Using Regression and Classification Techniques on Chicago Crime Dataset

Rahil Morjaria, Codie Wood, Rachel Wood

January 2023

```
devtools::install_github("codiewood/chicago_crime/chicri")
```

```
library(chicri)
theme_set(theme_minimal())
cbpal <- c("#000000", "#E69F00", "#56B4E9", "#009E73","#F0E442", "#0072B2", "#D55E00",
                    "#CC79A7")
```

Computational Techniques

Package Building

Packages are a key part of producing reproducible R code. They include R functions which can be reused, the documentation that describes how to use them, and sometimes sample data. In the construction of our `chicri` package, we mainly used the `devtools` and `usethis` packages. We followed the guidelines and usage principles detailed in R Packages (Second Edition) by Hadley Wickham and Jenny Bryan, in order to ensure our package was consistent with widely used conventions.

Documentation

When documenting the various functions in the package, we used the `roxygen2` package. This package automatically creates `.Rd` documentation files for functions from comments added as part of the function definitions. Using this framework allows the code and documentation to coexist and be updated alongside eachother, ensuring consistency.

Application Programming Interface (API) Querying

The Chicago Crime dataset stored online is regularly updated on a daily basis. It has millions of rows of data, and each row contains a total of 22 features which have been recorded. As such, the full data is incredibly large and so costly to store. As such, we chose not to include data files in the package. Instead, we used the `RSocrata` package to interact with the online data portal and pull the data. The function `load_crimes_API()` allows the user to directly download the dataset into R, via the function `read.socrata()`, and also to specify a particular year from which the data should be loaded.

Testing, GitHub and Continuous Integration

For tests, we used the `testthat` package. This creates a new directory `./tests/testthat/` to be populated with unit tests for the package functions. Although the tests can be run alone, generally, we made use of the

`usethis::check()` command. This relies on the function `rcmdcheck::rcmdcheck()`, which not only runs the tests, but also carries out other checks for the package build. These include:

- Checking the package installs correctly,
- Checking for missing documentation,
- Checking for undefined variables,
- Checking for missing dependencies.

In order to ensure that tests and checks were run on a regular basis, continuous integration was set up using Github Actions. The status of this action can be seen in our package description markdown file. For each pull request and push to the main branch, this checked our package on multiple operating systems (Windows, Ubuntu and MacOS). This helped to ensure that when writing our code, we did not introduce any changes which broke functionality.

Parallel Programming

We used parallel programming, which allows code to be run across multiple cores, in order to speed up code with long run times. We use the package `doParallel` in order to implement this. In particular, we use parallel programming in the cross-validation for our binary classifiers, allowing faster assessment of model performance, and more models to be assessed in a shorter time frame.

Exploratory Data Analysis

Our data can be loaded using the `load_crimes_API()` function from the `chicri` package:

```
df <- load_crimes_API(year = NULL) %>%
  process_data()

df %>% head()
```

Looking at the columns, we can see there are many variables which contain similar information to each other, so we ignore the following columns in our analysis:

- `ID`, `Case Number`, `Updated On` : We assume these are for administrative purposes and are non-informative for predicting crimes and arrests.
- `Block`, `Beat`, `Ward`, `X Coordinate`, `Y Coordinate`, `Latitude`, `Longitude` : For the spatial elements of our analysis, we only use `Community Area` and `District` to reduce redundancy and computational time.
- `IUCR`, `FBI Code`, `Description` : These have more levels than `Primary Type` while containing similar information, and thus we use `Primary Type` instead.

We also drop any missing data.

```
df <- df %>%
  select(-c(ID, `Case Number`, `Updated On`,
            `Block`, `Beat`, `Ward`, `X Coordinate`, `Y Coordinate`, `Latitude`,
            `Longitude`, `IUCR`, `FBI Code`, `Description`)) %>%
  drop_na()
```

We also consider some categorical variables which have many levels, with the aim of combining some of the levels for the sake of computational time and readability.

The only time which use the following variables is analysis on data from 2019, and so we combine factors based on the counts from this year only.

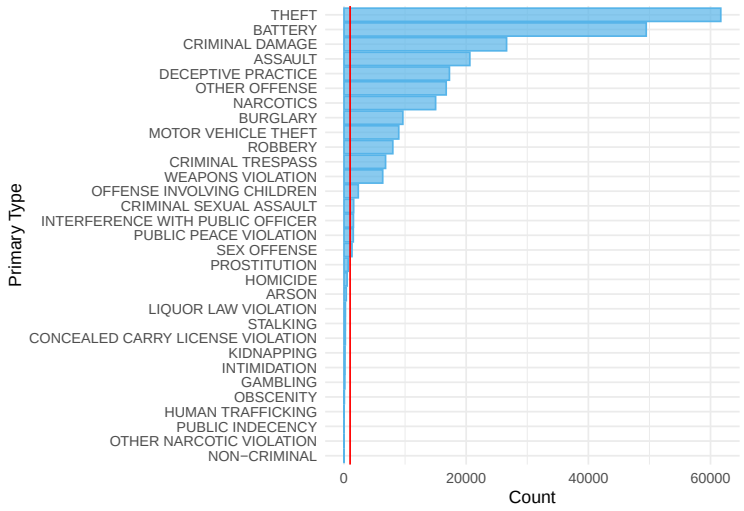
```
df_2019 <- df %>%
  filter(year(Date) == 2019)
```

Primary Type

We observe that there are initially 31 crime types as classified in our data set. Of these, 14 types have fewer than 1000 observations. In order to reduce the computational complexity of our model, we merge these uncommon levels of our factor variable into the level OTHER, in which we also include the pre-existing crime type OTHER OFFENSE. This leaves us with a total of 17 levels in our variable, with 7.57% of data values in the OTHER category. The least frequent category is SEX OFFENSE, which has 1310 occurrences.

The following plots show the distribution of values among the data before and after the combination of factors.

```
df_2019$`Primary Type` <- df_2019$`Primary Type` %>%
  fct_infreq() %>%
  fct_rev()

 # set threshold for othering
thr <- 1000

plot_factors(df_2019, "Primary Type", threshold = thr)
```



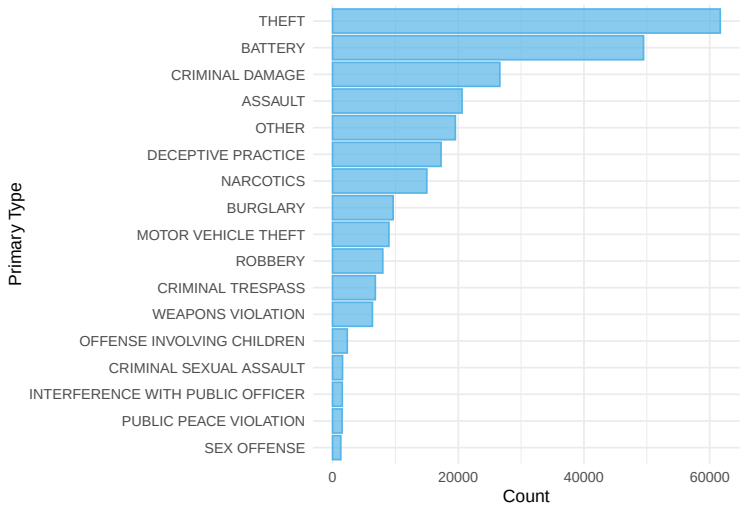

```
df_2019$`Primary Type` <- df_2019$`Primary Type` %>%
  fct_recode("OTHER" = "OTHER OFFENSE") %>%
  othering(thr,print_summary = T) %>%
  fct_infreq() %>%
  fct_rev()
```

```
## 14 out of 31 categories converted to OTHER, 7.57% of data values.
```

```
length(levels(df_2019$`Primary Type`))
```

```
## [1] 17
```

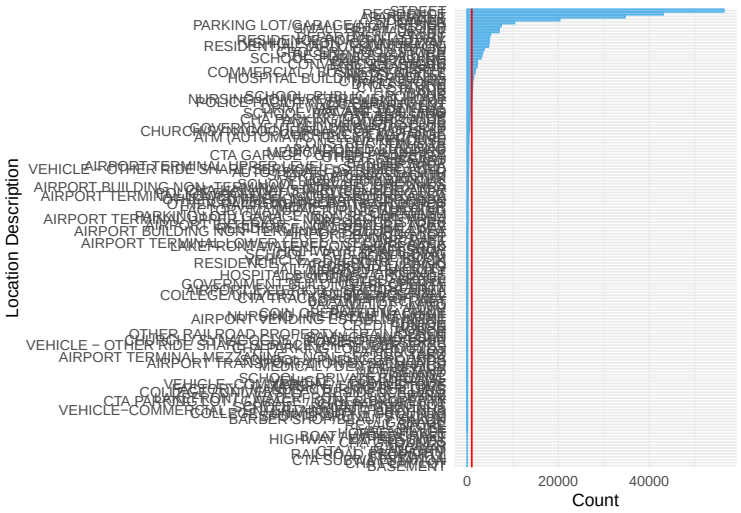
```
plot_factors(df_2019, "Primary Type")
```



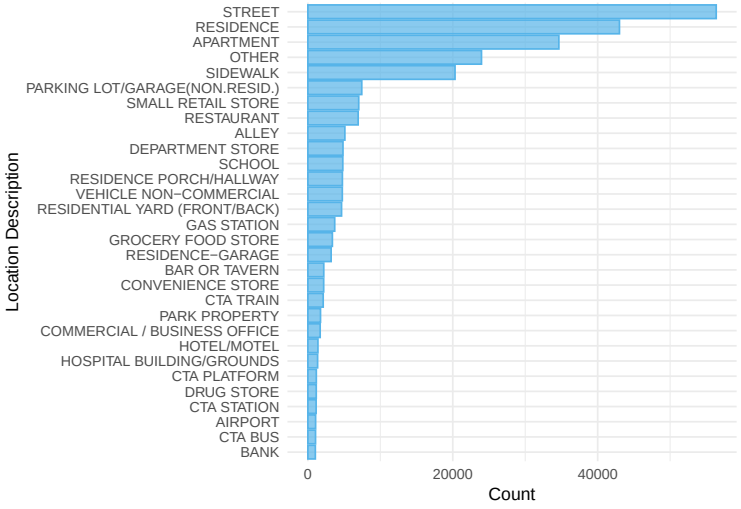
Location Description

We also wished to reduce the number of levels for the Location Descriptions in our data set. Initially, we have 156 location descriptions. Of these, 127 have less than 1000 observations. In order to reduce the computational complexity of our model, we merge these uncommon levels of our factor variable into the level OTHER, as well as merging some groups which have similar categories: for example, all descriptions containing the word "airport" were merged into one category.

```
# Reordering factor levels (for plots and analysis purposes)
df_2019$`Location Description` <- df_2019$`Location Description` %>%
  fct_infreq() %>%
  fct_rev()

plot_factors(df_2019, "Location Description", thr)
```




```
df_2019 <- regroup_locations(df_2019,thr)

df_2019$`Location Description` <- df_2019$`Location Description` %>%
  fct_infreq() %>%
  fct_rev()

plot_factors(df_2019, "Location Description")
```

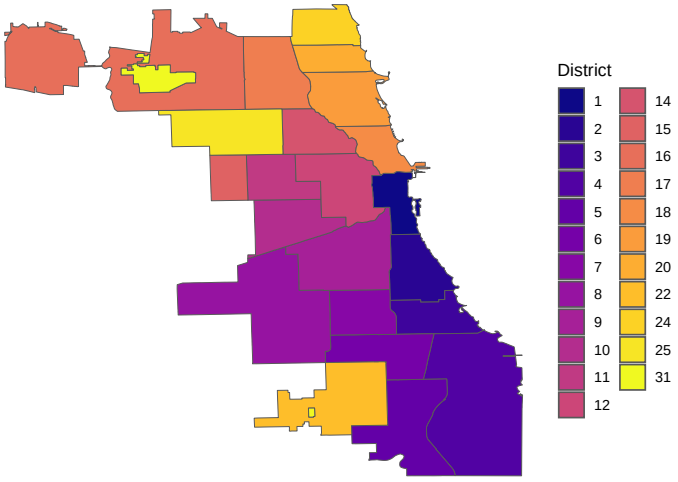


Even after this removal of categories, after some exploration we found that inclusion in the models led to an inability for the model to converge, and so we remove this from our final analysis.

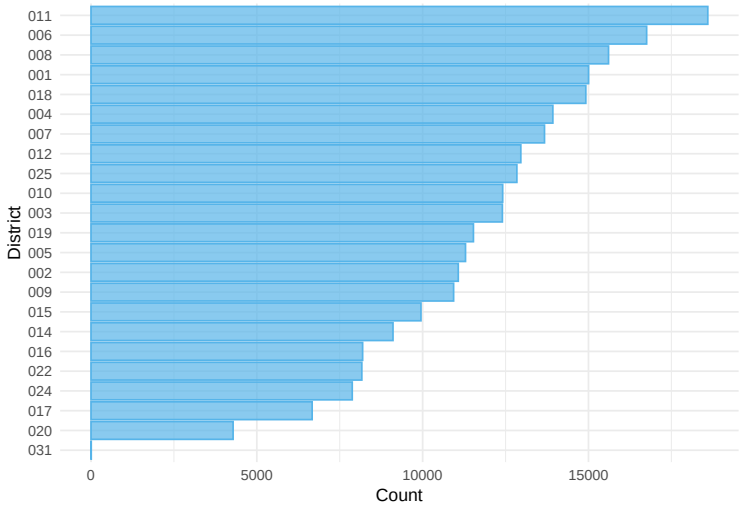
District

The following plots show a map of the district bounds and the distribution of crimes among districts.

```
district_map <- get_map(map = "district")

plot_heat_map_d(district_map,"District")
```




```
df_2019$District <- df_2019$District %>%
  fct_infreq() %>%
  fct_rev()

plot_factors(df_2019, "District")
```



We remove District 031 due to inconsistencies in definition, we will see in the plot below that the District Area isn't properly defined, and due to the very low amount of observations we remove it from our analysis.

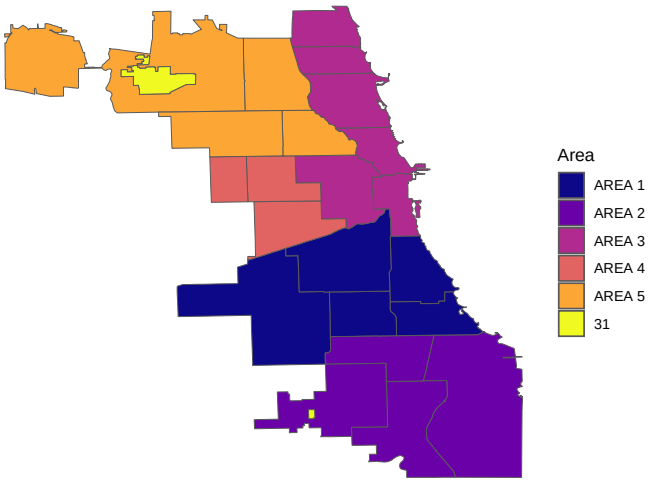
```
# Removing district 31 from analysis

df_2019 <- df_2019 %>%
  filter(!(District == "031")) %>%
  mutate(District = fct_drop(District))
```

In order to reduce the computational complexity of our model, whilst still maintaining location information, we can also consider grouping districts further into policing district areas, of which there are 5. [1]

[1] Source: http://oururbantimes.com/district-stations/command-changes-12th-and-14th-chicago-police-department-districts

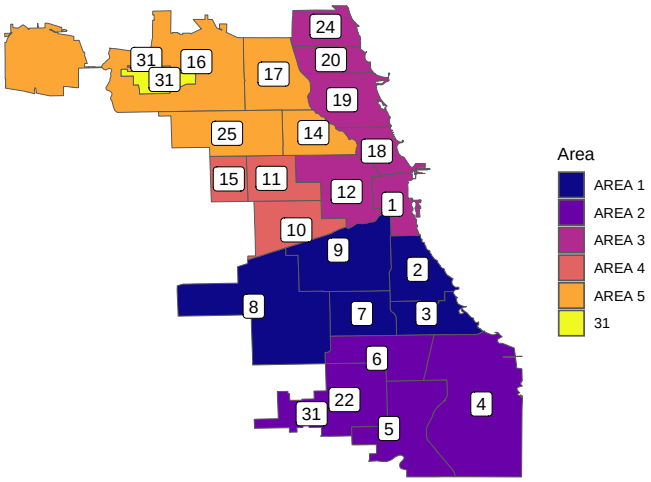
```r
# Adding area groupings
area1 <- c("002","003","008","009","007")
area2 <- c("004","005","006","022")
area3 <- c("001","012","018","019","020","024")
area4 <- c("010","011","015")
area5 <- c("014","016", "017","025")

df_2019 <- df_2019 %>%
  mutate(`District Area` = fct_collapse(District,
                                        "AREA 1" = area1,
                                        "AREA 2" = area2,
                                        "AREA 3" = area3,
                                        "AREA 4" = area4,
                                        "AREA 5" = area5),
         `District Area` = fct_relevel(`District Area`, sort))
```

```r
district_map$Area <- fct_collapse(as.factor(district_map$District),
                                  "AREA 1" = as.character(as.integer(area1)),
                                  "AREA 2" = as.character(as.integer(area2)),
                                  "AREA 3" = as.character(as.integer(area3)),
                                  "AREA 4"= as.character(as.integer(area4)),
                                  "AREA 5" = as.character(as.integer(area5))) %>%
  fct_relevel(sort) %>%
  fct_relevel("31", after = Inf)

#Plot for just area
area_plot <- plot_heat_map_d(district_map,"Area")
area_plot
```

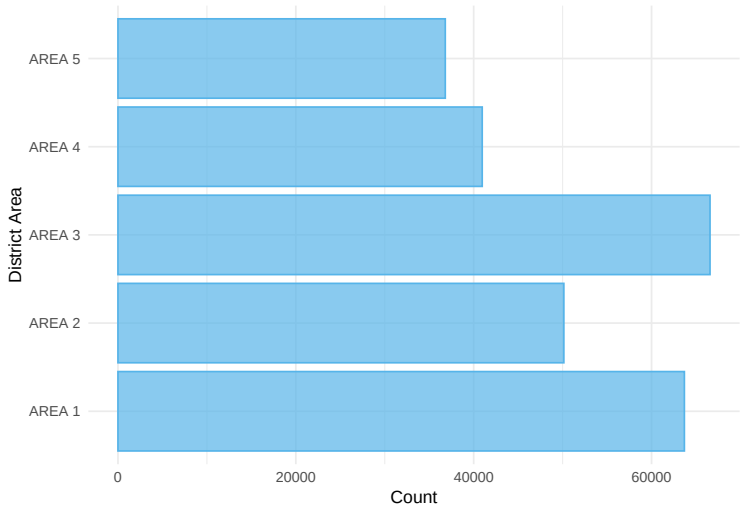



```r
# Get district labels and locations
district_labels <- sort(as.integer(district_map$District))
district_markers <- district_map %>%
  dplyr::arrange(as.integer(District)) %>%
  sf::st_centroid() %>%
  sf::st_coordinates() %>%
```

```
  as_tibble() %>%
  dplyr::mutate(district_labels = district_labels)

# Plot with district labels and coloured area
area_plot_markers <- area_plot +
  geom_sf_label(data = district_map, aes(label = dist_num)) +
  theme_void()

area_plot_markers
```




```
# Plot showing number of crimes in each district area
plot_factors(data = df_2019, "District Area")
```



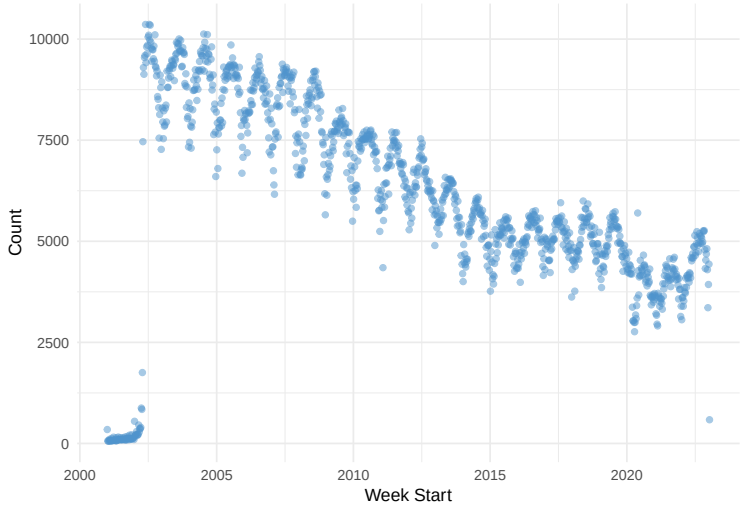
Predicting Crimes Spatially and Temporally

For this task we implement a general additive model (GAM) using the `gam()` function from the `mgcv` package.

Weekly Predictions

We use the `count_cases()` function to obtain weekly and monthly counts

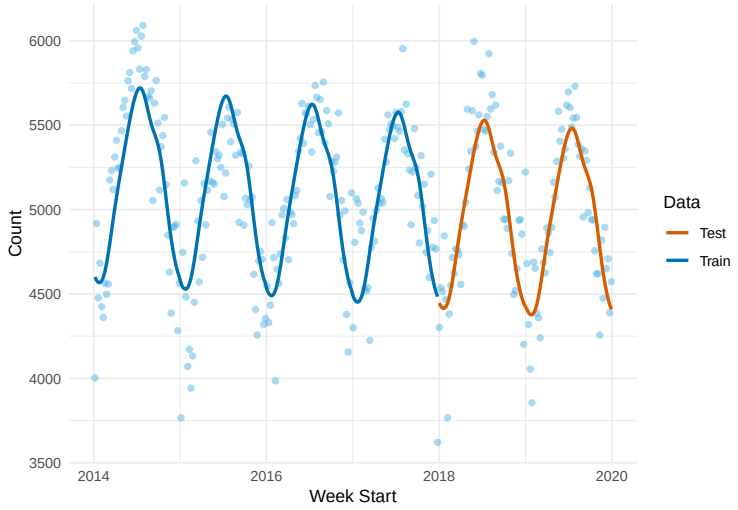
```
week_count <- df %>% count_cases(date_level = "week")

ggplot(data =week_count) +
  geom_point(aes(x= `Week Start`, y = Count), color = "steelblue3", alpha = 0.5)
```



We can see that there is a drop in early 2020 (likely due to the COVID pandemic), and so we only consider data from 2019 or earlier when fitting our model. We can also see the trend of the data changes at around 2014 so we only use data from this point forward to train our model. We then split the data into testing and training sets:

```
library(mgcv)

week_count <- week_count %>% filter(Year <= 2019, Year >= 2014)
week_train <- week_count %>% filter(Year < 2018)
week_test <- week_count %>% filter(Year >= 2018)

model_week <- gam(Count ~ s(Week, bs = "cc") + Year,
                  data = week_train, family = "poisson")

week_train <- week_train %>%
  cbind("Predicted" = model_week$fitted.values, "Data" = "Train")

week_test <- week_test %>%
  cbind("Predicted" = as.numeric(predict(model_week, week_test,
                                         type = "response")), "Data" = "Test")


week_dat <- rbind(week_train, week_test)


p <-ggplot(data = week_dat) +
  geom_point(aes(x = `Week Start` , y = Count), color = cbpal[3], alpha =0.5) +
```

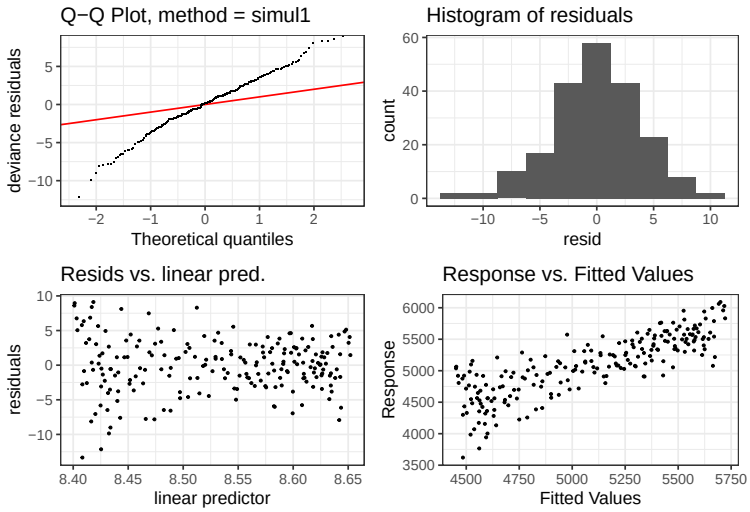
```
  geom_line(aes(x = `Week Start`, y = Predicted, color = Data), size = 1) +
  scale_colour_manual(values= cbpal[c(7,6)])
p
```



While this model looks reasonable, we also perform some model checking. The `draw()` function allows us to visualise the partial effects of the model:

```
library(gratia)
library(patchwork)
library(mgcViz)

check <- model_week %>% getViz()

check(check,
      a.qq = list(a.cipoly = list(fill = "light blue")),
      a.respoi = list(size = 0.5),
      a.hist = list(bins = 10)
      )
```

```
##
## Method: UBRE   Optimizer: outer newton
## full convergence after 5 iterations.
## Gradient range [6.745428e-06,6.745428e-06]
## (score 14.16042 & scale 1).
## Hessian positive definite, eigenvalue range [0.003478659,0.003478659].
## Model rank =  10 / 10
##
## Basis dimension (k) checking results. Low p-value (k-index<1) may
## indicate that k is too low, especially if edf is close to k'.
##
##           k'  edf k-index p-value
## s(Week) 8.00 7.46    0.94    0.18
```



Weekly Predictions in Space

We can now consider how the crime rates vary within Chicago using the community bounds. The shapefiles for these are available on the Chicago Data Portal, and the `chicri` package contains a function for loading them.

To include them in our `gam` model however, we need to use a Markov Random Field smooth. This creates an adjacency matrix for the community areas (based on whether any two areas share a vertex).

```
library(RSocrata)
library(sf)
library(chicri)
community_map <- get_map() %>%
  mutate(Area = as.factor(Area))

community_list <- setNames(as.list(community_map$the_geom),
                           as.character(community_map$Area))

n <- nrow(community_map)
for (i in 1:n){
  object <- community_list[[i]]
  #class(object)
  community_list[[i]] <- unlist(st_coordinates(object), recursive =TRUE)
}
names(community_list) <- as.character(community_map$Area)
xt <- list(polys = community_list)
```

As before, we obtain the count data, this time with the community area included, and obtain testing and training data set

```
spatial_count <- df %>% count_cases(location_level = "community_area") %>%
  mutate(CommunityArea = as.factor(`Community Area`)) %>%
  select(- `Community Area`)
```

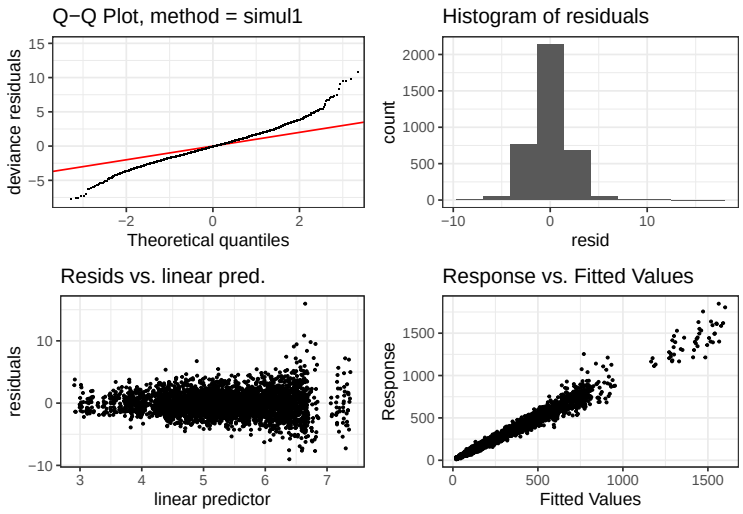
```r
spatial_count <- spatial_count %>% filter(Year <= 2019, Year >= 2014)
spatial_train <- spatial_count %>% filter(Year < 2018)
spatial_test <- spatial_count %>% filter(Year >= 2018)


spatial_model <-
  gam(Count ~ s(Month, bs = "cc") + Year + s(CommunityArea, bs = "mrf", xt = xt),
      data = spatial_train, family = "poisson")



summary(spatial_model)
```

```
##
## Family: poisson
## Link function: log
##
## Formula:
## Count ~ s(Month, bs = "cc") + Year + s(CommunityArea, bs = "mrf",
##     xt = xt)
##
## Parametric coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept) 20.2408053  1.7493186  11.571   <2e-16 ***
## Year        -0.0074086  0.0008679  -8.536   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##                      edf Ref.df Chi.sq p-value
## s(Month)           7.969      8   7785  <2e-16 ***
## s(CommunityArea) 75.984     76 574080  <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.974   Deviance explained = 98.1%
## UBRE = 2.5954  Scale est. = 1         n = 3696
```

```r
check_spatial <- spatial_model %>% getViz()

check(check_spatial,
      a.qq = list(a.cipoly = list(fill = "light blue")),
      a.respoi = list(size = 0.5),
      a.hist = list(bins = 10)
      )
```

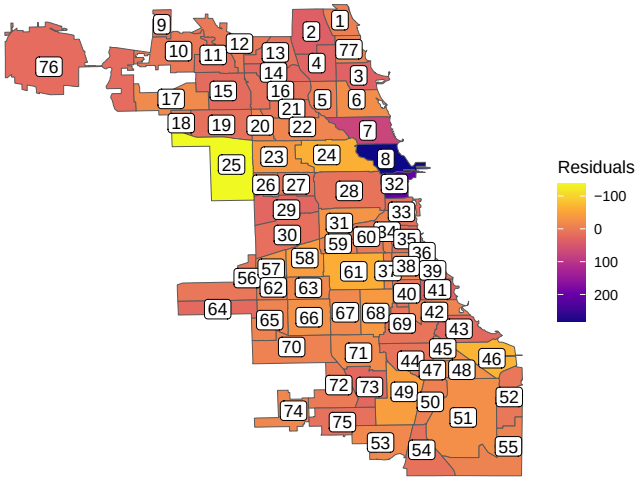
```
##
## Method: UBRE   Optimizer: outer newton
## full convergence after 15 iterations.
## Gradient range [7.31684e-16,3.298548e-06]
## (score 2.59535 & scale 1).
## Hessian positive definite, eigenvalue range [1.541818e-05,1.639796e-05].
## Model rank =  86 / 86
```

```
##
## Basis dimension (k) checking results. Low p-value (k-index<1) may
## indicate that k is too low, especially if edf is close to k'.
##
##                       k'   edf k-index p-value
## s(Month)           8.00  7.97     0.8  <2e-16 ***
## s(CommunityArea) 76.00 75.98      NA      NA
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```



We can also visualise the residuals on a map for the first month of the test data:

```
plot_dat <- spatial_test %>%
  filter(Year ==2018, Month == 1)


plot_dat <- plot_dat %>%
  cbind("Fitted" = predict(spatial_model,plot_dat, type = "response"))

plot_dat <- plot_dat %>%cbind( "Residuals" = (plot_dat$Count - plot_dat$Fitted) )

plot_dat <- left_join(community_map, plot_dat, by = c("Area" = "CommunityArea"))

res_plot <- plot_heat_map(plot_dat, "Residuals", trans = "reverse")

res_plot +
  geom_sf_label(data = plot_dat, aes(label = Area), label.padding = unit(0.15,"lines"))
```



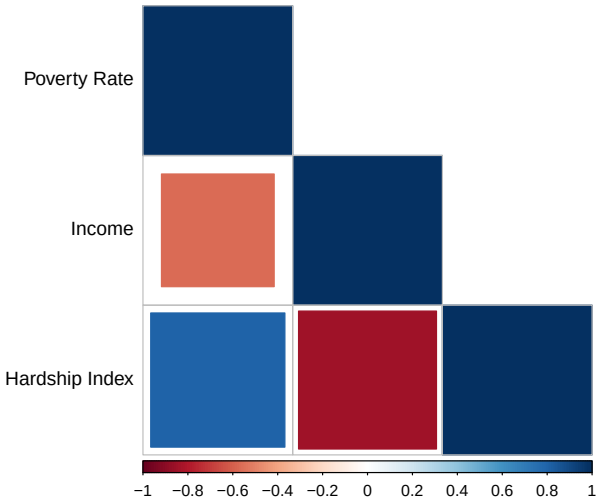
We can see the model significantly underestimates counts in areas 7, 8 and 32, and over estimates the counts in area 25. However overall the model performs well.

Incorporating Socio-Economic Indicators

The Chicago Data Portal also contains a dataset with three variables for each community area:

- `percent_households_below_poverty`: Percentage of households living the federal poverty line
- `per_capita_income_`: This is an estimation, calculated by aggregating incomes and dividing by the total population
- `hardship_index`: A score from 1-100 (a higher score indicates a greater level of hardship), incorporating 6 socio-economic indicators.

```
library(corrplot)
socio_ind <- get_indicators()


socio_ind %>% select(-`Community Area`) %>% cor() %>%
  corrplot(method = "square", type = "lower", tl.pos = "l", tl.col = "black")
```



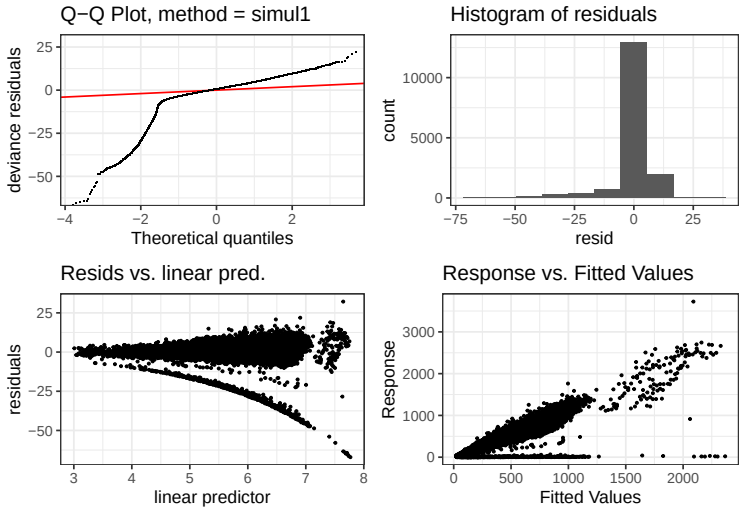
We can see that all of these values are highly correlated, and so we only need to include one in an analysis. For this we choose the hardship index as it is made up partly of the other two indicators. We now fit the model with the same method as before:

```
spatial_count <- df %>% count_cases(location_level = "community_area") %>%
  rename("CommunityArea" = "Community Area") %>%
  mutate(CommunityArea = as.factor(CommunityArea))

socio_data <- spatial_count %>% left_join(socio_ind,
                                          by = c("CommunityArea" = "Community Area")) %>%
  rename(HardshipIndex = `Hardship Index`, `PovertyRate` = `Poverty Rate`)

socio_train <- socio_data %>% filter(Year < 2019)
socio_test <- socio_data %>% filter(Year >= 2019)

socio_model <-
  gam(Count ~ s(Month, bs = "cc") + Year + s(CommunityArea, bs = "mrf", xt = xt) +
        HardshipIndex, data = socio_train, family = "poisson")

summary(socio_model)
```

```
##
## Family: poisson
## Link function: log
##
## Formula:
## Count ~ s(Month, bs = "cc") + Year + s(CommunityArea, bs = "mrf",
##     xt = xt) + HardshipIndex
##
## Parametric coefficients:
##                 Estimate Std. Error  z value Pr(>|z|)
## (Intercept)    3.872e+01  1.685e-01  229.747  < 2e-16 ***
## Year          -1.658e-02  7.832e-05 -211.654  < 2e-16 ***
## HardshipIndex  3.396e-03  1.219e-03    2.787  0.00532 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##                     edf Ref.df  Chi.sq p-value
## s(Month)          7.984      8   67005  <2e-16 ***
## s(CommunityArea) 74.963     75 2999756  <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.766   Deviance explained = 77.2%
## UBRE = 65.662  Scale est. = 1         n = 16487
```

```
check_socio <- socio_model %>% getViz()

check(check_socio,
      a.qq = list(a.cipoly = list(fill = "light blue")),
      a.respoi = list(size = 0.5),
      a.hist = list(bins = 10)
      )
```

```
##
## Method: UBRE   Optimizer: outer newton
## full convergence after 12 iterations.
## Gradient range [6.817589e-07,7.968835e-05]
## (score 65.66182 & scale 1).
## Hessian positive definite, eigenvalue range [3.291674e-06,0.00016332].
## Model rank =  87 / 87
##
## Basis dimension (k) checking results. Low p-value (k-index<1) may
## indicate that k is too low, especially if edf is close to k'.
##
##                       k'   edf k-index p-value
## s(Month)            8.00  7.98       1    0.62
## s(CommunityArea) 76.00 74.96      NA      NA
```



We can see that the socio-economic data contributes meaningfully to the model.

Binary Classification

In this section we attempt to fit a binary classification model, to predict given certain information whether the crime would lead to an arrest or not. Choosing only to use 2019 data, due to scale of the data set (leaving us with roughly 250,000 observations), as data recorded after is affected strongly by COVID.

The features we choose to include in this model are:

$$\mathbf{x} = (\texttt{Domestic, Primary Type, Day, Time, Week Day, District})$$

We choose `Primary Type` over `FBI Code` to determine the type of crime as during our EDA we found inconsistencies with the classification of `FBI Code`. With `Primary Type`, we merge uncommon levels of our factor variable into the level OTHER, in which we also include the pre-existing crime type OTHER OFFENSE.

For temporal data we include `Week Day` as a factor variable, encoding date using `Day` (1-365) and `Time` as a numeric character between 0-1.

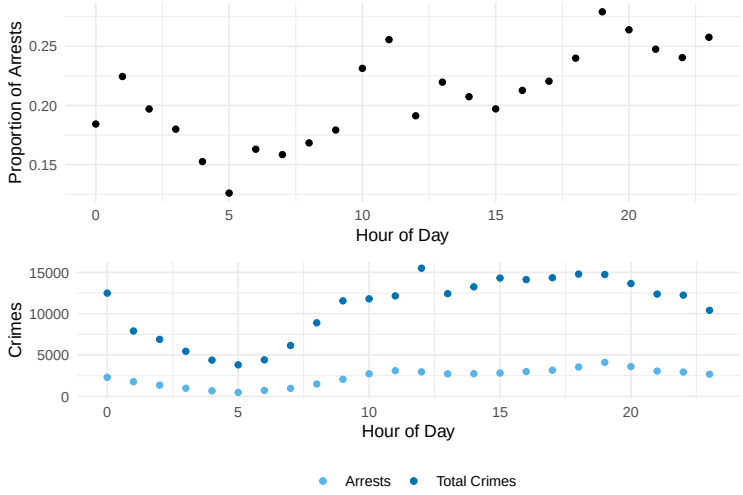
In order to include spatial information, we consider the `District` variable.

While including more information might improve our models, during our EDA we found the inclusion of highly correlated variables to improve the model insignificantly while drastically increasing computational time.

```
crimes <- df_2019 %>%
  mutate(day = yday(Date),
         wkday = wday(Date),
         Hour = as.numeric(hour(Date) + minute(Date)/60)/24)


time.data <- crimes %>%
  mutate(Hour = as.factor(hour(crimes$Date))) %>%
  group_by(Hour) %>%
  dplyr::count(Arrest)

crimes <- crimes %>%
  mutate(wkday = as.factor(wkday)) %>%
  droplevels() %>%
  drop_na()
```

Here we plot the proportion of crimes that resulted in arrest for depending on the hour of the day. We can see that the likelihood of an Arrest Resulting from a crime might have dependence on the hour the crime was committed this could be due to the nature of the crimes). However, this seems to indicates that it would be worthwhile to include time as a predictor in our binary classification model.

```
library(ggplot2)
library(cowplot)

arrest_rates <- crimes %>%
  mutate(Hour = hour(Date)) %>%
  dplyr::group_by(Hour) %>%
  dplyr::summarise(`Total Crimes` = n(),
                   Arrests = sum(Arrest),
                   `Arrest Prob` = mean(Arrest))

crimes <- crimes %>%
  select(- c(Date,  'X Coordinate','Y Coordinate','District Area',
             'Location Description'))

plot1 <- ggplot(arrest_rates) +
  geom_point(aes(x =Hour, y = `Arrest Prob`)) +
  labs(y = 'Proportion of Arrests', x= 'Hour of Day')

plot2 <- ggplot(arrest_rates) +
  geom_point(aes(x =Hour, y = Arrests, color = 'Arrests' )) +
  geom_point(aes(x =Hour, y = `Total Crimes`, color = 'Total Crimes'))+
  labs(y = 'Crimes', x= 'Hour of Day')+
  scale_color_manual(values = cbpal[c(3,6)]) +
  theme(legend.position = "bottom", legend.title = element_blank())

plot_grid(plot1, plot2, nrow=2)
```



Logistic Regression

```
library(caret)
library(doParallel)
library(boot)
library(kernlab)
set.seed(30700) # Setting seed for reproducibility

crimes <- crimes %>% mutate(Arrest = as.factor(Arrest))
```

Using the package `doParallel` and `caret` we can easily fit classification models. We define a control method which allows our function to do 5-fold Cross Validation, using `doParallel` to do the cross validation in parallel to decrease computational time. Setting seeds for our cross-validation function to ensure reproducibility. The function will return the model with the highest accuracy, however we can see the accuracy for the other models in the cross validation process.

```
#Create Seeds for the Cross Validation Function
seeds <- vector(mode = "list", length = 6)
for(i in 1:6) seeds[[i]] <- sample.int(n=1000, 3)

#Define Cross Validation Function
control <- trainControl(method = 'cv', number = 5, allowParallel = TRUE,
                        seeds=seeds)

cl <- makePSOCKcluster(detectCores() - 1)

registerDoParallel(cl)

#Fitting Logistic Regression Classifier
log.model <- train(Arrest ~ .,
                   data = crimes,
                   trControl = control,
                   method = "glm",
                   family=binomial())
```

```
stopCluster(cl)
```

Support Vector Machine

In this section we fit Support Vector Machines with the Radial Basis Kernel (due to the unclear relationship between our predictors and class).

As SVM models scale badly with data points we have to train our models on samples of our data set (which has greater than $250,000$ data points). We will be fitting 3 SVM models, each using a different sampling technique.

- The first method is to take a proportional sample, which attempts to generate a sample where the proportion of each classes emulates the original data set.
- The next is known as 'Down Sampling' where we randomly sample the majority class till we have the same number of observations as the minority class.
- Similarly we will use the method known as 'Up Sampling', randomly samples (with replacement) the minority class until we have the same number of observations as the majority class.

It is clear than in our data there is a significant imbalance in the class proportions, only 21.7 of the data has `Arrest = TRUE`. For our first SVM model, we will take a representative sample of roughly 13,000 data points. The `createDataPartition()` function in `caret` allows us to easily do this.

```
trainIndex <- createDataPartition(crimes$Arrest, p = .05,
                                  list = TRUE)

crimesTest <- crimes[trainIndex$Resample,]
```

Just like with the Logistic Regression Classifier we use the `train` function from `caret` using our previously defined train function - which has the exact same implementation for the SVMs we fit. The `svmRadial` method uses `kernlab::sigest` to determine a good predictor for σ. It will then test multiple values for Cost and will choose the one which maximizes accuracy. The `train` function will work in this manner for the other 2 models we train as well.

```
cl <- makePSOCKcluster(detectCores() - 1)

registerDoParallel(cl)

#Fitting SVM Model with a Proportional Sample
svm.model <- train(Arrest ~ .,
                   data = crimesTest,
                   trControl = control,
                   method = 'svmRadial',
                   family = binomial(),
                   allowParallel = TRUE)

stopCluster(cl)
```

As we will be taking a somewhat small sample of our whole data set to train this model on, we will now try a method called down sampling, this a method of sampling a data set in which we take all of the data points from our lowest proportion class, and sample an equal amount of the other proportions of the classes.

First we use the `downSample` function from `caret` this takes a sample of our data set, giving us an equal sample of both `Arrest = TRUE` and `Arrest = FALSE`.

```
down_sample <- downSample(x = crimes, y = crimes$Arrest)
down_sample %>% count(Arrest)
```

```
##   Arrest     n
## 1  FALSE 56071
## 2   TRUE 56071
```

We see that the sample involves all the TRUE data points and gives a random sample of our FALSE data points, we can now use the `createDataPartition` function to take a representative sample of `down_sample` (of equal size to our other sample).

```
trainIndexD <- createDataPartition(down_sample$Arrest, p = nrow(crimesTest)/nrow(down_sample),
                                   list = TRUE)
crimesTestD <- down_sample[trainIndexD$Resample, ]  %>% select(-Class)

cl <- makePSOCKcluster(detectCores() - 1)

registerDoParallel(cl)

#Training a SVM using down sampling
svm.modelD <- train(Arrest ~ .,
                    data = crimesTestD,
                    trControl = control,
                    method = 'svmRadial',
                    family = binomial())

stopCluster(cl)
```

There is another method of sampling in a similar vein to down sampling, known as up sampling where we sample with replacement the non majority classes until they are of the same length as the majority class in a similar manner with down sampling we will train a model.

```
up_sample <- upSample(x = crimes, y = crimes$Arrest)
up_sample %>% count(Arrest)
```

```
##   Arrest      n
## 1  FALSE 202098
## 2   TRUE 202098
```

```
trainIndexU <- createDataPartition(up_sample$Arrest, p = nrow(crimesTest)/nrow(up_sample),
                                   list = TRUE)

crimesTestU <- up_sample[trainIndexU$Resample, ] %>% select(-Class)

cl <- makePSOCKcluster(detectCores() - 1)

registerDoParallel(cl)

#Training a SVM using up sampling
```

```
svm.modelU <- train(Arrest ~ .,
                    data = crimesTestU,
                    trControl = control,
                    method = 'svmRadial',
                    family = binomial())

stopCluster(cl)
```

Model Performance

To test the performance of our Models, we will simulate 10 proportional (in respect to number of arrests), testing data sets. For each of these testing sets we will some metrics. The metrics below will be ones measured for each of models.

$$Accuracy = Pr(\hat{Y} = Y)$$
$$Sensitivity = Pr(\hat{Y} = 1|Y = 1) = Pr(\text{True Positive})$$
$$Specificity = Pr(\hat{Y} = 0|Y = 0) = Pr(\text{True Negative})$$
$$\text{Balanced Accuracy} = \frac{Sensitivity + Specificity}{2}$$

An interesting benchmark for the usefulness of the model is the No-Information Rate which is the accuracy if we just guess for each data point is the largest proportion class. For the crimes data set that is 0.7828, however this is not some foolproof benchmark, for example if our goal is to predict to lowest proportion class consistently.

```
blank <- data.frame(Model = as.character(), ACC=numeric(0),SENS = numeric(0),
                    SPEC = numeric(0), BALACC = numeric(0))
up <- blank
down <- blank
logistic <- blank
repSVM <- blank

#Creating data frames of our metrics
for (testrow in 1:10){
  sampIndex <- createDataPartition(crimes$Arrest, p = .025,
                                   list = TRUE)

  sampTest <- crimes[sampIndex$Resample,]

  u <- confusionMatrix(data = predict(object = svm.modelU, newdata = sampTest),
                       reference = sampTest$Arrest, positive = 'TRUE')
  d <- confusionMatrix(data = predict(object = svm.modelD, newdata = sampTest),
                       reference = sampTest$Arrest, positive = 'TRUE')
  l <- confusionMatrix(data = predict(object = log.model, newdata = sampTest),
                       reference = sampTest$Arrest, positive = 'TRUE')
  r <- confusionMatrix(data = predict(object =  svm.model, newdata = sampTest),
                       reference = sampTest$Arrest, positive = 'TRUE')

  up[testrow, ] <-
    c('UpSVM',u[["overall"]][["Accuracy"]], u[["byClass"]][["Sensitivity"]],
              u[["byClass"]][["Specificity"]], u[["byClass"]][["Balanced Accuracy"]])
```

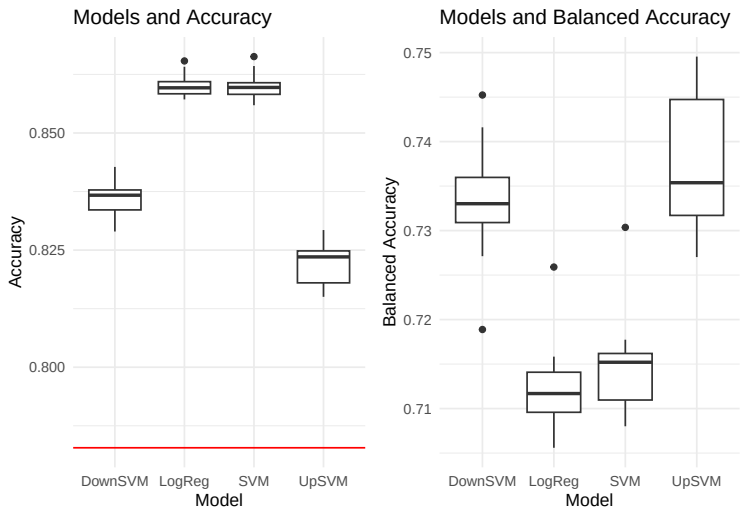
```
  down[testrow, ] <-
    c('DownSVM',d[["overall"]][["Accuracy"]], d[["byClass"]][["Sensitivity"]],
              d[["byClass"]][["Specificity"]], d[["byClass"]][["Balanced Accuracy"]])

  logistic[testrow, ] <-
    c('LogReg',l[["overall"]][["Accuracy"]], l[["byClass"]][["Sensitivity"]],
              l[["byClass"]][["Specificity"]], l[["byClass"]][["Balanced Accuracy"]])

  repSVM[testrow, ] <-
    c('SVM',r[["overall"]][["Accuracy"]], r[["byClass"]][["Sensitivity"]],
              r[["byClass"]][["Specificity"]], r[["byClass"]][["Balanced Accuracy"]])
}
```

Below we have a table of the mean values of the metrics previously defined for each model.

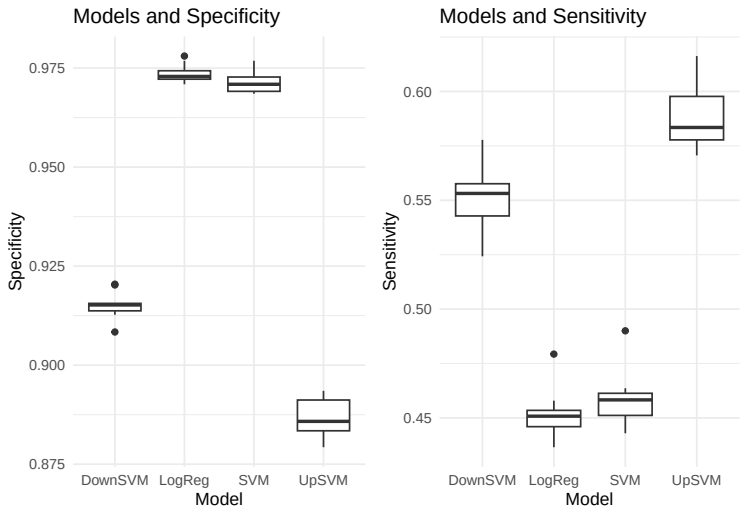
```
FullBinTest <- rbind(up,down,logistic,repSVM) %>%  mutate(Model = as.factor(Model),
                                                          ACC = as.numeric(ACC),
                                                          SENS = as.numeric(SENS),
                                                          SPEC = as.numeric(SPEC),
                                                          BALACC = as.numeric(BALACC))

FullBinTest %>% group_by(Model) %>% summarise(across(everything(), mean))
```

```
## # A tibble: 4 x 5
##   Model     ACC  SENS  SPEC BALACC
##   <fct>   <dbl> <dbl> <dbl>  <dbl>
## 1 DownSVM 0.836 0.551 0.915  0.733
## 2 LogReg  0.860 0.452 0.974  0.713
## 3 SVM     0.860 0.459 0.971  0.715
## 4 UpSVM   0.822 0.588 0.887  0.738
```

```
ACCPlot <- ggplot(FullBinTest, aes(x = Model, y = ACC)) + geom_boxplot() +
  geom_hline(mapping = aes(yintercept = 0.7828), color = 'red') +
  labs(title = 'Models and Accuracy', y = 'Accuracy')
SENSPlot <- ggplot(FullBinTest, aes(x = Model, y = SENS)) + geom_boxplot() +
  labs(title = 'Models and Sensitivity', y = 'Sensitivity')
SPECPlot <- ggplot(FullBinTest, aes(x = Model, y = SPEC)) + geom_boxplot() +
  labs(title = 'Models and Specificity', y = 'Specificity')
BALACCPlot <- ggplot(FullBinTest, aes(x = Model, y = BALACC)) + geom_boxplot() +
  labs(title = 'Models and Balanced Accuracy', y = 'Balanced Accuracy')

plot_grid(ACCPlot,BALACCPlot , nrow=1, greedy = TRUE)
```




```
plot_grid(SPECPlot, SENSPlot, nrow=1, greedy = TRUE)
```



We can see that both the proportional SVM Model and Logistic Model performed very similarly, both with high accuracy and specificity, this is to be expected, as both models were trained and tested with far more `FALSE` data points. We can see that with the trade-off being in sensitivity, both performing poorly at distinguishing positive data points.

Another important point, is that we might have expected the SVM model to perform better than the Logistic Regression Model, due to it's abilities to take into account non-linear relationships; however, we can see they performed very similarly in nearly all aspects, this is probably due to SVM being trained on only 5% of the 2019 data set while our Logistic Model was trained on the full 2019 data set.

As expected up and down sampling had a positive affect on the sensitivity of the model due to the training data involving a larger proportion of `TRUE` values. If we look at the down sampled model, we see that the high sensitivity gives a large trade off with specificity and accuracy (pushing it below the no information rate), this model is far more likely than the other models to give a false positive. This could be due to the down sample involving sampling the `FALSE` data points possibly giving a less informative sample.

Compared to down sampling, up sampling did not increase the sensitivity as much, but instead found a more balanced trade off with specificity and accuracy - this is interesting given that both models were trained on

the same amount of `TRUE` and `FALSE` data points, further analysis would be needed to determine whether this was due to an intrinsic property of the sampling technique or just due to randomness.

The better balanced accuracy of the up and down sampled models, shows that accuracy of the Logistic and proportional models comes heavily from it's lack of sensitivity combined with the imbalanced class proportions, and that depending on what the goal of our models it might be preferable to pick a less overall accurate model.

Multi-class Classification

In this section we consider the multi-class classification task of predicting the type of a reported crime, given data on the crime. We also extend this to address the secondary question of if we can predict the time of day at which a crime occurred, given data on the crime. We will focus here on the method of multinomial logistic regression for both of these tasks.

We will select several feasible models, and use multiple metrics to assess their performance. We will use repeated k-fold cross validation to calculate average values of these metrics across different training sets. We will then compare our models using these averages.

Due to the extremely large size of the full data set, it is computationally intensive to train models. As such, we instead train and test the multinomial logistic regression models only on data from 2019:

```
dat <- df_2019
```

As well as the data pre-processing described earlier in the report, we also manipulate the date variable for our purposes.

Time and Date

In order to incorporate temporal data, we include the date as the day of the year, `Day`, as well as time of day,`Time`. We standardize both of these variables to take values between 0 and 1. This is to ensure the convergence rate is not slow when we fit our model.

In order to facilitate our second model, which attempts to predict the time of day at which a crime occurs, we also add the variable `Time of Day` which encodes the time as a categorical variable with the following levels:

- `MORNING`: between 5am and 12 noon
- `AFTERNOON`: between 12 noon and 5pm
- `EVENING`: between 5pm and 9pm
- `NIGHT`: between 9pm and 5am

After this processing, we then remove the `Date` variable.

```
t <- lubridate::hour(dat$Date) + lubridate::minute(dat$Date)/60
tod <- rep("NIGHT",length(t))
for(i in 1:length(t)){
  if(5 <= t[i] & t[i] < 12){
    tod[i] <- "MORNING"
    }
  else if(12 <= t[i] & t[i] < 17){
    tod[i] <- "AFTERNOON"
```

```
    }
  else if(17 <= t[i] & t[i] < 21){
    tod[i] <- "EVENING"
    }
}
tod <- as.factor(tod)

dat <- dat %>%
  mutate(dat,
         Day = yday(dat$Date) / 365,
         Time = t/24,
         `Time of Day` = tod) %>%
  select(-Date)
```

Multinomial Logistic Regression

We use the `multinom()` function from the `nnet` package in order to implement multinomial logistic regression. We calculate various model performance metrics using the `confusionMatrix()` function from the `caret` package. Namely we focus here on calculating overall accuracy and no-information rate (NIR), as well as class specific sensitivity, specification and accuracy values. We implement 5-fold repeated cross validation, with $n = 5$ repeats, and calculate the average of each metric across all folds for each repeat.

```
#setting seed for reproducibility
set.seed(200899)
metrics <- c("Sensitivity", "Specificity", "Balanced Accuracy")
```

Predicting Crime Type

We begin by splitting our data into our response variable and the variables we want to include in our models. We look at three potential models:

1. No Location: Primary Type ~ Arrest + Domestic + Day + Time
2. District Area: Primary Type ~ Arrest + Domestic + Day + Time + District Area
3. District: Primary Type ~ Arrest + Domestic + Day + Time + District

```
# Splitting response data and different model variable sets
y <- dat$`Primary Type`
X_noloc <- dat %>% select(c(Arrest, Domestic, Day, Time))
X_area <- dat %>% select(c(Arrest, Domestic, Day, Time, `District Area`))
X_district <- dat %>% select(c(Arrest, Domestic, Day, Time, District))
```

We evaluate the model performance using the `mnlr_kfold_cv.df()` function, which outputs a data frame containing the metrics as evaluated for the model fitted on each fold for each repeat. Our package also contains the `mnlr_kfold_cv.list()` function, which performs the same operations and outputs a list based format of our values, as well as their averages. We use the data frame function for our analysis, however, as its output can be easily manipulated to calculate averages, investigate metric distributions. Further, the data frame outputs can be easily converted into an easy to read table, ideal for producing result documentation, which the list output cannot.

```
mnlr_results_noloc.df <- mnlr_kfold_cv.df(X = X_noloc, y = y, k = 5,
                                          n_reps = 5, metrics = metrics)
```

```
mnlr_results_area.df <- mnlr_kfold_cv.df(X = X_area, y = y, k = 5,
                                         n_reps = 5, metrics = metrics)
```

```
mnlr_results_district.df <-
  mnlr_kfold_cv.df(X = X_district, y = y, k = 5,
                   n_reps = 5, metrics = metrics)
```

We then create tables of our results, using the `mean_report_metrics()` function, and then for the class means we also average over the repeats.

```
noloc_class_means <- mean_repeat_metrics(mnlr_results_noloc.df,
                                         type = "class")
noloc_overall_means <- mean_repeat_metrics(mnlr_results_noloc.df,
                                           type = "overall")
```

```
noloc_avg_class_means <- noloc_class_means %>%
  dplyr::summarise("Average Sensitivity" = mean(`Average Sensitivity`),
                   "Average Specificity" = mean(`Average Specificity`),
                   "Average Balanced Accuracy" = mean(`Average Balanced Accuracy`))
```

We use the `knitr::kable()` functionality, as well as the `kableExtra` package, to export our results into a format suitable for our report: namely, a latex table. This is contained within the `read_table()` function.

```
noloc_kab <- report_table(noloc_overall_means, rep("c",4))
noloc_kab
```

Repeat	Average Overall Accuracy	Average NIR	Maximum P Value (Acc > NIR)
1	0.357	0.2389	0
2	0.357	0.2389	0
3	0.357	0.2389	0
4	0.357	0.2389	0
5	0.357	0.2389	0

```
noloc_kab_class <- report_table(noloc_avg_class_means, c("l",rep("c",3)))
noloc_kab_class
```

We repeat this process for each of our 3 models.

```
area_class_means <- mean_repeat_metrics(mnlr_results_area.df,
                                        type = "class")
area_overall_means <- mean_repeat_metrics(mnlr_results_area.df,
                                          type = "overall")
```

```
area_avg_class_means <- area_class_means %>%
  dplyr::summarise("Average Sensitivity" = mean(`Average Sensitivity`),
                   "Average Specificity" = mean(`Average Specificity`),
                   "Average Balanced Accuracy" = mean(`Average Balanced Accuracy`))
```

Class	Average Sensitivity	Average Specificity	Average Balanced Accuracy
ASSAULT	0.0000	1.0000	0.5000
BATTERY	0.4820	0.9073	0.6946
BURGLARY	0.0000	1.0000	0.5000
CRIMINAL DAMAGE	0.0000	1.0000	0.5000
CRIMINAL SEXUAL ASSAULT	0.0000	1.0000	0.5000
CRIMINAL TRESPASS	0.0000	1.0000	0.5000
DECEPTIVE PRACTICE	0.0000	1.0000	0.5000
INTERFERENCE WITH PUBLIC OFFICER	0.0000	1.0000	0.5000
MOTOR VEHICLE THEFT	0.0000	1.0000	0.5000
NARCOTICS	0.9993	0.8621	0.9307
OFFENSE INVOLVING CHILDREN	0.0000	1.0000	0.5000
OTHER	0.0000	1.0000	0.5000
PUBLIC PEACE VIOLATION	0.0000	1.0000	0.5000
ROBBERY	0.0000	1.0000	0.5000
SEX OFFENSE	0.0000	1.0000	0.5000
THEFT	0.8649	0.4242	0.6446
WEAPONS VIOLATION	0.0000	1.0000	0.5000

```
area_kab <- report_table(area_overall_means, rep("c",4))
area_kab
```

Repeat	Average Overall Accuracy	Average NIR	Maximum P Value (Acc > NIR)
1	0.3642	0.2389	0
2	0.3642	0.2389	0
3	0.3642	0.2389	0
4	0.3642	0.2389	0
5	0.3642	0.2389	0

```
area_kab_class <- report_table(area_avg_class_means,c("l",rep("c",3)))
area_kab_class
```

```
district_class_means <- mean_repeat_metrics(mnlr_results_district.df,
                                            type = "class")
district_overall_means <- mean_repeat_metrics(mnlr_results_district.df,
                                              type = "overall")
```

```
district_avg_class_means <- district_class_means %>%
  dplyr::summarise("Average Sensitivity" = mean(`Average Sensitivity`),
                   "Average Specificity" = mean(`Average Specificity`),
                   "Average Balanced Accuracy" = mean(`Average Balanced Accuracy`))
```

```
district_kab <- report_table(district_overall_means, rep("c",4))
district_kab
```

```
district_kab_class <- report_table(district_avg_class_means, c("l",rep("c",3)))
district_kab_class
```

Class	Average Sensitivity	Average Specificity	Average Balanced Accuracy
ASSAULT	0.0000	1.0000	0.5000
BATTERY	0.4822	0.9071	0.6946
BURGLARY	0.0000	1.0000	0.5000
CRIMINAL DAMAGE	0.0002	0.9999	0.5000
CRIMINAL SEXUAL ASSAULT	0.0000	1.0000	0.5000
CRIMINAL TRESPASS	0.0000	1.0000	0.5000
DECEPTIVE PRACTICE	0.0000	1.0000	0.5000
INTERFERENCE WITH PUBLIC OFFICER	0.0000	1.0000	0.5000
MOTOR VEHICLE THEFT	0.0000	1.0000	0.5000
NARCOTICS	0.9252	0.8979	0.9115
OFFENSE INVOLVING CHILDREN	0.0000	1.0000	0.5000
OTHER	0.0000	1.0000	0.5000
PUBLIC PEACE VIOLATION	0.0000	1.0000	0.5000
ROBBERY	0.0000	1.0000	0.5000
SEX OFFENSE	0.0000	1.0000	0.5000
THEFT	0.9126	0.3898	0.6512
WEAPONS VIOLATION	0.0000	1.0000	0.5000

Repeat	Average Overall Accuracy	Average NIR	Maximum P Value (Acc > NIR)
1	0.3648	0.2389	0
2	0.3648	0.2389	0
3	0.3647	0.2389	0
4	0.3647	0.2389	0
5	0.3647	0.2389	0

Class	Average Sensitivity	Average Specificity	Average Balanced Accuracy
ASSAULT	0.0000	1.0000	0.5000
BATTERY	0.4823	0.9070	0.6946
BURGLARY	0.0000	1.0000	0.5000
CRIMINAL DAMAGE	0.0010	0.9995	0.5003
CRIMINAL SEXUAL ASSAULT	0.0000	1.0000	0.5000
CRIMINAL TRESPASS	0.0000	1.0000	0.5000
DECEPTIVE PRACTICE	0.0000	1.0000	0.5000
INTERFERENCE WITH PUBLIC OFFICER	0.0000	1.0000	0.5000
MOTOR VEHICLE THEFT	0.0000	1.0000	0.5000
NARCOTICS	0.9169	0.9023	0.9096
OFFENSE INVOLVING CHILDREN	0.0000	1.0000	0.5000
OTHER	0.0000	1.0000	0.5000
PUBLIC PEACE VIOLATION	0.0000	1.0000	0.5000
ROBBERY	0.0000	1.0000	0.5000
SEX OFFENSE	0.0000	1.0000	0.5000
THEFT	0.9163	0.3865	0.6514
WEAPONS VIOLATION	0.0034	0.9994	0.5014

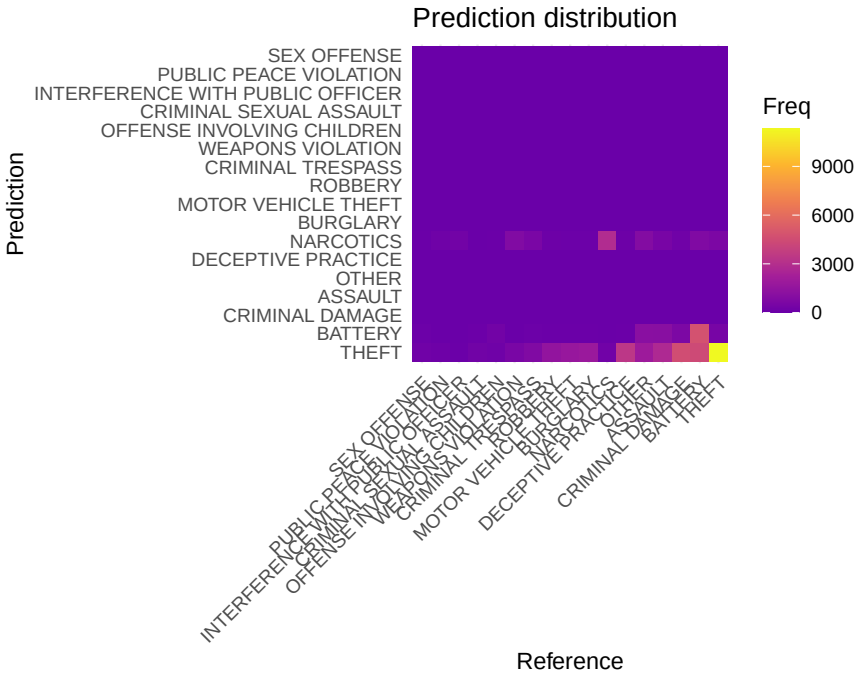
We observe that, whilst there is a progressive increase in the overall accuracy of the models, the increase from using `District Area` to using `District` is very marginal. As such, we prefer to use the model which includes location as `District Area` as this balances accuracy with computational cost.

Next, we perform further analysis on our chosen model. We refit our model using a training data set, using an 80-20 split of training to test data.

```
set.seed(1)
ind <- sample(2,nrow(dat),replace=TRUE, prob = c(0.8,0.2))
index <- which(ind == 2)
mnlr <- mnlr_cv_indexed(X_area,y,index,return_model = TRUE)
```

We can look at a subset of the predictive probabilities produced by the model, as well as the coefficient values, in order to interpret the relationships between our observed variables and the log odds of a particular class. We can also extract data from our `confusionMatrix` object to produce a confusion matrix plot, which shows the frequency of various prediction and reference combinations.

```
# Extract confusion matrix for plot
tab <- mnlr$conf.matrix$table
confusion_df <- as.data.frame(tab)

confusion_plot <- ggplot(confusion_df, aes(x=Reference, y = Prediction, fill=Freq)) +
  coord_equal() +
  geom_tile() +
  labs(title = "Prediction distribution") +
  scale_fill_viridis_c(option = "plasma", begin = 0.2) +
  scale_y_discrete(limits = rev) +
  scale_x_discrete(expand = expansion(mult = c(0,0)), guide = guide_axis(angle = 45))
confusion_plot
```



Predicting Time of Crime

As an extension to this multiclass classification task, we will also evaluate the performance of another multinomial regression classifier used to predict the time of day at which a crime occurred. We use the same methods as seen previously but now time of day is response variable and we exclude Time from our model. This gives us the three models:

1. No Location: Time of Day ~ Primary Type + Arrest + Domestic + Day
2. District: Time of Day ~ Primary Type + Arrest + Domestic + Day + District
3. District Area: Time of Day ~ Primary Type + Arrest + Domestic + Day + District Area

We repeat the exact steps as when we considered the `Primary Type` response variable. We observed here that, when fitting our models, convergence occurred for the first two models but not for the third. This was in contrast to the models of type, where we saw no examples of convergence.

```
#setting seed for reproducibility
set.seed(200899)
# Splitting response data and different model variable sets
y_tod <- dat$`Time of Day`
X_tod_noloc <- dat %>% select(c(`Primary Type`,Arrest,Domestic,Day))
X_tod_area <- dat %>% select(c(`Primary Type`,Arrest,Domestic,Day,`District Area`))
X_tod_district <- dat %>% select(c(`Primary Type`,Arrest,Domestic,Day,District))
```

```
tod_results_noloc <-
  mnlr_kfold_cv.df(X = X_tod_noloc, y = y_tod, k = 5,
                   n_reps = 5, metrics = metrics)
```

```
tod_results_area <-
  mnlr_kfold_cv.df(X = X_tod_area, y = y_tod, k = 5,
                   n_reps = 5, metrics = metrics)
```

```
tod_results_district <-
  mnlr_kfold_cv.df(X = X_tod_district, y = y_tod, k = 5,
                   n_reps = 5, metrics = metrics)
```

```
tod_noloc_class_means <- mean_repeat_metrics(tod_results_noloc,
                                             type="class")
tod_noloc_overall_means <- mean_repeat_metrics(tod_results_noloc,
                                               type="overall")
```

```
tod_noloc_avg_class_means <- tod_noloc_class_means %>%
  dplyr::summarise("Average Sensitivity" = mean(`Average Sensitivity`),
                   "Average Specificity" = mean(`Average Specificity`),
                   "Average Balanced Accuracy" = mean(`Average Balanced Accuracy`))
```

```
tod_noloc_kab <- report_table(tod_noloc_overall_means, rep("c",4))
tod_noloc_kab
```

Repeat	Average Overall Accuracy	Average NIR	Maximum P Value (Acc > NIR)
1	0.3346	0.2796	0
2	0.3346	0.2796	0
3	0.3346	0.2796	0
4	0.3346	0.2796	0
5	0.3346	0.2796	0

```
tod_noloc_kab_class <- report_table(tod_noloc_avg_class_means, c("l",rep("c",3)))
tod_noloc_kab_class
```

Class	Average Sensitivity	Average Specificity	Average Balanced Accuracy
AFTERNOON	0.5113	0.6007	0.5560
EVENING	0.0755	0.9476	0.5115
MORNING	0.0871	0.9370	0.5121
NIGHT	0.5723	0.6051	0.5887

```
tod_area_class_means <- mean_repeat_metrics(tod_results_area,type="class")

tod_area_overall_means <- mean_repeat_metrics(tod_results_area,type="overall")
```

```
tod_area_avg_class_means <- tod_area_class_means %>%
  dplyr::summarise("Average Sensitivity" = mean(`Average Sensitivity`),
                   "Average Specificity" = mean(`Average Specificity`),
                   "Average Balanced Accuracy" = mean(`Average Balanced Accuracy`))
```

```
tod_area_kab <- report_table(tod_area_overall_means, rep("c",4))
tod_area_kab
```

Repeat	Average Overall Accuracy	Average NIR	Maximum P Value (Acc > NIR)
1	0.3353	0.2796	0
2	0.3349	0.2796	0
3	0.3347	0.2796	0
4	0.3350	0.2796	0
5	0.3349	0.2796	0

```
tod_area_kab_class <- report_table(tod_area_avg_class_means, c("l",rep("c",3)))
tod_area_kab_class
```

Class	Average Sensitivity	Average Specificity	Average Balanced Accuracy
AFTERNOON	0.5208	0.5947	0.5577
EVENING	0.0643	0.9562	0.5103
MORNING	0.0855	0.9375	0.5115
NIGHT	0.5747	0.6019	0.5883

```
tod_district_class_means <- mean_repeat_metrics(tod_results_district,
                                                type="class")

tod_district_overall_means <- mean_repeat_metrics(tod_results_district,
                                                  type="overall")
```

```
tod_district_avg_class_means <- tod_district_class_means %>%
  dplyr::summarise("Average Sensitivity" = mean(`Average Sensitivity`),
                   "Average Specificity" = mean(`Average Specificity`),
                   "Average Balanced Accuracy" = mean(`Average Balanced Accuracy`))
```

```
tod_district_kab <- report_table(tod_district_overall_means, rep("c",4))
tod_district_kab
```

Repeat	Average Overall Accuracy	Average NIR	Maximum P Value (Acc > NIR)
1	0.3356	0.2796	0
2	0.3353	0.2796	0
3	0.3350	0.2796	0
4	0.3349	0.2796	0
5	0.3356	0.2796	0

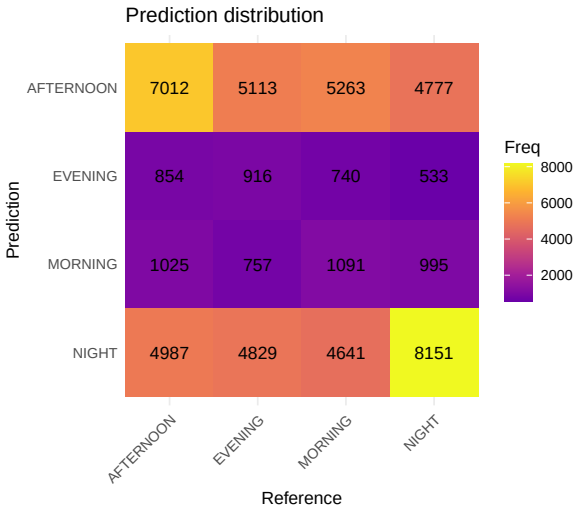
```
tod_district_kab_class <- report_table(tod_district_avg_class_means,
                                       c("l",rep("c",3)))
tod_district_kab_class
```

Class	Average Sensitivity	Average Specificity	Average Balanced Accuracy
AFTERNOON	0.5381	0.5812	0.5596
EVENING	0.0472	0.9683	0.5078
MORNING	0.0858	0.9351	0.5105
NIGHT	0.5725	0.6056	0.5891

Each of the models have similar overall accuracy across our repeats, so we prefer the simplest model with the fewest variables as this has minimal computational cost to run.

```
set.seed(1)
ind <- sample(2,nrow(dat),replace=TRUE, prob = c(0.8,0.2))
index <- which(ind == 2)
tod_mnlr <- mnlr_cv_indexed(X_tod_noloc,y_tod,index,return_model = TRUE)
```

```
# Extract confusion matrix for plot
tod_tab <- tod_mnlr$conf.matrix$table
tod_confusion_df <- as.data.frame(tod_tab)

tod_confusion_plot <- ggplot(tod_confusion_df, aes(x=Reference,
                                                   y = Prediction, fill=Freq)) +
  coord_equal() +
  geom_tile() +
  scale_fill_viridis_c(option = "plasma", begin = 0.2) +
  labs(title = "Prediction distribution") +
  scale_y_discrete(limits = rev) +
  scale_x_discrete(expand = expansion(mult = c(0,0)), guide = guide_axis(angle = 45)) +
  geom_text(aes(label=Freq), color="black") # printing values

tod_confusion_plot
```


Prediction distribution
AFTERNOON
EVENING
MORNING
NIGHT
7012
5113
5263
4777
854
916
740
533
1025
757
1091
995
4987
4829
4641
8151
Prediction
Reference
Freq
8000
6000
4000
2000