

Sankay Plots of Workshop Comments

Curtis C. Bohlen

2022-12-07

Contents

Introduction	2
Load Packages	2
Load Data	3
Sankay Plot Methods	4
Data Wrangling	4
Initial Graphic	5
Custom Colors	6
Simplified Graphic	9
Functions for Generating the Graphics	11
Define Color String	11
Assemble the Nodes and Links Data Frames	12
Draw Sankey Plot	13
From User to Data Types	14
Full Data	14
Simplified Plot	16
From User to Model Extnsions	18
Full Data	18
Simplified Plot	20
From Data Requests to Time Domain	22
Full Data	22
Simplified Plot	24

From Time Domain to Data Requests	26
Full Data	26
Simplified Plot	28

Introduction

CBEP recently received a grant from NSF’s CIVIC Innovation Challenge to work on developing hydrodynamic models that address community needs in Portland Harbor. As part of the project, CBEP hosted three community workshops in November of 2022.

Facilitators produced both “live” notes during the meeting – visible to all on a screen at the front of the meeting room – and detailed meeting transcripts. CBEP staff then reviewed those notes paragraph by paragraph, and coded each paragraph in terms of six characteristics:

- Potential users and uses of hydrodynamic models,
- Data or information needs identified by community members,
- Implied extensions of the initial Casco Bay Model required to fully address those data needs, and
- Ideas for improving communications of model results (e.g., communications channels and user interface design),
- Specifications for model performance or capabilities such as resolution, geographic coverage or ability to conduct simulations.
- Suggestions about monitoring or data collection that could improve information availability.

If a paragraph or live note included something relevant to one or more of these categories, we summarized the related idea, and then assigned each paragraph or comment to categories. In this way we can look at what ideas were expressed most commonly during the workshops.

Of course, not all paragraphs include information related to each of the five types of information, so there is not a perfect one-to-one correspondence between categories.

In this R Notebook, I explore these data principally by looking at cross-correlations among ideas in the different categories.

Load Packages

```
library(tidyverse)
#> -- Attaching packages ----- tidyverse 1.3.2 --
#> v ggplot2 3.4.0      v purrr 0.3.5
#> v tibble 3.1.8       v dplyr 1.0.10
#> v tidyr 1.2.1        v stringr 1.5.0
#> v readr 2.1.3       v forcats 0.5.2
#> -- Conflicts ----- tidyverse_conflicts() --
#> x dplyr::filter() masks stats::filter()
#> x dplyr::lag()     masks stats::lag()
library(readxl)
library(networkD3) #an alternative might be the riverplot package

theme_set(theme_classic())
```

Load Data

```
the_data <- read_excel("Export_Data_Query.xlsx" ) %>%
  mutate(ID = as.integer(ID))
head(the_data)
#> # A tibble: 6 x 15
#>   ID Category Day Comment User_~1 Inter~2 Inter~3 Data_~4 Data_~5 Exten~6
#>   <int> <chr> <chr> <chr> <chr> <chr> <chr> <chr> <dbl> <chr>
#> 1 1 Live Comm~ Day ~ How ca~ Shore ~ <NA> <NA> <NA> NA Waters~
#> 2 2 Live Comm~ Day ~ Use of~ Marine~ <NA> <NA> Waves 4 <NA>
#> 3 2 Live Comm~ Day ~ Use of~ Shore ~ <NA> <NA> Waves 4 <NA>
#> 4 3 Live Comm~ Day ~ MS4 pr~ Water ~ <NA> <NA> <NA> NA Waters~
#> 5 4 Live Comm~ Day ~ Consid~ <NA> <NA> <NA> <NA> <NA>
#> 6 5 Live Comm~ Day ~ Unders~ Shore ~ <NA> <NA> <NA> NA Waters~
#> # ... with 5 more variables: Extension_Timing <dbl>, Monitoring_Category <chr>,
#> # Monitoring_Data_Group <chr>, Performance_Apply <chr>,
#> # Performance_Criterion <chr>, and abbreviated variable names
#> # 1: User_Category, 2: Interface_Category, 3: Interface_Group, 4: Data_Group,
#> # 5: Data_Timing, 6: Extension_Category
```

Our coding was generated in a somewhat sloppy Access database, and because of the way SQL works, it is easier to replace numerical values for some groups here, in R, rather than before we exported the data from Access. I read in the dictionaries here.

```
timing_table <- read_excel("Timing Category.xlsx",
  col_types = c("numeric", "text", "text"))
```

And finally I correct the data table to all text entries.

```
the_data <- the_data %>%
  mutate(Data_Timing = timing_table$Timing[match(Data_Timing, timing_table$ID)],
    Extension_Timing = timing_table$Timing[match(Extension_Timing,
      timing_table$ID)])
```

#A Warning about Uniqueness We have to be careful here, because each note or comment can be represented in this data table multiple times. Each paragraph in the meeting transcript might imply several different users, for example. But if there are multiple users and multiple data types, the records got duplicated (in part) in the SQL query. So for any analysis, we need to test for uniqueness of the data. always

We actually have over 400 records, built out of just over 200 unique comments.

```
cat("All rows in the data:\t\t")
#> All rows in the data:
nrow(the_data)
#> [1] 452

cat("Unique comments reviewed:\t")
#> Unique comments reviewed:
the_data %>%
  select(ID) %>%
  unique() %>%
  nrow()
#> [1] 206
```

Sankay Plot Methods

We need to create a network data set, which has to identify links between nodes by **Source**, **Target** and **Value**. The `sankeyNetwork()` function from the `networkD3` package on which we rely wants the data in a very specific format.

- It wants to define NODES via a data frame of node labels (and, optionally, node groups that can be used to assign colors to nodes).
- It wants to define links via a data frame containing NUMERIC (integer?) values identifying nodes. The data frame can also include data on link groups for assigning colors to links.
- The integers defining each node must match the order of those nodes in the nodes data frame.

Data Wrangling

For clarity, I break the data wrangling up into discrete steps. That simplifies both explanation and checking intermediate results.

First, we select the user and data information.

We are headed for a display that shown User or Use groups on the left, and Data groups on the right. Both User and Data classifications included a category called simply “Other”, for comments that clearly articulated a data request that I could not figure out how to categorize.

Sankey plots need not be unidirectional, so logically, nodes are neither sources nor targets. If I use the same name for both the “User Other” and the “Data Other” categories, the package sees them as the same category, which has unintended effects on plot geometry (since we can then have “Other” as either or both a source or a target).

TO avoid that problem, we have to edit the category names so the two “Other” categories don’t look equivalent.

Finally, we convert the (string) data to factors, and sort them in decreasing order of frequency, which may help with later graphic design.

```
tmp <- the_data %>%
  select(ID, User_Category, Data_Group) %>%
  filter(! is.na(User_Category)) %>%
  filter(! is.na(Data_Group)) %>%
  mutate(User_Category = if_else(User_Category == 'Other',
                                'Other User', User_Category),
         Data_Group = if_else(Data_Group == 'Other',
                              'Other Data',Data_Group)) %>%
  mutate(User_Category = if_else(User_Category == 'Water Quality',
                                'Water Quality User', User_Category),
         Data_Group = if_else(Data_Group == 'Water Quality',
                              'Water Quality Data',Data_Group)) %>%
  mutate(User_Category = fct_infreq(User_Category),
         Data_Group = fct_infreq(Data_Group)) %>%
  unique()
```

We next calculate the number of links between each pair of source and target nodes.

```
tmp2 <- tmp %>%
  group_by(User_Category, Data_Group) %>%
  summarize(weight = n(),
            .groups = 'drop')
```

I add a `groups` variable, to allow me to control the color of the source and target nodes in the diagram.

```
nodes <- tibble(node_name = c(unique(tmp2$User_Category),
                              unique(tmp2$Data_Group)),
               groups = c(rep("A", length(unique(tmp2$User_Category))),
                          rep("B", length(unique(c(tmp2$Data_Group))))))
```

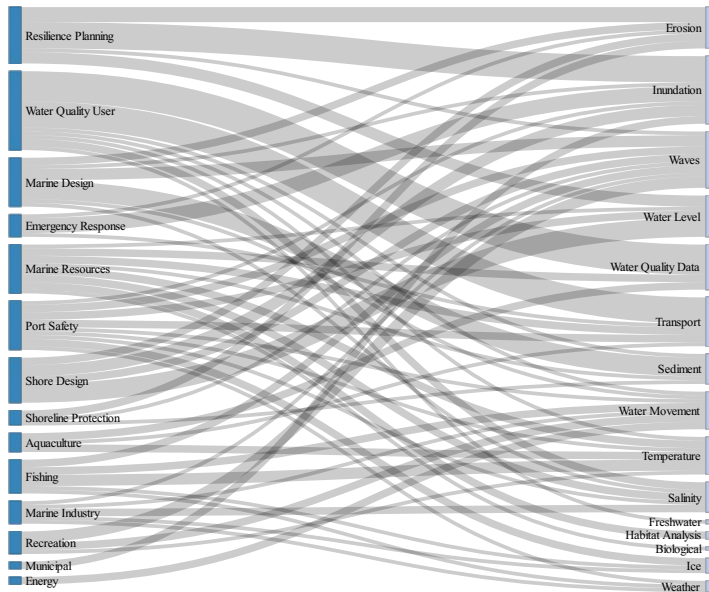
Then we have to convert the character string values in our links data frame (`tmp2`) to zero-valued numerical IDs, as required by the `networkD3` package.

`match()` here is a quick way of implicitly extracting the row number of a matching record from each vector (column) in the `tmp2` data frame.

```
tmp3 <- tmp2 %>%
  mutate(link_group = User_Category,
         User_Category = match(User_Category, nodes$node_name) - 1,
         Data_Group = match(Data_Group, nodes$node_name) - 1)
```

Initial Graphic

```
plt <- sankeyNetwork(Links = tmp3,
                    Source = "User_Category",
                    Target = "Data_Group",
                    Value = "weight",
                    Nodes = nodes,
                    NodeID = "node_name",
                    NodeGroup = 'groups',
                    height = 800,
                    fontSize = 16,
                    iterations = 0)
#> Links is a tbl_df. Converting to a plain data frame.
#> Nodes is a tbl_df. Converting to a plain data frame.
plt
```



Custom Colors

It might be nice to color the nodes or links according to categories. This is surprisingly difficult, because the package uses one color scale for both nodes and links.

I found it hard to figure out how colors were ordered, and then found it hard to assemble the correct color specification as required for **network3D**.

I generate new node group names. For the “Source” nodes, we use the name of the user category. For the “Target” nodes, we use an arbitrary string, here “B”.

```
users <- as.character(levels(tmp2$User_Category))
types <- as.character(levels(tmp2$Data_Group))

nodes <- tibble(node_name = c(users, types),
                groups = c(users,
                           rep("B", length(unique(c(tmp2$Data_Group))))))
```

We want the links to match the color of the source nodes, so we specify matching color group names for links based on the source (here, left hand) nodes.

```
tmp3 <- tmp2 %>%
  mutate(link_group = as.character(User_Category),
         User_Category = match(User_Category, nodes$node_name) - 1,
         Data_Group = match(Data_Group, nodes$node_name) - 1)
```

Now I have to define my color palette. This is more work than it should be because you have to encapsulate a D3 graphics command in a string so that the `sankeyNetwork()` function can pass it on to the underlying Java / D3 code.

The command requires you to define the names and the colors together as the domain and range of the palette.

I randomize color order to minimize conflict with adjacent nodes. I specify the last color – for the right hand nodes – as a medium gray.

```
domain = c(users, "B")
(grps <- length(domain))
#> [1] 15
range <- sample(hcl.colors(15, palette = "viridis"), grps - 1, replace = FALSE)
range <- c(range, "#9090c0")
range
#> [1] "#471D67" "#006290" "#25C771" "#1E4D85" "#73D25B" "#00BA82" "#FDE333"
#> [8] "#007796" "#3C3777" "#008A98" "#4B0055" "#D4E02D" "#00AC8E" "#009B95"
#> [15] "#9090c0"
```

In the next step I build a data frame containing the domain and range. It is probably not strictly necessary, but it did avoid some odd artifacts with escaped quotes.

```
cols <- tibble(d = domain, r = range)
cols
#> # A tibble: 15 x 2
#>   d               r
#>   <chr>          <chr>
#> 1 Resilience Planning #471D67
#> 2 Water Quality User   #006290
#> 3 Marine Design        #25C771
#> 4 Emergency Response   #1E4D85
#> 5 Marine Resources     #73D25B
#> 6 Port Safety          #00BA82
#> 7 Shore Design         #FDE333
#> 8 Shoreline Protection #007796
#> 9 Aquaculture          #3C3777
#> 10 Fishing             #008A98
#> 11 Marine Industry     #4B0055
#> 12 Recreation          #D4E02D
#> 13 Municipal           #00AC8E
#> 14 Energy              #009B95
#> 15 B                   #9090c0
```

The colors are assembled as a string, which takes some fussy work to do programmatically using `paste()`. Basically, I concatenate the text needed for the D3 function call with the lists of domain and range I just built.

```

the_colors <- paste0('d3.scaleOrdinal() .domain(['',
  paste(cols$d, collapse='', ''),
  '']) .range(['',
    paste(cols$r, collapse='', ''),
    ''])')
cat(the_colors)
#> d3.scaleOrdinal() .domain(["Resilience Planning", "Water Quality User", "Marine Design", "Emergency

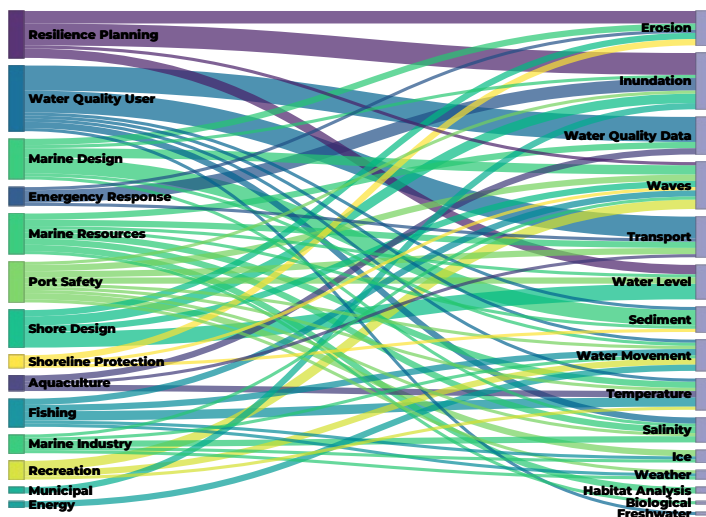
```

Revised Graphic

```

plt <- sankeyNetwork(Links = tmp3,
  Source = "User_Category",
  Target = "Data_Group",
  Value = "weight",
  LinkGroup = "link_group",
  NodeGroup = "groups",
  Nodes = nodes,
  NodeID = "node_name",
  colourScale = the_colors,
  nodeWidth = 20,
  nodePadding = 10,
  height = 700,
  fontSize = 16,
  fontFamily = 'Montserrat ExtraBold',
  iterations = 0)
#> Links is a tbl_df. Converting to a plain data frame.
#> Nodes is a tbl_df. Converting to a plain data frame.
plt

```

Simplified Graphic

That graphic looks like a bowl of spaghetti. Most connections represent only one mention during the workshops. Since the data is based on both the live notes and the later transcript, one mention could show up either once or twice in my classification of the notes.

So, what I want to do is simplify the data by hiding all those “once or twice” links. It turns out if you drop rare links, a few nodes are never linked, so I want to drop them too.

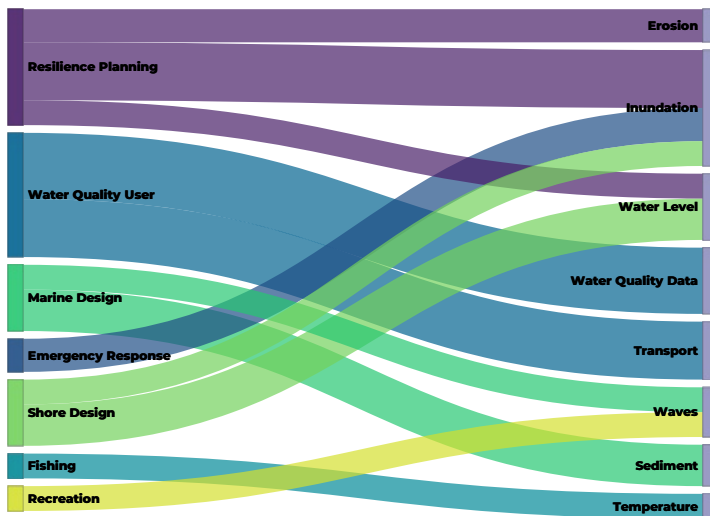
```
tmp4 <- tmp2 %>%
  filter(weight > 2)
```

```
types <- unique(as.character(tmp4$Data_Group))
users <- unique(as.character(tmp4$User_Category))

nodes <- tibble(node_name = c(users, types),
  groups = c(users,
    rep("B", length(types))))
```

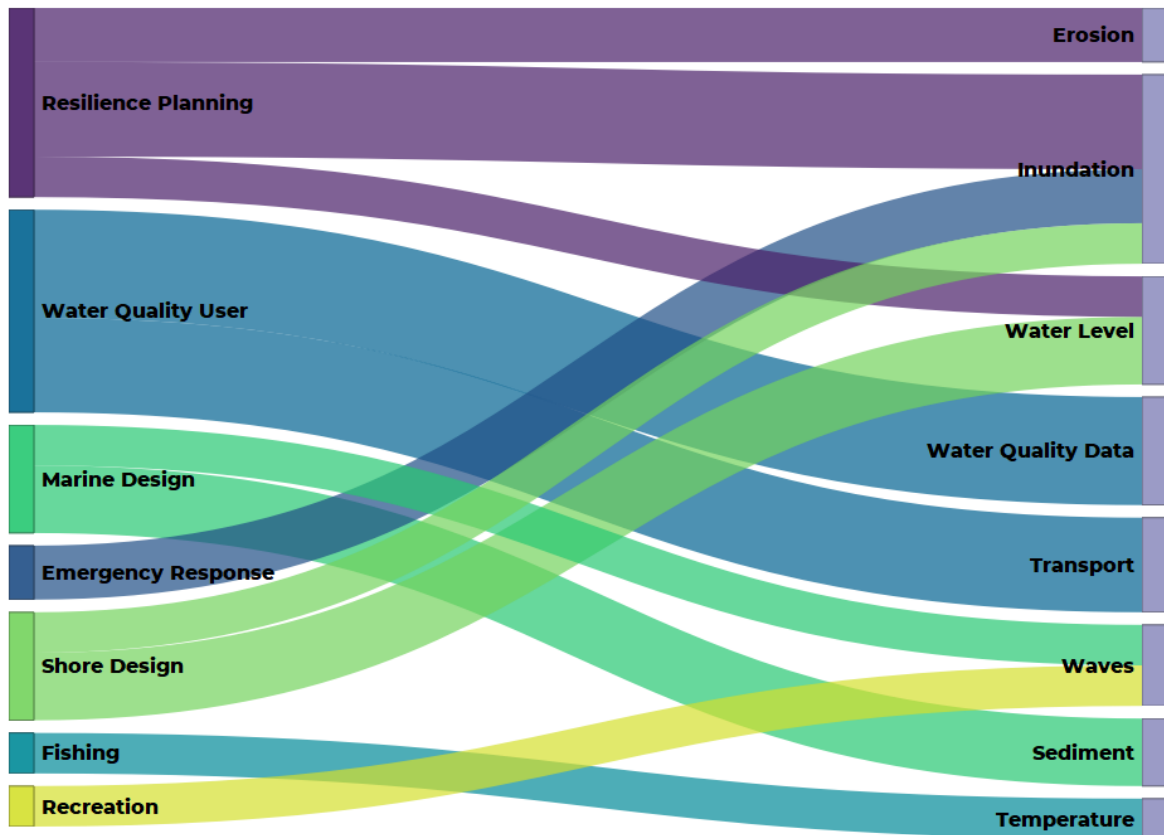
```
tmp5 <- tmp4 %>%
  mutate(link_group = as.character(User_Category),
    User_Category = match(User_Category, nodes$node_name) - 1,
    Data_Group = match(Data_Group, nodes$node_name) - 1)
```

```
plt <- sankeyNetwork(Links = tmp5,
  Source = "User_Category",
  Target = "Data_Group",
  Value = "weight",
  LinkGroup = "link_group",
  NodeGroup = "groups",
  Nodes = nodes,
  NodeID = "node_name",
  colourScale = the_colors,
  nodeWidth = 20,
  nodePadding = 10,
  height = 700,
  fontSize = 16,
  fontFamily = 'Montserrat ExtraBold',
  iterations = 0)
#> Links is a tbl_df. Converting to a plain data frame.
#> Nodes is a tbl_df. Converting to a plain data frame.
plt
```



```
saveNetwork(plt, file = 'users and data simpler.html', selfcontained = TRUE)
webshot::webshot('users and data simpler.html', 'users and data simpler.png')
```

'''



Functions for Generating the Graphics

I develop three functions to encapsulate all of that logic. All three are local utility functions, so I eschew error checking. The three functions are:

1. A function to assemble the D3 color function call
2. A function to convert our source data frames, with a column of source and target names, into the format required by `sankeyNetwork()`.
3. A function that actually generates the Sankey Plot (and calls the other two functions).

Define Color String

This function could be generalized to allow automated dropping of weak links.

```
d3_colors <- function(left_names, right_name, final_color = NULL) {
  #left_names is a vector of string node labels used for the source nodes.
  #right.name is the name for the (single) target node group.
  #final_color is a string defining a color for the right (target) nodes.
```

```

domain = c(left_names, right_name)
grps <- length(left_names) + 1
#browser()
if(! is.null(final_color)) {
  range <- sample(hcl.colors(grps-1, palette = "viridis"), grps - 1, replace = FALSE)
  range <- c(range, final_color)
}
else
  #print(grps)
  range <- sample(hcl.colors(grps, palette = "viridis"), grps, replace = FALSE)
cols <- tibble(d = domain, r = range)
the_colors <- paste0('d3.scaleOrdinal() .domain(['',
  paste(cols$d, collapse='', ''),
  '']) .range(['',
  paste(cols$r, collapse='', ''),
  ''])')
return(the_colors)
}

```

```

d3_colors(c("cat", "dog", "squirrel"), "foods", "#ffffff")
#> [1] "d3.scaleOrdinal() .domain(['cat\", \"dog\", \"squirrel\", \"foods\"]) .range(['#FDE333\", \"#

```

Assemble the Nodes and Links Data Frames

We have to be able to filter the data to remove rare I'm going to forego error checking here and keep this as simple as possible...

```

assemble_frames <- function(.dat, .left, .right,
  right_name = 'right') {
  # .data is the RAW data with left-right string pairs (in two variables) that
  # show a link between source (.left) and target (.right).

  grouped <- .dat %>%
    group_by({{ .left }}, {{ .right }}) %>%
    summarize(weight = n(),
      .groups = 'drop')

  # browser()
  # there is probably a more efficient way to extract labels, that avoids
  # building an unnecessary data frame, but this works, and lets me use tidyverse
  # indirection....

  labs <- grouped %>%
    mutate(left = factor({{ .left }}),
      right = factor({{ .right }}))

  #I can ignore order here because the Sankey function does....
  # By calling `factor()` here, it re-levels and drops empty categories.
  left = levels(factor(labs$left))
  right = levels(factor(labs$right))
  rm(labs)
}

```

```

nodes <- tibble(node_name = c(left, right),
               groups = c(left,
                           rep(right_name, length(right))))

links <- grouped %>%
  mutate(link_group = as.character({{.left}}),
         "{{.left}}" := match({{.left}}, nodes$node_name) - 1,
         "{{.right}}" := match({{.right}}, nodes$node_name) - 1)

return(list(Nodes = as.data.frame(nodes), Links = as.data.frame(links)))
}

```

```

test <- tmp %>%
  assemble_frames(User_Category, Data_Group, 'bicycle')

```

```

test$Nodes[10:20,]
#>      node_name      groups
#> 10      Fishing      Fishing
#> 11 Marine Industry Marine Industry
#> 12      Recreation      Recreation
#> 13      Municipal      Municipal
#> 14      Energy      Energy
#> 15      Erosion      bicycle
#> 16      Inundation      bicycle
#> 17 Water Quality Data      bicycle
#> 18      Waves      bicycle
#> 19      Transport      bicycle
#> 20      Water Level      bicycle

```

```

head(test$Links,5)
#>   User_Category Data_Group weight      link_group
#> 1             0         14      4 Resilience Planning
#> 2             0         15      7 Resilience Planning
#> 3             0         17      1 Resilience Planning
#> 4             0         19      3 Resilience Planning
#> 5             1         16      8 Water Quality User

```

Draw Sankey Plot

This encapsulates data frame preparation, color assignment and my selected plot characteristics.

```

my_sankey <- function(.dat, .left, .right,
                     final_color = NULL, drop_below = NULL) {

  right_name <- 'right'
  left_str <- as.character(ensym(.left))
  right_str <- as.character(ensym(.right))

  if (! is.null(drop_below)) {
    .dat <- .dat %>%
      group_by({{.left}}, {{.right}}) %>%
      mutate(links = n()) %>%

```

```

    filter(links >= drop_below)%>%
    select(-links)
  }

my_data <- assemble_frames(.dat, {{.left}}, {{.right}},
                           right_name = right_name)

left <- my_data$Links %>%
  mutate(left = as.character(link_group)) %>%
  pull(left)
left <- unique(as.character(left))

the_colors <- d3_colors(left, right_name = right_name,
                        final_color = final_color)
#browser()
the_graphic <- sankeyNetwork(Links = my_data$Links,
                             Source = left_str,
                             Target = right_str,
                             Value = "weight",
                             LinkGroup = "link_group",
                             NodeGroup = "groups",
                             Nodes = my_data$Nodes,
                             NodeID = "node_name",
                             colourScale = the_colors,
                             nodeWidth = 20,
                             nodePadding = 10,
                             height = 700,
                             fontSize = 16,
                             fontFamily = 'Montserrat ExtraBold',
                             iterations = 0)

return(the_graphic)
}

```

From User to Data Types

Full Data

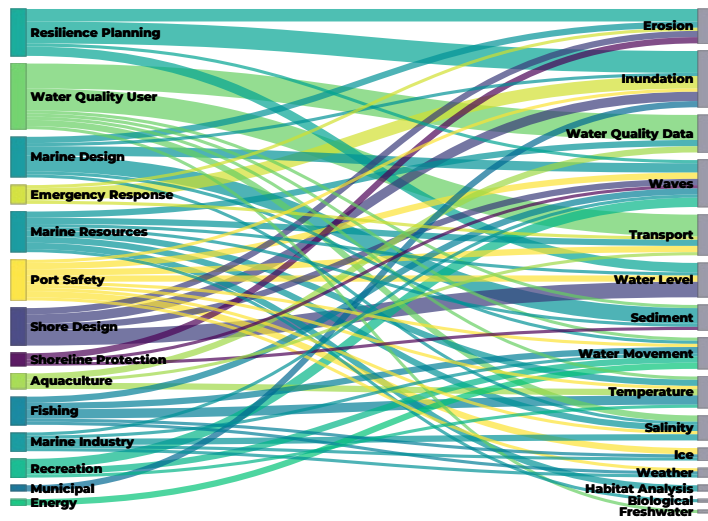
Based on data organized previously.

```

plt <- tmp %>%
  my_sankey(User_Category, Data_Group, final_color = "#9090a0")

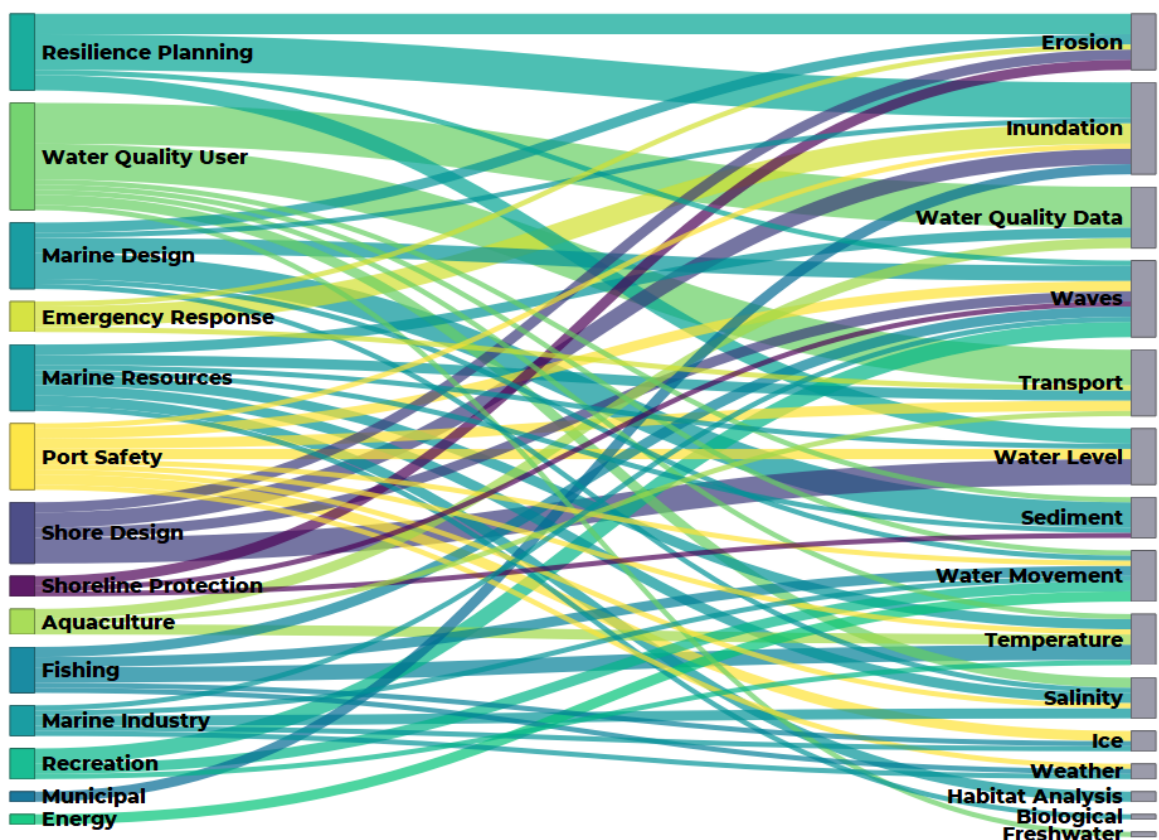
```

```
plt
```



```
saveNetwork(plt, file = 'users and data.html', selfcontained = TRUE)
webshot::webshot('users and data.html', 'users and data.png')
```

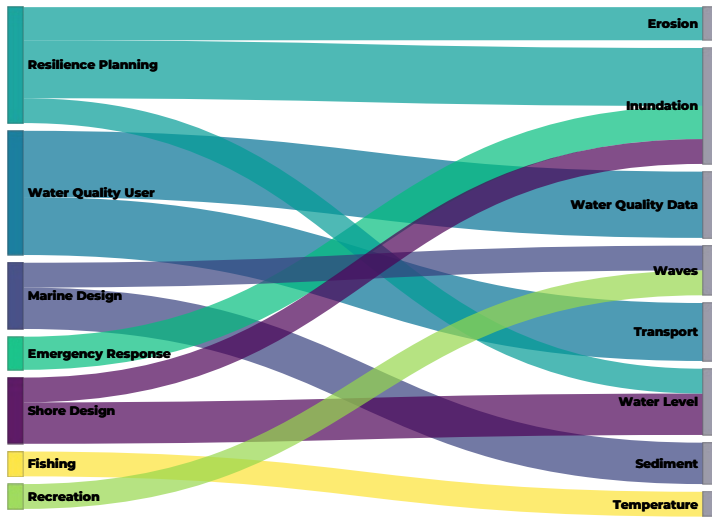
'''



Simplified Plot

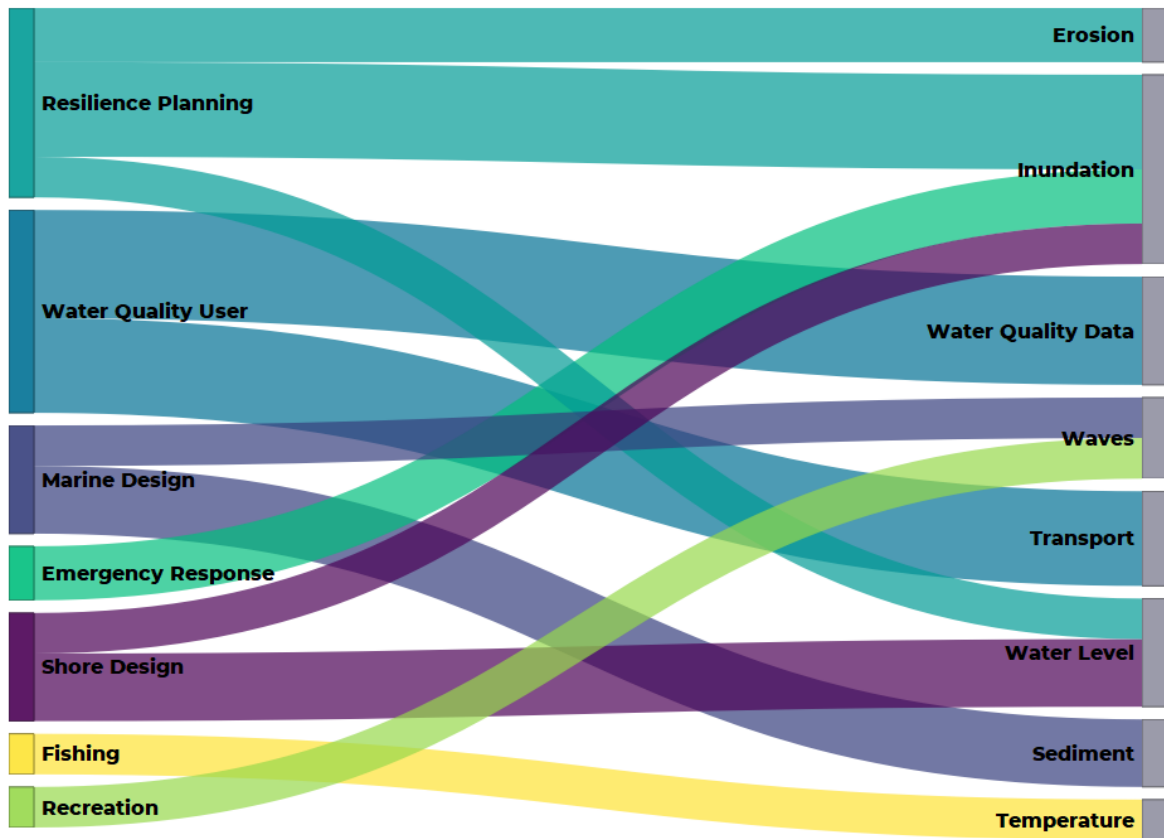
```
plt <- tmp %>%
  my_sankey(User_Category, Data_Group, final_color = "#9090a0", drop_below = 3)
```

```
plt
```

```
saveNetwork(plt, file = 'users and data simple.html', selfcontained = TRUE)
webshot::webshot('users and data simple.html', 'users and data simple.png')
```

'''

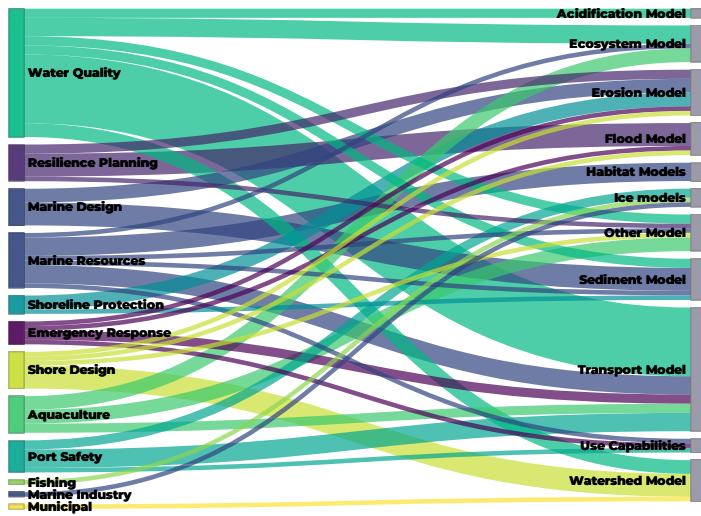


From User to Model Extnsions

Full Data

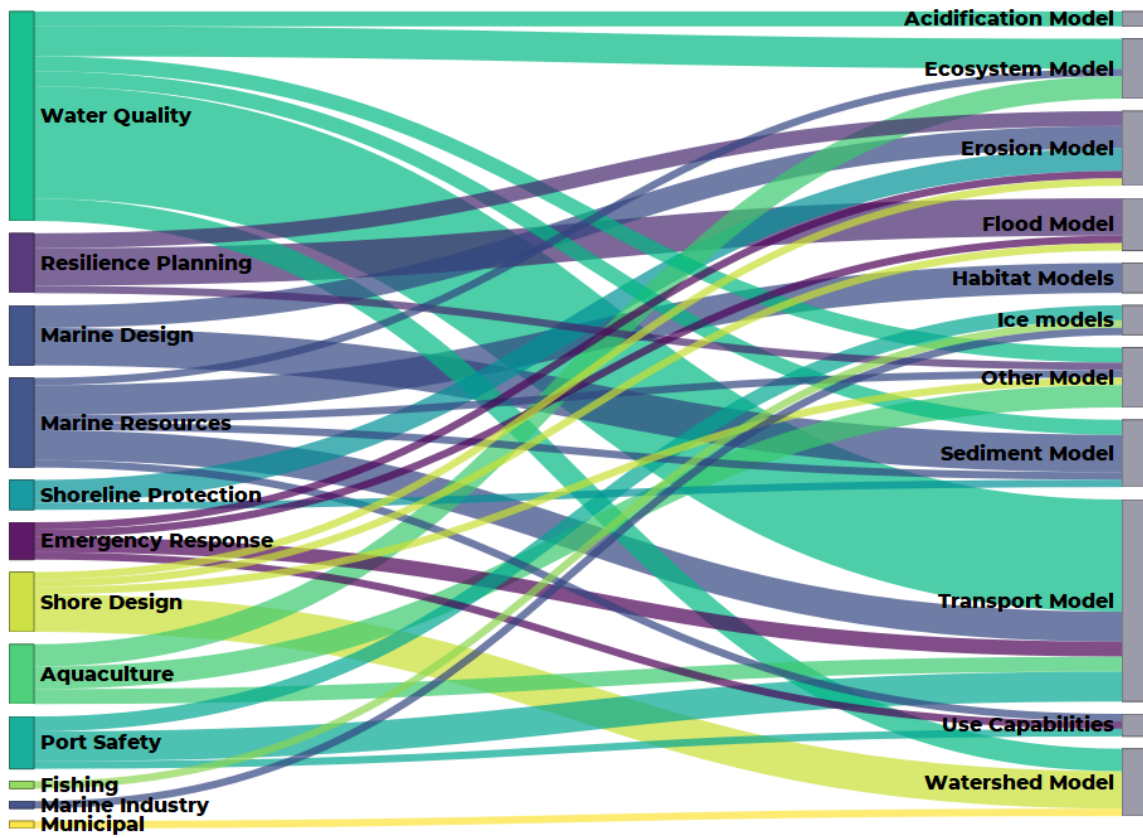
```
tmp <- the_data %>%
  select(ID, User_Category, Extension_Category) %>%
  filter(! is.na(User_Category)) %>%
  filter(! is.na(Extension_Category)) %>%
  mutate(User_Category = if_else(User_Category == 'Other',
                                'Other User', User_Category),
         Extension_Category = if_else(Extension_Category == 'Other',
                                      'Other Model', Extension_Category)) %>%
  mutate(User_Category = fct_infreq(User_Category),
         Data_Group = fct_infreq(Extension_Category)) %>%
  unique()
```

```
plt <- tmp %>%
  my_sankey(User_Category, Extension_Category, final_color = "#9090a0")
plt
```



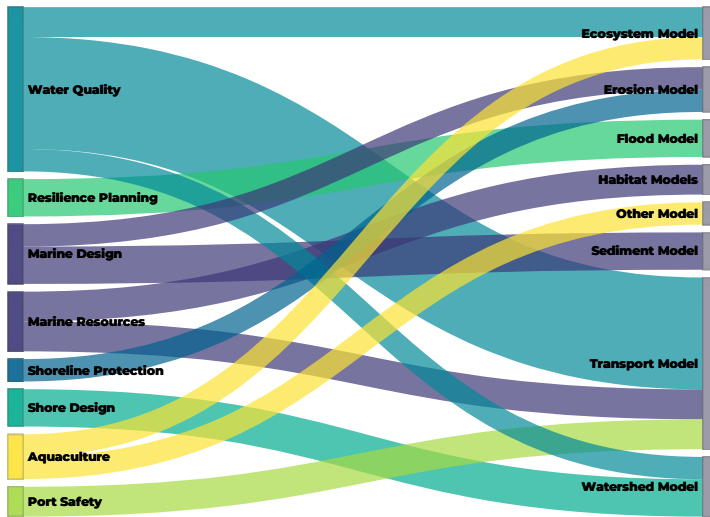
```
saveNetwork(plt, file = 'users and models.html', selfcontained = TRUE)
webshot::webshot('users and models.html', 'users and models.png')
```

'''



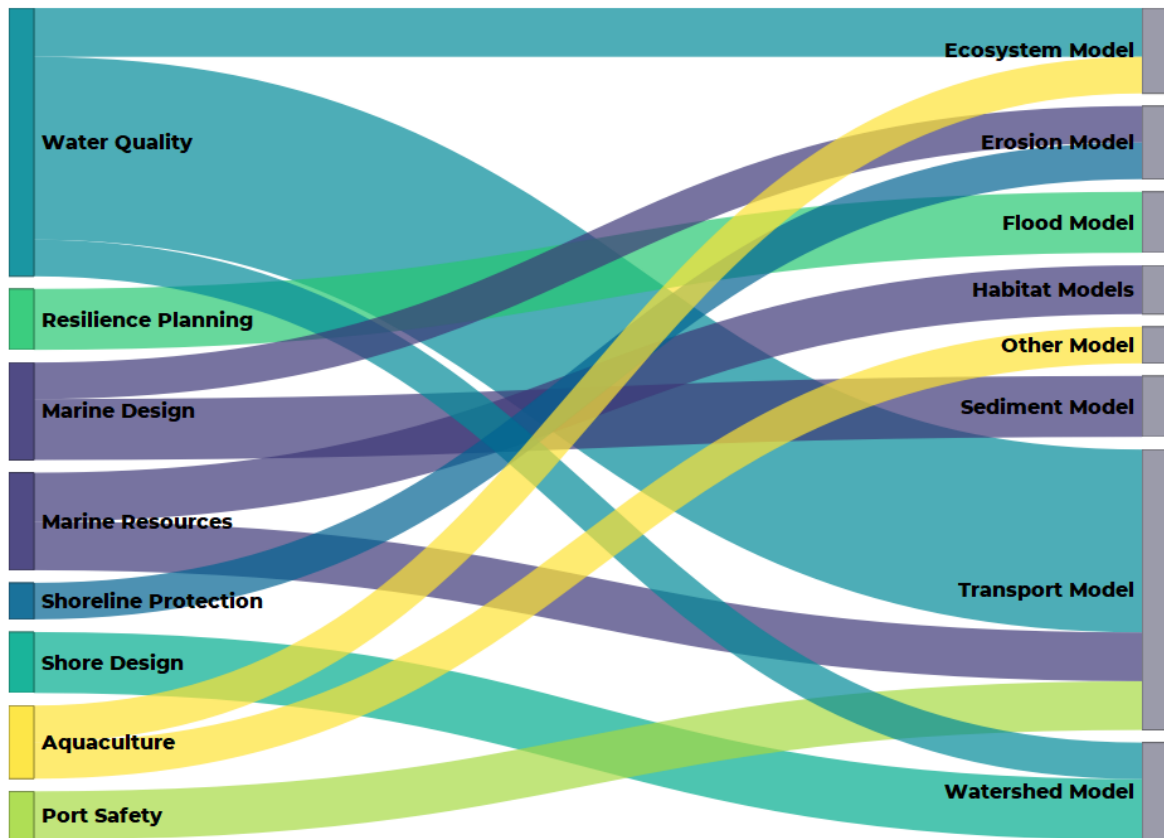
Simplified Plot

```
plt <- tmp %>%
  my_sankey(User_Category, Extension_Category, final_color = "#9090a0", drop_below = 3)
plt
```



```
saveNetwork(plt, file = 'users and models simple.html', selfcontained = TRUE)
webshot::webshot('users and models simple.html', 'users and data simple.png')
```

'''

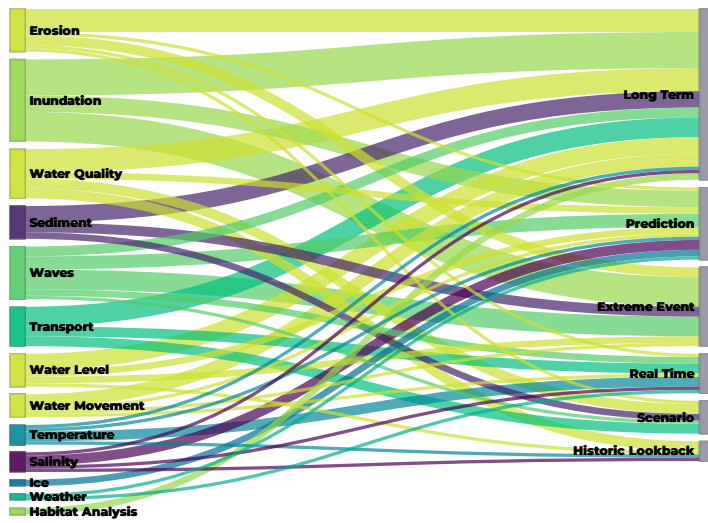


From Data Requests to Time Domain

Full Data

```
tmp <- the_data %>%
  select(ID, Data_Group, Data_Timing) %>%
  filter(! is.na(Data_Timing)) %>%
  filter(! Data_Timing == "Unclear or Unknown") %>%
  filter(! is.na(Data_Group)) %>%
  mutate(Data_Timing = fct_infreq(Data_Timing),
         Data_Group = fct_infreq(Data_Group)) %>%
  unique()
```

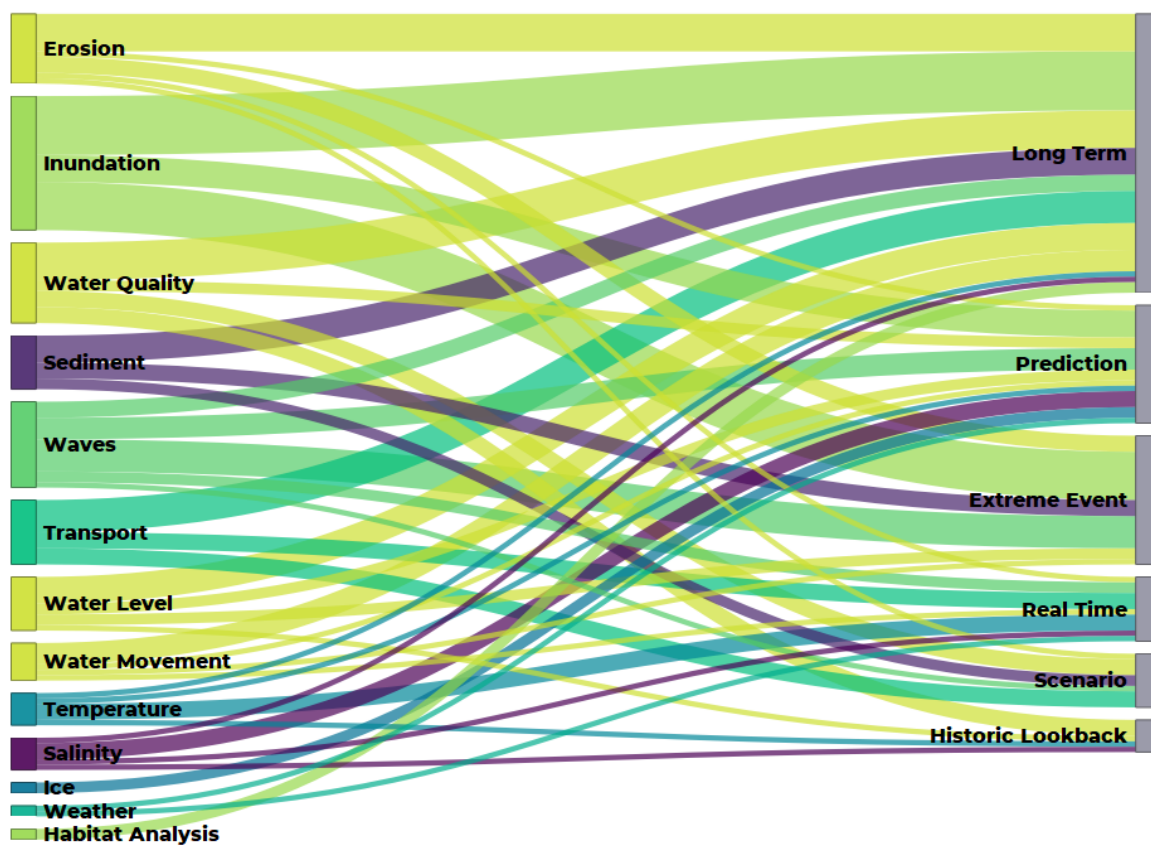
```
plt <- tmp %>%
  my_sankey(Data_Group, Data_Timing, final_color = "#9090a0")
plt
```



```
saveNetwork(plt, file = 'data to timing.html', selfcontained = TRUE)

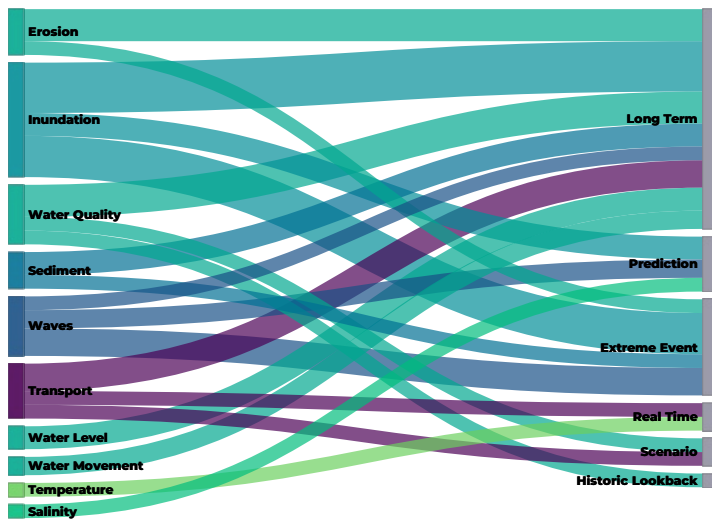
webshot::webshot('data to timing.html', 'data to timing.png')
```

'''



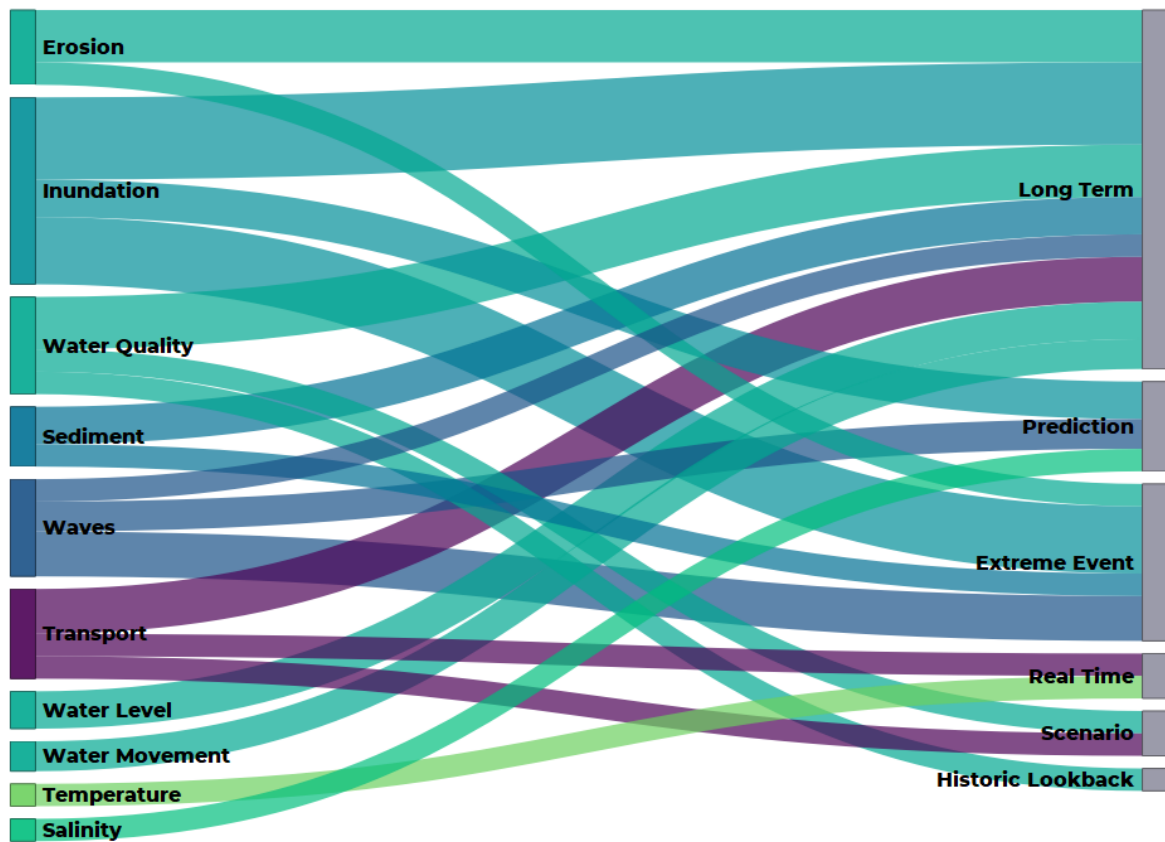
Simplified Plot

```
plt <- tmp %>%
  my_sankey(Data_Group, Data_Timing, final_color = "#9090a0", drop_below = 3)
plt
```

```
saveNetwork(plt, file = 'data to timing simple.html', selfcontained = TRUE)
webshot::webshot('data to timing simple.html', 'data to timing simple.png')
```

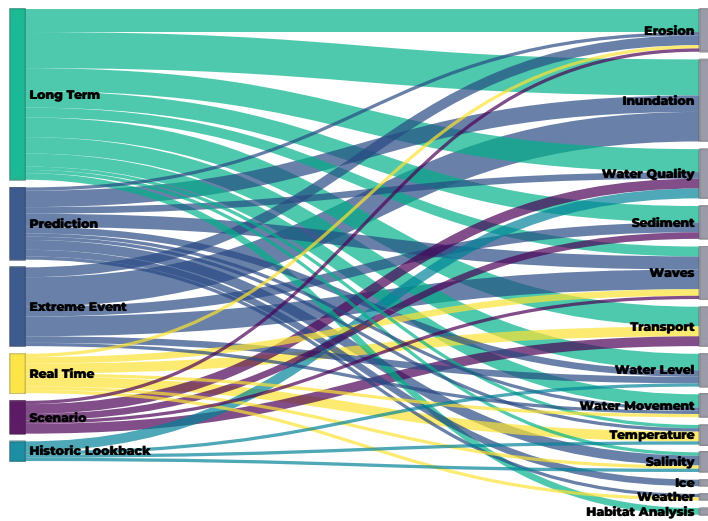
'''



From Time Domain to Data Requests

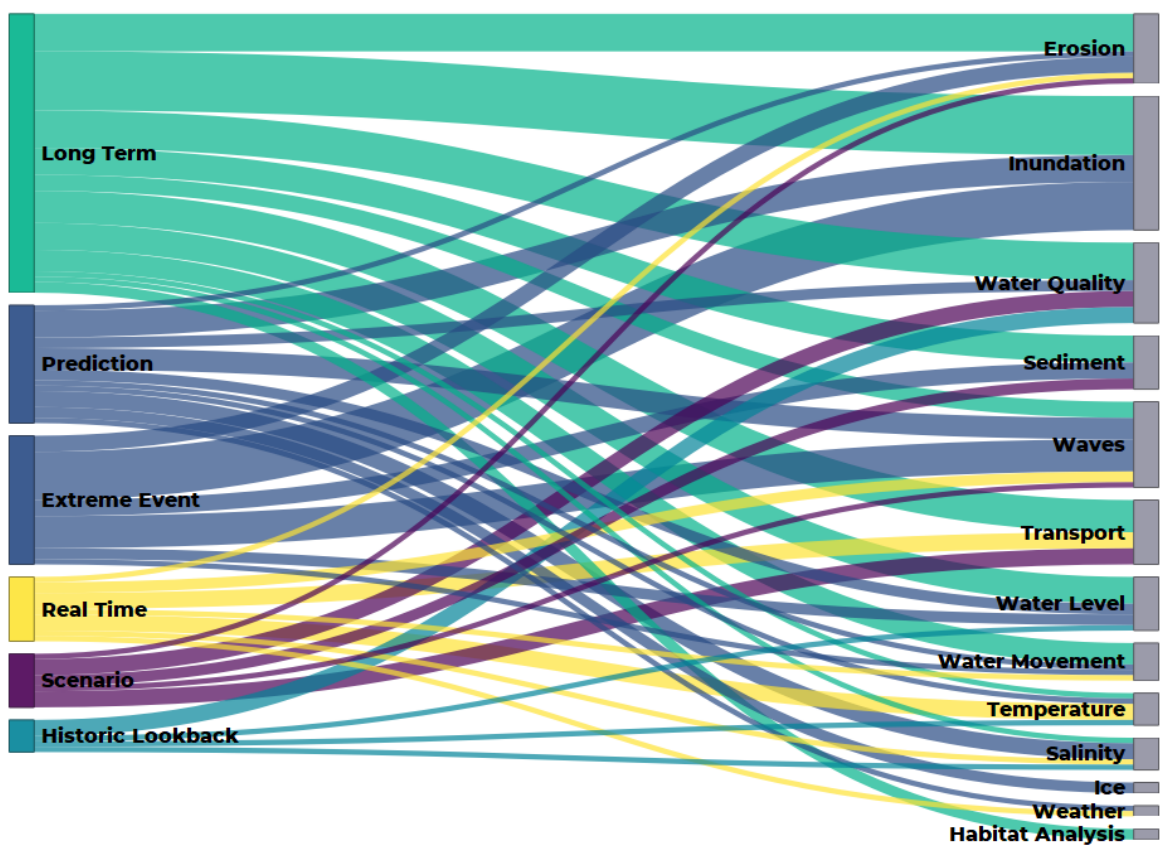
Full Data

```
plt <- tmp %>%
  my_sankey(Data_Timing, Data_Group, final_color = "#9090a0")
plt
```



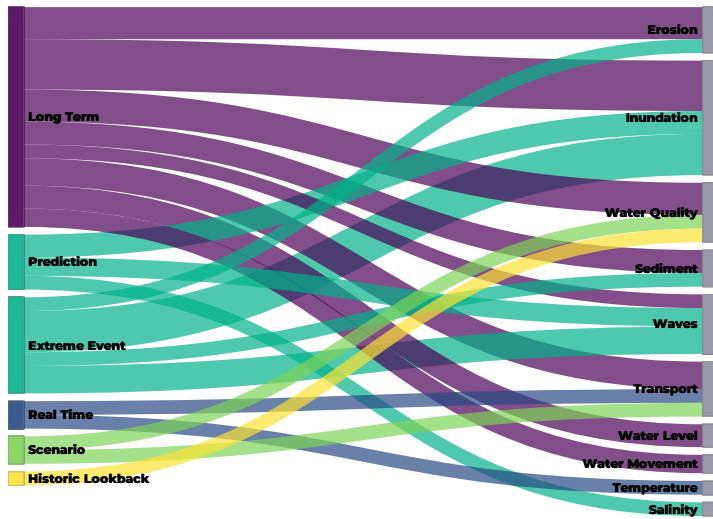
```
saveNetwork(plt, file = 'timing to data.html', selfcontained = TRUE)
webshot::webshot('timing to data.html', 'timing to data.png')
```

'''



Simplified Plot

```
plt <- tmp %>%
  my_sankey(Data_Timing, Data_Group, final_color = "#9090a0", drop_below = 3)
plt
```



```
saveNetwork(plt, file = 'timing to data simple.html', selfcontained = TRUE)
webshot::webshot('timing to data simple.html', 'timing to data simple.png')
```

