Baseline Model

Georg Grunsky

2025-06-04

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."))
```

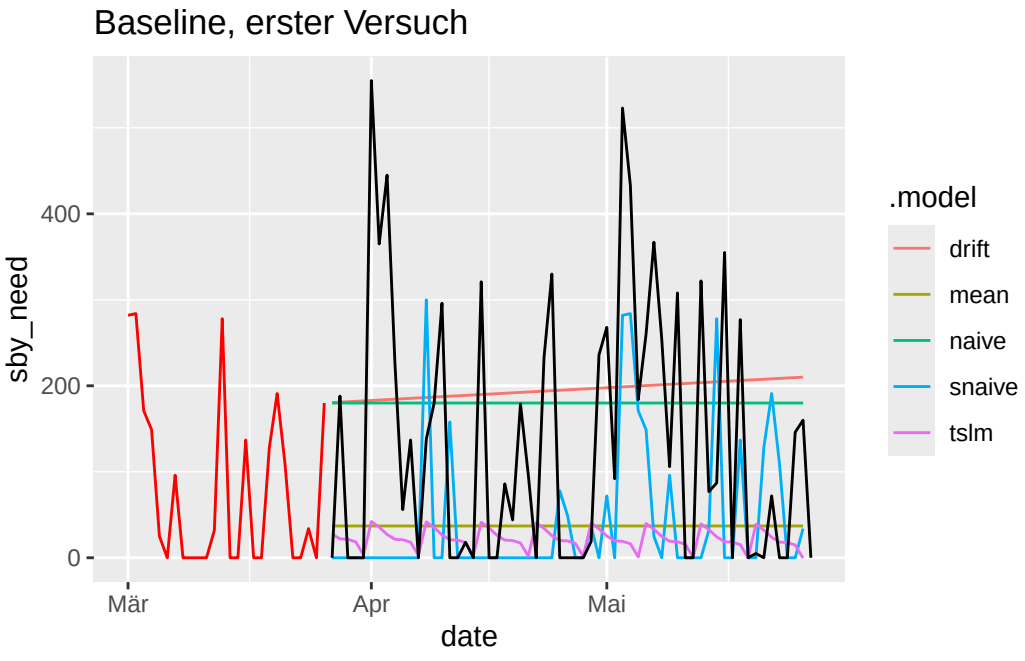
```
[1] " Die Benötigte Zeit für das Training betrug 0.07 Sekunden."
```

```
# 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

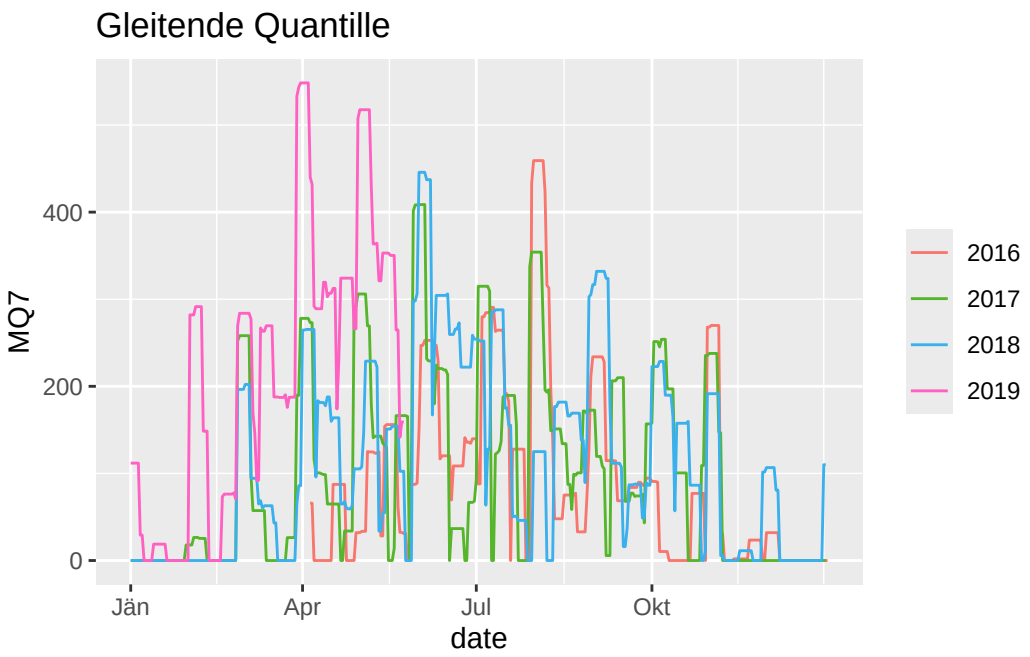
Der Vorhersageplot des ersten Verusches zeigt gut dessen Scheitern. Die hohen Ausschläge zu Beginn jedes Monats werden nicht durch die Modelle abgedeckt, die Peaks sind aber gleichzeitig auch zu unregeläßig um zB von einem *Snaïve*-Model “getroffen” zu werden. *TSLM* lässt schönt die wöchentliche Komponente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

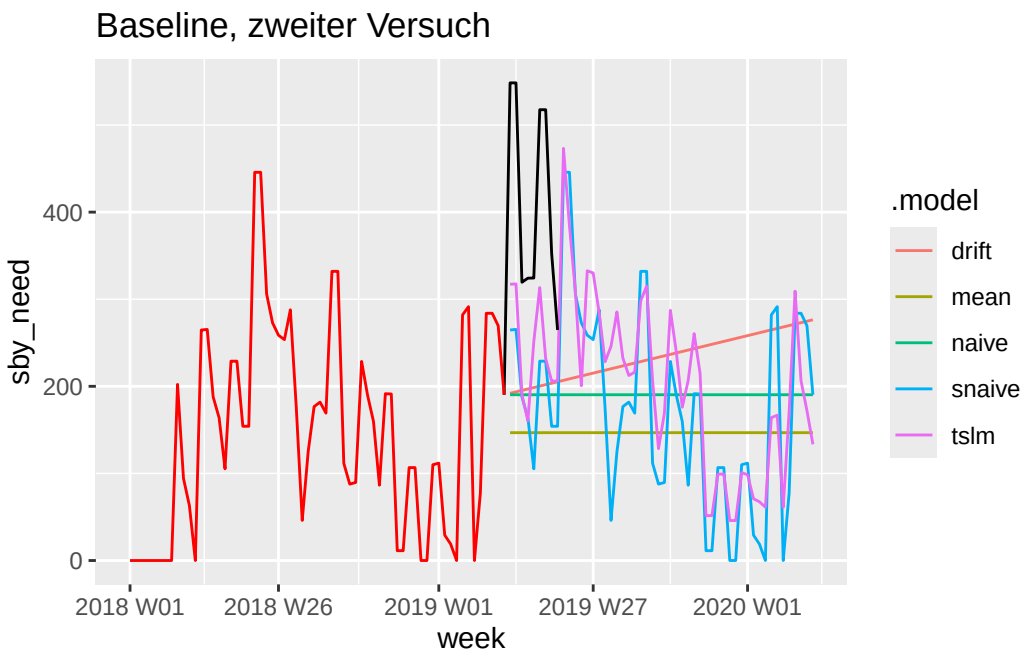
```
# Vorhersage
t <- system.time(
  progressr::with_progress(
    sby_basefc <- sby_basemodel %>%
      forecast(h = "12 months", level = c(99))
  ), gcFirst = TRUE
)

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")
```



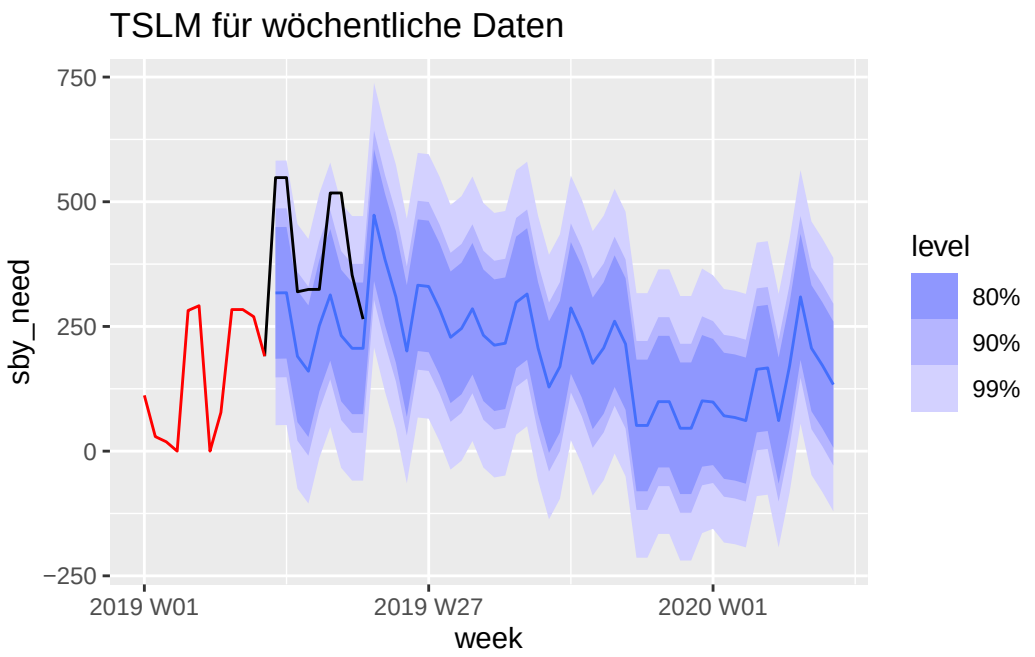
Nachdem im Saisonalitätsplot ein Aufwärtstrend klar erkennbar ist, wurde der Trainingszeitraum auf zwei etwa zwei Jahre und der Vorhersagezeitraum auf ein Jahr erweitert. Die Berechnungszeit für das Training und die Vorhersage kann auch bei diesem Ansatz vernachlässigt werden.

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 Konfidenz-intervall 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.

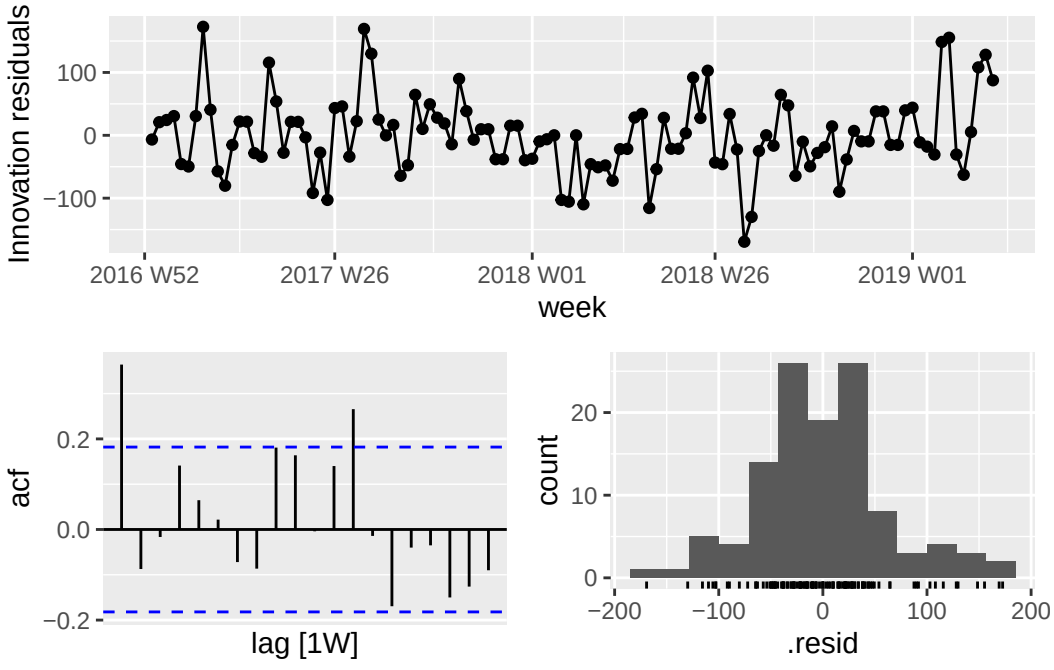
```
sby_basefc %>%
  filter(.model == "tslm") %>%
  autoplot(ts_sby_test,
           level = c(80, 90, 99)) +
  autolayer(ts_sby_train %>% filter(year(week) >= 2019), colour = "red" ) +
  labs(y = "sby_need",
       title = "TSLM für wöchentliche Daten")
```



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)
```

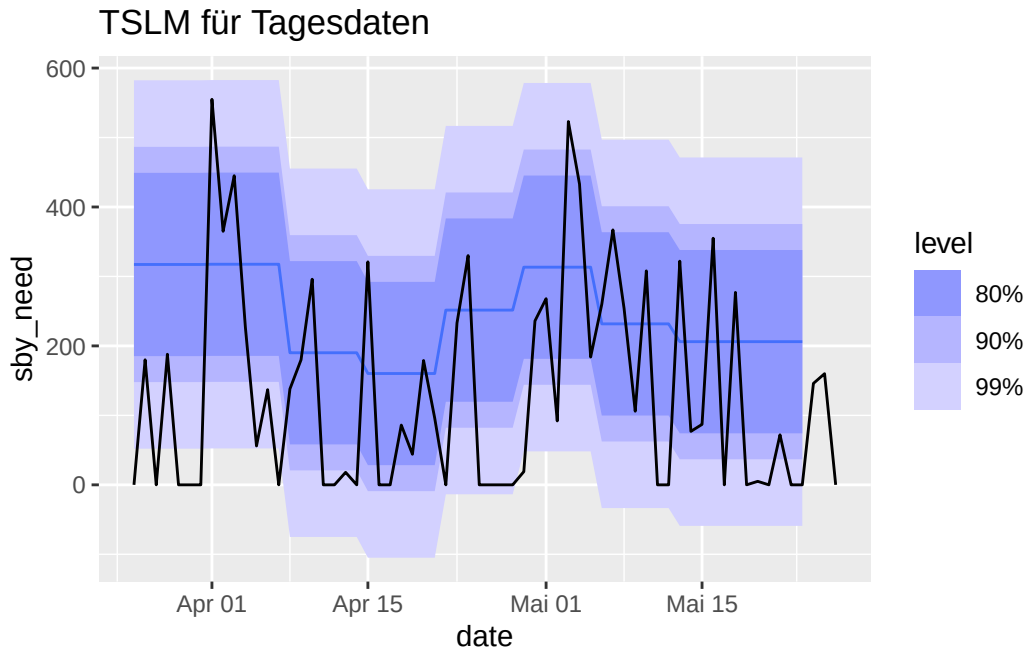


Für diesen Fall werden nachfolgend einige Bewertungsmethoden durchgeführt, die die Güte des *TSLM* belegen sollen. Die Anforderungen des Use Case sehen eine tageweise Vorhersage vor, das Modell berechnet allerdings wöchentliche Daten. Auch für die Vergleichbarkeit mit zukünftigen Modellen ist es sinnvoll 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) %>%
  right_join(ts_sby, by = "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 wurde zuw
```

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

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

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

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

```
[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 wurde zuw
```

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

```
print(paste0("Summe der mangelhaften negativen Vorhersagen (sum low): ", sum(eval[eval$act_diff < 0,]$a
```

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

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

```
[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/.