Case Study ETA-Prognosen fuer Binnenschiffe

Klara Hinze, Nicola Leschke, Carlo Schmid, Ronny Georgi,

30/07/2021

Case Study ETA-Prognosen fuer Binnenschiffe

Vorbereitung: benoetigte R Packages laden

```
#Die folgenden Packages muessen installiert und geladen werden
library(knitr) #zum "Knitten" des Rmd Dokuments
library(rmarkdown) #zum "Knitten" des Rmd Dokuments
library(tidyverse) #enthaelt benoetigte Packages wie dplyr, tidyr, ggplot2, stringr
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --
```

```
## v ggplot2 3.3.3     v purrr   0.3.4
## v tibble  3.1.1     v dplyr   1.0.5
## v tidyr   1.1.3     v stringr 1.4.0
## v readr   1.4.0     v forcats 0.5.1
```

```
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library(readxl) #zum Laden der Excel-Datei
library(ggplot2) #fuer visuelle Darstellung
library(zoo) # fuer Zeitreihenanalyse
```

```
##
## Attaching package: 'zoo'
```

```
## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric
```

```
library(forecast) # fuer Zeitreihenanalyse
library(Metrics) #fuer lineare Regression
```

```
##
## Attaching package: 'Metrics'
```

```
## The following object is masked from 'package:forecast':
##
```

```
##     accuracy
```

```
library(dummies) #fuer lineare Regression
```

```
## dummies-1.5.6 provided by Decision Patterns
```

```
library(GGally)  #fuer lineare Regression
library("geosphere") # distance function for geometrical points
```

1. Datenanalyse

1.1. Laden der Daten

```
#Vom Case Study Partner bereitgestellte Daten
waterLevels = read.csv("water_levels.csv")
tripsAggregated = read.csv("trips_aggregated.csv")
tripsRaw= read.csv("adjusted_trips_raw.csv")
#tripsRaw = read.csv("trips_raw.csv")
waterLevelsStations = read.csv("water_levels_stations.csv")

#eigene erstellte Datensaetze
shiptype = read.csv2("ship_type.csv", fileEncoding = 'UTF-8-BOM')
#Begruendung: fuer das Mapping der einzelnen Tripdaten mit den Schiffstypen
#anhand der typeOfShipId benoetigen wir eine separate CSV-Datei zum einlesen
#der Daten und der geordneten Ermittlung der Schiffstypen
#Quelle: https://api.vtexplorer.com/docs/ref-aistypes.html

vacation = read_xlsx("Feriendaten.xlsx")
#Begruendung siehe code chunk Feriendaten-Klara,
#Quellen: https://www.schulferien.org/holland/ferien/2019/;
#https://www.kalenderpedia.de/ferien/ferien-2019.html

river_data_germany = read.csv("wasserstrassen.csv")
#Open Data zu Fluessen in Deutschland, gefunden unter:
#https://opendata-esri-de.opendata.arcgis.com/datasets/esri-de-content:
#:wsv-bundeswasserstra%C3%9Fen/explore?location=51.133692%2C10.411170%2C6.82&showTable=true

river_data_netherlands = read.csv("status_vaarweg.csv")
#Open Data zu Fluessen in den Niederlanden, gefunden unter:
#https://data.overheid.nl/en/dataset/16060-vaarweg-informatie-status-vaarwegen--lijnen-
```

1.2. Datenexploration

1.2.1. Struktur der Daten

Um sich einen Ueberblick ueber die Daten zu verschaffen, wird die Struktur ausgegeben, da sie einen guten Ueberblick ueber die Variablen gibt:

```
## Struktur waterLevels:
##

## 'data.frame':    6280 obs. of  4 variables:
##  $ X           : int  198844 198845 198846 198848 198849 198852 198853 198857 198858 198859
##  $ id          : Factor w/ 11 levels "FTS49972718333",..: 5 9 8 7 6 2 1 3 4 10 ...
##  $ measuretime: Factor w/ 981 levels "2019-01-01 05:00:00",..: 1 1 1 1 1 1 1 1 1 2 ...
##  $ value       : num  231 291 397 288 318 202 187 163 220 491 ...
```

```
## Struktur tripsAggregated:
##
## 'data.frame':    150 obs. of  20 variables:
##  $ X                    : int  0 1 2 3 4 5 6 7 8 9 ...
##  $ tripName             : Factor w/ 150 levels "trip_0","trip_1",..: 1 2 63 74 85 96 107 11
##  $ vesselName           : Factor w/ 63 levels "vessel_0","vessel_1",..: 1 2 13 24 35 46 57
##  $ timeStart            : Factor w/ 150 levels "2019-01-03 14:25:41",..: 82 139 47 106 133
##  $ timeEnd              : Factor w/ 150 levels "2019-01-04 16:52:11",..: 82 139 47 105 133
##  $ longitudeStart       : num  8.53 8.52 8.52 8.53 8.56 ...
##  $ longitudeEnd         : num  4.65 4.65 4.67 4.66 4.68 ...
##  $ latitudeStart        : num  50.1 50.1 50.1 50.1 50.1 ...
##  $ latitudeEnd          : num  51.8 51.8 51.8 51.8 51.8 ...
##  $ currentSpeedOverGround: num  6.2 0 0 1.6 7.3 0 7 3.3 8 6.6 ...
##  $ timestampEta         : Factor w/ 108 levels "","2018-11-30 00:00:00",..: 46 90 93 70 87
##  $ destination          : Factor w/ 58 levels "","ANDERNACH",..: 20 10 48 23 52 12 1 23 25
##  $ typeOfShipId         : num  79 80 89 79 79 80 79 79 80 70 ...
##  $ length               : num  135 110 105 135 105 110 136 135 110 105 ...
##  $ width                : num  12 11 12 12 10 12 12 12 12 10 ...
##  $ draught              : num  2.4 2 2.8 0.2 2.5 2.4 NA 0.2 NA 2.5 ...
##  $ dimA                 : num  120 98 95 120 16 110 136 120 110 85 ...
##  $ dim                  : num  15 12 10 15 89 0 0 15 0 20 ...
##  $ dimC                 : num  12 7 8 4 10 12 12 4 5 2 ...
##  $ dimD                 : num  0 4 4 8 0 0 0 8 7 8 ...

## Struktur tripsRaw:
##
## 'data.frame':    21113 obs. of  22 variables:
##  $ X.1              : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ X                : int  0 1 2 3 4 5 6 7 8 9 ...
##  $ tripName         : Factor w/ 150 levels "trip_0","trip_1",..: 41 41 41 41 41 41 41 41 41
##  $ vesselName       : Factor w/ 63 levels "vessel_0","vessel_1",..: 47 47 47 47 47 47 47 47
##  $ timestampPosition: Factor w/ 21040 levels "2019-01-03 15:47:41",..: 664 665 666 667 668 6
##  $ speedOverGround  : num  9.2 9.2 7.6 1 0.5 0 5.7 9.4 9.5 9.5 ...
##  $ courseOverGround : num  242 257 255 273 NA ...
##  $ longitude        : num  8.44 8.41 8.36 8.34 8.34 ...
##  $ latitude         : num  50 50 50 50 50 ...
##  $ timestampVoyage  : Factor w/ 555 levels "2019-01-01 00:51:01",..: 52 52 58 58 58 58 58 58
##  $ destination      : Factor w/ 133 levels "","ANDERNACH",..: 129 129 74 74 74 74 74 74 74 7
##  $ timestampEta     : Factor w/ 181 levels "","2018-11-30 00:00:00",..: 8 8 8 8 8 8 8 8 8 8
##  $ typeOfShipId     : int  89 89 89 89 89 89 89 89 89 89 ...
##  $ length           : int  110 110 110 110 110 110 110 110 110 110 ...
##  $ width            : int  12 12 12 12 12 12 12 12 12 12 ...
##  $ draught          : num  1 1 1 1 1 1 1 1 1 1 ...
##  $ dimA             : int  98 98 98 98 98 98 98 98 98 98 ...
##  $ dim              : int  12 12 12 12 12 12 12 12 12 12 ...
##  $ dimC             : int  4 4 4 4 4 4 4 4 4 4 ...
##  $ dimD             : int  8 8 8 8 8 8 8 8 8 8 ...
```

```
##  $ distanceAchieved : num  7.03 9.07 11.23 12.12 12.38 ...
##  $ waterLocksPassed : int  1 1 1 1 1 1 2 2 2 2 ...

## Struktur waterLevelsStations:
##

## 'data.frame':    11 obs. of  5 variables:
##  $ X          : int  62 65 66 68 70 71 72 75 78 82 ...
##  $ id         : Factor w/ 11 levels "FTS49972718333",..: 1 2 3 4 5 6 7 8 9 10 ...
##  $ latitude   : num  50 50.1 50.1 50.4 50.6 ...
##  $ longitude  : num  7.9 7.76 8.72 7.61 7.21 ...
##  $ measuretype: Factor w/ 1 level "exact": 1 1 1 1 1 1 1 1 1 1 ...
```

Die Ausgabe der Struktur und zusaetzliche visuelle Datenexploration hat einen Bedarf zur Bereinigung ergeben. Dennoch wird im Folgenden keine Notwendigkeit in der Bereinigung gesehen, da der Aufwand nicht dem Nutzen entspricht und Informationen verloren gehen wuerden, wenn beispielsweise NA-Werte geloescht werden. Stattdessen werden diese Werte an den Stellen, an denen sie die Berechnung verfaelschen wuerden, ausgeschlossen.

Allerdings gab es waehrend der Gruppenarbeit Probleme, da die Variablen bei den Gruppenmitgliedern teilweise unterschiedlich interpretiert werden. Daher wird fuer nicht-numerische Variablen der Tabellen *tripsRaw* und *tripsAggregated* der Datentyp explizit angepasst.

```
## Es wird die Struktur der Datensaetze nach der ersten Transformation der Daten angezeigt.

## Struktur tripsAggregated:
##

## 'data.frame':    150 obs. of  22 variables:
##  $ X                     : int  0 1 2 3 4 5 6 7 8 9 ...
##  $ tripName              : Factor w/ 150 levels "trip_0","trip_1",..: 1 2 63 74 85 96 107 11
##  $ vesselName            : Factor w/ 63 levels "vessel_0","vessel_1",..: 1 2 13 24 35 46 57
##  $ timeStart             : chr  "2019-06-29 15:30:33" "2019-11-24 19:02:53" "2019-04-02 18:3
##  $ timeEnd               : chr  "2019-06-30 21:50:58" "2019-11-25 20:50:24" "2019-04-04 04:3
##  $ longitudeStart        : num  8.53 8.52 8.52 8.53 8.56 ...
##  $ longitudeEnd          : num  4.65 4.65 4.67 4.66 4.68 ...
##  $ latitudeStart         : num  50.1 50.1 50.1 50.1 50.1 ...
##  $ latitudeEnd           : num  51.8 51.8 51.8 51.8 51.8 ...
##  $ currentSpeedOverGround: num  6.2 0 0 1.6 7.3 0 7 3.3 8 6.6 ...
##  $ timestampEta          : chr  "2019-05-27 19:09:00" "2019-11-21 01:00:00" "2019-11-30 00:0
##  $ destination           : Factor w/ 58 levels "","ANDERNACH",..: 20 10 48 23 52 12 1 23 25
##  $ typeOfShipId          : num  79 80 89 79 79 80 79 79 80 70 ...
##  $ length                : num  135 110 105 135 105 110 136 135 110 105 ...
##  $ width                 : num  12 11 12 12 10 12 12 12 12 10 ...
##  $ draught               : num  2.4 2 2.8 0.2 2.5 2.4 NA 0.2 NA 2.5 ...
##  $ dimA                  : num  120 98 95 120 16 110 136 120 110 85 ...
##  $ dim                   : num  15 12 10 15 89 0 0 15 0 20 ...
##  $ dimC                  : num  12 7 8 4 10 12 12 4 5 2 ...
##  $ dimD                  : num  0 4 4 8 0 0 0 8 7 8 ...
##  $ timeEnd_num           : num  1.56e+09 1.57e+09 1.55e+09 1.57e+09 1.57e+09 ...
##  $ timeStart_num         : num  1.56e+09 1.57e+09 1.55e+09 1.57e+09 1.57e+09 ...
```

```
## Struktur tripsRaw:
##
## 'data.frame':    21113 obs. of  23 variables:
##  $ X.1                  : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ X                    : int  0 1 2 3 4 5 6 7 8 9 ...
##  $ tripName             : Factor w/ 150 levels "trip_0","trip_1",..: 41 41 41 41 41 41 41 4
##  $ vesselName           : Factor w/ 63 levels "vessel_0","vessel_1",..: 47 47 47 47 47 47 47
##  $ timestampPosition    : chr  "2019-01-21 20:10:33" "2019-01-21 20:20:51" "2019-01-21 20:3
##  $ speedOverGround      : num  9.2 9.2 7.6 1 0.5 0 5.7 9.4 9.5 9.5 ...
##  $ courseOverGround     : num  242 257 255 273 NA ...
##  $ longitude            : num  8.44 8.41 8.36 8.34 8.34 ...
##  $ latitude             : num  50 50 50 50 50 ...
##  $ timestampVoyage      : chr  "2019-01-18 09:21:13" "2019-01-18 09:21:13" "2019-01-21 20:2
##  $ destination          : Factor w/ 133 levels "","ANDERNACH",..: 129 129 74 74 74 74 74 74
##  $ timestampEta         : chr  "2019-01-11 07:21:00" "2019-01-11 07:21:00" "2019-01-11 07:2
##  $ typeOfShipId         : int  89 89 89 89 89 89 89 89 89 89 ...
##  $ length               : int  110 110 110 110 110 110 110 110 110 110 ...
##  $ width                : int  12 12 12 12 12 12 12 12 12 12 ...
##  $ draught              : num  1 1 1 1 1 1 1 1 1 1 ...
##  $ dimA                 : int  98 98 98 98 98 98 98 98 98 98 ...
##  $ dim                  : int  12 12 12 12 12 12 12 12 12 12 ...
##  $ dimC                 : int  4 4 4 4 4 4 4 4 4 4 ...
##  $ dimD                 : int  8 8 8 8 8 8 8 8 8 8 ...
##  $ distanceAchieved     : num  7.03 9.07 11.23 12.12 12.38 ...
##  $ waterLocksPassed     : int  1 1 1 1 1 1 2 2 2 2 ...
##  $ timestampPosition_num: num  1.55e+09 1.55e+09 1.55e+09 1.55e+09 1.55e+09 ...
```

In den naechsten Abschnitten werden ausgewaehlte Variablen naeher untersucht.

1.2.2. Untersuchung ausgewaehlter Variablen

a) Destinationen Bei der visuellen Datenexploration ist aufgefallen, dass die Destinationen bei TripsRaw zwischenzeitlich geaendert wurden. Dementsprechend werden die Destinationen von tripsAggregated und tripsRaw naeher untersucht und verglichen, um deren Verlaesslichkeit zu pruefen:

```
tripsAggregated_Dest = left_join(tripsAggregated,
                                 tripsRaw[,c("tripName", "destination")],
                                 by = "tripName")

tripsAggregated_Dest2 = distinct(tripsAggregated_Dest, tripName,
                                 destination.x, destination.y,
                                 .keep_all = TRUE)

#Beispielhafter Auszug
kable(head(tripsAggregated_Dest2[,c("tripName", "vesselName",
                                    "timeStart", "timeEnd",
```

```
                                    "longitudeStart","longitudeEnd",
                                    "destination.y")]))
```

tripName	vesselName	timeStart	timeEnd	longitudeStart	longitudeEnd	destination.y
trip_0	vessel_0	2019-06-29 15:30:33	2019-06-30 21:50:58	8.530362	4.653430	DORDRECHT
trip_1	vessel_1	2019-11-24 19:02:53	2019-11-25 20:50:24	8.523685	4.650205	DEFAM03902TERMC00187
trip_1	vessel_1	2019-11-24 19:02:53	2019-11-25 20:50:24	8.523685	4.650205	NLRTM00116LOCKB00412
trip_1	vessel_1	2019-11-24 19:02:53	2019-11-25 20:50:24	8.523685	4.650205	NLRTM00116LOCKB00410
trip_1	vessel_1	2019-11-24 19:02:53	2019-11-25 20:50:24	8.523685	4.650205	NLRTM00116LOCKB0041L
trip_2	vessel_2	2019-04-02 18:33:23	2019-04-04 04:30:16	8.523333	4.668117	NVT

Erkenntnis der visuellen Datenexploration von tripsAggregated_Dest: Die Destinationen sind unterschiedlich bei tripsAggregated und tripsRaw bzw. unvollstaendig bei tripsAggregated. Hier scheint immer die zu Beginn der Route eingegebene Zieldestination erfasst worden zu sein. Diese stimmt jedoch nicht immer mit dem tatsaechlichen Ziel ueberein. Darauf deuten auch die longitude-Werte hin. Um dies genauer zu untersuchen, werden nun die doppelten Eintraege geloescht, damit uebersichtlich dargestellt werden kann bei welchen Trips die Zieldestinationen im Verlaufe des Trips geaendert wurden. Dies ist in der oben stehenden Tabelle zu erkennen.

b) Sendefrequenzen Fuer ein naeheres Verstaendnis der Datenerfassung in der Binnenschifffahrt wird die Sendefrequenz der Schiffe untersucht. Die angewandte Schleife soll lediglich einen Ueberblick verschaffen. Man muss beachten, dass diese nicht zwischen unterschiedlichen Trips / Schiffen unterscheidet.

```
#Sendefrequenz jedes Schiffes bestimmen und ueber einen Trip:
for (i in 0:length(tripsRaw$timestampPosition)) {
  tripsRaw$sendingFreq_in_Min[i] = difftime(tripsRaw$timestampPosition[i+1],
                                            tripsRaw$timestampPosition[i])
}
```

Erkenntnis: Es werden recht regelmaeszig in kurzen Zeitabstaenden von ca. 10-20min Daten erfasst. Dies deutet darauf hin, dass auch waehrend eines Stopps weiter Daten erfasst werden.

c) Schiffstyp Die Daten enthalten eine typeOfShipId, die nur in Kombination mit weiteren Informationen eine Bedeutung fuer die Schifffahrt hat: die ID ist nicht so aussagekraeftig, der Schiffstyp ist wichtig. Deswegen wird dieser den beiden Tabellen hinzugefuegt.

```
#Schifftyp bestimmen
tripsAggregated = merge(tripsAggregated, shiptype, by="typeOfShipId")
tripsRaw = merge(tripsRaw, shiptype, by="typeOfShipId")
```

```r
# Format anpassen:
tripsAggregated$shiptype = as.character(tripsAggregated$shiptype)#
tripsRaw$shiptype = as.character(tripsRaw$shiptype)

#Uebergeordnete Kategorie extrahieren
tripsAggregated$isCargo= grepl("Cargo", tripsAggregated$shiptype, fixed= TRUE)*1
tripsAggregated$isTanker = grepl("Tanker", tripsAggregated$shiptype, fixed= TRUE)*1
tripsAggregated$isHazardous = grepl("Hazardous", tripsAggregated$shiptype, fixed= TRUE)*1

#Kategorie als String, da fuer Visualisierung benoetigt
tripsAggregated$shiptype_category = as.factor(ifelse(tripsAggregated$isCargo==1,
                                                     "Cargo", "Tanker"))


cat("Nachfolgend werden die Schiffstypen je vesselName untersucht: \n")
```

```
## Nachfolgend werden die Schiffstypen je vesselName untersucht:
```

```r
kable(head(distinct(tripsAggregated[order(tripsAggregated$isTanker, decreasing = TRUE),
                                    c("vesselName","isCargo", "isTanker", "isHazardous")])))
```

vesselName	isCargo	isTanker	isHazardous
vessel_30	0	1	0
vessel_15	0	1	0
vessel_5	0	1	0
vessel_51	0	1	0
vessel_45	0	1	0
vessel_13	0	1	0

```r
#Fuer Tripsraw
for(j in 1:nrow(tripsRaw)){
  tripsRaw[j, "isCargo"] = tripsAggregated[
    which(tripsAggregated$tripName == tripsRaw[j, "tripName"]), "isCargo"]
  tripsRaw[j, "isTanker"] = tripsAggregated[
    which(tripsAggregated$tripName == tripsRaw[j, "tripName"]), "isTanker"]
  tripsRaw[j, "isHazardous"] = tripsAggregated[
    which(tripsAggregated$tripName == tripsRaw[j, "tripName"]), "isHazardous"]
}
```

Aus der obigen Tabelle kann man entnehmen das ausschlieszlich Tanker Gefahrgut geladen haben koennen.

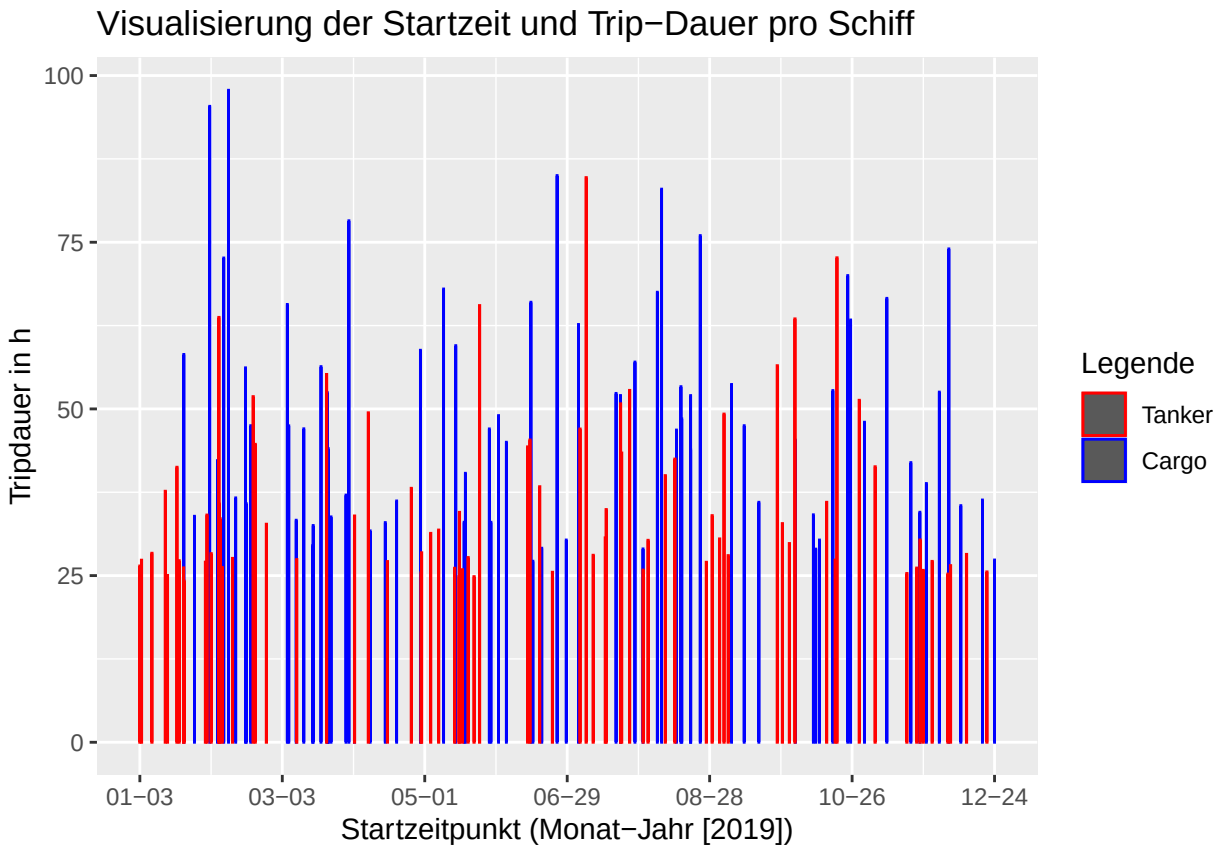
d) Trip Dauer Fuer ein naeheres Verstaendnis der Trips der Binnenschiffe wird die Trip-Dauer untersucht:

```r
#Zeit jedes Trips in tripsAggregated bestimmen
tripsAggregated$tripTime = difftime(tripsAggregated$timeEnd,
                                    tripsAggregated$timeStart, units = "hours")

#Bestimmung der durchschnittlichen Dauer eines Trips
tripsAggregated$tripsMeaninh = mean(tripsAggregated$tripTime)

cat("Die durschnittliche Trip-Dauer betraegt:",
    round(mean(tripsAggregated$tripTime),2),"Stunden. \n")
```

```
## Die durschnittliche Trip-Dauer betraegt: 42.3 Stunden.
```

```r
#Zeit jedes Trips in tripsAggregated bestimmen
tripsAggregated$tripTime = difftime(tripsAggregated$timeEnd,
                                    tripsAggregated$timeStart, units = "hours")
tripsAggregated$tripTime = as.numeric(tripsAggregated$tripTime)

#Nach Startdatum sortieren
tripsAggregated = tripsAggregated[order(tripsAggregated$timeStart), ]

#Visualisierung vorbereiten
#Extrahieren der Zeit von TimeStart
tripsAggregated$timestartonly <- format(tripsAggregated$timeStart, format = "%H:%M:%S")

#Erstellen separater Triptime Tabelle
tripTimes = data.frame(seq (min(tripsAggregated$timeStart_num),
                            max(tripsAggregated$timeStart_num),
                            (30654280/6)))
colnames(tripTimes) = c("uhrzeitNumeric")
tripTimes$uhrzeitRegular = as.POSIXct(tripTimes$uhrzeitNumeric,
                                      origin = '1970-01-01', tz = "GMT", )
tripTimes$uhrzeitRegularonly <- format(tripTimes$uhrzeitRegular, format = "%m-%d")

#Visualisierung der realisierten Tripdauer von Cargo- und Tankerschiffen
ggplot(data = tripsAggregated, aes(xmin = min(tripsAggregated$timeStart_num),
                                   xmax = max(tripsAggregated$timeStart_num)))+
  #Graph Cargo
   geom_col(data=subset(tripsAggregated, isCargo==1, c(timeStart_num, tripTime)),
            aes(x=timeStart_num,
                y=tripTime,
                colour = 'Cargo'))+
  #Graph Tanker
   geom_col(data=subset(tripsAggregated, isTanker==1, c(timeStart_num, tripTime)),
            aes(x=timeStart_num,
                y=tripTime,
                colour = 'Tanker'))+
  #Titel hinzufuegen
  ggtitle("Visualisierung der Startzeit und Trip-Dauer pro Schiff")+
```

```
  #Farben anpassen
  scale_color_manual(breaks = c("Tanker", "Cargo"), values = c("red", "blue"))+
  #Beschriftung der X-Achse
  xlab("Startzeitpunkt (Monat-Jahr [2019])")+
  scale_x_continuous(breaks = seq(min(tripsAggregated$timeStart_num),
                                  max(tripsAggregated$timeStart_num),
                                  (30654280/6)),
                     labels = tripTimes$uhrzeitRegularonly) +
    #Beschriftung der Y-Achse
  ylab("Tripdauer in h")+
  #Beschriftung der Legende
  labs(colour = "Legende")
```




```
#Zeitreihenanalyse
#Modell mit automatischen Parametern erstellen
model_aggregated = ets(tripsAggregated$timeEnd_num, model="ZZZ")
# Interpretation: Fehler, Trend, Saisonalitaet
#1 Ausggabe der Zusammenfassung
cat("Das Modell mit automatisch gewaehlten Parametern kann wie folgt
      zusammegefasst werden \n")
```

```
## Das Modell mit automatisch gewaehlten Parametern kann wie folgt
##       zusammegefasst werden
```

```
summary(model_aggregated)
```

```
## ETS(A,A,N)
##
## Call:
##  ets(y = tripsAggregated$timeEnd_num, model = "ZZZ")
##
##   Smoothing parameters:
##     alpha = 0.8875
##     beta  = 1e-04
##
##   Initial states:
##     l = 1546710682.1696
##     b = 200128.5409
##
##   sigma:  196530
##
##      AIC     AICc      BIC
## 4414.112 4414.529 4429.165
##
## Training set error measures:
##                     ME     RMSE      MAE           MPE        MAPE      MASE
## Training set 4131.092 193891.9 155118.1 0.0002534846 0.009943301 0.7079628
##                      ACF1
## Training set -0.003874301
```

```
cat("Es kann ein additiver Trend, jedoch keine Saisonalitaet identifiziert werden.
Das kann dadurch erklaert werden, dass ausschliesslich Daten aus 2019 vorliegen.
Es kann also eine Saisonalitaet existieren, die mit den vorliegenden Daten
nicht identifiziert werden kann. \n")
```

```
## Es kann ein additiver Trend, jedoch keine Saisonalitaet identifiziert werden.
## Das kann dadurch erklaert werden, dass ausschliesslich Daten aus 2019 vorliegen.
## Es kann also eine Saisonalitaet existieren, die mit den vorliegenden Daten
## nicht identifiziert werden kann.
```

```
#2 Ausgabe der urspruenglichen Zeitreihe UND
#3 Ausgabe der Residuen
kable(head(data.frame(model_aggregated$x,model_aggregated$residuals),
      col.names = c("Urspruengliche Zeitreihe", "Residuen")))
```

model_aggregated.x	model_aggregated.residuals
1546617131	-293679.7
1546679404	-170868.9
1547056497	157786.0
1547569984	331142.1
1547601860	-130997.2
1548001432	184715.3

Um die Eintraege in der Tabelle *tripsRaw* besser in den Verlauf eines Trips einordnen zu koennen wird als Zielgroesze die verbleibende Tripdauer berechnet.

```
# tripsRaw
for(i in 1:nrow(tripsRaw)){
# identify corresponding entry in tripsRaw
trip = tripsAggregated[which(tripsRaw$tripName[i] == tripsAggregated$tripName),]
trip = droplevels(trip)

# Identify timeEnd and complete trip time
tripsRaw$tripTime[i] = trip$tripTime
tripsRaw$timeEnd[i] = trip$timeEnd
tripsRaw$tripTime_num[i] = as.numeric(trip$tripTime)
tripsRaw$timeEnd_num[i] = trip$timeEnd_num
}

#Zielgroesze
tripsRaw$remainingTripTime = tripsRaw$timeEnd_num- tripsRaw$timestampPosition_num
```

2. Datenaufbereitung

2.1. Zuordnung des Wasserstands zu den Trip-Abschnitten aus tripsRaw

```
#Verbinden der Daten waterLevels und waterLevelsStations ueber die ID der Messstationen
#Nutzung der Subset-Funktion, um jeweilige Index-Angaben zu loeschen
waterLevelsTime= subset(merge(waterLevels, waterLevelsStations, by = "id"),
                        select = -c(X.x, X.y))

#Zuordnung Wasserstand zu Trip-Abschnitten

#1) Zuordnung der naechsten Messstation
#Hilfsdataframe erstellen fuer For-Schleife
HilfsDf = data.frame(waterLevelsStations$id)

#Erstellung der Eintraege in TripsRaw
for (i in 1:nrow(tripsRaw)) {
  #Berechnung der Distanzen zu jeder Messstation und temporaere Speicherung im HilfsDf
  for(j in 1:nrow(waterLevelsStations)){
  dx = 71.5 * (tripsRaw$longitude[i]- waterLevelsStations$longitude[j])
  dy = 111.3 * (tripsRaw$latitude[i] - waterLevelsStations$latitude[j])
  HilfsDf$Distance[j] = sqrt(dx * dx + dy * dy)
  }
  #Wahl der geringsten euklidischen Entfernung
  tripsRaw$Entfernung_Naechste_Messstation[i] = min(HilfsDf$Distance)
```

```
  #Wahl der Messstation mit der geringsten euklidischen Entfernung
  #Bugfix
  minValue = as.character(HilfsDf$waterLevelsStations.id[which.min(HilfsDf$Distance)])
  tripsRaw$Naechste_Messstation[i] = minValue
}

#2) Zuordnung des naechsten zeitlichen Abschnitts von Wasserstandsmessung
#   und Positionserfassung des Schiffs

# Ansatz1 mit zu hoher Rechenzeit (ueber 20min):

# Previous_value = 999999 #muss hoch angesetzt werden,
# weil sonst x immer niedriger sein wird
# for (i in 1:nrow(tripsRaw)) {
#   for(j in 1:nrow(waterLevelsTime)){
#     if(tripsRaw$Naechste_Messstation[i] == waterLevelsTime$id[j]){
#       x = difftime(tripsRaw$timestampPosition[i], waterLevelsTime$measuretime[j])
#       if(x < Previous_value){
#         tripsRaw$StationTimeOrientation[i] = x
#         Previous_value = x
#       }else{
#         tripsRaw$StationTimeOrientation[i] = Previous_value
#       }
#     }
#   }
# }

#Ansatz2 zur Senkung der Rechenzeit:
#Pruefung wann an den einzelnen Stationen gemessen wird
waterLevelsTime$TimeStamp = waterLevelsTime$measuretime
waterLevelsTime = separate(waterLevelsTime, TimeStamp,
                           c("DatePosition", "TimePosition"), sep = " " )
waterLevelsTime = arrange(waterLevelsTime, measuretime)
waterLevelsTime_Explore = distinct (waterLevelsTime, TimePosition, id, .keep_all = FALSE)
kable(arrange(waterLevelsTime_Explore, id))
```

id	TimePosition
FTS49972718333	05:00:00
FTS49972718333	13:00:00
FTS50083051667	05:00:00
FTS50083051667	13:00:00
FTS5010648	05:00:00
FTS5010648	13:00:00
FTS50358843333	05:00:00
FTS50358843333	13:00:00
FTS50614718333	05:00:00
FTS50614718333	13:00:00

id	TimePosition
FTS50937113333	05:00:00
FTS50937113333	13:00:00
FTS51226176667	05:00:00
FTS51226176667	13:00:00
FTS51646538333	05:00:00
FTS51646538333	13:00:00
FTS51827628333	05:00:00
FTS51827628333	13:00:00
FTS51900378333	06:00:00
FTS51950926667	06:00:00

```
cat("Erkenntnis: Bis auf zwei Messstationen (FTS51900378333 - Station10,
FTS51950926667 - Station11) wird immer um 5Uhr und um 13Uhr gemessen,
bei den anderen beiden nur um 6Uhr.")
```

```
## Erkenntnis: Bis auf zwei Messstationen (FTS51900378333 - Station10,
## FTS51950926667 - Station11) wird immer um 5Uhr und um 13Uhr gemessen,
## bei den anderen beiden nur um 6Uhr.
```

```
#Hinzufuegen Spalte Wasserstand
tripsRaw$WaterLevel = NULL

#Erstellen von Subsets fuer alle Messstationen, da dies den Abgleich der Messtationen
#innerhalb der groesseren Datensaetze spart und die Rechenzeit deutlich senkt
#Ziel: Abgleich der Wasserstaende an einer Messstation und den Wasserstaenden
#auf dem Routenverlauf ueber die Messstation

#1. Schleife geht die Messstationen nacheinander durch
for(k in 1:nrow(waterLevelsStations)){

  #Subset-Gruppe: Filtern aller Trips, die entlang einer bestimmten Messstation verliefen
  SubsetTrip = subset(tripsRaw, Naechste_Messstation == waterLevelsStations$id[k])
  SubsetTrip$TimeStamp = SubsetTrip$timestampPosition
  SubsetTrip = separate(SubsetTrip, TimeStamp,
                        c("DatePosition", "TimePosition"),
                        sep = " ")

  #Subset-Gruppe: Filtern aller Wasserstaende, die an einer Messstation aufgetreten sind
  SubsetStation = subset(waterLevelsTime, id == waterLevelsStations$id[k])
  SubsetStation$TimeStamp = SubsetStation$measuretime
  SubsetStation = separate(SubsetStation, TimeStamp, c("DatePosition", "TimePosition"),
                           sep = " " )
  SubsetStation = arrange(SubsetStation, measuretime)

  for (i in 1:nrow(SubsetTrip)){
    for(j in 2:(nrow(SubsetStation)-2)){
```

```
      if(SubsetStation$DatePosition[j] == SubsetTrip$DatePosition[i]){
        #Startpunkt fuer den Abgleich: selber Tag
        a = abs(difftime(SubsetTrip$timestampPosition[i],
                         SubsetStation$measuretime[j-1]))
        #Abgleich Vortag Messung 13Uhr
        b = abs(difftime(SubsetTrip$timestampPosition[i],
                         SubsetStation$measuretime[j]))
        #Abgleich selber Tag, Messung 5Uhr
        c = abs(difftime(SubsetTrip$timestampPosition[i],
                         SubsetStation$measuretime[j+1]))
        #Abgleich selber Tag, Messung 13Uhr
        d = abs(difftime(SubsetTrip$timestampPosition[i],
                         SubsetStation$measuretime[j+2]))
        #Abgleich naechster Tag, Messung 5Uhr

        #Zuordnung der Messung mit dem geringsten zeitlichen Abstand
        if(a == min(a, b, c, d)){
          SubsetTrip$WaterLevel[i] = SubsetStation$value[
            SubsetStation$measuretime == SubsetStation$measuretime[j-1]]
        }else if (b == min(a, b, c, d)){
          SubsetTrip$WaterLevel[i] = SubsetStation$value[
            SubsetStation$measuretime == SubsetStation$measuretime[j]]
        }else if (c == min (a, b, c, d)){
          SubsetTrip$WaterLevel[i] = SubsetStation$value[
            SubsetStation$measuretime == SubsetStation$measuretime[j+1]]
        }else{
          SubsetTrip$WaterLevel[i] = SubsetStation$value[
            SubsetStation$measuretime == SubsetStation$measuretime[j+2]]
        }

      }
    }
  }
  #Nummerierung bzw. Benennung der einzelnen Subsets
  assign(paste("Station", k, sep = ""), SubsetStation)
  assign(paste("tripsRaw_Station", k, sep = ""), SubsetTrip)
}

#Verbinden der Subsets
tripsRaw = rbind(tripsRaw_Station1,tripsRaw_Station2, tripsRaw_Station3,
                 tripsRaw_Station4, tripsRaw_Station5, tripsRaw_Station6,
                 tripsRaw_Station7, tripsRaw_Station8, tripsRaw_Station9,
                 tripsRaw_Station10, tripsRaw_Station11)

#Test mit Station FTS5010648, 2019-07-04 und 2019-07-05,
#Wechsel von 152 zu 153 ersichtlich
```

3. Feature Engineering

Im Folgenden sollen verschiedene Features erstellt werden, die in die Modellierung mit einflieszen sollen:

1. Der Wasserstand

Annahme: Der Wasserstand kann die Geschwindigkeit des Schiffes beeinflussen und eventuell eine Unterbrechung der Fahrt erzwingen.

2. Feriendaten

Annahme: Werksschlieszungen zu Ferienzeiten beeinflussen die Verkehrsdichte und somit die Trip-Dauer.

3. Der Schiffstyp

Annahme: Dieser kann die Geschwindigkeit des Schiffes aufgrund der Wendigkeit, Schwere und weiteren individuellen Eigenschaften des Schiffstyps beeinflussen.

4. Euklidische Distanz

Annahme: Die Distanz reduziert die Positionsangabe von zweidimensionalen Koordinaten auf die eindimensionale Entfernung.

5. Distanz Flussverlauf

Annahme: Die euklidische Distanz beschreibt nicht die tatsaechlich zurueckgelegte Strecke. Diese wird stattdessen ueber die Koordinaten von Main, Rhein und Waal ermittelt. Die tatsaechliche Entfernung beeinflusst die Ankunftszeit, insbesondere bei der *-to-Rotterdam Prognose.

6. Anzahl der passierten Schleusen

Annahme: Das Passieren einer Schleuse beeinflusst die Dauer und ETA des Trips, da Wartezeiten an den Schleusen entstehen bis das Schiff die Fahrt fortsetzen kann.

7. Anzahl der Stopps

Annahme: Die Anzahl der Stopps, bspw. zum Laden, beeinflusst durch Unterbrechungen des Trips die ETA des Trips.

3.1. Wasserstand

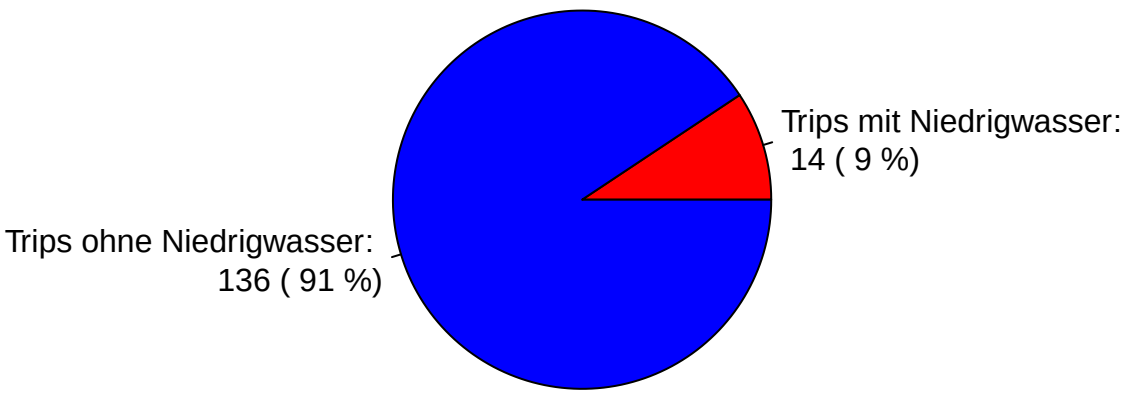
Mithilfe weiterer Datenquellen (https://www.elwis.de/DE/dynamisch/gewaesserkunde/wasserstaende/index.php?target=2&fs=RHEINGEBIET) wurden die Werte GlW (Gleichwertigen Wasserstand) und HSW (Hoechster Schifffahrtswasserstand) ermittelt Wasserlevel unter dem GLW werden als Niedrigwasser interpretiert, Wasserlevel ueber dem HSW als Hochwasser.

```
# 1) Daten der Wasserstationen anpassen mit HSW/GlW
# aus elwis tabelle + wasserstationen gps, manuell eintragen
waterLevelsStations = read.csv("water_levels_stations_adjusted.csv")

# 2) Feature fuer jeden Pegel: ist Hochwasser? ist Niedigwasser?
for(i in 1:nrow(tripsRaw)){
```

```
  tripsRaw[i, "high_water"] = tripsRaw[i, "WaterLevel"] > waterLevelsStations[
    which(tripsRaw[i, "Naechste_Messstation"]==waterLevelsStations$id), "HSW"]
  tripsRaw[i, "low_water"] = tripsRaw[i, "WaterLevel"] < waterLevelsStations[
    which(tripsRaw[i, "Naechste_Messstation"]==waterLevelsStations$id), "GlW"]
}
# Datenexploration: kein Hochwasser aufgezeichnet
# 3) Feature in Aggregated: Niedrigwasser
tripsAggregated$low_water = FALSE
low_water_points = subset(tripsRaw, low_water, select = tripName)
cat("Feststellung: bei " ,  nrow(low_water_points),
    " Eintraegen in tripsRaw gibt es Niedrigwasser.")
```

```
## Feststellung: bei  147  Eintraegen in tripsRaw gibt es Niedrigwasser.
```

```
for(j in 1:nrow(low_water_points)){
  tripsAggregated[which(tripsAggregated$tripName ==
                          low_water_points[j, "tripName"]),
                  "low_water"] = TRUE
}

#Visualisierung
#Erstellen eines Kreisdiagrammes zur Visualisierung der Trips mit und ohne Niedrigwasser
No_Trips_LowWater = sum(tripsAggregated$low_water)
No_Trips_NoLowWater = sum(!tripsAggregated$low_water)
slices = c(No_Trips_LowWater, No_Trips_NoLowWater)
pct = round(slices/sum(slices)*100)
slice_name = c("Trips mit Niedrigwasser: \n", "Trips ohne Niedrigwasser: \n")
lbls = paste(slice_name, slices, "(", pct, "%)")
pie(slices, labels = lbls, col = c("red", "blue"),
    main = "Kritischer Wasserstand bei Trips")
```



```
cat("Es konnten ", No_Trips_LowWater, " trips mit Niedrigwasser identifiziert werden. \n")
```

```
## Es konnten  14  trips mit Niedrigwasser identifiziert werden.
```

```
# Zusaetzliches Feature: Verhaeltnis von Wasserlevel zu Tiefgang
tripsRaw$WaterLevelRatio = tripsRaw$WaterLevel/tripsRaw$draught
```

3.2. Feriendaten

In der Datei Feriendaten wurden Daten zu Ferienzeiten fuer Deutschland und die Niederlande gesammelt. Im Folgenden werden jedoch lediglich die Daten der Weihnachts- sowie Sommerferien betrachtet, da zu diesen Zeiten von Werksschlieszungen auszugehen ist. Diese koennen einen Einfluss auf den Verkehr der Binnenschiffe haben, da weniger Material transportiert werden muss. Daher wird hier eine Variable erstellt, die ueber TRUE bestaetigt, dass waehrend eines Trips Ferien in den Niederlande und/oder Deutschland waren.

```
#Bestimmen des Ferienzeitraums ueber das fruehste und spaeteste Datum
#unter der Annahme, dass zu diesen Zeiten Produktionsstaetten geschlossen werden
Vacation_Summary = data.frame("Vacation_Start" = c(min(vacation$`Sommerferien Start`),
                              min(vacation$`Weihnachtsferien 19/20 Start`),
                              min(vacation$`Weihnachtsferien 18/19 Start`)),
                              "Vacation_End"= c(max(vacation$`Sommerferien Ende`),
                              max(vacation$`Weihnachtsferien 19/20 Ende`),
                              max(vacation$`Weihnachtsferien 18/19 Ende`)))

#Pruefung, ob der Zeitpunkt des Trips mit dem Ferienzeitraum uebereinstimmt
tripsRaw$Vacation = ((tripsRaw$DatePosition >= Vacation_Summary$Vacation_Start[1] &
                        tripsRaw$DatePosition <= Vacation_Summary$Vacation_End[1]) |
                       (tripsRaw$DatePosition >= Vacation_Summary$Vacation_Start[2] &
                          tripsRaw$DatePosition <= Vacation_Summary$Vacation_End[2]))

#zu tripsAggregated hinzufuegen:
for(i in 1:nrow(tripsAggregated)){
  tripsAggregated[i, "Vacation"] = tripsRaw[which(tripsAggregated[i, "tripName"] ==
                                                    tripsRaw$tripName), "Vacation"][1]
}
```

3.4. euklidische Distanz

```
#Distanz euklidisch bestimmen
#Wenn man Länge und Breite in Grad angibt ergibt sich die Entfernung in Kilometern
# Die Konstante 71.5 beschreibt den durchschnittlichen Abstand zwischen zwei Längengeraden
dx = 71.5 * (tripsAggregated$longitudeStart - tripsAggregated$longitudeEnd)
# Die Konstante 111.3 beschreibt dabei den Abstand zwischen zwei Breitengraden
dy = 111.3 * (tripsAggregated$latitudeStart - tripsAggregated$latitudeEnd)
```

```
tripsAggregated$Distanceinkm = sqrt(dx * dx + dy * dy)
tripsAggregated$Distanceinnmi = tripsAggregated$Distanceinkm * 0.54
```

3.5. Distanz Flussverlauf

1) Flussdaten Deutschland

```
river_data_germany = subset(river_data_germany,
                            select = c(latitude, longitude,
                                       coordinates, FID, BWASTR_ID, NAME))
river_data_germany = river_data_germany[river_data_germany$NAME == "Rhein, Hauptstrecke" |
                                          river_data_germany$NAME == "Main, Hauptstrecke", ]
river_data_germany = droplevels(river_data_germany)

river_germany = data.frame(matrix(nrow = 0, ncol = length(colnames(river_data_germany))))
colnames(river_germany) = colnames(river_data_germany)

coordinates = as.character(river_data_germany[river_data_germany$FID == 542, "coordinates"])
coordinates_array = as.numeric(unlist(strsplit(coordinates, ',')))

# Da die Berechnung ggf. sehr lange dauern kann werden Informationen
# zum Stand der Berechnung angezeigt
cat("Computing river data germany... ")
```

```
## Computing river data germany...
```

```
#Aufteilen der Koordinaten, Reihenfolge durch Stichproben: longitude, latitude
for (i in 1:nrow(river_data_germany)) {
  # Alle Koordinaten dieses Eintrages lesen und aufteilen
  coordinates = as.character(river_data_germany[i, "coordinates"])
  coordinates_array = as.numeric(unlist(strsplit(coordinates, ',')))
  # Runden der Werte, hinzufuegen der einzigartien Werte zu einem neuen DF
  for (j in seq(1, by = 2, len = length(coordinates_array) / 2)) {
    rounded_latitude = round(coordinates_array[j + 1], 3)
    rounded_longitude = round(coordinates_array[j], 3)

    if (is.na(coordinates_array[j])) {
      next
    }
    number_of_rows = nrow(river_germany[river_germany$FID ==
                                          river_data_germany[i, "FID"] &
                                        river_germany$longitude == rounded_longitude &
                                        river_germany$latitude == rounded_latitude,])
    if (number_of_rows > 0) {
      next
    }
    new_row = c(
```

```
      rounded_latitude,
      rounded_longitude,
      paste(coordinates_array[j + 1], coordinates_array[j], sep =
              " "),
      river_data_germany[i, "FID"],
      river_data_germany[i, "BWASTR_ID"],
      as.character(river_data_germany[i, "NAME"])
    )
    river_germany[nrow(river_germany) + 1, ] = new_row
  }
}
rownames(river_germany) = 1:nrow(river_germany)
cat(" ... DONE \n")
```

```
##  ... DONE
```

```
# Durch Experimente: Fluss in relevante Abschnitte aufteilen
# Finde Koordinaten: Rhein/Bijlands Kanaal (Grenze Dtl/NL)
# Google Maps zeigt: latitude  51.840 , longitude  8.167
start_rhein = which(river_germany$latitude == 51.840 &
                                    river_germany$longitude == 6.167)
# Finde Koordinaten: Rhein/Main
# Google Maps zeigt:  latitude  49.995 , longitude  8.289
stop_rhein = which(river_germany$latitude == 49.995 &
                                  river_germany$longitude == 8.289)
# finde Koordinaten: Main/Rhein
# Google Maps zeigt:  latitude  49.994 , longitude  8.293
start_main = which(river_germany$latitude == 49.994 &
                                 river_germany$longitude == 8.293)
# finde Koordinaten: Frankfurt
# Google Maps zeigt:  latitude 49.995 , longitude 8.289
# Identifiziert in tripsAggregated: 50.076 8.523
stop_main = which(river_germany$latitude == 50.076 &
                                  river_germany$longitude == 8.523)

#Fehler in der Anordnung der Daten gefunden, Zwischenpunkte erforderlich
#Aus der visuellen Analyse folgt:
intermediate =which(river_germany$latitude == 50.654 &
                                  river_germany$longitude == 7.208)

river = rbind(river_germany[stop_main:start_main,],
              river_germany[stop_rhein:intermediate[2],],
              river_germany[intermediate[1]:start_rhein,] )
rownames(river)=1:nrow(river)

cat("Der Flussverlauf kann eindimensional von Frankfurt bis Rotterdamm berechnet werden.
    Exemplarisch werden die ersten 6 Elemente gezeigt: \n")
```

```
## Der Flussverlauf kann eindimensional von Frankfurt bis Rotterdamm berechnet werden.
##     Exemplarisch werden die ersten 6 Elemente gezeigt:
```

```
kable(head(river))
```

latitude	longitude	coordinates	FID	BWASTR_ID	NAME
50.076	8.523	50.0755724050001 8.52293033700005	329	2901	Main, Hauptstrecke
50.075	8.523	50.074590782 8.52333389400007	329	2901	Main, Hauptstrecke
50.074	8.523	50.074376466 8.52344366900007	329	2901	Main, Hauptstrecke
50.074	8.524	50.0735107180001 8.52397176300008	329	2901	Main, Hauptstrecke
50.073	8.524	50.0728201480001 8.52444949500006	329	2901	Main, Hauptstrecke
50.073	8.525	50.0726156590001 8.52459095700004	329	2901	Main, Hauptstrecke

2) Flussdaten Niederlande

```
#Relevante Zeile 28, relevante Spalte "SHAPE"
river_string_netherlands = as.character(river_data_netherlands[28, "SHAPE"])
river_string_netherlands = substring(river_string_netherlands, 19,
                                     nchar(river_string_netherlands)-2)

river_netherlands = data.frame(matrix(nrow = 0, ncol = 3))
colnames(river_netherlands) = c("latitude", "longitude",  "NAME")

river_nl = data.frame(coordinates = unlist(strsplit(river_string_netherlands, ",")))
cat("Computing river data netherlands ... ")
```

```
## Computing river data netherlands ...
```

```
for(i in 1:nrow(river_nl)){
  coords = as.character(river_nl[i, "coordinates"])
  coords_array = as.numeric(unlist(strsplit(coords, " ")))
  if(!is.na(coords_array[1])){
  rounded_latitude = round(coords_array[1], 3)
  rounded_longitude = round(coords_array[2], 3)
  }else{
  rounded_latitude = round(coords_array[2], 3)
  rounded_longitude = round(coords_array[3], 3)
  }
  river_netherlands[i,] = c(rounded_latitude,rounded_longitude, "Waal")
}

#Finde Koordinaten Bijlands Kanaal/Rhein
#Dies entspricht dem ersten Eintrag, ist jedoch bereits in den deutschen
#Flussdaten enthalten. Der erste neue Eintrag ist in Reihe 3 zu finden

# Finde Koordinaten von Dordrecht/Rotterdam (destination)
# Google Maps zeigt:  latitude  51.8 , longitude  4.8
# Identifiziert in tripsAggregated: 51.889 4.619 = Reihe 942
```

```
for(i in 4:942){
  latitude = river_netherlands[i, "latitude"]
  longitude = river_netherlands[i, "longitude"]
  coordinates = paste(latitude," ", longitude)
  new_row = c(latitude, longitude, coordinates, NA, NA,
              "Waal/Boven Merwede/Beneden Merwede/Noord")
  river = rbind(river,new_row)
}
rownames(river) = 1:nrow(river)

#Konvertieren als Zahl
river$longitude = as.numeric(river$longitude)
river$latitude = as.numeric(river$latitude)

#Spalte distanceToFrankfurt initialisieren
river[1, "distanceToFrankfurt"] = 0

#Spalte distanceToFrankfurt fuellen
for (i in 2:nrow(river)) {
  reference_point = river[i, c("longitude", "latitude")]
  prev_point = river[i - 1, c("longitude", "latitude")]
  river[i, "distanceToFrankfurt"] = distHaversine(reference_point, prev_point)* 0.00054 +
                                    river[i - 1, "distanceToFrankfurt"]
  # konvertiere in nm (nautic miles)
}

# Distanzberechnung in tripsRaw
tripsRaw$distanceOutstanding  = river[
  nrow(river), "distanceToFrankfurt"] - tripsRaw$distanceAchieved
```

```
# Identifizierter Startpunkt in tripsAggregated: latitude 50.07 longitude 8.52
# --> 50.076 8.523 in river data

#Entfernung fuer tripsRaw berechnen
# ACHTUNG: UEBER 25 MIN LAUFZEIT!
distances = distm(tripsRaw[, c("longitude", "latitude")],
                  river[, c("longitude", "latitude")],
                  fun=distHaversine)
#tripsRaw$minDistance = rowMins(distances)
distances = data.frame(distances)
colnames(distances) = 1:nrow(river)

for (i in 1:nrow(tripsRaw)) {
  cat("iteration ", i , " \n")
  #min_distance = tripsRaw[i, "minDistance"]
  min_distance = min(distances[i,])
  #min_index = which(distances == min_distance, arr.ind = TRUE)
```

```
  #min_index = which.min(distances[i,])
  min_index = which.min(distances[i,])
  tripsRaw[i, "distanceAchieved"] = min_distance * 0.00054 +
    river[min_index, "distanceToFrankfurt"]
  tripsRaw[i, "waterLocksPassed"] = river[min_index, "numberWaterLocks"]
}


# Um Zeit zu sparen wird diese Berechnung gespeichert
# write.csv(tripsRaw, "adjusted_trips_raw.csv")
```

3.6. Anzahl der passierten Schleusen

Durch Recherche mit Hilfe von externen Quellen (https://atlas.wsv.bund.de/clients/desktop/?zoom=10&center=13.329587%2C52.519844&vl=wadaba%2Ctopplus_grau&route_option=bsf *und* https://www.elwis.de/DE/dynamisch/mvc/main.php?modul=schleuseninfo&choice=1&show_sperr=1&specialcontacts=34#w_34) konnte herausgefunden werden, das weder auf dem Rhein noch auf der Waal Schleusen in dem betrachteten Abschnitt liegen. Auf dem Main konnten zwei Schleusen (Kostheim, Eddersheim) identifiziert werden.

```
# identified start from tripsAggregated: latitude 50.07 longitude 8.52,
# which translates to 50.076 8.523 in river data

#coordinates of identified water locks
lock_kostheim = c(49.999,8.334)
lock_eddersheim = c(50.038, 8.477)

rownames(river)=1:nrow(river)
#inititalize
river$waterLock = 0
river[which(river$longitude == lock_kostheim[2] &
              river$latitude == lock_kostheim[1]), "waterLock"] = 1
river[which(river$longitude == lock_eddersheim[2] &
              river$latitude == lock_eddersheim[1]), "waterLock"] = 1

river[1, "numberWaterLocks"] = 0
for(i in 2:nrow(river)){
  river[i, "numberWaterLocks"] = river[i, "waterLock"] + river[i-1, "numberWaterLocks"]
}
```

3.7. Stopps

Zur Analyse der Stopps die Schiffe auf dem Trip vornehmen wurde die Variable SpeedOverGround als Grundlage genommen. Die Analyse erfolgt unter der Annahme dass ein Stopp zu dem Zeitpunkt startet an dem das erste Mal eine Geschwindigkeit von 0 kn beobachtet wird, und endet wenn das erste mal wieder eine Geschwindigkeit von über 0 kn beobachtet wird. Als Resultat wird ein

Dataframe gefüllt das je Zeile einen Stopp eines Schiffes enthält. Um weitere Informationen zu dem Stopp zu erlangen, wird die Stoppdauer und die Veränderung des Tiefgangs berechnet.

```
options(scipen=999) #Wissenschaftliche Notation ausstellen
#rm(stops) #stops aus dem Arbeitsspeicher loeschen

#rawTrips wieder wie im Original sortieren:
tripsRaw = tripsRaw[order(tripsRaw$X),]

#Dataframe "stops" initialisieren, dass je Zeile einen Stopp eines Schiffes speichert.
stops = data.frame(tripsRaw[1,])
stops$beginOfStop = ""
stops$endOfStop = ""
stops = stops[-c(1),]

for (i in 2:nrow(tripsRaw)) {
  #Boolsche Variablen fuer Fallunterscheidung berechnen:
  shipStopped = tripsRaw$speedOverGround[i] == 0.0
  sameTrip = tripsRaw$tripName[i] == tripsRaw$tripName[i-1]
  shipStoppedBefore = tripsRaw$speedOverGround[i-1] == 0.0

  #Errorhandling: Wenn eine der Bool Variablen NA ist, zur naechsten Iteration springen
  if (is.na(shipStopped+sameTrip+shipStoppedBefore)){next}

  #Fall 1:
  #Wenn das Schiff steht, es noch derselbe Trip ist und das Schiff vorher noch nicht stand:
  #Zeile fuer neuen Stopp hinzufuegen und Startzeit des Stopps eintragen
  if (shipStopped && sameTrip && !shipStoppedBefore) {
    stops[nrow(stops)+1,1:ncol(tripsRaw)] = tripsRaw[i,]
    stops$beginOfStop[nrow(stops)] = as.character(tripsRaw$timestampPosition[i])
    stops$draughtBeforeStop[nrow(stops)] = tripsRaw$draught[i]
    stops$idBeforeStop[nrow(stops)] = tripsRaw$X[i]
  }
  #Fall 2:
  #Wenn das Schiff nach dem Stillstand wieder losfaehrt und es noch derselbe Trip ist:
  #In Zeile des letzten Stopps Endzeit des Stopps eintragen.
  if (!shipStopped && sameTrip && shipStoppedBefore){
    stops$endOfStop[nrow(stops)] = as.character(tripsRaw$timestampPosition[i])
    stops$draughtAfterStop[nrow(stops)] = tripsRaw$draught[i]
    stops$idAfterStop[nrow(stops)] = tripsRaw$X[i]
  }
  #Fall 3:
  #Edgecase: Ein Trip startet im Stillstand.
  if (shipStopped && !sameTrip) {
    stops[nrow(stops)+1,1:ncol(tripsRaw)] = tripsRaw[i,]
    stops$beginOfStop[nrow(stops)] = as.character(tripsRaw$timestampPosition[i])
    stops$draughtBeforeStop[nrow(stops)] = tripsRaw$draught[i]
  }
```

```
  #Fall 4:
  #Edgecase: Ein Trip endet im Stillstand
  #Als Endzeit des Stopps wird die Zeit des letzten Datenpunkts des Trips genommen
  if (!shipStopped && !sameTrip && shipStoppedBefore){
    stops$endOfStop[nrow(stops)] = as.character(tripsRaw$timestampPosition[i-1])
    stops$draughtAfterStop[nrow(stops)] = tripsRaw$draught[i-1]
  }
}
#Dataframe Stops um Spalten und Zeilen bereinigen die nicht benoetigt werden:
stops[,c("X","timestampPosition","speedOverGround","courseOverGround",
         "timestampVoyage","draught","SchiffStoppt")]=list(NULL)
stops = stops[-c(1),]

#Laenge des Stopps berechnen:
stops$stopDuration = round(difftime(strptime(stops$endOfStop,
                                             "%Y-%m-%d %H:%M:%S"),
                                    strptime(stops$beginOfStop,
                                             "%Y-%m-%d %H:%M:%S"),
                                    units = "mins"),0)

#Veraenderung des Tiefgangs berechnen:
stops$ladevorgang = (stops$draughtAfterStop - stops$draughtBeforeStop) != 0
```

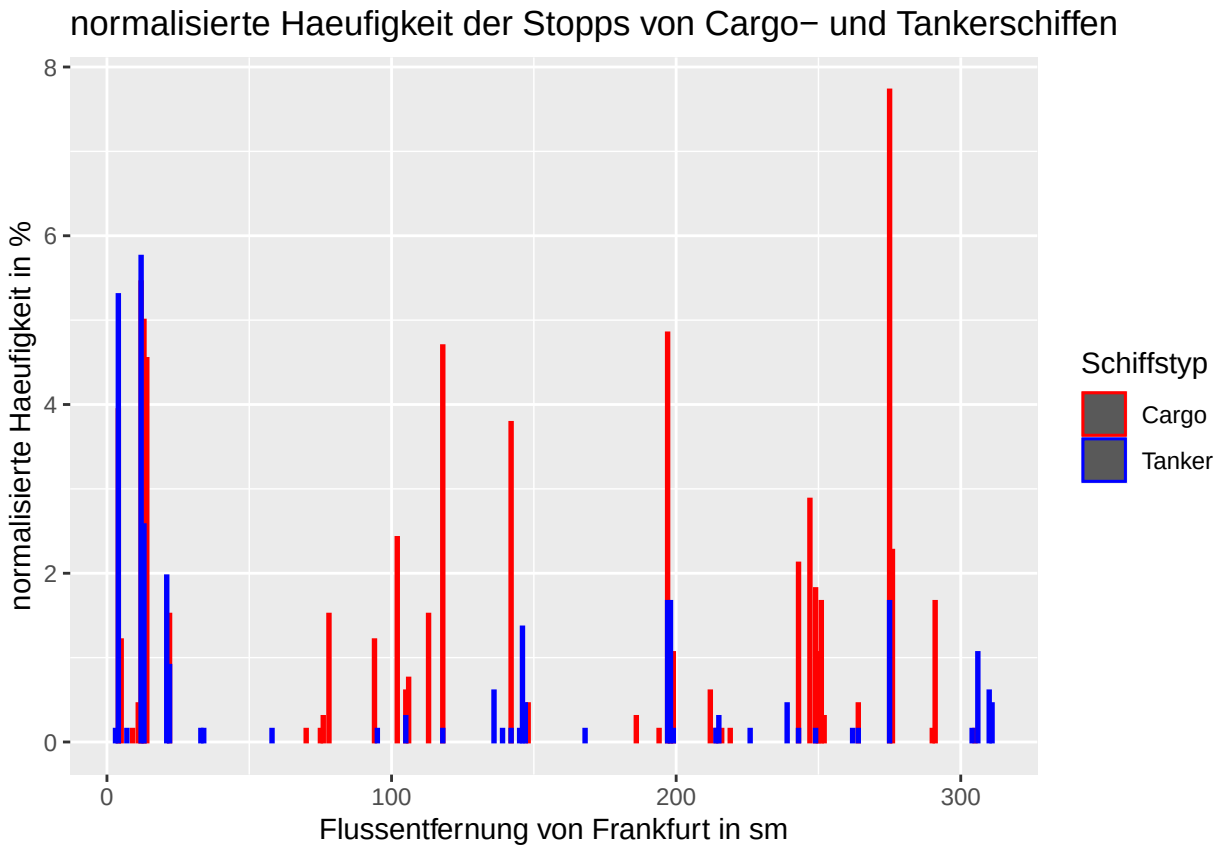
Im Anschluss werden die Stopps je Ort und Schiffstyp aggregiert um weitere Informationen daraus zu erlangen. Beim Schiffstyp wird der einfachheitshalber nur noch zwischen Cargo und Tanker unterschieden.

```
#Position runden:
#Mit Rundung auf keine Nachkommastellen wird jede Positionangabe einem ca. 1000 m
#groszen Flussabschnitt zugeordnet.
#Dies ist genau genug um einen Hafen, Schleuse oder eine Engstelle als Ursache
#fuer den Stopp zuordnen zukoennen, ermoeglicht aber ein geografisches Clustering.
stops$distanceAchieved = round(stops$distanceAchieved,0)

#Bei der Untersuchung soll nur zwischen Cargo- und Tankeschiff unterschieden werden.
#Daher wird dies hier zusammengefasst:
stops[grepl('Cargo', stops$shiptype, fixed = TRUE),c('shiptype')]='Cargo'
stops[grepl('Tanker', stops$shiptype, fixed = TRUE),c('shiptype')]='Tanker'


#Orte an denen gestoppt wird je Schiffstyp aggregieren mit Flussverlaufpositionsangabe:
stopLocationsCoordinates = aggregate(stopDuration~distanceAchieved + shiptype,
                                     data = stops,
                                     mean)
#normalisierte Haeufigkeit des Stopps:
stopLocationsOccurence = aggregate(tripName~distanceAchieved + shiptype,data = stops,NROW)
stopLocationsOccurence[,3] = stopLocationsOccurence[,3]/sum(stopLocationsOccurence[,3])*100
stopLocations = merge(stopLocationsCoordinates,stopLocationsOccurence)
```

```
stopLocations$stopDuration = round(stopLocations$stopDuration,0)
colnames(stopLocations) = c("DistanceFromFrankfurt","shiptype",
                            "meanStopDuration","normalizedOccurence")
#Anzeigen des Tabellenkopfs
kable(head(stopLocations))
```

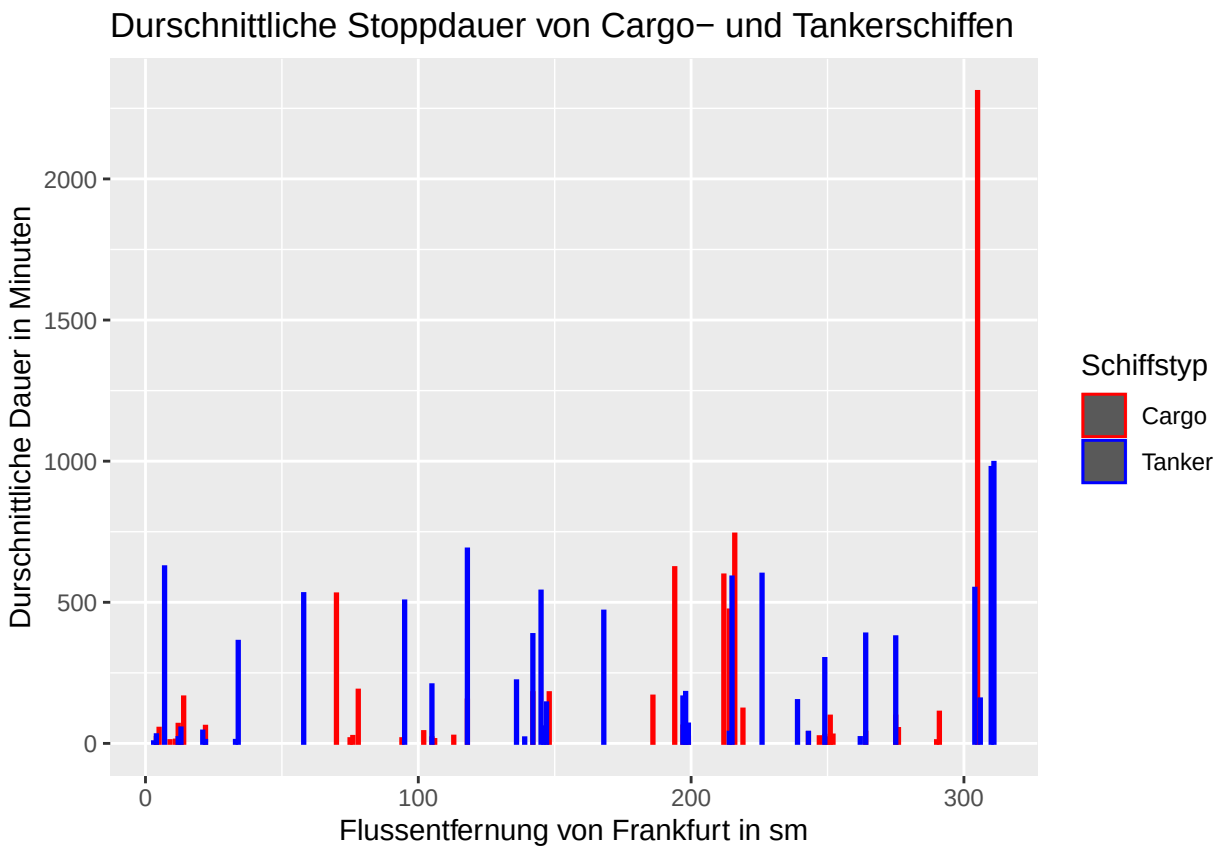
DistanceFromFrankfurt	shiptype	meanStopDuration	normalizedOccurence
102	Cargo	42	2.4242424
105	Cargo	20	0.6060606
105	Tanker	208	0.3030303
106	Cargo	14	0.7575758
11	Cargo	12	0.4545455
113	Cargo	26	1.5151515

```
#Visualisierung der normalisierten Anzahl beobachteter Stopps von Cargo- und Tankerschiffen
ggplot(data = stopLocations,aes())+
  #Graph der Baseline
  geom_bar(data = stopLocations[grepl('Cargo', stopLocations$shiptype, fixed = TRUE),],
           aes(x=DistanceFromFrankfurt,
               y=normalizedOccurence,
               colour = 'Cargo'), stat ="identity")+
  geom_bar(data = stopLocations[grepl('Tanker', stopLocations$shiptype, fixed = TRUE),],
           aes(x=DistanceFromFrankfurt,
               y=normalizedOccurence,
               colour = 'Tanker'), stat ="identity")+
  #Titel hinzufuegen
  ggtitle("normalisierte Haeufigkeit der Stopps von Cargo- und Tankerschiffen")+
  #Farben anpassen
  scale_color_manual(breaks = c("Cargo", "Tanker"), values = c("red", "blue"))+
  #Beschriftung der X-Achse
  xlab("Flussentfernung von Frankfurt in sm")+
  #Beschriftung der Y-Achse
  ylab("normalisierte Haeufigkeit in %")+
  #Beschriftung der Legende
  labs(colour = "Schiffstyp")+
  scale_fill_gradient(low="blue", high="red")
```



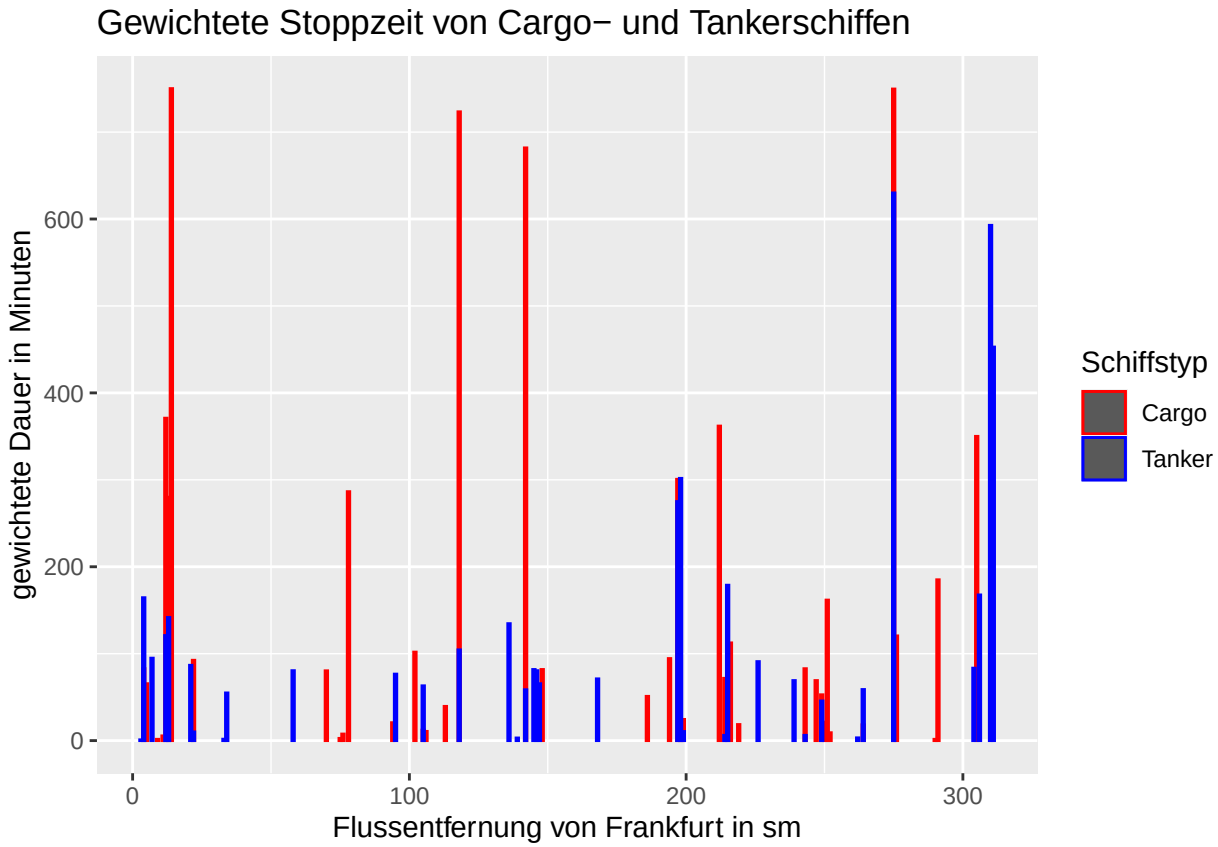

```
#Weiterhin wird nun die Stoppdauer untersucht
#Visualisierung der Dauer beobachteter Stopps von Cargo- und Tankerschiffen
ggplot(data = stopLocations,aes())+
  #Graph der Baseline
  geom_bar(data = stopLocations[grepl('Cargo', stopLocations$shiptype, fixed = TRUE),],
           aes(x=DistanceFromFrankfurt,
               y=meanStopDuration,
               colour = 'Cargo'), stat ="identity")+
  geom_bar(data = stopLocations[grepl('Tanker', stopLocations$shiptype, fixed = TRUE),],
           aes(x=DistanceFromFrankfurt,
               y=meanStopDuration,
               colour = 'Tanker'), stat ="identity")+
  #Titel hinzufuegen
  ggtitle("Durschnittliche Stoppdauer von Cargo- und Tankerschiffen")+
  #Farben anpassen
  scale_color_manual(breaks = c("Cargo", "Tanker"), values = c("red", "blue"))+
  #Beschriftung der X-Achse
  xlab("Flussentfernung von Frankfurt in sm")+
  #Beschriftung der Y-Achse
  ylab("Durschnittliche Dauer in Minuten")+
  #Beschriftung der Legende
  labs(colour = "Schiffstyp")+
  scale_fill_gradient(low="blue", high="red")
```

```
## Don't know how to automatically pick scale for object of type difftime. Defaulting to contin
```




```
#Bugfix: convert difftime to num
stopLocations$meanStopDuration = as.numeric(stopLocations$meanStopDuration)

#Gewichtete Stoppzeit: normalisierte Haeufigkeit * durschnittliche Dauer
stopLocations$weightedStopTime = stopLocations$meanStopDuration *
  stopLocations$normalizedOccurence

#Visualisierung der gewichteten Stoppzeit von Cargo- und Tankerschiffen
ggplot(data = stopLocations,aes())+
  #Graph der Baseline
  geom_bar(data = stopLocations[grepl('Cargo', stopLocations$shiptype, fixed = TRUE),],
           aes(x=DistanceFromFrankfurt,
               y=weightedStopTime,
               colour = 'Cargo'), stat ="identity")+
  geom_bar(data = stopLocations[grepl('Tanker', stopLocations$shiptype, fixed = TRUE),],
           aes(x=DistanceFromFrankfurt,
               y=weightedStopTime,
               colour = 'Tanker'), stat ="identity")+
  #Titel hinzufuegen
  ggtitle("Gewichtete Stoppzeit von Cargo- und Tankerschiffen")+
  #Farben anpassen
  scale_color_manual(breaks = c("Cargo", "Tanker"), values = c("red", "blue"))+
```

```
  #Beschriftung der X-Achse
  xlab("Flussentfernung von Frankfurt in sm")+
  #Beschriftung der Y-Achse
  ylab("gewichtete Dauer in Minuten")+
  #Beschriftung der Legende
  labs(colour = "Schiffstyp")+
  scale_fill_gradient(low="blue", high="red")
```




```
#Feature in tripsRaw hinzufuegen: Wie viel Stoppzeit
#kann noch erwartet werden bis Rotterdamm?
#stopLocations nach DistanceFromFrankfurt sortieren:
stopLocations = stopLocations[order(stopLocations$DistanceFromFrankfurt),]
#Subsets fuer Tanker und Cargoschiffe:
stopLocationsCargo = stopLocations[stopLocations$shiptype=='Cargo',]
stopLocationsTanker = stopLocations[stopLocations$shiptype=='Tanker',]
for (i in 1:nrow(tripsRaw)) {
  if (tripsRaw$isCargo[i]){
    tripsRaw$predictedStopTime[i] = sum(stopLocationsCargo$weightedStopTime[
                                        stopLocationsCargo$DistanceFromFrankfurt >
                                        tripsRaw$distanceAchieved[i]])
  }
  if (tripsRaw$isTanker[i]){
    tripsRaw$predictedStopTime[i] = sum(stopLocationsTanker$weightedStopTime[
                                        stopLocationsTanker$DistanceFromFrankfurt >
```

```
                                        tripsRaw$distanceAchieved[i]])
  }
}
```

Zuletzt werden die Anzahl und kumulierte Dauerder Stopps je Trip in tripsAggregated übernommen.

```
#Anzahl Stopps pro Trip aggregieren und in tripsAggregated einfuegen:
tripsAggregated = merge(tripsAggregated,aggregate(stopDuration~tripName,
                                                  data = stops,NROW))
colnames(tripsAggregated)[colnames(tripsAggregated)=='stopDuration'] = 'stops'

#kummulierte Stoppdauer pro trip in tripsaggregated:
tripsAggregated = merge(tripsAggregated,aggregate(stopDuration~tripName,
                                                  data = stops,sum))
```

4. Modellierung Port-to-Port Modell

Als erster Schritt soll ein statisches Modell entwickelt werden, das fuer jedes Schiff im Hafen von Frankfurt anhand von Informationen, die bereits vor dem Trip verfuegbar sind, eine Prognose fuer die vorraussichtliche Ankunftszeit im Hafen von Rotterdam erstellen kann. Die Datengrundlage fuer diese Modell bildet die Tabelle tripsAggregated. Zielgroesze dieses Modells ist die vorraussichtliche Ankunftszeit (estimated time of arrival, ETA). Verglichen wird diese mit der tatsaechlichen Ankunftszeit (timeEnd).

4.1. Baseline

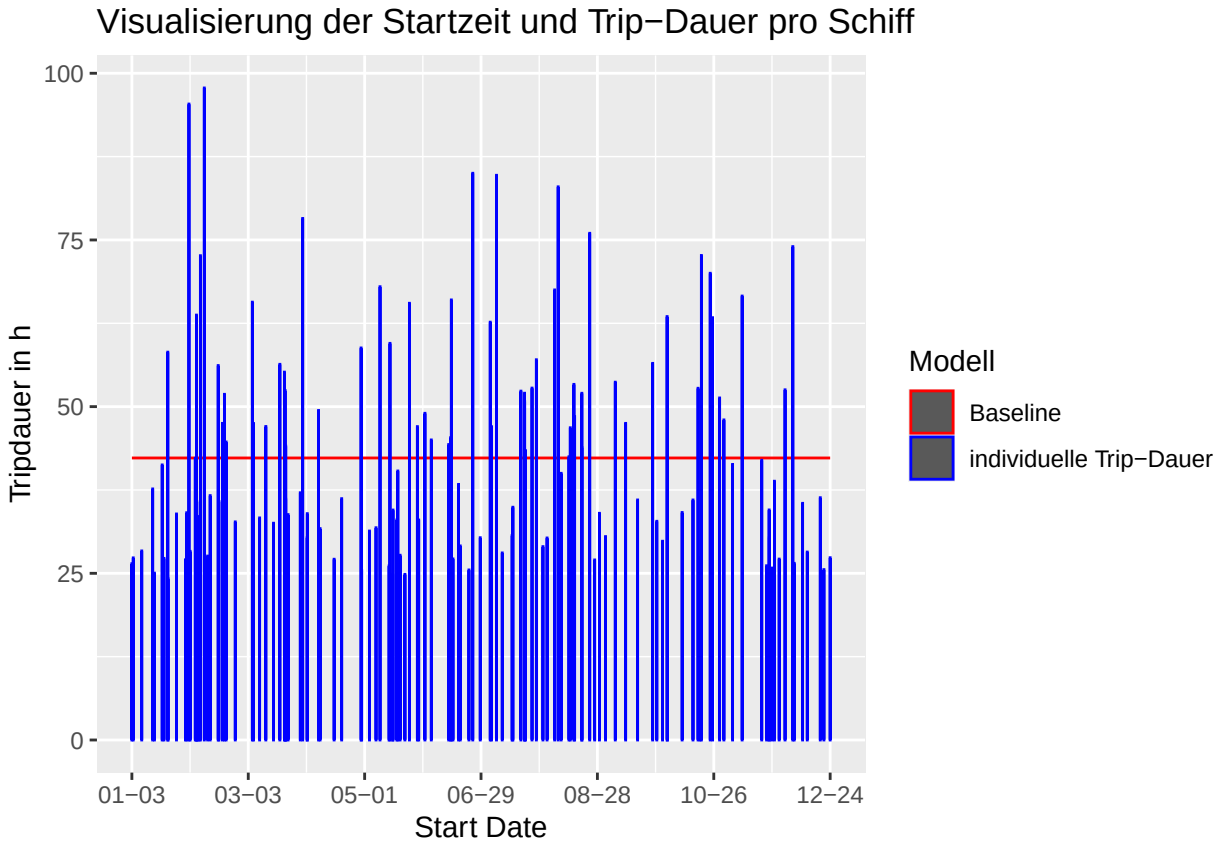
Als Baseline wird die durchschnittliche Trip-Dauer verwendet, da diese im Endeffekt individuell fuer jedes Schiff bzw. jeden Trip vorhergesagt werden soll. Der Mittelwert der Trip-Dauer liefert ein naives Vergleichsmodell, da die Werte zum Teil stark von der durschnittlichen Trip-Dauer abweichen koennen. Die Idee timestampETA einzubinden wurde wieder verworfen, da die manuelle Eingabe der Daten zu haeufig vernachlaessigt wird (Daten ETA liegen zum Teil in der Vergangenheit) und daher nicht als Vergleichsmodell dienen kann.

4.1.1 Baseline erstellen und visualisieren

```
#Erstellen der Baseline
tripsAggregated$baselinetripsMean = tripsAggregated$tripsMeaninh

#Durchschnittliche Distanz aller Tripps berechnen
tripsAggregated$distanceMeaninkm = mean(tripsAggregated$Distanceinkm)
#Umwandlung der Entfernung in Seemeilen
tripsAggregated$distanceMeaninnmi = tripsAggregated$distanceMeaninkm * 0.54

#euklidische Distanz Rotterdam - Frankfurt (distanceMean)/ meanTripDauer -
#--> man erhaelt die Durchschnittsgeschwindigkeit

tripsAggregated$speedMeaninkmperh = tripsAggregated$distanceMeaninkm /
                                    tripsAggregated$tripsMeaninh
#Umwandlung der Geschwindigkeit in Knoten
tripsAggregated$speedMeaninkn = tripsAggregated$speedMeaninkmperh * 0.54

#Statisches Modell (alles Tripsaggregated):
# ETA = TripsMeaninh + Startzeit

tripsAggregated$timeStart = as.POSIXct(tripsAggregated$timeStart,
                                       tz = "GMT", "%Y-%m-%d %H:%M:%OS")
tripsAggregated$baselineETA = tripsAggregated$tripsMeaninh * 60 * 60 +
                              tripsAggregated$timeStart
tripsAggregated$baselineETA_num = as.numeric(as.POSIXct(tripsAggregated$baselineETA))
```

```r
#Visualisierung von Startzeit und Tripdauer aus TripsAggregated
ggplot(data = tripsAggregated, aes(xmin = min(tripsAggregated$timeStart_num),
                                   xmax = max(tripsAggregated$timeStart_num)))+
    #Graph der Baseline
  geom_line(data = tripsAggregated, aes(
    x=timeStart_num,
    y=baselinetripsMean,
    colour = 'Baseline'))+
  #Graph
   geom_bar(data=tripsAggregated, aes(
    x=timeStart_num,
    y=tripTime,
    colour = 'individuelle Trip-Dauer'), stat ="identity")+
  #Titel hinzufuegen
  ggtitle("Visualisierung der Startzeit und Trip-Dauer pro Schiff")+
  #Farben anpassen
  scale_color_manual(breaks = c("Baseline", "individuelle Trip-Dauer"),
                     values = c("red", "blue"))+
  #Beschriftung der X-Achse
  xlab("Start Date")+
  scale_x_continuous(breaks = seq(min(tripsAggregated$timeStart_num),
                                  max(tripsAggregated$timeStart_num),
                                  (30654280/6)),
                     labels = tripTimes$uhrzeitRegularonly) +
    #Beschriftung der Y-Achse
  ylab("Tripdauer in h")+
  #Beschriftung der Legende
  labs(colour = "Modell")
```

```
## Warning: Use of `tripsAggregated$timeStart_num` is discouraged. Use
## `timeStart_num` instead.

## Warning: Use of `tripsAggregated$timeStart_num` is discouraged. Use
## `timeStart_num` instead.

## Warning: Use of `tripsAggregated$timeStart_num` is discouraged. Use
## `timeStart_num` instead.

## Warning: Use of `tripsAggregated$timeStart_num` is discouraged. Use
## `timeStart_num` instead.
```



4.1.2. Baseline bewerten

Model	Rsquared	MAE	MAPE	pValue
Baseline	0	47297.21	0.0000303	

Je geringer die Fehlerkennzahlen MAE und MAPE sind, desto besser ist das Modell. Die Baseline zeigt grosze Unterschiede zwischen dem MAE und dem MAPE. Das ist damit zu erklaeren, dass die Zielgroesze ein Datum ist. Dieses wird in R als Anzahl der Sekunden seit dem 1.1.1970 angegeben. Somit ist die hohe Abweichung durch die Skalierung der Zielgroesze zu erklaeren. Die prozentuale Abweichung ist jedoch gering und laesst darauf schlieszen, dass die Baseline die Zielgroesze recht gut beschreibt.

4.2. Vorbereitung

Fuer die lineare Regression werden die unabhaengigen Variablen (Features) mit der Wrapper Methode ausgewaehlt. Diese Methode erstellt nacheinander mehrere Teilmodelle mit unterschiedlichen Features. Die Forward Selection Methode beginnt mit einem leeren Modell und fuegt nacheinander einzelne Features hinzu. Die verschiedenen Modelle werden anhand von Fehlerkennzahlen miteinander verglichen und das beste Modell wird weitergefuehrt, bis keine Verbesserung mehr erfolgt.

4.2.1. Aufteilung Training-/Test-Daten

Um das zu erstellende Modell im Nachgang besser zu bewerten wird die Datenquelle zunaechst in ein Trainings- und ein Testdatenset aufgeteilt. Das Modell wird mit den Trainingsdaten erstellt und im Anschluss mit Hilfe der Testdaten auf Overfitting untersucht.

```
#Ein zufaelliger Zustand wird hergestellt
set.seed(4141)
#Eine Zufallsauswahl erstellen: Aus der Liste von Zahlen 1 bis
#Laenge von tripsAggregated werden 80% der Daten zufaellig ausgewaehlt
zufall = sample(1:nrow(tripsAggregated), nrow(tripsAggregated)*0.8)
training_data_aggregated = tripsAggregated[zufall,]
test_data_aggregated = tripsAggregated[-zufall,]
```

4.2.2. Korrelation berechnen

Fuer die Auswahl der Features, die in den Modellierungsprozess aufgenommen werden sollen, ist es wichtig ihre mathematische Relevanz in Bezug auf die Zielgroesze zu kennen.

	Startzeit	Geschwindigkeit	Niedrigwasser	Ferien	isCargo	isTanker	isHazardous	Ankunftszeit
Startzeit	1.00000	0.19678	0.18851	0.26207	0.05331	-0.05331	-0.09074	0.99998
Geschwindigkeit	0.19678	1.00000	0.03117	0.14772	0.05953	-0.05953	0.06785	0.19736
Niedrigwasser	0.18851	0.03117	1.00000	0.15435	-0.05838	0.05838	0.04505	0.18813
Ferien	0.26207	0.14772	0.15435	1.00000	0.01836	-0.01836	0.04965	0.26284
isCargo	0.05331	0.05953	-0.05838	0.01836	1.00000	-1.00000	-0.22125	0.05593
isTanker	-0.05331	-0.05953	0.05838	-0.01836	-1.00000	1.00000	0.22125	-0.05593
isHazardous	-0.09074	0.06785	0.04505	0.04965	-0.22125	0.22125	1.00000	-0.09057
Ankunftszeit	0.99998	0.19736	0.18813	0.26284	0.05593	-0.05593	-0.09057	1.00000

Aus der obigen Tabelle kann man erkennen, dass ausschlieszlich die Startzeit eine starke Korrelation zur Endzeit aufweiszt. Alle anderen Korrelationswerte sind eher gering. Aufgrund der Verteilung der Korrelation haben wir uns entschieden im weiteren Prozess alle Features, die eine Korrelation mit einem Betrag ueber 0.15 aufweisen, zu betrachten. Damit ist timeStart der groeszte Einflussfaktor. CurrentSpeedOverGround, low_water und Vacation weisen ebenfalls eine geringe Korrelation auf. Zwischen diesen Features scheint keine merkenswerte Multikollinearitaet vorzuliegen.

4.3. Lineare Regression

Ausgehend von den Vorbetrachtungen wird nun fuer jedes relevante Feature ein lineares Regressionsmodell erstellt. Diese werden anhand des Bestimtheitsmaszes R^2, des MAE, MAPE und des Signifikanzniveaus (pValue) bewertet und das beste Modell wird ausgewaehlt.

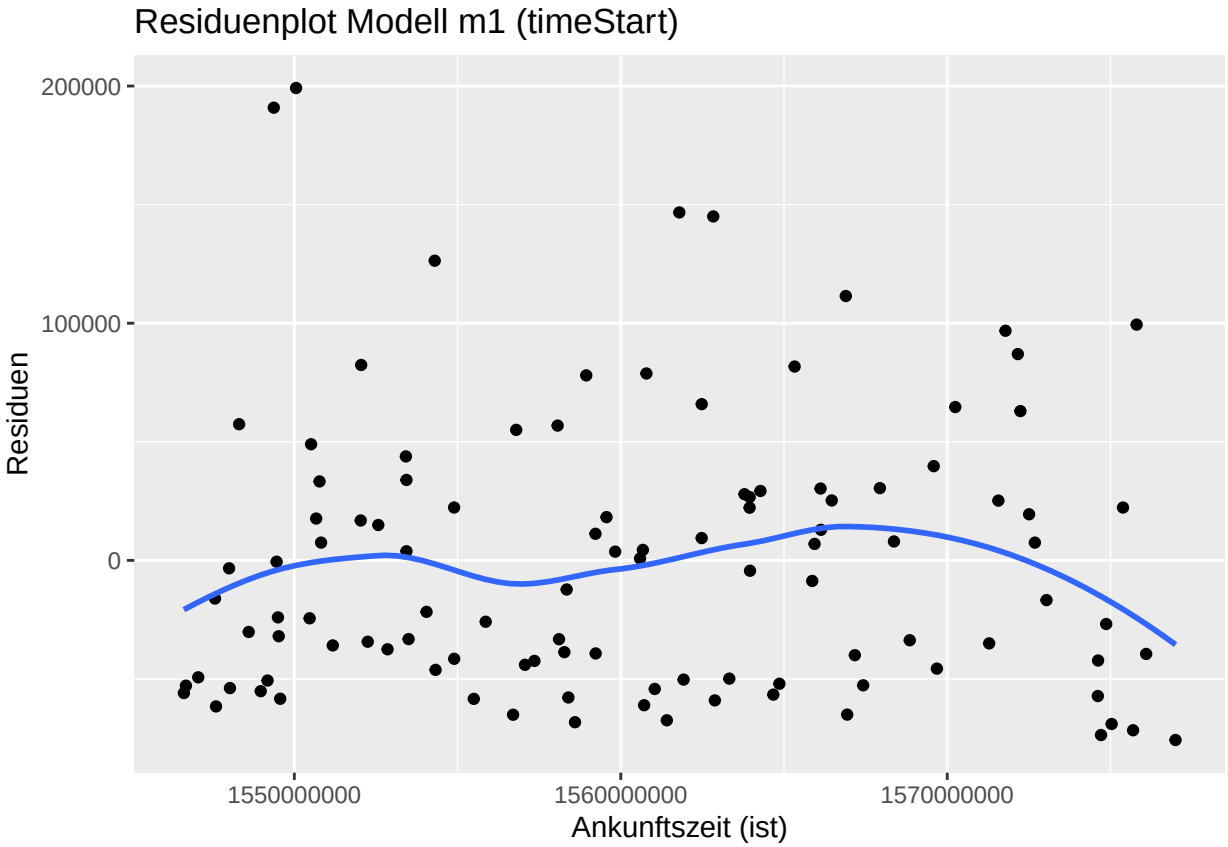
```
# Uni-Variate Modelle werden erzeugt
aggregated_m1 = lm(timeEnd_num ~ timeStart_num, data=training_data_aggregated)
aggregated_m2 = lm(timeEnd_num ~ currentSpeedOverGround, data=training_data_aggregated)
aggregated_m3 = lm(timeEnd_num ~ low_water, data=training_data_aggregated)
aggregated_m4 = lm(timeEnd_num ~ Vacation, data=training_data_aggregated)

#Bewertung Uni-variater Modelle ueber Fehlerkennzahlen
#m1 (timeStart)
evaluation_aggregated = rbind(evaluation_aggregated, data.frame(
                                        Model = c("m1_timeStart"),
                                        Rsquared = numeric(1),
                                        MAE = numeric(1),
                                        MAPE = numeric(1),
                                        pValue = character(1)))
evaluation_aggregated[evaluation_aggregated$Model == "m1_timeStart",]$Rsquared =
  summary(aggregated_m1)$r.squared
evaluation_aggregated[evaluation_aggregated$Model == "m1_timeStart",]$MAE =
  mean(abs(aggregated_m1$residuals))
evaluation_aggregated[evaluation_aggregated$Model == "m1_timeStart",]$MAPE =
  mape(aggregated_m1$model$timeEnd_num, aggregated_m1$fitted.values)
evaluation_aggregated[evaluation_aggregated$Model == "m1_timeStart",]$pValue =
  as.character(summary(aggregated_m1)$coefficients[2,3])
#m2 (currentSpeedOverGround)
evaluation_aggregated = rbind(evaluation_aggregated, data.frame(
                                        Model = c("m2_currentSpeedOverGround"),
                                        Rsquared = numeric(1),
                                        MAE = numeric(1),
                                        MAPE = numeric(1),
                                        pValue = character(1)))
evaluation_aggregated[evaluation_aggregated$Model == "m2_currentSpeedOverGround",]$Rsquared =
  summary(aggregated_m2)$r.squared
evaluation_aggregated[evaluation_aggregated$Model == "m2_currentSpeedOverGround",]$MAE =
  mean(abs(aggregated_m2$residuals))
evaluation_aggregated[evaluation_aggregated$Model == "m2_currentSpeedOverGround",]$MAPE =
  mape(aggregated_m2$model$timeEnd_num, aggregated_m2$fitted.values)
evaluation_aggregated[evaluation_aggregated$Model == "m2_currentSpeedOverGround",]$pValue =
  as.character(summary(aggregated_m2)$coefficients[2,3])
#m3 (low_water)
evaluation_aggregated = rbind(evaluation_aggregated, data.frame(
                                        Model = c("m3_low_water"),
                                        Rsquared = numeric(1),
                                        MAE = numeric(1),
```

```
                                          MAPE = numeric(1),
                                          pValue = character(1)))
evaluation_aggregated[evaluation_aggregated$Model == "m3_low_water",]$Rsquared =
  summary(aggregated_m3)$r.squared
evaluation_aggregated[evaluation_aggregated$Model == "m3_low_water",]$MAE =
  mean(abs(aggregated_m3$residuals))
evaluation_aggregated[evaluation_aggregated$Model == "m3_low_water",]$MAPE =
  mape(aggregated_m3$model$timeEnd_num, aggregated_m3$fitted.values)
evaluation_aggregated[evaluation_aggregated$Model == "m3_low_water",]$pValue =
  as.character(summary(aggregated_m3)$coefficients[2,3])
#m4 (Vacation)
evaluation_aggregated = rbind(evaluation_aggregated, data.frame(
                                          Model = c("m4_Vacation"),
                                          Rsquared = numeric(1),
                                          MAE = numeric(1),
                                          MAPE = numeric(1),
                                          pValue = character(1)))
evaluation_aggregated[evaluation_aggregated$Model == "m4_Vacation",]$Rsquared =
  summary(aggregated_m4)$r.squared
evaluation_aggregated[evaluation_aggregated$Model == "m4_Vacation",]$MAE =
  mean(abs(aggregated_m4$residuals))
evaluation_aggregated[evaluation_aggregated$Model == "m4_Vacation",]$MAPE =
  mape(aggregated_m4$model$timeEnd_num, aggregated_m4$fitted.values)
evaluation_aggregated[evaluation_aggregated$Model == "m4_Vacation",]$pValue =
  as.character(summary(aggregated_m4)$coefficients[2,3])

#Fehler anzeigen
kable(evaluation_aggregated)
```

Model	Rsquared	MAE	MAPE	pValue
Baseline	0.0000000	47297.21	0.0000303	
m1_timeStart	0.9999548	46508.28	0.0000298	1532.09433708915
m2_currentSpeedOverGround	0.0474870	7078112.55	0.0045340	2.29881883265196
m3_low_water	0.0567939	7133482.53	0.0045695	2.52638933400756
m4_Vacation	0.0753245	6490520.50	0.0041558	2.93850521228364

```
#Residuenplot fuer Modell m1
ggplot(data = NULL, aes(x = aggregated_m1$model$timeEnd, y = aggregated_m1$residuals)) +
  geom_point() +
  geom_smooth(se = FALSE, method = loess) +
  ggtitle("Residuenplot Modell m1 (timeStart)")+
  xlab("Ankunftszeit (ist)")+
  ylab("Residuen")
```

```
## `geom_smooth()` using formula 'y ~ x'
```

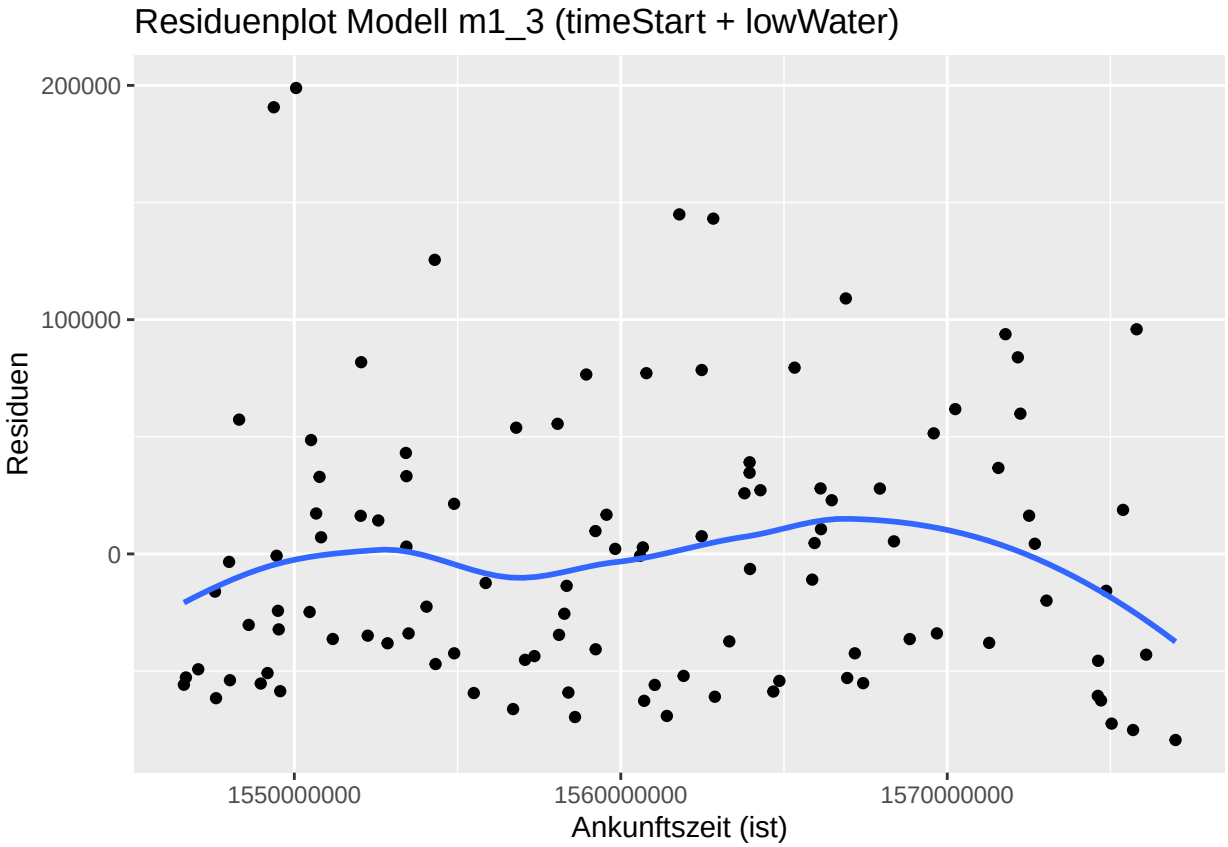


Modell m1, das die Startzeit abetrachtet, wird anhand der regressionsspezifischen Kennzahlen ausgewaehlt. Dies entspricht gleichzeitig dem Feature mit der hoechsten Korrelation. Es besteht keine Multikollinearitaet zwischen der Startzeit und den verbleibenden Fehlerkennzahlen. Auf dieser Basis werden im Folgenden alle Regressionsmodelle mit zwei Features evaluiert.

Hinweis Bei allen weiteren Regressionsiterationen werden nur noch die Ergebnisse ausgegeben.

Model	Rsquared	MAE	MAPE	pValue
Baseline	0.0000000	47297.21	0.0000303	
m1_timeStart	0.9999548	46508.28	0.0000298	1532.09433708915
m2_currentSpeedOverGround	0.0474870	7078112.55	0.0045340	2.29881883265196
m3_low_water	0.0567939	7133482.53	0.0045695	2.52638933400756
m4_Vacation	0.0753245	6490520.50	0.0041558	2.93850521228364
m1_2_timeStart_currentSpeedOverGround	0.9999552	45926.86	0.0000294	1494.36278316609
m1_3_timeStart_low_water	0.9999551	46257.27	0.0000296	1485.19377689069
m1_4_startTime_Vacation	0.9999553	45986.76	0.0000295	1474.10393624624

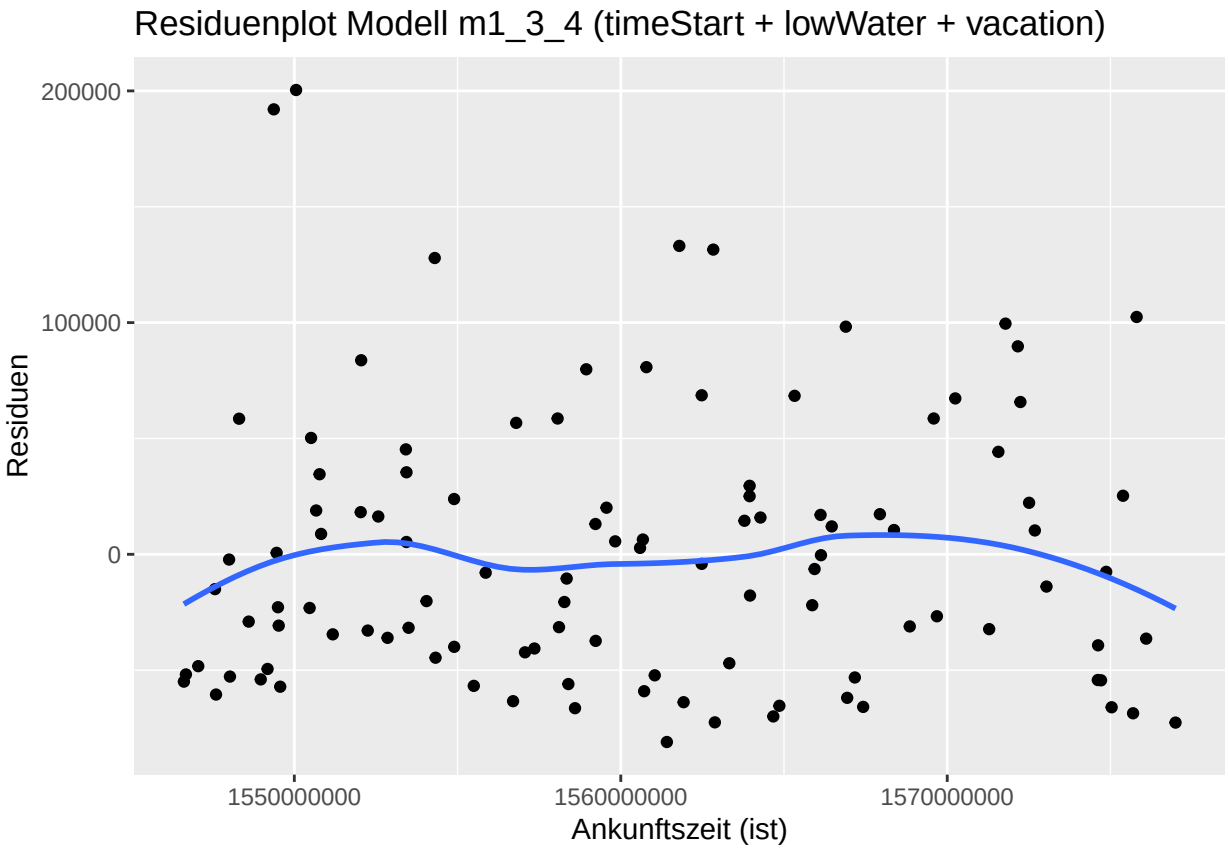
```
## `geom_smooth()` using formula 'y ~ x'
```



Modell m1_3, das neben der Startzeit auch Informationen zu Niedrigwasser betrachtet, wird anhand der regressionsspezifischen Kennzahlen ausgewaehlt. Es beseteht keine Multikollinearitaet zwischen Niedrigwasser und den beiden verbleibenden Features. Auf dieser Basis werden im Folgenden alle Regressionsmodelle mit drei Features evaluiert.

Model	Rsquared	MAE	MAPE	pValue
Baseline	0.0000000	47297.21	0.0000303	
m1_timeStart	0.9999548	46508.28	0.0000298	1532.09433708915
m2_currentSpeedOverGround	0.0474870	7078112.55	0.0045340	2.29881883265196
m3_low_water	0.0567939	7133482.53	0.0045695	2.52638933400756
m4_Vacation	0.0753245	6490520.50	0.0041558	2.93850521228364
m1_2_timeStart_currentSpeedOverGround	0.9999552	45926.86	0.0000294	1494.36278316609
m1_3_timeStart_low_water	0.9999551	46257.27	0.0000296	1485.19377689069
m1_4_startTime_Vacation	0.9999553	45986.76	0.0000295	1474.10393624624
m1_3_2_currentSpeedOverGround	0.9999554	45731.32	0.0000293	1443.1257692037
m1_3_4_Vacation	0.9999556	45761.19	0.0000293	1440.08092618814

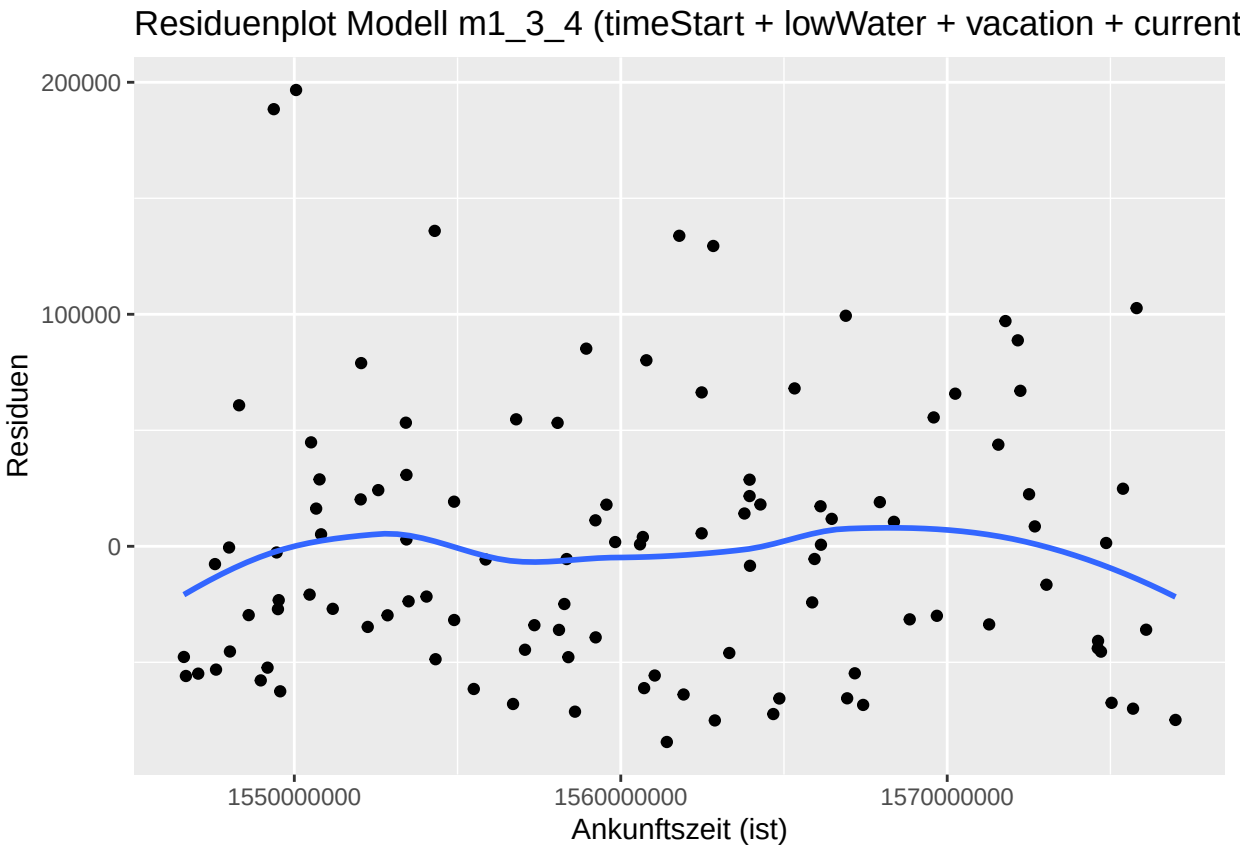
```
## `geom_smooth()` using formula 'y ~ x'
```



Modell m1_3_4, das neben Startzeit und Niedrigwasser auch Feriendaten betrachtet, wird anhand der regressionsspezifischen Kennzahlen ausgewaehlt. Es besteht keine Multikollinearitaet zwischen Ferien und der Anfangsgeschwindigkeit. Auf dieser Basis werden im Folgenden alle Regressionsmodelle mit vier Features evaluiert.

Model	Rsquared	MAE	MAPE	pValue
Baseline	0.0000000	47297.21	0.0000303	
m1_timeStart	0.9999548	46508.28	0.0000298	1532.09433708915
m2_currentSpeedOverGround	0.0474870	7078112.55	0.0045340	2.29881883265196
m3_low_water	0.0567939	7133482.53	0.0045695	2.52638933400756
m4_Vacation	0.0753245	6490520.50	0.0041558	2.93850521228364
m1_2_timeStart_currentSpeedOverGround	0.9999552	45926.86	0.0000294	1494.36278316609
m1_3_timeStart_low_water	0.9999551	46257.27	0.0000296	1485.19377689069
m1_4_startTime_Vacation	0.9999553	45986.76	0.0000295	1474.10393624624
m1_3_2_currentSpeedOverGround	0.9999554	45731.32	0.0000293	1443.1257692037
m1_3_4_Vacation	0.9999556	45761.19	0.0000293	1440.08092618814
m1_3_4_2_currentSpeedOverGround	0.9999559	45144.94	0.0000289	1407.40580222984

```
## `geom_smooth()` using formula 'y ~ x'
```



Diese Modell hat keine statistisch signifikante Verbesserung der Regressionskennzahlen gebracht. Deswegen wird das vorherige Modell angenommen und im folgenden gegen die Testdaten validiert.

Anmerkung zum statischen Modell: Bei der Aufbereitung des Codes fuer die Abgabe wurde ein Fehler im Residuenplot entdeckt. Dieser wurde behoben. Die Residuen haben nun nicht mehr die Ausgleichsgerade auf der Nulllinie, sondern streuen nah um den Nullpunkt herum. Die Aussage der beiden Grafiken bleibt jedoch aehnlich: die regressionsspezifischen Kennzahlen weisen auf ein nahezu perfektes Modell hin.

4.4. Vergleich des Modells mit Baseline und Testdaten

Um das Modell auf Overfitting zu testen wird eine Vorhersage mit den Testdaten erzeugt.

Model	Rsquared	MAE	MAPE	pValue
Baseline	0.0000000	47297.21	0.0000303	
m1_timeStart	0.9999548	46508.28	0.0000298	1532.09433708915
m2_currentSpeedOverGround	0.0474870	7078112.55	0.0045340	2.29881883265196
m3_low_water	0.0567939	7133482.53	0.0045695	2.52638933400756
m4_Vacation	0.0753245	6490520.50	0.0041558	2.93850521228364
m1_2_timeStart_currentSpeedOverGround	0.9999552	45926.86	0.0000294	1494.36278316609
m1_3_timeStart_low_water	0.9999551	46257.27	0.0000296	1485.19377689069
m1_4_startTime_Vacation	0.9999553	45986.76	0.0000295	1474.10393624624
m1_3_2_currentSpeedOverGround	0.9999554	45731.32	0.0000293	1443.1257692037
m1_3_4_Vacation	0.9999556	45761.19	0.0000293	1440.08092618814

Model	Rsquared	MAE	MAPE	pValue
m1_3_4_2_currentSpeedOverGround	0.9999559	45144.94	0.0000289	1407.40580222984
m1_3_4_test	NA	50241.00	0.0000322	NA

```
## Die Fehlerkennzahlen der Testdaten weichen leicht vom Trainings-Datensatz ab.
##  Der Unterschied liegt bei 4479.809 (MAE) und 0.00000283329 (MAPE).
```

Die Fehlerkennzahlen des Modells m1_3_4 sind im Vergleich zur Baseline nur leicht verbessert. Das liegt jedoch vor allem daran, dass auch die Baseline nahezu perfekte Werte aufweiszt.

Im Bezug auf die Groeszenordnung der regressionsspezifischen Kennzahlen ist die identifizierte Abweichung zwischen den Trainings und Testdaten signifikant. Das deutet auf ein Overfitting hin.

4.5. Auswertung und Interpretation des Modells

Das statische Modell wurde mit der Zielgroesze Ankunftszeit (timeEnd_num) erzeugt. Die regressionsspezifischen Kennzahlen, insbesondere das Bestimmtheitsmasz R^2 und der MAPE, weisen nahezu perfekte Werte auf. Dies entspricht jedoch nicht der tatsaechlichen Beobachtung. Wir haben herausgefunden, dass diese Werte durch die Interpretation der Daten in der genutzten Programmiersprache entstehen. Zeiten werden als Anzahl von Sekunden seit dem 01.01.1970 gespeichert. Eine Abweichung von wenigen Stunden bei der Vorhersage der Ankunftszeit faellt somit im Vergleich kaum ins Gewicht. Die Zielgroesze ist somit nicht dafuer geeignet die exakte Ankunftszeit zu ermitteln. Sie dient lediglich als Orientierung fuer den Tag der Ankunft (dieser kann dafuer sehr zuverlaessig vorhergesagt werden). Die Vorhersage der taggenauen ETA kann jedoch laut Projektbeschreibung auch mit den bisherigen Informationen erfolgen.

Zusaetzlich konnte bei unserem Modell auch noch ein Overfitting identifiziert werden. Das statische Modell kann demnach nicht fuer den Einsatz in der Praxis empfohlen werden.

5. Modellierung des *-to-Rotterdam Modells

Das dynamische Modell soll von einem beliebigen Punkt der Strecke zwischen Frankfurt und Rotterdam aus die verbleibende Tripdauer vorhersagen. In Kombination mit dem aktuellen Zeitstempel kann so die vorraussichtliche Ankunftszeit vorhergesagt werden. Die Zielgroesze Ankunftszeit wurde wegen der Skalierung des Datums verworfen, stattdessen wird die Tripdauer (remainingTripTime) betrachtet. Die Datenquelle ist die Tabelle *tripsRaw*. Das sonstige Vorgehen ist analog zum statischen Modell.

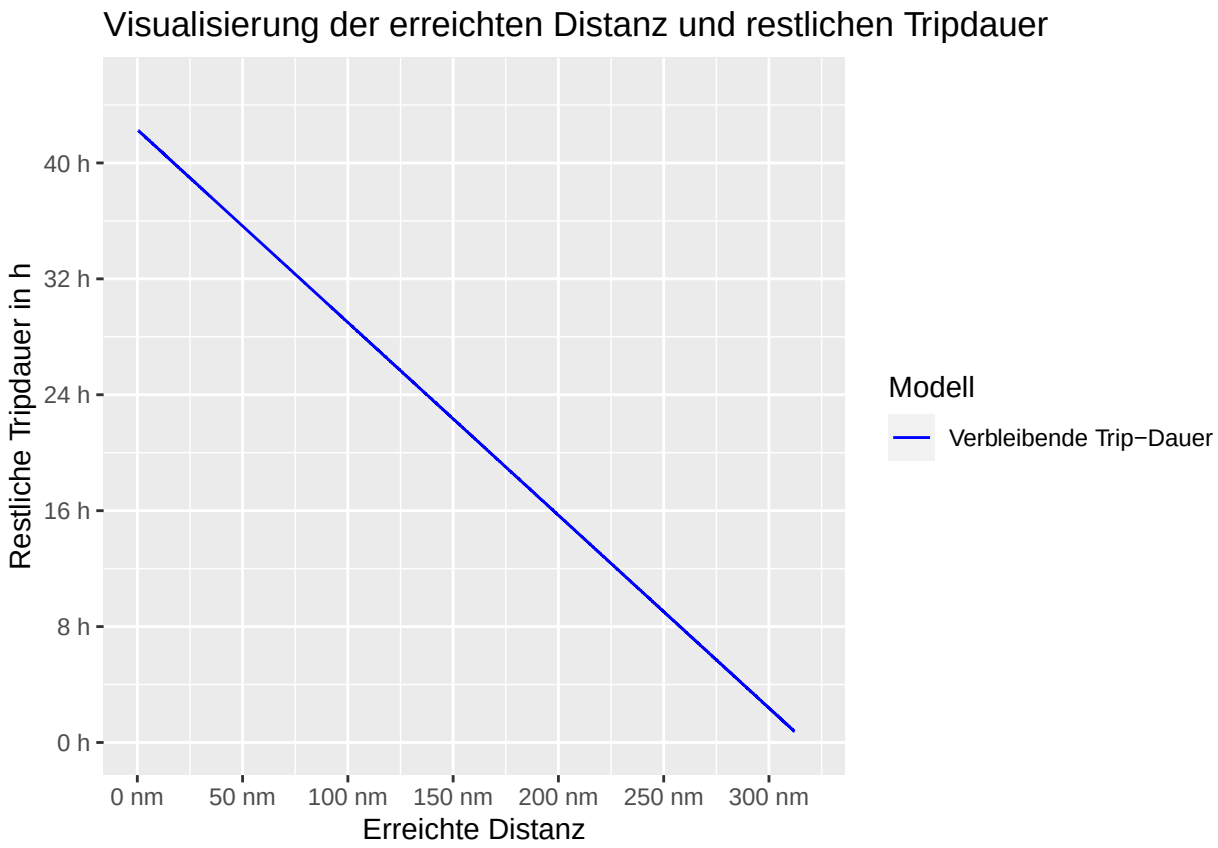
5.1. Baseline

Als Baseline wird die durchschnittliche Trip-Dauer verwendet, da diese im Endeffekt individuell fuer jedes Schiff bzw. jeden Trip vorhergesagt werden soll. Der Mittelwert der Trip-Dauer liefert ein naives Vergleichsmodell, da die Werte zum Teil stark von der durschnittlichen Trip-Dauer abweichen koennen.

5.1.1. Baseline erstellen und visualisieren

```
#Endposition ist immer gleich: 51.889; 4.619
#aktuelle Zeit, ist TimeStampPosition
# ETA = TripsMeaninh * Anteil uebriger Strecke(adjustedtripsraw$distanceoutstanding)
#       / Laenge Gesamtstrecke(317.8761)
tripsRaw$tripsMeaninh = 42.29928
tripsRaw$timeOutstandinginh = tripsRaw$tripsMeaninh *
                              (tripsRaw$distanceOutstanding / 317.8761)
tripsRaw$timestampPosition = as.POSIXct(tripsRaw$timestampPosition,
                                        tz = "GMT", "%Y-%m-%d %H:%M:%OS")
tripsRaw$baselineETA = tripsRaw$timeOutstandinginh

  #Visualisierung von Distance Achieved und verbleibender Tripzeit aus TripsRaw
  ggplot(data = tripsRaw, aes(xmin = 0, xmax = 320, ymin = 0, ymax = 45))+

  #Graph
   geom_line(data = tripsRaw, aes(
    x=distanceAchieved,
    y=timeOutstandinginh,
    colour = 'Verbleibende Trip-Dauer'), stat ="identity")+
  #Titel hinzufuegen
  ggtitle("Visualisierung der erreichten Distanz und restlichen Tripdauer")+
  #Farben anpassen
  scale_color_manual(breaks = c("Baseline", "Verbleibende Trip-Dauer"),
                     values = c("red", "blue"))+
  #Beschriftung der X-Achse
  xlab("Erreichte Distanz")+
  scale_x_continuous(breaks = seq(0, 300, (300/6)),
```

```
                    labels = c("0 nm", "50 nm", "100 nm", "150 nm",
                               "200 nm", "250 nm", "300 nm"))+
  #Beschriftung der Y-Achse
ylab("Restliche Tripdauer in h")+
scale_y_continuous(breaks = seq(0, max(tripsRaw$timeOutstandinginh), (40/5)),
                    labels = c("0 h", "8 h", "16 h", "24 h", "32 h", "40 h"))+
#Beschriftung der Legende
labs(colour = "Modell")
```



Die Visualisierung zeigt die Plausibilitaet der Daten. Je kuerzer die verbleibende Strecke, destso kuerzer ist auch die verbleibende Zeit.

5.1.2. Baseline bewerten

Model	Rsquared	MAE	MAPE	pValue
Baseline	0	78901.84	0.9996864	

Die hier gezeigten Fehlerkennzahlen sind deutlich realistischer als die unter 4.1. gezeigten Zahlen.

5.2. Vorbereitung

5.2.1 Aufteilung Training-/Test-Daten

```
#Ein zufaelliger Zustand wird hergestellt
set.seed(4141)
#Eine Zufallsauswahl erstellen: Aus der Liste von Zahlen 1 bis
#Laenge von tripsRaw werden 80% gewaehlt
zufall = sample(1:nrow(tripsRaw), nrow(tripsRaw) * 0.8)

#Die Eintraege in der Zufallsauswahl werden in das TrainingsSet aufgenommen
training_data_raw = tripsRaw[zufall, ]
test_data_raw = tripsRaw[-zufall,]
```

5.2.2. Korrelation berechnen

	Verbleibende Distanz	Schleusen	Tiefgang	Wasserstand	Verhaeltnis Wasserstand Tiefgang	Niedrigwasser
Verbleibende Distanz	1.0000000	-0.4398888	-0.0647999	-0.4518215	-0.0764999	0.1024745
Schleusen	-0.4398888	1.0000000	-0.0166739	0.2447558	0.0703664	-0.2394893
Tiefgang	-0.0647999	-0.0166739	1.0000000	0.1014320	-0.7517555	-0.0396583
Wasserstand	-0.4518215	0.2447558	0.1014320	1.0000000	0.2031451	-0.0963546
Verhaeltnis Wasserstand Tiefgang	-0.0764999	0.0703664	-0.7517555	0.2031451	1.0000000	-0.0158500
Niedrigwasser	0.1024745	-0.2394893	-0.0396583	-0.0963546	-0.0158500	1.0000000

	isCargo	isHazardous	aktuelle Geschwindigkeit	Ferien	Gewichtete Stoppzeit	Aktuelle Zeit	Verbleibende Dauer
isCargo	1.0000000	-0.2621142	-0.2755800	-0.0223386	0.1902208	0.0566569	0.2453336
isHazardous	-0.2621142	1.0000000	-0.0071393	0.0232181	-0.0477815	-0.1189317	-0.0198043
aktuelle Geschwindigkeit	-0.2755800	-0.0071393	1.0000000	-0.0587281	-0.2664997	-0.0246612	-0.3678139
Ferien	-0.0223386	0.0232181	-0.0587281	1.0000000	-0.0593658	0.2345523	0.1439912

	isCargo	isHazardous	aktuelle Geschwindigkeit	Ferien	Gewichtete Stoppzeit	Aktuelle Zeit	Verbleibende Dauer
Gewichtete Stoppzeit	0.1902208	-0.0477815	-0.2664997	-0.0593658	1.0000000	0.0102360	0.5966584
Aktuelle Zeit	0.0566569	-0.1189317	-0.0246612	0.2345523	0.0102360	1.0000000	0.0586664
Verbleibende Dauer	0.2453336	-0.0198043	-0.3678139	0.1439912	0.5966584	0.0586664	1.0000000

```
## Gefiltert nach der Korrelation zur verbleibenden Trip Dauer ergeben sich
## die folgenden Werte: (X steht hierbei fuer die Korrelation)
```

	x
Verbleibende Distanz	0.5685239
Schleusen	-0.2986191
Tiefgang	0.2058432
Wasserstand	-0.2787517
Verhaeltnis Wasserstand Tiefgang	-0.1868185
Niedrigwasser	0.1031542
isCargo	0.2453336
isHazardous	-0.0198043
aktuelle Geschwindigkeit	-0.3678139
Ferien	0.1439912
Gewichtete Stoppzeit	0.5966584
Aktuelle Zeit	0.0586664
Verbleibende Dauer	1.0000000

```
## Nachfolgend wird eine Tabelle der 10 am staerksten mit der restlichen Tripdauer
## korrelierenden Variablen ausgegeben. Der Wert x gibt die Korrelationsstaerke an:
```

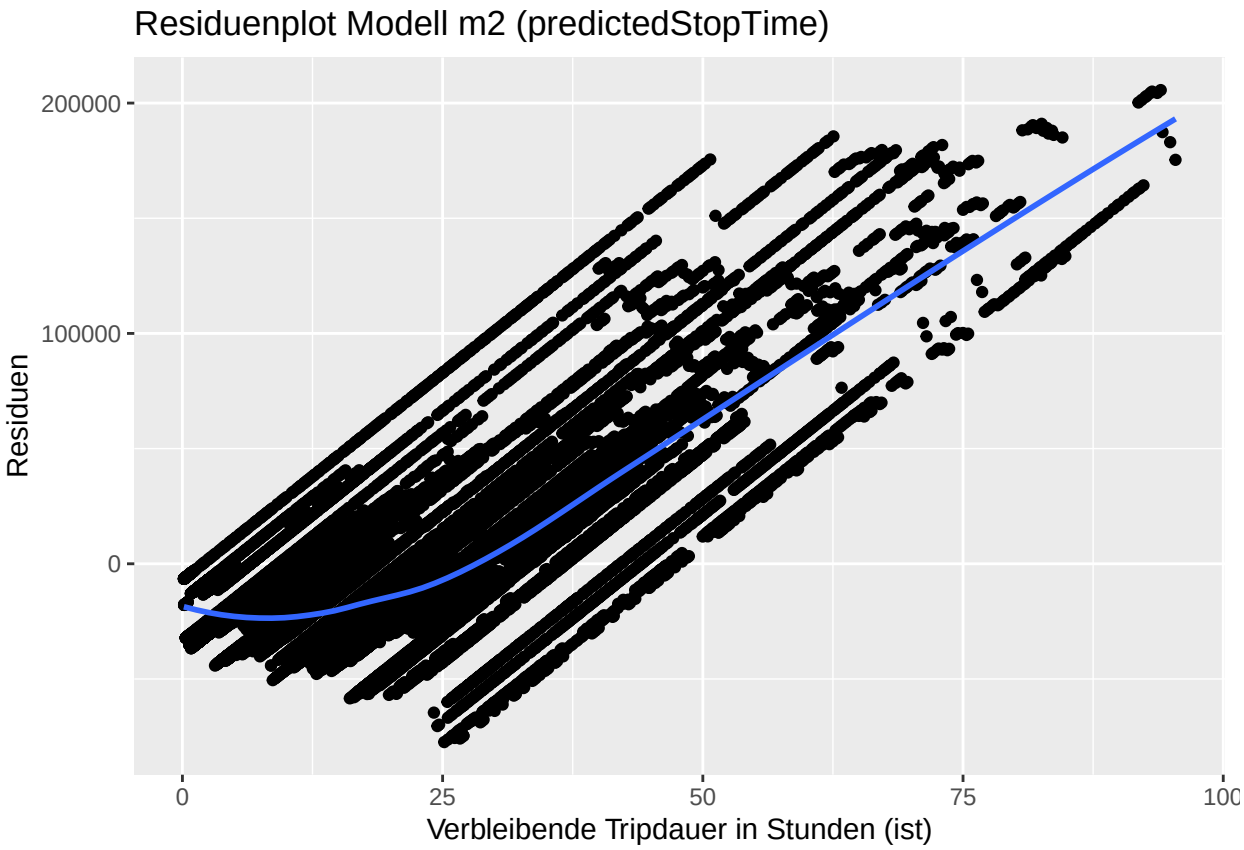
	x
Verbleibende Dauer	1.0000000
Gewichtete Stoppzeit	0.5966584
Verbleibende Distanz	0.5685239
aktuelle Geschwindigkeit	-0.3678139
Schleusen	-0.2986191
Wasserstand	-0.2787517
isCargo	0.2453336
Tiefgang	0.2058432
Verhaeltnis Wasserstand Tiefgang	-0.1868185
Ferien	0.1439912

5.3. Lineare Regression

Zunaechst werden alle Univariaten Modelle erstellt.

Model	Rsquared	MAE	MAPE	pValue
Baseline	0.0000000	78901.84	0.9996864	
m1_timestampPosition	0.0022119	47193.57	2.6376768	6.11861848950379
m2_predictedStopTime	0.4392841	33381.65	0.9106335	115.024512517349
m3_distanceOutstanding	0.4366308	33412.79	0.9586479	114.406259238445
m4_speedOverGround	0.0963120	43914.03	2.3585158	-42.4235660992757
m5_waterLocksPassed	0.1015825	44781.37	2.4697040	-43.6977879848703
m6_WaterLevel	0.1852713	41975.81	2.0848742	-61.9707387165761
m7_isCargo	0.0366475	46284.60	2.5883859	25.3465307851041
m8_draught	0.0156956	47569.74	2.6169242	15.4523149993537
m9_Vacation	0.0127240	46961.10	2.6270285	14.7530632347849

```
## `geom_smooth()` using formula 'y ~ x'
```



Modell m2 mit der vorhergesagten, gewichteten Stoppzeit wird anhand der regressionsspezifischen Kennzahlen ausgewaehlt. Dieses Feature beruht auf der verbleibenden Distanz und beinhaltet auch die Anzahl passierter Schleusen. Auszerdem konnte eine Multikollinearitaet zum Wasserstand festgestellt werden. Deswegen werden diese Features im folgenden nicht evaluiert.

Model	Rsquared	MAE	MAPE	pValue
Baseline	0.0000000	78901.84	0.9996864	
m1_timestampPosition	0.0022119	47193.57	2.6376768	6.11861848950379
m2_predictedStopTime	0.4392841	33381.65	0.9106335	115.024512517349
m3_distanceOutstanding	0.4366308	33412.79	0.9586479	114.406259238445
m4_speedOverGround	0.0963120	43914.03	2.3585158	-42.4235660992757
m5_waterLocksPassed	0.1015825	44781.37	2.4697040	-43.6977879848703
m6_WaterLevel	0.1852713	41975.81	2.0848742	-61.9707387165761
m7_isCargo	0.0366475	46284.60	2.5883859	25.3465307851041
m8_draught	0.0156956	47569.74	2.6169242	15.4523149993537
m9_Vacation	0.0127240	46961.10	2.6270285	14.7530632347849
m2_4_predictedStopTime_speedOverGround	0.5019635	30132.97	0.7286956	117.276045900091
m2_7_predictedStopTime_isCargo	0.4534185	32563.03	0.9037468	113.474298879897
m2_8_predictedStopTime_draught	0.4671831	32305.63	0.9358530	112.638877116893
m2_9_predictedStopTime_Vacation	0.4646472	32030.70	0.8728865	119.395641958071

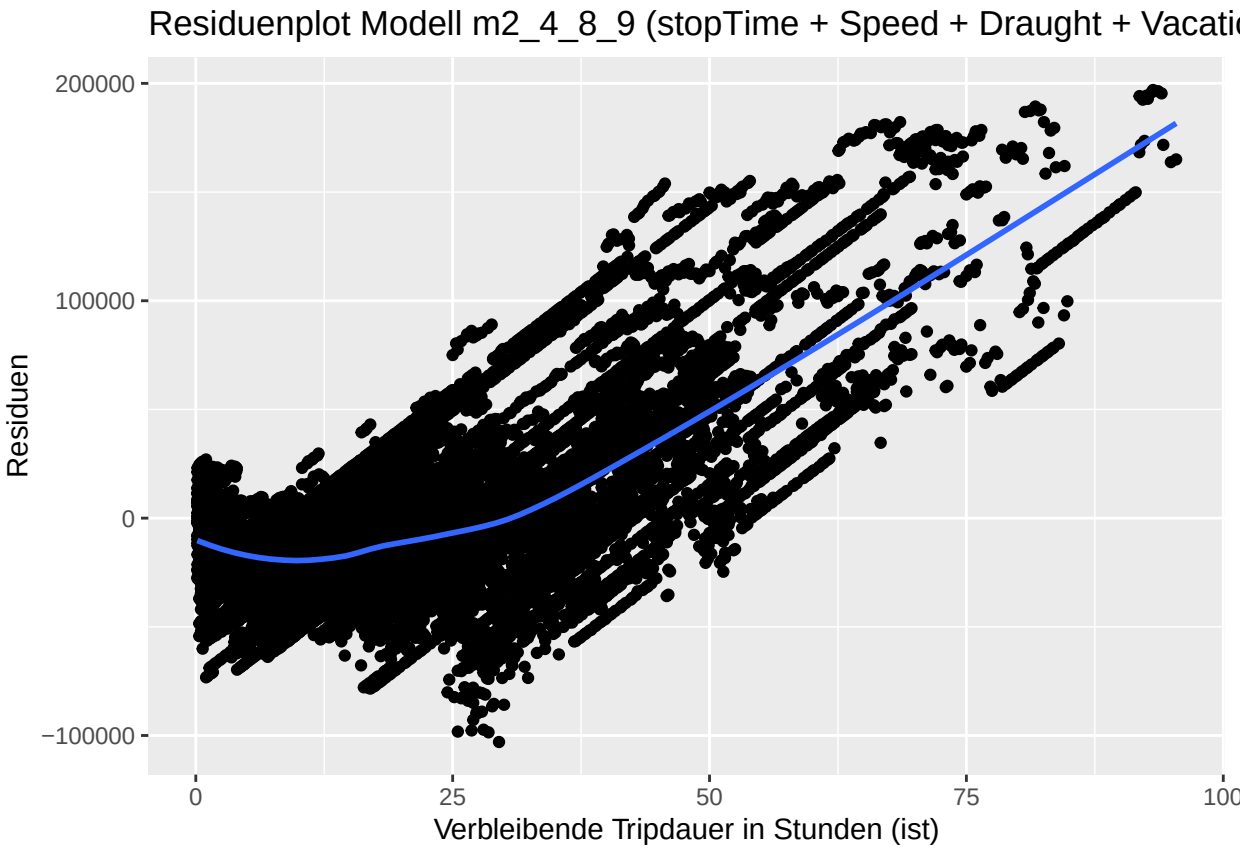
Modell m2_4, das neben der gewichteten Stoppzeit auch die aktuelle Geschwindigkeit betrachtet, wird anhand der regressionsspezifischen Kennzahlen ausgewaehlt. Es beseteht keine Multikollinearitaet zwischen der Geschwindigkeit und verbleibenden Features Auf dieser Basis werden im Folgenden alle Regressionsmodelle mit drei Features evaluiert.

Model	Rsquared	MAE	MAPE	pValue
Baseline	0.0000000	78901.84	0.9996864	
m1_timestampPosition	0.0022119	47193.57	2.6376768	6.11861848950379
m2_predictedStopTime	0.4392841	33381.65	0.9106335	115.024512517349
m3_distanceOutstanding	0.4366308	33412.79	0.9586479	114.406259238445
m4_speedOverGround	0.0963120	43914.03	2.3585158	-42.4235660992757
m5_waterLocksPassed	0.1015825	44781.37	2.4697040	-43.6977879848703
m6_WaterLevel	0.1852713	41975.81	2.0848742	-61.9707387165761
m7_isCargo	0.0366475	46284.60	2.5883859	25.3465307851041
m8_draught	0.0156956	47569.74	2.6169242	15.4523149993537
m9_Vacation	0.0127240	46961.10	2.6270285	14.7530632347849
m2_4_predictedStopTime_speedOverGround	0.5019635	30132.97	0.7286956	117.276045900091
m2_7_predictedStopTime_isCargo	0.4534185	32563.03	0.9037468	113.474298879897
m2_8_predictedStopTime_draught	0.4671831	32305.63	0.9358530	112.638877116893
m2_9_predictedStopTime_Vacation	0.4646472	32030.70	0.8728865	119.395641958071
m2_4_7_StopTime_Speed_isCargo	0.5063112	29925.31	0.7353899	116.15479027962
m2_4_8_StopTime_Speed_draught	0.5213330	29699.33	0.7838099	114.519314893686
m2_4_9_StopTime_Speed_Vacation	0.5189012	29393.53	0.7523420	120.898749737067

Modell m2_4_8, das neben der gewichteten Stoppzeit und der aktuellen Geschwindigkeit auch den Tiefgangbetrachtet, wird anhand der regressionsspezifischen Kennzahlen ausgewaehlt. Es beseteht keine Multikollinearitaet zwischen der Geschwindigkeit und verbleibenden Features Auf dieser Basis werden im Folgenden alle Regressionsmodelle mit vier Features evaluiert.

Model	Rsquared	MAE	MAPE	pValue
Baseline	0.0000000	78901.84	0.9996864	
m1_timestampPosition	0.0022119	47193.57	2.6376768	6.11861848950379
m2_predictedStopTime	0.4392841	33381.65	0.9106335	115.024512517349
m3_distanceOutstanding	0.4366308	33412.79	0.9586479	114.406259238445
m4_speedOverGround	0.0963120	43914.03	2.3585158	-42.4235660992757
m5_waterLocksPassed	0.1015825	44781.37	2.4697040	-43.6977879848703
m6_WaterLevel	0.1852713	41975.81	2.0848742	-61.9707387165761
m7_isCargo	0.0366475	46284.60	2.5883859	25.3465307851041
m8_draught	0.0156956	47569.74	2.6169242	15.4523149993537
m9_Vacation	0.0127240	46961.10	2.6270285	14.7530632347849
m2_4_predictedStopTime_speedOverGround	0.5019635	30132.97	0.7286956	117.276045900091
m2_7_predictedStopTime_isCargo	0.4534185	32563.03	0.9037468	113.474298879897
m2_8_predictedStopTime_draught	0.4671831	32305.63	0.9358530	112.638877116893
m2_9_predictedStopTime_Vacation	0.4646472	32030.70	0.8728865	119.395641958071
m2_4_7_StopTime_Speed_isCargo	0.5063112	29925.31	0.7353899	116.15479027962
m2_4_8_StopTime_Speed_draught	0.5213330	29699.33	0.7838099	114.519314893686
m2_4_9_StopTime_Speed_Vacation	0.5189012	29393.53	0.7523420	120.898749737067
m2_4_8_7_isCargo	0.5248951	29406.15	0.7681692	113.369606009636
m2_4_8_9_Vacation	0.5411410	29017.40	0.8069021	118.877359522851

```
## `geom_smooth()` using formula 'y ~ x'
```

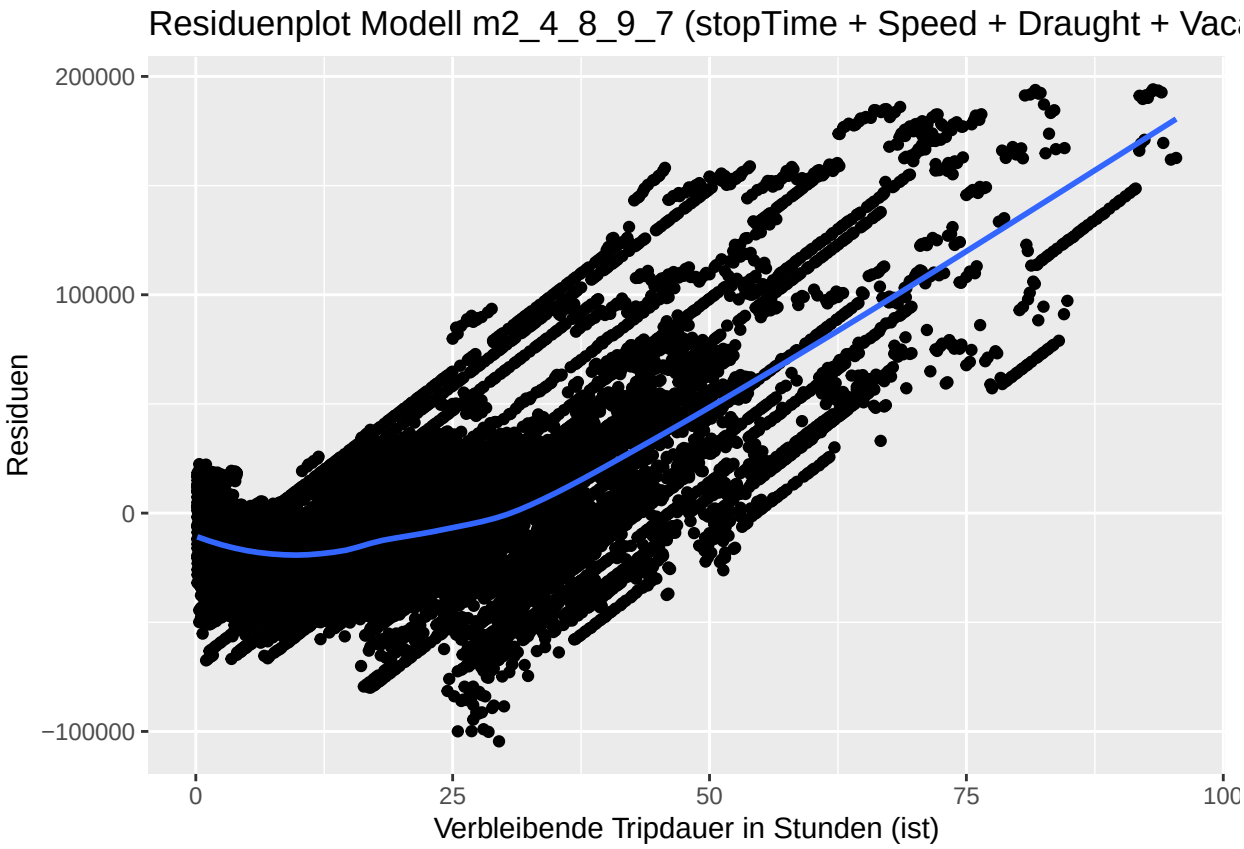


Modell m2_4_8_9, das neben der gewichteten Stoppzeit, der aktuellen Geschwindigkeit und dem Tiefgang auch die Feriendaten betrachtet, wird anhand der regressionsspezifischen Kennzahlen ausgewaehlt. Es beseteht keine Multikollinearitaet zu dem verbleibenden Feature. Auf dieser Basis werden im Folgenden alle Regressionsmodelle mit fuenf Features evaluiert.

Model	Rsquared	MAE	MAPE	pValue
Baseline	0.0000000	78901.84	0.9996864	
m1_timestampPosition	0.0022119	47193.57	2.6376768	6.11861848950379
m2_predictedStopTime	0.4392841	33381.65	0.9106335	115.024512517349
m3_distanceOutstanding	0.4366308	33412.79	0.9586479	114.406259238445
m4_speedOverGround	0.0963120	43914.03	2.3585158	-42.4235660992757
m5_waterLocksPassed	0.1015825	44781.37	2.4697040	-43.6977879848703
m6_WaterLevel	0.1852713	41975.81	2.0848742	-61.9707387165761
m7_isCargo	0.0366475	46284.60	2.5883859	25.3465307851041
m8_draught	0.0156956	47569.74	2.6169242	15.4523149993537
m9_Vacation	0.0127240	46961.10	2.6270285	14.7530632347849
m2_4_predictedStopTime_speedOverGround	0.5019635	30132.97	0.7286956	117.276045900091
m2_7_predictedStopTime_isCargo	0.4534185	32563.03	0.9037468	113.474298879897
m2_8_predictedStopTime_draught	0.4671831	32305.63	0.9358530	112.638877116893
m2_9_predictedStopTime_Vacation	0.4646472	32030.70	0.8728865	119.395641958071
m2_4_7_StopTime_Speed_isCargo	0.5063112	29925.31	0.7353899	116.15479027962

Model	Rsquared	MAE	MAPE	pValue
m2_4_8_StopTime_Speed_draught	0.5213330	29699.33	0.7838099	114.519314893686
m2_4_9_StopTime_Speed_Vacation	0.5189012	29393.53	0.7523420	120.898749737067
m2_4_8_7_isCargo	0.5248951	29406.15	0.7681692	113.369606009636
m2_4_8_9_Vacation	0.5411410	29017.40	0.8069021	118.877359522851
m2_4_8_9_7_isCargo	0.5455779	28550.10	0.7713301	117.828740514666

```
## `geom_smooth()` using formula 'y ~ x'
```



Dieses Modell hat keine weitere signifikante Verbesserung gezeigt. Deswegen wird die Berechnung an dieser Stelle abgebrochen und das vorherige Modell wird als resultierendes Prognosemodell angenommen.

5.4. Vergleich des Modells mit Baseline und Testdaten

Um das Modell auf Overfitting zu testen wird eine Vorhersage mit den Testdaten erzeugt.

Model	Rsquared	MAE	MAPE	pValue
Baseline	0.0000000	78901.84	0.9996864	
m1_timestampPosition	0.0022119	47193.57	2.6376768	6.11861848950379
m2_predictedStopTime	0.4392841	33381.65	0.9106335	115.024512517349
m3_distanceOutstanding	0.4366308	33412.79	0.9586479	114.406259238445

Model	Rsquared	MAE	MAPE	pValue
m4_speedOverGround	0.0963120	43914.03	2.3585158	-42.4235660992757
m5_waterLocksPassed	0.1015825	44781.37	2.4697040	-43.6977879848703
m6_WaterLevel	0.1852713	41975.81	2.0848742	-61.9707387165761
m7_isCargo	0.0366475	46284.60	2.5883859	25.3465307851041
m8_draught	0.0156956	47569.74	2.6169242	15.4523149993537
m9_Vacation	0.0127240	46961.10	2.6270285	14.7530632347849
m2_4_predictedStopTime_speedOverGround	0.5019635	30132.97	0.7286956	117.276045900091
m2_7_predictedStopTime_isCargo	0.4534185	32563.03	0.9037468	113.474298879897
m2_8_predictedStopTime_draught	0.4671831	32305.63	0.9358530	112.638877116893
m2_9_predictedStopTime_Vacation	0.4646472	32030.70	0.8728865	119.395641958071
m2_4_7_StopTime_Speed_isCargo	0.5063112	29925.31	0.7353899	116.15479027962
m2_4_8_StopTime_Speed_draught	0.5213330	29699.33	0.7838099	114.519314893686
m2_4_9_StopTime_Speed_Vacation	0.5189012	29393.53	0.7523420	120.898749737067
m2_4_8_7_isCargo	0.5248951	29406.15	0.7681692	113.369606009636
m2_4_8_9_Vacation	0.5411410	29017.40	0.8069021	118.877359522851
m2_4_8_9_7_isCargo	0.5455779	28550.10	0.7713301	117.828740514666
m2_4_8_9_test	NA	29474.76	0.9999639	NA

```
## Die Fehlerkennzahlen der Testdaten weichen leicht vom Trainings-Datensatz ab.
##  Der Unterschied liegt bei 457.3619 (MAE) und 0.1930618 (MAPE).
```

Die Fehlerkennzahlen der Testdaten weichen prozentual gesehen nicht grosz vom Modell ab. Ein Overfitting kann somit nicht identifiziert werden.

Im Vergleich zur Baseline hat sich insbesondere der MAE deutlich gebessert und auch der MAPE schneidet im erstellten Prognosemodell deutlich besser ab. Dieses Modell bringt demnach einen echten Mehrwert und koennte auch in der Praxis eingesetzt werden.

5.5. Auswertung und Interpretation des Modells

Das dynamische Modell wurde mit der Zielgroesze verbleibende Tripdauer (remainingTripTime) erzeugt. Die regressionsspezifischen Kennzahlen sind im Vergleich zum vorherigen Modell deutlich schlechter - das macht sie jedoch gleichzeitig deutlich plausibler. Die Zielgroesze ist viel besser fuer die exakte Ermittlung der Ankunftszeit geeignet.

Im Residuenplot kann man jedoch zwei Probleme des Modells erkennen:

1) Fuer Trips die innerhalb der durchschnittlichen Tripdauer (rund 42 Stunden) durchgefuehrt werden kann das Modell eine gute Prognose abgeben. Die Qualitaet der Prognose nimmt jedoch bei auszerplanmaeszigen Fahrten schnell ab.

2) Es gibt auffaellige lineare Muster im Residuenplot. Diese konnten auch im endgueltigen Modell nicht vollstaendig behoben werden. Dies weiszt darauf hin, dass es moeglicherweise weitere Features gibt, die in unserem Modell keine Beachtung finden.