EASC2410 Lecture 10

Python Basics: Data wrangling with Pandas

Dr. Binzheng Zhang
Department of Earth Sciences

Review of Lecture 9

In Lecture 9, we learned:

- **Basics concepts of file system**
- **Intro to Pandas - load data**
- **The concept of a data frame**

In Lecture 10, you will learn:

- **Data wrangling with Pandas: filtering, cleaning**
- **Dealing with multiple data files: the glob module**
- **Format strings**

Recall in Lecture 9, we used the head() and tail() functions

In [1]:

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

yutu = pd.read_excel("datasets/Yutu.xlsx")

yutu.head()
```

Out[1]:

	ear	Month	Day	Hour	Lat.	Long.	Pressure (hPa)	Wind (kt)	Class
0	2018	10	20	18	8.4	160.7	1008	0	0.0
1	2018	10	21	0	8.5	159.9	1008	0	2.0
2	2018	10	21	6	8.6	158.9	1004	0	2.0
3	2018	10	21	12	8.7	158.0	1006	0	2.0
4	2018	10	21	18	8.9	157.1	1004	0	2.0

Observations 1:
quite a few zeros in the wind data

In [2]:

```python
yutu.tail()
```

Out[2]:

	ear	Month	Day	Hour	Lat.	Long.	Pressure (hPa)	Wind (kt)	Class
49	2018	11	2	0	20.7	116.4	1008	35	3.0
50	2018	11	2	6	20.7	116.1	1008	200	2.0
51	2018	11	2	12	20.5	116.0	1012	0	NaN
52	2018	11	2	18	20.2	115.9	1012	0	NaN
53	2018	11	3	0	19.9	115.7	1014	0	NaN

Observations 2:
NaN (Not a Number) in the data

More tricks on the head() and tail() functions

In [3]:
```
yutu["Pressure (hPa)"].head()
```

Out[3]:
```
0    1008
1    1008
2    1004
3    1006
4    1004
Name: Pressure (hPa), dtype: int64
```

Or display multiple columns by specifying the column names as a list:

In [4]:
```
yutu[["Pressure (hPa)","Wind (kt)","Class"]].head()
```

Out[4]:

	Pressure (hPa)	Wind (kt)	Class
0	1008	0	0.0
1	1008	0	2.0
2	1004	0	2.0
3	1006	0	2.0
4	1004	0	2.0

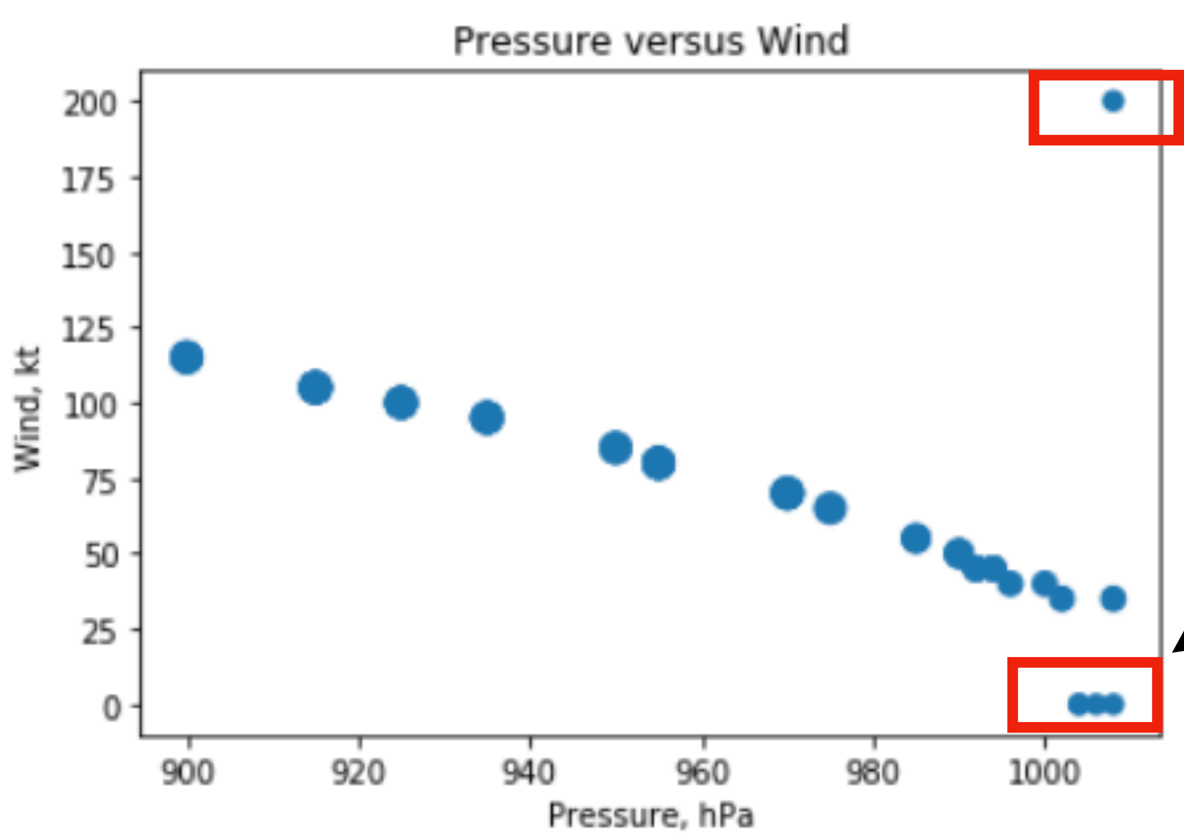
You can show the data rows of selected columns using the head() or tail() function

Outliers in a dataset

Recall the plot in the previous lecture, we made a plot to show the relationship between Pressure and Wind speed during a Typhoon track:

In [5]:

```
plt.scatter(yutu["Pressure (hPa)"], yutu["Wind (kt)"],yutu["Class"]*20)
plt.xlabel('Pressure, hPa')
plt.ylabel('Wind, kt')
plt.title("Pressure versus Wind")
plt.show()
```



Observation

These data points don't follow the linear relation as other data points, why?

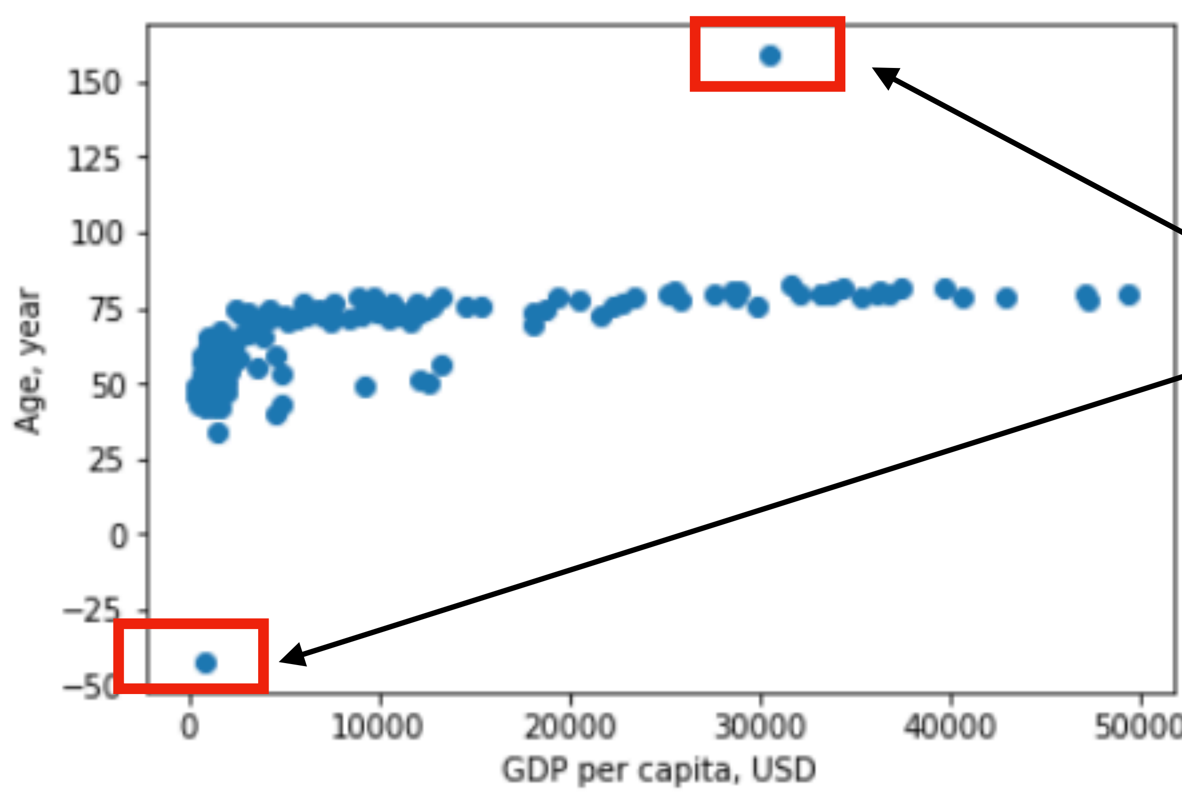
Outliers in a dataset

Here's another example you've worked on yesterday:

In [11]:

```python
file = "datasets/gdp_life.csv"
gdp = pd.read_csv(file)

print(gdp.head())

plt.scatter(gdp['GDP'],gdp['Life Exp'])
plt.xlabel('GDP per capita, USD')
plt.ylabel('Age, year')
plt.title('GDP versus Life Expectancy')
plt.show()
```

```
            GDP  Life Exp  Population      continent
0   49357.19017    80.196    4.627926         Europe
1   47306.98978    77.588    2.505559           Asia
2   47143.17964    79.972    4.553009           Asia
3   42951.65309    78.242  301.139947  North America
4   40675.99635    78.885    4.109086         Europe
```



Observation

These data points are way out of the range, why?

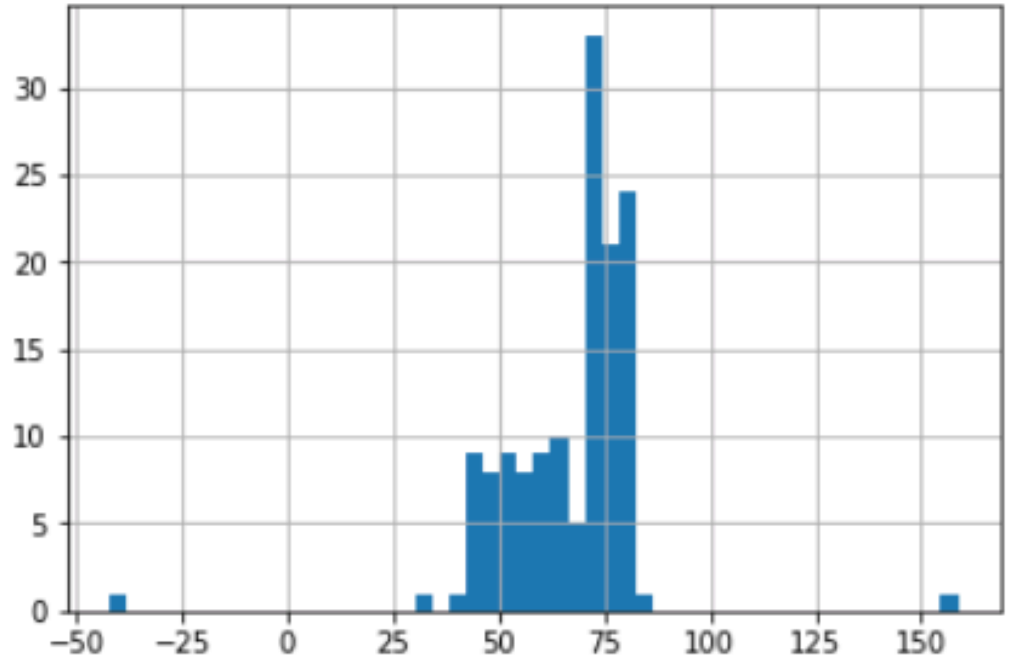
Find outliers in your data set: histograms

In [14]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

file = "datasets/gdp_life.csv"
gdp = pd.read_csv(file)
gdp.head()
```

Out[14]:

	GDP	Life Exp	Population	continent
0	49357.19017	80.196	4.627926	Europe
1	47306.98978	77.588	2.505559	Asia
2	47143.17964	79.972	4.553009	Asia
3	42951.65309	78.242	301.139947	North America
4	40675.99635	78.885	4.109086	Europe

In [17]:
```python
gdp['Life Exp'].hist(bins=50)
plt.show()
```



Outliers in a dataset

Typical data outliers:

- **NaN**s: invalid data or measurements
- **zero**s: possibly bad data
- **out of range**: way too large or too small
- **non-physical data**: e.g., negative age, negative pressure

To remove/fix these bad data points in a data file, it's called "data wrangling", it's a basic step towards sophisticated data analysis. In lots of data analysis practices, preparing the datasets for processing, is more than 50% of the job!

Data wrangling is relatively straightforward in Pandas (could be tricky), and you improve as you practise more.

Data Wrangling: Remove NaNs

Recall that there are a couple of NaNs in the last a few lines of the Yutu dataFrame:

In [6]:
```
yutu = pd.read_excel("datasets/Yutu.xlsx")
yutu.tail()
```

Out[6]:

	ear	Month	Day	Hour	Lat.	Long.	Pressure (hPa)	Wind (kt)	Class
49	2018	11	2	0	20.7	116.4	1008	35	3.0
50	2018	11	2	6	20.7	116.1	1008	200	2.0
51	2018	11	2	12	20.5	116.0	1012	0	NaN
52	2018	11	2	18	20.2	115.9	1012	0	NaN
53	2018	11	3	0	19.9	115.7	1014	0	NaN

Let's drop the NaN data rows using the **dropna()** function:

In [7]:
```
yutu = pd.read_excel("datasets/Yutu.xlsx")
yutu_no_nan = yutu.dropna()
yutu_no_nan.tail()
```

Function: `dropna()`
Syntax: DataFram.`dropna()`

Out[7]:

	ear	Month	Day	Hour	Lat.	Long.	Pressure (hPa)	Wind (kt)	Class
46	2018	11	1	6	19.9	116.8	994	45	3.0
47	2018	11	1	12	20.2	116.8	994	45	3.0
48	2018	11	1	18	20.6	116.8	1000	40	3.0
49	2018	11	2	0	20.7	116.4	1008	35	3.0
50	2018	11	2	6	20.7	116.1	1008	200	2.0

What the dropna() function does is just simply **remove** all the rows with NaNs in them

Data Wrangling: Change NaNs to something else

You could also fill the NaNs with zeros by using the **fillna()** function with an argument 0:

In [30]:
```
yutu = pd.read_excel("datasets/Yutu.xlsx")
yutu.tail()
```

Out[30]:

	ear	Month	Day	Hour	Lat.	Long.	Pressure (hPa)	Wind (kt)	Class
49	2018	11	2	0	20.7	116.4	1008	35	3.0
50	2018	11	2	6	20.7	116.1	1008	200	2.0
51	2018	11	2	12	20.5	116.0	1012	0	NaN
52	2018	11	2	18	20.2	115.9	1012	0	NaN
53	2018	11	3	0	19.9	115.7	1014	0	NaN

Function: `fillna(arg)`
Syntax: DataFram.`fillna(arg)`

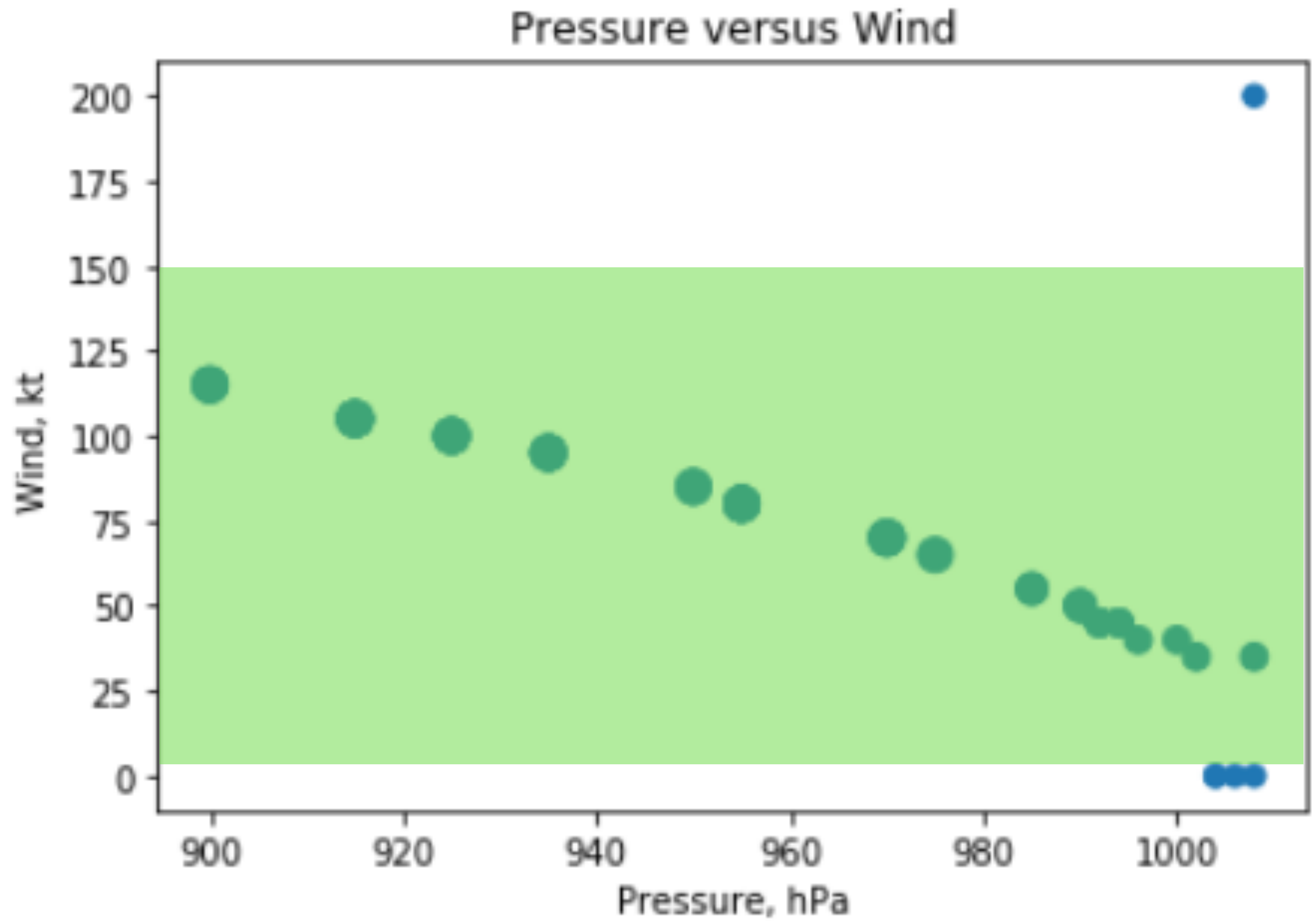
In [29]:
```
yutu_no_nan = yutu.fillna(0)
yutu_no_nan.tail()
```

Out[29]:

	ear	Month	Day	Hour	Lat.	Long.	Pressure (hPa)	Wind (kt)	Class
49	2018	11	2	0	20.7	116.4	1008	35	3.0
50	2018	11	2	6	20.7	116.1	1008	200	2.0
51	2018	11	2	12	20.5	116.0	1012	0	0.0
52	2018	11	2	18	20.2	115.9	1012	0	0.0
53	2018	11	3	0	19.9	115.7	1014	0	0.0

What the dropna() function does is just simply **replaces** all the rows with NaNs to be `arg`

Data Wrangling: Filter inappropriate (non-physical) data



Good Range

In the Typhoon Yutu data, it is clear that there are a couple of outliers in the wind speed data:

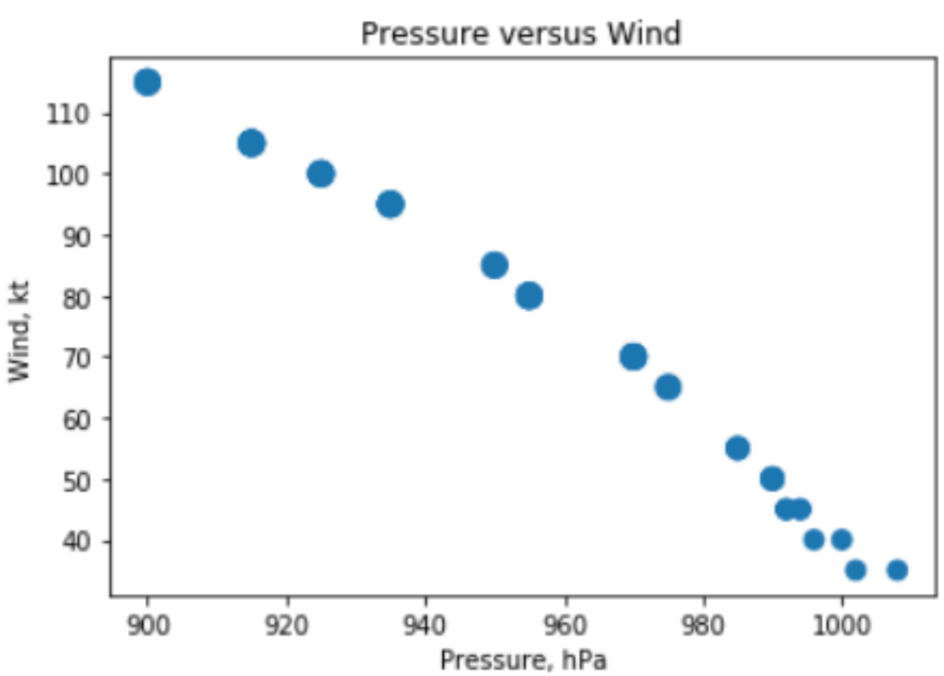
- a bunch of zeros and,
- one data point with a wind speed of 200!

Now we need to do something to exclude (or filter) these data points before doing real analysis!

Data Wrangling: Filter inappropriate (non-physical) data

Method 1, The NumPy way (column access):

In [10]:
```
yutu = pd.read_excel("datasets/Yutu.xlsx") # load data

yutu = yutu.dropna() # drop NaNs first

data_good = (yutu["Wind (kt)"] > 0) & (yutu["Wind (kt)"] <= 150) # find the indices of good data points
                                                                  # 0 < wind <= 150
Pressure = yutu["Pressure (hPa)"][data_good] # using NumPy index slicing to generate a subset of the Pressure data
Wind = yutu["Wind (kt)"][data_good] # using NumPy index slicing to generate a subset of the Wind data
Class = yutu["Class"][data_good]   # the same thing for the Class of the Typhoon

plt.scatter(Pressure,Wind,Class*20) # a bubble plot
plt.xlabel('Pressure, hPa')  #labels
plt.ylabel('Wind, kt')
plt.title("Pressure versus Wind")
plt.show()
```



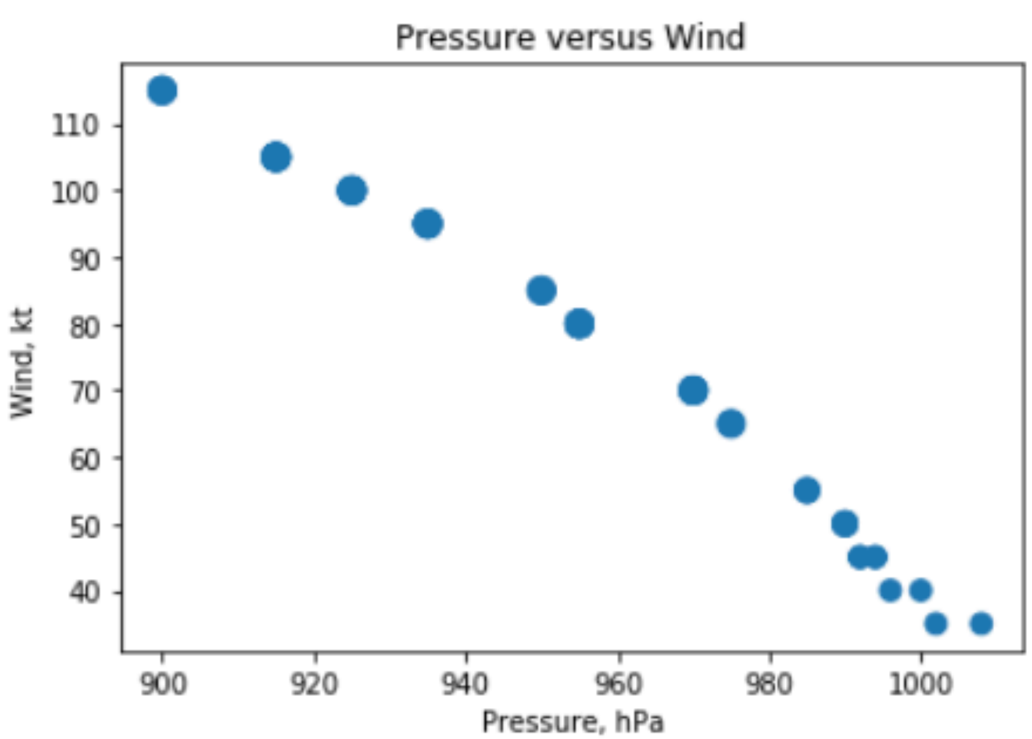
Steps:

1. **Load file into Python as a data frame**
2. **Remove NaNs**
3. **Use relational operation to set "conditions" for good data points**
4. **Apply the "conditions" to the column data using index slicing**
5. **Processing!**

Data Wrangling: Filter inappropriate (non-physical) data

Method 2, The .loc() function (row access):

In [11]:
```
yutu = pd.read_excel("datasets/Yutu.xlsx") # load data

yutu = yutu.dropna() # drop NaNs first

yutu_good = yutu.loc[ (yutu["Wind (kt)"]>0) & (yutu["Wind (kt)"]<=150) ]

# now lets plot the data in the new data Frame called yutu_good
plt.scatter(yutu_good["Pressure (hPa)"], yutu_good["Wind (kt)"],yutu_good["Class"]*20)
plt.xlabel('Pressure, hPa')
plt.ylabel('Wind, kt')
plt.title("Pressure versus Wind")
plt.show()
```



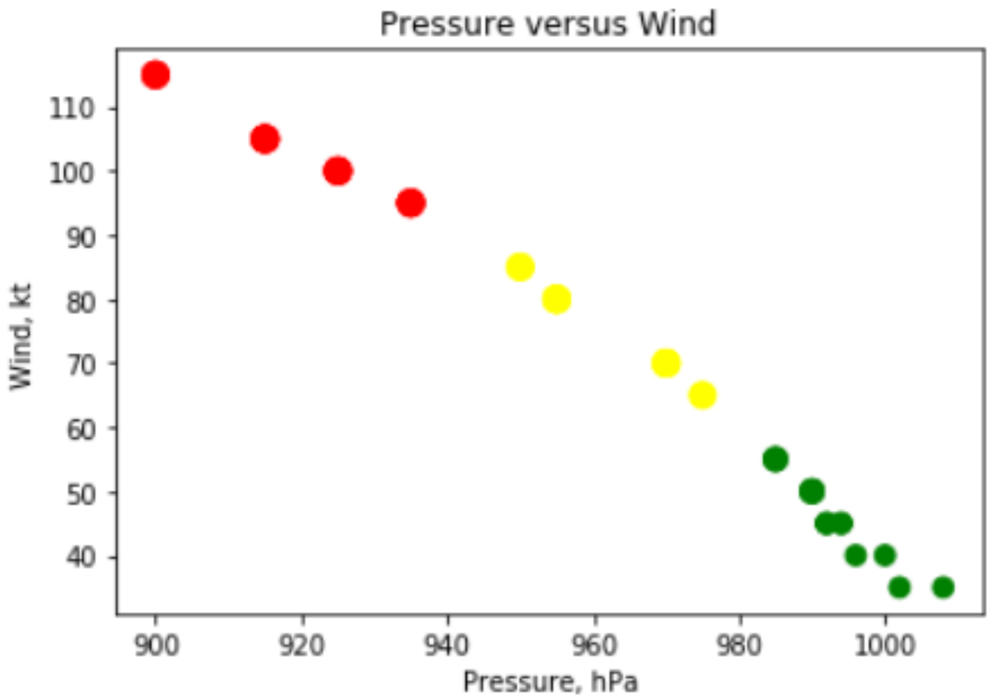
Steps:

1. **Load file into Python as a data frame**
2. **Remove NaNs**
3. **Use loc() function with relational operators to the columns to select data with appropriate range**
4. **Processing!**

Data Wrangling: The .cut() function

For numeric data points, we can use the pd.cut() function to bin the datasets, for example:

```
yutu = pd.read_excel("datasets/Yutu.xlsx") # load data

yutu = yutu.dropna() # remove NaN first
yutu_good = yutu.loc[ (yutu["Wind (kt)"]>0) & (yutu["Wind (kt)"]<=150) ] # filter data

bins = [0, 30, 60, 90, 120,150] # define 5 groups (bins) based on the wind speed
group_names = ['Calm','Light','Medium','Large','Super'] # define group names
color_names = ['blue','green','yellow','red','magenda'] # set colors to each group

yutu_good['Danger']=pd.cut(yutu_good['Wind (kt)'],bins,labels=group_names) # bin the data, create a new column
yutu_good['Color']=pd.cut(yutu_good['Wind (kt)'],bins,labels=color_names) # bin the data, create a new column

# now let's color the bubbles!
plt.scatter(yutu_good["Pressure (hPa)"], yutu_good["Wind (kt)"],yutu_good["Class"]*20, yutu_good.Color)
plt.show()

yutu_good.head()
```



	ear	Month	Day	Hour	Lat.	Long.	Pressure (hPa)	Wind (kt)	Class	Danger	Color
5	2018	10	22	0	9.4	156.1	1002	35	3.0	Light	green
6	2018	10	22	6	10.2	155.2	996	40	3.0	Light	green
7	2018	10	22	12	10.9	154.0	992	45	3.0	Light	green
8	2018	10	22	18	11.3	152.8	990	50	4.0	Light	green
9	2018	10	23	0	11.6	151.8	975	65	5.0	Medium	yellow

Data Wrangling: Filter Rows based on Conditions (sub-setting your data Frame)

Now let's see the GDP versus Life expectancy full dataset:

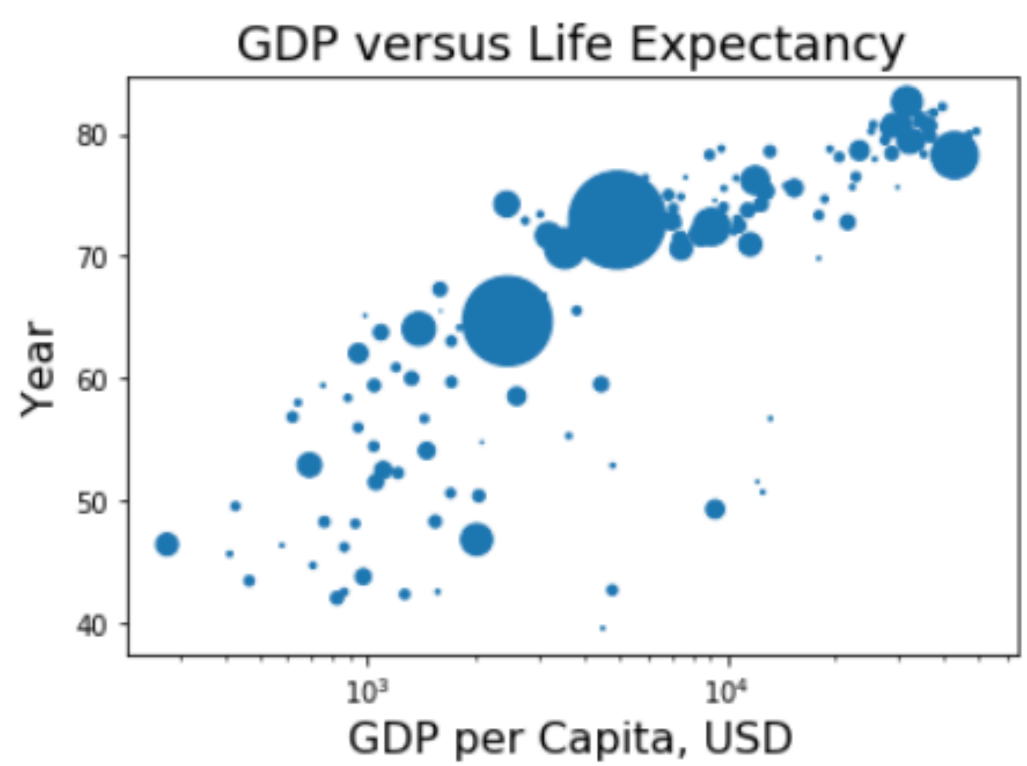
In [13]:
```python
file = "datasets/gdp_data.txt"
gdp = pd.read_csv(file)
gdp.head()
```

Out[13]:

	country	year	pop	continent	lifeExp	gdpPercap
0	Afghanistan	1952	8425333.0	Asia	28.801	779.445314
1	Afghanistan	1957	9240934.0	Asia	30.332	820.853030
2	Afghanistan	1962	10267083.0	Asia	31.997	853.100710
3	Afghanistan	1967	11537966.0	Asia	34.020	836.197138
4	Afghanistan	1972	13079460.0	Asia	36.088	739.981106

```python
file = "datasets/gdp_data.txt"
gdp = pd.read_csv(file)

gdpPC = gdp.gdpPercap[gdp.year==2007] # select year 2007
life = gdp['lifeExp'][gdp.year==2007]
pop = gdp["pop"][gdp.year==2007]/1000000

plt.scatter(gdpPC,life,pop)
plt.xlabel('GDP per Capita, USD',fontsize=16)
plt.ylabel('Year',fontsize=16)
plt.title('GDP versus Life Expectancy',fontsize=18)
plt.xscale('log')
```

- Data frame is named gdp (not a good name though)
- The data file contains data of 10 years for each country
- Can filter the data by either year or continent
  - by year: gdp[*Column*][gdp.year==2007]
  - by continent: gdp[*Column*][gdp.continent==‘Asia’]
- Process the data!



Data Wrangling: Filter Rows based on Conditions (sub-setting your data Frame)

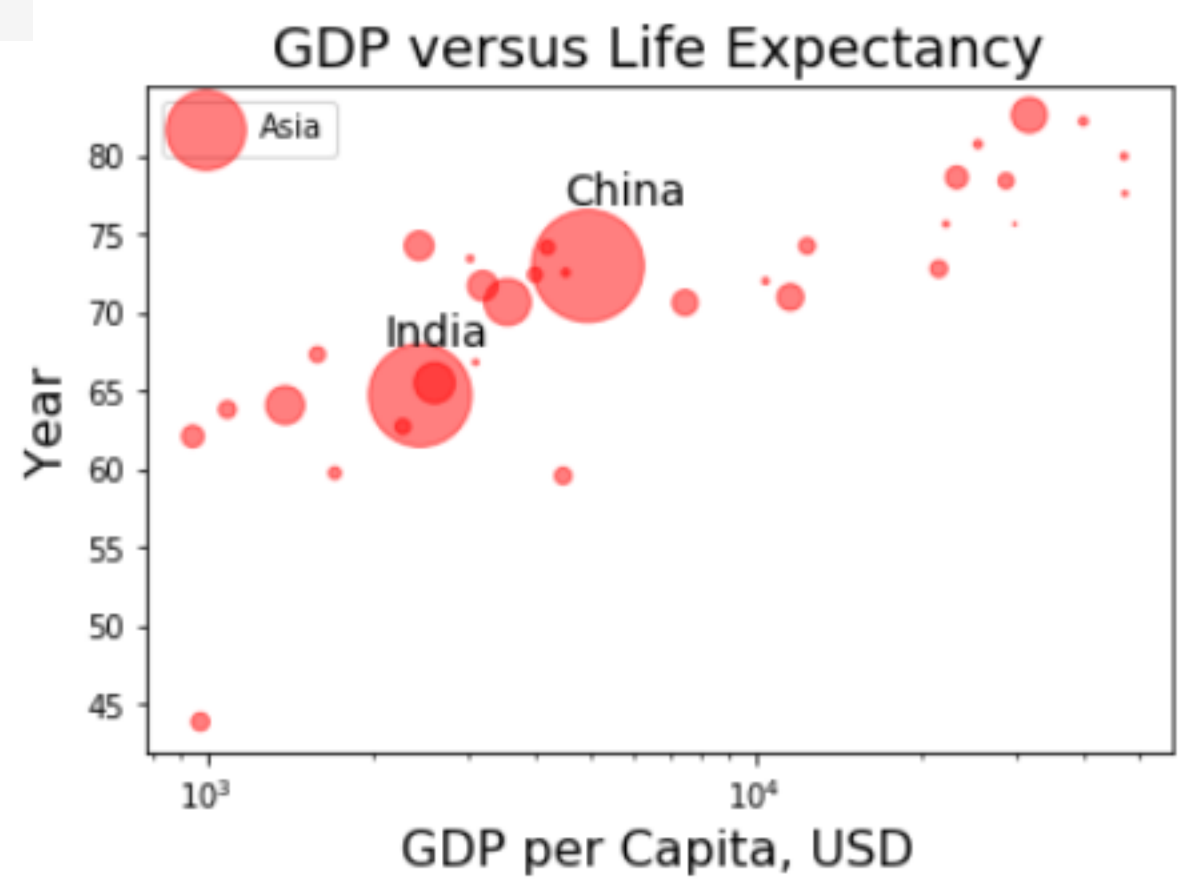
Now let's re-do the GDP versus life expectancy plot as you've done in HW 3:

```
file = "datasets/gdp_data.txt"
gdp = pd.read_csv(file)

gdpPC = gdp.gdpPercap[(gdp.year==2007)&(gdp.continent=='Asia')]
life = gdp['lifeExp'][(gdp.year==2007)&(gdp.continent=='Asia')]
pop = gdp["pop"][(gdp.year==2007)&(gdp.continent=='Asia')]/1000000
plt.scatter(gdpPC,life,pop,color='r',alpha=0.5,label='Asia')

plt.text(4500,77,'China',fontsize=14)
plt.text(2100,68,'India',fontsize=14)
plt.xscale('log')
plt.xlabel('GDP per Capita, USD',fontsize=16)
plt.ylabel('Year',fontsize=16)
plt.title('GDP versus Life Expectancy',fontsize=18)

plt.xscale('log')
plt.legend()
gdp.head()
```

	country	year	pop	continent	lifeExp	gdpPercap
0	Afghanistan	1952	8425333.0	Asia	28.801	779.445314
1	Afghanistan	1957	9240934.0	Asia	30.332	820.853030
2	Afghanistan	1962	10267083.0	Asia	31.997	853.100710
3	Afghanistan	1967	11537966.0	Asia	34.020	836.197138
4	Afghanistan	1972	13079460.0	Asia	36.088	739.981106

- Now we are doing more filtering to the datasets by using two conditions simultaneously.

Think: How to loop over all the continents use a for-loop?



Loading multiple Excel files (or csv, txt files)

Up to now, we have only opened single files and put their data into individual dataframes. Sometimes we will need to process a bunch of datasets from several Excel files in Python. How to do it?

- The **long way**: type in the _file*names* of individual data file and copy-paste the processing codes

In [35]:

```
all_data = pd.DataFrame() # create an empty data frame

# load the first data file named data1.xlsx
df = pd.read_excel("datasets/data1.xlsx")
all_data = all_data.append(df,ignore_index=True) # append to the empty data frame

# load the first data file named data2.xlsx
df = pd.read_excel("datasets/data2.xlsx")
all_data = all_data.append(df,ignore_index=True) # append to the empty data frame

# load the first data file named data3.xlsx
df = pd.read_excel("datasets/data3.xlsx")
all_data = all_data.append(df,ignore_index=True) # append to the empty data frame

all_data.head()
```

Out[35]:

	fname	age	grade
0	Baker	14	90
1	Josephine	19	100
2	Calvin	15	66
3	Aretha	17	84
4	Britanney	19	66

The long way seems to work fine. What's the problem?

Loading multiple Excel files (or csv, txt files)

Up to now, we have only opened single files and put their data into individual dataframes. Sometimes we will need to process a bunch of datasets from several Excel files in Python. How to do it?

- The **short way**: let Python do it automatically

we use the "glob" module

In [22]:
```
import glob

files = glob.glob("datasets/data*.xlsx") # generate a list of all the files with name pattern data*.xlsx

print(files)
```

```
['datasets/data3.xlsx', 'datasets/data2.xlsx', 'datasets/data1.xlsx']
```

"loop" over files using the "glob" module:

In [26]:
```
all_data = pd.DataFrame() # create an empty data frame

for f in files:
    df = pd.read_excel(f)
    print(f)
    all_data = all_data.append(df,ignore_index=True)

all_data.head()
```

```
datasets/data3.xlsx
datasets/data2.xlsx
datasets/data1.xlsx
```

Out[26]:

	fname	age	grade
0	Aretha	18	86
1	Amber	18	65
2	Serena	14	71
3	Jada	14	99
4	Althea	19	100

Format numeric strings in Python

Remember how the output of the print statement when printing float numbers, it prints out all the decimal places, which is very ugly. We can do better! To show only the first decimal place we can use *string formatting*.

The structure of a formatting statement is:

'%FMT'%(DATA),

where **FMT** is a 'format string' and **DATA** is the variable name whose value we want to format. Here is an example in which the FMT is:

3.2f.

The first number (3) is the number of characters in the output. The second number (2) is the number of characters AFTER the decimal place. The 'f' means that DATA is a floating point variable.

Other format strings include: %s for a string, %i for an integer, %e for 'scientific notation'.

For example:

In [38]:

```
Data = np.pi

print ('no formatting: ',Data) # no formatting
print ('formatted: ','%3.2f'%(Data)) # with formatting

# or can use round(Delta,1)
print ('rounded: ',round(Data,2))
```

```
no formatting:  3.141592653589793
formatted:  3.14
rounded:  3.14
```

Now let’s practice some data wrangling!