W271 Assignment 6

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
## # i 336,766 more rows
## # i 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>
```

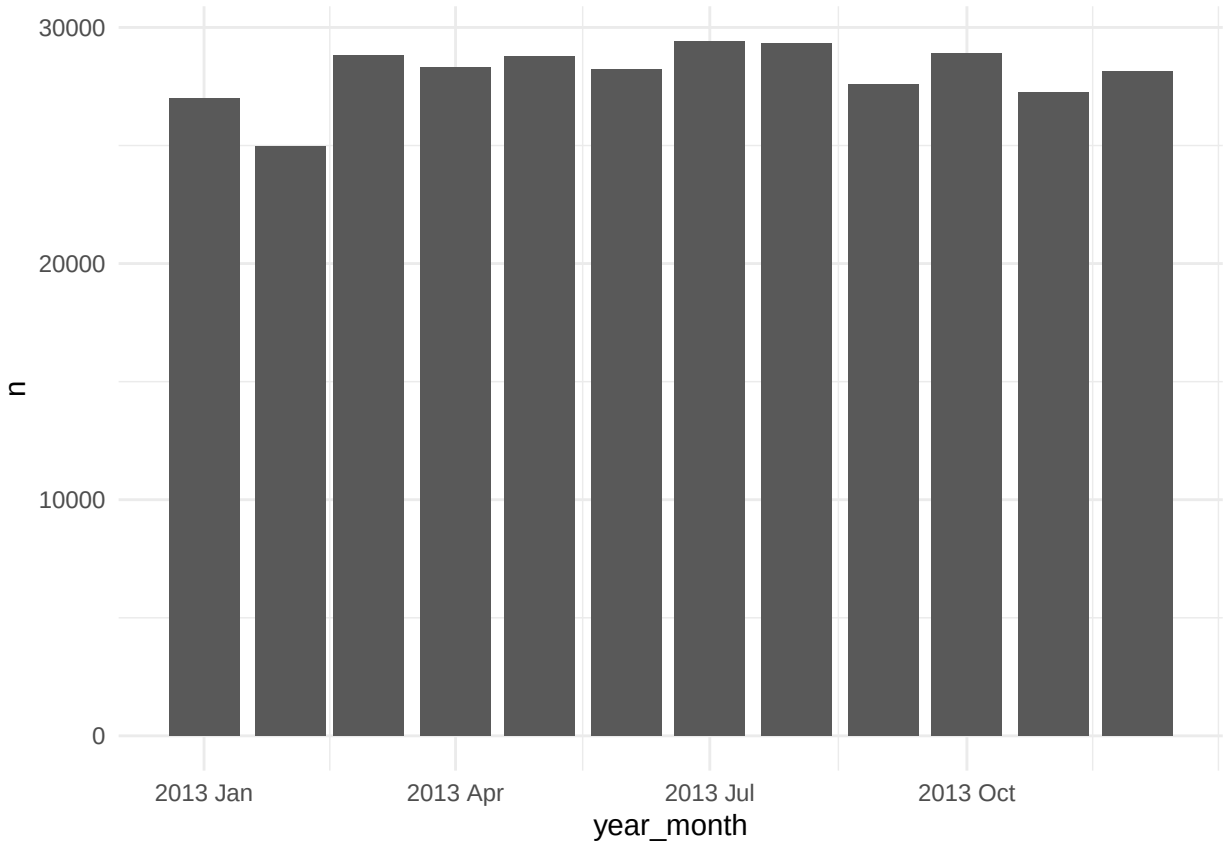
```
flights$time_index <- as_datetime(1511870400) #Set a random time for all entries
year(flights$time_index)<-flights$year #Assign year etc into time index
month(flights$time_index)<-flights$month
day(flights$time_index)<-flights$day
hour(flights$time_index)<-flights$hour
minute(flights$time_index)<-flights$minute
```

```
flights_ts <- as_tsibble(flights, key = (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_ts  %>%
  index_by(year_month = ~ yearmonth(.)) %>% # monthly aggregates
  tally() %>%
  ggplot() +
        aes(x = year_month, y = n) +
        geom_col()
```



(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.

```
airports$is_tropical <- (airports$lat< 23.43624)
flights_df <- merge(x = flights_ts, y = airports, by.x = "dest", by.y = "faa", all.x = FALSE)
```

```
flights_df  %>%
 group_by(month, is_tropical)  %>%
```

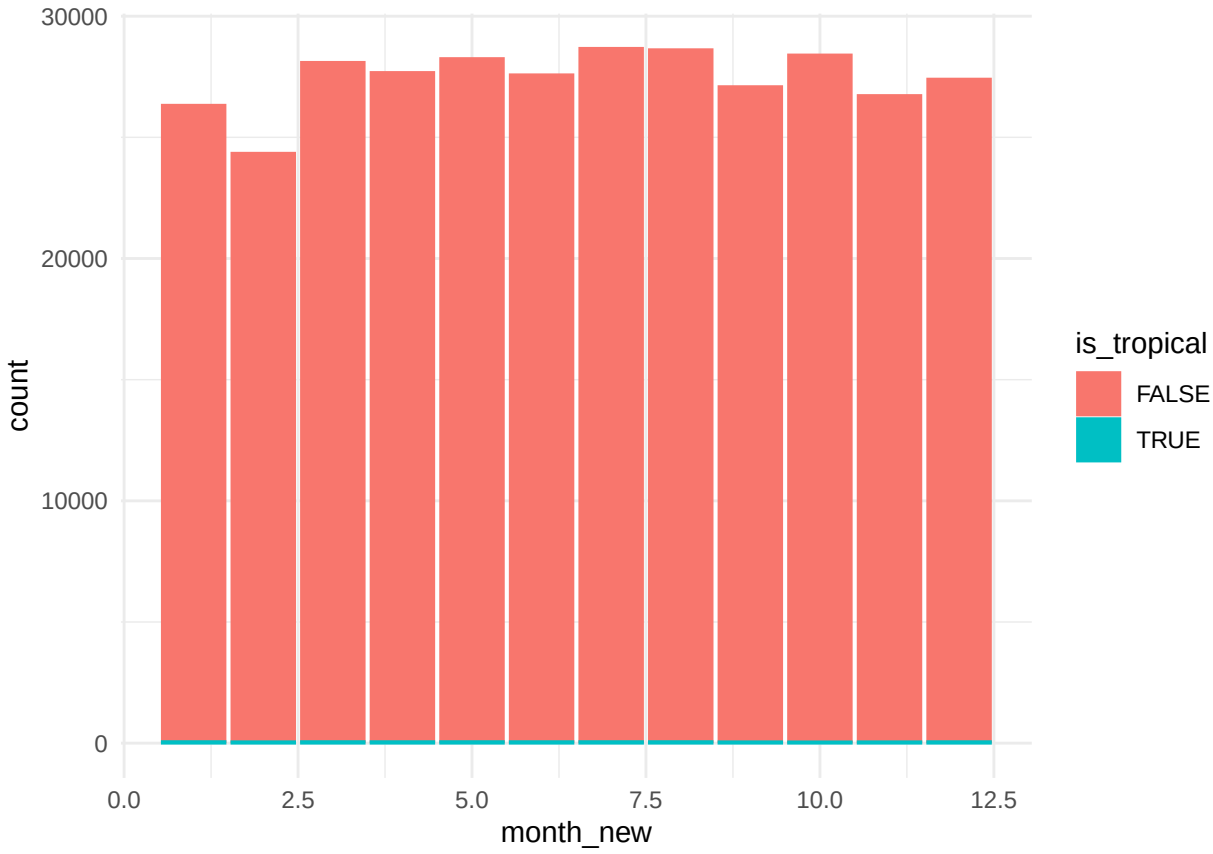
```
 summarise(total_flights = n())
```

```
## # A tibble: 24 x 3
## # Groups:   month [12]
##    month is_tropical total_flights
##    <int> <lgl>               <int>
##  1     1 FALSE               26262
##  2     1 TRUE                   62
##  3     2 FALSE               24287
##  4     2 TRUE                   56
##  5     3 FALSE               28032
##  6     3 TRUE                   62
##  7     4 FALSE               27616
##  8     4 TRUE                   60
##  9     5 FALSE               28188
## 10     5 TRUE                   62
## # i 14 more rows
```

```
flights_ts_new <- as_tsibble(flights_df, key = (carrier|flight), index = time_index)
flights_ts_new  %>%
 group_by(month, is_tropical)  %>%
 summarise(total_flights = n())
```

```
## # A tsibble: 125,719 x 4 [1m] <UTC>
## # Key:       month, is_tropical [24]
## # Groups:    month [12]
##    month is_tropical time_index          total_flights
##    <int> <lgl>       <dttm>                      <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
## # i 125,709 more rows
```

```
flights_ts_new  %>%
 index_by(month_new = ~month(.))%>%
  ggplot() + aes(x = month_new, color = is_tropical, fill = is_tropical) +
  geom_bar()
```



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 <- nycflights13::weather
weather$time_index <- make_datetime(year = weather$year,
                                    month= weather$month,
                                    day = weather$day,
```

```
                                    hour = weather$hour)
summary(weather)
```

```
##     origin               year          month             day
##  Length:26115       Min.   :2013   Min.   : 1.000   Min.   : 1.00
##  Class :character   1st Qu.:2013   1st Qu.: 4.000   1st Qu.: 8.00
##  Mode  :character   Median :2013   Median : 7.000   Median :16.00
##                     Mean   :2013   Mean   : 6.504   Mean   :15.68
##                     3rd Qu.:2013   3rd Qu.: 9.000   3rd Qu.:23.00
##                     Max.   :2013   Max.   :12.000   Max.   :31.00
##
##       hour            temp             dewp           humid
##  Min.   : 0.00   Min.   : 10.94   Min.   :-9.94   Min.   : 12.74
##  1st Qu.: 6.00   1st Qu.: 39.92   1st Qu.:26.06   1st Qu.: 47.05
##  Median :11.00   Median : 55.40   Median :42.08   Median : 61.79
##  Mean   :11.49   Mean   : 55.26   Mean   :41.44   Mean   : 62.53
##  3rd Qu.:17.00   3rd Qu.: 69.98   3rd Qu.:57.92   3rd Qu.: 78.79
##  Max.   :23.00   Max.   :100.04   Max.   :78.08   Max.   :100.00
##                  NA's   :1        NA's   :1       NA's   :1
##     wind_dir       wind_speed        wind_gust         precip
##  Min.   :  0.0   Min.   :   0.000   Min.   :16.11   Min.   :0.000000
##  1st Qu.:120.0   1st Qu.:   6.905   1st Qu.:20.71   1st Qu.:0.000000
##  Median :220.0   Median :  10.357   Median :24.17   Median :0.000000
##  Mean   :199.8   Mean   :  10.518   Mean   :25.49   Mean   :0.004469
##  3rd Qu.:290.0   3rd Qu.:  13.809   3rd Qu.:28.77   3rd Qu.:0.000000
##  Max.   :360.0   Max.   :1048.361   Max.   :66.75   Max.   :1.210000
##  NA's   :460     NA's   :4          NA's   :20778
##     pressure          visib          time_hour
##  Min.   : 983.8   Min.   : 0.000   Min.   :2013-01-01 01:00:00.0
##  1st Qu.:1012.9   1st Qu.:10.000   1st Qu.:2013-04-01 21:30:00.0
##  Median :1017.6   Median :10.000   Median :2013-07-01 14:00:00.0
##  Mean   :1017.9   Mean   : 9.255   Mean   :2013-07-01 18:26:37.7
##  3rd Qu.:1023.0   3rd Qu.:10.000   3rd Qu.:2013-09-30 13:00:00.0
##  Max.   :1042.1   Max.   :10.000   Max.   :2013-12-30 18:00:00.0
##  NA's   :2729
##    time_index
##  Min.   :2013-01-01 01:00:00.00
##  1st Qu.:2013-04-01 21:30:00.00
##  Median :2013-07-01 14:00:00.00
##  Mean   :2013-07-01 18:05:51.25
##  3rd Qu.:2013-09-30 13:00:00.00
##  Max.   :2013-12-30 18:00:00.00
##
```

```
#The command below fails as described in the prompt
# weather_ts <- as_tsibble(weather, key = origin, index = time_index)
```

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

```
## # A tibble: 6 x 16
##   origin  year month   day  hour  temp  dewp humid wind_dir wind_speed wind_gust
##   <chr>  <int> <int> <int> <int> <dbl> <dbl> <dbl>    <dbl>      <dbl>     <dbl>
## 1 EWR     2013    11     3     1  52.0  39.0  61.2      310       6.90        NA
```

```
## 2 EWR     2013    11     3     1  50    39.0  65.8      290       5.75        NA
## 3 JFK     2013    11     3     1  54.0  37.9  54.5      320       9.21        NA
## 4 JFK     2013    11     3     1  52.0  37.9  58.6      310       6.90        NA
## 5 LGA     2013    11     3     1  55.0  39.0  54.7      330       9.21        NA
## 6 LGA     2013    11     3     1  54.0  39.9  58.9      310       8.06        NA
## # i 5 more variables: precip <dbl>, pressure <dbl>, visib <dbl>,
## #   time_hour <dttm>, time_index <dttm>
```

It appears that there are few duplicates of the data that were taken at the same time, at the same location. The observed values are similar and valid. Possibly, they are from different sesnors or taken within the same hour. A reasonable approach would be to replace groups of repetitive observations with their median.

```
weather_clean <- weather %>%
 group_by(origin, time_index)  %>%
 summarise(across(1:14, ~median(.x)))
```

Now this dataframe can be coerced into tsibble:

```
weather_ts <- as_tsibble(weather_clean, key = origin, index = time_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 <- weather_ts  %>%
  ggplot() + aes(x = time_index, y = temp, color = origin) +
  geom_line()
```

```
january_plot <- weather_ts %>% filter_index("2013-01-01" ~ "2013-01-31")  %>%
  ggplot() + aes(x = time_index, y = temp, color = origin) +
  geom_line()

july_plot <-   weather_ts %>% filter_index("2013-07-01" ~ "2013-07-31")  %>%
  ggplot() + aes(x = time_index, y = temp, color = origin) +
  geom_line()
```

```
january_first_week <- weather_ts %>% filter_index("2013-01-01" ~ "2013-01-07")  %>%
  ggplot() + aes(x = time_index, y = temp, color = origin) +
  geom_line()
july_first_week <- weather_ts %>% filter_index("2013-07-01" ~ "2013-07-07")  %>%
  ggplot() + aes(x = time_index, y = temp, color = origin) +
  geom_line()
```

```
yearly_plot /
   (january_plot | july_plot) /
```

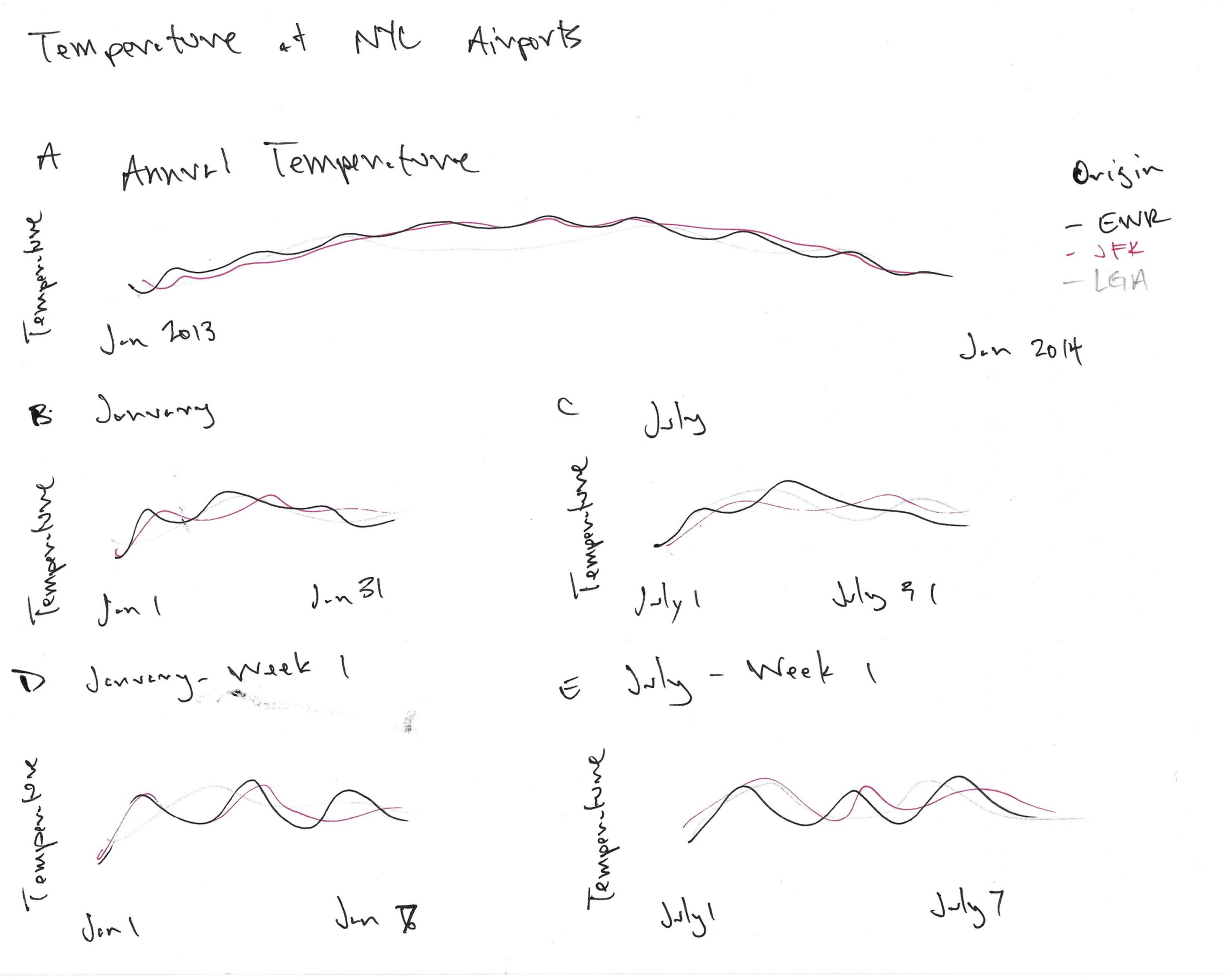
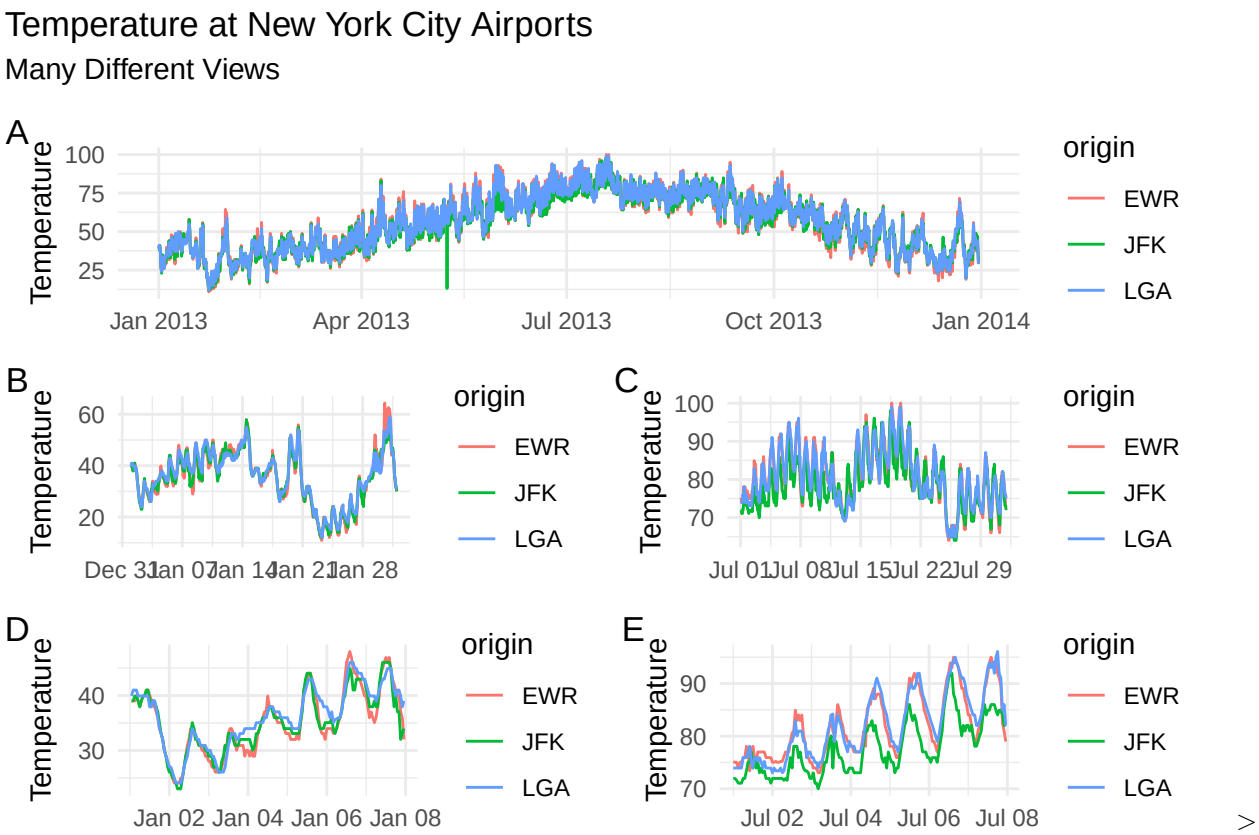


Figure 1: sample plot

```
  (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')
```



Plots offer few surprises. On the scale of a year, tempearture raises around July and ges down toward December. On the scale of a month heat waves and cold spells become apparent. Interestingly, on the weekly scale it becomes obvious that in the summer JFK experiences generally coller temps, as compared to La Guardia and Newark. This is probably becasue JFK is located right on the shores of New York bay, while the other two airports are further away from large bodies of water. It is also apprent that dayly variability is mauch lower in the winter then in the summer, possibly becasue the teperatures are much closerto the phase transition temperature of water.

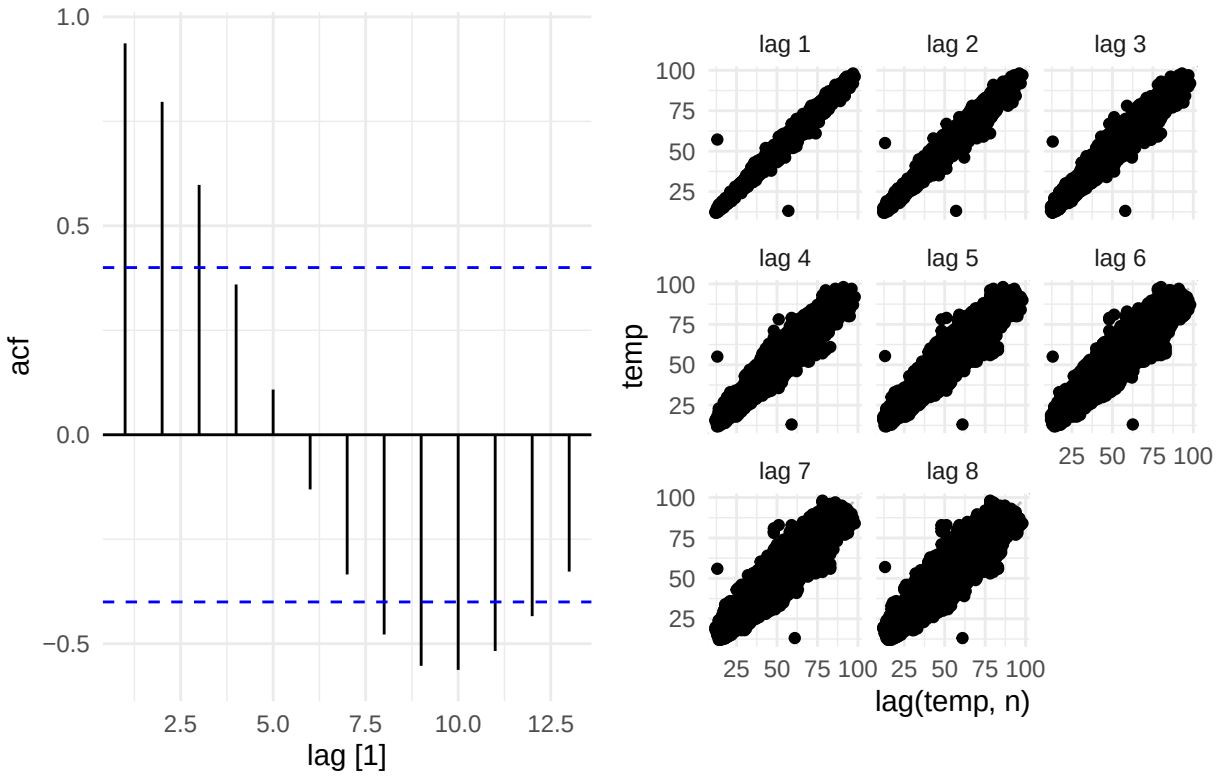
(1 point) Hourly ACF

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 <- weather_ts %>%filter(origin == "JFK") %>%
                           index_by(hour_new = ~hour(.) )%>%
                           summarise(temp = mean(temp)) %>%
                           ACF(temp) %>% autoplot()

hourly_lag <- weather_ts %>%filter(origin == "JFK") %>%
  gg_lag(temp, lag = 1:8, geom = "point", color = 1)
```

```
hourly_acf | hourly_lag
```



Both ACF and lag plots show strong correlation between consequtive hourly measurements. The correlation is becoming weaker as the lag increases.

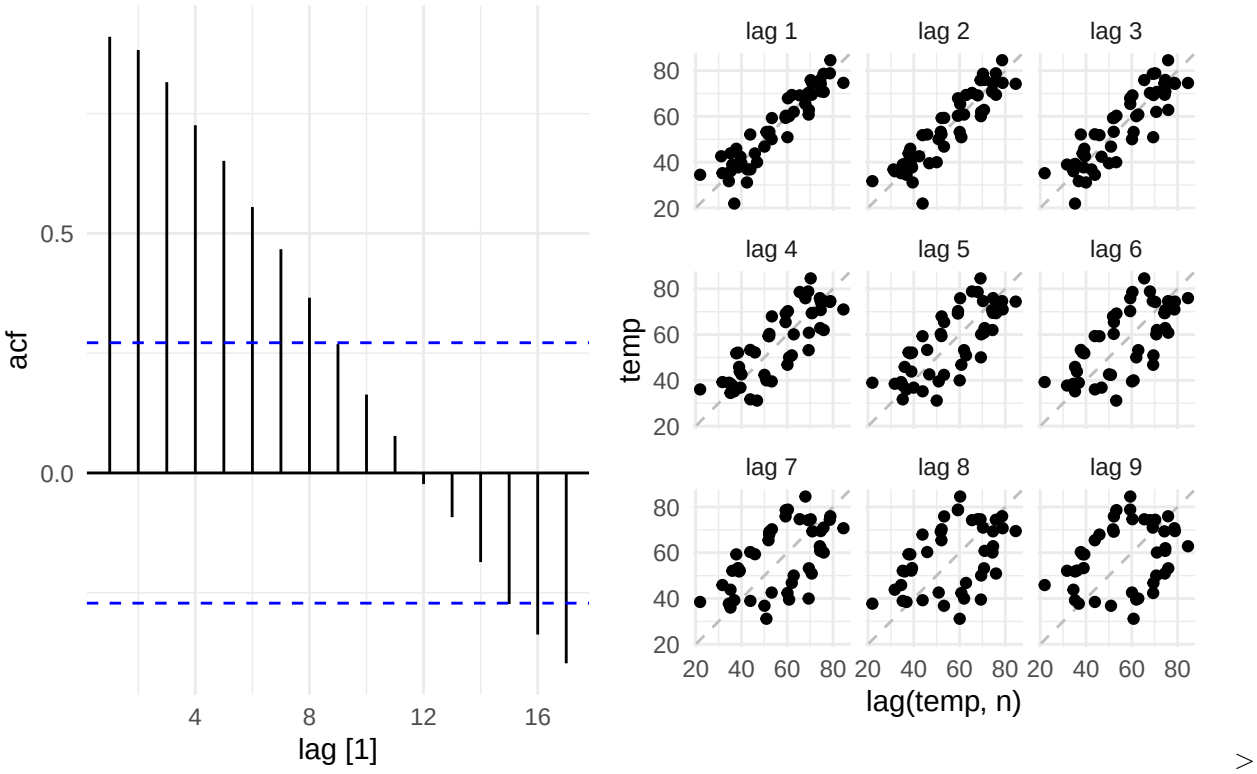
(1 point) Weekly ACF

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_ts %>% filter(origin == "JFK") %>%
 index_by(week = ~week(.))%>% summarise(temp = mean(temp)) %>% ACF(temp) %>% autoplot()

weekly_lag <- weather_ts %>% filter(origin == "JFK") %>%
 index_by(week = ~week(.)) %>% summarise(temp = mean(temp))%>%
  gg_lag(temp, lag = 1:9, geom = "point", color = 1)

weekly_acf | weekly_lag
```



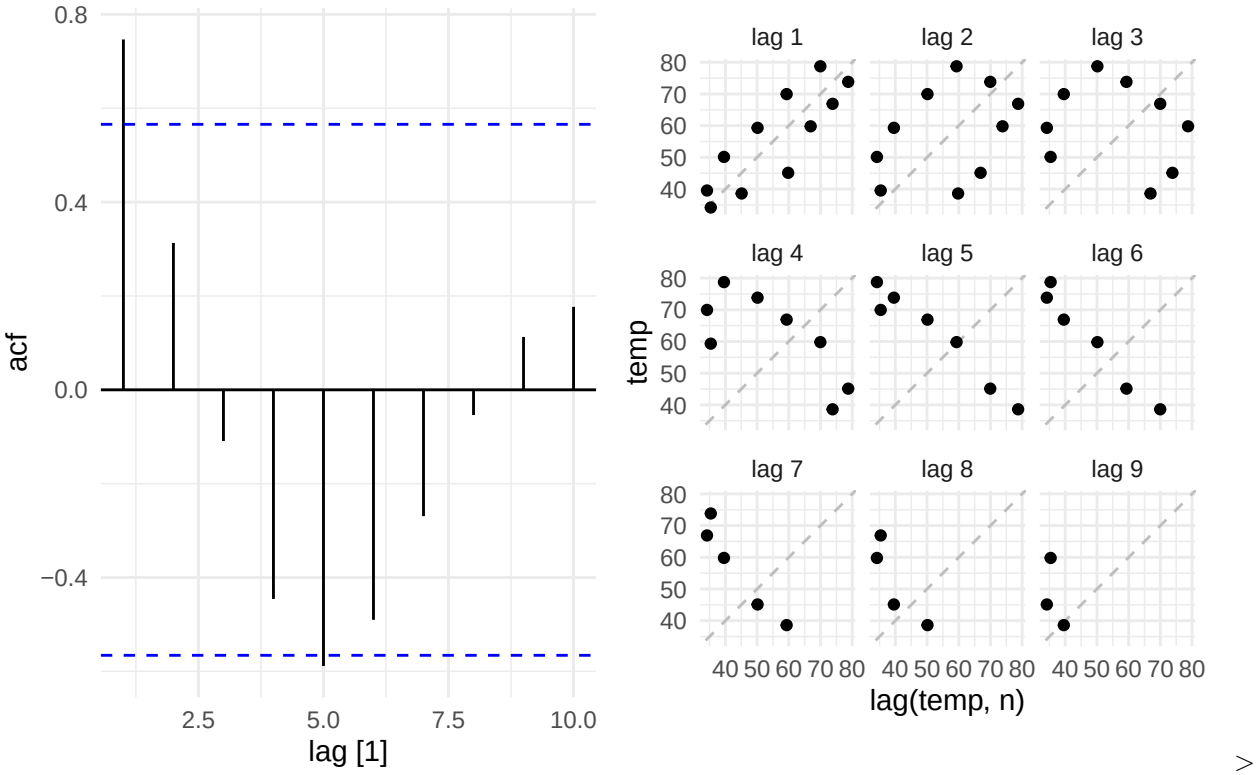
Both ACF and lag plots show strong correlation between consequtive weekly measurements, which is not surprising, given the fact that the weather does not change completely randomly from one day to the next. The correlation is becoming weaker as the lag increases and at some point, when the lag extends to the "oposite" season, the correlation inverts.

(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_ts %>% filter(origin == "JFK") %>%
 index_by(mnth = ~month(.))%>% summarise(temp = mean(temp)) %>% ACF(temp) %>% autoplot()
monthly_lag <-  weather_ts %>% filter(origin == "JFK") %>%
 index_by(mnth = ~month(.)) %>% summarise(temp = mean(temp))%>%
  gg_lag(temp, lag = 1:9, geom = "point", color = 1)

monthly_acf | monthly_lag
```



Both ACF and lag plots show strong correlation between consequtive monthly averages, which is not surprising, given yearly cycles of climate. The correlation is becoming weaker as the lag increases and at some point, when the lag extends into to the "oposite" season, the correlation inverts.

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

```
plot1 <- dataset_1 %>% autoplot()
```

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

```
plot2 <- dataset_2 %>% autoplot()
```

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

```
plot3 <- dataset_3 %>% autoplot()
```

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

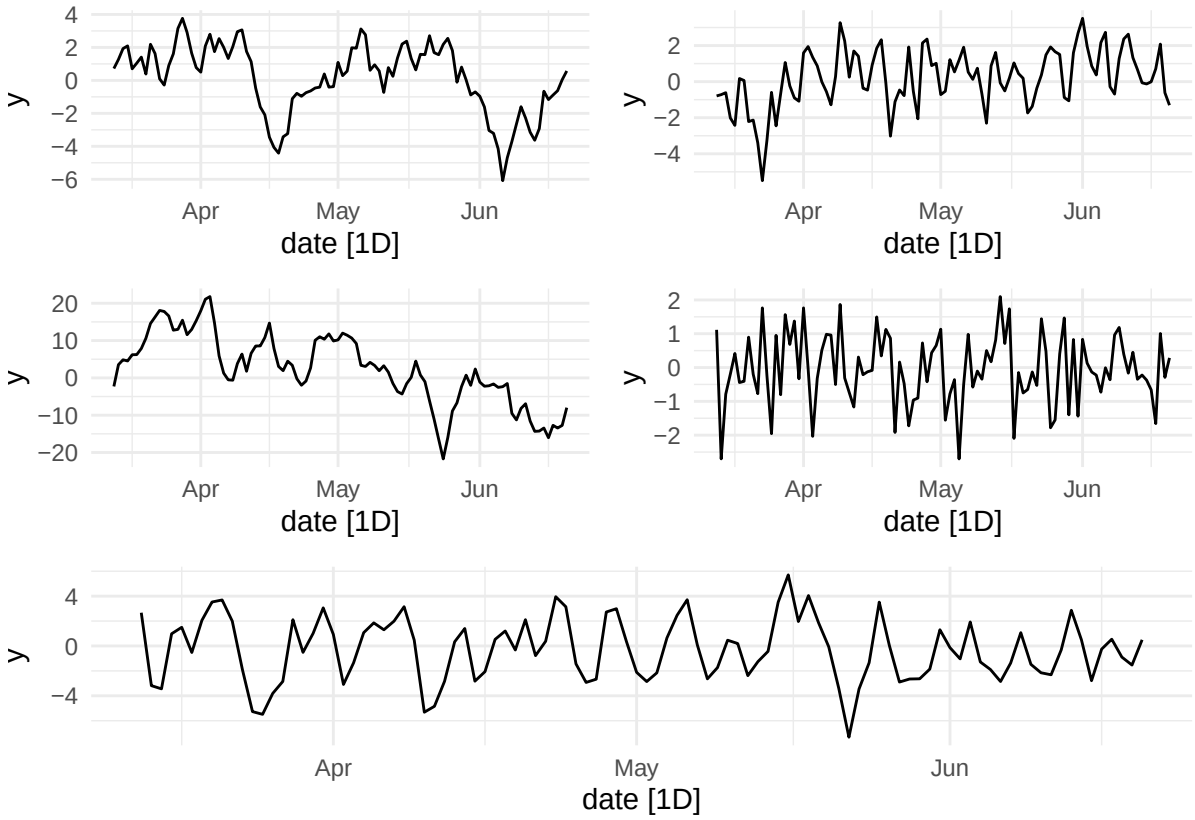
```
plot4 <- dataset_4 %>% autoplot()
```

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

```
plot5 <- dataset_5 %>% autoplot()
```

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

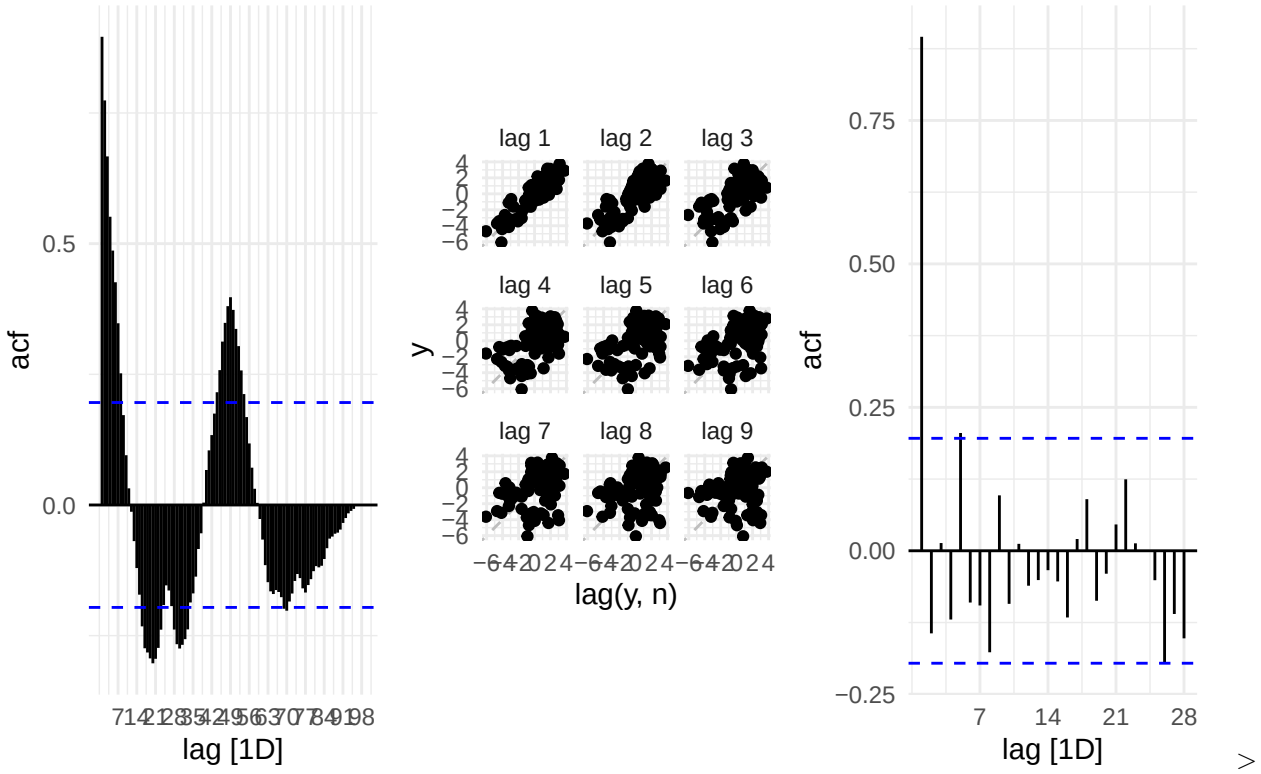
```
   (plot1| plot2) /
   (plot3 | plot4) /
   (plot5)
```



(1 point) Dataset One

```
plot1_acf <- dataset_1 %>% as_tsibble() %>% ACF(y, lag_max = 200) %>% autoplot()
plot1_pcf <- dataset_1 %>% as_tsibble() %>% ACF(y, type = c("partial"), lag_max = 28) %>% autoplot()
plot1_lag <- dataset_1 %>% as_tsibble() %>% gg_lag(y, lag = 1:9, geom = "point", color = 1)

plot1_acf | plot1_lag | plot1_pcf
```

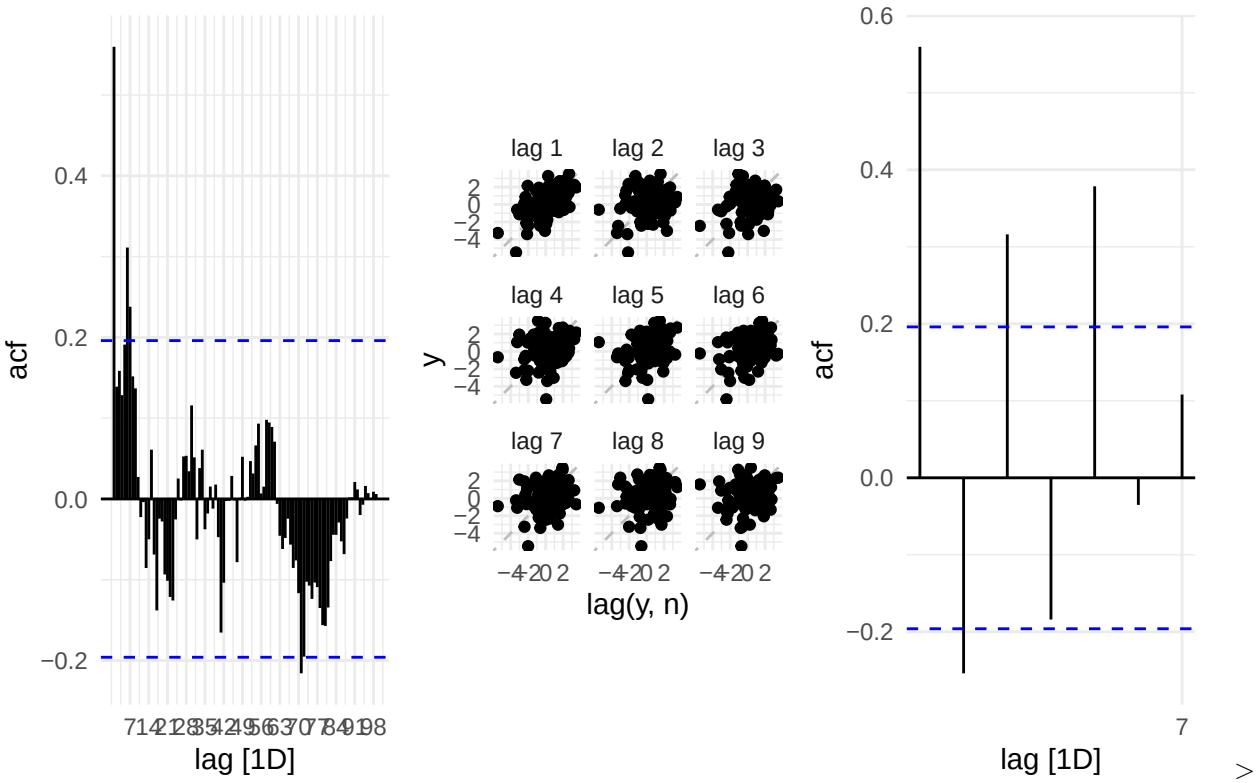


The series is chrachterised by short-lived trends that last for a few weeks. There is a strong autocorrelation with lag 1, that fades away almost completely beyond lag 3.

(1 point) Dataset Two

```
plot2_acf <- dataset_2 %>% as_tsibble() %>% ACF(y, lag_max = 200) %>% autoplot()
plot2_pcf <- dataset_2 %>% as_tsibble() %>% ACF(y, type = c("partial"), lag_max = 7) %>% autoplot()
plot2_lag <- dataset_2 %>% as_tsibble() %>% gg_lag(y, lag = 1:9, geom = "point", color = 1)

plot2_acf | plot2_lag | plot2_pcf
```

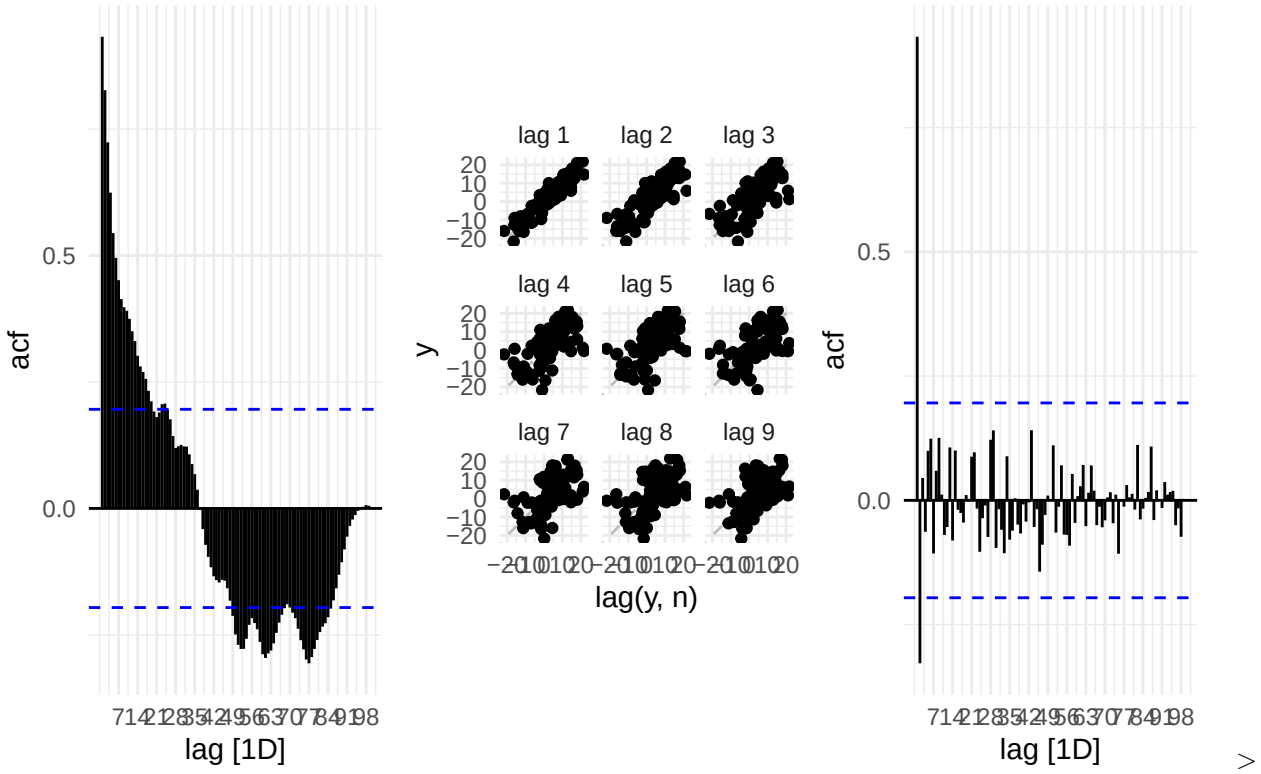


The series lack any long or medium lived trend. There appears to be a noticible autocorrelation with lag 1 and switching sign, i.e. oscislation with 1 day period.

(1 point) Dataset Three

```
plot3_acf <- dataset_3 %>% as_tsibble() %>% ACF(y, lag_max = 200) %>% autoplot()
plot3_pcf <- dataset_3 %>% as_tsibble() %>% ACF(y, type = c("partial"), lag_max = 200) %>% autoplot()
plot3_lag <- dataset_3 %>% as_tsibble() %>% gg_lag(y, lag = 1:9, geom = "point", color = 1)

plot3_acf | plot3_lag | plot3_pcf
```

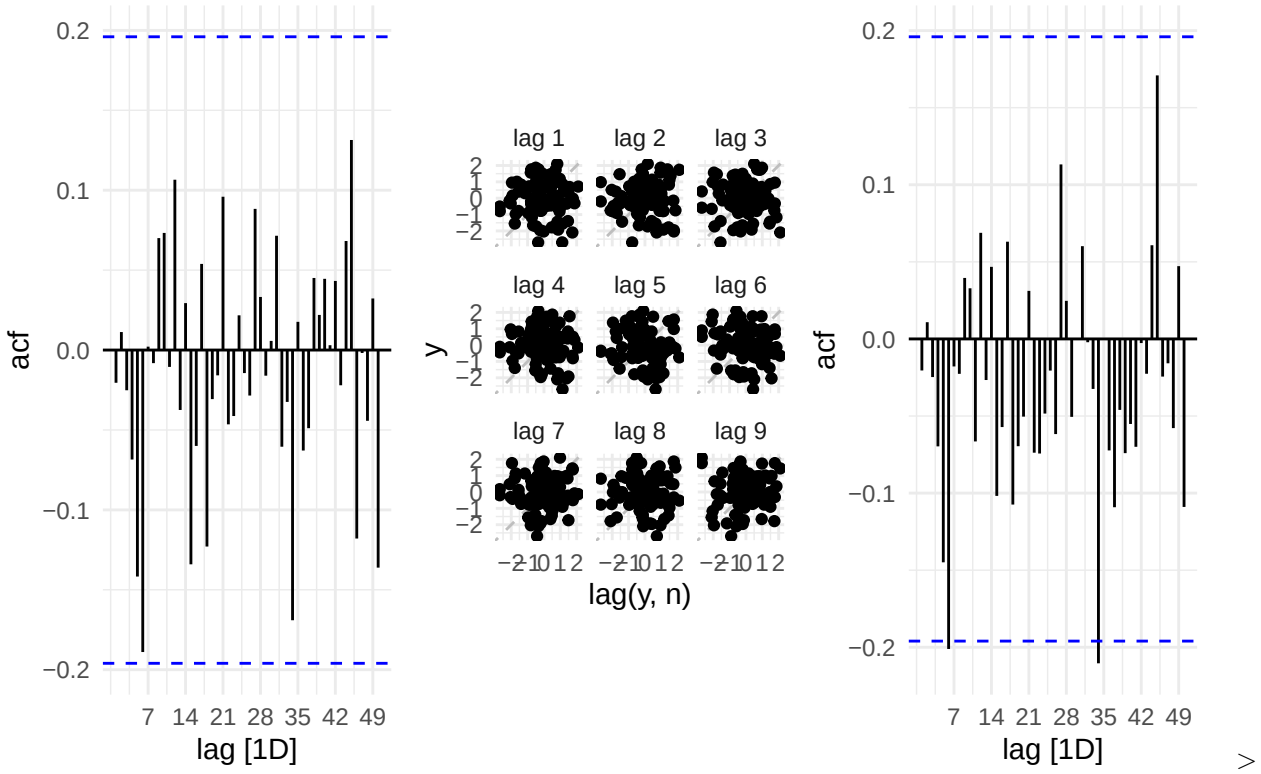


The series has both long-term trend and and medium-term trends that switch directions.

(1 point) Dataset Four

```
plot4_acf <- dataset_4 %>% as_tsibble() %>% ACF(y, lag_max = 50) %>% autoplot()
plot4_pcf <- dataset_4 %>% as_tsibble() %>% ACF(y, type = c("partial"), lag_max = 50) %>% autoplot()
plot4_lag <- dataset_4 %>% as_tsibble() %>% gg_lag(y, lag = 1:9, geom = "point", color = 1)

plot4_acf | plot4_lag | plot4_pcf
```

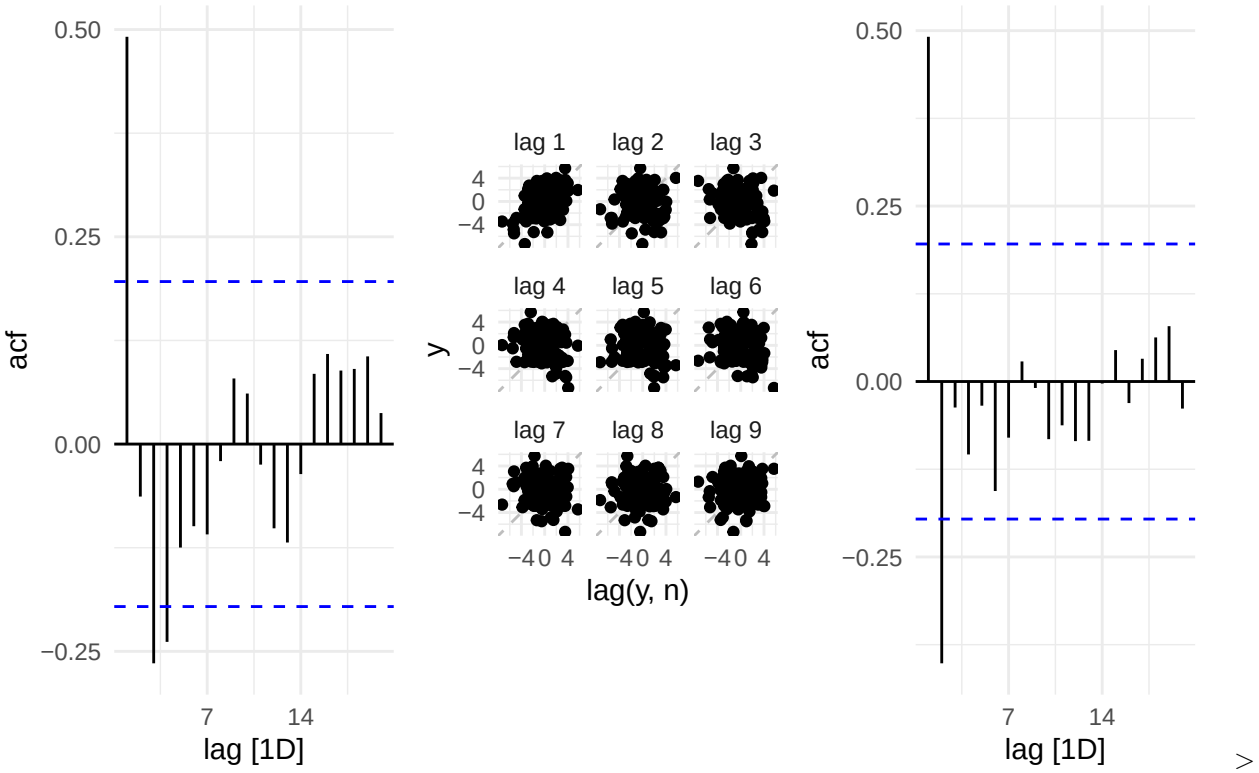


The series is very close to white noise, but it is suspicious how noise tends to form cluster, with 4-5 day periods. There is a chance of weak weekly oscilations masked by strong noise.

(1 point) Dataset Five

```
plot5_acf <- dataset_5 %>% as_tsibble() %>% ACF(y, lag_max = 20) %>% autoplot()
plot5_pcf <- dataset_5 %>% as_tsibble() %>% ACF(y, type = c("partial"), lag_max = 20) %>% autoplot()
plot5_lag <- dataset_5 %>% as_tsibble() %>% gg_lag(y, lag = 1:9, geom = "point", color = 1)

plot5_acf | plot5_lag | plot5_pcf
```



The series similar to the previos one, but oscialtions are even more visible.

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
```

```
series_ids <- list(uer.total ="LNS14000000", uer.men ="LNS14000001", uer.women = 'LNS14000002')
uer_series <- get_n_series_table(series_ids,
                                 '913c5fe448924ea584e043f736c95080',
                                 start_year = 2000,
                                 end_year = 2023)
```

```
## Year 2000 to 2023 is longer than 20 year API limit. Performing 2 requests.
```

```
uer_series$period <- as.integer(str_sub(uer_series$period, 2, 3))
names(uer_series) <- c("year", "month", "overall", "male", "female")
unemployment <- pivot_longer(uer_series, cols = 3:5, names_to = "name", values_to = "value")
unemployment$time_index <- yearmonth(paste(unemployment$year, unemployment$month))
```

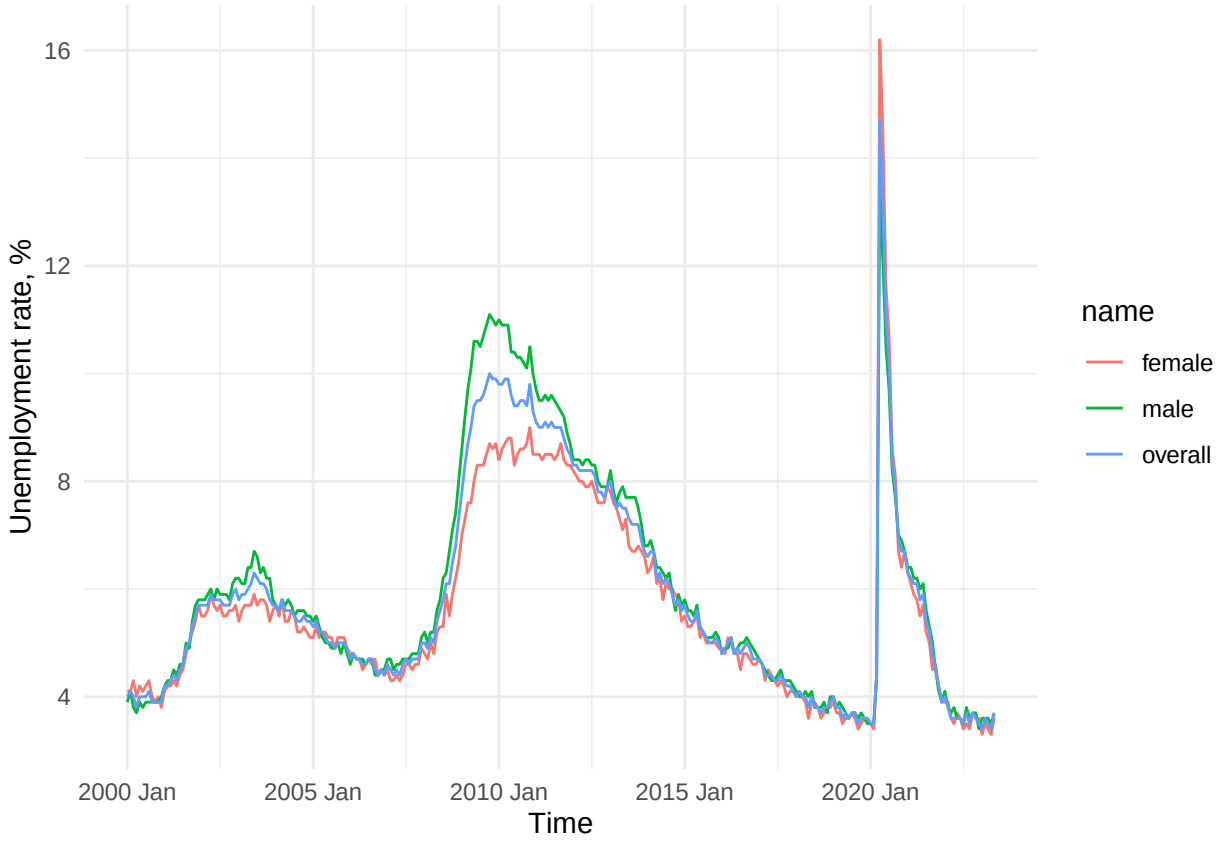
```
head(unemployment)
```

```
## # A tibble: 6 x 5
##    year month name    value time_index
##   <int> <int> <chr>   <dbl>      <mth>
## 1  2000     1 overall   4     2000 Jan
## 2  2000     1 male      3.9   2000 Jan
## 3  2000     1 female    4.1   2000 Jan
## 4  2000     2 overall   4.1   2000 Feb
## 5  2000     2 male      4.1   2000 Feb
## 6  2000     2 female    4.1   2000 Feb
```

(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 = time_index, y = value, color = name) + geom_line() + xlab("Time") +
```

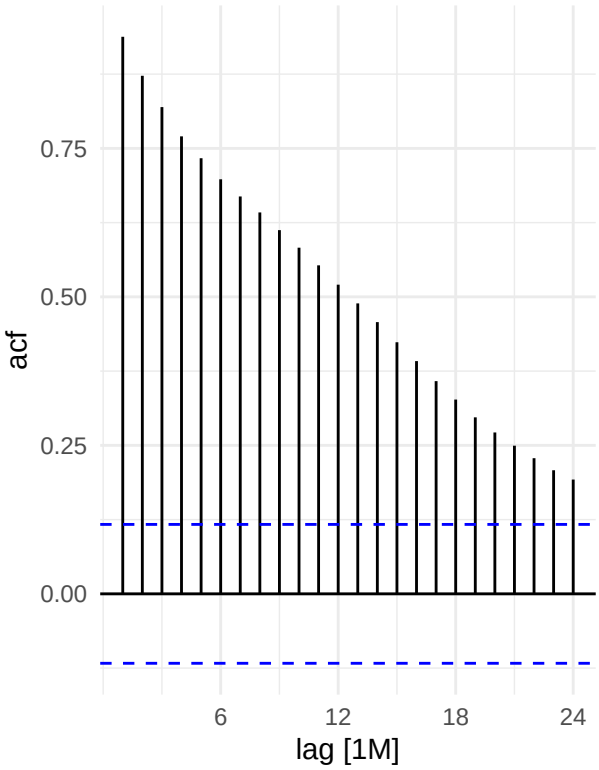


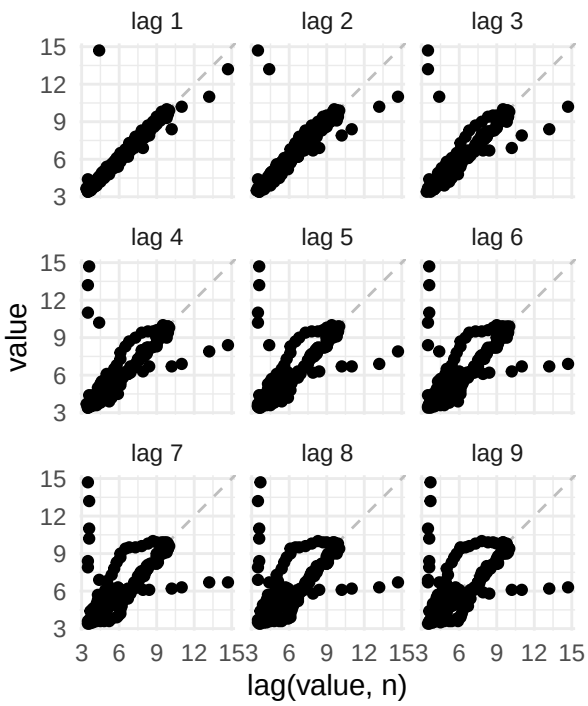
(1 point) Plot the ACF and Lags

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

```
unemp_ts <- unemployment %>% filter(name == "overall") %>% as_tsibble(key = 'name', index = 'time_index
unemployment_acf <- unemp_ts %>% ACF(value) %>% autoplot()
unemployment_lag <- unemp_ts %>% gg_lag(value, lag = 1:9, geom = "point", color = 1)

unemployment_acf | unemployment_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
lag(value, n)