Forecasting of Fuel prices in germany

I want to analyze fuel prices in germany and investigate in some forecasting models to predict the fuel price in the future.

When it is good time for refueling?
Could I find a model to forecast the fuel price?

Therefore I used the history data available from Tankerkoenig.de (https://dev.azure.com/tankerkoenig/_git/tankerkoenig-data), they are available in the azure cloud for private use with the following license <https://creativecommons.org/licenses/by-nc-sa/4.0/>.

Data Reading

Because of the huge 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. Although I remove some wrong data I' found before. There are some prices with -0.01 € and that is not possible. So I set all wrong prices (< 0.5€ because it wasn't below in the last 20 years) with np.nan. Also I set the date field to datetime64 to better handling the data in later steps.

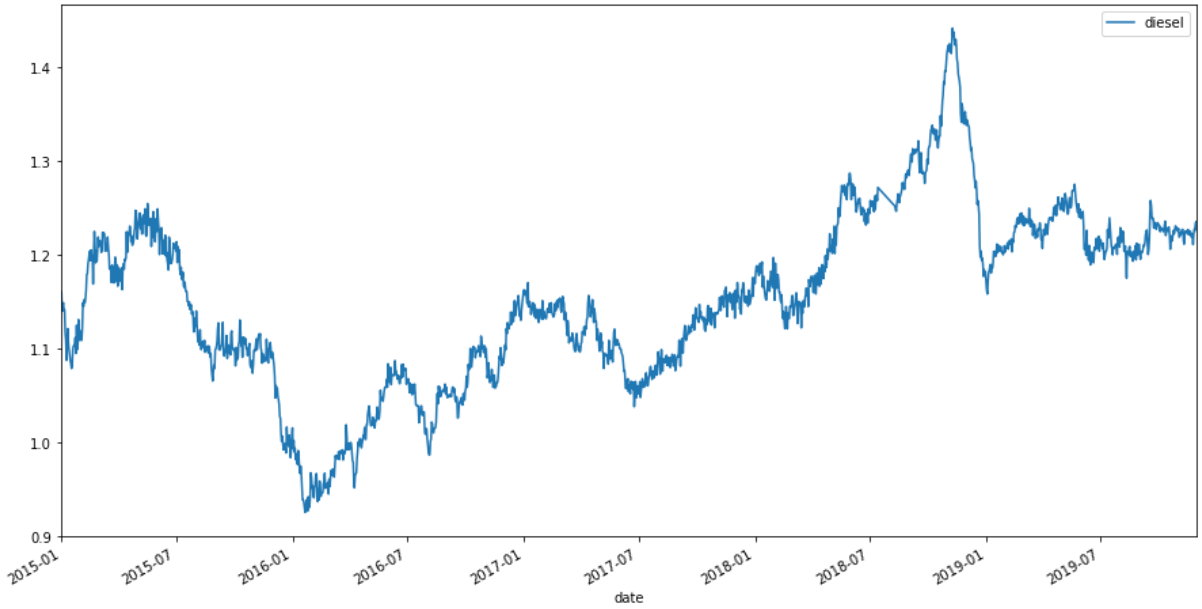
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2629941 entries, 74 to 240632
Data columns (total 5 columns):
date            datetime64[ns, UTC]
station_uuid    category
diesel          float64
e5              float64
e10             float64
dtypes: category(1), datetime64[ns, UTC](1), float64(3)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

Data Preparation and Analysis

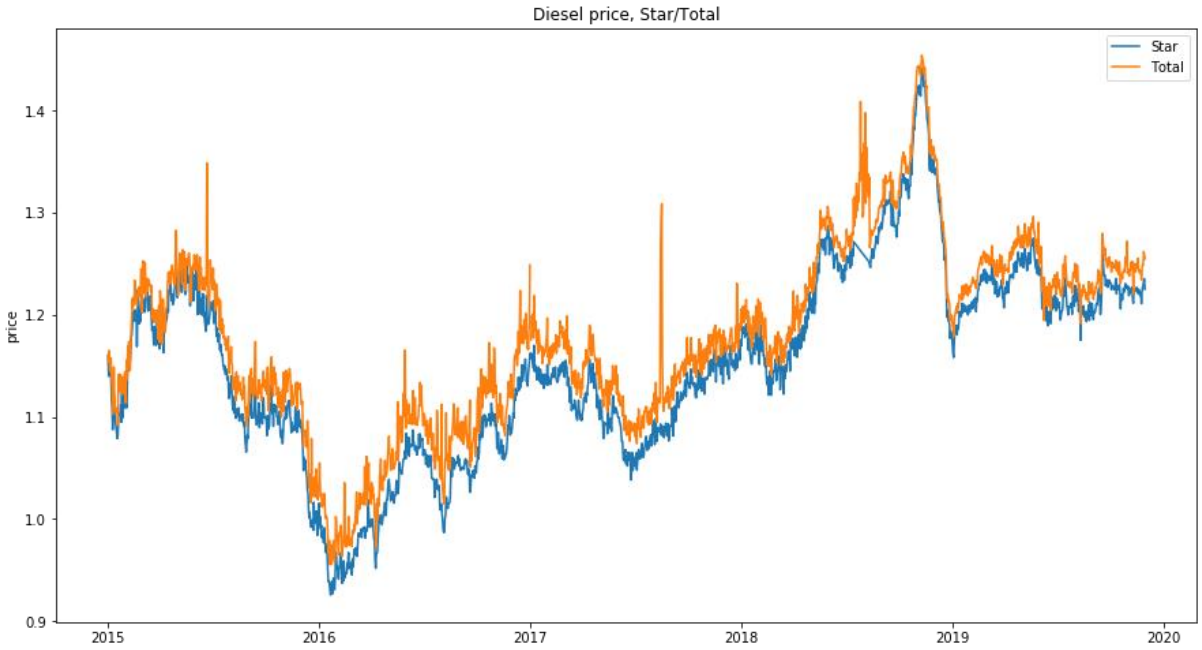
For the data analysis I've first wrote me a function to extract data for one station and type(of gasoline) out of the data:

```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
```

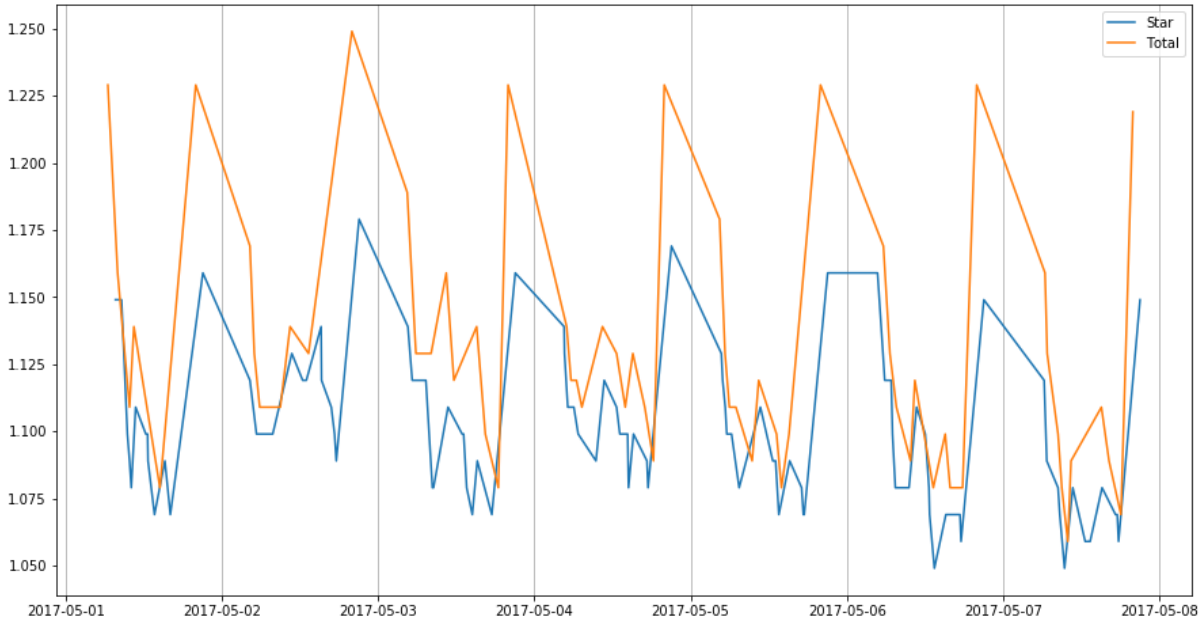
I resampled the data to daily values and have a first look in the data of one station for “diesel”:



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.



At the first view it seems correlating, but for a better analysis let’s drill down in one week:

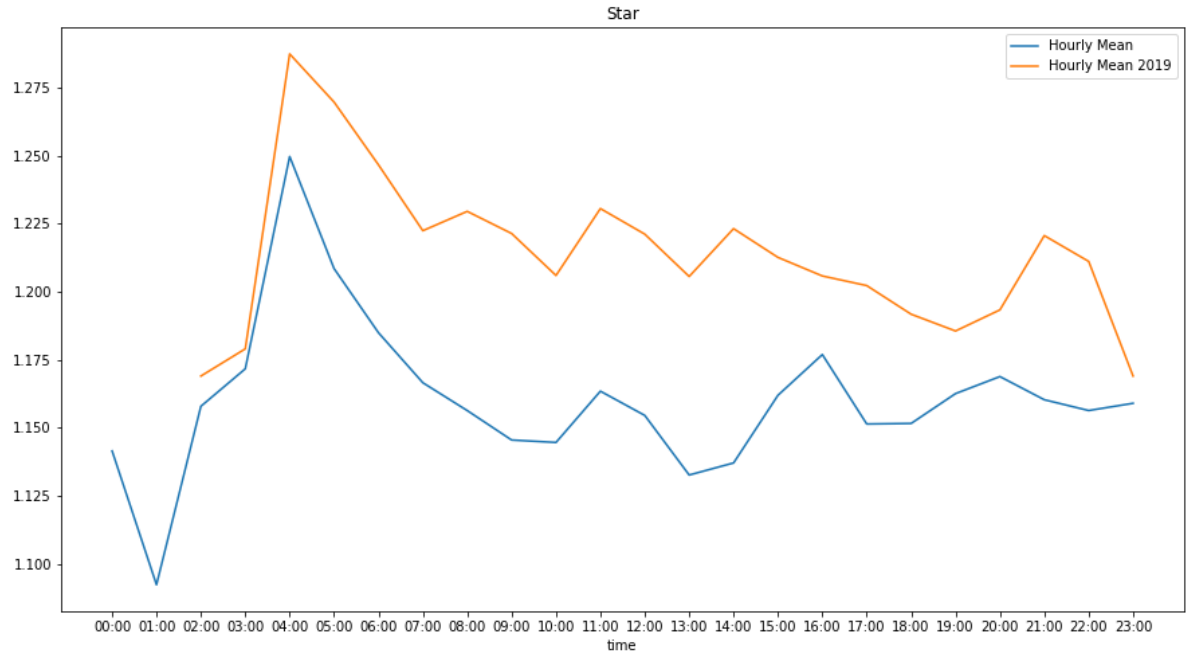


Here we can see some interesting points:

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

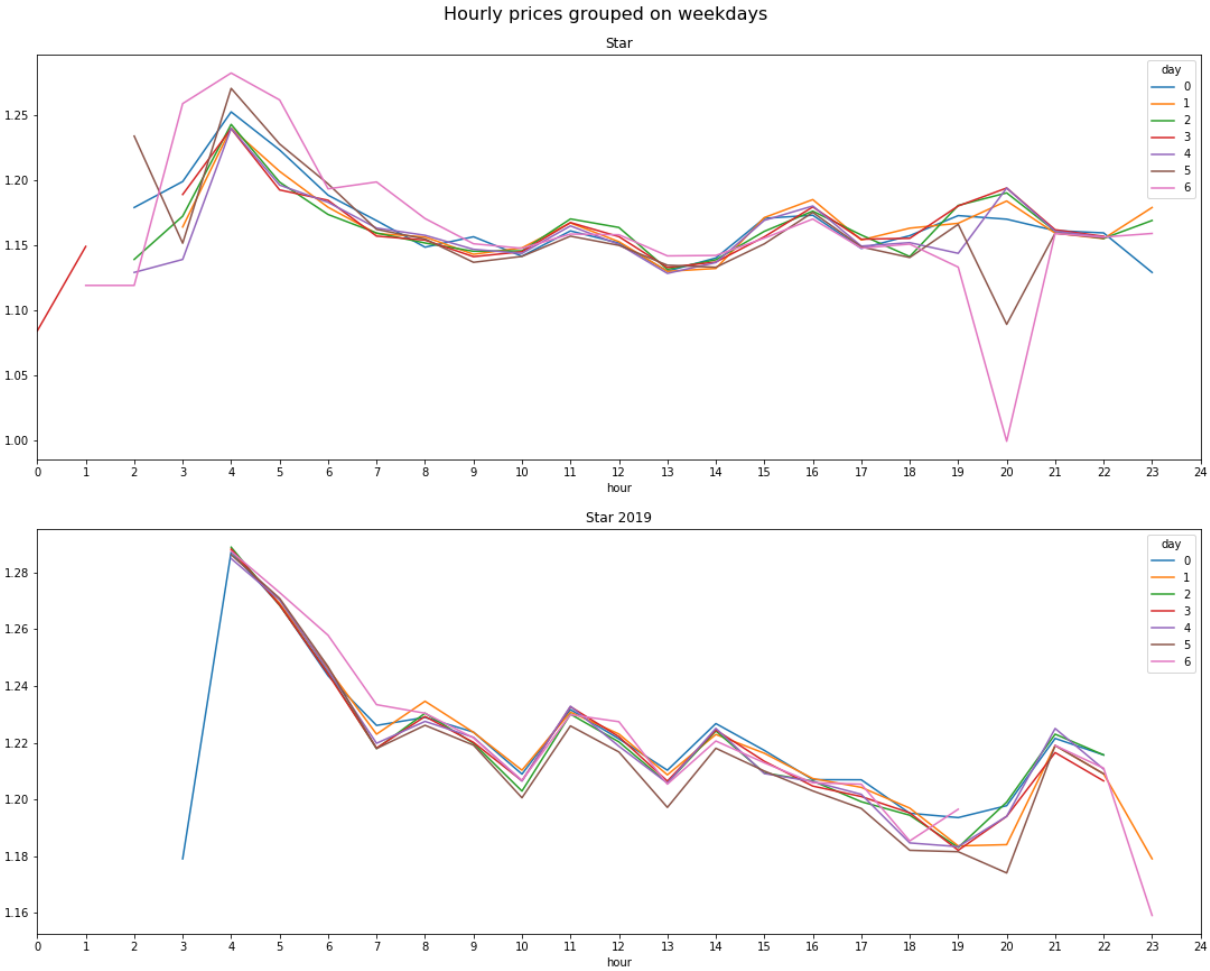
Distribution of prices on a day

One thing I want to know is when it is a good time to fill the car. 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.



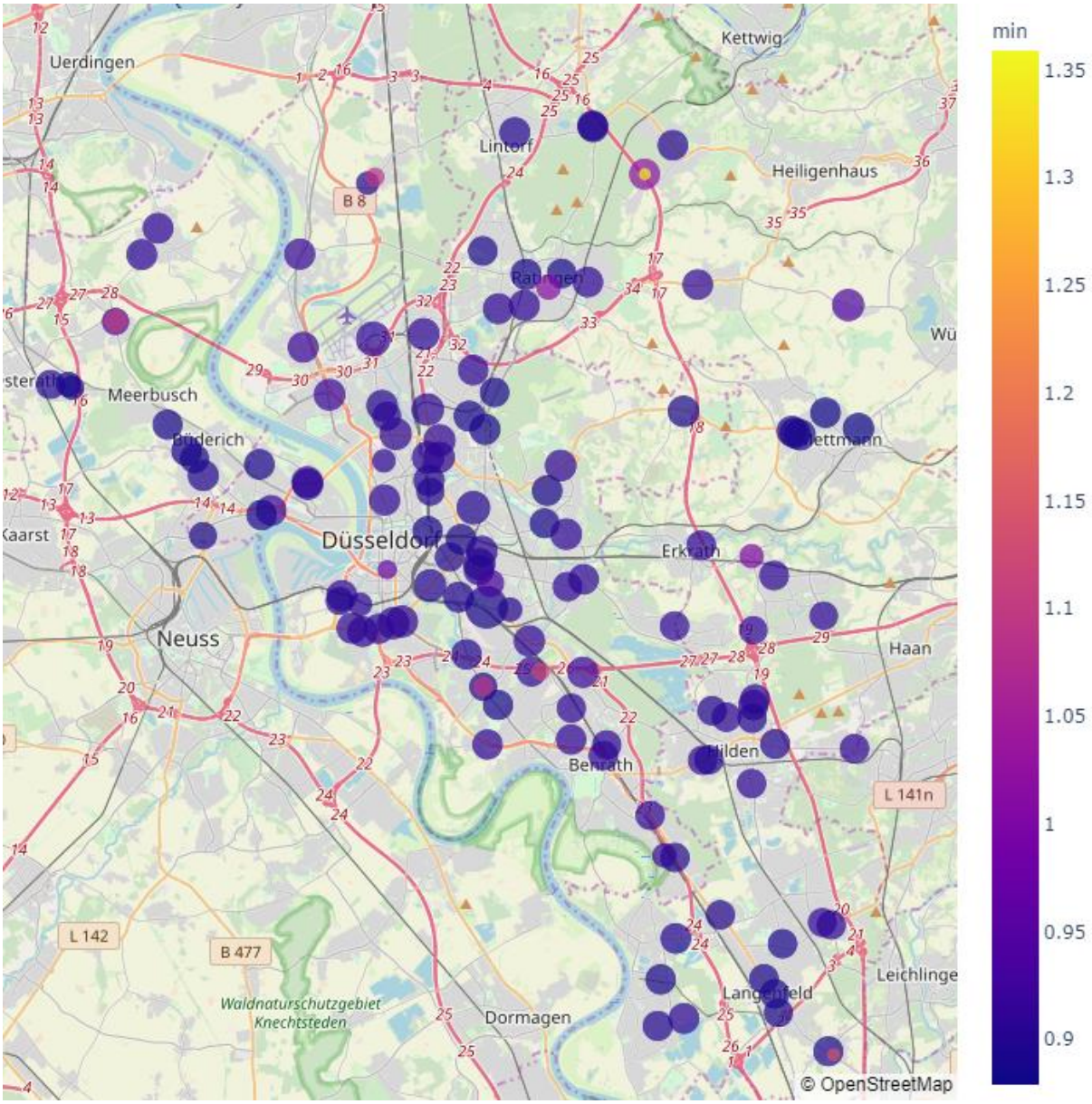
At first we can see, that the mean price for diesel raised in the last years, because the grouped values based on 2019 are higher. Also we can see that the price in last years was low on 1 pm but in the current year you’ll get the lowest price mostly at 7 pm. The “Star” closes on 11 pm, therefore the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

Is the distribution the same on every weekday?



Here we have a big difference in the years before 2019, there was Saturday and Sunday at 8 pm a 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_mabox 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.



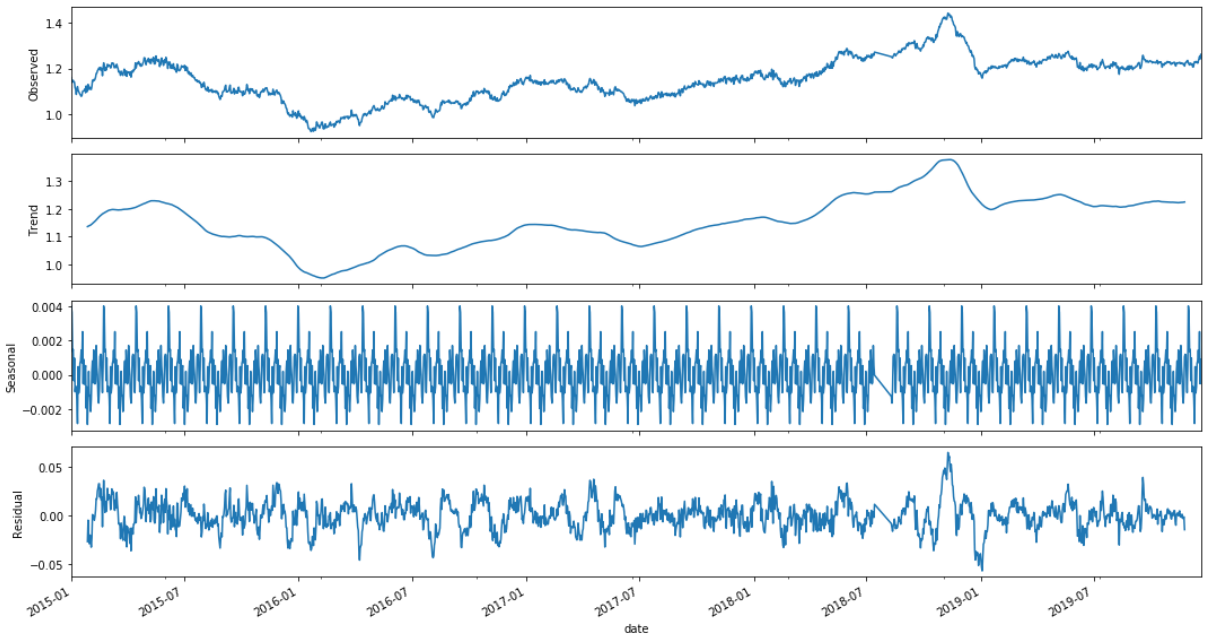
Like expected the highest minimum is on a station on our highway “A3” (the yellow circle in the top) and has also no big variance. In the notebook you can also zoom in and out, I used plotly and therefore you can have a closer look.

Forecasting

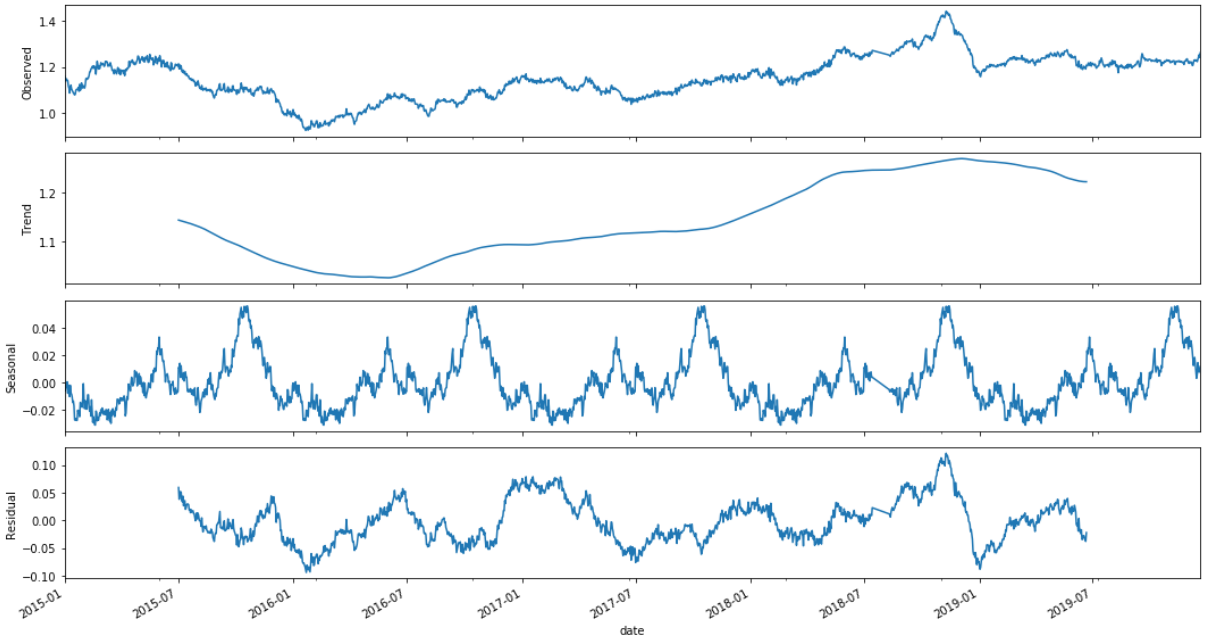
Now I’ll have a look in forecasting the fuel price, therefore exists many models: SARIMAX, Prophet, LSTM and so on. I’ll have a closer look to SARIMAX and LSTM.

But first we have a look in the data with different statistical methods to see if the data has seasonal or trend inside.

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

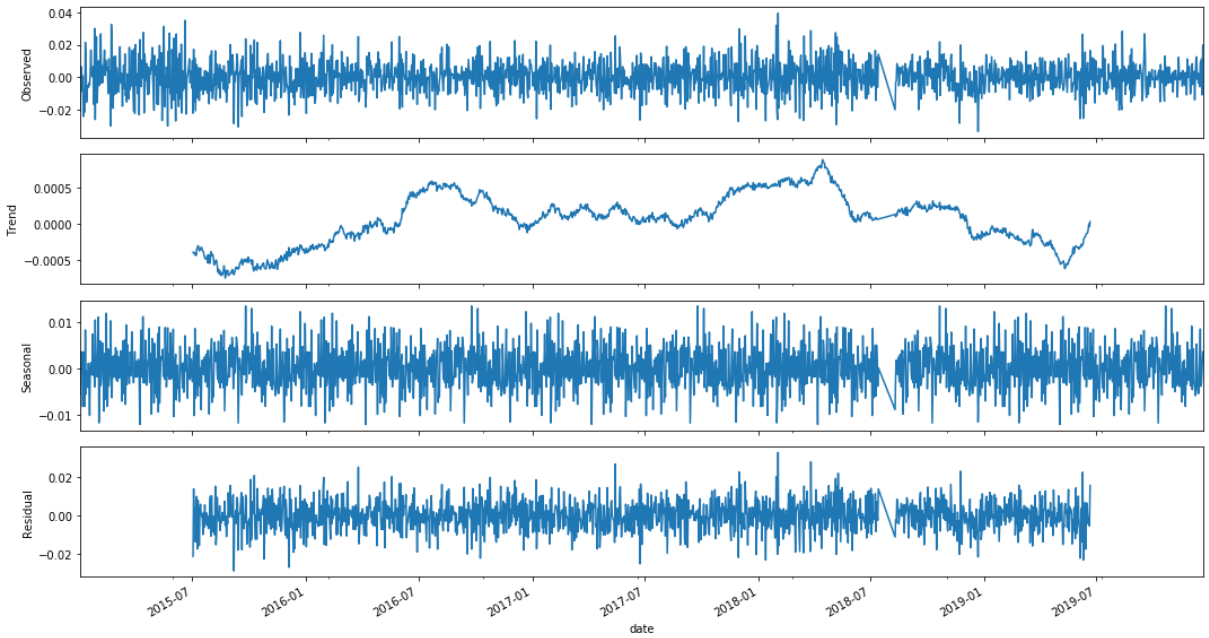



```
decompostion = sm.tsa.seasonal_decompose(star_day.dropna(), freq=365, model='additive'
```



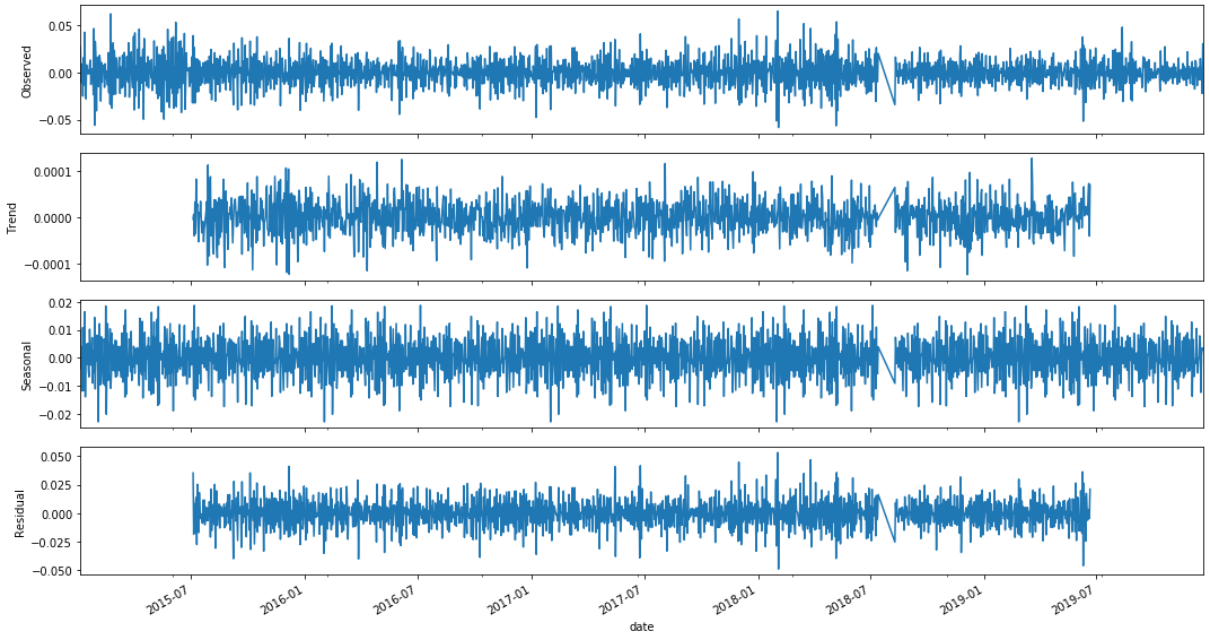
In the decomposition plots we see a seasonal trend for weeks and also for year. So if we want to use SARIMAX, we have to get a stationary data. Therefore we will have a look at the difference.

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



There is still a trend in the first difference, therefore 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.

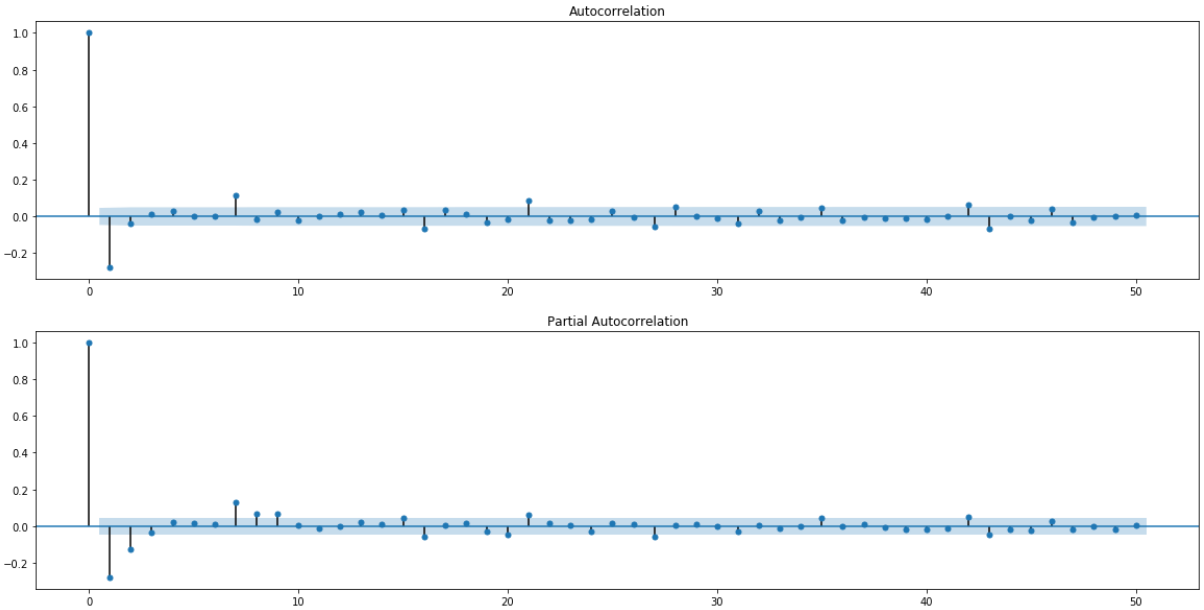
The Adfuller and KPSS Test gives for the first difference a good result, meaning that the first difference is not more stationary.

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)
```

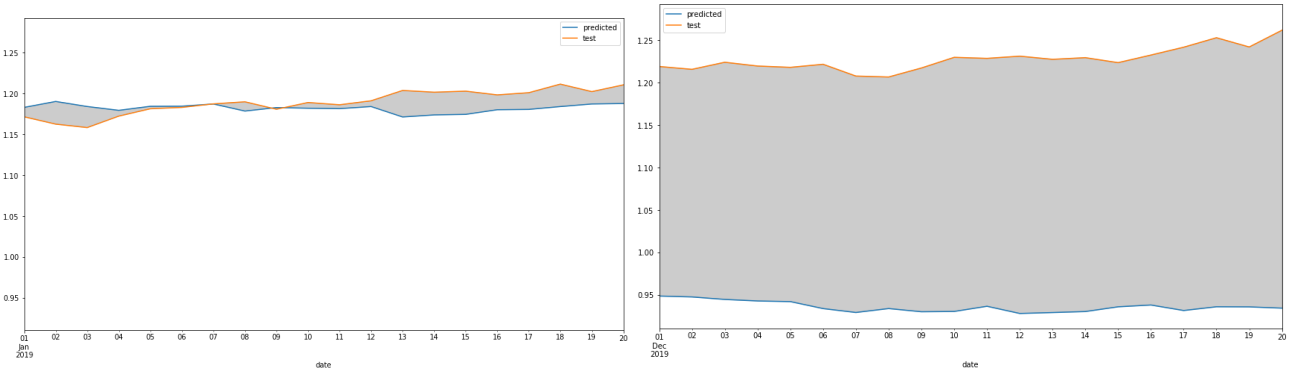
The auto correlation and the partial autocorrelation gives hints for the p and q for the ARIMA model.



The first SARIMAX model I've tested has following parameter:
`order=(3,1,0), seasonal_order=(3,1,0,52)`

The forecast of the test data gives a MSE of 0.032 and the diagnostic plot is also promising, but let's see how the prediction is in comparison to the test data. So plot it together:

The prediction is not good especially if you go further than a few days. Here are the first 20 days and the last days of test data in comparison:



LSTM Model

Next I've tried a LSTM model, because I've read many articles that describe the good prediction of LSTM. So let's see.

The data has to be normalized for LSTM, so I use the MixMaxScaler 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)
```

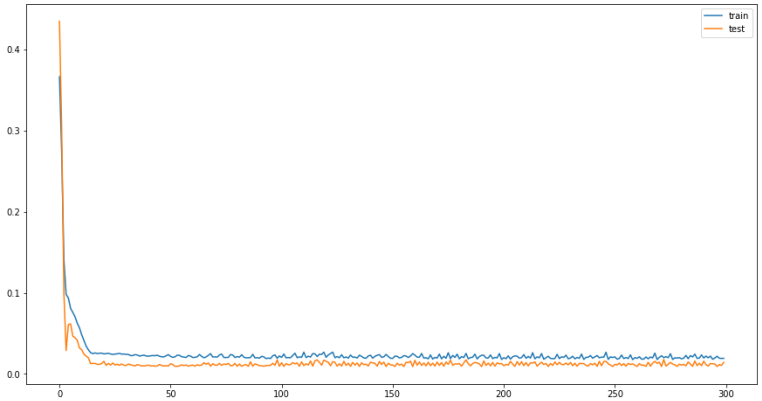
```
# use the same data for trainine up to 2018
train_size = len(star_day[:'2018-12-31'])
vali_size = 31 # let's take 1 month as validation set for fitting
test_size = len(scaled) - train_size
train, vali, test = scaled[:train_size,:], scaled[train_size:train_size+vali_size,:], scaled[train_size+vali_size:,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

```
1435 31 323
```

I'll try to use the following parameters:

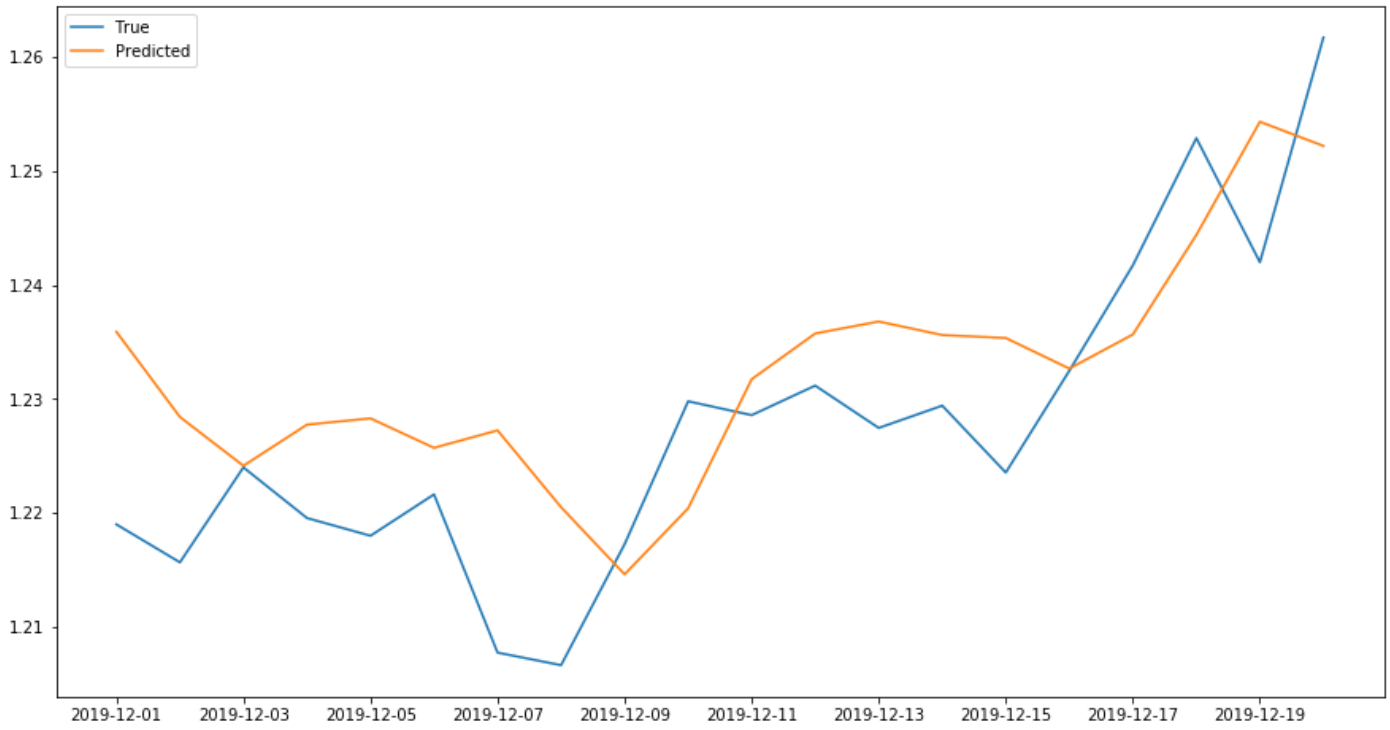
```
# build a LSTM model
model = Sequential()
model.add(LSTM(50, input_shape=(trainX.shape[1], trainX.shape[2]), return_sequences=True))
model.add(Dropout(0.1))
model.add(LSTM(50))
model.add(Dense(1))
model.compile(loss='mae', optimizer='adam')
history = model.fit(trainX, trainY, epochs=300, batch_size=100,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

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



And the mean squared error for the test data is 0.009. Seems to be really good. Let's see a plot of predictions of train and test data:

In the first view the prediction of the test data on the right side in green seems to be very similar to the original data. 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. When the prediction will be done with one values, 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 go to a limes. But especially about this behavior you can write an own article.

Conclusion and lookout

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 and extract only some data out of the zip code area of 40xxx. I made some interesting analysis on the data before coming to the forecast modelling.

The raw data showed trend and seasonality and was not stationary. So i used difference for ARIMA to have stationarity. The ARIMA model has problems with this data. On the test set you saw that the prediction after a few days differ more and more from the original data.

One possibility to get ARIMA better is to use other parameter for p, d, q. For example to use d=2, but in my tests the calculation time exhausted. Therefore i left it to the above model.

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.

For LSTM we have the possibility to take more lookbacks in account or to choose other parameters to get better predictions.

Also it could be better to combine data of some stations to become a better prediction.