Capstone Projekt Rossmann

XDi - Certified Data Scientist

Christoph Gödecke

Predictive Modeling

Selection of models that will be tested to predict the sales of Rossmann stores

I want to use:
Linear Regression Models:

- LinearRegression
- RidgeRegression
- LassoRegression

Tree-based Models:

- DecisionTreeRegressor
- RandomForestRegressor
- GradiantBoostingRegressor
- XGBRegressor

Support Vector Machines (SVM):

- SVR

Nearest Neighbors:

- KNN

Neural Networks:

- KerasRegressor
- MLPRegressor

Time Series Models:

- Prophet

Definition of KPIs for model evaluation

- Mean Absolute Error (MAE): The average of the absolute differences between predictions and actual values. It gives an idea of the magnitude of the error, but no information about the direction (over or under predicting).
- Mean Squared Error (MSE): The average of the squared differences between predictions and actual values. It gives more weight to larger errors and is more useful in practice than MAE.
- Root Mean Squared Error (RMSE): The square root of the MSE, it is more interpretable than the MSE as it is in the same units as the response variable.
- R-squared (R2): The proportion of the variance in the dependent variable that is predictable from the independent variables. It provides an indication of the goodness of fit and therefore a measure of how well unseen samples are likely to be predicted by the model, through the proportion of explained variance.
- Adjusted R-squared: The R-squared value adjusted for the number of predictors in the model. It is useful for comparing models with different numbers of predictors.

-> I will focus on MAE and R2

Functions needed for testing

In [18]:
```
import pandas as pd
import numpy as np
from datetime import datetime

import seaborn as sns
import matplotlib.pyplot as plt
import plotly.express as px
from pandas.api.types import infer_dtype

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.svm import SVR
from sklearn.neighbors import KNeighborsRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.neural_network import MLPRegressor

from xgboost import XGBRegressor
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from scikeras.wrappers import KerasRegressor

from prophet import Prophet
from prophet.diagnostics import cross_validation
from tqdm import tqdm
tqdm.pandas(disable=True)  # Für pandas Operationen, falls verwendet

from sklearn.metrics import mean_absolute_error as mae, mean_squared_error as mse, r2_score
from sklearn.metrics import mean_absolute_error, r2_score
from math import sqrt

from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import RobustScaler

pd.set_option('display.max_columns', None)

import logging
logging.getLogger('cmdstanpy').setLevel(logging.WARNING)
import warnings
warnings.filterwarnings(action='ignore', category=UserWarning, module='xgboost')
```

Test models with test and tain data. Test includes the last 8 weeks from each store

In [2]:
```
## Test models with test and tain data. Test includes the last 8 weeks from each store

def build_neural_network(X_train):
    model = Sequential()
    model.add(Dense(128, activation='relu', input_dim=X_train.shape[1]))
    model.add(Dropout(0.2))
    model.add(Dense(64, activation='relu'))
    model.add(Dropout(0.2))
    model.add(Dense(1, activation='linear'))
    model.compile(optimizer='adam', loss='mean_squared_error')
```

```
    return model

def testModelsTestSplit8W(df, scaler):

        train_data = []
        test_data = []

        # Group by store and split into training and test data
        amount_test_weeks = 8
        for store_id, group in df.groupby('Store'):
                train_data.append(group[: -amount_test_weeks])
                test_data.append(group[-amount_test_weeks:])

        # Combine the list entries to one dataframe
        train_df = pd.concat(train_data)
        test_df = pd.concat(test_data)

        # Create feature and target data frames
        X_train = train_df.drop(columns=['Future_Sales'])
        y_train = train_df['Future_Sales']
        X_test = test_df.drop(columns=['Future_Sales'])
        y_test = test_df['Future_Sales']

        # Scaling of the data
        if scaler:
                print("Scaler applied")
                X_train = scaler.fit_transform(X_train)
                X_test = scaler.transform(X_test)

        def adj_r2_score(model, X, y):
                n = X.shape[0]
                p = X.shape[1]
                r2 = r2_score(y, model.predict(X))
                return 1 - (1 - r2) * ((n - 1) / (n - p - 1))

        # Defining the models to test
        models = [
                ('LinearRegression', LinearRegression(n_jobs=-1)),
                ('XGBRegressor', XGBRegressor(objective='reg:squarederror', n_estimators=100, max_depth=3, learning_rate=0.1, n_jobs=-1, random_state=42, device="cuda")),
                ('XGBRegressor_grid', XGBRegressor(objective='reg:squarederror', colsample_bytree=0.7, learning_rate= 0.01, max_depth= 5, n_estimators= 500, subsample= 0.8, random_state=42, n_jobs=-1, dev
                #('GradientBoostingRegressor', GradientBoostingRegressor(n_estimators=100, learning_rate=0.1, max_depth=3, random_state=42)),
                #('NeuralNetwork', KerasRegressor(build_fn=build_neural_network(X_train), epochs=100, batch_size=10, verbose=0)),
                #('MLPRegressor', MLPRegressor(hidden_layer_sizes=(100,), activation='relu', solver='adam', max_iter=200, shuffle=False, random_state=42))
                #('RidgeRegression', Ridge(random_state=42)),
                #('LassoRegression', Lasso(random_state=42)),
                #('DecisionTreeRegressor', DecisionTreeRegressor(random_state=42)),
                #('RandomForestRegressor', RandomForestRegressor(n_jobs=-1, max_depth=10, random_state=42, n_estimators=100)),
                #('SVR', SVR()),
                #('KNN', KNeighborsRegressor())
        ]

        results = []
        # Train models and calculate metrics
        for name, model in models:
                model.fit(X_train, y_train)
                y_train_pred = model.predict(X_train)
                y_test_pred = model.predict(X_test)

                results.append({
                        'Model': name,
                        'RMSE_Train': sqrt(mse(y_train, y_train_pred)),
                        'MAE_Train': mae(y_train, y_train_pred),
                        'R2_Train': r2_score(y_train, y_train_pred),
                        'Adj_R2_Train': adj_r2_score(model, X_train, y_train),
                        'RMSE_Test': sqrt(mse(y_test, y_test_pred)),
                        'MAE_Test': mae(y_test, y_test_pred),
                        'R2_Test': r2_score(y_test, y_test_pred),
                        'Adj_R2_Test': adj_r2_score(model, X_test, y_test)
                })
                #print last result
                print(results[-1])

        results_df = pd.DataFrame(results)
        return results_df
```

Cross Validation - Creates x splits in test and train where the last 8 weeks of each store are included in the respective test split and the splits are distributed evenly using gap

In [14]:

```
#Creates x splits in test and train where the last 8 weeks of each store are included in the respective test split and the splits are distributed
# evenly using gap

def testModelsCV8W(df, scaler):

    n_splits = 5
    window_size = 8
    total_weeks =109
    train_size = window_size / 0.2
    gap = int((total_weeks - window_size - train_size) // (n_splits))

    results = []

    for split in range(n_splits):
        train_data = []
        test_data = []

        for store_id, group in df.groupby('Store'):
            # calculate start and end index for test data
            if split == 0:
                test_start_index = -window_size
                test_df_store = group[test_start_index:] # no end index for the first split
            else:
                test_start_index = -(window_size + gap * split)
                test_end_index = test_start_index + window_size
                test_df_store = group[test_start_index:test_end_index]
                #print("test:", test_df_store.shape, "Test Start Index:", test_start_index, "Test End Index:", test_end_index)

            train_start_index = -int(-test_start_index + gap + train_size)
            train_df_store = group[train_start_index:test_start_index]
            #print("Train:", train_df_store.shape, "Train Start Index:", train_start_index, "Train End Index:", test_start_index)
            # Check if test set contains data
            if not test_df_store.empty:
                train_data.append(train_df_store)
                test_data.append(test_df_store)
            else:
                print(f"Store {store_id} has not enough data for splitting {split}")

        # Combine the list entries to one dataframe
        train_df_combined = pd.concat(train_data)
        test_df_combined = pd.concat(test_data)

        # Create feature and target data frames
        X_train = train_df_combined.drop(columns=['Future_Sales'])
        y_train = train_df_combined['Future_Sales']
        X_test = test_df_combined.drop(columns=['Future_Sales'])
        y_test = test_df_combined['Future_Sales']

        # Scaling of the data
        if scaler:
            X_train = scaler.fit_transform(X_train)
            X_test = scaler.transform(X_test)
```

```python
        def adj_r2_score(y_test_pred, X, y_test):
            n = X.shape[0]
            p = X.shape[1]
            r2 = r2_score(y_test, y_test_pred)
            return 1 - (1 - r2) * ((n - 1) / (n - p - 1))

        # Defining the models to test
        models = [
                ('LinearRegression', LinearRegression(n_jobs=-1)),
                ('XGBRegressor', XGBRegressor(objective='reg:squarederror', n_estimators=100, max_depth=3, learning_rate=0.1, n_jobs=-1, random_state=42, device="cuda")),
                #('GradientBoostingRegressor', GradientBoostingRegressor(n_estimators=100, learning_rate=0.1, max_depth=3, random_state=42)),
                #('NeuralNetwork', KerasRegressor(build_fn=build_neural_network(X_train), epochs=100, batch_size=10, verbose=0)),
                #('MLPRegressor', MLPRegressor(hidden_layer_sizes=(100,), activation='relu', solver='adam', max_iter=200, shuffle=False, random_state=42))
                #('RidgeRegression', Ridge(random_state=42)),
                #('LassoRegression', Lasso(random_state=42)),
                #('DecisionTreeRegressor', DecisionTreeRegressor(random_state=42)),
                #('RandomForestRegressor', RandomForestRegressor(n_jobs=-1, max_depth=10, random_state=42, n_estimators=100)),
                #('SVR', SVR()),
                #('KNN', KNeighborsRegressor())
        ]

        # Train models and calculate metrics
        for name, model in models:
            model.fit(X_train, y_train)
            y_train_pred = model.predict(X_train)
            y_test_pred = model.predict(X_test)

            results.append({
                'Model': name,
                'RMSE_Train': sqrt(mse(y_train, y_train_pred)),
                'MAE_Train': mae(y_train, y_train_pred),
                'R2_Train': r2_score(y_train, y_train_pred),
                'Adj_R2_Train': adj_r2_score(y_test_pred, X_test, y_test),
                'RMSE_Test': sqrt(mse(y_test, y_test_pred)),
                'MAE_Test': mae(y_test, y_test_pred),
                'R2_Test': r2_score(y_test, y_test_pred),
                'Adj_R2_Test': adj_r2_score(y_test_pred, X_test, y_test)
            })
            #print last result
            print(results[-1])

    results_df = pd.DataFrame(results)

    # calculate mean of all splits
    model_list = results_df['Model'].unique()
    # create resulte_mean_df
    resulte_mean_df = pd.DataFrame(columns=results_df.columns)
    # iterate over model_list
    for model in model_list:
        # get mean of each model
        mean = results_df[results_df['Model'] == model].mean(numeric_only=True)
        mean['Model'] = model
        # append mean to resulte_mean_df
        resulte_mean_df = pd.concat([resulte_mean_df, pd.DataFrame([mean], columns=results_df.columns)], ignore_index=True)

    return results_df, resulte_mean_df
```

Function to create a model for single store forecast

In [4]:
```python
## Test models with test and tain data. Test includes the last 8 weeks from each store

def storeForecastTestSplit8W(model, df, scaler):
        train_data = []
        test_data = []

        # Group by store and split into training and test data
        amount_test_weeks = 8
        for store_id, group in df.groupby('Store'):
                train_data.append(group[: -amount_test_weeks])
                test_data.append(group[-amount_test_weeks:])

        # Combine the list entries to one dataframe
        train_df = pd.concat(train_data)
        test_df = pd.concat(test_data)

        # Create feature and target data frames
        X_train = train_df.drop(columns=['Future_Sales'])
        y_train = train_df['Future_Sales']
        X_test = test_df.drop(columns=['Future_Sales'])
        y_test = test_df['Future_Sales']

        # Scaling of the data
        if scaler:
                X_train = scaler.fit_transform(X_train)
                X_test = scaler.transform(X_test)

        # Train models and calculate metrics
        model.fit(X_train, y_train)
        return model, scaler
```

Performance Reference (rolling mean)

In [5]:
```python
df = pd.read_csv('df_nans_handeled_cat_power.csv')
```

Performance Reference with one model for all stores

In [6]:
```python
df_to_use = df[['Store', 'Future_Sales', 'Open', 'Promo', 'IsPromo', 'StateHoliday_b', 'Sales_Lag_1',
        'Sales_Lag_1_MA_6', 'Sales_Lag_1_MA_8', 'Sales_Lag_2_MA_8',
       'Sales_Lag_3_MA_8', 'Sales_Lag_5_MA_8', 'Sales_Lag_8_MA_8',
       'SalesPerCustomer_Lag_3_MA_6', 'SalesPerCustomer_Lag_8',
       'SalesPerOpenDay_Lag_1', 'SalesPerOpenDay_Lag_4_MA_6',
       'Customers_Lag_1_MA_6', 'Customers_Lag_3_MA_4', 'Customers_Lag_6',
       'Customers_Lag_8_MA_8', 'CustomersPerOpenDay_Lag_6']]

def adj_r2_score(r2, n, p):
    """
    Parameters:
    - r2: Das R^2 des Modells.
    - n: Die Anzahl der Beobachtungen.
    - p: Die Anzahl der erklärenden Variablen im Modell.
    """
    return 1 - (1 - r2) * ((n - 1) / (n - p - 1))

amount_test_weeks = 8
rolling_mean_weeks = 4

# Defining the models to test
models = [
('LinearRegression', LinearRegression(n_jobs=-1)),
('XGBRegressor', XGBRegressor(objective='reg:squarederror', n_estimators=100, max_depth=3, learning_rate=0.1, n_jobs=-1, random_state=42, device="cuda")),
#('GradientBoostingRegressor', GradientBoostingRegressor(n_estimators=100, learning_rate=0.1, max_depth=3, random_state=42)),
#('NeuralNetwork', KerasRegressor(build_fn=build_neural_network(X_train), epochs=100, batch_size=10, verbose=0)),
#('MLPRegressor', MLPRegressor(hidden_layer_sizes=(100,), activation='relu', solver='adam', max_iter=200, shuffle=False, random_state=42))
#('RidgeRegression', Ridge(random_state=42)),
#('LassoRegression', Lasso(random_state=42)),
#('DecisionTreeRegressor', DecisionTreeRegressor(random_state=42)),
```

```
#('RandomForestRegressor', RandomForestRegressor(n_jobs=-1, max_depth=10, random_state=42, n_estimators=100)),
#('SVR', SVR()),
#('KNN', KNeighborsRegressor())
]

results_df = []

for name, model in models:
    # Assuming storeForecastTestSplit8W is a function returning a fitted model and scaler for the entire dataset
    model, scaler = storeForecastTestSplit8W(model, df_to_use, MinMaxScaler())

    metrics = {'Model': name, 'Train_MAE': [], 'Train_R2': [], 'Train_Adj_R2': [], 'Test_MAE': [], 'Test_R2': [], 'Test_Adj_R2': [], 'Rolling_Mean_MAE': [], 'Rolling_Mean_R2': []}

    for store_id in df_to_use['Store'].unique():
        df_store = df_to_use[df_to_use['Store'] == store_id]

        # Calculate rolling mean for the entire store data
        rolling_mean = df_store['Future_Sales'].rolling(window=rolling_mean_weeks, min_periods=1).mean()

        # Split into training and test data using the last 8 weeks
        train_df = df_store[:-amount_test_weeks]
        test_df = df_store[-amount_test_weeks:]

        # Prepare features and targets
        X_train = train_df.drop(columns=['Future_Sales'])
        y_train = train_df['Future_Sales']
        X_test = test_df.drop(columns=['Future_Sales'])
        y_test = test_df['Future_Sales']

        # Scaling of the data
        X_train_scaled = scaler.transform(X_train)
        X_test_scaled = scaler.transform(X_test)

        # Make predictions
        y_train_pred = model.predict(X_train_scaled)
        y_test_pred = model.predict(X_test_scaled)

        # Use rolling mean as predictions for baseline model
        y_train_baseline = rolling_mean.loc[train_df.index]
        y_test_baseline = rolling_mean.loc[test_df.index]

        # Calculate metrics for ML model
        train_mae = mae(y_train, y_train_pred)
        train_r2 = r2_score(y_train, y_train_pred)
        train_adj_r2 = adj_r2_score(train_r2, y_train.shape[0], X_train.shape[1])

        test_mae = mae(y_test, y_test_pred)
        test_r2 = r2_score(y_test, y_test_pred)
        test_adj_r2 = adj_r2_score(test_r2, y_test.shape[0], X_test.shape[1])

        # Calculate metrics for rolling mean baseline
        baseline_mae = mae(y_test, y_test_baseline)
        baseline_r2 = r2_score(y_test, y_test_baseline)

        # Store results
        metrics['Train_MAE'].append(train_mae)
        metrics['Train_R2'].append(train_r2)
        metrics['Train_Adj_R2'].append(train_adj_r2)
        metrics['Test_MAE'].append(test_mae)
        metrics['Test_R2'].append(test_r2)
        metrics['Test_Adj_R2'].append(test_adj_r2)
        metrics['Rolling_Mean_MAE'].append(baseline_mae)
        metrics['Rolling_Mean_R2'].append(baseline_r2)

    # Aggregate and append the results for the current model
    results_df.append({
        'Model': name,
        'Train_MAE': np.mean(metrics['Train_MAE']),
        'Train_R2': np.mean(metrics['Train_R2']),
        'Train_Adj_R2': np.mean(metrics['Train_Adj_R2']),
        'Test_MAE': np.mean(metrics['Test_MAE']),
        'Test_R2': np.mean(metrics['Test_R2']),
        'Test_Adj_R2': np.mean(metrics['Test_Adj_R2']),
        'Rolling_Mean_MAE': np.mean(metrics['Rolling_Mean_MAE']),
        'Rolling_Mean_R2': np.mean(metrics['Rolling_Mean_R2'])
    })

# Convert results to DataFrame for display
results_df = pd.DataFrame(results_df)

results_df
```

Out[6]:

	Model	Train_MAE	Train_R2	Train_Adj_R2	Test_MAE	Test_R2	Test_Adj_R2	Rolling_Mean_MAE	Rolling_Mean_R2
0	LinearRegression	6673.629827	0.107103	-0.121566	4295.672176	0.161466	1.419267	5694.444367	0.025724
1	XGBRegressor	6669.648039	0.114468	-0.112315	4920.265378	0.126968	1.436516	5694.444367	0.025724

Pre Test with 20 Features and MinMaxScaler (Single Store Metric):

	Model	Train_MAE	Train_R2	Train_Adj_R2	Test_MAE	Test_R2	Test_Adj_R2	Rolling_Mean_MAE	Rolling_Mean_R2
0	LinearRegression	6673.629827	0.107103	-0.121566	4295.672176	0.161466	1.419267	5694.444367	0.025724
1	XGBRegressor	6669.648185	0.114468	-0.112315	4920.264590	0.126968	1.436516	5694.444367	0.025724
2	GradientBoostingRegressor	6606.003615	0.119351	-0.106181	4959.455672	0.109270	1.445365	5694.444367	0.025724
3	MLPRegressor	7492.932380	-0.037149	-0.302760	5197.128449	-0.100918	1.550459	5694.444367	0.025724
4	RidgeRegression	6818.104648	0.100474	-0.129893	4460.137452	0.158568	1.420716	5694.444367	0.025724
5	LassoRegression	6784.490264	0.104267	-0.125127	4392.647343	0.174918	1.412541	5694.444367	0.025724
6	DecisionTreeRegressor	235.779668	0.984534	0.980573	9774.419395	-4.751631	3.875816	5694.444367	0.025724
7	RandomForestRegressor	5933.068451	0.269086	0.081900	5180.182098	0.006682	1.496659	5694.444367	0.025724
8	SVR	11376.303163	-1.797806	-2.514317	9902.526045	-3.704320	3.352160	5694.444367	0.025724
9	KNN	5226.791195	0.390114	0.233924	6847.233610	-1.170525	2.085262	5694.444367	0.025724

Performance Reference with own model for each store

In [7]:
```
df_to_use = df[['Store', 'Future_Sales', 'Open', 'Promo', 'IsPromo', 'StateHoliday_b', 'Sales_Lag_1',
        'Sales_Lag_1_MA_6', 'Sales_Lag_1_MA_8', 'Sales_Lag_2_MA_8',
       'Sales_Lag_3_MA_8', 'Sales_Lag_5_MA_8', 'Sales_Lag_8_MA_8',
       'SalesPerCustomer_Lag_3_MA_6', 'SalesPerCustomer_Lag_8',
       'SalesPerOpenDay_Lag_1', 'SalesPerOpenDay_Lag_4_MA_6',
       'Customers_Lag_1_MA_6', 'Customers_Lag_3_MA_4', 'Customers_Lag_6',
       'Customers_Lag_8_MA_8', 'CustomersPerOpenDay_Lag_6']]


def adj_r2_score(r2, n, p):
```

```python
    return 1 - (1 - r2) * ((n - 1) / (n - p - 1))

amount_test_weeks = 8
rolling_mean_weeks = 4

models = [
    ('LinearRegression', LinearRegression(n_jobs=-1)),
    ('XGBRegressor', XGBRegressor(objective='reg:squarederror', n_estimators=100, max_depth=3, learning_rate=0.1, n_jobs=-1, random_state=42, device="cuda")),
    # Add 'Mean Model' as a placeholder for rolling mean calculations
    ('Mean Model', None),
]

results_df = []

scaler = MinMaxScaler()

for name, model in models:
    results = {'Train_R2': [], 'Train_Adj_R2': [], 'Train_MAE': [], 'Test_R2': [], 'Test_Adj_R2': [], 'Test_MAE': []}

    for store_id in df_to_use['Store'].unique():
        df_store = df_to_use[df_to_use['Store'] == store_id]

        # Calculate rolling mean for the entire dataset for this store
        entire_df_rolling_mean = df_store['Future_Sales'].rolling(window=rolling_mean_weeks, min_periods=1).mean()

        train_df = df_store[:-amount_test_weeks]
        test_df = df_store[-amount_test_weeks:]

        if name == 'Mean Model':
            # Use the pre-calculated rolling mean directly for train and test sets
            # Ensure the indices match for y_train and y_test with their respective predictions
            y_train_pred = entire_df_rolling_mean.loc[train_df.index]
            y_train = train_df['Future_Sales']

            y_test_pred = entire_df_rolling_mean.loc[test_df.index]
            y_test = test_df['Future_Sales']
        else:
            X_train = train_df.drop(columns=['Future_Sales']).dropna()
            y_train = train_df.loc[X_train.index, 'Future_Sales']
            X_test = test_df.drop(columns=['Future_Sales']).dropna()
            y_test = test_df.loc[X_test.index, 'Future_Sales']

            scaler.fit(X_train)
            X_train_scaled = scaler.transform(X_train)
            X_test_scaled = scaler.transform(X_test)

            model.fit(X_train_scaled, y_train)

            y_train_pred = model.predict(X_train_scaled)
            y_test_pred = model.predict(X_test_scaled)

        # Calculate metrics for training and testing sets
        train_r2 = r2_score(y_train, y_train_pred)
        test_r2 = r2_score(y_test, y_test_pred)
        train_mae = mae(y_train, y_train_pred)
        test_mae = mae(y_test, y_test_pred)

        if name != 'Mean Model':
            train_adj_r2 = adj_r2_score(train_r2, y_train.size, X_train.shape[1])
            test_adj_r2 = adj_r2_score(test_r2, y_test.size, X_test.shape[1])
        else:
            # Adjusted R2 is not applicable for the 'Mean Model'
            train_adj_r2 = np.nan
            test_adj_r2 = np.nan

        results['Train_R2'].append(train_r2)
        results['Train_Adj_R2'].append(train_adj_r2)
        results['Train_MAE'].append(train_mae)
        results['Test_R2'].append(test_r2)
        results['Test_Adj_R2'].append(test_adj_r2)
        results['Test_MAE'].append(test_mae)

    avg_results = {k: np.mean(v) for k, v in results.items()}
    avg_results['Model'] = name
    results_df.append(avg_results)

results_df = pd.DataFrame(results_df).fillna('N/A')
results_df
```

Out[7]:

	Train_R2	Train_Adj_R2	Train_MAE	Test_R2	Test_Adj_R2	Test_MAE	Model
0	0.337336	0.16763	5910.366996	-0.552859	1.77643	5151.323838	LinearRegression
1	0.958393	0.947738	1300.982586	-1.564526	2.282263	6163.539893	XGBRegressor
2	0.347695	N/A	5844.124345	0.025724	N/A	5694.444367	Mean Model

Result:

- The results with just one model for all stores are better than the results with a model for each store (e.g. LinearRegression MAE 4295 vs 5151)
- The MAE of the simple mean forecast by rolling 4 weeks is 5694
- The result of the LinearRegression model forecast is with MAE:4295 and R2: 0.161466 the best. -> The model forecast is better (1399€/ -24,6%) then the simple mean forecast

Pre-Test of different Models

In [8]:
```python
df = pd.read_csv('df_nans_handeled_cat_power.csv')
```

Pre Test with 20 Features and MinMaxScaler and simple splitting train and test data for all stroes together:

	Model	RMSE_Train	MAE_Train	R2_Train	Adj_R2_Train	RMSE_Test	MAE_Test	R2_Test	Adj_R2_Test
0	LinearRegression	9536.329180	6673.629827	0.726042	0.725993	5775.476283	4295.672176	0.871601	0.871298
1	XGBRegressor	9457.068927	6669.648185	0.730577	0.730529	6298.189651	4920.264590	0.847307	0.846947
2	GradientBoostingRegressor	9351.755223	6606.003615	0.736545	0.736497	6378.135890	4959.455672	0.843406	0.843037
3	NeuralNetwork	9073.800592	6234.208134	0.751973	0.751928	6215.743831	4754.046991	0.851279	0.850928
4	MLPRegressor	10510.445004	7492.932380	0.667215	0.667155	6713.051469	5197.128449	0.826529	0.826119
5	RidgeRegression	9593.022556	6818.104648	0.722775	0.722725	5885.607949	4460.137452	0.866657	0.866342
6	LassoRegression	9580.166863	6784.490264	0.723518	0.723468	5828.177766	4392.647343	0.869247	0.868938
7	DecisionTreeRegressor	2251.202726	235.779668	0.984733	0.984730	14317.108685	9774.419395	0.210962	0.209100
8	RandomForestRegressor	8493.446640	5933.068451	0.782685	0.782646	6718.930475	5180.182098	0.826225	0.825815
9	SVR	16477.007577	11376.303163	0.182143	0.181994	14625.881350	9902.526045	0.176561	0.174618
10	KNN	7823.636664	5226.872647	0.815610	0.815576	9052.378771	6847.233610	0.684563	0.683819

In [9]:
```
pd.set_option('display.max_rows', 10000)
```

In [10]:
```
df.isna().sum()
```

```
Out[10]: Store                             0
         CW                                0
         Month                             0
         Year                              0
         Open                              0
         Promo                             0
         IsPromo                           0
         IsStateHoliday                    0
         SchoolHoliday                     0
         IsSchoolHoliday                   0
         NumStateHoliday                   0
         CompetitionDistance               0
         IsCompetition                     0
         Promo2                            0
         Promo2Member                      0
         Promo2Active                      0
         StateHoliday_0                    0
         StateHoliday_a                    0
         StateHoliday_b                    0
         StateHoliday_c                    0
         StoreType_a                       0
         StoreType_b                       0
         StoreType_c                       0
         StoreType_d                       0
         Assortment_a                      0
         Assortment_b                      0
         Assortment_c                      0
         PromoInterval_0                   0
         PromoInterval_Feb,May,Aug,Nov     0
         PromoInterval_Jan,Apr,Jul,Oct     0
         PromoInterval_Mar,Jun,Sept,Dec    0
         Sales_Lag_1                       0
         Sales_Lag_1_MA_4                  0
         Sales_Lag_1_MA_6                  0
         Sales_Lag_1_MA_8                  0
         Sales_Lag_2                       0
         Sales_Lag_2_MA_4                  0
         Sales_Lag_2_MA_6                  0
         Sales_Lag_2_MA_8                  0
         Sales_Lag_3                       0
         Sales_Lag_3_MA_4                  0
         Sales_Lag_3_MA_6                  0
         Sales_Lag_3_MA_8                  0
         Sales_Lag_4                       0
         Sales_Lag_4_MA_4                  0
         Sales_Lag_4_MA_6                  0
         Sales_Lag_4_MA_8                  0
         Sales_Lag_5                       0
         Sales_Lag_5_MA_4                  0
         Sales_Lag_5_MA_6                  0
         Sales_Lag_5_MA_8                  0
         Sales_Lag_6                       0
         Sales_Lag_6_MA_4                  0
         Sales_Lag_6_MA_6                  0
         Sales_Lag_6_MA_8                  0
         Sales_Lag_7                       0
         Sales_Lag_7_MA_4                  0
         Sales_Lag_7_MA_6                  0
         Sales_Lag_7_MA_8                  0
         Sales_Lag_8                       0
         Sales_Lag_8_MA_4                  0
         Sales_Lag_8_MA_6                  0
         Sales_Lag_8_MA_8                  0
         SalesPerCustomer_Lag_1            0
         SalesPerCustomer_Lag_1_MA_4       0
         SalesPerCustomer_Lag_1_MA_6       0
         SalesPerCustomer_Lag_1_MA_8       0
         SalesPerCustomer_Lag_2            0
         SalesPerCustomer_Lag_2_MA_4       0
         SalesPerCustomer_Lag_2_MA_6       0
         SalesPerCustomer_Lag_2_MA_8       0
         SalesPerCustomer_Lag_3            0
         SalesPerCustomer_Lag_3_MA_4       0
         SalesPerCustomer_Lag_3_MA_6       0
         SalesPerCustomer_Lag_3_MA_8       0
         SalesPerCustomer_Lag_4            0
         SalesPerCustomer_Lag_4_MA_4       0
         SalesPerCustomer_Lag_4_MA_6       0
         SalesPerCustomer_Lag_4_MA_8       0
         SalesPerCustomer_Lag_5            0
         SalesPerCustomer_Lag_5_MA_4       0
         SalesPerCustomer_Lag_5_MA_6       0
         SalesPerCustomer_Lag_5_MA_8       0
         SalesPerCustomer_Lag_6            0
         SalesPerCustomer_Lag_6_MA_4       0
         SalesPerCustomer_Lag_6_MA_6       0
         SalesPerCustomer_Lag_6_MA_8       0
         SalesPerCustomer_Lag_7            0
         SalesPerCustomer_Lag_7_MA_4       0
         SalesPerCustomer_Lag_7_MA_6       0
         SalesPerCustomer_Lag_7_MA_8       0
         SalesPerCustomer_Lag_8            0
         SalesPerCustomer_Lag_8_MA_4       0
         SalesPerCustomer_Lag_8_MA_6       0
         SalesPerCustomer_Lag_8_MA_8       0
         SalesPerOpenDay_Lag_1             0
         SalesPerOpenDay_Lag_1_MA_4        0
         SalesPerOpenDay_Lag_1_MA_6        0
         SalesPerOpenDay_Lag_1_MA_8        0
         SalesPerOpenDay_Lag_2             0
         SalesPerOpenDay_Lag_2_MA_4        0
         SalesPerOpenDay_Lag_2_MA_6        0
         SalesPerOpenDay_Lag_2_MA_8        0
         SalesPerOpenDay_Lag_3             0
         SalesPerOpenDay_Lag_3_MA_4        0
         SalesPerOpenDay_Lag_3_MA_6        0
         SalesPerOpenDay_Lag_3_MA_8        0
         SalesPerOpenDay_Lag_4             0
         SalesPerOpenDay_Lag_4_MA_4        0
         SalesPerOpenDay_Lag_4_MA_6        0
         SalesPerOpenDay_Lag_4_MA_8        0
         SalesPerOpenDay_Lag_5             0
         SalesPerOpenDay_Lag_5_MA_4        0
         SalesPerOpenDay_Lag_5_MA_6        0
         SalesPerOpenDay_Lag_5_MA_8        0
         SalesPerOpenDay_Lag_6             0
         SalesPerOpenDay_Lag_6_MA_4        0
         SalesPerOpenDay_Lag_6_MA_6        0
         SalesPerOpenDay_Lag_6_MA_8        0
         SalesPerOpenDay_Lag_7             0
         SalesPerOpenDay_Lag_7_MA_4        0
         SalesPerOpenDay_Lag_7_MA_6        0
         SalesPerOpenDay_Lag_7_MA_8        0
         SalesPerOpenDay_Lag_8             0
         SalesPerOpenDay_Lag_8_MA_4        0
         SalesPerOpenDay_Lag_8_MA_6        0
         SalesPerOpenDay_Lag_8_MA_8        0
         Customers_Lag_1                   0
         Customers_Lag_1_MA_4              0
         Customers_Lag_1_MA_6              0
         Customers_Lag_1_MA_8              0
```

```
Customers_Lag_2                 0
Customers_Lag_2_MA_4            0
Customers_Lag_2_MA_6            0
Customers_Lag_2_MA_8            0
Customers_Lag_3                 0
Customers_Lag_3_MA_4            0
Customers_Lag_3_MA_6            0
Customers_Lag_3_MA_8            0
Customers_Lag_4                 0
Customers_Lag_4_MA_4            0
Customers_Lag_4_MA_6            0
Customers_Lag_4_MA_8            0
Customers_Lag_5                 0
Customers_Lag_5_MA_4            0
Customers_Lag_5_MA_6            0
Customers_Lag_5_MA_8            0
Customers_Lag_6                 0
Customers_Lag_6_MA_4            0
Customers_Lag_6_MA_6            0
Customers_Lag_6_MA_8            0
Customers_Lag_7                 0
Customers_Lag_7_MA_4            0
Customers_Lag_7_MA_6            0
Customers_Lag_7_MA_8            0
Customers_Lag_8                 0
Customers_Lag_8_MA_4            0
Customers_Lag_8_MA_6            0
Customers_Lag_8_MA_8            0
CustomersPerOpenDay_Lag_1       0
CustomersPerOpenDay_Lag_1_MA_4  0
CustomersPerOpenDay_Lag_1_MA_6  0
CustomersPerOpenDay_Lag_1_MA_8  0
CustomersPerOpenDay_Lag_2       0
CustomersPerOpenDay_Lag_2_MA_4  0
CustomersPerOpenDay_Lag_2_MA_6  0
CustomersPerOpenDay_Lag_2_MA_8  0
CustomersPerOpenDay_Lag_3       0
CustomersPerOpenDay_Lag_3_MA_4  0
CustomersPerOpenDay_Lag_3_MA_6  0
CustomersPerOpenDay_Lag_3_MA_8  0
CustomersPerOpenDay_Lag_4       0
CustomersPerOpenDay_Lag_4_MA_4  0
CustomersPerOpenDay_Lag_4_MA_6  0
CustomersPerOpenDay_Lag_4_MA_8  0
CustomersPerOpenDay_Lag_5       0
CustomersPerOpenDay_Lag_5_MA_4  0
CustomersPerOpenDay_Lag_5_MA_6  0
CustomersPerOpenDay_Lag_5_MA_8  0
CustomersPerOpenDay_Lag_6       0
CustomersPerOpenDay_Lag_6_MA_4  0
CustomersPerOpenDay_Lag_6_MA_6  0
CustomersPerOpenDay_Lag_6_MA_8  0
CustomersPerOpenDay_Lag_7       0
CustomersPerOpenDay_Lag_7_MA_4  0
CustomersPerOpenDay_Lag_7_MA_6  0
CustomersPerOpenDay_Lag_7_MA_8  0
CustomersPerOpenDay_Lag_8       0
CustomersPerOpenDay_Lag_8_MA_4  0
CustomersPerOpenDay_Lag_8_MA_6  0
CustomersPerOpenDay_Lag_8_MA_8  0
Future_Sales                    0
dtype: int64
```

Feature Elimination

In [11]:
```
# Results with all features
testModelsTestSplit8W(df, MinMaxScaler())
```

```
Scaler applied
{'Model': 'LinearRegression', 'RMSE_Train': 9081.178921413037, 'MAE_Train': 6229.184072093825, 'R2_Train': 0.7515692210307627, 'Adj_R2_Train': 0.7511593471555716, 'RMSE_Test': 5941.163691018397, 'MAE_Test': 4395.438901345292, 'R2_Test': 0.8641279045091326, 'Adj_R2_Test': 0.8611545348667453}
{'Model': 'XGBRegressor', 'RMSE_Train': 7410.980873242441, 'MAE_Train': 5030.000189809052, 'R2_Train': 0.8345478857952568, 'Adj_R2_Train': 0.8342749143885373, 'RMSE_Test': 6058.148984876124, 'MAE_Test': 4515.474892052621, 'R2_Test': 0.858724407972136, 'Adj_R2_Test': 0.8556327904105729}
{'Model': 'XGBRegressor_grid', 'RMSE_Train': 6794.565583008348, 'MAE_Train': 4474.087838614279, 'R2_Train': 0.8609265186874501, 'Adj_R2_Train': 0.8606970681058499, 'RMSE_Test': 6232.247312329171, 'MAE_Test': 4603.617606808786, 'R2_Test': 0.8504878127705697, 'Adj_R2_Test': 0.8472159489116305}
```

Out[11]:

	Model	RMSE_Train	MAE_Train	R2_Train	Adj_R2_Train	RMSE_Test	MAE_Test	R2_Test	Adj_R2_Test
0	LinearRegression	9081.178921	6229.184072	0.751569	0.751159	5941.163691	4395.438901	0.864128	0.861155
1	XGBRegressor	7410.980873	5030.000190	0.834548	0.834275	6058.148985	4515.474892	0.858724	0.855633
2	XGBRegressor_grid	6794.565583	4474.087839	0.860927	0.860697	6232.247312	4603.617607	0.850488	0.847216

In [12]:
```
testModelsTestSplit8W(df[['Store', 'Future_Sales', 'CW', 'Month', 'Open', 'Promo', 'IsPromo', 'IsSchoolHoliday',
       'StateHoliday_a', 'StateHoliday_b', 'Assortment_b', 'Assortment_c',
       'Sales_Lag_1', 'Sales_Lag_1_MA_6', 'Sales_Lag_1_MA_8',
       'Sales_Lag_2_MA_8', 'Sales_Lag_3', 'Sales_Lag_3_MA_8',
       'Sales_Lag_4_MA_4', 'Sales_Lag_5_MA_8', 'Sales_Lag_7_MA_8',
       'Sales_Lag_8_MA_8', 'SalesPerCustomer_Lag_3_MA_6',
       'SalesPerCustomer_Lag_4_MA_6', 'SalesPerCustomer_Lag_7',
       'SalesPerCustomer_Lag_8', 'SalesPerOpenDay_Lag_1',
       'SalesPerOpenDay_Lag_4_MA_6', 'SalesPerOpenDay_Lag_5_MA_8',
       'SalesPerOpenDay_Lag_7', 'Customers_Lag_1_MA_4', 'Customers_Lag_1_MA_6',
       'Customers_Lag_3_MA_4', 'Customers_Lag_4', 'Customers_Lag_4_MA_8',
       'Customers_Lag_6', 'Customers_Lag_7_MA_8', 'Customers_Lag_8_MA_8',
       'CustomersPerOpenDay_Lag_6', 'CustomersPerOpenDay_Lag_6_MA_4',
       'CustomersPerOpenDay_Lag_7_MA_4', 'CustomersPerOpenDay_Lag_8']], MinMaxScaler())
```

```
Scaler applied
{'Model': 'LinearRegression', 'RMSE_Train': 9243.960853576737, 'MAE_Train': 6346.849079061771, 'R2_Train': 0.7425830551488228, 'Adj_R2_Train': 0.7424920072120149, 'RMSE_Test': 6076.866293178483, 'MAE_Test': 4604.18222365417, 'R2_Test': 0.857850086874331, 'Adj_R2_Test': 0.8571936162234916}
{'Model': 'XGBRegressor', 'RMSE_Train': 7586.747425730487, 'MAE_Train': 5128.276978852914, 'R2_Train': 0.8266067494562147, 'Adj_R2_Train': 0.8265454205575768, 'RMSE_Test': 6139.361351649675, 'MAE_Test': 4697.845359576443, 'R2_Test': 0.8549112868641222, 'Adj_R2_Test': 0.854241244372731}
{'Model': 'XGBRegressor_grid', 'RMSE_Train': 7143.720043187459, 'MAE_Train': 4777.614334612462, 'R2_Train': 0.8462660487949659, 'Adj_R2_Train': 0.8462116733571615, 'RMSE_Test': 6588.582946658205, 'MAE_Test': 5147.650721473865, 'R2_Test': 0.8329019920174645, 'Adj_R2_Test': 0.8321303071416722}
```

Out[12]:

	Model	RMSE_Train	MAE_Train	R2_Train	Adj_R2_Train	RMSE_Test	MAE_Test	R2_Test	Adj_R2_Test
0	LinearRegression	9243.960854	6346.849079	0.742583	0.742492	6076.866293	4604.182224	0.857850	0.857194
1	XGBRegressor	7586.747426	5128.276979	0.826607	0.826545	6139.361352	4697.845360	0.854911	0.854241
2	XGBRegressor_grid	7143.720043	4777.614335	0.846266	0.846212	6588.582947	5147.650721	0.832902	0.832130

In [13]:
```
testModelsTestSplit8W(df[['Store', 'Future_Sales', 'Open', 'Promo', 'IsPromo', 'StateHoliday_b', 'Sales_Lag_1',
       'Sales_Lag_1_MA_6', 'Sales_Lag_1_MA_8', 'Sales_Lag_2_MA_8',
       'Sales_Lag_3_MA_8', 'Sales_Lag_5_MA_8', 'Sales_Lag_8_MA_8',
       'SalesPerCustomer_Lag_3_MA_6', 'SalesPerCustomer_Lag_8',
       'SalesPerOpenDay_Lag_1', 'SalesPerOpenDay_Lag_4_MA_6',
       'Customers_Lag_1_MA_6', 'Customers_Lag_3_MA_4', 'Customers_Lag_6',
       'Customers_Lag_8_MA_8', 'CustomersPerOpenDay_Lag_6']], MinMaxScaler())
```

```
Scaler applied
{'Model': 'LinearRegression', 'RMSE_Train': 9536.329179581324, 'MAE_Train': 6673.629827190578, 'R2_Train': 0.7260423678114407, 'Adj_R2_Train': 0.7259927455109356, 'RMSE_Test': 5775.4762831794915, 'MAE_Test': 4295.6721755917715, 'R2_Test': 0.8716006437108907, 'Adj_R2_Test': 0.8712976108403501}
{'Model': 'XGBRegressor', 'RMSE_Train': 9457.068947317517, 'MAE_Train': 6669.648039198498, 'R2_Train': 0.7305773851169427, 'Adj_R2_Train': 0.7305285842499919, 'RMSE_Test': 6298.190214583256, 'MAE_Test': 4920.265377574972, 'R2_Test': 0.8473071214064962, 'Adj_R2_Test': 0.8469467538575567}
{'Model': 'XGBRegressor_grid', 'RMSE_Train': 9234.867751490798, 'MAE_Train': 6474.754588917429, 'R2_Train': 0.7430892380147797, 'Adj_R2_Train': 0.7430427034359385, 'RMSE_Test': 6288.75298016257, 'MAE_Test': 4914.484669501471, 'R2_Test': 0.8477643698672545, 'Adj_R2_Test': 0.8474050814616816}
```

Out[13]:

	Model	RMSE_Train	MAE_Train	R2_Train	Adj_R2_Train	RMSE_Test	MAE_Test	R2_Test	Adj_R2_Test
0	LinearRegression	9536.329180	6673.629827	0.726042	0.725993	5775.476283	4295.672176	0.871601	0.871298
1	XGBRegressor	9457.068947	6669.648039	0.730577	0.730529	6298.190215	4920.265378	0.847307	0.846947
2	XGBRegressor_grid	9234.867751	6474.754589	0.743089	0.743043	6288.752980	4914.484670	0.847764	0.847405

In [15]:

```
result_df, result_mean = testModelsCV8W(df[['Store', 'Future_Sales', 'Open', 'Promo', 'IsPromo', 'StateHoliday_b', 'Sales_Lag_1',
       'Sales_Lag_1_MA_6', 'Sales_Lag_1_MA_8', 'Sales_Lag_2_MA_8',
       'Sales_Lag_3_MA_8', 'Sales_Lag_5_MA_8', 'Sales_Lag_8_MA_8',
       'SalesPerCustomer_Lag_3_MA_6', 'SalesPerCustomer_Lag_8',
       'SalesPerOpenDay_Lag_1', 'SalesPerOpenDay_Lag_4_MA_6',
       'Customers_Lag_1_MA_6', 'Customers_Lag_3_MA_4', 'Customers_Lag_6',
       'Customers_Lag_8_MA_8', 'CustomersPerOpenDay_Lag_6']], MinMaxScaler())
print("Result_mean:", result_mean)
result_df
```

```
{'Model': 'LinearRegression', 'RMSE_Train': 10173.094867249885, 'MAE_Train': 6696.5698519947055, 'R2_Train': 0.7296512320463691, 'Adj_R2_Train': 0.8536973684802993, 'RMSE_Test': 6157.728962698496, 'MAE_Tes
t': 4441.731377054631, 'R2_Test': 0.8540418415447588, 'Adj_R2_Test': 0.8536973684802993}
{'Model': 'XGBRegressor', 'RMSE_Train': 9965.527271304221, 'MAE_Train': 6511.40832745491, 'R2_Train': 0.7405708517744283, 'Adj_R2_Train': 0.8319023848966189, 'RMSE_Test': 6600.476303651255, 'MAE_Test': 468
4.788395656706, 'R2_Test': 0.8322981747740907, 'Adj_R2_Test': 0.8319023848966189}
{'Model': 'LinearRegression', 'RMSE_Train': 9936.525631469427, 'MAE_Train': 6434.446275365776, 'R2_Train': 0.7399556315739964, 'Adj_R2_Train': 0.7303165247604226, 'RMSE_Test': 8612.752731085693, 'MAE_Tes
t': 6342.801958019805, 'R2_Test': 0.7309515009887029, 'Adj_R2_Test': 0.7303165247604226}
{'Model': 'XGBRegressor', 'RMSE_Train': 10079.341107384458, 'MAE_Train': 6649.8464043833965, 'R2_Train': 0.7324267925563053, 'Adj_R2_Train': 0.7481473657383917, 'RMSE_Test': 8323.15648246845, 'MAE_Test': 6
655.922918892762, 'R2_Test': 0.7487403588227615, 'Adj_R2_Test': 0.7481473657383917}
{'Model': 'LinearRegression', 'RMSE_Train': 9050.47328290472, 'MAE_Train': 5671.209864176966, 'R2_Train': 0.7806208377949678, 'Adj_R2_Train': 0.18495642679008673, 'RMSE_Test': 14869.83692970142, 'MAE_Tes
t': 8689.317001452782, 'R2_Test': 0.18687546648482922, 'Adj_R2_Test': 0.18495642679008673}
{'Model': 'XGBRegressor', 'RMSE_Train': 9203.779062828593, 'MAE_Train': 5957.7934678080655, 'R2_Train': 0.7731257733558766, 'Adj_R2_Train': 0.09630742882921728, 'RMSE_Test': 15657.634402521715, 'MAE_Test':
9053.840946406206, 'R2_Test': 0.09843519472164775, 'Adj_R2_Test': 0.09630742882921728}
{'Model': 'LinearRegression', 'RMSE_Train': 9894.32389290344, 'MAE_Train': 6535.97403218884, 'R2_Train': 0.7032038544402803, 'Adj_R2_Train': 0.7674619610404199, 'RMSE_Test': 10052.72496502322, 'MAE_Test':
8148.525191407744, 'R2_Test': 0.7680094774456392, 'Adj_R2_Test': 0.7674619610404199}
{'Model': 'XGBRegressor', 'RMSE_Train': 9876.044631898656, 'MAE_Train': 6743.336878927768, 'R2_Train': 0.704299473072675, 'Adj_R2_Train': 0.868750016618244, 'RMSE_Test': 7552.425092141192, 'MAE_Test': 585
0.035633959257, 'R2_Test': 0.8690590478606498, 'Adj_R2_Test': 0.868750016618244}
{'Model': 'LinearRegression', 'RMSE_Train': 8243.017330423001, 'MAE_Train': 5863.316046815271, 'R2_Train': 0.758863449242825, 'Adj_R2_Train': -4.665781084192161e+19, 'RMSE_Test': 142944147133700.72, 'MAE_T
est': 20305804632452.098, 'R2_Test': -4.654795390418416e+19, 'Adj_R2_Test': -4.665781084192161e+19}
{'Model': 'XGBRegressor', 'RMSE_Train': 8209.356334441856, 'MAE_Train': 5949.823731142474, 'R2_Train': 0.7608288274909547, 'Adj_R2_Train': 0.34435226052687196, 'RMSE_Test': 16944.918442261744, 'MAE_Test':
10687.825740961323, 'R2_Test': 0.3458959988976462, 'Adj_R2_Test': 0.34435226052687196}
Result_mean:                Model   RMSE_Train    MAE_Train  R2_Train  Adj_R2_Train  \
0  LinearRegression  9459.487001  6240.303214  0.742459 -9.331562e+18
1      XGBRegressor  9466.809682  6362.441762  0.742250  5.778919e-01

      RMSE_Test      MAE_Test       R2_Test   Adj_R2_Test
0  2.858883e+13  4.061161e+12 -9.309591e+18 -9.331562e+18
1  1.101572e+04  7.386483e+03  5.788858e-01  5.778919e-01
```

Out[15]:

	Model	RMSE_Train	MAE_Train	R2_Train	Adj_R2_Train	RMSE_Test	MAE_Test	R2_Test	Adj_R2_Test
0	LinearRegression	10173.094867	6696.569852	0.729651	8.536974e-01	6.157729e+03	4.441731e+03	8.540418e-01	8.536974e-01
1	XGBRegressor	9965.527271	6511.408327	0.740571	8.319024e-01	6.600476e+03	4.684788e+03	8.322982e-01	8.319024e-01
2	LinearRegression	9936.525631	6434.446275	0.739956	7.303165e-01	8.612753e+03	6.342802e+03	7.309515e-01	7.303165e-01
3	XGBRegressor	10079.341107	6649.846404	0.732427	7.481474e-01	8.323156e+03	6.655923e+03	7.487404e-01	7.481474e-01
4	LinearRegression	9050.473283	5671.209864	0.780621	1.849564e-01	1.486984e+04	8.689317e+03	1.868755e-01	1.849564e-01
5	XGBRegressor	9203.779063	5957.793468	0.773126	9.630743e-02	1.565763e+04	9.053841e+03	9.843519e-02	9.630743e-02
6	LinearRegression	9894.323893	6535.974032	0.703204	7.674620e-01	1.005272e+04	8.148525e+03	7.680095e-01	7.674620e-01
7	XGBRegressor	9876.044632	6743.336879	0.704299	8.687500e-01	7.552425e+03	5.850036e+03	8.690590e-01	8.687500e-01
8	LinearRegression	8243.017330	5863.316047	0.758863	-4.665781e+19	1.429441e+14	2.030580e+13	-4.654795e+19	-4.665781e+19
9	XGBRegressor	8209.356334	5949.823731	0.760829	3.443523e-01	1.694492e+04	1.068783e+04	3.458960e-01	3.443523e-01

GridSearch for XGBRegressor

In [40]:

```
df = pd.read_csv('df_nans_handeled_cat_power_with_date.csv')
df = df.sort_values(by='Date')
df
```

Out[40]:

	Store	Date	CW	Month	Year	Open	Promo	IsPromo	IsStateHoliday	SchoolHoliday	IsSchoolHoliday	NumStateHoliday	CompetitionDistance	IsCompetition	Promo2	Promo2Member	Promo2Active	StateHolida
0	1	2013-04-21	16	4	2013	6	0	0	0	0	0	0	1270.0	1	0	0	0	
1	1	2013-04-28	17	4	2013	6	5	1	0	0	0	0	1270.0	1	0	0	0	
2	1	2013-05-05	18	5	2013	5	5	1	1	0	0	1	1270.0	1	0	0	0	
3	1	2013-05-12	19	5	2013	5	0	0	1	0	0	1	1270.0	1	0	0	0	
4	1	2013-05-19	20	5	2013	6	5	1	0	0	0	0	1270.0	1	0	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
124875	1115	2015-05-10	19	5	2015	6	5	1	0	0	0	0	5350.0	0	1	1	0	
124876	1115	2015-05-17	20	5	2015	5	0	0	1	0	0	1	5350.0	0	1	1	0	
124877	1115	2015-05-24	21	5	2015	6	5	1	0	0	0	0	5350.0	0	1	1	0	
124878	1115	2015-05-31	22	5	2015	5	0	0	1	0	0	1	5350.0	0	1	1	0	
124879	1115	2015-06-07	23	6	2015	5	5	1	1	0	0	1	5350.0	0	1	1	1	

124880 rows × 193 columns

In [42]:

```
#Find best hyperparameters for XGBRegressor

df_to_use = df[['Store', 'Future_Sales', 'Open', 'Promo', 'IsPromo', 'StateHoliday_b', 'Sales_Lag_1',
       'Sales_Lag_1_MA_6', 'Sales_Lag_1_MA_8', 'Sales_Lag_2_MA_8',
       'Sales_Lag_3_MA_8', 'Sales_Lag_5_MA_8', 'Sales_Lag_8_MA_8',
       'SalesPerCustomer_Lag_3_MA_6', 'SalesPerCustomer_Lag_8',
       'SalesPerOpenDay_Lag_1', 'SalesPerOpenDay_Lag_4_MA_6',
       'Customers_Lag_1_MA_6', 'Customers_Lag_3_MA_4', 'Customers_Lag_6',
       'Customers_Lag_8_MA_8', 'CustomersPerOpenDay_Lag_6']]

X_train = df_to_use.drop(columns=['Future_Sales'])
y_train = df_to_use['Future_Sales']

from xgboost import XGBRegressor
from sklearn.model_selection import GridSearchCV, TimeSeriesSplit
from sklearn.metrics import mean_squared_error, make_scorer

# Define your XGBRegressor model
model = XGBRegressor(random_state=42)

# Define the parameter grid to search
param_grid = {
```

```
    'max_depth': [3, 5, 7],
    'learning_rate': [0.01, 0.1, 0.2],
    'n_estimators': [100, 500, 1000],
    'subsample': [0.7, 0.8, 0.9],
    'colsample_bytree': [0.7, 0.8, 0.9],
}

# Define the TimeSeriesSplit cross-validator
tscv = TimeSeriesSplit(n_splits=5)

# Define your scoring function, e.g., negative mean squared error
scorer = make_scorer(mean_squared_error, greater_is_better=False)

# Set up the GridSearchCV object
grid_search = GridSearchCV(estimator=model, param_grid=param_grid, cv=tscv, scoring=scorer, n_jobs=-1)

# Assume X_train, y_train are your features and targets
# Fit the GridSearchCV object to the data
grid_search.fit(X_train, y_train)

# Print the best parameters and the corresponding score
print("Best parameters found: ", grid_search.best_params_)
print("Best score found: ", grid_search.best_score_)
```

```
Best parameters found:  {'colsample_bytree': 0.7, 'learning_rate': 0.01, 'max_depth': 5, 'n_estimators': 500, 'subsample': 0.8}
Best score found:  -135745424.80994418
```

Model RMSE_Train MAE_Train R2_Train Adj_R2_Train RMSE_Test MAE_Test R2_Test Adj_R2_Test 1 XGBRegressor 9457.068947 6669.648039 0.730577 0.730529 6298.190215 4920.265378 0.847307 0.846947 2 XGBRegressor_grid 9234.867751 6474.754589 0.743089 0.743043 6288.752980 4914.484670 0.847764 0.847405

Result: -> new parameter have no significant impact. MAE just better by 10.

Prophet

In [27]:
```
df = pd.read_csv('weekly_sales_with_store_info.csv')
df['Date'] = pd.to_datetime(df['Date'])
```

In [31]:
```
# Cross validation for a single store

df_store = df[df['Store'] == 836]
df_store = df_store[['Date', 'Sales', 'IsPromo', 'NumStateHoliday', 'Open', 'SchoolHoliday', 'IsSchoolHoliday']]

amount_test_weeks = 8
df_prophet = df_store.rename(columns={'Date': 'ds', 'Sales': 'y'})
df_train = df_prophet[: -amount_test_weeks]
df_test = df_prophet[-amount_test_weeks:]#.drop(columns=['y'])


model = Prophet()
model.add_regressor('IsPromo')
model.add_regressor('NumStateHoliday')
model.add_regressor('SchoolHoliday')
model.add_regressor('IsSchoolHoliday')
model.add_regressor('Open')
model.fit(df_train)

df_cv = cross_validation(model,
                         initial='547 days',  # Initial Training Period
                         period='90 days',  # Cross-Validation all x days
                         horizon='56 days')  # prediction horizon

mae_per_cutoff = []
r2_per_cutoff = []

# Interate through each unique cutoff
for cutoff in df_cv['cutoff'].unique():
    # Filter the rows that belong to the current cutoff
    df_cutoff = df_cv[df_cv['cutoff'] == cutoff]

    y_true = df_cutoff['y']
    y_pred = df_cutoff['yhat']

    mae = mean_absolute_error(y_true, y_pred)
    r2 = r2_score(y_true, y_pred)

    mae_per_cutoff.append(mae)
    r2_per_cutoff.append(r2)

# Calculate the average of the metrics over all cutoffs
average_mae = np.mean(mae_per_cutoff)
average_r2 = np.mean(r2_per_cutoff)

print("Average MAE over all Cutoffs:", average_mae)
print("Average R2 over all Cutoffs:", average_r2)

df_cv
```

```
  0%|          | 0/4 [00:00<?, ?it/s]
Average MAE over all Cutoffs: 1196.5055662878392
Average R2 over all Cutoffs: 0.9362410073904671
```

Out[31]:

	ds	yhat	yhat_lower	yhat_upper	y	cutoff
0	2014-07-20	41510.880774	39428.990521	43419.120245	41804	2014-07-16
1	2014-07-27	28559.650771	26493.596957	30489.643312	27037	2014-07-16
2	2014-08-03	39965.288377	37954.764939	41785.835830	39041	2014-07-16
3	2014-08-10	39507.367579	37445.086636	41539.840147	36025	2014-07-16
4	2014-08-17	27586.299353	25585.754464	29709.842247	28112	2014-07-16
5	2014-08-24	38507.192156	36522.362460	40325.411918	35854	2014-07-16
6	2014-08-31	26838.719863	24784.316692	28642.596515	27464	2014-07-16
7	2014-09-07	40035.225472	38059.645555	41979.898673	38687	2014-07-16
8	2014-10-19	29168.728397	27269.860656	31109.415818	29806	2014-10-14
9	2014-10-26	38826.338725	36924.284493	40669.153442	34861	2014-10-14
10	2014-11-02	24287.875560	22394.200759	26092.891231	25810	2014-10-14
11	2014-11-09	41794.725439	39899.858958	43584.724828	43156	2014-10-14
12	2014-11-16	40899.518062	39079.952634	42750.461087	37194	2014-10-14
13	2014-11-23	25111.325175	23134.903592	26970.809806	27333	2014-10-14
14	2014-11-30	41919.045833	40099.866640	43854.808928	42914	2014-10-14
15	2014-12-07	45511.530176	43467.393829	47337.069461	45342	2014-10-14
16	2015-01-18	38348.938521	36351.076600	40326.858734	38751	2015-01-12
17	2015-01-25	28220.641807	26210.717029	30284.976245	28817	2015-01-12
18	2015-02-01	40614.865698	38913.614752	42650.812832	39217	2015-01-12
19	2015-02-08	41479.163769	39518.285212	43391.184415	41129	2015-01-12
20	2015-02-15	28026.319013	26130.548972	30001.689547	28889	2015-01-12
21	2015-02-22	39234.584051	37252.473265	41098.036939	39304	2015-01-12
22	2015-03-01	30753.029483	28758.017480	32755.515708	30781	2015-01-12
23	2015-03-08	40968.682196	38932.387362	42834.354996	41084	2015-01-12
24	2015-04-19	41021.543890	39242.094605	42929.174055	42736	2015-04-12
25	2015-04-26	29058.375626	27062.171880	31024.025318	29654	2015-04-12
26	2015-05-03	36616.985950	34669.428040	38447.645960	37657	2015-04-12
27	2015-05-10	40787.162874	38914.735394	42787.936159	42428	2015-04-12
28	2015-05-17	27212.524365	25290.159615	28901.178130	26947	2015-04-12
29	2015-05-24	41334.515380	39459.941021	43234.302703	41466	2015-04-12
30	2015-05-31	26281.450515	24446.645561	28077.819615	28411	2015-04-12
31	2015-06-07	41532.657330	39728.173268	43502.207752	42529	2015-04-12

In [29]:

```python
# Cross validation for all stores
from prophet.diagnostics import cross_validation

MAE_all_stores = []
R2_all_stores = []

for store_id in df['Store'].unique():

    df_store = df[df['Store'] == store_id]
    df_store = df_store[['Date', 'Sales', 'IsPromo', 'NumStateHoliday', 'Open', 'SchoolHoliday', 'IsSchoolHoliday']]

    amount_test_weeks = 8
    df_prophet = df_store.rename(columns={'Date': 'ds', 'Sales': 'y'})
    df_train = df_prophet[: -amount_test_weeks]
    df_test = df_prophet[-amount_test_weeks:]#.drop(columns=['y'])


    model = Prophet()
    model.add_regressor('IsPromo')
    model.add_regressor('NumStateHoliday')
    model.add_regressor('SchoolHoliday')
    model.add_regressor('IsSchoolHoliday')
    model.add_regressor('Open')
    model.fit(df_train)

    df_cv = cross_validation(model,
                             initial='547 days',  # Initial Training Period
                             period='90 days',  # Cross-Validation all x days
                             horizon='56 days',  # prediction horizon
                             parallel="processes")

    mae_per_cutoff = []
    r2_per_cutoff = []

    # Interate through each unique cutoff
    for cutoff in df_cv['cutoff'].unique():
        # Filter the rows that belong to the current cutoff
        df_cutoff = df_cv[df_cv['cutoff'] == cutoff]

        y_true = df_cutoff['y']
        y_pred = df_cutoff['yhat']

        mae = mean_absolute_error(y_true, y_pred)
        r2 = r2_score(y_true, y_pred)

        mae_per_cutoff.append(mae)
        r2_per_cutoff.append(r2)

    # Calculate the average of the metrics over all cutoffs
    average_mae = np.mean(mae_per_cutoff)
    average_r2 = np.mean(r2_per_cutoff)
    MAE_all_stores.append(average_mae)
    R2_all_stores.append(average_r2)

print("Average MAE over all Cutoffs and stores:", np.mean(MAE_all_stores))
print("Average R2 over all Cutoffs:", np.mean(R2_all_stores))
```

```
Average MAE over all Cutoffs and stores: 2549.5879159821643
Average R2 over all Cutoffs: 0.5843135134596111
```

Result: -> Also with cross validationvthe model is stable.

Comparing Prophet with the simple mean forecast for all stores

In [21]:

```python
df = pd.read_csv('weekly_sales_with_store_info.csv')
df['Date'] = pd.to_datetime(df['Date'])
df = df[['Store', 'Date', 'Sales', 'IsPromo', 'NumStateHoliday', 'Open', 'SchoolHoliday', 'IsSchoolHoliday']]

amount_test_weeks = 8
rolling_mean_weeks = 4
```

```python
results_R2_model_train = []
results_MAE_model_train = []
results_R2_model_test = []
results_MAE_model_test = []
results_R2_mean = []
results_MAE_mean = []

for store_id in df['Store'].unique():
    df_store = df[df['Store'] == store_id].sort_values(by='Date')

    df_store = df_store.rename(columns={'Date': 'ds', 'Sales': 'y'}).drop(columns=['Store'])
    train_df = df_store[:-amount_test_weeks]
    test_df = df_store[-amount_test_weeks:]

    # Prophet-Model
    model = Prophet()
    model.add_regressor('IsPromo')
    model.add_regressor('NumStateHoliday')
    model.add_regressor('SchoolHoliday')
    model.add_regressor('IsSchoolHoliday')
    model.add_regressor('Open')
    model.fit(train_df)

    # Predictions and performance evaluation for the test set
    forecast_test = model.predict(test_df.drop(columns=['y']))
    y_test = test_df['y'].reset_index(drop=True)
    y_test_pred = forecast_test['yhat'].reset_index(drop=True)
    results_MAE_model_test.append(mean_absolute_error(y_test, y_test_pred))
    results_R2_model_test.append(r2_score(y_test, y_test_pred))

    # Predictions and performance evaluation for the training set
    forecast_train = model.predict(train_df.drop(columns=['y']))
    y_train = train_df['y'].reset_index(drop=True)
    y_train_pred = forecast_train['yhat'].reset_index(drop=True)
    results_MAE_model_train.append(mean_absolute_error(y_train, y_train_pred))
    results_R2_model_train.append(r2_score(y_train, y_train_pred))

    # Calculation of the rolling averages for the test data
    rolling_means = train_df['y'].rolling(window=rolling_mean_weeks, min_periods=1).mean().iloc[-amount_test_weeks:].reset_index(drop=True)
    results_MAE_mean.append(mean_absolute_error(y_test, rolling_means))
    results_R2_mean.append(r2_score(y_test, rolling_means))

# Average results for the model and the rolling means
print("Prophet Model Training: MAE:", np.mean(results_MAE_model_train), "R2:", np.mean(results_R2_model_train))
print("Prophet Model Test: MAE:", np.mean(results_MAE_model_test), "R2:", np.mean(results_R2_model_test))
print("rolling mean: MAE:", np.mean(results_MAE_mean), "R2:", np.mean(results_R2_mean))
```

```
Prophet Model Training: MAE: 1955.7192048128795 R2: 0.9000108660360868
Prophet Model Test: MAE: 2481.3612197359307 R2: 0.6839116472416753
rolling mean: MAE: 5080.371356502242 R2: 0.1248924864258999
```

Test if better performance when adding holidays df

In [23]:

```python
df = pd.read_csv('weekly_sales_with_store_info.csv')
df['Date'] = pd.to_datetime(df['Date'])
df = df[['Store', 'Date', 'Sales', 'IsPromo', 'IsStateHoliday', 'NumStateHoliday', 'Open', 'IsSchoolHoliday', 'SchoolHoliday']]

amount_test_weeks = 8
rolling_mean_weeks = 4

results_R2_model_train = []
results_MAE_model_train = []
results_R2_model_test = []
results_MAE_model_test = []
results_R2_mean = []
results_MAE_mean = []

for store_id in df['Store'].unique():
    df_store = df[df['Store'] == store_id].sort_values(by='Date')


    # prepare holiday information
    listOfStateholidays = df_store[df_store['IsStateHoliday'] == 1]['Date'].to_list()
    stateHolidays = pd.DataFrame({'holiday': 'stateHoliday', 'ds': listOfStateholidays})
    listOfSchoolholidays = df_store[df_store['IsSchoolHoliday'] == 1]['Date'].to_list()
    schoolHolidays = pd.DataFrame({'holiday': 'schoolHoliday', 'ds': listOfSchoolholidays})
    holidays_df = pd.concat((stateHolidays, schoolHolidays))



    df_store = df_store.rename(columns={'Date': 'ds', 'Sales': 'y'}).drop(columns=['Store'])
    train_df = df_store[:-amount_test_weeks]
    test_df = df_store[-amount_test_weeks:]

    # Prophet-Model
    model = Prophet(holidays=holidays_df)
    model.add_regressor('IsPromo')
    model.add_regressor('NumStateHoliday')
    model.add_regressor('SchoolHoliday')
    model.add_regressor('IsSchoolHoliday')
    model.add_regressor('Open')
    model.fit(train_df)

    # Predictions and performance evaluation for the test set
    forecast_test = model.predict(test_df.drop(columns=['y']))
    y_test = test_df['y'].reset_index(drop=True)
    y_test_pred = forecast_test['yhat'].reset_index(drop=True)
    results_MAE_model_test.append(mean_absolute_error(y_test, y_test_pred))
    results_R2_model_test.append(r2_score(y_test, y_test_pred))

    # Predictions and performance evaluation for the training set
    forecast_train = model.predict(train_df.drop(columns=['y']))
    y_train = train_df['y'].reset_index(drop=True)
    y_train_pred = forecast_train['yhat'].reset_index(drop=True)
    results_MAE_model_train.append(mean_absolute_error(y_train, y_train_pred))
    results_R2_model_train.append(r2_score(y_train, y_train_pred))

    # Calculation of the rolling averages for the test data
    rolling_means = train_df['y'].rolling(window=rolling_mean_weeks, min_periods=1).mean().iloc[-amount_test_weeks:].reset_index(drop=True)
    results_MAE_mean.append(mean_absolute_error(y_test, rolling_means))
    results_R2_mean.append(r2_score(y_test, rolling_means))

# Average results for the model and the rolling means
print("Prophet Model Training: MAE:", np.mean(results_MAE_model_train), "R2:", np.mean(results_R2_model_train))
print("Prophet Model Test: MAE:", np.mean(results_MAE_model_test), "R2:", np.mean(results_R2_model_test))
print("rolling mean: MAE:", np.mean(results_MAE_mean), "R2:", np.mean(results_R2_mean))
```

```
Prophet Model Training: MAE: 1910.8849820882488 R2: 0.9056173164988949
Prophet Model Test: MAE: 2578.5673379086993 R2: 0.6616483114857065
rolling mean: MAE: 5080.371356502242 R2: 0.1248924864258999
```

Prophet Modell Training: MAE: 1972.5419565565041 R2: 0.899967279726696
Prophet Modell Test: MAE: 2595.84282407716 R2: 0.6582060611672573
Rollierendes Mittel: MAE: 5080.371356502242 R2: 0.1248924864258999

IsSchoolHoliday+IsPromo+Open+NumStateHoliday:
Prophet Model Training: MAE: 1942.732257379376 R2: 0.9023055976075106
Prophet Model Test: MAE: 2587.272702955574 R2: 0.6604024093107642
rolling mean: MAE: 5080.371356502242 R2: 0.1248924864258999

SchoolHoliday+IsPromo+Open+NumStateHoliday:
Prophet Model Training: MAE: 1927.147668566777 R2: 0.9040004968090126
Prophet Model Test: MAE: 2576.9241189380964 R2: 0.6625738314245126
rolling mean: MAE: 5080.371356502242 R2: 0.1248924864258999

IsSchoolHoliday+SchoolHoliday+IsPromo+Open+NumStateHoliday:
Prophet Model Training: MAE: 1955.7192048128795 R2: 0.9000108660360868
Prophet Model Test: MAE: 2481.3612197359307 R2: 0.6839116472416753
rolling mean: MAE: 5080.371356502242 R2: 0.1248924864258999

IsSchoolHoliday+SchoolHoliday+IsPromo+Open+NumStateHoliday+IsStateHoliday:
Prophet Model Training: MAE: 1910.888730056242 R2: 0.9056072452971143
Prophet Model Test: MAE: 2578.91497850813 R2: 0.6619103771567493
rolling mean: MAE: 5080.371356502242 R2: 0.1248924864258999

IsSchoolHoliday+SchoolHoliday+IsPromo+Open+NumStateHoliday+Promo2Active:
Prophet Model Training: MAE: 1894.3922079071583 R2: 0.9070264454671425
Prophet Model Test: MAE: 2612.276241385625 R2: 0.6552970589931474
rolling mean: MAE: 5080.371356502242 R2: 0.1248924864258999

Without all:
Prophet Model Training: MAE: 6950.83533447483 R2: 0.26128739713834326
Prophet Model Test: MAE: 9173.198195088955 R2: -5.064410667165887
rolling mean: MAE: 5080.371356502242 R2: 0.1248924864258999

With school and state holidays df + IsPromo+Open:
Prophet Model Training: MAE: 1955.7147462874266 R2: 0.901314501507588
Prophet Model Test: MAE: 2613.7823949554468 R2: 0.6587847319848437
rolling mean: MAE: 5080.371356502242 R2: 0.1248924864258999

With school and state holidays df + IsPromo+Open+NumStateHoliday:
Prophet Model Training: MAE: 1942.7642797501753 R2: 0.9022927995739541
Prophet Model Test: MAE: 2588.1424252794377 R2: 0.6595419524091702
rolling mean: MAE: 5080.371356502242 R2: 0.1248924864258999

With school and state holidays df + IsPromo+Open+NumStateHoliday+SchoolHoliday:
Prophet Model Training: MAE: 1910.9879975952124 R2: 0.9056030976194543
Prophet Model Test: MAE: 2579.6846059423483 R2: 0.6618115583295695
rolling mean: MAE: 5080.371356502242 R2: 0.1248924864258999

With school and state holidays df + IsPromo+Open+NumStateHoliday+SchoolHoliday+IsSchoolHoliday:
Prophet Model Training: MAE: 1910.8849820882488 R2: 0.9056173164988949
Prophet Model Test: MAE: 2578.5673379086993 R2: 0.6616483114857065
rolling mean: MAE: 5080.371356502242 R2: 0.1248924864258999

Result: -> no advantage in adding the holidays df

Prophet with simple splitting and just one modell

In [24]:
```
# Prophet with simple splitting and just one modell

amount_test_weeks = 8
rolling_mean_weeks = 4

df = pd.read_csv('weekly_sales_with_store_info.csv')
df['Date'] = pd.to_datetime(df['Date'])
df = df[['Store', 'Date', 'Sales', 'IsPromo', 'IsStateHoliday', 'NumStateHoliday', 'Open']]
df = df.rename(columns={'Date': 'ds', 'Sales': 'y'})

train_data = []
test_data = []
# Group by store and split into training and test data
amount_test_weeks = 8
for store_id, group in df.groupby('Store'):
        train_data.append(group[: -amount_test_weeks])
        test_data.append(group[-amount_test_weeks:])
# Combine the list entries to one dataframe
train_df = pd.concat(train_data).drop(columns=['Store'])
test_df = pd.concat(test_data).drop(columns=['Store'])

# Prophet-Model
model = Prophet()
model.add_regressor('IsPromo')
model.add_regressor('IsStateHoliday')
model.add_regressor('NumStateHoliday')
model.add_regressor('Open')
model.fit(train_df)
forecast = model.predict(test_df)

y_test = test_df['y'].reset_index(drop=True)
y_test_pred = forecast['yhat'].reset_index(drop=True)

y_train = train_df['y'].reset_index(drop=True)
y_train_pred = model.predict(train_df)['yhat'].reset_index(drop=True)

results_MAE_model_test = mean_absolute_error(y_test, y_test_pred)
results_R2_model_test = r2_score(y_test, y_test_pred)

print("Prophet Modell Test: MAE:", results_MAE_model_test, "R2:", results_R2_model_test)
print("Prophet Modell Training: MAE:", mean_absolute_error(y_train, y_train_pred), "R2:", r2_score(y_train, y_train_pred))
```

```
Prophet Modell Test: MAE: 12779.320119709748 R2: -0.10523563635582889
Prophet Modell Training: MAE: 15148.117992418662 R2: -0.3289759263369212
```

Result: -> One model for all stores together is realy bad.

Summary of choosing a model

- Prophet is the best model to predict the sales of the next 8 weeks.
- LinearRegression is the second best model with MAE of 4295 which is a difference of 1399 compared to the rolling mean of 5694 (-24,6%).
- Testet on all stores and taking the average MAE it is with 2481 better then from the second best model LinearRegressor with MAE of 4295 which is a difference of 1814. (-42%).
- Simple rolling mean has MAE 5080, so prophet model is 2599 units (-51%) better *(the different rolling means comes from the reduced data set due to the lag features)

Forecast of single store

Prophet

In [25]:
```
df = pd.read_csv('weekly_sales_with_store_info.csv')
df['Date'] = pd.to_datetime(df['Date'])
```

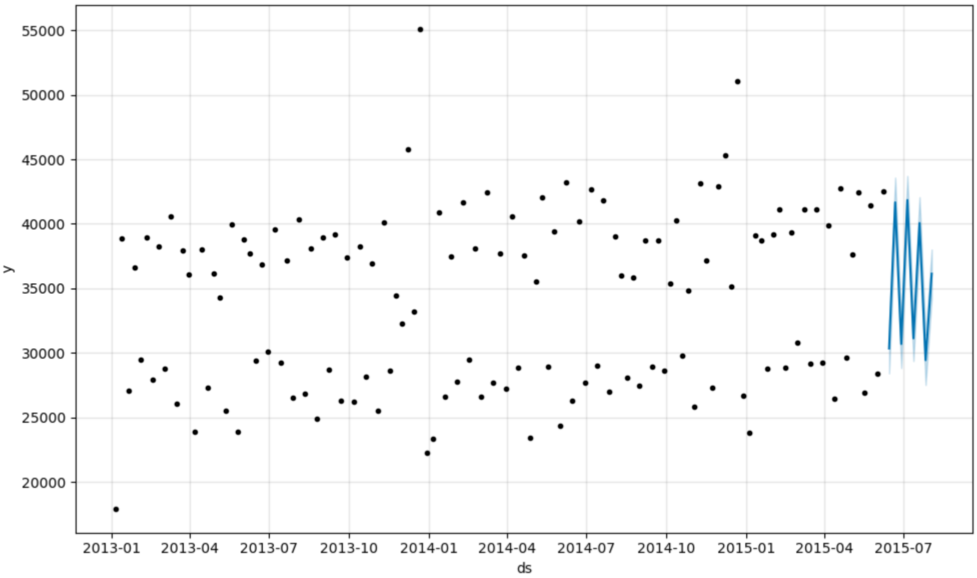
In [26]:
```
df_store = df[df['Store'] == 836]
df_store = df_store[['Date', 'Sales', 'IsPromo', 'NumStateHoliday', 'Open', 'SchoolHoliday', 'IsSchoolHoliday']]

amount_test_weeks = 8
```

```python
df_prophet = df_store.rename(columns={'Date': 'ds', 'Sales': 'y'})
df_train = df_prophet[: -amount_test_weeks]
df_test = df_prophet[-amount_test_weeks:]#.drop(columns=['y'])


model = Prophet()
model.add_regressor('IsPromo')
model.add_regressor('NumStateHoliday')
model.add_regressor('SchoolHoliday')
model.add_regressor('IsSchoolHoliday')
model.add_regressor('Open')
model.fit(df_train)

forecast = model.predict(df_test)
fig = model.plot(forecast)
results = forecast.set_index('ds')[['yhat', 'yhat_lower', 'yhat_upper']].join(df_test.set_index('ds')['y'])

y_train = df_train['y']
y_train_pred = model.predict(df_train)['yhat']

y_true = df_test['y'].reset_index(drop=True)
y_pred = forecast['yhat'].reset_index(drop=True)

mae = mean_absolute_error(y_true, y_pred)
mae_train = mean_absolute_error(y_train, y_train_pred)

print("Model Train:", "MAE:", mae_train, "R2:", r2_score(y_train, y_train_pred))
print("Model Test:", "MAE:", mae, "R2:", r2_score(y_true, y_pred))

results
```

```
Model Train: MAE: 1133.8449415173889 R2: 0.9530844084523857
Model Test: MAE: 1531.1734479919842 R2: 0.8922165820415624
```

Out[26]:

ds	yhat	yhat_lower	yhat_upper	y
2015-06-14	30343.934576	28414.838469	32172.910620	30366
2015-06-21	41668.150834	39701.169808	43582.779472	41807
2015-06-28	30687.725624	28830.243304	32537.053508	28618
2015-07-05	41845.499272	40049.583284	43710.151752	45770
2015-07-12	31124.505141	29372.040093	32898.961510	32638
2015-07-19	40064.597750	38140.562432	42064.344602	41183
2015-07-26	29435.660199	27518.066170	31405.908411	31599
2015-08-02	36136.990267	34281.787697	38011.442342	37436



In [9]:
```python
df[df['Store'] == 836]
```

Out[9]:

	Store	Date	CW	Month	Year	DayOfWeek	Sales	SalesPerCustomer	SalesPerOpenDay	Customers	CustomersPerOpenDay	Open	Promo	IsPromo	StateHoliday	IsStateHoliday	SchoolHoliday	IsSchoolHoliday
112725	836	2013-01-06	1	1	2013	6	17937	7.171931	4484.250000	2501	625.250000	4	0	0	a	1	2	1
112726	836	2013-01-13	2	1	2013	6	38907	8.141243	6484.500000	4779	796.500000	6	5	1	0	0	0	0
112727	836	2013-01-20	3	1	2013	6	27071	7.213163	4511.833333	3753	625.500000	6	0	0	0	0	0	0
112728	836	2013-01-27	4	1	2013	6	36644	8.402660	6107.333333	4361	726.833333	6	5	1	0	0	0	0
112729	836	2013-02-03	5	2	2013	6	29513	7.618224	4918.833333	3874	645.666667	6	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
112855	836	2015-07-05	27	7	2015	6	45770	9.734156	7628.333333	4702	783.666667	6	5	1	0	0	0	0
112856	836	2015-07-12	28	7	2015	6	32638	8.260693	5439.666667	3951	658.500000	6	0	0	0	0	0	0
112857	836	2015-07-19	29	7	2015	6	41183	9.159920	6863.833333	4496	749.333333	6	5	1	0	0	5	1
112858	836	2015-07-26	30	7	2015	6	31599	8.638327	5266.500000	3658	609.666667	6	0	0	0	0	5	1
112859	836	2015-08-02	31	8	2015	6	37436	9.429723	7487.200000	3970	794.000000	5	5	1	0	0	5	1

135 rows × 31 columns