

W271 Assignment 6

Hector Rincon

```
library(tidyverse)
library(patchwork)

library(lubridate)

library(tsibble)
library(feasts)
library(forecast)

library(sandwich)
library(lmtest)

library(nycflights13)
library(blsR)

theme_set(theme_minimal())
```

Plot Flights and Weather Data

To start with this homework, you will be using the same data that Jeffrey uses in the lecture – US flights data. The data comes from the packages **nycflights13**.

Question Goals

Our goal with the tasks in this question are to try to familiarize yourself with some of the key programming concepts related to time series data – setting time indexes and key variables, grouping and indexing on those variables, and producing descriptive plots of data that is stored in a time series form.

Question 1 - Flights to nice places

In the package declarations, we have loaded the **nycflights13** package. This provides three objects that we are going to use:

1. **flights**;
2. **airports**; and,
3. **weather**.

You can investigate these objects more by issuing a `?` before them, to access their documentation.

(1 point) Create Data

As stored, both **flights** and **weather** are stored as “plain” data frames. To begin, cast the **flights** dataset into a time series dataset, a **tsibble**.

- Use the combination of year, month, day, hour, and minute to produce the time index. Call this newly mutated variable `time_index`. There is very good handling of dates inside of the `lubridate` package. There is a nice one-page cheatsheet that Rstudio makes available. For this task you might be looking for `lubridate::make_datetime`.
- Although it may not generally be true, for this work, also assume that you can uniquely identify a flight by the carrier and the flight number, so you can use these two pieces of information to define the `key`. We need to define a key because in some cases there are more than one flight that leave at the same time – this is because the granularity of our time measure is at the minute and it is possible for two planes to leave within the same minute.

```
flights
```

```
## # A tibble: 336,776 x 19
##   year month   day dep_time sched_dep_t~1 dep_d~2 arr_t~3 sched~4 arr_d~5 carrier flight tailnum o
##   <int> <int> <int>   <int>         <int>   <dbl>   <int>   <int>   <dbl> <chr>   <int> <chr>   <
## 1  2013     1     1     517           515     2     830     819     11 UA      1545 N14228 EV
## 2  2013     1     1     533           529     4     850     830     20 UA      1714 N24211 L
## 3  2013     1     1     542           540     2     923     850     33 AA      1141 N619AA J
## 4  2013     1     1     544           545    -1    1004    1022    -18 B6       725 N804JB J
## 5  2013     1     1     554           600    -6     812     837    -25 DL       461 N668DN L
## 6  2013     1     1     554           558    -4     740     728     12 UA      1696 N39463 EV
## 7  2013     1     1     555           600    -5     913     854     19 B6       507 N516JB EV
## 8  2013     1     1     557           600    -3     709     723    -14 EV      5708 N829AS L
## 9  2013     1     1     557           600    -3     838     846     -8 B6        79 N593JB J
## 10 2013     1     1     558           600    -2     753     745      8 AA       301 N3ALAA L
## # ... with 336,766 more rows, and abbreviated variable names 1: sched_dep_time, 2: dep_delay, 3: arr
## # 7: distance
```

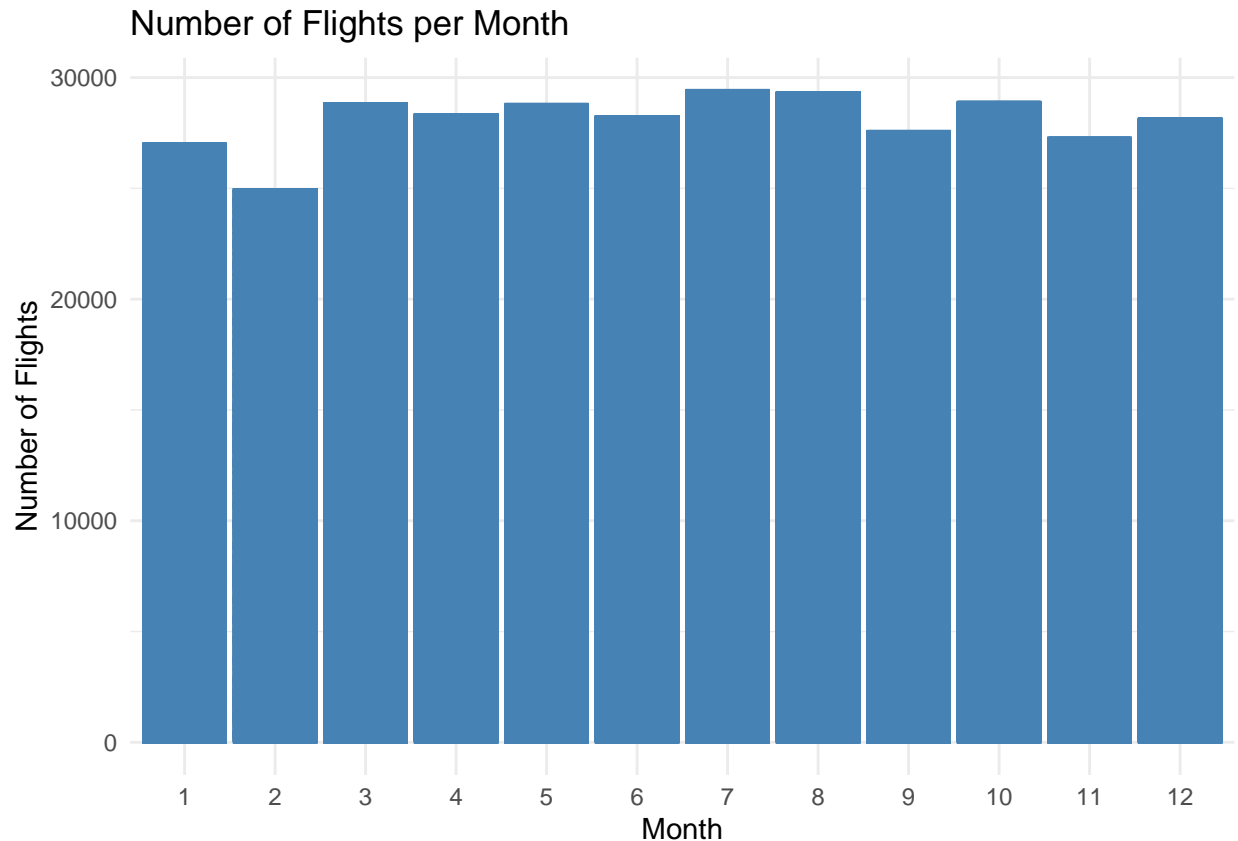
```
flights <- nycflights13::flights %>% mutate(time_index = make_datetime(year, month, day, hour, minute,
flights <- as_tsibble(
  flights,
  key = c(carrier, flight),
  index = time_index,
  regular = FALSE
)
weather <- as_tsibble(weather, key=origin, index=time_hour)
```

(1 point) Flights Per Month

Using `ggplot`, create a plot of the number of flights per month. What, if anything, do you note about the total volume of flights throughout the year? (Don't worry if the plot doesn't tell something interesting about the data. This data is pretty... boring.)

```
monthly_flights <- flights %>% group_by(month = as.factor(month)) %>% summarise(total_flights = n())

# Create the bar chart
ggplot(monthly_flights, aes(x = month, y = total_flights)) +
  geom_bar(color = "steelblue", stat="identity") +
  ggtitle("Number of Flights per Month") +
  xlab("Month") +
  ylab("Number of Flights")
```



From the plot we get that the volume of flights per month stays relatively consistent throughout the year except for January and February, where there is a significant dip.

(1 point) The Tropics

Is there a difference in flights to tropical destinations throughout the year? Use the following concept of a tropical destination:

A tropical destination is one who is “in the tropics” – that is, they are located between the Tropic of Cancer and the Tropic of Capricorn.

1. Using the `airports` dataset, create a new variable, `is_tropical` that notes whether the destination airport is in a tropical latitude.
2. Join this `airports` data onto the `flights` data.
3. Produce a plot that shows the volume of flights to tropical and non-tropical destinations, counted as a monthly total, throughout the year.
 - a. First, try to do this using a `group_by` call that groups on `month` and `is_tropical`. Why does this not work? What is happening when grouping by `month` while also having a time index? *This does not work because we told the tsibble that `time_index` is indexing each `key` (i.e. we are already grouping by `key`, where we have multiple observations).*
 - b. Instead, you will need to look into `tsibble::index_by` and combine this with a `lubridate` “extractor” to pull the time object that you want out of the `time_index` variable that you created.
 - c. To produce the plot, `group_by(is_tropical)`, and `index_by` the month that you extract from your `time_index`. (This is a bit of a strange part of the `geom_*` API, but this might be a useful place to use the `geom_step` geometry to highlight changes in this series.)
4. Comment on what you see in the flights to the tropics, compared to flight to non-tropical destinations.

```

tropic.lat <- 23.4394
airports <- mutate(airports, is_tropical = between(lat, -tropic.lat, tropic.lat))
flights <- left_join(flights, rename(airports, dest=faa), by="dest")

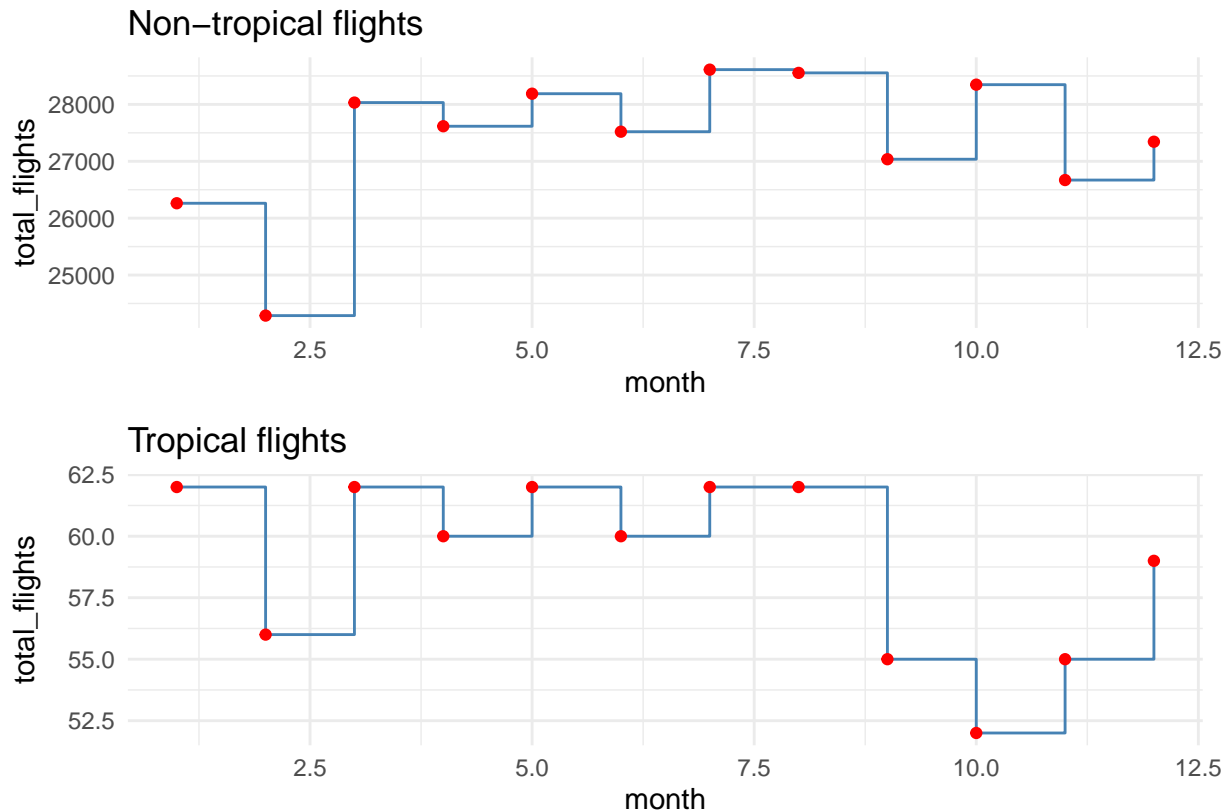
flights %>%
  group_by(month, is_tropical) %>%
  summarise(total_flights = n())

## # A tibble: 132,642 x 4 [!] <America/New_York>
## # Key:      month, is_tropical [36]
## # Groups:   month [12]
##   month is_tropical time_index      total_flights
##   <int> <lgl>      <dtm>          <int>
## 1     1 FALSE      2013-01-01 05:15:00           1
## 2     1 FALSE      2013-01-01 05:29:00           1
## 3     1 FALSE      2013-01-01 05:40:00           1
## 4     1 FALSE      2013-01-01 05:58:00           1
## 5     1 FALSE      2013-01-01 05:59:00           1
## 6     1 FALSE      2013-01-01 06:00:00          17
## 7     1 FALSE      2013-01-01 06:05:00           1
## 8     1 FALSE      2013-01-01 06:07:00           1
## 9     1 FALSE      2013-01-01 06:08:00           1
## 10    1 FALSE      2013-01-01 06:10:00           5
## # ... with 132,632 more rows

flights.by.tropical <- flights %>% filter(!is.na(is_tropical)) %>%
  index_by(month= ~ lubridate::month(.)) %>%
  group_by(is_tropical) %>%
  summarise(total_flights=n())

p1 <- ggplot(filter(flights.by.tropical, is_tropical == T), aes(month, total_flights)) +
  geom_step(color="steelblue") +
  geom_point(color="red") +
  labs(title="Tropical flights")
p2 <- ggplot(filter(flights.by.tropical, is_tropical == F), aes(month, total_flights)) +
  geom_step(color="steelblue") +
  geom_point(color="red") +
  labs(title="Non-tropical flights")
p2 / p1

```



Flights to the tropics tend to be lowest in the last quarter of the year, whereas they tend to be relatively high in within the first 9 months of the year. Non-tropical flights, in comparison, tend to remain low for Jan. and Feb., but remain high towards the latter part of the year. In both series we see February represents a dip in the flights.

Question 2 - Weather at New York Airports

Our goal in this question is to ask you to re-apply what you know about producing time series objects to very similarly structured data.

(1 point) Create a time series of weather

Turn your attention to the weather data that is provided in the `nycflights13::weather` dataset. Produce a `tsibble` that uses time as a time index, and `origin` as a key for this data. You will notice that there are three origins, “EWR”, “JFK” and “LGA”.

(Hint: We anticipate that you are going to see the following error on the first time that you try to convert this data frame:

```
Error in `validate_tsibble()` :
A valid tsibble must have distinct rows identified by key and index.
Please use `duplicates()` to check the duplicated rows.
Run `rlang::last_error()` to see where the error occurred.
```

This is a *very* helpful error, with a helpful error message. If you see this error message, we suggest doing as the message suggests, and look into the `duplicates()` function to determine what the issue is. Once you have found the issue, (1) document the issue; (2) propose a solution that seems reasonable; and, (3) implement your proposed solution and keep it moving to answer this question.

```
weather %>%
  mutate(time_index = make_datetime(year, month, day, hour)) %>%
  duplicates(key = origin, index = time_index) %>%
  glimpse()
```

```
## Rows: 6
## Columns: 16
## $ origin    <chr> "EWR", "EWR", "JFK", "JFK", "LGA", "LGA"
## $ year      <int> 2013, 2013, 2013, 2013, 2013, 2013
## $ month     <int> 11, 11, 11, 11, 11, 11
## $ day       <int> 3, 3, 3, 3, 3, 3
## $ hour      <int> 1, 1, 1, 1, 1, 1
## $ temp      <dbl> 51.98, 50.00, 53.96, 51.98, 55.04, 53.96
## $ dewp      <dbl> 39.02, 39.02, 37.94, 37.94, 39.02, 39.92
## $ humid     <dbl> 61.15, 65.80, 54.51, 58.62, 54.67, 58.89
## $ wind_dir  <dbl> 310, 290, 320, 310, 330, 310
## $ wind_speed <dbl> 6.90468, 5.75390, 9.20624, 6.90468, 9.20624, 8.05546
## $ wind_gust <dbl> NA, NA, NA, NA, NA, NA
## $ precip    <dbl> 0, 0, 0, 0, 0, 0
## $ pressure  <dbl> 1009.8, 1010.5, 1009.8, 1010.5, 1009.3, 1010.2
## $ visib     <dbl> 10, 10, 10, 10, 10, 10
## $ time_hour <dtm> 2013-11-03 01:00:00, 2013-11-03 01:00:00, 2013-11-03 01:00:00, 2013-11-03 01:00:00, 2013-11-03 01:00:00, 2013-11-03 01:00:00
## $ time_index <dtm> 2013-11-03 01:00:00, 2013-11-03 01:00:00, 2013-11-03 01:00:00, 2013-11-03 01:00:00, 2013-11-03 01:00:00, 2013-11-03 01:00:00
```

Note: The initial tsibble creation for weather did not fail with the above suggested error. In any case what I would do is probably add a small delta to each duplicate value of `time_hour` to have distinct combinations of (key, index).

(4 points) Plot temperature

With this weather data, produce the following figure of the temperature every hour, for each of the origins.

This figure contains five separate plots:

- One that shows the entire year's temperature data;
- Two that show the month of January and July; and,
- Two that show the first week of January and July.

You might think of these plots as “zooming in” on the time series to show more detail.

In your workflow, first create each of the plots. Then, use the `patchwork` package to compose each of these plots into a single figure.

After you produce this figure, comment on what you notice at each of these scales and the figure overall.

```
yearly_plot <- ggplot(weather, aes(time_hour, temp)) +
  geom_line(aes(color=origin), alpha=.5) +
  labs(title="Arrival temperature", y="Temperature", x="")
```

```
january_plot <- filter(weather, month == 1) %>% ggplot( aes(time_hour, temp)) +
  geom_line(aes(color=origin), alpha=.8) +
  labs(title="January", y="Temperature", x="")
```

```
july_plot <- filter(weather, month == 7) %>% ggplot( aes(time_hour, temp)) +
  geom_line(aes(color=origin), alpha=.8) +
  labs(title="July", y="Temperature", x="")
```

```
january_first_week <- filter(weather, month == 1, day <= 7) %>% ggplot( aes(time_hour, temp)) +
  geom_line(aes(color=origin), alpha=.8) +
```

Temperature at NYC Airports

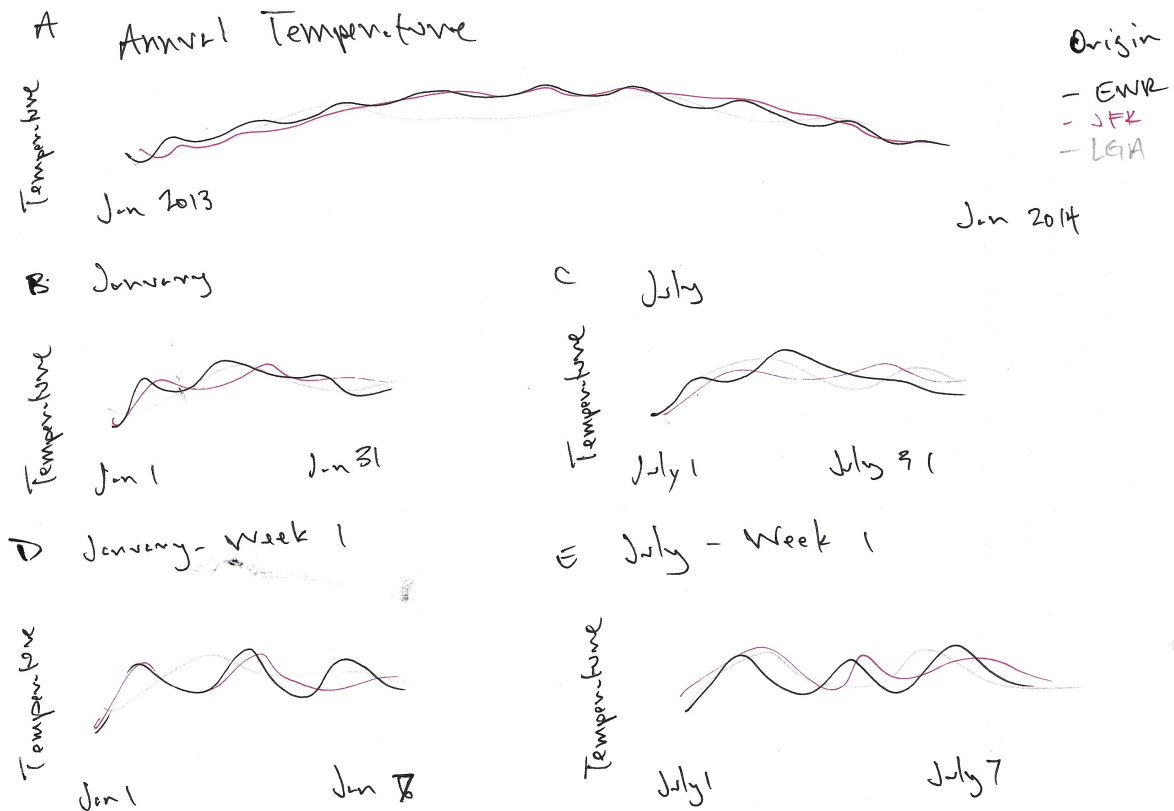


Figure 1: sample plot

```

labs(title="January - Week 1", y="Temperature", x="")
july_first_week <- filter(weather, month == 7, day <= 7) %>% ggplot( aes(time_hour, temp)) +
  geom_line(aes(color=origin), alpha=.8) +
  labs(title="July - Week 1", y="Temperature", x="")

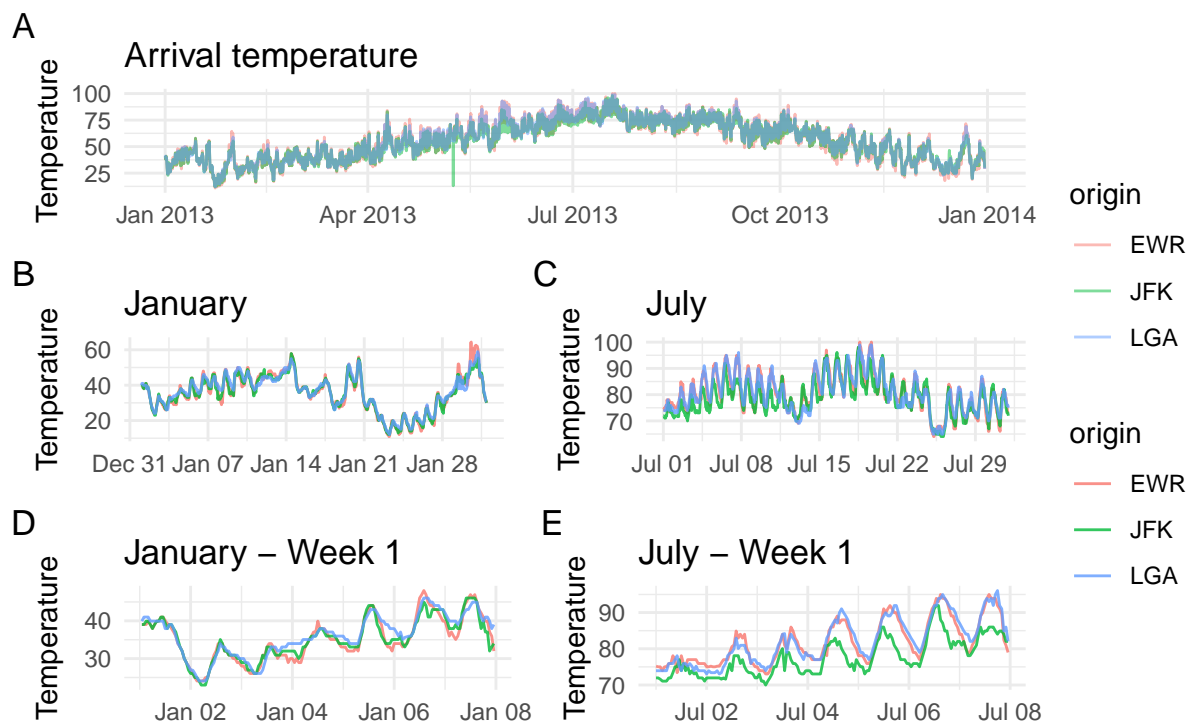
## Uncomment

yearly_plot /
(january_plot | july_plot) /
(january_first_week | july_first_week) +
  plot_layout(guides="collect") +
  plot_annotation(
    title = 'Temperature at New York City Airports',
    subtitle = 'Many Different Views',
    tag_levels = 'A') &
  labs(x = NULL, y = 'Temperature')

```

Temperature at New York City Airports

Many Different Views



(1 point) Hourly ACF

At the hourly level, produce an ACF and a lag plot at JFK. What do you learn from these plots? (Note that you can suppress all the coloring in the `gg_lag` call if you pass an additional argument, `color = 1`.)

```

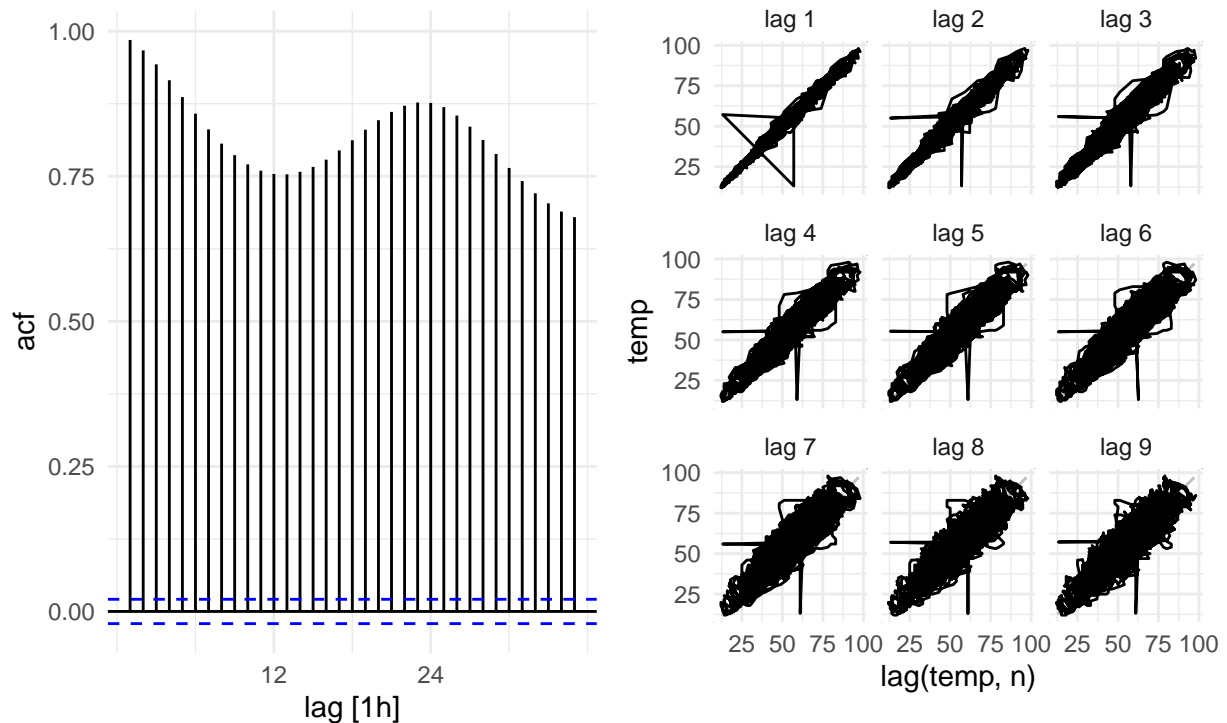
jfk.weather <- filter(weather, origin=="JFK")
hourly_acf <- ACF(jfk.weather %>% fill_gaps(), temp) %>% autoplot()
hourly_lag <- jfk.weather %>% gg_lag(temp, color=1)

```



```
(hourly_acf | hourly_lag) + plot_annotation(title="Hourly ACF and lag plots for JFK weather")
```

Hourly ACF and lag plots for JFK weather



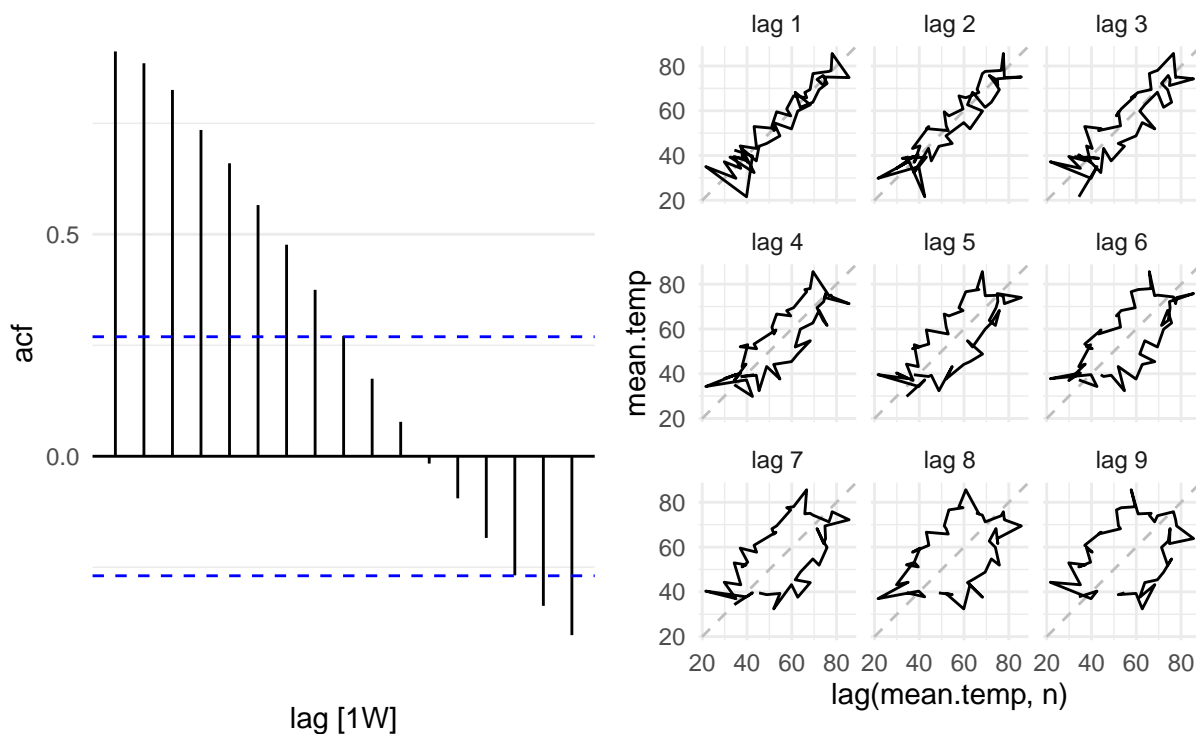
(1 point) Weekly ACF

At the weekly level, produce an ACF and a lag plot of the weekly average temperature at JFK. What do you learn from these plots?

```
weekly.weather.jfk <- jfk.weather %>%
  group_by_key() %>%
  index_by(Year_week = ~ yearweek(.)) %>%
  summarise(mean.temp = mean(temp))
weekly_acf <- ACF(weekly.weather.jfk %>% fill_gaps(), mean.temp) %>% autoplot()
weekly_lag <- weekly.weather.jfk %>% gg_lag(mean.temp, color=1)

## make the plot
(weekly_acf | weekly_lag) +
  plot_annotation(
    title = 'Weekly ACF and lag plots for JFK weather',
  )
```

Weekly ACF and lag plots for JFK weather



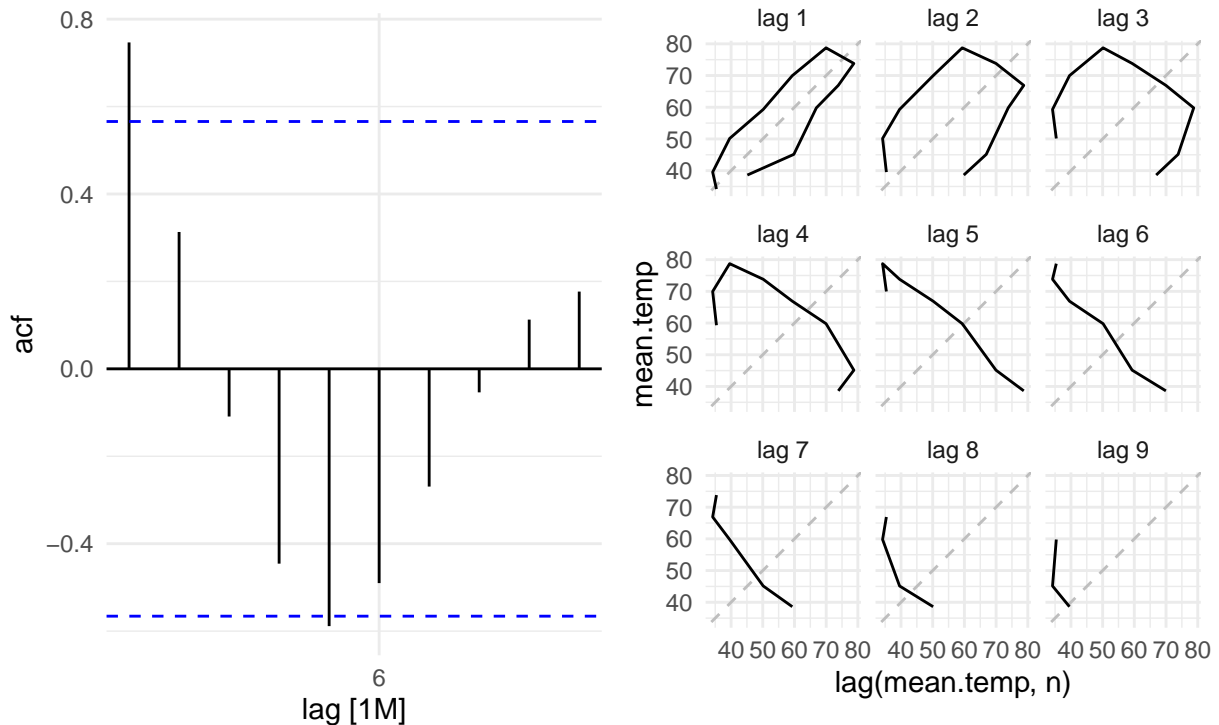
(1 point) Monthly ACF

At the monthly level, produce an ACF plot of the monthly average temperature at JFK. What do you learn from these plots?

```
monthly.weather.jfk <- jfk.weather %>%
  group_by_key() %>%
  index_by(Year_month = ~ yearmonth(.)) %>%
  summarise(mean.temp = mean(temp))
monthly_acf <- ACF(monthly.weather.jfk %>% fill_gaps(), mean.temp) %>% autoplot()
monthly_lag <- monthly.weather.jfk %>% gg_lag(mean.temp, color=1)

## make the plot
(monthly_acf | monthly_lag) + plot_annotation(title="Monthly ACF and lag plots for JFK weather")
```

Monthly ACF and lag plots for JFK weather



Question 3 - Evaluate Time Series Objects

In this section, we are asking you to use the plotting tools that you have learned in the course to evaluate a time series of “unknown” origin. This week, we will simply be describing what we see in the time series; in future weeks we will also be conducting tests to evaluate whether these are stationary and the order of the time series.

For each time series that you evaluate, provide enough understanding of the series using plots and summaries that a collaborator would agree with your assessment, but do not use more than one printed page per dataset.

This will assuredly mean that not *every* plot or diagnostic that you produce initially will make it to what you present. Edit with intent – what you show your audience should move forward your assessment.

To begin, load the data set constructor, which is stored in `./dataset_generator/`. Within this folder, there is a file named `make_datasets.R`.

- By issuing the `source()` call, you will bring this function into the global namespace, and so can then execute the function by issuing `make_datasets()`.
- We have elected to use one (clumsy) idiom in this function – we have elected to assign objects that are generated *within* the function scope out into the global scope. If you look into `make_datasets()`, which is possible by issuing the function name without the parentheses, you will see that we are assigning using `<<-`. This reads in a similar way to the standard assignment, `<-`, but works globally. We have elected to do this so that all the time series objects that you create are available to you (and to us as graders) at the top, global-level of your session.
- While you *could* try to reverse engineer the randomization that we’ve built in this function to figure out which generator is associated with which data – please don’t. Or, at least, don’t until you’ve finished your diagnostics.

```
source('./dataset_generator/make_datasets.R')
```

This function will make five data sets and store them in the global environment. They will be named, rather creatively:

- dataset_1
- dataset_2
- dataset_3
- dataset_4
- dataset_5

Your task is to use the plots and concepts covered in lecture and in *Forecasting, Principles and Practices* to describe the series that you see. Are any of these series white noise series? Do any of these series show trends or seasonal patterns?

For each series, using the `patchwork` library to layout your plots, produce a figure that:

- Shows the time series in the first plot;
- Shows the relevant diagnostic plots in one or two more plots within the same figure.

The additional plots could be lag plots, autocorrelation plots, partial autocorrelation plots, or whatever you think makes it the most clear for an interested audience who is as familiar as you with time series analysis come to an understanding of the series.

Along with each plot, include descriptive text (at least several sentences, but not more than a paragraph or two) that describes what, if anything you observe in the series. This is your chance to state what you see, so that your audience can (a) be informed of your interpretation; and (b) come to their own interpretation.

Your analysis and description of each dataset should fit onto a single PDF page.

```
make_datasets()
```

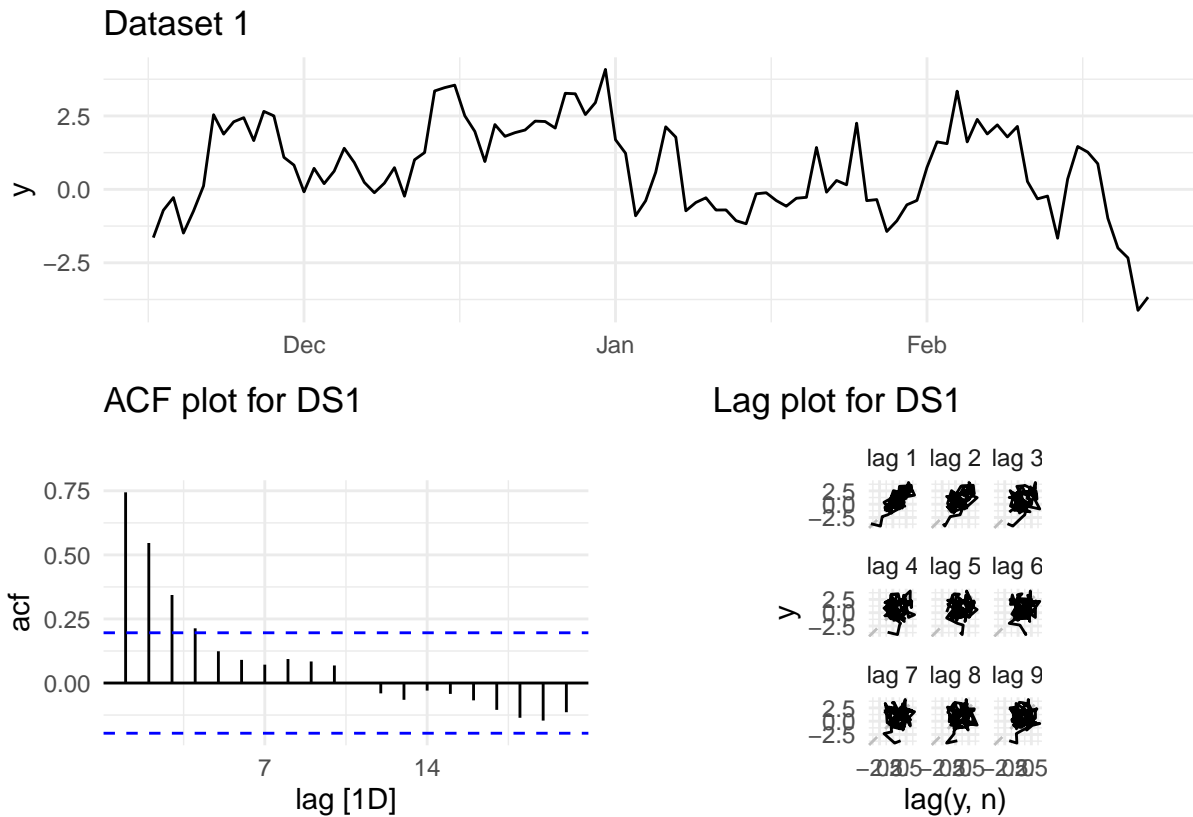
```
## Using `date` as index variable.  
## Using `date` as index variable.  
## Using `date` as index variable.
```

```
## Warning in min(Mod(polyroot(c(1, -model$ar)))): no non-missing arguments to min; returning Inf
```

```
## Using `date` as index variable.  
## Using `date` as index variable.
```

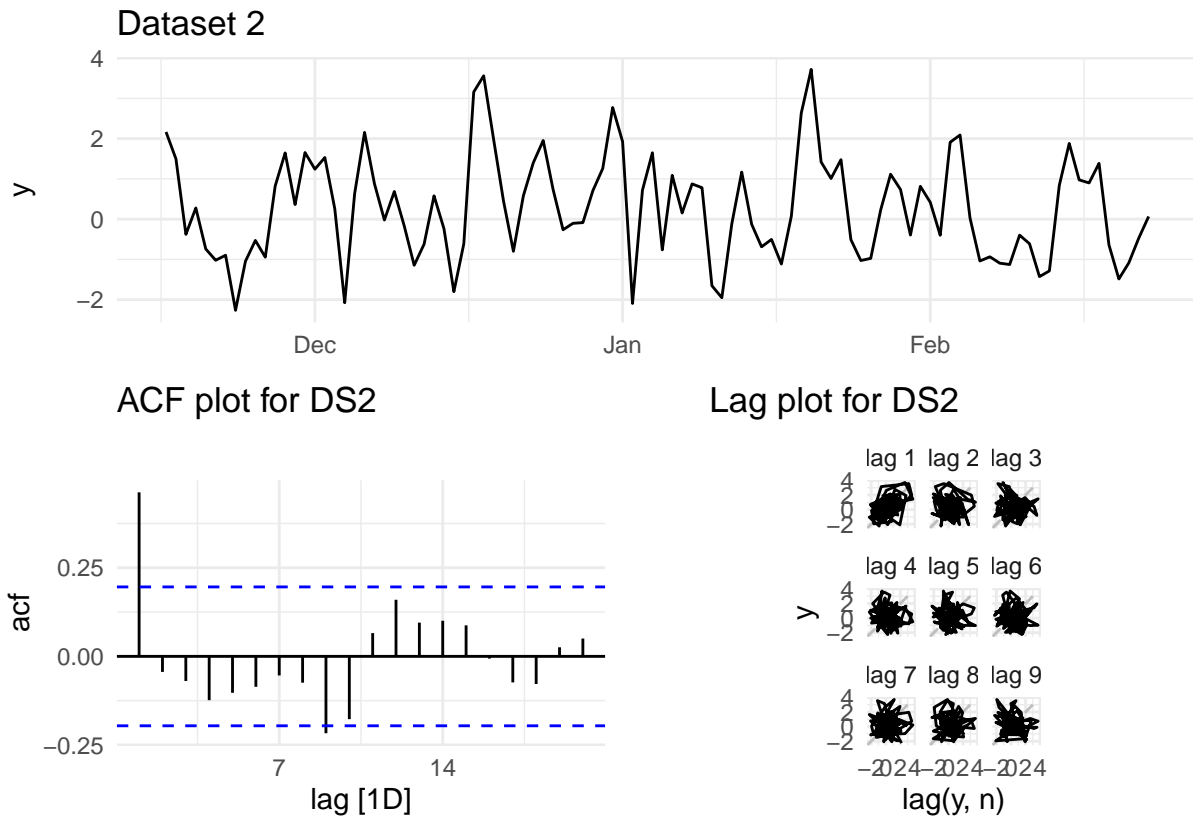
(1 point) Dataset One

```
ds1.ts <- dataset_1 %>% autoplot(y) + labs(title="Dataset 1", x=NULL)  
ds1.acf <- dataset_1 %>% ACF(y) %>% autoplot() + labs(title="ACF plot for DS1")  
ds1.lag <- dataset_1 %>% gg_lag(y, color=1) + labs(title="Lag plot for DS1")  
ds1.ts / (ds1.acf + ds1.lag)
```



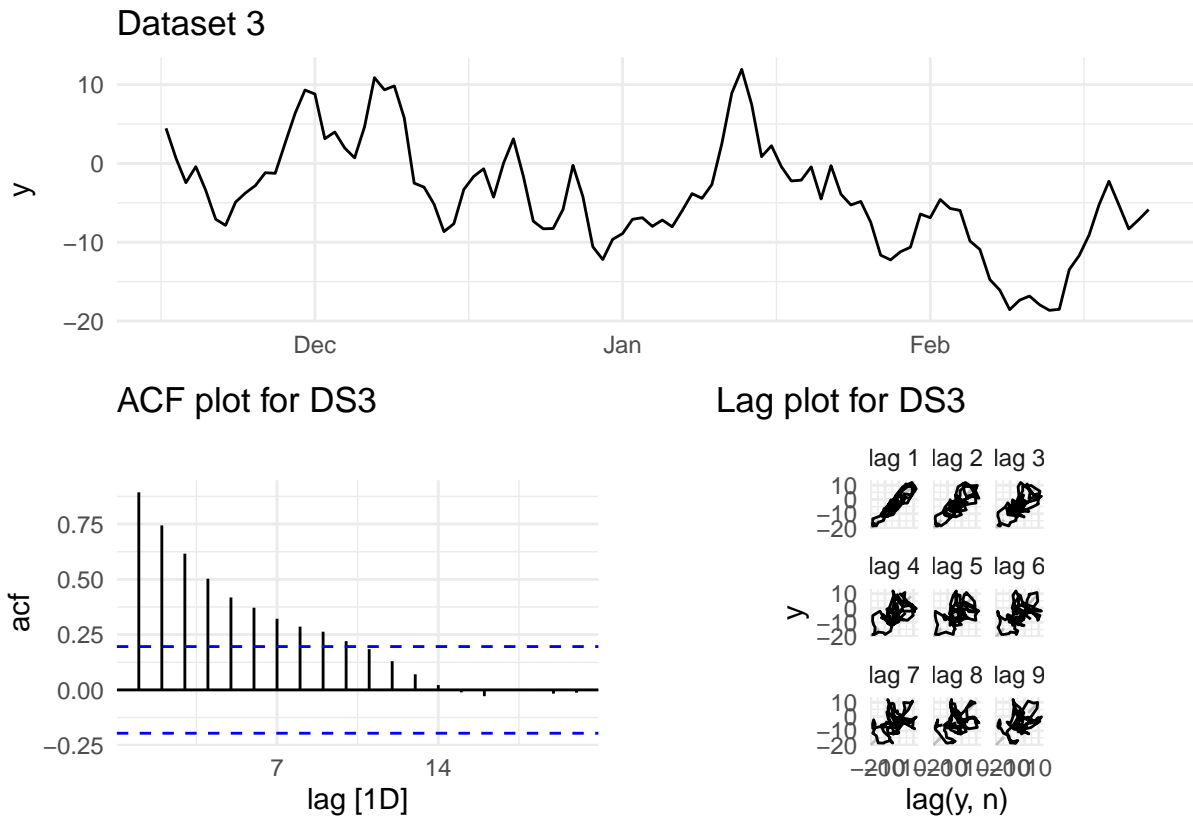
(1 point) Dataset Two

```
ds2.ts <- dataset_2 %>% autoplot(y) + labs(title="Dataset 2", x=NULL)
ds2.acf <- dataset_2 %>% ACF(y) %>% autoplot() + labs(title="ACF plot for DS2")
ds2.lag <- dataset_2 %>% gg_lag(y, color=1) + labs(title="Lag plot for DS2")
ds2.ts / (ds2.acf + ds2.lag)
```



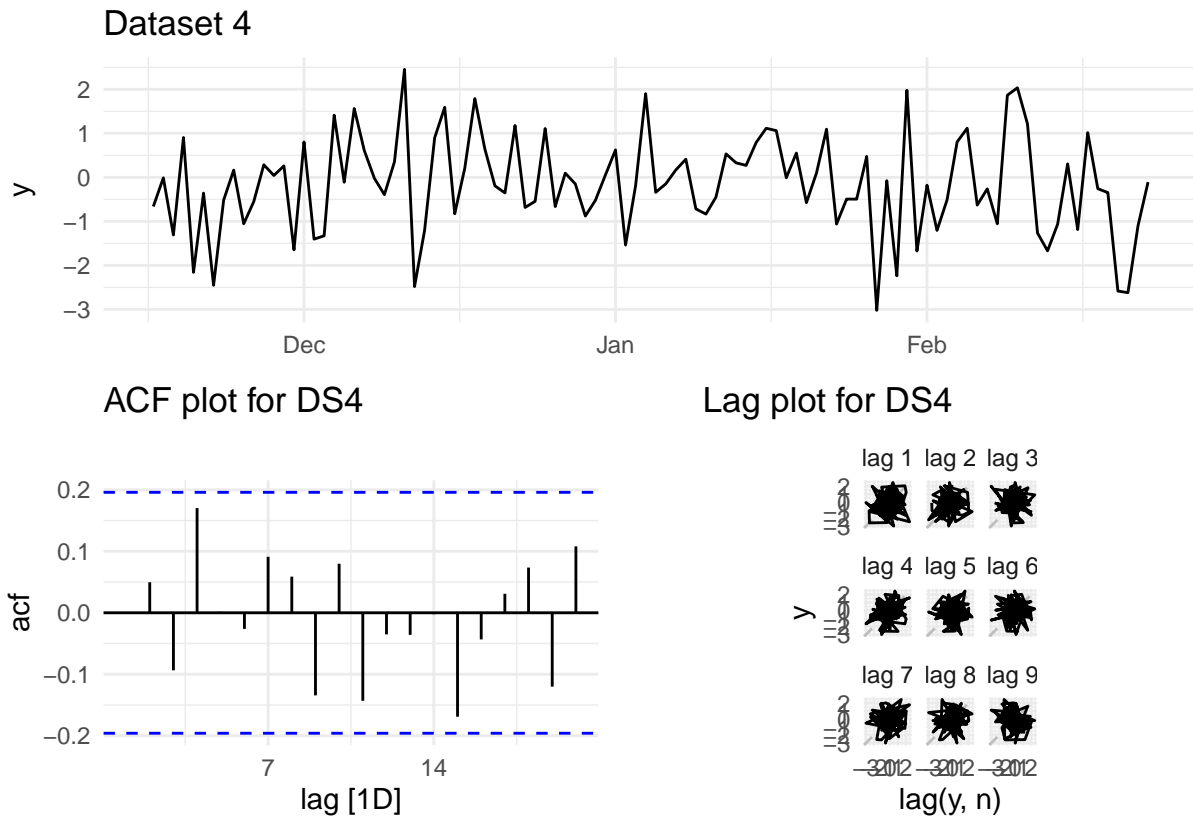
(1 point) Dataset Three

```
ds3.ts <- dataset_3 %>% autoplot(y) + labs(title="Dataset 3", x=NULL)
ds3.acf <- dataset_3 %>% ACF(y) %>% autoplot() + labs(title="ACF plot for DS3")
ds3.lag <- dataset_3 %>% gg_lag(y, color=1) + labs(title="Lag plot for DS3")
ds3.ts / (ds3.acf + ds3.lag)
```



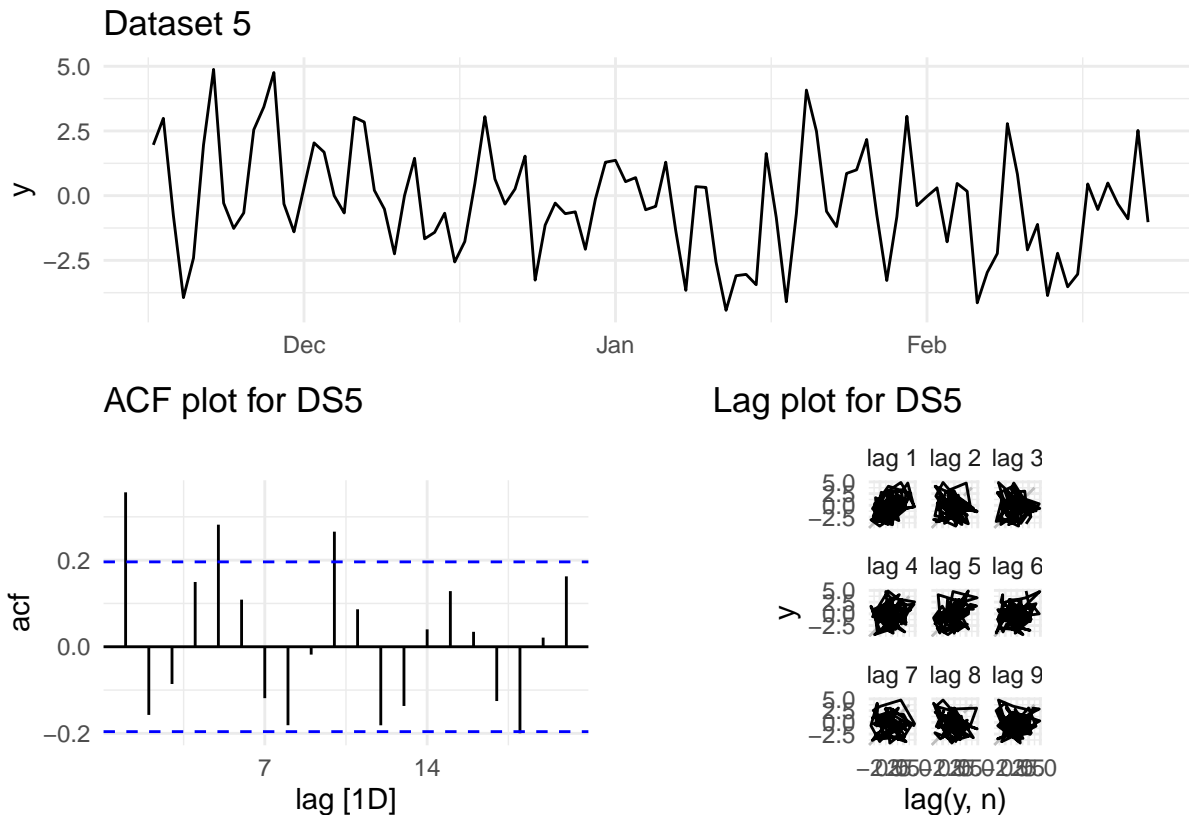
(1 point) Dataset Four

```
ds4.ts <- dataset_4 %>% autoplot(y) + labs(title="Dataset 4", x=NULL)
ds4.acf <- dataset_4 %>% ACF(y) %>% autoplot() + labs(title="ACF plot for DS4")
ds4.lag <- dataset_4 %>% gg_lag(y, color=1) + labs(title="Lag plot for DS4")
ds4.ts / (ds4.acf + ds4.lag)
```



(1 point) Dataset Five

```
ds5.ts <- dataset_5 %>% autoplot(y) + labs(title="Dataset 5", x=NULL)
ds5.acf <- dataset_5 %>% ACF(y) %>% autoplot() + labs(title="ACF plot for DS5")
ds5.lag <- dataset_5 %>% gg_lag(y, color=1) + labs(title="Lag plot for DS5")
ds5.ts / (ds5.acf + ds5.lag)
```

Question 4 - BLS Data

This is the last exercise for this assignment. Here, we're going to do the same work that you have done twice before, but against "live" data that comes from the United States' Bureau of Labor Statistics.

Recall that in the lecture, Jeffrey identifies the unemployment rate as an example of a time series. You can get to this data from the public web-site. To do so, head here:

- www.bls.gov > Data Tools > BLS Popular Series
- Then check the box for **Unemployment Rate (Seasonally Adjusted)** and **Retrieve Data**. Take note when you check the **Unemployment Rate (Seasonally Adjusted)**, what is the series number that is associated with this?

What do you see when you get to the next page? A rectangular data series that has months on the columns, years on the rows, and values as the internals to the cells? :facepalm:

- Does this meet the requirements of tidy data, or time series tidy data?
- If you were to build an analytic pipeline against data that you accessed in this way, what would be the process to update your analysis when the next edition of data is released? Would it require a manual download, then cleaning, then movement into your analysis? Could this be problematic?

This motivates the idea of using the BLS' data API. The data API provides consistently formatted JSON objects that can be converted to data of an arbitrary (that is, useful to us) formatting. Because the data is being provided in a JSON object, there is some work to coerce it to be useful, but we'll find that there are so many people who are doing this same coercion that there are ready-made wrappers that will help us to do this work.

As an example, you can view how these JSON objects are formatted by navigating to an API endpoint in

your browser. Here is the endpoint for the national unemployment: [\[link\]](#).

Let's pull unemployment from the BLS data API.

1. Register for an API key with the BLS. You can register for this from the BLS' "Getting Started" page. They will then send you an API key to the email that you affiliate.
2. Find the series that we want to access. Frankly, this is part of accessing this API that is the most surprisingly difficult – the BLS does not publish a list of the data series. From their Data Retrieval Tools page there are links to popular series, a table lookup, and a Data Finder. Elsewhere they provide pages that describe how series IDs are formatted, but finding series still requires considerable meta-knowledge.

For this assignment, consider the following three series:

1. Total unemployment: LNS14000000
2. Male unemployment: LNS14000001
3. Female unemployment: LNS14000002

Our goal is to analyze these three series for the last 20 years.

To articulate the BLS API, we have found the `blsR` library to be the most effective (at the time that we wrote the assignment in 2022). Here are links to get you read into the package. Rather than providing you with a *full* walk-through for how to use this package to manipulate the BLS data API, instead a learning goal is for you to read these documents and come to an understanding of how the package works.

- CRAN Homepage
- GitHub
- Vignette (Called, incorrectly a README on the CRAN page)

(2 points) Form a successful query and tidy of data

Your task is to create an object called `unemployment` that is a `tsibble` class, that contains the overall unemployment rate, as well as the unemployment rate for male and female people.

Your target dataframe should have the following shape but extend to the current time period.

	year	month	time_index	name	value
	<int>	<int>	<mth>	<chr>	<dbl>
1	2000	1	2000 Jan	overall	4
2	2000	1	2000 Jan	male	3.9
3	2000	1	2000 Jan	female	4.1
4	2000	2	2000 Feb	overall	4.1
5	2000	2	2000 Feb	male	4.1
6	2000	2	2000 Feb	female	4.1
7	2000	3	2000 Mar	overall	4
8	2000	3	2000 Mar	male	3.8
9	2000	3	2000 Mar	female	4.3
10	2000	4	2000 Apr	overall	3.8

```
uedf <- get_n_series_table(
  list(uer.overall='LNS14000000', uer.men='LNS14000001', uer.women='LNS14000002'),
  start_year = 2000, end_year=2022, tidy=TRUE,
  api_key='6e9389b28c934a2e8d25fa510fafb98d'
)
```

Year 2000 to 2022 is longer than 20 year API limit. Performing 2 requests.

```
uedf <- mutate(uedf, time_index=make_yearmonth(year, month)) %>%
  gather("name", "value", starts_with('uer'))
uedf$name <- stringr::str_replace(uedf$name, "^uer\\.\\. ", "")
uedf <- as_tsibble(uedf, key=c(name), index=time_index)
```

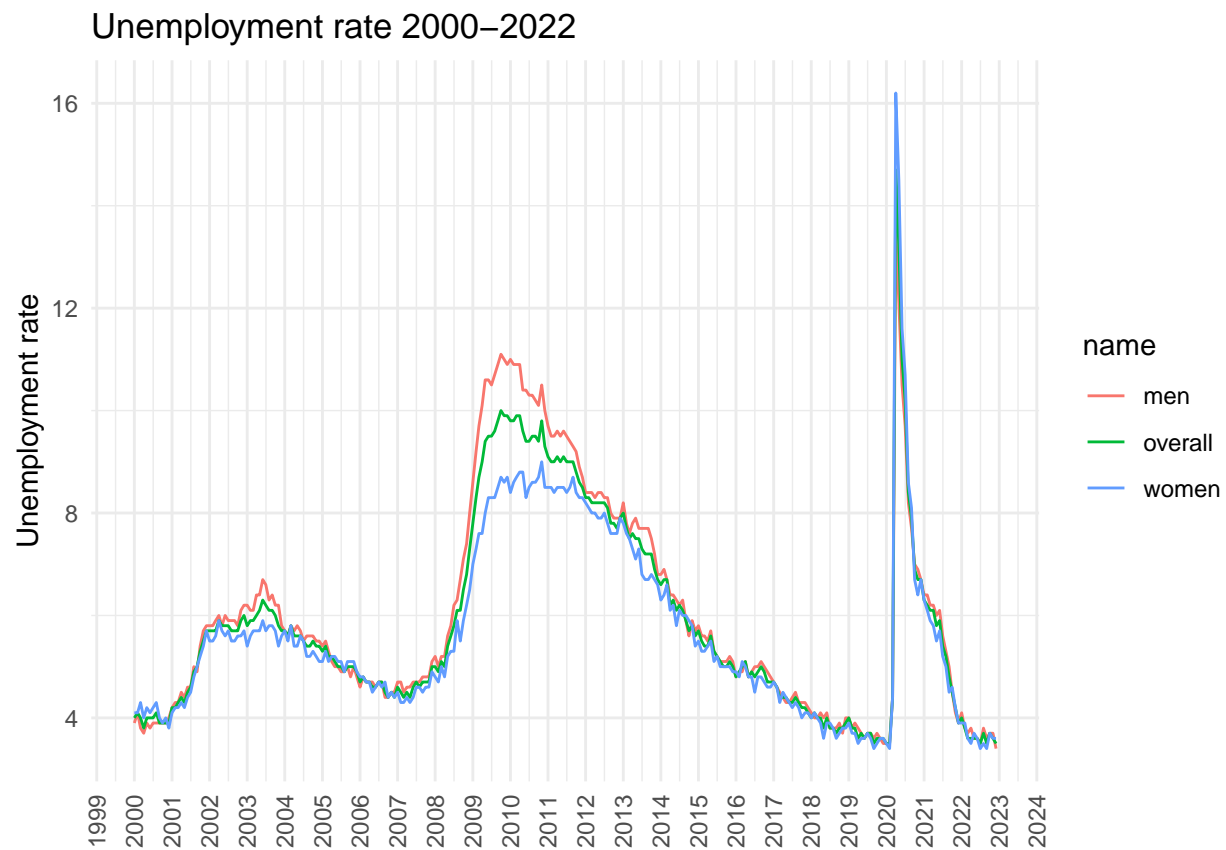
```
unemployment <- uedf
head(unemployment)
```

```
## # A tsibble: 6 x 5 [1M]
## # Key:      name [1]
##   year month time_index name  value
##   <int> <int>      <mth> <chr> <dbl>
## 1  2000     1    2000 Jan  men    3.9
## 2  2000     2    2000 Feb  men    4.1
## 3  2000     3    2000 Mar  men    3.8
## 4  2000     4    2000 Apr  men    3.7
## 5  2000     5    2000 May  men    3.9
## 6  2000     6    2000 Jun  men    3.8
```

(1 point) Plot the Unemployment Rate

Once you have queried the data and have it successfully stored in an appropriate object, produce a plot that shows the unemployment rate on the y-axis, time on the x-axis, and each of the groups (overall, male, and female) as a different colored line.

```
ggplot(unemployment, aes(time_index, value, color=name)) +
  geom_line() +
  labs(title="Unemployment rate 2000-2022", x=NULL, y="Unemployment rate") +
  scale_x_yearmonth(date_breaks = "1 year", date_labels = "%Y") +
  theme(axis.text.x = element_text(angle = 90, vjust = 0.5, hjust=1))
```

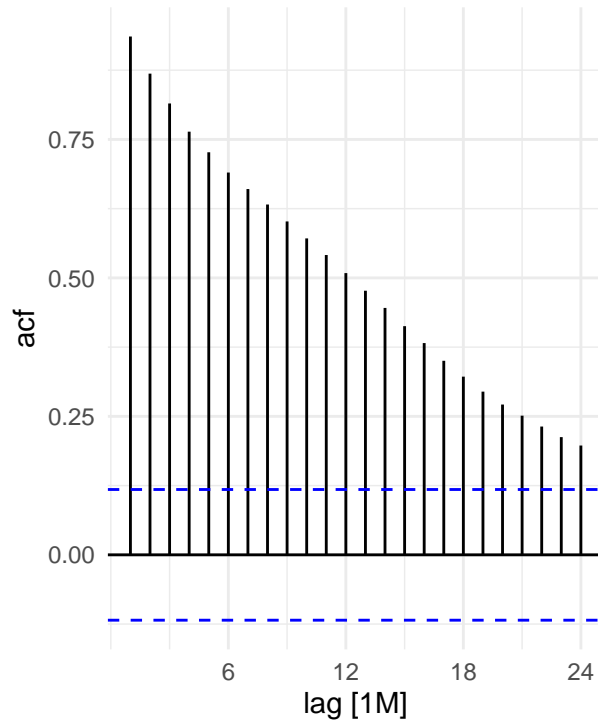


(1 point) Plot the ACF and Lags

This should feel familiar by now: Produce the ACF and lag plot of the `overall` unemployment series. What do you observe?

```
acf.overall <- filter(unemployment, name == "overall") %>% ACF(value) %>% autoplot() + labs(title="ACF plot for unemployment rate")
lag.overall <- filter(unemployment, name == "overall") %>% gg_lag(value, color=1) + labs(title="Lag plot for unemployment rate")
acf.overall | lag.overall
```

ACF plot for unemployment rate



Lag plot for unemployment rate

