Programming for Data Analytics

Lecture 7: tidyr and lubridate

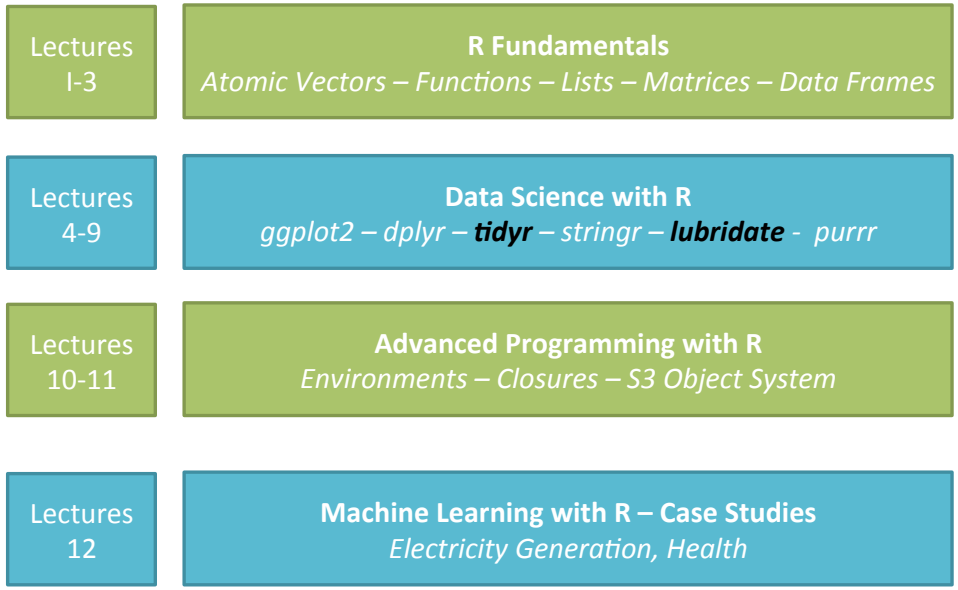
Dr. Jim Duggan,
School of Engineering & Informatics
National University of Ireland Galway.

https://twitter.com/_jimduggan

Lecture Overview

"Tidy datasets are all alike, but every messy dataset is messy in its own way."
Hadley Wickham

- Tidy data with tidyr
  - gather() & spread()
  - separate() & unite()
- lubridate
  - Reading and Processing Dates



Overview

- What is data tidying?
  - Structuring datasets to facilitate analysis
- The tidy data standard is designed to:
  - Facilitate initial exploration and analysis of data
  - Simplify the development of data analysis tools that work well together
- Principles closely related to relational algebra (Codd 1990)
- Related packages: tidyr, ggplot2, dplyr

Why tidy data? (Wickham et al. p150)

- Advantage to picking one consistent way of storing data. Easier to learn tools that work with tidy data because they have a underlying uniformity
- Specific advantage to placing variables in columns because it allows R's vectorised functions to shine.
- dplyr, ggplot2 designed to work with tidy data

Typical Structure: Rows and Columns (Wickham 2014)

	treatmenta	treatmentb
John Smith	—	2
Jane Doe	16	11
Mary Johnson	3	1

Table 1: Typical presentation dataset.

	John Smith	Jane Doe	Mary Johnson
treatmenta	—	16	3
treatmentb	2	11	1

Table 2: The same data as in Table 1 but structured differently.

Numbers refer to the result of the treatments on a given person.

Rules for a Tidy Dataset

- Each variable must have its own column
- Each observation must have its own row
- Each value must have its own cell

- *Put every dataset in a tibble*
- *Put each variable in a column*

```
> table1
# A tibble: 6 x 4
      country  year  cases population
        <chr> <int>  <int>      <int>
1 Afghanistan  1999    745   19987071
2 Afghanistan  2000   2666   20595360
3      Brazil  1999  37737  172006362
4      Brazil  2000  80488  174504898
5       China  1999 212258 1272915272
6       China  2000 213766 1280428583
```

https://rpubs.com/bradleyboehmke/data_wrangling

Example in R

```
untidy <- data.frame(
  name = c("John Smith", "Jane Doe", "Mary Johnson"),
  treatmenta = c(NA, 16, 3),
  treatmentb = c(2, 11, 1)
)
```

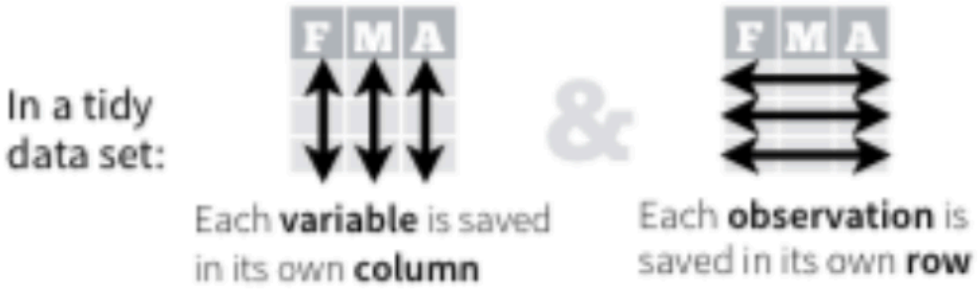
```
>
> untidy
           name treatmenta treatmentb
1    John Smith         NA          2
2      Jane Doe         16         11
3 Mary Johnson          3          1
```

In a tidy data set...

Variables

- Person (John, Jane, and Mary)
- Treatments (a or b)
- Result (6 values including NA)
- 6 observations

```
>
> untidy
           name treatmenta treatmentb
1    John Smith          NA          2
2      Jane Doe          16         11
3 Mary Johnson           3          1
```



https://rpubs.com/bradleyboehmke/data_wrangling

The goal…

```
> untidy
          name treatmenta treatmentb
1   John Smith          NA          2
2     Jane Doe          16         11
3 Mary Johnson           3          1
```

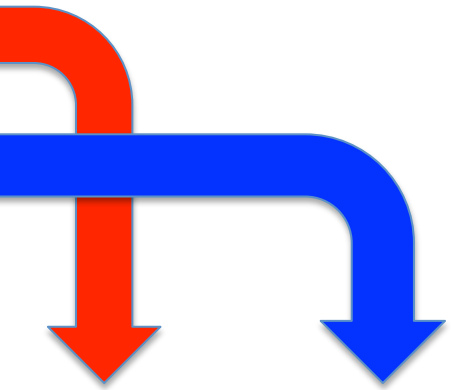
```
> tidy
          name  Treatment Outcome
1   John Smith treatmenta      NA
2     Jane Doe treatmenta      16
3 Mary Johnson treatmenta       3
4   John Smith treatmentb       2
5     Jane Doe treatmentb      11
6 Mary Johnson treatmentb       1
```

tidyr package – four fundamental functions of data tidying

- **gather()** takes multiple columns, and gathers them into key-value pairs: it makes "wide" data longer
- **spread()** takes two columns (key and value) and spreads into multiple columns, it makes long data wider
- **separate()** splits a single column into multiple columns
- **unite()** combines multiple columns into a single column

gather() process

```
>
> untidy
            name treatmenta treatmentb
1   John Smith          NA          2
2     Jane Doe          16         11
3 Mary Johnson           3          1
>
```



```
> tidy
            name  Treatment Outcome
1   John Smith  treatmenta      NA
2     Jane Doe  treatmenta      16
3 Mary Johnson  treatmenta       3
4   John Smith  treatmentb       2
5     Jane Doe  treatmentb      11
6 Mary Johnson  treatmentb       1
```

gather()

https://rpubs.com/bradleyboehmke/data_wrangling

```
Function:       gather(data, key, value, ..., na.rm = FALSE, convert = FALSE)
Same as:        data %>% gather(key, value, ..., na.rm = FALSE, convert = FALSE)

Arguments:
        data:           data frame
        key:            column name representing new variable
        value:          column name representing variable values
        ...:            names of columns to gather (or not gather)
        na.rm:          option to remove observations with missing values (represented by NAs)
        convert:        if TRUE will automatically convert values to logical, integer, numeric, complex or
                        factor as appropriate
```

```
> tidy <- gather(untidy,key=Treatment,value=Outcome,treatmenta:treatmentb)
>
> tidy
          name  Treatment Outcome
1   John Smith treatmenta      NA
2     Jane Doe treatmenta      16
3 Mary Johnson treatmenta       3
4   John Smith treatmentb       2
5     Jane Doe treatmentb      11
6 Mary Johnson treatmentb       1
```

```
> untidy
          name treatmenta treatmentb
1   John Smith          NA          2
2     Jane Doe          16         11
3 Mary Johnson           3          1
```

Challenge 7.1

- Convert the following to tidy data format

StudentID	CX1000	CX1001	CX1002	CX1003	CX1004	CX1005	CX1006	CX1007	CX1008	CX1009
1111111	56	51	78	85	63	45	55	59	52	76
1111112	56	64	68	80	70	39	46	60	55	74
1111113	52	61	63	81	71	49	54	61	54	76
1111114	50	42	72	81	63	44	62	59	56	68
1111115	67	53	77	84	65	52	63	62	52	71
1111116	45	57	62	32	61	56	62	51	55	79
1111117	67	58	54	77	75	44	58	62	57	77
1111118	69	50	66	78	72	39	60	58	57	84
1111119	70	56	62	80	71	52	60	63	54	70
1111120	51	52	46	82	74	42	66	63	55	73

spread()

https://rpubs.com/bradleyboehmke/data_wrangling

```
Function:       spread(data, key, value, fill = NA, convert = FALSE)
Same as:        data %>% spread(key, value, fill = NA, convert = FALSE)

Arguments:
        data:           data frame
        key:            column values to convert to multiple columns
        value:          single column values to convert to multiple columns' values
        fill:           If there isn't a value for every combination of the other variables and the key
                        column, this value will be substituted
        convert:        if TRUE will automatically convert values to logical, integer, numeric, complex or
                        factor as appropriate
```

```
> tidy
          name  Treatment Outcome
1   John Smith treatmenta      NA
2     Jane Doe treatmenta      16
3 Mary Johnson treatmenta       3
4   John Smith treatmentb       2
5     Jane Doe treatmentb      11
6 Mary Johnson treatmentb       1
```

```
> spread(tidy,Treatment,Outcome)
          name treatmenta treatmentb
1     Jane Doe          16          11
2   John Smith          NA           2
3 Mary Johnson           3           1
```

Spreading

- Spreading is the opposite of gathering
- Useful when observations are scattered across multiple rows

```
> table2
# A tibble: 12 x 4
       country  year       type       count
         <chr> <int>      <chr>       <int>
 1 Afghanistan  1999      cases         745
 2 Afghanistan  1999 population    19987071
 3 Afghanistan  2000      cases        2666
 4 Afghanistan  2000 population    20595360
 5      Brazil  1999      cases       37737
 6      Brazil  1999 population   172006362
 7      Brazil  2000      cases       80488
 8      Brazil  2000 population   174504898
```

To tidy up the data

- Two parameters needed
- The column that contains the variable names (key). Here it is type.
- The column that contains values from multiple variables (value). Here it's count.

```
> table2
# A tibble: 12 x 4
       country  year       type      count
         <chr> <int>      <chr>      <int>
 1 Afghanistan  1999      cases        745
 2 Afghanistan  1999 population   19987071
 3 Afghanistan  2000      cases       2666
 4 Afghanistan  2000 population   20595360
 5      Brazil  1999      cases      37737
 6      Brazil  1999 population  172006362
 7      Brazil  2000      cases      80488
 8      Brazil  2000 population  174504898
```

The spread operation…

```
> spread(table2,key=type,value=count)
# A tibble: 6 x 4
      country  year  cases population
*       <chr> <int>  <int>      <int>
1 Afghanistan  1999    745   19987071
2 Afghanistan  2000   2666   20595360
3      Brazil  1999  37737  172006362
4      Brazil  2000  80488  174504898
5       China  1999 212258 1272915272
6       China  2000 213766 1280428583
```

separate()

- Separate pulls apart one column into multiple columns
- It splits the information based on finding a non-alphanumeric character
- Separator can be defined (sep="/")
- A converter can find best type for the result, if needed.

```
> table3
# A tibble: 6 x 3
      country  year              rate
*       <chr> <int>             <chr>
1 Afghanistan  1999       745/19987071
2 Afghanistan  2000      2666/20595360
3      Brazil  1999    37737/172006362
4      Brazil  2000    80488/174504898
5       China  1999 212258/1272915272
6       China  2000 213766/1280428583
```

```
> table3 %>%
+   separate(rate,into=c("cases","population"),
+            convert=TRUE)
# A tibble: 6 x 4
      country  year  cases population
*       <chr> <int>  <int>      <int>
1 Afghanistan  1999    745   19987071
2 Afghanistan  2000   2666   20595360
3      Brazil  1999  37737  172006362
4      Brazil  2000  80488  174504898
5       China  1999 212258 1272915272
6       China  2000 213766 1280428583
```

unite()

- The inverse of separate()
- Combines multiple columns into a single column
- Can use this to revert the transformed table3 back to its original

```
> x
# A tibble: 6 x 4
      country  year  cases population
*       <chr> <int>  <int>      <int>
1 Afghanistan  1999    745   19987071
2 Afghanistan  2000   2666   20595360
3      Brazil  1999  37737  172006362
4      Brazil  2000  80488  174504898
5       China  1999 212258 1272915272
6       China  2000 213766 1280428583
```

```
> unite(x,rate,cases,population,sep="/")
# A tibble: 6 x 3
      country  year              rate
*       <chr> <int>             <chr>
1 Afghanistan  1999      745/19987071
2 Afghanistan  2000     2666/20595360
3      Brazil  1999   37737/172006362
4      Brazil  2000   80488/174504898
5       China  1999 212258/1272915272
6       China  2000 213766/1280428583
```

Summary of Functions

```
Function:       separate(data, col, into, sep = " ", remove = TRUE, convert = FALSE)
Same as:        data %>% separate(col, into, sep = " ", remove = TRUE, convert = FALSE)

Arguments:
        data:           data frame
        col:            column name representing current variable
        into:           names of variables representing new variables
        sep:            how to separate current variable (char, num, or symbol)
        remove:         if TRUE, remove input column from output data frame
        convert:        if TRUE will automatically convert values to logical, integer, numeric, complex or
                        factor as appropriate
```

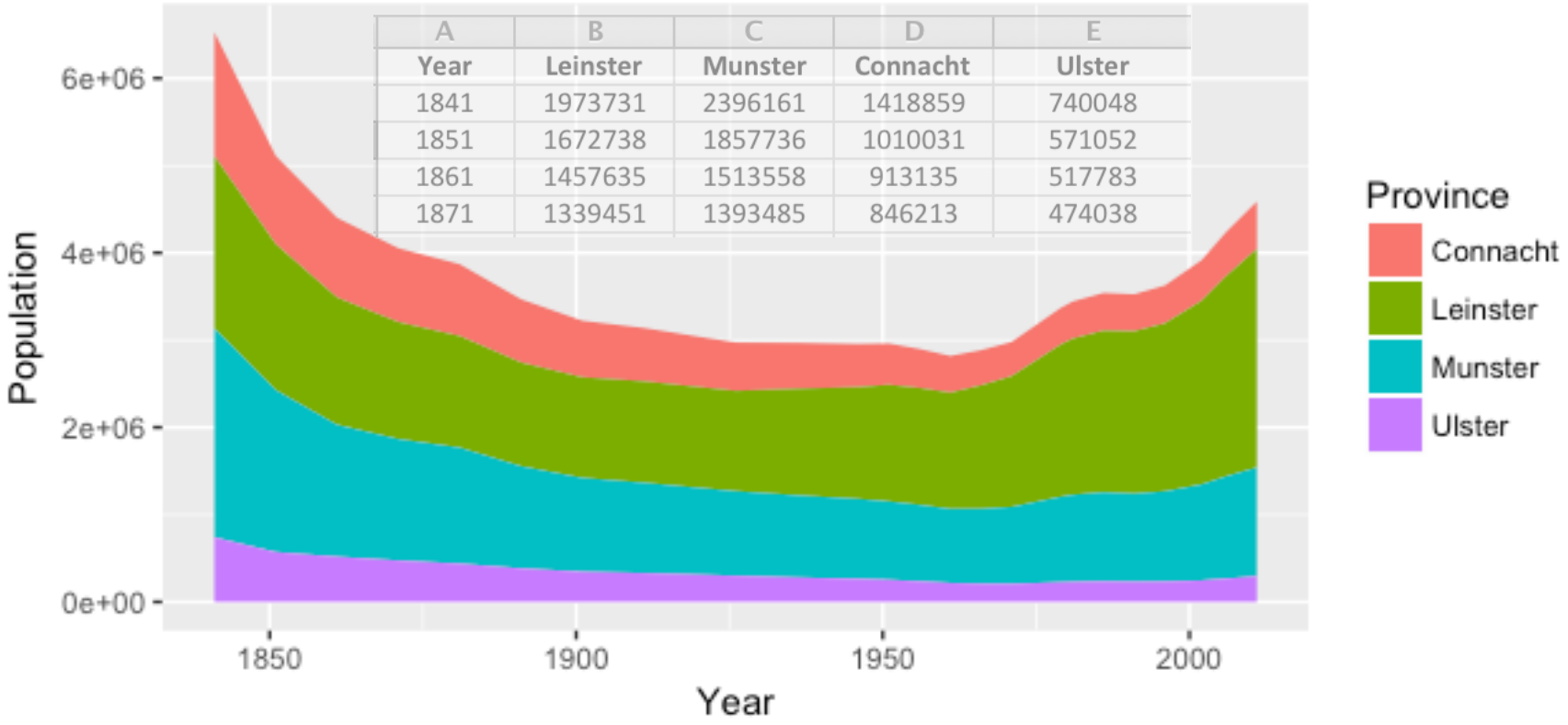
```
Function:       unite(data, col, ..., sep = " ", remove = TRUE)
Same as:        data %>% unite(col, ..., sep = " ", remove = TRUE)

Arguments:
        data:           data frame
        col:            column name of new "merged" column
        ...:            names of columns to merge
        sep:            separator to use between merged values
        remove:         if TRUE, remove input column from output data frame
```

Challenge 7.2

- Transform the census data to tidy format and create the following plot

A	B	C	D	E
Year	Leinster	Munster	Connacht	Ulster
1841	1973731	2396161	1418859	740048
1851	1672738	1857736	1010031	571052
1861	1457635	1513558	913135	517783
1871	1339451	1393485	846213	474038



lubridate package

- Helps users to:
  - Identify and parse date-time data
  - Extract and modify components of date-time, such as years, months, days, hours, minutes and seconds
  - Perform accurate calculations with date-times and timespans
  - Handle time zones and daylight savings time

Example

- Given a character string:
  - Read it in as a date-time object
  - Extract the month
  - Change the month to February
- Approaches
  - Base R method
  - lubridate method

Using Base R

```
>
> mydate <- as.POSIXct("01-01-2010",
+   format="%d-%m-%Y", tz="UTC")
>
> mydate
[1] "2010-01-01 UTC"
>
> month <- as.numeric(format(mydate,"%m"))
>
> month
[1] 1
>
> mydate <- as.POSIXct(format(mydate,
+   "%Y-2-%d"),tz="UTC")
>
> mydate
[1] "2010-02-01 UTC"
```

With lubridate

```
>
> mydate <- dmy("01-01-2010")
>
> mydate
[1] "2010-01-01"
>
> m <- month(mydate)
>
> m
[1] 1
>
> month(mydate) <- 2
>
> mydate
[1] "2010-02-01"
```

Creating Date/Times

- There are three types of date/time data that refer to an instant in time
  - A *date*. Tibbles print this as <date>
  - A *time* within a day. Tibbles print this as <time>
  - A *date-time* is a date plus a time. Tibbles print this as <dttm>
- To get the current date of date-time print today() or now()
- Otherwise there are three ways to create a date/time:
  - From a string
  - From individual date-time components
  - From an existing date/time object

Using today() and now()

```
> tibble(
+     Today=today(),
+     Now=now())
# A tibble: 1 x 2
        Today                 Now
       <date>              <dttm>
1 2017-06-28 2017-06-28 17:07:38
```

From Strings

- Date/time data often comes as strings.
- **lubridate** provides functions to work out the format once the order is specified
- Identify the order in which year, month and day appear
- Use “y” “m” and “d”

```
> ymd("2017-01-31")
[1] "2017-01-31"
>
> dmy("31-01-2017")
[1] "2017-01-31"
>
> mdy("01-31-2017")
[1] "2017-01-31"
```

To create a date-time

- To create a date-time, add an underscore and one or more of:
  - "h" for hour
  - "m" for minute
  - "s" for second

```
> ymd_h("2017-01-31 10")
[1] "2017-01-31 10:00:00 UTC"
>
> ymd_hm("2017-01-31 10:10")
[1] "2017-01-31 10:10:00 UTC"
>
> ymd_hms("2017-04-30 10:10:22")
[1] "2017-04-30 10:10:22 UTC"
>
> ymd_hms("2017-04-30 10:10:22.111")
[1] "2017-04-30 10:10:22 UTC"
```

Summary of parsing functions...

Order of elements in date-time	Parse function
year, month, day	`ymd()`
year, day, month	`ydm()`
month, day, year	`mdy()`
day, month, year	`dmy()`
hour, minute	`hm()`
hour, minute, second	`hms()`
year, month, day, hour, minute, second	`ymd_hms()`

Time zones can be used

```
> t1 <- ymd_hms("2017-07-24 10:10:22",tz="UTC")
> t1
[1] "2017-07-24 10:10:22 UTC"
>
> t2 <- ymd_hms("2017-07-24 10:10:22",tz="CET")
> t2
[1] "2017-07-24 10:10:22 CEST"
>
> abs(t2 - t1)
Time difference of 2 hours
```

Finding out time zones

```
> OlsonNames()
  [1] "Africa/Abidjan"
  [2] "Africa/Accra"

> t <- now()
>
> t1 <- ymd_hms(t,tz="America/New_York")
>
> t2 <- ymd_hms(t,tz="Europe/Dublin")
>
> t2 - t1
Time difference of -5 hours
```

Times from Individual Components

- make_date()
- make_datetime()

```
> flights %>%
+   select(year,month,day,hour,minute)
# A tibble: 336,776 x 5
    year month   day  hour minute
   <int> <int> <int> <dbl>  <dbl>
 1  2013     1     1     5     15
 2  2013     1     1     5     29
 3  2013     1     1     5     40
 4  2013     1     1     5     45
 5  2013     1     1     6      0
 6  2013     1     1     5     58
 7  2013     1     1     6      0
 8  2013     1     1     6      0
 9  2013     1     1     6      0
10  2013     1     1     6      0
# ... with 336,766 more rows
```

Examples from the flights dataset

```
> flights %>%
+   transmute(
+     TestDate = make_date(year,month,day),
+     TestTime = make_datetime(year,month,day,hour,minute)
+   )
# A tibble: 336,776 x 2
     TestDate            TestTime
        <date>              <dttm>
 1 2013-01-01 2013-01-01 05:15:00
 2 2013-01-01 2013-01-01 05:29:00
 3 2013-01-01 2013-01-01 05:40:00
 4 2013-01-01 2013-01-01 05:45:00
 5 2013-01-01 2013-01-01 06:00:00
```

Date-time Components

- Each element of a date-time object can be extracted
- Accessor functions allow this
- For month() and wday(), setting label = TRUE returns abbreviated name
- abbr = FALSE returns the full name

Date component	Accessor
Year	`year()`
Month	`month()`
Week	`week()`
Day of year	`yday()`
Day of month	`mday()`
Day of week	`wday()`
Hour	`hour()`
Minute	`minute()`
Second	`second()`
Time zone	`tz()`

Examples

```
> t1 <- ymd_hms("2017-07-24 10:10:2
> t1
[1] "2017-07-24 10:10:22 UTC"
>
>
> year(t1)
[1] 2017
>
> month(t1)
[1] 7
>
> day(t1)
[1] 24
>
> yday(t1)
[1] 205
```

Date component	Accessor
Year	`year()`
Month	`month()`
Week	`week()`
Day of year	`yday()`
Day of month	`mday()`
Day of week	`wday()`
Hour	`hour()`
Minute	`minute()`
Second	`second()`
Time zone	`tz()`

Arithmetic With Dates

```
difftime(time1, time2, tz,
        units = c("auto", "secs", "mins", "hours",
                "days", "weeks"))
```

```
> (t1 <- ymd_hms("2017-07-24 10:10:22"))
[1] "2017-07-24 10:10:22 UTC"
>
> (t2 <- ymd_hms("2017-07-25 10:30:22"))
[1] "2017-07-25 10:30:22 UTC"
>
> difftime(t2,t1,units = "days")
Time difference of 1.013889 days
>
> difftime(t2,t1,units = "hours")
Time difference of 24.33333 hours
>
> difftime(t2,t1,units = "mins")
Time difference of 1460 mins
```

Challenge 7.3

- Read in the following file and then convert the time to date format
- Calculate and plot the average wave height by hour of day

Id	Time	Long	Lat	NoZero_x	Havg_m	Tz_sec	Hmax_m	airmar data	Tsig_sec	H10_m
264014461	06/11/14 00:04	-8.235403	51.768173	64	0.95	4.2	1.99	1.39	4.5	1.72
264017993	06/11/14 00:09	-8.235403	51.768173	64	0.94	4.1	1.65	1.33	4.2	1.52
264021497	06/11/14 00:15	-8.235403	51.768173	63	0.92	4.1	1.74	1.34	4.3	1.56
264025022	06/11/14 00:21	-8.235403	51.768173	78	0.72	3.4	1.79	1.2	4	1.48
264028556	06/11/14 00:27	-8.235403	51.768173	66	0.79	4	1.72	1.19	4.6	1.47
264032091	06/11/14 00:33	-8.235403	51.768173	73	0.84	3.6	1.94	1.28	4	1.61
264035635	06/11/14 00:39	-8.235403	51.768173	69	0.9	3.8	1.87	1.37	4.6	1.69
264039177	06/11/14 00:45	-8.235403	51.768173	68	0.94	3.9	2.05	1.42	4.6	1.68
264042720	06/11/14 00:51	-8.235403	51.768173	68	0.96	4	1.87	1.43	4.4	1.71
264046220	06/11/14 00:56	-8.235403	51.768173	64	1.05	4.1	2.04	1.51	4.6	1.8
264049768	06/11/14 01:02	-8.235403	51.768173	67	0.98	4	2.91	1.62	4.5	2
264053292	06/11/14 01:08	-8.235403	51.768173	64	1	4	2.14	1.56	4.6	1.88
264056826	06/11/14 01:14	-8.235403	51.768173	68	1.03	3.9	2.29	1.59	4.3	1.93
264060350	06/11/14 01:20	-8.235403	51.768173	65	1.21	4.1	2.6	1.75	4.4	2.06