- 1. Recap
- 2 Tune again
    - Modelling
    - Retuning
    - 2.1 Retune Random Forest
    - 2.2 Retune Prophet boost
    - 2.3 Performance (after retuning)
- 3 Ensemble
    - 3.1 Peformance (ensemble)
- 4 Performance (individual levels)
    - Hospital
    - Cluster level
    - National level
- 5 The future
    - Hospital
    - Cluster
    - National
- 6 KIV Plans
    - Errors

1. Recap

The aim of this series of blog is to predict monthly admissions to Singapore public acute adult hospitals. EDA for the dataset was explored in past posts (part 1 ; part 2). The admissions were treated as a hierarchical time series as every country has a hierarchical order to its public hospitals including Singapore. The levels are:

National level
  |– Cluster level (Clusters are a network of hospitals based on geographical regions. There are 3 health clusters in Singapore.)
    |– Hospital level (There are 8 public acute adult hospitals.)

Admissions were forecasted at each level for Mar 21- Dec 21

```
library(tidyverse)
library(tidymodels)
library(timetk)
library(modeltime)
library(modeltime.ensemble)

# dataset (training data and future forecast data and pre-processing
recipes were done in the past post. cv too) The output was uploaded
onto my github.
url_datasets<-url("https://github.com/notast/hierarchical-forecasting/blob/
main/5Data_CV.RData?raw=true")
load(url_datasets)
close(url_datasets)

to_predictfuture %>% distinct(Date)
## # A tibble: 10 x 1
```

```
##    Date
##    <date>
##  1 2021-03-01
##  2 2021-04-01
##  3 2021-05-01
##  4 2021-06-01
##  5 2021-07-01
##  6 2021-08-01
##  7 2021-09-01
##  8 2021-10-01
##  9 2021-11-01
## 10 2021-12-01
```

The training dataset was from Jan 16 – Apr 20.

```
training(splits) %>% distinct(Date) %>% tail(10)
## # A tibble: 10 x 1
##    Date
##    <date>
##  1 2019-07-01
##  2 2019-08-01
##  3 2019-09-01
##  4 2019-10-01
##  5 2019-11-01
##  6 2019-12-01
##  7 2020-01-01
##  8 2020-02-01
##  9 2020-03-01
## 10 2020-04-01
```

The testing dataset was from May 20- Feb 21.

```
testing(splits) %>% distinct(Date) %>% tail(10)
## # A tibble: 10 x 1
##    Date
##    <date>
##  1 2020-05-01
##  2 2020-06-01
##  3 2020-07-01
##  4 2020-08-01
##  5 2020-09-01
##  6 2020-10-01
##  7 2020-11-01
##  8 2020-12-01
##  9 2021-01-01
## 10 2021-02-01
```

The admissions were forecasted using:

1. Classical Approach:
    ii. Traditional techniques e.g. bottoms up

ii. Newer techniques e.g. reconciliation were employed

2. Machine Learning Approach
    i. The best combination of features to be used for machine learning were screened
    ii. Several machine learning techniques were trial
    iii. In this post, the best models (Random Forest and Prophet Boost) were retuned and ensemble model was created to achieve better performance

```
# tune grid and wf
url_TgWf<- url("https://github.com/notast/hierarchical-forecasting/blob/main/5Retunning_objects.RData?raw=true")
load(url_TgWf)
close(url_TgWf)

# finalized and fitted wf
url_ffwf<- url("https://github.com/notast/hierarchical-forecasting/blob/main/5FFWf.RData?raw=true")
load(url_ffwf)
close(url_ffwf)

# metric
metrics_custom= metric_set( rmse, mae)
```

2 Tune again

Modelling and retuning is easy with `tidymodels`

Modelling

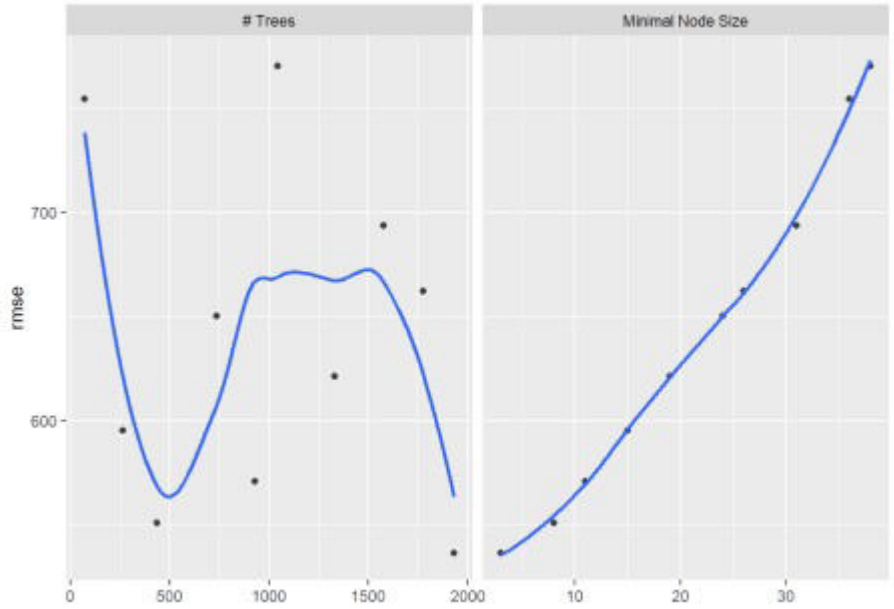
i. Set up the model
ii. Add the recipe and model into a workflow
iii. Tune the workflow which in turn tunes parameters of the recipe and/or the model inside the workflow
iv. Finalize the workflow with the best tuned parameters
v. Fit the finalized workflow with its best tuned recipe and best tuned model onto the whole training data.

Retuning

vi. Plot the current parameters against its corresponding `rmse`
vii. Identify a range of parameters which correspond to better `rmse` and create an updated set of parameter information.
viii. Retune the workflow with the updated set of parameter information.
ix. Finalized the updated workflow with the best retuned parameters.
x. Fit the new finalized workflow onto the whole training data.

2.1 Retune Random Forest

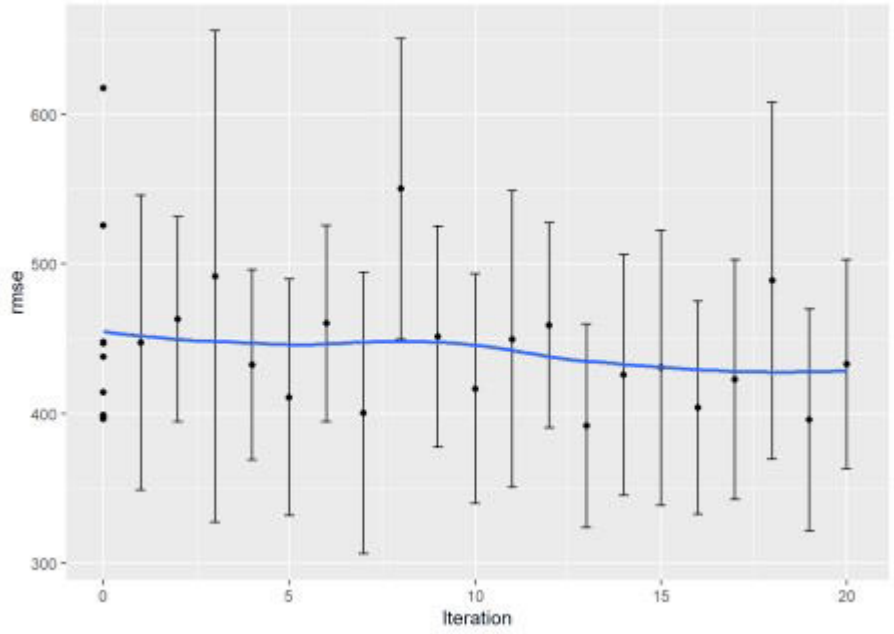
```
rf_t %>% autoplot(metric="rmse") + geom_smooth(se=F)
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



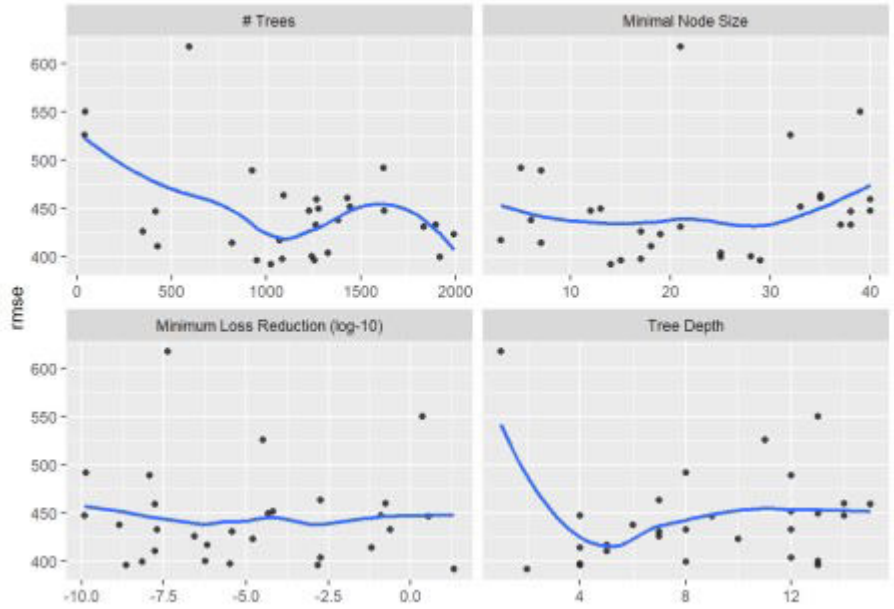

```
param_rf2<- rf_wf %>% parameters() %>%
  update(trees=dials::trees(range=c(1600, 2500)),
         min_n= dials::min_n(range(2, 9)))

set.seed(69)
rf2_t<- tune_grid(
  object= rf_wf,
  resamples = folds,
  param_info = param_rf2,
  grid=20,
  metrics = metrics_custom)

fun_fwf<- function(wf, t_wf){
    finalize_workflow(
                    x= wf,
                    parameters= select_best(t_wf, "rmse")) %>%
    fit(splits_train)
    }

rf2_f<-fun_fwf(rf_wf, rf2_t)
```

2.2 Retune Prophet boost

```
pb_t %>% autoplot(metric="rmse", "performance") + geom_smooth(se=F)
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```




```
pb_t %>% autoplot(metric="rmse") + geom_smooth(se=F)
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```




```
param_pb2<- pb_wf %>% parameters() %>%
  update(min_n= dials::min_n(range=c(10,30)),
         tree_depth= dials::tree_depth(range=c(3,10)))

all_cores <- parallel::detectCores(logical = FALSE)
library(doParallel)
cl <- makePSOCKcluster(all_cores)
registerDoParallel(cl)

set.seed(69)
pb2_t<- tune_bayes(
    object= pb_wf,
    resamples = folds,
    iter = 25, initial = 9,
```

```
    param_info = param_pb2,
    metrics = metrics_custom,
    control = control_bayes(no_improve = 25))

pb2_f<- fun_fwf(pb_wf, pb2_t)

# save retuned finalized and fitted wf
save(rf2_f, pb2_f,  file = "6Retuned_FinalFitWf.RData")
```

2.3 Performance (after retuning)

Both Prophet Boost and Random Forest benefited from retuning. Prophet Boost benefited the most as XGB performs more optimally when its parameters are tweak.

```
e2_table<-modeltime_table(rf_f, pb_f,rf2_f, pb2_f) %>%
  update_model_description(1, "RF") %>% update_model_description(2,
"PB") %>%
  update_model_description(3, "RF_retuned") %>%
update_model_description(4, "PB_retuned")
e2_cal<-e2_table %>% modeltime_calibrate(testing(splits))
e2_cal %>% modeltime_accuracy(metric_set = metrics_custom) %>%
arrange(rmse, sort=T)
## # A tibble: 4 x 5
##   .model_id .model_desc .type  rmse   mae
##       <int> <chr>       <chr> <dbl> <dbl>
## 1         3 RF_retuned  Test   545.  409.
## 2         1 RF          Test   549.  410.
## 3         4 PB_retuned  Test   945.  673.
## 4         2 PB          Test  1137.  799.
```

3 Ensemble

Both tuned and original Random Forests were trial in the ensemble as the performance between the models were minimal. The retuned Prophet Boost was nominated to be the default Prophet Boost for the ensemble as its accuracy was markedly better than the original version. As Random Forest performed much better than Prophet Boost, the weightage given to Random Forest was at least 80%.

```
fun_ensemble<- function(M1, W1, W2){
  ensemble_table<-e2_table %>%
    filter(.model_desc %in% c(M1, "PB_retuned")) %>%
    ensemble_weighted(loadings = c(W1, W2)) %>%
    modeltime_table() %>%
    modeltime_accuracy(testing(splits), metric_set = metrics_custom)

  ensemble_table %>%
    mutate(Model1_wt= paste0(M1,"::",W1 ),
           Model2_wt=paste0("PB_retuned::", W2)) %>%
  select(Model1_wt, Model2_wt, rmse, mae)
                      }

ensemble_rmse<-bind_rows(fun_ensemble("RF_retuned", 9,1),
          fun_ensemble("RF_retuned", 8,2),
```

```
          fun_ensemble("RF", 9,1),
          fun_ensemble("RF", 8,2))
```

3.1 Peformance (ensemble)

All the ensemble models performed better than its member models. Better performing ensemble models had a stronger bias to Random Forest.

```
ensemble_rmse%>%  arrange(rmse,sort=T)
## # A tibble: 4 x 4
##   Model1_wt     Model2_wt      rmse   mae
##   <chr>         <chr>         <dbl> <dbl>
## 1 RF_retuned::9 PB_retuned::1  535.  412.
## 2 RF::9         PB_retuned::1  538.  412.
## 3 RF_retuned::8 PB_retuned::2  539.  421.
## 4 RF::8         PB_retuned::2  543.  423.
e3_ensembled_table <- e2_table %>%
    filter(.model_desc %in% c("RF_retuned", "PB_retuned")) %>%
    ensemble_weighted(loadings = c(9, 1)) %>%
    modeltime_table()
```

4 Performance (individual levels)

The performance above was for all hierarchical levels, here the performance for each specific level was reviewed and visualized.

```
e3_ensembled_cal <-modeltime_calibrate(e3_ensembled_table, new_data =
testing(splits))
```

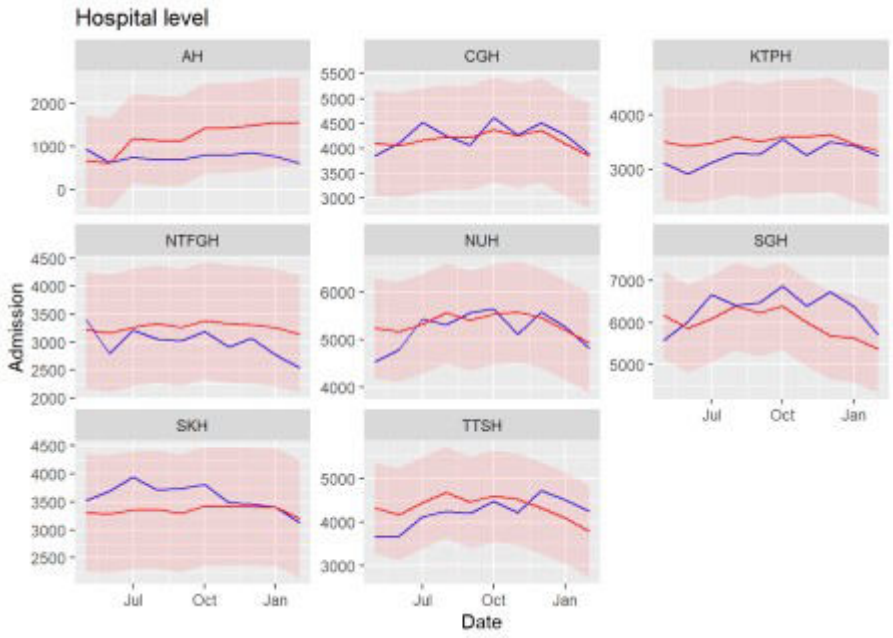
`modeltime_accuracy` and was unable to evaluate the accuracy for individual levels, a customised function was built.

```
e3_ensembled_fcast<-  e3_ensembled_cal %>%
  modeltime_forecast(new_data = testing(splits), actual_data =
to_train, keep_data = T) %>%
  filter(.key!="actual") %>%
  select(c(Date, Level, Name, Admission, .value, .conf_lo, .conf_hi))

fun_lvlacc<- function(L){
e3_ensembled_fcast %>%
  filter(Level==L) %>%
  metrics_custom(truth=.value, estimate=Admission)
                      }

fun_lvlplt<- function(DF, L, TT){
  DF %>% filter(Level==L) %>%
          ggplot(aes(Date))+
          geom_line(aes(y=Admission), col="blue")+
          geom_line(aes(y=.value), col="red")+
          geom_ribbon(aes(ymin=.conf_lo, ymax=.conf_hi),  fill = "red",
alpha = 0.1)+
          facet_wrap(vars(Name), scales = "free_y") +
          labs(title = paste(TT))
```

}

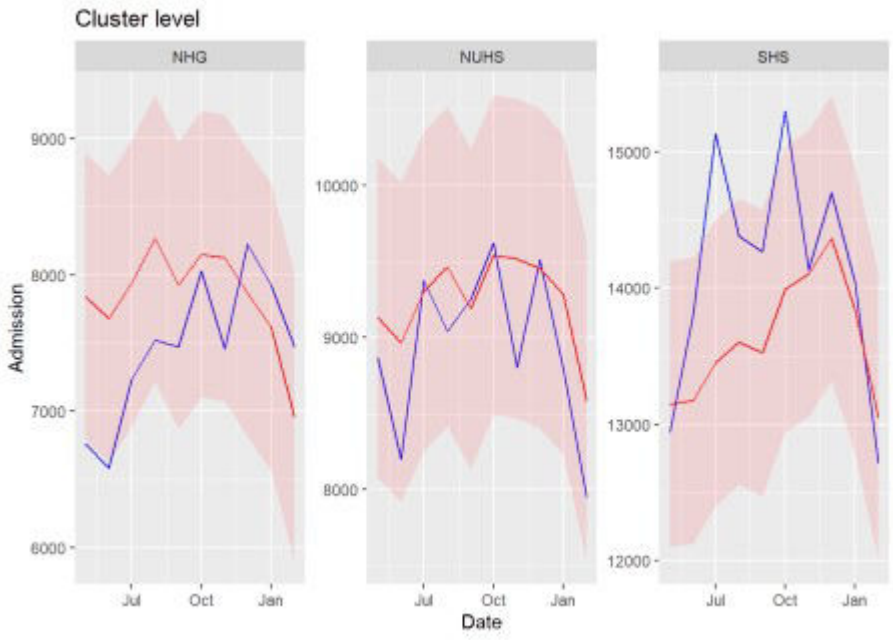
Hospital

```
(fun_lvlacc("Hospital_id"))
## # A tibble: 2 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard        393.
## 2 mae     standard        321.
(fun_lvlplt(e3_ensembled_fcast, "Hospital_id", "Hospital level"))
```



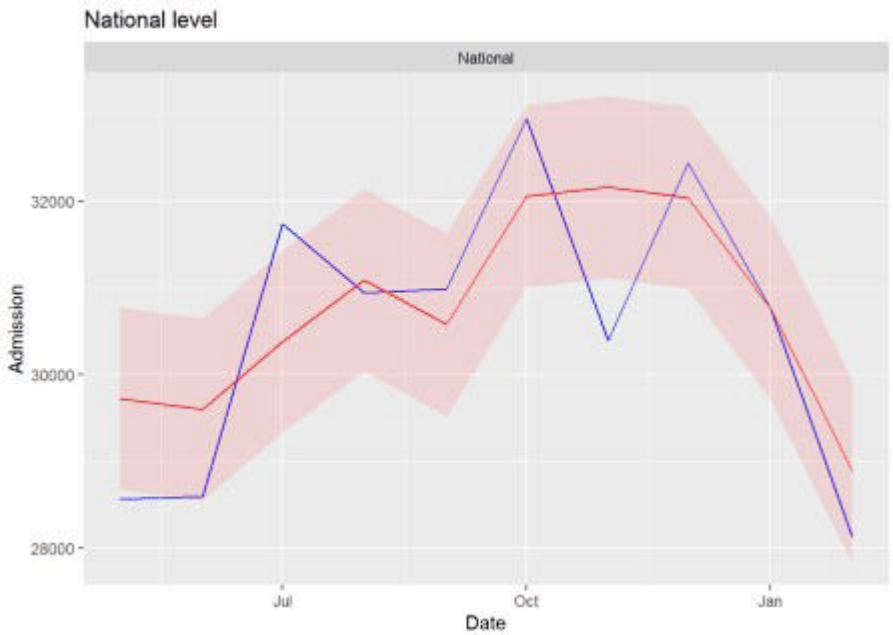
Cluster level

```
(fun_lvlacc("Cluster_id"))
## # A tibble: 2 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard        657.
## 2 mae     standard        528.
(fun_lvlplt(e3_ensembled_fcast, "Cluster_id", "Cluster level"))
```



National level

```
(fun_lvlacc("National_id"))
## # A tibble: 2 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard        949.
## 2 mae     standard        789.
(fun_lvlplt(e3_ensembled_fcast, "National_id", "National level"))
```



5 The future

Below are the forecasted future admissions for Mar 21- Dec 21.

```
Future<-modeltime_refit(e3_ensembled_cal, to_train, control =
control_refit(allow_par = T)) %>%
  modeltime_forecast(new_data = to_predictfuture, actual_data =
```

```
to_train, keep_data = T)

fun_lvlplt_Future<- function(L){
  Future%>%
  filter_by_time(.start_date = last(Date) %-time% "24 month", .end_date
= "end") %>%
    filter(Level==L) %>%
    group_by(Name) %>%
    plot_modeltime_forecast(
        .facet_ncol         = 3,
        .conf_interval_show = TRUE,
        .interactive        = F) + guides(x =  guide_axis(angle = 30))
                              }
```

Hospital

Most of the hospital level forecast were relatively flat; perhaps, due to insufficient data compared to cluster and national level which had more observations from aggregation of subordinate levels.

```
fun_lvlplt_Future("Hospital_id")
```

Cluster

```
fun_lvlplt_Future("Cluster_id")
```

National

The forecast for cluster and national level appeared more plausible with some peaks and dips with an upward trend. Nonetheless, forecasting during this Covid period is challenging. Any forecast can be thrown off the rails as the situation is erratic and dynamic. For instance, the Covid infection rate was stable after Aug 20 but became more serious in May 21 with the Singapore government implementing stricter social distancing measures.

```
fun_lvlplt_Future("National_id")
```

6 KIV Plans

The `modeltime` package has tremendously facilitated time series forcasting the `tidyverse` way. The forecasting could have been improved by

- trying a greater variety of machine learning techniques offered by `tidymodels`
- replacing `XGB` in Prophet Boost with other tree-based boost models like `Catboost` or

`lightGBM`

- using deep learning.

Errors

A stacked ensembled was attempted but there was an error with resample predictions which was needed before feeding into `ennsemble_model_spec`.

```
r_resamples<-splits_train %>%
    time_series_cv(date_var=Date, assess = 10, cumulative = T, skip =
5)
## Overlapping Timestamps Detected. Processing overlapping time series
together using sliding windows.
e_table<-modeltime_table(glm_f, mars_f, rf_f, xgb_f, pb_f)

r_fitresamples <- e_table  %>% modeltime_fit_resamples(r_resamples,
        control   = control_resamples(allow_par = F, save_pred = T,
verbose = T))
## -- Fitting Resamples -------------------------------------------
## * Model ID: 1 GLMNET
## i Slice1: preprocessor 1/1
## x Slice1: preprocessor 1/1: Error: Only one factor level in Covid:
no
## i Slice2: preprocessor 1/1
## x Slice2: preprocessor 1/1: Error: Only one factor level in Covid:
no
## i Slice3: preprocessor 1/1
## x Slice3: preprocessor 1/1: Error: Only one factor level in Covid:
no
## i Slice4: preprocessor 1/1
## x Slice4: preprocessor 1/1: Error: Only one factor level in Covid:
no
## i Slice5: preprocessor 1/1
## x Slice5: preprocessor 1/1: Error: Only one factor level in Covid:
no
## i Slice6: preprocessor 1/1
## x Slice6: preprocessor 1/1: Error: Only one factor level in Covid:
no
## i Slice7: preprocessor 1/1
## x Slice7: preprocessor 1/1: Error: Only one factor level in Covid:
no
## i Slice8: preprocessor 1/1
## x Slice8: preprocessor 1/1: Error: Only one factor level in Covid:
no
## * Model ID: 2 EARTH
## i Slice1: preprocessor 1/1
## x Slice1: preprocessor 1/1: Error: Only one factor level in Covid:
no
## i Slice2: preprocessor 1/1
## x Slice2: preprocessor 1/1: Error: Only one factor level in Covid:
no
```

```
## i Slice3: preprocessor 1/1
## x Slice3: preprocessor 1/1: Error: Only one factor level in Covid:
no
## i Slice4: preprocessor 1/1
## x Slice4: preprocessor 1/1: Error: Only one factor level in Covid:
no
## i Slice5: preprocessor 1/1
## x Slice5: preprocessor 1/1: Error: Only one factor level in Covid:
no
## i Slice6: preprocessor 1/1
## x Slice6: preprocessor 1/1: Error: Only one factor level in Covid:
no
## i Slice7: preprocessor 1/1
## x Slice7: preprocessor 1/1: Error: Only one factor level in Covid:
no
## i Slice8: preprocessor 1/1
## x Slice8: preprocessor 1/1: Error: Only one factor level in Covid:
no
## * Model ID: 3 RANGER
## i Slice1: preprocessor 1/1
## x Slice1: preprocessor 1/1: Error: Only one factor level in Covid:
no
## i Slice2: preprocessor 1/1
## x Slice2: preprocessor 1/1: Error: Only one factor level in Covid:
no
## i Slice3: preprocessor 1/1
## x Slice3: preprocessor 1/1: Error: Only one factor level in Covid:
no
## i Slice4: preprocessor 1/1
## x Slice4: preprocessor 1/1: Error: Only one factor level in Covid:
no
## i Slice5: preprocessor 1/1
## x Slice5: preprocessor 1/1: Error: Only one factor level in Covid:
no
## i Slice6: preprocessor 1/1
## x Slice6: preprocessor 1/1: Error: Only one factor level in Covid:
no
## i Slice7: preprocessor 1/1
## x Slice7: preprocessor 1/1: Error: Only one factor level in Covid:
no
## i Slice8: preprocessor 1/1
## x Slice8: preprocessor 1/1: Error: Only one factor level in Covid:
no
## * Model ID: 4 XGBOOST
## i Slice1: preprocessor 1/1
## x Slice1: preprocessor 1/1: Error: Only one factor level in Covid:
no
## i Slice2: preprocessor 1/1
## x Slice2: preprocessor 1/1: Error: Only one factor level in Covid:
no
## i Slice3: preprocessor 1/1
## x Slice3: preprocessor 1/1: Error: Only one factor level in Covid:
```

```
no
## i Slice4: preprocessor 1/1
## x Slice4: preprocessor 1/1: Error: Only one factor level in Covid:
no
## i Slice5: preprocessor 1/1
## x Slice5: preprocessor 1/1: Error: Only one factor level in Covid:
no
## i Slice6: preprocessor 1/1
## x Slice6: preprocessor 1/1: Error: Only one factor level in Covid:
no
## i Slice7: preprocessor 1/1
## x Slice7: preprocessor 1/1: Error: Only one factor level in Covid:
no
## i Slice8: preprocessor 1/1
## x Slice8: preprocessor 1/1: Error: Only one factor level in Covid:
no
## * Model ID: 5 PROPHET W/ XGBOOST ERRORS
## i Slice1: preprocessor 1/1
## x Slice1: preprocessor 1/1: Error: Only one factor level in Covid:
no
## i Slice2: preprocessor 1/1
## x Slice2: preprocessor 1/1: Error: Only one factor level in Covid:
no
## i Slice3: preprocessor 1/1
## x Slice3: preprocessor 1/1: Error: Only one factor level in Covid:
no
## i Slice4: preprocessor 1/1
## x Slice4: preprocessor 1/1: Error: Only one factor level in Covid:
no
## i Slice5: preprocessor 1/1
## x Slice5: preprocessor 1/1: Error: Only one factor level in Covid:
no
## i Slice6: preprocessor 1/1
## x Slice6: preprocessor 1/1: Error: Only one factor level in Covid:
no
## i Slice7: preprocessor 1/1
## x Slice7: preprocessor 1/1: Error: Only one factor level in Covid:
no
## i Slice8: preprocessor 1/1
## x Slice8: preprocessor 1/1: Error: Only one factor level in Covid:
no
## 6.99 sec elapsed
modeltime_resample_accuracy( r_fitresamples)
## Error: Only strings can be converted to symbols
# snap shot of resamples
#r_resamples %>%  tk_time_series_cv_plan() %>% filter(Name== "CGH") %>%
filter(.id %in% c("Slice1","Slice2","Slice3","Slice8")) %>%
group_by(.id) %>%  plot_time_series(.date_var = Date, .value=
Admission, .interactive=F, .color_var= `.key`, .smooth=F)
```

The error with calculating the accuracy was likely due to missing predictions when the resample was fitted due to the Covid dummy variable. The peak period of Covid was only from Jan-Jul 20

thus there would have been plenty of samples with only `no` Covid peak periods. One alternative was to omit `Covid` as a predictor but as the pandemic still looms on, it was more appropriate to retain this variable when forecasting.

```
r_fitresamples$.resample_results[[1]]$.predictions
## [[1]]
## NULL
##
## [[2]]
## NULL
##
## [[3]]
## NULL
##
## [[4]]
## NULL
##
## [[5]]
## NULL
##
## [[6]]
## NULL
##
## [[7]]
## NULL
##
## [[8]]
## NULL
r_fitresamples$.resample_results[[2]]$.predictions
## [[1]]
## NULL
##
## [[2]]
## NULL
##
## [[3]]
## NULL
##
## [[4]]
## NULL
##
## [[5]]
## NULL
##
## [[6]]
## NULL
##
## [[7]]
## NULL
##
## [[8]]
## NULL
```