Data Manipulation with Pandas

Pandas is the most widely used library of python for data science. It is incredibly helpful in manipulating the data so that you can derive better insights and build great machine learning models.

In this notebook, we will have a look at some of the intermediate concepts of working with pandas.

Table of Contents

Loading dataset

In this notebook we will use the Big Mart Sales Data. You can download the data from : https://datahack.analyticsvidhya.com/contest/practice-problem-big-mart-sales-iii/download/train-file

In [1]:
```
import pandas as pd
import numpy as np

# read the dataset
data_BM = pd.read_csv('bigmart_data.csv')
# drop the null values
data_BM = data_BM.dropna(how="any")
# view the top results
data_BM.head()
```

Out[1]:

	Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_Identifier	Outlet_Establishment_Year	Outlet_Size	Outlet_Locatior
0	FDA15	9.300	Low Fat	0.016047	Dairy	249.8092	OUT049	1999	Medium	
1	DRC01	5.920	Regular	0.019278	Soft Drinks	48.2692	OUT018	2009	Medium	
2	FDN15	17.500	Low Fat	0.016760	Meat	141.6180	OUT049	1999	Medium	
4	NCD19	8.930	Low Fat	0.000000	Household	53.8614	OUT013	1987	High	
5	FDP36	10.395	Regular	0.000000	Baking Goods	51.4008	OUT018	2009	Medium	

1. Sorting dataframes

Pandas data frame has two useful functions

- **sort_values()**: to sort pandas data frame by one or more columns
- **sort_index()**: to sort pandas data frame by row index

Each of these functions come with numerous options, like sorting the data frame in specific order (ascending or descending), sorting in place, sorting with missing values, sorting by specific algorithm etc.

Suppose you want to sort the dataframe by "Outlet_Establishment_Year" then you will use **sort_values**

In [ ]:
```
# sort by year
sorted_data = data_BM.sort_values(by='Outlet_Establishment_Year')
# print sorted data
sorted_data[:5]
```

- Now `sort_values` takes multiple options like:
  - `ascending` : The default sorting order is ascending, when you pass False here then it sorts in descending order.
  - `inplace` : whether to do inplace sorting or not

In [ ]:
```
# sort in place and descending order
data_BM.sort_values(by='Outlet_Establishment_Year', ascending=False, inplace=True)
data_BM[:5]
```

You might want to sort a data frame based on the values of multiple columns. We can specify the columns we want to sort by as a list in the argument for sort_values().

In [ ]:
```
# read the dataset
data_BM = pd.read_csv('bigmart_data.csv')
# drop the null values
data_BM = data_BM.dropna(how="any")

# sort by multiple columns
data_BM.sort_values(by=['Outlet_Establishment_Year', 'Item_Outlet_Sales'], ascending=False)[:5]
```

- Note that when sorting by multiple columns, pandas sort_value() uses the first variable first and second variable next.
- We can see the difference by switching the order of column names in the list.

In [ ]:
```
# changed the order of columns
data_BM.sort_values(by=['Item_Outlet_Sales', 'Outlet_Establishment_Year'], ascending=False, inplace=True)
data_BM[:5]
```

- We can use **sort_index()** to sort pandas dataframe to sort by row index or names.
- In this example, row index are numbers and in the earlier example we sorted data frame by 'Item_Outlet_Sales', 'Outlet_Establishment_Year' and

therefore the row index are jumbled up.
- We can sort by row index (with inplace=True option) and retrieve the original dataframe.

In [ ]:
```
# sort by index
data_BM.sort_index(inplace=True)
data_BM[:5]
```

2. Merging dataframes

- Joining and merging DataFrames is the core process to start with data analysis and machine learning tasks.
- It is one of the toolkits which every Data Analyst or Data Scientist should master because in almost all the cases data comes from multiple source and files.
- Pandas has two useful functions for merging dataframes:
  - **concat()**
  - **merge()**

Creating dummy data

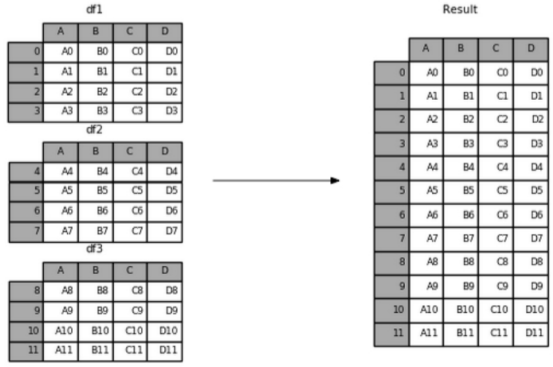
In [2]:
```
# create dummy data
df1 = pd.DataFrame({'A': ['A0', 'A1', 'A2', 'A3'],
                    'B': ['B0', 'B1', 'B2', 'B3'],
                    'C': ['C0', 'C1', 'C2', 'C3'],
                    'D': ['D0', 'D1', 'D2', 'D3']},
                   index=[0, 1, 2, 3])


df2 = pd.DataFrame({'A': ['A4', 'A5', 'A6', 'A7'],
                    'B': ['B4', 'B5', 'B6', 'B7'],
                    'C': ['C4', 'C5', 'C6', 'C7'],
                    'D': ['D4', 'D5', 'D6', 'D7']},
                   index=[4, 5, 6, 7])


df3 = pd.DataFrame({'A': ['A8', 'A9', 'A10', 'A11'],
                    'B': ['B8', 'B9', 'B10', 'B11'],
                    'C': ['C8', 'C9', 'C10', 'C11'],
                    'D': ['D8', 'D9', 'D10', 'D11']},
                   index=[8, 9, 10, 11])
```

a. concat() for combining dataframes

- Suppose you have the following three dataframes: df1, df2 and df3 and you want to combine them **"row-wise"** so that they become a single dataframe like the given image:



- You can use **concat()** here. You will have to pass the names of the DataFrames in a list as the argument to the concat().

In [3]:
```
# combine dataframes
result = pd.concat([df1, df2, df3])
result
```

Out[3]:

	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3
4	A4	B4	C4	D4
5	A5	B5	C5	D5
6	A6	B6	C6	D6
7	A7	B7	C7	D7
8	A8	B8	C8	D8
9	A9	B9	C9	D9
10	A10	B10	C10	D10
11	A11	B11	C11	D11

- pandas also provides you with an option to label the DataFrames, after the concatenation, with a key so that you may know which data came from which DataFrame.
- You can achieve the same by passing additional argument **keys** specifying the label names of the DataFrames in a list.

In [4]:
```
# combine dataframes
result = pd.concat([df1, df2, df3], keys=['x', 'y', 'z'])
result
```

Out[4]:

		A	B	C	D

x	0	A0	B0	C0	D0
	1	A1	B1	C1	D1
	2	A2	B2	C2	D2
	3	A3	B3	C3	D3
y	4	A4	B4	C4	D4
	5	A5	B5	C5	D5
	6	A6	B6	C6	D6
	7	A7	B7	C7	D7
z	8	A8	B8	C8	D8
	9	A9	B9	C9	D9
	10	A10	B10	C10	D10
	11	A11	B11	C11	D11

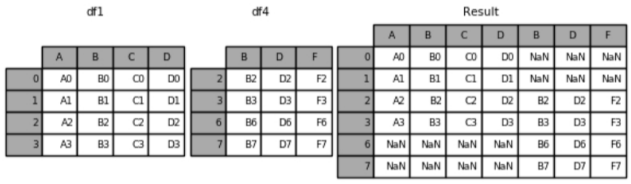
- Mentioning the keys also makes it easy to retrieve data corresponding to a particular DataFrame.
- You can retrieve the data of DataFrame df2 which had the label `y` by using the `loc` method.

In [5]:
```
# get second dataframe
result.loc['y']
```

Out[5]:

	A	B	C	D
4	A4	B4	C4	D4
5	A5	B5	C5	D5
6	A6	B6	C6	D6
7	A7	B7	C7	D7

- When gluing together multiple DataFrames, you have a choice of how to handle the other axes (other than the one being concatenated). This can be done in the following three ways:
  - Take the union of them all, `join='outer'`. This is the default option as it results in zero information loss.
  - Take the intersection, `join='inner'`.
  - Use a specific index, as passed to the `join_axes` argument.
- Here is an example of each of these methods. First, the default `join='outer'` behavior:



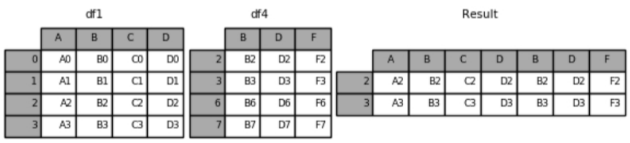
In [6]:
```
df4 = pd.DataFrame({'B': ['B2', 'B3', 'B6', 'B7'],
                    'D': ['D2', 'D3', 'D6', 'D7'],
                    'F': ['F2', 'F3', 'F6', 'F7']},
                   index=[2, 3, 6, 7])


result = pd.concat([df1, df4], axis=1, sort=False)
result
```

Out[6]:

	A	B	C	D	B	D	F
0	A0	B0	C0	D0	NaN	NaN	NaN
1	A1	B1	C1	D1	NaN	NaN	NaN
2	A2	B2	C2	D2	B2	D2	F2
3	A3	B3	C3	D3	B3	D3	F3
6	NaN	NaN	NaN	NaN	B6	D6	F6
7	NaN	NaN	NaN	NaN	B7	D7	F7

- Here is the same thing with `join='inner'`:

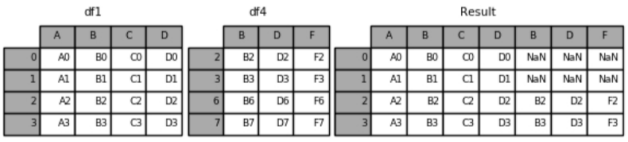


In [7]:
```
result = pd.concat([df1, df4], axis=1, join='inner')
result
```

Out[7]:

	A	B	C	D	B	D	F
2	A2	B2	C2	D2	B2	D2	F2
3	A3	B3	C3	D3	B3	D3	F3

- Lastly, suppose we just wanted to `reuse the exact index` from the original DataFrame:



In [8]:
```
result = pd.concat([df1, df4], axis=1, join_axes=[df1.index])
result
```

Out[8]:

	A	B	C	D	B	D	F
0	A0	B0	C0	D0	NaN	NaN	NaN
1	A1	B1	C1	D1	NaN	NaN	NaN
2	A2	B2	C2	D2	B2	D2	F2
3	A3	B3	C3	D3	B3	D3	F3

b. merge() for combining dataframes using SQL like joins

- Another ubiquitous operation related to DataFrames is the merging operation.
- Two DataFrames might hold different kinds of information about the same entity and linked by some common feature/column.
- We can use **merge()** to combine such dataframes in pandas.

Creating dummy data

In [9]:
```
# create dummy data
df_a = pd.DataFrame({
        'subject_id': ['1', '2', '3', '4', '5'],
        'first_name': ['Alex', 'Amy', 'Allen', 'Alice', 'Ayoung'],
        'last_name': ['Anderson', 'Ackerman', 'Ali', 'Aoni', 'Atiches']})

df_b = pd.DataFrame({
        'subject_id': ['4', '5', '6', '7', '8'],
        'first_name': ['Billy', 'Brian', 'Bran', 'Bryce', 'Betty'],
        'last_name': ['Bonder', 'Black', 'Balwner', 'Brice', 'Btisan']})

df_c = pd.DataFrame({
        'subject_id': ['1', '2', '3', '4', '5', '7', '8', '9', '10', '11'],
        'test_id': [51, 15, 15, 61, 16, 14, 15, 1, 61, 16]})
```

Now these are our dataframes:

	subject_id	first_name	last_name
0	1	Alex	Anderson
1	2	Amy	Ackerman
2	3	Allen	Ali
3	4	Alice	Aoni
4	5	Ayoung	Atiches

df_a

	subject_id	first_name	last_name
0	4	Billy	Bonder
1	5	Brian	Black
2	6	Bran	Balwner
3	7	Bryce	Brice
4	8	Betty	Btisan

df_b

	subject_id	test_id
0	1	51
1	2	15
2	3	15
3	4	61
4	5	16
5	7	14
6	8	15
7	9	1
8	10	61
9	11	16

df_c

- Let's start with a basic join, we want to combine `df_a` with `df_c` based on the `subject_id` column.

In [10]:
```
pd.merge(df_a, df_c, on='subject_id')
```

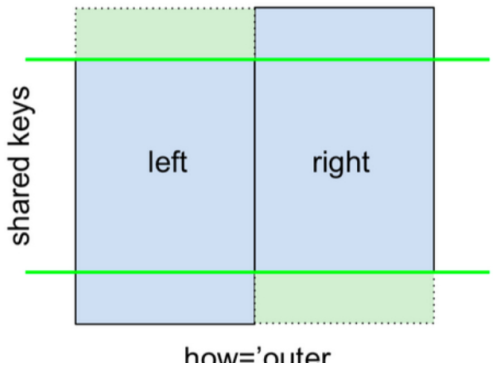
Out[10]:

	subject_id	first_name	last_name	test_id
0	1	Alex	Anderson	51
1	2	Amy	Ackerman	15
2	3	Allen	Ali	15
3	4	Alice	Aoni	61
4	5	Ayoung	Atiches	16

- Now that we have done a basic join, let's get into **some commmon SQL joins.**

Merge with outer join

- "Full outer join produces the set of all records in Table A and Table B, with matching records from both sides where available. If there is no match, the missing side will contain null."



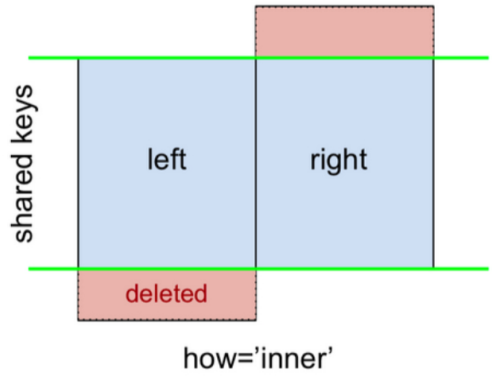
```
In [11]: 1 pd.merge(df_a, df_b, on='subject_id', how='outer')
```

Out[11]:

	subject_id	first_name_x	last_name_x	first_name_y	last_name_y
0	1	Alex	Anderson	NaN	NaN
1	2	Amy	Ackerman	NaN	NaN
2	3	Allen	Ali	NaN	NaN
3	4	Alice	Aoni	Billy	Bonder
4	5	Ayoung	Atiches	Brian	Black
5	6	NaN	NaN	Bran	Balwner
6	7	NaN	NaN	Bryce	Brice
7	8	NaN	NaN	Betty	Btisan

Merge with inner join

- "Inner join produces only the set of records that match in both Table A and Table B."



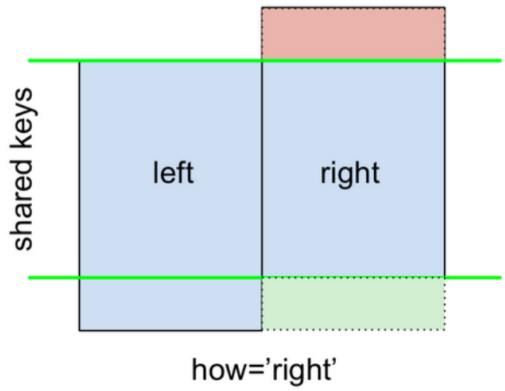

```
In [12]: 1 pd.merge(df_a, df_b, on='subject_id', how='inner')
```

Out[12]:

	subject_id	first_name_x	last_name_x	first_name_y	last_name_y
0	4	Alice	Aoni	Billy	Bonder
1	5	Ayoung	Atiches	Brian	Black

Merge with right join

- "Right outer join produces a complete set of records from Table B, with the matching records (where available) in Table A. If there is no match, the left side will contain null."



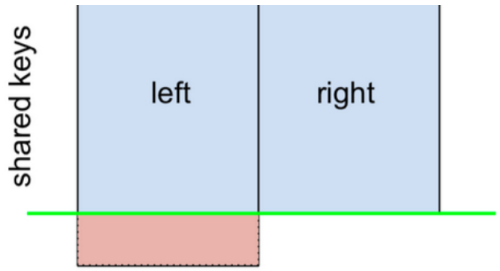

```
In [13]: 1 pd.merge(df_a, df_b, on='subject_id', how='right')
```

Out[13]:

	subject_id	first_name_x	last_name_x	first_name_y	last_name_y
0	4	Alice	Aoni	Billy	Bonder
1	5	Ayoung	Atiches	Brian	Black
2	6	NaN	NaN	Bran	Balwner
3	7	NaN	NaN	Bryce	Brice
4	8	NaN	NaN	Betty	Btisan

Merge with left join

- "Left outer join produces a complete set of records from Table A, with the matching records (where available) in Table B. If there is no match, the right side will contain null."



how='left'

In [14]:
```
1 pd.merge(df_a, df_b, on='subject_id', how='left')
```

Out[14]:

	subject_id	first_name_x	last_name_x	first_name_y	last_name_y
0	1	Alex	Anderson	NaN	NaN
1	2	Amy	Ackerman	NaN	NaN
2	3	Allen	Ali	NaN	NaN
3	4	Alice	Aoni	Billy	Bonder
4	5	Ayoung	Atiches	Brian	Black

Merge OR Concat : Which to use when?

1. After learning both of the functions in detail, chances are that you might be confused which to use when.
2. One major difference is that `merge()` is used to combine dataframes on the basis of values of **common columns**. While `concat()` is used to **append dataframes** one below the other (or sideways, depending on whether the axis option is set to 0 or 1).
3. Exact usage depends upon the kind of data you have and analysis you want to perform.