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. 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 basis.

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.

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

Problem statement

Does the data have other useful insights?
When it is good time for refueling?
Is there a possibility to forecast the fuel price in Germany?
Could I find a model to forecast the fuel price for next day?

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.
To answer the questions I'll go through some visualizations and analysis. For building and fitting the right model, I'll use Adfuller test and autocorrelation functions.

Data Reading

Because of the huge amount of data, I've decided to read year per year and extract only a part from it. I used all stations starting with zip code 40.

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

And read afterwards only data of stations, that are in this list. Also I remove some wrong data I've found before. There are some prices with -0.01 € in the data and that is no real values. So I set all wrong prices (< 0.5€ because it wasn't below this in the last 20 years) to np.nan. Than I removed also any wrong data I'll found, see the jupyter notebook for details. Also I set the date field to datetime64 to better handling the data in later steps.

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2629941 entries, 74 to 240632
```

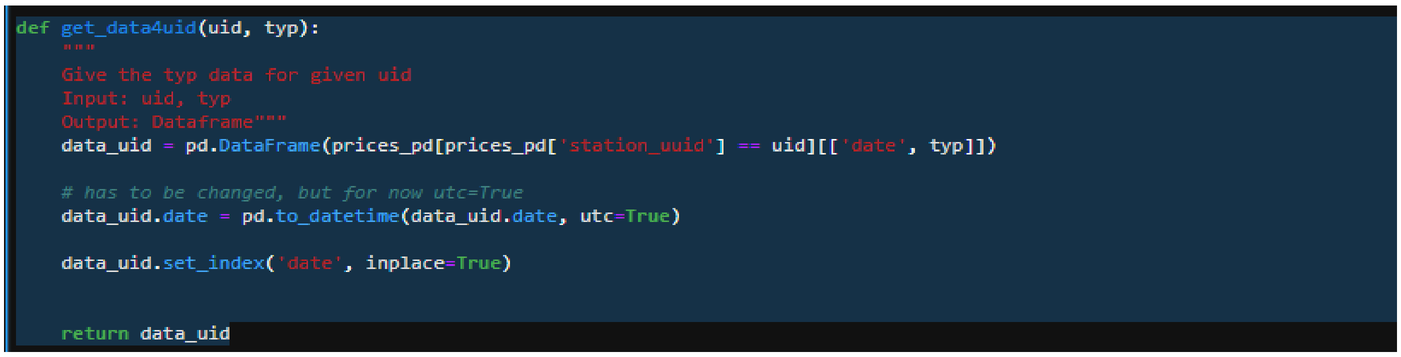
```
Data columns (total 5 columns):
date            datetime64[ns, UTC]
station_uuid    category
diesel          float64
e5              float64
e10             float64
dtypes: category(1), datetime64[ns, UTC](1), float64(3)
memory usage: 105.3 MB
```

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.

```
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']]
```

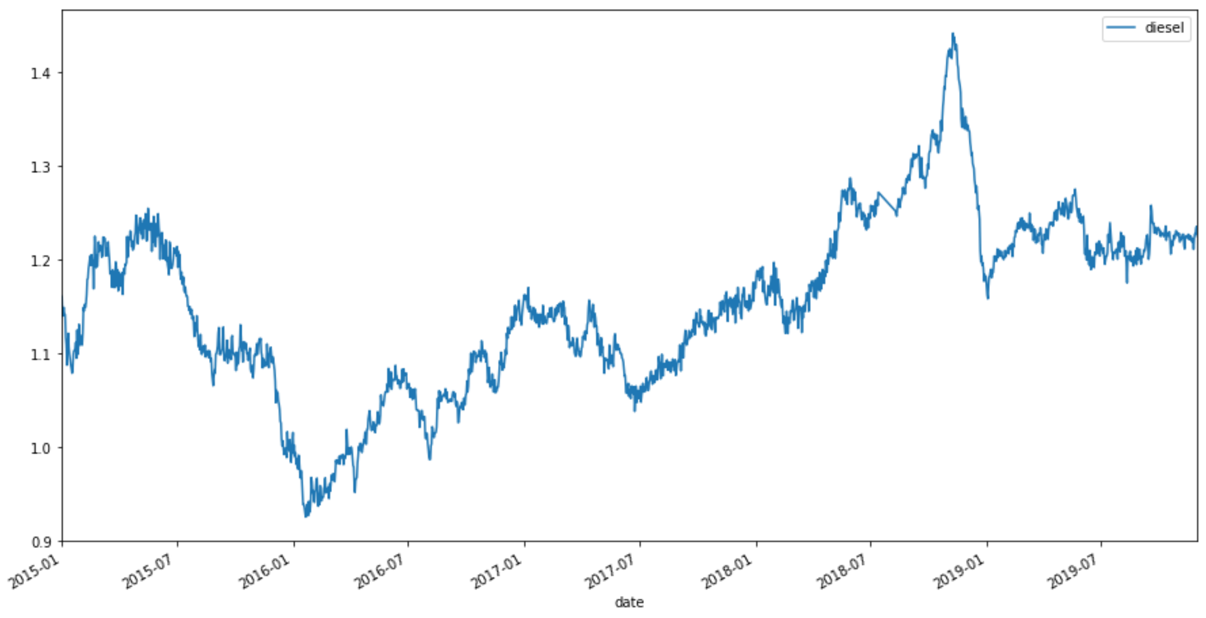
Data Preparation and Analysis

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:

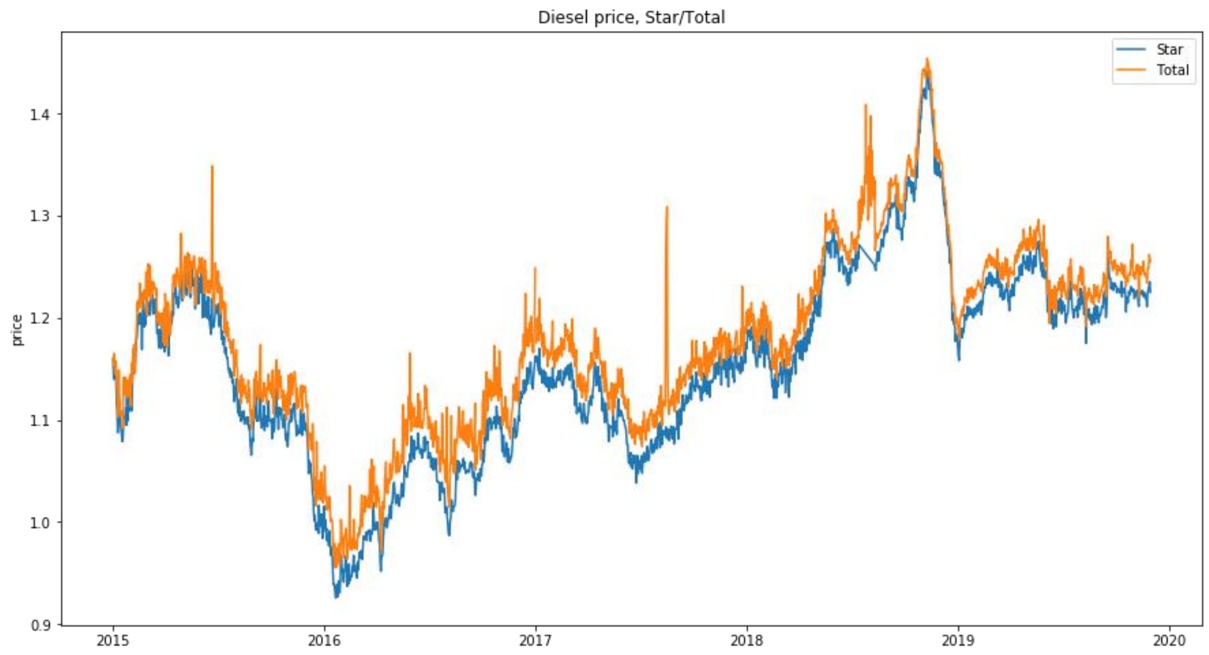


```python
def get_data4uid(uid, typ):
    """
    Give the typ data for given uid
    Input: uid, typ
    Output: Dataframe"""
    data_uid = pd.DataFrame(prices_pd[prices_pd['station_uuid'] == uid][['date', typ]])

    # 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
```

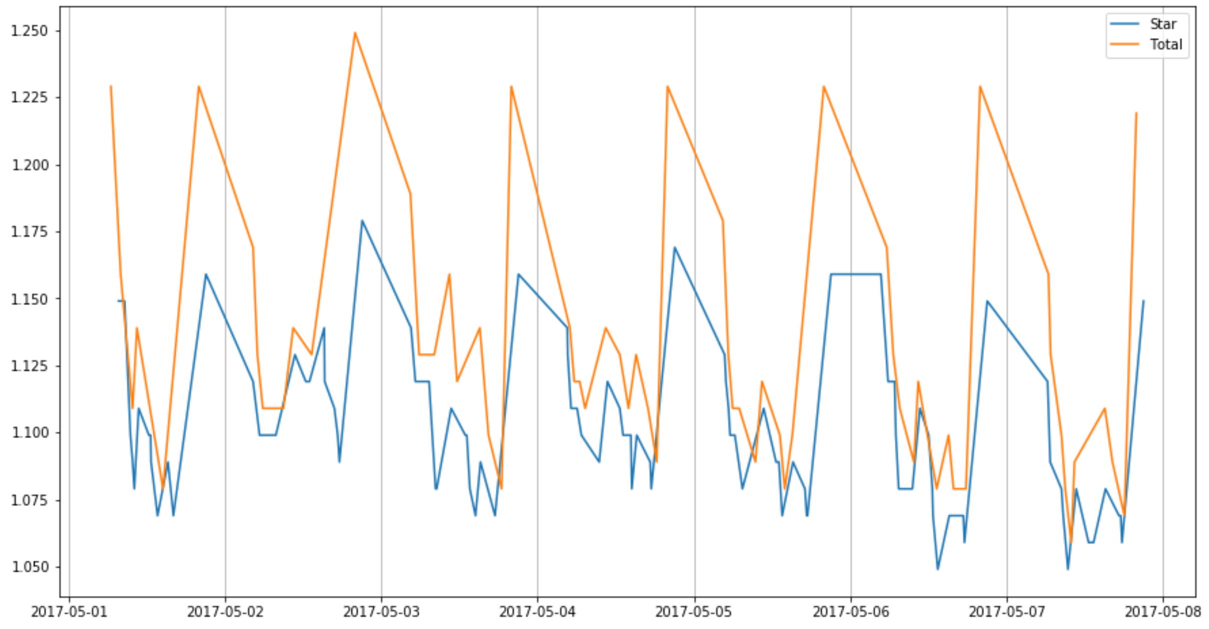
I resampled the data to daily mean values and have a first look in the data of one station for “diesel”:



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:

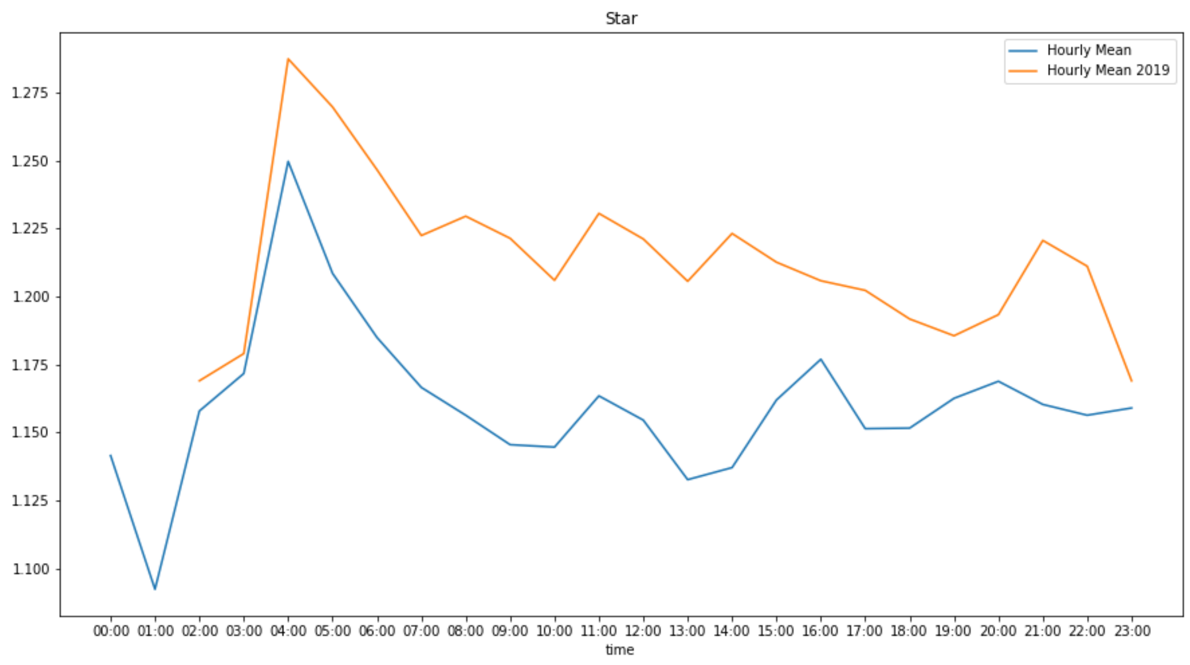


Here we can see some interesting points:

1) The “Total” is most time a little expensier 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

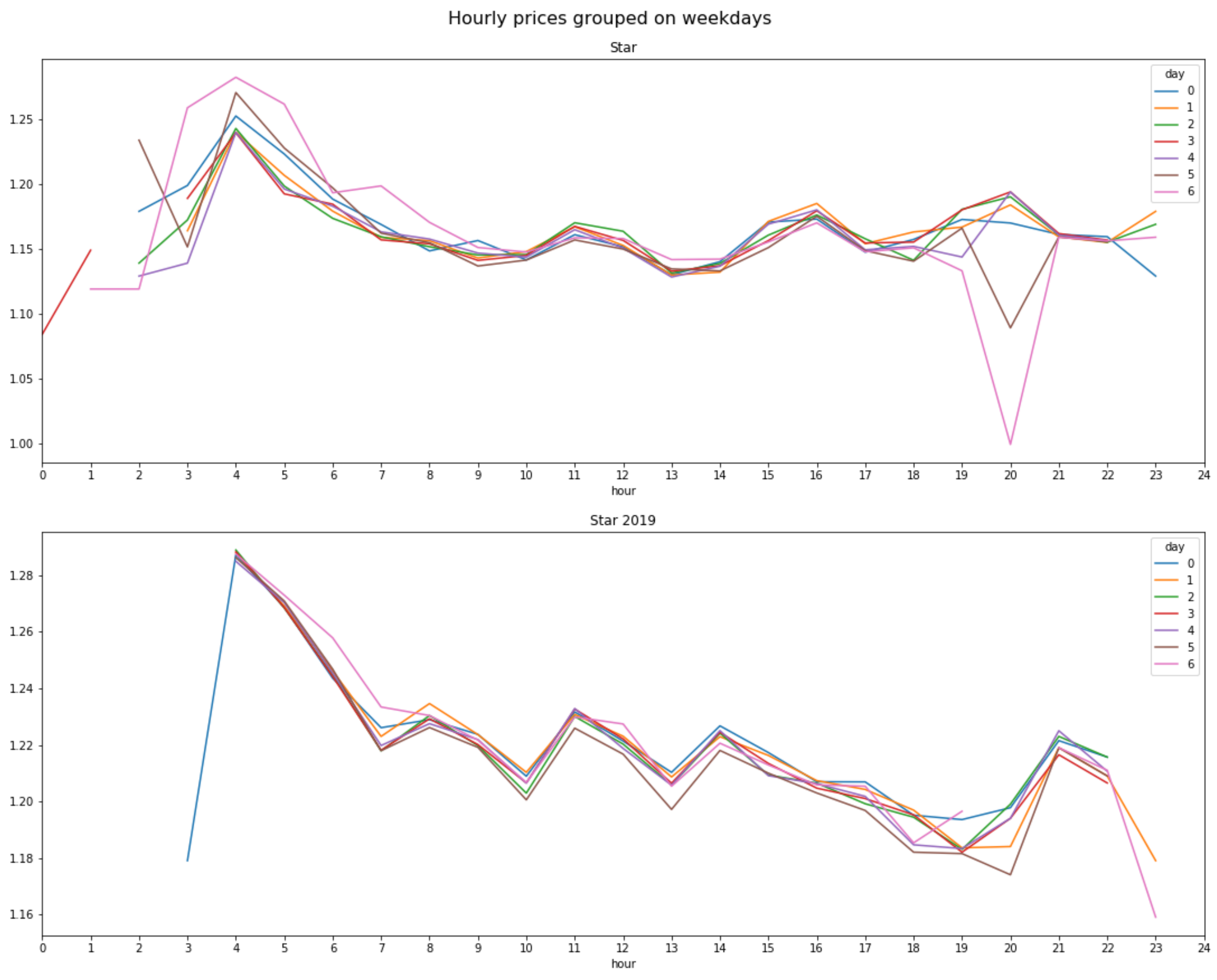
Distribution of prices on a day

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.



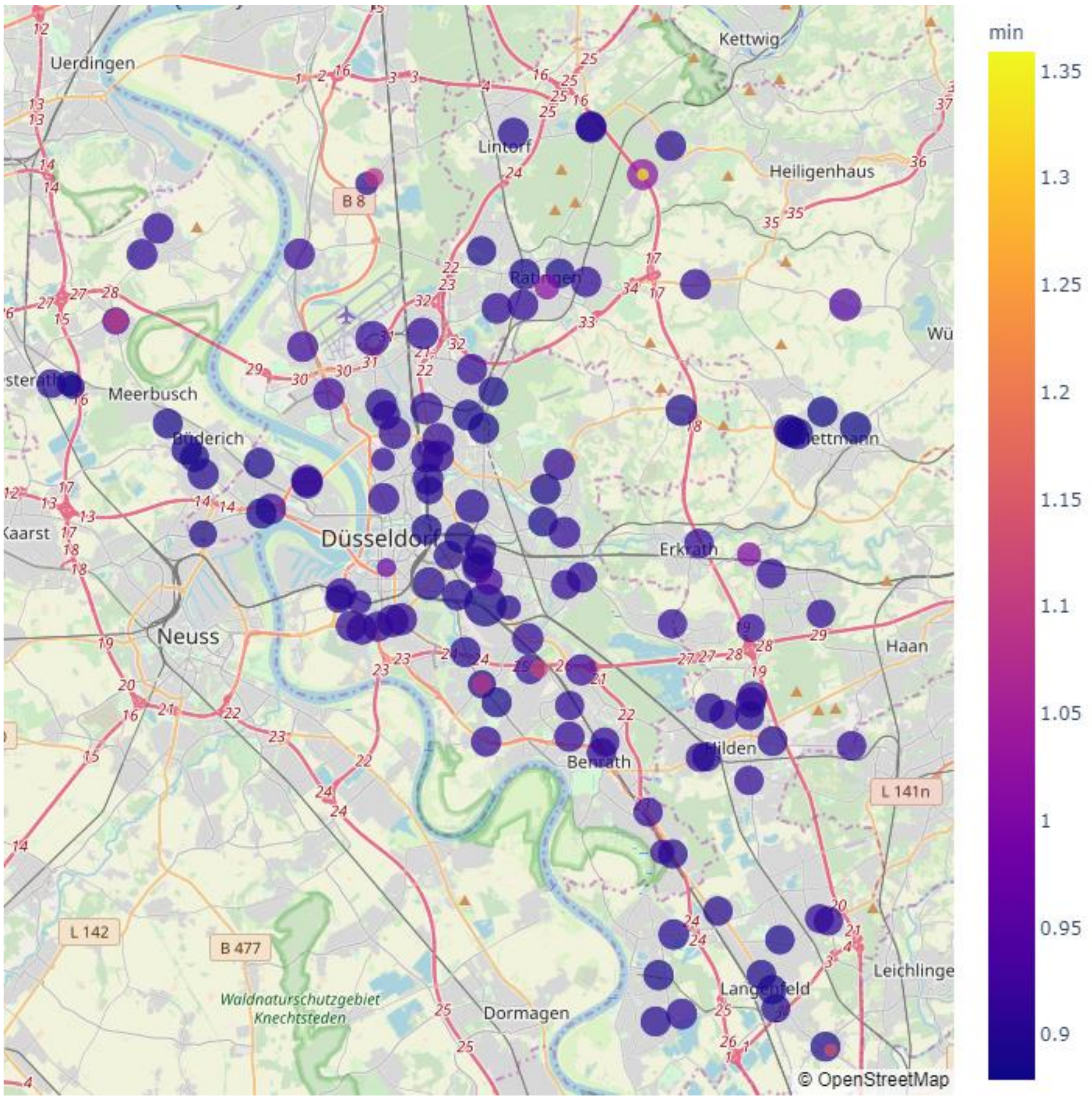
At first we can see, that the mean price for diesel raised in the last years, because the grouped values based on 2019 are higher. Also we can see that the price in last years was low on 1 pm but in the current year you’ll get the lowest price mostly at 7 pm. The “Star” closes on 11 pm, therefore the 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.



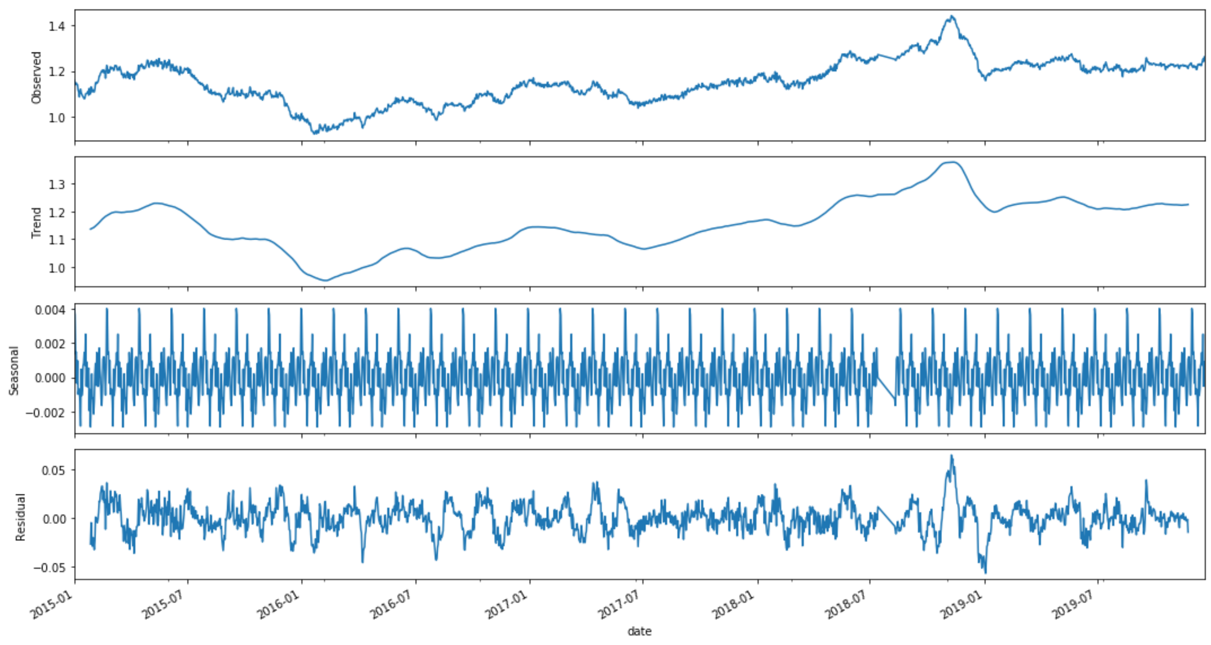
Like expected the highest minimum is on a station on our highway “A3” (the yellow circle in the top) and has also no big variance. In the notebook you can also zoom in and out, I used plotly and therefore you can have a closer look. (You will not see all the points if you use the data from github, there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

Forecasting

Now I’ll have a look in forecasting the fuel price, therefore exists many models: SARIMAX, Prophet, LSTM and so on. I’ll have further a closer look to SARIMAX and LSTM.

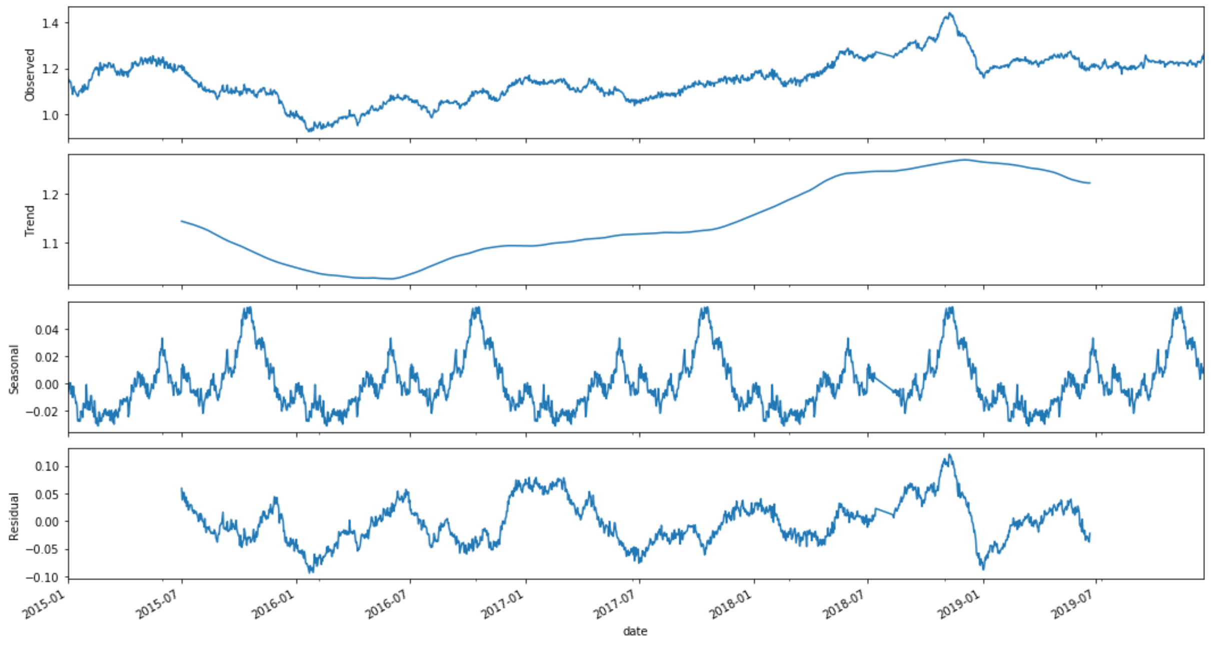
To choose the right model we have a look in the data with different statistical methods to see if the data has seasonal component or trend inside. First I’ll will have a look at weekly seasonal component.

```
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:

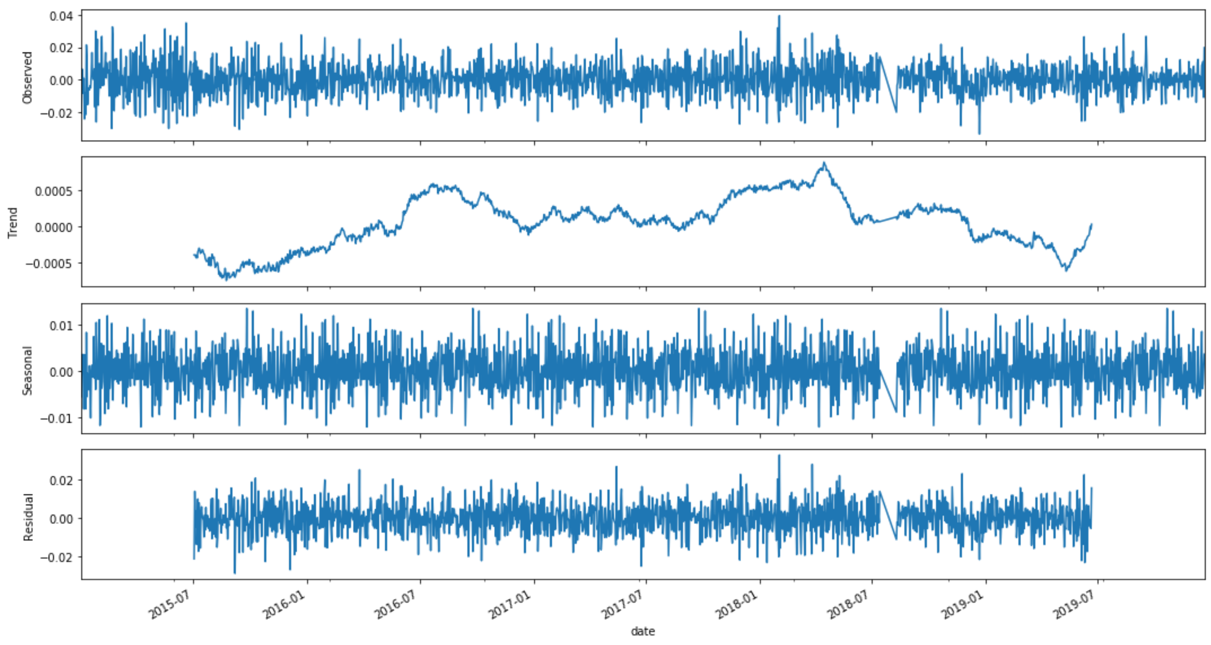
```
decompostion = sm.tsa.seasonal_decompose(star_day.dropna(), freq=365, model='additive'
```



In the decomposition plots we see seasonal component for weeks and also for year.

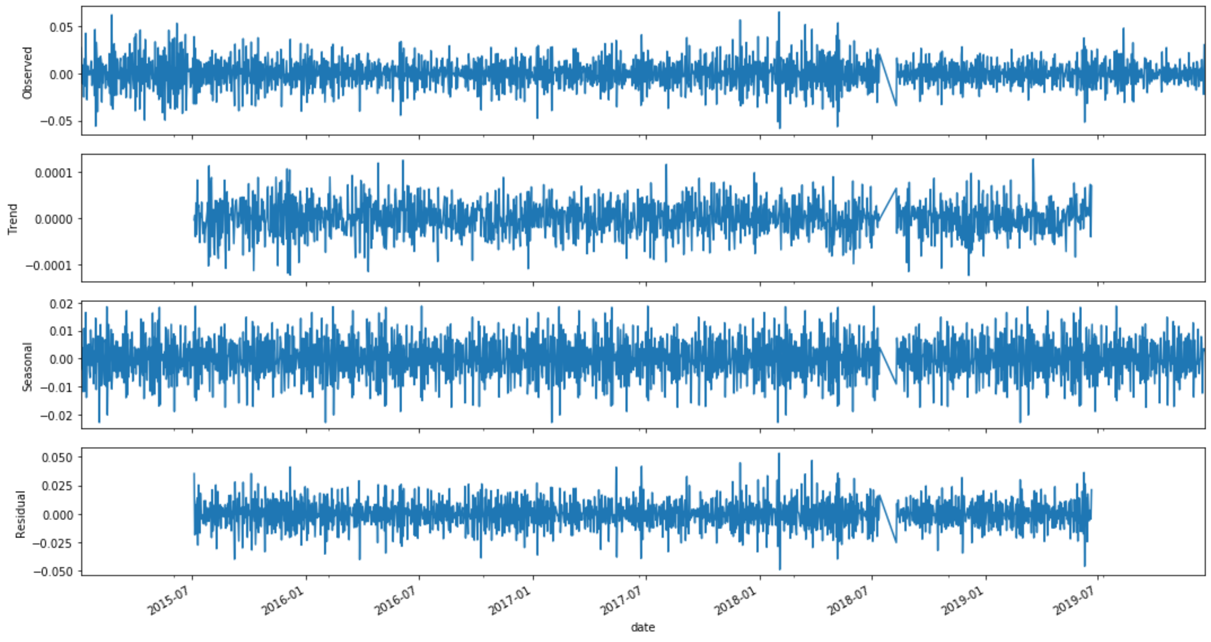
If we want to use SARIMAX, we have to get a stationary data. Therefore we will have a look at the difference.

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



There is still a trend in the first difference, but the seasonal component looks like random. Let’s see what is if we take the difference a second time.

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



The trend is gone and therefore this should be not more stationary.

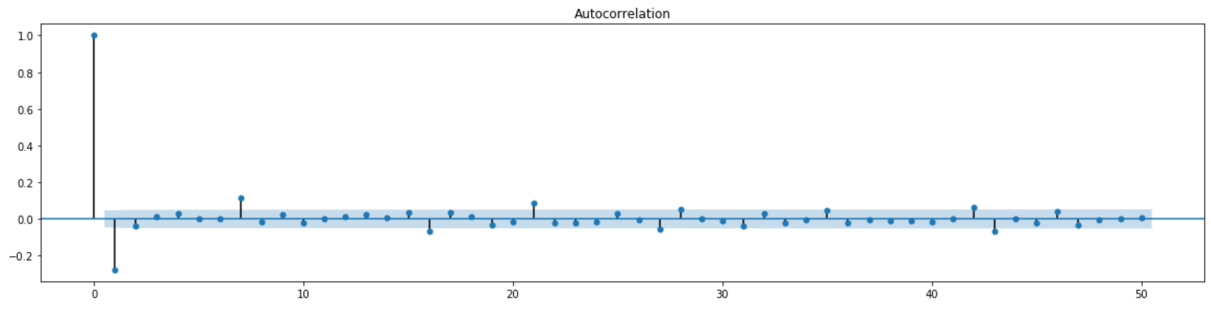
To really be sure if the data is stationary or not I will use Adfuller test. ADF gives for the first difference a good result, p-values are close to zero meaning that the first difference is not more stationary.

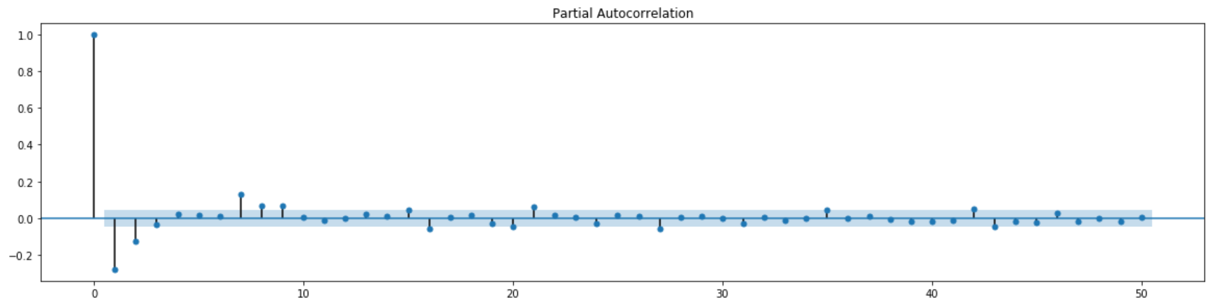
ARIMA Model

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.





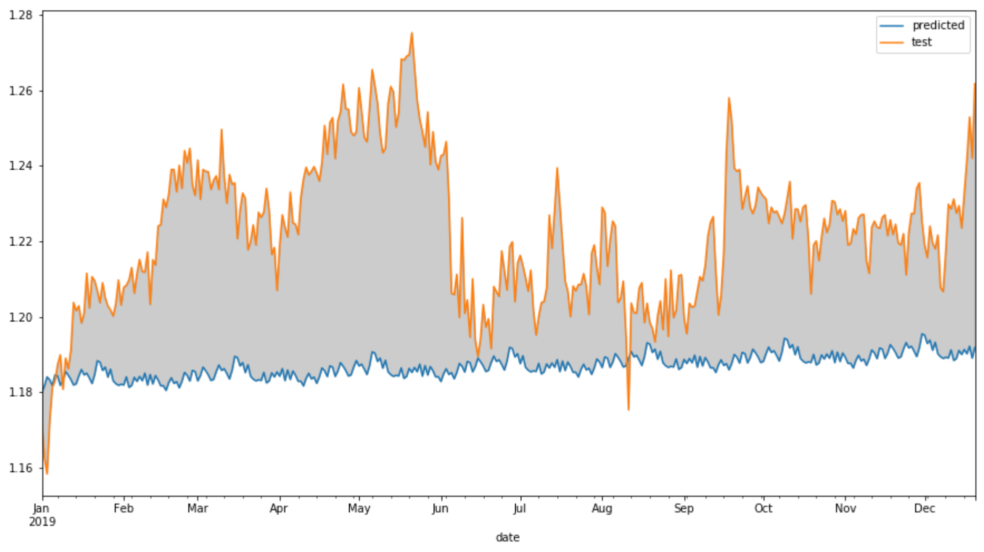
In the ACF/PACF plot from first difference you can see in ACF lag 1 and 7 outside the benchmark and for the PACF lag 1, 2, (3), 7, 8, 9 outside the benchmark. As a rule of thumb you should start with the least amount of lags outside the benchmark. So i'll start with q=1.

```
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.

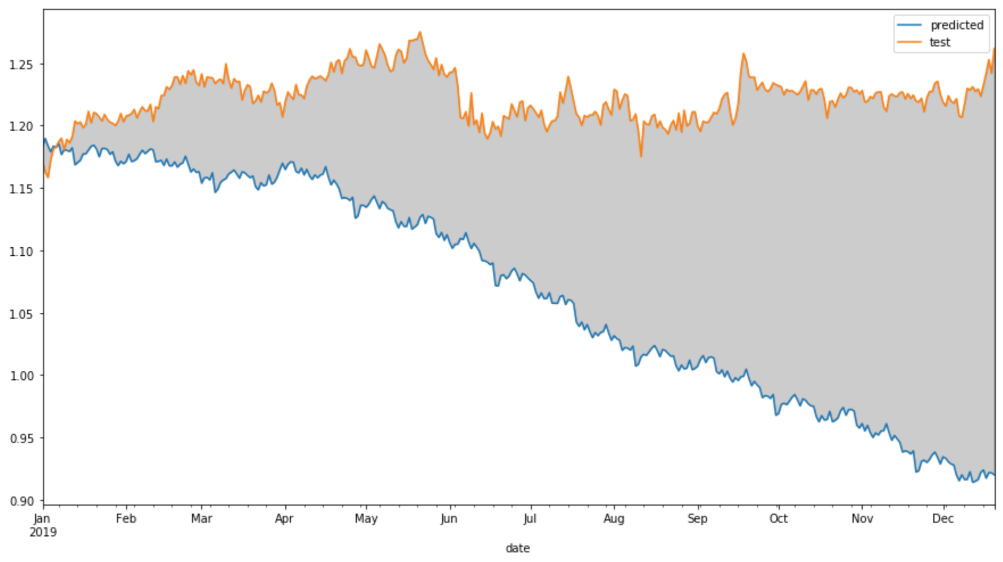


Let’s see the next model, now we take the AR parameter p=3, too.

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

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

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



We can see, that the forecast is not really good and the difference between forecast and real test values are getting greater.

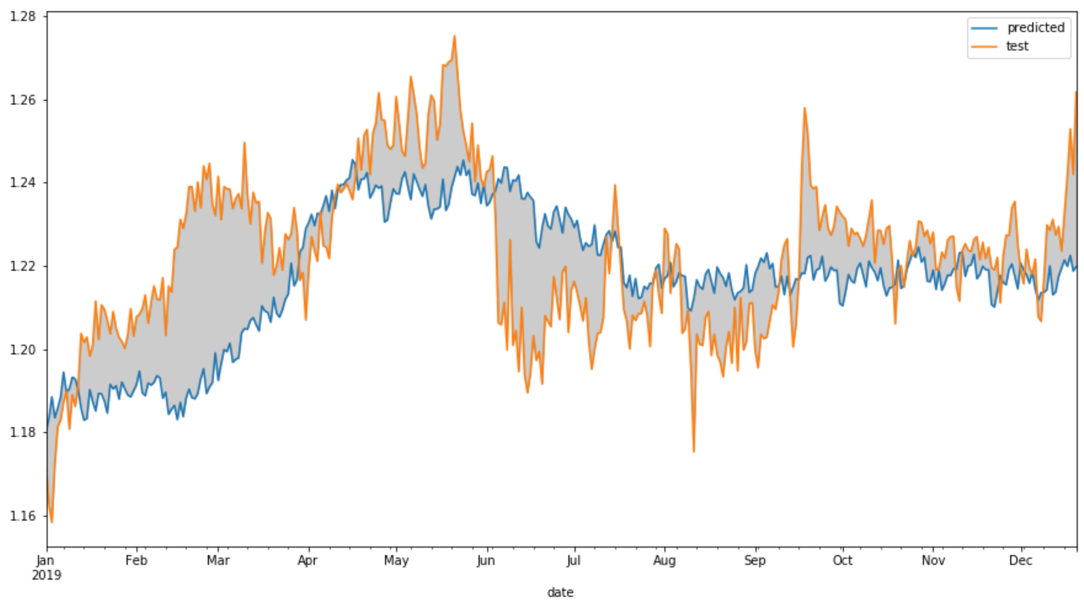
So let’s see if we can get a better model

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

This model gives really good values for the two metrics.

```
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 curve is not so bad. The prediction follows the trend up and down.

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:0.0007421789448008891
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.

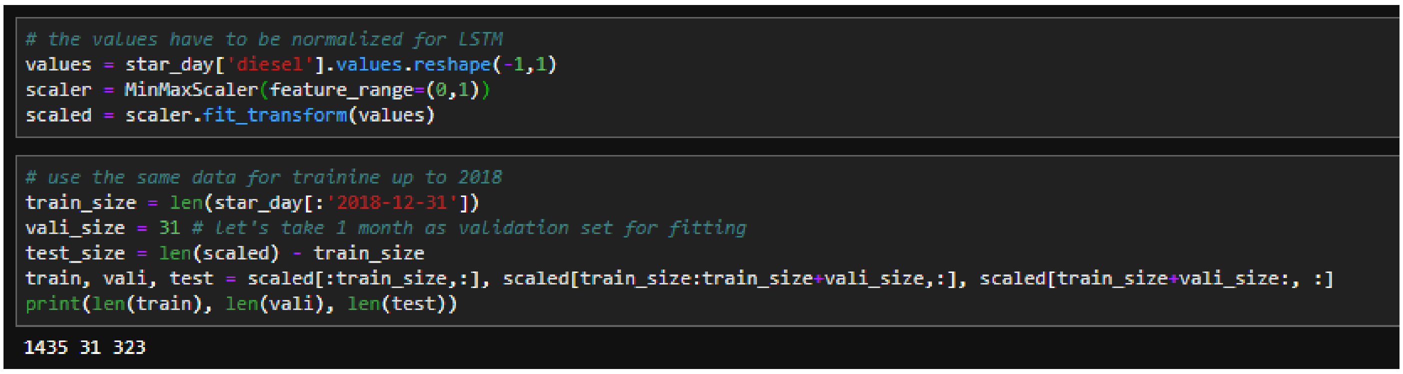
So the best model we have found for the fuel data has following parameters: order=(3,1,2), seasonal_order=(3,1,2,52). As mentioned above the model fits not every spike and has also big differences in the forecasted prices in month February and March but is not so bad for the whole test data.

In my notebook I've got some problems with the ARIMA model, it seems to have small bugs (memory leaks?) inside, because I'll cannot calculate 3 different models in a row without run into an error. After restarting the Kernel, I could calculate another model. That said I left it to the above models and have a look to another model.

LSTM Model

Next I’ve tried a LSTM model, because I’ve read many articles that describe the good prediction of LSTM for timeseries. 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.

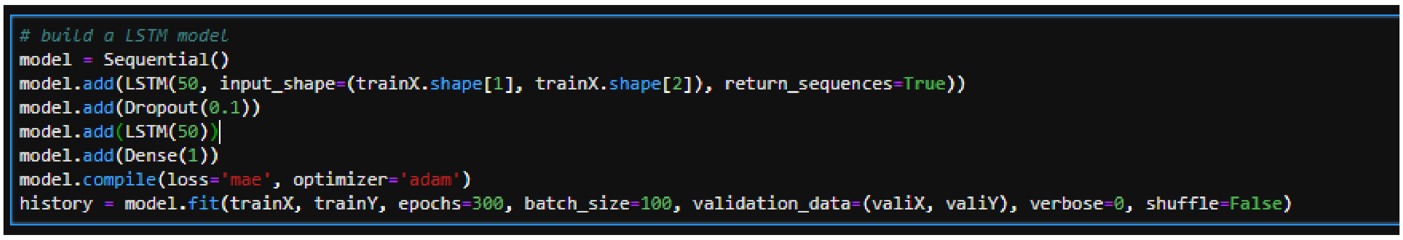


```
# the values have to be normalized for LSTM
values = star_day['diesel'].values.reshape(-1,1)
scaler = MinMaxScaler(feature_range=(0,1))
scaled = scaler.fit_transform(values)
```

```
# 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))
```

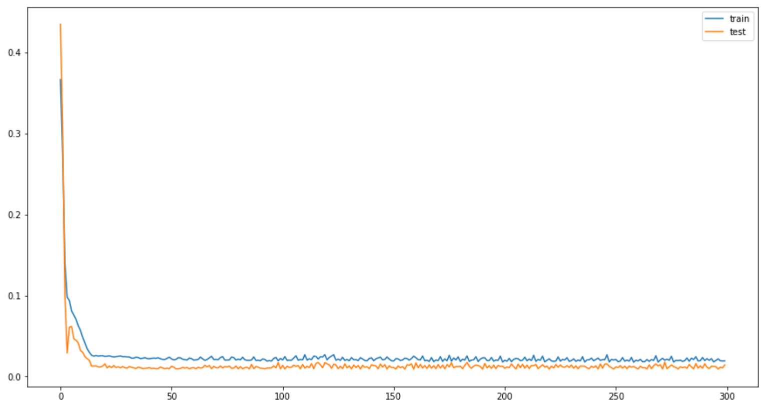
```
1435 31 323
```

I’ll try to use the following parameters:

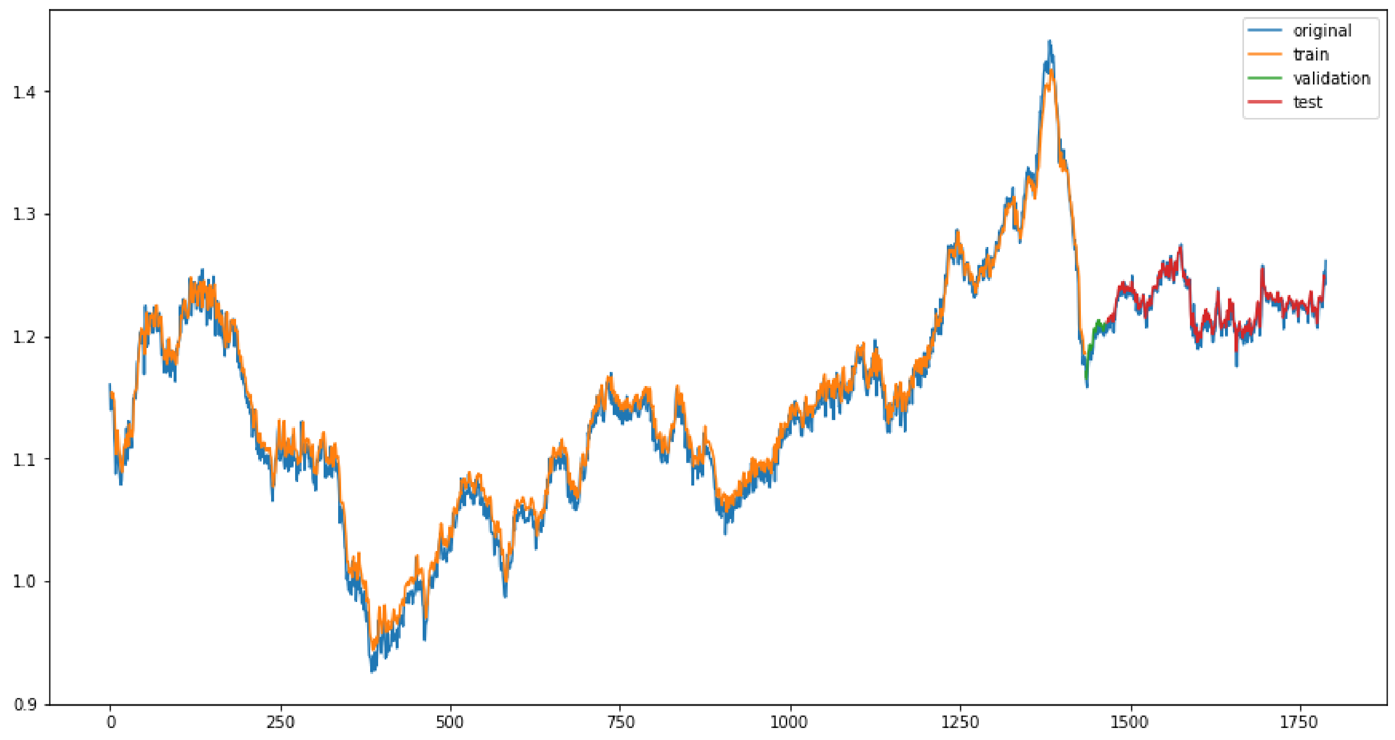


```
# build a LSTM model
model = Sequential()
model.add(LSTM(50, input_shape=(trainX.shape[1], trainX.shape[2]), return_sequences=True))
model.add(Dropout(0.1))
model.add(LSTM(50))
model.add(Dense(1))
model.compile(loss='mae', optimizer='adam')
history = model.fit(trainX, trainY, epochs=300, batch_size=100,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

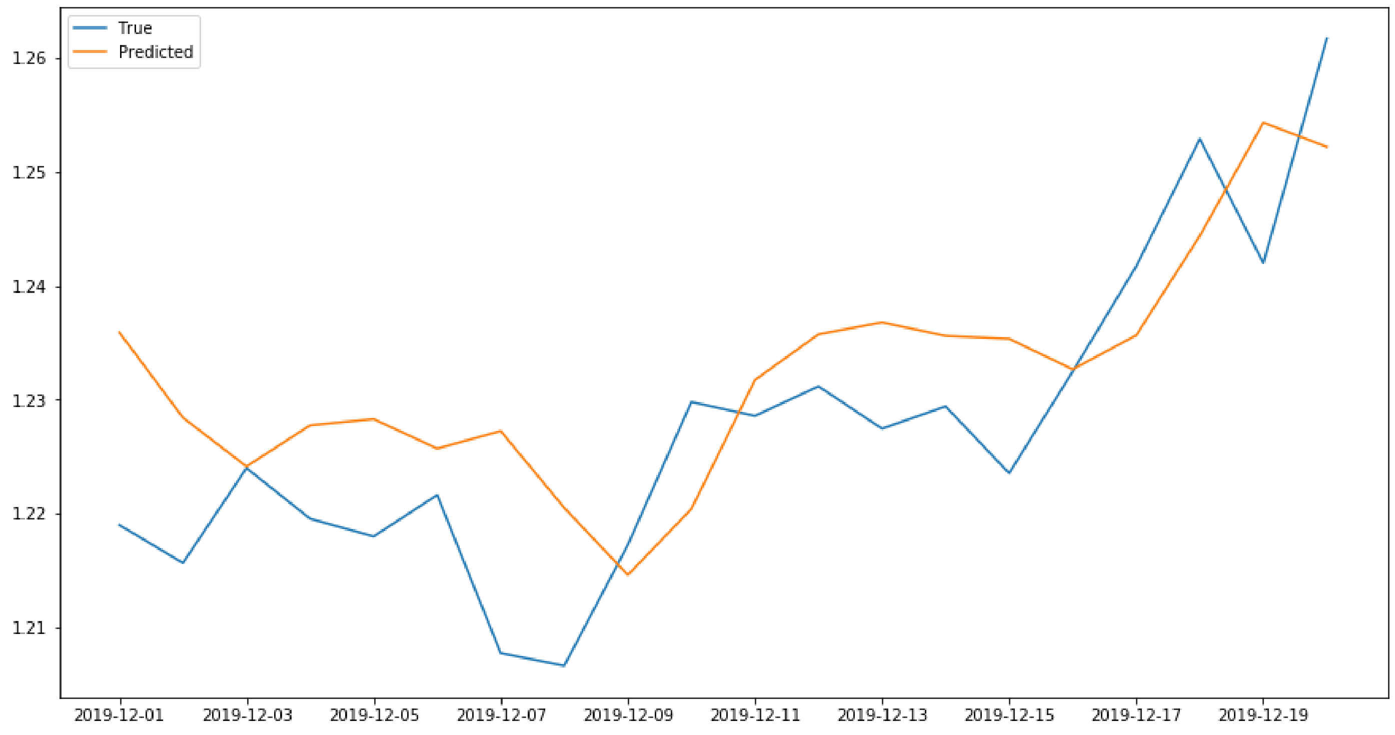
The training is very fast and plotting the loss and val_loss of the training epochs are very good:



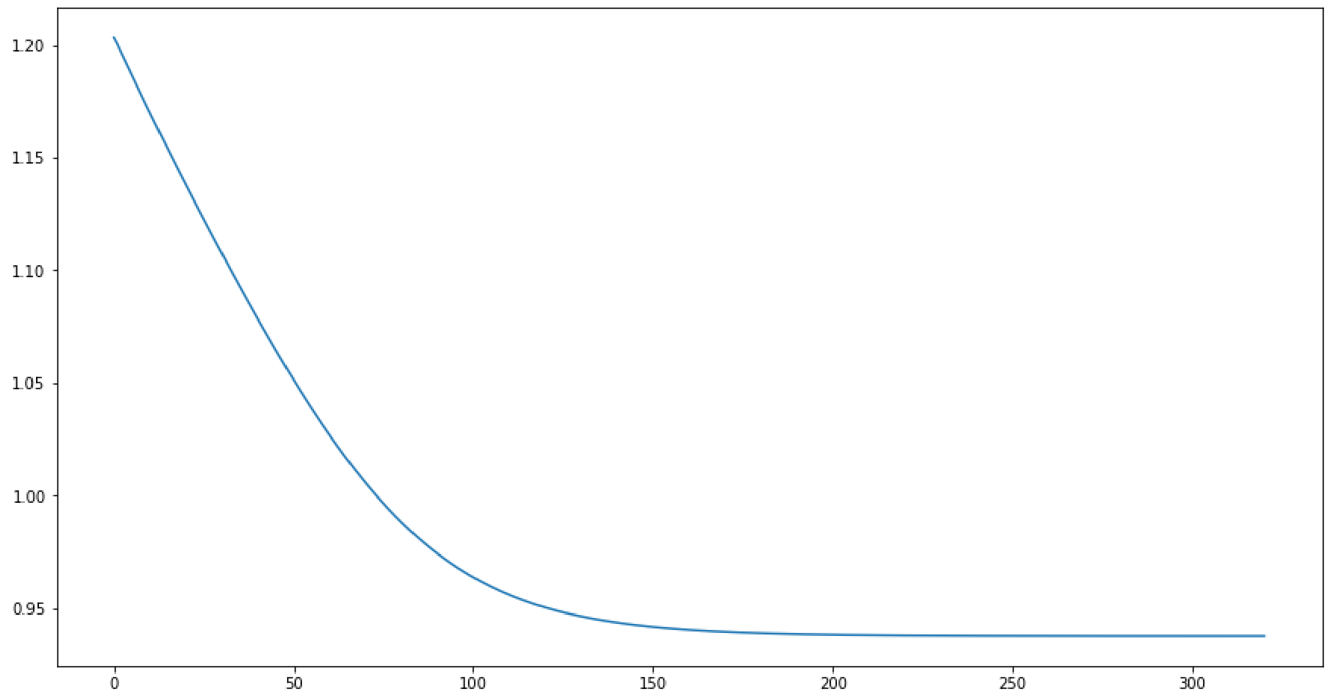
And the mean squared error for the test data is 0.008 and R2 Score is also really good with 0.78. Seems to be really good fitting of this model. Let’s see a plot of predictions of train, validation and test data:



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. When the prediction will be done with 10 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 wrote the function predict_sequence_full. This function starts with the first values of the original data, but will use in further steps own prediction. If we use this function with our test data, we get the following plot:



So the model is overfitting, means the original data is well fitted, but new data as in our predict full sequence, is not really good. But especially about this behavior of LSTM you can write an own article. Therefore I will leave it here and come to a conclusion.

Conclusion

I'll used the diesel price data freely available from [Tankerkoenig](www.tankerkoenig.de) 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.

Lookout

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.

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.

In general the fuel price seems to be hard to predict(alike the stock prices), the ARIMA model is not really well predicting and the LSTM model tends to overfitting.

Also it seems, that there are many more features outside the data that influence the price and therefore the forecasting is not so easy.