About the project

Case Study 6.1: NYC Taxi Trips

Case Study Description: To predict the trip duration of a New York taxi cab ride, we can build different types of features and evaluate them. We will start by describing what is a feature in this context; then we will develop some elementary features and add features using the software package featuretools. We will assess how these features perform in predicting trip duration.

Setup

Libraries

Note:
This uses an alternative to **featuretools** for R called **featuretoolsR**

- To Install featuretoolsR tou need to have "devtools" installed.
- Then you can run: devtools::install_github("magnusfurugard/featuretoolsR")

In [1]:

```
# You wil need to define (if not already) your Python path.
# reticulate::py_discover_config()
library("reticulate")
use_python("~/Library/r-miniconda/envs/r-reticulate/bin/python")
```

In [2]:

```
library("featuretoolsR")
library("dplyr")
library("gbm")
print("Functions loaded")
```

```
Registered S3 methods overwritten by 'ggplot2':
  method         from
  [.quosures     rlang
  c.quosures     rlang
  print.quosures rlang
```

```
╔══════════════════════╗
║   featuretoolsR 0.4.4 ║
╚══════════════════════╝
```

```
✔ Using Featuretools 0.16.0
```

```
Attaching package: 'dplyr'


The following objects are masked from 'package:stats':

    filter, lag


The following objects are masked from 'package:base':

    intersect, setdiff, setequal, union


Loaded gbm 2.1.5
```

```
[1] "Functions loaded"
```

Functions

- **featuretoolsR** does not include the original **encode_features** function, so a custom one was made called **custom_encode_features** which returns one-hot encoding of categorical variables.
- Additionally, custom function to replace python **SimpleImputer** class and Rscores were made.

In [1]:

```
# R^2 Calculation
r_squared <- function(predicted,actual) {
  r_squared <- 1 - (sum((actual - predicted)^2) / sum((actual - mean(actual))^2))
  return(r_squared)
}

# Custom encode feature to support missing function un featuretoolsR
  custom_encode_features <- function(feature_matrix, to_encode, include_unknown, top_n) {
    temp <- feature_matrix
    for(feature in to_encode) {

      # We generate the list of top_n unique values for feature to encode
      if(include_unknown == TRUE) {
        encoded_feature <- names(head(sort(table(feature_matrix[[feature]],exclude = NULL),decreasi
ng=TRUE),top_n))
      } else {
        encoded_feature <- names(head(sort(table(feature_matrix[[feature]]),decreasing=TRUE),top_n
))
      }

      for(i in 1:length(encoded_feature)) {
        temp <- temp %>%
          mutate(
            !!paste(feature,"=",encoded_feature[i]) := ifelse(feature_matrix[[feature]] == encoded_
feature[i] , 1, 0)
          )
      }
    }
    return(temp[,!names(temp) %in% to_encode])
  }

# Compute features function based on dsx-cs6 utils
  compute_features <- function(features,entities) {

    # We create feature matrix using featuretools calculate_feature_matrix function
    feature_matrix <- calculate_feature_matrix(entities,features,
                                               approximate='36d',
                                               verbose=TRUE)

    # Since encode_features function is missing, we hot-encode pickup_neighborhood and
dropoff_neighborhood
    # with a custom function created: custom_encode_features
    print("Finishing computing...")
    feature_matrix <- custom_encode_features(feature_matrix, to_encode=c("pickup_neighborhood","dro
poff_neighborhood"), include_unknown=FALSE, top_n=1000)

    # Return Output
    return(feature_matrix)

  }

# Logical to integer
logical_to_integer <- function(feature_matrix) {
  # Convert logical values to integers
  logical_features <- names(feature_matrix[,unlist(lapply(feature_matrix, is.logical))])
  for(feature in logical_features) {
    feature_matrix[[feature]] <- as.numeric(as.integer(feature_matrix[[feature]]))
  }
  return(feature_matrix)
```

```
}

logical_to_factor <- function(feature_matrix) {
  # Convert logical values to integers
  logical_features <- names(feature_matrix[,unlist(lapply(feature_matrix, is.logical))])
  for(feature in logical_features) {
    feature_matrix[[feature]] <- as.factor(feature_matrix[[feature]])
  }
  return(feature_matrix)
}

custom_fit <- function(data) {
  out <- list()
  numeric_features <- names(data[,unlist(lapply(data, is.numeric))])
  for(feature in numeric_features) {
    out[[feature]] <- mean(data[[feature]] ,na.rm = TRUE)
  }
  return(out)
}

# Custom fit_transform
custom_transform <- function(data,fit) {
  numeric_features <- names(fit)
  for(feature in numeric_features) {
    data[[feature]][which(is.na(data[[feature]]))] <-fit[[feature]]
  }
  return(data)
}

# get_train_test_fm based on dsx-cs6 utils
get_train_test_fm <- function(feature_matrix, original_data, percentage) {

  out <- list()

  nrows <- nrow(feature_matrix)
  head <- nrows * percentage
  tail <- nrows - head

  X_train <- head(feature_matrix,head)
  y_train <- head(original_data$trip_duration,head)

  # Emulating fit_transform
  fit <- custom_fit(X_train)
  X_train = custom_transform(X_train,fit)

  X_test = tail(feature_matrix,tail)
  y_test = tail(original_data$trip_duration,tail)
  X_test = custom_transform(X_test,fit)

  # Return values
  out$X_train <- X_train
  out$y_train <- y_train
  out$X_test <- X_test
  out$y_test <- y_test

  return(out)

}
print("Functions Loaded")
```

```
[1] "Functions Loaded"
```

Data Load

In [4]:

```
# Needed to load picke file
source_python("read_pickle.py")
```

- Opening a pickle file in R additional info

In [5]:

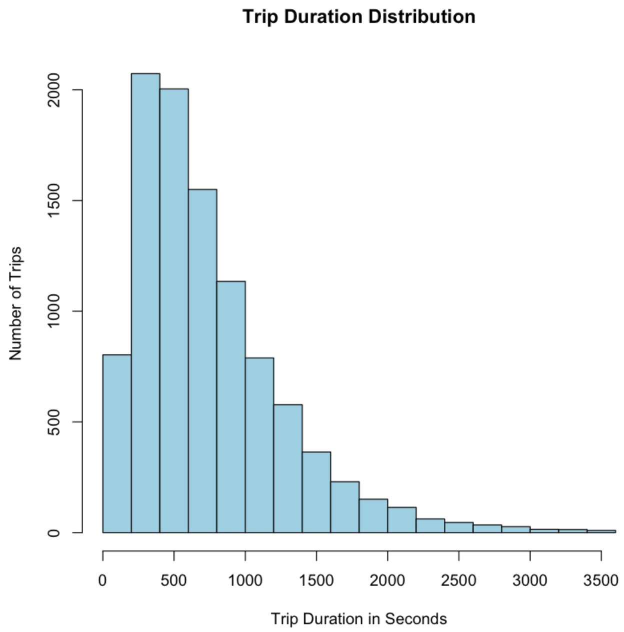
```
# We load data
trips <- read_pickle_file("./trips.pkl")
dropoff_neighborhoods <- read.csv("./dropoff_neighborhoods.csv")
pickup_neighborhoods <- read.csv("./pickup_neighborhoods.csv")

# We set type
trips$pickup_neighborhood <- as.character(trips$pickup_neighborhood)
trips$dropoff_neighborhood <- as.character(as.character(trips$dropoff_neighborhood))
dropoff_neighborhoods$neighborhood_id <- as.character(dropoff_neighborhoods$neighborhood_id)
pickup_neighborhoods$neighborhood_id <- as.character(pickup_neighborhoods$neighborhood_id)
print("Data Loaded")
```

```
[1] "Data Loaded"
```

In [6]:

```
# Histogram
hist(trips$trip_duration, main="Trip Duration Distribution",
  xlab="Trip Duration in Seconds",
  ylab="Number of Trips",
  col="lightblue")

# Tells us how many trips are in the dataset
paste("Trips in dataset:",nrow(trips))
```

'Trips in dataset: 10000'



Entities and Relationships

In [7]:

```
# Create entityset
entities <- as_entityset(
  trips,
  index = "id",
  entity_id = "trips",
  id = "trips",
  time_index = "pickup_datetime"
) %>%
add_entity(
  pickup_neighborhoods,
  entity_id = "pickup_neighborhoods",
  index = "neighborhood_id"
```

```r
) %>%
add_entity(
  dropoff_neighborhoods,
  entity_id = "dropoff_neighborhoods",
  index = "neighborhood_id"
)

# Add index relationships
entities <- entities %>%
add_relationship(
  parent_set = "pickup_neighborhoods",
  child_set = "trips",
  parent_idx = "neighborhood_id",
  child_idx = "pickup_neighborhood"
)  %>%
  add_relationship(
    parent_set = "dropoff_neighborhoods",
    child_set = "trips",
    parent_idx = "neighborhood_id",
    child_idx = "dropoff_neighborhood"
  )
print("Entities and Relationships generated")
```

```
[1] "Entities and Relationships generated"
```

1.1 First Model

Transform Primitives

In [8]:

```r
trans_primitives <- c("is_weekend")
features <- dfs(
  entityset=entities,
  target_entity="trips",
  trans_primitives=trans_primitives,
  agg_primitives=NULL,
  ignore_variables= list(trips=c("pickup_latitude", "pickup_longitude",
                                 "dropoff_latitude", "dropoff_longitude", "trip_duration")),
  features_only=TRUE
)
print(paste("Number of features:",length(features)))
```

```
[1] "Number of features: 12"
```

Dataset Creation

In [9]:

```r
feature_matrix <- compute_features(features, entities)
head(feature_matrix,5)
```

```
[1] "Finishing computing..."
```

A data.frame: 5 × 108

	vendor_id	passenger_count	trip_distance	payment_type	IS_WEEKEND(dropoff_datetime)	IS_WEEKEND(pickup_datetime)	pickup_ne
	<dbl>	<dbl>	<dbl>	<dbl>	<lgl>	<lgl>	
1	2	1	2.46	1	TRUE	TRUE	
2	1	2	7.90	1	TRUE	TRUE	
3	1	1	1.00	1	TRUE	TRUE	
4	2	1	0.02	2	TRUE	TRUE	

5	vendor_id	1	passenger_count	2	trip_distance	19.00	payment_type	1	IS_WEEKEND(dropoff_datetime)	TRUE	IS_WEEKEND(pickup_datetime)	TRUE	pickup_ne	

Split dataset

In [10]:

```
train_test = get_train_test_fm(feature_matrix, trips, .75)

y_train = log(train_test$y_train + 1)
y_test = log(train_test$y_test + 1)
X_train <- logical_to_integer(train_test$X_train)
X_test <- logical_to_integer(train_test$X_test)

trainingDataset = cbind(y_train,X_train)
print('Data split successful!')
```

```
[1] "Data split successful!"
```

Model 1.1: Fitting Generalized Boosted Regression Model

In [11]:

```
gbm <- gbm(y_train  ~ ., data = trainingDataset,verbose = TRUE,
           n.trees = 100,
           distribution="gaussian",
           interaction.depth = 3,
           n.minobsinnode = 1,
           shrinkage = 0.1)
```

```
Warning message in gbm.fit(x = x, y = y, offset = offset, distribution = distribution, :
“variable 5: `IS_WEEKEND(dropoff_datetime)` has no variation.”
Warning message in gbm.fit(x = x, y = y, offset = offset, distribution = distribution, :
“variable 6: `IS_WEEKEND(pickup_datetime)` has no variation.”
```

```
Iter   TrainDeviance   ValidDeviance   StepSize   Improve
     1        0.4814             nan     0.1000    0.0643
     2        0.4293             nan     0.1000    0.0509
     3        0.3847             nan     0.1000    0.0443
     4        0.3480             nan     0.1000    0.0367
     5        0.3179             nan     0.1000    0.0310
     6        0.2921             nan     0.1000    0.0253
     7        0.2710             nan     0.1000    0.0208
     8        0.2526             nan     0.1000    0.0193
     9        0.2379             nan     0.1000    0.0153
    10        0.2255             nan     0.1000    0.0124
    20        0.1603             nan     0.1000    0.0029
    40        0.1348             nan     0.1000    0.0004
    60        0.1278             nan     0.1000    0.0002
    80        0.1243             nan     0.1000    0.0001
   100        0.1211             nan     0.1000    0.0000
```

Model 1.1: Results from predictions

R^2

In [12]:

```
predicted_gbm <- predict(gbm,X_test,n.trees = 100)
paste("R Squared:",round(r_squared(y_test,predicted_gbm),4))
```

'R Squared: 0.6776'

Model 1.1: Feature influence

In [13]:

```
head(round(relative.influence(gbm,sort= TRUE)/sum(relative.influence(gbm)),4),25)
```

```
n.trees not given. Using 100 trees.
n.trees not given. Using 100 trees.
```

trip_distance: 0.9102 **dropoff_neighborhoods.longitude:** 0.0336 **dropoff_neighborhoods.latitude:** 0.0213 **pickup_neighborhoods.longitude:** 0.0078 **pickup_neighborhoods.latitude:** 0.0049 **`dropoff_neighborhood = AA`:** 0.003 **payment_type:** 0.0022 **`pickup_neighborhood = AR`:** 0.0016 **`dropoff_neighborhood = AU`:** 0.0014 **`pickup_neighborhood = AU`:** 0.0012 **`dropoff_neighborhood = AO`:** 0.0012 **vendor_id:** 0.0011 **`dropoff_neighborhood = AC`:** 0.0011 **`dropoff_neighborhood = H`:** 9e-04 **`pickup_neighborhood = D`:** 9e-04 **`pickup_neighborhood = AB`:** 9e-04 **`pickup_neighborhood = AG`:** 8e-04 **`pickup_neighborhood = X`:** 7e-04 **`dropoff_neighborhood = AR`:** 6e-04 **`pickup_neighborhood = AN`:** 6e-04 **`pickup_neighborhood = S`:** 5e-04 **passenger_count:** 4e-04 **`dropoff_neighborhood = AP`:** 4e-04 **`dropoff_neighborhood = J`:** 4e-04 **`dropoff_neighborhood = E`:** 4e-04

1.2 Second Model

Transform Primitives

In [14]:

```
trans_primitives <- c("minute","day","week","month","weekday","is_weekend")
features <- dfs(
  entityset=entities,
  target_entity="trips",
  trans_primitives=trans_primitives,
  agg_primitives=NULL,
  ignore_variables= list(trips=c("pickup_latitude", "pickup_longitude",
                                  "dropoff_latitude", "dropoff_longitude", "trip_duration")),
  features_only=TRUE
)
print(paste("Number of features:",length(features)))
```

```
[1] "Number of features: 22"
```

Dataset Creation

In [15]:

```
feature_matrix <- compute_features(features, entities)
head(feature_matrix,5)
```

```
[1] "Finishing computing..."
```

A data.frame: 5 × 118

	vendor_id	passenger_count	trip_distance	payment_type	MINUTE(dropoff_datetime)	MINUTE(pickup_datetime)	DAY(dropoff_datetim
	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl
1	2	1	2.46	1	17	0	
2	1	2	7.90	1	24	0	
3	1	1	1.00	1	19	0	
4	2	1	0.02	2	1	0	

5	vendor_id	passenger_count	trip_distance	payment_type	MINUTE(dropoff_datetime)	MINUTE(pickup_datetime)	DAY(dropoff_datetime
	1	2	19.00	1	58	1	

Split dataset

In [16]:

```
train_test = get_train_test_fm(feature_matrix, trips, .75)

y_train = log(train_test$y_train + 1)
y_test = log(train_test$y_test + 1)
X_train <- logical_to_integer(train_test$X_train)
X_test <- logical_to_integer(train_test$X_test)

trainingDataset = cbind(y_train,X_train)
print('Data split successful!')
```

```
[1] "Data split successful!"
```

Model 1.2: Fitting Generalized Boosted Regression Model

In [17]:

```
gbm <- gbm(y_train  ~ ., data = trainingDataset,verbose = TRUE,
           n.trees = 100,
           distribution="gaussian",
           interaction.depth = 3,
           n.minobsinnode = 1,
           shrinkage = 0.1)
```

```
Warning message in gbm.fit(x = x, y = y, offset = offset, distribution = distribution, :
“variable 9: `WEEK(dropoff_datetime)` has no variation.”
Warning message in gbm.fit(x = x, y = y, offset = offset, distribution = distribution, :
“variable 10: `WEEK(pickup_datetime)` has no variation.”
Warning message in gbm.fit(x = x, y = y, offset = offset, distribution = distribution, :
“variable 11: `MONTH(dropoff_datetime)` has no variation.”
Warning message in gbm.fit(x = x, y = y, offset = offset, distribution = distribution, :
“variable 12: `MONTH(pickup_datetime)` has no variation.”
Warning message in gbm.fit(x = x, y = y, offset = offset, distribution = distribution, :
“variable 15: `IS_WEEKEND(dropoff_datetime)` has no variation.”
Warning message in gbm.fit(x = x, y = y, offset = offset, distribution = distribution, :
“variable 16: `IS_WEEKEND(pickup_datetime)` has no variation.”
```

```
Iter   TrainDeviance   ValidDeviance   StepSize   Improve
     1        0.4812             nan     0.1000    0.0629
     2        0.4299             nan     0.1000    0.0522
     3        0.3862             nan     0.1000    0.0441
     4        0.3509             nan     0.1000    0.0358
     5        0.3208             nan     0.1000    0.0315
     6        0.2956             nan     0.1000    0.0246
     7        0.2743             nan     0.1000    0.0218
     8        0.2557             nan     0.1000    0.0186
     9        0.2408             nan     0.1000    0.0150
    10        0.2270             nan     0.1000    0.0137
    20        0.1623             nan     0.1000    0.0039
    40        0.1346             nan     0.1000    0.0005
    60        0.1279             nan     0.1000    0.0003
    80        0.1236             nan     0.1000   -0.0002
   100        0.1210             nan     0.1000    0.0000
```

Model 1.2: Results from predictions

R^2

In [18]:

```
predicted_gbm <- predict(gbm,X_test,n.trees = 100)
paste("R Squared:",round(r_squared(y_test,predicted_gbm),4))
```

'R Squared: 0.7085'

Model 1.2: Feature influence

In [19]:

```
head(round(relative.influence(gbm,sort= TRUE)/sum(relative.influence(gbm)),4),25)
```

```
n.trees not given. Using 100 trees.
n.trees not given. Using 100 trees.
```

trip_distance: 0.9057 **dropoff_neighborhoods.longitude:** 0.0358 **dropoff_neighborhoods.latitude:** 0.02 **pickup_neighborhoods.longitude:** 0.0072 **pickup_neighborhoods.latitude:** 0.0048 **payment_type:** 0.0025 **`dropoff_neighborhood = AA`:** 0.0024 **`DAY(pickup_datetime)`:** 0.0022 **`pickup_neighborhood = X`:** 0.0019 **`dropoff_neighborhood = AU`:** 0.0018 **`dropoff_neighborhood = AC`:** 0.0014 **`DAY(dropoff_datetime)`:** 0.0012 **`dropoff_neighborhood = E`:** 0.0011 **`pickup_neighborhood = AN`:** 9e-04 **`pickup_neighborhood = AR`:** 9e-04 **`MINUTE(dropoff_datetime)`:** 7e-04 **`pickup_neighborhood = AB`:** 7e-04 **`pickup_neighborhood = S`:** 7e-04 **`dropoff_neighborhood = D`:** 6e-04 **`dropoff_neighborhood = H`:** 6e-04 **`pickup_neighborhood = AG`:** 5e-04 **`pickup_neighborhood = AP`:** 5e-04 **`dropoff_neighborhood = AP`:** 4e-04 **`dropoff_neighborhood = J`:** 4e-04 **`MINUTE(pickup_datetime)`:** 4e-04

1.3 Third Model

Transform Primitives

In [20]:

```
trans_primitives <- c("minute","day","week","month","weekday","is_weekend")
aggregation_primitives = c("count","sum","mean","median","std","max","min")
features <- dfs(
  entityset=entities,
  target_entity="trips",
  trans_primitives=trans_primitives,
  agg_primitives=aggregation_primitives,
  ignore_variables= list(trips=c("pickup_latitude", "pickup_longitude",
                                  "dropoff_latitude", "dropoff_longitude", "trip_duration")),
  features_only=TRUE
)
print(paste("Number of features:",length(features)))
```

```
[1] "Number of features: 72"
```

Dataset Creation

In [21]:

```
feature_matrix <- compute_features(features, entities)
head(feature_matrix,5)
```

```
[1] "Finishing computing..."
```

A data.frame: 5 × 168

vendor_id passenger_count trip_distance payment_type MINUTE(dropoff_datetime) MINUTE(pickup_datetime) DAY(dropoff_datetim

	vendor_id	passenger_count	trip_distance	payment_type	MINUTE(dropoff_datetime)	MINUTE(pickup_datetime)	DAY(dropoff_datetime
	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl
1	2	1	2.46	1	17	0	
2	1	2	7.90	1	24	0	
3	1	1	1.00	1	19	0	
4	2	1	0.02	2	1	0	
5	1	2	19.00	1	58	1	

Split dataset

In [22]:

```
train_test = get_train_test_fm(feature_matrix, trips, .75)

y_train = log(train_test$y_train + 1)
y_test = log(train_test$y_test + 1)
X_train <- logical_to_integer(train_test$X_train)
X_test <- logical_to_integer(train_test$X_test)

trainingDataset = cbind(y_train,X_train)
print('Data split successful!')
```

```
[1] "Data split successful!"
```

Model 1.3: Fitting Generalized Boosted Regression Model

In [23]:

```
gbm <- gbm(y_train  ~ ., data = trainingDataset,verbose = TRUE,
           n.trees = 100,
           distribution="gaussian",
           interaction.depth = 3,
           n.minobsinnode = 1,
           shrinkage = 0.1)
```

```
Warning message in gbm.fit(x = x, y = y, offset = offset, distribution = distribution, :
“variable 9: `WEEK(dropoff_datetime)` has no variation.”
Warning message in gbm.fit(x = x, y = y, offset = offset, distribution = distribution, :
“variable 10: `WEEK(pickup_datetime)` has no variation.”
Warning message in gbm.fit(x = x, y = y, offset = offset, distribution = distribution, :
“variable 11: `MONTH(dropoff_datetime)` has no variation.”
Warning message in gbm.fit(x = x, y = y, offset = offset, distribution = distribution, :
“variable 12: `MONTH(pickup_datetime)` has no variation.”
Warning message in gbm.fit(x = x, y = y, offset = offset, distribution = distribution, :
“variable 15: `IS_WEEKEND(dropoff_datetime)` has no variation.”
Warning message in gbm.fit(x = x, y = y, offset = offset, distribution = distribution, :
“variable 16: `IS_WEEKEND(pickup_datetime)` has no variation.”
Warning message in gbm.fit(x = x, y = y, offset = offset, distribution = distribution, :
“variable 55: `dropoff_neighborhoods.MEDIAN(trips.passenger_count)` has no variation.”
Warning message in gbm.fit(x = x, y = y, offset = offset, distribution = distribution, :
“variable 65: `dropoff_neighborhoods.MAX(trips.vendor_id)` has no variation.”
Warning message in gbm.fit(x = x, y = y, offset = offset, distribution = distribution, :
“variable 67: `dropoff_neighborhoods.MIN(trips.passenger_count)` has no variation.”
Warning message in gbm.fit(x = x, y = y, offset = offset, distribution = distribution, :
“variable 69: `dropoff_neighborhoods.MIN(trips.vendor_id)` has no variation.”
Warning message in gbm.fit(x = x, y = y, offset = offset, distribution = distribution, :
“variable 70: `dropoff_neighborhoods.MIN(trips.payment_type)` has no variation.”
```

```
Iter   TrainDeviance   ValidDeviance   StepSize   Improve
     1        0.4813             nan     0.1000    0.0646
     2        0.4281             nan     0.1000    0.0519
     3        0.3847             nan     0.1000    0.0430
     4        0.3482             nan     0.1000    0.0363
     5        0.3164             nan     0.1000    0.0296
     6        0.2912             nan     0.1000    0.0249
```

```
      7        0.2704             nan     0.1000    0.0211
      8        0.2529             nan     0.1000    0.0174
      9        0.2370             nan     0.1000    0.0137
     10        0.2249             nan     0.1000    0.0120
     20        0.1621             nan     0.1000    0.0028
     40        0.1345             nan     0.1000    0.0003
     60        0.1266             nan     0.1000    0.0001
     80        0.1214             nan     0.1000    0.0002
    100        0.1187             nan     0.1000   -0.0001
```

Model 1.3: Results from predictions

R^2

In [24]:

```
predicted_gbm <- predict(gbm,X_test,n.trees = 100)
paste("R Squared:",round(r_squared(y_test,predicted_gbm),4))
```

'R Squared: 0.7137'

Model 1.3: Feature influence

In [25]:

```
head(round(relative.influence(gbm,sort= TRUE)/sum(relative.influence(gbm)),4),25)
```

```
n.trees not given. Using 100 trees.
n.trees not given. Using 100 trees.
```

trip_distance: 0.8968 **dropoff_neighborhoods.longitude:** 0.0274 **dropoff_neighborhoods.latitude:** 0.0181 **`MINUTE(dropoff_datetime)`:** 0.0083 **`pickup_neighborhoods.MEAN(trips.passenger_count)`:** 0.0045 **`dropoff_neighborhoods.SUM(trips.trip_distance)`:** 0.0045 **`pickup_neighborhoods.COUNT(trips)`:** 0.0042 **`pickup_neighborhoods.SUM(trips.trip_distance)`:** 0.0039 **pickup_neighborhoods.longitude:** 0.0029 **`DAY(pickup_datetime)`:** 0.0025 **`dropoff_neighborhoods.MAX(trips.trip_distance)`:** 0.0021 **`dropoff_neighborhoods.SUM(trips.vendor_id)`:** 0.0018 **`pickup_neighborhood = AU`:** 0.0016 **`dropoff_neighborhoods.SUM(trips.passenger_count)`:** 0.0015 **`pickup_neighborhoods.MEAN(trips.trip_distance)`:** 0.0014 **payment_type:** 0.0014 **`dropoff_neighborhoods.MEDIAN(trips.trip_distance)`:** 0.0014 **`dropoff_neighborhoods.MEAN(trips.vendor_id)`:** 0.0012 **`dropoff_neighborhoods.MEAN(trips.trip_distance)`:** 0.0012 **pickup_neighborhoods.latitude:** 0.0011 **`dropoff_neighborhood = AA`:** 9e-04 **`dropoff_neighborhoods.SUM(trips.payment_type)`:** 9e-04 **`pickup_neighborhoods.MEDIAN(trips.payment_type)`:** 8e-04 **`pickup_neighborhoods.MEDIAN(trips.trip_distance)`:** 8e-04 **`pickup_neighborhoods.MAX(trips.trip_distance)`:** 7e-04

Evaluate on Test Data

In [26]:

```
y_pred <- predict(gbm,X_test,n.trees = 100)
y_pred <- exp(y_pred) - 1 # undo the log we took earlier

y_test <- train_test$y_test

print('y_pred and y_test computation successful!')
```

```
[1] "y_pred and y_test computation successful!"
```

In [27]:

```
# Print the first 5 predictions and real values
head(round(y_pred),10)
```

```r
head(round(y_test),10)
```

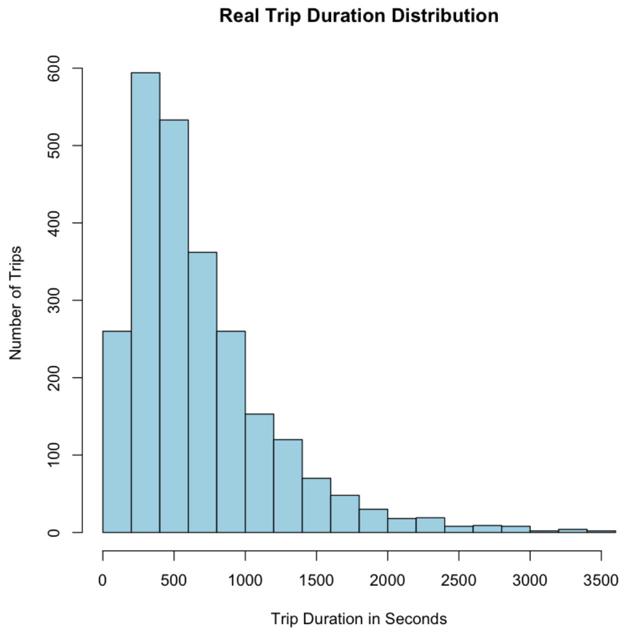
505 · 764 · 766 · 705 · 508 · 1531 · 520 · 1219 · 805 · 661

357 · 570 · 520 · 519 · 390 · 1146 · 553 · 1050 · 603 · 599

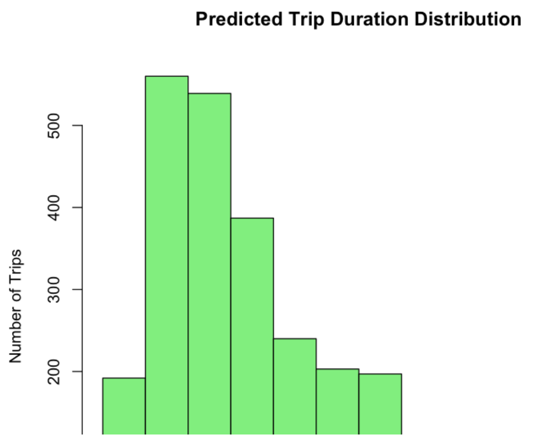
In [28]:

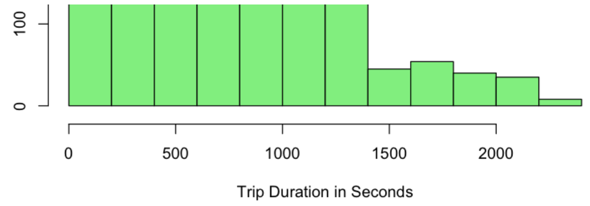
```r
# Histogram of y_test
hist(y_test, main="Real Trip Duration Distribution",
  xlab="Trip Duration in Seconds",
  ylab="Number of Trips",
  col="lightblue")
summary(y_test)

# Histogram of y_pred
hist(y_pred, main="Predicted Trip Duration Distribution",
  xlab="Trip Duration in Seconds",
  ylab="Number of Trips",
  col="lightgreen")
summary(y_pred)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
    2.0   321.0   546.5   681.0   885.2  3573.0
```




```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  12.63  356.67  583.79  700.14  956.31 2361.49
```




100
0
0
500
1000
1500
2000
Trip Duration in Seconds