BUSA8090 - A2T3

Samuel Gardiner - 44952619

14 May 2020

Note to the marker:

All source code, text, figures and data for this project are contained within the GitHub repo at https://github.com/gardiners/a2t3. To get a copy of all of the scripts and Tableau workbooks for marking, please run the following on your local computer (in a git bash console if you are on Windows or in Terminal on a Mac):

```
git clone https://github.com/gardiners/a2t3 gardiners-a2t3
cd gardiners-a2t3
```

or alternatively, download the repository as a zip file at https://github.com/gardiners/a2t3/archive/master.zip

Question 1

(a)

Here Wickham (2016) is demonstrating the creation of a choropleth map, which generally displays a numeric quantity associated with a geographic region using a colour scale. In this case he wants to plot population data from the dataframe `mi_census` using vector geometries that are in the dataframe `mi_counties`. The dataframes need to be combined so that the plot can present both county borders and county population data.

To combine the two dataframes Wickham uses `left_join` from the `dplyr` package. This is equivalent to a SQL left join. The "left-hand" table (which will retain all of its rows) is `mi_census`; the "right-hand" table is `mi_counties`, which will retain only those rows which have a matching row in the left table. The join key is called `county` in `mi_census` (ie, the county name), and `id` in `mi_counties` (which is also the county name). The join is described very succinctly in `help(left_join)`:

```
left_join()
  return all rows from x, and all columns from x and y. Rows in x with no match in
  y will have NA values in the new columns. If there are multiple matches between x
  and y, all combinations of the matches are returned.
```

As the geographic data contains many matching rows for each county in `mi_census` (each row being a point which defines part of the county boundary polygon), the dataframe returned by `left_join` contains duplicated population data. This is what Wickham means when he says "This is not particularly space efficient," and what `help` means where it says "`all combinations of the matches are returned`". Each row in the joined dataframe is now a point in space, with associated metadata such as the county name and its population. In this format, the joined dataframe is now ready for plotting with `ggplot` and especially `geom_polygon`.

(b)

The operation `full_join` performs a similar role in the tutorial as in Wickham's example above: to combine two datasets so that spatial (geographic) data and tabular (GDP per capita) data can both be presented on the same plot. The difference is that this time, using `full_join` returns all rows from `dat_map` (the spatial dataframe) and all rows from `wdi` (the GDP dataframe). Both of the dataframes being joined have a column `ccode`, their country code - this will be the join key. From `help(full_join)`:

```
full_join()
  return all rows and all columns from both x and y. Where there are not matching
  values, returns NA for the one missing.
```

Therefore any rows from either dataframe which do not match on `ccode` will be retained in the joined dataframe, but will be missing data in the newly added columns. For example, if a country has geographic boundary data in `dat_map`, but its country code `ccode` does match with a `ccode` in the `wdi` dataframe, its geographic data will be retained in the joined dataset (`merged`), but it will have missing data `NA` for any columns which were added from `wdi`. This is useful in this context, as it still allows us to plot the geometry for the country, even if we are missing its GDP data.

It is worth noting that the same outcome (preservation of all country geometries, even with missing GDP data) could have been effected using a left join, as long as the geometry data frame was the left table in the join.

Question 2

(a)

Load the required packages: `tidyverse` (for `ggplot`, `dplyr` and others); `maps` for the world map geometry data; `WDI` to access the World Bank World Development Indicators (WDI) datasets; and `countrycode` to assist in matching country names to their ISO3C codes.

You will need to install the packages if you don't already have them. The following snippet will install the packages if you are missing them:

```r
# Packages we need
packages <- c("tidyverse", "WDI", "maps", "countrycode")
# List of installed packages
installed <- installed.packages()[,1]
# Install the packages we need which aren't in the list of installed packages:
install.packages(packages[!packages %in% installed])
```

Load the packages. We don't need to explicitly load `maps`, as the `map_data` function from `ggplot` only requires that it be installed, not loaded.

```r
library(tidyverse)
library(WDI)
library(countrycode)
```

Search the WDI catalogue for an appropriate dataset:

```r
WDIsearch("(CO2).*(PC)", field = "indicator")
```

```
##                                   indicator
##                              "EN.ATM.CO2E.PC"
##                                        name
## "CO2 emissions (metric tons per capita)"
```

Load the emissions data. Limit the query to the year 2014, since this is the most recent year in the WDI dataset which contains values for the `EN.ATM.CO2E.PC` indicator of interest.

```
emissions <- WDI(indicator = "EN.ATM.CO2E.PC", start = 2014, end = 2014,
                 extra = TRUE)
glimpse(emissions)
```

```
## Rows: 264
## Columns: 11
## $ iso2c          <chr> "1A", "1W", "4E", "7E", "8S", "AD", "AE", "AF", "AG"...
## $ country        <chr> "Arab World", "World", "East Asia & Pacific (excludi...
## $ EN.ATM.CO2E.PC <dbl> 4.8869875, 4.9807069, 5.7761284, 7.4210733, 1.456572...
## $ year           <int> 2014, 2014, 2014, 2014, 2014, 2014, 2014, 2014, 2014...
## $ iso3c          <fct> ARB, WLD, EAP, ECA, SAS, AND, ARE, AFG, ATG, ALB, AR...
## $ region         <fct> Aggregates, Aggregates, Aggregates, Aggregates, Aggr...
## $ capital        <fct> , , , , , Andorra la Vella, Abu Dhabi, Kabul, Saint ...
## $ longitude      <fct> , , , , , 1.5218, 54.3705, 69.1761, -61.8456, 19.817...
## $ latitude       <fct> , , , , , 42.5075, 24.4764, 34.5228, 17.1175, 41.331...
## $ income         <fct> Aggregates, Aggregates, Aggregates, Aggregates, Aggr...
## $ lending        <fct> Aggregates, Aggregates, Aggregates, Aggregates, Aggr...
```

Load the geospatial data into a dataframe.

```
world <- map_data("world")
```

To combine the spatial data (geometries) to the emissions data, we need to join on a common field. Although both datasets contain the country name (as a column named `region` in the spatial data and `country` in the WDI emissions data), these do not perfectly map to each other. Fortunately the `countrycode` package has a function `countrycode` which maps country names to country codes. In our case, since the WDI `emissions` dataframe already includes the three-character ISO country code (as `iso3c`), we will add ISO3C codes to our spatial dataset, instead. Note that we specify the destination code type as "wb", which is the World Bank's variant of ISO3, to ensure perfect matching when we perform the join. We use `dplyr::mutate` to add the ISO3C code as a new column to our spatial dataset.

```
world_cc <- world %>%
  mutate(iso3c = countrycode(region, origin = "country.name", destination = "wb"))
glimpse(world_cc)
```

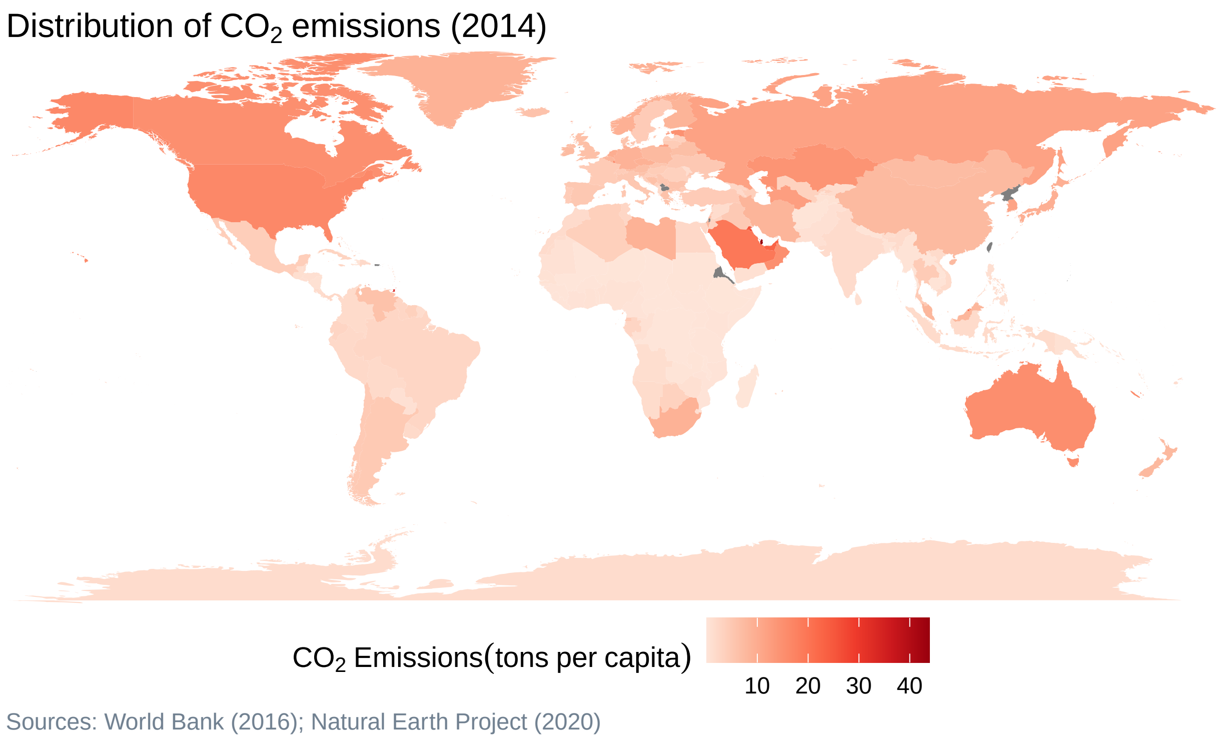
```
## Rows: 99,338
## Columns: 7
## $ long      <dbl> -69.89912, -69.89571, -69.94219, -70.00415, -70.06612, -7...
## $ lat       <dbl> 12.45200, 12.42300, 12.43853, 12.50049, 12.54697, 12.5970...
## $ group     <dbl> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, ...
## $ order     <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 13, 14, 15, 16, 17, 18...
## $ region    <chr> "Aruba", "Aruba", "Aruba", "Aruba", "Aruba", "Aruba", "Ar...
## $ subregion <chr> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
## $ iso3c     <chr> "ABW", "ABW", "ABW", "ABW", "ABW", "ABW", "ABW", "ABW", "...
```

Now that we have a common attribute to act as a joining key, we are able to join the datasets. Here we use a left join to ensure that we keep all of the country geometries, even where there is missing emissions data (for North Korea, Eritrea and Kosovo, for example).

```r
world_emissions <- world_cc %>%
  left_join(emissions, by = "iso3c")
```

We are ready to plot our choropleth (Figure 1):

```r
ggplot(world_emissions, aes(long, lat, group = group, fill = EN.ATM.CO2E.PC)) +
  geom_polygon() +
  theme_void() +
  scale_fill_distiller(palette = "Reds", direction = 1,
                       name = expression(CO[2] ~ Emissions (tons ~ per ~ capita))) +
  theme(legend.position = "bottom",
        plot.caption = element_text(hjust = 0, colour = "slategrey")) +
  scale_x_continuous(expand = c(0, 0)) +
  scale_y_continuous(expand = c(0, 0)) +
  coord_quickmap() +
  labs(title = expression("Distribution of" ~ CO[2] ~ "emissions (2014)"),
       caption = "Sources: World Bank (2016); Natural Earth Project (2020)")
```



Distribution of CO_2 emissions (2014)
CO_2 Emissions(tons per capita)
10
20
30
40
Sources: World Bank (2016); Natural Earth Project (2020)

Figure 1: *Distribution of CO_2 emissions by country in 2014.*

Explaining the `ggplot` call line by line:

1. Call `ggplot`. Map `long` and `lat` to the x- and y-axes, respectively. Map the `emissions` column to the *fill* aesthetic, which will determine country colour on our choropleth. Map the `group` column,

which identifies all of the spatial points belonging to a contiguous geographic polygon (a country or an island, for example), to the *group* aesthetic - this will ensure that each polygon we plot will map to a single spatial polygon.
2. Use `geom_polygon` to draw a polygon for each `group`. It has inherited the x, y, *group* and *fill* aesthetics from the call to ggplot, so we do not need to define them again.
3. Set `theme_void` to remove most of the default `ggplot` decorations (grid lines, grey background etc) which are not necessary on a map.
4. Overwrite the *fill* scale with a ColorBrewer palette. In this case I have chosen the default "Reds" palette, and set direction `1` so that as CO_2 emissions increase, so does the intensity of red for that country.
5. Name the *fill* scale using a `plotmath` expression so that we can get a subscript on the "2" in CO_2.
6. Move the legend to the bottom of the plot so that there is more room for the map itself.
7. Move the citation to the left of the plot and make it light grey so it does not distract.
8. Remove the horizontal padding from the x-axis so that there is more room for the map.
9. Likewise, remove the vertical padding from the y-axis.
10. Use `coord_quickmap` to compute a geographic projection (from a globe to a rectangle) for the spatial data. This improves the aesthetic quality of the map by ensuring that country shapes and sizes are approximately preserved, regardless of the aspect ratio of the plotted image.
11. Set the plot title (again as a `plotmath` expression to get a subscript on "2").
12. Set the plot data citation.

The `R` script `q2a.R` which obtains the data, performs the join and generates the plot is available to download at https://github.com/gardiners/a2t3/raw/master/q2a.R. For the best experience, run the script line-by-line in RStudio.

(b)

We need the dataset `Bali` from the package `UserNetR` (Luke 2019). Since `UserNetR` is not available on CRAN, but is available via the package author's GitHub repository, we must install it via `devtools` or `remotes` (one of which must be also installed). I have elected here to use `remotes` as it has fewer dependencies. We also need `tidygraph` and `ggraph` for graph wrangling and plotting respectively.

```r
# Packages we need
packages <- c("tidyverse", "remotes", "tidygraph", "ggraph")
# List of installed packages
installed <- installed.packages()[,1]
# Install the packages we need which aren't in the list of installed packages:
install.packages(packages[!packages %in% installed])
```

Install `UserNetR` from its GitHub repo at https://github.com/DougLuke/UserNetR. Here I have also specified a particular git commit on `DougLuke/UserNetR`, for reproducibility (ie this script will always pull the same version of `UserNetR`).

```r
remotes::install_github("https://github.com/DougLuke/UserNetR", ref = "67972b7")
```

Load the required packages:

```r
library(tidyverse)
library(tidygraph)
library(ggraph)
```

We don't need to load `UserNetR` as we won't be calling any of its functions (in fact, the package contains no functions, only data). Instead we make a call to `data` to load the `Bali` dataset into our environment.

```r
data(Bali, package = "UserNetR")
class(Bali)
```

```
## [1] "network"
```

We plot the Bali network as an undirected graph using `ggraph` (Pedersen 2020), a `ggplot` interface to `tidygraph` structures. See Figure 2.

```r
Bali %>%
  as_tbl_graph() %>%
  ggraph() +
  geom_edge_link() +
  geom_node_label(aes(label = name)) +
  theme_graph() +
  scale_x_continuous(expand = c(0.1,0.1))
```

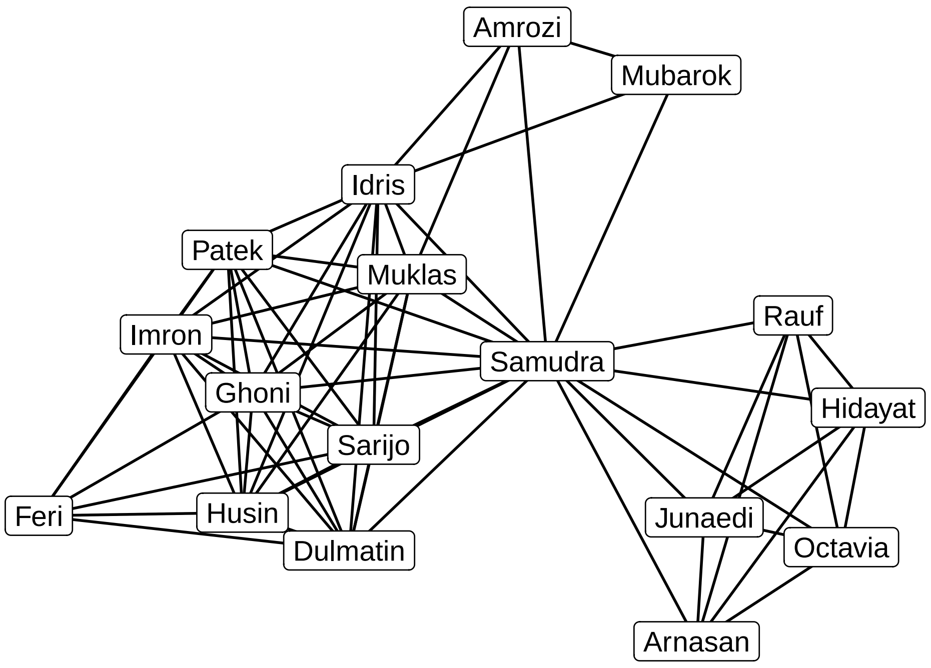


Figure 2: *Social network of the Bali bombers. Nodes are named individuals; edges represent known relationships.*

Line by line:

1. Pipe the `Bali` network object to the next function in our call.
2. Use `tidygraph::as_tbl_graph` to convert from a `network` object (as `UserNetR::Bali` is provided) into the `tbl_graph` data structure used by the `tidygraph` ecosystem.
3. Call to `ggraph`. This is equivalent to a call to `ggplot` on a tibble or dataframe, but for tidygraph objects. We do not need to explicitly map spatial aesthetics to edges or nodes.

4. Add a geometry layer to plot the edges between the bombers. `geom_edge_link` plots straight lines between nodes.
5. Add a geometry layer to plot the nodes. We have chosen to plot each node as the name of the bomber represented by that node. We need to explicitly map the `name` column to the *label* aesthetic.
6. Overwrite the theme with `theme_graph`, which removes most of the default `ggplot` decoration (axis titles and gridlines, which are not required for a graph).
7. Adjust the x-axis horizontal padding, otherwise part of Hidayat's name is clipped from the plot.

The `R` script `q2b.R` which obtains the data and generates the plot is available to download at https://github.com/gardiners/a2t3/raw/master/q2b.R. For the best experience, run the script line-by-line in RStudio.

(c)

We repeat the plot above, but use the bombers' roles instead of their names. The mapping from the short codes provided in the dataset column `role` to the role description can be found in `help(Bali, "UserNetR")`. The output is presented at Figure 3.

```
1  Bali %>%
2    as_tbl_graph() %>%
3    activate(nodes) %>%
4    mutate(role = factor(role, levels = c("OA", "CT", "BM", "TL", "SB")),
5           role_desc = factor(role, labels = c("Operational assistant", "Command team",
6                                               "Bomb maker", "Team Lima",
7                                               "Suicide bomber"))) %>%
8    ggraph() +
9    geom_edge_link() +
10   geom_node_label(aes(label = role, fill = role_desc)) +
11   theme_void() +
12   guides(fill = guide_legend(override.aes = list(label = c("OA", "CT", "BM", "TL", "SB")),
13                              title = "Role"))
```

Line by line:

1. Pipe the `Bali` network into the next function.
2. Convert `Bali` from a `network` to a `tbl_graph`, so that we can use `tidygraph` functions (like `activate`) and `dplyr` verbs (like `mutate`).
3. Select the `nodes` tibble for further operations. A `tidygraph` is a list of tibbles; one for edges (and their attributes or metadata) and another for nodes (and their attributes). The `activate` function allows us to act on one of the tibbles at a time.
4. Use `dplyr::mutate` to convert role to a factor (categorical variable). Explicitly define the order of its levels, so that the order in the legend matches their approximate order on the graph.
5. Use `dplyr::mutate` to create a new variable, `role_desc` which contains a more verbose description of the roles. The order of these levels matches `role`.
6. (5 continues. . .)
7. (5 continues. . .)
8. Call `ggraph` to begin plotting.
9. Add a geometry to plot the edges. As above, `geom_edge_link` plots straight lines between nodes.
10. Add a geometry to plot the nodes. This time we map the `role` factor to the *label* aesthetic, rather than using the bombers' names. We also map our descriptive column `role_desc` to the *fill* aesthetic, so that each role gets a distinct colour on the graph, and to make the legend more interpretable.
11. Use `theme_void` to remove gridlines, axis markers, ticks etc.

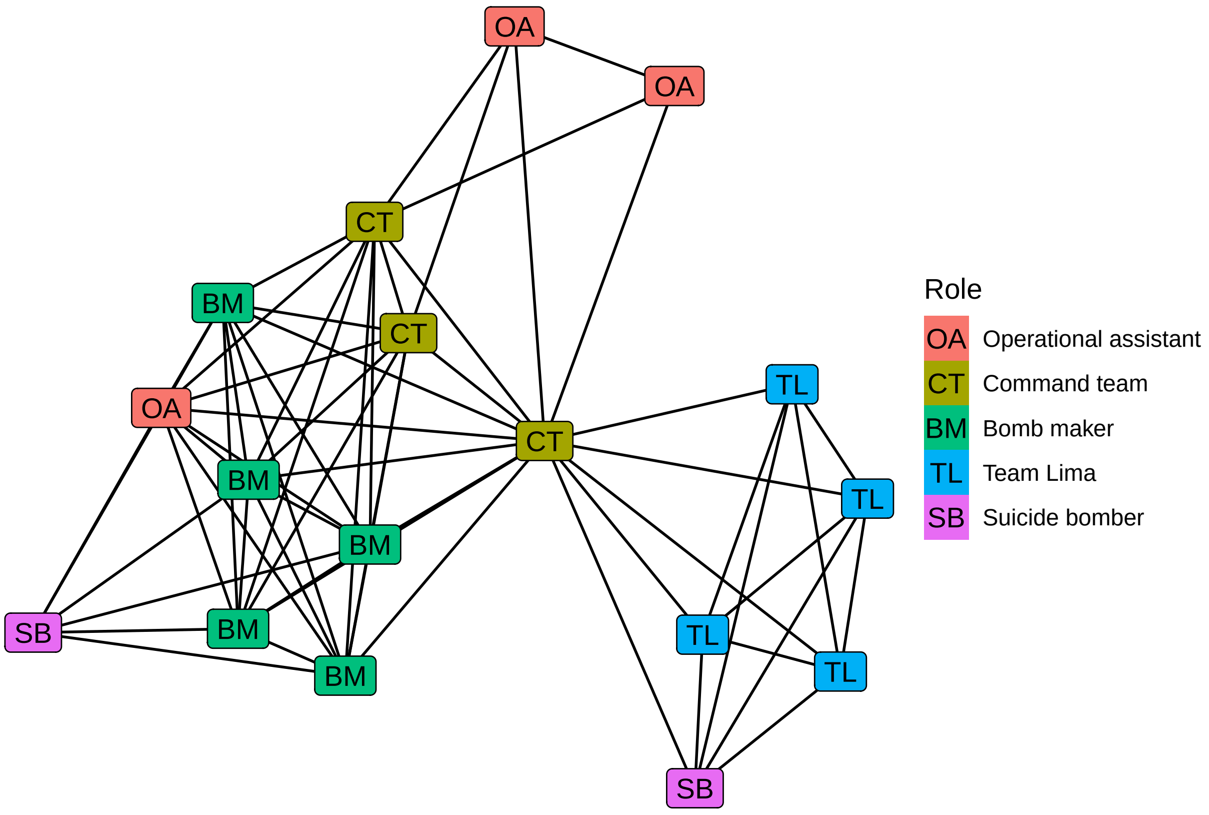


Figure 3: *Social network of the Bali bombers. Nodes are individuals, labelled and coloured by their role in the attack. Edges represent known relationships.*

12. Overwrite the legend's *label* aesthetic so that the symbols on the legend have the short code for each role, as they are presented on the plot.
13. Set the legend's title to "Role" rather than the name of the aesthetic is is a guide for, "fill".

The `R` script `q2c.R` which obtains the data and generates the plot is available to download at https://github.com/gardiners/a2t3/raw/master/q2c.R. For the best experience, run the script line-by-line in RStudio.

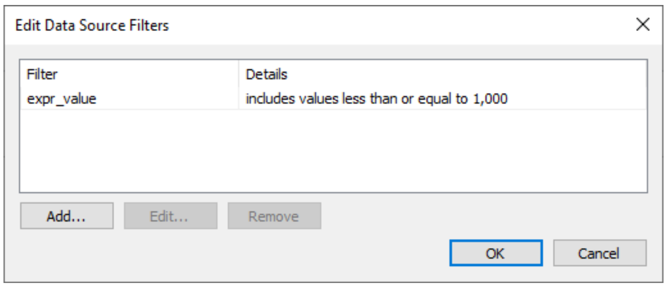
Question 3

We continue to work with our files `expression.tsv` and `annotation.tsv` from the previous assignment.

(a)

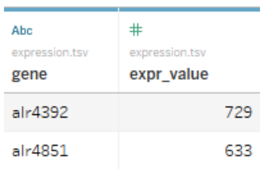
My approach is to connect to the `expression.tsv` file and add a data connection filter. The process is as follows:

- In Tableau, from the "Connect" panel and within the "To a file" subpanel, select "Text file".
- Navigate to and select `expression.tsv`.
- With the `expression` connection selected, navigate to the "Data" main menu, and choose "Edit Data Source Filters. . . "
- In the resulting "Edit Data Source Filters" dialog box, click the "Add" button.
- In the resulting "Add Filter" dialog box, select the `expr_value` field from the list and click "OK".
- In the resulting "Filter [expr_value]" dialog box, select the "At most" tab and set the value to 1000. Click "OK" to close the "Filter [expr_value]" dialog, and then click "OK" again to close the "Edit Data Source Filters" dialog.


Edit Data Source Filters

Filter	Details
expr_value	includes values less than or equal to 1,000

(a) *Filter to retain only rows with `expr_value` less than 1000.*



Abc expression.tsv gene	# expression.tsv expr_value
alr4392	729
alr4851	633

(b) *Result of applying the filter to the data connection. Only two genes are retained.*

Figure 4: *Filter settings and results for Question 3 (a).*

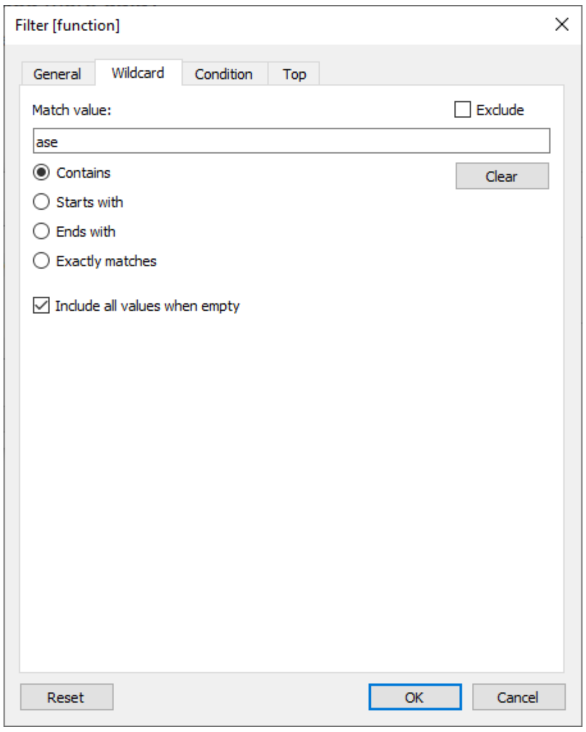
This results in a data connection called `expression` which contains only the rows with genes which have expression under 1000. The resulting data table can be inspected directly by remaining on the "Data source" panel, or can be used to create visualisations in a worksheet.

A packaged Tableau workbook `q3a.twbx` which contains the data connection and the applied filter is available to download at https://github.com/gardiners/a2t3/raw/master/q3a.twbx. Be sure to switch to the "Data Source" tab to inspect the filter settings.

(b)

We can use a similar approach to filter the annotation table.

- Connect to the text file `annotation.tsv`.
- Add a data source filter to the `function` field.
- Select the “Wildcard” tab.
- In the “Match value” field, add the text “ase” (without the quotes).
- Click “OK” to close the “Filter [function]” dialog, and then click “OK” again to close the “Edit Data Source Filters” dialog.



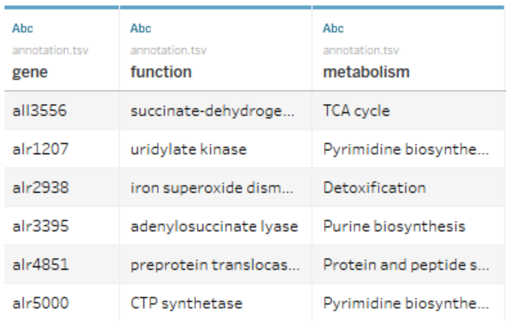
Match value: ase

[✓ Contains]
Starts with
Ends with
Exactly matches

Exclude [ ]

Include all values when empty [✓]

(a) *Filter to retain only rows containing “ase” in the column* `function`*.*



Abc annotation.tsv gene	Abc annotation.tsv function	Abc annotation.tsv metabolism
all3556	succinate-dehydroge...	TCA cycle
alr1207	uridylate kinase	Pyrimidine biosynthe...
alr2938	iron superoxide dism...	Detoxification
alr3395	adenylosuccinate lyase	Purine biosynthesis
alr4851	preprotein translocas...	Protein and peptide s...
alr5000	CTP synthetase	Pyrimidine biosynthe...

(b) *Result of the filter applied to the data connection.*

Figure 5: *Filter settings and result for Question 3 (b).*

The resulting data connection is called `annotation` and only contains the rows which have the wildcard string “ase” in the column `function`.

A packaged Tableau workbook `q3b.twbx` which contains the data connection and the applied filter is available to download at https://github.com/gardiners/a2t3/raw/master/q3b.twbx. Be sure to switch to the “Data Source” tab to inspect the filter settings.

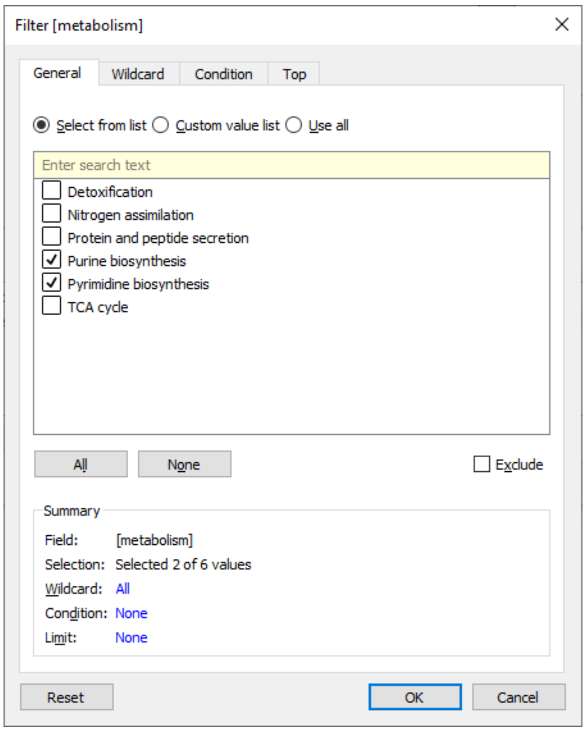
(c)

As for **(a)** and **(b)** we can use a data connection filter to retain only the rows we require. The process is as follows:

- Connect to the text file `annotation.tsv`.
- Add a data source filter to the `metabolism` field.

- Under the "General" tab, choose the "Select from list" radio box.
- Check the boxes for "Purine biosynthesis" and "Pyrmidine biosynthesis".
- Click "OK" to close the "Filter [metabolism]" dialog, and then click "OK" again to close the "Edit Data Source Filters" dialog.

The resulting filtered dataset contains only the rows which match these conditions.



Filter [metabolism]

General | Wildcard | Condition | Top

[✓ Select from list] Custom value list Use all

Enter search text

- [ ] Detoxification
- [ ] Nitrogen assimilation
- [ ] Protein and peptide secretion
- [x] Purine biosynthesis
- [x] Pyrimidine biosynthesis
- [ ] TCA cycle

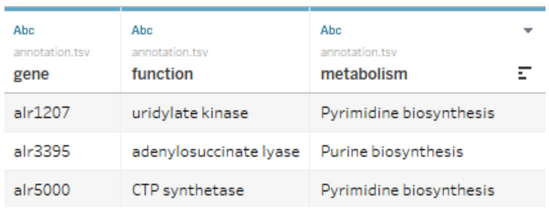
All | None | ☐ Exclude

Summary

Field:	[metabolism]
Selection:	Selected 2 of 6 values
Wildcard:	All
Condition:	None
Limit:	None

Reset | OK | Cancel

(a) *Filter settings to retain only rows with the desired strings in column* `metabolism`*.*



gene	function	metabolism
alr1207	uridylate kinase	Pyrimidine biosynthesis
alr3395	adenylosuccinate lyase	Purine biosynthesis
alr5000	CTP synthetase	Pyrimidine biosynthesis

(b) *Result of applying the filter.*

Figure 6: *Filter and results for Question 3 (c).*

A packaged Tableau workbook `q3c.twbx` which contains the data connection and the applied filter is available to download at https://github.com/gardiners/a2t3/raw/master/q3c.twbx. Be sure to switch to the "Data Source" tab to inspect the filter settings.

References

Luke, Douglas. 2019. *UserNetR: Data sets for A User's Guide to Network Analysis in R.*

Pedersen, Thomas Lin. 2020. *ggraph: An Implementation of Grammar of Graphics for Graphs and Networks.* https://cran.r-project.org/package=ggraph.

Wickham, Hadley. 2016. *ggplot2: Elegant Graphics for Data Analysis.* 2nd editio. Use R! Cham: Springer International Publishing. https://doi.org/10.1007/978-3-319-24277-4.

World Bank. 2016. *World Development Indicators 2016.* The World Bank. https://datacatalog.worldbank.org/dataset/world-development-indicators.