Snow Emergency Tickets and Tows

Overview:

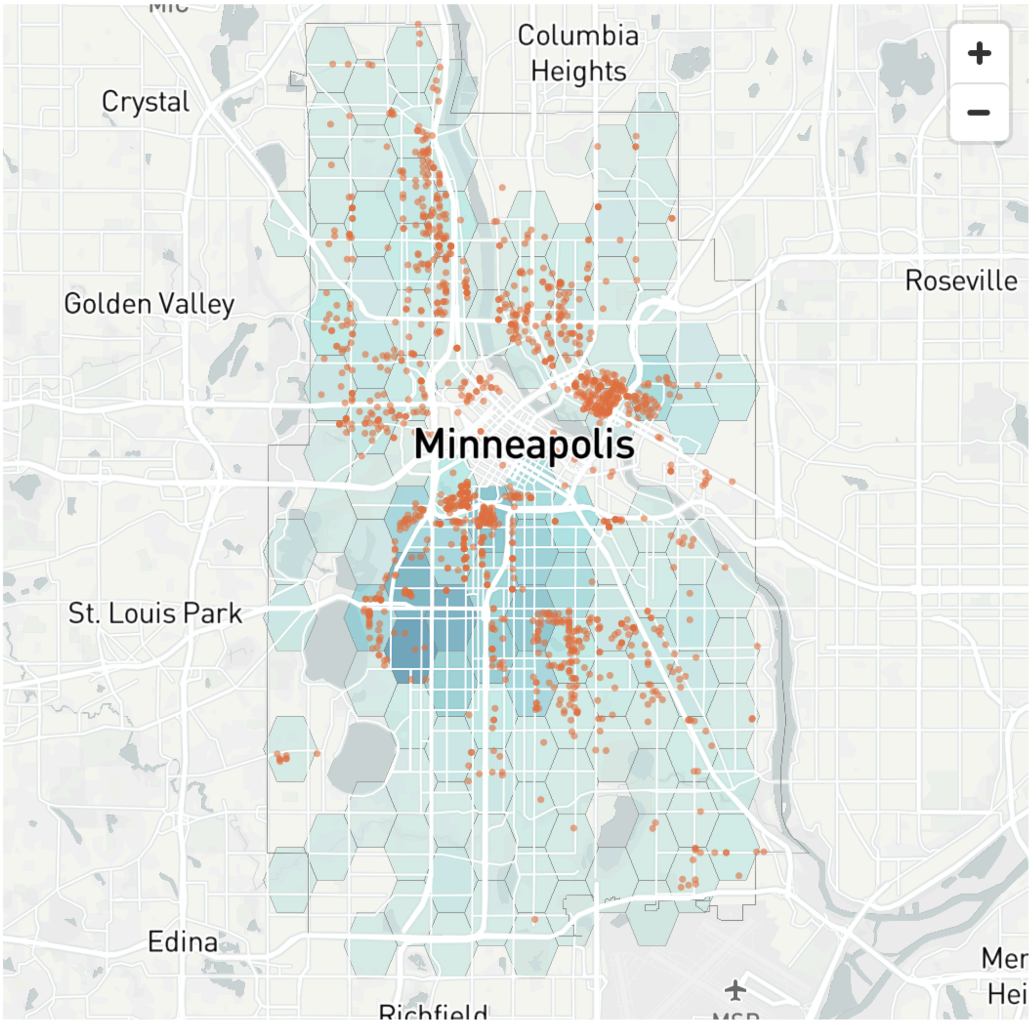
When Minneapolis declares a snow emergency, parking restrictions are in place for three days. Vehicles are towed, or tagged (ticketed) if they are in violation of the parking restrictions.

http://www.ci.minneapolis.mn.us/snow/snow_parking-info

This project examines some factors that affect whether a vehicle is more likely to be towed or tagged. Towing is more expensive and more disruptive to the vehicle owner so the financial impact is more considerable. What factors may affect how the snow emergency crews decide to tag or tow a vehicle? And are there any conclusions that the City of Minneapolis could use to target and improve communication with residents about snow emergency parking restrictions to minimize improperly parked vehicles? If vehicles park in accordance with the snow parking rules, snow can be cleared more effectively, so efforts to increase awareness would benefit both the city and residents.

The distribution of towed/tagged cars varies considerably across the city, with a higher proportion easily visible in high-density neighborhoods such as Uptown, and very few in southwest Minneapolis (as noted by this Star Tribune story from February 13 http://www.startribune.com/after-snow-emergencies-tow-trucks-are-rare-visitors-to-southwest-minneapolis/505734202/). In the accompanying graphic, the distribution of vehicles tagged is not the same as vehicles towed. Tags appear to be common in neighborhoods like Uptown, Whittier and Phillips. Tows are concentrated in areas like Loring Park, Cedar-Riverside, University areas, and neighborhoods immediately surrounding the I-94 corridor in North Minneapolis.

Shows the density of **vehicles tagged** during January and February Minneapolis snow emergencies – shaded from less to **more** – compared to **vehicles towed**.



Data source: City of Minneapolis | Graphic by Jeff Hargarten, Star Tribune | Note: tagging data is preliminary

Method

Factors considered

- Day of snow emergency
- Neighborhood
- Type of road – local, residential, commercial, highway…
- Driving distance to impound lot
- Driving distance to impound lot
- Tow zone

Factors to be considered in a future extension to this project

- Density of cars in neighborhood
- Density of people in neighborhood
- Percent renters vs. owners (as a very rough approximation of off-street parking availability, apartments tend to lack off-street parking)
- English fluency of residents (to investigate if multi-lingual communications could be improved)
- Other factors, as identified (suggestions welcome)

Data sources

Tows from one snow emergency (Westminster, February 12, 13 and 14, 2019), approximately 900 tows
http://opendata.minneapolismn.gov/datasets/snow-emergency-westminster-tows-2019?geometry=-93.394%2C44.903%2C-93.124%2C44.988

Tags (tickets) from the same snow emergency (Westminster), approximately 4000 tags
http://opendata.minneapolismn.gov/datasets/snow-emergency-westminster-tags-2019?geometry=-93.334%2C44.926%2C-93.199%2C44.969

Pavement management data set shapefile, includes type of street (local, residential, commercial, highway) …
http://opendata.minneapolismn.gov/datasets/public-works-street-pavement-mgmt?geometry=-93.668%2C44.884%2C-93.13%2C45.054

Distances from each tow or tag location to the Minneapolis Impound Lot from Open Routing Service's Directions API https://openrouteservice.org/dev/#/api-docs/directions/

Data Processing

The CSV files for tags and tows were manually joined, and column for the type of event (TAG or TOW) was added.

Nearest neighbor analysis using the NNJoin plugin for QGIS to create a distance matrix from each ticket/tag and street and join with the pavement management data set, to add a column for the street type of each tow and tag. Reference: https://plugins.qgis.org/plugins/NNJoin/

Distance from each tag or tow event to the Minneapolis Impound Lot was queried from the Open Route Service API using a Python script (I got this working with R too but ORS limits requests to 2000 per day so it couldn't request all ~5000 distances in one go and would lose everything once the first request was rejected. I'm sure R can save the work it's done so far and then pick up where it left off, but I'm running of time and I know how to do it in Python). Python and R versions here: https://git.io/fjWCM

```python
import requests, csv, time

key = 'ORS KEY GOES HERE'
file = 'data/SNOW_TAG_TOW_TYPES.csv'

def get_driving(location):  # end in the format of = '-95.4545,45.343'
    distance_url = 'https://api.openrouteservice.org/v2/directions/driving-car'
    impound_lot = '-93.291796,44.977125'
    params = { 'api_key': key , 'start': location, 'end': impound_lot }
    try:
        response = requests.get( distance_url, params = params).json()
        return response['features'][0]['properties']['summary']
    except Exception as e:
        print(e)
        return None  # Caller will use this to know if the request has been rejected.

counter = 1  # For slowing down the request rate

with open(file) as csvfile:
    reader = csv.reader(csvfile, delimiter=',')
    header = reader.__next__()
    rows = list(reader)
    try:
        for row in rows:
            distance = row[15]
            drivetime = row[16]
            if distance and drivetime:
                continue

            time.sleep(2)       # Throttle requests or get blocked after ~40 requests.
            counter += 1
            if counter == 30:
                time.sleep(10)
                counter = 0

            loc = f'{row[1]},{row[2]}'
            driving = get_driving(loc)

            if driving:
                row[15] = str(driving['distance'])
                row[16] = str(driving['duration'])
            else:
                break
    except Exception as e:
        print("error", e)
        pass

with open(file, 'w') as csvfile:  # Write all the data to the CSV file
    writer = csv.writer(csvfile, delimiter=',')
    writer.writerow(header)
    for row in rows:
        writer.writerow(row)
```

Analysis

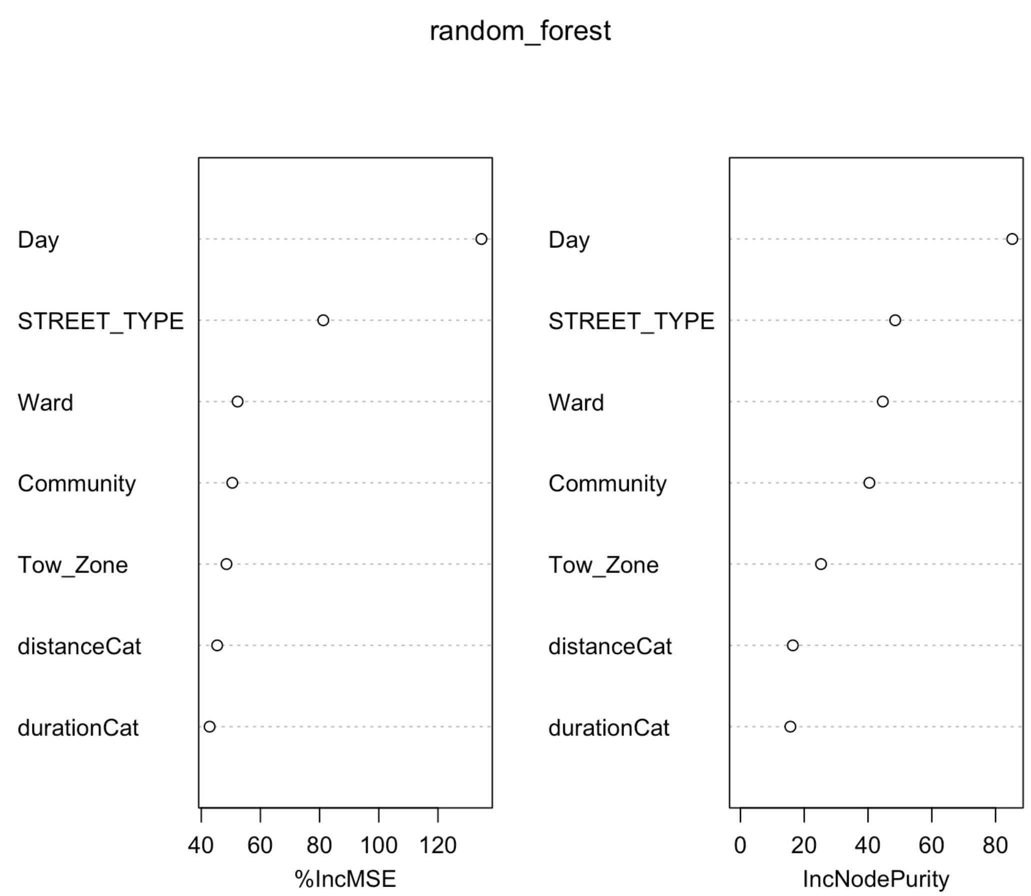
The CSV was loaded into a dataframe in R.

The distance and duration were classified into quantiles (distanceCat and durationCat) so they can be rasterized.

The start of the data set then looks like this,

Type	X	Y	OBJE	Call_Taken	Location	Latitude	Longitude	Ward	Community	Neighborho	Tow_Z	Day	Snow_Emerg	STREET_TYP	distance	duration	distanceCat	durationCat	was_tow
TOW	-93.288021	45.013111	1	02/12/2019	3201-3299 ly	45.013111	-93.288021	4	Camden	McKinley	1	1	Westminste	MSA	4450	438	2	2	1
TOW	-93.262588	44.94139	2	02/12/2019	3398 Chicag	44.94139	-93.262588	9	Powderhorn	Central	5	1	Westminste	MSA	6623.4	631.3	4	4	1
TOW	-93.262606	44.943159	3	02/12/2019	3301 Chicag	44.943159	-93.262606	9	Powderhorn	Central	5	1	Westminste	MSA	6427	615.6	4	4	1
TOW	-93.288036	45.016708	4	02/12/2019	3400-3498 ly	45.016708	-93.288036	4	Camden	McKinley	1	1	Westminste	MSA	4848.8	469.9	2	2	1
TOW	-93.288049	45.01849	5	02/12/2019	3498 Lyndale	45.01849	-93.288049	4	Camden	McKinley	1	1	Westminste	RES	5049.3	486.2	2	2	1
TOW	-93.277559	44.963797	6	02/12/2019	1 E 19TH ST,	44.963797	-93.277559	6	Central	Steven's Squ	3	1	Westminste	RES	2909.4	314.1	1	1	1
TOW	-93.288036	45.016708	7	02/12/2019	3400 Lyndale	45.016708	-93.288036	4	Camden	McKinley	1	1	Westminste	MSA	4848.8	469.9	2	2	1
TOW	-93.276939	44.964115	8	02/12/2019	16 EAST / 19	44.964115	-93.276939	6	Central	Steven's Squ	3	1	Westminste	RES	2957.9	325.7	1	1	1
TOW	-93.276939	44.964115	9	02/12/2019	16 19th st e,	44.964115	-93.276939	6	Central	Steven's Squ	3	1	Westminste	RES	2957.9	325.7	1	1	1
TOW	-93.262901	44.939686	10	02/12/2019	3458 Chicag	44.939686	-93.262901	9	Powderhorn	Central	5	1	Westminste	MSA	6852.6	649.4	4	4	1
TOW	-93.235311	44.987678	11	02/12/2019	1116 Como A	44.987678	-93.235311	2	University	Como	4	1	Westminste	MSA	5520	627.8	4	4	1
TOW	-93.279701	44.948359	12	02/12/2019	3000 Blaisde	44.948359	-93.279701	10	Powderhorn	Whittier	3	1	Westminste	CSAH	4508.9	447.2	2	2	1
TOW	-93.288113	45.029786	13	02/12/2019	4100 lyndale	45.029786	-93.288113	4	Camden	Webber - Ca	1	1	Westminste	CSAH	6800.6	539	4	3	1

Random forest was run on the dataset, for the columns **Day, STREET_TYPE, Ward, Community, Tow_Zone, distanceCat and durationCat.** The day of the snow emergency appears to be the most important factor regarding whether a vehicle is tagged or towed.



A quick look at the numbers supports this – more cars are towed than tagged on Day 1, more cars are tagged than towed on Day 2 and 3.

```
> table(dataframe[dataframe$Day == 1, ]$Type)

TAG TOW
156 305
> table(dataframe[dataframe$Day == 2, ]$Type)

 TAG  TOW
1714  393
> table(dataframe[dataframe$Day == 3, ]$Type)

 TAG  TOW
1130  282
```

STREET_TYPE was the next most important. Looking at counts of tows and tags by street type, a vehicle parked on a collector street (CSAH in the table) (https://en.wikipedia.org/wiki/Collector_road) during day 1 of a snow emergency has a high probability of a tow.   Vehicles on residential and local streets are more likely to be tagged.

```
Browse[2]> count(dataframe, var=c("STREET_TYPE", "Type"))
   STREET_TYPE Type freq
1         CSAH  TAG   82
2         CSAH  TOW   69
3        LOCAL  TAG  165
4        LOCAL  TOW   64
5          MSA  TAG  181
6          MSA  TOW  210
7       PRIVAT  TOW    2
8        PRKBD  TOW    3
9          RES  TAG 2546
10         RES  TOW  616
11         ROW  TOW    1
12        STFR  TOW    1
13         STH  TAG   20
14         STH  TOW   13
15        UofM  TAG    6
16        UofM  TOW    1
```

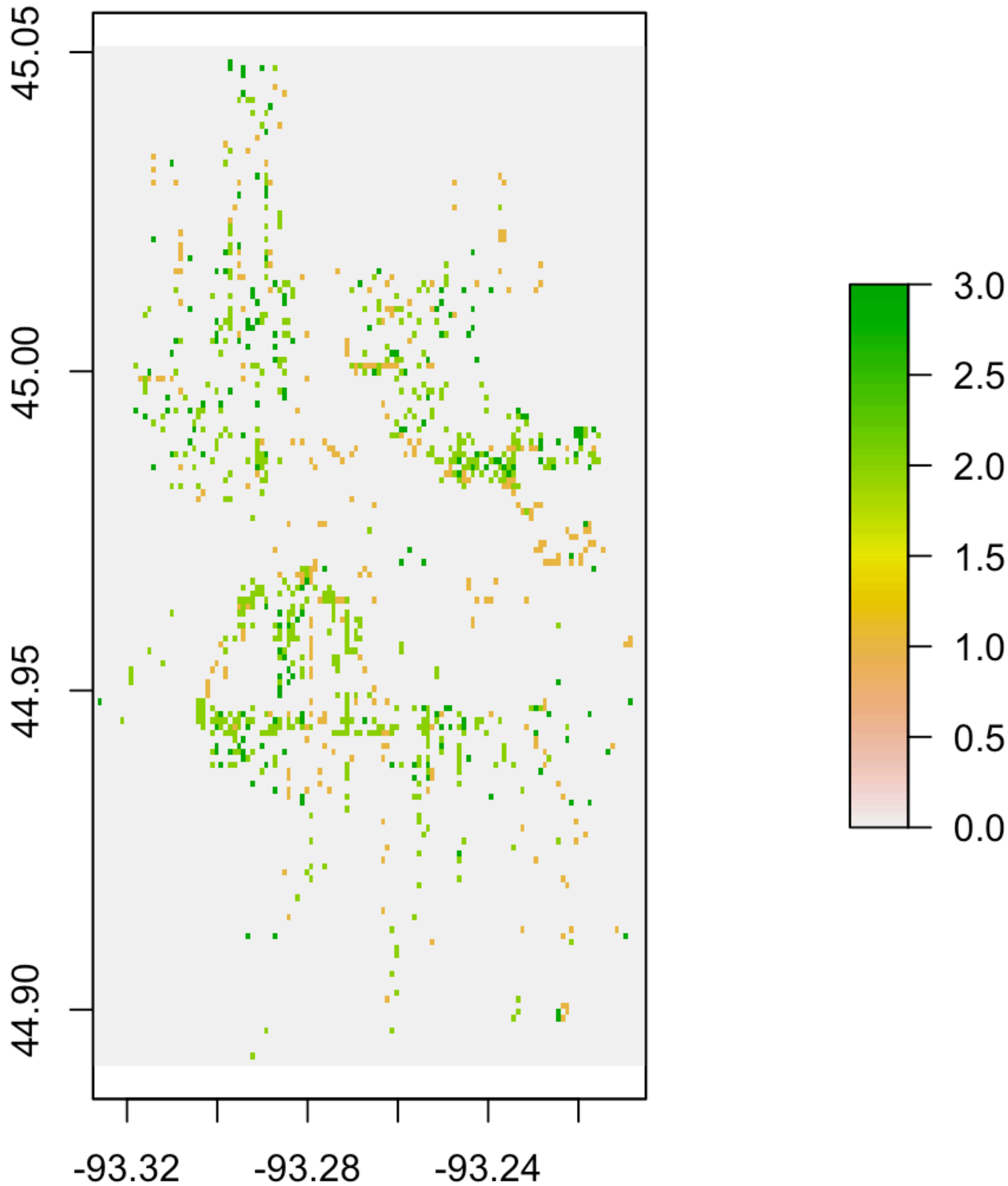
Same data, broken down by Day

```
Browse[2]> count(dataframe, var=c("STREET_TYPE", "Day", "Type"))
   STREET_TYPE Day Type freq
1         CSAH   1  TAG    4
2         CSAH   1  TOW   57
3         CSAH   2  TAG   52
4         CSAH   2  TOW    9
5         CSAH   3  TAG   26
6         CSAH   3  TOW    3
7        LOCAL   1  TAG   13
8        LOCAL   1  TOW   32
9        LOCAL   2  TAG   79
10       LOCAL   2  TOW   14
11       LOCAL   3  TAG   73
12       LOCAL   3  TOW   18
13         MSA   1  TAG   74
14         MSA   1  TOW  125
15         MSA   2  TAG   64
16         MSA   2  TOW   51
17         MSA   3  TAG   43
18         MSA   3  TOW   34
19      PRIVAT   2  TOW    2
20       PRKBD   1  TOW    1
21       PRKBD   2  TOW    2
22         RES   1  TAG   49
23         RES   1  TOW   81
24         RES   2  TAG 1511
25         RES   2  TOW  311
26         RES   3  TAG  986
27         RES   3  TOW  224
28         ROW   1  TOW    1
29        STFR   1  TOW    1
30         STH   1  TAG   10
31         STH   1  TOW    7
32         STH   2  TAG    8
33         STH   2  TOW    4
34         STH   3  TAG    2
35         STH   3  TOW    2
36        UofM   1  TAG    6
37        UofM   3  TOW    1
```

Probability Raster

A raster was created from each of the columns of interest.

Example raster – distribution of snow emergency day (Day 1, Day 2, or Day 3) for tags and tows.



I attempted to run the predict function using the rasters, and the output from the Random Forest analysis, but the output raster was always blank.

Code follows. The call to predict is at the end of the script. Any suggestions or pointers for what I'm missing would be gratefully received!

R project and data file at https://github.com/claraj/snow_emergencies

```
# Using Random Forest prediction on sample snow emergency data

library("randomForest")
library("raster")

setwd("/Users/student1/Development/r/snow_proj/data")

dataframe <- read.csv(file="SNOW_TAG_TOW_TYPES.csv")
head(dataframe)

# Ward (1, 2, 3....), Tow_Zone (1 - 6), Day (1, 2, 3) are numerical
and interpreted as numeric type.
# But here, they should be treated as categorical data, so convert to
factors
dataframe$Ward <- factor(dataframe$Ward)
dataframe$Tow_Zone <- factor(dataframe$Tow_Zone)
# dataframe$Day <- factor(dataframe$Day)

# Create categories for driving distance and driving duration
# Help from http://rcompanion.org/handbook/E_05.html categorizing data
per_00 <- min(dataframe$distance)
per_25 <- quantile(dataframe$distance, 0.25)
per_50 <- quantile(dataframe$distance, 0.5)
per_75 <- quantile(dataframe$distance, 0.55)
per_100 <- max(dataframe$distance)

dataframe$distanceCat[dataframe$distance >= per_00 &
dataframe$distance < per_25] = 1
dataframe$distanceCat[dataframe$distance >= per_25 &
dataframe$distance < per_50] = 2
dataframe$distanceCat[dataframe$distance >= per_50 &
dataframe$distance < per_75] = 3
dataframe$distanceCat[dataframe$distance >= per_75 &
dataframe$distance <= per_100] = 4

# Repeat for duration. Todo look up if there's a built-in way to do
this in R
per_00 <- min(dataframe$duration)
per_25 <- quantile(dataframe$duration, 0.25)
per_50 <- quantile(dataframe$duration, 0.5)
```

```
per_75 <- quantile(dataframe$duration, 0.55)
per_100 <- max(dataframe$duration)

dataframe$durationCat[dataframe$duration >= per_00 &
dataframe$duration < per_25] = 1
dataframe$durationCat[dataframe$duration >= per_25 &
dataframe$duration < per_50] = 2
dataframe$durationCat[dataframe$duration >= per_50 &
dataframe$duration < per_75] = 3
dataframe$durationCat[dataframe$duration >= per_75 &
dataframe$duration <= per_100] = 4

# Create a numerical column from Tag and Tow for Random Forest
dataframe$was_tow[dataframe$Type == "TOW"] = 1
dataframe$was_tow[dataframe$Type == "TAG"] = 0

# Save categories to file
summary(dataframe)
write.csv(dataframe, "categorize_snow_emergency.csv")

############### Running Random Forest Model #################3

# Run the random forest model with the columns given

# random_forest <- randomForest( Type ~ Ward + Community + Day +
Tow_Zone + STREET_TYPE + distanceCat + durationCat, data=dataframe,
ntree=500, importance=TRUE, proximity=TRUE)
random_forest <- randomForest( was_tow ~ Ward + Community + Day +
Tow_Zone + STREET_TYPE + distanceCat + durationCat, data=dataframe,
ntree=500, importance=TRUE, proximity=TRUE)

importance(random_forest)
# dev.off()
varImpPlot(random_forest)

# Day is by far the most important factor.

table(dataframe[dataframe$Day == 1, ]$Type)
table(dataframe[dataframe$Day == 2, ]$Type)
table(dataframe[dataframe$Day == 3, ]$Type)

############### Creating predictive raster layer ###############


# Create coordinates for dataframe, which converts dataframe to a
SpatialPointsDataFrame
```

```
coordinates(dataframe) <- ~Longitude+Latitude

## Create rasters for each column of interest

# Extent of points in Minneapolis
lonMin <- -93.327527
lonMax <- -93.205057
latMin <- 44.891232
latMax <- 45.050941

cell_size <- 0.001
ncols <- (( lonMax - lonMin) / cell_size) + 1
nrows <- (( latMax - latMin) / cell_size) + 1

ext <- extent(lonMin, lonMax, latMax, latMax)

r_d <- raster(ncols=ncols, nrows=nrows, xmn=lonMin, xmx=lonMax,
ymn=latMin, ymx=latMax)
day_raster = rasterize(dataframe, r_d, "Day", fun="min",
filename="Day.tif", background=0, overwrite=TRUE)

r_di <- raster(ncols=ncols, nrows=nrows, xmn=lonMin, xmx=lonMax,
ymn=latMin, ymx=latMax)
distance_raster = rasterize(dataframe, r_di, "distanceCat", fun="min",
filename="distanceCat.tif", background=0, overwrite=TRUE)

r_du <- raster(ncols=ncols, nrows=nrows, xmn=lonMin, xmx=lonMax,
ymn=latMin, ymx=latMax)
duration_raster = rasterize(dataframe, r_du, "durationCat", fun=mean,
filename="durationCat.tif", background=0, overwrite=TRUE)

# Everything else is a factor - how to convert to Raster? What value
to write for factor's levels?
r_w <- raster(ncols=ncols, nrows=nrows, xmn=lonMin, xmx=lonMax,
ymn=latMin, ymx=latMax)
ward_raster = rasterize(dataframe, r_w, "Ward", fun=function(x, na.rm)
{ max(as.numeric(x)) },  background=0, filename="Ward.tif",
overwrite=TRUE)

r_t <- raster(ncols=ncols, nrows=nrows, xmn=lonMin, xmx=lonMax,
ymn=latMin, ymx=latMax)
tow_zone_raster = rasterize(dataframe, r_t, "Tow_Zone",
fun=function(x, na.rm) { max(as.numeric(x)) }, background=0,
filename="Tow_Zone.tif", overwrite=TRUE)
```

```
r_c <- raster(ncols=ncols, nrows=nrows, xmn=lonMin, xmx=lonMax,
ymn=latMin, ymx=latMax)
community_raster = rasterize(dataframe, r_c, "Community",
fun=function(x, na.rm) { max(as.numeric(x)) }, background=0,
filename="Community.tif", overwrite=TRUE)

r_s <- raster(ncols=ncols, nrows=nrows, xmn=lonMin, xmx=lonMax,
ymn=latMin, ymx=latMax)
street_type_raster = rasterize(dataframe, r_s, "STREET_TYPE",
fun=function(x, na.rm) { max(as.numeric(x)) }, background=0,
filename="STREET_TYPE.tif", overwrite=TRUE)

# Vector of rasters
raster_combo <- c(ward_raster, community_raster, day_raster,
tow_zone_raster, street_type_raster, distance_raster, duration_raster)

# Set the extents of the rasters to be the same. Uses real, actual, R
syntax
for (r in raster_combo) {
  extent(r) <- ext
}

# Create a stack of all the rasters
raster_stack <- stack(raster_combo)

# set names, must match column names in the dataframe used to generate
names(raster_stack) <- c("Ward", "Community", "Day", "Tow_Zone",
"STREET_TYPE", "distanceCat", "durationCat")

# The output raster is blank. What am I doing wrong?
predict_raster_layer <- predict(raster_stack, random_forest,
"predictive_snow_emergency_raster.tif", overwrite=TRUE)
#dev.off()
plot(predict_raster_layer)
```