Data analysis and Forecasting of fuel prices in Germany

Project Overview

The fuel price is like the stock prices changing heavily on a day, but mostly not with the same amplitude. But even if you have a look on a day by day base, there are many up and downs. Therefore I want to have a look at the data and see what's inside. Furthermore I want to build a forecasting model for the price on daily base. I'll choose this as capstone project to become familiar with time series data and also forecasting model ARIMA.

The history of all fuel stations in Germany is available from Tankerkoenig.de at https://dev.azure.com/tankerkoenig/_git/tankerkoenig-data for private use under following license https://creativecommons.org/licenses/by-nc-sa/4.0/ .

In this project I'll have a closer look at the data, make some analysis and plots and finally try to fit some forecasting models and have a discussion on it.

All necessary information about libraries and so on are in the Readme.md in the github(https://github.com/klogges5/capstone) repository.

Problem statement

How to handle time series data?
How to prepare the data for building models?
Does the data have other useful insights?
When it is good time for refueling to save money?
Could I find a model to forecast the fuel price?

Therefore I'll have a look at the history data(2015-2019). For handling the amount of data I'll first reduced it to zip codes starting with 40. This leads to a 250 MB sql file. Because github is only storing 100 MB I'll reduced it further to 8 stations.

First I'll have a look at the data and see what is in. Therefore I'll use some plot functions to find some insights. For forecast model I'll use the SARIMAX from (Seabold, 2010) and will build the model based on ACF/PACF. Even it is not necessary in capstone project to use 2 models, I'll have a look also on LSTM as a forecast model.

Metrics

To answer the questions I'll go through some visualizations and analysis in the first notebooks.
For building and fitting the right model, I'll use Adfuller test and autocorrelation functions (ACF/PACF). With the Adfuller test I'll first test the stationarity of the data, because ARIMA

can handle only stationary data. To get stationarity of data I'll use the diff function.
The ACF/PACF will be used to find the right parameters for ARIMA model, therefore I'll used the described mechanism in "Mining Data from Time Series" course from Pluralsight (Burger, 2019).

To see how well the model fits I'll use the MSE and the R2 score. Because the differences of the prices are not so big and mostly only a few cent, like 0.01 € and so on, the MSE will get very small. That's why the R2 score as metric is also used. The R2 score should approximate 1 if the model is getting better.

Data Exploration

First I'll have a look at the stations.csv file and filtered it after reading only to the stations with zip code starting with 40.

```
short_pd = short_pd[short_pd['post_code'].str.match(pat = '40\d{3}')]
```

Than I've looked at two files out of all prices.csv files to see how they are look like. There I'll found some wrong prices:

min	-0.001000	-0.001000	-0.001000

The min fuel price could not be negative, so I've have to replace these values. To be shure I'll replace all prices < 0.5 € in my data reading part with np.nan (the fuel price wasn't below 0.5 € in the last 20 years in Germany). Also I'll found a joke in the data from one station, they have 2.015 € for all fuels and a short time on change to the 1st January 2015.

Than I read all price data year by year and reduced it directly to only use the stations in the short_pd and also replace wrong prices as mentioned above. You can see the details in jupyter notebook DataRead.ipynb, function `read_data_year` and `read_data`.
I'll converted the station_uuid to "category" to save memory. Also I set the date field to datetime64 for handling the data in later steps.

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2629941 entries, 74 to 240632
Data columns (total 5 columns):
date            datetime64[ns, UTC]
station_uuid    category
diesel          float64
e5              float64
e10             float64
dtypes: category(1), datetime64[ns, UTC](1), float64(3)
memory usage: 105.3 MB
```

After having a look at prices_pd.describe() I'll found, that we have a big difference between 75% percentile and the max value. So I'll have looked at the data and found reasonable max values and replaced incorrect data also with np.nan:

```
prices_pd['diesel'] = prices_pd['diesel'].apply(lambda x: np.nan if x > 1.7 else x)
prices_pd['e10'] = prices_pd['e10'].apply(lambda x: np.nan if x > 1.75 else x)
prices_pd['e5'] = prices_pd['e5'].apply(lambda x: np.nan if x > 1.82 else x)
```

Because I'll cannot use the np.nan in the forecasting model, I drop them later before I use the data in the model.
The resulting dataset is written in an sqllite database. Because the data is too big for storing it in github, I've further reduced it to 8 stations and saved the reduced prices_pd in prices_40.sql. If you download the raw data from above link, you can also read and use all data. The data_read process will took a while…

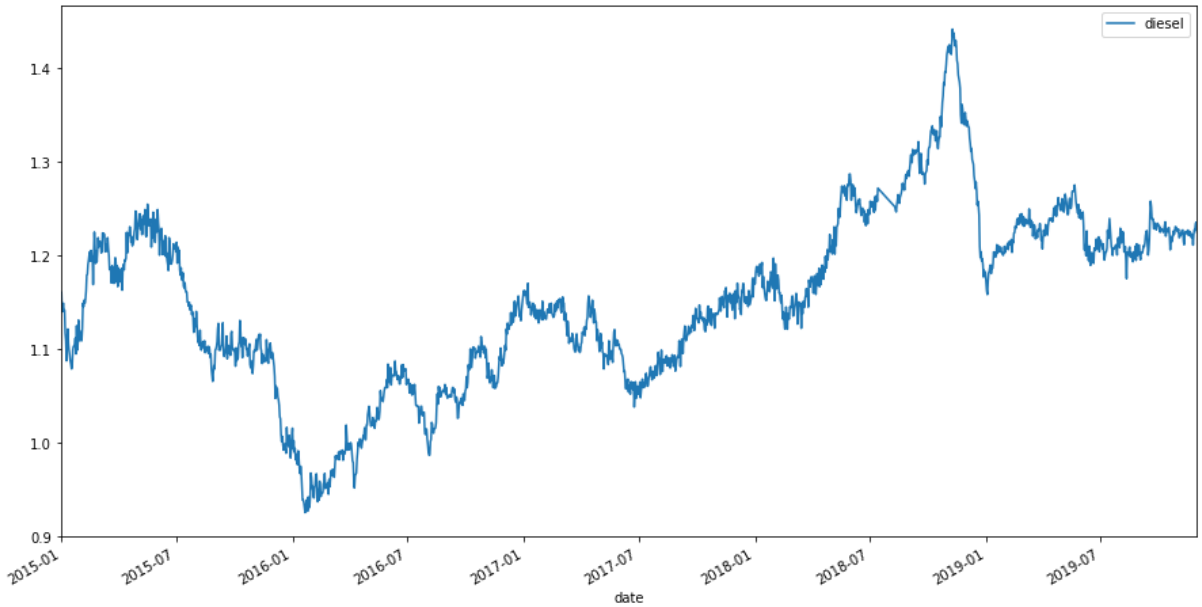
```
uuids = ['005056ba-7cb6-1ed2-bceb-82ea369c0d2d', '79fb1f24-bebb-489e-841f-728f9053b555', '51d4b59c-a095-1aa0-e100-80009459e03a',
'005056ba-7cb6-1ed2-bceb-a46e32000d3e',
       'e43c09b0-5baa-4738-962a-c94388e93c30', '82119d5d-775f-42af-ac56-000d7c91413f', 'e7807347-796f-4aac-997d-07d0c988e109',
'5bf85d09-ea6b-4146-b23f-4b902e2e1554']

prices_pd = prices_pd[prices_pd['station_uuid'].isin(uuids)][['date', 'station_uuid', 'diesel', 'e5', 'e10']]
```

For the data analysis I've first wrote me a function to extract data for one station and type(of gasoline) out of the data:

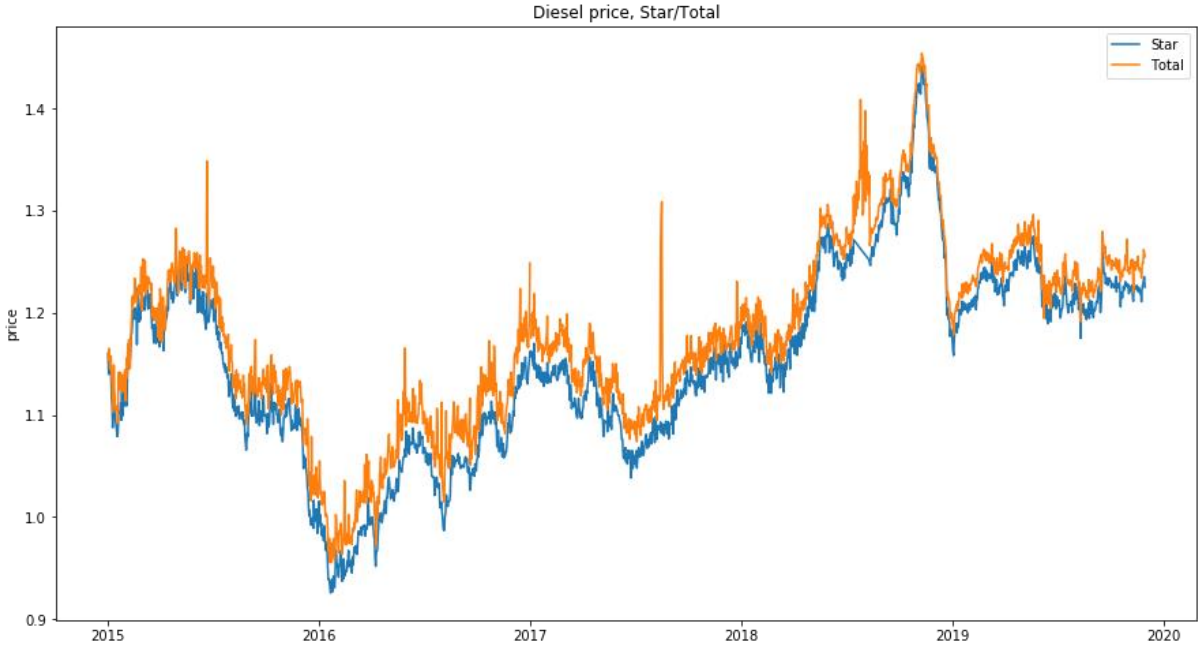
```
def get_data4uid(uid, typ):
    """
    Give the typ data for given uid
    Input: uid, typ
    Output: Dataframe"""
    data_uid = pd.DataFrame(prices_pd[prices_pd['station_uuid'] == uid][['date', typ]])

    # has to be changed, but for now utc=True
    data_uid.date = pd.to_datetime(data_uid.date, utc=True)

    data_uid.set_index('date', inplace=True)


    return data_uid
```

Data visualization

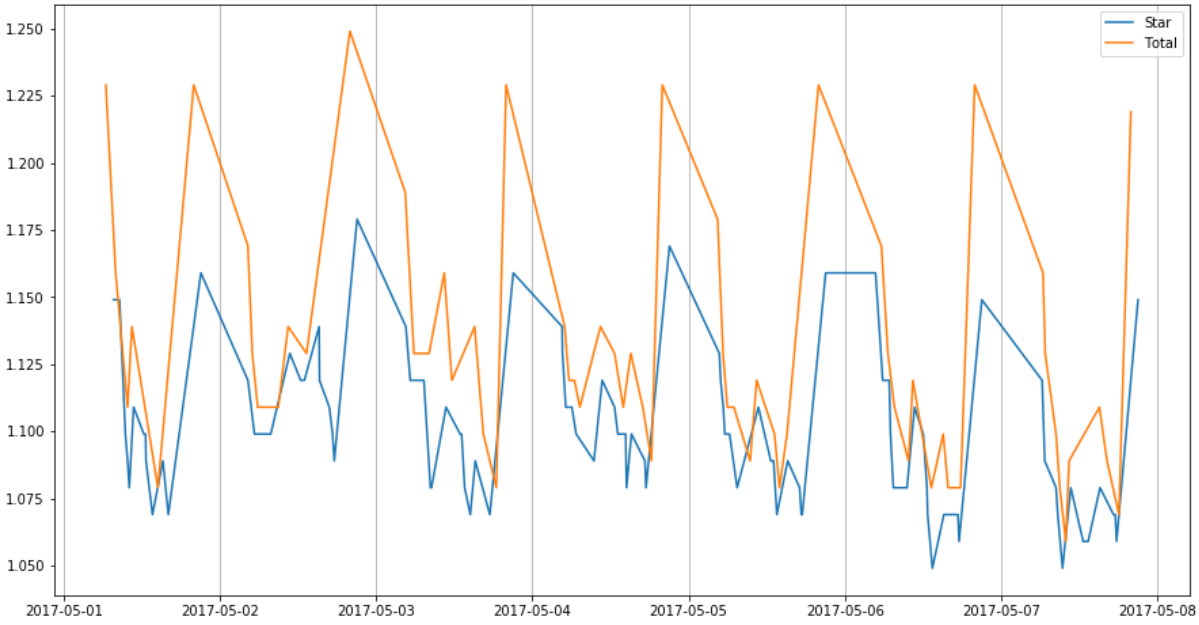
I resampled the data to daily mean values and have a first look in the data of one station for "diesel":



The data seems not really following a trend. We have some low at the beginning of 2016 and and high in last quarter of 2018. Let's have a look in two stations located side by side, that means they are in 500m distance.



At the first view it seems correlating, except some spike in total curve, but for a better analysis let's drill down in one week and take for that the raw data again:



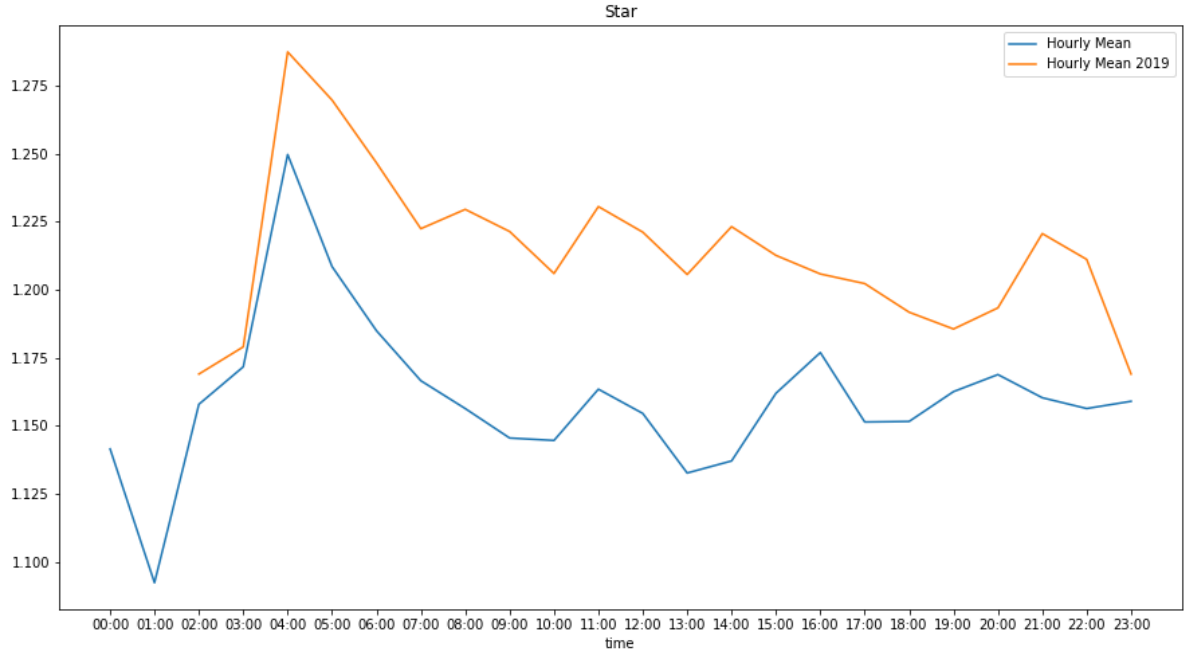
Here we can see some interesting points:

1) The "Total" is most time a little more expensive than "Star"
2) If the price is increasing, the "Total" is most time the first
3) If the price is decreasing, the "Star" is most time the first

Distribution of prices on a day

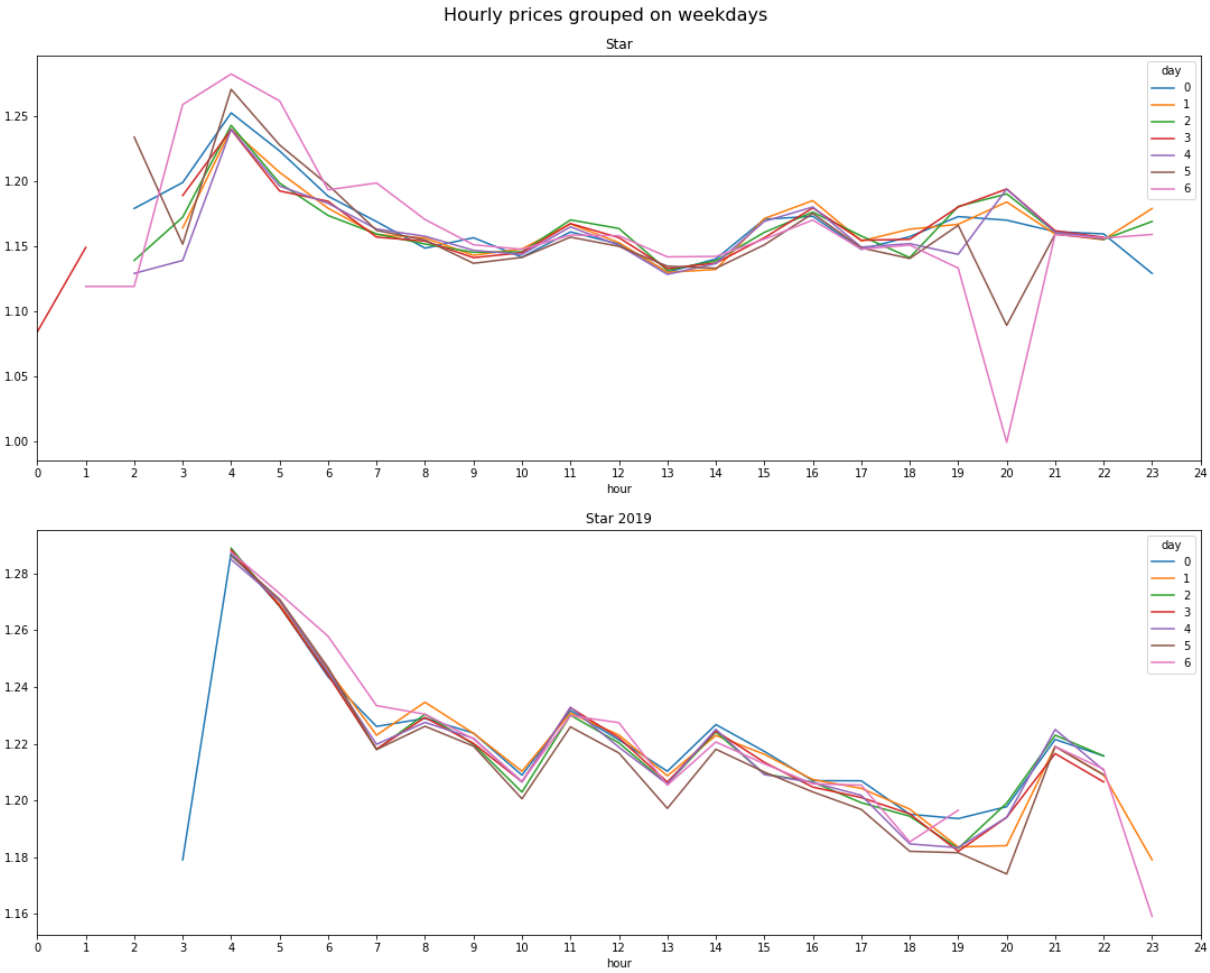
One thing I want to know is when it is a good time to fill the car during the day. Is there a good point in average? Therefore I sampled the raw data on hours and grouped them. To

have a current hourly mean, I've plotted the mean for the whole time and only for data from 2019.



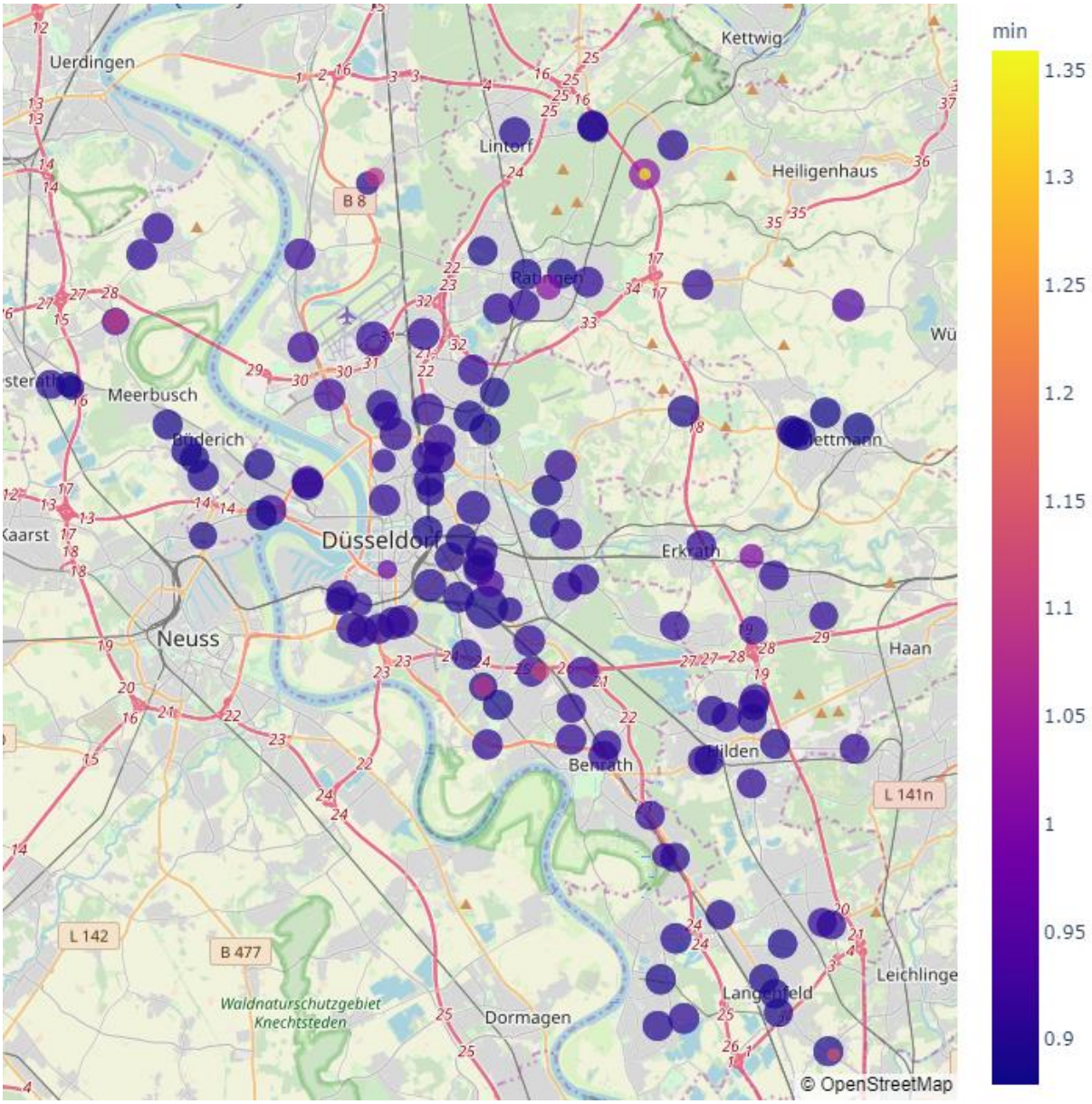
At first we can see, that the mean price for diesel raised in the last years, because the grouped values based on 2019 are higher. Also we can see that the price in last years was low on 1 pm but in the current year you'll get the lowest price mostly at 7 pm. The "Star" closes on 11 pm, therefore the last low is often not possible to use.

Is the distribution the same on every weekday? For answering this question I grouped the data on weekdays and then on hour.



Here we have a big difference in the years before 2019, there was Saturday and Sunday at 8 pm an obvious lowest price. In 2019 this is not more noticeable on Sundays but on Saturday it seems to be still the best time for cheapest prices.

Following that I want to get an overview of all the selected stations. Therefore I grouped the data by station and calculate the variance and the min price. This values I put in a scatter_mapbox from plotly, so you can see in the color the minimum price and the size of the circle shows you the variance in diesel price per station.



Like expected the highest minimum is on a station on our highway “A3” (the yellow circle in the top) and has also no big variance. In the notebook you can also zoom in and out, I used plotly and therefore you can have a closer look. (You will not see all the points if you use the data from github, there are only 8 stations included).

Data PreProcessing

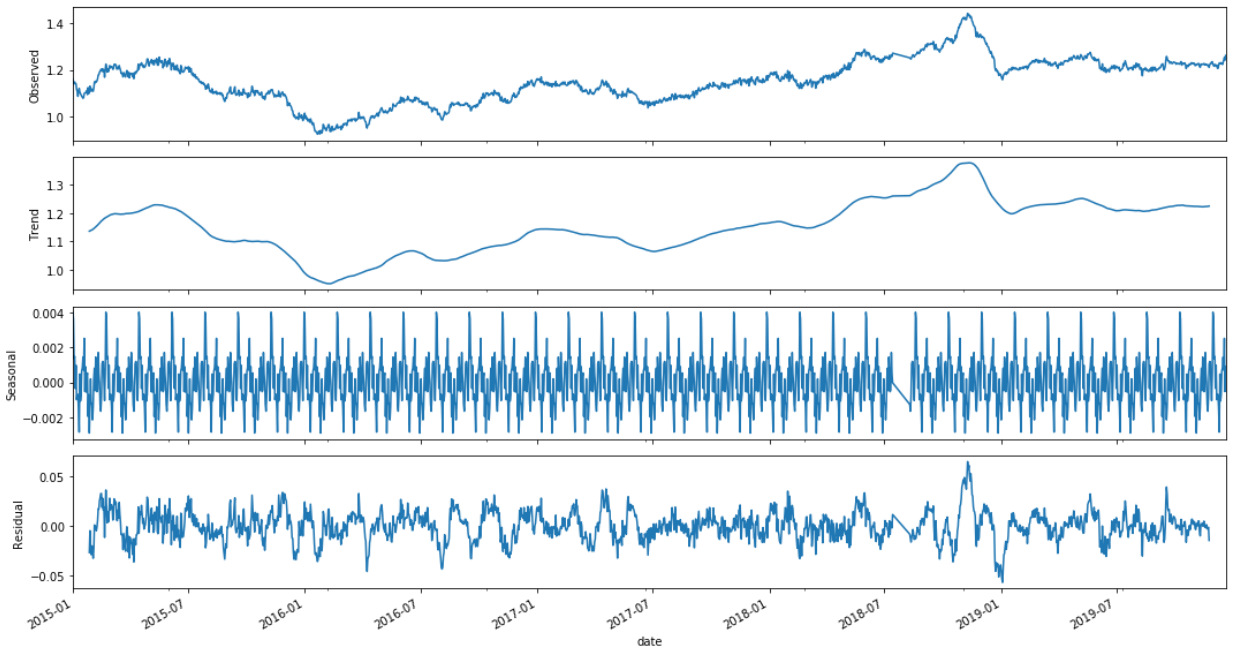
As mentioned in the data exploration, there were some wrong data in the dataset. I replaced the wrong data with np.nan and dropped the np.nan’s afterwards, because all following models cannot handle np.nan. Also I resampled the data to daily values and removed the localization out of the dateindex.

Implementation

To choose the right model i have a look in the data with different statistical methods to see if the data has seasonal component or trend inside. Therefore I use the statsmodel library (Seabold, 2010).

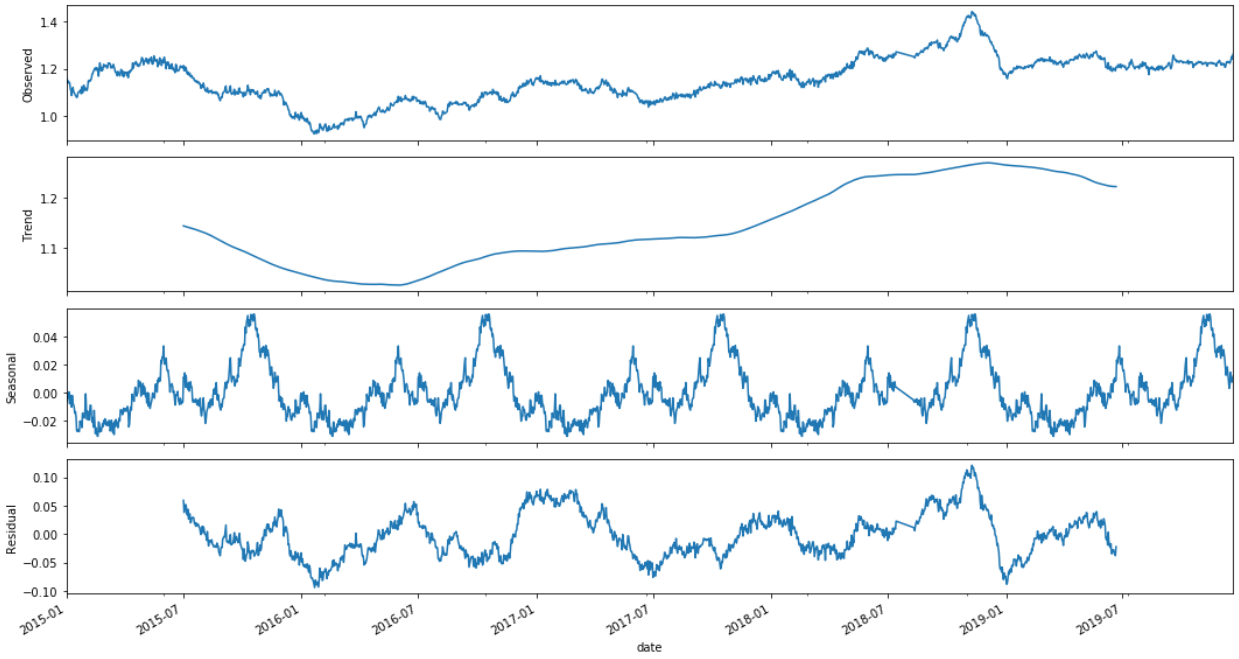
First I’ll will have a look at weekly seasonal component.

```
decompostion = sm.tsa.seasonal_decompose(star_day.dropna(), freq=52, model='additive')
```



There seems to be a weekly seasonal component inside. Let's look at a yearly component, too:

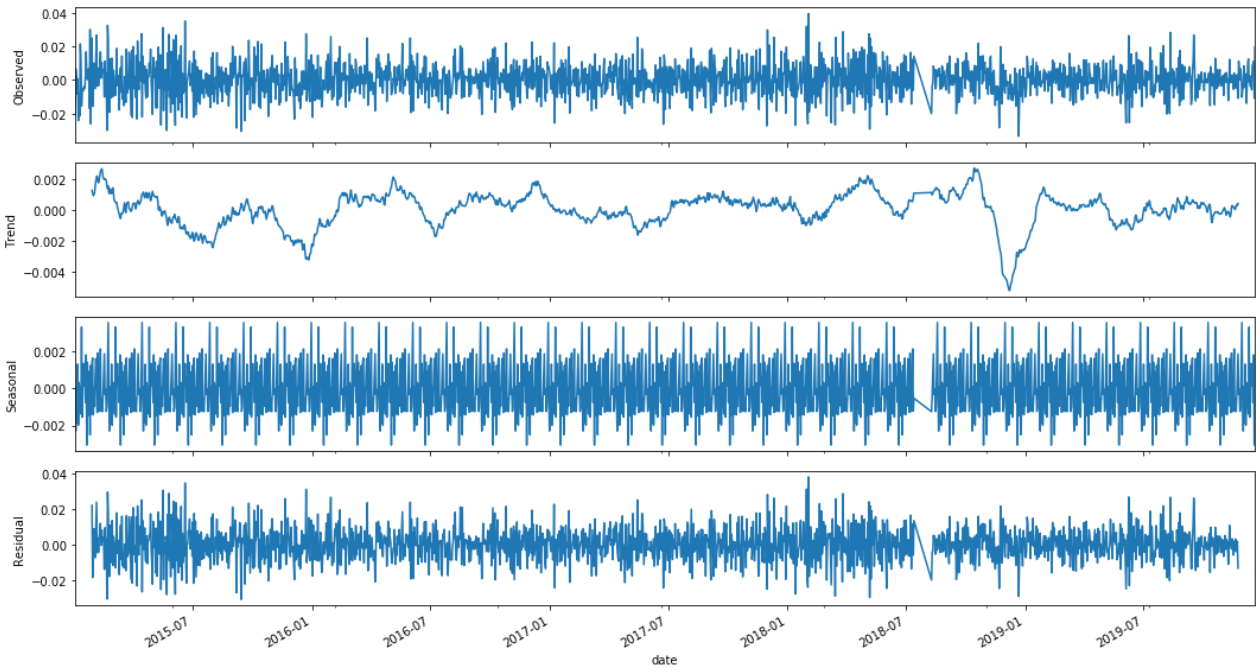
```
decompostion = sm.tsa.seasonal_decompose(star_day.dropna(), freq=365, model='additive'
```



In the decomposition plots we see seasonal component for weeks and also for year. For the model I decided to use weeks, therefore I'll go on with the frequency=52.

If we want to use SARIMAX, we have to get a stationary data. Therefore we will have a look at the difference.

```
decompostion = sm.tsa.seasonal_decompose(star_day.diff().dropna(), freq=52, model='additive')
```



The trend seems to be gone and is more randomly and the seasonal component is more or less random.

To really be sure if the data is stationary or not I use Adfuller test. ADF gives for the first difference a good result, p-values are close to zero meaning that the first difference is not more stationary.

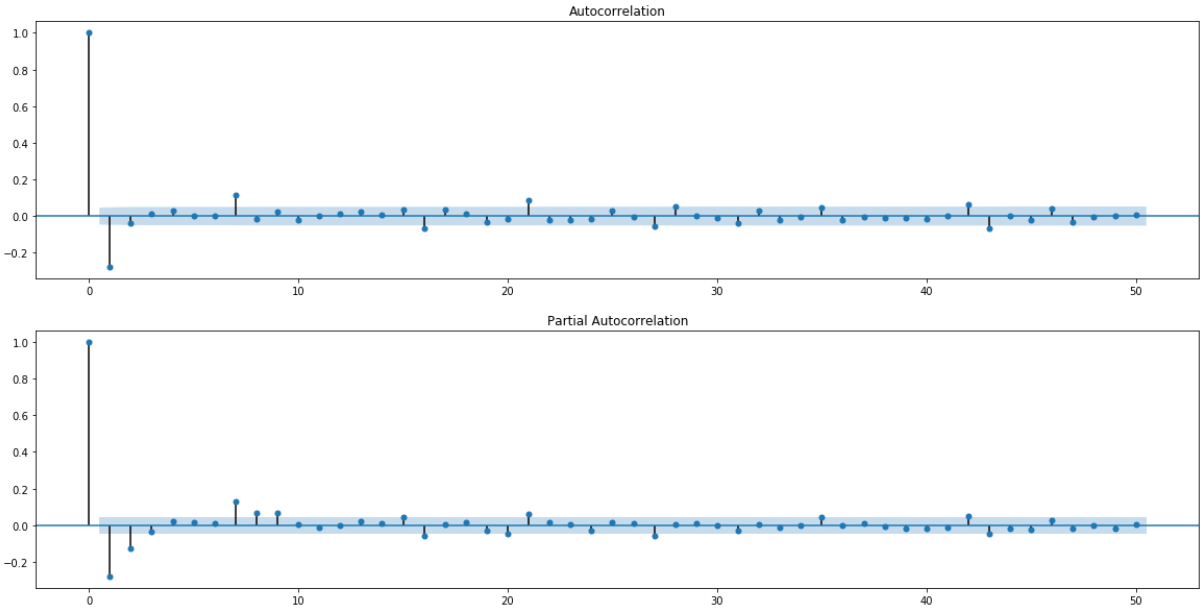
```
Results of ADF Test:
Test Statistic          -1.170910e+01
p-value                 1.503941e-21
Number of lags          8.000000e+00
Critical Value (1%)     -3.434031e+00
Critical Value (5%)     -2.863166e+00
Critical Value (10%)    -2.567636e+00
```

ARIMA Model

First I splitted the data in train und test data, I've used all data before 2019-01-01 as training data and the rest as test data. For better handling I removed also the localization out of the dateindex.

```
train = star_day['2015-01-01':'2018-12-31'].diesel
test = star_day['2019-01-01':].diesel
train = train.tz_localize(None)
test = test.tz_localize(None)
```

The auto correlation and the partial autocorrelation gives hints for the p and q for the ARIMA model. The Autocorrelation gives the q (MA) part and the Partial autocorrelation the p (AR) part of ARIMA.



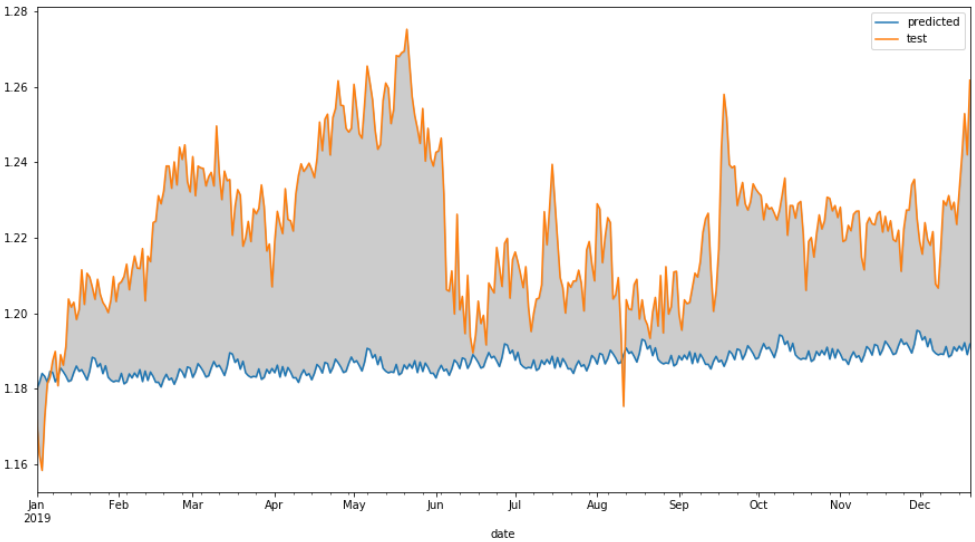
In the ACF/PACF plot from first difference you can see in ACF lag 1, (2) and 7 outside the benchmark and for the PACF lag 1, 2, (3), 7, 8, 9 outside the benchmark. As a rule of thumb you should start with the least amount of lags outside the benchmark (Burger, 2019). So i'll start with q=1.

order=(0,1,1), seasonal_order=(0,1,1,52)

Let's have a look if the forecast fits to the test data, therefore I'll calculated the mean squared error and the R2 Score.

```
ARIMA model MSE:0.0016330242769229013
R2 Score: -3.369946749624135
```

The MSE seems to be promising, but the R2 Score tells another story, it should be near to 1, if it is negative it is even more away, the forecasted values are not really good. In the next plot you'll see that the forecasted values are not really good in respect to the test data.

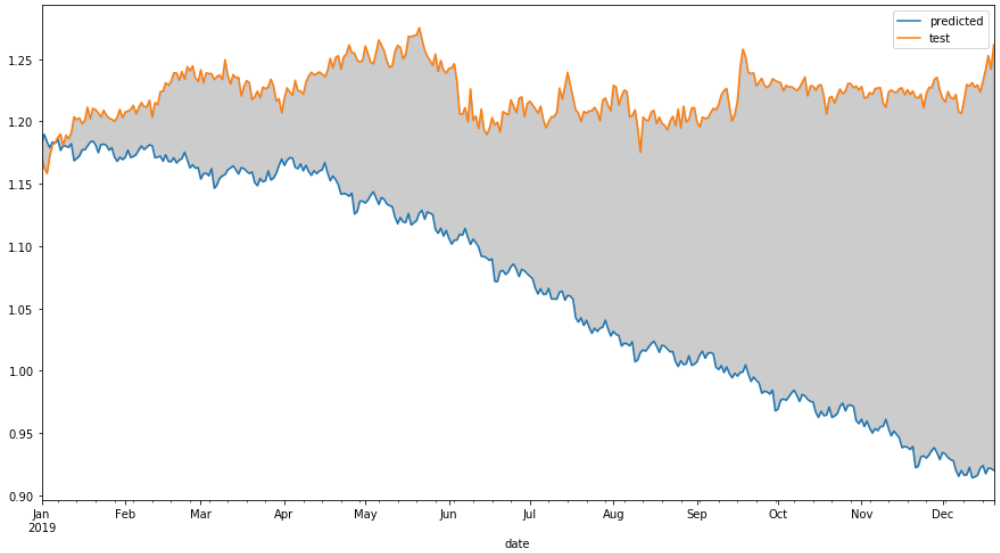


Let's see the next model, now we take the AR parameter p=3, too.

order=(3,1,1), seasonal_order=(3,1,1,52)

```
ARIMA model MSE:0.028432350988555427
R2 Score: -75.08451481243739
```

Here the R2 Score and also the MSE is even worse, so the plot should be also worse:



We can see, that the forecast is not good and the difference between forecast and real test values are getting greater the more you look in the future.

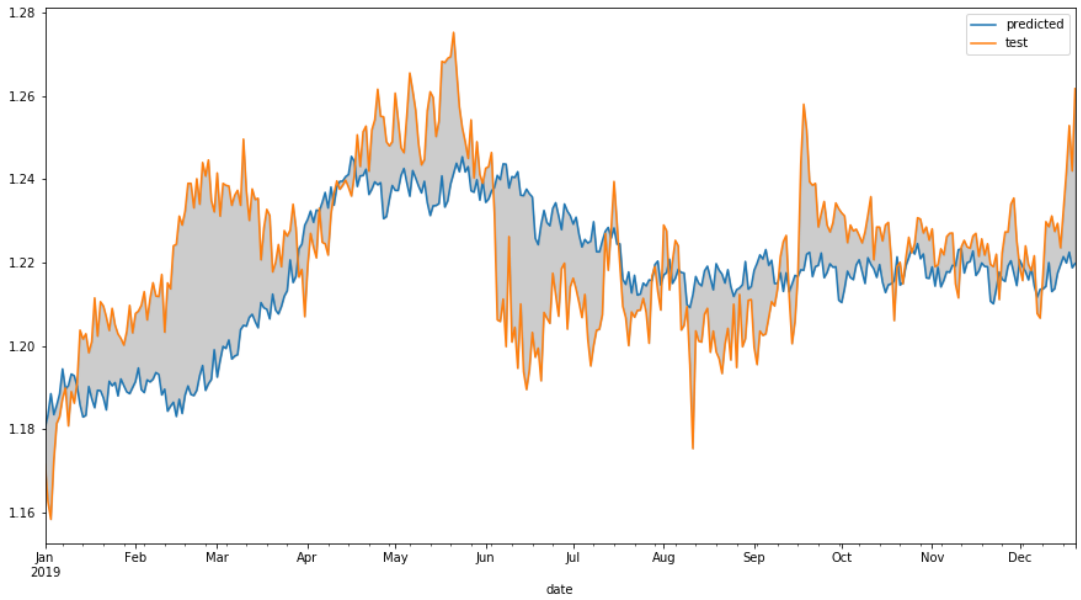
So let's see if we can get a better model if we increase q. In the ACF plot lag 2 is nearly on the benchmark:

order=(3,1,2), seasonal_order=(3,1,2,52)

This model gives really good values for the two metrics, also R2 score is still negative, but it is near zero.

```
ARIMA model MSE:0.0003974078621178883
R2 Score: -0.06345705932155798
```

So the plot should be also better:



So we have not all spikes predicted, but the prediction for the whole test data is not so bad. The prediction follows the trend over the year.

Let's see how the models fit if we use order=(3,1,3), seasonal_order=(3,1,3,52). Will it get better?

```
ARIMA model MSE:0.000742178944800891
R2 Score: -0.9860589418691452
```

Here we the R2 Score is getting worse, so the model is not better, the plot can be found in the jupyter notebook.

As last I'll want to test order=(2,1,2), seasonal_order=(2,1,2,52) and see how the metrics are for this model:

```
ARIMA model MSE:0.0010792673960818722
R2 Score: -1.8881022261164766
```

The metrics are even worse as from the model before, so I will not plot the prediction here, you'll can find it the jupyter notebook.

So the best model we have found for the fuel data has following parameters: order=(3,1,2), seasonal_order=(3,1,2,52). As mentioned above the model fits not every spike and has also reasonable differences in the forecasted prices in month February and March but is not so bad for predicting nearly the whole year.

Refinement

The Refinement was done during implementation of the model and was based on the ACF/PACF plots and the metrics MSE and R2 score. So I started with the least amount of lags outside the benchmark (q=1), because the second lag was not clearly outside. The first model does not fit well and therefore I added the AR part p=3 in the model. This model was also not really fitting and therefore I modified the MA parameter q=2 as the lag 2 could also be on the benchmark line and come to a good model.
The relevant implementation and refinement could be found in the notebook "Forecasting.ipynb".

LSTM Model

Next I've tried a LSTM model, because I've read some articles that describe the good prediction of LSTM for timeseries. This could be seen as an additional task, because training and evaluation of ARIMA should be satisfactory for capstone project. I am interested if LSTM could be better, so let's see.

The data has to be normalized for LSTM, so I use the MixMaxScaler from sklearn to have only values between 0 and 1. Also I created a function that build me an X and Y array with a given lookback and I will divide the data in train, validation and test.

```
# the values have to be normalized for LSTM
values = star_day['diesel'].values.reshape(-1,1)
scaler = MinMaxScaler(feature_range=(0,1))
scaled = scaler.fit_transform(values)
```

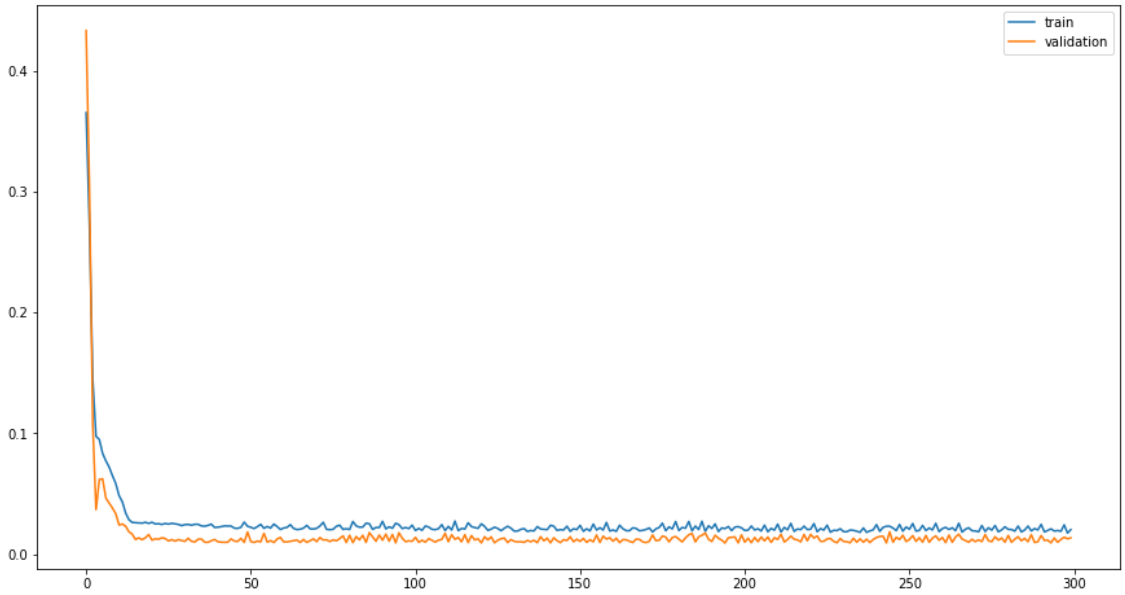
```
# use the same data for trainine up to 2018
train_size = len(star_day[:'2018-12-31'])
vali_size = 31 # let's take 1 month as validation set for fitting
test_size = len(scaled) - train_size
train, vali, test = scaled[:train_size,:], scaled[train_size:train_size+vali_size,:], scaled[train_size+vali_size:, :]
print(len(train), len(vali), len(test))
```

```
1435 31 323
```

I'll try to use the following parameters:

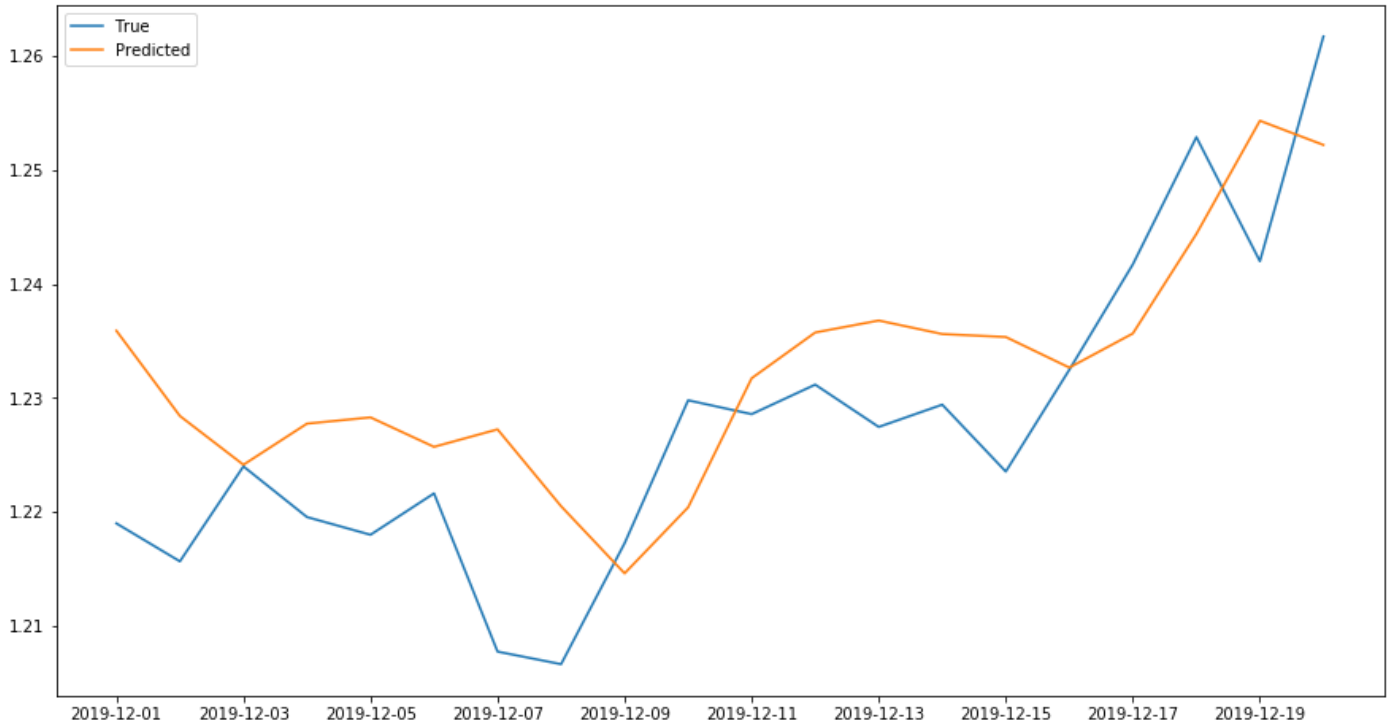
```
# build a LSTM model
model = Sequential()
model.add(LSTM(50, input_shape=(trainX.shape[1], trainX.shape[2]), return_sequences=True))
model.add(Dropout(0.1))
model.add(LSTM(50))
model.add(Dense(1))
model.compile(loss='mae', optimizer='adam')
history = model.fit(trainX, trainY, epochs=300, batch_size=100, validation_data=(valiX, valiY), verbose=0, shuffle=False)
```

The training is very fast and plotting the loss and val_loss of the training epochs are very good:



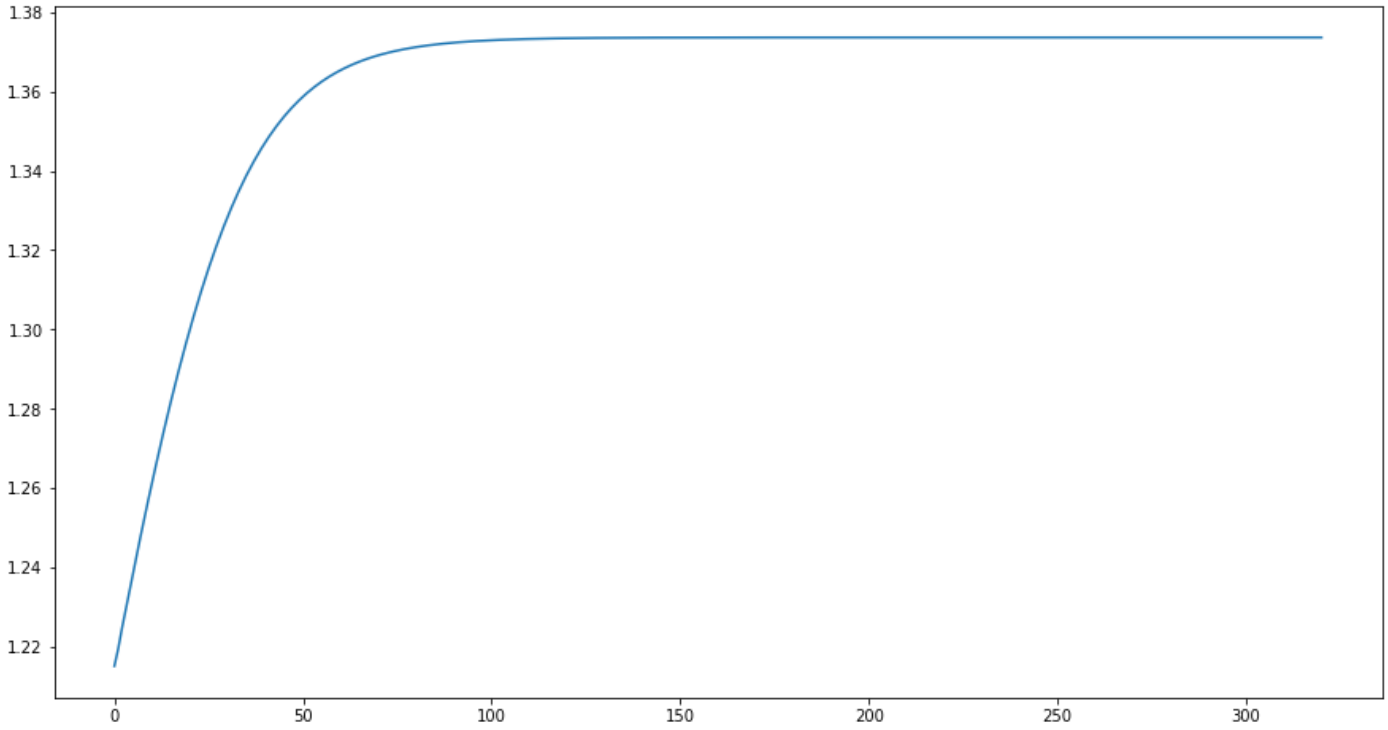
And the mean squared error for the test data is 0.00010629034344355478 and R2 Score is also really good with 0.658041508369376. Seems to be really good fitting of this model. Let's see a plot of predictions of train, validation and test data:

In the first view the prediction of the test data on the right side in red seems to be very similar to the original data. You can see, that not every spike is predicted, but the prediction of test data is very good. We will have a closer look inside, so i'll choose only the last days of the test data:



The prediction of LSTM is more a "follower" than a real forecast. We see that wrong predictions are corrected in the next prediction because we've used original data for the next prediction again, as we use the test data set. When the prediction will be done with more days in the future, so let's say we'll start with the last data set predict one value and use the prediction as input for the new prediction, we'll see, that LSTM has also problem. Therefore I'll adapt the function `predict_sequence_full` from (Raval, 2017). This function

starts with the first values of the original test data, but will use in further steps the own prediction as input. If we use this function with our test data, we get the following plot.



So the model is going to an limit and not following the original test data. So the model is good on real world values, but not on generated next values, means it overfits. But especially about this behavior of LSTM you can write an own article and therefore could not more handled here in detail.

Model Evaluation and Validation

For the ARIMA model I've used different parameters for p and q and calculated the MSE and the R2 score. Here are the values:

Model	Parameters	MSE	R2 Score
0	p=0, q=1	0.0016330242769229013	-3.369946749624135
1	p=3, q=1	0.028432350988555427	-75.08451481243739
2	p=3, q=2	0.0003974078621178883	-0.06345705932155798
3	p=3, q=3	0.000742178944800891	-0.9860589418691452
4	p=2, q=2	0.0010792673960818722	-1.8881022261164766

The model 2 has the best values for both metrics. The MSE is smallest and the R2 Score should come to 1 the better the model is. So for the ARIMA models tested, the parameters order=(3,1,2), seasonal_order=(3,1,2,52) gives the best result. And the plot for forecasting the test data is also not bad if we see the whole year.

In comparison the LSTM seems to be better if you just have a look on the metrics MSE and R2 Score: MSE 0.00010629034344355478 and R2 Score: 0.658041508369376. And also if you have a look at the plots as prediction and real value of test data is really close together.

But as we have seen, it has good values on validation and also test data, but the forecast for some days in the future is not really good and go to some limit instead of following the data points.

Justification

ARIMA work good on this data and the metrics are promising, but in general I come to the point, that forecasting a time series like fuel price is not as easy. The data has a trend inside and also a seasonal aspect. I handled the trend and the non stationarity of data with differencing the data points. To handle the seasonal aspect I used the SARIMAX model and added a seasonal_order part.
The additional model LSTM shows good metrics on the test data and therefore seems to be better than the ARIMA in first view. But if you have a look on really forecasting you'll see that LSTM approximate to a limit instead of really forecasting. So ARIMA does here a better job.

Conclusion

I'll used the diesel price data freely available from (Tankerkoenig, www.tankerkoenig.de, 2020)to make a daily forecast of the diesel price. Therefore i first load the data, removed wrong data and extract only some data out of the zip code area of 40xxx and reduced it further to make it github compatible. I made some interesting analysis and plots on the data, for example the average lowest price on a daily basis, before coming to the forecast modelling.

The raw data showed trend and seasonality and was not stationary. So i used difference for the ARIMA model to have stationarity. The Augemented Dickey Fuller (ADF) Test gave the result, that the 1st difference of the data is stationary. With the help of ACF and PACF we found some hints for the parameters and tested different models. We found some model, that have overall promising metrics, but if you have a closer look the prediction differs extremely on many days. Also I've found some problems with this package.

Then i've looked in LSTM as model, the training was really fast and the first view on the test data set was promising. If you have a closer look, you see the predictions are "following", so it could be, that a prediction for next day is not so good, but in average the prediction is good. Also I showed that the prediction for many days in the future is not good and leads to a border.

Reflection

This project was very interesting to work with time series data. To get the data in the right format was one aspect and to handle times in different algorithms was also challenging. Also I found it useful to hava a look at to different models for forecasting, even that one model was challenging enough. For both models exists plenty of examples, but nevertheless it was ambitious to get the data in the right format and create a good model. At some point and many calculations later I've came to conclusion, that there are some bugs in ARIMA

model implementation, because I cannot calculate many different models in a row in one notebook. I was forced to restart the kernel many times.
One finding is, that many of the examples that you can find in the internet don't have a look at the quality of really forecasting and only looks at the forecast for the test data. But as shown with the LSTM model this is not right.

And for the forecasting of the fuel price in general I've come to the conclusion, that the price cannot be forecasted as easy with only historic data. I took only daily means and it was hard to predict, but if you take into account to predict the price on hourly base it is even more difficult.

Improvement

One possibility to get ARIMA better is to use other parameter for p, d, q. For example you can use the auto_arima function out of pmdarima to have good starting values. Another possibility could be to use another method to come to stationary time series, for example subtracting the rolling mean.

For LSTM we have the possibility to take more lookbacks in account or to combine data of some similar stations. Especially the lookback could be a possibility to have a better forecast for some days in the future. Also we could use gridsearch method to find better model parameters.

In general the fuel price seems to be hard to predict(like the stock prices), the ARIMA model does a good job in general but is not really well predicting. And the LSTM model tends to overfitting, means the original time series and also the test data is well predicted, but if you want to look more in the future, LSTM is not working well. Also it seems, that there are many more features outside the data that influence the price and therefore the forecasting with LSTM could be improved considering more other features.

Bibliography:

Burger, M. (2019, Jun). Retrieved from Pluralsight: "Mining Data from Time Series": https://app.pluralsight.com/id?redirectTo=/library/courses/3fc3207c-3659-4a68-9451-8ed0ae0e2095

Raval, S. (2017, Feb). Retrieved from How-to-Predict-Stock-Prices-Easily-Demo / lstm.py : https://github.com/llSourcell/How-to-Predict-Stock-Prices-Easily-Demo/blob/master/lstm.py

Seabold, S. a. (2010). *"statsmodels: Econometric and statistical modeling with python." Proceedings of the 9th Python in Science Conference.* Retrieved from http://conference.scipy.org/proceedings/scipy2010/pdfs/seabold.pdf

Tankerkoenig. (n.d.). Retrieved from History Data, tankerkoenig-data: https://dev.azure.com/tankerkoenig/_git/tankerkoenig-data

Tankerkoenig. (2020). Retrieved from www.tankerkoenig.de: http://www.tankerkoenig.de/