workshop02(1)

February 3, 2019

```
In [4]: library(plotly)
        library(tidyverse)
        library(RPostgreSQL)

        # Plot size deppening on your screen resolution to 5 x 3
        options(repr.plot.width=5, repr.plot.height=3)
```

```
Loading required package: ggplot2

Attaching package: plotly

The following object is masked from package:ggplot2:

    last_plot

The following object is masked from package:stats:

    filter

The following object is masked from package:graphics:

    layout

 Attaching packages  tidyverse 1.2.1
 tibble  1.4.2      purrr   0.2.5
 tidyr   0.8.1      dplyr   0.7.5
 readr   1.1.1      stringr 1.3.1
 tibble  1.4.2      forcats 0.3.0
 Conflicts  tidyverse_conflicts()
 dplyr::filter() masks plotly::filter(), stats::filter()
 dplyr::lag()    masks stats::lag()
Loading required package: DBI
```

1 Welcome to Workshop 2

1.0.1 Exercise 1

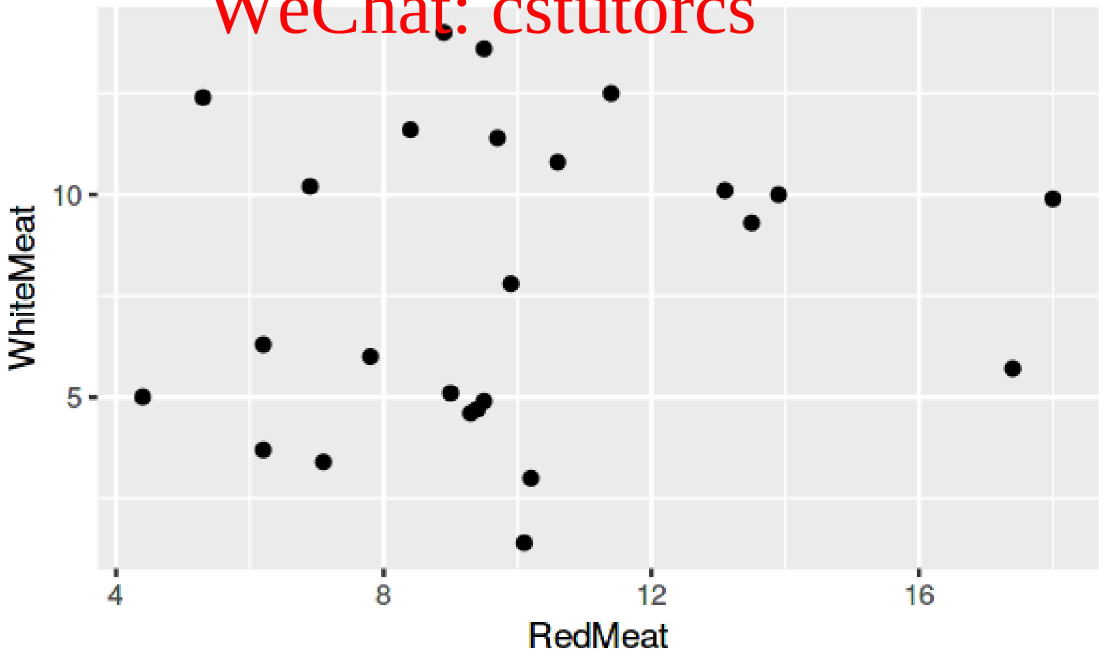
Let's continue from the last class with the protein data-set, load it again into the variable `protein.df`

```
In [4]: # your code here
        protein <- read.table(file = "protein.csv",
                              header = TRUE,
                              sep = ",")
        protein%>%head()
```

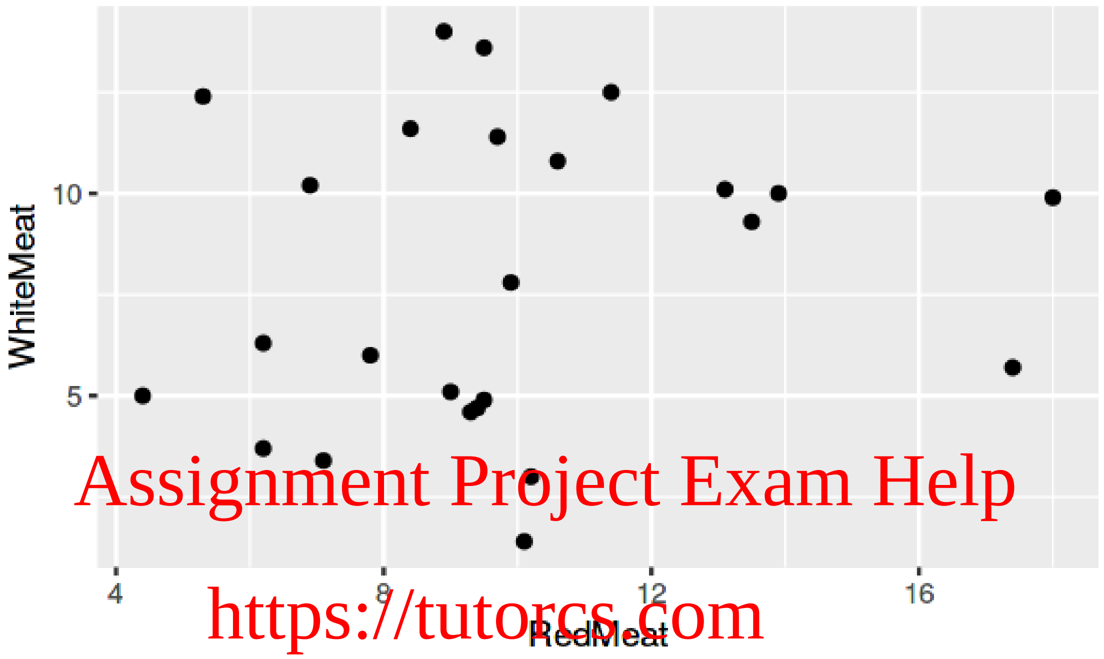
Country	Location	RedMeat	WhiteMeat	Eggs	Milk	Fish	Cereals	Starch	Nuts	Fr.Veg
Albania	E	10.1	1.4	0.5	8.9	0.2	42.3	0.6	5.5	1.7
Austria	W	8.9	14.0	4.3	19.9	2.1	28.0	3.6	1.3	4.3
Belgium	W	13.5	9.3	4.1	17.5	4.5	26.6	5.7	2.1	4.0
Bulgaria	E	7.8	6.0	1.6	8.3	1.2	56.7	1.1	3.7	4.2
Czechoslovakia	E	9.7	11.4	2.8	12.5	2.0	34.3	5.0	1.1	4.0
Denmark	S	10.6	10.8	3.7	25.0	9.9	21.9	4.8	0.7	2.4

Create a scatter plot between red and white meat:

```
In [5]: # your code here
        ggplot(data=protein,
               aes(x=RedMeat,
                 y=WhiteMeat))+
        geom_point()
```

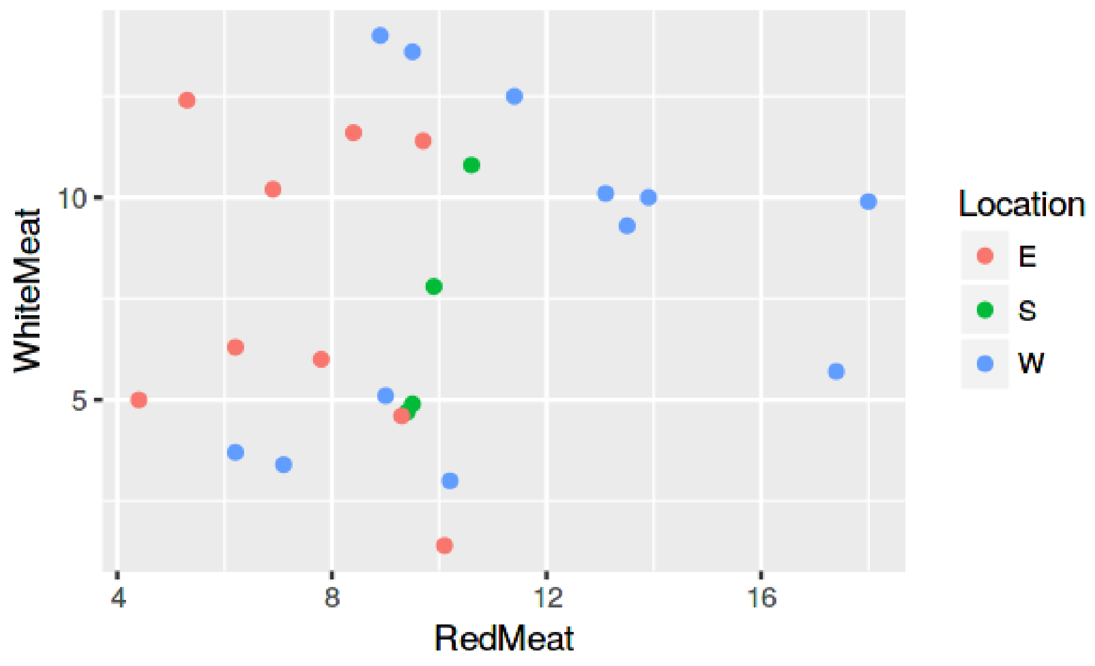


```
In [6]: protein%>%
        ggplot(data=.,
               aes(x=RedMeat,
                   y=WhiteMeat))+
        geom_point()
```

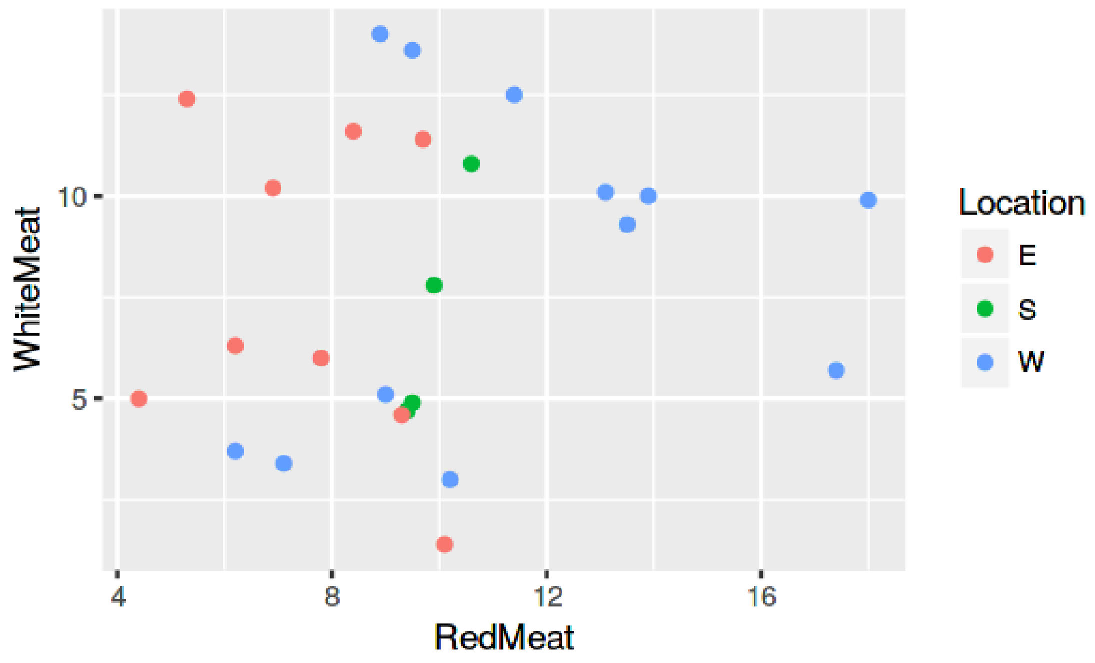


It often helps to colour a graph, let's add colour by location, for this add `colour = Location` as an attribute (same as `x=RedMeat`) for the `aes` function. Assign the ggplot object to the variable p
`p<-ggplot(..`

```
In [7]: # your code here
        p<-ggplot(data=protein,
               aes(x=RedMeat,
                   y=WhiteMeat,
                   colour=Location))+
        geom_point()
        p
```



```
In [8]: p<-ggplot(data=protein,
                  aes(x=RedMeat,
                      y=WhiteMeat))+
        geom_point(aes(colour=Location))
        p
```



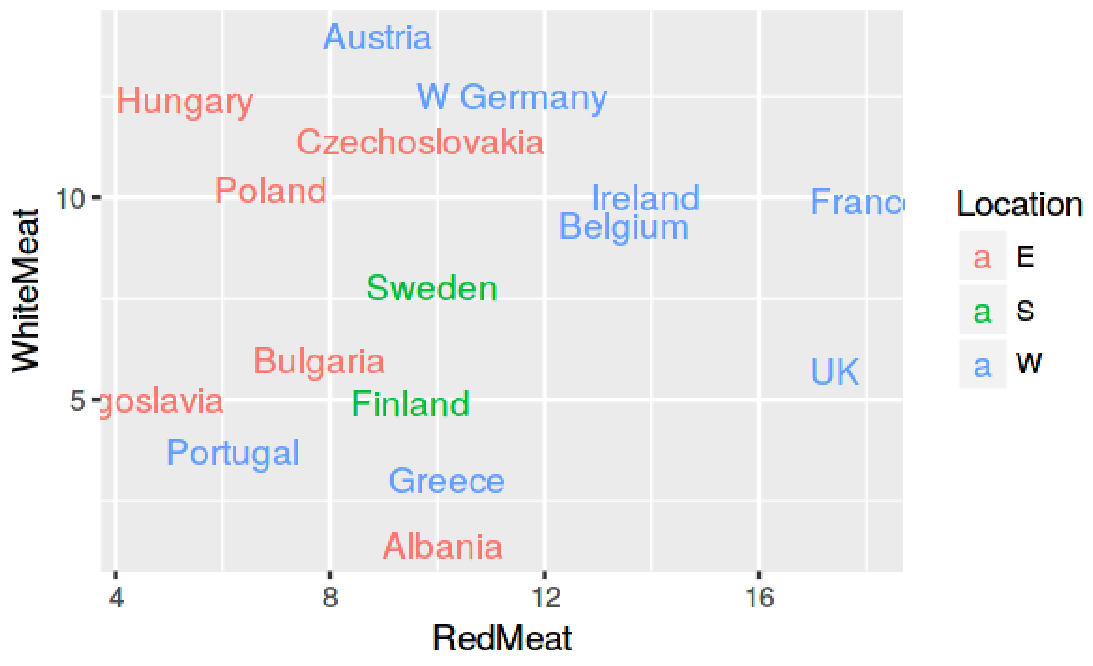
Note: A legend was included automatically. Let's explore the plot in interactive mode:

```
In [7]: interactive <- ggplotly(p)
        as_widget(interactive)
        #embed_notebook(interactive)
```

```
We recommend that you use the dev version of ggplot2 with `ggplotly()`
Install it with: `devtools::install_github('hadley/ggplot2')`


HTML widgets cannot be represented in plain text (need html)
```

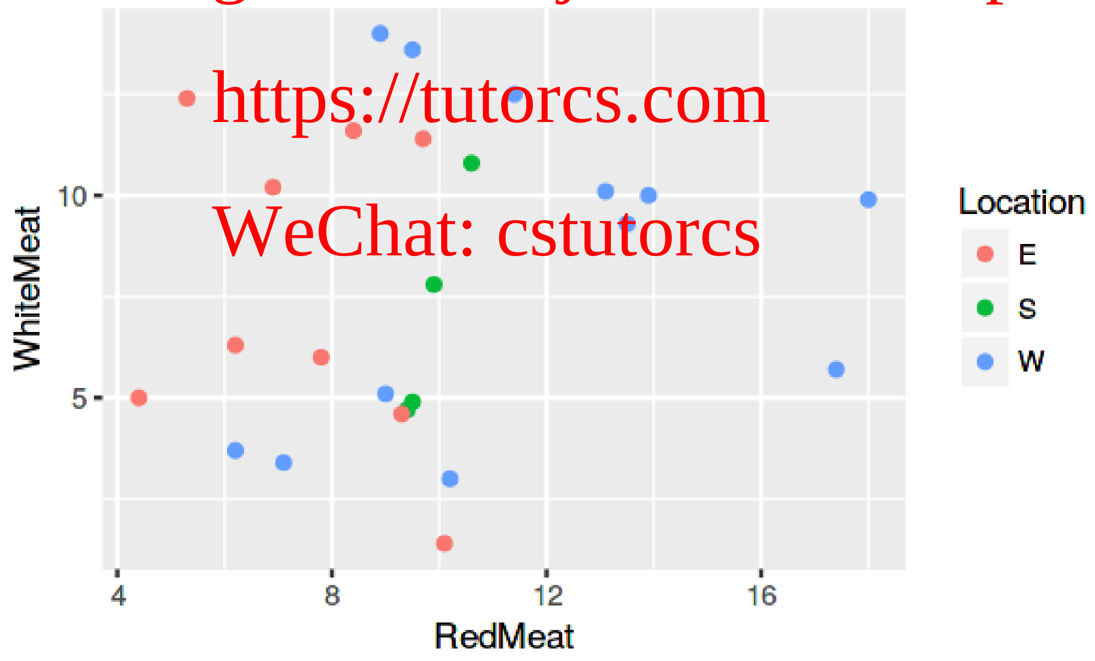
Interactive data exploration helps to get a better feeling of the data and the underlying processes. After a while, there will be way more questions you would like to explore from the data. One question that comes to mind is which countries are displayed on the graph? Let's represent the dots by the name. For this add `label = Country` to the aes mapping as well in addition to your previous ggplot commands append `+ geom_text()`. Experiment the effect of adding the option `check_overlap = TRUE` to `geom_text`.

```
In [8]: # your code here
        ggplot(data=protein,
               aes(x=RedMeat,
                   y=WhiteMeat,
                   colour=Location))+
        geom_text(aes(label=Country),
                  check_overlap=TRUE)
```



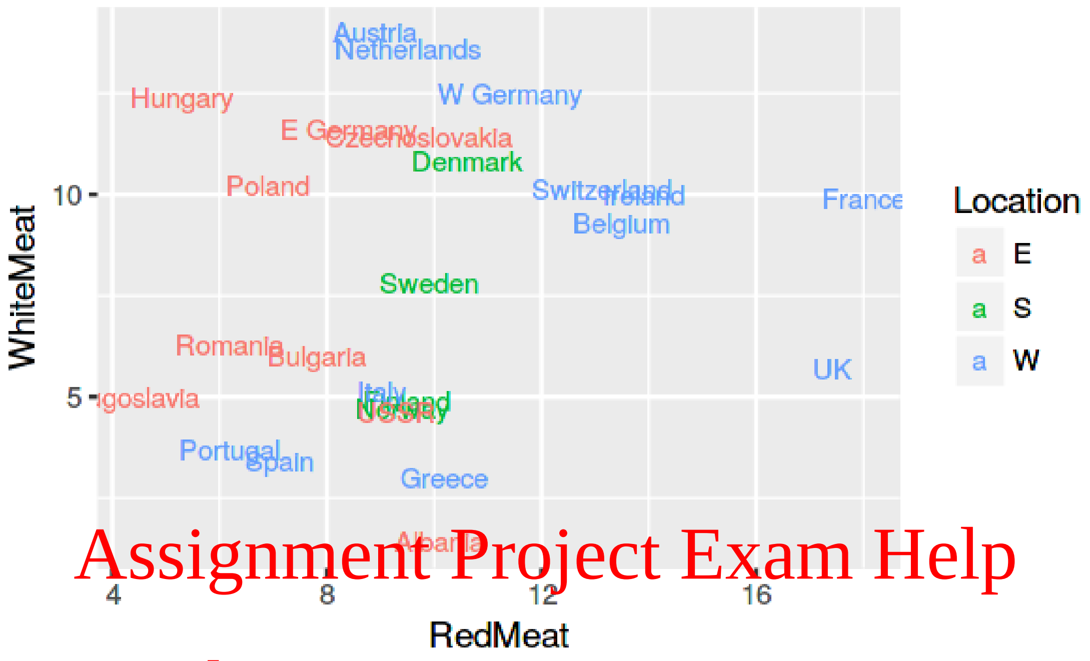
```
In [12]: ggplot(data=protein,
                aes(x=RedMeat,
                   y=WhiteMeat))+
         geom_point(aes(labs=Country,colour=Location))
         geom_text(aes(check_overlap=TRUE))
```

```
Warning message:
Ignoring unknown aesthetics: labs

Warning message:
Ignoring unknown aesthetics: check_overlap

mapping: check_overlap = TRUE
geom_text: parse = FALSE, check_overlap = FALSE, na.rm = FALSE
stat_identity: na.rm = FALSE
position_identity
```



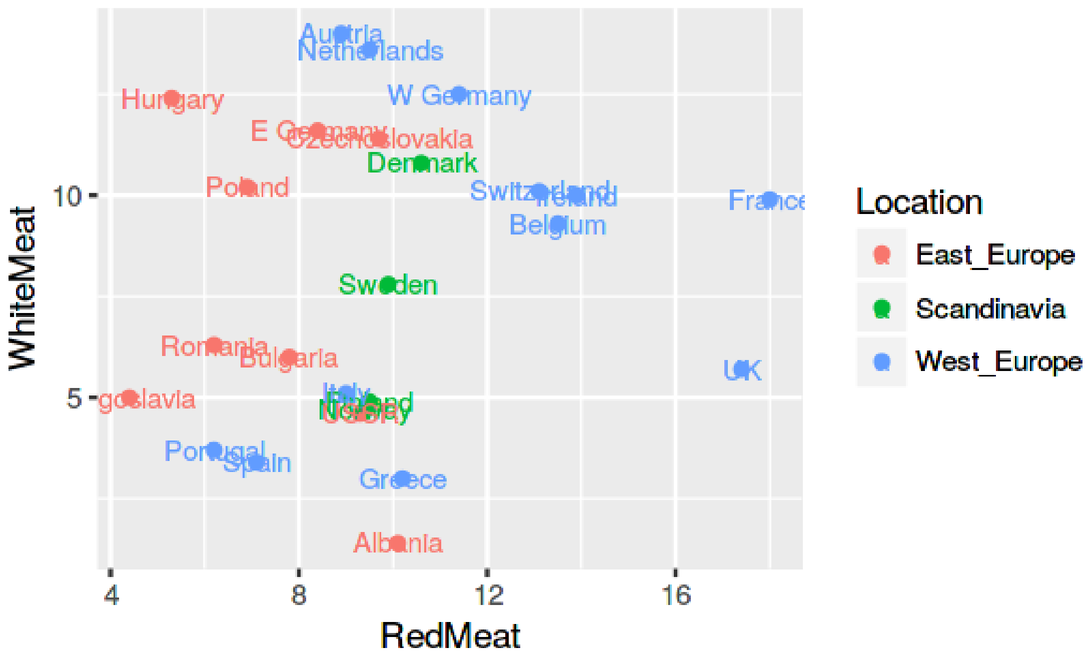
To make the text smaller use the `size = 3` parameter of `geom_text`. Experiment the effect by commenting out the whole line of `geom_point`. What do you see? Discuss with your table

```
In [9]: # your code here
        ggplot(data=protein,
               aes(x=RedMeat,
                  y=WhiteMeat,
                  colour=Location))+
        geom_text(aes(label=Country),size=3)
```



The legend currently shows only E, S, W replace this with the names `East Europe`, `Scandinavia` and `West Europe` without changing the original data frame. Pass the result directly into the plot.

```
In [13]: # your code here
         protein%>%
         mutate(Location=factor(Location,levels =c("E","S","W"),labels=c("East_Europe","Scandina
         ggplot(data=.,
                aes(x=RedMeat,
                   y=WhiteMeat,
                   colour=Location))+
         geom_point()+
         geom_text(aes(label=Country),size=3)
```



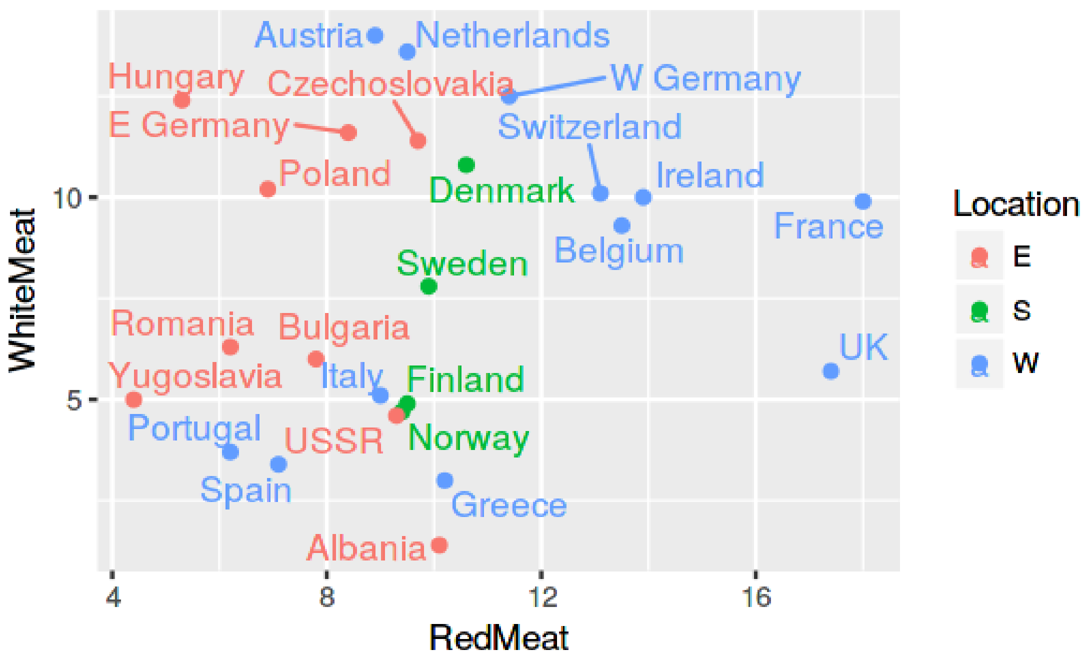
This graph provides a lot more information to the reader, without requiring any extra expertise in interpreting the chart. But it is still not perfect: 1. Yugoslavia and France may be chopped off 2. The whole graph should have a heading 3. The axis labels should be improved 4. Some countries are not readable as they are overlapping

R is a community of individuals around the globe of users like you. They had come across the same problem with the text and developed an extension. One of them is the package `ggrepel`. To load it type `library(ggrepel)`.

```
In [16]: # your code here
         library(ggrepel)
```

Now, instead of writing `geom_text` you can use `geom_text_repel` without the `check_overlap` command.

```
In [17]: # your code here
         ggplot(data=protein,
                aes(x=RedMeat,
                    y=WhiteMeat,
                    colour=Location))+
         geom_point()+
         geom_text_repel(aes(label=Country))
```



1.0.2 Exercise 2: Introduction to SQL

First, create a connection with the database. For this you need to create a csv file with the username and password as mentioned in the lectuer. Take a look at the code below to figure out the data format: (Hint, you should look at the previous CSV files we loaded that were given to you)

```
In [5]: # your code here
        auth <- read.csv('auth.csv', header = TRUE)
        auth%>% head()
```

username	password
student	4XcxqUo6AHPn

```
In [6]: auth <- read.csv('auth.csv', header = TRUE) # never save a password in a notebook direct
        drv <- dbDriver('PostgreSQL')
        # you need to close the connection
        con <- dbConnect(
          drv,
          host = "118.138.234.161",
          dbname = "summer2019",
          user = toString(auth$username),
          password = toString(auth$password)
        )
```

The goal of this exercise is to repeat the data aggregation steps from workshop 1, but this time connect to the database.

Query the whole table:

```
In [7]: query <- "
        SELECT
            country,
            location,*
        FROM
            public.protein;
        "

        protein.df <- dbGetQuery(con, query)
        head(protein.df)
        # dbDisconnect(con) usually you would need to close the connection after querying the da
```

country	location	country	location	redmeat	whitemeat	eggs	milk	fish	cereal
Albania	E	Albania	E	10.1	1.4	0.5	8.9	0.2	42.3
Austria	W	Austria	W	8.9	14.0	4.3	19.9	2.1	28.0
Belgium	W	Belgium	W	13.5	9.3	4.1	17.5	4.5	26.6
Bulgaria	E	Bulgaria	E	7.8	6.0	1.6	8.3	1.2	56.7
Czechoslovakia	E	Czechoslovakia	E	9.7	11.4	2.8	12.5	2.0	34.3
Denmark	S	Denmark	S	10.6	10.8	3.7	25.0	9.9	21.9

Calculate the total meat as before and add it as extra column:

```
In [21]: #meat.df <- protein %>%
         #    mutate(TotMeat = RedMeat + WhiteMeat + Fish) %>%
         #    select(Location, TotMeat)
         #meat.df %>% head()
         # your code here
         query<-'
         SELECT
            location,
            redmeat+whitemeat+fish as totmeat

         FROM
            public.protein;'

         protein.df <- dbGetQuery(con, query)
         head(protein.df)
```

location	totmeat
E	11.7
W	25.0
W	27.3
E	15.0
E	23.1
S	31.3

Count the rows by location:

```
In [22]: # to convert
         #protein %>%
         #    group_by(Location) %>%
         #     summarise(count=n())

         # your code here
         query<-"
         SELECT
            location,
            count(*),
            sum(eggs),
            avg(eggs)

         FROM
            public.protein
         GROUP BY
            location
         ORDER BY
            count;
         "

         protein.df <- dbGetQuery(con, query)
         head(protein.df)
```

location	count	sum	avg
S	4	12.6	3.150000
E	9	19.0	2.111111
W	12	41.8	3.483333

Output the previous result ordered acending and rename "count" to "n":

```
In [23]: # your code here
         query<-"
         SELECT
            location,
            count(*) as n,
            sum(eggs),
            avg(eggs)

         FROM
            public.protein
         GROUP BY
            location
         ORDER BY
            n;
         "

         protein.df <- dbGetQuery(con, query)
         head(protein.df)
```

location	n	sum	avg
S	4	12.6	3.150000
E	9	19.0	2.111111
W	12	41.8	3.483333