

▼ Multivariate time series: data analysis and preprocessing

▼ Dataset

The data for this assignment contains time series for hourly weather parameters and pollution in the Chinese city of Beijing. The total data set contains 5 years of measurements, of which the last year is to be used as a test set.

In this notebook, we show how to import and visualise the raw data and demonstrate some useful ways of preprocessing in order to make the features more useful as inputs to a neural network.

**** Warning:**** the provided feature transformations are plausible ones, for which we provide code to help students who are less proficient in coding. It is up to you to decide which features (original or transformed) you will use in your model(s).

No code for detrending is provided. Visual inspection shows no obvious global trend but possibly a weak seasonal trend in the labels (pollution). You may use detrending on the features and the labels if you choose (a hint is given below). However, as this is not the main focus of this assignment, it should probably not be the first thing you invest in. Also, make sure any trends are **ONLY** determined on the training data! This means that, for your final models (trained on train+val), you will use a second detrender that is fitted on that larger data set!

▼ Imports and Drive mounting

```
#imports and Drive mounting
try:
    # %tensorflow_version only exists in Colab.
    %tensorflow_version 2.x
except Exception:
    pass

# TensorFlow and tf.keras
import tensorflow as tf
```

```

from tensorflow import keras

print(tf.__version__)


# Helper libraries
import numpy as np
import matplotlib.pyplot as plt
import sklearn as sk
import pandas as pd
from pandas import read_csv
from datetime import datetime
import math
import os

# fix random seed for reproducibility
seed = 2020
np.random.seed(seed)

from sklearn.model_selection import train_test_split

from tensorflow.keras.datasets import cifar100
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Flatten, BatchNormalization, Activation
from tensorflow.keras.layers import Conv1D, MaxPooling1D, GlobalMaxPooling1D
from tensorflow.keras.layers import LSTM, GRU
from tensorflow.keras.regularizers import L1L2
from tensorflow.keras.constraints import max_norm
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
from tensorflow.keras.models import load_model
from tensorflow.keras import optimizers

```

 TensorFlow 2.x selected.
 2.1.0

```

from google.colab import drive

```

```
drive.mount('/content/gdrive')
```

```
!ls '/content/gdrive/My Drive/Colab Notebooks/DL2020/GA3/'
```

🔗 Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.a

Enter your authorization code:

.....

Mounted at /content/gdrive

checkpoints	preprocessed_train_data_all.csv
pollution_data.csv	preprocessed_train_data.csv
preprocessed_pollution_data.csv	preprocessed_val_data.csv
preprocessed_test_data.csv	

▼ Reading the data set

The data is provided as a ".csv" file along with the getting started notebooks on Ufora. The easiest way to use it is to put it in your Drive folder and import it from there.

For reading 'csv' files, using the pandas library is the easiest. This gives you back the data as a pandas dataframe. The code below imports the data and visualises the raw time series.

```
# load dataset
```

```
# The PATH setting below assumes you just uploaded the data file to your Colab session
```

```
# When using Drive: replace this by the path where you put the data file
```

```
DATAPATH = '/content/gdrive/My Drive/Colab Notebooks/DL2020/GA3/'
```

```
DATAFILE = DATAPATH+'pollution_data.csv'
```

```
dataset = read_csv(DATAFILE, header=0, index_col=0)
```

```
dataset.head()
```



	year	month	day	hour	pollution	dew	temp	press	wnd_dir	wnd_spd	snow	rain
24	2010	1	2	0	129	-16	-4.0	1020.0	SE	1.79	0	0
25	2010	1	2	1	148	-15	-4.0	1020.0	SE	2.68	0	0
26	2010	1	2	2	159	-11	-5.0	1021.0	SE	3.57	0	0
27	2010	1	2	3	181	-7	-5.0	1022.0	SE	5.36	1	0
28	2010	1	2	4	138	-7	-5.0	1022.0	SE	6.25	2	0

```

values = dataset.values

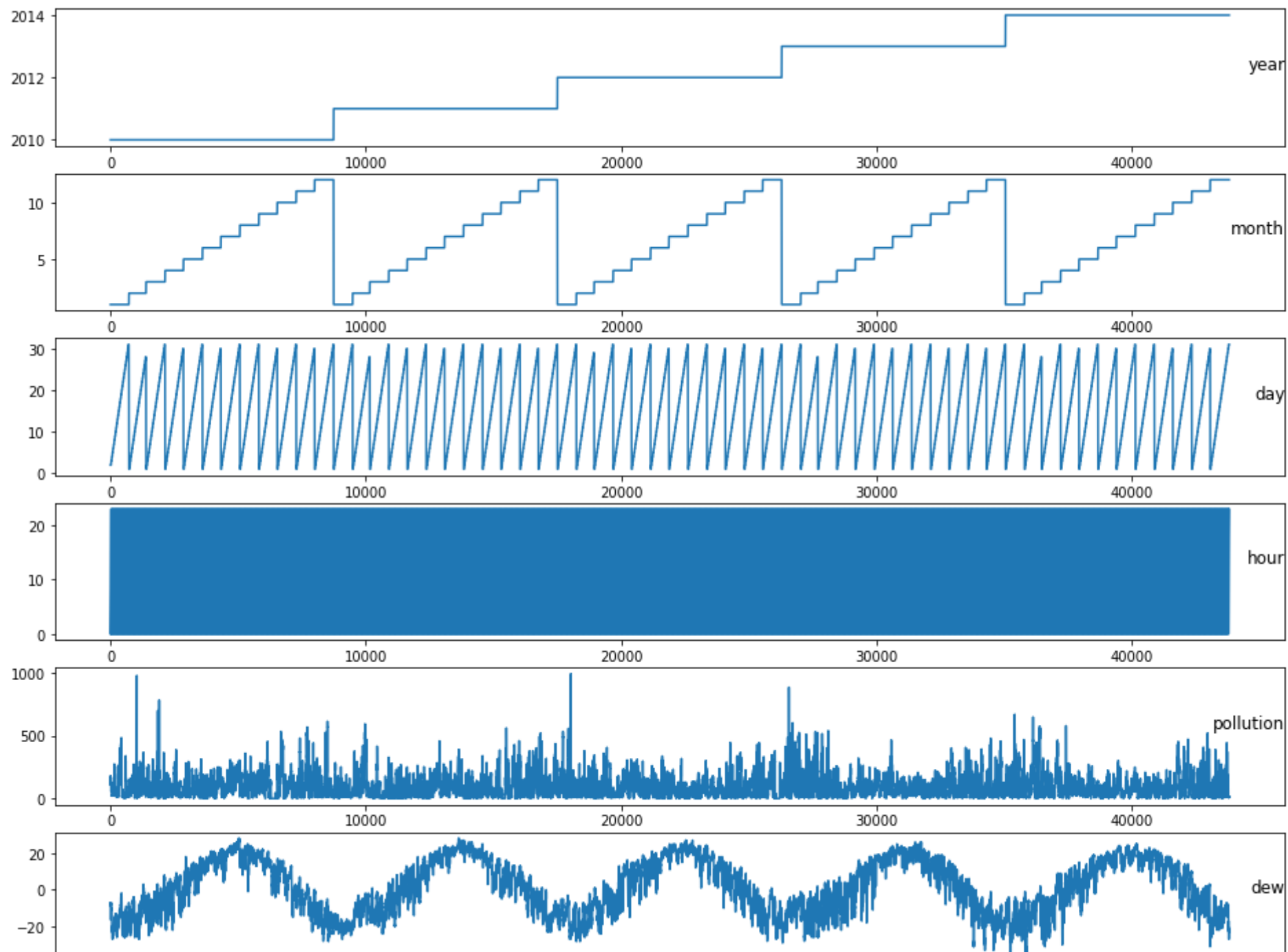
# plotting the first k samples of the raw time series
# setting below for all samples
samples = values.shape[0]
# setting below for smaller number of samples to "zoom in", e.g., 1000
#samples = 1000

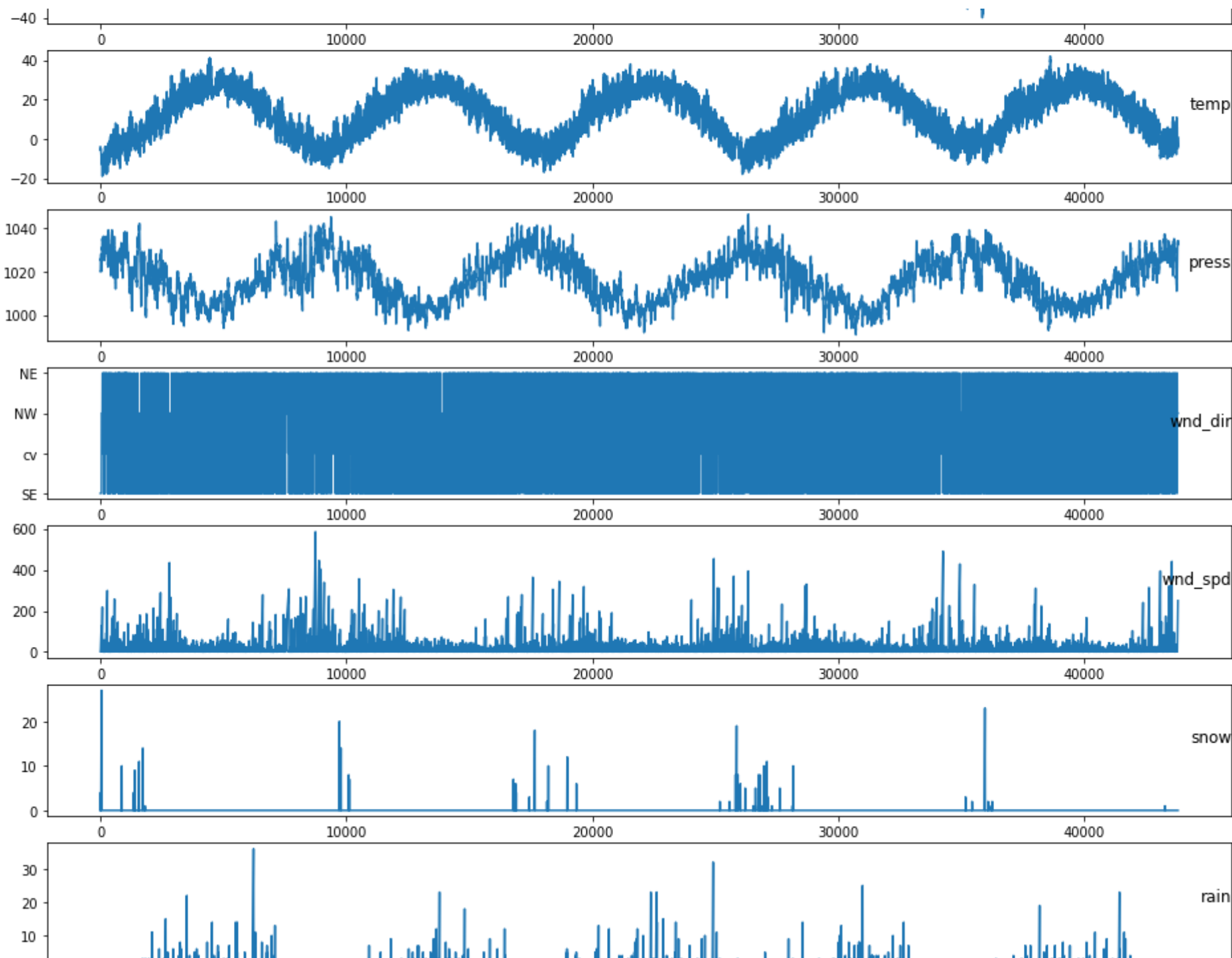
print("Dataset has ",values.shape[0]," samples and ",values.shape[1], " features")
# specify columns to plot
groups = [0,1,2,3,4,5,6,7,8,9,10,11]
i = 1
# plot each column
plt.figure(figsize=(16,26))
for group in groups:
    plt.subplot(len(groups), 1, i)
    plt.plot(values[:samples, group])
    plt.title(dataset.columns[group], y=0.5, loc='right')
    i += 1
plt.show()

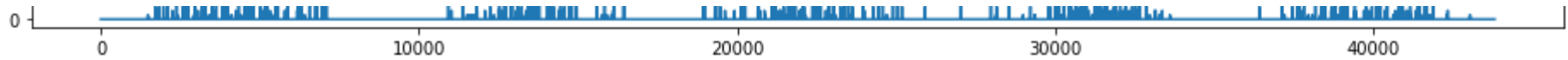
```



Dataset has 43800 samples and 12 features







▼ Discrete and categorical features

Most of the features are numeric. However, some of those numbers are discrete:

- year: 2010, 2011, 2012, 2013, 2014
- month: 1, ..., 12
- day: 1, ..., 31 (note that not all months have the same number of days)
- hour: 0, ..., 23

Wind directions are categorical values, coded as ['NE' 'NW' 'SE' 'cv'].

In order to decide what to do with those, we should think about the problem again (domain knowledge).

Year: We need to use the year in order to extract the test set further on. Aside from that, there are no indications that the pollution is systematically rising, so it will probably not be useful for detrending or as an input feature to the model.

Month: There is a vague seasonal trend in the pollution, so knowing the month is definitely could be useful. However, we can see clear seasonal trends in some of the weather parameters (dew, temperature and pressure), so the month information may already be sufficiently present in these features. This means that in a first attempt, you could probably ignore the month feature.

If you do use the month, its current 'encoding' does not match very well with the observed seasonal trends, which look like shifted sine signals. Below, we show how you can re-encode the months into sine and cosine signals, with which you can approximate any shifted sinusoidal signal with the same period. You could try to use these to detrend the 'periodic' features (trend is a linear combination of sine, cosine and a constant value). Either you can also detrend the labels, or you can input sine and cosine as separate features.

Day: The day of the month does not seem very relevant to predict pollution, but the day of the week could be. Below, we provide code examples with which you can extract the day of the week and convert it to one-hot code in two ways: one way encodes each day separately, the other only discriminates between weekend and week days. In Belgium, the second option would be very relevant, but since we don't know commuter patterns in Beijing, it may be better to keep all days. You could also plot statistical data for each weekday to check whether there is any significant difference between different days of the week.

Hour: Time of day is probably relevant. You could use it as it is or you could consider a similar approach to what we did with the month: transforming into periodic sine and cosine signals and use these as features instead. Again, investigating the statistics for different times of day can help you decide.

▼ Season

```
# statistical data per season
# statistical data for month of the year
# month of year is already given in the dataset

avgs = np.zeros((12))
stds = np.zeros((12))
for i in range(1,12+1):
    pollution_by_month_of_year = np.array(values[values[:,2]==i,4], dtype=float)
    avgs[i-1] = np.average(pollution_by_month_of_year)
    stds[i-1] = np.std(pollution_by_month_of_year)

t = np.arange(12)
plt.figure(figsize=(16,4))
plt.subplot(1,2,1)
plt.plot(avgs)
plt.xticks(t, ('Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec'))
plt.subplot(1,2,2)
plt.plot(avgs)
plt.fill_between(t, avgs+stds, avgs-stds, facecolor='blue', alpha=0.25)
plt.xticks(t, ('Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec'))
plt.show()

# statistical data for week of the year
week_of_year = np.arange(values.shape[0])//((24*7)%365 + 1)

avgs = np.zeros((52))
stds = np.zeros((52))
for i in range(1,52+1):
    pollution_by_day_of_year = np.array(values[week_of_year==i,4], dtype=float)
```



```

avgs[i-1] = np.average(pollution_by_day_of_year)
stds[i-1] = np.std(pollution_by_day_of_year)

t = np.arange(1, 52+1, step=4.214)
plt.figure(figsize=(16,4))
plt.subplot(1,2,1)
plt.plot(avgs)
plt.xticks(t, ('Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec', 'Jan'))
plt.subplot(1,2,2)
plt.plot(avgs)
plt.fill_between(np.arange(52), avgs+stds, avgs-stds, facecolor='blue', alpha=0.25)
plt.xticks(t, ('Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec', 'Jan'))
plt.show()

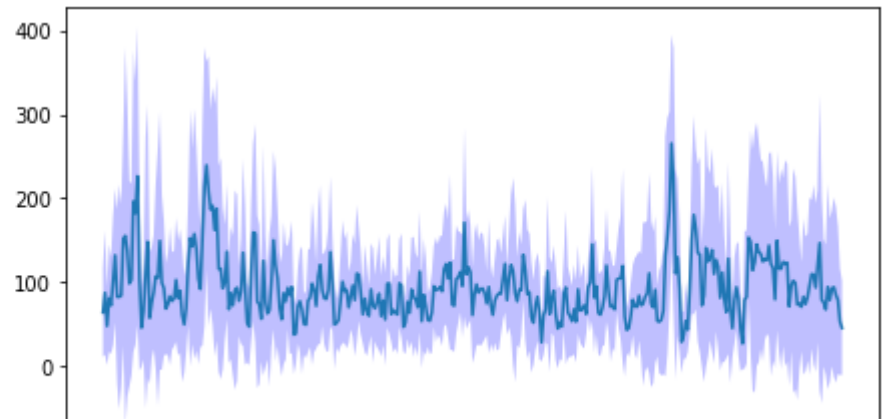
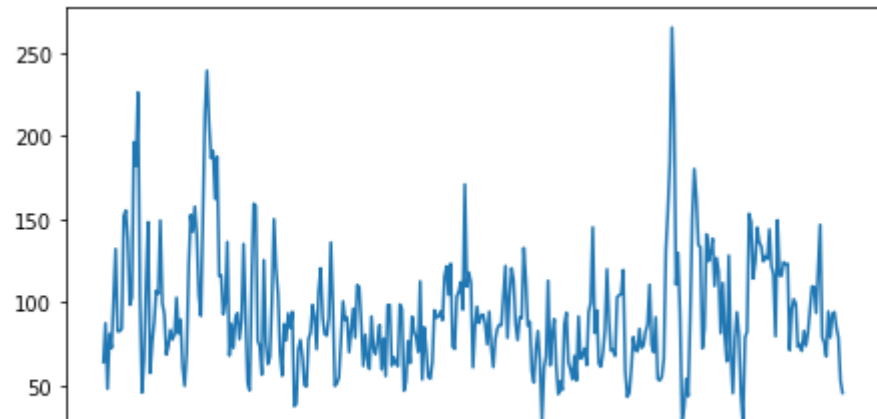
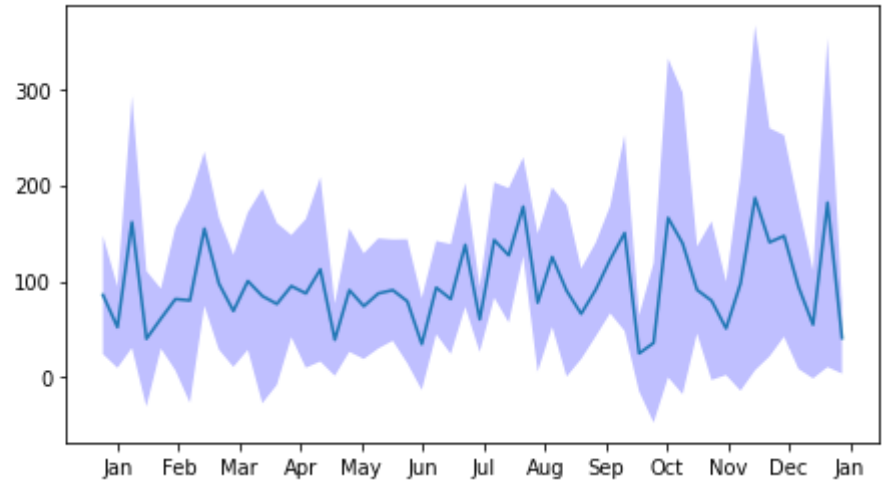
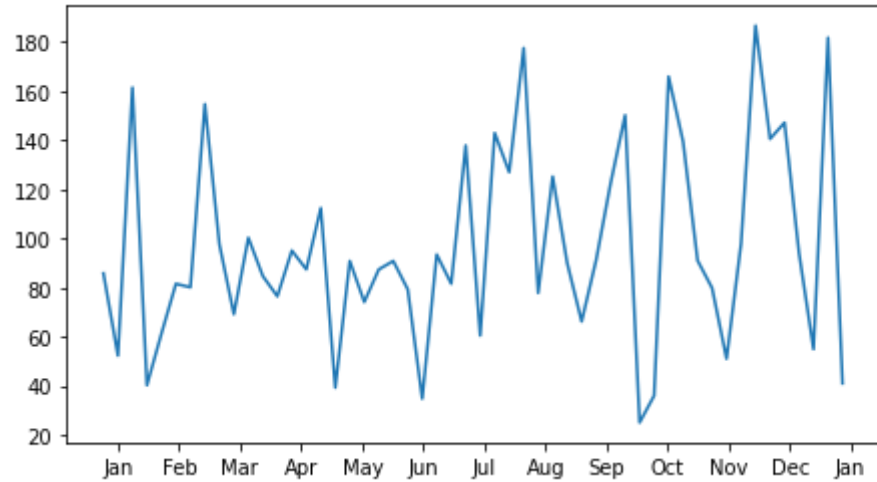
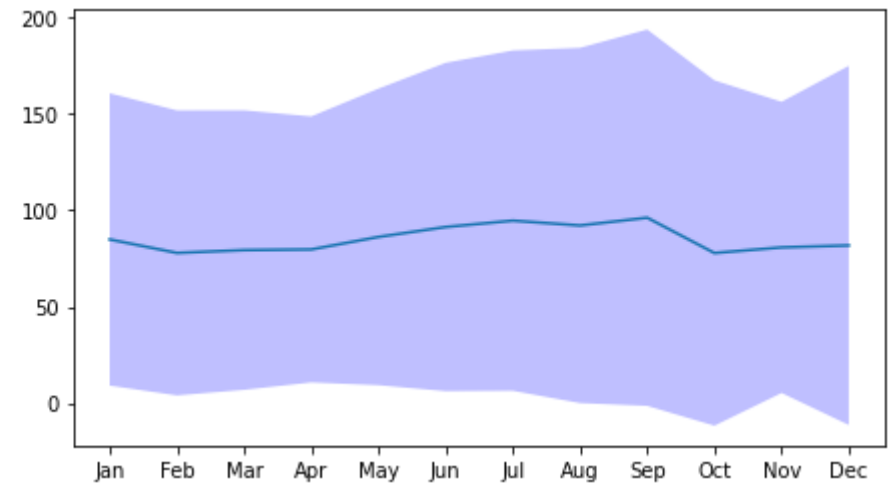
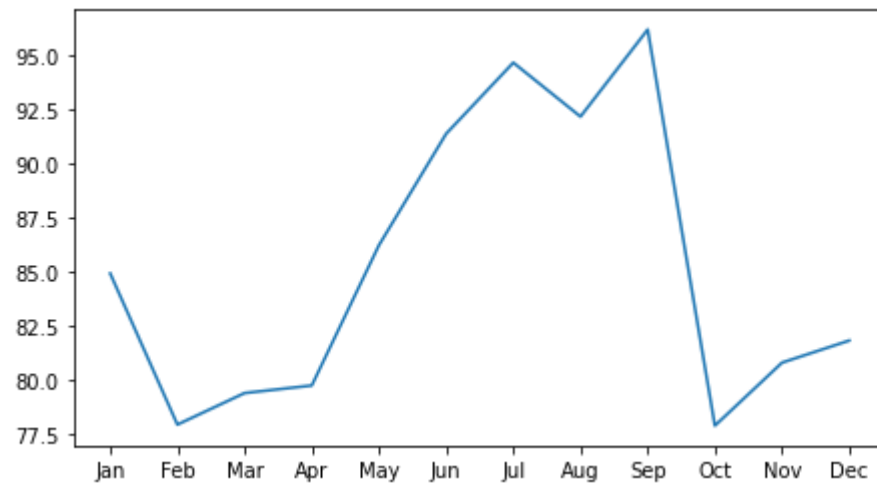
# statistical data for day of the year
day_of_year = np.arange(values.shape[0])//24%365 + 1

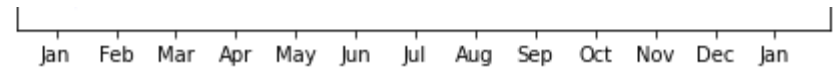
avgs = np.zeros((365))
stds = np.zeros((365))
for i in range(1,365+1):
    pollution_by_day_of_year = np.array(values[day_of_year==i,4], dtype=float)
    avgs[i-1] = np.average(pollution_by_day_of_year)
    stds[i-1] = np.std(pollution_by_day_of_year)

t = np.arange(1, 365+1, step=29.5)
plt.figure(figsize=(16,4))
plt.subplot(1,2,1)
plt.plot(avgs)
plt.xticks(t, ('Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec', 'Jan'))
plt.subplot(1,2,2)
plt.plot(avgs)
plt.fill_between(np.arange(365), avgs+stds, avgs-stds, facecolor='blue', alpha=0.25)
plt.xticks(t, ('Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec', 'Jan'))
plt.show()

```







```
# Translating periodic discrete features into sines and cosines
monthp = values[:,1]*2.0*np.pi/12.0
month_sin = np.sin(np.array(monthp,dtype=np.float64))
month_cos = np.cos(np.array(monthp,dtype=np.float64))

# check for 5 years
plt.figure(figsize=(16,8))
plt.subplot(3,1,1)
plt.plot(values[:,1:2])
```

```

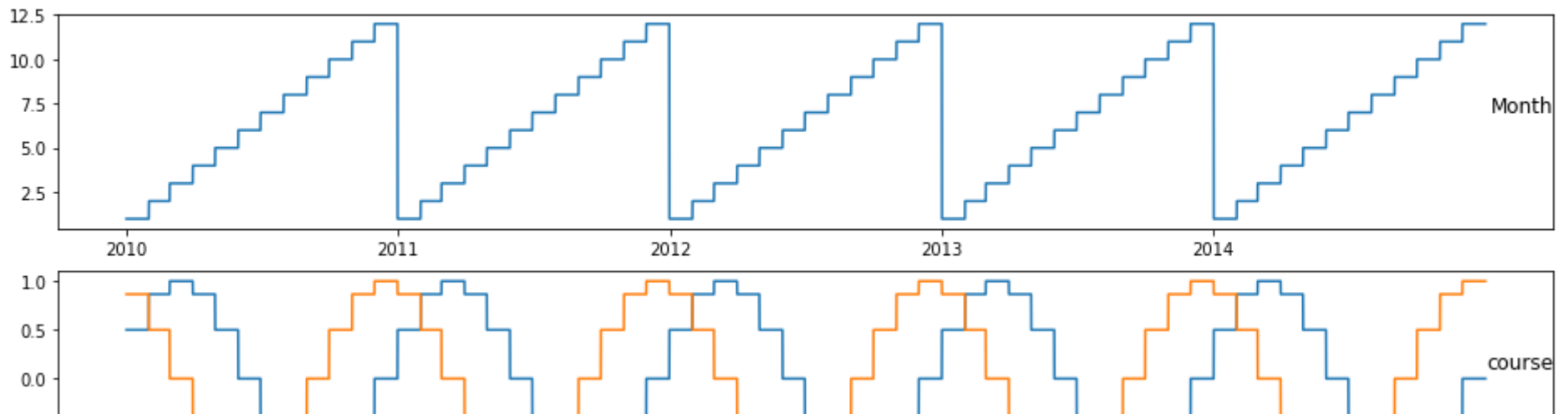
plt.title('Month', y=0.5, loc='right')
plt.xticks(np.arange(365*24*5, step=365*24), ('2010', '2011', '2012', '2013', '2014'))
plt.subplot(3,1,2)
plt.plot(month_sin)
plt.plot(month_cos)
plt.title('course', y=0.5, loc='right')
plt.xticks(np.arange(365*24*5, step=365*24), ('2010', '2011', '2012', '2013', '2014'))

# We see that the transformed month curves are not very smooth. In fact, it is probably better to ignore the months
# altogether and just use the day-of-the-year index to generate the periodic features
seasons = np.reshape(np.arange(values.shape[0])*2.0*np.pi/(365.0*24),(values.shape[0]))
seasons_sin = np.sin(np.array(seasons,dtype=np.float64))
seasons_cos = np.cos(np.array(seasons,dtype=np.float64))

# check for 5 years
plt.subplot(3,1,3)
plt.plot(seasons_sin)
plt.plot(seasons_cos)
plt.title('fine', y=0.5, loc='right')
plt.xticks(np.arange(365*24*5, step=365*24), ('2010', '2011', '2012', '2013', '2014'))
plt.show()

```





▼ Week

Day of the week: take into account that 02/01/2010 was a Saturday (you can look this up online)

```
day_of_week = np.zeros((values.shape[0]))
```

choose Monday to be day 1

```
hour = 0
```

day = 5 # Saturday is the 6th day of the week

```
for idx in range(values.shape[0]):
```

```
    day_of_week[idx] = day + 1
```

```
    hour = (hour + 1) % 24
```

```
    if hour == 0:
```

```
        day = (day + 1) % 7
```

```
weekdays = np.ones_like(day_of_week)
```

```
weekdays[np.where(day_of_week>5)] = 0.0
```

one-hot encode days of the week

```
mondays = np.zeros_like(day_of_week)
```

```
mondays [np.where(day_of_week==1)] = 1.0
```

```
# tuesdays = np.zeros_like(day_of_week)
```

```
# tuesdays [np.where(day_of_week==2)] = 1.0
```

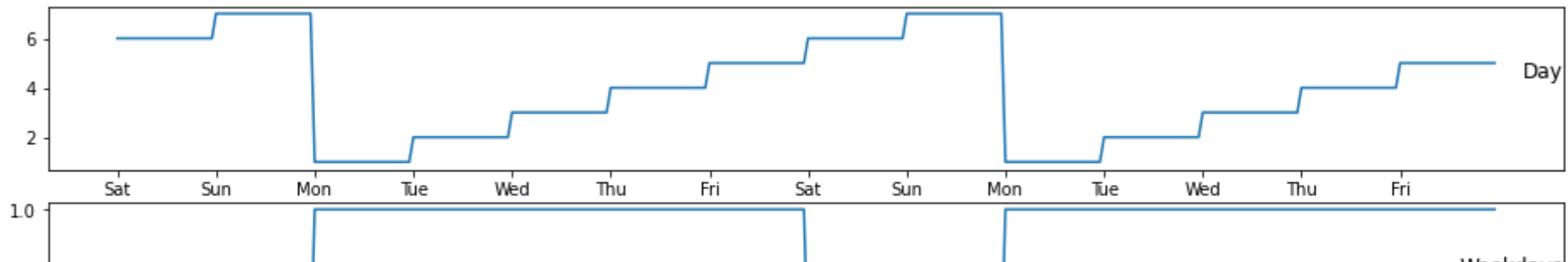
```

# wednesdays = np.zeros_like(day_of_week)
# wednesdays [np.where(day_of_week==3)] = 1.0
# thursdays = np.zeros_like(day_of_week)
# thursdays [np.where(day_of_week==4)] = 1.0
# fridays = np.zeros_like(day_of_week)
# fridays [np.where(day_of_week==5)] = 1.0
# saturdays = np.zeros_like(day_of_week)
# saturdays [np.where(day_of_week==6)] = 1.0
# sundays = np.zeros_like(day_of_week)
# sundays [np.where(day_of_week==7)] = 1.0

# check for 2 weeks
plt.figure(figsize=(16,6))
plt.subplot(3,1,1)
plt.plot(day_of_week[:168*2])
plt.xticks(np.arange(168*2, step=24), ('Sat', 'Sun', 'Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun', 'Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun', 'Mon', 'Tue', 'Wed', 'Thu', 'Fri'))
plt.title('Day', y=0.5, loc='right')
plt.subplot(3,1,2)
plt.plot(weekdays[:168*2])
plt.xticks(np.arange(168*2, step=24), ('Sat', 'Sun', 'Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun', 'Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun', 'Mon', 'Tue', 'Wed', 'Thu', 'Fri'))
plt.title('Weekdays', y=0.5, loc='right')
plt.subplot(3,1,3)
plt.plot(mondays[:168*2])
plt.xticks(np.arange(168*2, step=24), ('Sat', 'Sun', 'Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun', 'Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun', 'Mon', 'Tue', 'Wed', 'Thu', 'Fri'))
plt.title('Mondays', y=0.5, loc='right')
plt.show()

```





```
# statistical data per week
# statistical data for day of the week
# tranformation for day of the week is already given before

avgs = np.zeros((7))
stds = np.zeros((7))
for i in range(1,7+1):
    pollution_by_day_of_week = np.array(values[day_of_week==i,4], dtype=float)
    avgs[i-1] = np.average(pollution_by_day_of_week)
    stds[i-1] = np.std(pollution_by_day_of_week)

t = np.arange(7)
plt.figure(figsize=(16,4))
plt.subplot(1,2,1)
plt.plot(avgs)
plt.xticks(t, ('Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun'))
plt.subplot(1,2,2)
plt.plot(avgs)
plt.fill_between(t, avgs+stds, avgs-stds, facecolor='blue', alpha=0.25)
plt.xticks(t, ('Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun'))
plt.show()

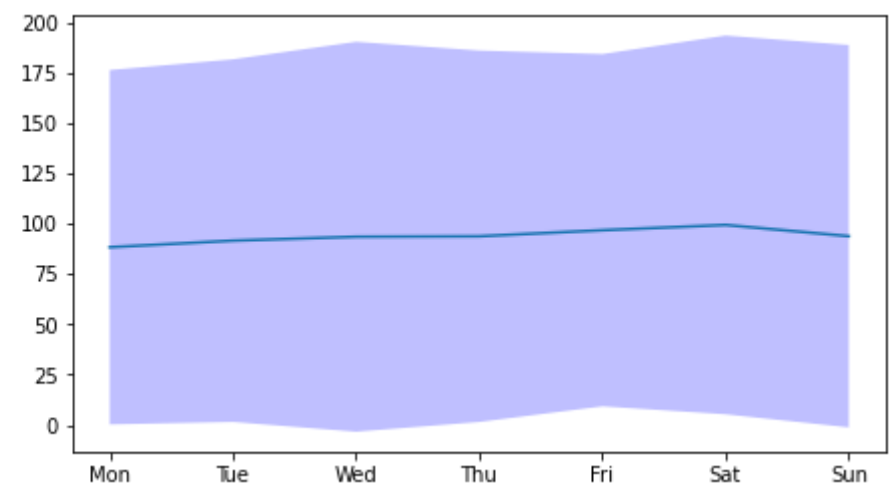
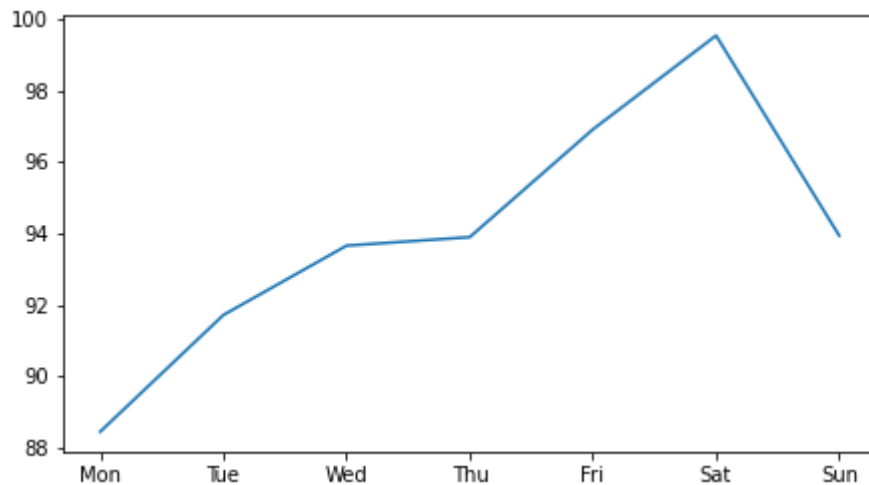
# statistical data for hour of the week
hour_of_week = (day_of_week - 1)*24 + values[:,3]

avgs = np.zeros((168))
stds = np.zeros((168))
for i in range(168):
    pollution_by_hour_of_week = np.array(values[hour_of_week==i,4], dtype=float)
```

```
pollution_by_hour_of_week = np.array(values[hour_of_week==1,4], dtype=float)
avgs[i-1] = np.average(pollution_by_hour_of_week)
stds[i-1] = np.std(pollution_by_hour_of_week)

t = np.arange(168, step=24)
plt.figure(figsize=(16,4))
plt.subplot(1,2,1)
plt.plot(avgs)
plt.xticks(t, ('Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun'))
plt.subplot(1,2,2)
plt.plot(avgs)
plt.fill_between(np.arange(168), avgs+stds, avgs-stds, facecolor='blue', alpha=0.25)
plt.xticks(t, ('Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun'))
plt.show()
```



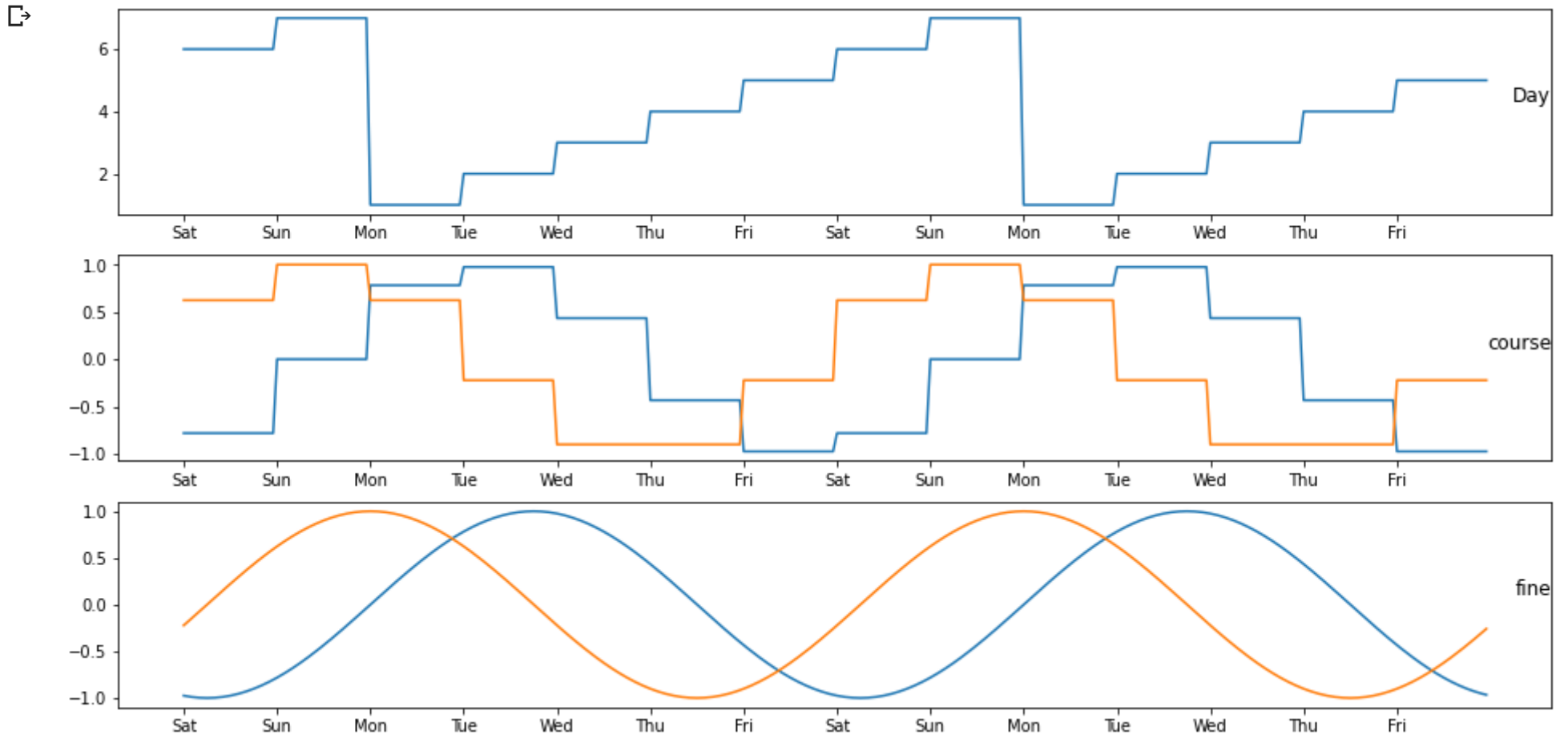


```
# Translating periodic discrete features into sines and cosines
day_of_weekp = day_of_week*2.0*np.pi/7.0
day_of_week_sin = np.sin(np.array(day_of_weekp,dtype=np.float64))
day_of_week_cos = np.cos(np.array(day_of_weekp,dtype=np.float64))

# check for 2 weeks
plt.figure(figsize=(16,8))
plt.subplot(3,1,1)
plt.plot(day_of_week[:168*2])
plt.title('Day', y=0.5, loc='right')
plt.xticks(np.arange(168*2, step=24), ('Sat', 'Sun', 'Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun', 'Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun', 'Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun'))
plt.subplot(3,1,2)
plt.plot(day_of_week_sin[:168*2])
plt.plot(day_of_week_cos[:168*2])
plt.title('course', y=0.5, loc='right')
plt.xticks(np.arange(168*2, step=24), ('Sat', 'Sun', 'Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun', 'Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun', 'Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun'))

# We see that the transformed month curves are not very smooth. In fact, it is probably better to ignore the day of week
# altogether and just use the hour-of-the-week index to generate the periodic features
hour_of_weekp = hour_of_week*2.0*np.pi/168
hour_of_week_sin = np.sin(np.array(hour_of_weekp,dtype=np.float64))
hour_of_week_cos = np.cos(np.array(hour_of_weekp,dtype=np.float64))
```

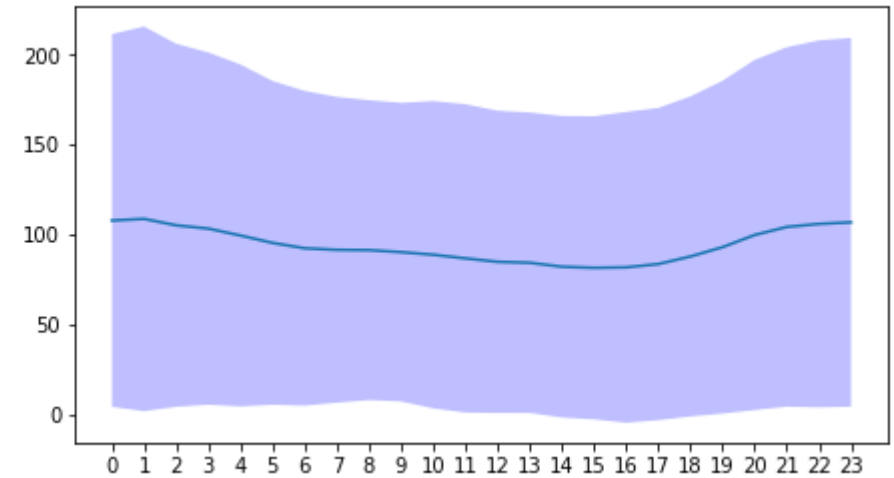
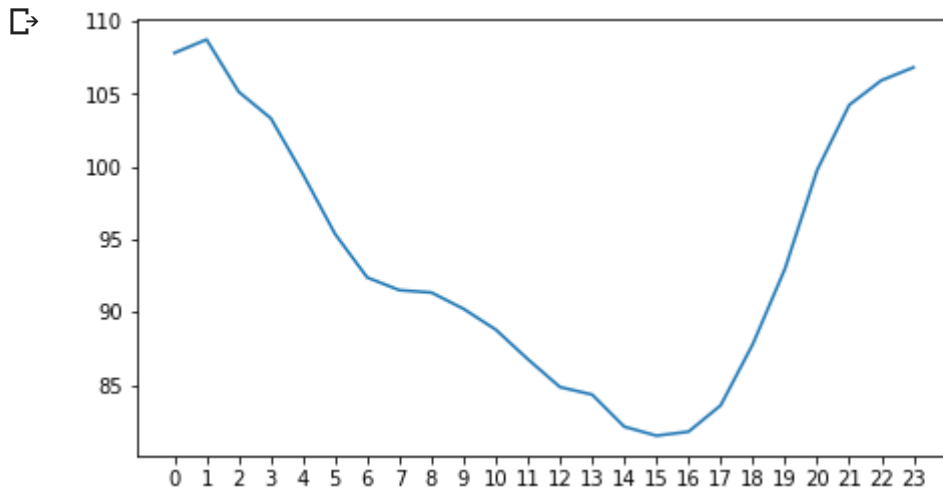
```
# check for 2 weeks
plt.subplot(3,1,3)
plt.plot(hour_of_week_sin[:168*2])
plt.plot(hour_of_week_cos[:168*2])
plt.title('fine', y=0.5, loc='right')
plt.xticks(np.arange(168*2, step=24), ('Sat', 'Sun', 'Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun', 'Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun', 'Mon', 'Tue', 'Wed', 'Thu', 'Fri'))
plt.show()
```



▼ Day

```
# statistical data per day
# statistical data for hour of the day
# hour of the day is already given in the dataset
avgs = np.zeros((24))
stds = np.zeros((24))
for i in range(24):
    pollution_by_hour_of_day = np.array(values[values[:,3]==i,4], dtype=float)
    avgs[i] = np.average(pollution_by_hour_of_day)
    stds[i] = np.std(pollution_by_hour_of_day)

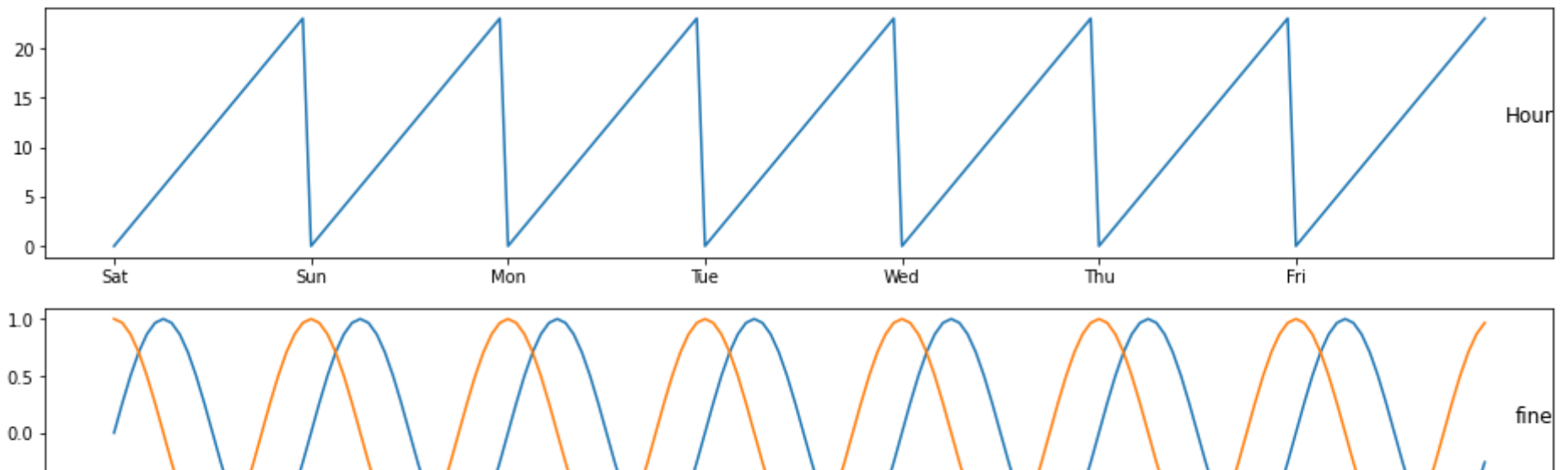
t = np.arange(24)
plt.figure(figsize=(16,4))
plt.subplot(1,2,1)
plt.plot(avgs)
plt.xticks(t)
plt.subplot(1,2,2)
plt.plot(avgs)
plt.fill_between(t, avgs+stds, avgs-stds, facecolor='blue', alpha=0.25)
plt.xticks(t)
plt.show()
```



```
# Translating periodic discrete features into sines and cosines
hour_of_dayp = values[:,3]*2.0*np.pi/24.0
hour_of_day_sin = np.sin(np.array(hour_of_dayp,dtype=np.float64))
hour_of_day_cos = np.cos(np.array(hour_of_dayp,dtype=np.float64))

# check for 1 week
plt.figure(figsize=(16,6))
plt.subplot(2,1,1)
plt.plot(values[:168,3])
plt.title('Hour', y=0.5, loc='right')
plt.xticks(np.arange(168, step=24), ('Sat', 'Sun', 'Mon', 'Tue', 'Wed', 'Thu', 'Fri'))
plt.subplot(2,1,2)
plt.plot(hour_of_day_sin[:168])
plt.plot(hour_of_day_cos[:168])
plt.title('fine', y=0.5, loc='right')
plt.xticks(np.arange(168, step=24), ('Sat', 'Sun', 'Mon', 'Tue', 'Wed', 'Thu', 'Fri'))
plt.show()
```





▼ Wind

We must now also convert the wind direction features to sensible numbers: possible options are integer, one-hot, 2D, polar, ...

From the original paper in which this dataset was introduced, we see that the three unique categories ['NE' 'NW' 'SE' 'cv'] are actually aggregates. The first three are bucketed according to the positions of large industrial sites around Beijing, whereas the fourth corresponds to "calm and variable":

Source: <https://royalsocietypublishing.org/doi/10.1098/rspa.2015.0257> :

Quote: "* ... northwest (NW), which includes W, WNW, NW, NNW and N; northeast (NE), for NNE, NE and ENE; southeast (SE), covering E, ESE, SE, SSE and S; southwest (SW), having SSW, SW and WSW; and calm and variable (CV).*"

From this perspective, one-hot seems to be the most sensible option for encoding these.

```
# Wind direction block

# creating two versions of converted feature sets for wind:
# - one-hot and wind speed separately (5 feature set)
# - 4 feature set in which ones of one-hot have been replaced with wind speed
```

```

# show unique wind direction values
winddirs = np.unique(values[:,8])
print(winddirs)

dirindices = values[:,8]

encoded_winddir = np.zeros((len(dirindices),4))
encoded_dirstrength = np.zeros((len(dirindices),4))

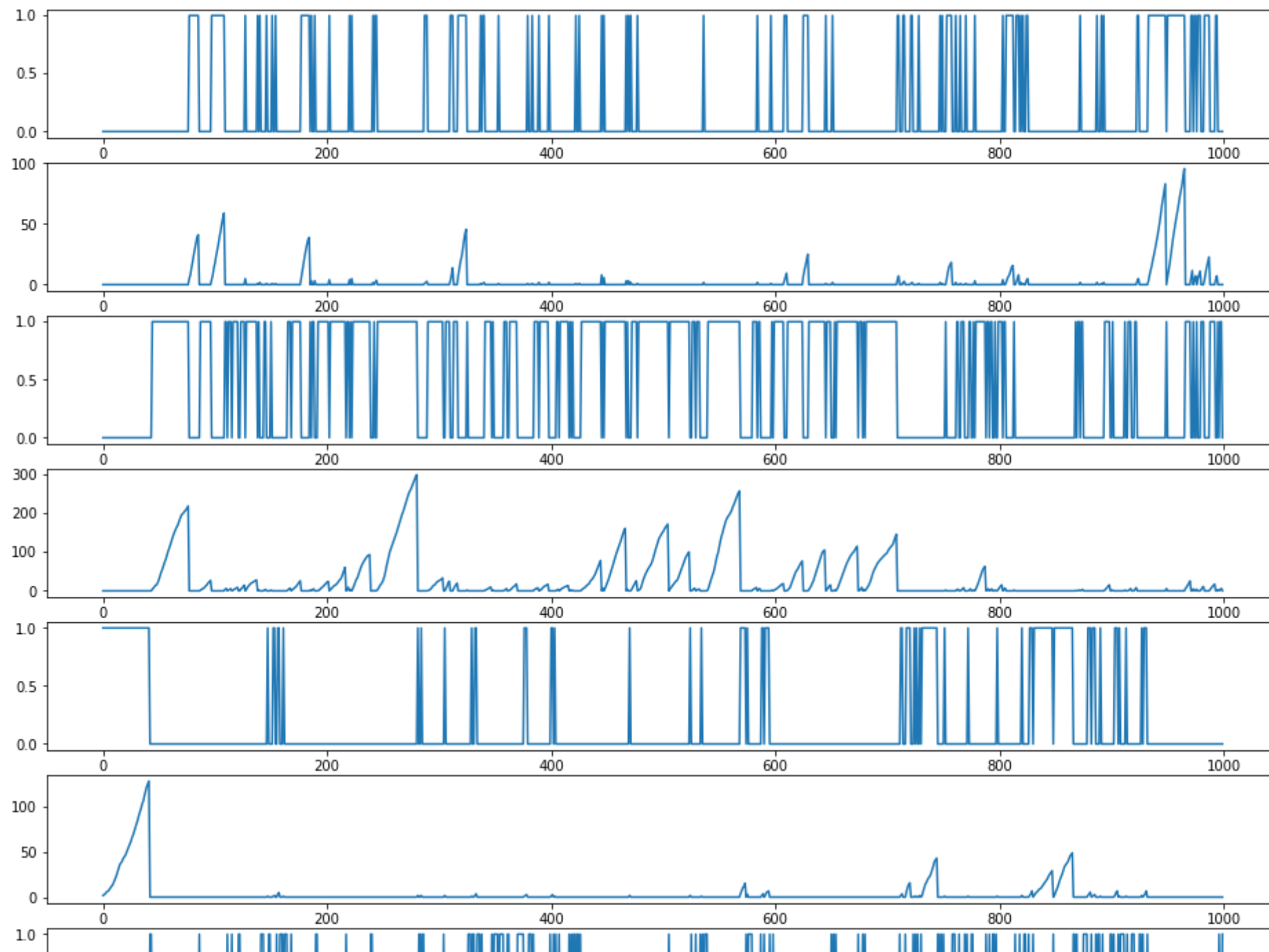
for idx in range(4):
    indices = np.where(dirindices == winddirs[idx])
    encoded_winddir[indices,idx] = 1.0
    encoded_dirstrength[indices,idx] = values[indices,9]

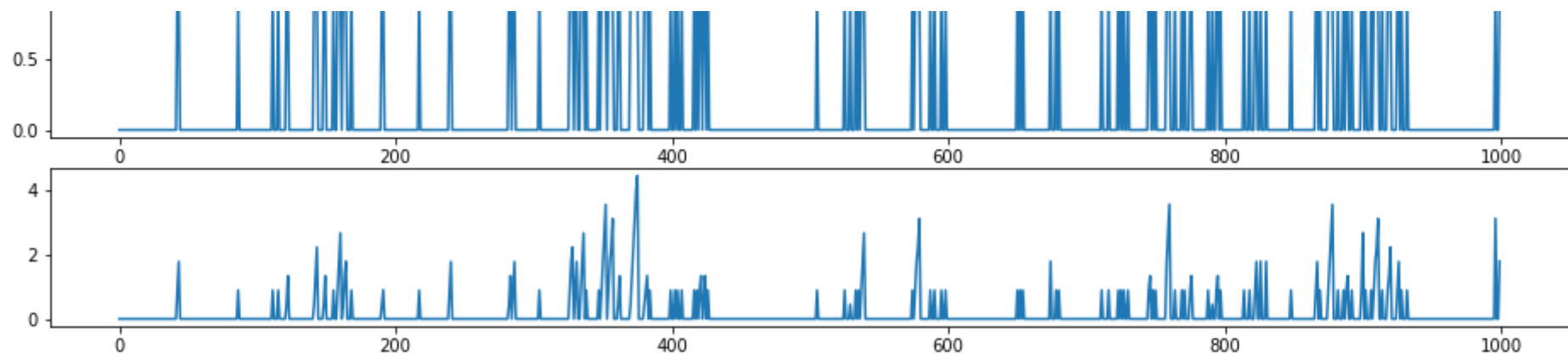
# check
# Each block of two plots corresponds to a single wind direction
# the top of both is the one-hot version, the bottom is one-hot*strength
plt.figure(figsize=(16,16))
for idx in range(4):
    plt.subplot(8, 1, 2*idx+1)
    plt.plot(encoded_winddir[:1000, idx])
    plt.subplot(8, 1, 2*idx+2)
    plt.plot(encoded_dirstrength[:1000, idx])
plt.show()

```



['NE' 'NW' 'SE' 'cv']






```
# use wind strength as separate features
NE_strength = encoded_dirstrength[:,0]
NW_strength = encoded_dirstrength[:,1]
NS_strength = encoded_dirstrength[:,2]
cv_strength = encoded_dirstrength[:,3]
```

▼ Combining Features

With the code above, you should be able to create the features you think are most useful. You can then join them together in a single feature matrix to use for training models:

```
# helper function
def combine_features(feature_list):
    features = np.concatenate(feature_list,axis=1)
    return features

# year
year = values[:,0:1]

# extract pollution features
fpoll = values[:,4:5]

# select and combine the features needed
seasons_sin = seasons_sin[:,np.newaxis]
seasons_cos = seasons_cos[:,np.newaxis]
hour_of_week_sin = hour_of_week_sin[:,np.newaxis]
hour_of_week_cos = hour_of_week_cos[:,np.newaxis]
hour_of_day_sin = hour_of_day_sin[:,np.newaxis]
hour_of_day_cos = hour_of_day_cos[:,np.newaxis]

# create wheather feature set
fweather = values[:,5:8]

# extract precipitation features (rain/snow)
fprecip = values[:, 10:11]
```

```

fwet = values[:,10.]

# combine all features
data = combine_features([
    # year
    year,
    # pollution
    fpoll,
    # season as sin and cos
    seasons_sin,
    seasons_cos,
    # hour of week as sin and cos
    hour_of_week_sin,
    hour_of_week_cos,
    # hour of day as sin and cos
    hour_of_day_sin,
    hour_of_day_cos,
    # wind strength with direction one-hot encoded
    encoded_dirstrength,
    # weather
    fweather,
    # precipitation
    fwet
])

print(data.shape)

```

➞ (43800, 17)

```

# you can also add the new features to the pandas dataframe and save that for later reuse
# just a few features added here for demonstration purposes
dataset = pd.DataFrame(data=data, columns=['Year',
    'Pollution',
    'seasons_sin',
    'seasons_cos',
    'hour_of_week_sin',
    'hour_of_week_cos',
    'hour_of_day_sin',
    'hour_of_day cos',

```

```

        'NE_strength',
        'NW_strength',
        'NS_strength',
        'cv_strength',
        'dew',
        'temp',
        'press',
        'snow',
        'rain'])

print(dataset.shape)
dataset.head()

```

↗ (43800, 17)

	Year	Pollution	seasons_sin	seasons_cos	hour_of_week_sin	hour_of_week_cos	hour_of_day_sin	hour_of_day_cos	NE_strength
0	2010	129	0	1	-0.974928	-0.222521	0	1	0
1	2010	148	0.000717259	1	-0.982566	-0.185912	0.258819	0.965926	0
2	2010	159	0.00143452	0.999999	-0.988831	-0.149042	0.5	0.866025	0
3	2010	181	0.00215177	0.999998	-0.993712	-0.111964	0.707107	0.707107	0
4	2010	138	0.00286903	0.999996	-0.997204	-0.0747301	0.866025	0.5	0

```

NEWDATAFILE = DATAPATH+'preprocessed_pollution_data.csv'
dataset.to_csv(path_or_buf=NEWDATAFILE)

#check
check = read_csv(NEWDATAFILE, header=0, index_col=0)
print(check.head()) # should be exactly the same as dataset.head() above

newvalues=check.values

print(newvalues.shape)

labels = check.columns

```

```
print(labels)
```

```
↗   Year  Pollution  seasons_sin  seasons_cos  ...  temp  press  snow  rain
0   2010         129    0.000000    1.000000  ...  -4.0  1020.0    0    0
1   2010         148    0.000717    1.000000  ...  -4.0  1020.0    0    0
2   2010         159    0.001435    0.999999  ...  -5.0  1021.0    0    0
3   2010         181    0.002152    0.999998  ...  -5.0  1022.0    1    0
4   2010         138    0.002869    0.999996  ...  -5.0  1022.0    2    0

[5 rows x 17 columns]
(43800, 17)
Index(['Year', 'Pollution', 'seasons_sin', 'seasons_cos', 'hour_of_week_sin',
      'hour_of_week_cos', 'hour_of_day_sin', 'hour_of_day_cos', 'NE_strength',
      'NW_strength', 'NS_strength', 'cv_strength', 'dew', 'temp', 'press',
      'snow', 'rain'],
      dtype='object')
```

```
labels[np.arange(2,17,1)]
```

```
↗   Index(['seasons_sin', 'seasons_cos', 'hour_of_week_sin', 'hour_of_week_cos',
      'hour_of_day_sin', 'hour_of_day_cos', 'NE_strength', 'NW_strength',
      'NS_strength', 'cv_strength', 'dew', 'temp', 'press', 'snow', 'rain'],
      dtype='object')
```

▼ Splitting into train, validation, and test sets

In order to avoid any problems with data leakage between train and test sets, it is safest to make the split from the very beginning. As stated: the last year is the test set.

```
# The last year is meant for testing, the year of the data is in the first column
testyear = max(newvalues[:,0])
print("Test set year is ", testyear)

# Find indices of all rows in test year
teststart = np.min(np.where(newvalues[:,0]==testyear)[0])
num_train_samples=teststart
num_test_samples=newvalues.shape[0]-teststart
```

```


print("There are ",num_train_samples," samples in the training years")
print("and ",num_test_samples," samples in the test year")

# extract train and test data
train_data_all = pd.DataFrame(data=newvalues[:teststart,:],columns=labels)
TRAINDATAALLFILE = DATAPATH+'preprocessed_train_data_all.csv'
train_data_all.to_csv(path_or_buf=TRAINDATAALLFILE)

test_data = pd.DataFrame(data=newvalues[teststart:,:],columns=labels)
TESTDATAFILE = DATAPATH+'preprocessed_test_data.csv'
test_data.to_csv(path_or_buf=TESTDATAFILE)

# check
test_data.head()

```

 Test set year is 2014.0
 There are 35040 samples in the training years
 and 8760 samples in the test year

	Year	Pollution	seasons_sin	seasons_cos	hour_of_week_sin	hour_of_week_cos	hour_of_day_sin	hour_of_day_cos	NE_strengt
0	2014.0	24.0	-9.797174e-16	1.000000	0.974928	-0.222521	0.000000	1.000000	0.0
1	2014.0	53.0	7.172585e-04	1.000000	0.965926	-0.258819	0.258819	0.965926	0.0
2	2014.0	65.0	1.434517e-03	0.999999	0.955573	-0.294755	0.500000	0.866025	0.0
3	2014.0	70.0	2.151774e-03	0.999998	0.943883	-0.330279	0.707107	0.707107	0.0
4	2014.0	79.0	2.869030e-03	0.999996	0.930874	-0.365341	0.866025	0.500000	0.0

```
# The last year is meant for validation, the year of the data is in the first column
train_data_all = train_data_all.values
valyear = max(train_data_all[:,0])
print("Validation set year is ", valyear)

# Find indices of all rows in test year
valstart = np.min(np.where(train_data_all[:,0]==valyear)[0])
num_train_samples=valstart
num_val_samples=train_data_all.shape[0]-valstart

print("There are ",num_train_samples," samples in the training years")
print("and ",num_val_samples," samples in the validation year")

# extract train and test data
train_data = pd.DataFrame(data=train_data_all[:valstart,:],columns=labels)
TRAINDATAFILE = DATAPATH+'preprocessed_train_data.csv'
train_data.to_csv(path_or_buf=TRAINDATAFILE)

val_data = pd.DataFrame(data=train_data_all[valstart:,:],columns=labels)
VALDATAFILE = DATAPATH+'preprocessed_val_data.csv'
val_data.to_csv(path_or_buf=VALDATAFILE)

# check
val_data.head()
```



Validation set year is 2013.0
 There are 26280 samples in the training years
 and 8760 samples in the validation year

	Year	Pollution	seasons_sin	seasons_cos	hour_of_week_sin	hour_of_week_cos	hour_of_day_sin	hour_of_day_cos	NE_strengt
0	2013.0	35.0	-7.347881e-16	1.000000	0.781831	0.623490	0.000000	1.000000	0.0
1	2013.0	31.0	7.172585e-04	1.000000	0.804598	0.593820	0.258819	0.965926	0.0
2	2013.0	32.0	1.434517e-03	0.999999	0.826239	0.563320	0.500000	0.866025	0.0
3	2013.0	31.0	2.151774e-03	0.999999	0.846724	0.532032	0.707107	0.707107	0.0

In addition, it is best to also have a validation set. Here you have 3 options:

1. You split off an additional year: this makes sure you validate accross all seasons, but then you only have three of the original 5 years left, which may be too little for tuning - you should expect to see an improvement between validate and test (after retrain) scores, though.
2. You split off only a small part, but then you tune your network only on its performance for the end of the year
3. you cut the whole dataset into non-overlapping windows and use a random subset for validation: this spreads you validation accross the seasons, but will contain more data leakage and will introduce problems if the data is not stationary. You could also write more complex code in which you discard any window between training and validation samples (cf. slides).

No approach is perfect. When using option 2, it would be good to check afterwards on the test data whether the errors are not systematically smaller on the part of the year on which you validated. Option three will have to be re-done for each model separately, after converting the training sequence into windows, since it depends on the window-width you use for training.

We leave it up to you how to choose. Since some approaches require more coding than others (i.e., less re-use of the provided code), we will accept that part of your choice is motivated by lack of coding experience. Make sure you motivate your decision and analyse the consequences.

This concludes our example code for data manipulation.

Assignment overview

In your last graded assignment, you will build a model to predict pollution in a big city on past pollution info as well as past wheather info. You receive a multi-variate time series data set which contains hourly data. **Your task is to predict pollution 6 hours ahead** in order to be able to build a citizen warning system. The error measure you need to minimise is Mean Absolute Error. You will **use the pollution samples from the last year for testing**. Obviously, it is also advisable to use a validation set.

Your assignment consists of three steps (please also check the instructions in the slides):

1) First make the following baseline model:

a model that only predicts pollution based on the current values of all input time series (pollution and wheather) to predict for 6 hours ahead, i.e. a purely feature based dense model that does not take history into account; you can also use this model for an initial investigation of the importance of each feature! Note that we already provide an even simpler baseline that simply uses the current pollution value as a prediction.

2) After this baseline, you will make two models that predict pollution 6 hours ahead based on **current pollution and pollution history**, so a single timeseries:

- a window-based 1D convolution network (the window size is an important hyperparameter here)
- a RNN network (LSTM and/or GRU)

3) Combine both into a model that uses current values and histories of multiple time series. You do not need to use all of them if they do not improve validation score. You will analyse the impact of including the different wheather features into your model (feature selection).

4) You will finally try to crank up your performance by making at least one ensemble. For example, for some features it may not be useful to use history information. You could consider buildin an ensemble of a sequential model with a dense model or of models trained on different subsets of the features.

Analysis

Since this is a regression task, there are no misclassifications: the analysis should focus on visualising the predictions on train and validate or test sets. Code for this is suggested in the training notebook. Please make sure to not only make the necessary plots, ut also to discuss them for the final model(s) in each stage.

Timing

This assignment is considerably more extensive than the previous ones! try to maximally use the experience you already gained to efficiently tune the networks. Also, once you find a network that is properly tuned, do not waste time on trying to get that extra small improvement. If you have time left after finishing all the tasks, you can still revisit your best model(s) to try and improve!

The deadline for this assignment is **Friday, March 20, 23:30**. This means you have 2 weeks: for a good planning, aim at finishing at least parts 1 and 2 in the first week! That way, you can use the hands-on support session to ask us whether you are on the right track.