Plan:

- make graphs from a .csv (a simpler version of speadsheet and askin to text file)
- use Python and Pandas library to clean data, explore and make simple graph
- work with built-in function to discover statistics

Before you start:

- this Jupyter notebook, code, and software were prepared using `Ubuntu 18.04LTS`, `Python3.6.9`
- If you are using Windows, or even a Mac, I recommend to jump on [Anaconda (https://www.anaconda.com/products/individual)](https://www.anaconda.com/products/individual) suite. Scroll to the bottom of the page, you will see the package for your system. Select `Graphic` option to make your life a bit easier (for now).

Prepare folder structure

- all data file is stored in `data` folder

In [1]:
```python
import os # to create folder, right click `Create Folder` works
```

In [3]:
```python
# current folder structure in top layer
os.listdir()
```

Out[3]:
```
['README.md',
 'Basic-display-data.ipynb',
 '.gitignore',
 '.git',
 '.ipynb_checkpoints']
```

In [4]:
```python
if 'data' not in os.listdir():
    os.makedirs('data')
else:
    print('folder named data existed')
```

In [7]:
```python
# also create `graph, img` folders
[os.makedirs(folder) for folder in ['graph', 'img'] if folder not in
os.listdir()] #list comprehension
```

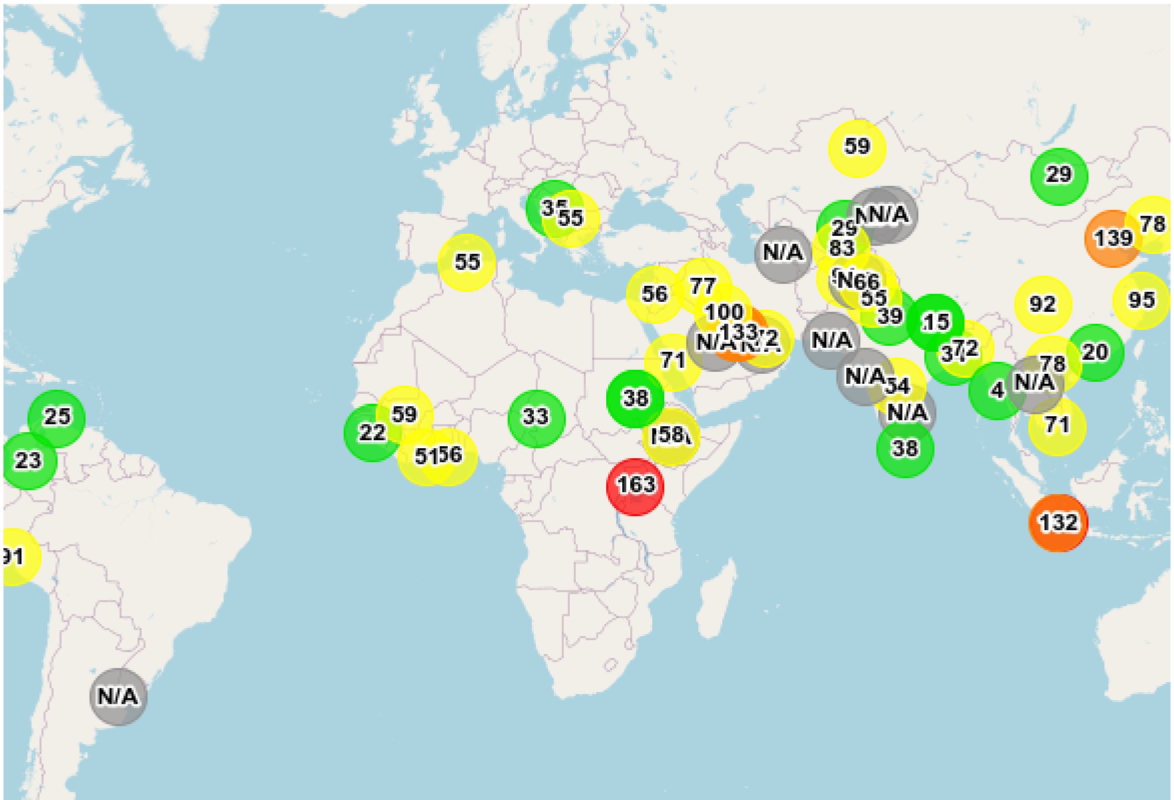
Out[7]:
```
[None, None]
```

In [8]:
```python
# check again and a folder name data existed
os.listdir()
```

Out[8]:
```
['img',
 'README.md',
 'data',
 'Basic-display-data.ipynb',
 'graph',
 '.gitignore',
 '.git',
 '.ipynb_checkpoints']
```

Download CSV files

- let work with `AirNow.gov` 's data archived by years and up-to-date.



- Click to one location (**Hanoi**), a list of CSV files under *Historical* tab blow the map

```
2015 PM2.5 MTD
2015 PM2.5
2016 PM2.5 MTD
2016 PM2.5
2017 PM2.5 MTD
2017 PM2.5
2018 PM2.5 MTD
2018 PM2.5
2019 PM2.5 MTD
2019 PM2.5
2020 PM2.5 MTD
2020 PM2.5 YTD
```

- and the link to a file `http://dosairnowdata.org/dos/historical/Hanoi/2016/Hanoi_PM2.5_2016_12_MTD.csv`
- [Ref: Airnow.gov (https://www.airnow.gov/international/us-embassies-and-consulates/)](https://www.airnow.gov/international/us-embassies-and-consulates/)

In [12]:
```
# let get year contain the whole year. For Hanoi, I selected 2018.
# Right click and Save As `data` folder or
!wget http://dosairnowdata.org/dos/historical/Hanoi/2018/Hanoi_PM2.5_2018_YTD.csv -P ./data/
```

```
--2020-07-20 10:33:08--  http://dosairnowdata.org/dos/historical/Hano
i/2018/Hanoi_PM2.5_2018_YTD.csv
Resolving dosairnowdata.org (dosairnowdata.org)... 74.208.236.6, 260
7:f1c0:100f:f000::279
Connecting to dosairnowdata.org (dosairnowdata.org)|74.208.236.6|:8
0... connected.
HTTP request sent, awaiting response... 200 OK
Length: 862331 (842K) [text/csv]
Saving to: ‘./data/Hanoi_PM2.5_2018_YTD.csv’

Hanoi_PM2.5_2018_YT 100%[===================>] 842.12K   498KB/s    i
n 1.7s

2020-07-20 10:33:10 (498 KB/s) - ‘./data/Hanoi_PM2.5_2018_YTD.csv’ sa
ved [862331/862331]
```

In [14]:
```
# check to see if the file is in data
os.listdir('./data')
```

Out[14]:
```
['Hanoi_PM2.5_2018_YTD.csv']
```

Why I downloaded the file?

- The file is available in your local drive, you can examine by text editor or Excel-liked program
- Reduce load on the server, especially when one first tries out the code unintentionally request one file multiple times
- Alternatively, a `csv` file can be read directly into a `DataFrame` (similar to a `Sheet`) by `pandas`

Explore file by Pandas

- pandas (https://pandas.pydata.org/) Python Data Analysis Library is a must-have tool to work with tabular data
- Install library (on linux or Mac), assumed you have `pip` installed

```
pip install pandas --user# process tatular data
pip install matplotlib --user #powerful to make graph
pip install seaborn --user # make the graph look good
```

In [79]:
```python
# import pandas
import pandas as pd
# load the data in the memory
df = pd.read_csv('./data/Hanoi_PM2.5_2018_YTD.csv')
```

In [22]:
```python
# let see the first 5 row of the file
df.head()
```

Out[22]:

	Site	Parameter	Date (LT)	Year	Month	Day	Hour	NowCast Conc.	AQI	AQI Category	Raw Conc.	Conc. Unit
0	Hanoi	PM2.5 - Principal	2018-01-01 01:00 AM	2018	1	1	1	68.9	158	Unhealthy	69.2	UG/M3
1	Hanoi	PM2.5 - Principal	2018-01-01 02:00 AM	2018	1	1	2	72.2	160	Unhealthy	75.5	UG/M3
2	Hanoi	PM2.5 - Principal	2018-01-01 03:00 AM	2018	1	1	3	81.2	164	Unhealthy	90.2	UG/M3
3	Hanoi	PM2.5 - Principal	2018-01-01 04:00 AM	2018	1	1	4	89.4	169	Unhealthy	97.6	UG/M3
4	Hanoi	PM2.5 - Principal	2018-01-01 05:00 AM	2018	1	1	5	89.2	168	Unhealthy	89.1	UG/M3

In [24]:
```python
# `.info` can be handy for high-level summary
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8339 entries, 0 to 8338
Data columns (total 14 columns):
Site              8339 non-null object
Parameter         8339 non-null object
Date (LT)         8339 non-null object
Year              8339 non-null int64
Month             8339 non-null int64
Day               8339 non-null int64
Hour              8339 non-null int64
NowCast Conc.     8339 non-null float64
AQI               8339 non-null int64
AQI Category      8100 non-null object
Raw Conc.         8339 non-null float64
Conc. Unit        8339 non-null object
Duration          8339 non-null object
QC Name           8339 non-null object
dtypes: float64(2), int64(5), object(7)
memory usage: 912.2+ KB
```

- there is many conlumns included for its completedness. such as `Site` , `Parameter` , `Conc. (entration) Unit` ...
- Most columns contain 8339 rows, AQI Category has 8100 rows. The lesser row is resulted from the method to calculate AQI (Air Quality Index), a final number for public.
- Three important columns are `Date (LT)` , `Raw Conc.` , `QC Name` . Other columns are derived from these three columns.

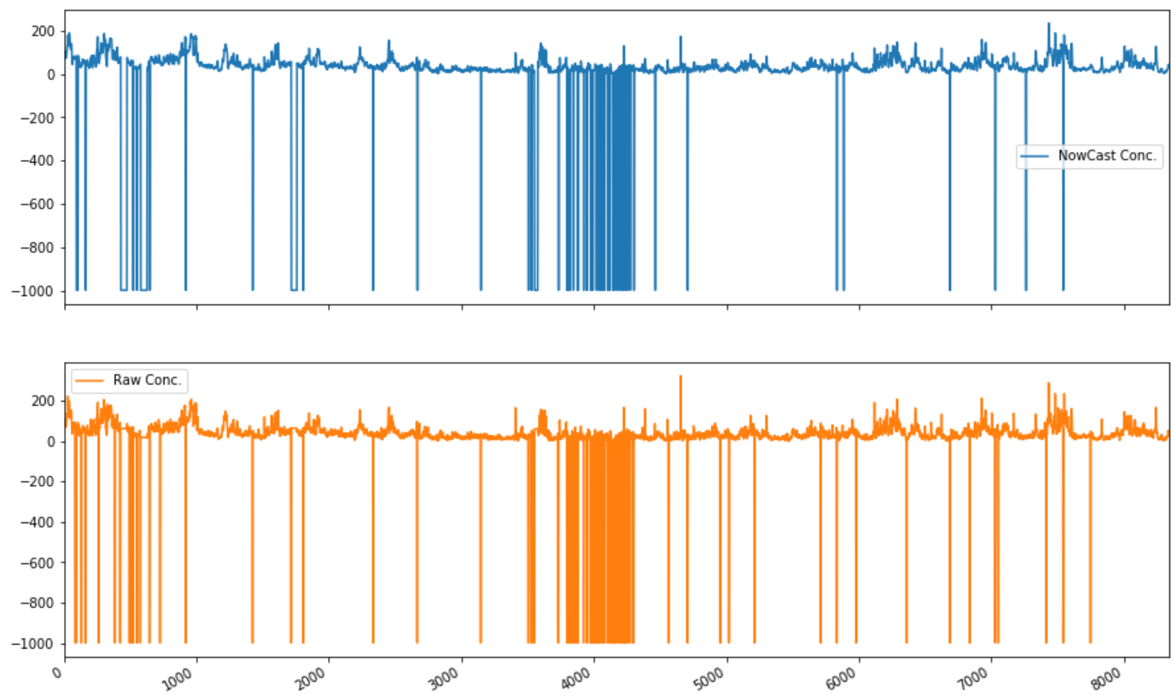
In [25]:
```python
# to have a look at the distribution (statistic)
df.describe()
```

Out[25]:

	Year	Month	Day	Hour	NowCast Conc.	AQI	Raw C
count	8339.000000	8339.000000	8339.000000	8339.000000	8339.000000	8339.000000	8339.00
mean	2018.000120	6.584123	15.971939	11.561338	10.679398	70.699125	22.64
std	0.010951	3.485221	8.801531	6.907012	175.955430	188.781578	139.44
min	2018.000000	1.000000	1.000000	0.000000	-999.000000	-999.000000	-999.00
25%	2018.000000	4.000000	8.000000	6.000000	19.000000	66.000000	18.85
50%	2018.000000	7.000000	16.000000	12.000000	31.300000	91.000000	31.70
75%	2018.000000	10.000000	24.000000	18.000000	49.900000	136.000000	51.80
max	2019.000000	12.000000	31.000000	23.000000	235.800000	286.000000	323.00

- only numeric columns are listed here
- notice `-999` in Conc columns
- for summary statistics, this table is already overwhelming
- the mean (raw) concentration is `22 microgram/cubic meter`, did you spot what is wrong with this number?
- 50% label is called `median`, a value of concentration (for example) that devided the sample pool into two, so that 50 percent of the sample is smaller than the median (18.85), and 50% is larger the medium.
- the median is lower than the mean (average), why is that?

In [26]:
```
# let visualize to concentration, use the .plot function from pandas,
# I used subplots=True to separate two graphs, and figsize=(15,6) ind
icate the size of the graph
df[['NowCast Conc.', 'Raw Conc.']].plot(subplots=True, figsize=(15,10
));
```



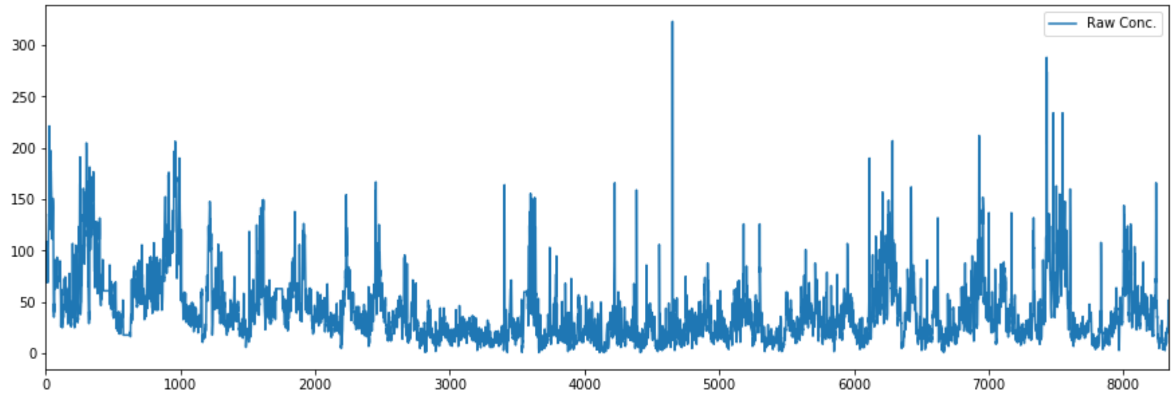
- uhm, this is not really make the data is easier to see the trend,
- the `-999` s make the graph skewed and cannot see the trend.
- Make a quick fix

In [32]:
```
df.columns
```

Out[32]:
```
Index(['Site', 'Parameter', 'Date (LT)', 'Year', 'Month', 'Day', 'Hou
r',
       'NowCast Conc.', 'AQI', 'AQI Category', 'Raw Conc.', 'Conc. Un
it',
       'Duration', 'QC Name'],
      dtype='object')
```

In [71]:
```
df[df['Raw Conc.'] > 0]['Raw Conc.'].plot(figsize=(15,5), legend=True
)
```

Out[71]: <matplotlib.axes._subplots.AxesSubplot at 0x7f4484456940>



- this techniue is called `filtering`
- first `df['Raw Conc.'] > 0` yield a table with `False` or `True` value for each cell
- only cells wiht `True` value selected by `df[df['Raw Conc.']]`
- Next the column `Raw Conc.` is selected by `df[df['Raw Conc.']['Raw Conc.']`
- finally, `plot` function is called to display to clean data

Make data is more insightful

Make the timeseries data

- convert a string represented date and time to a `datetime` object
- set the datetime as the index
- remove redundant columns

In [80]:
```python
# convert string to datetime and set this column as the index
df['Date (LT)'] = pd.to_datetime(df['Date (LT)'])
# set a column as the index
df.set_index('Date (LT)', inplace=True)
df.head()
```

Out[80]:

Date (LT)	Site	Parameter	Year	Month	Day	Hour	NowCast Conc.	AQI	AQI Category	Raw Conc.	Conc. Unit
2018-01-01 01:00:00	Hanoi	PM2.5 - Principal	2018	1	1	1	68.9	158	Unhealthy	69.2	UG/M3
2018-01-01 02:00:00	Hanoi	PM2.5 - Principal	2018	1	1	2	72.2	160	Unhealthy	75.5	UG/M3
2018-01-01 03:00:00	Hanoi	PM2.5 - Principal	2018	1	1	3	81.2	164	Unhealthy	90.2	UG/M3
2018-01-01 04:00:00	Hanoi	PM2.5 - Principal	2018	1	1	4	89.4	169	Unhealthy	97.6	UG/M3
2018-01-01 05:00:00	Hanoi	PM2.5 - Principal	2018	1	1	5	89.2	168	Unhealthy	89.1	UG/M3

In [91]:
```python
# check data type, the index has `DatetimeIndex`
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 8339 entries, 2018-01-01 01:00:00 to 2019-01-01 00:00:
00
Data columns (total 6 columns):
Month              8339 non-null int64
NowCast Conc.      8339 non-null float64
AQI                8339 non-null int64
AQI Category       8100 non-null object
Raw Conc.          8339 non-null float64
QC Name            8339 non-null object
dtypes: float64(2), int64(2), object(2)
memory usage: 456.0+ KB
```

In [115]:
```python
df = dfs.copy(deep=True)
```

In [116]:
```
# remove first 5 columns and two columns near the last one
# inplace=True specifies the change (remove columns) in df object
df.drop(columns=['Site', 'Parameter', 'Year', 'Month', 'Day', 'Hour',
'Conc. Unit', 'Duration'], inplace=True)
df.head()
```

Out[116]:

	NowCast Conc.	AQI	AQI Category	Raw Conc.	QC Name
Date (LT)					
2018-01-01 01:00:00	68.9	158	Unhealthy	69.2	Valid
2018-01-01 02:00:00	72.2	160	Unhealthy	75.5	Valid
2018-01-01 03:00:00	81.2	164	Unhealthy	90.2	Valid
2018-01-01 04:00:00	89.4	169	Unhealthy	97.6	Valid
2018-01-01 05:00:00	89.2	168	Unhealthy	89.1	Valid

In [117]:
```
# filter the data and assign the cleaned DataFrame to df2
df2 = df[df['Raw Conc.']>=0]
df2.describe()
```

Out[117]:

	NowCast Conc.	AQI	Raw Conc.
count	8190.000000	8190.000000	8190.000000
mean	10.626288	70.710134	40.752259
std	176.425577	189.279538	31.456565
min	-999.000000	-999.000000	0.000000
25%	19.000000	66.000000	19.000000
50%	31.400000	92.000000	32.000000
75%	50.075000	137.000000	52.000000
max	235.800000	286.000000	323.000000

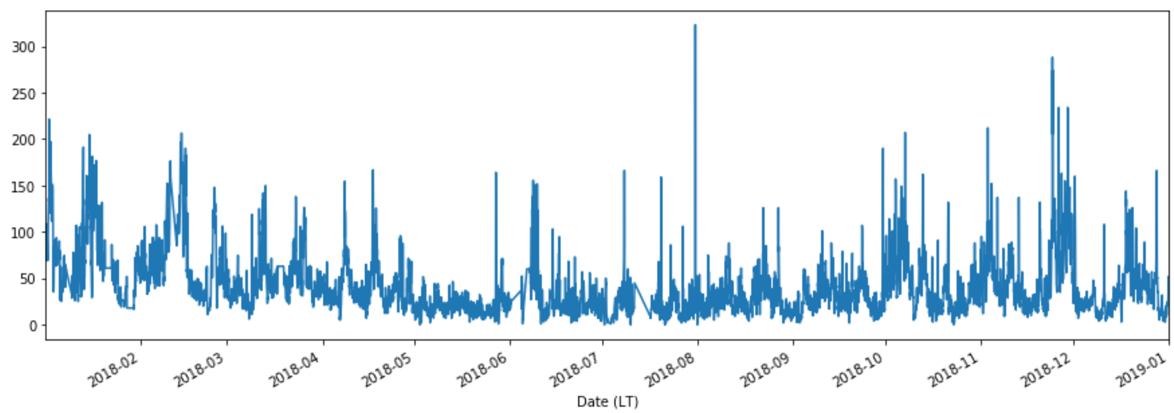
- `-999` values are removed from `Raw Conc.` columns, but some are still in the `AQI` and `NowCast Conc.`
- less rows in `df2` (8190) vs. 8339 in `df`
- the mean value for concentration is 40.7 (ug/m3), and the median is 32 (ug/m3) in cleaned version (in `df`, the mean value `22.6` (ug/m3)
- small mistakes could lead to an inaccurate results, and a wrong interpretation (ie. mean, median)

In [118]:
```
df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 8190 entries, 2018-01-01 01:00:00 to 2019-01-01 00:00:
00
Data columns (total 5 columns):
NowCast Conc.    8190 non-null float64
AQI              8190 non-null int64
AQI Category     7954 non-null object
Raw Conc.        8190 non-null float64
QC Name          8190 non-null object
dtypes: float64(2), int64(1), object(2)
memory usage: 383.9+ KB
```

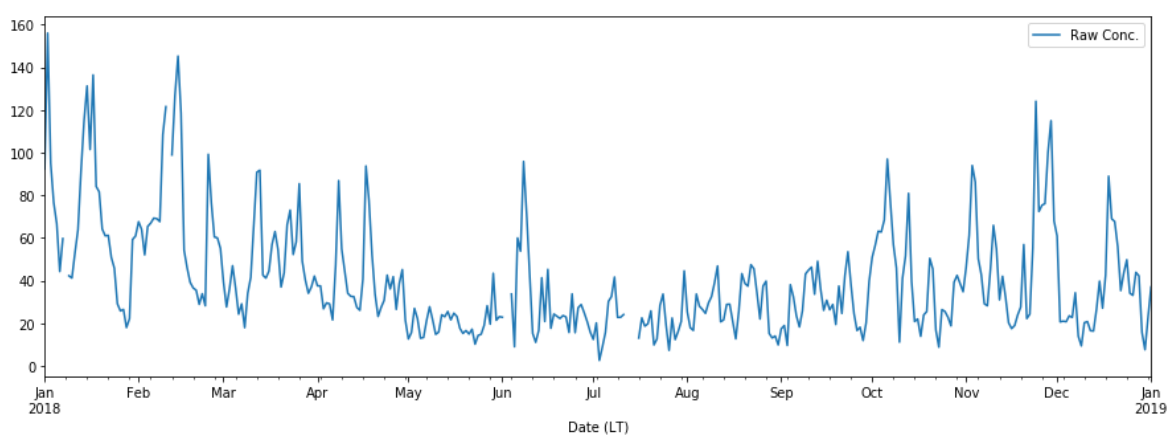
In [95]:
```
# let see concentration in 2018 with timeseris
df2['Raw Conc.'].plot(figsize=(15,5))
```

Out[95]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f44cc22bf60>
```



In [110]:
```
# a daily average could make the graph less messy
df2[['Raw Conc.']].resample('1D').mean().plot(figsize=(15,5), kind='l
ine')
```

Out[110]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f447ce6d2b0>
```



Operations

one line of codev above essentially performed three things:

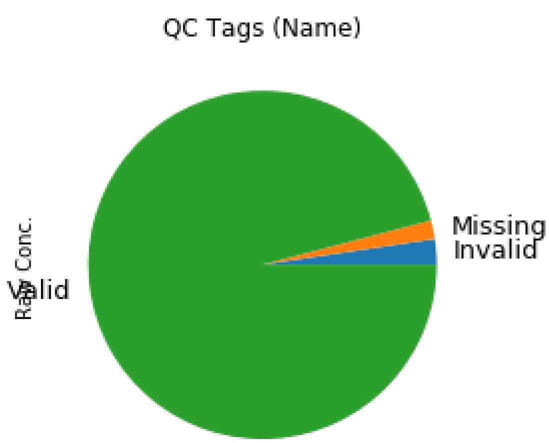
1. Reduced dimesion from 5 columns to one column (in addition to the index column) by `df2[['Raw Conc.']]` (the double square brackets are key here)
2. Grouped `Raw Conc.` by an interval of one day in `resample('1D')`, change `1D` to `10D`, or `30D` adjusts the interval
3. Calculated `mean()` of aggregated data, other function such as `std()` works as well
4. finally, plotting

Interpretation

- PM2.5 or particulate mattters that has a diameter of 2.5 micrometer or less is one of outdoor pollutant regulated
- depend on the country, the standard (or recommendation) of daily concentration is different
- The recommendation of WHO is 25 μg/m^3 (https://apps.who.int/iris/bitstream/handle/10665/69477/WHO_SDE_PHE_OEH_06.02_eng.pdf) daily average, 35 μg/m^3 (https://www.epa.gov/pm-pollution/2006-national-ambient-air-quality-standards-naaqs-particulate-matter-pm25) by US EPA, and 50 μg/m^3 (https://www.env.go.jp/air/tech/ine/asia/vietnam/files/law/QCVN%2005-2013.pdf) by Vietname Environmental Administration

In [131]:
```python
# before moving on the make the graph more useful, let look as the Quality Control (QC) of the raw data
# for environmnetal data, a valid QC (about 98%) is solid
df.groupby('QC Name')['Raw Conc.'].count().plot.pie(title='QC Tags (Name)',table=True, fontsize=13)
```

Out[131]: <matplotlib.axes._subplots.AxesSubplot at 0x7f44843fd160>



	Invalid	Missing	Valid
Raw Conc.	196	145	7998

this one line of code performs three primary operations as one:

1. Group all values in `QC Name` columns (`df.groupby('QC Name')` into category
2. filter by one column `Raw Conc.` to reduce the DataFrame (matrix mxn) to series (two columns x rows)
3. `count()` the value of each tag (`Valid` , `Missing` , `Invalid` , `Suspect` (not in here but you may found with other files)
4. Call `plot` to display to count of each instances

In [128]:
```
# let save clean file back to local drive
df2.to_csv('./data/cleaned_Hanoi_PM2.5_2018_YTD.csv')
```

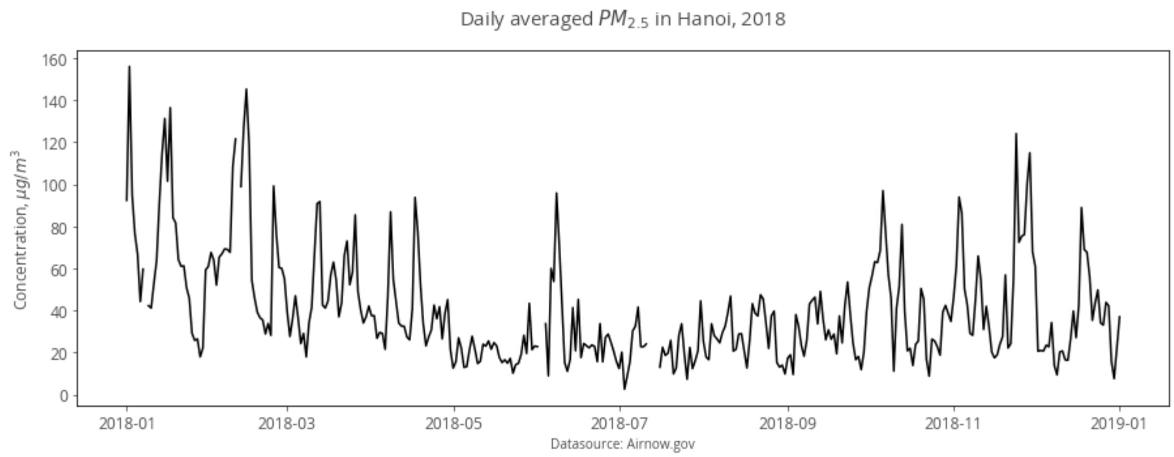
Advance visualization with matplotlib and seaborn

- `pandas` is a powerful library to process data, with some handy plot tools. `pandas` is a good choice for data exploration
- `matplotlib` is a proper tool for visualization. `pandas` "borrows" some plotting functions from matplotlib

line plots

In [133]:
```
# import matplotlib
import matplotlib.pyplot as plt
%matplotlib inline
plt.rcParams['figure.figsize'] =  (15,5)
plt.rcParams['font.sans-serif'] = 'Open Sans'
plt.rcParams['font.family'] = 'sans-serif'
plt.rcParams['text.color'] = '#4c4c4c'
plt.rcParams['axes.labelcolor']= '#4c4c4c'
plt.rcParams['xtick.color'] = '#4c4c4c'
plt.rcParams['ytick.color'] = '#4c4c4c'
plt.rcParams['font.size']=12
```

In [140]:
```
# recreate a plot from above
# with title and label
plt.title('Daily averaged $PM_{2.5}$ in Hanoi, 2018', fontsize=15, y=
1.05)
plt.ylabel('Concentration, $\mu g/m^3$')
plt.xlabel('Datasource: Airnow.gov', fontsize=10)
dft = df2[['Raw Conc.']].resample('1D').mean()
# change the line color, thickness
plt.plot(dft, color='black', linewidth=1.5)
# savefile to local
plt.savefig('img/2020Jul_hanoi.png')
```



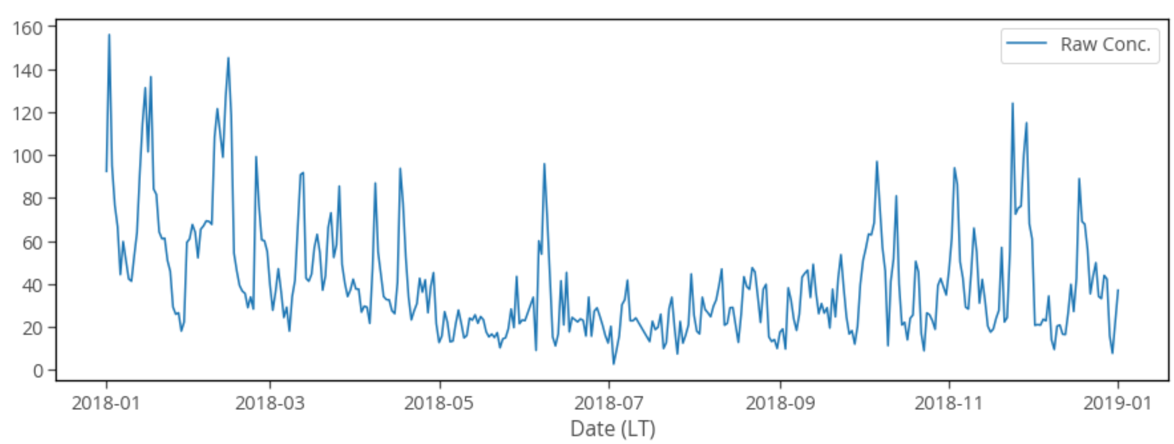
In [141]:
```
# check to see if the image is actullay there
! ls ./img
```

```
2020Jul_hanoi.png  airmonitors_location.png
```

In [142]:
```
# recreate this graph by seaborn
import seaborn as sns
sns.set_context("notebook", font_scale=1.3)
```

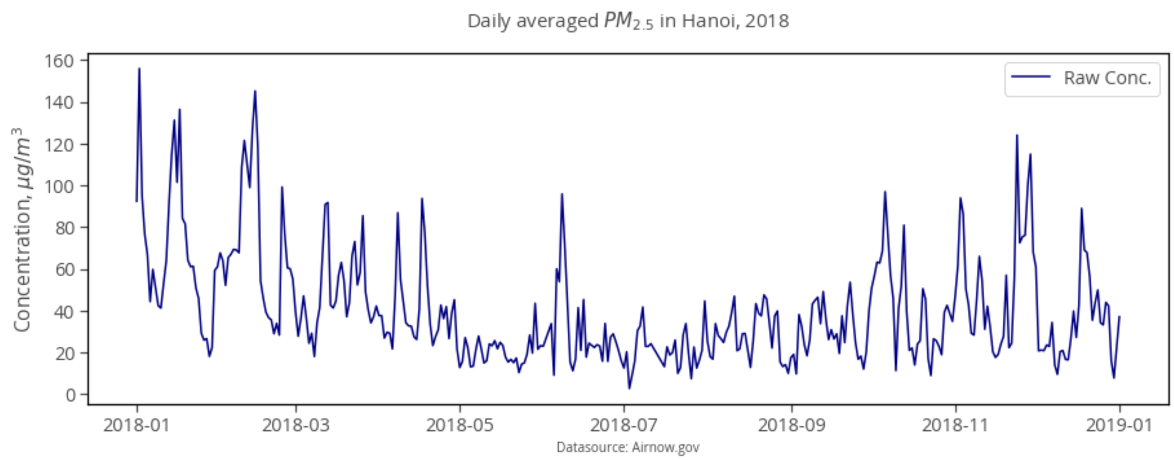
In [146]:
```
# minimal setup, and the axes and font look really nice already
sns.lineplot(data=dft)
```

Out[146]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f447ca074a8>
```



In [161]:
```
# of course, you can combine both flexiblity of matplotlib and the ni
ce setup of seaborn
ax = sns.lineplot(data=dft, palette = ['navy'])
ax.set_xlabel('Datasource: Airnow.gov', fontsize=10)
plt.title('Daily averaged $PM_{2.5}$ in Hanoi, 2018', fontsize=15, y=
1.05)
plt.ylabel('Concentration, $\mu g/m^3$')
# ax.
```

Out[161]:
```
Text(0, 0.5, 'Concentration, $\\mu g/m^3$')
```



- lineplot is the most simple one (beside scatter), for this setup, seaborn has not demonstrated its advantages,

In [ ]:
```

```

In [162]:
```
fmri = sns.load_dataset("fmri")
fmri.head()
```

Out[162]:

	subject	timepoint	event	region	signal
0	s13	18	stim	parietal	-0.017552
1	s5	14	stim	parietal	-0.080883
2	s12	18	stim	parietal	-0.081033
3	s11	18	stim	parietal	-0.046134
4	s10	18	stim	parietal	-0.037970

In [175]:
```
colors = ['purple',  'red', 'orange', 'yellow', 'green']
```

In [193]:
```
orders = ['Very Unhealthy', 'Unhealthy', 'Unhealthy for Sensitive Gro
ups', 'Moderate', 'Good']
```

```
In [196]: colormap = dict(zip(orders, colors))
          colormap
```

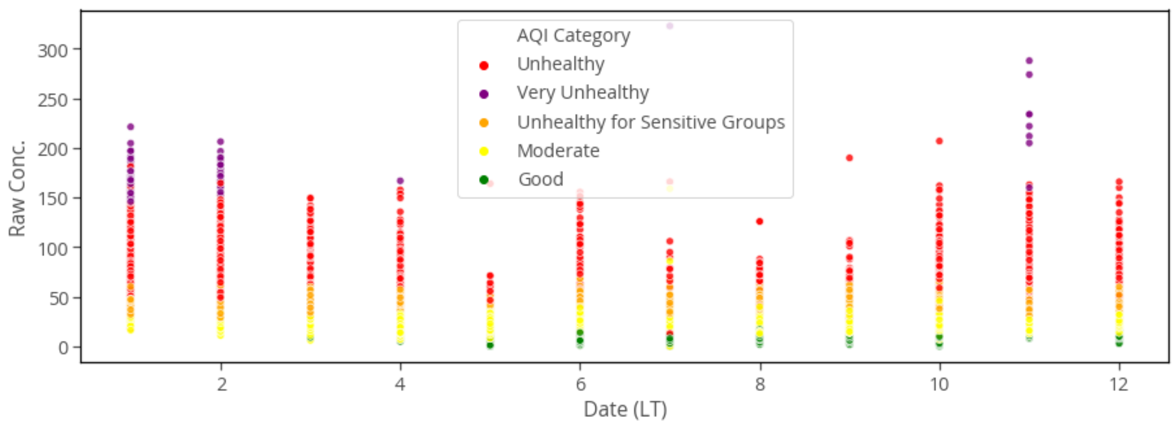
```
Out[196]: {'Very Unhealthy': 'purple',
           'Unhealthy': 'red',
           'Unhealthy for Sensitive Groups': 'orange',
           'Moderate': 'yellow',
           'Good': 'green'}
```

```
In [192]: df2['AQI Category'].value_counts()
```

```
Out[192]: Moderate                          3730
          Unhealthy for Sensitive Groups    1847
          Unhealthy                         1611
          Good                               684
          Very Unhealthy                      82
          Name: AQI Category, dtype: int64
```

```
In [208]: sns.scatterplot(data=df2, x=df2.index.month, y=df2['Raw Conc.'],
                          hue='AQI Category', palette=colormap, alpha=0.8)
```

```
Out[208]: <matplotlib.axes._subplots.AxesSubplot at 0x7f444f4d0860>
```

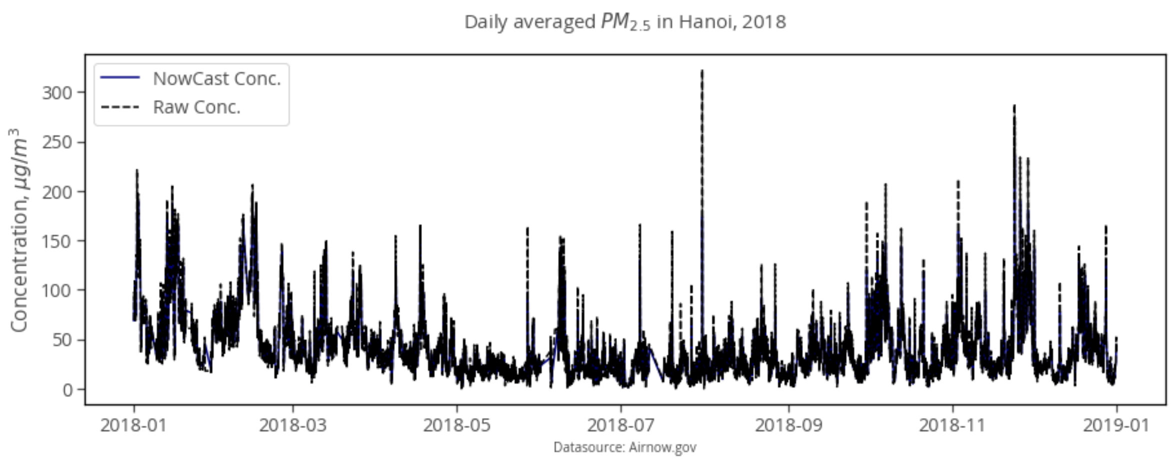


- notice that there are some overlap between `AQI Category` and `Raw Conc.`
- this is because AQI calculated from `NowCast Conc.` , and `NowCast Conc.` is a predicting value of daily concentration by calculating the last twelve hourly values of `Raw Conc.`

In [214]:
```
df2.head()
```
Out[214]:

Date (LT)	NowCast Conc.	AQI	AQI Category	Raw Conc.	QC Name
2018-01-01 01:00:00	68.9	158	Unhealthy	69.2	Valid
2018-01-01 02:00:00	72.2	160	Unhealthy	75.5	Valid
2018-01-01 03:00:00	81.2	164	Unhealthy	90.2	Valid
2018-01-01 04:00:00	89.4	169	Unhealthy	97.6	Valid
2018-01-01 05:00:00	89.2	168	Unhealthy	89.1	Valid

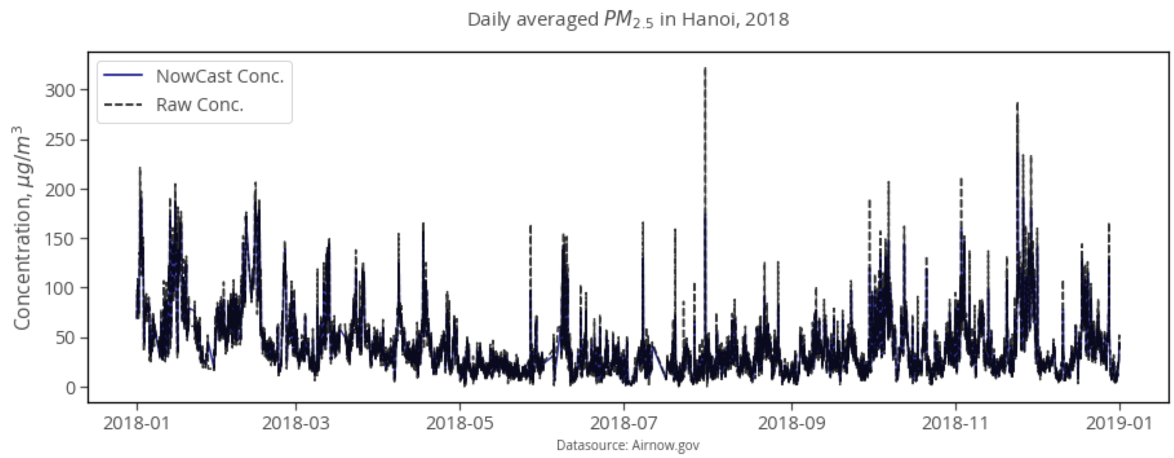
In [259]:
```
# let see how the Raw and NowCast Concentration look on graph
# of course, you can combine both flexiblity of matplotlib and the ni
ce setup of seaborn
ax = sns.lineplot(data=df2[['NowCast Conc.', 'Raw Conc.']], palette =
['navy', 'black'])
ax.set_xlabel('Datasource: Airnow.gov', fontsize=10)
plt.title('Daily averaged $PM_{2.5}$ in Hanoi, 2018', fontsize=15, y=
1.05)
plt.ylabel('Concentration, $\mu g/m^3$');
```



In [235]:
```
# this is not great, messy instead, let replace a NULL value with -99
9s error code in NowCast Conc.
df2.loc[df2['NowCast Conc.'] < 0, 'NowCast Conc.'] = None
```

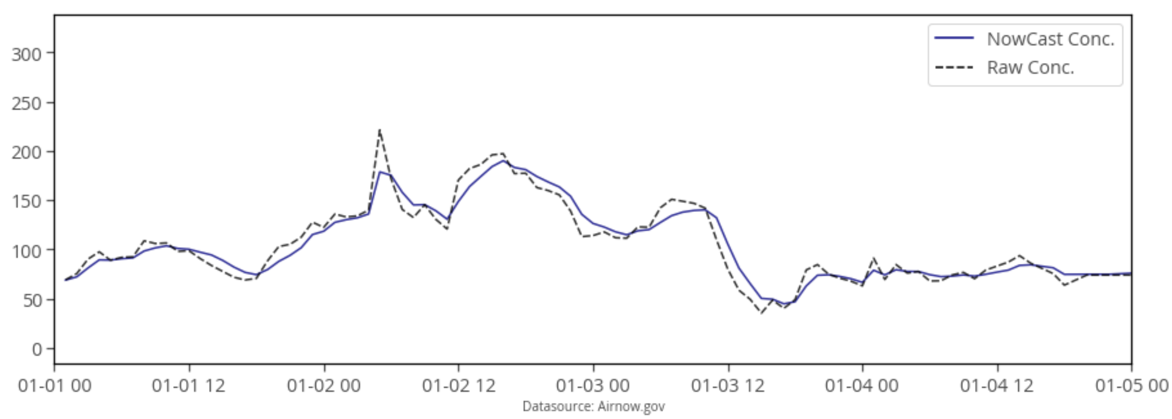
In [250]:
```python
# this is not great either, but the minus values are filtered out
ax = sns.lineplot(data=df2[['NowCast Conc.', 'Raw Conc.']], palette =
['navy', 'black'], alpha=0.8)
ax.set_xlabel('Datasource: Airnow.gov', fontsize=10)
plt.title('Daily averaged $PM_{2.5}$ in Hanoi, 2018', fontsize=15, y=
1.05)
plt.ylabel('Concentration, $\mu g/m^3$')
```

Out[250]:
```
Text(0, 0.5, 'Concentration, $\\mu g/m^3$')
```

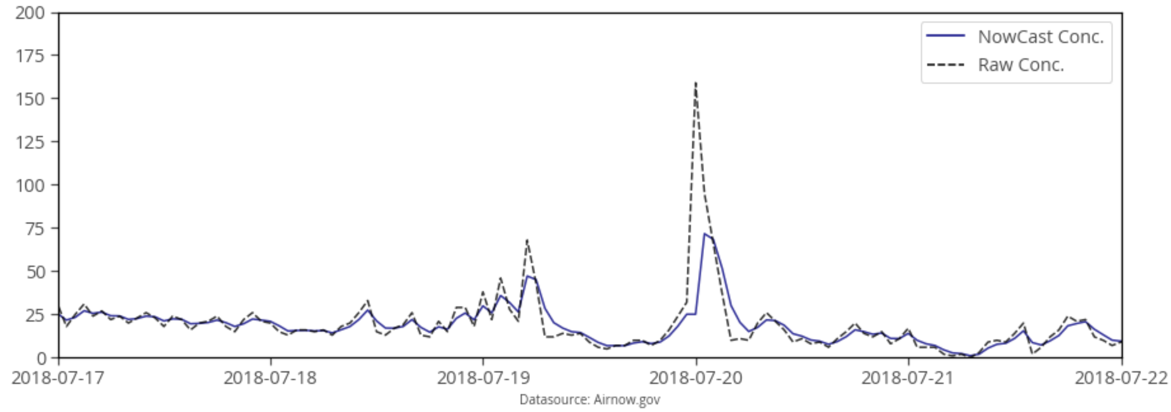


In [239]:
```python
# so let zoom in a few instances, first let set up the limits
from datetime import datetime as dt
```

In [258]:
```python
left = dt(2018,1,1)
right = dt(2018,1,5)
ax = sns.lineplot(data=df2[['NowCast Conc.', 'Raw Conc.']], palette =
['navy', 'black'], alpha=0.8)
ax.set_xlabel('Datasource: Airnow.gov', fontsize=10)
ax.set_xlim(left, right);
```



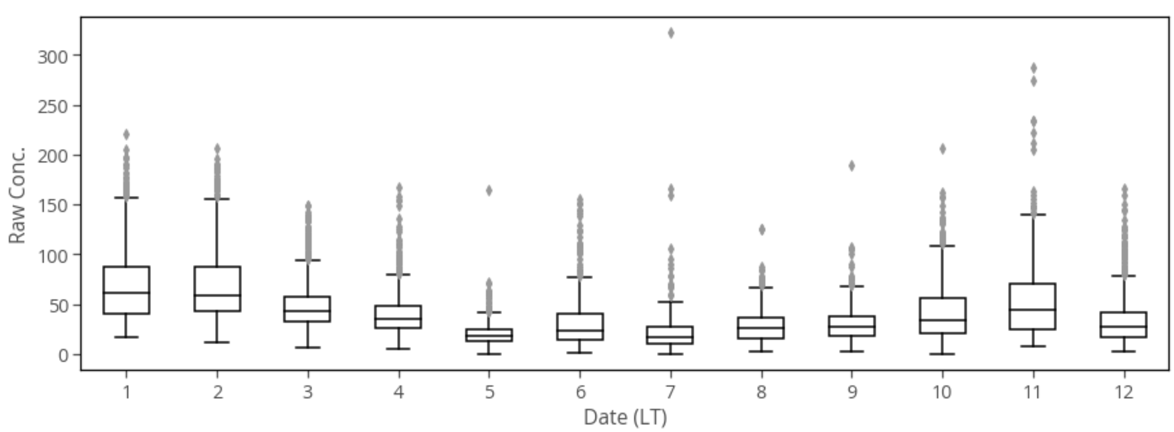
In [257]:
```
left = dt(2018,7,17)
right = dt(2018,7,22)
ax = sns.lineplot(data=df2[['NowCast Conc.', 'Raw Conc.']], palette =
['navy', 'black'], alpha=0.8)
ax.set_xlabel('Datasource: Airnow.gov', fontsize=10)
ax.set_xlim(left, right)
ax.set_ylim(0,200);
```



- NowCast Conc. is similar to the moving average that it smooths out the peak and present a more likely value for a longer period (day)

In [263]:
```
# if we want to have statistics look, the boxplot is a good place sta
rt
ax = sns.boxplot(data=df2, x=df2.index.month, y=df2['Raw Conc.'], wid
th=0.5, palette=['white'])
for i,box in enumerate(ax.artists):
    box.set_edgecolor('black')
    box.set_facecolor('white')

    # iterate over whiskers and median lines
    for j in range(6*i,6*(i+1)):
        ax.lines[j].set_color('black')
```

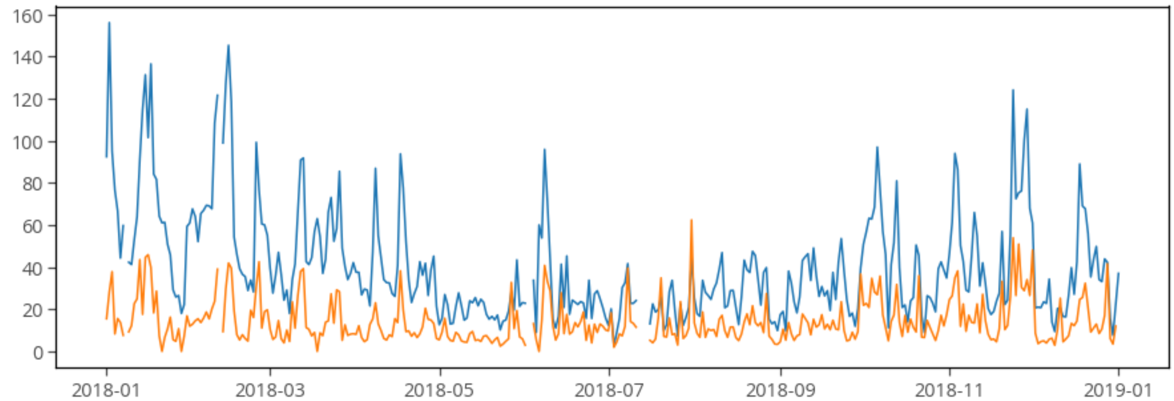


In [ ]:
```
# let comeback to the dft, or a daily average
```

In [265]:
```
dft2 = df2[['Raw Conc.']].resample('1D')
```

In [267]:
```
# this is not really informed, the standard deviation (std) should be presented by a band
plt.plot(dft)
plt.plot(dft2.std())
```

Out[267]:
```
[<matplotlib.lines.Line2D at 0x7f4450852080>]
```



In [270]:
```
std = dft2.std()
std.head()
```

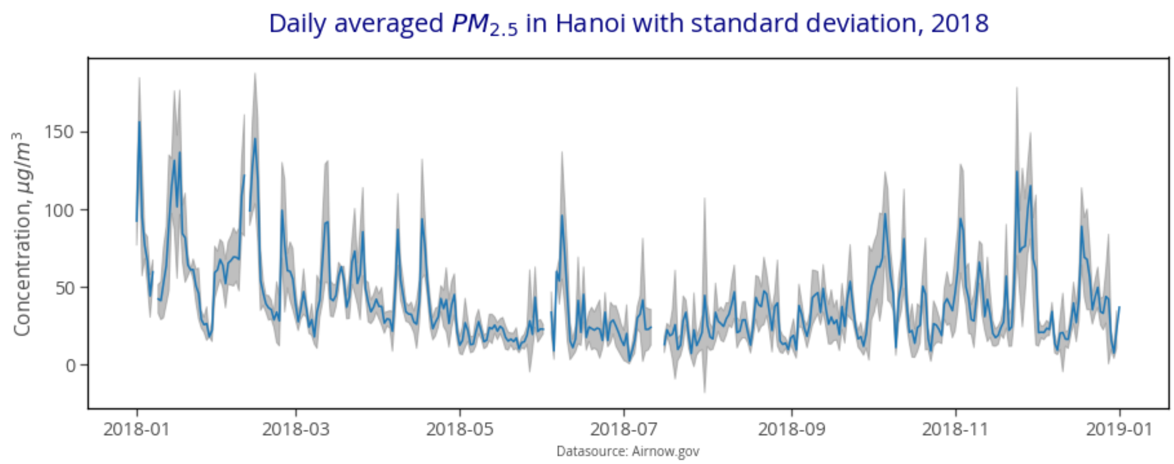
Out[270]:

	Raw Conc.
Date (LT)	
2018-01-01	15.487127
2018-01-02	28.224719
2018-01-03	37.818703
2018-01-04	8.274930
2018-01-05	15.594304

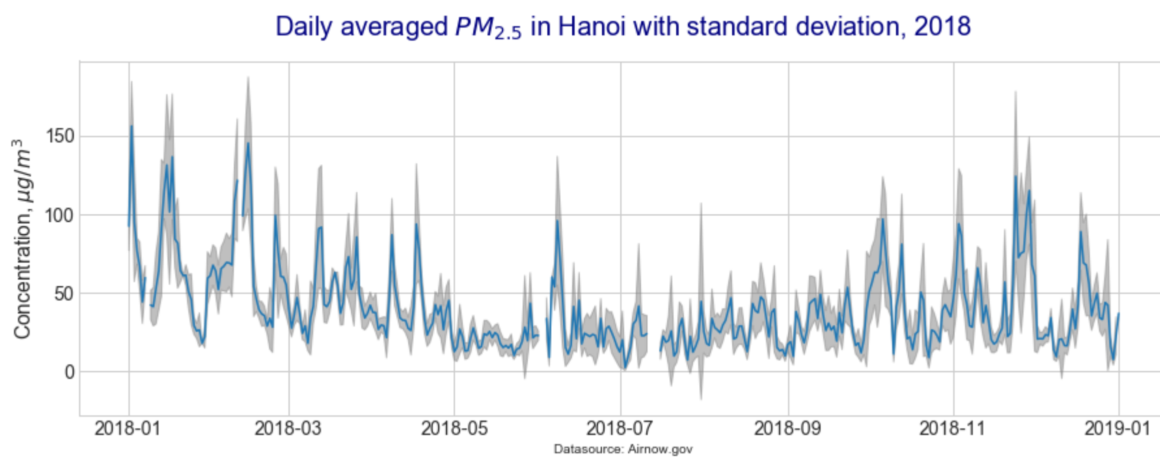
In [283]:
```
std_ = std.values.reshape(1, -1)[0].shape
mean_ = dft.values.reshape(1, -1)[0].shape
```

In [296]:

```
# dft[dft.columns[0]] select the first column in the dataframe, alter
natively, dft['Raw Conc.']
plt.fill_between(std.index,
                 dft[dft.columns[0]] - std[std.columns[0]],
                 dft[dft.columns[0]] + std[std.columns[0]],
                 color='gray',
                 alpha=0.5)
plt.plot(dft.index, dft.values)
plt.xlabel('Datasource: Airnow.gov', fontsize=10)
plt.title('Daily averaged $PM_{2.5}$ in Hanoi with standard deviatio
n, 2018',
          color='navy',
          fontsize=20, y=1.05)
plt.ylabel('Concentration, $\mu g/m^3$');
```



In [297]:
```
# use a setup style
plt.style.use('seaborn-whitegrid')
plt.fill_between(std.index,
                 dft[dft.columns[0]] - std[std.columns[0]],
                 dft[dft.columns[0]] + std[std.columns[0]],
                 color='gray',
                 alpha=0.5)
plt.plot(dft.index, dft.values)
plt.xlabel('Datasource: Airnow.gov', fontsize=10)
plt.title('Daily averaged $PM_{2.5}$ in Hanoi with standard deviatio
n, 2018',
          color='navy',
          fontsize=20, y=1.05)
plt.ylabel('Concentration, $\mu g/m^3$');
```



analyze by AQI levels

- group by AQI label

In [302]:
```
print(df2.shape)
dfv = df2[df2['QC Name'] == 'Valid']
print(dfv.shape)
```

```
(8190, 5)
(7997, 5)
```

In [377]:
```
for_pie = dfv['AQI Category'].value_counts()
type(for_pie)
for_pie
```

Out[377]:
```
Moderate                          3727
Unhealthy for Sensitive Groups    1843
Unhealthy                         1603
Good                               684
Very Unhealthy                      82
Name: AQI Category, dtype: int64
```

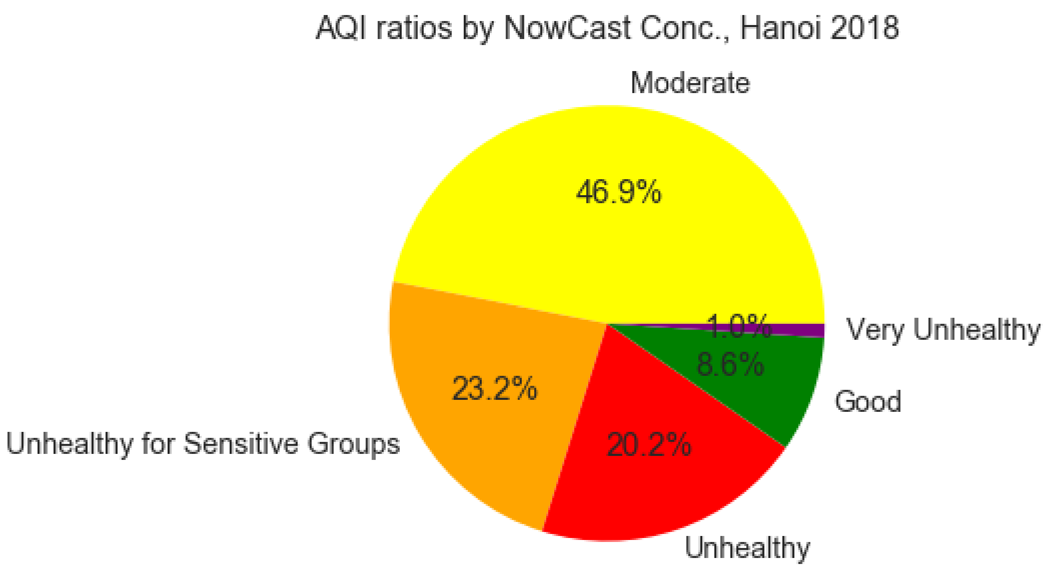
In [333]:
```
list(for_pie.index)
```

Out[333]:
```
['Moderate',
 'Unhealthy for Sensitive Groups',
 'Unhealthy',
 'Good',
 'Very Unhealthy']
```

In [337]:
```
colormap
```

Out[337]:
```
{'Very Unhealthy': 'purple',
 'Unhealthy': 'red',
 'Unhealthy for Sensitive Groups': 'orange',
 'Moderate': 'yellow',
 'Good': 'green'}
```

In [378]:
```
plt.pie(for_pie,
        labels=list(for_pie.index),
        colors=['yellow', 'orange', 'red', 'green', 'purple'], autopc
t='%1.1f%%');
plt.title('AQI ratios by NowCast Conc., Hanoi 2018');
```



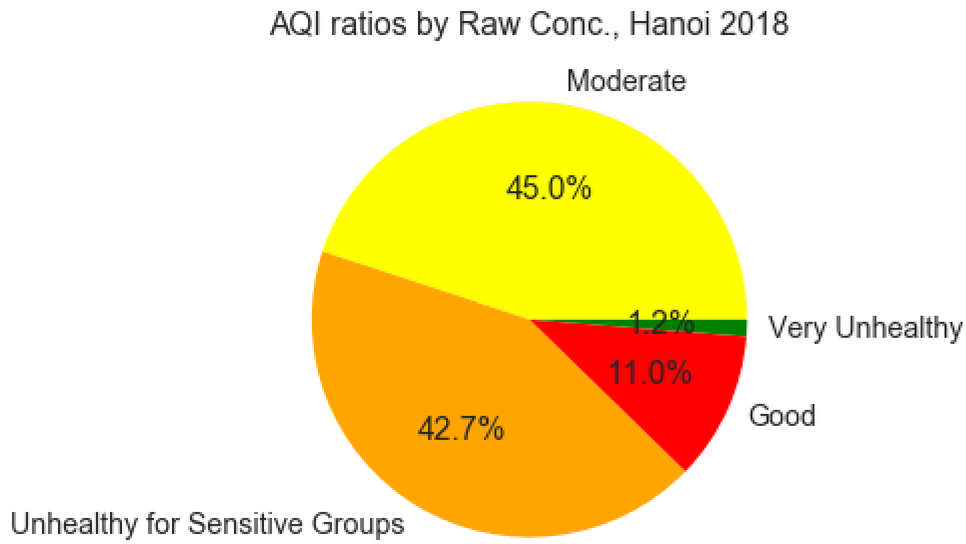
In [334]:
```
# comprehensive for pie: https://blog.algorexhealth.com/2018/03/almos
t-10-pie-charts-in-10-python-libraries/
```

In [349]:
```
dft.head()
```
Out[349]:

	Raw Conc.
Date (LT)	
2018-01-01	92.373913
2018-01-02	156.020833
2018-01-03	94.995833
2018-01-04	76.527273
2018-01-05	66.666667

In [441]:
```
aqi = {
    'Good':{'pm2.5': [0, 12], 'color': 'green'},
    'Moderate': {'pm2.5': [12.1, 35.4], 'color': 'yellow'},
    'Unhealthy for Sensitive Groups':{'pm2.5': [35.5, 55,4], 'color':
'orange'},
    'Unhealthy': {'pm2.5': [55.5, 150.4], 'color': 'red'},
    'Very Unhealthy': {'pm2.5': [150.5, 250.5], 'color': 'purple'},
    'Hazardous': {'pm2.5': [250.5, 500.4], 'color': 'maroon'}}
```
In [426]:
```
bins = [x['pm2.5'][0] for x in list(aqi.values())]
```
In [435]:
```
bins.append(aqi['Hazardous']['pm2.5'][-1])
```
In [436]:
```
bins
```
Out[436]:
```
[0, 12.1, 35.5, 55.5, 150.5, 250.5, 500.4]
```
In [386]:
```
dfvc = dfv[['Raw Conc.']]
```
In [387]:
```
for_pie2 = pd.cut(dfvc['Raw Conc.'], bins=bins, labels= list(aqi.keys
()), include_lowest=True).value_counts()
```
In [390]:
```
for_pie3 = pd.cut(dft['Raw Conc.'], bins=bins, labels= list(aqi.keys
()), include_lowest=True).value_counts()
```
In [391]:
```
for_pie3
```
Out[391]:
```
Moderate                          177
Unhealthy for Sensitive Groups     91
Unhealthy                          75
Good                               14
Very Unhealthy                      1
Name: Raw Conc., dtype: int64
```

In [371]:
```
plt.pie(for_pie2,
        labels=list(for_pie2.index),
        colors=['yellow', 'orange', 'red', 'green', 'purple'], autopc
t='%1.1f%%');
plt.title('AQI ratios by Raw Conc., Hanoi 2018');
```



In [446]:
```
# let combine three pies in one plate
all_pies = pd.concat([for_pie, for_pie2, for_pie3], axis=1)
all_pies.columns = ['NowCast,h', 'Raw,h', 'Raw,d']
all_pies = all_pies.reindex(aqi.keys())
all_pies
```

Out[446]:

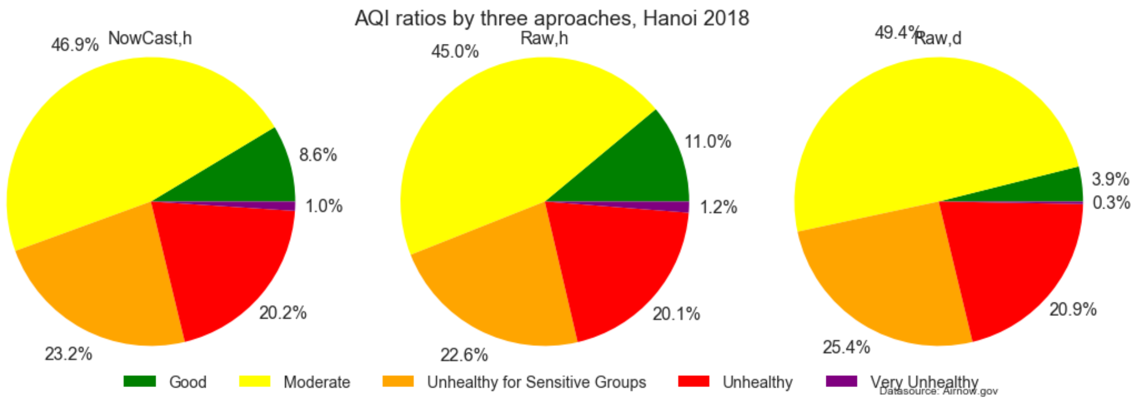
	NowCast,h	Raw,h	Raw,d
Good	684.0	883.0	14.0
Moderate	3727.0	3595.0	177.0
Unhealthy for Sensitive Groups	1843.0	1810.0	91.0
Unhealthy	1603.0	1607.0	75.0
Very Unhealthy	82.0	99.0	1.0
Hazardous	NaN	NaN	NaN

In [455]:
```
# drop the last column, otherwise the percetage will not work
all_pies.drop(labels='Hazardous', inplace=True)
```

In [450]:
```
colors = [x['color'] for x in aqi.values()]
colors
```

Out[450]: ['green', 'yellow', 'orange', 'red', 'purple', 'maroon']

In [485]:
```
fig, axes = plt.subplots(nrows=1, ncols=3)
for i, col in enumerate(all_pies.columns):
    axes[i].pie(all_pies[col],
                colors=colors, autopct='%1.1f%%', pctdistance=1.2)
    axes[i].set_title(col, y=0.92)
fig.legend(list(all_pies.index), ncol=6, loc='lower center')
fig.suptitle('AQI ratios by three aproaches, Hanoi 2018')
axes[2].set_xlabel('Datasource: Airnow.gov', fontsize=10)
fig.tight_layout()
fig.savefig('img/2020Jul-AQI.png', dpi=120)
```



Concluding notes

- Python, `pandas` , `matplotlib` , `seaborn` are more approachable to work with data (than we presu