Data Processing with Pandas

Changing working directory.

In [72]:

```
%cd D:\Data Science\GitHub\Python Learning\Python-for-Data-Science\Data Files\Python Le
arning
```

```
D:\Data Science\GitHub\Python Learning\Python-for-Data-Science\Data Files
\Python Learning
```

1. The Series Data Structure

Series is the one-dimensional data structure in *pandas*.

In [73]:

```
import pandas as pd
import numpy as np
```

In [74]:

```
animals = ['Tiger', 'Bear', 'Moose']
pd.Series(animals)
```

Out[74]:

```
0    Tiger
1     Bear
2    Moose
dtype: object
```

In [75]:

```
type(animals)
```

Out[75]:

```
list
```

- **Handling Missing Values**

In [76]:

```
animals = ['Tiger', 'Bear', None]
pd.Series(animals)
```

Out[76]:

```
0    Tiger
1     Bear
2     None
dtype: object
```

In [77]:

```
numbers = [1, 2, None]
pd.Series(numbers)
```

Out[77]:

```
0    1.0
1    2.0
2    NaN
dtype: float64
```

Here pandas series will turn the missing value to be **NaN**, which is different from **None**.
To check if a value is **Nan**,

In [78]:

```
np.isnan(pd.Series(numbers)[2])
```

Out[78]:

```
True
```

- **Creating Series from Dictionaries**

In [79]:

```
sports = {'Archery': 'Bhutan',
          'Golf': 'Scotland',
          'Sumo': 'Japan',
          'Taekwondo': 'South Korea'}
s = pd.Series(sports)
s
```

Out[79]:

```
Archery           Bhutan
Golf            Scotland
Sumo               Japan
Taekwondo    South Korea
dtype: object
```

- **Creating Series with Separate Index Creation**

In [80]:

```
s = pd.Series(['Tiger', 'Bear', 'Moose'], index=['India', 'America', 'Canada'])
s
```

Out[80]:

```
India      Tiger
America     Bear
Canada     Moose
dtype: object
```

If the number of indexes are not aligned with keys, *pandas* will assign NaN to those missing values.

In [81]:

```
sports = {'Archery': 'Bhutan',
          'Golf': 'Scotland',
          'Sumo': 'Japan',
          'Taekwondo': 'South Korea'}
s = pd.Series(sports, index=['Golf', 'Sumo', 'Hockey'])
s
```

Out[81]:

```
Golf      Scotland
Sumo         Japan
Hockey         NaN
dtype: object
```

1.1 Querying a Series

In [82]:

```
sports = {'Archery': 'Bhutan',
          'Golf': 'Scotland',
          'Sumo': 'Japan',
          'Taekwondo': 'South Korea'}
s = pd.Series(sports)
s
```

Out[82]:

```
Archery           Bhutan
Golf            Scotland
Sumo               Japan
Taekwondo    South Korea
dtype: object
```

- Using **iloc** and **loc** attribute to query series

In [83]:

```
s.iloc[3]
```

Out[83]:

```
'South Korea'
```

In [84]:

```
s.loc['Golf']
```

Out[84]:

```
'Scotland'
```

Be careful when using the indexing operator on the series itself. And the safer option is to be more explicit and use the iloc or loc attributes directly.

- **Appending New Values**

In [85]:

```
s = pd.Series([1, 2, 3])
s.loc['Animal'] = 'Bears'
s
```

Out[85]:

```
0             1
1             2
2             3
Animal    Bears
dtype: object
```

- **Non-unique Index**

In [86]:

```
original_sports = pd.Series({'Archery': 'Bhutan',
                             'Golf': 'Scotland',
                             'Sumo': 'Japan',
                             'Taekwondo': 'South Korea'})
cricket_loving_countries = pd.Series(['Australia',
                                      'Barbados',
                                      'Pakistan',
                                      'England'],
                                     index=['Cricket',
                                            'Cricket',
                                            'Cricket',
                                            'Cricket'])
all_countries = original_sports.append(cricket_loving_countries)
```

In [87]:

```
print(all_countries)
print(original_sports)
```

```
Archery           Bhutan
Golf            Scotland
Sumo               Japan
Taekwondo    South Korea
Cricket        Australia
Cricket         Barbados
Cricket         Pakistan
Cricket          England
dtype: object
Archery           Bhutan
Golf            Scotland
Sumo               Japan
Taekwondo    South Korea
dtype: object
```

Note that even though we applied *.append* method to *original_sports*, the original series was not changed.

2. DataFrame Data Structure

Series is the two-dimensional data structure in *pandas*.

2.1 Query from DataFrame

In [88]:

```
import pandas as pd

purchase_1 = pd.Series({'Name': 'Chris',
                        'Item Purchased': 'Dog Food',
                        'Cost': 22.50})

purchase_2 = pd.Series({'Name': 'Kevyn',
                        'Item Purchased': 'Kitty Litter',
                        'Cost': 2.50})

purchase_3 = pd.Series({'Name': 'Vinod',
                        'Item Purchased': 'Bird Seed',
                        'Cost': 5.00})

# Define the DataFrame here:
df = pd.DataFrame([purchase_1, purchase_2, purchase_3], index = ['Store 1', 'Store 2',
'Store 3'])
df.head()
```

Out[88]:

	Name	Item Purchased	Cost
Store 1	Chris	Dog Food	22.5
Store 2	Kevyn	Kitty Litter	2.5
Store 3	Vinod	Bird Seed	5.0

- Query element from DataFrame using **loc**.

In [89]:

```
df.loc['Store 3']
```

Out[89]:

```
Name                  Vinod
Item Purchased    Bird Seed
Cost                      5
Name: Store 3, dtype: object
```

- Index can be **non-unique**

In [90]:

```
df2 = pd.DataFrame([purchase_1, purchase_2, purchase_3], index = ['Store 1', 'Store 1',
'Store 3'])
df2.loc['Store 1']
```

Out[90]:

	Name	Item Purchased	Cost
Store 1	Chris	Dog Food	22.5
Store 1	Kevyn	Kitty Litter	2.5

- Selecting columns

In [91]:

```
df.loc[:, ['Name', 'Cost']]
```

Out[91]:

	Name	Cost
Store 1	Chris	22.5
Store 2	Kevyn	2.5
Store 3	Vinod	5.0

- **NOTE:** Avoid Query by Chaining Selecting
  This will creating a copy of dataset, which will be quit inefficient.

In [92]:

```
df.loc['Store 1']['Cost']
```

Out[92]:

```
22.5
```

2.2 Dropping & Adding Data from DataFrame

- Dropping Rows

In [93]:

```
df.drop('Store 1')
```

Out[93]:

	Name	Item Purchased	Cost
Store 2	Kevyn	Kitty Litter	2.5
Store 3	Vinod	Bird Seed	5.0

NOTE: The original dataset is untouched.

In [94]:

```
df
```

Out[94]:

	Name	Item Purchased	Cost
Store 1	Chris	Dog Food	22.5
Store 2	Kevyn	Kitty Litter	2.5
Store 3	Vinod	Bird Seed	5.0

- Dropping Columns

In [95]:

```
copy_df = df.copy()
del copy_df['Name']
copy_df
```

Out[95]:

	Item Purchased	Cost
Store 1	Dog Food	22.5
Store 2	Kitty Litter	2.5
Store 3	Bird Seed	5.0

- Adding Columns

In [96]:

```
df['Location'] = None
df
```

Out[96]:

	Name	Item Purchased	Cost	Location
Store 1	Chris	Dog Food	22.5	None
Store 2	Kevyn	Kitty Litter	2.5	None
Store 3	Vinod	Bird Seed	5.0	None

3. DataFrame Indexing and Loading

NOTE: If you want to explicitly use a **copy**, then you should consider calling the **copy method** on the DataFrame for it first.

3.1 Creating DataFrame from .csv files.

In [97]:

```
%cd D:\Data Science\GitHub\Python Learning\Python-for-Data-Science\Data Files\Python Le
arning

df = pd.read_csv('expense_report.csv', skiprows = 0)
df.head()
```

```
D:\Data Science\GitHub\Python Learning\Python-for-Data-Science\Data Files
\Python Learning
```

Out[97]:

	TRXN_MONTH	TRXN_DT	TRXN_AMT	MERCHT_NAME	CATG
0	Mar-19	01/03/2019	61.20	VIRGIN ACTIVE	Entertainment
1	Mar-19	04/03/2019	1.31	LOON FUNG LIMITED	Glossary
2	Mar-19	04/03/2019	16.89	LOON FUNG LIMITED	Glossary
3	Mar-19	04/03/2019	20.00	LNK CAPITAL ONE	Other
4	Mar-19	05/03/2019	2.85	SAINSBURY'S SPRMRKTS LT NOTTINGHAM	Glossary

- Show Names of All Columns in the DataFrame.

In [98]:

```
df.columns
```

Out[98]:

```
Index(['TRXN_MONTH', 'TRXN_DT', 'TRXN_AMT', 'MERCHT_NAME', 'CATG'], dtype
='object')
```

4. Querying a DataFrame

4.1 Querying DataFrame by Boolean Masking

- **Step 1.** creating Boolean masking array.

In [99]:

```
df['TRXN_AMT'] > 5
```

Out[99]:

```
0        True
1       False
2        True
3        True
4       False
5        True
6        True
7        True
8        True
9        True
10       True
11      False
12       True
13       True
14       True
15       True
16       True
17       True
18       True
19       True
20       True
21      False
22       True
23       True
24      False
25       True
26       True
27      False
28      False
29      False
        ...
1177     True
1178    False
1179    False
1180    False
1181     True
1182    False
1183    False
1184    False
1185     True
1186    False
1187     True
1188    False
1189     True
1190     True
1191     True
1192     True
1193     True
1194     True
1195     True
1196     True
1197     True
1198     True
1199     True
1200     True
1201     True
1202     True
1203     True
1204     True
```

```
1205     True
1206     True
Name: TRXN_AMT, Length: 1207, dtype: bool
```

- **Step 2.** overlay the masking on the DataFrame by **where** function.

In [100]:

```
df_500 = df.where(df['TRXN_AMT'] > 5)
df_500.head()
```

Out[100]:

	TRXN_MONTH	TRXN_DT	TRXN_AMT	MERCHT_NAME	CATG
0	Mar-19	01/03/2019	61.20	VIRGIN ACTIVE	Entertainment
1	NaN	NaN	NaN	NaN	NaN
2	Mar-19	04/03/2019	16.89	LOON FUNG LIMITED	Glossary
3	Mar-19	04/03/2019	20.00	LNK CAPITAL ONE	Other
4	NaN	NaN	NaN	NaN	NaN

NOTE: rows that do not match the condition will give NaN values.
To drop those **NaN** rows.

In [101]:

```
df_500 = df_500.dropna()
df_500.head()
```

Out[101]:

	TRXN_MONTH	TRXN_DT	TRXN_AMT	MERCHT_NAME	CATG
0	Mar-19	01/03/2019	61.20	VIRGIN ACTIVE	Entertainment
2	Mar-19	04/03/2019	16.89	LOON FUNG LIMITED	Glossary
3	Mar-19	04/03/2019	20.00	LNK CAPITAL ONE	Other
5	Mar-19	06/03/2019	6.49	TESCO STORE 5660 5660TE WELWYN GARDEN C	Glossary
6	Mar-19	07/03/2019	35.40	SHANGHAI SHANGHAI SHANG NOTTINGHAM	Eating Out

- **However**, there is a quicker way to do that.

In [102]:

```
df_500 = df[df['TRXN_AMT'] > 5]
df_500.head()
```

Out[102]:

	TRXN_MONTH	TRXN_DT	TRXN_AMT	MERCHT_NAME	CATG
0	Mar-19	01/03/2019	61.20	VIRGIN ACTIVE	Entertainment
2	Mar-19	04/03/2019	16.89	LOON FUNG LIMITED	Glossary
3	Mar-19	04/03/2019	20.00	LNK CAPITAL ONE	Other
5	Mar-19	06/03/2019	6.49	TESCO STORE 5660 5660TE WELWYN GARDEN C	Glossary
6	Mar-19	07/03/2019	35.40	SHANGHAI SHANGHAI SHANG NOTTINGHAM	Eating Out

5. Indexing DataFrame

5.1 Setting Index of DataFrame

- Re-indexing DataFrame

In [103]:

```
df = df.set_index('CATG')
df.head()
```

Out[103]:

	TRXN_MONTH	TRXN_DT	TRXN_AMT	MERCHT_NAME
CATG				
Entertainment	Mar-19	01/03/2019	61.20	VIRGIN ACTIVE
Glossary	Mar-19	04/03/2019	1.31	LOON FUNG LIMITED
Glossary	Mar-19	04/03/2019	16.89	LOON FUNG LIMITED
Other	Mar-19	04/03/2019	20.00	LNK CAPITAL ONE
Glossary	Mar-19	05/03/2019	2.85	SAINSBURY'S SPRMRKTS LT NOTTINGHAM

- Reset Index by Number Series

In [104]:

```
df = df.reset_index()
df.head()
```

Out[104]:

	CATG	TRXN_MONTH	TRXN_DT	TRXN_AMT	MERCHT_NAME
0	Entertainment	Mar-19	01/03/2019	61.20	VIRGIN ACTIVE
1	Glossary	Mar-19	04/03/2019	1.31	LOON FUNG LIMITED
2	Glossary	Mar-19	04/03/2019	16.89	LOON FUNG LIMITED
3	Other	Mar-19	04/03/2019	20.00	LNK CAPITAL ONE
4	Glossary	Mar-19	05/03/2019	2.85	SAINSBURY'S SPRMRKTS LT NOTTINGHAM

5.2 Multi-level Index in DataFrame

- Changing the DataFrame to be Multi-level Indexed

In [105]:

```
df['TRXN_MONTH'].unique()
df = df[df['TRXN_MONTH'] == 'Oct-19']
df.head()
```

Out[105]:

	CATG	TRXN_MONTH	TRXN_DT	TRXN_AMT	MERCHT_NAME
807	Transport	Oct-19	01/10/2019	5.30	TFL TRAVEL CHARGE TFL.GOV.UK/CP
808	Transport	Oct-19	01/10/2019	2.90	TFL TRAVEL CHARGE TFL.GOV.UK/CP
809	Glossary	Oct-19	01/10/2019	2.50	WAITROSE CONVENIENCE NOTTINGHAM
810	Eating Out	Oct-19	01/10/2019	6.55	STARBUCKS TESCO LONDON
811	Transport	Oct-19	01/10/2019	5.70	TFL TRAVEL CHARGE TFL.GOV.UK/CP

In [106]:

```
df = df.set_index(['TRXN_DT', 'CATG'])
df.head()
```

Out[106]:

		TRXN_MONTH	TRXN_AMT	MERCHT_NAME
TRXN_DT	**CATG**			
01/10/2019	**Transport**	Oct-19	5.30	TFL TRAVEL CHARGE TFL.GOV.UK/CP
	Transport	Oct-19	2.90	TFL TRAVEL CHARGE TFL.GOV.UK/CP
	Glossary	Oct-19	2.50	WAITROSE CONVENIENCE NOTTINGHAM
	Eating Out	Oct-19	6.55	STARBUCKS TESCO LONDON
	Transport	Oct-19	5.70	TFL TRAVEL CHARGE TFL.GOV.UK/CP

- Querying the data

In [107]:

```
df.loc[[('01/10/2019', 'Transport'), ('02/10/2019', 'Glossary')]]
```

Out[107]:

		TRXN_MONTH	TRXN_AMT	MERCHT_NAME
TRXN_DT	**CATG**			
01/10/2019	**Transport**	Oct-19	5.30	TFL TRAVEL CHARGE TFL.GOV.UK/CP
	Transport	Oct-19	2.90	TFL TRAVEL CHARGE TFL.GOV.UK/CP
	Transport	Oct-19	5.70	TFL TRAVEL CHARGE TFL.GOV.UK/CP
	Transport	Oct-19	4.40	TFL TRAVEL CHARGE TFL.GOV.UK/CP
	Transport	Oct-19	5.30	TFL TRAVEL CHARGE TFL.GOV.UK/CP
	Transport	Oct-19	5.80	TFL TRAVEL CHARGE TFL.GOV.UK/CP
	Transport	Oct-19	5.70	TFL TRAVEL CHARGE TFL.GOV.UK/CP
02/10/2019	**Glossary**	Oct-19	3.91	CAPITAL ONE

6. Missing Values

In [111]:

```
df.fillna
df.head()
```

Out[111]:

TRXN_DT	CATG	TRXN_MONTH	TRXN_AMT	MERCHT_NAME
01/10/2019	**Transport**	Oct-19	5.30	TFL TRAVEL CHARGE TFL.GOV.UK/CP
	Transport	Oct-19	2.90	TFL TRAVEL CHARGE TFL.GOV.UK/CP
	Glossary	Oct-19	2.50	WAITROSE CONVENIENCE NOTTINGHAM
	Eating Out	Oct-19	6.55	STARBUCKS TESCO LONDON
	Transport	Oct-19	5.70	TFL TRAVEL CHARGE TFL.GOV.UK/CP