Forecasting traffic for retail stores

Introduction

The PDF while will include general explanations of the methodologies used.

Objective:

- Include code and business insight
- Explain methodology and modeling process

Challenges: Sensors sometimes do not work due to Sensor issue

Note :

For key business takeaways and insights, please skip to the **conclusion** section in the last page.

This document contains all of the code used, workflow such as methodologies, models, forecasting results and business insight.We will not use exogenous data such as holiday or events.

Workflow

This is the framework that will be used below.

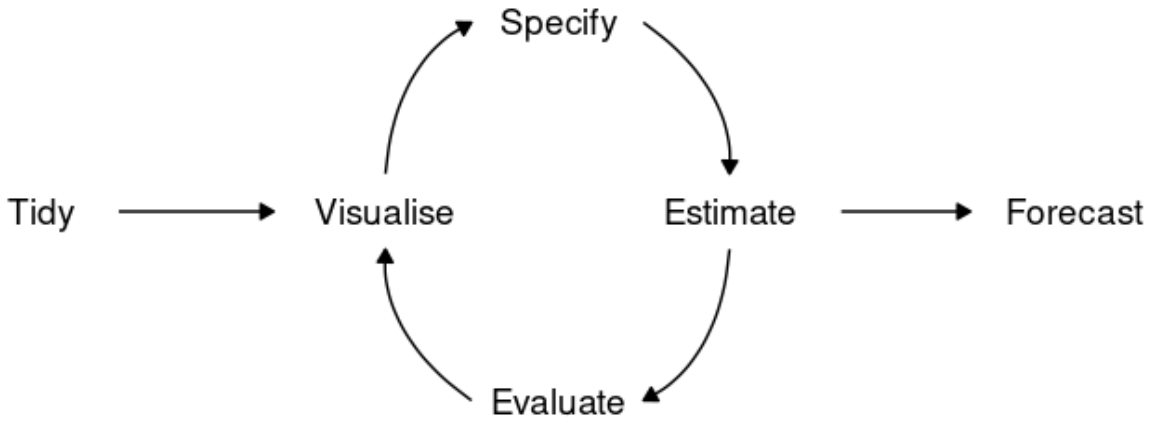


Figure 1: Workflow is based on Forecasting Principles by Robert Hydman.

- Tidy : Load, transform and clean the data to the appropriate format
- Visualize : Explanatory Data Analysis. This includes summary statistics, visualizing our time series, visualizing different components to understand its behavior and to decide what model to build
- Specify : Choose the appropriate model for the time series and objective.
- Estimate: Fit the model
- Evaluate : See the performance / accuracy of the model

Tidy

```
df <- read_excel("/Users/shaanbarca/Desktop/Side projects/data/Traffic_2017-2021.xlsx"
)  #reads excel data

head(df) #  After checking the dataframe we see that the df time series is of daily frequency
```

```
## # A tibble: 6 x 2
##   Date                Traffic
##   <dttm>                <dbl>
## 1 2017-01-01 00:00:00    8520
## 2 2017-01-02 00:00:00    7546
## 3 2017-01-03 00:00:00    6461
## 4 2017-01-04 00:00:00    3401
## 5 2017-01-05 00:00:00    1634
## 6 2017-01-06 00:00:00    1520
```

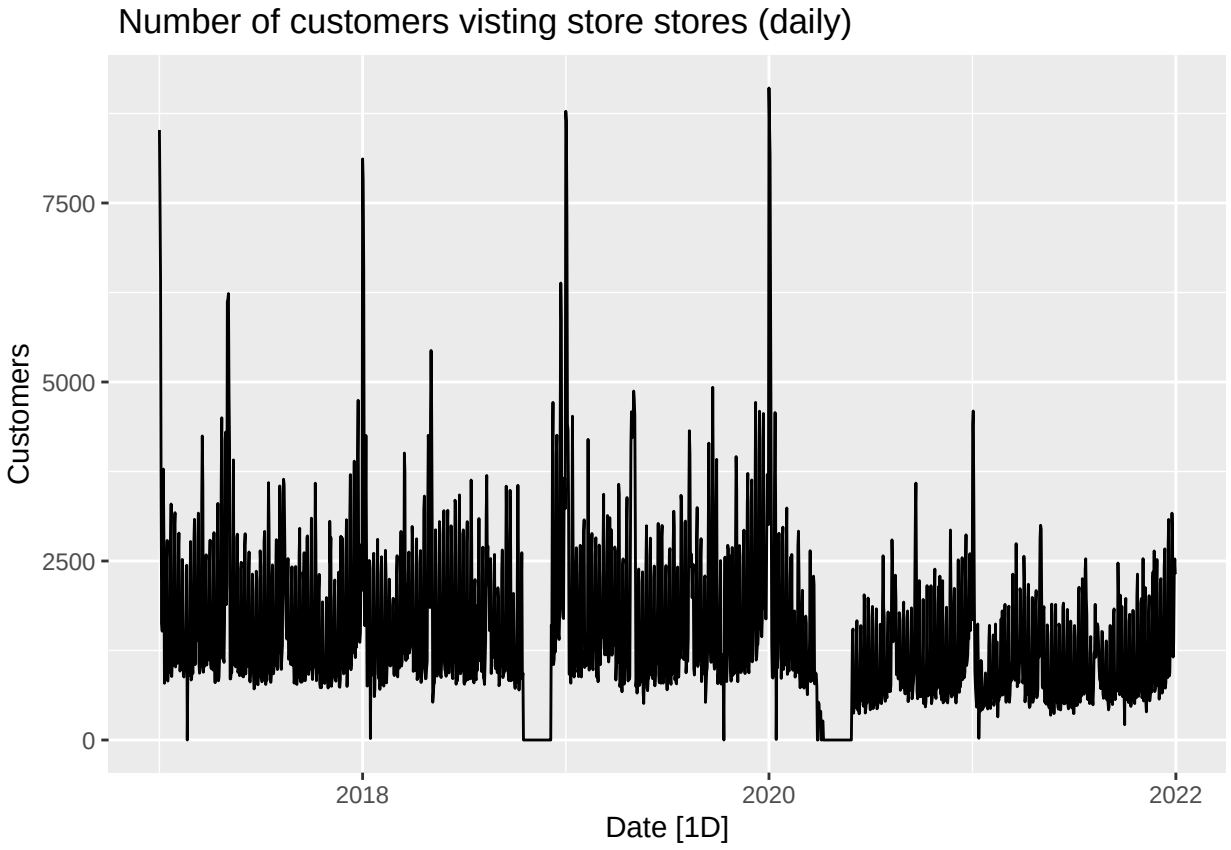
As we are dealing with daily frequency, we convert specify the interval and convert it to a tsibble object which is necessary for us to perform further TS operations.

```
## # A tsibble: 6 x 2 [1D]
##   Date       Traffic
##   <date>       <dbl>
## 1 2017-01-01    8520
## 2 2017-01-02    7546
## 3 2017-01-03    6461
## 4 2017-01-04    3401
## 5 2017-01-05    1634
## 6 2017-01-06    1520
```

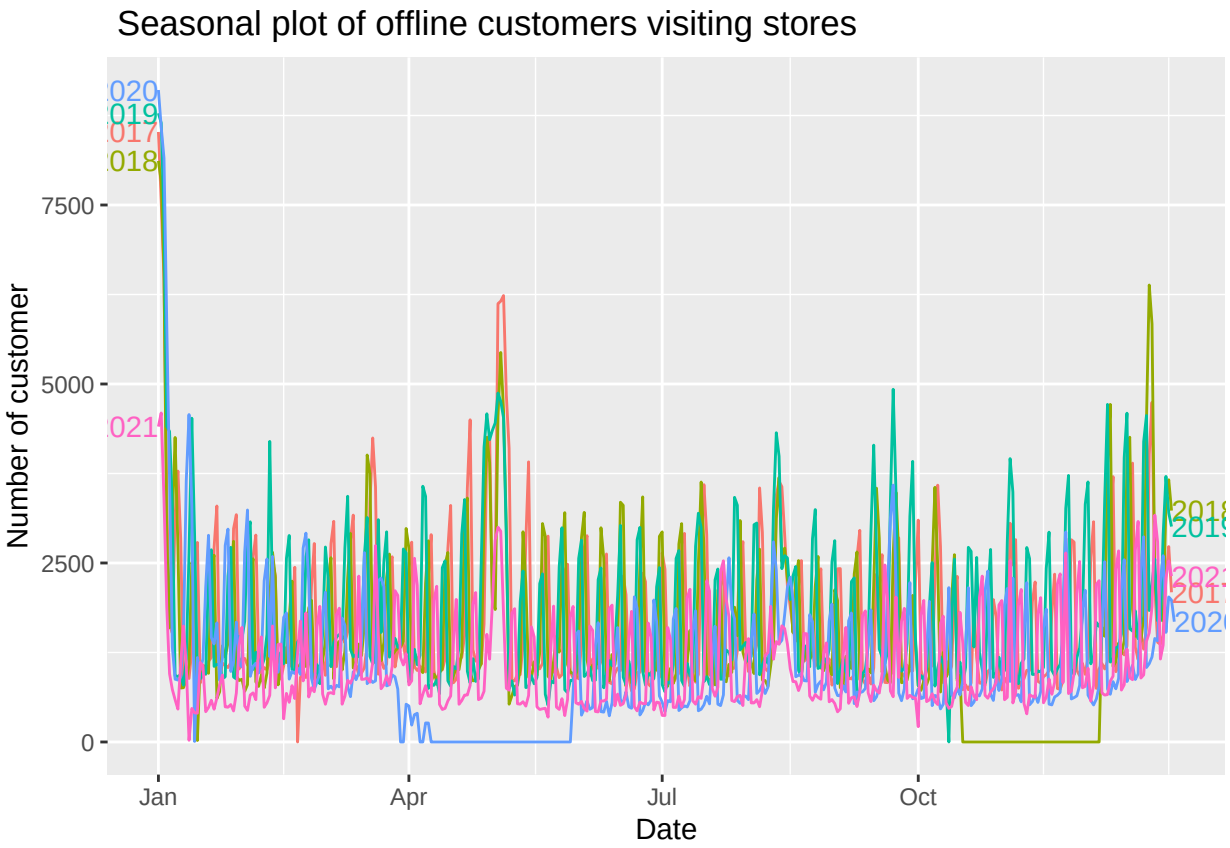
Explanatory Data Analysis / Visualize

```
## # A tibble: 1 x 5
##    `0%` `25%` `50%` `75%` `100%`
##   <dbl> <dbl> <dbl> <dbl>  <dbl>
## 1     0   776  1056  1977   9106
```

We then want to visualize the time series data first to see what we are working with

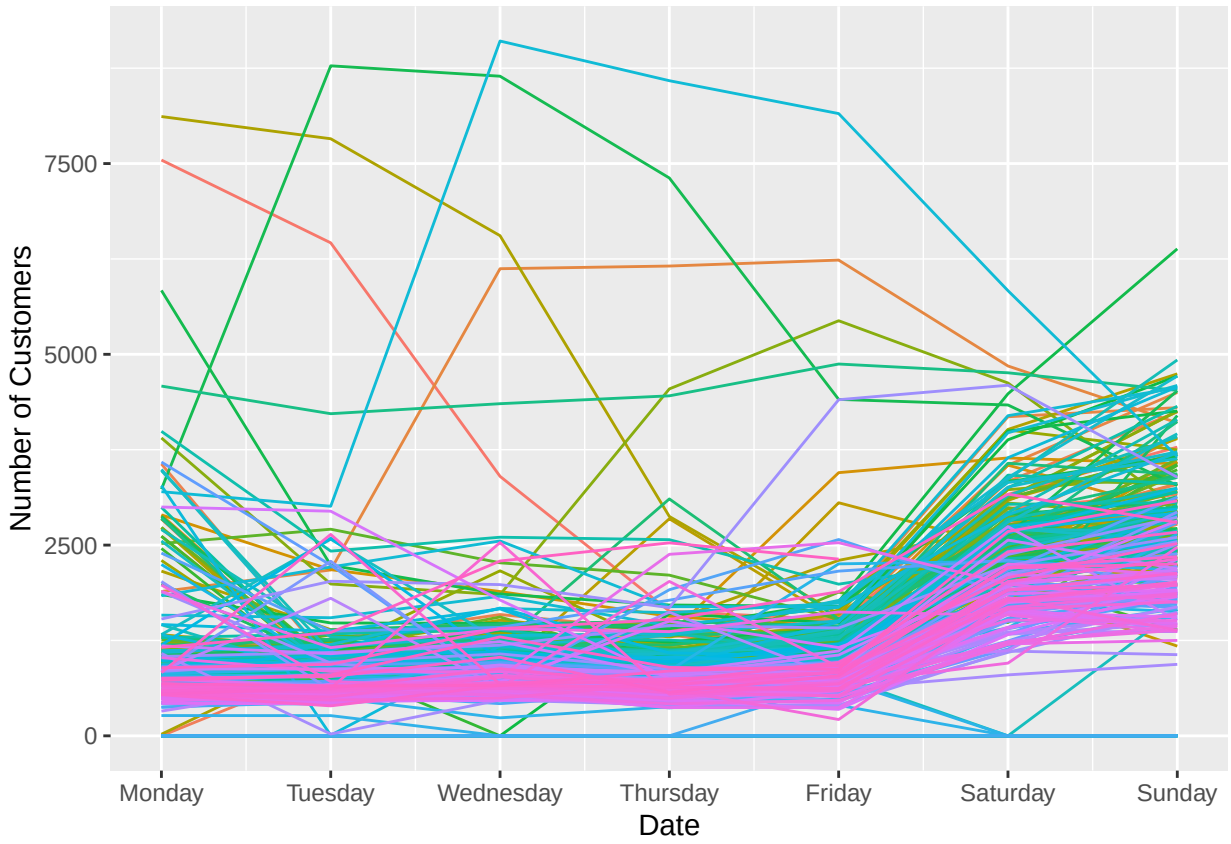


It is important to visualize the time series and see if there are any patterns or potential problems with the data set. We clearly see there are dips. According to the background information given this is due to sensors not functioning (certain days not being recorded). We need to amend this. Since we are given a data set with high granularity (daily), this is not a big problem as we can simple fill it with the median. The median visitors were chosen over the mean because it is less prone to outliers making it a more stable replacement. An alternative is to use a forecasting method such as ARIMA and estimate the missing data.



After filling the missing data with the median, there are still long periods where there were no customers. This could be because either sensors were not working but data was still inputed or for 2020 in particular, this was due to covid and stores were closed. We can double check this.

We can deal with potential outliers later by decomposing the timeseries before we start making our models.



Since we are dealing with daily data, we might be experiencing multiple seasonality. To address this we can visualize date for weekly and annual seasonality.

We first start with weekly seasonality.

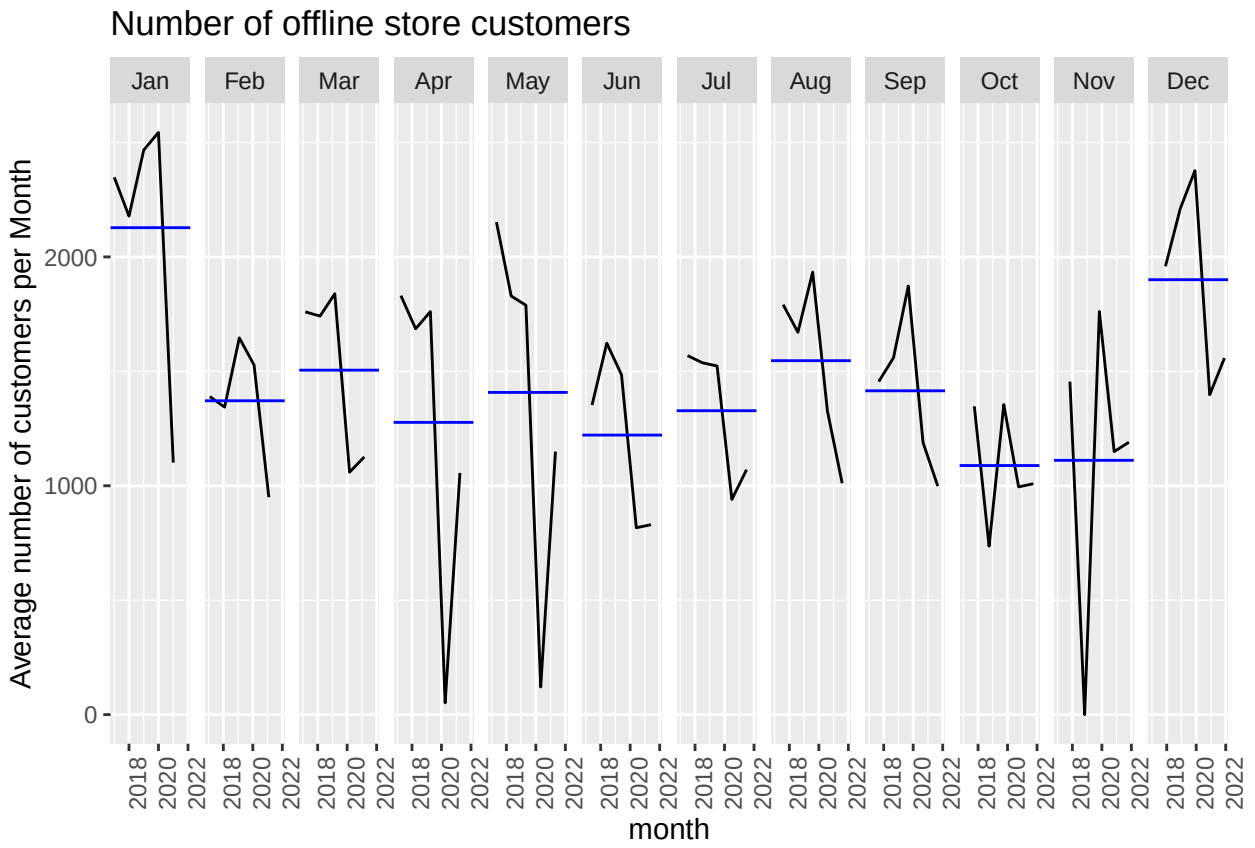
Key insights:

We can see that the number of customers typically rise starting from Friday and peaks on Sunday There are a few outliers but this is most likely due to public holidays but we can check again later to be safe.

```
## # A tibble: 6 x 2
##   month                mean
##   <dttm>              <dbl>
## 1 2017-01-01 00:00:00 2348.
## 2 2017-02-01 00:00:00 1390.
## 3 2017-03-01 00:00:00 1760.
## 4 2017-04-01 00:00:00 1831.
## 5 2017-05-01 00:00:00 2152.
## 6 2017-06-01 00:00:00 1353.
```

We then want to see annual seasonality. To do so, we must first aggregate our data from daily to monthly. To do so we group ny month and get the average customers each month. After this is done we can then see if there is any patterns by each month

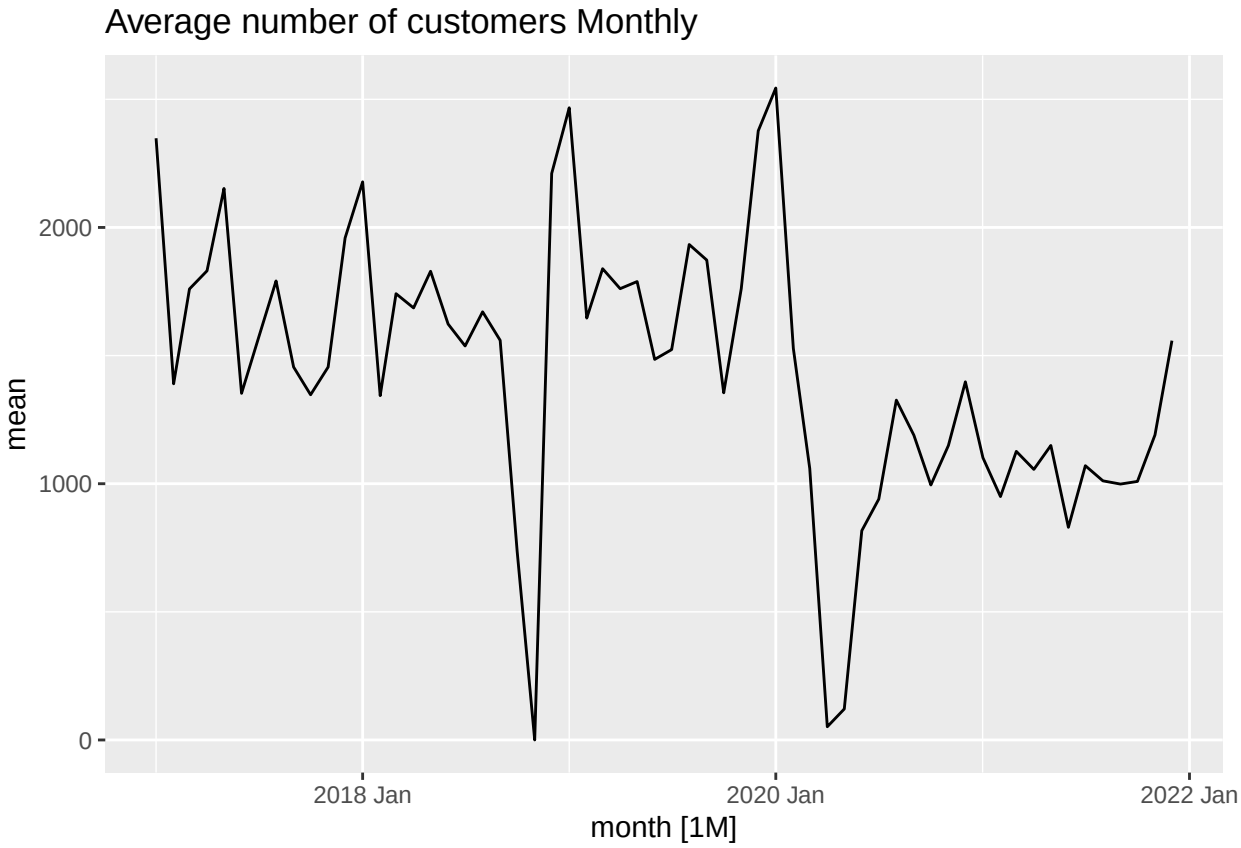
```
monthly_traffic %>% mutate(month = yearmonth(month)) %>%  # we convert df to tsibble
  as_tsibble(index = month) -> monthly_traffic

monthly_traffic %>% gg_subseries(mean) +
  labs( y = "Average number of customers per Month",
        title = "Number of offline store customers")
```




```
# we then want to see seasonal pattern for each month
```

Key Insight

The blue lines indicate the mean number of customers for each month from 2017 - 2022. This allows us to see which are peak and low months.



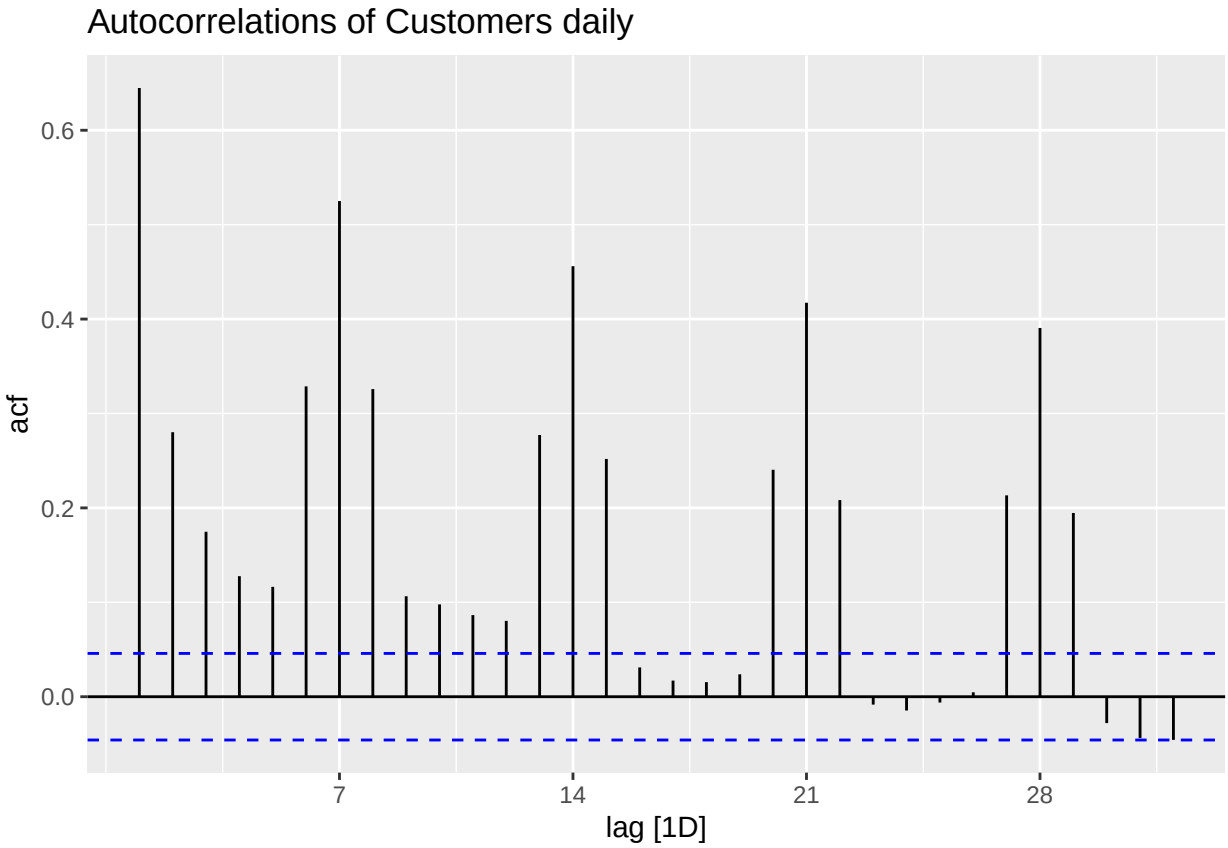
We can see from 2017 - 2022, on average the number of customers typically peak during Jan and December. While the number of customers are usually lowest at October and November

Autocorrelation Before we move onto decomposing and modeling, it is also helpful to check for autocorrelation. Just like correlation autocorrelation measures the relationship between 2 variables. The difference here is that since we only have a single variable, we want to measure the linear relationship between the lagged values of the time series.

An example of this is would be : measuring the relationship of variable yt vs @ yt - 1

```
## Warning: The `...` argument of `PACF()` is deprecated as of feasts 0.2.2.
## i ACF variables should be passed to the `y` argument. If multiple variables are
##   to be used, specify them using `vars(...)`.
```

```
## Warning: ACF currently only supports one column, `Traffic` will be used.
```

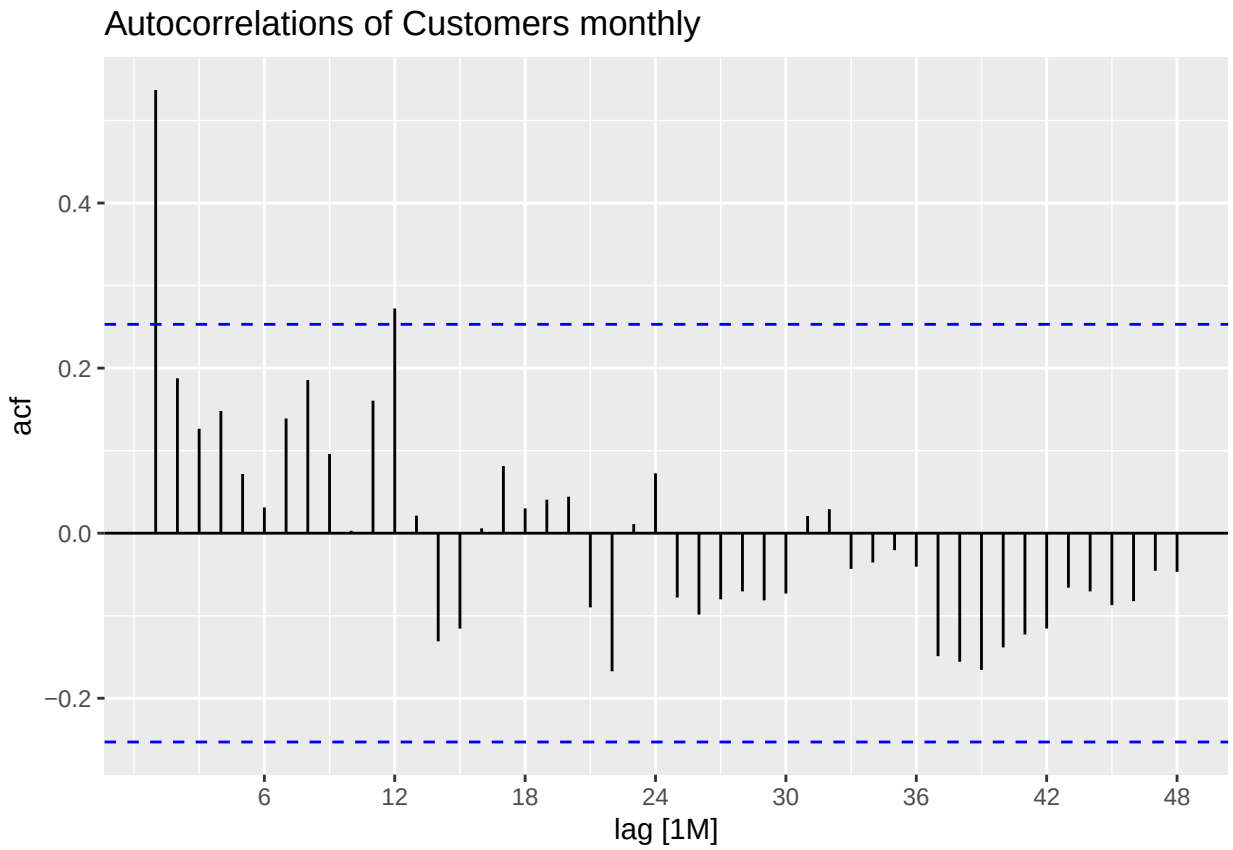


We first measure the autocorrelation of the our disaggregated (daily) data. The lags exhibiting a correlation value past the blue line indicates that its correlations is significantly different from 0.

The blue line is determined using the formula +- 2/ sqrt T where t = length of time series. If a lag spikes greater than this, as show in our data, it is most likely not white noise.

We can see that the series contains trend as seen by the large correlations values of nearby lags that slowly decreases. We also see weekly seasonality from the peaks.

```
monthly_traffic %>% ACF(mean,lag_max = 48) %>% autoplot() + labs(title="Autocorrelations of Customers m
```

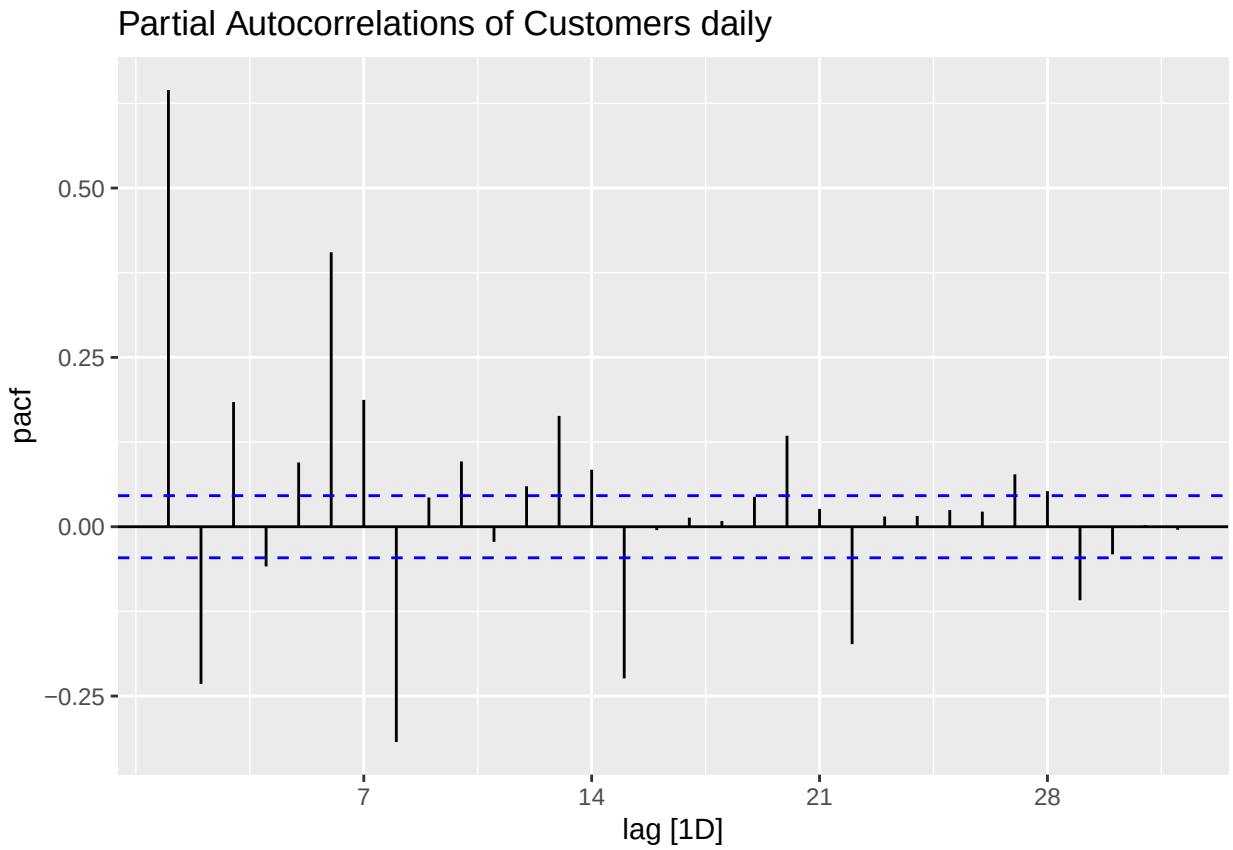


For our aggregated data. We can also see there are 2 months that are correlated.

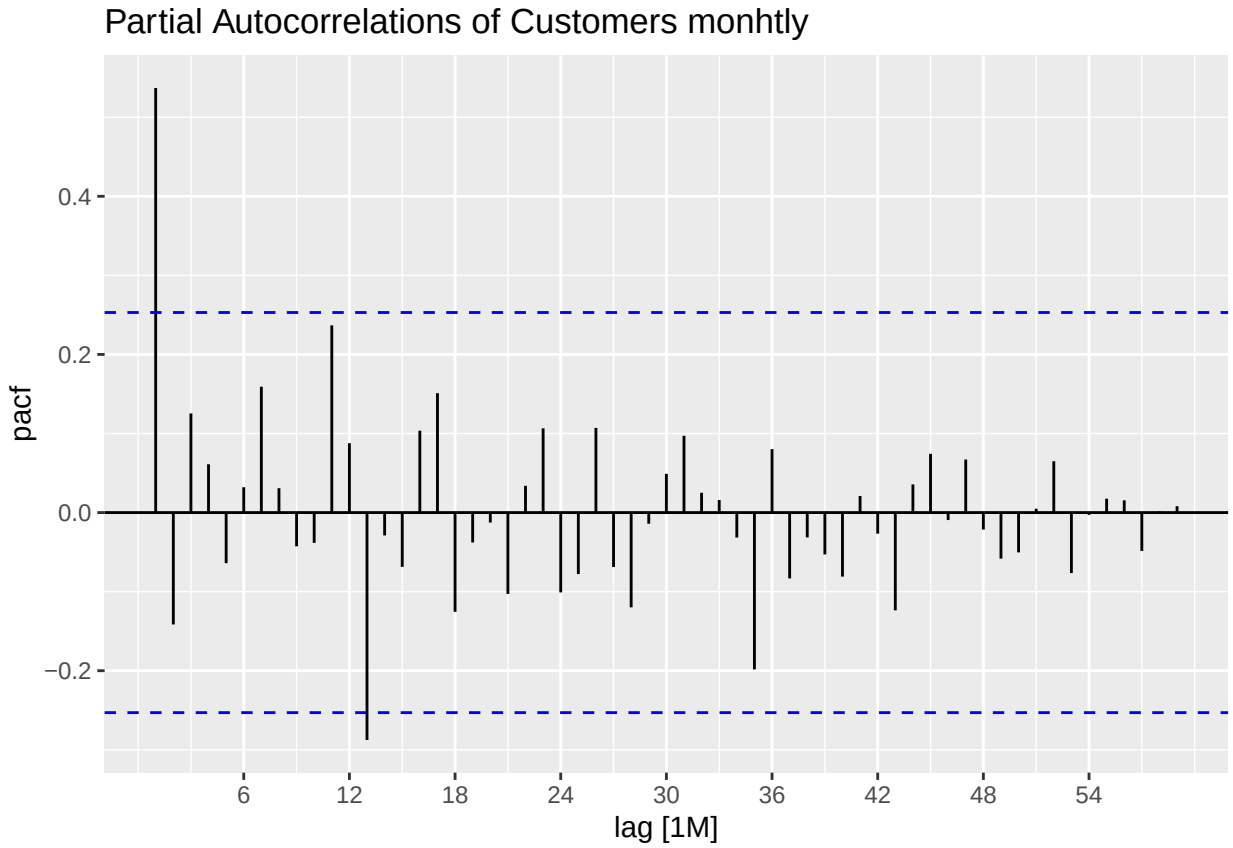
```
fill_traffic_df %>% PACF(Traffic,lag_max ) %>% autoplot() + labs(title="Partial Autocorrelations of Cust
```

```
## Warning: The `...` argument of `PACF()` is deprecated as of feasts 0.2.2.
## i PACF variables should be passed to the `y` argument. If multiple variables
##   are to be used, specify them using `vars(...)`.
```

```
## Warning: PACF currently only supports one column, `Traffic` will be used.
```




```
monthly_traffic %>% PACF(mean,lag_max = 96 ) %>% autoplot() + labs(title="Partial Autocorrelations of Cu
```



We also can see the Partial Autocorrelation (PACF). The PACF show us how certain lags directly for example the number of customers visiting on January might directly affect number of customers visiting on March is store held a Sale event every 2 months.

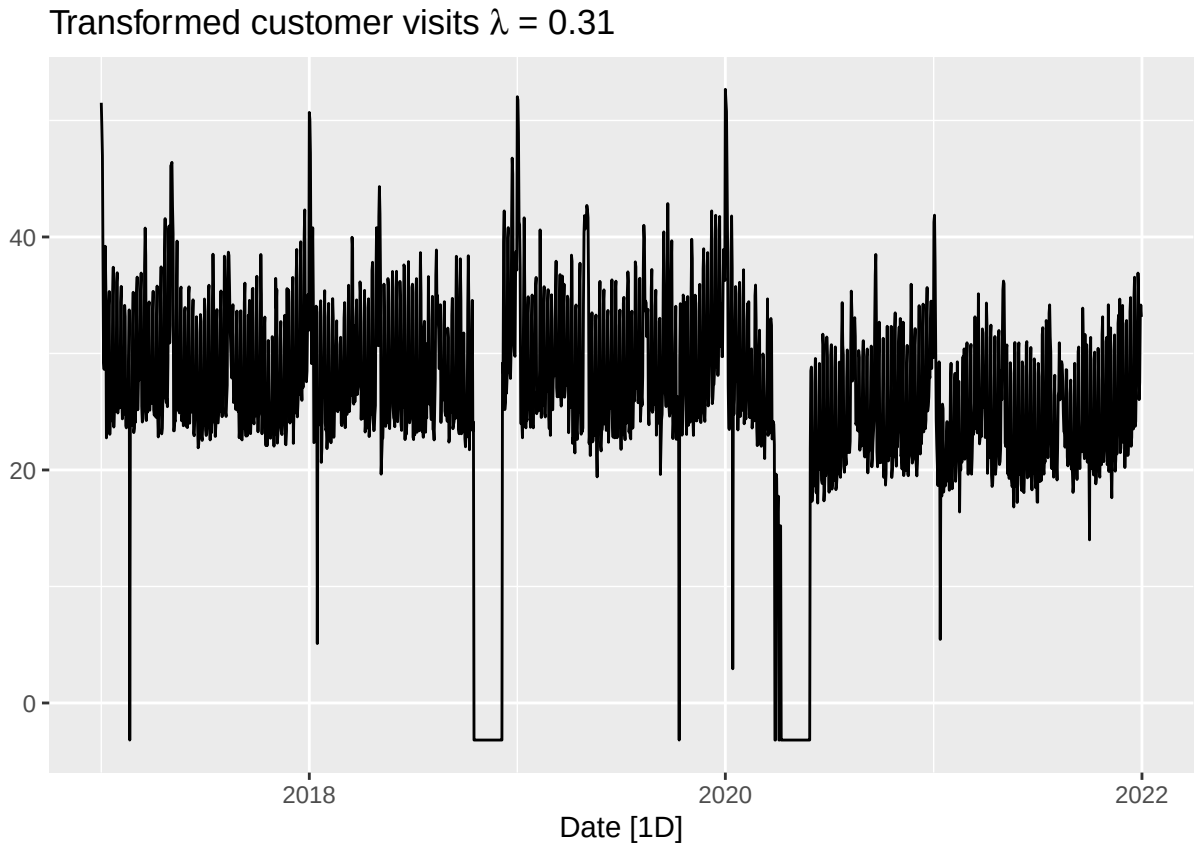
The ACF and PACF will also be more relevant when we start modeling our (S)ARIMA based models.

Decomposition

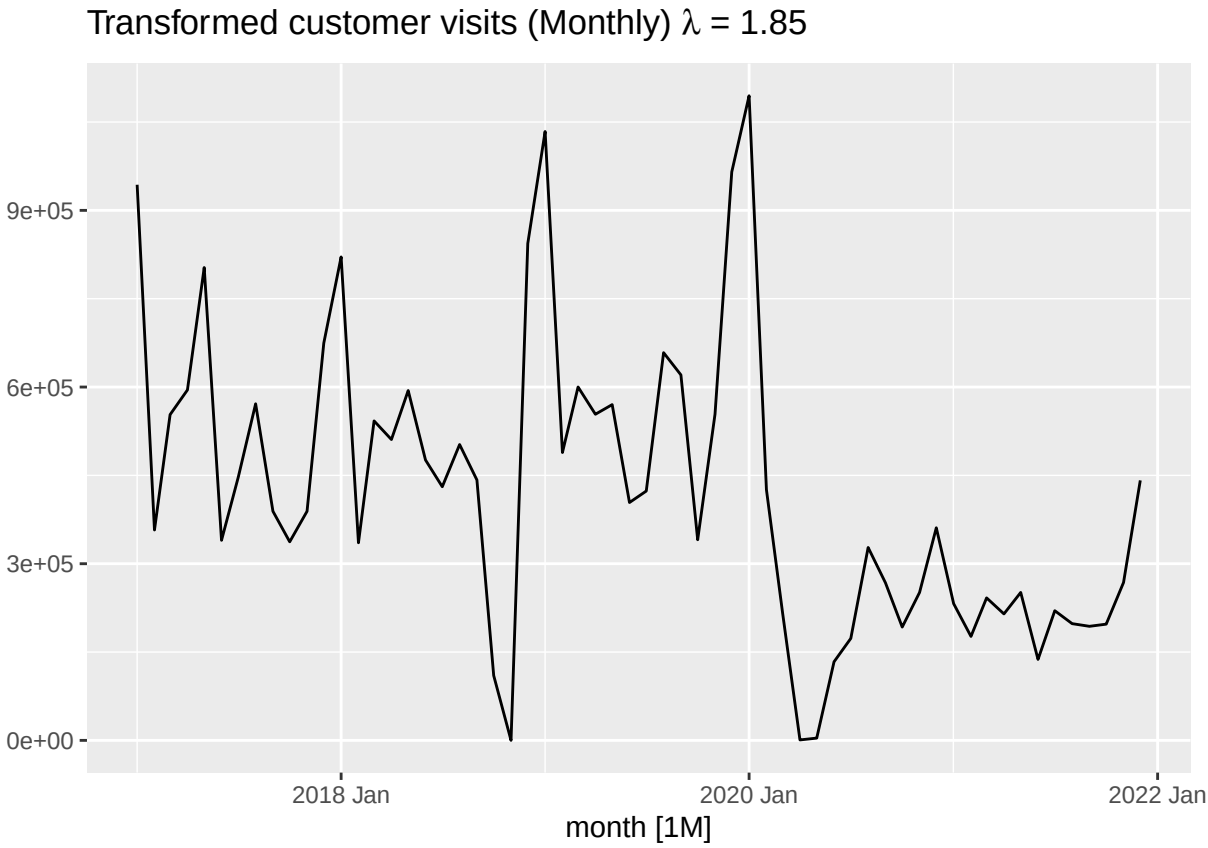
Time series data can be decomposed to individual components. Trend, seasonality and cycle. Not only can this allow us to better understand our Time series but also improve our accuracy when modeling.

Looking at the previous figures, we can see that the magnitude of the seasonal fluctuations does vary slightly with the level of the time series. We can use some mathematical transformations to deal with this.

```
lambda_1 <- fill_traffic_df %>% features(Traffic, features = guerrero) %>%
  pull(lambda_guerrero)

fill_traffic_df %>% autoplot(box_cox(Traffic, lambda_1)) + labs(y = "",title = latex2exp::TeX(paste0(
         "Transformed customer visits $\\lambda$ = ",
         round(lambda_1,2))))
```




```
lambda_2 <- monthly_traffic %>% features(mean, features = guerrero) %>%
  pull(lambda_guerrero)

monthly_traffic %>% autoplot(box_cox(mean, lambda_2)) + labs(y = "",title = latex2exp::TeX(paste0(
         "Transformed customer visits (Monthly) $\\lambda$ = ",
         round(lambda_2,2))))
```



We can see that the variation does not increase and decrease as dramatically as before after we have transformed the data

There are 2 primary ways to decompose time series. Additive and multiplicative decomposition.

multiplicative decomposition is used when the the seasonal magnitude varies with the level of our series (which is what our ts is doing.)

STL decomposition The STL decompostiton is a better version of the classical decomposition. Several drawbacks of classical decomposition are

-
-
-
-

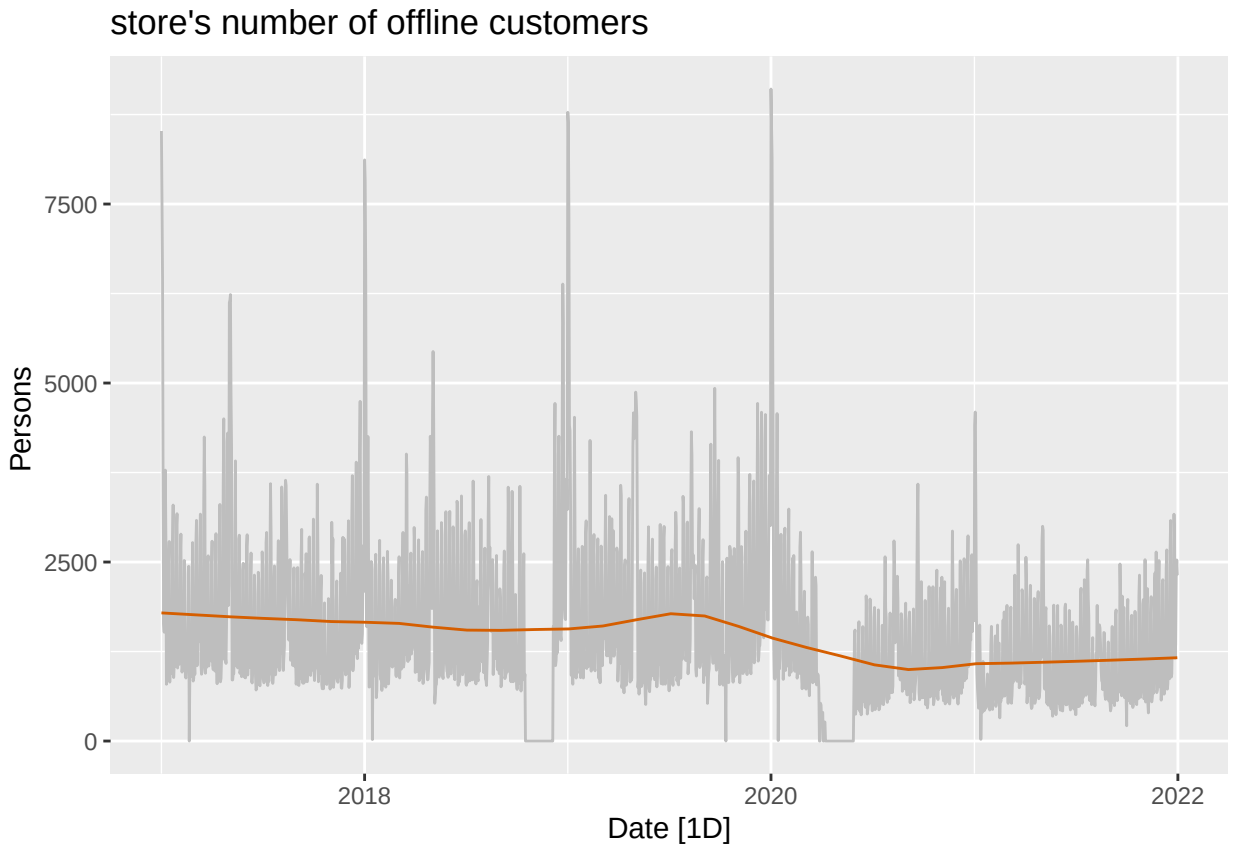
which is why we opted for using STL instead.

```
dcmp1 <- fill_traffic_df %>% model(stl = STL(Traffic))

components(dcmp1)
```

```
## # A dable: 1,826 x 8 [1D]
## # Key:     .model [1]
## # :        Traffic = trend + season_year + season_week + remainder
##    .model Date       Traffic trend season_week season_year remainder
##    <chr>  <date>       <dbl> <dbl>       <dbl>       <dbl>     <dbl>
##  1 stl    2017-01-01    8520 1790.       1632.      5244.    -145.
##  2 stl    2017-01-02    7546 1789.       -152.      6285.    -377.
##  3 stl    2017-01-03    6461 1789.       -418.      5351.    -262.
```

```
##  4 stl    2017-01-04    3401 1788.       -634.      2553.    -307.
##  5 stl    2017-01-05    1634 1788.       -745.      1018.    -427.
##  6 stl    2017-01-06    1520 1787.       -569.       450.    -149.
##  7 stl    2017-01-07    3164 1787.        909.       459.       9.23
##  8 stl    2017-01-08    3784 1787.       1610.        87.6    300.
##  9 stl    2017-01-09    2842 1786.       -178.       -14.2   1248.
## 10 stl    2017-01-10     793 1786.       -420.      -359.    -214.
## # ... with 1,816 more rows, and 1 more variable: season_adjust <dbl>
```

```
components(dcmp1) %>%
  as_tsibble() %>%
  autoplot(Traffic, colour="gray") +
  geom_line(aes(y=trend), colour = "#D55E00") +
  labs(
    y = "Persons ",
    title = "store's number of offline customers"
  )
```



The Trend plot ignores seasonality and white noise and highlights the overall movement of the series.

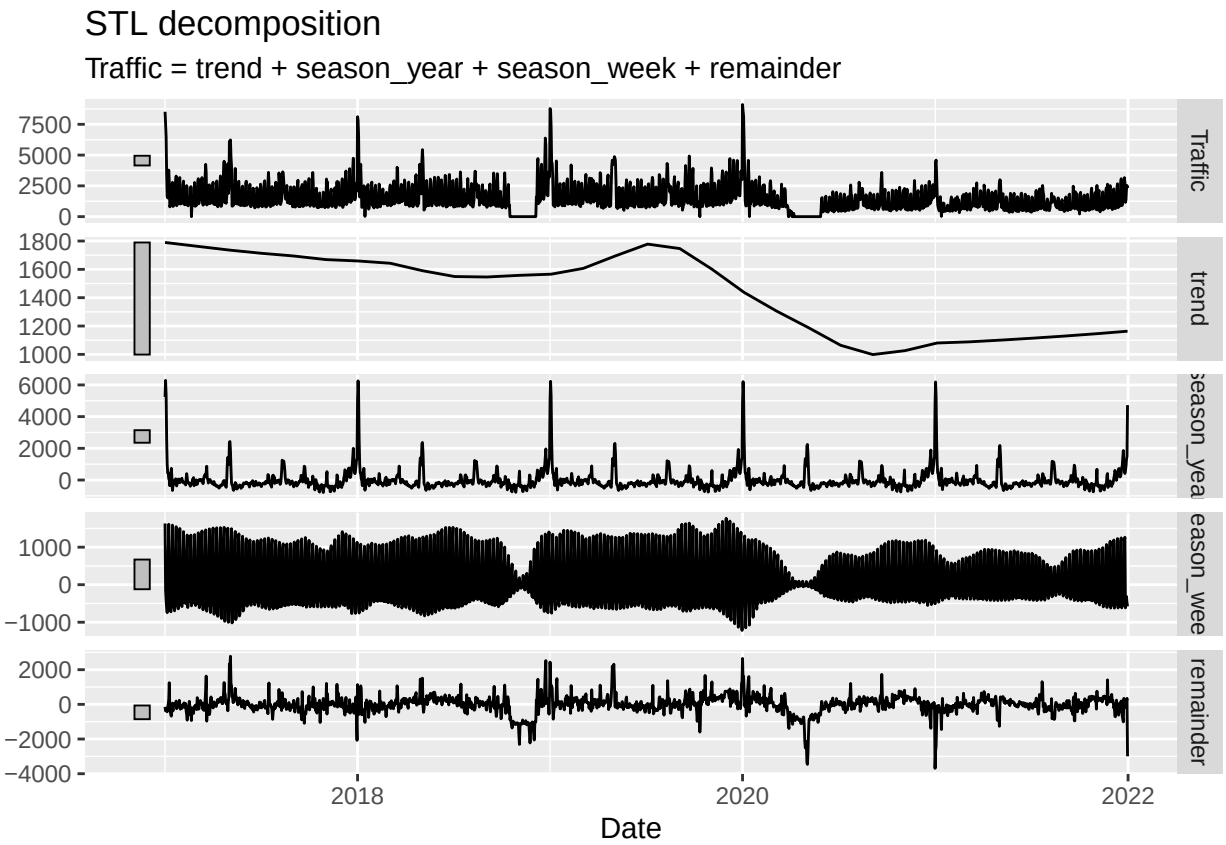
We see here that the number of offline customers has been trending downwards but started to rise during 2019. Perhaps There was a successful campaign. After checking the news it seemed the rise was probably due to the Michael B Jordan campaign where he became first ever men's ambassador. https://www.complex.com/style/2019/01/michael-b-jordan-stars-store-spring-2019-campaign

Perhaps seeing data of sales of menswear and women's wear items will demonstrate the effectiveness of the campaign

We then see it decreasing again in 2020 and even more so during the 2020 most likely due to covid but has

started to regain the number of offline customers overtime but has yet to reach pre-covid averages.

```
components(dcmp1) %>% autoplot()
```



We then also fully decompose the function to 4 parts. The Trend, Yearly seasonality, Weekly seasonality and Remainder (aka White Noise). Due to the time series being daily, as mentioned before it contains multiple seasonality

We can also perform the same operations to the aggregated data to see a cleaner version of the decomposition.

What we do see from our decomposition is there are clear trend and seasonal patters. This means our time series is not stationary.

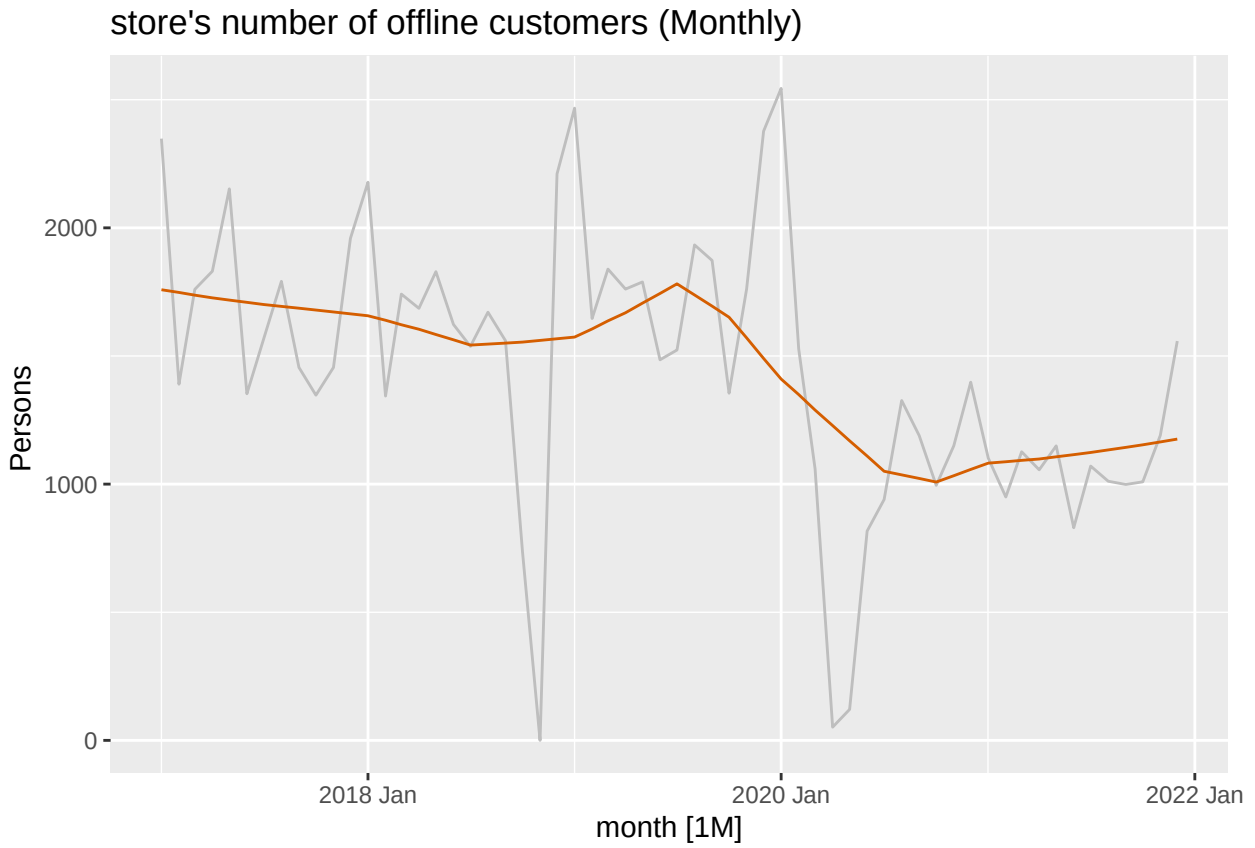
The output of our decomposition is consistent with our previous visualizations where customers peak tend to peak during certain days of the week and months.

```
dcmp2 <- monthly_traffic %>% model(stl = STL(mean))

components(dcmp2)
```

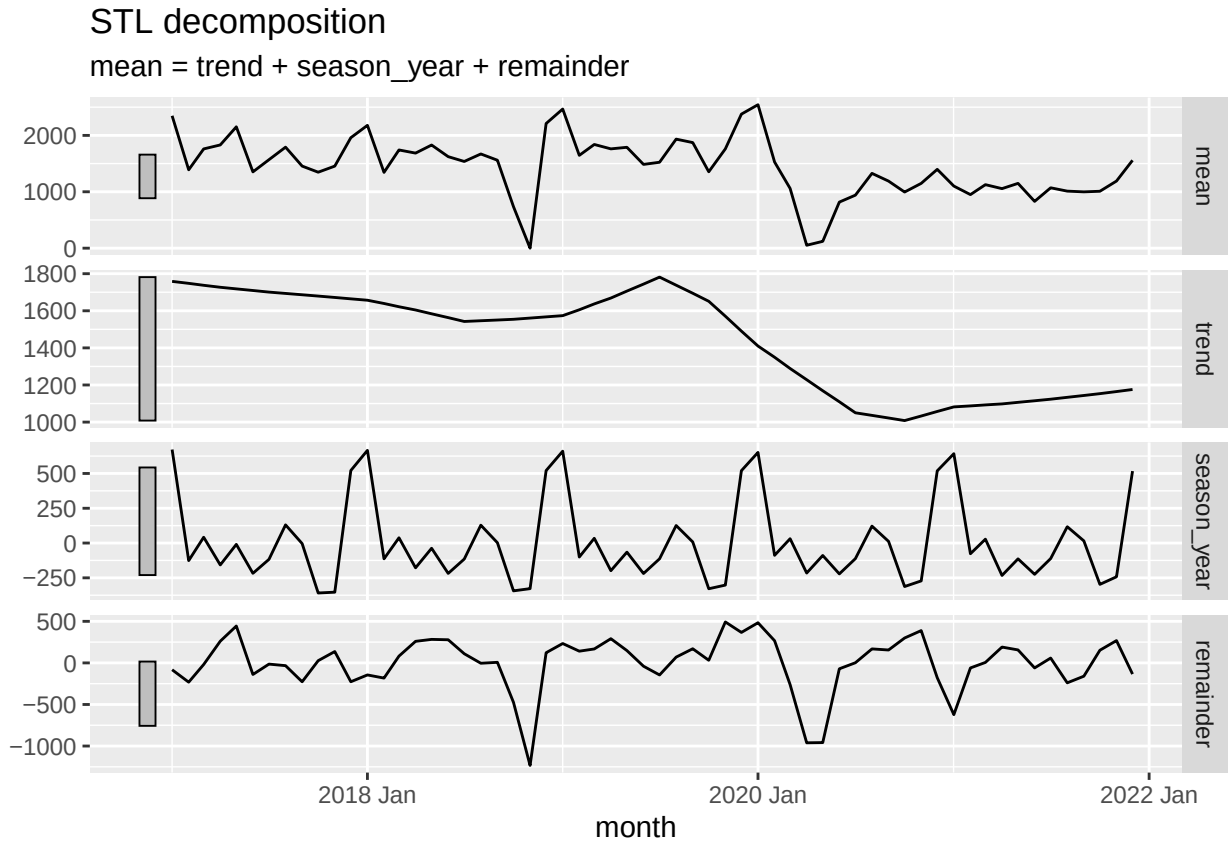
```
## # A dable: 60 x 7 [1M]
## # Key:     .model [1]
## # :        mean = trend + season_year + remainder
##    .model    month  mean trend season_year remainder season_adjust
##    <chr>     <mth> <dbl> <dbl>       <dbl>     <dbl>         <dbl>
##  1 stl    2017 Jan 2348. 1758.      672.       -82.3         1676.
##  2 stl    2017 Feb 1390. 1748.     -126.      -232.          1516.
##  3 stl    2017 Mar 1760. 1737.       41.6      -19.0         1718.
##  4 stl    2017 Apr 1831. 1727.     -157.       261.          1988.
##  5 stl    2017 May 2152. 1718.       -9.64     444.          2162.
##  6 stl    2017 Jun 1353. 1709.     -217.      -139.          1570.
```

```
##  7 stl    2017 Jul 1569. 1701.     -117.       -14.6         1686.
##  8 stl    2017 Aug 1791. 1694.      131.       -33.0         1661.
##  9 stl    2017 Sep 1455. 1686.       -3.77    -227.          1459.
## 10 stl    2017 Oct 1347. 1679.     -359.        26.9         1706.
## # ... with 50 more rows
```

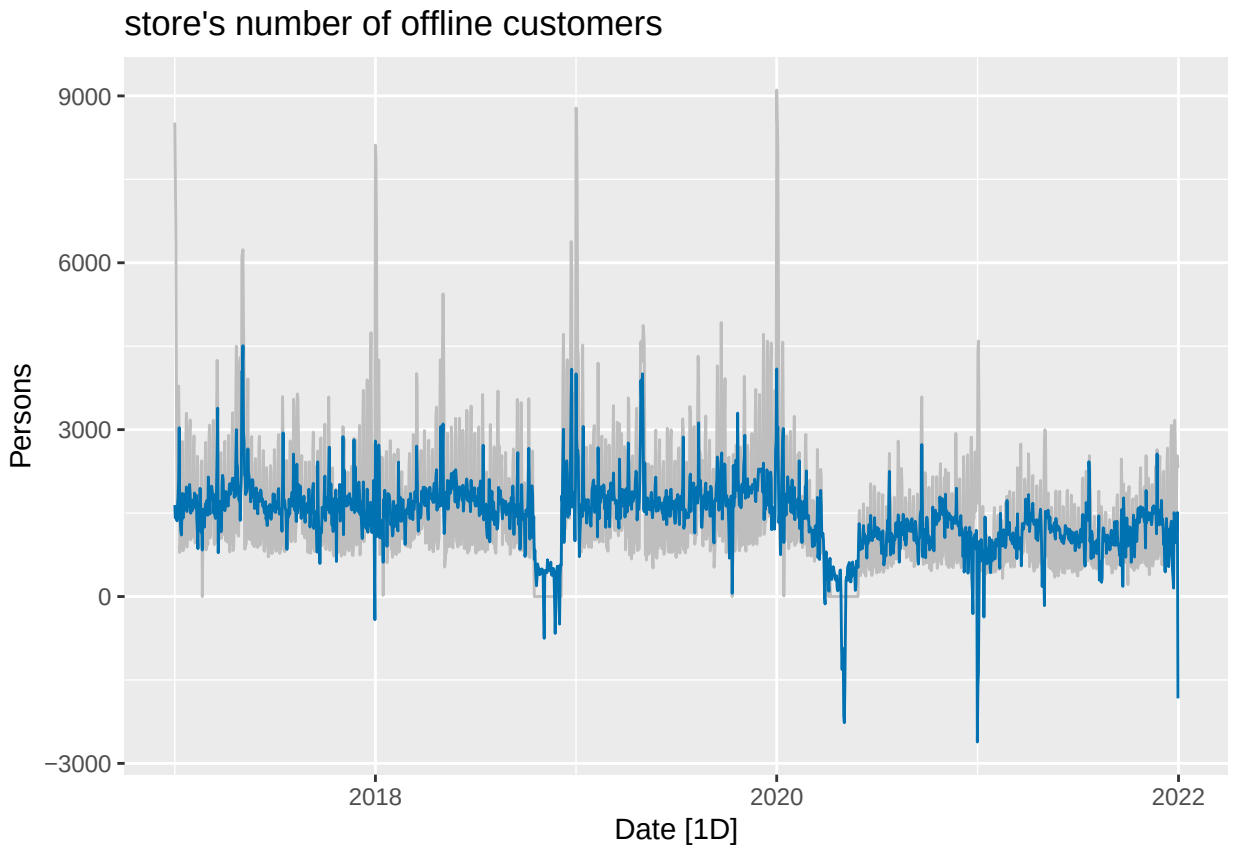
```
components(dcmp2) %>%
  as_tsibble() %>%
  autoplot(mean, colour="gray") +
  geom_line(aes(y=trend), colour = "#D55E00") +
  labs(
    y = "Persons ",
    title = "store's number of offline customers (Monthly)"
  )
```



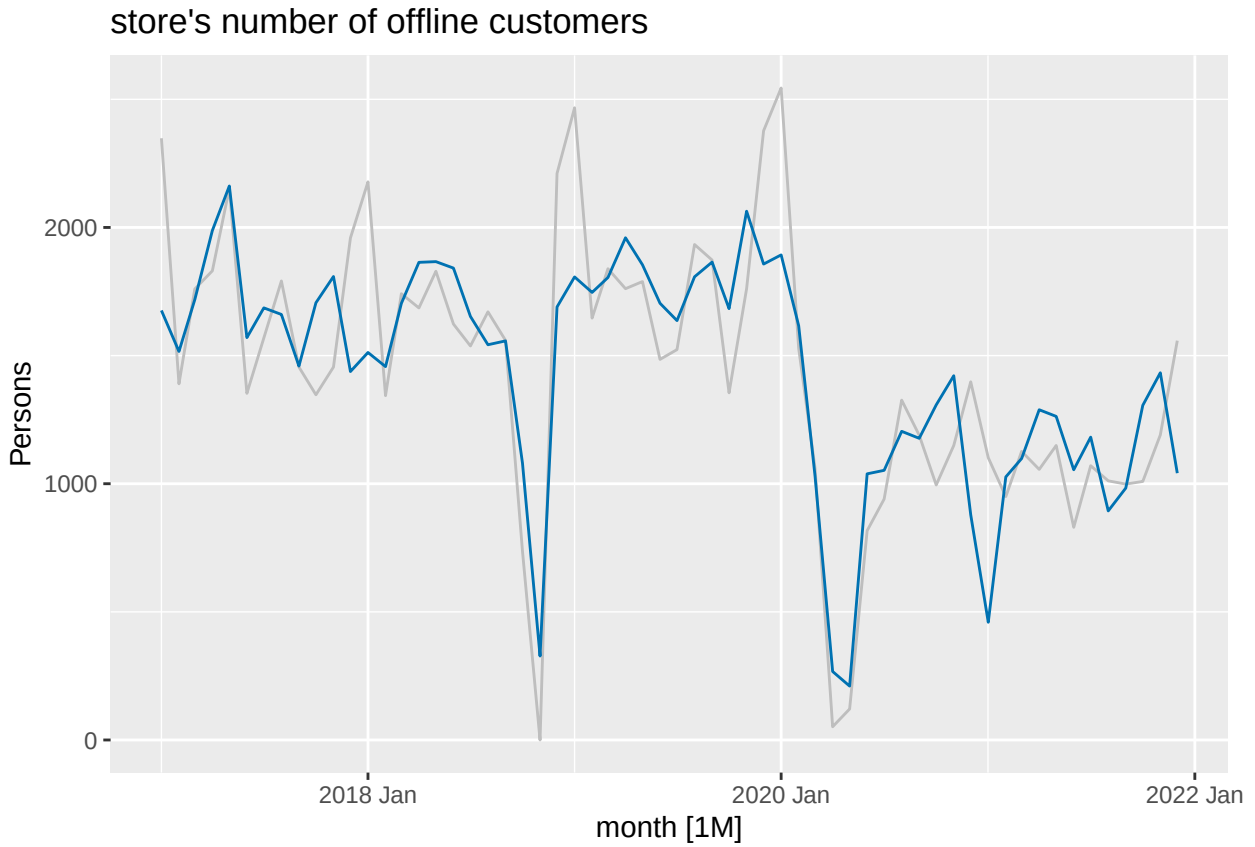

```
components(dcmp2) %>% autoplot()
```




```
components(dcmp1) %>%
  as_tsibble() %>%
  autoplot(Traffic, colour = "gray") +
  geom_line(aes(y=season_adjust), colour = "#0072B2") +
  labs(y = "Persons",
       title = "store's number of offline customers")
```




```
components(dcmp2) %>%
  as_tsibble() %>%
  autoplot(mean, colour = "gray") +
  geom_line(aes(y=season_adjust), colour = "#0072B2") +
  labs(y = "Persons",
       title = "store's number of offline customers")
```



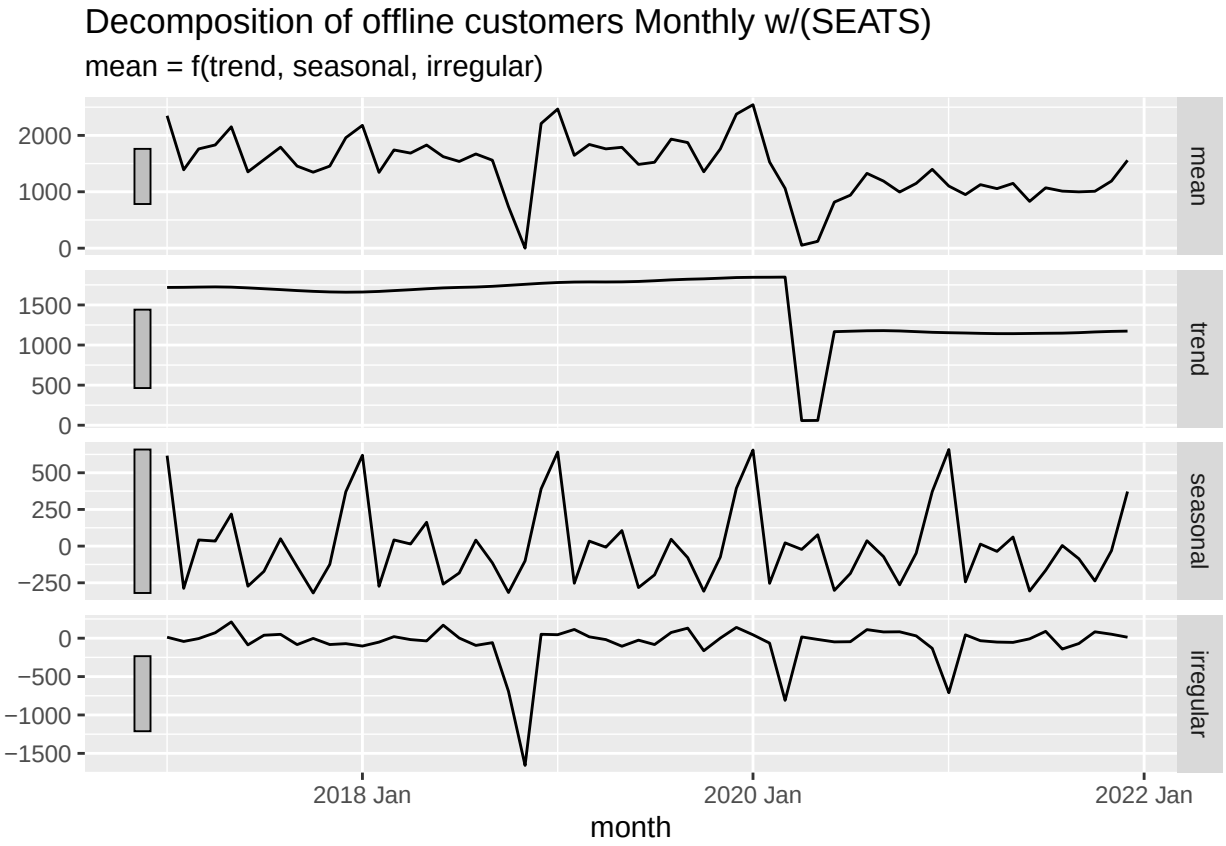
The 2 plots above show the seasonal adjusted data. These still contain the trend and white noise components. This plot allows us to see the overall the number of people visiting offline stores ignoring the seasonal variation. We can see that there was now a clear decline though it is slowly recovering. There are also sharp declines which may indicate the broken sensors. We will address the issue of outliers later

We were able to get the trend cycle of the data by getting the moving average. The moving average allows us to get the average value of our data for a specified period. This allows us to have a smoother line the is less prone to fluctuations.

This was applied to get moving averages of seasonal data by getting "centered moving averages" or moving averages of moving average".

For this data in particular since we are dealing with daily data with weekly seasonality we can use a 7-MA to estimate the trend cycle.

```
seats_dcmp <- monthly_traffic %>%
  model(seats = X_13ARIMA_SEATS(mean ~ seats())) %>%
  components()
autoplot(seats_dcmp) +
  labs(title =
    "Decomposition of offline customers Monthly w/(SEATS)")
```



Seasonal Features

```
fill_traffic_df %>%
  features(Traffic, feat_stl)
```

```
## # A tibble: 1 x 9
##   trend_strength seasonal_strength_~ seasonal_peak_w~ seasonal_trough~ spikiness
##            <dbl>               <dbl>            <dbl>            <dbl>     <dbl>
## 1          0.760               0.682                1                3   219633.
## # ... with 4 more variables: linearity <dbl>, curvature <dbl>,
## #   stl_e_acf1 <dbl>, stl_e_acf10 <dbl>
```

```
monthly_traffic %>%
  features(mean, feat_stl)
```

```
## # A tibble: 1 x 9
##   trend_strength seasonal_strength_~ seasonal_peak_y~ seasonal_trough~ spikiness
##            <dbl>               <dbl>            <dbl>            <dbl>     <dbl>
## 1          0.446               0.450                1               10 18956864.
## # ... with 4 more variables: linearity <dbl>, curvature <dbl>,
## #   stl_e_acf1 <dbl>, stl_e_acf10 <dbl>
```

Takeaways

The number of customers peaks during the 1st week and 1st month (January).

The number of customers dips during the 3rd week and the 10th month (October).

Modeling

In this section, there are several process that will be covered other than just selecting models. We will also modify our time series to prepare for modeling, how to evaluate our models, how to select the best models and prediction interval.

Stationarity As we see can see from the visualizations and decomposition that has been produced, there is clear trend and seasonality. We need to address this before we fit certain models such as ARMA based models (SARIMA, ARIMA, SARIMAX).

To solve this we can stabilize our time series through difference. By computing difference of consecutive observation, we it will stabilize the mean and reduce or remove trend and seasonality.

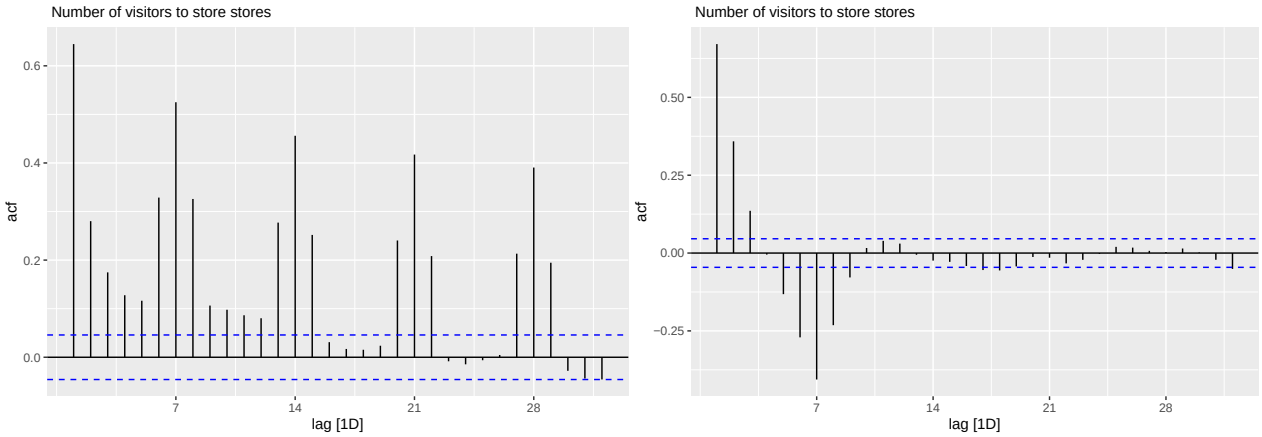
Certain plot such as ACF can be used to check whether or not the series is stationary or not. If a series has been transformed from non stationary to stationary, it should look like white noise.

There also more objective tests such as unit root tests to detect for non stationary. If the p value of the test is above 0.01 then it is stationary, if not we have to difference the data again.

We typically difference the data at most 2 times (Second order difference). Going past this can result to false correlation. We should use the min number of difference for our data to be stationary.

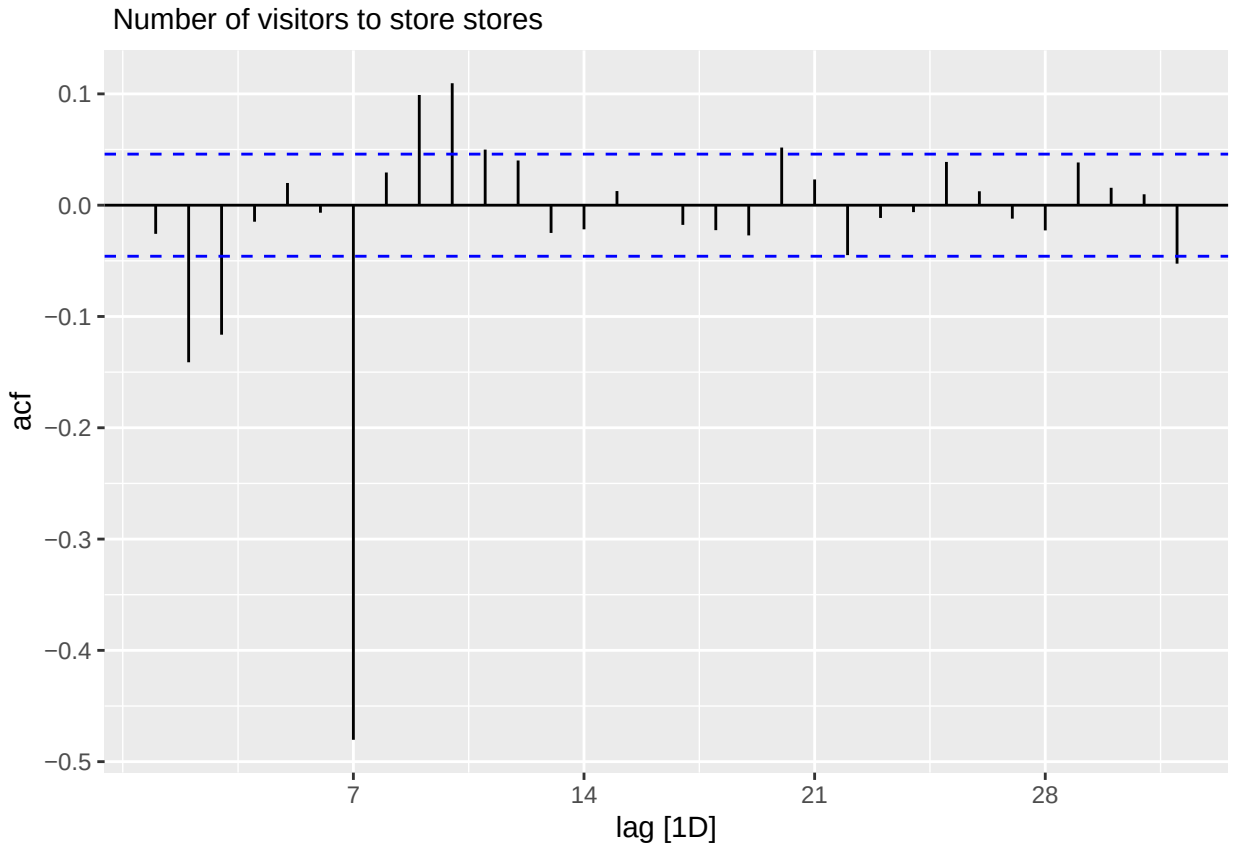
As our data contains both seasonal and non seasonal data, we can apply both seasonal differences and non seasonal differences.

```
# differencing our series


fill_traffic_df %>% ACF(Traffic) %>%
  autoplot() + labs(subtitle = " Number of visitors to store stores")

fill_traffic_df %>% ACF(difference(Traffic,7)) %>%
  autoplot() + labs(subtitle = " Number of visitors to store stores")
```



As applying a seasonal difference to our data has reduced correlation but is still yet to be stationary, we will apply an additional first order difference.

```
fill_traffic_df %>% ACF(difference(difference(Traffic,7),1)) %>%
  autoplot() + labs(subtitle = " Number of visitors to store stores")
```



A more objective test rather than simply eyeballing our ACF plots is by using a unit root test. We will use two forms of unit root test, 1 to determine if a first order differencing is required and another to check if a seasonal difference is required.

- For the first unit test a p-value less than 0.1 means the the data is not stationary
- For the second unit test a value < 0.64 requires no seasonal difference.

```
fill_traffic_df %>%
  features(Traffic, unitroot_kpss)
```

```
## # A tibble: 1 x 2
##   kpss_stat kpss_pvalue
##       <dbl>       <dbl>
## 1      2.61        0.01
```

```
fill_traffic_df %>%
  mutate(diff_traffic = difference(Traffic,1)) %>%
  features(diff_traffic, unitroot_kpss)
```

```
## # A tibble: 1 x 2
##   kpss_stat kpss_pvalue
##       <dbl>       <dbl>
## 1    0.0406         0.1
```

```
fill_traffic_df %>%
  features(Traffic, unitroot_ndiffs)
```

```
## # A tibble: 1 x 1
##   ndiffs
##    <int>
```

```
## 1      1
```

As we can see from the result of our data. A seasonal difference and a first difference is both required.

Specify

ARIMA

One of the models suggested to be used was the ARIMA model. ARIMA or AutoRegressive Integrated Moving Average model is composed of the AR, MA and the order of difference, which we have already obtained in the previous part.

Auto Regressive: The auto regressive part of is similar to a linear / multi-linear regression. The only difference is that were are regressing on the past values of our series.

We can select the order of our AR(P) model to best fit our data.

Moving Average : This is the same as our AR model however the MA model instead uses past errors as predictors.

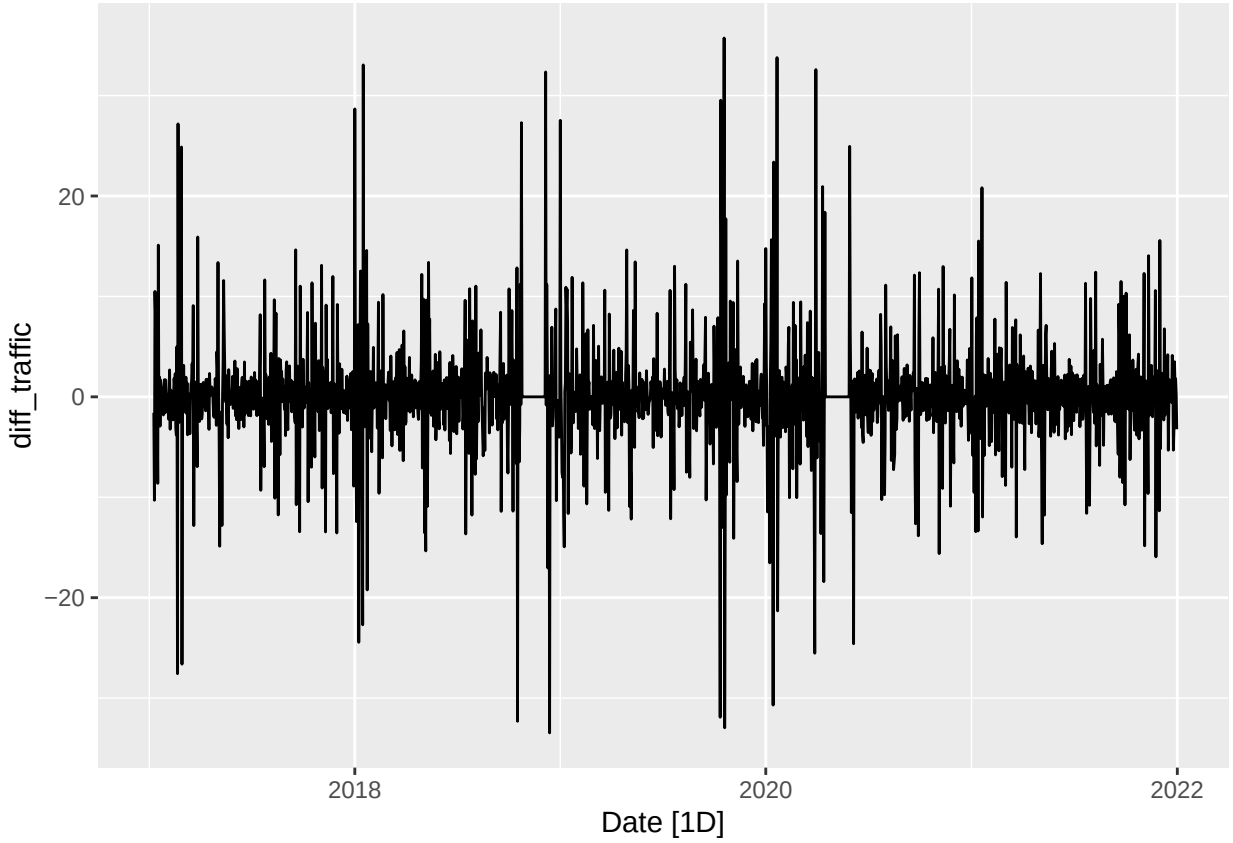
By combing them we can have a ARIMA(p,d,q) model.

- p = Order of AR
- q = Order of MA
- d = degree of first differencing involved.

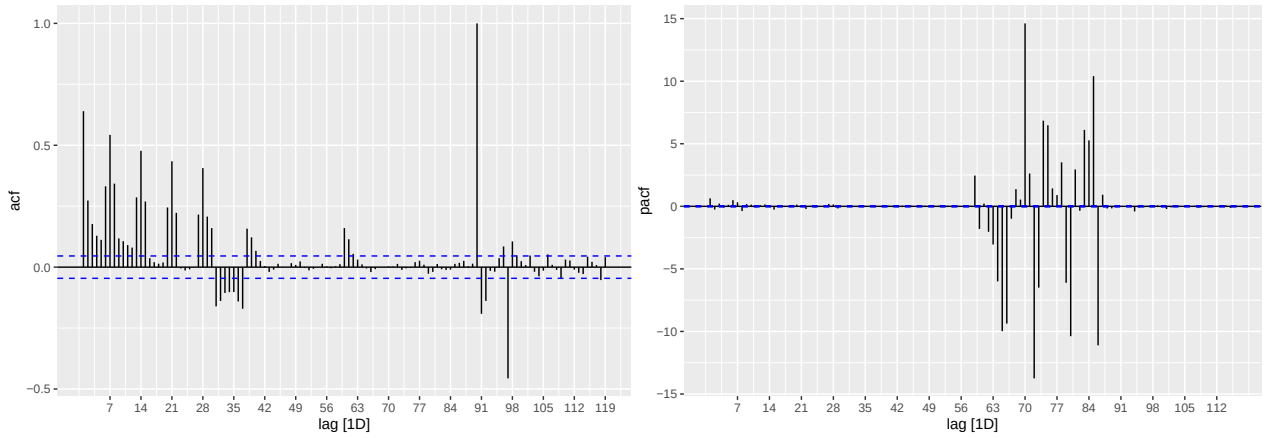
As mentioned before, we can use a PACF plot and an ACF plot to determine the order of our p,q variables.

```
fill_traffic_df %>%
  mutate(diff_traffic = difference(difference(box_cox(Traffic,lambda_1),7),1)) -> differenced_traffic

differenced_traffic %>% autoplot(diff_traffic)
```

```
## Warning: Removed 8 row(s) containing missing values (geom_path).
```




```
differenced_traffic %>% ACF(differenced_traffic) %>% autoplot()

differenced_traffic %>% PACF(differenced_traffic) %>% autoplot()
```



ACF and PACF plots can be used to determined the value of p and q if the data is a ARIMA(p,d,0) or ARIMA(0,d,q).

Patterns of a ARIMA(p,d,0) or ARIMA(0,d,q) from Forecasting Principles by Robert Hydman.

If the data are from an ARIMA(p,d,0) or ARIMA(0,d,q) model, then the ACF and PACF plots can be helpful in determining the value of p or q. If p and q are both positive, then the plots do not help in finding suitable values of p and q.

The data may follow an ARIMA(p,d,0) model if the ACF and PACF plots of the differenced data show the following patterns:

- the ACF is exponentially decaying or sinusoidal;
- there is a significant spike at lag p in the PACF, but none beyond lag p.

As the model exhibits both positive p and q variables, there are better methods to find the optimal values.

As an alternative we can use **AIC** to select the appropriate values.

The fable packages uses AIC to determine the appropriate p and q variables. The better the fit of the model is, the lower the AIC values is.

Maximum Likelihood Estimation to find the best parameter once the p,d,q variable has been found.

ARIMA ()

The ARIMA function provided by has an auto ARIMA function. Based on the documentation, these are the steps taken by the function.

1. Find the optimal number of difference through the KPSS test
2. Find the p and q values once the d has been chosen using a stepwise search. The best model has the smallest AICc value.It will also add or remove a constant to see if it reduces the AICc. This process is repeated until the AIC value no longer decreases.

As we know our model contains seasonality, we will also be using a Seasonal ARIMA model or SARIMA.

A seasonal ARIMA model has 2 parts the non seasonal part of the model and the seasonal part.

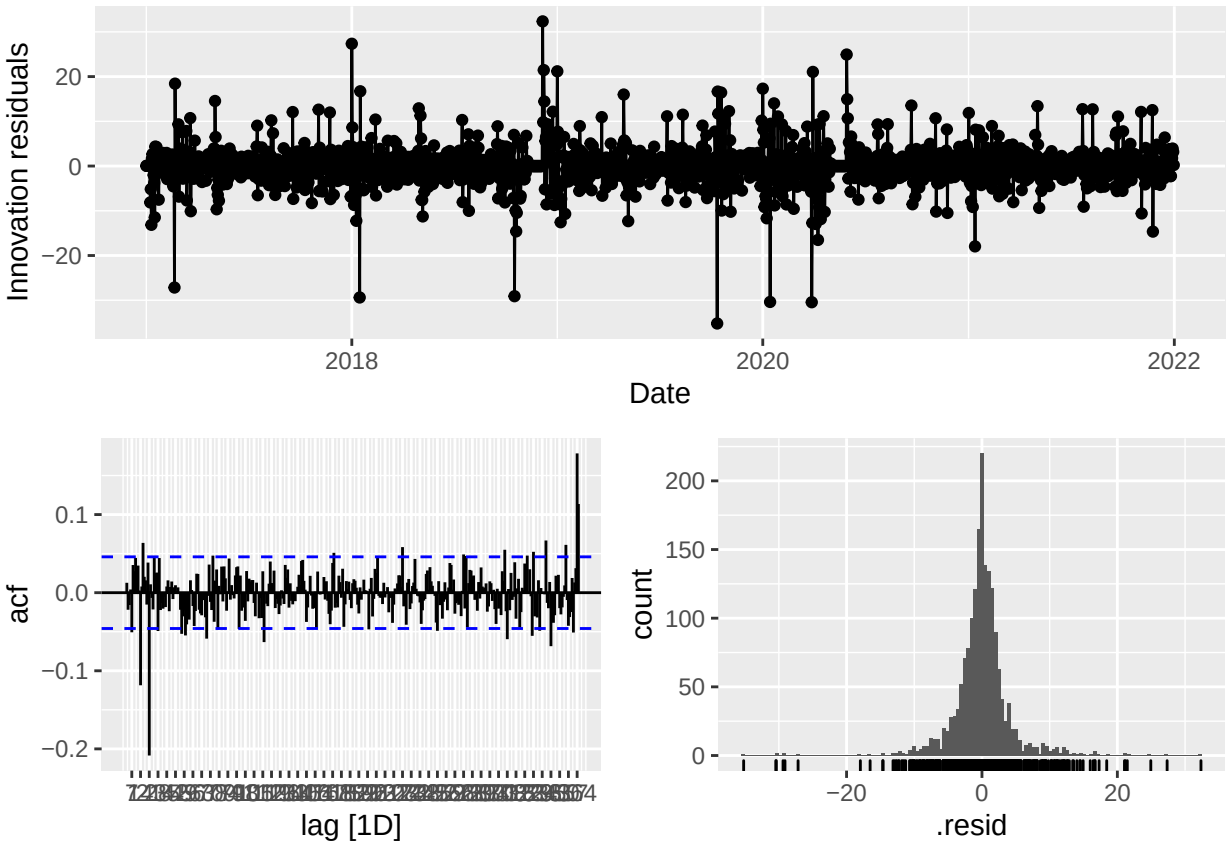
Fitting the model

```
## # A mable: 1 x 2
## # Key:     Model name [1]
##   `Model name`                  Orders
##   <chr>                        <model>
## 1 auto         <ARIMA(2,0,2)(2,1,0)[7]>
```

```
glance(fit) %>% arrange(AICc) %>% select(.model:BIC)
```

```
## # A tibble: 1 x 6
##   .model sigma2 log_lik    AIC   AICc    BIC
##   <chr>   <dbl>   <dbl>  <dbl>  <dbl>  <dbl>
## 1 auto     18.8  -5250. 10513. 10513. 10552.
```

```
fit %>% select(auto) %>% gg_tsresiduals(lag=365)
```



It is good to check and see if the residuals of our model look like white noise or not. If it does not, this means there are still ways to optimize our model

```
## # A tibble: 1 x 3
##   .model lb_stat lb_pvalue
##   <chr>    <dbl>     <dbl>
## 1 auto      183.  2.11e-15
```

Based on the ljung_box test, the value is greater than 0.1 and indeed white noise.

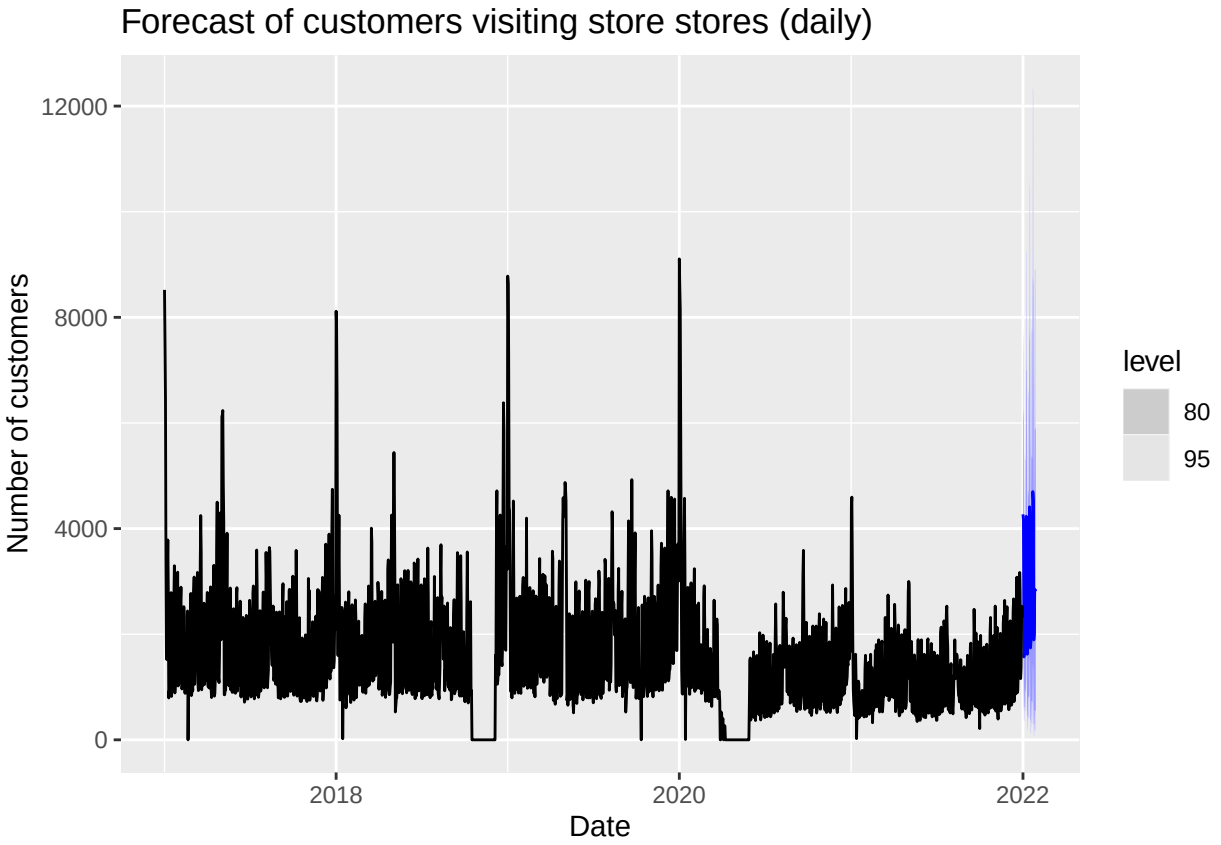
Forecast values

The exact forecast values from 2022/1/1 ~ 2022/1/28 can be seen through the .rmd file attached in the email.

```
## # A fable: 28 x 4 [1D]
## # Key:     .model [1]
##    .model Date            Traffic .mean
##    <chr>  <date>           <dist> <dbl>
##  1 auto   2022-01-01 t(N(40, 19)) 4270.
##  2 auto   2022-01-02 t(N(40, 27)) 4165.
##  3 auto   2022-01-03 t(N(28, 31)) 1569.
##  4 auto   2022-01-04 t(N(28, 34)) 1655.
##  5 auto   2022-01-05 t(N(33, 36)) 2420.
##  6 auto   2022-01-06 t(N(32, 37)) 2302.
##  7 auto   2022-01-07 t(N(32, 38)) 2275.
##  8 auto   2022-01-08 t(N(40, 46)) 4234.
##  9 auto   2022-01-09 t(N(39, 50)) 4140.
## 10 auto   2022-01-10 t(N(28, 52)) 1620.
```

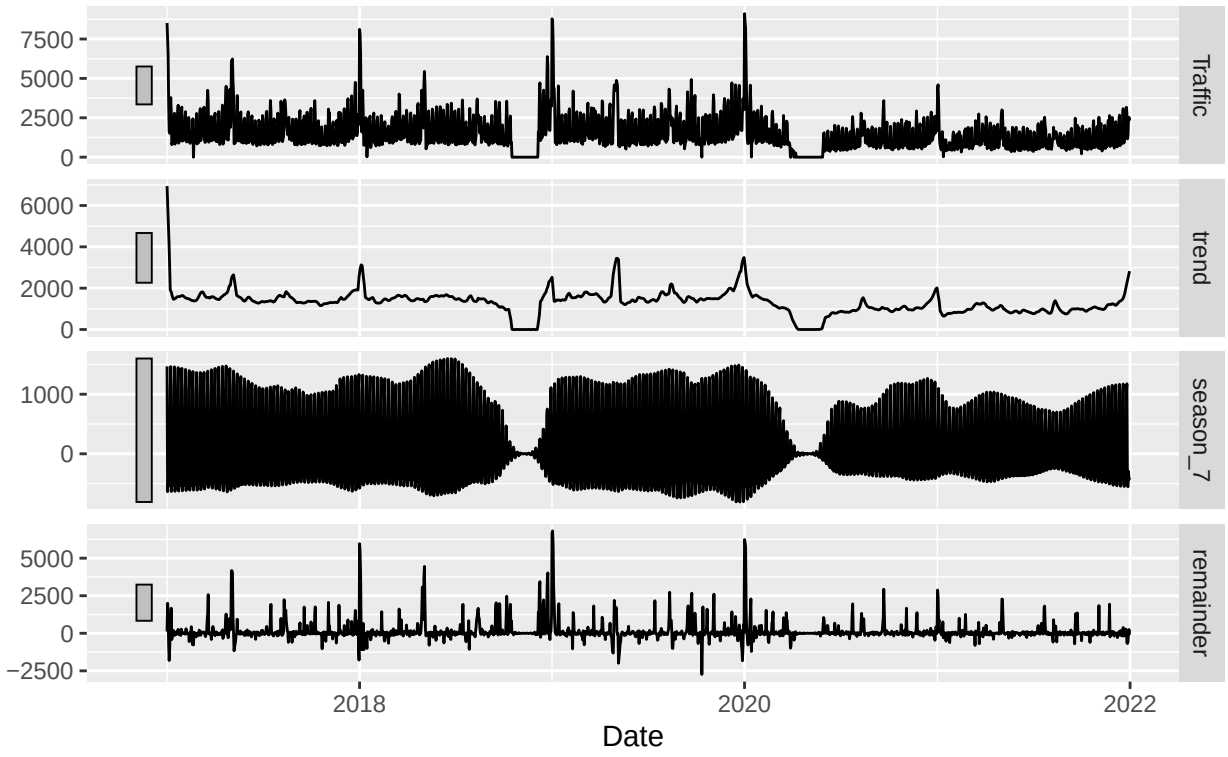
```
## # ... with 18 more rows
```

If our goal is to maximize sales we can use the hilo() function to see the upper values. Similarly if our goal is to reduce cost we can use the lower values.



As mentioned before there were several outliers.

```
decomp_traffic <- fill_traffic_df %>% model(stl = STL(Traffic ~ season(period = 7), robust = TRUE)) %>%

decomp_traffic %>% autoplot()
```




```
outliers <- decomp_traffic %>%
  filter(
    remainder < quantile(remainder, 0.25) - 3*IQR(remainder) |
    remainder > quantile(remainder, 0.75) + 3*IQR(remainder)
  )

outliers
```

```
## # A dable: 254 x 7 [1D]
## # Key:     .model [1]
## # :        Traffic = trend + season_7 + remainder
##    .model Date       Traffic trend season_7 remainder season_adjust
##    <chr>  <date>       <dbl> <dbl>    <dbl>     <dbl>         <dbl>
##  1 stl    2017-01-02    7546 6183.    -635.     1998.         8181.
##  2 stl    2017-01-03    6461 5428.    -499.     1532.         6960.
##  3 stl    2017-01-04    3401 4703.    -515.     -787.         3916.
##  4 stl    2017-01-05    1634 3978.    -535.    -1810.         2169.
##  5 stl    2017-01-06    1520 2949.    -368.    -1061.         1888.
##  6 stl    2017-01-09    2842 1809.    -632.     1665.         3474.
##  7 stl    2017-02-12    2321 1476.    1396.     -551.          925.
##  8 stl    2017-02-18    1689 1404.    1034.     -749.          655.
##  9 stl    2017-02-20       0 1386.    -610.     -776.          610.
## 10 stl    2017-03-18    3334 1550.     967.      817.         2367.
## # ... with 244 more rows
```

```
missing_cust <- fill_traffic_df %>%

  anti_join(outliers) %>%
```

```
  fill_gaps()
```

```
## Joining, by = c("Date", "Traffic")
```

```
traffic_clean_fill <- missing_cust %>%
  # Fit ARIMA model to the data containing missing values
  model(ARIMA(Traffic)) %>%
  # Estimate Trips for all periods
  interpolate(missing_cust)
traffic_clean_fill %>%
  # Only show outlying periods
  right_join(outliers %>% select(-Traffic))
```

```
## Joining, by = "Date"
```

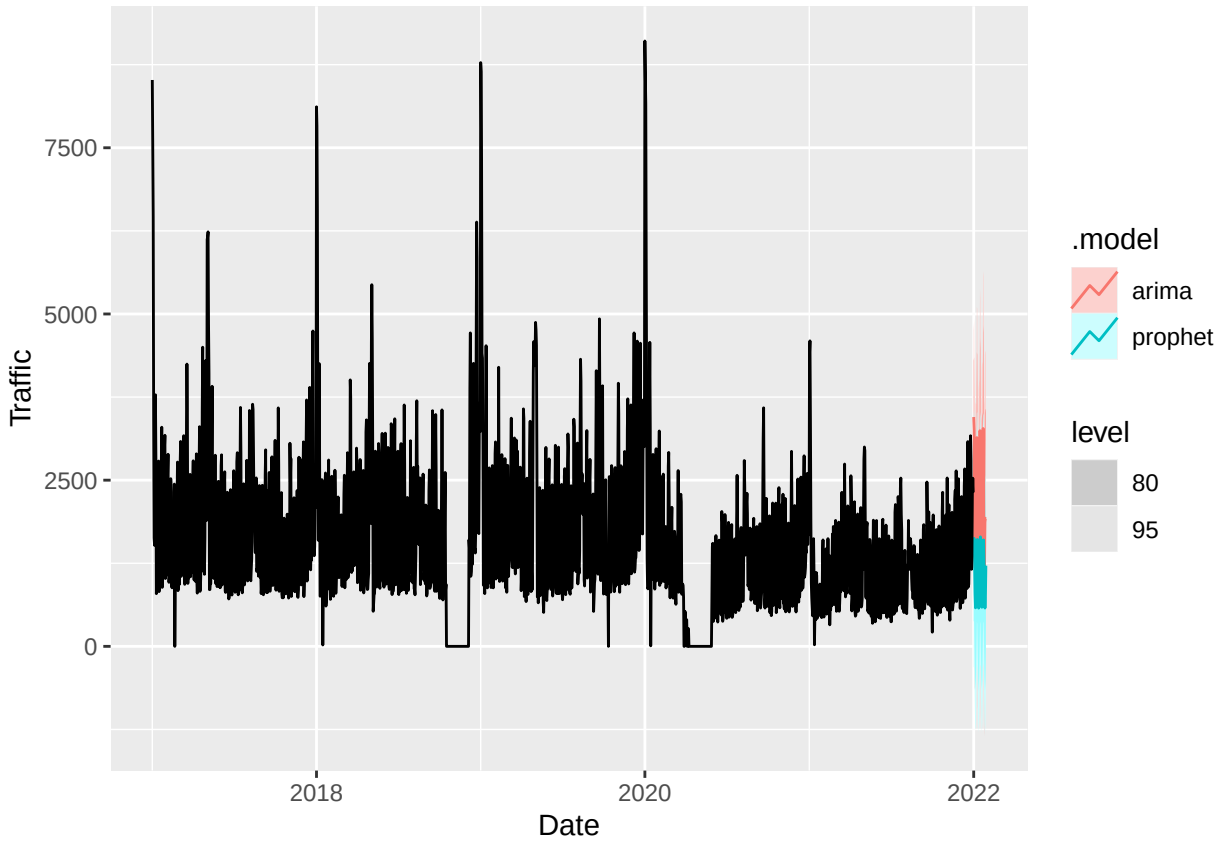
```
## # A tsibble: 254 x 7 [1D]
##    Date       Traffic .model trend season_7 remainder season_adjust
##    <date>       <dbl> <chr>  <dbl>    <dbl>     <dbl>         <dbl>
##  1 2017-01-02   6863. stl    6183.    -635.     1998.         8181.
##  2 2017-01-03   4966. stl    5428.    -499.     1532.         6960.
##  3 2017-01-04   4249. stl    4703.    -515.     -787.         3916.
##  4 2017-01-05   3723. stl    3978.    -535.    -1810.         2169.
##  5 2017-01-06   3136. stl    2949.    -368.    -1061.         1888.
##  6 2017-01-09   2048. stl    1809.    -632.     1665.         3474.
##  7 2017-02-12   2853. stl    1476.    1396.     -551.          925.
##  8 2017-02-18   2242. stl    1404.    1034.     -749.          655.
##  9 2017-02-20    679. stl    1386.    -610.     -776.          610.
## 10 2017-03-18   2592. stl    1550.     967.      817.         2367.
## # ... with 244 more rows
```

Here we can identify the outliers and estimate appropriate values using ARIMA

Alternative models

There are several alternatives that can be consider such as Facebook Prophet

```
# Prophet

fit <- fill_traffic_df %>%
model( arima = ARIMA(Traffic),
prophet = prophet(Traffic ~ season(period=7,order = 1, type = "multiplicative"))

)

fc <- fit %>% forecast(h = 28)

fc %>% autoplot(fill_traffic_df)
```



Alternative models such as prophet can also be used to check the results of our forecast. As we can see, Prophet has a more conservative forecast prediction compared to ARIMA

Due to time constraints, this was not explored but Fourier terms can greatly benefit our model as we are dealing with complex seasonality, both weekly and annual seasonality is present.

Conclusion

There are several key business insights that can be made from this analysis aside from the forecasting results only.

1. The peak days of the week are Friday, Saturday, Sunday
2. The peak months are January and December and lowest at October and November.

With the assumption that there is high correlation between number of sales rep present in a store with sales conversion, We can increase the number of sales rep during the key periods listed above, while reduce during low periods to reduce cost.

We have also see an upward trend as COVID has slowly subsided to pre-pandemic levels. This means that the general direction of store is to hire more sales rep as the number of people visiting stores has been steadily increasing again recently.

Moreover, we can also anticipate rise is Traffic due to successful campaigns as observed in the Micahel B Jordan fall campaign 2019. We can also evaluate the success of campaigns by observing if there is a surge. To so so we can use Dynamic regression models + lagged predictors. For the Michael B Jordan campaign in particular, we can truly evaluate it success by filtering sales by male products pre and post campaign to see how it affected the target audience rather than general customer.