DengAI: Predicting Disease Spread - Benchmark (in R)

Dengue fever is bad. It's real bad. Dengue is a mosquito-borne disease that occurs in tropical and sub-tropical parts of the world. In mild cases, symptoms are similar to the flu: fever, rash and muscle and joint pain. But severe cases are dangerous, and dengue fever can cause severe bleeding, low blood pressure and even death.

Because it is carried by mosquitoes, the transmission dynamics of dengue are related to climate variables such as temperature and precipitation. Although the relationship to climate is complex, a growing number of scientists argue that climate change is likely to produce distributional shifts that will have significant public health implications worldwide.

We've launched a competition to use open data to predict the occurrence of Dengue based on climatological data. Here's a first look at the data and how to get started!

As always, we begin with imports of useful

```
# load libraries
pkgs <- c('tidyverse', 'corrplot', 'magrittr', 'zoo', 'RColorBrewer', 'gridExtra','MASS')
invisible(lapply(pkgs, require, character.only = T))
```

A Tale of Two Cities

This dataset has two cities in it: **San Juan**, Puerto Rico and **Iquitos**, Peru. Since we hypothesize that the spread of dengue may follow different patterns between the two, we will divide the dataset, train seperate models for each city, and then join our predictions before making our final submission.

```
## Data Loading
train_features = read_csv('dengue_features_train.csv')
train_labels   = read_csv('dengue_labels_train.csv')


# Seperate data by city
sj_train_features = train_features %>% filter(city == 'sj')
sj_train_labels   = train_labels   %>% filter(city == 'sj')

iq_train_features = train_features %>% filter(city == 'iq')
iq_train_labels   = train_labels   %>% filter(city == 'iq')


# data shape for each city
cat('\nSan Juan\n',
    '\t features: ', sj_train_features %>% ncol,
    '\t entries: ' , sj_train_features %>% nrow,
    '\t labels: '  , sj_train_labels %>% nrow)
```

```
##
## San Juan
##       features:  24   entries:  936   labels:  936
```

```
cat('\nIquitos\n',
    '\t features: ', iq_train_features %>% ncol,
    '\t entries: ' , iq_train_features %>% nrow,
    '\t labels: '  , iq_train_labels %>% nrow)
```

```
##
```

```
## Iquitos
##       features:  24   entries:  520   labels:  520
```

The problem description gives a good overview of the available variables, but we'll look at the head of the data here as well:

```
# glimspe at the data (first 7 columns only to maintain neat display)
head(sj_train_features[1:7])
```

```
## # A tibble: 6 × 7
##    city  year weekofyear week_start_date   ndvi_ne   ndvi_nw   ndvi_se
##   <chr> <int>      <int>           <date>     <dbl>     <dbl>     <dbl>
## 1    sj  1990         18      1990-04-30 0.1226000 0.1037250 0.1984833
## 2    sj  1990         19      1990-05-07 0.1699000 0.1421750 0.1623571
## 3    sj  1990         20      1990-05-14 0.0322500 0.1729667 0.1572000
## 4    sj  1990         21      1990-05-21 0.1286333 0.2450667 0.2275571
## 5    sj  1990         22      1990-05-28 0.1962000 0.2622000 0.2512000
## 6    sj  1990         23      1990-06-04        NA 0.1748500 0.2543143
```

There are **a lot** of climate variables here, but the first thing that we'll note is that the week_start_date is included in the feature set. This makes it easier for competitors to create time based features, but for this first-pass model, we'll drop that column since we shouldn't use it as a feature in our model.

```
# Remove `week_start_date` string.
sj_train_features %<>% dplyr::select(-week_start_date)
iq_train_features %<>% dplyr::select(-week_start_date)
```

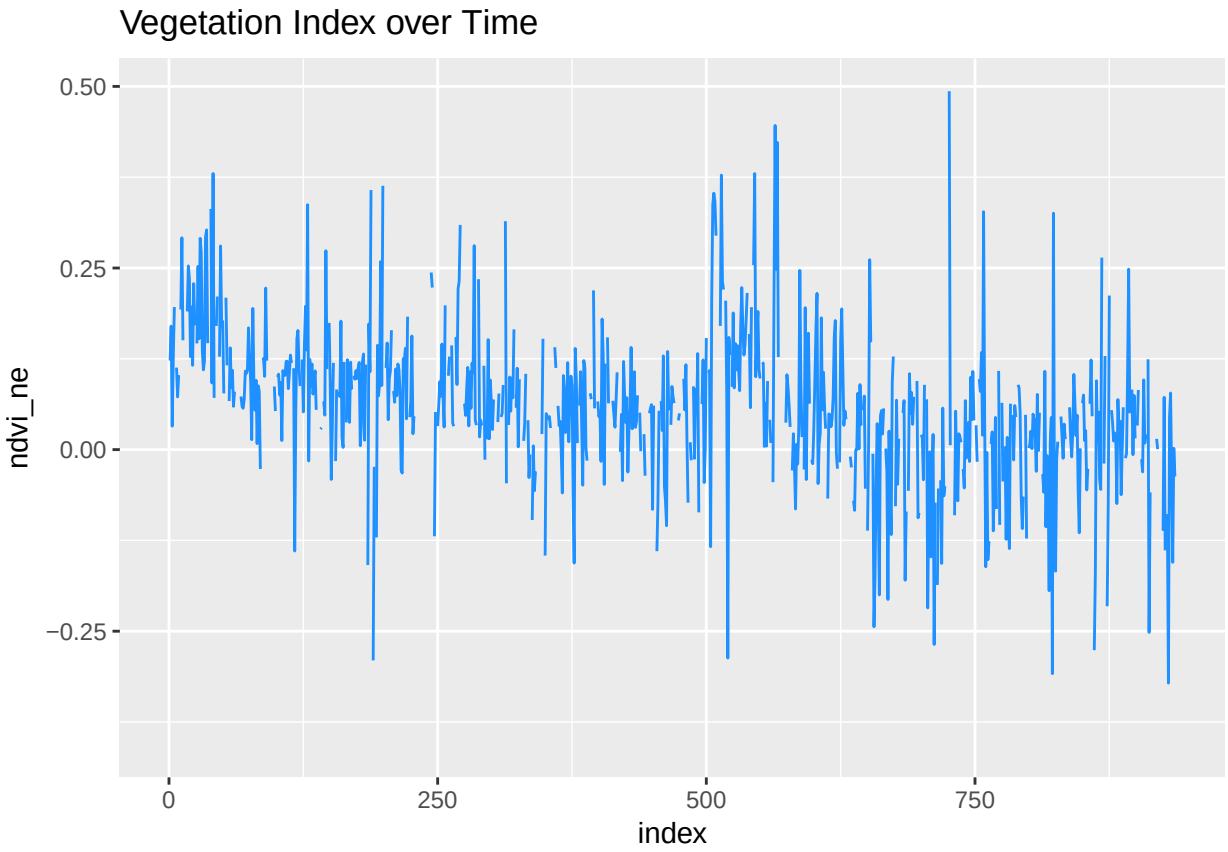
Next, let's check to see if we are missing any values in this dataset:

```
# count missing values (as percent)
apply(sj_train_features, 2, function(x)
  round(100 * (length(which(is.na(x))))/length(x) , digits = 1)) %>%
  as.data.frame() %>%
  `names<-`('Percent of Missing Values')
```

```
##                                       Percent of Missing Values
## city                                                        0.0
## year                                                        0.0
## weekofyear                                                  0.0
## ndvi_ne                                                    20.4
## ndvi_nw                                                     5.2
## ndvi_se                                                     2.0
## ndvi_sw                                                     2.0
## precipitation_amt_mm                                        1.0
## reanalysis_air_temp_k                                       0.6
## reanalysis_avg_temp_k                                       0.6
## reanalysis_dew_point_temp_k                                 0.6
## reanalysis_max_air_temp_k                                   0.6
## reanalysis_min_air_temp_k                                   0.6
## reanalysis_precip_amt_kg_per_m2                             0.6
## reanalysis_relative_humidity_percent                        0.6
## reanalysis_sat_precip_amt_mm                                1.0
## reanalysis_specific_humidity_g_per_kg                       0.6
## reanalysis_tdtr_k                                           0.6
## station_avg_temp_c                                          0.6
## station_diur_temp_rng_c                                     0.6
## station_max_temp_c                                          0.6
```

```
## station_min_temp_c                                    0.6
## station_precip_mm                                     0.6
```

```
sj_train_features %>%
  mutate(index = as.numeric(row.names(.))) %>%
  ggplot(aes(index, ndvi_ne)) +
  geom_line(colour = 'dodgerblue') +
  ggtitle("Vegetation Index over Time")
```



Since these are time-series, we can see the gaps where there are `NaNs` by plotting the data. Since we can't build a model without those values, we'll take a simple approach and just fill those values with the most recent value that we saw up to that point. This is probably a good part of the problem to improve your score by getting smarter.

```
# impute NAs by the latest value
sj_train_features$ndvi_ne %<>% na.locf(fromLast = TRUE)
iq_train_features$ndvi_ne %<>% na.locf(fromLast = TRUE)
```

Distribution of labels

Our target variable, total_cases is a non-negative integer, which means we're looking to make some **count predictions**. Standard regression techniques for this type of prediction include

1. *Poisson regression*
2. *Negative binomial regression*

Which techniqe will perform better depends on many things, but the choice between Poisson regression and negative binomial regression is pretty straightforward. Poisson regression fits according to the assumption

that the mean and variance of the population distributiona are equal. When they aren't, specifically when the variance is much larger than the mean, the negative binomial approach is better. Why? It isn't magic. The negative binomial regression simply lifts the assumption that the population mean and variance are equal, allowing for a larger class of possible models. In fact, from this perspective, the Poisson distribution is but a special case of the negative binomial distribution.

Let's see how our labels are distributed!

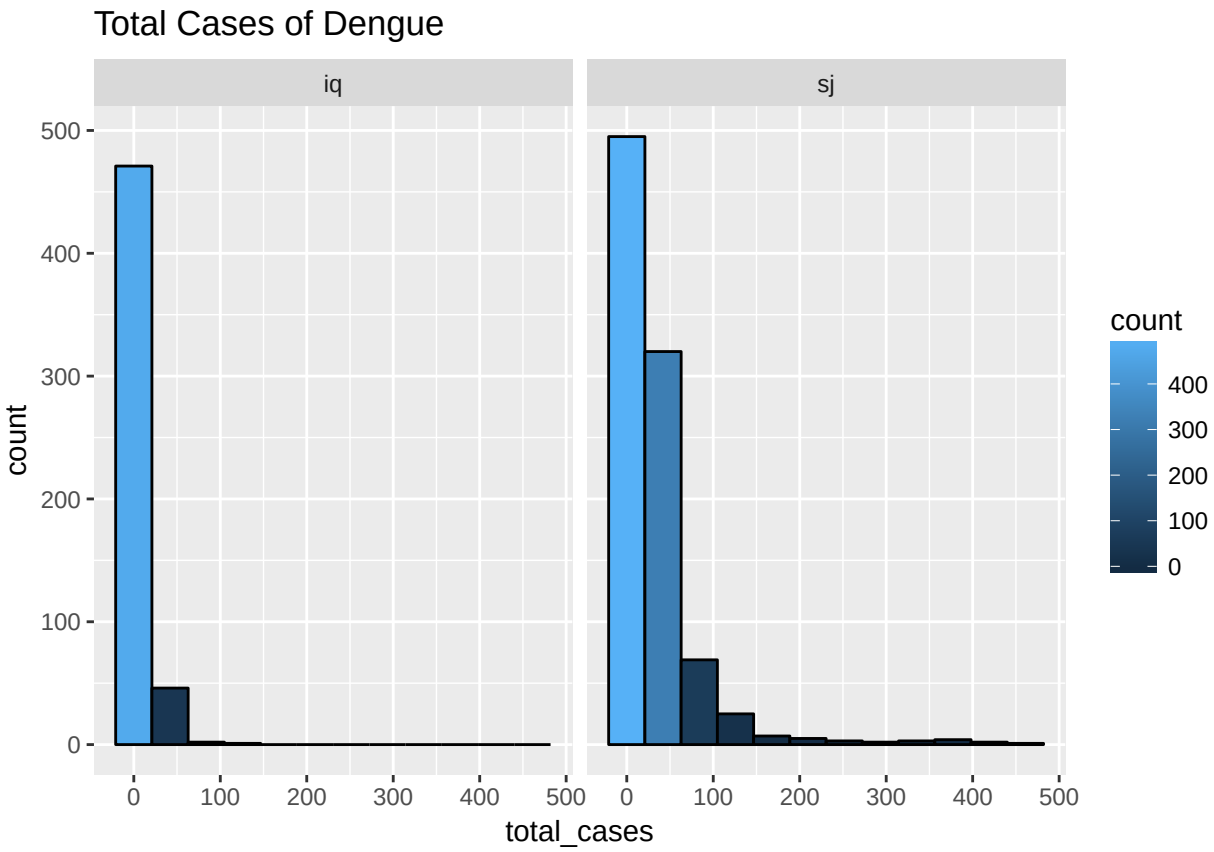
```
# distibution of labels
cat('\nSan Juan\n',
    '\t total cases mean: ',      sj_train_labels$total_cases %>% mean(),
    '\t total cases variance: ' , sj_train_labels$total_cases %>% var() )
```

```
##
## San Juan
##       total cases mean:  34.18056     total cases variance:  2640.045
```

```
cat('\nIquitos\n',
    '\t total cases mean: ',      iq_train_labels$total_cases %>% mean(),
    '\t total cases variance: ' , iq_train_labels$total_cases %>% var() )
```

```
##
## Iquitos
##       total cases mean:  7.565385     total cases variance:  115.8955
```

```
# total cases of dengue: histograms
rbind(iq_train_labels, sj_train_labels) %>%
  ggplot(aes(x = total_cases,fill = ..count..)) +
  geom_histogram(bins = 12, colour = 'black') + ggtitle('Total Cases of Dengue') +
  scale_y_continuous(breaks = seq(0,700,100)) + facet_wrap(~city)
```



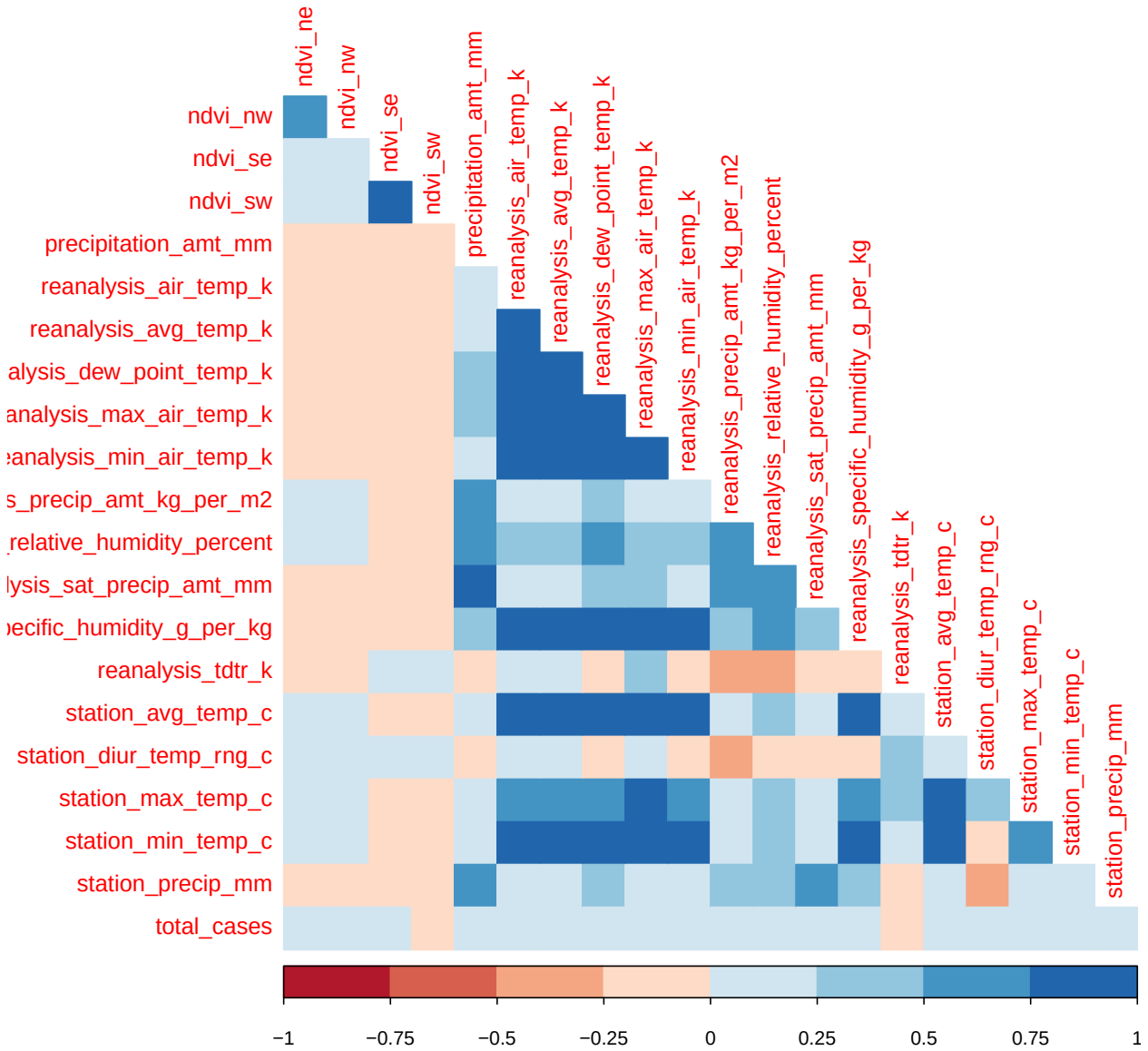
It's looking like a negative-binomial sort of day in these parts.

variance >> mean suggests total_cases can be described by a negative binomial distribution, so we'll use a negative binomial regression below.

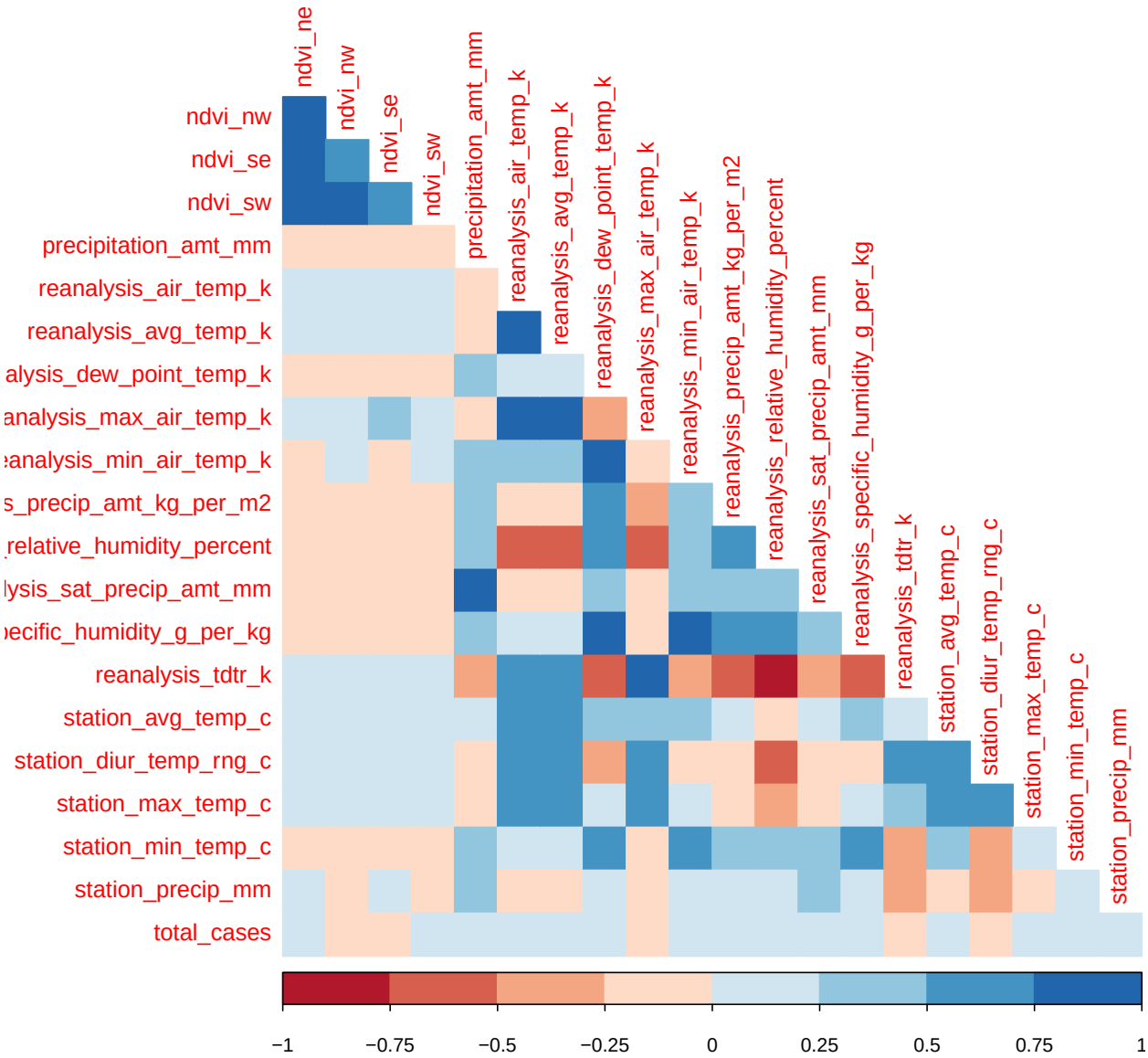
Which inputs strongly correlate with total_cases?

Our next step in this process will be to select a subset of features to include in our regression. Our primary purpose here is to get a better understanding of the problem domain rather than eke out the last possible bit of predictive accuracy. The first thing we will do is to add the total_cases to our dataframe, and then look at the correlation of that variable with the climate variables.

```
# corerlations between features
sj_train_features %<>% mutate('total_cases' = sj_train_labels$total_cases)
iq_train_features %<>% mutate('total_cases' = iq_train_labels$total_cases)

# plot san juan correlation matrix
sj_train_features %>%
  dplyr::select(-city, -year, -weekofyear) %>%
  cor(use = 'pairwise.complete.obs') -> M1

corrplot(M1, type="lower", method="color",
           col=brewer.pal(n=8, name="RdBu"),diag=FALSE)
```




```
# plot iquitos correlation matrix
iq_train_features %>%
  dplyr::select(-city, -year, -weekofyear) %>%
  cor(use = 'pairwise.complete.obs') -> M2

corrplot(M2, type="lower", method="color",
           col=brewer.pal(n=8, name="RdBu"),diag=FALSE)
```

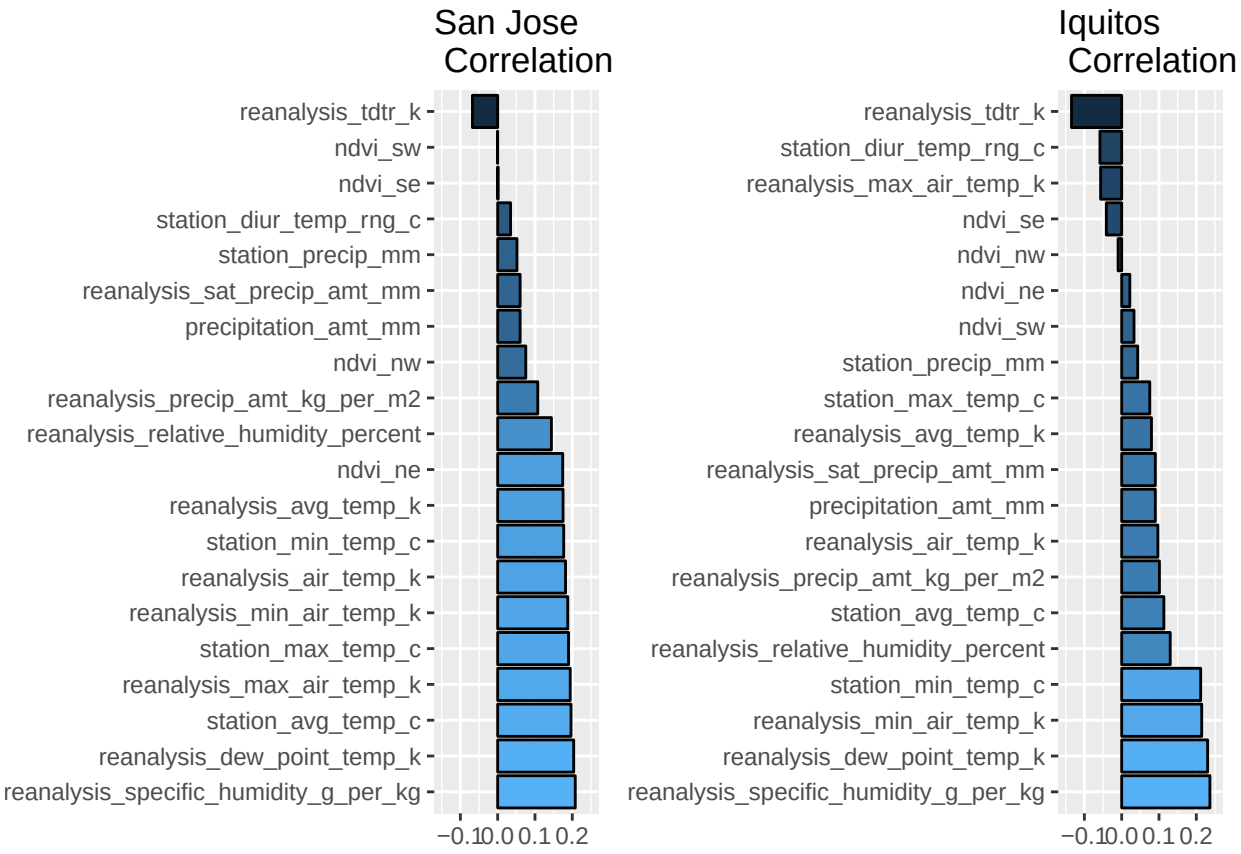


Many of the temperature data are strongly correlated, which is expected. But the total_cases variable doesn't have many obvious strong correlations.

Interestingly, `total_cases` seems to only have weak correlations with other variables. Many of the climate variables are much more strongly correlated. Interestingly, the vegetation index also only has weak correlation with other variables. These correlations may give us some hints as to how to improve our model that we'll talk about later in this post. For now, let's take a sorted look at `total_cases` correlations.

```
# see the correlations as barplot
sort(M1[21,-21]) %>%
  as.data.frame %>%
  `names<-`('correlation') %>%
  ggplot(aes(x = reorder(row.names(.), -correlation), y = correlation, fill = correlation)) +
  geom_bar(stat='identity', colour = 'black') + scale_fill_continuous(guide = FALSE) + scale_y_continuou
  labs(title = 'San Jose\n Correlations', x = NULL, y = NULL) + coord_flip() -> cor1

# can use ncol(M1) instead of 21 to generalize the code
sort(M2[21,-21]) %>%
```

```
  as.data.frame %>%
  `names<-`('correlation') %>%
  ggplot(aes(x = reorder(row.names(.), -correlation), y = correlation, fill = correlation)) +
  geom_bar(stat='identity', colour = 'black') + scale_fill_continuous(guide = FALSE) + scale_y_continuou
  labs(title = 'Iquitos\n Correlations', x = NULL, y = NULL) + coord_flip() -> cor2

grid.arrange(cor1, cor2, nrow = 1)
```



A few observations

The wetter the better

- **The correlation strengths differ for each city, but it looks like reanalysis_specific_humidity_g_per_kg and reanalysis_dew_point_temp_k are the most strongly correlated with total_cases. This makes sense: we know mosquitos thrive wet climates, the wetter the better!**

Hot and heavy

- **As we all know, "cold and humid" is not a thing. So it's not surprising that as minimum temperatures, maximum temperatures, and average temperatures rise, the total_cases of dengue fever tend to rise as well.**

Sometimes it rains, so what

- **Interestingly, the precipitation measurements bear little to no correlation to total_cases, despite strong correlations to the humidity measurements, as evident by the heatmaps above.**

This is just a first pass

Precisely **none** of these correlations are very strong. Of course, that doesn't mean that some **feature engineering wizardry** can't put us in a better place (`standing_water` estimate, anyone?). Also, it's always useful to keep in mind that **life isn't linear**, but out-of-the-box correlation measurement is – or at least, it measures linear dependence.

Nevertheless, for this benchmark we'll focus on the linear **wetness** trend we see above, and reduce our inputs to

A few good variables

- `reanalysis_specific_humidity_g_per_kg`
- `reanalysis_dew_point_temp_k`
- `station_avg_temp_c`
- `station_min_temp_c`

A mosquito model

Now that we've explored this data, it's time to start modeling. Our first step will be to build a function that does all of the preprocessing we've done above from start to finish. This will make our lives easier, since it needs to be applied to the test set and the traning set before we make our predictions.

```
preprocessData <- function(data_path, labels_path = NULL)
{
  # load data
  df <- read_csv(data_path)

  # features we want
  features = c("reanalysis_specific_humidity_g_per_kg", "reanalysis_dew_point_temp_k",
               "station_avg_temp_c", "station_min_temp_c")

  # fill missing values
  df[features] %<>% na.locf(fromLast = TRUE)

  # add city if labels data aren't provided
  if (is.null(labels_path)) features %<>% c("city", "year", "weekofyear")

  # select features we want
  df <- df[features]

  # add labels to dataframe
  if (!is.null(labels_path)) df %<>% cbind(read.csv(labels_path))

  # filter by city
  df_sj <- filter(df, city == 'sj')
  df_iq <- filter(df, city == 'iq')
```

```
  # return a list with the 2 data frames
  return(list(df_sj, df_iq))
}
```

```
# preprocess the .csv files
preprocessData(data_path = 'dengue_features_train.csv', labels_path = 'dengue_labels_train.csv') -> trai
```

```
## Parsed with column specification:
## cols(
##   .default = col_double(),
##   city = col_character(),
##   year = col_integer(),
##   weekofyear = col_integer(),
##   week_start_date = col_date(format = "")
## )
```

```
## See spec(...) for full column specifications.
```

```
sj_train <- trains[[1]]; iq_train <- as.data.frame(trains[2])
```

Now we can take a look at the smaller dataset and see that it's ready to start modelling:

```
summary(sj_train)
```

```
##  reanalysis_specific_humidity_g_per_kg reanalysis_dew_point_temp_k
##  Min.   :11.72                         Min.   :289.6
##  1st Qu.:15.23                         1st Qu.:293.8
##  Median :16.83                         Median :295.4
##  Mean   :16.54                         Mean   :295.1
##  3rd Qu.:17.85                         3rd Qu.:296.4
##  Max.   :19.44                         Max.   :297.8
##  station_avg_temp_c station_min_temp_c city          year
##  Min.   :22.84      Min.   :17.80      iq:  0   Min.   :1990
##  1st Qu.:25.81      1st Qu.:21.70      sj:936   1st Qu.:1994
##  Median :27.21      Median :22.80               Median :1999
##  Mean   :27.00      Mean   :22.59               Mean   :1999
##  3rd Qu.:28.18      3rd Qu.:23.90               3rd Qu.:2003
##  Max.   :30.07      Max.   :25.60               Max.   :2008
##    weekofyear     total_cases
##  Min.   : 1.00   Min.   :  0.00
##  1st Qu.:13.75   1st Qu.:  9.00
##  Median :26.50   Median : 19.00
##  Mean   :26.50   Mean   : 34.18
##  3rd Qu.:39.25   3rd Qu.: 37.00
##  Max.   :53.00   Max.   :461.00
```

```
summary(iq_train)
```

```
##  reanalysis_specific_humidity_g_per_kg reanalysis_dew_point_temp_k
##  Min.   :12.11                         Min.   :290.1
##  1st Qu.:16.12                         1st Qu.:294.6
##  Median :17.43                         Median :295.9
##  Mean   :17.10                         Mean   :295.5
##  3rd Qu.:18.19                         3rd Qu.:296.6
##  Max.   :20.46                         Max.   :298.4
##  station_avg_temp_c station_min_temp_c city          year
```

```
##  Min.   :21.40      Min.   :14.70      iq:520   Min.   :2000
##  1st Qu.:27.00      1st Qu.:20.60      sj:  0   1st Qu.:2003
##  Median :27.57      Median :21.35               Median :2005
##  Mean   :27.52      Mean   :21.20               Mean   :2005
##  3rd Qu.:28.09      3rd Qu.:22.00               3rd Qu.:2007
##  Max.   :30.80      Max.   :24.20               Max.   :2010
##    weekofyear     total_cases
##  Min.   : 1.00   Min.   :  0.000
##  1st Qu.:13.75   1st Qu.:  1.000
##  Median :26.50   Median :  5.000
##  Mean   :26.50   Mean   :  7.565
##  3rd Qu.:39.25   3rd Qu.:  9.000
##  Max.   :53.00   Max.   :116.000
```

Split it up!

Since this is a timeseries model, we'll use a strict-future holdout set when we are splitting our train set and our test set. We'll keep around three quarters of the original data for training and use the rest to test. We'll do this separately for our San Juan model and for our Iquitos model.

```
# split up the data
sj_train_subtrain <- head(sj_train, 800)
sj_train_subtest  <- tail(sj_train, nrow(sj_train) - 800)

iq_train_subtrain <- head(iq_train, 400)
iq_train_subtest  <- tail(iq_train, nrow(sj_train) - 400)
```

Training time

This is where we start getting down to business. As we noted above, we'll train a NegativeBinomial model, which is often used for count data where the mean and the variance are very different. In this function we have three steps. The first is to specify the functional form

```
# function that returns Mean Absolute Error
mae <- function(error) return(mean(abs(error)) )

get_bst_model <- function(train, test)
{

  # Step 1: specify the form of the model
  form <- "total_cases ~ 1 +
            reanalysis_specific_humidity_g_per_kg +
            reanalysis_dew_point_temp_k +
            station_avg_temp_c +
            station_min_temp_c"

  grid = 10 ^(seq(-8, -3,1))

  best_alpha = c()
  best_score = 1000

  # Step 2: Find the best hyper parameter, alpha
  for (i in grid)
```

```
    {
      model = glm.nb(formula = form,
                     data = train,
                     init.theta = i)

      results <-  predict(model, test)
      score   <-  mae(test$total_cases - results)

      if (score < best_score) {
          best_alpha <- i
          best_score <- score
          cat('\nbest score = ', best_score, '\twith alpha = ', best_alpha)
        }
  }

  # Step 3: refit on entire dataset
  combined <- rbind(train, test)
  combined_model = glm.nb(formula=form,
                          data = combined,
                          init.theta = best_alpha)

  return (combined_model)
}
```

```
sj_model <- get_bst_model(sj_train_subtrain, sj_train_subtest)
```

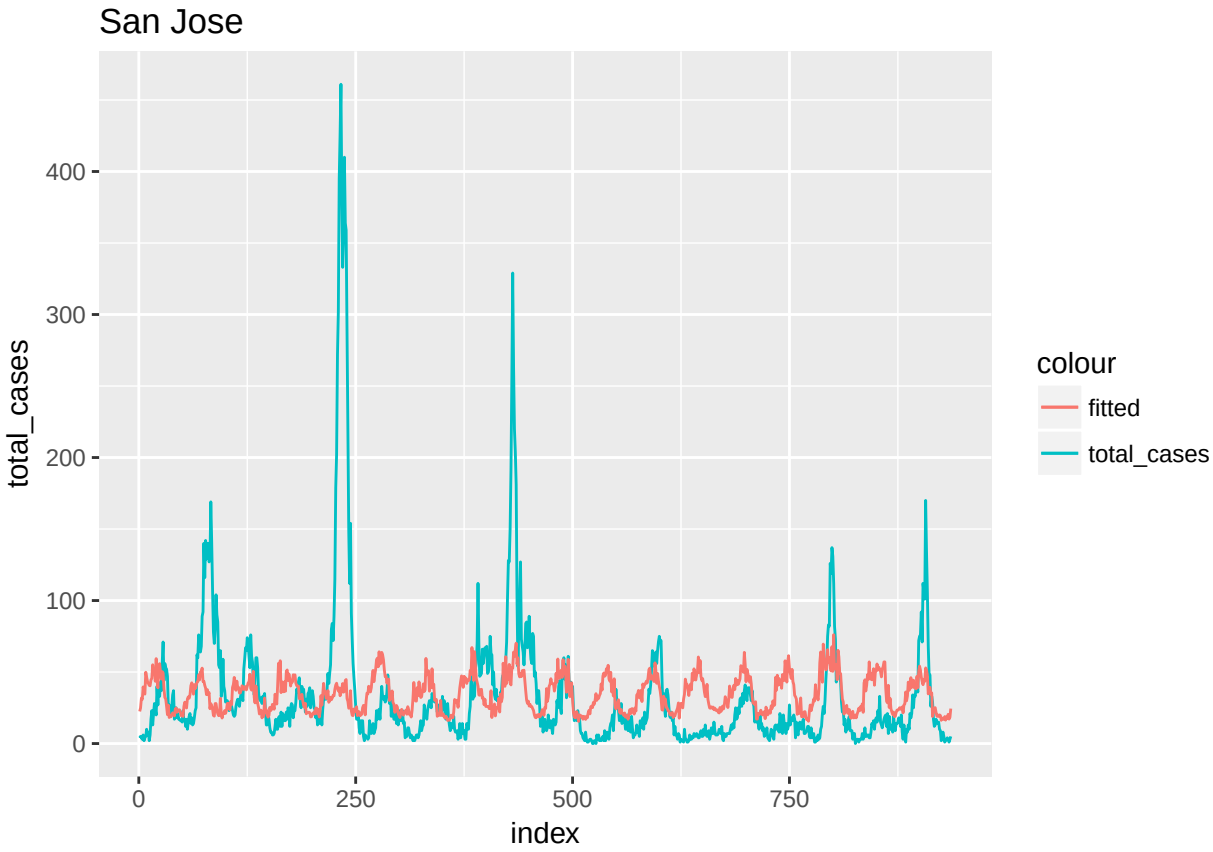
```
##
## best score =  21.01167   with alpha =  1e-08
```

```
iq_model <- get_bst_model(iq_train_subtrain, iq_train_subtest)
```

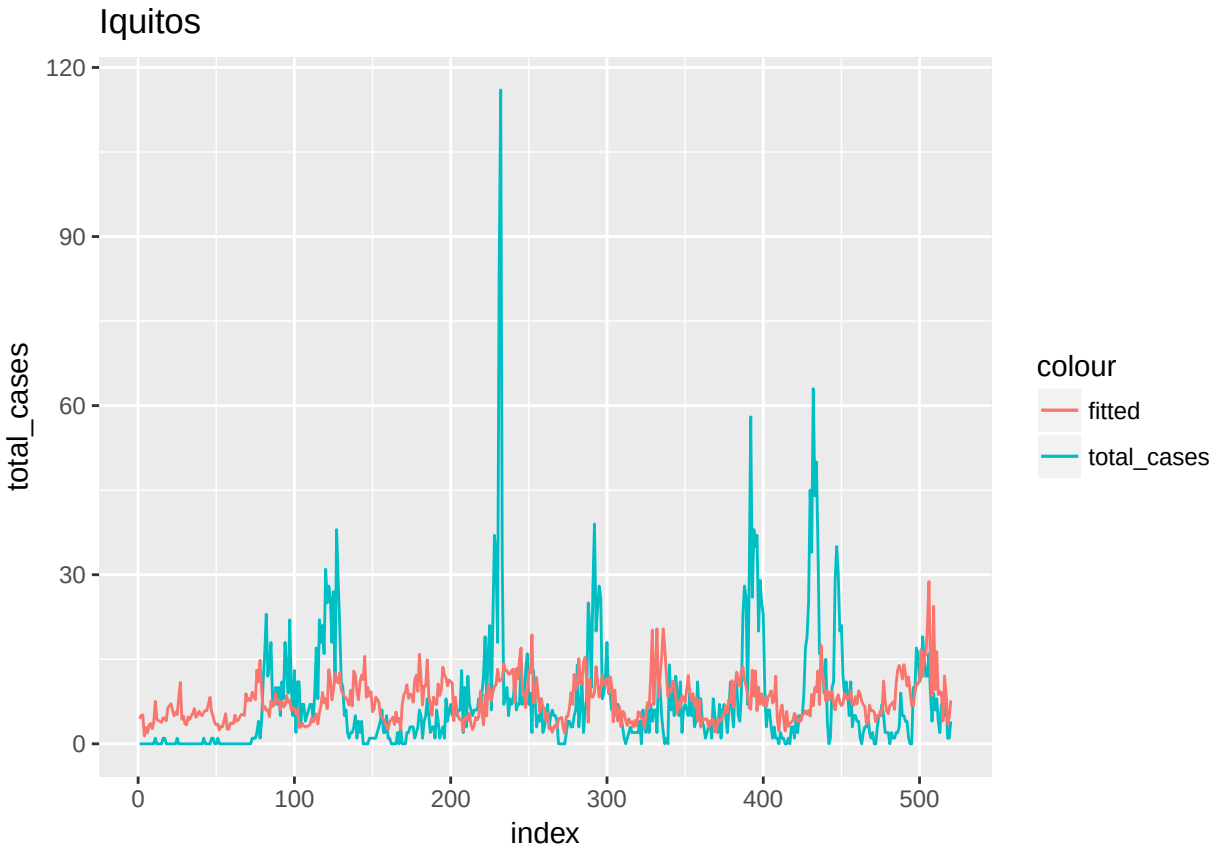
```
##
## best score =  6.421811   with alpha =  1e-08
## best score =  6.421811   with alpha =  1e-06
## best score =  6.421811   with alpha =  1e-05
```

Dengue Predicted Cases vs. Actual Cases

```
# plot sj
sj_train$fitted = predict(sj_model, sj_train, type = 'response')
sj_train %>%
  mutate(index = as.numeric(row.names(.))) %>%
  ggplot(aes(x = index)) + ggtitle("San Jose") +
  geom_line(aes(y = total_cases, colour = "total_cases")) +
  geom_line(aes(y = fitted, colour = "fitted"))
```




```
# plot iq
iq_train$fitted = predict(iq_model, iq_train, type = 'response')
iq_train %>%
  mutate(index = as.numeric(row.names(.))) %>%
  ggplot(aes(x = index)) + ggtitle("Iquitos") +
  geom_line(aes(y = total_cases, colour = "total_cases")) +
  geom_line(aes(y = fitted, colour = "fitted"))
```



Reflecting on our performance

These graphs can actually tell us a lot about where our model is going wrong and give us some good hints about where investments will improve the model performance. For example, we see that our model in blue does track the seasonality of Dengue cases. However, the timing of the seasonality of our predictions has a mismatch with the actual results. One potential reason for this is that our features don't look far enough into the past–that is to say, we are asking to predict cases at the same time as we are measuring percipitation. Because dengue is misquito born, and the misquito lifecycle depends on water, we need to take both the life of a misquito and the time between infection and symptoms into account when modeling dengue. This is a critical avenue to explore when improving this model.

The other important error is that our predictions are relatively consistent–we miss the spikes that are large outbreaks. One reason is that we don't take into account the contagiousness of dengue. A possible way to account for this is to build a model that progressively predicts a new value while taking into account the previous prediction. By training on the dengue outbreaks and then using the predicted number of patients in the week before, we can start to model this time dependence that the current model misses.

So, we know we're not going to win this thing, but let's submit the model anyway!

```
# submitting the predictions
tests <- preprocessData('dengue_features_test.csv')
```

```
## Parsed with column specification:
## cols(
##   .default = col_double(),
##   city = col_character(),
##   year = col_integer(),
```

```
##   weekofyear = col_integer(),
##   week_start_date = col_date(format = "")
## )
```

```
## See spec(...) for full column specifications.
```

```
sj_test <- tests[[1]]; iq_test <- tests[[2]]

sj_test$predicted = predict(sj_model , sj_test, type = 'response')
iq_test$predicted = predict(iq_model , iq_test, type = 'response')

submissions = read_csv('submission_format.csv')
```

```
## Parsed with column specification:
## cols(
##   city = col_character(),
##   year = col_integer(),
##   weekofyear = col_integer(),
##   total_cases = col_integer()
## )
```

```
inner_join(submissions, rbind(sj_test,iq_test)) %>%
  dplyr::select(city, year, weekofyear, total_cases = predicted) ->
predictions
```

```
## Joining, by = c("city", "year", "weekofyear")
```

```
predictions$total_cases %<>% round()
write.csv(predictions, 'submissions/predictions.csv', row.names = FALSE)
```