Data Handling

The most commonly used pandas functions for reading tabular data are 'read_csv' and 'read_table'

In [1]:
```
# To view the data in small csv files we can use !type 'file name or file path'
```

In [2]:
```
!type dataset\ex1.csv
```

```
A, B, C, RAJ
1, 2, 3, RAM
S,D, F, GIRI
A1, B1, C1, GURU
A2, B2, C2, GUNA
A3, B3, C3, RAJA
```

In [3]:
```
# import Pandas and Numpy
import pandas as pd
import numpy as np
```

In [4]:
```
# Let's see the signature of 'read_csv'
pd.read_csv?
```

In [5]:
```
# By default the 'read_csv' will infer first row as the header if it is a tabular row
# for non tabular row we use 'skiprows' option of 'read_csv' method
data = pd.read_csv('dataset/ex1.csv')
data
```

Out[5]:

	A	B	C	RAJ
0	*1*	*2*	*3*	*RAM*
1	*S*	*D*	*F*	*GIRI*
2	*A1*	*B1*	*C1*	*GURU*
3	*A2*	*B2*	*C2*	*GUNA*
4	*A3*	*B3*	*C3*	*RAJA*

In [6]:
```
# Let's read the file which has no header by default
data_nh = pd.read_csv('dataset/ex2.csv', header=None)
data_nh
```

Out[6]:

	0	1	2	3
0	*1*	*2*	*3*	*PYTHON*
1	*4*	*5*	*6*	*JAVA*
2	*7*	*8*	*9*	*C*
3	*10*	*11*	*12*	*RUBY*
4	*13*	*14*	*15*	*R*

In [7]:
```
# Let's create or rename a new header for this file manually
data_nh = pd.read_csv('dataset/ex2.csv', header=None, names=['N1', 'N2', 'N3', 'Prog_name'])
data_nh
```

Out[7]:

	N1	N2	N3	Prog_name
0	1	2	3	PYTHON
1	4	5	6	JAVA
2	7	8	9	C
3	10	11	12	RUBY
4	13	14	15	R

In [8]:
```
# we can set any column as the index of the returned DataFrame passing column name in 'index_col' by default it is set to None
data_ic = pd.read_csv('dataset/ex2.csv', header=None, names=['N1', 'N2', 'N3', 'Prog_name'], index_col='Prog_name')
data_ic
```

Out[8]:

	N1	N2	N3
Prog_name			
PYTHON	1	2	3
JAVA	4	5	6
C	7	8	9
RUBY	10	11	12
R	13	14	15

In [9]:
```
# we can also pass a list of names as the header name and specify the index_col
names_index = ['N1', 'N2', 'N3', 'Prog_name']
data_lc = pd.read_csv('dataset/ex2.csv', header=None, names= names_index, index_col='Prog_name')
data_lc
```

Out[9]:

Prog_name	N1	N2	N3
PYTHON	1	2	3
JAVA	4	5	6
C	7	8	9
RUBY	10	11	12
R	13	14	15

In [10]:
```
# Let's see the file which has different types of columns
!type dataset\hindex.csv
```

```
cat,type,Student_Name,Sub1,Sub2,Sub3,Sub4,Sub5,Sub6
Med,a,Student1,25,10,10,15,15,16
Med,b,Student2,23,15,15,15,15,17
Med,c,Student3,21,11,12,16,15,19
Med,d,Student4,22,12,15,12,18,15
Med,e,Student5,21,15,16,16,17,18
Aveg,a,Student1,20,18,17,18,16,17
Aveg,b,Student2,20,19,18,13,17,19
Aveg,c,Student3,22,20,15,17,18,20
Aveg,d,Student4,21,22,14,18,16,23
Aveg,e,Student5,15,20,15,19,17,25
```

In [11]:
```
# To form a hierachical index from multiple columns pass a list of column names or numbers to 'index_col'
#data_hi = pd.read_csv('dataset/hindex.csv')
data_hi = pd.read_csv('dataset/hindex.csv', index_col=['cat', 'type'])
data_hi
```

Out[11]:

		Student_Name	Sub1	Sub2	Sub3	Sub4	Sub5	Sub6
cat	**type**							
Med	**a**	*Student1*	*25*	*10*	*10*	*15*	*15*	*16*
	b	*Student2*	*23*	*15*	*15*	*15*	*15*	*17*
	c	*Student3*	*21*	*11*	*12*	*16*	*15*	*19*
	d	*Student4*	*22*	*12*	*15*	*12*	*18*	*15*
	e	*Student5*	*21*	*15*	*16*	*16*	*17*	*18*
Aveg	**a**	*Student1*	*20*	*18*	*17*	*18*	*16*	*17*
	b	*Student2*	*20*	*19*	*18*	*13*	*17*	*19*
	c	*Student3*	*22*	*20*	*15*	*17*	*18*	*20*
	d	*Student4*	*21*	*22*	*14*	*18*	*16*	*23*
	e	*Student5*	*15*	*20*	*15*	*19*	*17*	*25*

In [12]:
```
# How to handle a text file with significant ammount of white space?
!type dataset\ex3_ws.txt
```

```
            A         B         C
aaa -1 -2.02 -0.61
bbb  0.82  0.40 -0.39
ccc -0.42 -0.58 -0.21
ddd -0.97 -0.64 1.10
```

In [13]:
```
# Let's see the signature of the 'read_table'
# it is similar to like 'read_csv' but with slight different options for its arguments: like 'sep=\t'
pd.read_table?
```

In [14]:
```
# in oreder handle this type of file we can pass regular expression as a delimeter i.e., 'sep=\s+' to 'read_table'
# the 'read_table' ifers by dafault that the first line would be the index of the DataFrame
data_ws = pd.read_table('dataset/ex3_ws.txt', sep='\s+')
data_ws
```

Out[14]:

	A	B	C
aaa	*-1.00*	*-2.02*	*-0.61*
bbb	*0.82*	*0.40*	*-0.39*
ccc	*-0.42*	*-0.58*	*-0.21*
ddd	*-0.97*	*-0.64*	*1.10*

In [15]:
```
# Let's see how to skip the rows in 'read_csv' method
!type dataset\skip_rows.csv
```

```
Students Marks list
cat, type, Student_Name, Sub1, Sub2, Sub3, Sub4, Sub5, Sub6
These are the cat1 type studenst marks list
cat1, a, Student1, 25, 10, 10, 15, 15, 16
cat1, b, Student2, 23, 15, 15, 15, 15, 17
cat1, c, Student3, 21, 11, 12, 16, 15, 19
cat1, d, Student4, 22, 12, 15, 12, 18, 15
cat1, e, Student5, 21, 15, 16, 16, 17, 18
These are the cat2 type studenst marks list
cat2, a, Student1, 20, 18, 17, 18, 16, 17
cat2, b, Student2, 20, 19, 18, 13, 17, 19
cat2, c, Student3, 22, 20, 15, 17, 18, 20
cat2, d, Student4, 21, 22, 14, 18, 16, 23
cat2, e, Student5, 15, 20, 15, 19, 17, 25
```

In [16]:
```
# we can use 'skiprows=' argument to skip the specific rows. To skip multiple rows just pass the list rows numbers to
 skip
# remember in Python index starts from '0'
data_sr = pd.read_csv('dataset/skip_rows.csv', skiprows=[0, 1, 2, 8])
data_sr
```

Out[16]:

	cat1	a	Student1	25	10	10.1	15	15.1	16
0	cat1	b	Student2	23	15	15	15	15	17
1	cat1	c	Student3	21	11	12	16	15	19
2	cat1	d	Student4	22	12	15	12	18	15
3	cat1	e	Student5	21	15	16	16	17	18
4	cat2	a	Student1	20	18	17	18	16	17
5	cat2	b	Student2	20	19	18	13	17	19
6	cat2	c	Student3	22	20	15	17	18	20
7	cat2	d	Student4	21	22	14	18	16	23
8	cat2	e	Student5	15	20	15	19	17	25

In [17]:
```
# How to handle missing values in Pandas DataFrame
# in Pandas the missing values are represented by the sentinels NA or NaN: we can use our own sentinels to missing dat
a
!type dataset\missing_file.csv
```

```
cat,type,Student_Name,Sub1,Sub2,Sub3,Sub4,Sub5,Sub6
Med,a,Student1,25,10,10,15,15,16
Med,b,Student2,23,15,,15,15,17
Med,c,Student3,21,,12,16,15,19
Med,d,Student4,22,,15,12,18,15
Med,e,Student5,21,15,16,16,17,
Aveg,a,Student1,20,,17,18,16,17
Aveg,b,Student2,20,19,,13,17,19
Aveg,c,Student3,22,20,,17,18,20
Aveg,d,Student4,21,22,,18,16,23
Aveg,e,Student5,15,20,15,19,17,
```

In [18]:
```
# By default the missing data values are represented by NaN
data_miss = pd.read_csv('dataset/missing_file.csv')
data_miss
```

Out[18]:

	cat	type	Student_Name	Sub1	Sub2	Sub3	Sub4	Sub5	Sub6
0	Med	a	Student1	25	10.0	10.0	15	15	16.0
1	Med	b	Student2	23	15.0	NaN	15	15	17.0
2	Med	c	Student3	21	NaN	12.0	16	15	19.0
3	Med	d	Student4	22	NaN	15.0	12	18	15.0
4	Med	e	Student5	21	15.0	16.0	16	17	NaN
5	Aveg	a	Student1	20	NaN	17.0	18	16	17.0
6	Aveg	b	Student2	20	19.0	NaN	13	17	19.0
7	Aveg	c	Student3	22	20.0	NaN	17	18	20.0
8	Aveg	d	Student4	21	22.0	NaN	18	16	23.0
9	Aveg	e	Student5	15	20.0	15.0	19	17	NaN

In [19]:
```
# 'isnull' object will results the boolean object with the True or False for the missing and non-missing values respectively
data_miss.isnull()
```

Out[19]:

	cat	type	Student_Name	Sub1	Sub2	Sub3	Sub4	Sub5	Sub6
0	False	False	False	False	False	False	False	False	False
1	False	False	False	False	False	True	False	False	False
2	False	False	False	False	True	False	False	False	False
3	False	False	False	False	True	False	False	False	False
4	False	False	False	False	False	False	False	False	True
5	False	False	False	False	True	False	False	False	False
6	False	False	False	False	False	True	False	False	False
7	False	False	False	False	False	True	False	False	False
8	False	False	False	False	False	True	False	False	False
9	False	False	False	False	False	False	False	False	True

In [20]:
```
# we can use our own sentinels for missing values with 'na_values' argument: like 'na_values=['NULL']'
data_missna = pd.read_csv('dataset/missing_file.csv', na_values=['NULL'])
data_missna
```

Out[20]:

	cat	type	Student_Name	Sub1	Sub2	Sub3	Sub4	Sub5	Sub6
0	Med	a	Student1	25	10.0	10.0	15	15	16.0
1	Med	b	Student2	23	15.0	NaN	15	15	17.0
2	Med	c	Student3	21	NaN	12.0	16	15	19.0
3	Med	d	Student4	22	NaN	15.0	12	18	15.0
4	Med	e	Student5	21	15.0	16.0	16	17	NaN
5	Aveg	a	Student1	20	NaN	17.0	18	16	17.0
6	Aveg	b	Student2	20	19.0	NaN	13	17	19.0
7	Aveg	c	Student3	22	20.0	NaN	17	18	20.0
8	Aveg	d	Student4	21	22.0	NaN	18	16	23.0
9	Aveg	e	Student5	15	20.0	15.0	19	17	NaN

In [21]:
```
# we can manually specify different sentinels for each column
sentinels = {'type':['a'], 'cat':['Med'], 'Sub1':[25, 20]}
data_miss_sent = pd.read_csv('dataset/missing_file.csv', na_values=sentinels)
data_miss_sent
```

Out[21]:

	cat	type	Student_Name	Sub1	Sub2	Sub3	Sub4	Sub5	Sub6
0	NaN	NaN	Student1	NaN	10.0	10.0	15	15	16.0
1	NaN	b	Student2	23.0	15.0	NaN	15	15	17.0
2	NaN	c	Student3	21.0	NaN	12.0	16	15	19.0
3	NaN	d	Student4	22.0	NaN	15.0	12	18	15.0
4	NaN	e	Student5	21.0	15.0	16.0	16	17	NaN
5	Aveg	NaN	Student1	NaN	NaN	17.0	18	16	17.0
6	Aveg	b	Student2	NaN	19.0	NaN	13	17	19.0
7	Aveg	c	Student3	22.0	20.0	NaN	17	18	20.0
8	Aveg	d	Student4	21.0	22.0	NaN	18	16	23.0
9	Aveg	e	Student5	15.0	20.0	15.0	19	17	NaN

How to Read Text Files in Pieces?

In [22]:
```
# 'read_csv' method provides 'chunksize=None' option to read large file in chunks with smaller size
# Let's see the "global_superstore" file
# Before that we need to set the display settings of pandas to read fewer lines
pd.options.display.max_rows = 5
```

In [23]:
```
large_file = pd.read_csv(r'dataset/global_superstore_2016.csv', header = None, encoding = 'latin1', low_memory=False)
large_file
```

Out[23]:

	0	1	2	3	4	5	6	7	8	9	...	14	15	16
0	Row ID	Order ID	Order Date	Ship Date	Ship Mode	Customer ID	Customer Name	Segment	Postal Code	City	...	Product ID	Category	Sub-Category
1	40098	CA-2014-AB10015140-41954	11/11/2014	11/13/2014	First Class	AB-100151402	Aaron Bergman	Consumer	73120	Oklahoma City	...	TEC-PH-5816	Technology	Phones
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
51289	9596	MX-2013-RB1979518-41322	2/17/2013	2/21/2013	Standard Class	RB-1979518	Ross Baird	Home Office	NaN	Valinhos	...	OFF-BI-2919	Office Supplies	Binders
51290	6147	MX-2013-MC1810093-41416	5/22/2013	5/26/2013	Second Class	MC-1810093	Mick Crebagga	Consumer	NaN	Tipitapa	...	OFF-PA-3990	Office Supplies	Paper

51291 rows × 24 columns

In [24]:
```
large_file1 = pd.read_csv(r'dataset/global_superstore_2016.csv', header = 0, encoding = 'latin1', low_memory=False, nrows=5)
large_file1
```

Out[24]:

	Row ID	Order ID	Order Date	Ship Date	Ship Mode	Customer ID	Customer Name	Segment	Postal Code	City	...	Product ID	Category	Sub-Category
0	40098	CA-2014-AB10015140-41954	11/11/2014	11/13/2014	First Class	AB-100151402	Aaron Bergman	Consumer	73120.0	Oklahoma City	...	TEC-PH-5816	Technology	Phones
1	26341	IN-2014-JR162107-41675	2/5/2014	2/7/2014	Second Class	JR-162107	Justin Ritter	Corporate	NaN	Wollongong	...	FUR-CH-5379	Furniture	Chairs
2	25330	IN-2014-CR127307-41929	10/17/2014	10/18/2014	First Class	CR-127307	Craig Reiter	Consumer	NaN	Brisbane	...	TEC-PH-5356	Technology	Phones
3	13524	ES-2014-KM1637548-41667	1/28/2014	1/30/2014	First Class	KM-1637548	Katherine Murray	Home Office	NaN	Berlin	...	TEC-PH-5267	Technology	Phones
4	47221	SG-2014-RH9495111-41948	11/5/2014	11/6/2014	Same Day	RH-9495111	Rick Hansen	Consumer	NaN	Dakar	...	TEC-CO-6011	Technology	Copiers

5 rows × 24 columns

In [25]:
```
large_file_chunker = pd.read_csv(r'dataset/global_superstore_2016.csv', header = None, encoding = 'latin1', low_memory=False, chunksize=1000)
large_file_chunker
# the pandas 'get_chunk' method that enables us to read pieces of an arbitrary size that can be seen later in this course
```

Out[25]:
```
<pandas.io.parsers.TextFileReader at 0x6d0abf0>
```

How to Write Data in Text Format?

In [26]:
```
# Pandas provides 'to_csv' to write the text data to comma delimeted format
```

In [27]:
```
# Let's see the different arguments of 'to_csv': Note that it is a pandas DataFrame's object
pd.DataFrame.to_csv?
```

In [28]:
```
# Let's import data file that has some missing values
import pandas as pd
data= pd.read_csv('dataset/missing_file.csv', nrows=5)
data
```

Out[28]:

	cat	type	Student_Name	Sub1	Sub2	Sub3	Sub4	Sub5	Sub6
0	Med	a	Student1	25	10.0	10.0	15	15	16.0
1	Med	b	Student2	23	15.0	NaN	15	15	17.0
2	Med	c	Student3	21	NaN	12.0	16	15	19.0
3	Med	d	Student4	22	NaN	15.0	12	18	15.0
4	Med	e	Student5	21	15.0	16.0	16	17	NaN

In [29]:
```
# We can export or write to a different file
data.to_csv('dataset/new_missing.csv')
```

In [30]:
```
# we can also export to excel file using 'to_excel' method
# you can check the signature of 'to_excel' similar to 'to_csv'
data.to_excel('dataset/new_missingexcel.xlsx', encoding= 'uft-8')
```

In [31]:
```
# we can use different delimeter to export the file: Ex.: sep='|'
data.to_csv('dataset/new_sep.csv', sep='|')
!type dataset\new_sep.csv
```

```
|cat|type|Student_Name|Sub1|Sub2|Sub3|Sub4|Sub5|Sub6
0|Med|a|Student1|25|10.0|10.0|15|15|16.0
1|Med|b|Student2|23|15.0||15|15|17.0
2|Med|c|Student3|21||12.0|16|15|19.0
3|Med|d|Student4|22||15.0|12|18|15.0
4|Med|e|Student5|21|15.0|16.0|16|17|
```

In [32]:
```
# missing values can be represented with different sentinels: Ex.: na_rep='NULL'
data.to_csv('dataset/new_missing_na.csv', na_rep='NULL')
!type dataset\new_missing_na.csv
```

```
,cat,type,Student_Name,Sub1,Sub2,Sub3,Sub4,Sub5,Sub6
0,Med,a,Student1,25,10.0,10.0,15,15,16.0
1,Med,b,Student2,23,15.0,NULL,15,15,17.0
2,Med,c,Student3,21,NULL,12.0,16,15,19.0
3,Med,d,Student4,22,NULL,15.0,12,18,15.0
4,Med,e,Student5,21,15.0,16.0,16,17,NULL
```

In [33]:
```
# we can also dissable both row and column labels with 'index=False' and 'header=False'
data.to_csv('dataset/new_missing_nhi.csv', index=False, header=False, na_rep='NULL')
!type dataset\new_missing_nhi.csv
```

```
Med,a,Student1,25,10.0,10.0,15,15,16.0
Med,b,Student2,23,15.0,NULL,15,15,17.0
Med,c,Student3,21,NULL,12.0,16,15,19.0
Med,d,Student4,22,NULL,15.0,12,18,15.0
Med,e,Student5,21,15.0,16.0,16,17,NULL
```

In [34]:
```
# we can also export only subset of columns as the new file
data.to_csv('dataset/new_missing_ssc.csv', index=False, columns=['cat', 'type', 'Student_Name'], na_rep='NULL')
!type dataset\new_missing_ssc.csv
```

```
cat,type,Student_Name
Med,a,Student1
Med,b,Student2
Med,c,Student3
Med,d,Student4
Med,e,Student5
```

In [35]:
```
# There are many ways to write or export the data, we will see later in this course.
```

How To Work With Delimeted Format?

In [36]:
```
# sometimes it is neccessary to work with different delimeted formats
# Let's see the small file with doubly quoted values
```

In [37]:
```
!type dataset\dqtext.csv
```

```
"A","B","C","D"
"1","2","3","4"
"a","b","c","d"
```

In [38]:
```
# Let's work with the Python's csv module
import csv
csv.reader??
```

In [39]:
```
# In order to work with this type of file Python's 'csv' module is a good opion.
import csv
f = open('dataset/dqtext.csv')
reader = csv.reader(f)
reader
```

Out[39]:
```
<_csv.reader at 0x6f0ac70>
```

In [40]:
```
# iterating over the file lines yields a tuples of values with quote charecters removed
for line in reader:
    print(line)
```

```
['A', 'B', 'C', 'D']
['1', '2', '3', '4']
['a', 'b', 'c', 'd']
```

In [41]:
```
# Let's do some data wranling
# First read the file as a list of lines
with open('dataset/dqtext.csv') as f:
    lines = list(csv.reader(f))

print(lines)
```

```
[['A', 'B', 'C', 'D'], ['1', '2', '3', '4'], ['a', 'b', 'c', 'd']]
```

In [42]:
```
#  Split the lines into header lines and the data lines
header, values = lines[0], lines[1:]
# Then we can create a dictionary of data columns using a dictionary comprehension and the expression zip(*values),
# which transposes rows to columns
data_dict = {i: v for i, v in zip(header, zip(*values))}
data_dict
```

Out[42]:
```
{'A': ('1', 'a'), 'B': ('2', 'b'), 'C': ('3', 'c'), 'D': ('4', 'd')}
```

In [43]:
```
# we can also split the data as header lines and data lines as a separate lists.
header, values = lines[0], lines[1:]
print(header, values, end='\n')
```

```
['A', 'B', 'C', 'D'] [['1', '2', '3', '4'], ['a', 'b', 'c', 'd']]
```

In [44]:
```
# We can also write our own data to a new csv file using 'csv.writer', a Python's csv module's method
# Let's see its docstring:
csv.writer?
```

In [45]:
```
# Let's write some data to a 'myfile' csv document line by line. we can also use a iterable sequences to write the fil
e
with open('dataset/myfile.csv', 'w') as f:
    writer = csv.writer(f)
    writer.writerow(('one', 'two', 'three'))
    writer.writerow(('1', '2', '3'))
    writer.writerow(('4', '5', '6'))
    writer.writerow(('7', '8', '9'))
```

In [46]:
```
# read the file
!type dataset\myfile.csv
```

```
one,two,three

1,2,3

4,5,6

7,8,9
```

In [ ]:

In [ ]:

In [ ]:

In [ ]: