot of the trailing averages
- Notice the overall shape of the **orange** curve
 - It follows the general pattern
 - It lacks the sharp peaks and valleys
 - It makes the series **smoother** and yet preserves some of its variability!



Simple (trailing) moving averages: key properties

- The new **MA** series will be
 - **Smoother**: it averages the observations
 - **Correlated**: the successive **MA** values share observations
 - Equally weighted: $weight = 1/w$
- If $w = n$ (where n is the total number of time steps), the **MA** is a simple constant mean model
- If $w = 1$, then you have your data back (no smoothing)
- The forecast uses **MA** for time $Y_{t+1} = m_t$
- The above properties make **MA** a great, but simple technique to extract the **trend component** component T_t from the series!



Centered moving averages

- We've discussed the **trailing** moving averages that are used for smoothing and **prediction**
- We use **centered** moving averages for **seasonal decomposition**
- The formula to compute **centered** m_t is the same except for 1 thing: the w is **centered** around t , instead of **trailing** it

$$m_t = \frac{1}{w} (Y_{t-k} + \dots + Y_{t+k})$$

- Rolled into a summation, it looks like this:

$$\frac{1}{w} \sum_{j=-k}^k Y_{t+j}$$

Moving averages of moving averages

- For **odd values**, the **midpoint** of w is used to center the window
- For **even values** of w , **moving average of a moving average** is used
 - It's known as $2 \times w - MA$ and it centers the MA around the t
 - Below we show an example of $2 \times 4 - MA$, a moving average of $w = 4$ is computed first and then another moving average of $w = 2$ is computed on top of it

month	value	ma4_value	ma4_formula	ma2_value	ma2_formula
Dec	27	NA	NA	NA	NA
Jan	38	53.50	$4m_1 = \frac{1}{4}(\text{Dec} + \text{Jan} + \text{Feb} + \text{Mar})$	NA	NA
Feb	58	52.00	$4m_2 = \frac{1}{4}(\text{Jan} + \text{Feb} + \text{Mar} + \text{Apr})$	52.750	$2m_1 = \frac{1}{2}(4m_1 + 4m_2)$
Mar	91	65.00	$4m_3 = \frac{1}{4}(\text{Feb} + \text{Mar} + \text{Apr} + \text{May})$	58.500	$2m_2 = \frac{1}{2}(4m_2 + 4m_3)$
Apr	21	74.25	$4m_4 = \frac{1}{4}(\text{Mar} + \text{Apr} + \text{May} + \text{Jun})$	69.625	$2m_3 = \frac{1}{2}(4m_3 + 4m_4)$
May	90	68.25	$4m_5 = \frac{1}{4}(\text{Apr} + \text{May} + \text{Jun} + \text{Jul})$	71.250	$2m_4 = \frac{1}{2}(4m_4 + 4m_5)$
Jun	95	NA	NA	NA	NA
Jul	67	NA	NA	NA	NA

Moving averages of moving averages: why bother?

- We calculate this to make an even-order moving average **symmetric** AND **weighted**!
 - When $2 - MA$ follows another even-order MA, it **makes it centered around Y_t**
 - It **weights the first and the last observation** differently from the middle ones

$$T_t = \frac{1}{2} \left[\frac{1}{4} (Y_{t-2} + Y_{t-1} + Y_t + Y_{t+1}) + \frac{1}{4} (Y_{t-1} + Y_t + Y_{t+1} + Y_{t+2}) \right]$$

$$T_t = \frac{1}{8} Y_{t-2} + \frac{1}{4} Y_{t-1} + \frac{1}{4} Y_t + \frac{1}{4} Y_{t+1} + \frac{1}{8} Y_{t+2}$$

- This is a great way to estimate *trend-cycle* T_t from seasonal data
- When applied to quarterly data, each quarter of the year is given equal weight as the first and last terms apply to the same quarter in consecutive years
- The seasonal variation will be averaged out and the resulting values of T_t will have little or no seasonal variation remaining!

Calculate detrended series

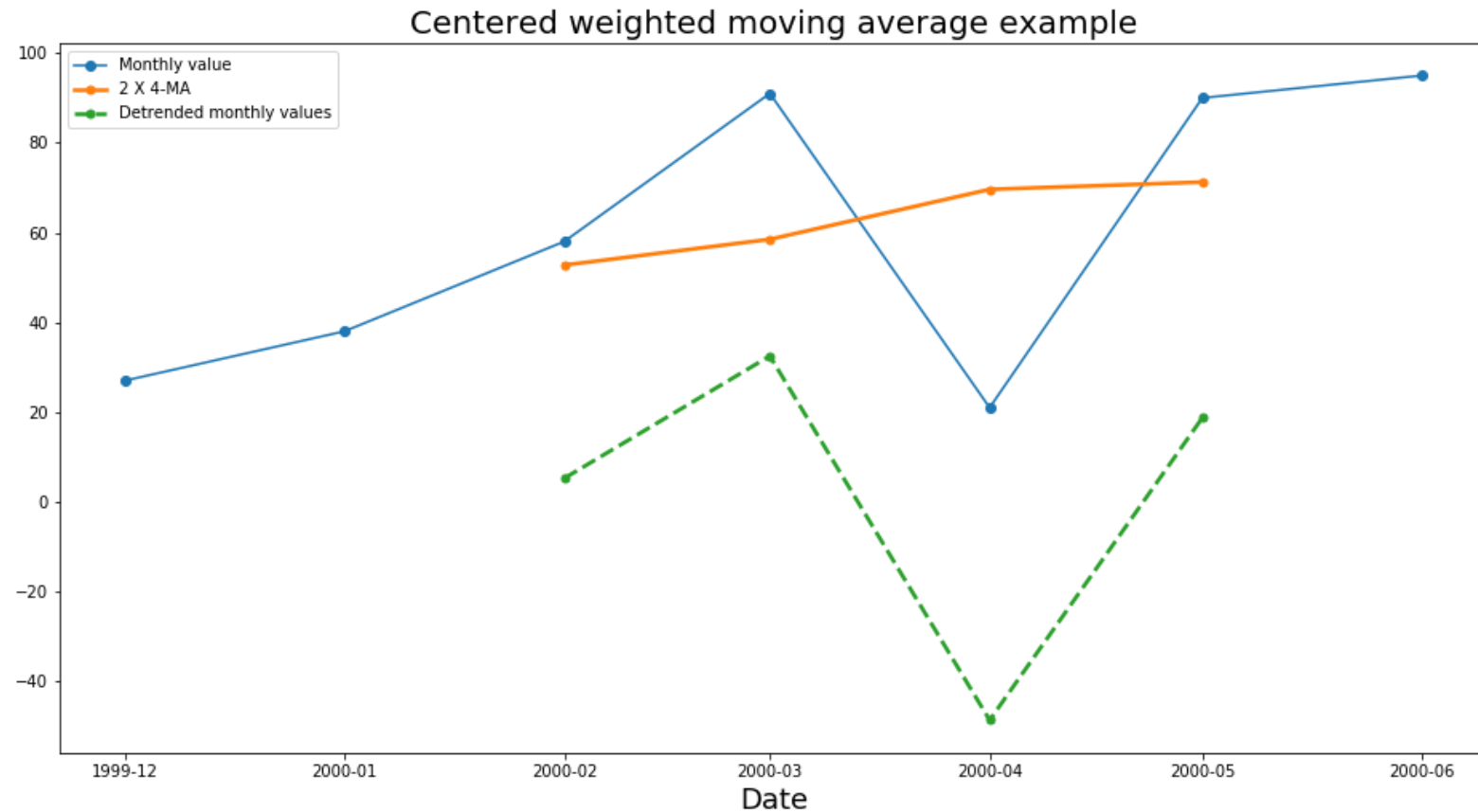
- Recall the seasonal (additive model) decomposition steps:

1. Compute the trend component T_t ✓
2. Calculate the detrended series:

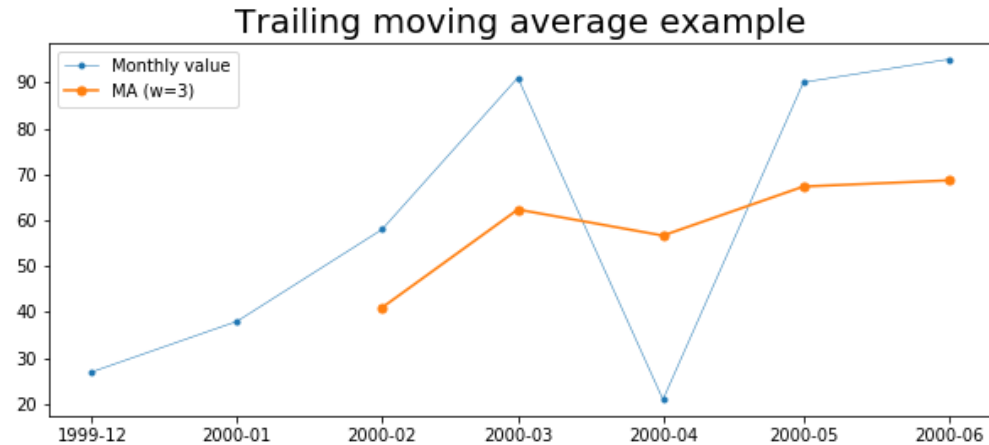
$$Y_t - T_t = S_t + \epsilon_t \quad \checkmark$$

month	value	ma2_value	detrended
Dec	27	NA	NA
Jan	38	NA	NA
Feb	58	52.750	5.250
Mar	91	58.500	32.500
Apr	21	69.625	-48.625
May	90	71.250	18.750
Jun	95	NA	NA

Series vs trend vs detrended series



Trailing vs centered weighted MA



- The **first** plot shows the simple trailing averages with $w = 3$
 - It **smooths** the series, BUT it **retains seasonal variations**
 - Good for **predictions**
- The **second** plot shows the $2 \times 4 - MA$ (weighted and centered) averages
 - It **smooths** the series, AND it **removes seasonal variations**
 - Good for **seasonal decomposition**

Knowledge check 3



Module completion checklist

Objective	Complete
Define the concept of seasonality and trend in a time series model	✓
Utilize <code>DatetimeIndex</code> properties to augment time series data	✓
Visualize time series data for different time periods using line and boxplots	✓
Explain the idea of moving averages and how it can be used to decompose time series	✓
Deseasonalize and detrend a time series model using moving averages	

Seasonal decomposition: what's left?

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 residuals (deseasonalized and detrended series, a.k.a. the error term):
$$\epsilon_t = Y_t - T_t - S_t$$
- The seasonal component estimation requires data for multiple years to demonstrate its inner workings
 - The table on the next slide shows the remaining steps already pre-computed
 - The real world data gets decomposed by using statistical packages and software that perform all of the steps in one swoop

Seasonal decomposition: sample data

month	year	value	trend	seasonal	residuals
Dec	1999	27	NA	11.4109848	NA
Jan	2000	38	NA	0.0916667	NA
Feb	2000	58	52.750	-3.4843434	8.734343
Mar	2000	91	58.500	3.9323232	28.567677
Apr	2000	21	69.625	-2.5260101	-46.098990
May	2000	90	71.250	9.5156566	9.234343

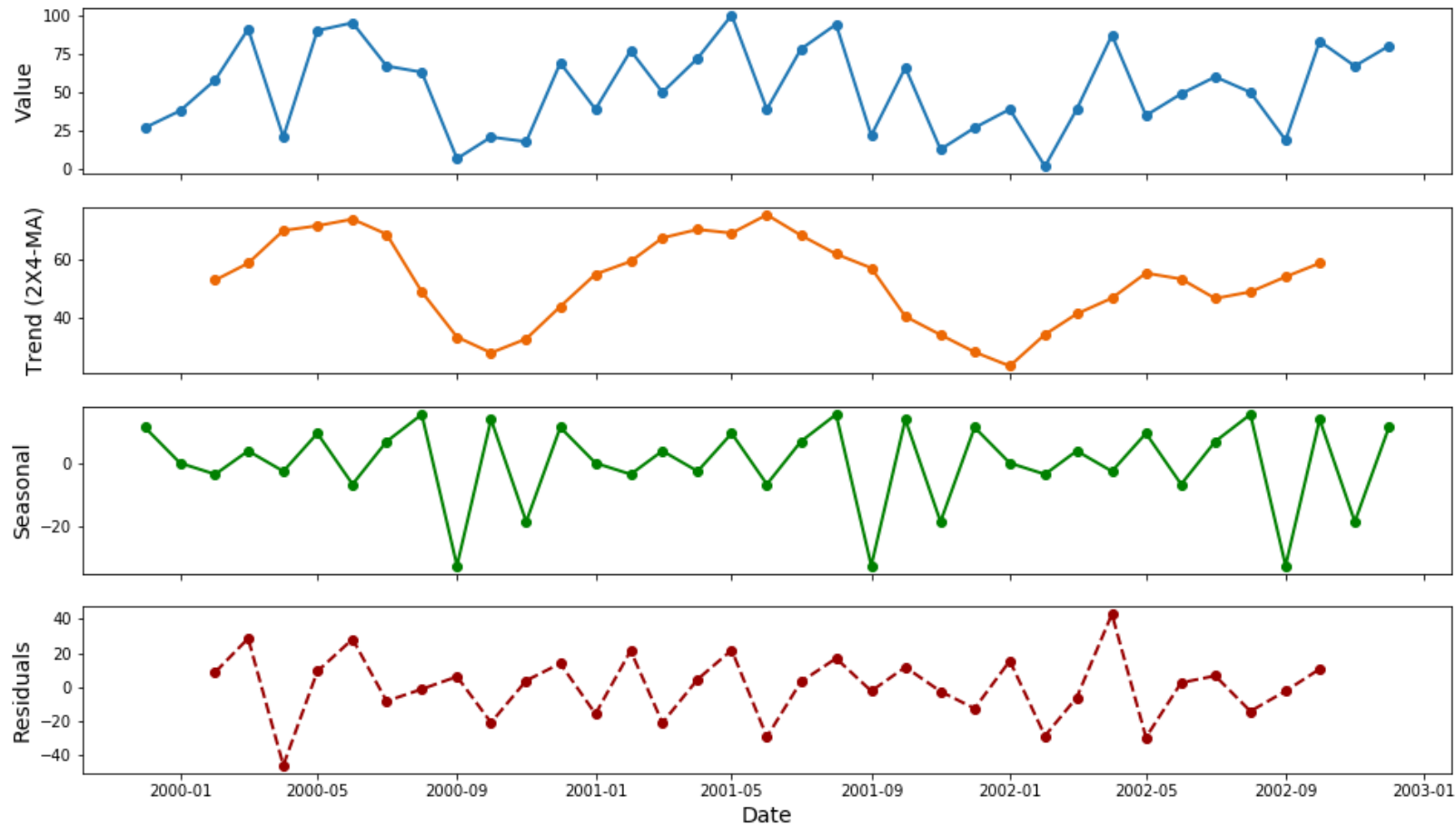
...

month	year	value	trend	seasonal	residuals
Jul	2002	60	46.500	6.932323	6.567677
Aug	2002	50	48.750	15.515657	-14.265657
Sep	2002	19	53.875	-32.484343	-2.390657
Oct	2002	83	58.500	13.932323	10.567677
Nov	2002	67	NA	-18.526515	NA
Dec	2002	80	NA	11.410985	NA

Seasonal decomposition: sample data

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 residuals (deseasonalized and detrended series, a.k.a. the error term):
 $\epsilon_t = Y_t - T_t - S_t$ ✓

What do components of the additive model look like?



Seasonal decomposition: methods

- We have used **classical decomposition** in the previous slides, which is a simple procedure that is a starting point for many other decomposition methods
- **X11** is a method developed by *US Census Bureau* and is built on classical decomposition to mitigate some of its drawbacks
- **SEATS** (Seasonal Extraction in ARIMA Time Series) was developed by the Bank of Spain and is great for decomposing monthly and quarterly seasonal data
- **STL** (Seasonal and Trend decomposition using LOESS) is a great method that uses LOESS smoothing and works equally well for seasonal data of virtually all types (e.g. hourly, daily, monthly, etc.)

Seasonal decomposition in python

- Although the seasonal decomposition is quite a process, the `statsmodels` library allows us to do it with a single line
- We will use `statsmodels.tsa.seasonal` module's function `seasonal_decompose()` that takes a single required argument, `time_series`

`statsmodels.tsa.seasonal.seasonal_decompose`

`statsmodels.tsa.seasonal.seasonal_decompose(x, model='additive', filt=None, freq=None, two_sided=True, extrapolate_trend=0)` [\[source\]](#)

Seasonal decomposition using moving averages

Parameters:

x : array-like

Time series. If 2d, individual series are in columns.

- The model type is *additive* by default - for more parameters, view the [documentation](#)

Seasonal decomposition: passenger miles data

- Let's decompose our passenger miles series

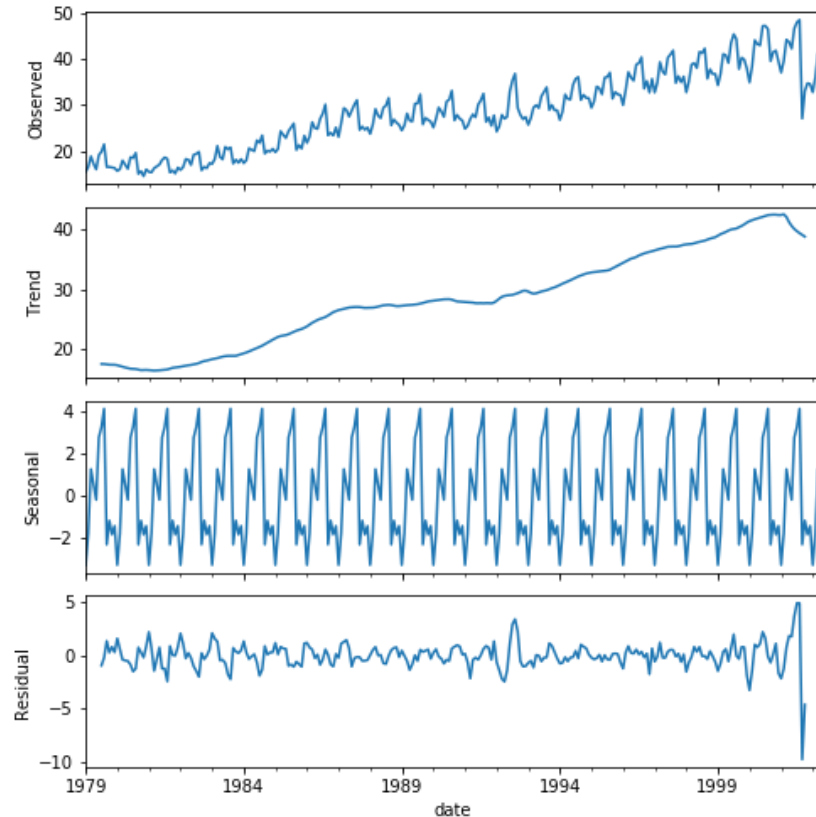
```
# Decompose revenue passenger miles into its deterministic components.  
res = seasonal_decompose(passenger_miles['revenue_passenger_miles'])  
print(res)
```

```
<statsmodels.tsa.seasonal.DecomposeResult object at 0x1c30eeb2e8>
```

- The resulting object called `DecomposeResult` and contains the individual components

Seasonal decomposition: visualize components

```
res.plot()  
plt.show()
```



The `DecomposeResult` object contains 4 components:

1. The original series Y_t
2. The trend component T_t
3. The seasonal component S_t
4. The residuals (the error component) ϵ_t

Seasonal decomposition: get trend component

```
# Extract just the trend component.  
trend = res.trend  
print(trend.head(5))
```

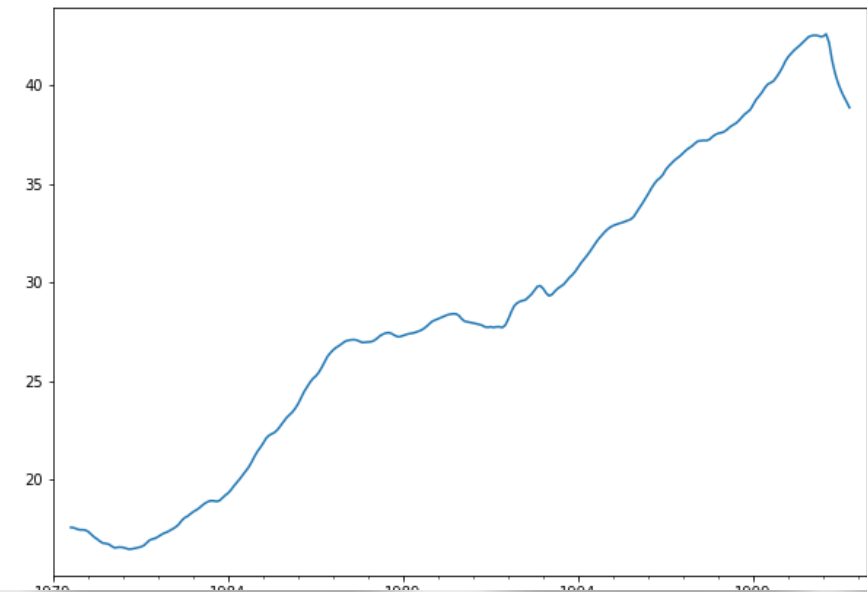
```
date  
1979-01-01    NaN  
1979-02-01    NaN  
1979-03-01    NaN  
1979-04-01    NaN  
1979-05-01    NaN  
Name: revenue_passenger_miles, dtype: float64
```

```
# Plot just the trend component.  
plt.subplots(figsize=(10,7))
```

```
(<Figure size 2000x1400 with 1 Axes>,  
<matplotlib.axes._subplots.AxesSubplot object at  
0x1c39975eb8>)
```

```
trend.plot()  
plt.show()
```

- What can you tell about the data by looking at the trend component alone?



Seasonal decomposition: get detrended series

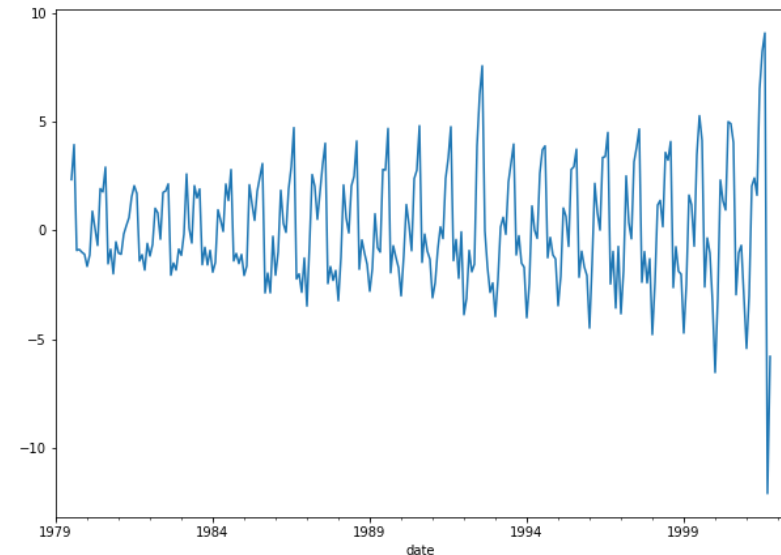
```
# Create detrended series by subtracting
# the trend component from the main series.
detrended =
passenger_miles['revenue_passenger_miles'] - trend
print(detrended.head(5))
```

```
date
1979-01-01    NaN
1979-02-01    NaN
1979-03-01    NaN
1979-04-01    NaN
1979-05-01    NaN
Name: revenue_passenger_miles, dtype: float64
```

```
# Plot detrended series.
plt.subplots(figsize = (10, 7))
```

```
(<Figure size 2000x1400 with 1 Axes>,
<matplotlib.axes._subplots.AxesSubplot
object at 0x1c34d3d780>)
```

```
detrended.plot()
plt.show()
```



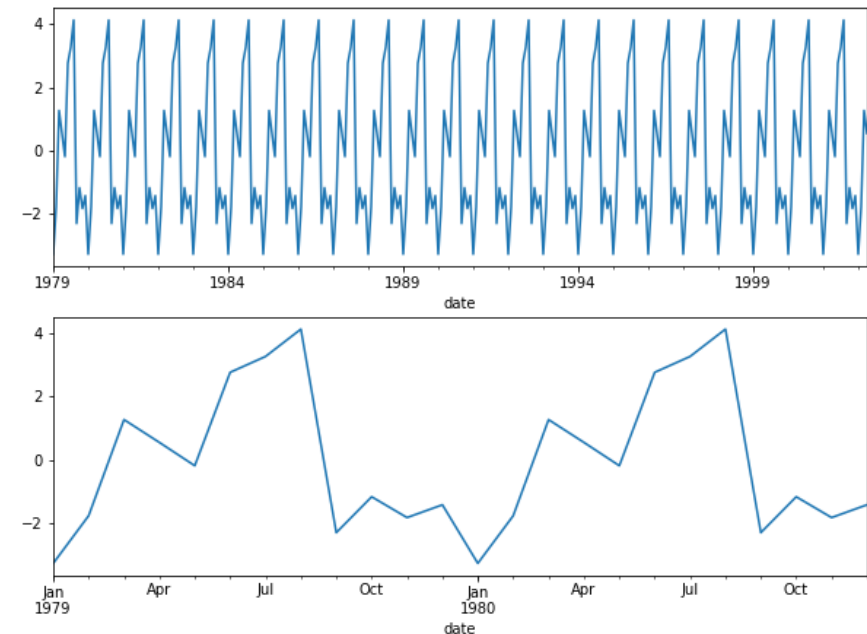
Seasonal decomposition: get seasonal component

```
# Extract just the seasonal component.  
seasonal = res.seasonal  
print(seasonal.head(5))
```

```
date  
1979-01-01    -3.273325  
1979-02-01    -1.770996  
1979-03-01     1.266012  
1979-04-01     0.546125  
1979-05-01    -0.188988  
Name: revenue_passenger_miles, dtype: float64
```

- What can you say about it when looking at just the seasonal component?

```
fig, axes = plt.subplots(ncols = 1, nrows = 2,  
                        figsize = (10, 7))  
  
# Plot the seasonal component.  
seasonal.plot(ax = axes[0])  
# Plot the seasonal component (zoomed in).  
seasonal.loc['1979':'1980'].plot(ax = axes[1])  
plt.show()
```



Seasonal decomposition: deseasonalized series

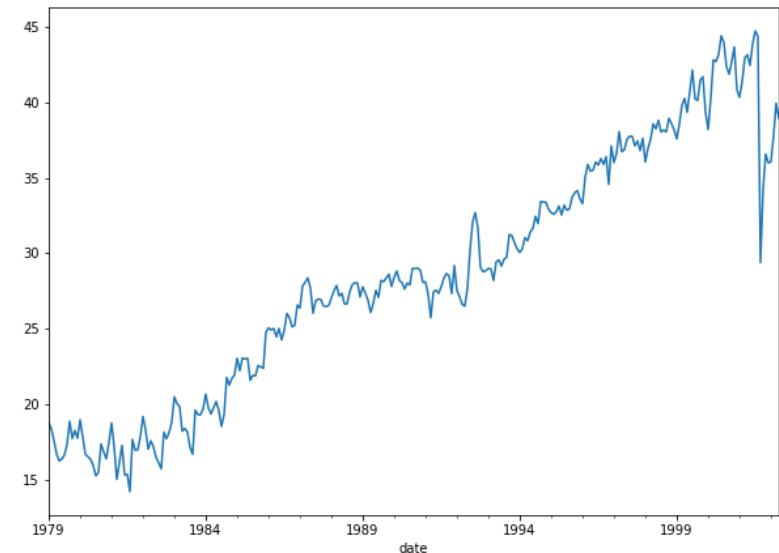
```
# Create deseasonalized series by subtracting
# the seasonal component from the main series.
deseasonalized =
passenger_miles['revenue_passenger_miles'] - seasonal
print(deseasonalized.head(20))
```

```
date
1979-01-01    18.773325
1979-02-01    18.350996
1979-03-01    17.583988
1979-04-01    16.683875
1979-05-01    16.228988
1979-06-01    16.349651
1979-07-01    16.617927
1979-08-01    17.344448
1979-09-01    18.853633
1979-10-01    17.721006
1979-11-01    18.247890
1979-12-01    17.734272
1980-01-01    18.953325
1980-02-01    17.840996
1980-03-01    16.673988
1980-04-01    16.503875
1980-05-01    16.338988
1980-06-01    15.909651
1980-07-01    15.237927
1980-08-01    15.474448
Name: revenue_passenger_miles, dtype: float64
```

```
# Plot deseasonalized series.
plt.subplots(figsize = (10, 7))
```

```
(<Figure size 2000x1400 with 1 Axes>,
<matplotlib.axes._subplots.AxesSubplot
object at 0x1c36ba0a20>)
```

```
deseasonalized.plot()
plt.show()
```



Seasonal decomposition: get residual component

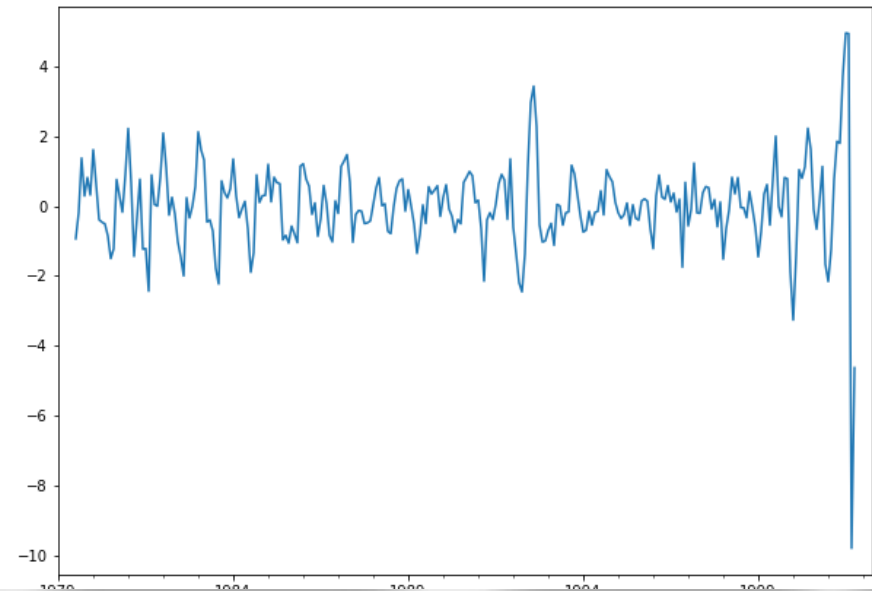
```
# Extract just the residuals.  
residuals = res.resid  
print(residuals.head(20))
```

```
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  
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  
1980-05-01    -0.506845  
1980-06-01    -0.845766  
1980-07-01    -1.497073  
1980-08-01    -1.223468  
Name: revenue_passenger_miles, dtype: float64
```

```
# Plot just the residuals.  
plt.subplots(figsize = (10, 7))
```

```
(<Figure size 2000x1400 with 1 Axes>,  
<matplotlib.axes._subplots.AxesSubplot object at  
0x1c3011eeb8>)
```

```
residuals.plot()  
plt.show()
```



Which components to use?

- Is the variation due to seasonality important?
 - Explore the seasonal component S_t
 - If you would like to know which day of the week is best for air travel, you would look at the seasonal component alone (at the summary statistics for each weekday!)
- Is the variation due to seasonality skewing the data?
 - Explore deseasonalized series $Y_t - S_t = T_t + \epsilon_t$
 - Seasonally adjusted series contain the residuals component as well as the trend-cycle and can be used for forecasting using regression, random walk (with or without drift), ARIMA and other methods
- Is the purpose of your analysis to look for turning points in a series and interpret any changes in direction?
 - Explore the trend-cycle component alone T_t
- Is model validity of primary concern?
 - Explore the residuals component ϵ_t

Passenger miles data: trend vs seasonality

When we look at the decomposed passenger miles data, we can

- Analyze seasonal patterns in how much people travel in different months of the year
- Analyze an overall trend
 - How much has passenger mobility changed over the years?
 - Were there any turning points in passenger mobility? If so, when?



Passenger miles decomposed data next steps

- In the next and last module of time series analysis, we will learn:
 - About ARIMA model and its components
 - How to recognize when to use certain cases of ARIMA models
 - About ARIMA model applications
 - How to forecast on deseasonalized series $Y_t - S_t = T_t + \epsilon_t$ using an ARIMA model

Knowledge check 4



Exercise 3



Module completion checklist

Objective	Complete
Define the concept of seasonality and trend in a time series model	✓
Utilize <code>DatetimeIndex</code> properties to augment time series data	✓
Visualize time series data for different time periods using line and boxplots	✓
Explain the idea of moving averages and how it can be used to decompose time series	✓
Deseasonalize and detrend a time series model using moving averages	✓

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:

- Augment time series data you have using `DatetimeIndex` properties
- Visualize time series data for different time periods using line and boxplots
- Decompose time series data using additive seasonal decomposition
- Discuss results and describe any patterns you have found

Congratulations on completing this module!

