Restaurant Visitor Forecasting

Executive Summary

Running a thriving local restaurant isn't always as charming as first impressions appear. There are often all sorts of unexpected troubles popping up that could hurt business.

One common predicament is that restaurants need to know how many customers to expect each day to effectively purchase ingredients and schedule staff members. This forecast isn't easy to make because many unpredictable factors affect restaurant attendance, like weather and local competition. It's even harder for newer restaurants with little historical data.

In this dataset, we are going use reservation and visitation data to predict the total number of visitors to a restaurant for future dates. This information will help restaurants be much more efficient and allow them to focus on creating an enjoyable dining experience for their customers.

Data Description

The data comes in the shape of 8 relational files which are derived from two separate Japanese websites that collect user information: "Hot Pepper Gourmet (hpg): similar to Yelp" (search and reserve) and "AirREGI / Restaurant Board (air): similar to Square" (reservation control and cash register). The training data is based on the time range of Jan 2016 - Feb 2017, while the test set includes the March 2017 and most part of the April 2017. The training set omits days where the restaurants were closed."

Those are the individual files:

`air_visit_data.csv`: historical visit data for the air restaurants. This is essentially the main training data set.

`air_reserve.csv / hpg_reserve.csv`: reservations made through the air / hpg systems.

`air_store_info.csv / hpg_store_info.csv`: details about the air / hpg restaurants including genre and location.

`store_id_relation.csv`: connects the air and hpg ids

`date_info.csv`: essentially flags the Japanese holidays.

`sample_submission.csv`: serves as the test set. The id is formed by combining the air id with the visit date.

```
pacman::p_load(dplyr,ggplot2, lubridate,stringr, tidyr, forcats, ggExtra, tibble, forecast,data.table, 
```

Reading Data

```
air_visits <- read.csv2('air_visit_data.csv',sep=",")

air_reserve <- read.csv2('air_reserve.csv',sep=",")
hpg_reserve <- read.csv2('hpg_reserve.csv',sep=",")

air_store <- read.csv2('air_store_info.csv',sep=",")
hpg_store <- read.csv2('hpg_store_info.csv',sep=",")

holidays <- read.csv2('date_info.csv',sep=",")
store_ids <- read.csv2('store_id_relation.csv',sep=",")
test <- read.csv2('test_data.csv',sep=",")
```

Summary statistics

```
summary(air_visits)
```

```
##                air_store_id         visit_date        visitors
##  air_5c817ef28f236bdf:   477   2017-03-17:   799   Min.   :  1.00
##  air_36bcf77d3382d36e:   476   2017-03-03:   791   1st Qu.:  9.00
##  air_a083834e7ffe187e:   476   2017-04-14:   791   Median : 17.00
##  air_d97dabf7aae60da5:   476   2017-03-24:   789   Mean   : 20.97
##  air_232dcee6f7c51d37:   475   2017-03-31:   786   3rd Qu.: 29.00
##  air_60a7057184ec7ec7:   475   2017-04-21:   784   Max.   :877.00
##  (Other)             :249253   (Other)   :247368
```

```
glimpse(air_visits)
```

```
## Observations: 252,108
## Variables: 3
## $ air_store_id <fct> air_ba937bf13d40fb24, air_ba937bf13d40fb24, air_b...
## $ visit_date   <fct> 2016-01-13, 2016-01-14, 2016-01-15, 2016-01-16, 2...
## $ visitors     <int> 25, 32, 29, 22, 6, 9, 31, 21, 18, 26, 21, 11, 24,...
```

```
air_visits %>% distinct(air_store_id) %>% nrow()
```

```
## [1] 829
```

We find that this file contains the visitors numbers for each visit_date and air_store_id. The date feature should be transformed into a time-series format. There are 829 different stores.

```
summary(air_reserve)
```

```
##                air_store_id               visit_datetime
##  air_8093d0b565e9dbdf: 2263   2016-12-24 19:00:00:  255
##  air_e55abd740f93ecc4: 1903   2016-12-29 18:00:00:  230
##  air_0a74a5408a0b8642: 1831   2016-12-17 18:00:00:  229
##  air_cf5ab75a0afb8af9: 1758   2016-12-24 18:00:00:  227
##  air_6d65542aa43b598b: 1436   2017-02-04 19:00:00:  226
##  air_de692863bb2dd758: 1355   2016-12-10 18:00:00:  219
##  (Other)             :81832   (Other)            :90992
##             reserve_datetime reserve_visitors
##  2016-11-24 18:00:00:  106   Min.   :  1.000
##  2016-12-24 17:00:00:  103   1st Qu.:  2.000
##  2016-12-08 18:00:00:   95   Median :  3.000
##  2016-12-06 19:00:00:   92   Mean   :  4.482
##  2016-12-07 17:00:00:   91   3rd Qu.:  5.000
##  2017-03-09 18:00:00:   90   Max.   :100.000
##  (Other)            :91801
```

```
glimpse(air_reserve)
```

```
## Observations: 92,378
## Variables: 4
## $ air_store_id     <fct> air_877f79706adbfb06, air_db4b38ebe7a7ceff, a...
## $ visit_datetime   <fct> 2016-01-01 19:00:00, 2016-01-01 19:00:00, 201...
## $ reserve_datetime <fct> 2016-01-01 16:00:00, 2016-01-01 19:00:00, 201...
## $ reserve_visitors <int> 1, 3, 6, 2, 5, 2, 4, 2, 2, 2, 3, 3, 2, 6, 7, ...
```

```
air_reserve %>% distinct(air_store_id) %>% nrow()
```

```
## [1] 314
```

We find that the air reservations include the date and time of the reservation, as well as those of the visit. We have reservation numbers for 314 air stores.

```
summary(hpg_reserve)
```

```
##                hpg_store_id                visit_datetime
##  hpg_2afd5b187409eeb4:   1155   2016-12-16 19:00:00:  10528
##  hpg_011e799ba201ad2e:    822   2016-12-22 19:00:00:  10475
##  hpg_9b20c78a9b8179d9:    778   2016-12-17 19:00:00:   7524
##  hpg_527c60506b80ac72:    740   2016-12-09 19:00:00:   7103
##  hpg_3f9e56ac6f9435c7:    729   2016-12-24 19:00:00:   6884
##  hpg_4ca09101fa3a220c:    712   2016-12-17 18:00:00:   6707
##  (Other)             :1995384   (Other)            :1951099
##             reserve_datetime   reserve_visitors
##  2016-12-12 21:00:00:    907   Min.   :  1.000
##  2016-12-13 21:00:00:    889   1st Qu.:  2.000
##  2016-12-14 21:00:00:    865   Median :  3.000
##  2016-12-07 21:00:00:    853   Mean   :  5.074
##  2016-12-06 21:00:00:    840   3rd Qu.:  6.000
##  2016-12-05 21:00:00:    837   Max.   :100.000
##  (Other)            :1995129
```

```
glimpse(hpg_reserve)
```

```
## Observations: 2,000,320
## Variables: 4
## $ hpg_store_id     <fct> hpg_c63f6f42e088e50f, hpg_dac72789163a3f47, h...
## $ visit_datetime   <fct> 2016-01-01 11:00:00, 2016-01-01 13:00:00, 201...
## $ reserve_datetime <fct> 2016-01-01 09:00:00, 2016-01-01 06:00:00, 201...
## $ reserve_visitors <int> 1, 3, 2, 5, 13, 2, 2, 2, 2, 6, 2, 2, 2, 2, 5,...
```

```
hpg_reserve %>% distinct(hpg_store_id) %>% nrow()
```

```
## [1] 13325
```

The hpg reservations file follows the same structure as the corresponding air file. We have reservation numbers for 13325 hpg stores.

```
summary(air_store)
```

```
##                air_store_id        air_genre_name
##  air_00a91d42b08b08d9:  1   Izakaya       :197
##  air_0164b9927d20bcc3:  1   Cafe/Sweets   :181
##  air_0241aa3964b7f861:  1   Dining bar    :108
##  air_0328696196e46f18:  1   Italian/French:102
##  air_034a3d5b40d5b1b1:  1   Bar/Cocktail  : 79
##  air_036d4f1ee7285390:  1   Japanese food : 63
##  (Other)             :823   (Other)       : 99
##                           air_area_name       latitude
##  Fukuoka-ken Fukuoka-shi DaimyÅ\215  : 64   33.5892157: 62
##  TÅ\215kyÅ\215-to Shibuya-ku Shibuya    : 58   35.6617773: 58
##  TÅ\215kyÅ\215-to Minato-ku ShibakÅ\215en  : 51   35.6580681: 51
##  TÅ\215kyÅ\215-to Shinjuku-ku KabukichÅ\215: 39   35.6938401: 39
##  TÅ\215kyÅ\215-to Setagaya-ku Setagaya  : 30   35.6706505: 29
##  TÅ\215kyÅ\215-to ChÅ«Å\215-ku Tsukiji     : 29   35.6465721: 28
##  (Other)                          :558   (Other)   :562
```

```
##        longitude
##  130.3928134: 62
##  139.7040506: 58
##  139.7515992: 51
##  139.7035494: 39
##  139.7718614: 29
##  139.6532473: 28
##  (Other)    :562
```

```
glimpse(air_store)
```

```
## Observations: 829
## Variables: 5
## $ air_store_id   <fct> air_0f0cdeee6c9bf3d7, air_7cc17a324ae5c7dc, air...
## $ air_genre_name <fct> Italian/French, Italian/French, Italian/French,...
## $ air_area_name  <fct> HyÅ go-ken KÅ be-shi KumoidÅ ri, HyÅ go-ken KÅ ...
## $ latitude       <fct> 34.6951242, 34.6951242, 34.6951242, 34.6951242,...
## $ longitude      <fct> 135.1978525, 135.1978525, 135.1978525, 135.1978...
```

```
air_store %>% distinct(air_store_id) %>% nrow()
```

```
## [1] 829
```

We find that the air_store info includes the name of the particular cuisine along with the name of the area.

```
summary(hpg_store)
```

```
##                hpg_store_id                hpg_genre_name
##  hpg_001ce40a1f873e4f:   1   Japanese style       :1750
##  hpg_001f8de5120ce935:   1   International cuisine: 700
##  hpg_0034f74a25be3cbe:   1   Creation            : 410
##  hpg_00394a75a35c427c:   1   Seafood             : 339
##  hpg_004d556cb3f7995f:   1   Grilled meat        : 325
##  hpg_0050eb4311ceeae4:   1   Italian             : 249
##  (Other)             :4684   (Other)             : 917
##                                hpg_area_name              latitude
##  TÅ\215kyÅ\215-to Shinjuku-ku None          : 257   35.6913840261435: 257
##  TÅ\215kyÅ\215-to ChÅ«Å\215-ku Ginza           : 198   35.6686002312873: 198
##  HyÅ\215go-ken KÅ\215be-shi None            : 163   34.6921090191178: 160
##  Å saka-fu Å saka-shi Shinsaibashisuji: 150   34.6695141138059: 150
##  Osaka Prefecture Osaka None          : 145   34.7015189521487: 145
##  Hiroshima-ken Hiroshima-shi HondÅ\215ri : 135   34.3921061951814: 135
##  (Other)                              :3642   (Other)         :3645
##             longitude
##  139.701256018243: 257
##  139.763042710922: 198
##  135.19169838649 : 160
##  135.501425073088: 150
##  135.498858596496: 145
##  132.461913549062: 135
##  (Other)         :3645
```

```
glimpse(hpg_store)
```

```
## Observations: 4,690
## Variables: 5
## $ hpg_store_id   <fct> hpg_6622b62385aec8bf, hpg_e9e068dd49c5fa00, hpg...
```

```
## $ hpg_genre_name <fct> Japanese style, Japanese style, Japanese style,...
## $ hpg_area_name  <fct> TÅ kyÅ -to Setagaya-ku TaishidÅ , TÅ kyÅ -to Se...
## $ latitude       <fct> 35.6436746642265, 35.6436746642265, 35.64367466...
## $ longitude      <fct> 139.668220854814, 139.668220854814, 139.6682208...
```

```
hpg_store %>% distinct(hpg_store_id) %>% nrow()
```

```
## [1] 4690
```

Again, the hpg_store info follows the same structure as the air info. Here the genre_name includes the word style. It's worth checking whether the same is true for the air data or whether it just refers to the specific "Japanese style". There are 4690 different hpg_store_ids, which are significantly fewer than we have reservation data for.

```
summary(holidays)
```

```
##     calendar_date    day_of_week  holiday_flg
##  2016-01-01:  1   Friday   :74   Min.   :0.0000
##  2016-01-02:  1   Monday   :74   1st Qu.:0.0000
##  2016-01-03:  1   Saturday :74   Median :0.0000
##  2016-01-04:  1   Sunday   :74   Mean   :0.0677
##  2016-01-05:  1   Thursday :73   3rd Qu.:0.0000
##  2016-01-06:  1   Tuesday  :74   Max.   :1.0000
##  (Other)   :511   Wednesday:74
```

```
glimpse(holidays)
```

```
## Observations: 517
## Variables: 3
## $ calendar_date <fct> 2016-01-01, 2016-01-02, 2016-01-03, 2016-01-04, ...
## $ day_of_week   <fct> Friday, Saturday, Sunday, Monday, Tuesday, Wedne...
## $ holiday_flg   <int> 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, ...
```

We called the date_info file holidays, because that's essentially the information it contains. Holidays are encoded as binary flags in integer format. This should become a logical binary feature for exploration purposes.

```
summary(store_ids)
```

```
##                air_store_id               hpg_store_id
##  air_00a91d42b08b08d9:  1   hpg_0183ec352e38903c:  1
##  air_04cae7c1bc9b2a0b:  1   hpg_03795a62e1c5f1c3:  1
##  air_066f0221b8a4d533:  1   hpg_03946d6b6b1e0332:  1
##  air_082908692355165e:  1   hpg_053676b196ae1e1e:  1
##  air_0867f7bebad6a649:  1   hpg_05fada27f04e4383:  1
##  air_08cb3c4ee6cd6a22:  1   hpg_08def9f764fcca4c:  1
##  (Other)             :144   (Other)             :144
```

```
glimpse(store_ids)
```

```
## Observations: 150
## Variables: 2
## $ air_store_id <fct> air_63b13c56b7201bd9, air_a24bf50c3e90d583, air_c...
## $ hpg_store_id <fct> hpg_4bc649e72e2a239a, hpg_c34b496d0305a809, hpg_c...
```

This is a relational file that connects the air and hpg ids. There are only 150 pairs, which is less than 20% of all air stores.

```
summary(test)
```

```
##                                 id           visitors
##  air_00a91d42b08b08d9_2017-03-01:    1   Min.   :  1.0
##  air_00a91d42b08b08d9_2017-03-02:    1   1st Qu.:  9.0
##  air_00a91d42b08b08d9_2017-03-03:    1   Median : 18.0
##  air_00a91d42b08b08d9_2017-03-04:    1   Mean   : 21.9
##  air_00a91d42b08b08d9_2017-03-06:    1   3rd Qu.: 30.0
##  air_00a91d42b08b08d9_2017-03-07:    1   Max.   :877.0
##  (Other)                         :38056
```

```
glimpse(test)
```

```
## Observations: 38,062
## Variables: 2
## $ id       <fct> air_00a91d42b08b08d9_2017-03-01, air_00a91d42b08b08d9...
## $ visitors <int> 17, 33, 29, 18, 18, 31, 28, 38, 39, 9, 39, 28, 42, 39...
```

```
test %>% distinct(id) %>% nrow()
```

```
## [1] 38062
```

Missing Data

```
sum(is.na(air_visits))
```

```
## [1] 0
```

```
sum(is.na(air_reserve))
```

```
## [1] 0
```

```
sum(is.na(hpg_reserve))
```

```
## [1] 0
```

```
sum(is.na(air_store))
```

```
## [1] 0
```

```
sum(is.na(hpg_store))
```

```
## [1] 0
```

```
sum(is.na(holidays))
```

```
## [1] 0
```

```
sum(is.na(store_ids))
```

```
## [1] 0
```

```
sum(is.na(test))
```

```
## [1] 0
```

There are no missing values in our data.

Reformatting of Features

```
air_visits <- air_visits %>%
  mutate(visit_date = ymd(visit_date))
```

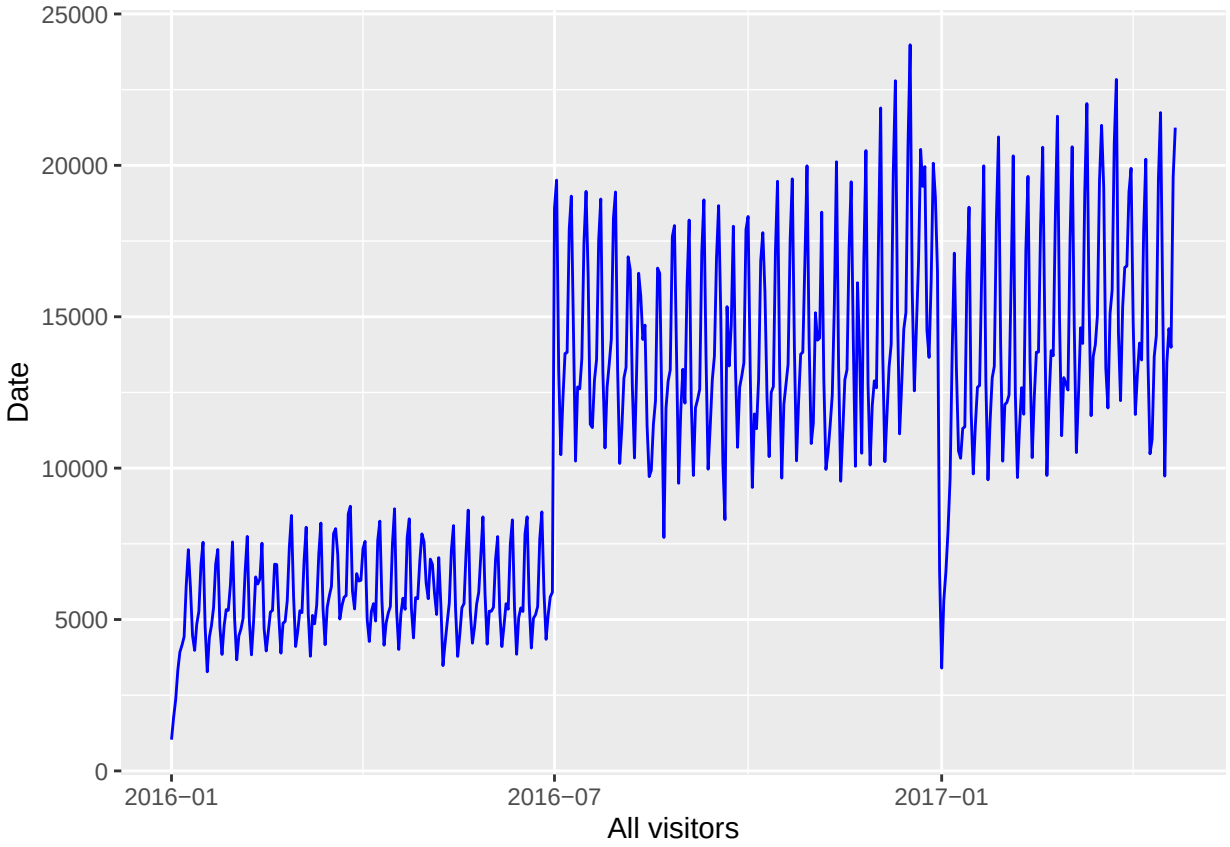
```
air_reserve <- air_reserve %>%
  mutate(visit_datetime = ymd_hms(visit_datetime),
         reserve_datetime = ymd_hms(reserve_datetime))

hpg_reserve <- hpg_reserve %>%
  mutate(visit_datetime = ymd_hms(visit_datetime),
         reserve_datetime = ymd_hms(reserve_datetime))

air_store <- air_store %>%
  mutate(air_genre_name = as.factor(air_genre_name),
         air_area_name = as.factor(air_area_name))

hpg_store <- hpg_store %>%
  mutate(hpg_genre_name = as.factor(hpg_genre_name),
         hpg_area_name = as.factor(hpg_area_name))

holidays <- holidays %>%
  mutate(holiday_flg = as.logical(holiday_flg),
         date = ymd(calendar_date),
         calendar_date = as.character(calendar_date))
```

We change the formatting of the date/time features and also reformat a few features to logical and factor variables for exploration purposes.

Exploratory Data Analysis

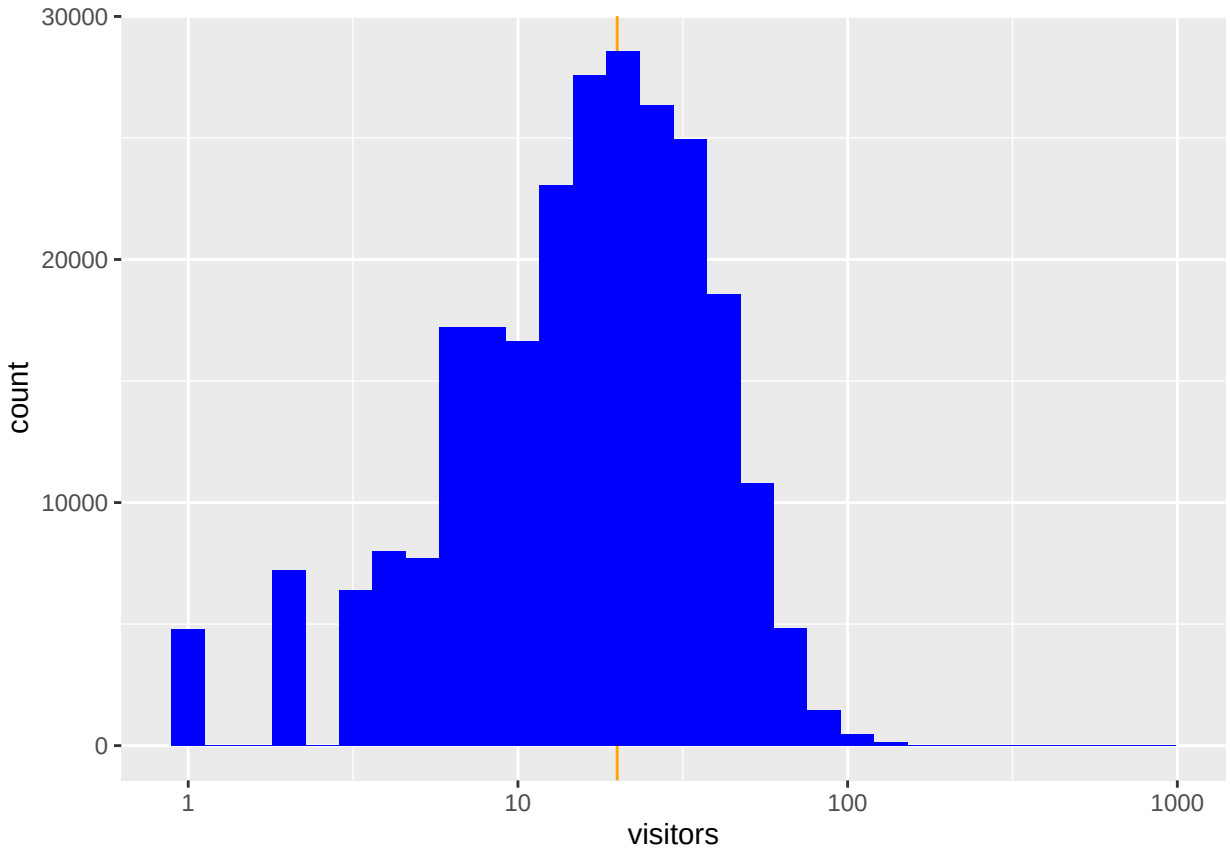
Here we have a first look at the distributions of the feature in our individual data files before combining them for a more detailed analysis. This initial visualization will be the foundation on which we build our analysis. We start with the number of visits to the air restaurants. Here we plot the total number of visitors per day over the full training time range.

```
air_visits %>%
  group_by(visit_date) %>%
  summarise(all_visitors = sum(visitors)) %>%
  ggplot(aes(visit_date,all_visitors,group=1)) +
  geom_line(col = "blue") +
  labs(x = "All visitors", y = "Date")
```



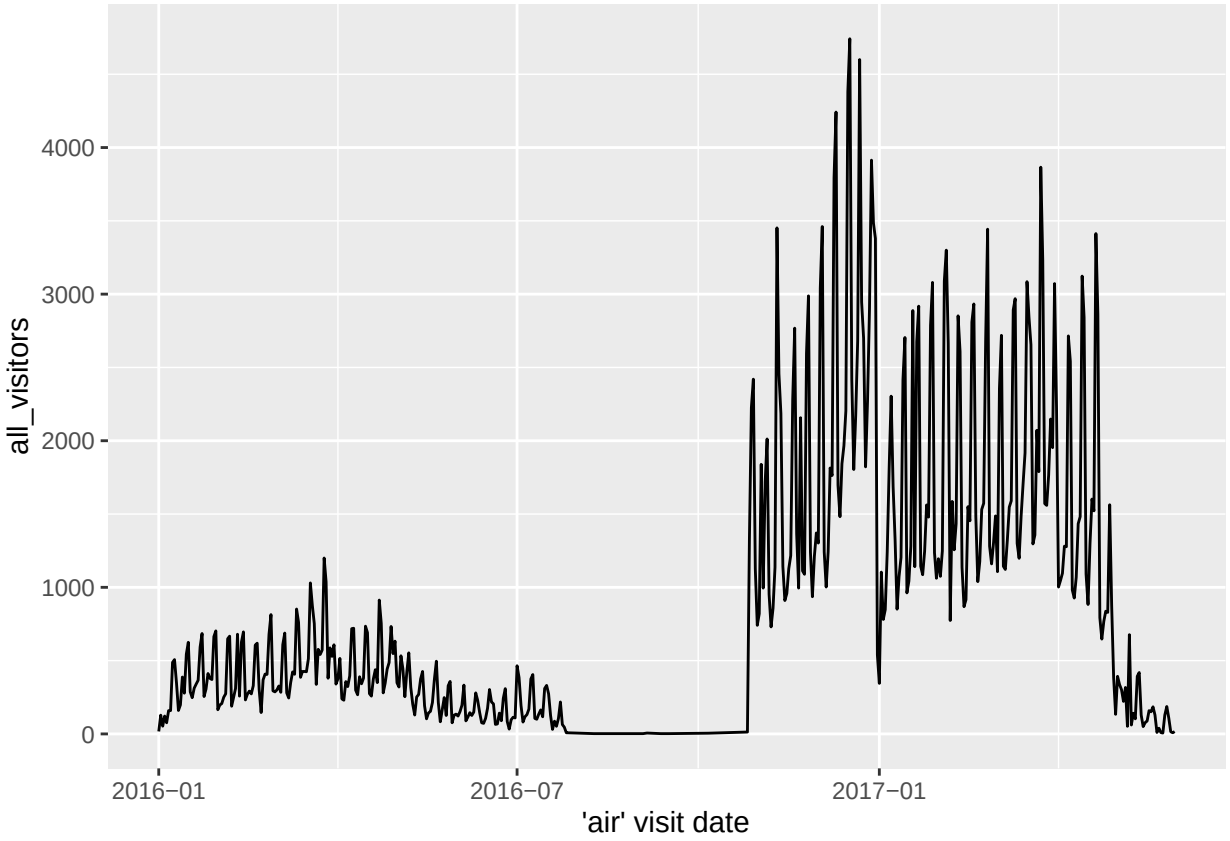
There is an interesting long-term step structure in the overall time series. This might be related to new restaurants being added to the data base. In addition, we already see a periodic pattern that most likely corresponds to a weekly cycle.

```
air_visits %>%
  ggplot(aes(visitors)) +
  geom_vline(xintercept = 20, color = "orange") +
  geom_histogram(fill = "blue", bins = 30) +
  scale_x_log10()
```



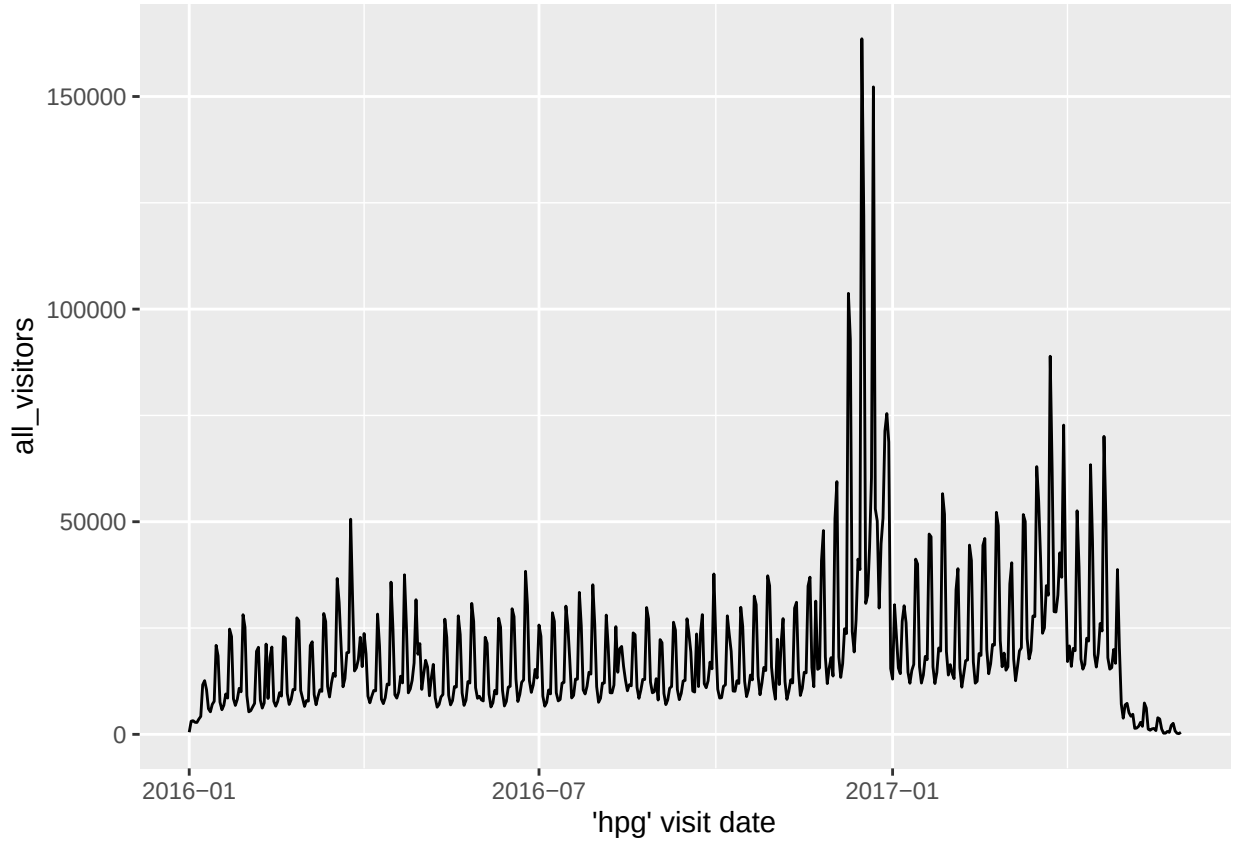
The number of guests per visit per restaurant per day peaks at around 20 (the orange line). The distribution extends up to 100 and, in rare cases, beyond.

```
foo <- air_reserve %>%
  mutate(reserve_date = date(reserve_datetime),
         reserve_hour = hour(reserve_datetime),
         reserve_wday = wday(reserve_datetime),
         visit_date = date(visit_datetime),
         visit_hour = hour(visit_datetime),
         visit_wday = wday(visit_datetime),
         diff_hour = time_length(visit_datetime - reserve_datetime, unit = "hour"),
         diff_day = time_length(visit_datetime - reserve_datetime, unit = "day")
  )

foo %>%
  group_by(visit_date) %>%
  summarise(all_visitors = sum(reserve_visitors)) %>%
  ggplot(aes(visit_date, all_visitors)) +
  geom_line() +
  labs(x = "'air' visit date")
```



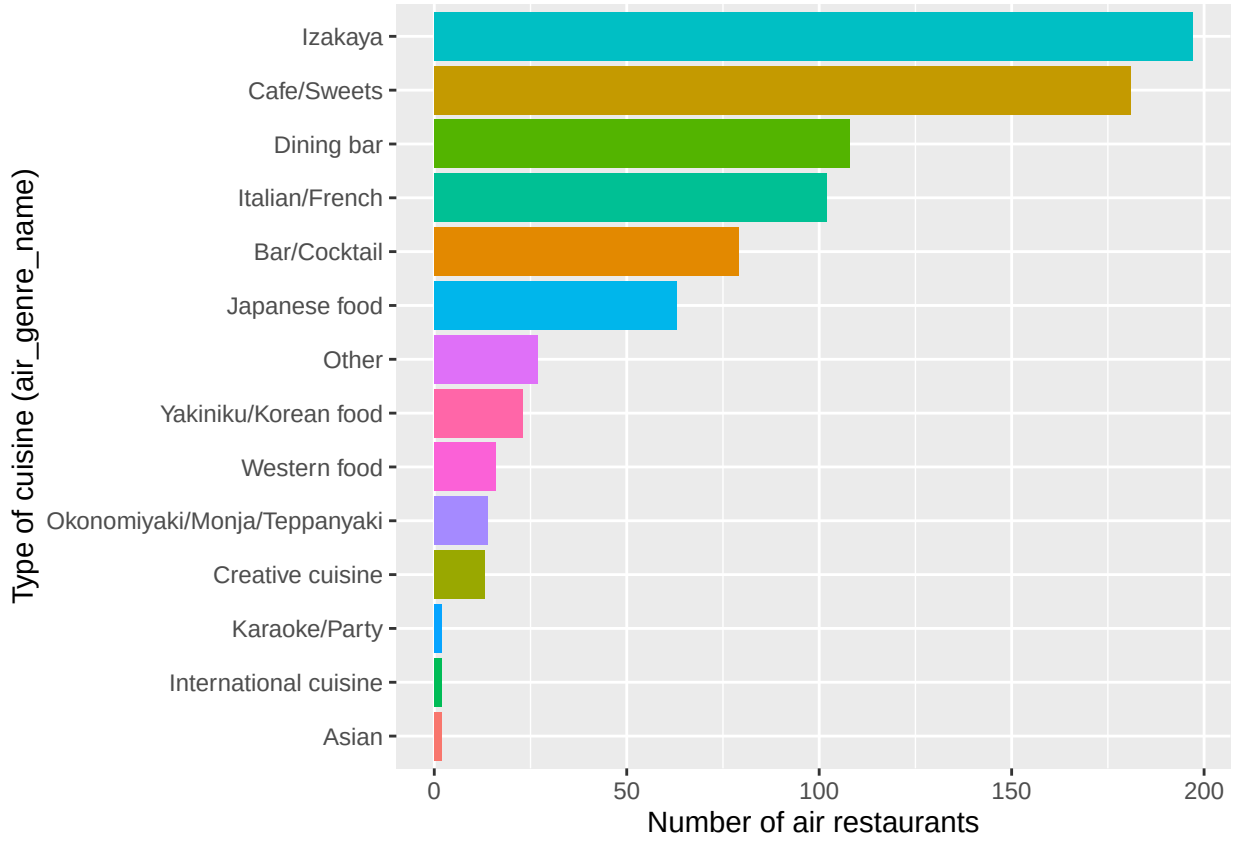
There were much fewer reservations made in 2016 through the air system; even none at all for a long stretch of time. The volume only increased during the end of that year. In 2017 the visitor numbers stayed strong. The artificial decline we see after the first quarter is most likely related to these reservations being at the end of the training time frame, which means that long-term reservations would not be part of this data set.

```
foo <- hpg_reserve %>%
  mutate(reserve_date = date(reserve_datetime),
         reserve_hour = hour(reserve_datetime),
         visit_date = date(visit_datetime),
         visit_hour = hour(visit_datetime),
         diff_hour = time_length(visit_datetime - reserve_datetime, unit = "hour"),
         diff_day = time_length(visit_datetime - reserve_datetime, unit = "day")
  )

foo %>%
  group_by(visit_date) %>%
  summarise(all_visitors = sum(reserve_visitors)) %>%
  ggplot(aes(visit_date, all_visitors)) +
  geom_line() +
  labs(x = "'hpg' visit date")
```



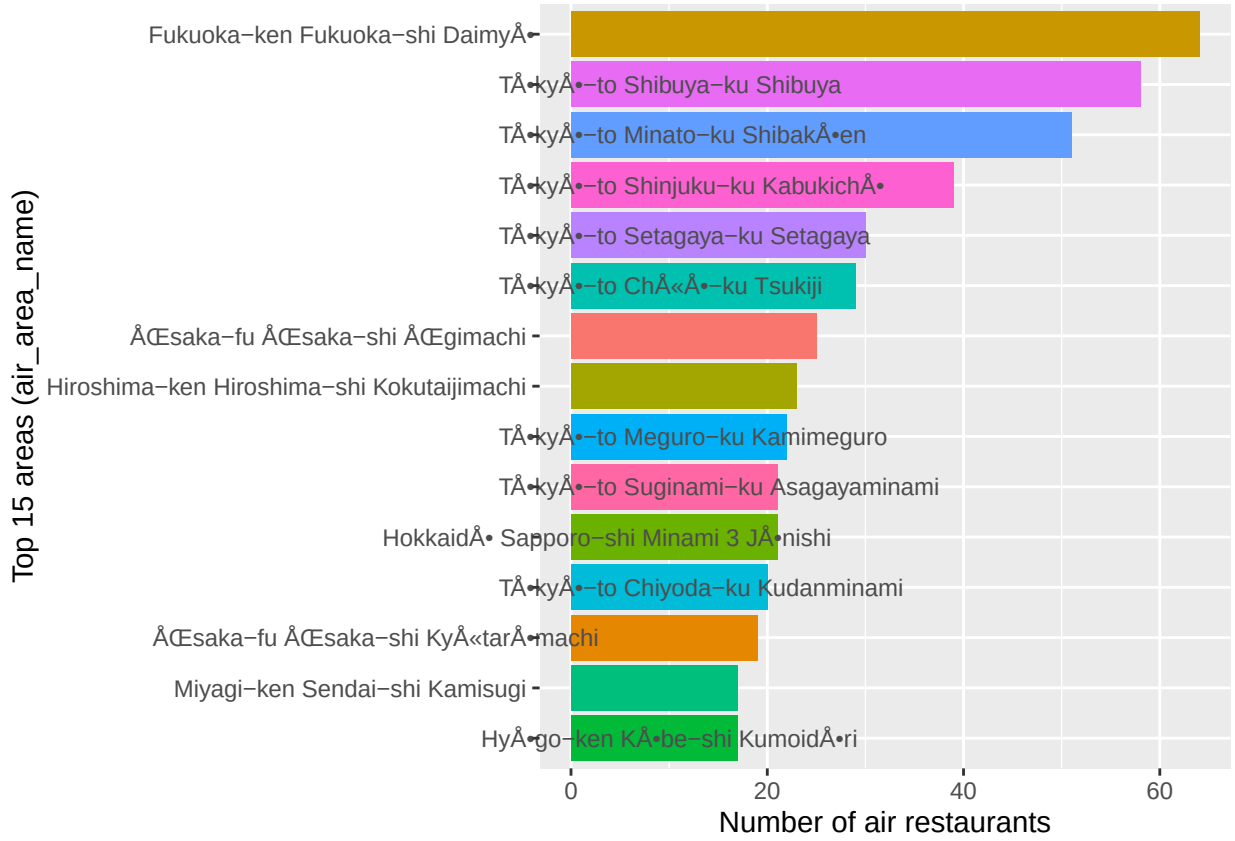
Here the visits after reservation follow a more orderly pattern, with a clear spike in Dec 2016. As above for the air data, we also see reservation visits dropping off as we get closer to the end of the time frame.

```
air_store %>%
  group_by(air_genre_name) %>%
  count() %>%
  ggplot(aes(reorder(air_genre_name, n, FUN = min), n, fill = air_genre_name)) +
  geom_col() +
  coord_flip() +
  theme(legend.position = "none") +
  labs(x = "Type of cuisine (air_genre_name)", y = "Number of air restaurants")
```



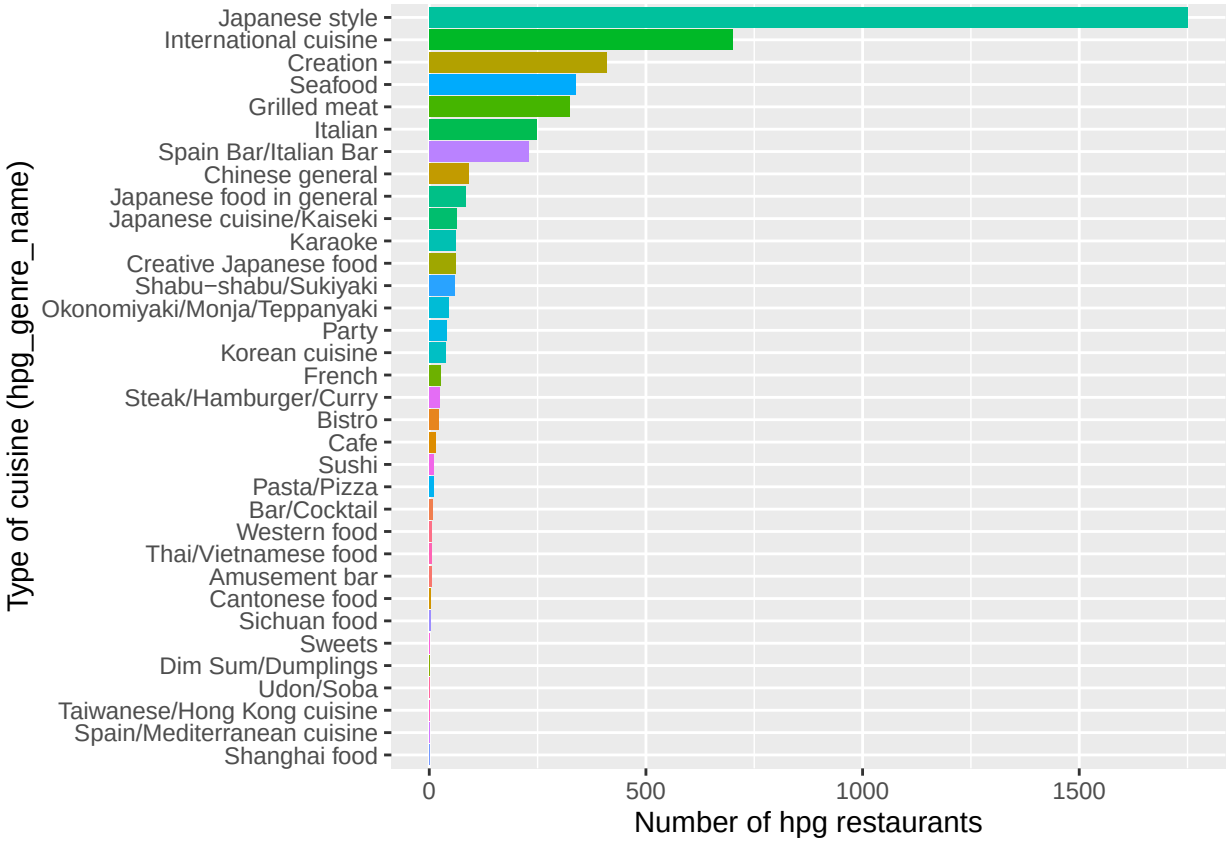
There are lots of Izakaya gastropubs in our data, followed by Cafe's. We don't have many Karaoke places in the air data set and also only a few that describe themselves as generically "International" or "Asian". I have to admit, I'm kind of intrigued by "creative cuisine".

```
air_store %>%
  group_by(air_area_name) %>%
  count() %>%
  ungroup() %>%
  top_n(15,n) %>%
  ggplot(aes(reorder(air_area_name, n, FUN = min) ,n, fill = air_area_name)) +
  geom_col() +
  theme(legend.position = "none") +
  coord_flip() +
  labs(x = "Top 15 areas (air_area_name)", y = "Number of air restaurants")
```



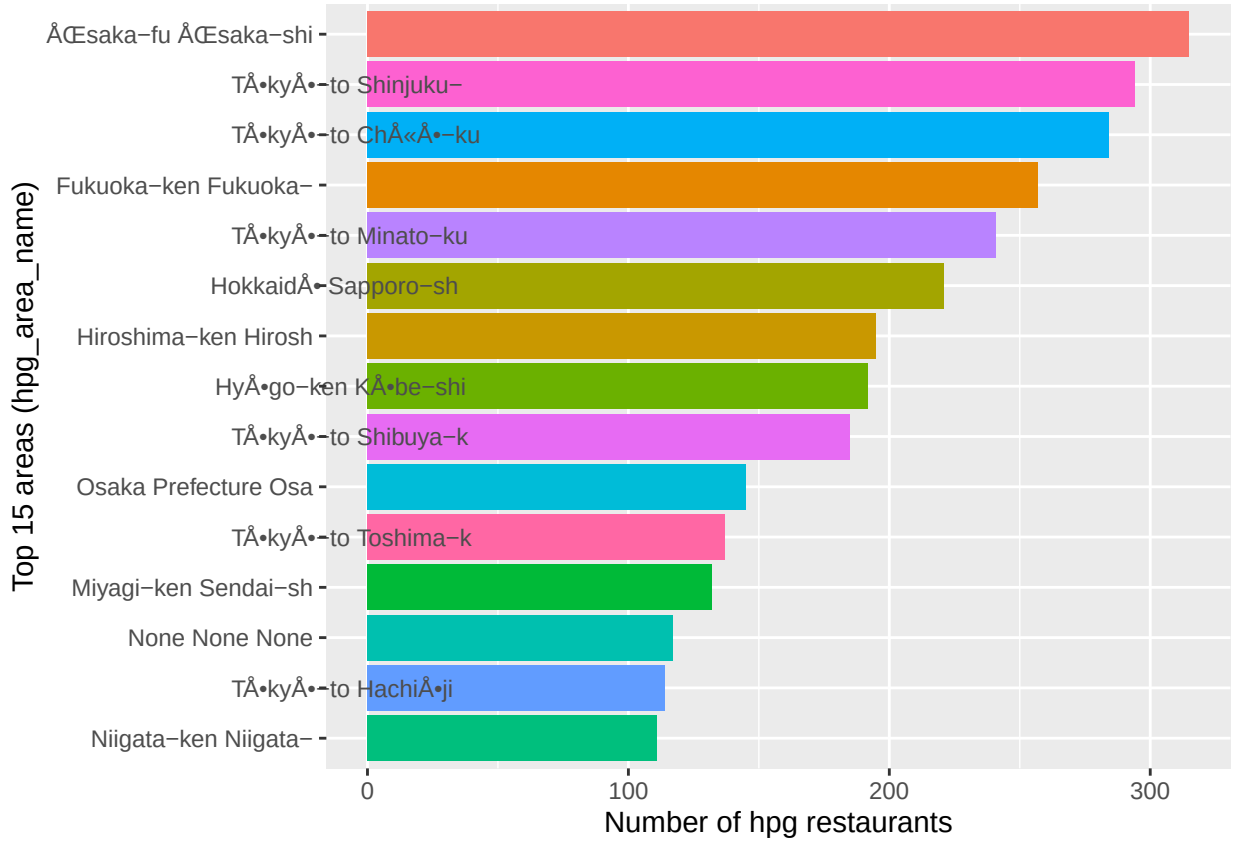
Fukuoka has the largest number of air restaurants per area, followed by many Tokyo areas.

```
hpg_store %>%
  group_by(hpg_genre_name) %>%
  count() %>%
  ggplot(aes(reorder(hpg_genre_name, n, FUN = min), n, fill = hpg_genre_name)) +
  geom_col() +
  coord_flip() +
  theme(legend.position = "none") +
  labs(x = "Type of cuisine (hpg_genre_name)", y = "Number of hpg restaurants")
```



The hpg description contains a larger variety of genres than in the air data. Here, "Japanese style" appears to contain many more places that are categorised more specifically in the air data. The same applies to "International cuisine".

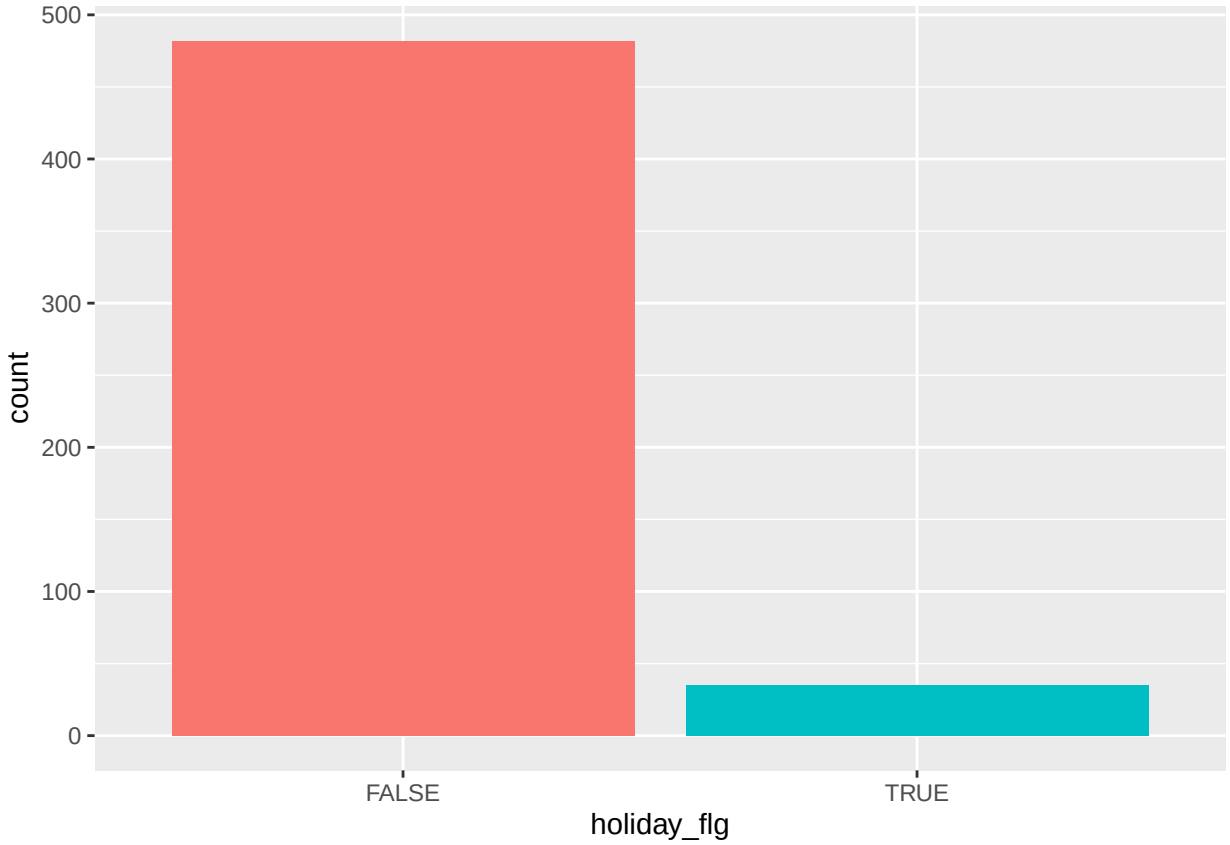
```
hpg_store %>%
  mutate(area = str_sub(hpg_area_name, 1, 20)) %>%
  group_by(area) %>%
  count() %>%
  ungroup() %>%
  top_n(15,n) %>%
  ggplot(aes(reorder(area, n, FUN = min) ,n, fill = area)) +
  geom_col() +
  theme(legend.position = "none") +
  coord_flip() +
  labs(x = "Top 15 areas (hpg_area_name)", y = "Number of hpg restaurants")
```



In the top 15 area we find again Tokyo and Osaka to be prominently present.

```
foo <- holidays %>%
  mutate(wday = wday(date))
```

```
foo %>%
  ggplot(aes(holiday_flg, fill = holiday_flg)) +
  geom_bar() +
  theme(legend.position = "none")
```



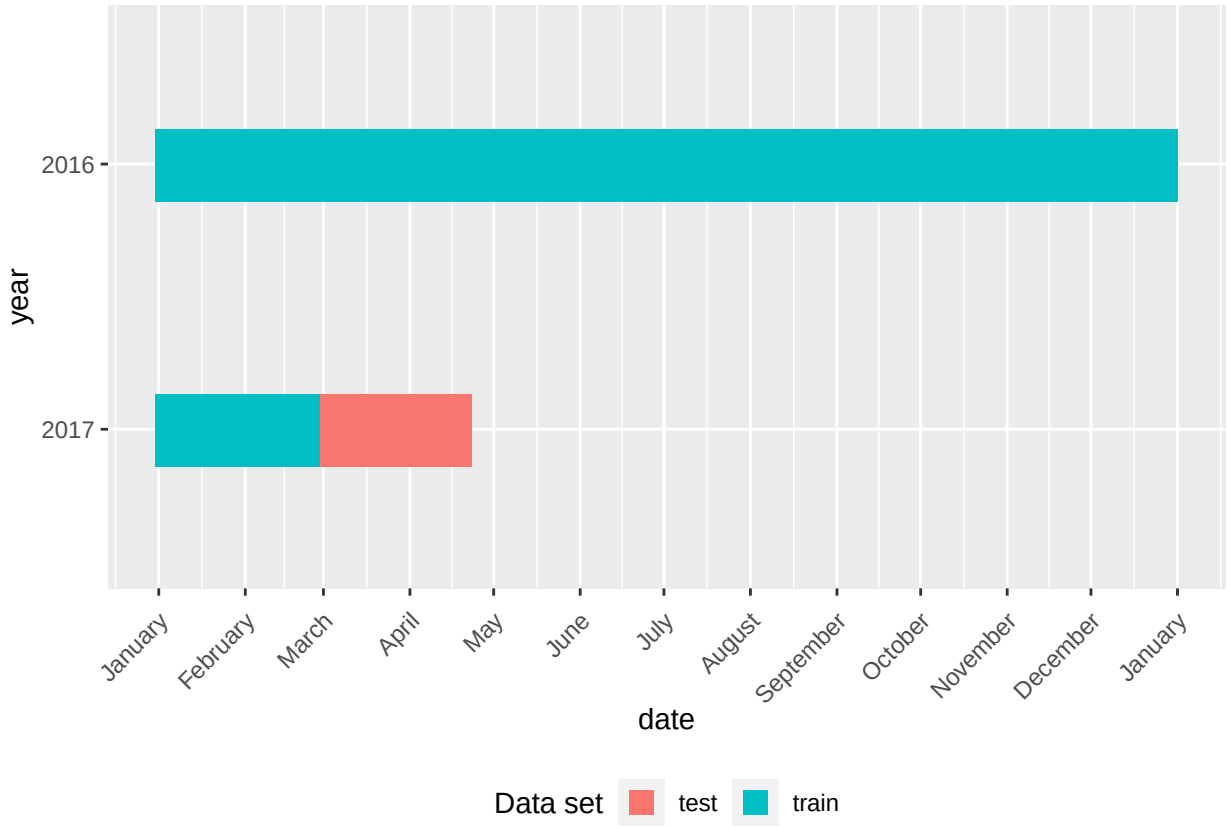

```
holidays %>% summarise(frac = mean(holiday_flg))
```

```
##         frac
## 1 0.06769826
```

There are about 7% holidays in our data

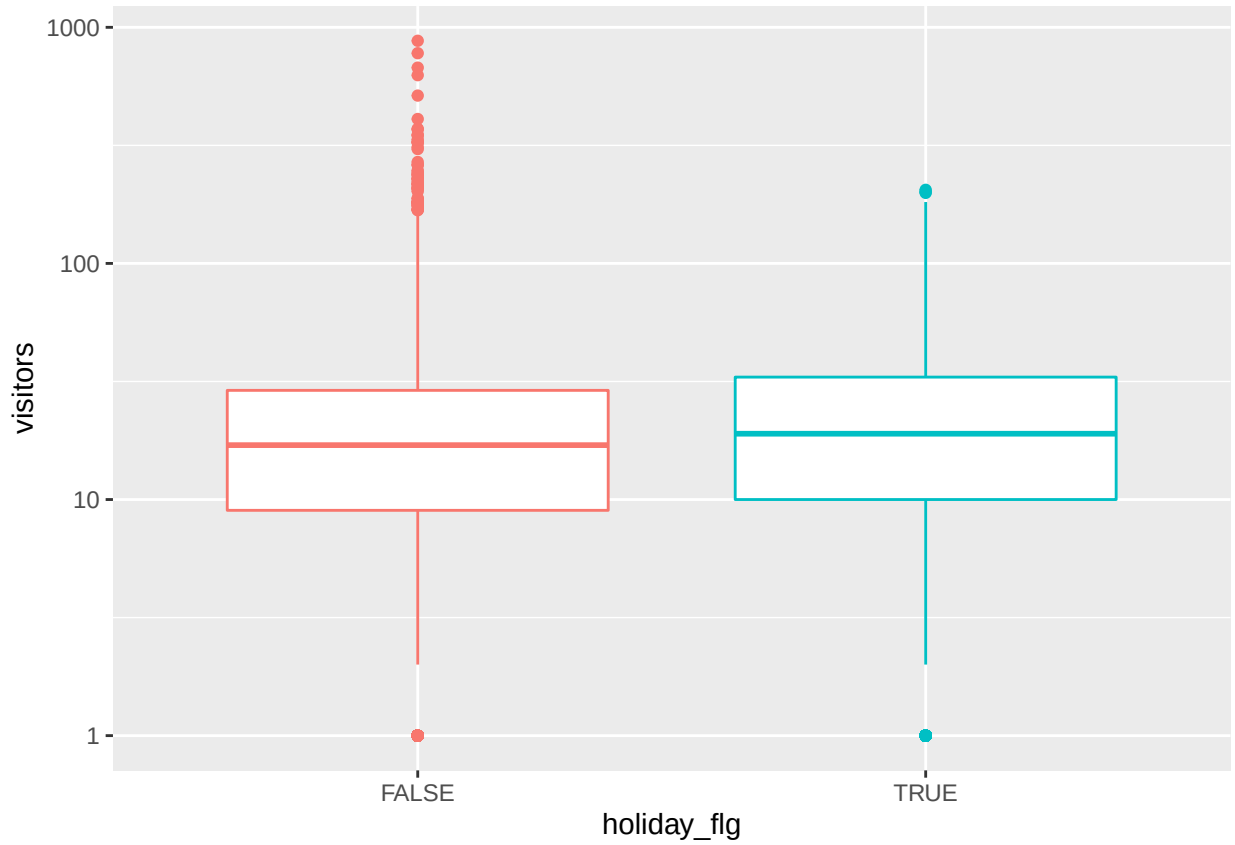
```
foo <- air_visits %>%
  rename(date = visit_date) %>%
  distinct(date) %>%
  mutate(dset = "train")

bar <- test %>%
  separate(id, c("foo", "bar", "date"), sep = "_") %>%
  mutate(date = ymd(date)) %>%
  distinct(date) %>%
  mutate(dset = "test")

foo <- foo %>%
  bind_rows(bar) %>%
  mutate(year = year(date))
year(foo$date) <- 2017

foo %>%
  filter(!is.na(date)) %>%
  mutate(year = fct_relevel(as.factor(year), c("2017","2016"))) %>%
  ggplot(aes(date, year, color = dset)) +
```

```
  geom_point(shape = "|", size = 10) +
  scale_x_date(date_labels = "%B", date_breaks = "1 month") +
  #scale_y_reverse() +
  theme(legend.position = "bottom", axis.text.x  = element_text(angle=45, hjust=1, vjust=0.9)) +
  labs(color = "Data set") +
  guides(color = guide_legend(override.aes = list(size = 4, pch = 15)))
```

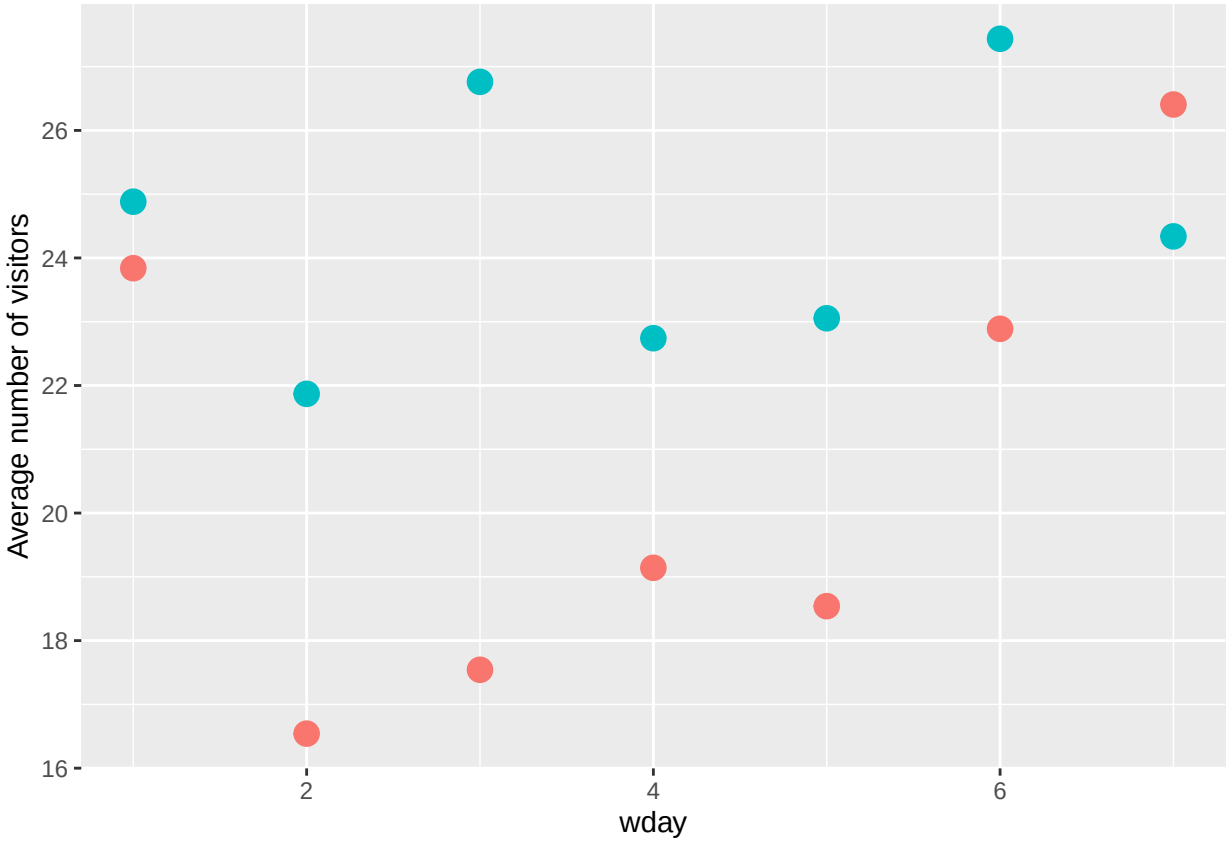


Feature relations

```
foo <- air_visits %>%
  mutate(calendar_date = as.character(visit_date)) %>%
  left_join(holidays, by = "calendar_date")

foo %>%
  ggplot(aes(holiday_flg, visitors, color = holiday_flg)) +
  geom_boxplot() +
  scale_y_log10() +
  theme(legend.position = "none")
```




```
foo %>%
  mutate(wday = wday(date)) %>%
  group_by(wday, holiday_flg) %>%
  summarise(mean_visitors = mean(visitors)) %>%
  ggplot(aes(wday, mean_visitors, color = holiday_flg)) +
  geom_point(size = 4) +
  theme(legend.position = "none") +
  labs(y = "Average number of visitors")
```



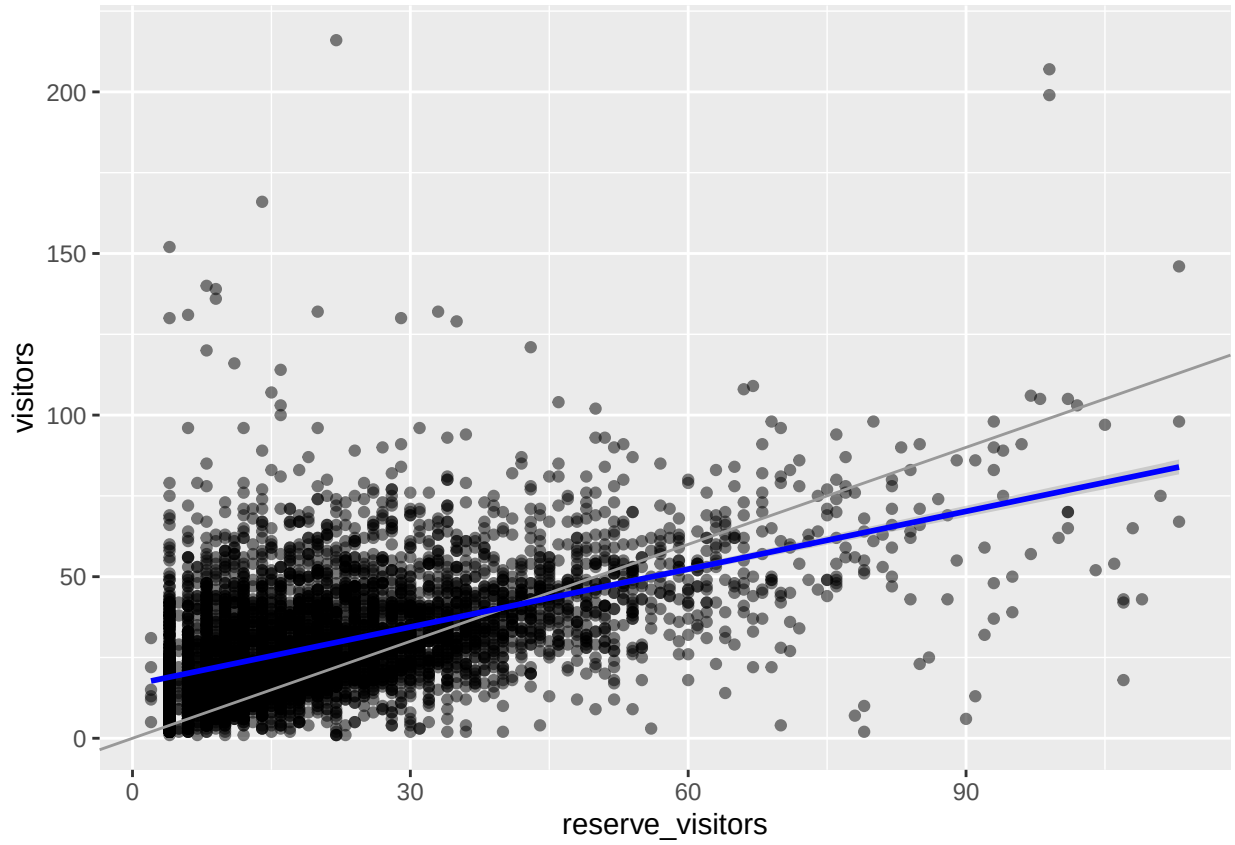
Overall, holidays don't have any impact on the average visitor numbers. As so often, more information is hidden in the details.

While a weekend holiday has little impact on the visitor numbers, and even decreases them slightly, there is a much more pronounced effect for the weekdays; especially Monday and Tuesday.

```
foo <- air_reserve %>%
  mutate(visit_date = date(visit_datetime)) %>%
  group_by(air_store_id,visit_date) %>%
  summarise(reserve_visitors_air = sum(reserve_visitors))


bar <- hpg_reserve %>%
  mutate(visit_date = date(visit_datetime)) %>%
  group_by(hpg_store_id,visit_date) %>%
  summarise(reserve_visitors_hpg = sum(reserve_visitors)) %>%
  inner_join(store_ids, by = "hpg_store_id")

all_reserve <- air_visits %>%
  inner_join(foo, by = c("air_store_id", "visit_date")) %>%
  inner_join(bar, by = c("air_store_id", "visit_date")) %>%
  mutate(reserve_visitors = reserve_visitors_air + reserve_visitors_hpg)
```

```
all_reserve %>%
  filter(reserve_visitors < 120) %>%
  ggplot(aes(reserve_visitors, visitors)) +
  geom_point(color = "black", alpha = 0.5) +
```

```
  geom_abline(slope = 1, intercept = 0, color = "grey60") +
  geom_smooth(method = "lm", color = "blue")
```




```
#ggMarginal(p, type="histogram", fill = "blue", bins=50)
```

The histograms show that the reserve_visitors and visitors numbers peak below ~20 and are largely confined to the range below 100.

The scatter points fall largely above the line of identity, indicating that there were more visitors that day than had reserved a table. This is not surprising, since a certain number of people will always be walk-in customers.

A notable fraction of the points is below the line, which probably indicates that some people made a reservation but changed their mind and didn't go. That kind of effect is probably to be expected and taking it into account will be one of the challenges in this competition.

The linear fit suggests a trend in which larger numbers of reserve_visitors are more likely to underestimate the eventual visitor numbers. This is not surprising either, since I can imagine that it is more likely that (a) a large reservation is cancelled than (b) a large group of people walk in a restaurant without reservation.

Forecasting

1. ARIMA

A popular method for forecasting is the autoregressive integrated moving average model; short ARIMA model. This kind of model consists of three building blocks which parametrised by the three indeces p, d, q as ARIMA(p, d, q):

Auto-regressive / p: we are using past data to compute a regression model for future data. The parameter p

indicates the range of lags; e.g. ARIMA(3,0,0) includes t-1, t-2, and t-3 values in the regression to compute the value at t.

Integrated / d: this is a differencing parameter, which gives us the number of times we are subtracting the current and the previous values of a time series. Differencing removes the change in a time series in that it stabilises the mean and removes (seasonal) trends. This is necessary since computing the lags (e.g. difference between time t and time t-1) is most meaningful if large-scale trends are removed. A time series where the variance (or amount of variability) (and the autocovariance) are time-invariant (i.e. don't change from day to day) is called stationary.

Moving average / q: this parameter gives us the number of previous error terms to include in the regression error of the model.

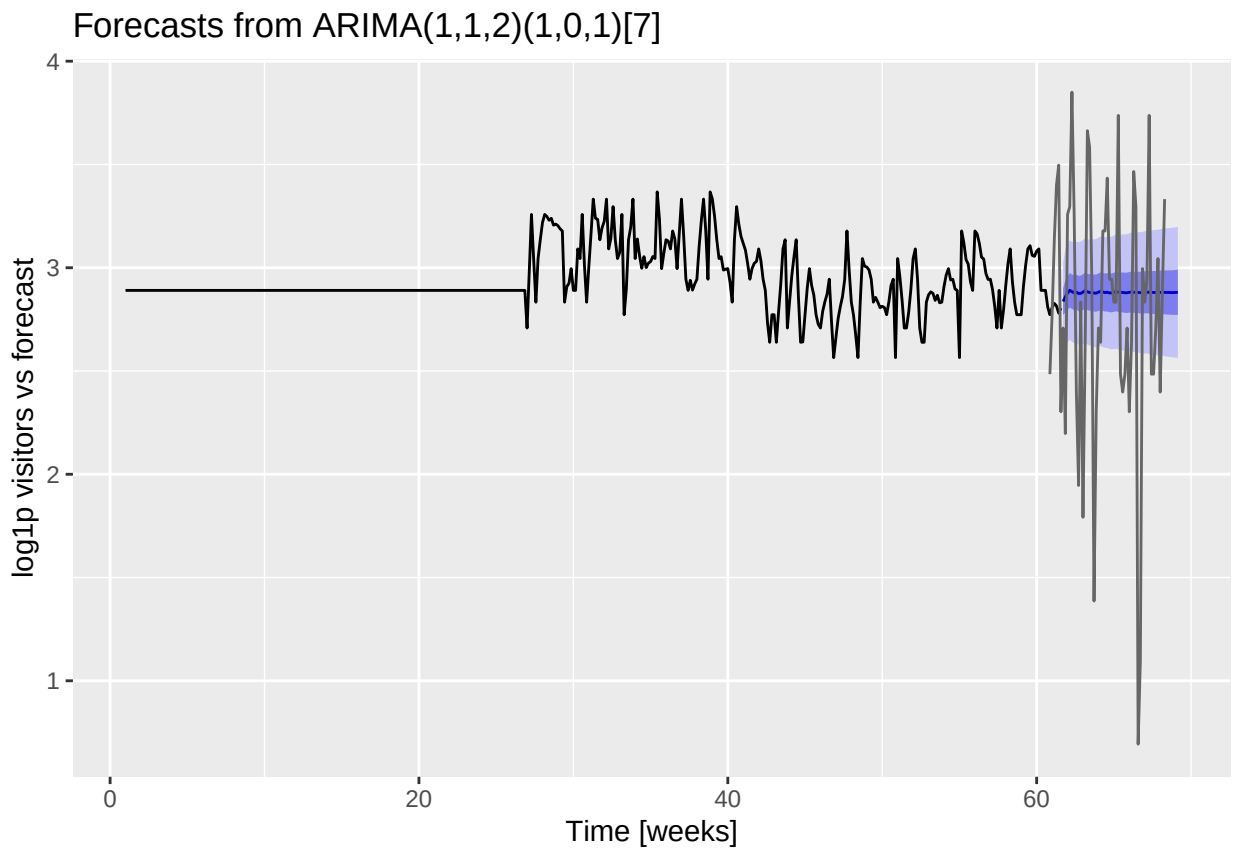
Here we will be using the auto.arima tool which estimates the necessary ARIMA parameters for each individual time series. In order to feed our data to auto.arima we need to turn them into a time-series object using the ts tool. We will also add a step for cleaning and outlier removal via the tsclean function of the forecast package. We have already seen that our data contain a strong weekly cycle, which will be one of the pre-set parameters of our model. We will include this knowledge when transforming our data. Let's set everything up step by step, with comments and explanations, and then turn it into a function. Unhide the code to see how it is implemented.

```
df <- data.frame(matrix(ncol = 2, nrow = 0))
```

```
# Fitting , Predicting and Plotting
plot_auto_arima_air_id <- function(air_id){

  pred_len <- test %>%
    separate(id, c("air", "store_id", "date"), sep = "_") %>%
    distinct(date) %>%
    nrow()

  max_date <- max(air_visits$visit_date)
  split_date <- max_date - pred_len
  all_visits <- tibble(visit_date = seq(min(air_visits$visit_date), max(air_visits$visit_date), 1))

  foo <- air_visits %>%
    filter(air_store_id == air_id)

  visits <- foo %>%
    right_join(all_visits, by = "visit_date") %>%
    mutate(visitors = log1p(visitors)) %>%
    replace_na(list(visitors = median(log1p(foo$visitors)))) %>%
    rownames_to_column()

  visits_train <- visits %>% filter(visit_date <= split_date)
  visits_valid <- visits %>% filter(visit_date > split_date)

  arima.fit <- auto.arima(tsclean(ts(visits_train$visitors, frequency = 7)),
                          stepwise = FALSE, approximation = FALSE)

  arima_visits <- arima.fit %>% forecast(h = pred_len, level = c(50,95))
  predictions <- data.frame(visits_valid$visitors,arima_visits$mean)
  colnames(df) <- colnames(predictions)
  df <- merge(df, predictions,by = colnames(df),all.x = T, all.y = T )
  p <-arima_visits %>%
    autoplot +
```

```
    geom_line(aes(as.integer(rowname)/7, visitors), data = visits_valid, color = "grey40") +
    labs(x = "Time [weeks]", y = "log1p visitors vs forecast")
  print(p)
  return(df)
}
```

```
plot_auto_arima_air_id("air_f3f9824b7d70c3cf")
```




```
##    visits_valid.visitors arima_visits.mean
## 1              0.6931472          2.880134
## 2              1.0986123          2.880535
## 3              1.3862944          2.881093
## 4              1.7917595          2.877247
## 5              1.9459101          2.881470
## 6              2.1972246          2.875207
## 7              2.3025851          2.877919
## 8              2.3025851          2.879035
## 9              2.3025851          2.879300
## 10             2.3978953          2.879649
## 11             2.3978953          2.879815
## 12             2.3978953          2.880072
## 13             2.4849066          2.836433
## 14             2.4849066          2.880178
## 15             2.4849066          2.880657
## 16             2.4849066          2.880760
## 17             2.4849066          2.881313
```

```
## 18             2.6390573          2.880600
## 19             2.6390573          2.880973
## 20             2.6390573          2.881295
## 21             2.7080502          2.878200
## 22             2.7080502          2.879168
## 23             2.7080502          2.880452
## 24             2.7080502          2.881894
## 25             2.7725887          2.859585
## 26             2.8332133          2.876837
## 27             2.8332133          2.878853
## 28             2.8332133          2.879606
## 29             2.8332133          2.880752
## 30             2.8903718          2.880426
## 31             2.9444390          2.878660
## 32             2.9444390          2.879846
## 33             2.9444390          2.880497
## 34             2.9444390          2.880834
## 35             2.9957323          2.879515
## 36             3.0445224          2.879754
## 37             3.1354942          2.876901
## 38             3.1780538          2.881793
## 39             3.1780538          2.884361
## 40             3.2580965          2.875837
## 41             3.2580965          2.883514
## 42             3.2958369          2.880985
## 43             3.2958369          2.881759
## 44             3.3322045          2.881171
## 45             3.4011974          2.890836
## 46             3.4339872          2.879980
## 47             3.4657359          2.882190
## 48             3.4965076          2.884163
## 49             3.5835189          2.882493
## 50             3.6635616          2.886243
## 51             3.7376696          2.881585
## 52             3.7376696          2.883073
## 53             3.8501476          2.888987
```

The time series above is reasonable complete, but we see that the long gaps (and our median filling) lead to problems in the predictions in the series where we loose the weekly periodicity.

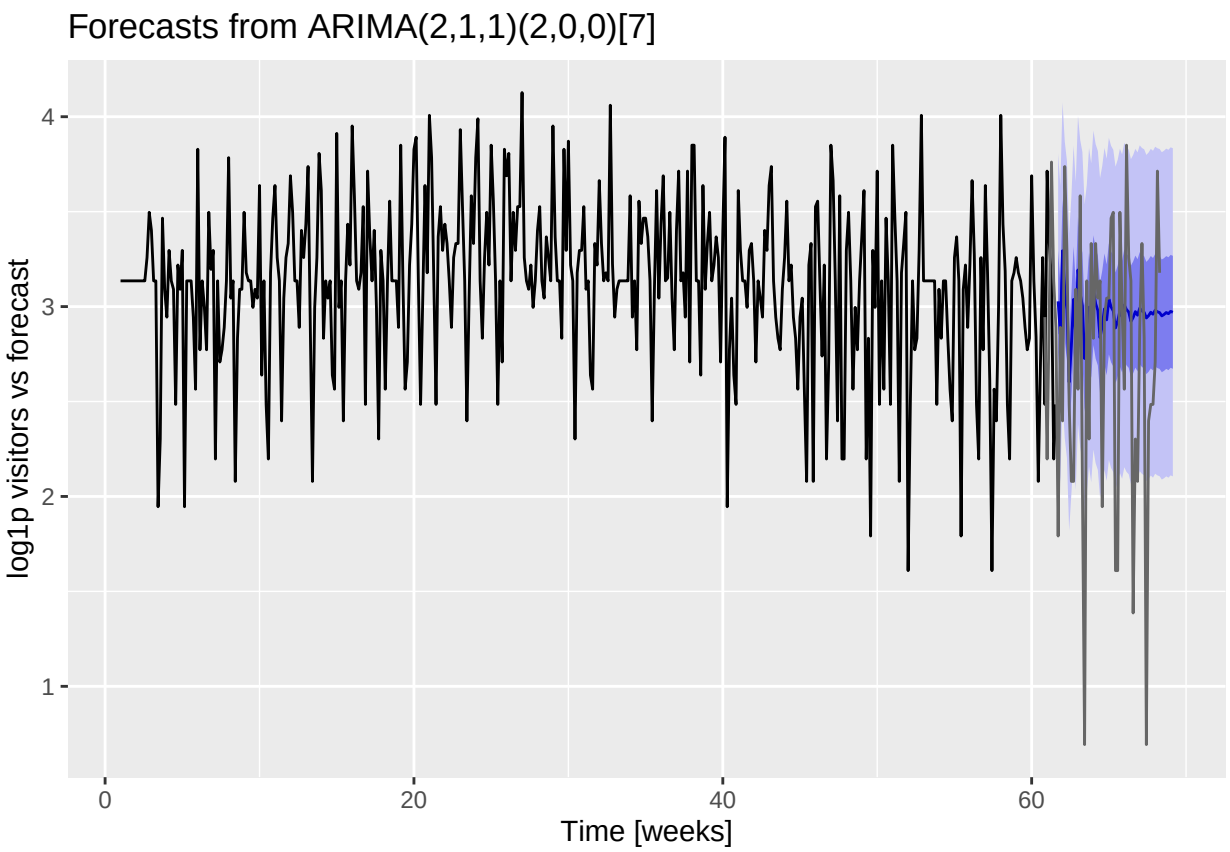
This was just for one store id. Now lets lets forecast for other stores too.

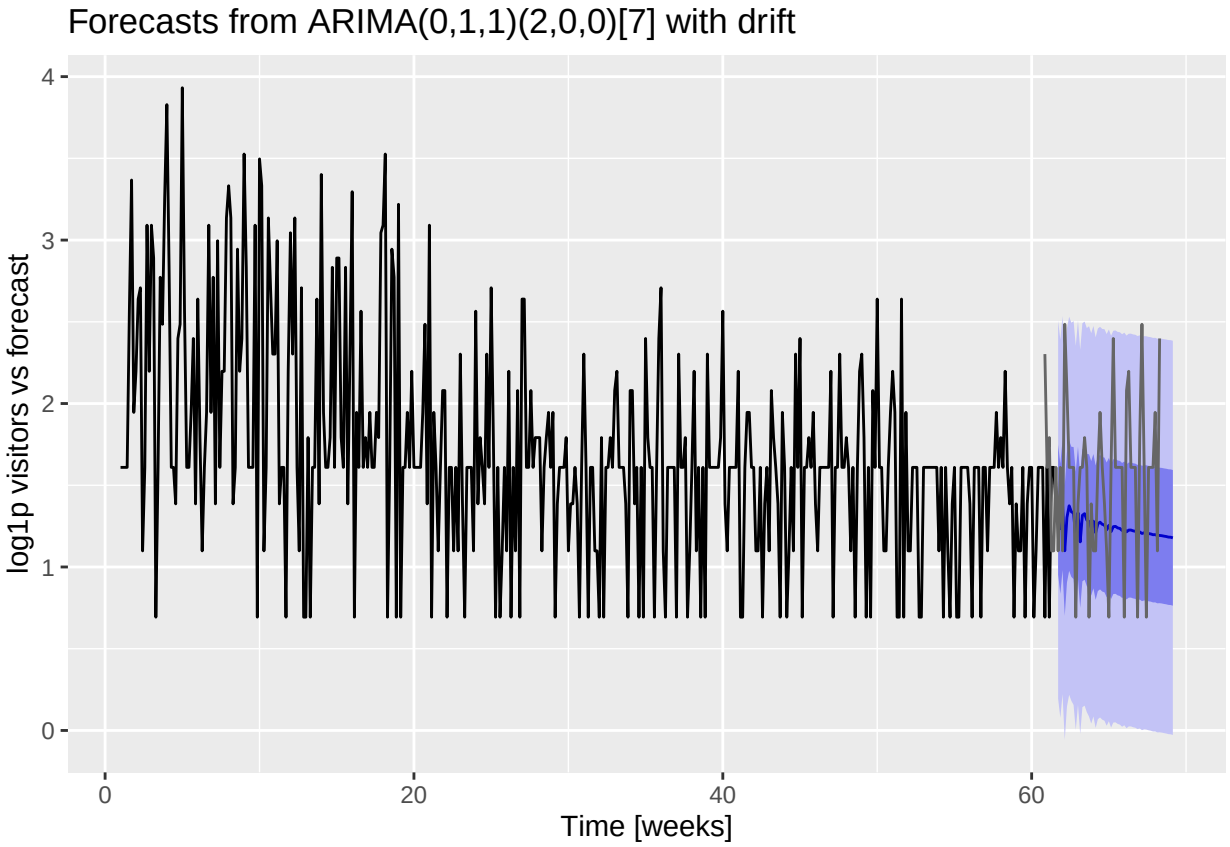
NOTE: We have more than 800 stores in the test data and if we forecast on the entire test dataset then it takes so much time (~3 to 4 hours) to forecast. Considering that I am taking only 5 stores for the forecast in all models. We can just remove If and Break statement from the below code and we are good to forecast on all store ids of the test data.

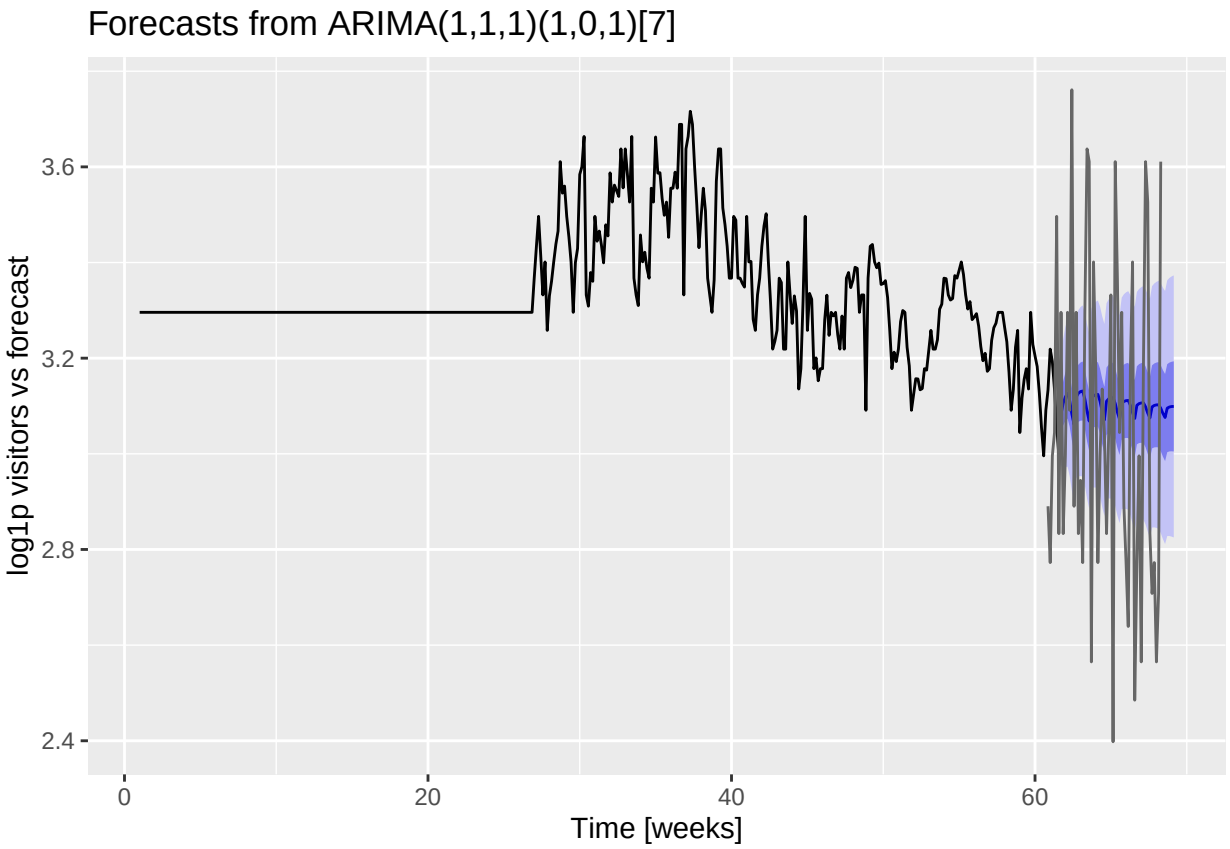
Below code will generate forecast plot for 5 stores.

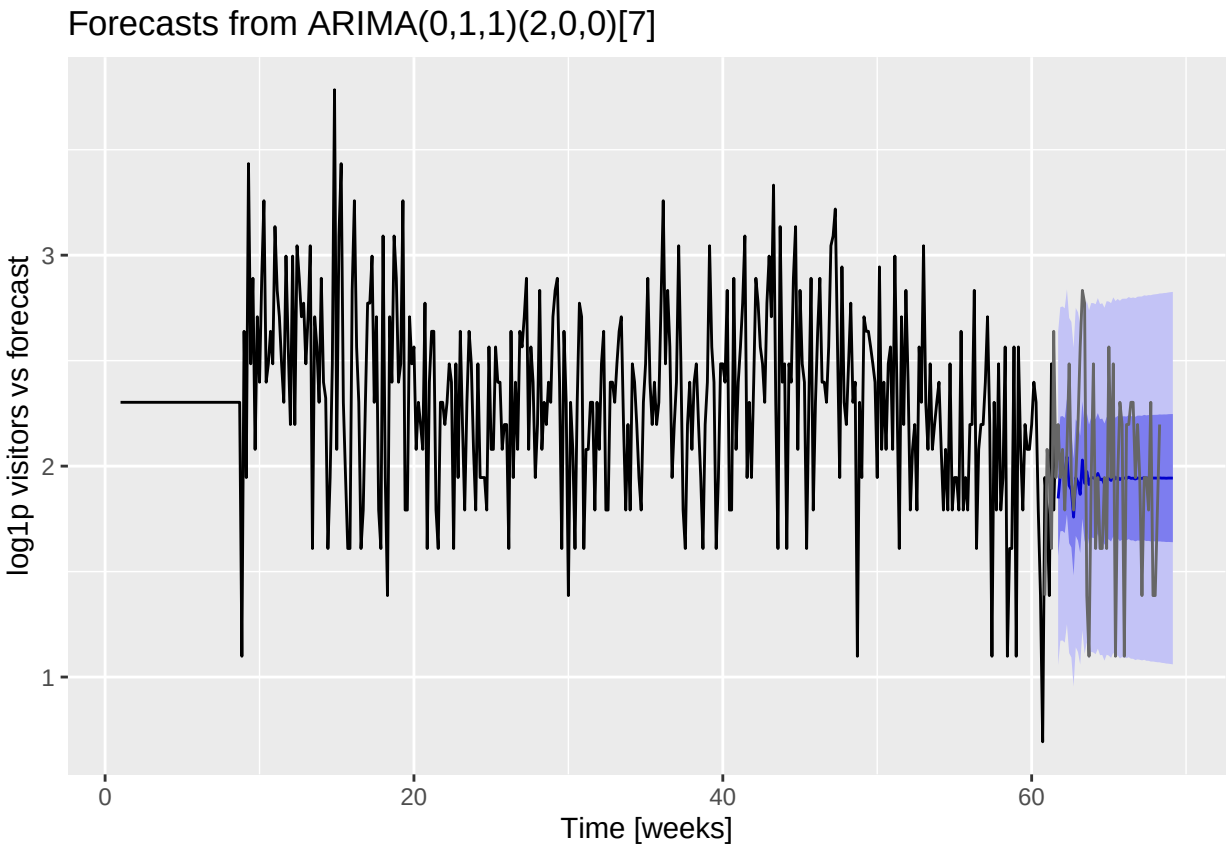
```
graph_list <- list()
index <- 1
for (i in unique(air_visits$air_store_id)){
    if (index==6){break}
    a <- plot_auto_arima_air_id(i)
    graph_list[[index]] <- data.frame(a)
    index <- index +1
```

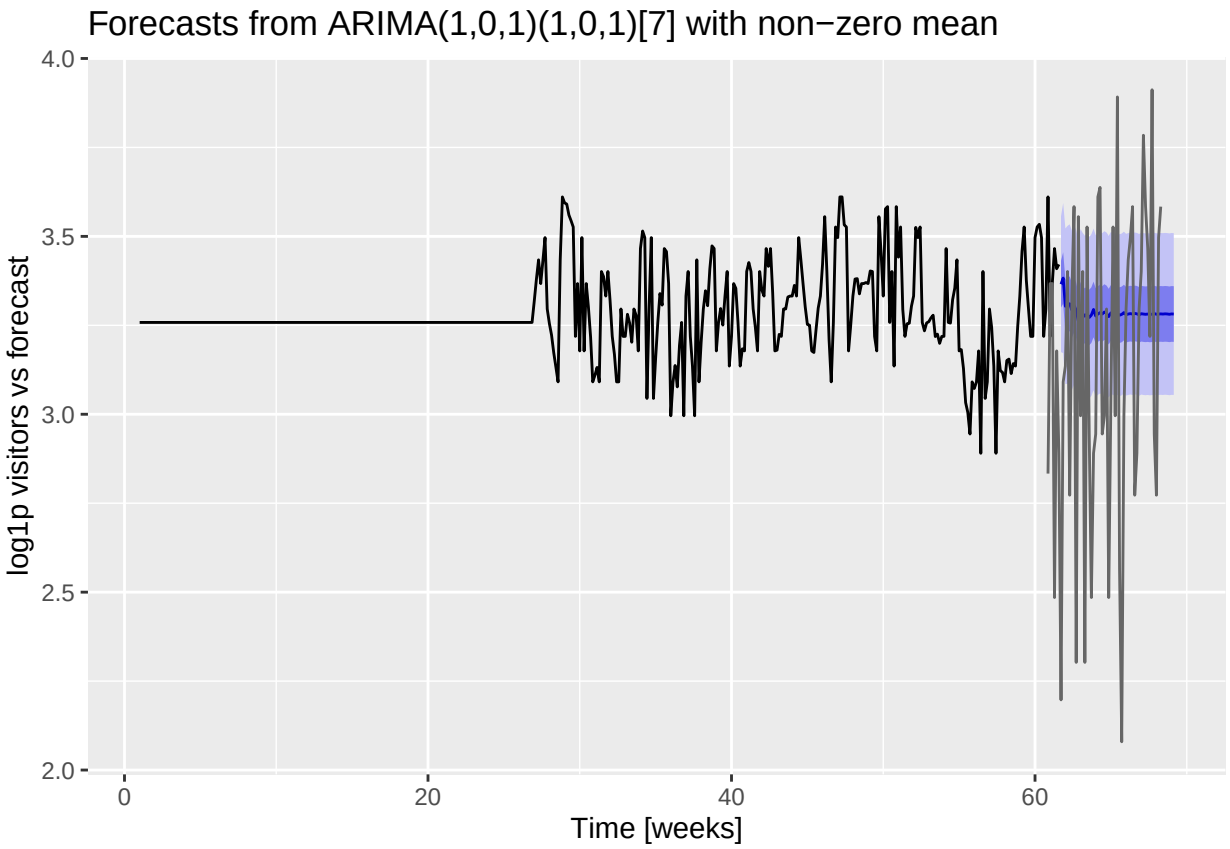
```
}
```


Forecasts from ARIMA(2,1,1)(2,0,0)[7]
log1p visitors vs forecast
4
3
2
1
0
20
40
60
Time [weeks]


Forecasts from ARIMA(0,1,1)(2,0,0)[7] with drift
log1p visitors vs forecast
Time [weeks]
0
1
2
3
4
0
20
40
60


Forecasts from ARIMA(1,1,1)(1,0,1)[7]
log1p visitors vs forecast
3.6
3.2
2.8
2.4
0
20
40
60
Time [weeks]


Forecasts from ARIMA(0,1,1)(2,0,0)[7]
log1p visitors vs forecast
3
2
1
0
20
40
60
Time [weeks]



Stitch together the results for all stores to calculate the RMSE.

```
base_df <- graph_list[[1]] %>%
  bind_rows(graph_list[[2]]) %>%
  bind_rows(graph_list[[3]]) %>%
  bind_rows(graph_list[[4]]) %>%
  bind_rows(graph_list[[5]])
```

```
rmse(base_df$visits_valid.visitors,base_df$arima_visits.mean)
```

```
## [1] 0.5151394
```

2. Holt Winters

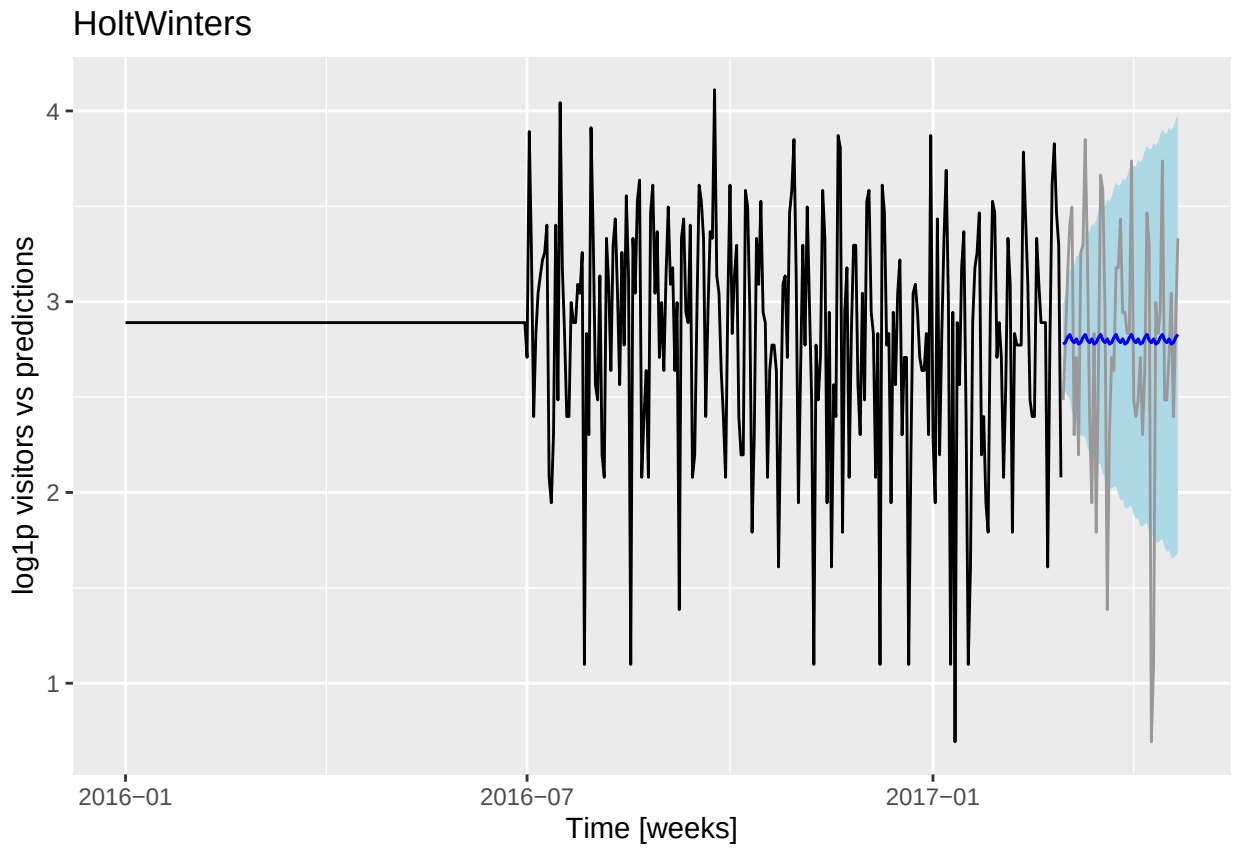
A more traditional time series filtering and forecasting is the Holt-Winters algorithm, as implemented in the stats package. This is an exponential smoothing method which uses moving averages to take into account the presence of a trend in the data. Here we define a default seasonal model in a fitting and plotting function.

```
df <- data.frame(matrix(ncol = 2, nrow = 0))
```

```
plot_hw_air_id <- function(air_id){

  pred_len <- test %>%
    separate(id, c("air", "store_id", "date"), sep = "_") %>%
    distinct(date) %>%
    nrow()

  max_date <- max(air_visits$visit_date)
```

```
  split_date <- max_date - pred_len
  all_visits <- tibble(visit_date = seq(min(air_visits$visit_date), max(air_visits$visit_date), 1))

  foo <- air_visits %>%
    filter(air_store_id == air_id)

  visits <- foo %>%
    right_join(all_visits, by = "visit_date") %>%
    mutate(visitors = log1p(visitors)) %>%
    replace_na(list(visitors = median(log1p(foo$visitors)))) %>%
    rownames_to_column()

  visits_train <- visits %>% filter(visit_date <= split_date)
  visits_valid <- visits %>% filter(visit_date > split_date)

  hw.fit <- HoltWinters(tsclean(ts(visits_train$visitors, frequency = 7)))

  hw_visits <- predict(hw.fit, n.ahead = pred_len, prediction.interval = T, level = 0.95) %>%
    as.tibble() %>%
    bind_cols(visits_valid)

  predictions <- data.frame(visits_valid$visitors,hw_visits$fit)
  colnames(df) <- colnames(predictions)
  df <- merge(df, predictions,by = colnames(df),all.x = T, all.y = T )

  p <- visits_train %>%
    ggplot(aes(visit_date, visitors)) +
    geom_line() +
    geom_ribbon(data = hw_visits, aes(x = visit_date, ymin = lwr, ymax = upr), fill = "light blue") +
    geom_line(data = hw_visits, aes(visit_date, visitors), color = "grey60") +
    geom_line(data = hw_visits, aes(visit_date, fit), color = "blue") +
    geom_line(data = hw_visits, aes(visit_date, fit), color = "blue") +
    labs(x = "Time [weeks]", y = "log1p visitors vs predictions") +
    ggtitle("HoltWinters")

  print(p)
  return(df)
}
```

```
plot_hw_air_id("air_f3f9824b7d70c3cf")
```

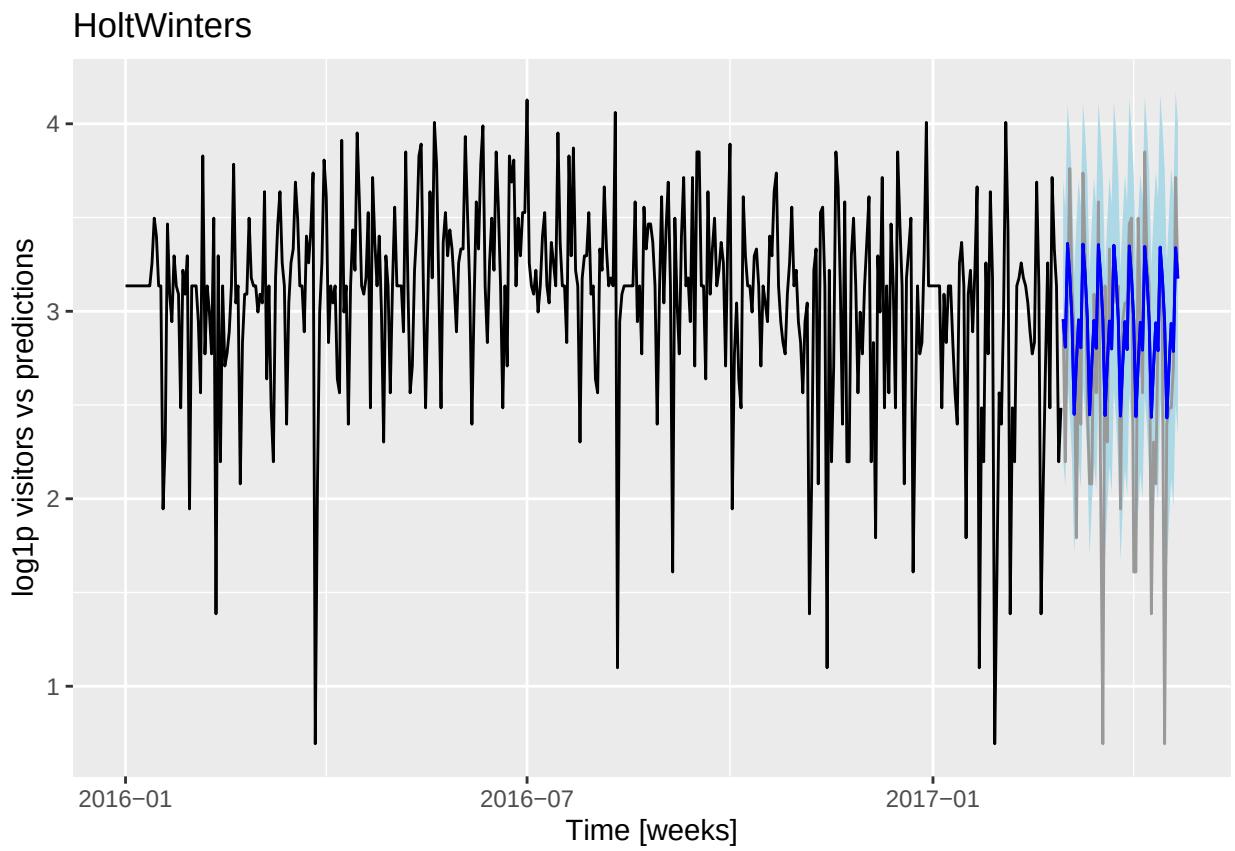


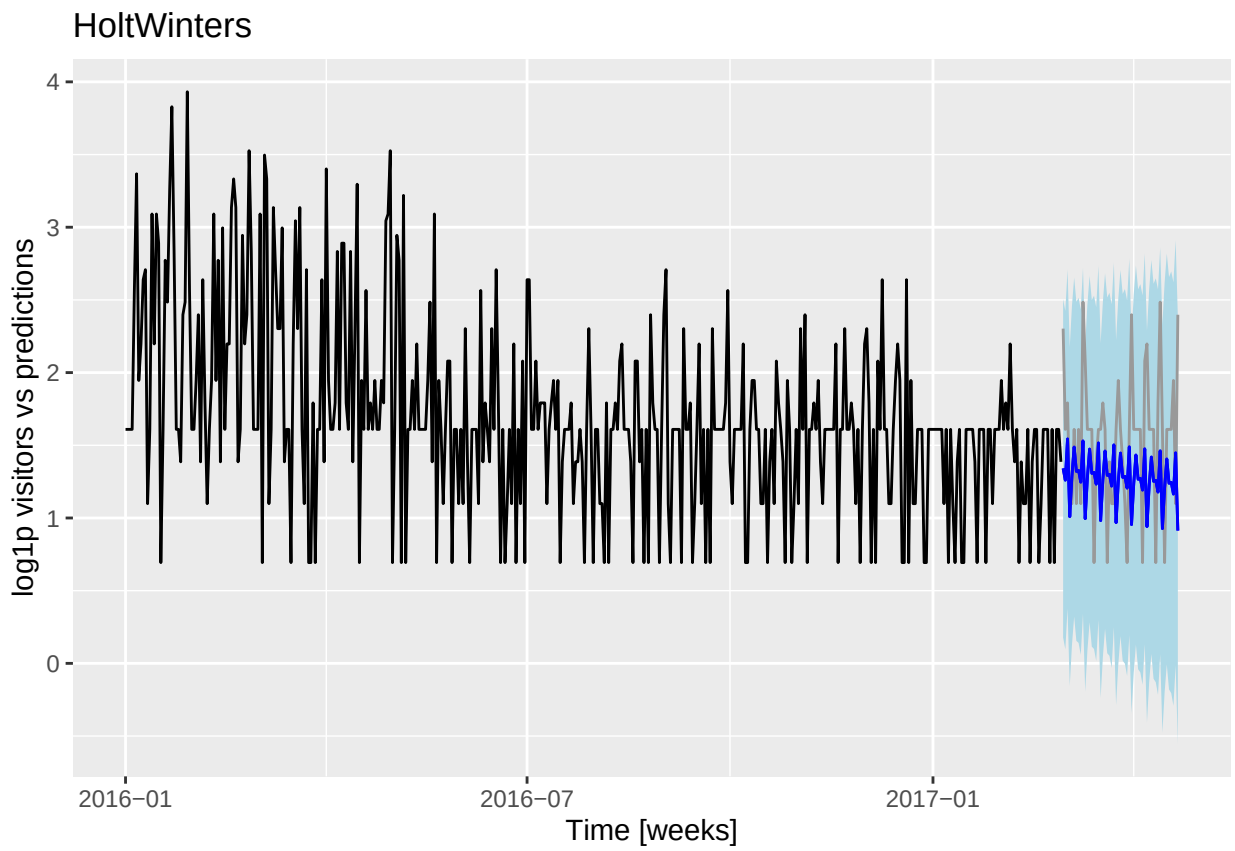

```
##    visits_valid.visitors hw_visits.fit
## 1              0.6931472      2.785647
## 2              1.0986123      2.804504
## 3              1.3862944      2.804504
## 4              1.7917595      2.785814
## 5              1.9459101      2.804504
## 6              2.1972246      2.778010
## 7              2.3025851      2.778010
## 8              2.3025851      2.785647
## 9              2.3025851      2.785814
## 10             2.3978953      2.785647
## 11             2.3978953      2.785647
## 12             2.3978953      2.785814
## 13             2.4849066      2.778010
## 14             2.4849066      2.785647
## 15             2.4849066      2.798097
## 16             2.4849066      2.798097
## 17             2.4849066      2.804504
## 18             2.6390573      2.811963
## 19             2.6390573      2.811963
## 20             2.6390573      2.811963
## 21             2.7080502      2.778010
## 22             2.7080502      2.785814
## 23             2.7080502      2.804504
## 24             2.7080502      2.804504
## 25             2.7725887      2.785814
```

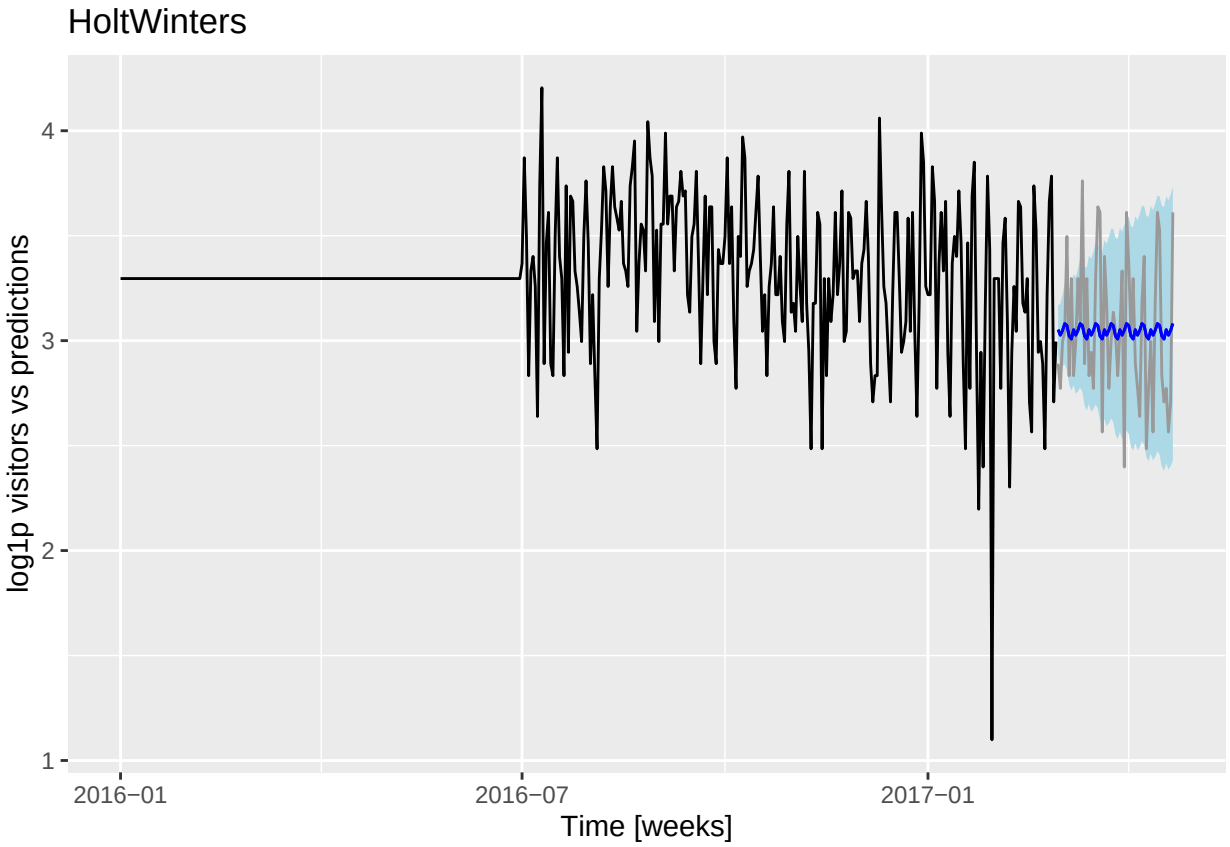
```
## 26             2.8332133      2.778010
## 27             2.8332133      2.785814
## 28             2.8332133      2.785814
## 29             2.8332133      2.811963
## 30             2.8903718      2.811963
## 31             2.9444390      2.778010
## 32             2.9444390      2.785647
## 33             2.9444390      2.804504
## 34             2.9444390      2.811963
## 35             2.9957323      2.778010
## 36             3.0445224      2.778010
## 37             3.1354942      2.811963
## 38             3.1780538      2.798097
## 39             3.1780538      2.827911
## 40             3.2580965      2.785814
## 41             3.2580965      2.798097
## 42             3.2958369      2.798097
## 43             3.2958369      2.811963
## 44             3.3322045      2.827911
## 45             3.4011974      2.827911
## 46             3.4339872      2.785647
## 47             3.4657359      2.827911
## 48             3.4965076      2.798097
## 49             3.5835189      2.798097
## 50             3.6635616      2.827911
## 51             3.7376696      2.827911
## 52             3.7376696      2.827911
## 53             3.8501476      2.827911
```

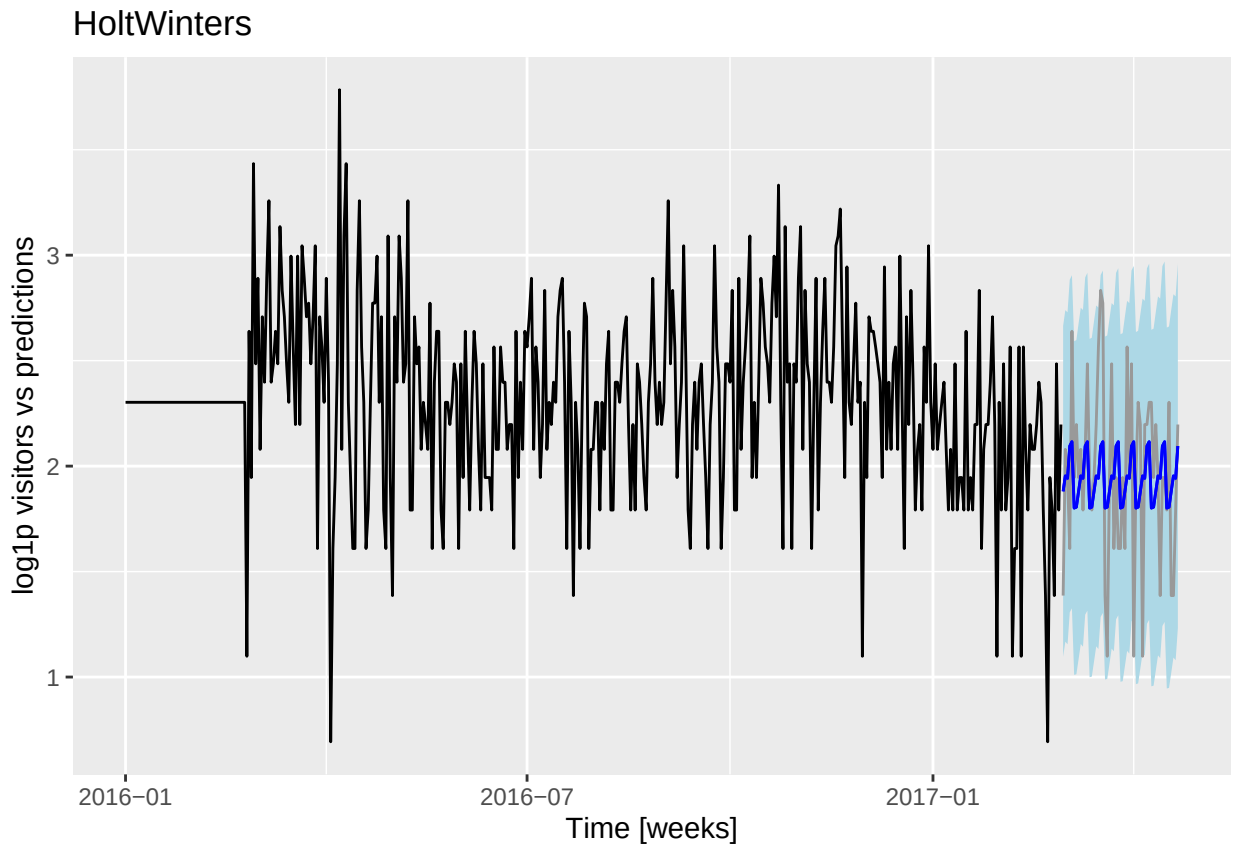
This was just for one store id. Now lets lets forecast for other stores too.Below code will generate forecast plot for 5 stores.

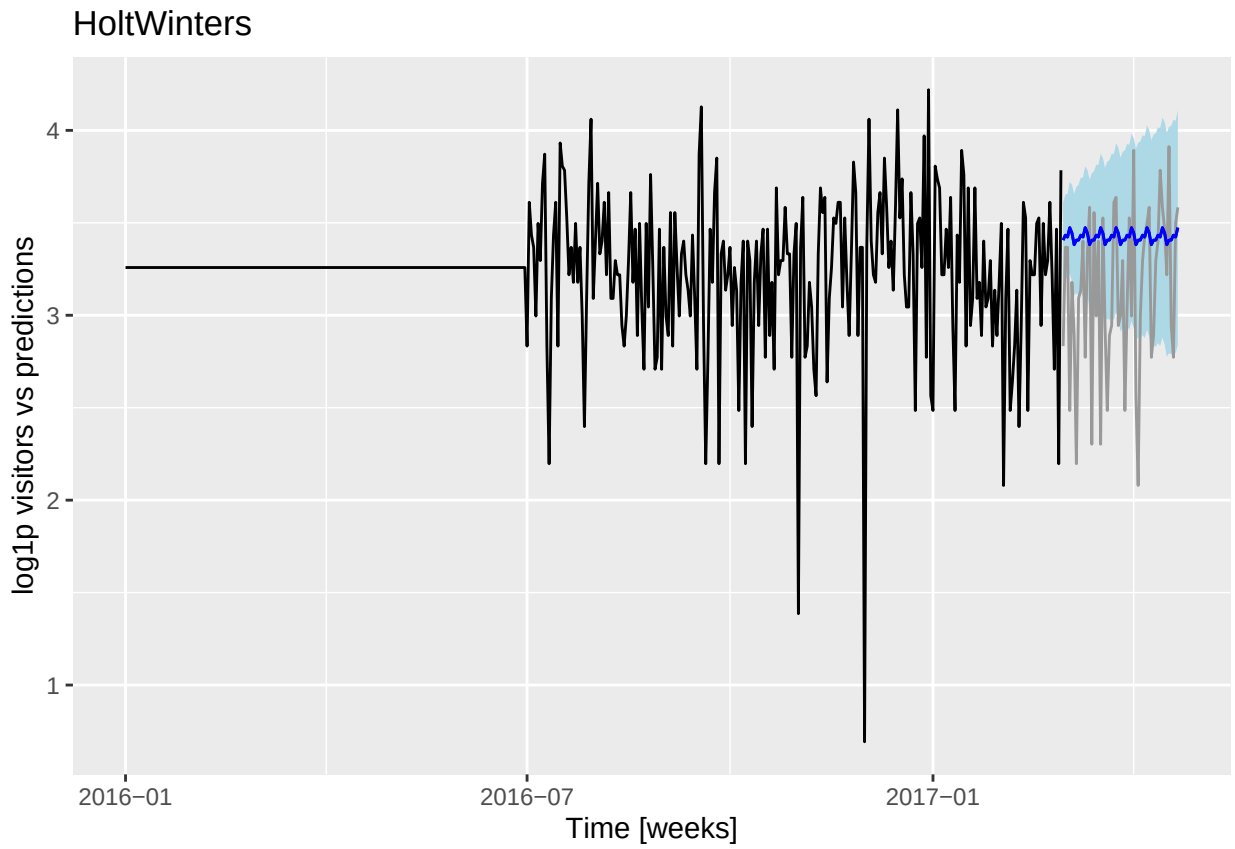
```
graph_list <- list()
index <- 1
for (i in unique(air_visits$air_store_id)){
    if (index==6){break}
    a <- plot_hw_air_id(i)
    graph_list[[index]] <- data.frame(a)
    index <- index +1
}
```


HoltWinters
log1p visitors vs predictions
4
3
2
1
2016−01
2016−07
2017−01
Time [weeks]


HoltWinters
log1p visitors vs predictions
4
3
2
1
0
2016−01
2016−07
2017−01
Time [weeks]


HoltWinters
log1p visitors vs predictions
4
3
2
1
2016−01
2016−07
2017−01
Time [weeks]


HoltWinters
log1p visitors vs predictions
3
2
1
2016−01
2016−07
2017−01
Time [weeks]



Stitch together the results for all stores to calculate the RMSE.

```
base_df <- graph_list[[1]] %>%
  bind_rows(graph_list[[2]]) %>%
  bind_rows(graph_list[[3]]) %>%
  bind_rows(graph_list[[4]]) %>%
  bind_rows(graph_list[[5]])
```

```
rmse(base_df$visits_valid.visitors,base_df$hw_visits.fit)
```

```
## [1] 0.5050079
```

Conclusion

We use RMSE (root mean square error) to evaluate our implemented models and also considered a log of visitors to avoid any skewness in the number of visitors. We can observe that we are getting the lowest RMSE score for the Holt-Winters method so we can choose that method as a final forecasting method. We can also extend this problem and apply advanced techniques prophet and LightGBM for better accuracy.