Working with dates and times in Stata

Gabriela Ortiz/Hua Peng
Applied Econometrician/Software Engineer
StataCorp LLC

Ideal dates

August 5, 2022

2022.08.05

08-05-2022

Aug 05 ‘22

What is your ideal date?

05aug2022

20220805

08/05/2022

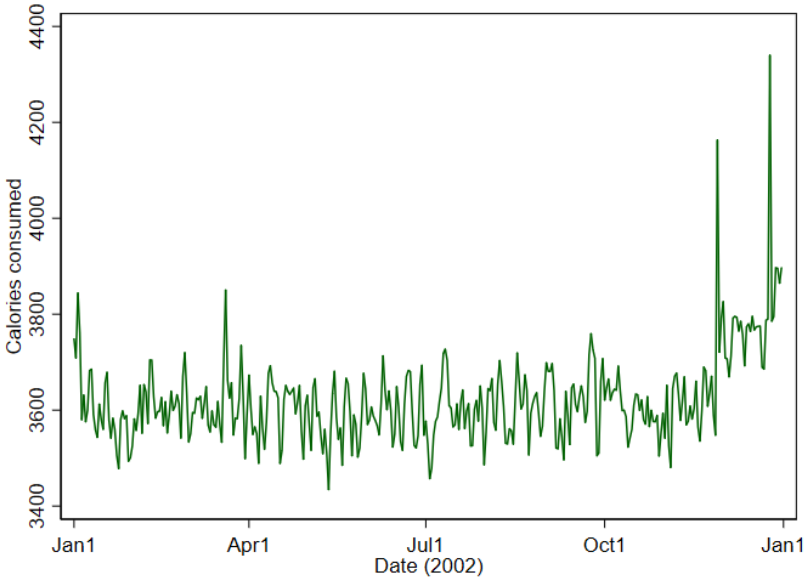
Fri Aug 05 2021

You may be working with dates for a couple different reasons

```
. sort birthday

. list patid birthday
```

	patid	birthday
1.	4	26aug1960
2.	3	15nov1975
3.	5	16dec1987
4.	2	01apr1999
5.	1	15may2001




```
. generate stay = discharge - admit

. list admit discharge stay
```

	admit	discharge	stay
1.	03/13/2011	03/26/2011	13
2.	06/25/2011	06/29/2011	4
3.	02/11/2012	02/16/2012	5
4.	08/01/2012	08/02/2012	1

Sort data chronologically

Perform time-series analysis

Compute the time between dates

Overview

- How Stata stores dates and times
- Converting dates and times stored as strings to numeric dates and times
- Formatting our dates and times for readability
- Converting among date types
- Using dates and times in expressions
- Computing durations
- Converting dates and times from other software to Stata dates and times

How and why Stata stores dates

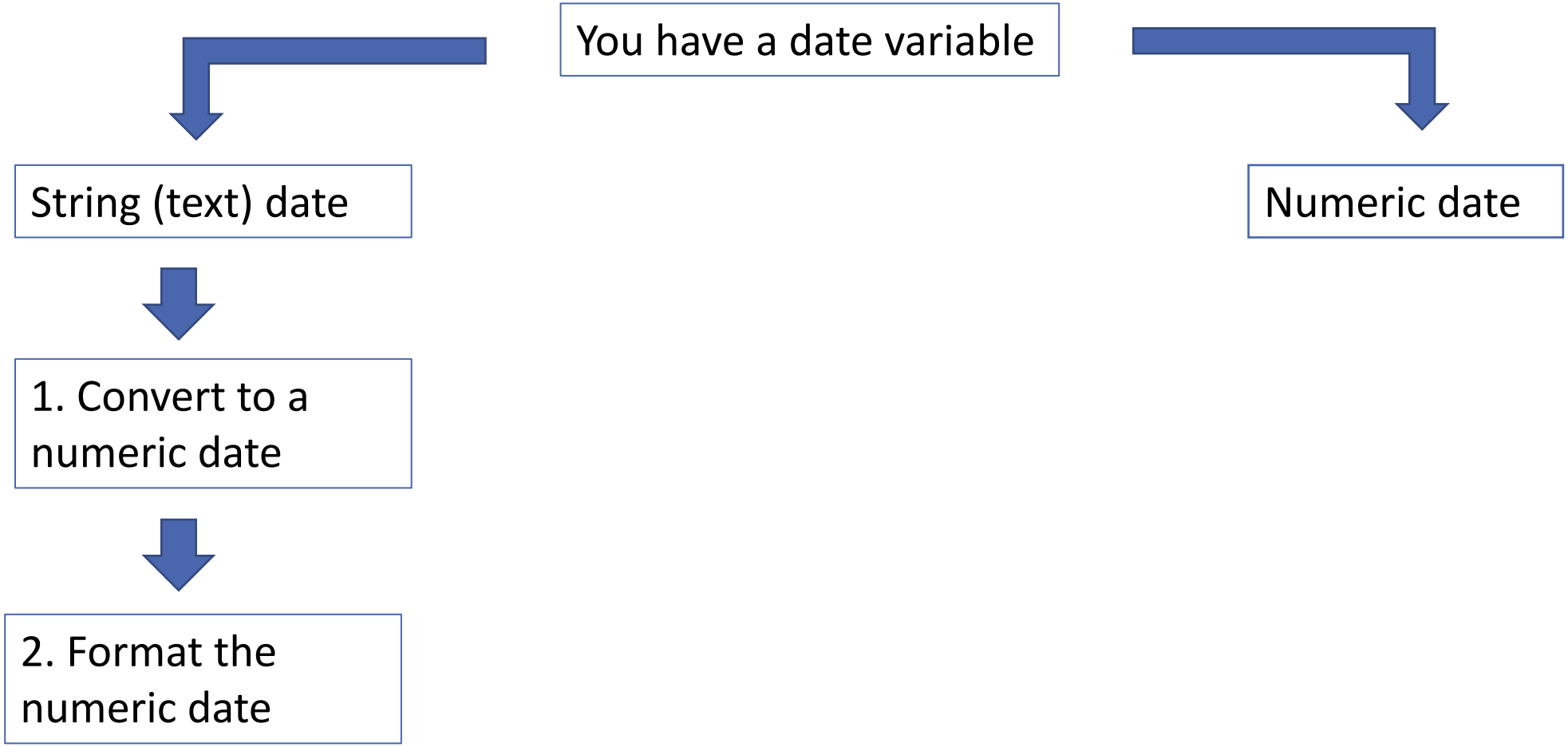
- Stata has dates and datetimes and they are stored differently. We'll begin by focusing on dates.

- Stata stores dates as the number of days elapsed since January 1, 1960
  - This means January 1, 1970, would be stored as 3653 $((365 \times 10) + 3)$

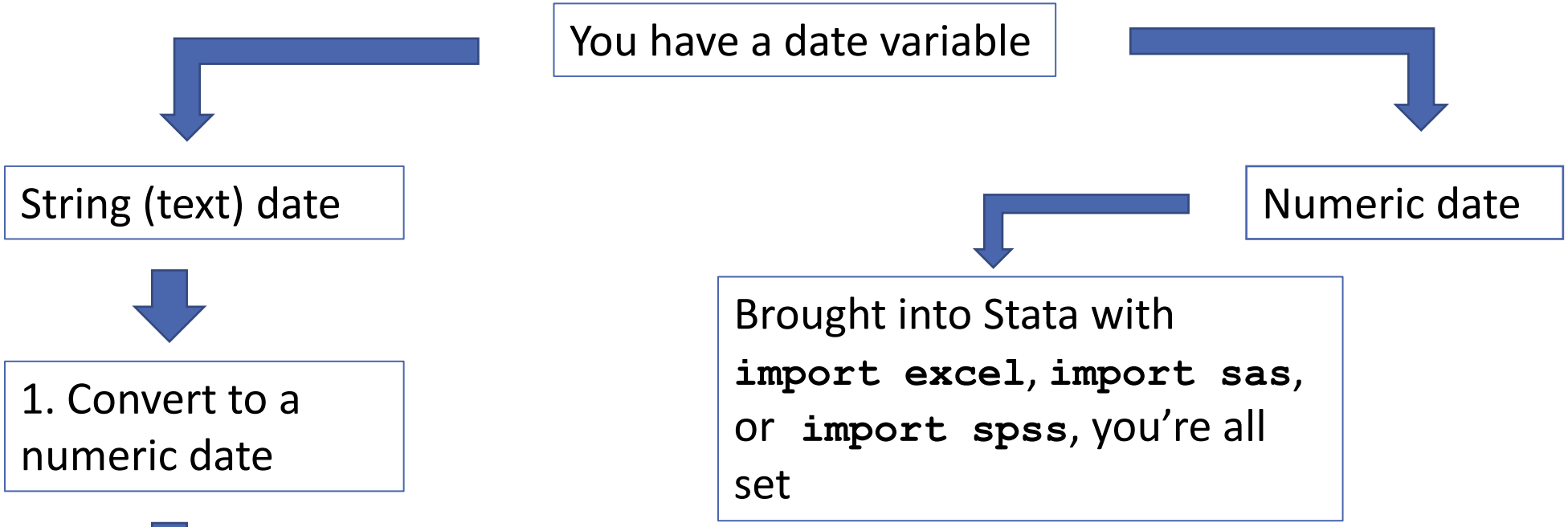
- We use display formats to display 3653 as January 1, 1970

- Numeric dates allow us to:
  - Sort our data chronologically
  - Prepare our data for time-series analysis
  - Compute the time between dates

How dates work in Stata



How dates work in Stata



How dates work in Stata

You have a date variable

String (text) date

1. Convert to a numeric date

2. Format the numeric date

Numeric date

Brought into Stata with **`import excel`**, **`import sas`**, or **`import spss`**, you're all set

Brought into Stata from a .csv or .txt file

1. Convert the numeric date to a numeric variable corresponding to Stata's base date

2. Format the numeric date

Fictional hospital admissions data

```
. describe

Contains data from visits.dta
 Observations:             5                  Fictional hospital visit data
    Variables:            13                  28 Oct 2020 14:41
```

Variable name	Storage type	Display format	Value label	Variable label
patid	byte	%9.0g		Patient ID
dateofbirth	str9	%9s		Date of birth
reason	str15	%15s		Reason for visit
admit_str	str8	%9s		Admission date
admittime_str	str20	%20s		Admission date and time
discharge_str	str9	%9s		Discharge date
dischargetime~r	str14	%14s		Discharge date and time
time_str	str11	%11s		Admission time
bmonth	byte	%9.0g		Birth month
bday	byte	%9.0g		Birth day
byear	int	%9.0g		Birth year
dmonth_str	str8	%9s		Month of discharge
dyear	int	%9.0g		Year of discharge

```
Sorted by:
```

Converting dates and times stored as strings to numeric dates and times

Dates of admission and discharge

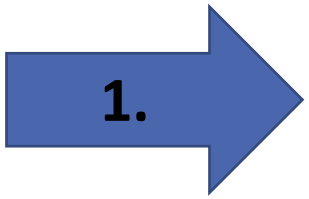
```
. list patid admit_str discharge_str, ab(13)
```

	patid	admit_str	discharge_str
1.	1	20110625	jun292011
2.	2	20110313	mar262011
3.	3	20110409	apr092011
4.	4	20120211	feb162012
5.	5	20120801	aug022012

Step 1: Convert string date to a numeric date

```
. list patid admit_str discharge_str, ab(13)
```

	patid	admit_str	discharge_str
1.	1	20110625	jun292011
2.	2	20110313	mar262011
3.	3	20110409	apr092011
4.	4	20120211	feb162012
5.	5	20120801	aug022012




```
. generate admit = date(admit_str, "YMD")

. list patid admit
```

	patid	admit
1.	1	18803
2.	2	18699
3.	3	18726
4.	4	19034
5.	5	19206

Step 2: Format the numeric date for readability

```
. list patid admit_str discharge_str, ab(13)
```

	patid	admit_str	discharge_str
1.	1	20110625	jun292011
2.	2	20110313	mar262011
3.	3	20110409	apr092011
4.	4	20120211	feb162012
5.	5	20120801	aug022012

1.

```
. generate admit = date(admit_str, "YMD")
```

2.

```
. format admit %td

. list patid admit
```

	patid	admit
1.	1	25jun2011
2.	2	13mar2011
3.	3	09apr2011
4.	4	11feb2012
5.	5	01aug2012

Step 1: Convert string date to a numeric date

```
. generate discharge = date(discharge_str, "MDY")

. list discharge_str discharge, ab(13)
```

	discharge_str	discharge
1.	jun292011	18807
2.	mar262011	18712
3.	apr092011	18726
4.	feb162012	19039
5.	aug022012	19207

Step 2: Format the numeric date for readability

```
. generate discharge = date(discharge_str, "MDY")

. list discharge_str discharge, ab(13)
```

	discharge_str	discharge
1.	jun292011	18807
2.	mar262011	18712
3.	apr092011	18726
4.	feb162012	19039
5.	aug022012	19207

```
. format discharge %td

. list patid discharge
```

	patid	discharge
1.	1	29jun2011
2.	2	26mar2011
3.	3	09apr2011
4.	4	16feb2012
5.	5	02aug2012

The %td display format

```
. list admit_str discharge_str admit discharge, ab(13)
```

	admit_str	discharge_str	admit	discharge
1.	20110625	jun292011	25jun2011	29jun2011
2.	20110313	mar262011	13mar2011	26mar2011
3.	20110409	apr092011	09apr2011	09apr2011
4.	20120211	feb162012	11feb2012	16feb2012
5.	20120801	aug022012	01aug2012	02aug2012

Customized date formats

Format	Date displayed
`%tdnn-dd-yy`	8-1-12
`%tdnn/dd/yy`	8/1/12

Customized date formats

Format	Date displayed
`%tdnn-dd-yy`	8-1-12
`%tdnn/dd/yy`	8/1/12
`%tdNN/DD/YY`	08/01/12
`%tdNN/DD/CCYY`	08/01/2012

Customized date formats

Format	Date displayed
`%tdnn-dd-yy`	8-1-12
`%tdnn/dd/yy`	8/1/12
`%tdNN/DD/YY`	08/01/12
`%tdNN/DD/CCYY`	08/01/2012
`%tdMon_dd_!'yy`	Aug 1 '12
`%tdMonth_dd,_ccyy`	August 1, 2012
`%tdDayname_Mon_dd`	Wednesday Aug 1

See [D] Datetime display formats for more options.

Formatting dates: Spelling out the month

```
. format admit %tdMonth_dd,_ccyy

. list admit in 4/5
```

	admit
4.	February 11, 2012
5.	August 1, 2012

Date and time variables

- Stata stores datetimes as the number of milliseconds elapsed since January 1, 1960 00:00:00.000
  - This is assuming there are 86,400 seconds in a day (60 seconds × 60 minutes × 24 hours)

- Once or twice a year, leap seconds are added to atomic clocks so that they're better synchronized with the Earth's rotation
  - We'll see how to adjust for leap seconds

Date and time variables

```
. list admittime_str dischargetime_str, ab(17)
```

	admittime_str	dischargetime_str
1.	20110625 5:15:06 am	20110629 10:27
2.	20110313 8:30:45 am	20110326 14:15
3.	20110409 10:17:08 am	20110409 19:35
4.	20120211 10:30:12 pm	20120216 14:22
5.	20120801 6:45:59 pm	20120802 21:59

Converting dates and times stored as strings to numeric datetime variables

```
. list admittime_str dischargetime_str, ab(17)
```

	admittime_str	dischargetime_str
1.	20110625 5:15:06 am	20110629 10:27
2.	20110313 8:30:45 am	20110326 14:15
3.	20110409 10:17:08 am	20110409 19:35
4.	20120211 10:30:12 pm	20120216 14:22
5.	20120801 6:45:59 pm	20120802 21:59

```
. generate double admit_time = clock(admittime_str, "YMDhms")

. generate double discharge_time = clock(dischargetime_str, "YMDhm")
```

Always use storage type **`double`** when working with datetime variables.

Converting dates and times stored as strings to numeric datetime variables

```
. list admittime_str dischargetime_str, ab(17)
```

	admittime_str	dischargetime_str
1.	20110625 5:15:06 am	20110629 10:27
2.	20110313 8:30:45 am	20110326 14:15
3.	20110409 10:17:08 am	20110409 19:35
4.	20120211 10:30:12 pm	20120216 14:22
5.	20120801 6:45:59 pm	20120802 21:59

```
. generate double admit_time = clock(admittime_str, "YMDhms")

. generate double discharge_time = clock(dischargetime_str, "YMDhm")

. list admittime_str dischargetime_str admit_time discharge_time, ab(17)
```

	admittime_str	dischargetime_str	admit_time	discharge_time
1.	20110625 5:15:06 am	20110629 10:27	1.625e+12	1.625e+12
2.	20110313 8:30:45 am	20110326 14:15	1.616e+12	1.617e+12
3.	20110409 10:17:08 am	20110409 19:35	1.618e+12	1.618e+12
4.	20120211 10:30:12 pm	20120216 14:22	1.645e+12	1.645e+12
5.	20120801 6:45:59 pm	20120802 21:59	1.659e+12	1.660e+12

Formatting numeric datetime variables

```
. list admittime_str dischargetime_str, ab(17)
```

	admittime_str	dischargetime_str
1.	20110625 5:15:06 am	20110629 10:27
2.	20110313 8:30:45 am	20110326 14:15
3.	20110409 10:17:08 am	20110409 19:35
4.	20120211 10:30:12 pm	20120216 14:22
5.	20120801 6:45:59 pm	20120802 21:59

```
. generate double admit_time = clock(admittime_str, "YMDhms")

. generate double discharge_time = clock(dischargetime_str, "YMDhm")

. list admittime_str dischargetime_str admit_time discharge_time, ab(17)

. format %tc admit_time discharge_time

. list admit_time discharge_time
```

	admit_time	discharge_time
1.	25jun2011 05:15:06	29jun2011 10:27:00
2.	13mar2011 08:30:45	26mar2011 14:15:00
3.	09apr2011 10:17:08	09apr2011 19:35:00
4.	11feb2012 22:30:12	16feb2012 14:22:00
5.	01aug2012 18:45:59	02aug2012 21:59:00

Converting a strictly time variable to a numeric datetime variable

```
. list time_str
```

	time_str
1.	5:15:06 am
2.	8:30:45 am
3.	10:17:08 am
4.	10:30:12 pm
5.	6:45:59 pm

Converting a strictly time variable to a numeric datetime variable

```
. list time_str
```

	time_str
1.	5:15:06 am
2.	8:30:45 am
3.	10:17:08 am
4.	10:30:12 pm
5.	6:45:59 pm

```
. generate double time = clock(time_str, "hms")

. format time %tc

. list time_str time
```

	time_str	time
1.	5:15:06 am	01jan1960 05:15:06
2.	8:30:45 am	01jan1960 08:30:45
3.	10:17:08 am	01jan1960 10:17:08
4.	10:30:12 pm	01jan1960 22:30:12
5.	6:45:59 pm	01jan1960 18:45:59

Customized time formats

Format	Time displayed
`%tcHHMM`	1845
`%tcHH:MM`	18:45

See [D] Datetime display formats for more options.

Customized time formats

Format	Time displayed
%tcHHMM	1845
%tcHH:MM	18:45
%tchh:MM	6:45
%tcHh:MM	06:45

See [D] Datetime display formats for more options.

Customized time formats

Format	Time displayed
`%tcHHMM`	1845
`%tcHH:MM`	18:45
`%tchh:MM`	6:45
`%tcHh:MM`	06:45
`%tcHh:MM_a.m.`	06:45 p.m.
`%tcHh:MM_A.M.`	06:45 P.M.

See [D] Datetime display formats for more options.

Customizing the display format for the datetime variables

```
. format admit_time %tcnn/dd/yy_HH:MM

. format discharge_time %tcHH:MM

. list admit_time discharge_time, ab(14)
```

	admit_time	discharge_time
1.	6/25/11 05:15	10:27
2.	3/13/11 08:30	14:15
3.	4/9/11 10:17	19:35
4.	2/11/12 22:30	14:22
5.	8/1/12 18:45	21:59

```
. sort admit_time
```

Obtaining leap-second adjusted times

```
. generate double admit_Time = Clock(admittime_str, "YMDhms")

. format admit_Time %tC

. list admittime_str admit_Time, ab(13)
```

	admittime_str	admit_Time
1.	20110313 8:30:45 am	13mar2011 08:30:45
2.	20110409 10:17:08 am	09apr2011 10:17:08
3.	20110625 5:15:06 am	25jun2011 05:15:06
4.	20120211 10:30:12 pm	11feb2012 22:30:12
5.	20120801 6:45:59 pm	01aug2012 18:45:59

String-to-numeric conversion functions

Date type	String-to-numeric conversion function	Example
Datetime	**clock(***string, "order_of_components"***)**	**clock(timevar, "YMDhms")**
*Datetime (UTC)	**Clock(***string, "order_of_components"***)**	**Clock(timevar, "YMDhms")**
Daily date	**date(***string, "order_of_components"***)**	**date(datevar, "YMD")**
Weekly date	**weekly(***string, "order_of_components"***)**	**weekly(weekvar, "YW")**
Monthly date	**monthly(***string, "order_of_components"***)**	**monthly(monthvar, "YM")**
Quarterly date	**quarterly(***string, "order_of_components"***)**	**quarterly(qvar, "YQ")**

* Adjusted for leap seconds.

Date types and their units

Date type	Unit
Datetime (assuming 86,400 s/day)	Milliseconds since 01jan1960 00:00:00.000
Datetime (UTC)	Milliseconds since 01jan1960 00:00:00.000, adjusted for leap seconds
Daily date	Days since 01jan1960 (01jan1960 = 0)
Weekly date	Weeks since 1960w1
Monthly date	Months since 1960m1
Quarterly date	Quarters since 1960q1

Date and time display formats

Date type	Display format	Date and time displayed
Datetime	`%tc`	04feb2020 05:15:00
Datetime adjusted for leap seconds	`%tC`	04feb2020 05:15:00
Daily date	`%td`	04feb2020
Weekly date	`%tw`	2020w6
Monthly date	`%tm`	2020m2
Quarterly date	`%tq`	2020q1

Date and time display formats

Date type	Display format	Date and time displayed	Customized display format
Datetime	**%tc**	04feb2020 05:15:00	**%tc**[*details*]
Datetime adjusted for leap seconds	**%tC**	04feb2020 05:15:00	**%tC**[*details*]
Daily date	**%td**	04feb2020	**%td**[*details*]
Weekly date	**%tw**	2020w6	**%tw**[*details*]
Monthly date	**%tm**	2020m2	**%tm**[*details*]
Quarterly date	**%tq**	2020q1	**%tq**[*details*]

Converting among date types

Converting among date types

- Sometimes the dates we are given are not of the form we need
- We can easily convert a datetime variable to a daily date, a daily date to a monthly date, etc.
  - In these cases, we have more information than we need
- We can also convert, for example, a monthly date to a daily date
  - In this case, we don’t have all the information we need, so Stata uses defaults
- Suppose we only had the date and time variable admit_time, but we are not interested in the time aspect

Converting a datetime variable to a daily date

1)

```
. generate datefromtime = dofc(admit_time)

. list admit_time datefromtime, ab(12)
```

	admit_time	datefromtime
1.	3/13/11 08:30	18699
2.	4/9/11 10:17	18726
3.	6/25/11 05:15	18803
4.	2/11/12 22:30	19034
5.	8/1/12 18:45	19206

Converting a datetime variable to a daily date

1) `. generate datefromtime = dofc(admit_time)`

`. list admit_time datefromtime, ab(12)`

	admit_time	datefromtime
1.	3/13/11 08:30	18699
2.	4/9/11 10:17	18726
3.	6/25/11 05:15	18803
4.	2/11/12 22:30	19034
5.	8/1/12 18:45	19206

2) `. format datefromtime %td`

`. list admit_time datefromtime, ab(12)`

	admit_time	datefromtime
1.	3/13/11 08:30	13mar2011
2.	4/9/11 10:17	09apr2011
3.	6/25/11 05:15	25jun2011
4.	2/11/12 22:30	11feb2012
5.	8/1/12 18:45	01aug2012

Converting a daily date to a monthly date

1)

```
. generate mfromdate = mofd(datefromtime)

. list datefromtime mfromdate, ab(12)
```

	datefromtime	mfromdate
1.	13mar2011	614
2.	09apr2011	615
3.	25jun2011	617
4.	11feb2012	625
5.	01aug2012	631

Converting a daily date to a monthly date

1)

```
. generate mfromdate = mofd(datefromtime)

. list datefromtime mfromdate, ab(12)
```

	datefromtime	mfromdate
1.	13mar2011	614
2.	09apr2011	615
3.	25jun2011	617
4.	11feb2012	625
5.	01aug2012	631

2)

```
. format mfromdate %tm

. list datefromtime mfromdate, ab(12)
```

	datefromtime	mfromdate
1.	13mar2011	2011m3
2.	09apr2011	2011m4
3.	25jun2011	2011m6
4.	11feb2012	2012m2
5.	01aug2012	2012m8

Nesting datetime functions

```
. generate monthly = mofd(dofc(admit_time))

. format monthly %tm

. list admit_time monthly
```

	admit_time	monthly
1.	3/13/11 08:30	2011m3
2.	4/9/11 10:17	2011m4
3.	6/25/11 05:15	2011m6
4.	2/11/12 22:30	2012m2
5.	8/1/12 18:45	2012m8

Converting an existing datetime variable to UTC

```
. generate double basictoutc = Cofc(admit_time)

. format admit_time basictoutc admit_Time %16.0f

. list admit_time basictoutc admit_Time
```

	admit_time	basictoutc	admit_Time
1.	1615624245000	1615624269000	1615624269000
2.	1617963428000	1617963452000	1617963452000
3.	1624598106000	1624598130000	1624598130000
4.	1644618612000	1644618636000	1644618636000
5.	1659465959000	1659465984000	1659465984000

This time variable should be formatted as `%tC`, but let's look at the underlying values

Converting from UTC to non-leap-second adjusted datetimes

```
. generate double utctobasic = cofC(admit_Time)

. format utctobasic %16.0f

. list admit_Time admit_time utctobasic
```

	admit_Time	admit_time	utctobasic
1.	1615624269000	1615624245000	1615624245000
2.	1617963452000	1617963428000	1617963428000
3.	1624598130000	1624598106000	1624598106000
4.	1644618636000	1644618612000	1644618612000
5.	1659465984000	1659465959000	1659465959000

Conversions with insufficient information

```
. generate dailyofmonthly = dofm(monthly)

. format dailyofmonthly %td

. list monthly dailyofmonthly, ab(14)
```

	monthly	dailyofmonthly
1.	2011m3	01mar2011
2.	2011m4	01apr2011
3.	2011m6	01jun2011
4.	2012m2	01feb2012
5.	2012m8	01aug2012

Default values for date components

Stata stores datetimes as the number of milliseconds elapsed since January 1, 1960 00:00:00.000.

This falls on the first quarter, and the first week, of 1960.

When converting to a date type for which you don't have all the components (e.g., quarterly date to monthly date), the missing elements will be set to their default.

Date component	Default
Year	1960
Month	1
Day	1
Quarter	1
Week	1
Hour	00
Minute	00
Second	00

Converting across dates and times

From	To: Datetime	To: Datetime (UTC)	To: Daily date
Datetime		`Cofc()`	`dofc()`
Datetime (UTC)	`cofC()`		`dofC()`
Daily date	`cofd()`	`Cofd()`	

Converting across different types of dates

To

From	Daily date	Weekly date	Monthly date	Quarterly date
Daily date		`wofd()`	`mofd()`	`qofd()`
Weekly date	`dofw()`		`mofd(dofw())`	`qofd(dofw())`
Monthly date	`dofm()`	`wofd(dofm())`		`qofd(dofm())`
Quarterly date	`dofq()`	`wofd(dofq())`	`mofd(dofq())`	

For more conversion functions, see [D] Datetime.

Using dates and times in expressions

Using dates in expressions

- Dates are stored numerically, but formatted to display dates as we know them
- Rather than trying to think of the numeric value for a given date, we can use functions to tell Stata the date we are referring to
- This is useful when examining portions of your data based on dates, and when converting dates from other software, which we'll see shortly

Using dates in expressions

```
. list patid admit if admit > td(01-05-2011)
```

	patid	admit
3.	1	June 25, 2011
4.	4	February 11, 2012
5.	5	August 1, 2012

```
. list patid monthly if monthly > tm(2012-06)
```

	patid	monthly
5.	5	2012m8

Pseudofunctions for using dates in expressions

Date type	Pseudofunction
Datetime	**tc(**[*day-month-year*] *hh:mm*[:*ss*[.*sss*]]**)**
Datetime (UTC)	**tC(**[*day-month-year*] *hh:mm*[:*ss*[.*sss*]]**)**
Daily date	**td(***day-month-year***)**
Weekly date	**tw(***year-week***)**
Monthly date	**tm(***year-month***)**
Quarterly date	**tq(***year-quarter***)**

Using string-to-numeric conversion functions in expressions

```
. list patid admit if admit > date("May 1, 2011", "MDY")
```

	patid	admit
3.	1	June 25, 2011
4.	4	February 11, 2012
5.	5	August 1, 2012

```
. list patid monthly if monthly > monthly("June 2012", "MY")
```

	patid	monthly
5.	5	2012m8

The string-to-numeric conversion functions can also be used in expressions, and they allow you to specify the components in any order you wish.

Computing durations

Computing patients' ages

```
. use visits2, clear
(Fictional hospital visit data)

. describe

Contains data from visits2.dta
 Observations:             5                  Fictional hospital visit data
    Variables:             5                  10 Nov 2020 15:26
-------------------------------------------------------------------------------
Variable      Storage   Display    Value
    name         type    format    label      Variable label
-------------------------------------------------------------------------------
patid           byte    %9.0g                 Patient ID
birthday        int     %td                   Date of birth
reason          str15   %15s                  Reason for visit
admit           int     %td                   Date of admission
discharge       float   %td                   Date of discharge
-------------------------------------------------------------------------------
Sorted by:
```

Computing age on the day of admission

```
. generate age = age(birthday,admit)

. list birthday admit age
```

	birthday	admit	age
1.	15may2001	15may2011	10
2.	29feb2000	28feb2011	10
3.	15nov1975	14nov2011	35
4.	26aug1960	25aug2012	51
5.	16dec1987	16dec2012	25

Specifying when nonleap-year birthdays are observed

```
. display isleapyear(2011)
0

. generate age2 = age(birthday,admit,"28feb")

. list birthday admit age age2
```

	birthday	admit	age	age2
1.	15may2001	15may2011	10	10
2.	29feb2000	28feb2011	10	11
3.	15nov1975	14nov2011	35	35
4.	26aug1960	25aug2012	51	51
5.	16dec1987	16dec2012	25	25

Compute the difference between two dates

```
. generate daysofstay = datediff(admit, discharge, "day")

. list admit discharge daysofstay, ab(10)
```

	admit	discharge	daysofstay
1.	15may2011	19may2011	4
2.	28feb2011	01mar2011	1
3.	14nov2011	16nov2011	2
4.	25aug2012	29aug2012	4
5.	16dec2012	20dec2012	4

Functions for calculating durations

Description	Function
Age	**age(**$e_{d\ DOB}$**,** e_d **[,** s_{nl}**])**
Age with fraction	**age_frac(**$e_{d\ DOB}$**,** e_d **[,** s_{nl}**])**
Date difference	**datediff(**e_{d1}**,** e_{d2}**,** s_{du} **[,** s_{nl}**])**
Date difference with fraction	**datediff_frac(**e_{d1}**,** e_{d2}**,** s_{du} **[,** s_{nl}**])**

* $e_{d\ DOB}$, e_d, e_{d1}, and e_{d2} are Stata dates.

* s_{du} is a string specifying date units ("d", "m", or "y").

* s_{nl} is a string specifying nonleap-year birthdays ("01mar" or "28feb").

Functions for calculating durations

Description	Function
Age	**age(**$e_{d\ DOB}$**,** e_d **[,** s_{nl}**])**
Age with fraction	**age_frac(**$e_{d\ DOB}$**,** e_d **[,** s_{nl}**])**
Date difference	**datediff(**e_{d1}**,** e_{d2}**,** s_{du} **[,** s_{nl}**])**
Date difference with fraction	**datediff_frac(**e_{d1}**,** e_{d2}**,** s_{du} **[,** s_{nl}**])**
Datetime/C difference	**Clockdiff(**e_{tC1}**,** e_{tC2}**,** s_{tu}**)**
Datetime/c difference	**clockdiff(**e_{tc1}**,** e_{tc2}**,** s_{tu}**)**

* e_{tc1} and e_{tc2} are Stata datetime values (non leap-second adjusted).
* e_{tC1} and e_{tC2} are Stata datetime values (leap-second adjusted).
* s_{tu} is a string specifying time units ("d", "h", "m", "s", or "ms").

Functions for calculating durations

Description	Function
Age	**age(**$e_{d\ DOB}$, e_d [, s_{nl}]**)**
Age with fraction	**age_frac(**$e_{d\ DOB}$, e_d [, s_{nl}]**)**
Date difference	**datediff(**e_{d1}, e_{d2}, s_{du} [, s_{nl}]**)**
Date difference with fraction	**datediff_frac(**e_{d1}, e_{d2}, s_{du} [, s_{nl}]**)**
Datetime/C difference	**Clockdiff(**e_{tC1}, e_{tC2}, s_{tu}**)**
Datetime/c difference	**clockdiff(**e_{tc1}, e_{tc2}, s_{tu}**)**
Datetime/C difference with fraction	**Clockdiff_frac(**e_{tC1}, e_{tC2}, s_{tu}**)**
Datetime/c difference with fraction	**clockdiff_frac(**e_{tc1}, e_{tc2}, s_{tu}**)**

Fictional epinephrine delivery data

Figure: Image of auto-injector. Emergency Medical Products. https://www.buyemp.com/product/epinephrine-auto-injector-2-pack. Accessed 20 Jul. 2022.

- Epinephrine is commonly used to help people with severe allergic reactions
- Consider two different methods of delivery: auto-injector and manual syringe
- Patients were administered epinephrine once with an auto-injector and once with a manual syringe
  - Each time, we recorded the time until peak epinephrine concentration
  - We wish to compute the difference in those recorded times

Fictional epinephrine delivery data

```
. use epinephrine
(Fictional data on epinephrine delivery)

. describe

Contains data from epinephrine.dta
 Observations:             8                  Fictional data on epinephrine delivery
    Variables:             4                  20 Jul 2022 14:53
-------------------------------------------------------------------------------
Variable      Storage   Display    Value
    name         type    format    label      Variable label
-------------------------------------------------------------------------------
patid           float   %9.0g                 Patient ID
weight          float   %9.0g                 Weight (lbs)
auto_inject     float   %tcmm:SS              Time to peak concentration
manual_syringe  float   %tcmm:SS              Time to peak concentration
-------------------------------------------------------------------------------
Sorted by:
```

Recorded time until peak concentration

```
. list, ab(14)
```

	patid	weight	auto_inject	manual_syringe
1.	1	126	2:35	3:05
2.	2	130	3:25	3:20
3.	3	135	8:36	9:15
4.	4	145	30:14	29:28
5.	5	164	40:29	39:35
6.	6	185	20:05	19:25
7.	7	240	35:47	28:50
8.	8	255	42:05	38:35

Simple differences

```
. gen diff = manual_syringe-auto_inject

. list auto_inject manual_syringe diff, ab(14)
```

	auto_inject	manual_syringe	diff
1.	2:35	3:05	30000
2.	3:25	3:20	-5000
3.	8:36	9:15	39000
4.	30:14	29:28	-46000
5.	40:29	39:35	-54000
6.	20:05	19:25	-40000
7.	35:47	28:50	-417000
8.	42:05	38:35	-210000

Differences in milliseconds

Difference in minutes

```
. generate min_diff = clockdiff(auto_inject, manual_syringe,  "minute")

. list auto_inject manual_syringe min_diff, ab(14)
```

	auto_inject	manual_syringe	min_diff
1.	2:35	3:05	0
2.	3:25	3:20	-1
3.	8:36	9:15	0
4.	30:14	29:28	-1
5.	40:29	39:35	-1
6.	20:05	19:25	-1
7.	35:47	28:50	-7
8.	42:05	38:35	-4

Each difference is rounded down to an integer; this is not too useful for our example.

Difference in minutes, with fractional part

```
. generate min_frac = clockdiff_frac(auto_inject, manual_syringe, "minute")

. list weight auto_inject manual_syringe min_frac, ab(14)
```

	weight	auto_inject	manual_syringe	min_frac
1.	126	2:35	3:05	.5
2.	130	3:25	3:20	-.0833333
3.	135	8:36	9:15	.65
4.	145	30:14	29:28	-.7666667
5.	164	40:29	39:35	-.9
6.	185	20:05	19:25	-.6666667
7.	240	35:47	28:50	-6.95
8.	255	42:05	38:35	-3.5

Better, but not very intuitive.

Difference in seconds

```
. generate sec_diff = clockdiff(auto_inject, manual_syringe, "second")

. list patid weight auto_inject manual_syringe sec_diff, ab(14)
```

	patid	weight	auto_inject	manual_syringe	sec_diff
1.	1	126	2:35	3:05	30
2.	2	130	3:25	3:20	-5
3.	3	135	8:36	9:15	39
4.	4	145	30:14	29:28	-46
5.	5	164	40:29	39:35	-54
6.	6	185	20:05	19:25	-40
7.	7	240	35:47	28:50	-417
8.	8	255	42:05	38:35	-210

Obtaining dates and date information from other dates

Description	Function
Birthday in year	**birthday(**$e_{d\,DOB}$**,** Y **[,** s_{nl}**])**
Previous birthday	**previousbirthday(**$e_{d\,DOB}$**,** e_d **[,** s_{nl}**])**
Next birthday	**nextbirthday(**$e_{d\,DOB}$**,** e_d **[,** s_{nl}**])**

* e_d and $e_{d\,DOB}$ are Stata dates.
* s_{nl} is a string specifying nonleap-year birthdays ("01mar" or "28feb").
* Y is a numeric year.

Obtaining dates and date information from other dates

Description	Function
Birthday in year	**birthday(**$e_{d\,DOB}$**,** Y **[,** s_{nl}**])**
Previous birthday	**previousbirthday(**$e_{d\,DOB}$**,** e_d **[,** s_{nl}**])**
Next birthday	**nextbirthday(**$e_{d\,DOB}$**,** e_d **[,** s_{nl}**])**
Days in month	**daysinmonth(**e_d**)**
First day of month	**firstdayofmonth(**e_d**)**
Last day of month	**lastdayofmonth(**e_d**)**

* e_d and $e_{d\,DOB}$ are Stata dates.
* s_{nl} is a string specifying nonleap-year birthdays ("01mar" or "28feb").
* Y is a numeric year.

Obtaining dates and date information from other dates

Description	Function
Birthday in year	**birthday(**$e_{d\,DOB}$**,** Y **[,** s_{n1}**])**
Previous birthday	**previousbirthday(**$e_{d\,DOB}$**,** e_d **[,** s_{n1}**])**
Next birthday	**nextbirthday(**$e_{d\,DOB}$**,** e_d **[,** s_{n1}**])**
Days in month	**daysinmonth(**e_d**)**
First day of month	**firstdayofmonth(**e_d**)**
Last day of month	**lastdayofmonth(**e_d**)**
Today	**today()**
Current date and time	**now()**

Obtaining dates and date information from other dates

Description	Function
Leap year indicator	**isleapyear(**Y**)**
Previous leap year	**previousleapyear(**Y**)**
Next leap year	**nextleapyear(**Y**)**
Leap second indicator	**isleapsecond(**e_{tC}**)**

* Y is a numeric year.

* e_{tC} is a Stata datetime value (leap-second adjusted).

Tell me more about these functions

[D] Datetime durations

- Compute age
- Compute the number of days, months, or years between two dates
- Compute the number of days, hours, minutes, seconds, or milliseconds between two datetimes

[D] Datetime relative dates

- Check whether a given year was a leap year
- Determine when the next leap year will be, or determine the most recent one before a given year
- Create dates based on birthdays or anniversaries in a given year
- Determine when the next birthday or anniversary will take place, or determine the most recent one before a given date

Extracting date components from daily dates

Desired component	Function	Example
Year	**datepart(**e_d**, "year")**	2012
Month	**datepart(**e_d**, "month")**	12
Day	**datepart(**e_d**, "day")**	16
Day of week (0=Sunday)	**dow(**e_d**)**	0
Julian day of year (1=first day)	**doy(**e_d**)**	351
* Week within year (1=first week)	**week(**e_d**)**	51
Quarter within year (1=first quarter)	**quarter(**e_d**)**	4

* The examples provided are for the date December 16, 2012.

* Week 52 will contain 8 days for non-leap years, and 9 days for leap years.

Extracting time-of-day components from datetime variables

	Function	
Desired component	Datetime	Datetime (UTC)
Hour of day	**hh(**e_{tc}**)**	**hhC(**e_{tC}**)**
Minutes of day	**mm(**e_{tc}**)**	**mmC(**e_{tC}**)**
Seconds of day	**ss(**e_{tc}**)**	**ssC(**e_{tC}**)**
Year, month, day, hour, minute, second, or millisecond	**clockpart(**e_{tc}**,** s_u**)**	**Clockpart(**e_{tC}**,** s_u**)**

* e_{tc} is a Stata datetime value (non-leap-second adjusted).
* e_{tC} is a Stata datetime value (leap-second adjusted).
* S_u is a string specifying time units (`"year"`, `"month"`, `"day"`, `"hour"`, `"minute"`, `"second"`, or `"millisecond"`).

Converting dates and times from other software to Stata dates and times

Working with dates from other software

- **`import excel`**, **`import sas`**, and **`import spss`** will properly convert numerically encoded dates to Stata dates

Working with dates from other software

- **`import excel`**, **`import sas`**, and **`import spss`** will properly convert numerically encoded dates to Stata dates

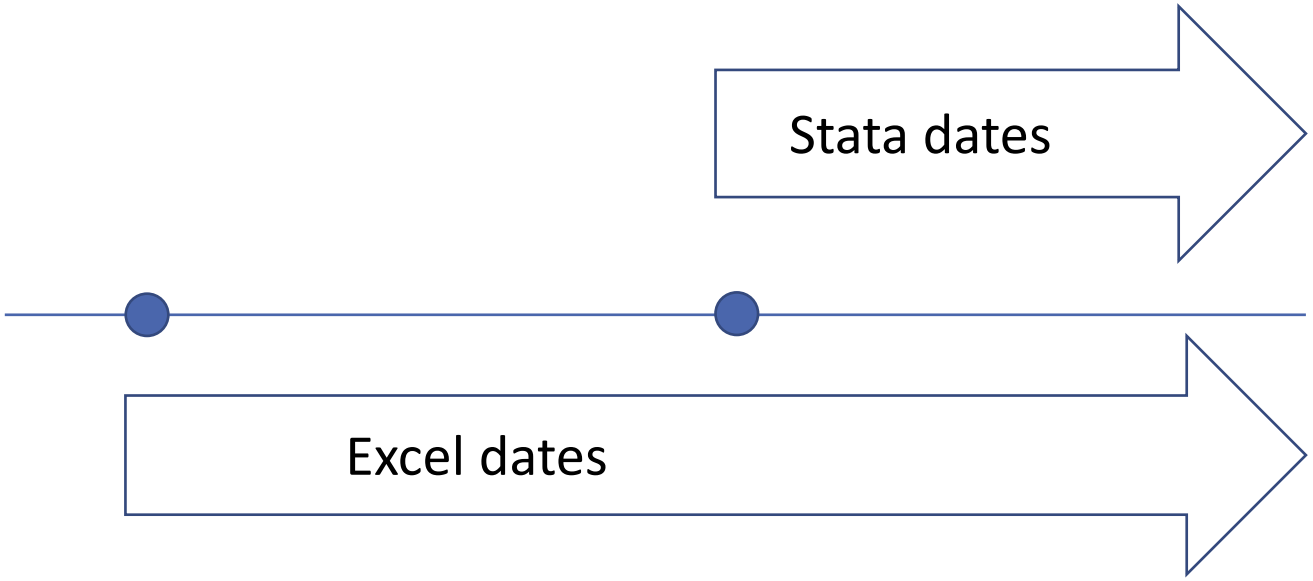
- If you export data from another software to a general format, such as .csv or .txt, and the dates are stored as the underlying numeric values that the other software used, you'll have to convert those to Stata dates.

  - If the other software has a base date earlier than Stata's, you'll have to add the number of days elapsed since that base date

  - If the other software has a base date after Stata's, you'll have to subtract the number of days elapsed since that base date

Working with dates from other software

- For dates on or after 01mar1900, Excel stores dates as days since 30dec1899.



Note that dates prior to January 1, 1960, are supported in Stata, they are simply negative.

Converting dates and times from other software

	A. Convert to a Stata date	B. Convert to a Stata datetime
SAS	`sasdate==statadate`	`sastime*1000`
SPSS	`dofc((spsstime*1000) + tc(14oct1582 00:00))`	`(spsstime*1000) + tc(14oct1582 00:00)`
R	`rdate - td(01jan1970)`	`rtime-tC(01jan1970 00:00)` (*UTC)
Excel	`xldate + td(30dec1899)`	`round((xltime+td(30dec1899))*86400)*1000`

To convert datetime values from SAS, SPSS, and Excel to datetimes adjusted for leap seconds (UTC), use the **`Cofc(`***B***`)`** conversion function, replacing *B* with the contents of column B.

Final notes

Conclusion

Today we

- converted dates and times stored as strings to numeric date and time variables
- formatted dates and times using simple and customized formats
- converted daily dates to monthly dates, and basic datetimes to UTC
- computed patients' ages
- computed the difference between two timestamps

Resources

- Documentation: Data management reference manual
- Quick guide: Dates and times in Stata
- Stata YouTube video: Creating a numeric date variable from a string variable
- The Stata Blog: A tour of datetime in Stata
- The Stata Blog: Using dates and times from other software
  (Contains advice on whether you should work with basic datetimes or with leap-second adjusted datetimes)
- The Stata Blog: Handling gaps in time series using business calendars
- Stata Technical Support: tech-support@stata.com

Thank you !