Working with Data Collected Over Time

1

1.1 INTRODUCTION

Welcome to the fascinating world of Time Series Analysis! Especially for the statistician, data scientist, or quantitative analyst who must analyze a wide variety of data types, time series analysis is an important field of study. The applicability of time series analysis to "real-world" data will be frequently exemplified throughout the text. While the focus of this book is on application, sufficient theoretical detail is provided for the reader to appreciate the underlying techniques.

In recent years, there has been a major increase in the availability of vast amounts of data. This relatively new phenomenon has (fortunately, for you!) created a shortage of individuals who can analyze data, offering attractive employment and career opportunities for those willing to learn how to analyze such data. Furthermore, of the many types of available data, it is important to note that a substantial share is categorized as *time series data*. Consequently, industrial analytics groups have realized the value of utilizing time series analysis techniques for analyzing real data. As this developing pattern continues, it is inevitable that knowledge of time series will become a common expectation of professional data analysts, so learning how to analyze time series data will be beneficial (and lucrative!) for your career.

Why are time series data different?

It is important to understand that time series data cannot be appropriately or accurately analyzed using standard statistical techniques most frequently learned in undergraduate and even graduate statistics courses. In these courses, a key assumption is that the data are randomly sampled, resulting in independent observations. This assumption is violated for time series data, and it is inappropriate to apply classical statistical analysis techniques in such time-dependent settings. Fortunately, important (but different) information is present in time series data, and corresponding adjustments in the methodology take advantage of this information. The goal of this book is to address this very issue; that is, the book is designed so that readers will be equipped with the tools necessary to analyze time series data that are encountered in practice.

Various time series applications

To which types of naturally occurring settings does time series apply, and why? Consider the following scenarios and identify a common theme. A cardiologist monitors a patient's cardiogram results to recognize irregular heartbeats over time. A geologist studies a seismograph to identify unusual deviations in recent days which may help predict a future earthquake. A stockbroker closely tracks stock prices over a time period of interest to optimize investment strategies. A major retail chain utilizes historical data to predict consumer demand of a particular product throughout the year to better manage inventory. A professional sports team uses data on annual ticket sales to correlate the delayed time effect of widespread advertising campaigns on ticket sales. A climatologist studies global temperatures over an extended period of time to detect potential shifting patterns in temperatures. What key word do all of these scenarios have

DOI: 10.1201/9781003089070-1

in common? *Time!* In each of these instances, data are measured sequentially in time, and the questions of interest revolve specifically around how the corresponding variables are dependent on time.

Relevance of time series

Why is time series analysis a relevant field of study? As illustrated in the previous examples, many important questions are associated with time series data, and the analysis of such data can help answer these questions. For example, what if the cardiologist discovers, by studying cardiogram data, that lives could be saved by detecting a previously unidentified abnormality in heartbeats over time? Or, what if the retail chain determines with a high degree of certainty how seasonal patterns predict customer demand of a product of interest, suggesting a better inventory strategy to maximize profit? What if the stockbroker is able to identify subtle indicators in the recent daily markets which give evidence that a particular stock purchase is likely to yield high returns? The wide variety of this relatively short list of examples further supports the applicability and relevance of time series data.

Perhaps an even more visible example of relevant time series data occurred (ironically) during the writing of this book in 2020–2021. During that time, the world-wide COVID-19 pandemic raced rampantly across the globe, creating a flood of concerned interest in daily, weekly, and monthly data that would provide crucial information about new cases, cured cases, and deaths caused by COVID-19. This was a unique time frame in which essentially all news stations, radio stations, and other media sources frequently reported variations of these time series data to inform the public about the most up-to-date status of the COVID-19 virus. The daily tracking of virus deaths was a question of interest to all, and because of this widespread interest, the relevance of summarizing and processing data, and in particular, time series data, was suddenly in the world's spotlight.

A final example of time series data that has potentially significant world-wide implications had also gained much momentum around the same time frame as the COVID-19 pandemic. The introduction of Bitcoin in 2009 led, according to Forbes Advisor (2021), to the world's first "cryptocurrency"; that is, Bitcoin is a decentralized digital currency which allows individual investors to buy and sell directly without a bank. Forbes describes Bitcoin as "digital gold" that investors can use to hedge against market volatility and inflation. Interestingly, the price of the first coins was under $150 per coin, but this price had consistently surged and as of April 2021 (the time of the writing of this book) had risen to nearly $50,000. It is only speculative to guess at this time where the price will go, but it is nearly certain it will be a focus of governments and markets for years to come! It is also clear that time series analysis will be used to provide informed forecasts of future behavior.

Where to start?

These examples illustrate that time series data are prevalent and are of valuable interest to a world-wide audience. As a result, several natural questions come to mind.

- How does one gain access to such data?
- What software and coding skills are required to begin analysis of time series data?
- What mathematical and statistical knowledge is necessary to really understand time series data and to have a working knowledge of the techniques used to adequately analyze the data?
- What assumptions must be considered before making conclusions, and what are the limitations and qualifications of these conclusions?

These are the very questions that this book will answer.

Guided by invaluable industrial experience and from teaching many semesters of applied time series students, the authors have devoted much time and consideration to the presentation of topics covered in this book. It will soon become apparent that our strategy is rather different from most textbooks. In addition to presenting a plethora of useful techniques, our goal is to provide enough discussion to help readers understand the techniques without having to take a "deep dive" into the theory. The journey is intended to be a particularly "hands-on experience" – we highly encourage you to participate as you follow along! The

reader will frequently be invited to enter a set of codes to reproduce results from an example; in addition, QR codes throughout the text are provided which are linked to websites of interest and to supplementary short teaching videos that allow the interested student to learn more detail about various topics. Such activities and exercises have been designed and included to enhance your learning experience. We are pleased to be a part of your time series education, so with these thoughts in mind, have fun as you learn about the following time series topics!

1.2 TIME SERIES DATASETS

Time series data exhibit a variety of behaviors which we will discuss in the following sections. We will illustrate these behavior types using "real-world" examples such as the intriguing sunspot data, temperature data, DOW Jones and individual stock prices, sales data (monthly, daily, …), and more. We begin by discussing data that have some sort of cyclic behavior.

1.2.1 Cyclic Data

Many time series datasets have a cyclic pattern, by which we will mean that the data display rises and falls in somewhat of a repetitive fashion. Such data are sometimes referred to as "pseudo-periodic", a term that we will use synonymously with "cyclic".[1] The sunspot data in Figure 1.1 and the Dallas-Ft. Worth (DFW) monthly temperature data in Figure 1.3 are examples of cyclic data. We discuss these two datasets and examine their similarities and differences.

1.2.1.1 Sunspot Data

Figure 1.1 shows annual sunspot data for the years 1700–2020. Sunspots are areas of solar explosions or extreme atmospheric disturbances on the sun. In 1848, the Swiss astronomer Rudolf Wolf introduced a method of counting sunspot activity, and monthly data using his method are available since 1749. See Waldmeier (1961).

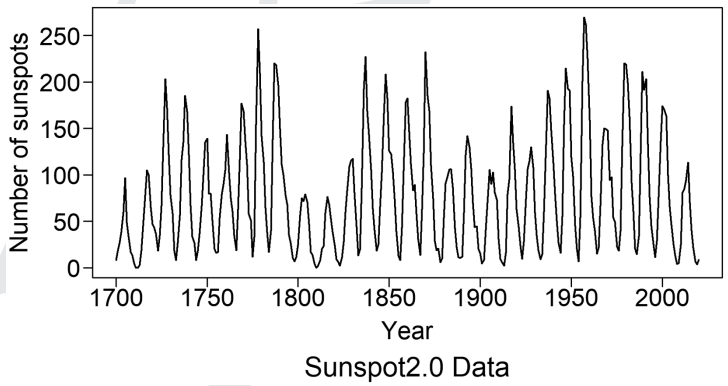


FIGURE 1.1 Annual number of sunspots from 1700 through 2020.

1 Data that are *truly periodic* have a behavior that repeats *exactly* over a fixed time frame. A good example of purely periodic data is the sine curve. Thus, pseudo-periodic (or cyclic) data are data that *tend* to repeat behaviors.

Sunspots have created a considerable amount of interest in the scientific community for two main reasons:

(1) Sunspot activity tends to affect us here on earth. For example, high sunspot activity causes interference with radio communication and is associated with higher ultraviolet light intensity and northern light activity.
(2) Sunspot activity has a cyclic behavior that has a cycle length of about 11 years. Examining Figure 1.1 shows that there are 29 cycles in the 321 years for an average cycle length of about 11 years. Actually, cycle lengths tend to randomly vary from 9 to 13 years.

While the cyclic behavior in Figure 1.1 is clear, it is often useful to examine short snippets of the data to better visualize the specific behavior. Figure 1.2 shows the number of sunspots from 1867 through 1950. The vertical lines identify the years at which there was a peak in the sunspot numbers and the horizontal arrows represent the time between the peaks. For the years plotted in Figure 1.2, the cycle lengths were 13, 10, 12, 12, 11, and 9 years, respectively. The cycle lengths seem to vary randomly, and there does not appear to be an "adjustment to a fixed cycle length". In fact, it is the understanding of these authors that scientists do not have a physical explanation for the approximately 11-year cycle. The sunspot data are a classic example of cyclic data with varying cycle lengths. In fact, Yule (1927) developed the autoregressive process as a means of describing the "disturbed" periodic behavior of the sunspot data. Recognizing this behavior will be critical when we forecast data in future chapters.

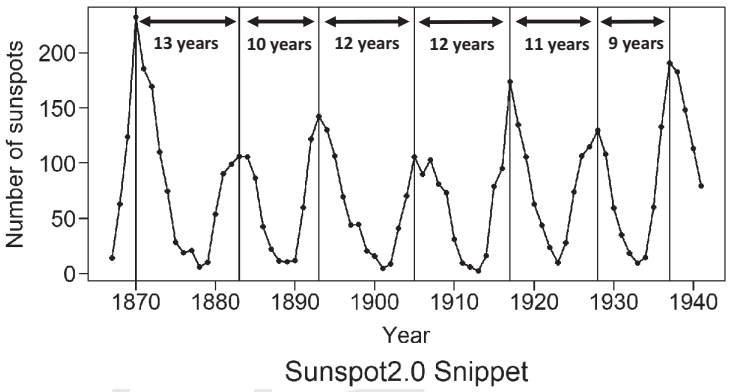


FIGURE 1.2 Snippet from Figure 1.1 showing years 1867–1950.

Key Point: Viewing time series realizations over a short time span can improve the interpretation of the data behavior.
– We will refer to this as viewing data over a *snippet* of time

It is important to note that beginning in July, 2015 a new method for enumerating sunspot activity has replaced the Wolfer[2] method, and a revised dataset has been developed. See Clette, Cliver, Lefêvre, Svalgaard, Vaquero, and Leibacher (2016). Using the new method, annual data are available from 1700 to the present, and monthly values exist for the years 1749 to the present. To avoid confusion, the new sunspot counting version is numbered 2.0. The new data are available at http://sidc.oma.be/silso/home which is discussed in Section 1.3.7.[3]

2 "Wolfer" is the commonly used name for the famous sunspot data originated by Rudolf Wolf.
3 SILSO data/image, Royal Observatory of Belgium, Brussels.

Note

1. Sunspot data used in this book will always be version 2.0.
   - Although the original version of the sunspot data has been widely published and modeled, future sunspot numbers will be of the 2.0 form.
   - We use version 2.0 to provide the ability to evaluate forecasts of future values which will be available only in 2.0 form.
2. As of the writing of this book, the sunspot data available in Base R are the original Wolfer version, and as such, are not available after July 2015.
3. We caution readers to be alert to the fact that there are now two sets of sunspot data.

1.2.1.2 DFW Temperature Data

Figure 1.3 shows the average monthly temperature for Dallas-Ft. Worth (DFW) (where the authors live) from January 1900 through December 2020. Although it is difficult to see clearly, temperatures follow the expected pattern. That is, they are low in the winter and high (for DFW very high!) in the summer.

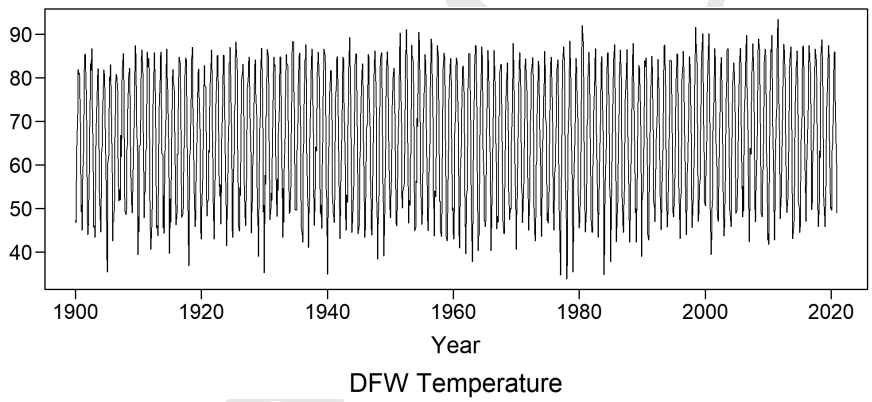


FIGURE 1.3 Average monthly temperature for Dallas-Ft. Worth for January 1900 through December 2020.

By viewing the data in Figure 1.4, we again see the value of focusing on a snippet of time, which in this case is for the years 1979 through 1986. For our purposes, we call a cycle the number of months between the hottest month of each year. We see that, in DFW, the hottest month is either July or August. Note that the temperature data have a smooth progression from summer to winter and again from winter to summer with the average temperatures in the fall and spring being similar. The resulting overall pattern is "sort of" sinusoidal, or pseudo-sinusoidal.

It is useful to note that random variations (noise) will be present in data. The hottest month in 1979 through 1981 was July.[4] The hottest month in 1982 was August, so the third cycle in the plot has length 13 months. August was also the hottest month for 1983 through 1985, so the next three cycles are of length 12 (August to August). Finally, in 1986 the hottest month was July, so that the corresponding cycle length is 11 months. While the cycle lengths were not all equal to 12 months, note that whenever a 13-month cycle occurs, it is always followed by either a 12-month or an 11-month cycle to "stay in or get back in sync". That is, suppose the hottest month in one year is July and then the two succeeding years had temperature cycle lengths of 13. That would indicate that the hottest month in the third year was September, which has never been the hottest month in DFW in the dataset available back to September of 1898. The DFW

4 Those living in the DFW area in the summer of 1980 will have that summer "burned" in their memory.

temperature data are an example of cyclic data with a *fixed* cycle length. In this case, the 12-month cycle has a physical cause − regular and predictable motion of the Earth around the Sun. Because the cycles in the temperature are related to the calendar year, the temperature data are referred to as *seasonal* data.

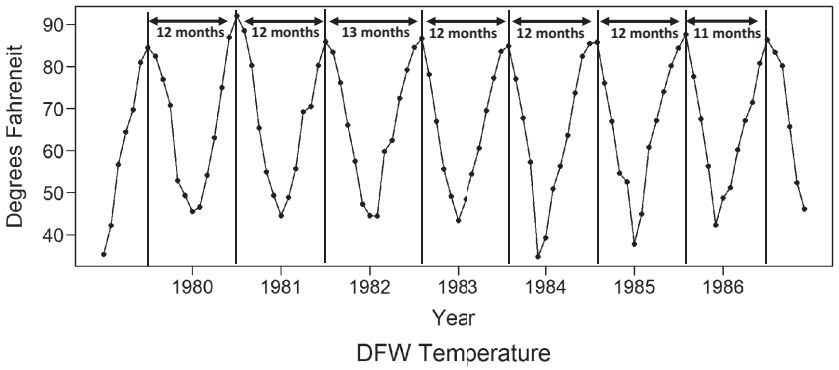


FIGURE 1.4 Average monthly temperature in Dallas, Texas from 1979 to 1986.

Note: The key difference between the sunspot data and the temperature data is that while the sunspot cycle lengths tend to vary randomly between 9 and 13 years, the temperature cycle lengths are fixed to the point that they stay "in sync" with the 12-month year.

Key Points: The sunspot and temperature data are *cyclic* data. However:

1. The sunspot cycles lengths seem to vary randomly from 9 to 13 years.
2. The temperature data have a *fixed*, physically explainable cycle length.
   - Because the cycles in the temperature are related to the calendar year, the temperature data are an example of *seasonal* data.
   - The temperature data can also be described as pseudo-sinusoidal.

1.2.1.3 Air Passengers Data

Figure 1.5 is a dataset containing the total number (in thousands) of international airline passengers per month for the 12 years from 1949−1960. This dataset has been extensively analyzed and is a classical dataset in the time series literature. The data go through a 12-month cyclic pattern that is similar from year to year and is based on the calendar year. Thus, the Air Passengers data are another example of *seasonal* data. Additionally, the data tend to be trending upward in time. That is, the number of airline passengers is increasing in time. Trending behavior in time series will be discussed in Section 1.2.2. There is also an expanding within-year variability. This type of behavior, referred to as multiplicative seasonality, will be discussed in Chapter 2.

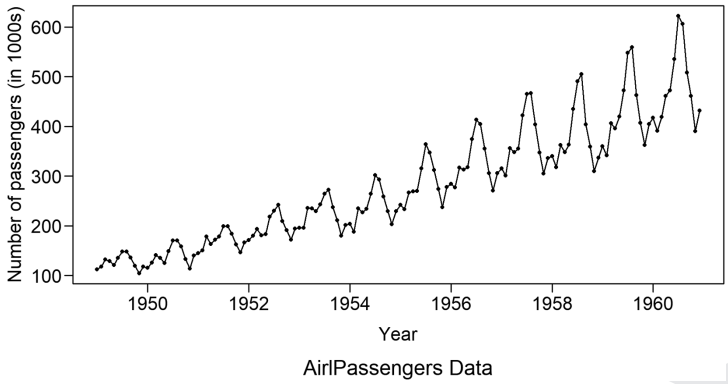


FIGURE 1.5 Number of International Passengers on Airlines from 1949 to 1960.

Figure 1.6 is a snippet of the air passenger data from 1957 to 1960. It can be seen that air travel is light from January through April, is high during the summer months, and begins to drop in September through November with a slight increase in December. This pattern, although not sinusoidal, is repeated from year to year. The cyclic behavior of the air passenger data is repeated on an annual basis, and is an example of *seasonal* data that is not pseudo-sinusoidal.

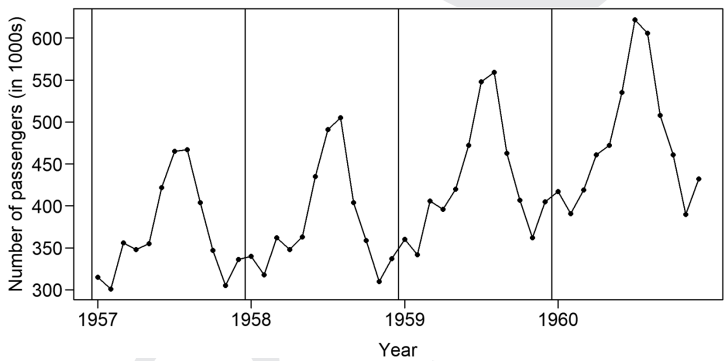


FIGURE 1.6 Number of International Passengers on Airlines from 1957 to 1960.

1.2.2 Trends

A trend is a tendency for data to increase (or decrease) steadily over time. We noted that the Air Passengers data in Figure 1.5 have an increasing trending behavior in addition to the seasonal pattern noted. A *linear* trend would be a "tendency" for the data to increase (or decrease) in a *linear* fashion (see Figure 1.7(a)). Trends might tend to follow a curve such as the exponential trending shape in Figure 1.7(b). Figure 1.7(c) is a time series that has a downward trend, but is more irregular in nature than the trends in Figures 1.7(a) and (b). A typical pattern for datasets is *random trending behavior* such as that in Figure 1.7(d), which has the appearance of aimless *wandering*. That is, there may be a series of short or long trends, sometimes in opposite directions.

Key Point: Data with trending and random wandering behavior are not cyclic in nature. They are sometimes called aperiodic (not periodic) because there is no regular rise and fall behavior.

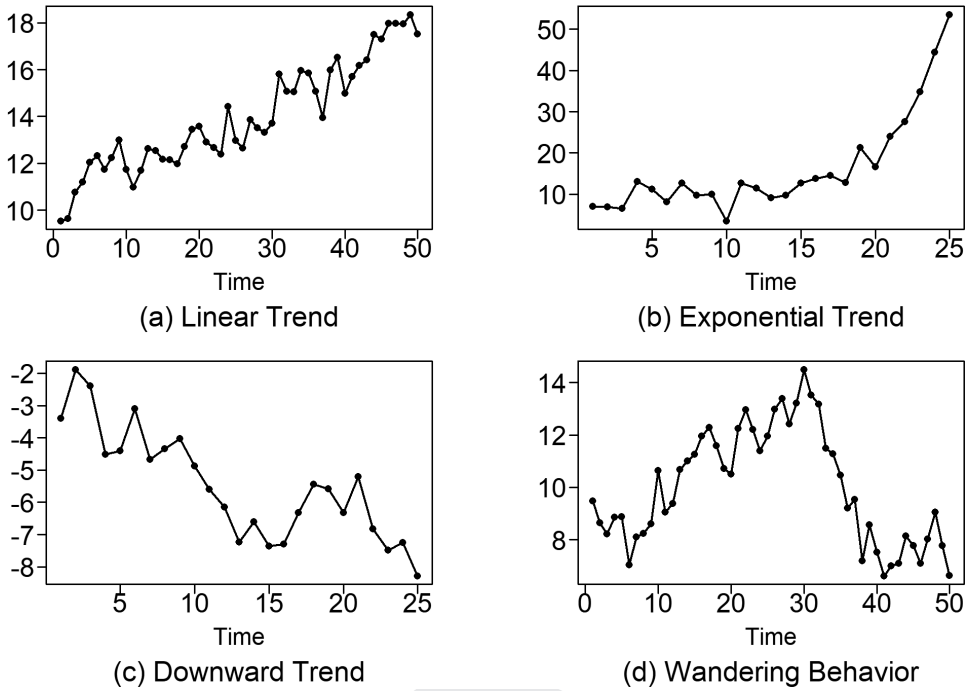


FIGURE 1.7 Plots showing (a) a linear trend, (b) an exponential trend, (c) a downward irregular trend, and (d) wandering pattern.

Trends may be the main feature of a set of time series data such as for the time series in Figure 1.7. However, the air passenger data in Figure 1.5 show that a set of data may have seasonal and trending behavior. The possible variations are endless.

1.2.2.1 Real Datasets That Have Trending Behavior

(1) Monthly Dow Jones Closing Average

The Dow Jones (DOW) has been used as a measure of the health of the US stock market for over 100 years. Figure 1.8(a) shows the monthly closing averages for the years 1985 through 1995. There we see an upward trending behavior with a dip associated with Black Monday.[5] However, the DOW averages recovered and continued to increase far beyond the pre-Black Monday levels.

(2) West Texas Intermediate Crude Oil Price

Figure 1.8(b) is a plot of monthly price of West Texas Intermediate Crude Oil (WTI) from January 1990 through June 2008. The price of this grade of oil (affectionately known as "Texas Light Sweet") is used as a benchmark for fuel prices around the world and can often be seen in professional reports and heard in newscasts. The data shown in Figure 1.8(b) appear to have an even more explosively increasing trend than the DOW data in Figure 1.8(a).

(3) Bitcoin

Figure 1.8(c) shows daily Bitcoin closing prices from January 1, 2021 through April 30, 2021. These data tend to be trending upward and are of interest because the use of cryptocurrency (or virtual money) is in

5 October 19, 1987 is referred to as Black Monday because the DOW dropped almost 22% to become one of the most notorious days in US financial history.

its infancy. The financial experts, at the time of this writing, are split on their predictions concerning the growth (or even the permanence) of this type of currency.

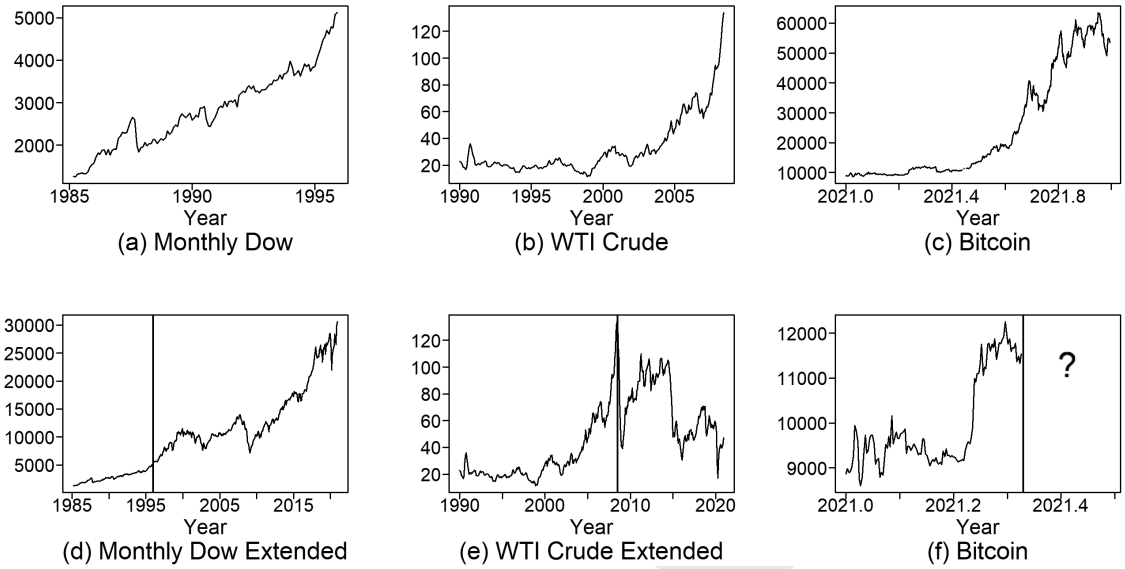


FIGURE 1.8 (a) Monthly DOW closing averages from March 1985 through December 1995, (b) monthly WTI Crude oil prices from January 1990 through June 2008, (c) Daily Bitcoin prices from January 1, 2021 through April 30, 2021, (d) monthly DOW closing averages from March 1985 through December 2020, (e) WTI crude oil prices from January 1990 through December 2020, and (f) Bitcoin prices through April 2021.

1.2.2.2 The Problem with Trends

Plots of data such as those in Figures 1.8(a)−(c) are informative in the sense that we gain information about the behavior of a time series *within the time frame of the plot*. For example, Black Monday shows up clearly and the plot helps us to understand its impact. However, in most cases the question of interest is, "*Will the current trending behavior continue?*" To illustrate the "flaky" nature of trends, we extend the time frame for the DOW and WTI datasets. Figure 1.8(d) shows the DOW data from 1985 through 2020 with a line drawn at the end of 1995 where Figure 1.8(a) ended. There it is seen that the trending behavior continues after 1995 "on the whole". The increase was not without its temporary "dips". For example, it is easy to see the effects of the Great Recession between 2007−2009 and the COVID effects on the market in 2020. However, the stock market continues to increase, and as of the writing of this book, is above "pre-COVID" levels.

Note that in June 2008, based on the WTI data in Figure 1.8(b), one would probably predict a continued rise in oil prices such as had been seen in the previous 15 years. Banking heavily on such a prediction would not have been wise! Note that, in contrast to the DOW data, the WTI data in Figure 1.8(e) dipped precipitously in late 2008 and then wandered without displaying much evidence of sinusoidal, seasonal, or overall trending behavior. Any perceived trends in such data may vanish at a moment's notice. This is an important characteristic that is prevalent in many time series datasets, for example, stock market data for individual stocks. Again, understanding this behavior will be useful in modeling and forecasting such datasets.

In Figure 1.8(f) we again show the Bitcoin prices for 2021 through April (the time of this writing). The data show a nice upward trend, but time will tell whether the trend continues.

Key Points:

1. Trending behavior in a time series provides information about the time frame *in which the data were collected*.
2. Predictions that trending behavior will continue is analogous to *extrapolating* in regression beyond the range of the predictor variables.
3. We will discuss trending behavior and tests for trend in Chapters 7–8.
4. At this point we note that predictions about future values should involve domain knowledge as well as statistical analysis.
   - For example, while in the short term, the stock market tends to "bounce around". a longer range look indicates that a continued long-term trend should be expected to continue unless conditions change regarding the strength of the US economy. See Figure 1.9.

Figure 1.9 shows the year-end DOW price for the years 1915 through 2020. A continued upward trend overshadows isolated behaviors like the Great Depression beginning in 1929, Black Monday in 1987, the Great Recession in 2007–2009, and so forth. Data such as that in Figure 1.9 might lead the analyst to predict a long-term increase in the DOW. *Caution:* This obviously does not hold for individual stocks because a corporation or entire industry may cease to be viable.

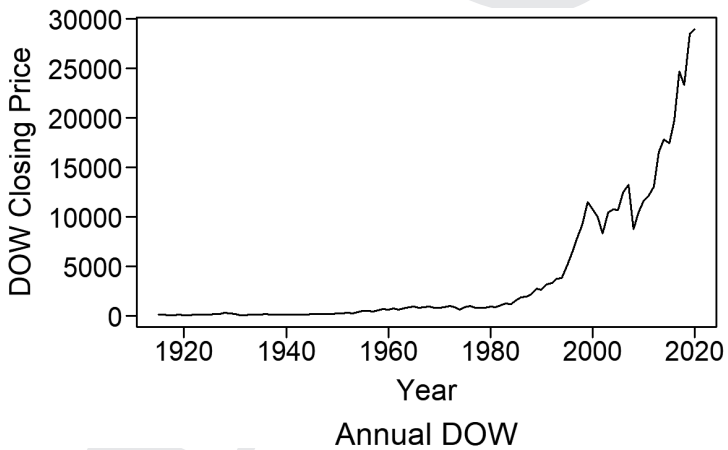


FIGURE 1.9 Annual year-end DOW closing prices for 1915 through 2020.

1.3 THE PROGRAMMING LANGUAGE R

Modern time series analysis requires proficiency in one or more of the readily available software tools such as R, Python, SAS, SPSS, Minitab, and STATA just to name a few. We will focus on R in this text as it is easily available on Mac, PC, and Linux operating systems, is open source with over 12,000 user supplied packages in the CRAN repository, is widely used by the academic, industrial, and commercial communities, and has a free development environment called RStudio.

There are an increasing number of data science consulting companies. One of the leaders is a company named *Fingent*, which states in its comparison between R and other software (Python, etc.) with respect to time series analysis:

R offers one of the richest ecosystems to perform data analysis. Since there are 12,000 packages in the open-source repository, it is easy to find a library for any required analysis. Business managers will find that its rich library makes R the best choice for statistical analysis, particularly for specialized analytical work.

…

Since R is developed by academicians and scientists, it is designed to answer statistical problems. It is equipped to perform time series analysis. It is the best tool for business forecasting.

–Fingent

A minimal level of R experience is assumed, although we provide video resources to bring those new to R up to this minimal level. The video resources associated with this book are available via QR code as well as by means of a list of URLs on the website. The QR code can be accessed by simply pointing the camera of most smartphones at the QR Code, after which a prompt automatically accesses the video on the web. This is a nice feature because of the ability to access video instruction on a smartphone or PC while working on the computer. We believe that this feature is both quick and intuitive. The first QR code accessible video in this book is a tutorial about loading R and RStudio, which is a great way to get a running start!

QR 1.1 Installing R and RStudio

1.3.1 The *tswge* Time Series Package

The R package ***tswge,*** which is available both on CRAN and on the book's GitHub site, provides a set of functions and datasets to accompany this text. As mentioned above, time series data may possess a wide variety of characteristics and behaviors. This fact, combined with the various types of questions of interest that may be paired with the data, motivates the methods, models, tests, and techniques presented in this book. These tools have been collected in ***tswge***. The ***tswge*** functions will be introduced in the text, and then details about each function's purpose, input parameters involved in the function statement, and output variables are given in an appendix to the chapter in which the function is introduced.

We believe that both inexperienced and seasoned R users will find the videos and examples in this book easy to follow and hopefully informative. Most of the coding examples are designed so that the readers can (and are very much encouraged to) run the code directly into their own R console and experience the example for themselves!

Note 1: In order to use the R code provided in the book, you will need to load R onto your computer. From this point going forward, we will assume that you have downloaded R, installed ***tswge*** from CRAN, and loaded ***tswge***.

Note 2: If you haven't done this already, NOW is the time to perform these tasks! The following step-by-step instructions will help.

Note 3: The instructions below are applicable to both PCs and Macs.

Step 1: Download R onto your computer. (RStudio is covered in the tutorial and will not be discussed here.)

(a) Go to https://cran.r-project.org/mirrors.html
(b) Select a CRAN mirror site close to you

(c) In the box labeled "Download and Install R", click on the link associated with your operating system.
(d) Click on the link *Download R xx.xx.xx* (The R version will change in time, thus the xx.xx.xx.)
(e) Open the .exe file and answer the questions to install R on your system. (We recommend that you choose "base" and the default options at each step.)

Step 2: Open the R package at which time you should see the command console.

Step 3: Install tswge

(a) Select *Packages* in the top menu on the R screen
(b) Select *Install Package(s)…*
(c) Choose a CRAN mirror closest to you
(d) You will be given a *long* list of packages. Choose ***tswge***.

Load tswge

In order to use the ***tswge*** package you must access it for use. You can use either of the following methods:

(i) From the top menu under Packages, select *Load Package…*
  – you will be given a list of packages available to you. Select ***tswge***.
(ii) Enter the command
  `library(tswge)`
  at the beginning of any code that uses ***tswge***.

QR 1.2 Loading tswge

1.3.2 Base R

As you downloaded R onto your computer, you chose "base", which includes what we will refer to in this book as "Base R". Base R is a collection of functions and datasets, many of which relate to time series analysis. The functions and datasets are always available to you in an R session. The `plot` function, which we will discuss in the next section, is a readily accessible Base R function.

1.3.3 Plotting Time Series Data in R

The first thing a time series analyst should do when analyzing a time series is to ***plot the data***. In the following example, we discuss considerations involved in plotting time series data in R.

Example 1.1 Plotting a Time Series Dataset

A standard dot plot: One dataset in ***tswge*** is **`dfw.mon`,** which contains the average monthly temperatures in DFW from 1900 through 2020. This dataset is plotted in Figure 1.3. In this example, we will only use the data for the 10 years from January 2011 through December 2020, which for our purposes is contained in the ***tswge*** dataset **`dfw.2011`**. To access the data (once you have loaded the ***tswge*** package), issue the command

```
data(dfw.2011)
```

By typing the command

```
dfw.2011
```

the dataset's contents are listed as follows:

```
  [1] 42.8 49.5 61.3 70.8 72.8 86.8 91.4 93.4 80.0 68.2 57.9 47.6 50.4 52.5
 [15] 64.3 70.3 77.9 84.3 87.7 86.5 80.0 67.0 59.7 51.2 49.1 52.0 56.4 63.0
 [29] 72.3 82.6 84.5 87.1 82.4 68.2 53.5 43.1 45.3 47.0 55.1 66.3 74.4 82.4
 [43] 83.8 86.2 80.3 71.6 51.5 50.1 44.5 45.7 56.1 65.8 70.9 82.1 87.1 87.3
 [57] 82.7 71.2 58.7 53.7 47.0 55.2 61.2 68.1 72.5 84.0 87.4 85.8 81.5 74.1
 [71] 63.5 49.7 51.2 60.6 65.7 69.3 75.4 82.5 86.6 84.4 80.6 69.6 62.4 49.7
 [85] 45.8 51.1 63.3 61.6 79.0 85.7 88.8 85.2 78.1 66.2 52.5 48.4 45.8 50.2
 [99] 55.0 66.0 73.4 79.9 84.6 87.4 85.5 65.5 53.5 50.0 50.3 49.6 63.4 64.6
[113] 73.8 81.9 85.7 86.0 74.7 65.0 60.4 49.0
```

Using the plot command in Base R, that is,

```
plot(dfw.2011)
```

we obtain the plot in Figure 1.10(a). To put it lightly, this plot is a mess. The plot looks like a random pattern of points. Certainly, the seasonal cyclic behavior of the DFW temperature data is not visible. This is because the R function **`plot`** produces a dot plot (or scatterplot) by default. The scatterplot is useful for showing the relationship between two variables, like height and weight. However, given the data in **`dfw.2011`**, the variable on the horizontal axis is the vector index $t = 1, 2, \ldots, n$ where n is the length of the vector, in our case $n = 120$. So, Figure 1.10(a) is a scatterplot between the vector index and the data in the vector, and is clearly inadequate for our purposes.

Connecting the dots: Time series data are collected along a time axis, and it is important to know how the data evolve in time. To see this more clearly, we "connect the dots (points)". Figure 1.10(b) uses the R command

```
plot(dfw.2011,type= 'l')
```

where **`type=`** **`'l'`** tells the computer to connect the points with lines.

Note: The value for **`type`** is the letter **`l`** (for line), not the number 1.

The data in Figure 1.10(b) now show the seasonal pattern previously seen in Figure 1.4 but for a different time period.

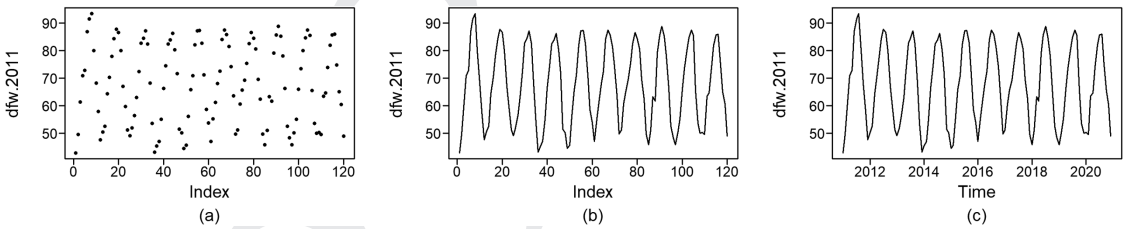


FIGURE 1.10 Monthly DFW temperature data: (a) as a dot plot, (b) the figure in (a) connecting the dots, and (c) the data in (b) with the horizontal axis correctly labeled in time.

The plot still has problems: Specifically, the horizontal axis in Figure 1.10(b) is indexed from time points 1 to 120 instead of years. In Section 1.3.4 we will discuss the ***ts*** object, which is a special object in R that is designed specifically to store time series data. Note that Figure 1.10(c) is a plot of a ***ts*** object, and we will discuss it in the next section.

1.3.4 The *ts* Object

We will introduce several sources and examples of time series data in this chapter. We begin our exploration with the **`dfw.2011`** dataset. The missing information in Figure 1.10(b) results from the fact that

the dataset **dfw.2011** does not contain any date information. Examination of the data, which are listed above, shows that the data reside in a vector which is by default indexed from 1 to 120.

1.3.4.1 Creating a ts Object

As mentioned, R makes it easy to work with time series data by facilitating its storage in an aptly named ***ts*** object.

Example 1.2 Creating a *ts* Object using the dfw.2011 Data
We recall from Example 1.1 that the data in **dfw.2011** are monthly average temperatures at DFW for the years 2011−2020. We will create a ***ts*** object that incorporates this date information, and call it **dfw.2011.ts**. The command we use to create the ***ts*** object is the following:

```
dfw.2011.ts = ts(dfw.2011,start=c(2011,1),frequency=12)
```

This command says to place the vector data, **dfw.2011**, into a ***ts*** object and to associate it with dates that start with January 2011 (**start=c(2011,1)**). It also stipulates that the data are monthly (**frequency=12**). Now by typing the command

```
dfw.2011.ts
```

the contents of this ***ts*** object are listed as follows:

```
      Jan   Feb   Mar   Apr   May   Jun   Jul   Aug   Sep   Oct   Nov   Dec
2011 42.8  49.5  61.3  70.8  72.8  86.8  91.4  93.4  80.0  68.2  57.9  47.6
2012 50.4  52.5  64.3  70.3  77.9  84.3  87.7  86.5  80.0  67.0  59.7  51.2
2013 49.1  52.0  56.4  63.0  72.3  82.6  84.5  87.1  82.4  68.2  53.5  43.1
2014 45.3  47.0  55.1  66.3  74.4  82.4  83.8  86.2  80.3  71.6  51.5  50.1
2015 44.5  45.7  56.1  65.8  70.9  82.1  87.1  87.3  82.7  71.2  58.7  53.7
2016 47.0  55.2  61.2  68.1  72.5  84.0  87.4  85.8  81.5  74.1  63.5  49.7
2017 51.2  60.6  65.7  69.3  75.4  82.5  86.6  84.4  80.6  69.6  62.4  49.7
2018 45.8  51.1  63.3  61.6  79.0  85.7  88.8  85.2  78.1  66.2  52.5  48.4
2019 45.8  50.2  55.0  66.0  73.4  79.9  84.6  87.4  85.5  65.5  53.5  50.0
2020 50.3  49.6  63.4  64.6  73.8  81.9  85.7  86.0  74.7  65.0  60.4  49.0
```

It is clear from the data listing that the time series starts in January 2011 and ends in December 2020.

Now that we have the data in the ***ts*** object **dfw.2011.ts**, we can issue the command

```
plot(dfw.2011.ts)
```

and the resulting plot is in Figure 1.10(c). This plot "connects the points" and provides the accurate date information.

If we issue the command

```
class(dfw.2011.ts)
```

the output

```
[1] "ts"
```

informs us that **dfw.2011.ts** is a ***ts*** object.[6] In contrast, by issuing the command

```
class(dfw.2011)
```

6 The ***ts*** objects need not have a **".ts"** extension. We have named the file **dfw.2011.ts** as a reminder that it is a ***ts*** file. However, it could have been named **dfw.2011** or **dfw.last10**.

we obtain the output

```
[1] "numeric"
```

indicating that **dfw.2011** is simply a numeric vector.

Example 1.3 The lynx dataset

Another classical dataset is the "lynx" data. This dataset contains the annual number of Canadian lynx that were trapped in the McKenzie River District in Canada between 1821 and 1934. It has been referenced in several seminal time series papers and software vignettes and is so popular that it is available in Base R as the *ts* object **lynx**. It may seem unusual that this dataset would be of such interest, so we will examine it further. We begin our examination by listing and plotting the data using the following commands:

```
data(lynx)
plot(lynx)
```

The **lynx** data are plotted in Figure 1.11.

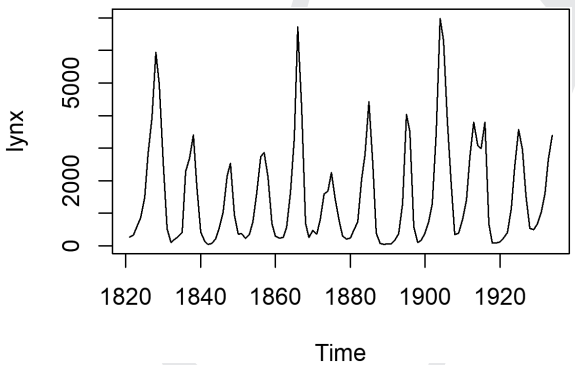


FIGURE 1.11 Annual number of lynx trapped in the McKenzie River district of Canada from 1821 to 1934.

From observing the data, we notice the following:

(a) The data appear to (surprisingly) have a 10-to-11-year cycle (period), and there is some evidence that the variable amplitude of the peaks has a pattern as well. This behavior is an example of cyclic behavior similar to the sunspot data in Figure 1.1(a). The somewhat puzzling 10-11-year cycle is the reason for the interest in this dataset.

(b) Examination of the plot shows that the dataset **lynx** must have some date information associated with it because the horizontal axis is in years, not the simple index 1 to 114. By issuing the command

```
class(lynx)
```

the output

```
[1] "ts"
```

indicates that the dataset **lynx** is already in *ts* form. Furthermore, we can see the information stored in this *ts* object by typing

```
lynx
```

after which we observe the following output:

```
Time Series:
Start = 1821
End = 1934
Frequency = 1
```

```
  [1]  269  321  585  871 1475 2821 3928 5943 4950 2577  523   98  184  279  409
 [16] 2285 2685 3409 1824  409  151   45   68  213  546 1033 2129 2536  957  361
 [31]  377  225  360  731 1638 2725 2871 2119  684  299  236  245  552 1623 3311
 [46] 6721 4254  687  255  473  358  784 1594 1676 2251 1426  756  299  201  229
 [61]  469  736 2042 2811 4431 2511  389   73   39   49   59  188  377 1292 4031
 [76] 3495  587  105  153  387  758 1307 3465 6991 6313 3794 1836  345  382  808
 [91] 1388 2713 3800 3091 2985 3790  674   81   80  108  229  399 1132 2432 3574
[106] 2935 1537  529  485  662 1000 1590 2657 3396
```

Along with the 114 annual counts of the number of lynx trapped, this ***ts*** object contains the start (1821) and end (1934) years of the time series. Recall that for the monthly data in the ***ts*** object **`dfw.2011.ts`**, the **`frequency`** attribute took on the value 12 indicating that there were 12 equally spaced observations per unit of time (year). For the **`lynx`** ***ts*** object, **`frequency=1`**, indicating that each observation represents a full year.

1.3.4.2 More About ts *Objects*

Returning our focus to the ***ts*** object itself, we discuss some features and uses for this powerful tool which is available for storing time series data.

(1) Extracting Vector Data from a ts Object

A ***ts*** object contains numeric data along with other attributes that assist in time series plotting. Suppose, however, that we have a ***ts*** object and want to extract the numeric vector containing (only) the data. This can be accomplished, using the ***ts*** object **`dfw.2011.ts`** as an example, with the following R command:

```
dfw.2011.num = as.numeric(dfw.2011.ts)
```

To verify that **`dfw.2011.num`** is a numeric vector, we issue the command

```
class(dfw.2011.num)
```

and obtain the output

```
[1] "numeric"
```

To actually view the data in the vector, we use the command

```
dfw.2011.num
```

and obtain the output

```
  [1] 42.8 49.5 61.3 70.8 72.8 86.8 91.4 93.4 80.0 68.2 57.9 47.6 50.4 52.5 64.3
 [16] 70.3 77.9 84.3 87.7 86.5 80.0 67.0 59.7 51.2 49.1 52.0 56.4 63.0 72.3 82.6
 [31] 84.5 87.1 82.4 68.2 53.5 43.1 45.3 47.0 55.1 66.3 74.4 82.4 83.8 86.2 80.3
 [46] 71.6 51.5 50.1 44.5 45.7 56.1 65.8 70.9 82.1 87.1 87.3 82.7 71.2 58.7 53.7
 [61] 47.0 55.2 61.2 68.1 72.5 84.0 87.4 85.8 81.5 74.1 63.5 49.7 51.2 60.6 65.7
 [76] 69.3 75.4 82.5 86.6 84.4 80.6 69.6 62.4 49.7 45.8 51.1 63.3 61.6 79.0 85.7
 [91] 88.8 85.2 78.1 66.2 52.5 48.4 45.8 50.2 55.0 66.0 73.4 79.9 84.6 87.4 85.5
[106] 65.5 53.5 50.0 50.3 49.6 63.4 64.6 73.8 81.9 85.7 86.0 74.7 65.0 60.4 49.0
```

Looking back at Example 1.1 we see that **`dfw.2011.num`** is identical to the original vector dataset **`dfw.2011`** .

(2) More on Creating a ts Object

Consider a time series defined in the vector **`x`** below:

```
x = c(10,20,30,40,50,60,70,80,90,100,110,120,130,140,150,160,170)
```

Assume this is monthly data (**frequency = 12**) which start in June, 2018 (**c(2018,6)**). To construct a ***ts*** object from this vector form of the time series, we invoke the following R commands (output follows):

```
xTSmonth = ts(x,start = c(2018,6),frequency = 12)
xTSmonth
```

```
      Jan  Feb  Mar  Apr  May  Jun  Jul  Aug  Sep  Oct  Nov  Dec
2018                          10   20   30   40   50   60   70
2019   80   90  100  110  120  130  140  150  160  170
```

Alternatively, if the same data were actually quarterly data (**frequency = 4**) and started in the 3rd quarter of 1986 (**c(1986,3)**), we could construct a ***ts*** object for the series stored in **x** using the code (output follows):

```
xTSquarter = ts(x,start = c(1986,3),frequency = 4)
xTSquarter
```

```
       Qtr1  Qtr2  Qtr3 Qtr4
1986                10   20
1987    30    40    50   60
1988    70    80    90  100
1989   110   120   130  140
1990   150   160   170
```

While the ***ts*** object has nice default formats for data recorded in months and quarters, we recognize that many time series will not fit this mold. Suppose that each observation is a day, and the unit of time is a week rather than a year. Furthermore, assume that the first observation was on the 4th day of the first week (**c(1,4)**). In this case the frequency would be 7 and the ***ts*** object would be constructed accordingly (output follows):

```
xTSweek = ts(x,start = c(1,4),frequency = 7)
xTSweek
Time Series:
Start = c(1, 4)
End = c(3, 6)
Frequency = 7
[1] 10  20 30  40 50  60 70  80  90 100  110  120 130 140  150 160  170
```

While the visual format is not as descriptive as it was for months and quarters, the information available is still useful. The input **start=c(1,4)** indicates that the data start on the 4th day of the first week. Because the series is 17 observations in length, it will take four observations to finish the first week, another seven to fill the second week, and then the remaining six observations will end on the 6th day of the 3rd week, consistent with the output above: **end=(3,6)**. While the daily data in this example are not presented in a convenient visual format, the period of seven in the data is stored in the **frequency** field.

Finally, we will often want to analyze only a subset of a ***ts*** object. This can be accomplished using the **window** function. With respect to the previous example, suppose we wanted to only work with data from the second week. We could accomplish this with the following code:

```
window(xTSweek,start = c(2,1),end = c(2,7))
```

```
Time Series:
Start = c(2, 1)
End = c(2, 7)
Frequency = 7
[1]  50  60  70 80  90  100  110
```

As another example, assume we want to subset the **AirPassengers** data to yield only the number of airline passengers during 1950:

```
window(AirPassengers,start = c(1950,1),end = c(1950,12))
```

```
      Jan Feb Mar  Apr  May Jun Jul  Aug  Sep  Oct  Nov Dec
1950  115 126 141  135  125 149 170  170  158  133  114 140
```

Key Points:

1. The ***ts*** object is a valuable tool for storing time series and their most important attributes in a single place.
2. The ***ts*** object can be useful in plotting and subsetting the series, and we will see that it is helpful in modeling as well.
3. There is a *multivariate* version of a ***ts*** object called an ***mts*** object for multivariate data structures (see footnote 9). Analysis of multivariate time series will be covered in Chapters 10 and 11.

1.3.5 The `plotts.wge` Function in *tswge*

The **plotts.wge** command in ***tswge*** extends the Base R plotting function, **plot**. The **plotts.wge** function can plot datasets consisting of numeric vectors as well as ***ts*** objects. This function makes it easy to produce useful plots that quickly provide the information needed by the analyst without specifying a lot of parameters for the graph. On the other hand, if it is important to obtain a higher quality plot for written reports, presentations, etc., **plotts.wge** allows you to "dress up" your plots as desired with colors, labels, line widths, and more.

Consider again the **lynx** dataset plotted in Figure 1.11. Note the asymmetric behavior of the lynx dataset, specifically that the peaks are much more variable than troughs from cycle to cycle. Most analysts who work with the **lynx** data analyze the logarithm of the **lynx** data instead of the **lynx** data themselves. An analyst may simply want to view this series quickly (Figure 1.12) to get an idea of any periodic behavior. The following code will take the logarithms and plot the log data.

```
data(lynx)
log.lynx=log(lynx)
# Note that log.lynx retains the ts file information contained in file lynx
plotts.wge(log.lynx)
```

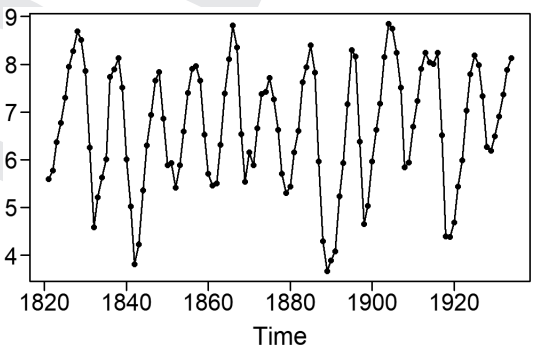


FIGURE 1.12 Basic **plotts.wge** plot of the log-lynx data.

From the plot we notice that the cyclic behavior is more symmetric in terms of variability of peaks versus troughs.

Key Point: The *tswge* function **`plotts.wge`** produces plots that:
- include data values as points in the graph if the realization length is less than or equal to 200
- do not include the points in the graph if the realization length is greater than 200

1.3.5.1 Modifying the Appearance of Plots Using the tswge *`plotts.wge`* Function

You may want to add customized labels, a title, color, or other plotting options that make the plot more informative and visually appealing. An example of code that would render more detail to the plot is given in Figure 1.13 below. A description of the function can be displayed in R with the command: **`?plotts.wge`**. The description is also included in Appendix 1A.

```
plotts.wge(log.lynx, style = 1, xlab = "Date", ylab = "Logarithm of Lynx Data",
title = "Natural Logarithm of Number of Lynx Trapped from 1821 and 1934. ",
text_size = 12)
```

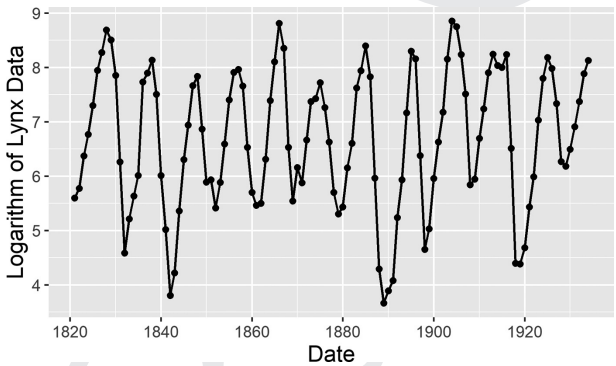


FIGURE 1.13 Example of a plot of the classic log-lynx data with style = 1.

Key Point: *tswge* functions will be described in the appendix of the chapter in which they are introduced.
- **`plotts.wge`** is described in Appendix 1A

1.3.6 Loading Time Series Data into R

To this point, we have worked with data that were "native to R"; in practice, of course, we will usually want to load external data into R. The comma separated value (**`.csv`**) format is the format of many time series datasets we will discuss in Section 1.3.7. Also, external data may be stored in text (**`.txt`**) files and Excel (**`.xlsx`**) files. Sections 1.3.6.1 and 1.3.6.2 provide useful information for reading **`.csv`** and **`.txt`** files, respectively.

1.3.6.1 The .csv file

In a **.csv** file, each observation is separated by a comma. While *.csv* stands for "comma-separated values", other delimiters such as semicolons, are allowed. On the book's GitHub site, a file named **AirPassengers.csv** is a **.csv** file containing the air passenger data discussed earlier. Download this file to your computer. It is common practice to open this type of file using a spreadsheet software package like Excel, although opening it in a common text editing application will show the separating commas explicitly. Figure 1.14 displays the first 10 values of this dataset in both Excel (left) and raw text format. Note that the columns (values) are separated by commas and the rows (observations) are separated by a new line.

	A	B
1	Date	NumPassengers
2	Jan-49	112
3	Feb-49	118
4	Mar-49	132
5	Apr-49	129
6	May-49	121
7	Jun-49	135
8	Jul-49	148
9	Aug-49	148
10	Sep-49	136
11	Oct-49	119

(a) Microsoft Excel

```
Date,NumPassengers
Jan-49,112
Feb-49,118
Mar-49,132
Apr-49,129
May-49,121
Jun-49,135
Jul-49,148
Aug-49,148
Sep-49,136
Oct-49,119
```

(b) Text Editor

FIGURE 1.14 Screen shot of first 10 rows of the AirPassengers data stored in comma separated format in (a) Excel and (b) a standard text editor.

(1) The read.csv Function

R makes it easy to read in **.csv** files using the **read.csv** function. We will use the **read.csv** function to read the **AirPassengers.csv** file that that is stored in some location on your computer. Note that the first line of the dataset is the header (variable name) of each column. To tell R to treat this first row as the header we specify **header = TRUE**. For example, to read the **AirPassengers.csv** file you could use the command

```
AirPassengersData=read.csv("Yourfile location//AirPassengers.csv",header=TRUE)
```

Note: To reiterate, the file path will be specific to your computer.
For example, in Windows the path might be something like
"c:\\Documents and Settings\\My Documents\\My Data Files"

A simple way to use the **read.csv** function is to use it along with the **file.choose** function in order to select the file without having to identify the path in the user's file system. The complete function call is

```
AirPassengersData=read.csv(file.choose(),header=TRUE)
```

This command will bring up a file selection screen in which one can navigate to the file to be loaded. At this point, the user selects **AirPassengers.csv** in the subdirectory in which it resides, and the data will be loaded into R. It is always a good idea to check to make sure the data were read in as expected. The following line of code will print the first 10 lines of the dataset.[7]

7 The R function "**head**" prints the first *n* lines of a dataset.

```
head(AirPassengersData,n=10)
```

```
    Date     NumPassengers
1   Jan-49   112
2   Feb-49   118
3   Mar-49   132
4   Apr-49   129
5   May-49   121
6   Jun-49   135
7   Jul-49   148
8   Aug-49   148
9   Sep-49   136
10  Oct-49   119
```

QR 1.3 Import Data with read.csv

1.3.6.2 The .txt file

As mentioned, sometimes data are saved in standard text (**.txt**) files. Below are screenshots of two **.txt** files: **sample1.txt** and **sample2.txt.**

```
sample1.txt     sample2.txt
34              34 42 55
42              23 36 33
55
23
36
33
```

(1) The Scan Function

To follow this example, create the two files in a text editor (not much typing is involved) and save them in a location of your choice. The scan function can be used to read each of these datasets into the same numerical R vector.

```
s1=scan("Your file location//Sample1.txt")
s2=scan("Your file location//Sample2.txt")
```

The numeric vectors **s1** and **s2** are both equal to

```
[1]  34  42  55  23  36  33
```

That is, the scan function reads the data row-wise into a vector and there is no header option. You can also use the **file.choose** function with **scan**.

1.3.6.3 Other File Formats

In addition to the common **.csv** and **.txt** file formats, one may also encounter any number of other formats such as Excel formats **(.xlsx**), **JSON,** or **XML**, just to name a few. Because R is a well-established language, it is likely that an input function exists which accommodates nearly any format. Also, because R is open source, if a new format comes into vogue in the future, it will usually not take long for someone to write a function that is compatible with the new format. While R can read **.xlsx** files, the instructions given in this chapter involve reading **.csv** files. An Excel file is easily converted into a **.csv** file.

1.3.7 Accessing Time Series Data

So far, we have discussed how to read and plot time series data. But who is collecting time series data and where can it be found? We have seen that Base R and individual R packages themselves (including ***tswge***) are a great source of time series data. For example, the **`AirPassengers`** and **`lynx`** datasets are ***ts*** objects in Base R which can be made available for use by simply issuing the commands

```
data(AirPassengers)
data(lynx)
```

Oh, if it were always this easy! The datasets available in R are just the tip of the iceberg! In this section, we will discuss the internet as a source of time series data and provide instruction on how to access data from internet websites.

1.3.7.1 Accessing Data from the Internet

There are an incredible number of websites that contain time series data. (Sometimes access requires a fee, but that is not the case for the data in the following examples.) Some of the websites are privately owned and some are government sources of data. In this section we will briefly discuss the following websites and illustrate the procedure for accessing data from them:

(1) FRED: The Federal Reserve Economic Database https://fred.stlouisfed.org
(2) Silso: Sunspot Index and Long-Term Solar Observations www.sidc.oma.be/silso/
(3) Yahoo! Finance https://finance.yahoo.com/
(4) National Weather Service www.weather.gov
(5) New York City Taxi & Limousine website https://www1.nyc.gov/site/tlc/about/about-tlc.page

Two additional websites containing many time series datasets are https://census.gov and https://epa.gov

(1) FRED: The Federal Reserve Economic Database
At the intersection of government and research lies the Federal Reserve Economic Database, or *FRED*. Created in 1991 by the Research Department at the Federal Reserve Bank of St. Louis, *FRED* is a repository of over *a hundred thousand* economic time series from around the world! The website is https://fred.stlouisfed.org . The homepage is shown in Figure 1.15.

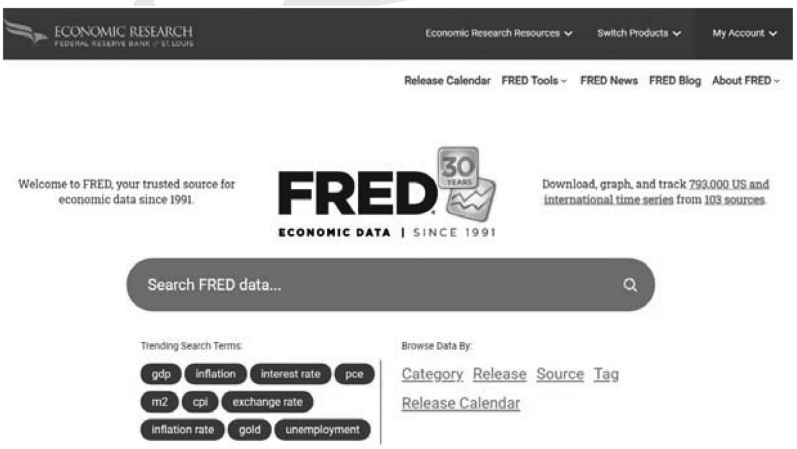


FIGURE 1.15 FRED Homepage.

The various datasets are found by selecting *Category* from the above menu, after which a screen appears which shows various options. These include *Money, Finance, & Banking*, *Production & Business*, and *Prices* among others. We will examine two of the datasets (a) West Texas crude oil prices and (b) median days houses stay on the market before being sold.

(a) The West Texas Intermediate (WTI) Crude Oil Data

The webpage in Figure 1.16(a) has a plot of the same data as Figure 1.8(e). For comparison, Figure 1.8(e) is reproduced in Figure 1.16(b). This dataset is contained in the *Prices* category under the subcategory *Commodities*. This is accessed on the FRED website by navigating to the WTI data using the following steps:

- *Categories* → *Prices* → *Commodities*
- Scroll down and choose: *Global Price of WTI Crude*[8] and select *monthly*
- Click on *Global Price of WTI Crude* after which the graph below will appear
- To access the data, select *Download* and then *CSV(data)*
- Save the downloaded file, **`POILWTIUSDM.csv`,** to a subdirectory of choice
- In R, read the **`.csv`** file using
  `WTI=read.csv(file.choose(),header=TRUE)`
- A ***ts*** file can be created using the command
  `wtcrude2020=ts(WTI$POILWTIUSDM,start=c(1990,1),frequency=12)`

QR 1.4 Download Data from FRED

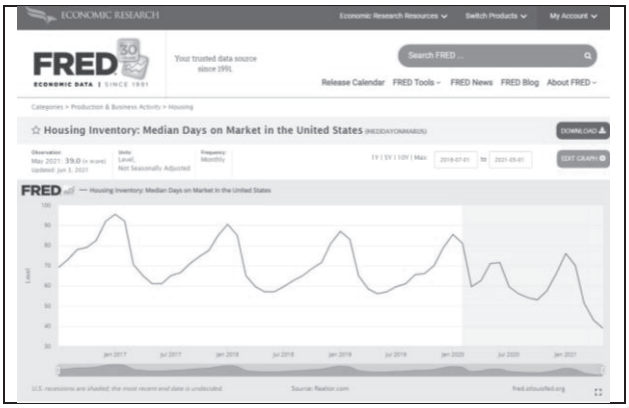
(a) (b)

FIGURE 1.16 (a) Screenshot of WTI Crude data from January 1990 through April 2021 and (b) R-based plot of the ***ts*** object wtcrude2020 that extends through December 2020.

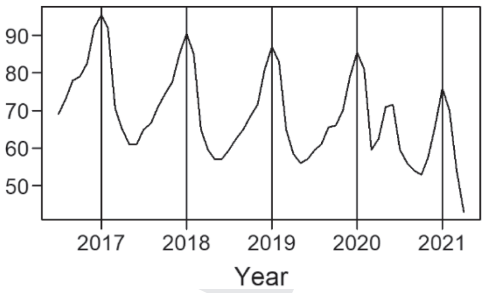
(b) Median Days on Market

Another dataset on the FRED website is a monthly tally of the median number of days a house stays on the market before being sold. The webpage is shown in Figure 1.17(a).

8 One category under *Commodities* is Crude Oil Prices: West Texas Intermediate (WTI) – Cushing Oklahoma. This is NOT the crude oil data set we are using. You should scroll down to *Global Price of WTI Crude*.



(a)



(b)

FIGURE 1.17 (a) Screenshot of the FRED website for median number of days to sell a house from July 2016 through April 2021 and (b) R plot of the data in (a).

This dataset can be accessed using the following steps:

- *Categories* → *Production & Business* → *Housing*
- Scroll down and choose: *Housing Inventory: Median Days on Market in the United States* and select *Monthly, Not Seasonally Adjusted.* Figure 1.17(a) will appear.
- To access the data select *Download* and then *CSV(data)*
- Save the downloaded file **`MEDDAYSONMARUS.csv`** to a subdirectory of choice
- In R, read the **`.csv`** file using
  `MedDays=read.csv(file.choose(),header=TRUE)`
- A ***ts*** file can be created using the command
  `MedDays.ts=ts(MedDays$MEDDAYONMARUS,start=c(2016,7),frequency=12)`

To obtain the plot of the data in Figure 1.17(b) use the command

`plotts.wge(MedDays.ts)`

Figure 1.17(b) is much easier to read than the screenshot of the website in Figure 1.17(a). Although we do not have a lengthy record, examination of this plot shows that a seasonal pattern occurs from 2017 through 2019 during which time houses typically sold faster (fewer days on the market) in May, June, and July while during December, January, and February, the number of days on the market is higher. Notice that the seasonal pattern abruptly changed during 2020 with May and June having higher than usual days on the market. One would suspect that this was due to the COVID outbreak declared a pandemic in March 2020. Notice also that things "turned around" in July through November with fewer than the normal number of days on the market. Also, during March and April of 2021, houses were selling unusually quickly.

(2) Silso: Sunspot Index and Long-Term Solar Observations
This website provides the new Sunspot2.0 numbers that were mentioned in Section 1.2.1. The link is www.sidc.oma.be/silso/ . The steps for retrieving the annual data are given below.

(a) Annual Data (Available for 1700 through the Present)

- *Click: Data* → *Sunspot Number* → *Total Sunspot Number*
- Select *Yearly mean total sunspot number [1700 - now]* and choose *CSV*
- The file **`SN_y_tot_v2.0.csv`** will be downloaded. Save this file to a subdirectory of choice

- *Note:* This **.csv** file uses a semicolon delimiter instead of a comma(default). To read the **.csv** file in R use the command
  ss=read.csv(file.choose(),';',header=FALSE)
- The data frame[9] **ss** has four columns, the second of which contains the sunspot data. The second variable has the default name **V2**. A ***ts*** file can be created using the command
  sunspot2.0=ts(ss$V2,start=1700,frequency=1)

QR 1.5 Download the Sunspot Data

A plot of the **sunspot2.0** is given in Figure 1.1 and is repeated in Figure 1.18(a).

(b) Monthly Data (Available for 1749 through the Present)

- *Click: Data → Sunspot Number → Total Sunspot Number*
- Select *Monthly mean total sunspot* number [1/1749–now] and choose CSV
- The file **SN_m_tot_v2.0.csv** will be downloaded. Save this file to a subdirectory of choice
- *Note:* As in the case of yearly means, this **.csv** file uses a semicolon delimiter instead of a comma. To read the file **SN_m_tot_v2.0.csv** file in R use the command **ss.month=read.csv(file.choose(),';',header=FALSE)**
- The data frame **ss.month** has seven columns. The fourth column contains the sunspot numbers and has the default name **V4**. A ***ts*** file can be created using the command
  sunspot2.0.month=ts(ss.month$V4,start=c(1749,1),frequency=12)

A plot of the **sunspot2.0.month** is given in Figure 1.18(b).

Notes: In most (but not all) cases the datasets we will analyze in this book extend through 2020. For the sunspot data, in order to replicate the datasets plotted here, remove any data recorded on or after January 1, 2021.

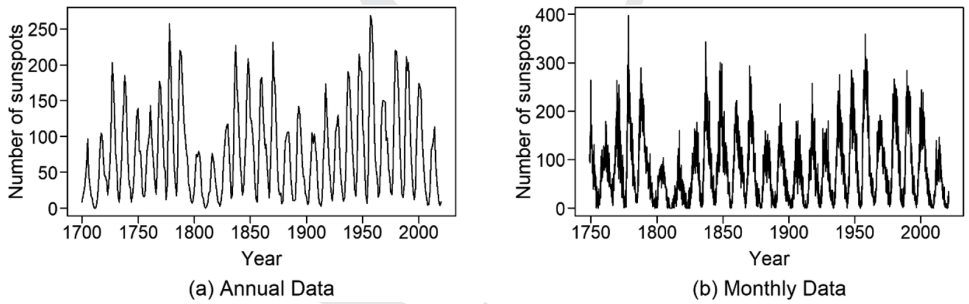


FIGURE 1.18 (a) Plot of annual sunspot2.0 numbers from 1700 through 2020 and (b) plot of monthly sunspot2.0 numbers from 1749 through 2020.

(3) Yahoo! Finance

Stock market and other financial data are available on numerous sites on the internet. For our purposes we will use *Yahoo! Finance* which can be found at https://finance.yahoo.com/ . We will discuss stock prices for *Tesla*.

9 A "data frame" is a data structure in R that is similar to a matrix but can have various object classes rather than only numbers. It is one of, if not the, most common data structures in R. For more information on data frames simply type **?data.frame** into the R console.

(a) Tesla

Figure 1.19 below is a plot of the Tesla stock prices from January 1, 2020 through April 30, 2021. A dramatically increasing trending behavior until January 2021 is observed at which time the stock stayed fairly constant until February 2021 at which time it dropped. At the time of the writing of this section, the stock seems to be making a slight recovery.[10]

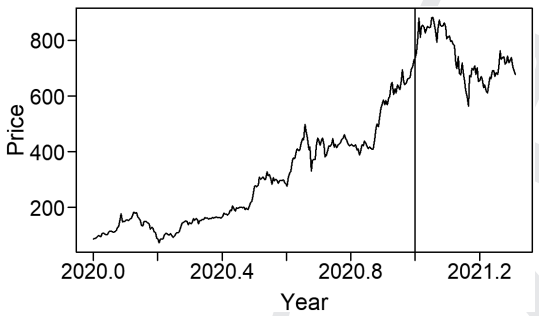


FIGURE 1.19 Tesla stock prices from January 1, 2020 through April 30, 2021.[a]
[a] For example, the date 2020.4 is 40% through year 2020 or late May.

In order to access the Tesla data and prepare it for plotting, use the following steps:

- In the search box at the top of the Yahoo! Finance homepage enter *Tesla*; the ticker symbol will appear as *TSLA*.
- Select *TSLA* and on the resulting screen select *Historical Data* and *Daily*
- Fix dates from 01/01/2020 to 04/30/2021
- Select *Apply* and then *Download*
- The file **`TSLA.csv`** will be downloaded. Save this file to a subdirectory of choice.
- To read the file **`TSLA.csv`** in R, use the command
  `tesla=read.csv(file.choose(),header=TRUE)`
- The data frame **`tesla`** has seven columns. The sixth column contains the adjusted closing price for that particular day with the variable name **`Adj.Close`**
- A ***ts*** file can be created using the command
  `tesla=ts(tesla$Adj.Close,start=c(2020,1),frequency=254)`
  (There were 254 business days in 2020)
- The command **`plotts.wge(tesla.ts)`** produces Figure 1.17 (without the more descriptive labels and vertical line at January 2021 shown there.)

(4) National Weather Service

This is a website published by the National Weather Service that provides an abundance of information about local forecasts and historical data at www.weather.gov. The monthly average temperature in Dallas Ft. Worth (DFW) from September 1898 through the current month can be obtained from the link www.weather.gov/fwd/dmotemp . In ***tswge*** there are two files obtained from this link:

(a) **`dfw.mon`**: DFW monthly average temperatures from January 1900 through December 2020. *Note:* These data are plotted in Figure 1.3.
(b) **`dfw.yr`**: DFW average annual temperatures from 1900 through 2020.

These ***ts*** files are plotted in Figure 1.20.

10 Recall the caution to not assume that observed trends will necessarily continue. The first author went against his own advice and bought Tesla stock in early January 2021.

Note: The above link for the DFW temperatures does not provide for the option to download a **`.csv`** file. In Section 1.4.1.2 we will discuss procedures for obtaining the temperature data from the link.

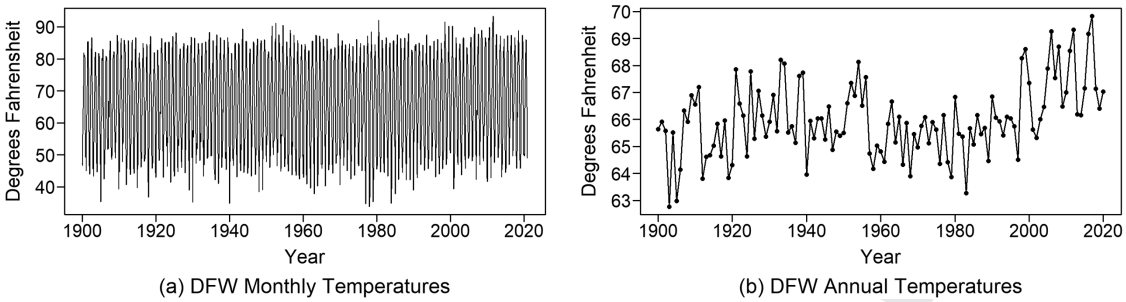


FIGURE 1.20 (a) DFW monthly temperatures from 1900 through 2020 and (b) DFW annual average temperatures for the same time period.

Notes: Figure 1.20(a) is the plot in Figure 1.3. Figure 1.20(b) has a wandering behavior with an upward trend beginning in the 1980s that suggests a warming trend which is the topic of much discussion. However, in DFW the years 2018–2020 have been cooler.

(5) New York City Taxi & Limousine website

Suppose that an analyst wants to analyze the effect of COVID-19 on taxicab trips in New York City. Beginning in 2009, the city of New York mandated that all taxis report the distance, length, cost, pickup and drop-off location, and other trip information for each taxicab ride in the city. The data can be obtained from the New York City Taxi & Limousine website www1.nyc.gov/site/tlc/about/about-tlc.page shown in Figure 1.21.

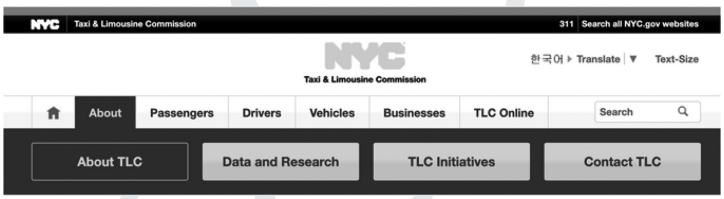


FIGURE 1.21 New York Taxi & Limousine Commission homepage.

The file **`yellowcab.precleaned.ts`** is a ***ts*** file in ***tswge*** containing the number of trips per month for Yellow Cabs (a taxicab company), and these data are plotted in Figure 1.22. COVID-19 was declared a "pandemic" in March 2020, and it of interest to understand the degree to which this affected demand for taxi service in New York City. Figure 1.22 dramatically shows the devastating impact.

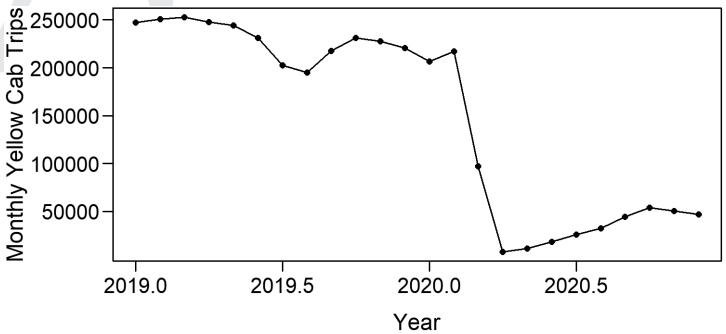


FIGURE 1.22 Number of Yellow Taxicab trips per month from January 2019 to February 2021.

Key Points

1. The file **`yellowcab.precleaned`** is *not* directly attainable by downloading a .csv or .txt file from the NYC Taxicab website.
2. A considerable amount of data manipulation was required to "pre-clean" the data.
3. Section 1.4.1.3 discusses techniques for pre-cleaning data and specifically for producing the ***ts*** file **`yellowcab.precleaned`**.

1.3.7.2 Business / Proprietary Data: Ozona Bar and Grill

Time series data may also be obtained directly from a business or organization. For example, one of the most popular restaurants in Dallas, Texas (and frequented quite often by the authors of this text) is a place near the campus of Southern Methodist University named *Ozona Bar and Grill*. While Ozona has a full menu full of Dallasite favorites, it is famous for its chicken fried steak! We asked the Director of Operations for relevant historical data, and he graciously provided us with a time series of the daily number of chicken fried steaks that were sold in June and July of 2019.[11] The data are provided in the **`ozona`** data frame in the ***tswge*** package and are displayed below (Figure 1.23). Stop in next time you are in the neighborhood! Thank you, Director of Operations, Cory Wauson!

```
data(ozona)
ozona.ts=ts(ozona$CFS_Sold)
plot(ozona.ts,type='o',xlab='Week')
```

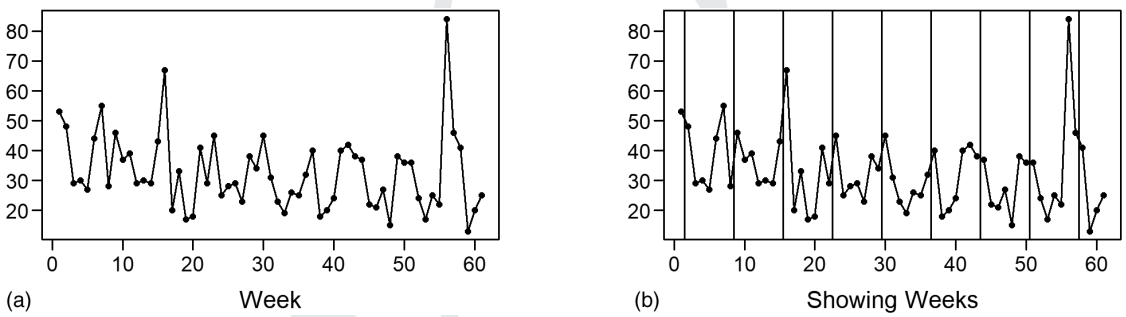


FIGURE 1.23 (a) Number of chicken fried steaks sold at Ozona Bar and Grill in June and July, 2019 and (b) plot (a) with vertical lines separating weeks

In Figure 1.23(a) a fairly random-looking pattern of sales is observed, with a slight decreasing trend except for a large spike in late July. Figure 1.23(b) separates weeks with vertical lines between Saturday and Sunday for each week. In this plot we see a definite "seasonal" pattern in which weekends, particularly Friday and Sunday, tend to have the most sales.

Key Point: When plotting data from a private source such as Ozona, be sure that permission to use the data has been granted.

11 As all good statisticians should do, we sampled the (delicious) chicken fried steak data!

1.4 DEALING WITH MESSY DATA

Despite the wide variety of types and sources of time series data the analyst may encounter, one common denominator is that such data are often not available as data files in R or convenient `.csv` files on the internet that are immediately suitable for visualization and analysis. In many cases the data must be manipulated into a suitable form.

After obtaining the data, the next step is to perform any necessary cleaning, wrangling,[12] and/or handling of missing values in order to obtain the data in the form needed for analysis. Nearly all statisticians and data analysts discover that after they leave the classroom and enter the "real world of data", a large part of analyzing data is *preparing the data for analysis.* This is an important and complex topic that we will only briefly introduce here using a couple of examples.

1.4.1 Preparing Time Series Data for Analysis: Cleaning, Wrangling, and Imputation

In this section, we give two examples in which the data need to be cleaned or otherwise manipulated before analysis can take place. These examples address three types of "problem data".

- Data with missing values.
- Data on websites that do not provide a download option.
- Data that can be obtained from a larger data file but must be assembled and organized before analysis takes place.

1.4.1.1 Missing Data

It is important to note that the data analytic procedures discussed in this book assume that data are obtained at equally spaced time intervals and that there are no missing data values. (For information about analyzing irregularly spaced time series data see Jones, 2016; Wang, Woodward and Gray, 2009). The Bitcoin dataset discussed below has three missing values. Before analysis can proceed, these values need to be "imputed". That is, they need to be approximated and then put into the dataset in the place of the missing values. There are many ways to impute missing data. Suppose x_t is missing in a dataset. Two simple approaches are:

- Set $x_t = x_{t-1}$ where x_{t-1} is the previous data value (which is known). This approach is commonly known as Last Observation Carried Forward (LOCF).
- Linear interpolation: that is, conceptually draw a straight line connecting x_{t-1} and x_{t+1} which are the known values on each side of the missing value. Then assign x_t to be the value on the line at time t. See Figure 1.24(a).

In the Bitcoin data, there is one instance in which two adjacent values are missing. In this case, linear interpolation is illustrated in Figure 1.24(b) and the procedure is analogous. Connect the two known values x_{t-1} and x_{t+2} with a straight line and find the values on that line at t and $t+1$. This procedure is the basis for the linear interpolation formulas in the code below.

12 Wrangling is the processing of transforming and mapping data from one form to another for purposes of preparing it for analysis.

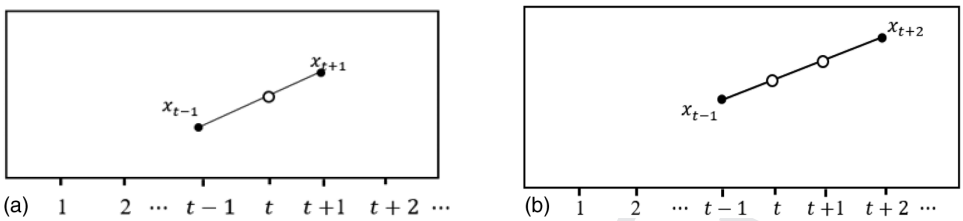


FIGURE 1.24 Illustration of linear interpolation. (a) with one missing value (x_t) and (b) with two adjacent missing values (x_t and x_{t+1}).

Key Points

1. The techniques described in this book for analyzing time series data require that data are equally spaced and that there are no missing values.
2. Methods do exist for analyzing data with unequally spaced and missing data.

(a) Bitcoin Prices

Bitcoin prices can be downloaded from the *Yahoo! Finance* website at https://finance.yahoo.com/. The Bitcoin prices are missing in 2020 for October 9,12, and 13. We will use linear interpolation to impute the missing values. Figure 1.25(a) shows Bitcoin prices for the year from May 1, 2020 through April 30, 2021. The horizontal axis is Index. The variable Index begins at 1 for May 1, 2020 and goes through 365 (a full year) on April 30, 2021. Table 1.1 pairs some key dates with the corresponding Index.

TABLE 1.1 Date/Index Pairs for Bitcoin data

DATE	*INDEX*
May 1, 2020	1
October 1, 2020	154
October 9, 2020	162
October 12, 2020	165
October 13, 2020	166
October 22, 2020	174
April 30,2021	365

Dealing with Messy Data

Bitcoin has continued to rise as of April 30, 2021 with some indication of "slowing down". As mentioned, the Bitcoin prices for October 9, 12, 13 in 2020 (Index values 162, 165, 166) are missing. In order to better see the missing data area, we will focus on the snippet of time from October 1 through October 22, 2020. According to Table 1.1 these dates have Index values 154 and 174, respectively. Figure 1.25(b) shows the snippet range within the two vertical lines. If you look very closely at the data between the lines you can see tiny gaps. Figure 1.25(c) shows the data within the focused range and we can see that there are three missing values at Indices 162, 165, and 166. Finally, Figure 1.25(d) shows the snippet data with missing values imputed using linear interpolation.

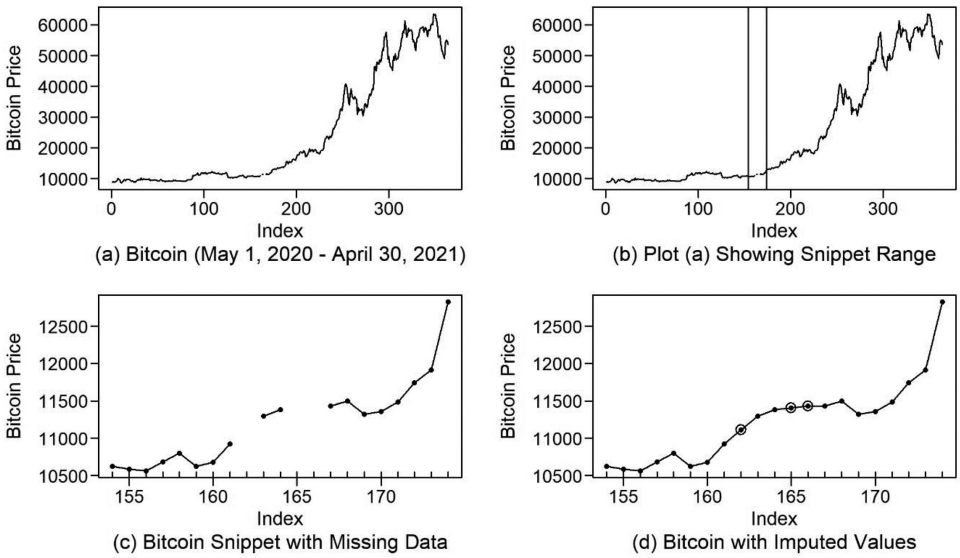


FIGURE 1.25 (a) Bitcoin data from May 1, 2020 through April 30, 2021, (b) Plot in (a) showing the snippet of time from October 1 through October 22, 2020 with the vertical strip, (c) Snippet of Bitcoin data from October 1 through October 22, 2020 showing missing values on October 9, 12, and 13 and (d) Figure (c) with imputed values.

You can use the following steps to download the Bitcoin data and replace missing values using linear interpolation.

- In the search box enter *Bitcoin*; the ticker symbol will appear as *BTC-USD*.
- Select *BTC-USD* and on the resulting screen select *Historical Data* and *Daily*.
- Fix dates from 05/01/2020 to 04/30/2021 to collect "12 months" or "a year" of daily data.
- Select *Apply* and then *Download*
- The file **`BTC-USD.csv`** will be downloaded; Adjusted Close (variable **`Adj.Close`**) is the variable of interest

Note: **`Adj.Close`** values for October 9, 12, and 13 in 2020 are missing. It will be necessary to impute values for these dates before analysis can continue.

- Save this file to a subdirectory of choice.
- To read the file **`BTC-USD.csv`** in R use the command
  `BTC=read.csv(file.choose(),header=TRUE)`
- The data frame **`BTC`** has seven columns. The sixth column contains the adjusted closing price for that day with the variable name **`Adj.Close`**
- The data in **`BTC$Adj.Close`** are character values, so enter the command
  `Bitcoin=as.numeric(BTC$Adj.Close)`
- A warning will be issued that there are missing values, and by examining the file **`Bitcoin`** it follows that the missing values are at time points $t = 162,165$, and 166.
- The snippet data are the subset **`Bitcoin[154:174]`**
- Impute missing values using linear interpolation

```
# Linear interpolation with one missing value
Bitcoin[162]=Bitcoin[161]+(Bitcoin[163]-Bitcoin[161])/2
# Linear interpolation with two adjacent missing values
```

```
Bitcoin[165]=Bitcoin[164]+(Bitcoin[167]-Bitcoin[164])/2
Bitcoin[166]=Bitcoin[164]+2*(Bitcoin[167]-Bitcoin[164])/2
```

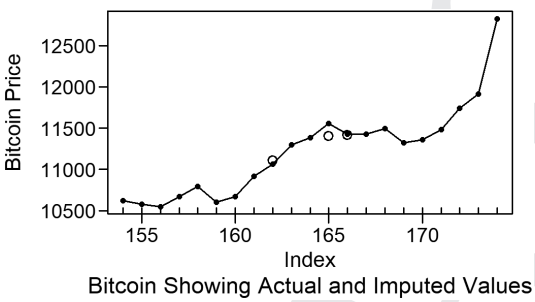
Key Points

1. There are many imputation techniques (we used linear interpolation).
2. While the *mean* is an often-used imputed value for missing data in a random sample, the correlation structure of time series data is such that the mean is rarely a wise imputation choice in a time series setting.
3. It is very important to consider why the value is missing in the first place.
4. For a deeper discussion of these topics please check out the video using the QR code.

1.4.1.2 Downloading When no .csv Download Option Is Available

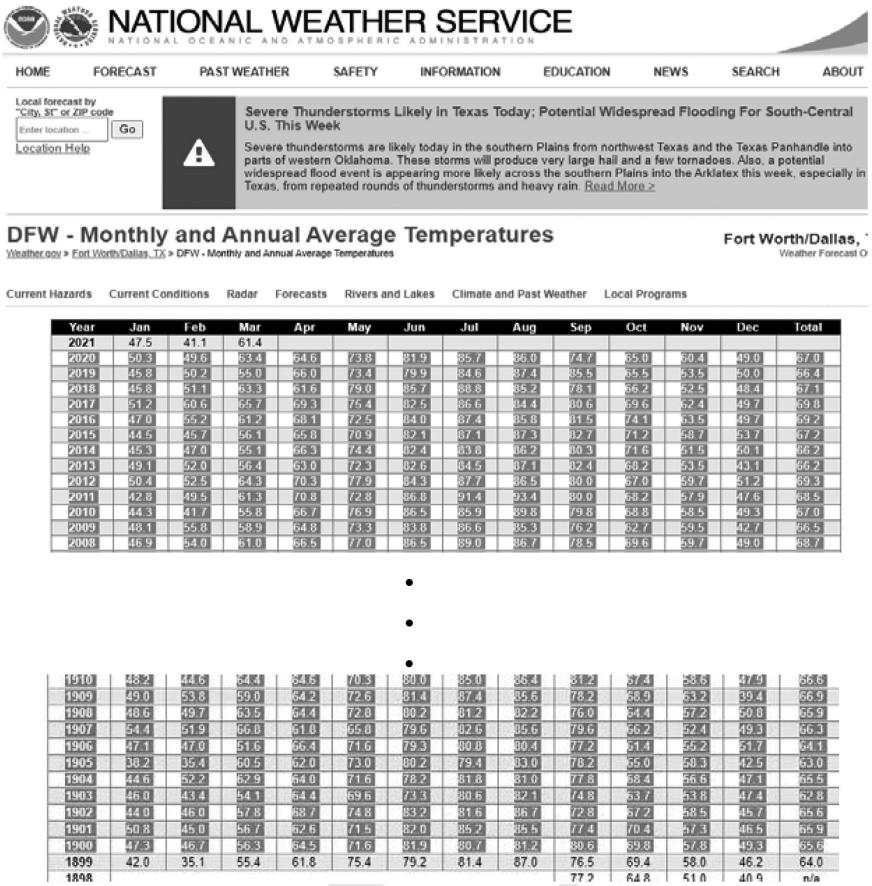
The previous examples have focused on websites for which the data can be downloaded onto a .csv (or other formatted) file. However, the DFW temperature data obtained from the link weather.gov link www.weather.gov/fwd/dmotemp does not have a download option. Figure 1.27 shows the monthly DFW temperatures from September 1898 through the current month (April 2021 as of the writing of this book). In this example, we will utilize the data from January 1900 through December 2020. The following steps allow the user to access these monthly averages for analysis in R even though there is no download option.

- Select the data (selection shown with shading in Figure 1.20) and use Ctrl-C for copying.



QR 1.6 Bitcoin Imputation

FIGURE 1.26 Bitcoin data from October 1 through October 22, 2020 as solid dots connected by solid lines and the imputed values for October 9, 2, and 13 as open circles.



NATIONAL WEATHER SERVICE
NATIONAL OCEANIC AND ATMOSPHERIC ADMINISTRATION

HOME FORECAST PAST WEATHER SAFETY INFORMATION EDUCATION NEWS SEARCH ABOUT

Local forecast by "City, St" or ZIP code
Enter location ... Go
Location Help

Severe Thunderstorms Likely in Texas Today; Potential Widespread Flooding For South-Central U.S. This Week

Severe thunderstorms are likely today in the southern Plains from northwest Texas and the Texas Panhandle into parts of western Oklahoma. These storms will produce very large hail and a few tornadoes. Also, a potential widespread flood event is appearing more likely across the southern Plains into the Arklatex this week, especially in Texas, from repeated rounds of thunderstorms and heavy rain. Read More >

DFW - Monthly and Annual Average Temperatures

Weather.gov » Fort Worth/Dallas, TX » DFW - Monthly and Annual Average Temperatures

Fort Worth/Dallas,
Weather Forecast O

Current Hazards Current Conditions Radar Forecasts Rivers and Lakes Climate and Past Weather Local Programs

Year	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec	Total
2021	47.5	41.1	61.4										
2020	50.3	49.6	53.4	64.6	73.8	81.9	85.7	86.0	74.7	65.0	50.4	49.0	67.0
2019	45.8	50.2	55.0	66.0	73.4	79.9	84.6	87.4	85.5	65.5	53.5	50.0	66.4
2018	45.8	51.1	63.3	61.6	79.0	85.7	88.8	85.2	78.1	66.2	52.5	48.4	67.1
2017	51.2	60.6	65.7	69.3	75.4	82.5	86.6	84.4	80.6	69.6	62.4	49.7	69.8
2016	47.0	55.2	61.2	68.1	72.5	84.0	87.4	85.8	81.5	74.1	63.5	49.7	69.2
2015	44.5	45.7	56.1	65.8	70.9	82.1	87.1	87.3	82.7	71.2	58.7	53.7	67.2
2014	45.3	47.0	55.1	66.3	74.4	82.4	83.6	86.2	80.3	71.6	61.5	50.1	66.2
2013	49.1	52.0	56.4	63.0	72.3	82.6	84.5	87.1	82.4	68.2	53.5	43.1	66.2
2012	50.4	52.5	64.3	70.3	77.9	84.3	87.7	86.5	80.0	67.0	59.7	51.2	69.3
2011	42.8	49.5	61.3	70.8	72.8	86.8	91.4	93.4	80.0	68.2	57.9	47.6	68.5
2010	44.3	41.7	55.8	66.7	76.9	86.5	85.9	89.8	79.8	68.8	58.5	49.3	67.0
2009	48.1	55.8	58.9	64.8	73.3	83.8	86.6	85.3	76.2	62.7	59.5	42.7	66.5
2008	46.9	54.0	61.0	66.5	77.0	86.5	89.0	86.7	78.5	69.6	59.7	49.0	68.7

•
•
•

1910	48.2	44.6	54.4	64.6	70.3	80.0	85.0	86.4	81.2	67.4	58.6	47.9	66.6
1909	49.0	53.8	59.0	64.2	72.6	81.4	87.4	85.6	78.2	68.9	53.2	39.4	66.9
1908	48.6	49.7	63.5	64.4	72.8	80.2	81.2	82.2	76.0	64.4	57.2	50.8	65.9
1907	54.4	51.9	66.8	61.8	65.8	79.6	82.6	85.6	79.6	66.2	52.4	49.3	66.3
1906	47.1	47.0	51.6	66.4	71.6	79.3	80.0	80.4	77.2	61.4	55.2	51.7	64.1
1905	38.2	35.4	50.5	62.0	73.0	80.2	79.4	83.0	78.2	65.0	58.3	42.5	63.0
1904	44.6	52.2	62.9	64.0	71.6	78.2	81.8	81.0	77.8	68.4	56.6	47.1	65.5
1903	46.0	43.4	54.1	64.4	69.6	73.3	80.6	82.1	74.8	63.7	53.8	47.4	62.8
1902	44.0	46.0	57.8	68.7	74.8	83.2	81.6	86.7	72.8	67.2	58.5	45.7	65.6
1901	50.8	45.0	56.7	62.6	71.5	82.0	85.2	85.5	77.4	70.4	57.3	46.5	65.9
1900	47.3	46.7	56.3	64.5	71.6	81.9	80.7	81.2	80.6	59.8	57.8	49.3	65.6
1899	42.0	35.1	55.4	61.8	75.4	79.2	81.4	87.0	76.5	69.4	58.0	46.2	64.0
1898									77.2	64.8	51.0	40.9	n/a

FIGURE 1.27 National Weather Service monthly data for Dallas Ft. Worth.

- Paste (Ctrl-V) the data into a blank Excel file.
- Delete the last column containing annual averages.
- Select all of the columns (A–M).
- Click *Data* → *Sort*.
- The checkbox "My data has headers" should not be checked.
- For "Sort by" choose *Column A*.
- For "Order" choose *Smallest to Largest*.
  The data should now be sorted from earliest dates to the most recent.
- Delete the first column containing years.
- Select all remaining columns (A–L), copy (Ctrl-C) and paste them (Ctrl-V) into a blank text file, and name it **`dallastemp.txt`**
- Read the data into R using the scan function
  `dfw.mon=scan("c:\\Your subdirectory\\dallastemp.txt")`
- Convert **`dfw.mon`** to a ***ts*** file
  `dfw.mon=ts(dfw.mon,start=c(1900,1),frequency=12)`
- Calculate the annual data using the aggregate command
  `dfw.yr=aggregate(dfw.mon,FUN=mean)`

These ***ts*** files are plotted in Figure 1.20.

Note: The ***ts*** files obtained above are identical to the DFW temperatures files by the same name in ***tswge***.

1.4.1.3 Data that Require Cleaning and Wrangling

In this section, we discuss the cleaning and wrangling steps that were used to create the "nice" ***ts*** file **`yellowcab.precleaned`** that was plotted in Figure 1.22. The New York City Taxicab information is available on the website www1.nyc.gov/site/tlc/about/about-tlc.page which is shown in Figure 1.21. In order to access the data, use the following steps:

QR 1.7 Wrangling

- On the homepage select *Data and Research*.
- Select *Data* and then *Aggregated Reports*.
- Select *Monthly Data Reports(CSV)* near the bottom of the screen
  The file **`data_reports_monthly.csv`** will be downloaded. Save this file to a subdirectory of choice.
- In R, read the csv file using
  `NYCabRaw = read.csv(file.choose(), header = TRUE)`
- *Note:* The data in data frame **NYCabRaw** are shown below. There it can be seen that the data will not be ready for use by simply creating a ***ts*** file as we have done in previous examples.[13]

`head(NYCabRaw)`

	Month.Year	License.Class	Trips.Per.Day	Farebox.Per.Day	Unique.Drivers	Unique.Vehicles	Vehicles.Per.Day
1	2021-02	FHV - High Volume	414,755	-	47,206	47,267	28,637
2	2021-02	Green	2,305	52,276	904	881	446
3	2021-02	Yellow	48,499	755,115	5,277	4,496	3,182
4	2021-01	FHV - Black Car	10,386	-	3,816	3,806	1,752
5	2021-01	FHV - High Volume	383,932		47,622	47,592	28,582
6	2021-01	FHV - Livery	23,890	-	5,132	5,030	3,086

	Avg.Days.Vehicles.on.Road	Avg.Hours.Per.Day.Per.Vehicle	Avg.Days.Drivers.on.Road	Avg.Hours. Per.Day.Per.Driver
1	17.0	7.1	17.1	7.1
2	14.2	4.1	13.9	4.1
3	19.8	7.9	17.6	7.6
4	14.3	4.4	14.3	4.4
5	18.6	6.9	18.7	6.8
6	19.0	5.1	18.9	5.0

	Avg.Minutes.Per.Trip	Percent.of.Trips.Paid.with.Credit.Card	Trips.Per.Day.Shared
1	16	-	-
2	18.8	79%	-
3	12.4	76%	-
4	22	-	-
5	15	-	-
6	15	-	-

"Wrangling" the dataset, in this context, involves filtering and subsetting the data to obtain the appropriate number of columns and rows (months). This will involve:

(a) Selecting only the **`Month.Year, License.Class`** and **`Trips.Per.Day`** columns
(b) Filtering the dataset to contain only the Yellow taxicabs
(c) Reversing the **`Trips.Per.Day`** column to reorder the data from earliest to latest dates. (The data originate from the website in descending order from latest to earliest dates.)
(d) Subsetting the dataset to only include the data from January 2019 to February 2021.
"Cleaning", on the other hand, will involve converting the trips per day into a numeric format in order to pass them to the plot function. This will involve only one step:
(e) Deleting the comma from the **`Trips.Per.Day`** column

13 Note that if you access the NYC Taxi Cab dataset using the procedure outlined above, it will be different from the one accessed here which was obtained in early 2021.

*(1) The **dplyr** Package*

To wrangle and clean this dataset in R, the ***dplyr*** package will be utilized. This package has become a leading source of functions for cleaning and wrangling data in R. While we will not provide a full tutorial here,[14] a key characteristic of the ***dplyr*** package is that the code is easy to implement and is intuitive to understand; thus, the following examples are intended to be both instructive and easy to follow.

(a) Selecting the `Month.Year, License.Class` *and* `Trips.Per.Day` *columns*

In order to select only the three columns of interest, use the **`select()`** function in ***dplyr***:

```
NYCabNew = NYCabRaw %>% select(c(Month.Year,License.Class,Trips.Per.Day))
head(NYCabNew)
```

```
  Month.Year       License.Class Trips.Per.Day
1    2021-02 FHV - High Volume       414,755
2    2021-02               Green         2,305
3    2021-02              Yellow        48,499
4    2021-01    FHV - Black Car        10,386
5    2021-01 FHV - High Volume       383,932
6    2021-01        FHV - Livery        23,890
```

(b) Filtering the Data to Retain Only the Yellow Taxicabs

Next, filter the dataset to contain only the trips per day for the Yellow cabs. This can be accomplished using the **`filter()`** function in ***dplyr***:

```
NYCabNew = NYCabNew %>% filter(License.Class == "Yellow")
head(NYCabNew)
```

```
  Month.Year License.Class Trips.Per.Day
1    2021-02        Yellow        48,499
2    2021-01        Yellow        44,052
3    2020-12        Yellow        47,145
4    2020-11        Yellow        50,285
5    2020-10        Yellow        54,221
6    2020-09        Yellow        44,646
```

We next perform the cleaning step.

(c) Deleting the Comma from the `Trips.Per.Day` *Column*

Visually, the **`Trips.Per.Day`** column already appears to be numeric; however, R considers this column to contain character strings. To verify this fact we issue the Base R **`class`** command

```
class(NYCabNew$Trips.Per.Day)
```

```
[1] "character"
```

To remove the comma we call the **`sub()`** function in base R which finds a specified pattern (the first argument), and replaces it with a character value (the second argument) in the given data (the third argument), and then returns the results as character values. For this reason, the result must be changed (cast) to a numeric value to complete the process. Here is the code to do both:

```
NoCommaTrips = sub(",","",NYCabNew$Trips.Per.Day)
NoCommaTrips = as.numeric(NoCommaTrips)
NoCommaTrips
```

```
  [1]  48499  44052  47145  50285  54221  44646  32491  25816  18325  11237   7928  96993 217216
 [14] 206604 220786 227654 231171 217747 194798 202443 231335 244017 247742 252634 250654 247315
 [27] 263609 271501 284121 267983 253182 253186 290362 297508 310169 304169 303280 282565 306706
```

14 A great resource to learn more about the ***dplyr*** and other useful R packages is Wickham and Grolemund (2017).

```
  [40] 309471 315084 298163 271676 277042 321877 325857 334865 332075 327451 313229 337071 336737
  [53] 350380 337321 320718 332231 371257 381878 397780 393886 392470 351816 369686 377076 397244
  [66] 374156 359029 372979 410831 424459 435701 430669 444633 411238 419776 440576 459087 445782
  [79] 409305 422771 460393 476569 487275 497661 466522 444537 450634 479527 483921 470147 406246
  [92] 445861 479148 492962 503249 507914 499556 476544 474005 459114 468377 484783 463841 463784
 [105] 503139 502097 515848 520764 516585 482811 481410 484122 506532 487470 427740 475475 503152
 [118] 501708 490577 518212 507167 434297 445727 463701 457996 517972 404115 472752 494137 499374
 [131] 504798 415567 397969 479376
```

(d) Reversing the **`Trips.Per.Day`** *Column to Reorder the Data from Earliest to Latest Dates*
Now that we have cleaned the **`Trips.Per.Day`** column, we need to reverse the order of the data so that the dates run from earliest to latest. We reverse the data **`NoCommaTrips`** using the Base R **`rev`** function.

```
NoCommaTrips = rev(NoCommaTrips)
NoCommaTrips
```

```
  [1] 479376 397969 415567 504798 499374 494137 472752 404115 517972 457996 463701 445727 434297
 [14] 507167 518212 490577 501708 503152 475475 427740 487470 506532 484122 481410 482811 516585
 [27] 520764 515848 502097 503139 463784 463841 484783 468377 459114 474005 476544 499556 507914
 [40] 503249 492962 479148 445861 406246 470147 483921 479527 450634 444537 466522 497661 487275
 [53] 476569 460393 422771 409305 445782 459087 440576 419776 411238 444633 430669 435701 424459
 [66] 410831 372979 359029 374156 397244 377076 369686 351816 392470 393886 397780 381878 371257
 [79] 332231 320718 337321 350380 336737 337071 313229 327451 332075 334865 325857 321877 277042
 [92] 271676 298163 315084 309471 306706 282565 303280 304169 310169 297508 290362 253186 253182
[105] 267983 284121 271501 263609 247315 250654 252634 247742 244017 231335 202443 194798 217747
[118] 231171 227654 220786 206604 217216  96993   7928  11237  18325  25816  32491  44646  54221
[131]  50285  47145  44052  48499
```

This listing is in the correct earliest to latest date order.

(e) Subsetting the Dataset to Only Include the Data from January 2019 to February 2021
We complete the wrangling by subsetting the data to contain only the trips for January 2019 to February 2021. To do this we will need to first create a ***ts*** object for the data so we can then use the **`window`** function.

```
NYCabNewts=ts(NoCommaTrips,start=c(2010,1),frequency=12)
NYCabNewtsShort=window(NYCabNewts,start=c(2019,1),end = c(2021,02))
NYCabNewtsShort
```

```
         Jan     Feb     Mar     Apr     May     Jun     Jul     Aug     Sep     Oct     Nov     Dec
2019  247315  250654  252634  247742  244017  231335  202443  194798  217747  231171  227654  220786
2020  206604  217216   96993    7928   11237   18325   25816   32491   44646   54221   50285   47145
2021   44052   48499
```

Notice that this is the same as ***tswge ts*** object **`taxicab.precleaned.ts`** .

Note: Now it is clear why the descriptive "pre-cleaned" was included in the ***tswge*** taxicab ***ts*** object.[15] We can now plot the data to complete the example.

```
plotts.wge(NYCabNewtsShort, xlab = "Data", ylab = "Monthly Yellow Taxi Trips")
```

This code produces the plot previously shown in Figure 1.22.

15 Note that if you access the NYC Taxi Cab dataset using the procedure described above, it will be different from the one accessed here which was obtained in early 2021.

1.4.1.4 Programatic Method of Ingestion and Wrangling Data from Tables on Web Pages

Earlier, it was shown that time series may be obtained from tables on web pages by copying and pasting the data into a spreadsheet, saving it as a .csv file and then loading it into R. While this is a very useful method, it has two significant drawbacks: a) It may not always work or may take significant wrangling to get it into the right form. b) An actual person will need to be available to perform the steps. If the data are updated often this may create a significant strain on human and other resources. The video below demonstrates a programatic method that may work when the copy and paste method presents challenges and, even when the other method does work, this method may be preferred as it automates the process thus freeing up valuable resources.

QR 1.8 Web Scraping

1.5 CONCLUDING REMARKS

In this opening chapter, the importance of the field of time series data has been motivated. Time series data are widely available, and the analysis of such data is in demand. The first step in analyzing time series data is having the skillset to access and manipulate time series data in a way that is compatible for statistical software, in our case, R. Common, basic R code has been introduced that will serve as a great starting place in the beginning steps of the analysis of a time series dataset. In this text, the authors' R package ***tswge*** will be the primary programming tool for analyses. Now that you have the tools to produce a time series dataset that is in a form that can be analyzed, you are ready to learn the next steps in the process! In the chapters ahead, a comprehensive array of statistical methods will be introduced that will be beneficial in providing *useful* forecasts and inferences to a wide variety of time series settings.

APPENDIX 1A

tswge function **plotts.wge**

```
plotts.wge = function (x,style = 0,ylab = "",xlab = "Time", main = "", col =
"black", text_size= 12, lwd = .75, cex = .5, cex.lab = .75, cex.axis = .75, xlim
= NULL, ylim = NULL)
```

is a function designed to produce time series plots. It has two uses: (1) using default options it plots a dataset as a time series, connecting the dots, etc. with no parameter specifications needed (2) creates custom plots using the parameters in base R function **`plot`** along with the **`style`** parameter described below.

`style` specifies the appearance of the output. **`style=0`** produces a plot similar to Figure 1.12, and Figure 1.13 was obtained using **`style=1`** which uses ***ggplot2***. (See the related discussion in Section 1.3.5)

Notes: Color, axis labeling and numbering, and other plot-related options can be modified using the plot parameters in the call statement. For more information on plot parameters use **`?plot`**.

TSWGE DATASETS RELATED TO THIS CHAPTER

Some of the following datasets were downloaded from websites. In order to avoid the necessity to perform the download for future use we have included the following datasets in ***tswge***.

`bitcoin` − Bitcoin data from May 1, 2020 through April 30, 2021 (includes imputed values from Section 1.4.1.1)
`dfw.mon` – DFW monthly temperature data from January 1900 through December 2020
`dfw.yr` – DFW annual temperature data from January 1900 through December 2020
`dfw.2011` – DFW monthly temperatures from January 2011 through December 2020
`dow1985` – Monthly DOW closing averages from March 1985 through December 2020
`ozona` – Daily number of chicken-fried steaks sold at Ozona Bar and Grill during June and July, 2019
`sunspot2.0` – Sunspot2.0 annual data from 1700 through 2020
`sunspot2.0.month` – Sunspot2.0 monthly data from January 1749 through December 20020
`tesla` – Tesla stock prices from January 1, 2020 through April 30, 2021
`wtcrude2020` – Monthly WTI crude oil prices from January 1990 through December 2020
`yellowcab.precleaned` – Number of trips per month for Yellow Cabs in New York City from January 2019 through December 2020

PROBLEMS

1.1 Google search "sunspot data" and find the classical (Wolfer) *annual* sunspot data from 1749 through 2014. Plot a subset of the sunspot2.0 data from 1749 through 1914. On the same graph plot the classical sunspot data (using R function **`points`** (or **`lines`**)). Label the axes and compare the similarities and differences between the two time series.
1.2 Repeat Problem 1.1 using *monthly* sunspot data from 1749 through 2014.
1.3 The US Bureau of Transportation records monthly data on the number of air passengers in the US. Use this data to create a time series plot of the number of air passengers in the US from 2002 to the present.
1.4 Use the World Bank API (WDI package) to gather data that will allow you to create a time series plot of the total worldwide railroad passengers. Compare any trends you see in this plot to that of the air passenger data displayed in Problem 1.3.
1.5 Use the World Bank API (WDI package) to gather data that will allow you to construct a time series plot that compares the GDP of the US, Mexico, and Canada over the last 30 years.
1.6 Using the New York Taxi and Limousine Commission website, construct and plot a time series of the number of Yellow Taxis per month for 2010 to 2020 and label the axes appropriately.