

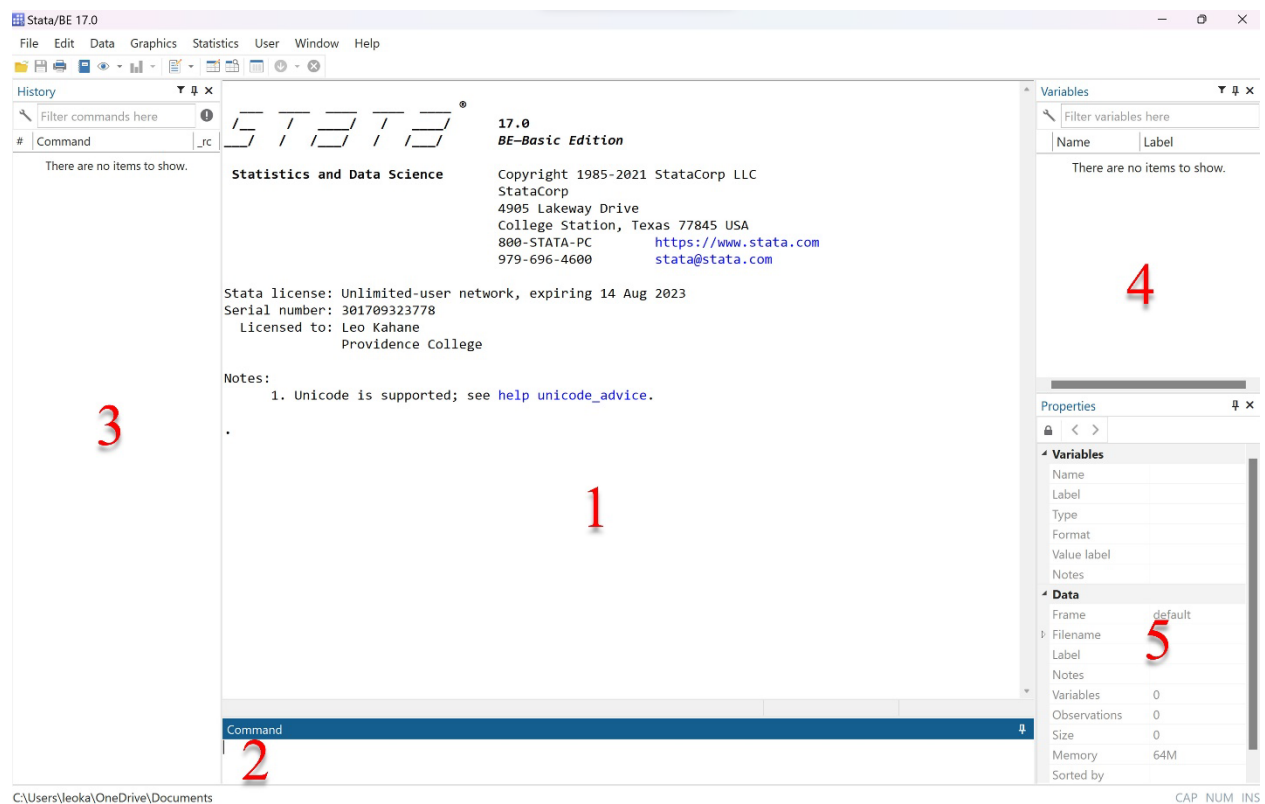
Instructions for Stata®

Stata is produced by StataCorp LLC., and can be obtained [here](#).

1. Appearance

[Note: The screen shots provided below are from a Windows version of Stata 17 (Basic Edition). Versions for other operating systems (e.g. for MacOS) may appear slightly different.]

Once downloaded and Stata is launched you will see an opening screen like that shown below:



As indicated, there are 5 areas shown on the opening screen.

Area **1**: This is the output window. This is where Stata shows the user what has been done following a command.

Area **2**: This is the command line. This is where the user can write commands. Hitting the ‘Enter’ key will submit the command.

Area **3**: A history of commands submitted on the command line will be populated here. Once a command appears in this list, it can be clicked, and the command will appear again on the command line (useful for repeating a command without having to re-type it).

Area **4**: This area shows the names of the variables that are in the current data set (including variables created by the user).

Area **5**: This area shows the properties of variables. Click on a variable in Area **4** and the properties of that variable will be displayed in area **5**.

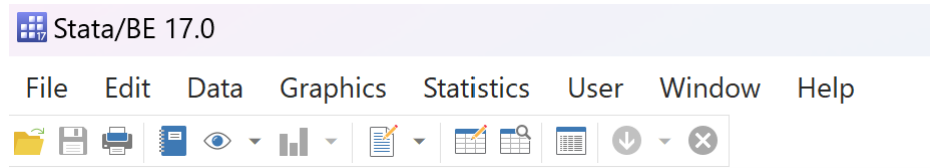
(Note that the size of these areas can be adjusted by clicking and dragging their borders within the opening window.)



At the very bottom of the screen, we can see the current working directory. If the user wishes to change the working directory, this can be done by clicking ‘File’ on the main menu, then click ‘Change working directory...’, and then select the desired working directory. Alternatively, the user can type into the command line `cd` (short for “change directory”) followed by the address of the desired work directory. For example, if we type the command,

```
cd C:\Users\leoka\project1
```

hitting ‘Enter’ would change the working directory to `C:\Users\leoka\project1`, (assuming this location exists already).

The main menu is shown near the top edge of the opening screen,



We will discuss these main menu items as we progress through the instructions on how to use Stata. Below the main menu is a series of icons. Two of them have to do with viewing and/or editing the current data in memory:  . Click the spreadsheet icon with a pencil, and you will be shown a new window called the ‘Data Editor (Edit)’ containing the data. For example, if we have the NBA data set in memory (“nba2021_22.csv”) we would see,

Data Editor (Edit) - [nba]

File Edit View Data Tools

player[1] Find Aaron Gordon

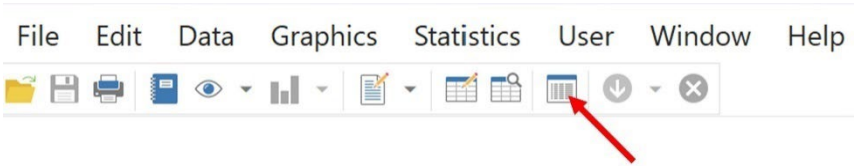
	player	position	age	salary	team	weight	height	games
1	Aaron Gordon	PF	26	20	ORL	235	80	453
2	Aaron Holiday	PG	25	2.61934	IND	185	72	182
3	Aaron Nesmith	SF	22	4.13205	BOS	215	77	46
4	Al Horford	C	35	27.25	ATL	240	81	881
5	Alec Burks	SG	30	10.0128	UTH	214	78	544
6	Aleksej Pokusevski	PF	20	3.58728	OKC	190	84	45
7	Alex Caruso	SG	27	9.245	LAL	186	77	184
8	Alex Len	C	28	3.8253	PHX	250	84	531
9	Andre Iguodala	SF	38	2.64169	GSW	215	78	1192
10	Andrew Wiggins	SF	26	29.542	MIN	205	79	525
11	Anfernee Simons	SG	22	2.54398	POR	181	75	154
12	Anthony Edwards	SG	20	11.0678	MIN	225	77	72
13	Anthony Gill	PF	29	1.20815	WAS	230	80	26
14	Austin Rivers	PG	29	2.40154	LAC	200	76	588

If we click on a cell and begin typing, and then hit enter, this will replace the contents of the cell with the newly typed data, (and a message about the changed data will appear in the output window). Clicking on the spreadsheet icon with the magnifying glass opens a similar looking

window called the ‘Data Editor (Browse)’, which is for viewing only and no changes can be made to the data.

Note that the first column of data are simply observation numbers (1, 2, 3...). Also note that the variable names appear above each column of data. Data entries in red type (e.g., ‘player’, ‘position’, and ‘team’) indicate that these data are strings, not numeric values.

The Variables Manager icon is also displayed below the main menu (see the red arrow below),



Clicking this icon opens the Variables Manager window,

Variables Manager

Filter variables here

Drag a column header here to group by that column.

#	Name	Label	Type
	player	Player first and last name	str24
	position	Center, Point Guard, Power Forward, Small Forward, Shooting Guard	str2
	age		byte
	salary		float
	team		str3
	weight		int
	height		byte
	games		int
	minspg		float
	orebsp		float
	drebsp		float
	trebsp		float
	astspg		float
	stlspg		float
	blockspg		float
	tovspg		float
	foulspg		float
	pointspg		float
	seasons		byte

Variable properties

Name:position

Label:Center, Point Guard, Power Forward, Small Forward, Shooting Guard

Type:str2

Format:%9sCreate...

Value label:Manage...

Notes:No notesManage...

<>ResetApply

Here we can find all the variables by Name, Label (if they have one) and Type. Clicking on a variable allows us to change the Name and to add or edit the Label in the Variable properties box on the right-hand side.

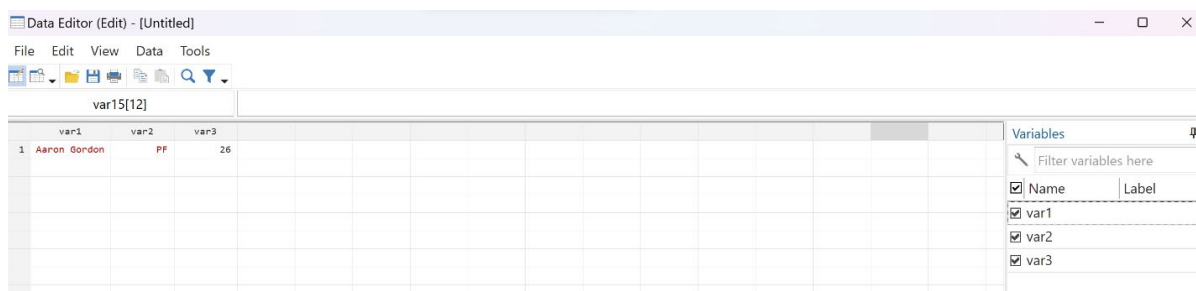
2. Entering Data

Manually

There are two primary ways to enter data into Stata: manually and reading in a data file. As an example, we can enter data from our NBA data set (“nba2021_22.csv”, available on the *Regression Basics* companion website) manually. Here is some of the information about our first player in the data set:

player:	Aaron Gordon
position:	PF
age:	26
salary:	20

We can begin by opening the Data Editor (Edit) window and typing the data into the spreadsheet,



Notice that as the data were typed into Stata, the program assigns each column a generic ‘var1’, ‘var2’, ‘var3’,... variable name. This can be changed later by opening the Variables Manager

window and editing the names. Alternatively, variable names can be changed using the command line. Simply type into the command line,

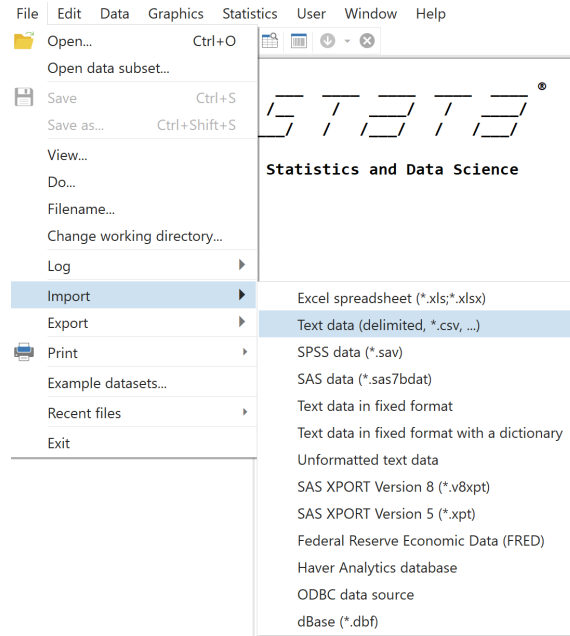
```
rename var1 player
```

hit enter, and the var1 name is changed to player. A similar command can be used for the other variables.

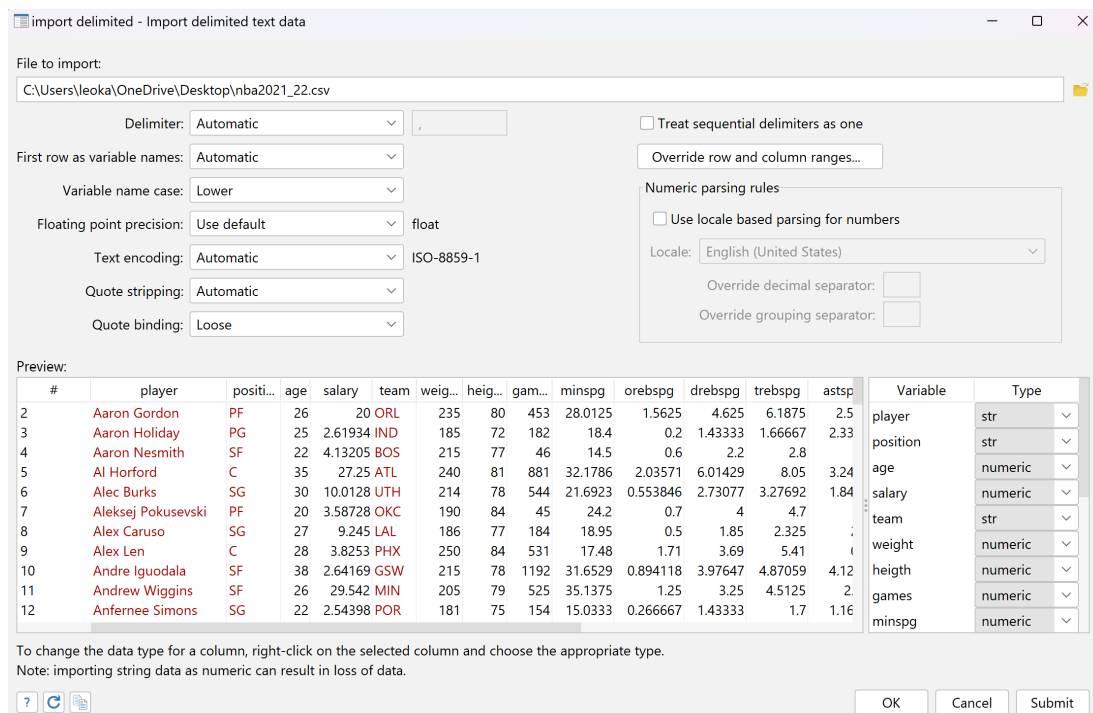
Once finished with the first observation, we would move to the second row and enter the information for the next observation, and so on. The user should save the file by clicking on 'File' on the main menu, then click 'Save as...', and then chose a location and name for the file that will be saved. Subsequent saving of the file can be done by clicking the save icon on the main menu, or by holding down the control key, Ctrl, and then hitting the 'S' key.

Reading Data Files into Stata

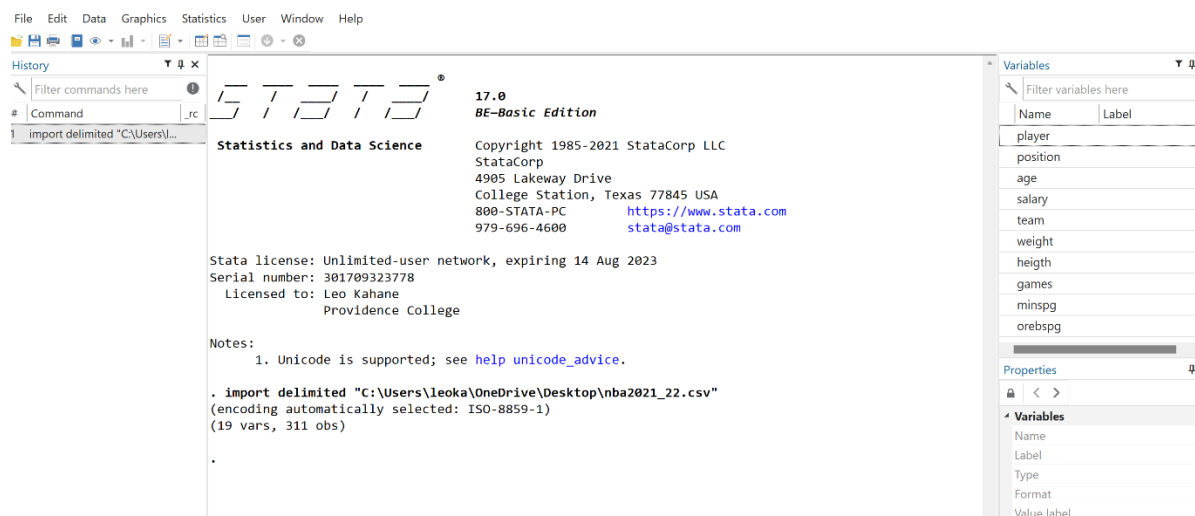
While entering data into Stata manually may be necessary in some cases, most often we will be working with data sets that can be read into Stata as a whole. Stata can import a variety of file types, including Excel, CSV, Text, SAS, SPSS, and others. Importing data into Stata can be done using the drop-down menus. Begin by clicking 'File' on the main menu, then 'Import', then choose the file type to import. We will choose 'Text data (delimited, *.csv,...)' since the NBA data are stored as a .csv file,



This will open a dialog box where we enter the name and location of the file to import. Clicking the small folder icon on the far right allows the user to browse for the file to import and then once located, clicking on it will populate the location box and also provide a preview of what the imported data look like,



Clicking ‘OK’ will submit the command and the data will be read into Stata. The output window will display what command has just been executed, and the names of the variables for the imported data will be displayed in the Variables box on the right-hand side.



Note that in the output window we see the code that was used to import the data set,

```
import delimited "C:\Users\leoka\OneDrive\Desktop\nba2021_22.csv"
```

This code could have been typed directly into Stata’s command line, then hitting enter would have executed the command and the data would have been read into Stata. This illustrates the utility of using the command line to execute commands instead of going through the drop-down menus.

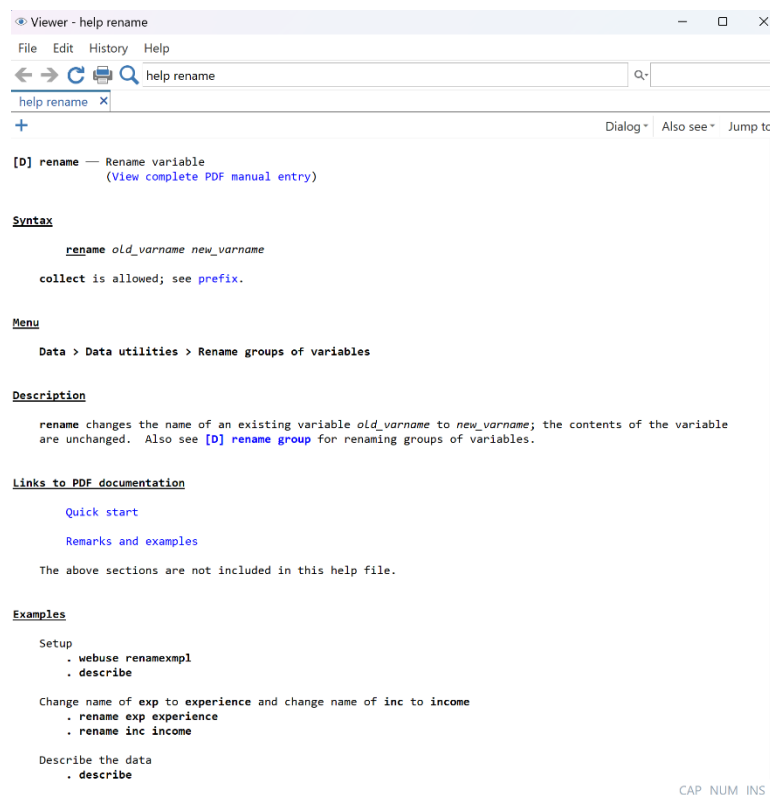
At this point the user would normally save the data set as a Stata data file (.dta) by clicking ‘File’ on the main menu, and then ‘Save As...’ This will bring up a dialog box where you can give the file a name (e.g., nba.dta), and choose the location where you would like the Stata data file to be saved. Later, the data file can be opened by either starting Stata, clicking ‘File’, then ‘Open...’, then navigating to the destination where the file is stored, then clicking it, and then choosing

Open. Alternatively, the user can simply navigate to the saved Stata data file (e.g., nba.dta) and then double click it. This will launch Stata and the file will be read into memory.

[Note: Excel and CSV files can sometimes simply be copied from the source, and then pasted directly into Stata's Data Editor (Edit) screen. This works fine when the source data are well-formatted as just columns of data (which can include variable names as the first row).]

3. Getting Help with Commands

Help with Stata commands can be found by simply typing `help command_name` at the command line, where `command_name` is the name of the command you need help with. For example, getting help for the `rename` command we used earlier would be done by typing `help rename`, then hitting the 'Enter' key. Stata respond by opening a new window with the following,



```

Viewer - help rename
File Edit History Help
help rename
[0] rename — Rename variable
      (View complete PDF manual entry)

Syntax
  rename old_varname new_varname
  collect is allowed; see prefix.

Menu
  Data > Data utilities > Rename groups of variables

Description
  rename changes the name of an existing variable old_varname to new_varname; the contents of the variable
  are unchanged. Also see [0] rename group for renaming groups of variables.

Links to PDF documentation
  Quick start
  Remarks and examples
  The above sections are not included in this help file.

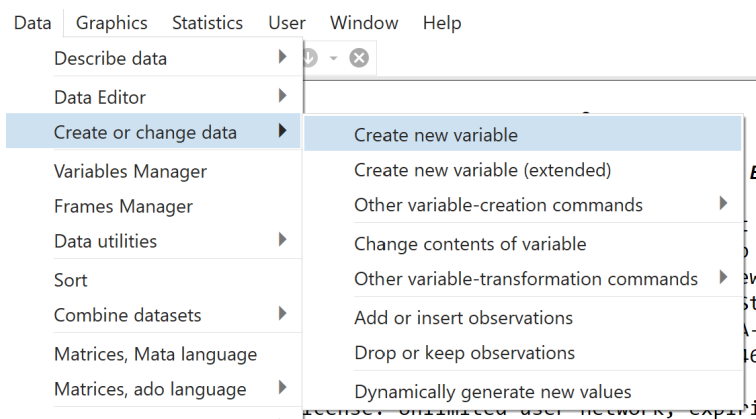
Examples
  Setup
  . webuse renamexmpl
  . describe

  Change name of exp to experience and change name of inc to income
  . rename exp experience
  . rename inc income

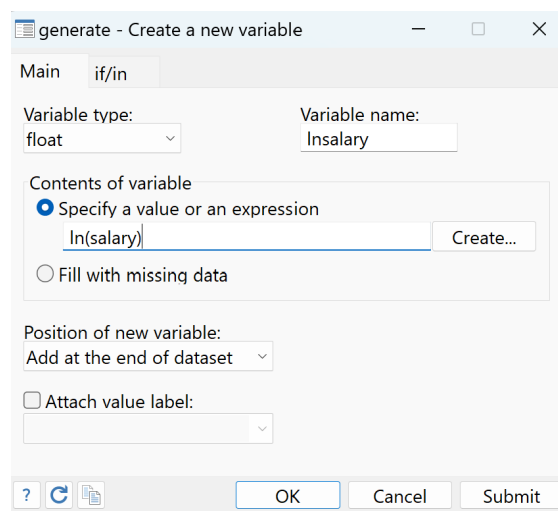
  Describe the data
  . describe
  
```

4. Transforming and Dropping Variables

Once we have entered data into Stata, we may want to transform some variables. For example, we may want to compute the natural log of a variable or compute the squared value of a variable. This can be done by using the drop-down menus, or by submitting a command directly to Stata. As an example, we will create the natural log of the variable salary in the NBA data set. Using the drop-down menus, we start by choosing ‘Data’ from the main menu, then ‘Create or change data’, then ‘Create new variable’,



This will bring up a new dialog box where we provide the name of the new variable in the ‘Variable name:’ box, and the expression to create it in the ‘Contents of variable’ box,



Thus, we will create a new variable called `lnsalary`, and the expression is `ln(salary)`, where the ‘`ln(.)`’ command is used to compute the natural log of a variable in Stata. Clicking OK, the new variable is created and added as a new column of data, and the new variable `lnsalary` is added to our Variables list. Note, once again, Stata shows the code used to create the new variable in the output window,

```
generate lnsalary = ln(salary)
```

This command could have been typed into the command line, and then after hitting ‘Enter’, the variable would have been created. Again, this shows the utility of typing in commands directly into the command line. Note that most Stata commands, like `generate` can be shortened to just the first three letters of the command, such as,

```
gen lnsalary = ln(salary)
```

Other common transformations include the following:

```
gen agesq = age^2      ← this will create a new variable called agesq equal to age2
```

```
gen absres = abs(residual) ← this will create a new variable absres equal to the absolute  
value of the variable residual
```

```
gen ht_wt = height*weight ← this will create a new variable ht_wt equal to the product of the  
variables height and weight
```

In each case, the `generate` command would be typed into the command line, and then hitting the ‘Enter’ key would create the new variable. [Note that, rather than typing the existing variable names into the command line, the user can move the mouse to the Variables window and hover over a variable, then click the ‘down arrow’ button to add the variable into the command line.]

Dropping a Variable

If one wishes to drop one or more variables from the data set, this can easily be done using the command line. For example, to drop `agesq` and `absres` we would type,

```
drop agesq absres
```

Hitting the ‘Enter’ key would drop both these variables from the data set.

5. Summary Statistics and Correlations

Summary Statistics

One of the first things that a researcher will compute once their data set has been read into Stata are summary statistics (also called descriptive statistics). This will allow the researcher to get a feel for the values in the data set (e.g., how big are typical values, and how much they vary). It may also reveal unusual observations that will show up as minimum or maximum values. (In some cases, it may uncover errors in data. For example, if the minimum value for players’ weight was -160, then this would indicate that there is at least one data point that was entered incorrectly into the data set since weight cannot be negative.)

To produce summary statistics, we can use the drop-down menus by choosing ‘Statistics’, then ‘Summaries, tables, tests’, then ‘Summary and Descriptive statistics’, and then ‘Summary statistics’. This will bring up a dialog box where we can choose the variables for which summary statistics will be computed (or we can leave the box blank to choose all variables). As is evident in this case, using the drop-down menus for many tasks in Stata is cumbersome and inefficient.

In the case of summary statistics, it is *much* easier to use the command line. For example, to produce summary statistics for all variables in our data set we would type:

```
summarize
```

Hitting ‘Enter’ would produce summary statistics for all of our variables. Even simpler, we can shorten the command to:

```
sum
```

and then hit ‘Enter’. As an example, using our NBA data set we get the following output,

```
. sum
```

Variable	Obs	Mean	Std. dev.	Min	Max
player	0				
position	0				
age	311	26.72669	4.314181	20	41
salary	311	9.848679	10.64244	.954267	44.0664
team	0				
weight	311	214.5145	23.76618	164	290
height	311	77.98392	2.987565	72	87
games	311	329.8875	274.5789	3	1310
minspg	311	22.6147	7.560721	3.5	37.9944
orebsp	311	.8477645	.5996349	0	3.39
drebsp	311	2.954631	1.362276	.4	8.6
trebsp	311	3.801459	1.825524	.5	11.0154
astspg	311	2.263241	1.783535	0	9.4
stlspg	311	.7538795	.3682665	0	2.13125
blockspg	311	.4204017	.3525703	0	2.1625
tovspg	311	1.303802	.7756512	0	4.23333
foulspg	311	1.851306	.6032863	.1	3.2625
pointspg	311	10.20276	5.39261	.8	26.9333
seasons	311	5.961415	4.438663	1	18

Note that the default for Stata is to produce summary statistics for all variables in the data set if no variables are specified. We can see in the above output that the summary statistics for player, position, and team are shown as 0. This is because these variables are not numeric, but strings. If we wanted only the summary statistics for specific variables, we would follow the command `sum`

with the names of the variables for which we want summary statistics. For example, to compute summary statistics for only salary, seasons and pointspg we would type:

```
sum salary seasons pointspg
```

hitting ‘Enter’ we get,

```
. sum salary seasons pointspg
```

Variable	Obs	Mean	Std. dev.	Min	Max
salary	311	9.848679	10.64244	.954267	44.0664
seasons	311	5.961415	4.438663	1	18
pointspg	311	10.20276	5.39261	.8	26.9333

Correlations

In addition to descriptive statistics, a researcher may wish to study the correlation between various pairs of variables. This may be particularly useful when we suspect that high multicollinearity is a problem in our regression analysis, (see Chapter 8 in *Regression Basics*).

Suppose we wish to compute the correlations between the variables age, weight, and height.

This can be done in Stata using the drop-down menus by choosing ‘Statistics’, then ‘Summaries, tables, tests’, then ‘Summary and Descriptive statistics’, and then ‘Pairwise correlations’, and then we would select the variables we want. Again, this is less efficient than using the command line. Here, using the NBA data set, we would type at the command line:

```
cor age weight height
```

Hitting ‘Enter’ we get the following correlation matrix,

```
. cor age weight height
(obs=311)
```

	age	weight	height
age	1.0000		
weight	0.1081	1.0000	
height	-0.0419	0.7252	1.0000

If we had simply typed `cor` with no variable names to follow and hit ‘Enter’ we would get a correlation matrix for all the variables (excluding string variables).

6. Ordinary Least Squares (OLS) Regression

Estimating an OLS Regression

Suppose we wish to estimate an OLS regression using our NBA data set with salary as the dependent variable, and seasons and pointspg (points per game) as our independent variables. This can be accomplished using the drop-down menus by going to ‘Statistics’, ‘Linear models and related’, and then ‘Linear regression’, which will bring up a dialog box where we would choose the ‘Dependent’ and ‘Independent variables’. We would then hit ‘OK’ or ‘Submit’. [Note, the difference between hitting ‘OK’ versus ‘Submit’ is that the former will close the dialog box and the command will be executed by Stata. The latter will *not* close the dialog box, and the command will be executed by Stata. The use of ‘Submit’ may be preferable when we expect to estimate multiple versions of a regression model and we don’t want to repeatedly click through the menus each time.]

Most Stata users would not use the drop-down menus to estimate regression models. Instead, the command line is much more efficient. The basic command for estimating regression models with

Stata is to type regress (or reg for short), and then list the dependent variable and the independent variable(s), and then hit the ‘Enter’ key. For example, we would type the following into the command line:

```
reg salary seasons pointspg
```

Hitting ‘Enter’ we get,

```
. reg salary seasons pointspg
```

Source	SS	df	MS	Number of obs	=	311
Model	17776.1577	2	8888.07884	F(2, 308)	=	157.92
Residual	17334.9436	308	56.2822844	Prob > F	=	0.0000
				R-squared	=	0.5063
				Adj R-squared	=	0.5031
Total	35111.1013	310	113.261617	Root MSE	=	7.5022

salary	Coefficient	Std. err.	t	P> t	[95% conf. interval]	
seasons	.2998411	.1025005	2.93	0.004	.0981514	.5015309
pointspg	1.298562	.0843683	15.39	0.000	1.132551	1.464573
_cons	-5.18771	.9625665	-5.39	0.000	-7.081748	-3.293671

Saving the Predicted Values and Residuals

Once a regression has been estimated, there are a variety of ‘post estimation’ commands that can be used. For example, we can save the predicted values (the \hat{Y} ’s) and the residuals (the e ’s) by submitting the following predict commands,

predict yhat ← this will create a new column of data called yhat with the predicted values for the dependent variable.

predict resid, r ← this will create a new column of data called resid with the predicted residuals for the regression.

Note that the first post-estimation command, `predict yhat`, has nothing following it. Thus, the default in this case is to produce the predicted values for the dependent variable. The second post estimation command, `predict resid, r` includes a comma and then the option `r`, short for residuals. This illustrates how many Stata commands work. Many commands have defaults (including `regress`), that will be executed, and any options that are desired are specified by using a comma and then typing the relevant option. As an example, using our NBA data set and after running the above regression, we can type,

```
predict salhat
```

and then hit ‘Enter’. Next, we can type,

```
predict resid, r
```

and then hit ‘Enter’ again. This will produce a new column of data called `salhat` which are the predicted salaries from the regression, and a new column of data called `resid` which are the residual values from the regression. Checking the Variables window, we see that Stata now lists these new variables and shows the Labels as Fitted values (i.e., the predicted values) and Residuals,

salhat	Fitted values
resid	Residuals

Creating Dummy Variables

In some cases, we would like to create a dummy variable (also known as an indicator variable) to cover categories represented by a string variable (or a numeric variable representing categories)

in our data set. For example, in our NBA data set we have a variable called ‘position’ which is a string variable for a player’s position. For example, the entry ‘C’ stands for the position ‘Center’, ‘PF’ is ‘Power Forward’, and so on. We would like to create a dummy variable that equals 1 if a player is, say, a center, 0 otherwise. Similarly, we can create another dummy variable equal to 1 if a player is a power forward, 0 otherwise. We can do this for the other positions as well. There are several ways we can create these dummy variables in Stata. One is to create dummy variables one at a time with a generate command,

```
generate center = (position=="C")
```

This command illustrates a couple of Stata coding features. Here, the generate command is creating a new variable called center whose value depends on whether the statement in the parentheses is true or not. If it is true, the value is 1, if it is not true, the value is 0. Note that in Stata the single equals sign (=) is used as a ‘set equal to’ operator. The double equals sign (==) is used as a ‘test of equality’. Thus, the code in the parentheses, (position=="C") essentially says, ‘if the entry for position is equal to C’. The double quotes (") are included here in the statement because the entries for position are strings. If the entries were numeric, then no quotes are used in the statement. A process like this could be used to create dummies for the other positions as well. Note that the statement in the parentheses can include other conditions. For example, if we wanted to create a dummy variable that equals 1 if a player is over 30 years old, 0 otherwise, we could type,

```
gen over30 = (age>30)
```

hitting ‘Enter’ would create a new variable named over30 which would equal 1 if a player was over 30 years of age, 0 otherwise.

While the above approach works fine, there are more efficient ways to create dummy variables in Stata. One is to use the `tab variable, gen(stubname)` command. This command combines the `tab` command (short for tabulate), with a `gen` (short for generate) option. This command will first tabulate the categories for *variable*, and then will generate dummy variables that begin with *stubname*, one for each category in *variable*. Thus, for our NBA data set we could type the following code into the command line:

```
tab position, gen(pos)
```

Hitting ‘Enter’ will have Stata perform a tabulation for `position`, and then create dummy variables labeled `pos1`, `pos2`, `pos3`, `pos4`, and `pos5` – one for each position. Here is the output from implementing this command,

```
. tab position, gen(pos)
```

position	Freq.	Percent	Cum.
C	31	9.97	9.97
PF	54	17.36	27.33
PG	68	21.86	49.20
SF	70	22.51	71.70
SG	88	28.30	100.00
Total	311	100.00	

In the variables window we see that five new variables have been created,

Variables ▾ ▴ ✕	
Filter variables here	
Name	Label
blockspg	
tovspg	
foulspg	
pointspg	
seasons	
pos1	position==C
pos2	position==PF
pos3	position==PG
pos4	position==SF
pos5	position==SG

In addition, Stata has created labels for us to define what each dummy variable represents. Thus, `pos1` is dummy for `position==C`, and so on. And, if we open the data viewer window, we can see that five new columns of data have been added, where 1's mean that a player plays a given position, 0's if not,

pos1	pos2	pos3	pos4	pos5
0	1	0	0	0
0	0	1	0	0
0	0	0	1	0
1	0	0	0	0
0	0	0	0	1
0	1	0	0	0
0	0	0	0	1
1	0	0	0	0
0	0	0	1	0
0	0	0	1	0
0	0	0	0	1

We can now include these dummies in a regression model. For example, we can regress salary on seasons, `pointspg` (points per game), and dummies for player positions. The following command could be typed into the command line:

```
reg salary seasons pointspg pos2-pos5
```

Notice that Stata allows us to use shorthand here, `pos2-pos5`, to include the four dummy variables `pos2`, `pos3`, `pos4`, and `pos5`, which works when variables have the same stubname. The dummy `pos1` is excluded and it serves as our base category (see Chapter 6 in *Regression Basics*). The results of this regression are,

```
. reg salary seasons pointspg pos2-pos5
```

Source	SS	df	MS	Number of obs	=	311
Model	17875.9234	6	2979.32056	F(6, 304)	=	52.55
Residual	17235.1779	304	56.6946642	Prob > F	=	0.0000
				R-squared	=	0.5091
				Adj R-squared	=	0.4994
Total	35111.1013	310	113.261617	Root MSE	=	7.5296

salary	Coefficient	Std. err.	t	P> t	[95% conf. interval]	
seasons	.2997645	.1036613	2.89	0.004	.09578	.503749
pointspg	1.318686	.0876209	15.05	0.000	1.146266	1.491106
pos2	.4997257	1.703521	0.29	0.769	-2.85246	3.851911
pos3	-.4280528	1.668379	-0.26	0.798	-3.711085	2.854979
pos4	1.239715	1.628922	0.76	0.447	-1.965675	4.445106
pos5	.2007169	1.594815	0.13	0.900	-2.937556	3.33899
_cons	-5.72158	1.576144	-3.63	0.000	-8.823113	-2.620047

The `tab variable, gen(stubname)` command is much more efficient than creating dummies one at a time, particularly if there are many categories. There is, however, an even more efficient way to include dummy variables in a regression model when the categories are numeric (not string) variables. For example, suppose we wanted to see if the team that a player plays on has an impact on their salary. Do players on, say, the Golden State Warriors, tend to earn more than players on other teams? This possible ‘team effect’ can be explored by creating dummy variables for each team and including them (except one dummy variable for the base case) in our regression model. The variable `team` in the NBA data set is a string variable. We could use the `tab variable, gen(stubname)` method. Alternatively, we could create a new variable, call it `tid`, (short for ‘team id’), that is a numeric team identifier, and then we can take advantage of Stata’s use of ‘factor variables’ to create a dummy variable for each team.

Here are the two steps:

1. Use the `encode` command to create a numeric identifier `tid` for each team:

```
encode(team), gen(tid)
```

Hitting ‘Enter’ will create a new column of data that has a unique, numeric identifier for each team,

22			
foulspg	pointspg	seasons	tid
1.9625	12.4875	8	ORL
1.53333	7.53333	3	IND
1.9	4.7	1	BOS
2.18571	14.0357	14	ATL
1.67692	10.1923	13	UTH
1.3	8.2	1	OKC
1.75	6.175	4	LAL
2.37	6.89	10	PHX
1.76471	11.0118	17	GSW
2.175	19.675	8	MIN
1.3	6.63333	3	POR
1.8	19.3	1	MIN
1	3.1	1	WAS
2.14167	8.70833	12	LAC
2.14286	10.6857	14	BOS

Notice that the entries are blue and appear to be strings. These are actually ‘value labels’ created automatically by Stata. Behind this blue value label is a numeric entry. For example, by clicking on the first entry, [ORL](#), we see in the value box above that this team (the Orlando Magic) has been assigned the numeric value 22.¹ To see the numbers that were assigned to all the teams, we can use the `labelbook` command for `tid`,

```
labelbook tid
```

Hitting ‘Enter’ produces the following table in the output viewer,

¹ Note that creating value labels for numeric categorical variables can sometimes be very useful. A simple online search of ‘how to create value labels in Stata’ will lead to instructions.

```
. labelbook tid
```

```
Value label tid
```

Values	Labels
Range: [1,30]	String length: [3,3]
N: 30	Unique at full length: yes
Gaps: no	Unique at length 12: yes
Missing .*: 0	Null string: no
	Leading/trailing blanks: no
	Numeric -> numeric: no


```
Definition
  1  ATL
  2  BKN
  3  BOS
  4  CHA
  5  CHI
  6  CLE
  7  DAL
  8  DEN
  9  DET
 10  GSW
 11  HOU
 12  IND
 13  LAC
 14  LAL
 15  MEM
 16  MIA
 17  MIL
 18  MIN
 19  NOP
 20  NYK
 21  OKC
 22  ORL
 23  PHI
 24  PHX
 25  POR
 26  SAC
 27  SAS
 28  TOR
 29  UTH
 30  WAS

Variables:  tid
```

```
.
```

2. Use the `i.variable` factor to include dummy variables in the regression.

```
reg salary seasons pointspg i.tid
```

Hitting ‘Enter’ produces the following regression,

```
. reg salary seasons pointspg i.tid
```

Source	SS	df	MS	Number of obs	=	311
Model	19731.6181	31	636.503809	F(31, 279)	=	11.55
Residual	15379.4832	279	55.1235957	Prob > F	=	0.0000
				R-squared	=	0.5620
				Adj R-squared	=	0.5133
Total	35111.1013	310	113.261617	Root MSE	=	7.4245

salary	Coefficient	Std. err.	t	P> t	[95% conf. interval]
seasons	.2345514	.1052268	2.23	0.027	.0274121 .4416907
pointspg	1.388683	.0869892	15.96	0.000	1.217444 1.559921
tid					
BKN	2.016734	4.310772	0.47	0.640	-6.469034 10.5025
BOS	6.199893	2.985673	2.08	0.039	.3225868 12.0772
CHA	-.0545089	3.044178	-0.02	0.986	-6.046984 5.937966
CHI	2.875136	3.395835	0.85	0.398	-3.809576 9.559849
CLE	2.636765	2.973054	0.89	0.376	-3.215701 8.48923
DAL	-3.416706	3.95302	-0.86	0.388	-11.19824 4.364826
DEN	7.615436	3.057628	2.49	0.013	1.596486 13.63439
DET	3.35717	3.188888	1.05	0.293	-2.920166 9.634505
GSW	4.93673	3.034301	1.63	0.105	-1.036302 10.90976
HOU	3.315463	3.712385	0.89	0.373	-3.992378 10.6233
IND	5.560287	3.965204	1.40	0.162	-2.24523 13.3658
LAC	-1.758193	3.535746	-0.50	0.619	-8.71832 5.201934
LAL	6.254707	2.880352	2.17	0.031	.5847251 11.92469
MEM	.5087913	3.276103	0.16	0.877	-5.940228 6.957811
MIA	2.649466	3.183959	0.83	0.406	-3.618168 8.9171
MIL	6.795449	3.721127	1.83	0.069	-.5296021 14.1205
MIN	2.713182	3.102713	0.87	0.383	-3.394518 8.820882
NOP	2.391313	3.278738	0.73	0.466	-4.062893 8.845519
NYK	.8181806	3.183512	0.26	0.797	-5.448572 7.084933
OKC	2.570755	3.278844	0.78	0.434	-3.883659 9.025169
ORL	2.588742	3.039313	0.85	0.395	-3.394155 8.571639
PHI	4.769941	2.98747	1.60	0.111	-1.110903 10.65078
PHX	3.057365	2.849216	1.07	0.284	-2.551325 8.666054
POR	5.21822	3.046213	1.71	0.088	-.7782609 11.2147
SAC	2.091103	3.53272	0.59	0.554	-4.863068 9.045273
SAS	6.115282	2.943132	2.08	0.039	.3217164 11.90885
TOR	7.47522	2.886977	2.59	0.010	1.792198 13.15824
UTH	6.515725	2.893741	2.25	0.025	.8193873 12.21206
WAS	5.169197	3.277483	1.58	0.116	-1.282539 11.62093
_cons	-9.43949	2.37396	-3.98	0.000	-14.11264 -4.766342

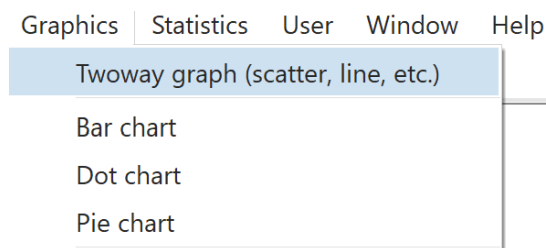
The use of the `i.tid` factor variable produces a dummy for each `tid`, and then includes them in the regression. Note that the dummies for each `tid` are not retained as new variables in our variables list. This is useful since having an additional 30 dummy variables (one for each team)

in our data set may not be desirable. Note also that Stata uses the value labels for each `tid` entry in the regression output.

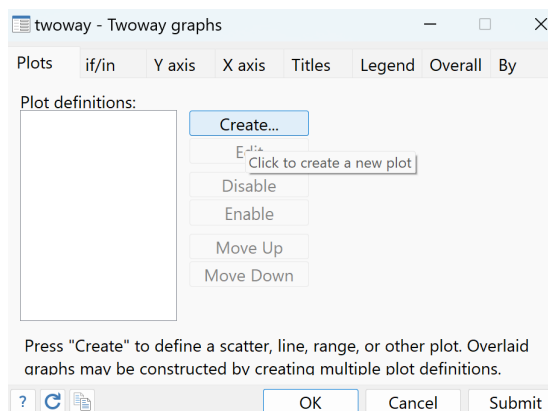
7. Creating a Scatterplot with Reference Line

It may be useful to create a scatterplot for a pair of variables and include a reference line in the graph. This is a simple way of visually understanding the relationship between two variables, and it may also be helpful for visually checking for heteroskedasticity (see Chapter 8 in *Regression Basics*). This can be done using Stata's drop-down menus, or by typing a command directly into the command line. We will start using the drop-down menus, and then focus on using the command line.

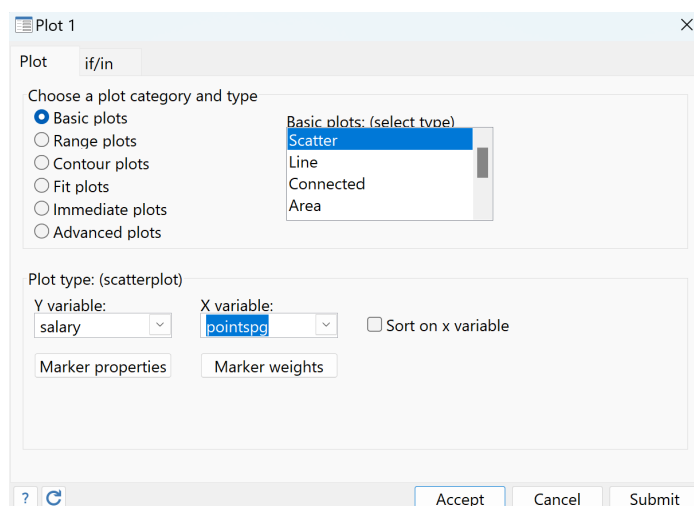
Working with our NBA data set, on the main menu we can choose 'Graphics', then 'Twoway graph (scatter, line, etc.)'



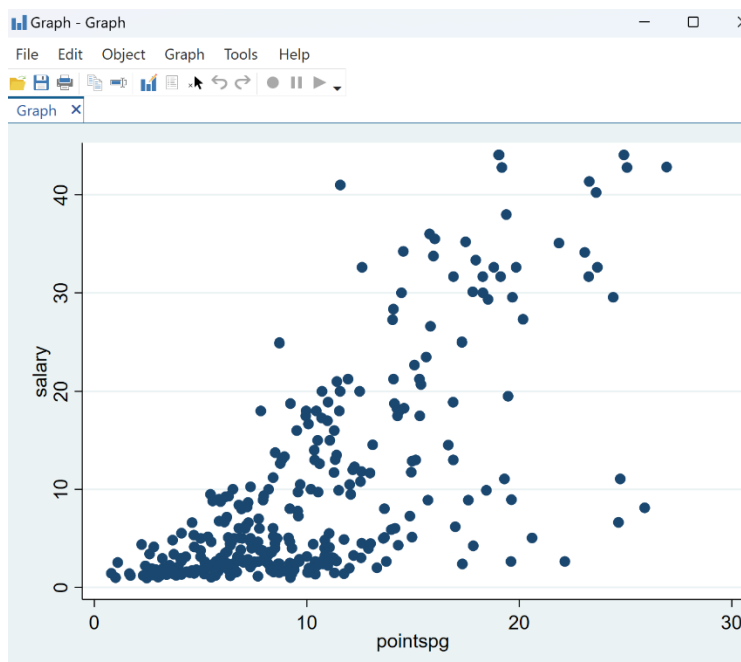
This will bring up a dialog box,



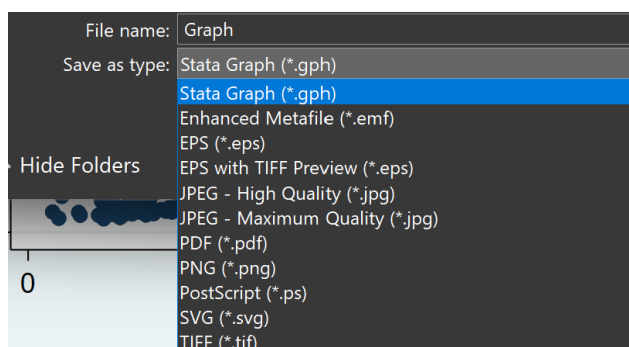
Click 'Create' and a new dialog box appears,



We will use the default choices here, 'Basic plots', and 'Scatter'. We can choose salary as the Y variable, and pointspg as the X variable. Click 'Submit', and the following graph appears in a separate window,



Choosing ‘File’ and then ‘Save as...’ allows the user to give the graphic file a name and location where it will be saved. The default format is to save the files as a ‘.gph’, which is a Stata graphics file format. Using the drop-down menus on the ‘save’ screen, different file types can be chosen (e.g., .jpg, .png, etc.),



Returning to the main Stata screen, we can see in the output window the command that was used to create the scatterplot,

```
twoway (scatter salary pointspg)
```

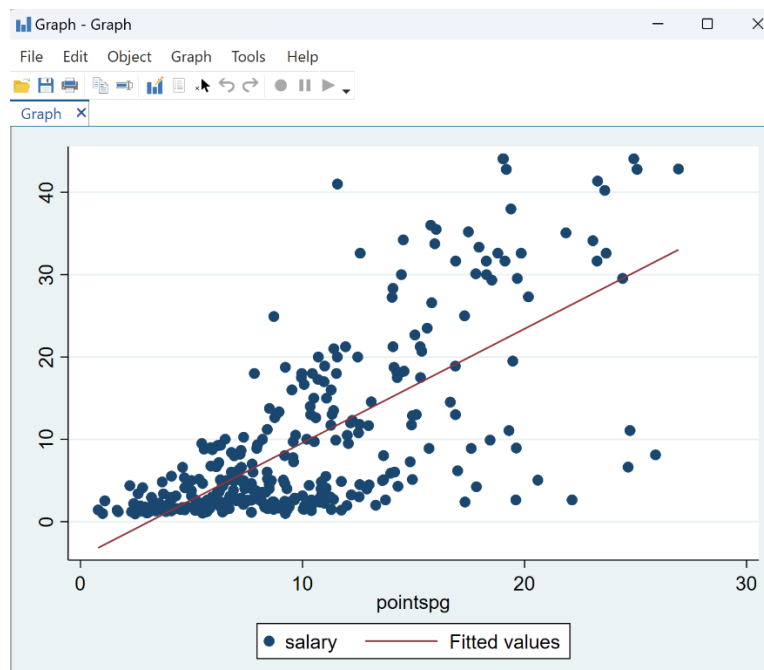
Typing this command into the command line and hitting ‘Enter’ would produce the same scatterplot. In fact, since scatter plots are so common, Stata will allow an even simpler command,

```
scatter salary pointspg
```

To add an OLS fitted line to the above graph, we can use the following command in the command line,

```
twoway (scatter salary pointspg) (lfit salary pointspg)
```

Where `lfit` stands for ‘line fit’. Hitting ‘Enter’, we get a new graph that includes an OLS fitted line,



Alternatively, we could use the following command,

```
scatter salary pointspg || lfit salary pointspg
```

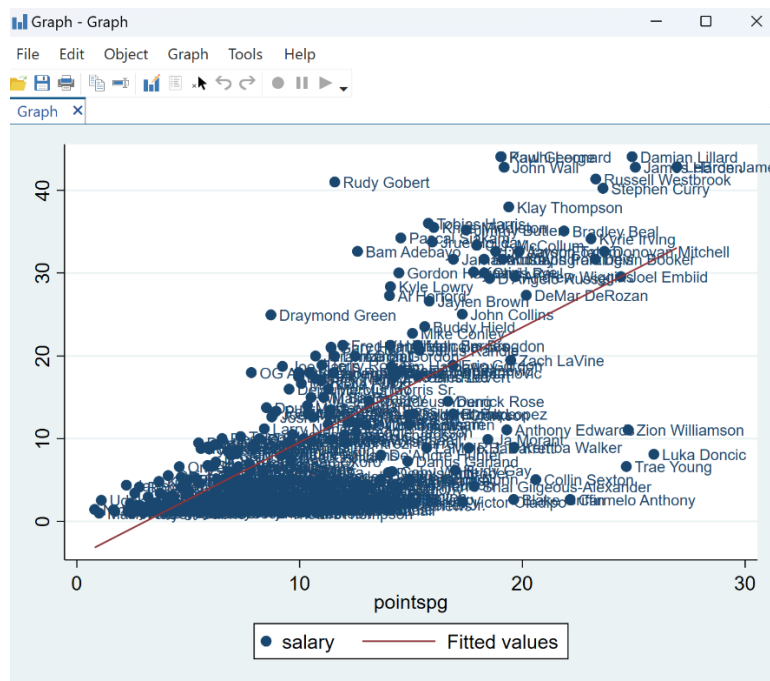
The two vertical lines, ‘||’ tell Stata that we are stacking a line fit on top of the scatter plot.

(Note that Stata allows us to add even more layers to our graph if we wish, just simply add the two vertical lines, ‘||’, again and then the code for the next layer.)

If we desire to add labels to the markers in a plot, we can use the `,mlabel()` option to the scatter command. For example, adding player names as markers to our previous graph could be done using,

```
scatter salary pointspg, mlabel(player)||lfit salary pointspg
```

The resulting graph is,



In this case, the labels are too big and make the graph a bit of a mess, but they may help us find influential observations (e.g., Damian Lillard, Luka Doncic, or Trae Young).

Another useful scatterplot is one where the residuals from a regression estimation are plotted against the predicted (also known as fitted) values for the dependent variable. As discussed in Chapter 8 of *Regression Basics*, this is a common way to visually check for heteroskedasticity.

For example, we can run the regression,

```
reg salary seasons pointspg
```

Then use the post-estimation commands to save the predicted values (`salhat`), and the residuals (`resid`),

```
predict salhat
```

And,

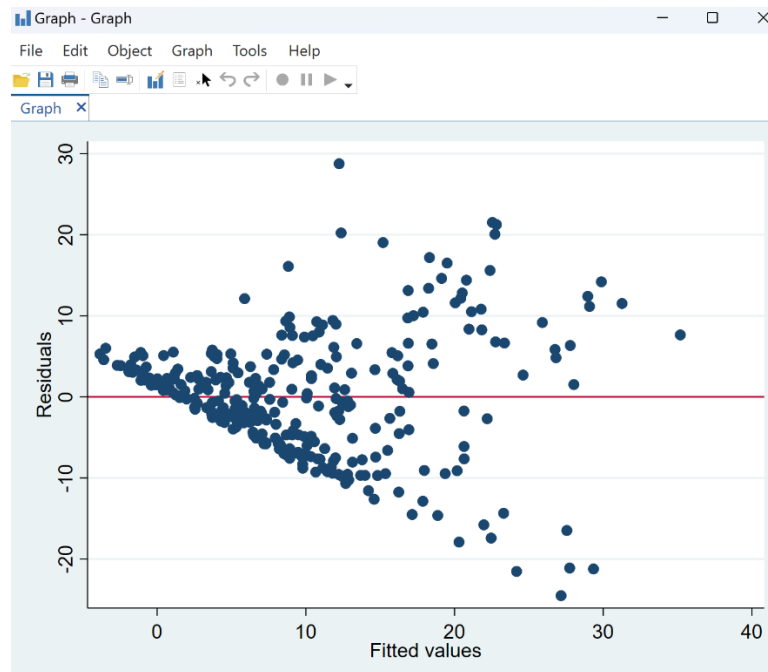
```
predict resid, r
```

Now we can create a scatterplot with the residuals on the vertical axis and fitted values on the horizontal axis, and we can add a reference line at zero on the vertical axis,

```
scatter resid salhat, yline(0)
```

Where the `, yline(0)` option tells Stata to add a horizontal line at 0.

Hitting 'Enter' we get,



Since a ‘residuals versus fitted’ plot is so common, Stata allows for a simpler post-estimation command to create this graph. We, again, run the regression,

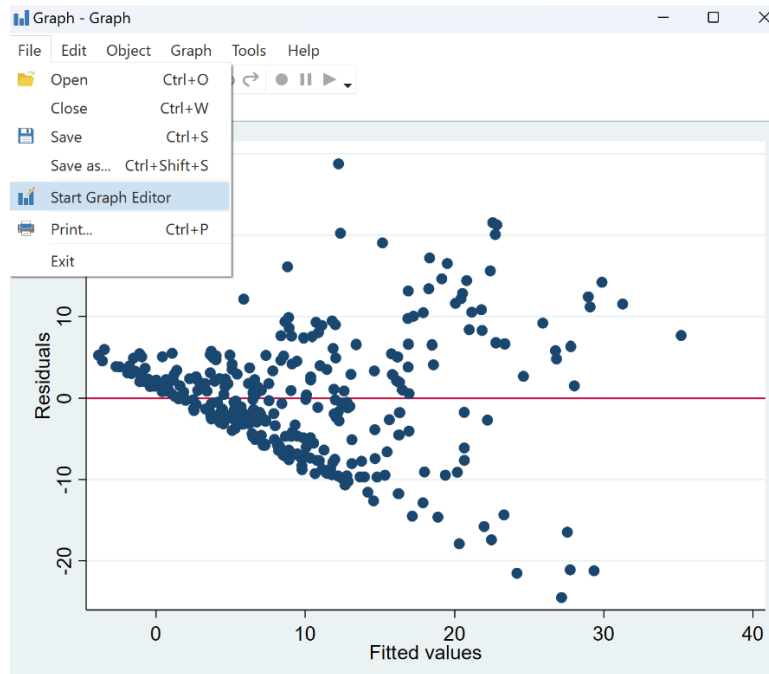
```
reg salary seasons pointspg
```

Then afterwards use the command,

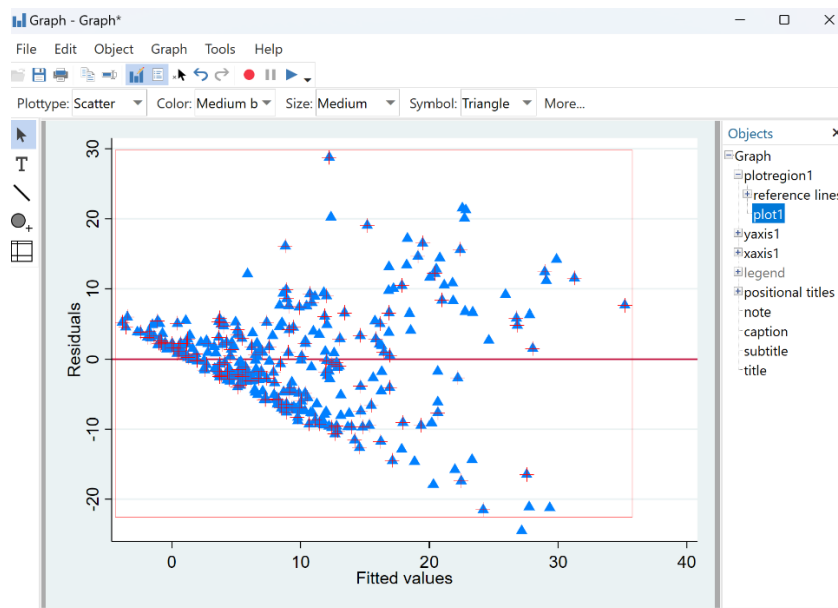
```
rvfplot, yline(0)
```

This will produce the same ‘residuals versus fitted’ plot as seen above. As is evident from the graph, we appear to have heteroskedasticity as the spread of the dots gets wider as the predicted (i.e., fitted) value increases.

Stata’s graphic capabilities are enormous. Many other types of graphs are possible. Furthermore, just about every element in a graph can be customized. The user can change the colors, types, and the size of observation markers, lines, etc., as well as change axis titles and the font used in the graph. To edit a Stata-created graph, choose ‘File’, and then ‘Start Graph Editor’,



Then simply click on an element in the graph and use the tool bars to make changes. For example, we can click on one of the observation markers and change the color and symbol used for all the markers,



Stata has a very useful website that shows how to create all sorts of graphs, see:

<https://www.stata.com/support/faqs/graphics/gph/stata-graphs/>.

8. Breusch-Pagan Test for Heteroskedasticity and OLS with Robust Standard Errors

As discussed in Chapter 8 of *Regression Basics*, the presence of heteroskedasticity creates problems as the regular OLS coefficient standard errors used for inference tests are no longer legitimate. We saw in our scatterplot above that there seems to be evidence of heteroskedasticity in the NBA salary regression. To be more careful about our conclusion, we can conduct the Breusch-Pagan test for heteroskedasticity, and if present then use robust standard errors to fix the problem for the standard errors and allow for legitimate inference tests. As a benchmark, below is the simple OLS regression results for our NBA data set with `salary` as the dependent variable, and `seasons` and `pointspg` (points per game) as our independent variables,

```
. reg salary seasons pointspg
```

Source	SS	df	MS	Number of obs	=	311
Model	17776.1577	2	8888.07884	F(2, 308)	=	157.92
Residual	17334.9436	308	56.2822844	Prob > F	=	0.0000
				R-squared	=	0.5063
				Adj R-squared	=	0.5031
Total	35111.1013	310	113.261617	Root MSE	=	7.5022

salary	Coefficient	Std. err.	t	P> t	[95% conf. interval]	
seasons	.2998411	.1025005	2.93	0.004	.0981514	.5015309
pointspg	1.298562	.0843683	15.39	0.000	1.132551	1.464573
_cons	-5.18771	.9625665	-5.39	0.000	-7.081748	-3.293671

The Breusch-Pagan test for heteroskedasticity is implemented by a post-estimation command.

After running the OLS regression model, we simply type in the command line:

```
hettest
```

Hitting the ‘Enter’ key executes the command and we get,

```
. hettest

Breusch-Pagan/Cook-Weisberg test for heteroskedasticity
Assumption: Normal error terms
Variable: Fitted values of salary

H0: Constant variance

      chi2(1) = 145.28
Prob > chi2 = 0.0000
```

We see that the Chi-Square statistic (145.284) and the associated p-value (0.0000) strongly rejects the null hypothesis of ‘constant variance’ (i.e., homoskedasticity). Given clear evidence of heteroskedasticity, we now can re-estimate the regression model, but use robust standard errors. This is done by simply adding the option , vce(robust) to our regression command, `reg salary seasons pointspg, vce(robust)`

The regression output with robust standard errors is,

```
. reg salary seasons pointspg, vce(robust)

Linear regression              Number of obs   =       311
                             F(2, 308)         =       85.79
                             Prob > F           =       0.0000
                             R-squared          =       0.5063
                             Root MSE       =       7.5022
```

salary	Coefficient	Robust std. err.	t	P> t	[95% conf. interval]	
seasons	.2998411	.1177799	2.55	0.011	.0680861	.5315962
pointspg	1.298562	.119127	10.90	0.000	1.064156	1.532968
_cons	-5.18771	.9005532	-5.76	0.000	-6.959725	-3.415695

Notice that the estimated coefficients are the same (as expected), but the standard errors, t statistics, and associated p -values have changed. This correction to the standard errors allows for legitimate inference tests for the estimated coefficients.²

9. Testing for Multicollinearity: Variance Inflation Factor (VIF)

As discussed in Chapter 8 of *Regression Basics*, high multicollinearity can be a nuisance in multiple regression models. The variance inflation factor (VIF) is a common measure used to test for high multicollinearity. To illustrate how to produce VIF estimates with Stata, we will add an additional independent variable to our NBA salary model, namely ‘games,’ which is the career number of games a player has played in the NBA. This variable would likely be highly correlated with ‘seasons,’ which is already in our regression model. Running the regression we get,

```
. reg salary seasons pointspg games
```

Source	SS	df	MS	Number of obs	=	311
Model	18269.597	3	6089.86566	F(3, 307)	=	111.01
Residual	16841.5043	307	54.8583202	Prob > F	=	0.0000
				R-squared	=	0.5203
				Adj R-squared	=	0.5156
Total	35111.1013	310	113.261617	Root MSE	=	7.4066

salary	Coefficient	Std. err.	t	P> t	[95% conf. interval]
seasons	-.7243241	.3561657	-2.03	0.043	-1.425159 -.0234892
pointspg	1.176407	.0927192	12.69	0.000	.993962 1.358853
games	.0181333	.0060462	3.00	0.003	.0062361 .0300305
_cons	-3.817867	1.054377	-3.62	0.000	-5.892586 -1.743147

Notice that, while all three estimated coefficients have a p -value of less than 0.05, suggesting that they are statistically significant, the coefficient to seasons is unexpectedly negative. [Note

² As noted in Chapter 8 of *Regression Basics*, (see footnotes 31 and 32), there are several ways that the standard errors can be adjusted. Stata’s default is to use the ‘HC1’ method. R’s default is ‘HC2’, whereas SPSS lets the user choose between ‘HC1’, ‘HC2’, and ‘HC3’. The results for the above regression are very similar for each method.

that this regression also suffers from heteroskedasticity, but we will ignore this issue for now.]

An unexpected sign is one of the indicators that we may have high multicollinearity present in the model. In order to have Stata produce the VIF values, we simply add a post-estimation

command: `estat vif`

Hitting ‘Enter’ produces the following results,

```
. estat vif
```

Variable	VIF	1/VIF
games	15.57	0.064207
seasons	14.12	0.070806
pointspg	1.41	0.707854
Mean VIF	10.37	

We see that both seasons and games have VIF values greater than our benchmark value of 10, indicating high multicollinearity.

10. Testing for Autocorrelation and Prais-Winsten Estimation

Autocorrelation, (also known as serial correlation) in the error structure is a common problem with time series data. When present the variance of the residuals (the e_t 's) tends to underestimate the variance of the true population's errors (the u_t 's). Since the standard errors of the parameter estimates are based on the variance of the residuals, this means the t statistics and their associated p-values are invalid. To illustrate how to test and correct for autocorrelation we will work with our alcoholic beverages sales example using the “alc_sales.csv” data set, (available on the companion website for *Regression Basics*). Our sample regression model was, (see Chapter 7),

$$\ln alc_sales_t = a + b_1(tindex) + b_2q2_t + b_3q3_t + b_4q4_t + e_t$$

Where $\ln alc_sales_t$ is the natural log of alcoholic beverages sales in the U.S., $tindex$ is a time index, and $q2_t$, $q3_t$, and $q4_t$, were quarter dummies to pick up seasonal effects.

Graphical Detection of Autocorrelation

As discussed in Chapter 8, one method of detecting autocorrelation is to plot the errors from an OLS regression and check for a pattern. This can be done in this case by estimating the above regression, saving the residuals, and then creating a connected line graph of the residuals over the time periods. When working with time series data, the first thing to do, before running a regression, is to tell Stata that the data are ordered as a time series. This is done using the `tsset` command and then providing the variable that serves as the time index:

```
tsset tindex
```

Hitting ‘Enter’, Stata responds with:

```
. tsset tindex
      Time variable: tindex, 1 to 123
             Delta: 1 unit
```

We see that Stata has determined our time index (`tindex`) ranges from 1 to 123. (The ‘Delta: 1 unit’ part simply says that Stata has determined that the time index increases by one unit from one observation to the next.) Now that we have declared the data set as a time series, this unlocks some very useful features (as we will see).

Estimating our model we have,

```
. reg lna1c_sales tindex q2 q3 q4
```

Source	SS	df	MS	Number of obs	=	123
Model	16.5201767	4	4.13004418	F(4, 118)	=	2850.76
Residual	.170952912	118	.001448753	Prob > F	=	0.0000
				R-squared	=	0.9898
				Adj R-squared	=	0.9894
Total	16.6911296	122	.136812538	Root MSE	=	.03806

lna1c_sales	Coefficient	Std. err.	t	P> t	[95% conf. interval]	
tindex	.0100028	.0000967	103.46	0.000	.0098114	.0101943
q2	.1176668	.0096684	12.17	0.000	.0985208	.1368128
q3	.1383091	.0096698	14.30	0.000	.1191603	.157458
q4	.2444325	.0097486	25.07	0.000	.2251276	.2637374
_cons	8.372448	.0090285	927.34	0.000	8.354569	8.390326

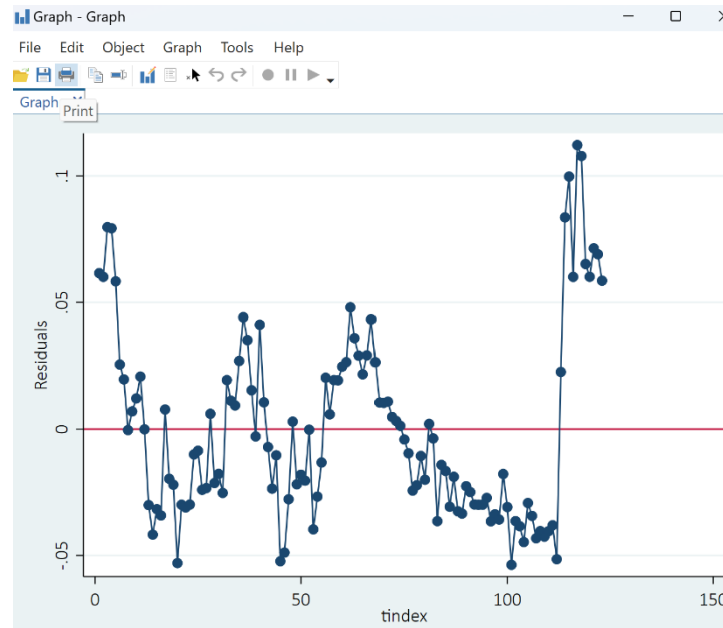
Now we use the post-estimation command to save the residuals,

```
predict resid, r
```

With the residuals now saved as the variable `resid`, we can create a connected graph with the residuals on the vertical axis and our time index on the horizontal axis. This can be done by using the drop-down menu for ‘Graphics’, then ‘Twoway graph’, and then selecting ‘Connected’ in the dialog box, (or equivalently ‘Graphics’, ‘Time-series graphs’ then ‘Line plots’). Or, more simply, we can type in the command line:

```
twoway (connected resid tindex), yline(0)
```

This will produce a connected graph, along with a reference line at 0,



Viewing the graph, (which is the same as Figure 8.6a in *Regression Basics*), we see evidence of positive autocorrelation, with long strings of positive errors, then long strings of negative errors. In fact, from periods 82 to 112 we have thirty-one negative errors in a row – this is clearly *not* a random pattern.³

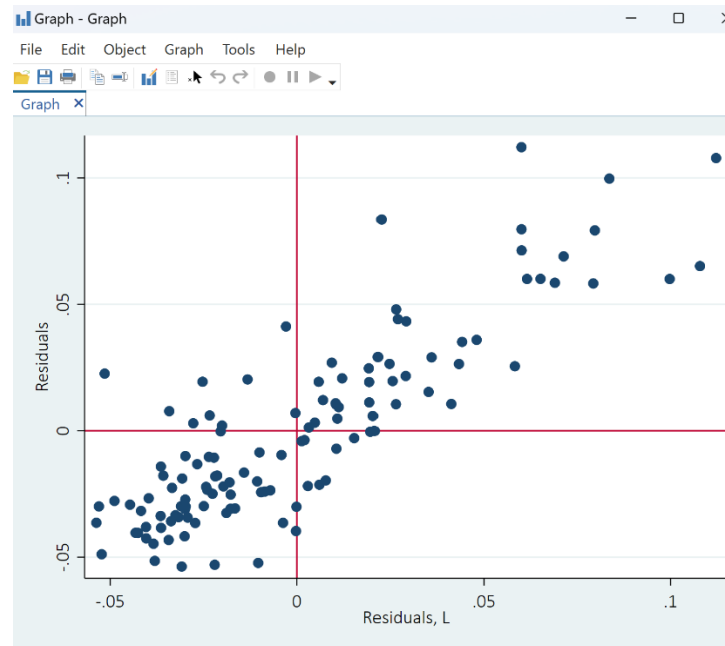
An alternative graph plots the current period residuals against the previous period's residuals (see Figure 8.6b). This is easily accomplished now since we used the `tsset` command. This is best illustrated with the appropriate `scatterplot` command,

```
scatter resid l.resid, yline(0) xline(0)
```

The above line of code illustrates one of the benefits of using the `'tsset'` command to declare a times series. For example, we can now use `'lag'` operators. The prefix `'l.'` tells Stata to use the

³ Note that, since we `'tsset'` the data set, we can also use the following code to graph the residuals across the time periods: `tsline resid, yline(0)`

lagged value of `resid` for the horizontal axis variable. We have also added horizontal and vertical reference lines at 0. Hitting ‘Enter’ the resulting graph (which is the same as Figure 8.6b) is,



We see that the general pattern of upward sloping dots suggests positive autocorrelation.

Durbin-Watson Test of Autocorrelation

As described in Chapter 8, the Durbin-Watson statistic is often used to detect autocorrelation.

Stata can produce this test statistic as a post-estimation command, so long as we have used the `tsset` command before the regression is run. The command for producing the Durbin-Watson statistic is,

`dwstat`

Hitting ‘Enter’, we get,

```
. dwstat
```

```
Durbin-Watson d-statistic( 5, 123) = .2802168
```


The values in the parentheses, 5 and 123, are the number of estimated coefficients (including the constant term) and the number of observations used in the regression, respectively. The value of about 0.28 for the Durbin-Watson statistic, when compared to the appropriate Durbin-Watson table (see Chapter 8 in *Regression Basics*), indicates evidence of positive correlation.

Prais-Winsten Estimation

One possible solution to the problem of autocorrelation is to use the Prais-Winsten estimator.

This estimator uses an iterative approach to estimate the autocorrelation coefficient ($\hat{\rho}$) and then using this estimate transforms the model, hopefully purging it of autocorrelation, and then using OLS on the transformed model. This model is easily estimated in Stata, again, so long as we have used the `tsset` command to declare a time series data set. The following command produces the Prais-Winsten estimation,

```
prais lnalc_sales tindex q2 q3 q4
```

Thus, we simply replace the `reg` command with `prais` command. Hitting ‘Enter’ produces the following results (including the command to `tsset` the data),

```
. tsset tindex
```

```
Time variable: tindex, 1 to 123
Delta: 1 unit
```

```
. prais lnalc_sales tindex q2 q3 q4
```

```
Iteration 0: rho = 0.0000
Iteration 1: rho = 0.8559
Iteration 2: rho = 0.8604
Iteration 3: rho = 0.8606
Iteration 4: rho = 0.8606
Iteration 5: rho = 0.8606
```

Prais-Winsten AR(1) regression with iterated estimates

Source	SS	df	MS	Number of obs	=	123
Model	11.6041575	4	2.90103939	F(4, 118)	=	7583.36
Residual	.045141275	118	.000382553	Prob > F	=	0.0000
				R-squared	=	0.9961
				Adj R-squared	=	0.9960
Total	11.6492988	122	.095486056	Root MSE	=	.01956

lnalc_sales	Coefficient	Std. err.	t	P> t	[95% conf. interval]	
tindex	.0099966	.000309	32.35	0.000	.0093847	.0106085
q2	.1176322	.0032717	35.95	0.000	.1111533	.1241112
q3	.138353	.003793	36.48	0.000	.1308419	.1458642
q4	.2463764	.0033077	74.49	0.000	.2398263	.2529265
_cons	8.377875	.0226907	369.22	0.000	8.332942	8.422809
rho	.8606231					

```
Durbin-Watson statistic (original) = 0.280217
Durbin-Watson statistic (transformed) = 2.032605
```

Stata shows us the iteration history as $\hat{\rho}$ is being estimated, the regression results, and at the bottom of the table we see the original Durbin-Watson value, and the one after using $\hat{\rho}$ to transform the model.

11. Studentized Residuals and Leverage

In Chapter 8 of *Regression Basics* there is a discussion on influential observations. These are observations that contain the ingredients for having a strong impact on the coefficients of a regression estimation. The two main ingredients are being a strong outlier and having strong leverage. The measure called “studentized residuals” was used to detect strong outliers. This

measure essentially looks for observations that have unusually large residuals (e_i 's). A studentized residual greater than 3 in absolute terms generally indicates a strong outlier.

The second ingredient for being an influential observation is having strong leverage. As noted in Chapter 8, this essentially means that an observation's value for one or more of the independent (X_i) variables is unusually large or small, compared to the other observations. In cases where leverage is more than 3 times the mean value for leverage for all the observations, these may indicate strong leverage.

Computing studentized residuals and leverage values can be done using post-estimation commands. For example, using the NBA data, we can estimate a regression with salary as the dependent variable and seasons and pointspg as the independent variables,

```
reg salary seasons pointspg
```

Hitting 'Enter' produces the regression results. We can then use the following two post-estimation commands,

```
predict rstud, rstudent
```

And,

```
predict lev, leverage
```

Hitting 'Enter' following each of these commands will create two new variables in our data set, rstud (studentized residuals), and lev (leverage values); a partial listing is shown below,

rstud	lev
.8776698	.0041496
-.3836299	.0049021
.389943	.0086989
1.345865	.0138168
-.2588559	.0124714
-.290281	.0072473
.6963054	.0051342
-.3921358	.0090893
-1.565258	.0250822
.9097754	.0132583
-.2377516	.0053251

Using our ‘rule of thumb’ values for studentized residuals and leverage values, we can look for observations where the absolute value of `rstud` is greater than or equal to 3, and `lev` values greater than 3 times the mean value for `lev`, which is $3 \times 0.00965 = 0.029$. To see if any observations meet these two criteria, we can use a `list` command, which will list values of variables that meet a given criteria. For example, we can have Stata list the entries for the variables `player`, `rstud`, and `lev` listed if the absolute value of `rstud` is greater than 3 *and* `lev` is greater than 0.029. The command would look like this,

```
list player rstud lev if abs(rstud) > 3 & lev > 0.029
```

Stata returns no results since no player achieves both criteria. If we lower the thresholds to 2.5 and 0.025, respectively, we get,

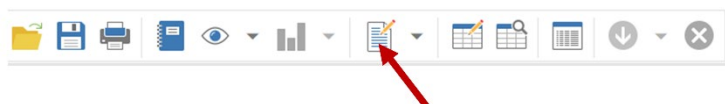
	player	rstud	lev
194.	Luka Doncic	-2.924642	.0410229
286.	Trae Young	-2.900236	.0359251

Thus, we see that Luka Doncic and Trae Young both exceed the new leverage threshold, and both come very close to the studentized residual threshold.

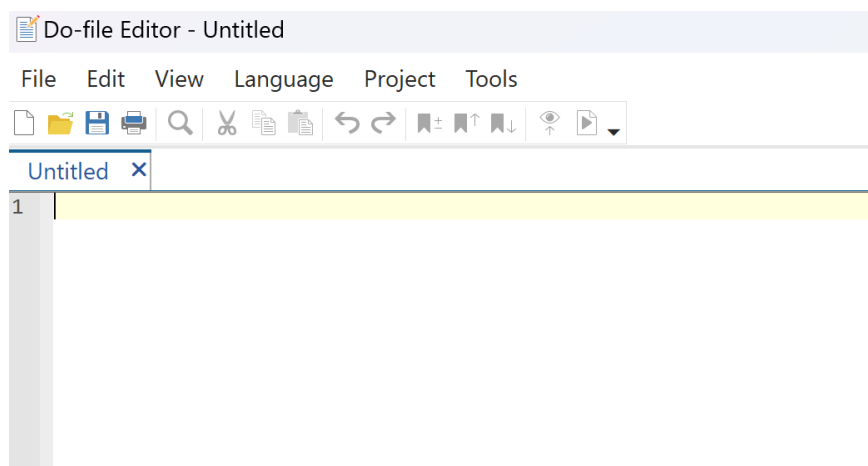
12. Do-files

Thus far we have seen that we can carry out commands with Stata by using the drop-down menus, or by typing commands directly into the command line. A third way, one that is used by most Stata users, is to create a ‘do-file’, which is a text file that contains a list of various commands that we want Stata to carry out. This approach is preferable for several reasons. First, it allows us to keep a record of the steps we have taken in an analysis, which is important for reproducibility. Second, it allows for easier debugging. If we encounter an error in our analysis or code, or if we want to make some simple changes like using the natural log of a variable, we can simply edit the do-file, and then resubmit it to Stata.

Stata has a built-in do-file editor that is initiated by clicking on the appropriate icon below the main menus; see the red arrow below,



This will bring up a new do-file where the user can begin typing commands.⁴



⁴ Note that any text editor can also be used to create a do-file. Completed do-files should be saved with a .do extension.

Once the do-file is completed, it can be saved with a name, e.g., `project1.do`, and then run.

When typing in the do-file, it is highly recommended that the user provide comments about what steps are being taken. This makes it much easier to follow what is being done in the do-file if you pause your research and then later come back to it. Comments are identified with an asterisk (*) and are automatically colored green by Stata's do-file editor. Each command line is ended by hitting the 'Enter' key. When commands/options are being typed Stata's do-file editor will sometimes offer suggestions for completing the command/option. In order to keep a log of all the steps carried out by Stata, we can start a log file by typing a command near the top of the do-file like: `log using project1.log, replace`

Note that using the `.log` extension will produce a text file that can be opened by any text editor, not including any extension will produce a default `.smc1` file that must be opened with Stata. The option `,replace` tells Stata to write over any pre-existing log file named `project1`. Without this option, if you make a change to the do-file and re-run it, you will get an error, colored in red, that the `project1.log` file already exists, and the program will not complete. Of course, as always, save your do-file frequently.

To run a do-file, the user can simply open the Stata data set and type: `do project1` into the command line.⁵ So long as the Stata data set is located in the same location as the do-file, Stata will locate the do-file and execute it. If the do-file is located elsewhere, then a path to the do-file will need to be added. It is best to simply create a folder, such as `Project1`, and put the Stata data set and do-file in the folder. Once the Stata data set is opened, the `Project1` folder becomes the default location and Stata will look there for any do-files.

⁵ Alternatively, click 'File' on Stata's main menu, then select 'Do....' and navigate to the do-file.

If a do-file is open in Stata's do-file editor, then it can also be run by clicking the arrow button icon below the do-file editor's main menu,



Running a do-file from the editor window is often a good method as errors that may come up in the do-file program may be quickly spotted, fixed, and then the do-file can be re-run.

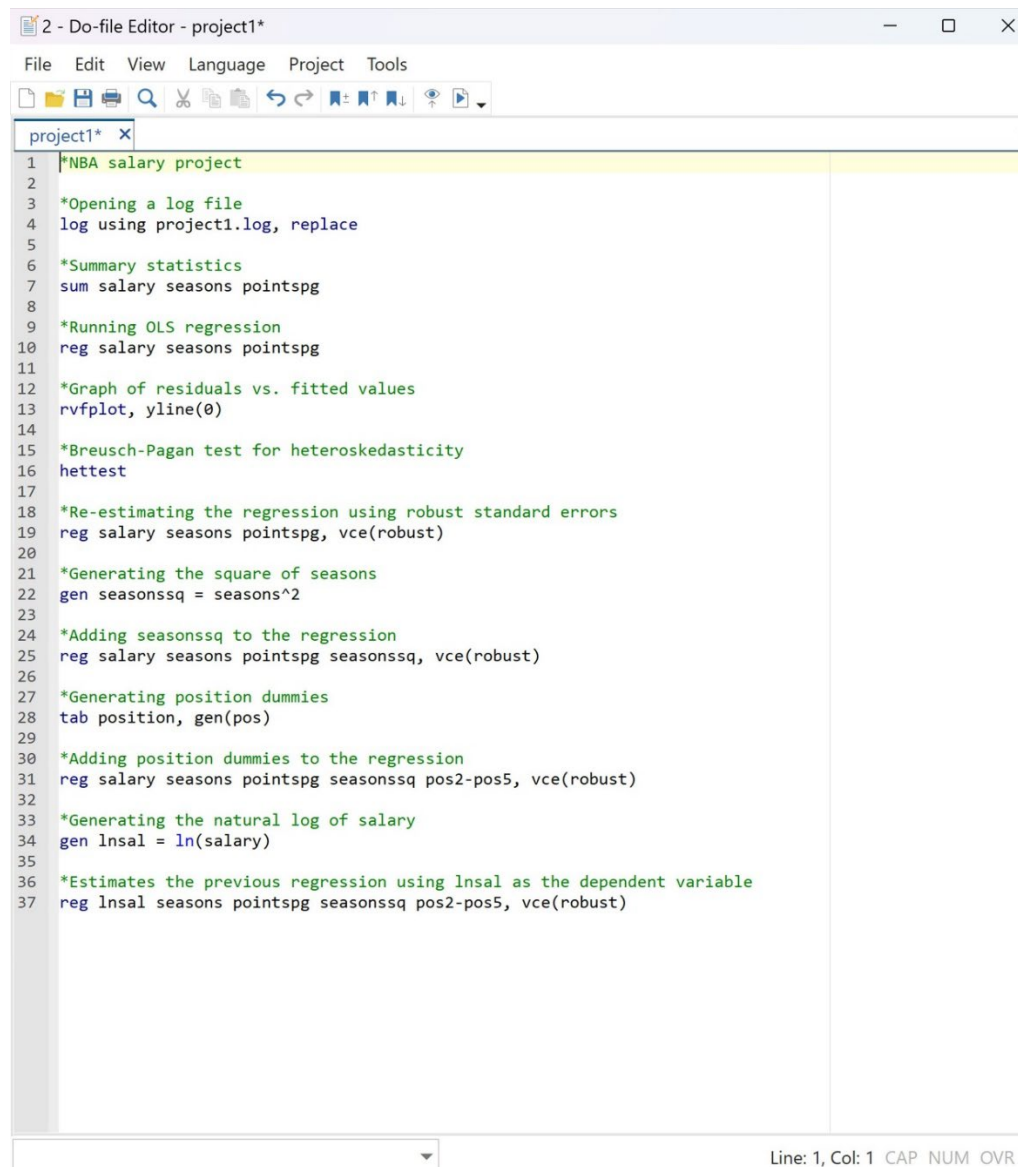
The progress of a do-file execution can be seen in Stata's main output viewer (and will be recorded on the `.log` file).

As an example of how to use a do-file, we will create one that does the following with our NBA data set:

1. Starts a log file called `project1.log`
2. Computes summary statistics for `salary`, `seasons`, and `pointspg`
3. Estimates an OLS regression of `salary` on `seasons` and `pointspg`
4. Creates a graph of the residuals vs. the fitted values (with a reference line at 0) to check for visual evidence of heteroskedasticity (the graph will appear in a separate window).
5. Runs a Breusch-Pagan test for heteroskedasticity (which will confirm its presence).
6. Re-estimates the original regression, but with robust standard errors.
7. Generates the squared value of `seasons` and adds it to the previous regression.
8. Generates player position dummies and adds them to the previous regression (with 'Center' being the base case).

9. Generates the natural log of salary and uses this dependent variable in the previous regression.

A copy of the do-file that will carry out all of these commands is presented below,



```

2 - Do-file Editor - project1*
File Edit View Language Project Tools
project1* x
1 *NBA salary project
2
3 *Opening a log file
4 log using project1.log, replace
5
6 *Summary statistics
7 sum salary seasons pointspg
8
9 *Running OLS regression
10 reg salary seasons pointspg
11
12 *Graph of residuals vs. fitted values
13 rvfplot, yline(0)
14
15 *Breusch-Pagan test for heteroskedasticity
16 hettest
17
18 *Re-estimating the regression using robust standard errors
19 reg salary seasons pointspg, vce(robust)
20
21 *Generating the square of seasons
22 gen seasonssq = seasons^2
23
24 *Adding seasonssq to the regression
25 reg salary seasons pointspg seasonssq, vce(robust)
26
27 *Generating position dummies
28 tab position, gen(pos)
29
30 *Adding position dummies to the regression
31 reg salary seasons pointspg seasonssq pos2-pos5, vce(robust)
32
33 *Generating the natural log of salary
34 gen lnsal = ln(salary)
35
36 *Estimates the previous regression using lnsal as the dependent variable
37 reg lnsal seasons pointspg seasonssq pos2-pos5, vce(robust)
  
```

Line: 1, Col: 1 CAP NUM OVR

The results of the executed do-file are stored in the log file as shown below,


```
. *Summary statistics
. sum salary seasons pointspg
```

Variable	Obs	Mean	Std. dev.	Min	Max
salary	311	9.848679	10.64244	.954267	44.0664
seasons	311	5.961415	4.438663	1	18
pointspg	311	10.20276	5.39261	.8	26.9333

```
.
. *Running OLS regression
. reg salary seasons pointspg
```

Source	SS	df	MS	Number of obs	=	311
Model	17776.1577	2	8888.07884	F(2, 308)	=	157.92
Residual	17334.9436	308	56.2822844	Prob > F	=	0.0000
				R-squared	=	0.5063
				Adj R-squared	=	0.5031
Total	35111.1013	310	113.261617	Root MSE	=	7.5022

```
.
. *Graph of residuals vs. fitted values
. rvfplot, yline(0)
```

```
.
. *Breusch-Pagan test for heteroskedasticity
. hettest
```

Breusch-Pagan/Cook-Weisberg test for heteroskedasticity
Assumption: Normal error terms
Variable: Fitted values of salary

H0: Constant variance

chi2(1) = 145.28
Prob > chi2 = 0.0000

```
.
. *Re-estimating the regression using robust standard errors
. reg salary seasons pointspg, vce(robust)
```

Linear regression

Variable	Coefficient	Robust std. err.	t	P> t	[95% conf. interval]
salary	.2998411	.1025005	2.93	0.004	.0981514 .5015309
seasons	1.298562	.0843683	15.39	0.000	1.132551 1.464573
pointspg	-5.18771	.9625665	-5.39	0.000	-7.081748 -3.293671
_cons					

```
.
. *Generating the square of seasons
. gen seasonssq = seasons^2
```

```
.
. *Adding seasonssq to the regression
. reg salary seasons pointspg seasonssq, vce(robust)
```

Linear regression

Variable	Coefficient	Robust std. err.	t	P> t	[95% conf. interval]
salary	.2998411	.1177799	2.55	0.011	.0680861 .5315962
seasons	1.298562	.119127	10.90	0.000	1.064156 1.532968
pointspg	-5.18771	.9005532	-5.76	0.000	-6.959725 -3.415695
seasonssq					
_cons					

```
.
. *Adding seasonssq to the regression
. reg salary seasons pointspg seasonssq, vce(robust)
```

Linear regression

Variable	Coefficient	Robust std. err.	t	P> t	[95% conf. interval]
salary	1.941662	.3393946	5.72	0.000	1.273828 2.609496
seasons	1.252082	.1166127	10.74	0.000	1.02262 1.481543
pointspg	-1.104537	.0215986	-4.84	0.000	-.1470437 -.0620437
seasonssq	-8.732661	1.075569	-8.12	0.000	-10.84908 -6.616242
_cons					

```
. *Generating position dummies
. tab position, gen(pos)
```

position	Freq.	Percent	Cum.
C	31	9.97	9.97
PF	54	17.36	27.33
PG	68	21.86	49.20
SF	70	22.51	71.70
SG	88	28.30	100.00
Total	311	100.00	

```
.
. *Adding position dummies to the regression
. reg salary seasons pointspg seasonssq pos2-pos5, vce(robust)
```

Linear regression	Number of obs	=	311
	F(7, 303)	=	31.56
	Prob > F	=	0.0000
	R-squared	=	0.5477
	Root MSE	=	7.2393

salary	Coefficient	Robust std. err.	t	P> t	[95% conf. interval]	
seasons	1.954983	.3369963	5.80	0.000	1.291834	2.618133
pointspg	1.269594	.1187075	10.70	0.000	1.035999	1.50319
seasonssq	-.1053774	.0215852	-4.88	0.000	-.1478534	-.0629014
pos2	1.13598	1.687333	0.67	0.501	-2.184393	4.456354
pos3	-.0084028	1.615381	-0.01	0.996	-3.187189	3.170383
pos4	1.462988	1.538241	0.95	0.342	-1.564	4.489976
pos5	.3142074	1.489272	0.21	0.833	-2.616418	3.244833
_cons	-9.55836	1.652818	-5.78	0.000	-12.81081	-6.305905

```
.
. *Generating the natural log of salary
. gen lnslal = ln(salary)

.
. *Estimates the previous regression using lnslal as the dependent variable
. reg lnslal seasons pointspg seasonssq pos2-pos5, vce(robust)
```

Linear regression	Number of obs	=	311
	F(7, 303)	=	57.01
	Prob > F	=	0.0000
	R-squared	=	0.5481
	Root MSE	=	.70797

lnslal	Coefficient	Robust std. err.	t	P> t	[95% conf. interval]	
seasons	.2427121	.0308956	7.86	0.000	.181915	.3035092
pointspg	.1140511	.0085434	13.35	0.000	.0972393	.130863
seasonssq	-.0131047	.0018181	-7.21	0.000	-.0166824	-.009527
pos2	.1416589	.1754	0.81	0.420	-.2034976	.4868153
pos3	.1451976	.1652415	0.88	0.380	-.1799686	.4703639
pos4	.1238467	.1603578	0.77	0.441	-.1917093	.4394027
pos5	.1137404	.1595822	0.71	0.477	-.2002893	.42777
_cons	-.2510371	.1650031	-1.52	0.129	-.5757342	.0736599

```
.
.
.
.
end of do-file
.
```

Readers are invited to copy the following commands into a do-file and execute them using the NBA basketball data set:

```
*NBA salary project

*Opening a log file
log using project1.log, replace

*Summary statistics
sum salary seasons pointspg

*Running OLS regression
reg salary seasons pointspg

*Graph of residuals vs. fitted values
rvfplot, yline(0)

*Breushc-Pagan test for heteroskedasticity
hettest

*Re-estimating the regression using robust standard errors
reg salary seasons pointspg, vce(robust)

*Generating the square of seasons
gen seasonssq = seasons^2

*Adding seasonssq to the regression
reg salary seasons pointspg seasonssq, vce(robust)

*Generating position dummies
tab position, gen(pos)

*Adding position dummies to the regression
reg salary seasons pointspg seasonssq pos2-pos5, vce(robust)

*Generating the natural log of salary
gen lnsal = ln(salary)

*Estimates the previous regression using lnsal as the dependent variable
reg lnsal seasons pointspg seasonssq pos2-pos5, vce(robust)
```

13. Producing Publication-Quality Tables in Stata

By now the reader should be aware of how powerful Stata is when it comes to carrying out empirical research. Throughout *Regression Basics* there are examples of statistical output produced by this and other programs, and the output shown are simply ‘screen shots’ of what is displayed by these programs on the computer screen. While easy to copy and paste, this kind of output is generally not appropriate for inclusion in empirical research manuscripts. Well-formatted tables with easy-to-read variable names and values should be employed. Fortunately, Stata can create such tables. There are multiple programs designed for this purpose, but I will focus on one that, in my experience, is easy to use, is highly customizable, and works very well. I will show how to use it to produce a well-formatted table of summary statistics (also known as descriptive statistics), and a well-formatted table of regression results.

The outreg2 Program

The user-created outreg2 program will be used to create publication-quality tables with Stata. Before getting into the details, we need to install this program by typing the following into the Stata command line,

```
ssc install outreg2
```

Hitting ‘Enter’ will install the program (typically this needs to be done just once).

Summary Statistics

The basic syntax for using `outreg2` is,

```
outreg2 using file_name, options
```

Where *file_name* is the name of the output file to be created. Note that the output file will be saved in the current working directory (unless a different path is provided). Regarding options, there are many, but we will just focus on a few.

Given we are looking to create a table of summary statistics, we will add the `sum(log)` option which will compute summary statistics for all the variables in our data set.

Regarding output file types, `outreg2` supports several, including MS Word®, MS Excel®, and LaTeX. To produce a Word output file, we would add the option, `word`, for Excel we would add, `excel`, and for LaTeX we would add, `tex`.

As an example, consider our NBA salary data set, `nba2021_22.csv`, that we have been using in *Regression Basics*. We can import this data set into Stata and then we can type,

```
outreg2 using nba_sum, sum(log) excel replace
```

We see in this code that we will create an output file titled `nba_sum`. The `sum(log)` option tells Stata to create a table of summary statistics, and the `excel` option is used to save the file as an Excel file (the file will actually be saved as an `.xml` file, which opens with Excel). The last option is `replace`, which tells Stata that if there is already a file with the name `nba_sum` in the working

directory, then it should be replaced (i.e., written over) with the new one. Loading the NBA data into Stata and running the above code, we get the following in Stata's output window,

```
. outreg2 using nba_sum, sum(log) excel replace
```

Variable	Obs	Mean	Std. dev.	Min	Max
age	311	26.72669	4.314181	20	41
salary	311	9.848679	10.64244	.954267	44.0664
weight	311	214.5145	23.76618	164	290
height	311	77.98392	2.987565	72	87
games	311	329.8875	274.5789	3	1310
minspg	311	22.6147	7.560721	3.5	37.9944
orebsp	311	.8477645	.5996349	0	3.39
drebsp	311	2.954631	1.362276	.4	8.6
trebsp	311	3.801459	1.825524	.5	11.0154
astspg	311	2.263241	1.783535	0	9.4
stlspg	311	.7538795	.3682665	0	2.13125
blockspg	311	.4204017	.3525703	0	2.1625
tovspg	311	1.303802	.7756512	0	4.23333
foulspg	311	1.851306	.6032863	.1	3.2625
pointspg	311	10.20276	5.39261	.8	26.9333
seasons	311	5.961415	4.438663	1	18
lnsal	311	1.752835	1.041172	-.0468117	3.785697

```

Following variable is string, not included:
player position team
nba_sum.xml
dir : seeout

```

As we can see, at the bottom of the output in blue are two links. The first one, [nba_sum.xml](#), will open the Excel table, then second, [dir : seeout](#), will open a text version of the table (which is created by default). Clicking on [nba_sum.xml](#) we see, (assuming Excel is installed on the computer being used),⁶

⁶ Note that copies of these two files will also be stored in the working directory location.

	(1)	(2)	(3)	(4)	(5)
VARIABLES	N	mean	sd	min	max
age	311	26.73	4.314	20	41
salary	311	9.849	10.64	0.954	44.07
weight	311	214.5	23.77	164	290
height	311	77.98	2.988	72	87
games	311	329.9	274.6	3	1,310
minspg	311	22.61	7.561	3.500	37.99
orebsp	311	0.848	0.600	0	3.390
drebsp	311	2.955	1.362	0.400	8.600
trebsp	311	3.801	1.826	0.500	11.02
astspg	311	2.263	1.784	0	9.400
stlspg	311	0.754	0.368	0	2.131
blockspg	311	0.420	0.353	0	2.162
tovspg	311	1.304	0.776	0	4.233
foulspg	311	1.851	0.603	0.100	3.263
pointspg	311	10.20	5.393	0.800	26.93
seasons	311	5.961	4.439	1	18
lnsal	311	1.753	1.041	-0.0468	3.786

This is a fully editable Excel file; thus, we can change the font type, delete columns/rows, rename variables, etc. Making a few changes, we can then copy and paste the table into a Word document yielding the following,

VARIABLES	N	mean	sd	min	max
age	311	26.73	4.314	20	41
salary (millions of \$s)	311	9.849	10.64	0.954	44.07
weight	311	214.5	23.77	164	290
height	311	77.98	2.988	72	87
games	311	329.9	274.6	3	1,310
minspg	311	22.61	7.561	3.500	37.99
orebsp	311	0.848	0.600	0	3.390
drebsp	311	2.955	1.362	0.400	8.600
trebsp	311	3.801	1.826	0.500	11.02
astspg	311	2.263	1.784	0	9.400
stlspg	311	0.754	0.368	0	2.131
blockspg	311	0.420	0.353	0	2.162
tovspg	311	1.304	0.776	0	4.233
foulspg	311	1.851	0.603	0.100	3.263
pointspg	311	10.20	5.393	0.800	26.93
seasons	311	5.961	4.439	1	18

If we use the word option in place of excel, then Stata will output a file that is a .rtf file type that is openable in Word.

Regression Tables

Tables showing regression output can vary in terms of layout and content. Estimates for models with just a few independent variables are sometimes written in the form of a function. As an example, suppose we estimate a simple regression with our NBA data set where the natural log of salary is regressed on seasons and pointspg. We can present the results in the following equation (with rounding to 3 decimal places),

$$\log(\widehat{salary}_i) = 0.302 + 0.036seasons_i + 0.121pointspg_i$$

(0.096) (0.010) (0.008)

n = 311 R² = 0.485

We see in the above equation the estimated intercept and coefficients to the independent variables are reported, with their standard errors in parentheses below each.

When more complicated models are estimated with many independent variables, the equation format does not work well. A common format in this case is to have columns of output that contain estimated coefficients, standard errors (typically in parentheses) below each coefficient, and some symbol, (often an asterisk or ‘star’) indicating if an estimated coefficient is statistically different from zero. The `outreg2` program can produce such tables with just one line of code. Following a regression command, we can use the `outreg2` command as a post-estimation command. For example, we can regress the natural log of salary on seasons and pointspg, and then following the estimation we can create a regression table using `outreg2`. Assuming the NBA data set has already been read into Stata, the following code can be used one line at a time in Stata’s command line, or put all together into a do-file and run,

```
gen lnsal = ln(salary)
reg lnsal seasons pointspg
outreg2 using nba_reg, excel replace
```

The first line of code generates the natural log of salary, the second line estimates the regression, and the third line creates a regression table called `nba_reg`, which is openable in Excel, (notice

that we omit the `sum(log)` option, which is only used when creating summary statistics). We see in the Stata output window,

```
. gen lnsal = ln(salary)

. reg lnsal seasons pointspg
```

Source	SS	df	MS	Number of obs	=	311
Model	162.945468	2	81.4727342	F(2, 308)	=	144.96
Residual	173.106699	308	.562034736	Prob > F	=	0.0000
				R-squared	=	0.4849
				Adj R-squared	=	0.4815
Total	336.052167	310	1.08403925	Root MSE	=	.74969

lnsal	Coefficient	Std. err.	t	P> t	[95% conf. interval]
seasons	.0361621	.0102429	3.53	0.000	.0160073 .056317
pointspg	.1210878	.0084309	14.36	0.000	.1044984 .1376773
_cons	.3018281	.0961892	3.14	0.002	.1125569 .4910993

```
. outreg2 using nba_reg, excel replace
nba_reg.xml
dir : seeout
```

Clicking on the blue [nba_reg.xml](#) link opens the following table in Excel,

The screenshot shows an Excel spreadsheet with the following data:

	A	B	C
1			
2		(1)	
3	VARIABLES	lnsal	
4			
5	seasons	0.0362***	
6		(0.0102)	
7	pointspg	0.121***	
8		(0.00843)	
9	Constant	0.302***	
10		(0.0962)	
11			
12	Observations	311	
13	R-squared	0.485	
14	Standard errors in parentheses		
15	*** p<0.01, ** p<0.05, * p<0.1		
16			

As with the summary statistics table, the Excel file is fully editable. Note that `outreg2` by default uses ‘stars’ (i.e., asterisks) to indicate the level of significance of each estimated coefficient, and it puts a key to those stars at the bottom of the table (one asterisk means the estimated coefficient is statistically different from zero at the 10 percent level of significance. Two and three asterisks indicate significance at the 5 and 1 percent levels, respectively). It also adds a note that standard errors are reported in parentheses. Editing the table to match fonts, etc., and copying and pasting we get,

VARIABLES	log(salary)
seasons	0.0362*** (0.0102)
pointspg	0.121*** (0.00843)
Constant	0.302***
Observations	311
R-squared	0.485
Standard errors in parentheses	
*** p<0.01, ** p<0.05, * p<0.1	

Up to this point, we have been considering a very simple regression model with just two independent variables. Often, we will be considering more complex models, and furthermore we may start with a simple version of a model, and then ‘build out’ a more comprehensive model. In this case, we can create a regression table with multiple columns of output, one for each model estimated. This is easily done with `outreg2` by incorporating an `append` option. This option takes an existing output table created by `outreg2` and adds or ‘appends’ a new column of regression results to that table. Thus, we can consider adding to our `nba_reg` table the results of two other regression models: one that adds the variables, `height` and `weight`, and another that adds the squared value of `seasons`. Assuming the NBA data has been loaded into Stata, we can add the

following code into the command line one line at a time, or add all the code to a do-file and then run the do-file,

```
gen lnsal = ln(salary)
reg lnsal seasons pointspg
outreg2 using nba_reg, excel replace

reg lnsal seasons pointspg height weight
outreg2 using nba_reg, excel append

gen seasonssq = seasons^2
reg lnsal seasons pointspg height weight seasonssq
outreg2 using nba_reg, excel append
```

Running the above code creates a new output table readable by Excel that looks like the following,

VARIABLES	(1) lnsal	(2) lnsal	(3) lnsal
seasons	0.0362*** (0.0102)	0.0368*** (0.0104)	0.240*** (0.0331)
pointspg	0.121*** (0.00843)	0.122*** (0.00861)	0.116*** (0.00815)
height		0.0120 (0.0214)	0.00725 (0.0201)
weight		-0.00123 (0.00266)	-0.000967 (0.00250)
seasonssq			-0.0129*** (0.00201)
Constant	0.302*** (0.0962)	-0.384 (1.330)	-0.503 (1.250)
Observations	311	311	311
R-squared	0.485	0.485	0.547
Standard errors in parentheses			
*** p<0.01, ** p<0.05, * p<0.1			

We now see that outreg2 has combined the three models into a single table. As before, we can edit this table's font, etc., to make it fit the style of our document. In addition, it may be desirable

to replace some of the variable names, which can sometimes appear a bit cryptic to readers, with more descriptive versions,

VARIABLES	(1)	(2)	(3)
seasons in the NBA	0.0362*** (0.0102)	0.0368*** (0.0104)	0.240*** (0.0331)
career points per game	0.121*** (0.00843)	0.122*** (0.00861)	0.116*** (0.00815)
height (in inches)		0.0120 (0.0214)	0.00725 (0.0201)
weight (in pounds)		-0.00123 (0.00266)	-0.000967 (0.00250)
seasons squared			-0.0129*** (0.00201)
Constant	0.302*** (0.0962)	-0.384 (1.330)	-0.503 (1.250)
Observations	311	311	311
R-squared	0.485	0.485	0.547
Standard errors in parentheses			
*** p<0.01, ** p<0.05, * p<0.1			

The above table is now something we might find in a professional publication. As noted earlier, `outreg2` is *highly* customizable. To find out more, the user can type `help outreg2` at the command line and hit the ‘Enter’ key.