Practicum 1

Jordan Lian

2/26/2021

```
library(ggplot2)
library(tidyverse)
library(lubridate)
library(ggpubr)
library(tidyr)
library(Metrics)
```

```
## -- Attaching packages ---------------------------------------------------------------------------
```

```
## v tibble  3.0.3     v dplyr   1.0.2
## v tidyr   1.1.2     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.5.0
## v purrr   0.3.4
```

```
## -- Conflicts ------------------------------------------------------------------------------------
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
##
## Attaching package: 'lubridate'
```

```
## The following objects are masked from 'package:base':
##
##     date, intersect, setdiff, union
```

Problem 1 (60 Points)

1. (0 pts) Download the data set Taxi Fare NYC (May 2020) (source: NYC Taxi & Limousine Commission (Links to an external site.)).

```
origin_taxi <- read_csv('yellow_tripdata_2020-05.csv')
```

```
## Parsed with column specification:
## cols(
##   VendorID = col_double(),
##   tpep_pickup_datetime = col_datetime(format = ""),
##   tpep_dropoff_datetime = col_datetime(format = ""),
```

```
##   passenger_count = col_double(),
##   trip_distance = col_double(),
##   RatecodeID = col_double(),
##   store_and_fwd_flag = col_character(),
##   PULocationID = col_double(),
##   DOLocationID = col_double(),
##   payment_type = col_double(),
##   fare_amount = col_double(),
##   extra = col_double(),
##   mta_tax = col_double(),
##   tip_amount = col_double(),
##   tolls_amount = col_double(),
##   improvement_surcharge = col_double(),
##   total_amount = col_double(),
##   congestion_surcharge = col_double()
## )
```

```
taxi <- origin_taxi
head(taxi)
```

```
## # A tibble: 6 x 18
##   VendorID tpep_pickup_dateti~ tpep_dropoff_datet~ passenger_count trip_distance
##      <dbl> <dttm>              <dttm>                         <dbl>         <dbl>
## 1        1 2020-05-01 00:02:28 2020-05-01 00:18:07                1           0
## 2        1 2020-05-01 00:23:21 2020-05-01 00:26:01                2           0.4
## 3        1 2020-05-01 00:54:58 2020-05-01 00:57:11                1           0.3
## 4        1 2020-05-01 00:07:10 2020-05-01 00:12:46                1           1.7
## 5        1 2020-05-01 00:55:47 2020-05-01 01:01:54                0           0.9
## 6        1 2020-05-01 00:38:37 2020-05-01 01:03:08                0          12
## # ... with 13 more variables: RatecodeID <dbl>, store_and_fwd_flag <chr>,
## #   PULocationID <dbl>, DOLocationID <dbl>, payment_type <dbl>,
## #   fare_amount <dbl>, extra <dbl>, mta_tax <dbl>, tip_amount <dbl>,
## #   tolls_amount <dbl>, improvement_surcharge <dbl>, total_amount <dbl>,
## #   congestion_surcharge <dbl>
```

2. (0 pts) Explore the data set to get a sense of the data and to get comfortable with it.

```
summary(taxi)
```

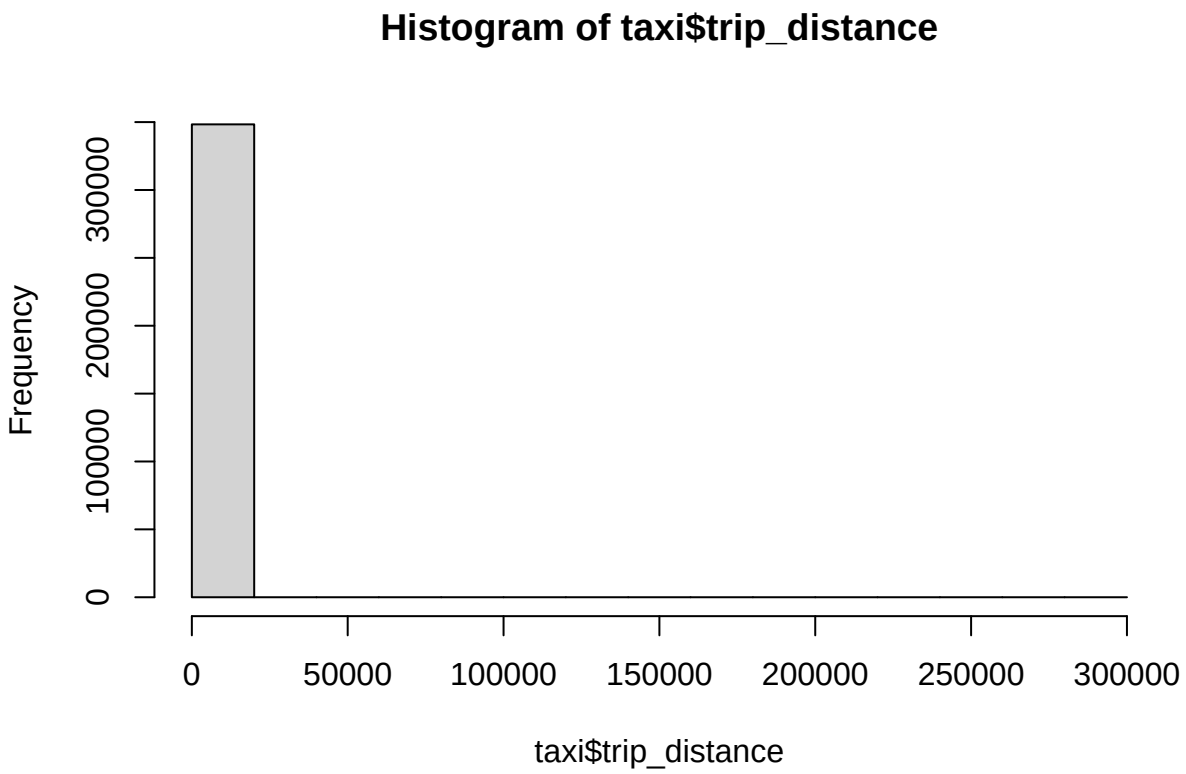
```
##     VendorID     tpep_pickup_datetime          tpep_dropoff_datetime
##  Min.   :1.00    Min.   :2008-12-31 23:05:47   Min.   :2008-12-31 23:33:33
##  1st Qu.:1.00    1st Qu.:2020-05-09 07:18:09   1st Qu.:2020-05-09 07:34:03
##  Median :2.00    Median :2020-05-16 17:21:48   Median :2020-05-16 17:33:30
##  Mean   :1.56    Mean   :2020-05-16 21:47:35   Mean   :2020-05-16 22:01:05
##  3rd Qu.:2.00    3rd Qu.:2020-05-24 16:29:27   3rd Qu.:2020-05-24 16:43:04
##  Max.   :2.00    Max.   :2020-11-01 15:41:04   Max.   :2020-11-01 15:57:02
##  NA's   :58891
##  passenger_count trip_distance        RatecodeID    store_and_fwd_flag
##  Min.   :0.00    Min.   :0.00e+00   Min.   : 1.00   Length:348371
##  1st Qu.:1.00    1st Qu.:1.05e+00   1st Qu.: 1.00   Class :character
##  Median :1.00    Median :1.99e+00   Median : 1.00   Mode  :character
```

```
##  Mean   :1.31    Mean   :8.33e+00   Mean   : 1.04
##  3rd Qu.:1.00    3rd Qu.:4.25e+00   3rd Qu.: 1.00
##  Max.   :9.00    Max.   :2.97e+05   Max.   :99.00
##  NA's   :58891                      NA's   :58891
##   PULocationID    DOLocationID    payment_type    fare_amount
##  Min.   :  1.0   Min.   :  1.0   Min.   :1.00    Min.   :  -240.0
##  1st Qu.: 90.0   1st Qu.: 75.0   1st Qu.:1.00    1st Qu.:     6.0
##  Median :143.0   Median :143.0   Median :1.00    Median :     9.0
##  Mean   :152.6   Mean   :148.5   Mean   :1.42    Mean   :    14.9
##  3rd Qu.:231.0   3rd Qu.:230.0   3rd Qu.:2.00    3rd Qu.:    16.0
##  Max.   :265.0   Max.   :265.0   Max.   :4.00    Max.   :429496.7
##                                  NA's   :58891
##      extra            mta_tax          tip_amount       tolls_amount
##  Min.   :-4.5000   Min.   :-0.5000   Min.   :-11.060   Min.   :-13.7500
##  1st Qu.: 0.0000   1st Qu.: 0.5000   1st Qu.:  0.000   1st Qu.:  0.0000
##  Median : 0.0000   Median : 0.5000   Median :  0.010   Median :  0.0000
##  Mean   : 0.9563   Mean   : 0.4902   Mean   :  1.428   Mean   :  0.4357
##  3rd Qu.: 2.5000   3rd Qu.: 0.5000   3rd Qu.:  2.450   3rd Qu.:  0.0000
##  Max.   :65.5300   Max.   : 3.3000   Max.   :442.180   Max.   :200.0000
##
##  improvement_surcharge  total_amount      congestion_surcharge
##  Min.   :-0.3000       Min.   :  -244.3   Min.   :-2.50
##  1st Qu.: 0.3000       1st Qu.:    10.3   1st Qu.: 0.00
##  Median : 0.3000       Median :    13.8   Median : 2.50
##  Mean   : 0.2971       Mean   :    19.7   Mean   : 1.84
##  3rd Qu.: 0.3000       3rd Qu.:    21.3   3rd Qu.: 2.50
##  Max.   : 0.3000       Max.   :429562.2   Max.   : 2.50
##
```

3. (5 pts) Create a histogram of column 5 (trip distance) and overlay a normal curve.

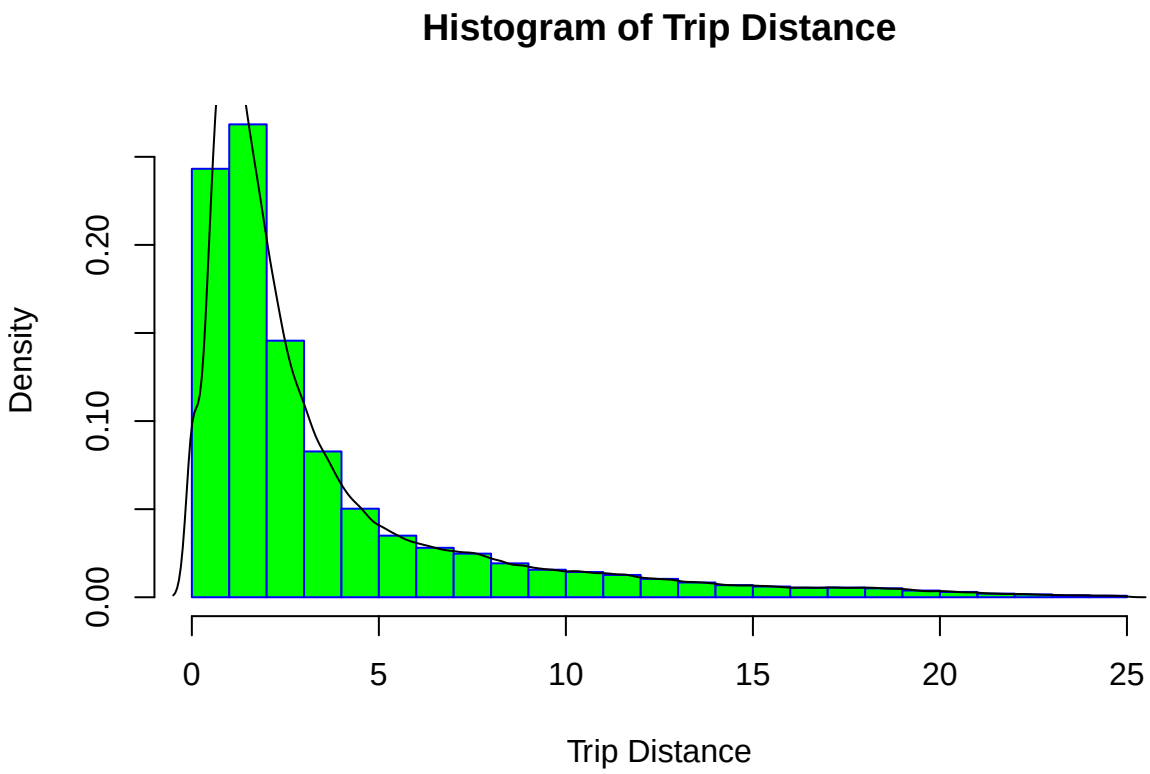
I plotted the initial data, but the histogram did not produce any meaningful result.

```
# Original Plot
hist(taxi$trip_distance)
```



So I took the values where the trip distance was less than or equal to 25, and the histogram produced showed some results that resembled a histogram. Plotting a normal curve was tough, so I just overlaid the density curve.

```
# Edit
mod_taxi <- taxi[taxi$trip_distance <= 25, ]
hist(mod_taxi$trip_distance,
     main = "Histogram of Trip Distance",
     xlab = "Trip Distance",
     border = "blue",
     col = "green",
     freq=FALSE)
lines(density(mod_taxi$trip_distance))
```



4. (3 pts) Test normality of column 5 by performing either a Shapiro-Wilk (tutorial (Links to an external site.)) or Kolmogorov-Smirnof test. Describe what you found.

I generated a few visuals for the trip distance, but it didn't generate much. I also randomly sampled 5000 points for the Shapiro-Wilk test because the limit was 5,000 rows, and our dataset had 348,371 rows.

```
mod_taxi <- taxi[sample(nrow(taxi), 5000), ]
shapiro.test(mod_taxi$trip_distance)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  mod_taxi$trip_distance
## W = 0.0036131, p-value < 2.2e-16
```

From the results, the p-value < 0.05 which implies that the distribution of the data is significantly different from a normal distribution. In other words, we cannot assume the normality.

5. (3 pts) Identify any outliers for the columns using a z-score deviation approach, i.e., consider any values that are more than 2 standard deviations from the mean as outliers. Which are your outliers for each column? What would you do? Summarize potential strategies in your notebook.

First you have to get the z-scores such that the mean is 0 and standard deviation is 1. Then get all of the values greater than 2 or less than -2.

Trip Distance

```
# Get z-scores
mod_taxi <- taxi %>%
  mutate(zscore = (trip_distance - mean(trip_distance))/sd(trip_distance))

# Get values with magnitudes greater than 2
mod_taxi <- mod_taxi[abs(mod_taxi$zscore) >= 2, ]
head(mod_taxi$zscore)
```

```
## [1]  14.15110 383.28673 164.37510  59.72324 166.30288 191.43151
```

Fare Amount

```
# Get z-scores
mod_taxi <- taxi %>%
  mutate(zscore = (fare_amount - mean(fare_amount))/sd(fare_amount))

# Get values with magnitudes greater than 2
mod_taxi <- mod_taxi[abs(mod_taxi$zscore) >= 2, ]
head(mod_taxi$zscore)
```

```
## [1] 590.1198
```

Extra

```
# Get z-scores
mod_taxi <- taxi %>%
  mutate(zscore = (extra - mean(extra))/sd(extra))

# Get values with magnitudes greater than 2
mod_taxi <- mod_taxi[abs(mod_taxi$zscore) >= 2, ]
head(mod_taxi$zscore)
```

```
## [1] 2.03701 2.03701 2.03701 2.03701 2.03701 2.03701
```

MTA Tax

```
# Get z-scores
mod_taxi <- taxi %>%
  mutate(zscore = (mta_tax - mean(mta_tax))/sd(mta_tax))

# Get values with magnitudes greater than 2
mod_taxi <- mod_taxi[abs(mod_taxi$zscore) >= 2, ]
head(mod_taxi$zscore)
```

```
## [1] -11.810248 -11.810248  -5.846407 -11.810248 -11.810248 -11.810248
```

Tip Amount

```r
# Get z-scores
mod_taxi <- taxi %>%
  mutate(zscore = (tip_amount - mean(tip_amount))/sd(tip_amount))

# Get values with magnitudes greater than 2
mod_taxi <- mod_taxi[abs(mod_taxi$zscore) >= 2, ]
head(mod_taxi$zscore)
```

```
## [1] 2.399833 7.401071 2.005313 2.045164 2.304192 4.312658
```

Tolls Amount

```r
# Get z-scores
mod_taxi <- taxi %>%
  mutate(zscore = (tolls_amount - mean(tolls_amount))/sd(tolls_amount))

# Get values with magnitudes greater than 2
mod_taxi <- mod_taxi[abs(mod_taxi$zscore) >= 2, ]
head(mod_taxi$zscore)
```

```
## [1] 2.974793 2.974793 6.177594 2.974793 2.974793 2.974793
```

Improvement Surcharge

```r
# Get z-scores
mod_taxi <- taxi %>%
  mutate(zscore = (improvement_surcharge - mean(improvement_surcharge))/sd(improvement_surcharge))

# Get values with magnitudes greater than 2
mod_taxi <- mod_taxi[abs(mod_taxi$zscore) >= 2, ]
head(mod_taxi$zscore)
```

```
## [1] -14.72651 -14.72651 -14.72651 -14.72651 -14.72651 -14.72651
```

Total Amount

```r
# Get z-scores
mod_taxi <- taxi %>%
  mutate(zscore = (total_amount - mean(total_amount))/sd(total_amount))

# Get values with magnitudes greater than 2
mod_taxi <- mod_taxi[abs(mod_taxi$zscore) >= 2, ]
head(mod_taxi$zscore)
```

```
## [1] 590.0964
```

Congestion Surcharge

```r
# Get z-scores
mod_taxi <- taxi %>%
  mutate(zscore = (congestion_surcharge - mean(congestion_surcharge))/sd(congestion_surcharge))

# Get values with magnitudes greater than 2
mod_taxi <- mod_taxi[abs(mod_taxi$zscore) >= 2, ]
head(mod_taxi$zscore)
```

```
## [1] -3.879937 -3.879937 -3.879937 -3.879937 -3.879937 -3.879937
```

Depending on what I want to do with the data, it might be best to remove the outliers in certain cases. You can also use the median instead of the mean if there outliers significantly skew the mean and the statistic is discrete.

6. (2 pts) Add a new column to the data set called trip_time that is the time of the trip in minutes, calculated from the tpep_pickup_datetime and tpep_dropoff_datetime columns.

```r
taxi$trip_time <- difftime(taxi$tpep_dropoff_datetime, taxi$tpep_pickup_datetime,
                           units="mins")
```

7. (2 pts) Remove any negative values for the column fare_amount.

```r
taxi <- taxi[taxi$fare_amount >= 0, ]
```

8. (3 pts) Create a new new data set (taxi_data_full) only containing columns (in this order): tip_amount, fare_amount, trip_distance, trip_time, congestion_surcharge, payment_type.

```r
taxi_data_full <- select(taxi, tip_amount, fare_amount, trip_distance, trip_time,
                         congestion_surcharge, payment_type)
taxi_data_full
```

```
## # A tibble: 346,842 x 6
##    tip_amount fare_amount trip_distance trip_time  congestion_surc~ payment_type
##         <dbl>       <dbl>         <dbl> <drtn>                <dbl>        <dbl>
##  1        2.4        12.2           0   15.650000~              2.5            1
##  2        0.5         4             0.4  2.666667~               0             1
##  3        0           3.5           0.3  2.216667~               0             2
##  4        0           7             1.7  5.600000~               2.5           2
##  5        1.2         6             0.9  6.116667~               2.5           1
##  6        0          35.5          12   24.516667~               2.5           1
##  7        0          12             2.9 17.966667~               2.5           2
##  8        2.15        7             1.5  6.166667~               2.5           1
##  9        0           9             2    9.816667~               2.5           2
## 10        0           4.5           0.5  3.400000~               2.5           2
## # ... with 346,832 more rows
```

9. (3 pts) Standardize the scales of the numeric columns, except the first one (tip_amount), using z-score standardization.

```
standard <- taxi_data_full %>%
  mutate_at(c(2:5), funs(c(scale(.))))
standard
```

```
## # A tibble: 346,842 x 6
##    tip_amount fare_amount trip_distance trip_time congestion_surch~ payment_type
##         <dbl>       <dbl>         <dbl>     <dbl>             <dbl>        <dbl>
##  1       2.4     -0.00392      -0.0108     0.0449             0.589            1
##  2       0.5     -0.0152       -0.0103    -0.230             -1.70             1
##  3       0       -0.0158       -0.0104    -0.240             -1.70             2
##  4       0       -0.0110       -0.00858   -0.168              0.589            2
##  5       1.2     -0.0124       -0.00961   -0.157              0.589            1
##  6       0        0.0280        0.00469    0.233              0.589            1
##  7       0       -0.00419      -0.00703    0.0941             0.589            2
##  8       2.15    -0.0110       -0.00884   -0.156              0.589            1
##  9       0       -0.00830      -0.00819   -0.0788             0.589            2
## 10       0       -0.0145       -0.0101    -0.215              0.589            2
## # ... with 346,832 more rows
```

10. (4 pts) The data set is sorted, so creating a validation data set requires random selection of elements. Create a stratified sample where you randomly select 15% of each of the cases for each payment type to be part of the validation data set. The remaining cases will form the training data set.

Firstly, I got rid of the NA values for payment_type, and I split the dataset according to the four payment types

```
# Get rid of NA values
mod_taxi <- taxi_data_full[!is.na(taxi_data_full$payment_type), ]

# Get unique values
unique(mod_taxi$payment_type)
```

```
## [1] 1 2 4 3
```

```
# Split the dataset according to the unique values
pay_one <- mod_taxi[mod_taxi$payment_type == 1, ]
pay_two <- mod_taxi[mod_taxi$payment_type == 2, ]
pay_three <- mod_taxi[mod_taxi$payment_type == 3, ]
pay_four <- mod_taxi[mod_taxi$payment_type == 4, ]
```

Secondly, I used the four split up datasets and took 15% of each as the validation, and the other 85% as the training set.

```
# Random seed
set.seed(123)

# Payment Type = 1
```

```
sample_1 <- sample.int(n = nrow(pay_one), size = floor(0.15*nrow(pay_one)), replace = F)
val_1 <- pay_one[sample_1, ]
train_1  <- pay_one[-sample_1, ]

# Payment Type = 2
sample_2 <- sample.int(n = nrow(pay_two), size = floor(0.15*nrow(pay_two)), replace = F)
val_2 <- pay_two[sample_2, ]
train_2 <- pay_two[-sample_2, ]

# Payment Type = 3
sample_3 <- sample.int(n = nrow(pay_three), size = floor(0.15*nrow(pay_three)),
                       replace = F)
val_3 <- pay_three[sample_3, ]
train_3 <- pay_three[-sample_3, ]

# Payment Type = 4
sample_4 <- sample.int(n = nrow(pay_four), size = floor(0.15*nrow(pay_four)),
                       replace = F)
val_4 <- pay_four[sample_4, ]
train_4 <- pay_four[-sample_4, ]
```

I then combined the validation and training datasets.

```
validation <- rbind(val_1, val_2, val_3, val_4)
training <- rbind(train_1, train_2, train_3, train_4)

head(validation)
```

```
## # A tibble: 6 x 6
##   tip_amount fare_amount trip_distance trip_time   congestion_surc~ payment_type
##        <dbl>       <dbl>         <dbl> <drtn>                <dbl>        <dbl>
## 1       3.66        15            4.18 11.200000 ~             2.5            1
## 2       1.56         4.5          0.94  2.366667 ~             2.5            1
## 3       3.65         8            1.7   8.900000 ~             2.5            1
## 4       1.45         5.5          1.1   4.483333 ~             0              1
## 5       2.35         8.5          2.3   6.500000 ~             2.5            1
## 6       2.45         9            1.7   9.616667 ~             2.5            1
```

```
head(training)
```

```
## # A tibble: 6 x 6
##   tip_amount fare_amount trip_distance trip_time   congestion_surc~ payment_type
##        <dbl>       <dbl>         <dbl> <drtn>                <dbl>        <dbl>
## 1       2.4         12.2          0    15.650000 ~             2.5            1
## 2       0.5          4            0.4   2.666667 ~             0              1
## 3       1.2          6            0.9   6.116667 ~             2.5            1
## 4       0           35.5         12    24.516667 ~             2.5            1
## 5       2.15         7            1.5   6.166667 ~             2.5            1
## 6       3.58        13            3.37 12.533333 ~             0              1
```

11. (20 pts) Implement the k-NN algorithm in R (do not use an implementation of k-NN from a package) and use your algorithm with a k=5 to predict the tip amount for the following new case:

fare_amount = 17.5, trip_distance = 4.8, trip_time = 28, congestion_surcharge = 2.5, payment_type = 1

Use only the training data set. Note that you need to normalize the values of the new cases the same way as you normalized the original data. If the data set is too large to handle on your computer, then create a smaller training data set by randomly sampling the original data set.

```
# Create normalization function
normalize <- function(x) {
  return ((x - min(x)) / (max(x) - min(x))) }

# Put the new case values in a list
unknown <- c(17.5, 4.8, 28, 2.5, 1)

# Convert the trip time values to numeric type so they can be normalized
training$trip_time <- as.numeric(training$trip_time)

# Create new dataframe with the new list and training dataset, and apply the normalization function
df <- as.data.frame(lapply(rbind(unknown, training[, 2:6]), normalize))

# Create array for predicted values, initialize to 0
pred_values <- c(0)

# Count rows in training dataset
n <- as.numeric(count(training))
# Iterate through data frame and get all the distances and predicted values
for(i in 2:n){
  # Omit NA values
  mod_df <- na.omit(df[c(1, i), ])

  # Store values in list
  pred_values <- c(pred_values, dist(mod_df))
}

# Get the closest distances and print the top 5 (k=5)
closest <- order(pred_values)
training[c(closest[2:6]) - 1, 1]
```

```
## # A tibble: 5 x 1
##   tip_amount
##        <dbl>
## 1       2.5
## 2       4.75
## 3       7.25
## 4       7.74
## 5       5.16
```

```
# Get the average of the 5 values
sum(training[c(closest[2:6]) - 1, 1]) / 5
```

```
## [1] 5.48
```

12. (5 pts) Apply the knn function from the class package with k=5 and redo the cases from Question (11). Compare your answers.

```
test_labels <- validation[, 1]
train_labels <- train_class[, 1]

library(class)
prc_test_pred <- knn(train = train_class, test = validation, cl = train_labels, k=5)

library(gmodels)
CrossTable(x = test_labels, y = prc_test_pred, prop.chisq = FALSE)
```

13. (10 pts) Using kNN from the class package, create a plot of k (x-axis) from 2 to 8 versus accuracy (percentage of correct classifications) using ggplot.

Problem 2 (30 Points)

1. (0 pts) Investigate this data set of home prices in King County (USA) (Links to an external site.).

```
origin_home <- read_csv('kc_house_data.csv')
```

```
## Parsed with column specification:
## cols(
##   .default = col_double(),
##   id = col_character(),
##   date = col_datetime(format = "")
## )

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

```
home <- origin_home
```

2. (5 pts) Save the price column in a separate vector/dataframe called target_data. Move all of the columns except the ID, date, price, yr_renovated, zipcode, lat, long, sqft_living15, and sqft_lot15 columns into a new data frame called train_data.

```
target_data <- home$price
train_data <- select(home, -one_of('id', 'date', 'price', 'yr_renovated', 'zipcode',
                                   'lat', 'long', 'sqft_living15', 'sqft_lot15'))
```

3. (5 pts) Normalize all of the columns (except the boolean columns waterfront and view) using min-max normalization.

```r
# create normalization function
normalize <- function(x) {
  return ((x - min(x)) / (max(x) - min(x))) }

# Normalize data
norm_home <- as.data.frame(lapply(train_data[-c(6:7)], normalize))

# Add the untouched columns to the normalized dataset
norm_home <- cbind(norm_home, train_data$waterfront, train_data$view)

# Change column names
norm_home <- norm_home %>%
  rename(waterfront = "train_data$waterfront", view = "train_data$view")

# Normalized Dataset
head(norm_home)
```

```
##     bedrooms bathrooms sqft_living    sqft_lot floors condition     grade
## 1 0.09090909   0.12500  0.06716981 0.003107511    0.0       0.5 0.5000000
## 2 0.09090909   0.28125  0.17207547 0.004071869    0.4       0.5 0.5000000
## 3 0.06060606   0.12500  0.03622642 0.005742535    0.0       0.5 0.4166667
## 4 0.12121212   0.37500  0.12603774 0.002713772    0.0       1.0 0.5000000
## 5 0.09090909   0.25000  0.10490566 0.004579490    0.0       0.5 0.5833333
## 6 0.12121212   0.56250  0.38716981 0.061429370    0.0       0.5 0.8333333
##   sqft_above sqft_basement  yr_built waterfront view
## 1 0.09758772    0.00000000 0.4782609          0    0
## 2 0.20614035    0.08298755 0.4434783          0    0
## 3 0.05263158    0.00000000 0.2869565          0    0
## 4 0.08333333    0.18879668 0.5652174          0    0
## 5 0.15241228    0.00000000 0.7565217          0    0
## 6 0.39473684    0.31742739 0.8782609          0    0
```

4. (12 pts) Build a function called knn.reg that implements a regression version of kNN that averages the prices of the k nearest neighbors using a weighted average where the weight is 3 for the closest neighbor, 2 for the second closest and 1 for the remaining neighbors (recall that a weighted average requires that you divide the sum product of the weight and values by the sum of the weights).

It must use the following signature:

knn.reg (new_data, target_data, train_data, k)

where new_data is a data frame with new cases, target_data is a data frame with a single column of prices from (2), train_data is a data frame with the features from (2) that correspond to a price in target_data, and k is the number of nearest neighbors to consider. It must return the predicted price.

```r
knn.reg <- function(new_data, target_data, train_data, k){
  # Add the target column for future reference
  train_data <- cbind(target_data, train_data)

  # Normalize the data
  df <- as.data.frame(lapply(rbind(new_data, train_data[,2:12]), normalize))
```

```
  # Create vector to store predicted prices
  pred_values <- c(0)

  # Get total rows for data frame
  n <- as.numeric(count(train_data))

  # Iterate through data frame and get all the distances and predicted values
  for(i in 2:n){
    mod_df <- na.omit(df[c(1, i), ])
    pred_values <- c(pred_values, dist(mod_df))
  }

  # Get the closest distances
  closest <- order(pred_values)

  # Get top 2 values
  top_2 <- train_data[c(closest[2:k]) - 1, 1]
  top_2

  # Get the average of the rest of the values
  rest <- mean(train_data[c(closest[k:n]) - 1, 1])
  top_3 <- c(top_2, rest)
  top_3

  # Get the weighted average
  weights <- c(3, 2, 1)
  pred_price <- weighted.mean(top_3, weights)
  return (pred_price)
    }
```

5. (4 pts) Forecast the price of this new home using your regression kNN using k = 3:

bedrooms = 3 | bathrooms = 3 | sqft_living = 4850 | sqft_lot = 11240 | floors = 3 | waterfront = 1 | view = 1 | condition = 3 | grade = 11 sqft_above = 2270 | sqft_basement = 820 | yr_built = 1986

```
# New Data
new_data <- c(3, 3, 4850, 11240, 3, 1, 1, 3, 11, 2270, 820, 1986)

# k = 3
k <- 3

# Call function from question 4
knn.reg(new_data, target_data, train_data, k)
```

```
## [1] 1624177
```

6. (4 pts) Calculate the Mean Squared Error (MSE) using a random sample of 10% of the data set as test data.

```
set.seed(1234)
test_data <- cbind(target_data, train_data)
```

```
sample <- sample.int(n = nrow(test_data), size = floor(0.10*nrow(test_data)), replace = F)
test <- test_data[sample, ]

demand <- test$target_data
n <- as.numeric(count(test))
prediction <- rep(NA, n)
for(i in 1:n){
  new_data <- as.numeric(test[i, 2:13])
  prediction[i] <- knn.reg(new_data, target_data, train_data, k)
}

mse(prediction, demand)
```

Problem 3 (10 Points)

1. (3 pts) Build a new data set with the four columns: tper, year, month, avg_price_sq_ft where tper is the time period starting at 1 (e.g., 1, 2, 3, 4, . . .), year and month are the year and month of the sale of a property extracted from the column date and avg_price_sq_ft is the average price per square foot of living space for all properties sold that month. The data set should contain the sales in order from least recent to most recent, i.e., the first column after the header column should be the property sold the furthest in the past.

```
# Create new column that combines the year and month (in that order)
home$mon_yr <- format(as.POSIXct(origin_home$date, format="%Y/%m/%d"),"%Y-%m")

# Group average price per living square footage by month and year
price_sqft <- home %>%
  group_by(mon_yr) %>%
  summarise(avg_price = mean(price/sqft_living))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
price_sqft
```

```
## # A tibble: 13 x 2
##    mon_yr  avg_price
##    <chr>       <dbl>
##  1 2014-05      263.
##  2 2014-06      265.
##  3 2014-07      260.
##  4 2014-08      260.
##  5 2014-09      260.
##  6 2014-10      263.
##  7 2014-11      259.
##  8 2014-12      255.
##  9 2015-01      257.
## 10 2015-02      260.
## 11 2015-03      274.
## 12 2015-04      279.
## 13 2015-05      284.
```

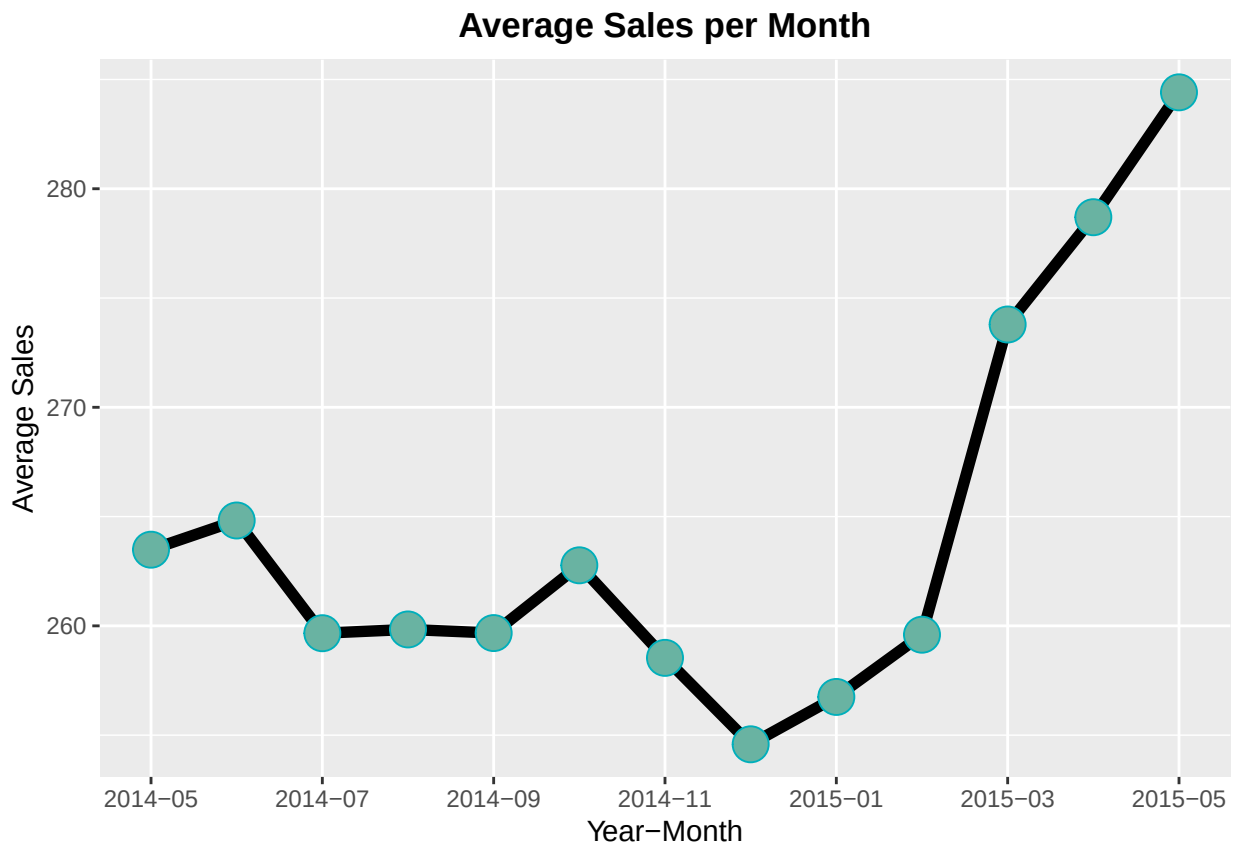
```r
# Convert dates to year
convert_year <- year(as.POSIXct(home$date, format="%Y/%m/%d"))
convert_month <- month(as.POSIXct(home$date, format="%Y/%m/%d"))

# Create new dataset
problem_3 <- data.frame("tper" = c(1:21613),
                        "year" = convert_year,
                        "month" = convert_month,
                        "avg_price_sq_ft" = price_sqft$avg_price[match(home$mon_yr,
                                                                       price_sqft$mon_yr)])
head(problem_3)
```

```
##   tper year month avg_price_sq_ft
## 1    1 2014    10        262.7701
## 2    2 2014    12        254.5807
## 3    3 2015     2        259.5951
## 4    4 2014    12        254.5807
## 5    5 2015     2        259.5951
## 6    6 2014     5        263.4855
```

2. (3 pts) Plot the average sales price per month as a time series line graph.

```r
# Use geom_line() and geom_point() to plot the points and connect them with group = 1
ggplot(data = price_sqft, aes(mon_yr, avg_price, group = 1)) +
  geom_line(color="black", size = 2) +
  geom_point(shape=21, fill="#69b3a2", size=6, color = "#00AFBB") +
  ggtitle("Average Sales per Month") +
  xlab("Year-Month") + ylab("Average Sales") +
  theme(plot.title = element_text(face = "bold", hjust = 0.5)) +
  scale_x_discrete(breaks = price_sqft$mon_yr[seq(1, length(price_sqft$mon_yr), by = 2)])
```



3. (4 pts) Forecast the average sales price for the next month is the time series using a weighted moving average of the most recent 3 months with weights of 2, 1.5, and 1.

```
# Store weights in a vector
weights <- c(1, 1.5, 2)

# Get the 3 most recent averages sales price
recent <- tail(price_sqft$avg_price, 3)

# Print the latest forecast for the next month
weighted.mean(recent, weights)
```

```
## [1] 280.1456
```