Introduction

For this study, I will dive deep into the Food Delivery Time dataset. The goal of this study is to correctly predict the amount of time (in minutes) that an order will take to get to its destination.

Dataset and Variables

The dataset is made up of 17 variables, described as follows:

- **ID:** Unique identifier for each delivery instance.
- **Delivery_person_ID:** Unique identifier for each delivery person.
- **Delivery_person_Age:** Age of the delivery person.
- **Delivery_person_Ratings:** Customer ratings for the delivery person.
- **Restaurant_latitude:** Geographical latitude coordinate of the restaurant's location.
- **Restaurant_longitude:** Geographical longitude coordinate of the restaurant's location.
- **Delivery_location_latitude:** Latitude coordinate of the delivery location where the order is delivered.
- **Delivery_location_longitude:** Longitude coordinate of the delivery location where the order is delivered.
- **Type_of_order:** Category of food in the order.
- **Type_of_vehicle:** Delivery vehicle.
- **Temperature:** Atmospheric temperature.
- **Humidity:** Humidity level during delivery.
- **Precipitation:** Indication of rain or mist.
- **Weather_description:** Text that describes the weather during delivery.
- **Traffic_Level:** Level of traffic during delivery.
- **Distance..km.:** Distance between restaurant and customer in kilometers.
- **TARGET:** Delivery time in minutes to be predicted.

In order to achieve the goal of this study, the data will be passed through an analysis process, in which different variables will undergo modifications and pattern recognition in order to later on build and train two effective machine learning algorithms.

The algorithms that will be implemented are:

- Linear Regression
- Random Forest

Analysis

In order to start with the analysis process, it is important to install and import two of the main packages for data cleaning and exploration (`dplyr` and `ggplot2`). Also, the dataset will be read from the `data/` directory and it's head (first 6 rows) and structure will be printed out in order to get an initial approach to the dataset itself:

```
if (!require(dplyr)) install.packages("dplyr")
```

```
## Loading required package: dplyr

##
## 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
```

```
library(dplyr)

if (!require(ggplot2)) install.packages("ggplot2")
```

```
## Loading required package: ggplot2
```

```
library(ggplot2)

data <- read.csv("data/Food_Time_Data_Set.csv")

str(data)
```

```
## 'data.frame':    10001 obs. of  18 variables:
##  $ ID                         : chr  "4607" "B379" "5D6D" "7A6A" ...
##  $ Delivery_person_ID         : chr  "INDORES13DEL02" "BANGRES18DEL02" "BANGRES19DEL01" "COIMBRES13D
##  $ Delivery_person_Age        : int  37 34 23 38 32 22 33 35 22 36 ...
##  $ Delivery_person_Ratings    : num  4.9 4.5 4.4 4.7 4.6 4.8 4.7 4.6 4.8 4.2 ...
##  $ Restaurant_latitude        : num  22.7 12.9 12.9 11 13 ...
##  $ Restaurant_longitude       : num  75.9 77.7 77.7 77 80.2 ...
##  $ Delivery_location_latitude : num  22.8 13 12.9 11.1 13 ...
##  $ Delivery_location_longitude: num  75.9 77.8 77.7 77 80.3 ...
##  $ Type_of_order              : chr  "Snack " "Snack " "Drinks " "Buffet " ...
##  $ Type_of_vehicle            : chr  "motorcycle " "scooter " "motorcycle " "motorcycle " ...
##  $ temperature                : num  17.1 19.5 20.4 23.9 26.6 ...
##  $ humidity                   : int  77 93 91 78 87 65 69 82 65 77 ...
##  $ precipitation              : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ weather_description        : chr  "haze" "mist" "mist" "mist" ...
##  $ X                          : logi  NA NA NA NA NA NA ...
##  $ Traffic_Level              : chr  "Low" "Very High" "Low" "Moderate" ...
##  $ Distance..km.              : chr  "" "37.17" "3.34" "10.05" ...
##  $ TARGET                     : chr  "21.66666667" "85.26666667" "28.58333333" "35.18333333" ...
```

Looking at the structure of the dataset, there are `chr` values (like string, normal text), `int` values (integers) and `nums` (floating-point numbers), with `logi` being values either TRUE or FALSE (having a similar behavior to `booleans` in other programming languages).

Now taking a look into the first 6 rows of the dataset:

```
head(data)
```

```
##     ID Delivery_person_ID Delivery_person_Age Delivery_person_Ratings
## 1 4607     INDORES13DEL02                  37                     4.9
## 2 B379     BANGRES18DEL02                  34                     4.5
## 3 5D6D     BANGRES19DEL01                  23                     4.4
## 4 7A6A    COIMBRES13DEL02                  38                     4.7
## 5 70A2     CHENRES12DEL01                  32                     4.6
## 6 9BB4      HYDRES09DEL03                  22                     4.8
##   Restaurant_latitude Restaurant_longitude Delivery_location_latitude
## 1            22.74505             75.89247                   22.76505
## 2            12.91304             77.68324                   13.04304
## 3            12.91426             77.67840                   12.92426
## 4            11.00367             76.97649                   11.05367
## 5            12.97279             80.24998                   13.01279
## 6            17.43167             78.40832                   17.46167
##   Delivery_location_longitude Type_of_order Type_of_vehicle temperature
## 1                   75.91247        Snack      motorcycle       17.11
## 2                   77.81324        Snack         scooter       19.50
## 3                   77.68840       Drinks      motorcycle       20.45
## 4                   77.02649       Buffet      motorcycle       23.86
## 5                   80.28998        Snack         scooter       26.55
## 6                   78.43832       Buffet      motorcycle       21.43
##   humidity precipitation weather_description  X Traffic_Level Distance..km.
## 1       77             0                haze NA           Low
## 2       93             0                mist NA     Very High         37.17
## 3       91             0                mist NA           Low          3.34
## 4       78             0                mist NA      Moderate         10.05
## 5       87             0                mist NA          High          9.89
## 6       65             0      broken clouds NA      Moderate          11.3
##        TARGET
## 1 21.66666667
## 2 85.26666667
## 3 28.58333333
## 4 35.18333333
## 5       43.45
## 6        30.6
```

From this we can see that the `ID` and `Delivery_person_ID` are some type of made-up strings that meant something to another database or system possibly.

Also, it can be pointed out that there are several categorical variables in the dataset like `traffic_level`. Additionally, the X column seems to be of no value to the dataset, although this will be studied later on.

Categorical Columns

`Type_of_order` seems to be a categorical column, but by default it has a 'chr' type, so it will be converted into a factor column (since factors are the main way of handling with categorical variables). The same happens with `type_of_vehicle`, `weather_description` and `traffic_level`, so these columns will also be converted into factors:

```
data <- data %>%
  mutate(Type_of_order = as.factor(Type_of_order),
         Type_of_vehicle = as.factor(Type_of_vehicle),
         weather_description = as.factor(weather_description),
         Traffic_Level = as.factor(Traffic_Level))

str(data)
```

```
## 'data.frame':    10001 obs. of  18 variables:
##  $ ID                         : chr  "4607" "B379" "5D6D" "7A6A" ...
##  $ Delivery_person_ID         : chr  "INDORES13DEL02" "BANGRES18DEL02" "BANGRES19DEL01" "COIMBRES13DE
##  $ Delivery_person_Age        : int  37 34 23 38 32 22 33 35 22 36 ...
##  $ Delivery_person_Ratings    : num  4.9 4.5 4.4 4.7 4.6 4.8 4.7 4.6 4.8 4.2 ...
##  $ Restaurant_latitude        : num  22.7 12.9 12.9 11 13 ...
##  $ Restaurant_longitude       : num  75.9 77.7 77.7 77 80.2 ...
##  $ Delivery_location_latitude : num  22.8 13 12.9 11.1 13 ...
##  $ Delivery_location_longitude: num  75.9 77.8 77.7 77 80.3 ...
##  $ Type_of_order              : Factor w/ 5 levels "","Buffet ","Drinks ",..: 5 5 3 2 5 2 4 4 2 5 ..
##  $ Type_of_vehicle            : Factor w/ 5 levels "","bicycle ",..: 4 5 4 4 5 4 5 4 4 4 ...
##  $ temperature                : num  17.1 19.5 20.4 23.9 26.6 ...
##  $ humidity                   : int  77 93 91 78 87 65 69 82 65 77 ...
##  $ precipitation              : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ weather_description        : Factor w/ 12 levels "","broken clouds",..: 6 8 8 8 8 2 3 11 2 3 ...
##  $ X                          : logi  NA NA NA NA NA NA ...
##  $ Traffic_Level              : Factor w/ 7 levels "","High","Low",..: 3 6 3 4 2 4 2 6 6 2 ...
##  $ Distance..km.              : chr  "" "37.17" "3.34" "10.05" ...
##  $ TARGET                     : chr  "21.66666667" "85.26666667" "28.58333333" "35.18333333" ...
```

Now the respective variables have their types as `Factor`.

Numerical columns

Most numerical columns seem to be correct except for the TARGET column and the 'Distance..km' columns. These should be numerical columns so they will be converted into the right type:

```
data <- data %>%
  mutate(Distance..km. = as.numeric(Distance..km.),
         TARGET = as.numeric(TARGET))
```

```
## Warning: There were 2 warnings in `mutate()`.
## The first warning was:
## i In argument: `Distance..km. = as.numeric(Distance..km.)`.
## Caused by warning:
## ! NAs introduced by coercion
## i Run `dplyr::last_dplyr_warnings()` to see the 1 remaining warning.
```

The conversion was done, but NAs were introduced by coercion (this means that those values that couldn't be converted into a numeric type by R were assinged the `NA` value).

Checking to see how much of these values couldn't be converted:

```
sum(is.na(data$Distance..km.))
```

```
## [1] 921
```

```
sum(is.na(data$TARGET))
```

```
## [1] 961
```

There were 921 values in the `Distance..km.` column who were assigned `NA` while 961 were assigned `NA` in the `TARGET` column.

These amounts seem to be very close to each other, so maybe the `NA` values in `Distance..km.` happen to also affect into an `NA` in `TARGET`?

```
data %>%
  filter(is.na(Distance..km.) & is.na(TARGET)) %>%
  head(.) %>%
  select(Delivery_person_ID, Distance..km., TARGET)
```

```
##   Delivery_person_ID Distance..km. TARGET
## 1   RANCHIRES02DEL01            NA     NA
## 2     AURGRES20DEL03            NA     NA
## 3      VADRES02DEL02            NA     NA
## 4      VADRES04DEL03            NA     NA
## 5      VADRES16DEL02            NA     NA
## 6      VADRES09DEL01            NA     NA
```

All values shown that have NA in distance also have NA in TARGET. To see if the result is a value close to 921 (the amount of total NA values in Distance), the amount of rows that have NA in both distance and target will be calculated:

```
data %>%
  filter(is.na(Distance..km.) & is.na(TARGET)) %>%
  summarize(nrow(.))
```

```
##   nrow(.)
## 1     916
```

These values are not useful towards the model later on because these are values that cannot be predicted at all (the target variable has no value). As a result, these values will be removed from the dataset:

```
data <- data %>%
  filter( !(is.na(Distance..km.) & is.na(TARGET)) )
```

Now, re-calculating the count of na values:

```
sum(is.na(data$Distance..km.))
```

```
## [1] 5
```

```r
sum(is.na(data$TARGET))
```

```
## [1] 45
```

The counts of NA values decreased by a big margin, leaving only 5 and 45 in `Distance..km.` and `TARGET` respectively.

Analyzing the remaining `Distance..km.` NA values:

```r
data %>%
  filter(is.na(Distance..km.))
```

```
##     ID Delivery_person_ID Delivery_person_Age Delivery_person_Ratings
## 1 4607     INDORES13DEL02                  37                     4.9
## 2 B4D6   RANCHIRES13DEL02                  24                     4.9
## 3 B473    COIMBRES02DEL03                  28                     4.4
## 4 5802    COIMBRES13DEL02                  30                     3.6
## 5  802      MUMRES18DEL03                  37                     4.6
##   Restaurant_latitude Restaurant_longitude Delivery_location_latitude
## 1            22.74505             75.89247                   22.76505
## 2            23.37499             85.33549                   23.42499
## 3            11.02248             76.99567                   11.11248
## 4            11.00367             76.97649                   11.08367
## 5            19.10930             72.82545                   19.19930
##   Delivery_location_longitude Type_of_order Type_of_vehicle temperature
## 1                    75.91247        Snack      motorcycle       17.11
## 2                    85.38549       Drinks      motorcycle       17.84
## 3                    77.08567       Drinks      motorcycle       25.12
## 4                    77.05649         Meal      motorcycle       25.97
## 5                    72.91545         Meal      motorcycle       26.96
##   humidity precipitation weather_description  X Traffic_Level Distance..km.
## 1       77             0                haze NA           Low            NA
## 2       68             0           clear sky NA          High            NA
## 3       73             0                mist NA          High            NA
## 4       73             0                haze NA          High            NA
## 5       57             0               smoke NA     Very High            NA
##     TARGET
## 1 21.66667
## 2 43.18333
## 3 45.85000
## 4 32.73333
## 5 70.33333
```

It seems that all of the other needed values are still there for these rows, so I will use the restaurant coordinates vs. the delivery location coordinates to fill the distance: Patterson (2019) Hijmans (2024)

```r
if(!require(geosphere)) install.packages("geosphere", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: geosphere
```

```
## Warning: package 'geosphere' was built under R version 4.3.3
```

```r
library(geosphere)

new_distances <- data %>%
  filter(is.na(Distance..km.)) %>%
  rowwise() %>%
  mutate(new_distance = distHaversine(c(Restaurant_longitude, Restaurant_latitude),
            c(Delivery_location_longitude, Delivery_location_latitude)) / 1000) %>%
  head(.) %>%
  select(Distance..km.,
         new_distance,
         Restaurant_latitude,
         Restaurant_longitude,
         Delivery_location_latitude,
         Delivery_location_longitude,
         TARGET) %>%
  pull(new_distance)

data[is.na(data$Distance..km.), "Distance..km."] <- new_distances

sum(is.na(data$Distance..km.))
```

```
## [1] 0
```

```r
rm(new_distances)
```

It can be seen that now there's no more NA values in the Distance..km. column. Now analyzing the leftover `TARGET` `NA` values:

```r
sum(is.na(data$TARGET))
```

```
## [1] 45
```

```r
head(data[is.na(data$TARGET),])
```

```
##        ID Delivery_person_ID Delivery_person_Age Delivery_person_Ratings
## 4531 60C2      HYDRES01DEL03                  26                     4.9
## 5436 2ECE      SURRES11DEL01                  24                     4.5
## 6352 3D19      SURRES13DEL01                  34                     3.6
## 6468 764E    CHENRES010DEL02                  21                     4.9
## 6659 D847      KOCRES20DEL02                  30                     4.5
## 6672 10F0     CHENRES01DEL01                  35                     4.6
##      Restaurant_latitude Restaurant_longitude Delivery_location_latitude
## 4531           17.410371             78.43722                  17.440371
## 5436           21.157735             72.76878                  21.267735
## 6352           21.170096             72.78912                  21.240096
## 6468           13.066762             80.25186                  13.196762
## 6659            9.979186             76.31736                   9.999186
## 6672           13.005801             80.25074                  13.115801
##      Delivery_location_longitude Type_of_order   Type_of_vehicle temperature
## 4531                    78.46722       Drinks          scooter          NA
## 5436                    72.87878       Drinks       motorcycle          NA
```

```
## 6352                      72.85912       Drinks  electric_scooter           NA
## 6468                      80.38187        Snack  electric_scooter        28.35
## 6659                      76.33736       Drinks  electric_scooter        25.68
## 6672                      80.36074       Drinks        motorcycle        28.40
##      humidity precipitation weather_description  X Traffic_Level Distance..km.
## 4531       NA            NA                     NA      Moderate          8.51
## 5436       NA            NA                     NA          High         22.71
## 6352       NA            NA                     NA          High         13.50
## 6468       76          0.17          light rain NA     Very High         28.92
## 6659       83          0.45          light rain NA           Low          4.47
## 6672       82          0.15          light rain NA     Very High         26.06
##      TARGET
## 4531     NA
## 5436     NA
## 6352     NA
## 6468     NA
## 6659     NA
## 6672     NA
```

The amount of rows that will be in the na-cleaned dataset is:

```
nrow(data) - sum(is.na(data$TARGET))
```

```
## [1] 9040
```

There are 45 missing values in the target variable, because the goal of this study is to train the algorithm on the most accurate data available, then the best option for these NA values is to delete them, since filling them in any way can introduce an incorrect bias that could prevent the model from learning correct patterns from the data.

```
data <- data[!is.na(data$TARGET),]

nrow(data)
```

```
## [1] 9040
```

The resulting row count after deleting the rest of `NA` values in the dataset is 9040.

Variable names In the provided dataset there's no standard naming convention for the columns, some start with a capital letter while others don't and some are in all-caps (TARGET and ID) while others aren't.

In order to fix this, all columns will be renamed to non-caps format.

```
colnames(data) <- c("id", "delivery_person_id", "delivery_person_age", "delivery_person_ratings",
  "restaurant_latitude", "restaurant_longitude", "delivery_loc_latitude",
  "delivery_loc_longitude", "order_type", "vehicle_type", "temperature", "humidity",
  "precipitation", "weather_type", "X", "traffic_level", "distance", "delivery_time_min")

str(data)
```

```
## 'data.frame':    9040 obs. of  18 variables:
##  $ id                     : chr  "4607" "B379" "5D6D" "7A6A" ...
##  $ delivery_person_id     : chr  "INDORES13DEL02" "BANGRES18DEL02" "BANGRES19DEL01" "COIMBRES13DEL02
```

```
##  $ delivery_person_age    : int  37 34 23 38 32 22 33 35 22 36 ...
##  $ delivery_person_ratings: num  4.9 4.5 4.4 4.7 4.6 4.8 4.7 4.6 4.8 4.2 ...
##  $ restaurant_latitude    : num  22.7 12.9 12.9 11 13 ...
##  $ restaurant_longitude   : num  75.9 77.7 77.7 77 80.2 ...
##  $ delivery_loc_latitude  : num  22.8 13 12.9 11.1 13 ...
##  $ delivery_loc_longitude : num  75.9 77.8 77.7 77 80.3 ...
##  $ order_type             : Factor w/ 5 levels "","Buffet ","Drinks ",..: 5 5 3 2 5 2 4 4 2 5 ...
##  $ vehicle_type           : Factor w/ 5 levels "","bicycle ",..: 4 5 4 4 5 4 5 4 4 4 ...
##  $ temperature            : num  17.1 19.5 20.4 23.9 26.6 ...
##  $ humidity               : int  77 93 91 78 87 65 69 82 65 77 ...
##  $ precipitation          : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ weather_type           : Factor w/ 12 levels "","broken clouds",..: 6 8 8 8 8 2 3 11 2 3 ...
##  $ X                      : logi  NA NA NA NA NA NA ...
##  $ traffic_level          : Factor w/ 7 levels "","High","Low",..: 3 6 3 4 2 4 2 6 6 2 ...
##  $ distance               : num  3.03 37.17 3.34 10.05 9.89 ...
##  $ delivery_time_min      : num  21.7 85.3 28.6 35.2 43.5 ...
```

Now there's a more uniform column name standard.

Per-Column Analysis

ID Variable

Checking the amount of distinct values in `id` vs. the amount of rows in the dataset:

```
n_distinct(data$id)
```

```
## [1] 9037
```

```
nrow(data)
```

```
## [1] 9040
```

Interistingly, there are less distinct ID values (9037) than rows (9040).

Checking the occurrences of duplicate `id` entries:

```
data %>%
  group_by(id) %>%
  summarize(appearances = n()) %>%
  arrange(desc(appearances))
```

```
## # A tibble: 9,037 x 2
##    id       appearances
##    <chr>          <int>
##  1 6.00E+02           2
##  2 6.00E+03           2
##  3 9.00E+02           2
##  4 09-Dec             1
##  5 1.00E+03           1
##  6 1.00E+12           1
```

```
##  7 1.00E+13           1
##  8 1.00E+20           1
##  9 1.00E+21           1
## 10 1.00E+23           1
## # i 9,027 more rows
```

```
data %>%
  filter(id == "6.00E+02")
```

```
##         id delivery_person_id delivery_person_age delivery_person_ratings
## 1 6.00E+02      VADRES06DEL01                  29                     4.6
## 2 6.00E+02      MUMRES01DEL01                  22                     4.8
##   restaurant_latitude restaurant_longitude delivery_loc_latitude
## 1            22.31279             73.17028              22.42279
## 2            19.12663             72.82998              19.13663
##   delivery_loc_longitude order_type      vehicle_type temperature humidity
## 1               73.28028      Snack  electric_scooter       22.47       44
## 2               72.83998      Snack           scooter       27.92       57
##   precipitation weather_type  X traffic_level distance delivery_time_min
## 1             0    clear sky NA          High    20.29          44.28333
## 2             0        smoke NA      Very Low     2.05          19.68333
```

There don't seem to be any abnormal values, the only remarkable detail is that the coordinates of both the restaurant and the delivery destination are similar between entries.

Looking at another `id` that appears more than once:

```
data %>%
  filter(id == "6.00E+03")
```

```
##         id delivery_person_id delivery_person_age delivery_person_ratings
## 1 6.00E+03      MUMRES05DEL01                  37                     4.4
## 2 6.00E+03     PUNERES13DEL02                  36                     4.6
##   restaurant_latitude restaurant_longitude delivery_loc_latitude
## 1            18.92758             72.83258              18.96758
## 2            18.56245             73.91662              18.69245
##   delivery_loc_longitude order_type vehicle_type temperature humidity
## 1               72.87259       Meal   motorcycle       27.05       44
## 2               74.04662      Snack   motorcycle       23.60       46
##   precipitation weather_type  X traffic_level distance delivery_time_min
## 1             0        smoke NA      Moderate     7.77          26.63333
## 2             0    clear sky NA     Very High    29.32          70.20000
```

Again the coordinates are similar, but that's about it.

Looking at yet another repeated `id`.

```
data %>%
  filter(id == "9.00E+02")
```

```
##         id delivery_person_id delivery_person_age delivery_person_ratings
## 1 9.00E+02   RANCHIRES04DEL01                  32                     4.6
## 2 9.00E+02     INDORES14DEL03                  28                     4.0
```

```
##   restaurant_latitude restaurant_longitude delivery_loc_latitude
## 1            23.35903             85.32535              23.39903
## 2            22.76159             75.88636              22.82159
##   delivery_loc_longitude order_type vehicle_type temperature humidity
## 1               85.36535     Snack   motorcycle       17.67       68
## 2               75.94636     Snack   motorcycle       17.14       72
##   precipitation weather_type  X traffic_level distance delivery_time_min
## 1             0    clear sky NA           Low     7.87          24.70000
## 2             0         haze NA      Moderate    11.75          31.36667
```

In here the longitudes between rows vary greatly, but the latitudes do kind of match. In these code blocks the ages of the delivery people differ, so the 'id' itself isn't a unique person identifier.

```
sum(is.na(data$id))
```

```
## [1] 0
```

There are no null values in `id`.

Delivery Person ID

Looking at the identifiers of each delivery person now:

```
head(data$delivery_person_id, n=8)
```

```
## [1] "INDORES13DEL02"   "BANGRES18DEL02"   "BANGRES19DEL01"   "COIMBRES13DEL02"
## [5] "CHENRES12DEL01"   "HYDRES09DEL03"    "RANCHIRES15DEL01" "MYSRES15DEL02"
```

```
n_distinct(data$delivery_person_id)
```

```
## [1] 1134
```

In this dataset there's only 1134 different delivery people, meaning that the dataset has multiple deliveries from a single person.

```
sum(is.na(data$delivery_person_id))
```

```
## [1] 0
```

There are no null values in `delivery_person_id`.

Delivery Person Age

Looking at the age of the delivery people now:

```
head(data$delivery_person_age, n=8)
```

```
## [1] 37 34 23 38 32 22 33 35
```

```
min(data$delivery_person_age)
```

```
## [1] 15
```

```
max(data$delivery_person_age)
```

```
## [1] 50
```

The ages of the delivery people range from 15 to 50.

```
data %>%
  filter(delivery_person_age == 15)
```

```
##    id delivery_person_id delivery_person_age delivery_person_ratings
## 1 CD0    INDORES010DEL03                  15                       1
## 2 91A      SURRES17DEL03                  15                       1
## 3 474     CHENRES15DEL03                  15                       1
## 4 73F     BANGRES05DEL01                  15                       1
##   restaurant_latitude restaurant_longitude delivery_loc_latitude
## 1            22.75004             75.90285              22.81004
## 2            21.14957             72.77270              21.20957
## 3            13.02629             80.27523              13.05629
## 4            12.97032             77.64575              13.08032
##   delivery_loc_longitude order_type vehicle_type temperature humidity
## 1               75.96285     Snack      scooter       17.19       72
## 2               72.83270    Buffet      bicycle       23.23       43
## 3               80.30523    Drinks      bicycle       28.56       81
## 4               77.75575    Buffet   motorcycle       22.33       73
##   precipitation weather_type  X traffic_level distance delivery_time_min
## 1             0         haze NA      Moderate    13.40          35.23333
## 2             0    clear sky NA      Moderate    12.53          25.36667
## 3             0         mist NA           Low     4.84          27.03333
## 4             0         haze NA          High    20.46          47.01667
```

There's 4 entries of a 15-year-old doing deliveries, each with a different `delivery_person_id`.

```
data %>%
  filter(delivery_person_age == 50)
```

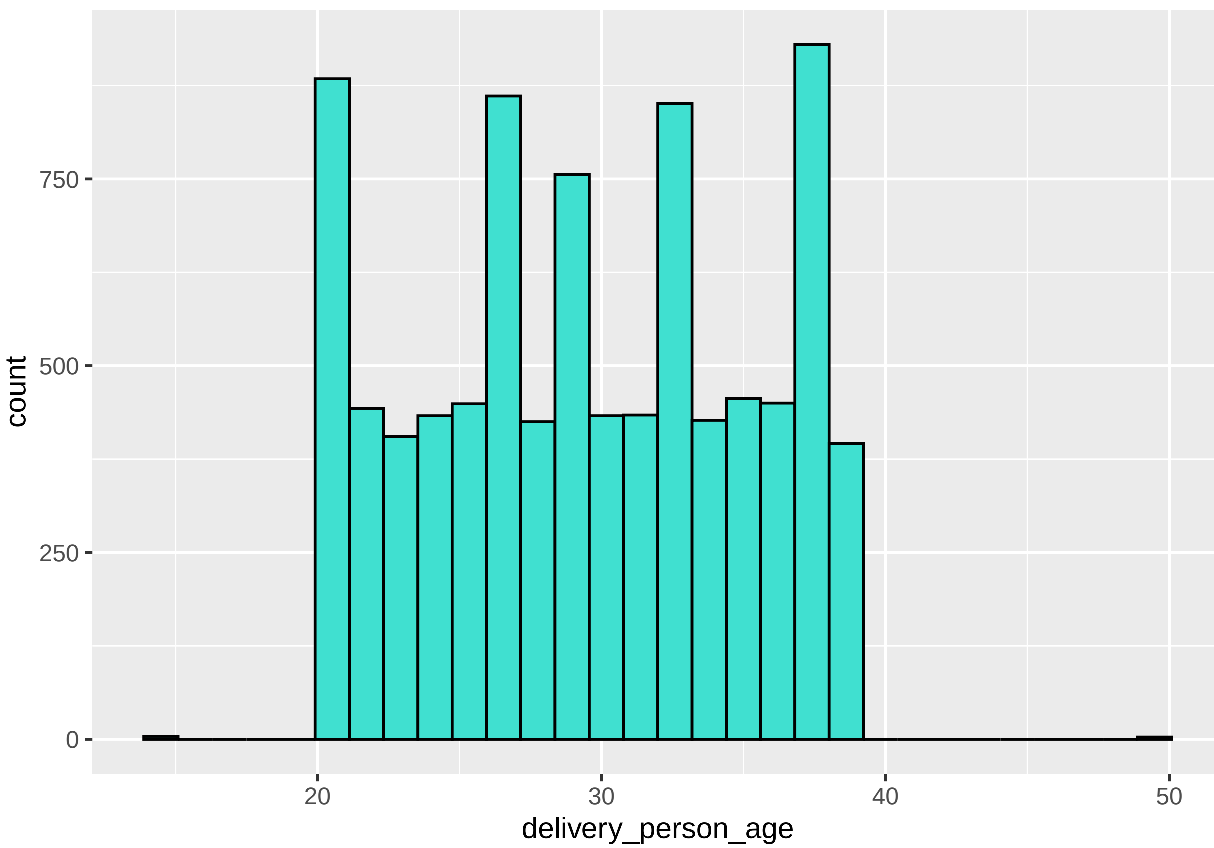
```
##    id delivery_person_id delivery_person_age delivery_person_ratings
## 1 430     BANGRES19DEL01                  50                       6
## 2 427      JAPRES06DEL02                  50                       6
## 3 3F0    BANGRES010DEL01                  50                       6
##   restaurant_latitude restaurant_longitude delivery_loc_latitude
## 1            12.91426             77.67840              13.02426
## 2            26.91193             75.79728              27.04193
## 3            12.93330             77.61429              13.00330
##   delivery_loc_longitude order_type      vehicle_type temperature humidity
## 1               77.78840      Meal  electric_scooter        22.86       74
## 2               75.92728      Meal  electric_scooter        23.71       29
```

```
## 3              77.68429    Drinks           scooter        22.89       73
##   precipitation weather_type  X traffic_level distance delivery_time_min
## 1             0         haze NA     Very High    27.03          55.80000
## 2             0    clear sky NA     Very High    28.81          59.68333
## 3             0         haze NA          High    14.22          46.86667
```

There are also 3 entries of a 50 year old delivering, with 2 different `delivery_person_id` values.

```r
data %>%
  ggplot(aes(delivery_person_age)) +
  geom_histogram(col = "black", fill = "turquoise")
```

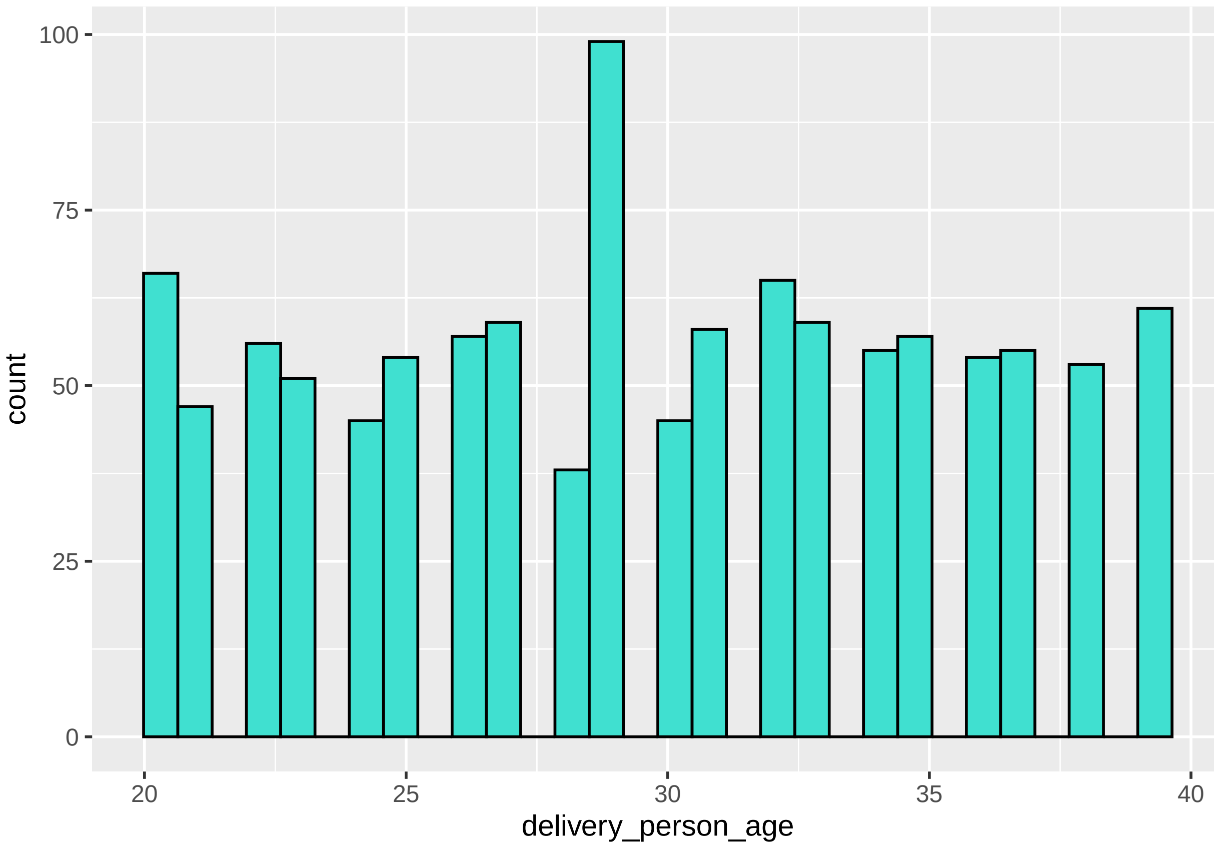
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



Most ages range from 20 to 40, with 15 and the 50 being the outliers. However, this graph can be wrong due to the fact that takes multiple deliveries as a different count for the age.

```r
data %>%
  distinct(delivery_person_id, .keep_all = TRUE) %>% # grabbing different people or else multiple deliv
  ggplot(aes(delivery_person_age)) +
  geom_histogram(col = "black", fill = "turquoise")
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



With this graph, the scale changes and now the 15 and 50 values don't appear. Could this be due to the fact that the same `delivery_person_id` has multiple ages assigned to it (in this case this would be human error)?

```
data %>%
  filter(delivery_person_id == "JAPRES15DEL03") %>%
  select(delivery_person_age)
```

```
##   delivery_person_age
## 1                  36
## 2                  21
## 3                  37
## 4                  26
## 5                  27
## 6                  30
## 7                  23
```

This is a big issue, there are multiple ages assigned to the same `delivery_person_id` (JAPRES15DEL03). Is this a common mistake? Does this happen for each `delivery_person_id`?

```
data %>%
  group_by(delivery_person_id) %>%
  distinct(delivery_person_age) %>%
  summarize(distinct_ages = n())
```

```
## # A tibble: 1,134 x 2
```

```
##    delivery_person_id distinct_ages
##    <chr>                      <int>
##  1 AGRRES010DEL01                 2
##  2 AGRRES010DEL02                 2
##  3 AGRRES010DEL03                 2
##  4 AGRRES01DEL01                  2
##  5 AGRRES01DEL02                  3
##  6 AGRRES01DEL03                  4
##  7 AGRRES03DEL01                  4
##  8 AGRRES03DEL02                  2
##  9 AGRRES03DEL03                  2
## 10 AGRRES04DEL01                  3
## # i 1,124 more rows
```

This is indeed a common mistake, since the docs for this dataset mention:

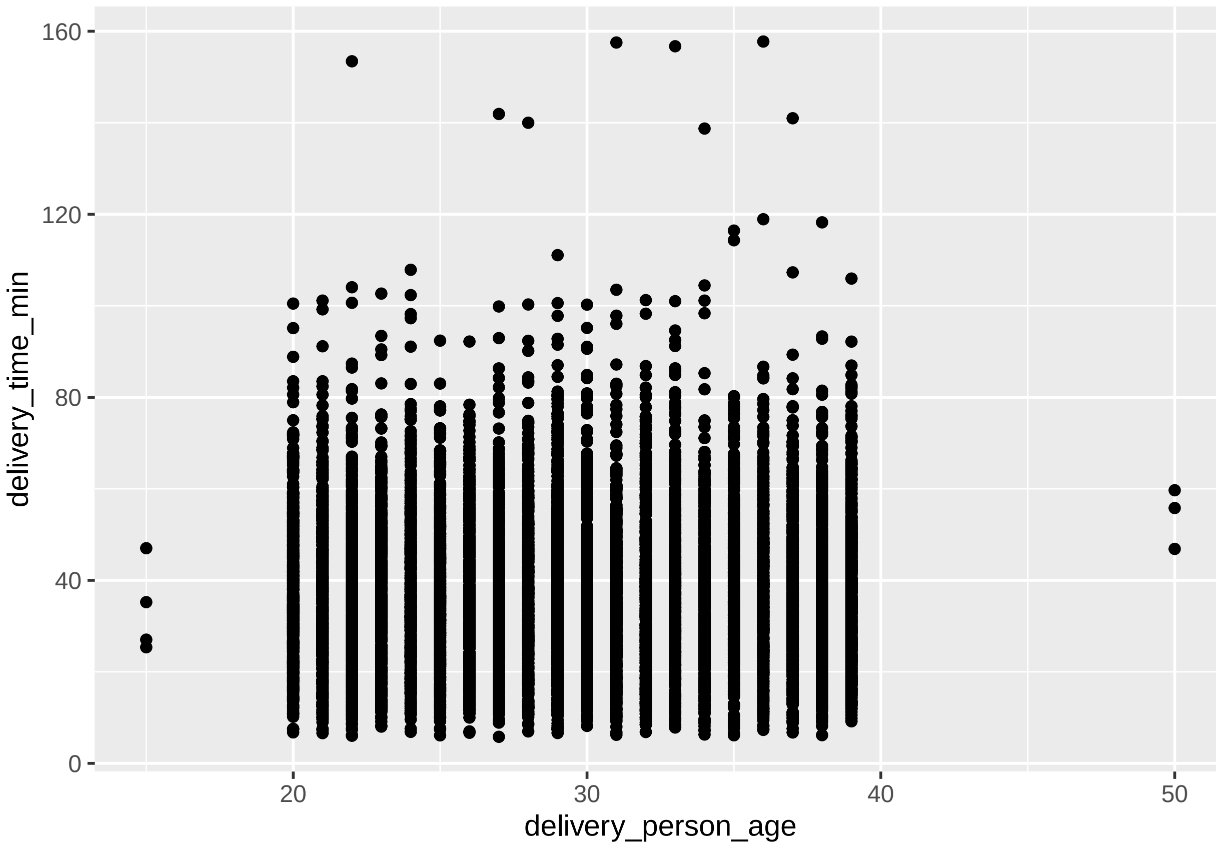
- "**Delivery_person_ID:** A unique identifier assigned to each delivery person for tracking purposes."

So, the Delivery_person_ID isn't a reliable identifier for a singular person. Therefore, an accurate count of each age's different workers (delivery person) cannot be obtained.

A possible cause for this is that the program made to assign `delivery_person_id`'s to workers isn't meant to generate one per worker, instead it could've been made to generate one per delivery in real-time, and when one `delivery_person_id` is no longer in use, then it could re-assign it to another delivery, hence another possible driver.

Checking to see if the driver's age has something to do with the delivery time:

```
data %>%
  ggplot(aes(delivery_person_age, delivery_time_min)) +
  geom_point(col = "black")
```



There's no clear effect of the age on the delivery time.

Delivery Person Ratings

```
min(data$delivery_person_ratings)
```

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

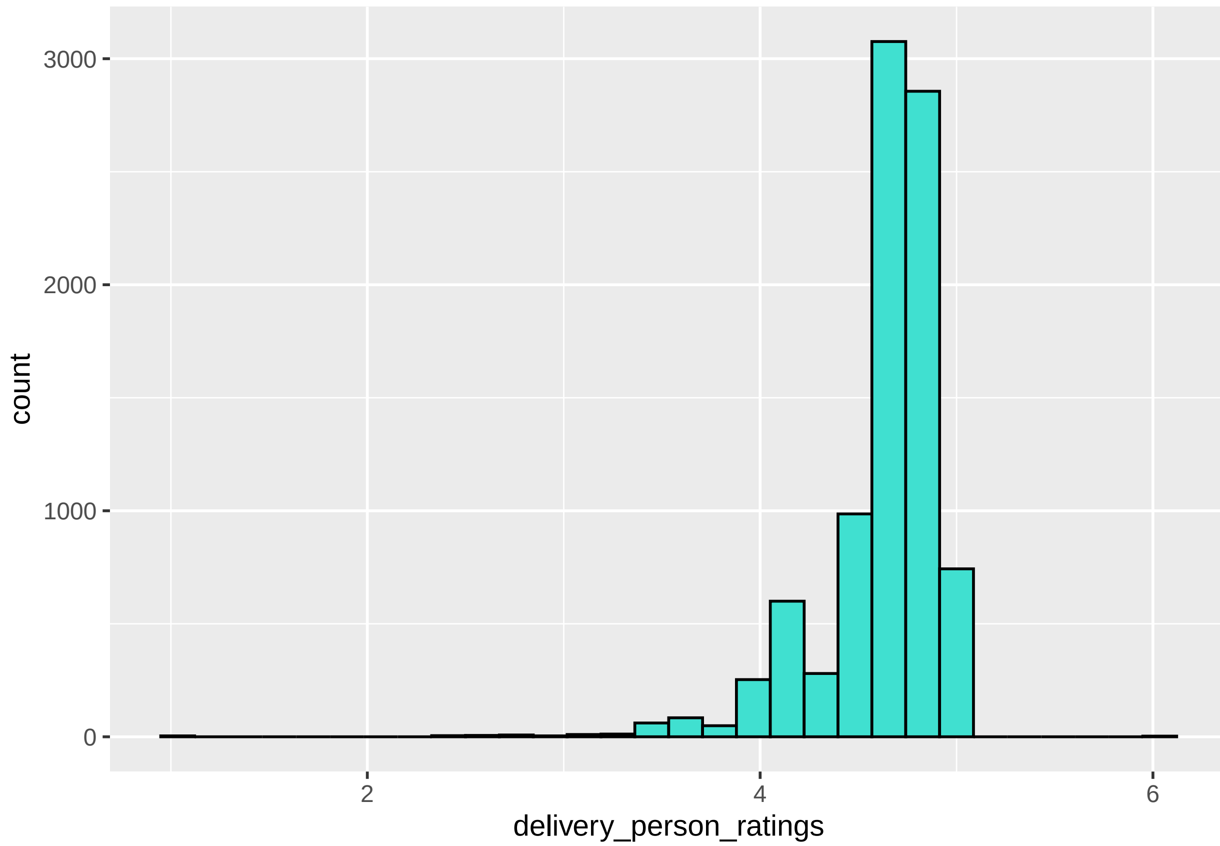
```
max(data$delivery_person_ratings)
```

```
## [1] 6
```

The rating scale is from 1 to 6, inclusive. This rating scale seems to deviate from 'normal' scales in the sense that most delivery apps (or other apps in genreal) have a 1-5 scale or a 1-10 scale, but a 1-6 scale is very uncommon.

```
data %>%
  ggplot(aes(delivery_person_ratings)) +
  geom_histogram(col = "black", fill = "turquoise")
```

```
## 'stat_bin()' using 'bins = 30'. Pick better value with 'binwidth'.
```



Most delivery people have a rating between 3 and 5. With the other ratings having very low counts.

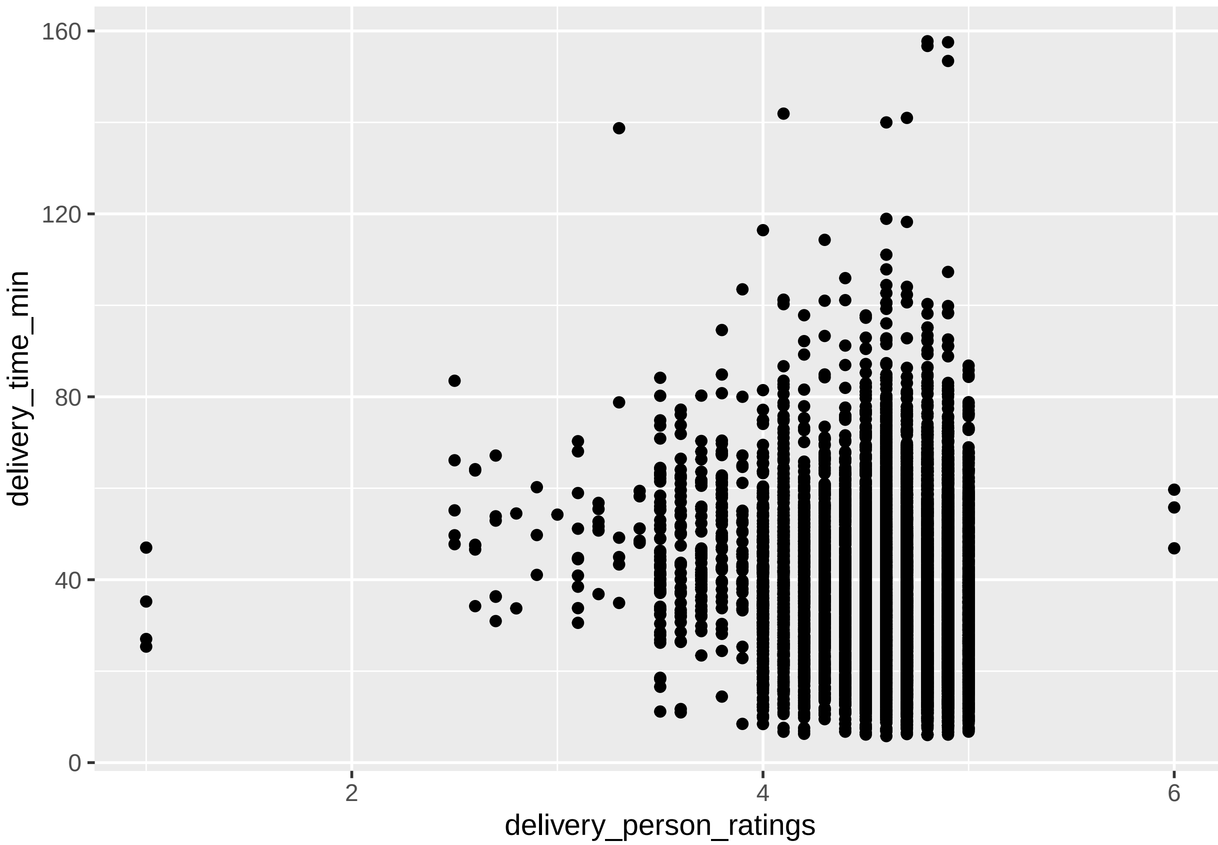
```
data %>%
  filter(delivery_person_ratings > 5)
```

```
##    id delivery_person_id delivery_person_age delivery_person_ratings
## 1 430     BANGRES19DEL01                  50                       6
## 2 427      JAPRES06DEL02                  50                       6
## 3 3F0    BANGRES010DEL01                  50                       6
##   restaurant_latitude restaurant_longitude delivery_loc_latitude
## 1            12.91426             77.67840              13.02426
## 2            26.91193             75.79728              27.04193
## 3            12.93330             77.61429              13.00330
##   delivery_loc_longitude order_type      vehicle_type temperature humidity
## 1               77.78840      Meal  electric_scooter       22.86       74
## 2               75.92728      Meal  electric_scooter       23.71       29
## 3               77.68429    Drinks           scooter       22.89       73
##   precipitation weather_type  X traffic_level distance delivery_time_min
## 1             0         haze NA     Very High    27.03          55.80000
## 2             0    clear sky NA     Very High    28.81          59.68333
## 3             0         haze NA          High    14.22          46.86667
```

An interesting detail is that the only delivery people who have a rating bigger than 5 (6 rating) are 50 year-olds. However, due to the recent findings stating that the `delivery_person_id` variable cannot correctly identify singular people, this is most likely a single person that is 50 years old that somehow has this 6 rating. So again, an accurate count of the number of distinct people (in order to count the amount of people per rating) cannot be obtained.

Looking at the possible effect of ratings on delivery time:

```
data %>%
  ggplot(aes(x = delivery_person_ratings, y = delivery_time_min)) +
  geom_point(col = "black")
```



The rating itself doesn't seem to have an effect on delivery time since all ratings deliver on very similar timeframes.

Restaurant Latitude and Longitude

Latitude and longitude values can be very raw, they just specify coordinates on a map. However, if a latitude and longitude pairing appears several times in a repeated manner, this means that there are several orders for the same restaurant (since the latitude and longitude pairing uniquely identifies the position of a specific restaurant). This technically assumes that no restaurant can be on top of another (this is very common in malls for example), so this is only to get an idea on the amount of deliveries that possibly come from the same restaurant.

This will be studied further to see if most people order from the same restaurant and therefore that restaurant needs more time than others in order to prepare the food.

```
options(pillar.sigfig = 6)

data %>%
  group_by(restaurant_latitude, restaurant_longitude) %>%
  summarize(order_count = n()) %>%
  arrange(desc(order_count))
```

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

## # A tibble: 388 x 3
## # Groups:   restaurant_latitude [388]
##    restaurant_latitude restaurant_longitude order_count
##                  <dbl>                <dbl>       <int>
##  1             11.0213              76.9950          54
##  2             26.8496              75.8005          48
##  3             11.0251              77.0154          46
##  4             13.0263              80.1746          46
##  5             18.5393              73.8979          46
##  6             23.3515              85.3243          46
##  7             26.9114              75.7890          46
##  8             26.9141              75.8057          46
##  9             17.4559              78.3755          45
## 10             21.1577              72.7687          45
## # i 378 more rows
```

54 orders seem to be from the same restaurant, 48 from another and so on. It is important to compare the average delivery time of this popular restaurant with one that has a low delivery count to see if these coordinate pairings introduce a bias on the target variable (delivery time).

```r
data <- data %>%
  mutate(restaurant_latitude = round(restaurant_latitude, 4),
         restaurant_longitude = round(restaurant_longitude, 4))

data %>%
  filter(restaurant_latitude == 11.0213 & restaurant_longitude == 76.995) %>%
  summarize(mean_delivery_time = mean(delivery_time_min))
```

```
##   mean_delivery_time
## 1           34.61142
```

The most sought-out restaurant has a mean delivery time of almost 35 minutes.

Making the same calculation for the restaurant with the lowest order count:

```r
data %>%
  group_by(restaurant_latitude, restaurant_longitude) %>%
  summarize(order_count = n()) %>%
  arrange(order_count)
```

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

## # A tibble: 388 x 3
## # Groups:   restaurant_latitude [377]
##    restaurant_latitude restaurant_longitude order_count
##                  <dbl>                <dbl>       <int>
##  1              9.9668               76.243           2
##  2              9.9828               76.2833          2
```

```
##   3              9.9598              76.2961           3
##   4             10.028               76.31             3
##   5             19.8753              75.3167           3
##   6             19.8759              75.3589           3
##   7             19.8803              75.3235           3
##   8             23.219               77.3736           3
##   9             26.47                80.35             3
##  10             26.4741              80.3481           3
## # i 378 more rows
```

```
data %>%
  filter(restaurant_latitude == 9.9668 & restaurant_longitude == 76.243) %>%
  summarize(mean_delivery_time = mean(delivery_time_min))
```

```
##   mean_delivery_time
## 1           35.94167
```

One of the least ordered restaurant has almost a 36 minute delivery time. The average time of both restaurants is extremely similar, indicating that there isn't a 'delay' effect by ordering from a 'popular' restaurant in this case.

Delivery Latitude and Longitude

Here we encounter the same effect on these values as before, these coordinate values are very raw so the amount of orders that came from the same home will be calculated. Again, this assumes that no home can be on top of another, and this is almost the normal way of living inside of capital cities, where renting an apartment is common.

```
data <- data %>%
  mutate(delivery_loc_latitude = round(delivery_loc_latitude, 4),
         delivery_loc_longitude = round(delivery_loc_longitude, 4))

data %>%
  group_by(delivery_loc_latitude, delivery_loc_longitude) %>%
  summarize(order_count = n()) %>%
  arrange(desc(order_count))
```

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

```
## # A tibble: 3,375 x 3
## # Groups:   delivery_loc_latitude [3,159]
##    delivery_loc_latitude delivery_loc_longitude order_count
##                    <dbl>                  <dbl>       <int>
##  1               13.0463                80.2952           9
##  2               13.0763                80.2246           9
##  3               21.2197                72.8426           9
##  4               21.2377                72.8487           9
##  5               12.3211                76.6649           8
##  6               13.023                 77.7932           8
##  7               13.0304                77.6905           8
```

```
##  8                13.043                 77.8132           8
##  9                13.0558                80.3007           8
## 10                18.6439                73.9954           8
## # i 3,365 more rows
```

There are several 'homes' that ordered 9 times in this data.

Order Type

Taking a glance at only the `order_type` column:

```r
head(data$order_type, n=8)
```

```
## [1] Snack   Snack   Drinks  Buffet  Snack   Buffet  Meal    Meal
## Levels:  Buffet  Drinks  Meal  Snack
```

It can be seen that this majorly describes what the order is mostly comprised of (drinks, meal, etc.).

```r
data %>%
  group_by(order_type) %>%
  summarize(appearances = n(), avg_delivery_time = mean(delivery_time_min)) %>%
  arrange(desc(appearances))
```

```
## # A tibble: 4 x 3
##   order_type appearances avg_delivery_time
##   <fct>            <int>             <dbl>
## 1 "Snack "          2310           38.0492
## 2 "Meal "           2281           33.9035
## 3 "Drinks "         2276           43.1350
## 4 "Buffet "         2173           35.4414
```

Thanks to the output, it can be seen that the most common order type is the snack, followed by meal, drinks and buffet being the last one.

It can be also inferred that ordering drinks usually takes longer than ordering meals, this could be due to the fact that these could be low-income orders, so restaurant managers could give a higher priority to meals since these usually cost more than just drinks. So the order type does matter when predicting the average delivery time.

Vehicle Type

Looking at the `vehicle_type` column:

```r
head(data$vehicle_type, n=8)
```

```
## [1] motorcycle  scooter     motorcycle  motorcycle  scooter     motorcycle
## [7] scooter     motorcycle
## Levels:  bicycle  electric_scooter  motorcycle  scooter
```

These values are very self-explanatory, describing the transportation method of the delivery person.

```
data %>%
  group_by(vehicle_type) %>%
  summarize(appearances = n(), avg_delivery_time = mean(delivery_time_min)) %>%
  arrange(desc(appearances))
```

```
## # A tibble: 4 x 3
##   vehicle_type        appearances avg_delivery_time
##   <fct>                     <int>             <dbl>
## 1 "motorcycle "              5350           37.6645
## 2 "scooter "                 2971           37.8225
## 3 "electric_scooter "         709           36.9921
## 4 "bicycle "                   10           31.3983
```

Interestingly enough, bicycle deliveries are the ones that have the least amount of average delivery time out of the four. This could be due to bicycle riders not taking up on large rides (in the sense of accepting deliveries from users that are far away from the restaurant) due to the increased physicality needed. This added physicality can also explain the low amount of people that use it for delivery purposes. This will be calculated for verification:

```
data %>%
  group_by(vehicle_type) %>%
  summarize(appearances = n(),
            avg_delivery_time = mean(delivery_time_min),
            delivery_distance = mean(distance)) %>%
  arrange(desc(appearances))
```

```
## # A tibble: 4 x 4
##   vehicle_type        appearances avg_delivery_time delivery_distance
##   <fct>                     <int>             <dbl>             <dbl>
## 1 "motorcycle "              5350           37.6645           14.2520
## 2 "scooter "                 2971           37.8225           14.3859
## 3 "electric_scooter "         709           36.9921           13.9945
## 4 "bicycle "                   10           31.3983           12
```

As it was hinted at previously, the bicycle riders indeed have a lower average delivery distance when compared to motorcycle riders for example. An interesting remark are the scooter riders. While using a scooter does include physicality during work (but not as much as bicycles do), it still remains the second most-used vehicle type for food transportation.

All of this hints at the possibility of the `vehicle_type` having an impact on delivery time.

Temperature

Looking at the first 8 rows of the temperature column:

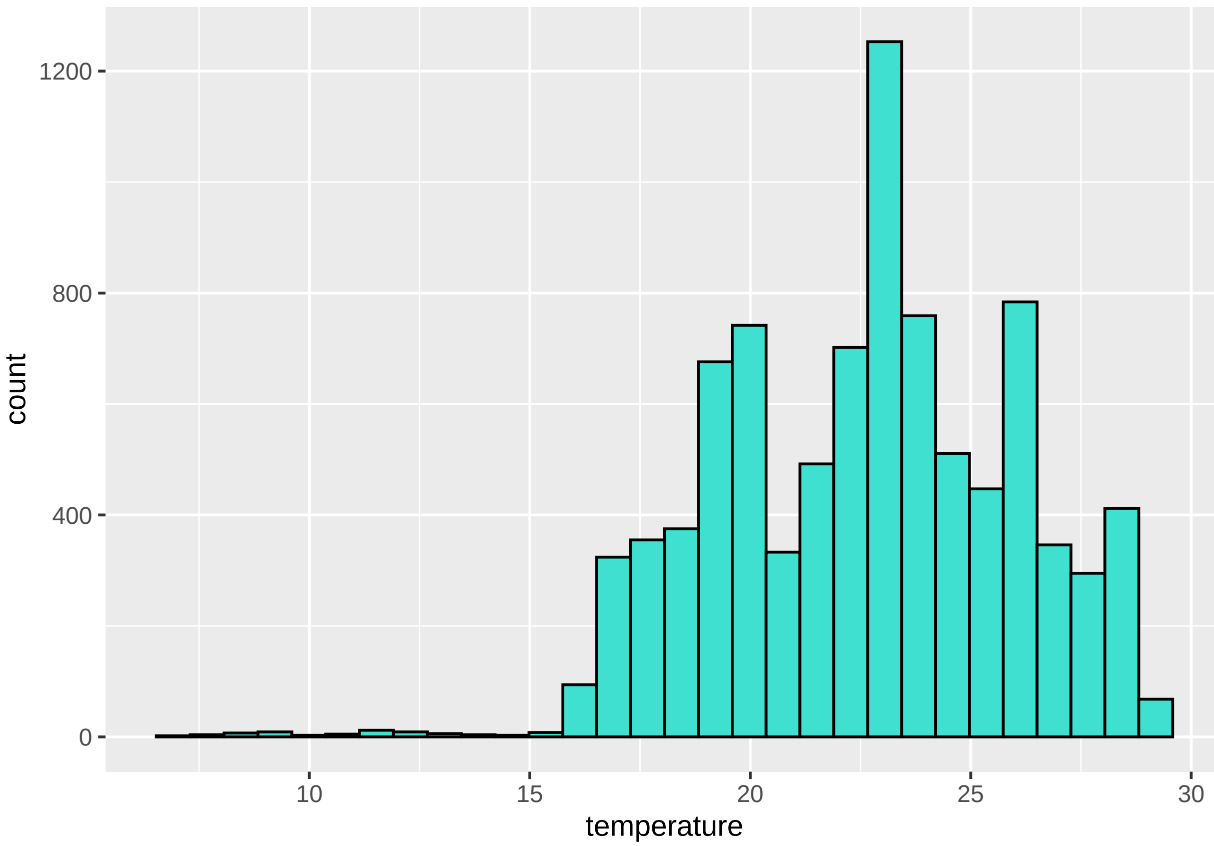
```
head(data$temperature, n=8)
```

```
## [1] 17.11 19.50 20.45 23.86 26.55 21.43 17.51 18.03
```

These temperatures seem to be in Celsius rather than Fahrenheit. This isn't clarified in the documentation for the dataset. However, this can be confirmed through the average temperatures of planet earth NASA Science (2023) .

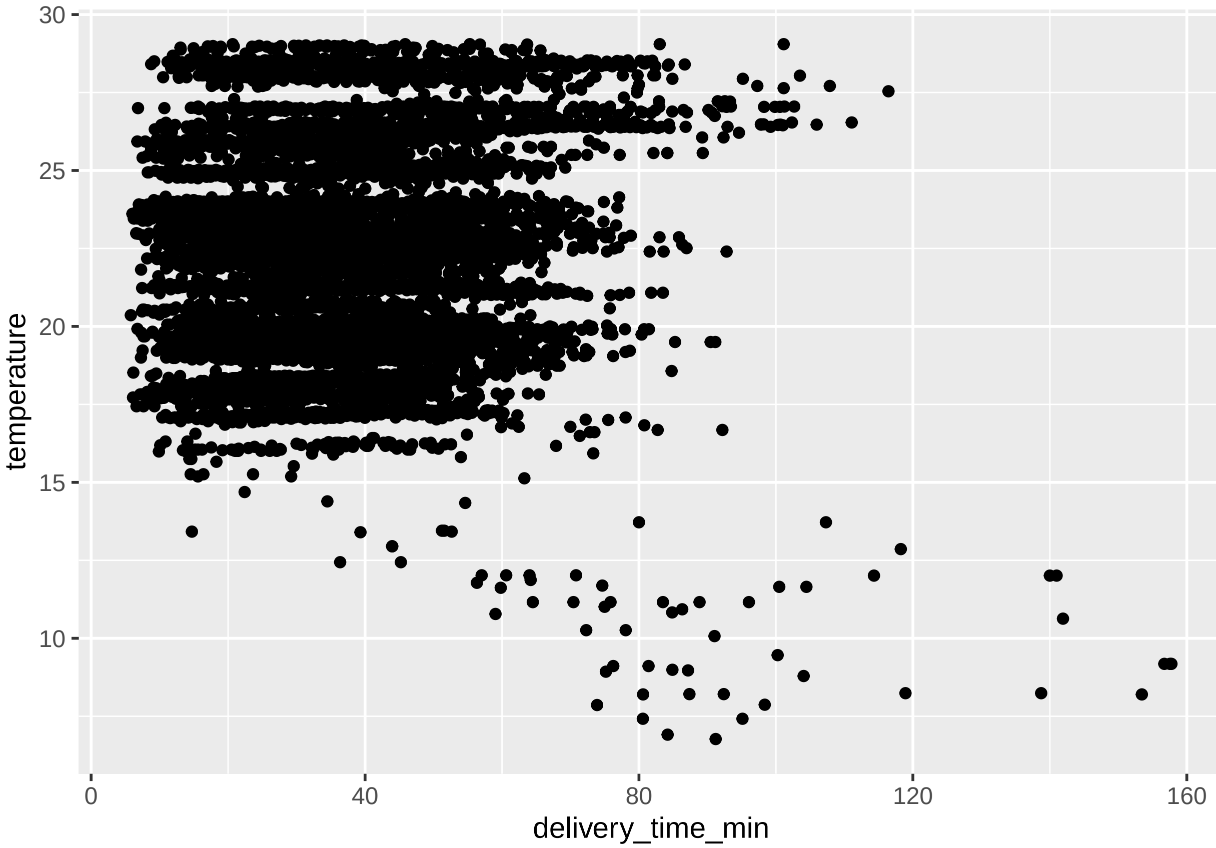
```r
data %>%
  ggplot(aes(temperature)) +
  geom_histogram(fill = "turquoise", col = "black")
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



Most temperatures seem to be around de 22.5 mark. Heat could introduce laziness and exhaustion for the delivery drivers, so it wouldn't be abnormal to see that higher temperatures lead to higher delivery times Belsky & Horowitz (2022) .

```r
data %>%
  ggplot(aes(x = delivery_time_min, y = temperature)) +
  geom_point(fill = "turquoise", col = "black")
```



It is surprising to see that the most heated deliveries are the ones who usually have lower delivery times. However, an argument can be made that states that this could be because people that live closer to these restaurants have a hotter climate than those who live further away. Things like the heat from gas car engines are prejudicial for the climate (therefore generating hotter temperatures).

So, the temperature does seem to correlate to the delivery time (although the effect is very low), but it does so in the contrary sense to the one thought before.

Humidity

Taking a look at the first 6 rows of `humidity`, as well as the minimum and maximum data values:

```
head(data$humidity)
```

```
## [1] 77 93 91 78 87 65
```

```
min(data$humidity)
```

```
## [1] 27
```

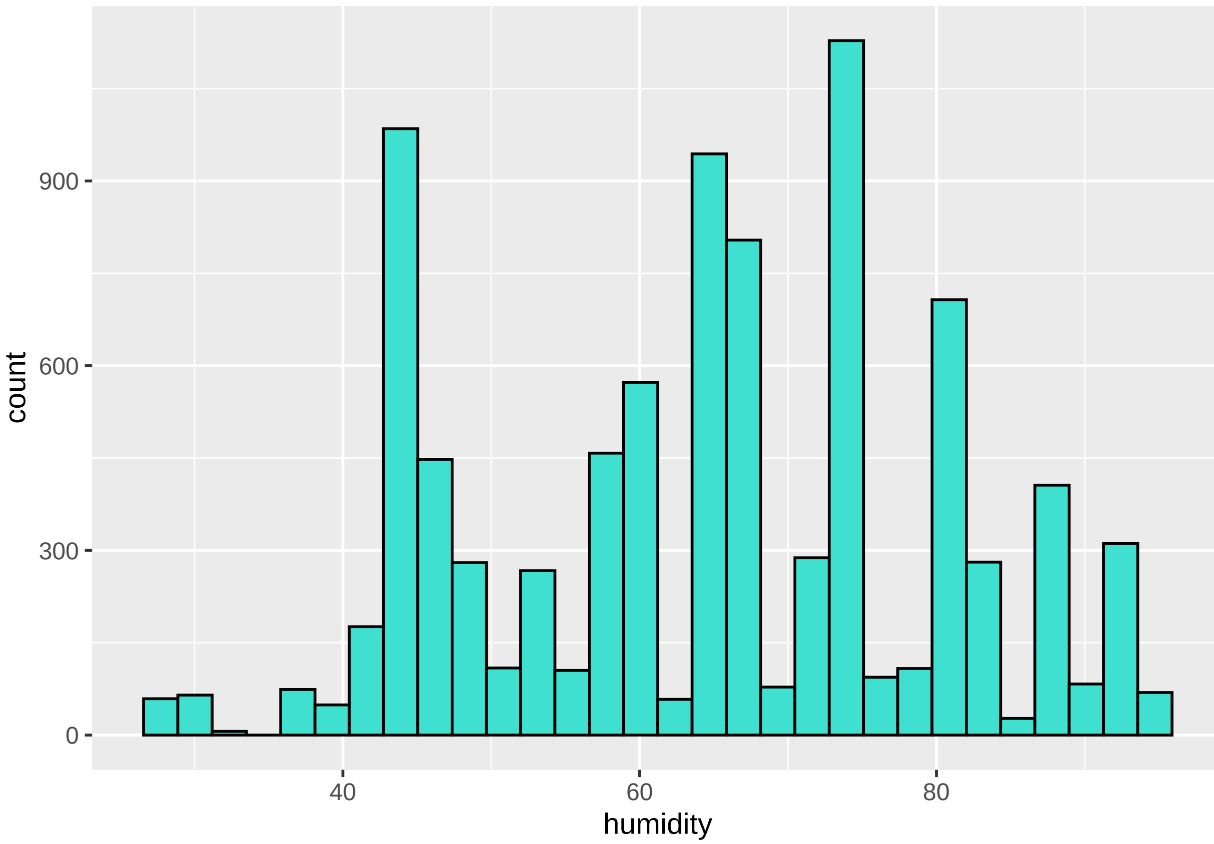
```
max(data$humidity)
```

```
## [1] 94
```

The values of the humidity seem to be percentages (this is the usual metric for the general population). This is another detail that isn't mentioned in the documentation of the dataset.

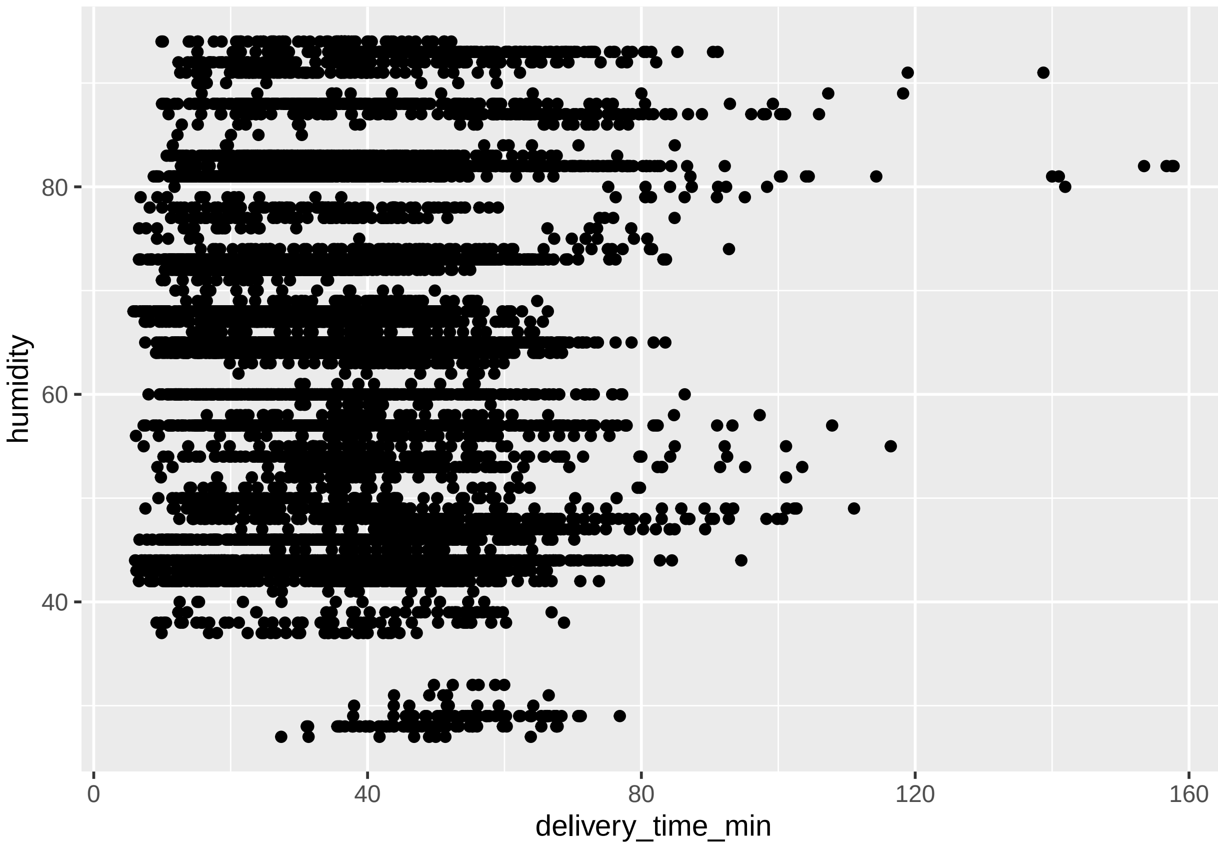
```
data %>%
  ggplot(aes(humidity)) +
  geom_histogram(fill = "turquoise", col = "black")
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



As per the histogram, it can be seen that the humidity values seem to be pretty scattered, there's no bell-like shaped curve present in the graph.

```
data %>%
  ggplot(aes(x = delivery_time_min, y = humidity)) +
  geom_point(col = "black")
```



No clear pattern can be extracted on the effect of humidity on delivery time since the values are very visually scattered.

Precipitation

Looking at the first values as well as the amount of distinct values in `precipitation`:

```
head(data$precipitation, n = 8)
```

```
## [1] 0 0 0 0 0 0 0 0
```

```
n_distinct(data$precipitation)
```

```
## [1] 5
```

There are 5 distinct values in all of the `precipitation` column, this means that this specific column doesn't have a binary behavior.

```
min(data$precipitation)
```

```
## [1] 0
```

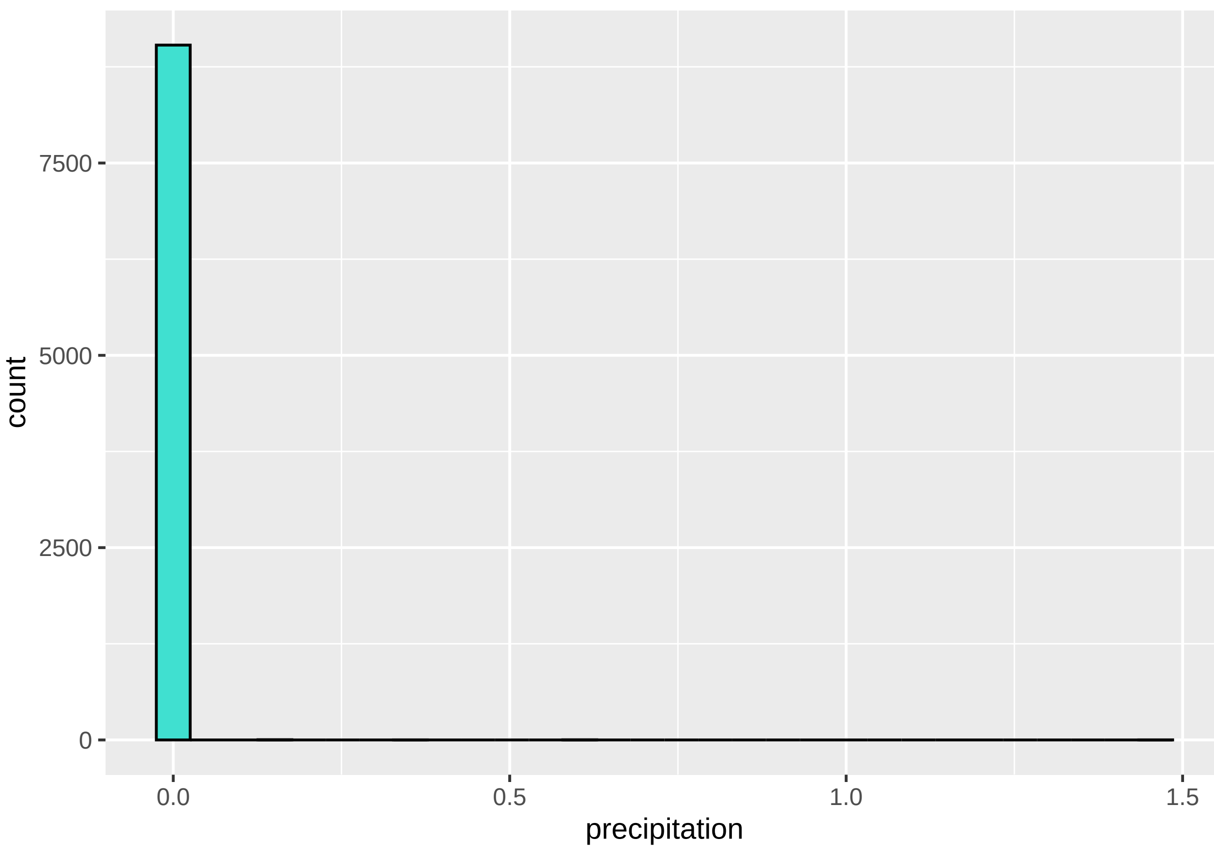
```
max(data$precipitation)
```

```
## [1] 1.46
```

The `precipitation` values range from 0 to 1.46. The details about the scale used here also are not present in the docs.

```
data %>%
  ggplot(aes(x = precipitation)) +
  geom_histogram(fill = "turquoise", col = "black")
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



The most prominent precipitation value is zero (no rain or snow). All of the other values on the scale barely appear, being almost a line at the bottom of the graph.

Inspecting those values different to zero:

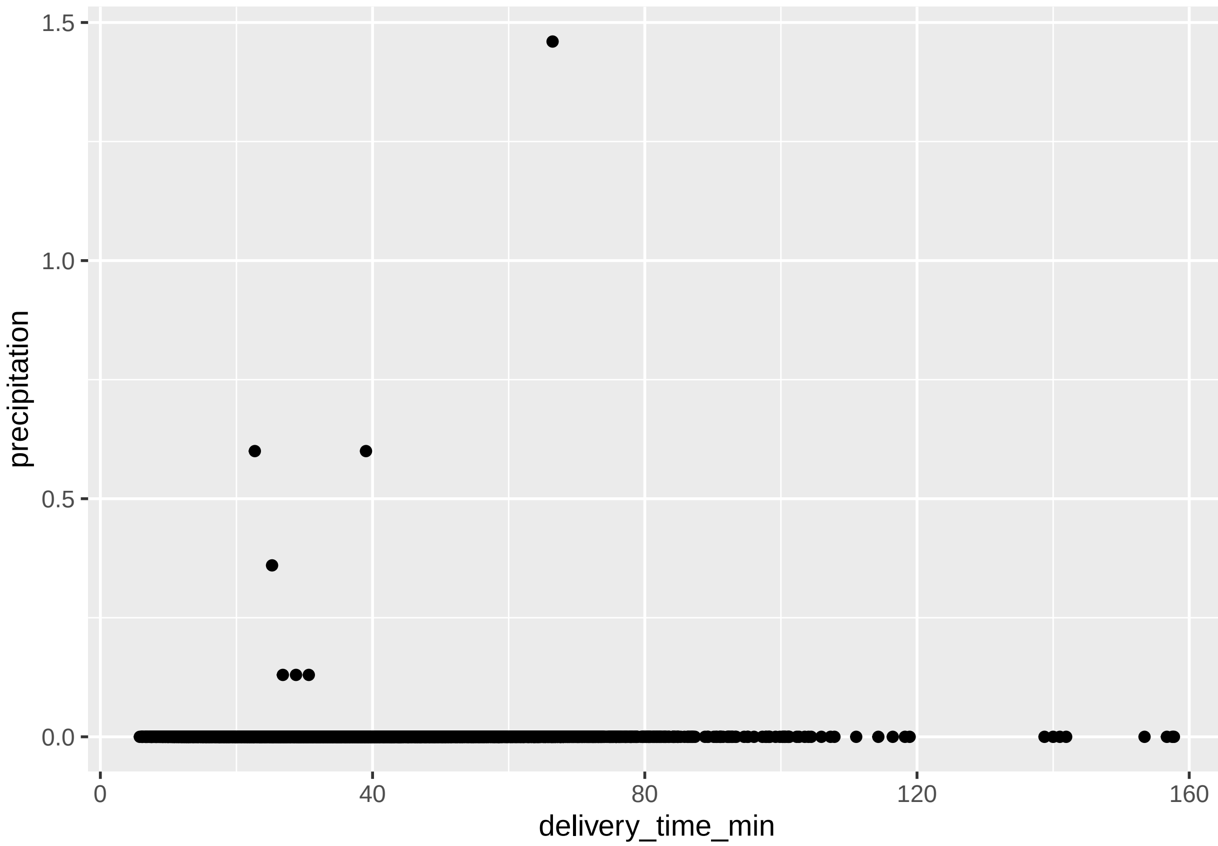
```
data %>%
  filter(precipitation != 0)
```

```
##     id delivery_person_id delivery_person_age delivery_person_ratings
## 1 A8AA     CHENRES11DEL01                  35                     4.0
## 2 5141     CHENRES08DEL01                  39                     4.1
## 3 6FDC     CHENRES17DEL02                  25                     4.6
```

```
## 4 2C40     CHENRES08DEL03                    21                      4.9
## 5 311E     CHENRES17DEL02                    33                      4.5
## 6  96B     CHENRES15DEL01                    29                      4.6
## 7 924C     CHENRES06DEL01                    35                      3.6
##   restaurant_latitude restaurant_longitude delivery_loc_latitude
## 1             13.0642              80.2364               13.1042
## 2             13.0224              80.2424               13.0624
## 3             13.0455              80.2331               13.0955
## 4             13.0224              80.2424               13.0824
## 5             13.0455              80.2331               13.0955
## 6             13.0263              80.2752               13.0663
## 7             13.0543              80.2572               13.1643
##   delivery_loc_longitude order_type      vehicle_type temperature humidity
## 1                80.2764     Buffet        motorcycle       28.25       82
## 2                80.2824      Snack        motorcycle       28.48       82
## 3                80.2831       Meal  electric_scooter       28.25       82
## 4                80.3024      Snack           scooter       28.50       82
## 5                80.2831     Buffet        motorcycle       28.25       82
## 6                80.3152       Meal           scooter       28.54       82
## 7                80.3672       Meal           scooter       28.45       87
##   precipitation  weather_type  X traffic_level distance delivery_time_min
## 1          0.13          mist NA      Moderate     9.25          30.63333
## 2          0.36          mist NA           Low     8.25          25.23333
## 3          0.13          mist NA      Moderate     9.40          26.81667
## 4          0.60          mist NA          High    12.88          39.03333
## 5          0.13          mist NA      Moderate     9.40          28.75000
## 6          0.60          mist NA           Low     8.77          22.70000
## 7          1.46 moderate rain NA     Very High    25.21          66.45000
```

There's just 1 entry that has a `precipitation` value bigger than 1, but this entry marks the weather as "moderate rain", so values bigger than 1 mean that there's definite rain (and not mist) and the .46 shows how heavy the rainfall is.

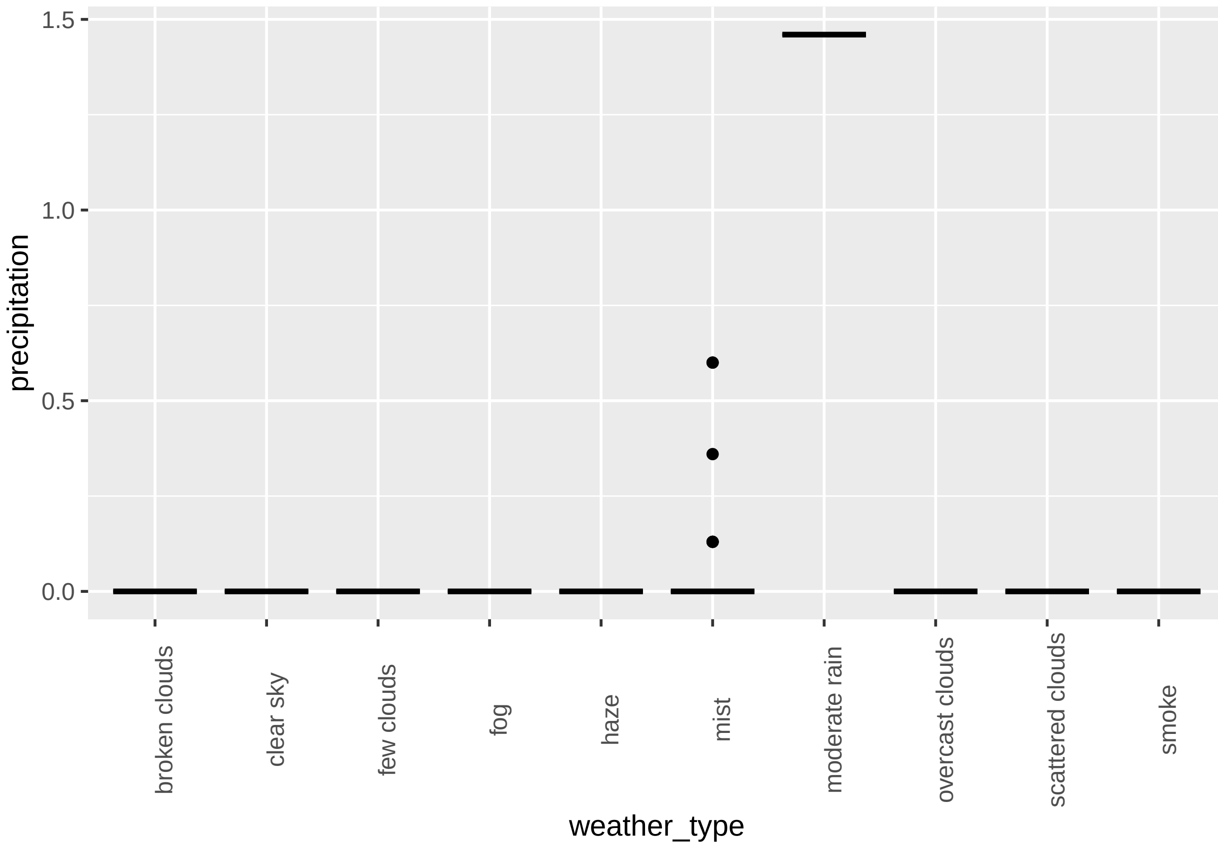
```
data %>%
  ggplot(aes(x = delivery_time_min, y = precipitation)) +
  geom_point(col = "black")
```



Watching the scatterplot, no clear pattern between the precipitation and the delivery time can be extracted.

Weather Type

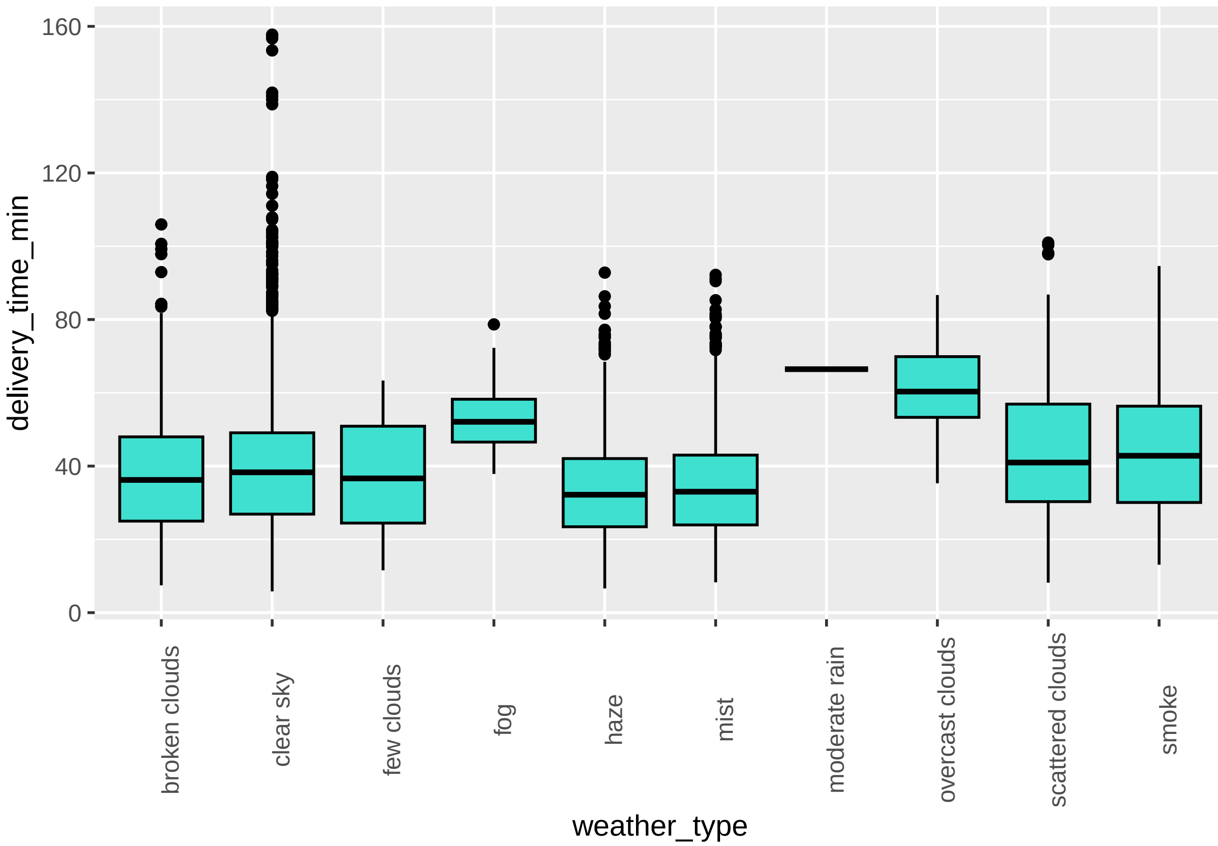
```
data %>%
  ggplot(aes(x = weather_type, y = precipitation)) +
  geom_boxplot(fill = "turquoise", col = "black") +
  theme(axis.text.x = element_text(angle = 90))
```



The only weather types to have non-zero values are mist and moderate rain.

Plotting the possible effect of the weather type on delivery time:

```
data %>%
  ggplot(aes(x = weather_type, y = delivery_time_min)) +
  geom_boxplot(fill = "turquoise", col = "black") +
  theme(axis.text.x = element_text(angle = 90))
```



The weather_type seems to have an extremely little effect on delivery time (see fog, moderate rain and overcast clouds), but the difference is so small that this could introduce almost no effect over the model.

'X' Column

Looking at this unnamed column:

```
head(data$X)
```

```
## [1] NA NA NA NA NA NA
```

```
n_distinct(data$X)
```

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

There is only 1 value in the column and it is NA.

Making a query to determine if there are non-NA row values:

```
data %>%
  filter(!is.na(X))
```

```
##  [1] id                       delivery_person_id      delivery_person_age
##  [4] delivery_person_ratings restaurant_latitude     restaurant_longitude
##  [7] delivery_loc_latitude   delivery_loc_longitude  order_type
```

```
## [10] vehicle_type            temperature             humidity
## [13] precipitation           weather_type            X
## [16] traffic_level           distance                delivery_time_min
## <0 rows> (or 0-length row.names)
```

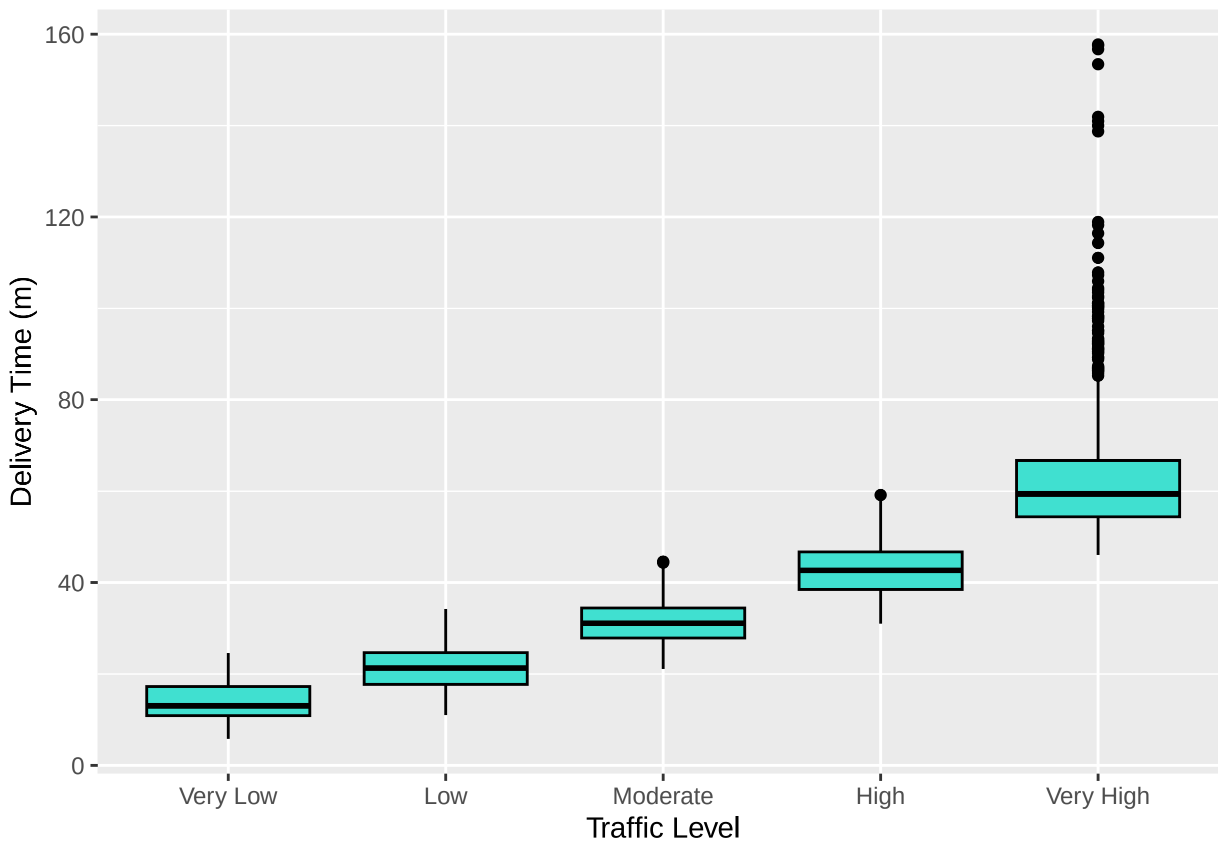
The filter returned 0 rows. Therefore, this column serves no purpose and it will be discarded from the dataset:

```
data <- data %>%
  select(-X)
```

Traffic Level

Naturally, traffic is a great cause for arriving late to a location, however, because the vehicle types here aren't 4 wheeled then I expect the traffic level to have little to no effect on the delivery time.

```
data %>%
  ggplot(aes(x = reorder(traffic_level, delivery_time_min, FUN = median), y = delivery_time_min)) +
  geom_boxplot(fill = "turquoise", col = "black") +
  labs(x = "Traffic Level",
       y = "Delivery Time (m)")
```



Despite the vehicle types in the dataset being 2-wheeled, the traffic does have quite a clear correlation with the delivery time.

This means that the `traffic_level` does have an effect in the delivery time.

Distance

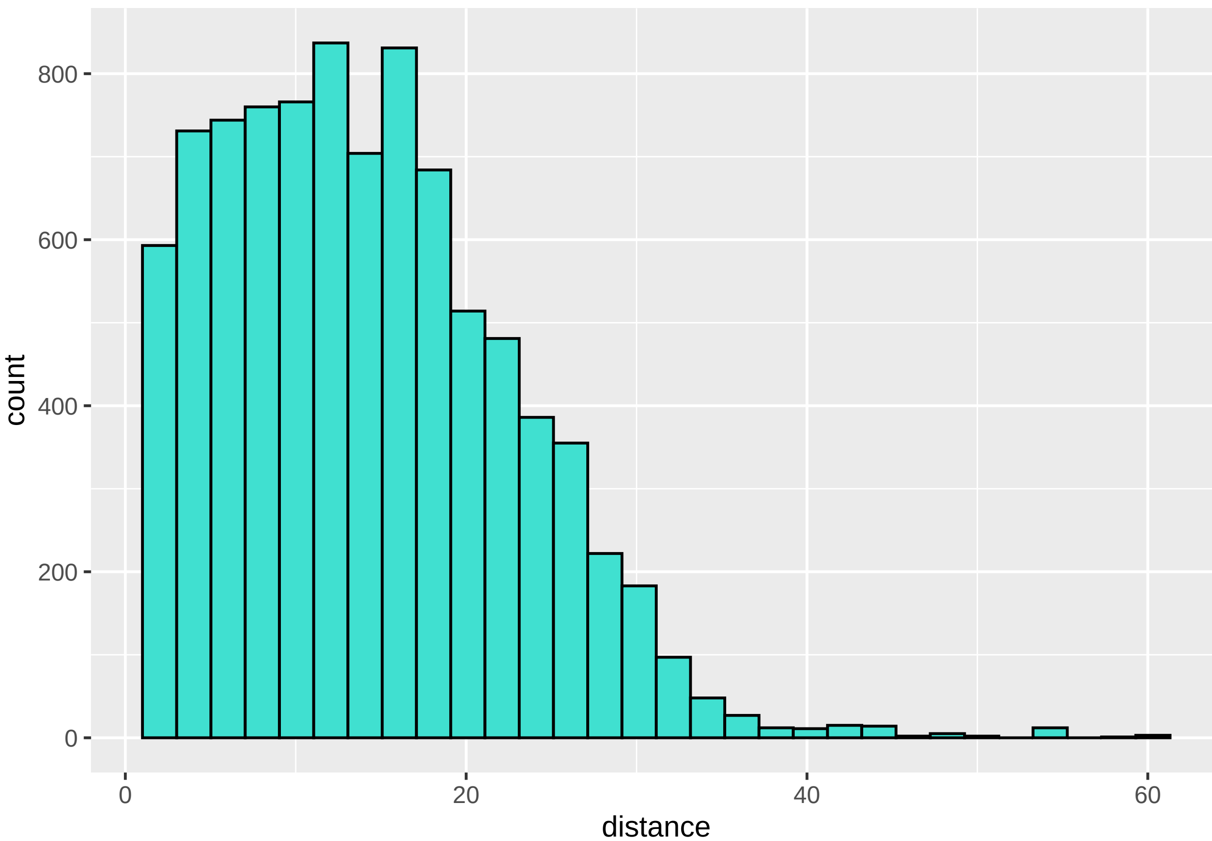
```
mean(data$distance)
```

```
## [1] 14.27331
```

The average distance traveled by the delivery people is around 14 kilometers.

```
data %>%
  ggplot(aes(distance)) +
  geom_histogram(fill = "turquoise", col = "black")
```

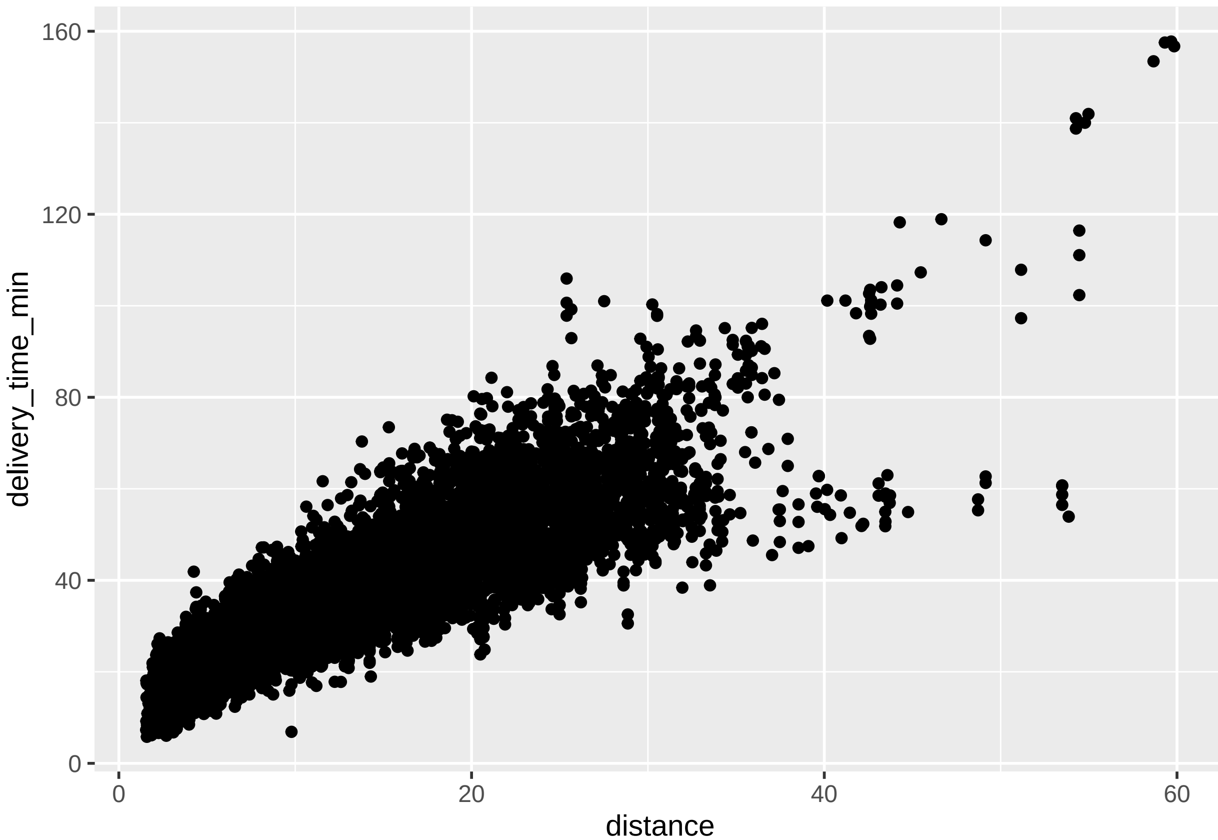
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



Shorter distances are more common than longer distances. This is actually the expected behavior of the data because different apps have a threshold on the maximum amount of kilometers that the restaurant can be away from the user that is making the order.

The distance itself is naturally a big factor in arrival time to a destination everywhere in the world so I expect this behavior to apply here:

```
data %>%
  ggplot(aes(x = distance, y = delivery_time_min)) +
  geom_point(col = "black")
```



There is a clear line-like pattern between distance and delivery time, as I expected. The bigger the distance, the greater the delivery time.

Delivery Time

This is the target variable. This variable measures the delivery time in minutes from the point of making the order to the point of receiving it.

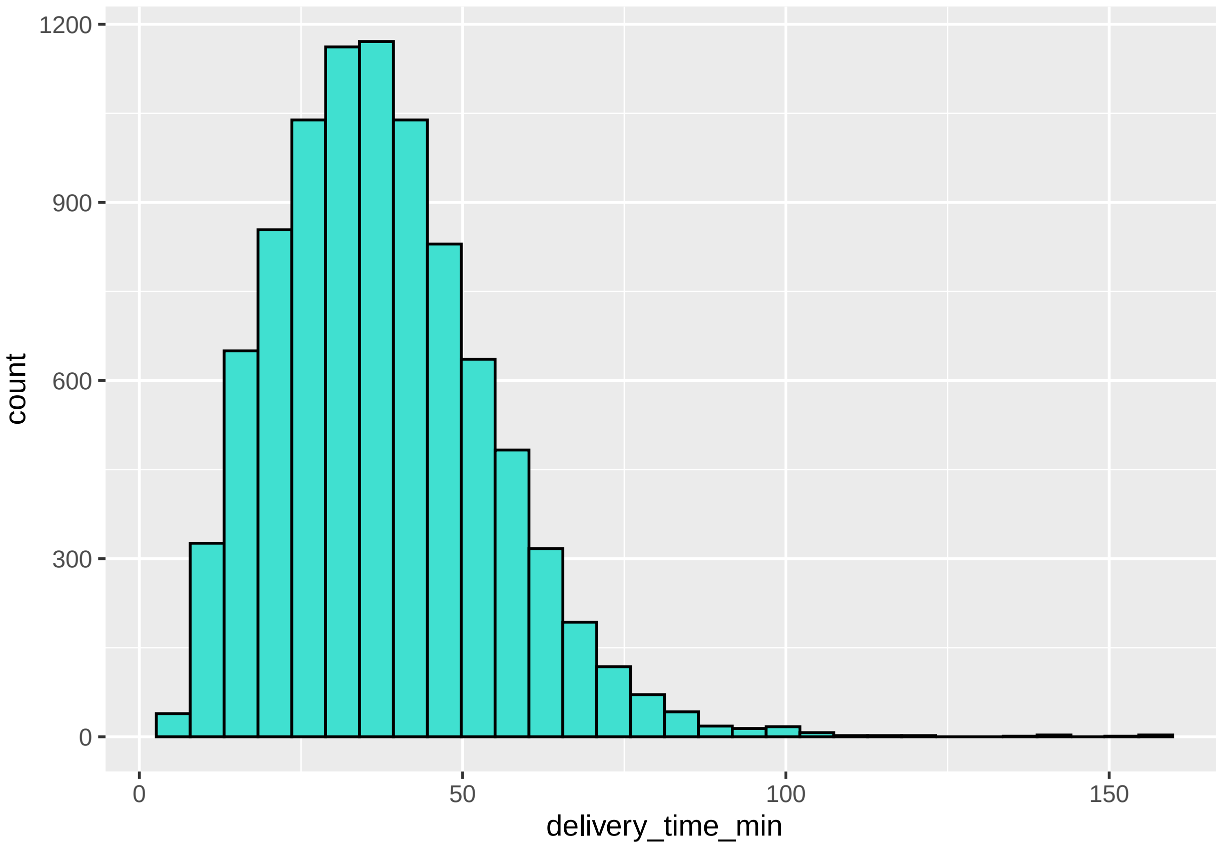
```
mean(data$delivery_time_min)
```

```
## [1] 37.65675
```

People usually have to wait around 38 minutes for their food delivery in this specific dataset.

```
data %>%
  ggplot(aes(delivery_time_min)) +
  geom_histogram(fill = "turquoise", col = "black")
```

```
## 'stat_bin()' using 'bins = 30'. Pick better value with 'binwidth'.
```



Usually, extremely quick deliveries are rare, as well as extremely delayed ones, with the middleground being more common. This 'natural' behavior is seen in the histogram.

Handling Missing Data

Although it is true that a lot of missing data was handled at the start of the analysis section, the presence of these NA values will be checked again in order to discard any possible unwanted behavior.

```
sum(is.na(data))
```

```
## [1] 0
```

There's no missing data in the whole dataset. As it was mentioned above, this is greatly due to the work done at the start of this section.

Handling Inconsistent Values

It was pointed out before that the `delivery_person_id` column is basically made up of inconsistent values (not the expected behavior for this study) since it can't correctly identify a singular delivery person.

As a reminder, the rows will be grouped by `delivery_person_id` and the amount of different ages assigned to that `delivery_person_id` will be printed out:

```
data %>%
  group_by(delivery_person_id) %>%
  distinct(delivery_person_age) %>%
  summarize(distinct_ages = n())
```

```
## # A tibble: 1,134 x 2
##    delivery_person_id distinct_ages
##    <chr>                      <int>
##  1 AGRRES010DEL01                 2
##  2 AGRRES010DEL02                 2
##  3 AGRRES010DEL03                 2
##  4 AGRRES01DEL01                  2
##  5 AGRRES01DEL02                  3
##  6 AGRRES01DEL03                  4
##  7 AGRRES03DEL01                  4
##  8 AGRRES03DEL02                  2
##  9 AGRRES03DEL03                  2
## 10 AGRRES04DEL01                  3
## # i 1,124 more rows
```

Again, it can be seen that the `deliery_person_id` AGRRES010DEL01 is assigned to two different age numbers, symbolizing different people.

These reasons set the ground for the explanation on the elimination of the `delivery_person_id` column from the dataset:

```
data <- data %>%
  select(-delivery_person_id)
```

The `id` column was also shown to be of little value to the current study, having no specific meaning over the dataset.

Showing the amount of times that the same `id` appears in the dataset:

```
data %>%
  group_by(id) %>%
  summarize(appearances = n()) %>%
  arrange(desc(appearances))
```

```
## # A tibble: 9,037 x 2
##    id       appearances
##    <chr>          <int>
##  1 6.00E+02           2
##  2 6.00E+03           2
##  3 9.00E+02           2
##  4 09-Dec             1
##  5 1.00E+03           1
##  6 1.00E+12           1
##  7 1.00E+13           1
##  8 1.00E+20           1
##  9 1.00E+21           1
## 10 1.00E+23           1
## # i 9,027 more rows
```

There are multiple `id`'s that appear more than once. The first `id` that appears will be grabbed for analysis:

```
data %>%
  filter(id == "6.00E+02")
```

```
##         id delivery_person_age delivery_person_ratings restaurant_latitude
## 1 6.00E+02                  29                     4.6             22.3128
## 2 6.00E+02                  22                     4.8             19.1266
##   restaurant_longitude delivery_loc_latitude delivery_loc_longitude order_type
## 1             73.1703               22.4228                73.2803     Snack
## 2             72.8300               19.1366                72.8400     Snack
##        vehicle_type temperature humidity precipitation weather_type
## 1 electric_scooter        22.47       44             0    clear sky
## 2          scooter        27.92       57             0        smoke
##   traffic_level distance delivery_time_min
## 1          High    20.29          44.28333
## 2      Very Low     2.05          19.68333
```

Here the same id is assigned to different people and different deliveries, confirming the fact that this column is of little value for the data observations.

The `id` column will be removed:

```
data <- data %>%
  select(-id)
```

Looking at the new structure of the dataset and checking for duplicate entries:

```
str(data)
```

```
## 'data.frame':    9040 obs. of  15 variables:
##  $ delivery_person_age    : int  37 34 23 38 32 22 33 35 22 36 ...
##  $ delivery_person_ratings: num  4.9 4.5 4.4 4.7 4.6 4.8 4.7 4.6 4.8 4.2 ...
##  $ restaurant_latitude    : num  22.7 12.9 12.9 11 13 ...
##  $ restaurant_longitude   : num  75.9 77.7 77.7 77 80.2 ...
##  $ delivery_loc_latitude  : num  22.8 13 12.9 11.1 13 ...
##  $ delivery_loc_longitude : num  75.9 77.8 77.7 77 80.3 ...
##  $ order_type             : Factor w/ 5 levels "","Buffet ","Drinks ",..: 5 5 3 2 5 2 4 4 2 5 ...
##  $ vehicle_type           : Factor w/ 5 levels "","bicycle ",..: 4 5 4 4 5 4 5 4 4 4 ...
##  $ temperature            : num  17.1 19.5 20.4 23.9 26.6 ...
##  $ humidity               : int  77 93 91 78 87 65 69 82 65 77 ...
##  $ precipitation          : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ weather_type           : Factor w/ 12 levels "","broken clouds",..: 6 8 8 8 8 2 3 11 2 3 ...
##  $ traffic_level          : Factor w/ 7 levels "","High","Low",..: 3 6 3 4 2 4 2 6 6 2 ...
##  $ distance               : num  3.03 37.17 3.34 10.05 9.89 ...
##  $ delivery_time_min      : num  21.7 85.3 28.6 35.2 43.5 ...
```

```
sum(duplicated(data))
```

```
## [1] 0
```

There are no duplicate entries in the data and the `id` column no longer appears as part of the structure of the dataset.

Dataset Split

In order to correctly apply the AI/ML models in the data, a division is needed (between training and testing) in order to evaluate the model later on on simulated "unseen" data.

The split will be done as follows:

First, the seed will be set to the current year (2025, this could be any number) to establish reproducible results (because the partition is done "randomly").

- `train_data`: dataset used for model training, is comprised of the 80% of the original `data`
- `test_data`: dataset used for model evaluation, comprised of the 20% of the original `data`

```r
if (!require(caret)) install.packages("caret")
```

```
## Loading required package: caret

## Loading required package: lattice
```

```r
library(caret)

set.seed(2025)

train_indexes <- createDataPartition(data$delivery_time_min, p = 0.8, list = FALSE)
train_data <- data[train_indexes,]
test_data <- data[-train_indexes,]
```

Outlier Deletion

No outliers will be deleted since important features like the `distance` correlate very well and give good insights for predicting the delivery time in specific situations (for example long distances mean longer delivery times). Deleting these values could affect the pattern-learning process of the future models.

Data Scaling

The models that are going to be trained are:

- Linear Regression
- Random Forest

Both of these models are not sensible to the scale of the values that they will be trained on. In this context, the scale of the values determine the range in which all of the numerical observations fall in.

Considering this and the benefit of the understandability in features as well as the target variable, the dataset will not be scaled (neither normalized or standardized).

Feature Selection

Along this analysis section, each variable was studied with its effect on the target variable. This was done in order to evaluate whether that specific variable could be a possible predictor for the delivery time duration.

In order to confirm the findings and solidify the inferences made, the correlation between each variable will be calculated. With this, a "correlation heatmap" can be done in order to visually examine the magnitude of the correlations between variables.

It is important to mention that these correlation calculations will be done on numerical data only, so filtering of these is needed first.

```
nums <- lapply(data, is.numeric)
nums <- unlist(nums)

head(data[, nums])
```

```
##   delivery_person_age delivery_person_ratings restaurant_latitude
## 1                  37                     4.9             22.7450
## 2                  34                     4.5             12.9130
## 3                  23                     4.4             12.9143
## 4                  38                     4.7             11.0037
## 5                  32                     4.6             12.9728
## 6                  22                     4.8             17.4317
##   restaurant_longitude delivery_loc_latitude delivery_loc_longitude temperature
## 1              75.8925               22.7650                75.9125       17.11
## 2              77.6832               13.0430                77.8132       19.50
## 3              77.6784               12.9243                77.6884       20.45
## 4              76.9765               11.0537                77.0265       23.86
## 5              80.2500               13.0128                80.2900       26.55
## 6              78.4083               17.4617                78.4383       21.43
##   humidity precipitation  distance delivery_time_min
## 1       77             0  3.028538          21.66667
## 2       93             0 37.170000          85.26667
## 3       91             0  3.340000          28.58333
## 4       78             0 10.050000          35.18333
## 5       87             0  9.890000          43.45000
## 6       65             0 11.300000          30.60000
```

```
correlation_matrix <- cor(data[,nums])
head(correlation_matrix)
```

```
##                         delivery_person_age delivery_person_ratings
## delivery_person_age             1.000000000            -0.092834440
## delivery_person_ratings        -0.092834440             1.000000000
## restaurant_latitude             0.017862603            -0.003074281
## restaurant_longitude            0.008535053             0.022201845
## delivery_loc_latitude           0.017878855            -0.003890764
## delivery_loc_longitude          0.008570516             0.020944020
##                         restaurant_latitude restaurant_longitude
## delivery_person_age             0.017862603          0.008535053
## delivery_person_ratings        -0.003074281          0.022201845
## restaurant_latitude             1.000000000          0.005112898
## restaurant_longitude            0.005112898          1.000000000
```

```
## delivery_loc_latitude          0.999977607          0.005064663
## delivery_loc_longitude         0.005655348          0.999946826
##                          delivery_loc_latitude delivery_loc_longitude
## delivery_person_age                0.017878855            0.008570516
## delivery_person_ratings           -0.003890764            0.020944020
## restaurant_latitude                0.999977607            0.005655348
## restaurant_longitude               0.005064663            0.999946826
## delivery_loc_latitude              1.000000000            0.005676019
## delivery_loc_longitude             0.005676019            1.000000000
##                           temperature      humidity precipitation     distance
## delivery_person_age      0.002669415  0.0008743522   0.006994943  0.005180805
## delivery_person_ratings -0.009955944  0.0161309005  -0.030616687 -0.116652166
## restaurant_latitude     -0.289902827 -0.5557676044  -0.022023998  0.046272299
## restaurant_longitude    -0.126766058  0.4347551184   0.019113758 -0.019324991
## delivery_loc_latitude   -0.290086375 -0.5565263777  -0.021976146  0.052529865
## delivery_loc_longitude  -0.127218082  0.4332748472   0.019175043 -0.009669051
##                         delivery_time_min
## delivery_person_age           0.005870754
## delivery_person_ratings      -0.100144418
## restaurant_latitude          -0.030036938
## restaurant_longitude          0.012226868
## delivery_loc_latitude        -0.024641356
## delivery_loc_longitude        0.020512847
```

These values contain a lot of numbers, so rounding will be used again here for ease-of-read purposes. The rounding will be done to 2 decimal places.

```
correlation_matrix <- round(correlation_matrix, 2)

head(correlation_matrix)
```

```
##                         delivery_person_age delivery_person_ratings
## delivery_person_age                    1.00                   -0.09
## delivery_person_ratings               -0.09                    1.00
## restaurant_latitude                    0.02                    0.00
## restaurant_longitude                   0.01                    0.02
## delivery_loc_latitude                  0.02                    0.00
## delivery_loc_longitude                 0.01                    0.02
##                         restaurant_latitude restaurant_longitude
## delivery_person_age                    0.02                 0.01
## delivery_person_ratings                0.00                 0.02
## restaurant_latitude                    1.00                 0.01
## restaurant_longitude                   0.01                 1.00
## delivery_loc_latitude                  1.00                 0.01
## delivery_loc_longitude                 0.01                 1.00
##                         delivery_loc_latitude delivery_loc_longitude
## delivery_person_age                      0.02                   0.01
## delivery_person_ratings                  0.00                   0.02
## restaurant_latitude                      1.00                   0.01
## restaurant_longitude                     0.01                   1.00
## delivery_loc_latitude                    1.00                   0.01
## delivery_loc_longitude                   0.01                   1.00
##                         temperature humidity precipitation distance
```

```
## delivery_person_age            0.00     0.00          0.01     0.01
## delivery_person_ratings       -0.01     0.02         -0.03    -0.12
## restaurant_latitude           -0.29    -0.56         -0.02     0.05
## restaurant_longitude          -0.13     0.43          0.02    -0.02
## delivery_loc_latitude         -0.29    -0.56         -0.02     0.05
## delivery_loc_longitude        -0.13     0.43          0.02    -0.01
##                         delivery_time_min
## delivery_person_age                  0.01
## delivery_person_ratings             -0.10
## restaurant_latitude                 -0.03
## restaurant_longitude                 0.01
## delivery_loc_latitude               -0.02
## delivery_loc_longitude               0.02
```

Converting the data into a long format (it's currently as a wide format since each variable is put as a column):

```
if (!require(reshape2)) install.packages("reshape2")
```

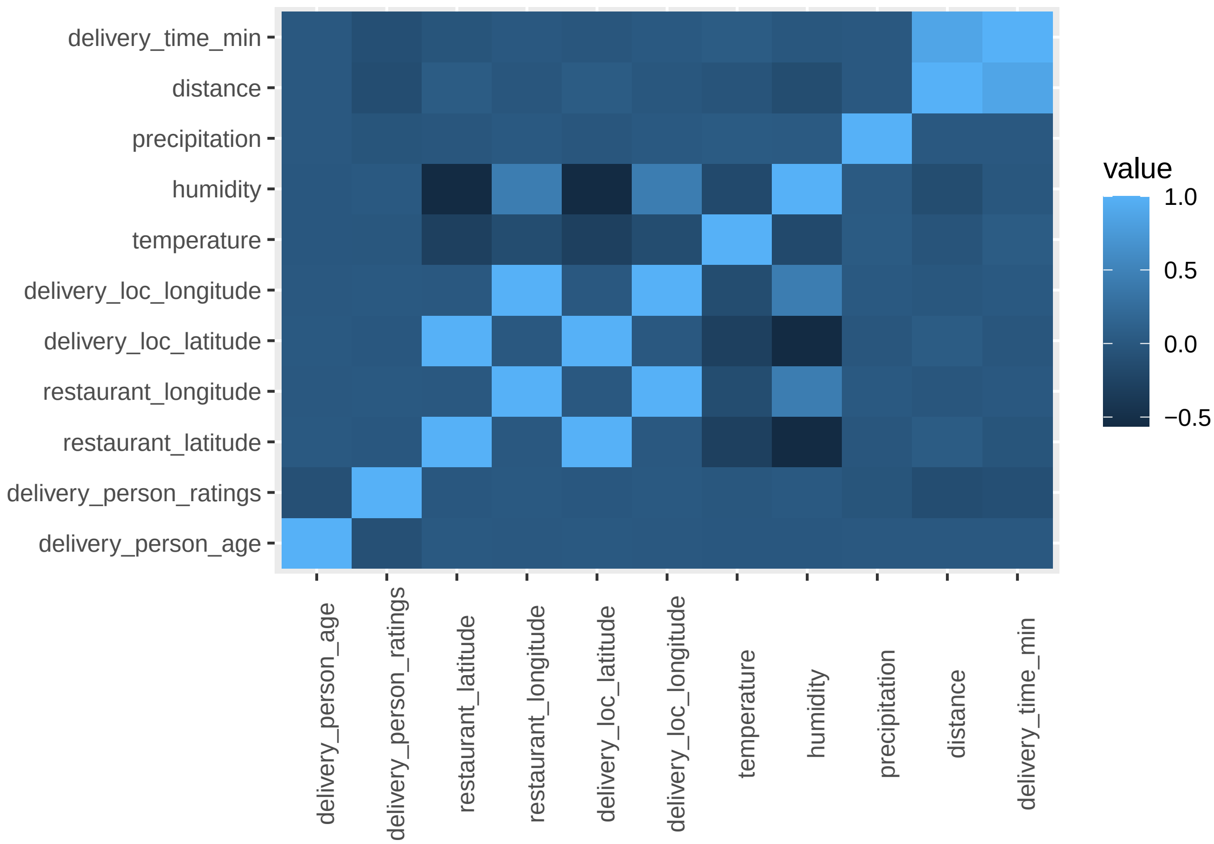
```
## Loading required package: reshape2
```

```
library(reshape2)

correlation_matrix <- melt(correlation_matrix)

head(correlation_matrix)
```

```
##                      Var1                Var2 value
## 1     delivery_person_age delivery_person_age  1.00
## 2 delivery_person_ratings delivery_person_age -0.09
## 3     restaurant_latitude delivery_person_age  0.02
## 4    restaurant_longitude delivery_person_age  0.01
## 5   delivery_loc_latitude delivery_person_age  0.02
## 6  delivery_loc_longitude delivery_person_age  0.01
```

Now it can be seen that the relationships between variables are described in just 2 columns, rather than a lot more (as it was before). Now creating a correlation heatmap:

```
correlation_matrix %>%
  ggplot(aes(x = Var1, y = Var2, fill = value)) +
  geom_tile() +
  theme(axis.text.x = element_text(angle = 90),
        axis.title = element_blank())
```



It can be seen that distance itself has the lightest color out of all of the possible features, with temperature being the next-lightest. This was hinted at in during the EDA process where both variables were examined (including their effect on delivery time). So, thanks to the EDA and the confirmation of this heatmap, the features of the model will be:

- `order_type`
- `vehicle_type`
- `temperature`
- `weather_type`
- `traffic_level`
- `distance`

Model Implementation

As aforementioned, the two models that will be trained and evaluated are:

- Linear Regression
- Random Forest

These models will be trained on the `train_data` and evaluated on the `test_data` using RMSE as the metric.

Linear Regression

Linear Regression is the simplest machine learning model for regression tasks. Regression tasks are those that predict a single numerical value (either floating point or integer), for example, predicting the amount of hours of sleep of an individual.

This is done by tracing a line across the target variable and whenever the model encounters a new value, it uses the features to know where in the line the new value should fall into.

More explicitly, it does this by assigning a "weight" to the predictor variables (since predictors like `distance` should be more influential than `order_type` for example).

```
lr_model <- lm(delivery_time_min ~ order_type + vehicle_type + temperature + weather_type +
     traffic_level + distance, data = train_data)
summary(lr_model)
```

```
##
## Call:
## lm(formula = delivery_time_min ~ order_type + vehicle_type +
##     temperature + weather_type + traffic_level + distance, data = train_data)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -26.896  -2.346  -0.134   1.945  72.749
##
## Coefficients:
##                              Estimate Std. Error t value Pr(>|t|)
## (Intercept)                  29.40217    1.89004  15.556  < 2e-16 ***
## order_typeDrinks              6.67227    0.17062  39.106  < 2e-16 ***
## order_typeMeal               -2.20768    0.17183 -12.848  < 2e-16 ***
## order_typeSnack               1.77332    0.17056  10.397  < 2e-16 ***
## vehicle_typeelectric_scooter  0.10089    1.82031   0.055 0.955802
## vehicle_typemotorcycle        0.45944    1.80928   0.254 0.799555
## vehicle_typescooter           0.63120    1.81075   0.349 0.727410
## temperature                  -0.09838    0.02041  -4.819 1.47e-06 ***
## weather_typeclear sky        -0.99921    0.26776  -3.732 0.000192 ***
## weather_typefew clouds        0.20270    0.95092   0.213 0.831210
## weather_typefog               0.52801    0.81259   0.650 0.515847
## weather_typehaze              0.13995    0.27840   0.503 0.615207
## weather_typemist              2.59968    0.29047   8.950  < 2e-16 ***
## weather_typemoderate rain     9.86480    5.11947   1.927 0.054029 .
## weather_typeovercast clouds   5.56659    0.52347  10.634  < 2e-16 ***
## weather_typescattered clouds  2.25063    0.37708   5.968 2.51e-09 ***
## weather_typesmoke             6.55957    0.38070  17.230  < 2e-16 ***
## traffic_levelLow            -13.12966    0.24525 -53.536  < 2e-16 ***
## traffic_levelModerate        -6.80292    0.18875 -36.042  < 2e-16 ***
## traffic_levelVery High       12.91546    0.21353  60.485  < 2e-16 ***
## traffic_levelVery Low       -18.07344    0.33629 -53.744  < 2e-16 ***
## distance                      0.73951    0.01492  49.579  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5.108 on 7211 degrees of freedom
## Multiple R-squared:  0.9033, Adjusted R-squared:  0.9031
## F-statistic:  3209 on 21 and 7211 DF,  p-value: < 2.2e-16
```

```
if (!require(Metrics)) install.packages("Metrics")
```

```
## Loading required package: Metrics

##
## Attaching package: 'Metrics'

## The following objects are masked from 'package:caret':
##
##     precision, recall
```

```
library(Metrics)

y_hat_lr <- predict(lr_model, newdata = test_data)

rmse(test_data$delivery_time_min, y_hat_lr)
```

```
## [1] 5.301928
```

The resulting Root Mean Squared Error from the Linear Regression model is 5.3

This means that the model averagely misses its prediction by 5 minutes. Considering the fact that people usually wait around 36 minutes for their food (calculated above), this error isn't big enough to discard the model. In fact, this model could be extremely useful for new upcoming delivery app companies to incorporate an ETA of the food while needing low computational power and energy.

Random Forest

Random Forest is a different type of Machine Learning algorithm compared to Linear Regression. What this algorithm does is that it creates multiple *decision trees* and takes the average resulting value from these trees generating a prediction. A decision tree what does is that it generates a prediction based on different thresholds from the features used in the model. For example if a person's height is higher than 1.7 meters, guess that the person's age is 25.

This approach is more computationally expensive and costly than using Linear Regression, but by doing this, this model is able to learn more complicated patterns in data that aren't necessarily linear.

First, the random forest package needs to be installed and loaded in order to train the random forest model:

```
if (!require(randomForest)) install.packages("randomForest")
```

```
## Loading required package: randomForest

## randomForest 4.7-1.2

## Type rfNews() to see new features/changes/bug fixes.

##
## Attaching package: 'randomForest'
```

```
## The following object is masked from 'package:ggplot2':
##
##     margin

## The following object is masked from 'package:dplyr':
##
##     combine
```

```
library(randomForest)
```

For this model, cross validation will be applied. What this does is that it divides the dataset into k folds, and it trains the model on k-1 folds while it evaluates it on the remaining fold. It does this k times so that each fold serves as the evaluation fold at least once.

Setting up k as 10 for k-fold cross validation:

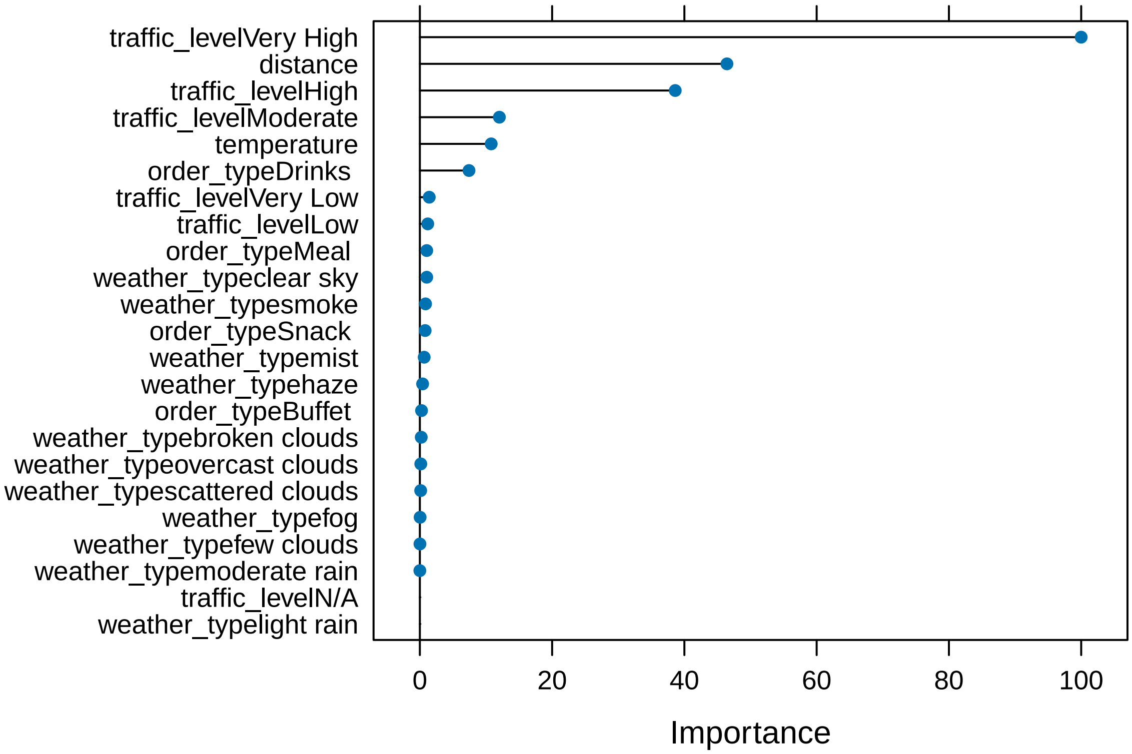
```
train_control <- trainControl(method = "cv",
                              number = 10)
```

Training the model on the `train_data` dataset:

```
rf_model <- train(delivery_time_min ~ order_type + temperature + weather_type + traffic_level + distance
                  data = train_data,
                  method = "rf",
                  trControl = train_control)
rf_model
```

```
## Random Forest
##
## 7233 samples
##    5 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 6509, 6509, 6511, 6509, 6509, 6510, ...
## Resampling results across tuning parameters:
##
##   mtry  RMSE      Rsquared   MAE
##    2    7.177006  0.9002355  4.977704
##   12    3.641954  0.9508883  2.577479
##   23    3.577116  0.9525000  2.533693
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was mtry = 23.
```

```
plot(varImp(rf_model))
```




```r
y_hat_rf <- predict(rf_model, newdata = test_data)

rmse(test_data$delivery_time_min, y_hat_rf)
```

```
## [1] 3.642145
```

The lowest RMSE obtained during training for the random forest model is 3.577116, with an `mtry` value of 23. Along with this, the RMSE of the predictions on the testing set was also calculated, and the obtained value is 3.642145. This value is significantly lower than the 5.301928 RMSE that Linear Regression yields.

It can also be seen that the most influential variable while predicting delivery time in random forest is the traffic level, especially when this variable has the "Very High" traffic value, with `distance` being the second most important variable after that level of traffic.

Results

After doing an extensive analysis section that studied each variable along with its intrinsic patterns, correlations with the target variable and inconsistencies, the best correlated features (variables) were selected. The selected features were:

- `order_type`
- `vehicle_type`
- `temperature`
- `weather_type`
- `traffic_level`

- `distance`

These features were used by a Linear Regression model and a Random Forest model. After training both models, the RMSE between them was compared in order to gather a consistent metric that permitted a correct and objective selection of the best model.

The resulting RMSE's are the following:

- **Linear Regression:** Less computationally expensive, RMSE = 5.301928.
- **Random Forest:** More computationally expensive, RMSE = 3.642145.

Both of these RMSE's reported are the RMSE values that result when comparing the actual testing data to the model's predictions.

From this, it can be seen that the Random Forest model is the clear winner in this case, having a RMSE that is lower than the Linear Regression counterpart by 1.659783.

Conclusion

This study analyzed the effects of different factors that come into play when a delivery of food is made to a specific home. Characteristics such as the type of food that is included in the order, the type of vehicle that the food will be delivered in, the ambient temperature while delivering, the weather, traffic level and distance proved to be good predictors for the duration time of a specific food delivery.

Before I mentioned how this model could benefit up and coming apps on the food delivery industry so that the end user (the one who initially ordered) can get an idea of how much the food is going to take getting to the destination (they could be ordering from work or home).

Additionally, this approach can also be fine-tuned to fit more and possibly more complex use cases. For example, a package delivery company that sends letters, boxes and more across a country, can use this model in order to know how much time the package is going to take to get to its destination based on traffic level, type of package (fragile, non-fragile, flammable, etc.) and more features.

A limitation of the current model is that it doesn't take into consideration the time of day when the order was made. This could've been a great feature to consider, since standard meal times (breakfast, lunch and dinner) are the times when the restaurants are the most filled up.

Another improvement that can be made to the dataset (and also the model) is adding a restaurant ID along with the observations. This is because, as I mentioned, in malls and shopping places the restaurants are usually on top of another, so only coordinates aren't a reliable source of restaurant identification.

Along with this, restaurant rating can be added to know whether specific restaurants have low order counts due to their bad food or their lack of good service times.

References

Belsky & Horowitz, LLC. 2022. "Tips for Avoiding Heat Sickness While Driving." https://www.belsky-weinberg-horowitz.com/tips-for-avoiding-heat-sickness-while-driving/.

Hijmans, Robert J. 2024. "Geosphere: Spherical Trigonometry for Geographical Applications." https://github.com/rspatial/geosphere.

NASA Science. 2023. "Solar System Temperatures." https://science.nasa.gov/resource/solar-system-temperatures/.

Patterson, Dan. 2019. "Distance on a Sphere: The Haversine Formula." https://community.esri.com/t5/coordinate-reference-systems-blog/distance-on-a-sphere-the-haversine-formula/ba-p/902128.