1 Introduction

This project's main focus and ambition was to develop tools to analyze data. The data in question was a set of measurements on air quality data for the year of 2014 to 2015 from the Grensásvegur 15 station. The data was taken from the station's website and is registered at 30 minute intervals throughout the whole period, with some exceptions that might be due to instrument failure or something else.

With this in mind, a dashboard was created which consists of two streaming charts, one of which contained data on NO_2 and H_2S levels while the other contained data on PM_{10} and $PM_{2.5}$ levels. These chemicals and particulates are thought to be correlated with traffic as they are produced by fossil fuel burning vehicles and spiked winter tyres. Seeing that the data was registered at such regular intervals it was decided to try and develop something which might resemble the effect of looking at the data while streaming. This was thought to be characteristic for the data and its main purpose as people vulnarable to bad air quality could benefit from having access to this kind of information. Furthermore it was thought to be plausible that this kind of a tool might be useful for analyzing historical data. Thus it was decided to make an attempt at creating a graphical animation out of air quality levels.

Seeing that car traffic and winter tyres are suspected to be the main cause of bad air quality it was decided that an important question to answer would be whether this would reflect as increased concentration of toxic levels depending on main traffic hours and winter time. In order to answer this question each month in the data set was analyzed by looking at the mean value for each time of measurement throughout the whole month. That is all the measurements that take place at the same HH:MM throughout the month in question are combined in a subset of the data set from which the mean value was calculated. From this an average day throughout the month was obtained, and thus the possibility of looking at the date in relation to traffic hours and winter time with the same tool.

In order to highlight the traffic hours, radiation levels measured were used,as most humans usually prefer driving during the day it is often bright during traffic hours(at least there are rarely traffic jams during the night). This was integrated into the graphics by making the background color of the animation a function of the radiation measured. A future extension of this tool could possibly integrate weather data in a similar way.

For the easiest comparison it was decided to put the ratio between the measured values and the severe health risk limit of measured quantities. Furthermore a horizontal line was drawn to mark the recommended health limit of measured quantities according to Icelandic authorities. These limits are different depending on authorities and thus this tool could potentially aid in making a decision when defining the proper health limits. In fact it is surprising to see that the particulates levels are rather high throughout the whole year.

The tool was made so that it can be easily modified to take other variables into account or to animate anything else with respect to quantities and time. It provided a clear picture of the measured quantities of dangerous toxics that that are a health risk to the public and thus should be carefully monitored. A further extesion to this tool could be to smoothen the plot and make the animation smoother throughout it's duration.

The Python packages Pandas, Datetime, dateutil and Matplotlib were the main packages that were used to develop the tool of analysis. Pandas is a very powerful when handling big data, matplotlib is the plotting package and Datetime and dateutil provide very smart ways of manipulating data which is based on timeseries.

2 Code

```
import numpy as np
from matplotlib.pylab import *
import matplotlib.animation as animation
from mpl_toolkits.axes_grid1 import host_subplot
from datetime import *
from matplotlib.colors import cnames
from dateutil.relativedelta import *
import calendar
import pdb
import matplotlib.dates as mdates
import va_read_data as parse

filename = 'air_data.csv'
# We read in the data
columns = parse.read_in(filename)
start = date(2014, 2, 1)
final = date(2015, 9, 1)
no2_data = parse.mean_rng(filename, start, final, 'NO2')
rad_data = parse.mean_rng(filename, start, final, 'Radiation')
h2s_data = parse.mean_rng(filename, start, final, 'H2S')
pm10_data = parse.mean_rng(filename, start, final, 'Dust_PM10')
pm25_data = parse.mean_rng(filename, start, final, 'Dust_PM2.5')
# We have to make a list of all the times animated
TimeAxis = []
# Count number of steps
n = 0
for month in no2_data:
  n = n + len(no2_data[month])
# Number of hours in each figure
nPerFrame = 24
# To build the x axis
for i in np.arange(0,n):
    TimeAxis.append(mdates.date2num(datetime.strptime(columns['Date'][i] + " " +
   columns['Time'][i],"%d.%m.%Y %H:%M")))


#---------------------------------- Set up the rc settings,
font = {'size' :9}
matplotlib.rc('font', **font)
#---------------------------------- Setup figure and subplots
f0 = figure(num = 0, figsize = (12,8))#, dpi = 100)
f0.suptitle('Measurements of air in Reykjavik', fontsize=12)

#-----------------------------Define the subplot/subplotgrid
ax01 = subplot2grid((2,2),(0,0), colspan=2, rowspan=1)
ax02 = subplot2grid((2,2),(1,0), colspan=2, rowspan=1)

#--------------------------------- Plot Settings
ax01.set_ylim(-0.1,3.5)
ax02.set_ylim(-0.1,5.0)

ax01.set_xlim(TimeAxis[0],TimeAxis[nPerFrame])
ax02.set_xlim(TimeAxis[0],TimeAxis[nPerFrame])

ax01.set_xlabel('Average Value at Time of Day During the Month')
ax02.set_xlabel('Average Value at Time of Day During the Month')

ax01.set_ylabel('Ratio between measured level and dangerous levels')
ax02.set_ylabel('Ratio between measured level and dangerous levels')
```

```python

ax01.set_xticks(TimeAxis[0:nPerFrame])
ax02.set_xticks(TimeAxis[0:nPerFrame])

ax01.xaxis.set_major_formatter(mdates.DateFormatter('%H:%M'))
ax02.xaxis.set_major_formatter(mdates.DateFormatter('%H:%M'))

Plot_limit1 = ax01.axhline(y = 75.0/30.0,color='r',lw = 2,label = 'Recommended Limit of NO2 Levels')
Plot_limit2 = ax02.axhline(y = 50.0/20.0, color='r',lw = 2, label = 'Recommended Limit of PM10 Levels')
# These vectors will be the ones used for the animation
time = []
pm10 = []
pm25 = []
no2 = []
h2s = []

# Set the plots
Plot_pm10, = ax02.plot_date(time, pm10, '-',color='#b58900', lw = 2, label = 'PM10')
Plot_no2, = ax01.plot_date(time, no2, '-',color='#b58900',lw = 2, label = 'NO2')
Plot_pm25, = ax02.plot_date(time, pm25, '-', color='g',lw = 2, label = 'PM2.5' )
Plot_h2s, = ax01.plot_date(time, h2s, '-', color='g', lw = 2, label = 'H2S')

# Set legends
ax02.legend([Plot_pm10, Plot_pm25, Plot_limit2], [Plot_pm10.get_label(), Plot_pm25.get_label(), Plot_limit2.get_label()])
ax01.legend([Plot_no2, Plot_h2s, Plot_limit1], [Plot_no2.get_label(), Plot_h2s.get_label(), Plot_limit1.get_label()])

# Format the x-axis for dates (label formatting, rotation)
f0.autofmt_xdate(rotation=45)
f0.tight_layout()
backgrCol = '#073642'
xmax = nPerFrame
x = 0
j = 0
month = date(2014,2,1)
maxRad = max(rad_data[month])
# animation function, this is called sequentially
def updateData(self):
    # We need to define global variables to run the loop through updateData
    global month
    global maxRad
    global x
    global j
    global time
    global pm10
    global alpha

    # We need to change months when we reach the end of the corresponding list
    if j == len(no2_data[month]):
        month = month + relativedelta(months=+1)
        j = 0
    # look at the maximum radiation for the month
        maxRad = max(rad_data[month])

    if x > xmax:
        # We can pop to save memory if x >= xmax we won't see those values any way
        time.pop(0)
```

```
        time.append(TimeAxis[x])
        no2.pop(0)
        no2.append(no2_data[month][j]*(1.0/30.0))
        pm10.pop(0)
        pm10.append(pm10_data[month][j]*(1.0/20.0))
        pm25.pop(0)
        pm25.append(pm25_data[month][j]*(1.0/20.0))
        h2s.pop(0)
        h2s.append(h2s_data[month][j]*(1.0/30.0))

        # Set the limits accordingly
        ax02.set_xlim(TimeAxis[x-xmax],TimeAxis[x+1])
        ax02.set_xticks(TimeAxis[x-xmax:x+1])

        ax01.set_xlim(TimeAxis[x-xmax],TimeAxis[x+1])
        ax01.set_xticks(TimeAxis[x-xmax:x+1])

        # The radiation function which is in charge of the background color
        radiation = np.abs(1.0 - rad_data[month][j]*1.0/maxRad)
        ax01.axvspan(TimeAxis[x],TimeAxis[x+1],facecolor=backgrCol,alpha=
   radiation,lw = 0)
        ax02.axvspan(TimeAxis[x],TimeAxis[x+1],facecolor=backgrCol,alpha=
   radiation,lw = 0)
    else:
        # This is run at the beginning, we don't want to pop here
        time.append(TimeAxis[x])
        no2.append(no2_data[month][j]*(1.0/30.0))
        pm10.append(pm10_data[month][j]*(1.0/20.0))
        pm25.append(pm25_data[month][j]*(1.0/20.0))
        h2s.append(h2s_data[month][j]*(1.0/30.0))
        # To set the background color
        radiation = np.abs(1.0 - rad_data[month][j]*1.0/maxRad)
        ax01.axvspan(TimeAxis[x],TimeAxis[x+1],facecolor=backgrCol,alpha=
   radiation,lw = 0)
        ax02.axvspan(TimeAxis[x],TimeAxis[x+1],facecolor=backgrCol,alpha=
   radiation,lw = 0)
    # The title is dynamic and tells the month
    ax01.set_title(month.strftime('%B, %Y'))
    Plot_no2.set_data(time,no2)
    Plot_pm10.set_data(time,pm10)
    Plot_pm25.set_data(time,pm25)
    Plot_h2s.set_data(time, h2s)
    # x and j must be incremented equally so that the dimensions remain the same!
    x = x+1
    j = j+1
    return Plot_no2, Plot_h2s, Plot_pm25, Plot_pm10

anim = animation.FuncAnimation(f0, updateData, frames=n, interval=10, blit=False,
    repeat=False)

anim.save('va_animation.mp4', fps=2, extra_args=['-vcodec','libx264'])

plt.show()
```

va_animate.py

```
#!/bin/python
import csv
import numpy as np
import pandas as pd
from datetime import *
from dateutil.relativedelta import *
```

```
import calendar
import pdb

def read_in(filename):

  with open(filename, 'rb') as csvfile:
    reader = csv.reader(csvfile, delimiter = '\t', quotechar = '|')

    headers = reader.next()

    data = {}
    for header in headers:
      data[header] = []

    for row in reader:
      for header, value in zip(headers, row):
        data[header].append(value)


  return data

# This function yields a list of the mean values at each mesurement interval
# for a given date range
def mean_rng(filename, start,final, chemical):
  dateparse = lambda x: pd.datetime.strptime(x, '%d.%m.%Y %H:%M')
  air_data = pd.read_csv(filename, sep='\t', parse_dates=[['Date', 'Time']], date_parser = dateparse, index_col=['Date_Time'], comment='#')
      # Make a list that contains the half an hourly mesurement intervals
      rng0 = pd.date_range('00:30','23:30',freq = 'H')
      rng1 = pd.date_range('00:00','23:00',freq = 'H')
      TimeRange = rng1.union(rng0)

  # Make a list of months
      # This list will contain the data for the given chemical for all the items in DateRanges
      # The mean value of each list in DateRanges_data
      DateRange_mean = {}

  while start < final:
    end = start + relativedelta(months=+1) - relativedelta(days=+1)
        # This list will contain the half an hourly mesurement intervals as strings
        TimeStr = []
        # This list will contain the different date ranges according to the items in TimeStr
        Measure_range = []
    DateRange_mean.update({start: []})
        for i in np.arange(0,48):
            # strings of measurement times
            TimeStr.append(TimeRange[i].strftime('%H:%M'))
            # date range for a given measurement time
            Measure_range.append(pd.date_range(start.strftime('%m-%d-%Y') + ' ' + TimeStr[i],end.strftime('%m-%d-%Y') + ' ' + TimeStr[i], freq = '24H'))
            # makes lists of the data during the Date ranges
            DateRanges_data = pd.Series(air_data[chemical], index = Measure_range[i])
      DateRanges_data.tolist()
      DateRange_mean[start].append(DateRanges_data.mean())
        start = start + relativedelta(months=+1)
      return DateRange_mean
```

va_read_data.py