W271 Assignment 6 (Solutions)

```
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 a "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 cheetsheet 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
```

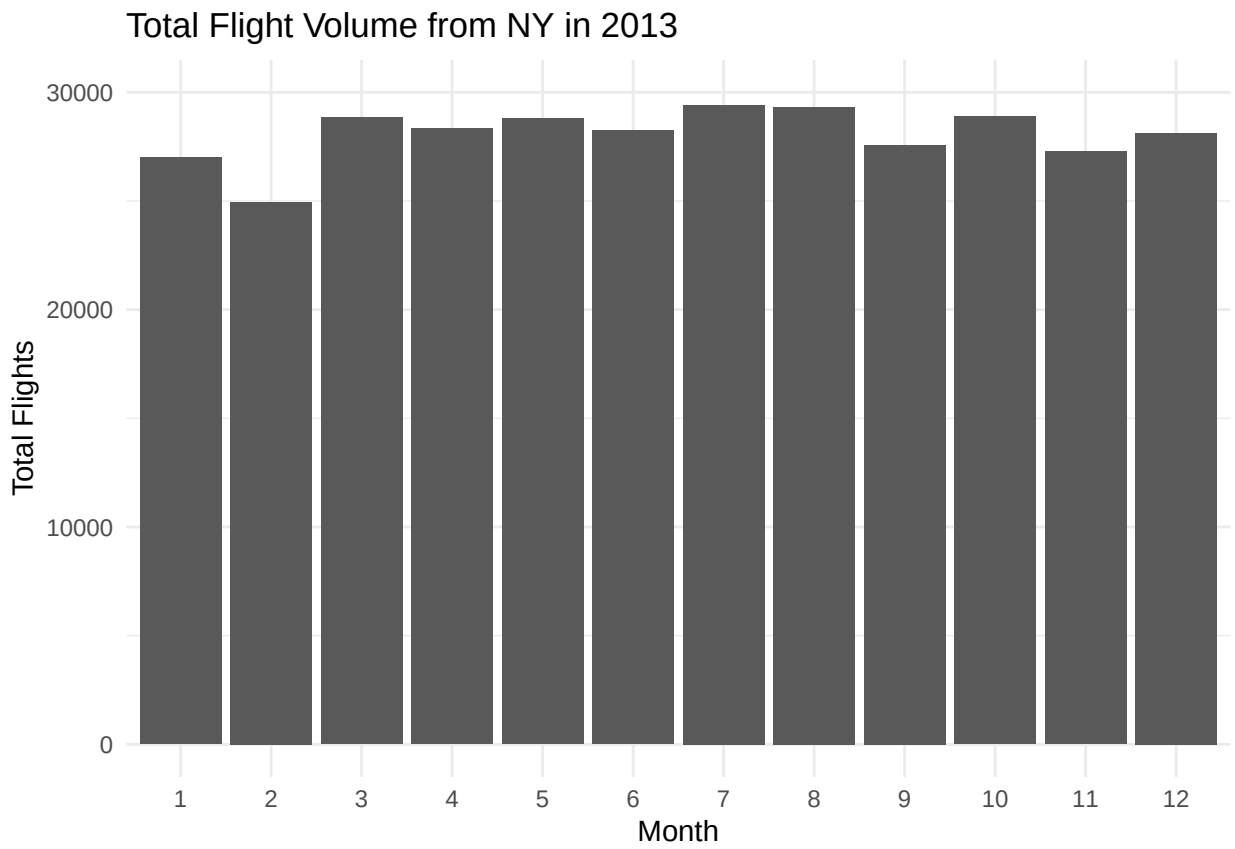
```
## # A tibble: 336,776 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
##  1  2013     1     1      517            515         2      830            819
##  2  2013     1     1      533            529         4      850            830
##  3  2013     1     1      542            540         2      923            850
##  4  2013     1     1      544            545        -1     1004           1022
##  5  2013     1     1      554            600        -6      812            837
##  6  2013     1     1      554            558        -4      740            728
##  7  2013     1     1      555            600        -5      913            854
##  8  2013     1     1      557            600        -3      709            723
##  9  2013     1     1      557            600        -3      838            846
## 10  2013     1     1      558            600        -2      753            745
## # ... with 336,766 more rows, and 11 more variables: arr_delay <dbl>,
## #   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
## #   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>
```

```
flights <- flights %>%
  mutate(time_index = make_datetime(year=year, month=month, day=day, hour=hour, min=minute)) %>%
  as_tsibble(
    key=c(carrier, flight),
    index=time_index)
```

(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.)

```
flights  %>%
  index_by(month)  %>%
  summarise(total_flights = n())  %>%
  ggplot() +
  aes(x=factor(month), y=total_flights) +
  geom_col() +
  lims(y=c(0,30000)) +
  labs(
    title = 'Total Flight Volume from NY in 2013',
    x = 'Month',
    y = 'Total Flights')
```



To be honest, there isn't much to say about these trends. Maybe the most interesting thing to say is that there are fewer flights that happen in February, but otherwise it seems to be pretty consistent.

(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.

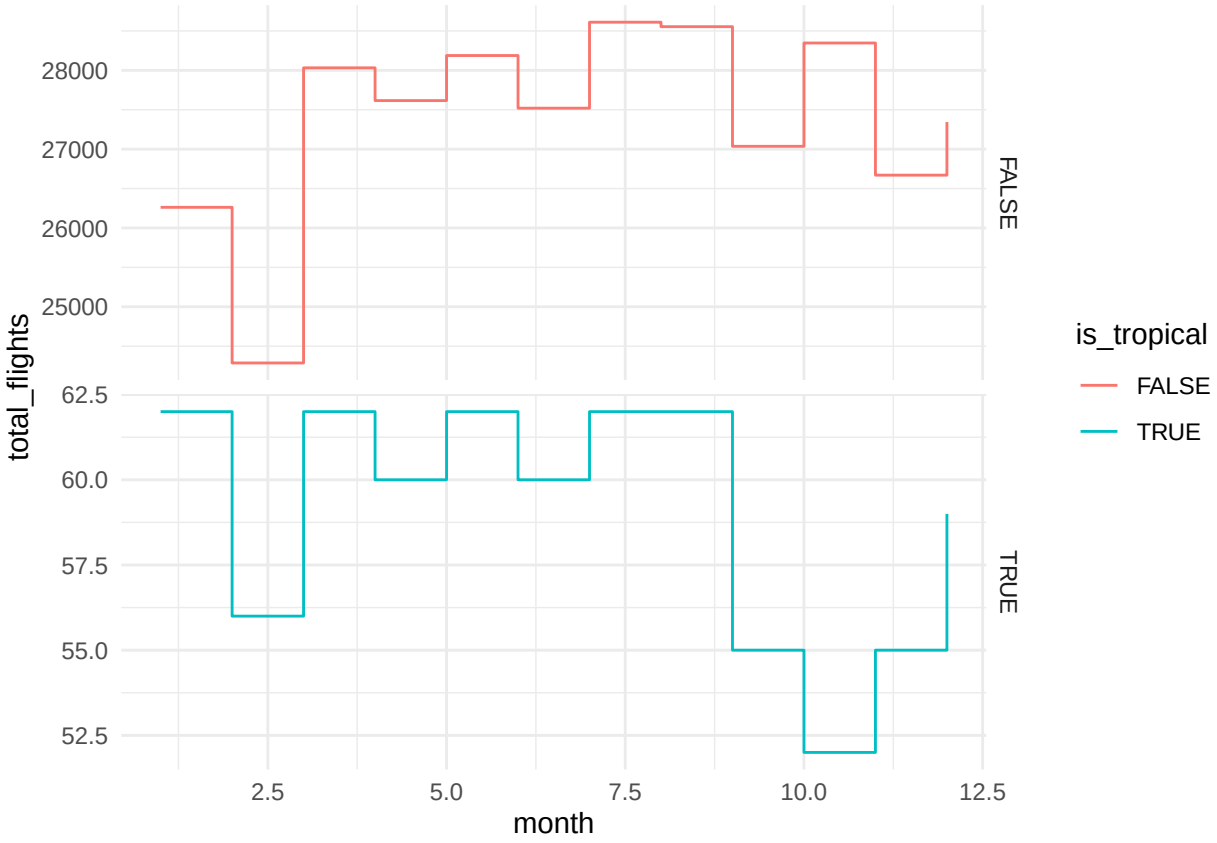
a. First, try to do this using a `group_by` call that groups on `month` and `is_tropical`. Why does this not work? What is happening when grouping by `month` while also having a time index?
b. Instead, you will need to look into `tsibble::index_by` and combine this with a `lubridate` "extractor" to pull 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.

```
flights <- flights %>%
  left_join(airports, by = c('dest' = 'faa')) %>%
  mutate(is_tropical = lat < 23.5)
```

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

```
flights %>%
  filter(!(is.na(is_tropical))) %>%
  group_by(is_tropical) %>%
  index_by(month = month(time_index)) %>%
  summarise(total_flights = n()) %>%
  ggplot() +
  aes(x=month, y=total_flights, color=is_tropical) +
  geom_step() +
  facet_grid(vars(is_tropical), scales="free_y")
```



There isn't *that* much to say for this series either – it seems that there are similar decreases in flights in February to both tropical and non-tropical destinations. The largest difference that we see when looking at this plot is the decrease in flights to tropical destinations in September and October. This trend is different than the trend to non-tropical destinations.

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 structurd 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.
```

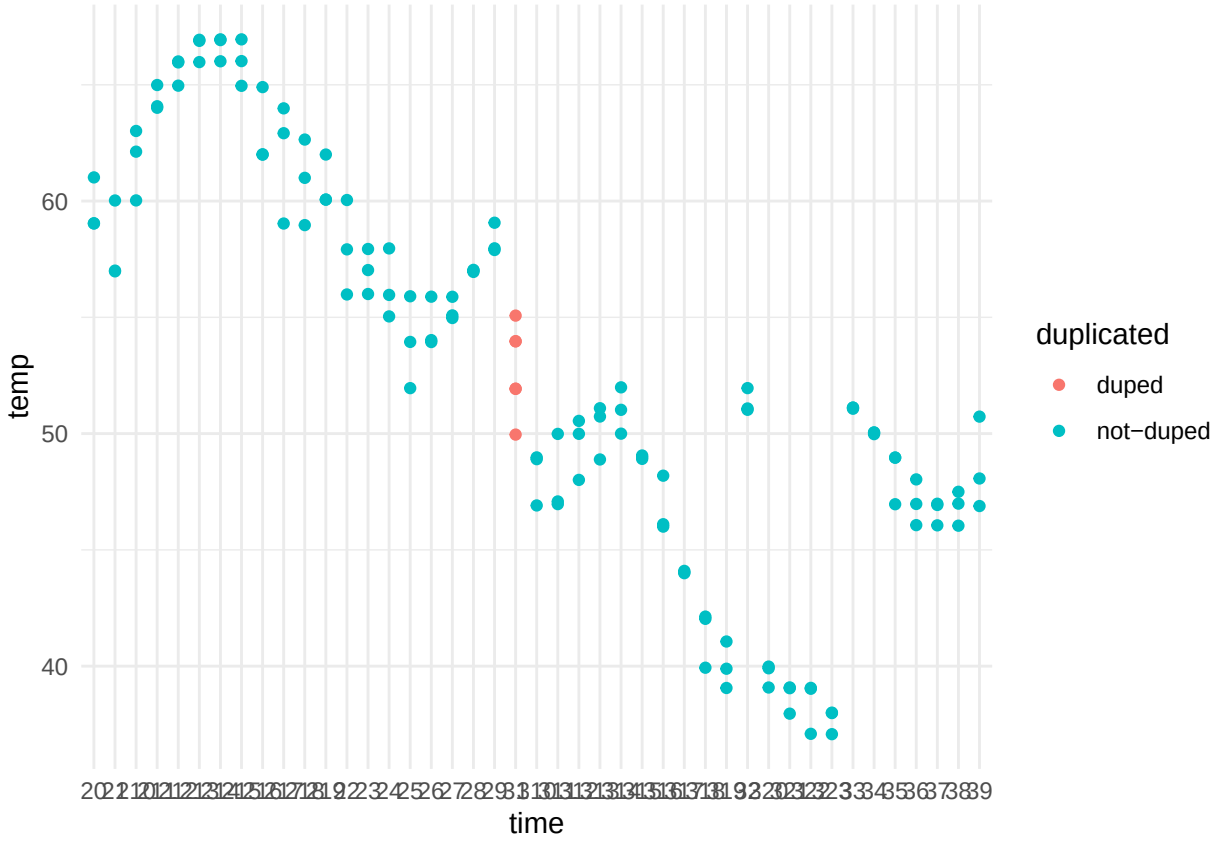
This is a *very* helpful error, with a helpful error message. If you see this error message, we suggest doing as the message suggests, and look into the `duplicates()` function to determine what the issue is. Once you have found the issue, (1) document the issue; (2) propose a solution that seems reasonable;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

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

The hard part about choosing what to do is that we have duplicated information – there is data that is present on November 3rd and 1:00 am that is recorded twice. What is worse, the data doesn't match!

One way of working against this is to look at the data that is nearby and seeing if there is any evidence that one (or another) of the data points is very strange.

```
weather %>%
  filter(month==11, day %in% c(2,3)) %>%
  mutate(
    time = paste0(day, hour),
    duplicated = case_when(
      day == 3 & hour == 1 ~ 'duped',
      TRUE                 ~ 'not-duped')) %>%
  ggplot() +
  aes(x=time,y=temp, color = duplicated) +
  geom_jitter(width=0)
```



Unfortunately, even with this look at the data, there isn't a clear way to make a call about which of these records is the correctly recorded record and which is erroneously recorded. And so, we will make a judgment call – to drop all data that is recorded at this point. Alternatives could be to keep one or another of the points at each origin, but doing so creates the appearance of truth of the point, which we cannot actually vouch for.

```
weather <- weather %>%
  mutate(
    duplicated = case_when(
      month == 11 & day == 3 & hour == 1 ~ 'duped',
      TRUE                   ~ 'not-duped')) %>%
  filter(duplicated == 'not-duped') %>%
  mutate(time_index = make_datetime(year, month, day, hour)) %>%
  as_tsibble(
    key   = origin,
    index = time_index)

weather
```

```
## # A tsibble: 26,109 x 17 [1h] <UTC>
## # Key:       origin [3]
##    origin  year month   day  hour  temp  dewp humid wind_dir wind_speed
##    <chr>  <int> <int> <int> <int> <dbl> <dbl> <dbl>    <dbl>      <dbl>
##  1 EWR     2013     1     1     1  39.0  26.1  59.4      270      10.4
##  2 EWR     2013     1     1     2  39.0  27.0  61.6      250       8.06
##  3 EWR     2013     1     1     3  39.0  28.0  64.4      240      11.5
##  4 EWR     2013     1     1     4  39.9  28.0  62.2      250      12.7
```

```
##  5 EWR     2013     1     1     5  39.0  28.0  64.4      260      12.7
##  6 EWR     2013     1     1     6  37.9  28.0  67.2      240      11.5
##  7 EWR     2013     1     1     7  39.0  28.0  64.4      240      15.0
##  8 EWR     2013     1     1     8  39.9  28.0  62.2      250      10.4
##  9 EWR     2013     1     1     9  39.9  28.0  62.2      260      15.0
## 10 EWR     2013     1     1    10  41    28.0  59.6      260      13.8
## # ... with 26,099 more rows, and 7 more variables: wind_gust <dbl>,
## #   precip <dbl>, pressure <dbl>, visib <dbl>, time_hour <dttm>,
## #   duplicated <chr>, time_index <dttm>
```

With this rather finicky task of removing the duplicated data addressed, we can return to the core goal – making a plot of the temperatures at each of the origins, across time.

(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 <- weather %>%
  ggplot() +
  aes(x=time_index, y=temp, color=origin) +
  geom_line() +
  labs(title = 'Annual Temperature')
```

```
january_plot <- weather %>%
  filter(month(time_index) == 1) %>%
  ggplot() +
  aes(x=time_index, y=temp, color=origin) +
  geom_line() +
  labs(title='January') +
  theme(legend.position="none")
july_plot <- weather %>%
  filter(month(time_index) == 7) %>%
  ggplot() +
  aes(x=time_index, y=temp, color=origin) +
  geom_line() +
  labs(title='July') +
  theme(legend.position="none")
```

```
january_first_week <- weather %>%
  filter(month(time_index) == 1 & week(time_index) == 1) %>%
  ggplot() +
  aes(x=time_index, y=temp, color=origin) +
  geom_line() +
  labs(title='January - Week One') +
  theme(legend.position = "none")
```

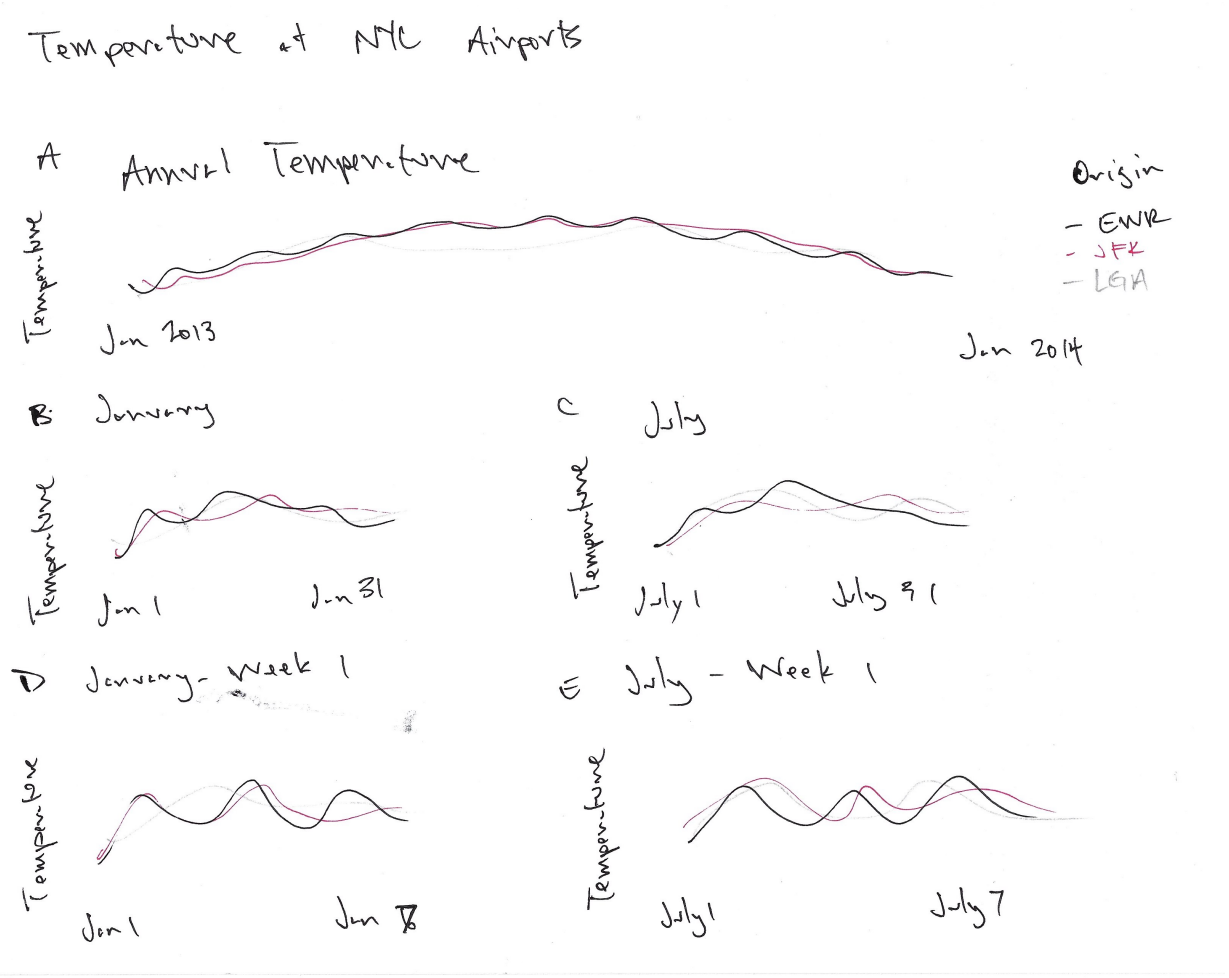
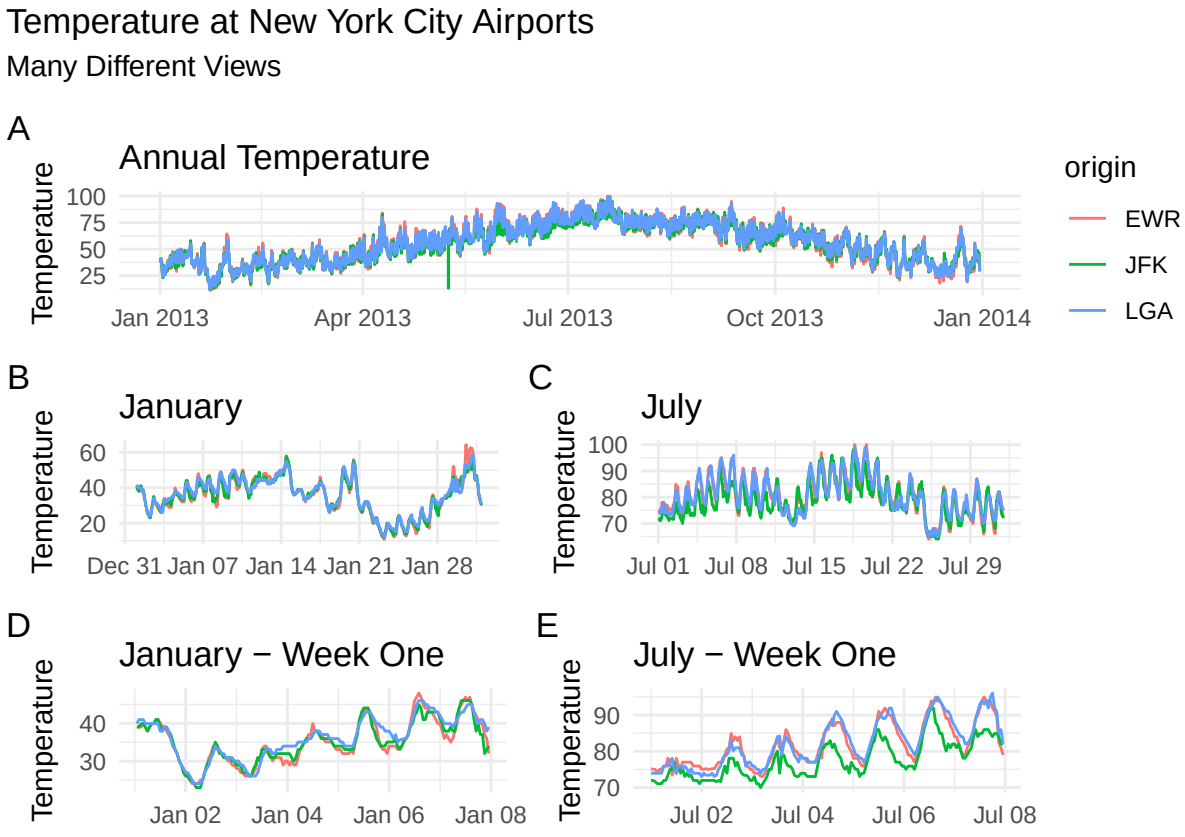


Figure 1: sample plot

```
july_first_week <- weather %>%
  filter(month(time_index) == 7 & day(time_index) %in% 1:7) %>%
  ggplot() +
  aes(x=time_index, y=temp, color=origin) +
  geom_line() +
  labs(title = 'July - Week One') +
  theme(legend.position = "none")
```

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



There are a few notable aspects about each of the plots. First and foremost, the temperatures at each of the airports are all very similar. Second, there is a clear seasonal trend that we anticipated – it is much cooler at New York airports in the winter and much warmer in the summer. This trend is present, despite the daily fluxuation. Third, there a clear random-walk pattern to the data at both the month- and day- resolution. Temperatures fluxuate, but are correlated through time. Fourth, and finally, there is a very clear "seasonal" trend at the daily level too; this also was to be expected due to the diurnal rythems.

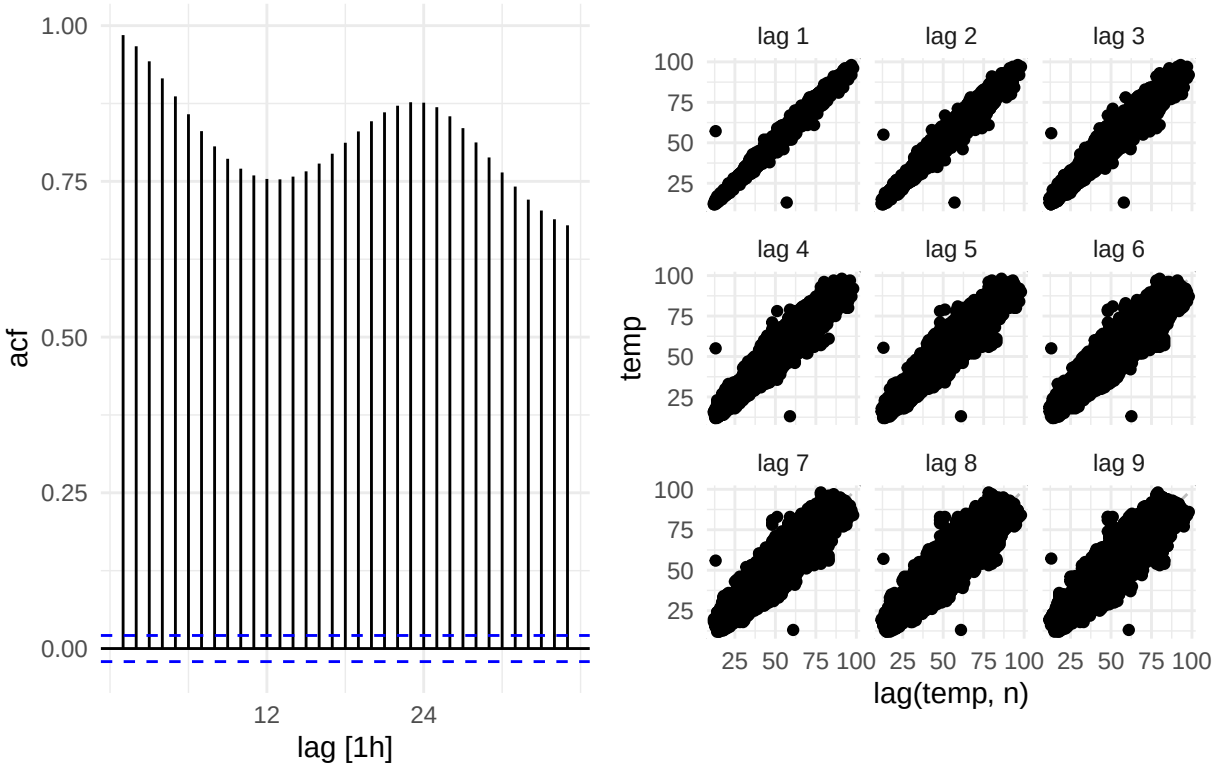
(1 point) Hourly ACF

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`.)

```
hourly_acf <- fill_gaps(weather) %>%
  filter(origin == 'JFK') %>%
  ACF(y=temp) %>%
  autoplot()
hourly_lag <- fill_gaps(weather) %>%
  filter(origin == 'JFK') %>%
  gg_lag(y=temp, geom = 'point', color = 1)
```

```
## Warning: Removed 26 rows containing missing values (gg_lag).
```

```
hourly_acf | hourly_lag
```



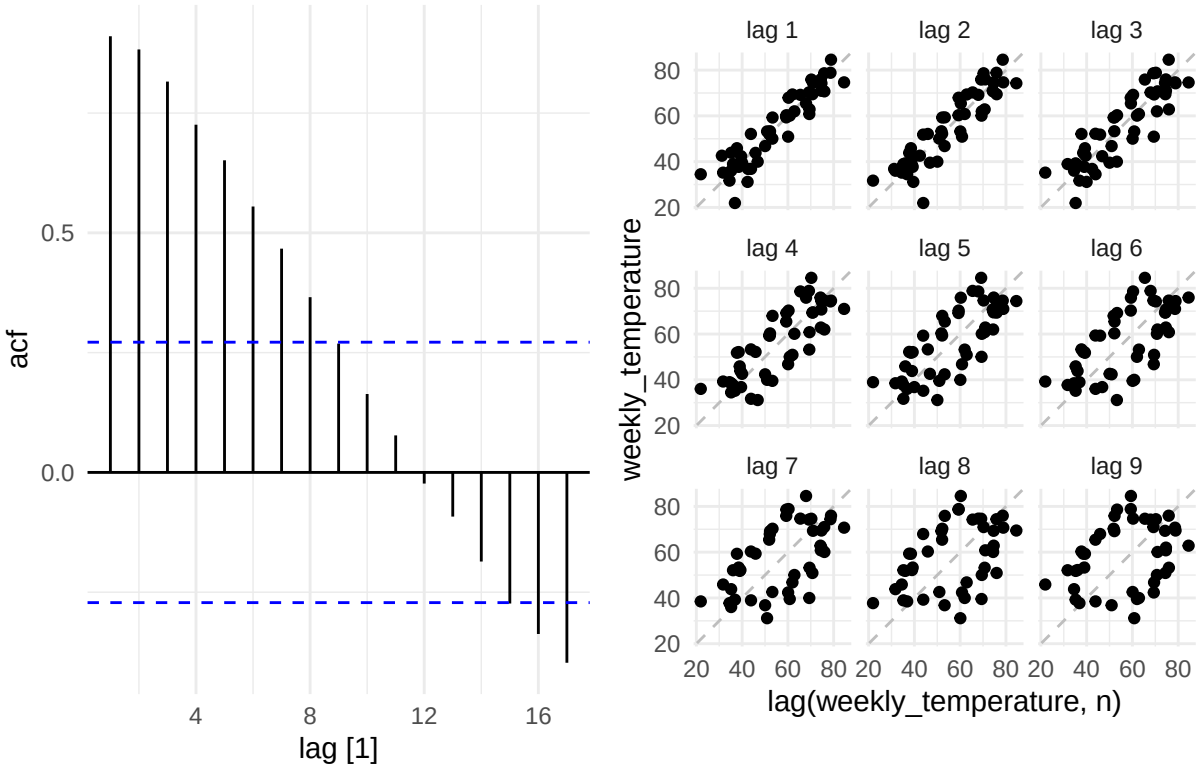
(1 point) Weekly ACF

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_acf <- weather %>%
  filter(origin == 'JFK') %>%
  index_by(week = week(time_index)) %>%
  summarise(weekly_temperature = mean(temp)) %>%
  ACF(y=weekly_temperature) %>%
  autoplot()
```

```
weekly_lag <- weather %>%
  filter(origin == 'JFK') %>%
  index_by(week = week(time_index)) %>%
  summarise(weekly_temperature = mean(temp)) %>%
  gg_lag(y=weekly_temperature, geom='point', color=1)

weekly_acf | weekly_lag
```

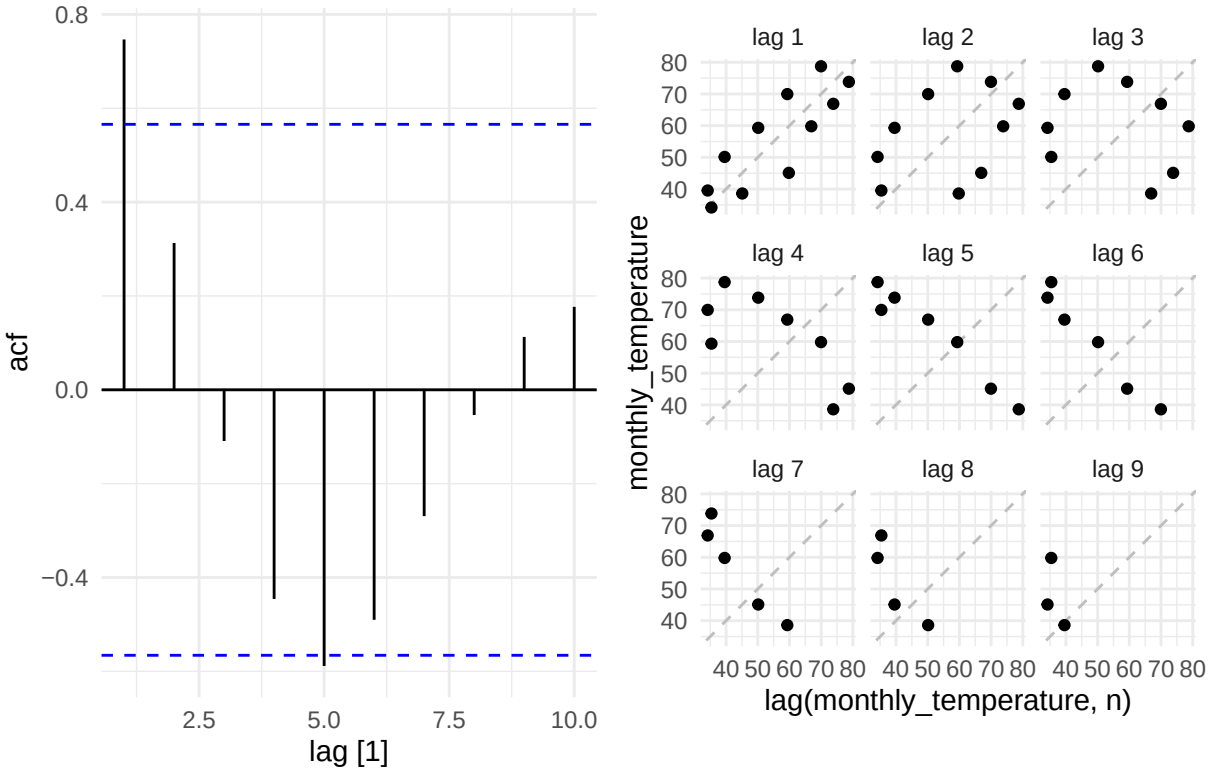


(1 point) Monthly ACF

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

```
monthly_acf <- weather %>%
  filter(origin == 'JFK') %>%
  index_by(month = month(time_index)) %>%
  summarise(monthly_temperature = mean(temp)) %>%
  ACF(y=monthly_temperature) %>%
  autoplot()
monthly_lag <- weather %>%
  filter(origin == 'JFK') %>%
  index_by(month = month(time_index)) %>%
  summarise(monthly_temperature = mean(temp)) %>%
  gg_lag(y=monthly_temperature, geom = 'point', color = 1)

monthly_acf | monthly_lag
```



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 you 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()
```

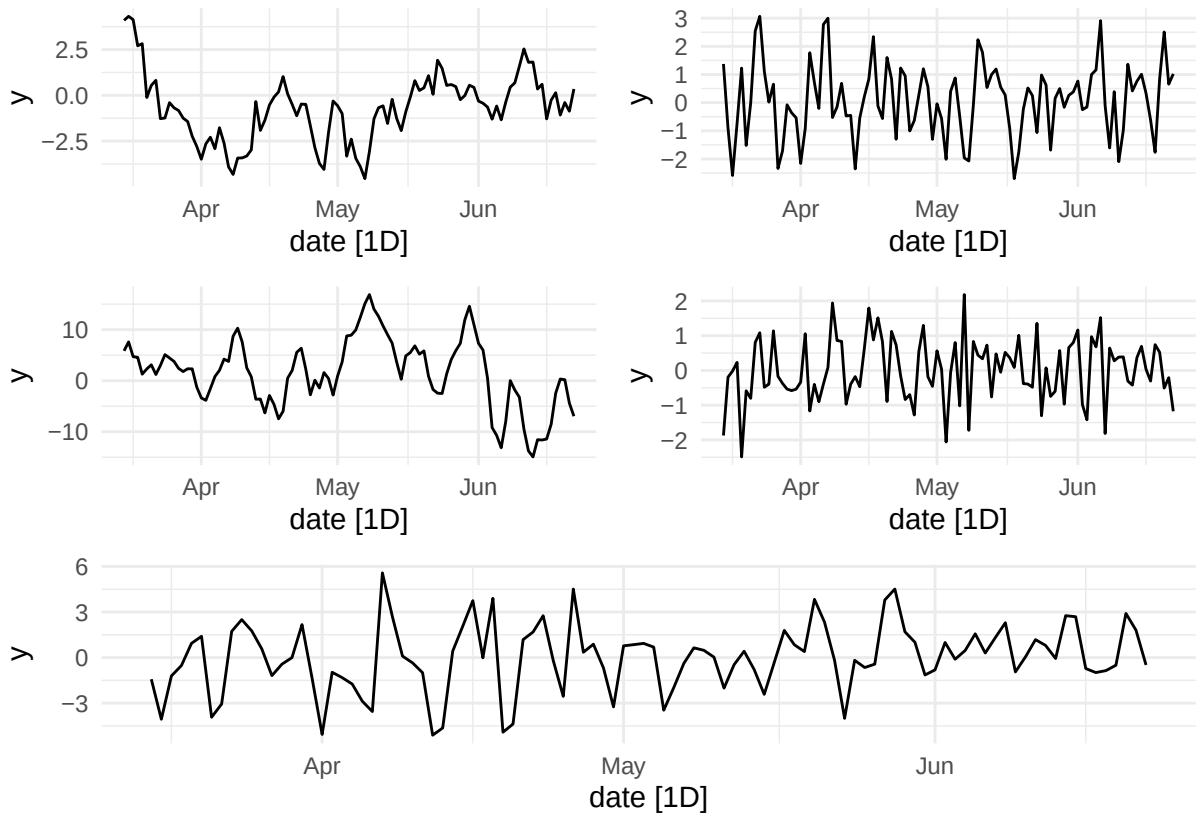
```
## Using ‘date‘ as index variable.
## Using ‘date‘ as index variable.
## Using ‘date‘ as index variable.
```

```
## 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.
```

```
## remove this before shipping to students.
(autoplot(dataset_1) + autoplot(dataset_2)) /
  (autoplot(dataset_3) + autoplot(dataset_4)) /
  (autoplot(dataset_5))
```

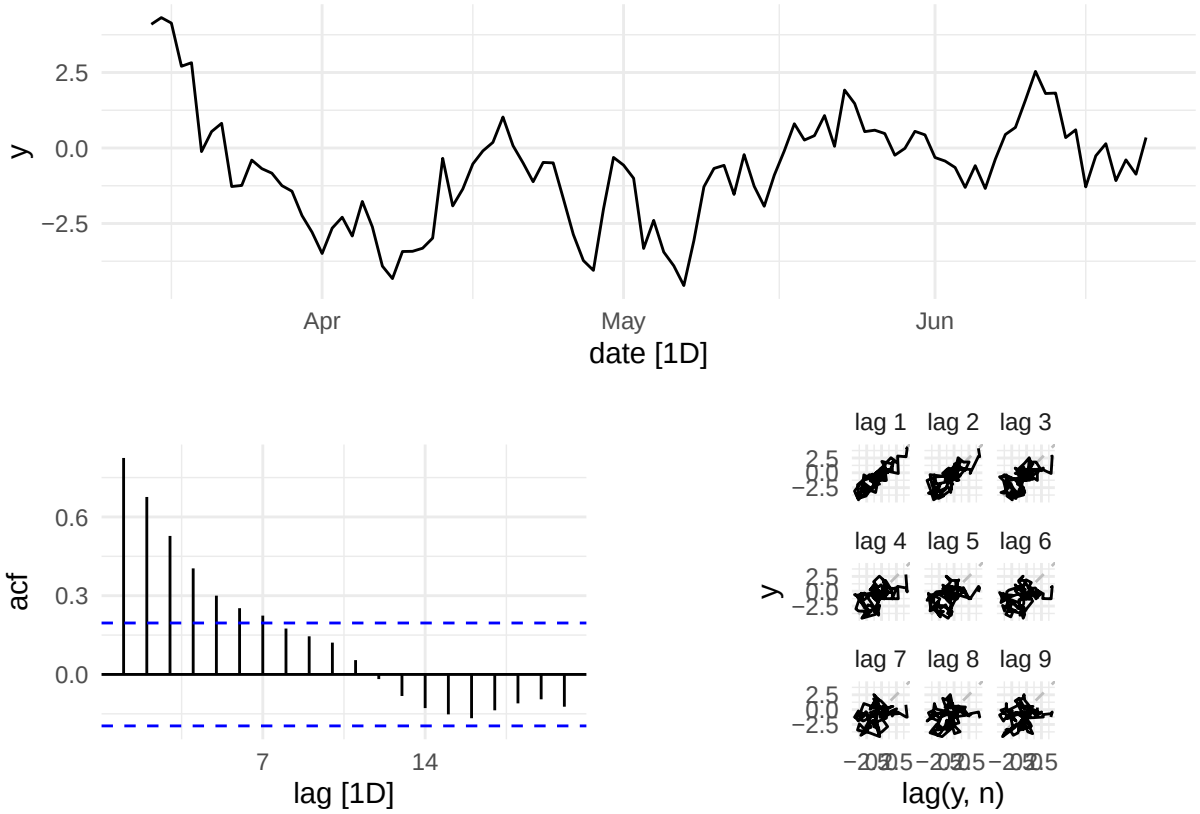
```
## Plot variable not specified, automatically selected ‘.vars = y‘
## Plot variable not specified, automatically selected ‘.vars = y‘
## Plot variable not specified, automatically selected ‘.vars = y‘
## Plot variable not specified, automatically selected ‘.vars = y‘
## Plot variable not specified, automatically selected ‘.vars = y‘
```



(1 point) Dataset One

```
autoplot(dataset_1) /
  (autoplot(ACF(dataset_1)) | gg_lag(dataset_1, color = 1))
```

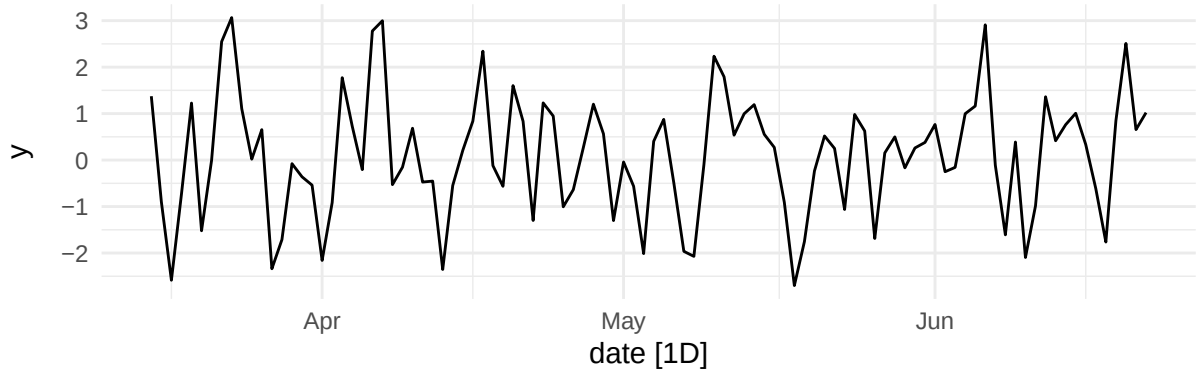
```
## Plot variable not specified, automatically selected `.vars = y`
## Response variable not specified, automatically selected `var = y`
## Plot variable not specified, automatically selected `y = y`
```

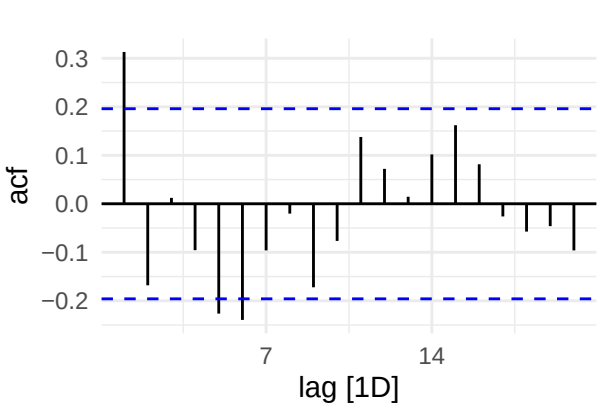


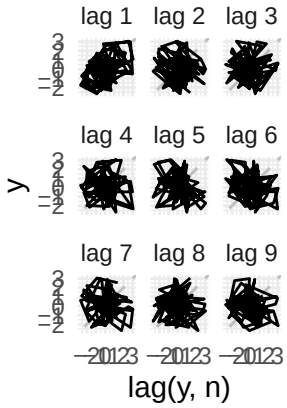
(1 point) Dataset Two

```
autoplot(dataset_2) /
  (autoplot(ACF(dataset_2)) | gg_lag(dataset_2, color = 1))
```

```
## Plot variable not specified, automatically selected `.vars = y`
## Response variable not specified, automatically selected `var = y`
## Plot variable not specified, automatically selected `y = y`
```



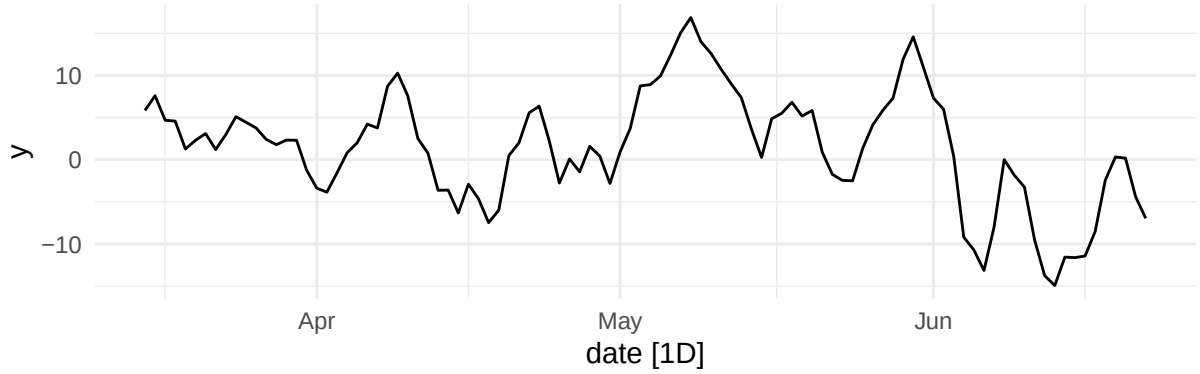


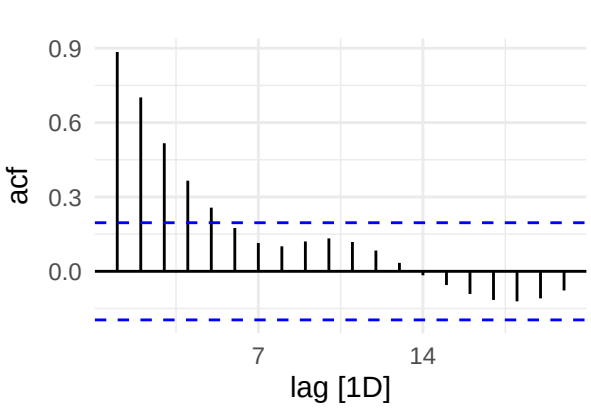


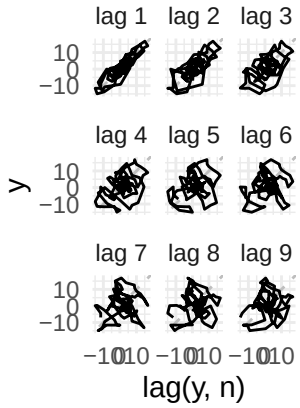
(1 point) Dataset Three

```
autoplot(dataset_3) /
  (autoplot(ACF(dataset_3)) | gg_lag(dataset_3, color = 1))
```

```
## Plot variable not specified, automatically selected `.vars = y`
## Response variable not specified, automatically selected `var = y`
## Plot variable not specified, automatically selected `y = y`
```



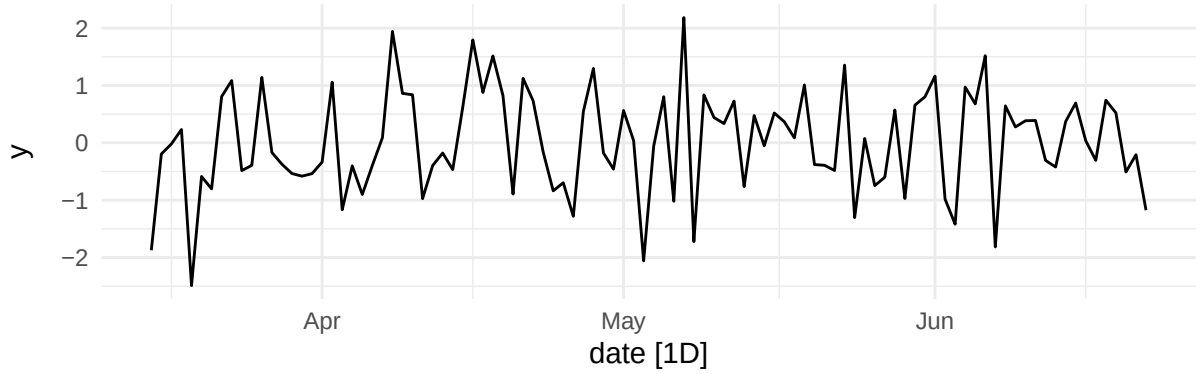


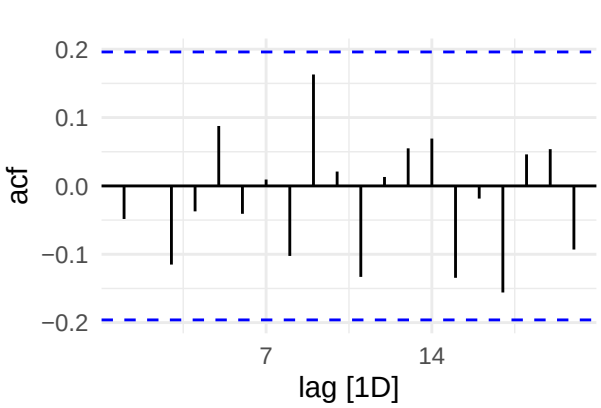


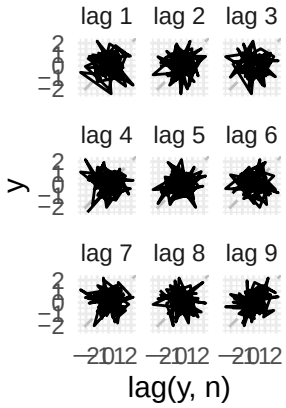
(1 point) Dataset Four

```
autoplot(dataset_4) /
  (autoplot(ACF(dataset_4)) | gg_lag(dataset_4, color = 1))
```

```
## Plot variable not specified, automatically selected `.vars = y`
## Response variable not specified, automatically selected `var = y`
## Plot variable not specified, automatically selected `y = y`
```



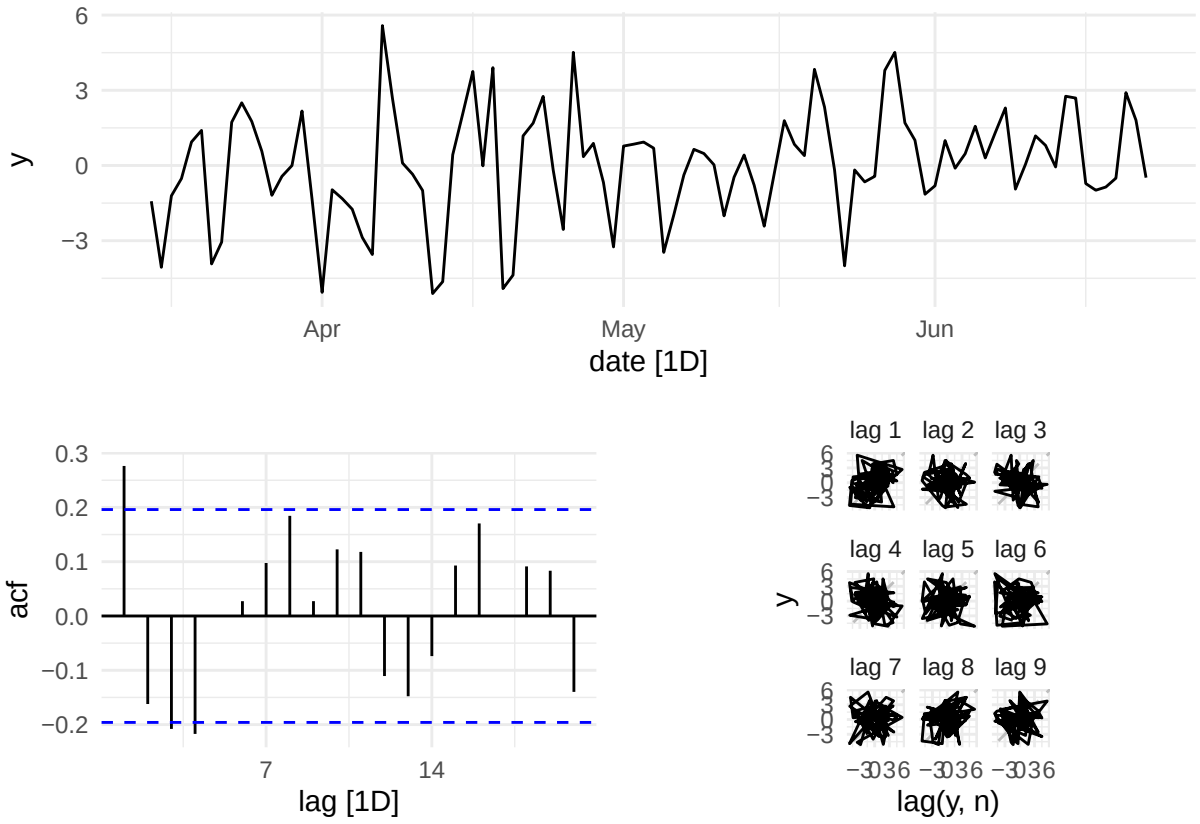




(1 point) Dataset Five

```
autoplot(dataset_5) /
  (autoplot(ACF(dataset_5)) | gg_lag(dataset_5, color = 1))
```

```
## Plot variable not specified, automatically selected `.vars = y`
## Response variable not specified, automatically selected `var = y`
## Plot variable not specified, automatically selected `y = y`
```



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 `Retreive 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
```

```
unemployment <- get_n_series_table(
  series_ids=list(
    overall='LNS14000000',
    male='LNS14000001',
    female='LNS14000002'),
  api_key = '21c01016e3a14d2888519292883a447a',
  start_year=2000,
  end_year=2023,
  tidy = TRUE
)
```

```
unemployment <- unemployment %>%
  mutate(time_index = make_datetime(year,month)) %>%
  mutate(time_index = yearmonth(time_index)) %>%
  as_tsibble(index=time_index) %>%
  pivot_longer(
    cols=c('overall', 'male', 'female'),
    names_prefix='unemployment')
```

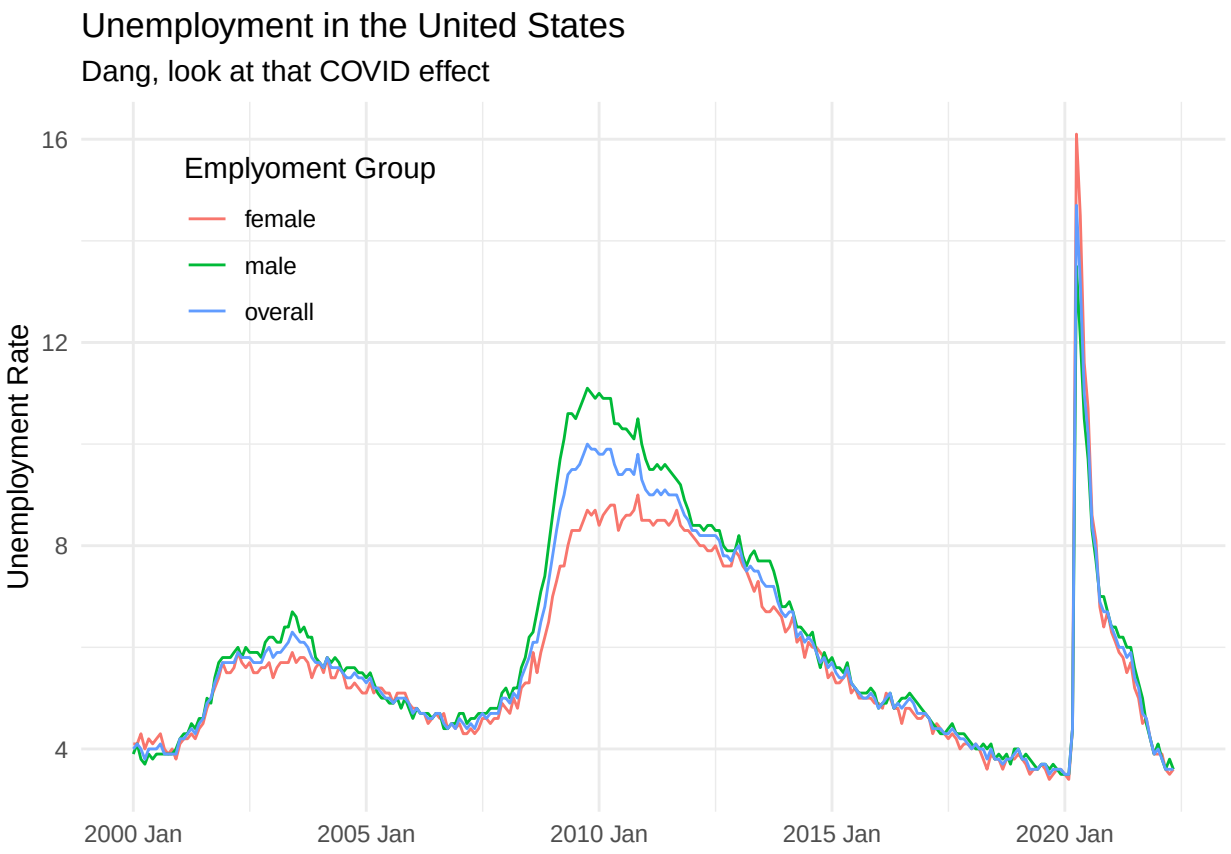
```
head(unemployment)
```

```
## # A tsibble: 6 x 5 [1M]
## # Key:       name [3]
##    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
```

(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.

```
unemployment %>%
  ggplot() +
  aes(x=yearmonth(time_index),y=value, color=name) +
  geom_line() +
  labs(
    title = 'Unemployment in the United States',
    subtitle = 'Dang, look at that COVID effect',
    x = NULL, y = 'Unemployment Rate',
    color = 'Emplyoment Group') +
  theme(legend.position = c(.2,.8))
```

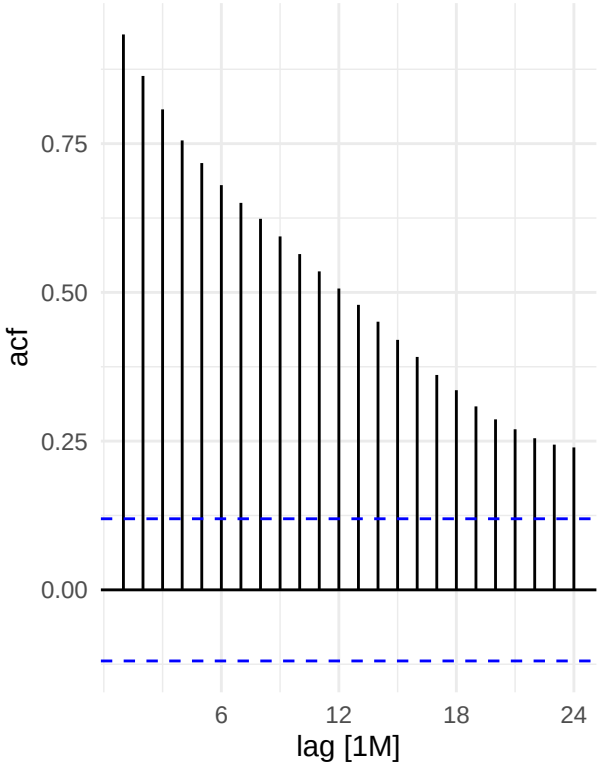


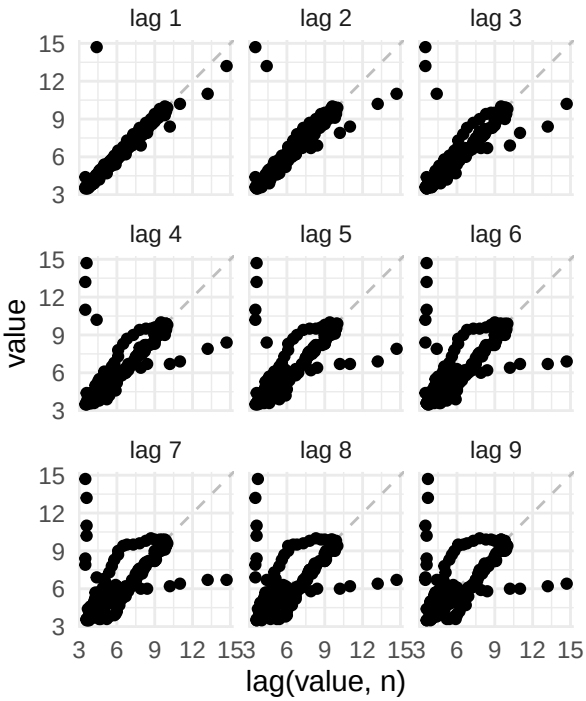
(1 point) Plot the ACF and Lags

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

```
overall_acf <- unemployment %>%
  filter(name == 'overall') %>%
  ACF(y=value) %>%
  autoplot()
overall_lag <- unemployment %>%
  filter(name == 'overall') %>%
  gg_lag(y=value, geom='point', color = 1)

overall_acf | overall_lag
```


acf
0.75
0.50
0.25
0.00
6
12
18
24
lag [1M]


lag 1
lag 2
lag 3
lag 4
lag 5
lag 6
lag 7
lag 8
lag 9
value
15
12
9
6
3
3 6 9 12 15
lag(value, n)