Prophet Model

Georg Grunsky

2025-06-04

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))
```

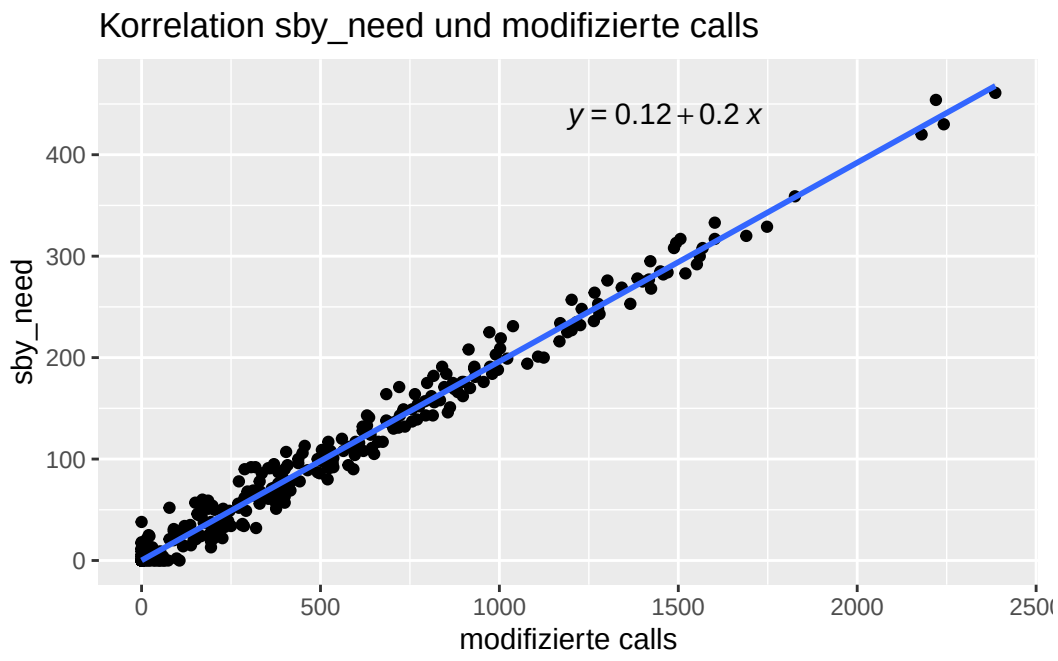
```r
ts_sby_test <- ts_sby %>%
  select(date, calls, MA2_8, n_duty, sby_need) %>%
  filter(!is.na(MA2_8)) %>%
  arrange(date) %>%
  slice((n() - 61):n())
```

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."
```

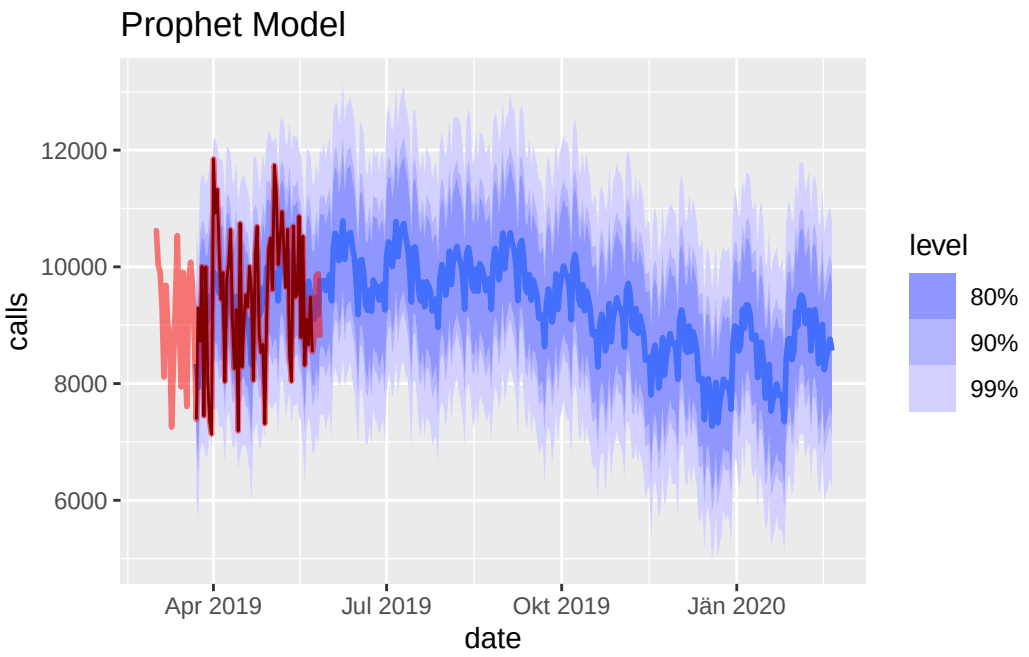
```
# Vorhersage
t <- system.time(
  progressr::with_progress(
    sby_fc_prophet <- sby_model_prophet %>%
      #forecast(h = "2 months") %>%
      forecast(h = "11 months")
  ), gcFirst = TRUE
)

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.

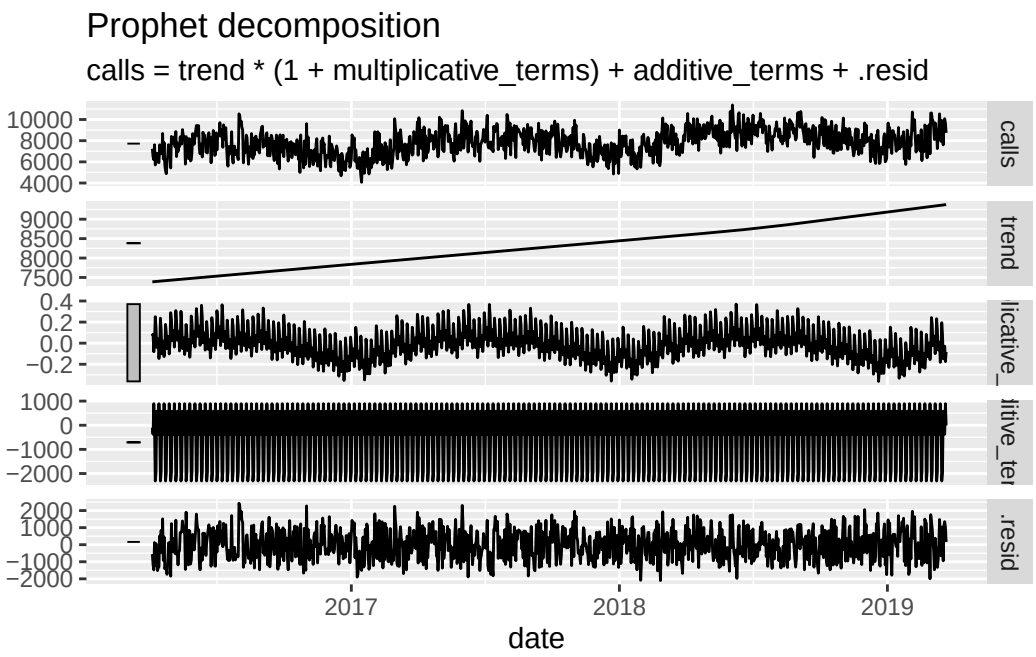
Das Modell schafft es daher mit Masse, die Spitzenwerte abzudecken. In einigen wenigen Fällen gelingt dies trotzdem nicht. Die Saisonalitäten sind deutlich erkennbar, auch wenn der zu erwartende Abfall in den Wintermonaten augenscheinlich zu wenig ausgeprägt ist.

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

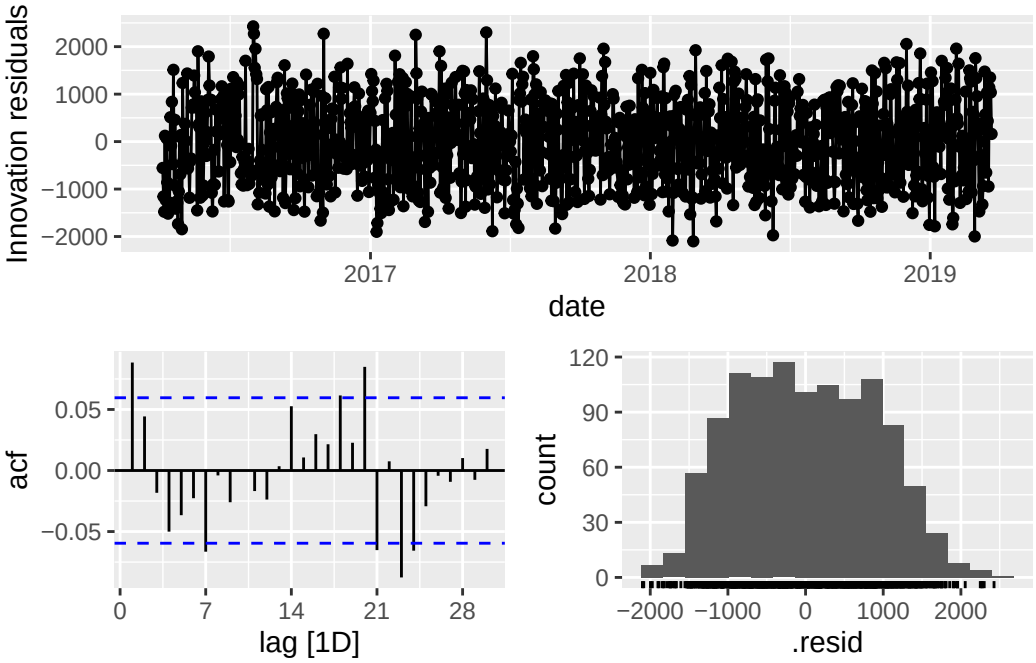
Gemäß den gesetzten Trainingsparametern unterscheidet der *prophet* Algorithmus additive und multiplikative Saisonalitäten. Der Unterschied liegt in der Variabilität der Saisonalitäten (multiplikativ ist hierbei variabler als additiv). Diese werden in der unten angeführten Decomposition des Modells gesondert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

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



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.

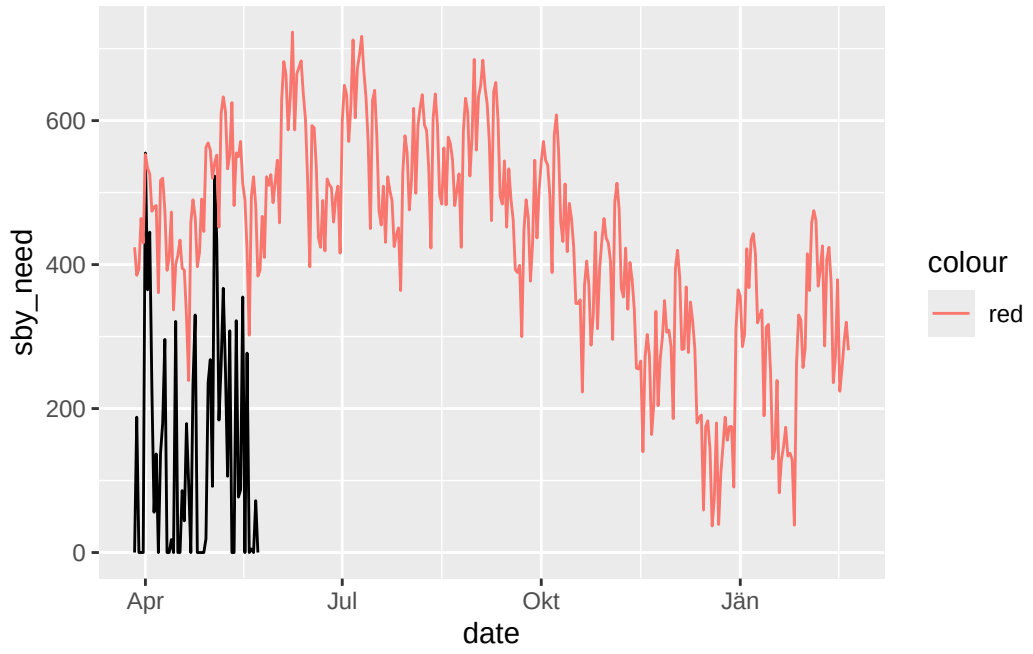
```
# vorgesehenes diensthabendes Personal
duty_pers <- 1800

# 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 wurde zuw
```

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

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

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

```
print(paste0("Durchschnittswert der mangelhaften negativen Vorhersagen: ", mean(eval[eval$sby_diff < 0,
```

```
[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 90% Konfidenzintervall.

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.