Competition 42039 - Assignment

Inesh Munaweera

2023-12-11

This document (generated by RMarkdown) contains answers for all Questions.

Question 1:

Loading required packages and data

```
library(data.table) # To use`fread`
library(httr)
library(tidyverse)
library(dplyr)
library(janitor)
library(stringr)
library(lubridate)
library(leaflet)
library(ggplot2)
library(tidymodels)
```

Automated function to import and load data from the Winnipeg Open Data Portal.

Note: No manual data downloading required to use this function. Provide the `file_path` and number of months you want to load `num_months` when calling the function. This will load the most recent months. e.g.; if `num_months` = 3, it will load the most recent three months.

To use a large number of months, you will need a fast internet connection and sufficient storage in your hard drive. Each month requires about 1GB of space in the hard disk after extracting the zip files and also 1.5GB RAM per month. If your computer cannot accommodate the data, you have to consider a distributed computing system such as `Apache Spark` or `DuckDB`.

```r
get_datasets <- function(file_path, num_months) {
  data_info <- read_csv(file_path)

  latest_data_info <- data_info %>%
    tail(num_months)

  urls <- latest_data_info$URL

  datasets <- lapply(urls, function(url) {

    temp_dir <- file.path(tempdir(), "downloaded_zip_files")
    dir.create(temp_dir, recursive = TRUE, showWarnings = FALSE)


    temp_zip <- tempfile(fileext = ".zip", tmpdir = temp_dir)


    download.file(url, temp_zip, mode="wb")


    unzip(temp_zip, exdir = temp_dir)


    files <- list.files(temp_dir, pattern = "\\.csv$", full.names = TRUE)
    data <- rbindlist(lapply(files, fread))

    unlink(temp_zip)
    unlink(temp_dir, recursive = TRUE)

    return(data)
  })

  return(rbindlist(datasets, use.names = TRUE, fill = TRUE))
}



data <- get_datasets("Transit_On-Time_Performance_Data_Archive.csv", 3)
```

Data cleaning

```
# Note: `fread` is better than the `read.csv` function in handling large datasets.
data <- data %>%
  mutate( `Stop Number` = as.factor(`Stop Number`),
          `Route Number` = as.factor(`Route Number`),
          Location = str_remove(Location, "POINT \\("),
          Location = str_remove(Location, "\\)"),
          lon = as.numeric(str_extract(Location, "^[^ ]+")),
          lat = as.numeric(str_extract(Location, "[^ ]+$")),
          Deviation = - Deviation/60   )   %>%    # Transform so that the delays will be denoted
by positive values and in minutes
  select(-Location)%>%
  clean_names()

glimpse(data,5)
```

```
## Rows: 14,051,788
## Columns: 10
## $ row_id            <int> …
## $ stop_number       <fct> …
## $ route_number      <fct> …
## $ route_name        <chr> …
## $ route_destination <chr> …
## $ day_type          <chr> …
## $ scheduled_time    <dttm> …
## $ deviation         <dbl> …
## $ lon               <dbl> …
## $ lat               <dbl> …
```

There are 14 million data records in the last three month.

Exploratory Data Analysis

```
str(data)
```

```
## Classes 'data.table' and 'data.frame':   14051788 obs. of  10 variables:
##  $ row_id           : int  1198723723 1198723725 1198723727 1198723729 1198723731 1198723733
1198723735 1198723737 1198723739 1198723741 ...
##  $ stop_number      : Factor w/ 5144 levels "10001","10002",..: 570 571 573 574 143 3234 3289
3291 3293 3296 ...
##  $ route_number     : Factor w/ 86 levels "10","11","12",..: 4 4 4 4 4 4 4 4 4 4 ...
##  $ route_name       : chr  "Ellice-St. Mary's" "Ellice-St. Mary's" "Ellice-St. Mary's" "Ellic
e-St. Mary's" ...
##  $ route_destination: chr  "South St. Vital via Dakota" "South St. Vital via Dakota" "South S
t. Vital via Dakota" "South St. Vital via Dakota" ...
##  $ day_type         : chr  "Weekday" "Weekday" "Weekday" "Weekday" ...
##  $ scheduled_time   : POSIXct, format: "2023-08-01 07:08:18" "2023-08-01 07:09:40" ...
##  $ deviation        : num  2.83 2.25 3.67 3.58 3.37 ...
##  $ lon              : num  -97.1 -97.1 -97.1 -97.1 -97.1 ...
##  $ lat              : num  49.9 49.9 49.9 49.9 49.9 ...
##  - attr(*, ".internal.selfref")=<externalptr>
```

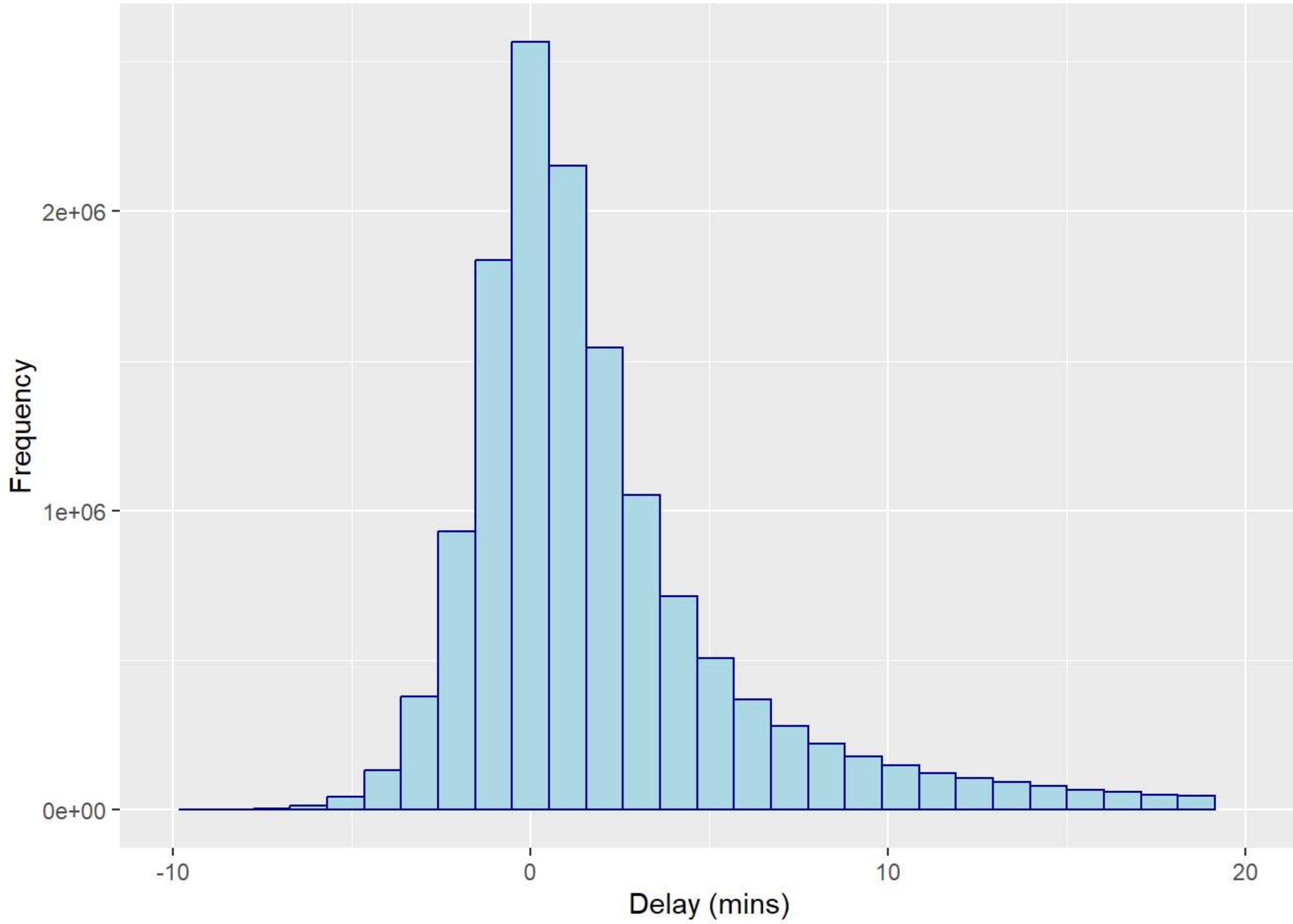
```
cat("Total number of routes:", length(unique(data$route_number )), "\n")
```

```
## Total number of routes: 86
```

```
cat("Total number of stops:", length(unique(data$stop_number )), "\n")
```

```
## Total number of stops: 5144
```

```
ggplot(data, aes(x = deviation)) +
  geom_histogram(fill = "lightblue", color = "darkblue") +
  xlab("Delay (mins)") +
  ylab("Frequency") +
  xlim(c(-10,20))
```




```
summary(data$deviation)
```

```
##      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
## -460.7167   -0.4667    1.0167    2.4143    3.4500  357.4833
```

The distribution of delay is skewed to the right with average around 2.4 minutes. We see the maximum to be 357 minutes which is n extreme case. I recommend using the median as the measure of central tendency due to the skewness, but I will still use the average for the illustrative purpose from here.

Examing outliers

```
print(paste0("Percentage of ourlier using the 1.5*IQR rule :",round(length(data$deviation[data$d
eviation %in% boxplot.stats(data$deviation)$out])/nrow(data)*100,2)))
```

```
## [1] "Percentage of ourlier using the 1.5*IQR rule :8.69"
```

There are too many outliers (9%) to remove when using the standard 1.5*IQR rule. Most of these should be actual observations. Hence I do not recommend removing them. This can be handled later in modelling. However, we will ignore the 5% of observations in the tails when calculating summary measures to minimize the effect from the observations at the tails.

Identifying the routes with largest average delay

```
# Calculate average deviation for each route
mean_delay_by_route <- data %>%
  filter(deviation > 0) %>%  # Consider only the delays
  group_by(route_number,route_name,route_destination) %>%
  summarise(n_stops= length(unique(stop_number)),observations = n(),mean_delay = mean(deviation,
trim = 0.05, na.rm = TRUE),sd_delay=sd(deviation, na.rm = TRUE)) %>%
  arrange(desc(mean_delay))
```

```
## `summarise()` has grouped output by 'route_number', 'route_name'. You can
## override using the `.groups` argument.
```

```
mean_delay_by_route[1:10,]
```

```
## # A tibble: 10 × 7
## # Groups:   route_number, route_name [8]
##    route_number route_name     route_destination n_stops observations mean_delay
##    <fct>        <chr>          <chr>               <int>        <int>      <dbl>
##  1 77           Crosstown Nor… Whellams Lane         108         1391      18.2
##  2 58           Dakota Express South St. Vital …      61        14277      15.0
##  3 58           Dakota Express South St. Vital …      54        11750      14.2
##  4 26           Logan - Berry  Portage & Tylehu…      59         2187      12.0
##  5 54           St. Mary's Ex… South St. Vital        45         8591      11.1
##  6 59           South St. Ann… Aldgate                62        14940      10.4
##  7 59           South St. Ann… Island Lakes           46        10905      10.0
##  8 68           Grosvenor      Stradbrook & Osb…      23          571       9.97
##  9 79           Charleswood    Portage & Tylehu…      48         1733       9.71
## 10 19           Marion-Logan-… Elizabeth & Drake      31         3031       9.57
## # ℹ 1 more variable: sd_delay <dbl>
```

Crosstown North towards Whellams Lane shows the largest delay.

Busiest routes

```
mean_delay_by_route_busy <- mean_delay_by_route %>% arrange(desc(observations))
print(mean_delay_by_route_busy[1:10,])
```

```
## # A tibble: 10 × 7
## # Groups:   route_number, route_name [8]
##    route_number route_name     route_destination n_stops observations mean_delay
##    <fct>        <chr>          <chr>               <int>        <int>      <dbl>
##  1 77           Crosstown Nor… Polo Park             146       233861       3.61
##  2 47           Transcona - P… University of Ma…     105       207677       3.95
##  3 77           Crosstown Nor… Kildonan Place        141       203498       3.67
##  4 18           North Main-Co… Tuxedo                107       188374       3.80
##  5 60           Pembina        Downtown               56       166162       4.63
##  6 14           Ellice-St. Ma… Ferry Road            117       164825       3.10
##  7 21           Portage Expre… City Hall              85       158698       2.55
##  8 18           North Main-Co… Garden City Cent…      95       158614       3.92
##  9 11           Portage-Kildo… Polo Park             100       146170       4.02
## 10 BLUE         Route BLUE     Downtown               42       142955       3.39
## # ℹ 1 more variable: sd_delay <dbl>
```

77 Crosstown North to Polo Park is the most active route with most data records.

Stop with larges average delay

```
mean_delay_by_stop <- data %>%
  filter(deviation > 0) %>%  # Consider only the delays
  group_by(stop_number) %>%
  summarise(observations = n(),
            mean_delay = mean(deviation, trim = 0.05, na.rm = TRUE),
            sd_delay=sd(deviation, na.rm = TRUE),
            lon = first(lon),
            lat = first(lat)) %>%
  arrange(desc(mean_delay))%>%
  filter(observations>20) # ignoring stops with a small number of observations

mean_delay_by_stop[1:12,]
```

```
## # A tibble: 12 × 6
##    stop_number observations mean_delay sd_delay   lon   lat
##    <fct>              <int>      <dbl>    <dbl> <dbl> <dbl>
##  1 50434                229       18.3     9.90 -97.1  49.8
##  2 50435                228       18.1     9.94 -97.1  49.8
##  3 50721                226       17.9     9.87 -97.1  49.8
##  4 50807                222       16.9     9.91 -97.1  49.8
##  5 50764                502       15.0    10.2  -97.1  49.8
##  6 50818                501       14.9    10.7  -97.1  49.8
##  7 50828                497       14.6    10.2  -97.1  49.8
##  8 50964                494       14.0    10.1  -97.1  49.8
##  9 50965                493       13.8    10.1  -97.1  49.8
## 10 50591                196       12.9     9.02 -97.1  49.8
## 11 51001                193       12.9     8.92 -97.1  49.8
## 12 50987                193       12.9     8.99 -97.1  49.8
```

Busiest stops

```
mean_delay_by_stop_busy <- mean_delay_by_stop %>% arrange(desc(observations))
print(mean_delay_by_stop_busy[1:10,])
```

```
## # A tibble: 10 × 6
##    stop_number observations mean_delay sd_delay   lon   lat
##    <fct>              <int>      <dbl>    <dbl> <dbl> <dbl>
##  1 10542              25830       3.77     5.95 -97.1  49.9
##  2 10541              25048       3.72     5.98 -97.1  49.9
##  3 10628              24807       4.10     6.51 -97.1  49.9
##  4 10629              24408       4.24     6.78 -97.1  49.9
##  5 60105              23894       3.44     4.84 -97.1  49.8
##  6 10638              23157       4.50     6.90 -97.1  49.9
##  7 10642              21949       4.77     7.05 -97.1  49.9
##  8 10543              21900       3.65     5.83 -97.1  49.9
##  9 10581              21816       4.08     6.78 -97.1  49.9
## 10 10583              21372       4.23     6.94 -97.1  49.9
```

Average delay at busiest stops

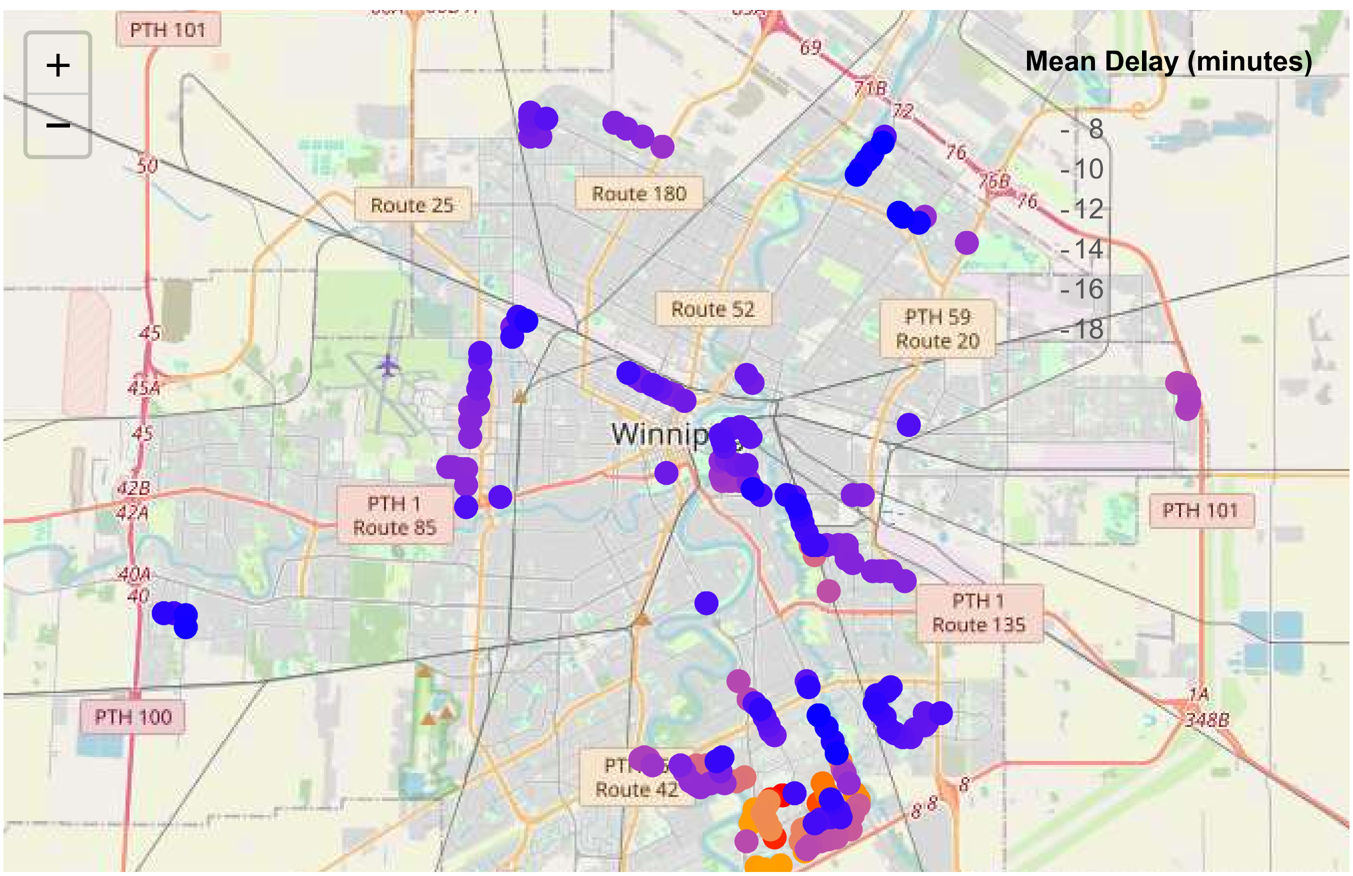
Below is an interactive map you can use to zoom in and see spatial distribution of delays by the top 200 busiest stop. (Note: You will not be able to interact in the pdf version of this map. Knit the rmd file to HTML or run the code within the rmd file directly to see the interactive map.)

```
mean_delay_by_stop_200 <- mean_delay_by_stop %>%
  slice(1:200)

my_cols <- c("red", "orange", "blue") ##1B9E77

pal <- colorNumeric(
  palette = my_cols,
  reverse = TRUE,
  domain = mean_delay_by_stop_200$mean_delay
)

#map
leaflet(mean_delay_by_stop_200) %>%
  addTiles() %>%
  addCircleMarkers(
    ~lon, ~lat,
    radius = 6,
    color = ~pal(mean_delay),
    popup = ~paste("Stop Number:", stop_number, "<br>Mean Delay:", mean_delay),
    fillOpacity = 1,
    stroke = FALSE
  ) %>%
  addLegend(
    pal = pal,
    values = ~mean_delay,
    title = "Mean Delay (minutes)",
    opacity = 1
  )
```



St. Vital area has some of the worst stops where average delays is very high.

Average delay by weekday

```
mean_delay_by_weekday <- data %>%
  filter(deviation > 0) %>%  # Consider only the delays
  group_by(day = wday(scheduled_time,label = TRUE)) %>%
  summarise(observations = n(),mean_delay = mean(deviation,trim = 0.05, na.rm = TRUE),sd_delay=s
d(deviation, na.rm = TRUE))

print(mean_delay_by_weekday)
```

```
## # A tibble: 7 × 4
##   day   observations mean_delay sd_delay
##   <ord>        <int>      <dbl>    <dbl>
## 1 Sun         393809       2.77     4.32
## 2 Mon        1297104       2.98     4.77
## 3 Tue        1894654       3.62     5.89
## 4 Wed        1687143       3.82     6.81
## 5 Thu        1725563       3.97     6.55
## 6 Fri        1702094       3.61     5.81
## 7 Sat         706992       3.28     5.73
```

Most delays happens on Thursdays on average. However, it is interesting to see low average dela on Monday.

Hourly delay on weekdays

```
mean_delay_by_hour_weekdays <- data %>%
  filter(deviation > 0,day_type=="Weekday") %>%  # Consider only the delays
  group_by(hour = hour(scheduled_time)) %>%
  summarise(observations = n(),mean_delay = mean(deviation,trim = 0.05, na.rm = TRUE),sd_delay=s
d(deviation, na.rm = TRUE))

print(mean_delay_by_hour_weekdays)
```

```
## # A tibble: 23 × 4
##     hour observations mean_delay sd_delay
##    <int>        <int>      <dbl>    <dbl>
##  1     0        48638       2.30     3.91
##  2     1        28160       2.08     4.70
##  3     2          559       2.92     4.74
##  4     4          785       2.44     2.69
##  5     5       179577       1.71     2.46
##  6     6       629741       1.74     2.53
##  7     7       701005       1.99     3.22
##  8     8       661782       3.22     4.90
##  9     9       460907       3.05     4.88
## 10    10       374304       2.49     3.89
## # ℹ 13 more rows
```

Average delay is largest during the morning and afternoon rush hours. It's very large aroung 5 p.m.

Question 2:

To build a system to predict late arrivals at a given stop in real-time, we have to go through some key steps. I will explain these steps to a non-technical audience first and extend the steps by adding some specific details for a technical audience.

For a policy audience:

The first step is to collect real-time data including real-time bus locations (using GPS systems), road and traffic conditions, and weather information. Real-time bus locations have to be stored on special servers that can manage a large amount of real-time data. Apache Kafka and RabbitMQ are two popular choices. To collect real-time weather and traffic data, we can use an API such as OpenWeatherMap and Traffic API.

The second step is to develop a model that can predict the bus arrival time (or delay) using real-time data. This is the most critical step since the accuracy of the predictions completely depends on this step. The simplest method we can use is the historical average delays. But this is not the most accurate. Hence, we can use more advanced machine learning methods such as improved specialized versions of neural networks and Bayesian statistical methods. We call them predictive models. The common practice is to try many of these methods and select the best model. To evaluate model performance, we can follow the training set test set approach. Here we train the model in a set of data and test it using the remaining data.

Once we finalize the model, we have to develop a user-friendly dashboard to display the predictions using a platform such as R Shiny.

For a technical audience:

In addition to the above explanation, for a technical audience, I will present more modelling details to the model development section to avoid repetition.

To predict bus delays in real-time, we can build a dynamic spatiotemporal model. There are a large number of such models already developed in the literature. Some of the widely used models include the k-nearest neighbour algorithm, kernel regression, neural networks such as LSTMs and Bayesian methods such as particle filtering, Bayesian networks and Bayesian state-space models (SSM). Each of these methods has advantages and disadvantages. I will use each of these methods and use the k-fold cross-validation method to select the best model. The Bayesian models can be computer-intensive for cross-validation. However, as a Bayesian expert, I identify that the Bayesian SSM approach as the most flexible and strongest candidate due to its natural ability to model complex data and the ability to quantify the precision of the estimates. Also, the Bayesian approach can incorporate prior data and sequentially update it over time making it the most natural approach to model spatiotemporal data such as real time bus delays. I will use the JAGS programming language and run the model on the WestGrid high-performance computer network to account for the computer burden issue. Bayesian models will be further evaluated using posterior predictive checks and compared using DIC and BIC criteria.

Word count: 496

Question 3: Toy example

KNN model

Below we develop a simplified version of the knn model. Here only the observations of route 635 within october 2023 were considered.

```
df <- data %>%
  filter(month(scheduled_time) == 10) %>%
  filter(route_number == "635") %>%
  filter(route_destination == "Harkness Station") %>%
  select(-c(route_number, row_id, route_name, lon, lat, route_destination)) %>%
  mutate(
         # day = as.factor(wday(scheduled_time)),
         hour = as.factor(hour(scheduled_time))) %>%
  select(-c(scheduled_time, day_type))


# data splitting
df_split <- initial_split(df, prop = 0.75, strata = deviation)
df_train <- training(df_split)
df_test <- testing(df_split)

print(df_split)
```

```
## <Training/Testing/Total>
## <7864/2624/10488>
```

The model uses only the stop number and the hour of the day to predict the delay. Those factor variables were converted into dummy variables before training.

```
# Pre-processing recipe
preprocess_recipe <- recipe(deviation ~ ., data = df_train) %>%
  step_dummy(all_nominal_predictors())


preprocess_recipe %>%
  prep()
```

```
##
```

```
## ── Recipe ──────────────────────────────────────────────────────────────────────
```

```
##
```

```
## ── Inputs
```

```
## Number of variables by role
```

```
## outcome:   1
## predictor: 2
```

```
##
```

```
## ── Training information
```

```
## Training data contained 7864 data points and no incomplete rows.
```

```
##
```

```
## ── Operations
```

```
## • Dummy variables from: stop_number, hour | Trained
```

k = 10 was used as the number of neighbors. In the complete analysis k should be tuned. Cross validation can be used for parameter tuning.

```
# Model specifications
knn_spec <- nearest_neighbor(weight_func = "rectangular",
                              neighbors = 10) %>%
  set_engine("kknn") %>%
  set_mode("regression")

# Cross validation folds
# df_folds <- vfold_cv(df_train, v = 5, strata = deviation)
# print(df_vfold)

knn_wkflw <- workflow() %>%
  add_recipe(preprocess_recipe) %>%
  add_model(knn_spec)

knn_wkflw
```

```
## ══ Workflow ════════════════════════════════════════════════════════════════
## Preprocessor: Recipe
## Model: nearest_neighbor()
##
## ── Preprocessor ────────────────────────────────────────────────────────────
## 1 Recipe Step
##
## • step_dummy()
##
## ── Model ───────────────────────────────────────────────────────────────────
## K-Nearest Neighbor Model Specification (regression)
##
## Main Arguments:
##   neighbors = 10
##   weight_func = rectangular
##
## Computational engine: kknn
```

The the model is fitted using the training data and tested on the test set.

```
library(kknn)
library(finetune)

set.seed(345)

knn_fit <- knn_wkflw %>%
  fit(data = df_train)

knn_fit
```

```
## ══ Workflow [trained] ══════════════════════════════════════════════════════
## Preprocessor: Recipe
## Model: nearest_neighbor()
##
## ── Preprocessor ────────────────────────────────────────────────────────────
## 1 Recipe Step
##
## • step_dummy()
##
## ── Model ───────────────────────────────────────────────────────────────────
##
## Call:
## kknn::train.kknn(formula = ..y ~ ., data = data, ks = min_rows(10,     data, 5), kernel = ~"r
ectangular")
##
## Type of response variable: continuous
## minimal mean absolute error: 2.884717
## Minimal mean squared error: 19.06016
## Best kernel: rectangular
## Best k: 10
```

```
test_results <- knn_fit %>%
  predict(df_test) %>%
  bind_cols(df_test) %>%
  metrics(truth = deviation, estimate = .pred)
  # filter(.metric %in% c('rmse', 'mae', 'rsq'))
print(test_results)
```

```
## # A tibble: 3 × 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard      4.49
## 2 rsq     standard      0.0752
## 3 mae     standard      3.01
```

This is a very poorly fitted model. See the low r-square. Above error criteria can be used to compare models.

Next steps:

First, we have to collect more data including environmental data, special event dates, and traffic data. For instance, an accident or major snowfall will significantly increase the number of long delays or the roads are flooded with vehicles of Blue Bombers fans on a game night. Also, there can be systematic delays due to construction on designated locations.

Then, we will use all that data as predictors to explain the delays. In addition, we have to add a few more parameters to address the spatial and temporal correlation among bus stops and different routes. For example, if there is an accident, the first bus to pass the scene will be delayed and this information can be used to predict the arrival of the next bus. This information be shared among different routes as well. The case is the same for snowfalls and constructions.

As I mentioned in Question 2, we cannot just simply fit one model. We must try the multiple models I mentioned in Q2 and select the best among them using some predictive error criteria.