Modern R with tidyverse [Solutions]

Jonathan de Bruin, Barbara Vreede

Contents

This document is part of the workshop ***Introduction to R & Data*** *by Utrecht University RDM Support.*

Introduction

In the following exercises, we explore the migration of the Eurasian Crane (*Grus grus*), through the GPS data of 39 tagged individuals during the fall migration of 2017. The data was kindly provided for this course by Sasha Pekarsky at the Department of Ecology, Evolution, and Behavior at HUJI. The dataset included in the course materials has been modified by the course developers and may not be used for any other purposes than this workshop.

Exercises

The following exercises are labeled as follows:

- **basic exercises**: The solutions to these exercises are part of the rest of the workshop. In most cases, the code for these exercises needs to work correctly, so you will need to complete these exercises before moving on to the next section. Ask questions if you are completely stuck; if all else fails, check the solutions manual.
- **optional exercises**: These questions can help you enhance your skills, but they are not essential for this workshop, so do not worry if you do not have time to complete them all. The (+) exercises are beginner exercises, (++) exercises are of a moderate level, and (+++) exercises are really challenging.

At the end of each chapter we have noted some recommended reading, in case you have time to spare, or would like a deeper understanding of the material. All reading material is from the book R for Data Science by Garrett Grolemund and Hadley Wickham.

Technical requirements

This project depends on the `tidyverse` package. To start, load tidyverse into your work environment by running the following chunk.

```
library(tidyverse)
```

```
## -- Attaching packages ---------------------------------------------------------------------------
```

```
## v ggplot2 3.1.0       v purrr   0.3.0
## v tibble  2.0.1       v dplyr   0.8.0.1
## v tidyr   0.8.2       v stringr 1.4.0
## v readr   1.3.1       v forcats 0.4.0
```

```
## -- Conflicts ------------------------------------------------------------------------------------ t
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

1. Read and save data

Basic exercise I - Read data into R

a) Read the data `HUJI_Crane_Israel_GPRS.csv` into R

In this exercise, we read the dataset `HUJI_Crane_Israel_GPRS.csv` into the memory of the computer.

You will need to know three things to complete this exercise: - The **function** to use to read data into R; - The **location** of your data; - The **delimiter** of the data.

You can find the dataset in the `data` folder of your project. Note that the file location (argument `file`) should be `data/HUJI_Crane_Israel_GPRS.csv`, as you are referring to its location (or 'path') relative to the location of this script.

The delimiter is the character that separates the values in a tabular data file. There are various ways to find out how the datasets is delimited. One way from inside Rstudio is to use the 'Import Dataset' button (in your 'Environment') window, choose 'from Text (readr)', locate your data, and play around with the settings. An **example** of the code you can use should now appear in the bottom right window. (This will also include a function!)

Complete the code in the chunk below:

```
data_crane <- read_csv('data/HUJI_Crane_Israel_GPRS.csv')
```

```
## Parsed with column specification:
## cols(
##   .default = col_double(),
##   visible = col_logical(),
##   timestamp = col_datetime(format = ""),
##   eobs_activity = col_logical(),
##   eobs_activity_samples = col_logical(),
##   eobs_start_timestamp = col_character(),
##   eobs_status = col_character(),
##   import_marked_outlier = col_logical(),
##   orn_transmission_protocol = col_character(),
##   sensor_type = col_character(),
##   individual_taxon_canonical_name = col_character(),
##   individual_local_identifier = col_character(),
##   study_name = col_character()
## )
```

```
## See spec(...) for full column specifications.
```

```
head(data_crane)
```

```
## # A tibble: 6 x 42
##   event_id visible timestamp           location_long location_lat
```

```
##      <dbl> <lgl>   <dttm>                      <dbl>        <dbl>
## 1   3.79e9 TRUE    2017-10-15 00:00:16          40.3         55.3
## 2   3.80e9 TRUE    2017-10-15 00:00:16          40.3         55.3
## 3   3.79e9 TRUE    2017-10-15 02:00:23          40.3         55.3
## 4   3.80e9 TRUE    2017-10-15 02:00:23          40.3         55.3
## 5   3.79e9 TRUE    2017-10-15 04:00:23          40.3         55.3
## 6   3.80e9 TRUE    2017-10-15 04:00:23          40.3         55.3
## # ... with 37 more variables: acceleration_raw_x <dbl>,
## #   acceleration_raw_y <dbl>, acceleration_raw_z <dbl>,
## #   bar_barometric_height <dbl>, battery_charge_percent <dbl>,
## #   battery_charging_current <dbl>, eobs_activity <lgl>,
## #   eobs_activity_samples <lgl>, eobs_battery_voltage <dbl>,
## #   eobs_fix_battery_voltage <dbl>,
## #   eobs_horizontal_accuracy_estimate <dbl>, eobs_key_bin_checksum <dbl>,
## #   eobs_speed_accuracy_estimate <dbl>, eobs_start_timestamp <chr>,
## #   eobs_status <chr>, eobs_temperature <dbl>, eobs_type_of_fix <dbl>,
## #   eobs_used_time_to_get_fix <dbl>, external_temperature <dbl>,
## #   gps_hdop <dbl>, gps_satellite_count <dbl>, gps_time_to_fix <dbl>,
## #   ground_speed <dbl>, heading <dbl>, height_above_ellipsoid <dbl>,
## #   height_above_msl <dbl>, import_marked_outlier <lgl>,
## #   gls_light_level <dbl>, mag_magnetic_field_raw_x <dbl>,
## #   mag_magnetic_field_raw_y <dbl>, mag_magnetic_field_raw_z <dbl>,
## #   orn_transmission_protocol <chr>, tag_voltage <dbl>, sensor_type <chr>,
## #   individual_taxon_canonical_name <chr>,
## #   individual_local_identifier <chr>, study_name <chr>
```

b) Load `readxl` (to read Excel files)

`read_csv` and `read_delim` are useful functions for most tabular data files, but they are unable to read excel files. For this, we need an additional package.

The package `readxl` is a tidyverse package to read Excel files. This package does not load by default when you call `library(tidyverse)`.

Load this package with the function `library()`:

```
library(readxl)
```

c) Read the additional observations `crane_additional_observations.xlsx` into R.

Included in the *package* `readxl` (a package is a collection of functions) is the *function* `read_excel`. You can learn how to use this function by exploring its help function. To do this, type `?read_excel` in your console.

Use the function `read_excel` to read the additional observations in data file `crane_additional_observations.xlsx` (an Excel file!), also located in your `data` folder, into R.

Complete the code in the chunk below:

```
data_crane_additional <- read_excel('data/crane_additional_observations.xlsx')

head(data_crane_additional)
```

```
## # A tibble: 6 x 4
##     event_id meas_1 meas_2 meas_3
##        <dbl>  <dbl>  <dbl>  <dbl>
## 1 3794510958  1.15   1.93   0.503
## 2 3796034354  0.376  0.307  0.341
```

```
## 3 3794510962  0.217  0.829  0.963
## 4 3796034355  0.684  0.250  0.129
## 5 3794510961  1.30   1.87   1.14
## 6 3796034356  0.847  5.14   0.964
```

Basic exercise II - Dataset properties

The structure of the dataset (i.e. columns, data types, number of observations) is very important for your initial data exploration. Tidyverse has the function `glimpse()` to output the structure of a dataframe or tibble.

Output the structure of both datasets in the cell below. What do `<chr>`, `<dbl>` and `<int>` mean? Are these values what you expect?

```
glimpse(data_crane)
```

```
## Observations: 20,873
## Variables: 42
## $ event_id                           <dbl> 3794510958, 3796034354, 3794...
## $ visible                            <lgl> TRUE, TRUE, TRUE, TRUE, TRUE...
## $ timestamp                          <dttm> 2017-10-15 00:00:16, 2017-1...
## $ location_long                      <dbl> 40.31509, 40.31509, 40.31481...
## $ location_lat                       <dbl> 55.34764, 55.34764, 55.34792...
## $ acceleration_raw_x                 <dbl> NA, NA, NA, NA, NA, NA, NA, ...
## $ acceleration_raw_y                 <dbl> NA, NA, NA, NA, NA, NA, NA, ...
## $ acceleration_raw_z                 <dbl> NA, NA, NA, NA, NA, NA, NA, ...
## $ bar_barometric_height              <dbl> NA, NA, NA, NA, NA, NA, NA, ...
## $ battery_charge_percent             <dbl> NA, NA, NA, NA, NA, NA, NA, ...
## $ battery_charging_current           <dbl> NA, NA, NA, NA, NA, NA, NA, ...
## $ eobs_activity                      <lgl> NA, NA, NA, NA, NA, NA, NA, ...
## $ eobs_activity_samples              <lgl> NA, NA, NA, NA, NA, NA, NA, ...
## $ eobs_battery_voltage               <dbl> 3784, 3818, 3784, 3818, 3784...
## $ eobs_fix_battery_voltage           <dbl> NA, 3801, NA, 3798, NA, 3796...
## $ eobs_horizontal_accuracy_estimate <dbl> NA, 48.90, NA, 2.56, NA, 3.8...
## $ eobs_key_bin_checksum              <dbl> 0, 2787211109, 0, 1067160123...
## $ eobs_speed_accuracy_estimate       <dbl> NA, 1.54, NA, 0.33, NA, 0.34...
## $ eobs_start_timestamp               <chr> NA, "2017,15+Oct", NA, "2017...
## $ eobs_status                        <chr> NA, "A", NA, "A", NA, "A", N...
## $ eobs_temperature                   <dbl> NA, -7, NA, -7, NA, -8, NA, ...
## $ eobs_type_of_fix                   <dbl> NA, 3, NA, 3, NA, 3, NA, 3, ...
## $ eobs_used_time_to_get_fix          <dbl> NA, 15, NA, 22, NA, 21, NA, ...
## $ external_temperature               <dbl> NA, NA, NA, NA, NA, NA, NA, ...
## $ gps_hdop                           <dbl> NA, NA, NA, NA, NA, NA, NA, ...
## $ gps_satellite_count                <dbl> NA, NA, NA, NA, NA, NA, NA, ...
## $ gps_time_to_fix                    <dbl> NA, NA, NA, NA, NA, NA, NA, ...
## $ ground_speed                       <dbl> NA, 0.34, NA, 0.01, NA, 12.8...
## $ heading                            <dbl> NA, 340.13, NA, 0.00, NA, 22...
## $ height_above_ellipsoid             <dbl> NA, 127.0, NA, 112.2, NA, 20...
## $ height_above_msl                   <dbl> NA, NA, NA, NA, NA, NA, NA, ...
## $ import_marked_outlier              <lgl> NA, NA, NA, NA, NA, NA, NA, ...
## $ gls_light_level                    <dbl> NA, NA, NA, NA, NA, NA, NA, ...
## $ mag_magnetic_field_raw_x           <dbl> NA, NA, NA, NA, NA, NA, NA, ...
## $ mag_magnetic_field_raw_y           <dbl> NA, NA, NA, NA, NA, NA, NA, ...
## $ mag_magnetic_field_raw_z           <dbl> NA, NA, NA, NA, NA, NA, NA, ...
```

```
## $ orn_transmission_protocol         <chr> NA, NA, NA, NA, NA, NA, NA, ...
## $ tag_voltage                       <dbl> NA, NA, NA, NA, NA, NA, NA, ...
## $ sensor_type                       <chr> "gps", "gps", "gps", "gps", ...
## $ individual_taxon_canonical_name   <chr> "Grus grus", "Grus grus", "G...
## $ individual_local_identifier       <chr> "L6037", "L6037", "L6037", "...
## $ study_name                        <chr> "Huji JNF Crane israel GPRS"...
```

```
glimpse(data_crane_additional)
```

```
## Observations: 20,873
## Variables: 4
## $ event_id <dbl> 3794510958, 3796034354, 3794510962, 3796034355, 37945...
## $ meas_1   <dbl> 1.1485201, 0.3756848, 0.2170456, 0.6836040, 1.2951363...
## $ meas_2   <dbl> 1.9269063, 0.3073029, 0.8286311, 0.2501690, 1.8683569...
## $ meas_3   <dbl> 0.50257465, 0.34084276, 0.96295022, 0.12898235, 1.136...
```

Optional exercise (+) - Save data to a CSV file with delimiter ;.

Save the tibble `data_crane` to a `.csv` file. Use `;` as delimiter. To do this, you can use the function `write_delim`. Again, you can make use of its help function if you want to understand how the function works: type `?write_delim` in your console.

Tip: Use a different file name, otherwise you overwrite the original data Tip: You can save the file to a specific folder by including it in the path, just like you did when you read the file in exercise I-1. For instance: `data/other_file_name.csv` *will save your file in the* `data` *folder.*

```
# create a directory for the output file
if (!dir.exists('tmp')){
  dir.create("tmp")
}

write_delim(data_crane, 'tmp/data_crane_csv_file.csv', delim = ';')
```

Optional exercise (++) - Write tibble `data_crane` to an Excel file.

While the tidyverse package `readxl` can be used to read from Excel files, there is no corresponding tidyverse package or function to write data to an Excel file. However, the package `writexl` comes close! Within it, the function `write_xlsx` can be used to write a data frame or tibble to an Excel file.

Use this function to write the tibble `data_crane` to an Excel file (extension `.xlsx`).

```
install.packages("writexl")
```

```
## Error in contrib.url(repos, "source"): trying to use CRAN without setting a mirror
```

```
library(writexl)

# create a directory
if (!dir.exists('tmp')){
  dir.create("tmp")
}

write_xlsx(data_crane, "tmp/data_crane.xlsx")

# or, if you want to write to multiple sheets
write_xlsx(list(data_crane = data_crane, additional_measurements = data_crane_additional), "tmp/data_cra
```

Optional exercise (++) - Read and write SPSS, SAS, and STATA data files

(Statistical) software like SPSS, SAS, and STATA are popular programs in the scientific world. These programs have their own file formats, and it is not possible to read and write SPSS, SAS, and STATA data files with base R. Tidyverse offers solutions to make this possible, with the package `haven`.

Load the package with the following line of code:

```
library(haven) # to read and write SPSS, STATA and SAS files
```

Create a code block and do at least one of the following exercises:

SPSS:

- Write the `data_crane` to a SPSS data file (extension `.sav`) with the function `write_sav`.
- Read the saved SPSS data file with the function `read_sav`.

SAS:

- Write the `data_crane` to a SAS data file (extension `.sas7bdat`) with the function `write_sas`.
- Read the saved SAS data file with the function `read_sas`.

STATA:

- Write the `data_crane` to a STATA data file (extension `.dta`) with the function `write_dta`.
- Read the saved STATA data file with the function `read_dta`.

```
# create a directory
if (!dir.exists('tmp')){
  dir.create("tmp")
}

# read and write SPSS file
write_sav(data_crane, file.path("tmp", "crane_spss.sav"))
read_sav(file.path("tmp", "crane_spss.sav"))
```

```
## # A tibble: 20,873 x 42
##    event_id visible timestamp           location_long location_lat
##       <dbl>   <dbl> <dttm>                      <dbl>        <dbl>
##  1   3.79e9       1 2017-10-15 00:00:16          40.3         55.3
##  2   3.80e9       1 2017-10-15 00:00:16          40.3         55.3
##  3   3.79e9       1 2017-10-15 02:00:23          40.3         55.3
##  4   3.80e9       1 2017-10-15 02:00:23          40.3         55.3
##  5   3.79e9       1 2017-10-15 04:00:23          40.3         55.3
##  6   3.80e9       1 2017-10-15 04:00:23          40.3         55.3
##  7   3.79e9       1 2017-10-15 05:00:14          40.1         55.3
##  8   3.80e9       1 2017-10-15 05:00:14          40.1         55.3
##  9   3.80e9       1 2017-10-15 06:00:23          40.1         55.3
## 10   3.80e9       1 2017-10-15 08:00:18          40.1         55.3
## # ... with 20,863 more rows, and 37 more variables:
## #   acceleration_raw_x <dbl>, acceleration_raw_y <dbl>,
## #   acceleration_raw_z <dbl>, bar_barometric_height <dbl>,
## #   battery_charge_percent <dbl>, battery_charging_current <dbl>,
## #   eobs_activity <dbl>, eobs_activity_samples <dbl>,
## #   eobs_battery_voltage <dbl>, eobs_fix_battery_voltage <dbl>,
## #   eobs_horizontal_accuracy_estimate <dbl>, eobs_key_bin_checksum <dbl>,
## #   eobs_speed_accuracy_estimate <dbl>, eobs_start_timestamp <chr>,
## #   eobs_status <chr>, eobs_temperature <dbl>, eobs_type_of_fix <dbl>,
## #   eobs_used_time_to_get_fix <dbl>, external_temperature <dbl>,
```

```
## #   gps_hdop <dbl>, gps_satellite_count <dbl>, gps_time_to_fix <dbl>,
## #   ground_speed <dbl>, heading <dbl>, height_above_ellipsoid <dbl>,
## #   height_above_msl <dbl>, import_marked_outlier <dbl>,
## #   gls_light_level <dbl>, mag_magnetic_field_raw_x <dbl>,
## #   mag_magnetic_field_raw_y <dbl>, mag_magnetic_field_raw_z <dbl>,
## #   orn_transmission_protocol <chr>, tag_voltage <dbl>, sensor_type <chr>,
## #   individual_taxon_canonical_name <chr>,
## #   individual_local_identifier <chr>, study_name <chr>
```

```
## The SAS file will not write because a column name is too long
## So we perform a quick fix to generate shorter but unique column names
data_crane_sas <- data_crane
names(data_crane_sas) <- str_c(
  str_sub(names(data_crane_sas),1,10),
  str_sub(names(data_crane_sas),-5),
  sep="_")

# read and write SAS file
write_sas(data_crane_sas, file.path("tmp", "crane_sas.sas7bdat"))
read_sas(file.path("tmp", "crane_sas.sas7bdat"))
```

```
## # A tibble: 20,873 x 42
##    event_id_nt_id visible_sible timestamp_stamp     location_l__long
##             <dbl>         <dbl> <dttm>                         <dbl>
##  1     3794510958             1 2017-10-15 00:00:16             40.3
##  2     3796034354             1 2017-10-15 00:00:16             40.3
##  3     3794510962             1 2017-10-15 02:00:23             40.3
##  4     3796034355             1 2017-10-15 02:00:23             40.3
##  5     3794510961             1 2017-10-15 04:00:23             40.3
##  6     3796034356             1 2017-10-15 04:00:23             40.3
##  7     3794510960             1 2017-10-15 05:00:14             40.1
##  8     3796034357             1 2017-10-15 05:00:14             40.1
##  9     3796034358             1 2017-10-15 06:00:23             40.1
## 10     3796034359             1 2017-10-15 08:00:18             40.1
## # ... with 20,863 more rows, and 38 more variables:
## #   location_l_n_lat <dbl>, accelerati_raw_x <dbl>,
## #   accelerati_raw_y <dbl>, accelerati_raw_z <dbl>,
## #   bar_barome_eight <dbl>, battery_ch_rcent <dbl>,
## #   battery_ch_rrent <dbl>, eobs_activ_ivity <dbl>,
## #   eobs_activ_mples <dbl>, eobs_batte_ltage <dbl>,
## #   eobs_fix_b_ltage <dbl>, eobs_horiz_imate <dbl>,
## #   eobs_key_b_cksum <dbl>, eobs_speed_imate <dbl>,
## #   eobs_start_stamp <chr>, eobs_statu_tatus <chr>,
## #   eobs_tempe_ature <dbl>, eobs_type__f_fix <dbl>,
## #   eobs_used__t_fix <dbl>, external_t_ature <dbl>, gps_hdop__hdop <dbl>,
## #   gps_satell_count <dbl>, gps_time_t_o_fix <dbl>,
## #   ground_spe_speed <dbl>, heading_ading <dbl>, height_abo_psoid <dbl>,
## #   height_abo_e_msl <dbl>, import_mar_tlier <dbl>,
## #   gls_light__level <dbl>, mag_magnet_raw_x <dbl>,
## #   mag_magnet_raw_y <dbl>, mag_magnet_raw_z <dbl>,
## #   orn_transm_tocol <chr>, tag_voltag_ltage <dbl>,
## #   sensor_typ__type <chr>, individual__name <chr>,
## #   individual_ifier <chr>, study_name__name <chr>
```

```r
# read and write STATA file
write_dta(data_crane, file.path("tmp", "crane_stata.dta"))
read_dta(file.path("tmp", "crane_stata.dta"))
```

```
## # A tibble: 20,873 x 42
##    event_id visible timestamp           location_long location_lat
##       <dbl>   <dbl> <dttm>                      <dbl>        <dbl>
##  1   3.79e9       1 2017-10-15 00:00:16          40.3         55.3
##  2   3.80e9       1 2017-10-15 00:00:16          40.3         55.3
##  3   3.79e9       1 2017-10-15 02:00:23          40.3         55.3
##  4   3.80e9       1 2017-10-15 02:00:23          40.3         55.3
##  5   3.79e9       1 2017-10-15 04:00:23          40.3         55.3
##  6   3.80e9       1 2017-10-15 04:00:23          40.3         55.3
##  7   3.79e9       1 2017-10-15 05:00:14          40.1         55.3
##  8   3.80e9       1 2017-10-15 05:00:14          40.1         55.3
##  9   3.80e9       1 2017-10-15 06:00:23          40.1         55.3
## 10   3.80e9       1 2017-10-15 08:00:18          40.1         55.3
## # ... with 20,863 more rows, and 37 more variables:
## #   acceleration_raw_x <dbl>, acceleration_raw_y <dbl>,
## #   acceleration_raw_z <dbl>, bar_barometric_height <dbl>,
## #   battery_charge_percent <dbl>, battery_charging_current <dbl>,
## #   eobs_activity <dbl>, eobs_activity_samples <dbl>,
## #   eobs_battery_voltage <dbl>, eobs_fix_battery_voltage <dbl>,
## #   eobs_horizontal_accuracy_estimate <dbl>, eobs_key_bin_checksum <dbl>,
## #   eobs_speed_accuracy_estimate <dbl>, eobs_start_timestamp <chr>,
## #   eobs_status <chr>, eobs_temperature <dbl>, eobs_type_of_fix <dbl>,
## #   eobs_used_time_to_get_fix <dbl>, external_temperature <dbl>,
## #   gps_hdop <dbl>, gps_satellite_count <dbl>, gps_time_to_fix <dbl>,
## #   ground_speed <dbl>, heading <dbl>, height_above_ellipsoid <dbl>,
## #   height_above_msl <dbl>, import_marked_outlier <dbl>,
## #   gls_light_level <dbl>, mag_magnetic_field_raw_x <dbl>,
## #   mag_magnetic_field_raw_y <dbl>, mag_magnetic_field_raw_z <dbl>,
## #   orn_transmission_protocol <chr>, tag_voltage <dbl>, sensor_type <chr>,
## #   individual_taxon_canonical_name <chr>,
## #   individual_local_identifier <chr>, study_name <chr>
```

Optional exercise (+++) - Parse datetime columns

Date and time variables can be very challenging to work with in programming languages (including R). In the `data_crane` dataset, there is a variable `eobs_start_timestamp`. This column is not recognized as a date variable, as you can see when you glimpse the data:

```r
glimpse(data_crane)
```

```
## Observations: 20,873
## Variables: 42
## $ event_id                          <dbl> 3794510958, 3796034354, 3794...
## $ visible                           <lgl> TRUE, TRUE, TRUE, TRUE, TRUE...
## $ timestamp                         <dttm> 2017-10-15 00:00:16, 2017-1...
## $ location_long                     <dbl> 40.31509, 40.31509, 40.31481...
## $ location_lat                      <dbl> 55.34764, 55.34764, 55.34792...
## $ acceleration_raw_x                <dbl> NA, NA, NA, NA, NA, NA, NA, ...
## $ acceleration_raw_y                <dbl> NA, NA, NA, NA, NA, NA, NA, ...
## $ acceleration_raw_z                <dbl> NA, NA, NA, NA, NA, NA, NA, ...
```

```
## $ bar_barometric_height             <dbl> NA, NA, NA, NA, NA, NA, NA, ...
## $ battery_charge_percent            <dbl> NA, NA, NA, NA, NA, NA, NA, ...
## $ battery_charging_current          <dbl> NA, NA, NA, NA, NA, NA, NA, ...
## $ eobs_activity                     <lgl> NA, NA, NA, NA, NA, NA, NA, ...
## $ eobs_activity_samples             <lgl> NA, NA, NA, NA, NA, NA, NA, ...
## $ eobs_battery_voltage              <dbl> 3784, 3818, 3784, 3818, 3784...
## $ eobs_fix_battery_voltage          <dbl> NA, 3801, NA, 3798, NA, 3796...
## $ eobs_horizontal_accuracy_estimate <dbl> NA, 48.90, NA, 2.56, NA, 3.8...
## $ eobs_key_bin_checksum             <dbl> 0, 2787211109, 0, 1067160123...
## $ eobs_speed_accuracy_estimate      <dbl> NA, 1.54, NA, 0.33, NA, 0.34...
## $ eobs_start_timestamp              <chr> NA, "2017,15+Oct", NA, "2017...
## $ eobs_status                       <chr> NA, "A", NA, "A", NA, "A", N...
## $ eobs_temperature                  <dbl> NA, -7, NA, -7, NA, -8, NA, ...
## $ eobs_type_of_fix                  <dbl> NA, 3, NA, 3, NA, 3, NA, 3, ...
## $ eobs_used_time_to_get_fix         <dbl> NA, 15, NA, 22, NA, 21, NA, ...
## $ external_temperature              <dbl> NA, NA, NA, NA, NA, NA, NA, ...
## $ gps_hdop                          <dbl> NA, NA, NA, NA, NA, NA, NA, ...
## $ gps_satellite_count               <dbl> NA, NA, NA, NA, NA, NA, NA, ...
## $ gps_time_to_fix                   <dbl> NA, NA, NA, NA, NA, NA, NA, ...
## $ ground_speed                      <dbl> NA, 0.34, NA, 0.01, NA, 12.8...
## $ heading                           <dbl> NA, 340.13, NA, 0.00, NA, 22...
## $ height_above_ellipsoid            <dbl> NA, 127.0, NA, 112.2, NA, 20...
## $ height_above_msl                  <dbl> NA, NA, NA, NA, NA, NA, NA, ...
## $ import_marked_outlier             <lgl> NA, NA, NA, NA, NA, NA, NA, ...
## $ gls_light_level                   <dbl> NA, NA, NA, NA, NA, NA, NA, ...
## $ mag_magnetic_field_raw_x          <dbl> NA, NA, NA, NA, NA, NA, NA, ...
## $ mag_magnetic_field_raw_y          <dbl> NA, NA, NA, NA, NA, NA, NA, ...
## $ mag_magnetic_field_raw_z          <dbl> NA, NA, NA, NA, NA, NA, NA, ...
## $ orn_transmission_protocol         <chr> NA, NA, NA, NA, NA, NA, NA, ...
## $ tag_voltage                       <dbl> NA, NA, NA, NA, NA, NA, NA, ...
## $ sensor_type                       <chr> "gps", "gps", "gps", "gps", ...
## $ individual_taxon_canonical_name   <chr> "Grus grus", "Grus grus", "G...
## $ individual_local_identifier       <chr> "L6037", "L6037", "L6037", "...
## $ study_name                        <chr> "Huji JNF Crane israel GPRS"...
```

The following code shows how you can columns with a specific data type. As in excercise 1.I (a), you can use the 'Import Dataset' button (in your 'Environment') window to play around with the settings, and see an example of the code this corresponds to in the bottom right window.

Insert the correct format for the date column:

```
read_csv('data/HUJI_Crane_Israel_GPRS.csv',
  col_types = cols(
    . = col_guess(),
    eobs_start_timestamp = col_date(format="%Y,%d+%b")
  ))
```

```
## Warning: The following named parsers don't match the column names: .
```

```
## # A tibble: 20,873 x 42
##    event_id visible timestamp           location_long location_lat
##       <dbl> <lgl>   <dttm>                      <dbl>        <dbl>
##  1   3.79e9 TRUE    2017-10-15 00:00:16          40.3         55.3
##  2   3.80e9 TRUE    2017-10-15 00:00:16          40.3         55.3
##  3   3.79e9 TRUE    2017-10-15 02:00:23          40.3         55.3
##  4   3.80e9 TRUE    2017-10-15 02:00:23          40.3         55.3
```

```
##  5   3.79e9 TRUE    2017-10-15 04:00:23          40.3         55.3
##  6   3.80e9 TRUE    2017-10-15 04:00:23          40.3         55.3
##  7   3.79e9 TRUE    2017-10-15 05:00:14          40.1         55.3
##  8   3.80e9 TRUE    2017-10-15 05:00:14          40.1         55.3
##  9   3.80e9 TRUE    2017-10-15 06:00:23          40.1         55.3
## 10   3.80e9 TRUE    2017-10-15 08:00:18          40.1         55.3
## # ... with 20,863 more rows, and 37 more variables:
## #   acceleration_raw_x <dbl>, acceleration_raw_y <dbl>,
## #   acceleration_raw_z <dbl>, bar_barometric_height <dbl>,
## #   battery_charge_percent <dbl>, battery_charging_current <dbl>,
## #   eobs_activity <lgl>, eobs_activity_samples <lgl>,
## #   eobs_battery_voltage <dbl>, eobs_fix_battery_voltage <dbl>,
## #   eobs_horizontal_accuracy_estimate <dbl>, eobs_key_bin_checksum <dbl>,
## #   eobs_speed_accuracy_estimate <dbl>, eobs_start_timestamp <date>,
## #   eobs_status <chr>, eobs_temperature <dbl>, eobs_type_of_fix <dbl>,
## #   eobs_used_time_to_get_fix <dbl>, external_temperature <dbl>,
## #   gps_hdop <dbl>, gps_satellite_count <dbl>, gps_time_to_fix <dbl>,
## #   ground_speed <dbl>, heading <dbl>, height_above_ellipsoid <dbl>,
## #   height_above_msl <dbl>, import_marked_outlier <lgl>,
## #   gls_light_level <dbl>, mag_magnetic_field_raw_x <dbl>,
## #   mag_magnetic_field_raw_y <dbl>, mag_magnetic_field_raw_z <dbl>,
## #   orn_transmission_protocol <chr>, tag_voltage <dbl>, sensor_type <chr>,
## #   individual_taxon_canonical_name <chr>,
## #   individual_local_identifier <chr>, study_name <chr>
```

Recommended reading - `readr` versus base R

You might wonder why we are not using base R functions for (flat) file reading like `read.csv()`, `read.csv2()` and `read.delimiter()`. These function are available by default in base R and do not require additional packages like `readr`.

Read Chapter 11.2.1 of R for Data Science and think about topics like reproducibility, shareablity, and performance.

2. Data visualisation

Visualizing data is an important step in getting a feel for the content of a data set. For the crane dataset, visualisation is used to explore the values of the columns. Base R has strong plotting functions. To get even better flexibility in plotting, we use the ggplot2 functions `qplot` and `ggplot`.

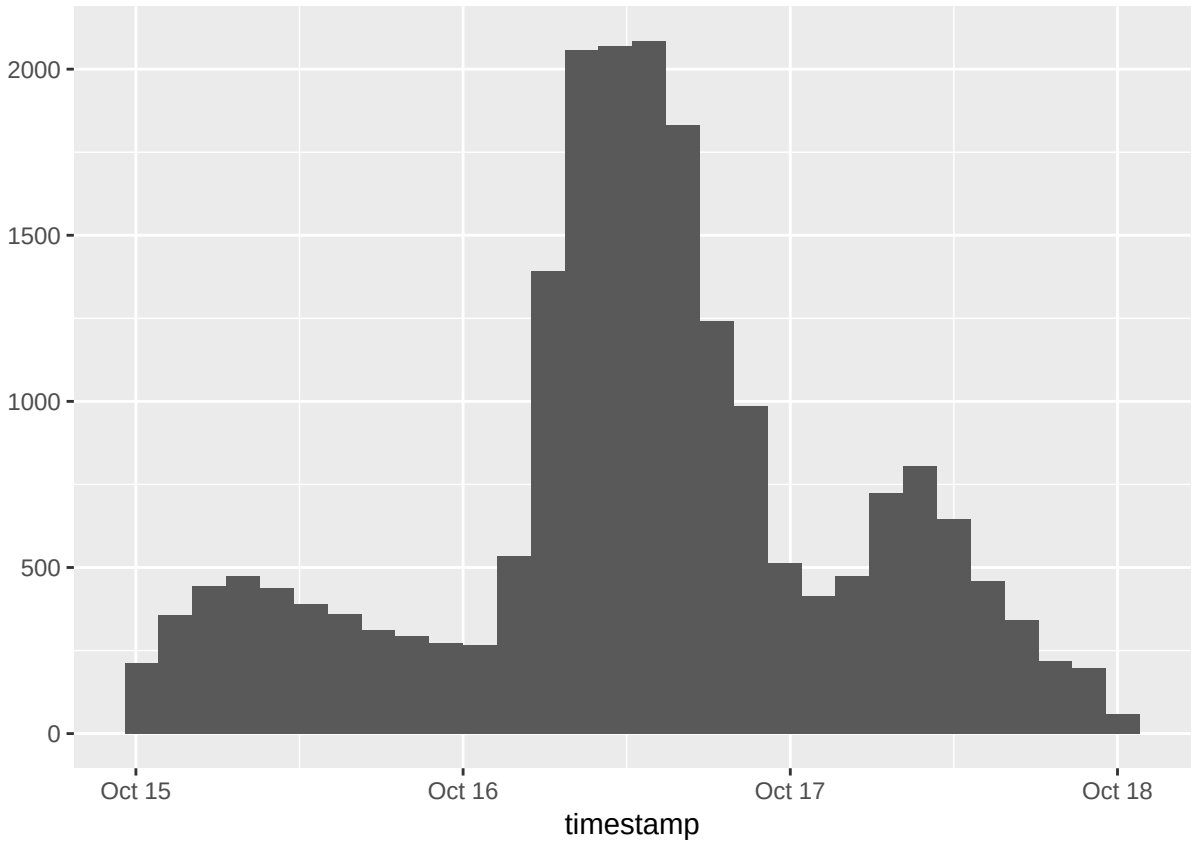
Basic exercise I - Quick plots of the `data_crane`.

a) Single column plots

Make a quick plot of at least 5 columns in the `data_crane` dataset. One example has been provided already.

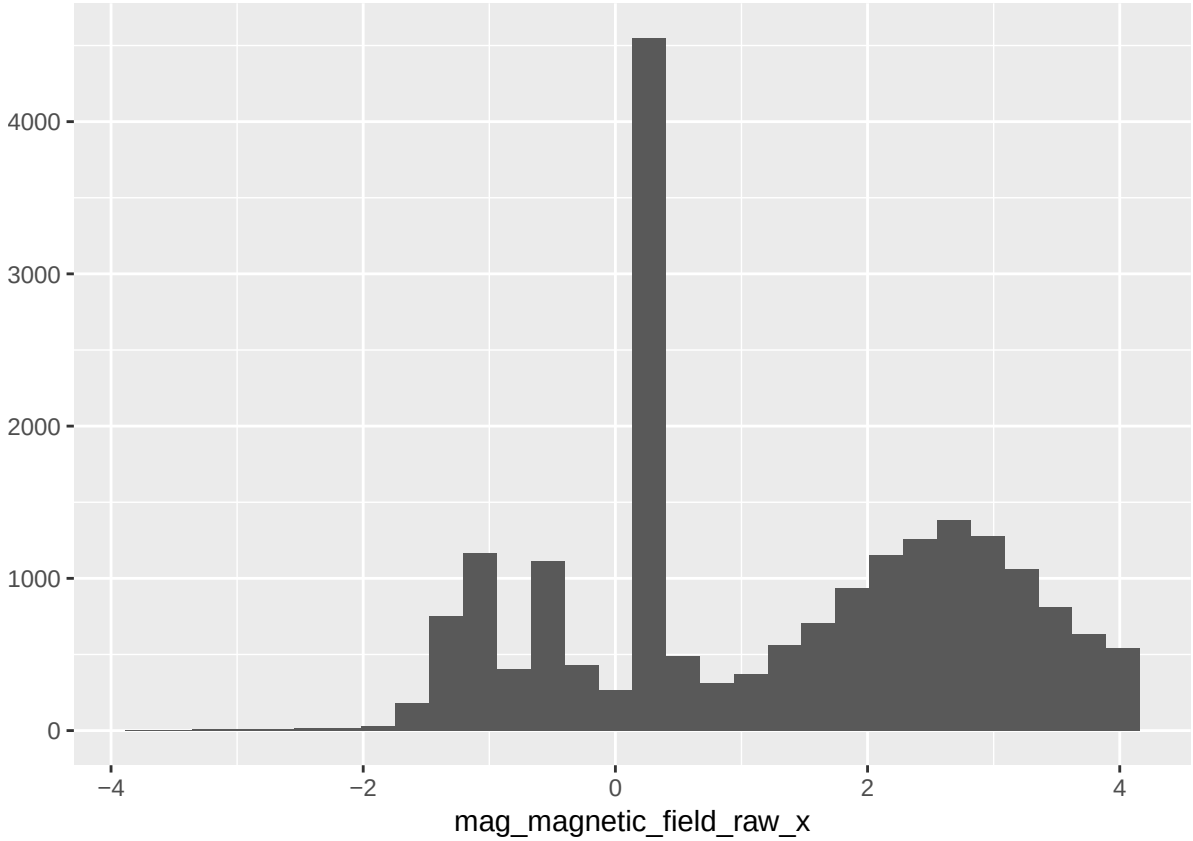
```
qplot(timestamp, data=data_crane)
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



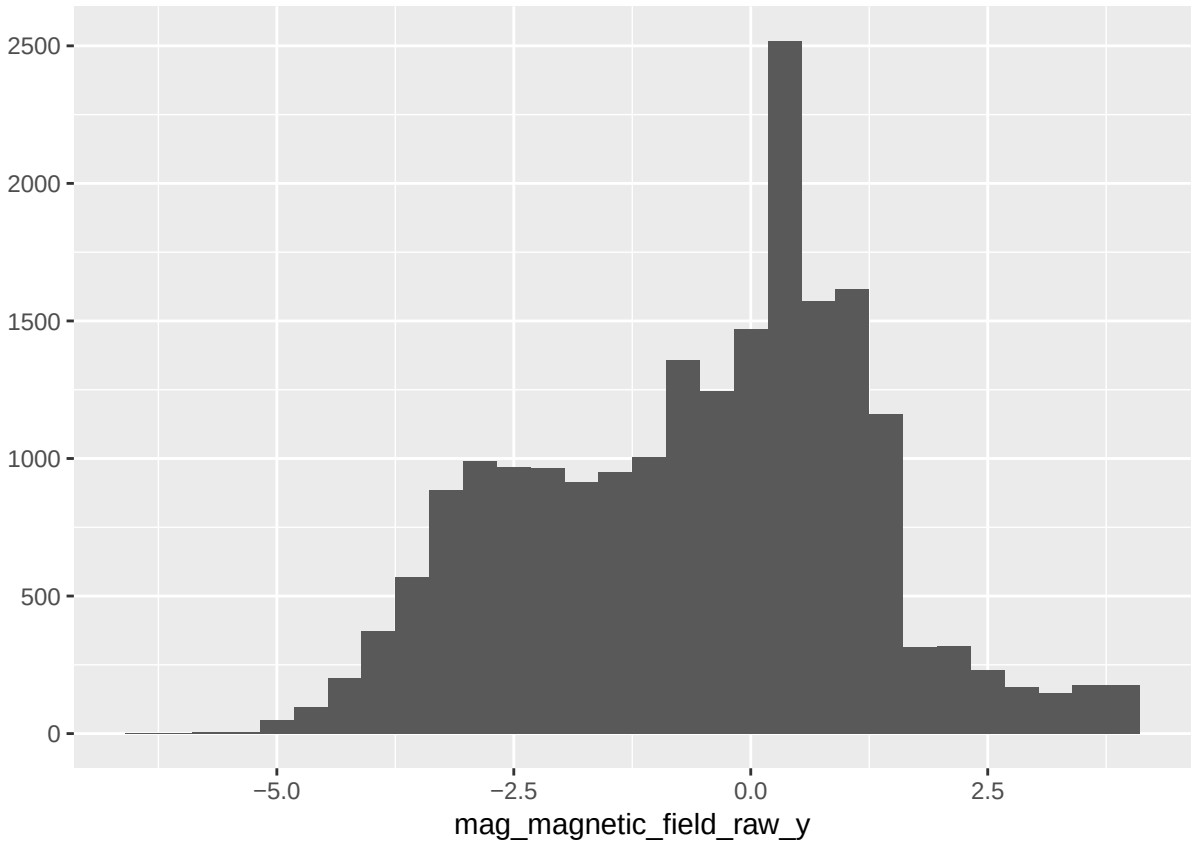

```
qplot(mag_magnetic_field_raw_x, data=data_crane)
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## Warning: Removed 432 rows containing non-finite values (stat_bin).
```



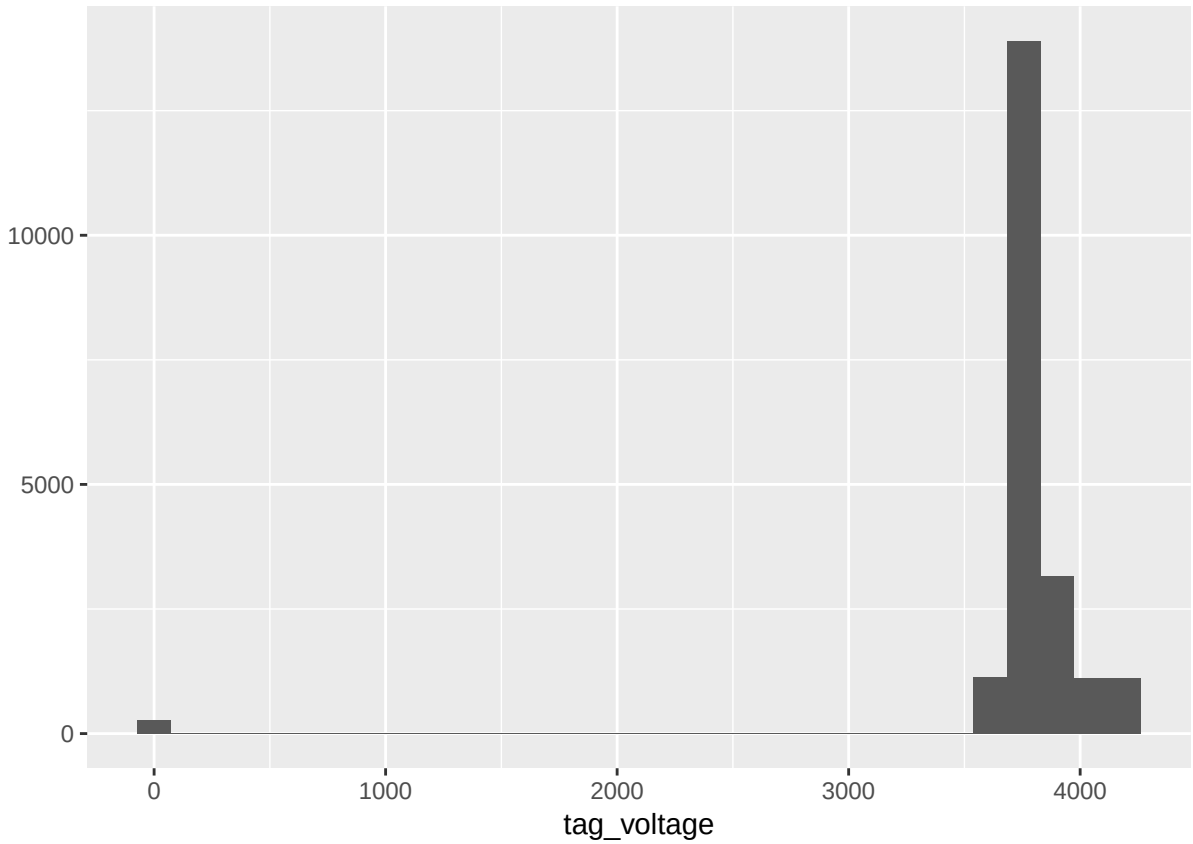

```
qplot(mag_magnetic_field_raw_y, data=data_crane)
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## Warning: Removed 432 rows containing non-finite values (stat_bin).
```

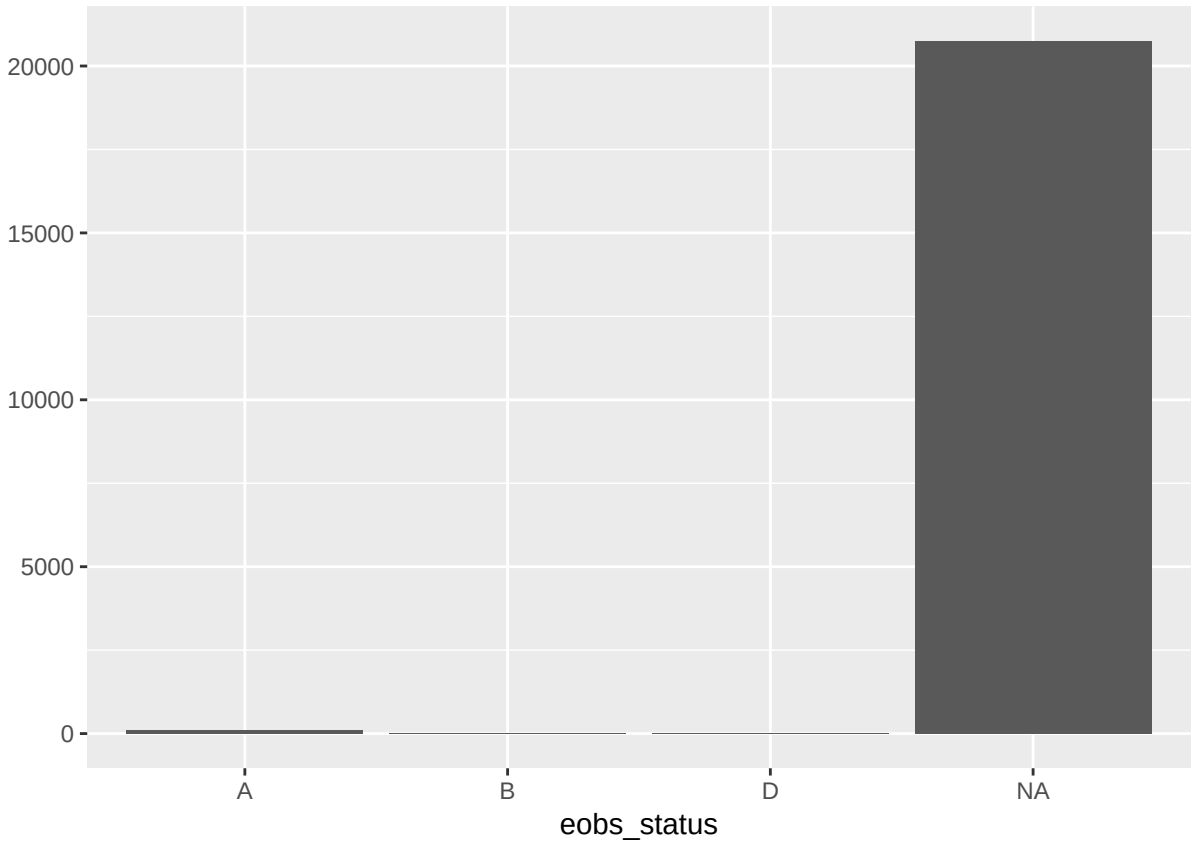



```
qplot(tag_voltage, data=data_crane)
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## Warning: Removed 155 rows containing non-finite values (stat_bin).
```




```
qplot(eobs_status, data=data_crane)
```

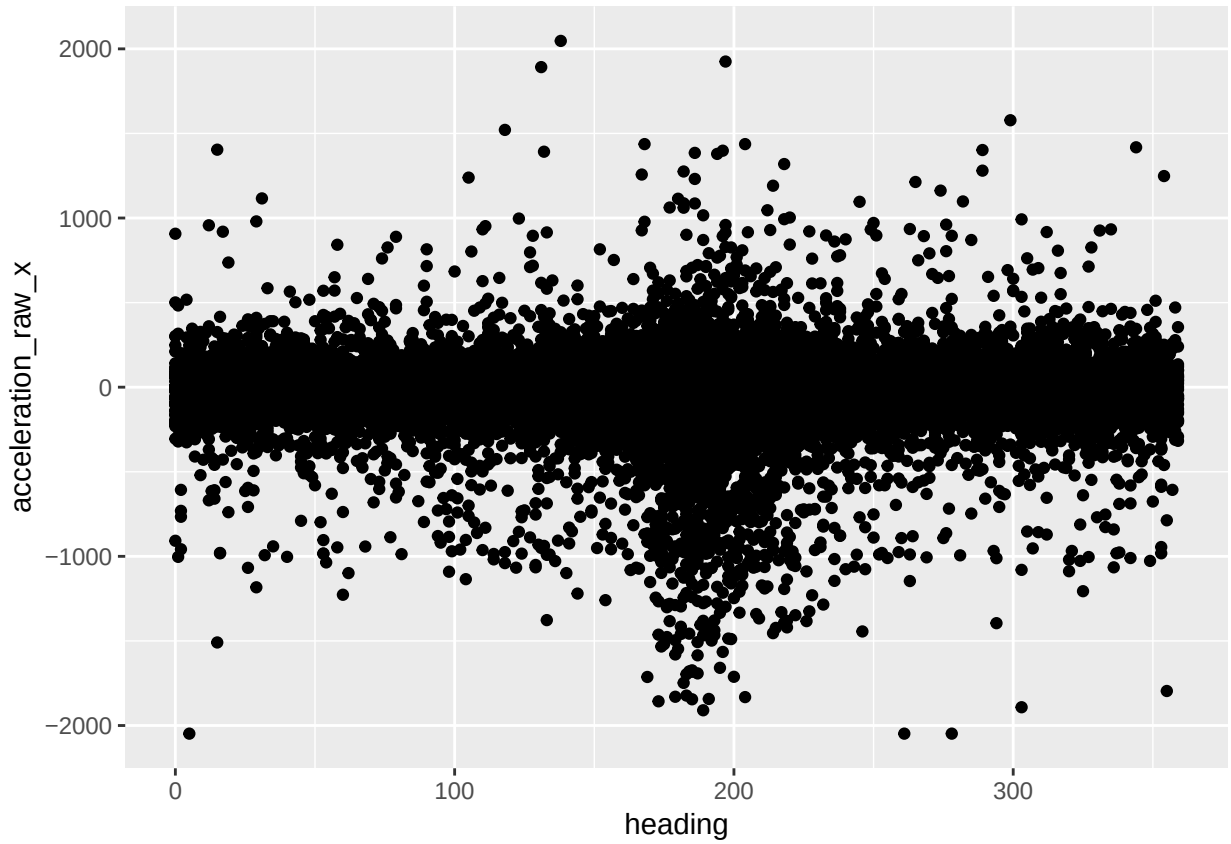


b) Two column plots

Make a few quick plots of at least 3 combinations of columns. One example has been provided already.

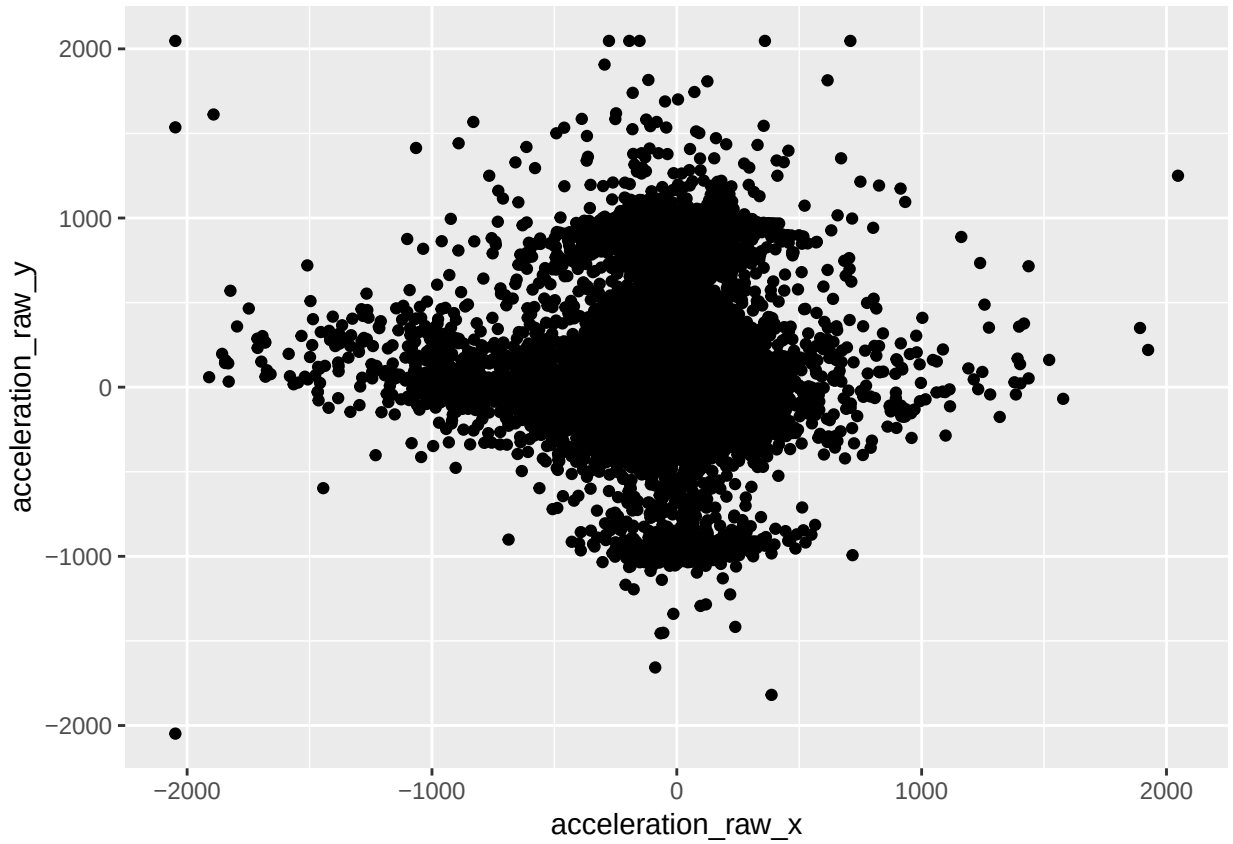
```
qplot(heading, acceleration_raw_x, data=data_crane)
```

```
## Warning: Removed 432 rows containing missing values (geom_point).
```



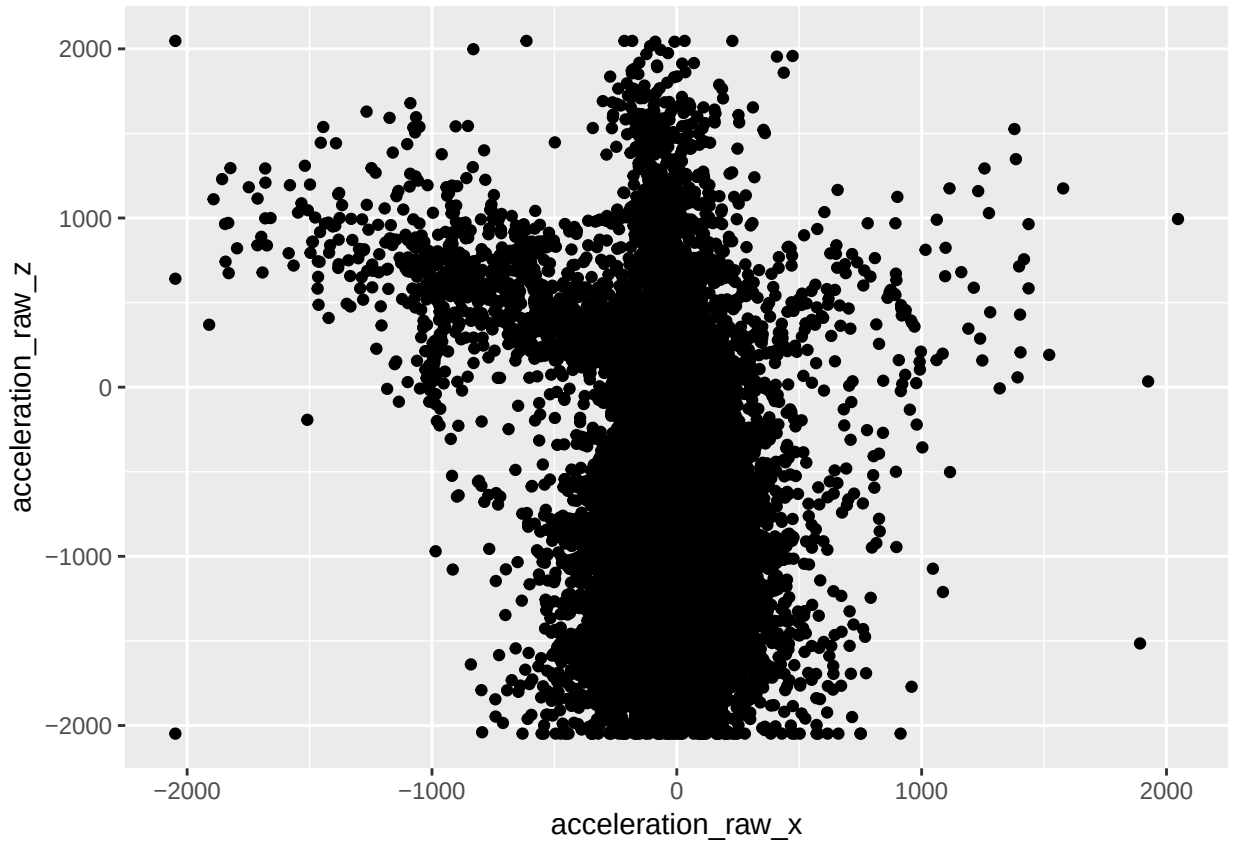

```
qplot(acceleration_raw_x, acceleration_raw_y, data=data_crane)
```

```
## Warning: Removed 432 rows containing missing values (geom_point).
```



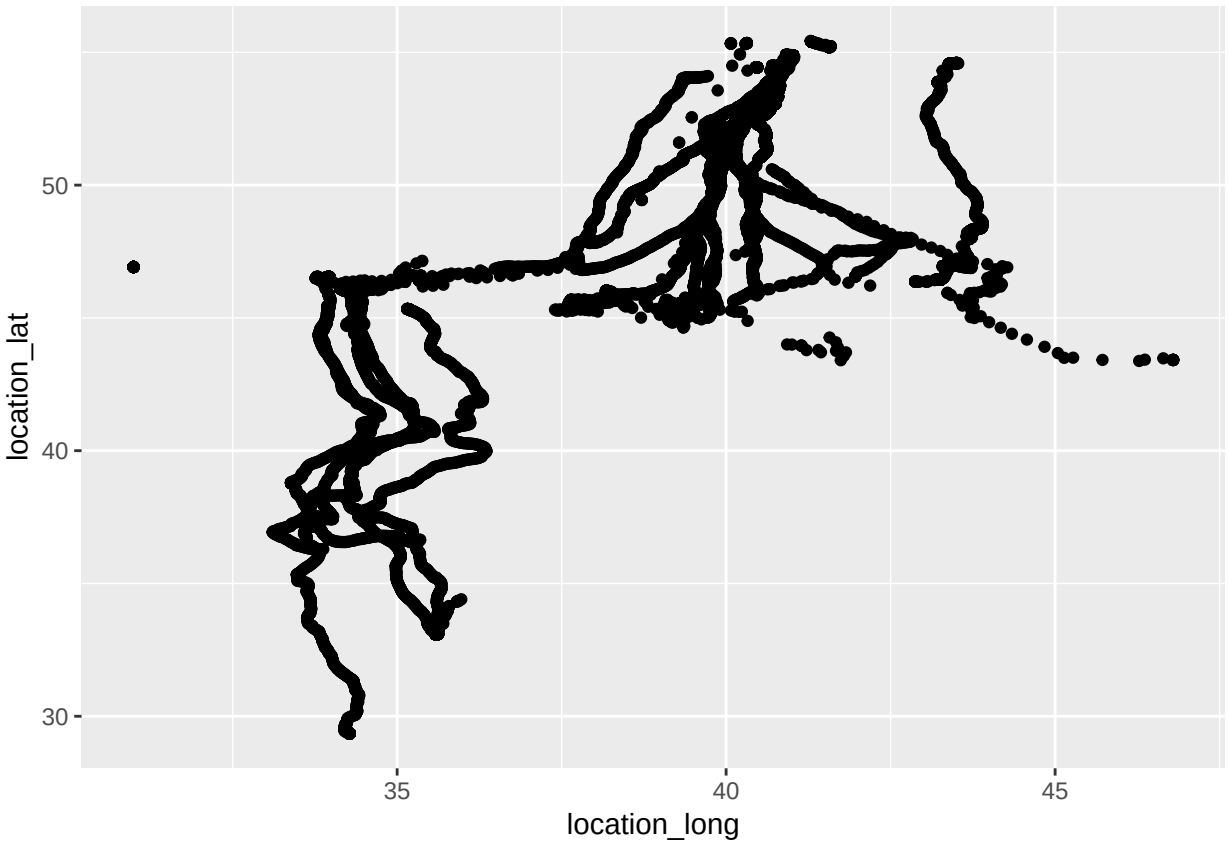

```
qplot(acceleration_raw_x, acceleration_raw_z, data=data_crane)
```

```
## Warning: Removed 432 rows containing missing values (geom_point).
```




```
qplot(location_long, location_lat, data=data_crane)
```

```
## Warning: Removed 200 rows containing missing values (geom_point).
```



Basic exercise II - Using ggplot for plotting

The power of ggplot is in the use of different layers that build up an image. Your plot consists of your data, its mapping to a plot, and one or multiple geometric layers that translate the data to a visual representation.

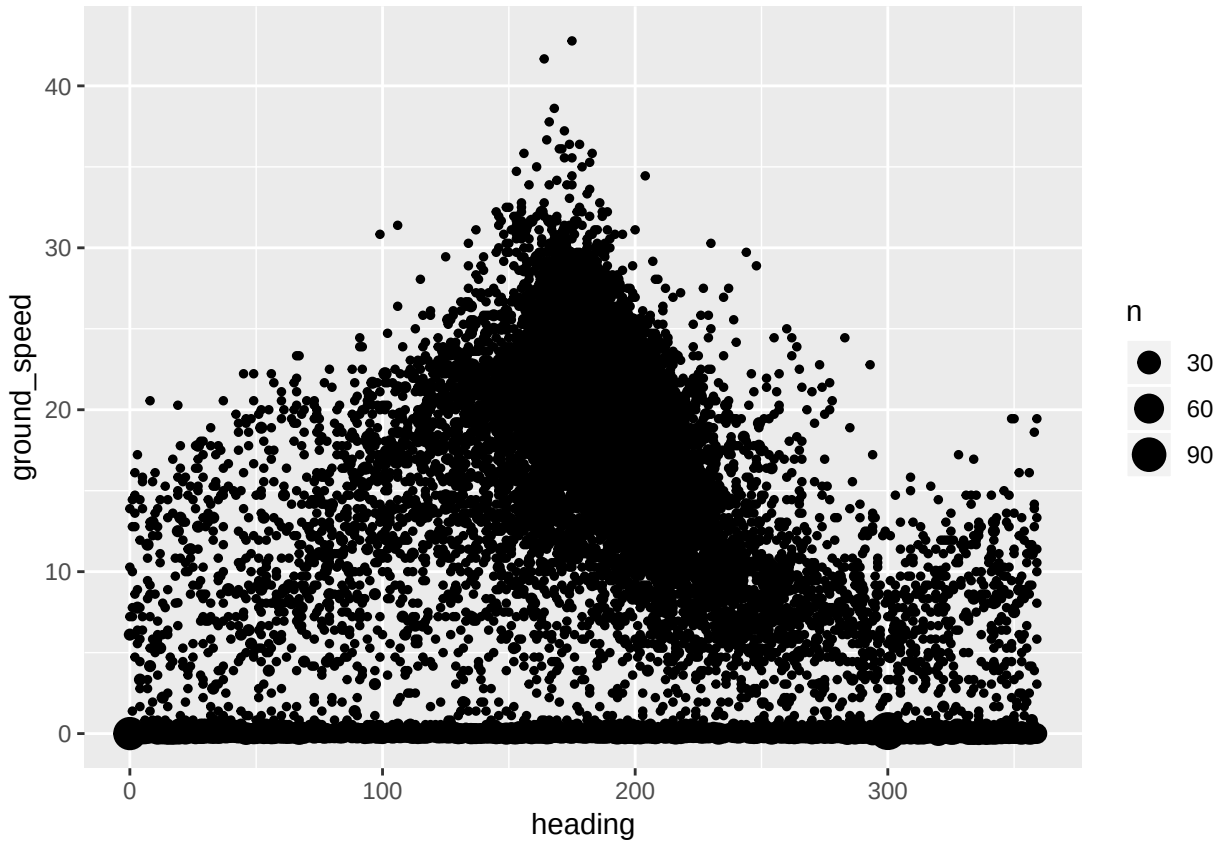
Play with that translation in showing the cranes' ground speed by the angle in which they are heading. Try various geometric layers and see what they do. Examples are `geom_point`, `geom_line`, `geom_count` and `geom_density_2d`, but feel free to search for others!

Tip: write `geom_` in the search field of the help tab in RStudio to find more geometric layers.

Complete the following code. (Note that the plot is first generated as p, but not printed yet: it is saved as a variable. You can then always call p and add layers to it.)

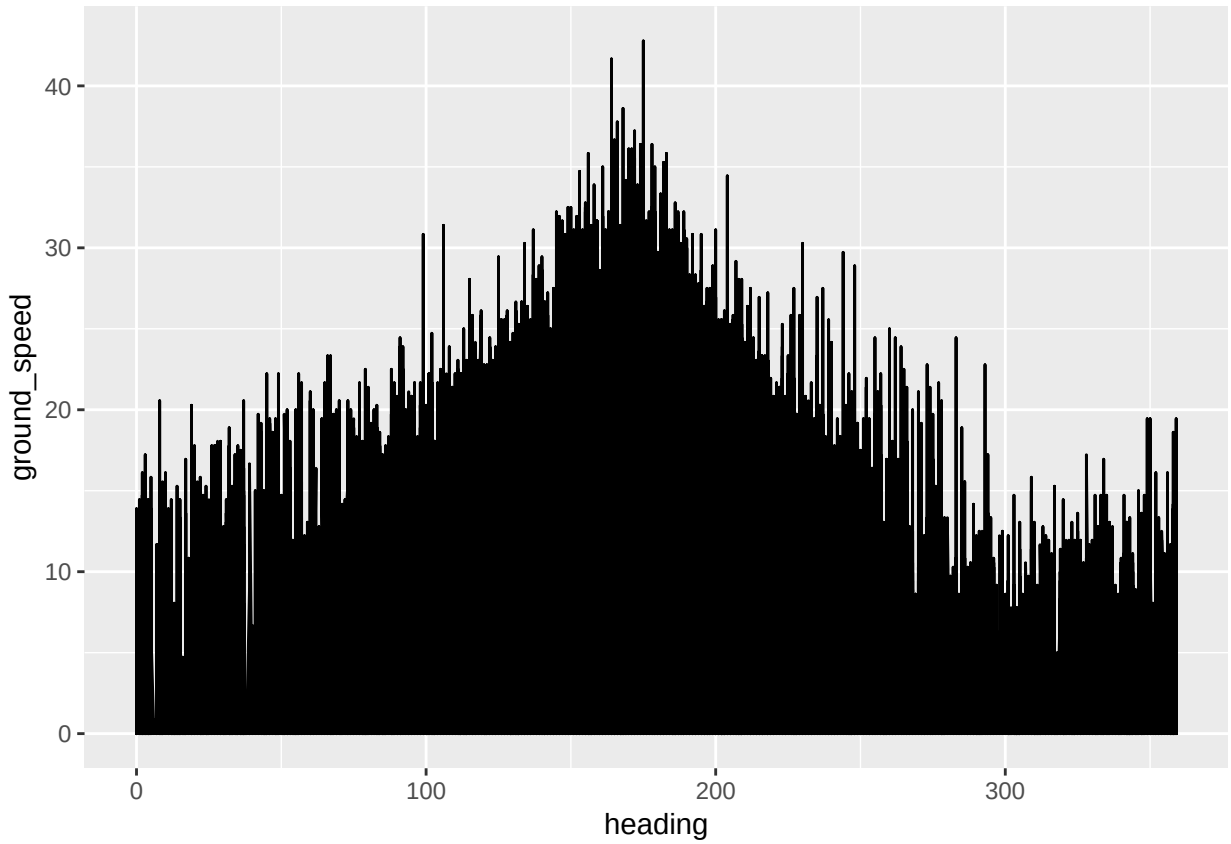
```
p <- ggplot(data_crane, aes(x = heading, y = ground_speed))

p + geom_count()
```

```
## Warning: Removed 40 rows containing non-finite values (stat_sum).
```




```
p + geom_line()
```

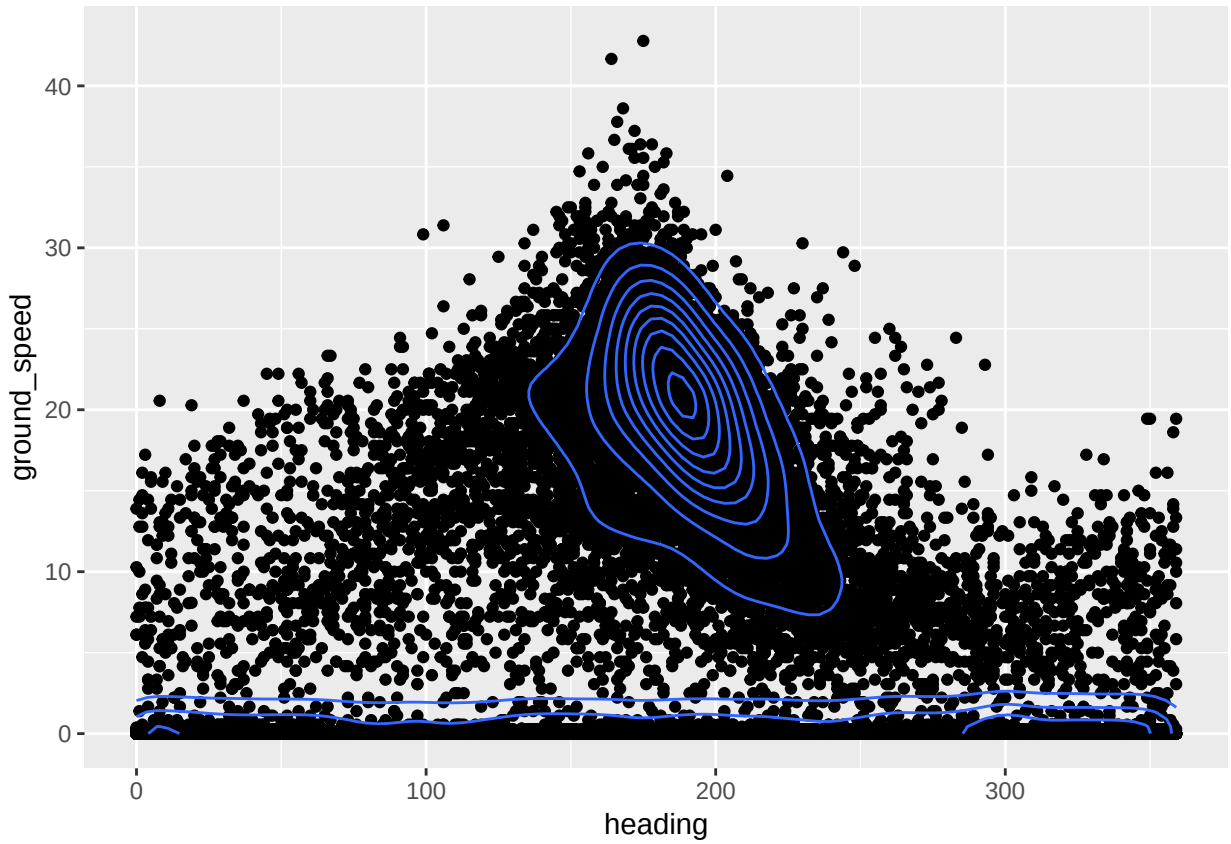
```
## Warning: Removed 40 rows containing missing values (geom_path).
```




```
p + geom_point() + geom_density_2d()
```

```
## Warning: Removed 40 rows containing non-finite values (stat_density2d).
```

```
## Warning: Removed 40 rows containing missing values (geom_point).
```



Optional exercise (+) - Statistical layers for graphs.

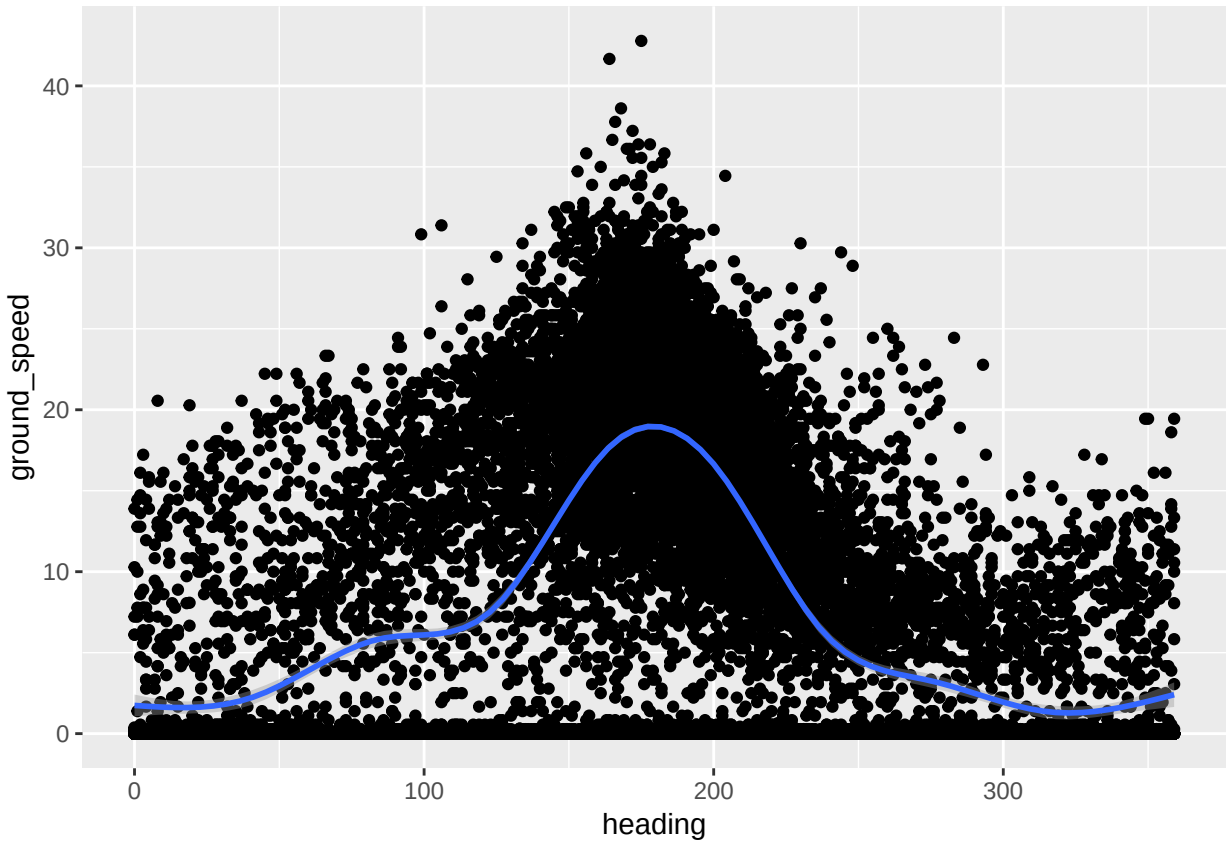
Statistical layers reveal a strong power of ggplot, and can be used as stand-alone geometries, or layered onto existing plots. In this exercise, add a smooth line to a scatterplot with the plot `p` from the previous exercise, summarizing the data. Consider the functions `stat_smooth()` and `geom_smooth()` for your statistical layer. What is the difference between these functions?

Tip: take a look at the recommended reading for this chapter.

Complete the code below:

```
p + geom_point() + geom_smooth()
```

```
## `geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'
## Warning: Removed 40 rows containing non-finite values (stat_smooth).
## Warning: Removed 40 rows containing missing values (geom_point).
```



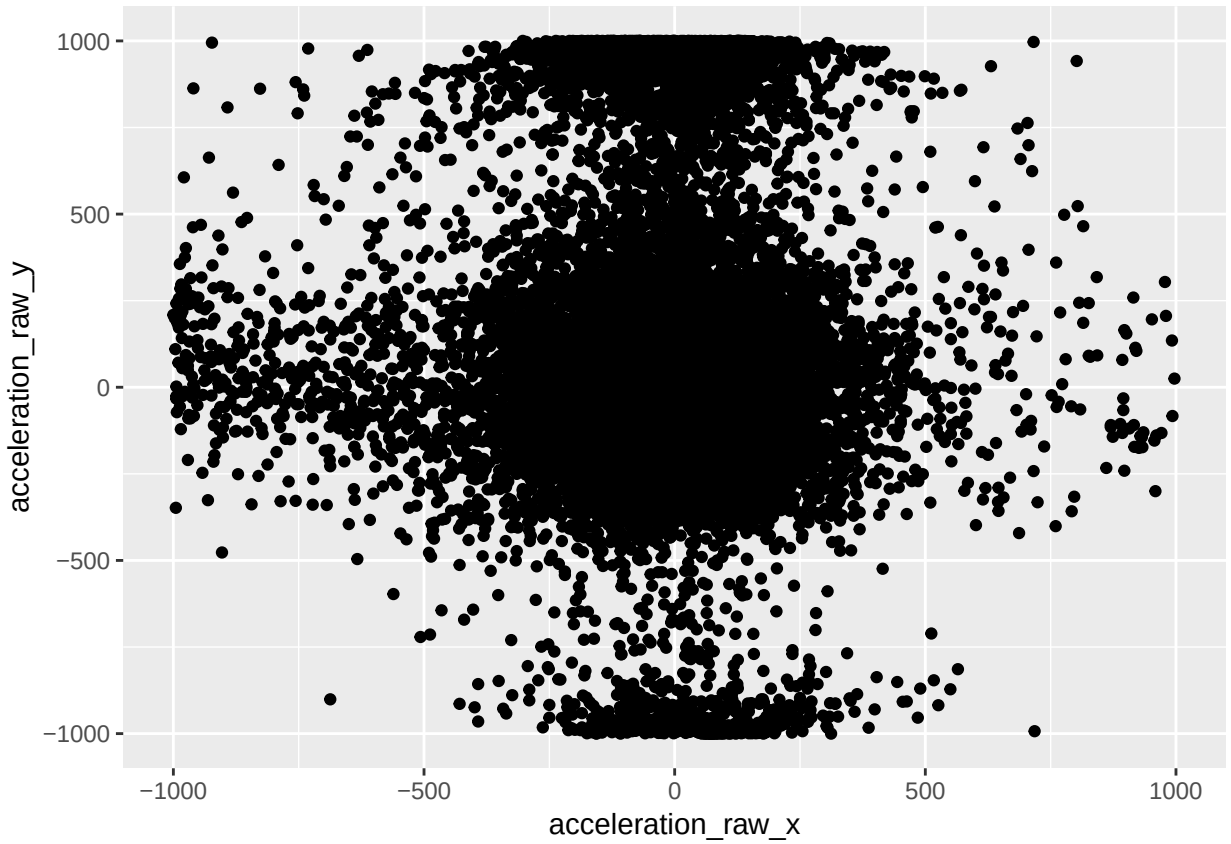
Optional exercise (+) - Scale axes

Scaling the axes with a ggplot is easy. Take a look at the Data Visualization cheatsheet. In the bottom right corner of the cheatsheet, you will find the code to scale the axes.

Zoom in on the acceleration between -1000 and 1000 for both the x-axis and y-axis.

```
ggplot(data_crane, aes(acceleration_raw_x, acceleration_raw_y)) +
  geom_point() +
  scale_x_continuous(limits = c(-1000, 1000)) +
  scale_y_continuous(limits = c(-1000, 1000))
```

```
## Warning: Removed 2436 rows containing missing values (geom_point).
```



Optional exercise (++) - Plot the crane positions on a map

Our dataset does not only contain the speed and angle of our migrating cranes, but their location as well. Using other packages in R, we can plot this location on a map, to get a complete visualization of the cranes' movement.

a) Install the package `maps`

Install the package `maps` and load the library in the cell below.

```
install.packages('maps')
```

```
## Error in contrib.url(repos, "source"): trying to use CRAN without setting a mirror
```

```
library(maps)
```

```
##
## Attaching package: 'maps'
```

```
## The following object is masked from 'package:purrr':
##
##     map
```

b) Plot the crane data on a map.

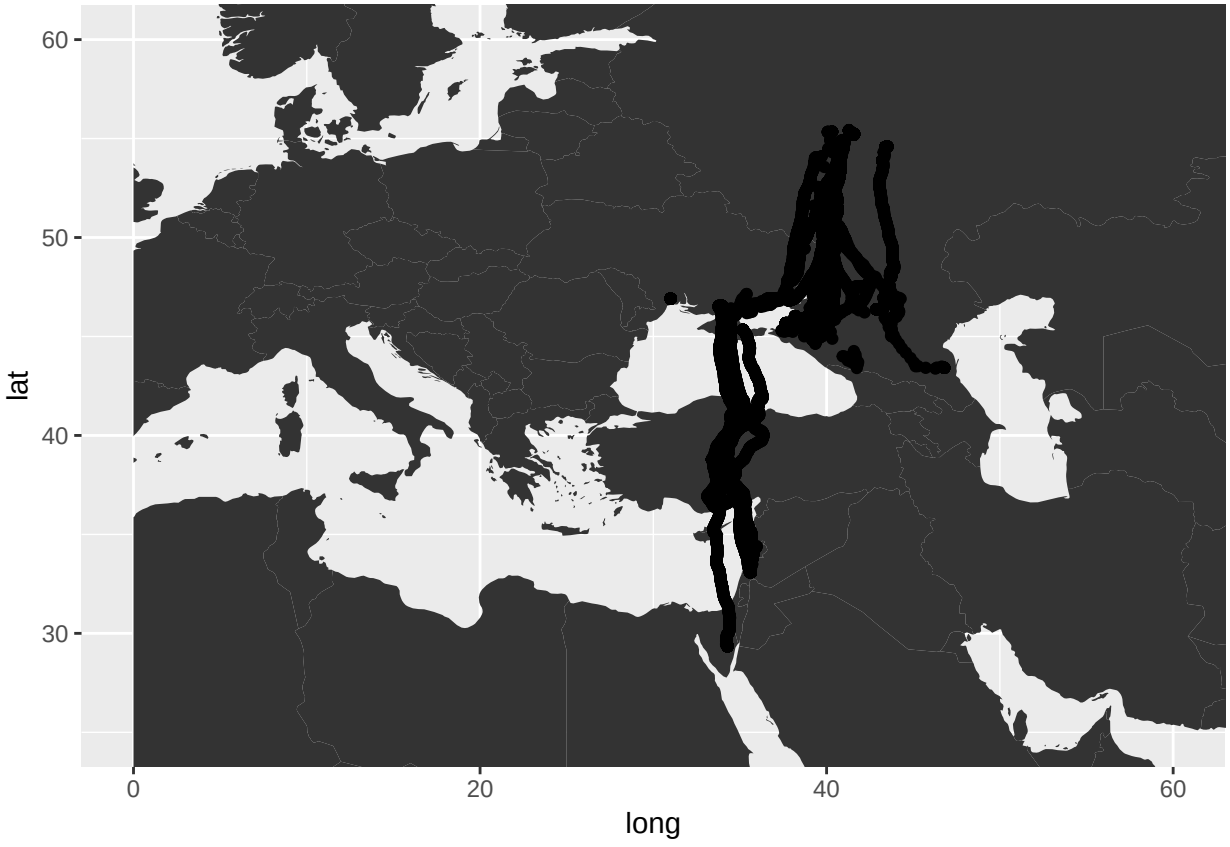
Add a layer with the location of the observations.

Tip: can you use the function `geom_point()` for this layer?

```
world_map_polygon <- map_data("world2")

ggplot(data_crane) +
  geom_map(data = world_map_polygon, map= world_map_polygon, aes(x=long, y = lat, map_id = region)) +
  scale_x_continuous(limits = c(0, 60)) +
  scale_y_continuous(limits = c(25, 60)) +
  geom_point(data = data_crane, aes(x = location_long, y = location_lat))
```

```
## Warning: Ignoring unknown aesthetics: x, y
```

```
## Warning: Removed 200 rows containing missing values (geom_point).
```



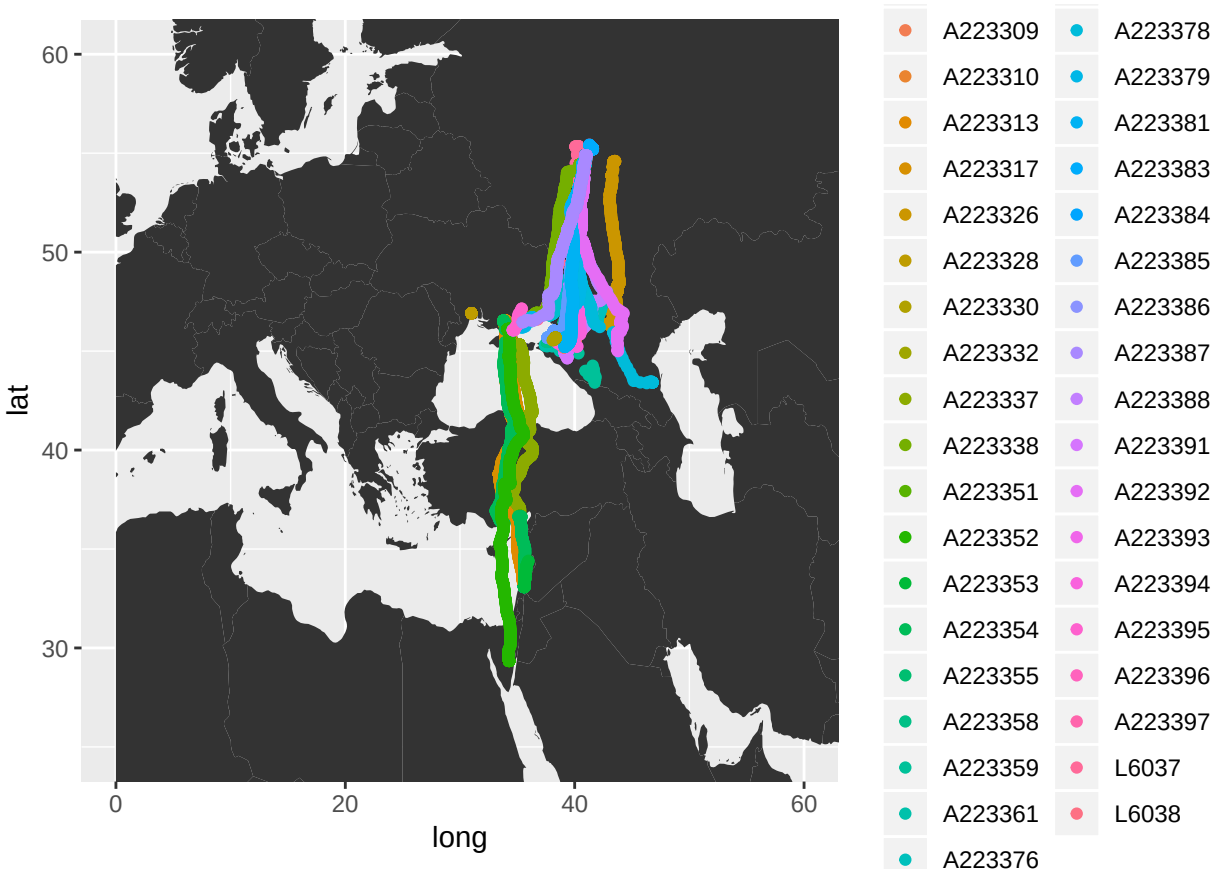
c) Use an individual identifier to colour the different cranes.

The variable `individual_local_identifier` identifies the cranes. Use this variable to color the points.

```
world_map_polygon <- map_data("world2")

ggplot(data_crane) +
  geom_map(data = world_map_polygon, map= world_map_polygon, aes(long, lat, map_id = region)) +
  scale_x_continuous(limits = c(0, 60)) +
  scale_y_continuous(limits = c(25, 60)) +
  geom_point(data = data_crane, aes(location_long, location_lat, colour=individual_local_identifier))
```

```
## Warning: Ignoring unknown aesthetics: x, y
```

```
## Warning: Removed 200 rows containing missing values (geom_point).
```

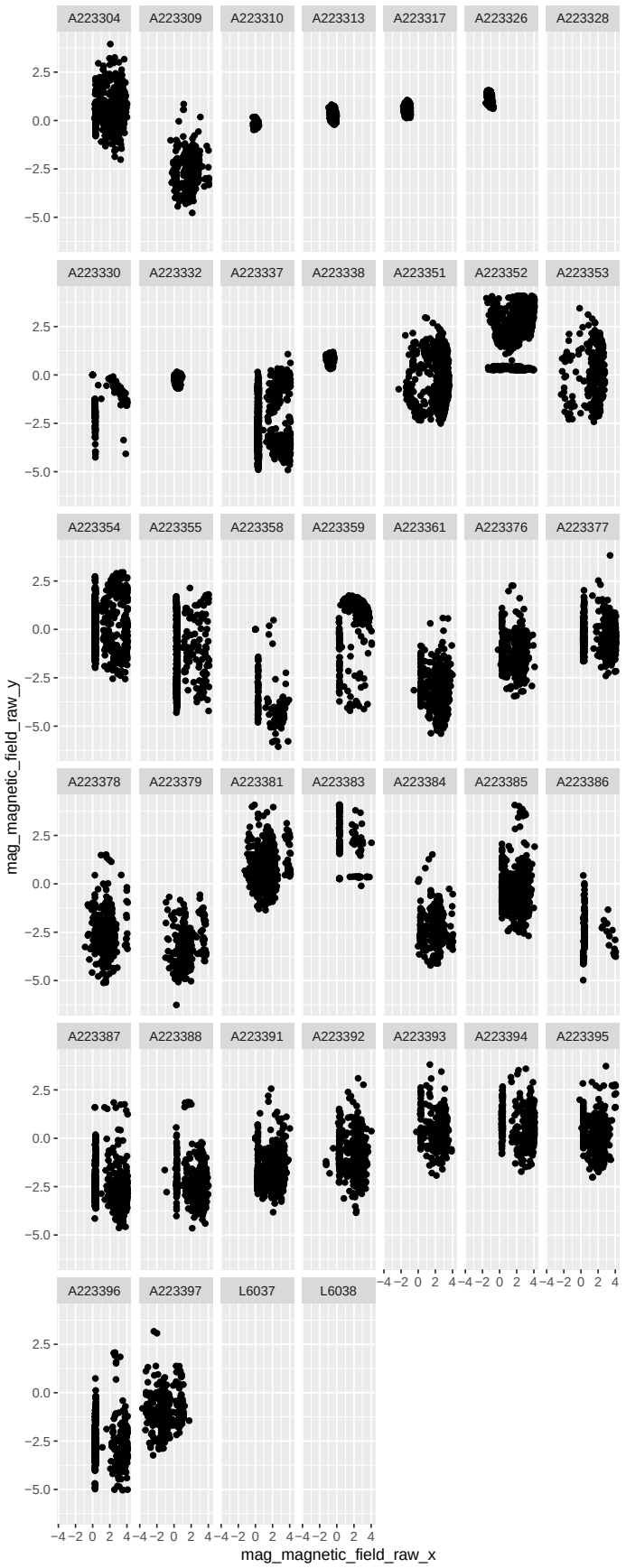


Optional exercise (+++) - Create facets.

Search on the internet for information about facets in ggplot. This unique feature is a must-have for data exploration. Create a graph (whatever kind of graph) and use facets to display all cranes (use the variable `individual_local_identifier`).

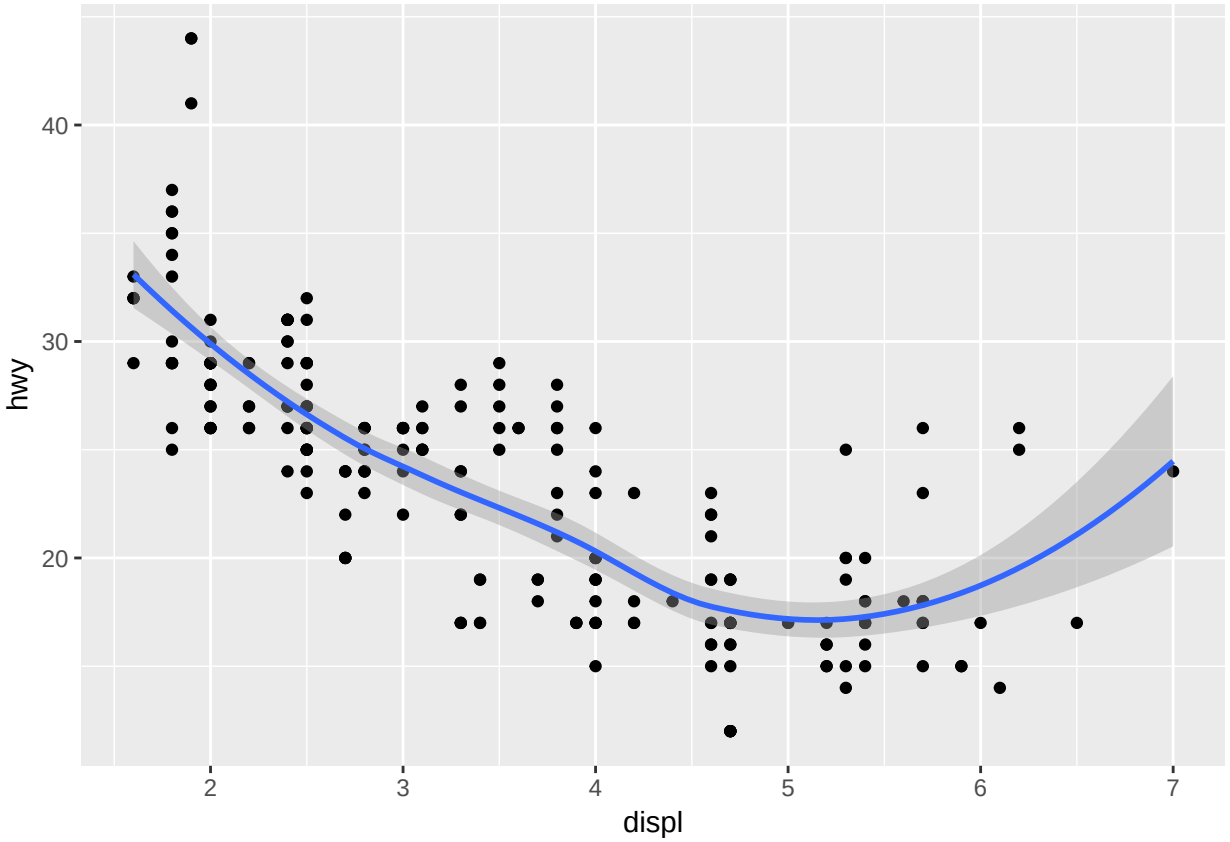
```
ggplot(data_crane, aes(mag_magnetic_field_raw_x, mag_magnetic_field_raw_y)) +
  geom_point() +
  facet_wrap(~individual_local_identifier)
```

```
## Warning: Removed 432 rows containing missing values (geom_point).
```


A223304
A223309
A223310
A223313
A223317
A223326
A223328
A223330
A223332
A223337
A223338
A223351
A223352
A223353
A223354
A223355
A223358
A223359
A223361
A223376
A223377
A223378
A223379
A223381
A223383
A223384
A223385
A223386
A223387
A223388
A223391
A223392
A223393
A223394
A223395
A223396
A223397
L6037
L6038
mag_magnetic_field_raw_y
mag_magnetic_field_raw_x
2.5
0.0
−2.5
−5.0
−4 −2 0 2 4

Recommended reading - Statistical layers for graphs.

You have seen the power of statistical layers in the exercises. For another look at this, consider the following graph:



For more on geometric objects, read Chapter 3.6 in R for Data Science.

3. Data transformation

The crane dataset is clean and well-structured. Nevertheless, you might be interested in doing additional transformations and selections. In this section, we transform and enrich the crane dataset.

Basic exercise I - Subset data

a) Filter data from one individual crane

Make a selection of observations of crane 'L6037' in column `individual_local_identifier`. Make use of the tidyverse (dplyr) function `filter()`. Have a look at the cheat sheet if you are stuck.

```
filter(data_crane, individual_local_identifier=='L6037')
```

```
## # A tibble: 61 x 42
##    event_id visible timestamp           location_long location_lat
##       <dbl> <lgl>   <dttm>                      <dbl>        <dbl>
##  1   3.79e9 TRUE    2017-10-15 00:00:16          40.3         55.3
##  2   3.80e9 TRUE    2017-10-15 00:00:16          40.3         55.3
```

```
##  3   3.79e9 TRUE    2017-10-15 02:00:23          40.3         55.3
##  4   3.80e9 TRUE    2017-10-15 02:00:23          40.3         55.3
##  5   3.79e9 TRUE    2017-10-15 04:00:23          40.3         55.3
##  6   3.80e9 TRUE    2017-10-15 04:00:23          40.3         55.3
##  7   3.79e9 TRUE    2017-10-15 05:00:14          40.1         55.3
##  8   3.80e9 TRUE    2017-10-15 05:00:14          40.1         55.3
##  9   3.80e9 TRUE    2017-10-15 06:00:23          40.1         55.3
## 10   3.80e9 TRUE    2017-10-15 08:00:18          40.1         55.3
## # ... with 51 more rows, and 37 more variables: acceleration_raw_x <dbl>,
## #   acceleration_raw_y <dbl>, acceleration_raw_z <dbl>,
## #   bar_barometric_height <dbl>, battery_charge_percent <dbl>,
## #   battery_charging_current <dbl>, eobs_activity <lgl>,
## #   eobs_activity_samples <lgl>, eobs_battery_voltage <dbl>,
## #   eobs_fix_battery_voltage <dbl>,
## #   eobs_horizontal_accuracy_estimate <dbl>, eobs_key_bin_checksum <dbl>,
## #   eobs_speed_accuracy_estimate <dbl>, eobs_start_timestamp <chr>,
## #   eobs_status <chr>, eobs_temperature <dbl>, eobs_type_of_fix <dbl>,
## #   eobs_used_time_to_get_fix <dbl>, external_temperature <dbl>,
## #   gps_hdop <dbl>, gps_satellite_count <dbl>, gps_time_to_fix <dbl>,
## #   ground_speed <dbl>, heading <dbl>, height_above_ellipsoid <dbl>,
## #   height_above_msl <dbl>, import_marked_outlier <lgl>,
## #   gls_light_level <dbl>, mag_magnetic_field_raw_x <dbl>,
## #   mag_magnetic_field_raw_y <dbl>, mag_magnetic_field_raw_z <dbl>,
## #   orn_transmission_protocol <chr>, tag_voltage <dbl>, sensor_type <chr>,
## #   individual_taxon_canonical_name <chr>,
## #   individual_local_identifier <chr>, study_name <chr>
```

b) Filter data with complete GPS information

Make a selection of all observations of crane 'L6037' where the variable `eobs_status` is not missing. To do this, you can use the function `is.na()`, but you will have to invert its results. Search in your help window for the operator `!`, to get an idea on how to do this.

How many records satisfy this condition?

```
data_crane_filtered <- filter(data_crane, individual_local_identifier=='L6037', !is.na(eobs_status))

# count the number of rows
nrow(data_crane_filtered)
```

```
## [1] 42
```

c) Select specific columns from your data

Use the function `select` to return only columns that have status information about the GPS.

Tip: Note that columns with GPS information start with `eobs_`, and recall the function `starts_with` that you can use inside `select`.

```
select(data_crane, starts_with("eobs_"))
```

```
## # A tibble: 20,873 x 12
##    eobs_activity eobs_activity_s~ eobs_battery_vo~ eobs_fix_batter~
##    <lgl>         <lgl>                      <dbl>            <dbl>
##  1 NA            NA                          3784               NA
##  2 NA            NA                          3818             3801
```

```
##  3 NA            NA                              3784              NA
##  4 NA            NA                              3818            3798
##  5 NA            NA                              3784              NA
##  6 NA            NA                              3814            3796
##  7 NA            NA                              3784              NA
##  8 NA            NA                              3814            3798
##  9 NA            NA                              3814            3796
## 10 NA            NA                              3814            3798
## # ... with 20,863 more rows, and 8 more variables:
## #   eobs_horizontal_accuracy_estimate <dbl>, eobs_key_bin_checksum <dbl>,
## #   eobs_speed_accuracy_estimate <dbl>, eobs_start_timestamp <chr>,
## #   eobs_status <chr>, eobs_temperature <dbl>, eobs_type_of_fix <dbl>,
## #   eobs_used_time_to_get_fix <dbl>
```

d) Combine `filter` and `select` using the pipe operator

Now, combine the results from exercise b) and c) using a pipe operator (`%>%`): make a selection of all observations of crane 'L6037' where the variable `eobs_status` is not missing, and return only columns that start with `eobs_`.

Tip: take a look at the recommended reading for this chapter for more information about the pipe operator.

```
filter(data_crane, individual_local_identifier=='L6037', !is.na(eobs_status)) %>%
  select(starts_with("eobs_"))
```

```
## # A tibble: 42 x 12
##    eobs_activity eobs_activity_s~ eobs_battery_vo~ eobs_fix_batter~
##    <lgl>         <lgl>                      <dbl>            <dbl>
##  1 NA            NA                          3818             3801
##  2 NA            NA                          3818             3798
##  3 NA            NA                          3814             3796
##  4 NA            NA                          3814             3798
##  5 NA            NA                          3814             3796
##  6 NA            NA                          3814             3798
##  7 NA            NA                          3815             3798
##  8 NA            NA                          3815             3801
##  9 NA            NA                          3811             3792
## 10 NA            NA                          3811             3791
## # ... with 32 more rows, and 8 more variables:
## #   eobs_horizontal_accuracy_estimate <dbl>, eobs_key_bin_checksum <dbl>,
## #   eobs_speed_accuracy_estimate <dbl>, eobs_start_timestamp <chr>,
## #   eobs_status <chr>, eobs_temperature <dbl>, eobs_type_of_fix <dbl>,
## #   eobs_used_time_to_get_fix <dbl>
```

Basic exercise II - Compute the magnitude of the magnetic field

The function `mutate()` (from the `dplyr` package) makes it easy to mutate and create new column variables in a data frame or tibble.

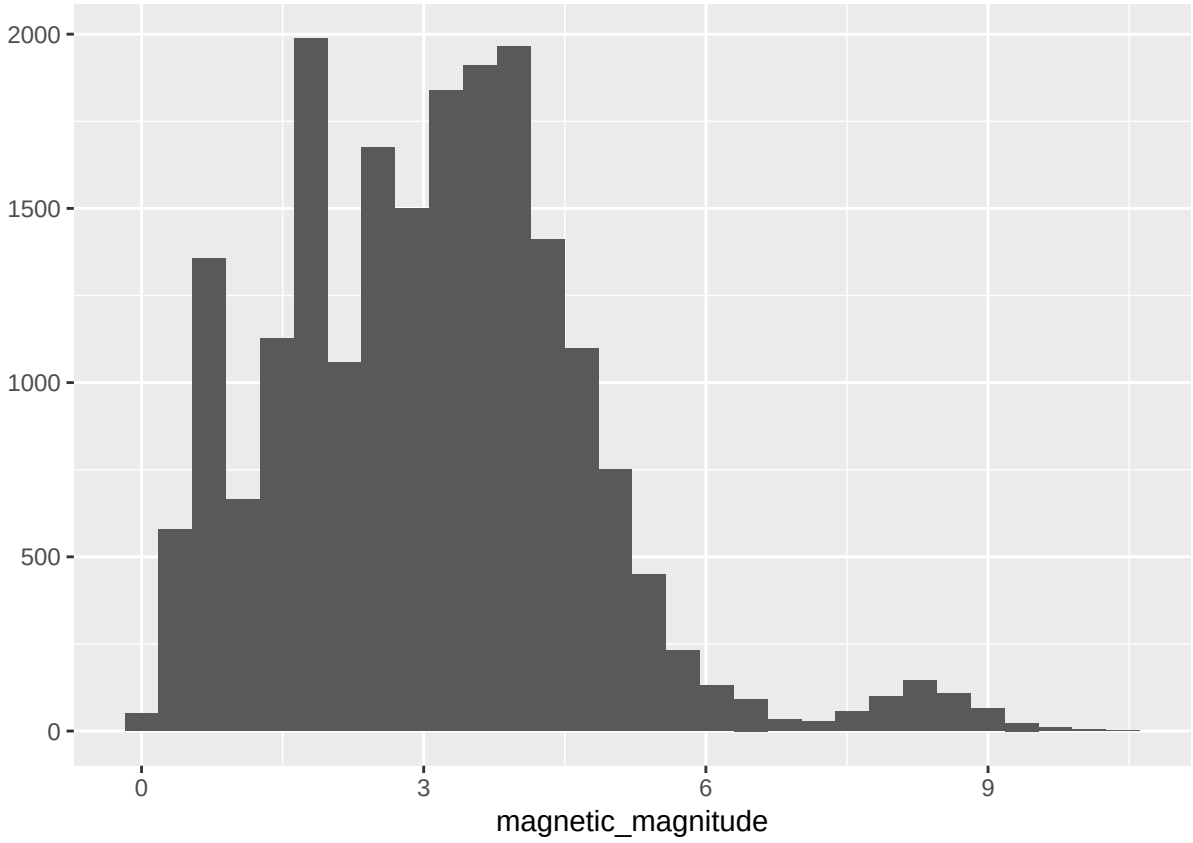
Create a new tibble from `data_crane`, with a new variable called `magnetic_magnitude`, which is the magnitude of the magnetic field. You can calculate the magnetic field by taking the square root of `mag_magnetic_field_raw_x ^ 2 + mag_magnetic_field_raw_y ^ 2 + mag_magnetic_field_raw_z ^ 2`

Try formatting your code so that it is as readable as possible.

```r
data_crane_magnetic <- mutate(
  data_crane,
  magnetic_magnitude = sqrt(mag_magnetic_field_raw_x ^ 2 +
                      mag_magnetic_field_raw_y ^ 2 +
                      mag_magnetic_field_raw_z ^ 2)
)

# take a look at the new variable
qplot(magnetic_magnitude, data=data_crane_magnetic)
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

```
## Warning: Removed 432 rows containing non-finite values (stat_bin).
```



Optional exercise (+) - Summarise results

The function `summarise()` can be used to summarise variables in tibbles. Generate a summary that contains the following information:

- the minimum latitude (the lowest point on the globe)
- the maximum latitude (the highest point on the globe)
- the earliest observation
- the last observation

and, if you want to be challenged (++):

- the mean magnitude of the acceleration (the square root of x^2 + y^2 + z^2)

Add to the code in the chunk below.

Tip: check if you can use the argument ***na.rm*** *in functions such as* ***mean*** *and* ***max****.*

```
summarise(
  data_crane,
  min_latitude = min(location_lat, na.rm=T),
  max_latitude = max(location_lat, na.rm=T),
  first_observation = min(timestamp),
  last_observation = max(timestamp),
  magnitude_acceleration = mean(sqrt(acceleration_raw_x ^ 2 + acceleration_raw_y ^ 2 + acceleration_raw
)
```

```
## # A tibble: 1 x 5
##   min_latitude max_latitude first_observation   last_observation
##          <dbl>        <dbl> <dttm>              <dttm>
## 1         29.3         55.4 2017-10-15 00:00:05 2017-10-17 23:59:33
## # ... with 1 more variable: magnitude_acceleration <dbl>
```

Optional exercise (++) - Join datasets

In this exercise we will append the crane dataset with the additional measures. We will do a full join, combining all data from `data_crane` with all data from `data_crane_additional`. (Note though that both datasets have the same unique identifiers, so both a right join and a left join would achieve the same results.)

Join the two data frames by adding to the code below. What column should you use as join key (the `by=` argument)?

Tip: for more information about joining data, take a look at the cheat sheet element 'Combine Tables', or the recommended reading for this chapter.

```
data_crane_with_measures <- full_join(data_crane, data_crane_additional, by='event_id')
```

Optional exercise (+++) - gather all acceleration data

As mentioned, this dataset is well-structured. Nevertheless, as an exercise, we could gather some columns to make the dataset longer. Can you gather the columns with acceleration data, and generate one column with all acceleration data values, and one column with the headers of the respective measurements (i.e. `acceleration_raw_x`, `acceleration_raw_y`, and `acceleration_raw_z`)?

Consider the example with iris:

```
gather(iris, # dataset
       measurement, # column name for headers
       value, # column name for values
       Sepal.Length, Sepal.Width, Petal.Length, Petal.Width # columns that will be gathered
       )
```

To perform the gather with data_crane, we are first performing a filter to remove the entries with missing acceleration data. Making this combination of tidyverse functions run smooth is the pipe operator.

Add your code to the chunk below, and complete the `gather()` function. Note that due to the pipe operator, you may forgo the name of the dataset in this function.

Tip: follow the structure as in the example with ***iris****, above.*

```r
data_crane_long <- data_crane %>%
  filter(!is.na(acceleration_raw_x) & !is.na(acceleration_raw_y) & !is.na(acceleration_raw_z)) %>%
  gather(acceleration_direction, # column name for headers
         acceleration_speed, # column name for values
         acceleration_raw_x, acceleration_raw_y, acceleration_raw_z # columns that will be gathered
         )

head(data_crane_long)
```

```
## # A tibble: 6 x 41
##   event_id visible timestamp           location_long location_lat
##      <dbl> <lgl>   <dttm>                      <dbl>        <dbl>
## 1   3.80e9 TRUE    2017-10-15 00:05:36          43.4         54.6
## 2   3.80e9 TRUE    2017-10-15 00:20:06          43.4         54.6
## 3   3.80e9 TRUE    2017-10-15 00:34:54          43.4         54.6
## 4   3.80e9 TRUE    2017-10-15 00:50:00          43.4         54.6
## 5   3.80e9 TRUE    2017-10-15 01:04:24          43.4         54.6
## 6   3.80e9 TRUE    2017-10-15 01:19:00          43.4         54.6
## # ... with 36 more variables: bar_barometric_height <dbl>,
## #   battery_charge_percent <dbl>, battery_charging_current <dbl>,
## #   eobs_activity <lgl>, eobs_activity_samples <lgl>,
## #   eobs_battery_voltage <dbl>, eobs_fix_battery_voltage <dbl>,
## #   eobs_horizontal_accuracy_estimate <dbl>, eobs_key_bin_checksum <dbl>,
## #   eobs_speed_accuracy_estimate <dbl>, eobs_start_timestamp <chr>,
## #   eobs_status <chr>, eobs_temperature <dbl>, eobs_type_of_fix <dbl>,
## #   eobs_used_time_to_get_fix <dbl>, external_temperature <dbl>,
## #   gps_hdop <dbl>, gps_satellite_count <dbl>, gps_time_to_fix <dbl>,
## #   ground_speed <dbl>, heading <dbl>, height_above_ellipsoid <dbl>,
## #   height_above_msl <dbl>, import_marked_outlier <lgl>,
## #   gls_light_level <dbl>, mag_magnetic_field_raw_x <dbl>,
## #   mag_magnetic_field_raw_y <dbl>, mag_magnetic_field_raw_z <dbl>,
## #   orn_transmission_protocol <chr>, tag_voltage <dbl>, sensor_type <chr>,
## #   individual_taxon_canonical_name <chr>,
## #   individual_local_identifier <chr>, study_name <chr>,
## #   acceleration_direction <chr>, acceleration_speed <dbl>
```

Recommended reading

The pipe operator `%>%` is an elegant way to tie (tidyverse) functions together. Read Chapter 18 of R for Data Science for more information on this operator.

For more on joins, take a look at chapter 13.4.1-13.4.3 from the same book, to (visually) guide you through the process.

Finally, if the last exercise had you down, R for Data Science devotes a chapter on tidy data, including the functions `gather` and `spread` that is well worth a read.