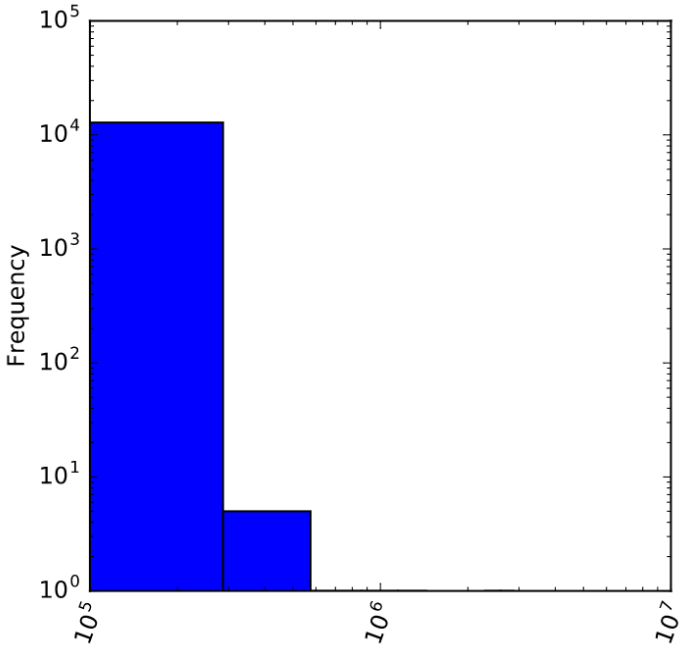
Chapter 1

1)
Visualizing single variables with histograms

Up until now, you've been looking at descriptive statistics of your data. One of the best ways to confirm what the numbers are telling you is to plot and visualize the data.
You'll start by visualizing single variables using a histogram for numeric values. The column you will work on in this exercise is `'Existing Zoning Sqft'`.
The `.plot()` method allows you to create a plot of each column of a DataFrame. The `kind` parameter allows you to specify the type of plot to use - `kind='hist'`, for example, plots a histogram.
In the IPython Shell, begin by computing summary statistics for the `'Existing Zoning Sqft'` column using the `.describe()` method. You'll notice that there are extremely large differences between the `min` and `max`values, and the plot will need to be adjusted accordingly. In such cases, it's good to look at the plot on a log scale. The keyword arguments `logx=True` or `logy=True` can be passed in to `.plot()` depending on which axis you want to rescale.

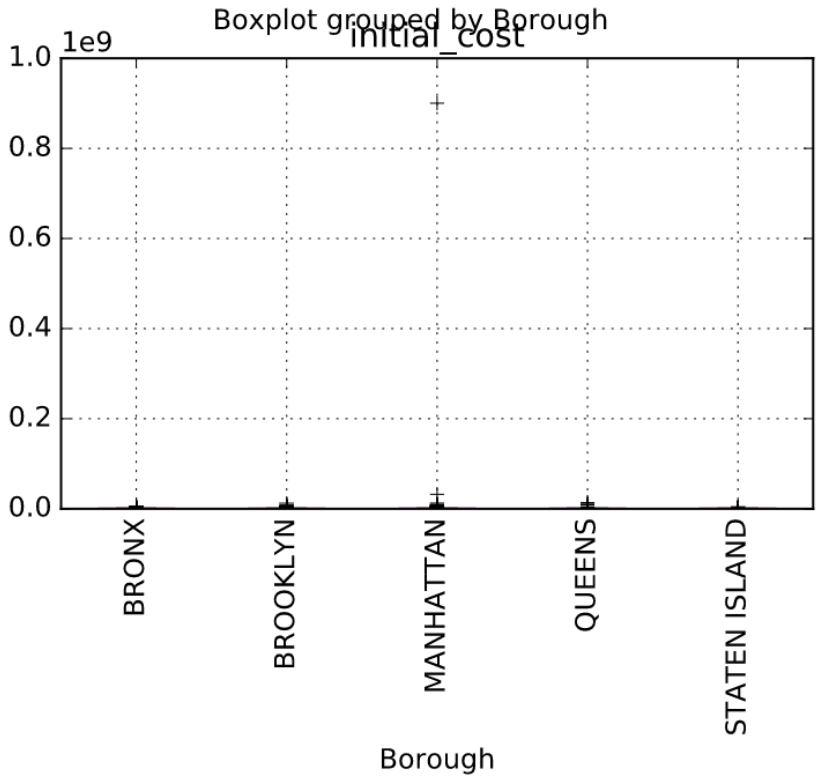
```
# Import matplotlib.pyplot
import matplotlib.pyplot as plt
# Plot the histogram
df['Existing Zoning Sqft'].plot(kind='hist', rot=70, logx=True, logy=True)
# Display the histogram
plt.show()
```



2)
Visualizing multiple variables with boxplots

The pandas `.boxplot()` method is a quick way to do this, in which you have to specify the `column` and `by` parameters. Here, you want to visualize how `'initial_cost'` varies *by* `'Borough'`.

```
# Import necessary modules
import pandas as pd
import matplotlib.pyplot as plt

# Create the boxplot
df.boxplot(column="initial_cost", by="Borough", rot=90)

# Display the plot
plt.show()
```

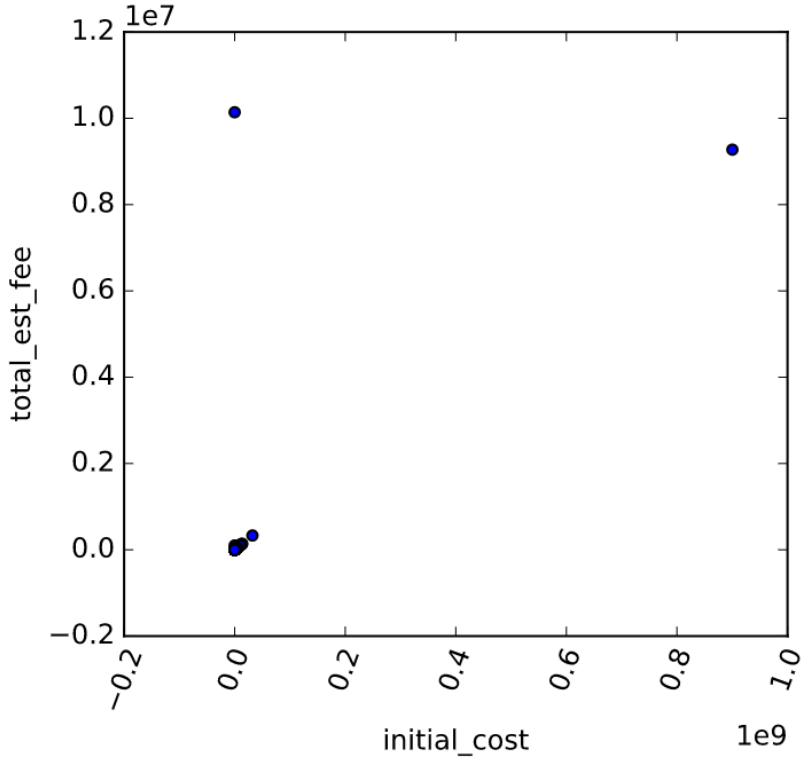


=> 2 outliers of MANHATTAN

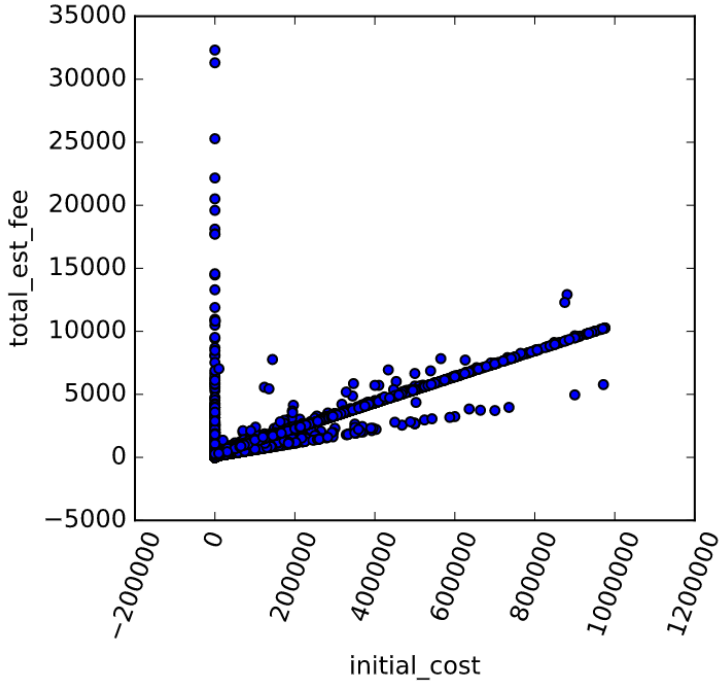
3)
Visualizing multiple variables with scatter plots

```
# Import necessary modules
import pandas as pd
```

```
import matplotlib.pyplot as plt

# Create and display the first scatter plot
df.plot(kind="scatter", x="initial_cost", y="total_est_fee", rot=70)
plt.show()
```




```
# Create and display the second scatter plot
df_subset.plot(kind="scatter",x="initial_cost",y="total_est_fee",rot=70)
plt.show()
```



Trong hình trên thì không thấy gì vì những ông outlier đã làm mất đi correlation và có quá nhiều ông = 0. Nhưng ở hình dưới có thể thấy rõ correlation giữa initial_cost và total_est_fee

Chapter2

1)

Recognizing tidy data

For data to be tidy, it must have:

- Each variable as a separate column.
- Each row as a separate observation.

Exactly! Notice that the variable column of df2 contains the values Solar.R, Ozone, Temp, and Wind. For it to be tidy, these should all be in separate columns, as in df1.

2)

In this exercise, you will practice melting a DataFrame using `pd.melt()`. There are two parameters you should be aware of: `id_vars` and `value_vars`. The `id_vars` represent the columns of the data you **do not** want to melt (i.e., keep it in its current shape), while the `value_vars` represent the columns you **do** wish to melt into rows. By default, if no `value_vars` are provided, all columns not set in the `id_vars` will be melted. This could save a bit of typing, depending on the number of columns that need to be melted.

```
# Print the head of airquality
print(airquality.head())
```

```
   Ozone  Solar.R  Wind  Temp  Month  Day
0   41.0    190.0   7.4    67      5    1
1   36.0    118.0   8.0    72      5    2
2   12.0    149.0  12.6    74      5    3
3   18.0    313.0  11.5    62      5    4
4    NaN      NaN  14.3    56      5    5
```

```
# Melt airquality: airquality_melt
airquality_melt = pd.melt(airquality, id_vars=["Month", "Day"])

# Print the head of airquality_melt
print(airquality_melt.head())
```

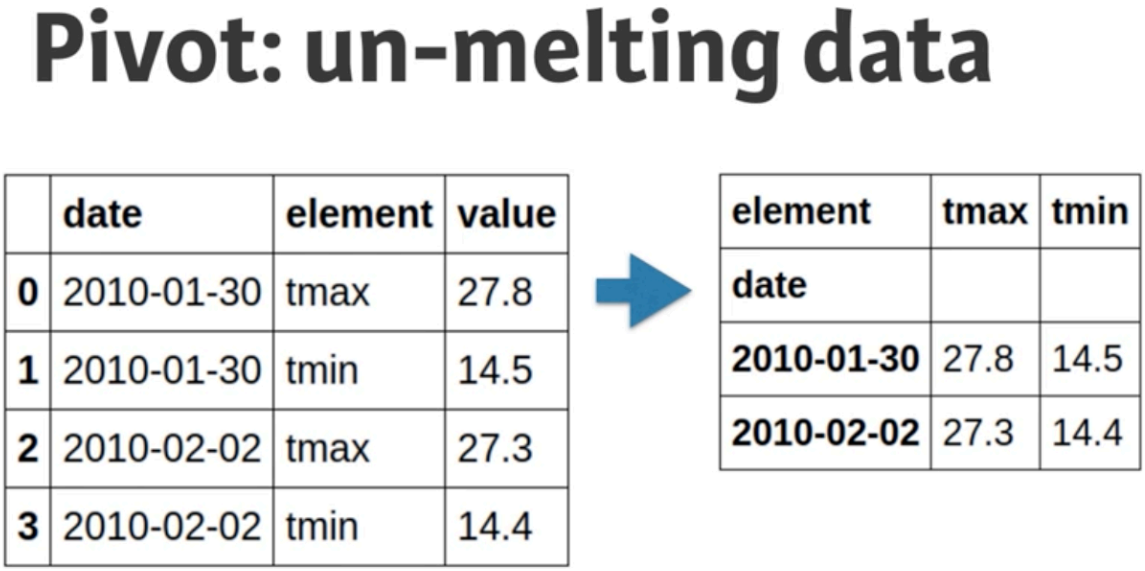
```
   Month  Day variable  value
0      5    1    Ozone   41.0
1      5    2    Ozone   36.0
2      5    3    Ozone   12.0
3      5    4    Ozone   18.0
4      5    5    Ozone    NaN
```

3)

```
# Print the head of airquality
print(airquality.head())

# Melt airquality: airquality_melt
airquality_melt = pd.melt(airquality, id_vars=["Month", "Day"],
var_name="measurement", value_name="reading")

# Print the head of airquality_melt
print(airquality_melt.head())
```

```
   Month  Day measurement  reading
0      5    1       Ozone     41.0
1      5    2       Ozone     36.0
2      5    3       Ozone     12.0
3      5    4       Ozone     18.0
4      5    5       Ozone      NaN
```

Note)



Nhưng với dữ liệu sau thì không thể Pivot được:

	date	element	value
0	2010-01-30	tmax	27.8
1	2010-01-30	tmin	14.5
2	2010-02-02	tmax	27.3
3	2010-02-02	tmin	14.4
4	2010-02-02	tmin	16.4

Bởi vì có 2 thằng cùng index mà lại có cùng giá trị tmin, python không biết giải quyết ra sao => Pivot_table ra đời để xử lý với những thằng bị duplicate như thế này.

4)

tb.head()

```
country  year  m014  m1524  m2534  m3544  m4554  m5564   m65  mu  f014  \
0      AD  2000   0.0    0.0    1.0    0.0    0.0    0.0   0.0 NaN   NaN
1      AE  2000   2.0    4.0    4.0    6.0    5.0   12.0  10.0 NaN   3.0
2      AF  2000  52.0  228.0  183.0  149.0  129.0   94.0  80.0 NaN  93.0
3      AG  2000   0.0    0.0    0.0    0.0    0.0    0.0   1.0 NaN   1.0
4      AL  2000   2.0   19.0   21.0   14.0   24.0   19.0  16.0 NaN   3.0

   f1524  f2534  f3544  f4554  f5564   f65  fu
0    NaN    NaN    NaN    NaN    NaN   NaN NaN
1   16.0    1.0    3.0    0.0    0.0   4.0 NaN
2  414.0  565.0  339.0  205.0   99.0  36.0 NaN
3    1.0    1.0    0.0    0.0    0.0   0.0 NaN
4   11.0   10.0    8.0    8.0    5.0  11.0 NaN
```

```
# Melt tb: tb_melt
tb_melt = pd.melt(tb, id_vars=["country", "year"])

# Create the 'gender' column
tb_melt['gender'] = tb_melt.variable.str[0]
```

```
# Create the 'age_group' column
tb_melt['age_group'] = tb_melt.variable.str[1:]

# Print the head of tb_melt
print(tb_melt.head())
```

```
  country  year variable  value gender age_group
0      AD  2000     m014    0.0      m       014
1      AE  2000     m014    2.0      m       014
2      AF  2000     m014   52.0      m       014
3      AG  2000     m014    0.0      m       014
4      AL  2000     m014    2.0      m       014
```

5)

```
#
ebola.head()
```

```
Date  Day  Cases_Guinea  Cases_Liberia  Cases_SierraLeone  \
0   1/5/2015  289        2776.0           NaN          10030.0
1   1/4/2015  288        2775.0           NaN           9780.0
2   1/3/2015  287        2769.0        8166.0           9722.0
3   1/2/2015  286           NaN        8157.0              NaN
4  12/31/2014  284        2730.0        8115.0           9633.0

  Cases_Nigeria  Cases_Senegal  Cases_UnitedStates  Cases_Spain  Cases_Mali  \
0           NaN           NaN                 NaN          NaN         NaN
1           NaN           NaN                 NaN          NaN         NaN
2           NaN           NaN                 NaN          NaN         NaN
3           NaN           NaN                 NaN          NaN         NaN
4           NaN           NaN                 NaN          NaN         NaN

  Deaths_Guinea  Deaths_Liberia  Deaths_SierraLeone  Deaths_Nigeria  \
0        1786.0            NaN              2977.0             NaN
1        1781.0            NaN              2943.0             NaN
2        1767.0         3496.0              2915.0             NaN
3           NaN         3496.0                 NaN             NaN
4        1739.0         3471.0              2827.0             NaN

  Deaths_Senegal  Deaths_UnitedStates  Deaths_Spain  Deaths_Mali
0            NaN                  NaN           NaN          NaN
1            NaN                  NaN           NaN          NaN
2            NaN                  NaN           NaN          NaN
3            NaN                  NaN           NaN          NaN
4            NaN                  NaN           NaN          NaN
```

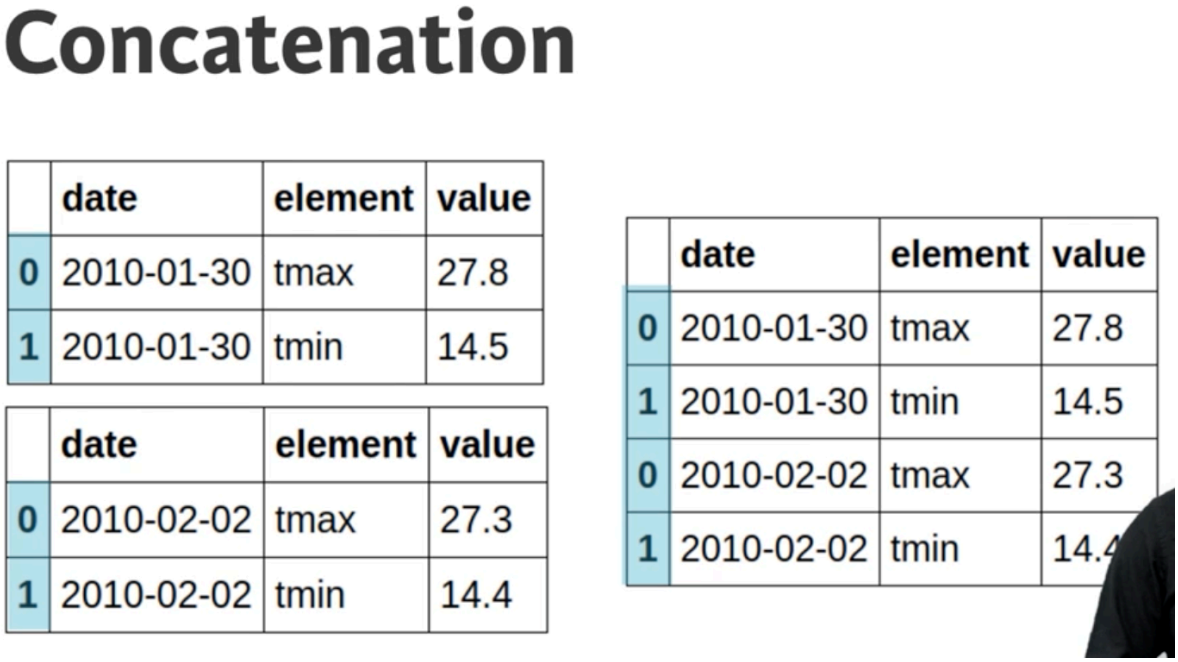
```
# Melt ebola: ebola_melt
ebola_melt = pd.melt(ebola, id_vars=["Date", "Day"],
var_name="type_country", value_name="counts")
```

```
# Create the 'str_split' column
ebola_melt['str_split'] = ebola_melt["type_country"].str.split("_")

# Create the 'type' column
ebola_melt['type'] = ebola_melt['str_split'].str.get(0)

# Create the 'country' column
ebola_melt['country'] = ebola_melt['str_split'].str.get(1)

# Print the head of ebola_melt
print(ebola_melt.head())
```

```
         Date  Day  type_country  counts       str_split   type country
0    1/5/2015  289  Cases_Guinea  2776.0  [Cases, Guinea]  Cases  Guinea
1    1/4/2015  288  Cases_Guinea  2775.0  [Cases, Guinea]  Cases  Guinea
2    1/3/2015  287  Cases_Guinea  2769.0  [Cases, Guinea]  Cases  Guinea
3    1/2/2015  286  Cases_Guinea     NaN  [Cases, Guinea]  Cases  Guinea
4  12/31/2014  284  Cases_Guinea  2730.0  [Cases, Guinea]  Cases  Guinea
```

Chapter 3

Note)



pandas concat

```
In [4]: pd.concat([weather_p1, weather_p2], ignore_index=True)
Out[4]:
     date     element  value
0  2010-01-30  tmax     27.8
1  2010-01-30  tmin     14.5
2  2010-02-02  tmax     27.3
3  2010-02-02  tmin     14.4
```

1)

Ghép thêm cột:

```
# Concatenate ebola_melt and status_country column-wise: ebola_tidy
ebola_tidy = pd.concat([ebola_melt,status_country],axis=1)
```

2)

Use Glob to load files

```
# Import necessary modules
import glob
import pandas as pd

# Write the pattern: pattern
pattern = '*.csv'

# Save all file matches: csv_files
csv_files = glob.glob(pattern)

# Print the file names
print(csv_files)

# Load the second file into a DataFrame: csv2
csv2 = pd.read_csv(csv_files[1])

# Print the head of csv2
print(csv2.head())
```

3)

```
# Create an empty list: frames
frames = []

#  Iterate over csv_files
for csv in csv_files:

    #  Read csv into a DataFrame: df
    df = pd.read_csv(csv)

    # Append df to frames
    frames.append(df)

# Concatenate frames into a single DataFrame: uber
uber = pd.concat(frames)
```

4)

Merge

site.head

```
    name    lat    long
0   DR-1 -49.85 -128.57
1   DR-3 -47.15 -126.72
2  MSK-4 -48.87 -123.40>
```

visited.head

```
    ident   site       dated
0    619   DR-1  1927-02-08
1    734   DR-3  1939-01-07
2    837  MSK-4  1932-01-14>
```

```
# Merge the DataFrames: o2o
o2o = pd.merge(left=site, right=visited, left_on="name", right_on="site")

# Print o2o
print(o2o)
```

```
      name     lat        long        ident   site        dated
   0   DR-1    -49.85  -128.57    619   DR-1      1927-02-08
   1   DR-3    -47.15  -126.72    734   DR-3      1939-01-07
   2  MSK-4 -48.87  -123.40    837   MSK-4  1932-01-14
```

5)

site

```
name    lat    long
0   DR-1 -49.85 -128.57
1   DR-3 -47.15 -126.72
2  MSK-4 -48.87 -123.40
```

visited

```
   ident   site       dated
0    619   DR-1  1927-02-08
1    622   DR-1  1927-02-10
2    734   DR-3  1939-01-07
3    735   DR-3  1930-01-12
4    751   DR-3  1930-02-26
5    752   DR-3         NaN
6    837  MSK-4  1932-01-14
7    844   DR-1  1932-03-22
```

survey

```
    taken person quant  reading
0     619   dyer   rad     9.82
1     619   dyer   sal     0.13
2     622   dyer   rad     7.80
3     622   dyer   sal     0.09
4     734     pb   rad     8.41
5     734   lake   sal     0.05
6     734     pb  temp   -21.50
7     735     pb   rad     7.22
8     735    NaN   sal     0.06
9     735    NaN  temp   -26.00
10    751     pb   rad     4.35
11    751     pb  temp   -18.50
12    751   lake   sal     0.10
13    752   lake   rad     2.19
14    752   lake   sal     0.09
15    752   lake  temp   -16.00
16    752    roe   sal    41.60
17    837   lake   rad     1.46
18    837   lake   sal     0.21
19    837    roe   sal    22.50
20    844    roe   rad    11.25
```

```
# Merge site and visited: m2m
m2m = pd.merge(left=site, right=visited, left_on="name", right_on="site")
m2m.head(5)
```

```
   name   lat   long  ident  site      dated
0  DR-1 -49.85 -128.57    619  DR-1  1927-02-08
1  DR-1 -49.85 -128.57    622  DR-1  1927-02-10
2  DR-1 -49.85 -128.57    844  DR-1  1932-03-22
3  DR-3 -47.15 -126.72    734  DR-3  1939-01-07
4  DR-3 -47.15 -126.72    735  DR-3  1930-01-12
```

# Merge m2m and survey: m2m
m2m = pd.merge(left=m2m, right=survey, left_on="ident", right_on="taken")
z# Print the first 5 lines of m2m
print(m2m.head(5))

```
   name   lat   long  ident  site      dated
0  DR-1 -49.85 -128.57    619  DR-1  1927-02-08
1  DR-1 -49.85 -128.57    622  DR-1  1927-02-10
2  DR-1 -49.85 -128.57    844  DR-1  1932-03-22
3  DR-3 -47.15 -126.72    734  DR-3  1939-01-07
4  DR-3 -47.15 -126.72    735  DR-3  1930-01-12
   name   lat   long  ident  site      dated  taken person quant  reading
0  DR-1 -49.85 -128.57    619  DR-1  1927-02-08    619   dyer   rad     9.82
1  DR-1 -49.85 -128.57    619  DR-1  1927-02-08    619   dyer   sal     0.13
2  DR-1 -49.85 -128.57    622  DR-1  1927-02-10    622   dyer   rad     7.80
3  DR-1 -49.85 -128.57    622  DR-1  1927-02-10    622   dyer   sal     0.09
4  DR-1 -49.85 -128.57    844  DR-1  1932-03-22    844    roe   rad    11.25
```

Chapter4

Note)

Cleaning bad data

```
In [5]: df['treatment a'] = pd.to_numeric(df['treatment a'],
   ...:                                   errors='coerce')

In [6]: df.dtypes
Out[6]:
name              object
sex             category
treatment a      float64
treatment b       object
dtype: object
```

1)

tips.info()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 244 entries, 0 to 243
Data columns (total 7 columns):
total_bill    244 non-null float64
tip           244 non-null float64
sex           244 non-null object
smoker        244 non-null object
day           244 non-null object
time          244 non-null object
size          244 non-null int64
dtypes: float64(2), int64(1), object(4)
memory usage: 13.4+ KB
```

tips.head(5)

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4

tips.dtypes

```
total_bill    float64
tip           float64
sex            object
smoker         object
day            object
time           object
size            int64
dtype: object
```

```
# Convert the sex column to type 'category'
tips.sex = tips.sex.astype("category")

# Convert the smoker column to type 'category'
tips.smoker = tips.smoker.astype("category")

# Print the info of tips
```

```
print(tips.info())
<script.py> output:
   <class 'pandas.core.frame.DataFrame'>
   RangeIndex: 244 entries, 0 to 243
   Data columns (total 7 columns):
   total_bill    244 non-null float64
   tip           244 non-null float64
   sex           244 non-null category
   smoker        244 non-null category
   day           244 non-null object
   time          244 non-null object
   size          244 non-null int64
   dtypes: category(2), float64(2), int64(1), object(2)
   memory usage: 10.1+ KB
   None
```

=> giảm size lưu trên memory từ 13KB -> 10KB

2)

```
print(tips.info())
   <class 'pandas.core.frame.DataFrame'>
   RangeIndex: 244 entries, 0 to 243
   Data columns (total 7 columns):
   total_bill    244 non-null object
   tip           244 non-null object
   sex           234 non-null category
   smoker        229 non-null category
   day           243 non-null category
   time          227 non-null category
   size          231 non-null float64
   dtypes: category(4), float64(1), object(2)
   memory usage: 6.9+ KB
# Convert 'total_bill' to a numeric dtype
tips['total_bill'] = pd.to_numeric(tips['total_bill'], errors="coerce")

# Convert 'tip' to a numeric dtype
tips['tip'] = pd.to_numeric(tips['tip'],errors="coerce")

# Print the info of tips
print(tips.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 244 entries, 0 to 243
Data columns (total 7 columns):
total_bill    202 non-null float64
tip           220 non-null float64
sex           234 non-null category
smoker        229 non-null category
day           243 non-null category
time          227 non-null category
size          231 non-null float64
dtypes: category(4), float64(3)
memory usage: 6.9 KB
```

3)

`\d` is the pattern required to find digits. This should be followed with a `+` so that the previous element is matched one or more times. This ensures that `10` is viewed as one number and not as `1` and

```
# Import the regular expression module
import re

# Find the numeric values: matches
matches = re.findall('\d+', 'the recipe calls for 10 strawberries and 1 banana')

# Print the matches
print(matches)
```

```
['10', '1']
```

4)

Use `\d{x}` to match `x` digits. Here you'll need to use it three times: twice to match `3` digits, and once to match `4` digits.

```
import re

# Compile the pattern: prog
prog = re.compile('\d{3}-\d{3}-\d{4}')

# See if the pattern matches
result = prog.match('123-456-7890')
print(bool(result))
```

```
True
```

```
# See if the pattern matches
result = prog.match('1123-456-7890')
```

```
print(bool(result))
```
```
False
```

5)

- A string of the format: A dollar sign, an arbitrary number of digits, a decimal point, 2 digits.
    - Use `\$` to match the dollar sign, `\d*` to match an arbitrary number of digits, `\.` to match the decimal point, and `\d{x}`to match `x` number of digits.
- A capital letter, followed by an arbitrary number of alphanumeric characters.
    - Use `[A-Z]` to match any capital letter followed by `\w*` to match an arbitrary number of alphanumeric characters.

```
# Write the first pattern
pattern1 = bool(re.match(pattern='\d{3}-\d{3}-\d{4}', string='123-456-7890'))
print(pattern1)

# Write the second pattern
pattern2 = bool(re.match(pattern='\$\d*\.\d{2}', string='$123.45'))
print(pattern2)

# Write the third pattern
pattern3 = bool(re.match(pattern='[A-Z]\w*', string='Australia'))
print(pattern3)
```

6)

```
print(tips.head())
```
```
   total_bill   tip     sex smoker  day    time  size
0       16.99  1.01  Female     No  Sun  Dinner   2.0
1       10.34  1.66    Male     No  Sun  Dinner   NaN
2       21.01  3.50    Male     No  Sun  Dinner   3.0
3       23.68  3.31    Male     No  Sun  Dinner   2.0
4         NaN   NaN  Female     No  Sun  Dinner   4.0
```
```
# Define recode_sex()
def recode_sex(sex_value):

    # Return 1 if sex_value is 'Male'
    if sex_value == "Male":
        return 1
```

```
    # Return 0 if sex_value is 'Female'
    elif sex_value == "Female":
        return 0

    # Return np.nan
    else:
        return np.nan

# Apply the function to the sex column
tips['sex_recode'] = tips['sex'].apply(recode_sex)

# Print the first five rows of tips
print(tips.head())
```

```
   total_bill   tip     sex smoker  day    time  size  sex_recode
0       16.99  1.01  Female     No  Sun  Dinner   2.0         0.0
1       10.34  1.66    Male     No  Sun  Dinner   3.0         1.0
2       21.01  3.50    Male     No  Sun  Dinner   3.0         1.0
3       23.68  3.31    Male     No  Sun  Dinner   2.0         1.0
4       24.59  3.61  Female     No  Sun  Dinner   4.0         0.0
```

7)Lambda function

```
# Write the lambda function using replace
tips['total_dollar_replace'] = tips.total_dollar.apply(lambda x: x.replace('$', ''))

# Write the lambda function using regular expressions
tips['total_dollar_re'] = tips.total_dollar.apply(lambda x: re.findall('\d+\.\d+',
x)[0])

# Print the head of tips
print(tips.head())
```

```
   total_bill   tip     sex smoker  day    time  size total_dollar  \
0       16.99  1.01  Female     No  Sun  Dinner     2      $16.99
1       10.34  1.66    Male     No  Sun  Dinner     3      $10.34
2       21.01  3.50    Male     No  Sun  Dinner     3      $21.01
3       23.68  3.31    Male     No  Sun  Dinner     2      $23.68
4       24.59  3.61  Female     No  Sun  Dinner     4      $24.59

   total_dollar_replace total_dollar_re
0                 16.99           16.99
1                 10.34           10.34
2                 21.01           21.01
3                 23.68           23.68
4                 24.59           24.59
```

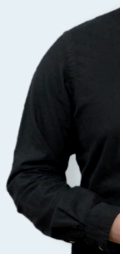
Note)

Count missing values

```
In [3]: tips_nan.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 244 entries, 0 to 243
Data columns (total 7 columns):
total_bill    202 non-null float64
tip           220 non-null float64
sex           234 non-null object
smoker        229 non-null object
day           243 non-null object
time          227 non-null object
size          231 non-null float64
dtypes: float64(3), object(4)
memory usage: 13.4+ KB
None
```

Fill missing values

```
In [6]: tips_nan['sex'] = tips_nan['sex'].fillna('missing')

In [7]: tips_nan[['total_bill', 'size']] = tips_nan[['total_bill',
   ...:                                              'size']].fillna(0)

In [8]: tips_nan.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 244 entries, 0 to 243
Data columns (total 7 columns):
total_bill    244 non-null float64
tip           220 non-null float64
sex           244 non-null object
smoker        229 non-null object
day           243 non-null object
time          227 non-null object
size          244 non-null float64
dtypes: float64(3), object(4)
memory usage: 13.4+ KB
```



8)

drop duplicates

```
# Create the new DataFrame: tracks
tracks = billboard[['year','artist','track','time']]

# Print info of tracks
print(tracks.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 24092 entries, 0 to 24091
Data columns (total 4 columns):
year      24092 non-null int64
artist    24092 non-null object
track     24092 non-null object
time      24092 non-null object
dtypes: int64(1), object(3)
memory usage: 753.0+ KB
```

```
# Drop the duplicates: tracks_no_duplicates
tracks_no_duplicates = tracks.drop_duplicates()

# Print info of tracks
print(tracks_no_duplicates.info())
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 317 entries, 0 to 316
Data columns (total 4 columns):
year      317 non-null int64
artist    317 non-null object
track     317 non-null object
time      317 non-null object
dtypes: int64(1), object(3)
memory usage: 12.4+ KB
```

9)

```
print(airquality.info())
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 153 entries, 0 to 152
Data columns (total 6 columns):
Ozone      86 non-null float64
Solar.R    146 non-null float64
Wind       153 non-null float64
Temp       153 non-null int64
Month      153 non-null int64
Day        153 non-null int64
dtypes: float64(3), int64(3)
memory usage: 7.2 KB
```

```
# Calculate the mean of the Ozone column: oz_mean
oz_mean = airquality.Ozone.mean()

# Replace all the missing values in the Ozone column with the mean
airquality['Ozone'] = airquality['Ozone'].fillna(oz_mean)

# Print the info of airquality
```

print(airquality.info())

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 153 entries, 0 to 152
Data columns (total 6 columns):
Ozone      153 non-null float64
Solar.R    146 non-null float64
Wind       153 non-null float64
Temp       153 non-null int64
Month      153 non-null int64
Day        153 non-null int64
dtypes: float64(3), int64(3)
memory usage: 7.2 KB
```

Note)

assert để test data

Test column

```
In [1]: assert google.Close.notnull().all()
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
<ipython-input-49-eec77130a77f> in <module>()
----> 1 assert google.Close.notnull().all()

AssertionError:
```

Test column

```
In [1]: google_0 = google.fillna(value=0)

In [2]: assert google_0.Close.notnull().all()
```

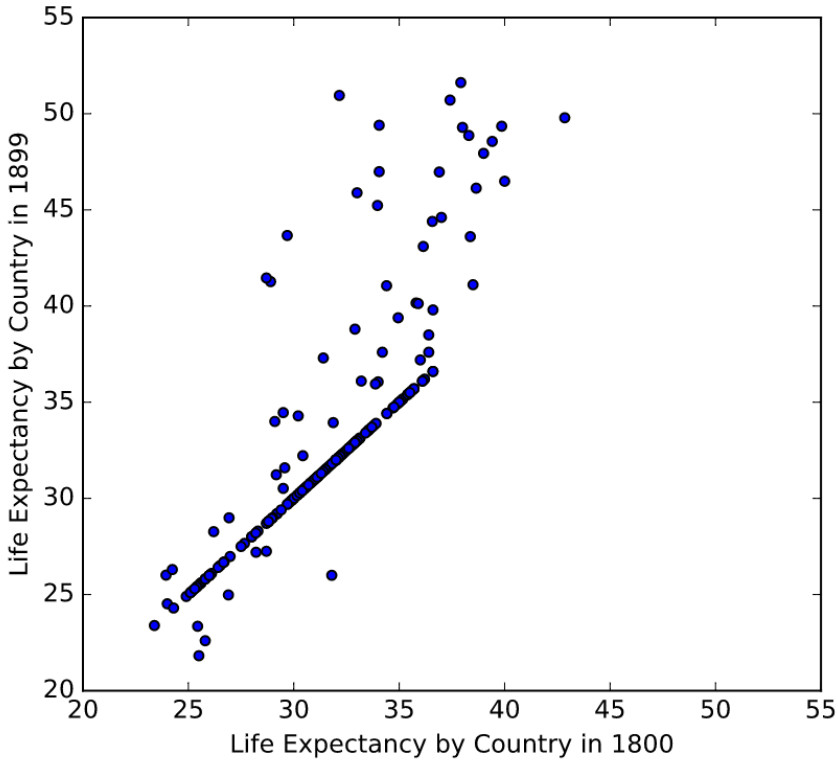
10)

```
# Assert that there are no missing values
assert pd.notnull(ebola).all().all()
## Có 2 .all() vì .all() đầu tiên trả về True, False của các columns, .all() thứ 2 để trả về True, False của tất cả column

# Assert that all values are >= 0
assert (ebola>=0).all().all()
```

Chapter 5

1)

```
# Import matplotlib.pyplot
import matplotlib.pyplot as plt

# Create the scatter plot
g1800s.plot(kind='scatter', x='1800', y='1899')

# Specify axis labels
plt.xlabel('Life Expectancy by Country in 1800')
plt.ylabel('Life Expectancy by Country in 1899')

# Specify axis limits
plt.xlim(20, 55)
plt.ylim(20, 55)

# Display the plot
plt.show()
```



2)

- Write an assert statement to test that all the values are valid for the `g1800s` DataFrame. Use the `check_null_or_valid()` function placed inside the `.apply()` method for this. Note that because you're applying it

over the entire DataFrame, and not just one column, you'll have to chain the `.all()` method twice, and remember that you don't have to use `()` for functions placed inside `.apply()`.

- Write an assert statement to make sure that each country occurs only once in the data. Use the `.value_counts()` method on the `'Life expectancy'` column for this. Specifically, index `0` of `.value_counts()` will contain the **most frequently occuring value**. If this is equal to `1` for the `'Life expectancy'` column, then you can be certain that no country appears more than once in the data.

```
def check_null_or_valid(row_data):
    """Function that takes a row of data,
    drops all missing values,
    and checks if all remaining values are greater than or equal to 0
    """
    no_na = row_data.dropna()[1:-1]
    numeric = pd.to_numeric(no_na)
    ge0 = numeric >= 0
    return ge0

# Check whether the first column is 'Life expectancy'
assert g1800s.columns[0] == "Life expectancy"

# Check whether the values in the row are valid
assert g1800s.iloc[:, 1:].apply(check_null_or_valid, axis=1).all().all()

# Check that there is only one instance of each country
assert g1800s['Life expectancy'].value_counts()[0] == 1
```

3)

```
# Concatenate the DataFrames row-wise
gapminder = pd.concat([g1800s,g1900s,g2000s])

# Print the shape of gapminder
print(gapminder.shape)

# Print the head of gapminder
print(gapminder.head())
```

4)

```
gapminder.head(1)
```

```
  1800  1801  1802  1803  1804  1805  1806  1807  1808  1809  \
0  NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN

      ...       2008  2009  2010  2011  2012  2013  2014  2015  2016  \
0     ...        NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN

  Life expectancy
0        Abkhazia

[1 rows x 218 columns]
```

```
# Melt gapminder: gapminder_melt
gapminder_melt = pd.melt(gapminder,id_vars='Life expectancy')

# Rename the columns
gapminder_melt.columns = ['country','year','life_expectancy']

# Print the head of gapminder_melt
print(gapminder_melt.head())
```

```
              country  year  life_expectancy
0            Abkhazia  1800              NaN
1         Afghanistan  1800            28.21
2  Akrotiri and Dhekelia  1800              NaN
3             Albania  1800            35.40
4             Algeria  1800            28.82
```

5)

```
# Convert the year column to numeric
gapminder.year = pd.to_numeric(gapminder.year)

# Test if country is of type object
assert gapminder.country.dtypes == np.object

# Test if year is of type int64
assert gapminder.year.dtypes == np.int64

# Test if life_expectancy is of type float64
assert gapminder.life_expectancy.dtypes == np.float64
```

6)

Create a Series called `countries` consisting of the `'country'` column of `gapminder`.

Drop all duplicates from `countries` using the `.drop_duplicates()`method.

Write a regular expression that tests your assumptions of what characters belong in `countries`:

- Anchor the pattern to match exactly what you want by placing a `^`in the beginning and `$` in the end.
- Use `A-Za-z` to match the set of lower and upper case letters, `\.`to match periods, and `\s` to match whitespace between words.

Use `str.contains()` to create a Boolean vector representing values that match the pattern.

Invert the mask by placing a `~` before it.

Subset the `countries` series using the `.loc[]` accessor and `mask_inverse`

```
# Create the series of countries: countries
countries = gapminder['country']

# Drop all the duplicates from countries
countries = countries.drop_duplicates()

# Write the regular expression: pattern
pattern = '^[A-Za-z\.\s]*$'
# ^ đầu, $ cuối là tìm chính xác.

# Create the Boolean vector: mask
mask = countries.str.contains(pattern)

# Invert the mask: mask_inverse
mask_inverse = ~mask

# Subset countries using mask_inverse: invalid_countries
invalid_countries = countries.loc[mask_inverse]

# Print invalid_countries
print(invalid_countries)
```

```
49          Congo, Dem. Rep.
50                Congo, Rep.
53           Cote d'Ivoire
73     Falkland Is (Malvinas)
93           Guinea-Bissau
98          Hong Kong, China
118    United Korea (former)\n
131           Macao, China
132         Macedonia, FYR
145     Micronesia, Fed. Sts.
161         Ngorno-Karabakh
187          St. Barthélemy
193    St.-Pierre-et-Miquelon
225           Timor-Leste
251     Virgin Islands (U.S.)
252      North Yemen (former)
253      South Yemen (former)
258                 Åland
Name: country, dtype: object
```

7)

```
# Assert that country does not contain any missing values
assert pd.notnull(gapminder.country).all()

# Assert that year does not contain any missing values
assert pd.notnull(gapminder.year).all()

# Drop the missing values
gapminder = gapminder.dropna()

# Print the shape of gapminder
print(gapminder.shape)
```

```
(43857, 3)
```

8)

```
# Add first subplot
plt.subplot(2, 1, 1)

# Create a histogram of life_expectancy
gapminder.life_expectancy.plot(kind='hist')

# Group gapminder: gapminder_agg
gapminder_agg = gapminder.groupby('year')['life_expectancy'].mean()
```

```
# Print the head of gapminder_agg
print ('head')
print(gapminder_agg.head())

# Print the tail of gapminder_agg
print(gapminder_agg.tail())

# Add second subplot
plt.subplot(2, 1, 2)

# Create a line plot of life expectancy per year
gapminder_agg.plot()

# Add title and specify axis labels
plt.title('Life expectancy over the years')
plt.ylabel('Life expectancy')
plt.xlabel('Year')

# Display the plots
plt.tight_layout()
plt.show()

# Save both DataFrames to csv files
gapminder.to_csv('gapminder.csv')
gapminder_agg.to_csv('gapminder_agg.csv')
```