1 Intro

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).

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

# cleaned up dataset downloaded from  my github. Clean up of OG dataset done in 1st post
raw<- read_csv("https://raw.githubusercontent.com/notast/hierarchical-forecasting/main/stat_sg_CLEAN.csv")
```

The admissions were treated as a hierarchical time series. Every country has a hierarchical order to its public hospitals. In Singapore, there are 3 levels:

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. Previously, classical approach using traditional techniques e.g. bottoms up and newer techniques e.g. reconciliation were employed. Machine learning approach would be trial in the next few posts. In this post, different combinations of predictors were screened to determine the set of predictors for machine learning.

2 Data wrangling

The dataset starts from Jan 2016 and ends in Feb 2021. The training set was from Jan 16 to Apr 20 (3 years, 4months) and the testing set was from May 20 to Feb 21 (10 months). The forecast horizon was 10 months as the data ended in Feb 21 and the goal was to forecast admissions till the end of 2021. The splitting of training, testing and future data are covered later in this section.

2.1 Long format with aggregated values

When `fpp3::reconcile` was used for classical approach, each column represented the members of that particular level and aggregated values for each level were not required.

```
tribble(
  ~Level1_Hospital, ~Level2_Cluster, ~Admission,
  "CGH", "SHS", 100,
  "SKH", "SHS", 200,
  "TTSH", "NHG", 900)
## # A tibble: 3 x 3
##   Level1_Hospital Level2_Cluster Admission
##   <chr>           <chr>              <dbl>
## 1 CGH             SHS                  100
## 2 SKH             SHS                  200
## 3 TTSH            NHG                  900
```

In the machine learning approach, all the hierarchical levels form a variable and the corresponding subordinate members form another variable. Hence, the aggregated values had to be calculated. The variable of all levels and the variable of the corresponding members were treated as categorical variables in this machine learning problem.

```
tribble(
  ~Level, ~Name, ~Admission,
  "Level 1", "CGH", 100,
  "Level 1", "SKH", 200,
  "Level 1", "TTSH", 900,
  "Level 2" ,"SHS", 300,
  "Level 2", "NHG", 900,
  "Level 3", "National", 1200)
## # A tibble: 6 x 3
##   Level   Name     Admission
##   <chr>   <chr>        <dbl>
## 1 Level 1 CGH            100
## 2 Level 1 SKH            200
## 3 Level 1 TTSH           900
## 4 Level 2 SHS            300
## 5 Level 2 NHG            900
## 6 Level 3 National      1200
df<-raw %>%
  # add national
  mutate(National_id= "National") %>%
  #  add id as suffix to all levels
   rename_with(.fn= ~paste0(.x, "_id"), .cols = c(Hospital, Cluster ))
%>%
  pivot_longer(cols=ends_with("id"), names_to = "Level",
values_to="Name")%>%
  group_by(Level, Name, Date) %>% summarise(Admission=
```

```
sum(Admission,na.rm=T), .groups="drop")
```

2.2 Extend into the future

The machine learning approach was supported with the `modeltime` meta-package which is like a time series equivalent to `tidymodels`. In `modeltime` when a machine learning approach is adopted with external regressors, the future period to be forecasted is appended to the dataset.

```
full<- df %>%
  group_by(Level, Name) %>%
# .bind_data to append
  future_frame(Date, .length_out = 10, .bind_data = TRUE) %>%
  ungroup()

full %>% tail(10)
## # A tibble: 10 x 4
##    Level       Name     Date       Admission
##    <chr>       <chr>    <date>         <dbl>
##  1 National_id National 2021-03-01        NA
##  2 National_id National 2021-04-01        NA
##  3 National_id National 2021-05-01        NA
##  4 National_id National 2021-06-01        NA
##  5 National_id National 2021-07-01        NA
##  6 National_id National 2021-08-01        NA
##  7 National_id National 2021-09-01        NA
##  8 National_id National 2021-10-01        NA
##  9 National_id National 2021-11-01        NA
## 10 National_id National 2021-12-01        NA
```

2.3 External regressor

2.3.1 Lags and rolling lags

Lags and rolling lags were external regressors included in the machine learning approach as these features have seen success in the M5 competition. While lags and rolling windows could have been added with `step_lags` and `step_slidify` during the `recipe` phase, these values couldn't be calculated for future dates. Thus, it was calculated in this dataset which included future dates.
Lag periods were set to be the same as the forecast horizon and the rolling lags were based temporal periods heuristically associated with a year e.g. 3,6,12 months.

```
full<- full %>% group_by(Level, Name) %>%
   tk_augment_lags(Admission, .lags = 10) %>%
    tk_augment_slidify(
        Admission_lag10,
        .f       = ~ mean(., na.rm = TRUE),
        .period  = c(3,6,12),
        .align   = "center",
        .partial = TRUE) %>% ungroup()
```

Impute

Lags and rolling lags resulted in missing values for earlier observations.

```
full %>% head(10)
## # A tibble: 10 x 8
##    Level      Name  Date       Admission Admission_lag10
Admission_lag10_roll_3
##    <chr>      <chr> <date>         <dbl>           <dbl>
<dbl>
##  1 Cluster_id NHG   2016-01-01      8035              NA
NaN
##  2 Cluster_id NHG   2016-02-01      7526              NA
NaN
##  3 Cluster_id NHG   2016-03-01      8419              NA
NaN
##  4 Cluster_id NHG   2016-04-01      7934              NA
NaN
##  5 Cluster_id NHG   2016-05-01      8048              NA
NaN
##  6 Cluster_id NHG   2016-06-01      8199              NA
NaN
##  7 Cluster_id NHG   2016-07-01      8230              NA
NaN
##  8 Cluster_id NHG   2016-08-01      8496              NA
NaN
##  9 Cluster_id NHG   2016-09-01      7991              NA
NaN
## 10 Cluster_id NHG   2016-10-01      8284              NA
8035
## # ... with 2 more variables: Admission_lag10_roll_6 <dbl>,
## #   Admission_lag10_roll_12 <dbl>
```

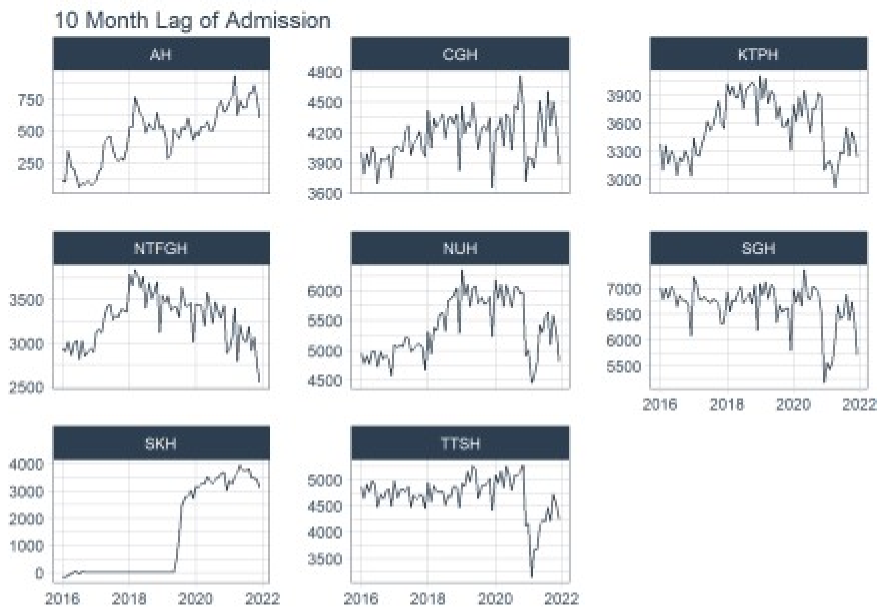
These observations were either to be discarded or imputed. As discarding might result in less than ideal number of observations for training , these values were imputed. Missing lags and rolling lags could have been imputed with `step_ts_clean` or `step_ts_impute`. However, these `step_`s do not operate on group data i.e. imputation at a global level and not for specific hospitals and clusters. Therefore, the imputation occurred earlier up here.

```
 (full<-full %>% group_by(Level, Name) %>%
  mutate(across(.cols= starts_with("Admission_"),
                .fns = ~ ts_impute_vec(.x, period = 12))) %>%
  ungroup() %>%  rowid_to_column(var = "row_id"))
## # A tibble: 864 x 9
##    row_id Level    Name  Date       Admission Admission_lag10
Admission_lag10_r~
##     <int> <chr>    <chr> <date>         <dbl>           <dbl>
<dbl>
##  1      1 Cluster~ NHG   2016-01-01      8035           8184.
7840.
##  2      2 Cluster~ NHG   2016-02-01      7526           7801.
8185.
##  3      3 Cluster~ NHG   2016-03-01      8419           8283.
8072.
##  4      4 Cluster~ NHG   2016-04-01      7934           7926.
8188.
```

```
## 5       5 Cluster~ NHG   2016-05-01      8048           8168.
8169.
## 6       6 Cluster~ NHG   2016-06-01      8199           8229.
8097.
## 7       7 Cluster~ NHG   2016-07-01      8230           7608.
7982.
## 8       8 Cluster~ NHG   2016-08-01      8496           7969.
7844.
## 9       9 Cluster~ NHG   2016-09-01      7991           7792.
8009.
## 10     10 Cluster~ NHG   2016-10-01      8284           8041.
8035
## # ... with 854 more rows, and 2 more variables:
Admission_lag10_roll_6 <dbl>,
## #   Admission_lag10_roll_12 <dbl>
```

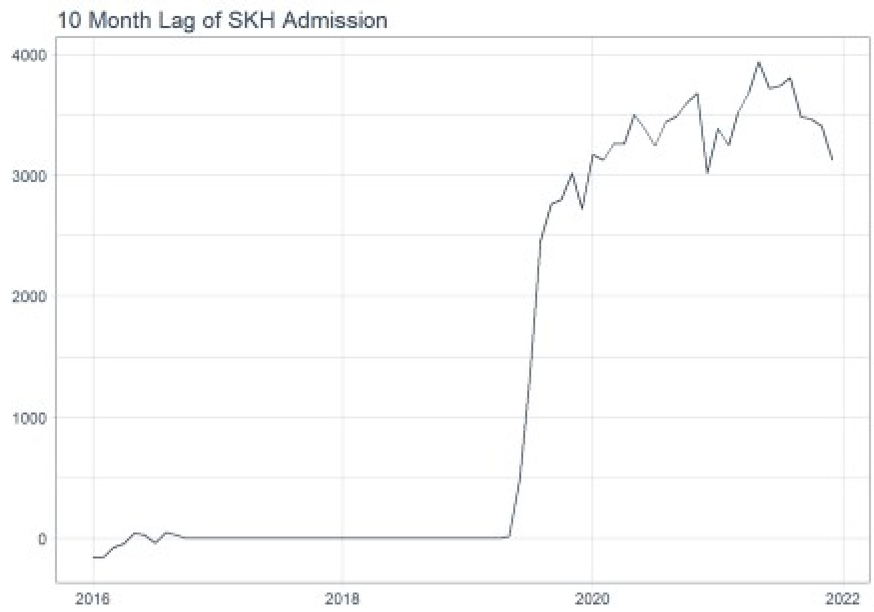
The imputed values were visually inspected.

```
full %>%
  filter(Level=="Hospital_id") %>%
  group_by(Name) %>% plot_time_series(Date, Admission_lag10,
.interactive=F, .smooth = F, .facet_ncol = 3, .title = "10 Month Lag of
Admission")
```



SKH had lags which were below 0 which is impossible as negative admission is absurd. There were also lags above 0 in 2016 which was not impossible as the hospital opened only in Jul 18.

```
  full %>%
  filter(Name=="SKH") %>%
   plot_time_series(Date, Admission_lag10, .interactive=F, .smooth = F,
                  .t= "10 Month Lag of SKH Admission")
```



Thus, the imputations were revised:

1. The minimum values for lags and rolling lags were set to 0
2. All lags and rolling lags for SKH before Jul 18 were set to 0

```
full<-full %>%
  mutate(across(.cols=(starts_with("Admission_lag")),
                .fns = ~ ifelse(.x<0, 0, .x))) %>%
    mutate(across(.cols = (starts_with("Admission_lag")),
                .fns = ~ifelse(Date<lubridate::ymd("2018-07-01")&
Name=="SKH",
                               0, .x)))
```

2.3.2 Covid

Covid peak periods occurred between Jan 21 to Jul 21 and this period form another categorical variable.

```
full<- full %>%
  mutate(Covid= ifelse(
    between(Date, lubridate::ymd("2020-01-01"),
lubridate::ymd("2020-07-01")),
        "yes", "no"))
```

2.3.3 Time series features

Time series features and statistics have been used in machine learning approaches for forecasting. Likewise, the time series features of this hierarchical time series were used as predictors for our machine learning approach.

```
feat_all_url<-url("https://github.com/notast/hierarchical-forecasting/blob/
main/3feat_all.RData?raw=true")
load(feat_all_url)
close(feat_all_url)

full<- left_join(full, feat_all, by=c("Level", "Name"))
```

Appending the future forecast periods and adding the external regressors, increased the dataset from 744 rows to 864 rows and from 4 columns to 47 columns.

```
glimpse(full)
## Rows: 864
## Columns: 47
## $ row_id                    <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11,
12, 13, 14~
## $ Level                     <chr> "Cluster_id", "Cluster_id",
"Cluster_id", "Cl~
## $ Name                      <chr> "NHG", "NHG", "NHG", "NHG", "NHG",
"NHG", "NH~
## $ Date                      <date> 2016-01-01, 2016-02-01,
2016-03-01, 2016-04-~
## $ Admission                 <dbl> 8035, 7526, 8419, 7934, 8048, 8199,
8230, 849~
## $ Admission_lag10           <dbl> 8184.009, 7801.307, 8282.589,
7925.547, 8167.~
## $ Admission_lag10_roll_3    <dbl> 7839.890, 8184.588, 8071.753,
8187.940, 8168.~
## $ Admission_lag10_roll_6    <dbl> 8049.493, 8091.948, 8236.410,
8140.612, 8162.~
## $ Admission_lag10_roll_12   <dbl> 8057.390, 8068.275, 8064.694,
8049.987, 8035.~
## $ Covid                     <chr> "no", "no", "no", "no", "no", "no",
"no", "no~
## $ F_trend_strength          <dbl> 0.6829768, 0.6829768, 0.6829768,
0.6829768, 0~
## $ F_seasonal_strength_year <dbl> 0.4330629, 0.4330629, 0.4330629,
0.4330629, 0~
## $ F_seasonal_peak_year      <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, ~
## $ F_seasonal_trough_year    <dbl> 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2, 2, ~
## $ F_spikiness               <dbl> 10383595, 10383595, 10383595,
10383595, 10383~
## $ F_linearity               <dbl> -1526.063, -1526.063, -1526.063,
-1526.063, -~
## $ F_curvature               <dbl> -2821.471, -2821.471, -2821.471,
-2821.471, -~
## $ F_stl_e_acf1              <dbl> 0.6609598, 0.6609598, 0.6609598,
0.6609598, 0~
## $ F_stl_e_acf10             <dbl> 1.239149, 1.239149, 1.239149,
1.239149, 1.239~
## $ F_acf1                    <dbl> 0.6453483, 0.6453483, 0.6453483,
0.6453483, 0~
## $ F_acf10                   <dbl> 1.279804, 1.279804, 1.279804,
1.279804, 1.279~
## $ F_diff1_acf1              <dbl> -0.4949473, -0.4949473, -0.4949473,
-0.494947~
## $ F_diff1_acf10             <dbl> 0.6419415, 0.6419415, 0.6419415,
0.6419415, 0~
```

```
## $ F_diff2_acf1            <dbl> -0.7519704, -0.7519704, -0.7519704,
-0.751970~
## $ F_diff2_acf10           <dbl> 1.209413, 1.209413, 1.209413,
1.209413, 1.209~
## $ F_season_acf1           <dbl> 0.1026575, 0.1026575, 0.1026575,
0.1026575, 0~
## $ F_kpss_stat             <dbl> 0.410627, 0.410627, 0.410627,
0.410627, 0.410~
## $ F_kpss_pvalue           <dbl> 0.07257455, 0.07257455, 0.07257455,
0.0725745~
## $ F_pp_stat               <dbl> -3.333663, -3.333663, -3.333663,
-3.333663, -~
## $ F_pp_pvalue             <dbl> 0.02307222, 0.02307222, 0.02307222,
0.0230722~
## $ F_ndiffs                <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, ~
## $ F_bp_stat               <dbl> 25.82141, 25.82141, 25.82141,
25.82141, 25.82~
## $ F_bp_pvalue             <dbl> 3.745109e-07, 3.745109e-07,
3.745109e-07, 3.7~
## $ F_lb_stat               <dbl> 27.09132, 27.09132, 27.09132,
27.09132, 27.09~
## $ F_lb_pvalue             <dbl> 1.940678e-07, 1.940678e-07,
1.940678e-07, 1.9~
## $ F_var_tiled_var         <dbl> 0.2356997, 0.2356997, 0.2356997,
0.2356997, 0~
## $ F_var_tiled_mean        <dbl> 0.696821, 0.696821, 0.696821,
0.696821, 0.696~
## $ F_shift_level_max       <dbl> 1404, 1404, 1404, 1404, 1404, 1404,
1404, 140~
## $ F_shift_level_index     <dbl> 50, 50, 50, 50, 50, 50, 50, 50, 50,
50, 50, 5~
## $ F_shift_var_max         <dbl> 1057581, 1057581, 1057581, 1057581,
1057581, ~
## $ F_shift_var_index       <dbl> 43, 43, 43, 43, 43, 43, 43, 43, 43,
43, 43, 4~
## $ F_shift_kl_max          <dbl> 2.08871, 2.08871, 2.08871, 2.08871,
2.08871, ~
## $ F_shift_kl_index        <dbl> 49, 49, 49, 49, 49, 49, 49, 49, 49,
49, 49, 4~
## $ F_spectral_entropy      <dbl> 0.7173186, 0.7173186, 0.7173186,
0.7173186, 0~
## $ F_n_crossing_points     <int> 18, 18, 18, 18, 18, 18, 18, 18, 18,
18, 18, 1~
## $ F_longest_flat_spot     <int> 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4,
4, 4, 4, ~
## $ F_stat_arch_lm          <dbl> 0.5968071, 0.5968071, 0.5968071,
0.5968071, 0~
```

3 Splitting

The `full` dataset was split into

1. Training (`to_train`) and future prediction (`to_predictfuture`) dataset
2. The training dataset (`to_train`) was further split into the analysis/training and assessment/testing sets.

```
# Datasets to train and predict
to_train<- full %>% filter(!is.na(Admission))
to_predictfuture<- full %>% filter(is.na(Admission))

# Spiltting `to_train`
splits<- to_train %>%
    time_series_split(Date, assess= "10 months", cumulative = T)
## Data is not ordered by the 'date_var'. Resamples will be arranged by
`Date`.
## Overlapping Timestamps Detected. Processing overlapping time series
together using sliding windows.
```

4 Pre-processing `recipes`

When using `modeltime` for forecasting, the date column is treated differently for each approach.

- Classical: The date column is left untoched.
- Machine learning: Features are engineered from the date column e.g. using `as.numeric(date)` or in this case with `step_timeseries_signature`.

The date column is then

i. either dropped. In the case of using `glmnet`, the date format cannot be implicitly converted into numeric format for the needed matrix structure used in `glmnet`
ii. or its role as a predictor is updated to something else.

4 combinations of predictors and pre-processing `step_`s were screened to determine the best combination of features for machine learning.

1. Basic (`rec_basic`)

i. Lags (`Admission_lag10`)
ii. Rolling lags (`Admission_lag10_roll_???`)
iii. Covid peak period (`Covid`)
iv. Relevant features engineered from `step_timeseries_signature`
v. Hierarchical levels (`Level`)
vi. Members in the corresponding level (`Name`)

2. Basic + Time series features and statistics (`rec_ft`)

i. Above `rec_basic`
ii. Time series features. (`F_???`)

3. Basic + PCA of the time series features and statistics (`rec_PC`)

i. Above `rec_basic`
ii. The first 5 principal components of the time series features. 5 components were shown to summarised 88% of the variance.

4. Basic + kernel PCA of the time series and features and statistics (`rec_kPC`)

   i. Above `rec_basic`
   ii. The first 5 principal components of the time series features. Kernel PCA would be more suitable if there were a non-linear relationship between the time series features and the number of admissions.

Pre-processing order

Pre-processing tend to follow this order:

1. Impute
2. Handle Factor levels
3. Individual transformations
4. Discretize
5. Lump minority observations
6. Create dummy variables
7. Create interactions
8. Remove variables with near zero variance
9. Normalize
10. Create splines / Multivariate transformation (e.g. PCA, spatial)

```
fun_rec<-  function(R){
   LHS= "Admission ~"
   RHS = paste(R, collapse = "+")
   formula<- as.formula(paste0(LHS, RHS))

   recipe(formula,  data = training(splits)) %>%
    update_role(c(row_id, Date), new_role = "id") %>%
    step_timeseries_signature(Date) %>%
step_rm(matches("(.xts$)|(.iso$)|(hour)|(minute)|(second)|(am.pm)|
(mweek)")) %>%
    step_rm(starts_with("Date_wday")) %>% step_rm(starts_with("Date_
mday")) %>% step_rm(Date_day) %>%
    step_dummy(all_nominal(), one_hot = TRUE)}

col_lag<- to_train %>% select(starts_with("Admission_lag")) %>%
colnames()
col_bare<- c("Level", "Name", "row_id", "Date", "Covid")
col_ft<- to_train %>% select(starts_with("F_")) %>% colnames()

rec_basic<-fun_rec(c(col_bare, col_lag))

rec_ft<-fun_rec(c(col_bare, col_lag, col_ft)) %>%
  step_corr(!!col_ft) %>%
  step_nzv(all_numeric_predictors())

rec_PC<- fun_rec(c(col_bare, col_lag, col_ft)) %>%
  step_normalize(!!col_ft) %>%
  step_pca(!!col_ft)

rec_kpca<- fun_rec(c(col_bare, col_lag, col_ft)) %>%
  step_normalize(!!col_ft) %>%
```

```
  step_kpca_rbf(!!col_ft)
```

5. Modelling

Random Forest was used as the model to screen the combinations of features as Random Forest tends to do relatively well without hyperparameter tuning. It has also been used in the M4 competition with success where time series features and statistics were predictors. *(Random Forest was just part of the algorithm used in the paper)*.

```
mod<-rand_forest(trees= 1000) %>% # increase from 500->100
  set_engine("ranger", verbose = TRUE) %>%  # use ranger as faster,
increased trees
  set_mode("regression")
```

Workflow

The multiple combination of `recipe`s was easily handed with `workflowsets`.

```
(wfsets <- workflowsets::workflow_set(
    preproc = list(basic = rec_basic, ft= rec_ft, PC= rec_PC,
kPC=rec_kpca),
    models  = list(rf=mod),
    cross   = F))
## # A workflow set/tibble: 4 x 4
##   wflow_id info             option      result
##   <chr>    <list>           <list>      <list>
## 1 basic_rf <tibble [1 x 4]> <wrkflw__ > <list [0]>
## 2 ft_rf    <tibble [1 x 4]> <wrkflw__ > <list [0]>
## 3 PC_rf    <tibble [1 x 4]> <wrkflw__ > <list [0]>
## 4 kPC_rf   <tibble [1 x 4]> <wrkflw__ > <list [0]>
```

6. Evaluate

6.1 Evaluate against the training set

`modeltime_fit_workflowset` only described the models used. It did not not carry forward the `wflow_id`.

```
(wfset_table<-modeltime_fit_workflowset(wfsets, training(splits)))
## 2021-06-12 14:05:33: Calculating kernel PCA
## 2021-06-12 14:05:33: Trying to calculate reverse
## 2021-06-12 14:05:33: DONE
## # Modeltime Table
## # A tibble: 4 x 3
##   .model_id .model     .model_desc
##       <int> <list>     <chr>
## 1         1 <workflow> RANGER
## 2         2 <workflow> RANGER
## 3         3 <workflow> RANGER
## 4         4 <workflow> RANGER
```

Before the models with various features were evaluated, the `.model_desc`riptions were updated to reflect the recipes of the various features and models used. The models from `modeltime_fit_workflowset` were arranged in the alphabetical order of the `wflow_id` which made updating of `.model_desc`riptions easier.

```
(wfset_table<-wfset_table %>%   update_model_description(1,
"rec_basic") %>%
  update_model_description(2, "rec_ft") %>%
  update_model_description(3,  "rec_kPC") %>%
  update_model_description(4, "rec_PC"))
## # Modeltime Table
## # A tibble: 4 x 3
##   .model_id .model     .model_desc
##       <int> <list>     <chr>
## 1         1 <workflow> rec_basic
## 2         2 <workflow> rec_ft
## 3         3 <workflow> rec_kPC
## 4         4 <workflow> rec_PC
```

Alternatively, one could refer to the `recipe` inside `.model` to infer the relevant feature combination

```
wfset_table$.model[[1]]
## == Workflow [trained] ==============================
============================
## Preprocessor: Recipe
## Model: rand_forest()
##
## -- Preprocessor ------------------------------
----------------------------------
## 6 Recipe Steps
##
## * step_timeseries_signature()
## * step_rm()
## * step_rm()
## * step_rm()
## * step_rm()
## * step_dummy()
##
## -- Model ------------------------------
-----------------------------------------
## Ranger result
##
## Call:
##  ranger::ranger(x = maybe_data_frame(x), y = y, num.trees = ~1000,
verbose = ~TRUE, num.threads = 1, seed = sample.int(10^5,         1))
##
## Type:                             Regression
## Number of trees:                  1000
## Sample size:                      624
## Number of independent variables:  44
## Mtry:                             6
## Target node size:                 5
## Variable importance mode:         none
```

```
## Splitrule:                            variance
## OOB prediction error (MSE):           286072.1
## R squared (OOB):                      0.9949897
```

Renaming could have been skipped with this very new package as it specifically deals with `workflowset` for `modeltime`.

The models were calibrated on the training set.

```
(wfset_calibrate<-wfset_table %>% modeltime_calibrate(training(
splits)))
## # Modeltime Table
## # A tibble: 4 x 5
##   .model_id .model     .model_desc .type .calibration_data
##       <int> <list>     <chr>       <chr> <list>
## 1         1 <workflow> rec_basic   Test  <tibble [624 x 4]>
## 2         2 <workflow> rec_ft      Test  <tibble [624 x 4]>
## 3         3 <workflow> rec_kPC     Test  <tibble [624 x 4]>
## 4         4 <workflow> rec_PC      Test  <tibble [624 x 4]>
```

What's inside the calibrated table

The `.calibration_data` from `modeltime_calibrate` contains the `.acutal` number of admissions, the `.prediction` and the `.residuals`.

```
# just one example
wfset_calibrate$.calibration_data[[1]]
## # A tibble: 624 x 4
##    Date       .actual .prediction .residuals
##    <date>       <dbl>       <dbl>      <dbl>
##  1 2016-01-01    8035       8145.     -110.
##  2 2016-01-01    7861       8152.     -291.
##  3 2016-01-01   10659      10554.      105.
##  4 2016-01-01      75        245.     -170.
##  5 2016-01-01    3977       3967.       10.5
##  6 2016-01-01    3214       3366.     -152.
##  7 2016-01-01    2944       3142.     -198.
##  8 2016-01-01    4842       4863.      -21.4
##  9 2016-01-01    6682       6606.       75.9
## 10 2016-01-01       0        202.     -202.
## # ... with 614 more rows
modeltime_accuracy(object=wfset_calibrate,
                   metric_set = metric_set(rmse,mae)) %>%
  arrange(rmse, sort=T)
## # A tibble: 4 x 5
##   .model_id .model_desc .type  rmse   mae
##       <int> <chr>       <chr> <dbl> <dbl>
## 1         4 rec_PC      Test   264.  151.
## 2         3 rec_kPC     Test   266.  151.
## 3         1 rec_basic   Test   277.  161.
## 4         2 rec_ft      Test   314.  182.
```

6.1 Evaluate with cross validation

To increase the robustness of the accuracy, cross validation was executed. The cross validation scores were different than without cross validation.

```
set.seed(69)
folds <- vfold_cv(training(splits), strata = Admission)

set.seed(69)
(wfsets_done<- wfsets %>%
   workflow_map("fit_resamples", resamples = folds))
## 2021-06-12 14:06:06: Calculating kernel PCA
## 2021-06-12 14:06:07: Trying to calculate reverse
## 2021-06-12 14:06:07: DONE
## 2021-06-12 14:06:08: Calculating kernel PCA
## 2021-06-12 14:06:08: Trying to calculate reverse
## 2021-06-12 14:06:08: DONE
## 2021-06-12 14:06:10: Calculating kernel PCA
## 2021-06-12 14:06:10: Trying to calculate reverse
## 2021-06-12 14:06:10: DONE
## 2021-06-12 14:06:11: Calculating kernel PCA
## 2021-06-12 14:06:11: Trying to calculate reverse
## 2021-06-12 14:06:11: DONE
## 2021-06-12 14:06:12: Calculating kernel PCA
## 2021-06-12 14:06:13: Trying to calculate reverse
## 2021-06-12 14:06:13: DONE
## 2021-06-12 14:06:14: Calculating kernel PCA
## 2021-06-12 14:06:14: Trying to calculate reverse
## 2021-06-12 14:06:14: DONE
## 2021-06-12 14:06:15: Calculating kernel PCA
## 2021-06-12 14:06:16: Trying to calculate reverse
## 2021-06-12 14:06:16: DONE
## 2021-06-12 14:06:17: Calculating kernel PCA
## 2021-06-12 14:06:17: Trying to calculate reverse
## 2021-06-12 14:06:17: DONE
## 2021-06-12 14:06:18: Calculating kernel PCA
## 2021-06-12 14:06:18: Trying to calculate reverse
## 2021-06-12 14:06:19: DONE
## 2021-06-12 14:06:20: Calculating kernel PCA
## 2021-06-12 14:06:20: Trying to calculate reverse
## 2021-06-12 14:06:20: DONE
## # A workflow set/tibble: 4 x 4
##   wflow_id info             option      result
##   <chr>    <list>           <list>      <list>
## 1 basic_rf <tibble [1 x 4]> <wrkflw__ > <rsmp[+]>
## 2 ft_rf    <tibble [1 x 4]> <wrkflw__ > <rsmp[+]>
## 3 PC_rf    <tibble [1 x 4]> <wrkflw__ > <rsmp[+]>
## 4 kPC_rf   <tibble [1 x 4]> <wrkflw__ > <rsmp[+]>
collect_metrics(wfsets_done, summarize = T ) %>% # summarised all folds
  filter(.metric=="rmse") %>%
  select(wflow_id, avg_rmse=mean) %>% arrange(avg_rmse,sort=T)
## # A tibble: 4 x 2
##   wflow_id avg_rmse
##   <chr>       <dbl>
```

```
## 1 PC_rf        514.
## 2 kPC_rf       516.
## 3 basic_rf     526.
## 4 ft_rf        543.
```

8 Conclusion

The cross validation scores were different than without cross validation.The bottom 2 performing feature combinations were consistent (`rec_basic` & `rec_ft`). However, when the time series features were condensed into principal components the accuracy improved. The relationship between the time series features and the number of admissions was linear-ish as the performance between kernel PCA and PCA were close for evaluations with and without cross validation. For the marginal difference in rmse and shorter computational time, PCA was selected. The feature combination for machine learning forecasting in the next few posts would be

i. Lags
ii. Rolling lags
iii. Covid peak period
iv. Relevant features engineered from `step_timeseries_signature`
v. The first 5 principal components (PCA) of the time series features
vi. Hierarchical levels and their members

```
# save the dataset for machine learning and the dataset for future
prediction; save the recipe
save(to_train, to_predictfuture, rec_PC, file= "4Dataset4ML.rds")
```