Applied Statistical Methods

Introduction to Python - Part IV

Xuemao Zhang
East Stroudsburg University

January 30, 2023

Outline

- Relative path
- Pandas series
- Data Frame
- Data Input
- Data output
- Summarizing data
- Sub-setting a data frame

Relative path

- 'Reading in' data is the first step of any data project/analysis

Relative vs. absolute paths (From Wiki)

- *An **absolute or full path** points to the same location in a file system, regardless of the current working directory. To do that, it must include the **root directory**.*
- This means if I try your code, and you use absolute paths, it won't work unless we have the exact same folder structure where Python is looking (bad).
- *By contrast, a **relative path starts from some given working directory**, avoiding the need to provide the full absolute path. A filename can be considered as a relative path based at the current working directory.*

Relative path

- Using relative paths simplifies the code since you don't need to write the full absolute path in order to find a file or directory in your Python project.
- Typical directory structure syntax applies
  - "../" - goes up one level
  - "./" - is the current directory
  - "~" - is your "home" directory

pandas series

- pandas have two data stuctures
  - Series
  - DataFrame
- A **Series** is a one-dimensional array-like object containing an array of data and an associated array of **data labels**, called its **index**.

```
import pandas as pd
obj = pd.Series([4, 7, -5, 3])
print(obj)
```

```
## 0    4
## 1    7
## 2   -5
## 3    3
## dtype: int64
```

pandas series

- `pandas.Series.values` return Series as ndarray or ndarray-like depending on the dtype.
  - https://pandas.pydata.org/docs/reference/api/pandas.Series.values.html

```
obj.values #it is a numpy.ndarray
```

```
## array([ 4,  7, -5,  3], dtype=int64)
```

```
obj.index
```

```
## RangeIndex(start=0, stop=4, step=1)
```

pandas series

- A Series can be created with a dictionary

```
sdata = {'Ohio': 35000, 'Texas': 71000, 'Oregon': 16000, 'Utah': 5000}
obj2 = pd.Series(sdata)
obj2.values
```

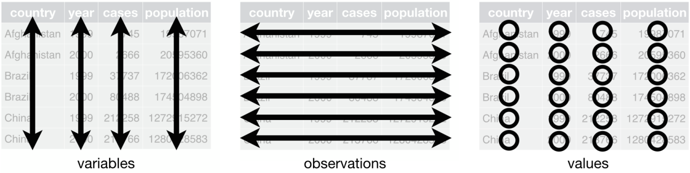
```
## array([35000, 71000, 16000,  5000], dtype=int64)
```

```
obj2.index
```

```
## Index(['Ohio', 'Texas', 'Oregon', 'Utah'], dtype='object')
```

Data Frame

- Data scientist are more comfortable dealing with tidy (Wickham and Grolemund, R for Data Science, 2017) data frames:
  - Each variable must have its own column.
  - Each observation must have its own row.
  - Each value must have its own cell.



Data Frame

- One way to create a pandas DataFrame is convert dictionaries

```
df=pd.DataFrame({'col1':[1,3,11,2], 'col2':[9,13,24,16]})
print(type(df))
```

```
## <class 'pandas.core.frame.DataFrame'>
```

```
print(df)
```

```
##    col1  col2
## 0     1     9
## 1     3    13
## 2    11    24
## 3     2    16
```

Data Frame

- Again, `pandas.DataFrame.values` return a *Numpy* representation of the DataFrame
  - Only the values in the DataFrame will be returned, the axes labels will be removed.
  - https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.values.html

```
df.index
```

```
## RangeIndex(start=0, stop=4, step=1)
```

```
df.values  # a numpy.ndarray
```

```
## array([[ 1,  9],
##        [ 3, 13],
##        [11, 24],
##        [ 2, 16]], dtype=int64)
```

Data Frame

- Or the method `pandas.DataFrame.to_numpy` can be used
  - https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.to_numpy.html

```
df2=df.to_numpy()
print(df2)
```

```
## [[ 1  9]
##  [ 3 13]
##  [11 24]
##  [ 2 16]]
```

Data Input

- The library `pandas` provides powerful tools for data import.
- We are going to focus on simple delimited files first
  - comma separated (e.g. '.csv')
  - tab delimited (e.g. '.txt')
  - Microsoft excel (e.g. '.xlsx')
- Throughout this course, we will be working with CSV files.

Data Input

- Consider the data set `earthquakes.csv` from https://raw.githubusercontent.com/stefmolin/Hands-On-Data-Analysis-with-Pandas-2nd-edition/master/ch_02/data/earthquakes.csv which is also downloaded to the folder `data`.
- https://pandas.pydata.org/docs/reference/api/pandas.read_csv.html
- The imported file is a `pandas.DataFrame`

```
earthquake= pd.read_csv("../data/earthquakes.csv", header=0)
# the data can be read from the website directly
print(type(earthquake))
```

```
## <class 'pandas.core.frame.DataFrame'>
```

Data Input

- column names.

```
vars=earthquake.columns
print(vars)
```

```
## Index(['alert', 'cdi', 'code', 'detail', 'dmin', 'felt', 'gap', 'ids', '
##        'magType', 'mmi', 'net', 'nst', 'place', 'rms', 'sig', 'sources',
##        'status', 'time', 'title', 'tsunami', 'type', 'types', 'tz', 'upd
##        'url'],
##       dtype='object')
```

```
print(type(vars))
```

```
## <class 'pandas.core.indexes.base.Index'>
```

Data Input

- `Index.to_list()` returns a list of the values.
    - https://pandas.pydata.org/docs/reference/api/pandas.Index.to_list.html

```
vars=earthquake.columns.to_list()
print(type(vars))
```

```
## <class 'list'>
```

```
print(vars)
```

```
## ['alert', 'cdi', 'code', 'detail', 'dmin', 'felt', 'gap', 'ids', 'mag',
```

Data Input

- Data types of the columns

```
print(earthquake.dtypes)
```

```
## alert       object
## cdi        float64
## code        object
## detail      object
## dmin       float64
## felt       float64
## gap        float64
## ids         object
## mag        float64
## magType     object
## mmi        float64
## net         object
## nst        float64
## place       object
## rms        float64
## sig          int64
## sources     object
## status      object
```

Data Input

- The following are some commonly used data frame attributes

Attribute	Returns
`dtypes`	The data types of each column
`shape`	Dimensions of the `DataFrame` object in a tuple of the form `(number of rows, number of columns)`
`index`	The `Index` object along the rows of the `DataFrame` object
`columns`	The name of the columns (as an `Index` object)
`values`	The data in the `DataFrame` object
`empty`	Check if the `DataFrame` object is empty

Data Input

- Check if the data frame is empty

```
print(earthquake.empty)
```

```
## False
```

- The data in the data frame

```
print(earthquake.values) #a 'numpy.ndarray'
```

```
## [[nan nan '37389218' ... -480.0 1539475395144
##   'https://earthquake.usgs.gov/earthquakes/eventpage/ci37389218']
##  [nan nan '37389202' ... -480.0 1539475253925
##   'https://earthquake.usgs.gov/earthquakes/eventpage/ci37389202']
##  [nan 4.4 '37389194' ... -480.0 1539536756176
##   'https://earthquake.usgs.gov/earthquakes/eventpage/ci37389194']
##  ...
##  [nan nan '2018261000' ... -240.0 1537243777410
##   'https://earthquake.usgs.gov/earthquakes/eventpage/pr2018261000']
##  [nan nan '38063959' ... -480.0 1537230211640
##   'https://earthquake.usgs.gov/earthquakes/eventpage/ci38063959']
##  [nan nan '38063935' ... -480.0 1537305830770
##   'https://earthquake.usgs.gov/earthquakes/eventpage/ci38063935']]
```

Data Input

- Check the dimensions
  - https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.shape.html

```
dim=earthquake.shape
print(type(dim))
```

```
## <class 'tuple'>
```

```
print(dim)
```

```
## (9332, 26)
```

Data Input

- Check the row names/index
  - https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.index.html#

```
rownames=earthquake.index
print(rownames)
```

```
## RangeIndex(start=0, stop=9332, step=1)
```

Data Input

- Get the head using the `head()` method
- https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.head.html

```
print(earthquake.head()) #Return the first n rows
```

```
##   alert  cdi  ...        updated
## 0   NaN  NaN  ...  1539475395144  https://earthquake.usgs.gov/earthquake
## 1   NaN  NaN  ...  1539475253925  https://earthquake.usgs.gov/earthquake
## 2   NaN  4.4  ...  1539536756176  https://earthquake.usgs.gov/earthquake
## 3   NaN  NaN  ...  1539475196167  https://earthquake.usgs.gov/earthquake
## 4   NaN  NaN  ...  1539477547926  https://earthquake.usgs.gov/earthquake
##
## [5 rows x 26 columns]
```

Data Input

- Get the tail using the `tail()` method
- https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.tail.html

```
print(earthquake.tail()) #Return the last n rows
```

```
##      alert  ...                                              url
## 9327   NaN  ...  https://earthquake.usgs.gov/earthquakes/eventp...
## 9328   NaN  ...  https://earthquake.usgs.gov/earthquakes/eventp...
## 9329   NaN  ...  https://earthquake.usgs.gov/earthquakes/eventp...
## 9330   NaN  ...  https://earthquake.usgs.gov/earthquakes/eventp...
## 9331   NaN  ...  https://earthquake.usgs.gov/earthquakes/eventp...
##
## [5 rows x 26 columns]
```

Data Input

- We can use the info() method to see how many non-null entries of each column we have and get information on our index.

```
print(earthquake.info()) #Return the first n rows
```

```
## <class 'pandas.core.frame.DataFrame'>
## RangeIndex: 9332 entries, 0 to 9331
## Data columns (total 26 columns):
##  #   Column   Non-Null Count  Dtype
## ---  ------   --------------  -----
##  0   alert    59 non-null     object
##  1   cdi      329 non-null    float64
##  2   code     9332 non-null   object
##  3   detail   9332 non-null   object
##  4   dmin     6139 non-null   float64
##  5   felt     329 non-null    float64
##  6   gap      6164 non-null   float64
##  7   ids      9332 non-null   object
##  8   mag      9331 non-null   float64
##  9   magType  9331 non-null   object
##  10  mmi      93 non-null     float64
##  11  net      9332 non-null   object
```

Data Input

- read_csv can be used to read a .txt file
  - quiz1.txt is a *tab-delimited* file

```
quiz=pd.read_csv("../data/quiz1.txt",header=0, delimiter='\t')
print(quiz.head())
```

```
##    ID    Q1   Q2    Q3    Q4    Q5   Q6
## 0   1   8.0  9.0  10.0   9.5  10.0  8.0
## 1   2   8.0  8.0   8.0  10.0   9.0  8.0
## 2   3  10.0  7.0  10.0  10.0  10.0  8.0
## 3   4   6.0  5.0   9.0   8.0   5.0  NaN
## 4   5  10.0  6.0   8.0   6.0   NaN  NaN
```

Data Input

- `read_excel` is used to read in Excel files
- https://pandas.pydata.org/docs/reference/api/pandas.read_excel.html?highlight=read_excel
  - `Install xlrd >= 1.0.0 for Excel support Use pip or conda to install xlrd.`

```
quiz2=pd.read_excel("../data/quiz2.xls",sheet_name=0, header=0)
print(quiz2.head())
```

```
##    ID    Q1   Q2    Q3    Q4    Q5   Q6
## 0   1   8.0  9.0  10.0   9.5  10.0  8.0
## 1   2   8.0  8.0   8.0  10.0   9.0  8.0
## 2   3  10.0  7.0  10.0  10.0  10.0  8.0
## 3   4   6.0  5.0   9.0   8.0   5.0  NaN
## 4   5  10.0  6.0   8.0   6.0   NaN  NaN
```

Data output

- Sometimes we want to save our data frame to a file so that we can share it with others.
  - https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.to_csv.html

```
df.to_csv('output.csv', index=False) #remove the row labels
```

- The following code save the data quiz to the file `quiz2.csv`

```
quiz.to_csv('quiz3.csv', sep=',', header=True)
```

- Check out the following resource in the pandas documentation for the full list of capabilities: https://pandas.pydata.org/pandas-docs/stable/user_guide/io.html

Summarizing data

- Method describe() gives us the 5-number summary, along with the count, mean, and standard deviation of the numeric columns

```
import pandas as pd
iris=pd.read_csv("../data/iris.csv",header=0, delimiter=',')
print(iris.describe())
```

```
##        Sepal.Length  Sepal.Width  Petal.Length  Petal.Width
## count    150.000000   150.000000    150.000000   150.000000
## mean       5.843333     3.057333      3.758000     1.199333
## std        0.828066     0.435866      1.765298     0.762238
## min        4.300000     2.000000      1.000000     0.100000
## 25%        5.100000     2.800000      1.600000     0.300000
## 50%        5.800000     3.000000      4.350000     1.300000
## 75%        6.400000     3.300000      5.100000     1.800000
## max        7.900000     4.400000      6.900000     2.500000
```

Summarizing data

- By default, `describe()` won't give us any information about the columns of type object, but we can either provide `include='all'` as an argument or run it separately for the data of type `np.object`

```
print(iris.describe(include='all'))
```

```
##         Sepal.Length  Sepal.Width  Petal.Length  Petal.Width Species
## count     150.000000   150.000000    150.000000   150.000000     150
## unique           NaN          NaN           NaN          NaN       3
## top              NaN          NaN           NaN          NaN  setosa
## freq             NaN          NaN           NaN          NaN      50
## mean        5.843333     3.057333      3.758000     1.199333     NaN
## std         0.828066     0.435866      1.765298     0.762238     NaN
## min         4.300000     2.000000      1.000000     0.100000     NaN
## 25%         5.100000     2.800000      1.600000     0.300000     NaN
## 50%         5.800000     3.000000      4.350000     1.300000     NaN
## 75%         6.400000     3.300000      5.100000     1.800000     NaN
## max         7.900000     4.400000      6.900000     2.500000     NaN
```

Summarizing data

- By default, `describe()` won't give us any information about the columns of type object, but we can either provide `include='all'` as an argument or run it separately for the data of type `object`

```
print(iris.describe(include=object))
```

```
##        Species
## count      150
## unique       3
## top     setosa
## freq        50
```

Summarizing data

- Sometimes, we just want a particular *statistic* for a specific column(s)
- A list of *calculation methods* for data frames

Method	Description	Data types
`count()`	The number of non-null observations	Any
`nunique()`	The number of unique values	Any
`sum()`	The total of the values	Numerical or Boolean
`mean()`	The average of the values	Numerical or Boolean
`median()`	The median of the values	Numerical
`min()`	The minimum of the values	Numerical
`idxmin()`	The index where the minimum values occurs	Numerical
`max()`	The maximum of the values	Numerical
`idxmax()`	The index where the maximum value occurs	Numerical
`abs()`	The absolute values of the data	Numerical
`std()`	The standard deviation	Numerical
`var()`	The variance	Numerical
`cov()`	The covariance between two `Series`, or a covariance matrix for all column combinations in a `DataFrame`	Numerical
`corr()`	The correlation between two `Series`, or a correlation matrix for all column combinations in a `DataFrame`	Numerical
`quantile()`	Calculates a specific quantile	Numerical
`cumsum()`	The cumulative sum	Numerical or Boolean

Summarizing data

- Distinct values of a column

```
iris['Species'].nunique()  # iris.Species.nunique()
```

```
## 3
```

```
iris['Species'].unique()
```

```
## array(['setosa', 'versicolor', 'virginica'], dtype=object)
```

- Frequency table

```
iris['Species'].value_counts()
```

```
## setosa        50
## versicolor    50
## virginica     50
## Name: Species, dtype: int64
```

Sub-setting a data frame

- We can drop a column using method `DataFrame.drop()`
  - `axis=1` means dropping columns
  - https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.drop.html

```
iris.drop(['Petal.Length','Petal.Width'],axis=1,inplace=True)
iris.columns
```

```
## Index(['Sepal.Length', 'Sepal.Width', 'Species'], dtype='object')
```

Sub-setting a data frame

- Select columns
  - using indexing
  - using dot notation

```
ID=quiz['ID']
print(type(ID))
```

```
## <class 'pandas.core.series.Series'>
```

```
print(ID)
```

```
## 0      1
## 1      2
## 2      3
## 3      4
## 4      5
## 5      6
## 6      7
## 7      8
## 8      9
## 9     10
## 10    11
## 11    12
```

Sub-setting a data frame

- Select columns
  - using indexing
  - using dot notation

```
quiz_sub=quiz[["ID","Q1","Q2","Q3"]]
print(type(quiz_sub))
```

```
## <class 'pandas.core.frame.DataFrame'>
```

```
quiz_sub.head()
```

```
##    ID    Q1   Q2    Q3
## 0   1   8.0  9.0  10.0
## 1   2   8.0  8.0   8.0
## 2   3  10.0  7.0  10.0
## 3   4   6.0  5.0   9.0
## 4   5  10.0  6.0   8.0
```

Sub-setting a data frame

- Select columns
  - using indexing
  - using dot notation (it is like the $ in R)

```
ID2=quiz.ID
print(type(ID2))
```

```
## <class 'pandas.core.series.Series'>
```

```
print(ID2)
```

```
## 0      1
## 1      2
## 2      3
## 3      4
## 4      5
## 5      6
## 6      7
## 7      8
## 8      9
## 9     10
## 10    11
## 11    12
```

Sub-setting a data frame

- Subsetting rows (or called `slicing`)

```
quiz[10:20]
```

```
##      ID    Q1    Q2    Q3    Q4    Q5    Q6
## 10   11  10.0   NaN   4.0   7.0   9.0   NaN
## 11   12   6.0   8.0   5.0   8.0  10.0   8.0
## 12   13  10.0  10.0   8.0  10.0  10.0   9.0
## 13   14  10.0  10.0  10.0  10.0   9.0   8.0
## 14   15  10.0  10.0  10.0  10.0  10.0  10.0
## 15   16  10.0   8.0   9.0   6.0   9.0   6.0
## 16   17  10.0   NaN   9.0   NaN   6.0   9.0
## 17   18  10.0  10.0  10.0   7.0   6.0   9.0
## 18   19  10.0   7.0   7.0   7.0  10.0   8.0
## 19   20  10.0  10.0  10.0  10.0  10.0   8.0
```

Sub-setting a data frame

- Subsetting rows (or called `slicing`)
- `pandas.DataFrame.iloc` can be used to subset both rows and columns using **integer-based** lookups
  - https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.iloc.html

```
import numpy as np
np.random.seed(1)
train = np.random.choice(40, 20, replace=False)
quiz.iloc[train, :]
```

```
##     ID    Q1    Q2    Q3    Q4    Q5    Q6
## 2    3  10.0   7.0  10.0  10.0  10.0   8.0
## 31  32  10.0  10.0  10.0  10.0   9.0   9.0
## 3    4   6.0   5.0   9.0   8.0   5.0   NaN
## 21  22  10.0  10.0  10.0   7.0  10.0   7.0
## 27  28   NaN   4.0   8.0   5.0  10.0   9.0
## 29  30  10.0   9.0  10.0   9.0   5.0   7.0
## 22  23  10.0  10.0   NaN   9.0  10.0   9.0
## 39  40  10.0  10.0  10.0   8.0  10.0   8.0
## 19  20  10.0  10.0  10.0  10.0  10.0   8.0
## 26  27  10.0  10.0  10.0  10.0  10.0  10.0
## 32  33  10.0   8.0   8.0   6.0   9.0   7.0
```

Sub-setting a data frame

- We can subeset rows using logic values

```
select = np.in1d(range(40), train)
quiz[select]
```

```
##      ID    Q1    Q2    Q3    Q4    Q5    Q6
## 2     3  10.0   7.0  10.0  10.0  10.0   8.0
## 3     4   6.0   5.0   9.0   8.0   5.0   NaN
## 4     5  10.0   6.0   8.0   6.0   NaN   NaN
## 10   11  10.0   NaN   4.0   7.0   9.0   NaN
## 14   15  10.0  10.0  10.0  10.0  10.0  10.0
## 17   18  10.0  10.0  10.0   7.0   6.0   9.0
## 19   20  10.0  10.0  10.0  10.0  10.0   8.0
## 21   22  10.0  10.0  10.0   7.0  10.0   7.0
## 22   23  10.0  10.0   NaN   9.0  10.0   9.0
## 26   27  10.0  10.0  10.0  10.0  10.0  10.0
## 27   28   NaN   4.0   8.0   5.0  10.0   9.0
## 28   29   6.0   9.0   4.0   5.0   9.0   0.0
## 29   30  10.0   9.0  10.0   9.0   5.0   7.0
## 30   31  10.0  10.0  10.0   7.0   6.0   NaN
## 31   32  10.0  10.0  10.0  10.0   9.0   9.0
## 32   33  10.0   8.0   8.0   6.0   9.0   7.0
```

Sub-setting a data frame

- Check if two dataframes are equal

```
quiz.iloc[train, :].equals(quiz[select])
```

```
## False
```

- The two data frames will be identical if we sort the index
  - https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.sort_index.html

```
quiz_train=quiz.iloc[train, :].sort_index(axis = 0)
quiz_train.equals(quiz[select])
```

```
## True
```

Sub-setting a data frame

```
quiz.iloc[range(10,15)]
```

```
##      ID    Q1    Q2    Q3    Q4    Q5    Q6
## 10   11  10.0   NaN   4.0   7.0   9.0   NaN
## 11   12   6.0   8.0   5.0   8.0  10.0   8.0
## 12   13  10.0  10.0   8.0  10.0  10.0   9.0
## 13   14  10.0  10.0  10.0  10.0   9.0   8.0
## 14   15  10.0  10.0  10.0  10.0  10.0  10.0
```

Sub-setting a data frame

- Subsetting discontinuous rows

```
quiz.iloc[ [*range(10,15),*range(20,25)],]
```

```
##      ID    Q1    Q2    Q3    Q4    Q5    Q6
## 10  11  10.0   NaN   4.0   7.0   9.0   NaN
## 11  12   6.0   8.0   5.0   8.0  10.0   8.0
## 12  13  10.0  10.0   8.0  10.0  10.0   9.0
## 13  14  10.0  10.0  10.0  10.0   9.0   8.0
## 14  15  10.0  10.0  10.0  10.0  10.0  10.0
## 20  21  10.0   9.0  10.0   8.5  10.0   5.0
## 21  22  10.0  10.0  10.0   7.0  10.0   7.0
## 22  23  10.0  10.0   NaN   9.0  10.0   9.0
## 23  24  10.0  10.0  10.0  10.0   9.0   9.0
## 24  25  10.0   9.0   8.0   6.0   9.0   8.0
```

Sub-setting a data frame

- Or equivalently using index only

```
print(quiz.iloc[0:8, [0,1,3]])
```

```
##     ID    Q1    Q3
## 0    1   8.0  10.0
## 1    2   8.0   8.0
## 2    3  10.0  10.0
## 3    4   6.0   9.0
## 4    5  10.0   8.0
## 5    6   NaN  10.0
## 6    7  10.0   9.0
## 7    8  10.0  10.0
```

Sub-setting a data frame

- We can subset both rows and columns

```
quiz[["ID","Q1","Q2","Q3"]][10:25]
#quiz_sub=quiz[["ID","Q1","Q2","Q3"]]
#quiz_sub[10:25]
```

```
##      ID    Q1    Q2    Q3
## 10   11  10.0   NaN   4.0
## 11   12   6.0   8.0   5.0
## 12   13  10.0  10.0   8.0
## 13   14  10.0  10.0  10.0
## 14   15  10.0  10.0  10.0
## 15   16  10.0   8.0   9.0
## 16   17  10.0   NaN   9.0
## 17   18  10.0  10.0  10.0
## 18   19  10.0   7.0   7.0
## 19   20  10.0  10.0  10.0
## 20   21  10.0   9.0  10.0
## 21   22  10.0  10.0  10.0
## 22   23  10.0  10.0   NaN
## 23   24  10.0  10.0  10.0
## 24   25  10.0   9.0   8.0
```

Sub-setting a data frame

- Pandas indexing operations provide us with a one-method way to select both the rows and the columns we want.
- `pandas.DataFrame.loc` can be used to subset both rows and columns using **label-based** lookups.
- Using both labels and integer index using `loc`

```
print(quiz.loc[0:8,["ID","Q1","Q3"]])
```

```
##    ID    Q1    Q3
## 0   1   8.0  10.0
## 1   2   8.0   8.0
## 2   3  10.0  10.0
## 3   4   6.0   9.0
## 4   5  10.0   8.0
## 5   6   NaN  10.0
## 6   7  10.0   9.0
## 7   8  10.0  10.0
## 8   9  10.0   6.0
```

Sub-setting a data frame

- To look up cell values, we use `at[]` and `iat[]`, which are faster.

```
print(quiz.at[0, 'Q3'])
```

```
## 10.0
```

```
print(quiz.iat[0, 3])
```

```
## 10.0
```

Sub-setting a data frame

- Filtering: we can subset rows based on the restrictions or **Boolean masks** on a column(s)

```
quiz[quiz.Q6>= 9]
```

```
##       ID    Q1    Q2    Q3    Q4    Q5    Q6
## 8      9  10.0  10.0   6.0   7.0   8.0  10.0
## 12    13  10.0  10.0   8.0  10.0  10.0   9.0
## 14    15  10.0  10.0  10.0  10.0  10.0  10.0
## 16    17  10.0   NaN   9.0   NaN   6.0   9.0
## 17    18  10.0  10.0  10.0   7.0   6.0   9.0
## 22    23  10.0  10.0   NaN   9.0  10.0   9.0
## 23    24  10.0  10.0  10.0  10.0   9.0   9.0
## 26    27  10.0  10.0  10.0  10.0  10.0  10.0
## 27    28   NaN   4.0   8.0   5.0  10.0   9.0
## 31    32  10.0  10.0  10.0  10.0   9.0   9.0
## 33    34  10.0  10.0   6.0   8.0   8.0   9.0
## 34    35  10.0  10.0   7.0   8.0  10.0  10.0
```

Sub-setting a data frame

- Filtering: we can subset rows based on the restrictions or **Boolean masks** on a column(s)

```
quiz.loc[quiz.Q6>= 9]
```

```
##       ID    Q1    Q2    Q3    Q4    Q5    Q6
## 8      9  10.0  10.0   6.0   7.0   8.0  10.0
## 12    13  10.0  10.0   8.0  10.0  10.0   9.0
## 14    15  10.0  10.0  10.0  10.0  10.0  10.0
## 16    17  10.0   NaN   9.0   NaN   6.0   9.0
## 17    18  10.0  10.0  10.0   7.0   6.0   9.0
## 22    23  10.0  10.0   NaN   9.0  10.0   9.0
## 23    24  10.0  10.0  10.0  10.0   9.0   9.0
## 26    27  10.0  10.0  10.0  10.0  10.0  10.0
## 27    28   NaN   4.0   8.0   5.0  10.0   9.0
## 31    32  10.0  10.0  10.0  10.0   9.0   9.0
## 33    34  10.0  10.0   6.0   8.0   8.0   9.0
## 34    35  10.0  10.0   7.0   8.0  10.0  10.0
```

Sub-setting a data frame

```
quiz.loc[(quiz.Q5>= 9) | (quiz.Q6>= 9)]
#quiz[(quiz.Q5>= 9) | (quiz.Q6>= 9)]
```

```
##      ID    Q1    Q2    Q3    Q4    Q5    Q6
## 0     1   8.0   9.0  10.0   9.5  10.0   8.0
## 1     2   8.0   8.0   8.0  10.0   9.0   8.0
## 2     3  10.0   7.0  10.0  10.0  10.0   8.0
## 5     6   NaN   9.0  10.0  10.0  10.0   NaN
## 6     7  10.0   5.0   9.0   8.0  10.0   7.0
## 7     8  10.0  10.0  10.0   9.0  10.0   7.0
## 8     9  10.0  10.0   6.0   7.0   8.0  10.0
## 9    10   8.0  10.0  10.0   4.0  10.0   7.0
## 10   11  10.0   NaN   4.0   7.0   9.0   NaN
## 11   12   6.0   8.0   5.0   8.0  10.0   8.0
## 12   13  10.0  10.0   8.0  10.0  10.0   9.0
## 13   14  10.0  10.0  10.0  10.0   9.0   8.0
## 14   15  10.0  10.0  10.0  10.0  10.0  10.0
## 15   16  10.0   8.0   9.0   6.0   9.0   6.0
## 16   17  10.0   NaN   9.0   NaN   6.0   9.0
## 17   18  10.0  10.0  10.0   7.0   6.0   9.0
## 18   19  10.0   7.0   7.0   7.0  10.0   8.0
## 19   20  10.0  10.0  10.0  10.0  10.0   8.0
```

Sub-setting a data frame

```
quiz.loc[(quiz.Q5>= 9) & (quiz.Q6>= 9)]
#quiz[(quiz.Q5>= 9) & (quiz.Q6>= 9)]
```

```
##      ID    Q1    Q2    Q3    Q4    Q5    Q6
## 12   13  10.0  10.0   8.0  10.0  10.0   9.0
## 14   15  10.0  10.0  10.0  10.0  10.0  10.0
## 22   23  10.0  10.0   NaN   9.0  10.0   9.0
## 23   24  10.0  10.0  10.0  10.0   9.0   9.0
## 26   27  10.0  10.0  10.0  10.0  10.0  10.0
## 27   28   NaN   4.0   8.0   5.0  10.0   9.0
## 31   32  10.0  10.0  10.0  10.0   9.0   9.0
## 34   35  10.0  10.0   7.0   8.0  10.0  10.0
```

Sub-setting a data frame

- **Important note**: When creating Boolean masks, we must use bitwise operators (`&`, `|`, `~`) instead of logical operators (`and`, `or`, `not`).

Sub-setting a data frame

```
quiz.loc[(quiz.Q5>= 9) & (quiz.Q6>= 9),['Q1','Q2','Q3','Q4']]
```

```
##        Q1    Q2    Q3    Q4
## 12  10.0  10.0   8.0  10.0
## 14  10.0  10.0  10.0  10.0
## 22  10.0  10.0   NaN   9.0
## 23  10.0  10.0  10.0  10.0
## 26  10.0  10.0  10.0  10.0
## 27   NaN   4.0   8.0   5.0
## 31  10.0  10.0  10.0  10.0
## 34  10.0  10.0   7.0   8.0
```

Sub-setting a data frame

- Two Boolean masks can be avoided using the `between()` method

```
quiz.loc[quiz.Q5. between(5,6)]
```

```
##     ID    Q1    Q2    Q3   Q4   Q5   Q6
## 3    4   6.0   5.0   9.0  8.0  5.0  NaN
## 16  17  10.0   NaN   9.0  NaN  6.0  9.0
## 17  18  10.0  10.0  10.0  7.0  6.0  9.0
## 29  30  10.0   9.0  10.0  9.0  5.0  7.0
## 30  31  10.0  10.0  10.0  7.0  6.0  NaN
## 35  36   8.0   7.0   5.0  8.0  5.0  0.0
```

Sub-setting a data frame

- The isin() method can be used to create a Boolean mask for values that match one of a list of values

```
earthquake= pd.read_csv("../data/earthquakes.csv", header=0)
earthquake1=earthquake.loc[earthquake.magType.isin(['mw', 'mwb']),
 ['alert', 'mag', 'magType', 'title', 'tsunami', 'type']]
earthquake1.head()
```

```
##        alert   mag  ... tsunami        type
## 995      NaN  3.35  ...       0  earthquake
## 1465   green  3.83  ...       0  earthquake
## 2414   green  3.83  ...       1  earthquake
## 4988   green  4.41  ...       1  earthquake
## 6307   green  5.80  ...       0  earthquake
##
## [5 rows x 6 columns]
```

Sub-setting a data frame

- We can use `idxmin()` and `idxmax()` for the indices of the minimum and maximum values of a variable for filtering.

```
earthquake.loc[[earthquake.mag.idxmin(), earthquake.mag.idxmax()],
['alert', 'mag', 'magType', 'title', 'tsunami', 'type']]
```

```
##       alert   mag  ... tsunami        type
## 2409   NaN -1.26  ...       0  earthquake
## 5263   red  7.50  ...       1  earthquake
##
## [2 rows x 6 columns]
```

License