# <u>USE CASE STUDY REPORT</u>

**Student Names**: Anisha Ganeshkumar and Sundarakishore Giri

## Estimation of Train Dwell time based on track occupation data

## I. Background and Introduction

Train dwell time estimation is a critical issue in both scheduling and rescheduling phases yet it is one of the most unpredictable components of railway operations, mainly because of the varying volumes in the number of passengers who board and disembark from the train at each stop. However, for reliable estimation of train running time tables, it is paramount to obtain a close to accurate estimation of the train dwell times mainly at short stops. Previous study found out that dwell time is highly dependent on the number of boarding and alighting passengers. However, these details are not readily available in real-time and it is not cost efficient to collect them as well. Therefore, our case study focuses on the possibility of a dwell time estimation model at short stops without passenger demand information and only using the track occupation data. For our study, Northeastern train stop which belongs to Green Line-E is used. This model shows an RMSE of .We conclude that dwell time estimation without passenger data is possible

## II. Data Exploration and Visualization

Variable identification was done during literature review, the target variable is dwell time with 6 predictor variables which will be explained in detail in upcoming sections . Firstly, the summary statistics of dwell times of 3 stations (Prudential – Symphony - Northeastern) were checked to find the central tendency and spread of the variable. Univariate analysis for dwell times at Northeastern, Symphony and Prudential stations were done. Secondly, to visualize the frequency of the dwell time distribution the numerical variable was split into bins and the histogram was plotted.

NA values were checked after collating individually collected data for different stations into a single dataset. Headways time was found to have NA values and it was replaced by the average headways time of the train on the day. There was no NA's for dwell time in the dataset as it a user calculated column from departure and arrival time.Outliers were detected and removed in this process

## III. Data Preparation and Preprocessing

MBTA provides their data in an unparsed format. The steps involved in making the data model ready is explained here .Since the API call provides details only for a time period of 22 hours, the data was programmatically collected for this time period starting from September 2018 to March 2019 .Once the data was collected ,only required column such as Trip id(unique),Train arrival time, departure time and service date was used. The difference between departure time and arrival time will give us the dwell time of train at target stop. The data type of the columns was in character format with time in Unix time stamp, steps were taken to convert the data types to required format such datatime object and numbers. The above-mentioned data collection process was followed for train stops such as Northeastern, Symphony and Prudential. Headways time which is the time difference two train arrivals was calculated for Northeastern stop to check if it influences the dwell time.

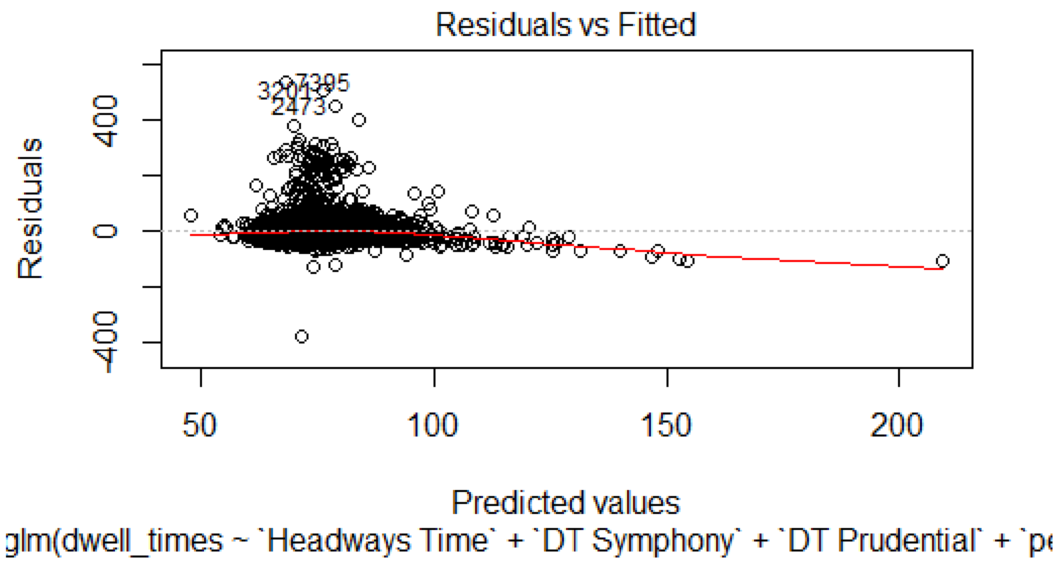
As main libraries we have used Jsonlite , anytime ,tidyr ,for visualization we have used ggplot2.Variable conversion was done as the raw data we acquired from MBTA was in character format .Once , the data was ready we did  correlational analysis was done to check how dwell times are being influenced by our predictor variables .

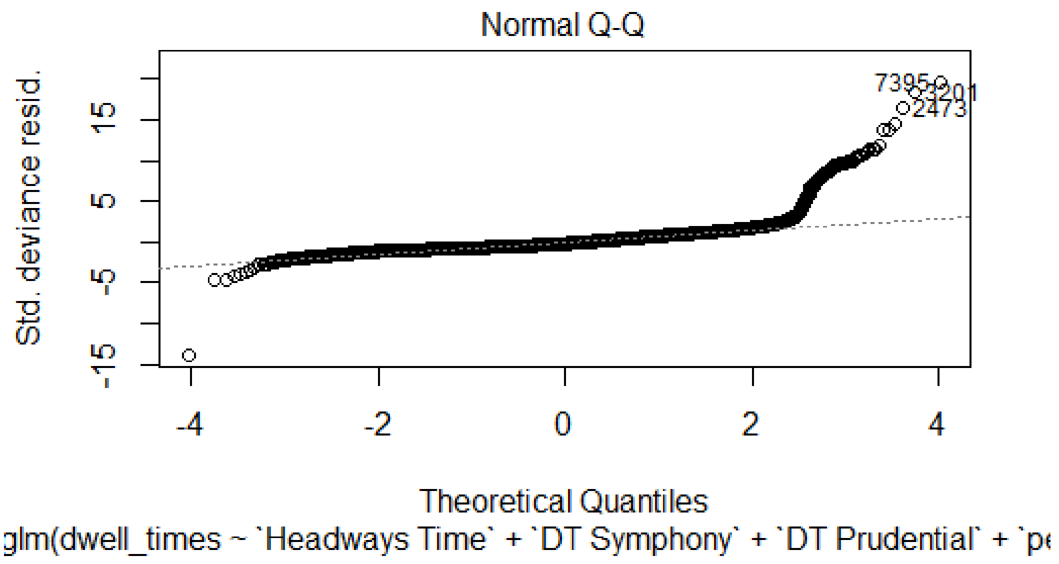
## IV. Data Mining Techniques and Implementation

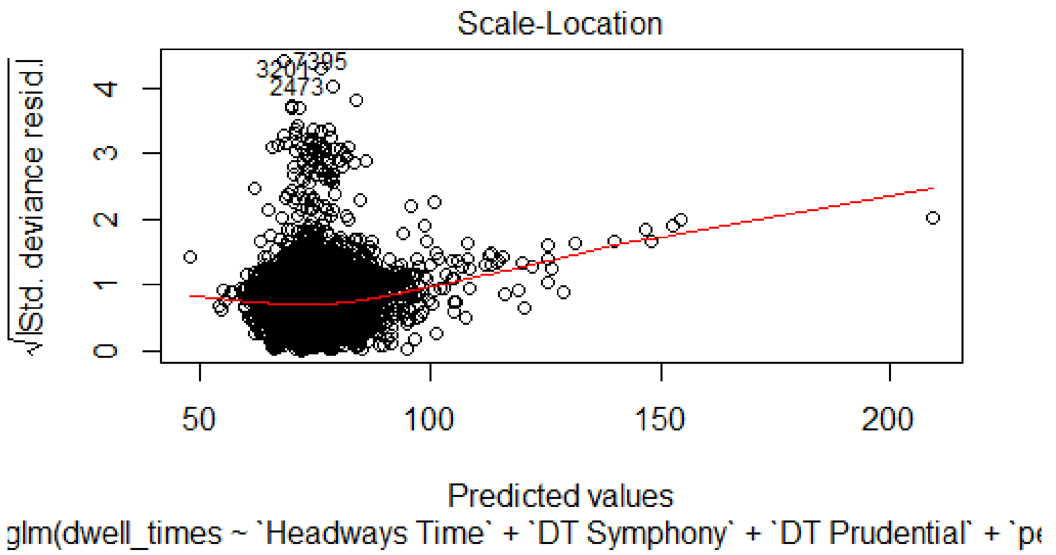
This is a prediction problem where we are predicting the dwell time of trains at a particular station (i.e.) in our case the dwell time of green E line trains arriving and departing at Northeastern university station. As there is no significant correlation between the dependent variable and the predictor variables, a Generalized linear model is proposed.

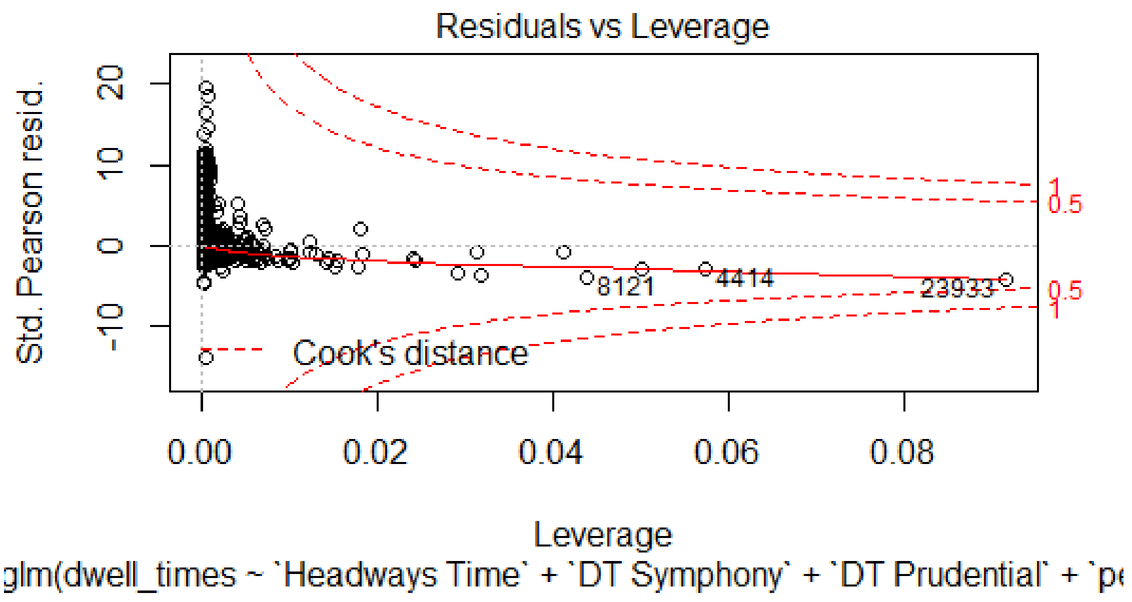
$$y=\beta_0+\beta_1x_1+\beta_2x_2+\cdots$$

After converting the difftime variables into numeric types, the dataset is split into training data set and test data set. The Generalized linear model is fitted to the dataset and a stepwise estimation approach is followed to choose the best fit model. Akaike information criterion (AIC) is used to compare the models. After selecting the better model, the dwell time of the trains in the test set is predicted. The fit is plotted to gain more insight on the model including the Residuals, Leverage and scale location.




Normal Q-Q
Std. deviance resid.
-15
-5
5
15
-4
-2
0
2
4
Theoretical Quantiles
glm(dwell_times ~ `Headways Time` + `DT Symphony` + `DT Prudential` + `pe
7395
3201
2473


Scale-Location
√|Std. deviance resid.|
0
1
2
3
4
50
100
150
200
Predicted values
glm(dwell_times ~ `Headways Time` + `DT Symphony` + `DT Prudential` + `pe
3201
7395
2473



The coefficients of the model are given by,

```
(Intercept)     `Headways Time`       `DT Symphony`
     46.569910344         0.002818831         0.338458343
   `DT Prudential`    `peak/off-peak`1  `weekday/weekend`1
      0.134248401        -2.209188998         5.541917797
```

## V. Performance Evaluation

The prediction accuracy and the error rate are calculated to evaluate the model. A correlation matrix of the actual and the predicted value is computed, with which the Min Max accuracy and the MAPE of the model is calculated.

```
             actual  predicted
actual    1.0000000 0.2053099
predicted 0.2053099 1.0000000
```

The Min Max accuracy is 0.8073 and the MAPE is 0.2444 which suggests a good model but the RMSE and the MAE measures does not give an ideal value. The model suggests an accuracy rate of 20%.
To further validate our model, K-fold cross validation is computed with the k value as 10. Again, the accuracy measures of the 10 folds of data set are calculated and the mean of the measures are computed which again gives a Min Max accuracy closer to 0.801 and the MAPE as 0.2401 which further validates the built model.

## VI. Discussion and Recommendation

The possibility of predicting dwell time without passenger information is explored in this case study. Most of the existing dwell time models is estimated as a function of number of boarding and alighting passengers and based on the station layout which is difficult to obtain in real-time and thus there is a need for a generalized model. One of main disadvantages of using passenger related information is the way people board in a train and how the station layout has been set up .For example there are train stops in Boston which requires the public to tap the Charlie before entering the station and other short stops where the public tap the card while getting into the train .This process influences the train dwell times and so a generalized model which can work for all stops has to be designed. The model we have designed here gives an idea of how a model performs based on track occupation data which is quite encouraging as it gives us hope that approaching in this direction will help us build a generalized model.Future study should be focused two directions ,one area is  where MBTA can provide vehicle specific details such as number of compartments in the train , length of the train so that those details can be imparted in the model in real-time and we expect them to have high correlation with the dwell time. The second direction is with changing the model ,randomness and weak relationship between dwell time and other factors makes dwell time estimation more difficult.To improve the accuracy of the model ,other non-parameter methods such as Kernel regression and the nearest neighbors method ,which do not require significant amount of data can be tested .

## VII. Summary

Generalized dwell time model using track occupation data is proposed in this case study .We have also provided steps to use raw data from MBTA and predict dwell times which can be used in mobile apps that predicts the train arrival and departure.

## Appendix: R Code for use case study

```
library(jsonlite)
library(MBTAr)
library(anytime)
library(lubridate)
library(tidyr)
library(reshape2)

############Automating data collection######
#epoch time in list from september 2018 to April 2019#
a<-list()
b<-list()
i<-0
a[1]<-as.numeric(as.POSIXct("2018-09-01 5:30:00 GMT2"))
b[1]<-as.numeric(as.POSIXct('2018-09-02 00:30:00 GMT2'))
for(i in 2:212){
    a[i]<-as.numeric(as.POSIXct("2018-09-01 5:30:00
GMT2")+days(i-1))
```

```
    b[i]<-as.numeric(as.POSIXct('2018-09-02 00:30:00
GMT2')+days(i-1))
}

#######Use the above epoch time in calling API
##################
url<-""
url<-
paste("http://realtime.mbta.com/developer/api/v2.1/events?a
pi_key=Wf00QzWzS0-
PU6vysSDmKA&format=json&from_datetime=",a[1],"&to_datetime=
",b[1],'&stop=70239',sep='')
df_temp<-fromJSON(url)
df<-as.data.frame(df_temp[["events"]])
for(i in 2:212){
    url<-
paste("http://realtime.mbta.com/developer/api/v2.1/events?a
pi_key=Wf00QzWzS0-
PU6vysSDmKA&format=json&from_datetime=",a[i],"&to_datetime=
",b[i],'&stop=70239',sep='')
    df_temp<-fromJSON(url)
    d<-as.data.frame(df_temp[["events"]])
    df<-rbind(df,d)
}
###########################################################
###
###check data typesof all columns
str(df)
####removing unwanted columns ####
df<-df[,-c(2,4,5,6,7,8,9,12)]
#####Check how many types are in event type
unique(df$event_type)
#####Convert PRD with DEP & PRA with ARR ##########
index<-df$event_type=='PRD'
df$event_type[index]<-'DEP'
index<-df$event_type=='PRA'
df$event_type[index]<-'ARR'
####Convert time from epoch to local time#####
df$event_time<-anytime(as.numeric(df$event_time))
###########Reshaping Data frame######
index1<-df$event_type=='ARR'
index2<-df$event_type=='DEP'
arr.df<-df[index1,]
dep.df<-df[index2,]
arr.df<-arr.df[,-c(3)]
dep.df<-dep.df[,-c(3)]
colnames(arr.df)[3]<-'Arrival time'
colnames(dep.df)[3]<-'Departure time'
prudential_df<-
merge.data.frame(arr.df,dep.df,by=c("trip_id","service_date
"))
prudential_df['dwell_times']<-prudential_df$`Departure
time`-prudential_df$`Arrival time`
###############################################
######Structuring dataset for Symphony#######
url1<-""
url1<-
paste("http://realtime.mbta.com/developer/api/v2.1/events?a
pi_key=Wf00QzWzS0-
PU6vysSDmKA&format=json&from_datetime=",a[1],"&to_datetime=
",b[1],'&stop=70241',sep='')
df_temp1<-fromJSON(url1)
df1<-as.data.frame(df_temp1[["events"]])
for(i in 2:212){
  url1<-
paste("http://realtime.mbta.com/developer/api/v2.1/events?a
pi_key=Wf00QzWzS0-
PU6vysSDmKA&format=json&from_datetime=",a[i],"&to_datetime=
",b[i],'&stop=70241',sep='')
  df_temp1<-fromJSON(url1)
  d1<-as.data.frame(df_temp1[["events"]])
  df1<-rbind(df1,d1)
}
###########################################################
###
###check data typesof all columns#####
str(df1)
#####Check how many types are in event type####
unique(df1$event_type)
#####Convert PRD with DEP & PRA with ARR ##########
index<-df1$event_type=='PRD'
df1$event_type[index]<-'DEP'
index<-df1$event_type=='PRA'
df1$event_type[index]<-'ARR'
####removing unwanted columns ########
df1<-df1[,-c(2,4,5,6,7,8,9,12)]
####Convert time from epoch to local time#####
df1$event_time<-anytime(as.numeric(df1$event_time))
###########Reshaping Data frame######
index1<-df1$event_type=='ARR'
index2<-df1$event_type=='DEP'
arr.df1<-df1[index1,]
dep.df1<-df1[index2,]
arr.df1<-arr.df1[,-c(3)]
```

```
dep.df1<-dep.df1[,-c(3)]
colnames(arr.df1)[3]<-'Arrival time'
colnames(dep.df1)[3]<-'Departure time'
symphony_df<-
merge.data.frame(arr.df1,dep.df1,by=c("trip_id","service_da
te"))
symphony_df['dwell_times']<-symphony_df$`Departure time`-
symphony_df$`Arrival time`
###############################################
#############Northeastern Dataset########
url2<-""
url2<-
paste("http://realtime.mbta.com/developer/api/v2.1/events?a
pi_key=Wf00QzWzS0-
PU6vysSDmKA&format=json&from_datetime=",a[1],"&to_datetime=
",b[1],'&stop=70243',sep='')
df_temp2<-fromJSON(url2)
df2<-as.data.frame(df_temp2[["events"]])
for(i in 2:212){
  url2<-
paste("http://realtime.mbta.com/developer/api/v2.1/events?a
pi_key=Wf00QzWzS0-
PU6vysSDmKA&format=json&from_datetime=",a[i],"&to_datetime=
",b[i],'&stop=70243',sep='')
  df_temp2<-fromJSON(url2)
  d3<-as.data.frame(df_temp2[["events"]])
  df2<-rbind(df2,d3)
}
###check data typesof all columns
str(df2)
#####Check how many types are in event type
unique(df2$event_type)
#####Convert PRD with DEP & PRA with ARR ##########
index<-df2$event_type=='PRD'
df2$event_type[index]<-'DEP'
index<-df2$event_type=='PRA'
df2$event_type[index]<-'ARR'
####removing unwanted columns ####
df2<-df2[,-c(2,4,5,6,7,8,9,12)]
####Convert time from epoch to local time#####
df2$event_time<-anytime(as.numeric(df2$event_time))
###########Reshaping Data frame######
index1<-df2$event_type=='ARR'
index2<-df2$event_type=='DEP'
arr.df2<-df2[index1,]
dep.df2<-df2[index2,]
arr.df2<-arr.df2[,-c(3)]
```

```
dep.df2<-dep.df2[,-c(3)]
colnames(arr.df2)[3]<-'Arrival time'
colnames(dep.df2)[3]<-'Departure time'
NEU_df<-
merge.data.frame(arr.df2,dep.df2,by=c("trip_id","service_da
te"))
NEU_df['dwell_times']<-NEU_df$`Departure time`-
NEU_df$`Arrival time`
#######################################

prudential_df['dwell_times']<-prudential_df$`Departure
time`-prudential_df$`Arrival time`
colnames(symphony_df)[5]<-"DT Symphony"
colnames(prudential_df)[5]<-"DT Prudential"
symphony_df['Dpt_delay_sumphony']<-symphony_df$`DT
Symphony`-30
#####Merge all 3 dataframes###
Neu_copy<-data.table::copy(NEU_df)
Neu_copy<-
merge(Neu_copy,symphony_df,by=c("trip_id","service_date"))

Neu_copy<-
merge(Neu_copy,prudential_df,by=c("trip_id","service_date")
)
Neu_copy<-Neu_copy[,-c(6,7,10,11)]

Neu_copy<-
Neu_copy[order(Neu_copy$service_date,Neu_copy$`Arrival
time.x`),]
Neu_copy$`Arrival time.x`<-strftime(Neu_copy$`Arrival
time.x`,format = "%H:%M")
Neu_copy['weekday/weekend']<-
ifelse(is.weekend(Neu_copy$service_date)==TRUE,0,1)
str(Neu_copy)
m_peak1<-strftime(as.POSIXct("06:30",format="%H:%M"),format
= "%H:%M")
m_peak2<-
strftime(as.POSIXct("9:00",format="%H:%M"),format="%H:%M")
m_peak3<-
strftime(as.POSIXct("18:30",format="%H:%M"),format="%H:%M")
m_peak4<-
strftime(as.POSIXct("20:00",format="%H:%M"),format="%H:%M")
Neu_copy['peak/off-peak']<-ifelse((Neu_copy$`Arrival
time.x`>=m_peak1 & Neu_copy$`Arrival time.x`<=m_peak2) |
(Neu_copy$`Arrival time.x`>=m_peak3 & Neu_copy$`Arrival
time.x`<=m_peak4),1,0)
Neu_copy<-Neu_copy[,-c(3)]
```

```
#################Data Collection for Headways time######
url1<-""
url1<-
paste("http://realtime.mbta.com/developer/api/v2.1/headways
?api_key=Wf00QzWzS0-
PU6vysSDmKA&format=json&stop=70243&from_datetime=",a[1],'&t
o_datetime=',b[1],sep='')
df_temp1<-fromJSON(url1)
df1<-as.data.frame(df_temp1[["headways"]])
df1<-df1[,-c(8,9,10)]

for(i in 2:212){
url<-
paste("http://realtime.mbta.com/developer/api/v2.1/headways
?api_key=Wf00QzWzS0-
PU6vysSDmKA&format=json&stop=70243&from_datetime=",a[i],'&t
o_datetime=',b[i],sep="")
df_temp1<-fromJSON(url)
d<-as.data.frame(df_temp1[["headways"]])
d<-d[,-c(8,9,10)]
df1<-rbind(df1,d)
}


#################
df1$current_dep_dt<-anytime(as.numeric(df1$current_dep_dt))
df1$previous_dep_dt<-
anytime(as.numeric(df1$previous_dep_dt))
#df1<-df1[,c(4,6)]
###########
df1_copy<-data.table::copy(df1)
df1_copy<-df1_copy[,c(4,6)]
df1_copy$current_dep_dt<-
anytime::anytime(as.numeric(df1_copy$current_dep_dt))
#df1['service_date']<-df1$current_dep_dt
#df1['current_dpt_time']<-
anytime(as.numeric(df1$current_dep_dt,format = "%H:%M:%S"))
#############################
time_ratio<-
as.hms(strftime(as.POSIXct("00:30:00",format="%H:%M:%S"),fo
rmat = "%H:%M:%S"))
df1_copy['Service_date']<-
ifelse(as.hms(strftime(df1_copy$current_dep_dt,format="%H:%
M:%S"))-time_ratio<0,df1_copy$current_dep_dt-
days(1),df1_copy$current_dep_dt)
```

```
#df1_copy['time difference']<-
as.hms(strftime(df1_copy$current_dep_dt,format="%H:%M:%S"))
-time_ratio
df1_copy$Service_date<-
anytime(as.numeric(df1_copy$Service_date,format="%H:%M:%S")
)
df1_copy$Service_date1<-as.Date(df1_copy$Service_date)
df1_copy<-df1_copy[,-c(3)]
colnames(df1_copy)[3]<-'service_date'
############################################
#new_df<-data.table::copy(Neu_copy)

#df1_copy<-df1_copy[,c(10,4,6)]
df1_copy['departure']<-
strftime(df1_copy$current_dep_dt,format="%H:%M")
df1_copy['primarykey']<-
paste(df1_copy$service_date,df1_copy$departure)
df1_copy<-df1_copy[,c(5,2)]

Neu_copy['departure']<-strftime(Neu_copy$`Departure
time.x`,format="%H:%M")
Neu_copy['primarykey']<-
paste(Neu_copy$service_date,Neu_copy$departure)
Neu_copy<-Neu_copy[,-c(10)]
#colnames(df1_copy)<-c('service_date','Departure
time','headways_NEU')
colnames(df1_copy)
colnames(new_df)
#df1_copy<-df1_copy[,-c(1,2,4)]
#new_df<-new_df[,-c()]
#setkey

Final_df<-
join(Neu_copy,df1_copy,type="left",by="primarykey",match="f
irst")
#Final_df$headway_time_sec<-
lapply(Final_df$headway_time_sec,function(x)
ifelse(is.na(x),mean(x,na.rm=TRUE),x))
Final_df['Headways Time']<-
ifelse(is.na(Final_df$headway_time_sec),round(mean(as.numer
ic(Final_df$headway_time_sec),na.rm=TRUE)),Final_df$headway
_time_sec)
Final_df<-Final_df[,-c(11)]
##count na's in Headway seconds ######
sum(is.na(Final_df$`Headways Time`))
Final_df<-Final_df[,c(2,3,9,10,7,6,8,11,5)]
```

```
### Building the model
### Loading the Dataset
load("finaldataset.rda")
### Preprocessing
Final_df$`weekday/weekend`<-
as.factor(Final_df$`weekday/weekend`)
Final_df$`peak/off-peak`<-as.factor(Final_df$`peak/off-
peak`)
Final_df$Dpt_delay_sumphony<-
as.numeric(Final_df$Dpt_delay_sumphony)
Final_df$`DT Symphony`<-as.numeric(Final_df$`DT Symphony`)
Final_df$`DT Prudential`<-as.numeric(Final_df$`DT
Prudential`)
Final_df$`Headways Time`<-as.numeric(Final_df$`Headways
Time`)
Final_df$dwell_times<-as.numeric(Final_df$dwell_times)
Final_dataset<-Final_df[3:9]
summary(Final_dataset)
###Splitting the Dataset into training set and test set
library(caTools)
set.seed(123)
split <-sample.split(Final_dataset$dwell_times,SplitRatio =
2/3)
training_set <-subset(Final_dataset,split==TRUE)
test_set<-subset(Final_dataset,split==FALSE)
###Fitting the model(Gaussian)
fit1<-glm(dwell_times~`Headways Time`,family =
gaussian(link = "identity"),data = training_set )
summary(fit1)
fit2<-glm(dwell_times~`Headways Time`+`DT Symphony`,family
= gaussian(link = "identity"),data = training_set )
summary(fit2)
fit3<-glm(dwell_times~`Headways Time`+`DT Symphony`+`DT
Prudential`,family = gaussian(link = "identity"),data =
training_set )
summary(fit3)
fit4<-glm(dwell_times~`Headways Time`+`DT Symphony`+`DT
Prudential`+`peak/off-peak`,family = gaussian(link =
"identity"),data = training_set )
summary(fit4)
fit<-glm(dwell_times~`Headways Time`+`DT Symphony`+`DT
Prudential`+`peak/off-peak`+`weekday/weekend`,family =
gaussian(link = "identity"),data = training_set )
summary(fit)
###Predicting the response on testdata
pred<-predict(fit,newdata = test_set)
pred<-as.data.frame(pred)
```

```
plot(fit)
coef(fit)
exp(coef(fit))
### Prediction accuracy and error rate
actual_pred<-
data.frame(cbind(actual=test_set$dwell_times,predicted=pred
$pred))
correlation_accuracy<-cor(actual_pred)
correlation_accuracy
head(actual_pred)
min_max_accuracy<-
mean(apply(actual_pred,1,min)/apply(actual_pred,1,max))
min_max_accuracy
mape<-mean(abs((actual_pred$predicted-
actual_pred$actual))/actual_pred$actual)
mape
rmse <- function(error)
{
  sqrt(mean(error^2))
}
mae <- function(error)
{
  mean(abs(error))
}
error <- test_set$dwell_times-pred$pred
rmse(error)
mae(error)
library(rcompanion)
accuracy<-accuracy(list(fit,fit4),digits = 3)
###cross-validation
library(caret)
folds<-createFolds(training_set$dwell_times,k=10)
cv<-lapply(folds,function(x){
  training_fold = training_set[-x,]
  test_fold = training_set[x,]
  fit_cv<-glm(dwell_times~`Headways Time`+`DT Symphony`+`DT
Prudential`+`peak/off-peak`+`weekday/weekend`,family =
gaussian(link = "identity"),data = training_fold )
  pred_cv<-predict(fit,newdata = test_fold)
  pred_cv<-as.data.frame(pred_cv)
  actual_pred_cv<-
data.frame(cbind(actual_cv=test_fold$dwell_times,predicted_
cv=pred_cv$pred_cv))
  correlation_accuracy_cv<-cor(actual_pred_cv)
  min_max_accuracy_cv<-
mean(apply(actual_pred_cv,1,min)/apply(actual_pred_cv,1,max
))
```

```
  mape_cv<-mean(abs((actual_pred_cv$predicted_cv-
actual_pred_cv$actual_cv))/actual_pred_cv$actual_cv)
  list_cv<-
list("min_max_accuracy_cv"=min_max_accuracy_cv,"mape_cv"=ma
pe_cv)
  return(list_cv)
})
cv
```