Dengue Fever Forecasting

Shaun Harrington

2023-08-18

This paper outlines the approach taken for DengAI: Predicting Disease Spread hosted by Driven Data. Three models are evaluated, a linear model, an ARIMAX model, and a Prophet model with boosted errors. An ensemble model is also calculated as a simple average of the models. The Boosted Prophet model appears to be the model most capable of adequately modeling the outbreaks however the ensemble model creates a better prediction on average. The Data Driven prediction submissions can be found under the user shaun17 where the best Mean Absolute Error scored was 22.762.

```
#|eval: true
#|echo: false

require(tidyverse)
require(fpp3)
library(modeltime)

if(!stringr::str_detect(basename(getwd()), "Time Series") & stringr::str_detect(dirname(ge
    repeat{
      setwd("../")
      if(stringr::str_detect(basename(getwd()), "Time Series")){
        break
      }
    }
  }

  if(basename(getwd()) != "Final") setwd(file.path(getwd(), "Final"))

source(file.path("00_Setup.R"))

#
all_cores <- parallel::detectCores()
```

```
cl <- makePSOCKcluster(all_cores)
registerDoParallel(cl)
```

Introduction

Dengue Fever is a disease transmitted by mosquitoes and affects a large portion of the developing world. Being able to accurately predict outbreaks of Dengue Fever will aid governments and non-profits to allocate resources more efficiently.

This paper will develop models to forecast weekly Dengue Fever cases in Iquitos, Peru and San Juan, Puerto Rico. Three types of models will be estimated: an average seasonality plus trend linear model, an ARIMAX model, and a prophet model with boosted errors.

Methodology

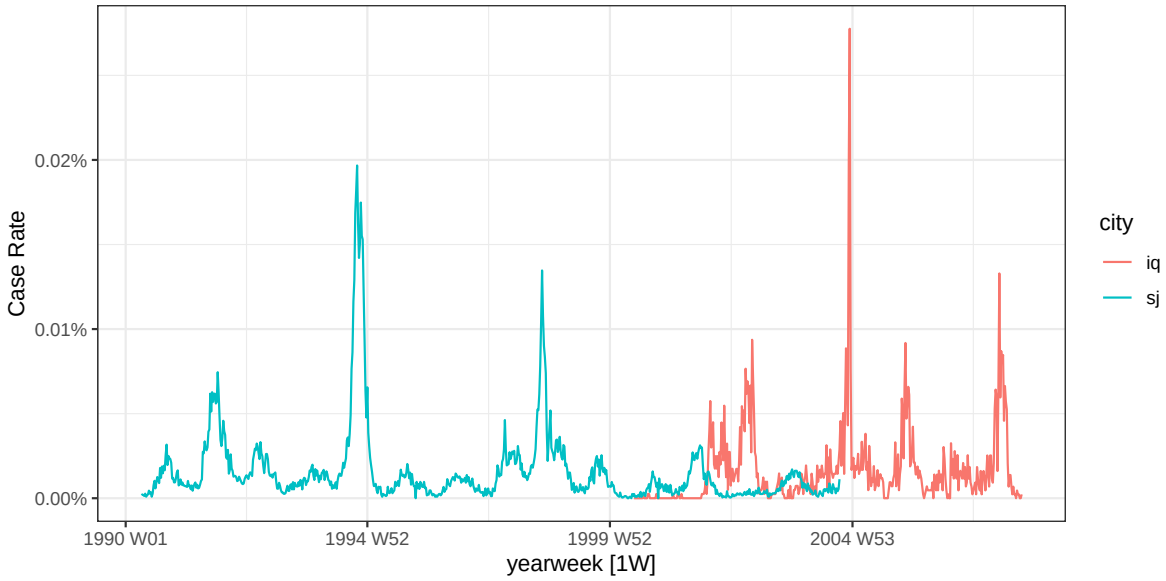
Data Description & Exploration

The dataset is provided by Driven Data where a training set and test set are given. The training dataset not only contains weekly Dengue Fever cases but various weather variables including precipitation, humidity, various temperatures, vegetation growth, and annual population levels for each city. Because Dengue Fever is transmitted from mosquitoes, we are essentially forecasting mosquito populations. This is used as a theoretical guide for feature engineering and modeling. Several other variables are created from these existing exogenous variables:

- Growing Degree Days (GDD): GDDs are used to approximate mosquito population growth rates as they are positively correlated. A baseline of 10°C is used with a cap of 45°C. These are used because mosquito population begin growing around 10°C and stop around 45°C (NRCC (2023)).
- Rolling sums & averages: because weather can be volatile, rolling sums and averages are included in the models to better capture the factors that would cause mosquito populations to grow.

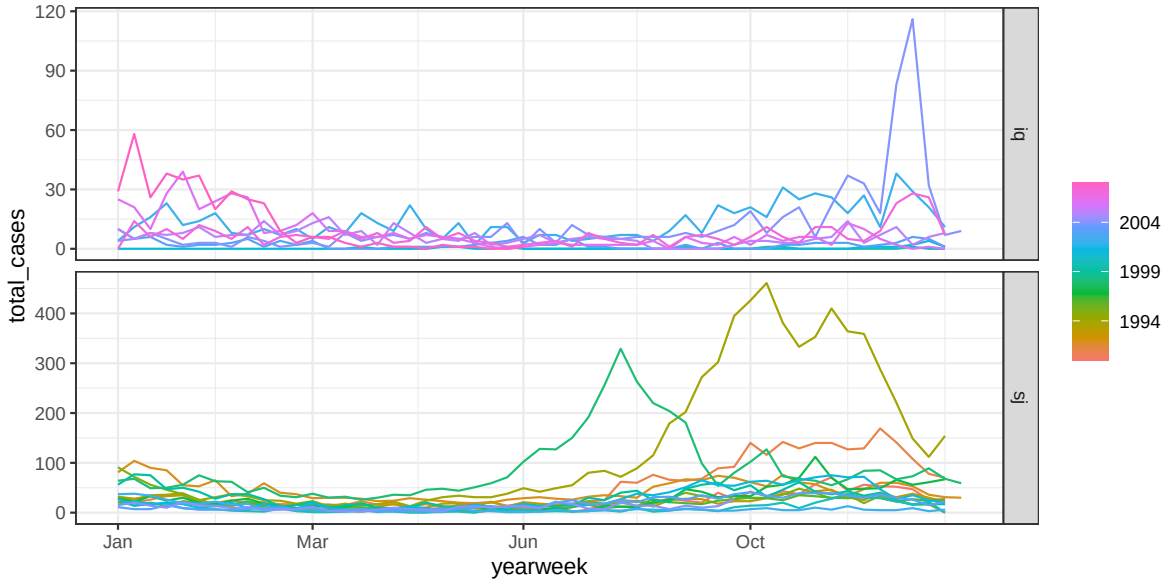
Cases have been relatively constant over time with occasional large outbreaks. San Juan appears to be possible getting the situation under more control, but Iquitos remains more elevated.

```
train %>%
  autoplot(case_rate) +
  scale_y_continuous("Case Rate", labels = label_percent())
```



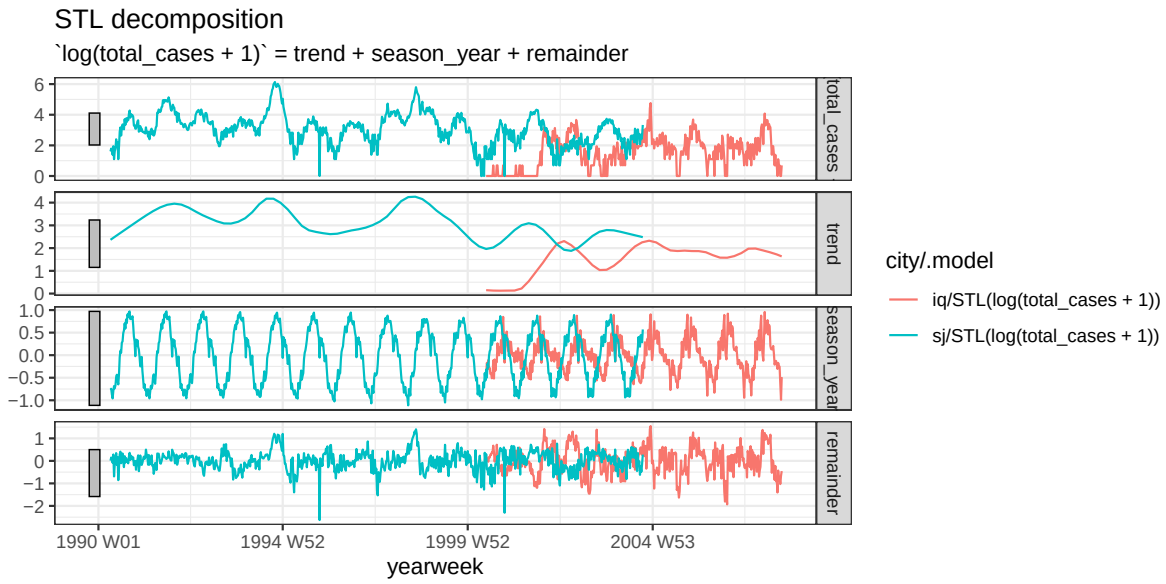
Cases have a strong seasonality that peaks in the fall to early winter, except for a single San Juan outbreak that occurred in the summer.

```
train %>%
  gg_season(total_cases)
```



The log transformed decomposition demonstrates these trends and seasonality.

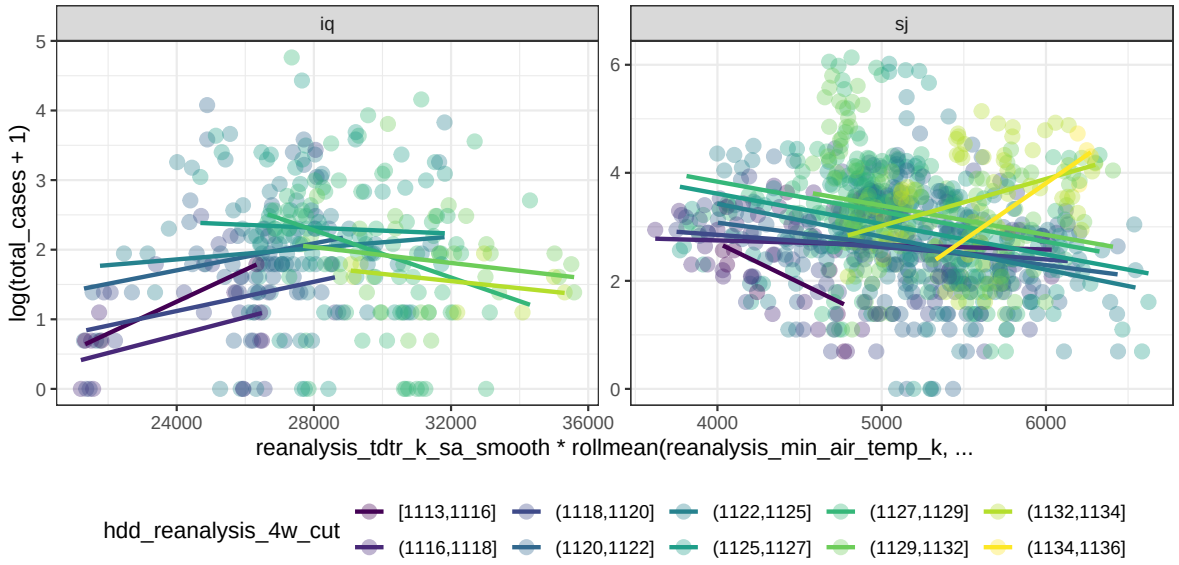
```
train %>%
  model(STL(log(total_cases+1))) %>%
  components() %>%
  autoplot()
```



A complicating factor is the relation between the exogenous variables and case counts. The chart below highlights this complexity The interaction between tdtr and the minimum air temperature does not appear correlated with case counts until 4-week GDD are also accounted for. Then when it is accounted for, there are differing effects depending on how many GDDs there have been in the past 4 weeks.

```
train.all %>%
  filter(cases_cumulative > 1000) %>%
  ggplot(aes(
    y = log(total_cases + 1),
    x = reanalysis_tdtr_k_sa_smooth *
      rollmean(reanalysis_min_air_temp_k, k = 8, NA, align = "right") *
      rollmean(station_diur_temp_rng_c, k = 8, NA, align = "right"),
    color = hdd_reanalysis_4w_cut
  )) +
  geom_point(size = 3, alpha = .35) +
```

```
  geom_smooth(method = "lm", se = F) +
  facet_wrap(city ~ ., scales = "free") +
  scale_color_viridis_d(option = "D")  +
  theme(legend.position = "bottom")
```



Modeling

Each of the three models attempts to forecast cases from a different perspective. The average season plus trend model implicitly makes the assumption that cases are not determined by exogenous variables, they're mostly a seasonal random walk with drift. This model forecasts Box-Cox transformed cumulative cases.

The ARIMAX likewise models Box-Cox transformed cumulative cases but includes an ordinary difference, thus models transformed weekly cases. Seasonality is captured in both these models with the inclusion of Fourier terms. ARIMAX also includes exogenous variables to make the prediction. It makes the assumption that cases are autocorrelated but also linearly dependent on other variables.

The last model is a prophet model with boosted errors. A Prophet model is first fit to the series which captures the seasonality and trend of the series. The xgboost algorithm is then fit to the errors using exogenous variables and the hyperparameters are selected through cross validation. Where the first two models are linear, both components of this model, the Prophet and boosted trees, are nonlinear and may better model the exponential nature of Dengue Fever cases. The two linear models necessitate that the dependent variable is transformed in a

way to more closely resemble linearity. Benedum (2020) found that machine learning models have better predictions than linear models when no exogenous variables are considered, but all performed about equally when exogenous weather variables are used.

Modeling Training

The training set provided by Driven Data is separated into two sets: a training and validation set. Iquitos models were trained on 2000-06-26 UTC–2008-06-23 UTC with the validation set covering 2008-06-30 UTC–2010-06-21 UTC. San Juan models were trained on 1990-04-30 UTC–2004-09-20 UTC with the validation set covering 2004-09-27 UTC–2008-04-21 UTC.

The validation set was used to guide the variable selection and the hyperparameter tuning.

Model 1: Average Seasonality Plus Trend

To capture the average seasonality, Fourier terms were used as exogenous variables. A trend component was also included. The dependent variable is a Box-Cox tranformed Cumulative Case count, the lambda was chosen based on validation set forecast performance.

```
fit1.iq <- train %>%
  filter(city == "iq") %>%
  filter(cases_cumulative >= 10) %>%
  model(
    fit1 = TSLM(box_cox(cases_cumulative, .8) ~ fourier(K = 1) + trend())
  )
```

```
fit1.sj <- train %>%
  filter(city == "sj") %>%
  filter(cases_cumulative >= 10) %>%
  model(
    fit1 = TSLM(box_cox(cases_cumulative, 2.15) ~ fourier(K = 1) + trend())
  )
```

Model 2: ARIMAX

The modeling diverges between the two cities in the ARIMAX models; variables seemed to have differing predictive power depending on the city. As such, the modeling was separated and variables were selected to increase predictive power. One method that was employed was by regressing exogenous variables on the model innovations in an OLS model to find which were statistically significant. Those that are were then included to determine if they aided the forecast on the validation set, they were removed if not.

```
fit2.iq <- train %>%
  filter(city == "iq") %>%
  filter(cases_cumulative > 10) %>%
  model(
    fit2 = ARIMA(
      box_cox(cases_cumulative, .7) ~ 1 +
        fourier(K = 5) + PDQ(D=0,Q=0) +
        lag(precip_4w, n = 10) +
        lag(hdd_reanalysis_4w_sa * humidity_rel_avg_4w_sa, n = 6)
      + lag(station_diur_temp_rng_c, n = 5)
      + lag(hdd_reanalysis_365d_sa, n = 5)
      + lag(hdd_station_365d, n = 10)
      + lag(precip_365d_sa, n = 8)
    )
  )
```

```
fit2.sj <- train %>%
  filter(city == "sj") %>%
  filter(cases_cumulative > 100) %>%
  model(
    fit2a = ARIMA(
      box_cox(cases_cumulative, 1.3) ~ 1 +
        fourier(K = 2) + PDQ(P=0, D=0,Q=0)
      + lag(rollmean(reanalysis_min_air_temp_k, k = 8, NA, align = "right"), n = 8)
      + lag(hdd_reanalysis_4w, n = 9)
      + lag(reanalysis_tdtr_k_sa_smooth, n = 6)
      + population
    )
  )
```

Model 3: Prophet with Boosted Errors

The modeltime package was used for the Prophet with boosted errors model. The reasoning in selecting this model was to hopefully capture the nonlinearity and complicated interactions with the boosted trees.

This package follows the tidymodels approach by defining recipes and workflows. The following recipe is a set of data preprocessing steps.

```
(recipe_spec_iq <- train %>%
  as_tibble() %>%
```

```
  recipe(total_cases ~ ., data = .) %>%
  step_filter(city == "iq") %>%
  step_timeseries_signature(week_start_date) %>%
  step_rm(
    any_of(setdiff(!!vars.y, "total_cases"))
  ) %>%
  step_rm(
    any_of(setdiff(!!vars.id, c("weekofyear", "week_start_date")))
  ) %>%
  step_rm(contains("PC")) %>%
  step_rm(
    contains("am.pm"), contains("hour"), contains("minute"),
    contains("second"), contains("xts"), contains("day"), contains("susc")
  ) %>%
  step_fourier(week_start_date, period = 52, K = 5) %>%
  step_dummy(all_nominal_predictors()) %>%
  step_lag(all_numeric_predictors(), lag = 1:10) %>%
  step_normalize(all_numeric_predictors()) %>%
  step_diff(contains("_sa")) %>%
  step_naomit(all_predictors(), total_cases) %>%
  step_log(total_cases, offset = 1))
```

```
recipe_spec_iq %>% prep() %>% juice() %>% #select(week_start_date, contains("cases")) #%>%
  colnames() %>% sort()

recipe_spec_iq$var_info #%>% View()
```

The model is specified and a workflow set up.

```
model_spec_iq <- prophet_boost(mtry = tune(), tree_depth = tune(), learn_rate = tune(), mi
  set_engine("prophet_xgboost", yearly.seasonality = TRUE, weekly.seasonality = TRUE)

workflow_fit_proph_boost_iq <- workflow() %>%
  add_model(model_spec_iq) %>%
  add_recipe(recipe_spec_iq)
```

The model will take multiple values for the 'mtry', 'tree_depth', 'learn_rate', and 'min_n' parameters. A grid is set up to cross validate the data for each parameter combination.

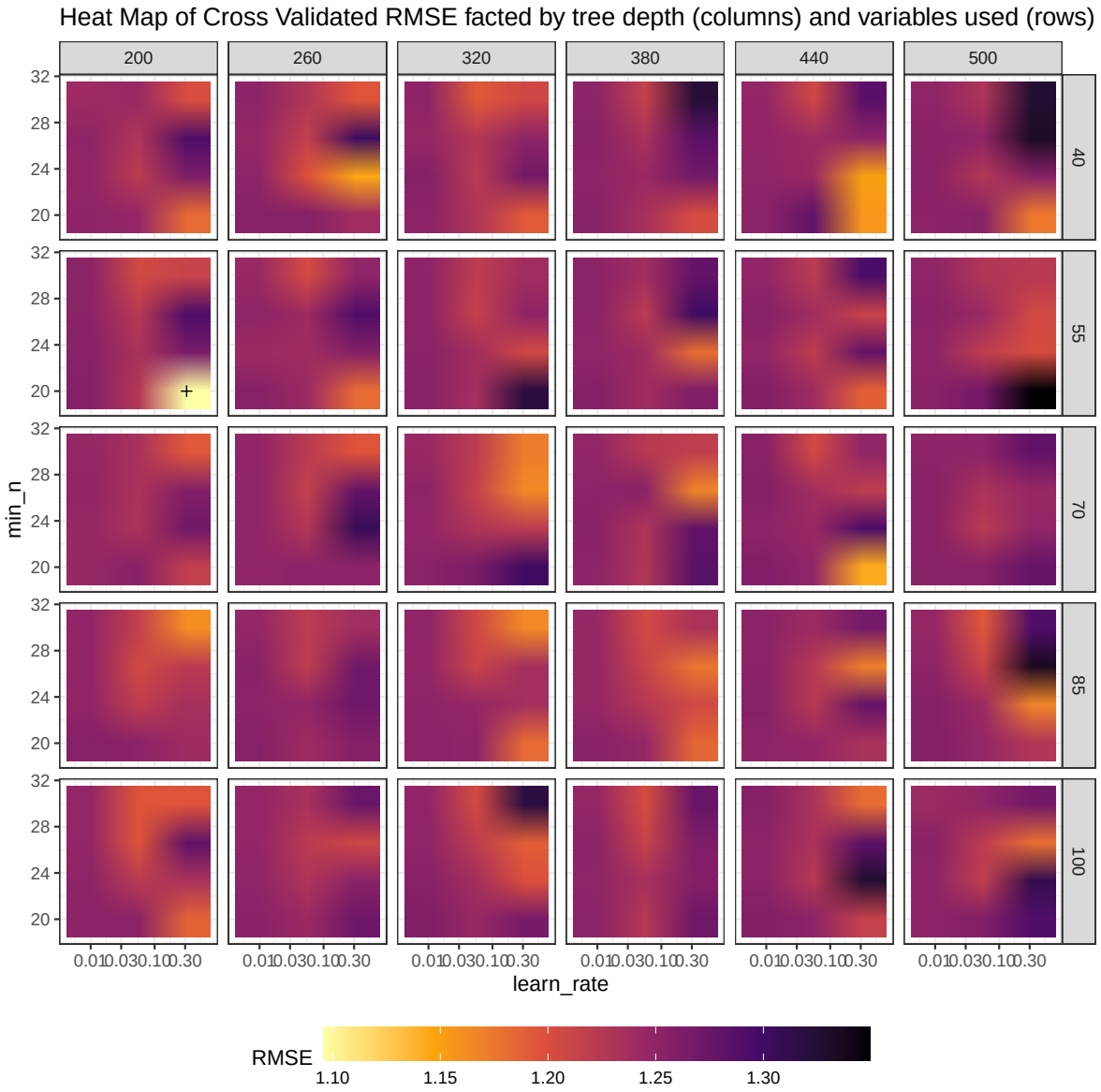
```
(tuning.grid_iq <- grid_regular(
  mtry(c(200, 500)),
```

```
  learn_rate(c(-2, -.5)),
  tree_depth(c(40, 100)),
  min_n(c(20,30)),
  levels = c(6, 3, 5, 4)
))

(folds_iq <- rolling_origin(
  train %>% filter(city == "iq"),
  cumulative = F,
  initial = 52*4, assess = 50, skip = 60
))

# Tune model
fit3.tuning_iq <- workflow_fit_proph_boost_iq %>%
  tune_grid(folds_iq, grid = tuning.grid_iq)
```

The follow faceted heatmap shows the average RMSE from the tuning parameters. The cross indicates the combination of parameters that minimizes the RMSE.

```
collect_metrics(fit3.tuning_iq) %>%
  filter(.metric == "rmse") %>%
  rename(RMSE = mean) %>%
  ggplot(aes(
    x = learn_rate,
    y = min_n,
    fill = RMSE
  )) +
  geom_raster(interpolate = T) +
  geom_point(
    data = select_best(fit3.tuning_iq, "rmse"),
    inherit.aes = F,
    aes(x = learn_rate, y = min_n),
    shape = 3
  ) +
  # geom_point(
  #   data = select_by_one_std_err(fit3.tuning_iq, "rmse"),
  #   inherit.aes = F,
  #   aes(x = learn_rate, y = min_n),
  #   shape = 4
  # ) +
  scale_x_log10() +
```

```
  facet_grid(tree_depth ~ mtry, scales = "free_x") +
  scale_y_continuous(labels = label_comma()) +
  scale_fill_viridis_c(option = "B", direction = -1) +
  ggtitle("Heat Map of Cross Validated RMSE facted by tree depth (columns) and variables u
  theme_bw() +
  theme(legend.position = "bottom", legend.key.width = unit(2, "cm"))
```



The parameters that minimize the RMSE are used to fit the entire training set.

```
(best.param_iq <- select_best(fit3.tuning_iq, "rmse"))
```

```
# A tibble: 1 x 5
   mtry min_n tree_depth learn_rate .config
  <int> <int>      <int>      <dbl> <chr>
1   200    20         55      0.316 Preprocessor1_Model031
```

```
wf.final_iq <- workflow_fit_proph_boost_iq %>%
  finalize_workflow(best.param_iq)
```

```
fit3.iq <- wf.final_iq %>%
  fit(train)
```

The same approach is taken for San Juan.

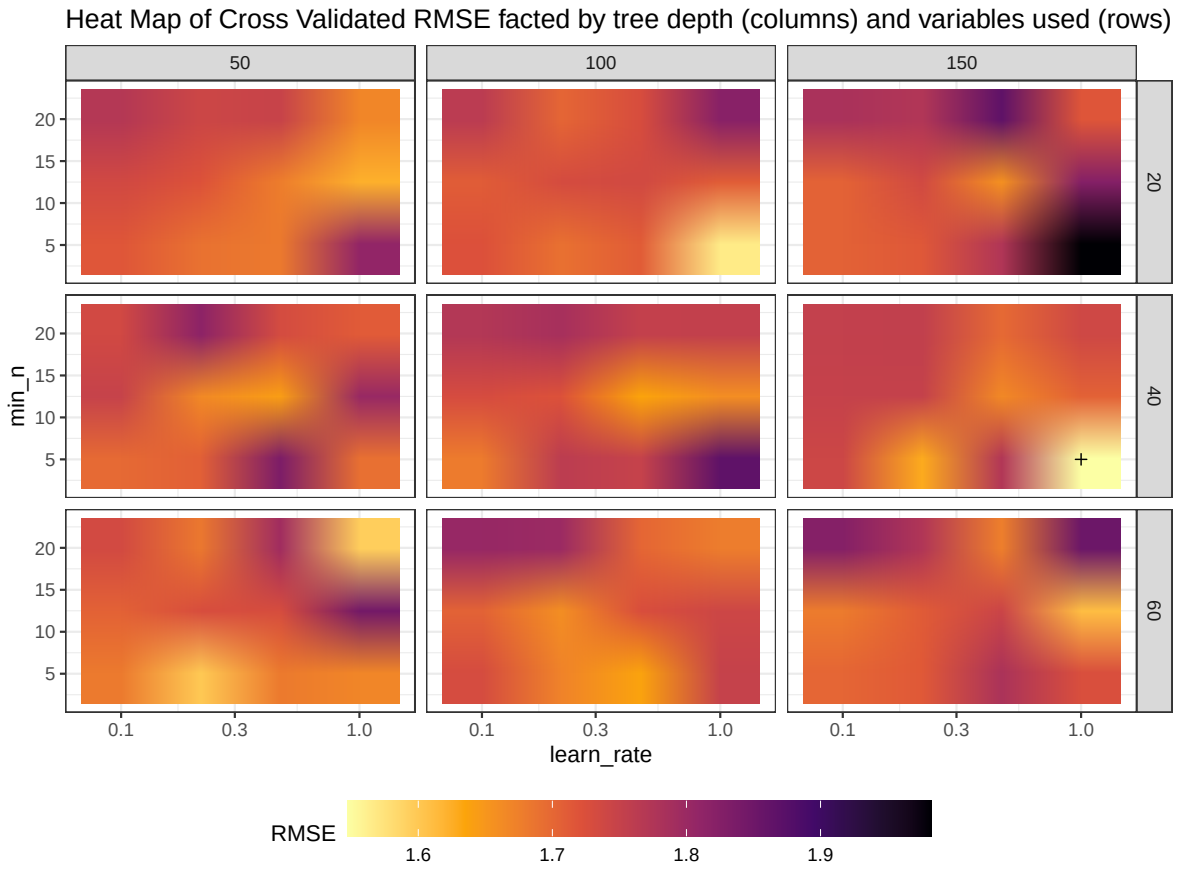
```
(recipe_spec_sj <- train %>%
  as_tibble() %>%
  filter(city == "sj") %>%
  recipe(case_rate ~ ., data = .) %>%
  step_timeseries_signature(week_start_date) %>%
  step_rm(
    any_of(setdiff(!!vars.y, "case_rate"))
  ) %>%
  step_rm(
    any_of(setdiff(!!vars.id, c("weekofyear", "week_start_date")))
  ) %>%
  step_rm(contains("PC")) %>%
  step_rm(
    contains("am.pm"), contains("hour"), contains("minute"),
    contains("second"), contains("xts"), contains("day"), contains("susc")
  ) %>%
  step_dummy(all_nominal_predictors()) %>%
  step_lag(all_numeric_predictors(), lag = 1:10) %>%
  step_normalize(all_numeric_predictors()) %>%
  step_diff(contains("_sa")) %>%
  step_naomit(all_predictors(), case_rate) %>%
  step_logit(case_rate, offset = .00001))
```

```r
model_spec_sj <- prophet_boost(mtry = tune(), tree_depth = tune(), learn_rate = tune(), mi
  set_engine("prophet_xgboost", yearly.seasonality = TRUE, weekly.seasonality = TRUE)

workflow_fit_proph_boost_sj <- workflow() %>%
  add_model(model_spec_sj) %>%
  add_recipe(recipe_spec_sj)
```

```r
(tuning.grid_sj <- grid_regular(
    mtry(c(50, 150)),
    learn_rate(c(-1, 0)),
    tree_depth(c(20, 60)),
    min_n(c(5,20)),
    # levels = c(2,2,1,1)
    levels = c(3, 4, 3, 3)
  ))

(folds_sj <- rolling_origin(
  train %>% filter(city == "sj"),
  cumulative = F,
  initial = 52*5, assess = 100, skip = 100
))

fit3.tuning_sj <- workflow_fit_proph_boost_sj %>%
  tune_grid(folds_sj, grid = tuning.grid_sj)
```

```r
collect_metrics(fit3.tuning_sj) %>%
  filter(.metric == "rmse") %>%
  rename(RMSE = mean) %>%
  ggplot(aes(
    x = learn_rate,
    y = min_n,
    fill = RMSE
  )) +
  geom_raster(interpolate = T) +
  geom_point(
    data = select_best(fit3.tuning_sj, "rmse"),
    inherit.aes = F,
    aes(x = learn_rate, y = min_n),
    shape = 3
  ) +
```

```
# geom_point(
#   data = select_by_one_std_err(fit3.tuning_sj, "rmse"),
#   inherit.aes = F,
#   aes(x = learn_rate, y = min_n),
#   shape = 4
# ) +
scale_x_log10() +
facet_grid(tree_depth ~ mtry, scales = "free_x") +
scale_y_continuous(labels = label_comma()) +
scale_fill_viridis_c(option = "B", direction = -1) +
ggtitle("Heat Map of Cross Validated RMSE facted by tree depth (columns) and variables u
theme_bw() +
theme(legend.position = "bottom", legend.key.width = unit(2, "cm"))
```




```
(best.param_sj <- select_best(fit3.tuning_sj, "rmse"))
```

```
# A tibble: 1 x 5
   mtry min_n tree_depth learn_rate .config
  <int> <int>      <int>      <dbl> <chr>
1   150     5         40          1 Preprocessor1_Model024
```

```
wf.final_sj <- workflow_fit_proph_boost_sj %>%
  finalize_workflow(best.param_sj)
```

```
fit3.sj <- wf.final_sj %>%
  fit(train)
```

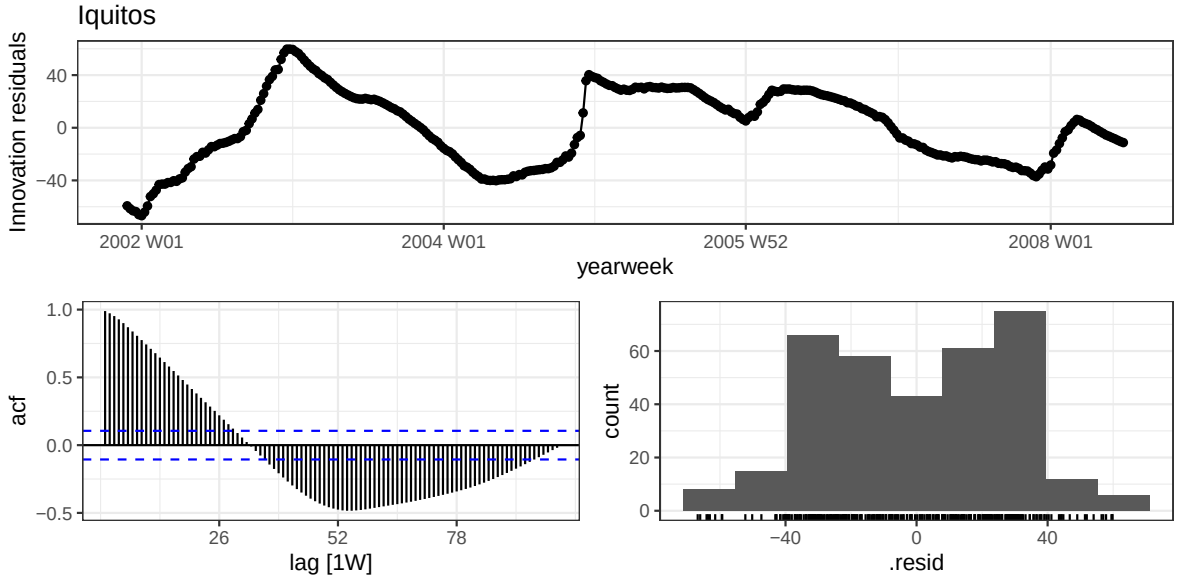
Results

Training Set

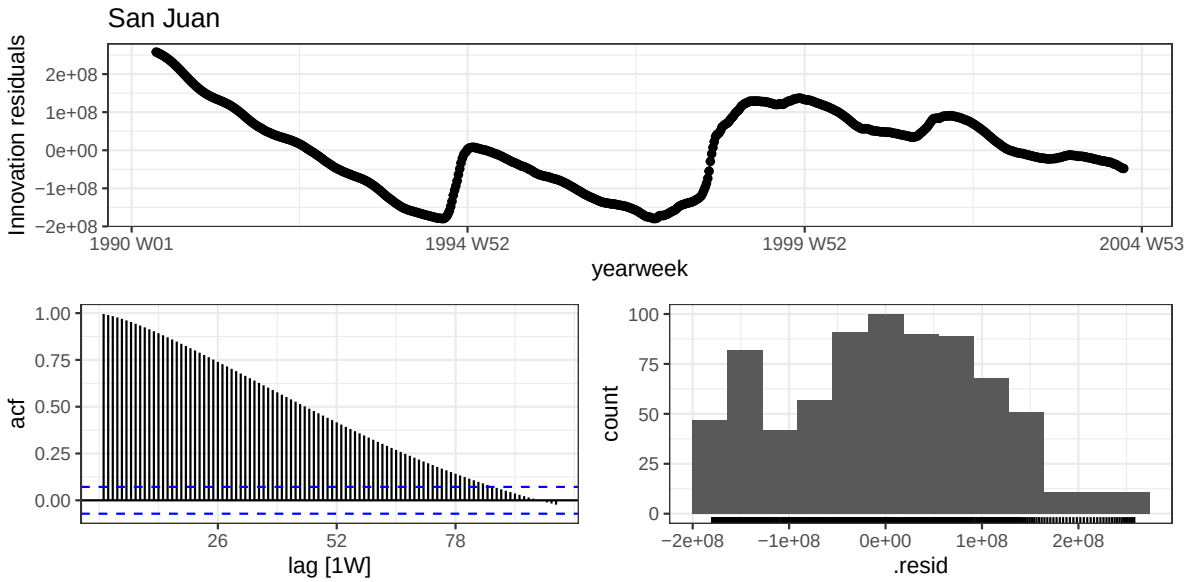
Model 1

The simple Average Seasonality + Trend model leaves a lot to be desired. Both cities remain highly autocorrelated with non-normal residuals.

```
fit1.iq %>%
  gg_tsresiduals(lag_max = 100) +
  ggtitle("Iquitos")
```



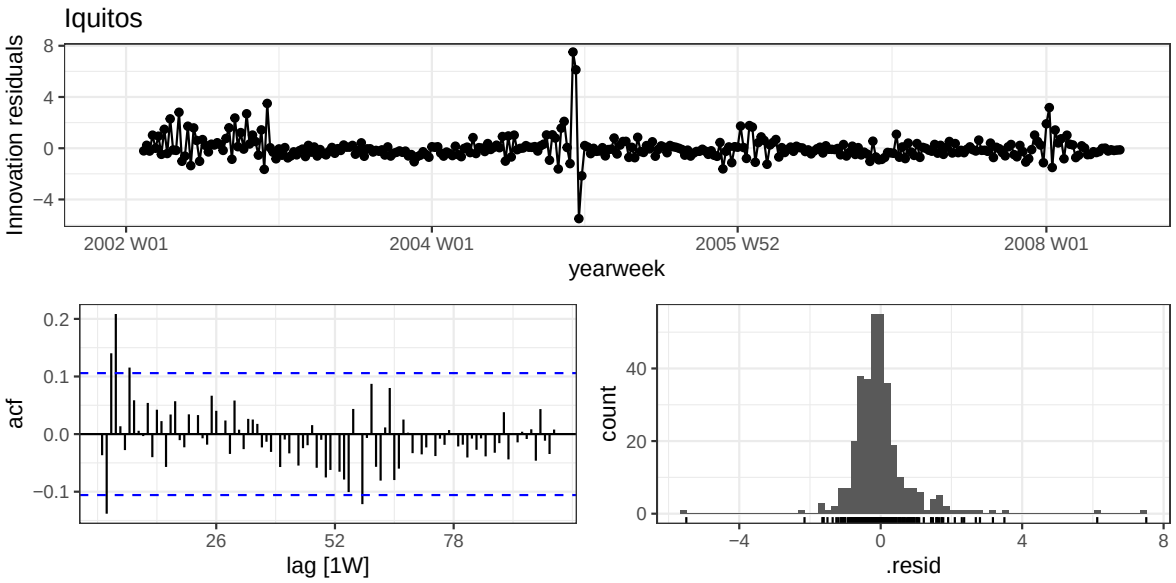
```
fit1.sj %>%
  gg_tsresiduals(lag_max = 100) +
  ggtitle("San Juan")
```



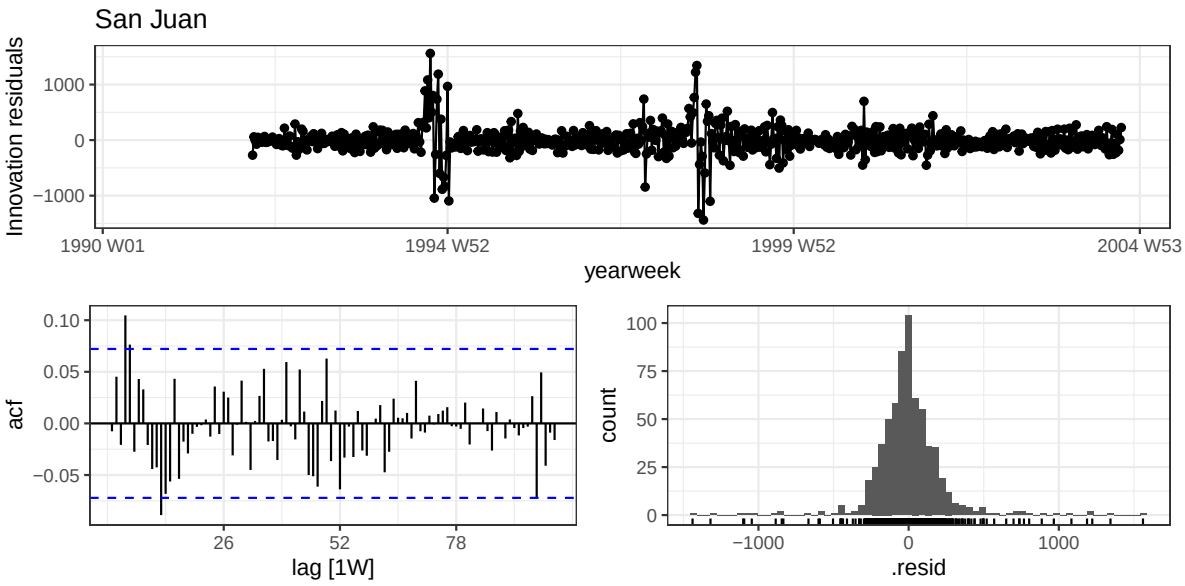
Model 2

The ARIMAX models improve the residuals tremendously, but still are far from perfect. They both appear to struggle with modeling periods of steep growth, likely due to the exponential nature of it. Some autocorrelation remains, but is still much better.

```
fit2.iq %>%
  gg_tsresiduals(lag_max = 100) +
  ggtitle("Iquitos")
```




```
fit2.sj %>%
  gg_tsresiduals(lag_max = 100) +
  ggtitle("San Juan")
```

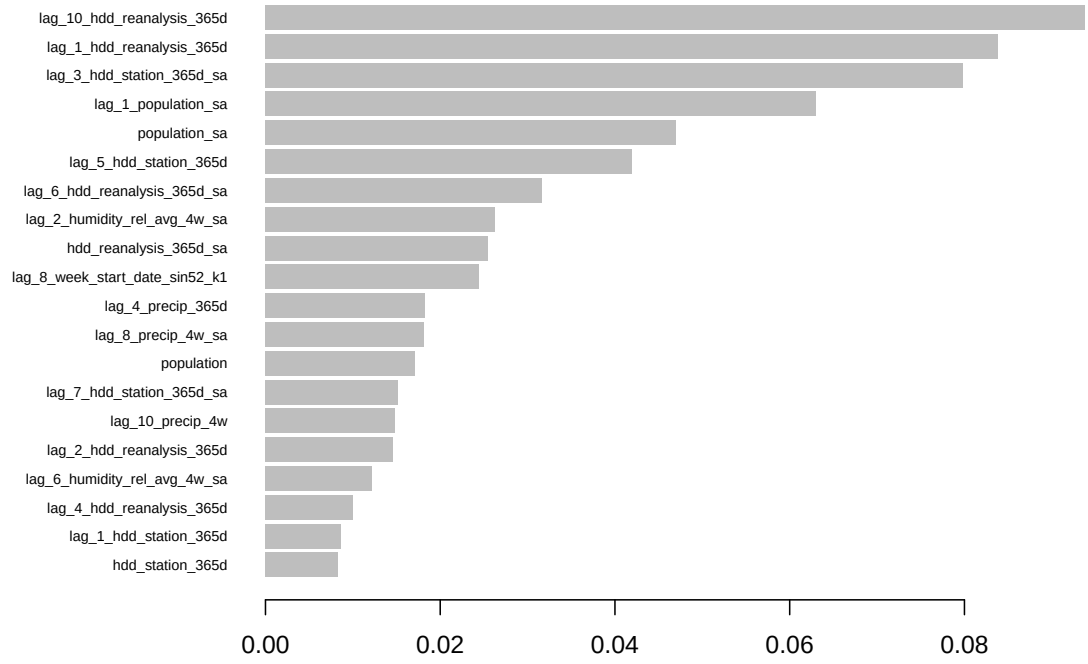


Model 3

Another advantage of the xgboost algorithm is the ability to view a variable importance plot. While it doesn't indicate how a variable is correlated with cases, it does indicate which variables have some correlation, linear or nonlinear. Both models found differing lags of GDD, precipitation, and humidity to be important variables.

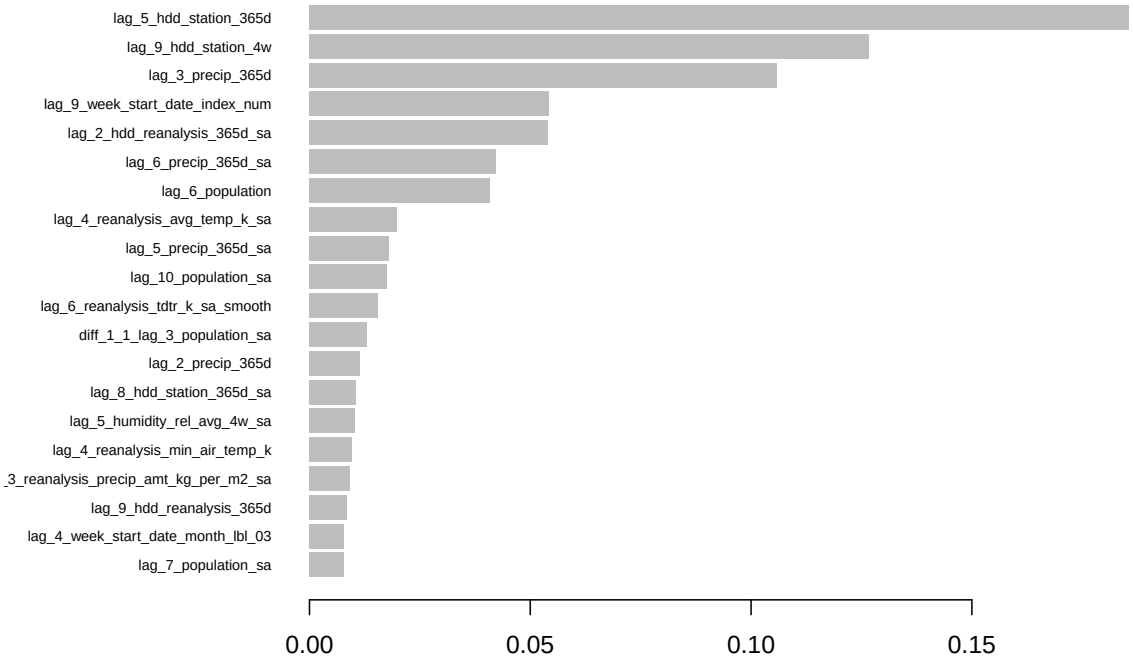
```
fit3.orig.iq <- extract_fit_engine(fit3.iq)

xgboost::xgb.importance(model = fit3.orig.iq$models$model_2) %>%
  xgboost::xgb.plot.importance(top_n = 20)
```

```
[17:39:07] WARNING: src/learner.cc:553:
  If you are loading a serialized model (like pickle in Python, RDS in R) generated by
  older XGBoost, please export the model by calling `Booster.save_model` from that version
  first, then load it back in current version. See:

    https://xgboost.readthedocs.io/en/latest/tutorials/saving_model.html

  for more details about differences between saving model and serializing.
```




```r
fit3.orig.sj <- extract_fit_engine(fit3.sj)

xgboost::xgb.importance(model = fit3.orig.sj$models$model_2) %>%
  xgboost::xgb.plot.importance(top_n = 20)
```

```
[17:39:07] WARNING: src/learner.cc:553:
  If you are loading a serialized model (like pickle in Python, RDS in R) generated by
  older XGBoost, please export the model by calling `Booster.save_model` from that version
  first, then load it back in current version. See:

    https://xgboost.readthedocs.io/en/latest/tutorials/saving_model.html

  for more details about differences between saving model and serializing.
```



Model Performance on Validation Set

Each of the models will forecast the validation set to compare their predictive power.

```
fx1.iq <- fn_standardize_fx(forecast(fit1.iq, valid), the_model = 1, the_city = "iq")
fx2.iq <- fn_standardize_fx(forecast(fit2.iq, valid), the_model = 2, the_city = "iq")

calibration_table_iq <- modeltime_table(fit3.iq) %>%
  modeltime_calibrate(valid)

fx3.iq_trans <- calibration_table_iq %>%
  modeltime_forecast(
    # new_data = as_tibble(train.all) %>% filter(city == "iq", year >= 2008)
    new_data = as_tibble(valid) %>% filter(city == "iq")
  )
```

```
fx3.iq <- fn_standardize_fx(fx3.iq_trans, the_model = 3, the_city = "iq", the_smooth = 5)
```

```
fx1.sj <- fn_standardize_fx(forecast(fit1.sj, valid), the_model = 1, the_city = "sj")
fx2.sj <- fn_standardize_fx(forecast(fit2.sj, valid), the_model = 2, the_city = "sj")

calibration_table_sj <- modeltime_table(fit3.sj) %>%
  modeltime_calibrate(valid %>% filter(city == "sj"))

fx3.sj_trans <- calibration_table_sj %>%
  modeltime_forecast(
    # new_data = as_tibble(train.all) %>% filter(city == "sj", year >= year(ymd(valid.star
    new_data = as_tibble(valid) %>% filter(city == "sj")
  )

fx3.sj <- fn_standardize_fx(fx3.sj_trans, the_model = 3, the_city = "sj", the_smooth = 5)
```

An ensemble forecast will also be calculated as the simple average of the three models. This could prove to be a reliable forecast, especially since each of the three models approach the forecast quite differently.

```
#iq
fx.iq <- bind_rows(
  fx1.iq,
  fx2.iq,
  fx3.iq
)

fx.iq_avg <- fx.iq %>% #filter(.model != "fit1" & .model != "ensemble") %>% bind_rows(fx1.
  as_tibble() %>%
  group_by(city, yearweek) %>%
  summarize(across(contains("predicted"), \(x){mean(x, na.rm = T)})) %>%
  arrange(yearweek) %>%
  mutate(.model = "ensemble") %>%
  tsibble(index = yearweek, key = c(city, .model))

fx.iq <- bind_rows(
  fx.iq,
  fx.iq_avg
)
```

```r
# sj
fx.sj <- bind_rows(
  fx1.sj,
  fx2.sj,
  fx3.sj
)

fx.sj_avg <- fx.sj %>% #filter(.model != "fit1" & .model != "ensemble") %>% bind_rows(fx1.
  as_tibble() %>%
  group_by(city, yearweek) %>%
  summarize(across(contains("predicted"), \(x){mean(x, na.rm = T)})) %>%
  arrange(yearweek) %>%
  mutate(.model = "ensemble") %>%
  tsibble(index = yearweek, key = c(city, .model))

fx.sj <- bind_rows(
  fx.sj,
  fx.sj_avg
)
```

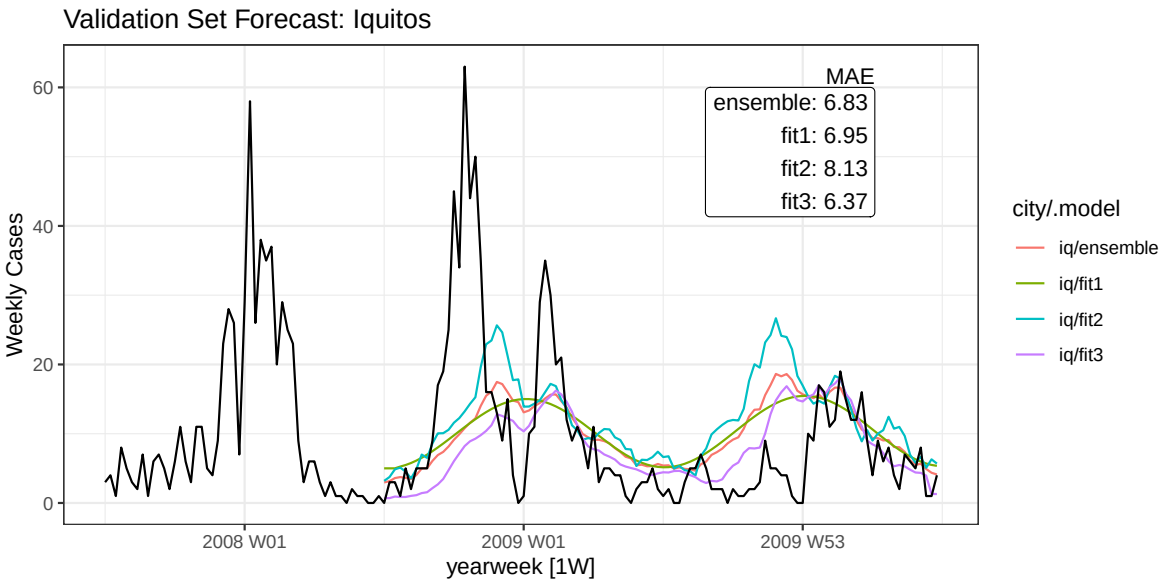
Comparison

Iquitos has a lower mean absolute error than San Juan, however, this is likely a result of having an average of 7.54 weekly cases to San Juan's average of 34.1 during the training set. Moderately surprising, the Seasonal Average approach (Model 1) is very comparable to the other two models. This possibly suggests the models using exogenous factors are not optimal and not getting much insight from them. The Boosted Prophet model performs the best for Iquitos yet the worst for San Juan.

The best model likely is the ensemble model as it's only slightly edged out for Iquitos and performs best for San Juan.

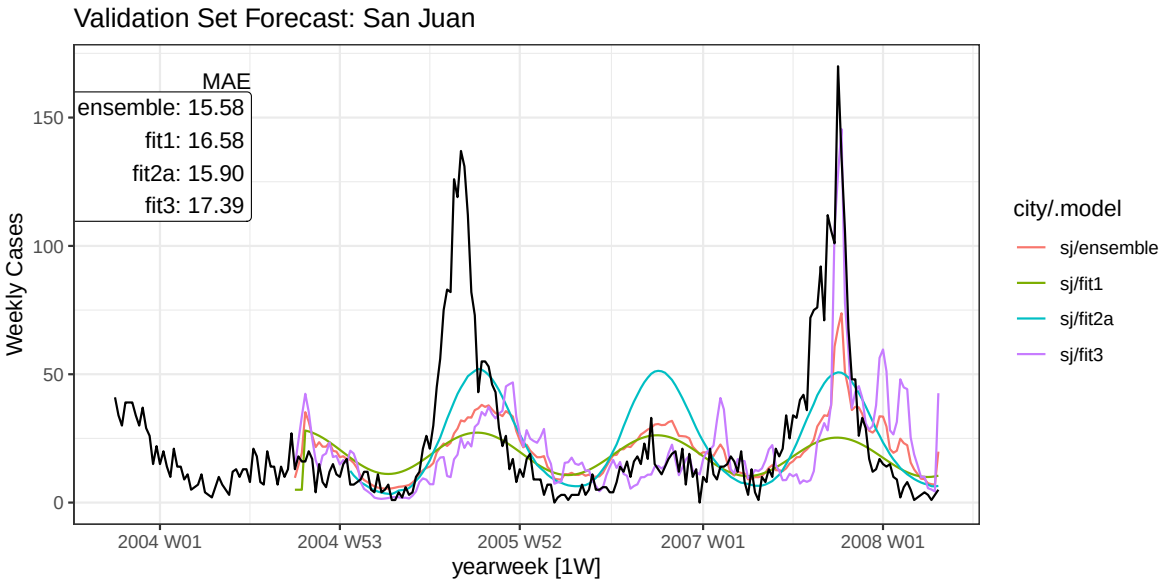
```r
mae.valid <- bind_rows(fx.iq, fx.sj) %>%
  left_join(y = train.all %>% select(city, yearweek, total_cases)) %>%
  as_tibble() %>%
  group_by(city, .model) %>%
  summarize(mae = MAE(predicted_cases - total_cases)) %>%
  ungroup()


fx.iq %>%
```

```
  autoplot(predicted_cases) +
  autolayer(
    train.all %>% filter(city == "iq") %>% filter(week_start_date >= ymd(valid.start.iq) -
    total_cases
  ) +
  ylab("Weekly Cases") +
  ylab("Weekly Cases") +
  annotate(
    "text", x = 14700, y = 60.5,
    label = "MAE", vjust = 0, hjust = 1
  ) +
  annotate(
    "label", x = 14700, y = 60,
    label = mae.valid %>% filter(city == "iq") %>% select(-city) %>%
      apply(., 1, \(x){paste0(x[1], ": ", label_comma(accuracy = .01)(as.numeric(x[2])))}) %>%
      paste(collapse = "\n"),
    vjust = 1, hjust = 1
  ) +
  ggtitle("Validation Set Forecast: Iquitos")
```




```
fx.sj %>% #filter(!(.model == "fit1" & city == "sj")) %>% bind_rows(fx1.sj) %>%
  autoplot(predicted_cases) +
  autolayer(
```

```
    train.all %>% filter(city == "sj") %>% filter(week_start_date >= ymd(valid.start.sj) -
    total_cases
  ) +
  ylab("Weekly Cases") +
  annotate(
    "text", x = 12600, y = 161,
    label = "MAE", vjust = 0, hjust = 1
  ) +
  annotate(
    "label", x = 12600, y = 160,
    label = mae.valid %>% filter(city == "sj") %>% select(-city) %>%
      apply(., 1, \(x){paste0(x[1], ": ", label_comma(accuracy = .01)(as.numeric(x[2])))}) %>%
      paste(collapse = "\n"),
    vjust = 1, hjust = 1
  ) +
  ggtitle("Validation Set Forecast: San Juan")
```




```
mae.valid %>%
  mutate(.model = ifelse(.model == "fit2a", "fit2", .model)) %>%
  pivot_wider(names_from = city, values_from = mae) %>%
  mutate(
    model_description = case_when(
      .model == "ensemble" ~ "Simple average of other models",
```

```
      .model == "fit1" ~ "Average seasonality plus trend",
      .model == "fit2" ~ "ARIMAX",
      .model == "fit3" ~ "Prophet with Boosted Trees",
    )
  ) %>%
  relocate(.model, model_description)
```

```
# A tibble: 4 x 4
  .model   model_description                iq    sj
  <chr>    <chr>                        <dbl> <dbl>
1 ensemble Simple average of other models  6.83  15.6
2 fit1     Average seasonality plus trend  6.95  16.6
3 fit2     ARIMAX                          8.13  15.9
4 fit3     Prophet with Boosted Trees      6.37  17.4
```

Final Fit

All models are then refit over the entire pre-test set data, both training and validation sets, before making their final predictions.

```
fit1.iq_final <- fit1.iq %>%
  refit(train.all %>% filter(city == "iq"))

fit2.iq_final <- fit2.iq %>%
  refit(train.all %>% filter(city == "iq"))

fit3.iq_final <- wf.final_iq %>%
  fit(train.all %>% filter(city == "iq"))


fit1.sj_final <- fit1.sj %>%
  refit(train.all %>% filter(city == "sj"))

fit2.sj_final <- fit2.sj %>%
  refit(train.all %>% filter(city == "sj"))

fit3.sj_final <- wf.final_sj %>%
  fit(train.all %>% filter(city == "sj"))
```

Forecast on Test Set

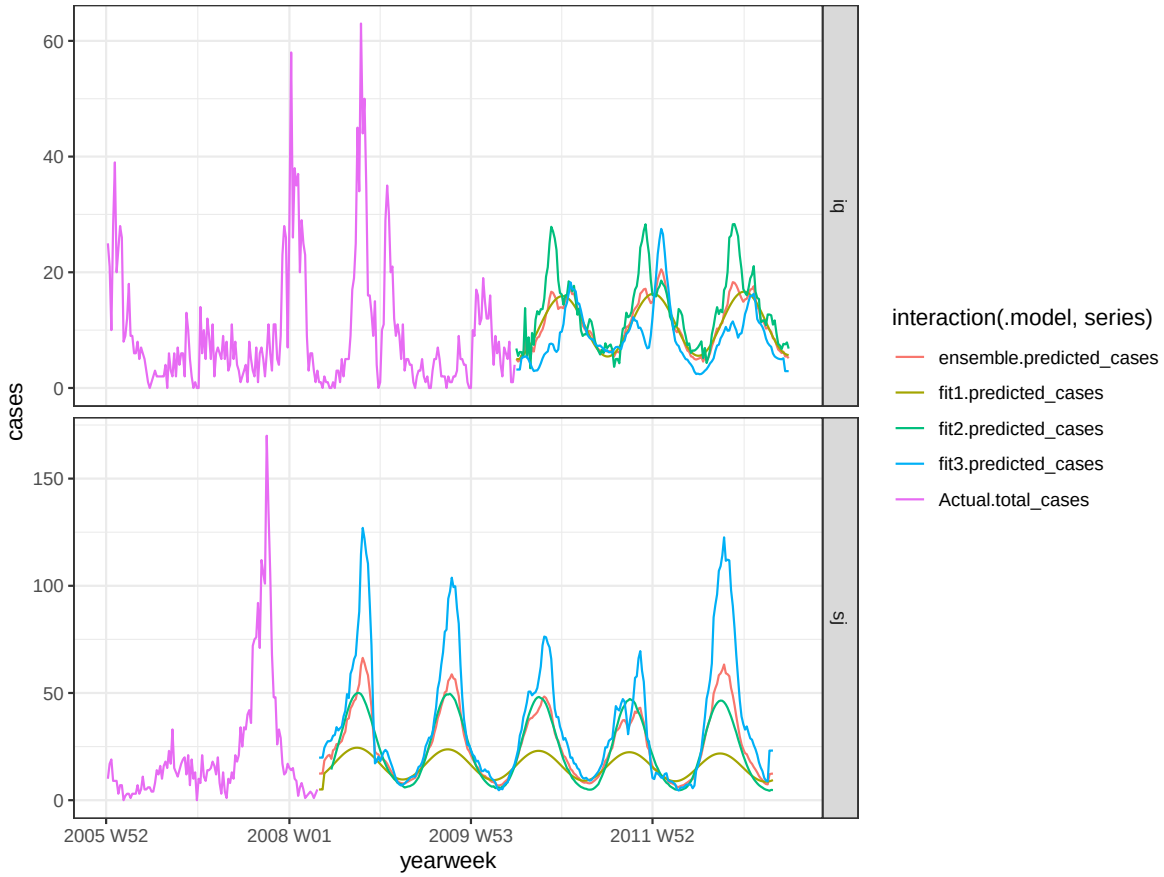
```r
fx1.test.iq <- fit1.iq_final %>%
  forecast(test) %>%
  fn_standardize_fx(., 1, "iq")

fx2.test.iq <- fit2.iq_final %>%
  forecast(test) %>%
  fn_standardize_fx(., 2, "iq")

fx3.test.iq <- modeltime_table(fit3.iq_final) %>%
  modeltime_calibrate(test) %>%
  modeltime_forecast(
    new_data = as_tibble(test) %>% filter(city == "iq")
  ) %>%
  fn_standardize_fx(., 3, "iq", 7)


fx.test.iq <- bind_rows(fx1.test.iq, fx2.test.iq, fx3.test.iq)

fx.test.iq_avg <- fx.test.iq %>%
  as_tibble() %>%
  group_by(city, yearweek) %>%
  summarize(across(contains("predicted"), \(x){mean(x, na.rm = T)})) %>%
  arrange(yearweek) %>%
  mutate(.model = "ensemble") %>%
  tsibble(index = yearweek, key = c(city, .model))

fx.test.iq <- bind_rows(fx.test.iq, fx.test.iq_avg)
```

```r
fx1.test.sj <- fit1.sj_final %>%
  forecast(test) %>%
  fn_standardize_fx(., 1, "sj")

fx2.test.sj <- fit2.sj_final %>%
  forecast(test) %>%
  fn_standardize_fx(., 2, "sj")

fx3.test.sj <- modeltime_table(fit3.sj_final) %>%
  modeltime_calibrate(test) %>%
  modeltime_forecast(
    new_data = as_tibble(test) %>% filter(city == "sj")
  ) %>%
```

```
  fn_standardize_fx(., 3, "sj", 7)


fx.test.sj <- bind_rows(fx1.test.sj, fx2.test.sj, fx3.test.sj)

fx.test.sj_avg <- fx.test.sj %>%
  as_tibble() %>%
  group_by(city, yearweek) %>%
  summarize(across(contains("predicted"), \(x){mean(x, na.rm = T)})) %>%
  arrange(yearweek) %>%
  mutate(.model = "ensemble") %>%
  tsibble(index = yearweek, key = c(city, .model))

fx.test.sj <- bind_rows(fx.test.sj, fx.test.sj_avg)
```

The competition submissions can be found under the user account shaun17. The Boosted Prophet model was the best performing at 22.76 MAE, followed by the ensemble at 24.50 MAE, the ARIMAX at 25.81, and the Average Seasonality model at 30.91 MAE.

```
bind_rows(
  fx_test,
  train.all %>% select(city, yearweek, total_cases)
) %>%
  as_tibble() %>%
  replace_na(list(.model = "Actual")) %>%
  filter(year(yearweek) >= 2006) %>%
  pivot_longer(c(predicted_cases, total_cases), names_to = "series", values_to = "cases") 
  filter(!is.na(cases)) %>%
  ggplot(aes(x = yearweek, y = cases, color = interaction(.model, series))) +
  geom_line() +
  facet_grid(city ~ ., scales = "free")
```



Note: Currently at the daily submission limit as of time of writing. Model 1 was updated and reduced the validation set MAE by .46 and 1.91 cases for Iquitos and San Juan, respectively. This in turn reduced the MAE for the ensemble model by .06 and .45 cases for Iquitos and San Juan, respectively.

Conclusion

The models struggled to truly capture the peaks on the validation set, though they appeared to capture the far-less-volatile shoulder months. The likely culprit is using linear models on exponential data without sufficiently transforming it. The boosted Prophet model was the only one able to semi-accurately predict a peak in validation and the predicted peaks in the test set are not identical. More fine-tuning and tweaking of the linear models will improve the ensemble even more and that should likely be the forecast most relied on. Another model that should be considered is a generalized linear model of the negative binomial family which should be better fit to handle exponentially rising cases.

The biggest limitation on these models is that they are location-specific. Different variables and parameters were used for each city making it difficult to generalize these to other locations. While a generalized approach could mean the same data could be used to make predictions elsewhere, we are constrained to only the methodology: new models must be estimated on that locations' data to forecast.

References

Benedum, Jenkins, Shea. 2020. "Weekly Dengue Forecasts in Iquitos, Peru; San Juan, Puerto Rico; and Singapore." *PLoS Neglected Tropical Diseases* 14 (10): e0008710. https://doi.org/https://doi.org/10.1371/journal.pntd.0008710.

Chen, Tianqi, Tong He, Michael Benesty, Vadim Khotilovich, Yuan Tang, Hyunsu Cho, Kailong Chen, et al. 2023. *Xgboost: Extreme Gradient Boosting*. https://github.com/dmlc/xgboost.

Dancho, Matt. 2023. *Modeltime: The Tidymodels Extension for Time Series Modeling*. https://CRAN.R-project.org/package=modeltime.

Dancho, Matt, and Davis Vaughan. 2023. *Timetk: A Tool Kit for Working with Time Series*. https://CRAN.R-project.org/package=timetk.

Hyndman, Rob. 2023. *Fpp3: Data for "Forecasting: Principles and Practice" (3rd Edition)*. https://CRAN.R-project.org/package=fpp3.

Kuhn, Max, and Hadley Wickham. 2020. *Tidymodels: A Collection of Packages for Modeling and Machine Learning Using Tidyverse Principles*. https://www.tidymodels.org.

———. 2022. *Tidymodels: Easily Install and Load the Tidymodels Packages*. https://CRAN.R-project.org/package=tidymodels.

NRCC. 2023. "Climate Information for Mosquito Control & Public Health Officials, Degree Days." http://www.nrcc.cornell.edu/industry/mosquito/degreedays.html.

R Core Team. 2019. *R: A Language and Environment for Statistical Computing*. Vienna, Austria: R Foundation for Statistical Computing. https://www.R-project.org.

Wickham, Hadley. 2016. *Ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York. https://ggplot2.tidyverse.org.

———. 2023. *Tidyverse: Easily Install and Load the Tidyverse*. https://CRAN.R-project.org/package=tidyverse.

Wickham, Hadley, Mara Averick, Jennifer Bryan, Winston Chang, Lucy D'Agostino McGowan, Romain François, Garrett Grolemund, et al. 2019. "Welcome to the tidyverse." *Journal of Open Source Software* 4 (43): 1686. https://doi.org/10.21105/joss.01686.

Wickham, Hadley, Winston Chang, Lionel Henry, Thomas Lin Pedersen, Kohske Takahashi, Claus Wilke, Kara Woo, Hiroaki Yutani, and Dewey Dunnington. 2023. *Ggplot2: Create Elegant Data Visualisations Using the Grammar of Graphics*. https://CRAN.R-project.org/package=ggplot2.