STAT 436 Midterm

- This exam lasts from 2:30 - 3:45 on March 23, 2023. There are 7 questions.
- This exam is closed notes and closed computer.
- You may use a 1-page cheat sheet (8.5 x 11in or A4 size). You may use both sides, but the cheat sheet must be handwritten.
- If you need extra space, you may write on the back of the page. Please indicate somewhere that your answer continues.
- The instructors can only answer clarifying questions during the exam.

Question	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Total
Score								
Possible	3	3	5	5	5	4	5	30

Q1

Circle whether the following statements about faceting and compound figures are TRUE or FALSE.

a. **TRUE** FALSE To arrange facets across separate rows, we can use `facet_grid(feature ~ .)`.

   If `feature` is a column name for a categorical variable, then `facet_grid(. ~ feature)` will split levels of `feature` across columns, while `facet_grid(feature ~ .)` will split levels across rows.

b. **TRUE** FALSE To improve readability in a compound figure, we should always align figure baselines.

   *To improve readability, the baselines **should** be aligned. Misalignment creates an unnecessary distraction when navigating the collection of figures.*

c. TRUE **FALSE** If `p1`, `p2`, and `p3` are `ggplot2` figures, then we can combine them horizontally (onto a single row) using the code,

```
library(patchwork)
p1 / p2 / p3
```

   To combine figures horizontally, we need to use the + symbol: We could use `p1 + p2 + p3`.

d. **TRUE** FALSE Each facet in a figure created with `facet_grid` shows a different subset of rows of the dataset appearing in the initial `ggplot()` call.

   Each facet shows a subset filtered down to one level of the faceted variable. The columns of the faceting panels indicate the level of the associated subset.

Q2

Circle whether the following statements about data tidying are TRUE or FALSE.

a. **TRUE** FALSE The output of the `mutate` function never changes the number of rows in the input `data.frame`.

   While `summarise` will reduce the number of rows in a `data.frame` to match the number of groups present, `mutate` will always preserve the number of rows. It potentially adds new columns if a new column name is included within the `mutate` argument.

b. TRUE **FALSE** If we want to tally the total number of gold medals won by each country in the olympics dataset,

```r
olympics <- read_csv("https://uwmadison.box.com/shared/static/rzw8h2x6dp5693gdbpgxaf2koqijo12l.csv") %
  select(Name, Country, Gold)
head(olympics, 4)
```

```
## # A tibble: 4 x 3
##   Name             Country                     Gold
##   <chr>            <chr>                      <dbl>
## 1 Lamusi A         People's Republic of China     0
## 2 A G Kruger       United States of America       0
## 3 Jamale Aarrass   France                         0
## 4 Abdelhak Aatakni Morocco                        0
```

then it will be sufficient to run,

```r
olympics %>%
  count(Country, Gold)
```

`count` only computes the number of unique combinations of the specified variables. To compute the total number of gold medals, we must sum across the `Gold` column using the `group_by + summarise` pattern. Specifically, we could use

```r
olympics %>%
  group_by(Country) %>%
  summarise(total = sum(Gold))
```

c. TRUE **FALSE** The dataset below gives country-level population and tuberculosis prevalence,

```r
head(table2, 4)
```

```
## # A tibble: 4 x 4
##   country      year type           count
##   <chr>       <dbl> <chr>          <dbl>
## 1 Afghanistan  1999 cases            745
## 2 Afghanistan  1999 population 19987071
## 3 Afghanistan  2000 cases           2666
## 4 Afghanistan  2000 population 20595360
```

If we are interested in computing the number of cases per 10000 people, then it will be sufficient to run,

```r
table2 %>%
  pivot_longer(c("type", "count")) %>%
  mutate(10000 * cases / population)
```

We would need to use `pivot_wider`, not `pivot_longer`, so that there are two new columns for the number of cases and population, respectively. Specifically, we could use the code below,

```r
table2 %>%
  pivot_wider(names_from = type, values_from = count) %>%
  mutate(rate = 10000 * cases / population)
```

d. **TRUE** FALSE A tidy dataset must store variables explicitly within a column rather than implicitly within column names.

A dataset with variable values stored in column names is not considered tidy. To make it tidy, we could use `pivot_longer` to create a new column containing the levels of that variable.

Q3

We will revisit the Pokemon dataset. For each part below, describe how the data would have to be transformed to support the associated visualization.

```r
pokemon <- read_csv("https://uwmadison.box.com/shared/static/hf5cmx3ew3ch0v6t0c2x56838er1lt2c.csv") %>%
  select(Name, type_1, Generation, Attack, Defense)
head(pokemon, 4)
```

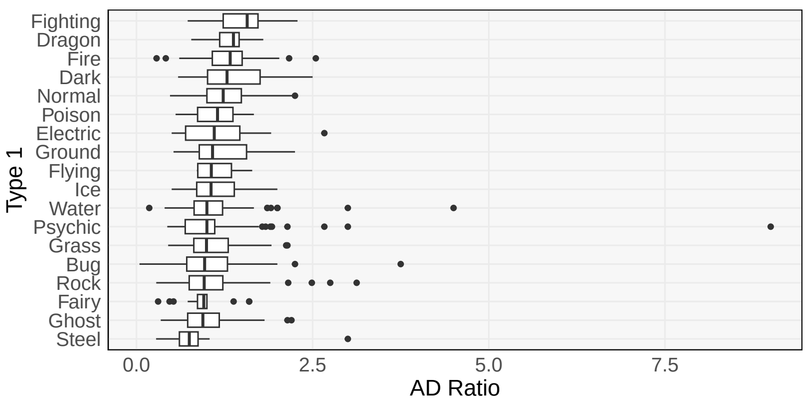
```
## # A tibble: 4 x 5
##   Name                  type_1 Generation Attack Defense
##   <chr>                 <chr>       <dbl>  <dbl>   <dbl>
## 1 Bulbasaur             Grass           1     49      49
## 2 Ivysaur               Grass           1     62      63
## 3 Venusaur              Grass           1     82      83
## 4 VenusaurMega Venusaur Grass           1    100     123
```

a. [1.25 points] Derive a new column containing the attack-to-defense ratio for each Pokemon, defined as $\frac{\text{Attack}}{\text{Defense}}$, as needed to generate the boxplot below.

```r
pokemon %>%
  mutate(ad_ratio = Attack / Defense)
```

Visualization code:

```r
pokemon %>%
  mutate(ad_ratio = Attack / Defense) %>%
  ggplot() +
  geom_boxplot(aes(ad_ratio, reorder(type_1, ad_ratio, median))) +
  labs(x = "AD Ratio", y = "Type 1")
```

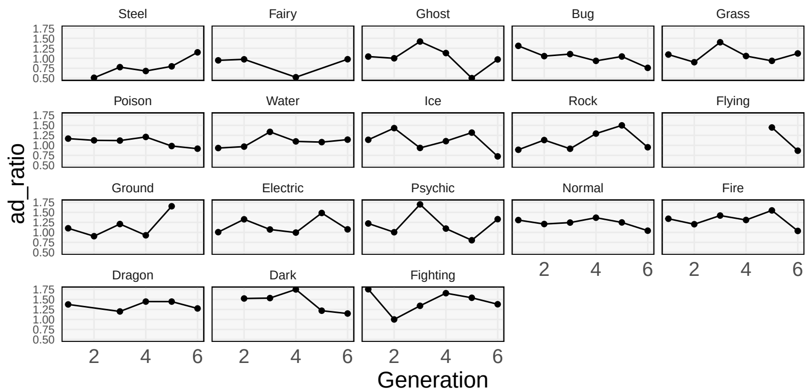


b. [1.25 points] Derive a dataset with the average attack-to-defense ratio for each `type_1` and `Generation` combination, as needed for the line plot below.

```r
pokemon %>%
  mutate(ad_ratio = Attack / Defense)
```

Visualization code:

```r
pokemon %>%
  group_by(type_1, Generation) %>%
  summarise(ad_ratio = mean(Attack / Defense)) %>%
  ggplot(aes(Generation, ad_ratio)) +
  geom_point() +
  geom_line(aes(group = type_1)) +
  facet_wrap(~ reorder(type_1, ad_ratio)) +
  theme(axis.text.y = element_text(size = 8))
```

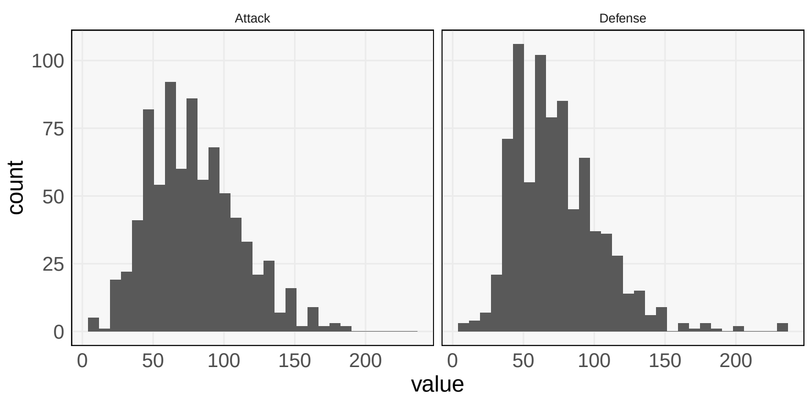


c. [1.25 points] Derive a dataset with the `Attack` and `Defense` variables within a single column which could be used as a faceting variable in the histogram below.

```
pokemon %>%
  select(Name, type_1, Attack, Defense) %>%
  pivot_longer(c("Attack", "Defense"), names_to = "Statistic")
```

Visualization code:

```
pokemon %>%
  select(Name, type_1, Attack, Defense) %>%
  pivot_longer(c("Attack", "Defense"), names_to = "Statistic") %>%
  ggplot() +
  geom_histogram(aes(value)) +
  facet_wrap(~ Statistic)
```



d. [1.25 points] Provide code for *one* of the three above visualizations. You may assume that the data have been appropriately reshaped.

The code is included for each of the figures above.

Q4

This problem asks you to study the reactive graph of a Shiny app built on the GCFN carbon emissions dataset. The app is built from two datasets, one of which includes time series across countries:

```
carbon_ts <- read_csv("https://raw.githubusercontent.com/krisrs1128/stat679_code/main/activities/week8/car
head(carbon_ts, 4)
```

```
## # A tibble: 4 x 4
##   country     `Country Code`  year emissions
##   <chr>       <chr>          <dbl>     <dbl>
## 1 Afghanistan AFG             1992         0
## 2 Afghanistan AFG             1993         0
```

```
## 3 Afghanistan AFG              1994         0
## 4 Afghanistan AFG              1995         0
```

and another which includes features derived from these series:

```r
carbon_features <- read_csv("https://raw.githubusercontent.com/krisrs1128/stat679_code/main/activities/wee
  select(country, trend_strength, curvature)
head(carbon_features, 4)
```

```
## # A tibble: 4 x 3
##   country     trend_strength curvature
##   <chr>                <dbl>     <dbl>
## 1 Afghanistan          0.972     0.218
## 2 Albania              0.945    -0.356
## 3 Algeria              0.164     0.232
## 4 Angola               0.931     0.502
```

The app includes two visualizations – a heatmap of the emissions time series and a scatterplot of the derived features. It also has a table showing the original data. The user can select countries to highlight by specifying a range of trend strengths and curvatures in the sliders. See the screenshot below.

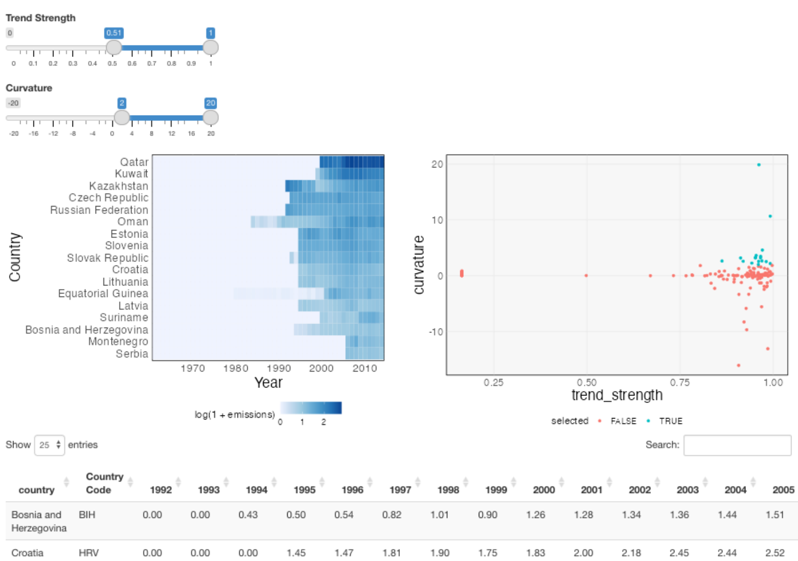


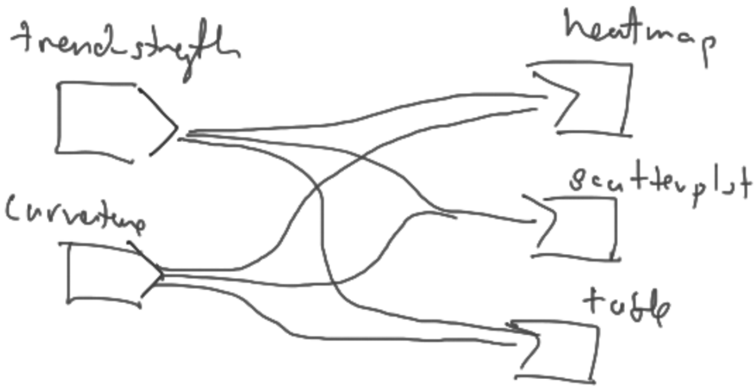
Figure 1: The shiny app to analyze in Problem Q4.

Here is the complete code for the app:

```r
ui <- fluidPage(
  sliderInput("trend_strength", "Trend Strength", 0, 1, c(0, 1)),
  sliderInput("curvature", "Curvature", -20, 20, c(-20, 20)),
  fluidRow(
    column(6, plotOutput("heatmap")),
    column(6, plotOutput("scatterplot"))
  ),
  dataTableOutput("table")
)

server <- function(input, output) {
  output$heatmap <- renderPlot({
    cur_countries <- carbon_features %>%
      filter(
        trend_strength >= input$trend_strength[1],
        trend_strength <= input$trend_strength[2],
```

```r
        curvature >= input$curvature[1],
        curvature <= input$curvature[2]
        ) %>%
        pull(country)

    carbon_ts %>%
      filter(country %in% cur_countries) %>%
      ggplot() +
      geom_tile(aes(year, reorder(country, emissions), fill = log(1 + emissions))) +
      labs(x = "Year", y = "Country") +
      scale_fill_distiller(direction = 1)
  })

  output$scatterplot <- renderPlot({
    cur_countries <- carbon_features %>%
      filter(
        trend_strength >= input$trend_strength[1],
        trend_strength <= input$trend_strength[2],
        curvature >= input$curvature[1],
        curvature <= input$curvature[2]
        ) %>%
        pull(country)

    carbon_features %>%
      mutate(selected = country %in% cur_countries) %>%
      ggplot() +
      geom_point(aes(trend_strength, curvature, col = selected))
  })

  output$table <- renderDataTable({
    cur_countries <- carbon_features %>%
      filter(
        trend_strength >= input$trend_strength[1],
        trend_strength <= input$trend_strength[2],
        curvature >= input$curvature[1],
        curvature <= input$curvature[2]
        ) %>%
        pull(country)

    carbon_ts %>%
      filter(country %in% cur_countries) %>%
      pivot_wider(names_from = year, values_from = emissions)
  })
}
```

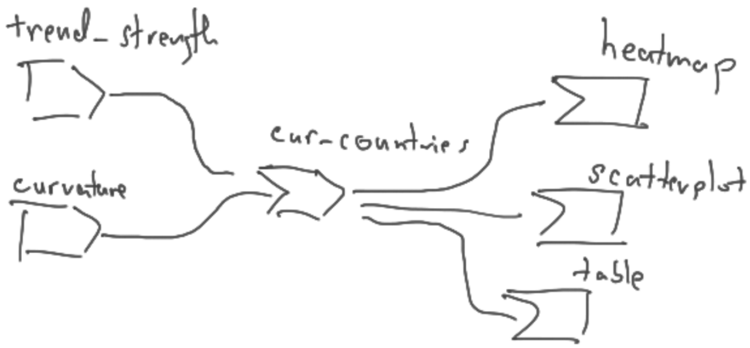
a. [2 points] Draw the reactive graph associated with this implementation. Be sure to distinguish input, output, and reactive nodes.



b. [3 points] Briefly describe how you would use `reactive({})` to simplify the implementation above. Sketch the updated reactive graph.

The code used to filter to the currently selected countries is repeated across each output. We can avoid this code duplication if we compute this within a reactive expression. Specifically, we could write a simplified version of the server:

This server now has the following reactive graph structure:



Q5

The `trees` data below includes a subset of data from the New York City Tree Census. Each row corresponds to one tree. For each tree, we have both its location and its health status.

```
trees <- read_sf("https://raw.githubusercontent.com/krisrs1128/stat679_code/main/activities/week7/trees.ge
  mutate(health = factor(health, levels = c("Good", "Fair", "Poor")))

head(trees, 4)
```

```
## Simple feature collection with 4 features and 14 fields
## Geometry type: POINT
## Dimension:     XY
## Bounding box:  xmin: -74.00477 ymin: 40.71486 xmax: -73.97391 ymax: 40.72647
## Geodetic CRS:  WGS 84
## # A tibble: 4 x 15
##   tree_id block_id species_group health tree_dbh stump_diam curb_loc steward sidewalk address
##     <dbl>    <dbl> <chr>         <fct>     <dbl>      <dbl> <chr>    <chr>   <chr>    <chr>
## 1  365757   102878 Other         Good         12          0 OnCurb   None    NoDamage 229 EAST 2 STREET
## 2  101366   100497 Callery pear  Good          6          0 OnCurb   1or2    NoDamage 290 BROADWAY
## 3  417933   103333 ginkgo        Fair          2          0 OnCurb   1or2    NoDamage 10-17 F D R DRIVE
## 4  103244   103112 pin oak       Good         20          0 OnCurb   None    Damage   121 AVENUE A
```

a. [1.5 points] Provide code for a scatterplot of the locations of all trees, coloring each in according to its `species_group`. See the figure below for an example result[1].

```r
tm_shape(trees) +
  tm_dots(col = "species_group", title = "Species", size = .01)
```

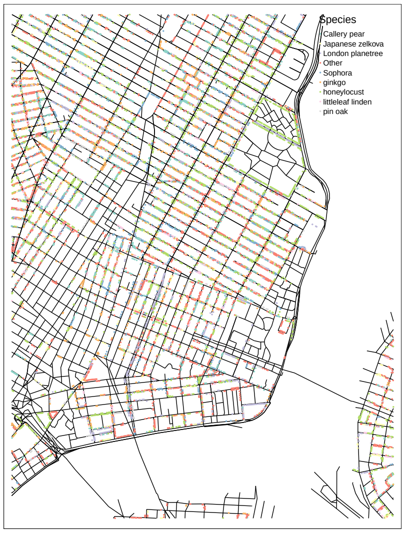
b. [1.5 points] The `roads` data below includes the trajectories of roads within the same geographic region. Each row corresponds to one street name.

```r
roads <- read_sf("https://uwmadison.box.com/shared/static/28y5003s1d0w9nqjnk9xme2n86xazuuj.geojson")
head(roads, 4)

## Simple feature collection with 4 features and 6 fields
## Geometry type: MULTILINESTRING
## Dimension:     XY
## Bounding box:  xmin: -73.99366 ymin: 40.7032 xmax: -73.9648 ymax: 40.70745
## Geodetic CRS:  WGS 84
## # A tibble: 4 x 7
##   STATEFP COUNTYFP LINEARID     FULLNAME      RTTYP MTFCC
##   <chr>   <chr>    <chr>        <chr>         <chr> <chr>
## 1 36      047      110422541625 New Dock St   M     S1400 ((-73.99309 40.70333, -73.99357 40.70446
## 2 36      047      110422543106 Washington St M     S1400                     ((-73.98954 40.70386
## 3 36      047      110422538238 Taylor St     M     S1400                     ((-73.96654 40.7037,
## 4 36      047      110422539438 Bedford Ave   M     S1400                     ((-73.96485 40.70745
```

Create a version of your visualization from (a) that includes the street map in the background. See the figure below for an example result.

```r
tm_shape(roads) +
  tm_lines() +
  tm_shape(trees) +
  tm_dots(col = "species_group", title = "Species", size = .01)
```

[1] Though printed in greyscale, the circles would appear in different colors on a computer screen.



c. [1 point] What additional layer would you add to your answer from (c) so that trees with different health statuses are placed into different facets? See the figure below for an example result.

```
tm_shape(roads) +
  tm_lines() +
  tm_shape(trees) +
  tm_dots(col = "species_group", title = "Species", size = .01) +
  tm_facets("health")
```

d. [1 point] For both the `trees` and `roads` datasets, state whether they are `raster` format, `vector` format, or neither. Briefly justify your choices.

Both datasets are examples of vector data. `trees` *is a collection of spatial points objects, and trees is a collection of spatial lines. Both are types of geometries that trace out coordinates in geographic space but without giving measurements across an even grid.*

Q6

The dataset below includes the number of Spotify streams of the most streamed song of 2017, "Shape of You." We have created a `tsibble` object from this dataset, filtering to plays from Japan and the US.

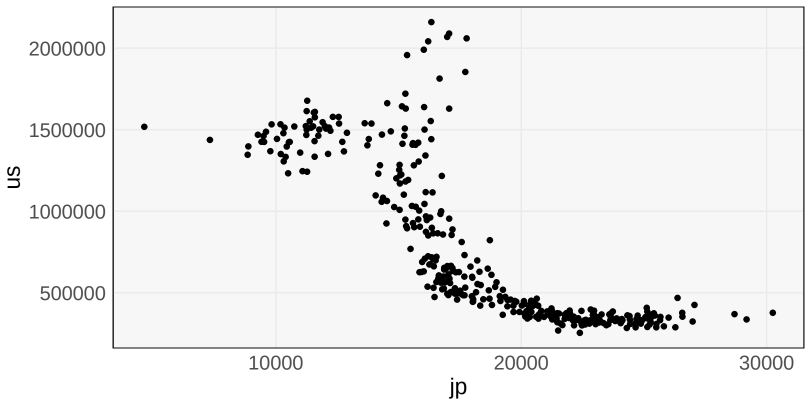
```
spotify <- read_csv("https://uwmadison.box.com/shared/static/hvplyr3jy6vbt7s80lqgfx81ai4hdl0q.csv") %>%
  filter(region %in% c("jp", "us")) %>%
  as_tsibble(region, index = date)

head(spotify, 4)

## # A tsibble: 4 x 5 [1D]
## # Key:       region [1]
##   artist     track_name   region date       streams
##   <chr>      <chr>        <chr>  <date>       <dbl>
```

```
## 1 Ed Sheeran Shape of You jp     2017-01-06    4639
## 2 Ed Sheeran Shape of You jp     2017-01-07    7313
## 3 Ed Sheeran Shape of You jp     2017-01-08    8851
## 4 Ed Sheeran Shape of You jp     2017-01-09    9827
```

a. [2 points] Provide code to create the scatterplot below, reshaping the data as necessary.

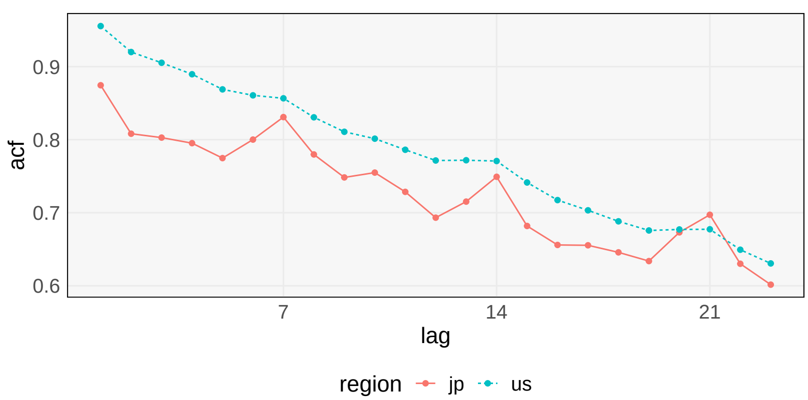
We first pivot the `region` *column so that the two countries appear separately. Then, we can use* `geom_point` *to create the scatterplot.*

```
spotify %>%
  pivot_wider(names_from = region, values_from = streams) %>%
  ggplot() +
  geom_point(aes(jp, us))
```



b. [2 points] The code below visualizes the associated autocorrelation functions for this dataset. Based on its output (figure on the next page), comment on both countries' time series trend strength and seasonality.

```
ACF(spotify, streams) %>%
  ggplot(aes(lag, acf, col = region, linetype = region)) +
  geom_line() +
  geom_point()
```



- *The US has a stronger trend, shown in its higher ACFs across most lags.*
- *Japan has stronger weekly seasonality since it shows distinct peaks at lags 7, 14, and 21.*

Q7

The dataset below describes hourly demand in a bikesharing service. Each row corresponds to a different hour (`hr`), and the variable `count` gives the number of bikes that have been checked out. The dataset additionally includes variables describing characteristics of the day during which the bike was checked out (e.g., `weekday` gives the day of the week) and the weather at the current hour (e.g., `temp` and `hum` are temperature and humidity).

```
bike <- read_csv("https://uwmadison.box.com/shared/static/f16jmkkskylfl1hnd5rpslzduja929g2.csv")
head(bike, 4)
```

```
## # A tibble: 4 x 11
##   dteday     season    yr  mnth    hr holiday weekday  temp   hum windspeed count
##   <date>      <dbl> <dbl> <dbl> <dbl>   <dbl>   <dbl> <dbl> <dbl>     <dbl> <dbl>
## 1 2011-01-01      1     0     1     0       0       6  0.24  0.81         0    16
## 2 2011-01-01      1     0     1     1       0       6  0.22  0.8          0    40
## 3 2011-01-01      1     0     1     2       0       6  0.22  0.8          0    32
## 4 2011-01-01      1     0     1     3       0       6  0.24  0.75         0    13
```

a. [2.5 points] Propose an interactive visualization to answer the questions:

   - How does bike demand vary over any given day?
   - In what temperature and humidity ranges is bike demand highest?

   Sketch and briefly annotate the layout of your proposed visualization. Describe what types of interactivity would be supported – how would the user provide input, and how would the visualization respond? Ensure that at least one graphical query is included.

   There are many possible answers to this problem. One proposal is to create a layout with two main components: A time series plot showing hourly demand over one day and a scatterplot plotting average daily temperature and humidity against each other. The time series plot allows us to answer the first question; for example, it highlights differences between mornings and evenings.

   For interactivity: Brushing the scatterplot would highlight the corresponding series in the time series plot. This would allow it to see whether certain temperature and humidity ranges bring up bike demand time series with systematically higher / lower demand. For example, during the very coldest days, the overall series should have a lower total number of bike checkouts.

b. [2.5 points] Provide code for a Shiny app's `server` component implementing your design from part (a). You may use shorthand for `ggplot2` figures (e.g., write ggplot code for "histogram of X" instead of the entire `ggplot` call). However, be as detailed as possible about how your graphical query would be implemented – include comments to ensure this part is clear.

   You were not expected to provide the full app associated with your proposal, but here is a working Shiny implementation of the proposal from part (a). The critical part of the implementation which we graded was the graphical query. Specifically, we looked for a graphical input within the ***`plotOutput`*** *definition, a reactive value in the server, and an appropriate observer.*

```
daily_summary <- bike %>%
  group_by(dteday) %>%
  summarise(
    mtemp = mean(temp),
    mhum = mean(hum)
  )

ui <- fluidPage(
  plotOutput("scatterplot", brush = "plot_brush"), # create a brush on the scatterplot
  plotOutput("time_series")
)

server <- function(input, output) {
  output$scatterplot <- renderPlot({
    ggplot(daily_summary) +
      geom_point(aes(mtemp, mhum))
  })

  # track which days are selected in the scatterplot
  selected <- reactiveVal(rep(TRUE, nrow(daily_summary)))
```

```r
  observeEvent(input$plot_brush, {

    # update the selected() variable
    selected(brushedPoints(daily_summary, input$plot_brush, allRows = TRUE)$selected_)
  })

  output$time_series <- renderPlot({
    current_days <- daily_summary %>%
      filter(selected()) %>%
      pull(dteday)

    bike %>%
      filter(dteday %in% current_days) %>%
      ggplot() +
      geom_line(aes(hr, count, group = dteday)) +
      ylim(0, 1000)
  })
}

shinyApp(ui, server)
```