4. Interactive Graphics with R

Introduction

While **ggplot2** is an excellent application for creating static R graphics it does not have interactive capabilities. In this section we will examine a number of R packages that allow for dynamic interactivity including plotly, leaflet, highcharts DT and visNetwork. **Plotly** is a general purpose graphics library which has the great advantage of converting a wide range of ggplot2 graphics to interactive graphs. **Leaflet** is the go-to R package for interactive maps while **highcharts** is excellent for time series plots. **DT** is not a graphics package but allows for the creation of **interactive data tables** which can be very useful for filtering and exporting data prior to creating visualisations. Finally, **visNetwork** is a powerful library for visualising network data.

These packages are developed by RStudio and collectively known as **htmlwidgets** and are R versions (or wrappers) of equivalent **Javascript** libraries. The website **htmlwidgits.com** contains a wealth of information on these packages as well as information on a number of other packages which are not featured in this course. In addition, user guides associated with each of the packages can be found by accessing each library in R studio together with worked examples.

4.1 Plotly

While the developer of ggplot2 **Hadley Wickham** (who has also developed a wide arrange of other R libraries to the extent that the packages are sometimes referred to as part of the **hadleyverse**!) is now working on an interactive version of ggplot2 known as **ggvis** the package plotly will generate interactive graphics for a wide range of ggplot2 graphics. Generating an interactive chart using plotly couldn't be easier. We just assign a name to a ggplot2 graphic and then enter the code **ggplotly**(*ggplot2 file name*). Please note that we must install plotly using the code **Install ("plotly")** as is the case with all R packages. Once installed the packages must be loaded in each session using the code **Library("plotly")**.

Example

Consider the following code used to generate the ggplot2 facet plot of accident type by day of week shown in Figure 2.10 on page 31.

ggplot(Dacc1,aes(Weekday,Number)) + geom_bar(stat ="identity",fill = "Blue")+ facet_wrap(~Type,ncol=2,scales = "free")+ scale_x_discrete(limits= c("Mon","Tues","Wed","Th","Fri","Sat","Sun"))

The ggplot2 output is shown below:

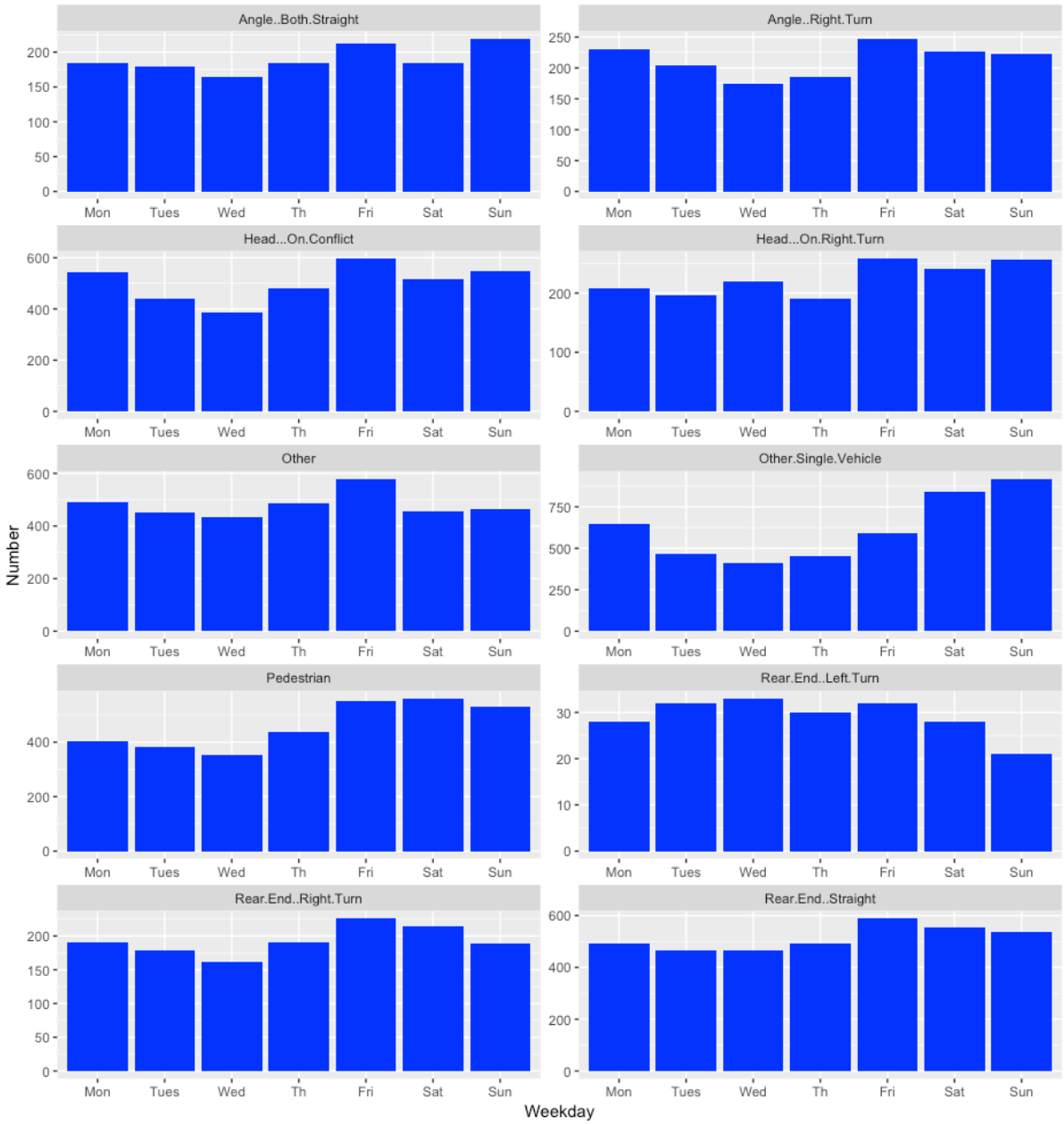


Figure 4.1: Trellis plot generated by ggplot2 of road traffic accidents by accident type and weekday, 2009-11.

To create an interactive plotly graph we need just two additional lines of code. The first allocates the name **accident** to the ggplot2 code:

accident <- *ggplot(Dacc1,aes(Weekday,Number)) + geom_bar(stat ="identity",fill = "Blue")+ facet_wrap(~Type,ncol=2,scales = "free")+ scale_x_discrete(limits= c("Mon","Tues","Wed","Th","Fri","Sat","Sun"))*

The second line is just ggplotly followed by **accident** which is placed in brackets i.e. **ggplotly**(accident)

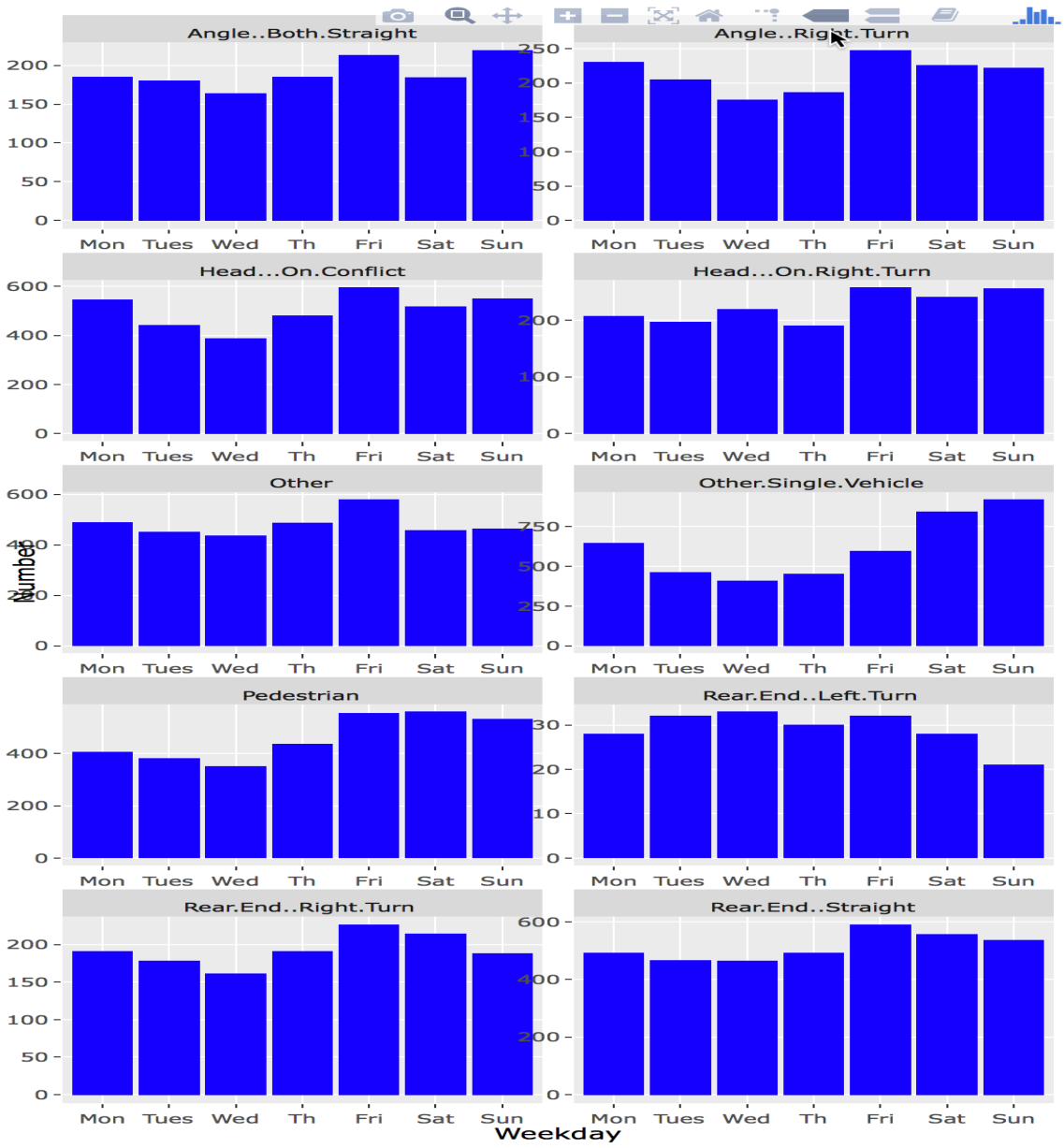


Figure 4.2: Plotly graph of road traffic accidents by accident type and weekday, 2009-11

The plotly graphic looks very similar to the ggplot2 graphic. The main difference between the two plots the array of interactive features which can be seen at the top of the plotly graph. Please note that plotly also has its own language which is independent of ggplot2.

Another example which is shown in Figure 4.3 is a plot of the number of crimes recorded each year between 2003 and 2015 in Ireland where each trellis plot is a specific crime. The legend on the right of the plot lists garda division which can be added or removed from the plot interactively by selecting the division.

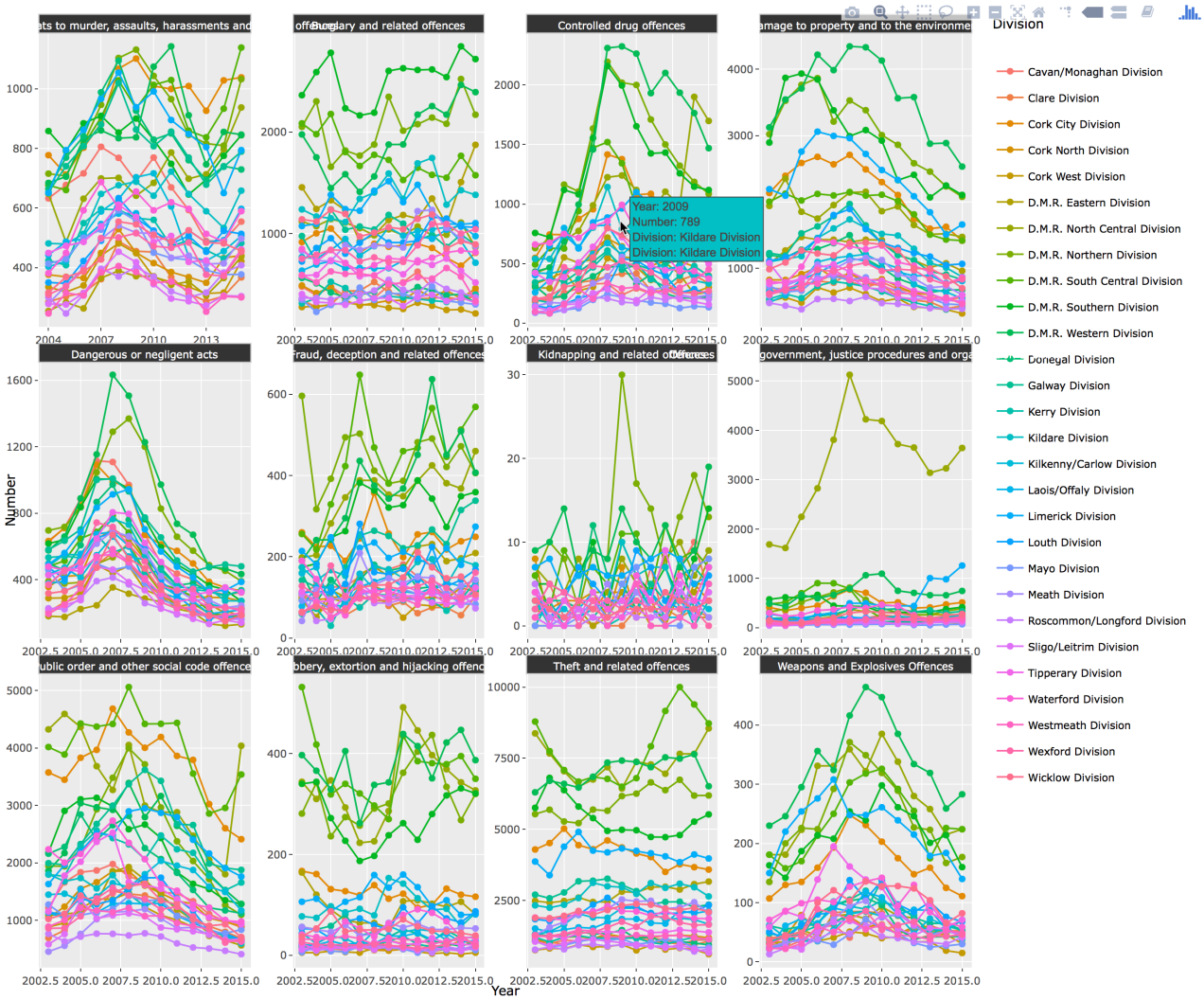


Figure 4.3: Plotly graph of reported crime by division, 2003-2015

The plotly code to generate Figure 4.3 is:

```
25
26 # ggploty crime example with grrouping by crime
27 gardacrime %>%
28   group_by(Division, Number)
29
30 gardacrime1 <- ggplot(data = gardacrime,aes(x = Year,y = Number,group = Division,col = Division)) +  geom_point()+ geom_line() +
31   facet_wrap(~Crime,ncol = 6,scales = "free")
32
33
34 ggplotly(gardacrime1)
```

On line 27 the **%>%** is known as a **pipe** (from the R package **magrittr**) which is used by all the htmlwidgets and is a shorthand way of writing code by piping the data from line to line automatically. For example, line 28 knows to **group** Divisions for the **gardacrime** data set without being explicitly told the data set is **gardacrime**.

Note the **group = Division and col = Division** code on line 30. This code allows each division to have its own line and colour in the plot. The code on line 31 **scales = "free"** allows each plot to have its own scale. Use **scales= "fixed"** if you require the same y-axis scale in each plot.

Plotly allows a wide range of plots to be generated including **interactive box plots** as shown in Figure 4.4. This plot visualises the the number of crimes reported in each garda division each year for the period 2003-15. The data for each box comprises 16 values representing the number of crimes in each of the 16 crime categories.

The code to generate the plot in Figure 4.4 is shown below:

```
35
36 # ggploty crime example with boxplots
37 gardacrime2 <- ggplot(data = gardacrime,aes(x = Year,y = Number,group = Division,col = Division)) +  geom_boxplot() +
38   facet_wrap(~Division,ncol = 4,scales = "free") +
39   theme(strip.background = element_rect(fill = "grey20",colour = "grey80",size =1),strip.text = element_text(colour = "white"))
40
41 ggplotly(gardacrime2)
```

On Line 37 the code **group = Division** allows the legend to be created which is not strictly required while **col = Division** assigns a colour to each of the Divisions. Line 39 is a **theme** template (of which ggplot2 has many) which gives the reversed white text on black background header for each Facet panel.

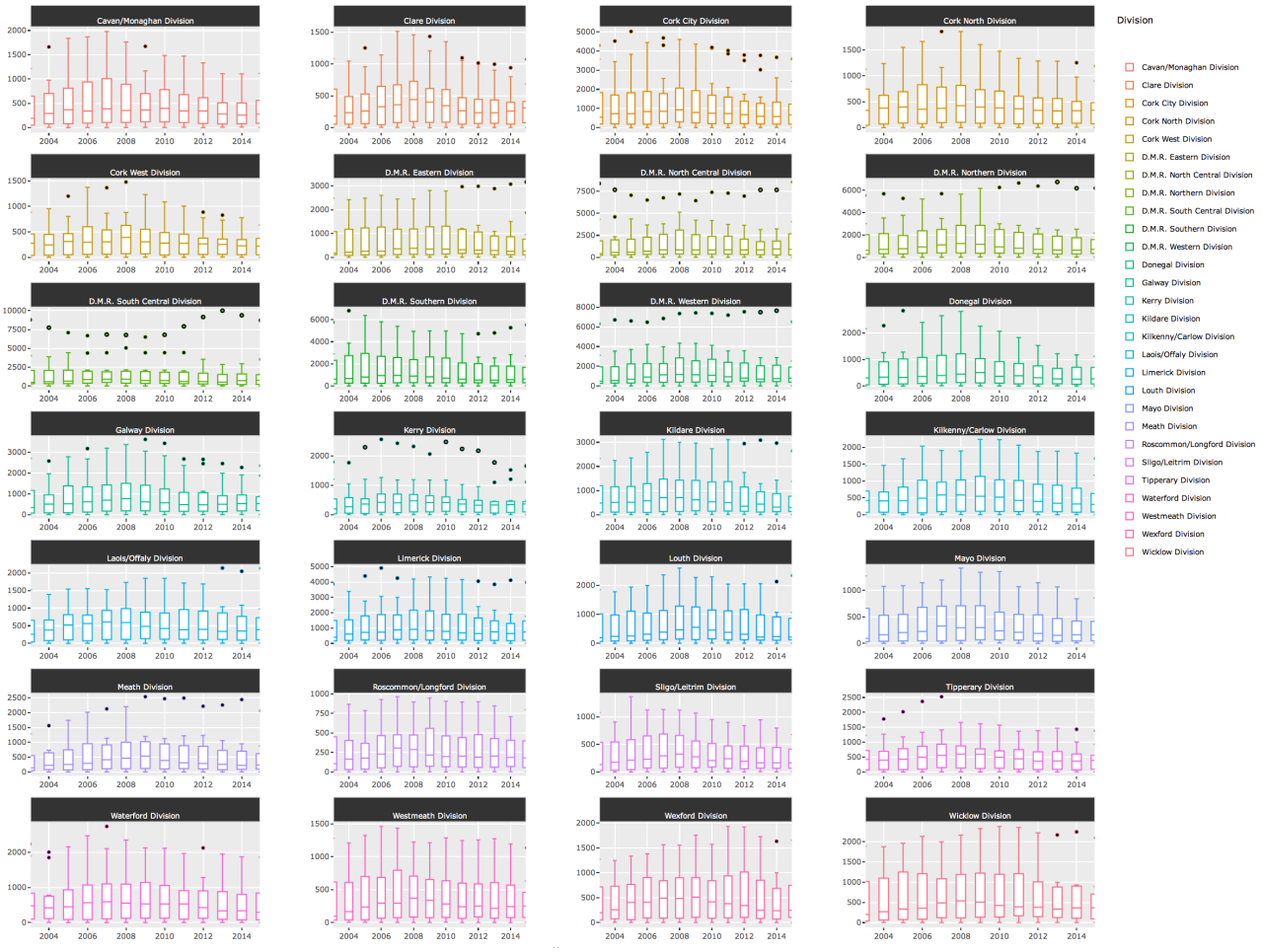


Figure 4.4: Interactive box plot of all crime by Year by Division, 2003-2015

Plotly has it owns scripting language which is similar to ggplot2. For example, the bar chart in Figure 4.5 uses plotly's own code which is shown below

```
Appreg %>%
  plot_ly(x = ~Number, y = ~Category) %>%
          add_bars() %>%
            layout(barmode = "stack", barnorm = "normal",
                   margin = list(l = 240),
                   yaxis = list(title = list(text = "")))
```

Note the function call **plot_ly** instead of **ggplot** while defined variables have **~** before their name e.g. **~Number**. Also note that Instead of **geom_bar** the command **add_bars** is used. Further details on plot_ly can be found in the package details or on the site htmlwidgets.com.

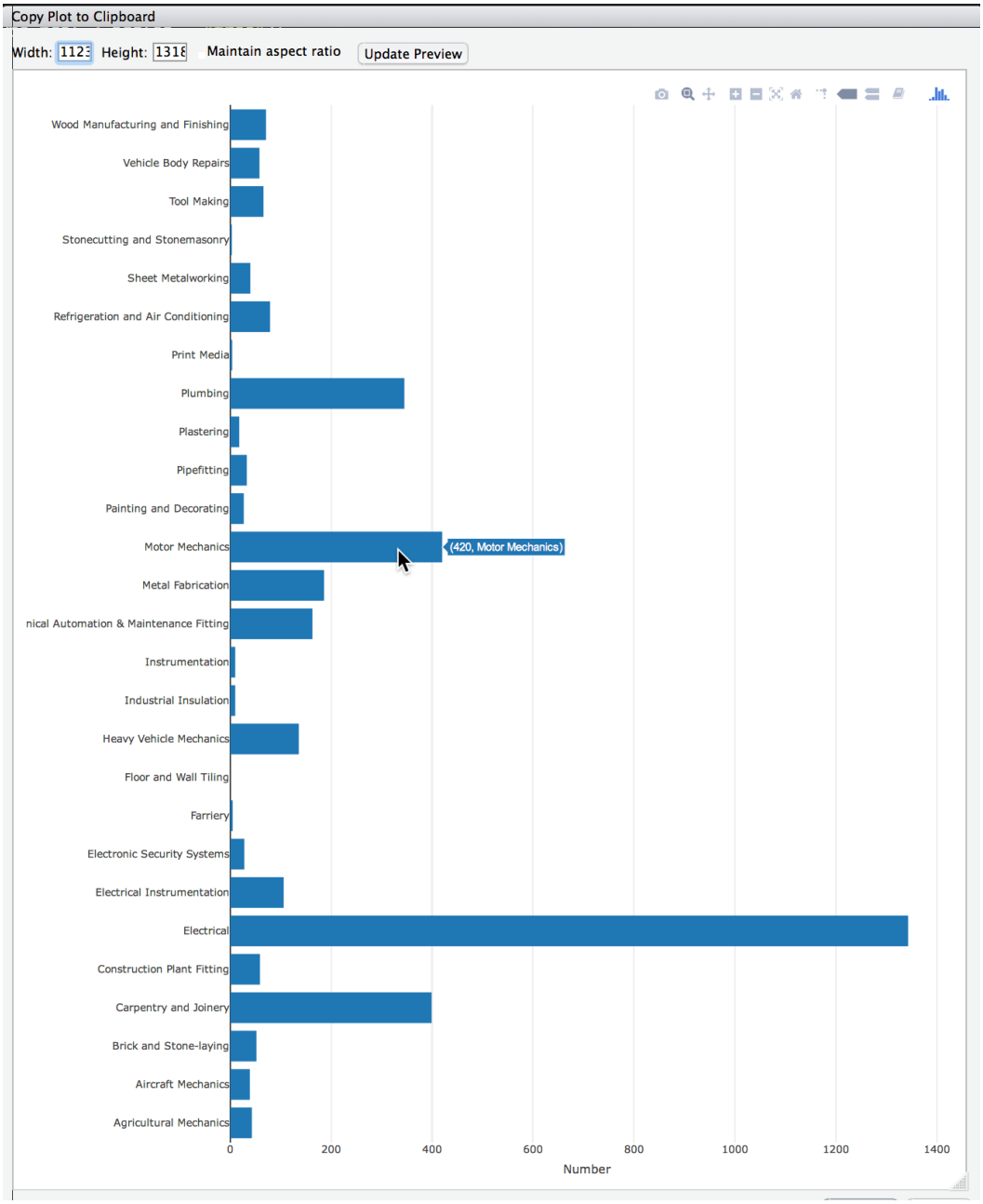


Figure 4.5: Plotly bar chart of apprentices by category, 2015

Exercise

1. Using plotly create an interactive facet chart using the data set **StLoc** which records the location of the home address of students entering iadt by course in 2016. Each plot in the facet will represent an iadt course. This data set was studied on page 46 in section 1.

2. Using ggplot2 and plotly create an interactive facet chart of grant awarded by educational training board. Each facet plot will be a bar chart. The data is contained in the file **grantsolaslong** and visualised in ggplot2 on page 47.

3. Using ggplot2 and plotly create an interactive facet chart of Dublin road traffic accidents by by accident **type**. Each facet plot will be a bar chart. The data is contained in the file **DubRTA**.

4.2 Leaflet

The leaflet htmlwidget is particularly good for spatial visualisation with a few lines of code providing high quality maps with interactive features. Geospatial visualisation has become increasingly important with mapping and statistical state authorities throughout the world working closely together. For example, in Ireland mapping agencies like Ordnance Survey (together with their partners ESRI) create shape files while statistical bodies like the Central Statistics Office generate the data that can be attached to the shape files. We saw the result of this collaboration in Section 3 where we mapped Census 2016 data to counties, electoral divisions and small areas. **Everything happens somewhere** is a good description of the importance of geospatial visualisation.

Example 1 - Scattermaps

The data file **DubRTA** contains the longitude and latitude of Dublin road traffic accidents over the past five years. The data set also lists the type of accident at each location.

To visualise this data using leaflet requires the following code:

```
11 DubRTA %>%
12   leaflet() %>%
13   addTiles() %>%
14   addCircleMarkers(lat = ~Latitude,lng = ~Longitude,label = ~paste("Label:",Type))
15
```

DubRTA is the data file. The symbol **%>%** is the **pipe** which puts **DubRTA** into leaflet on line 12. This is turn is piped into **add tiles** while line 14 adds circle markers using the latitude and longitude variables for each accident. In addition line 14 also allows the type of accident to be shown (**Type**) when a user mouses over the map. Note the **~** symbol which is needed for all variables in leaflet.

Entering the above code a zoomed map of Dublin is shown in Figure 4.6 which can be moused over to provide more details. As was the case with plotly (and all R generated plots) this graphic can be published on the web using **Rpub**.

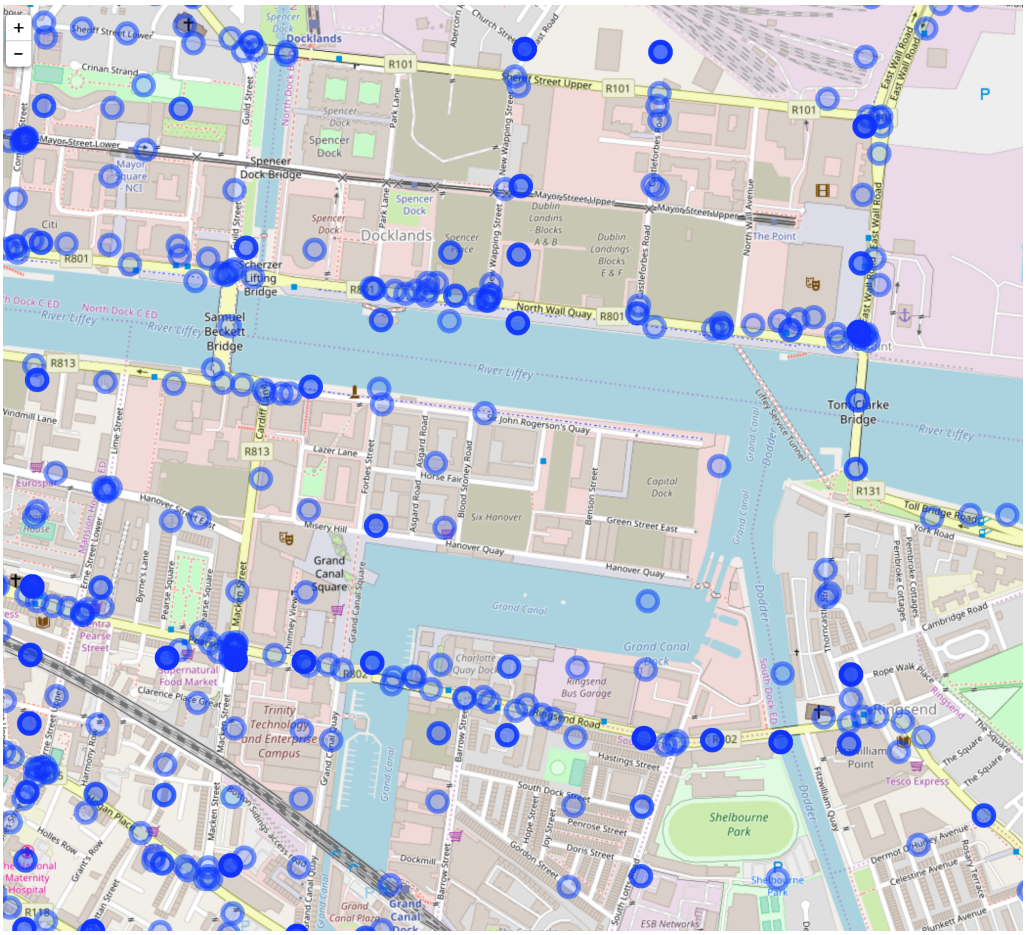


Figure 4.6: Road accidents in Dublin using leaflet, 2003-2015

One drawback with leaflet is that it can be difficult to work with legends and to interactively filter on variables. In Figure 4.7 a legend with **accident type** encoded as colour can be obtained with some additional code.

It would be useful to be able to select each accident type from the legend and view just those accidents e.g. example the map of pedestrian accidents only. However this level of interactivity may require the use of **Shiny** which is an environment for developing interactive web applications.

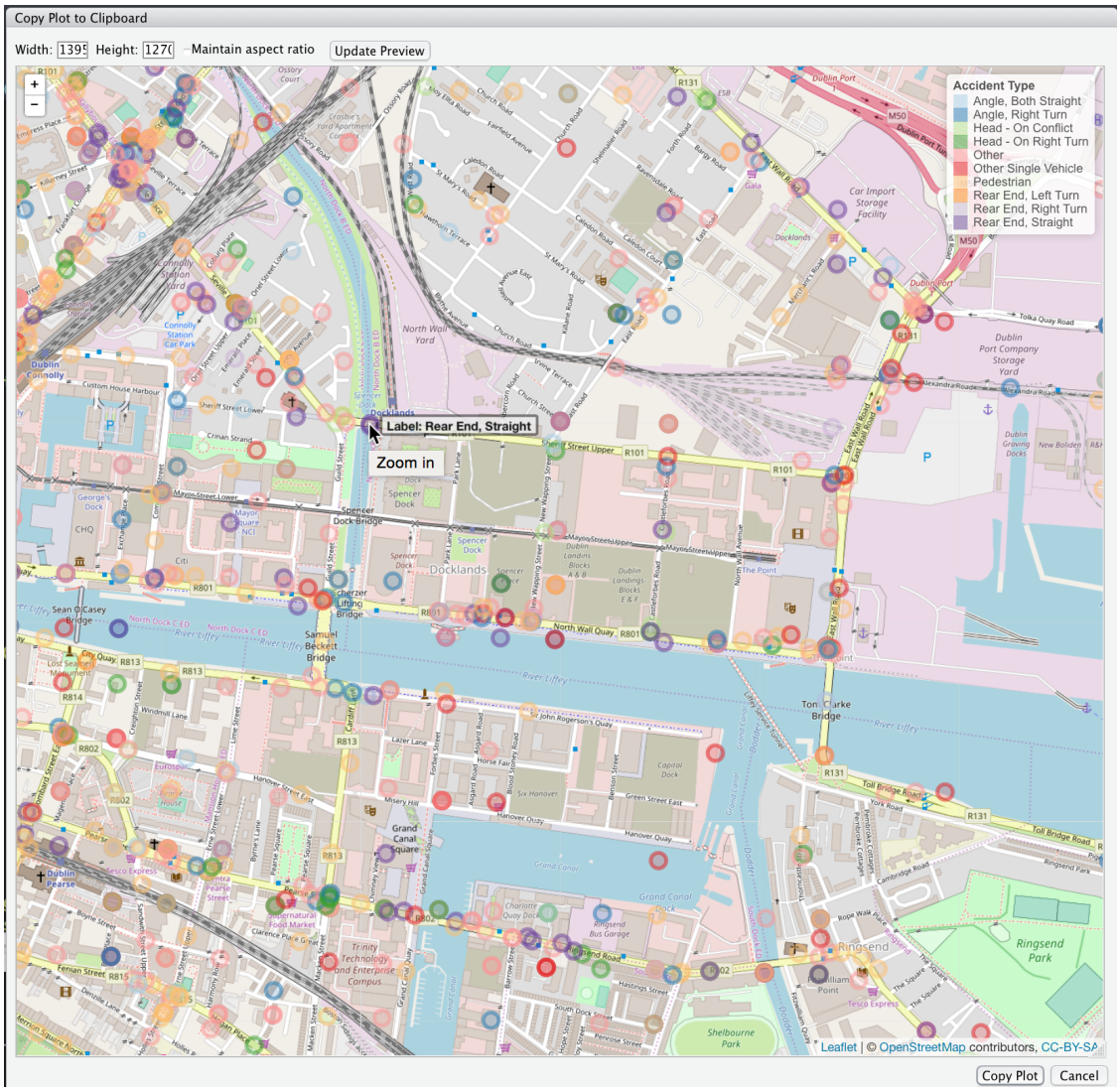


Figure 4.7: Road accidents in Dublin using leaflet with legend added

The new line of code on line 12 comes from a colour palette from a package called **colorbrewer** which will be explained in the next section. Line 16 assigns this colour to the accidents together with some opacity aesthetics.

```
12 qpal <- colorFactor("Paired", DubRTA$Type)
13 DubRTA %>%
14   leaflet() %>%
15   addTiles() %>%
16   addCircleMarkers(lat = ~Latitude,lng = ~Longitude,label = ~paste("Label:",Type),color = ~qpal(DubRTA$Type),opacity = 0.5,fillOpacity = 0.2)  %>%
17   addLegend(pal = qpal,value = ~Type, title = "Accident Type")
18
```

Example 2 - Working with Shape Files

Leaflet can also work with **shape files** whether in **ESRI** or **GeoJSON** formats. For example, to import a county shape file and link it to a CSO excel data file as we did in October (using Tableau) requires the code shown overleaf.

```
19
20  # Chloropleth Map using Shape Files
21  # Importing Shape Files (must not be stored on usb)
22
23  county_shapefiles <- read_sf(dsn = "data-raw/ShapeFiles/Admin_Counties_Generalised_20m_OSi_National_Boundaries/")
24
25
26  # CSO Theme10 Data already imported using import csv in RStudio
27  countycso <- County
28
29  # County Shape Files joined to CSO data
30  csoosi <- county_shapefiles %>%
31    left_join(countycso)
32
33
34  # setting colour for each county in theme 10 using colorbrewer.org
35  qpal <- colorNumeric("Blues", csoosi$T10_1_18M, na.color = "#808080")
36
37  csoosi %>%
38    leaflet() %>%
39    addTiles() %>%
40    addPolygons(
41      stroke = FALSE,
42      smoothFactor = 0.2,
43      fillOpacity = 1,
44      color = ~ qpal(T10_1_18M),
45      label = ~ COUNTY,
46      popup = ~ paste("County:", COUNTY, "<br/>", "Number of Males aged 18:", T10_1_18M)) %>%
47    addLegend(pal = qpal,value = ~T10_1_18M, title = "Number of Males aged 18 years")
```

Line 23 imports the shape files which are located in the data-raw folder. The import will **not** work if the shape files are stored on an USB key. Line 30 assigns the **County** csv file containing all the Census variables to the variable **countycso**. Lines 30 and 31 involve joining both files. Line 30 assigns the name **csoosi** to the shape file **county_shapefile** while line 31 joins the **countycso** file to it. Note R helpfully tells us we are joining on the variable GUID which is common to both files. Line 35 uses the code **colorNumeric** as we want a quantitative colour shading. The term "**Blues**" comes from our palette selection (single hue selection) from the website **colorbrewer.org** as shown in Figure 4.8 while shading is allocated to the variable CSO variable **T10_1_18M** which lists the number of males aged 18 years by county.

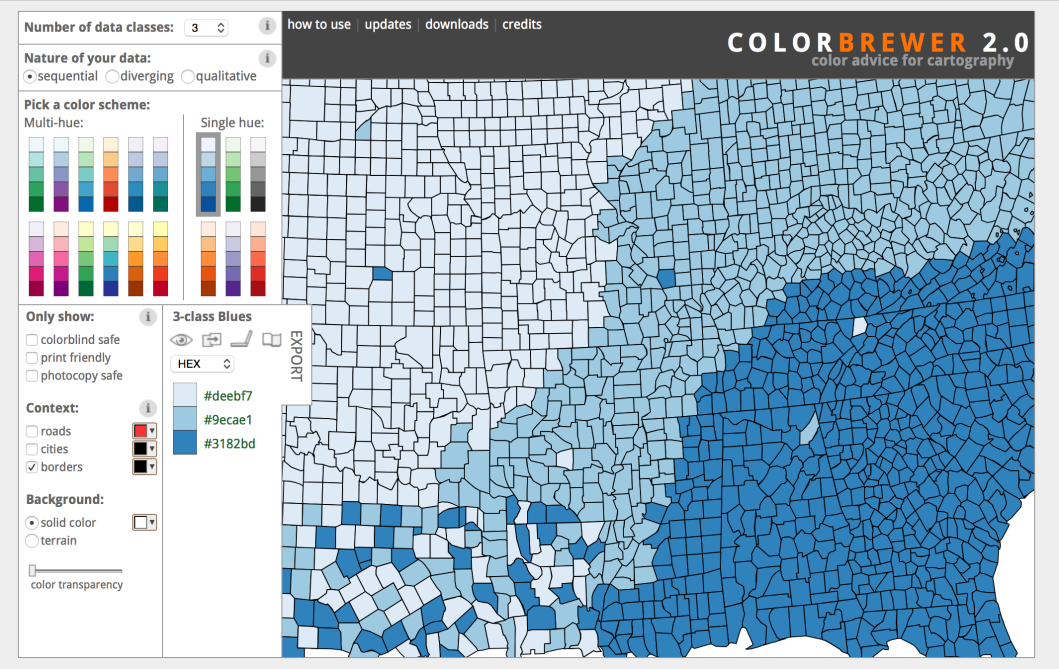


Figure 4.8: Selecting palette from ColorBrewer package

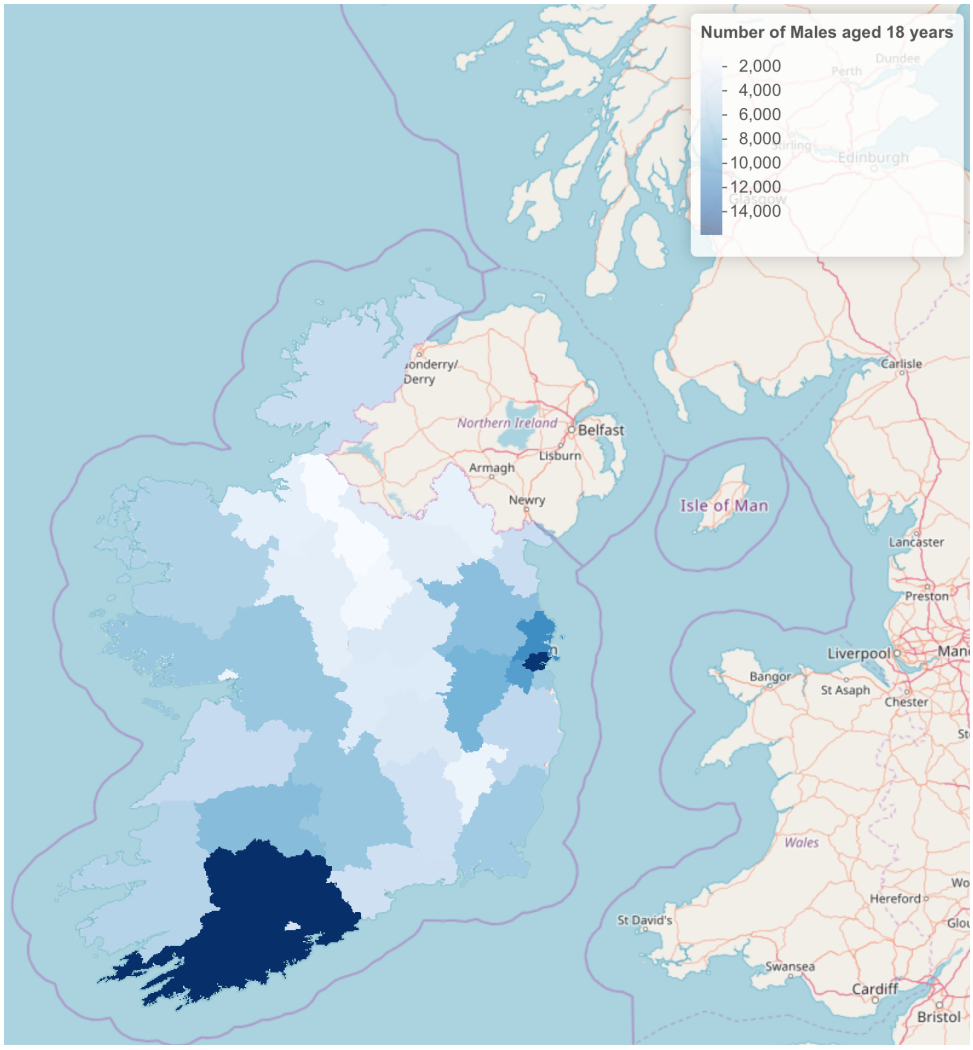


Figure 4.9: Number of Males aged 18 by county, 2016

Using the techniques outlined in this section it is possible to generate chloropleth maps based on electoral divisions and small areas. For example Figure 4.10 is a plot of the number of Males aged 18 years by electoral division.

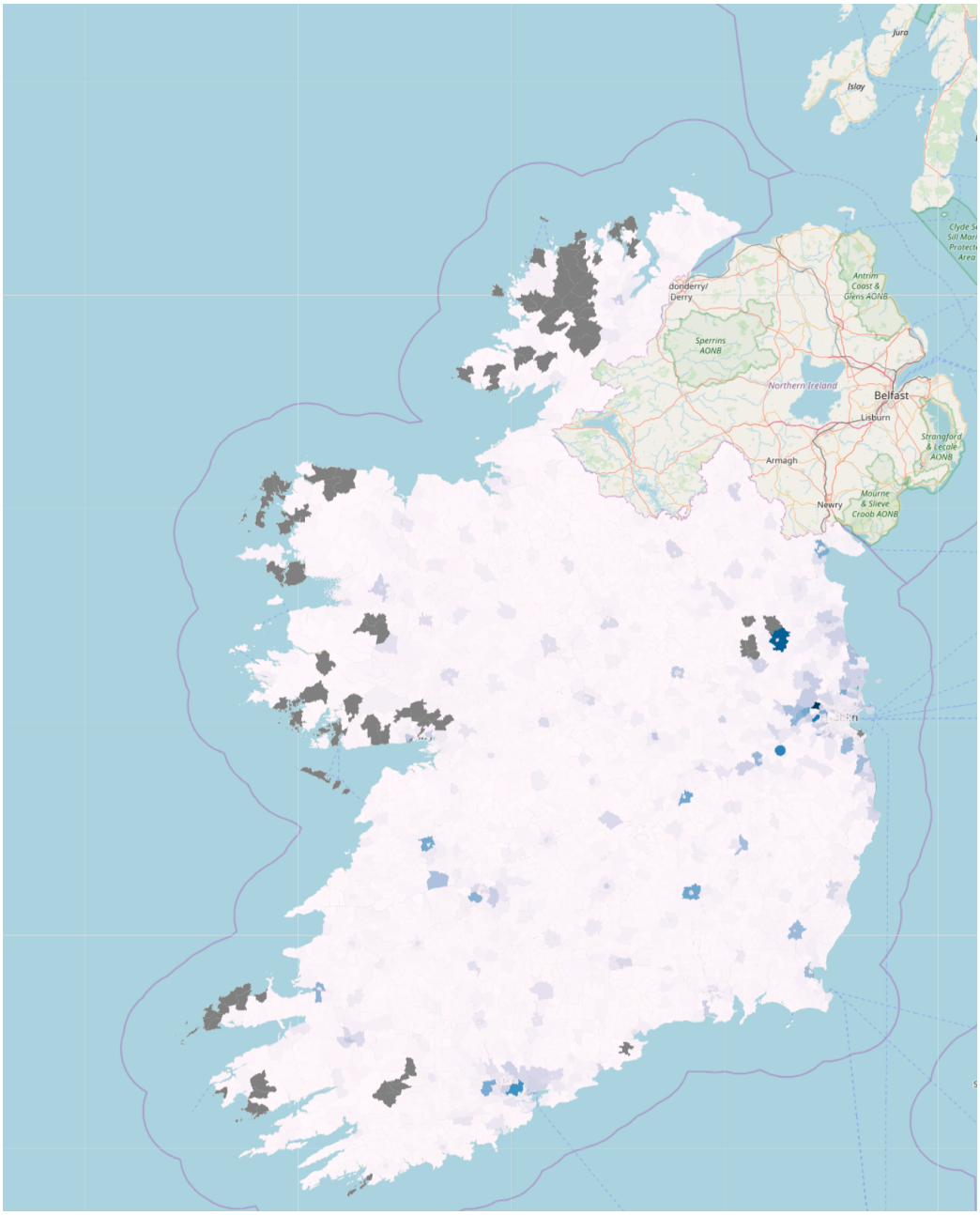


Figure 4.10: Number of Males aged 18 by electoral division, 2016

Exercises

1. Select a variable of interest to you from the Census 2016 data file and create a chloropleth map that shows the distribution of your chosen variable by **county** using the procedure outlined in this section.

2. a) Usingleaflet create chloropleth map using the data set **StLoc** which records the location of the home address of students entering iadt by course in 2016.

   b) Using colourbrewer create a legend for this map created in part a) using the course code variable **Program.**

   This data set was studied on page 46 in section 1.

4.3 Highcharter

The package **Highcharter** is particularly useful for plotting time series data and as is the case with the previous libraries a few lines of code can generate impressive graphics. For example, the graphic in Figure 4.11 is a plot of one crime category in Ireland grouped by garda division between 2003 and 2015. The code and used to generate Figure 4.11 is shown below:

```
 6
 7 highchart(type = "stock") %>%
 8   hc_add_series(data = gardacrime3,
 9                 type = "line",
10                 hcaes(
11                   x = gardacrime3$Year,
12                   y = gardacrime3$Number,
13                   group = Division
14                 )) %>%
15   hc_legend(enabled = TRUE)
16
```

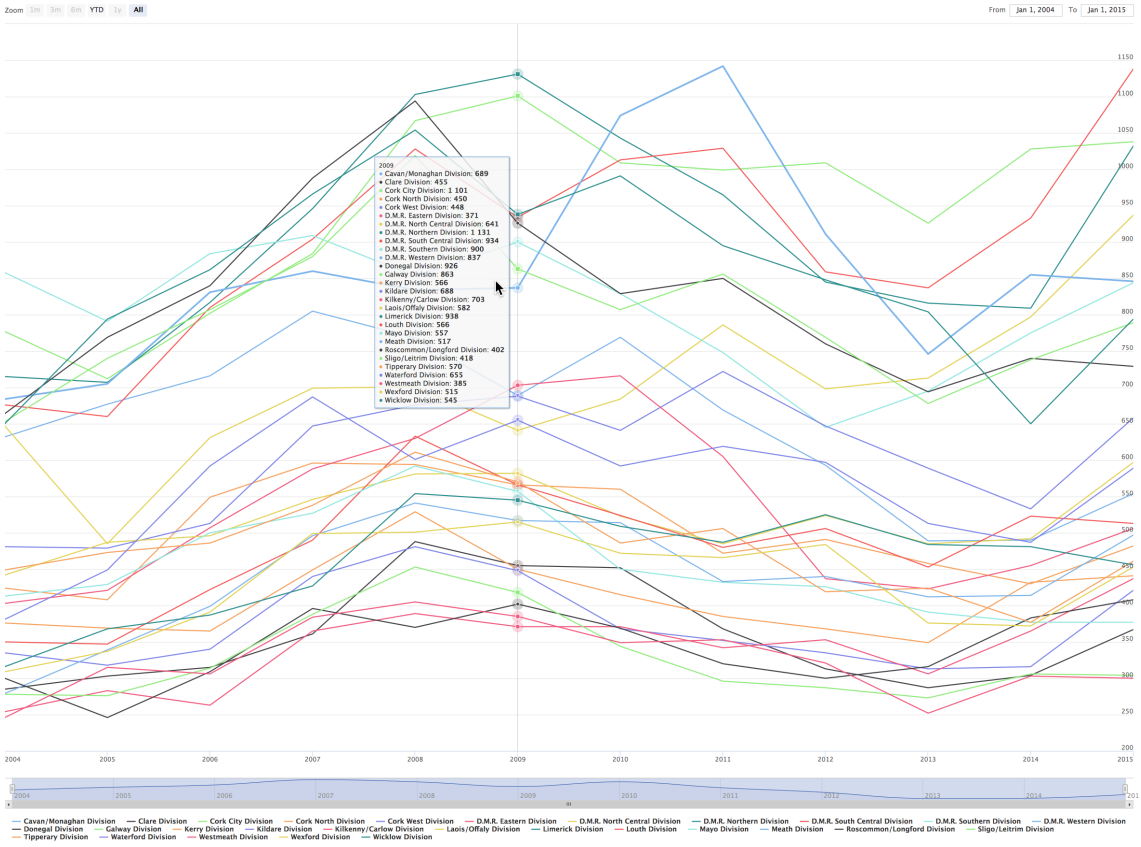


Figure 4.11: Crime in Ireland by garda division, 2003-2015

Line 7 defines the graphic as a **stock** type. Line 8 defines the data set and the x, y and **grouping** variables while line 15 allows the legend to be displayed.

Exercise

Using the datafile **Burglary1** create a time series plot of the number of burglary crimes in Ireland between 2003 and 2015 with the grouping variable **type of crime**.

4.4 DataTables

The library **DT** is an excellent package that allows for the creation of dynamic and interactive datatables. These tables allow for **filtering** and **searching** using just a few lines of code. It is also straightforward to download the result of searches in Excel and .csv formats.

Example

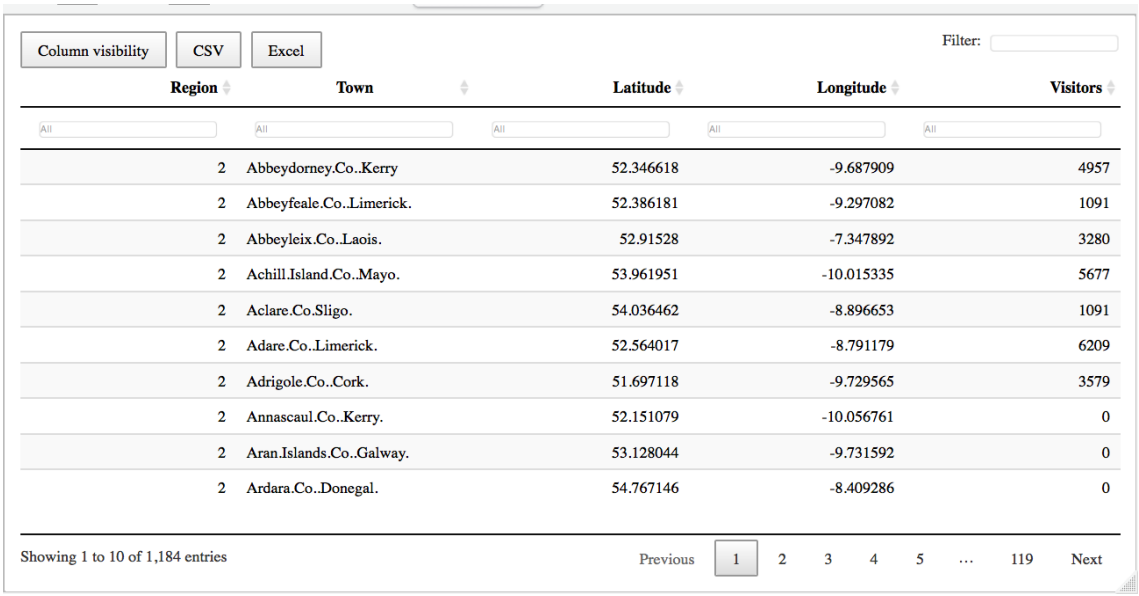
The data set **town** provides information on towns that tourists to Ireland visited in 2017. The longitude and latitude of each town is also included in addition to the region of the world the tourist comes from. To create a datatable of this data set we just enter the following code into our R script:

```
2  library("DT")
3
4
5  Touristown %>%
6    datatable(
7      rownames = FALSE,
8      extensions = c("Buttons", "Responsive"),
9      filter = list(position = "top"),
10     options = list(pagelength = 10,
11                    buttons = c("colvis","csv","excel"),
12                    language = list(sSearch = "Filter:"),
13                    dom = "Bfrtip"))
14
```

Line 3 **loads** the package DT while lines 7 to 15 define the dataset (Touristown) and includes options on how the table is displayed. Line 10 assigns buttons to the interface together with specifying a responsive design. This means the output will adjust to the specification of the device the user is viewing the content on e.g. iPad or mobile phone. Line 10 sets the number of displays to 10 records per page. Line 11 creates three buttons - **colvis** shows and hides columns while **csv** and **excel** are the two download options. Line 14 changes the name of the search page to **filter**.

Each table control element in DataTables has a single letter associated with it. The letter is specified in the **dom** configuration on line 13 to “Bfrtip” which means include **B**uttons, a **f**ilter, a **t**able, **i**nformation summary and **p**agination control on the table

Executing this code in the R script **datatables.R** we obtain the datatable as shown below.



Column visibility | CSV | Excel | Filter:

Region	Town	Latitude	Longitude	Visitors
2	Abbeydorney.Co..Kerry	52.346618	-9.687909	4957
2	Abbeyfeale.Co..Limerick.	52.386181	-9.297082	1091
2	Abbeyleix.Co..Laois.	52.91528	-7.347892	3280
2	Achill.Island.Co..Mayo.	53.961951	-10.015335	5677
2	Aclare.Co.Sligo.	54.036462	-8.896653	1091
2	Adare.Co..Limerick.	52.564017	-8.791179	6209
2	Adrigole.Co..Cork.	51.697118	-9.729565	3579
2	Annascaul.Co..Kerry.	52.151079	-10.056761	0
2	Aran.Islands.Co..Galway.	53.128044	-9.731592	0
2	Ardara.Co..Donegal.	54.767146	-8.409286	0

Showing 1 to 10 of 1,184 entries | Previous | 1 | 2 | 3 | 4 | 5 | … | 119 | Next

Further information on data tables can be found on **datatables.net**

Exercises

i) Create a datatable of the **long** file format of the file **RoadAccident(wide & long)** provided in the excel file Data2018.xlsx

ii) Download a file of the number Sunday accidents by accident type in Excel and .csv formats.

4.5 Network Visualisations

Network diagrams are very useful for visualising the connections and relationships between values in a variable. Networks are used less often than other types of visualisation but can be very useful if your data is in a format that shows movement in a **two-way direction**. For example, origin and destination data in a car transport study. In this case the values in the origin and destination are the same and we might be interested in visualising the number of cars travelling from Dublin (origin) to Cork (destination) but also from Cork(origin) to Dublin (destination).

Worked Example

In this session we will use domestic tourism data from **Failte Ireland** to illustrate the use of Network visualisations. The data set **TourismOD** gives the number of domestic holidaymakers travelling from where they normally reside (origin) to their holiday county (destination). An extract of the raw data file is shown in Table 4.1

	A	B	C	D	E	F
1	County	Carlow	Cavan	Clare	Cork	Donegal
2	Dublin	6	16	14	96	14
3	Carlow	0	0	0	0	0
4	Kildare	0	0	2	8	0
5	Kilkenny	0	0	0	26	0
6	Laois	0	0	2	0	0
7	Louth	0	0	0	7	0
8	Meath	0	0	5	0	0
9	Offaly	0	0	0	6	0
10	Westmeath	0	3	0	0	5
11	Wexford	5	6	6	7	0
12	Wicklow	0	0	0	9	0
13	Clare	0	8	3	21	0
14	Cork	3	3	9	75	8
15	Kerry	6	5	18	150	0
16	Limerick	0	0	7	11	0
17	Tipperary	0	3	0	13	0
18	Waterford	13	0	0	16	0
19	Galway	3	23	20	28	16
20	Leitrim	0	0	0	0	0
21	Mayo	6	3	4	9	10
22	Roscommon	0	0	0	0	0
23	Sligo	0	0	5	3	4
24	Cavan	0	0	0	0	3
25	Donegal	0	0	0	9	25
26	Monaghan	0	0	0	0	0
27	Longford	0	0	0	0	0

Table 4.1: Origin (column A) and destination (other columns) of domestic tourists

In Table 4.1 column A lists the origins (of which there are 26 representing each county) while each row gives the corresponding holiday destination. For example six Dublin residents holidayed in Carlow while 16 holidayed in Cavan and so on. To visualise this data we need to transform it into **long** format with just three columns - Origin, Destination and Total (number or trips). We can do this easily by importing the data shown in Table 4.1 into Tableau. We can then select all the destination columns as shown in the **Data Source** pane and select **pivot** as shown earlier in Section 1.3. The data set is now transformed to long format with three columns which we can rename to **Origin**, **Destination** and **Total** as shown in Table 4.2.

Abc Clipboard_20180... **Origin**	Abc Pivot **Destination**	# Pivot **Total**
Dublin	Carlow	6
Carlow	Carlow	0
Kildare	Carlow	0
Kilkenny	Carlow	0
Laois	Carlow	0
Louth	Carlow	0
Meath	Carlow	0
Offaly	Carlow	0
Westmeath	Carlow	0
Wexford	Carlow	5
Wicklow	Carlow	0
Clare	Carlow	0
Cork	Carlow	3
Kerry	Carlow	6
Limerick	Carlow	0
Tipperary	Carlow	0
Waterford	Carlow	13
Galway	Carlow	3
Leitrim	Carlow	0
Mayo	Carlow	6

Table 4.2 Long format of origin-destination data

The next stage is to visualise the data in Table 4.2 using the R libraries **visNetwork** and **Igraph**. **VisNetwork** is part of the htmlwidget library that binds to the Javascript library **vis.js** which is used to visualise network data. The library **igraph** is used for further analysis and customisation. One advantages of using **visNetwork** is that it integrates very well with **igraph**. As with the other html widgets examined in this section the output can be included in **RMarkdown** documents or incorporated into **Shiny apps** which is an interactive web based framework which we will be working with later in the programme.

Node and edges

Network visualisation involves nodes that are connected by paths. The nodes are the values of the variable while the paths (also called edges) are the data representing the flows between the nodes. To visualise a network we need to create two files - the **node** file and the **edges** file. The node file, as stated, contains all the origin/destinations i.e. the values of column A in Table 4.1. The edges file lists all the paths from the origin node (origin county) to the destination node (destination county) which is the long format shown in Table 4.2 which is taken from Tableau. However to use **visNetwork** we need the node and edges file files to be in a particular format. The node file requires an **id field** which gives each of the origin counties a unique number from 1 to 26 as shown in column A in Table 4.3.

	A	B	C	D	E	F
	id	county	latitude	longitude	location	label
	1	DUBLIN	712153	735780	DUBLIN	DUBLIN
	2	LAOIS	644825	693938	LAOIS	LAOIS
	3	ROSCOMMO	582317	778883	ROSCOMMO	ROSCOMMON
	4	WATERFORD	625792	604797	WATERFORD	WATERFORD
	5	LOUTH	699245	798377	LOUTH	LOUTH
	6	MONAGHAN	669141	824829	MONAGHAN	MONAGHAN
	7	MEATH	688919	762584	MEATH	MEATH
	8	LEITRIM	599868	819517	LEITRIM	LEITRIM
	9	WESTMEATH	635445	753975	WESTMEATH	WESTMEATH
	10	CORK	549968	582714	CORK	CORK
	11	DONEGAL	605132	904889	DONEGAL	DONEGAL
	12	LONGFORD	618694	777291	LONGFORD	LONGFORD
	13	GALWAY	543188	734366	GALWAY	GALWAY
	14	KILKENNY	653002	647665	KILKENNY	KILKENNY
	15	SLIGO	560387	823096	SLIGO	SLIGO
	16	OFFALY	619211	720628	OFFALY	OFFALY
	17	WICKLOW	709629	694080	WICKLOW	WICKLOW
	18	WEXFORD	696186	637381	WEXFORD	WEXFORD
	19	KILDARE	678852	718609	KILDARE	KILDARE
	20	KERRY	492785	601449	KERRY	KERRY
	21	TIPPERARY	599564	670540	TIPPERARY	TIPPERARY
	22	LIMERICK	549257	640003	LIMERICK	LIMERICK
	23	CAVAN	645942	803257	CAVAN	CAVAN
	24	MAYO	506350	808284	MAYO	MAYO
	25	CLARE	533997	684193	CLARE	CLARE
	26	CARLOW	680331	664233	CARLOW	CARLOW

Table 4.3 Node file format with column 1 containing the id for each node

It is best practice to place this **id field** in column A. Column E is the location field in case we require a different name for county while **label** in column F is the name of the node appearing on-screen in the visualisation. Note that the latitude and longitude fields are not required for the visualisation.

The edges file format requires **two id fields** for the origin and destination nodes (or counties) as shown in column A and B in Table 4.4 and are called **from** and **to**, respectively. The names of the counties are shown in columns C and D with the headings **send.location** and **receive.location**. Column E is the total number of trips for each connection and is called **total.items**. For example, row 1 shows that 6 tourists travelled from Dublin (id 1) to Carlow (id 26).

	A	B	C	D	E
1	from	to	send.location	receive.locatio	total.items
2	1	26	DUBLIN	CARLOW	6
3	26	26	CARLOW	CARLOW	0
4	19	26	KILDARE	CARLOW	0
5	14	26	KILKENNY	CARLOW	0
6	2	26	LAOIS	CARLOW	0
7	5	26	LOUTH	CARLOW	0
8	7	26	MEATH	CARLOW	0
9	16	26	OFFALY	CARLOW	0
10	9	26	WESTMEATH	CARLOW	0
11	18	26	WEXFORD	CARLOW	5
12	17	26	WICKLOW	CARLOW	0
13	25	26	CLARE	CARLOW	0
14	10	26	CORK	CARLOW	3
15	20	26	KERRY	CARLOW	6
16	22	26	LIMERICK	CARLOW	0
17	21	26	TIPPERARY	CARLOW	0
18	4	26	WATERFORD	CARLOW	13
19	13	26	GALWAY	CARLOW	3
20	8	26	LEITRIM	CARLOW	0
21	24	26	MAYO	CARLOW	6
22	3	26	ROSCOMMON	CARLOW	0
23	15	26	SLIGO	CARLOW	0
24	23	26	CAVAN	CARLOW	0
25	11	26	DONEGAL	CARLOW	0

Table 4.4 Edge file format

Now that the data is in a suitable form for analysis we can now use the R libraries VisNetwork and igraph and enter the code shown in Figure 4.12. Lines 2 and 3 load the libraries visNetwork and igraph. Lines 4 and 5 put the node and edges files from Table 4.3 and Table 4.4 into files called **map_nodes** and **map_edges,** respectively. Line 7 reduces the width of the path lines between nodes by a factor of 3 and creates a new column in the file edges called **width.** This allows a clearer view of the network visualisation (the reader can experiment with different scaling factors). Line 10 is the call to Viznetwork while line 11 sets to direction of movement in the form of an **arrow** to be located in the middle of the edge paths connecting origin and destination. Line 12 specials the colour of the edges to be blue while Line 13 uses igraph to show the network visualisation as a star but there are a number of other options available.

```
1  library("tidyverse")
2  library("visNetwork")
3  library("igraph")
4  map_nodes <- nodesR
5  map_edges <- edgesfinal
6
7  map_edges <-map_edges %>%
8    mutate(width = total.items/3)
9
10 visNetwork(map_nodes,map_edges,width = "100%")  %>%
11    visEdges(arrows = "middle") %>%
12   visEdges(color = "blue") %>%
13       visIgraphLayout(layout = "layout.star")
14
```

Figure 4.12: R code to create network visualisation

The network visualisation from running this code is shown in Figure 4.13.
Each of the 26 county nodes are represented as circles with Dublin located in the middle. The network shows some interesting patterns of domestic holidaymakers e.g. Kerry has a large number of residents that holiday in Cork, Limerick and Dublin while Galway residents record a large number of holidays in Dublin. The user can also interact with the visualisation by selecting and moving nodes which can allow for clearer representation for particular nodes of interest.

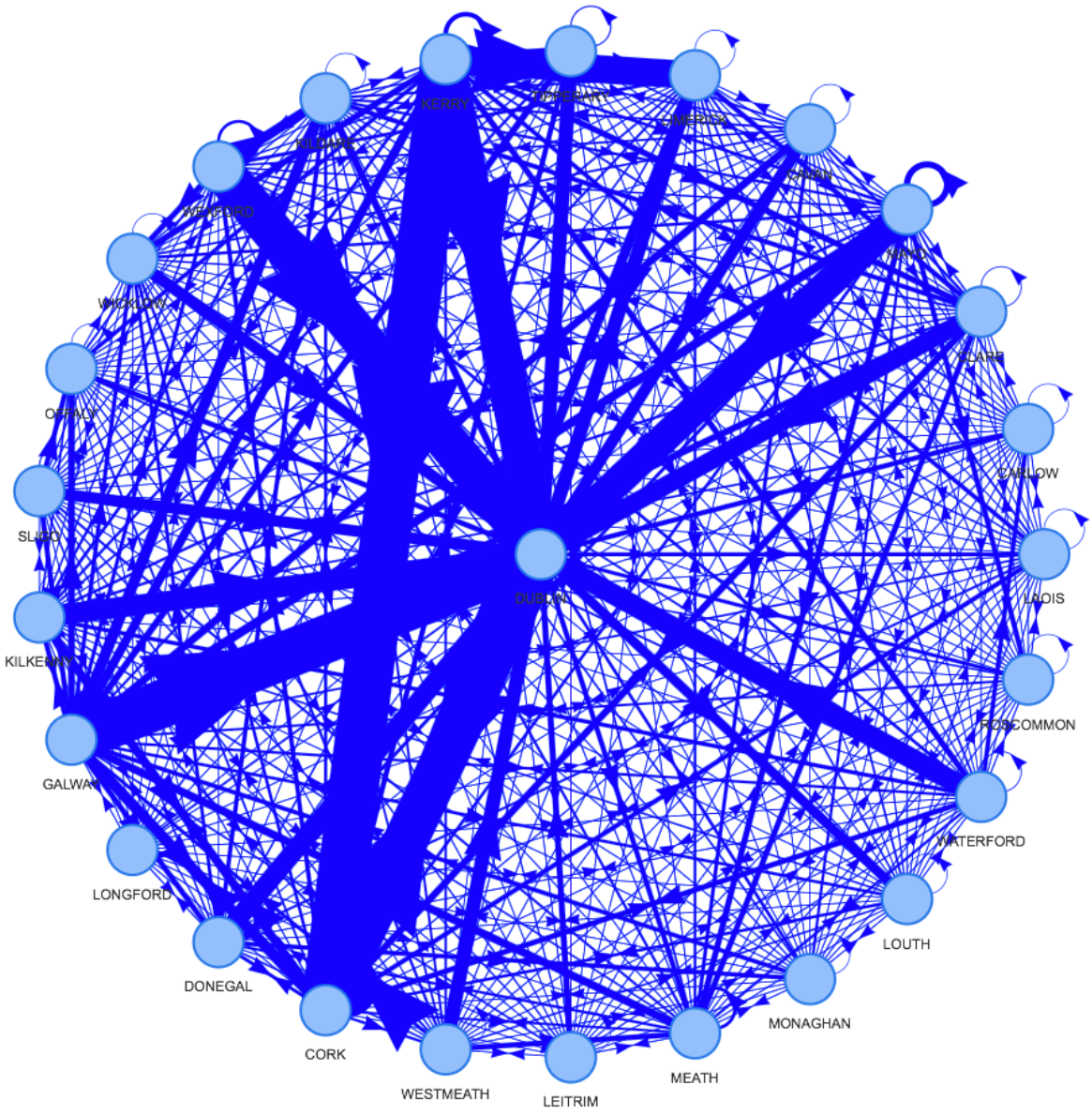


Figure 4.12 Network visualisation of domestic holidaymakers

Exercise

Create a network graph of the kind shown in Figure 4.12 by sourcing a data set containing origin and destination data.