NYPD Shooting Incident Data Report Victim/Perpetrator Relationship

Jacky Luo

8/3/2023

Topic

This document will primarily be an analysis of the demographics of victims and perpetrator of historical shooting incidents within the jurisdiction of the NYPD.

The data can be found **here**.

Import Data and Libraries

Edit the block below to select your mirror to download packages from (if required).

```
## [1] "No mirror selected, using default mirror."
```

```
## Selected Mirror:  http://lib.stat.cmu.edu/R/CRAN/
```

If you would like to manually install the required packages, here is a list:

1. `tidyverse`
2. `reshape`
3. `fastDummies`
4. `xgboost`
5. `caret`
6. `MLmetrics`
7. `naniar`
8. `ggplot2`
9. `purrr`
10. `lubridate`

If not, the packages will be installed by the code below.

```
## [1] "Data and Packages Imported Successfully."
```

Investigate Data & Data Quality

View Columns

```
#View column names
names(df) %>% print()
```

```
##  [1] "INCIDENT_KEY"            "OCCUR_DATE"
##  [3] "OCCUR_TIME"              "BORO"
##  [5] "LOC_OF_OCCUR_DESC"       "PRECINCT"
##  [7] "JURISDICTION_CODE"       "LOC_CLASSFCTN_DESC"
##  [9] "LOCATION_DESC"           "STATISTICAL_MURDER_FLAG"
## [11] "PERP_AGE_GROUP"          "PERP_SEX"
## [13] "PERP_RACE"               "VIC_AGE_GROUP"
## [15] "VIC_SEX"                 "VIC_RACE"
## [17] "X_COORD_CD"              "Y_COORD_CD"
## [19] "Latitude"                "Longitude"
## [21] "Lon_Lat"
```

The first 9 columns (`INCIDENT_KEY`, `OCCUR_DATE`, `OCCUR_TIME`, `BORO`, `LOC_OF_OCCUR_DESC`, `PRECINCT`, `JURISDICTION_CODE`, `LOC_CLASSFCTN_DESC`, `LOCATION_DESC`) are related to the time and general location of the incident. This includes jurisdictions which can be assumed to be administrative details for the police.

There appears to be a boolean flag relating to statistical murder (`STATISTICAL_MURDER_FLAG`) which needs to be further investigated.

There are then three columns which describe the perpetrator, followed by three columns which describe the victim of the incident. The descriptors are age group, race and sex for both perpetrator and victim.

The final five columns are related to the precise location of the incident. They include a X/Y coordinate system, a latitude, a longitude and a combined latitude/longitude pair to form a point.

Preview Data Head

```
#View first five rows with all columns
head(df) %>% print(width=Inf)
```

```
## # A tibble: 6 x 21
##   INCIDENT_KEY OCCUR_DATE OCCUR_TIME BORO     LOC_OF_OCCUR_DESC PRECINCT
##          <dbl> <chr>      <time>     <chr>    <chr>                <dbl>
## 1    228798151 05/27/2021 21:30      QUEENS   <NA>                   105
## 2    137471050 06/27/2014 17:40      BRONX    <NA>                    40
## 3    147998800 11/21/2015 03:56      QUEENS   <NA>                   108
## 4    146837977 10/09/2015 18:30      BRONX    <NA>                    44
## 5     58921844 02/19/2009 22:58      BRONX    <NA>                    47
## 6    219559682 10/21/2020 21:36      BROOKLYN <NA>                    81
##   JURISDICTION_CODE LOC_CLASSFCTN_DESC LOCATION_DESC STATISTICAL_MURDER_FLAG
##               <dbl> <chr>              <chr>         <lgl>
## 1                 0 <NA>               <NA>          FALSE
## 2                 0 <NA>               <NA>          FALSE
## 3                 0 <NA>               <NA>          TRUE
## 4                 0 <NA>               <NA>          FALSE
## 5                 0 <NA>               <NA>          TRUE
## 6                 0 <NA>               <NA>          TRUE
##   PERP_AGE_GROUP PERP_SEX PERP_RACE VIC_AGE_GROUP VIC_SEX VIC_RACE
##   <chr>          <chr>    <chr>     <chr>         <chr>   <chr>
```

```
## 1 <NA>           <NA>     <NA>      18-24         M       BLACK
## 2 <NA>           <NA>     <NA>      18-24         M       BLACK
## 3 <NA>           <NA>     <NA>      25-44         M       WHITE
## 4 <NA>           <NA>     <NA>      <18           M       WHITE HISPANIC
## 5 25-44          M        BLACK     45-64         M       BLACK
## 6 <NA>           <NA>     <NA>      25-44         M       BLACK
##   X_COORD_CD Y_COORD_CD Latitude Longitude
##        <dbl>      <dbl>    <dbl>     <dbl>
## 1   1058925     180924      40.7     -73.7
## 2   1005028     234516      40.8     -73.9
## 3   1007668.    209837.     40.7     -73.9
## 4   1006537.    244511.     40.8     -73.9
## 5   1024922.    262189.     40.9     -73.9
## 6   1004234.    186462.     40.7     -73.9
##   Lon_Lat
##   <chr>
## 1 POINT (-73.73083868899994 40.662964620000025)
## 2 POINT (-73.92494232599995 40.81035186300006)
## 3 POINT (-73.91549174199997 40.74260663300004)
## 4 POINT (-73.91945661499994 40.83778200300003)
## 5 POINT (-73.85290950899997 40.88623791800006)
## 6 POINT (-73.92795224099996 40.678456718000064)
```

Of the first five rows of the data the columns `LOC_OF_OCCUR_DESC`, `LOC_CLASSFCTN_DESC`, `LOCATION_DESC` all appear to be missing. These columns should be checked for number of total missing values. The `PERP_AGE_GROUP`, `PERP_SEX` and `PERP_RACE` columns are missing four of five initial values and could be due to an unsolved incident.

View Summary of Data

```r
#View descriptive statistics for all columns
summary(df) %>% print()
```

```
##   INCIDENT_KEY        OCCUR_DATE         OCCUR_TIME           BORO
##  Min.   :  9953245   Length:27312       Length:27312       Length:27312
##  1st Qu.: 63860880   Class :character   Class1:hms         Class :character
##  Median : 90372218   Mode  :character   Class2:difftime    Mode  :character
##  Mean   :120860536                      Mode  :numeric
##  3rd Qu.:188810230
##  Max.   :261190187
##
##  LOC_OF_OCCUR_DESC     PRECINCT      JURISDICTION_CODE LOC_CLASSFCTN_DESC
##  Length:27312       Min.   :  1.00   Min.   :0.0000    Length:27312
##  Class :character   1st Qu.: 44.00   1st Qu.:0.0000    Class :character
##  Mode  :character   Median : 68.00   Median :0.0000    Mode  :character
##                     Mean   : 65.64   Mean   :0.3269
##                     3rd Qu.: 81.00   3rd Qu.:0.0000
##                     Max.   :123.00   Max.   :2.0000
##                                      NA's   :2
##  LOCATION_DESC      STATISTICAL_MURDER_FLAG PERP_AGE_GROUP
##  Length:27312       Mode :logical           Length:27312
##  Class :character   FALSE:22046             Class :character
```

```
##  Mode  :character   TRUE :5266              Mode  :character
##
##
##
##
##    PERP_SEX          PERP_RACE         VIC_AGE_GROUP        VIC_SEX
##  Length:27312       Length:27312       Length:27312       Length:27312
##  Class :character   Class :character   Class :character   Class :character
##  Mode  :character   Mode  :character   Mode  :character   Mode  :character
##
##
##
##
##    VIC_RACE           X_COORD_CD        Y_COORD_CD        Latitude
##  Length:27312       Min.   : 914928   Min.   :125757   Min.   :40.51
##  Class :character   1st Qu.:1000029   1st Qu.:182834   1st Qu.:40.67
##  Mode  :character   Median :1007731   Median :194487   Median :40.70
##                     Mean   :1009449   Mean   :208127   Mean   :40.74
##                     3rd Qu.:1016838   3rd Qu.:239518   3rd Qu.:40.82
##                     Max.   :1066815   Max.   :271128   Max.   :40.91
##                                                        NA's   :10
##    Longitude        Lon_Lat
##  Min.   :-74.25   Length:27312
##  1st Qu.:-73.94   Class :character
##  Median :-73.92   Mode  :character
##  Mean   :-73.91
##  3rd Qu.:-73.88
##  Max.   :-73.70
##  NA's   :10
```

As the ranges for longitude and latitude are less than half a degree of each, the `Latitude`, `Longitude` and `Lon_Lat` columns can be dropped. There is also somewhat redundant data with a much higher level of accuracy with the `X_COORD_CD` and `Y_COORD_CD` columns.

View Missing Data

```
#Save number of rows for future reference
df_size <- nrow(df)
#Print number of rows
df_size %>% print()
```

```
## [1] 27312
```

We expect to have 27,312 rows of data. With 21 columns, we expect to have 573552 total values.

```
sum(is.na(df)) %>% print()
```

```
## [1] 94165
```

We can see 94165 of the values are missing, or 16.4% of the data. We can further investigate by looking at the columns with a high number of missing values in the head of the data.

Filtering for `<NA>` values for `LOC_OF_OCCUR_DESC` gives:

```r
#Filter by only <NA> values, print the number of <NA> values in LOC_OF_OCCUR_DESC
new_df_size <- df %>% filter(is.na(LOC_OF_OCCUR_DESC)) %>% nrow()
new_df_size %>% print()
```

```
## [1] 25596
```

```r
#Print number of <NA> values as percentage
print(new_df_size / df_size)
```

```
## [1] 0.9371705
```

This means 93.7% of the values in `LOC_OF_OCCUR_DESC` are missing and the column can be safely dropped as there is very little information contained.

Repeating this process for the other columns with missing values:

`LOC_CLASSFCTN_DESC`:

```
## [1] 25596
```

```
## [1] 0.9371705
```

It appears this column has the same missing values as `LOC_OF_OCCUR_DESC`

`LOCATION_DESC`:

```
## [1] 14977
```

```
## [1] 0.548367
```

The `LOCATION_DESC` column is still mostly missing values at over 50%, but can be included in the analysis.

`PERP_AGE_GROUP`:

```
## [1] 9344
```

```
## [1] 0.3421207
```

`PERP_SEX`:

```
## [1] 9310
```

```
## [1] 0.3408758
```

`PERP_RACE`:

```
## [1] 9310
```

```
## [1] 0.3408758
```

Around $\frac{1}{3}$ of perpetrator data is missing, although more age group data is missing than sex and race data.

Therefore, after this analysis, the `Latitude`, `Longitude`, `Lon_Lat`, `LOC_OF_OCCUR_DESC`, `LOC_CLASSFCTN_DESC` columns can be excluded.

```
#Remove selected columns, print head with all columns
filtered_df <- df %>% select(-c(Latitude, Longitude, Lon_Lat, LOC_OF_OCCUR_DESC, LOC_CLASSFCTN_DESC))
head(filtered_df) %>% print(width=Inf)
```

```
## # A tibble: 6 x 16
##   INCIDENT_KEY OCCUR_DATE OCCUR_TIME BORO     PRECINCT JURISDICTION_CODE
##          <dbl> <chr>      <time>     <chr>       <dbl>              <dbl>
## 1    228798151 05/27/2021 21:30      QUEENS        105                  0
## 2    137471050 06/27/2014 17:40      BRONX          40                  0
## 3    147998800 11/21/2015 03:56      QUEENS        108                  0
## 4    146837977 10/09/2015 18:30      BRONX          44                  0
## 5     58921844 02/19/2009 22:58      BRONX          47                  0
## 6    219559682 10/21/2020 21:36      BROOKLYN       81                  0
##   LOCATION_DESC STATISTICAL_MURDER_FLAG PERP_AGE_GROUP PERP_SEX PERP_RACE
##   <chr>         <lgl>                   <chr>          <chr>    <chr>
## 1 <NA>          FALSE                   <NA>           <NA>     <NA>
## 2 <NA>          FALSE                   <NA>           <NA>     <NA>
## 3 <NA>          TRUE                    <NA>           <NA>     <NA>
## 4 <NA>          FALSE                   <NA>           <NA>     <NA>
## 5 <NA>          TRUE                    25-44          M        BLACK
## 6 <NA>          TRUE                    <NA>           <NA>     <NA>
##   VIC_AGE_GROUP VIC_SEX VIC_RACE       X_COORD_CD Y_COORD_CD
##   <chr>         <chr>   <chr>               <dbl>      <dbl>
## 1 18-24         M       BLACK             1058925     180924
## 2 18-24         M       BLACK             1005028     234516
## 3 25-44         M       WHITE             1007668.    209837.
## 4 <18           M       WHITE HISPANIC    1006537.    244511.
## 5 45-64         M       BLACK             1024922.    262189.
## 6 25-44         M       BLACK             1004234.    186462.
```

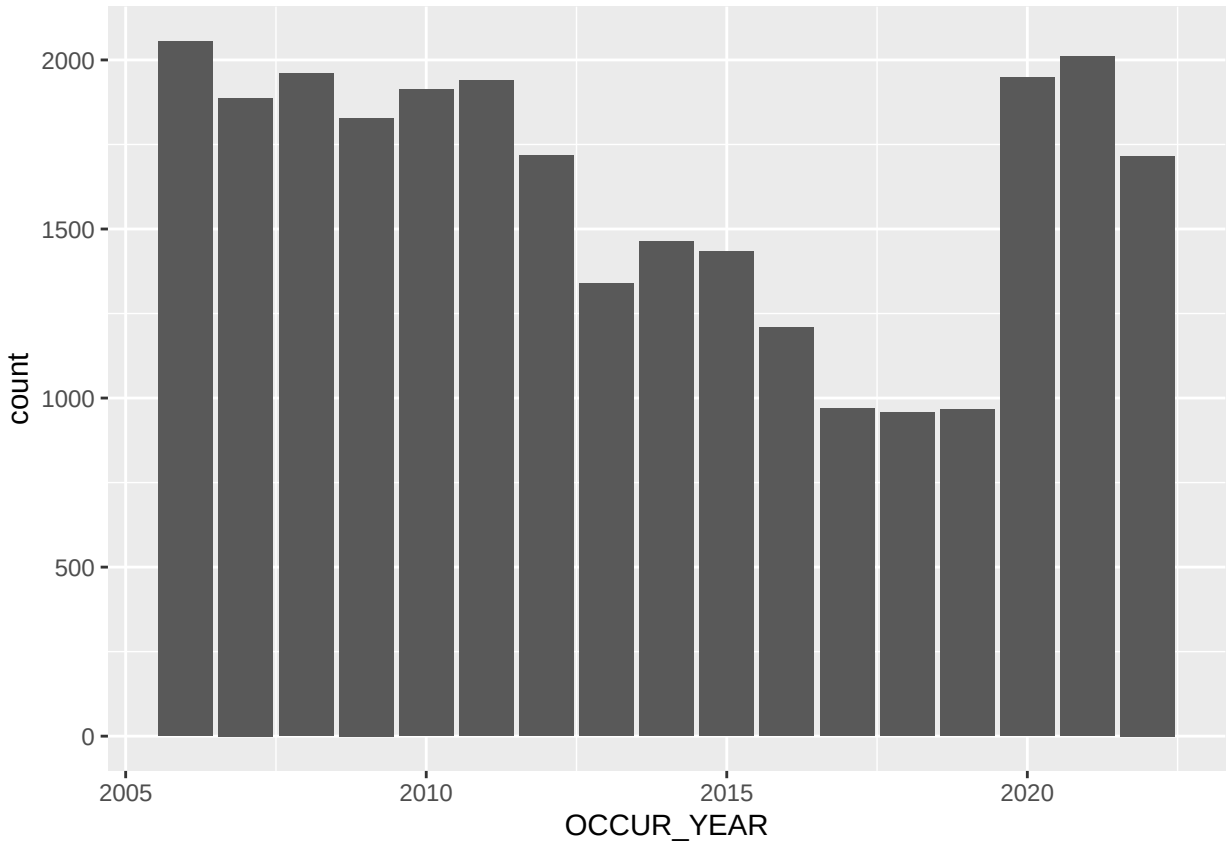
Analysis

Date Time of Incident Analysis

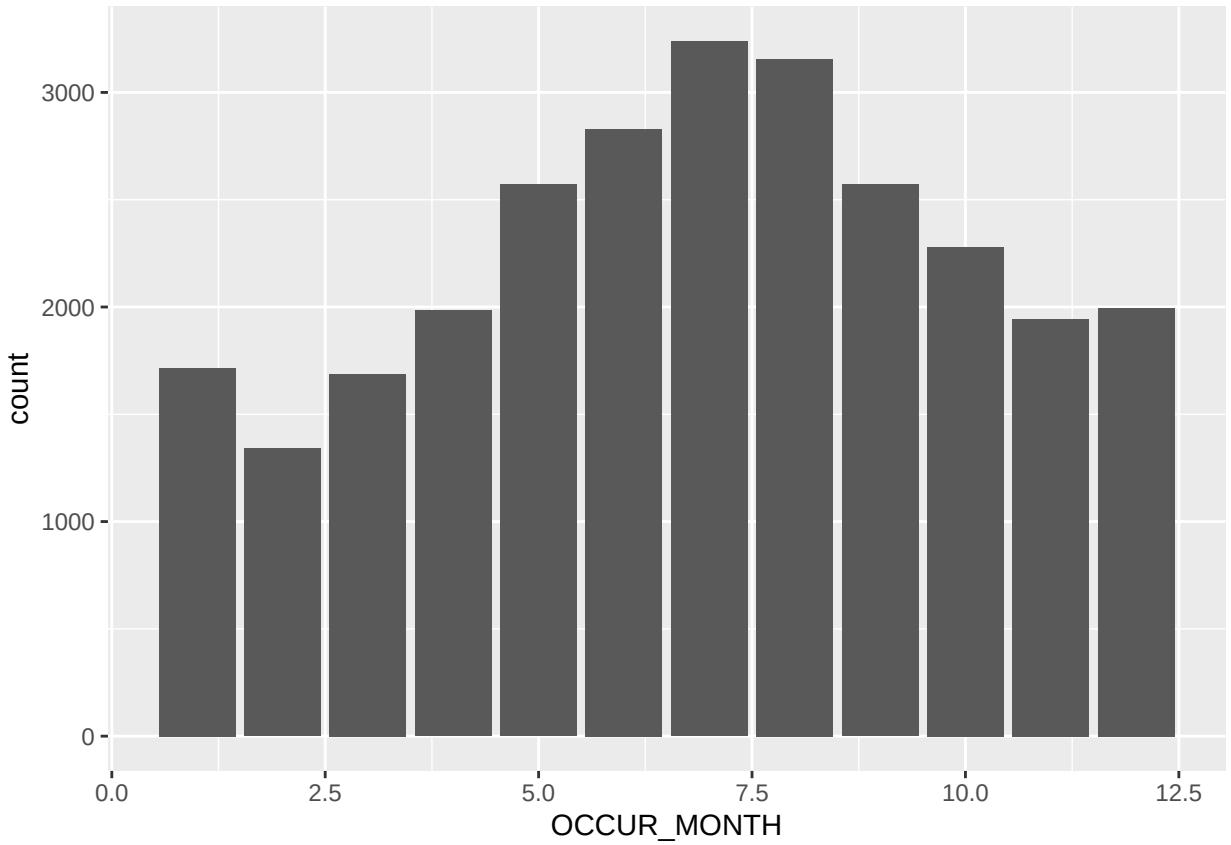
Plot the dates of shooting incidents. However, first the `OCCUR_DATE` must be converted to datetime format.

```
filtered_df <- filtered_df %>% mutate(OCCUR_DATE=mdy(OCCUR_DATE))
filtered_df <- filtered_df %>% mutate(OCCUR_YEAR=year(OCCUR_DATE), OCCUR_MONTH=month(OCCUR_DATE))
```

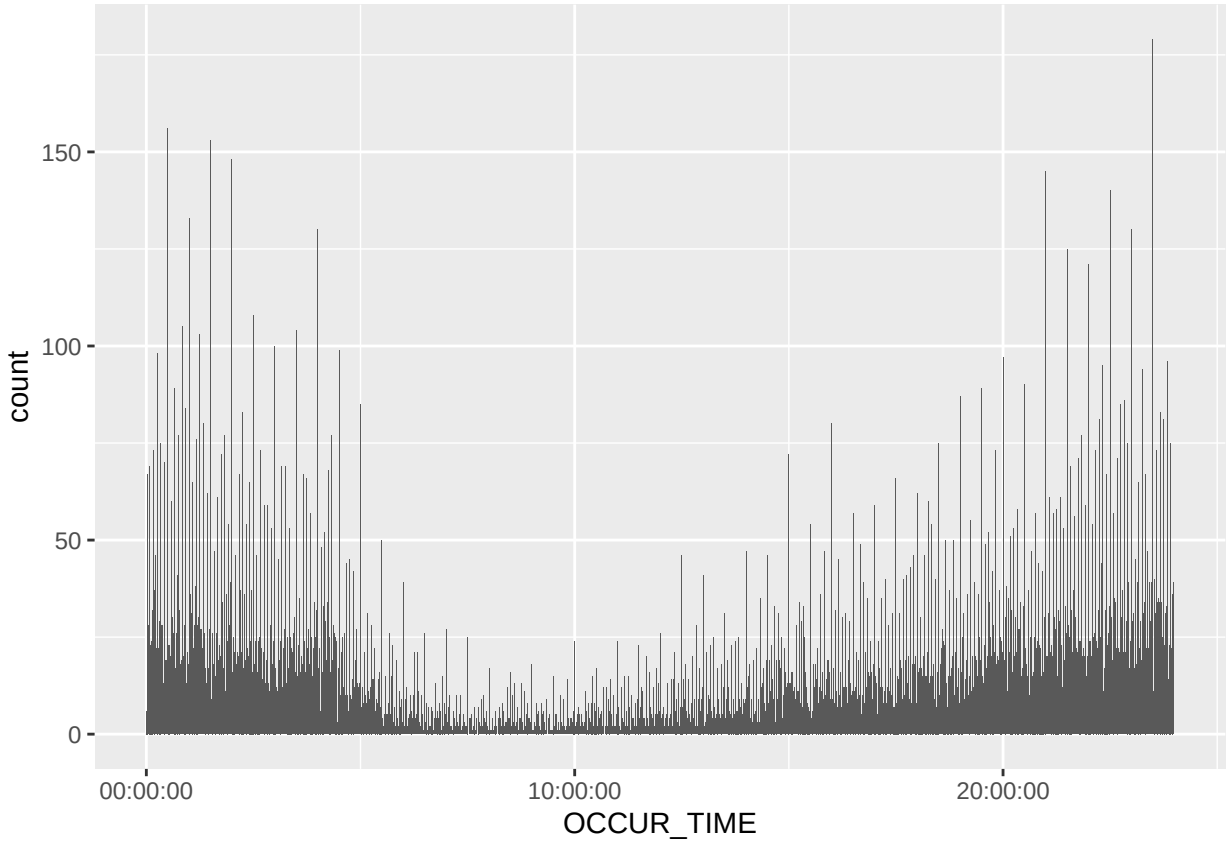
Now the real plotting can begin.



From the plot, it appears as if the number of incidents steadily decreases from 2006 through 2016. For a period between 2016 and 2019 it appears as if the number of incidents remains mostly constant around 1000 cases per year. However, in 2020 the number of incidents increases to 2006 levels at nearly 2000 incidents and has remained around that number since.



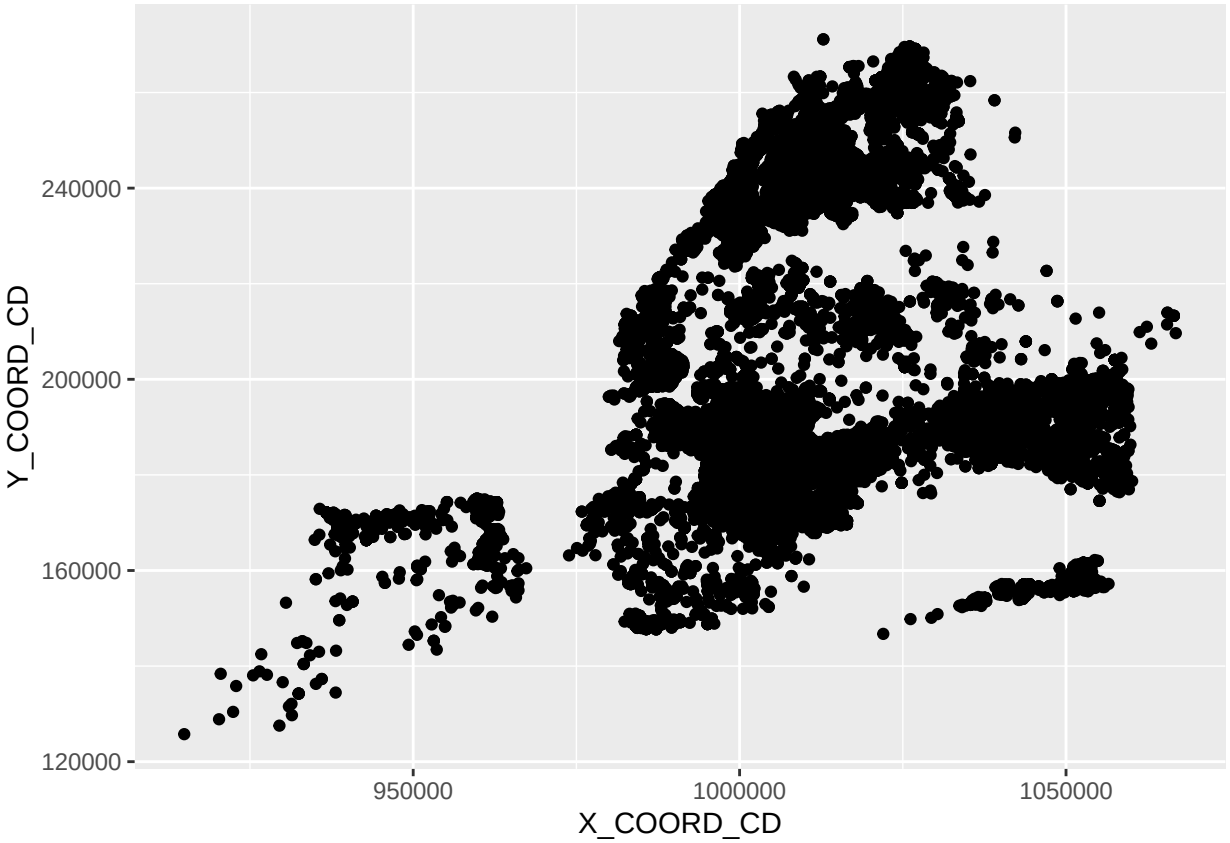
There appears to be the summer months including late spring and early fall have the highest number of incidents, with the fewest number of incidents happening in the month of February. The month with the highest number of incidents is July.



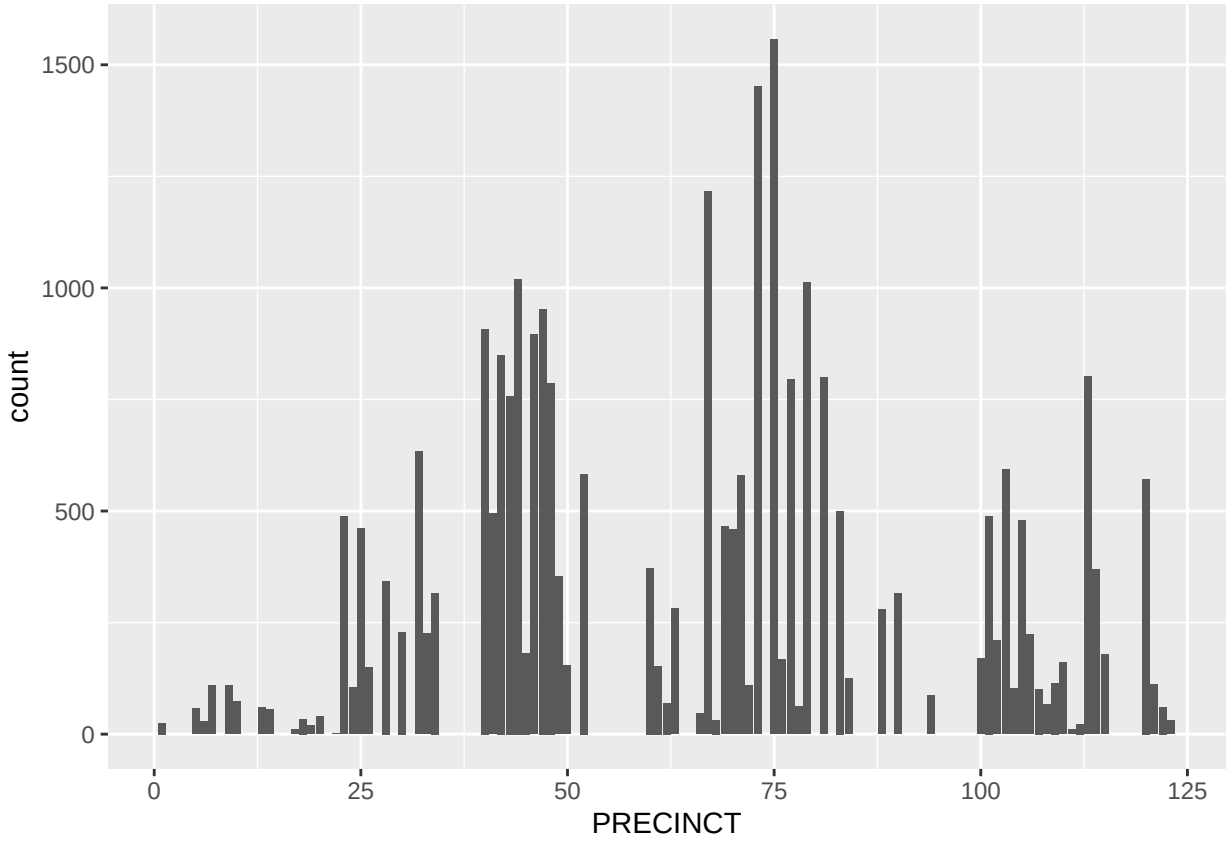
From this plot, it can be determined that the majority of incidents occur during nighttime, between midnight and 5am and between 5pm and midnight.

Location of Incident Analysis

Plot the location of shooting incidents using `X_COORD_CD` and `Y_COORD_CD`



It seems these incidents largely occur in a central area with smaller concentrations.



It appears as if some precincts between 60 and 90 have the highest frequency of incidents.

Use aggregation to find the top 10 precincts with the most incidents.

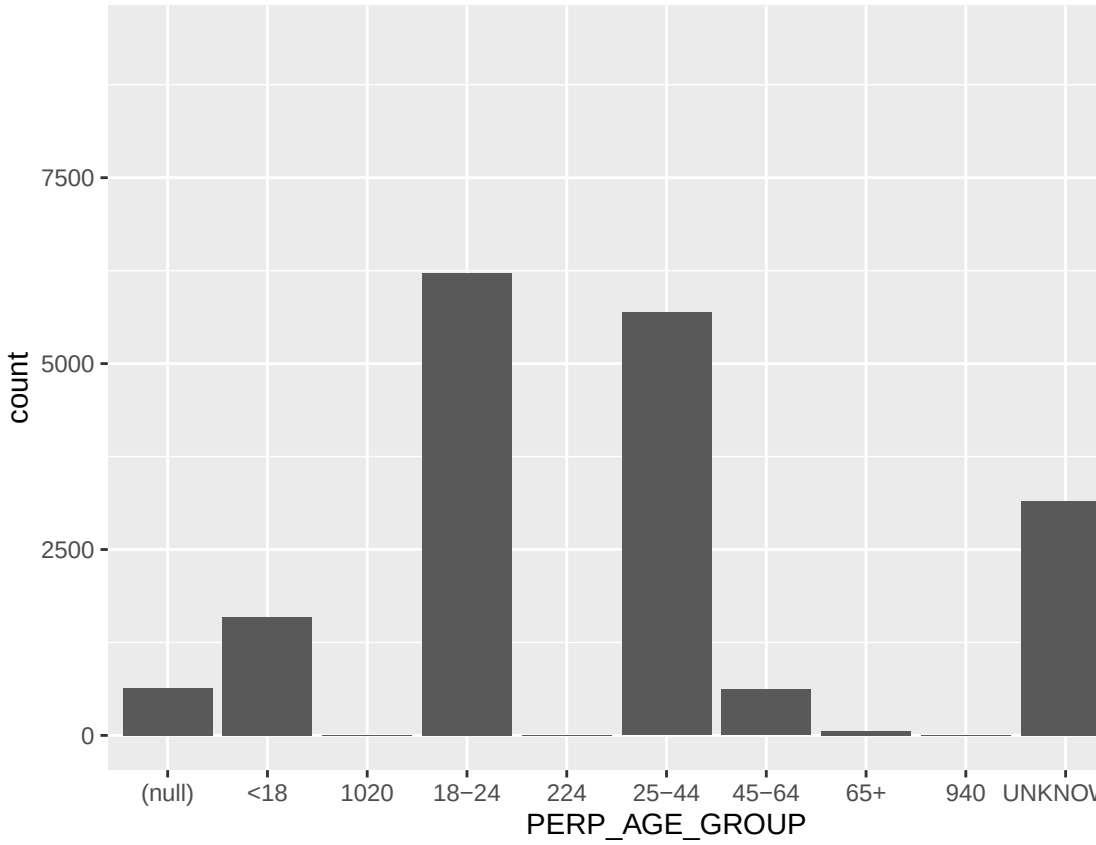
```
filtered_df %>%
    #Group by precint
    group_by(PRECINCT) %>%
    #Get number of incidents per precinct
    summarize(Num_Incidents=n()) %>%
    #Sort by number of incidents descending
    arrange(desc(Num_Incidents)) %>%
    #As dataframe, top 10 rows
    as.data.frame() %>%
    head(10)
```

```
##    PRECINCT Num_Incidents
## 1        75          1557
## 2        73          1452
## 3        67          1216
## 4        44          1020
## 5        79          1012
## 6        47           953
## 7        40           908
## 8        46           895
## 9        42           850
## 10      113           802
```

The 75th, 73rd and 67th precincts have the most incidents. Further analysis should be done by finding the locations of these precincts and comparing it to an overlayed map with the coordinates from the scatter plot above.

Victim/Perpetrator Age Group Frequency Analysis

Plot the frequency of perpetrator sex, race and age group and victim sex,race and age group to determine

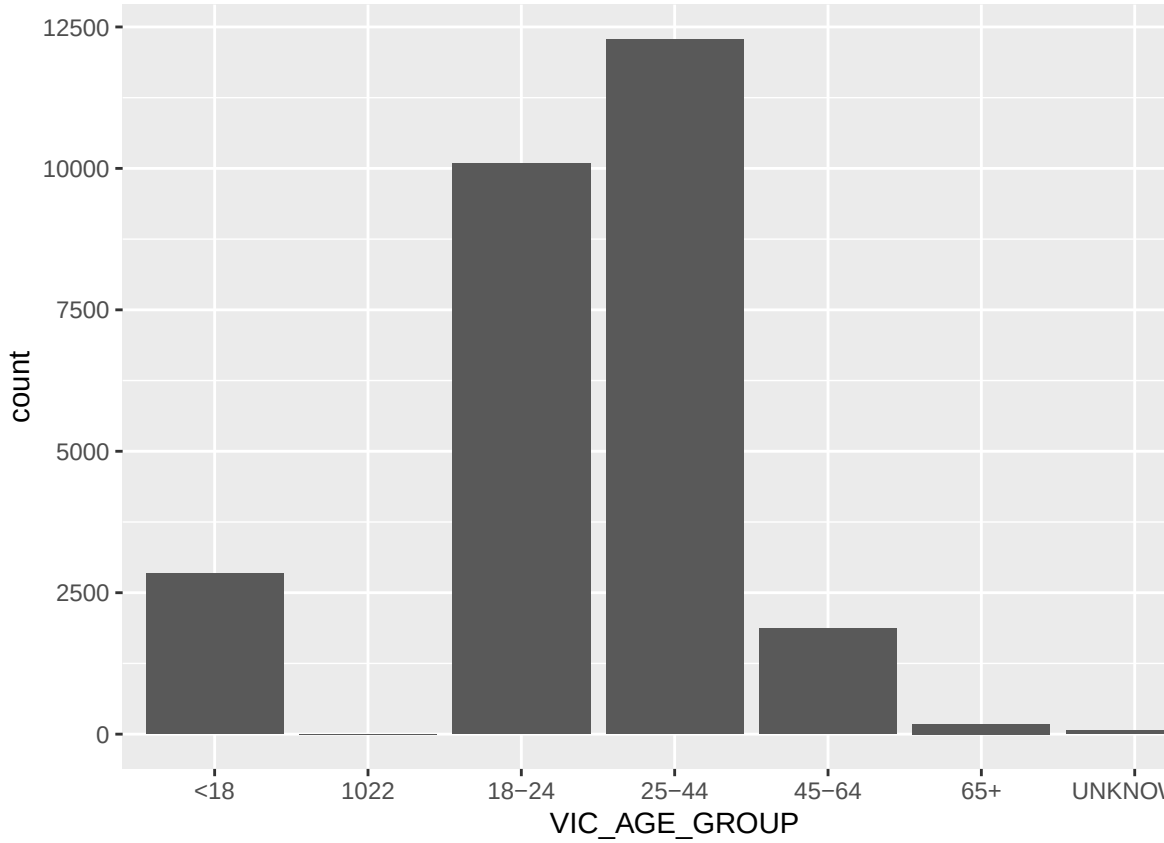


Perpetrator Age Group

A large number of ages are `UNKNOWN`, `NA` or `(null)`. However, excluding these missing or unknown values, the most common age groups for perpetrators are $18-24$ and $25-44$. This makes sense as this range makes up the majority of the population.

The third largest group of perpetrators are juveniles, followed by the $45-64$ age group.

There also appears to be incorrectly input data with values of 1020, 224 and 940.

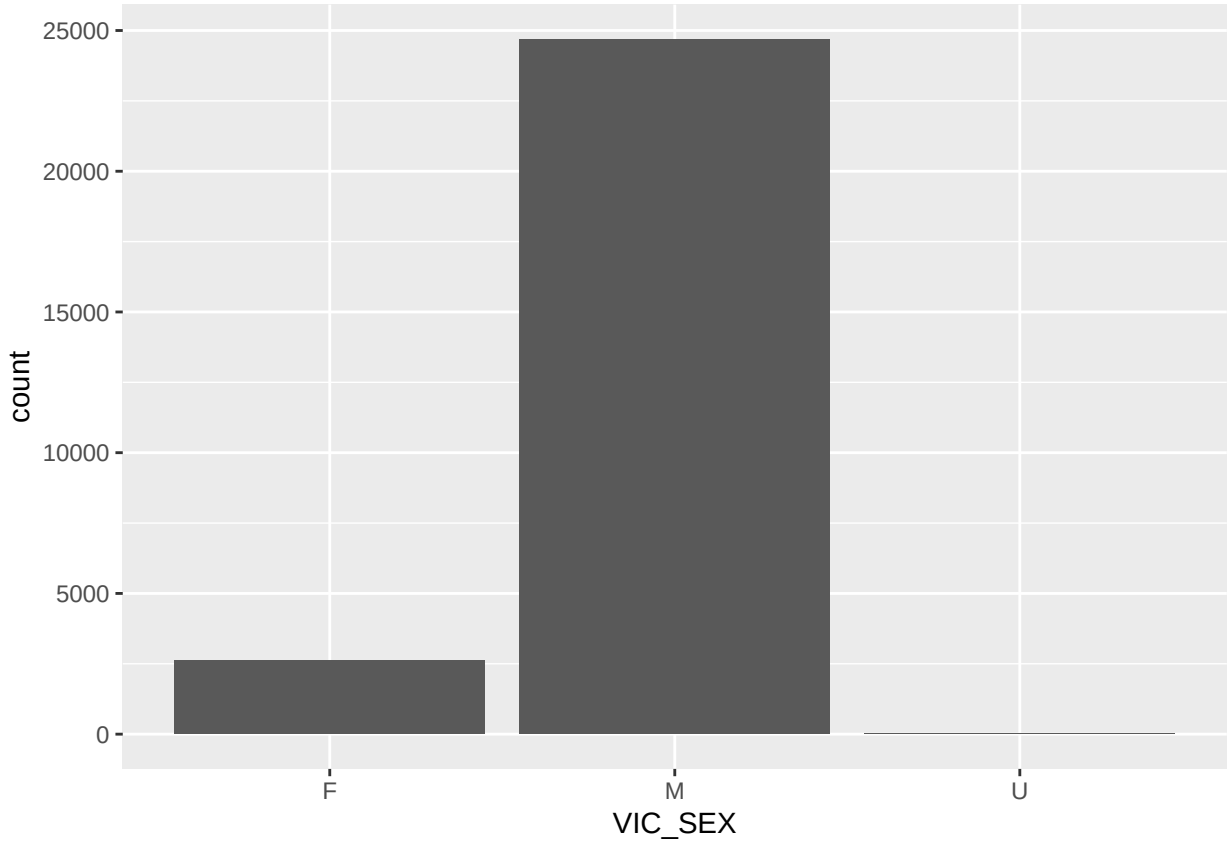


Victim Age Group

Although the distribution of ages for victims is similar to the distribution for perpetrators, there are more victims in the $25-44$ age group than the $18-24$ age group whereas the reverse is true for perpetrators. This means the average age of the victim is higher than the average age of the perpetrator of these incidents.

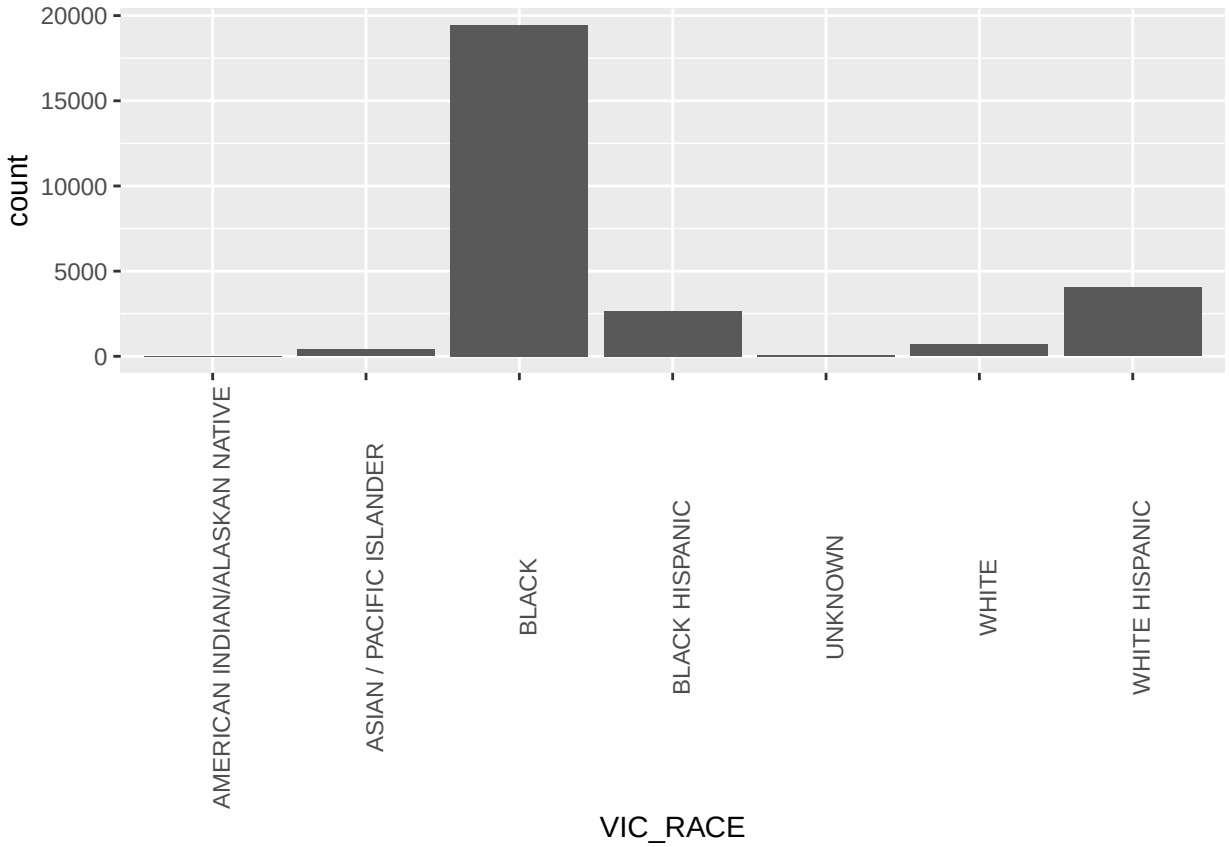
There is also only one incorrectly input label of 1022, and only one version of missing data of `UNKNOWN` compared to three versions for perpetrators.

Further investigation should be done to compare age data from the incident dataset to the true age census data for the city of New York to see if the distribution of perpetrators and victims reflects the true age distribution of the population.



Victim Sex

There is an extreme class imbalance between male victims and female victims in an approximate $10:1$ ratio. There also very few `U` cases for unknown.



Victim race

There is also an extreme class imbalance in the race of the victim.

There are very few cases for `UNKNOWN` race which means those examples can be safely removed.

Model

We will build a model using xgboost to attempt to determine perpetrator age group from the complete demographics of the victim.

First, the categorical data in the age groups must be replaced with numerical data. We can do this by first cleaning up the three versions of unknown, as well as mislabeled data that is not in a valid age group with `NA`. For this we will use `naniar`.

Also in this step, we can drop the `NA` values to further clean the data. This will leave us with only examples where

```
model_df <- filtered_df %>%
    #Select target category and predictor categories (All demographics)
    select(c(PERP_AGE_GROUP,VIC_AGE_GROUP,VIC_SEX,VIC_RACE)) %>%
    #Replace all versions of incorrect entries, unknown values and missing values with <NA>
    replace_with_na(replace=list(
        PERP_AGE_GROUP=c("1020","224","940","UNKNOWN","(null)"),
        VIC_AGE_GROUP=c("1022","UNKNOWN"),
        VIC_SEX=c("U"),
        VIC_RACE=c("UNKNOWN")
    )) %>%
```

```
    #Remove rows with <NA> values
    filter(!is.na(VIC_AGE_GROUP),!is.na(VIC_SEX),!is.na(VIC_RACE),!is.na(PERP_AGE_GROUP))
#Print column names, unique values to ensure replacement successful
names(model_df) %>% print()
```

```
## [1] "PERP_AGE_GROUP" "VIC_AGE_GROUP"  "VIC_SEX"        "VIC_RACE"
```

```
unique(model_df$PERP_AGE_GROUP) %>% print()
```

```
## [1] "25-44" "18-24" "45-64" "<18"   "65+"
```

```
unique(model_df$VIC_AGE_GROUP) %>% print()
```

```
## [1] "45-64" "25-44" "18-24" "<18"   "65+"
```

```
unique(model_df$VIC_RACE) %>% print()
```

```
## [1] "BLACK"                          "WHITE"
## [3] "BLACK HISPANIC"                 "WHITE HISPANIC"
## [5] "ASIAN / PACIFIC ISLANDER"       "AMERICAN INDIAN/ALASKAN NATIVE"
```

```
#Print number of rows to view sample data size
nrow(model_df) %>% print()
```

```
## [1] 14093
```

```
#Ensure no <NA> values remain after replacement
model_df %>% summarise(NA_per_row = sum(is.na(.)))
```

```
## # A tibble: 1 x 1
##   NA_per_row
##        <int>
## 1          0
```

This leaves us with a dataset of 14093 (reduced from the original 27312) with four columns or features (`PERP_AGE_GROUP`, `VIC_AGE_GROUP`, `VIC_SEX`, `VIC_RACE`). This can further be split 70/30 into a training and test set.

However, because all the values are currently in text format and are categorical variables, these must be converted to numerical. To do this, each column will be converted to dummies.

First convert the target feature `PERP_AGE_GROUP` to numerical with ordinal encoding:

```
#Function to ordinally encode the age group feature row-wise
ordinal_encode <- function(x) {
    if (x == "<18") {
        result <- "Zero"
    } else if (x == "18-24") {
        result <- "One"
    } else if (x == "25-44") {
```

```
        result <- "Two"
    } else if (x == "45-64") {
        result <- "Three"
    } else if (x == "65+") {
        result <- "Four"
    }
    return(first(result))
}

#Apply ordinal encoding across target feature perp age group then convert to factor
model_df <- model_df %>% mutate(PERP_AGE_GROUP=pmap(across(PERP_AGE_GROUP),
    ~ ordinal_encode(..1)))
model_df$PERP_AGE_GROUP <- sapply(model_df$PERP_AGE_GROUP,first)
model_df$PERP_AGE_GROUP <- factor(model_df$PERP_AGE_GROUP)
#Ensure ordinal encoding complete
unique(model_df$PERP_AGE_GROUP) %>% print()
```

```
## [1] Two   One   Three Zero  Four
## Levels: Four One Three Two Zero
```

Next convert to dummies the remaining features using the `fastDummies` library:

```
#Use dummy encoding to convert categorical features to binary features usable by the model
model_df <- model_df %>% dummy_cols(
    select_columns=c("VIC_AGE_GROUP","VIC_SEX","VIC_RACE"),remove_selected_columns = TRUE)
names(model_df) %>% print()
```

```
##  [1] "PERP_AGE_GROUP"
##  [2] "VIC_AGE_GROUP_<18"
##  [3] "VIC_AGE_GROUP_18-24"
##  [4] "VIC_AGE_GROUP_25-44"
##  [5] "VIC_AGE_GROUP_45-64"
##  [6] "VIC_AGE_GROUP_65+"
##  [7] "VIC_SEX_F"
##  [8] "VIC_SEX_M"
##  [9] "VIC_RACE_AMERICAN INDIAN/ALASKAN NATIVE"
## [10] "VIC_RACE_ASIAN / PACIFIC ISLANDER"
## [11] "VIC_RACE_BLACK"
## [12] "VIC_RACE_BLACK HISPANIC"
## [13] "VIC_RACE_WHITE"
## [14] "VIC_RACE_WHITE HISPANIC"
```

Finally, we can split this dataset into a training and test dataset with a 70/30 split. Then X and y will be split into a X,y set where X is `PERP_AGE_GROUP` and y is the dummies for `VIC_AGE_GROUP`, `VIC_SEX`, `VIC_RACE`. This will form four sets: `X_test`, `X_train`, `y_test` and `y_train`.

```
#Fix seed to ensure reproducability
set.seed(71)
#Split the data into 70% training data 30% test data
sample <- createDataPartition(model_df$PERP_AGE_GROUP, p=0.7, list=FALSE)

train <- model_df[sample,]
```

```
test <- model_df[-sample,]
#Further split into X and y for training features and target features
y_train <- train %>% select(c(PERP_AGE_GROUP)) %>%
    pull(PERP_AGE_GROUP)
y_test <- test %>% select(c(PERP_AGE_GROUP)) %>%
    pull(PERP_AGE_GROUP)
X_train <- train %>% select(-c(PERP_AGE_GROUP))
X_test <- test %>% select(-c(PERP_AGE_GROUP))
#Save and print the size of the training and test sets to ensure proper split
train_size <- nrow(train)
test_size <- nrow(test)

cat("Train Size: ", train_size , " Test Size: ", test_size)
```

```
## Train Size:  9866  Test Size:  4227
```

The next step is to perform grid search as hyperparameter tuning to find the optimal values of `max_depth` and `nrounds` in our model. We will use 5-fold cross validation, minimizing logloss as our performance metric.

```
#Use caret train to perform 5-fold cross validation grid search on max_depth and nrounds
train_control <- trainControl(method="cv", number=5, search="grid", classProbs=TRUE,
                              summaryFunction=multiClassSummary, savePredictions="all")
tune_grid <- expand.grid(max_depth = c(3,5,7), nrounds = (1:10)*25, eta=0.3, gamma=0,
                         colsample_bytree=0.6, min_child_weight=1, subsample=1)
model <- train(x=X_train, y=y_train, method="xgbTree", metric="logLoss", maximize=FALSE,
                      trControl=train_control, tuneGrid=tune_grid, verbosity=0)
```

```
## Warning: Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
## Setting row names on a tibble is deprecated.
```

```
## Warning in nominalTrainWorkflow(x = x, y = y, wts = weights, info = trainInfo,
## : There were missing values in resampled performance measures.
```

```
## Warning: Setting row names on a tibble is deprecated.
```

```
#Look at the tuned hyperparameters and results w/ loss and acc
model$results %>% select(max_depth, nrounds, logLoss, Accuracy)#Mean_Sensitivity, Mean_Specificity, Mea
```

```
##    max_depth nrounds  logLoss  Accuracy
## 1          3      25 1.067029 0.5185509
## 11         5      25 1.069056 0.5179428
## 21         7      25 1.069087 0.5181453
## 2          3      50 1.066811 0.5181454
## 12         5      50 1.069791 0.5182467
## 22         7      50 1.070075 0.5180441
## 3          3      75 1.068297 0.5183483
## 13         5      75 1.071070 0.5173347
## 23         7      75 1.071370 0.5173346
## 4          3     100 1.069288 0.5183483
## 14         5     100 1.071856 0.5176387
## 24         7     100 1.072016 0.5173346
## 5          3     125 1.069921 0.5182467
## 15         5     125 1.072342 0.5173346
## 25         7     125 1.072544 0.5180442
## 6          3     150 1.070492 0.5182467
## 16         5     150 1.072792 0.5179428
## 26         7     150 1.073002 0.5169291
## 7          3     175 1.070917 0.5181454
## 17         5     175 1.073077 0.5168278
## 27         7     175 1.073381 0.5171318
## 8          3     200 1.071193 0.5179427
## 18         5     200 1.073421 0.5182468
## 28         7     200 1.073717 0.5180442
## 9          3     225 1.071492 0.5178413
## 19         5     225 1.073731 0.5174359
## 29         7     225 1.074113 0.5173346
## 10         3     250 1.071783 0.5179426
## 20         5     250 1.074003 0.5173346
## 30         7     250 1.074407 0.5175373
```

Now the model can be created using a softmax objective for multiclass classification, using the optimal hyperparameters.

```
#Because the train method provides the best model trained, we will just display the hyperparameters
model$finalModel$tuneValue %>% print()
```

```
##   nrounds max_depth eta gamma colsample_bytree min_child_weight subsample
## 2      50         3 0.3     0              0.6                1         1
```

Make predictions using the model and compute the confusion matrix:

```
#Make predictions using the model
y_pred <- predict(model, X_test)
#Decode from ordinal
decode <- function(x) {
    if (x == "Zero") {
        result <- 0
    } else if (x == "One") {
        result <- 1
    } else if (x == "Two") {
        result <- 2
```

```
    } else if (x == "Three") {
        result <- 3
    } else if (x == "Four") {
        result <- 4
    }
    return(result)
}
y_pred <- sapply(y_pred, decode)
y_test <- sapply(y_test, decode)
#Create confusion matrix from the predictions compared to known test values
confusion_matrix <- table(y_pred, y_test) %>% melt()
```

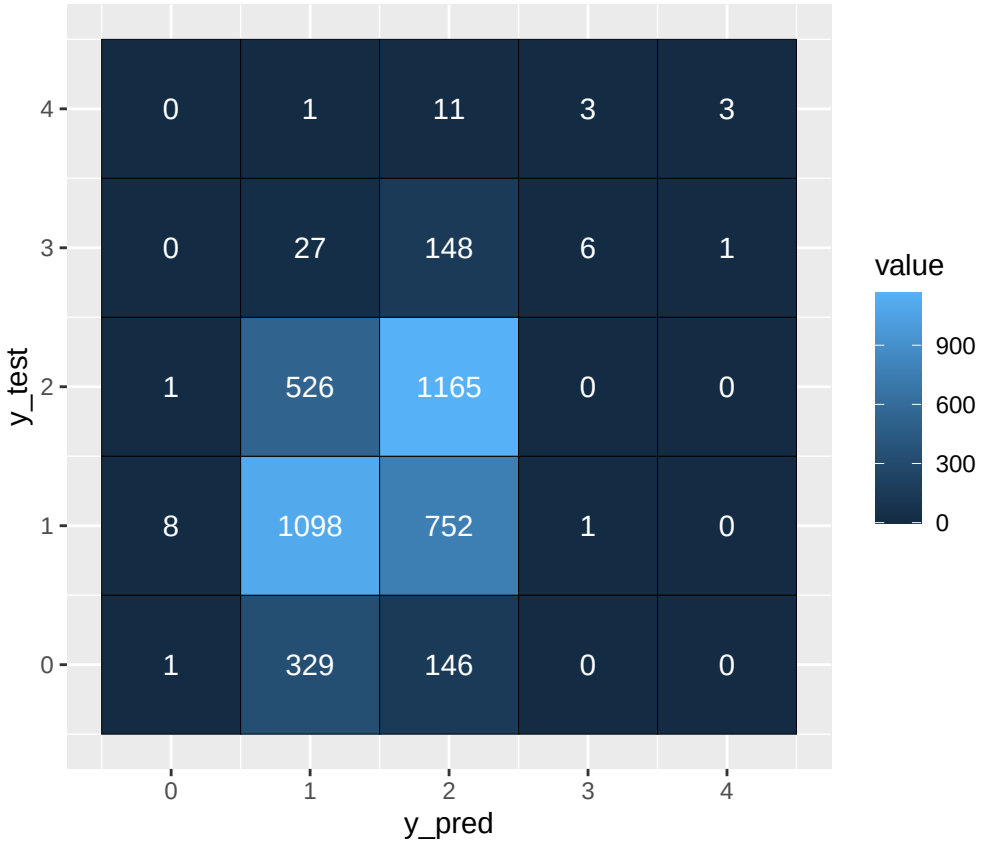
```
## Warning in type.convert.default(X[[i]], ...): 'as.is' should be specified by
## the caller; using TRUE

## Warning in type.convert.default(X[[i]], ...): 'as.is' should be specified by
## the caller; using TRUE
```

```
#Print the confusion matrix
confusion_matrix %>% print()
```

```
##    y_pred y_test value
## 1       0      0     1
## 2       1      0   329
## 3       2      0   146
## 4       3      0     0
## 5       4      0     0
## 6       0      1     8
## 7       1      1  1098
## 8       2      1   752
## 9       3      1     1
## 10      4      1     0
## 11      0      2     1
## 12      1      2   526
## 13      2      2  1165
## 14      3      2     0
## 15      4      2     0
## 16      0      3     0
## 17      1      3    27
## 18      2      3   148
## 19      3      3     6
## 20      4      3     1
## 21      0      4     0
## 22      1      4     1
## 23      2      4    11
## 24      3      4     3
## 25      4      4     3
```

Plot the confusion matrix:



Interestingly, the model does not seem to predict class 0, class 3, or class 4. These classes map to the age groups < 18, $45 - 64$ and $65+$ respectively. This could be due to the low amount of training examples in these classes and should be investigated further.

However, it seems to have a high degree of accuracy classifying these smaller age groups, while it struggles to classify the two large groups and often confuses them with each other.

Accuracy

Accuracy can be calculated by the number of correct predictions (the diagonal) divided by the number of total predictions.

$$= \frac{TP}{TP + TN + FP + FN} = \frac{2273}{4227}$$

Where TP is the number of true positives, TN is the number of true negatives, FP is the number of false positives and FN is the number of false negatives.

This results in an accuracy of 53.8%.

Precision

Precision is a metric that measures how many of the positive predicted samples are true positives. This is a good metric for when the cost of a false positive prediction is high.

Precision for each class can be calculated by taking the number of correct predictions divided by the sum of number of predictions for that class. In other words, the number of correct predictions for the given class divided by the sum of all other values in that row of the confusion matrix.

As an example, the precision for class 1 is calculated below.

$$= \frac{TP_1}{TP_1 + FP_1} = \frac{1098}{8 + 1098 + 752 + 1 + 0}$$

This results in a precision for class 1 of 0.59.

Recall

Recall is a metric that measures how many of the positive predicted samples are labelled positive by the model. This is a good metric for when the cost of a false negative is prediction is high.

Recall for each can be calculated by taking the number of correct predictions divided by the sum of the number of true positive predictions for that class. In other words, the number of correct predictions for the given class divided by the sum of of all other values in that column of the confusion matrix.

$$= \frac{TP_1}{TP_1 + FN_1} = \frac{1098}{1 + 27 + 526 + 1098 + 329}$$

This results in a recall for class 1 of 0.55.

Per Class Results

The results per class can be found in the table below.

Class	n True	n Predicted	Accuracy	Precision	Recall	F1
0	476	10	88.55%	0.10	0.0021	0.0041
1	1859	1981	61.11%	0.55	0.59	0.57
2	1692	2222	62.53%	0.52	0.69	0.60
3	182	10	95.74%	0.60	0.033	0.063
4	18	4	99.62%	0.75	0.17	0.27

Conclusion

In conclusion, there appears to be some correlation between victim and perpetrator demographics in historical NYPD shooting incidents, but only in the cases where the perpetrator is in between the ages of 18 and 65 as the model poorly performs on juveniles and senior citizens.

This performance could likely be increased by performing better data sampling as there is a heavily imbalanced dataset. The most simple solution would be oversampling the minority classes by resampling cases in the < 18 and 65+ age groups. Other solutions include more advanced techniques such as SMOTE or ADASYN.

Bias

One major source of bias is the imbalanced dataset of ages and demographics previously mentioned. Due to the very low number of samples of some age groups, and the overwhelming imbalance of male victims to female victims, as well as racial bias, the dataset will be biased towards the majority class. This will further lead to bias towards the majority class in the model.

As another source of bias, all of the data comes from a single source. As the NYPD is the sole source of the data and is also the organization in charge of reporting the data, there could be biases or inconsitencies with the reporting of this data.