Term Project

Parth
pnshah@ucdavis.edu

Simerpal
sswhala@ucdavis.edu

Brian
ccblai@ucdavis.edu

Claire
chcwong@ucdavis.edu

March 18, 2021

Problem 1

The Sky Is Not the Limit: Multitasking Across GitHub Projects Parth Shah

The paper analyzed here is https://www.cs.ucdavis.edu/ filkov/papers/icse2016focus.pdf

The paper in question aims to analyze how Multitasking affects a programmers's performance. Tools like GitHub have allowed programmers to work on multiple Projects at ease, however, multitasking comes at a cognitive cost. Frequently switching projects and contexts can lead to distractions, sup-par work, and greater stress. This paper aims to analyze ecosystem-level data on a group of programmers working on a large collection of projects and in turn develop models to measure the rate and breadth of a developers' context-switching behavior. The paper manages to conclude that the most common reason for multitasking is interrelationships and dependencies between projects and that the rate of switching and breadth (number of projects) of a developer's work matter.

To reach the conclusion, the paper uses p-values and the 95% confidence intervals in order to test the hypothesis. However we know we cannot rely solely on them as that can lead to small P values even if the declared test hypothesis is correct, and can lead to large P values even if that hypothesis is incorrect. For instance, in the paper a p-value of 0.01 is achived for most of the factors however, factors such as "Feeling more productive" is a subjective matter and a p-value shoudn't be the only thing that decides the success of a research study.

Introduction

Database *train.csv*

In section B of the term project, we were given a data set of Porto taxi trips *train.csv*. The dataset was based off of 448 Taxis in the city of Porto, Portugal. The data was split among 9 categories: TRIP_ID, CALL_TYPE, ORIGIN_CALL, ORIGIN_STAND, TAXI_ID, TIMESTAMP, DAYTYPE, MISSING_DATA, POLYTIME and consisted of 1710669 objects sized at 1.94 GB. The TRIP_ID was a unique identifier for every trip and TAXI_ID was a unique identifier for the taxi driver who performed the trip. CALL_TYPE identified the method used to call the taxi: 'A' represented a call from central, 'B' represented if the taxi driver was demanded at a specific stand, and 'C' otherwise. ORIGIN_CALL represented the phone number of the customer who demanded the taxi (Sets CALL_TYPE to 'A'). ORIGIN_STAND gives each taxi stand a unique identifier (Sets CALL_TYPE to 'B') .TIMESTAMP shows the unix Timestamp of the start of the trip. DAYTYPE represents the type of day it is at the start of the trip: 'B ' being a holiday or special day, 'C' being the day before a type 'B' day, and 'A' being everything else.Lastly, POLYLINE contains a set of arrays of two elements of global coordination, the longitude and latitude.

Dealing with a massive dataset

In order to use the dataset, we had to import it into our R data table which at first we did by using the read() function. This created an issue because the data set was taking too long to load in and often times crashing the IDE due to insufficient memory. A temporary fix we decided to use was to only pass in 1000 rows of the full data set. This allowed us to get past the issues of read() and start working on the tasks in part B. In order to fully fix this problem and use the entire data set we decided to use fread()from the data.table library. This allowed us to read the data set faster and use less memory.

Tasks to solve

- Find density models that accurately represented the trip duration and the time when the driver was busy.
- Explore if the CALL_TYPE had an effect on the meantime of taxi trips
- Create models that predicted trip times, distances, . . . (Add other explicit models)
- Compare how those predictive models relate to one another.

Language and Libraries used

To solve these tasks our group used R to code the entire project as specified in the instructions. We also used the following libraries throughout the project

- Devtools: Used to assist in the installation of Regtools later in the project.
- Dplyr: Utilized this library to create and edit data frames.
- Data.table: Primarily used for fread() in order to speed up reading of *train.csv*.
- Regtools: Allowed us to utilize many machine learning models

Exploring the possibility of trip duration

The trip duration in this database

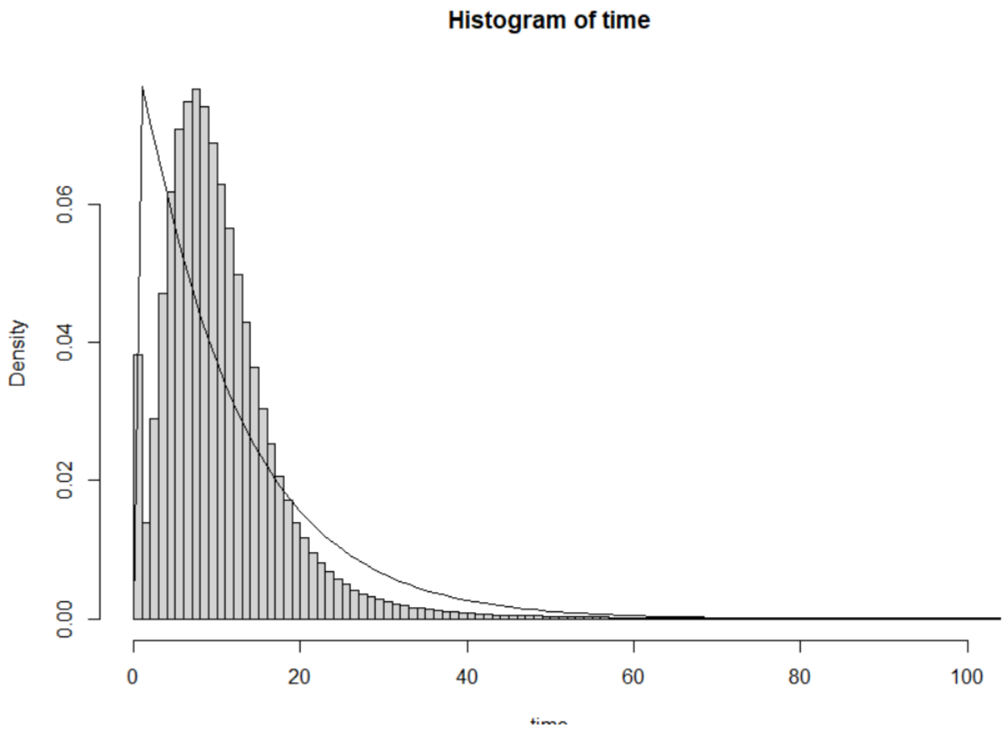
Since POLYLINE represents a set of coordinates recorded every fifteen seconds since the start of the trip, The duration of a trip can be interpreted as the number of intervals existing in the set. In terms of calculation, letting N_i donates to the number of intervals in i^{th} set, The duration of i^{th} trip in second is

$$T_i = 15(N_i - 1) \tag{1}$$

To be more general, we convert the duration into minutes by dividing it by 60.

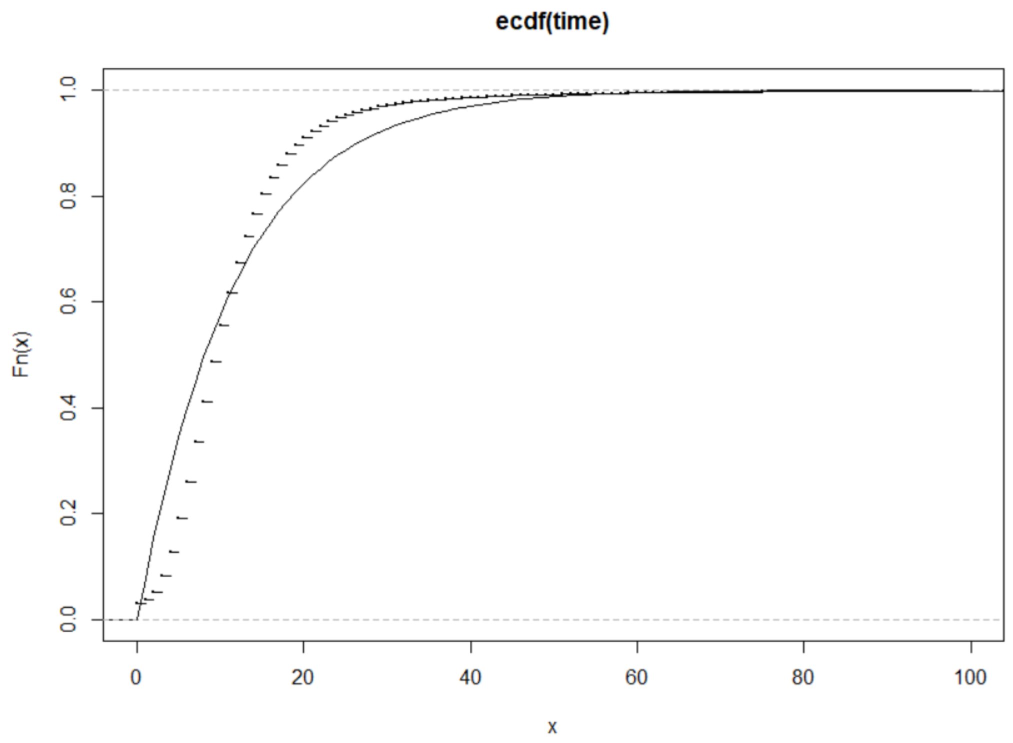
The probability family of distribution of trip duration:

Now we have a column called trip duration, so we use hist to plot the trip duration into a histogram with the x-axis as the trip duration and the y-axis as the density. The probability of trip duration is found to be a gamma distribution family in the interval [0, 970], but because most of the time is distributed within the [0,20], which makes the original histogram hard to be observed, we shrink the range of x-axis to [0, 100]. However, the gamma distribution seems to only exist in [1,100] because the probability that the trip time is under a minute is too high to fit into the gamma distribution.



Verify the hypothesis:

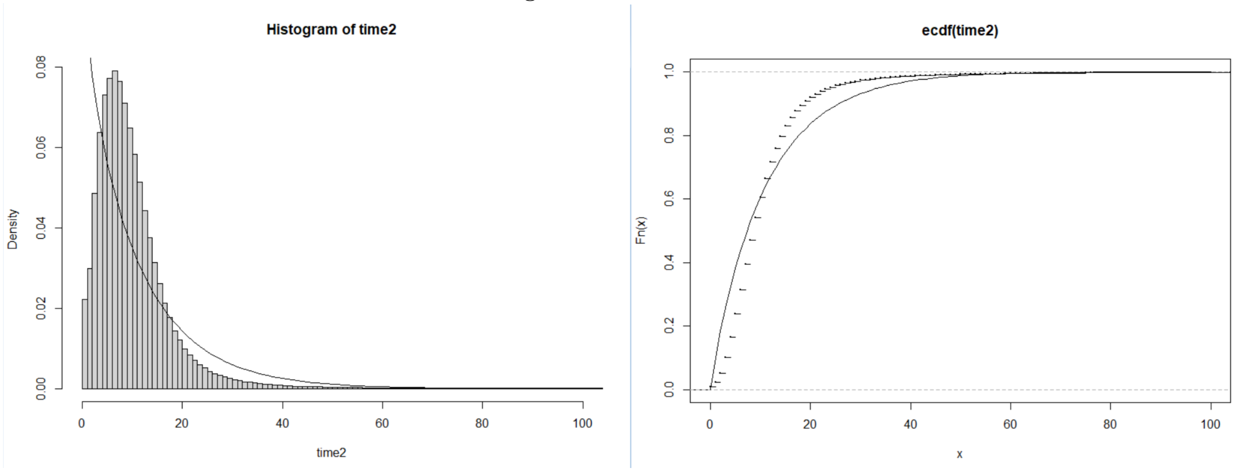
To test whether the probability density of trip duration is the gamma family of distribution, we apply the actual curve of the gamma distribution. Therefore, we need to find the r and lambda value using mean and variance, and the mean and variance of time duration are found to be approximately 11.57 and 130.13, and r and lambda are found to be 1.0294 and 0.0889. However, the gamma distribution curve does not fit the histogram so much, r value seeming to be too small.



Then, We compare the empirical cdf of T with the cdf of gamma distribution with r and lambda which we just found. However, they don't match well either.

First Correction

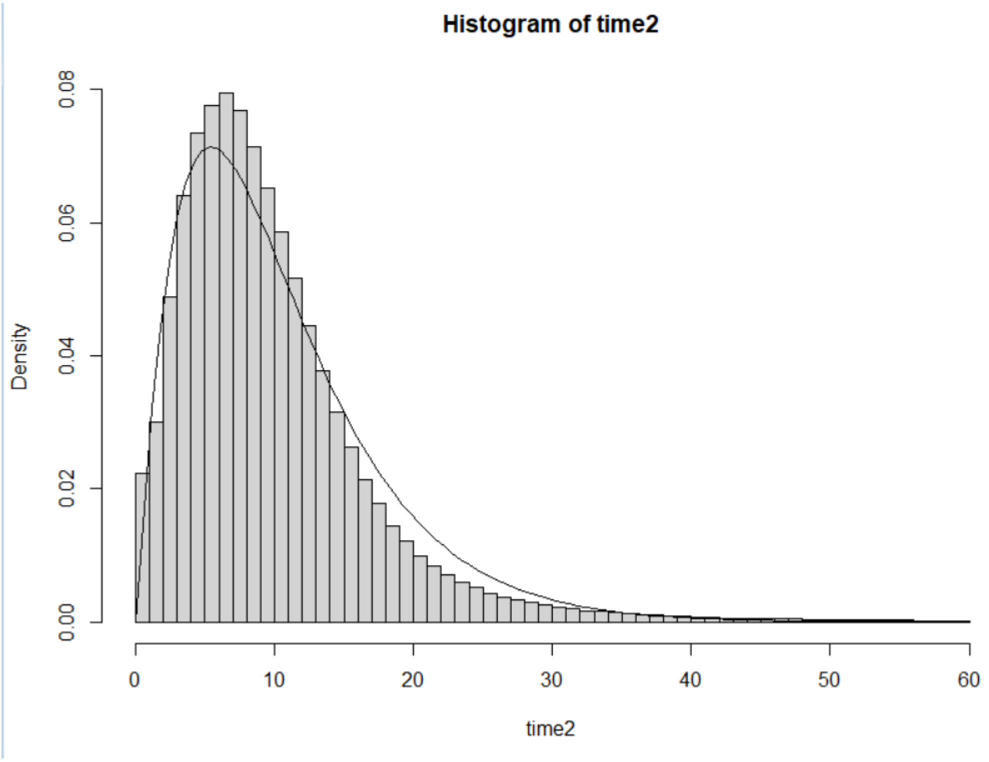
Since the proportion that the time duration is under one minute is too high, we decide to exclude it from the dataset and get a more rational look in the histogram. Then, we get the new mean, variance, r, and lambda,10.94, 129.88, 1.029, and 0.0889. The new gamma distribution curve turns out to be more inaccurate, but the slope is actually a little bit similar to the histogram's downward slope. When we compare the empirical cdf with the cdf of our new database and new gamma distribution, the difference between them is not smaller than the original difference.

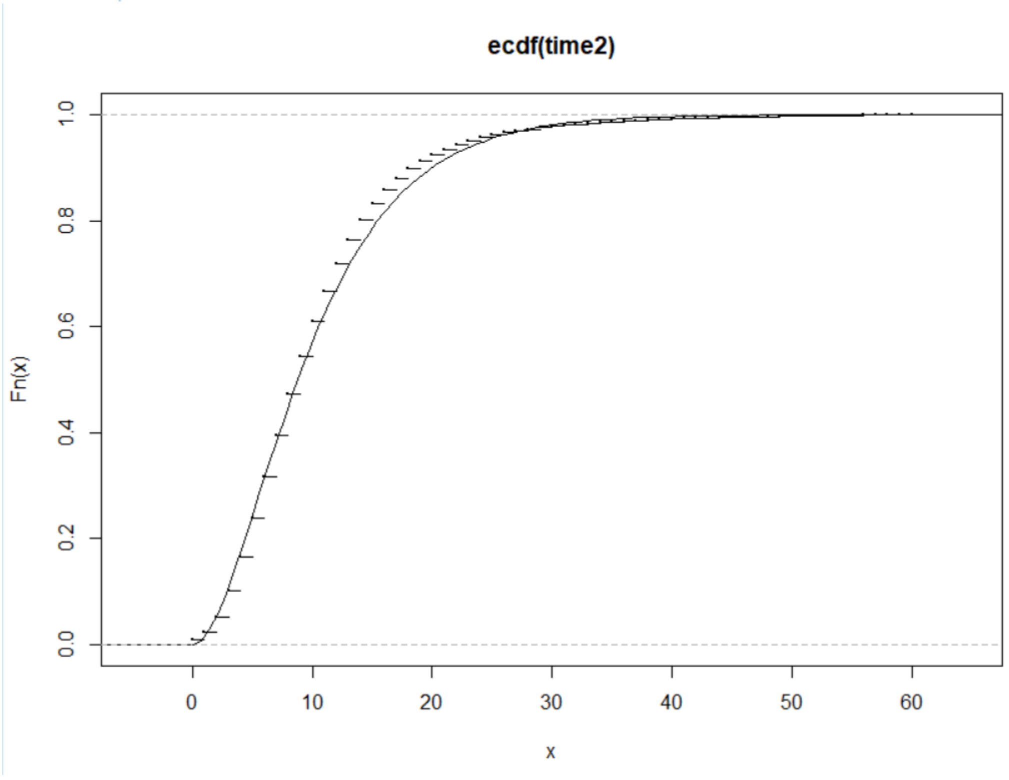


Second Correction

The unsuccess of fitting a gamma distribution curve to the histogram is due to the high value of variance, which is 130 for the distribution excluding the aberrantly high proportion, that yields a small

r. In fact, the support of T could be any value less or equal to 970, and that can lead to a very large variance with respect to the concentrated proportion [0,20]. With a large value of r, gamma distribution turns out to be steep and inaccurate; therefore. We decide to shrink the range to [1,61] by excluding every other value from the vector, and that forms a smaller mean and much smaller variance, 10.39 and 51.56, which yields a larger r and lambda, 2.09 and 0.201. Adding the curve, it fits much closer; comparing both empirical cdf and cdf of the gamma distribution, the result, as expected, comes up ideally. Therefore, we can confidently say the density of probability of trip time is the distribution of the gramma family since the proportion of interval [1,61] occupies 96.4% out of all distribution.





The trip duration shows that people averagely take about 11 minutes to their destination but most likely within 20 minutes. Because most of the probability, about 95%, is distributed in [0,25].

The extremely high proportion of trip duration being under one minute may be caused by human errors and instrument errors: drivers mistakenly or accidentally press the in-operation button, passengers' instantly calling it off, and technique issues. We observe that many POLYLINE don't have value, which explains that the coordination recorder doesn't work correctly in some cases.

Exploring the distribution of proportion of driver is busy:

Definition of Busy:

Literally, drivers being busy means they are working, but in order to calculate the proportion of drivers being busy, we also need to know when they are not busy, waiting for passengers or including they time are are off work, We assume that taxi drivers don't have specifical time to work and amount of operating time, so we decide to define the total time as ones career life recorded in the database, from the beginning of a driver's first trip to the ending time of his last trip. Since each driver has an unique id number, we can track the sum of trip duration as the total working time of i^{th} driver, denoting as U_i, and Let T_i denotes to the total time of i^{th} driver, R_{i_j} denotes to the j^{th} trip duration of the i^{th} driver, N_i denotes to the number of trips i^{th} driver have completed, and B_i denotes to the proportion of time i^{th} driver being busy; the time can be calculated as

$$T_i = TimeStamp_{ilast} - TimeStamp_{ifirst} + R_{iN_i} \tag{2}$$

The busy time is as

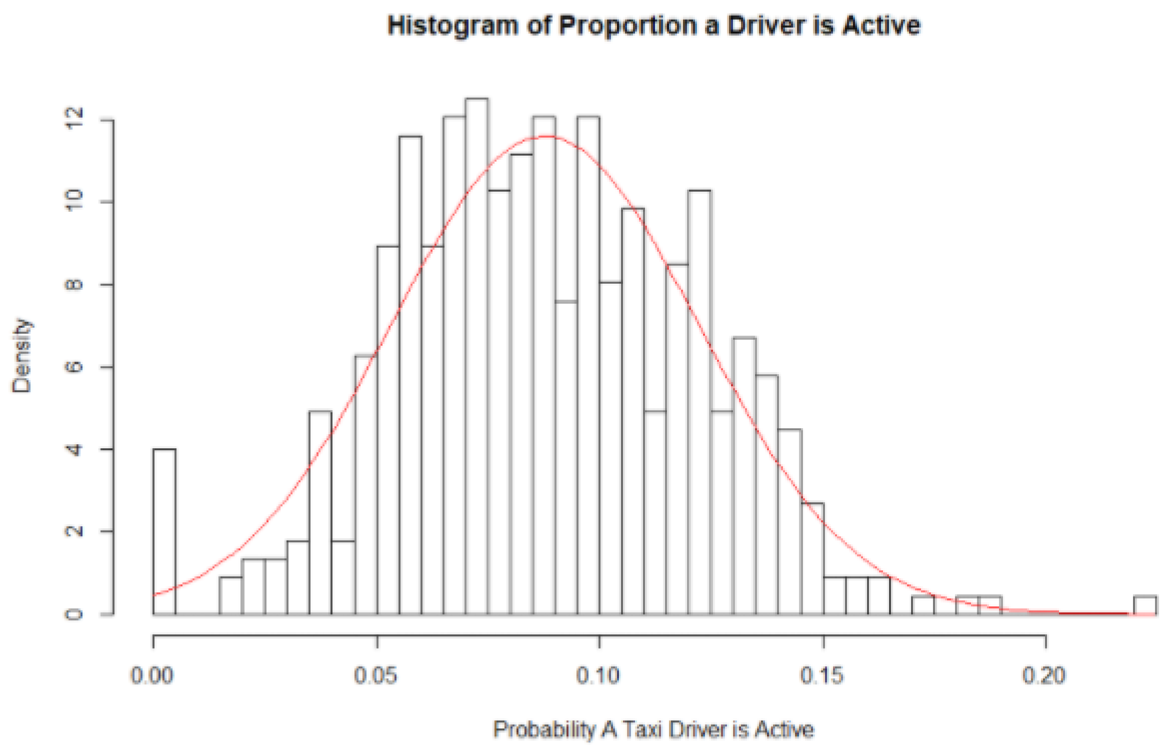
$$U_i = \sum_{k=1}^{N_i} R_{ik} \tag{3}$$

Then, the proportion is

$$B_i = \frac{U_i}{T_i} \tag{4}$$

Probability Distribution:

Plotting B into the histogram, we get an approximate normal distribution in the range of (0,0.23]. As we observe there are some extreme cases that drivers only work about 0.005 of their time involved in this career, averagely 0.12 hour a day.



Investigation of average trip times using CALLTYPE:

```
callSplit <- split(trainData, trainData$CALL_TYPE)

i <- 1
res <- list()
for (call in callSplit) {
        res[[i]] <- apply(call, 1, getDur);
        i <- i + 1
}

print(paste0('A: mean: ', mean(res[[1]]), 'var: ', var(res[[1]])))
print(paste0('B: mean: ', mean(res[[2]]), 'var: ', var(res[[2]])))
print(paste0('C: mean: ', mean(res[[3]]), 'var: ', var(res[[3]])))
```

In order to obtain the trip times we took the length of the POLYTIME and multiplied it by 15 to obtain how many seconds a trip took. This was then divided by 60 to get the time in minutes.

We filtered the original dataset to contain only the CALL_TYPE column and appended our time column to this new dataset. We then applied the mean() and var() functions to the newly converted minutes variable to obtain the mean and variance of the time.

General Dataset Mean (TIMESTAMP):

$$Mean(Overall) = \frac{Minutes(Converted)}{1710670} = 11.940 \tag{5}$$

On the other hand, to get the values with the dependency of CALLTYPE, we used filter() to sort through the data in order to get the correct TIMESTAMP values in regard to the three CALLTYPE's A,B, and C. After we used mean() to get a mean of the TIMESTAMPS sorted by CALLTYPE.

$$Mean(CALLTYPE = A) = \frac{Minutes(Converted)}{1710670} = 12.506 \tag{6}$$

$$Mean(CALLTYPE = B) = \frac{Minutes(Converted)}{1710670} = 11.086 \tag{7}$$

$$Mean(CALLTYPE = C) = \frac{Minutes(Converted)}{1710670} = 12.873 \tag{8}$$

Verification:

In order to see whether the means change with the different call types, let's check the 95% confidence interval of each mean:

$$\begin{array}{rl} Mean(CALLTYPE) & \qquad\qquad Interval \\ A: & 12.478 \leq 12.511 \leq 12.533 \\ B: & 11.069 \leq 11.056 \leq 11.103 \\ C: & 12.829 \leq 13.179 \leq 12.917 \end{array}$$

From here, we can say we are 95% confident that the true mean of trip time for the call type of A falls within the confidence interval range of A, the true mean of trip time for the call type of B falls within the confidence interval range of B, and the true mean of C falls within the confidence interval range of C.

Analysis:

Difference between trip times using different call types:

$$CallTypeA = 11.940 - 12.506 = -0.566(33.96 Seconds Slower) \tag{9}$$

$$CallTypeB = 11.940 - 11.086 = 0.854(51.24 Seconds Faster) \tag{10}$$

$$CallTypeC = 11.940 - 12.873 = -0.933(55.98 Seconds Slower) \tag{11}$$

$$CallTypeB > CallTypeA > CallTypeC \tag{12}$$

In general, the type of call made to summon a taxi doesn't make much of a difference in regards to trip time, but the fastest would be from a specific stand. Furthermore, we can examine the variance of each call type to determine the spread of the data set compared to the mean.

$$Variance(Overall) = 130.24 \tag{13}$$

$$Variance(CallTypeA) = 72.26 \tag{14}$$

$$Variance(CallTypeB) = 64.76 \tag{15}$$

$$Variance(CallTypeC) = 269.50 \tag{16}$$

The Variance of the 3 Call Types are fairly close with the exception of Call Type C. This indicates that there is a lot of volatility in the trip times for Call Type C. This could be caused by the fact that Call Type C is the least specific of the three call types, taking riders from various methods instead of a set way of getting riders. Further, there are many explanations as to why the different call types have different mean trip times. As Call Type 'C' has the greatest variance, it can be inferred that most of the longer trip times had Call Type 'C'. This is shown below:

$$\begin{aligned} Max(Overall) &= 970 \\ Max(CallTypeA) &= 580.75 \\ Max(CallTypeB) &= 958.75 \\ Max(CallTypeC) &= 970 \end{aligned}$$

Moddling the Data

Models:

In the term project we decided to use the following models to visualize our data:

- Linear
- Linear Polynomial
- k-Nearest Neighbors
- Boosting

Moddling Distance and Speed:

The first model we decided to take on was modeling the distance vs the speed. In order to plot these two variables on a model we had to obtain the distance and speed. The distance we obtained by utilizing the coordinates from POLYLINE column in the dataset. After we just took the start and end points of the formula and applied the Haversine distance formula. The Haversine (or great circle) distance is the angular distance between two points on the surface of a sphere. It is a very accurate way of computing distances between two points on the surface of a sphere using the latitude and longitude of the two points.

$$\begin{aligned} (x1, y1) &= StartingCoordinates \\ (x1, y1) &= EndingCoordinates \end{aligned}$$

In order to figure out the average speed we now have distance and can obtain time elapsed during a trip by figuring out trip duration. In order to figure out trip duration we took the length of the POLYLINE variable where each length represents 15 seconds. So, we used the following formula:

$$Speed = \frac{Distance}{Length(POLYLINE) * 15/60}$$

Moddling Time:

Looking for other parameters, we decided to check if using the day of the year and time of the day affect the accuracy of the predictions. In order to get this data, we use the timestamp and convert it to a DayTime using

```
as.POSIXct(date_ref$TIMESTAMP, origin = "1970-01-01")
```

Next, we extract the Date, Day of the year, hour, minute and second from that

```
date_ref$DATE <- format(test, format = "%Y-%m-%d")
date_ref$DAY_OF_YEAR <- as.integer(format(test, format = '%j'))
date_ref$HOUR <- as.integer(format(test, format = "%H"))
date_ref$MIN <- as.integer(format(test, format = "%M"))
date_ref$SEC <- as.integer(format(test, format = "%S"))
```

Now that we have the essential data, we can create 2 variable to use for our predictions,

```
time <- (date_ref$HOUR * 60 + date_ref$MIN + date_ref$SEC / 60)
dyear <- data.frame(date_ref$DAY_OF_YEAR + time / 24)
```

In short, the time of day was calculated by extracting the hour, minute, and second (eg. 00:00:00) from the Unix timestamp in the TIMESTAMP column of the dataset. Then, the hour was added to the minutes/60 plus the seconds /3600. Divisions were done to convert the time into hour.

Thus, in a 24 hour day, 0 is midnight and 23.99 is right before midnight of the same day. For the time of day predictions, no distinctions were made on the day of the year, only the time of day.

For the day of the year, the date (eg 01/01/2013) was extracted from the Unix timestamp and was converted into the day of the year, where 01/01/2013 is day 1 and 12/31/2013 is day 365.

Moddling Call Type:

Since we were talking about call types affecting the duration, we decided to use them in order to increase the prediction accuracy. Since there can be only A,B or C type, we can convert them to 1,2 or 3 so that we can directly feed it to the model.

```
call <- apply(trainData, 1, function(row) {
        return(as.integer(charToRaw(row[[2]]))-64)
})
```

Predictions and Parameters

In the prediction part of modeling we decided we will be predicting the time duration of a trip depending on multiple factors. The 3 different sets of parameters we decided to use:

- Predict Time Duration using Distance only
- Predict Time Duration using Speed and time of the day only
- Predict Time Duration using Distance, Speed, Time, and Type of Call

Analysis

Predict Time Duration using Distance only

We found that this gave us really good results up to an extent with the density curves somewhat going along the actual values.

Predict Time Duration using Speed and time of the day only

We found that this results in a really bad prediction since the speed and time of day alone cannot really predict the duration of the trip. We can see in the density graphs that this no way resembles the actual data.

Predict Time Duration using Distance, Speed, Time, and Type of Call

This gave us the most accurate results with the KNN model almost exactly coinciding with the actual data. We deciphered that giving the model more parameters to train on, the model can correlate more values and thus can give us better predictions.

Confidence Interval for Linear model

To find the confidence interval for the linear model, we first had to find the $q_{\alpha/2}$ values. We found them by using the qgamma function with the rate and shape same as those of the actual data and p as 0.025.

```
z = qgamma(0.025, mean(actual$dur) ^ 2 / var(actual$dur),
    mean(actual$dur) / var(actual$dur))
```

Now the next step was to find the standard error which can be found using a simple formula.

```
se = sqrt(sum((actual - prediction) ^ 2) / n)
```

Now that we have both, the z and se, we can find the intervals as $T \pm q_{\alpha/2} s.e.(T)$

Appendix

```r
library('regtools');
library('rjson');
# library('ggplot2')

"/" <- function(x, y) ifelse(y == 0, 0, base:::"/"(x, y))

readData <- function() {
  allData <- read.csv("train.csv", colClasses = c("TRIP_ID" = "character"))

  # trainData <- allData[1:n,]

  return(allData)
}

getDur <- function(row) {
  return(length(fromJSON(row[[9]])) * 15 / 60);
}

spl <- function(l) {
  df <- data.frame(l)
  l <- list()
  d <- list()

  if (nrow(df) <= 1) {
    return(c(c(getDur(df)), c(0)))
  }

  for (n in 1:(nrow(df) - 1)) {
    # print(getDur(df[n,]))
    l[[n]] <- (df[n + 1, 6] - df[n, 6]) / 60
    d[[n]] <- getDur(df[n,])
  }
  # print(d)
  return(list(d, l))
}

HaversineDistance <- function(lat1, lon1, lat2, lon2) {
  # returns the distance in m
  REarth <- 6371000
  lat <- abs(lat1 - lat2) * pi / 180
  lon <- abs(lon1 - lon2) * pi / 180
  lat1 <- lat1 * pi / 180
  lat2 <- lat2 * pi / 180
  a <- sin(lat / 2) * sin(lat / 2) + cos(lat1) * cos(lat2) * sin(lon / 2) * sin(
      lon / 2)
  d <- 2 * atan2(sqrt(a), sqrt(1 - a))
  d <- REarth * d
  return(d)
}

get_dist <- function(row) {
  lonlat <- fromJSON(row[[9]])
  snapshots <- length(lonlat)
  if (snapshots != 0) {
    start <- lonlat[[1]]
    end <- lonlat[[snapshots]]
    # # d <- sqrt((start[1] - end[1]) ^ 2 + (start[2] - end[2]) ^ 2)
    d <- HaversineDistance(start[[1]], start[[2]], end[[1]], end[[2]])
```

```
      return(d)
    }
    else return(0)
  }

  getInter <- function(x, model, actual) {
    n = nrow(actual)
    prediction = predict(model, subset(actual, select = -c(dur)))
    T = predict(model, data.frame('dist' = x))
    z = qgamma(0.025, mean(actual$dur) ^ 2 / var(actual$dur), mean(actual$dur) /
        var(actual$dur))
    # z = 1.96
    # se = sqrt(sum((actual$dur - prediction) ^ 2)) / n
    se = sqrt(sum((actual$dur - prediction) ^ 2) / n)
    # se = (1 / n + (T - mean(actual$dist)) ^ 2 / (sum((actual$dist - mean(actual$
        dist)) ^ 2)))
    inter = z * se
    return(c(T - inter, T + inter))
  }

  partB <- function() {
    p2 <- subset(copy, select = -c(TRIP_ID, CALL_TYPE, ORIGIN_CALL, ORIGIN_STAND,
        DAY_TYPE, MISSING_DATA, POLYLINE))
    p2$time <- time
    p2 <- p2[order(p2$TIMESTAMP),]

    #convert hour format to minutes to use in ml
    pnext <- subset(p2, select = c(TAXI_ID, TIMESTAMP, time))

    taxi_id <- split(pnext, pnext$TAXI_ID)

    busy <- vector(length = 448)
    not_busy <- vector(length = 448)

    as.POSIXct(end, origin = "1970-01-01")

    difftime((as.POSIXct(end, origin = "1970-01-01")), (as.POSIXct(start, origin =
         "1970-01-01")), units = 'mins')
    start = p2$TIMESTAMP[1]
    end = p2$TIMESTAMP[1000000]

    #448 rows
    for (i in 1:488) {
      tot_busy <- 0
      tot_notbusy <- 0
      last <- 0
      for (j in taxi_id[i]) {
        rows <- nrow(j)
        for (k in 1:rows) {
          current <- j[k,]
          if (k == 1) {
            start <- current$TIMESTAMP
          }
          else if (k == rows) {
            end <- current$TIMESTAMP
            last <- current$time
          }
          tot_busy <- tot_busy + current$time
        }
      }
      busy[i] <- tot_busy
      not_busy[i] <- as.numeric(difftime(as.POSIXct(end, origin = "1970-01-01"),
         as.POSIXct(start, origin = "1970-01-01"), units = 'mins')) + last
    }

    busy
    not_busy
    prop <- busy / (not_busy)
    prop_dat <- data.frame(prop)
    prop_dat[is.na(prop_dat)] = 0

    maximum_prop = max(prop_dat$prop, na.rm = TRUE)
    mean_prop <- mean(prop_dat[["prop"]])
    sd_prop <- sd(prop_dat[["prop"]])

    mean_prop
    sd_prop

    prop_dat[is.na(prop_dat)] = 0

    hist(prop_dat$prop, main = "Histogram of Proportion a Driver is Active", freq
       = FALSE, breaks = 80, xlab = 'Probability A Taxi Driver is Active')
    curve(dnorm(x, mean = mean_prop, sd = sd_prop), 0, 0.224220729583627, add =
       TRUE, col = "red")

    plot(ecdf(prop_dat$prop), main = 'Comparison of ecdf with Predicted cdf', xlab
        = 'Proportion of active time', ylab = 'F(x)', xlim = c(0, 1))
    curve(pnorm(x, mean_prop, sd_prop), 0, 1, add = TRUE, col = 'red')
    legend(0.8, 0.2, legend = c("Actual", "pnorm"), col = c("black", "red"), lty =
        1:1, cex = 0.8)
  }

  partAB <- function() {
    train_split <- split(trainData, trainData$TAXI_ID)

    r <- lapply(train_split, spl)
    d <- list()
    w <- list()
    for (i in r) {
      d <- c(d, i[[1]])
      w <- c(w, i[[2]])
    }
    d <- unlist(d)
    w <- unlist(w)

    m1 <- mean(d)
    v1 <- var(d)
    r1 <- m1 / v1
    s1 <- m1 * m1 / v1

    h1 <- hist(d, freq = FALSE, breaks = 200, xlim = c(0, 200))
    curve(dgamma(x, s1, r1), 0, max(d), add = TRUE, col = "red")
    curve(dgamma(x, 4, 0.4), 0, max(d), add = TRUE, col = "red")

    p <- d / w

    m2 <- mean(p)
    v2 <- var(p)
    r2 <- m2 / v2
    s2 <- m2 * m2 / v2
    h2 <- hist(p, freq = FALSE, breaks = 20000, xlim = c(0, 1))
    print(h2)
    curve(dgamma(x, s2, 1 / r2), 0, max(p), add = TRUE, col = "red")

    # partB()
  }

  partC <- function() {
    callSplit <- split(trainData, trainData$CALL_TYPE)

    i <- 1
    res <- list()
    for (call in callSplit) {
      res[[i]] <- apply(call, 1, getDur);
      # print(res)
      i <- i + 1
    }

    print(paste0('A: mean: ', mean(res[[1]]), 'var: ', var(res[[1]])))
    print(paste0('B: mean: ', mean(res[[2]]), 'var: ', var(res[[2]])))
    print(paste0('C: mean: ', mean(res[[3]]), 'var: ', var(res[[3]])))
  }

  drawPlot <- function(m1, m2) {
    lin1 <- qeLin(m1, yName = "dur")
    lin2 <- qeLin(m2, yName = "dur")
    poly1 <- qePolyLin(m1, yName = "dur")
    poly2 <- qePolyLin(m2, yName = "dur")
    knn1 <- qeKNN(m1, yName = "dur", 100)
    knn2 <- qeKNN(m2, yName = "dur", 100)

    pre1 <- predict(poly1, subset(m1, select = -c(dur)))
    pre2 <- predict(poly2, subset(m2, select = -c(dur)))
    pre3 <- predict(knn1, subset(m1, select = -c(dur)))
    pre4 <- predict(knn2, subset(m2, select = -c(dur)))
    pre5 <- predict(lin1, subset(m1, select = -c(dur)))
    pre6 <- predict(lin2, subset(m2, select = -c(dur)))

    par(mfrow = c(1, 2))

    plot(density(m1$dur), xlim = c(0, 30))
    lines(density(pre1), col = "red")
    lines(density(pre2), col = "green")
    lines(density(pre3), col = "cyan")
    lines(density(pre4), col = "purple")
    lines(density(pre5), col = "orange")
    lines(density(pre6), col = "yellow")

    legend(20, 0.08, legend = c("Actual", "Lin1", "Lin2", "PolyLin1", "PolyLin2",
        "KNN1", "KNN2"),
         col = c("black", "orange", "Yellow", "red", "green", "cyan", "purple"),
           lty = 1, cex = 0.8)

    plot(m1$dist, m1$dur, xlim = c(0, 10 ^ 5))
    points(m1$dist, pre1, col = "red", cex = 0.1)
    points(m1$dist, pre4, col = "purple", cex = 0.1)

  }
```

```
partD <- function() {
  date_ref <- trainData
  date_ref <- subset(date_ref, select = c(TIMESTAMP))
  test = as.POSIXct(date_ref$TIMESTAMP, origin = "1970-01-01")
  date_ref$DATE <- format(test, format = "%Y-%m-%d")
  date_ref$DAY_OF_YEAR <- as.integer(format(test, format = '%j'))
  date_ref$HOUR <- as.integer(format(test, format = "%H"))
  date_ref$MIN <- as.integer(format(test, format = "%M"))
  date_ref$SEC <- as.integer(format(test, format = "%S"))
  date_ref <- subset(date_ref, select = -c(TIMESTAMP))

  distance <- apply(trainData, 1, get_dist)
  duration <- apply(trainData, 1, getDur)
  speed <- dist / dur
  call <- apply(trainData, 1, function(row) {
    return(as.integer(charToRaw(row[[2]])) - 64)
  })
  #convert hour format to minutes to use in ml
  time <- (date_ref$HOUR * 60 + date_ref$MIN + date_ref$SEC / 60)
  dyear <- data.frame(date_ref$DAY_OF_YEAR + time / 24)

  m1 <- data.frame('dist' = distance, 'dur' = duration)
  m2 <- data.frame('dyear' = dyear, 'speed' = speed, 'dur' = duration)
  m3 <- data.frame('dist' = distance, 'speed' = speed, 'time' = time, 'call' =
      call, 'dur' = duration)

  drawPlot(m1, m3)
}

trainData <- readData()

partAB()
partC()
partD()
```