Getting Started with Stata

★ From Your First Do-file to Time-Series Forecasts ★

Xuexin Li

Halloween Edition · October 31, 2026

Stata looks a little spooky? Try one command and see what happens. No coding experience is needed. You will install Stata, run your first do-file, and follow the clues in a monthly sales series. Each example shows you what to type and what to look for in the results.

Already have Stata up and running? Skip Section 1 and head straight to your first command.

1 Installing and Activating Stata

1.1 Log in to the Campus Software Library

Visit the UW–Madison software page:

`https://it.wisc.edu/services/software/`

Click **Log in to the Campus Software Library**, as shown in Figure 1.

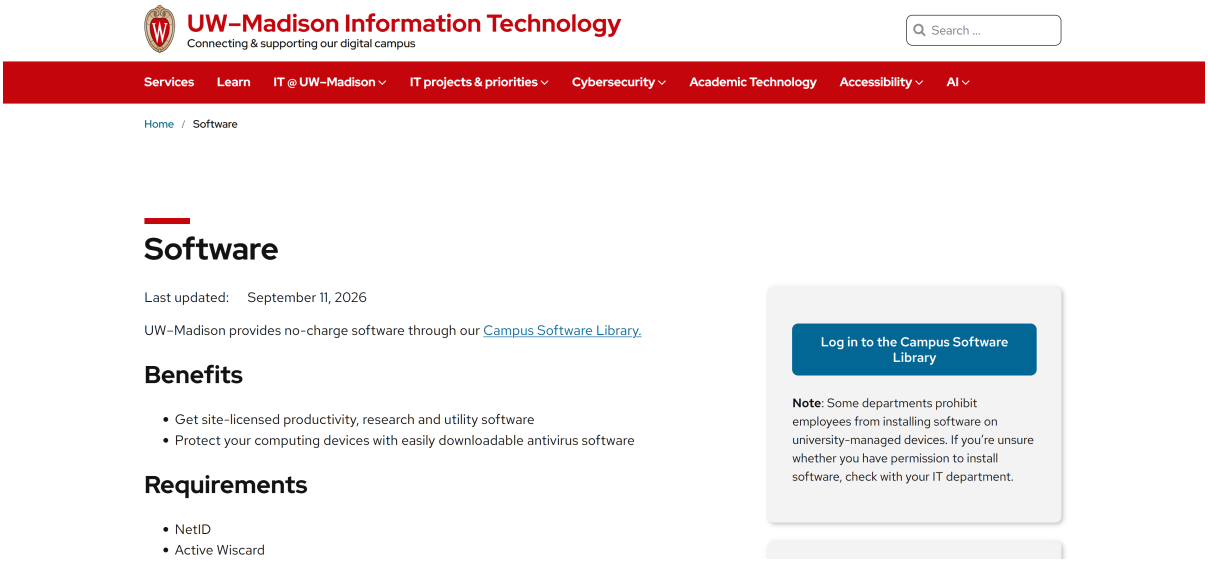


Figure 1: The UW–Madison software page.

Sign in with your **NetID** and password and complete the login prompts.

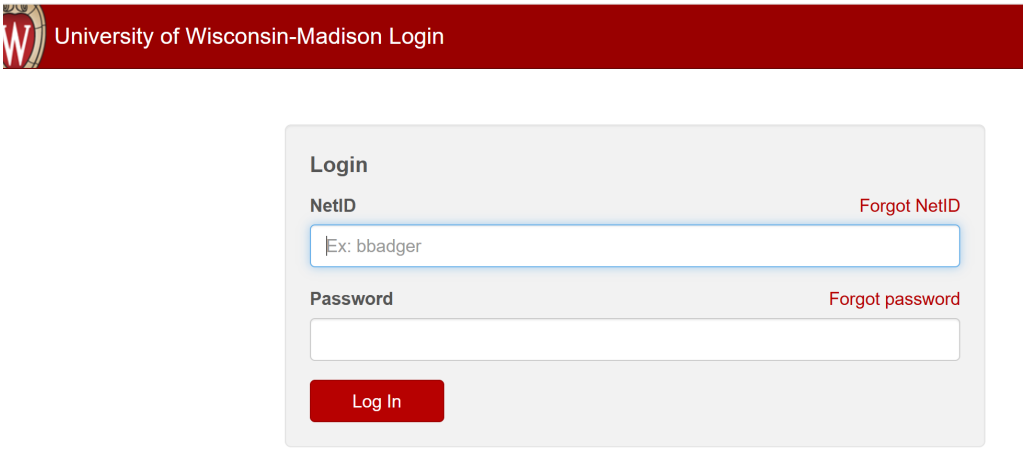


Figure 2: Log in using your UW–Madison NetID.

After logging in, you should arrive at the **Campus Software Library**.

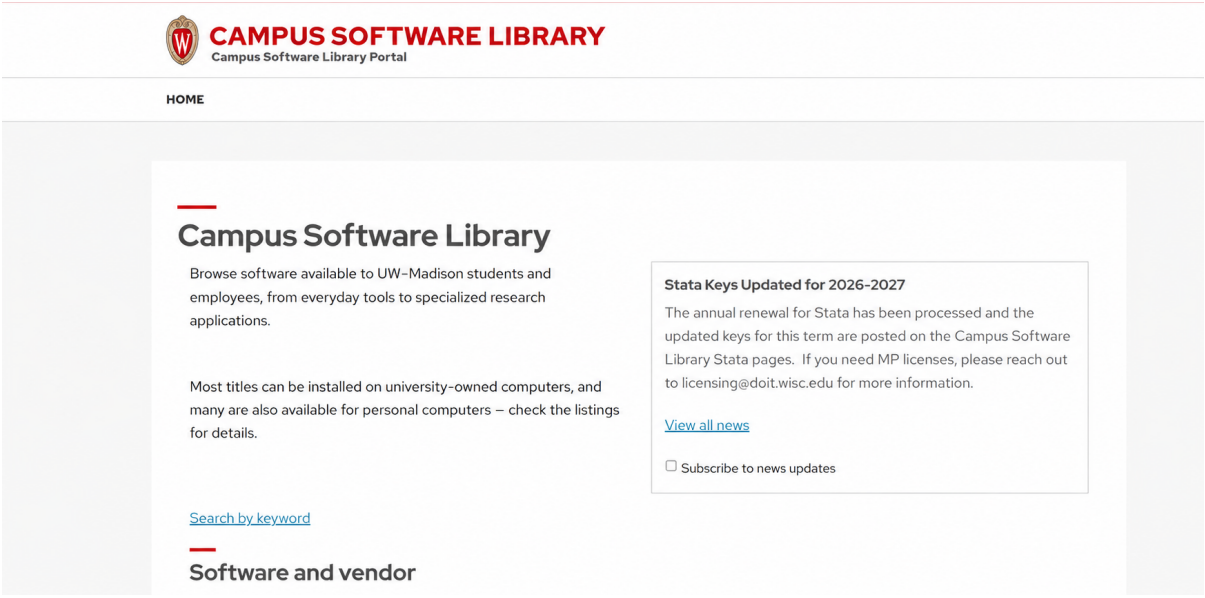


Figure 3: The Campus Software Library.

1.2 Find Stata and choose your operating system

Scroll down through the software list and select **STATA by StataCorp**.

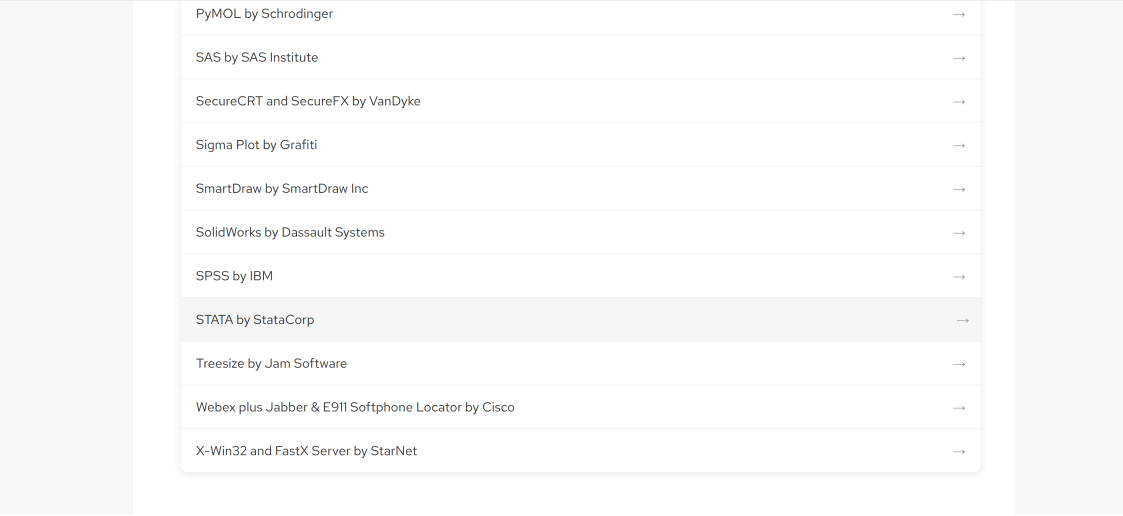


Figure 4: Find STATA by StataCorp in the software list.

On the Stata page, expand the installer entry that matches your operating system: **Windows**, **Mac**, or **Linux**.

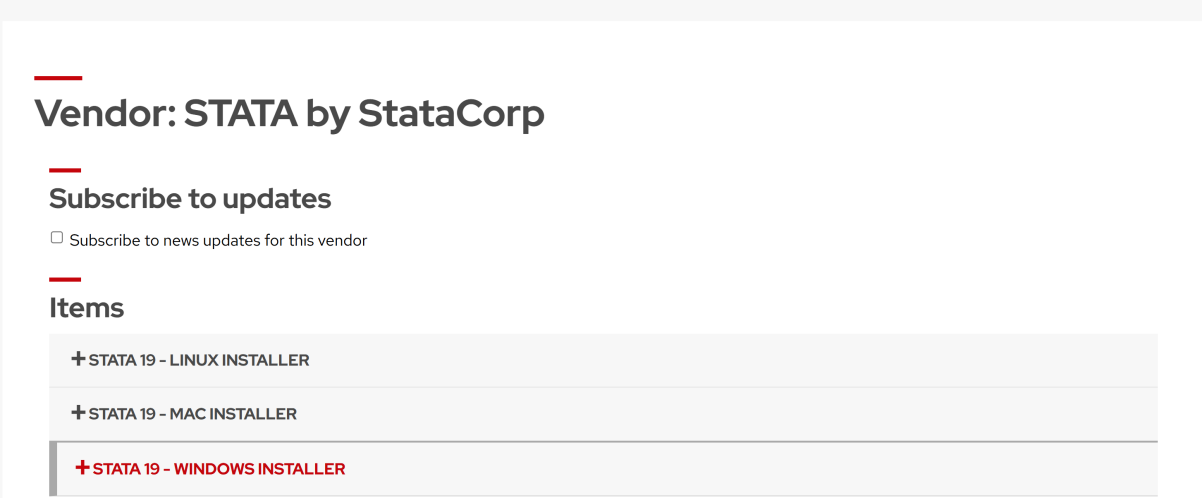


Figure 5: Expand the installer entry for your operating system.

Open the download link for your operating system and follow the installation instructions. Keep the license information on that page handy.

When asked to choose a Stata edition, select **Stata/SE**, the edition we use in this course.

1.3 Activate your installation

Keep the following three items from the Campus Software Library available:

- **Serial number**
- **Code**
- **Authorization**

During setup or the first launch, copy each license item into the field with the same name.

Use the authorization code from the Campus Software Library. Your NetID password is for signing in to the library.

Once activation is complete, open Stata. You should see a window similar to Figure 6.

For additional platform-specific installation instructions, see the official Stata installation guide.

2 Getting Familiar with Stata

The screenshots below show Stata on Windows. The appearance may differ slightly on a Mac or in another version of Stata, but the basic workflow is the same.

2.1 The main window

Start with two areas: the **Results window** and the **Command window**.

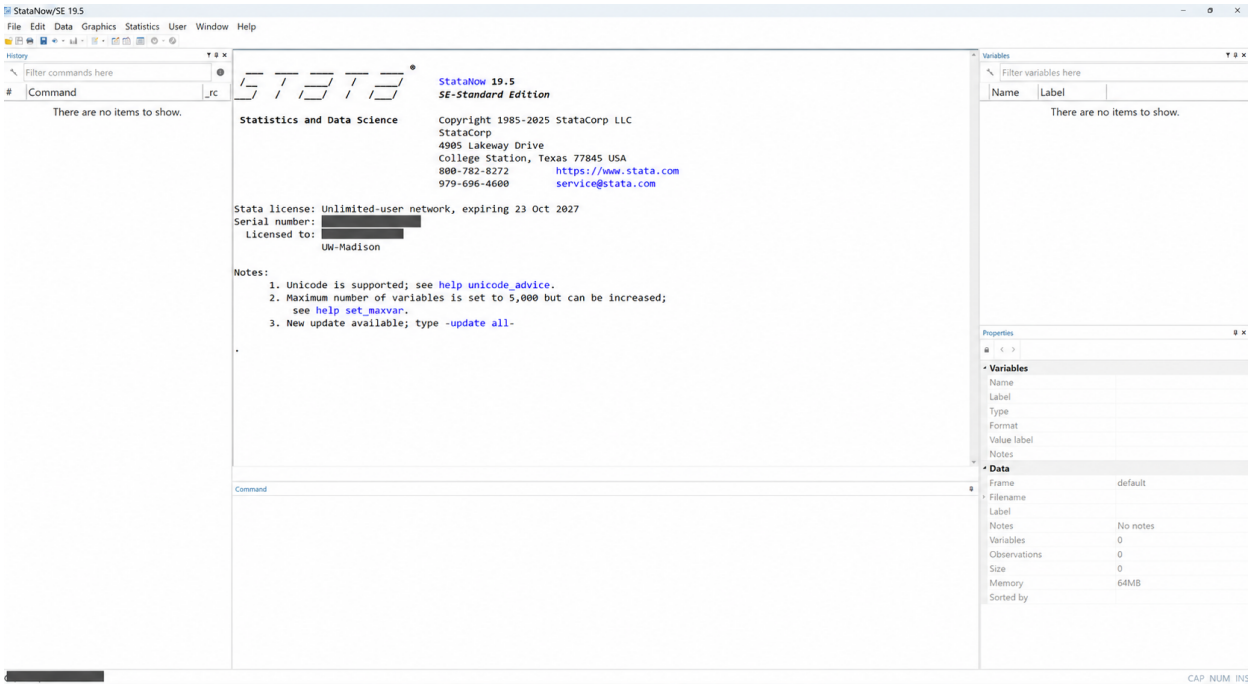


Figure 6: The main Stata window. Personal and license information is hidden.

- **Results window:** The large central area displays commands, statistical results, and messages. When something goes wrong, the error message usually appears here.
- **Command window:** The area below the Results window lets you type a command and run it by pressing Enter.
- **History window:** The panel on the left lists commands you have already run.
- **Variables window:** The panel on the right lists the variables in the dataset currently loaded into Stata.
- **Properties window:** This panel shows information about the dataset and selected variables.

2.2 Try your first command

Click inside the Command window, type the following, and press Enter:

```
display 2 + 2
```

You should see **4** in the Results window. This is where Stata displays the output of each command you run. For the rest of the exercise, type the commands as shown in the code boxes. In the Results window, Stata displays a dot before each command. You can leave that dot out when typing.

2.3 The toolbar

The toolbar near the top-left corner provides shortcuts to common tasks.

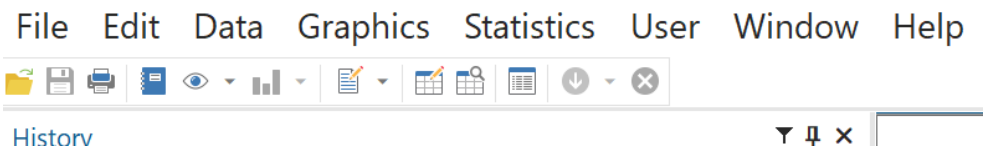


Figure 7: The Stata toolbar.

Hover over a toolbar button to see its name. Find the Do-file Editor for writing commands and the Data Editor for viewing observations.

3 Writing Code in the Do-file Editor

Use the Command window to try a single command. For the worked example, use the **Do-file Editor** to write and save a series of commands.

The file you save is called a **do-file**. When you run it, Stata executes the commands in order. You can edit a line and rerun the analysis without retyping the other commands.

3.1 Open the Do-file Editor

Click the **New Do-file Editor** button on the toolbar.

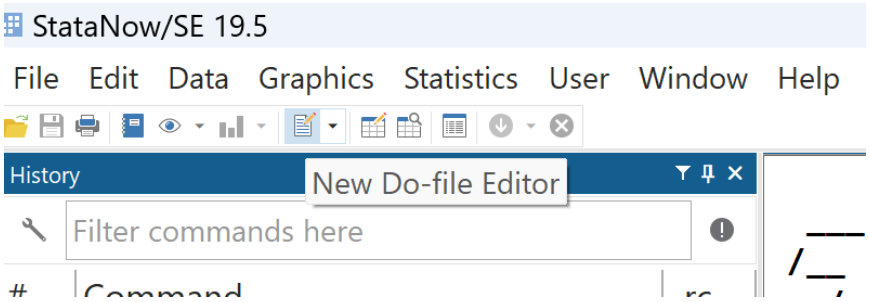


Figure 8: Open a new Do-file Editor from the toolbar.

You can also open it by typing the following in the Command window:

```
doedit
```

A blank editor window will appear.

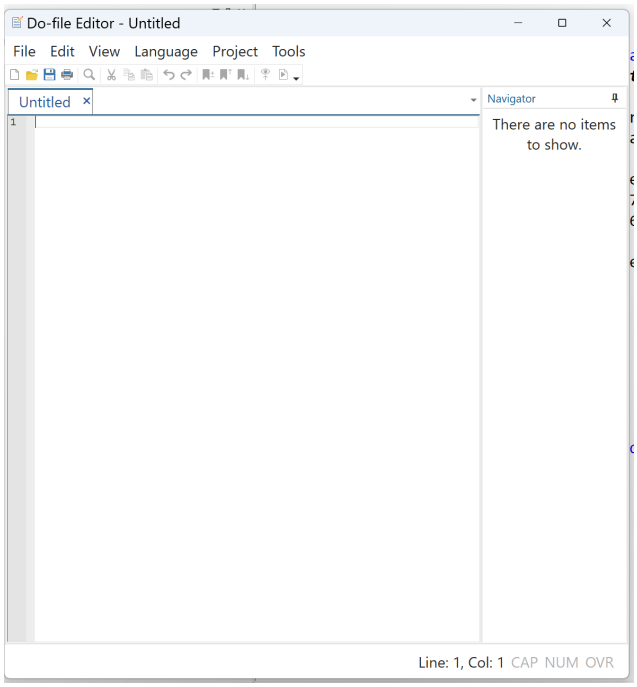


Figure 9: A blank Do-file Editor.

3.2 Write, save, and run a short script

Type the following lines into the Do-file Editor:

```
// My first Stata do-file
display "Hello, Stata!"
display 2 + 2
```

The first line is a comment. Stata skips lines beginning with //, so you can use them to explain your code.

The two `display` commands print a greeting and the number 4.

1. Choose **File > Save As** and save the file as **practice.do** in a folder for this course.
2. Click the small triangular **Execute (do)** button on the Do-file Editor toolbar to run your code.
3. Return to the main Stata window and look at the Results window.

If you highlight part of the code before clicking **Execute (do)**, Stata runs only the selected code. If nothing is selected, it runs the whole do-file.

Pressing Enter inside the Do-file Editor starts a new line; it does not execute the command as it does in the Command window.

Saving the do-file saves your commands. Later you will use a log to keep the text output and export graphs as image files.

4 Viewing Data

The **Data Editor** displays the dataset currently loaded into Stata in a spreadsheet-like layout.

- Each **row** represents an observation. In our monthly time-series example, one row will represent one month.
- Each **column** represents a variable, such as the date or a sales index.

The toolbar button shown in Figure 10 opens **Data Editor (Edit)**.

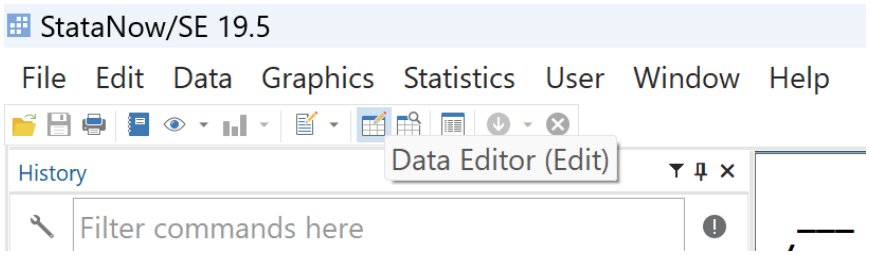


Figure 10: The Data Editor (Edit) toolbar button.

The editor will be empty before a dataset has been imported, as shown below.

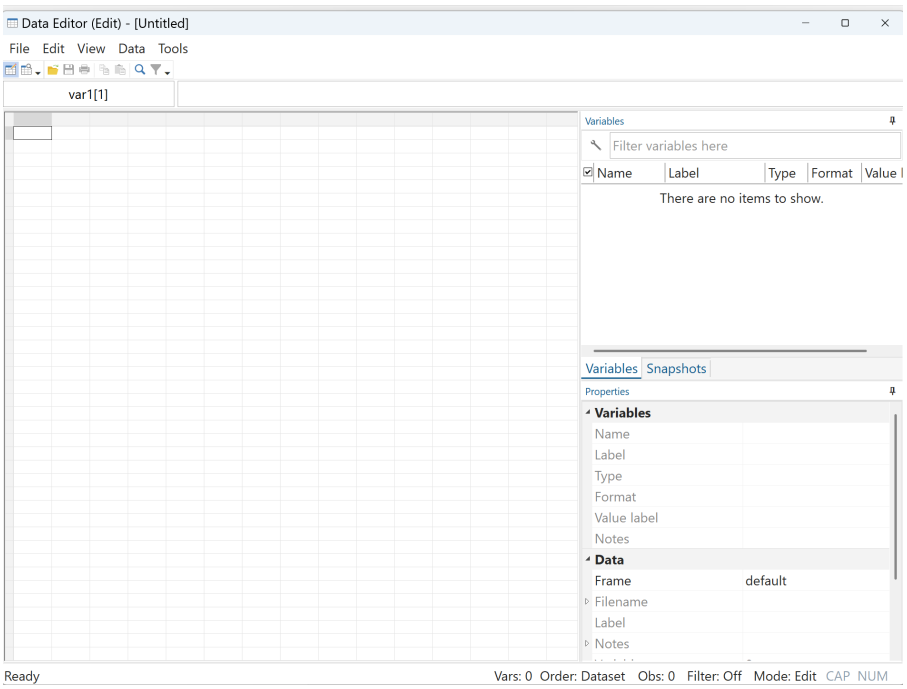


Figure 11: An empty Data Editor before data have been loaded.

Stata provides two modes:

- **Edit mode** allows you to change values.
- **Browse mode** allows you to inspect the data without changing values.

For checking data, we recommend Browse mode. After loading a dataset, type:

```
browse
```

To keep a record of changes to the data, make them in your do-file.

5 Using Stata's Help

You do not need to memorize every Stata command. Even experienced users regularly use Stata's help pages to check how a command works. To open a help page, type `help` followed by the command name in the Command window.

For example, to look up `summarize`, type:

```
help summarize
```

This opens the help page for the command used to calculate summary statistics.

When reading a help page, start with:

1. **Description:** What does the command do?
2. **Syntax:** How is the command written?
3. **Examples:** What does a typical use look like?

In a help page, square brackets usually mark an optional part of a command. For example, the syntax for `summarize` shows `[varlist]`, which means you can choose which variables to summarize.

Typing `summarize` by itself gives summary statistics for all numeric variables in the dataset. To summarize only the sales index, type `summarize sales_index`. Enter the variable name without the brackets.

Later, when we examine autocorrelations and partial autocorrelations, the following help page will be useful:

```
help corrgram
```

If you do not know the command's name, you can search by topic:

```
search partial autocorrelation
```

6 Ready for the Worked Example

Keep your Do-file Editor open and add each code block to the same do-file as you go. You will import and plot monthly sales data, explore how sales are related across months, and compare two autoregressive models. Finally, you will use one model to forecast sales and check how close its forecasts are to the observed values.

6.1 Importing and Checking the Monthly Sales Data

Save `simulated_sales.xlsx` in your exercise folder. The `Data` worksheet contains one row per month from January 2015 through December 2025. In your do-file, enter the commands below. Replace the path in quotation marks with the full path to the folder containing your copy of the Excel file.

```
clear all
set more off
capture log close

cd "path/to/your/exercise/folder"
log using "manual_example.log", text replace
pwd
```

These commands start a clean session and open a plain-text log in your exercise folder. When you rerun the do-file, `replace` overwrites the previous log. Check that the path printed by `pwd` matches the folder where you saved the Excel file.

Before running the command below, try finding the same options in Stata's menus. Open **File > Import** and look for the Excel import option. Browse to `simulated_sales.xlsx`, select the `Data` worksheet, and use the preview to find the row containing the column headings. Look for the checkbox that tells Stata to use that row as variable names.

Click **Cancel** when you have finished exploring. Then add the command below to your do-file and run it. Notice how `sheet("Data")` selects the worksheet and `firstrow` uses its first row as variable names.

```
import excel "simulated_sales.xlsx", sheet("Data") firstrow clear
```

Check the variable names, count the observations, and check the sales index for missing values.

```
describe
count
misstable summarize sales_index
```

```
Contains data
 Observations:           132
    Variables:             2
-------------------------------------------------------------------------------
Variable      Storage   Display    Value
    name         type    format    label      Variable label
-------------------------------------------------------------------------------
month           str7    %9s                   month
sales_index     double  %10.0g                sales_index
-------------------------------------------------------------------------------
```

Figure 12: Output from `describe` after importing the monthly sales data.

Compare your output with Figure 12. You should see 132 observations and two variables. In the storage type column, `double` tells you that `sales_index` is numeric, while `str7` tells you that Stata imported `month` as text. The final command should find no missing sales values.

6.2 Setting Up the Monthly Time Variable

Stata imported `month` as text. Create a monthly date variable called `mdate` so that Stata can use the dates in time-series commands.

```
generate mdate = monthly(month, "YM")
format mdate %tm
```

The letters `YM` tell Stata to read the year first and the month second. The new variable stores a numeric monthly date. The format `%tm` displays January 2015 as `2015m1`.

Use `tsset` to tell Stata that you are working with time-series data. The command identifies `mdate` as the date variable, so Stata can use the calendar to find earlier observations.

```
tsset mdate
```

In the Results window, look for a time step of one month. With monthly data, `L.sales_index` refers to the sales index exactly one month earlier.

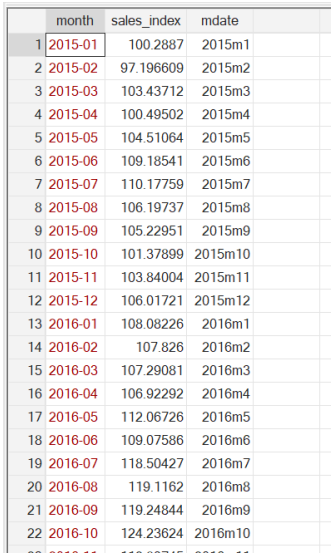
For example, suppose January is missing and the row for February comes immediately after December. Stata recognizes that the two dates are two months apart. For February, `L.sales_index` is missing because there is no January observation, while `L2.sales_index` gives December's value.

Open the Data Editor in **Browse** mode. In Figure 13, January and February 2015 appear as `2015m1` and `2015m2` in the new column.

6.3 Summarizing and Plotting the Sales Index

Start by summarizing the monthly sales index.

```
summarize sales_index
```



	month	sales_index	mdate
1	2015-01	100.2887	2015m1
2	2015-02	97.196609	2015m2
3	2015-03	103.43712	2015m3
4	2015-04	100.49502	2015m4
5	2015-05	104.51064	2015m5
6	2015-06	109.18541	2015m6
7	2015-07	110.17759	2015m7
8	2015-08	106.19737	2015m8
9	2015-09	105.22951	2015m9
10	2015-10	101.37899	2015m10
11	2015-11	103.84004	2015m11
12	2015-12	106.01721	2015m12
13	2016-01	108.08226	2016m1
14	2016-02	107.826	2016m2
15	2016-03	107.29081	2016m3
16	2016-04	106.92292	2016m4
17	2016-05	112.06726	2016m5
18	2016-06	109.07586	2016m6
19	2016-07	118.50427	2016m7
20	2016-08	119.1162	2016m8
21	2016-09	119.24844	2016m9
22	2016-10	124.23624	2016m10

Figure 13: The original month labels and the converted Stata monthly dates in the Data Editor.

Read across the `sales_index` row. `Obs` should be 132, `Mean` gives the average index, and `Min` and `Max` give its range. Plot the series to see when the highs and lows occurred.

```
tsline sales_index, ///
    title("Monthly Sales Index (Simulated Data)") ///
    xtitle("Month") ///
    ytitle("Sales index") ///
    name(sales_plot, replace)
```

In the Do-file Editor, select the entire block, from `tsline` through the final `name()` option, and run it all at once. The `///` at the end of a line tells Stata that the same command continues on the next line. Running only part of the block can leave the command incomplete or cause Stata to read an option as a separate command and report an error.

Stata uses the monthly dates declared with `tsset` for the horizontal axis. The `name()` option gives the graph a name so you can refer to it later in the do-file.

Save the displayed graph to your working folder.

```
graph export "sales_series.png", replace
```

Open your working folder and look for `sales_series.png`. If you run the command again, `replace` overwrites the existing file with the current graph.

Look at Figure 14. When the sales index is high in one month, does it tend to stay high the next month? The lag-one autocorrelation helps answer this question by comparing each month's index with the previous month's index.

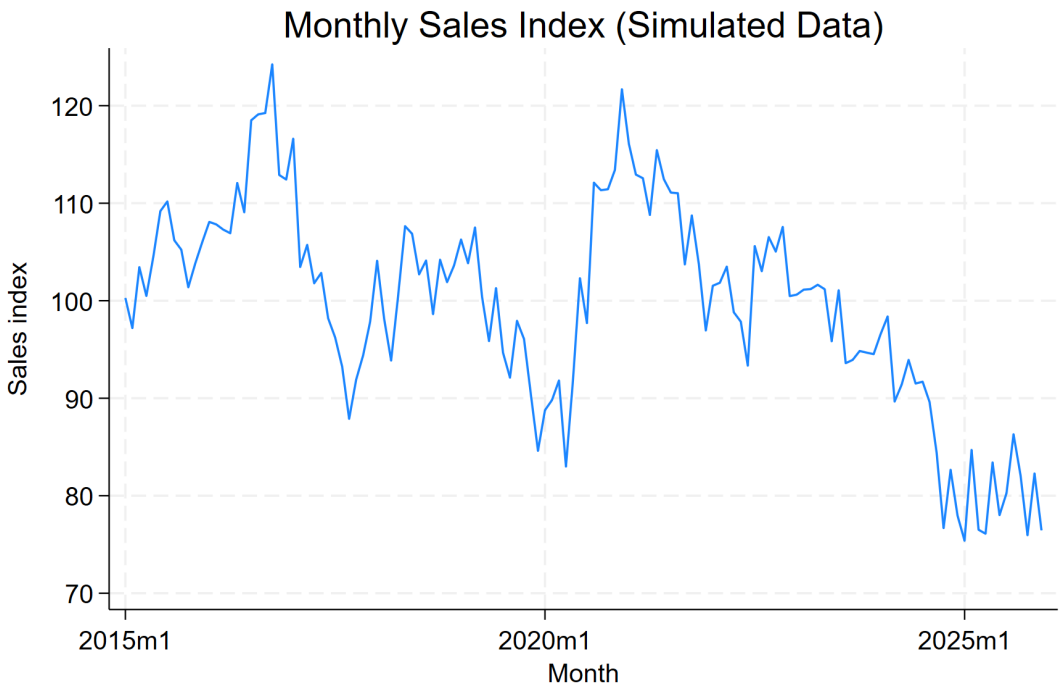


Figure 14: Simulated monthly sales index, January 2015 through December 2025.

6.4 Reading the ACF and PACF

A lag counts months back from the current observation. Lag 1 refers to last month and lag 12 to the same month a year earlier.

First, calculate the reference bounds used in class. For a white noise series with T observations, $\pm 1.96/\sqrt{T}$ gives approximate 95% reference bounds at an individual lag. Run the following lines together so that Stata can use the local values defined in the block.

```
quietly count if !missing(sales_index)
local T_series = r(N)
local bound = 1.96 / sqrt(‘T_series’)

display "Number of observations: " ‘T_series’
display "Approximate 95% reference bounds: +/- " %6.3f ‘bound’
```

Stata stores the count in `r(N)`. With $T = 132$ months, the reference lines are at approximately -0.171 and $+0.171$.

Use `corrgram` to calculate the autocorrelation function (ACF) and the partial autocorrelation function (PACF) for the first 12 lags.

```
corrgram sales_index, lags(12)
```

Find lag 2 in Figure 15 and compare the two columns.

```
.
. display "Number of observations: " `T_series'
Number of observations: 132

. display "Approximate 95% reference bounds: +/- " %6.3f `bound'
Approximate 95% reference bounds: +/-  0.171

.
. // ============================================================
. // 12. Display the ACF and PACF in a table
. // ============================================================
.
. corrgram sales_index, lags(12)

                                          -1       0       1 -1       0       1
 LAG       AC       PAC      Q     Prob>Q  [Autocorrelation]  [Partial autocor]
-------------------------------------------------------------------------------
1        0.8743   0.9071   103.21  0.0000
2        0.8216   0.2672   195.05  0.0000
3        0.7675   0.1180   275.81  0.0000
4        0.6778  -0.2054    339.3  0.0000
5        0.6239   0.0052   393.51  0.0000
6        0.5580  -0.0146   437.22  0.0000
7        0.4609  -0.1443   467.28  0.0000
8        0.3827  -0.1064   488.17  0.0000
9        0.3133   0.0668   502.28  0.0000
10       0.2334   0.0024   510.18  0.0000
11       0.1814   0.0389   514.99  0.0000
12       0.1501   0.2069   518.31  0.0000
```

Figure 15: Reference bounds and sample autocorrelations (`AC`) and partial autocorrelations (`PAC`) for lags 1 through 12.

- **AC** is 0.8216. This is the sample autocorrelation between the current sales index and its value two months earlier. The positive value indicates that months with a high sales index tend to be paired with high values two months earlier.
- **PAC** is 0.2672. This measures the relationship with sales two months earlier after accounting for sales one month earlier. In a regression of the current sales index on its first two lags, 0.2672 is the coefficient on the second lag. It exceeds the upper reference bound of 0.171.

At lag 3, `AC` is still large (0.7675), while `PAC` is inside the bounds (0.1180). The association is much weaker once the two intervening months are accounted for.

Check both sides of zero. The lag-4 `PAC` (-0.2054) falls below the lower bound, and the lag-12 value (0.2069) exceeds the upper bound. The bounds apply to one lag at a time, so an occasional crossing can occur by chance.

6.5 Plotting the ACF and PACF

Use `ac` to plot the `AC` column. Each dot gives the autocorrelation at one lag, from 1 to 12 months.

Run the plotting commands in the same do-file run as the lines that calculate `bound`. This lets both graphs use the reference bounds from the previous step, approximately -0.171 and $+0.171$.

```
ac sales_index, lags(12) ///
    ciopts(fcolor(none) lcolor(none)) ///
    yline(0, lcolor(gs10)) ///
    yline(-‘bound’ ‘bound’, lcolor(gs6) lpattern(dash)) ///
    xlabel(1(1)12) ///
    xtitle("Lag (months)") ///
    ytitle("Autocorrelation") ///
    title("ACF of the Sales Index") ///
    note("Dashed lines: +/- 1.96/sqrt(T), approximate pointwise 95% bounds") ///
    legend(off) ///
    name(sales_acf, replace)
```

```
graph export "sales_acf.png", replace width(1800)
```

The `ciopts()` option hides Stata's default confidence band. The two `yline()` options draw a line at zero and dashed lines at the lower and upper reference bounds. The graph is named `sales_acf` in Stata, and `graph export` saves a PNG copy in your working folder.

Use `pac` for the `PAC` column. Keep the same lags and reference lines so you can compare the plots.

```
pac sales_index, lags(12) ///
    ciopts(fcolor(none) lcolor(none)) ///
    yline(0, lcolor(gs10)) ///
    yline(-`bound' `bound', lcolor(gs6) lpattern(dash)) ///
    xlabel(1(1)12) ///
    xtitle("Lag (months)") ///
    ytitle("Partial autocorrelation") ///
    title("PACF of the Sales Index") ///
    note("Dashed lines: +/- 1.96/sqrt(T), approximate pointwise 95% bounds") ///
    legend(off) ///
    name(sales_pacf, replace)

graph export "sales_pacf.png", replace width(1800)
```

Pass the two graph names to `graph combine`. The option `cols(2)` places them side by side. Export the result as a single image.

```
graph combine sales_acf sales_pacf, cols(2) ///
    name(sales_correlations, replace)

graph export "sales_acf_pacf.png", replace width(2400)
```

Read Figure 16 from left to right. The ACF declines gradually and first falls inside the reference bounds at lag 12. The PACF has its largest spike at lag 1. At lag 2, it is still above the upper line; at lag 3, it is inside the bounds. You can also see the negative spike at lag 4 and the positive spike at lag 12 cross their respective reference lines.

The spike at lag 2 gives us a reason to try an AR(2) model. An AR(1) uses only last month's sales index. The lag-2 PACF suggests that the index from two months ago may still help explain this month's sales after we account for last month. The plot alone does not settle how many lags to include. In the next section, we fit AR(1) and AR(2) to the same months and examine what changes when we add the second lag.

6.6 Estimating and Comparing AR(1) and AR(2) Models

Does knowing the index two months ago help once we know last month's index? Fit a one-lag model, AR(1), and a two-lag model, AR(2), using data through December 2024. Set aside 2025 to evaluate forecasts.

First, mark the months that both models can use.

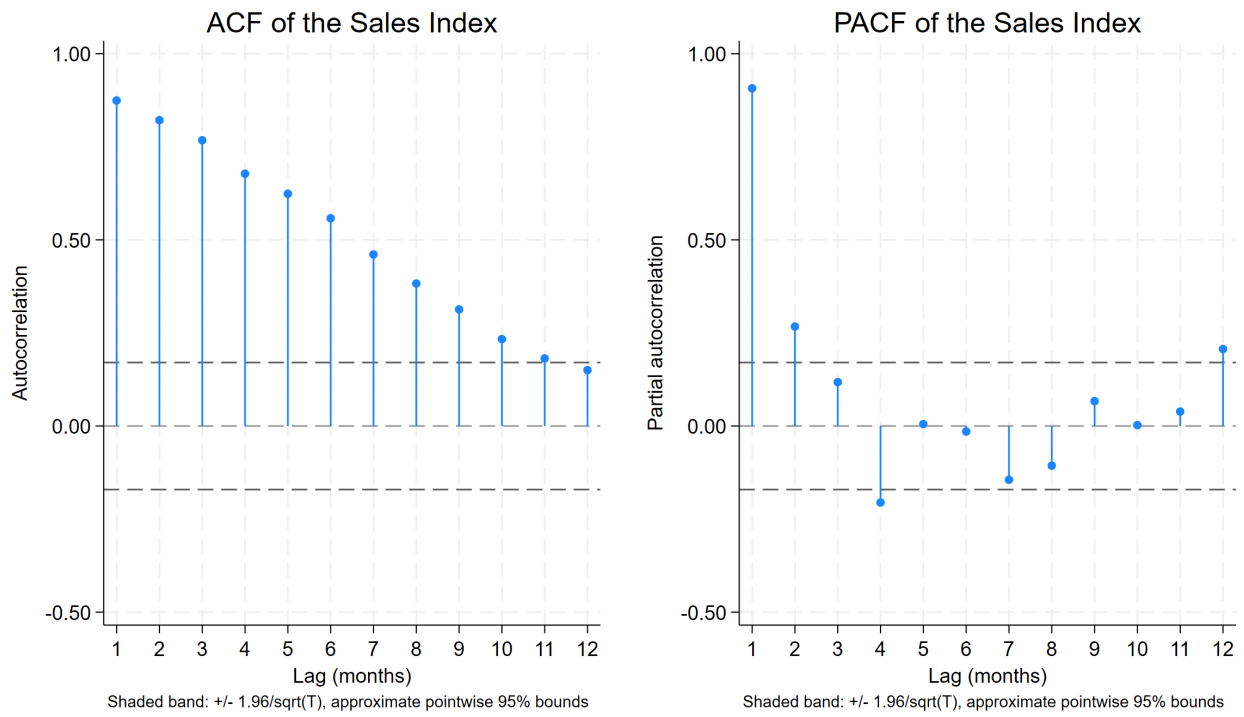


Figure 16: ACF and PACF of the monthly sales index. The dashed reference lines are at approximately -0.171 and $+0.171$.

```
generate byte train = ///
    mdate <= tm(2024m12) & ///
    !missing(sales_index, L.sales_index, L2.sales_index)

count if train
```

The first month with both lags available is March 2015, when January and February supply the two lagged values. The count should be 118. Use `if train` in both regressions to compare them over these same months.

Estimate the AR(1). Regress the current sales index on last month's index. Store the regression results under the name `AR1`, then save the residuals for the next part of the exercise.

```
regress sales_index L.sales_index if train
estimates store AR1

predict double resid_ar1 if e(sample), residuals
```

A residual is the observed index minus its fitted value. The condition `if e(sample)` saves residuals for the months used to fit the model.

In Figure 17, find the `L1.` row and read across to the `Coefficient` column. The estimate is 0.876, so a

```
. regress sales_index L.sales_index if train
```

Source	SS	df	MS	Number of obs	=	118
				F(1, 116)	=	302.69
Model	6888.41005	1	6888.41005	Prob > F	=	0.0000
Residual	2639.82612	116	22.7571217	R-squared	=	0.7229
				Adj R-squared	=	0.7206
Total	9528.23616	117	81.4379159	Root MSE	=	4.7704

sales_index	Coefficient	Std. err.	t	P>|t|	[95% conf.	interval]
sales_index						
L1.	.8755827	.0503265	17.40	0.000	.7759048	.9752606
_cons	12.49718	5.139866	2.43	0.017	2.31703	22.67733

Figure 17: AR(1) regression of the monthly sales index on its first lag.

one-point higher sales index last month is associated with a 0.876-point higher fitted index this month. The `_cons` row gives the intercept. Using y_t for the sales index in month t, the fitted equation is

$$\widehat{y}_t = 12.497 + 0.876\, y_{t-1}.$$

Estimate the AR(2). Add `L2.sales_index`, the sales index from two months ago. Keep the same sample and save this model's residuals separately.

```
regress sales_index L.sales_index L2.sales_index if train
estimates store AR2

predict double resid_ar2 if e(sample), residuals
```

```
. regress sales_index L.sales_index L2.sales_index if train
```

Source	SS	df	MS	Number of obs	=	118
				F(2, 115)	=	160.27
Model	7012.35868	2	3506.17934	Prob > F	=	0.0000
Residual	2515.87749	115	21.8771955	R-squared	=	0.7360
				Adj R-squared	=	0.7314
Total	9528.23616	117	81.4379159	Root MSE	=	4.6773

sales_index	Coefficient	Std. err.	t	P>|t|	[95% conf.	interval]
sales_index						
L1.	.6915012	.0917375	7.54	0.000	.5097868	.8732155
L2.	.2229397	.0936618	2.38	0.019	.0374138	.4084657
_cons	8.509767	5.310652	1.60	0.112	-2.009612	19.02915

Figure 18: AR(2) regression of the monthly sales index on its first two lags.

In Figure 18, the `L2.` coefficient is 0.223. Holding last month's index fixed, a one-point higher index

two months ago is associated with a 0.223-point higher fitted index today. The standard error is 0.094 and the 95% confidence interval is 0.037 to 0.408. The p-value of 0.019 tests whether the second-lag coefficient is zero. The fitted equation is

$$\widehat{y}_t = 8.510 + 0.692\, y_{t-1} + 0.223\, y_{t-2}.$$

Read the models side by side. Use the stored model names to display both sets of estimates in one table.

```
estimates table AR1 AR2, ///
    b(%9.3f) se(%9.3f) stats(N rmse)
```

```
. estimates table AR1 AR2, ///
>     b(%9.3f) se(%9.3f) stats(N rmse)
```

Variable	AR1	AR2
sales_index		
L1.	0.876	0.692
	0.050	0.092
L2.		0.223
		0.094
_cons	12.497	8.510
	5.140	5.311
N	118	118
rmse	4.770	4.677

Legend: b/se

Figure 19: AR(1) and AR(2) estimates for March 2015 through December 2024. Each coefficient has its standard error directly below it.

Each column in Figure 19 is one model. The legend `b/se` means the standard error appears below its coefficient. For example, the AR(1) estimate for `L1.` is 0.876, with a standard error of 0.050.

1. Check the `N` row. Both columns show 118, confirming that the two models use the same number of observations.
2. Read across the lag rows. The `L1.` coefficient falls from 0.876 to 0.692 when the second lag enters the model. The `L2.` entry is blank for AR(1) because that model includes only one lag. For AR(2), the entry is 0.223 with a standard error of 0.094.
3. Compare the `rmse` row. This is the residual standard error, labeled `Root MSE` in the individual regression outputs. It falls from 4.770 to 4.677 sales index points, a modest improvement in fit over the estimation period. Forecast accuracy will be evaluated separately using the 2025 observations.

6.7 Checking the Residuals for Serial Correlation

If the model captures the dependence between months, little correlation should remain in its residuals. Check the first five lags for each model.

Check the AR(1) residuals. There are 118 residuals for each model, giving reference bounds of approximately $\pm 1.96/\sqrt{118} = \pm 0.180$. Run the bounds calculation and the plotting commands in the same do-file run so Stata can use the local values.

```
quietly count if !missing(resid_ar1)
local bound1 = 1.96 / sqrt(r(N))

corrgram resid_ar1 if train, lags(5)
```

In the `corrgram` table, read the `AC` and `PAC` columns just as you did for the sales index. This time, the entries describe the model's errors. For example, the lag-1 `AC` is -0.1775. Its negative sign means that a month in which the model underpredicts sales tends to be followed by a month in which it overpredicts, and vice versa.

Use `ac` and `pac` to plot the two columns. Give each graph its own name so that Stata can combine them later.

```
ac resid_ar1 if train, lags(5) ///
    ciopts(fcolor(none) lcolor(none)) ///
    yline(0, lcolor(gs10)) ///
    yline(-`bound1' `bound1', lcolor(gs6) lpattern(dash)) ///
    xlabel(1(1)5) ///
    ylabel(-1(.5)1) ///
    xtitle("Lag (months)") ///
    ytitle("Autocorrelation") ///
    title("AR(1): Residual ACF") ///
    note("Dashed lines: +/- 1.96/sqrt(T)") ///
    name(ar1_resid_acf, replace)

pac resid_ar1 if train, lags(5) ///
    ciopts(fcolor(none) lcolor(none)) ///
    yline(0, lcolor(gs10)) ///
    yline(-`bound1' `bound1', lcolor(gs6) lpattern(dash)) ///
    xlabel(1(1)5) ///
    ylabel(-1(.5)1) ///
    xtitle("Lag (months)") ///
    ytitle("Partial autocorrelation") ///
    title("AR(1): Residual PACF") ///
    note("Dashed lines: +/- 1.96/sqrt(T)") ///
    name(ar1_resid_pacf, replace)
```

Check the AR(2) residuals. Repeat the same steps using `resid_ar2`. Keeping the same five lags and axis scales makes it easier to see what changes after the second lag is added to the regression.

```
quietly count if !missing(resid_ar2)
local bound2 = 1.96 / sqrt(r(N))

corrgram resid_ar2 if train, lags(5)
```

```
ac resid_ar2 if train, lags(5) ///
    ciopts(fcolor(none) lcolor(none)) ///
    yline(0, lcolor(gs10)) ///
    yline(-`bound2' `bound2', lcolor(gs6) lpattern(dash)) ///
    xlabel(1(1)5) ///
    ylabel(-1(.5)1) ///
    xtitle("Lag (months)") ///
    ytitle("Autocorrelation") ///
    title("AR(2): Residual ACF") ///
    note("Dashed lines: +/- 1.96/sqrt(T)") ///
    name(ar2_resid_acf, replace)

pac resid_ar2 if train, lags(5) ///
    ciopts(fcolor(none) lcolor(none)) ///
    yline(0, lcolor(gs10)) ///
    yline(-`bound2' `bound2', lcolor(gs6) lpattern(dash)) ///
    xlabel(1(1)5) ///
    ylabel(-1(.5)1) ///
    xtitle("Lag (months)") ///
    ytitle("Partial autocorrelation") ///
    title("AR(2): Residual PACF") ///
    note("Dashed lines: +/- 1.96/sqrt(T)") ///
    name(ar2_resid_pacf, replace)
```

Compare the four plots. Combine the four graphs in two columns. The order below puts AR(1) above AR(2), and `ycommon` gives them the same vertical scale.

```
graph combine ///
    ar1_resid_acf ar1_resid_pacf ///
    ar2_resid_acf ar2_resid_pacf, ///
    cols(2) ycommon ///
    name(residual_diagnostics, replace)

graph export "residual_diagnostics.png", replace width(2400)
```

In Figure 20, compare the panels vertically, starting with lag 1 in the left column. The AR(1) residual ACF is -0.1775, just inside the lower reference line. Directly below it, the AR(2) value is 0.0010, almost exactly zero. Adding the second lag has largely removed the month-to-month correlation visible in the AR(1) residuals.

In the right column, the AR(1) partial autocorrelation at lag 3 (0.1965) lies just above the upper line. The AR(2) value (0.1517) falls inside. All five AR(2) lags lie within the bounds in both panels. When a dot is close to a line, check its value in the `corrgram` table.

6.8 Forecasting the Sales Index for 2025

Imagine standing at the end of December 2024 with sales data through that month. This is the *forecast origin*. From here, use the fitted AR(2) model to forecast all 12 months of 2025.

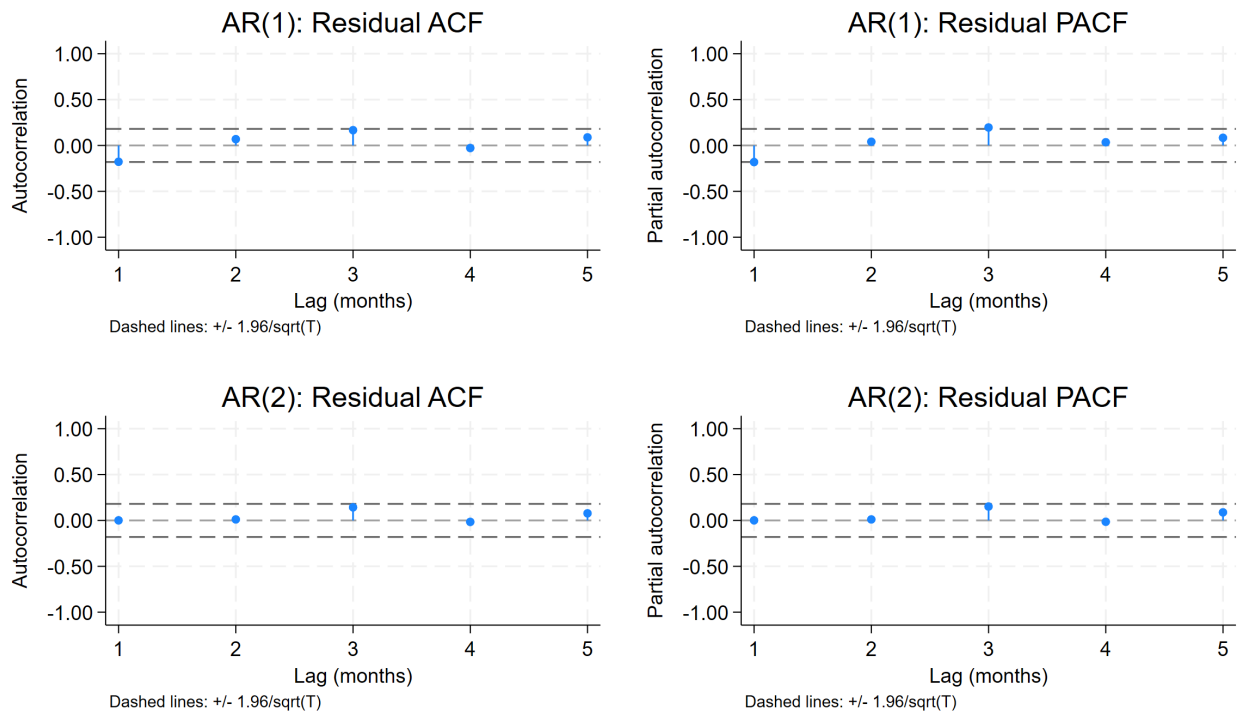


Figure 20: Residual ACF (left) and PACF (right) for the AR(1) model (top) and the AR(2) model (bottom). Dashed lines mark the approximate reference bounds of ±0.180.

Forecast January and February. Restore the stored AR(2) regression and save its three estimated coefficients. In Stata, a `scalar` stores a single number. Here, `b0` holds the intercept, while `b1` and `b2` hold the coefficients on the first and second lags.

```
estimates restore AR2

scalar b0 = _b[_cons]
scalar b1 = _b[L.sales_index]
scalar b2 = _b[L2.sales_index]
```

Start with the last two observed values, December and November 2024. Each command selects one month, so its mean, `r(mean)`, is that month's index.

```
quietly summarize sales_index if mdate == tm(2024m12), meanonly
scalar y_T = r(mean)

quietly summarize sales_index if mdate == tm(2024m11), meanonly
scalar y_Tminus1 = r(mean)
```

January's forecast uses observed sales from December and November. For February, use the January *forecast* and the observed December index.

```
scalar forecast_h1 = b0 + b1*y_T + b2*y_Tminus1
scalar forecast_h2 = b0 + b1*forecast_h1 + b2*y_T

display "January 2025 forecast:  " %9.2f forecast_h1
display "February 2025 forecast: " %9.2f forecast_h2
```

You should obtain **80.84** for January and **81.78** for February. For March, both inputs will be forecasts: February's and January's. Using earlier forecasts as inputs for later ones is called *recursive forecasting*.

Generate all 12 forecasts. The loop advances one month at a time. It uses `previous1` and `previous2` for the two most recent inputs, stores the forecast in that month's row, then shifts the inputs forward.

```
scalar previous1 = y_T
scalar previous2 = y_Tminus1

generate double forecast_ar2 = .

forvalues h = 1/12 {
    scalar next_forecast = ///
        b0 + b1*previous1 + b2*previous2

    replace forecast_ar2 = next_forecast ///
        if mdate == tm(2024m12) + `h'

    scalar previous2 = previous1
    scalar previous1 = next_forecast
}
```

List the observed and forecast values side by side. Check that the first two entries in `forecast_ar2` match the January and February forecasts you calculated above.

```
format sales_index forecast_ar2 %9.2f

list mdate sales_index forecast_ar2 ///
    if inrange(mdate, tm(2025m1), tm(2025m12)), noobs
```

Compare the forecasts with observed sales. Use `twoway` to draw both series on the same axes. The first `line` plots observed sales from January 2023 onward so you can see the series leading up to the forecast origin. The second plots the 12 forecasts. The vertical line marks December 2024.

```
twoway ///
    (line sales_index mdate ///
        if mdate >= tm(2023m1), lcolor(navy)) ///
    (line forecast_ar2 mdate ///
        if inrange(mdate, tm(2025m1), tm(2025m12)), ///
        lcolor(maroon) lpattern(dash)), ///
    xline(`=tm(2024m12)', lcolor(gs8) lpattern(dash)) ///
```

```
    title("AR(2) Forecasts and Observed Sales") ///
    subtitle("Forecast origin: December 2024") ///
    xtitle("Month") ///
    ytitle("Sales index") ///
    legend(order(1 "Observed" 2 "Forecast")) ///
    name(forecast_comparison, replace)

graph export "forecast_comparison.png", replace width(1800)
```

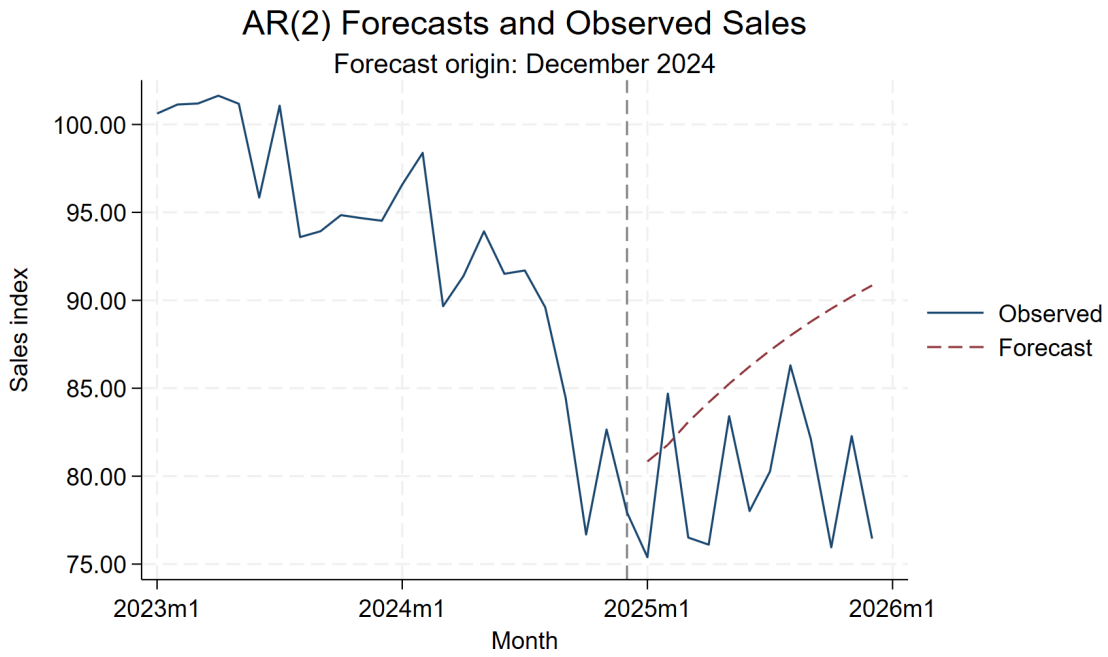


Figure 21: Observed monthly sales and the 12 recursive AR(2) forecasts made from December 2024.

In Figure 21, read to the right of the vertical line. January's forecast (80.84) exceeds the observed index (75.39). In February the observed index (84.70) is higher than the forecast (81.78). The forecast path rises smoothly as the model uses its own predictions. By December it reaches 90.85, while the observed index is 76.45.

6.9 Evaluating Forecast Accuracy

For each month of 2025, subtract the forecast from the observed index. A negative error means the forecast was too high. A positive error means it was too low.

```
generate double forecast_error = ///
    sales_index - forecast_ar2 ///
    if inrange(mdate, tm(2025m1), tm(2025m12))

generate double squared_error = forecast_error^2
generate double absolute_error = abs(forecast_error)
```

Open the Data Editor in **Browse** mode and scroll to January 2025. Figure 22 shows the 12 months that were held out of the estimation sample. The last three columns contain the signed error, its square, and its absolute value.

	month	sales_index	mdate	train	_est_AR1	resid_ar1	_est_AR2	resid_ar2	forecast_ar2	forecast_e...	squared_e...	absolute_e...
21	2025-01	75.39	2025m1	0	0	.	0	.	80.84	-5.4413925	29.608752	5.4413925
22	2025-02	84.70	2025m2	0	0	.	0	.	81.78	2.9110624	8.4742842	2.9110624
23	2025-03	76.51	2025m3	0	0	.	0	.	83.09	-6.5758576	43.241904	6.5758576
24	2025-04	76.11	2025m4	0	0	.	0	.	84.20	-8.0902401	65.451985	8.0902401
25	2025-05	83.41	2025m5	0	0	.	0	.	85.25	-1.8451985	3.4047573	1.8451985
26	2025-06	78.02	2025m6	0	0	.	0	.	86.23	-8.2168267	67.516241	8.2168267
27	2025-07	80.27	2025m7	0	0	.	0	.	87.15	-6.8785405	47.314319	6.8785405
28	2025-08	86.30	2025m8	0	0	.	0	.	88.00	-1.698921	2.8863327	1.698921
29	2025-09	82.11	2025m9	0	0	.	0	.	88.79	-6.677131	44.584078	6.677131
30	2025-10	75.96	2025m10	0	0	.	0	.	89.53	-13.567688	184.08216	13.567688
31	2025-11	82.28	2025m11	0	0	.	0	.	90.21	-7.9330989	62.934058	7.9330989
32	2025-12	76.45	2025m12	0	0	.	0	.	90.85	-14.402238	207.42445	14.402238

Figure 22: Observed sales, AR(2) forecasts, and forecast errors for January through December 2025.

In January, the observed index is 75.39 and the forecast is 80.84. Using the unrounded values, Stata gives a signed error of −5.44 points and an absolute error of 5.44 points. February's signed error is positive (2.91) because sales exceeded the forecast.

Now summarize the errors across all 12 months. Each `summarize` command stores the mean of the selected variable in `r(mean)`. The `display` commands print the results to three decimal places.

```
quietly summarize forecast_error, meanonly
display "Mean prediction error: " %9.3f r(mean)

quietly summarize squared_error, meanonly
display "MSPE:  " %9.3f r(mean)
display "RMSPE: " %9.3f sqrt(r(mean))

quietly summarize absolute_error, meanonly
display "Mean absolute error: " %9.3f r(mean)
```

Read the four results in Figure 23 alongside the monthly errors.

- **Mean prediction error** is −6.535 index points. The forecasts were too high by about 6.54 points on average. This agrees with the monthly results: February is the only month with a positive error.
- **Mean squared prediction error (MSPE)** is 63.910 squared index points. Squaring gives larger misses more weight. For example, December's error of about −14.40 contributes 207.42 to the sum of squared errors, while February's error of about 2.91 contributes 8.47.
- **Root mean squared prediction error (RMSPE)** is $\sqrt{63.910} \approx 7.994$ index points. Taking the square root expresses the result in the same units as the sales index.
- **Mean absolute error** is 7.020 index points. Counting misses in either direction, the forecast was about 7.02 points away from the observed index on average.

The regression's Root MSE of 4.677 describes the fit over the 118 estimation months. The RMSPE of

```
.
. // Mean prediction error: indicates the direction of errors.
. quietly summarize forecast_error, meanonly

. display "Mean prediction error: " %9.3f r(mean)
Mean prediction error:    -6.535

.
. // Mean squared prediction error and its square root.
. quietly summarize squared_error, meanonly

. display "MSPE:  " %9.3f r(mean)
MSPE:     63.910

. display "RMSPE: " %9.3f sqrt(r(mean))
RMSPE:     7.994

.
. // Mean absolute error.
. quietly summarize absolute_error, meanonly

. display "Mean absolute error: " %9.3f r(mean)
Mean absolute error:     7.020
```

Figure 23: Stata output summarizing the 12 monthly forecast errors.

7.994 describes the 12 held-out months. Because all forecasts were made in December 2024, the latter figure covers horizons from one to twelve months ahead.

Close the log to finish the exercise. You can find the commands and their text output in `manual_example.log` in your working folder.

```
log close
```



Happy forecasting!



Now it's your turn. Pick a question you're curious about and see what you can learn from the data.

If you spot a typo, a confusing step, or a little gremlin in the code, please let me know. Tell me where you found it so I can fix it for the next reader.

