# W271 Assignment 6

```
Sys.setlocale("LC_TIME", "English")
```

```
## [1] "English_United States.1252"
```

```
library(tidyverse)
library(patchwork)

library(lubridate)

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

library(sandwich)
library(lmtest)

library(nycflights13)
library(blsR)
library(reshape2)
```

```
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>
```

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

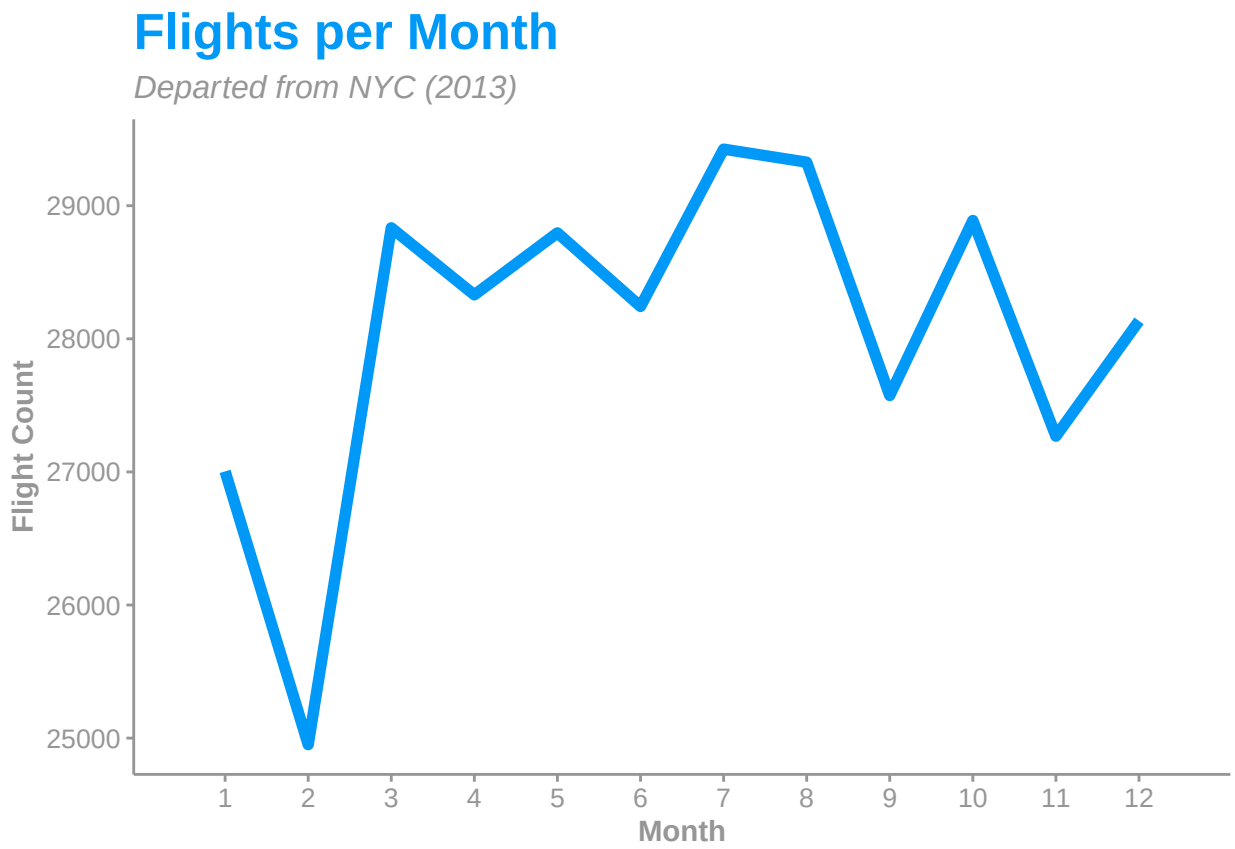
```
## # A tsibble: 336,776 x 20 [1m] <UTC>
## # Key:       carrier, flight [5,725]
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
##  1  2013    11     3     1531           1540        -9     1653           1725
##  2  2013    11     4     1539           1540        -1     1712           1725
##  3  2013    11     5     1548           1540         8     1708           1725
##  4  2013    11     6     1535           1540        -5     1657           1725
##  5  2013    11     7     1549           1540         9     1733           1725
##  6  2013    11     8     1539           1540        -1     1706           1725
##  7  2013    11     9     1535           1540        -5     1714           1723
##  8  2013    11    10     1538           1540        -2     1718           1725
```

```
##  9  2013    11    11     1527           1540       -13     1710           1725
## 10  2013    11    12     1535           1540        -5     1709           1725
## # ... with 336,766 more rows, and 12 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>,
## #   time_index <dttm>
```

## (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.)

```
flight_count <- ggplot(
  flights,
  aes(
    x = month
    )
  ) +
  geom_line(
    aes(fill = ..count..),
    stat = "bin",
    binwidth = 1,
    color = "#0099F8",
    size = 2
    ) +
  labs(
    title = "Flights per Month",
    subtitle = "Departed from NYC (2013)",
    x = "Month",
    y = "Flight Count"
    ) +
  theme_classic() +
  theme(
    plot.title = element_text(color = "#0099F8",
                              size = 19,
                              face = "bold"),
    plot.subtitle = element_text(color="#969696",
                                 size = 12,
                                 face = "italic"),
    axis.title = element_text(color = "#969696",
                              size = 11,
                              face = "bold"),
    axis.text = element_text(color = "#969696", size = 10),
    axis.line = element_line(color = "#969696"),
    axis.ticks = element_line(color = "#969696")
    ) +
  scale_x_continuous(breaks= 1:12)
flight_count
```




We can observe that we have the lowest values in the first two months and the highest count during July and August (perhaps due to the summer holidays). Besides that, there's not any special pattern we can notice from just this plot.

## (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 <- airports %>%
  mutate(is_tropical = ifelse(lat < 23.4394, "Yes", "No"))

# We use Inner Join because there are 7602 missing values
flights <- inner_join(
  flights,
  airports,
  by = c("dest" = "faa")
  )
```

When using group by month while we have a time index, the grouping is made by this whole index instead of the specific month contained in it as we can see in the following example:

```
flights  %>%
  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> <chr>       <dttm>                      <int>
##  1     1 No          2013-01-01 05:15:00             1
##  2     1 No          2013-01-01 05:29:00             1
##  3     1 No          2013-01-01 05:40:00             1
##  4     1 No          2013-01-01 05:58:00             1
##  5     1 No          2013-01-01 05:59:00             1
##  6     1 No          2013-01-01 06:00:00            17
##  7     1 No          2013-01-01 06:05:00             1
##  8     1 No          2013-01-01 06:07:00             1
##  9     1 No          2013-01-01 06:08:00             1
## 10     1 No          2013-01-01 06:10:00             5
## # ... with 125,709 more rows
```

Instead, we need to extract the month from the index as follows:

```
flight_trop <- flights  %>%
  index_by(Month = ~ month(.)) %>%
  group_by(is_tropical)  %>%
  summarise(total_flights = n())
flight_trop
```

```
## # A tsibble: 24 x 3 [1]
## # Key:       is_tropical [2]
##    is_tropical Month total_flights
##    <chr>       <dbl>         <int>
##  1 No              1         26262
##  2 No              2         24287
##  3 No              3         28032
##  4 No              4         27616
```

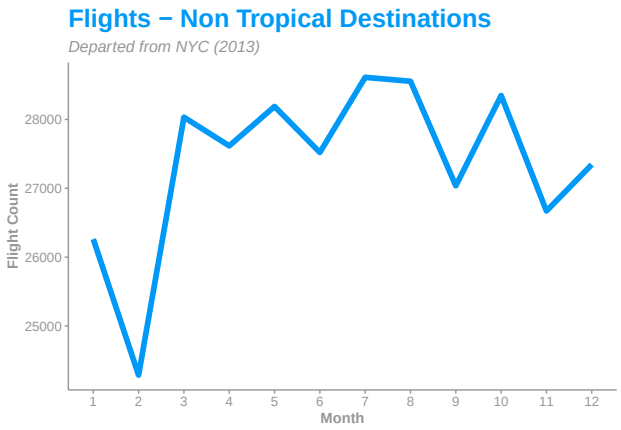
```
##  5 No              5         28188
##  6 No              6         27520
##  7 No              7         28611
##  8 No              8         28554
##  9 No              9         27036
## 10 No             10         28347
## # ... with 14 more rows
```

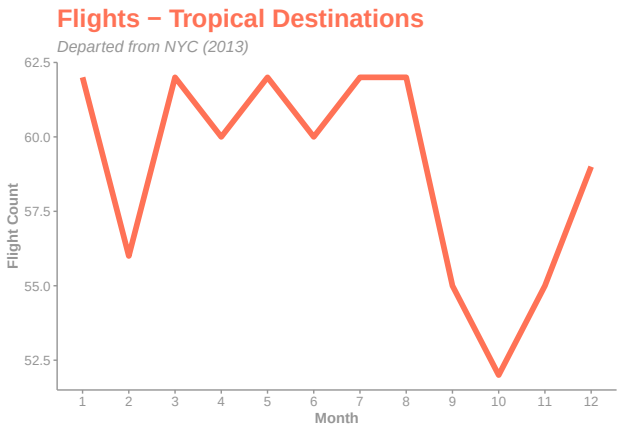
```
no_trop_graph <- ggplot(
  flight_trop %>% filter(is_tropical == "No"),
  aes(
    x = Month,
    y = total_flights
    )
  ) +
  geom_line(
    aes(color = is_tropical),
    color = "#0099F8",
    size = 2
    ) +
  labs(
    title = "Flights - Non Tropical Destinations",
    subtitle = "Departed from NYC (2013)",
    x = "Month",
    y = "Flight Count"
    ) +
  theme_classic() +
  theme(
    plot.title = element_text(color = "#0099F8",
                              size = 19,
                              face = "bold"),
    plot.subtitle = element_text(color="#969696",
                                 size = 12,
                                 face = "italic"),
    axis.title = element_text(color = "#969696",
                              size = 11,
                              face = "bold"),
    axis.text = element_text(color = "#969696", size = 10),
    axis.line = element_line(color = "#969696"),
    axis.ticks = element_line(color = "#969696")
    ) +
  scale_x_continuous(
    breaks= 1:12
    )

trop_graph <- ggplot(
  flight_trop %>% filter(is_tropical == "Yes"),
  aes(
    x = Month,
    y = total_flights
    )
  ) +
  geom_line(
    aes(color = is_tropical),
```

```
    color = "coral1",
    size = 2
    ) +
  labs(
    title = "Flights - Tropical Destinations",
    subtitle = "Departed from NYC (2013)",
    x = "Month",
    y = "Flight Count"
    ) +
  theme_classic() +
  theme(
    plot.title = element_text(color = "coral1",
                              size = 19,
                              face = "bold"),
    plot.subtitle = element_text(color="#969696",
                                 size = 12,
                                 face = "italic"),
    axis.title = element_text(color = "#969696",
                              size = 11,
                              face = "bold"),
    axis.text = element_text(color = "#969696", size = 10),
    axis.line = element_line(color = "#969696"),
    axis.ticks = element_line(color = "#969696")
    ) +
  scale_x_continuous(
    breaks= 1:12
    )

no_trop_graph
trop_graph
```




By looking at the scale on both graphs we can notice that there are not many Flights to tropical destinations. Nevertheless, we can also observe that these few flights don't follow the same pattern as the rest, because they have their lowest point in the last few months (especially Sep, Oct & Nov), instead of the first months of the year.

# 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  <dttm> 2013-11-03 01:00:00, 2013-11-03 01:00:00, 2013-11-03 01:00:~
## $ time_index <dttm> 2013-11-03 01:00:00, 2013-11-03 01:00:00, 2013-11-03 01:00:~
```

We have 3 duplicated entries (same time_index and origin), and this can cause us trouble to create a tsibble object from this weather data. So we'll first remove one of each duplicate to keep just a single value, since there are only three of them and the measurements don't differ too much between duplicates.

```
weather <- distinct(weather, origin, year, month, day, hour, .keep_all= TRUE)
```

By doing this we'll be able to create our tsibble object from this data, with 3 entries less than the original tibble:

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

```
## # A tsibble: 26,112 x 16 [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 26,102 more rows, and 6 more variables: wind_gust <dbl>,
## #   precip <dbl>, pressure <dbl>, visib <dbl>, time_hour <dttm>,
## #   time_index <dttm>
```

## (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(
    x = time_index,
    y = temp
    )
  ) +
  geom_line(
```

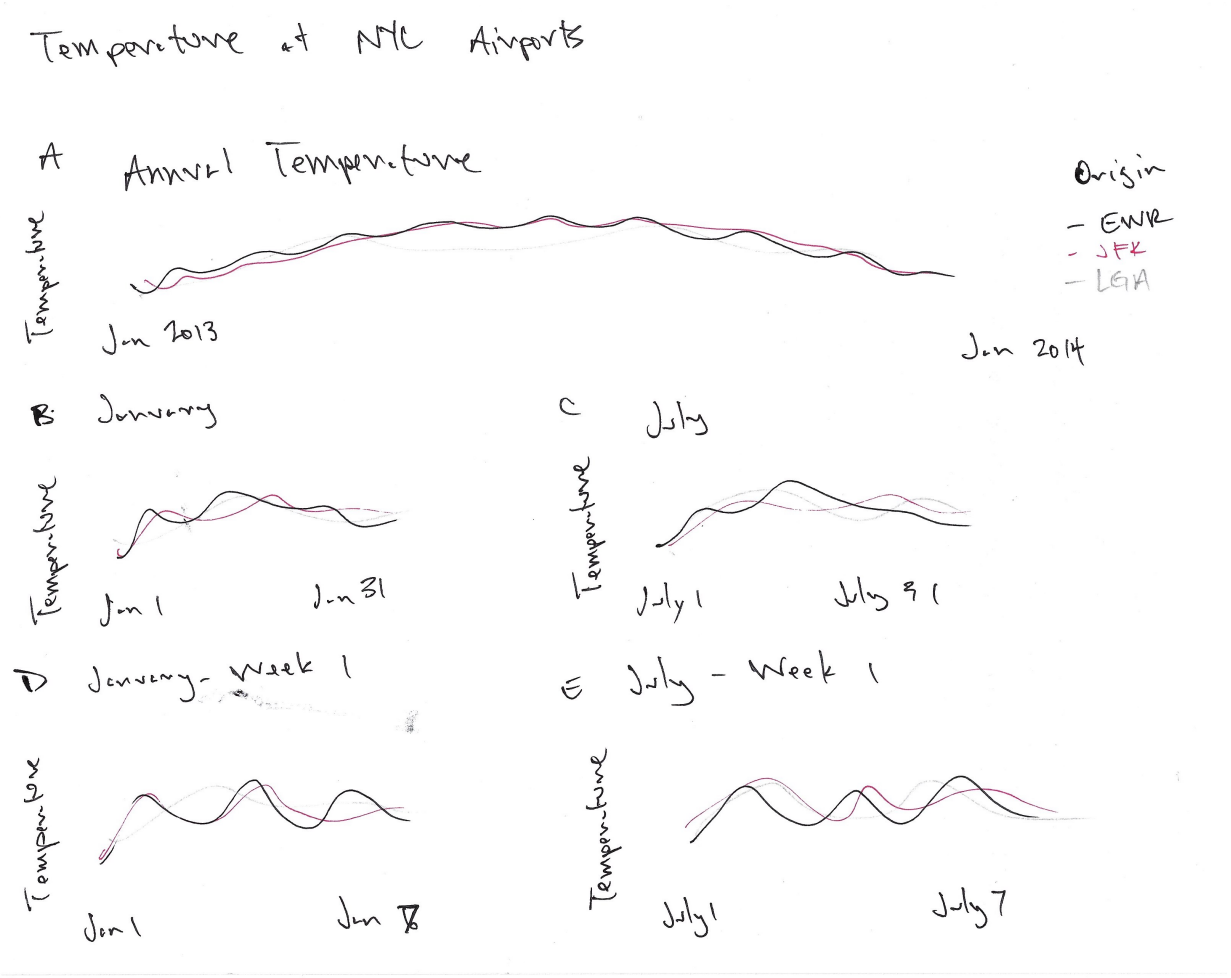



Figure 1: sample plot

```
    aes(color = origin),
    size = 0.5,
    show.legend = FALSE
    ) +
  labs(
    title = "Annual Temperature",
    x = "Month",
    y = "Temperature"
    ) +
  theme_classic() +
  theme(
    plot.title = element_text(color = "#0099F8",
                              size = 10,
                              face = "bold"),
    axis.title = element_text(color = "#969696",
                              size = 6),
    axis.text = element_text(color = "#969696", size = 6),
    axis.line = element_line(color = "#969696"),
    axis.ticks = element_line(color = "#969696")
    )
```

```
january_plot <- ggplot(
  weather %>% filter(month == 1),
  aes(
    x = time_index,
    y = temp
    )
  ) +
  geom_line(
    aes(color = origin),
    size = 0.5,
    show.legend = FALSE
    ) +
  labs(
    title = "January",
    x = "Month",
    y = "Temperature"
    ) +
  theme_classic() +
  theme(
    plot.title = element_text(color = "#0099F8",
                              size = 10,
                              face = "bold"),
    axis.title = element_text(color = "#969696",
                              size = 6),
    axis.text = element_text(color = "#969696", size = 6),
    axis.line = element_line(color = "#969696"),
    axis.ticks = element_line(color = "#969696"),
    ) +
  scale_y_continuous(
    expand = c(0, 0),
    limits = c(0, 100)
    )
```

```
july_plot <- ggplot(
  weather %>% filter(month == 7),
  aes(
    x = time_index,
    y = temp
    )
  ) +
  geom_line(
    aes(color = origin),
    size = 0.5,
    show.legend = FALSE
    ) +
  labs(
    title = "July",
    x = "Month",
    y = "Temperature"
    ) +
  theme_classic() +
  theme(
    plot.title = element_text(color = "#0099F8",
                              size = 10,
                              face = "bold"),
    axis.title = element_text(color = "#969696",
                              size = 6),
    axis.text = element_text(color = "#969696", size = 6),
    axis.line = element_line(color = "#969696"),
    axis.ticks = element_line(color = "#969696"),
    ) +
  scale_y_continuous(
    expand = c(0, 0),
    limits = c(0, 100)
    )
```

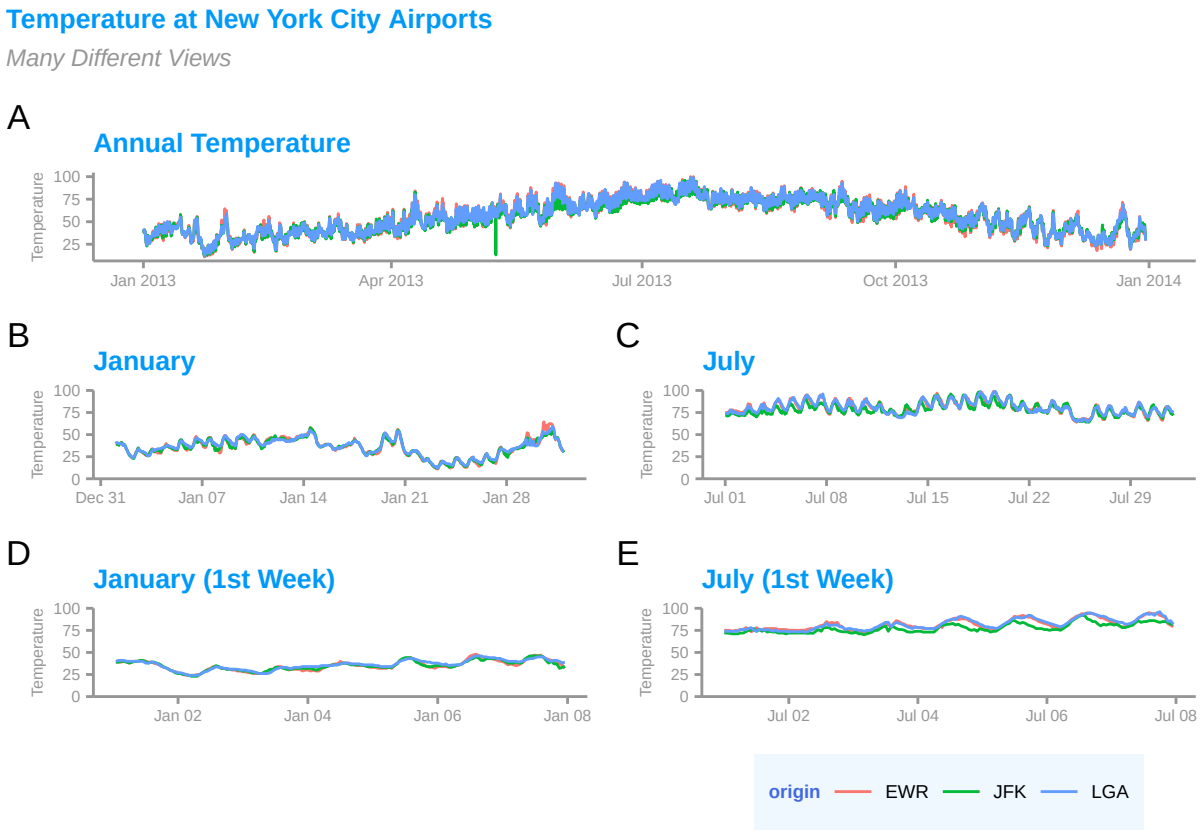
```
january_first_week <- ggplot(
  weather %>% filter(month == 1 & day < 8),
  aes(
    x = time_index,
    y = temp
    )
  ) +
  geom_line(
    aes(color = origin),
    size = 0.5,
    show.legend = FALSE
    ) +
  labs(
    title = "January (1st Week)",
    x = "Month",
    y = "Temperature"
    ) +
  theme_classic() +
  theme(
    plot.title = element_text(color = "#0099F8",
```

```
                              size = 10,
                              face = "bold"),
    axis.title = element_text(color = "#969696",
                              size = 6),
    axis.text = element_text(color = "#969696", size = 6),
    axis.line = element_line(color = "#969696"),
    axis.ticks = element_line(color = "#969696"),
    ) +
  scale_y_continuous(
    expand = c(0, 0),
    limits = c(0, 100)
    )

july_first_week <- ggplot(
  weather %>% filter(month == 7 & day < 8),
  aes(
    x = time_index,
    y = temp
    )
  ) +
  geom_line(
    aes(color = origin),
    size = 0.5,
    ) +
  labs(
    title = "July (1st Week)",
    x = "Month",
    y = "Temperature"
    ) +
  theme_classic() +
  theme(
    plot.title = element_text(color = "#0099F8",
                              size = 10,
                              face = "bold"),
    axis.title = element_text(color = "#969696",
                              size = 6),
    axis.text = element_text(color = "#969696", size = 6),
    axis.line = element_line(color = "#969696"),
    axis.ticks = element_line(color = "#969696"),
    legend.title = element_text(color = "royalblue",
                                size = 7,
                                face = "bold"),
    legend.background = element_rect(fill ="aliceblue"),
    legend.text=element_text(size=7),
    legend.position="bottom"
    ) +
  scale_y_continuous(
    expand = c(0, 0),
    limits = c(0, 100)
    )
```

```
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') &
theme(
  plot.title = element_text(color = "#0099F8",
                            size = 10,
                            face = "bold"),
  plot.subtitle = element_text(color="#969696",
                               size = 9,
                               face = "italic"),
)
```




We can immediately notice that there's certain trend and seasonality in the annual temperature. Moreover, the temperatures are very similar for all 3 airports, which is consistent since all 3 airports are nearby. For that very reason we an also notice there's an outlier in JFK's data that makes no sense. The scales quickly indicate lower temperatures in january than in july, but we can also observe that the trend and seasonality are less obvious as we zoom in.

## (1 point) Hourly ACF

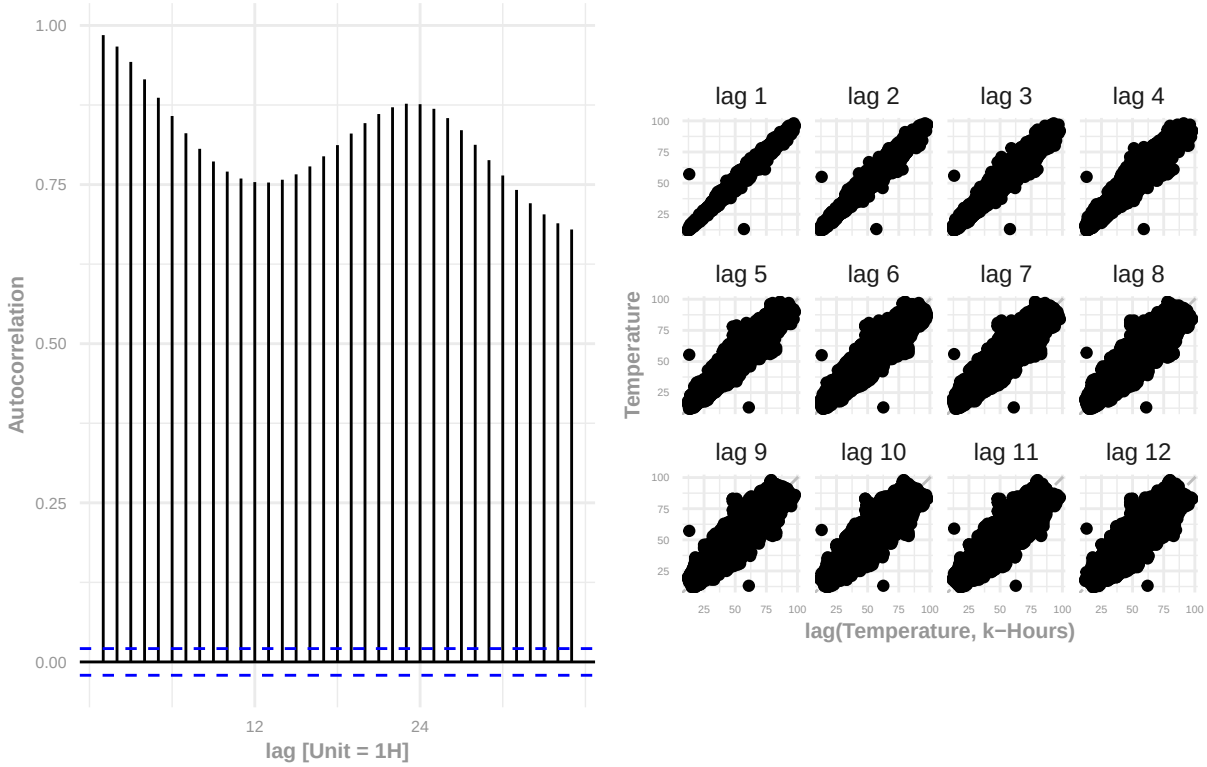
At the hourly level, produce an ACF and a lag plot at JFK. What do you learn from these plots? (Note that you can suppress all the coloring in the `gg_lag` call if you pass an additional argument, `color = 1`.)

```r
# Filtering just JFK data
JFK <- weather %>% filter(origin =="JFK")

hourly_acf <- ACF(
  fill_gaps(JFK, .full = TRUE),
  temp
) %>% autoplot() + labs(
  x = "lag [Unit = 1H]",
  y = "Autocorrelation"
) + theme(
   axis.title = element_text(color = "#969696",
                             size = 8,
                             face = "bold"),
   axis.text = element_text(color = "#969696", size = 6),
)

hourly_lag <- gg_lag(
  JFK,
  temp,
  color = 1,
  lags = 1:12,
  geom = "point"
) + labs(
  x = "lag(Temperature, k-Hours)",
  y = "Temperature"
) + theme(
   axis.title = element_text(color = "#969696",
                             size = 8,
                             face = "bold"),
   axis.text = element_text(color = "#969696", size = 4),
)

hourly_acf | hourly_lag
```




From these graphs we can notice a very high autocorrelation even after many lags. Since the values fall outside of the blue dotted lines (CI at 95%) in the ACF graph we can reject the null hypothesis that $\rho_k = 0$ for many values of $k$, as expected there seems to be a lot of dependency to the previous temperature values when the unit of lag is 1 Hour. The peaks in the ACF graph seem to indicate seasonality every 24 hours and certain trend, perhaps indicating an autoregressive model.

## (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?

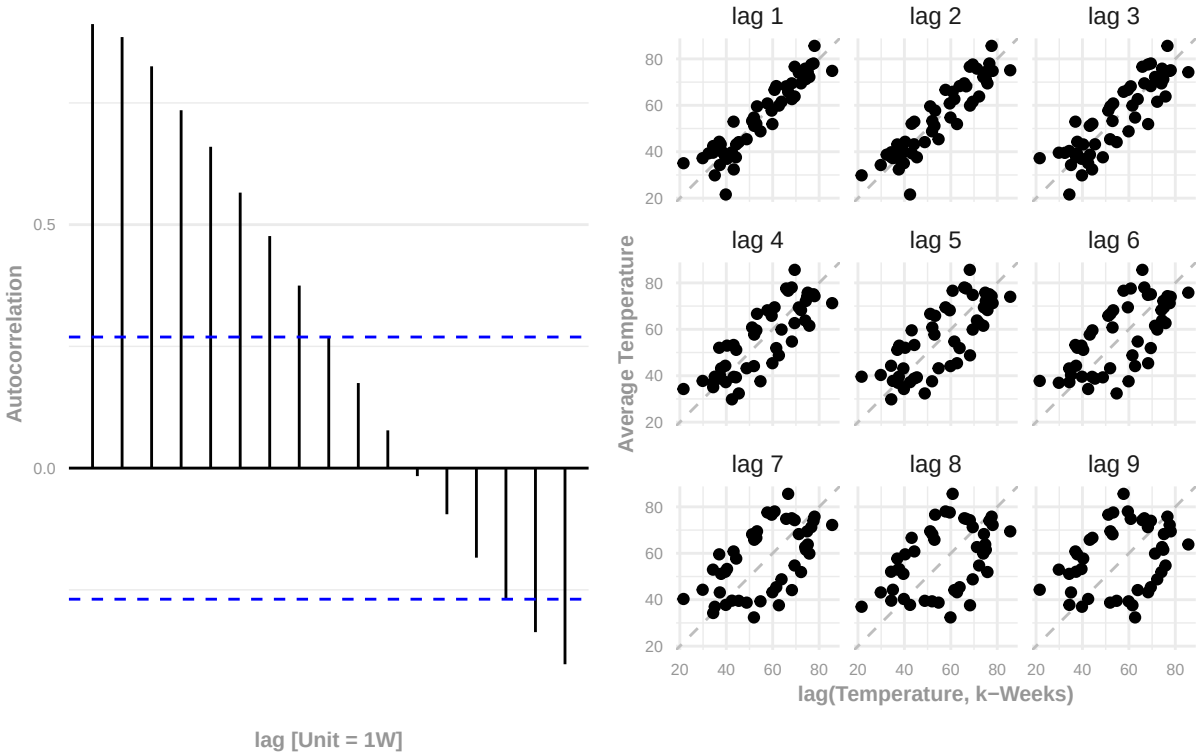
```
JFK_Wk <- JFK %>%  group_by_key() %>%
  index_by(Week = ~ yearweek(.)) %>%
  summarise(
    Avg_Temp = mean(temp)
  )

weekly_acf <- ACF(
  fill_gaps(JFK_Wk, .full = TRUE),
  Avg_Temp
) %>% autoplot() + labs(
  x = "lag [Unit = 1W]",
  y = "Autocorrelation"
) + theme(
   axis.title = element_text(color = "#969696",
                             size = 8,
```

```
                                face = "bold"),
    axis.text = element_text(color = "#969696", size = 6),
)

weekly_lag <- gg_lag(
  JFK_Wk,
  Avg_Temp,
  color = 1,
  geom = "point"
) + labs(
  x = "lag(Temperature, k-Weeks)",
  y = "Average Temperature"
) + theme(
    axis.title = element_text(color = "#969696",
                              size = 8,
                              face = "bold"),
    axis.text = element_text(color = "#969696", size = 6),
)

weekly_acf | weekly_lag
```




When looking at the weekly level we can observe that there's still high autocorrelation for some lag values, but it starts to fade quickly, and it would seem like eventually we will fail to reject the null hypothesis ($H_0 : \rho_k = 0$), meaning that temperature seems to have some dependency on the previous weeks, but that effect decreases as $k$ gets bigger. This might also be an autoregressive model (low order). Understanding the context, perhaps we should not discard seasonality from these results, but analyze bigger lags instead.

## (1 point) Monthly ACF

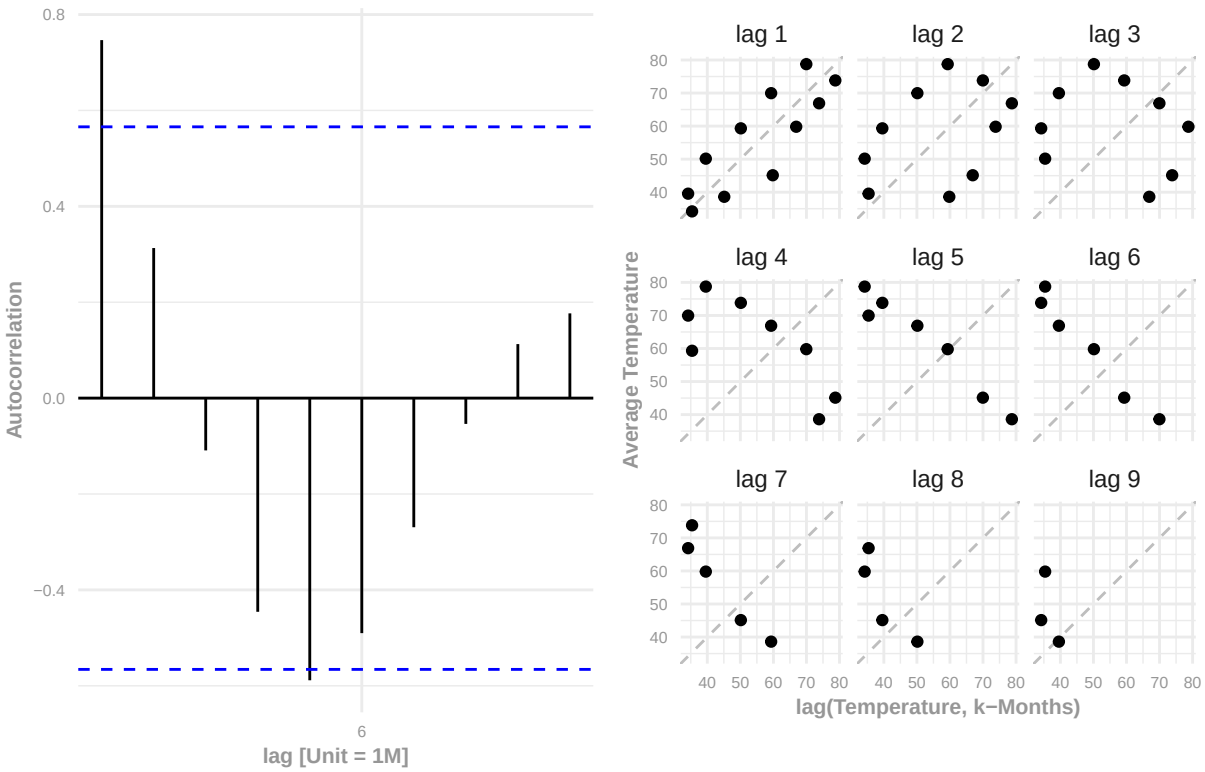
At the monthly level, produce an ACF plot of the monthly average temperature at JFK. What do you learn from these plots?

```
JFK_Mt <- JFK %>%  group_by_key() %>%
  index_by(Month = ~ yearmonth(.)) %>%
  summarise(
    Avg_Temp = mean(temp)
  )

monthly_acf <- ACF(
  fill_gaps(JFK_Mt, .full = TRUE),
  Avg_Temp
) %>% autoplot() + labs(
  x = "lag [Unit = 1M]",
  y = "Autocorrelation"
) + theme(
   axis.title = element_text(color = "#969696",
                             size = 8,
                             face = "bold"),
   axis.text = element_text(color = "#969696", size = 6),
)

monthly_lag <- gg_lag(
  JFK_Mt,
  Avg_Temp,
  color = 1,
  geom = "point"
) + labs(
  x = "lag(Temperature, k-Months)",
  y = "Average Temperature"
) + theme(
   axis.title = element_text(color = "#969696",
                             size = 8,
                             face = "bold"),
   axis.text = element_text(color = "#969696", size = 6),
)

monthly_acf | monthly_lag
```




Finally, for the monthly level we can notice autocorrelation decreases very fast to a point where we will fail to reject the null hypothesis ($H_0 : \rho_k = 0$), meaning that temperature seems to have some dependency perhaps on the immediate previous month but that effect disappears when looking further back, it might even look like a Moving Average Process of low order, like MA(1). The peaks might indicate certain seasonal effect, which would be understandable.

## 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.
```

```
# Create a graphing function to use with every dataset we have

dash_graph <- function(
    data = NULL     # Time Series to be used
    ) {

  line_plot <- ggplot(
```

```
data,
aes(
  x = date,
  y = y
  )
) +
geom_line(
  size = 0.5,
  color = "#0099F8",
  show.legend = FALSE
  ) +
labs(
  title = "Daily Series",
  x = "Date [1D]",
  y = "y"
  ) +
theme_classic() +
theme(
  plot.title = element_text(color = "#0099F8",
                            size = 12),
  axis.title = element_text(color = "#969696",
                            size = 8,
                            face = "bold"),
  axis.text = element_text(color = "#969696", size = 6),
  axis.line = element_line(color = "#969696"),
  axis.ticks = element_line(color = "#969696")
  )

acf_plot <- ACF(
  fill_gaps(data, .full = TRUE),
  y
) %>% autoplot() + labs(
  x = "lag [Unit = 1D]",
  y = "Autocorrelation",
  title = "Autocorrelation Function Plot",
) +
theme_classic() +
theme(
  plot.title = element_text(color = "#0099F8",
                            size = 12),
  axis.title = element_text(color = "#969696",
                            size = 8,
                            face = "bold"),
  axis.text = element_text(color = "#969696", size = 6),
  axis.line = element_line(color = "#969696"),
  axis.ticks = element_line(color = "#969696")
  )

lag_plot <- gg_lag(
  data,
  y,
  color = "#0099F8",
  lags = 1:16,
```

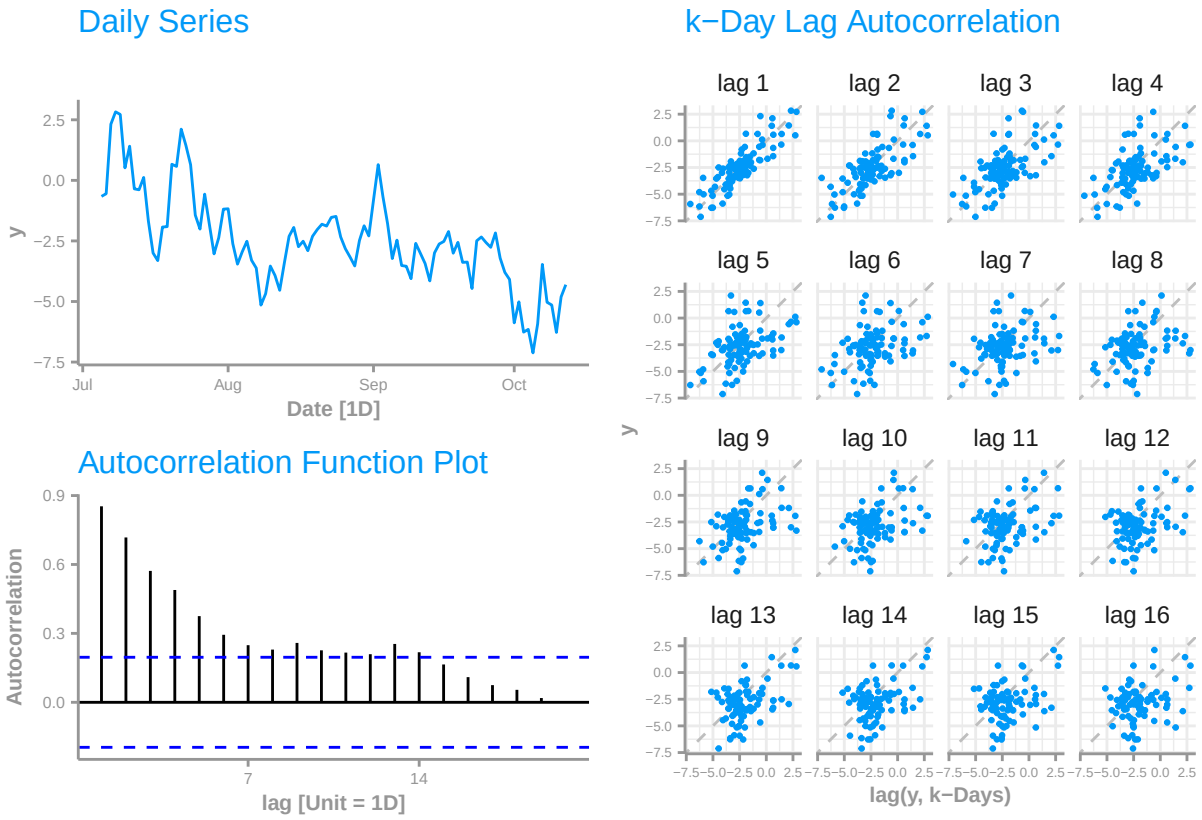
```
    geom = "point",
    size = 0.5
  ) + labs(
    x = "lag(y, k-Days)",
    y = "y",
    title = "k-Day Lag Autocorrelation"
  ) +
  theme(
    plot.title = element_text(color = "#0099F8",
                              size = 12),
    axis.title = element_text(color = "#969696",
                              size = 8,
                              face = "bold"),
    axis.text = element_text(color = "#969696", size = 5),
    axis.line = element_line(color = "#969696"),
    axis.ticks = element_line(color = "#969696")
  )

  graph <- (line_plot / acf_plot) | lag_plot

  return(graph)
}
```

### (1 point) Dataset One

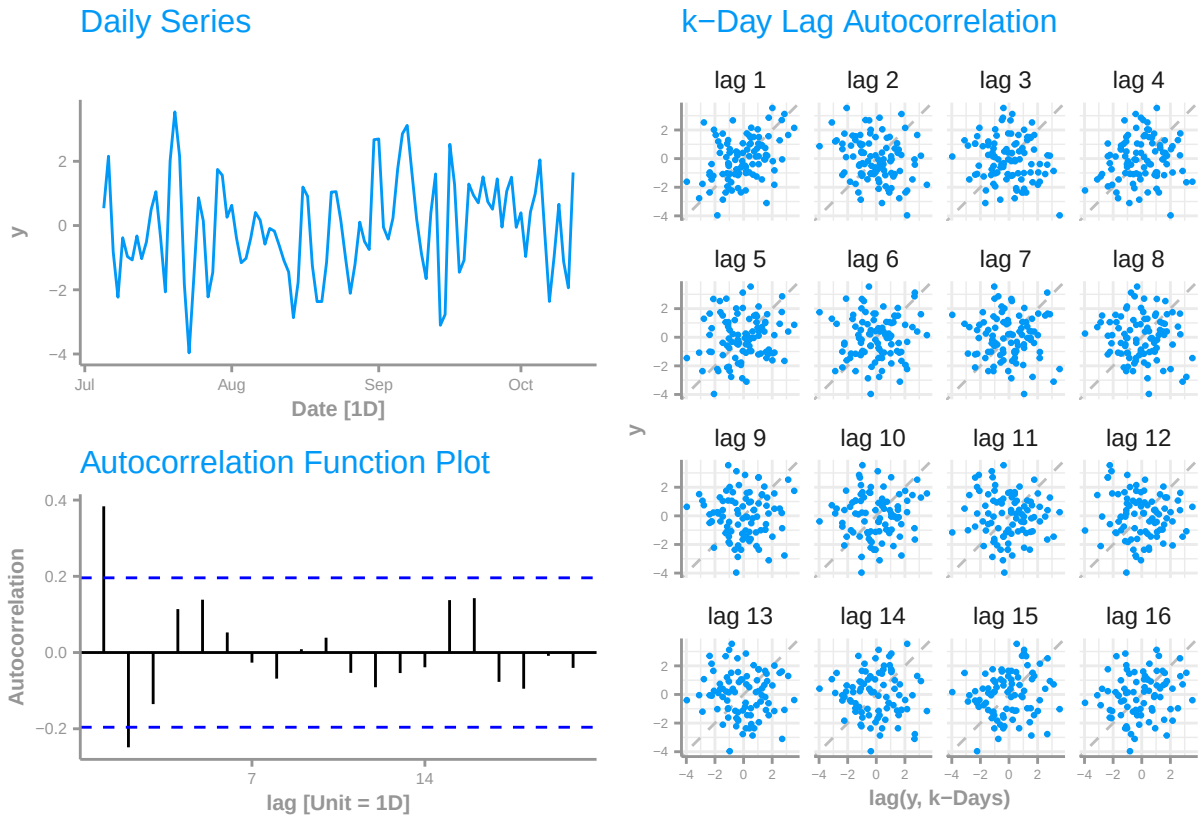
```
dash_graph(
  data = dataset_1
  )
```




Starting with dataset #1 we have a daily plot that seems centered around 0, and the variance doesn't seem to increase or decrease noticeably as time passes. But looking at the autocorrelation we see that it is high for the first lags and starts to decrease as time passes, this information can be confirmed in the ACF as well and we also might have some seasonal effect every 14 days. This could be an example of an autoregressive process (low order) with some seasonal effect.

### (1 point) Dataset Two

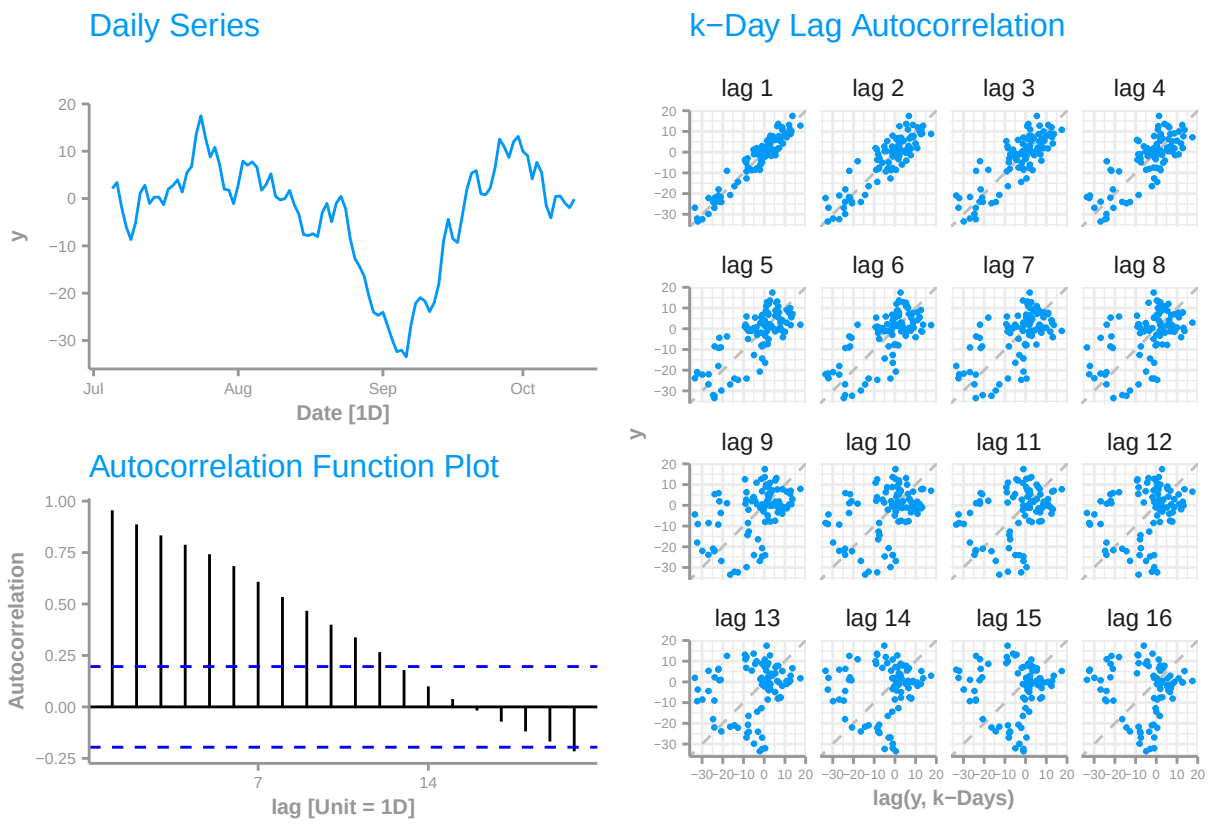
```
dash_graph(
  data = dataset_2
  )
```




Looking at this case we have a daily plot that doesn't seem exacly centered around 0 (although it might still be constant), and the variance might be changing as time passes. There doesn't seem to be significant autocorrelation after the first lag, which is confirmed by looking at the ACF graph, but the spikes could mean a seasonal effect. So we could be talking at a Moving Average Process of Order 1 with a seasonal component.

## (1 point) Dataset Three

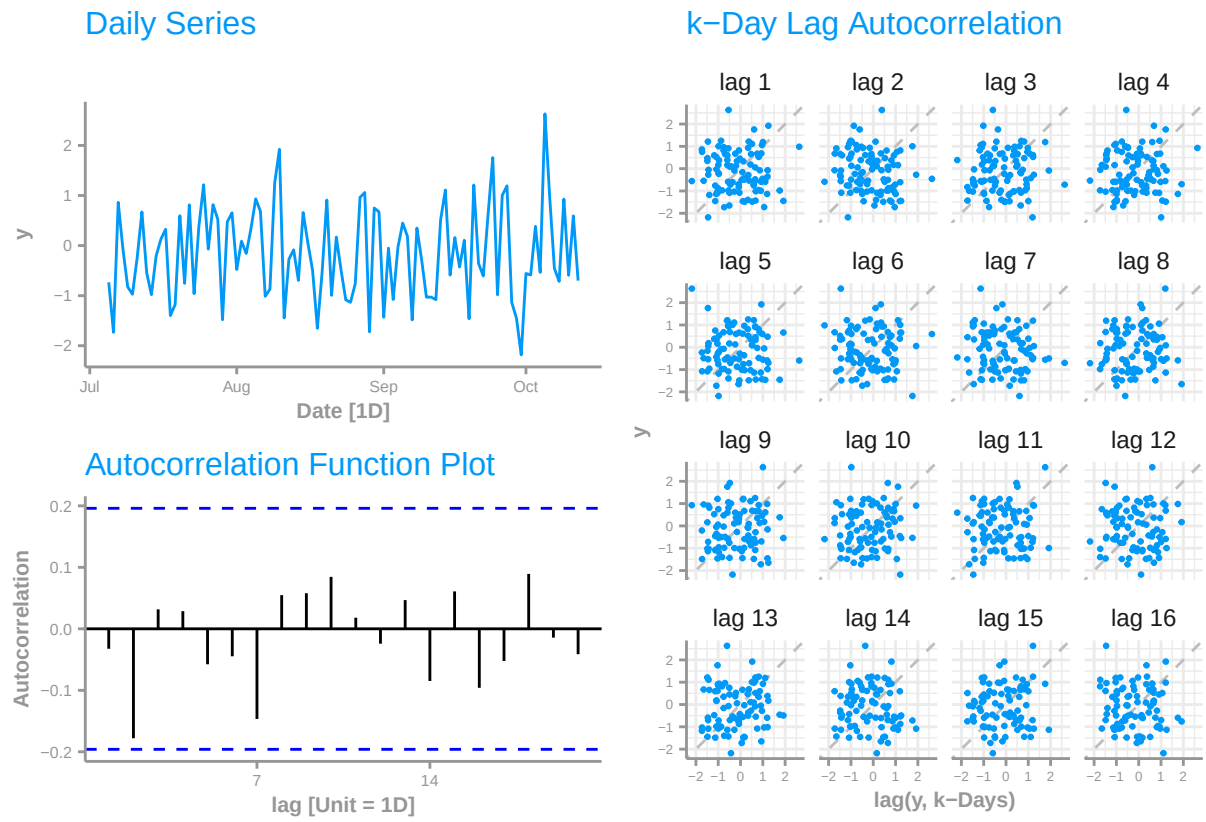
```
dash_graph(
  data = dataset_3
  )
```




We have a daily series that seems to follow certain trend with seasonal behaviors. By looking at the autocorrelations we notice that this graph is heavily correlated to the first lags. The autocorrelation function also confirms this, so we could be talking of an autoregressive process again with seasonal effects.

## (1 point) Dataset Four

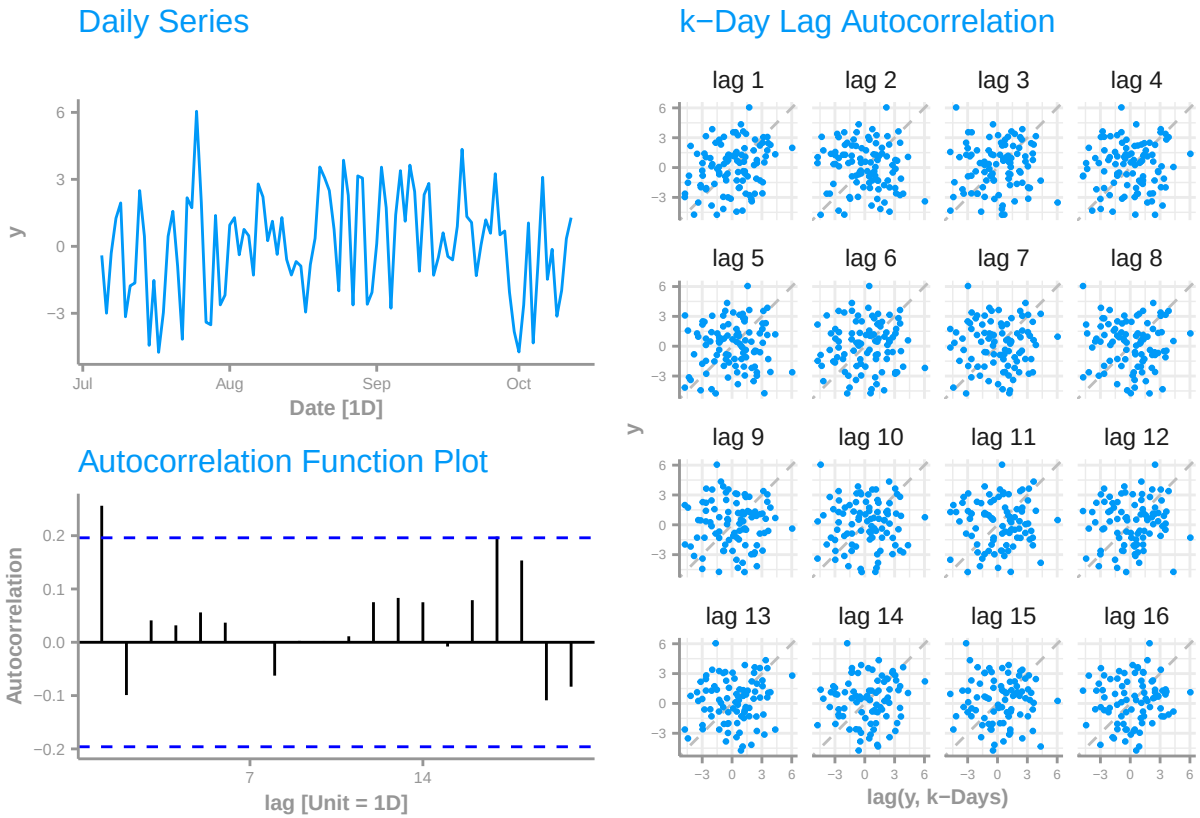
```
dash_graph(
  data = dataset_4
  )
```




In this case we have a daily plot that appears random around 0, the variance doesn't seem to increase or decrease after time passes, and finally we have low autocorrelation and an ACF that seems to move between the dotted lines that represent the 95% confidence interval so we wouldn't be able to reject the null hypothesis of $\rho_k = 0$, meaning that we could be looking at a white noise process.

### (1 point) Dataset Five

```
dash_graph(
  data = dataset_5
  )
```




For this specific series we have a daily plot with which seems centered around 0, and the variance doesn't seem to depend on time, and since we also observe low autocorrelation and an ACF that drops abruptly after the first lag we could be talking about a moving average model of Order 1 for this specific series.

## 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
```

```
unm.original <- get_n_series_table(
  list(overall = 'LNS14000000', male ='LNS14000001', female = 'LNS14000002'),
  start_year = 2000, end_year=2023, tidy=TRUE
)
```

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

```
unemployment <- unm.original
# melting 3 columns into one categorical name
unemployment <- melt(unemployment, variable.name = "name", id = c("year", "month"))
# creating tsibble object with time_index
unemployment <- unemployment %>%
  mutate(time_index = yearmonth(make_datetime(year, month))) %>%
  as_tsibble(index = time_index, key = name)
# ordering columns
unemployment <- unemployment %>%
  select(year, month, time_index, name, value)
# Ordering rows
unemployment <- unemployment[order(unemployment$year, unemployment$month),]
```

```
head(unemployment, 10)
```

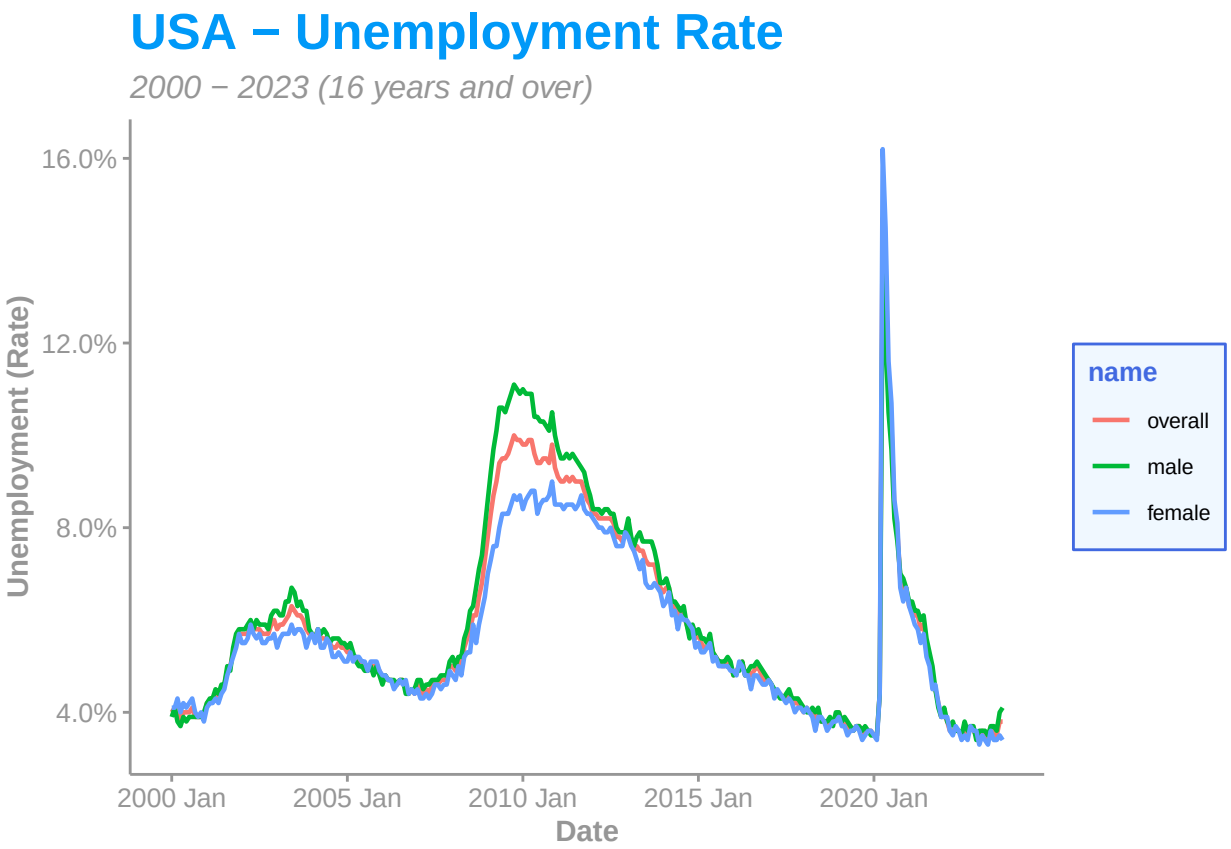
```
## # A tsibble: 10 x 5 [1M]
## # Key:       name [3]
##     year month time_index name    value
##    <int> <int>      <mth> <fct>   <dbl>
##  1  2000     1   2000 Jan overall   4
##  2  2000     1   2000 Jan male      3.9
##  3  2000     1   2000 Jan female    4.1
##  4  2000     2   2000 Feb overall   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
```

## (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.

```r
unm_plot <- ggplot(
  unemployment,
  aes(
    x = time_index,
    y = value / 100
    )
  ) +
  geom_line(
    aes(color = name),
    size = 0.8
    ) +
  labs(
    title = "USA - Unemployment Rate",
    subtitle = "2000 - 2023 (16 years and over)",
    x = "Date",
    y = "Unemployment (Rate)"
    ) +
  theme_classic() +
  theme(
    plot.title = element_text(color = "#0099F8",
                              size = 19,
                              face = "bold"),
    plot.subtitle = element_text(color="#969696",
                                 size = 12,
                                 face = "italic"),
    axis.title = element_text(color = "#969696",
                              size = 11,
                              face = "bold"),
    axis.text = element_text(color = "#969696", size = 10),
    axis.line = element_line(color = "#969696"),
    axis.ticks = element_line(color = "#969696"),
    legend.title = element_text(color = "royalblue",
                                size = 10,
                                face = "bold"),
    legend.background = element_rect(fill ="aliceblue",
                                     color ="royalblue"),
    legend.text=element_text(size=8)
    ) + scale_y_continuous(labels = scales::percent)

unm_plot
```




### (1 point) Plot the ACF and Lags

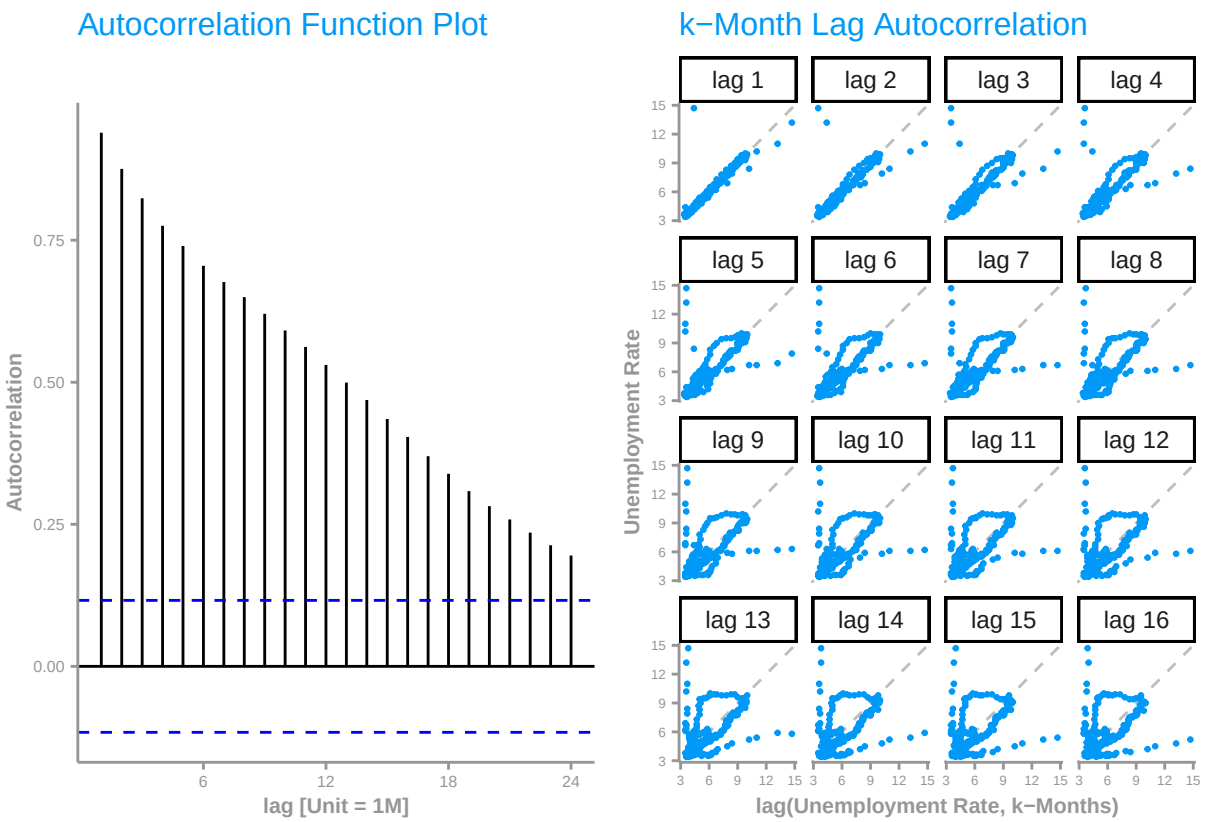
This should feel familiar by now: Produce the ACF and lag plot of the `overall` unemployment series. What do you observe?

```
acf_unm_plot <- ACF(
  fill_gaps(
    unemployment %>% filter(name =="overall"),
    .full = TRUE
    ),
  value
) %>% autoplot() + labs(
  x = "lag [Unit = 1M]",
  y = "Autocorrelation",
  title = "Autocorrelation Function Plot",
) +
theme_classic() +
theme(
  plot.title = element_text(color = "#0099F8",
                            size = 12),
  axis.title = element_text(color = "#969696",
                            size = 8,
                            face = "bold"),
  axis.text = element_text(color = "#969696", size = 6),
  axis.line = element_line(color = "#969696"),
```

```
  axis.ticks = element_line(color = "#969696")
)

lag_unm_plot <- gg_lag(
  unemployment %>% filter(name =="overall"),
  value,
  color = "#0099F8",
  lags = 1:16,
  geom = "point",
  size = 0.5
) + labs(
  x = "lag(Unemployment Rate, k-Months)",
  y = "Unemployment Rate",
  title = "k-Month Lag Autocorrelation"
) +
theme_classic() +
theme(
  plot.title = element_text(color = "#0099F8",
                            size = 12),
  axis.title = element_text(color = "#969696",
                            size = 8,
                            face = "bold"),
  axis.text = element_text(color = "#969696", size = 5),
  axis.line = element_line(color = "#969696"),
  axis.ticks = element_line(color = "#969696")
)

acf_unm_plot | lag_unm_plot
```




We can observe that the mean could be constant, or perhaps vary by a small amount as time passes but variance seems to be related to time. This data is highly autocorrelated with its lags. It might look like an Autoregressive Process with certain random component. Seasonality is not obvious, but perhaps looking at a different unit (certain granularity) it might be more evident.