fin — Commands for analysis of financial data[+]

[+]These features are part of StataNow.

Description Remarks and examples Also see

Description

This entry provides an overview of the commands for analyzing financial data and an example of a workflow that uses these commands. We demonstrate the use of the commands described in this manual, as well as commands available for analyzing time-series data generally, in a financial context.

Here is a list of the manual entries that provide useful additional information.

Financial statistics commands

[FIN] **finportfolio**	Financial portfolio selection[+]
[FIN] **finregress capm**	Capital asset pricing model (CAPM)[+]
[FIN] **finregress capm postestimation**	Postestimation tools for finregress capm[+]
[FIN] **finregress fmb**	Fama–MacBeth regression[+]
[FIN] **finregress fmb postestimation**	Postestimation tools for finregress fmb[+]
[FIN] **finreturns**	Generate financial returns[+]
[FIN] **finsummarize**	Financial summary statistics[+]
[FIN] **finvalrisk**	Value at risk[+]

Date and time management

[TS] **tsset**	Declare data to be time-series data
[D] **bcal**	Business calendar file manipulation[+]
[D] **Datetime business calendars**	Business calendars
[D] **Datetime business calendars creation**	Business calendars creation

Unit-root tests

[TS] **dfgls**	DF-GLS unit-root test
[TS] **dfuller**	Augmented Dickey–Fuller unit-root test
[TS] **pperron**	Phillips–Perron unit-root test
[XT] **xtunitroot**	Panel-data unit-root tests

Time-series smoothers

[TS] **tsfilter**	Filter a time series for cyclical components
[TS] **tssmooth**	Smooth and forecast univariate time-series data

Factor models and PCA

[TS] **dfactor**	Dynamic-factor models
[MV] **factor**	Factor analysis
[MV] **pca**	Principal component analysis

Time-series commands

[TS] **arch**	Autoregressive conditional heteroskedasticity (ARCH) family of estimators
[TS] **arima**	ARIMA, ARMAX, and other dynamic regression models
[TS] **arimasoc**	Obtain lag-order selection statistics for ARMAs
[TS] **ivlpirf**	Instrumental-variables local-projection impulse–response functions
[TS] **lpirf**	Local-projection impulse–response functions
[TS] **mgarch**	Multivariate GARCH models
[TS] **mgarch ccc**	Constant conditional correlation multivariate GARCH model
[TS] **mgarch dcc**	Dynamic conditional correlation multivariate GARCH model
[TS] **mgarch dvech**	Diagonal vech multivariate GARCH model
[TS] **mgarch vcc**	Varying conditional correlation multivariate GARCH model
[TS] **mswitch**	Markov-switching regression models
[TS] **newey**	Regression with Newey–West standard errors
[TS] **threshold**	Threshold regression
[TS] **var**	Vector autoregressive models
[TS] **var svar**	Structural vector autoregressive models
[TS] **var ivsvar**	Instrumental-variables structural vector autoregressive models
[TS] **vec**	Vector error-correction models

Panel-data commands

[XT] **xtreg**	Linear models for panel data+
[XT] **xtivreg**	Instrumental variables and two-stage least squares for panel-data models
[XT] **xtvar**	Panel-data vector autoregressive models

Remarks and examples

Remarks are presented under the following headings:

Returns and data management

Throughout this introduction, we use a running example consisting of data on the stock prices of 25 fictional companies, along with data on a stock-market index and a risk-free rate of return. Our data consist of 780 monthly observations containing the end-of-month values for these stock prices.

Let's open the data and describe the first few variables.

```
. use https://www.stata-press.com/data/r19/finex
(Fictional stock price data)
. describe datestr-wgt
```

Variable name	Storage type	Display format	Value label	Variable label
datestr	str11	%11s		String date
datem	int	%tm		Monthly date
sp500	double	%10.0g		S&P 500
vol	float	%9.0g		Volatility index
fedfunds	float	%9.0g		Federal funds rate
acme	float	%9.0g		Aciron Medical, Inc.
bat	float	%9.0g		Boron Advanced Technologies
iron	float	%9.0g		Industrial Operations Network
dune	float	%9.0g		Digital Urban Network Enterprise
tyr	float	%9.0g		Tyndale Resources Group
glo	float	%9.0g		Green Logistics, Inc.
spa	float	%9.0g		Space Rocket MFG
wgt	float	%9.0g		Widget Gadgets

The first two variables contain string and numeric dates. The latter is used to tell Stata about the time-series structure of the data. The third variable, `sp500`, contains a stock-market index. The variable `fedfunds` contains the annualized federal funds rate, which we take as the risk-free rate of return. The remaining variables are the stock prices of fictional firms.

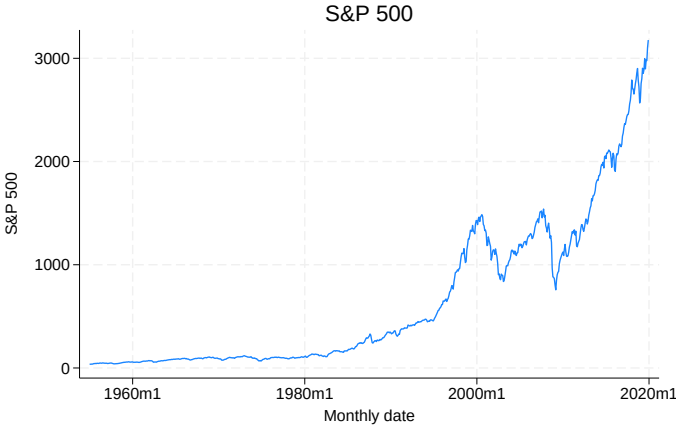
The `tsreport` command confirms that the dataset has been preset with a time-series structure and provides some information about the date variable. See [TS] **tsreport** for more on understanding the structure of your time-series data.

```
. tsreport
Time variable: datem
-----------------------------
Starting period =  1955m1
Ending period   = 2019m12
Number of obs   =     780
Number of gaps  =       0
```

The monthly date variable ranges from 1955 through 2019 without gaps, for a total of 780 observations. We are starting with a clean dataset, with minimal data processing needed and all the time-series issues (gaps, missing values, etc.) already resolved. The data you use may not come in so neat a format, necessitating data management. In addition, there are special considerations needed for using daily data. We will address some of these considerations in *Date and time management* below.

We begin our analysis by looking at one of the series, say, the stock-market index. We use the `tsline` command as follows:

```
. tsline sp500, title("S&P 500")
```



The index rises over time, albeit with periodic substantial declines. One of our goals is to analyze the behavior of the market price. A second goal is to understand how the behavior of individual assets and portfolios of assets are related to the behavior of the overall market.

The return on an asset price is a measure of how the asset price changes over time. There are various types of returns. The most fundamental is the simple return, defined as the proportional change in the asset price each period. Other concepts of return include logarithmic returns, annualized returns, and seasonalized returns.

The `finreturns` command computes returns for one or more variables. To compute the simple returns for the stock-market index, we type

```
. finreturns sp500, simple(rmkt) multiply(100)
(all returns multiplied by 100)
Simple returns generated for variable sp500:
```

	datem	sp500	rmkt
1.	1955m1	35.6	.
2.	1955m2	36.79	3.3426966
3.	1955m3	36.5	-.78825768
4.	1955m4	37.76	3.4520548
5.	1955m5	37.6	-.42372881
6.	1955m6	39.78	5.7978723
7.	1955m7	42.69	7.3152338
8.	1955m8	42.43	-.60904193
9.	1955m9	44.34	4.5015319
10.	1955m10	42.11	-5.0293189

We specify that we want to compute returns for `sp500`. The `simple()` option specifies that we want the simple return, and the `rmkt` variable is a new variable containing the return we calculated. The `multiply()` option multiplies the result by the number specified, in this case, 100, so that the result can be interpreted as percentage change from the previous month. A preview of the computed returns

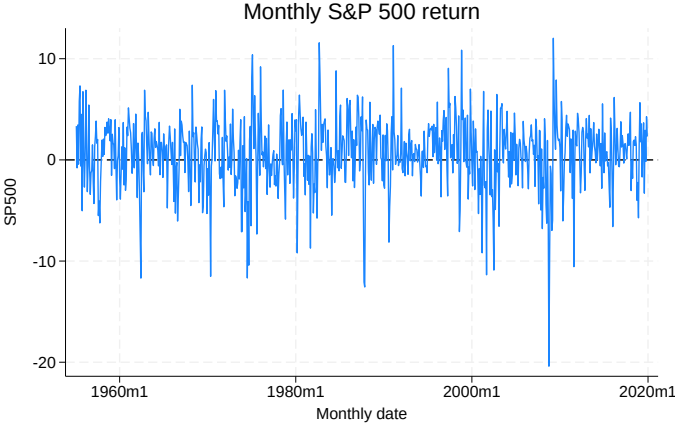
is shown, indicating the date, the original variable, and the new return variable. In observation 2, for example, the simple return 3.34 is equal to $100 \times (36.79/35.6 - 1)$ and indicates a 3.34% increase over the previous month.

```
. summarize rmkt
```

Variable	Obs	Mean	Std. dev.	Min	Max
rmkt	779	.6387889	3.458349	-20.39114	12.02171

The average monthly return is 0.64%, slightly more than one-half of a percent. The standard deviation of these returns is large, about 3.46, with some months being disasters (a minimum monthly return of -20%) and some months showing strong positive returns (a maximum one-month gain of 12%). We can graph the monthly returns:

```
. tsline rmkt, yline(0) title("Monthly S&P 500 return")
```



The pattern of returns shows a mild degree of persistence: months with higher than typical returns are followed by months that also have higher than typical returns. More visually apparent is the clustering of volatility. There are periods where the variance of returns expands and periods of relative calm where the variance is lower. The question of whether financial time series are predictable and the question of characterizing their volatility have been longstanding topics for financial statistics. We explore the predictability of returns and the nature of the time-varying volatility below.

In addition to the market return, we are also interested in the returns on the stock prices of our collection of firms. We use `finreturns` again, this time with the full set of stock prices. Instead of typing all 25 firm names, we use the varlist range notation *varname1*`-`*varnameK* to specify every variable between the two named variables in the dataset, inclusive.

```
. finreturns acme-tks, simple(r_) multiply(100) nopreview
(all returns multiplied by 100)

Simple returns generated for variables acme, bat, iron, dune, tyr, glo,
spa, wgt, bar, yum, aaa, afh, ard, cph, das, dil, ege, epg, goa, jml, khc,
krg, kth, nhb, and tks.
```

Three aspects of the results are worth noting. First, because there was more than one asset specified, the name `r_` is interpreted as a stub; the resulting new variables are `r_acme`, `r_bat`, and so on, up to `r_tks`. The return variables are stored in order in the dataset, so it is easy to use the varlist range notation in subsequent commands. Second, we specified `nopreview` to suppress the preview of the results. By

default, a preview of the returns for all 25 assets would be displayed. Third, if we `describe` the returns, we see that each return has been labeled with the name of the variable it was derived from in capital letters.

```
. describe r_acme-r_tyr
Variable      Storage   Display    Value
    name         type    format    label      Variable label
------------------------------------------------------------------------------
r_acme          double  %10.0g                ACME
r_bat           double  %10.0g                BAT
r_iron          double  %10.0g                IRON
r_dune          double  %10.0g                DUNE
r_tyr           double  %10.0g                TYR
```

These automatic labels make it easy to keep track of how a given return was generated.

We can summarize the characteristics of the returns using the `finsummarize` command. To save space, we summarize only the returns for the first five assets:

```
. finsummarize r_acme-r_tyr
Financial summary statistics
Number of obs = 779
Sample: 1955m2 thru 2019m12
```

	Mean	Std. dev.	Sharpe ratio
ACME	0.4517	1.1856	0.3810
BAT	0.9052	5.4073	0.1674
IRON	1.0214	6.5625	0.1556
DUNE	1.0593	6.6088	0.1603
TYR	0.7028	3.9642	0.1773

By default, the mean, standard deviation, and Sharpe ratio of each return is displayed. The Sharpe ratio is the ratio of the mean (possibly adjusted for a risk-free rate) over the standard deviation. Higher values indicate assets with higher returns, lower volatility, or a combination of the two. As before, these are monthly returns in percentages. The mean return varies across assets, from 0.45% to 1.06%, with a ratio of about 2. One goal of asset pricing is to understand the variation in average returns and see whether any other variables can predict which kinds of assets will show high returns. Also notice that the standard deviation seems to rise as mean returns rise.

Portfolio construction

Portfolios are linear combinations of assets. Assets can be combined to reduce variability by diversifying one's investments. The `finportfolio` command produces portfolios based on various criteria. In this section, we build 4 portfolios from the 25 asset returns.

The first portfolio we create will be the simplest one: an equally weighted portfolio of all 25 stocks. This is accomplished with `finportfolio equal`. We specify the `generate()` option to generate the `p1` variable with the portfolio returns. We will use this portfolio return variable later, comparing its performance with the individual stocks and with other portfolios.

```
. finportfolio equal r_acme-r_tks, generate(p1, label("Equal"))
Equally weighted portfolio
Number of obs = 779
Sample: 1955m2 thru 2019m12
```

	Weight
ACME	.04
BAT	.04
IRON	.04
DUNE	.04
TYR	.04
GLO	.04
SPA	.04
WGT	.04
BAR	.04
YUM	.04
AAA	.04
AFH	.04
ARD	.04
CPH	.04
DAS	.04
DIL	.04
EGE	.04
EPG	.04
GOA	.04
JML	.04
KHC	.04
KRG	.04
KTH	.04
NHB	.04
TKS	.04

```
Portfolio return    = 0.6697
Portfolio std. dev. = 2.7483
Risk-free rate      = 0.0000
Sharpe ratio        = 0.2437
```

The main table provides the weights associated with each return. The equally weighted portfolio does not have much going on; each weight is 1/25, equal to a 4% share of the portfolio. Below the table of weights is some information about the portfolio performance. We have the portfolio mean and standard deviation; if a risk-free rate was specified, its return is also displayed. Then the Sharpe ratio is displayed. We see that the equally weighted portfolio returns 0.67% per month with a standard deviation of 2.75%.

Now, a savvy investor might not be satisfied with an equally weighted portfolio. Instead, an investor might want to craft a portfolio that met certain criteria, such as the lowest overall possible variance or the lowest variance while still hitting a target return or maximizing the reward-to-risk ratio (the Sharpe ratio). The minimum variance portfolio chooses the combination of weights that delivers the lowest variance. The minimum variance portfolio can be chosen to meet the global minimum or to minimize variance for a specified, target return. Let's specify `target(0.67)` to match the monthly return on the equally weighted portfolio and specify `generate(p2)` to store this new portfolio's return in a new variable.

```
. finportfolio minvariance r_acme-r_tks, target(0.67)
> generate(p2, label("MinVar"))
Minimum variance portfolio with fixed return
Short selling allowed
Number of obs = 779
Sample: 1955m2 thru 2019m12
```

	Weight
ACME	.0234078
BAT	.1227878
IRON	.0562631
DUNE	.145313
TYR	-.2108211
GLO	.0593816
SPA	.1825036
WGT	-.0500623
BAR	-.0048355
YUM	.0970868
AAA	.1528412
AFH	-.067757
ARD	.0436591
CPH	-.1453048
DAS	.0166238
DIL	.1613341
EGE	-.112362
EPG	.0370243
GOA	.5820644
JML	-.0728716
KHC	.0665349
KRG	-.1612624
KTH	-.081871
NHB	.0315904
TKS	.1287319

```
Portfolio return    = 0.6700 (fixed return)
Portfolio std. dev. = 1.0311
Risk-free rate      = 0.0000
Sharpe ratio        = 0.6498
```

The asset weights now vary in the combination that delivers the lowest portfolio variance for the specified return. Some of the weights are negative, implying a short position in those assets. Below the weight table, we see that the minimum variance portfolio delivers the same 0.67% monthly return as the equally weighted portfolio but does so with less than half of the standard deviation: 1.03% instead of 2.75%.

One might wish to construct portfolios that permit only positive weights on all assets; these are no-short portfolios. `finportfolio` allows this constraint with the `noshort` option. We run the same exercise, choosing the minimum variance portfolio subject to the restriction of no short sales.

```
. finportfolio minvariance r_acme-r_tks, target(0.67) noshort
> generate(p3, label("NoShort"))
Iteration 0:  Penalized variance =  7.5745874
Iteration 1:  Penalized variance =  6.0526077
Iteration 2:  Penalized variance =  5.7479174
Iteration 3:  Penalized variance =  5.4642297
Iteration 4:  Penalized variance =  5.2008879
Iteration 5:  Penalized variance =  4.9567827
Iteration 6:  Penalized variance =  4.7298329
Iteration 7:  Penalized variance =  4.3146947
Iteration 8:  Penalized variance =  3.9610262
Iteration 9:  Penalized variance =  3.6530792
  (iteration log omitted)
Iteration 15: Penalized variance =  2.0569964
Iteration 16: Penalized variance =   1.821173
Iteration 17: Penalized variance =  1.6890968
Iteration 18: Penalized variance =  1.6151142
Iteration 19: Penalized variance =   1.544494
Iteration 20: Penalized variance =    1.52092
Iteration 21: Penalized variance =  1.5208523
Iteration 22: Penalized variance =  1.5208441
Iteration 23: Penalized variance =  1.5208416
Iteration 24: Penalized variance =  1.5208415

Minimum variance portfolio with fixed return
Short selling not allowed

Number of obs = 779
Sample: 1955m2 thru 2019m12
```

	Weight
ACME	.0001638
BAT	.0443116
IRON	.0007996
DUNE	.1363128
TYR	.0000514
GLO	.0001381
SPA	.0572409
WGT	.000116
BAR	.0001778
YUM	.0254636
AAA	.0390746
AFH	.0000705
ARD	.0104981
CPH	.0000717
DAS	.0001209
DIL	.0863905
EGE	.0000611
EPG	.0001493
GOA	.5966434
JML	.0000725
KHC	.0005557
KRG	.0000507
KTH	.0000748
NHB	.0005682
TKS	.0008225

```
Portfolio return    = 0.6700 (fixed return)
Portfolio std. dev. = 1.2285
Risk-free rate      = 0.0000
Sharpe ratio        = 0.5454
```

One new aspect of the output is the iteration log. Minimizing variance subject to a no-short constraint is a nonlinear problem that requires an iterative approach to solve. The iteration log displays progress toward the goal, minimum variance. We can see that, at each step, the variance of the portfolio is reduced. The process stops when the weights are found that deliver as small a variance as possible, subject to the constraints. The resulting weights are all positive, as intended. By forcing all weights to be positive, we see that relative weights within the portfolio have shifted somewhat. Notice also that the no-short requirement has increased the portfolio standard deviation somewhat, from 1.03% to 1.23%.

As a fourth portfolio, we construct the portfolio with the highest ratio of return to standard deviation. This is the maximum Sharpe ratio portfolio. It does not have a target return; instead it chooses the target return to deliver the best return-to-risk ratio.

```
. finportfolio maxsharpe r_acme-r_tks, noshort gamma(0.001)
> generate(p4, label("MaxSharpe"))
Iteration 0:  Penalized Sharpe ratio =  .16318784  (not concave)
Iteration 1:  Penalized Sharpe ratio =  .19263253  (not concave)
Iteration 2:  Penalized Sharpe ratio =   .2746519  (not concave)
Iteration 3:  Penalized Sharpe ratio =  .32067251  (not concave)
Iteration 4:  Penalized Sharpe ratio =  .33244638  (not concave)
Iteration 5:  Penalized Sharpe ratio =   .3513766  (not concave)
Iteration 6:  Penalized Sharpe ratio =  .38394785  (not concave)
Iteration 7:  Penalized Sharpe ratio =  .44470668  (not concave)
Iteration 8:  Penalized Sharpe ratio =  .51944305  (not concave)
Iteration 9:  Penalized Sharpe ratio =  .55928097  (not concave)
Iteration 10: Penalized Sharpe ratio =  .57315047  (not concave)
Iteration 11: Penalized Sharpe ratio =  .58966733  (not concave)
Iteration 12: Penalized Sharpe ratio =   .5922082  (not concave)
Iteration 13: Penalized Sharpe ratio =  .59788206  (not concave)
Iteration 14: Penalized Sharpe ratio =  .64981786
Iteration 15: Penalized Sharpe ratio =  .71444962
Iteration 16: Penalized Sharpe ratio =  .74339013
Iteration 17: Penalized Sharpe ratio =  .75825625
Iteration 18: Penalized Sharpe ratio =  .76269832
Iteration 19: Penalized Sharpe ratio =  .76439582
Iteration 20: Penalized Sharpe ratio =  .76507737
Iteration 21: Penalized Sharpe ratio =  .76516526
Iteration 22: Penalized Sharpe ratio =  .76516566
Iteration 23: Penalized Sharpe ratio =  .76516566
```

```
Maximum Sharpe ratio portfolio
Short selling not allowed
Number of obs = 779
Sample: 1955m2 thru 2019m12
```

	Weight
ACME	.1005565
BAT	.0016509
IRON	.0010381
DUNE	.0011628
TYR	.0018101
GLO	.0079593
SPA	.1879825
WGT	.0034215
BAR	.0045021
YUM	.0206193
AAA	.0575591
AFH	.0045532
ARD	.1706966
CPH	.0323272
DAS	.0153331
DIL	.0069934
EGE	.0015328
EPG	.0016666
GOA	.281921
JML	.0176177
KHC	.0020375
KRG	.0015561
KTH	.0033547
NHB	.0681455
TKS	.0040023

```
Portfolio return    = 0.4817
Portfolio std. dev. = 0.5458
Risk-free rate      = 0.0000
Sharpe ratio        = 0.8826
```

The Sharpe ratio jumps from 0.55 in the previous minimum variance portfolio to 0.88 in the maximum Sharpe ratio portfolio. This return-to-risk ratio is achieved with a lower monthly return, 0.48% instead of 0.67%, but a much lower standard deviation, 0.55% versus 1.03% to 2.75% in the case of the minimum variance and equally weighted portfolios, respectively.

Portfolio summaries

Next we continue to explore the properties of the portfolios we have constructed. Again, we use `finsummarize`.

```
. finsummarize p*
Financial summary statistics
Number of obs = 779
Sample: 1955m2 thru 2019m12
```

	Mean	Std. dev.	Sharpe ratio
Equal	0.6697	2.7483	0.2437
MinVar	0.6700	1.0311	0.6498
NoShort	0.6700	1.2285	0.5454
MaxSharpe	0.4817	0.5458	0.8826

We use the notation `p*` to indicate all variables in the dataset that begin with `p` and have any additional text after this first letter. The characteristics of the portfolios are displayed, repeating what we have already seen from `finportfolio` but now in a more easy-to-read format. Adding the S&P 500 market return `rmkt` in the `benchmark()` option allows `finsummarize` to compute more information.

```
. finsummarize p*, benchmark(rmkt)
Financial summary statistics
Number of obs = 779
Sample: 1955m2 thru 2019m12
```

	Mean	Std. dev.	Sharpe ratio	Treynor index	Jensen's alpha	beta
Equal	0.6697	2.7483	0.2437	0.8563	0.1701	0.7820
MinVar	0.6700	1.0311	0.6498	7.8706	0.6156	0.0851
NoShort	0.6700	1.2285	0.5454	4.9402	0.5834	0.1356
MaxSharpe	0.4817	0.5458	0.8826	12.0049	0.4561	0.0401

Note: Variable **rmkt** used as benchmark asset.

Adding a market index allows us to compute the Treynor index, Jensen's alpha, and market beta. The Treynor index, like the Sharpe ratio, is a return-to-risk measure. Jensen's alpha is the intercept in a regression of the asset return on the market return. Positive values of Jensen's alpha indicate returns to the portfolio that are attained over and above those returns derived from the asset's association with the market. Market beta is the covariance of the portfolio return with the market. Higher values indicate an asset or a portfolio that covaries more strongly with the market. The Treynor index is a function of the market beta; it divides the return by market beta. Higher values of Treynor's index indicate the existence of higher returns that are achieved without also exposing the investment to additional market risk.

The equally weighted portfolio has the highest standard deviation at 2.75%, the lowest Sharpe ratio at 0.24, and much higher exposure to market risk than the other portfolios with a beta of 0.78. Hence, the equally weighted portfolio moves closely with the market. By contrast, both of the minimum variance portfolios (`MinVar` and `NoShort`) attain a much lower standard deviation, at 1.03% and 1.23%, respectively. They thus attain the same monthly return at half the monthly volatility. Both minimum variance portfolios have relatively high alpha, of about 0.60% monthly. This may be interpreted as these assets

yielding about 0.60% returns per month in a way that is uncorrelated with market risk. Their market betas are quite low, 0.09 and 0.14, respectively, indicating that the portfolio weights in the minimum variance portfolios shield them from market risk.

The maximum Sharpe ratio portfolio has a lower standard deviation than even the minimum variance portfolios, at 0.55%. This is possible because the maximum Sharpe ratio portfolio trades off some average return: 0.48% per month instead of 0.67% per month. This portfolio is nearly unrelated to the market, with a beta of 0.04. Thus, the maximum Sharpe ratio portfolio trades off lower return for lower variance, maximizing the return-to-risk ratio. This portfolio also achieves its returns in a way that is almost totally insulated from market fluctuations.

Financial regressions

In constructing models for returns, log returns are a more convenient object than simple returns. Log returns approximate simple returns when returns are small, and they have the advantage of being symmetric. For example, a 100% increase in log return followed by a 100% decrease in log return brings the asset back to its initial value. This is not true for simple returns, where a 100% increase would have to be followed by a 50% decrease to land back at the initial value. Log returns can thus be added and averaged arithmetically. This is useful in a modeling setting, where the statistical properties of the model are understood as the behavior of averages of the time series.

We use `finreturns` again to create variables holding the log return on the market and the log return on the stocks we want to analyze. We multiply log returns by 100 so that they approximate percentage changes.

```
. finreturns sp500, log(lnr_mkt) multiply(100)
(all returns multiplied by 100)
Log returns generated for variable sp500:
```

	datem	sp500	lnr_mkt
1.	1955m1	35.6	.
2.	1955m2	36.79	3.2880431
3.	1955m3	36.5	-.79138085
4.	1955m4	37.76	3.3938081
5.	1955m5	37.6	-.42462909
6.	1955m6	39.78	5.6360223
7.	1955m7	42.69	7.0600427
8.	1955m8	42.43	-.61090416
9.	1955m9	44.34	4.4031545
10.	1955m10	42.11	-5.1601962

```
. finreturns acme-tks, log(lnr_) multiply(100) nopreview
(all returns multiplied by 100)
Log returns generated for variables acme, bat, iron, dune, tyr, glo, spa,
wgt, bar, yum, aaa, afh, ard, cph, das, dil, ege, epg, goa, jml, khc, krg,
kth, nhb, and tks.
```

The `lnr_mkt` variable now holds the log market return. We have 25 variables containing the log returns for each stock, `lnr_acme` through `lnr_tks`.

We will also need a monthly risk-free rate of return for our analysis. We generate it using double precision to achieve more accuracy when we use it for further computations. To construct it, we type

```
. generate double rf = 100 * ln(1+fedfunds/100)/12
```

This construction does two things. First, it converts the `fedfunds` interest rate, measured in annual percentage change, into a log return equivalent to the log returns used for the other assets. Second, dividing the log return by 12 gives a monthly equivalent to the annual return. The result is the short-term federal interest rate at a monthly frequency. We will use this risk-free rate to compute excess returns on our stocks, among other things.

Capital asset pricing model

The capital asset pricing model (CAPM) relates an asset's return to the market return, after adjusting for the risk-free rate. Apart from the market return, other characteristics or "factors" describing the economy, the market, the industry, and the asset itself can be included as independent variables in the model to serve as controls. The return of asset i at time t, net of the risk-free rate, is $r_{it} - r_t^f$. The market excess return is $r_t^m - r_t^f$. Without controls, the linear regression relating these quantities is

$$r_{it} - r_t^f = \alpha_i + \beta_i(r_t^m - r_t^f) + e_{it}$$

Each asset has its own asset-specific intercept α_i and slope coefficient β_i. Both the slope coefficient and the intercept have financial interpretation. The slope coefficient measures how much the asset's excess return is expected to rise when the market excess return rises by one unit. The intercept captures the average excess return of the asset when the market excess return is zero. Hence, α_i measures the average return on the asset that is independent of the market return. `finregress capm` fits this regression for a set of assets. We now use it to compute intercepts and slope coefficients for the first five log returns we obtained earlier.

```
. finregress capm lnr_acme-lnr_tyr = lnr_mkt, rfrate(rf) adjust
Capital asset pricing model
Sample: 1955m2 thru 2019m12                          Number of obs = 779
```

	Coefficient	Robust std. err.	z	P>|z|	[95% conf.	interval]
lnr_acme						
lnr_mkt	.1316993	.0102147	12.89	0.000	.111679	.1517197
_cons	.0322367	.0378174	0.85	0.394	-.041884	.1063575
lnr_bat						
lnr_mkt	1.530731	.0114377	133.83	0.000	1.508314	1.553148
_cons	.0760026	.0416309	1.83	0.068	-.0055925	.1575977
lnr_iron						
lnr_mkt	1.871524	.012273	152.49	0.000	1.847469	1.895579
_cons	.0556327	.0454103	1.23	0.221	-.0333698	.1446352
lnr_dune						
lnr_mkt	1.878887	.0123726	151.86	0.000	1.854637	1.903137
_cons	.0895071	.046235	1.94	0.053	-.0011119	.1801261
lnr_tyr						
lnr_mkt	1.087576	.0117232	92.77	0.000	1.064599	1.110553
_cons	.0282078	.0443328	0.64	0.525	-.0586828	.1150984

```
Notes: Dependent variables adjusted for risk-free rate rf.
       Independent variable lnr_mkt adjusted for risk-free rate rf.
```

The basic syntax is *asset returns* = *independent variables*. When a risk-free rate is specified in `rfrate()`, the asset returns on the left-hand side are adjusted by the risk-free rate. However, the independent variables (the factors) on the right-hand side are not. This is because the independent variables may already be excess returns or may be a factor that does not need to be adjusted. To indicate that an independent variable needs to be adjusted by the risk-free rate, specify the `adjust()` option.

The CAPM results here show variation in the slope coefficients and in the intercepts. These slope coefficients have the standard interpretation of regression coefficients. Because we have used log returns and adjusted for the risk-free rate, our dependent variables are excess log returns for the five assets we specified (ACME, BAT, IRON, DUNE, and TYR). Similarly, the independent variable is the excess log market return. We can interpret the estimates in these units. As the excess log market return rises by one unit, the expected excess log return for ACME rises by 0.13, so we expect excess log returns for ACME to rise less than those for the market. Expected excess log returns for BAT, IRON, and DUNE all rise more than one for one with excess log market returns. We expect excess log returns for these assets to be up (between 1.53 and 1.88). Finally, the expected excess log return for TYR has a slope coefficient of 1.09, so it moves approximately one for one with the excess log market return. These coefficients are also called an asset's market exposure.

Under a strict interpretation of the CAPM, all variation in asset returns should be driven by the independent variables. If this is true, then all the intercepts will be zero. A key test of the theory, then, is a test for whether all the intercepts are jointly zero. The Gibbons–Ross–Shanken test performs this test. We use `estat grstest` to perform the test and specify the `finite` option to obtain a finite-sample F test.

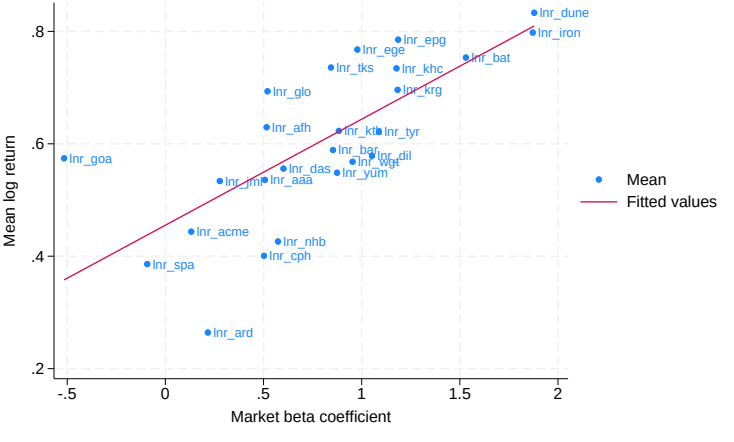
```
. estat grstest, finite
Gibbons-Ross-Shanken test
H0: All intercept terms are zero

  No. of dependent vars. =       5
No. of independent vars. =       1
     No. of time periods =     779
                 F(5, 773) =  1.321
                  Prob > F = 0.2532
```

For these five test assets, we cannot reject the null that the intercepts are jointly zero. The interpretation is that the specified factors, in this case the sole factor being the market return, adequately explain the pattern of average returns across assets.

Fama–MacBeth regression

While the time-series regressions above capture variation in returns over time, many questions in asset pricing revolve around the cross-section of average (or expected) returns. Why does one asset return more than another, consistently? One answer is that the higher-returning factor bears more risk, for which the asset holder is "paid" by also offering a higher average return. In the time-series regressions above, an asset's risk is measured by its covariance with the market. Riskier assets move with the market. By contrast, an asset that does not move with the market pays off precisely when the market does not, which is an advantage to a risk-adverse investor. A risk-adverse investor would hold highly risky assets only if those assets also paid higher average returns. This is the key behind the next regressions.

The Fama–MacBeth regression begins with the coefficients estimated in the time-series CAPM and then runs a second-stage regression of average returns on the first-stage coefficients. Visually, the regression being run is the following:



The intuition is that assets that are riskier (have higher beta) should also have higher returns. The slope of this line captures the rise in average return resulting from taking on one more "unit" of risk, where a unit of risk is measured by the responsiveness of the asset to the market.

But simply running this regression on the averages is unwise for two reasons. First, by taking averages, the regression understates the variability in the second-stage estimate. Second, the betas themselves are estimated, and a naive regression understates the variability of the second-stage estimate by not accounting for the variability of the first-stage estimates.

The Fama–MacBeth regression addresses the first problem by running this regression T times, once per time period, on the cross-section of returns. The final estimate is the average of the T cross-sectional estimates, and its variability is just the standard deviation of the cross-sectional estimates.

```
. finregress fmb lnr_acme-lnr_tks = lnr_mkt, rfrate(rf) adjust
Fama-MacBeth regression
Sample: 1955m2 thru 2019m12                          Number of obs     = 779
Method: OLS                                          Number of depvars =  25
```

depvar_means	Coefficient	Std. err.	z	P>|z|	[95% conf.	interval]
beta						
lnr_mkt	.1886384	.1270712	1.48	0.138	-.0604165	.4376934
_cons	.06874	.0159916	4.30	0.000	.0373971	.1000829

```
Dependent variables: lnr_acme lnr_bat lnr_iron lnr_dune lnr_tyr lnr_glo
                     lnr_spa lnr_wgt lnr_bar lnr_yum lnr_aaa ... lnr_tks
Notes: Dependent variables adjusted for risk-free rate rf.
       Independent variable lnr_mkt adjusted for risk-free rate rf.
```

We have log returns for 25 assets used as dependent variables in the first stage. For brevity, the command does not list all dependent variables below the table, but you can click on the dots to see the full list. Because we specified the `rfrate(rf)` and `adjust` options, the first-stage models regress excess log returns for these assets on excess log market return.

The first coefficient reported is the slope coefficient. It is equivalent to the slope of a regression of mean returns on beta risk. The dependent variable is the cross-section of mean returns, denoted here by `depvar_means`. The independent variable is beta risk of `lnr_mkt`, denoted as `lnr_mkt` in the `beta` equation, which was estimated in the first-stage regressions.

In our example, `depvar_means` refers to the average excess log returns of the listed dependent variables. The coefficient `lnr_mkt` captures the increase in expected average excess log return for accepting one more unit of beta risk. This slope is often called the price-of-risk coefficient. The coefficient `_cons` is the constant term. It can be interpreted as the average excess log return on an asset with zero beta risk.

The average price-of-risk coefficient for `lnr_mkt` is positive, indicating that as market beta risk rises, expected return rises. For a one-unit increase in beta risk for excess log market returns, the expected average log return of considered assets would rise by about 0.19. For relatively small return values, the average price-of-risk coefficient estimated based on log returns can approximate the coefficient based on simple returns. This approximation holds in our example and, more generally, with daily and monthly data. So we could equivalently state that for a one-unit increase in market beta risk for excess simple returns, we expect average excess simple returns to rise by about 0.19. Thus, as investors take on more risk, on average, their returns also rise. However, the p-value on the coefficient of `lnr_mkt` suggests that its effect is not different from zero. A coefficient of exactly zero would indicate that investors are not rewarded for taking on additional risk. A negative coefficient would indicate that as exposure to the factor rises, average returns fall, indicating a "negative" average return for exposure to that factor.

Value at risk

We now turn to risk management. Summary statistics can provide measures of the average performance of an asset or portfolio, such as the mean return, and can provide summary measures of dispersion, such as the standard deviation. However, standard deviation is just one measure of dispersion. Also useful to know are other characteristics of a return's distribution, such as the so-called value at risk (VaR).

VaR is a quantity representing the potential loss on the value of an asset or a portfolio over a specified length of time for a given confidence level. If the VaR for an asset is \$300 at a 95% confidence level at the one-month horizon, then there is a 5% chance that the asset will lose \$300 or more in the next month. We would say that the 5% one-month VaR for this asset is \$300. A different asset might have a 5% one-month VaR of \$200. In that case, the latter asset would have a 5% chance of a loss of \$200 or more in the next month. Larger values of VaR indicate that the asset has greater risk, because larger values indicate that, at the specified confidence level (usually 99% or 95%), the asset has a more extreme threshold of losses.

In *Portfolio construction*, we built four example portfolios: an equally weighted portfolio, a minimum variance portfolio, a minimum variance portfolio with only positive weights, and a portfolio that maximized the return-to-risk ratio. We now assess these portfolios based on their VaR by using the `finvalrisk` command.

```
. finvalrisk p*, percentiles(1)
Value-at-risk percentiles
Number of obs = 779
Sample: 1955m2 thru 2019m12
```

	Empirical	Normal
1%		
Equal	-8.405127	-5.723881
MinVar	-1.831169	-1.728795
NoShort	-2.272815	-2.187996
MaxSharpe	-.7692161	-.787994

The output shows percentiles for each portfolio corresponding to the 1% VaR or, equivalently, VaR at the 99% confidence level. The interpretation is that there is a 1% chance that the portfolio returns this amount or less in the next month. The percentiles reported here are negative returns, but the VaR is typically written as positive, representing the potential loss. The default output shows two methods of calculating VaR. The first method is the empirical method, which takes the appropriate quantile (in this case, the 1% quantile) of the empirical distribution of the data. We see that the equally weighted portfolio has a 1% VaR of 8.41%, so it has a 1% chance of having a monthly return of -8.41% or worse. Meanwhile, the minimum variance portfolio has a 1% chance of having a monthly return -1.83% or worse. The minimum variance portfolio without short selling, which is restricted to having positive weights, suffers somewhat from this restriction; its 1% VaR is 2.27%. Finally, the maximum Sharpe ratio portfolio has a 1% VaR of only 0.77%.

The second method uses a normal approximation fit to the returns. This approach models returns as being drawn independently from a normal distribution, where the mean and standard deviation of the normal distribution are equal to the mean and standard deviation of the observed return series. The VaRs estimated from the normal approximation here are mostly similar to the empirical VaRs, except for the equally weighted portfolio. Normal-based VaR for the equally weighted portfolio is 5.72%, whereas its empirical VaR is 8.41%, indicating that the equally weighted portfolio has especially fat tails and that the normal approximation may be underestimating VaR for this portfolio.

Different quantiles can be requested. Model-based VaR can also be computed, in which a time-series model is fit to the asset returns and the model-implied VaR is reported. Available models include the autoregressive moving-average model (ARMA), the exponentially weighted moving-average model, and the family of autoregressive conditional heteroskedasticity models. See [FIN] **finvalrisk** for all the features of this command.

Other time-series tools

In this section, we demonstrate the use of some of the broader time-series tools of Stata that are applicable to financial data. This is not an exhaustive compendium. See [TS] **Time series** for an overview of Stata's time-series capabilities. While all time-series commands may be useful for analyzing financial data, we focus here on some that are frequently used.

Modeling returns

We would like to investigate some of the time-series properties of the log market return. A longstanding question in financial research has been the degree of predictability of asset returns. The pure random walk model, for instance, proposes that returns cannot be forecasted from prior returns. We explore predictability here by fitting some simple ARMA models to the log market return.

The `arimasoc` command fits ARMA models with various combinations of autoregressive and moving-average components to find a specification that describes the dynamics well. To explore specifications for the log market returns with at most 6 autoregressive lags and at most 1 moving-average lag, we type

```
. arimasoc lnr_mkt, maxar(6) maxma(1)
Fitting models (14): .............. done
Lag-order selection criteria
Sample: 1955m2 thru 2019m12                     Number of obs = 779
```

Model	LL	df	AIC	BIC	HQIC
ARMA(0,0)	-2079.026	2	4162.052	4171.368	4165.635
ARMA(0,1)	-2054.852	3	4115.703	4129.677	4121.078
ARMA(1,0)	-2056.289	3	4118.578	4132.552	4123.953
ARMA(1,1)	-2054.838	4	4117.677	4136.309	4124.843
ARMA(2,0)	-2055.202	4	4118.405	4137.037	4125.571
ARMA(2,1)	-2053.431	5	4116.863	4140.153	4125.821
ARMA(3,0)	-2054.478	5	4118.956	4142.246	4127.915
ARMA(3,1)	-2053.852	6	4119.705	4147.653	4130.455
ARMA(4,0)	-2053.801	6	4119.603	4147.551	4130.353
ARMA(4,1)	-2053.459	7	4120.917	4153.523	4133.459
ARMA(5,0)	-2051.501	7	4117.002	4149.608	4129.543
ARMA(5,1)	-2048.608	8	4113.215	4150.479	4127.548
ARMA(6,0)	-2044.992	8	4105.983	4143.247	4120.316
ARMA(6,1)	-2044.971	9	4107.942	4149.864	4124.067

```
Selected (max) LL:   ARMA(6,1)
Selected (min) AIC:  ARMA(6,0)
Selected (min) BIC:  ARMA(0,1)
Selected (min) HQIC: ARMA(6,0)
```

A more complex model will better fit the data but brings costs in the form of higher complexity. Model selection criteria weigh model fit against model complexity. Different criteria penalize model complexity to differing degrees, so the selected model can differ by criterion. Here the Akaike information criterion (AIC) and Hannan–Quinn information criterion (HQIC) select an autoregressive model with 6 lags, while the Bayesian information criterion (BIC) selects a model with 1 moving-average lag. We will investigate the autoregressive model with 6 lags. To learn more about ARIMA model selection, see [TS] **arimasoc**.

The arima command fits ARMA models. To fit a model with 6 autoregressive lags, we use the ar() option:

```
. arima lnr_mkt, ar(1/6)
(setting optimization to BHHH)
Iteration 0:  Log likelihood = -2044.9949
Iteration 1:  Log likelihood =  -2044.992
Iteration 2:  Log likelihood = -2044.9917
Iteration 3:  Log likelihood = -2044.9916
Iteration 4:  Log likelihood = -2044.9916
(switching optimization to BFGS)
Iteration 5:  Log likelihood = -2044.9916
Iteration 6:  Log likelihood = -2044.9916
ARIMA regression
Sample: 1955m2 thru 2019m12                     Number of obs     =        779
                                                Wald chi2(6)      =      76.12
Log likelihood = -2044.992                      Prob > chi2       =     0.0000
```

lnr_mkt	Coefficient	OPG std. err.	z	P>|z|	[95% conf.	interval]
lnr_mkt						
_cons	.5761579	.1778133	3.24	0.001	.2276502	.9246656
ARMA						
ar						
L1.	.257773	.0330009	7.81	0.000	.1930925	.3224535
L2.	-.0610409	.0344089	-1.77	0.076	-.1284811	.0063993
L3.	.0428401	.0338751	1.26	0.206	-.0235538	.1092341
L4.	.0129247	.0324971	0.40	0.691	-.0507685	.0766178
L5.	.1095655	.0333176	3.29	0.001	.0442642	.1748667
L6.	-.129245	.029311	-4.41	0.000	-.1866935	-.0717964
/sigma	3.34029	.0611667	54.61	0.000	3.220406	3.460175

```
Note: The test of the variance against zero is one sided, and the two-sided
      confidence interval is truncated at zero.
```

The results indicate that market returns are mildly forecastable. A 1 percentage point higher return in a given period forecasts a 0.26 percentage point higher return in the subsequent period. The coefficients on lags greater than the one period are smaller, ranging from −0.13 to 0.11. We conclude there is mild evidence against a pure random walk.

Finally, we store the ARMA model results for later use.

```
. estimates store armamodel
```

The model selection we performed earlier used in-sample information criteria that trade off model fit and model complexity. But this is not the only approach available. We could have instead chosen a model according to out-of-sample fit, for example, by minimizing out-of-sample mean squared error. We could split the sample, say, by holding out a fraction of the most recent observations and assess fit on the held-out observations. To hold out all observations after the year 1999, we can first create a variable that is 1 for observations before the year 2000 and 0 afterward.

```
. generate insample = (datem < tm(2000m1))
```

Then, we could fit a number of models to the in-sample observations. As a baseline, an AR(0) model has no dependence at all, simply estimating a mean.

```
. arima lnr_mkt if insample
```

The model's predictions could then be obtained with `predict`, and the mean squared error of the prediction could be generated as a new variable.

```
. predict double ar0
. generate double mse0 = (lnr_mkt - ar0)^2
```

Next, an AR(1) model could add some structure and would be fit by typing

```
. arima lnr_mkt if insample, ar(1)
```

This model's predictions could also be obtained with `predict`, and the mean squared error of the prediction could again be generated as a new variable.

```
. predict double ar1
. generate double mse1 = (lnr_mkt - ar1)^2
```

We could then do the same for an AR(2) model,

```
. arima lnr_mkt  if insample, ar(1/2)
. predict double ar2
. generate double mse2 = (lnr_mkt - ar1)^2
```

and continue fitting as many models as theory suggests are plausible. If we did so, say, up to an AR(6) model, then the in-sample mean squared errors can be compared with

```
. summarize mse* if insample
```

Variable	Obs	Mean	Std. dev.	Min	Max
mse0	539	11.5329	23.26755	.0000422	199.0792
mse1	539	10.86623	21.21991	5.23e-09	168.1059
mse2	539	10.83973	21.28822	.0000248	172.7485
mse3	539	10.839	21.28625	1.50e-06	171.6705
mse4	539	10.8258	21.33891	1.78e-06	176.124
mse5	539	10.79703	21.17437	4.96e-06	169.7509
mse6	539	10.68633	20.4926	.0000626	164.5492

The in-sample mean squared error of the baseline model, with no autoregressive structure, is 11.53. Adding structure reduces in-sample mean squared error, but the gains from adding lags beyond the first is small. Still, these are in-sample observations, and more important are out-of-sample tests.

The out-of-sample mean squared errors can be compared with

```
. summarize mse* if !insample
```

Variable	Obs	Mean	Std. dev.	Min	Max
mse0	240	13.67175	41.57669	.0000284	551.7558
mse1	240	12.91045	37.46704	.0019075	487.759
mse2	240	12.86826	37.26065	.000111	482.0199
mse3	240	12.84568	37.09217	.0003288	479.2123
mse4	240	12.81103	36.93194	.0000197	474.2833
mse5	240	12.68043	37.2259	.0000468	480.6316
mse6	240	12.35502	36.11761	.0016538	467.5045

The selected model would be the one with the smallest out-of-sample prediction error. Looking across the models, again we see that there is a reduction in mean squared error by including some autoregressive structure rather than none at all. However, adding structure beyond the first brings only modest reduction in mean squared error. The best model is the most complex, the AR(6), though it is only marginally better than the AR(1). For more information about fitting ARIMA models in Stata, see [TS] **arima**.

This out-of-sample procedure is one option among many and one that focused on lagged values as independent variables. We could instead use `bmaregress` to perform Bayesian model averaging of multiple models. Or apply the `h2oml` and `python` commands to analyze financial returns using the vast machine learning toolkits in H2O and Python.

Modeling volatility

Next we investigate persistence in volatility. Previously, we were interested in understanding the predictability of returns themselves. Now we turn to models of volatility to investigate the question of whether periods of high, or low, volatility are themselves autocorrelated.

As an initial pass, we use `estat archlm`, a tool available after `regress`, to check for the presence of time-varying volatility (specifically, conditional heteroskedasticity) in the errors from an autoregressive model.

```
. regress lnr_mkt L(1/6).lnr_mkt
```

Source	SS	df	MS		
Model	787.378292	6	131.229715	Number of obs	= 773
				F(6, 766)	= 11.67
Residual	8614.83162	766	11.2465165	Prob > F	= 0.0000
				R-squared	= 0.0837
				Adj R-squared	= 0.0766
Total	9402.20991	772	12.1790284	Root MSE	= 3.3536

lnr_mkt	Coefficient	Std. err.	t	P>|t|	[95% conf.	interval]
lnr_mkt						
L1.	.2581365	.0357805	7.21	0.000	.1878971	.3283758
L2.	-.0631958	.036761	-1.72	0.086	-.1353601	.0089684
L3.	.0428656	.0368258	1.16	0.245	-.029426	.1151571
L4.	.0132665	.0368247	0.36	0.719	-.0590229	.0855559
L5.	.1078361	.0367977	2.93	0.003	.0355998	.1800724
L6.	-.1291023	.0357914	-3.61	0.000	-.1993632	-.0588415
_cons	.4280706	.1267854	3.38	0.001	.1791825	.6769587

```
. estat archlm, lag(1/6)
LM test for autoregressive conditional heteroskedasticity (ARCH)
```

lags(*p*)	chi2	df	Prob > chi2
1	4.771	1	0.0289
2	5.012	2	0.0816
3	7.083	3	0.0693
4	7.651	4	0.1052
5	8.833	5	0.1159
6	24.327	6	0.0005

H0: no ARCH effects *vs.* H1: ARCH(*p*) disturbance

The estat archlm command performs a Lagrange multiplier test for the presence of autoregressive conditional heteroskedasticity (ARCH) behavior in the errors of the regression. We see sufficient evidence of ARCH behavior, so we will investigate it more fully.

Continuing with the suggested specification from the previous section, here we use the arch command to fit an autoregressive model of order 6 to capture any dependence in the level of returns and add GARCH(1,1) terms to capture time-varying volatility.

```
. arch lnr_mkt, ar(1/6) arch(1) garch(1)
(setting optimization to BHHH)
Iteration 0:  Log likelihood = -2037.5614
Iteration 1:  Log likelihood = -2029.6178
Iteration 2:  Log likelihood = -2023.7281
Iteration 3:  Log likelihood = -2021.6537
Iteration 4:  Log likelihood = -2020.3526
(switching optimization to BFGS)
Iteration 5:  Log likelihood = -2019.7859
Iteration 6:  Log likelihood = -2019.1407
Iteration 7:  Log likelihood =  -2019.067
Iteration 8:  Log likelihood = -2019.0626
Iteration 9:  Log likelihood = -2019.0623
Iteration 10: Log likelihood = -2019.0622
Iteration 11: Log likelihood = -2019.0622
ARCH family regression -- AR disturbances
Sample: 1955m2 thru 2019m12                     Number of obs     =        779
                                                Wald chi2(6)      =      35.92
Log likelihood = -2019.062                      Prob > chi2       =     0.0000
```

lnr_mkt	Coefficient	OPG std. err.	z	P>|z|	[95% conf.	interval]
lnr_mkt						
_cons	.7233727	.1403091	5.16	0.000	.448372	.9983734
ARMA						
ar						
L1.	.2211217	.0429604	5.15	0.000	.1369209	.3053225
L2.	-.0539058	.03845	-1.40	0.161	-.1292664	.0214548
L3.	.0299586	.0386123	0.78	0.438	-.0457201	.1056374
L4.	.0025786	.0389335	0.07	0.947	-.0737297	.078887
L5.	.0760806	.0380917	2.00	0.046	.0014223	.1507389
L6.	-.0951334	.0329823	-2.88	0.004	-.1597775	-.0304892
ARCH						
arch						
L1.	.1382138	.0266536	5.19	0.000	.0859737	.1904539
garch						
L1.	.7834001	.0480069	16.32	0.000	.6893084	.8774919
_cons	.9715679	.3603296	2.70	0.007	.2653348	1.677801

The ARCH and GARCH terms are both positive, indicating persistence in volatility. The autoregressive model parameters have changed slightly, but the main features remain: average returns continue to be slightly forecastable. If returns are 1 percentage point higher than average in the current period, we would forecast the return being 0.22 percentage points higher than average in the subsequent period, all else equal. For more information on ARCH and GARCH models, see [TS] **arch**.

Forecasting

Stata has a forecasting environment to make predictions given a model; see [TS] **forecast**. We can, for example, use the previously fit ARMA model to generate a forecast for market returns over the first six months out of sample, along with confidence intervals that account for estimation uncertainty and the random errors arising from the stochastic nature of the model.

To begin the process, we add six periods (months) to the end of the dataset. These will be the months for which we forecast market returns.

```
. tsappend, add(6)
```

To start a forecast, we use `forecast create` and give the forecast model a name.

```
. forecast create arimafc
  Forecast model arimafc started.
```

We add the previously fit model `armamodel` to the forecast using `forecast estimates`.

```
. forecast estimates armamodel
  Added estimation results from arima.
  Forecast model arimafc now contains 1 endogenous variable.
```

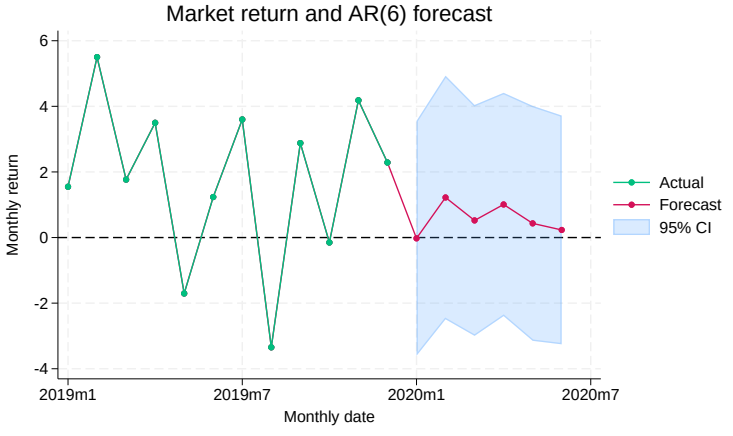
We now solve the forecast model, where "solve" means to actually make our forecast. We use the `prefix()` option to add a prefix to the newly generated forecast variables and use the `begin()` option to begin the forecast at a certain date. In the `armamodel` estimates, the dependent variable is the log market return `lnr_mkt`. Thus, the newly created forecast will be stored in a variable called `arma_lnr_mkt`. We use the first month of 2020 as our beginning forecast date.

```
. forecast solve, prefix(arma_) begin(tm(2020m1))
> simulate(betas errors, statistic(stddev, prefix(sd_)) reps(100))
Computing dynamic forecasts for model arimafc.
------------------------------------------------------------

Starting period:  2020m1
Ending period:    2020m6
Forecast prefix:  arma_

2020m1:  ..............
2020m2:  ..............
2020m3:  ..............
2020m4:  ..............
2020m5:  ..............
2020m6:  ..............

Performing simulations (100): ..................................................... 50
.....................................................  100

Forecast 1 variable spanning 6 periods.
------------------------------------------------------------
```

The second line of the `forecast solve` command above specifies how uncertainty is treated. We create an additional variable with prefix `sd_` to store the standard deviation of the forecast variable. This standard deviation incorporates two sources of uncertainty. The specification `betas` accounts for the fact that the parameters were estimated and thus subject to uncertainty. The specification `errors` accounts for the randomness inherent in the stochastic process. For more details on uncertainty specification, see example 2 of [TS] **forecast solve**.

The output from `forecast solve` informs us that we made predictions for 6 forecast periods and that our uncertainty estimates were drawn from 100 simulations of the parameter estimates (`betas`) and stochastic uncertainty (`errors`).

A graph of the forecast shows the following:



Producing this graph takes two steps. First, we produce variables for the upper and lower confidence band. Second, we graph the actual market return, the forecasted return, and the confidence band. We use some of the tools in the `graph twoway` command to add nice titles to the axes, add nice labels to the series, and order the legend.

```
. generate lnr_mkt_up = arma_lnr_mkt + invnormal(0.975) * sd_lnr_mkt
(780 missing values generated)
. generate lnr_mkt_dn = arma_lnr_mkt + invnormal(0.025) * sd_lnr_mkt
(780 missing values generated)
. twoway  (rarea lnr_mkt_dn lnr_mkt_up datem if tin(2019m1,), color(stblue%20))
>         (connected arma_lnr_mkt datem if tin(2019m1,))
>         (connected lnr_mkt datem if tin(2019m1, )),
> legend(label(1 "95% CI")) legend(label(2 "Forecast"))
> legend(label(3 "Actual")) legend(order( 3 2 1))
> ytitle("Monthly return") title("Market return and AR(6) forecast") yline(0)
```

Turning to the results, we see that the forecast eventually settles at the constant reported in the `arima` output, which is close to the sample average market return. The confidence intervals are wide, reflecting considerable uncertainty in forecasted returns.

The `forecast` command allows for dynamic, multistep forecasting after virtually all Stata estimation commands. Also of interest in a time-series context are the smoothers, which can also be used for forecasting. See [TS] **tssmooth** for details.

Multivariate models

So far, we have mainly examined one time series in isolation, the log market return. But it is also possible to analyze time series jointly. The main benefit of doing so arises when the two series exhibit jointly dependent behavior, either contemporaneously or over time. As an example, we can jointly model the market return and the risk-free rate of return, `lnr_mkt` and `rf`, in a vector autoregression model. Whereas a univariate autoregression allows a time series to depend on its own lags, a vector autoregression allows a time series to depend on both its own lags and on lags of other series. The following uses the `var` command to fit a vector autoregression model for the two series, using two lags of each variable as explanatory variables.

```
. var lnr_mkt rf
Vector autoregression
Sample: 1955m4 thru 2019m12                     Number of obs     =       777
Log likelihood =  -516.1938                     AIC               =  1.354424
FPE            =   .0132822                     HQIC              =  1.377473
Det(Sigma_ml)  =   .0129447                     SBIC              =   1.41434
Equation           Parms      RMSE     R-sq      chi2     P>chi2
------------------------------------------------------------------------------
lnr_mkt                5   3.37616   0.0717   59.99281   0.0000
rf                     5   .033964   0.9855   52702.85   0.0000
------------------------------------------------------------------------------
------------------------------------------------------------------------------
             |      Coefficient  Std. err.      z    P>|z|     [95% conf. interval]
-------------+----------------------------------------------------------------
lnr_mkt      |
     lnr_mkt |
         L1. |      .2397095   .0357808     6.70   0.000     .1695804    .3098385
         L2. |     -.0447416   .0358032    -1.25   0.211    -.1149146    .0254314
             |
          rf |
         L1. |     -9.924366   3.270127    -3.03   0.002    -16.3337   -3.515035
         L2. |      9.438268   3.269906     2.89   0.004     3.02937    15.84717
             |
       _cons |      .6519097   .2094337     3.11   0.002     .2414272    1.062392
-------------+----------------------------------------------------------------
rf           |
     lnr_mkt |
         L1. |      .0008055     .00036     2.24   0.025        .0001    .001511
         L2. |      .0009662   .0003602     2.68   0.007     .0002602    .0016721
             |
          rf |
         L1. |      1.378865   .0328971    41.91   0.000     1.314388    1.443342
         L2. |     -.3897623   .0328949   -11.85   0.000    -.4542351   -.3252894
             |
       _cons |      .0032188   .0021069     1.53   0.127    -.0009106    .0073482
------------------------------------------------------------------------------
```

In the first equation, we are regressing `lnr_mkt` on its first two lags and on the first two lags of the risk-free rate. We see that market returns are mildly forecastable in that the lagged market return provides information on the current market return. But there is also cross-dependence: the market return also reacts to the lagged risk-free rate. The first cross-lag coefficient is -9.92, indicating that an increase in the risk-free rate leads to negative market returns one period (one month) later. The second lag is actually positive, of about the same magnitude. This too can be interpreted: it means that the market return reacts to the change in the risk-free rate, not the level. To see this, write the estimated equation in the following way. Letting r_t^m be the market return and r_t^f be the risk-free rate, the first equation is approximately

$$r_t^m = 0.65 + 0.24 \times r_{t-1}^m - 0.04 \times r_{t-2}^m - 9.9 \times r_{t-1}^f + 9.4 \times r_{t-2}^f$$

where all the numbers come from the coefficients in the first equation in the output table. A little algebraic rearranging gives

$$r_t^m = 0.65 + 0.24 \times r_{t-1}^m - 0.04 \times r_{t-2}^m - 0.5 \times r_{t-1}^f - 9.4 \times (r_{t-1}^f - r_{t-2}^f)$$

which is more interpretable: the market return responds to the lagged level of the risk-free rate and reacts to lagged changes in the risk-free rate.

Turning to the risk-free rate itself, we see that it responds to its own lags and also responds in a small way to the lagged market return.

The model is illustrative. We could have chosen different lag lengths, perhaps using `varsoc` to choose the lag order similarly to how we used `arimasoc` earlier. In addition, after fitting a vector autoregression, we can obtain unconditional forecasts with `fcast compute` or `fcast graph`, assess the impact of shocks with `irf`, and test for the significance of those cross-lag terms jointly with `vargranger`. For much more information on how to fit, analyze, and interpret vector autoregressions, see [TS] **var**.

The `var` analogue for the study of patterns of conditional volatility in a multivariate context is a multivariate GARCH model, implemented in the `mgarch` command. In such a model, periods of volatility cluster together, just as in a univariate GARCH model; but unlike a univariate GARCH model, the multivariate GARCH model can also characterize correlations in volatility across time series. We can ask, for instance, whether one series (like the market return) is volatile precisely when another series (like the risk-free rate) is volatile or whether the two experience periods of volatility independently. This kind of question is often asked of multiple asset returns or exchange rates across countries, in which the patterns of correlated volatility can indicate the presence of linkages in the financial markets across countries. For more on the multivariate GARCH model, including four different submodels of volatility and the ways to interpret them, see [TS] **mgarch**.

Date and time management

At the beginning of this introduction, we used `tsset` to declare to Stata that we are working with time-series data. This section provides additional information on `tsset`, especially for use with daily data. Much financial data arrive daily but with regular and expected gaps. Financial markets are open only on weekdays, so that the "lag" of a Monday is the previous Friday, not the preceding Sunday. Financial markets can be closed for holidays, for example, Christmas and Thanksgiving (in the United States) or Showa Day (in Japanese markets, near the end of April) or Chinese New Year (in Chinese markets). These regularly observed holidays cause gaps in the data that are only apparent, not true gaps.

Stata has the concept of business calendars to handle such situations. A business calendar begins with daily data, then applies some rules specified in the calendar to appropriately treat certain gaps. We provide a few example business calendars for some stock exchanges around the world and encourage you to make your own calendar to fit your specific needs.

A business calendar is a file, ending in `.stbcal`, containing rules for how certain gaps should be handled. Let's look at one example calendar for the New York Stock Exchange for the years 2024–2028.

begin _nyse.stbcal

```
*! version 1.0.0 05may2026
*  New York Stock Exchange business calendar (_nyse.stbcal)
*  Source: https://www.nyse.com/markets/hours-calendars
*  Range: January 1, 2024, to December 31, 2028
version 19.5
purpose "NYSE business calendar"
dateformat dmy
range 02jan2024 29dec2028
centerdate 02jan2024

* Weekends
omit dayofweek(Sa Su)

* New Year's Day
omit date 01jan2024
omit date 01jan2025
omit date 01jan2026
omit date 01jan2027

* Martin Luther King Jr. Day
omit date 15jan2024
omit date 20jan2025
omit date 19jan2026
omit date 18jan2027
omit date 17jan2028

* Washington's Birthday
omit date 19feb2024
omit date 17feb2025
omit date 16feb2026
omit date 15feb2027
omit date 21feb2028

* Good Friday
omit date 19mar2024
omit date 18apr2025
omit date 03apr2026
omit date 26mar2027
omit date 14apr2028

* Memorial Day
omit date 27may2024
omit date 26may2025
omit date 25may2026
omit date 31may2027
omit date 29may2028

* Juneteenth National Independence Day
omit date 19jun2024
omit date 19jun2025
omit date 19jun2026
omit date 18jun2027
omit date 19jun2028
```

```
* Independence Day
omit date 04jul2024
omit date 04jul2025
omit date 03jul2026
omit date 05jul2027
omit date 04jul2028

* Labor Day
omit date 02sep2024
omit date 01sep2025
omit date 07sep2026
omit date 06sep2027
omit date 04sep2028

* Thanksgiving Day
omit date 28nov2024
omit date 27nov2025
omit date 26nov2026
omit date 25nov2027
omit date 23nov2028

* Christmas Day
omit date 25dec2024
omit date 25dec2025
omit date 25dec2026
omit date 24dec2027
omit date 25dec2028
```

end _nyse.stbcal

The business calendar begins with some comments describing its date of construction, source material, and date range. The first few lines fix the `version`, specify a `purpose`, identify the date format, give the range of the calendar, and finally provide a `centerdate`, the date that, for the calendar, represents "time 0". Days are counted in positive integers after the center date and as negative integers before the center date.

Next, the `omit dayofweek(Sa Su)` command marks all weekends as anticipated gaps. Then, New Year's Day is omitted (the first day in January, each year). And we see Martin Luther King Jr. Day omitted; its date changes slightly each year because it is the third Monday in January. Example calendars are provided for five stock exchanges, showing the basic structure of a business calendar and how to carefully omit dates. To learn how to download these example calendars, see [D] **Datetime business calendars** and [D] **bcal**.

As a second example, let's look at _stock.stbcal, which is designed to work with stocks.dta.

begin _stock.stbcal

```
* version 1.0.0  07may2026
* Business calendar for -webuse stocks-
* Range: January 1, 2003, to December 31, 2010
version 19.5
purpose "Stock calendar for 2003-2010"
dateformat dmy
range 02jan2003 31dec2010
centerdate 02jan2003

* Weekends
omit dayofweek (Sa Su)

* New Year's Day
omit date 01jan*
omit date 01jan* and (+1) if dow(Su)
omit date 02jan2007

* Martin Luther King Jr. Day
omit dowinmonth +3 Mo of Jan

* President's Day
omit dowinmonth +3 Mo of Feb
  (output omitted)

* Memorial Day
omit dowinmonth -1 Mo of May
  (output omitted)

* Christmas
omit date 25dec*
omit date 25dec* and (-1) if dow(Sa)
omit date 25dec* and (+1) if dow(Su)
```

end _stock.stbcal

In this calendar, we again set version, specify purpose, set dateformat, and give range over which the calendar is valid. The calendar is valid for dates between January 2, 2003, and December 31, 2010, with a center date (0 date) of January 2, 2003. Here are some comments on the syntax and rules:

- As before, we omit all weekends via omit dayofweek(Sa Su).
- For New Year's Day, we use the wildcard * to mark all January 1s as an anticipated gap, regardless of year. Also, if January 1 falls on a Sunday, we additionally omit the following Monday.
- For Martin Luther King Jr. Day, we use some of the rules available in business calendars. Specifically, we omit the third Monday in each January with omit dowinmonth +3 Mo of Jan; this one line marks the third Monday in each January as an anticipated gap.
- We do the same for President's Day, omitting the third Monday in each February.
- For Memorial Day, which falls on the final Monday of each May, we use the rule -1 Mo of May, which omits this date.

Many other rules can be specified. See [D] **Datetime business calendars** for details.

Once you have carefully constructed the correct calendar for your application, how do you apply it to the data? Let's look at `stocks.dta`.

```
. use https://www.stata-press.com/data/r19/stocks, clear
(Data from Yahoo! Finance)
. describe
Contains data from https://www.stata-press.com/data/r19/stocks.dta
 Observations:         2,015                  Data from Yahoo! Finance
    Variables:             5                  9 May 2024 11:48
-------------------------------------------------------------------------------
Variable      Storage   Display    Value
    name         type    format    label      Variable label
-------------------------------------------------------------------------------
date            int     %td                   Date
t               int     %9.0g                 Generic time variable
toyota          float   %9.0g                 Daily return on Toyota stock
nissan          float   %9.0g                 Daily return on Nissan stock
honda           float   %9.0g                 Daily return on Honda stock
-------------------------------------------------------------------------------
Sorted by: t
. tsset date
Time variable: date, 02jan2003 to 31dec2010, but with gaps
        Delta: 1 day
. tsreport
Time variable: date
-----------------------------
Starting period = 02jan2003
Ending period   = 31dec2010
Number of obs   =     2,015
Number of gaps  =       433
```

We used `tsreport` to get an overview of the span of the data and to alert us to the presence of gaps. This dataset has daily data from January 2, 2003, to December 31, 2010. If we list the first few observations of the Toyota stock price and its lag, we obtain the following:

```
. list date toyota L.toyota in 1/10
```

	date	toyota	L. toyota
1.	02jan2003	.0151675	.
2.	03jan2003	.0048201	.0151675
3.	06jan2003	.0199587	.
4.	07jan2003	-.0133226	.0199587
5.	08jan2003	-.0270011	-.0133226
6.	09jan2003	.0116346	-.0270011
7.	10jan2003	-.0093722	.0116346
8.	13jan2003	.0016935	.
9.	14jan2003	-.0062234	.0016935
10.	15jan2003	-.0039806	-.0062234

There are unintended gaps due to weekends and holidays. So let's load our stock calendar and apply it.

```
. bcal webcopy stock
(business calendar _stock.stbcal copied to working directory)
. generate dateb = bofd("_stock", date)
```

We used the generate command to create a new variable, dateb. This new variable starts from the daily date date and applies the rules in stocks.stbcal to create a running date that omits the gaps we specified.

```
. tsset dateb
Time variable: dateb, 0 to 2014
        Delta: 1 unit
. format dateb %tb_stock
. tsreport
Time variable: dateb
------------------------------
Starting period = 02jan2003
Ending period   = 31dec2010
Number of obs   =     2,015
Number of gaps  =         0
```

The business calendar has rid us of the gaps because it correctly recognizes holidays and weekends. If we look at the current and lagged values of the first few observations on the Toyota stock, we see

```
. list date dateb toyota L.toyota in 1/10
```

	date	dateb	toyota	L. toyota
1.	02jan2003	02jan2003	.0151675	.
2.	03jan2003	03jan2003	.0048201	.0151675
3.	06jan2003	06jan2003	.0199587	.0048201
4.	07jan2003	07jan2003	-.0133226	.0199587
5.	08jan2003	08jan2003	-.0270011	-.0133226
6.	09jan2003	09jan2003	.0116346	-.0270011
7.	10jan2003	10jan2003	-.0093722	.0116346
8.	13jan2003	13jan2003	.0016935	-.0093722
9.	14jan2003	14jan2003	-.0062234	.0016935
10.	15jan2003	15jan2003	-.0039806	-.0062234

This result is as desired.

Summing up

This entry has introduced the fin suite of commands for processing asset prices, creating returns, creating portfolios, summarizing returns, running financial regressions, and assessing VaR. It has also provided a brief introduction to some of Stata's time-series commands that are relevant for finance, like arima, arch, and var. This introduction was not comprehensive; we could have shown you mgarch for multivariate volatility models, vec for models of variables with common trends, pca and factor for static unobserved factor models, or dfactor for unobserved dynamic factor models.

Also see

[FIN] **Intro** — Introduction to financial statistics+