Rationale and logic implementation of the test for gas analyst

Amina Talipova

January 15th, 2021

Contents

Summary

The test consisted of three tasks: prepare temperature data, develop the function of profit maximization with the given parameters, and the third one requires to provide the forecast of the given variable. Before the central part, I supply a short instruction if there is a need to check my code. The main findings are described below.

First, I have developed two methods to solve the Task 1 problem in Python. Also, I have described four ways to do that in the most manageable -> advanced order. Given the time constraint, I implemented two medium-advanced methods.

Second, I have developed the demonstrative mechanism in Task 2 explanation.xlsx support file to show Task 2 problem-solving logic. Also, the Python programming technique has been leading in Task 2 explanation.xlsx support file. Finally, I have implemented problem-solving in Python, where the output is the profit-maximizing function.

Third, I have implemented the gas demand forecast for North Carolina in Task 3 in Python. Furthermore, I have done it in the Facebook Prophet package, which I find a great instrument to carry out the given task.

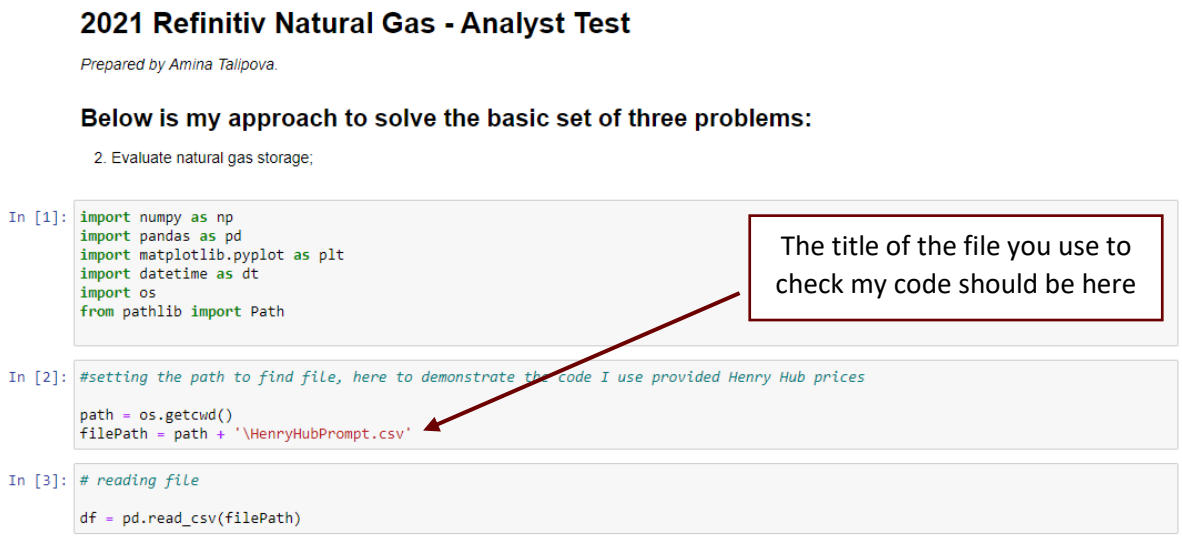
Bonus questions for Task 2 and Task 3 are presented in the last section of Bonus Questions. I have supplied the mechanism description and suggestions on how to realize the propositions.

This report ends with the reference used to solve three tasks.

Some instructions to check the test solutions

General instructions

For Jupyter notebooks from the GitHub repository, please consider that your machine's dataset file name should be the same as in the code. By default, codes will work if the original datasets will be in the same folder as the codes themselves. You may change the filePath variable if needed.



Instructions for Task 2

For this task, I also supply the "Task 2 explanation.xlsx" supporting document. I would recommend t take a look at it to understand the result I have given below.

Instructions for Task 3

It may not be easy to proceed with the Jupyter Notebook code, and I would recommend installing it in Anaconda fbprophet 6th version package. It is critical to have this version, not the last 7th one because it has many bugs yet. First, uninstall any system, fbprophet. Then follow the steps below, python.exe -m pip install pystan==2.17.1.0 python.exe -m pip install fbprophet==0.6 python.exe -m pip install --upgrade fbprophet.

Task 1. Preparing Data

For this interesting task, I developed two functions to calculate the daily temperature, given each station's weight. But first, I would like to mention all the possible ways to solve the problem and then describe why I have chosen these two particular ways. First, the main problem is how to deal with the missing observations because we don't have a lot of measurements (every 6 hours in most of p), and those we have are critical for the final average daily temperature calculation. I believe that there are four possible ways:

1. Suppose we don't care about accuracy or don't have much time for a "fast-and-dirty" analysis. In that case, it is usually Ok to *just drop all missed data (NaNs)* and calculate the daily averages without missed data points. However, finding the weighted average will consider the missing number as equal to zero. Though this way is straightforward to implement, I find it inappropriate.
2. To solve the NaNs problem by *interpolating them across columns or rows*, implying a particular imputation strategy (backward, forward, or interpolation: linear, polynomial, spline, etc.) spatial-temporal data. As we already have the weights, it would be reasonable to do the temporal relationship by interpolating data and the spatial one by applying the weights. The output is a function that interpolates across the columns with the principle of equidistant time measurements (every 6 hours)[1] and then calculates the weighted average. It is quite correct, but not the ideal way.
3. Here we come the most accurate (in my expectation and understanding) way. To have both temporal and spatial relationship accountancy, we need to interpolate across both columns and rows and then calculate the weighted average. However, this dual interpolation requires additional data about weather station coordinates that we don't have. But nobody told us we could not find them. In other words, *if we have station coordinates, we can interpolate NaNs first temporally across column (hourly) and then across rows using the principle "nearest neighbor," as we have coordinates*, to interpolate them spatially. After that step, we use weights to calculate the daily average temperature when we have accurately solved NaNs. This approach is similar to the so-called Kriging method[2] , and I used the methodology close to the described by (Gerber et al., 2018). In my solution, I implemented the 2nd and the 3rd described methods. The second one is that there are many ready told to easy and fast accomplish it, and the third is because I had time to do it. Though given not so limited time, I would recommend the 3rd one only.

[1] Important! I first resembled the whole data frame because there are missing six-hourly observations.
[2] For more can be found here. URL: https://desktop.arcgis.com/en/arcmap/10.3/tools/3d-analyst-toolbox/how-kriging-works.htm

4. Random Forest Regressor. This machine learning technique is excellent for solving the problems of both spatial and temporal relations. Given the time constraints, I could not realize it though I can do so if required.

Method № 2 can also be implemented in different ways. I suggest two most appropriate to account temporal relations:

1. **Method 2.1.** The temporal relations to solve the NaNs problem are defined as the average temperature at the weather station resampled to the equidistant hours. For example, missing observation on January 1st at 6 pm is equal to the average of all observations on this day.
2. **Method 2.2.** The temporal relations to solve the NaNs problem are defined as the average temperature of the particular missing observation as the average on the day before and the day after at the same time. For example, missing observation on January 1st at 6 pm is equal to the average temperature on December 31st at 6 pm and January 2nd at 6 pm.

Below are the snippets of problem-solving in Python using the given dataset implying Method 2.1 and 2.2. In Figure 1, I show the part of the code where the described method is suggested.

```
In [35]: df1s = []
for hour, data in df1.groupby('hour'):
    df1s.append(
        ((data.fillna(method='bfill') +
         data.fillna(method='ffill'))/2).
        fillna(method='bfill').
        fillna(method='ffill')
    )
df_filled = pd.concat(df1s, axis=0).sort_index()
```

```
In [36]: df_filled['date'] = df_filled.index.date
df_filled.drop(labels=['hour'], axis=1, inplace=True)
df_daily=df_filled.groupby('date').agg(np.mean)
print(df_daily.head(10))
```

```
            Brice Norton  Herstmonceux   Heathrow  Nottingham   Shawbury  \
date
2016-01-01      5.833333      8.133333   7.233333    3.666667   5.733333
2016-01-02     10.791667     10.816667  12.125000   11.079167  11.108333
2016-01-03      8.612500      7.541667   8.687500    8.091667   8.970833
2016-01-04      7.262500      6.062500   7.850000    6.983333   6.358333
2016-01-05      8.952083      8.389583   9.850000    8.137500   7.854167
2016-01-06     11.691667     12.579167  12.754167   10.550000  13.787500
2016-01-07     15.062500     14.875000  15.733333   13.991667  13.633333
2016-01-08     13.975000     14.591667  15.354167   14.529167  13.966667
2016-01-09     16.331250     16.406250  18.075000   16.404167  15.758333
2016-01-10     10.210417     11.637500  11.654167    9.493750   9.058333

            Waddington
date
2016-01-01    4.866667
2016-01-02   11.437500
2016-01-03    8.166667
```

Figure 1. Method 2.1 snippet

Results of the weighted average daily temperatures in the UK six weather stations is given in Figure 2.

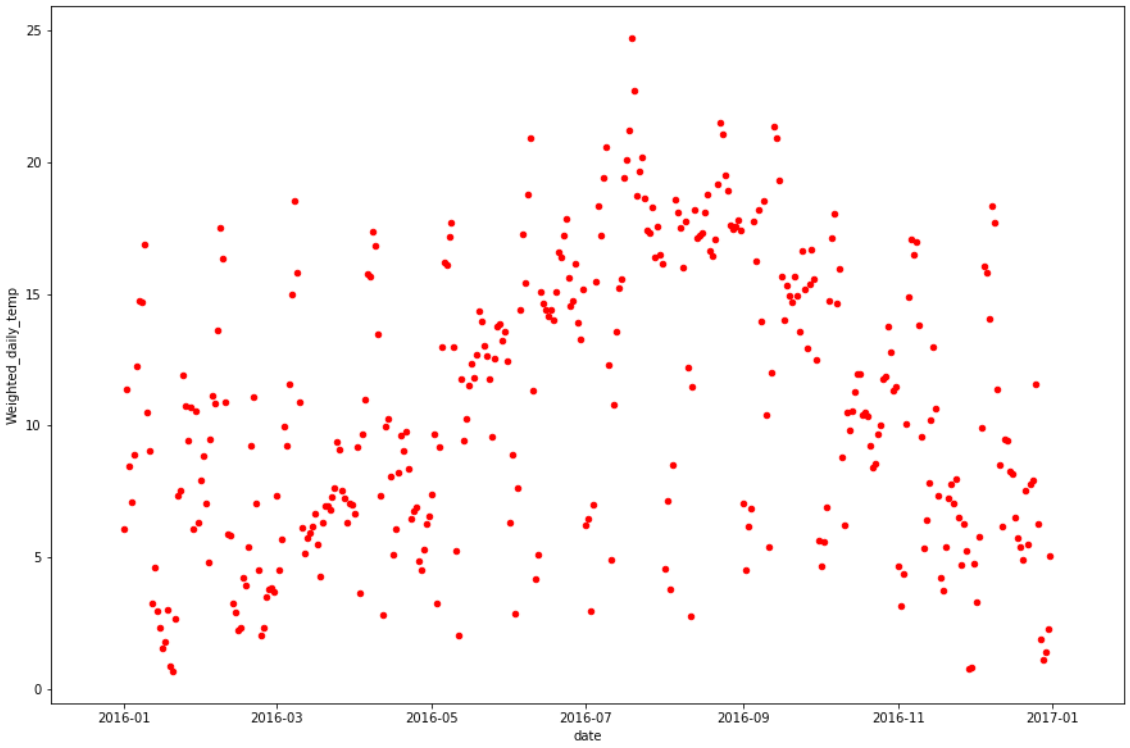


Figure 2. Weighted average daily temperature in UK 6 stations with missing observations interpolation using Method 2.1 results

In Figure 3, I provide the snippet of Method 2.2.

Method 2.2.

```
In [ ]: # Method 2.2 First, we need to resample the dataframe because the dates and hours of observations defferent


df_new = df.resample('3H').interpolate(method = 'spline', order = 1, limit_direction='both')

print(df_new.head(20))
print(df_new.info())
```

```
In [ ]: # and after resampling make grouping by day

df_new = df_new.resample('D').mean()
print(df_new.head(20))
print(df_new.info())
```

```
In [ ]: # analyze dataframe

df_new.info()
```

```
In [ ]: # check if data frames satisfy to be multiplied to get the weithed average

df_new.values.shape
df_dictionary.Weight.values[:, np.newaxis].shape
```

```
In [ ]: # finally, calculate weighted daily average

df_new['daily_temp'] = df_new.values.dot(
    df_dictionary.Weight.values[:, np.newaxis]
)
```

```
In [ ]: print(df_new.head(10))
```

Figure 3. Method 2.2 snippet

The results of Method 2.2. are given below.

```
In [19]: df_new.reset_index().plot(x='DateTime', y='daily_temp', color='red', kind='scatter', figsize=(15, 10))
         plt.show()
```

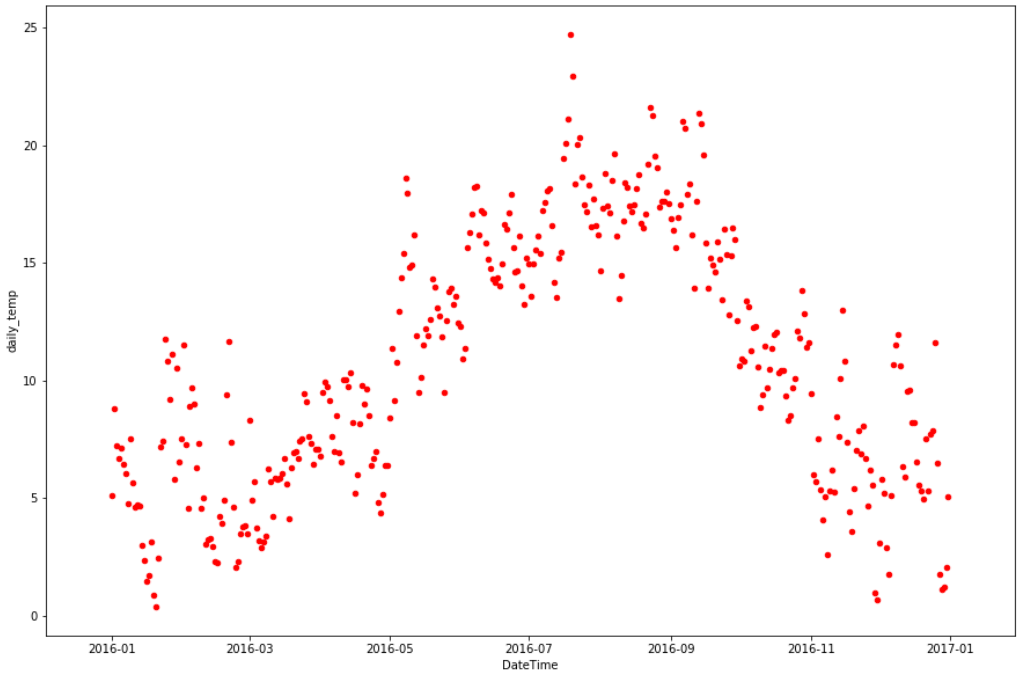


Figure 4. Weighted average daily temperature in UK 6 stations with missing observations interpolation using Method 2.1 results

Finally, Method № 3 has been implemented to interpolate both spatially and temporally. However, it requires more attention to be fully completed, and for now, I demonstrate only the snippet (Figure 5). Furthermore, the capacity of my laptop processes this method quite slowly.

```
# finally, using griddata and loc functions, finally, I interpolate the dataframe using the data in the training one.
# Specifically, I interpolate temporally using calculated seconds, and spatially using the coordinates

# from scipy.interpolate import griddata



print('interpolation started')
#[La, Lo, Ho] = training_df.loc[:,['Lat','Long','Hours']].values
#[mLa, mLo, mHo] = (filling_df.loc[:,['Lat', 'Long', 'Hours']].values)

#Z = training_df.loc[:,"Temperature"].values

df_2['NewTemp']=gd((training_df[['Lat','Long','Hours']].values),
        (training_df["Temperature"].values),
        (df_2[['Lat', 'Long', 'Hours']].values),
        method='linear')
```

Figure 5. Both spatially and temporally interpolation Method № 3 snippet

Task 2. Storage Value Assessment

The price of natural gas in free open markets changes within the day that we call a spot trade, within the month that we call futures commonly, etc. The situation, coupled with other factors, opens up the possibility of temporal arbitrage: buying gas at a low price, storing it, and selling it later at a higher price. To successfully execute any arbitrage strategy (daily, monthly), some amount of confidence in future prices is required to be able to expect to make a profit. In the case of gas arbitrage, the storage system's constraints must also be considered taking into consideration seasonal fluctuations of prices. To solve this task with given conditions, we also need to set three sufficient and necessary conditions:

1. *The price array is defined.* We say that the price array is one we rely on when calculating the final profit. We need this condition because, in reality, traders commonly operate with prices forecast that do not allow to satisfy that there are no further limits on daily injection and withdrawal rate/volume and make them to set the hedge rate;
2. *All available volume of storage is bought or sold.* If the price array is defined, the only sufficient and necessary condition that allows finding the maximum profit function from buy/sell/storage is that all available storage is filled (when gas is bought) and vice versa. In any other case, the profit cannot be maximized;
3. *In one day, we can only sell or buy gas.*

I also assume that the storage costs are equal to zero in this task and proceed with the maximization profit margin given in Figure *6* and Figure *7*. Figure 1 shows the components of the storage amount.

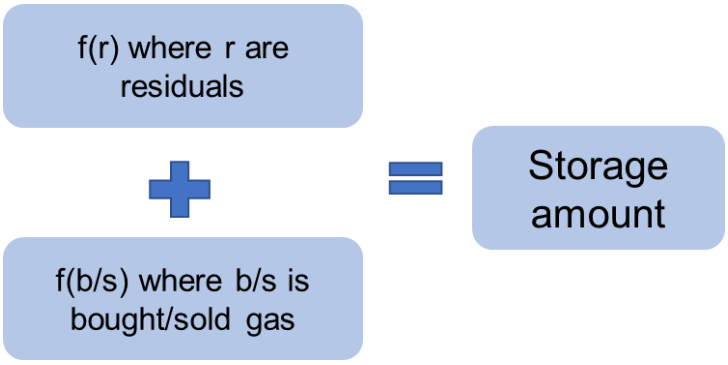


Figure 6. Gas storage decomposition

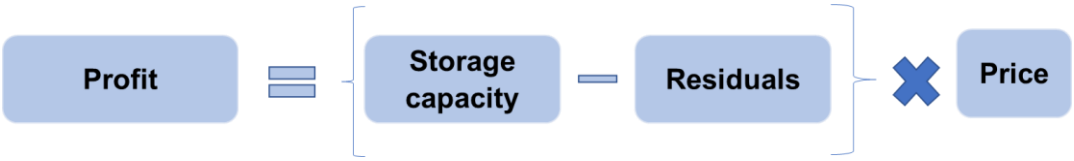


Figure 7. Profit definition

Thus, the profit maximization function is as follows:

$$f(profit) = \sum_{i=1}^{n}(storage\ capacity - residuals_i) * price_i \quad (1)$$

where storage capacity is fixed, residuals are the amount of gas stored after sale/buy, and the price is a daily gas price. I have solved this Task 2 implying the prices on Henry Hub.

First, I follow the simple logic that gas strategy means "buy when the price is the lowest and sell when the price is highest." As we do not constrain the gas amount withdrawal, the only possible strategy that maximizes the profit is when we buy and sell all the storage amount at a time of the lowest or highest peak.

To implement this logic, I created two functions in Python. The first function (in my code, it is findPeaks) should find the pairs of days, when it is peak to sell or buy, given the price array. The second function (in my code, mxProf) should use the first to calculate the maximum profit with the given amount of storage and prices. The second function's output should provide the maximum profit and trade strategy as an array of day indexes to buy and sell. In Figure *8* and Figure *9*, I show the screenshots of these two functions. They are also available to check in the Jupyter. Finally, Figure *10* shows the output.

```python
# Lets realize the logic (demonstrated in the Excel file "Task 2 explanation") of the trade strategy when we buy at the
# record low and sell at the record high prices
# Here the output is an array of positions where even positions are the end of decreasing at the lower peak and
# odd positions are the end of increasing at the higher peak

def findPeaks(arr):
    '''
    findPeaks () returns the array of indexes fo those days when we sell or buy
    as list of pairs [[buy_day1, sell_day1],...,[[buy_dayN, sell_dayN]]]
    '''
    indexes = []
    lower = True
    prev = arr[0]
    for (pos, val) in enumerate(arr):
        if pos == 0:
            continue
        if (lower and val > prev) or (not lower and val < prev):
            lower = not lower
            indexes.append(pos - 1) # previous day is end of sequence
        else:
            prev = val
  # the last element hasn't been added; it is always the end of an increasing or decreasing sequences
    indexes.append(len(arr) - 1)
    return indexes
```

Figure 8. findPeaks() function snippet

```
In [7]: # Finally, we need to write the function that will take our logic of trading and multiply storage capacity and
        # prices to calculate the maximum profit

        def mxProf(arr, capacity):
            '''
            mxProf() returns max_profit and the trade strategy as list of pairs [[buy_day1, sell_day1],...,[[buy_dayN, sell_dayN]]]
            inputs:
            arr - array of prices
            capacity - storage capacity
            '''
            profits = 0
            if len(arr) < 2: # here is writen the condtion that we cannot sell and buy on one day
                return profits, []
            seqs = findPeaks(arr)

            if len(seqs) == 1: # if our array is dectreasing in a constant way; or minimize loss
                prev = arr[1]
                ind = 1
                maxDiff = arr[1] - arr[0]
                for (pos, val) in enumerate(arr):
                    if pos == 0 or pos == 1:
                        continue
                    if val - prev > maxDiff:
                        maxDiff = val - prev
                        ind = pos
                    prev = val
                return [[ind - 1, ind]]

            if len(seqs) % 2 == 1: # define when the lowering is ended
                seqs = seqs[:len(seqs)-1]

          # here we sell at the highest peak, buy at the the lowest, and wait between
            days = []
            for i in range(1, len(seqs), 2):
                days.append([seqs[i-1], seqs[i]])
                profits = profits + capacity*(arr[seqs[i]] - arr [seqs[i-1]])
            return profits, days
```

Figure 9. mxProf() function snippet

```
In [8]: # finaly, check if the function to find trading strategy and maximized profit works

        mxProf(df.Close.values, 200)
```

```
Out[8]: (3065.2000000000025,
         [[0, 1],
          [3, 4],
          [6, 8],
          [9, 11],
          [13, 15],
          [16, 18],
          [19, 22],
          [24, 26],
          [28, 30],
          [34, 37],
          [39, 40],
          [42, 44],
          [46, 48],
          [51, 54],
          [56, 60],
          [63, 65],
          [67, 71],
          [74, 77],
```

Figure 10. mxProf() function output

Interpreting the output, the maximum profit the company can earn from trading natural gas with the given amount of capacity, taking Henry Hub daily prices provided as the sample, and with the proposed strategy is 3.07 mln dollars.

Task 3. Data Analysis

Time series forecasting finds wide application in data analytics. There are quite a few different methods to predict future trends, such as ARIMA, ARCH, regressive models, autoregressive, vector autoregressive, and neural networks. I do not doubt that all analytics are familiar with all those models and know how to implement them in R or Python, using standard libraries. *Therefore, I would like to share with you (or maybe you already KNOW IT) the library, which has been recently become public and developed by Facebook. It is called Prophet and was initially designed to create high-quality business forecasts*[3]. This library tries to address the following difficulties common to many business time series:

1. Seasonal effects caused by human behavior: weekly, monthly, and yearly cycles, dips, and peaks on public holidays.
2. Changes in trend due to new products and market events.
3. Outliers.

I find it great to analyze the task with the natural gas demand dataset and its dependence on the temperature implying the Facebook Prophet package. It is significant to consider the autoregression gas demand model as a function of time. Why? Simply because this component exists, primarily when it is used for power generation purposes. For example, during holidays or weekdays, households consume a potentially more considerable amount of electricity.

I start data analysis, not with r model where gas demand is a dependent variable, and the temperature is the independent one, but from the autoregressive model (Taylor and Letham, 2017) of only gas demand because **my first assumption is that the gas demand in North Carolina or anywhere else only depends on a particular day.** In turn, the temperature is just relying on the weather. The latter also depends on the specific day of the year (except for the temperature anomalies). Thus, the gas demand should depend more on the day of the week rather than on temperature. As mentioned, I use the fbprophet package[4] with an additive model (also transformable to the multiplicative) and fit the gas demand model in North Carolina (Figure *11*).

[3] For more see here. URL: https://facebook.github.io/prophet/

[4] Note: You will need Prophet v.0.6 for this code to work. The easiest way to install it is via Anaconda.

```
# Here I create new dataet to manipulate with in Prophet
dfd = df_d
dfd.columns = ['ds', 'y']
dfd.head(5)

# I define the prediction size and create new training dataset
prediction_size = 92
train_dfd = dfd #-prediction_size]
train_dfd.tail(5)

# Here I tell Prophet to make the prediction with defined periods and print
# the results
model = Prophet(daily_seasonality=False, seasonality_mode='multiplicative')
print("... start fitting")
model.fit(train_dfd)
print("fit is done")
future = model.make_future_dataframe(periods=prediction_size)
new_dfd = model.predict(future)
print(new_dfd.info())
print(new_dfd.tail(10))
print(new_dfd.columns)
fig1 = model.plot(new_dfd)
axes = fig1.get_axes()
axes[0].set_xlabel('')
axes[0].set_ylabel('North Carolina gas demand forecast, MMsf')
fig2 = model.plot_components(new_dfd)
```

Here I set the number of days to predict according to the task

Model fitting

Figure 11. Autoregression model for the gas demand in North Carolina

The forecast result of the gas demand is given in Figure *12*.

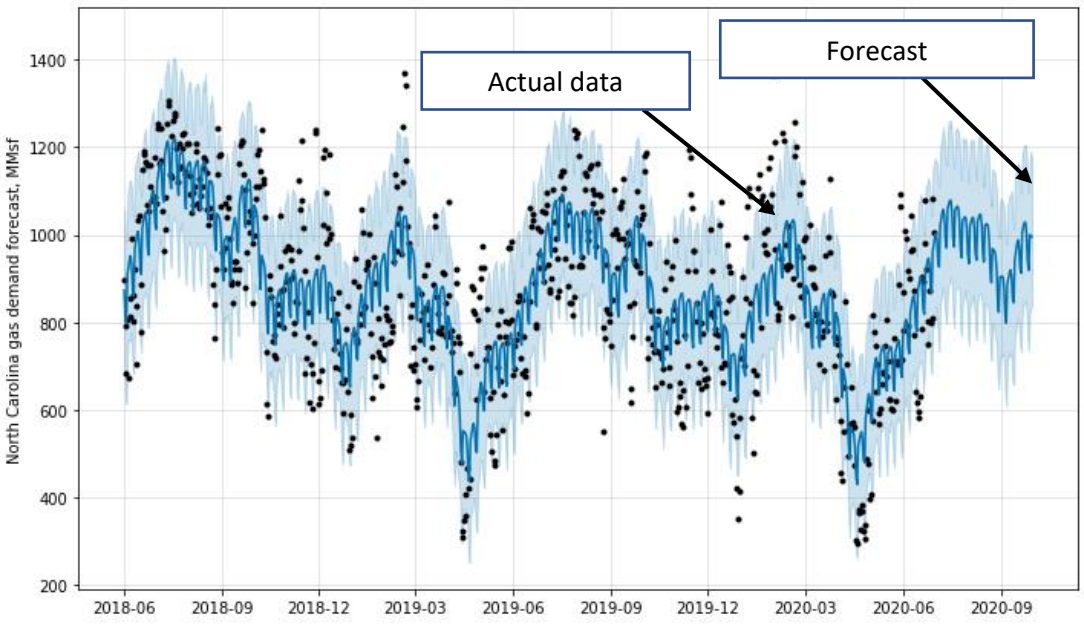


Figure 12. NC gas demand forecast [MMscf]. Actual data - black dots, forecast - solid blue line.

Analyzing the components (Figure 13), we can see that the most important ones are the trend and the week's day. It can be easily explained. From Sunday to Friday, the electricity demand (which is generated from natural gas) is mostly consumed by the offices or manufacturers, while they are closed on weekends. Also, there is a continually decreasing trend, especially in 2020.

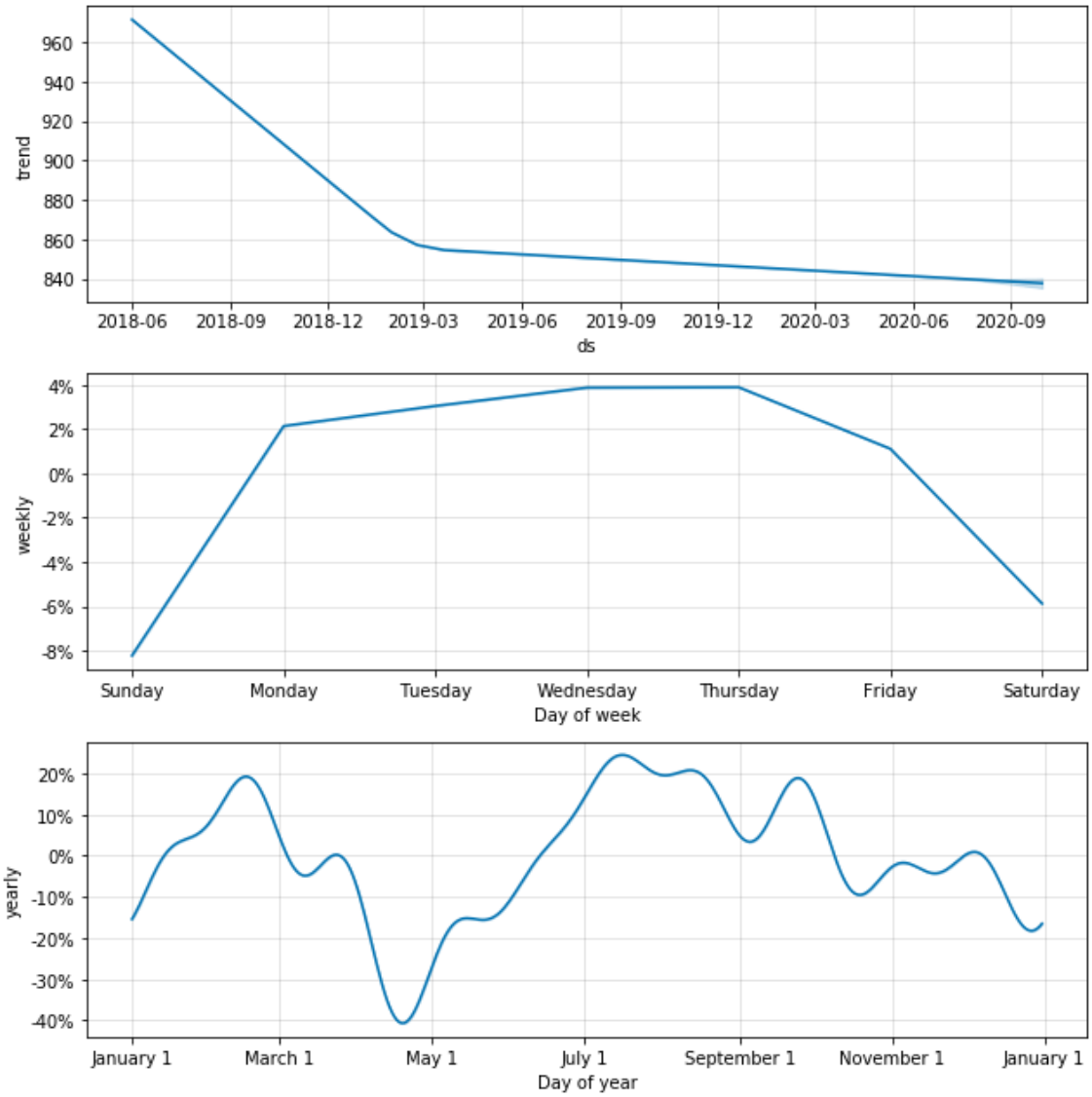


Figure 13. Forecast model components without temperature as a feature

After that, I analyze the temperature (*Figure 14*), and most importantly, I build the forecast for the temperature and further calculate "temperature surprise".

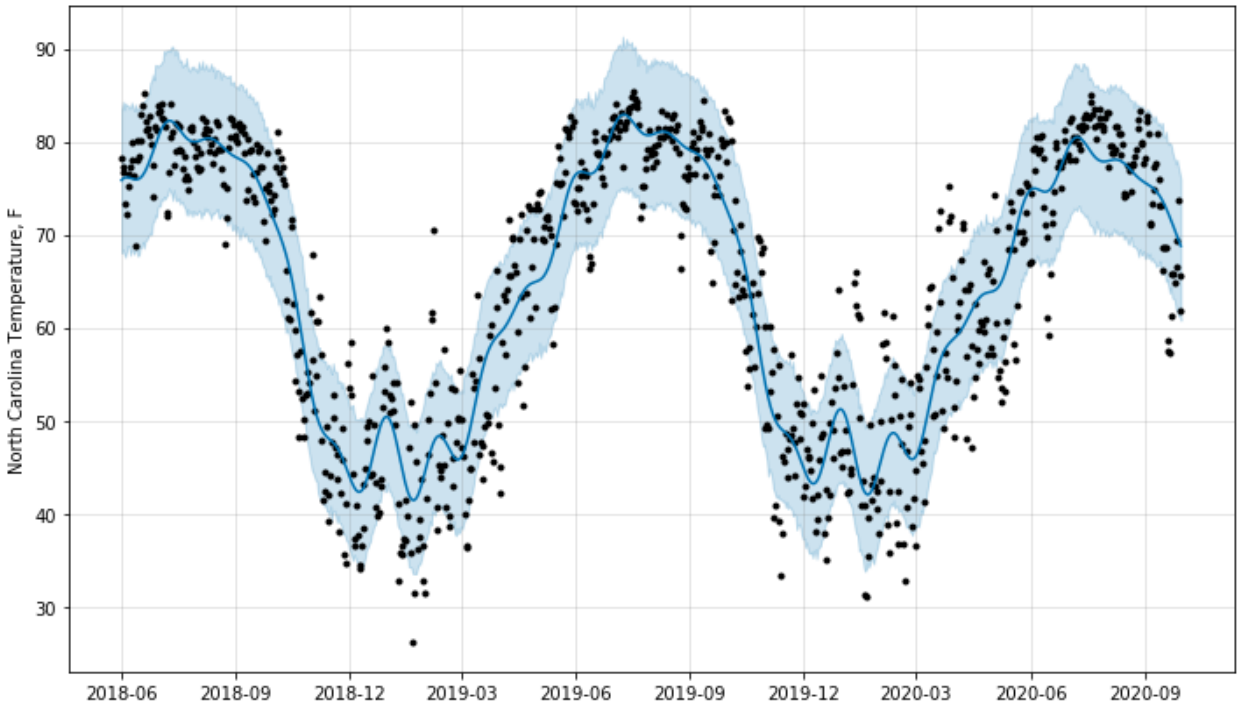


Figure 14. North Carolina daily temperature

However, when building a regression model forecast with demand and temperature, I do not just include the temperature observations. **Instead, I use the newly created "temperature surprise" as a regressor, which is defined as the difference between the actual measured and predicted temperature (autocorrelated trend).**

Why is it critical to just simply include temperature? The reason is that the temperature autocorrelation trend is already accounted for in the gas demand model via temporal correlation.

The gas demand forecast in North Carolina with the temperature as a regressor is given in Figure *15*.

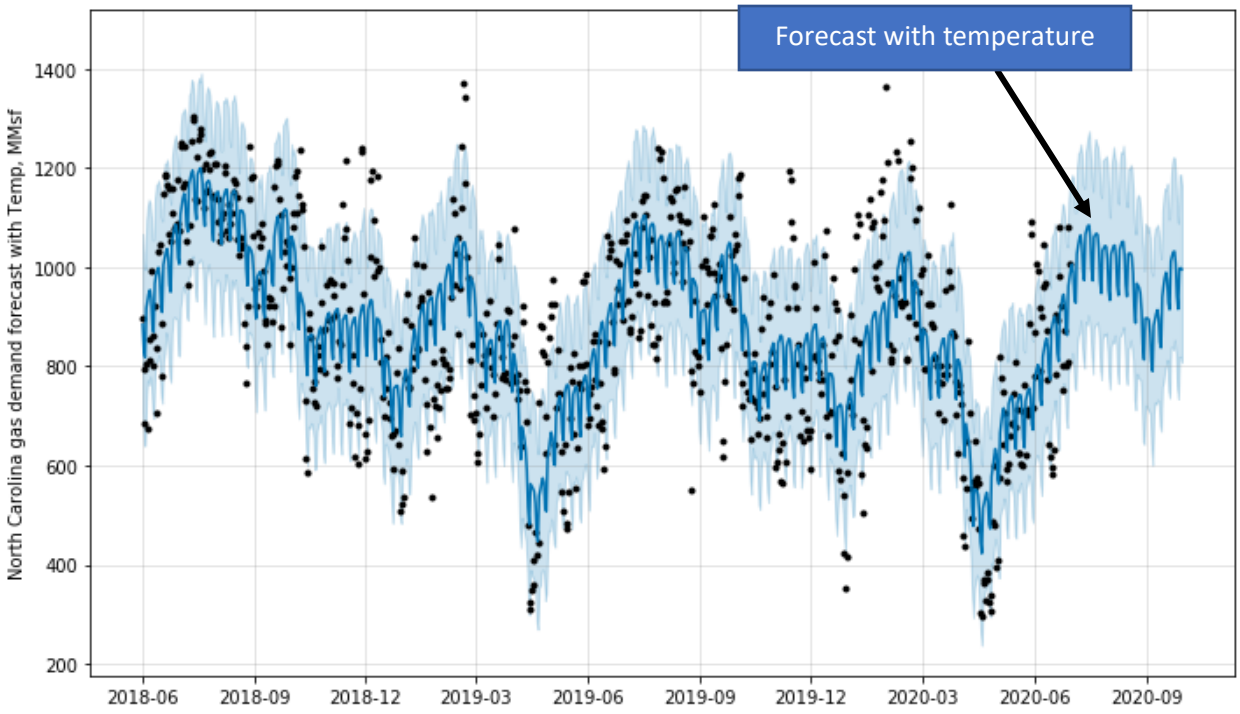


Figure 15. NC gas demand forecast [MMscf] without temperature as a feature. Actual data - black dots, forecast - solid blue line.

Below is a snippet of the model in Python fbprophet.

```
# Here I create new dataset to model with temperature surprise effects
df_dt = df_d
df_dt['Temp_surprise'] = train_dft['y']-new_dft['yhat']
df_dt.columns = ['ds', 'y', 'Temp_surprise']
print(df_dt.tail(10))

# I define the prediction size and create new training dataset
prediction_size = 92

train_df_dt = df_dt
print('train_df_dt.tail(5):')
print(train_df_dt.tail(5))

# Here I tell Prophet to make the prediction with defined periods and print
model_T_incl = Prophet(daily_seasonality=False)
model_T_incl.add_regressor('Temp_surprise')
print("... start fitting with Temperature surprise")
model_T_incl.fit(train_df_dt)
print("fit is done")
future_wTemp = model_T_incl.make_future_dataframe(periods=prediction_size)
future_wTemp['Temp_surprise'] = train_dft['y']-new_dft['yhat']
print(future_wTemp.tail(30))

new_df_dt = model_T_incl.predict(future_wTemp)
print(new_df_dt.info())
print(new_df_dt.tail(10))
print(new_df_dt.columns)
fig5 = model_T_incl.plot(new_df_dt)
axes = fig5.get_axes()
axes[0].set_xlabel('')
axes[0].set_ylabel('North Carolina gas demand forecast with Temp, MMsf')
fig6 = model_T_incl.plot_components(new_df_dt)
```

Temperature as regressor

Figure 16. NC gas demand model with temperature as a regressor snippet.

Finally, the analysis of the components (Figure *17*) supports my assumption that the temperature is not the main feature for the gas demand forecast.

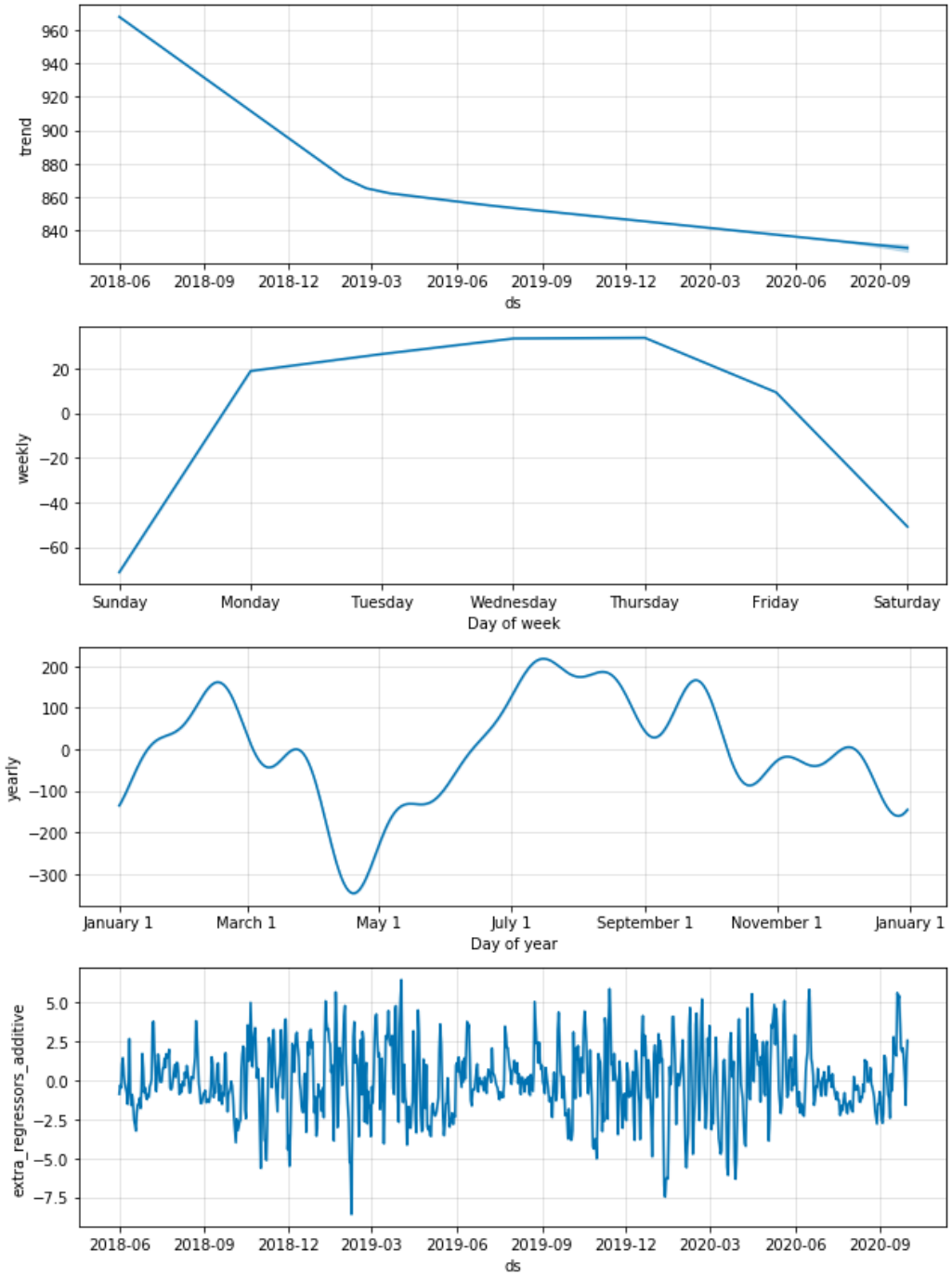


Figure 17. NC gas demand forecast model with temperature components

My numerical experiments have shown that temperature is not a significant feature because all temperature fluctuations are already included in the gas demand autocorrelation model. The temperature as a regressor only affects the demand at the amount of 2-2.5%, while the day of the year affects the demand +/- 40% seasonally and +/- 4% weekly.

Bonus questions

Task 2: Storage Value Assessment Task 2a

The first question asks how the ask-bid spreads for the price, risk-free rates, and injection/withdrawal constraints could impact the max profit (implied value of the storage facility).

First, I believe that **bid-ask Spread** will increase transaction costs to the trading strategy and decrease the max profit available (implied value of the storage facility). Consequently, it will:

1. decrease achievable profit for individual buy-sell pair;
2. reduce the number of economical transactions since all recommended transactions with initial theoretical (assuming zero Spread) profit gain below the bid-ask Spread will become uneconomical.

So, two effects combined will decrease the total profit more than:

$$dMaxProfit < - Spread*N*Capacity \quad (2)$$

where Spread = bid-ask spread ($), N – number of transactions in the original strategy (Task 2), and Capacity – storage capacity (200 MMcf).

Second, **a risk-free rate** above 0% will allow investing cash on balance instead of buying gas. So again, some transactions will become inefficient (in economic profit terms = comparing to deposit cash at a risk-free rate). The impact will be like the bid-ask Spread, but the individual deal storage cost will be proportional to gas price and exp(time_of_storage*risk_free_rate) for each transaction.

Finally, **injection/withdrawal constraints** will reduce our ability to benefit from an exact knowledge of future prices since we can buy or sell only a limited amount of gas a day. Instead, we will buy and sell gas in the earlier and next days from local extremums. That problem can be effectively solved by linear programming techniques honoring all constraints.

Task 2: Storage Value Assessment Task 2b

Task 2b: How does a super-contango gas forward structure impact the value of the storage facility? Super contango means that the spot price for a commodity is significantly below the futures price. Super contango usually occurs when storage I scarce due to excess supply - meaning that the cost of storage and profit from the buy-and-hold strategy is increasing. The value of empty storage capacity will increase the stronger super contango becomes.

Task 3. Data Analysis Task 3a:

You can also try to include other variables in the power supply stack, such as nuclear outage and gas price data. We have provided daily Henry Hub prompt prices (HenryHubPrompt.csv) in the attachment. You are also encouraged to use other third-party data, such as the nuclear power reactor outage data, which can be retrieved on the NRC website: https://www.nrc.gov/reading-rm/doccollections/event-status/reactor-status/index.html . Please provide your code along with commented lines to explain your methodology and discuss potential issues.

Task 3. Data Analysis Task 3b:

Illustrate how COVID-19 impacted North Carolina gas-for-power demand. You may have to choose the calibration period that excludes the COVID-19 impact period.

References

Gerber, F., De Jong, R., Schaepman, M.E., Schaepman-Strub, G., Furrer, R., 2018. Predicting Missing Values in Spatio-Temporal Remote Sensing Data. IEEE Trans. Geosci. Remote Sens. 56, 2841–2853. https://doi.org/10.1109/TGRS.2017.2785240

Taylor, S.J., Letham, B., 2017. Business Time Series Forecasting at Scale. PeerJ Prepr. 35, 48–90.