Fallstudie

Model Engineering (DLMDWME01)

im Master Studiengang Data Science

Vorhersagemodell für den täglichen Bedarf an Bereitschaftsfahrenden beim DRK Berlin

Verfasser: Georg Grunsky

Matrikelnummer: IU14072015

Tutor: Markus Pak

Datum: 6. Juni 2025

I Abstract

Der Berliner Rot Kreuz Rettungsdienst (DRK Berlin) projektiert den Einsatz eines Machine Learning Systems zur effizienten Planung eines Bereitschaftsdienstplanes für Einsatzfahrer:innen. Auf Basis einer erfolgten Use Case Analyse, stellt diese Arbeit den Projektplan vor und behandelt sowohl die Projektstruktur und -teamplanung als auch die Zeitplanung, die Kostenschätzung, die Stakeholderanalyse und das Risikomanagement des Projektes.

Für den Projektmanagement-Ansatz wird eine hybride Vorgehensweise verfolgt. Agile Methoden, wie Scrum, werden mit klassischen Elementen des V-Modell XT kombiniert. Das schafft eine klare Projektstruktur mit definierten Phasenübergängen und Planungswerkzeugen und gewährleistet gleichzeitig die Vorteile der flexiblen und iterativen Entwicklung. Das V-Modell XT wurde auf das notwendigste beschränkt um ein kleines Projektteam, einen geringen „Overhead“ und eine modulare Struktur zu ermöglichen.

Mit den verwendeten Elementen des V-Modell XT wurde in manchen Bereichen zwar über die Vorgaben der Aufgabenstellung hinausgeschossen, die Dokumentation des Rahmenprozesses bietet jedoch eine hervorragende Unterstützung bei gleichzeitiger Ermutigung zur Flexibilität. Man bekommt das Gefühl, für den Projektstart „bereit“ zu sein. Der, in der Aufgabenstellung verlangte, Projektstrukturplan kommt in der Dokumentation des V-Modell XT (Angermeier et al. 2024) nicht vor. Leider ist auch in den vorgestellten Modellen des Studienmoduls „Management von IT-Projekten“ (IU Internationale Hochschule 2024), das diesem Modul üblicherweise vorangeht, kein Projektstrukturplan abgebildet. Den ersten zufriedenstellenden Anhalt dazu lieferte Wikipedia (2022). Beim Projektplan finden die Anforderungen der Aufgabenstellung sowie das V-Modell XT aber wieder zusammen. Als mögliche Ansatzpunkte wurden weiters The PM Minimalist Quick Start Guide (Greer 2011) sowie die Lektüre von How Big Things Get Done (Flyvbjerg / Gardner 2023) herangezogen, diese lieferten jedoch nicht die gehoffte Unterstützung für die Aufgabe.

Die inhaltliche Vorgabe des Projektes ist es, kosteneffizient die tägliche Anzahl des benötigten Bereitschaftspersonals vorherzusagen, gleichzeitig jedoch nie zu wenig Einsatzfahrende als Reserve vorzusehen. Das System und das Projekt bauen sich rund um diese Vorgabe auf. Die Planung sieht vor mit einem Entwicklungsteam in mehreren, teils parallelen Sprints, diese Minimalanforderung umzusetzen, sinnvoll um weitere Funktionen zu erweitern und das System, mit entsprechender Hardware und benutzerfreundlichen Frontends, in eine stabile Umgebung einzubetten, die sich gut in den IT-Betrieb des DRK Berlin integrieren lässt. Die wahrscheinlich langen Lieferzeiten der Hardwarekomponenten geben der Entwicklung einen zeitlichen Rahmen für die Umsetzung der Sprints. Inhaltlich wird dieses Projekt nicht als „zeitkritisch“ eingestuft. Gegebenenfalls wirkt sich daher eine Verzögerung zwar auf die, der Kostenstelle des Projektes zugeschriebenen, Personalkosten jedoch nicht auf sonstige Aspekte und Abhängigkeiten der Umsetzung aus. Der Fokus der Entwicklung liegt, auch in der Priorisierung der Arbeitspakete, auf den, für das Machine Learning Modell relevanten Aufgaben. Das Vorhersagemodell selbst bildet das Herzstück der Aufgabe. In der Zeit- und Kostenschätzung wurde

daher der Einsatz eines/r weiteren Data Scientist zwar erwogen, jedoch auch wieder verworfen.

Anhand der exemplarisch dargestellten Risiken, ist auch ersichtlich, dass nicht nur ein potentieller Reputationsschaden für das DRK (aufgrund einer zu niedrig prognostizierten Anzahl nötiger Bereitschaftsfahrender und damit mangelnder Einsatzbereitschaft) das Projekt gefährden kann, sondern auch eine mögliche Ablehnung durch die Nutzer, die durch die Umsetzung mehr Verantwortung übernehmen müssen als in den derzeitigen Arbeitsprozessen. Dies sind u.a. zwei wesentliche Punkte, denen im Rahmen der Projektumsetzung, bei der Gestaltung des Modells und der Regelung der zukünftigen Arbeitsprozesse, unbedingt Rechnung getragen werden muss. Die zukünfigen Nutzer werden daher bereits ab Projektbeginn sowohl für das Requirements Engineering als auch für die Qualitätssicherung hinzugezogen.

I.1 Making of

Ausgangspunkt dieser Arbeit war ein MachineLearning Szenario, das dem Modul „Model Engineering“ (Pak 2024) dieses Studienganges entnommen wurde. Dank der Zustimmung der zuständigen Tutoren wurde es möglich, dasselbe Thema in drei unabhängigen aber inhaltlich dennoch aneinander anschließenden Modulen zu bearbeiten. Dadurch ergab sich eine durchgängige Bearbeitung des Problemstellung durch mehrere Entstehungsphasen eines Machine Learning Systems hindurch.

1. **Modul „Data Science UseCase“ :** Analyse des Anwendungsfalls (ML-Canvas) bis zur Präsentation für die Freigabe durch das Präsidium.
2. **Modul „Technische Projektplanung“ :** die gegenständliche Arbeit
3. **Modul „Model Engineering“ :** Die Umsetzung des Vorhersagemodells mit bereitgestellten Trainingsdaten

Die ganzheitliche Betrachtungsweise der Thematik ließ bereits jetzt einen besseren Einblick in die Prozesse und Herausforderungen der Softwareprojektabwicklung mit Machine Learning Aspekten entstehen.

Für die technische Bearbeitung der Aufgabenstellung wurden mehrere Werkzeuge verwendet. In vorangegangenen Modulen hat sich bereits die Kombination eines Git-Repositories für die LATEX-Dateien der Bearbeitung bewährt. Zusätzlich wurde probehalber das Repository auch einem Git-Projekt hinzugefügt um die Zeitschätzung des Projektes in Form von Iterations, Meilensteinen und Issues über das Git-Projekt zu gestalten. Dies erwies sich zwar als machbar, aber nicht als „sauber“ , da im Repository, die tatsächlichen Issues der Portfoliophasen mit den fiktiven Issues des Projektes vermischt wurden. Eine Testversion der Software „objectiF RPM“ (microTool GmbH 2024) unterstützte zwar der Entwicklung eines Verständnisses für die Herausforderungen in der Projektplanung, eine sinnvolle Herangehensweise und auch für Details der Umsetzung (wie zB Reviewmöglichkeiten für Anforderungen und „Slicing“ von Use Cases), war jedoch für die tatsächliche Verwendung im Rahmen der Portfolioaufgabe zu Umfangreich und hätte den Rahmen gesprengt. Das GANTT-Diagramm wurde anschließend mit dem relativ intuitiv gehaltenem Tool „YouTrack“ von JetBrains erstellt. Gerade, die Möglichkeit die Arbeitspakete hierarchisch zu strukturieren und zeitlich einfach anzupassen, machten das GANTT-Diagramm zu einem wertvollen Werkzeug, dass gute Einblicke in die Erfordernisse der Projektabwicklung bot.

Inhaltsverzeichnis

II Abkürzungsverzeichnis

AI Artificial Intelligence
KI Künstliche Intelligenz
GPT Generative Pre-trained Transformer Model
PPO Proximal Policy Optimization
NLP Natural Language Processing
RLHF Reinforcement Learning from Human Feedback
FAQ Frequently Asked Questions

1 Einleitung

> Conversational interfaces are popular, but they are far from intelligent. Amazon' Alexa and other voice-response interfaces don't understand language. They simply launch computerized sequences in response to sonic sequences, which humans call verbal commands.

Die Abkürzung „GPT“ steht hierbei für Generative Pre-trained Transformer Model (GPT). In einem Interview mit Melanie Subbiah, einer Autorin des ersten GPT-3 Konzepts und Artificial Intelligence (AI)-Ingenieur bei OpenAI, wird die Bedeutung der Abkürzung anschaulich erklärt:

Im gewählten Szenario geht es um die Herausforderung, die Planungslogik des Bereitschaftsdienstplans für den Berliner Rotkreuz-Rettungsdienst zu verbessern. Das, weiter unten beschriebene, Thema ist dem Modul „Model Engineering“ des Masterstudienganges Data Science entnommen (Pak 2024), und wurde für zwei weitere Module adaptiert:

1. **Data Science UseCase:** Analyse des Anwendungsfalls bis zur Präsentation für die Freigabe durch das Management.
2. **Technische Projektplanung:** Aufbauend auf dem PitchDeck vom Projektstrukturplan über eine Kostenschätzung und das Risikomanagement bis zur Stakeholder-Analyse.
3. **Model Engineering:** Die Umsetzung des UseCase als Projekt in einem Git-Repository mit bereitgestellten Daten bis zu einem fertiggestellten Vorhersagemodel.

Die Motivation, dieses Thema auch im Modul „Data Science UseCase“ zu bearbeiten, besteht darin, die Problemstellung aus mehreren Blickwinkeln zu behandeln und so ein durchgängiges Verständnis für die Abwicklung realer Szenarien zu erlangen. Die ganzheitliche Betrachtungsweise der Thematik lässt einen tieferen Einblick in die Prozesse der Data Science entstehen, erhöht die Identifikation mit dem UseCase und macht somit schlussendlich auch mehr Spaß.

2 Das Vorhersagemodell

2.1 Organisiere

das Projekt mithilfe der CRISP-DM oder der MS Team Data Science Methode. Mache einen Vorschlag, wie die Ordnerstruktur eines Git-Repositories für das Projekt aufgebaut werden soll. Beachte, dass Du den finalen Code des Projekts nicht nach dieser Ordnerstruktur aufbauen musst.

Warum für TDSP entschieden moderner - optimiert für agile Entwicklung - gegenständliches Projekt hat ebenfalls Anfang und Ende - flexiblere Gestaltung des Arbeitsprozesses da alle großen „Überschriften“ von einander abhängen (oder so...)

Der Link zu TDSP aus dem Studienscript, aber auch von den referenzierenden GitHub-Seiten aus ist auf Microsoft nicht mehr verfügbar. Das stattdessen angezeigte Dokument AI-Implementierung enthält zwar einen Verweis auf TDSP mit einem Link, dieser führt jedoch ebenfalls wieder zum selben AI-Implementierungsdokument. => Anm: schnell verändernde Branche

Schritte TDSP (**?**)

2.1.1 Business Understanding

(Referenz auf vorangegangene Arbeiten)

2.1.2 Data Acquisition and Understanding

Referenz UseCase Analyse Bestehende Daten und Hinweise auf mögliche Erweiterung Explorative Datenanalyse

2.1.3 Modeling

Referenz UseCase Analyse Referenz Monitoring (UseCase Analyse)

2.1.4 Deployment

Referenz Monitoring (UseCase Analyse) Referenz Projektplanung

2.1.5 Customer Acceptance

Referenz Risiko in der Projektplanung

2.1.6 Ordnerstruktur

Git Link Das Programm wurde vom Autor mit dem Namen *„functionfinder“* betitelt und steht auf GitHub unter dem Link

`https://github.com/GGProjects/DLMDWPMP01`

zum Download zur Verfügung.

Die weitere Ordnerstruktur orientiert sich an einem Blog-Post von **?**, der, nach Meinung des Autors, ein leicht verständliches Framework für Pythonprojekte bietet. Bereitgestellt über (**?**)

Fusioniert mit (**?**) ... Umsetzung nicht in einer Azure Umgebung, daher diesbezgl modifiziert.

In der nachfolgenden Abbildung werden die wesentlichen Ordner und Dateien der bereitgestellten Programmstruktur dargestellt. Hervorzuheben ist hierbei das Package *functionfinder*, das sämtliche Pythonmodule beinhaltet, die für die Programmfunktionalität verantwortlich sind, sowie das Modul *ffrunner*, über das die eigentliche Ausführung des Programmes gestartet wird. Die übrigen Verzeichnisse des Projekts stehen für die bereitgestellten Daten (*data*), die Dokumentation (*docs*), die vom Programm erzeugten Ausgaben (*output*), sowie für die vorgesehenen UnitTests (*tests*) zur Verfügung.

Ordnerstruktur ist für den Bedarf dieses Projektes zu umfangereich, Idee ist aber ein template für zukünftige Projekte zu schaffen, wo Modelle in eine Gesamtsoftware eingebetet sind. Die einzelnen Modell- und Packageordner könnten auch als eigene Git-Repos angelegt werden. Die Nummerierung sorgt für gleichbleibende Anordnung und leicht verwaltbare Strukturen

Ordnerstruktur ist sicher noch ausbaufähig, Änderungen werden sich noch ergeben.

2.2 Beurteile die Qualität

des zur Verfügung gestellten Datensatzes. Bereite Deine Erkenntnisse so auf und visualisiere sie so, dass Businesspartner in einer klaren und einfachen Weise die wichtigen Zusammenhänge verstehen können.

Stelle ein

2.3 erstes Basismodell

(ein sogenanntes Baseline-Modell) auf, sowie ein präzises Vorhersagemodell, das den Businessanforderungen genügt, nämlich die Erfolgsrate der Kreditkartenzahlungen zu erhöhen und gleichzeitig die Transaktionskosten gering zu halten.

Damit die Businesspartner Vertrauen in Dein neues

2.4 Modell

entwickeln, solltest du die Wichtigkeit der einzelnen erklärenden

2.4.1 Variablen

diskutieren und die

2.4.2 Modellresultate

so interpretierbar wie möglich gestalten. Außerdem ist eine detaillierte

2.4.3 Fehleranalyse

sehr wichtig, damit die Businesspartner auch die Schwachstellen Deiner Herangehensweise verstehen.

Im letzten Schritt des Projekts soll ein

2.5 Vorschlag

unterbreitet werden, wie Dein Modell in die tägliche Arbeit des Fachbereichs eingebunden werden kann, beispielsweise wie eine graphische Benutzeroberfläche (GUI) aussehen könnte.

Abb. 1: Struktur des Ordneraufbaus

```
DMLDWME01
|
+---00_docs                          # Dokumente
|   +---01_general_basics            # unspezifische Grundlagen
|   +---02_project_basics            # projekspezifische Grundlagen
|   \---03_artifacts                 /# Projektprodukte
|       +---00_paper                 # Latex Files dieser Arbeit
|       +---04a_model_standby        # Dokumentation eines Modells
|           +---01_dataanalysis      # Reports der Datenanalyse
|           \---02_modeling          # Reports der Modellerstellung
|               \---[...models]      # Subfolder einzelner Modelle
|
+---01_data                          # Datenbereitstellung
|   +---01_lake                      # Data Lake - Rohdatenspeicher
|   |   \---01_use_case_2            # Rohdaten dieser Aufgabenstellung
|   |
|   \---02_warehouse                 # Data Warehouse - Aufbereitete Daten
|
+---02_config                        # Konfigurationsdateien
|
+---03_main                          # Hauptmodul der Anwendung
|
+---03[a-y]_...                      # einzelne Packages der Anwendung
|
+---03z_helpers                      # Hilfscode - wird bei Abschluss entfernt
|
+---04a_model_standby                /# 04[a-z] Ordner der ML-Modelle
|   +---00_sample_data               # Beispieldaten zur Dokumentation
|   |   +---01_raw                   # Rohdatenauszug
|   |   +---02_processed             # Auszug aufbereiteter Daten
|   |   \---03_for_modelling         # Auszug der Trainingsdaten
|   |
|   +---01_dataanalysis              # Code explorativer Datenanalyse
|   +---02_modeling                  # Code der Modellerstellung
|   \---03_deployment                # Code des Modeldeployments
|       \---[modelname].R            # projektspezifische Namenskonvention
|
+---05_output                        # Generierte Ergebnisse
|  +---01_documentation              # Softwaredokumentation
|  +---02_data                       # Datenausgabe
|  +---03_figures                    # Grafiken
|  +---04_logs                       # Logausgaben
|  \---05_apps                       # User Interfaces
|
\---06_tests                         # Unittests
```

Quelle: Eigene Darstellung.

3 Zusammenfassung

4 Literaturverzeichnis

Angermeier, D./Bartelt, C./Bauer, O./Beneken, G./ Wittmann, M. (**2024**): *V-Modell-XT Das deutsche Referenzmodell für Systementwicklungsprojekte Version 2.4*.

Flyvbjerg, B./ Gardner, D. (**2023**): *How Big Things Get Done: The Surprising Factors That Determine the Fate of Every Project, from Home Renovations to Space Exploration and Everything in Between*. Currency, New York.

Greer, M. (**2011**): *The PM Minimalist Quick Start Guide: The Absolute Least You Can Do!*

IU Internationale Hochschule (**2024**): *Management von IT-Projekten*.

microTool GmbH (**2024**): *objectiF RPM Benutzerhandbuch*.

Pak, M. (**2024**): *Aufgabenstellung_DLMDWME01*.

Wikipedia (**2022**): *Projektstrukturplan*.

5 Abbildungsverzeichnis

6 Tabellenverzeichnis

A Annexes

A.1 Datenbeschreibung

Dateiname: ***sickness_table.csv***

Dateigröße (Kb): ***48.09***

Originaldaten gespeichert in: ***01_data/01_lake/01_use_case_2/***

Aufgrund der geringen Größe wurde der gesamte Datensatz ebenso in *** **00_sample_data/01_raw*** hinterlegt.

Der Datensatz enthält ***1152*** Datenpunkte mit jeweils ***8*** Merkmalen und wurde als Datentyp ***spec_tbl_df, tbl_df, tbl, data.frame*** angelegt.

Die Daten weisen folgende Struktur auf:

```
cols(
  ...1 = col_double(),
  date = col_date(format = "%Y-%m-%d"),
  n_sick = col_double(),
  calls = col_double(),
  n_duty = col_double(),
  n_sby = col_double(),
  sby_need = col_double(),
  dafted = col_double()
)
```

Die u.a. Beschreibung der Daten wurde dem Dokument “Zusatzinformationen_DLMDWME01 (**pak_zusatzinformationen_dlmdwme01_2024?**) entnommen. Hiernach geben die enthaltenen Daten Auskunft über Metriken (auf Tagesbasis) des Einsatzfahrten-Bereitschaftspersonals des Berliner RotKreuz Rettungsdienstes. Diese tageweisen Zeitreihendaten umfassen den Zeitraum von ***2016-04-01*** bis ***2019-05-27***.

Beschreibung der einzelnen Datenmerkmale

Variable	Beschreibung
…1	Zähler
date	Datum
n_sick	Anzahl der Einsatzfahrer, die einen Krankenstand angemeldet haben
calls	Anzahl der Notrufe
n_duty	Anzahl der Einsatzfahrer im Dienst
n_sby	Anzahl der verfügbaren Ersatzfahrer
sby_need	Anzahl der Ersatzfahrer, die aktiviert werden
dafted	Anzahl der zusätzlichen Einsatzfahrer, die aktiviert werden müssen, wenn die Anzahl der Einsatzfahrer im Bereitschaftsdienst nicht ausreicht

A.2 explorative Datenanalyse

Einleitung

Dieses Dokument beschreibt die vorbereitenden Abschnitte **Prüfung der Datenqualität** und **Explorative Datenanalyse** für den bestehenden Use Case. Ziel der exporativen Analyse ist dabei auch das Finden eines möglichen Ansatzes für die Vorhersage und die damit einhergehende Modifikation und Aufbereitung der Daten. Die Modellierungen selbst werden in den jeweiligen Modellverzeichnissen behandelt.

Prüfung der Datenqualität

Datensätze einlesen

Die Daten werden in zwei Dateien bereitgestellt: ***sickness_table.csv*** und ***sickness_table.xlsx***. Augenscheinlich handelt es sich dabei um den selben Datensatz,was jedoch eingangs noch zu überprüfen ist.

```
require(readr)
path_to_samples <- "../00_sample_data/"
sickness_csv <- read_csv(paste0(path_to_samples,"01_raw/sickness_table.csv"),
    col_types = cols(date = col_datetime(format = "%Y-%m-%d")))
head(sickness_csv)
```

```
# A tibble: 6 x 8
   ...1 date                n_sick calls n_duty n_sby sby_need dafted
  <dbl> <dttm>               <dbl> <dbl>  <dbl> <dbl>    <dbl>  <dbl>
1     0 2016-04-01 00:00:00     73  8154   1700    90        4      0
2     1 2016-04-02 00:00:00     64  8526   1700    90       70      0
3     2 2016-04-03 00:00:00     68  8088   1700    90        0      0
4     3 2016-04-04 00:00:00     71  7044   1700    90        0      0
5     4 2016-04-05 00:00:00     63  7236   1700    90        0      0
6     5 2016-04-06 00:00:00     70  6492   1700    90        0      0
```

```
require(readxl)
sickness_xls <- read_excel(paste0(path_to_samples,"01_raw/sickness_table.xlsx"),
    col_types = c("numeric", "date", "numeric",
        "numeric", "numeric", "numeric", "numeric",
        "numeric"))
head(sickness_xls)
```

```
# A tibble: 6 x 8
   ...1 date                n_sick calls n_duty n_sby sby_need dafted
  <dbl> <dttm>               <dbl> <dbl>  <dbl> <dbl>    <dbl>  <dbl>
1     0 2016-04-01 00:00:00     73  8154   1700    90        4      0
2     1 2016-04-02 00:00:00     64  8526   1700    90       70      0
3     2 2016-04-03 00:00:00     68  8088   1700    90        0      0
4     3 2016-04-04 00:00:00     71  7044   1700    90        0      0
5     4 2016-04-05 00:00:00     63  7236   1700    90        0      0
6     5 2016-04-06 00:00:00     70  6492   1700    90        0      0
```

Datensätze vergleichen

```
identical(sickness_csv, sickness_xls)
```

```
[1] FALSE
```

Die Überprüfung mittels *identical* besagt, dass die Datensätze nich identisch sind. Die in den jeweiligen Daten-Dictionaries beschriebenen Datenstrukturen lassen jedoch schon darauf schließen. Eine Summary der Wertebereiche der vorliegenden Daten schafft mehr Klarheit.

Daten Summary

sickness_csv

```
summary(sickness_csv)
```

```
      ...1            date                         n_sick      
 Min.   :   0.0   Min.   :2016-04-01 00:00:00   Min.   : 36.00  
 1st Qu.: 287.8   1st Qu.:2017-01-13 18:00:00   1st Qu.: 58.00  
 Median : 575.5   Median :2017-10-28 12:00:00   Median : 68.00  
 Mean   : 575.5   Mean   :2017-10-28 12:00:00   Mean   : 68.81  
 3rd Qu.: 863.2   3rd Qu.:2018-08-12 06:00:00   3rd Qu.: 78.00  
 Max.   :1151.0   Max.   :2019-05-27 00:00:00   Max.   :119.00  
     calls           n_duty         n_sby       sby_need          dafted       
 Min.   : 4074   Min.   :1700   Min.   :90   Min.   :  0.00   Min.   :  0.00  
 1st Qu.: 6978   1st Qu.:1800   1st Qu.:90   1st Qu.:  0.00   1st Qu.:  0.00  
 Median : 7932   Median :1800   Median :90   Median :  0.00   Median :  0.00  
 Mean   : 7920   Mean   :1821   Mean   :90   Mean   : 34.72   Mean   : 16.34  
 3rd Qu.: 8828   3rd Qu.:1900   3rd Qu.:90   3rd Qu.: 12.25   3rd Qu.:  0.00  
 Max.   :11850   Max.   :1900   Max.   :90   Max.   :555.00   Max.   :465.00  
```

sickness_xls

```
summary(sickness_xls)
```

```
      ...1            date                         n_sick      
 Min.   :   0.0   Min.   :2016-04-01 00:00:00   Min.   : 36.00  
 1st Qu.: 287.8   1st Qu.:2017-01-13 18:00:00   1st Qu.: 58.00  
 Median : 575.5   Median :2017-10-28 12:00:00   Median : 68.00  
 Mean   : 575.5   Mean   :2017-10-28 12:00:00   Mean   : 68.81  
 3rd Qu.: 863.2   3rd Qu.:2018-08-12 06:00:00   3rd Qu.: 78.00  
 Max.   :1151.0   Max.   :2019-05-27 00:00:00   Max.   :119.00  
     calls           n_duty         n_sby       sby_need          dafted       
 Min.   : 4074   Min.   :1700   Min.   :90   Min.   :  0.00   Min.   :  0.00  
 1st Qu.: 6978   1st Qu.:1800   1st Qu.:90   1st Qu.:  0.00   1st Qu.:  0.00  
 Median : 7932   Median :1800   Median :90   Median :  0.00   Median :  0.00  
```

```
 Mean   : 7920   Mean   :1821   Mean   :90   Mean   : 34.72   Mean   : 16.34
 3rd Qu.: 8828   3rd Qu.:1900   3rd Qu.:90   3rd Qu.: 12.25   3rd Qu.:  0.00
 Max.   :11850   Max.   :1900   Max.   :90   Max.   :555.00   Max.   :465.00
```

Gemäß den angezeigten Wertebereichen handelt es sich tatsächlich um die gleichen Datensätze. Anhand der .csv Datei wird noch überprüft ob der Datensatz fehlende Werte beinhaltet, bevor mit der weiteren Datenexploration fortgefahren wird.

NAs und “Missing Values”

```
alldays <- seq(from = as.Date("2016-04-01"),
               to = as.Date("2019-05-27"),
               by = 1)

if (identical(alldays, as.Date(sickness_csv$date))) {
  print("no missing dates")
  } else {
  print("you have to deal w/ missing dates")}
```

```
[1] "no missing dates"
```

```
if (!anyNA.data.frame(sickness_csv)) {
  print("no missing values")
  } else {
    print("you have to deal w/ missing values")}
```

```
[1] "no missing values"
```

In dieser Zeitreihe müssen weder mit fehlenden Zeitwerten noch NAs in den Datenpunkten behandelt werden. Die Prüfung der Datenqualität wird somit abgeschlossen.

Zusammenfassung

1. Die bereitgestellten Datensätze scheinen identisch zu sein. Es wird nur mit der .csv Datei weitergearbeitet.
2. Es gibt keine fehlenden Zeitwerte und/oder NAs in den Datenpunkten
3. Der bisherige Fixwert von 90 Bereitschaftspersonen (*n_sby*) scheint in den meisten Fällen im Vergleich zu *sby_need* zu hoch zu sein. Es gibt jedoch auf Tage wo mehr als 90 Personen gebraucht werden.
4. Die Zeitreihe behandelt den Zeitraum von 2016-04-01 bis 2019-05-27. Eine Periode von drei Jahren vor der COVID Pandemie. Es ist davon auszugehen, dass COVID die aktuellen Zahlen verändert hat. Denoch hilft eine erfolgreiche Umsetzung einer Vorhersage auf den Testdaten, das Projekt mit aktuellen Daten voranzutreiben und zukünftig Bereitschaftskosten sparen zu können.
5. Die Anzahl der Anrufe und die Anzahl des krankgemeldeten Personals scheint ähnliche Proportionen aufzuweisen, nur um etwa den Faktor 100 kleiner.

Explorative Datenanalyse

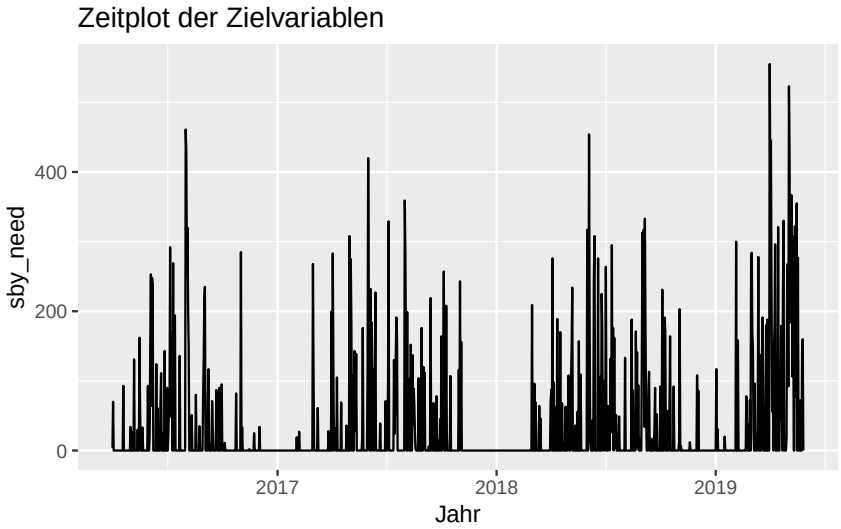
Hyndman und Athanasopoulos (2021) bietet von der Exploration bis zur Modellierung einen sehr ausführlichen Anhalt für die Arbeit mit Zeitreihendaten. Für die Analyse in diesem Dokument wird das dazugehörige R-Package *fpp3* verwendet.

Ein weiteres R-Package, das gute Werkzeuge für die explorative Analyse aber auch für die Erstellung von Models anbietet ist *caret*. Das Paket eignet sich zwar weniger gut für Zeitreihendaten, aus der Dokumentation alleine kann aber bereits viel gelernt werden. (Kuhn 2019).

Visualisierung der Zielvariable

Die Aufgabenstellung liegt in der Vorhersage des bereitzuhaltenden Bereitschaftspersonals auf Basis einer vermuteten Saisonalität im historischen Bedarf. Die Zielvariable für diese Vorhersage ist daher *sby_need*, die das tatsächlich benötigte Bereitschaftspersonal pro Tag ausdrückt. Die u.a. Darstellung der Zielvariablen gibt einen ersten Eindruck über deren Ausprägung und mögliche Herausforderungen. Um das R-Package *fpp3* für Forcasting in Zeitreihen zu verwenden, wird der Datensatz vorerst in ein entsprechendes Objekt konvertiert. Hierbei werden auch die erste Spalte (*id*), die bisher konstante Einteilung des Bereitschaftspersonals (*n_sby*), sowie die Anzahl der zusätzlich benötigten Fahrer:innen (*dafted*) entfernt.

```
ts_sby <- sickness_csv %>%
  select(-c("...1", "n_sby", "dafted")) %>%
  mutate(date = as.Date(date)) %>%
  as_tsibble(index = date)

ts_sby %>%
  autoplot(sby_need) +
  labs(x = "Jahr",
       title = "Zeitplot der Zielvariablen")
```

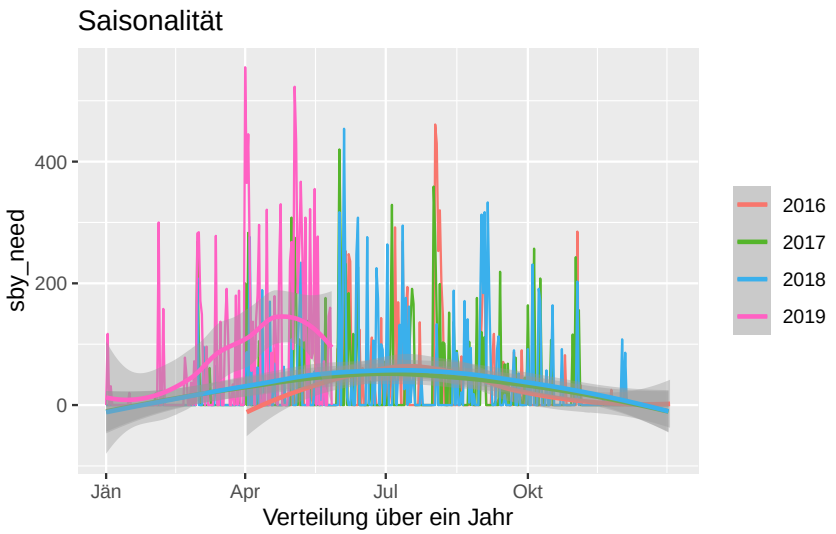


Die vermutete Saisonalität ist in der zeitlichen Darstellung des Bedarfs gut erkennbar. Augenscheinlich scheint es monatliche Spitzen in der Zielvariablen zu geben und um den Jahreswechsel ist oft nur

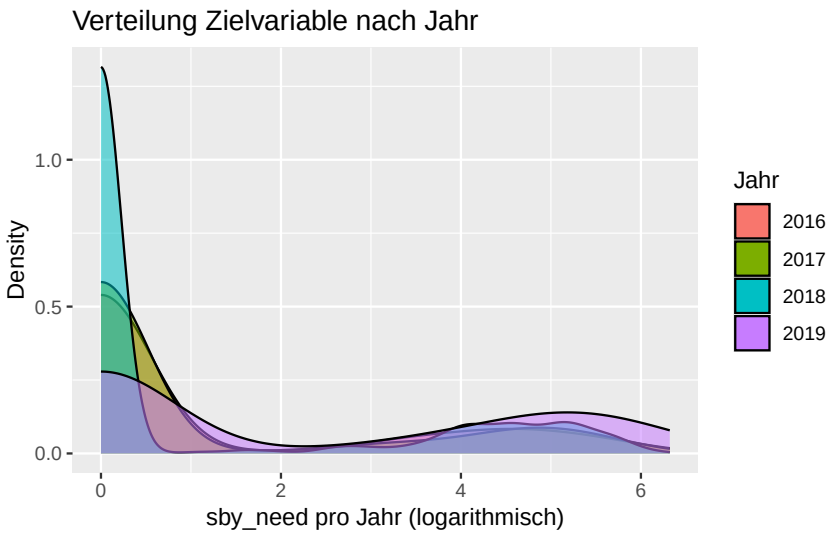
geringer Bedarf vorhanden. Mit dem Jahreswechsel auf 2019 ändert sich das Muster der Bedarf weist ab dem Jahreswechsel deutlich höhere Spitzen auf und auch um in der “sonst ruhigen” Zeit zwischen 2018 und 2019 sind *Peaks* erkennbar. Die rapiden Wechsel zwischen “kein Bedarf” und “sehr hoher Bedarf” an Bereitschaftspersonal könnten ein Vorhersagemodell vor eine Herausforderung stellen, vor allem da eine Vorgabe lautet, nie zu wenig Bereitschaftspersonal vorzusehen. Auch der Wechsel der Muster mit 2019 könnte zur Auswirkung haben, dass sich die historischen Daten nur schlecht als Prädiktoren für die zukünftige Entwicklung eignen.

Im u.a. Saisonalitätsplot werden die deutlich stärker ausgeprägten “Peaks” im Jahr 2019 auch durch die Smooth-Curve gut ersichtlich. Diese Spitzenwerte zeigen auch im Vergleich zwischen den Jahren immer wieder Abweichungen zu einander und treten nicht zu exakt den gleichen Zeitpunkten auf. Auch das dargestellte 99% Confidence Level der Smooth-Curve deckt diese nicht ab. Die explizite Vorgabe “immer ausreichend” Bereitschaftspersonal vorzusehen” wird dadurch erschwert.

```
ts_sby %>%
  gg_season(sby_need, period = "1y") +
  geom_smooth(level = 0.99) +
  labs(title = "Saisonalität",
       y = "sby_need",
       x = "Verteilung über ein Jahr")
```




```
ts_sby %>%
  ggplot() +
  geom_density(aes(x = log(sby_need + 1), fill = as.factor(year(date)), alpha = 0.3),
                   show.legend = c(fill = TRUE, alpha = FALSE)) +
  labs(title = "Verteilung Zielvariable nach Jahr",
       y = "Density",
       x = "sby_need pro Jahr (logarithmisch)",
       fill = "Jahr")
```

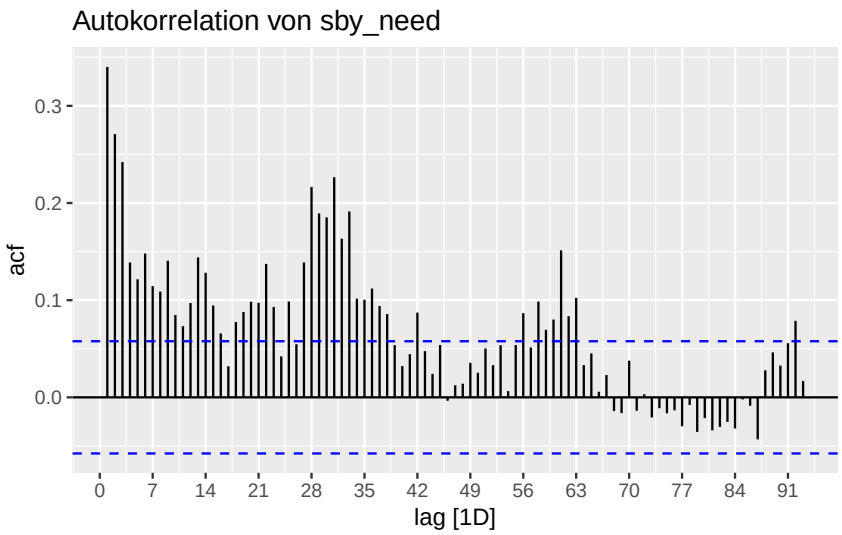


Wie bereits zuvor erläutert, sieht man auch in der Dichtefunktion des logarithmisch dargestellten Wertes von *sby_need* den Wechsel zwischem geringen und sehr hohem Bedarf. Eine Normalverteilung liegt logischerweise nicht vor, da der Modus zwar bei 0 zu liegen scheint, ein negativer Wert aber freilicherweise nicht vorkommen kann. Sieht man von der Häufung zum den Wert 0 ab, erkennt man eine zweite kleinere Akkumulation von Werten zwischen grob 3.5 und 6.5. Auch hier zieht das Jahr 2019 die Werte wieder etwas nach oben. In nicht logarithmische ausgedrückten Werten wäre das zwischen ca. 33 und 665 Fahrer:innen, die bereitgehalten werden müssen. Man könnte argumentieren, dass ein Vorhalten von 0 Standby-Fahrer:innen unrealistisch ist und aus dieser Darstellung bereits einen Mindestwert von ca. 35 Personen in Bereitschaft ableiten. Kostenmäßig wäre dies bereits eine Ersparnis gegenüber den derzeit 90 vorgehaltenen Bereitschaftsfahrenden. Die Herausforderung liegt abermals in der treffsicheren Vorhersage der, in augenscheinlich monatlichen Abständen auftretenden, Spitzenwerte.

Autokorrelation der Zielvariable

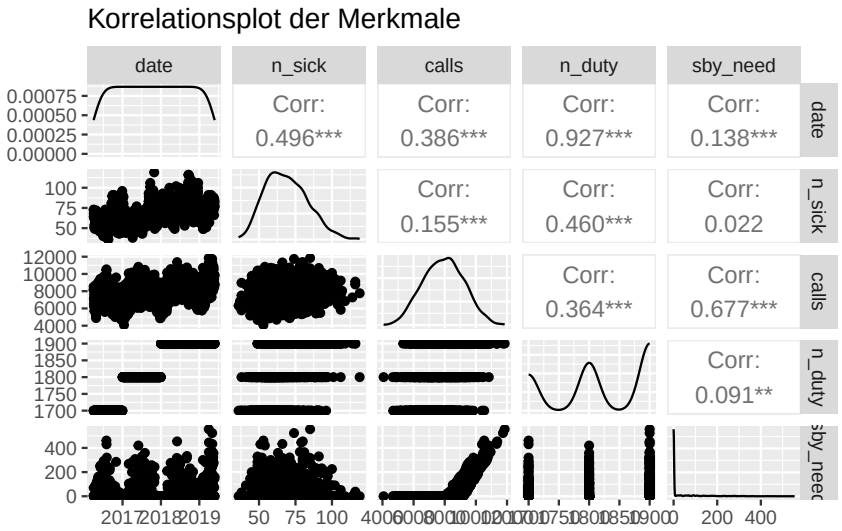
Die Autokorrelation der Zielvariable (s.u.) zeigt, neben der erwarteten monatlichen Korrelation in den *lags* 28 bis 33, auch wöchentliche Peeks Die rasche Abnahme der Korrelation zu Beginn des Plots lässt auf einen nur schwach ausgeprägten Trend in den Werten schließen, vmtl. da dieser augenscheinlich erst mit 2019 auftritt. Ab *lag* 63 zeigt die Autokorrelation teils negative Werte und wird daher mehr oder weniger unbrauchbar.

```
ts_sby %>%
  ACF(y = sby_need, lag_max = 93) %>%
  autoplot() +
  labs(title = "Autokorrelation von sby_need")
```



Übersicht über die weiteren Datenmerkmale

```
ts_sby %>%
  GGally::ggpairs(title = "Korrelationsplot der Merkmale")
```



sby_need (die Zielvariable) korreliert augenscheinlich am meisten mit der Anzahl der Notfallanrufe (*calls*). Hier liegt, ab einer gewissen Anzahl an *calls* nahezu eine lineare Korrelation vor. Das klingt durchaus plausibel. Die vermutete Saisonalität ist in der Übersicht der Variablen *sby_need*, *calls* und *n_sick* (der Anzahl der krankgemeldeten Bereitschaftsfahrer:innen), mit dem menschlichen Auge, gut zu erkennen. Bei dem Merkmal *calls* wirkt diese jedoch am stabilsten und auch die Wertevereilung scheint hier am besten einer Normalverteilung zu folgen.

Das Merkmal *n_duty* (die Anzahl der diensthabenden Bereitschaftsfahrer:innen) weist nur drei unterschiedliche Werte auf, die ausschließlich eine Abhängigkeit zum Datum haben. Hierbei wurde die Anzahl der Diensthabenden jedes Jahr zum ersten Januar um 100 Personen erhöht. Am 01.01.2019 ist dies jedoch nicht geschehen. Dieser Umstand muss vermutlich gesondert mit den Entscheidungsträgern besprochen und ggf. nachgezogen werden.

n_sick zeigt zwar die angesprochene Saisonalität, weist, aufgrund der o.a. Grafik, aber nur eine geringe Korrelation zur Zielvariablen auf. Es scheint plausibel, dass eine höhere Anzahl an Krankenständen, bei gleichbleibenden oder steigenden Notfällen zu einem höheren Bedarf an Bereitschaftspersonal führt. Möglicherweise unterstützt die Kombination dieser beiden Merkmale eine erfolgsversprechende Vorhersage.

Für die Vorhersage des Merkmals *sby_need* gibt es zu diesem Zeitpunkt daher fünf mögliche Ansatzpunkte.

1. Die direkte Vorhersage aufgrund der eigenen Saisonalität
2. Eine indirekte Vorhersage aufgrund der Saisonalität des Merkmals *calls* und der, ab einem gewissen Wert, nahezu linear anzunehmenden Korrelation mit *sby_need*. Letztere zeigt jedoch “drei Liniaritäten” und ist wahrscheinlich durch *n_duty* beeinflusst.
3. Wie in Punkt 2., nur dass zusätzlich eine jährliche Steigerung von *n_duty* berücksichtigt wird um aufgetretene Trends abzuflachen und mehr Fokus auf Saisonalität legen zu können. Dieses zu generierende Merkmal wird als regulierte calls, *reg_calls* bezeichnet.
4. Ein indirekte Vorhersage aufgrund der Saisonaltität eines neuen kombinierten Merkmals aus *calls* und *n_sick*, die jedoch zuerst zu evaluieren ist. Das neue Merkmal wird *calls_sick* genannt.
5. Wie in Punkt 4., aber wiederum unter Berücksichtigung einer jährlichen Steigerung von *n_duty* um aufgetretene Trends abzuflachen. Als Bezeichnung wird *reg_calls_sick* verwendet.

Die weitere Analyse konzentriert sich daher auf die ursprünglichen Merkmale *date*, *sby_need* und *calls* sowie auf die neu generierten Variablen.

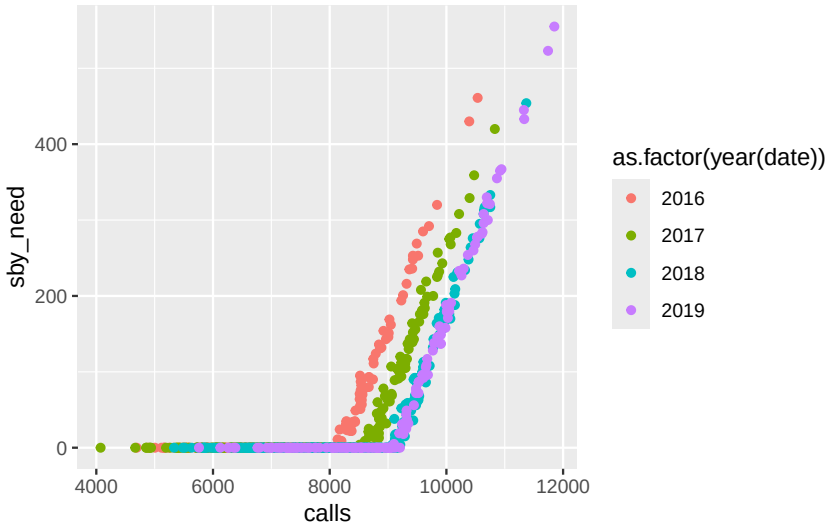
Merkmalsgenerierung

Die in Punkt 4. beschriebene kombinierte Variable *calls_sick* aus *calls* und *n_sick* wird als Anrufe je krankgemeldetem/r Einsatzfahr:in verstanden und daher berechnet als

$$calls\_sick_t = \frac{calls_t}{n\_sick_t}$$

Wie der u.a. Korrelationsplot von *calls* und *sby_need* zeigt, hatten die Erhöhungen des diensthabenden Personals Auswirkungen auf die Notwendigkeit Bereitschaftspersonal zu aktivieren. Die Tatsache, dass 2018 und 2019 genauso wie bei dem Merkmal *n_duty* eine gleiche Korrelation vorliegt, bestätigt die Annahme der Abhängigkeit zu *n_duty* vorliegt.

```
ggplot(ts_sby,
            aes(x = calls,
                y = sby_need,
                colour = as.factor(year(date))
                )) +
  geom_point()
```



Die Abstände im augenscheinlichen "Intercept" betragen zwischen den Jahren jeweils etwa 500 (2016: ~8.000 *calls*, 2017: ~8.500 *calls*, 2018 und 2019: ~9.000 *calls*). Das ist jeweils das fünffache des jährlichen Anstiegs von *n_duty*.

Wurden weniger Anrufe als der jeweilige "Intercept" getätigt wurde auch kein Bereitschaftspersonal aktiviert. Sollte ein Ansatz gewählt werden, in dem *calls* oder *reg_calls* für die Vorhersage verwendet wird, wird dieser Intercept, vmtl. in einem linearen Modell zu bedenken sein.

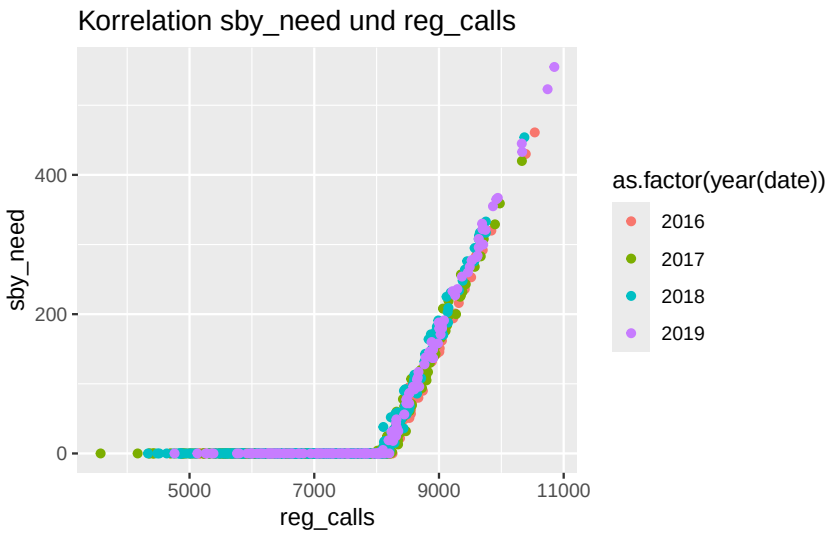
Die oben festgestellte Auswirkung der Erhöhung mit einem Faktor 5 wird in der Merkmalsgenerierung berücksichtigt.

$$reg\_calls_t = calls_t - n\_duty_t \mid n\_duty_t = (n\_duty_{year} - 1700) * 5$$

Das neue Merkmal *reg_calls_sick* leitet sich davon ab.

$$reg\_calls\_sick_t = \frac{reg\_calls_t}{n\_sick_t}$$

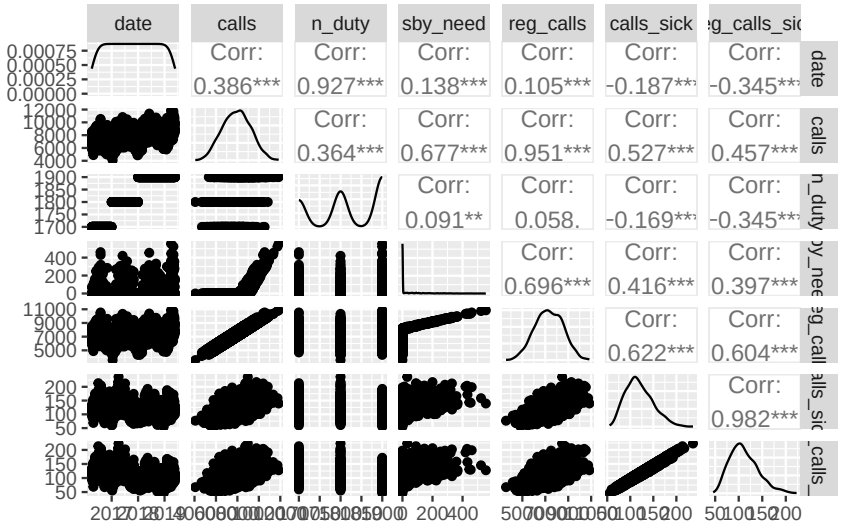
```
ts_sby <- ts_sby %>%
  mutate(reg_calls = calls - (n_duty - 1700) * 5,
         calls_sick = calls/n_sick,
         reg_calls_sick = reg_calls/n_sick) %>%
  select(-c("n_sick"))

ggplot(ts_sby,
            aes(x = reg_calls,
                y = sby_need,
                colour = as.factor(year(date))
                )) +
  geom_point() +
  labs(title = "Korrelation sby_need und reg_calls")
```



Die Auswirkung der vorgenommenen Anpassungen an *reg_calls* sind in der Grafik gut erkennbar.Der mittlere Intercept liegt bei etwa 8150 Anrufen. Bei weniger Anrufen an einem Tag wird kein zusätzliches Bereitschaftspersonal benötigt.

Wie die nachgestellte Grafik zeigt konnte die Korrelation von *reg_calls* mit *sby_need* gegenüber *calls* leicht gesteigert werden. Bei den neuen Merkmalen *calls_sick* und *reg_calls_sick* ist diese jedoch vergleichsweise nicht ausreichend gegeben.

```
ts_sby %>%
  GGally::ggpairs()
```



Um festzustellen welcher Ansatz möglicherweise erfolgversprechender ist, lohnt es sich die Variablen auf die vermeintliche Vorhersagbarkeit ihrer Saisonalität hin zu überprüfen. Hyndman und Athanasopoulos (2021) verweist hierbei auf die spektrale Entropy (Shannon) einer Zeitreihe, die einen Wert zwischen 0 und 1 ausgibt. Je niedriger der Wert, desto stärker ist die Saisonalität und der Trend der Zeitreihe.

```
ts_sby %>%
  select(-c("n_duty")) %>%
  features_all(feat_spectral)
```

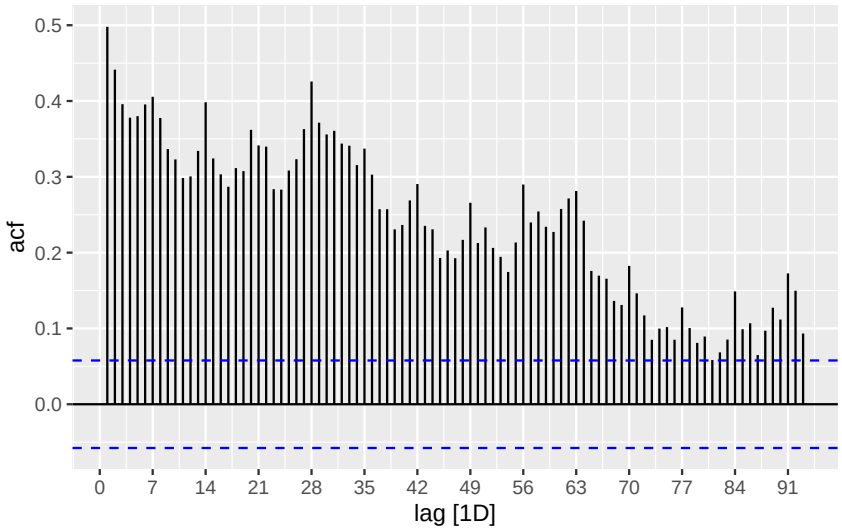
```
# A tibble: 1 x 5
  calls_spectral_entropy sby_need_spectral_entropy reg_calls_spectral_entropy
                   <dbl>                     <dbl>                      <dbl>
1                  0.812                     0.945                      0.880
# i 2 more variables: calls_sick_spectral_entropy <dbl>,
#   reg_calls_sick_spectral_entropy <dbl>
```

Interessanterweise, ist trotz einer augenscheinlich gut erkennbaren Saisonalität der Entropie-Wert durchgehend relativ hoch. Am ehesten scheint sich an dieser Stelle das unregulierte Merkmal der Notfallanrufe (*calls*) für eine saisonal-bedingte Vorhersage zu eignen. Dieses Merkmal scheint annähernd normalverteilt zu sein und die Korrelation zur Zielvariablen *sby_need* liegt nur leicht unter dem regulierten Wert *reg_calls*. *calls_sick* und *reg_calls_sick* schneiden bei der Betrachtung der letzten beiden Faktoren deutlich schlechter ab.

Untersuchung des Merkmals “calls”

Der Autokorrelationsplot der Variablen *calls* soll weiteren Aufschluss über dieses Merkmal geben. Hierfür werden 93 lags, also etwa drei Monate betrachtet.

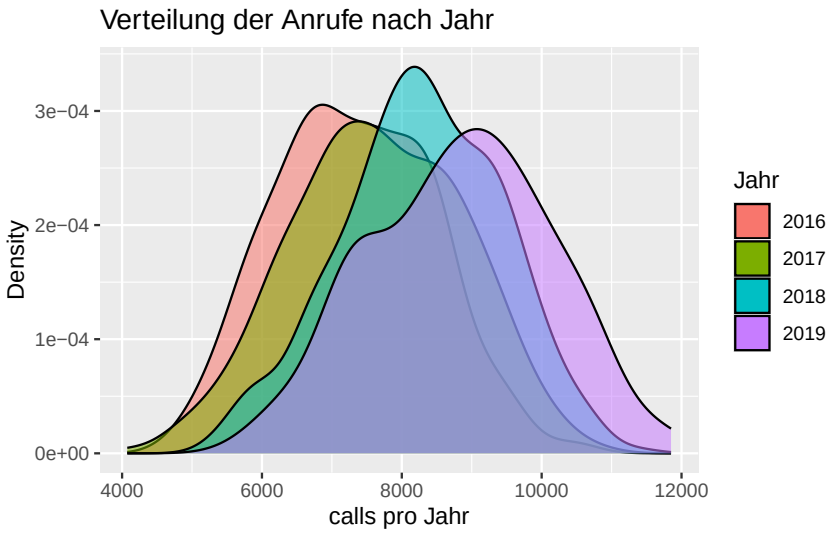
```
ts_sby %>%
  ACF(y = calls, lag_max = 93) %>%
  autoplot()
```



Die höhere Autokorrelation in den niedrigen “lags”, die im Verlauf abnimmt, zeigt, dass die Daten einem Trend unterliegen. Die “kleinen” Peaks bei 7, 14, 21, etc. zeigen eine leichte wochenweise Saisonalität, der etwas höhere Peak bei lag 28 weist auf eine ähnliche, aber augeschneinlich stabilere, monatliche Saisonalität wie die Zielvariable hin. Auch hier nimmt der Autokorrelationswert ab lag 63 schneller ab, rutscht jedoch nicht ins Negative sondern zeigt weiters verwendbare Werte.

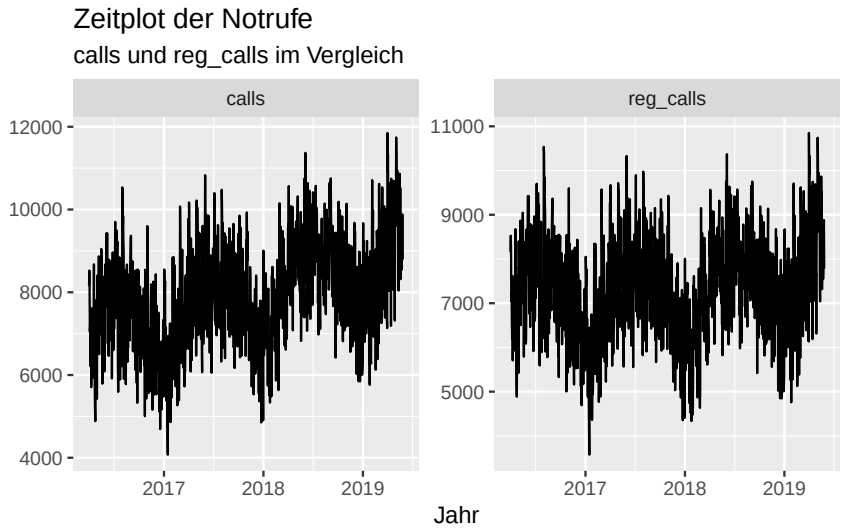
Die Verteilung des möglichen Prädiktors gibt Auschluss, bzw. einen Überblick über die zu erwartenden Werte.

```r
ggplot() +
  geom_density(data = ts_sby, aes(x = calls, fill = as.factor(year(date)), alpha = 0.3),
                   show.legend = c(fill = TRUE, alpha = FALSE)) +
  labs(title = "Verteilung der Anrufe nach Jahr",
       y = "Density",
       x = "calls pro Jahr",
       fill = "Jahr")
```



Wie auch bereits im Autokorrelationsplot ist hier der Trend über die Jahre in der Verschiebung der Kurven nach rechts gut erkennbar. Die Verteilung des Merkmals nähert sich einer Normverteilung deutlich besser an als die Zielvariable (von den kleinen Seitenhügeln einmal abgesehen).

```r
ts_sby %>%
  autoplot(vars(calls, reg_calls)) +
  labs(x = "Jahr",
       title = "Zeitplot der Notrufe",
       subtitle = "calls und reg_calls im Vergleich")
```

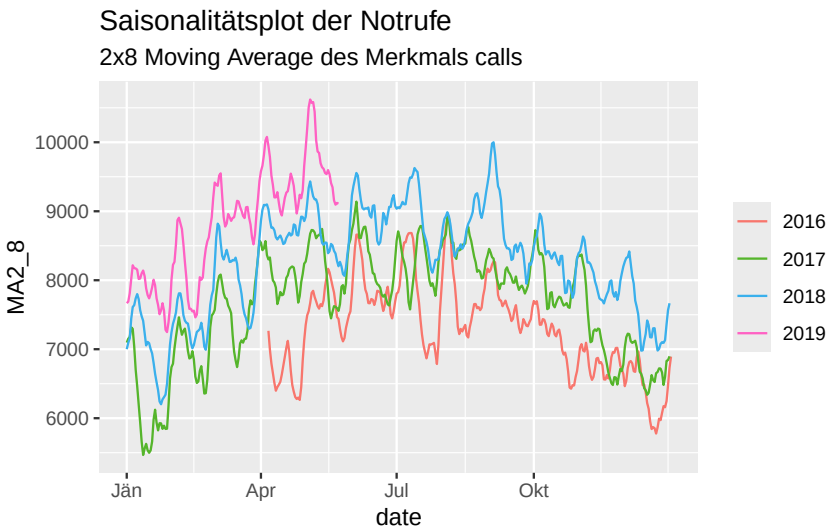


Im Vergleich der Variablen *calls* und *reg_calls* ist die Modifikation der letzteren gut an dem fehlenden Trend zu erkennen. Wie auch bereits die spektrale Entropy zeigte, ist das unregulierte Merkmal vermutlich leichter vorherzusagen. Schließlich ist der Trend ein wesentliches Element eines Forcasting. Gleichzeitig muss die Regularisierung in Bezug auf das diensthabende Personal (*n_duty*) in der Korrelation mit der Zielvariablen berücksichtigt werden, da dies wiederum Einfluss auf den Bedarf an Bereitschaftsfahrenden hat. Andernfalls wäre es falsch, bereits jetzt rechnerisch das diensthabende Personal der zukünftigen Zeiträume in einem Vorhersagemodell festzulegen. Vielmehr wäre es sinnvoll das flexibel zu gestalten. Über diese Punkte wird jedenfalls in der Modelerstellung nachgedacht werden müssen.

Auch wenn in den oberen beiden Plots eine jährliche und monatliche Saisonalität gut erkennbar ist, zeigt der Verlauf dennoch eine hohe Volatilität (vmlt. durch die leichte wochenweise Autokorrelation, wie weiter oben beschrieben). Ein 2x8 Moving Average würde die wöchentliche Saisonalität glätten, ohne die monatliche Saisonalität zu sehr zu beeinflussen. Dazu wird das Merkmal *calls* nochmals modifiziert.

```
ts_sby <- ts_sby %>%
  mutate(
    MA8 = slider::slide_dbl(calls, mean,
                            .before = 3, .after = 4,
                            .complete = TRUE),
    MA2_8 = slider::slide_dbl(MA8, mean,
                              .before = 1, .after = 0,
                              .complete = TRUE)
       )

ts_sby %>%
  gg_season(MA2_8, period = "1y") +
  labs(title = "Saisonalitätsplot der Notrufe",
       subtitle = "2x8 Moving Average des Merkmals calls")
```



Das Ergebnis zeigt, wie erwartet, die monatliche und jährliche Saisonalität im direkten Vergleich der Jahre. Auch hier ist der Trend wiederum gut erkennbar.

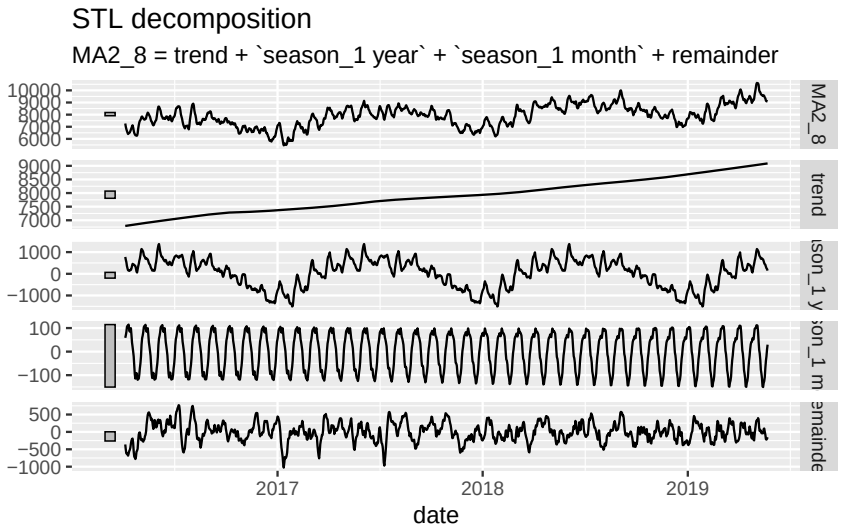
```
ts_sby %>%
  features(MA2_8,feat_spectral)
```

```
# A tibble: 1 x 1
  spectral_entropy
             <dbl>
1            0.494
```

Auch die spektrale Entropy des Merkmals wird durch den gleitenden Durchschnitt deutlich verbessert. Wie sehr darunter die Tauglichkeit als Prädiktor leidet, wird in der Modellerstellung zu evaluieren sein.

Decomposition von calls

```
ts_sby %>%
  filter(!is.na(MA2_8)) %>%
  model(
    STL(MA2_8 ~
          trend(window = 365) +
          season(period = "1 year", window = 540) +
          season(period = "1 month", window = 61))) %>%
  components() %>%
  autoplot()
```



Mit Hilfe der Methode “Seasonal and Trend decomposition using Loess” (STL), zeigt eine erste Dekomposition des Merkmals die besprochenen Komponenten. Der Trend ist nahezu linear, und die monatliche Saisonalität regelmäßig. In der Jährlichen Saisonalität scheint noch ein modifizierender Teil der Monatsperioden enthalten zu sein. Augenscheinlich könnte auch im *remainder*, dem Restanteil, noch eine Periodizität herauszuarbeiten sein. Dies wird jedoch ggf. die Aufgabe einer Modellerstellung.

Fazit

Der in der Merkmalsübersicht angesprochene zweite Ansatz zur Vorhersage von *sby_need* scheint aufgrund der explorativen Analyse am vielversprechensten. Bei diesem Ansatz wurde eine indirekte Vorhersage aufgrund der Saisonalität des Merkmals *calls* und der, ab einem gewissen Wert, nahezu linear anzunehmenden Korrelation mit *sby_need* untersucht, wobei letztere bewiesen durch *n_duty* beeinflusst ist. Die Qualität dieses Ansatzes wird in der Modellerstellung geprüft.

Für ein Benchmarking mit einem einfachen Baselinemodel kann möglicherweise auf Basis der Zielvariablen eine brauchbare Lösung gefunden werden,

Speichern aufbereiteter Daten

```
save(ts_sby, file = "../00_sample_data/02_processed/data_explorative.rda")
```

Literaturverzeichnis

Hyndman, Rob J., und George Athanasopoulos. 2021. *Forecasting: Principles and Practice*. 3. Auflage. https://otexts.com/fpp3/accuracy.html.
Kuhn, Max. 2019. *The caret Package*. https://topepo.github.io/caret/.

A.3 Baseline Model

Einleitung

Dieses Dokument beschreibt die Erarbeitung eines Baseline- oder Benchmarkmodel. Diese Modell dient der Anwendung einer simplen Lernmethode auf die gegebene Aufgabenstellung. Die Ergebnisse dieses Modells werden in weiterer Folge zur Qualitätsmessung und Evaluierung von ausgefeilteren Modells verwendet.

(**hyndman_forecasting_2021?**) schlägt für die Erstellung von Benchmarks im Zeitreihenbereich die Algorithmen

- **Mean**: Der Mittelwert der Zeitreihe
- **Naïve**: Der Wert der letzten Observation
- **Seasonal Naïve**: Der Wert der letzten gleichen Saison zB das gleiche Monat des Vorjahres, oder der gleiche Tag des Vormonats (abhängig von der Saisonalität)
- **Drift**: Eine Variation von Naïve, die die durchschnittliche Veränderungsrate der Daten berücksichtigt und somit auch steigen oder fallen kann

Im R-package *fpp3* ist außerdem auch ein **TSLM** Algorithmus enthalten, der ein Lineares Modell mit Trend- und saisonalen Komponenten auf die Daten anwendet. Dieser Algorithmus könnte sich in diesem Anwenungsfall auch gut für ein Baselinemodel eignen.

Nachdem die Modelle wenig Rechenkapazität benötigen, bietet es sich an, einfach alle Modelle zu trainieren und anschließend die, für diese Aufgabe gültige Benchmark zu wählen. Außerdem gibt die aktuelle Vorgehensweise, nämlich ein konstantes Vorhalten von 90 Bereitschaftsfahrenden ebenso bereits eine zu berücksichtigende Messvariable vor, schließlich sollen ja langfristig Bereitschaftskosten durch die Anwendung eines Modell gespart werden können.

Ergebnisse der explorativen Datenanalyse

Die Erkenntnisse der explorativen Datenanalyse liefern eine Grundlage für die Erstellung eines Baseline Models und lassen sich in Bezug auf die Zielvariable wie folgt zusammenfassen:

1. Die vermutete Saisonalität ist augenscheinlich gegeben, der rasche Wechsel zwischen sehr hohem und fast keinem Bedarf an Bereitschaftspersonal könnten eine Vorhersage erschweren. Ab 2019 ändert sich das saisonale Verhalten der Daten und auch ein durchschnittlicher Anstieg in den Werten der Zielvariablen ist zu beobachten.
2. Eine lineare Annäherung in der Ansicht, die sogenannte Smooth-Curve, zeigt, dass die Spitzenwerte auch mit einem 99% Konfidenzinterwall nicht abgedeckt werden.
3. Ein Mindestmaß an etwa 35 Personen in Bereitschaft sinnvoll ist.
4. Ebenso wurde eine jährliche und (instabile) monatliche Saisonalität mit schwacher ausgeprägten wöchentlichen Perioden erkannt. Die Autokorrelation zeigt, dass eine Vorhersage über einen längeren Zeitraum nicht zielführend ist. Das wird vmtl. über die Änderungen im Muster ab Beginn 2019 erklärt werden können.
5. Es wurde außerdem festgehalten, dass eine nahezu lineare Abhängigkeit des Bedarfs an StandBy-Personal zu der Anzahl eingehender Notrufe besteht, diese jedoch auch vom diensthabenden Personal beeinflusst wird.

Am erfolgversprechensten wurde eine indirekte Vorhersage aufgrund der Saisonalität des Merkmals *calls* und der, ab einem gewissen Wert, nahezu linear anzunehmenden Korrelation mit *sby_need* beurteilt, wobei Letztere durch *n_duty* beeinflusst ist.

Für ein Baselinemodel scheint der obige Ansatz jedoch zu aufwendig. Deshalb wird dieses direkt auf die Zielvariable trainiert. Um den oben genannten Herausforderungen in der Vorhersage zu begegnen, wird jedoch noch eine weitere Vorverarbeitung der Daten nötig sein.

Daten laden

```
load(file = "../../00_sample_data/02_processed/data_explorative.rda")
```

Aufsplitten der Daten in Trianings- und Testdaten

Gemäß Aufgabe soll der Vorhersagezeitraum etwa 2 Monate betragen. Der Zeitraum, der durch das Datenmaterial abgedeckt wird, umfasst etwa drei Jahre (36 Monate). Üblliche Aufteilungen von Datensätzen in Trainings- und Testdaten, sehen ein Verhältnis von 70:30 oder 80:20 Prozent vor (je nach Größe des Datensatzes), der Umfang des gewünschten Vorhersagezeitraums sollte aber jedenfalls gegeben sein. Ein Verhältnis von 80:20 entspricht bei den vorliegenden Daten etwa 29 Monate:7 Monate. Der Vorhersagezeitraum wäre damit abgedeckt. Die, in der explorativen Datenanalyse festgestellte Änderung des saisonalen Verhaltens mit dem Jahreswechsel auf 2019 sowie ein deutlich schlechterer Autokorrelationswert ab lag 63 müssen in den Trainingsdaten umbedingt berücksichtigt werden. Daher wird der Testtdatensatz auf 2 Monate reduziert.

Die Anzahl der Variablen wird gleichhzeitig ebenfalls auf ein benötigtes Mindestmaß beschränkt.

```
ts_sby_train <- ts_sby %>%
  select(date, sby_need) %>%
  slice(0:(n() - 62))


ts_sby_test <- ts_sby %>%
  select(date, sby_need) %>%
  slice((n() - 61):n())
```

Modelerstellung

Erster Versuch

Der erste Versuch der Modellerstellung wendet die beschriebenen Benchmarkmodelle, trotz der erkannten Herausforderungen, direkt auf die Zielvariable an, um zu sehen, wie sich die Berechnungen verhalten. Ein brauchbares Ergebnis wird an dieser Stelle nicht erwartet. Um die monatlichen saisonalen Muster einzufangen, aber die Änderungen zu Beginn des Jahres 2019 hervorzuheben, wird der Trainingszeitraum weiter auf ein Jahr beschränkt.

```
# Training
t <- system.time(
  progressr::with_progress(
    sby_basemodel <- ts_sby_train %>%
      filter(date >= "2018-03-26") %>%
      model(
```

```
        mean = MEAN(sby_need),
        naive = NAIVE(sby_need),
        snaive = SNAIVE(sby_need ~ lag(63), drift = TRUE),
        drift = RW(sby_need ~ drift()),
        tslm = TSLM(sby_need ~ trend() + season()),
      )
  ),  gcFirst = TRUE
)

print(paste0(" Die Benötigte Zeit für das Training betrug ", round(t[3],2), " Sekunden."))
```

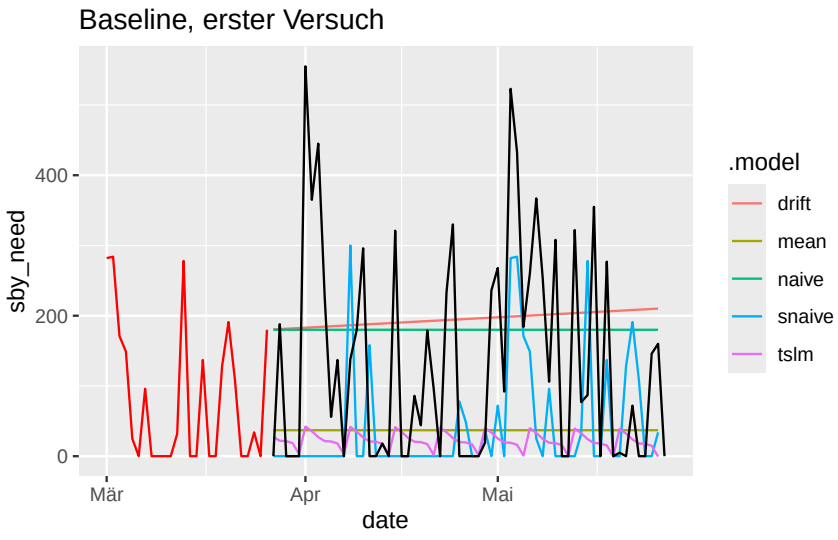
```
[1] " Die Benötigte Zeit für das Training betrug 0.07 Sekunden."
```

```
# Vorhersage
t <- system.time(
  progressr::with_progress(
    sby_basefc <- sby_basemodel %>%
      forecast(h = "2 months", level = c(80))
  ), gcFirst = TRUE
)

print(paste0(" Die Benötigte Zeit für die Vorhersage war ", round(t[3],2), " Sekunden."))
```

```
[1] " Die Benötigte Zeit für die Vorhersage war 0.13 Sekunden."
```

```
sby_basefc %>%
  autoplot(ts_sby_test,
           level = NULL) +
  autolayer(ts_sby_train %>% filter(date >= "2019-03-01"), colour = "red" ) +
  labs(y = "sby_need",
       title = "Baseline, erster Versuch")
```



In dem o.a. Plot sind die Trainingsdaten Rot und die Testdaten Schwarz dargestellt. Die Vorhersagen durch die einzelnen Algorithmen sind farblich in der Legende kodiert. Wenig überraschend ist die Vorhersage nicht brauchbar. Der *Drift* zeigt einen erwartbaren positiven Trend und *Naïve* eben nur den letzten Wert (in diesem Fall 180). Der *Mean* liegt bei ca. 37 und damit etwa dort, wo gem. explorativer Datenanalyse ein Mindestwert angesetzt werden sollte.

TSLM erkennt wochenweise Schwankungen (wie auch im Autokorrelationsplot beobachtet), setzt diese allerdings viel zu gleichmäßig und vor allem zu niedrig an. *Snaïve* kann offenbar den rapiden Wechsel zwischen 0 und auftretenden Spitzen modellieren. Setzt diese Spitzen allerdings zu niedrig an und, viel fataler, nicht dort wo sie tatsächlich auftreten.

Positiv hervorzuheben ist die benötigte Zeit für Training und Vorhersage. Diese betrug hier nur wenige hunderstel Sekunden. Der dazu verwendete Rechner war ein älteres Notebook vom Typ: Dell Latitude 5490, Intel i7 vPro (8th Gerneration) mit 16 Gb RAM.

Die Berechnung der Modelle auf neueren Systemen sollte die Hardware demnach nicht vor große Herausforderungen stellen.

Zweiter Versuch mit modifizierten Daten

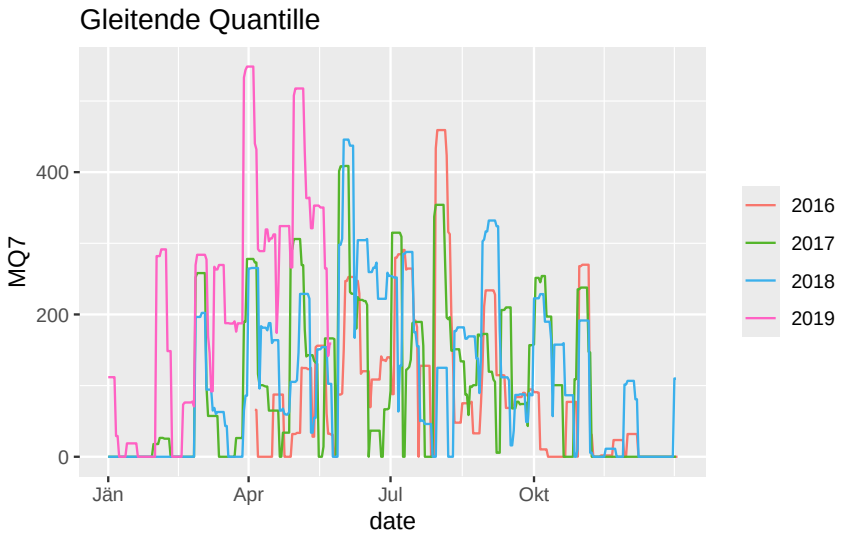
Der Vorhersageplot des ersten Verusches zeigt gut dessen Scheitern. Die hohen Ausschläge zu Beginn jedes Monats werden nicht durch die Modelle abgedeckt, die Peaks sind aber gleichzeitig auch zu unregeläßig um zB von einem *Snaïve*-Model “getroffen” zu werden. *TSLM* lässt schönt die wöchentliche Komponente in den Daten erkennen, die aber eigentlich nur schwach ausgeprägt ist.

Die Idee für den zweiten Versuch eines Benchmarkmodels ist es, die rapiden Wechsel zwischen Spitzen und 0-Linie zu entfernen, die ensprechende Höhe des möglichen Bedarfes abzudecken und durch Aggregation die “Treffsicherheit” für auftretenden Bedarf zu erhöhen. Auf die intra-wöchentlichen Schwankungen wird keine Rücksicht mehr genommen.

Dazu bietet sich, wie auch in der explorativen Datenanalyse, die Verwendung eines gleitenden Durchschnitts an, um den Wochenbedarf zu glätten. Das Problem der Durchschnittsbildung ist jedoch, dass die Spitzen “per design” nie richtig vorhergesagt werden können. Als gleitendes Maß wird in diesem Fall daher die 99% Quantile mit einem Fenster von sieben Tagen getestet.

```
ts_sby <- ts_sby %>%
  mutate(
    MQ7 = slider::slide_dbl(sby_need,
                            ~ quantile(x = .x,
                                       probs = 0.99),
                            .before = 3, .after = 3,
                            .complete = TRUE))

ts_sby %>%
  gg_season(MQ7, period = "1y") +
    labs(title = "Gleitende Quantille")
```



Im saisonalen Plot sieht man, dass die Spitzenwerte, durch die Glättung, deutlich besser übereinander liegen und auch die monatliche Komponente der Saisonalität ist gut erkennbar. Der u.a. Wert der spektralen Entropy hat sich gegenüber jenem der Zielvariablen, in der explorativen Datenanalyse merkbar verbessert.

```
ts_sby %>%
  features(MQ7,feat_spectral)
```

```
# A tibble: 1 x 1
  spectral_entropy
             <dbl>
1            0.695
```

Die Treffsicherheit des Baselinemodells soll weiters durch eine wochenweise Aggregation erhöht werden. Das bedeutet in der Praxis, dass jeweils für eine ganze Woche die selbe Anzahl an Bereitschaftspersonal vorzuhalten wäre, da nicht genau bestimmt werden kann, wann genau der Spitzenbedarf auftreten wird. Nach der Aggregation werden ein neuer Trainings und Testdatensatz gebildet und die Modelle trainiert.

```
```{r}
#| message: false
#| warning: false

# Aggregation der täglichen Daten
ts_sby_week <- ts_sby %>%
  filter(!is.na(MQ7)) %>%
  index_by(week = ~ yearweek(.)) %>%
  summarise(w_need = max(MQ7)) %>%
  filter(!is.na(w_need))

# Datenaufteilung (61 Tage werden zu 9 Wochen)
ts_sby_train <- ts_sby_week %>%
  select(week, w_need) %>%
```
```

```
  slice(0:(n() - 9))

ts_sby_test <- ts_sby_week %>%
  select(week, w_need) %>%
  slice((n() - 9):n())
```

```
# Training
t <- system.time(
  progressr::with_progress(
    sby_basemodel <- ts_sby_train %>%
      filter(year(week) >= 2017) %>%
      model(
        mean = MEAN(w_need),
        naive = NAIVE(w_need),
        snaive = SNAIVE(w_need ~ lag(52), drift = TRUE),
        drift = RW(w_need ~ drift()),
        tslm = TSLM(w_need ~ trend() + season()),
      )
  ),  gcFirst = TRUE
)

print(paste0(" Die Benötigte Zeit für das Training betrug ", round(t[3],2), " Sekunden."))
```

```
[1] " Die Benötigte Zeit für das Training betrug 0.06 Sekunden."
```

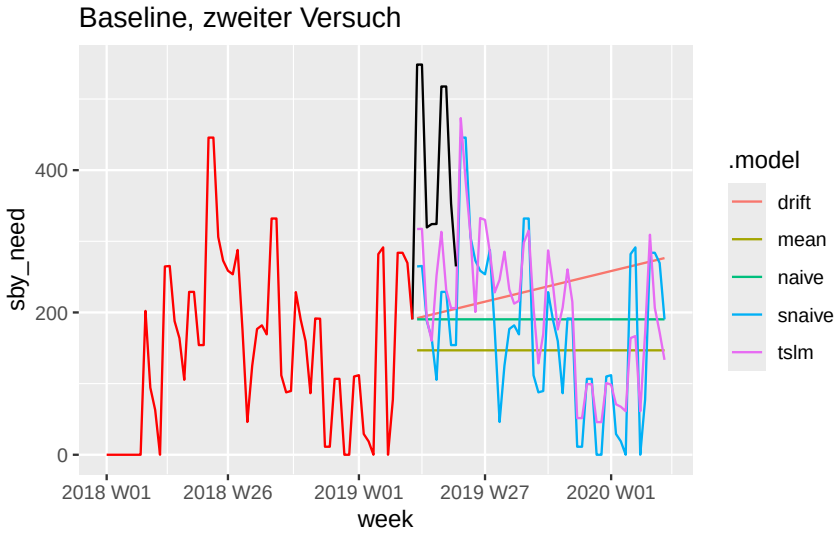
```
# Vorhersage
t <- system.time(
  progressr::with_progress(
    sby_basefc <- sby_basemodel %>%
      forecast(h = "12 months", level = c(99))
  ), gcFirst = TRUE
)

print(paste0(" Die Benötigte Zeit für die Vorhersage war ", round(t[3],2), " Sekunden."))
```

```
[1] " Die Benötigte Zeit für die Vorhersage war 0.22 Sekunden."
```

```
# Visualisierung der Vorhersage

sby_basefc %>%
  autoplot(ts_sby_test,
           level = NULL) +
  autolayer(ts_sby_train %>% filter(year(week) >= 2018), colour = "red" ) +
  labs(y = "sby_need",
       title = "Baseline, zweiter Versuch")
```



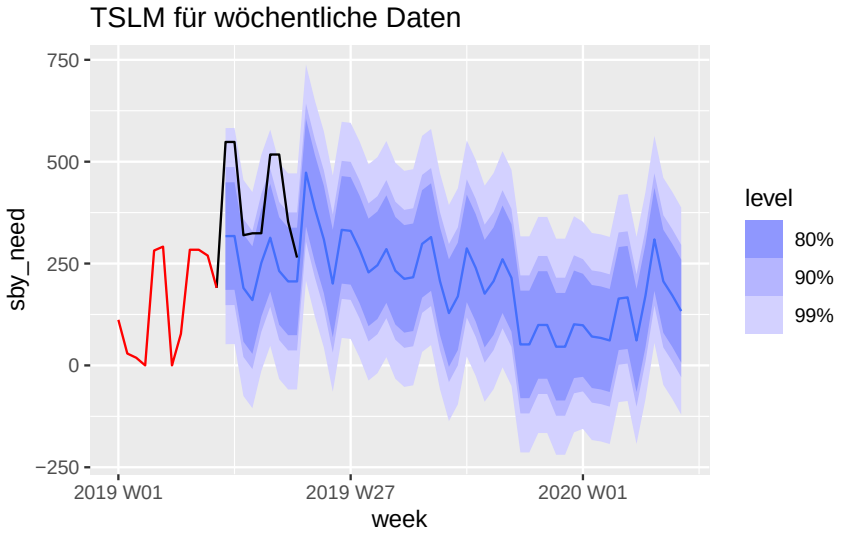
Nachdem im Saisonalitätsplot ein Aufwärtstrend klar erkennbar ist, wurde der Trainingszeitraum auf zwei etwa zwei Jahre und der Vorhersagezeitraum auf ein Jahr erweitert. Die Berechnungszeit für das Training und die Vorhersage kann auch bei diesem Ansatz vernachlässigt werden.

Der o.a. Plot folgt dem Farbschema des vorigen Vorhersageplots. Die unterschiedlichen Qualitäten der Vorhersagen sind gut erkennbar. *Drift* und *Naïve* können auch hier wieder verworfen werden. *Mean* gibt einen sehr hohen Durchschnitt aus. *Snaïve* bildet zwar die Saisonalität ab, lässt aber den Trend unbeachtet.

TSLM scheint, im aggregierten Ansatz, für ein Baselinemodel zu funktionieren. Die Vorhersage über den Zeitraum von einem Jahr zeigt, dass das Modell sowohl den Trend als auch die Saisonalität aufnimmt. Der erhöhte Bedarf zu Beginn jedes Monats ist deutlich herausgearbeitet. Die, sich mit den Traininsdaten überschneidenden, Vorhersagewerte sind zu niedrig, was durch den plötzlich störkeren Aufwärtstrend erklärbar ist, das Model scheint sich aber im weiteren Verlauf einer möglichen Realität anzunähern. Was bei dieser Variante nicht wiederzufinden ist, sind die Ruhephasen. Im Gegensatz zu aktuellen Vorgehensweise wird bei diesem Modell, außer in den Wintermonaten, sonst nie mit 90 Personen im Bereitschaftsdienst das Auslangen gefunden werden. Das bedeutet, für das Deutsche Rote Kreuz Berlin wird der Bereitschaftsdienst damit in Summe vermutlich teuerer, jedoch besser planbar und es müssen weniger Bereitschaftsfahrer zusätzlich aktiviert werden.

TSLM wird als Baselinemodel für diesen UseCase akzeptiert und u.a. noch einmal im Fokus mit 80%, 90% und 99% Konfidenzintervall darstellt. Das 99% Intervall hätte, im Vergleich mit den Testdaten, auch die hohen Bedarfsspitzen ausreichend prognostiziert, allerdings würden mit diesem Konfidenzintervall in Monaten mit weniger Bedarf trotzdem über 250 Bereitschaftsfahrer:innen vorgesehen werden. Zumindest beim Baseline-Model werden hier Abstriche in den Anforderungen gemacht werden müssen.

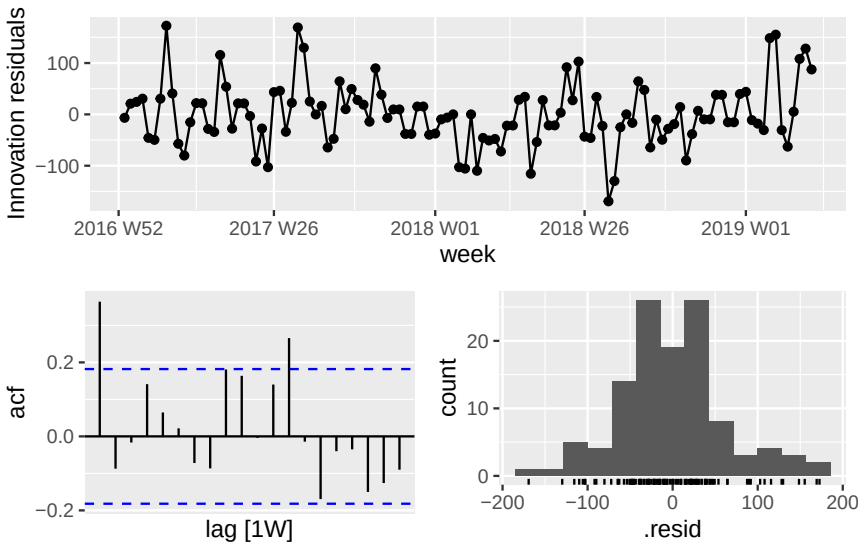
```
sby_basefc %>%
  filter(.model == "tslm") %>%
  autoplot(ts_sby_test,
           level = c(80, 90, 99)) +
  autolayer(ts_sby_train %>% filter(year(week) >= 2019), colour = "red" ) +
  labs(y = "sby_need",
       title = "TSLM für wöchentliche Daten")
```



Evaluierung

Der u. a. angeführte Residualsplot zeigt ein nahezu optimales Verhalten des Algorithmus. In den Residuals ist auf den ersten Blick kein deutliches “Rest-Pattern” in den Daten zu beobachten und genauso wenig im Autokorrelationswert. Die Residuals wären im besten Fall mit einem arithmetischen Mittel von 0 normal verteilt. Das trifft nicht ganz zu, die Annäherung an das gewünschte Verhalten ist aber gut erkennbar. Dieser Plot zeigt allerdings nur, wie gut die Realität mit dem erstellten Modell (den sogenannten *fitted values*) erklärt werden kann. Es ist keine Bewertung der Vorhersagefähigkeit.

```
mod <- select(sby_basemodel, tslm)
gg_tsresiduals(mod)
```

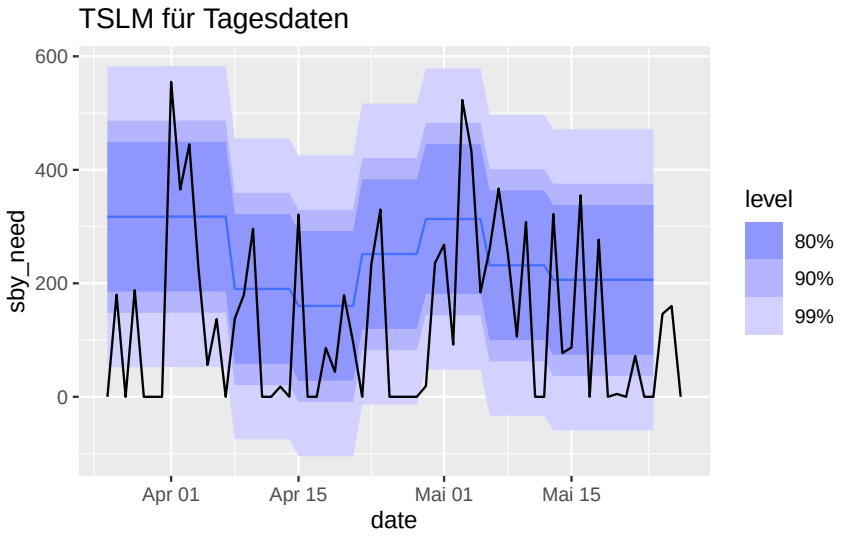


Für diesen Fall werden nachfolgend einige Bewertungsmethoden durchgeführt, die die Güte des *TSLM* belegen sollen. Die Anforderungen des Use Case sehen eine tageweise Vorhersage vor, das Modell berechnet allerdings wöchentliche Daten. Auch für die Vergleichbarkeit mit zukünftigen Modellen ist es sinnvoll die aggregierten Daten wieder auf Tagesdaten aufzuteilen. Beim *TSLM* resultiert das natürlich darin, dass innerhalb einer Woche alle Tage den gleichen Wert haben.

```
week2day <- ts_sby %>%
  filter(!is.na(MQ7)) %>%
  mutate(week = yearweek(date)) %>%
  select(week, date)

fc <- sby_basefc %>%
  as_tibble() %>%
  left_join(week2day, by = "week") %>%
  filter(!is.na(date)) %>%
  mutate(sby_need = w_need) %>%
  select(.model, date, sby_need, .mean) %>%
  as_fable(index = date, key = .model, response = "sby_need", distribution = sby_need)

fc %>%
  filter(.model == "tslm") %>%
  autoplot(filter(ts_sby, date >= "2019-03-25"),
           level = c(80, 90, 99)) +
  labs(y = "sby_need",
       title = "TSLM für Tagesdaten")
```



Wie die Grafik zeigt wird durch das Baseline Model zumeist ein zu hoher Wert vorhergesagt, die Spitzenwerte der Testdaten werden durch die Forecast aber nicht erreicht. Als einfaches Benchmark-Modell ist *TSLM* in diesem Fall jedoch geeignet.

```
fc %>%
  accuracy(ts_sby, list(RMSE = RMSE, MAE = MAE)) %>%
  select(-.type)
```

```
# A tibble: 5 x 3
  .model  RMSE   MAE
  <chr>  <dbl> <dbl>
1 drift   165.  147.
2 mean    154.  134.
3 naive   162.  144.
4 snaive  161.  142.
5 tslm    186.  164.
```

Die Messwerte RMSE und MAE zeigen unterschiedliche Bewertungen, in beiden Fällen wäre jedoch der einfache Durchschnitt (*mean*) das am wenigsten fehleranfällige Modell. Dass den Anforderungen an den Use Case dabei nicht ensprochen wird, ist aus den Visualisierungen gut erkennbar. Außerdem besteht hier die “Benchmark” in der bisherigen Vorgehensweise und einem Fixwert von 90 Bereitschaftsfahrenden. Im Allgemeinen sind die Fehlerwerte sehr hoch zu bewerten, aber für den zukünfigen Vergleich mit weiter entwickelten Modellen sind sie hilfreich.

```
fc %>%
  accuracy(ts_sby,
    list(quantile_score = quantile_score), probs = 0.90) %>%
  select(-.type)
```

```
# A tibble: 5 x 2
  .model quantile_score
  <chr>           <dbl>
1 drift            80.2
2 mean             66.4
3 naive            77.3
4 snaive           59.1
5 tslm             55.7
```

Ergänzend zu den irreführenden vorangegangenen Gütekriterien bietet sich für den vorliegenden Use Case zuätzlich die Bewertung der Verteilung zur Gütebestimmung an. Der ***Quantile Score*** vergleicht die vorhergesagten mit den aufgetretenen Werten unter der Annahme, dass (gem. gesetztem Parameter) 90% der realen Werte unter der entsprechenden Quantille der Verteilung liegen. Die Messung folgt einer *Pinball Loss Function* (Vermorel 2012) wobei, bei dem gesetzten 90% Parameter, Fehler die überhalb dieser Marke liegen gewichteter bewertet werden, als jene die unterhalb des Wertes liegen. Der Durchschnitt der einzelnen Bewertungen gibt den Quantile Score, der als absoluter Fehler zu verstehen ist. Je niedriger der Quantile Score, desto besser.

Weitere Messwerte beschreiben zusätzlich die Fähigkeit des Modells den Anforderungen des Use Case hinsichtlich zu geringer Schätzungen gerecht zu werden. Die bisherige Vorgehensweise einer fixen Einteilung von 90 Fahrer:innen im Bereitschaftsdienst wird hier für einen Vergleich ebenfalls berechnet.

```
# Evaluierungsdatensat - Abweichungen aus Forecast und realen Werten "sby_diff"
eval <- fc %>%
  filter(.model == "tslm") %>%
  as.tibble() %>%
  select(date, .mean) %>%
  mutate(sby = .mean) %>%
  filter(!is.na(sby)) %>%
  as_tsibble(index = date) %>%
```

```
   select(date, sby, sby_need) %>%
  filter(!is.na(sby)) %>%
  mutate(sby_diff = sby - sby_need,
         act_diff = 90 - sby_need) # Abweichungen der bisherigen Vorgehensweise
```

Positive Werte von *sby_diff* und *act_diff* bedeuten, dass zuviel Bereitschaftspersonal vorhergesagt wurde und negative Werte besagen, dass der Bedarf nicht durch die Vorhersage abgedeckt wurde. Generell sind daher positive Werte gegenüber negativen vorzuziehen.

```
# Messgrößen von TSLM

print(paste0("RMSE: ", sqrt(mean(eval$sby_diff^2))))
```

```
[1] "RMSE: 185.938781697687"
```

```
print(paste0("MAE: ", mean(abs(eval$sby_diff))))
```

```
[1] "MAE: 164.48169709389"
```

```
print(paste0("Durchschnittswert der Abweichungen (MeanError): ", mean(eval$sby_diff)))
```

```
[1] "Durchschnittswert der Abweichungen (MeanError): 108.639431818182"
```

```
print(paste0("In ", round(nrow(eval[eval$sby_diff < 0,]) / nrow(eval) * 100, 2), "% der Fälle w
```

```
[1] "In 26.23% der Fälle wurde zuwenig Personal vorhergesagt (% low)"
```

```
print(paste0("Summe der mangelhaften negativen Vorhersagen (sum low): ", sum(eval[eval$sby_diff
```

```
[1] "Summe der mangelhaften negativen Vorhersagen (sum low): -1703.18909090909"
```

```
print(paste0("Durchschnittswert der mangelhaften negativen Vorhersagen (mean low): ", mean(eval
```

```
[1] "Durchschnittswert der mangelhaften negativen Vorhersagen (mean low): -106.449318181818"
```

```
print(paste0("Größte negative Vorhersage (max low): ", min(eval[eval$sby_diff < 0,]$sby_diff)))
```

```
[1] "Größte negative Vorhersage (max low): -237.440568181818"
```

```
# Messgrößen der bisherigen Vorgehensweise

print(paste0("RMSE: ", sqrt(mean(eval$act_diff^2))))
```

```
[1] "RMSE: 160.051477374737"
```

```
print(paste0("MAE: ", mean(abs(eval$act_diff))))
```

```
[1] "MAE: 125.590163934426"
```

```
print(paste0("Durchschnittswert der Abweichungen (MeanError): ", mean(eval$act_diff)))
```

```
[1] "Durchschnittswert der Abweichungen (MeanError): -46.3770491803279"
```

```
print(paste0("In ", round(nrow(eval[eval$act_diff < 0,]) / nrow(eval) * 100, 2), "% der Fälle w
```

```
[1] "In 47.54% der Fälle wurde zuwenig Personal vorhergesagt (% low)"
```

```
print(paste0("Summe der mangelhaften negativen Vorhersagen (sum low): ", sum(eval[eval$act_diff
```

```
[1] "Summe der mangelhaften negativen Vorhersagen (sum low): -5245"
```

```
print(paste0("Durchschnittswert der mangelhaften negativen Vorhersagen (mean low): ", mean(eval
```

```
[1] "Durchschnittswert der mangelhaften negativen Vorhersagen (mean low): -180.862068965517"
```

```
print(paste0("Größte negative Vorhersage (max low): ", min(eval[eval$sby_diff < 0,]$act_diff)))
```

```
[1] "Größte negative Vorhersage (max low): -465"
```

Die nachfolgende Tabelle zeigt den Vergleich des Baseline Modells zum aktuellen Vorgehen.

Modell	RMSE	MAE	MeanError	% low	sum low	mean low	max low
Baseline	185,94	164,48	108,64	26,23	1703,19	106,45	237,44
Aktuelles Vorgehen	160,05	125,59	46,38	47,54	5245	180,86	465

Wie auch im obigen Vergleich zum Mean “Algorithmus” sind sowohl der RMSE als auch der MAE beim aktuellen Vorgehen besser bewertet. Die Verbesserung von TSLM wird deutlich, wenn es darum geht Tage mit hohem Personalbedarf ausreichend vorherzusagen. So wird bisher etwa bei der Hälfte der Tage zu wenig Bereitschaftspersonal vorgesehen. In Summe mussten in der aktuellen Vorgehensweise, in zwei Monaten Vorhersagezeitraum 5245 zusätzliche Bereitschaftsfahrer:innen aktiviert werden. Mit *TSLM* können die zwei Werte wesentlich verbessert werden.

Speichern modelspezifischer Daten

```
# Speichern der Sampledaten (hier alles)
save(ts_sby_week, ts_sby_test, ts_sby_train, eval, ts_sby,
     file = "../../00_sample_data/03_for_modeling/baseline.rda")
```

Das Modell selbst wird im Ordner **03_deployment** des Modellverzeichnisses als **sby_baseline.R** Datei abgelegt.

```
rm(list = ls()) # nach Durchlauf alles löschen
```

Literaturverzeichnis

Vermorel, Joannès. 2012. „Pinball Loss Function Definition“. https://www.lokad.com/pinball-loss-function-definition/.

A.4 Adanced Model

Einleitung

Dieses Dokument beschreibt die Erarbeitung eines ersten fortschrittlicheren Modells. Hierbei wird auf Erkenntnissen einer vorangegangenen Use Case Analyse (Grunsky 2024), sowie auf den Ergebnissen der explorativen Datenaanalyse aufgebaut.

Der Fokus liegt hierbei auf einer Modellierung der in der explorativen Datenanalyse festgehaltenen Saisonalität der eingegangenen Notrufe (*calls*) und der, ab einem festgelegten Schwellwert, linearen Korrelation dieser zum benötigten Bereitschaftspersonal (*sby_need*) - unter Berücksichtigung der Anzahl des Dienshabenden Personals (*n_duty*).

Die in der Use Case Analyse vorgeschlagene, fallspezifische Kostenfunktion wird in der u.a. Modellentwicklung jedoch noch nicht implementiert und die Evaluierung der Einfachheit halber mit herkömmlichen Methoden durchgeführt. Bekannte Messgrößen erleichtern hierbei auch die Interpretierbarkeit der Modellgüte und müssen nicht erst im Voraus erklärt werden.

Im R-package *fpp3* (**hyndman_forecasting_2021?**) ist unter anderem der Algorithmus **STL** (Seasonal and Trend decomposition using Loess) enthalten, auf den sich die gegebene Modellentwicklung konzentriert. Weitere saisonale Algorithmen wie **TSLM** und **ETS**, sowie die einfache Darstellung des **Drift** dienen in der optischen Performancemessung als Vergleich zu STL.

Wie auch in der Entwicklung des Baselinemodells liegt die Herausforderung vor allem im raschen Wechsel zwischen sehr hohem und fast keinem Bedarf an Bereitschaftspersonal. Das empfohlene Mindestmaß an etwa 35 Personen in Bereitschaft wird in der Entwicklung des Modells noch nicht berücksichtigt, aber in einem finalen Deployment als Mindestwert implementiert.

Das gewünschte Ergebnis ist eine möglichst akkurate Vorhersage der Bedarfsspitzen, eine ausreichende Generalisierung des Modells und ein Erhalt der Volatilität um, im Vergleich zu den bisher fix eingesetzten 90 Fahrer:innen, Bereitschaftskosten einsparen zu können.

Daten laden

```
load(file = "../../00_sample_data/02_processed/data_explorative.rda")
```

Aufsplitten der Daten in Trianings- und Testdaten

Die Autokorrelation des Merkmals *calls* ist, entsprechend der explorativen Datenanalyse, deutlich besser zu bewerten als jene der Zielvariablen. Dennoch soll auch in diesem Modell die Änderung des saisonalen Verhaltens mit dem Jahreswechsel auf 2019 mit berücksichtigt werden. Der Testdatensatz wird daher auch in diesem Modell mit 2 Monaten angesetzt.

```
ts_sby_train <- ts_sby %>%
  select(date, calls, MA2_8, n_duty, sby_need) %>%
  filter(!is.na(MA2_8)) %>%
  arrange(date) %>%
  slice(0:(n() - 62))

ts_sby_test <- ts_sby %>%
  select(date, calls, MA2_8, n_duty, sby_need) %>%
  filter(!is.na(MA2_8)) %>%
```

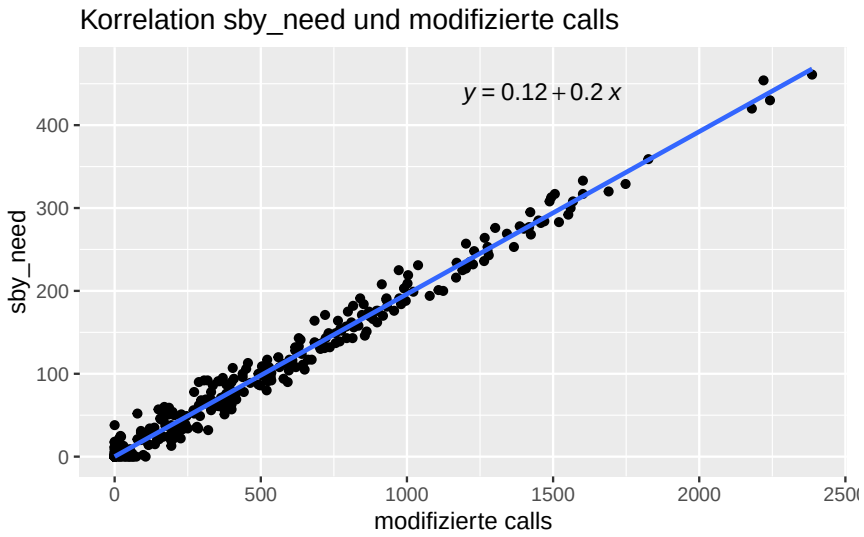
```
  arrange(date) %>%
  slice((n() - 61):n())
```

Modelerstellung

Lineares Modell

Im ersten Schritt wird das lineare Modell umgesetzt, das die Korrelation von *calls* und *sby_need* unter Einfluss von *n_duty* abbildet. Dabei wurde festgehalten, dass sich eine Änderung des diensthabenden Personals um etwa das fünffache auf den Schwellwert auswirkt, ab dem Bereitschaftspersonal benötigt wird. Wie auch in der explorativen Datenanalyse beschrieben, wird dieser Umstand im linearen Modell berücksichtigt. Die angeführte Grafik zeigt die lineare Abhängigkeit, nach Anpassung der Notrufe aufgrund von *n_duty* und nach Abzug eines einheitlichen Schwellwertes von 8150 Anrufen im Trainingsdatensatz.

```
ggplot(ts_sby_train,
            aes(x = (pmax(0,(calls - (n_duty - 1700) * 5) - 8150)),
                y = sby_need
                )) +
  geom_point() +
  stat_smooth(method = "lm", se = TRUE) +
  stat_regline_equation(label.x.npc = "center") +
  labs(x = "modifizierte calls",
       title = "Korrelation sby_need und modifizierte calls")
```



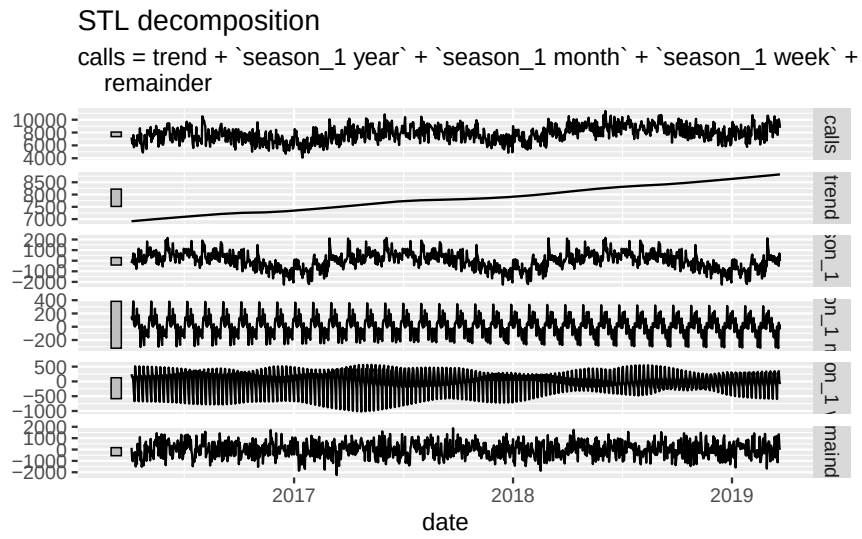

```
# Lineares Modell erzeugen
lm_sby = lm(sby_need ~ (pmax(0,(calls - (n_duty - 1700) * 5) - 8150)),
            data = ts_sby_train)
```

Advanced Forecast

In der explorativen Datenanalyse wurde das Merkmal der eingegangenen Notrufe *calls* in seiner Saisonalität als das am besten geeignete Merkmal für eine Vorhersage identifiziert. Eine erste Dekomposition der saisonalen Komponenten mit STL zeigte bereits gute Ergebnisse jedoch mit einer vermuteten "Restsaisonalität" im Remainder. Dieser wurde mit Hinzufügen der Wochenperioden begegnet, die das Ergebnis noch weiter verbessert.

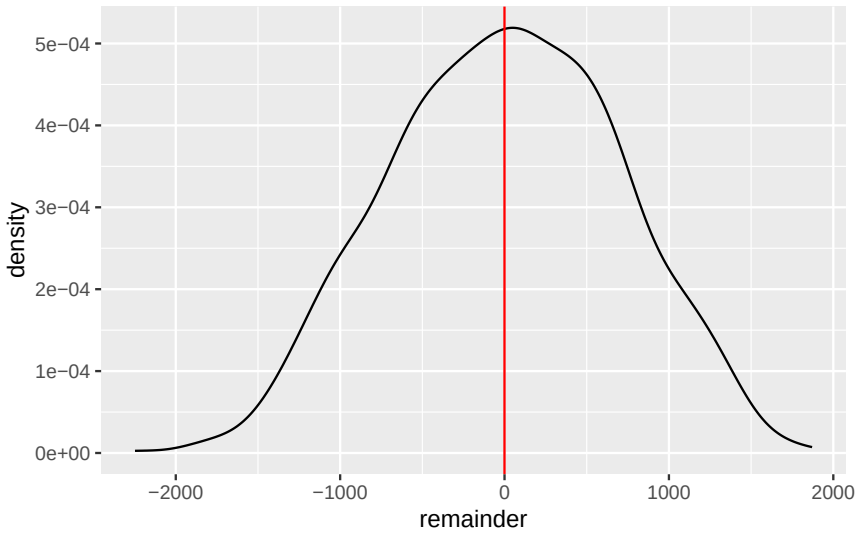
```
decomp <- ts_sby_train %>%
  model(
    STL(calls ~ trend(window = 365) +
                  season(period = "1 year", window = 540) +
                  season(period = "1 month", window = 61) +
                  season(period = "1 week", window = 28))) %>%
  components()

  decomp %>% autoplot()
```



Wie die u.a. Darstellung verdeutlicht sind die "Remainder" nahezu um den Wert 0 (*mean* in Rot) Normverteilt und entsprechen damit einer Annäherung an ein optimales Ergebnis.

```
ggplot(data = decomp, aes(x = remainder)) +
  geom_density() +
  geom_vline(xintercept = mean(decomp$remainder), color = "red")
```



Der nachfolgende Code erstellt die zu vergleichenden Modells, wobei der Fokus auf STL als *decomposition model* liegt. Das Modell ermittelt die oben dargestellte Saisonalität und wendet einen einfachen *Drift* Algorithmus auf die saisonal bereinigten Daten an.

```
decomp_spec <- decomposition_model(
  STL(calls ~ trend(window = 365) +
                    season(period = "1 year", window = 540) +
                    season(period = "1 month", window = 61) +
                    season(period = "1 week", window = 28)),
    RW(season_adjust ~ drift())
)

# Training
t <- system.time(
  progressr::with_progress(
    sby_model_adv <- ts_sby_train %>%
      model(
        stl = decomp_spec,
        ets = ETS(calls ~ error("A") + trend("A") + season("A")),
        drift = RW(calls ~ drift()),
        tslm = TSLM(calls ~ trend() + season())
      )
  ), gcFirst = TRUE
)

print(paste0(" Die Benötigte Zeit für das Training betrug ", round(t[3],2), " Sekunden."))
```

```
[1] " Die Benötigte Zeit für das Training betrug 0.37 Sekunden."
```

```
# Vorhersage
t <- system.time(
  progressr::with_progress(
    sby_fc_adv <- sby_model_adv %>%
```

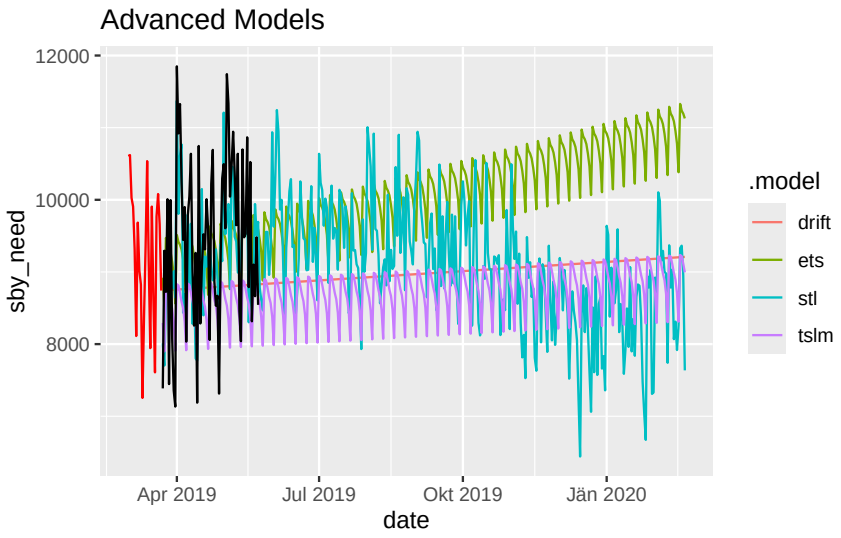
```
      forecast(h = "11 months", level = c(80))
  ), gcFirst = TRUE
)

print(paste0(" Die Benötigte Zeit für die Vorhersage war ", round(t[3],2), " Sekunden."))
```

```
[1] " Die Benötigte Zeit für die Vorhersage war 0.6 Sekunden."
```

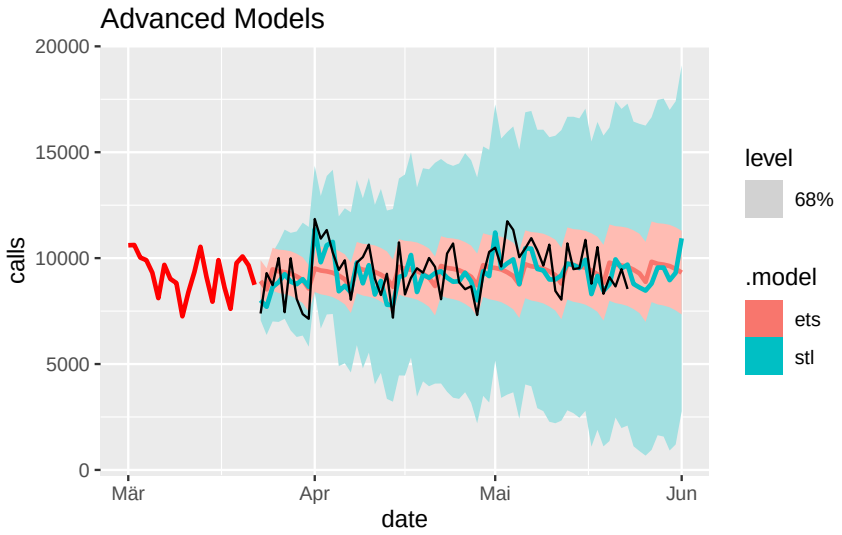
Die benötigte Zeit für die Berechnung der Modelle sowie für das Forecasting kann durchaus als "verschwindend" bezeichnet werden. Von dieser Seite ist keine weitere Hürde für die Umsetzung der Modelle zu erwarten. In der späteren Anwendung wird natürlich nur ein Forecastzeitraum von 2 Monaten benötigt, hier ermöglichen parametrisierten 11 Monate jedoch die Beurteilung, ob das Modell die jährliche Saisonalität und den Trend "verstanden" hat.

```
sby_fc_adv %>%
  autoplot(ts_sby_test,
           level = NULL) +
  autolayer(ts_sby_train %>% filter(date >= "2019-03-01"), colour = "red" ) +
  labs(y = "sby_need",
       title = "Advanced Models")
```



In dem o.a. Plot sind die Trainingsdaten Rot und die Testdaten Schwarz dargestellt. Die Vorhersagen durch die einzelnen Algorithmen sind farblich in der Legende kodiert. In der visuellen Bewertung wurde der allgemeine Trend im *Drift* gut erkannt und dargestellt. Selbsverständlich wird dieser Algorithmus aber zu keinem zufriedenstellenden Ergebnis führen. *ETS* und *TSLM* lassen zusätzlich noch eine periodische monatliche Saisonalität erkennen, setzen aber beide zu tief an. Eventuell könnte bei diesen Modellen mit einer Quantilsschätzung ein brauchbares Ergebnis erzielt werden. Die Tage mit wenig Bedarf sowie die jährliche Saisonalität sind allerdings nicht abgebildet. Überraschend gut wird die Volatilität und die Jahresperiode im *STL* modelliert. Bei näherer Betrachtung erkennt man, dass die Monatspeaks nahezu am Punkt sind (wenn auch meistens einen Tick zu niedrig), aber oft eben auch um ein oder zwei Tage verschoben. Ein Overfitting auf die Trainingsdaten scheint naheliegend. Möglicherweise hilft der Einsatz des Konfidenzintervalls um die Kurve zu glätten und auch die hohen

```r
sby_fc_adv %>%
  filter(.model == "stl" | .model == "ets",
         date <= "2019-06-01") %>%
  autoplot(ts_sby_test,
           level = c(68),
           linewidth = 1) +
  autolayer(ts_sby_train %>% filter(date >= "2019-03-01"), colour = "red", linewidth = 1) +
  labs(y = "calls",
       title = "Advanced Models")
```



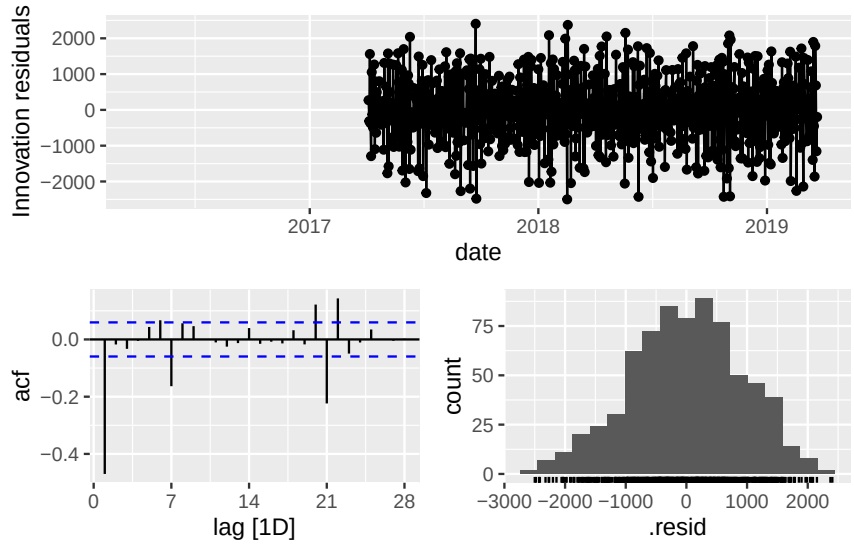
Im Vergleich von *STL* und *ETS* mit 68% Konfidenzinterwall (+/- eine Standardabweichung) für eine Vorhersage von zwei Monaten sieht man, dass *ETS* mit guter Generalisierung die Spitzen meistens abdeckt, aber die Volatilität nicht hinbekommt. Bei *STL* wird mit zunehmender vorhersagedauer die Verwendung des Konfidenzintervalls unbrauchbar, weil die Streuung enorm ist. Die Entscheidung fällt auf eine Verwendung von *STL* in der Weiterentwicklung des Modells. Zu beachten ist, dass die fehlende Generalisierung nicht durch Verwendung des Konfidenzintervalls erzielt werden kann. Um die Volatilität des Modells zu erhalten ist der Punktschätzer zu verwenden und die Generalisierung anderweitig zu ermöglichen.

Evaluierung

Die nachstehenden Werte berechnen die Güte des Modells analog zum Baselinemodell. Vergleichbar sind diese, aufgrund der unterschieldichen Prädiktoren aber nicht. Sollten noch weitere Modelle auf Basis von *calls* erstellt werden, können diese Werte herangesogen werden.

Der u. a. angeführte Residualsplot zeigt ein nahezu optimales Verhalten des Algorithmus. In den Residuals ist auf den ersten Blick kein “Rest-Pattern” in den Daten zu beobachten und genauso wenig im Autokorrelationswert. Die Residuals wären im besten Fall mit einem arithmetischen Mittel von 0 normal verteilt. Das trifft nicht ganz zu, die Annäherung an das gewünschte Verhalten ist aber gut erkennbar. Dieser Plot zeigt allerdings nur, wie gut die Realität mit dem erstellten Modell (den sogenannten *fitted values*) erklärt werden kann. Es ist keine Bewertung der Vorhersagefähigkeit.

```
sby_model_adv %>%
  select(stl) %>%
  gg_tsresiduals()
```




```
sby_fc_adv %>%
  accuracy(ts_sby, list(RMSE = RMSE, MAE = MAE)) %>%
  select(-.type)
```

```
# A tibble: 4 x 3
  .model  RMSE   MAE
  <chr>  <dbl> <dbl>
1 drift  1314. 1086.
2 ets    1063.  915.
3 stl     990.  828.
4 tslm   1388. 1131.
```

Auch die verwendeten Messwerte RMSE und MAE für die Messung der Güte zeigen, dass *STL* am Besten bewertet wird. Wie auch in der Bewertung des Baselinemodells ist zu berücksichtigen, dass diese Messwerte nicht immer ausreichend aussagekräftig den Bedarf des Use Case abbilden. Deswegen gilt es mehrere Faktoren anzusehen und sich die Vorhersagen auch zu veranschaulichen, was hier auch getan wird.

```
sby_fc_adv %>%
  accuracy(ts_sby,
    list(quantile_score = quantile_score), probs = 0.68) %>%
  select(-.type)
```

```
# A tibble: 4 x 2
  .model quantile_score
  <chr>           <dbl>
1 drift           1783.
```

```
3 stl             1448.
4 tslm            1071.
```

In der Bewertung der Verteilung durch den Quantile Score, sieht man die Auswirkung der breiten Streuung von *STL* und die bereits festgestellte Unbrauchbarkeit einer Schätzung durch das Konfidenzintervall. Wie ebenso bereits anhand der Visualisierung beurteilt schneidet *ETS* hier deutlich besser ab.

Vorhersage sby_need

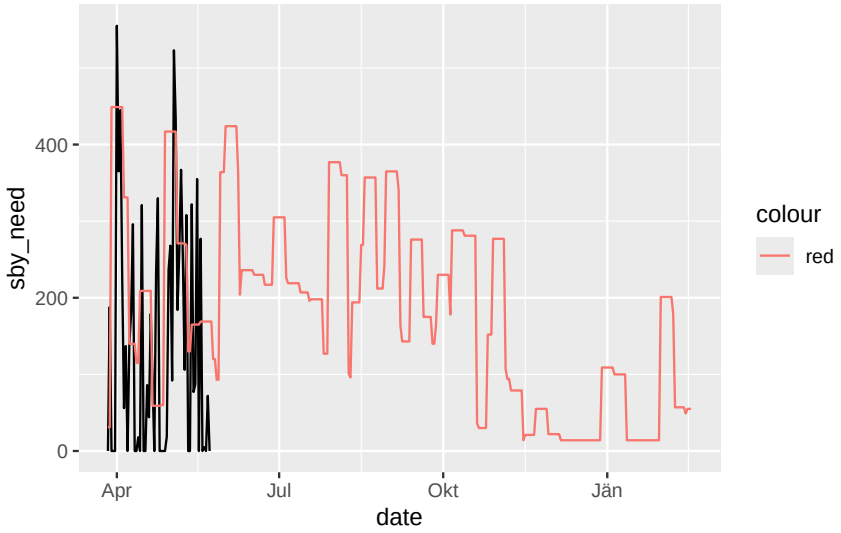
In diesem Abschnitt wird nun das zuvor erstellte lineare Modell auf die Vorhersage von *calls* angewandt. In der Einleitung wurden die Anforderungen an das Ergebnis bereits festgehalten

- akkurate Vorhersage der Bedarfsspitzen
- ausreichende Generalisierung des Modells
- Erhalt der Volatilität um Bereitschaftskosten einzusparen

Die fehlende Generalisierung wird erreicht indem ein gleitendes Maximum über die vorhergesagten *calls* laufen gelassen wird. “Verschobene Spitzen” sollten dadurch ebenfalls besser abgedeckt werden.

Der für das lineare Modell benötigte Wert für das diensthabende Personal, wird im u.a. Code festgelegt und dem neu gebildeten Datensatz mitgegeben. Diese Vorgehensweise ermöglicht später in der praktischen Anwendung ebenso, vorab das geplante diensthabende Personal für den Vorhersagezeitraum flexibel einzugeben, und erst dann die Vorhersage arbeiten zu lassen.

```
# vorgesehenes diensthabendes Personal
duty_pers <- 1900

# neuer Datensatz mit gleitenden Maxima zu Generalisierung (Prediction-Dataset)
call_pred_adv <- sby_fc_adv %>%
  filter(.model == "stl") %>%
  as.tibble() %>%
  select(date, .mean) %>%
  mutate(n_duty = duty_pers,
         calls = slider::slide_dbl(.mean,
                                   max,
                                   .before = 3, .after = 3,
                                   .complete = TRUE)) %>%
  filter(!is.na(calls)) %>%
  as_tsibble(index = date)

# Anwendung lineares Modell und hinzufügen von sby zu Prediction-Dataset
sby_pred <- predict(lm_sby, call_pred_adv, interval = "prediction")
call_pred_adv$sby <- pmax(0,round(sby_pred[,3],0))

# Anzeige im Vergleich zu tatsächlichem sby_need
ggplot() +
  geom_line(data = filter(ts_sby_test, date > "2019-03-26"),
            aes(x = date, y = sby_need)) +
  geom_line(data = filter(call_pred_adv, date > "2019-03-26"),
            aes(x = date, y = sby, colour = "red"))
```



Die Generalisierung wurde erreicht, die Volatilität erhalten. Die Spitzenwerte werden oft nicht erreicht. Das Modell ist dennoch in dieser Form anwendbar. Eine korrekte Vorhersage der Spitzenwerte im Personalbedarf ist, auch wie im Basismodell, immer mit einer Gesamtkostenerhöhung für den Bereitschaftsdienst verbunden, da nahezu immer mehr als 90 Personen als Bedarf vorhergesagt werden.

Evaluierung sby_need

Nach dem es sich bei dem Prediction-Dataset nicht mehr um eine Forecast gem. *fpp3*-Package handelt, können die dort bereitgestellten Funktionen zur Bewertung nicht mehr angewandt werden. Für die Vergleichbarkeit mit dem Baselinemodell und ggf. weiteren fortgeschrittenen Modellen werden einige Bewertungskriterien daher “manuell” bereitgestellt.

```
# Evaluierungsdatensat - Abweichungen aus Forecast und realen Werten "sby_diff"
eval <- call_pred_adv %>%
  right_join(ts_sby_test, by = "date") %>%
   select(date, sby, sby_need) %>%
  filter(!is.na(sby)) %>%
  mutate(sby_diff = sby - sby_need)
```

Positive Werte von *sby_diff* bedeuten, dass zuviel Bereitschaftspersonal vorhergesagt wurde und negative Werte besagen, dass der Bedarf nicht durch die Vorhersage abgedeckt wurde. Generell sind daher positive Werte gegenüber negativen vorzuziehen.

```
print(paste0("RMSE: ", sqrt(mean(eval$sby_diff^2))))
```

```
[1] "RMSE: 190.310094854135"
```

```
print(paste0("MAE: ", mean(abs(eval$sby_diff))))
```

```
[1] "MAE: 153.35593220339"
```

```
print(paste0("Durchschnittswert der Abweichungen: ", mean(eval$sby_diff)))
```

```
[1] "Durchschnittswert der Abweichungen: 88.4745762711864"
```

```
print(paste0("In ", round(nrow(eval[eval$sby_diff < 0,]) / nrow(eval) * 100, 2), "% der Fälle w
```

```
[1] "In 27.12% der Fälle wurde zuwenig Personal vorhergesagt"
```

```
print(paste0("Summe der mangelhaften negativen Vorhersagen: ", sum(eval[eval$sby_diff < 0,]$sby
```

```
[1] "Summe der mangelhaften negativen Vorhersagen: -1914"
```

```
print(paste0("Durchschnittswert der mangelhaften negativen Vorhersagen: ", mean(eval[eval$sby_d
```

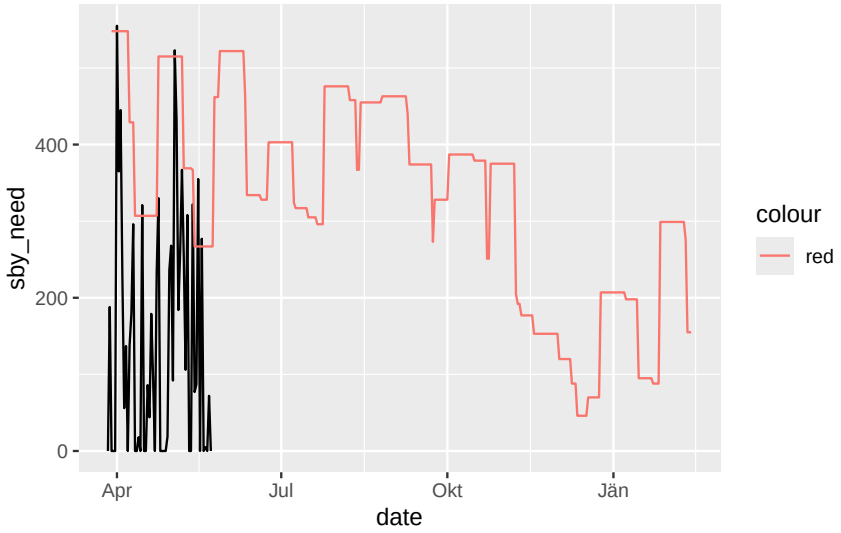
```
[1] "Durchschnittswert der mangelhaften negativen Vorhersagen: -119.625"
```

```
print(paste0("Größte negative Vorhersage: ", min(eval[eval$sby_diff < 0,]$sby_diff)))
```

```
[1] "Größte negative Vorhersage: -271"
```

Testhalber wird in der nachgestellten Grafik die Anzahl des diensthabenden Personals um 100 gesenkt, um die Auswirkungen zu beobachten. Gleichzeitig wird der Zeitraum für das gleitende Maximum auf zwei Wochen erhöht um noch mehr Generalisierung zu erreichen.

```
# vorgesehenes diensthabendes Personal (Wert um 100 gesenkt!)
duty_pers <- 1800

# neuer Datensatz mit gleitenden Maxima zu Generalisierung (Prediction-Dataset)
call_pred_adv <- sby_fc_adv %>%
  filter(.model == "stl") %>%
  as.tibble() %>%
  select(date, .mean) %>%
  mutate(n_duty = duty_pers,
         calls = slider::slide_dbl(.mean,
                                   max,
                                   .before = 6, .after = 7,
                                   .complete = TRUE)) %>%
  filter(!is.na(calls)) %>%
  as_tsibble(index = date)

# Anwendung lineares Modell und hinzufügen von sby zu Prediction-Dataset
sby_pred <- predict(lm_sby, call_pred_adv, interval = "prediction")
call_pred_adv$sby <- pmax(0,round(sby_pred[,3],0))

# Anzeige im Vergleich zu tatsächlichem sby_need
ggplot() +
  geom_line(data = filter(ts_sby_test, date > "2019-03-26"),
            aes(x = date, y = sby_need)) +
  geom_line(data = filter(call_pred_adv, date > "2019-03-26"),
```



Die zweite Version sieht deutlich besser aus. Die Modifikation des dienshabenden Personals wird im Code für das Modell implementiert werden müssen, dadurch wird der vorhergesagte Bedarf höher angesetzt und ein Puffer eingebaut.

```
# Evaluierungsdatensat - Abweichungen aus Forecast und realen Werten "sby_diff"
eval <- call_pred_adv %>%
  right_join(ts_sby_test, by = "date") %>%
   select(date, sby, sby_need) %>%
  filter(!is.na(sby)) %>%
  mutate(sby_diff = sby - sby_need)
```

```
print(paste0("RMSE: ", sqrt(mean(eval$sby_diff^2))))
```

```
[1] "RMSE: 311.912046897839"
```

```
print(paste0("MAE: ", mean(abs(eval$sby_diff))))
```

```
[1] "MAE: 270.589285714286"
```

```
print(paste0("Durchschnittswert der Abweichungen: ", mean(eval$sby_diff)))
```

```
[1] "Durchschnittswert der Abweichungen: 266.053571428571"
```

```
print(paste0("In ", round(nrow(eval[eval$sby_diff < 0,]) / nrow(eval) * 100, 2), "% der Fälle w
```

```
[1] "In 8.93% der Fälle wurde zuwenig Personal vorhergesagt"
```

```
print(paste0("Summe der mangelhaften negativen Vorhersagen: ", sum(eval[eval$sby_diff < 0,]$sby
```

```
[1] "Summe der mangelhaften negativen Vorhersagen: -127"
```

```
print(paste0("Durchschnittswert der mangelhaften negativen Vorhersagen: ", mean(eval[eval$sby_d
```

```
[1] "Durchschnittswert der mangelhaften negativen Vorhersagen: -25.4"
```

```
print(paste0("Größte negative Vorhersage: ", min(eval[eval$sby_diff < 0,]$sby_diff)))
```

```
[1] "Größte negative Vorhersage: -88"
```

Jene Werte, die die allgmeine Modellgüte ausdrücken wurden durch die Anpassung verschlechtert. Profitiert hat hingegen die Anforderung ausreichend Bereitschaftspersonal vorherzusagen. Für eine entgültige Entscheidung wird ggf, das Management mit einbezogen, bzw. mit dem Management gemeinsam eine gültige Kostenfunktion zur Bewertung erstellt werden müssen. Ein Beispiel für so eine Kostenfunktion wurde bereits in der Use Case Analyse vorgeschlagen.

Vergleich der Modelle

Modell	RMSE	MAE	MeanError	% low	sum low	mean low	max low
Baseline	185,94	164,48	108,64	26,23	1703,19	106,45	237,44
Aktuelles Vorgehen	160,05	125,59	46,38	47,54	5245	180,86	465
Advanced Version 1	190,31	153,36	88,47	27,12	1914	119,62	271
Advanced Version 2	311,91	270,59	266,05	8,93	127	25,4	88

Im Vergleich ensprechen der RMSE und MAE der ersten Version dieses Modells dem Baseline-Modell, ansonsten ist das Baseline-Modell jedoch besser zu bewerten als Version 1 des Advanced Modells. Die Version 2 schneidet beim RMSE und MAE wesentlich schlechter ab. Dennoch werden in der zweiten Version deutlich weniger Fälle, zu niedriger Schätzungen verzeichnet. Hier liegt die wesentliche Verbesserung im Advanced-Modell (V2). Es gilt zu klären, wo der Schwerpunkt der Vorhersage liegen soll. Vorerst wird mit der Version 2 des Advanced-Modells weitergearbeitet.

Speichern modelspezifischer Daten

```
# Speichern der Sampledaten (hier alles)
save(ts_sby_test, ts_sby_train, eval, call_pred_adv, ts_sby,
     file = "../../00_sample_data/03_for_modeling/advanced.rda")
```

Das Modell selbst wird im Ordner **03_deployment** des Modellverzeichnisses als **sby_model_advanced.** Datei abgelegt.

```
rm(list = ls()) # nach Durchlauf alles löschen
```

Literaturverzeichnis

Grunsky, Georg. 2024. „Rettungsdienst Berliner Rotes Kreuz MachineLearning-Einsatz in der Bereit-

A.5 Prophet Model

Einleitung

Dieses Dokument beschreibt die Erarbeitung eines weiteren fortschrittlichen Modells. Hierbei wird ebenso auf Erkenntnissen der vorangegangenen Use Case Analyse (Grunsky 2024), sowie auf den Ergebnissen der explorativen Datenaanalyse aufgebaut.

Der Fokus liegt auch hierbei auf einer Modellierung der in der explorativen Datenanalyse festgehaltenen Saisonalität der eingegangenen Notrufe (*calls*) und der, ab einem festgelegten Schwellwert, linearen Korrelation dieser zum benötigten Bereitschaftspersonal (*sby_need*) - unter Berücksichtigung der Anzahl des Dienshabenden Personals (*n_duty*).

In der Use Case Analyse wurden die überblicksmäßig die Stärken und Schwächen verschiedener Machine Learning Algorithmen verglichen. Aufgrund der Einfachheit, Interpretierbarkeit und Tauglichkeit für saisonale Zeitreihen mit wenig Features schien sich **Facebook Prophet** als möglicher Algorithmus für den gegenständlichen Anwendungsfall zu eignen. Dieser Algorithmus wurde für Zeitreihenvorhersagen entwickelt und verwendet dabei ein zusammengesetztes Modell aus Trend, Saisonalität und speziellen Ereignissen in Zeitreihen. (Taylor und Letham, o. J.).

Der Vorteil von **Facebook Prophet** liegt darin, dass in einer möglichen, zukünftigen Weiterentwicklung des Modells auch Tage mit besonderen Ereignissen, wie zB Veranstaltungen und speziellen Wetterverhältnissen, mit modelliert werden und a priori in eine Vorhersage mit einfliesen könnten.

Im R-package *fpp3* (**hyndman_forecasting_2021?**) ist zwar *Prophet* als nicht enthalten, kann aber mit laden des Packages *fable.prophet* hinzugefügt werden.

Wie auch in der Entwicklung des Baselinemodells liegt die Herausforderung vor allem im raschen Wechsel zwischen sehr hohem und fast keinem Bedarf an Bereitschaftspersonal. Das empfohlene Mindestmaß an etwa 35 Personen in Bereitschaft wird in der Entwicklung des Modells noch nicht berücksichtigt, aber in einem finalen Deployment als Mindestwert implementiert.

Das gewünschte Ergebnis ist eine möglichst akkurate Vorhersage der Bedarfsspitzen, eine ausreichende Generalisierung des Modells und ein Erhalt der Volatilität um, im Vergleich zu den bisher fix eingesetzten 90 Fahrer:innen, Bereitschaftskosten einsparen zu können.

Daten laden

```
load(file = "../../00_sample_data/02_processed/data_explorative.rda")
```

Aufsplitten der Daten in Trianings- und Testdaten

Die Autokorrelation des Merkmals *calls* ist, entsprechend der explorativen Datenanalyse, deutlich besser zu bewerten als jene der Zielvariablen. Dennoch soll auch in diesem Modell die Änderung des saisonalen Verhaltens mit dem Jahreswechsel auf 2019 mit berücksichtigt werden. Der Testdatensatz wird daher auch in diesem Modell mit 2 Monaten angesetzt.

```
ts_sby_train <- ts_sby %>%
  select(date, calls, MA2_8, n_duty, sby_need) %>%
  filter(!is.na(MA2_8)) %>%
  arrange(date) %>%
  slice(0:(n() - 62))
```

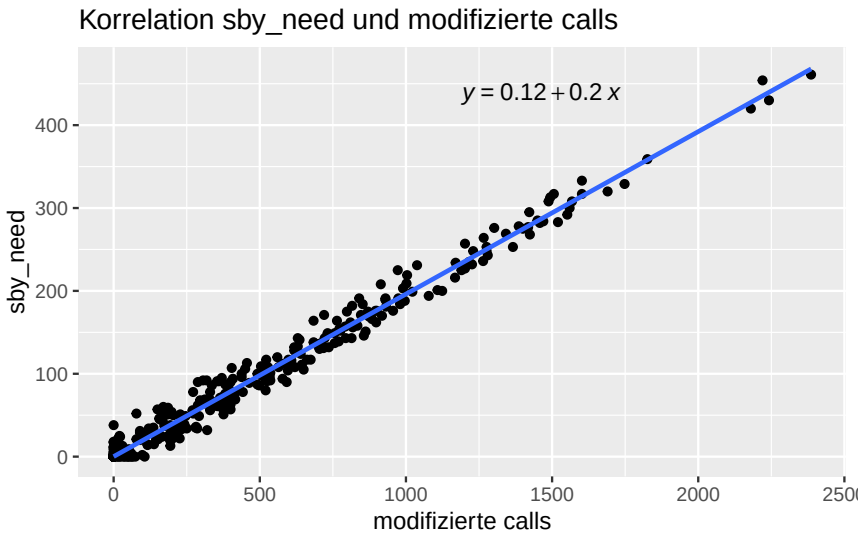
```r
ts_sby_test <- ts_sby %>%
  select(date, calls, MA2_8, n_duty, sby_need) %>%
  filter(!is.na(MA2_8)) %>%
  arrange(date) %>%
  slice((n() - 61):n())
```

Modelerstellung

Lineares Modell

Im ersten Schritt wird erneut das lineare Modell umgesetzt, das die Korrelation von *calls* und *sby_need* unter Einfluss von *n_duty* abbildet. Die angeführte Grafik zeigt, wie auch im Advanced-Modell die lineare Abhängigkeit, nach Anpassung der Notrufe aufgrund von *n_duty* und nach Abzug eines einheitlichen Schwellwertes von 8150 Anrufen im Trainingsdatensatz.

```r
ggplot(ts_sby_train,
            aes(x = (pmax(0,(calls - (n_duty - 1700) * 5) - 8150)),
                y = sby_need
                )) +
  geom_point() +
  stat_smooth(method = "lm", se = TRUE) +
  stat_regline_equation(label.x.npc = "center") +
  labs(x = "modifizierte calls",
       title = "Korrelation sby_need und modifizierte calls")
```




```r
lm_sby = lm(sby_need ~ (pmax(0,(calls - (n_duty - 1700) * 5) - 8150)),
            data = ts_sby_train)
```

Prophet Forecast

Der nachfolgende Code erstellt das Prophet-Modell mit einem linearen Trend, sowie jährlichen, monatlichen und wöchentlichen Saisonalitäten.

```
# Training
t <- system.time(
  progressr::with_progress(
    sby_model_prophet <- ts_sby_train %>%
      model(
        prophet = prophet(calls ~ growth("linear",
                                         n_changepoints = 15) +
                            season(period = 7,
                                   order = 20,
                                   type = "additive",
                            name = "week") +
                            season(period = 28,
                                   order = 10,
                                   type = "multiplicative",
                                   name = "month") +
                            season(period = "year",
                                   order = 15,
                                   type = "multiplicative",
                                   name = "year")
                            )
      )
  )
)

print(paste0(" Die Benötigte Zeit für das Training betrug ", round(t[3],2), " Sekunden."))
```

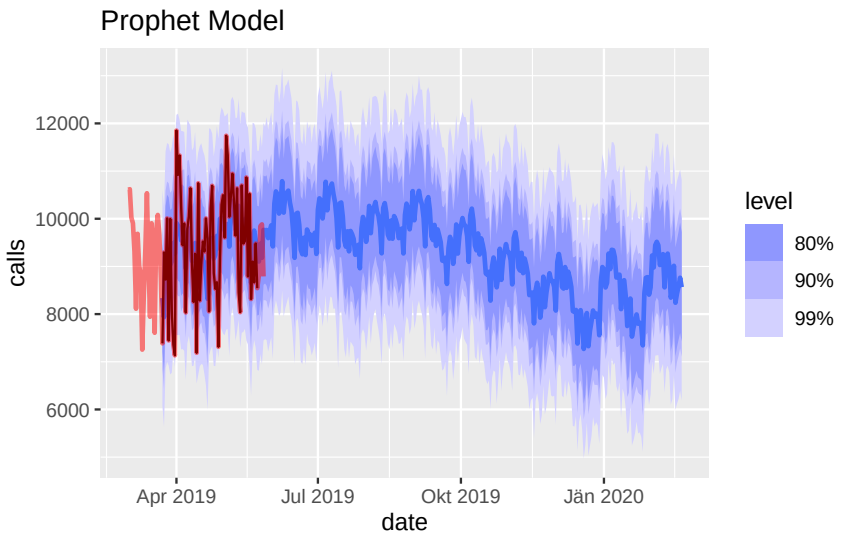
```
[1] " Die Benötigte Zeit für das Training betrug 3.32 Sekunden."
```

```
# Vorhersage
t <- system.time(
  progressr::with_progress(
    sby_fc_prophet <- sby_model_prophet %>%
      #forecast(h = "2 months") %>%
      forecast(h = "11 months")
  ), gcFirst = TRUE
)

print(paste0(" Die Benötigte Zeit für die Vorhersage war ", round(t[3],2), " Sekunden."))
```

```
[1] " Die Benötigte Zeit für die Vorhersage war 10.28 Sekunden."
```

Die benötigte Rechenzeit ist bei *prophet* zwar deutlich höher als bei den vorangegangenen Modellen, liegt aber immer noch im Bereich weniger Sekunden und ist damit gut einsetzbar.

```
sby_fc_prophet %>%
  autoplot(ts_sby_test,
           level = c(80,90,99), linewidth = 1) +
  autolayer(ts_sby %>% filter(date >= "2019-03-01"),
            colour = "red", linewidth = 1, alpha = 0.5) +
  labs(y = "calls",
       title = "Prophet Model")
```



Die oben angeführte Grafik zeigt die tatsächlich eingegangenen Anrufe in Rot, die gleitende 99% Quantille in Schwarz und die Vorhersage für elf Monate in blauer Farbe.

Die Glättung durch die gleitenden Quantille ist gut erkennbar und hüllt die tatsächlichen Notrufe ein. Die Spitzenwerte werden dadurch erfolgreich abgedeckt, was jedoch auf Kosten der Tage mit niedriegem Bedarf geschieht.

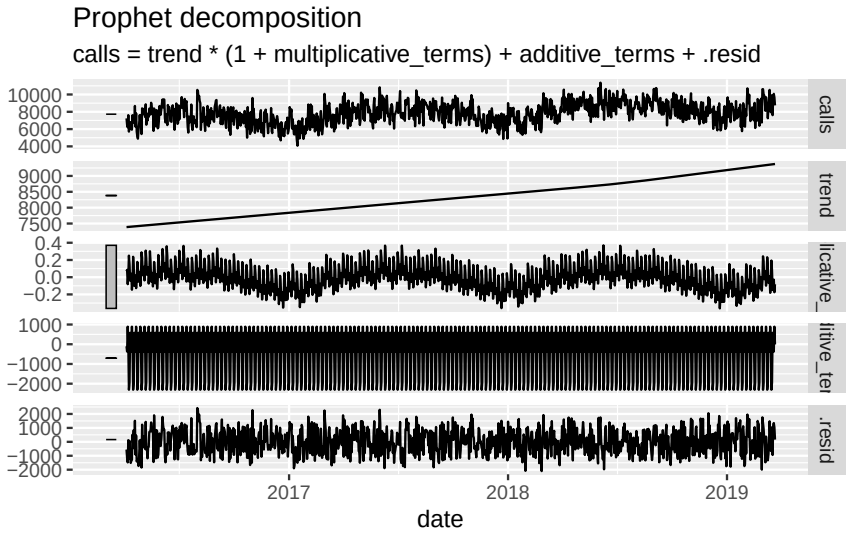
Das Modell schafft es daher mit Masse, die Spitzenwerte abzudecken. In einigen wenigen Fällen gelingt dies trotzdem nicht. Die Saisonalitäten sind deutlich erkennbar, auch wenn der zu erwartende Abfall in den Wintermonaten augenscheinlich zu wenig ausgeprägt ist.

In diesem Plot wurden testhalber auch die Konfidenzinteralle für die 80%, 90% und 99% Quantille dargestellt. Im Gegensatz zum *STL*-Algorithmus des Advanced-Modells, bleiben diese hier auch über einen Vorhersagezeitraum von 11 Monaten relativ konstant und weisen daher ebenso eine Volatilität auf. Diese Volatilität könnte für die Vorhersage von Tagen mit niedrigem Bedarf von Vorteil sein.

Evaluierung

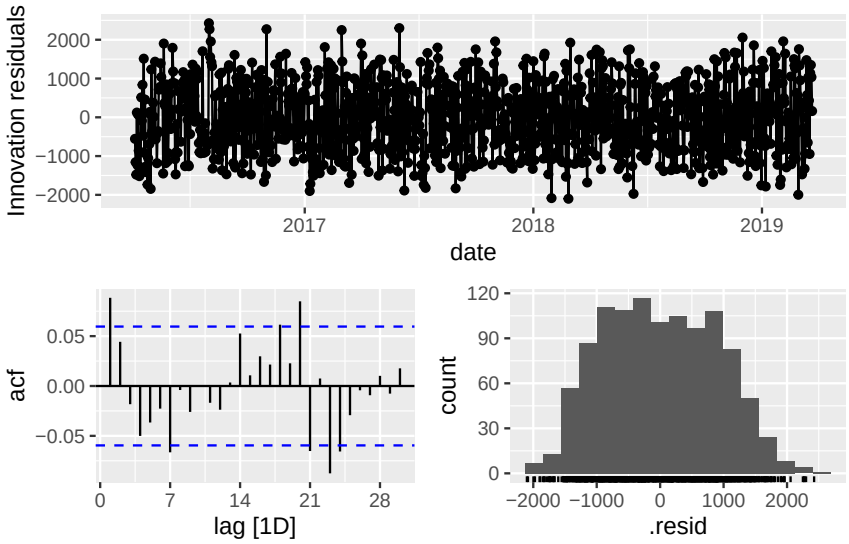
Gemäß den gesetzten Trainingsparametern unterscheidet der *prophet* Algorithmus additive und multiplikative Saisonalitäten. Der Unterschied liegt in der Variabilität der Saisonalitäten (multiplikativ ist hierbei variabler als additiv). Diese werden in der unten angeführten Decomposition des Modells gesondert angeführt. Die Residuals weisen augenscheinlich kein Auffälliges “Rest-Pattern” auf, die Modellierung scheint daher die Trainingsdaten gut aufzunehmen.

```
sby_model_prophet %>%
  components() %>%
```



Der u. a. angeführte Residualsplot zeigt auch bei *Prophet* ein nahezu optimales Verhalten des Algorithmus. In den Residuals ist auf den ersten Blick kein deutliches “Rest-Pattern” in den Daten zu beobachten und genauso wenig im Autokorrelationswert. Die Verteilung der Residuals ist im Vergleich zum Advanced-Modell etwas breiter.

```
gg_tsresiduals(sby_model_prophet)
```




```
sby_fc_prophet %>%
  accuracy(ts_sby, list(RMSE = RMSE,
                        MAE = MAE,
                        quantile_score = quantile_score),
           probs = 0.68) %>%
  select(-.type)
```

```
# A tibble: 1 x 4
  .model   RMSE   MAE quantile_score
  <chr>   <dbl> <dbl>          <dbl>
1 prophet 1026.  852.           746.
```

Der RMSE von Prohpet-Modells ist etwas schlechter als des des Advanced-Modells. Der MAE ist wiederum recht ähnlich. Hier wird allerdings nur die Vorhersage von *calls* in einem ersten Schuss bewertet, um zu sehen, ob eine weitere Arbeit mit diesem Modell sinnvoll erscheint.

Beim Quantile Score ist der Prophet-Algorithmus vergleichbar mit dem ETS-Algorithmus, der in der Modellierung des Advanced-Modells untersucht wurde, und damit deutlich besser als der des, schlussendlich verwendeten, STL-Algorithmus. Wie auch die o.a. Visualisierung zeigt, bleiben die Quantillen auch in einer weitreichenderen Forecast relativ konstant, wohingegen bei *STL* die unschärfe über die Zeit wesentlich zunahm und eine Verwendung der Quantille unbrauchbar machte.

Vorhersage sby_need

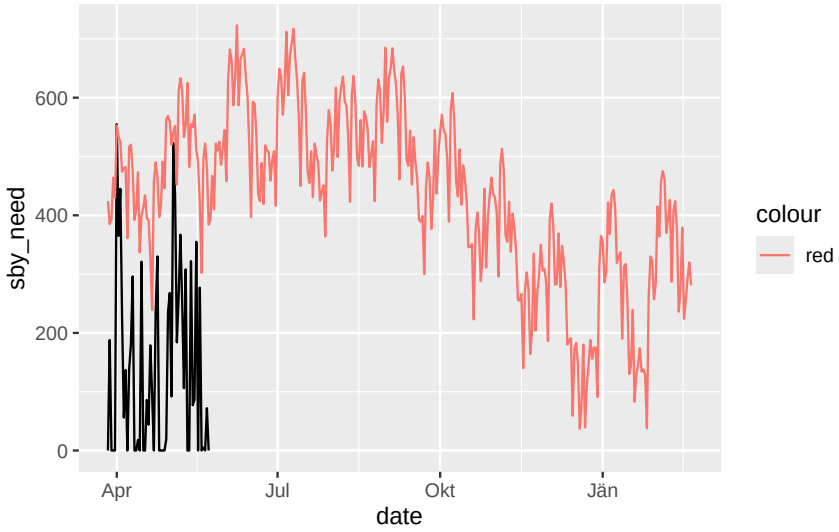
Im Advanced-Modell wurde versucht einen Puffer für die Spitzenwerte anhand eines gleitenden Maximums über die vorhergesagten Notrufe zu erreichen. Da bei *Prophet* die Verwendung von Konfidenzintervallen möglich erscheint, wird in diesem Ansatz der Puffer durch Verwendung der 90% Quantille als Vorhersagewert erreicht. Das Schöne hierbei ist, dass der zu verwendente Prozentsatz für die Berechnung einem codierten Modell auch als Hyperparameter mitgegeben werden kann.

```
sby_fc_prophet <- sby_fc_prophet %>%
  mutate(.mean = hilo(calls,90)$upper)
```

Wie auch im Advance-Modell wird, zum Vergleich der Ergebnisse, die Höhe des diensthabenden Personals um 100 reduziert. Damit wird die Anzahl des vorhergesagten Bereitschaftspersonals etwas angehoben. Auch die Höhe des vorgesehenen Bereitschaftspersonals wird für das Modell ein verpflichtender Parameter sein. Die Planungsstelle für den Bereitschaftsdienst, sollte aber im Vorhinein wissen, wie viel Personal im vorherzusagenden Zeitraum regulär im Dienst sein soll.

```
# vorgesehenes diensthabendes Personal
duty_pers <- 1800

# neuer Datensatz mit gleitenden Maxima zu Generalisierung (Prediction-Dataset)
call_pred_adv <- sby_fc_prophet %>%
  as.tibble() %>%
  select(date, .mean) %>%
  mutate(n_duty = duty_pers,
         calls = .mean) %>%
  filter(!is.na(calls)) %>%
  as_tsibble(index = date)

# Anwendung lineares Modell und hinzufügen von sby zu Prediction-Dataset
sby_pred <- predict(lm_sby, call_pred_adv, interval = "prediction")
call_pred_adv$sby <- pmax(0,round(sby_pred[,3],0))

# Anzeige im Vergleich zu tatsächlichem sby_need
ggplot() +
  geom_line(data = filter(ts_sby_test, date > "2019-03-26"),
            aes(x = date, y = sby_need)) +
```

```
  geom_line(data = filter(call_pred_adv, date > "2019-03-26"),
            aes(x = date, y = sby, colour = "red"))
```



Evaluierung sby_need

Nach dem es sich bei dem Prediction-Dataset nicht mehr um eine Forecast gem. *fpp3*-Package handelt, können die dort bereitgestellten Funktionen zur Bewertung nicht mehr angewandt werden. Für die Vergleichbarkeit mit dem Baselinemodell und ggf. weiteren fortgeschrittenen Modellen werden einige Bewertungskriterien daher “manuell” bereitgestellt.

```
# Evaluierungsdatensat - Abweichungen aus Forecast und realen Werten "sby_diff"
eval <- call_pred_adv %>%
  right_join(ts_sby_test, by = "date") %>%
   select(date, sby, sby_need) %>%
  filter(!is.na(sby)) %>%
  mutate(sby_diff = sby - sby_need)
```

Positive Werte von *sby_diff* bedeuten, dass zuviel Bereitschaftspersonal vorhergesagt wurde und negative Werte besagen, dass der Bedarf nicht durch die Vorhersage abgedeckt wurde. Generell sind daher positive Werte gegenüber negativen vorzuziehen.

```
print(paste0("RMSE: ", sqrt(mean(eval$sby_diff^2))))
```

```
[1] "RMSE: 356.535208413853"
```

```
print(paste0("MAE: ", mean(abs(eval$sby_diff))))
```

```
[1] "MAE: 328.516129032258"
```

```
print(paste0("Durchschnittswert der Abweichungen: ", mean(eval$sby_diff)))
```

```
[1] "Durchschnittswert der Abweichungen: 328.451612903226"
```

```
print(paste0("In ", round(nrow(eval[eval$sby_diff < 0,]) / nrow(eval) * 100, 2), "% der Fälle w
```

```
[1] "In 1.61% der Fälle wurde zuwenig Personal vorhergesagt"
```

```
print(paste0("Summe der mangelhaften negativen Vorhersagen: ", sum(eval[eval$sby_diff < 0,]$sby
```

```
[1] "Summe der mangelhaften negativen Vorhersagen: -2"
```

```
print(paste0("Durchschnittswert der mangelhaften negativen Vorhersagen: ", mean(eval[eval$sby_d
```

```
[1] "Durchschnittswert der mangelhaften negativen Vorhersagen: -2"
```

```
print(paste0("Größte negative Vorhersage: ", min(eval[eval$sby_diff < 0,]$sby_diff)))
```

```
[1] "Größte negative Vorhersage: -2"
```

Vergleich der Modelle

Modell	RMSE	MAE	MeanError	% low	sum low	mean low	max low
Baseline	185,94	164,48	108,64	26,23	1703,19	106,45	237,44
Aktuelles Vorgehen	160,05	125,59	46,38	47,54	5245	180,86	465
Advanced Version 1	190,31	153,36	88,47	27,12	1914	119,62	271
Advanced Version 2	311,91	270,59	266,05	8,93	127	25,4	88
Prophet (Q90)	356,43	328,61	328,03	1,61	18	18	18
Prophet (Q80)	299,07	268,85	264,82	3,23	125	62,5	81

Für den Vergleich wurde der Prophet Algorithmus in zwei Versionen ausgeführt. In beiden Fällen wurde mit dem Konfidenzintervall gearbeitet, einmals mit der 80% Quantille und einmal mit der 90% Quantille. Die obige Tabelle zeigt, dass das Prophet-Modell in Verwendung des 80% Konfidenzintervalls ähnliche RMSE, MAE und MeanError Werte aufweis, wie das Advanced Model (V2), jedoch deutlich weniger oft zu niedrig schätzt. Man sieht auch gut, welchen Einfluss die Parametrisierung des Konfidenzintervalls auf das Ergebnis hat. Bei dem 90% Konfidenzintervall wird überhaut nur einmal zu niedrig geschätzt, dafür sind eben die anderen Fehlerwerte deutlich schlechter. Die o.a. Grafik der Vorhersage lässt im Vergleich zu anderen Modellen jedoch erkennten, dass es Prophet, aufgrund der modellierten Volatilität schafft, in den Wintermonaten den Mindestbedarf wieder zu senken.

Der Quantillen-Parameters wird im deployten Modell flexibel setzbar sein. Der Defaultwert liegt bei

Speichern modelspezifischer Daten

```
# Speichern der Sampledaten (hier alles)
save(ts_sby_test, ts_sby_train, eval, call_pred_adv, ts_sby,
     file = "../../00_sample_data/03_for_modeling/prophet.rda")
```

Das Modell selbst wird im Ordner **03_deployment** des Modellverzeichnisses als **sby_model_prophet.R** Datei abgelegt.

```
rm(list = ls()) # nach Durchlauf alles löschen
```

Literaturverzeichnis

Grunsky, Georg. 2024. „Rettungsdienst Berliner Rotes Kreuz MachineLearning-Einsatz in der Bereitschaftsplanung - Use Case Analyse“, Dezember.

Taylor, Sean J, und Benjamin Letham. o. J. „Forecasting at scale“. https://doi.org/10.7287/peerj.preprints.3190v2.