Day-ahead Banknifty/Nifty Forecasting

Input - 15 mins candle data

Output - Forcast of 1 day of 15 mins close price

Special Notes:

- Don't open CSV file and change dateformat, it will change the format entirely and cause problem in code
- Output of Model changed if we dont delete NaN values (I saw that when comparing jupyter file with normal python file matching)

Define stock for which ML model needs to be prepared

In [5]:
```
# Get data from Python file--------------
import json

# Load the arguments
with open('arguments.json', 'r') as fid:
    arguments = json.load(fid)

stock = arguments['ticker']
print(stock)


#stock = "SUNPHARMA"
```

```
FINNIFTY
```

In [6]:

```
from datetime import datetime, timedelta
from pandas.tseries.offsets import DateOffset
import math
import pickle # Saving the final model
import time
import datetime as dt

##############
# Dependencies
##############
import json
import requests
import numpy as np
import pandas as pd
import os, time, math
from datetime import *
os.environ['TZ'] = 'Asia/Calcutta'
from sqlalchemy import create_engine
from scipy.signal import savgol_filter
pd.options.mode.chained_assignment = None




##################
# Visualizations
##################
import seaborn as sns
#import plotly.offline
#import cufflinks as cf
import matplotlib.pyplot as plt
from matplotlib import rc, rcParams



##################
# Display Changes
##################
# cf.go_offline()
%matplotlib inline
rc('axes', linewidth=2)
rc('font', weight='bold')
```

```
sns.set_style('whitegrid')
#pd.set_option('max_rows',2000)
#pd.set_option('max_columns',255)
rcParams['figure.figsize'] = 15,5
#cf.set_config_file(offline=False, world_readable=True)


##############
# Modeling
##############
# import xgboost as xgb
# from xgboost import plot_importance
from sklearn import preprocessing, svm
from sklearn.model_selection import TimeSeriesSplit
from sklearn.preprocessing import StandardScaler, MinMaxScaler, LabelEncoder, RobustScaler
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.metrics import mean_squared_error

import warnings
warnings.filterwarnings("ignore")
warnings.filterwarnings("ignore", category=UserWarning)
warnings.filterwarnings("ignore", category=FutureWarning)
warnings.filterwarnings("ignore", category=DeprecationWarning)
```

In [7]:

```
# ###################
# # Display Changes
# ###################
# from matplotlib import rc, rcParams
# pd.options.mode.chained_assignment = None
# # cf.go_offline()
# %matplotlib inline
# rc('axes', linewidth=2)
# #rc('font', weight='bold')

# #pd.set_option('max_rows', 1000)
# pd.set_option('display.max_rows', None)
# pd.set_option('max_columns', 255)
# # Useful line of code to set the display option so we could see all the columns in pd dataframe
# #pd.set_option('display.max_columns', None)

# #rcParams['figure.figsize'] = 8,3
# # cf.set_config_file(offline=False, world_readable=True)

# rc('xtick', labelsize=9)
# rc('ytick', labelsize=9)
# #rc('axes', linewidth=2)
# rc('axes', titlesize=10)


# # Matplot Theme setting
# from matplotlib import cycler
# colors = cycler('color',
#                 ['#EE6666', '#3388BB', '#9988DD',
#                  '#EECC55', '#88BB44', '#FFBBBB'])
# rc('axes', facecolor='#E6E6E6', edgecolor='none', axisbelow=True, grid=True, prop_cycle=colors)
# rc('grid', color='w', linestyle='solid')
# rc('xtick', direction='out', color='black')
# rc('ytick', direction='out', color='black')
# rc('patch', edgecolor='#E6E6E6')
# rc('lines', linewidth=1)
```

In [8]:
```
# Seaborn
rc = {
    "axes.facecolor": "#FAEEE9",
    "figure.facecolor": "#FAEEE9",
    "axes.edgecolor": "#000000",
    "grid.color": "#EBEBE7",
    "font.family": "calibri",
    "axes.labelcolor": "#000000",
    "xtick.color": "#000000",
    "ytick.color": "#000000",
    "grid.alpha": 0.9
}
sns.set(rc=rc)
sns.set_context("notebook", font_scale=0.9, rc={"lines.linewidth": 1})


#sns.set_style('whitegrid')
```

In [9]:
```
datetime.now()
```

Out[9]:
```
datetime.datetime(2024, 2, 1, 6, 46, 18, 279001)
```

Prevent plot from showing

In [10]:
```
#plt.ioff()
```

Analysis

In [11]:
```python
# I have make the function for Doing quick basic analysis as per given DataFrame

def analysis(df):

    print('################################################################')
    print(f'Shape of DF = {df.shape}\n')
    print('---------------------------------------------------')
    print(f'First 5 rows are as following \n {df.head()} \n')
    print('---------------------------------------------------')
    print(f'Basic info as following \n {df.info()} \n')
    print('---------------------------------------------------')

    if 'datetime' in df.columns:
        print(f"\n We have data from {pd.to_datetime(df['datetime']).min()}")
        print(f" We have data till {pd.to_datetime(df['datetime']).max()}")
    print('---------------------------------------------------')
    print(f'\n Basic description as following \n')
    print(df.describe())
    print('---------------------------------------------------')
    print(f'\n Missing values are as following \n')
    print(df.isnull().sum())
    print('################################################################')
```

In [12]:
```python
# from google.colab import drive
# drive.mount('/content/drive')
# folder_path = '/content/drive/MyDrive/Climate_Connect/IEX/01_Data_Tuesday_Forecast'
```

In [13]:
```
# As its difficult to upload files again and again, so I change code to access drive files.

#df = pd.read_parquet('./Data/historical_load_data.parquet')
#df = pd.read_excel(f'{folder_path}/DAM_multivariate_data.xlsx')


df = pd.read_csv(f'01_Data\data_{stock}.csv')
df.head()
```

Out[13]:

	datetime	open	high	low	close	volume	date
0	2021-01-01 09:15:00	15215.40	15282.45	15198.20	15265.05	0	NaN
1	2021-01-01 10:15:00	15264.65	15266.40	15224.60	15228.65	0	NaN
2	2021-01-01 11:15:00	15227.95	15239.90	15189.65	15214.00	0	NaN
3	2021-01-01 12:15:00	15213.65	15235.75	15205.65	15220.35	0	NaN
4	2021-01-01 13:15:00	15218.05	15238.30	15194.75	15232.25	0	NaN

Only retain "Close" & "Volume"

In [14]:
```
drop_col = ['open', 'high', 'low', 'volume', 'date']
df = df.drop(drop_col, axis=1)
df.head()
```

Out[14]:

	datetime	close
0	2021-01-01 09:15:00	15265.05
1	2021-01-01 10:15:00	15228.65
2	2021-01-01 11:15:00	15214.00
3	2021-01-01 12:15:00	15220.35
4	2021-01-01 13:15:00	15232.25

In [15]:
```
max_year = pd.to_datetime(df['datetime']).max().year
min_year = pd.to_datetime(df['datetime']).min().year


print(f"\n We have data from {pd.to_datetime(df['datetime']).min()}")
print(f" We have data till {pd.to_datetime(df['datetime']).max()}")
```

```
 We have data from 2021-01-01 09:15:00
 We have data till 2024-01-31 15:15:00
```

In [18]:
```
last_entry_df = pd.to_datetime(df['datetime']).max()
print(last_entry_df)
```

```
2024-01-31 15:15:00
```

In [12]:
```
#df.tz_convert('UTC')
#d = pd.to_datetime(df['datetime'],format='%m-%d-%Y %H:%M:%S').min()
#d.month
```

In [13]:
```
# DataType convertion

df['datetime'] = pd.to_datetime(df['datetime'])
df['date'] = df['datetime'].dt.date
```

In [14]:
```python
# Missing Data checking

df = df.sort_values('datetime').copy()
print('Missing Value summary')
print(f'{df.isnull().sum()} \n')

# Missing and Duplicate values checking
print('If there is any missing and duplicate values')
df.date.value_counts()[df.date.value_counts() != 7] # 7 As we have one hour data

# %age missing values
# df.isnull().sum()/df.shape[0]*100
```

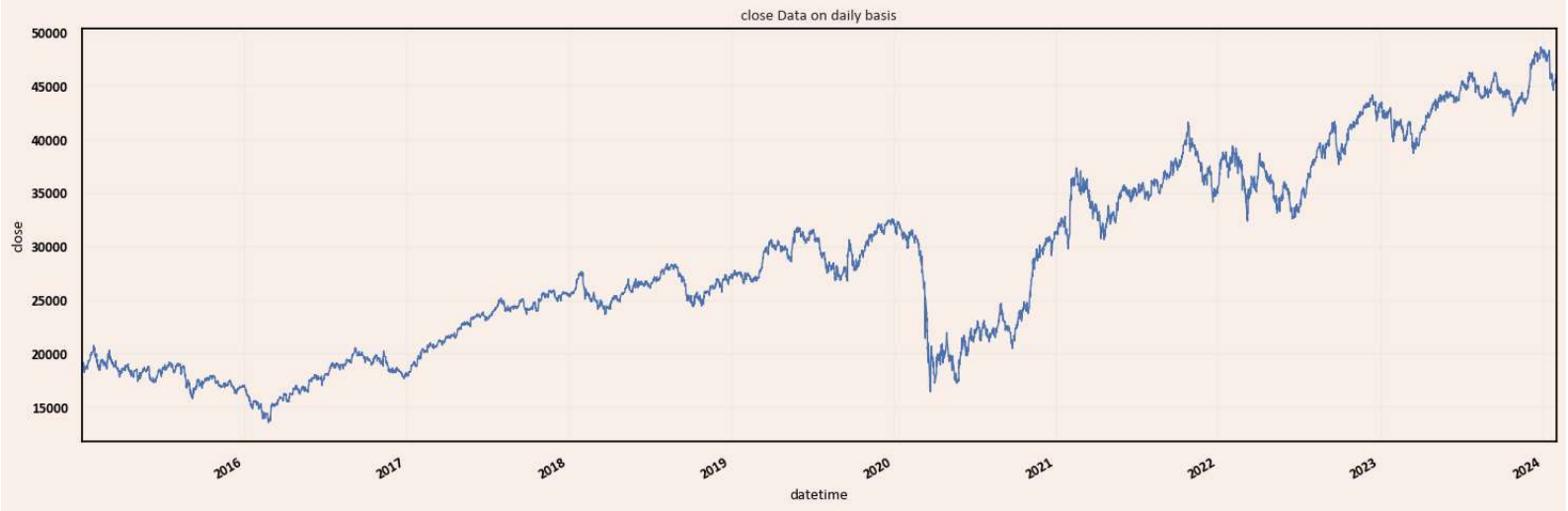
```
Missing Value summary
datetime    0
close       0
date        0
dtype: int64

If there is any missing and duplicate values
```

Out[14]:
```
2017-07-10    4
2020-11-14    3
2021-02-24    3
2019-10-27    3
2015-11-11    2
2021-11-04    2
2017-10-19    2
2016-10-30    2
2018-11-07    2
2022-10-24    1
2023-11-12    1
Name: date, dtype: int64
```

In [15]:
```python
# Drop the Duplicated values
df.drop_duplicates(['datetime'], keep='first', inplace=True)
```

In [16]:
```
## Plot as per original data on daily basis
def original_plot(df, feature):
    #plt.figure(figsize=(12,5))
    #df.set_index('datetime').resample('D')[feature].mean().plot()
    df.set_index('datetime')[feature].plot()
    plt.xlim([df.datetime.min(), df.datetime.max()])
    plt.ylabel(feature)
    plt.title(f"{feature} Data on daily basis")
    plt.tight_layout()
```

In [17]:
```
original_plot(df, 'close')
```



Holiday Data - Step not performed as holiday data is very less

DataFrame Preperation

In [18]:
```
print(pd.to_datetime(df['datetime']).dt.date.min())
print(pd.to_datetime(df['datetime']).dt.date.max())
print(pd.to_datetime(df['datetime']).dt.date.max() + timedelta(days=1)) # today + 1 day(forecast date)
```

```
2015-01-01
2024-01-31
2024-02-01
```

In [19]:

```
"""
# NO NEED to do as below due to the following:
 - We have many muhuraat days
 - We have holidays

"""



# # Prepare FInal Daraframe to avoid any missing time block

# #start_date = '2020-07-01'
# #end_date = '2023-02-17' # today + 1 day(forecast date)

# start_date = pd.to_datetime(df['datetime']).dt.date.min() # Data is available from
# end_date = pd.to_datetime(df['datetime']).dt.date.max() + timedelta(days=1) # Data available till



# z = pd.DataFrame(pd.date_range(start=start_date, end=end_date, freq='1H',closed='left'), columns=['dat
# df = pd.merge(z, df, on='datetime', how='left').copy()

# df['date'] = df['datetime'].dt.date
# # Make column for Time Block here
# df['tb'] = df['datetime'].apply(lambda x : ((x.hour*60 + x.minute)//60+1))

# print(df.shape)
# print(df.tail())
# print(f"\n Close data is missing for {df.close.isna().sum()} records")
```

Out[19]:

```
'\n# NO NEED to do as below due to the following:\n - We have many muhuraat days\n - We have holidays
\n \n'
```

In [20]:

```
# %age missing values
df.isnull().sum()/df.shape[0]*100
```

Out[20]:

```
datetime    0.0
close       0.0
date        0.0
dtype: float64
```

In [21]:
```
df.columns
```

Out[21]:
```
Index(['datetime', 'close', 'date'], dtype='object')
```

In [22]:
```
'''
Missing Values treatment
'''

## Interpolation Technique
df['close'] = df['close'].interpolate()

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

Out[22]:
```
datetime    0
close       0
date        0
dtype: int64
```

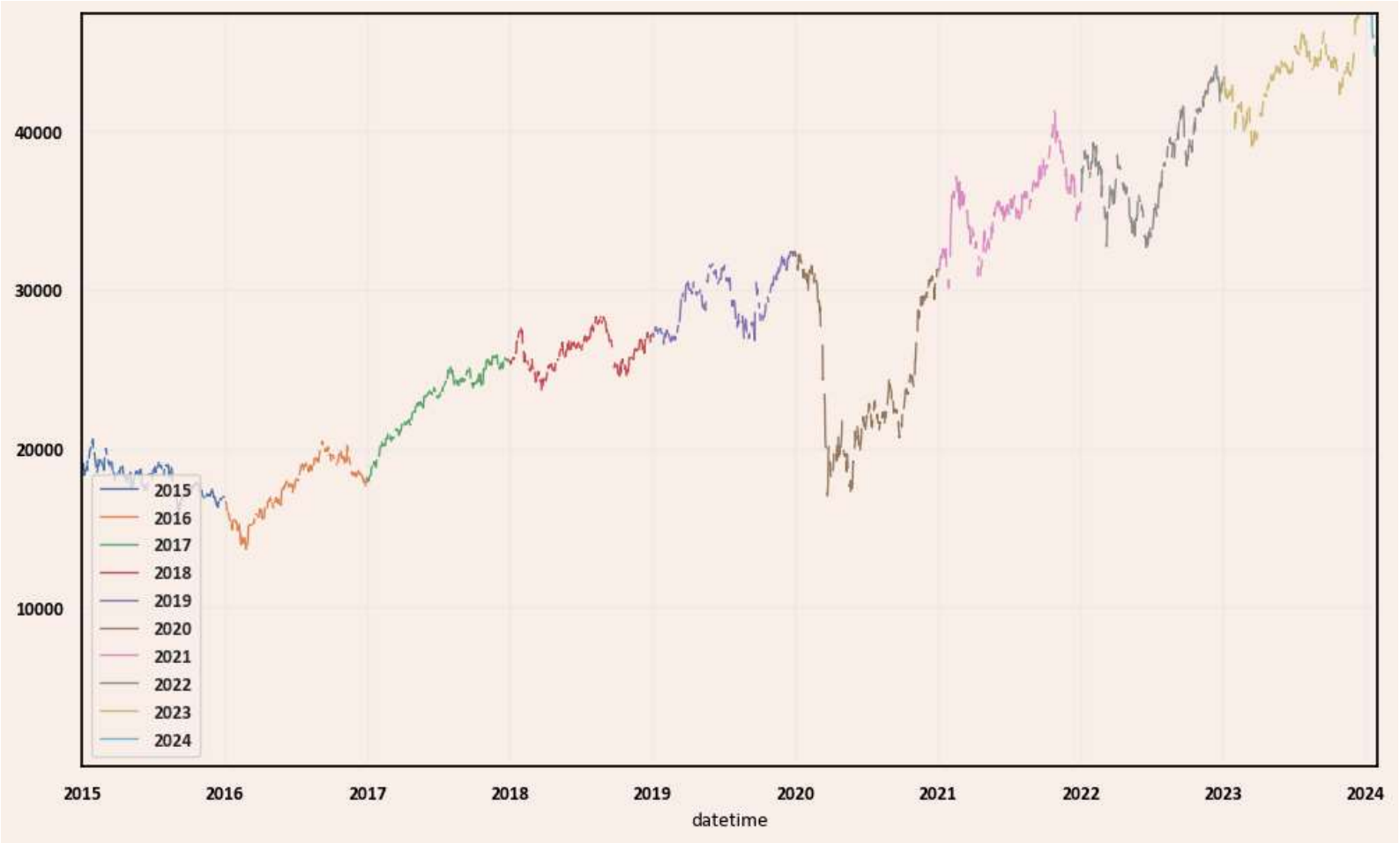
In [23]:
```
# %age missing values
df.isnull().sum()/df.shape[0]*100
```

Out[23]:
```
datetime    0.0
close       0.0
date        0.0
dtype: float64
```

EDA

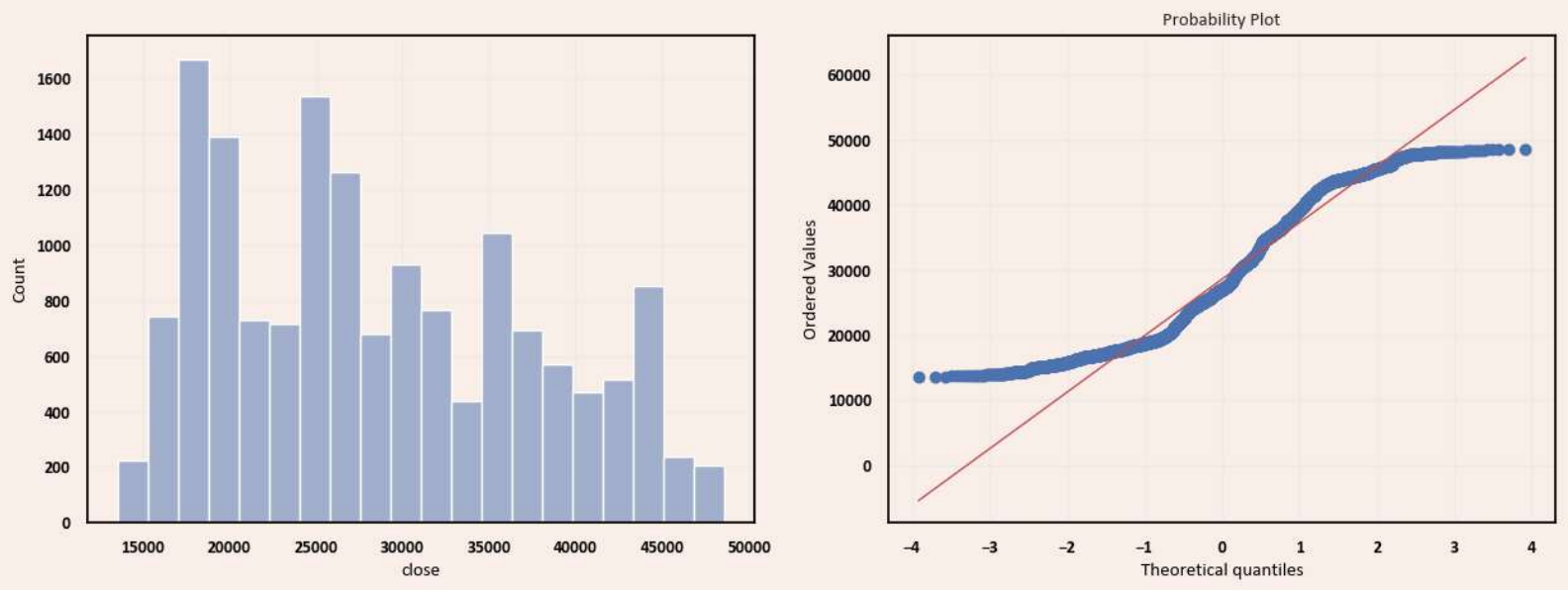
In [24]:

```
for x in [x for x in range(min_year, max_year+1)]:
    df[df.datetime.dt.year==x].set_index('datetime').resample('D')['close'].mean().plot(label=x, figsize
    plt.ylim([10, df.close.quantile(0.99)])
    plt.legend(loc="lower left")
```



In [25]:

```
# To check if mcp_dam is normally distributed or not
import scipy.stats as stat
import pylab

#plt.figure(figsize=(15,5))
plt.subplot(1,2,1)
sns.histplot(data=df,x='close', alpha=0.5, bins=20)

plt.subplot(1,2,2)
stat.probplot(df['close'],dist='norm',plot=pylab)
plt.show()
```



As curve is not following staright line, so close is not normally distributed.

In [26]:

```
## Checking zero or -ve mcp_dam values
df.loc[df['close']<=0, :]
```

Out[26]:

	datetime	close	date

So we have few zero values, we need to replace all with NaN and later will interpolate to get values

In [27]:
```
df.loc[df['close']==0, 'close'] = np.nan
## Interpolation Technique
df['close'] = df['close'].interpolate()
df.isnull().sum()
```

Out[27]:
```
datetime    0
close       0
date        0
dtype: int64
```

Feature Engineering

In [28]:

```python
def gen_datetime_features(data):

    try:
        #Adding time based features
        print('Adding time based features...')
        data['date'] = data.datetime.dt.date
        data['hour'] = data.datetime.dt.hour + 1      # Categorical 24
        data['dom' ] = data.datetime.dt.day       # Categorical 30/31
        data['month'] = data.datetime.dt.month        # Categorical 12
        data['year']=data.datetime.dt.year            # Categorical 12
        data['dow'] = data.datetime.dt.dayofweek      # Categorical 7
        data['doy'] = data.datetime.dt.dayofyear      # Categorical 365/366
        data['woy'] = data.datetime.dt.week # Categorical 52

        data['week_number'] = data['datetime'].apply(lambda time: time.strftime("%V"))
        # Above is Object DataType, so change to int
        data['week_number'] = data['week_number'].astype('int')

        # Friday
        data.loc[data.dow == 4, 'friday'] = 1
        data.loc[data.dow != 4, 'friday'] = 0


        # Covid
        data.loc[data.year.isin([2020,2021,2022]),'covid'] = 1
        data['covid'] = data['covid'].replace(np.nan, 0)
        # Covid 1st wave
        data.loc[(data.year==2020)&(data.month.isin([3,4,5,6,7,8,9])),'covid_first_wave'] = 1
        data['covid_first_wave'] = data['covid_first_wave'].replace(np.nan, 0)
        # Covid 2nd wave
        data.loc[(data.year==2021)&(data.month.isin([3,4,5,6])),'covid_second_wave'] = 1
        data['covid_second_wave'] = data['covid_second_wave'].replace(np.nan, 0)

        print(data.columns)
        return data
    except Exception as e:
        print("Error in adding time based features")



# Creating Cyclic Time Indicators

def cyclic(data,feature):
```

```
    import math
    data['norm'] = 2 * math.pi * data[f"{feature}"] / data[f"{feature}"].max()
    data[f"cos_{feature}"] = np.cos(data["norm"])
    data[f"sin_{feature}"] = np.sin(data["norm"])
    data.drop('norm',1,inplace=True)
    return data


def gen_cyclic_features(data):

    try:
        print('Adding Cyclic Time Indicators........')
        data = cyclic(data,'hour')
        data = cyclic(data,'dow')
        data = cyclic(data,'doy')
        return data
    except Exception as e:
        print("Error in adding Cyclic Time Indicators")




def gen_lag_features(data):
    try:
        # Encoding TS variables
        print('Adding TB Lag features...')
        variable_used = 'close'


        # As we have large data, so better we should drop NaN values
        data['lag1h'] = data[variable_used].shift(1*4)
        data['lag2h'] = data[variable_used].shift(2*4)
        data['lag4h'] = data[variable_used].shift(4*4)
        data['lag6h'] = data[variable_used].shift(6*4)
        data['lag12h'] = data[variable_used].shift(12*4)
        data['lag1d'] = data[variable_used].shift(24*4)
        data['lag2d'] = data[variable_used].shift(2*24*4)
        data['lag3d'] = data[variable_used].shift(3*24*4)
        data['lag4d'] = data[variable_used].shift(4*24*4)
        data['lag5d'] = data[variable_used].shift(5*24*4)
        data['lag6d'] = data[variable_used].shift(6*24*4)
        data['lag7d'] = data[variable_used].shift(7*24*4)
```

```
    # Exponential Moving Averages
        data['mcp_dam_ewm1h'] = data[variable_used].ewm(span = 1*4).mean()
        data['mcp_dam_ewm2h'] = data[variable_used].ewm(span = 2*4).mean()
        data['mcp_dam_ewm3h'] = data[variable_used].ewm(span = 3*4).mean()
        data['mcp_dam_ewm4h'] = data[variable_used].ewm(span = 4*4).mean()
        data['mcp_dam_ewm6h'] = data[variable_used].ewm(span = 6*4).mean()
        data['mcp_dam_ewm8h'] = data[variable_used].ewm(span = 8*4).mean()
        data['mcp_dam_ewm12h'] = data[variable_used].ewm(span = 12*4).mean()
        data['mcp_dam_ewm24h'] = data[variable_used].ewm(span = 24*4).mean()




    # Moving Averages
        #for x in [4,8,12,24,48,96]:
        for x in range(1,30):
            data[f'close_rolling_mean_{x}'] = data[variable_used].rolling(x,1).mean().shift()
            data[f'close_rolling_max_{x}'] = data[variable_used].rolling(x,1).max().shift()
            data[f'close_rolling_min_{x}'] = data[variable_used].rolling(x,1).min().shift()

    # Differences
    # As per analysis these features has least importance , so we will not use
        data['ramp1'] = data[variable_used] - data[variable_used].shift(1)
        data['ramp2'] = data[variable_used].shift(1) - data[variable_used].shift(2)
        data['ramp3'] = data[variable_used].shift(2) - data[variable_used].shift(3)
        data['ramp4'] = data[variable_used].shift(3) - data[variable_used].shift(4)


#     # Taking Last 7 day same TB Mean, These features are super important
#     data['avg_mcp_dam_7d'] = (data['mcp_dam'] + data['mcp_dam'].shift(96) + data['mcp_dam'].shift(2*96
#     data['avg_mcp_dam_9d'] = (data['avg_mcp_dam_7d']*7 + data['mcp_dam'].shift(7*96) + data['mcp_dam']
#     data['avg_mcp_dam_11d'] = (data['avg_mcp_dam_9d']*9 + data['mcp_dam'].shift(9*96) + data['mcp_dam'
#     data['avg_mcp_dam_13d'] = (data['avg_mcp_dam_11d']*11 + data['mcp_dam'].shift(11*96) + data['mcp_d
#     data['avg_mcp_dam_15d'] = (data['avg_mcp_dam_13d']*13 + data['mcp_dam'].shift(13*96) + data['mcp_d
#     data['avg_mcp_dam_17d'] = (data['avg_mcp_dam_15d']*15 + data['mcp_dam'].shift(15*96) + data['mcp_d
#     data['avg_mcp_dam_19d'] = (data['avg_mcp_dam_17d']*17 + data['mcp_dam'].shift(17*96) + data['mcp_d
#     data['avg_mcp_dam_21d'] = (data['avg_mcp_dam_19d']*19 + data['mcp_dam'].shift(19*96) + data['mcp_d
```

```
        print(data.columns)
        return data
    except Exception as e:
        print(f"Error in adding Lag based features--{e}")
```

In [29]:

```
df = gen_datetime_features(df).copy()
```

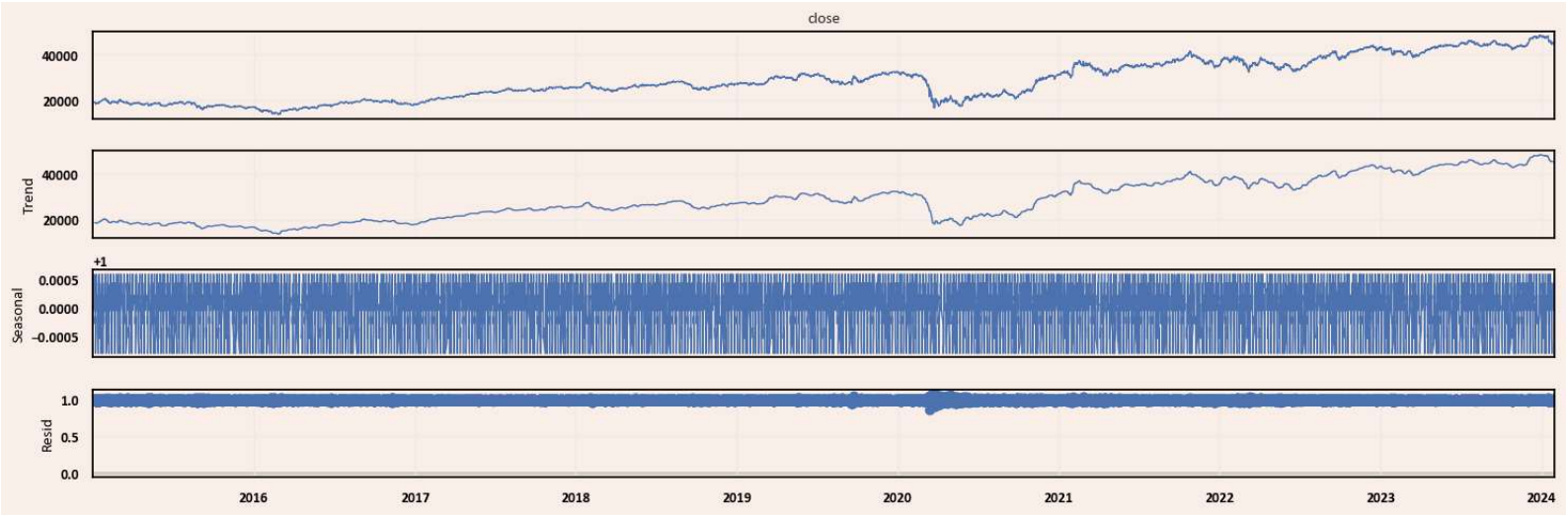
```
Adding time based features...
Index(['datetime', 'close', 'date', 'hour', 'dom', 'month', 'year', 'dow',
       'doy', 'woy', 'week_number', 'friday', 'covid', 'covid_first_wave',
       'covid_second_wave'],
      dtype='object')
```

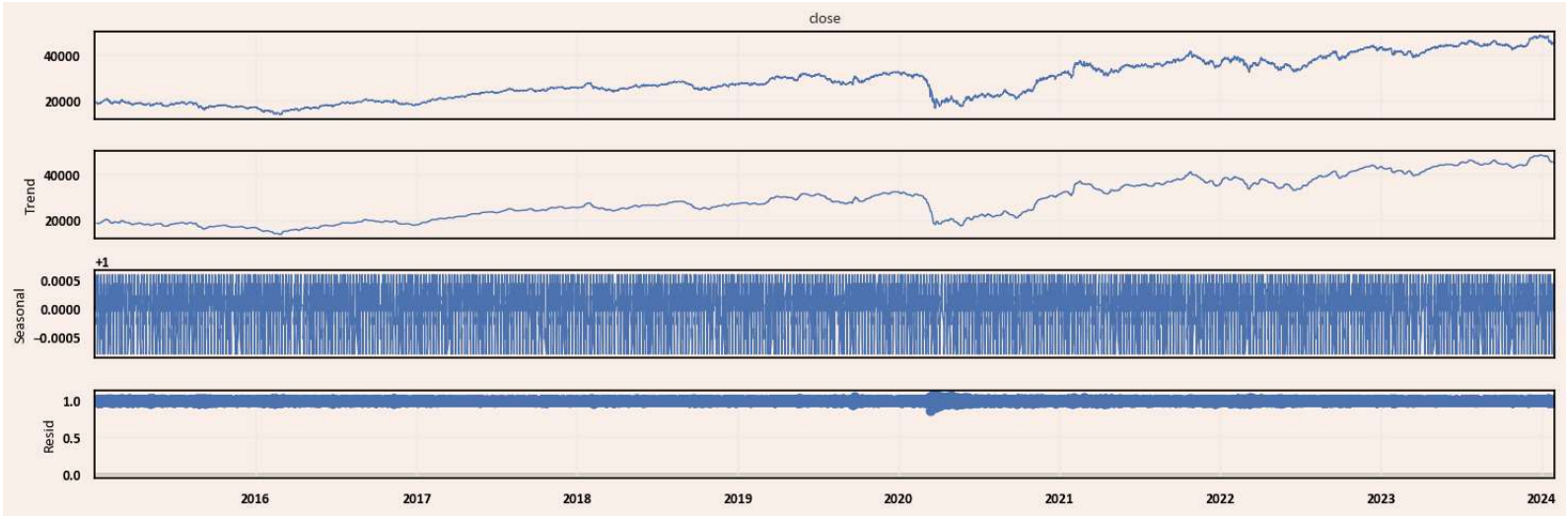
Outliers checking - Not Done

Check the Trend, Seasonality & Residual

In [30]:
```
# Let's try decomposing the load data
from statsmodels.tsa.seasonal import seasonal_decompose
import matplotlib as mpl
from pylab import rcParams
# Extra settings
seed = 42
np.random.seed(seed)
#plt.style.use('bmh')


#plt.figure(num=None, figsize=(50, 20), dpi=80, facecolor='w', edgecolor='k')
#series = df[(df['datetime']>='2020-01-01') & (df['datetime']<'2022-01-01')].set_index(['datetime'])['cl
series = df.set_index(['datetime'])['close'].dropna()

result = seasonal_decompose(series, model='multiplicative',period=30)
result.plot()
```

Out[30]:

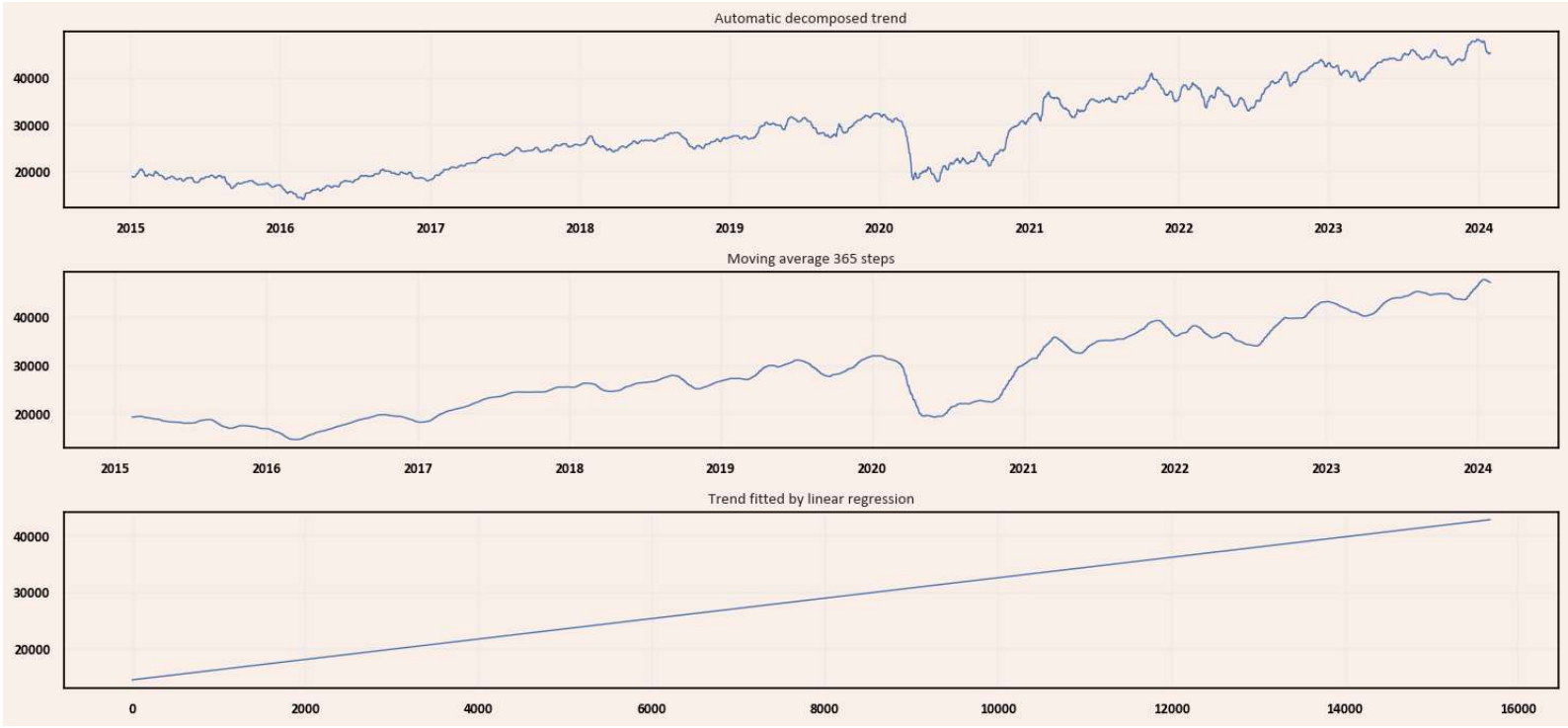




Seasonality is present

In [31]:

```
from sklearn.linear_model import LinearRegression
fig = plt.figure(figsize=(15, 7))
layout = (3, 2)
pm_ax = plt.subplot2grid(layout, (0, 0), colspan=2)
mv_ax = plt.subplot2grid(layout, (1, 0), colspan=2)
fit_ax = plt.subplot2grid(layout, (2, 0), colspan=2)

pm_ax.plot(result.trend)
pm_ax.set_title("Automatic decomposed trend")

mm = series.rolling(200).mean()
mv_ax.plot(mm)
mv_ax.set_title("Moving average 365 steps")

X = [i for i in range(0, len(series))]
X = np.reshape(X, (len(X), 1))
y = series.values
model = LinearRegression()
model.fit(X, y)
# calculate trend
trend = model.predict(X)
fit_ax.plot(trend)
fit_ax.set_title("Trend fitted by linear regression")
plt.tight_layout()
```

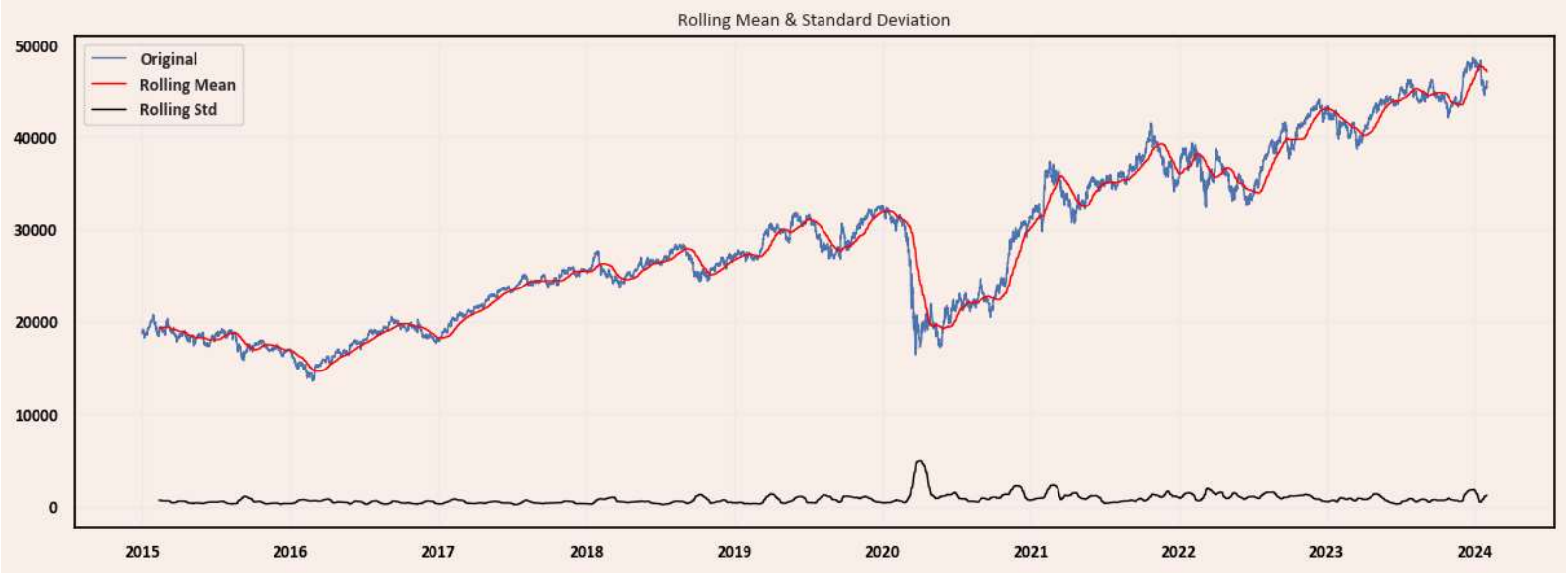


1. Seasonality is present
2. Data is non-stationary

Check for non-stationarity

In [32]:

```
# Determing rolling statistics
rolmean = series.rolling(window=200).mean()
rolstd = series.rolling(window=200).std()

# Plot rolling statistics:
orig = plt.plot(series, label='Original')
mean = plt.plot(rolmean, color='red', label='Rolling Mean')
std = plt.plot(rolstd, color='black', label='Rolling Std')
plt.legend(loc='best')
plt.title('Rolling Mean & Standard Deviation')
plt.show(block=False)
```



Mean and Standar deviation does not have a constant behaviour over the years, This proves us again a non-stationary series

Check ACF & PACF plots for autocorelation

In [33]:
```
# from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
# #series = df[(df['datetime']>='2020-01-01') & (df['datetime']<'2023-01-01')]['close'].dropna()
# series = df['close'].dropna()
# plot_acf(series, lags=12)
# plot_pacf(series, lags=12)
# #plot_pacf(series, lags=7)
# plt.show()
```

We can see from the ACF curve that there is autocorrelation. As per PACF graph, lag - 1,2,3,4,5 are statistically significant, so we will include all in feature engineering.

In [34]:
```
## Save a copy of DataFrame for using later for different lagged Model for Multioutput
df_multi = df.copy()
```

Creating Features

In [35]:
```
data = df.copy()
data.head(2)
```

Out[35]:

	datetime	close	date	hour	dom	month	year	dow	doy	woy	week_number	friday	covid	covid_first_wave	covid_se
0	2015-01-01 09:15:00	18699.25	2015-01-01	10	1	1	2015	3	1	1	1	0.0	0.0	0.0	
1	2015-01-01 10:15:00	18700.20	2015-01-01	11	1	1	2015	3	1	1	1	0.0	0.0	0.0	

In [36]:
```
#data = radial_features(data).copy() # Have to install sklego
data = gen_cyclic_features(data).copy()
data = gen_lag_features(data).copy()
```

```
Adding Cyclic Time Indicators........
Adding TB Lag features...
Index(['datetime', 'close', 'date', 'hour', 'dom', 'month', 'year', 'dow',
       'doy', 'woy',
       ...
       'close_rolling_mean_28', 'close_rolling_max_28', 'close_rolling_min_28',
       'close_rolling_mean_29', 'close_rolling_max_29', 'close_rolling_min_29',
       'ramp1', 'ramp2', 'ramp3', 'ramp4'],
      dtype='object', length=132)
```

In [37]:
```
# # Plotting how the Time Features look after getting a sin cos transformation

# sns.scatterplot(x=data.sin_hour, y=data.cos_hour)
# plt.show()
```

Shifting by Lag Time forecasting

In [38]:

```
#target_ahead_timeblocks = 7  # Forecasting Lead Time for 15 mins candle

# Automatic calculation of target_ahead_timeblocks

def target_ahead_TB(data):
    minutes_TB = (data['datetime'][1]-data['datetime'][0]).seconds/60 # minus gives Minutes difference b
    target_ahead_timeblocks = (6*60)/minutes_TB + 1
    target_ahead_timeblocks = int(target_ahead_timeblocks)
    minutes_TB = int(minutes_TB)
    return (target_ahead_timeblocks, minutes_TB)

target_ahead_timeblocks = target_ahead_TB(data)[0]
frequency_in_mins = str(target_ahead_TB(data)[1]) + "min" # We need that for generating DF for final for


print(target_ahead_timeblocks)
print(frequency_in_mins)
```

```
7
60min
```

In [39]:

```
# Shifting of Data as per Forecasting Lead Time
def shift_variable_by_TB(data, target_ahead_timeblocks):

    data['ramp_close'] = data['close'] - data['close'].shift(target_ahead_timeblocks) # Creating one fea
    features = data.columns.tolist()


  # Creating target variable
    data['target'] = data['close'].shift(-target_ahead_timeblocks)

    return data
```

In [40]:

```
data = shift_variable_by_TB(data, target_ahead_timeblocks)
```

In [41]:

```
# Save DataFrame to new variable to make a check point
data_for_training = data.set_index('datetime').copy()
data_for_training.tail(28)
```

Out[41]:

datetime	close	date	hour	dom	month	year	dow	doy	woy	week_number	...	close_rolling_min_28	close_rolling_mean
2024-01-25 09:15:00	44824.35	2024-01-25	10	25	1	2024	3	25	4	4	...	44709.10	45548.294
2024-01-25 10:15:00	44601.00	2024-01-25	11	25	1	2024	3	25	4	4	...	44709.10	45515.962
2024-01-25 11:15:00	44558.60	2024-01-25	12	25	1	2024	3	25	4	4	...	44601.00	45463.946
2024-01-25 12:15:00	44542.25	2024-01-25	13	25	1	2024	3	25	4	4	...	44558.60	45416.651
2024-01-25 13:15:00	44537.55	2024-01-25	14	25	1	2024	3	25	4	4	...	44542.25	45369.410
2024-01-25 14:15:00	44872.45	2024-01-25	15	25	1	2024	3	25	4	4	...	44537.55	45322.010
2024-01-25 15:15:00	44988.50	2024-01-25	16	25	1	2024	3	25	4	4	...	44537.55	45297.067
2024-01-29 09:15:00	45463.90	2024-01-29	10	29	1	2024	0	29	5	5	...	44537.55	45272.791
2024-01-29 10:15:00	45546.55	2024-01-29	11	29	1	2024	0	29	5	5	...	44537.55	45263.415
2024-01-29 11:15:00	45497.55	2024-01-29	12	29	1	2024	0	29	5	5	...	44537.55	45253.294
2024-01-29 12:15:00	45449.30	2024-01-29	13	29	1	2024	0	29	5	5	...	44537.55	45237.627
2024-01-29 13:15:00	45439.35	2024-01-29	14	29	1	2024	0	29	5	5	...	44537.55	45221.700

datetime	close	date	hour	dom	month	year	dow	doy	woy	week_number	...	close_rolling_min_28	close_rolling_mean
2024-01-29 14:15:00	45493.15	2024-01-29	15	29	1	2024	0	29	5	5	...	44537.55	45199.672
2024-01-29 15:15:00	45499.05	2024-01-29	16	29	1	2024	0	29	5	5	...	44537.55	45178.598
2024-01-30 09:15:00	45351.35	2024-01-30	10	30	1	2024	1	30	5	5	...	44537.55	45158.736
2024-01-30 10:15:00	45342.45	2024-01-30	11	30	1	2024	1	30	5	5	...	44537.55	45132.617
2024-01-30 11:15:00	45454.30	2024-01-30	12	30	1	2024	1	30	5	5	...	44537.55	45110.160
2024-01-30 12:15:00	45440.20	2024-01-30	13	30	1	2024	1	30	5	5	...	44537.55	45100.865
2024-01-30 13:15:00	45466.20	2024-01-30	14	30	1	2024	1	30	5	5	...	44537.55	45111.084
2024-01-30 14:15:00	45395.80	2024-01-30	15	30	1	2024	1	30	5	5	...	44537.55	45119.498
2024-01-30 15:15:00	45319.55	2024-01-30	16	30	1	2024	1	30	5	5	...	44537.55	45130.598
2024-01-31 09:15:00	45824.45	2024-01-31	10	31	1	2024	2	31	5	5	...	44537.55	45140.589
2024-01-31 10:15:00	46056.25	2024-01-31	11	31	1	2024	2	31	5	5	...	44537.55	45168.800
2024-01-31 11:15:00	46046.35	2024-01-31	12	31	1	2024	2	31	5	5	...	44537.55	45196.189

datetime	close	date	hour	dom	month	year	dow	doy	woy	week_number	...	close_rolling_min_28	close_rolling_mear
2024-01-31 12:15:00	46015.45	2024-01-31	13	31	1	2024	2	31	5	5	...	44537.55	45223.477
2024-01-31 13:15:00	46041.30	2024-01-31	14	31	1	2024	2	31	5	5	...	44537.55	45268.524
2024-01-31 14:15:00	45990.60	2024-01-31	15	31	1	2024	2	31	5	5	...	44537.55	45311.187
2024-01-31 15:15:00	45974.00	2024-01-31	16	31	1	2024	2	31	5	5	...	44537.55	45353.136

28 rows × 133 columns

In [42]:
```
data.columns
```

Out[42]:
```
Index(['datetime', 'close', 'date', 'hour', 'dom', 'month', 'year', 'dow',
       'doy', 'woy',
       ...
       'close_rolling_min_28', 'close_rolling_mean_29', 'close_rolling_max_29',
       'close_rolling_min_29', 'ramp1', 'ramp2', 'ramp3', 'ramp4',
       'ramp_close', 'target'],
      dtype='object', length=134)
```

In [43]:
```
data.isna().sum()
```

Out[43]:
```
datetime      0
close         0
date          0
hour          0
dom           0
              ..
ramp2         2
ramp3         3
ramp4         4
ramp_close    7
target        7
Length: 134, dtype: int64
```

In [44]:

```
data.tail(10)
#data.head()
```

Out[44]:

	datetime	close	date	hour	dom	month	year	dow	doy	woy	...	close_rolling_min_28	close_rolling_mean_29	cl
15674	2024-01-30 13:15:00	45466.20	2024-01-30	14	30	1	2024	1	30	5	...	44537.55	45111.084483	
15675	2024-01-30 14:15:00	45395.80	2024-01-30	15	30	1	2024	1	30	5	...	44537.55	45119.498276	
15676	2024-01-30 15:15:00	45319.55	2024-01-30	16	30	1	2024	1	30	5	...	44537.55	45130.598276	
15677	2024-01-31 09:15:00	45824.45	2024-01-31	10	31	1	2024	2	31	5	...	44537.55	45140.589655	
15678	2024-01-31 10:15:00	46056.25	2024-01-31	11	31	1	2024	2	31	5	...	44537.55	45168.800000	
15679	2024-01-31 11:15:00	46046.35	2024-01-31	12	31	1	2024	2	31	5	...	44537.55	45196.189655	
15680	2024-01-31 12:15:00	46015.45	2024-01-31	13	31	1	2024	2	31	5	...	44537.55	45223.477586	
15681	2024-01-31 13:15:00	46041.30	2024-01-31	14	31	1	2024	2	31	5	...	44537.55	45268.524138	
15682	2024-01-31 14:15:00	45990.60	2024-01-31	15	31	1	2024	2	31	5	...	44537.55	45311.187931	
15683	2024-01-31 15:15:00	45974.00	2024-01-31	16	31	1	2024	2	31	5	...	44537.55	45353.136207	

10 rows × 134 columns

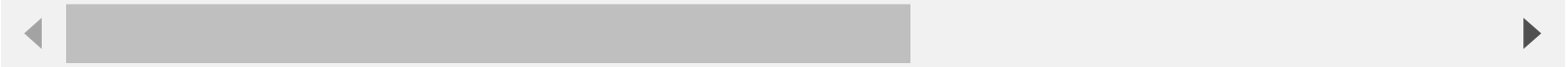
Check all NaN

In [45]:
```
data[data.isna().any(axis=1)].head(10)
```

Out[45]:

	datetime	close	date	hour	dom	month	year	dow	doy	woy	...	close_rolling_min_28	close_rolling_mean_29	close_
0	2015-01-01 09:15:00	18699.25	2015-01-01	10	1	1	2015	3	1	1	...	NaN	NaN	
1	2015-01-01 10:15:00	18700.20	2015-01-01	11	1	1	2015	3	1	1	...	18699.25	18699.250000	
2	2015-01-01 11:15:00	18723.80	2015-01-01	12	1	1	2015	3	1	1	...	18699.25	18699.725000	
3	2015-01-01 12:15:00	18699.40	2015-01-01	13	1	1	2015	3	1	1	...	18699.25	18707.750000	
4	2015-01-01 13:15:00	18734.90	2015-01-01	14	1	1	2015	3	1	1	...	18699.25	18705.662500	
5	2015-01-01 14:15:00	18745.90	2015-01-01	15	1	1	2015	3	1	1	...	18699.25	18711.510000	
6	2015-01-01 15:15:00	18759.75	2015-01-01	16	1	1	2015	3	1	1	...	18699.25	18717.241667	
7	2015-01-02 09:15:00	19033.70	2015-01-02	10	2	1	2015	4	2	1	...	18699.25	18723.314286	
8	2015-01-02 10:15:00	19037.90	2015-01-02	11	2	1	2015	4	2	1	...	18699.25	18762.112500	
9	2015-01-02 11:15:00	19045.15	2015-01-02	12	2	1	2015	4	2	1	...	18699.25	18792.755556	

10 rows × 134 columns



In [46]:
```
data[data.isna().any(axis=1)].tail(10)
```

Out[46]:

	datetime	close	date	hour	dom	month	year	dow	doy	woy	...	close_rolling_min_28	close_rolling_mean_29	cl
669	2015-05-26 13:15:00	18276.70	2015-05-26	14	26	5	2015	1	146	22	...	18299.10	18473.362069	
670	2015-05-26 14:15:00	18300.50	2015-05-26	15	26	5	2015	1	146	22	...	18276.70	18463.137931	
671	2015-05-26 15:15:00	18329.30	2015-05-26	16	26	5	2015	1	146	22	...	18276.70	18453.729310	
15677	2024-01-31 09:15:00	45824.45	2024-01-31	10	31	1	2024	2	31	5	...	44537.55	45140.589655	
15678	2024-01-31 10:15:00	46056.25	2024-01-31	11	31	1	2024	2	31	5	...	44537.55	45168.800000	
15679	2024-01-31 11:15:00	46046.35	2024-01-31	12	31	1	2024	2	31	5	...	44537.55	45196.189655	
15680	2024-01-31 12:15:00	46015.45	2024-01-31	13	31	1	2024	2	31	5	...	44537.55	45223.477586	
15681	2024-01-31 13:15:00	46041.30	2024-01-31	14	31	1	2024	2	31	5	...	44537.55	45268.524138	
15682	2024-01-31 14:15:00	45990.60	2024-01-31	15	31	1	2024	2	31	5	...	44537.55	45311.187931	
15683	2024-01-31 15:15:00	45974.00	2024-01-31	16	31	1	2024	2	31	5	...	44537.55	45353.136207	

10 rows × 134 columns

So above shows that we have top 671 rows as NaN due to moving averages and all

Also bottom 7 rows are with NaN as Target which is obvious as we have shifted by 7 (due to 15 min candle) to make time series as regression problem

In [47]:
```
# Function to make feature extraction systematic

def diff(list1, list2):
    c = set(list1).union(set(list2))  # or c = set(list1) | set(list2)
    d = set(list1).intersection(set(list2))  # or d = set(list1) & set(list2)
    return list(c - d)
```

In [48]:
```
# Feature segregation

def get_final_features(data):
    extra_features = ['weekend', 'date', 'year', 'target'] # Dont include 'datetime'
    X_features = [x for x in data.columns if x not in extra_features+['target']]
    return X_features

y_features = ['datetime', 'target']

X_features = get_final_features(data)
#print(X_features)
print(data[X_features].shape)
```

```
(15684, 131)
```

***** Filtered and Store the last rows for getting prediction for NEXT day***

In [49]:
```
next_day_df_prediction = data.iloc[-target_ahead_timeblocks:, :-1].copy()
next_day_df_prediction
```

Out[49]:

	datetime	close	date	hour	dom	month	year	dow	doy	woy	...	close_rolling_max_28	close_rolling_min_28	clos
15677	2024-01-31 09:15:00	45824.45	2024-01-31	10	31	1	2024	2	31	5	...	45546.55	44537.55	
15678	2024-01-31 10:15:00	46056.25	2024-01-31	11	31	1	2024	2	31	5	...	45824.45	44537.55	
15679	2024-01-31 11:15:00	46046.35	2024-01-31	12	31	1	2024	2	31	5	...	46056.25	44537.55	
15680	2024-01-31 12:15:00	46015.45	2024-01-31	13	31	1	2024	2	31	5	...	46056.25	44537.55	
15681	2024-01-31 13:15:00	46041.30	2024-01-31	14	31	1	2024	2	31	5	...	46056.25	44537.55	
15682	2024-01-31 14:15:00	45990.60	2024-01-31	15	31	1	2024	2	31	5	...	46056.25	44537.55	
15683	2024-01-31 15:15:00	45974.00	2024-01-31	16	31	1	2024	2	31	5	...	46056.25	44537.55	

7 rows × 133 columns

Dropping Missing Values

In [50]:

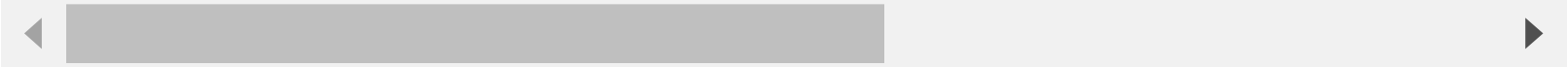
```
print(f'Before Dropping nans: {data_for_training.shape}')
data_for_training.dropna(inplace=True) # Dropping nans  # HERE PROBLEM IS THAT WE LOST NEXT DAY PREDICTI
data_for_training.reset_index(inplace=True) # Resetting index
print(data_for_training.shape)
data_for_training.tail(10)
```

```
Before Dropping nans: (15684, 133)
(15005, 134)
```

Out[50]:

	datetime	close	date	hour	dom	month	year	dow	doy	woy	...	close_rolling_min_28	close_rolling_mean_29	cl
14995	2024-01-29 13:15:00	45439.35	2024-01-29	14	29	1	2024	0	29	5	...	44537.55	45221.700000	
14996	2024-01-29 14:15:00	45493.15	2024-01-29	15	29	1	2024	0	29	5	...	44537.55	45199.672414	
14997	2024-01-29 15:15:00	45499.05	2024-01-29	16	29	1	2024	0	29	5	...	44537.55	45178.598276	
14998	2024-01-30 09:15:00	45351.35	2024-01-30	10	30	1	2024	1	30	5	...	44537.55	45158.736207	
14999	2024-01-30 10:15:00	45342.45	2024-01-30	11	30	1	2024	1	30	5	...	44537.55	45132.617241	
15000	2024-01-30 11:15:00	45454.30	2024-01-30	12	30	1	2024	1	30	5	...	44537.55	45110.160345	
15001	2024-01-30 12:15:00	45440.20	2024-01-30	13	30	1	2024	1	30	5	...	44537.55	45100.865517	
15002	2024-01-30 13:15:00	45466.20	2024-01-30	14	30	1	2024	1	30	5	...	44537.55	45111.084483	
15003	2024-01-30 14:15:00	45395.80	2024-01-30	15	30	1	2024	1	30	5	...	44537.55	45119.498276	
15004	2024-01-30 15:15:00	45319.55	2024-01-30	16	30	1	2024	1	30	5	...	44537.55	45130.598276	

10 rows × 134 columns



Train Test Split

In [51]:
```
## Experiments to check code

print(data_for_training['datetime'].max())

print(data_for_training['datetime'].max() - np.timedelta64(1, 'M'))

st_date = data_for_training['datetime'].max() - np.timedelta64(1, 'M')
f = dt.datetime(st_date.year, st_date.month, st_date.day, 9, 15)
print(f)
```

```
2024-01-30 15:15:00
2023-12-31 04:45:54
2023-12-31 09:15:00
```

In [52]:
```
#months_for_cutoff = 3
#months_for_cutoff = 2

val_cut = data_for_training['datetime'].max() - np.timedelta64(3, 'M')
test_cut = data_for_training['datetime'].max() - np.timedelta64(1, 'M')

validation_cutoff = dt.datetime(val_cut.year, val_cut.month, val_cut.day, 15, 15)
test_cutoff = dt.datetime(test_cut.year, test_cut.month, test_cut.day, 15, 15)

print(f"X Train Date till: {validation_cutoff}")
print(f"X Validation Date till: {test_cutoff}")
#print(f"X Test Date till: {data_for_training['datetime'].max()}")
```

```
X Train Date till: 2023-10-31 15:15:00
X Validation Date till: 2023-12-31 15:15:00
```

In [53]:
```
X = data_for_training[X_features].copy()
y = data_for_training[y_features].copy()
X.columns
```

Out[53]:
```
Index(['datetime', 'close', 'hour', 'dom', 'month', 'dow', 'doy', 'woy',
       'week_number', 'friday',
       ...
       'close_rolling_max_28', 'close_rolling_min_28', 'close_rolling_mean_29',
       'close_rolling_max_29', 'close_rolling_min_29', 'ramp1', 'ramp2',
       'ramp3', 'ramp4', 'ramp_close'],
      dtype='object', length=131)
```

In [54]:
```
def train_valid_test_split(X, y, validation_cutoff, test_cutoff):
    X_train = X[X['datetime'] <= validation_cutoff].iloc[:,1:].copy()
    y_train = y[y['datetime'] <= validation_cutoff].iloc[:,1:].copy()
    X_valid = X[(X['datetime'] > validation_cutoff) & (X['datetime'] <= test_cutoff)].iloc[:,1:].copy()
    y_valid = y[(y['datetime'] > validation_cutoff) & (y['datetime'] <= test_cutoff)].iloc[:,1:].copy()
    X_test = X[X['datetime'] > test_cutoff].iloc[:,1:].copy()
    y_test = y[y['datetime'] > test_cutoff].iloc[:,1:].copy()
    print(f'X Train: {X_train.shape}')
    print(f'X Validation: {X_valid.shape}')
    print(f'X Test: {X_test.shape}')
    print(f'y Train: {y_train.shape}')
    print(f'y Validation: {y_valid.shape}')
    print(f'y Test: {y_test.shape}')

    print("\n")

    X_train_date = X[X['datetime'] <= validation_cutoff].iloc[:,0:1].copy()
    X_valid_date = X[(X['datetime'] > validation_cutoff) & (X['datetime'] <= test_cutoff)].iloc[:,0:1].c
    X_test_date = X[X['datetime'] > test_cutoff].iloc[:,0:1].copy()
    print(f"X Train Date: {X_train_date['datetime'].max()}")
    print(f"X Validation Date: {X_valid_date['datetime'].max()}")
    print(f"X Test Date: {X_test_date['datetime'].max()}")

    return X_train, y_train, X_valid, y_valid, X_test, y_test
```

In [55]:
```
X_train, y_train, X_valid, y_valid, X_test, y_test = train_valid_test_split(X, y, validation_cutoff, tes
```

```
X Train: (14577, 130)
X Validation: (281, 130)
X Test: (147, 130)
y Train: (14577, 1)
y Validation: (281, 1)
y Test: (147, 1)


X Train Date: 2023-10-31 15:15:00
X Validation Date: 2023-12-29 15:15:00
X Test Date: 2024-01-30 15:15:00
```

In [56]:
```
X.columns
```

Out[56]:
```
Index(['datetime', 'close', 'hour', 'dom', 'month', 'dow', 'doy', 'woy',
       'week_number', 'friday',
       ...
       'close_rolling_max_28', 'close_rolling_min_28', 'close_rolling_mean_29',
       'close_rolling_max_29', 'close_rolling_min_29', 'ramp1', 'ramp2',
       'ramp3', 'ramp4', 'ramp_close'],
      dtype='object', length=131)
```

In [57]:
```
X_train.columns
```

Out[57]:
```
Index(['close', 'hour', 'dom', 'month', 'dow', 'doy', 'woy', 'week_number',
       'friday', 'covid',
       ...
       'close_rolling_max_28', 'close_rolling_min_28', 'close_rolling_mean_29',
       'close_rolling_max_29', 'close_rolling_min_29', 'ramp1', 'ramp2',
       'ramp3', 'ramp4', 'ramp_close'],
      dtype='object', length=130)
```

In [58]:
```
X_train.tail()
```

Out[58]:

	close	hour	dom	month	dow	doy	woy	week_number	friday	covid	...	close_rolling_max_28	close_rolling_min_2
14572	42909.45	12	31	10	1	304	44	44	0.0	0.0	...	43152.9	42144.
14573	43004.85	13	31	10	1	304	44	44	0.0	0.0	...	43073.6	42144.
14574	42993.10	14	31	10	1	304	44	44	0.0	0.0	...	43073.6	42144.
14575	42843.60	15	31	10	1	304	44	44	0.0	0.0	...	43073.6	42144.
14576	42827.75	16	31	10	1	304	44	44	0.0	0.0	...	43073.6	42144.

5 rows × 130 columns

Mode-1 (XGBoost)

In [59]:

```
### Feature Selection


# XGBoost
import xgboost as xgb
xgb_model = xgb.XGBRegressor(
    #tree_method = "gpu_hist", # To use GPU
    booster='gbtree',
    objective='reg:squarederror',
    n_estimators=1000,
    max_depth=10,
    learning_rate=0.1,
    gamma=0,alpha=0,
    seed=2
)


# Here we are not using scaled features as its not required for xgbboost reg
eval_set = [(X_train, y_train), (X_valid, y_valid)]
xgb_model.fit(X_train, y_train, early_stopping_rounds=30, eval_metric="mae", eval_set=eval_set, verbose=
#xgb_model.fit(X_train, y_train, eval_metric="mae", eval_set=eval_set, verbose=200)
```

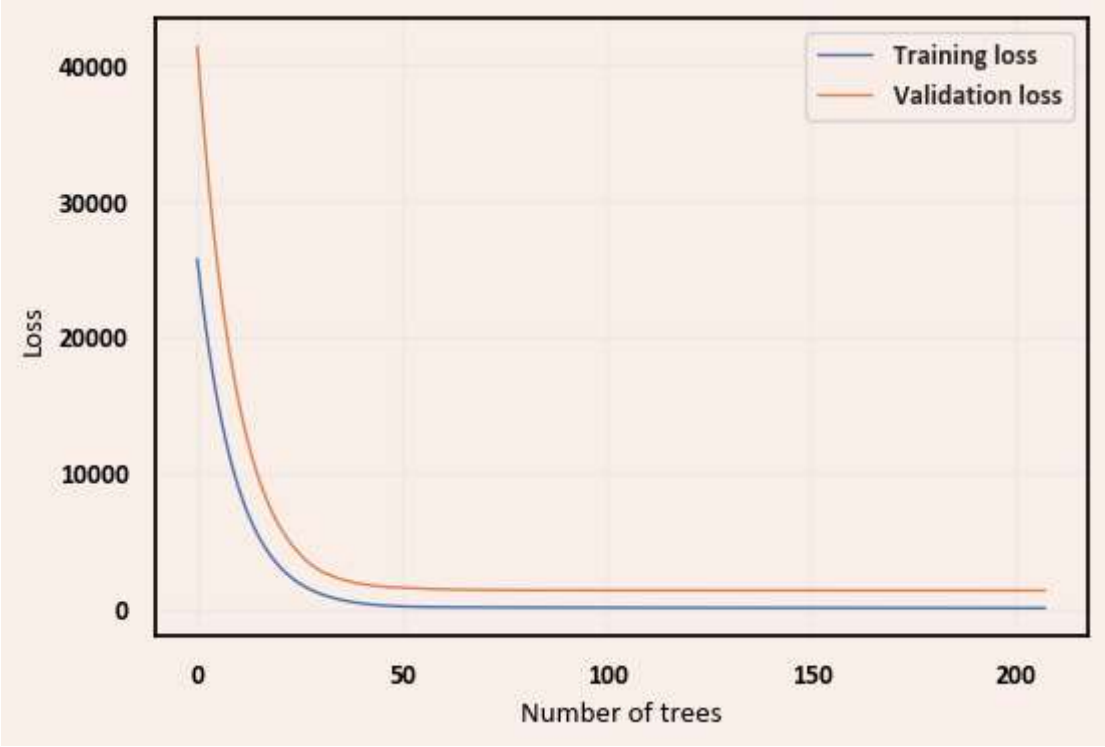
Out[59]:

```
XGBRegressor(alpha=0, base_score=None, booster='gbtree', 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=0, 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=10, max_leaves=None,
             min_child_weight=None, missing=nan, monotone_constraints=None,
             n_estimators=1000, n_jobs=None, num_parallel_tree=None,
             predictor=None, ...)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [60]:
```
results = xgb_model.evals_result()

plt.figure(figsize=(6,4))
plt.plot(results["validation_0"]["mae"], label="Training loss")
plt.plot(results["validation_1"]["mae"], label="Validation loss")
#plt.axvline(xgb_model.best_ntree_limit, color="gray", label="Optimal tree number")
plt.xlabel("Number of trees")
plt.ylabel("Loss")
plt.legend()
#print(f'No of Optimum Tress = {xgb_model.best_ntree_limit}')
```

Out[60]: <matplotlib.legend.Legend at 0x1e040cef400>



In [61]:
```
print(f'No of Optimum Tress = {xgb_model.best_ntree_limit}')
```

No of Optimum Tress = 178

In [62]:

```
feature_names= X_train.columns.tolist()
xgb_features = pd.DataFrame({'feature_names': feature_names,
                             'feature_importance': xgb_model.feature_importances_}).sort_values(by=['fea
xgb_features
```

Out[62]:

	feature_names	feature_importance
0	close_rolling_max_9	0.173944
1	close_rolling_max_17	0.145730
2	mcp_dam_ewm4h	0.134711
3	close_rolling_max_23	0.073766
4	close	0.065471
...	...	...
125	sin_hour	0.000001
126	week_number	0.000000
127	close_rolling_min_1	0.000000
128	close_rolling_max_1	0.000000
129	cos_hour	0.000000

130 rows × 2 columns

In [63]:
```
best_features = xgb_features[xgb_features['feature_importance']>=0.01]['feature_names'].tolist()
best_features
```

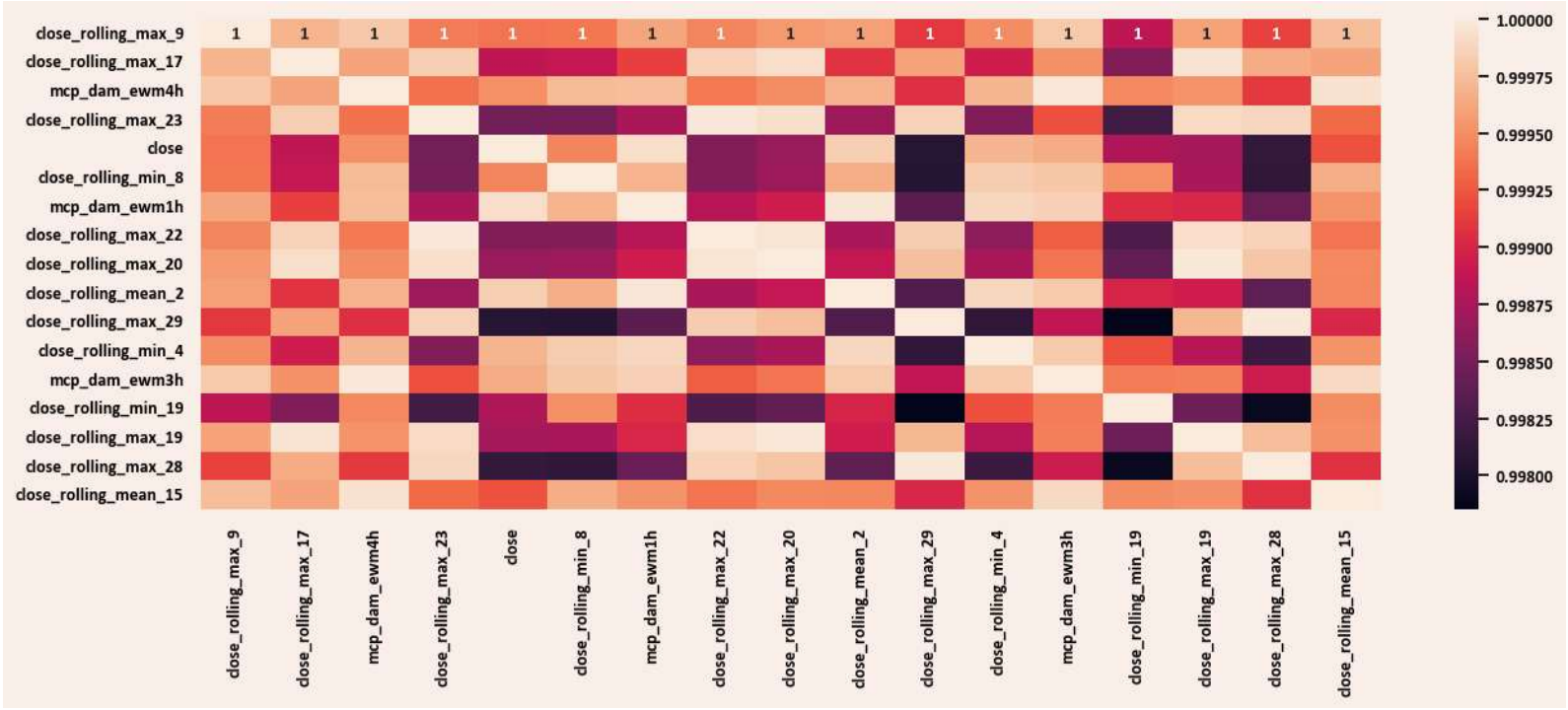
Out[63]:
```
['close_rolling_max_9',
 'close_rolling_max_17',
 'mcp_dam_ewm4h',
 'close_rolling_max_23',
 'close',
 'close_rolling_min_8',
 'mcp_dam_ewm1h',
 'close_rolling_max_22',
 'close_rolling_max_20',
 'close_rolling_mean_2',
 'close_rolling_max_29',
 'close_rolling_min_4',
 'mcp_dam_ewm3h',
 'close_rolling_min_19',
 'close_rolling_max_19',
 'close_rolling_max_28',
 'close_rolling_mean_15']
```

In [64]:
```
#xgb_features.to_csv('features.csv', index=False) # Storing important features for analyzing
```

Check all Best Features

In [65]:
```
sns.heatmap(data[best_features].corr(), annot=True)
```

Out[65]: <Axes: >



In [66]:
```
#sns.pairplot(data[best_features]) #later
```

Hyperparameter Tuning with Optuna

In [67]:
```
#pip install optuna
```

In [68]:

```
import xgboost as xgb
import optuna
from sklearn.metrics import mean_squared_error

def objective(trial):
    # Hyper Parameters
    param = {
        "booster": trial.suggest_categorical("booster", ["gbtree"]),
        "lambda": trial.suggest_int("lambda", 0, 100, step=10),
        "alpha": trial.suggest_int("alpha", 0, 100, step=10),
        "gamma": trial.suggest_float("gamma", 1e-8, 1.0, log=True),
        "subsample": trial.suggest_float("subsample", 0.8, 0.9),
        "min_child_weight": trial.suggest_int("min_child_weight", 2, 10),
        "colsample_bytree": trial.suggest_float("colsample_bytree", 0.8, 0.9),
        "max_leaves": trial.suggest_int("max_leaves", 1000, 2000,step=20),
        'max_depth': trial.suggest_int('max_depth', 10, 15),
        'learning_rate': trial.suggest_float('learning_rate', 0.01, 0.3),
        #"grow_policy": trial.suggest_categorical("grow_policy", ["depthwise", "lossguide"])
    }

    model = xgb.XGBRegressor(
        objective='reg:squarederror',
        #tree_method = "gpu_hist", # To use GPU
        n_estimators=2000,
        **param
        )
    model.fit(X_train[best_features],y_train,eval_set=[(X_valid[best_features],y_valid)],eval_metric='rm
    preds = model.predict(X_valid[best_features])
    error = mean_squared_error(y_valid, preds)
    return error

study = optuna.create_study(direction='minimize',study_name="XGB")
study.optimize(objective, n_trials=10) # We can increase n_trials to check more combinations, later incr

print('Number of finished trials:', len(study.trials))
print('Best trial:', study.best_trial.params)
```

```
[I 2024-01-31 21:21:06,294] A new study created in memory with name: XGB
```

```
[0]     validation_0-rmse:38674.93514
[100]   validation_0-rmse:1512.72347
[199]   validation_0-rmse:1521.98973

[I 2024-01-31 21:21:13,024] Trial 0 finished with value: 2282040.376690838 and parameters: {'booster':
'gbtree', 'lambda': 70, 'alpha': 0, 'gamma': 4.759817151211126e-05, 'subsample': 0.8528852055835079,
'min_child_weight': 9, 'colsample_bytree': 0.8051379587048194, 'max_leaves': 1040, 'max_depth': 10, 'l
earning_rate': 0.1813179992751221}. Best is trial 0 with value: 2282040.376690838.

[0]     validation_0-rmse:41890.94227
[100]   validation_0-rmse:1560.47153
[200]   validation_0-rmse:1550.83896
[300]   validation_0-rmse:1549.81846
[395]   validation_0-rmse:1552.20658

[I 2024-01-31 21:21:27,973] Trial 1 finished with value: 2399240.21249898 and parameters: {'booster':
'gbtree', 'lambda': 90, 'alpha': 20, 'gamma': 0.9260947185237114, 'subsample': 0.8747585774418413, 'mi
n_child_weight': 7, 'colsample_bytree': 0.8002618334911247, 'max_leaves': 1000, 'max_depth': 11, 'lear
ning_rate': 0.10560180681392357}. Best is trial 0 with value: 2282040.376690838.

[0]     validation_0-rmse:33879.74763
[100]   validation_0-rmse:1538.49929
[200]   validation_0-rmse:1543.21006
[210]   validation_0-rmse:1543.05187

[I 2024-01-31 21:21:37,905] Trial 2 finished with value: 2354546.247264171 and parameters: {'booster':
'gbtree', 'lambda': 60, 'alpha': 60, 'gamma': 0.00029794243641872526, 'subsample': 0.8429691715787617,
'min_child_weight': 8, 'colsample_bytree': 0.8987367592265758, 'max_leaves': 1200, 'max_depth': 12, 'l
earning_rate': 0.2995760199356628}. Best is trial 0 with value: 2282040.376690838.

[0]     validation_0-rmse:40689.87742
[100]   validation_0-rmse:1539.51799
[177]   validation_0-rmse:1546.88466

[I 2024-01-31 21:21:46,309] Trial 3 finished with value: 2358371.622531672 and parameters: {'booster':
'gbtree', 'lambda': 10, 'alpha': 80, 'gamma': 6.340676274909261e-07, 'subsample': 0.8827516721635694,
'min_child_weight': 9, 'colsample_bytree': 0.8711079644867904, 'max_leaves': 1820, 'max_depth': 14, 'l
earning_rate': 0.12167642167531119}. Best is trial 0 with value: 2282040.376690838.

[0]     validation_0-rmse:41207.35922
[100]   validation_0-rmse:1535.29770
[178]   validation_0-rmse:1532.42095
```

```
[I 2024-01-31 21:21:52,536] Trial 4 finished with value: 2344320.180199273 and parameters: {'booster':
'gbtree', 'lambda': 80, 'alpha': 30, 'gamma': 1.0784495164897512e-05, 'subsample': 0.8440712913757761,
'min_child_weight': 8, 'colsample_bytree': 0.8310899480348363, 'max_leaves': 1580, 'max_depth': 10, 'l
earning_rate': 0.12445982341240742}. Best is trial 0 with value: 2282040.376690838.

[0]     validation_0-rmse:37123.11769
[100]   validation_0-rmse:1636.00973
[185]   validation_0-rmse:1633.66441

[I 2024-01-31 21:21:59,545] Trial 5 finished with value: 2658406.974954214 and parameters: {'booster':
'gbtree', 'lambda': 30, 'alpha': 60, 'gamma': 0.0003780277198164919, 'subsample': 0.82677008362888, 'm
in_child_weight': 4, 'colsample_bytree': 0.8112172248536068, 'max_leaves': 1180, 'max_depth': 11, 'lea
rning_rate': 0.21046779158242487}. Best is trial 0 with value: 2282040.376690838.

[0]     validation_0-rmse:43936.21907
[100]   validation_0-rmse:2097.80710
[200]   validation_0-rmse:1599.79566
[300]   validation_0-rmse:1590.00735
[400]   validation_0-rmse:1590.34790
[453]   validation_0-rmse:1590.58213

[I 2024-01-31 21:22:16,907] Trial 6 finished with value: 2525899.795437916 and parameters: {'booster':
'gbtree', 'lambda': 10, 'alpha': 0, 'gamma': 2.2090633497770628e-07, 'subsample': 0.8637384157202157,
'min_child_weight': 5, 'colsample_bytree': 0.8687286709203834, 'max_leaves': 1720, 'max_depth': 11, 'l
earning_rate': 0.0428078474666052}. Best is trial 0 with value: 2282040.376690838.

[0]     validation_0-rmse:38498.15467
[100]   validation_0-rmse:1599.05953
[200]   validation_0-rmse:1594.74042
[300]   validation_0-rmse:1594.03851
[369]   validation_0-rmse:1594.94770

[I 2024-01-31 21:22:37,713] Trial 7 finished with value: 2537236.2869849703 and parameters: {'booste
r': 'gbtree', 'lambda': 90, 'alpha': 50, 'gamma': 1.2601241426403053e-05, 'subsample': 0.8289594730466
13, 'min_child_weight': 5, 'colsample_bytree': 0.8766298735417895, 'max_leaves': 1240, 'max_depth': 1
5, 'learning_rate': 0.19997942654634326}. Best is trial 0 with value: 2282040.376690838.
```

```
[0]     validation_0-rmse:44086.51835
[100]   validation_0-rmse:2806.14025
[200]   validation_0-rmse:1646.08108
[300]   validation_0-rmse:1616.20578
[400]   validation_0-rmse:1613.20948
[500]   validation_0-rmse:1611.20518
[600]   validation_0-rmse:1611.28254
[700]   validation_0-rmse:1610.36405
[767]   validation_0-rmse:1610.00857

[I 2024-01-31 21:23:09,462] Trial 8 finished with value: 2592000.4939281438 and parameters: {'booste
r': 'gbtree', 'lambda': 90, 'alpha': 100, 'gamma': 0.0002950257232718744, 'subsample': 0.8530327226187
013, 'min_child_weight': 5, 'colsample_bytree': 0.88313859502002, 'max_leaves': 1040, 'max_depth': 11,
'learning_rate': 0.04472199165923203}. Best is trial 0 with value: 2282040.376690838.

[0]     validation_0-rmse:41185.29382
[100]   validation_0-rmse:1553.82591
[200]   validation_0-rmse:1549.14075
[295]   validation_0-rmse:1553.50538

[I 2024-01-31 21:23:22,351] Trial 9 finished with value: 2395563.469883506 and parameters: {'booster':
'gbtree', 'lambda': 20, 'alpha': 90, 'gamma': 2.350187639274146e-07, 'subsample': 0.8407710511977055,
'min_child_weight': 9, 'colsample_bytree': 0.8910954937195925, 'max_leaves': 1020, 'max_depth': 12, 'l
earning_rate': 0.11012619246420535}. Best is trial 0 with value: 2282040.376690838.

Number of finished trials: 10
Best trial: {'booster': 'gbtree', 'lambda': 70, 'alpha': 0, 'gamma': 4.759817151211126e-05, 'subsampl
e': 0.8528852055835079, 'min_child_weight': 9, 'colsample_bytree': 0.8051379587048194, 'max_leaves': 1
040, 'max_depth': 10, 'learning_rate': 0.1813179992751221}
```

In [69]:
```
study.best_params
```

Out[69]:
```
{'booster': 'gbtree',
 'lambda': 70,
 'alpha': 0,
 'gamma': 4.759817151211126e-05,
 'subsample': 0.8528852055835079,
 'min_child_weight': 9,
 'colsample_bytree': 0.8051379587048194,
 'max_leaves': 1040,
 'max_depth': 10,
 'learning_rate': 0.1813179992751221}
```

In [70]:
```
param = study.best_params
xgbreg = xgb.XGBRegressor(
    objective='reg:squarederror',
    #tree_method = "gpu_hist", # To use GPU
    n_estimators=1000,
    **param
    )
xgbreg.fit(X_train[best_features],y_train,eval_set=[(X_valid[best_features],y_valid)],eval_metric='rmse'
```

```
[0]     validation_0-rmse:38674.93514
[94]    validation_0-rmse:1511.41745
```

Out[70]:
```
XGBRegressor(alpha=0, base_score=None, booster='gbtree', callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=0.8051379587048194, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=4.759817151211126e-05, gpu_id=None, grow_policy=None,
             importance_type=None, interaction_constraints=None, lambda=70,
             learning_rate=0.1813179992751221, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=10, max_leaves=1040,
             min_child_weight=9, missing=nan, monotone_constraints=None,
             n_estimators=1000, n_jobs=None, num_parallel_tree=None, ...)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [71]:
```
# Making predictions with the tuned model
pred_train = xgbreg.predict(X_train[best_features])
pred_valid = xgbreg.predict(X_valid[best_features])
pred_test = xgbreg.predict(X_test[best_features])
```

In [72]:

```
# Evaluate model accuracy
from sklearn.metrics import mean_squared_error
print(f"Train MSE: {mean_squared_error(y_train['target'],pred_train)}")
print(f"Train RMSE: {np.sqrt(mean_squared_error(y_train['target'],pred_train))}")
print(f"valid MSE: {mean_squared_error(y_valid['target'],pred_valid)}")
print(f"valid RMSE: {np.sqrt(mean_squared_error(y_valid['target'],pred_valid))}")
print(f"Test MSE: {mean_squared_error(y_test['target'],pred_test)}")
print(f"Test RMSE: {np.sqrt(mean_squared_error(y_test['target'],pred_test))}")
```

```
Train MSE: 86161.08313401835
Train RMSE: 293.53208195019903
valid MSE: 2283371.971062408
valid RMSE: 1511.0830457199922
Test MSE: 2978713.0632435074
Test RMSE: 1725.894858687373
```

In [73]:

```
result = y[y['datetime'] > test_cutoff].copy()
result[f'prediction_xgb_{stock}'] = pred_test
result.tail(10)
```

Out[73]:

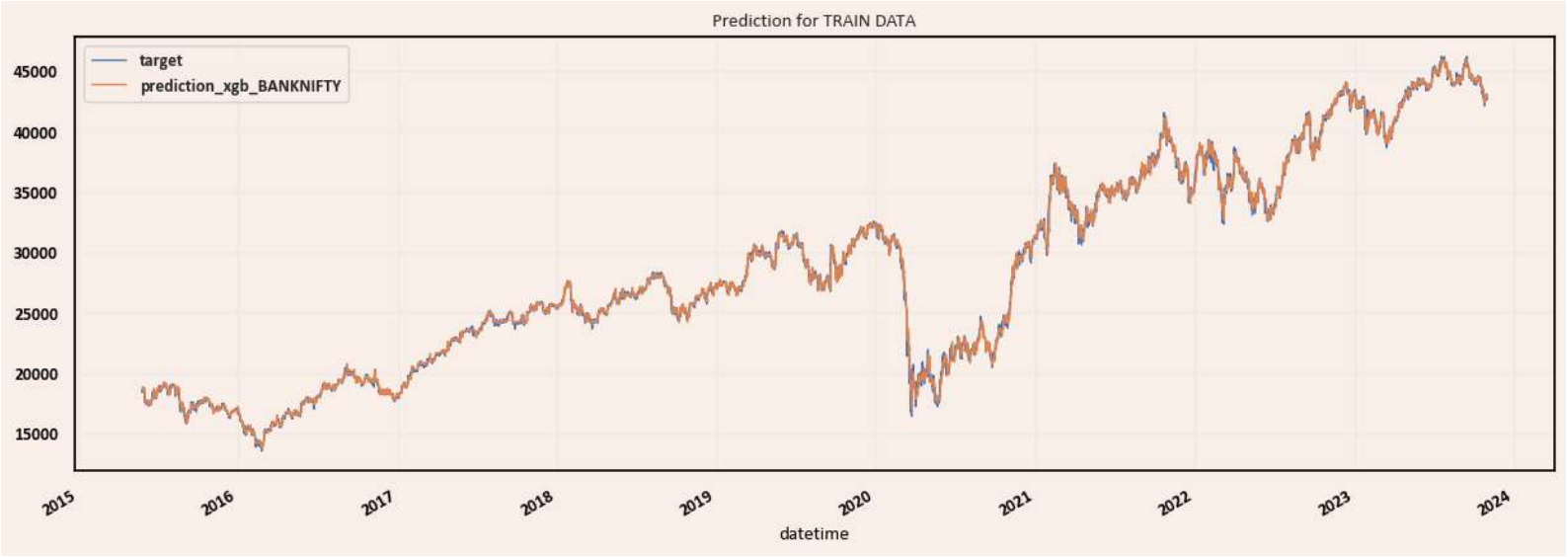
	datetime	target	prediction_xgb_BANKNIFTY
14995	2024-01-29 13:15:00	45466.20	45381.515625
14996	2024-01-29 14:15:00	45395.80	45458.226562
14997	2024-01-29 15:15:00	45319.55	45443.054688
14998	2024-01-30 09:15:00	45824.45	45214.722656
14999	2024-01-30 10:15:00	46056.25	45249.875000
15000	2024-01-30 11:15:00	46046.35	45401.175781
15001	2024-01-30 12:15:00	46015.45	45443.902344
15002	2024-01-30 13:15:00	46041.30	45555.699219
15003	2024-01-30 14:15:00	45990.60	45478.988281
15004	2024-01-30 15:15:00	45974.00	45373.769531

In [74]:

```
result_train = y[(y['datetime'] <= validation_cutoff)].copy()
result_train[f'prediction_xgb_{stock}'] = pred_train
plt.figure(figsize=(10,5))
result_train.set_index('datetime').plot()
plt.title(f"Prediction for TRAIN DATA")
```

Out[74]:

```
Text(0.5, 1.0, 'Prediction for TRAIN DATA')

<Figure size 1000x500 with 0 Axes>
```

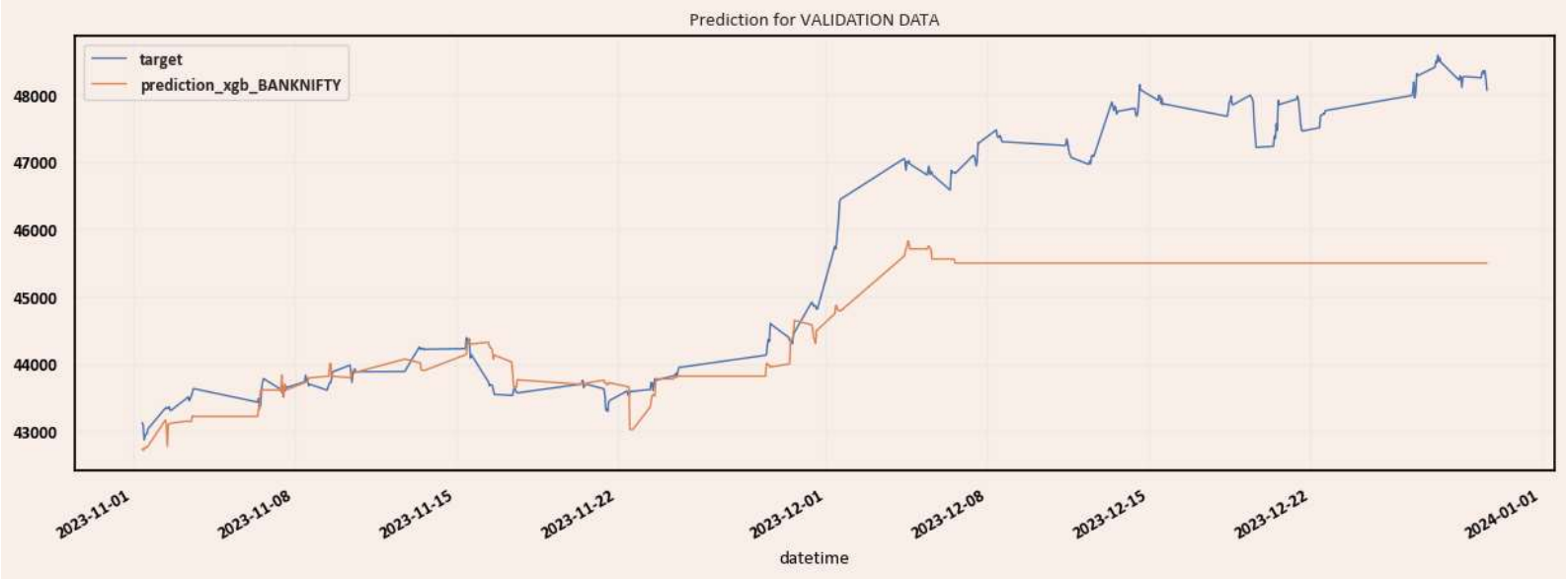


In [75]:

```
#We have data from 2020-07-01 00:00:00
#We have data till 2023-02-13 23:45:00

result_valid = y[(y['datetime'] > validation_cutoff) & (y['datetime'] <= test_cutoff)].copy()
result_valid[f'prediction_xgb_{stock}'] = pred_valid
plt.figure(figsize=(10,5))
result_valid.set_index('datetime').plot()
plt.title(f"Prediction for VALIDATION DATA")
```

Out[75]:

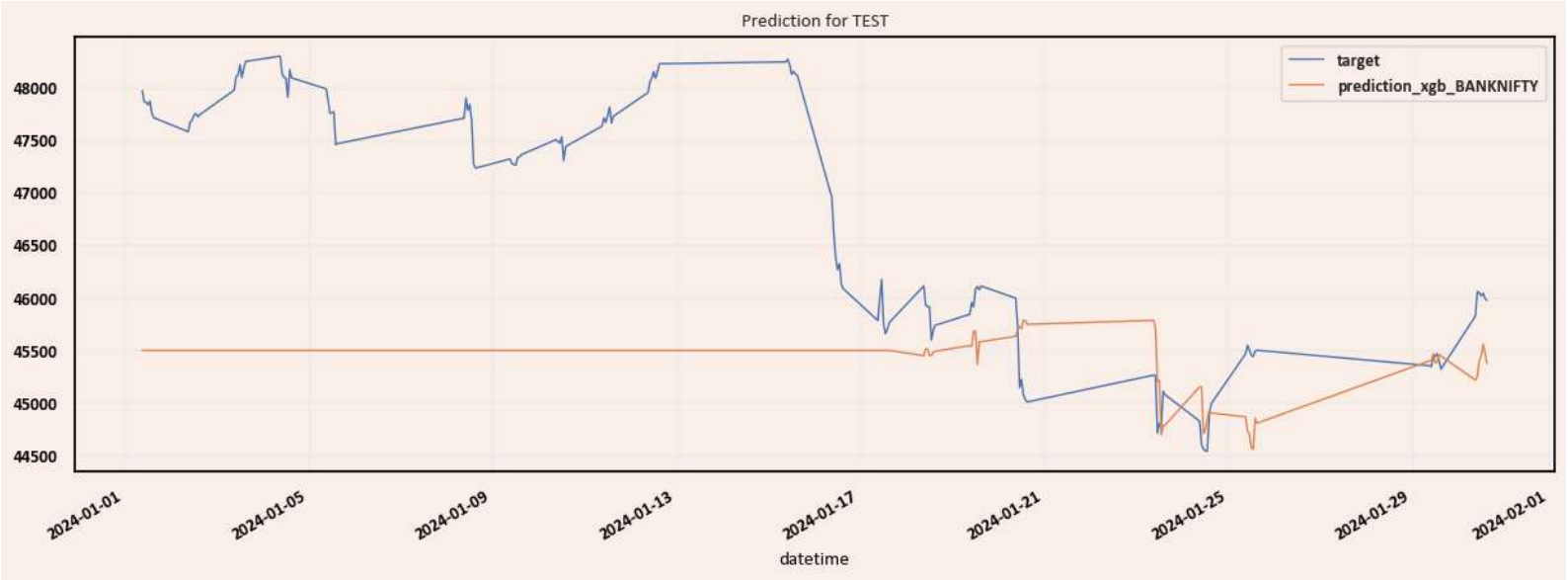
```
Text(0.5, 1.0, 'Prediction for VALIDATION DATA')

<Figure size 1000x500 with 0 Axes>
```



In [76]:

```
mask = (result.datetime > test_cutoff)
result.loc[mask,:].set_index('datetime')
result.loc[mask,:].set_index('datetime').plot()
plt.title("Prediction for TEST")

#result.loc[mask,:].set_index('datetime')
```

Out[76]: Text(0.5, 1.0, 'Prediction for TEST')



Find Last working and Next working DAY

In [77]:
```python
def check_is_inbetween_market_time(test_date):
    #----This block of code will check if we are running code before 3:30PM then it will not take today'
    if test_date not in holiday_list and test_date.weekday() not in [5, 6]:
        now = datetime.now()
        today15_30 = now.replace(hour=15, minute=30, second=0)
        if now <= today15_30:
            test_date = test_date-timedelta(days=1)
    return test_date
```

In [78]:
```python
def find_last_working_day(test_date):

    if test_date.weekday() == 0:
        diff = 3
    elif test_date.weekday() == 6:
        diff = 2
    else :
        diff = 1

    res = test_date - timedelta(days=diff)

    while str(res) in holiday_list:
        res = res - timedelta(days=1)

    if res.weekday() in [0, 6]:
        res = find_last_working_day(res) # Recursive call beacuse
        # may be we have today tuesday and on monday we have holiday
        # then without recursive call it will give sunday as last working day

    return res
```

In [79]:

```
def find_Next_working_day(test_date):

    if test_date.weekday() == 5:
        add = 2
    elif test_date.weekday() == 6:
        add = 1
    else :
        add = 0

    res = test_date + timedelta(days=add)

    while str(res) in holiday_list:
        res = res + timedelta(days=1)

    if res.weekday() in [5, 6]:
        res = find_Next_working_day(res) # Recursive call

    return res
```

In [80]:

```
holiday_list = ['2023-11-27', '2023-12-25']
#holiday_list = [x+' 00:00:00' for x in holiday_list]
#holiday_list = [datetime.strptime(x, '%Y-%m-%d') for x in holiday_list]


#test_date = datetime(2023, 11, 16)
test_date = dt.date.today()
test_date = check_is_inbetween_market_time(test_date)


last_working = find_last_working_day(test_date)
next_working = find_Next_working_day(test_date)

print(f"Input Date:{test_date} ~~~~ Last Working date:{str(last_working)}")
print(f"Input Date:{test_date} ~~~~ NEXT Working date:{str(next_working)}")
```

```
Input Date:2024-01-31 ~~~~ Last Working date:2024-01-30
Input Date:2024-01-31 ~~~~ NEXT Working date:2024-01-31
```

In [ ]:

In [ ]:

In [81]:
```
# result.datetime.to_numpy()[0] > dt.date.today()
```

In [82]:
```
#(result.datetime > pd.Timestamp.today())
#result.datetime > np.datetime64('today', 'D')
```

In [83]:
```
# ajj = dt.date.today()
# #ajj = np.datetime64('today', 'D')
# end_date = find_last_working_day(ajj)
# np.datetime64(end_date)
```

In [ ]:

In [ ]:

Peformance for LAST working DAY

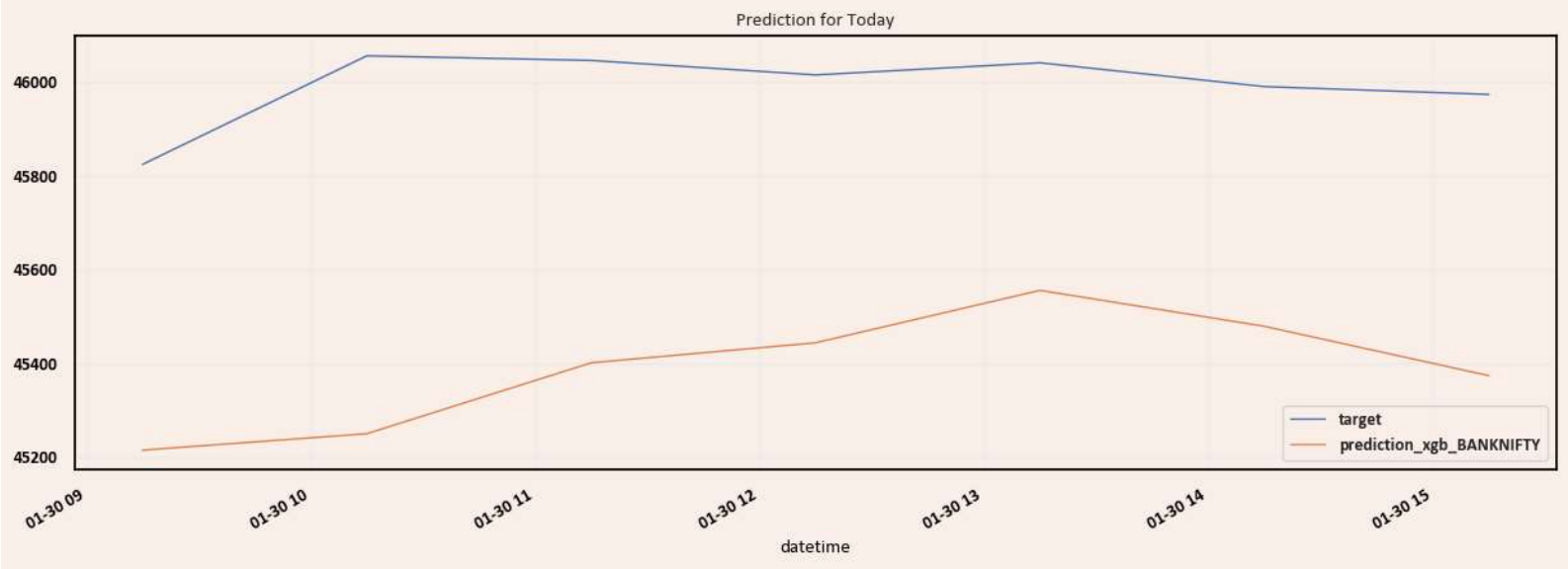
In [84]:

```
ajj = dt.date.today() # I guess its in python only
#ajj = np.datetime64('today', 'D') # How we define in numpy
#ajj = pd.Timestamp.today()  # How we define in pandas

end_date = find_last_working_day(ajj)



#---Below will work for weekdays as well--------------
#----- to run code for Saturday and Sunday------------------
if ajj.weekday() == 5:
    ajj = ajj - timedelta(days=1)
    end_date = end_date - timedelta(days=1)

if ajj.weekday() == 6:
    ajj = ajj - timedelta(days=2)
    end_date = end_date - timedelta(days=2)
#----- End of to run code for Saturday and Sunday------------------




print(f"Forcast for: {ajj}")
print("Basicall for today's prediction, Datetime will be yesterday's only as per logic As we have shifte
mask = (result.datetime > np.datetime64(end_date))
#mask = (result.datetime >= end_date)

result.loc[mask,:].set_index('datetime').plot()
plt.title("Prediction for Today")

result.loc[mask,:].set_index('datetime')
```

```
Forcast for: 2024-01-31
Basicall for today's prediction, Datetime will be yesterday's only as per logic As we have shifted tar
get by 1, so for Date 15th prediction Datetime will show 14th
```

Out[84]:

datetime	target	prediction_xgb_BANKNIFTY
2024-01-30 09:15:00	45824.45	45214.722656
2024-01-30 10:15:00	46056.25	45249.875000
2024-01-30 11:15:00	46046.35	45401.175781
2024-01-30 12:15:00	46015.45	45443.902344
2024-01-30 13:15:00	46041.30	45555.699219
2024-01-30 14:15:00	45990.60	45478.988281
2024-01-30 15:15:00	45974.00	45373.769531



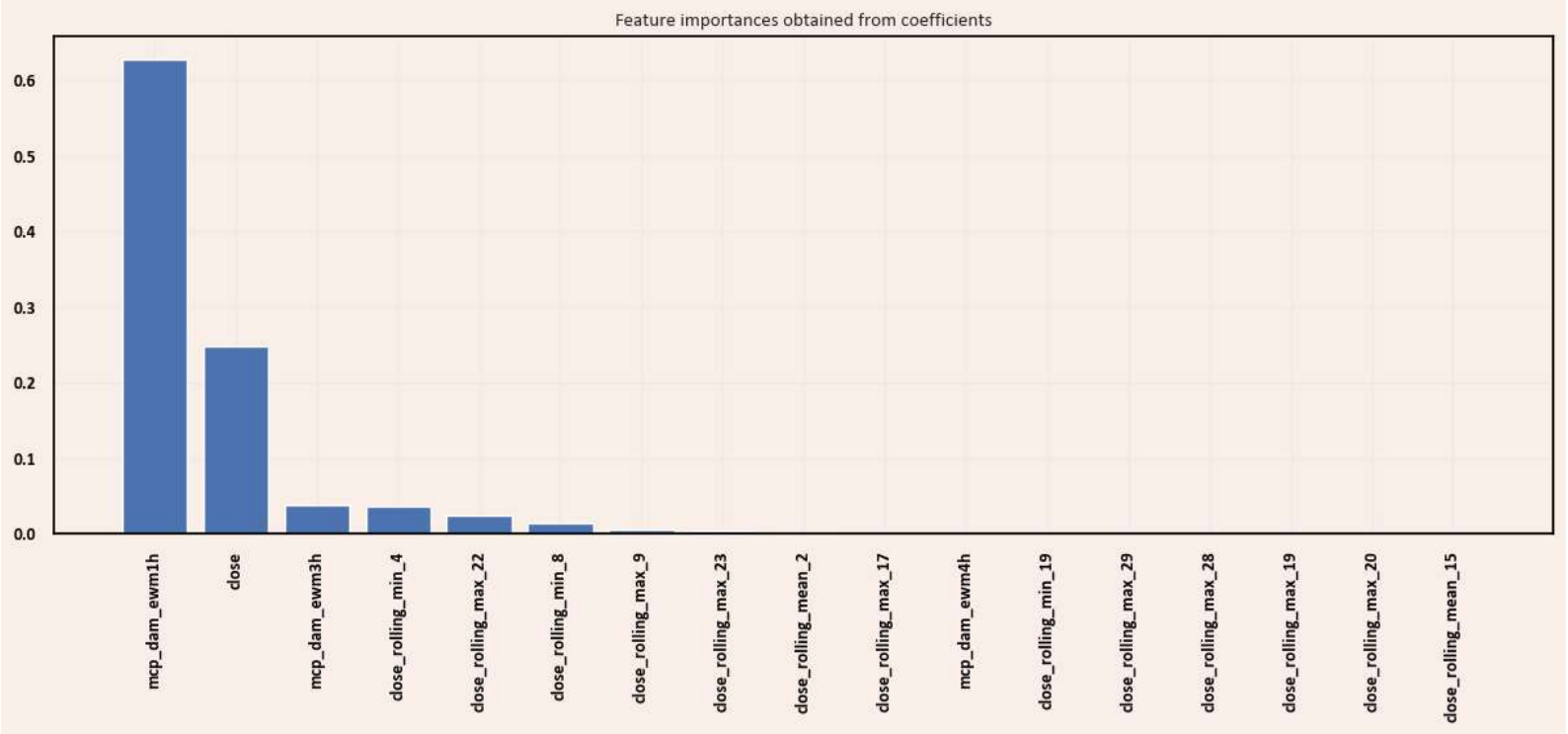
In [85]:
```
# # JUST ANOTHER WAY FOR ABOVE CODE

# end_date = dt.date.today()
# days_sub = 1 # XXXXXXXXXXXXXXXXXXXXXX
# end_date = dt.datetime(end_date.year, end_date.month, end_date.day-days_sub, 9, 15)

# print(f"Forcast Date~~~~~{end_date}")

# mask = (result.datetime >= end_date)
# result.loc[mask,:].set_index('datetime').plot()
# plt.title("Prediction for Today")


# result.loc[mask,:].set_index('datetime')
```

In [86]:
```
def feature_importance(features, importance_score):
    importances = pd.DataFrame(data={
        'Attribute': features,
        'Importance': importance_score
    })
    importances = importances.sort_values(by='Importance', ascending=False)
    plt.bar(x=importances['Attribute'], height=importances['Importance'])
    plt.title('Feature importances obtained from coefficients')
    plt.xticks(rotation='vertical')
    plt.show()
```

In [87]:
```
print(X_train.columns.shape)
print(xgbreg.feature_importances_.shape)
```

```
(130,)
(17,)
```

In [88]:
```
feature_importance(best_features, xgbreg.feature_importances_)
```



Final Model saving

In [89]:
```
# Saving the final Best features used by model
model_name = f"{stock}_best_features_XGBoost"
pickle.dump(best_features, open(f'03_Saved_Model/{model_name}.pkl', 'wb'))
```

In [90]:
```
# Saving the final model
model_name = f"{stock}_XGBoost"
pickle.dump(xgbreg, open(f'03_Saved_Model/{model_name}.pkl', 'wb'))
```

****Final Prediction (XGBoost)****

In [91]:
```
test_date = dt.date.today()
next_working = find_Next_working_day(test_date)
next_working = dt.datetime(next_working.year, next_working.month, next_working.day, 9, 15)
next_working
```

Out[91]:
```
datetime.datetime(2024, 1, 31, 9, 15)
```

In [92]:
```
pred_day_ahead = pd.DataFrame(pd.date_range(start=next_working, periods=target_ahead_timeblocks, freq=fr
pred_day_ahead.set_index("datetime", inplace = True)
pred_day_ahead[f'prediction_xgb_{stock}'] = xgbreg.predict(next_day_df_prediction[best_features])
pred_day_ahead
```

Out[92]:

datetime	prediction_xgb_BANKNIFTY
2024-01-31 09:15:00	45681.296875
2024-01-31 10:15:00	45790.476562
2024-01-31 11:15:00	45877.570312
2024-01-31 12:15:00	45888.953125
2024-01-31 13:15:00	45906.531250
2024-01-31 14:15:00	45906.531250
2024-01-31 15:15:00	45906.531250

In [93]:
```
# pred_day_ahead = pd.DataFrame()
# pred_day_ahead.index = next_day_df_prediction['datetime']
# pred_day_ahead[f'prediction_xgb_{stock}'] = xgbreg.predict(next_day_df_prediction[best_features])
# pred_day_ahead
```

Model-2 - Logistic Regression

In [94]:
```
#We Have the following:
#X_train, y_train, X_valid, y_valid, X_test, y_test = train_valid_test_split(X, y, validation_cutoff, te
```

In [ ]:

Multicollinearity check by VIF score

In [95]:

```
# """
# Let's now keep on removing features till we have a feature with vif<5
# """
# import statsmodels.api as sm

# scaler_vif = StandardScaler()
# X_train_vif_s = scaler_vif.fit_transform(X_train)




# vif_thr = 20
# #r2_thr = 0.85
# i = 0
# feats_removed = []

# cols2=X.columns

# while True:

#     vif = pd.DataFrame()
#     X_t = pd.DataFrame(X_train_vif_s, columns=X.columns)[cols2]
#     vif['Features'] = cols2
#     vif['VIF'] = [variance_inflation_factor(X_t.values, i) for i in range(X_t.shape[1])]
#     vif['VIF'] = round(vif['VIF'], 2)
#     vif = vif.sort_values(by = "VIF", ascending = False)

#   #cols2 = vif["Features"][1:].values
#     cols2 = vif["Features"][i:].values
#     X2 = pd.DataFrame(X_train_vif_s, columns=X.columns)[cols2]

#     #X2_sm = sm.add_constant(X2)  #Statmodels default is without intercept, to add intercept we need t
#     #sm_model = sm.OLS(list(y_train), X2_sm).fit()

#     print('Feature no. ', i)
#     print('VIF Score ', vif.iloc[i]['VIF'])
#     #print('Adjusted R2 ', sm_model.rsquared_adj)


#     #if (vif.iloc[i]['VIF'] < vif_thr) or (sm_model.rsquared_adj > r2_thr):
#     if (vif.iloc[i]['VIF'] < vif_thr):
#         print('Reached threshold')
```

```
#         print('Highest vif:',vif.iloc[i])
#         #print('Current adj.R2',sm_model.rsquared_adj)
#         print('Features removed:', i)
#         print('List of features removed:', feats_removed)
#         break
#     feats_removed.append(vif.iloc[i]['Features'])
#     i += 1
```

In [96]:
```
X_train
```

Out[96]:

	close	hour	dom	month	dow	doy	woy	week_number	friday	covid	...	close_rolling_max_28	close_rolling_min_2
0	18412.85	10	27	5	2	147	22	22	0.0	0.0	...	18706.5	18276.
1	18412.75	11	27	5	2	147	22	22	0.0	0.0	...	18706.5	18276.
2	18440.65	12	27	5	2	147	22	22	0.0	0.0	...	18706.5	18276.
3	18540.55	13	27	5	2	147	22	22	0.0	0.0	...	18706.5	18276.
4	18488.30	14	27	5	2	147	22	22	0.0	0.0	...	18706.5	18276.
...	...	...	...	...	...	...	...	...	...	...	...	...	.
14572	42909.45	12	31	10	1	304	44	44	0.0	0.0	...	43152.9	42144.
14573	43004.85	13	31	10	1	304	44	44	0.0	0.0	...	43073.6	42144.
14574	42993.10	14	31	10	1	304	44	44	0.0	0.0	...	43073.6	42144.
14575	42843.60	15	31	10	1	304	44	44	0.0	0.0	...	43073.6	42144.
14576	42827.75	16	31	10	1	304	44	44	0.0	0.0	...	43073.6	42144.

14577 rows × 130 columns

In [97]:
```
11638-(11056+441+141)
```

Out[97]:
```
0
```

Scaling

In [98]:
```
print(X_train.columns.tolist())
```

```
['close', 'hour', 'dom', 'month', 'dow', 'doy', 'woy', 'week_number', 'friday', 'covid', 'covid_first_
wave', 'covid_second_wave', 'cos_hour', 'sin_hour', 'cos_dow', 'sin_dow', 'cos_doy', 'sin_doy', 'lag1
h', 'lag2h', 'lag4h', 'lag6h', 'lag12h', 'lag1d', 'lag2d', 'lag3d', 'lag4d', 'lag5d', 'lag6d', 'lag7
d', 'mcp_dam_ewm1h', 'mcp_dam_ewm2h', 'mcp_dam_ewm3h', 'mcp_dam_ewm4h', 'mcp_dam_ewm6h', 'mcp_dam_ewm8
h', 'mcp_dam_ewm12h', 'mcp_dam_ewm24h', 'close_rolling_mean_1', 'close_rolling_max_1', 'close_rolling_
min_1', 'close_rolling_mean_2', 'close_rolling_max_2', 'close_rolling_min_2', 'close_rolling_mean_3',
'close_rolling_max_3', 'close_rolling_min_3', 'close_rolling_mean_4', 'close_rolling_max_4', 'close_ro
lling_min_4', 'close_rolling_mean_5', 'close_rolling_max_5', 'close_rolling_min_5', 'close_rolling_mea
n_6', 'close_rolling_max_6', 'close_rolling_min_6', 'close_rolling_mean_7', 'close_rolling_max_7', 'cl
ose_rolling_min_7', 'close_rolling_mean_8', 'close_rolling_max_8', 'close_rolling_min_8', 'close_rolli
ng_mean_9', 'close_rolling_max_9', 'close_rolling_min_9', 'close_rolling_mean_10', 'close_rolling_max_
10', 'close_rolling_min_10', 'close_rolling_mean_11', 'close_rolling_max_11', 'close_rolling_min_11',
'close_rolling_mean_12', 'close_rolling_max_12', 'close_rolling_min_12', 'close_rolling_mean_13', 'clo
se_rolling_max_13', 'close_rolling_min_13', 'close_rolling_mean_14', 'close_rolling_max_14', 'close_ro
lling_min_14', 'close_rolling_mean_15', 'close_rolling_max_15', 'close_rolling_min_15', 'close_rolling
_mean_16', 'close_rolling_max_16', 'close_rolling_min_16', 'close_rolling_mean_17', 'close_rolling_max
_17', 'close_rolling_min_17', 'close_rolling_mean_18', 'close_rolling_max_18', 'close_rolling_min_18',
'close_rolling_mean_19', 'close_rolling_max_19', 'close_rolling_min_19', 'close_rolling_mean_20', 'clo
se_rolling_max_20', 'close_rolling_min_20', 'close_rolling_mean_21', 'close_rolling_max_21', 'close_ro
lling_min_21', 'close_rolling_mean_22', 'close_rolling_max_22', 'close_rolling_min_22', 'close_rolling
_mean_23', 'close_rolling_max_23', 'close_rolling_min_23', 'close_rolling_mean_24', 'close_rolling_max
_24', 'close_rolling_min_24', 'close_rolling_mean_25', 'close_rolling_max_25', 'close_rolling_min_25',
'close_rolling_mean_26', 'close_rolling_max_26', 'close_rolling_min_26', 'close_rolling_mean_27', 'clo
se_rolling_max_27', 'close_rolling_min_27', 'close_rolling_mean_28', 'close_rolling_max_28', 'close_ro
lling_min_28', 'close_rolling_mean_29', 'close_rolling_max_29', 'close_rolling_min_29', 'ramp1', 'ramp
2', 'ramp3', 'ramp4', 'ramp_close']
```

In [99]:
```
len(X_train.columns)
```

Out[99]: 130

In [100]:
```
# rescale the features
#scaler = MinMaxScaler()
scaler = StandardScaler()
#scaler = RobustScaler()

X_train_scaled = scaler.fit_transform(X_train)
X_valid_scaled = scaler.transform(X_valid) # Transform to Validation data as well
X_test_scaled = scaler.transform(X_test) # Transform to test data as well
```

In [101]:
```
print('Train =', X_train_scaled.shape, y_train.shape)
print('Train =', X_valid_scaled.shape, y_valid.shape)
print('Test =', X_test_scaled.shape, y_test.shape)
```

```
Train = (14577, 130) (14577, 1)
Train = (281, 130) (281, 1)
Test = (147, 130) (147, 1)
```

In [102]:
```
X_train.columns
```

Out[102]:
```
Index(['close', 'hour', 'dom', 'month', 'dow', 'doy', 'woy', 'week_number',
       'friday', 'covid',
       ...
       'close_rolling_max_28', 'close_rolling_min_28', 'close_rolling_mean_29',
       'close_rolling_max_29', 'close_rolling_min_29', 'ramp1', 'ramp2',
       'ramp3', 'ramp4', 'ramp_close'],
      dtype='object', length=130)
```

In [103]:
```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

reg_model = LinearRegression(n_jobs=-1)
reg_model.fit(X_train_scaled, y_train)

# Make predictions using the testing set
y_train_pred = reg_model.predict(X_train_scaled)
X_valid_pred = reg_model.predict(X_valid_scaled)
y_test_pred = reg_model.predict(X_test_scaled)
```

In [104]:
```
# Making predictions with the tuned model
pred_train = reg_model.predict(X_train_scaled)
pred_valid = reg_model.predict(X_valid_scaled)
pred_test = reg_model.predict(X_test_scaled)
```

In [105]:
```
# Evaluate model accuracy
from sklearn.metrics import mean_squared_error
print(f"Train MSE: {mean_squared_error(y_train['target'],pred_train)}")
print(f"Train RMSE: {np.sqrt(mean_squared_error(y_train['target'],pred_train))}")
print(f"valid MSE: {mean_squared_error(y_valid['target'],pred_valid)}")
print(f"valid RMSE: {np.sqrt(mean_squared_error(y_valid['target'],pred_valid))}")
print(f"Test MSE: {mean_squared_error(y_test['target'],pred_test)}")
print(f"Test RMSE: {np.sqrt(mean_squared_error(y_test['target'],pred_test))}")
```

```
Train MSE: 150127.38016551328
Train RMSE: 387.4627468099524
valid MSE: 159313.64838844317
valid RMSE: 399.1411384315618
Test MSE: 312524.98399693705
Test RMSE: 559.0393402945244
```

In [106]:

```
#result = y[y['datetime'] > test_cutoff].copy()
result[f'prediction_LR_{stock}'] = pred_test
result.tail(10)
```

Out[106]:

	datetime	target	prediction_xgb_BANKNIFTY	prediction_LR_BANKNIFTY
14995	2024-01-29 13:15:00	45466.20	45381.515625	45328.739962
14996	2024-01-29 14:15:00	45395.80	45458.226562	45351.790498
14997	2024-01-29 15:15:00	45319.55	45443.054688	45423.129659
14998	2024-01-30 09:15:00	45824.45	45214.722656	45166.857862
14999	2024-01-30 10:15:00	46056.25	45249.875000	45165.913049
15000	2024-01-30 11:15:00	46046.35	45401.175781	45237.425906
15001	2024-01-30 12:15:00	46015.45	45443.902344	45222.501426
15002	2024-01-30 13:15:00	46041.30	45555.699219	45251.487454
15003	2024-01-30 14:15:00	45990.60	45478.988281	45205.509254
15004	2024-01-30 15:15:00	45974.00	45373.769531	45171.034366

In [107]:
```
result_train = y[(y['datetime'] <= validation_cutoff)].copy()
result_train[f'prediction_LR_{stock}'] = pred_train
plt.figure(figsize=(10,5))
result_train.set_index('datetime').plot()
plt.title(f"Prediction for TRAIN DATA")
```

Out[107]:
```
Text(0.5, 1.0, 'Prediction for TRAIN DATA')

<Figure size 1000x500 with 0 Axes>
```

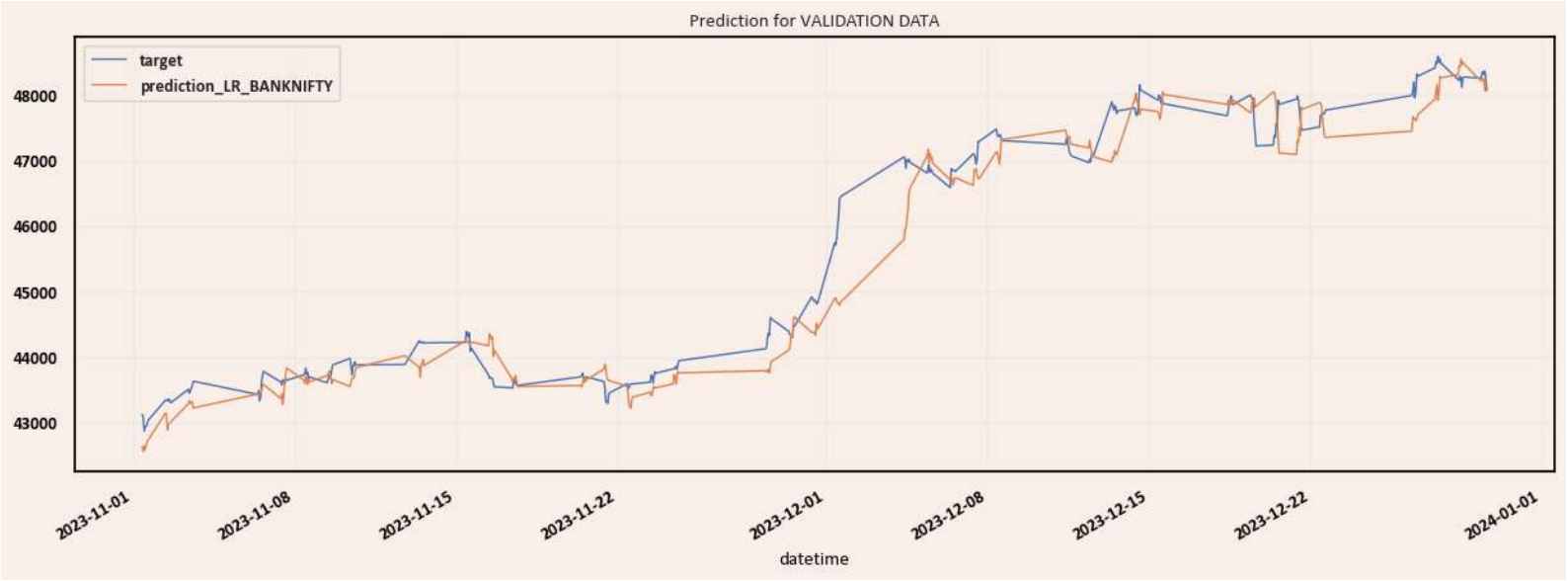
In [108]:

```
result_valid = y[(y['datetime'] > validation_cutoff) & (y['datetime'] <= test_cutoff)].copy()
result_valid[f'prediction_LR_{stock}'] = pred_valid
plt.figure(figsize=(10,5))
result_valid.set_index('datetime').plot()
plt.title(f"Prediction for VALIDATION DATA")
```

Out[108]:

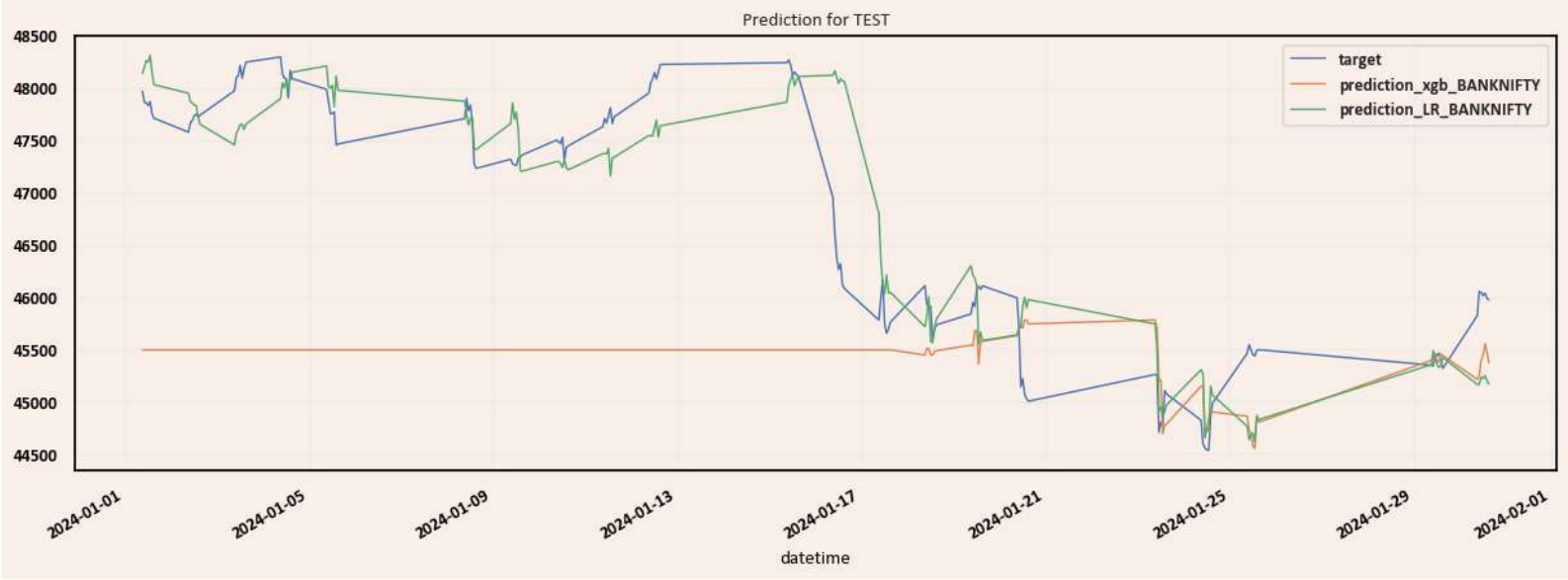
```
Text(0.5, 1.0, 'Prediction for VALIDATION DATA')
```

```
<Figure size 1000x500 with 0 Axes>
```



In [109]:
```
mask = (result.datetime > test_cutoff)
result.loc[mask,:].set_index('datetime')
result.loc[mask,:].set_index('datetime').plot()
plt.title("Prediction for TEST")
```

Out[109]:
```
Text(0.5, 1.0, 'Prediction for TEST')
```



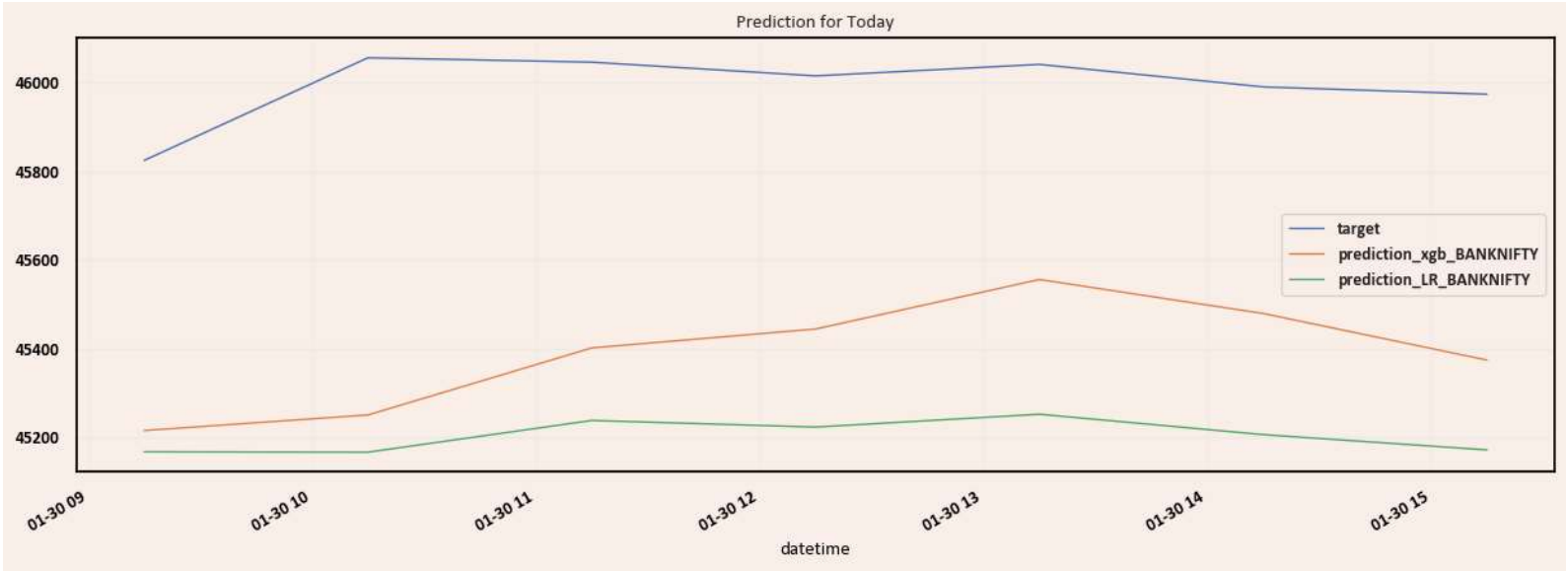
In [110]:

```
print(f"Forcast for: {ajj}")
print("Basicall for today's prediction, Datetime will be yesterday's only as per logic As we have shifte

mask = (result.datetime > np.datetime64(end_date))
#mask = (result.datetime >= end_date)

result.loc[mask,:].set_index('datetime').plot()
plt.title("Prediction for Today")

final_prediction_df = result.loc[mask,:].set_index('datetime').copy()
final_prediction_df
```

```
Forcast for: 2024-01-31
Basicall for today's prediction, Datetime will be yesterday's only as per logic As we have shifted tar
get by 1, so for Date 15th prediction Datetime will show 14th
```

Out[110]:

datetime	target	prediction_xgb_BANKNIFTY	prediction_LR_BANKNIFTY
2024-01-30 09:15:00	45824.45	45214.722656	45166.857862
2024-01-30 10:15:00	46056.25	45249.875000	45165.913049
2024-01-30 11:15:00	46046.35	45401.175781	45237.425906
2024-01-30 12:15:00	46015.45	45443.902344	45222.501426
2024-01-30 13:15:00	46041.30	45555.699219	45251.487454
2024-01-30 14:15:00	45990.60	45478.988281	45205.509254
2024-01-30 15:15:00	45974.00	45373.769531	45171.034366


Prediction for Today
46000
45800
45600
45400
45200
target
prediction_xgb_BANKNIFTY
prediction_LR_BANKNIFTY
01-30 09
01-30 10
01-30 11
01-30 12
01-30 13
01-30 14
01-30 15
datetime

Feature Importance checking

In [111]:
```
feature_importance(X_train.columns, reg_model.coef_[0])
```

Final Model saving

In [112]:
```
# Saving the final scaler model
model_name = f"{stock}_scaler_LR"
pickle.dump(scaler, open(f'03_Saved_Model/{model_name}.pkl', 'wb'))
```

In [113]:
```
# Saving the final model
model_name = f"{stock}_LR"
pickle.dump(reg_model, open(f'03_Saved_Model/{model_name}.pkl', 'wb'))
```

In [114]:
```
# Checking Current day Prediction of stock along with name
# Keep it mind that this is not final prediction for next day

final_prediction_df.insert(0,'Stock', stock)  # This is code to add column at specific location
final_prediction_df
```

Out[114]:

datetime	Stock	target	prediction_xgb_BANKNIFTY	prediction_LR_BANKNIFTY
2024-01-30 09:15:00	BANKNIFTY	45824.45	45214.722656	45166.857862
2024-01-30 10:15:00	BANKNIFTY	46056.25	45249.875000	45165.913049
2024-01-30 11:15:00	BANKNIFTY	46046.35	45401.175781	45237.425906
2024-01-30 12:15:00	BANKNIFTY	46015.45	45443.902344	45222.501426
2024-01-30 13:15:00	BANKNIFTY	46041.30	45555.699219	45251.487454
2024-01-30 14:15:00	BANKNIFTY	45990.60	45478.988281	45205.509254
2024-01-30 15:15:00	BANKNIFTY	45974.00	45373.769531	45171.034366

****Final Prediction (Linear Regression)****

In [115]:
```
tobe_used_features = get_final_features(next_day_df_prediction)
tobe_used_features.remove('datetime') # Remove DateTime as well as we dont need it
print(len(tobe_used_features))
```

```
130
```

In [116]:
```
next_day_df_prediction_LR = next_day_df_prediction[tobe_used_features]
#print(next_day_df_prediction_LR.columns.tolist())

next_day_df_prediction_LR_scaled = scaler.transform(next_day_df_prediction_LR) # Scaled the DF

pred_day_ahead[f'prediction_LR_{stock}'] = reg_model.predict(next_day_df_prediction_LR_scaled)
pred_day_ahead
```

Out[116]:

datetime	prediction_xgb_BANKNIFTY	prediction_LR_BANKNIFTY
2024-01-31 09:15:00	45681.296875	45700.323957
2024-01-31 10:15:00	45790.476562	46008.250044
2024-01-31 11:15:00	45877.570312	46068.141976
2024-01-31 12:15:00	45888.953125	45994.488114
2024-01-31 13:15:00	45906.531250	46044.866842
2024-01-31 14:15:00	45906.531250	46036.455516
2024-01-31 15:15:00	45906.531250	46049.413000

In [117]:
```
final_prediction_df_excel = pred_day_ahead.copy()
```

In [118]:

```
import xlwings as xw

df_date_check = final_prediction_df_excel.index[0]

wb = xw.Book('AngelOne_Option.xlsx')
exl_ML_prediction = wb.sheets['ML_prediction']


a = exl_ML_prediction[11,2].value
if a == final_prediction_df_excel.index[0]:  # we check if dates are already there and
                                             # if they are for correct forecasrt day else delete them
    print("Data Appended")
    for c in range(2,300):
        check_blank = exl_ML_prediction[10,c].value
        if check_blank is None:
            exl_ML_prediction[10,c].options(pd.DataFrame, header=1, index=False, expand='table').value =
            break
else:
    print("Brand New Data")
    exl_ML_prediction.range("B10:HQ200").clear_contents()
    exl_ML_prediction["C11"].options(pd.DataFrame, header=1, index=True, expand='table').value = final_p
```

```
Data Appended
```

In [119]:

```
print("All done and push a little more for more accuracy")
```

```
All done and push a little more for more accuracy
```