Vergleich verschiedener Prognosemethoden für die Absatzprognose von Lebensmitteln in einem FMCG-Unternehmen

Autor: Daniel Bich

Geschäftsverständnis

Das Ziel dieses Projekts ist die Verbesserung der Prognosequalität für den Absatz von mit Werbeaktionen beworbenen Lebensmittelartikeln in einem Unternehmen im FMCG-Markt. Das Projekt konzentriert sich auf die Untersuchung der Möglichkeiten zur Anwendung fortgeschrittener Data-Science-Methoden wie Autoregression, Saisonalanalyse, Random Forest/Boosting, Regression und neuronale Netze, um aktuelle Verkaufstrends zu berücksichtigen.

Die Forschungsarbeit konzentriert sich auf eine bestimmte Produktpalette des Vertriebsnetzes und verwendet Daten aus dem firmeneigenen Data Warehouse, einschließlich Informationen wie Produktpreis, täglicher Verkauf und Umsatz im Geschäft, Art der Werbeaktion und Preis vor der Aktion. Es ist wichtig, die Besonderheiten des FMCG-Marktes, den Wettbewerb sowie die Vorlieben und Bedürfnisse der Kunden zu verstehen, um das Werbeangebot besser an ihre Erwartungen anzupassen.

Experimente mit Verkaufsdaten der letzten 3 Jahre werden durchgeführt, um die Wirksamkeit verschiedener Methoden bei der Vorhersage des Absatzes von Werbeartikeln zu vergleichen. Die Ergebnisse dieser Studien können für Unternehmen von großer Bedeutung sein, die ihre Werbekampagnen optimieren und ihre Gewinne steigern möchten.

Der Implementierungsplan umfasst die Datenanalyse, die Entwicklung und das Testen verschiedener Vorhersagemodelle sowie die praktische Umsetzung des Modells mit den besten Parametern. Die Durchführung des Projekts erfolgt gemäß der CRISP-DM-Methodik.

Import

Bibliotheken importieren

```
%%capture
pip install pmdarima
```

```
# Grundlegende Bibliotheken
import os
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from matplotlib.figure import Figure
from matplotlib.ticker import MaxNLocator
```

```
import warnings

# Optionen
pd.set_option('display.max_rows', 50)
pd.set_option('display.max_columns', 50)
sns.color_palette('magma', as_cmap=True)
warnings.filterwarnings('ignore', category=FutureWarning)
%matplotlib inline

import functions as ff

# Clusteranalyse
from scipy.cluster.hierarchy import linkage, dendrogram
from scipy.cluster.vq import whiten, kmeans, vq
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import AgglomerativeClustering
from sklearn.cluster import KMeans
from sklearn.metrics import davies_bouldin_score

# Random
import random
random.seed(42)

#EDA
from statsmodels.tsa.seasonal import seasonal_decompose

# Metriken
from sklearn.metrics import mean_squared_error, mean_absolute_error,
r2_score

# Modellparameterabstimmung
from itertools import product
from sklearn.model_selection import ParameterGrid
from sklearn.model_selection import RandomizedSearchCV
import random
from random import randint

# Zeitreihen Cross Validation
from sklearn.model_selection import TimeSeriesSplit

# Basismodell
from sklearn.neighbors import KNeighborsRegressor

# SARIMAX
from pmdarima import auto_arima
from statsmodels.tsa.stattools import adfuller, kpss
from statsmodels.tsa.statespace.sarimax import SARIMAX

# Prophet
```

```
from prophet import Prophet
import holidays

# Boosting
import xgboost as xgb
from xgboost import XGBRFRegressor
import scipy.sparse as sp

# Daten-Transformation
from sklearn.model_selection import TimeSeriesSplit
from sklearn.preprocessing import OneHotEncoder, OrdinalEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import OneHotEncoder, OrdinalEncoder,
MinMaxScaler

# Neuronale Netze
import tensorflow as tf
from tensorflow.keras.layers import LSTM, Dense, Dropout, Input,
Embedding, Concatenate, Flatten
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping,
ReduceLROnPlateau
from keras.preprocessing.sequence import TimeseriesGenerator
from keras.utils import pad_sequences
from tensorflow.keras.preprocessing.text import Tokenizer
from functools import partial
from tensorflow.keras.utils import plot_model
```

Daten Import

```
# CSV Import
df = pd.read_csv('CSV/dane.csv', sep=',', parse_dates=['DATA'])
# Einschränkung der Daten
random_values = [random.randint(1, 238) for _ in range(150)]
df = df[df['MARKET_ID'].isin(random_values)]

# CSV Import
kalendarz = pd.read_csv('CSV/kalendarz.csv',sep=';',
parse_dates=['DATA'], date_parser=ff.dateparser)
del kalendarz[kalendarz.columns[10]]
HICP = pd.read_csv('CSV/HICP.csv', sep=';',parse_dates=['DATA'],
date_parser=ff.dateparser)
promo_ceny = pd.read_csv('CSV/PROMO_CENY.csv',
sep=';',parse_dates=['DATA'])
```

Datenbeschreibung

```
df.head()
```

```
          DATA  MARKET_ID  GRUPA_ID  ART_ID  OBROT  ILOSC  \
696 2019-09-02          2        19       1    0.0    0.0
697 2019-09-02          2         8       2    0.0    0.0
698 2019-09-02          2         8       3    0.0    0.0
699 2019-09-02          2         8       4    0.0    0.0
700 2019-09-02          2         8       5    0.0    0.0

                           NAZWA
696  FCHKWROVC QD JROULRGU FDSYVR
697                JHSDL ERSCR
698            JHSDL FCHS QH YRFD
699                JHSDL WDSGU
700              JHSDL KRLGRSDZFH
```

Der DataFrame **df** bildet die Hauptachse und enthält Daten über den Verkauf verschiedener Lebensmittel in verschiedenen Märkten an einem bestimmten Tag. Die Spalten enthalten folgende Informationen:

DATA: Verkaufsdatum (z. B. 2020-02-13), *MARKET_ID*: Eindeutige Markt-ID, an dem der Verkauf stattgefunden hat (z. B. 1, 4, 6), *GRUPA_ID*: Produktgruppenkennung, zu der der Artikel gehört (z. B. 19, 8), *ART_ID*: Eindeutige Artikel-ID (z. B. 1, 2), *OBROT*: Gesamter Verkaufswert des Artikels in PLN (z. B. 3,31, 22,21), *ILOSC*: Anzahl der verkauften Stücke des Artikels (z. B. 1,00, 2,92), *NAZWA*: Kodierter Artikelname (z. B. 'FCHKWROVC QD JROULRGU FDSYVR').

```
df.describe().T
```

```
                count        mean         std      min    25%    50%
75%  \
MARKET_ID  89277712.0  119.642269   69.127846     2.00   58.0  118.0
177.0
GRUPA_ID   89277712.0   13.910693    8.134427     1.00    5.0   14.0
22.0
ART_ID     89277712.0  348.548119  200.995705     1.00  175.0  349.0
522.0
OBROT      89277712.0   17.559285   89.004770  -699.20    0.0    0.0
0.0
ILOSC      89277712.0    4.080592   26.660253 -1024.34    0.0    0.0
0.0

                max
MARKET_ID    237.00
GRUPA_ID      26.00
ART_ID       697.00
OBROT      10973.85
ILOSC       6003.23
```

```
df.shape
```

```
(89277712, 7)
```

Beobachtung: Die Spalten 'OBROT' und 'ILOSC' enthalten negative Werte - eine Datenbereinigung ist erforderlich.

```
kalendarz.head()
```

```
        DATA  Nr_dn_tyg  Nr_dn_mies  Nr_mies  Nr_tyg  Nr_rok
Dn_handlowy  \
0 2019-01-01          2           1        1       1    2019
0
1 2019-01-02          3           2        1       1    2019
1
2 2019-01-03          4           3        1       1    2019
1
3 2019-01-04          5           4        1       1    2019
1
4 2019-01-05          6           5        1       1    2019
1

   Hot_day  Hot_day_Xmass  Hot_day_Wlkn
0        1              0             0
1        0              0             0
2        0              0             0
3        0              0             0
4        0              0             0
```

```
kalendarz.describe().T
```

```
                count         mean        std     min     25%     50%
75%  \
Nr_dn_tyg      1826.0     4.001643   1.999862     1.0     2.0     4.0
6.0
Nr_dn_mies     1826.0    15.727820   8.801735     1.0     8.0    16.0
23.0
Nr_mies        1826.0     6.523549   3.449478     1.0     4.0     7.0
10.0
Nr_tyg         1826.0    26.615553  15.065085     1.0    14.0    27.0
40.0
Nr_rok         1826.0  2020.999452   1.414407  2019.0  2020.0  2021.0
2022.0
Dn_handlowy    1826.0     0.855422   0.351772     0.0     1.0     1.0
1.0
Hot_day        1826.0     0.035597   0.185334     0.0     0.0     0.0
0.0
Hot_day_Xmass  1826.0    -0.217963   1.752220   -14.0     0.0     0.0
0.0
Hot_day_Wlkn   1826.0    -0.210843   1.766325   -14.0     0.0     0.0
0.0

                max
Nr_dn_tyg       7.0
Nr_dn_mies     31.0
```

```
Nr_mies          12.0
Nr_tyg           53.0
Nr_rok         2023.0
Dn_handlowy       1.0
Hot_day           1.0
Hot_day_Xmass     7.0
Hot_day_Wlkn      7.0
```

Für das Projekt wird ein externer DataFrame namens **kalendarz** verwendet, der Informationen wie Wochentagsnummern, Monatsnummern, Jahresnummern, ob ein bestimmter Tag ein Geschäftstag war, und andere spezielle Tagkennzeichnungen enthält. Die Spalten enthalten folgende Informationen:

DATA: Datum (z. B. 2019-01-01, 2019-01-02), Nr_dn_tyg: Wochentagsnummer (z. B. 2 - Dienstag, 3 - Mittwoch), *Nr_dn_mies*: Monatstagnummer (z. B. 1, 2), *Nr_mies*: Monatsnummer (z. B. 1 - Januar), *Nr_tyg*: Wochennummer im Jahr (z. B. 1), *Nr_rok*: Jahr (z. B. 2019), *Dn_handlowy*: Kennzeichnung, ob der Tag ein Geschäftstag war (1 - ja, 0 - nein), *Hot_day*: Kennzeichnung, ob der Tag ein besonderer Tag war (1 - ja, 0 - nein), *Hot_day_Xmass*: Kennzeichnung, ob der Tag ein Feiertag war (Skala von -14 bis 7, wobei -14 14 Tage vor dem Feiertag bedeutet), *Hot_day_Wlkn*: Kennzeichnung, ob der Tag ein arbeitsfreier Tag war (Skala von -14 bis 7, wobei -14 14 Tage vor dem Feiertag bedeutet).

```
promo_ceny.head()
```

```
        DATA  ART_ID  CENA_NP  CENA_AP  PRZECENA  PROMO_KOD
0 2019-09-02       1     3.49     3.49       0.0          0
1 2019-09-03       1     3.49     3.49       0.0          0
2 2019-09-04       1     3.49     3.49       0.0          0
3 2019-09-05       1     3.49     3.49       0.0          0
4 2019-09-06       1     3.49     3.49       0.0          0
```

Der DataFrame **promo_ceny** enthält Daten zu Preisen und Produktaktionen zu einem bestimmten Zeitpunkt. Die Spalten enthalten folgende Informationen:

DATA: Datum, auf das sich der entsprechende Datensatz bezieht (z. B. 2019-06-06, 2019-06-07), *ART_ID*: Eindeutige Produktkennung (z. B. 362), *CENA_NP*: Normaler Preis, bei Aktionen der letzte Preis vor der Aktion (z. B. 1,99), *CENA_AP*: Aktionspreis, wenn eine Aktion stattfindet, andernfalls der Wert in der Spalte CENA_NP (z. B. 2,99), *PRZECENA*: Prozentsatz der Preissenkung während einer Aktion (z. B. 0,333 = 33,3%), *PROMO_KOD*: Aktionscode, der mit dem jeweiligen Produkt verbunden ist (z. B. 13).

```
promo_ceny.describe().T
```

```
              count        mean         std  min     25%     50%
75%  \
ART_ID     681470.0  320.239845  187.725068  1.0  156.00  318.00
483.00
CENA_NP    681470.0    7.269196    5.321974  0.1    4.49    5.99
8.99
CENA_AP    681470.0    7.211979    5.315129  0.1    4.49    5.99
```

```
8.56
PRZECENA   681470.0    0.007850    0.053133  0.0    0.00    0.00
0.00
PROMO_KOD  681470.0    0.621079    7.751325  0.0    0.00    0.00
0.00

               max
ART_ID     697.000
CENA_NP     55.920
CENA_AP     55.920
PRZECENA     0.841
PROMO_KOD  154.000
```

Beobachtung: In den Tabellen *kalendarz* und *promo_ceny* gibt es Spalten, die kategoriale Variablen darstellen - eine Anpassung des Datentyps ist erforderlich.

```
HICP.head(2)
```

```
        DATA   HICP
0 2019-01-01  103.0
1 2019-01-02  103.0
```

Der DataFrame **HICP** enthält Daten zum harmonisierten Verbraucherpreisindex (HICP) zu einem bestimmten Zeitpunkt. Die Spalten enthalten folgende Informationen:

DATA: Datum, auf das sich der entsprechende Datensatz bezieht (z. B. 2019-01-01, 2019-01-02), *HICP*: Wert des harmonisierten Verbraucherpreisindex (z. B. 103,0).

HICP ist ein Maß für die Inflation, das von der Europäischen Union und anderen internationalen Organisationen verwendet wird, um die Preise von Konsumgütern und -dienstleistungen zwischen Ländern zu überwachen und zu vergleichen. Dieser DataFrame kann zur Analyse von Inflationsänderungen im Laufe der Zeit oder zum Vergleich von Inflationsniveaus zwischen verschiedenen Perioden verwendet werden.

Daten Preprocessing und feature engineering

Preprocessing

```
# Nullsetzen von Zeilen, bei denen der Verkaufswert Null oder negativ
ist (Datenfehler)
df.loc[df['ILOSC'] <= 0, 'ILOSC'] = 0
df.loc[df['ILOSC'] <= 0, 'OBROT'] = 0
# Nullsetzen von Zeilen, bei denen der Umsatzwert Null oder negativ
ist (Datenfehler)
df.loc[df['OBROT'] <= 0, 'OBROT'] = 0
df.loc[df['OBROT'] <= 0, 'ILOSC'] = 0
```

```
df.head()
```

```
          DATA  MARKET_ID  GRUPA_ID  ART_ID  OBROT  ILOSC  \
696 2019-09-02          2        19       1    0.0    0.0
697 2019-09-02          2         8       2    0.0    0.0
```

```
698 2019-09-02          2         8       3    0.0    0.0
699 2019-09-02          2         8       4    0.0    0.0
700 2019-09-02          2         8       5    0.0    0.0

                           NAZWA
696  FCHKWROVC QD JROULRGU FDSYVR
697                  JHSDL ERSCR
698            JHSDL FCHS QH YRFD
699                  JHSDL WDSGU
700              JHSDL KRLGRSDZFH
```

```
# Daten Sortierung
df.sort_values(['ART_ID', 'MARKET_ID', 'DATA'], inplace=True)

# Maske zur Vermeidung von Divisionen durch Null
mask = df['ILOSC'] != 0
# Berechnung des durchschnittlichen Verkaufspreises pro Markt - Preise
pro Markt unter Berücksichtigung von Preissenkung oder Abschriften
df['CENA_MARKET'] = np.round(np.divide(df['OBROT'][mask], df['ILOSC']
[mask]),4)

df.isnull().sum()
```

```
DATA                  0
MARKET_ID             0
GRUPA_ID              0
ART_ID                0
OBROT                 0
ILOSC                 0
NAZWA                 0
CENA_MARKET    68247708
dtype: int64
```

Beobachtung: Der DataFrame enthält NaN-Werte für *'CENA_MARKET'* aufgrund von Kombinationen von Artikel und Tag für nicht-handelsübliche Tage, an denen kein Verkauf stattgefunden hat.

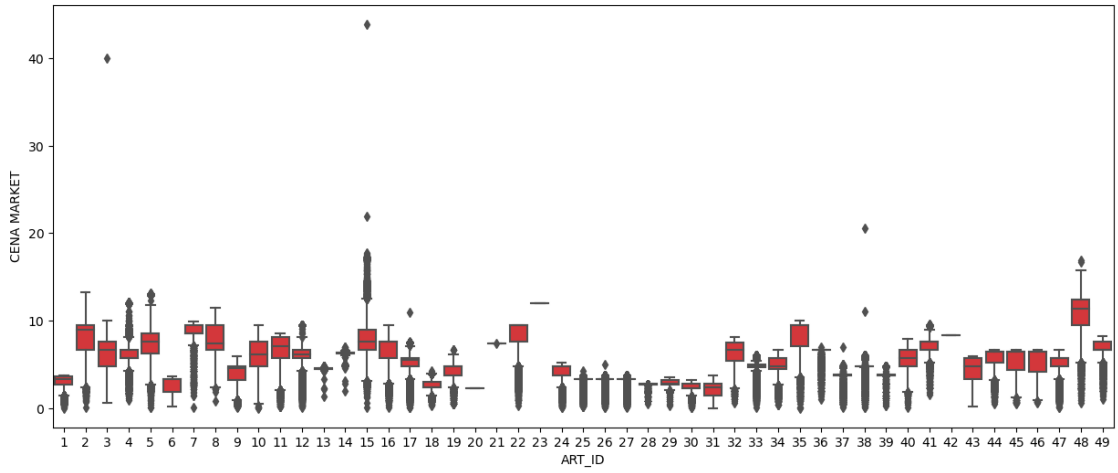
```
df.describe().T
```

```
                  count        mean         std   min      25%
50%  \
MARKET_ID    89277712.0  119.642269   69.127846  2.00   58.000
118.0000
GRUPA_ID     89277712.0   13.910693    8.134427  1.00    5.000
14.0000
ART_ID       89277712.0  348.548119  200.995705  1.00  175.000
349.0000
OBROT        89277712.0   17.559483   89.004489  0.00    0.000
0.0000
ILOSC        89277712.0    4.080637   26.659994  0.00    0.000
0.0000
```

```
CENA_MARKET  21030004.0    5.932942    3.799394  0.01    3.515
4.7546

                75%       max
MARKET_ID     177.0    237.00
GRUPA_ID       22.0     26.00
ART_ID        522.0    697.00
OBROT           0.0  10973.85
ILOSC           0.0   6003.23
CENA_MARKET     7.6   1184.50
```

Beobachtung: Die Spalte 'CENA_MARKET' enthält Ausreißerwerte (fehlerhafte Werte) - es ist erforderlich, die Daten zu bereinigen.

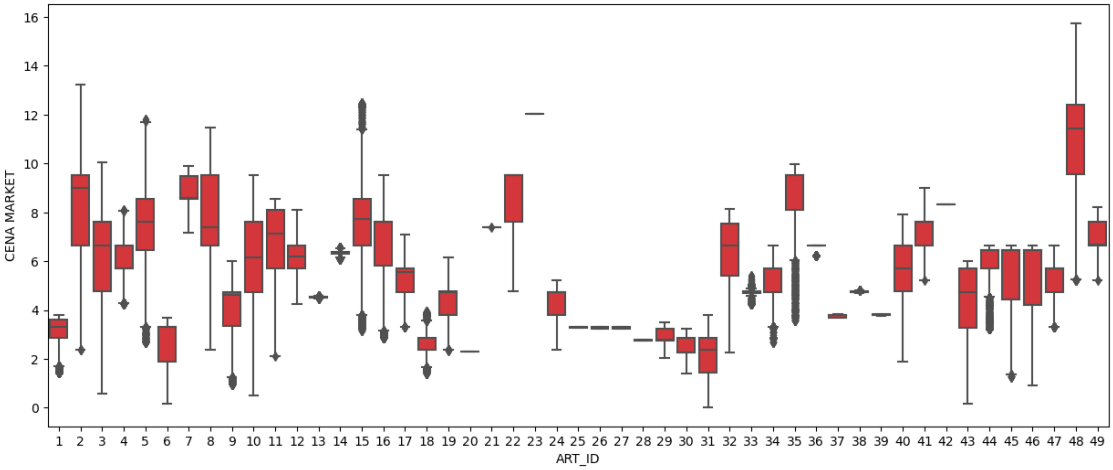
```
# Erstellung einer Teilmenge von Daten auf der Grundlage der Bedingung
ART_ID < 50
filtered_df = df[df['ART_ID'] < 50]

# Erstellen eines Boxplots
plt.figure(figsize=(15, 6))
sns.boxplot(data=filtered_df, x='ART_ID', y='CENA_MARKET',
palette=sns.color_palette(colors *
len(filtered_df['ART_ID'].unique())))

# Hinzufügen von Achsenbeschriftungen
plt.xlabel('ART_ID')
plt.ylabel('CENA MARKET')

# Chart-Anzeige
plt.show()
```



Beobachtung: Es gibt Artikel mit extrem hohen Preisen - dies ist ein Datenfehler und die entsprechenden Beobachtungen werden entfernt.

Die Konvertierung von Ausreißern auf Mittelwert unter Verwendung der IQR-Methode:

```
# Erstellen einer Maske, um Nullen aus der Berechnung auszuschließen
mask = df['CENA_MARKET'] != 0

# Berechnung von Quartilen, Interquartilen und Ausreißergrenzen
q1 = df[mask].groupby('ART_ID')['CENA_MARKET'].quantile(0.25)
q3 = df[mask].groupby('ART_ID')['CENA_MARKET'].quantile(0.75)
iqr = q3 - q1

upper_bound = q3 + 1.5 * iqr
lower_bound = q1 - 1.5 * iqr

# Berechnung der Durchschnittswerte für jede ART_ID
mean_values = df[mask].groupby('ART_ID')['CENA_MARKET'].mean()

# Ersetzung der Werte außerhalb der Ausreißer durch die
Durchschnittswerte für die entsprechende ART_ID
df['CENA_MARKET'] = df.groupby('ART_ID')['CENA_MARKET'].transform(
    lambda x: x.mask(x > upper_bound.loc[x.name], mean_values[x.name])
              .mask(x < lower_bound.loc[x.name], mean_values[x.name]))

# Erstellung einer Teilmenge von Daten auf der Grundlage der Bedingung
ART_ID < 50
filtered_df = df[df['ART_ID'] < 50]

# Erstellen eines Boxplots
plt.figure(figsize=(15, 6))
sns.boxplot(data=filtered_df, x='ART_ID', y='CENA_MARKET',
palette=sns.color_palette(colors *
len(filtered_df['ART_ID'].unique())))

# Hinzufügen von Achsenbeschriftungen
plt.xlabel('ART_ID')
plt.ylabel('CENA MARKET')

# Chart-Anzeige
plt.show()
```




```
df.describe().T
```

```
                 count        mean         std     min      25%
50%  \
MARKET_ID    89277712.0  119.642269   69.127846  2.0000   58.000
118.0000
GRUPA_ID     89277712.0   13.910693    8.134427  1.0000    5.000
14.0000
ART_ID       89277712.0  348.548119  200.995705  1.0000  175.000
349.0000
OBROT        89277712.0   17.559483   89.004489  0.0000    0.000
0.0000
ILOSC        89277712.0    4.080637   26.659994  0.0000    0.000
0.0000
CENA_MARKET  21030004.0    5.959987    3.713719  0.0193    3.665
4.7609

                75%       max
MARKET_ID     177.0    237.00
GRUPA_ID       22.0     26.00
ART_ID        522.0    697.00
OBROT           0.0  10973.85
ILOSC           0.0   6003.23
CENA_MARKET     7.6     53.00
```

```
df.isnull().sum()
```

```
DATA                  0
MARKET_ID             0
GRUPA_ID              0
ART_ID                0
OBROT                 0
ILOSC                 0
NAZWA                 0
CENA_MARKET    68247708
dtype: int64
```

```
df.shape
```

```
(89277712, 8)
```

Tabellen verknüpfen

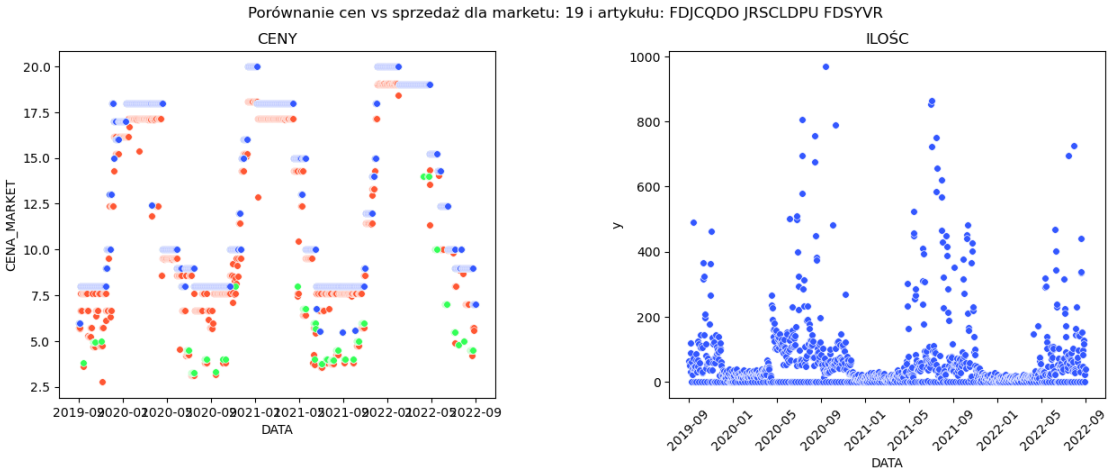
```
# Tabellen verknüpfen
df = pd.merge(df, kalendarz, how='left', on='DATA')
df = pd.merge(df, promo_ceny, how='left', on = ['DATA','ART_ID'])

# Ergänzen von Preisen
df['CENA_MARKET'] = df['CENA_MARKET'].fillna(df['CENA_AP'])
df['CENA_MARKET'] = np.where(df['CENA_MARKET'] > df['CENA_AP'],
df['CENA_AP'], df['CENA_MARKET'])

df.isnull().sum()
```

```
DATA                   0
MARKET_ID              0
GRUPA_ID               0
ART_ID                 0
OBROT                  0
ILOSC                  0
NAZWA                  0
CENA_MARKET      9559134
Nr_dn_tyg              0
Nr_dn_mies             0
Nr_mies                0
Nr_tyg                 0
Nr_rok                 0
Dn_handlowy            0
Hot_day                0
Hot_day_Xmass          0
Hot_day_Wlkn           0
CENA_NP          9559134
CENA_AP          9559134
PRZECENA         9559134
PROMO_KOD        9559134
dtype: int64
```

```
# Zeilen löschen, für die kein Preis gefunden wurde (NaN)
df.dropna(how='any', inplace=True)

# Tabellen verknüpfen
df = pd.merge(df, HICP, how='left', on = 'DATA')

# Zuordnung von Datentypen zu einzelnen Spalten
int32 = ['MARKET_ID', 'GRUPA_ID', 'ART_ID', 'PROMO_KOD']
df[int32] = df[int32].astype('int32')

category = ['MARKET_ID', 'GRUPA_ID', 'ART_ID', 'Nr_dn_tyg',
'Nr_dn_mies', 'Nr_mies', 'Nr_tyg',
            'Nr_rok', 'Dn_handlowy', 'Hot_day', 'Hot_day_Xmass',
```

```
'Hot_day_Wlkn', 'PROMO_KOD']
df[category] = df[category].astype('category')

float32 =['ILOSC', 'CENA_MARKET', 'CENA_AP', 'CENA_NP', 'PRZECENA',
'HICP']
df[float32] = df[float32].astype('float32')

df.info()

<class 'pandas.core.frame.DataFrame'>
Int64Index: 79718578 entries, 0 to 79718577
Data columns (total 22 columns):
 #   Column         Dtype
---  ------         -----
 0   DATA           datetime64[ns]
 1   MARKET_ID      category
 2   GRUPA_ID       category
 3   ART_ID         category
 4   OBROT          float64
 5   ILOSC          float32
 6   NAZWA          object
 7   CENA_MARKET    float32
 8   Nr_dn_tyg      category
 9   Nr_dn_mies     category
 10  Nr_mies        category
 11  Nr_tyg         category
 12  Nr_rok         category
 13  Dn_handlowy    category
 14  Hot_day        category
 15  Hot_day_Xmass  category
 16  Hot_day_Wlkn   category
 17  CENA_NP        float32
 18  CENA_AP        float32
 19  PRZECENA       float32
 20  PROMO_KOD      category
 21  HICP           float32
dtypes: category(13), datetime64[ns](1), float32(6), float64(1),
object(1)
memory usage: 5.3+ GB

# Umbenennung der Zielspalte
df.rename(columns = {'ILOSC':'y'}, inplace = True)

# Ändern der Reihenfolge der Spalten
new_order = ['DATA', 'MARKET_ID', 'GRUPA_ID', 'ART_ID', 'NAZWA',
'CENA_NP', 'CENA_AP', 'PRZECENA', 'CENA_MARKET', 'OBROT', 'PROMO_KOD',
'Nr_dn_tyg', 'Nr_dn_mies', 'Nr_mies', 'Nr_tyg', 'Nr_rok',
'Dn_handlowy', 'Hot_day', 'Hot_day_Xmass', 'Hot_day_Wlkn', 'HICP',
'y'] # Nowa kolejność kolumn
df = df.reindex(columns=new_order)
```

```
# Preisdiagramm
ff.plot_cena(df, market=19, artykul=158)
```



Auf beiden Diagrammen (Preise und Umsatz) kann man eine jährliche Saisonalität für den Artikel erkennen. Die Interpretation des linken Diagramms lautet wie folgt: Der höchste Preis entspricht dem Normalpreis (CENA_NP), der zentral festgelegt wird. In Aktionszeiträumen ist der Aktionspreis (CENA_AP) aufgrund von Werbeaktionen niedriger. Der niedrigste Preis ist der Marktpreis (CENA_MARKET), der Rabatte auf Marktebene berücksichtigt (und/oder Abschreibungen).

Die bereinigten Daten sehen wie folgt aus:

```
df.head(3)
```

```
        DATA MARKET_ID GRUPA_ID ART_ID                          NAZWA
CENA_NP  \
0 2019-09-02         2       19      1  FCHKWROVC QD JROULRGU FDSYVR
3.49
1 2019-09-03         2       19      1  FCHKWROVC QD JROULRGU FDSYVR
3.49
2 2019-09-04         2       19      1  FCHKWROVC QD JROULRGU FDSYVR
3.49

   CENA_AP  PRZECENA  CENA_MARKET  OBROT PROMO_KOD Nr_dn_tyg
Nr_dn_mies  \
0     3.49       0.0         3.49    0.0         0         1
2
1     3.49       0.0         3.49    0.0         0         2
3
2     3.49       0.0         3.49    0.0         0         3
4

  Nr_mies Nr_tyg Nr_rok Dn_handlowy Hot_day Hot_day_Xmass Hot_day_Wlkn
\
0       9     36   2019           1       0             0            0
```

```
1       9     36   2019           1       0             0           0

2       9     36   2019           1       0             0           0


         HICP    y
0  105.199997  0.0
1  105.199997  0.0
2  105.199997  0.0
```

```
df.describe().T
```

```
                  count       mean        std         min     25%
50%  \
CENA_NP      79718578.0   3.880406   5.734397    0.100000    4.49
5.990000
CENA_AP      79718578.0   3.850663   5.692187    0.100000    4.49
5.990000
PRZECENA     79718578.0   0.007868   0.052690    0.000000    0.00
0.000000
CENA_MARKET  79718578.0   3.803628   5.621907    0.019300    3.99
5.990000
OBROT        79718578.0  19.665058  93.969695    0.000000    0.00
0.000000
HICP         79718578.0  52.634510  41.521702  105.199997  109.00
113.199997
y            79718578.0   4.256569  27.676815    0.000000    0.00
0.000000

                   75%           max
CENA_NP        8.990000     55.919998
CENA_AP        8.560000     55.919998
PRZECENA       0.000000      0.841000
CENA_MARKET    7.990929     55.919998
OBROT          3.490000  10973.850000
HICP         118.599998    131.399994
y              1.000000   6003.229980
```

Clustering

Vorbereitung von Daten

```
# Copy von DataFrame
df_temp = df[['DATA', 'MARKET_ID', 'ART_ID', 'y', 'CENA_MARKET',
'PROMO_KOD']].copy()
df_temp = df_temp.loc[df['DATA'] >= '2021-09-01']

# Erstellung einer Maske
mask = df['PROMO_KOD'] == 0
```

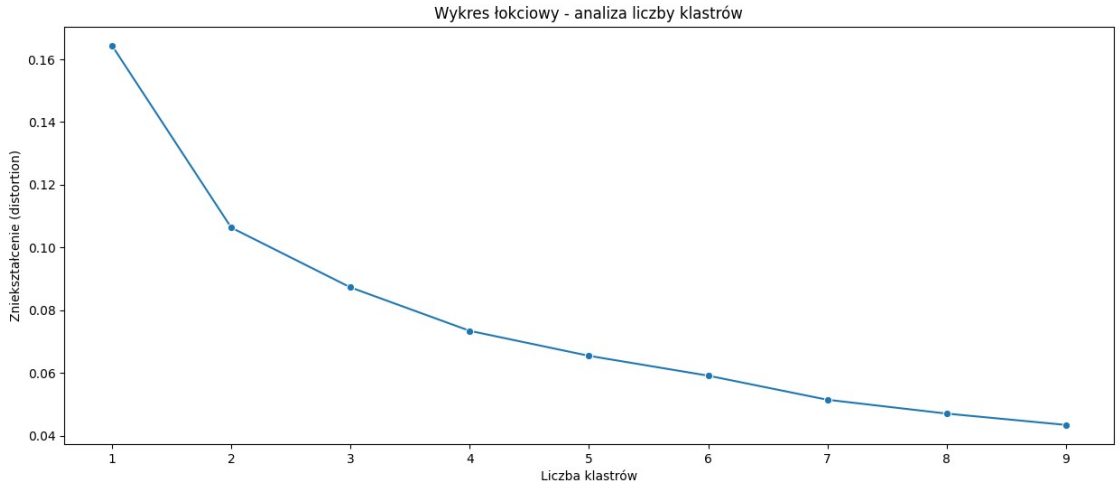
```
# Trennung der Daten in Bezug auf die Maske
df_temp.loc[mask, 'ILOSC_NIEPROMO'] = df_temp.loc[mask, 'y']
df_temp.loc[mask, 'CENA_NIEPROMO'] = df_temp.loc[mask, 'CENA_MARKET']
df_temp.loc[~mask, 'ILOSC_PROMO'] = df_temp.loc[~mask, 'y']
df_temp.loc[~mask, 'CENA_PROMO'] = df_temp.loc[~mask, 'CENA_MARKET']

# Entfernung von unnötigen Spalten
df_temp.drop(['DATA', 'ART_ID', 'y', 'CENA_MARKET'], axis=1,
inplace=True)

# Gruppierung und Berechnung von Durchschnittswerten
df_k = df_temp.groupby(['MARKET_ID'])[['ILOSC_NIEPROMO',
'ILOSC_PROMO', 'CENA_NIEPROMO', 'CENA_PROMO']].agg(np.nanmean)

# Min-Max-Normalisierung
df_k['ILOSC_NIEPROMO'] = df_k['ILOSC_NIEPROMO'] /
df_k['ILOSC_NIEPROMO'].max()
df_k['ILOSC_PROMO'] = df_k['ILOSC_PROMO'] / df_k['ILOSC_PROMO'].max()
df_k['CENA_NIEPROMO'] = df_k['CENA_NIEPROMO'] /
df_k['CENA_NIEPROMO'].max()
df_k['CENA_PROMO'] = df_k['CENA_PROMO'] / df_k['CENA_PROMO'].max()

df_k2 = df_k.copy()
```

Elbow method

Die Ellbogenmethode ("elbow method") ist eine beliebte Technik zur Auswahl der optimalen Anzahl von Clustern in der Clusteranalyse. Ihr Name leitet sich von der Form des Graphen ab, der an eine Ellbogenbeugung erinnert. Diese Methode basiert auf der Bewertung der Differenz der innerhalb der Cluster liegenden Summe der Quadrate (WCSS - "Within-Cluster Sum of Squares") für verschiedene Clusteranzahlen.

```
distortions = []
num_clusters = range(1, 10)

# Berechnung der Verzerrungen mit dem k-Means-Algorithmus für
verschiedene Clusteranzahlen
for i in num_clusters:
    cluster_centers, distortion = kmeans(df_k[['ILOSC_NIEPROMO',
'ILOSC_PROMO', 'CENA_NIEPROMO', 'CENA_PROMO']], i)
    distortions.append(distortion)

# Erstellen eines Datenrahmens mit zwei Listen: num_clusters und
distortions
elbow_plot = pd.DataFrame({'num_clusters': num_clusters,
'distortions': distortions})

# Erzeugen eines Liniendiagramms, das die Anzahl der Cluster in
Abhängigkeit von den Verzerrungen darstellt
plt.figure(figsize=(15, 6))
```

```
sns.lineplot(data=elbow_plot, x='num_clusters', y='distortions',
marker='o')
plt.xticks(num_clusters)
plt.title('Wykres łokciowy - analiza liczby klastrów')
plt.xlabel('Liczba klastrów')
plt.ylabel('Zniekształcenie (distortion)')
plt.show()
```



Kalinowski-Harabasz-Index

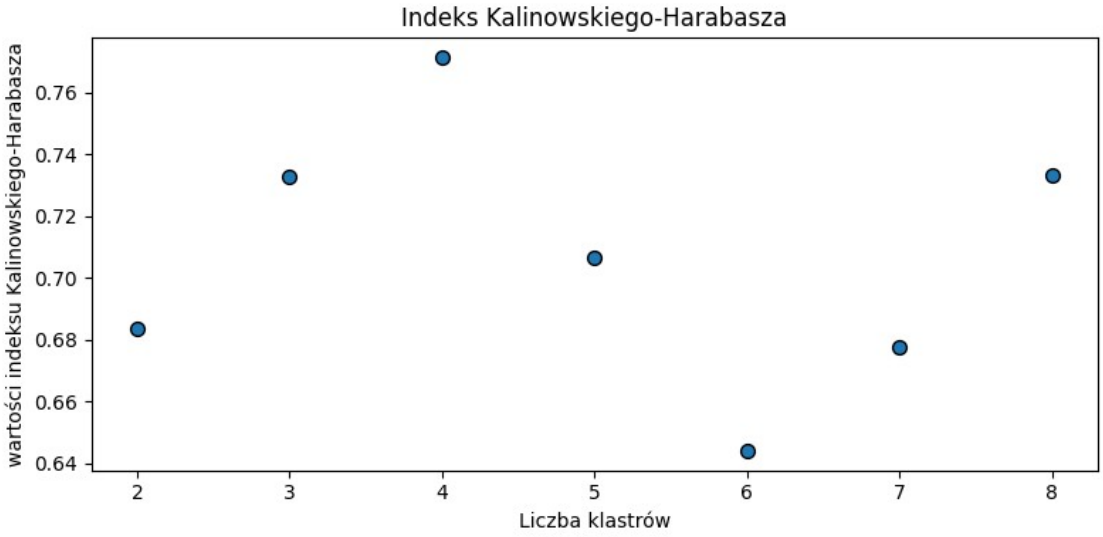
Der Kalinowski-Harabasz-Index, auch als Davies-Bouldin-Index (DBI) bekannt, ist ein Maß für die Bewertung der Clusterqualität in der Clusteranalyse. Dieser Index wurde 1979 von Davis und Bouldin vorgeschlagen und später von Kalinowski und Harabasz erweitert.

Der Kalinowski-Harabasz-Index vergleicht die interne Ähnlichkeit der Cluster mit der externen Diversität zwischen den Clustern. Zur Berechnung des DBI wird zunächst für jeden Cluster der Durchschnittsabstand zwischen den Punkten im Cluster berechnet und anschließend der Abstand zwischen den Clusterzentroiden ermittelt. Der endgültige DBI-Wert ist der Durchschnitt der relativen Abstandsmaße zwischen den Clustern.

Je kleiner der Wert des Kalinowski-Harabasz-Index (DBI), desto besser die Clusterqualität.

```
# Initialisierung der Listen zur Speicherung der Ergebnisse
db_score = []
km_scores = []

# Iteration über verschiedene Clusteranzahlen
for i in range(2, 9):
    # Erstellen einer Instanz und Anpassen des KMeans-Modells
    km = KMeans(n_clusters=i, random_state=0).fit(df_k2)
    preds = km.predict(df_k2)

    # Berechnung und Speicherung der Ergebnisse
    km_scores.append(-km.score(df_k2))
    db = davies_bouldin_score(df_k2, preds)
```

```
    db_score.append(db)

    # Anzeige der Ergebnisse
    print("Wynik dla liczby klastrów {}: {}".format(i,
km.score(df_k2)))
    print("Indeks Daviesa-Bouldina dla liczby klastrów {}:
{}".format(i, db))

# Erstellung eines Punktplots
plt.figure(figsize=(8, 4))
plt.scatter(x=[i for i in range(2,9)],y=db_score,s=50,edgecolor='k')
plt.title('Indeks Kalinowskiego-Harabasza')
plt.xlabel('Liczba klastrów')
plt.ylabel('wartości indeksu Kalinowskiego-Harabasza')
# Anpassung und Anordnung der Plots im Anzeigebereich
plt.tight_layout()

# Anzeige der Plots
plt.show()
```



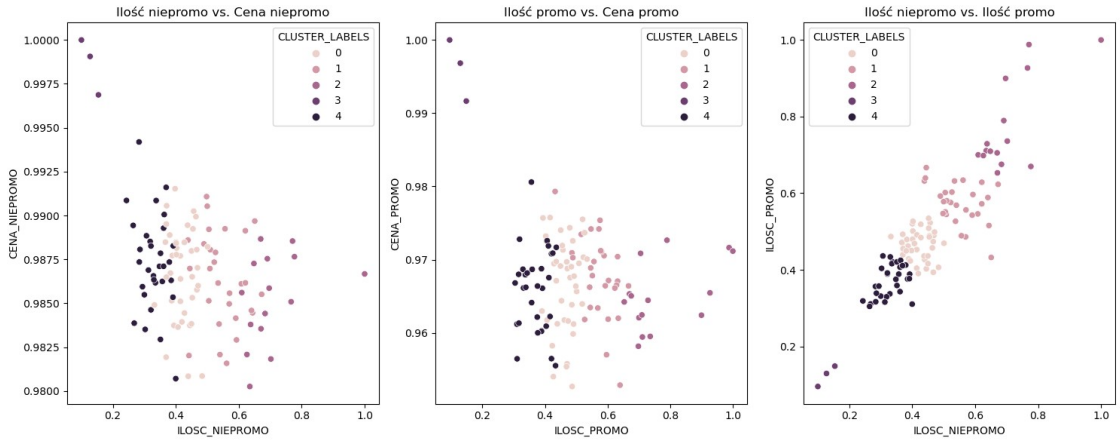
Beobachtung: Gemäß der Interpretation des Kalinowski-Harabasz-Index sollten die Märkte in 6 Gruppen aufgeteilt werden. Dies bedeutet, dass die Aufteilung in 6 Gruppen kohärenter ist und die Datenstruktur besser widerspiegelt als Aufteilungen in eine größere (oder kleinere) Anzahl von Gruppen.

KMeans Clustering

KMeans Clustering, auch als k-Means-Algorithmus bekannt, ist ein beliebter Clustering-Algorithmus in der Clusteranalyse. Das Ziel besteht darin, eine gegebene Datensammlung in k Cluster aufzuteilen, wobei jeder Cluster eine Gruppe von ähnlichen Punkten repräsentiert.

Der KMeans-Algorithmus zielt darauf ab, die Summe der quadrierten Abstände zwischen den Datenpunkten und den Zentroiden ihrer Cluster zu minimieren. Dies wird erreicht, indem die Punkte iterativ den nächstgelegenen Zentroiden zugeordnet und die Positionen der Zentroide aktualisiert werden, um die Zielfunktion zu minimieren.

```
# Aufruf des K-Means-Algorithmus für die Daten df_k[['ILOSC_NIEPROMO',
'ILOSC_PROMO', 'CENA_NIEPROMO', 'CENA_PROMO']] und 5 Cluster
cluster_centers, distortion = kmeans(df_k[['ILOSC_NIEPROMO',
'ILOSC_PROMO', 'CENA_NIEPROMO', 'CENA_PROMO']], 5)

# Zuordnung der Cluster-Labels zu den Daten aus df_k und Erhalt einer
Liste der Verzerrungen
df_k['CLUSTER_LABELS'], distortion_list = vq(df_k[['ILOSC_NIEPROMO',
'ILOSC_PROMO', 'CENA_NIEPROMO', 'CENA_PROMO']], cluster_centers)

# Erstellung von drei Subplots in einer Zeile
_, axes = plt.subplots(1, 3, figsize=(15, 6))

# Punktplot, der die Beziehung zwischen ILOSC_NIEPROMO und
CENA_NIEPROMO mit Cluster-Unterteilung darstellt
sns.scatterplot(x='ILOSC_NIEPROMO', y='CENA_NIEPROMO',
hue='CLUSTER_LABELS', data=df_k, ax=axes[0])
axes[0].set_title('Ilość niepromo vs. Cena niepromo')

# Punktplot, der die Beziehung zwischen ILOSC_PROMO und CENA_PROMO mit
Cluster-Unterteilung darstellt
sns.scatterplot(x='ILOSC_PROMO', y='CENA_PROMO', hue='CLUSTER_LABELS',
data=df_k, ax=axes[1])
axes[1].set_title('Ilość promo vs. Cena promo')

# Punktplot, der die Beziehung zwischen ILOSC_NIEPROMO und ILOSC_PROMO
mit Cluster-Unterteilung darstellt
sns.scatterplot(x='ILOSC_NIEPROMO', y='ILOSC_PROMO',
hue='CLUSTER_LABELS', data=df_k, ax=axes[2])
axes[2].set_title('Ilość niepromo vs. Ilość promo')

# Anpassung und Anordnung der Subplots innerhalb des Anzeigebereichs
plt.tight_layout()

# Anzeige der Plots
plt.show()
```



Hierarchical clustering

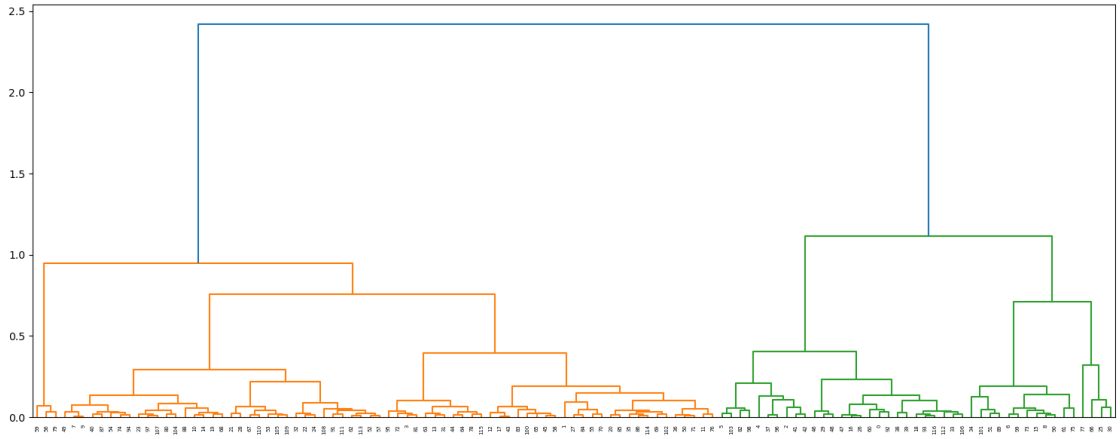
Hierarchisches Clustering, auch als hierarchische Klassifizierung bekannt, ist einer der beliebten Clustering-Algorithmen in der Clusteranalyse. Dieser Algorithmus zielt darauf ab, eine Datensammlung in hierarchische Clusterstrukturen aufzuteilen.

Das hierarchische Clustering kann in zwei verschiedenen Ansätzen durchgeführt werden: dem agglomerativen (aufsteigenden) oder dem divisiven (diskjunkten) Ansatz. Beim agglomerativen Ansatz beginnen wir mit einzelnen Punkten als Clustern und fügen sie schrittweise zusammen, um größere Cluster zu bilden. Beim divisiven Ansatz starten wir mit einem großen Cluster und teilen es iterativ in kleinere Cluster auf.

Das hierarchische Clustering bietet Flexibilität bei der Auswahl der Anzahl von Clustern, da das resultierende Dendrogramm auf verschiedenen Ebenen beschnitten werden kann, um die endgültige Aufteilung der Daten zu erhalten. Darüber hinaus kann das hierarchische Clustering zur Visualisierung der Datenstruktur verwendet werden und ermöglicht die Interpretation der Ergebnisse im Kontext der Hierarchie der Gruppen.

```
# Berechnung der Distanzmatrix mit hierarchischem Linkage für die
Daten df_k2[['ILOSC_NIEPROMO', 'ILOSC_PROMO', 'CENA_NIEPROMO',
'CENA_PROMO']]
# Verwendete Linkage-Methode: ward, Metrik: euklidisch
distance_matrix = linkage(df_k2[['ILOSC_NIEPROMO', 'ILOSC_PROMO',
'CENA_NIEPROMO', 'CENA_PROMO']], method='ward', metric='euclidean')

# Erstellen einer Abbildung und Achsen für das Dendrogramm
fig, ax = plt.subplots(figsize=(15, 6))

# Generierung des Dendrogramms basierend auf der Distanzmatrix und
Zuweisung an die Variable dn
dn = dendrogram(distance_matrix, ax=ax)

# Anpassung und Anordnung des Dendrogramms innerhalb des
Anzeigebereichs
plt.tight_layout()
```

```
# Anzeige des Dendrogramms
plt.show()
```




```
# Anzahl der Cluster
num_clusters = 3

# Initialisierung des agglomerativen Clusterings mit angegebener
Anzahl von Clustern
clustering_model = AgglomerativeClustering(n_clusters=num_clusters)

# Anpassung des Modells an die Daten df_k2[['ILOSC_NIEPROMO',
'ILOSC_PROMO', 'CENA_NIEPROMO', 'CENA_PROMO']]
clustering_model.fit(df_k2[['ILOSC_NIEPROMO', 'ILOSC_PROMO',
'CENA_NIEPROMO', 'CENA_PROMO']])

# Zuordnung der Cluster-Labels zu den Daten im DataFrame df_k2
df_k2['CLUSTER'] = clustering_model.labels_

# Erstellung von Abbildungen und Achsen für drei Streudiagramme
_, axes = plt.subplots(1, 3, figsize=(15, 6))

# Streudiagramm 1: Normalabsatz vs. Normalpreis
sns.scatterplot(x='ILOSC_NIEPROMO', y='CENA_NIEPROMO', hue='CLUSTER',
data=df_k2, ax=axes[0])
axes[0].set_title('Ilość niepromo vs. Cena niepromo')

# Streudiagramm 2: Werbemenge vs. Werbepreis
sns.scatterplot(x='ILOSC_PROMO', y='CENA_PROMO', hue='CLUSTER',
data=df_k2, ax=axes[1])
axes[1].set_title('Ilość promo vs. Cena promo')

# Streudiagramm 3: Normalabsatz vs. Werbemenge
sns.scatterplot(x='ILOSC_NIEPROMO', y='ILOSC_PROMO', hue='CLUSTER',
data=df_k2, ax=axes[2])
axes[2].set_title('Ilość niepromo vs. Ilość promo')
```

```
# Anpassung und Anordnung der Diagramme innerhalb des Anzeigebereichs
plt.tight_layout()

# Anzeige der Diagramme
plt.show()

# Entfernen unnötiger Spalten
df_k2.drop(['ILOSC_NIEPROMO', 'ILOSC_PROMO', 'CENA_NIEPROMO',
'CENA_PROMO'], axis=1, inplace=True)

# Zusammenführen von Tabellen
df = df.merge(df_k2, on='MARKET_ID', how='left')

df.head(3)
```

```
        DATA MARKET_ID GRUPA_ID ART_ID                          NAZWA
CENA_NP  \
0 2019-09-02         2       19      1  FCHKWROVC QD JROULRGU FDSYVR
3.49
1 2019-09-03         2       19      1  FCHKWROVC QD JROULRGU FDSYVR
3.49
2 2019-09-04         2       19      1  FCHKWROVC QD JROULRGU FDSYVR
3.49

   CENA_AP  PRZECENA  CENA_MARKET  OBROT PROMO_KOD Nr_dn_tyg
Nr_dn_mies  \
0     3.49       0.0         3.49    0.0         0         1
2
1     3.49       0.0         3.49    0.0         0         2
3
2     3.49       0.0         3.49    0.0         0         3
4

  Nr_mies Nr_tyg Nr_rok Dn_handlowy Hot_day Hot_day_Xmass Hot_day_Wlkn
\
0       9     36   2019           1       0             0            0

1       9     36   2019           1       0             0            0

2       9     36   2019           1       0             0            0


         HICP    y  CLUSTER
0  105.199997  0.0        2
1  105.199997  0.0        2
2  105.199997  0.0        2
```

```
# Gruppierung der Daten im DataFrame df nach den Spalten 'DATA',
'ART_ID' und 'CLUSTER' und Aggregation der Summen von 'OBROT' und 'y'
df_temp = df.groupby(['DATA', 'ART_ID', 'CLUSTER']).agg({'OBROT':
'sum', 'y': 'sum'}).reset_index()
```

```
# Berechnung von CENY_MARKET
df_temp['CENA_MARKET'] = df_temp['OBROT'] / df_temp['y']

# Entfernen unnötiger Spalten
df = df.drop(['MARKET_ID', 'OBROT', 'y', 'CENA_MARKET', 'CLUSTER'],
axis=1)
df.drop_duplicates(inplace=True)

# Zusammenführen von Tabellen
df = pd.merge(df_temp, df, how='left', on=['DATA', 'ART_ID'])

# Ergänzen und Bereinigen von CENY_MARKET
df['CENA_MARKET'] = df['CENA_MARKET'].fillna(df['CENA_AP'])
df['CENA_MARKET'] = np.where(df['CENA_MARKET'] > df['CENA_AP'],
df['CENA_AP'], df['CENA_MARKET'])

df.isnull().sum()
```

```
DATA                  0
ART_ID                0
CLUSTER               0
OBROT                 0
y                     0
CENA_MARKET      245106
GRUPA_ID         245106
NAZWA            245106
CENA_NP          245106
CENA_AP          245106
PRZECENA         245106
PROMO_KOD        245106
Nr_dn_tyg        245106
Nr_dn_mies       245106
Nr_mies          245106
Nr_tyg           245106
Nr_rok           245106
Dn_handlowy      245106
Hot_day          245106
Hot_day_Xmass    245106
Hot_day_Wlkn     245106
HICP             245106
dtype: int64
```

```
# Entfernen von NaN-Werten (NaN entstehen durch Kombinationen von
Artikel-Tage, die vor der Einführung bestimmter Artikel in das
Sortiment liegen)
df.dropna(inplace=True)

df.head()
```

```
        DATA ART_ID  CLUSTER   OBROT           y  CENA_MARKET GRUPA_ID
\
0 2019-09-02      1        0  190.46   73.000000     2.609041       19
```

```
1 2019-09-02      1         1   53.14   16.000000     3.321250        19

2 2019-09-02      1         2   68.77   23.000000     2.990000        19

3 2019-09-02      2         0  624.58  111.639999     5.594590         8

4 2019-09-02      2         1  473.01   86.529999     5.466428         8


                           NAZWA  CENA_NP  CENA_AP  PRZECENA PROMO_KOD
\
0  FCHKWROVC QD JROULRGU FDSYVR     3.49     3.49       0.0         0

1  FCHKWROVC QD JROULRGU FDSYVR     3.49     3.49       0.0         0

2  FCHKWROVC QD JROULRGU FDSYVR     3.49     3.49       0.0         0

3                   JHSDL ERSCR     5.99     5.99       0.0         0

4                   JHSDL ERSCR     5.99     5.99       0.0         0


  Nr_dn_tyg Nr_dn_mies Nr_mies Nr_tyg Nr_rok Dn_handlowy Hot_day  \
0         1          2       9     36   2019           1       0
1         1          2       9     36   2019           1       0
2         1          2       9     36   2019           1       0
3         1          2       9     36   2019           1       0
4         1          2       9     36   2019           1       0

  Hot_day_Xmass Hot_day_Wlkn        HICP
0             0            0  105.199997
1             0            0  105.199997
2             0            0  105.199997
3             0            0  105.199997
4             0            0  105.199997

df.shape

(2041254, 22)

# Änderung der Reihenfolge der Spalten
new_order = ['DATA', 'CLUSTER', 'GRUPA_ID', 'ART_ID', 'NAZWA',
'CENA_NP', 'CENA_AP', 'PRZECENA', 'CENA_MARKET', 'PROMO_KOD',
'Nr_dn_tyg', 'Nr_dn_mies', 'Nr_mies', 'Nr_tyg', 'Nr_rok',
'Dn_handlowy', 'Hot_day', 'Hot_day_Xmass', 'Hot_day_Wlkn', 'HICP',
'y']
df = df.reindex(columns=new_order)
```

EDA

Beschreibung der Daten

Der DataFrame enthält Daten über den Verkauf verschiedener Lebensmittelartikel an einem bestimmten Tag über einen Zeitraum von 3 Jahren. Die Spalten enthalten folgende Informationen:

ART_ID: Verkaufsdatum (z.B. 2019-09-02), **CLUSTER:** eine eindeutige ID des Clusters, der eine Gruppe von Märkten mit ähnlichen Verkaufsmerkmalen repräsentiert (z.B. 1, 2), **GRUPA_ID:** bestimmt die Zugehörigkeit des Produkts zu einer bestimmten Kategorie (z.B. 2, 19), **NAZWA:** codierter Artikelname (z.B. 'FCHKWROVC QD JROULRGU FDSYVR'), **CENA_NP**: Preis des Produkts im Einzelhandel (z.B. 3.3250, 5.7442), **CENA_AP:** Preis der Produkte während des Werbezeitraums (z.B. 2.9900, 4.4900), **PRZECENA:** Prozentsatz der Preisreduzierung während des Werbezeitraums im Vergleich zum Vor-Promotionspreis (z.B. 0.3333 = 33,3%), **CENA_MARKET**: Preis des Produkts im Markt unter Berücksichtigung eventueller Markt-Rabatte oder Abschreibungen (z.B. 3.026236, 5.573676), **PROMO_KOD:** Aktionscode für das Produkt (z.B. 0, 13.0), **Nr_dn_tyg:** Wochentagsnummer (z.B. 1, 2), **Nr_dn_mies:** Tagesnummer im Monat (z.B. 2, 9), **Nr_mies:** Monatsnummer (z.B. 9), Nr_tyg: Wochennummer (z.B. 36), **Nr_rok:** Jahresnummer (z.B. 2019), **Dn_handlowy:** Handelstag (z.B. 1, 0), Hot_day: heißer Tag (z.B. 0, 1), **Hot_day_Xmass:** heißer Tag vor Weihnachten (z.B. 0, 1), **Hot_day_Wlkn:** heißer Tag vor Ostern (z.B. 0, 1), **HICP:**Verbraucherpreisindex für Waren und Dienstleistungen (z.B. 105,2), **y:** Anzahl der verkauften Stücke des jeweiligen Artikels (z.B. 0, 2).

Analiza sprzedaży i sezonowości

```
# Dodatkowa kolumna na potrzeby analiz
df['TYDZIEN'] = df['DATA'].dt.to_period('W').dt.to_timestamp()
```

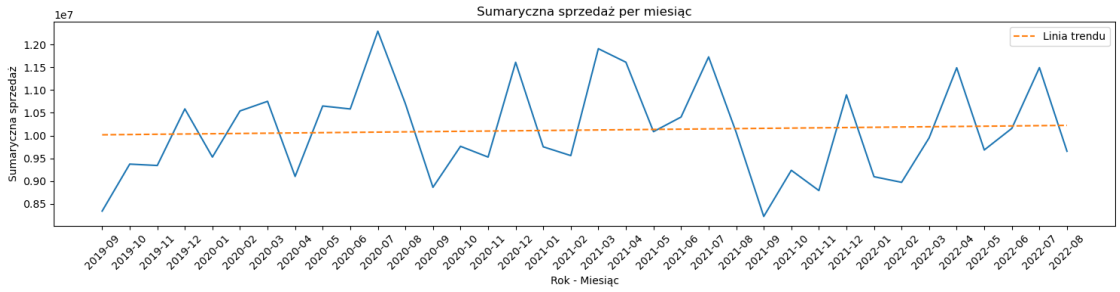
```
df.head()
```

```
        DATA  CLUSTER GRUPA_ID ART_ID                          NAZWA
CENA_NP  \
0 2019-09-02        0       19      1  FCHKWROVC QD JROULRGU FDSYVR
3.49
1 2019-09-02        1       19      1  FCHKWROVC QD JROULRGU FDSYVR
3.49
2 2019-09-02        2       19      1  FCHKWROVC QD JROULRGU FDSYVR
3.49
3 2019-09-02        0        8      2                   JHSDL ERSCR
5.99
4 2019-09-02        1        8      2                   JHSDL ERSCR
5.99

   CENA_AP  PRZECENA  CENA_MARKET PROMO_KOD Nr_dn_tyg Nr_dn_mies
Nr_mies  \
0     3.49       0.0     2.609041         0         1          2
9
1     3.49       0.0     3.321250         0         1          2
```

```
9
2     3.49       0.0     2.990000         0         1          2
9
3     5.99       0.0     5.594590         0         1          2
9
4     5.99       0.0     5.466428         0         1          2
9

  Nr_tyg Nr_rok Dn_handlowy Hot_day Hot_day_Xmass Hot_day_Wlkn
HICP  \
0     36   2019           1       0             0            0
105.199997
1     36   2019           1       0             0            0
105.199997
2     36   2019           1       0             0            0
105.199997
3     36   2019           1       0             0            0
105.199997
4     36   2019           1       0             0            0
105.199997

            y    TYDZIEN
0   73.000000 2019-09-02
1   16.000000 2019-09-02
2   23.000000 2019-09-02
3  111.639999 2019-09-02
4   86.529999 2019-09-02
```

```
# Aggregation von Daten
df['Rok-Miesiąc'] = df['DATA'].apply(lambda x: x.strftime('%Y-%m'))
monthly_sales = df.groupby('Rok-Miesiąc')['y'].sum().reset_index()

# Erstellen eines Liniendiagramms
plt.figure(figsize=(15, 4))
sns.lineplot(data=monthly_sales, x='Rok-Miesiąc', y='y')

# Berechnung von Trendlinien
x = np.arange(len(monthly_sales['Rok-Miesiąc']))
y = monthly_sales['y']
z = np.polyfit(x, y, 1)
p = np.poly1d(z)

# Hinzufügen einer Trendlinie zum Diagramm
plt.plot(monthly_sales['Rok-Miesiąc'], p(x), linestyle='--',
label='Linia trendu')
plt.title('Sumaryczna sprzedaż per miesiąc')
plt.xlabel('Rok - Miesiąc')
plt.ylabel('Sumaryczna sprzedaż')
plt.xticks(rotation=45)
plt.legend()
```

```
plt.tight_layout()
plt.show()
```

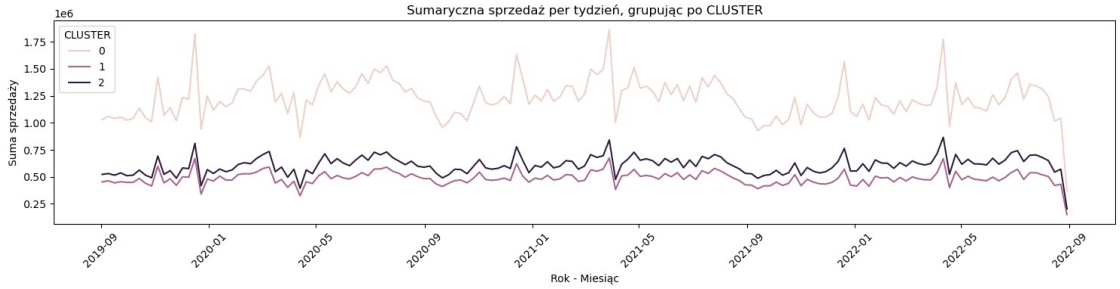


Beobachtung: Auf dem Diagramm ist zu erkennen, dass es einen geringfügigen Aufwärtstrend für den Gesamtverkauf gibt. Tatsächlich ist der Anstieg im Netzwerk größer, jedoch wird nur eine begrenzte Anzahl von Märkten analysiert.

```
df.head(3)
```

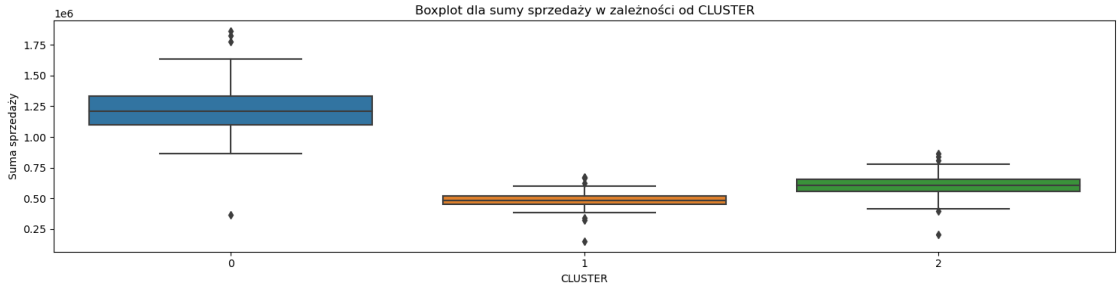
```
        DATA  CLUSTER GRUPA_ID ART_ID                          NAZWA
CENA_NP  \
0 2019-09-02        0       19      1  FCHKWROVC QD JROULRGU FDSYVR
3.49
1 2019-09-02        1       19      1  FCHKWROVC QD JROULRGU FDSYVR
3.49
2 2019-09-02        2       19      1  FCHKWROVC QD JROULRGU FDSYVR
3.49

   CENA_AP  PRZECENA  CENA_MARKET PROMO_KOD Nr_dn_tyg Nr_dn_mies
Nr_mies  \
0     3.49       0.0     2.609041         0         1          2
9
1     3.49       0.0     3.321250         0         1          2
9
2     3.49       0.0     2.990000         0         1          2
9

  Nr_tyg Nr_rok Dn_handlowy Hot_day Hot_day_Xmass Hot_day_Wlkn
HICP  \
0     36   2019           1       0             0            0
105.199997
1     36   2019           1       0             0            0
105.199997
2     36   2019           1       0             0            0
105.199997

      y    TYDZIEN Rok-Miesiąc
0  73.0 2019-09-02     2019-09
1  16.0 2019-09-02     2019-09
2  23.0 2019-09-02     2019-09
```

```
# Gruppierung der Daten und Berechnung der Umsatzsummen für jeden
CLUSTER in der Woche
df_grouped = df.groupby(['TYDZIEN', 'CLUSTER']).agg({'y':
'sum'}).reset_index()

# Liniendiagramm der aggregierten Verkäufe pro Cluster/Gruppe von
Maerkten
plt.figure(figsize=(15, 4))
sns.lineplot(data=df_grouped, x='TYDZIEN', y='y', hue='CLUSTER')
plt.title('Sumaryczna sprzedaż per tydzień, grupując po CLUSTER')
plt.xlabel('Rok - Miesiąc')
plt.ylabel('Suma sprzedaży')
plt.legend(title='CLUSTER', loc='upper left')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

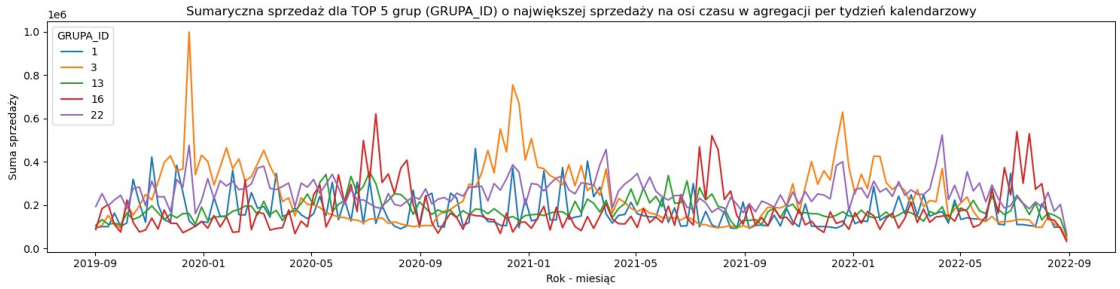


Beobachtung: Die Liniendiagramme für die einzelnen Marktkategorien (CLUSTER_LABELS) zeigen eine sehr ähnliche 'Charakteristik', das heißt, der Verlauf der Diagramme ist für alle Kategorien sehr ähnlich, nur die Diagramme sind auf der Y-Achse verschoben.

```
# Gruppierung der Daten und Berechnung der Umsatzsummen für jeden
CLUSTER_LABELS pro Woche
df_grouped = df.groupby(['TYDZIEN', 'CLUSTER']).agg({'y':
'sum'}).reset_index()

# Erstellen eines Boxplots
plt.figure(figsize=(15, 4))
sns.boxplot(data=df_grouped, x='CLUSTER', y='y')
plt.title('Boxplot dla sumy sprzedaży w zależności od CLUSTER')
plt.xlabel('CLUSTER')
plt.ylabel('Suma sprzedaży')
plt.tight_layout()
plt.show()
```

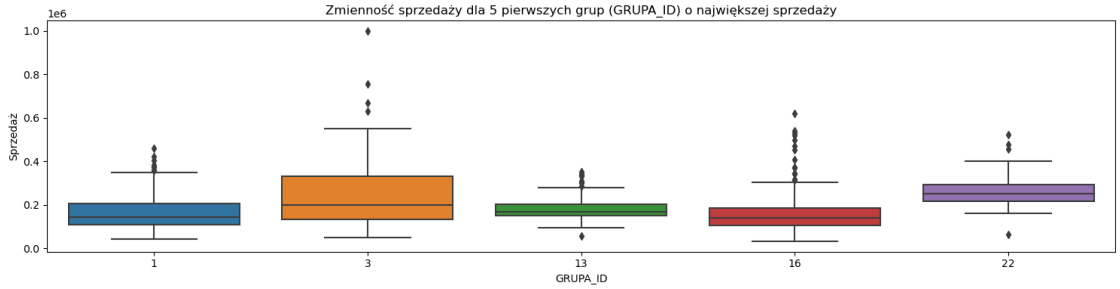



```
%%capture
# Gruppierung der Daten und Berechnung der Umsatzsummen für jede
GROUP_ID pro Woche
df_grouped = df.groupby(['TYDZIEN', 'GRUPA_ID']).agg({'y':
'sum'}).reset_index()

# Beschränkung des Diagramms auf die 5 Gruppen mit den höchsten
Umsätzen
top5_grupa_id = df_grouped.groupby('GRUPA_ID').agg({'y':
'sum'}).nlargest(5, 'y').index
df_top5 = df_grouped[df_grouped['GRUPA_ID'].isin(top5_grupa_id)]
df_top5['GRUPA_ID'] = df_top5['GRUPA_ID'].astype('int32')
df_top5['GRUPA_ID'] = df_top5['GRUPA_ID'].astype('category')

# Diagramm der Verkäufe in der aggregierten Artikelgruppe
plt.figure(figsize=(15, 4))
sns.lineplot(data=df_top5, x='TYDZIEN', y='y', hue='GRUPA_ID',
legend='auto')
plt.title('Sumaryczna sprzedaż dla TOP 5 grup (GRUPA_ID) o największej
sprzedaży na osi czasu w agregacji per tydzień kalendarzowy')
plt.xlabel('Rok - miesiąc')
plt.ylabel('Suma sprzedaży')
plt.legend([1, 3, 13, 16, 22], title='GRUPA_ID', loc='upper left')
plt.tight_layout()
plt.show()
```



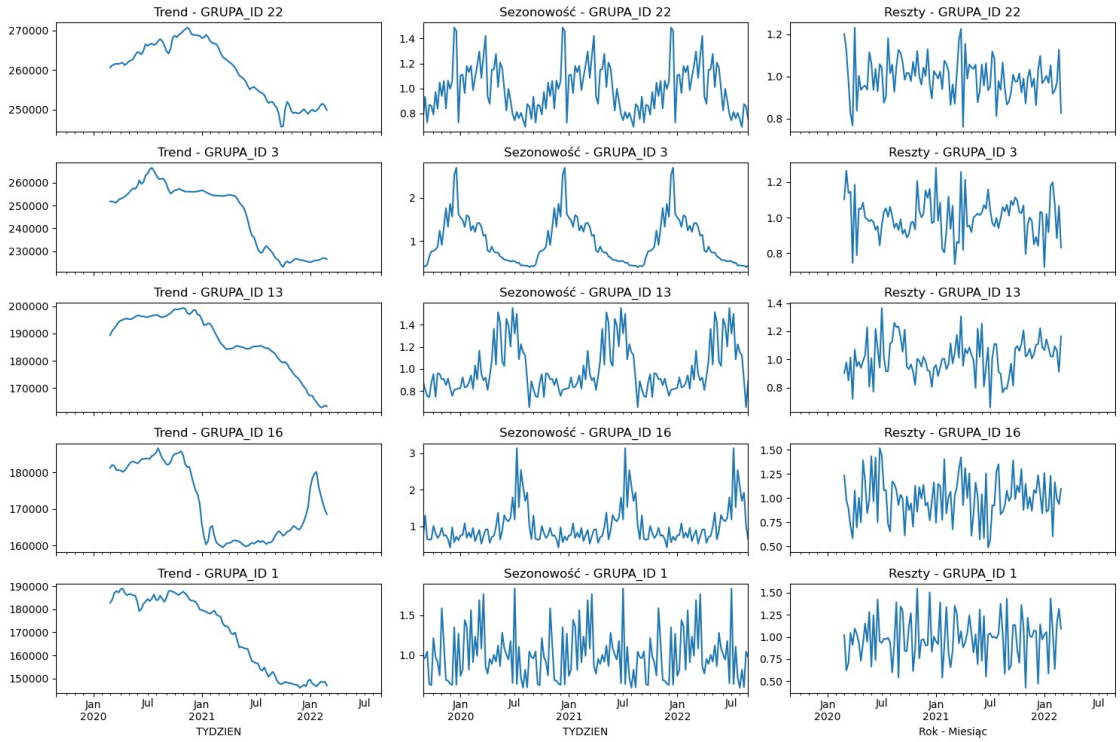
Beobachtung: Auf dem Diagramm ist eine hohe Woche-zu-Woche-Variabilität und Verkaufsspitzen zu erkennen (zu diesem Zeitpunkt ist es schwierig, festzustellen, was diese Variabilität beeinflusst). Für GRUPA_ID mit der Nummer 3 kann aus dem Diagramm auch eine jährliche Saisonalität abgelesen werden.

```
# Boxplot zur Darstellung der Umsatzvolatilität:
plt.figure(figsize=(15, 4))
sns.boxplot(data=df_top5, x='GRUPA_ID', y='y')
plt.title('Zmienność sprzedaży dla 5 pierwszych grup (GRUPA_ID) o
największej sprzedaży')
plt.xlabel('GRUPA_ID')
plt.ylabel('Sprzedaż')
plt.tight_layout()
plt.show()
```



Da das Schaubild saisonale Schwankungen aufweist, lohnt es sich, eine Zerlegung der Saisonalität vorzunehmen.

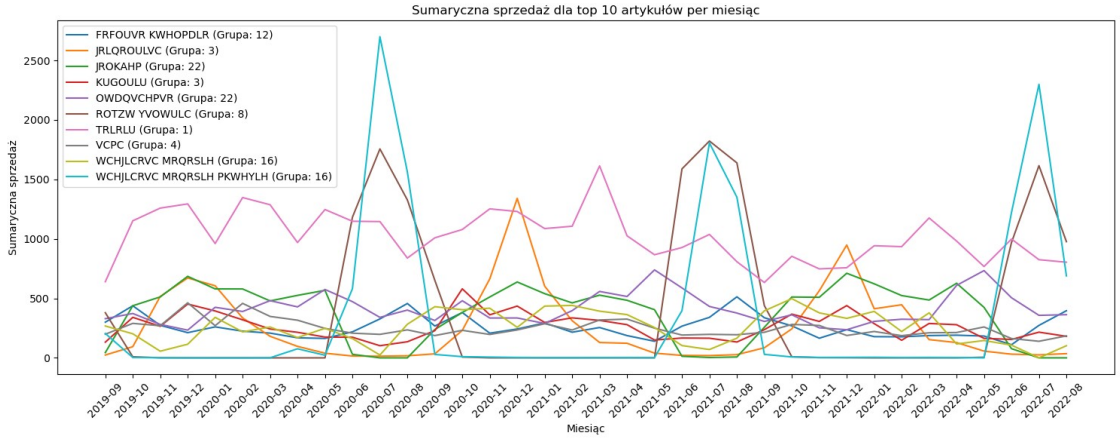
```
# Gruppierung der Daten und Berechnung des Gesamtumsatzes pro Woche
df_grouped_total = df.groupby(['TYDZIEN']).agg({'y':
'sum'}).reset_index()
df_grouped_total = df_grouped_total.set_index('TYDZIEN')

fig, axes = plt.subplots(len(top5_grupa_id), 3, figsize=(15, 2 *
len(top5_grupa_id)), sharex=True)
for i, grupa_id in enumerate(top5_grupa_id):
    df_group = df_top5[df_top5['GRUPA_ID'] ==
grupa_id].set_index('TYDZIEN')

    # Zerlegung der Saisonalität
    result = seasonal_decompose(df_group['y'], model='multiplicative',
period=52)  # Wir legen den Zeitraum auf 52 Wochen (1 Jahr) fest.

    # Diagramm zur Zerlegung der Gruppensaisonalität
    result.trend.plot(ax=axes[i, 0], title=f'Trend - GRUPA_ID
{grupa_id}')
    result.seasonal.plot(ax=axes[i, 1], title=f'Sezonowość - GRUPA_ID
{grupa_id}')
    result.resid.plot(ax=axes[i, 2], title=f'Reszty - GRUPA_ID
{grupa_id}')
plt.xlabel('Rok - Miesiąc')
plt.tight_layout()
plt.show()
```



Beobachtung: Die Diagramme zeigen, dass verschiedene Gruppen unterschiedliche Trends und unterschiedliche Saisonalitäten aufweisen. Diese Beobachtung stimmt mit dem alltäglichen Verständnis überein, dass es viele Produkte gibt, deren Verkaufsvolumen durch Angebot und Nachfrage bestimmt wird (z. B. durch den Preis, zu dem Verbraucher kaufen möchten) und durch die Verfügbarkeit (z. B. aufgrund saisonaler Produktion von Lebensmitteln).

```
# Datenaggregation
df['Rok-Miesiąc'] = df['DATA'].apply(lambda x: x.strftime('%Y-%m'))
top10_articles = df.groupby(['ART_ID', 'GRUPA_ID', 'NAZWA'])
['y'].sum().nlargest(10).reset_index()
top10_articles['NAZWA_GRUPA'] = top10_articles['NAZWA'] + ' (Grupa: '
+ top10_articles['GRUPA_ID'].astype(str) + ')'
df['NAZWA_GRUPA'] = df['NAZWA'] + ' (Grupa: ' +
df['GRUPA_ID'].astype(str) + ')'
top10_articles_df =
df[df['ART_ID'].isin(top10_articles['ART_ID'])].groupby(['Rok-
Miesiąc', 'ART_ID', 'NAZWA_GRUPA'])['y'].sum().reset_index()

# Liniendiagramm der Verkäufe für TOP10-Artike
plt.figure(figsize=(15, 4))
sns.lineplot(data=top10_articles_df, x='Rok-Miesiąc', y='y',
hue='NAZWA_GRUPA', legend='auto', ci=None)
plt.title('Gesamtumsatz für die 10 wichtigsten Artikel pro Monat')
plt.xlabel('Monat')
plt.ylabel('Gesamtumsatz')
plt.xticks(rotation=45)
plt.legend(loc='upper left')
```

```
plt.tight_layout()
plt.show()
```

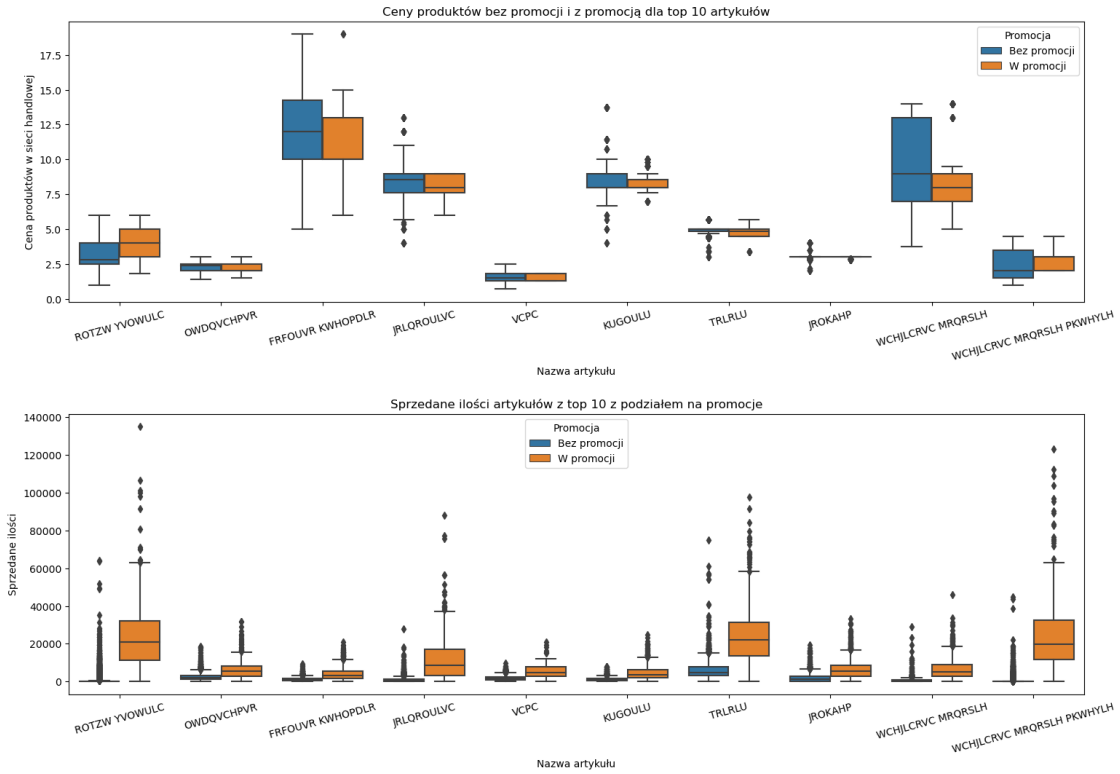


Beobachtung: Auf dem Diagramm ist zu erkennen, dass es 2 Artikel aus den TOP10 gibt, die eine sehr starke jährliche Saisonalität im Sommer aufweisen, sowie einen Artikel mit einer starken saisonalen Nachfrage im Winter, mit einem Verkaufshöhepunkt im Dezember.

Analyse der Auswirkungen von Werbeaktionen auf Preise und Verkäufe

```
# TOP10 Artikel mit den höchsten Umsätzen
top10_art = df.groupby('ART_ID').agg({'y': 'sum'}).sort_values(by='y',
ascending=False).head(10).index

# Daten nur für TOP10-Artikel
top10_df = df[df['ART_ID'].isin(top10_art)].copy()
top10_df['ART_ID'] = top10_df['ART_ID'].astype(int)

# Wörterbuch zur Zuordnung von ART_ID zu NAME
art_id_to_name = top10_df[['ART_ID',
'NAZWA']].drop_duplicates().set_index('ART_ID').to_dict()['NAZWA']

# Änderung von ART_ID in NAME in TOP10 DataFrame
top10_df['NAZWA'] = top10_df['ART_ID'].apply(lambda x:
art_id_to_name[x])

# Hinzufügung von "Promotion", die Informationen darüber enthält, ob
ein Produkt in der Promotion war
top10_df['Promocja'] = top10_df['PROMO_KOD'].apply(lambda x: 'W
promocji' if x != 0 else 'Bez promocji')
fig, axes = plt.subplots(nrows=2, ncols=1, figsize=(15, 8))
plt.subplots_adjust(hspace=0.4)

# Boxplot für Produktpreise gruppiert nach Artikel und Aktion
sns.boxplot(ax=axes[0], data=top10_df, x='NAZWA', y='CENA_NP',
hue='Promocja')
axes[0].set_title('Ceny produktów bez promocji i z promocją dla TOP10
```

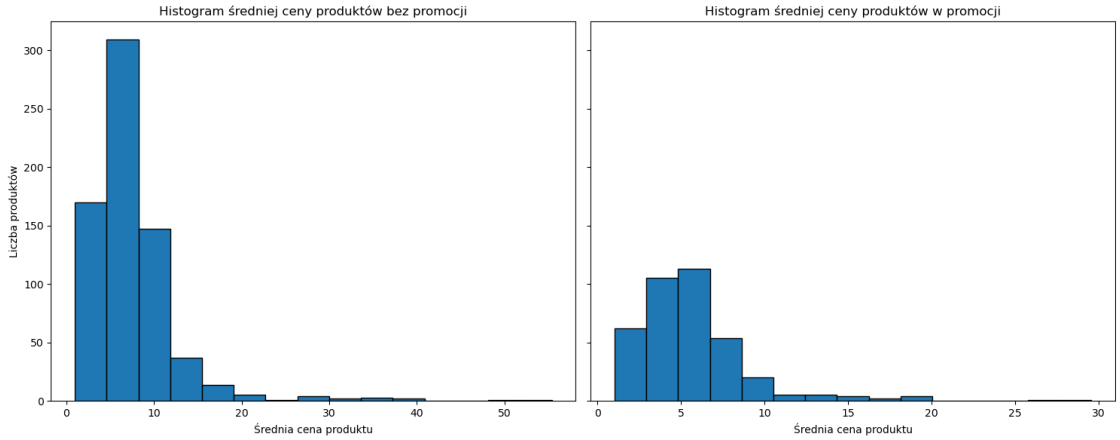
```
artykułów')
axes[0].set_xlabel('Nazwa artykułu')
axes[0].set_ylabel('Cena produktów w sieci handlowej')
axes[0].tick_params(axis='x', rotation=15)
axes[0].legend(title='Promocja')

# Boxplot für die verkauften Mengen der TOP10-Artikel nach Aktion
sns.boxplot(ax=axes[1], data=top10_df, x='NAZWA', y='y',
hue='Promocja')
axes[1].set_title('Sprzedane ilości artykułów z TOP10 z podziałem na
promocje')
axes[1].set_xlabel('Nazwa artykułu')
axes[1].set_ylabel('Sprzedane ilości')
axes[1].tick_params(axis='x', rotation=15)
axes[1].legend(title='Promocja')

plt.show()
```



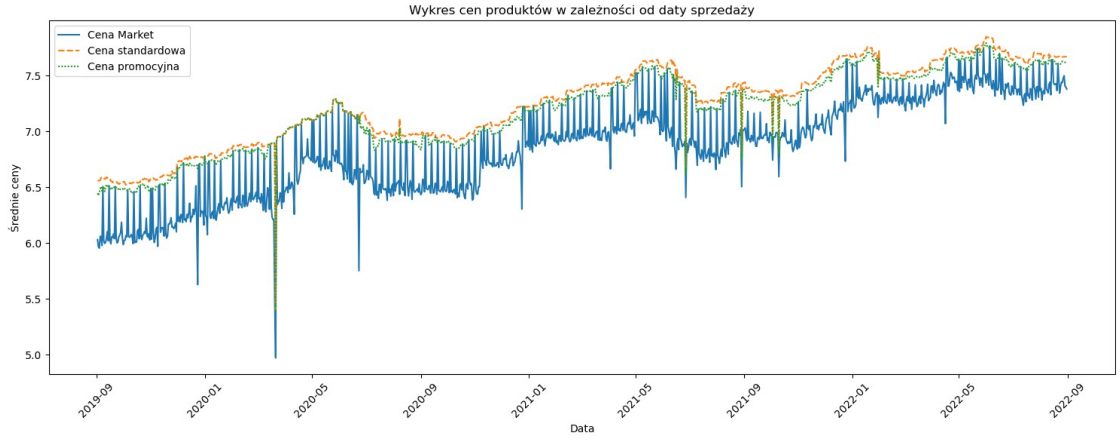
Beobachtung: Auf den Diagrammen ist zu erkennen, dass die Preise der Produkte während der Werbeaktionen niedriger sind, was zu einem höheren Absatz führt.

```
# Hinzufügung einer Spalte "Werbeaktion", um anzuzeigen, ob das
Produkt im Angebot war
df['Promocja'] = df['PROMO_KOD'].apply(lambda x: 'W promocji' if x !=
0 else 'Bez promocji')

# Berechnung des Durchschnittspreises für jedes Produkt (ART_ID),
```

```
aufgeschlüsselt nach Aktionen
avg_price_per_product_promo = df.groupby(['ART_ID', 'Promocja'])
['CENA_AP'].mean().reset_index()

# Zwei Histogramme nebeneinander erstellen
fig, axes = plt.subplots(1, 2, figsize=(15, 4), sharey=True)

# Histogramm für "normall" Preise
axes[0].hist(avg_price_per_product_promo[avg_price_per_product_promo['
Promocja'] == 'Bez promocji']['CENA_AP'], bins=15, edgecolor='black')
axes[0].set_title('Histogram średniej ceny produktów bez promocji')
axes[0].set_xlabel('Średnia cena produktu')
axes[0].set_ylabel('Liczba produktów')

# Histogramm für Aktionspreise
axes[1].hist(avg_price_per_product_promo[avg_price_per_product_promo['
Promocja'] == 'W promocji']['CENA_AP'], bins=15, edgecolor='black')
axes[1].set_title('Histogram średniej ceny produktów w promocji')
axes[1].set_xlabel('Średnia cena produktu')

plt.tight_layout()
plt.show()
```

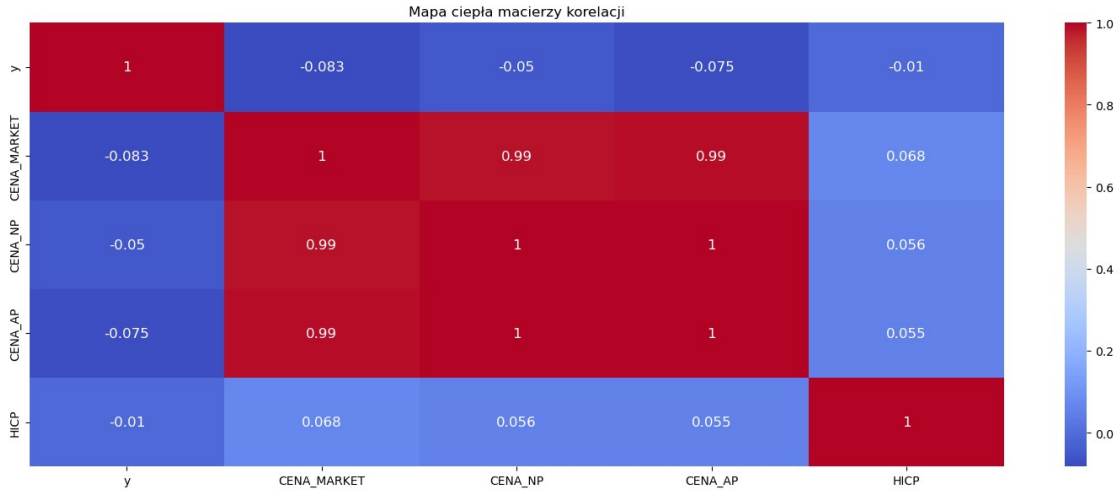



```
# Liniendiagramm der Produktpreise nach Verkaufsdatum
plt.figure(figsize=(15, 4))
sns.lineplot(data=df.groupby('DATA').agg({'CENA_MARKET': 'mean',
'CENA_NP': 'mean', 'CENA_AP': 'mean'}))
plt.title('Wykres cen produktów w zależności od daty sprzedaży')
plt.xlabel('Data')
plt.ylabel('Średnie ceny')
plt.legend(['Cena Market', 'Cena standardowa', 'Cena promocyjna'],
loc='upper left')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

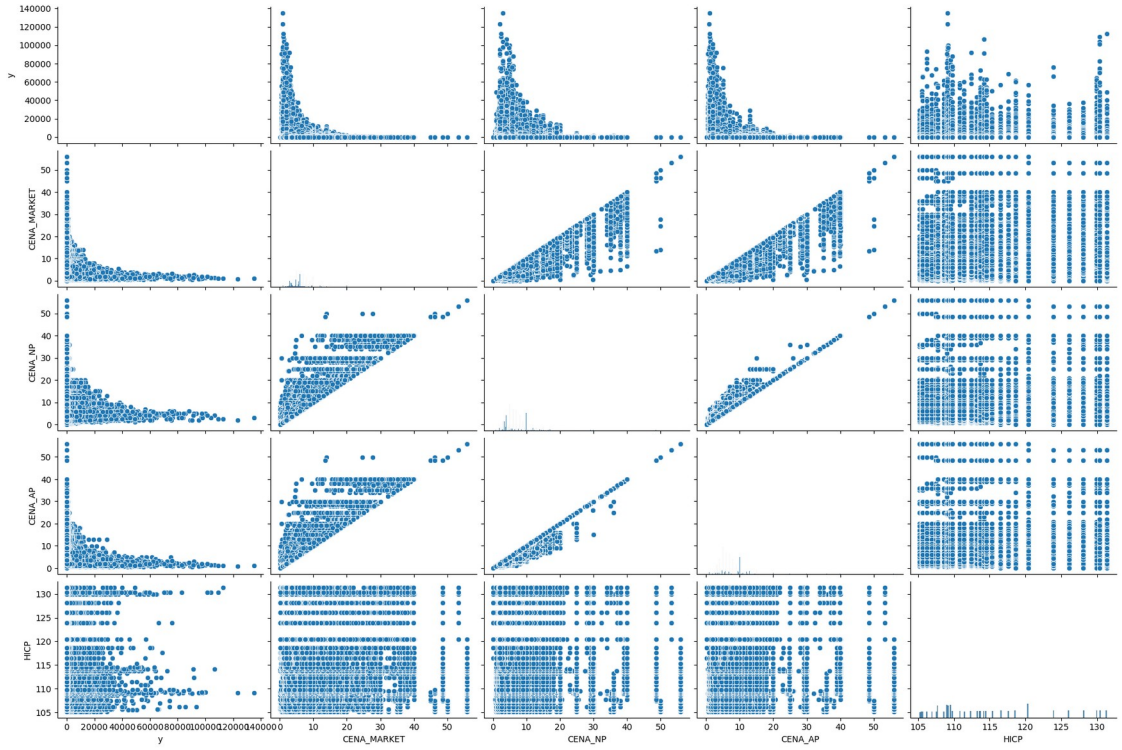


Andere Analysen

```
# Korrelationsmatrix zwischen numerischen Spalten
num_columns = ['y', 'CENA_MARKET', 'CENA_NP', 'CENA_AP', 'HICP']
correlation_matrix = df[num_columns].corr()

# Heatmap der Korrelationsmatrix
plt.figure(figsize=(15, 4))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm',
annot_kws={'size': 12})
plt.title('Heatmap der Korrelationsmatrix')
plt.tight_layout()
plt.show()
```



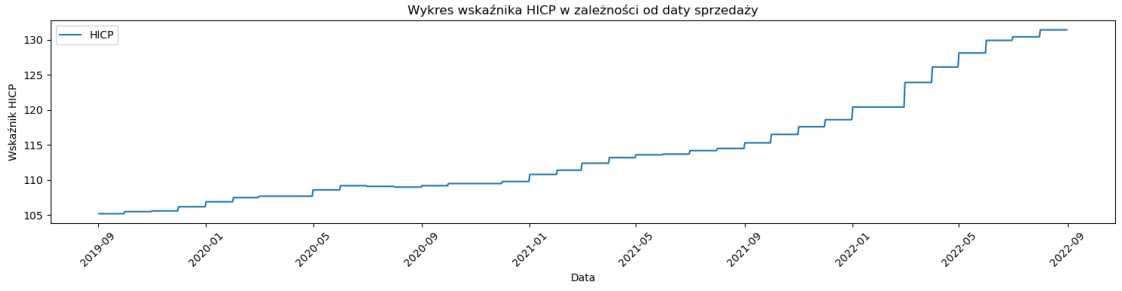

```
# Punktdiagramme für Variablenpaare mit hohem
Korrelationskoeffizienten
sns.pairplot(df[['y', 'CENA_MARKET', 'CENA_NP', 'CENA_AP', 'HICP']],
aspect = 15/10)
plt.figure(figsize=(15, 8))
plt.tight_layout()
```




```
<Figure size 1500x1000 with 0 Axes>
```

```
plt.show()
```

```
# Liniendiagramm des Indikators  HICP
plt.figure(figsize=(15, 4))
sns.lineplot(data=df.groupby('DATA').agg({'HICP': 'mean'}))
plt.title('Liniendiagramm des Indikators HICP')
plt.xlabel('Datum')
plt.ylabel('Indikator HICP')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



Modellierung

Wahl der Modellierungstechniken

In diesem Kapitel werden verschiedene Modelle zur Vorhersage von Zeitreihen im Rahmen der CRISP-DM-Methodologie angewendet. Diese umfassen sowohl Modelle für eindimensionale Zeitreihen als auch für mehrdimensionale Zeitreihen. Im Folgenden finden Sie Details zu den einzelnen Modellen:

Für eindimensionale Zeitreihen:

1. **KNN Regressor** - Ein auf den k-nächsten Nachbarn basierendes Modell, das den Durchschnittswert für Vorhersagen basierend auf den nächsten Beobachtungen aus der Vergangenheit berechnet.
2. **SARIMAX** - Eine erweiterte ARIMA-Modellvariante, die Saisonalität und exogene Variablen berücksichtigt. Besonders nützlich für Zeitreihen mit saisonalen Mustern.
3. **Prophet** - Ein von Facebook Research entwickeltes Modell, das Saisonalität, Trends sowie Feiertags- und spezielle Ereigniseffekte berücksichtigt.
4. **XGBoost Boosting** - Ein auf Entscheidungsbäumen basierender Algorithmus, der den Gradienten optimiert, um bessere Vorhersagen zu erzielen. Es handelt sich um eine fortgeschrittene Ensemble-Technik, die mehrere Entscheidungsbäume kombiniert und so zur Verbesserung der Vorhersagequalität beiträgt.
5. **Prophet + XGBoost** - Eine Kombination aus dem Prophet-Modell und XGBoost, die die Vorteile beider Ansätze nutzt, um bessere Ergebnisse zu erzielen.
6. **Dense 'Vanilla'-Neuronales Netzwerk** - Ein einfaches neuronales Netzwerkmodell, das sich auf grundlegende Netzwerkstrukturen und Aktivierungen konzentriert.
7. **Erweitertes Dense neuronales Netzwerk** - Ein neuronales Netzwerkmodell mit mehr dichten Schichten, das das Lernen komplexerer Datenrepräsentationen ermöglicht.
8. **LSTM-Recurrent Neural Network** - Ein spezieller Typ eines rekurrenten neuronalen Netzwerks, der gut zur Modellierung von Zeitreihendaten geeignet ist und langfristige zeitliche Abhängigkeiten berücksichtigt.

Für mehrdimensionale Zeitreihen:

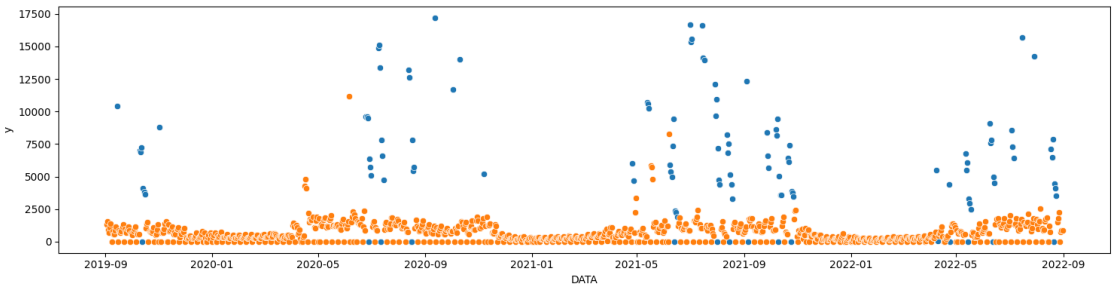
1. **XGBoost** - Eine an mehrdimensionale Zeitreihen angepasste XGBoost-Implementierung.
2. **Erweitertes Dense Netzwerk** - Ein neuronales Netzwerk mit mehr dichten Schichten, das sich auf die Modellierung von Abhängigkeiten zwischen Merkmalen konzentriert.
3. **Erweitertes Dense Netzwerk mit Embedding-Schicht** - Ein neuronales Netzwerk mit zusätzlicher Embedding-Schicht, die eine effizientere Modellierung von mehrdimensionalen Daten ermöglicht.

Im Prozess der Auswahl des besten Modells für die Vorhersage von Zeitreihen werden alle oben genannten Modelle getestet und ihre Ergebnisse verglichen. In den folgenden Phasen des CRISP-DM-Prozesses werden Schritte zur Datenvorbereitung, Modelltraining und Evaluierung ihrer Leistung vorgestellt. Das Ziel besteht darin, den effektivsten Ansatz zur Lösung des Zeitre

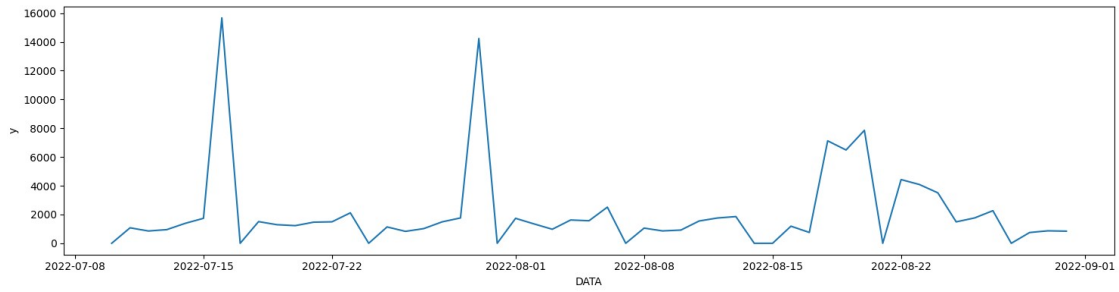
Budowa modeli oraz testów dla jednowymiarowych szeregów czasowych

Import i preprocesing danych

```
# Laden von Daten
df = pd.read_pickle('PICKLE/df_3.pkl')

# Beschränkung der Daten auf ein Cluster und Auswahl eines Artikels
df = df.loc[df['CLUSTER']==2].drop(['CLUSTER'], axis=1)
df = df.loc[df['ART_ID']==158].drop(['ART_ID', 'GRUPA_ID', 'NAZWA'],
axis=1)

df.shape

(1095, 17)

# Einstellung von DATA als Index
df.set_index('DATA', inplace=True)

# Aufschlüsselung der Daten nach verkaufsfördernden und nicht
verkaufsfördernden Maßnahmen
df_promo = df.loc[df['PROMO_KOD']!=0]
df_nonpromo = df.loc[df['PROMO_KOD']==0]

# Streudiagramm der Verkäufe
plt.figure(figsize=(15, 4))
sns.scatterplot(data=df_promo, x=df_promo.index, y='y')
sns.scatterplot(data=df_nonpromo, x=df_nonpromo.index, y='y')
plt.tight_layout()
plt.show()
```




```
# Lineplot der Verkäufe für den "Test"-Zeitraum
plt.figure(figsize=(15, 4))
sns.lineplot(data=df.loc[df.index>='2022-07-10'],
x=df.index[df.index>='2022-07-10'], y='y')
plt.tight_layout()
plt.show()
```



Bemerkung: Der ausgewählte Zeitraum umfasst 3 Aktionen: zwei eintägige Aktionen und eine wöchentliche Aktion.

Kodierung der Daten

```
# Definition von Spalten pro Datentyp
nominal_columns = ['PROMO_KOD', 'Dn_handlowy', 'Hot_day']
ordinal_columns = ['Hot_day_Xmass', 'Hot_day_Wlkn', 'Nr_dn_tyg',
'Nr_dn_mies', 'Nr_mies', 'Nr_tyg', 'Nr_rok']
numeric_columns = ['CENA_MARKET', 'CENA_AP', 'CENA_NP', 'PRZECENA',
'HICP']

# Erstellung von Transformationen
nominal_transformer = OneHotEncoder(handle_unknown='ignore')
ordinal_transformer =
OrdinalEncoder(handle_unknown='use_encoded_value', unknown_value=-1)
numeric_transformer = MinMaxScaler()

# Erstellung von ColumnTransformer
preprocessor = ColumnTransformer(
    transformers=[
        ('nominal', nominal_transformer, nominal_columns),
        ('ordinal', ordinal_transformer, ordinal_columns),
        ('numeric', numeric_transformer, numeric_columns)
    ])

# Erstellung der Pipeline
pipeline = Pipeline(steps=[('preprocessor', preprocessor)])

# Umwandlung von Daten
X_train, y_train = df.loc[df.index <='2022-07-10 00:00:00'],
df.loc[df.index <= '2022-07-10 00:00:00']
X_test, y_test = df.loc[df.index>'2022-07-10 00:00:00'],
df.loc[df.index > '2022-07-10 00:00:00']
X_train = X_train.drop(columns=['y'])
X_test = X_test.drop(columns=['y'])
y_train = y_train['y']
y_test = y_test['y']

# Anwendung der Pipeline
X_trans_transformed = pipeline.fit_transform(X_train)
```

```
X_train_transformed = pipeline.fit_transform(X_train)
X_test_transformed = pipeline.transform(X_test)

# Umwandlung von Nennspalten mit OneHotEncoder und Abrufen ihrer Namen
nominal_transformed_columns =
nominal_transformer.fit(X_train[nominal_columns]).get_feature_names_ou
t(nominal_columns)

# Kombination von nominalen, ordinalen und numerischen Spaltennamen
transformed_columns = np.concatenate((nominal_transformed_columns,
ordinal_columns, numeric_columns))

# Erstellung von transformierten DataFrames mit transformierten
Spaltennamen
X_train_transformed_df = pd.DataFrame(X_train_transformed,
columns=transformed_columns)
X_test_transformed_df = pd.DataFrame(X_test_transformed,
columns=transformed_columns)
X_trans_transformed = pd.DataFrame(X_trans_transformed,
columns=transformed_columns)

# Dimensionen der einzelnen DataFrames
print(X_train_transformed.shape)
print(y_train.shape)
print(X_test_transformed.shape)
print(y_test.shape)

(1043, 26)
(1043,)
(52, 26)
(52,)

X_train_transformed_df.head(2)

   PROMO_KOD_0  PROMO_KOD_3  PROMO_KOD_4  PROMO_KOD_5  PROMO_KOD_8  \
0          1.0          0.0          0.0          0.0          0.0
1          1.0          0.0          0.0          0.0          0.0

   PROMO_KOD_20  PROMO_KOD_58  PROMO_KOD_107  PROMO_KOD_115
PROMO_KOD_118  \
0           0.0           0.0            0.0            0.0
0.0
1           0.0           0.0            0.0            0.0
0.0

   Dn_handlowy_0  Dn_handlowy_1  Hot_day_0  Hot_day_1
Hot_day_Xmass  \
0            0.0            1.0        1.0        0.0           14.0

1            0.0            1.0        1.0        0.0           14.0
```

```
   Hot_day_Wlkn  Nr_dn_tyg  Nr_dn_mies  Nr_mies  Nr_tyg  Nr_rok
CENA_MARKET  \
0          14.0        0.0         1.0      8.0    35.0     0.0
0.150303
1          14.0        1.0         2.0      8.0    35.0     0.0
0.095115

    CENA_AP   CENA_NP  PRZECENA  HICP
0  0.161677  0.034483       0.0   0.0
1  0.161677  0.034483       0.0   0.0
```

Basismodell - KNN

Die k-Nächste-Nachbarn-Methode (KNN) ist eine beliebte Technik zur Regression in Zeitreihen. Der KNN-Regressor für Zeitreihen besteht darin, den Wert eines Datenpunkts basierend auf seiner Ähnlichkeit zu seinen k nächsten Nachbarn vorherzusagen.

In Zeitreihen sind die Daten nach Zeit geordnet, daher ist es wichtig, diese Struktur bei der Vorhersage zu berücksichtigen. Der KNN-Regressor für Zeitreihen basiert auf der Idee, dass Datenpunkte, die zeitlich nahe beieinander liegen, ähnliche Werte haben. Der KNN-Algorithmus sucht nach den k nächsten Nachbarn eines gegebenen Datenpunkts und berechnet den vorhergesagten Wert, zum Beispiel durch den Durchschnitt der Werte dieser Nachbarn.

Um den KNN-Regressor für Zeitreihen anzuwenden, kann man eine Ähnlichkeitsmetrik wie den euklidischen Abstand oder die Korrelation verwenden, um den Abstand zwischen den Datenpunkten zu berechnen. Anschließend werden die k nächsten Nachbarn ausgewählt und die Vorhersage basierend auf ihren Werten berechnet. Der Parameter k gibt an, wie viele Nachbarn bei der Vorhersage berücksichtigt werden sollen.

```
# Parameter
n_neighbors_values = np.arange(1, 25, 1)
weights_values = ['uniform', 'distance']
algorithm_values = ['auto', 'ball_tree', 'kd_tree', 'brute']
leaf_size_values = np.arange(1, 5, 1)
p_values = [1, 2]

# Erstellung von Parameterkombinationen
param_combinations_KNR = list(product(n_neighbors_values,
weights_values, algorithm_values, leaf_size_values, p_values))
len(param_combinations_KNR)
```

```
1536
```

```
# Initialisierung eines Datenrahmens, um die Ergebnisse der Metriken
für jedes Folio zu speichern
results_df_KNR = pd.DataFrame(columns=['Fold', 'n_neighbors',
'weights', 'algorithm', 'leaf_size', 'p', 'MAE', 'RMSE', 'R2',
'WMAPE', 'MASE'])
```

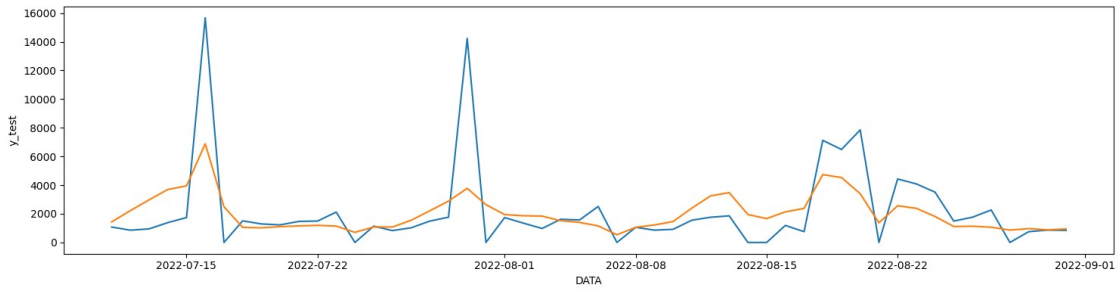
```
# TimeSeriesSplit-Objekt erstellen
tscv = TimeSeriesSplit(n_splits=5)

# Initialisierung eines Datenrahmens zur Speicherung der besten
Metrik-Ergebnisse für jeden Fold
best_results_df_KNR = pd.DataFrame(columns=['Fold', 'n_neighbors',
'weights', 'algorithm', 'leaf_size', 'p', 'MAE', 'RMSE', 'R2', 'MAPE',
'MASE'])

# Iteration nach Fold's
for fold, (train_index, test_index) in enumerate(tscv.split(X_train)):
    X_train_KNR = X_train_transformed_df.iloc[train_index]
    X_test_KNR = X_train_transformed_df.iloc[test_index]
    y_train_KNR = y_train.iloc[train_index]
    y_test_KNR = y_train.iloc[test_index]

    best_rmse = float('inf')
    best_params = None
    best_mae, best_rmse, best_r2, best_mape, best_mase = float('inf'),
float('inf'), float('-inf'), float('inf'), float('inf')


    for params in param_combinations_KNR:
        n_neighbors, weights, algorithm, leaf_size, p = params
        KNR = KNeighborsRegressor(
            n_neighbors=n_neighbors,
            weights=weights,
            algorithm=algorithm,
            leaf_size=leaf_size,
            p=p
        )
        KNR.fit(X_train_KNR, y_train_KNR)
        y_pred_KNR = KNR.predict(X_test_KNR)
        rmse = mean_squared_error(y_test_KNR, y_pred_KNR,
squared=False)

        if rmse < best_rmse:
            best_rmse = rmse
            best_params = params
            best_mae, best_rmse, best_r2, best_wmape, best_mase =
ff.metryki(y_test_KNR, y_pred_KNR)

    # Das beste Modell erstellen
    n_neighbors, weights, algorithm, leaf_size, p = best_params
    KNR = KNeighborsRegressor(
        n_neighbors=n_neighbors,
        weights=weights,
        algorithm=algorithm,
        leaf_size=leaf_size,
```

```
        p=p
    )
    KNR.fit(X_train_KNR, y_train_KNR)
    y_pred = KNR.predict(X_test_KNR)

    # Aufzeichnung der besten Metrik-Ergebnisse für die Fold's
    best_results_df_KNR = best_results_df_KNR.append({'Fold': fold,
'n_neighbors': n_neighbors, 'weights': weights,
                                      'algorithm': algorithm,
'leaf_size': leaf_size, 'p': p,
                                      'MAE': best_mae, 'RMSE':
best_rmse, 'R2': best_r2, 'WMAPE': best_wmape, 'MASE': best_mase},
ignore_index=True)

# Kopieren der besten Ergebnisse der Metrik in den resultierenden
Datenrahmen
results_df_KNR = best_results_df_KNR.copy()

# Anzeige eines Datenrahmens mit den Ergebnissen der Metriken für
jeden Fold
results_df_KNR.head()
```

```
  Fold n_neighbors   weights algorithm leaf_size  p          MAE
RMSE  \
0    0           1  distance      auto         1  1  1199.202024
2674.990397
1    1           1   uniform      auto         1  1   403.199371
1238.140747
2    2          13   uniform      auto         1  1  1292.112305
2738.364258
3    3          13  distance      auto         1  1   866.763619
1855.168874
4    4          19  distance      auto         1  1   632.985193
1070.897645

         R2 MAPE      MASE      WMAPE
0  0.085450  NaN  0.966899   0.498234
1  0.606451  NaN  0.468190   2.056895
2  0.328096  NaN  1.094599   0.603830
3  0.398234  NaN  0.801210  15.581075
4  0.609607  NaN  0.878982   3.084313
```

```
# Forecast
y_predict_KNR = KNR.predict(X_test_transformed_df)
y_predict_KNR_past = KNR.predict(X_train_transformed_df)

# Erstellung eines DataFrame zur Visualisierung
KNR_pred = df.loc[df.index >'2022-07-10 00:00:00'][['y']]
KNR_pred['y_test'] = KNR_pred['y']
KNR_pred = KNR_pred.drop('y', axis=1)
KNR_pred['y_predict'] = y_predict_KNR
```

```
# Grafik - Vergleich der Vorhersagen
plt.figure(figsize=(15, 4))
sns.lineplot(data=KNR_pred, x=KNR_pred.index, y='y_test')
sns.lineplot(data=KNR_pred, x=KNR_pred.index, y='y_predict')
plt.tight_layout()
plt.show()
```




```
# Metriken
mae_KNR, rmse_KNR, r2_KNR, wmape_KNR, mase_KNR = ff.metryki(y_test,
y_predict_KNR)
print(f'MAE: {mae_KNR:.4f}, RMSE: {rmse_KNR:.4f}, R2: {r2_KNR:.4f},
WMAPE: {wmape_KNR:.4f}, MASE: {mase_KNR:.4f}')
```

```
MAE: 1385.8794, RMSE: 2331.3765, R2: 0.4135, WMAPE: 0.5490, MASE:
0.6765
```

SARIMAX

Die SARIMAX-Methode (Seasonal Autoregressive Integrated Moving Average with Exogenous Variables) ist eine fortgeschrittene Technik zur Modellierung und Vorhersage von Zeitreihen. Sie erweitert das klassische ARIMA-Modell (Autoregressive Integrated Moving Average) um die Berücksichtigung einer saisonalen Komponente sowie die Möglichkeit, exogene Variablen einzubeziehen.

Das SARIMAX-Modell besteht aus vier Hauptkomponenten:

Autoregression (**AR**): Die AR-Komponente bezieht sich auf die Abhängigkeit zwischen den Werten der Zeitreihe in der Vergangenheit, die die aktuellen Werte beeinflussen. AR spiegelt die Autokorrelation wider, also die Abhängigkeit zwischen den Werten der Zeitreihe zu verschiedenen Zeitpunkten.

Moving Average (**MA**): Die MA-Komponente bezieht sich auf die Abhängigkeit zwischen den Vorhersagefehlern in der Vergangenheit, die die Vorhersagefehler in der Gegenwart beeinflussen. MA spiegelt die Instabilität oder den Rausch in der Zeitreihe wider.

Integration (**I**): Die I-Komponente bezieht sich auf die Differenzierung der Zeitreihe, um sie stationär zu machen. Die Differenzierung besteht darin, die Differenzen zwischen den Werten der Zeitreihe zu verschiedenen Zeitpunkten zu berechnen. Der Differenzierungsprozess zielt darauf ab, den Trend und die Saisonalität aus der Zeitreihe zu entfernen.

Saisonale Komponente (**S**): Die saisonale Komponente bezieht sich auf wiederkehrende Muster oder Saisonalitäten in der Zeitreihe. Es kann notwendig sein, die Zyklen in den Daten zu berücksichtigen, zum Beispiel bei Zeitreihen mit jährlichen, quartalsweisen oder monatlichen Saisonalitäten.

Exogene Variablen (**X**): Externe Faktoren, die die Zeitreihe beeinflussen, aber nicht unmittelbar mit ihren vorherigen Werten zusammenhängen.

Prüfung der Stationarität der Daten

or der Verwendung des (S)ARIMA(X)-Modells ist es erforderlich, die Stationarität der Daten zu überprüfen. Dazu verwenden wir zwei Tests: den erweiterten Dickey-Fuller-Test (ADF) und den KPSS-Test, für die wir die folgenden Hypothesen aufstellen:

Erweiterter Dickey-Fuller-Test (ADF):

Nullhypothese (H0): Die Daten haben eine Einheitswurzel und sind somit nicht stationär.

Alternativhypothese (H1): Die Daten haben keine Einheitswurzel und sind somit stationär.

KPSS-Test:

Nullhypothese (H0): Die Daten sind in Bezug auf schwache Stationarität stationär (stationär im strengen Sinne oder mit Trend).

Alternativhypothese (H1): Die Daten sind nicht stationär.

```
# Erweiterter Dickey-Fuller-Test (ADF):
result_adf = adfuller(y_train)
print(f'ADF Statistic: {result_adf[0]}')
print(f'p-value: {result_adf[1]}')
```

```
ADF Statistic: -3.441555343129383
p-value: 0.00962123801316631
```

Bemerkung: Im Fall des ADF-Tests beträgt der p-Wert 0,0096, was kleiner ist als das Signifikanzniveau von 0,05. Das bedeutet, dass wir die Nullhypothese der Nicht-Stationarität ablehnen und akzeptieren, dass die Daten stationär sind.

```
# KPSS-Test
result_kpss = kpss(y_train)
print(f'KPSS Statistic: {result_kpss[0]}')
print(f'p-value: {result_kpss[1]}')
```

```
KPSS Statistic: 0.23627898343295728
p-value: 0.1

/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/
stattools.py:2022: InterpolationWarning: The test statistic is outside
of the range of p-values available in the
look-up table. The actual p-value is greater than the p-value
returned.
```

```
  warnings.warn(
```

Beobachtung: Im Fall des KPSS-Tests beträgt der p-Wert 0,236, was größer ist als das Signifikanzniveau von 0,05. Das bedeutet, dass es keine Grundlage gibt, die Nullhypothese abzulehnen, die besagt, dass die Daten stationär sind.

Zusammenfassend deuten die Ergebnisse beider Tests darauf hin, dass die Daten stationär sind. Es ist jedoch zu beachten, dass der KPSS-Test eine Warnung generiert, dass der Teststatistikwert außerhalb des Bereichs der verfügbaren p-Werte in der Tabelle liegt. In solchen Fällen kann der p-Wert ungenau sein. Dennoch weisen die Ergebnisse beider Tests auf die Stationarität der Daten hin.

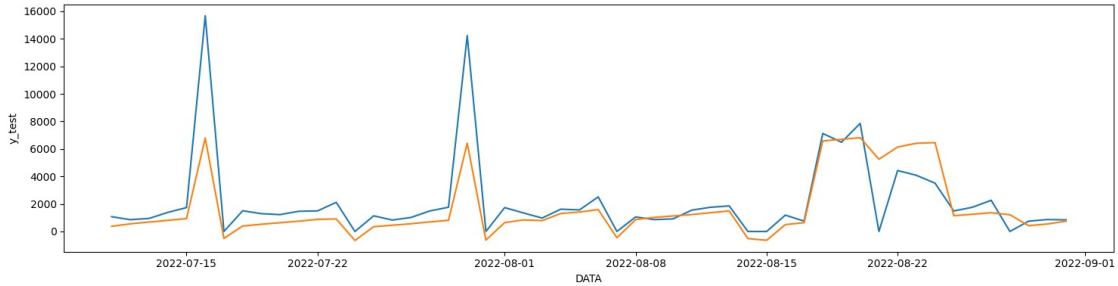
Modellbau

```
# Initialisierung eines Datenrahmens, um die Ergebnisse der Metriken
für jedes Folio zu speichern
results_df_SARIMAX = pd.DataFrame(columns=['Fold', 'order',
'seasonal_order', 'AIC', 'BIC', 'MAE', 'RMSE', 'R2', 'WMAPE', 'MASE'])

# TimeSeriesSplit-Objekt erstellen
tscv = TimeSeriesSplit(n_splits=5)

# Iteration nach Fold's
for fold, (train_index, test_index) in enumerate(tscv.split(X_train)):
    X_train_SARIMAX = X_train.iloc[train_index]
    X_test_SARIMAX = X_train.iloc[test_index]
    y_train_SARIMAX = y_train.iloc[train_index]
    y_test_SARIMAX = y_train.iloc[test_index]

    # AutoARIMA - automatische Anpassung des SARIMAX-Modells
    SARIMAX = auto_arima(X=X_train_SARIMAX, y=y_train_SARIMAX,
error_action='ignore')

    # Wertvorhersage
    y_pred = SARIMAX.predict(n_periods=len(X_test_SARIMAX),
X=X_test_SARIMAX)

    # Berechnung von Metriken
    mae, rmse, r2, wmape, mase = ff.metryki(y_test_SARIMAX, y_pred)

    # Aufzeichnung der Ergebnisse der Metriken für Fold's
    results_df_SARIMAX = results_df_SARIMAX.append({'Fold': fold,
'order': SARIMAX.order, 'seasonal_order': SARIMAX.seasonal_order,
                                                    'AIC':
SARIMAX.aic(), 'BIC': SARIMAX.bic(),
                                                    'MAE': mae,
'RMSE': rmse, 'R2': r2, 'WMAPE': wmape, 'MASE': mase},
                                                   ignore_index=True)
```

```
# DataFrame mit den Ergebnissen der Metriken für jeden Fold
results_df_SARIMAX.head()
```

```
  Fold      order seasonal_order           AIC           BIC
MAE  \
0    0  (0, 0, 0)   (0, 0, 0, 0)   2738.998285   2789.906821
10182.373648
1    1  (1, 0, 2)   (0, 0, 0, 0)   6004.571121   6077.926059
809.427482
2    2  (0, 0, 1)   (0, 0, 0, 0)   8859.160211   8931.605570
4517.155896
3    3  (0, 0, 1)   (0, 0, 0, 0)  11772.891223  11850.186575
830.723318
4    4  (1, 0, 0)   (0, 0, 0, 0)  14814.042450  14895.106835
624.099708

           RMSE          R2      WMAPE      MASE
0  33876.207769 -145.673841   1.599138  8.209899
1   1080.319862    0.700385   7.151459  0.939896
2  16467.747114  -23.299269   1.110639  3.826659
3   1581.596813    0.562627   9.622367  0.767895
4    978.420573    0.674121  17.282548  0.866644
```

```
# Forecast
y_predict_SARIMAX = SARIMAX.predict(n_periods=len(y_test), X=X_test)

# Erstellung eines DataFrame zur Visualisierung
SARIMAX_pred = df.loc[df.index >'2022-07-10 00:00:00'][['y']]
SARIMAX_pred['y_test'] = SARIMAX_pred['y']
SARIMAX_pred = SARIMAX_pred.drop('y', axis=1)
SARIMAX_pred = SARIMAX_pred.reset_index()
SARIMAX_pred['y_predict'] = y_predict_SARIMAX.reset_index(drop=True)

# Grafik - Vergleich der Vorhersagen
plt.figure(figsize=(15, 4))
sns.lineplot(data=SARIMAX_pred, x=SARIMAX_pred.DATA, y='y_test')
sns.lineplot(data=SARIMAX_pred, x=SARIMAX_pred.DATA, y='y_predict')
plt.tight_layout()
plt.show()
```




```
# Metiken
mae_SARIMAX, rmse_SARIMAX, r2_SARIMAX, wmape_SARIMAX, mase_SARIMAX =
```

```
ff.metryki(SARIMAX_pred['y_test'], SARIMAX_pred['y_predict'])
print(f'MAE: {mae_SARIMAX:.4f}, RMSE: {rmse_SARIMAX:.4f}, R2:
{r2_SARIMAX:.4f}, WMAPE: {wmape_SARIMAX:.4f}, MASE:
{mase_SARIMAX:.4f}')
```

```
MAE: 1047.7152, RMSE: 1975.9936, R2: 0.5787, WMAPE: 0.3844, MASE:
0.5114
```

Prophet

Das Prophet-Modell ist ein Werkzeug zur Vorhersage von Zeitreihendaten, das von Facebook entwickelt wurde. Es wird verwendet, um Trends und Saisonalitäten in Zeitreihendaten vorherzusagen, insbesondere bei Daten mit wiederkehrenden Mustern.

Ein charakteristisches Merkmal des Prophet-Modells ist seine Benutzerfreundlichkeit und Flexibilität. Es kann zur Vorhersage sowohl kurz- als auch langfristiger Trends verwendet werden und komplexe Saisonalitäten wie tägliche, wöchentliche oder jährliche Saisonalitäten berücksichtigen.

Das Prophet-Modell basiert auf einer Kombination von zwei Komponenten:

Saisonale Trends: Das Prophet-Modell erkennt automatisch und berücksichtigt saisonale Muster in den Daten, indem es sich wiederholende Muster im zeitlichen Verlauf identifiziert. Dies kann tägliche, wöchentliche, monatliche, quartalsweise oder jährliche Saisonalitäten umfassen. Das Modell ist in der Lage, sich flexibel an verschiedene Arten von Saisonalitäten anzupassen und sie in den Prognosen zu berücksichtigen.

Zusätzliche Komponenten: Neben der Saisonalität kann das Prophet-Modell auch andere Faktoren berücksichtigen, die Einfluss auf die Zeitreihendaten haben können. Dies können exogene Variablen wie Wetterdaten, Feiertage, Werbeaktionen usw. sein. Das Modell passt sich automatisch an diese Faktoren an und berücksichtigt sie in den Prognosen.

```
# Parameter
seasonality_prior_scale = [10, 20, 30]
holidays_prior_scale = [10, 20, 30]
changepoint_prior_scale = [0.01, 0.05, 0.1]
growth = ['linear']

# Wörterbuch der polnischen Feiertage für 2019-2022
years = [2019, 2020, 2021, 2022]
holidays = holidays.PL(years=years)

holidays = pd.DataFrame({
    'ds': list(holidays.keys()),
    'holiday': list(holidays.values())
})

# Trainings- und Testdaten
df_prophet = df.reset_index().rename(columns={'DATA': 'ds'})
```

```
train = df_prophet[df_prophet.ds < '2022-07-10']
test = df_prophet[df_prophet.ds >= '2022-07-10']

# Erstellung von Parameterkombinationen
param_combinations_prophet = list(product(seasonality_prior_scale,
holidays_prior_scale, changepoint_prior_scale, growth))
len(param_combinations_prophet)
```

```
27
```

```
# Initialisierung des DataFrame's mit Ergebnissen
results_df_prophet = pd.DataFrame(columns=['Fold',
'seasonality_prior_scale', 'holidays_prior_scale',
                                           'changepoint_prior_scale',
'MAE', 'RMSE', 'R2', 'WMAPE', 'MASE'])

# Erstellung eines TimeSeriesSplit-Objekts
tscv = TimeSeriesSplit(n_splits=5)

# Iterationt durch Fold
for fold, (train_index, test_index) in enumerate(tscv.split(train)):
    train_data = train.iloc[train_index]
    test_data = train.iloc[test_index]

    best_rmse = float('inf')
    best_params = None

    # Iteration durch Parameterkombinationen
    for params in param_combinations_prophet:
        seasonality_prior_scale, holidays_prior_scale,
changepoint_prior_scale, growth = params

        prophet_model = Prophet(growth=growth,
                                yearly_seasonality=True,
                                weekly_seasonality=True,
                                daily_seasonality=False,
                                holidays=holidays,
                                seasonality_mode='multiplicative',

seasonality_prior_scale=seasonality_prior_scale,

holidays_prior_scale=holidays_prior_scale,

changepoint_prior_scale=changepoint_prior_scale)

        prophet_model.add_regressor('CENA_NP')
        prophet_model.add_regressor('CENA_AP')
        prophet_model.add_regressor('PRZECENA')
        prophet_model.add_regressor('CENA_MARKET')
        prophet_model.add_regressor('PROMO_KOD')
```

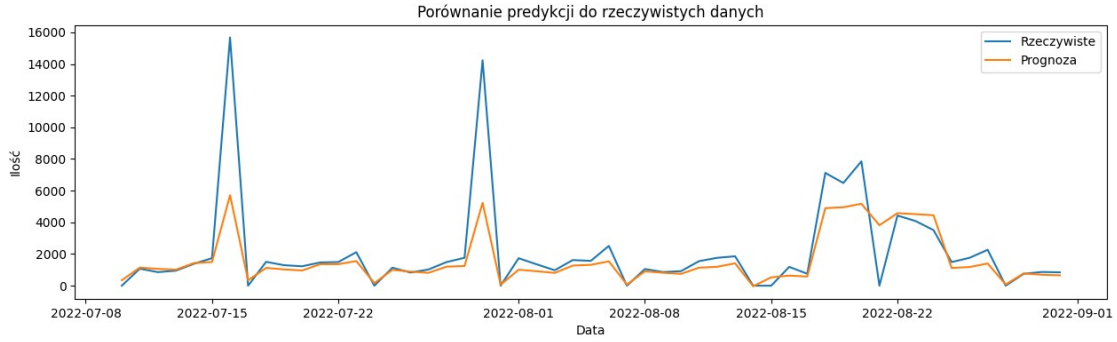
```
        prophet_model.add_regressor('Nr_dn_tyg')
        prophet_model.add_regressor('Nr_dn_mies')
        prophet_model.add_regressor('Nr_mies')
        prophet_model.add_regressor('Nr_tyg')
        prophet_model.add_regressor('Nr_rok')
        prophet_model.add_regressor('Dn_handlowy')
        prophet_model.add_regressor('Hot_day')
        prophet_model.add_regressor('Hot_day_Xmass')
        prophet_model.add_regressor('Hot_day_Wlkn')
        prophet_model.add_regressor('HICP')

        prophet_model.fit(train_data)

        future = prophet_model.predict(test_data)

        y_pred_prophet = future[-len(test_data):]['yhat']
        rmse = mean_squared_error(test_data['y'], y_pred_prophet,
squared=False)

        if rmse < best_rmse:
            best_rmse = rmse
            best_params = params

    # Das beste Modell erstellen
    seasonality_prior_scale, holidays_prior_scale,
changepoint_prior_scale, growth = best_params

    prophet_model = Prophet(growth=growth,
                            yearly_seasonality=True,
                            weekly_seasonality=True,
                            daily_seasonality=False,
                            holidays=holidays,
                            seasonality_mode='multiplicative',

seasonality_prior_scale=seasonality_prior_scale,
                            holidays_prior_scale=holidays_prior_scale,

changepoint_prior_scale=changepoint_prior_scale)

    prophet_model.add_regressor('CENA_NP')
    prophet_model.add_regressor('CENA_AP')
    prophet_model.add_regressor('PRZECENA')
    prophet_model.add_regressor('CENA_MARKET')
    prophet_model.add_regressor('PROMO_KOD')
    prophet_model.add_regressor('Nr_dn_tyg')
    prophet_model.add_regressor('Nr_dn_mies')
    prophet_model.add_regressor('Nr_mies')
    prophet_model.add_regressor('Nr_tyg')
    prophet_model.add_regressor('Nr_rok')
```

```
    prophet_model.add_regressor('Dn_handlowy')
    prophet_model.add_regressor('Hot_day')
    prophet_model.add_regressor('Hot_day_Xmass')
    prophet_model.add_regressor('Hot_day_Wlkn')
    prophet_model.add_regressor('HICP')

    prophet_model.fit(train_data)

    future = prophet_model.predict(test_data)

    y_pred_prophet = future[-len(test_data):]['yhat']

    # Berechnung von Metriken
    y_true = test_data['y']
    y_true.reset_index(drop=True, inplace=True)
    best_mae, best_rmse, best_r2, best_wmape, best_mase =
ff.metryki(y_true, y_pred_prophet)

    # Hinzufügen der besten Ergebnisse der Metrik zum Datenrahmen
    results_df_prophet = results_df_prophet.append({'Fold': fold,
'seasonality_prior_scale': seasonality_prior_scale,

'holidays_prior_scale': holidays_prior_scale,

'changepoint_prior_scale': changepoint_prior_scale,
                                                    'MAE': best_mae,
'RMSE': best_rmse,
                                                    'R2': best_r2,
'WMAPE': best_wmape, 'MASE': best_mase},
                                                   ignore_index=True)

# Anzeige eines Datenrahmens mit den Ergebnissen der Metriken für
jeden Fold
results_df_prophet.head()
```

```
   Fold  seasonality_prior_scale  holidays_prior_scale  \
0   0.0                     20.0                  20.0
1   1.0                     30.0                  20.0
2   2.0                     20.0                  20.0
3   3.0                     20.0                  20.0
4   4.0                     10.0                  30.0

   changepoint_prior_scale          MAE          RMSE         R2
WMAPE  \
0                     0.10  1528.769416   3129.464153  -0.284223
0.748248
1                     0.01   715.078516    948.223160   0.784146
4.191848
2                     0.01  6202.670963  22696.872014 -47.840726
1.420975
```

```
3                    0.01   667.500355   1439.723685   0.673802
9.223618
4                    0.01   458.954458    816.267106   0.773063
4.338474

       MASE
0  1.278107
1  0.817585
2  5.562637
3  0.608982
4  0.646593
```

```python
# DataFrame für die Vorhersage
test_prophet = test.drop('y', axis=1)
test_prophet.reset_index(drop=True, inplace=True)

# Forecast
prophet_forecast = prophet_model.predict(test_prophet)

# Erstellung eines DataFrame zur Visualisierung
y_true = test['y'].reset_index(drop=True)
y_pred = prophet_forecast['yhat'].reset_index(drop=True)
result_df = pd.DataFrame({'y_true': y_true, 'y_pred': y_pred})

# Grafik - Vergleich der Vorhersagen
plt.figure(figsize=(15, 4))
plt.plot(test.ds, result_df['y_true'], label='Rzeczywiste')
plt.plot(test.ds, result_df['y_pred'], label='Prognoza')
plt.xlabel('Data')
plt.ylabel('Ilość')
plt.title('Porównanie predykcji do rzeczywistych danych')
plt.legend()
plt.show()
```




```python
# Metriken
mae_prophet, rmse_prophet, r2_prophet, wmape_prophet, mase_prophet =
ff.metryki(y_true, y_pred)
print(f'MAE: {mae_prophet:.4f}, RMSE: {rmse_prophet:.4f}, R2:
{r2_prophet:.4f}, WMAPE: {wmape_prophet:.4f}, MASE:
{mase_prophet:.4f}')
```

```
MAE: 821.7948, RMSE: 2020.3256, R2: 0.5553, WMAPE: 0.2295, MASE:
0.4048
```

XGBoost

XGBoost (Extreme Gradient Boosting) ist ein fortschrittliches maschinelles Lernmodell, das die Gradient Boosting-Technik zur Vorhersage oder Klassifizierung von Daten verwendet. Es ist eine der beliebtesten und effektivsten Techniken in maschinellem Lernen-Wettbewerben.

XGBoost basiert auf der Kombination vieler schwacher Modelle (meistens Entscheidungsbäume), um ein starkes Vorhersagemodell zu erstellen. Der Prozess des Trainings des XGBoost-Modells erfolgt iterativ, wobei jedes nachfolgende Modell angepasst wird, um Fehler der vorherigen Modelle zu korrigieren. Auf diese Weise konzentriert sich das XGBoost-Modell auf die Vorhersage von Residuen (d.h. die Differenz zwischen den tatsächlichen Werten und den vorhergesagten Werten), um eine immer bessere Anpassung an die Daten zu erreichen.

XGBoost verwendet eine Verlustfunktion (Loss Function) wie den mittleren quadratischen Fehler (MSE) oder den logarithmischen Verlust (Log Loss), um den Fehler des Modells in jeder Iteration zu bewerten. Basierend auf dieser Verlustfunktion werden die Gewichte der Entscheidungsbäume aktualisiert, um den Vorhersagefehler zu minimieren.

Vorbereitung der Daten

```
# Laden von Daten
df = pd.read_pickle('PICKLE/df_3.pkl')
# Beschränkung der Daten auf ein Cluster und Auswahl eines Artikels
df = df.loc[df['CLUSTER']==2].drop(['CLUSTER'], axis=1)
df = df.loc[df['ART_ID']==158].drop(['ART_ID', 'GRUPA_ID', 'NAZWA'],
axis=1)
df.set_index('DATA', inplace=True)
# Copy
df_XGB = df.copy()

# Definition von Spalten pro Datentyp
nominal_columns = ['PROMO_KOD', 'Dn_handlowy', 'Hot_day']
ordinal_columns = ['Hot_day_Xmass', 'Hot_day_Wlkn', 'Nr_dn_tyg',
'Nr_dn_mies', 'Nr_mies', 'Nr_tyg', 'Nr_rok']
numeric_columns = ['CENA_MARKET', 'CENA_AP', 'CENA_NP', 'PRZECENA',
'HICP']

# Erstellung von Transformationen
nominal_transformer = OneHotEncoder(handle_unknown='ignore')
ordinal_transformer =
OrdinalEncoder(handle_unknown='use_encoded_value', unknown_value=-1)
numeric_transformer = MinMaxScaler()

# Erstellung von ColumnTransformer
preprocessor = ColumnTransformer(
    transformers=[
```

```
        ('nominal', nominal_transformer, nominal_columns),
        ('ordinal', ordinal_transformer, ordinal_columns),
        ('numeric', numeric_transformer, numeric_columns)
    ])

# Erstellung der Pipeline
pipeline = Pipeline(steps=[('preprocessor', preprocessor)])

# Umwandlung von Daten
X_train, y_train = df_XGB.loc[df_XGB.index <='2022-07-10 00:00:00'],
df_XGB.loc[df_XGB.index <= '2022-07-10 00:00:00']
X_test, y_test = df_XGB.loc[df_XGB.index>'2022-07-10 00:00:00'],
df_XGB.loc[df_XGB.index > '2022-07-10 00:00:00']
X_train = X_train.drop(columns=['y'])
X_test = X_test.drop(columns=['y'])
y_train = y_train['y']
y_test = y_test['y']
```

Gradient Boosting

```
# Parameter
n_estimators_values = [50, 100, 300, 500, 1000, 1500]
learning_rate_values = [0.01, 0.02, 0.05, 0.1, 0.2]
max_depth_values = [2, 4, 6, 8, 10, 12]

# Erstellung von Parameterkombinationen
param_combinations_XGB = list(product(n_estimators_values,
learning_rate_values, max_depth_values))
len(param_combinations_XGB)
```

```
144
```

```
# Initialisierung eines Datenrahmens zur Speicherung der Ergebnisse
der Metriken für jeden Fold
results_df_XGB = pd.DataFrame(columns=['Fold', 'n_estimators',
'learning_rate', 'max_depth', 'MAE', 'RMSE', 'R2', 'WMAPE', 'MASE'])

# Erstellung eines TimeSeriesSplit-Objekts
tscv = TimeSeriesSplit(n_splits=5)

# Iterationt durch Fold
for fold, (train_index, test_index) in enumerate(tscv.split(X_train)):
    X_train_XGB = X_train.iloc[train_index]
    X_test_XGB = X_train.iloc[test_index]
    y_train_XGB = y_train.iloc[train_index]
    y_test_XGB = y_train.iloc[test_index]
    # Kodierung
    X_train_XGB = pipeline.fit_transform(X_train_XGB)
    X_test_XGB = pipeline.transform(X_test_XGB)

    best_rmse = float('inf')
    best_params = None
```

```
    # Iteration durch Parameterkombinationen
    for params in param_combinations_XGB:
        n_estimators, learning_rate, max_depth = params
        XGB = xgb.XGBRegressor(objective='reg:squarederror',
                               n_jobs=-1,
                               random_state=42,
                               n_estimators=n_estimators,
                               learning_rate=learning_rate,
                               max_depth=max_depth,
                               tree_method='gpu_hist')
        XGB.fit(X_train_XGB, y_train_XGB)
        y_pred_XGB = XGB.predict(X_test_XGB)
        rmse = mean_squared_error(y_test_XGB, y_pred_XGB,
squared=False)

        if rmse < best_rmse:
            best_rmse = rmse
            best_params = params

    # Das beste Modell erstellen
    n_estimators, learning_rate, max_depth = best_params
    XGB = xgb.XGBRegressor(objective='reg:squarederror',
                           n_jobs=-1,
                           random_state=42,
                           n_estimators=n_estimators,
                           learning_rate=learning_rate,
                           max_depth=max_depth,
                           tree_method='gpu_hist')
    XGB.fit(X_train_XGB, y_train_XGB)
    y_pred_XGB = XGB.predict(X_test_XGB)

    # Berechnung von Metriken
    best_mae, best_rmse, best_r2, best_wmape, best_mase =
ff.metryki(y_test_XGB, y_pred_XGB)

    # Hinzufügen der besten Ergebnisse der Metrik zum Datenrahmen
    results_df_XGB = results_df_XGB.append({'Fold': fold,
'n_estimators': n_estimators,

'learning_rate': learning_rate, 'max_depth': max_depth,
                                                        'MAE':
best_mae, 'RMSE': best_rmse, 'R2': best_r2,
                                                        'WMAPE':
best_wmape, 'MASE': best_mase}, ignore_index=True)

# Anzeige eines Datenrahmens mit den Ergebnissen der Metriken für
jeden Fold
results_df_XGB.head()
```

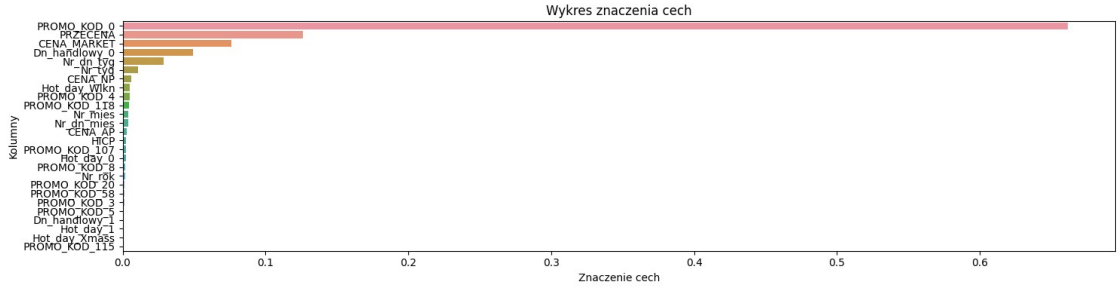
```
   Fold  n_estimators  learning_rate  max_depth         MAE
RMSE  \
0   0.0        1500.0           0.02        3.0  932.726440
1841.917847
1   1.0         500.0           0.02        3.0  404.422729
648.143066
2   2.0        1500.0           0.02        5.0  636.859192
1515.611084
3   3.0        1500.0           0.01        3.0  367.920868
659.642700
4   4.0         500.0           0.02        3.0  360.288879
704.063232

         R2 MAPE      MASE     WMAPE
0  0.566386  NaN  0.752044  0.394682
1  0.892155  NaN  0.469610  0.878559
2  0.794174  NaN  0.539508  0.332802
3  0.923919  NaN  0.340095  2.152482
4  0.831256  NaN  0.500308  1.921122
```

```
# Die besten Parameter fuer XGBRegressor
print('Najlepsze parametry znalezione:', XGB.get_params())
```

```
Najlepsze parametry znalezione: {'objective': 'reg:squarederror',
'base_score': None, 'booster': None, 'callbacks': None,
'colsample_bylevel': None, 'colsample_bynode': None,
'colsample_bytree': None, 'early_stopping_rounds': None,
'enable_categorical': False, 'eval_metric': None, 'feature_types':
None, 'gamma': None, 'gpu_id': None, 'grow_policy': None,
'importance_type': None, 'interaction_constraints': None,
'learning_rate': 0.02, 'max_bin': None, 'max_cat_threshold': None,
'max_cat_to_onehot': None, 'max_delta_step': None, 'max_depth': 3,
'max_leaves': None, 'min_child_weight': None, 'missing': nan,
'monotone_constraints': None, 'n_estimators': 500, 'n_jobs': -1,
'num_parallel_tree': None, 'predictor': None, 'random_state': 42,
'reg_alpha': None, 'reg_lambda': None, 'sampling_method': None,
'scale_pos_weight': None, 'subsample': None, 'tree_method': None,
'validate_parameters': None, 'verbosity': None}
```

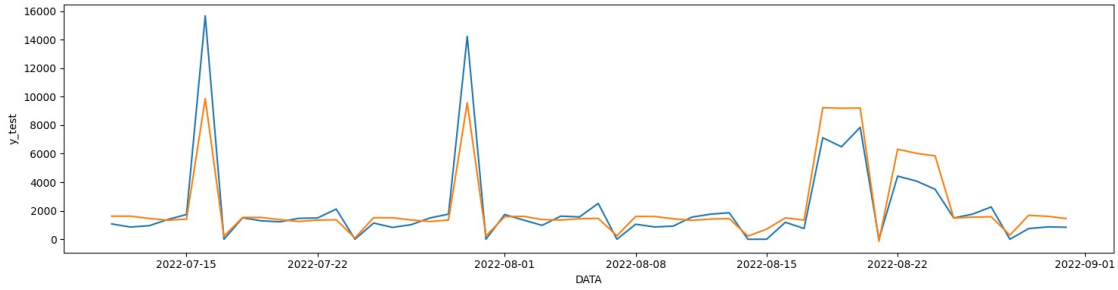
```
# Extrahieren der Merkmalsbedeutung aus dem XGBoost-Modell.
importance_scores = XGB.feature_importances_

# Erstellen eines Wörterbuchs, das die Indizes der Spalten auf die
ursprünglichen Namen abbildet.
column_names = []
column_transformers = preprocessor.transformers_
for transformer_name, transformer, column_indices in
column_transformers:
    if transformer_name == 'nominal':
        nominal_column_names =
transformer.get_feature_names_out(nominal_columns)
```

```
        column_names.extend(nominal_column_names)
    elif transformer_name == 'ordinal':
        ordinal_column_names = column_indices
        column_names.extend(ordinal_column_names)
    elif transformer_name == 'numeric':
        numeric_column_names = numeric_columns
        column_names.extend(numeric_column_names)

# Sortieren von Merkmalen nach ihrer Bedeutung
sorted_indices = importance_scores.argsort()[::-1]
sorted_column_names = [column_names[i] for i in sorted_indices]
sorted_importance_scores = importance_scores[sorted_indices]

# Erzeugen eines Diagramms zur Bedeutung von Merkmalen mit
benutzerdefinierten Beschriftungen
plt.figure(figsize=(15, 4))
sns.barplot(x=sorted_importance_scores, y=sorted_column_names,
orient='h')
plt.xlabel('Znaczenie cech')
plt.ylabel('Kolumny')
plt.title('Wykres znaczenia cech')
plt.tight_layout()
plt.show()
```




```
# Kodierung
X_train_XGB = pipeline.fit_transform(X_train)
X_test_XGB = pipeline.transform(X_test)
# Erstellungen von Predictions
XGB.fit(X_train_XGB, y_train)
y_predict_xgb = XGB.predict(X_test_XGB)

# Erstellung eines DataFrame zur Visualisierung
xgb_pred = df.loc[df.index >'2022-07-10 00:00:00'][['y']]
xgb_pred['y_test'] = xgb_pred['y']
xgb_pred = xgb_pred.drop('y', axis=1)
xgb_pred['y_predict'] = y_predict_xgb

# Grafik - Vergleich der Vorhersagen
plt.figure(figsize=(15, 4))
sns.lineplot(data=xgb_pred, x=xgb_pred.index, y='y_test')
sns.lineplot(data=xgb_pred, x=xgb_pred.index, y='y_predict')
```

```
plt.tight_layout()
plt.show()
```




```
# Metriken
mae_XGB, rmse_XGB, r2_XGB, wmape_XGB, mase_XGB = ff.metryki(y_test,
y_predict_xgb)
print(f'MAE: {mae_XGB:.4f}, RMSE: {rmse_XGB:.4f}, R2: {r2_XGB:.4f},
WMAPE: {wmape_XGB:.4f}, MASE: {mase_XGB:.4f}')
```

```
MAE: 766.6458, RMSE: 1325.4833, R2: 0.8104, WMAPE: 0.3697%, MASE:
0.3742
```

Prophet + XGBoost

Die Methode, die das XGBoost-Modell mit dem Prophet-Modell verbindet, besteht darin, die Vorhersagen des Prophet-Modells als zusätzliche Merkmale (exogene Variablen) für das XGBoost-Modell zu verwenden. Dieser Prozess zielt darauf ab, die ergänzenden Fähigkeiten beider Modelle zu nutzen und die Qualität der Prognosen zu verbessern.

Die Schritte, die unternommen werden können, um diese Methode anzuwenden, sind:

1. Training des Prophet-Modells: Zunächst trainieren wir das Prophet-Modell mit den verfügbaren Zeitreihendaten. Das Modell kann saisonale Muster, Trends und andere Faktoren berücksichtigen, die sich auf die Zeitreihe auswirken.

2. Generierung von Prophet-Modellvorhersagen: Nach dem Training des Prophet-Modells generieren wir Vorhersagen für zukünftige Zeitpunkte. Diese Vorhersagen enthalten Informationen über die erwarteten Werte der Zeitreihe basierend auf dem Prophet-Modell.

3. Auswahl geeigneter Merkmale: Anschließend wählen wir einige Spalten aus den Vorhersagen des Prophet-Modells aus, die wir für relevant halten und die als zusätzliche Merkmale für das XGBoost-Modell dienen können. Dies können beispielsweise Vorhersagewerte für verschiedene Zeiträume oder Unterschiede zwischen den vorhergesagten und den tatsächlichen Werten sein.

4. Verbindung mit dem XGBoost-Modell: Nach der Auswahl geeigneter Merkmale aus den Vorhersagen des Prophet-Modells werden diese Merkmale dem Trainingsdatensatz des XGBoost-Modells hinzugefügt. Anschließend wird das XGBoost-Modell mit diesen Daten trainiert und verwendet die zusätzlichen Merkmale als exogene Variablen.

5. Prognose mit dem kombinierten Modell: Nach dem Training des XGBoost-Modells mit den zusätzlichen Merkmalen können wir es verwenden, um zukünftige Werte der Zeitreihe vorherzusagen. Das XGBoost-Modell nutzt sowohl Informationen aus historischen Daten als auch aus den zusätzlichen Merkmalen des Prophet-Modells, um präzisere Prognosen zu liefern.

Diese Methode kombiniert die Fähigkeit des Prophet-Modells, Trends und Saisonalität vorherzusagen, mit der starken prädiktiven Fähigkeit des XGBoost-Modells. Durch die Nutzung von Informationen aus beiden Modellen können vielseitigere und genauere Zeitreihenprognosen erstellt werden.

```
# Laden von Daten
df = pd.read_pickle('PICKLE/df_3.pkl')
# Beschränkung der Daten auf ein Cluster und Auswahl eines Artikels
df = df.loc[df['CLUSTER']==2].drop(['CLUSTER'], axis=1)
df = df.loc[df['ART_ID']==158].drop(['ART_ID', 'GRUPA_ID', 'NAZWA'],
axis=1)
df.set_index('DATA', inplace=True)
# Copy
df_XGB_Prophet = df.copy()

# Forecast
forecast = prophet_model.predict(df_prophet.iloc[:,:-1])
forecast = forecast.set_index('ds').rename_axis('DATA', axis='index')
forecast.head(3)
```

```
                  trend  yhat_lower   yhat_upper  trend_lower
trend_upper  \
DATA

2019-09-02  1519.250927 -313.655182  2835.032926  1519.250927
1519.250927
2019-09-03  1518.997429 -366.548872  2768.541495  1518.997429
1518.997429
2019-09-04  1518.743932 -450.753683  2583.407650  1518.743932
1518.743932

            All Saints' Day  All Saints' Day_lower  All Saints'
Day_upper  \
DATA

2019-09-02              0.0                    0.0
0.0
2019-09-03              0.0                    0.0
0.0
2019-09-04              0.0                    0.0
0.0

            Assumption of the Virgin Mary  \
DATA
```

```
2019-09-02                              0.0
2019-09-03                              0.0
2019-09-04                              0.0

            Assumption of the Virgin Mary_lower  \
DATA
2019-09-02                                   0.0
2019-09-03                                   0.0
2019-09-04                                   0.0

            Assumption of the Virgin Mary_upper   CENA_AP
CENA_AP_lower  \
DATA

2019-09-02                                   0.0  0.168216
0.168216
2019-09-03                                   0.0  0.168216
0.168216
2019-09-04                                   0.0  0.168216
0.168216

            CENA_AP_upper  CENA_MARKET  CENA_MARKET_lower
CENA_MARKET_upper  \
DATA

2019-09-02       0.168216     0.336430           0.336430
0.336430
2019-09-03       0.168216     0.391699           0.391699
0.391699
2019-09-04       0.168216     0.347944           0.347944
0.347944

             CENA_NP  CENA_NP_lower  CENA_NP_upper  Christmas (Day 1)
\
DATA

2019-09-02 -0.109604      -0.109604      -0.109604                0.0

2019-09-03 -0.109604      -0.109604      -0.109604                0.0

2019-09-04 -0.109604      -0.109604      -0.109604                0.0


            Christmas (Day 1)_lower  Christmas (Day 1)_upper  \
DATA
2019-09-02                      0.0                      0.0
2019-09-03                      0.0                      0.0
2019-09-04                      0.0                      0.0
```

```
            Christmas (Day 2)  Christmas (Day 2)_lower  \
DATA
2019-09-02                0.0                      0.0
2019-09-03                0.0                      0.0
2019-09-04                0.0                      0.0

            Christmas (Day 2)_upper  Corpus Christi  Corpus
Christi_lower  \
DATA

2019-09-02                      0.0             0.0
0.0
2019-09-03                      0.0             0.0
0.0
2019-09-04                      0.0             0.0
0.0

            Corpus Christi_upper  Dn_handlowy  Dn_handlowy_lower  \
DATA
2019-09-02                   0.0     0.068431           0.068431
2019-09-03                   0.0     0.068431           0.068431
2019-09-04                   0.0     0.068431           0.068431

            Dn_handlowy_upper  Easter Monday  Easter Monday_lower  \
DATA
2019-09-02           0.068431            0.0                  0.0
2019-09-03           0.068431            0.0                  0.0
2019-09-04           0.068431            0.0                  0.0

            Easter Monday_upper  Easter Sunday  Easter Sunday_lower  \
DATA
2019-09-02                  0.0            0.0                  0.0
2019-09-03                  0.0            0.0                  0.0
2019-09-04                  0.0            0.0                  0.0

            Easter Sunday_upper  Epiphany  Epiphany_lower
Epiphany_upper  \
DATA

2019-09-02                  0.0       0.0             0.0
0.0
2019-09-03                  0.0       0.0             0.0
0.0
2019-09-04                  0.0       0.0             0.0
0.0

               HICP  HICP_lower  HICP_upper  Hot_day  Hot_day_lower  \
DATA
2019-09-02 -0.02847    -0.02847    -0.02847      0.0            0.0
```

```
2019-09-03 -0.02847    -0.02847    -0.02847      0.0            0.0
2019-09-04 -0.02847    -0.02847    -0.02847      0.0            0.0

            Hot_day_upper  Hot_day_Wlkn  Hot_day_Wlkn_lower  \
DATA
2019-09-02            0.0     -0.001039           -0.001039
2019-09-03            0.0     -0.001039           -0.001039
2019-09-04            0.0     -0.001039           -0.001039

            Hot_day_Wlkn_upper  Hot_day_Xmass  Hot_day_Xmass_lower  \
DATA
2019-09-02           -0.001039       0.001316             0.001316
2019-09-03           -0.001039       0.001316             0.001316
2019-09-04           -0.001039       0.001316             0.001316

            Hot_day_Xmass_upper  National Day  National Day_lower  \
DATA
2019-09-02             0.001316           0.0                 0.0
2019-09-03             0.001316           0.0                 0.0
2019-09-04             0.001316           0.0                 0.0

            National Day_upper  National Day of the Third of May  \
DATA
2019-09-02                 0.0                                0.0
2019-09-03                 0.0                                0.0
2019-09-04                 0.0                                0.0

            National Day of the Third of May_lower  \
DATA
2019-09-02                                     0.0
2019-09-03                                     0.0
2019-09-04                                     0.0

            National Day of the Third of May_upper  National
Independence Day  \
DATA

2019-09-02                                     0.0
0.0
2019-09-03                                     0.0
0.0
2019-09-04                                     0.0
0.0

            National Independence Day_lower  National Independence
Day_upper  \
DATA

2019-09-02                              0.0
```

```
0.0
2019-09-03                              0.0
0.0
2019-09-04                              0.0
0.0

            New Year's Day  New Year's Day_lower  New Year's Day_upper
\
DATA

2019-09-02             0.0                   0.0                   0.0

2019-09-03             0.0                   0.0                   0.0

2019-09-04             0.0                   0.0                   0.0


            Nr_dn_mies  Nr_dn_mies_lower  Nr_dn_mies_upper  Nr_dn_tyg
\
DATA

2019-09-02    0.069619          0.069619          0.069619   0.024560

2019-09-03    0.064509          0.064509          0.064509   0.016364

2019-09-04    0.059400          0.059400          0.059400   0.008168


            Nr_dn_tyg_lower  Nr_dn_tyg_upper   Nr_mies  Nr_mies_lower
\
DATA

2019-09-02         0.024560         0.024560 -0.005619      -0.005619

2019-09-03         0.016364         0.016364 -0.005619      -0.005619

2019-09-04         0.008168         0.008168 -0.005619      -0.005619


            Nr_mies_upper    Nr_rok  Nr_rok_lower  Nr_rok_upper
Nr_tyg  \
DATA

2019-09-02      -0.005619 -0.052208     -0.052208     -0.052208 -
0.010759
2019-09-03      -0.005619 -0.052208     -0.052208     -0.052208 -
0.010759
2019-09-04      -0.005619 -0.052208     -0.052208     -0.052208 -
```

```
0.010759

            Nr_tyg_lower  Nr_tyg_upper  PROMO_KOD  PROMO_KOD_lower  \
DATA
2019-09-02     -0.010759     -0.010759  -0.101946        -0.101946
2019-09-03     -0.010759     -0.010759  -0.101946        -0.101946
2019-09-04     -0.010759     -0.010759  -0.101946        -0.101946

            PROMO_KOD_upper  PRZECENA  PRZECENA_lower  PRZECENA_upper
\
DATA

2019-09-02        -0.101946 -0.333264       -0.333264       -0.333264

2019-09-03        -0.101946 -0.333264       -0.333264       -0.333264

2019-09-04        -0.101946 -0.333264       -0.333264       -0.333264


            Pentecost  Pentecost_lower  Pentecost_upper  \
DATA
2019-09-02        0.0              0.0              0.0
2019-09-03        0.0              0.0              0.0
2019-09-04        0.0              0.0              0.0

            extra_regressors_multiplicative  \
DATA
2019-09-02                         0.025662
2019-09-03                         0.067626
2019-09-04                         0.010565

            extra_regressors_multiplicative_lower  \
DATA
2019-09-02                                 0.025662
2019-09-03                                 0.067626
2019-09-04                                 0.010565

            extra_regressors_multiplicative_upper  holidays
holidays_lower  \
DATA

2019-09-02                                 0.025662       0.0
0.0
2019-09-03                                 0.067626       0.0
0.0
2019-09-04                                 0.010565       0.0
0.0

            holidays_upper  multiplicative_terms
```

```
multiplicative_terms_lower  \
DATA

2019-09-02             0.0             -0.203386                   -
0.203386
2019-09-03             0.0             -0.232652                   -
0.232652
2019-09-04             0.0             -0.327582                   -
0.327582

            multiplicative_terms_upper    weekly  weekly_lower
weekly_upper  \
DATA

2019-09-02                   -0.203386 -0.064826     -0.064826     -
0.064826
2019-09-03                   -0.232652 -0.141392     -0.141392     -
0.141392
2019-09-04                   -0.327582 -0.185022     -0.185022     -
0.185022

              yearly  yearly_lower  yearly_upper  additive_terms  \
DATA
2019-09-02 -0.164222     -0.164222     -0.164222             0.0
2019-09-03 -0.158886     -0.158886     -0.158886             0.0
2019-09-04 -0.153125     -0.153125     -0.153125             0.0

            additive_terms_lower  additive_terms_upper         yhat
DATA
2019-09-02                   0.0                   0.0  1210.256420
2019-09-03                   0.0                   0.0  1165.600052
2019-09-04                   0.0                   0.0  1021.231444
```

```
# Auswahl der relevanten Variablen aus dem Prognosedatenrahmen
prophet_variables = forecast.loc[:, ['trend',
'extra_regressors_multiplicative', 'multiplicative_terms', 'weekly',
'yearly', 'PROMO_KOD_lower']]

# Verknüpfung von Variablen mit dem Datenrahmen df_XGB_Prophet
df_XGB_Prophet = pd.concat([df_XGB_Prophet, prophet_variables],
axis=1)

# Anzeige der ersten paar Zeilen des Datenrahmens df_XGB_Prophet
df_XGB_Prophet.head()
```

```
            CENA_NP  CENA_AP  PRZECENA  CENA_MARKET PROMO_KOD
Nr_dn_tyg  \
DATA

2019-09-02     5.99     5.99       0.0     5.657993         0
```

```
1
2019-09-03     5.99     5.99       0.0     4.727133         0
2
2019-09-04     5.99     5.99       0.0     5.464078         0
3
2019-09-05     7.99     7.99       0.0     7.396029         0
4
2019-09-06     7.99     7.99       0.0     7.541655         0
5

           Nr_dn_mies Nr_mies Nr_tyg Nr_rok Dn_handlowy Hot_day
Hot_day_Xmass  \
DATA

2019-09-02          2       9     36   2019           1       0
0
2019-09-03          3       9     36   2019           1       0
0
2019-09-04          4       9     36   2019           1       0
0
2019-09-05          5       9     36   2019           1       0
0
2019-09-06          6       9     36   2019           1       0
0

           Hot_day_Wlkn        HICP            y        trend  \
DATA
2019-09-02            0  105.199997  1303.949951  1519.250927
2019-09-03            0  105.199997  1561.709961  1518.997429
2019-09-04            0  105.199997   894.440002  1518.743932
2019-09-05            0  105.199997   689.469971  1518.490434
2019-09-06            0  105.199997  1078.989990  1518.236936

            extra_regressors_multiplicative  multiplicative_terms
weekly  \
DATA

2019-09-02                         0.025662             -0.203386 -
0.064826
2019-09-03                         0.067626             -0.232652 -
0.141392
2019-09-04                         0.010565             -0.327582 -
0.185022
2019-09-05                        -0.138629             -0.041253
0.244339
2019-09-06                        -0.160581              0.021959
0.322969

              yearly  PROMO_KOD_lower
DATA
```

```
2019-09-02 -0.164222        -0.101946
2019-09-03 -0.158886        -0.101946
2019-09-04 -0.153125        -0.101946
2019-09-05 -0.146963        -0.101946
2019-09-06 -0.140429        -0.101946

# Definition von Spalten pro Datentyp
nominal_columns = ['PROMO_KOD', 'Dn_handlowy', 'Hot_day']
ordinal_columns = ['Hot_day_Xmass', 'Hot_day_Wlkn', 'Nr_dn_tyg',
'Nr_dn_mies', 'Nr_mies', 'Nr_tyg', 'Nr_rok']
numeric_columns = ['CENA_MARKET', 'CENA_AP', 'CENA_NP', 'PRZECENA',
'HICP', 'trend', 'extra_regressors_multiplicative',
'multiplicative_terms', 'weekly', 'yearly', 'PROMO_KOD_lower']

# Erstellung von Transformationen
nominal_transformer = OneHotEncoder(handle_unknown='ignore')
ordinal_transformer =
OrdinalEncoder(handle_unknown='use_encoded_value', unknown_value=-1)
numeric_transformer = MinMaxScaler()

# Erstellung von ColumnTransformer
preprocessor = ColumnTransformer(
    transformers=[
        ('nominal', nominal_transformer, nominal_columns),
        ('ordinal', ordinal_transformer, ordinal_columns),
        ('numeric', numeric_transformer, numeric_columns)
    ])

# Erstellung der Pipeline
pipeline = Pipeline(steps=[('preprocessor', preprocessor)])

# Umwandlung von Daten
X_train, y_train = df_XGB_Prophet.loc[df_XGB_Prophet.index <='2022-07-
10 00:00:00'], df_XGB_Prophet.loc[df_XGB_Prophet.index <= '2022-07-10
00:00:00']
X_test, y_test = df_XGB_Prophet.loc[df_XGB_Prophet.index>'2022-07-10
00:00:00'], df_XGB_Prophet.loc[df_XGB_Prophet.index > '2022-07-10
00:00:00']
X_train = X_train.drop(columns=['y'])
X_test = X_test.drop(columns=['y'])
y_train = y_train['y']
y_test = y_test['y']

# Parameter
n_estimators_values = [50, 100, 300, 500, 1000, 1500]
learning_rate_values = [0.01, 0.02, 0.05, 0.1]
max_depth_values = [3, 5, 6, 8, 10, 12]

# Erstellung von Parameterkombinationen
param_combinations_XGB_Prophet = list(product(n_estimators_values,
```

```
learning_rate_values, max_depth_values))
len(param_combinations_XGB_Prophet)
```

```
144
```

```
# Initialisierung eines Datenrahmens zur Speicherung der Ergebnisse
der Metriken für jeden Fold
results_df_XGB_Prophet = pd.DataFrame(columns=['Fold', 'n_estimators',
'learning_rate', 'max_depth', 'MAE', 'RMSE', 'R2', 'WMAPE', 'MASE'])

# Erstellung eines TimeSeriesSplit-Objekts
tscv = TimeSeriesSplit(n_splits=5)

# Iterationt durch Fold
for fold, (train_index, test_index) in enumerate(tscv.split(X_train)):
    X_train_XGB_Prophet = X_train.iloc[train_index]
    X_test_XGB_Prophet = X_train.iloc[test_index]
    y_train_XGB_Prophet = y_train.iloc[train_index]
    y_test_XGB_Prophet = y_train.iloc[test_index]
    # Kodoierung der Daten
    X_train_XGB_Prophet = pipeline.fit_transform(X_train_XGB_Prophet)
    X_test_XGB_Prophet = pipeline.transform(X_test_XGB_Prophet)

    best_rmse = float('inf')
    best_params = None

    # Iteration durch Parameterkombinationen
    for params in param_combinations_XGB_Prophet:
        n_estimators, learning_rate, max_depth = params
        XGB_Prophet = xgb.XGBRegressor(objective='reg:squarederror',
                                n_jobs=-1,
                                random_state=42,
                                n_estimators=n_estimators,
                                learning_rate=learning_rate,
                                max_depth=max_depth,
                                tree_method='gpu_hist')
        XGB_Prophet.fit(X_train_XGB_Prophet, y_train_XGB_Prophet)
        y_pred_XGB_Prophet = XGB_Prophet.predict(X_test_XGB_Prophet)
        rmse = mean_squared_error(y_test_XGB_Prophet,
y_pred_XGB_Prophet, squared=False)

        if rmse < best_rmse:
            best_rmse = rmse
            best_params = params

    # Das beste Modell erstellen
    n_estimators, learning_rate, max_depth = best_params
    XGB_Prophet = xgb.XGBRegressor(objective='reg:squarederror',
                            n_jobs=-1,
                            random_state=42,
```

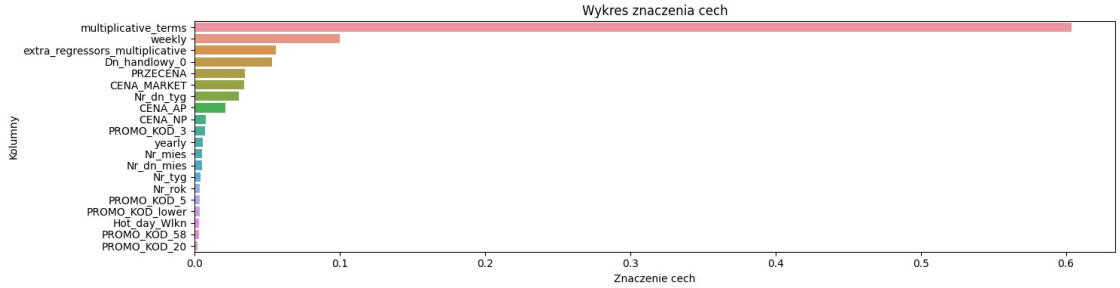
```
                            n_estimators=n_estimators,
                            learning_rate=learning_rate,
                            max_depth=max_depth,
                            tree_method='gpu_hist')
    XGB_Prophet.fit(X_train_XGB_Prophet, y_train_XGB_Prophet)
    y_pred_XGB = XGB_Prophet.predict(X_test_XGB_Prophet)

    # Berechnung von Metriken
    best_mae, best_rmse, best_r2, best_wmape, best_mase =
ff.metryki(y_test_XGB_Prophet, y_pred_XGB_Prophet)

    # Hinzufügen der besten Ergebnisse der Metrik zum Datenrahmen
    results_df_XGB_Prophet = results_df_XGB_Prophet.append({'Fold':
fold, 'n_estimators': n_estimators,

'learning_rate': learning_rate, 'max_depth': max_depth,
                                                        'MAE':
best_mae, 'RMSE': best_rmse, 'R2': best_r2,
                                                        'WMAPE':
best_wmape, 'MASE': best_mase}, ignore_index=True)

# Anzeige eines Datenrahmens mit den Ergebnissen der Metriken für
jeden Fold
results_df_XGB_Prophet.head()
```

```
   Fold  n_estimators  learning_rate  max_depth         MAE
RMSE  \
0   0.0         300.0           0.10        3.0  852.020935
1911.849243
1   1.0         100.0           0.10        6.0  240.709900
586.003174
2   2.0         500.0           0.01        3.0  490.060608
1083.381714
3   3.0          50.0           0.10        5.0  556.470154
1173.140991
4   4.0         100.0           0.10        5.0  334.201355
781.275757

         R2     WMAPE      MASE
0  0.532835  0.362159  0.686972
1  0.911843  0.424006  0.279509
2  0.894831  0.385142  0.415149
3  0.759364  1.062672  0.514384
4  0.792215  0.621996  0.464082
```

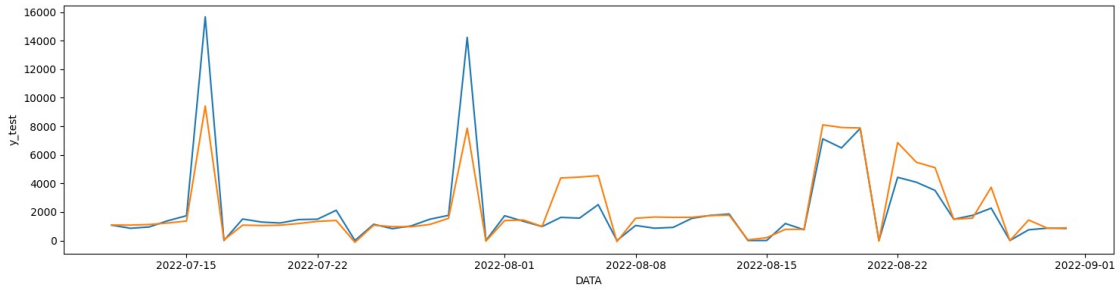
```
# Beste Parameter XGBRegressor
print('Najlepsze parametry znalezione:', XGB_Prophet.get_params())
```

```
Najlepsze parametry znalezione: {'objective': 'reg:squarederror',
'base_score': None, 'booster': None, 'callbacks': None,
'colsample_bylevel': None, 'colsample_bynode': None,
```

```
'colsample_bytree': None, 'early_stopping_rounds': None,
'enable_categorical': False, 'eval_metric': None, 'feature_types':
None, 'gamma': None, 'gpu_id': None, 'grow_policy': None,
'importance_type': None, 'interaction_constraints': None,
'learning_rate': 0.1, 'max_bin': None, 'max_cat_threshold': None,
'max_cat_to_onehot': None, 'max_delta_step': None, 'max_depth': 5,
'max_leaves': None, 'min_child_weight': None, 'missing': nan,
'monotone_constraints': None, 'n_estimators': 100, 'n_jobs': -1,
'num_parallel_tree': None, 'predictor': None, 'random_state': 42,
'reg_alpha': None, 'reg_lambda': None, 'sampling_method': None,
'scale_pos_weight': None, 'subsample': None, 'tree_method':
'gpu_hist', 'validate_parameters': None, 'verbosity': None}

# Extrahieren der Merkmalsbedeutung aus dem XGBoost-Modell.
importance_scores = XGB_Prophet.feature_importances_

# Erstellen eines Wörterbuchs zur Zuordnung von Spaltenindizes zu
Originalnamen.
column_names = []
column_transformers = preprocessor.transformers_
for transformer_name, transformer, column_indices in
column_transformers:
    if transformer_name == 'nominal':
        nominal_column_names =
transformer.get_feature_names_out(nominal_columns)
        column_names.extend(nominal_column_names)
    elif transformer_name == 'ordinal':
        ordinal_column_names = column_indices
        column_names.extend(ordinal_column_names)
    elif transformer_name == 'numeric':
        numeric_column_names = numeric_columns
        column_names.extend(numeric_column_names)

# Sortieren der Merkmale nach Bedeutung.
sorted_indices = importance_scores.argsort()[::-1]
sorted_column_names = [column_names[i] for i in sorted_indices]
sorted_importance_scores = importance_scores[sorted_indices]

# Erzeugen eines Diagramms zur Bedeutung von Merkmalen mit
benutzerdefinierten Beschriftungen
plt.figure(figsize=(15, 4))
sns.barplot(x=sorted_importance_scores[:20],
y=sorted_column_names[:20], orient='h')
plt.xlabel('Znaczenie cech')
plt.ylabel('Kolumny')
plt.title('Wykres znaczenia cech')
plt.tight_layout()
plt.show()
```




```
# Kodierung
X_train_XGB_Prophet = pipeline.fit_transform(X_train)
X_test_XGB_Prophet = pipeline.transform(X_test)
# Erstellungen von Predictions
XGB_Prophet.fit(X_train_XGB_Prophet, y_train)
y_predict_XGB_Prophet = XGB_Prophet.predict(X_test_XGB_Prophet)

# Erstellung eines DataFrame zur Visualisierung
XGB_Prophet_pred = df.loc[df.index >'2022-07-10 00:00:00'][['y']]
XGB_Prophet_pred['y_test'] = XGB_Prophet_pred['y']
XGB_Prophet_pred = XGB_Prophet_pred.drop('y', axis=1)
XGB_Prophet_pred['y_predict'] = y_predict_XGB_Prophet

# Grafik - Vergleich der Vorhersagen
plt.figure(figsize=(15, 4))
sns.lineplot(data=XGB_Prophet_pred, x=XGB_Prophet_pred.index,
y='y_test')
sns.lineplot(data=XGB_Prophet_pred, x=XGB_Prophet_pred.index,
y='y_predict')
plt.tight_layout()
plt.show()
```




```
# Metriken
mae_XGB_Prophet, rmse_XGB_Prophet, r2_XGB_Prophet, wmape_XGB_Prophet,
mase_XGB_Prophet = ff.metryki(y_test, y_predict_XGB_Prophet)
print(f'MAE: {mae_XGB_Prophet:.4f}, RMSE: {rmse_XGB_Prophet:.4f}, R2:
{r2_XGB_Prophet:.4f}, WMAPE: {wmape_XGB_Prophet:.4f}, MASE:
{mase_XGB_Prophet:.4f}')
```

```
MAE: 728.5372, RMSE: 1511.2217, R2: 0.7536, WMAPE: 0.3224, MASE:
0.3556
```

Neuronales Netzwerk - Dense, Vanilla-Modell

Das Dense Vanilla-Modell in TensorFlow ist ein grundlegendes neuronales Netzwerkmodell, das aus einer Sequenz von Dense-Schichten (Dichteschichten) in der TensorFlow-Bibliothek besteht. In diesem Modell ist jeder Neuron in einer bestimmten Schicht mit jedem Neuron in der vorherigen und nächsten Schicht verbunden, was eine vollständige Verbindung zwischen den Schichten schafft.

Die charakteristischen Merkmale des Dense Vanilla-Modells in TensorFlow sind:

1. Dense-Schichten: Das Modell besteht aus Dense-Schichten, wobei jede Schicht vollständig mit der vorherigen und nächsten Schicht verbunden ist. Jedes Neuron in einer bestimmten Schicht ist mit jedem Neuron in der vorherigen und nächsten Schicht verbunden, was eine vollständige Verbindung zwischen den Schichten ermöglicht.

2. Anzahl der Neuronen: Das Modell kann an verschiedene Probleme angepasst werden, indem die Anzahl der Neuronen in den Dense-Schichten geändert wird. Eine größere Anzahl von Neuronen kann die Kapazität des Modells erhöhen, kann jedoch auch mehr Rechenressourcen erfordern.

3. Aktivierungsfunktionen: Jede Dense-Schicht im Modell kann eine Aktivierungsfunktion haben, die Nichtlinearität in die Ergebnisse der Neuronen einführt. Beliebte Aktivierungsfunktionen sind ReLU (Rectified Linear Unit), Sigmoid oder Hyperbolic Tangent.

4. Kompilierung und Training: Vor dem Start des Trainings muss das Modell kompiliert werden, wobei die Verlustfunktion (z. B. Mean Squared Error, Cross-Entropy), der Optimierer (z. B. SGD, Adam) und Metriken festgelegt werden, die die Leistung des Modells bewerten. Anschließend wird das Modell mit den Trainingsdaten trainiert und die Gewichte in den Dense-Schichten werden aktualisiert, um die Verlustfunktion zu minimieren.

5. Bewertung und Vorhersage: Nach dem Training kann das Modell auf Testdaten anhand bestimmter Metriken wie Genauigkeit oder Mean Squared Error bewertet werden. Geschulte Modelle können auch verwendet werden, um Vorhersagen für neue Daten zu treffen.

Das Dense Vanilla-Modell in TensorFlow ist ein flexibles und einfaches Modell, das an verschiedene Aufgaben wie Klassifikation oder Regression angepasst werden kann. Seine Struktur, die auf Dense-Schichten basiert, ermöglicht es dem Modell, komplexe Abhängigkeiten in den Daten zu erlernen. Durch die praktischen Werkzeuge und Funktionen in TensorFlow ist die Implementierung des Dense Vanilla-Modells relativ einfach und effizient.

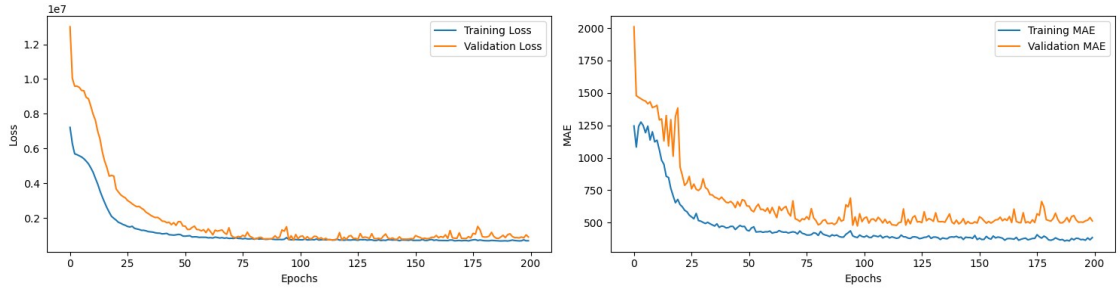
```
# Konvertierung von Daten in TensorFlow-Tensoren.
X_train_tensor = tf.convert_to_tensor(X_train_transformed)
X_test_tensor = tf.convert_to_tensor(X_test_transformed)
```

```
# Aufbau des Modells
model_vanilla = Sequential(name='vanilla')
model_vanilla.add(Dense(16, activation='relu',
input_shape=(X_train_tensor.shape[1],)))
model_vanilla.add(Dense(8, activation='relu'))
model_vanilla.add(Dense(1, activation='linear'))

# Kompilieren des Modells
optimizer = Adam(learning_rate=0.008)

model_vanilla.compile(optimizer=optimizer, loss='mse',
metrics=['mae'])

# Zusammenfassung des Modells
model_vanilla.summary()
```

```
Model: "vanilla"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 16)                432

 dense_1 (Dense)             (None, 8)                 136

 dense_2 (Dense)             (None, 1)                 9

=================================================================
Total params: 577
Trainable params: 577
Non-trainable params: 0
_________________________________________________________________
```

```
# Training des Modells
history_vanilla = model_vanilla.fit(X_train_tensor, y_train,
epochs=200, batch_size=32, validation_data=(X_test_tensor, y_test),
verbose=0)

# Abrufen von Daten aus der Historie
loss = history_vanilla.history['loss']
val_loss = history_vanilla.history['val_loss']
mae = history_vanilla.history['mae']
val_mae = history_vanilla.history['val_mae']

# Visualisierung der Verlaufsdaten der Verlustfunktion
plt.figure(figsize=(15, 4))
plt.subplot(1, 2, 1)
plt.plot(loss, label='Training Loss')
plt.plot(val_loss, label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
```

```
plt.legend()

# Visualisierung des Verlaufs der MAE-Metrik
plt.subplot(1, 2, 2)
plt.plot(mae, label='Training MAE')
plt.plot(val_mae, label='Validation MAE')
plt.xlabel('Epochs')
plt.ylabel('MAE')
plt.legend()

plt.tight_layout()
plt.show()
```



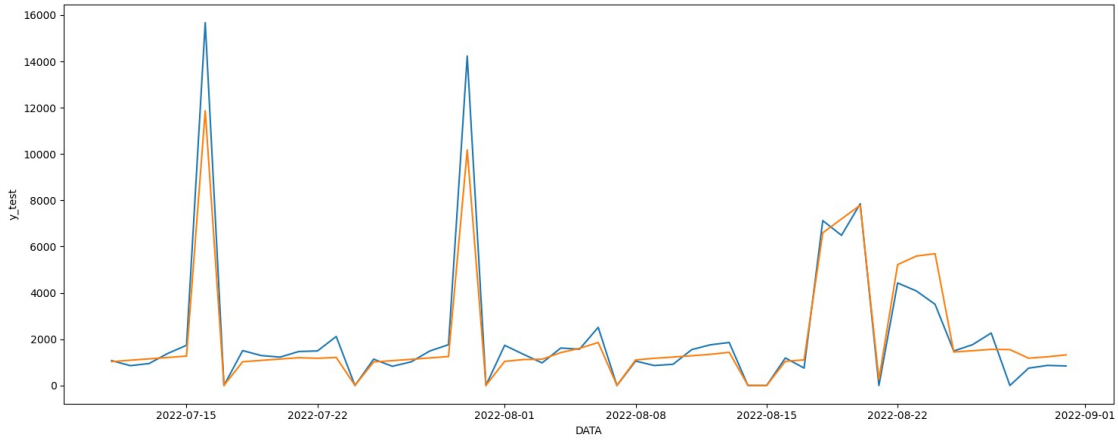

```
# Evaluation des Modells
loss, mae = model_vanilla.evaluate(X_test_tensor, y_test)
print('Strata:', loss)
print('MAE:', mae)
```

```
2/2 [==============================] - 0s 6ms/step - loss: 900542.1250
- mae: 513.3198
Strata: 900542.125
MAE: 513.3197631835938
```

```
# Erstellungen von Predictions
y_predict_dense_vanilla = model_vanilla.predict(X_test_transformed)
```

```
2/2 [==============================] - 0s 8ms/step
```

```
# Erstellung eines DataFrame zur Visualisierung
vanilla_pred = df.loc[df.index >'2022-07-10 00:00:00'][['y']]
vanilla_pred['y_test'] = vanilla_pred['y']
vanilla_pred = vanilla_pred.drop('y', axis=1)
vanilla_pred['y_predict'] = y_predict_dense_vanilla

# Grafik - Vergleich der Vorhersagen
plt.figure(figsize=(15, 6))
sns.lineplot(data=vanilla_pred, x=vanilla_pred.index, y='y_test')
sns.lineplot(data=vanilla_pred, x=vanilla_pred.index, y='y_predict')
plt.tight_layout()
plt.show()
```




```
# Metriken
mae_dense_vanilla, rmse_dense_vanilla, r2_dense_vanilla,
wmape_dense_vanilla, mase_dense_vanilla = ff.metryki(y_test,
y_predict_dense_vanilla.reshape(-1))
print(f'MAE: {mae_dense_vanilla:.4f}, RMSE: {rmse_dense_vanilla:.4f},
R2: {r2_dense_vanilla:.4f}, WMAPE: {wmape_dense_vanilla:.4f}, MASE:
{mase_dense_vanilla:.4f}')
```

```
MAE: 513.3198, RMSE: 948.9691, R2: 0.9028, WMAPE: 0.2353, MASE: 0.2506
```

Neuronales Netzwerk - Dense, erweitertes Modell

Das Dense-Modell mit einer größeren Anzahl von Neuronen und Schichten in TensorFlow ist eine erweiterte Version des Dense-Vanilla-Modells. In diesem Modell wird sowohl die Anzahl der Neuronen in den Schichten als auch die Anzahl der Schichten erhöht, um die Kapazität und die Fähigkeit des Modells zur Erkennung komplexer Muster in den Daten zu erhöhen.

Charakteristische Merkmale des Dense-Modells mit einer größeren Anzahl von Neuronen und Schichten in TensorFlow sind:

1. Anzahl der Neuronen: Das Modell verfügt über eine größere Anzahl von Neuronen in den Dense-Schichten im Vergleich zum Dense-Vanilla-Modell. Eine größere Anzahl von Neuronen erhöht die Kapazität des Modells und ermöglicht

2. Anzahl der Schichten: Das Modell enthält eine größere Anzahl von Dense-Schichten. Durch das Hinzufügen weiterer Schichten können tiefere neuronale Netzwerke aufgebaut werden, die hierarchische Merkmale in den Daten lernen können.

3. Nichtlinearität: Das Modell verwendet Aktivierungsfunktionen wie ReLU (Rectified Linear Unit), die Nichtlinearität in die Ausgabe der Neuronen einführen. Diese Nichtlinearität ist wichtig für die Fähigkeit des Modells, komplexe Abhängigkeiten zwischen den Daten zu modellieren.

4. Regularisierung: Bei größeren Modellen, insbesondere mit einer größeren Anzahl von Schichten und Neuronen, ist Regularisierung wichtig, um Overfitting zu

vermeiden. Techniken wie L1/L2-Regularisierung oder Dropout können angewendet werden, um den Einfluss von zufälligem Rauschen zu reduzieren und die Generalisierungsfähigkeit des Modells zu verbessern.

5. Parallele Berechnungen: Bei großen Modellen mit einer größeren Anzahl von Schichten und Neuronen können parallele Berechnungen wie die Nutzung von GPUs verwendet werden, um den Trainingsprozess zu beschleunigen.

Dense-Modelle mit einer größeren Anzahl von Neuronen und Schichten in TensorFlow haben eine größere Fähigkeit, komplexe Muster in den Daten zu lernen, erfordern jedoch möglicherweise mehr Rechenressourcen und Trainingszeit. Es ist wichtig, die Anzahl der Neuronen, die Anzahl der Schichten und Regularisierungstechniken sorgfältig auszuwählen, um Overfitting zu vermeiden und optimale Vorhersageergebnisse zu erzielen.

```
# Konvertierung von Daten in TensorFlow-Tensoren
X_train_tensor = tf.convert_to_tensor(X_train_transformed)
X_test_tensor = tf.convert_to_tensor(X_test_transformed)

# Aufbau des Modells
model_dense = Sequential(name='dense')
model_dense.add(Dense(64, activation='relu',
input_shape=(X_train_tensor.shape[1],)))
model_dense.add(Dropout(0.1))
model_dense.add(Dense(64, activation='relu'))
model_dense.add(Dropout(0.1))
model_dense.add(Dense(32, activation='relu'))
model_dense.add(Dropout(0.1))
model_dense.add(Dense(8, activation='relu'))
model_dense.add(Dense(1, activation='linear'))

# Kompilieren des Modells
optimizer = Adam(learning_rate=0.001)
model_dense.compile(optimizer=optimizer, loss='mse', metrics=['mae'])

# Zusammenfassung des Modells
model_dense.summary()
```

```
Model: "dense"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_203 (Dense)           (None, 64)                1728

 dropout_141 (Dropout)       (None, 64)                0

 dense_204 (Dense)           (None, 64)                4160

 dropout_142 (Dropout)       (None, 64)                0
```

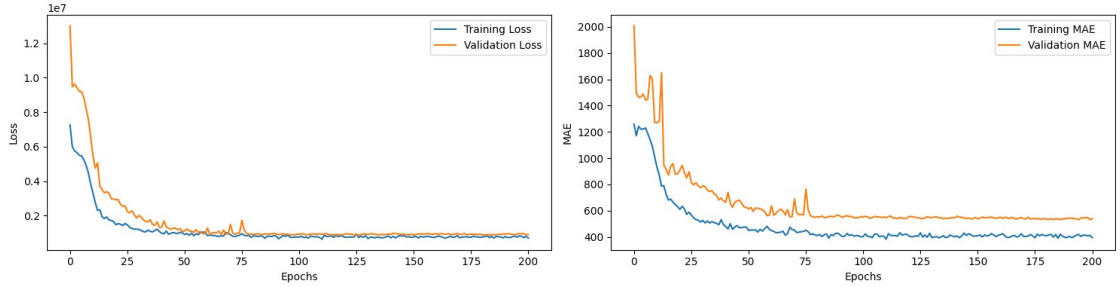
```
 dense_205 (Dense)           (None, 32)                2080

 dropout_143 (Dropout)       (None, 32)                0

 dense_206 (Dense)           (None, 8)                 264

 dense_207 (Dense)           (None, 1)                 9

=================================================================
Total params: 8,241
Trainable params: 8,241
Non-trainable params: 0
_________________________________________________________________
```

```python
# Callback's
early_stopping = EarlyStopping(monitor='val_loss', patience=50,
restore_best_weights=True)
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.1,
patience=10, min_lr=0.0001)

callbacks = [early_stopping, reduce_lr]

# Training des Modells mit callback's
history_dense = model_dense.fit(X_train_tensor, y_train, epochs=1000,
batch_size=16, validation_data=(X_test_tensor, y_test),
callbacks=callbacks, verbose=0)

# Evaluation des Modells
loss2, mae2 = model_dense.evaluate(X_test_tensor, y_test)
print('Strata:', loss2)
print('MAE:', mae2)
```

```
2/2 [==============================] - 0s 6ms/step - loss: 873150.9375
- mae: 536.9333
Strata: 873150.9375
MAE: 536.9332885742188
```

```python
# Abrufen von Daten aus der Historie
loss = history_dense.history['loss']
val_loss = history_dense.history['val_loss']
mae = history_dense.history['mae']
val_mae = history_dense.history['val_mae']

# Visualisierung der Verlaufsdaten der Verlustfunktion
plt.figure(figsize=(15, 4))
plt.subplot(1, 2, 1)
plt.plot(loss, label='Training Loss')
plt.plot(val_loss, label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
```

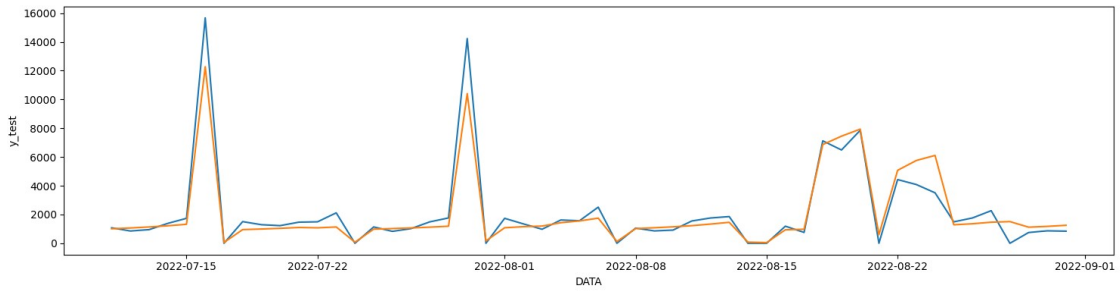
```
# Visualisierung des Verlaufs der MAE-Metrik
plt.subplot(1, 2, 2)
plt.plot(mae, label='Training MAE')
plt.plot(val_mae, label='Validation MAE')
plt.xlabel('Epochs')
plt.ylabel('MAE')
plt.legend()

plt.tight_layout()
plt.show()
```




```
# Erstellungen von Predictions
y_predict_dense = model_dense.predict(X_test_transformed)
```

```
2/2 [==============================] - 0s 4ms/step
```

```
# Erstellung eines DataFrame zur Visualisierung
dense_pred = df.loc[df.index >'2022-07-10 00:00:00'][['y']]
dense_pred['y_test'] = dense_pred['y']
dense_pred = dense_pred.drop('y', axis=1)
dense_pred['y_predict'] = y_predict_dense

# Grafik - Vergleich der Vorhersagen
plt.figure(figsize=(15, 4))
sns.lineplot(data=dense_pred, x=dense_pred.index, y='y_test')
sns.lineplot(data=dense_pred, x=dense_pred.index, y='y_predict')
plt.tight_layout()
plt.show()
```




```
# Metriken
mae_dense, rmse_dense, r2_dense, wmape_dense, mase_dense =
```

```
ff.metryki(y_test, y_predict_dense.reshape(-1))
print(f'MAE: {mae_dense:.4f}, RMSE: {rmse_dense:.4f}, R2:
{r2_dense:.4f}, WMAPE: {wmape_dense:.4f}, MASE: {mase_dense:.4f}')
```

```
MAE: 536.9332, RMSE: 934.4254, R2: 0.9058, WMAPE: 0.2388, MASE: 0.2621
```

Rekurrentes neuronales Netzwerk - LSTM

Rekurrentes neuronales Netzwerk - LSTM (Long Short-Term Memory)

Ein LSTM-Modell (Long Short-Term Memory) in TensorFlow ist ein Modell, das LSTM-Schichten verwendet, um Sequenzdaten zu analysieren und zu verarbeiten. LSTM ist eine Art von rekurrentem neuronalen Netzwerk, das in der Lage ist, langfristige Abhängigkeiten in sequenziellen Daten zu speichern.

Charakteristische Merkmale eines LSTM-Modells in TensorFlow sind:

1. LSTM-Schichten: Das Modell besteht aus LSTM-Schichten, die für die Verarbeitung von Sequenzdaten und das Speichern von Informationen im Langzeitgedächtnis verantwortlich sind. LSTM-Schichten verfügen über einen integrierten Gattermechanismus, der den Informationsfluss in dem Netzwerk kontrolliert und es ermöglicht, relevante Informationen abhängig vom Kontext zu lernen und zu speichern.

2. Langzeit- und Kurzzeitgedächtnis: LSTM-Schichten haben die Fähigkeit, Informationen im Langzeitgedächtnis zu speichern, was das Lernen von Abhängigkeiten über längere Distanzen in Sequenzen ermöglicht. Zusätzlich verfügen sie über ein Kurzzeitgedächtnis, das den aktuellen Zustand des Modells im Kontext der Sequenz speichert.

3. Rückwärtige Gradientenberechnung: Das LSTM-Modell verwendet die Methode der rückwärtigen Gradientenberechnung (backpropagation through time), um sequenzielle Abhängigkeiten zu lernen. Dabei werden Fehler rückwärts durch die gesamte Sequenz propagiert und die Gewichte werden entsprechend aktualisiert, um die Verlustfunktion zu minimieren.

4. Verwendung von Aktivierungsfunktionen: LSTM-Schichten verwenden verschiedene Aktivierungsfunktionen wie die Tangenshyperbolicus-Funktion (tanh) und die Sigmoid-Funktion, um den Informationsfluss zu regulieren und den Zustand des Gedächtnisses im Inneren des Netzwerks zu steuern.

5. Anwendung von Regularisierungstechniken: Ähnlich wie bei anderen Modellen können Regularisierungstechniken wie Dropout oder L1/L2-Regularisierung angewendet werden, um Overfitting des Modells zu vermeiden.

Rekurrente LSTM-Netzwerke in TensorFlow sind besonders effektiv bei der Analyse von Sequenzdaten wie Texten, Tonaufnahmen oder Zeitreihen. Sie sind in der Lage, langfristige Abhängigkeiten zu modellieren und werden häufig in Bereichen wie der natürlichen Sprachverarbeitung, der Spracherkennung und der Zeitreihenprognose eingesetzt.

```
# Funktion zur Datentransformation
def reshape_data(data, time_steps):
    samples, features = data.shape
    # Erstellung eines leeren DataFrame mit einer geeigneten Form
    reshaped_data = np.zeros((samples - time_steps + 1, time_steps,
features))

    # Umwandlung von Daten
    for i in range(samples - time_steps + 1):
        reshaped_data[i] = data[i:i + time_steps]

    return reshaped_data

#Festlegen der Anzahl der Zeitschritte
time_steps = 14

# Hinzufügen der letzten 'time_steps' Zeilen des Trainingsdatensatzes
zum Testdatensatz
# Hinweis: Laut dem Autor darf diese Transformation nicht als
Datenleck betrachtet werden
X_test_transformed_temp = np.concatenate((X_train_transformed[-
time_steps + 1:], X_test_transformed), axis=0)

# Umwandlung von Daten
X_train_3d = reshape_data(X_train_transformed, time_steps)
X_test_3d = reshape_data(X_test_transformed_temp , time_steps)

# Kopieren von Etiketten
y_train_3d = y_train[time_steps - 1:].values
y_test_3d = y_test

# Shape der Daten
print(X_train_3d.shape)
print(y_train_3d.shape)
print(X_test_3d.shape)
print(y_test_3d.shape)
```

```
(1030, 14, 26)
(1030,)
(52, 14, 26)
(52,)
```

```
# Modelparameter
input_shape = X_train_3d.shape[1:]
output_size = 1
input_shape
```

```
(14, 26)
```

```python
# Aufbau des Modells
model_lstm = Sequential(name='lstm')
model_lstm.add(LSTM(256, input_shape=input_shape,
return_sequences=True))
model_lstm.add(Dropout(0.1))

model_lstm.add(LSTM(128, return_sequences=False))
model_lstm.add(Dropout(0.1))

model_lstm.add(Dense(128, activation='relu'))
model_lstm.add(Dropout(0.1))

model_lstm.add(Dense(64, activation='relu'))
model_lstm.add(Dropout(0.1))

model_lstm.add(Dense(16, activation='relu'))
model_lstm.add(Dense(1))

# Kompilierung des Modells
optimizer = Adam(learning_rate=0.001)
model_lstm.compile(optimizer=optimizer, loss='mse', metrics=['mae'])

# Callback's
early_stopping = EarlyStopping(monitor='val_loss', patience=20,
restore_best_weights=True)
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.1,
patience=5, min_lr=0.0001)
callbacks = [early_stopping, reduce_lr]

# Training des Modells
history_lstm = model_lstm.fit(X_train_3d, y_train_3d, epochs=1000,
batch_size=8, validation_data=(X_test_3d, y_test_3d),
callbacks=callbacks, verbose=0)

# Charakteristika des Modellaufbaus
model_lstm.summary()
```

```
Model: "lstm"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_102 (LSTM)             (None, 14, 256)           289792

 dropout_167 (Dropout)       (None, 14, 256)           0

 lstm_103 (LSTM)             (None, 128)               197120

 dropout_168 (Dropout)       (None, 128)               0

 dense_231 (Dense)           (None, 128)               16512
```

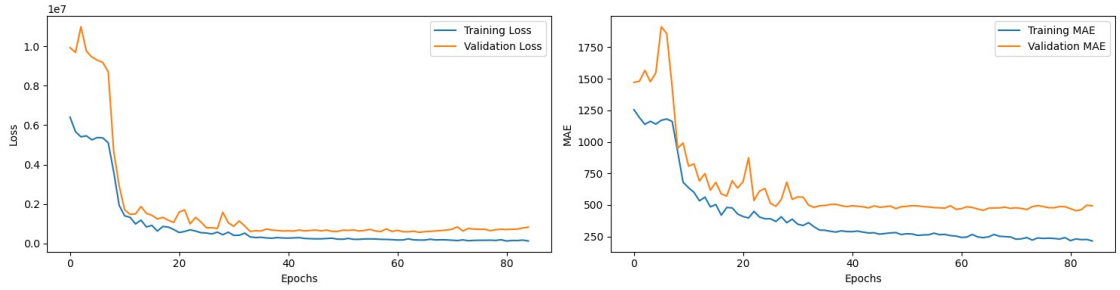
```
 dropout_169 (Dropout)       (None, 128)               0

 dense_232 (Dense)           (None, 64)                8256

 dropout_170 (Dropout)       (None, 64)                0

 dense_233 (Dense)           (None, 16)                1040

 dense_234 (Dense)           (None, 1)                 17

=================================================================
Total params: 512,737
Trainable params: 512,737
Non-trainable params: 0
_________________________________________________________________
```

```python
# Evaluation des Modells
loss_lstm, mae_lstm = model_lstm.evaluate(X_test_3d, y_test_3d)
print('Strata:', loss_lstm)
print('MAE:', mae_lstm)
```

```
2/2 [==============================] - 0s 9ms/step - loss: 593645.7500
- mae: 513.8962
Strata: 593645.75
MAE: 513.896240234375
```

```python
# Abrufen von Daten aus der Historie
loss = history_lstm.history['loss']
val_loss = history_lstm.history['val_loss']
mae = history_lstm.history['mae']
val_mae = history_lstm.history['val_mae']

# Visualisierung der Verlaufsdaten der Verlustfunktion
plt.figure(figsize=(15, 4))
plt.subplot(1, 2, 1)
plt.plot(loss, label='Training Loss')
plt.plot(val_loss, label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

# Visualisierung des Verlaufs der MAE-Metrik
plt.subplot(1, 2, 2)
plt.plot(mae, label='Training MAE')
plt.plot(val_mae, label='Validation MAE')
plt.xlabel('Epochs')
plt.ylabel('MAE')
plt.legend()
```

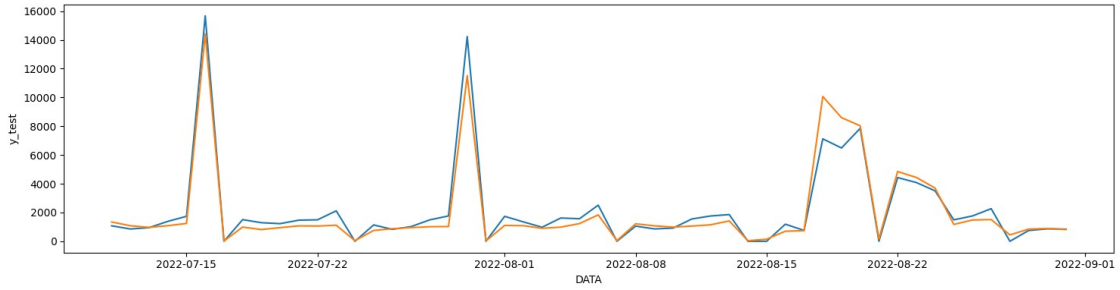
```
plt.tight_layout()
plt.show()
```




```
# Erstellungen von Predictions
y_predict_lstm = model_lstm.predict(X_test_3d)
```

```
2/2 [==============================] - 1s 5ms/step
```

```
# Erstellung eines DataFrame zur Visualisierung
lstm_pred = df.loc[df.index >'2022-07-10 00:00:00'][['y']]
lstm_pred['y_test'] = lstm_pred['y']
lstm_pred = lstm_pred.drop('y', axis=1)
lstm_pred['y_predict'] = y_predict_lstm.reshape(-1)

# Grafik - Vergleich der Vorhersagen
plt.figure(figsize=(15, 4))
sns.lineplot(data=lstm_pred, x=lstm_pred.index, y='y_test')
sns.lineplot(data=lstm_pred, x=lstm_pred.index, y='y_predict')
plt.tight_layout()
plt.show()
```




```
# Metriken
mae_lstm, rmse_lstm, r2_lstm, wmape_lstm, mase_lstm =
ff.metryki(lstm_pred['y_test'], lstm_pred['y_predict'])
print(f'MAE: {mae_lstm:.4f}, RMSE: {rmse_lstm:.4f}, R2: {r2_lstm:.4f},
WMAPE: {wmape_lstm:.4f}, MASE: {mase_lstm:.4f}')
```

```
MAE: 478.8448, RMSE: 840.8692, R2: 0.9237, WMAPE: 0.1876, MASE: 0.2337
```

Die Modell- und Testkonstruktion für mehrdimensionale Zeitreihen

Import und Vorverarbeitung der Daten

```
# Laden von Daten
df = pd.read_pickle('PICKLE/df_3.pkl')
df = df.loc[df['CLUSTER']==2].drop(['CLUSTER'], axis=1)

df.set_index('DATA', inplace=True)
NAZWA = df['NAZWA']
df.drop('NAZWA', inplace=True, axis=1)

df.head()
```

```
            GRUPA_ID ART_ID  CENA_NP  CENA_AP  PRZECENA  CENA_MARKET
PROMO_KOD  \
DATA

2019-09-02       19      1     3.49     3.49       0.0     2.990000
0
2019-09-02        8      2     5.99     5.99       0.0     5.498056
0
2019-09-02        8      3     4.99     4.99       0.0     4.543005
0
2019-09-02        8      4     4.99     4.99       0.0     4.241022
0
2019-09-02        8      5     5.99     5.99       0.0     5.308895
0

            Nr_dn_tyg Nr_dn_mies Nr_mies Nr_tyg Nr_rok Dn_handlowy
Hot_day  \
DATA

2019-09-02         1          2       9     36   2019           1
0
2019-09-02         1          2       9     36   2019           1
0
2019-09-02         1          2       9     36   2019           1
0
2019-09-02         1          2       9     36   2019           1
0
2019-09-02         1          2       9     36   2019           1
0

            Hot_day_Xmass Hot_day_Wlkn        HICP           y
DATA
2019-09-02              0            0  105.199997   23.000000
2019-09-02              0            0  105.199997   69.449997
2019-09-02              0            0  105.199997   50.110001
2019-09-02              0            0  105.199997  196.869995
2019-09-02              0            0  105.199997   90.839996
```

Datenkodierung

```
# Definition von Spalten pro Datentyp
nominal_columns = ['GRUPA_ID', 'ART_ID', 'PROMO_KOD', 'Dn_handlowy',
'Hot_day']
ordinal_columns = ['Hot_day_Xmass', 'Hot_day_Wlkn', 'Nr_dn_tyg',
'Nr_dn_mies', 'Nr_mies', 'Nr_tyg', 'Nr_rok']
numeric_columns = ['CENA_MARKET', 'CENA_AP', 'CENA_NP', 'PRZECENA',
'HICP']

# Erstellung von przekształceń
nominal_transformer = OneHotEncoder(handle_unknown='ignore')
ordinal_transformer =
OrdinalEncoder(handle_unknown='use_encoded_value', unknown_value=-1)
numeric_transformer = MinMaxScaler()

# Erstellung von ColumnTransformer
preprocessor = ColumnTransformer(
    transformers=[
        ('nominal', nominal_transformer, nominal_columns),
        ('ordinal', ordinal_transformer, ordinal_columns),
        ('numeric', numeric_transformer, numeric_columns)
    ])

# Erstellung der Pipeline
pipeline = Pipeline(steps=[('preprocessor', preprocessor)])

# Umwandlung von Daten
X_train, y_train = df.loc[df.index <='2022-07-10 00:00:00'],
df.loc[df.index <= '2022-07-10 00:00:00']
X_test, y_test = df.loc[df.index>'2022-07-10 00:00:00'],
df.loc[df.index > '2022-07-10 00:00:00']
X_train = X_train.drop(columns=['y'])
X_test = X_test.drop(columns=['y'])
y_train = y_train['y']
y_test = y_test['y']

# Anwendung der Pipeline
X_train_transformed = pipeline.fit_transform(X_train)
X_test_transformed = pipeline.transform(X_test)
```

XGBoost Gradient Boosting

```
# Parameter
n_estimators_values = [50, 100, 300, 500, 1000, 1500]
learning_rate_values = [0.01, 0.02, 0.05]
max_depth_values = [2, 4, 6, 8, 10, 12]

# Erstellung von Parameterkombinationen
param_combinations_XGB_2 = list(product(n_estimators_values,
learning_rate_values, max_depth_values))
len(param_combinations_XGB_2)
```

```
# Initialisierung eines Datenrahmens zur Speicherung der Ergebnisse
der Metriken für jeden Fold
results_df_XGB_2 = pd.DataFrame(columns=['Fold', 'n_estimators',
'learning_rate', 'max_depth', 'MAE', 'RMSE', 'R2', 'WMAPE', 'MASE'])

# Erstellung eines TimeSeriesSplit-Objekts
tscv = TimeSeriesSplit(n_splits=3)

# Iteration durch Fold
for fold, (train_index, test_index) in enumerate(tscv.split(X_train)):
    X_train_XGB_2 = X_train.iloc[train_index]
    X_test_XGB_2 = X_train.iloc[test_index]
    y_train_XGB_2 = y_train.iloc[train_index]
    y_test_XGB_2 = y_train.iloc[test_index]
    # Kodierung
    X_train_XGB_2 = pipeline.fit_transform(X_train_XGB_2)
    X_test_XGB_2 = pipeline.transform(X_test_XGB_2)

    best_rmse = float('inf')
    best_params = None

    # Iteration durch Parameterkombinationen
    eval_set = [(X_test_XGB_2, y_test_XGB_2)]
    for params in param_combinations_XGB_2:
        n_estimators, learning_rate, max_depth = params
        XGB_2 = xgb.XGBRegressor(objective='reg:squarederror',
                                 n_jobs=-1,
                                 random_state=42,
                                 n_estimators=n_estimators,
                                 learning_rate=learning_rate,
                                 max_depth=max_depth,
                                 tree_method='gpu_hist',
                                 early_stopping_rounds=20
                                 )
        XGB_2.fit(X_train_XGB_2, y_train_XGB_2, eval_set=eval_set)
        y_pred_XGB_2 = XGB_2.predict(X_test_XGB_2)
        rmse = mean_squared_error(y_test_XGB_2, y_pred_XGB_2,
squared=False)

        if rmse < best_rmse:
            best_rmse = rmse
            best_params = params

    # Das beste Modell erstellen
    n_estimators, learning_rate, max_depth = best_params
    XGB_2 = xgb.XGBRegressor(objective='reg:squarederror',
                             n_jobs=-1,
                             random_state=42,
```

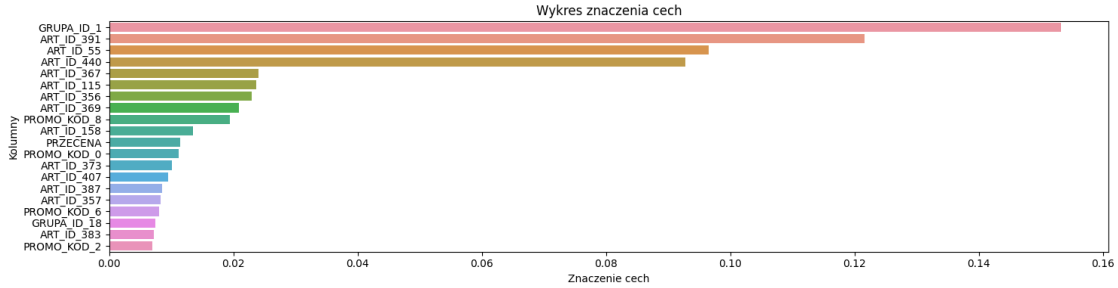
```
                            n_estimators=n_estimators,
                            learning_rate=learning_rate,
                            max_depth=max_depth,
                            tree_method='gpu_hist',
                            early_stopping_rounds=20
                            )
    eval_set = [(X_test_XGB_2, y_test_XGB_2)]
    XGB_2.fit(X_train_XGB_2, y_train_XGB_2, eval_set=eval_set)
    y_pred_XGB_2 = XGB_2.predict(X_test_XGB_2)

    # Berechnung von Metriken
    best_mae, best_rmse, best_r2, best_wmape, best_mase =
ff.metryki(y_test_XGB_2, y_pred_XGB_2)

    # Hinzufügen der besten Ergebnisse der Metrik zum Datenrahmen
    results_df_XGB_2 = results_df_XGB_2.append({'Fold': fold,
'n_estimators': n_estimators,

'learning_rate': learning_rate, 'max_depth': max_depth,
                                                    'MAE':
best_mae, 'RMSE': best_rmse, 'R2': best_r2,
                                                    'WMAPE':
best_wmape, 'MASE': best_mase}, ignore_index=True)

# Anzeige eines Datenrahmens mit den Ergebnissen der Metriken für
jeden Fold
results_df_XGB_2
```

```
   Fold  n_estimators  learning_rate  max_depth         MAE
RMSE  \
0   0.0         300.0           0.05        8.0  114.781975
514.749512
1   1.0        1000.0           0.02        8.0   97.824745
389.813049
2   2.0        1500.0           0.05        8.0   72.553925
280.054352

         R2      WMAPE      MASE
0  0.634074   7.813302  0.526647
1  0.718558  10.295111  0.465952
2  0.826443   4.784000  0.369052
```

```
# Beste Parameter fuer XGBRegressor
print('Najlepsze parametry znalezione:', XGB_2.get_params())
```

```
Najlepsze parametry znalezione: {'objective': 'reg:squarederror',
'base_score': None, 'booster': None, 'callbacks': None,
'colsample_bylevel': None, 'colsample_bynode': None,
'colsample_bytree': None, 'early_stopping_rounds': 20,
'enable_categorical': False, 'eval_metric': None, 'feature_types':
None, 'gamma': None, 'gpu_id': None, 'grow_policy': None,
```

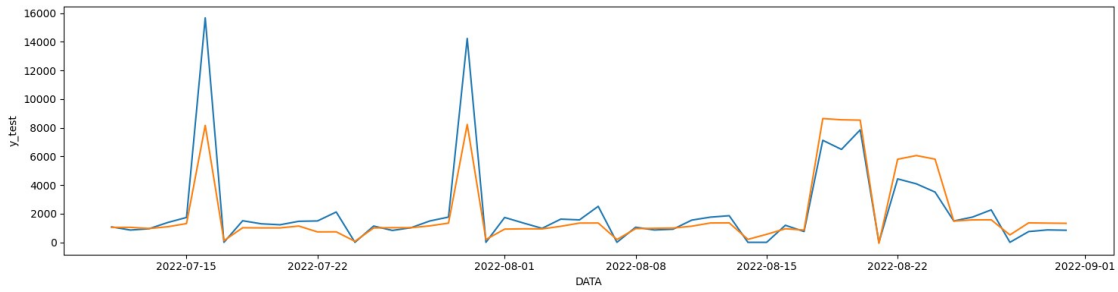
```
'importance_type': None, 'interaction_constraints': None,
'learning_rate': 0.05, 'max_bin': None, 'max_cat_threshold': None,
'max_cat_to_onehot': None, 'max_delta_step': None, 'max_depth': 8,
'max_leaves': None, 'min_child_weight': None, 'missing': nan,
'monotone_constraints': None, 'n_estimators': 1500, 'n_jobs': -1,
'num_parallel_tree': None, 'predictor': None, 'random_state': 42,
'reg_alpha': None, 'reg_lambda': None, 'sampling_method': None,
'scale_pos_weight': None, 'subsample': None, 'tree_method':
'gpu_hist', 'validate_parameters': None, 'verbosity': None}

#  Abrufen der Feature-Importanz aus dem XGBoost-Modell
importance_scores = XGB_2.feature_importances_

# Erstellen eines Wörterbuchs, das die Indexe der Spalten auf die
ursprünglichen Namen abbildet
column_names = []
column_transformers = preprocessor.transformers_
for transformer_name, transformer, column_indices in
column_transformers:
    if transformer_name == 'nominal':
        nominal_column_names =
transformer.get_feature_names_out(nominal_columns)
        column_names.extend(nominal_column_names)
    elif transformer_name == 'ordinal':
        ordinal_column_names = column_indices
        column_names.extend(ordinal_column_names)
    elif transformer_name == 'numeric':
        numeric_column_names = numeric_columns
        column_names.extend(numeric_column_names)

# Sortierung von Merkmalen nach Bedeutung
sorted_indices = importance_scores.argsort()[::-1]
sorted_column_names = [column_names[i] for i in sorted_indices]
sorted_importance_scores = importance_scores[sorted_indices]

# Generierung eines Merkmalsbedeutungsdiagramms mit angepassten
Beschriftungen
plt.figure(figsize=(15, 4))
sns.barplot(x=sorted_importance_scores[:20],
y=sorted_column_names[:20], orient='h')
plt.xlabel('Znaczenie cech')
plt.ylabel('Kolumny')
plt.title('Wykres znaczenia cech')
plt.tight_layout()
plt.show()
```




```
# Kodierung
X_test_XGB_2_one_art = X_test[X_test['ART_ID'] == 158]
X_test_XGB_one_art = pipeline.transform(X_test_XGB_2_one_art)
X_test_XGB_one_art.shape
```

```
(52, 817)
```

```
# Anpassen der Form der Testmatrix mit einer Zeile auf die Dimensionen
der Trainingsmatrix
target_shape = (52, 817)

if X_test_XGB_one_art.shape != target_shape:
    # Erstellen einer Nullmatrix mit einer neuen Dimension
    X_test_XGB_prepared = sp.csr_matrix((target_shape[0],
target_shape[1]), dtype=np.float64)

    # Kopiere die Werte von X_test_XGB_one_art in eine neue Matrix

X_test_XGB_prepared[:X_test_XGB_one_art.shape[0], :X_test_XGB_one_art.
shape[1]] = X_test_XGB_one_art
else:
    X_test_XGB_prepared = X_test_XGB_one_art


# Erstellungen von Predictions
y_predict_xgb_2 = XGB_2.predict(X_test_XGB_prepared)

# Erstellung eines DataFrame zur Visualisierung
df_xgb = df.loc[df['ART_ID'] == 158]
xgb_pred_2 = df_xgb.loc[df_xgb.index >'2022-07-10 00:00:00'][['y']]
xgb_pred_2['y_test'] = xgb_pred_2['y']
xgb_pred_2 = xgb_pred_2.drop('y', axis=1)
xgb_pred_2['y_predict'] = y_predict_xgb_2

# Grafik - Vergleich der Vorhersagen
plt.figure(figsize=(15, 4))
sns.lineplot(data=xgb_pred_2, x=xgb_pred_2.index, y='y_test')
sns.lineplot(data=xgb_pred_2, x=xgb_pred_2.index, y='y_predict')
plt.tight_layout()
plt.show()
```




```
# Metriken
mae_XGB_2, rmse_XGB_2, r2_XGB_2, wmape_XGB_2, mase_XGB_2 =
ff.metryki(xgb_pred_2['y_test'], xgb_pred_2['y_predict'])
print(f'MAE: {mae_XGB_2:.4f}, RMSE: {rmse_XGB_2:.4f}, R2:
{r2_XGB_2:.4f}, WMAPE: {wmape_XGB_2:.4f}, MASE: {mase_XGB_2:.4f}')
```

```
MAE: 739.3627, RMSE: 1515.0146, R2: 0.7523, WMAPE: 0.2758, MASE:
0.3609
```

Neuronales Netz - DENSE, ausgebautes Netz

```
# Konvertierung von Sparse-Daten in einen TensorFlow-Tensor
X_train_dense = X_train_transformed.toarray()
X_test_dense = X_test_transformed.toarray()

X_train_tensor = tf.convert_to_tensor(X_train_dense, dtype=tf.float32)
X_test_tensor = tf.convert_to_tensor(X_test_dense, dtype=tf.float32)

# Definieren der Eingabe
input_layer = Input(shape=(X_train_tensor.shape[1],))

# Dense layers
dense_layer_1 = Dense(512, activation='relu')(input_layer)
dropout_layer_1 = Dropout(0.1)(dense_layer_1)

dense_layer_2 = Dense(256, activation='relu')(dropout_layer_1)
dropout_layer_2 = Dropout(0.1)(dense_layer_2)

dense_layer_3 = Dense(128, activation='relu')(dropout_layer_2)
dropout_layer_3 = Dropout(0.1)(dense_layer_3)

dense_layer_4 = Dense(64, activation='relu')(dropout_layer_3)
dropout_layer_4 = Dropout(0.1)(dense_layer_4)

dense_layer_5 = Dense(32, activation='relu')(dropout_layer_4)
dropout_layer_5 = Dropout(0.1)(dense_layer_5)

output_layer = Dense(1, activation='linear')(dense_layer_5)

# Modell aufbau
model_dense_2 = Model(inputs=input_layer, outputs=output_layer,
name='dense_2')
```

```
# Kompilieren des Modells
optimizer = Adam(learning_rate=0.005)
model_dense_2.compile(optimizer=optimizer, loss='mse',
metrics=['mae'])

# Zusammenfassung des Modells
model_dense_2.summary()
```

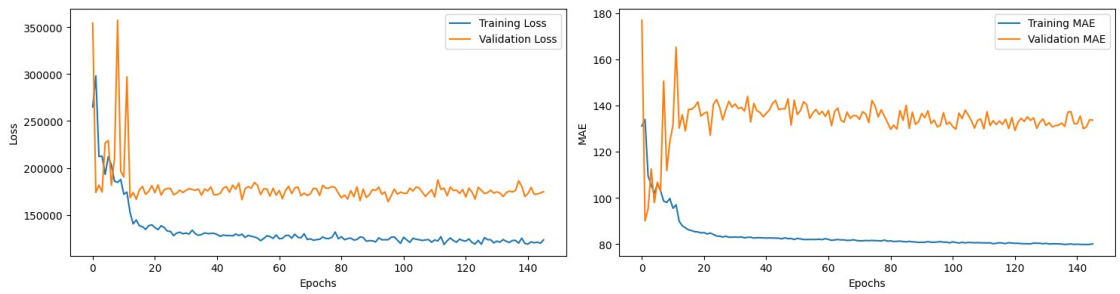
```
Model: "dense_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 877)]             0

 dense (Dense)               (None, 512)               449536

 dropout (Dropout)           (None, 512)               0

 dense_1 (Dense)             (None, 256)               131328

 dropout_1 (Dropout)         (None, 256)               0

 dense_2 (Dense)             (None, 128)               32896

 dropout_2 (Dropout)         (None, 128)               0

 dense_3 (Dense)             (None, 64)                8256

 dropout_3 (Dropout)         (None, 64)                0

 dense_4 (Dense)             (None, 32)                2080

 dense_5 (Dense)             (None, 1)                 33

=================================================================
Total params: 624,129
Trainable params: 624,129
Non-trainable params: 0
_________________________________________________________________
```

```
# Callback's
early_stopping = EarlyStopping(monitor='val_loss', patience=50,
restore_best_weights=True)
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.1,
patience=10, min_lr=0.0001)

callbacks = [early_stopping, reduce_lr]
```

```
# Training des Modells
history_dense_2 = model_dense_2.fit(X_train_tensor, y_train,
epochs=500, batch_size=256, validation_data=(X_test_tensor, y_test),
callbacks=callbacks, verbose=1)

# Evaluation des Modells
loss_dense_2, mae_dense_2 = model_dense_2.evaluate(X_test_tensor,
y_test)
print('Strata:', loss_dense_2)
print('MAE:', mae_dense_2)
```

```
1131/1131 [==============================] - 2s 2ms/step - loss:
164012.6875 - mae: 130.8593
Strata: 164012.6875
MAE: 130.85934448242188
```

```
# Abrufen von Daten aus der Historie
loss = history_dense_2.history['loss']
val_loss = history_dense_2.history['val_loss']
mae = history_dense_2.history['mae']
val_mae = history_dense_2.history['val_mae']

# Visualisierung der Verlaufsdaten der Verlustfunktion
plt.figure(figsize=(15, 4))
plt.subplot(1, 2, 1)
plt.plot(loss, label='Training Loss')
plt.plot(val_loss, label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

# Visualisierung des Verlaufs der MAE-Metrik
plt.subplot(1, 2, 2)
plt.plot(mae, label='Training MAE')
plt.plot(val_mae, label='Validation MAE')
plt.xlabel('Epochs')
plt.ylabel('MAE')
plt.legend()

plt.tight_layout()
plt.show()
```



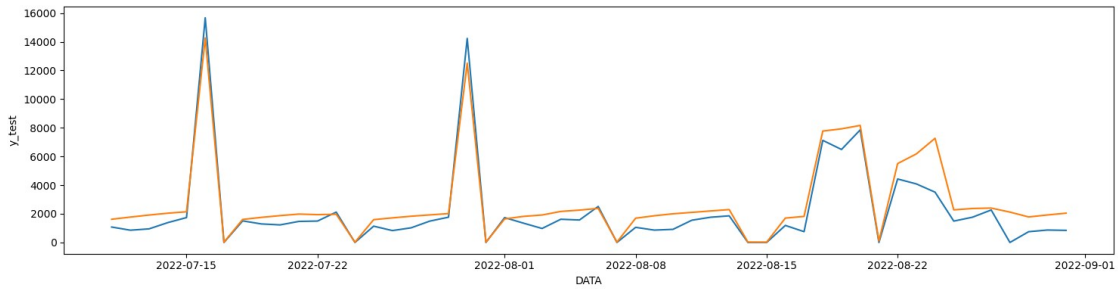
```
# Kodierung
X_train_dense_2 = pipeline.fit_transform(X_train)
X_test_dense_2  = X_test.loc[X_test['ART_ID'] == 158]
X_test_dense_2 = pipeline.transform(X_test_dense_2)
# Erstellungen von Predictions
y_predict_dense_2 = model_dense_2.predict(X_test_dense_2)
```

```
2/2 [==============================] - 0s 5ms/step
```

```
# Erstellung eines DataFrame zur Visualisierung
df_dense_2 = df.loc[df['ART_ID'] == 158]
dense_pred_2 = df_dense_2.loc[df_dense_2.index >'2022-07-10 00:00:00']
[['y']]
dense_pred_2['y_test'] = dense_pred_2['y']
dense_pred_2 = dense_pred_2.drop('y', axis=1)
dense_pred_2['y_predict'] = y_predict_dense_2
dense_pred_2.shape
```

```
(52, 2)
```

```
# Grafik - Vergleich der Vorhersagen
plt.figure(figsize=(15, 4))
sns.lineplot(data=dense_pred_2, x=dense_pred_2.index, y='y_test')
sns.lineplot(data=dense_pred_2, x=dense_pred_2.index, y='y_predict')
plt.tight_layout()
plt.show()
```




```
# Metriken
mae_dense_2, rmse_dense_2, r2_dense_2, wmape_dense_2, mase_dense_2 =
ff.metryki(dense_pred_2['y_test'], dense_pred_2['y_predict'])
print(f'MAE: {mae_dense_2:.4f}, RMSE: {rmse_dense_2:.4f}, R2:
{r2_dense_2:.4f}, WMAPE: {wmape_dense_2:.4f}, MASE:
{mase_dense_2:.4f}')
```

```
MAE: 708.4313, RMSE: 966.2647, R2: 0.8993, WMAPE: 0.5184, MASE: 0.3458
```

Neuronales Netz - DENSE + EMBEDDING

Eine zusätzliche Eingangsschicht vom Typ "Embedding", die Produktbeschreibungen akzeptiert, kann die Leistung eines Dense-Netzwerks bei der Textanalyse erheblich verbessern. Die Embedding-Schicht dient dazu, Wörter oder Phrasen in numerische Darstellungen mit niedrigerer Dimension umzuwandeln.

Die grundlegenden Eigenschaften einer zusätzlichen Eingangsschicht für Embeddings in einem Dense-Netzwerk sind:

1. Text in Zahlen umwandeln: Die Embedding-Schicht konvertiert Wörter oder Phrasen in numerische Vektoren, die von einem neuronalen Netzwerk verarbeitet werden können. Jedes Wort wird auf einen entsprechenden Vektor abgebildet, der seine Semantik oder Bedeutung im Kontext repräsentiert.

2. Berücksichtigung semantischer Beziehungen: Die Embedding-Schicht berücksichtigt die Ähnlichkeit zwischen Wörtern, was es dem neuronalen Netzwerk ermöglicht, ähnliche Wörter zu unterscheiden oder semantische Beziehungen zwischen ihnen zu erkennen. Diese Darstellung im Vektorraum ermöglicht es dem Modell, semantische Abhängigkeiten im Text effizienter zu erlernen.

3. Dimensionalitätsreduktion: Die Verwendung von Embeddings ermöglicht eine erhebliche Reduzierung der Dimensionalität von Textdaten. Dies bedeutet, dass das Dense-Netzwerk weniger Eingänge hat, was die Berechnungen beschleunigen und das Risiko von Overfitting verringern kann.

4. Generalisierungseigenschaften: Durch die numerische Repräsentation ermöglicht die Embedding-Schicht dem Dense-Netzwerk das Verallgemeinern von Mustern auf der Grundlage der Wortsemantik. Das bedeutet, dass das Netzwerk in der Lage sein wird, Informationen über Produktbeschreibungen vorherzusagen und zu generalisieren, die nicht im Trainingsdatensatz vorhanden waren.

Eine zusätzliche Embedding-Eingangsschicht ermöglicht es dem Dense-Netzwerk, Textbeschreibungen von Produkten besser zu verstehen und zu verarbeiten. Sie ermöglicht die Konvertierung von Wörtern in Zahlen, was es dem Modell ermöglicht, die semantische Bedeutung des Textes zu analysieren. Dadurch wird die Fähigkeit des Modells verbessert, Muster zu erkennen, was wiederum die Effizienz und Präzision der Vorhersagen erhöht.

```
# Konvertierung von dünnen Daten in einen TensorFlow-Tensor
X_train_dense = X_train_transformed.toarray()
X_test_dense = X_test_transformed.toarray()

X_train_tensor = tf.convert_to_tensor(X_train_dense, dtype=tf.float32)
X_test_tensor = tf.convert_to_tensor(X_test_dense, dtype=tf.float32)

# Vorbereitung der Daten z kolumny tekstowej
nazwa = NAZWA.tolist()
X_train_nazwa, X_test_nazwa = NAZWA.loc[NAZWA.index <= '2022-07-10
00:00:00'].tolist(), NAZWA.loc[NAZWA.index>'2022-07-10
00:00:00'].tolist()

# Erstellung eines Tokenizer-Objekts
num_words=1000
tokenizer = Tokenizer(num_words=num_words)

# Anpassen des Tokenizers an die Textdaten
tokenizer.fit_on_texts(nazwa)
```

```
# Textvektorisierung
sequences_train = tokenizer.texts_to_sequences(X_trening_nazwa)
sequences_test = tokenizer.texts_to_sequences(X_test_nazwa)

# Padding von Sequenzen auf feste Breite
maxlen = 10
padded_sequences_train = pad_sequences(sequences_train, maxlen=maxlen)
padded_sequences_test = pad_sequences(sequences_test, maxlen=maxlen)

#Definieren des Eingangs für tabellarische Daten
input_1 = Input(shape=(X_train_tensor.shape[1],), name='tabela')

# Eingabeebene für Textdaten
input_2 = Input(shape=(maxlen,), name='tekst')

# Einbettungsebene für Textdaten
embedding = Embedding(input_dim=num_words, output_dim=32,
input_length=maxlen)(input_2)

# Flatten Layer
flatten = Flatten()(embedding)

# Kombination von Eingabeschichten
concatenated = Concatenate()([input_1, flatten])

# Dense und Dropout layers
dense_layer_1 = Dense(512, activation='relu')(concatenated)
dropout_layer_1 = Dropout(0.1)(dense_layer_1)

dense_layer_2 = Dense(256, activation='relu')(dropout_layer_1)
dropout_layer_2 = Dropout(0.1)(dense_layer_2)

dense_layer_3 = Dense(128, activation='relu')(dropout_layer_2)
dropout_layer_3 = Dropout(0.1)(dense_layer_3)

dense_layer_4 = Dense(64, activation='relu')(dropout_layer_3)
dropout_layer_4 = Dropout(0.1)(dense_layer_4)

dense_layer_5 = Dense(32, activation='relu')(dropout_layer_4)
dropout_layer_5 = Dropout(0.1)(dense_layer_5)

output_layer = Dense(1, activation='linear')(dense_layer_5)

# Erstellung des Modells
model_dense_embedding = Model(inputs=[input_1, input_2],
outputs=output_layer, name='dense_embedding')

# Kompilieren des Modells
optimizer = Adam(learning_rate=0.005)
```

```
model_dense_embedding.compile(optimizer=optimizer, loss='mse',
metrics=['mae'])

# Zusammenfassung des Modells
model_dense_embedding.summary()
```

```
Model: "dense_embedding"
__________________________________________________________________________
_______________________
 Layer (type)                Output Shape              Param #
Connected to
==========================================================================
===========================
 tekst (InputLayer)          [(None, 10)]              0           []



 embedding_5 (Embedding)     (None, 10, 32)            32000
['tekst[0][0]']


 tabela (InputLayer)         [(None, 877)]             0           []



 flatten_2 (Flatten)         (None, 320)               0
['embedding_5[0][0]']


 concatenate_4 (Concatenate)    (None, 1197)           0
['tabela[0][0]',

'flatten_2[0][0]']


 dense_12 (Dense)            (None, 512)               613376
['concatenate_4[0][0]']


 dropout_10 (Dropout)        (None, 512)               0
['dense_12[0][0]']


 dense_13 (Dense)            (None, 256)               131328
['dropout_10[0][0]']


 dropout_11 (Dropout)        (None, 256)               0
['dense_13[0][0]']
```

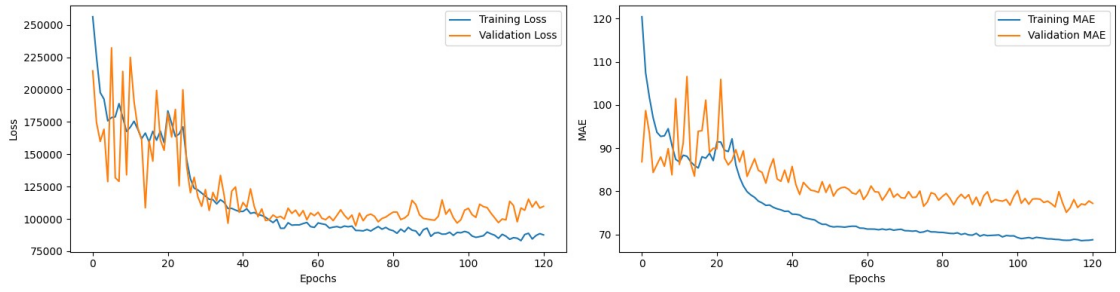
```
 dense_14 (Dense)               (None, 128)          32896
['dropout_11[0][0]']


 dropout_12 (Dropout)           (None, 128)          0
['dense_14[0][0]']


 dense_15 (Dense)               (None, 64)           8256
['dropout_12[0][0]']


 dropout_13 (Dropout)           (None, 64)           0
['dense_15[0][0]']


 dense_16 (Dense)               (None, 32)           2080
['dropout_13[0][0]']


 dense_17 (Dense)               (None, 1)            33
['dense_16[0][0]']


==================================================================================
Total params: 819,969
Trainable params: 819,969
Non-trainable params: 0
__________________________________________________________________________________
```

```python
# Callback's
early_stopping = EarlyStopping(monitor='val_loss', patience=50,
restore_best_weights=True)
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.1,
patience=10, min_lr=0.0001)

callbacks = [early_stopping, reduce_lr]

# Training des Modells
history_dense_embedding = model_dense_embedding.fit([X_train_tensor,
padded_sequences_train], y_train, epochs=500, batch_size=256,
validation_data=([X_test_tensor, padded_sequences_test], y_test),
callbacks=callbacks, verbose=1)

# Evaluation des Modells
loss_dense_embedding, mae_dense_embedding =
```

```
model_dense_embedding.evaluate([X_test_tensor, padded_sequences_test],
y_test)
print('Strata:', loss_dense_embedding)
print('MAE:', mae_dense_embedding)
```

```
1131/1131 [==============================] - 2s 2ms/step - loss:
94630.7891 - mae: 78.3972
Strata: 94630.7890625
MAE: 78.39720916748047
```

```
# Abrufen von Daten aus der Historie
loss = history_dense_embedding.history['loss']
val_loss = history_dense_embedding.history['val_loss']
mae = history_dense_embedding.history['mae']
val_mae = history_dense_embedding.history['val_mae']

# Visualisierung der Verlaufsdaten der Verlustfunktion
plt.figure(figsize=(15, 4))
plt.subplot(1, 2, 1)
plt.plot(loss, label='Training Loss')
plt.plot(val_loss, label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

# Visualisierung des Verlaufs der MAE-Metrik
plt.subplot(1, 2, 2)
plt.plot(mae, label='Training MAE')
plt.plot(val_mae, label='Validation MAE')
plt.xlabel('Epochs')
plt.ylabel('MAE')
plt.legend()

plt.tight_layout()
plt.show()
```




```
# Kodierung
X_train_dense_embedding = pipeline.fit_transform(X_train)
X_test_dense_embedding  = X_test.loc[X_test['ART_ID'] == 158]
padded_sequences_test_one_article =
padded_sequences_test[X_test['ART_ID'] == 158]
```

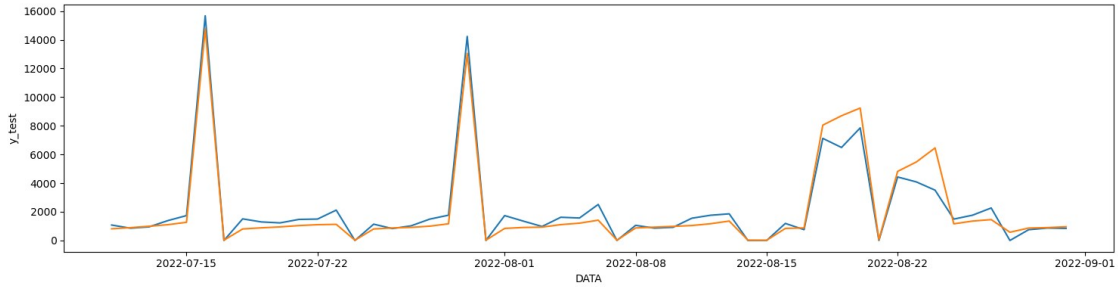
```
X_test_dense_embedding = pipeline.transform(X_test_dense_embedding)
# Erstellungen von Predictions
y_predict_dense_embedding =
model_dense_embedding.predict([X_test_dense_embedding,
padded_sequences_test_one_article])
```

```
2/2 [==============================] - 0s 5ms/step
```

```
# Erstellung eines DataFrame zur Visualisierung
df_dense_embedding = df.loc[df['ART_ID'] == 158]
dense_pred_embedding = df_dense_embedding.loc[df_dense_embedding.index
>'2022-07-10 00:00:00'][['y']]
dense_pred_embedding['y_test'] = dense_pred_embedding['y']
dense_pred_embedding = dense_pred_embedding.drop('y', axis=1)
dense_pred_embedding['y_predict'] = y_predict_dense_embedding
dense_pred_embedding.shape
```

```
(52, 2)
```

```
# Grafik - Vergleich der Vorhersagen
plt.figure(figsize=(15, 4))
sns.lineplot(data=dense_pred_embedding, x=dense_pred_embedding.index,
y='y_test')
sns.lineplot(data=dense_pred_embedding, x=dense_pred_embedding.index,
y='y_predict')
plt.tight_layout()
plt.show()
```




```
# Metriken
mae_dense_embedding, rmse_dense_embedding, r2_dense_embedding,
wmape_dense_embedding, mase_dense_embedding =
ff.metryki(dense_pred_embedding['y_test'],
dense_pred_embedding['y_predict'])
print(f'MAE: {mae_dense_embedding:.4f}, RMSE:
{rmse_dense_embedding:.4f}, R2: {r2_dense_embedding:.4f}, WMAPE:
{wmape_dense_embedding:.4f}, MASE: {mase_dense_embedding:.4f}')
```

```
MAE: 490.3275, RMSE: 743.3613, R2: 0.9404, WMAPE: 0.2441, MASE: 0.2393
```

Bewertung

Bewertung der Ergebnisse

```
# Erstellung eines Wörterbuchs mit modellspezifischen Metriken für
eindimensionale Zeitreihen
univariate_metrics = {
    'KNN': [mae_KNR, rmse_KNR, r2_KNR, wmape_KNR, mase_KNR],
    'SARIMAX': [mae_SARIMAX, rmse_SARIMAX, r2_SARIMAX, wmape_SARIMAX,
mase_SARIMAX],
    'XGB Boosting': [mae_XGB, rmse_XGB, r2_XGB, wmape_XGB, mase_XGB],
    'Prophet': [mae_prophet, rmse_prophet, r2_prophet, wmape_prophet,
mase_prophet ],
    'XGB+Prophet': [mae_XGB_Prophet, rmse_XGB_Prophet, r2_XGB_Prophet,
wmape_XGB_Prophet, mase_XGB_Prophet],
    'Vanilla Dense': [mae_dense_vanilla, rmse_dense_vanilla,
r2_dense_vanilla, wmape_dense_vanilla, mase_dense_vanilla],
    'Dense': [mae_dense, rmse_dense, r2_dense, wmape_dense,
mase_dense],
    'LSTM': [mae_lstm, rmse_lstm, r2_lstm, wmape_lstm, mase_lstm ]
    }

# Erstellung von DataFrame
univariate_metrics_df = pd.DataFrame(univariate_metrics, index=['MAE',
'RMSE', 'R2', 'WMAPE', 'MASE'])
univariate_metrics_df = univariate_metrics_df.round(decimals=4)

# Erstellung eines Wörterbuchs mit Metriken für individuelle
multivariate Zeitreihenmodelle
multivariate_metrics = {
    'XGB Boosting': [mae_XGB_2, rmse_XGB_2, r2_XGB_2, wmape_XGB_2,
mase_XGB_2],
    'Dense': [mae_dense_2, rmse_dense_2, r2_dense_2, wmape_dense_2,
mase_dense_2],
    'Dense+Embedding': [mae_dense_embedding, rmse_dense_embedding,
r2_dense_embedding, wmape_dense_embedding, mase_dense_embedding]
    }

# Erstellung von DataFrame
multivariate_metrics_df = pd.DataFrame(multivariate_metrics,
index=['MAE', 'RMSE', 'R2', 'WMAPE', 'MASE'])
multivariate_metrics_df = multivariate_metrics_df.round(decimals=4)

# Anzeige des DataFrame
univariate_metrics_df.T
```

	MAE	RMSE	R2	WMAPE	MASE
KNN	1385.8794	2331.3765	0.4135	0.5490	0.6765
SARIMAX	1047.7151	1975.9936	0.5787	0.3844	0.5114
XGB Boosting	766.6458	1325.4833	0.8104	0.3697	0.3742
Prophet	821.7948	2020.3256	0.5553	0.2295	0.4048

```
XGB+Prophet     728.5372  1511.2217  0.7536  0.3224  0.3556
Vanilla Dense   455.3268   796.9039  0.9315  0.2198  0.2223
Dense           536.9332   934.4254  0.9058  0.2388  0.2621
LSTM            457.9334   751.2274  0.9391  0.2200  0.2235
```

Die in der Tabelle dargestellten Ergebnisse sind Qualitätsmaße für acht Vorhersagemodelle. Hier ist die Interpretation der einzelnen Maße:

- **MAE** (Mean Absolute Error) misst den durchschnittlichen absoluten Unterschied zwischen den vorhergesagten und tatsächlichen Werten. Je niedriger der Wert, desto besser die Qualität des Modells. Die Modelle "Vanilla Dense" (455.3268) und "LSTM" (457.9334) erzielen die besten Ergebnisse, was darauf hinweist, dass sie präzisere Vorhersagen liefern.
- **RMSE** (Root Mean Square Error) ist die Quadratwurzel des durchschnittlichen quadratischen Fehlers zwischen den vorhergesagten und tatsächlichen Werten. Je niedriger der Wert, desto besser die Qualität des Modells. Ähnlich wie bei MAE erzielen die Modelle "Vanilla Dense" (796.9039) und "LSTM" (751.2274) die besten Ergebnisse.
- **R2** (Bestimmtheitsmaß) gibt an, wie gut das Modell zu den Daten passt. Ein Wert von 1 bedeutet eine perfekte Anpassung, während ein Wert von 0 auf keine Anpassung hinweist. Das höchste Ergebnis wurde für das "LSTM"-Modell (0.9391) erzielt, was darauf hindeutet, dass dieses Modell die besten Anpassungen an die Daten aufweist.
- **WMAPE** (Weighted Mean Absolute Percentage Error) ist der gewichtete durchschnittliche prozentuale Fehler zwischen den vorhergesagten und tatsächlichen Werten. Je niedriger der Wert, desto besser die Qualität des Modells. Das beste Ergebnis wurde mit dem "Vanilla Dense"-Modell (0.2198) erzielt, was darauf hindeutet, dass es am präzisesten in der Vorhersage von prozentualen Fehlern ist.
- MASE (Mean Absolute Scaled Error) vergleicht den Modellfehler mit dem Fehler eines "naiven" Modells (das den Wert aus dem vorherigen Zeitraum vorhersagt). Je niedriger der Wert, desto besser die Qualität des Modells. Die besten Ergebnisse wurden für die Modelle "Vanilla Dense" (0.2223) und "LSTM" (0.2235) erzielt, was auf ihre gute Fähigkeit zur Vorhersage basierend auf früheren Werten hinweist.

Zusammenfassend scheinen die "Vanilla Dense" und "LSTM"-Modelle die besten Vorhersagemodelle basierend auf den analysierten Qualitätsmaßen zu sein. Beide Modelle zeigen geringe Fehlerwerte und eine gute Anpassung an die Daten. Insbesondere das "LSTM"-Modell zeichnet sich durch einen hohen Bestimmtheitskoeffizienten (R2) aus, was darauf hinweist, dass es gut in der Lage ist, Werte basierend auf den analysierten Daten vorherzusagen.

```
# Anzeige des DataFrame
multivariate_metrics_df.T
```

	MAE	RMSE	R2	WMAPE	MASE
XGB Boosting	739.3627	1515.0146	0.7523	0.2758	0.3609
Dense	529.8080	920.3733	0.9086	0.2332	0.2586
Dense+Embedding	490.3275	743.3613	0.9404	0.2441	0.2393

Die in der Tabelle dargestellten Ergebnisse sind Qualitätsmaße für drei verschiedene Vorhersagemodelle. Hier ist die Interpretation der einzelnen Maße:

- **MAE** (Mean Absolute Error) - Durchschnittlicher absoluter Fehler: Der niedrigste MAE-Wert gehört zum Modell "Dense+Embedding" und beträgt 490.
- **RMSE** (Root Mean Square Error) - Quadratwurzel des durchschnittlichen quadratischen Fehlers: Erneut hat das Modell "Dense+Embedding" den niedrigsten RMSE-Wert, was auf eine geringere Fehlervarianz in den Prognosen dieses Modells hinweist.
- **R2** (Bestimmtheitskoeffizient) - Bestimmtheitsmaß: Das Modell "Dense+Embedding" hat den höchsten R2-Wert von 0,9404, was darauf hindeutet, dass dieses Modell die Datenvarianz besser erklärt.
- **WMAPE** (Weighted Mean Absolute Percentage Error) - Gewichteter durchschnittlicher absoluter prozentualer Fehler: Der WMAPE-Wert für das Modell "XGB Boosting" beträgt 0,2758, für das Modell "Dense" 0,2332 und für das Modell "Dense+Embedding" 0,2441. Das Modell "Dense" hat den niedrigsten WMAPE-Wert, was auf eine geringere prozentuale Differenz zwischen den vorhergesagten und tatsächlichen Werten hinweist.
- **MASE** (Mean Absolute Scaled Error) - Durchschnittlicher skalierte absolute Fehler: Erneut hat das Modell "Dense+Embedding" den niedrigsten MASE-Wert von 0,2393, was auf eine geringere normierte Differenz zwischen den vorhergesagten und tatsächlichen Werten hinweist.

Zusammenfassend erzielt das Modell "Dense+Embedding" die besten Ergebnisse für die meisten Maße, was darauf hindeutet, dass es am effektivsten bei der Vorhersage von Werten ist.

Implementierung

Geplante Umsetzung

Die Möglichkeit, ein neuronales Dense-Modell mit einer Embedding-Schicht einzusetzen, das die besten Ergebnisse erzielt hat, kann viele Vorteile bringen. Vor der Implementierung eines solchen Modells in die Produktion müssen jedoch bestimmte Bedingungen erfüllt sein und eine geeignete technische Infrastruktur bereitgestellt werden.

Bedingungen für die Implementierung eines neuronalen Dense-Modells mit Embedding-Schicht:

- Stabiler Datenpipeline: Es ist erforderlich, einen stabilen und skalierbaren Prozess zur Vorbereitung der Eingabedaten für das Modell zu erstellen. Dies umfasst die Tokenisierung von Text, die Umwandlung in numerische Repräsentationen mithilfe von Embeddings sowie eventuelle Datenbereinigungen oder -normalisierungen. Es ist wichtig, dass dieser Pipeline zuverlässig und flexibel ist, um sich ändernden Daten gerecht zu werden.
- Technische Infrastruktur: Um das Modell in die Produktion zu bringen, ist eine geeignete technische Infrastruktur erforderlich. Dies kann ausreichende Rechenleistung, Zugang zu GPU-Ressourcen, eine skalierbare Cloud-Umgebung, geeignete Tools und Bibliotheken zur Modellverarbeitung sowie ein Überwachungs- und Management-System umfassen.

Schritte zur Implementierung des Modells in die Produktion:

1. Datenbereitstellung: Bereitstellung einer stabilen und konfigurierten Datenpipeline zur Vorbereitung der Eingabedaten.

2. Modelltraining und -optimierung: Durchführung des Trainings und der Optimierung des Modells auf den vorbereiteten Daten. Dies umfasst die Auswahl optimaler Hyperparameter, Kreuzvalidierung und das Monitoring der Modellqualitätsmetriken.

3. Modellimplementierung: Implementierung des Modells in einer Produktionsumgebung, die Cloud- oder lokale Ressourcen umfassen kann. Dies erfordert eine entsprechende Anpassung des Modellcodes an die Produktionsumgebung und die Integration mit anderen Systemkomponenten.

4. Testen und Überprüfen: Gründliches Testen und Überprüfen des Modells in der Produktionsumgebung, um sicherzustellen, dass es wie erwartet funktioniert und genaue Vorhersagen liefert.

Vorteile:

- Das neuronale Dense-Modell mit Embedding-Schicht kann gute Ergebnisse bei der Analyse von Zeitreihendaten liefern, insbesondere wenn semantische Abhängigkeiten zwischen Wörtern/Artikeln vorhanden sind.
- Es kann die Fähigkeit des Modells verbessern, Muster zu erkennen, die zwischen Artikeln mit ähnlichem Inhalt auftreten.

Nachteile:

- Die Implementierung eines neuronalen Dense-Modells mit Embedding-Schicht erfordert angemessene Rechenressourcen und technische Infrastruktur.

- Das Modell kann große Trainingsdatensätze und eine längere Trainingszeit erfordern, was kostenintensiv sein kann.

Chancen:

- Möglichkeit zur Verbesserung der Genauigkeit und Effektivität des Modells bei der Textanalyse.
- Kann die Erkennung von Mustern und die Vorhersage in Zeitreihendaten signifikant verbessern.

Risiken:

- Das Modell kann empfindlich auf die Qualität der Eingabedaten reagieren und möglicherweise schlechte Ergebnisse liefern, wenn keine ausreichenden Trainingsdaten vorhanden sind.
- Es kann erforderlich sein, das Modell häufig zu aktualisieren, um sich verändernde Trends und semantische Abhängigkeiten im Text zu berücksichtigen. Dies erfordert kontinuierliche Überwachung und Aktualisierung des Modells, um sicherzustellen, dass es mit den neuesten Daten und Informationen arbeitet.

Es ist wichtig, diese potenziellen Risiken zu berücksichtigen und geeignete Maßnahmen zu ergreifen, um sicherzustellen, dass das implementierte Modell zuverlässig und effektiv arbeitet. Dazu gehört die regelmäßige Überprüfung der Modellleistung, das Monitoring von Metriken und das Feedback von Benutzern, um mögliche Probleme zu identifizieren und entsprechende Anpassungen vorzunehmen.