

Lab 3: Data Preparation

CPE232 Data Models

[1] Reviews on Pandas

1.1) Discover

- methods to explore and understand your DataFrame

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

df = pd.read_csv('nss15.csv')
```

```
In [2]: # see the shape of the dataframe
print(df.shape)
```

```
(334839, 12)
```

```
In [3]: # seeing the summary of the dataframe
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 334839 entries, 0 to 334838
Data columns (total 12 columns):
 #   Column                Non-Null Count  Dtype
---  -
 0   caseNumber            334839 non-null  int64
 1   treatmentDate         334839 non-null  object
 2   statWeight            334839 non-null  float64
 3   stratum               334839 non-null  object
 4   age                  334839 non-null  int64
 5   sex                  334837 non-null  object
 6   race                 205014 non-null  object
 7   diagnosis             334839 non-null  int64
 8   bodyPart              334839 non-null  int64
 9   disposition           334839 non-null  int64
10   location              334839 non-null  int64
11   product              334839 non-null  int64
dtypes: float64(1), int64(7), object(4)
memory usage: 30.7+ MB
None
```

```
In [4]: # seeing the stats of the column in dataframe
print(df.describe())
```

	caseNumber	statWeight	age	diagnosis \
count	3.348390e+05	334839.000000	334839.000000	334839.000000
mean	1.510271e+08	39.343028	31.385451	60.154591
std	1.720330e+06	34.142933	26.105098	6.170699
min	1.501032e+08	4.965500	0.000000	41.000000
25%	1.504405e+08	15.059100	10.000000	57.000000
50%	1.507358e+08	15.776200	23.000000	59.000000
75%	1.510231e+08	74.881300	51.000000	64.000000
max	1.603418e+08	97.923900	107.000000	74.000000

	bodyPart	disposition	location	product
count	334839.000000	334839.000000	334839.000000	334839.000000
mean	64.374192	1.307930	2.485451	2098.900854
std	24.002331	0.977627	3.217617	1332.222670
min	0.000000	1.000000	0.000000	106.000000
25%	35.000000	1.000000	0.000000	1211.000000
50%	75.000000	1.000000	1.000000	1807.000000
75%	82.000000	1.000000	5.000000	3265.000000
max	94.000000	9.000000	9.000000	5555.000000

In [5]: `# seeing the first 5 rows of the dataframe`
`print(df.head())`

	caseNumber	treatmentDate	statWeight	stratum	age	sex	race \
0	150733174	7/11/2015	15.7762	V	5	Male	NaN
1	150734723	7/6/2015	83.2157	S	36	Male	White
2	150817487	8/2/2015	74.8813	L	20	Female	NaN
3	150717776	6/26/2015	15.7762	V	61	Male	NaN
4	150721694	7/4/2015	74.8813	L	88	Female	Other

	diagnosis	bodyPart	disposition	location	product
0	57	33	1	9	1267
1	57	34	1	1	1439
2	71	94	1	0	3274
3	71	35	1	0	611
4	62	75	1	0	1893

In [6]: `# seeing the last 5 rows of the dataframe`
`print(df.tail())`

	caseNumber	treatmentDate	statWeight	stratum	age	sex	race \
334834	150739278	5/31/2015	15.0591	V	7	Male	NaN
334835	150733393	7/11/2015	5.6748	C	3	Female	Black
334836	150819286	7/24/2015	15.7762	V	38	Male	NaN
334837	150823002	8/8/2015	97.9239	M	38	Female	White
334838	150723074	6/20/2015	49.2646	M	5	Female	White

	diagnosis	bodyPart	disposition	location	product
334834	59	76	1	1	1864
334835	68	85	1	0	1931
334836	71	79	1	0	3250
334837	59	82	1	1	464
334838	57	34	1	9	3273

In [7]: `# seeing the list of columns in the dataframe`
`print(df.columns)`

```
Index(['caseNumber', 'treatmentDate', 'statWeight', 'stratum', 'age', 'sex',
      'race', 'diagnosis', 'bodyPart', 'disposition', 'location', 'product'],
      dtype='object')
```

1.2) Selecting variables

- select specific columns from the DataFrame to create a new DataFrame with only those columns

In [8]: `df['age']`

Out[8]:

0	5
1	36
2	20
3	61
4	88
	..
334834	7
334835	3
334836	38
334837	38
334838	5

Name: age, Length: 334839, dtype: int64

In [9]: `df['age'].head()`

Out[9]:

0	5
1	36
2	20
3	61
4	88

Name: age, dtype: int64

In [10]: `df[['caseNumber', 'age']]`

Out[10]:

	caseNumber	age
0	150733174	5
1	150734723	36
2	150817487	20
3	150717776	61
4	150721694	88
...
334834	150739278	7
334835	150733393	3
334836	150819286	38
334837	150823002	38
334838	150723074	5

334839 rows × 2 columns

In [11]: `# select columns based on the data type`
`df.select_dtypes(include=['number'])`

Out[11]:

	caseNumber	statWeight	age	diagnosis	bodyPart	disposition	location	proc
0	150733174	15.7762	5	57	33	1	9	1
1	150734723	83.2157	36	57	34	1	1	1
2	150817487	74.8813	20	71	94	1	0	3
3	150717776	15.7762	61	71	35	1	0	
4	150721694	74.8813	88	62	75	1	0	1
...	
334834	150739278	15.0591	7	59	76	1	1	1
334835	150733393	5.6748	3	68	85	1	0	1
334836	150819286	15.7762	38	71	79	1	0	3
334837	150823002	97.9239	38	59	82	1	1	
334838	150723074	49.2646	5	57	34	1	9	3

334839 rows × 8 columns



In [12]: `# select row by .loc`
`df.loc[0]`

Out[12]:

caseNumber	150733174
treatmentDate	7/11/2015
statWeight	15.7762
stratum	V
age	5
sex	Male
race	NaN
diagnosis	57
bodyPart	33
disposition	1
location	9
product	1267

Name: 0, dtype: object

In [13]: `# select column by .loc`
`df.loc[:,6,'treatmentDate':'diagnosis']`

Out[13]:

	treatmentDate	statWeight	stratum	age	sex	race	diagnosis
0	7/11/2015	15.7762	V	5	Male	NaN	57
1	7/6/2015	83.2157	S	36	Male	White	57
2	8/2/2015	74.8813	L	20	Female	NaN	71
3	6/26/2015	15.7762	V	61	Male	NaN	71
4	7/4/2015	74.8813	L	88	Female	Other	62
5	7/2/2015	5.6748	C	1	Female	White	71
6	6/8/2015	15.7762	V	25	Male	Black	51

In [14]: `df.loc[df['age']>80, ['treatmentDate', 'age']]`

Out[14]:

	treatmentDate	age
4	7/4/2015	88
8	7/16/2015	98
39	5/3/2015	88
46	4/15/2015	91
63	1/12/2015	97
...
334701	4/27/2015	86
334784	7/7/2015	82
334785	7/11/2015	86
334815	10/28/2015	85
334819	1/13/2015	85

20422 rows × 2 columns

In [15]: `# select row by .iloc
df.iloc[0:5]`

Out[15]:

	caseNumber	treatmentDate	statWeight	stratum	age	sex	race	diagnosis	br
0	150733174	7/11/2015	15.7762	V	5	Male	NaN	57	
1	150734723	7/6/2015	83.2157	S	36	Male	White	57	
2	150817487	8/2/2015	74.8813	L	20	Female	NaN	71	
3	150717776	6/26/2015	15.7762	V	61	Male	NaN	71	
4	150721694	7/4/2015	74.8813	L	88	Female	Other	62	



```
In [16]: # select column by .iloc
df.iloc[:,[0,1,2,3,4]]
```

```
Out[16]:
```

	caseNumber	treatmentDate	statWeight	stratum	age
0	150733174	7/11/2015	15.7762	V	5
1	150734723	7/6/2015	83.2157	S	36
2	150817487	8/2/2015	74.8813	L	20
3	150717776	6/26/2015	15.7762	V	61
4	150721694	7/4/2015	74.8813	L	88
...
334834	150739278	5/31/2015	15.0591	V	7
334835	150733393	7/11/2015	5.6748	C	3
334836	150819286	7/24/2015	15.7762	V	38
334837	150823002	8/8/2015	97.9239	M	38
334838	150723074	6/20/2015	49.2646	M	5

334839 rows × 5 columns

1.3) Filtering the data

```
In [17]: # filter rows based on the condition
df[df['age'] > 50]
```

```
Out[17]:
```

	caseNumber	treatmentDate	statWeight	stratum	age	sex	race	diagno
3	150717776	6/26/2015	15.7762	V	61	Male	NaN	
4	150721694	7/4/2015	74.8813	L	88	Female	Other	
7	150704114	6/14/2015	83.2157	S	53	Male	White	
8	150736558	7/16/2015	83.2157	S	98	Male	Black	
16	150901411	8/27/2015	83.2157	S	65	Female	White	
...	
334811	150702215	6/27/2015	15.7762	V	51	Female	NaN	
334815	151100368	10/28/2015	83.2157	S	85	Female	NaN	
334819	150528367	1/13/2015	49.2646	M	85	Female	NaN	
334826	150648619	6/17/2015	15.7762	V	52	Female	White	
334829	150633526	4/4/2015	49.2646	M	51	Female	NaN	

85235 rows × 12 columns



```
In [18]: # filter coloum based on column name  
df.filter(like='age')
```

```
Out[18]:
```

	age
0	5
1	36
2	20
3	61
4	88
...	...
334834	7
334835	3
334836	38
334837	38
334838	5

334839 rows × 1 columns

1.4) Sorting

- Sort the DataFrame by its index based on column

```
In [19]: # sort the dataframe based on column name and ascending order  
df.sort_values(by='statWeight', ascending=False)
```

Out[19]:

	caseNumber	treatmentDate	statWeight	stratum	age	sex	race	diagno
67072	150533084	5/15/2015	97.9239	M	89	Male	NaN	
313846	150521217	4/18/2015	97.9239	M	36	Female	NaN	
230135	150857760	8/25/2015	97.9239	M	14	Male	White	
141323	151039262	10/11/2015	97.9239	M	39	Female	White	
230141	150662453	6/5/2015	97.9239	M	11	Female	White	
...	
122009	151146792	11/15/2015	4.9655	C	2	Female	White	
211090	151253201	12/15/2015	4.9655	C	2	Male	White	
317625	160106638	12/25/2015	4.9655	C	1	Male	White	
33679	151256307	12/20/2015	4.9655	C	9	Female	Black	
229596	160148171	12/4/2015	4.9655	C	16	Female	Other	

334839 rows × 12 columns



In [20]:

```
# sort the index of the dataframe
df.sort_index()
```

Out[20]:

	caseNumber	treatmentDate	statWeight	stratum	age	sex	race	diagno
0	150733174	7/11/2015	15.7762	V	5	Male	NaN	
1	150734723	7/6/2015	83.2157	S	36	Male	White	
2	150817487	8/2/2015	74.8813	L	20	Female	NaN	
3	150717776	6/26/2015	15.7762	V	61	Male	NaN	
4	150721694	7/4/2015	74.8813	L	88	Female	Other	
...	
334834	150739278	5/31/2015	15.0591	V	7	Male	NaN	
334835	150733393	7/11/2015	5.6748	C	3	Female	Black	
334836	150819286	7/24/2015	15.7762	V	38	Male	NaN	
334837	150823002	8/8/2015	97.9239	M	38	Female	White	
334838	150723074	6/20/2015	49.2646	M	5	Female	White	

334839 rows × 12 columns



1.5) Add/Remove

- This section shows how to manipulate the DataFrame's structure

In [20]: `# Dropping the column`
`df.drop(columns=['disposition'])`

Out[20]:

	caseNumber	treatmentDate	statWeight	stratum	age	sex	race	diagno
0	150733174	7/11/2015	15.7762	V	5	Male	NaN	
1	150734723	7/6/2015	83.2157	S	36	Male	White	
2	150817487	8/2/2015	74.8813	L	20	Female	NaN	
3	150717776	6/26/2015	15.7762	V	61	Male	NaN	
4	150721694	7/4/2015	74.8813	L	88	Female	Other	
...
334834	150739278	5/31/2015	15.0591	V	7	Male	NaN	
334835	150733393	7/11/2015	5.6748	C	3	Female	Black	
334836	150819286	7/24/2015	15.7762	V	38	Male	NaN	
334837	150823002	8/8/2015	97.9239	M	38	Female	White	
334838	150723074	6/20/2015	49.2646	M	5	Female	White	

334839 rows × 11 columns



In [22]: `# Adding column and create into a new column`
`df.assign(new_column=df['diagnosis'] + df['bodyPart'])`

Out[22]:

	caseNumber	treatmentDate	statWeight	stratum	age	sex	race	diagno
0	150733174	7/11/2015	15.7762	V	5	Male	NaN	
1	150734723	7/6/2015	83.2157	S	36	Male	White	
2	150817487	8/2/2015	74.8813	L	20	Female	NaN	
3	150717776	6/26/2015	15.7762	V	61	Male	NaN	
4	150721694	7/4/2015	74.8813	L	88	Female	Other	
...
334834	150739278	5/31/2015	15.0591	V	7	Male	NaN	
334835	150733393	7/11/2015	5.6748	C	3	Female	Black	
334836	150819286	7/24/2015	15.7762	V	38	Male	NaN	
334837	150823002	8/8/2015	97.9239	M	38	Female	White	
334838	150723074	6/20/2015	49.2646	M	5	Female	White	

334839 rows × 13 columns



In [21]: *# Removing the column and assigning it to a new variable*
`ages = df.pop('age')`

1.6) Clean missing

- to remove rows with missing values or replace missing values with a specified value

In [22]: *# replacing the missing values with a specified value*
`df.fillna(value=0)`

Out[22]:

	caseNumber	treatmentDate	statWeight	stratum	sex	race	diagnosis	b
0	150733174	7/11/2015	15.7762	V	Male	0	57	
1	150734723	7/6/2015	83.2157	S	Male	White	57	
2	150817487	8/2/2015	74.8813	L	Female	0	71	
3	150717776	6/26/2015	15.7762	V	Male	0	71	
4	150721694	7/4/2015	74.8813	L	Female	Other	62	
...	
334834	150739278	5/31/2015	15.0591	V	Male	0	59	
334835	150733393	7/11/2015	5.6748	C	Female	Black	68	
334836	150819286	7/24/2015	15.7762	V	Male	0	71	
334837	150823002	8/8/2015	97.9239	M	Female	White	59	
334838	150723074	6/20/2015	49.2646	M	Female	White	57	

334839 rows × 11 columns



In [23]:

```
# Remove the rows with missing values
df.dropna()
```

Out[23]:

	caseNumber	treatmentDate	statWeight	stratum	sex	race	diagnosis	b
1	150734723	7/6/2015	83.2157	S	Male	White	57	
4	150721694	7/4/2015	74.8813	L	Female	Other	62	
5	150721815	7/2/2015	5.6748	C	Female	White	71	
6	150713483	6/8/2015	15.7762	V	Male	Black	51	
7	150704114	6/14/2015	83.2157	S	Male	White	57	
...	
334830	150628863	6/8/2015	15.7762	V	Female	White	64	
334831	150607637	5/22/2015	5.6748	C	Female	Black	59	
334835	150733393	7/11/2015	5.6748	C	Female	Black	68	
334837	150823002	8/8/2015	97.9239	M	Female	White	59	
334838	150723074	6/20/2015	49.2646	M	Female	White	57	

205014 rows × 11 columns



[2] Data Cleaning and Preparation

.isnull, .dropna, .fillna

2.1) checking

```
In [24]: df.columns
```

```
Out[24]: Index(['caseNumber', 'treatmentDate', 'statWeight', 'stratum', 'sex', 'race',  
              'diagnosis', 'bodyPart', 'disposition', 'location', 'product'],  
              dtype='object')
```

```
In [25]: # isnull checking  
df.isnull().sum()
```

```
Out[25]: caseNumber          0  
         treatmentDate      0  
         statWeight         0  
         stratum            0  
         sex                2  
         race              129825  
         diagnosis          0  
         bodyPart           0  
         disposition        0  
         location           0  
         product            0  
         dtype: int64
```

```
In [26]: # percentage of missing values for the race  
df.race.isnull().sum()/df.shape[0]*100
```

```
Out[26]: 38.772365226272925
```

```
In [27]: df.shape[0]
```

```
Out[27]: 334839
```


2.2) Drop column

```
In [28]: # remove column by using  
df = df.drop(columns=['race'])
```

```
In [29]: df.head()
```

Out[29]:

	caseNumber	treatmentDate	statWeight	stratum	sex	diagnosis	bodyPart	dis
0	150733174	7/11/2015	15.7762	V	Male	57	33	
1	150734723	7/6/2015	83.2157	S	Male	57	34	
2	150817487	8/2/2015	74.8813	L	Female	71	94	
3	150717776	6/26/2015	15.7762	V	Male	71	35	
4	150721694	7/4/2015	74.8813	L	Female	62	75	



2.3) Data imputation

```
In [30]: # fillna
df['age'] = df['age'].fillna(df['age'].median())
```

```

-----
KeyError                                Traceback (most recent call last)
File ~\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.11_qbz5n2kfra8p0
\LocalCache\local-packages\Python311\site-packages\pandas\core\indexes\base.py:38
05, in Index.get_loc(self, key)
    3804 try:
-> 3805     return self._engine.get_loc(casted_key)
    3806 except KeyError as err:

File index.pyx:167, in pandas._libs.index.IndexEngine.get_loc()

File index.pyx:196, in pandas._libs.index.IndexEngine.get_loc()

File pandas\_libs\hashtable_class_helper.pxi:7081, in pandas._libs.hashtable.Py
ObjectHashTable.get_item()

File pandas\_libs\hashtable_class_helper.pxi:7089, in pandas._libs.hashtable.Py
ObjectHashTable.get_item()

```

KeyError: 'age'

The above exception was the direct cause of the following exception:

```

KeyError                                Traceback (most recent call last)
Cell In[30], line 2
      1 # fillna
----> 2 df['age'] = df['age'].fillna(df['age'].median())

File ~\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.11_qbz5n2kfra8p0
\LocalCache\local-packages\Python311\site-packages\pandas\core\frame.py:4102, in
DataFrame.__getitem__(self, key)
    4100 if self.columns.nlevels > 1:
    4101     return self._getitem_multilevel(key)
-> 4102 indexer = self.columns.get_loc(key)
    4103 if is_integer(indexer):
    4104     indexer = [indexer]

File ~\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.11_qbz5n2kfra8p0
\LocalCache\local-packages\Python311\site-packages\pandas\core\indexes\base.py:38
12, in Index.get_loc(self, key)
    3807 if isinstance(casted_key, slice) or (
    3808     isinstance(casted_key, abc.Iterable)
    3809     and any(isinstance(x, slice) for x in casted_key)
    3810 ):
    3811     raise InvalidIndexError(key)
-> 3812     raise KeyError(key) from err
    3813 except TypeError:
    3814     # If we have a listlike key, _check_indexing_error will raise
    3815     # InvalidIndexError. Otherwise we fall through and re-raise
    3816     # the TypeError.
    3817     self._check_indexing_error(key)

```

KeyError: 'age'

[Q1] From the above cell, Why it showing an error?

Ans: ก่อนหน้านี้ใช้ ages = df.pop('age') ทำให้ตอนนี้ไม่มี column age แล้ว ดังนั้นเรียกใช้ column age ไม่ได้

[Q2] Fix the error from Q1 problem.

```
In [31]: # [Q2]

# hint: see the cell that run `df.pop()`
# Ans :
df["age"] = ages

# fillna again
df['age'] = df['age'].fillna(df['age'].median())

df.head()
```

```
Out[31]:
```

	caseNumber	treatmentDate	statWeight	stratum	sex	diagnosis	bodyPart	dis
0	150733174	7/11/2015	15.7762	V	Male	57	33	
1	150734723	7/6/2015	83.2157	S	Male	57	34	
2	150817487	8/2/2015	74.8813	L	Female	71	94	
3	150717776	6/26/2015	15.7762	V	Male	71	35	
4	150721694	7/4/2015	74.8813	L	Female	62	75	

2.4) Drop row that have missing value

```
In [32]: # remove column by using .dropna()
df = df.dropna()
```

```
In [33]: df.isnull().sum()
```

```
Out[33]: caseNumber      0
treatmentDate    0
statWeight       0
stratum          0
sex              0
diagnosis        0
bodyPart         0
disposition      0
location         0
product          0
age              0
dtype: int64
```

Datetime

2.5) Working with the datetime format

```
In [34]: df["treatmentDate"] = pd.to_datetime(df["treatmentDate"], format="%m/%d/%Y")
```

C:\Users\punch\AppData\Local\Temp\ipykernel_15332\3208943844.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

```
df["treatmentDate"] = pd.to_datetime(df["treatmentDate"], format="%m/%d/%Y")
```

In [35]: `df.info()`

```
<class 'pandas.core.frame.DataFrame'>
Index: 334837 entries, 0 to 334838
Data columns (total 11 columns):
#   Column          Non-Null Count  Dtype
---  -
0   caseNumber      334837 non-null  int64
1   treatmentDate   334837 non-null  datetime64[ns]
2   statWeight      334837 non-null  float64
3   stratum         334837 non-null  object
4   sex             334837 non-null  object
5   diagnosis       334837 non-null  int64
6   bodyPart        334837 non-null  int64
7   disposition     334837 non-null  int64
8   location        334837 non-null  int64
9   product         334837 non-null  int64
10  age             334837 non-null  int64
dtypes: datetime64[ns](1), float64(1), int64(7), object(2)
memory usage: 30.7+ MB
```

In [36]: `df['Year'] = df['treatmentDate'].dt.year`

C:\Users\punch\AppData\Local\Temp\ipykernel_15332\1686165144.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

```
df['Year'] = df['treatmentDate'].dt.year
```

In [37]: `df['Month'] = df['treatmentDate'].dt.month`

C:\Users\punch\AppData\Local\Temp\ipykernel_15332\404848564.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

```
df['Month'] = df['treatmentDate'].dt.month
```

In [38]: `df.head()`

Out[38]:

	caseNumber	treatmentDate	statWeight	stratum	sex	diagnosis	bodyPart	dis
0	150733174	2015-07-11	15.7762	V	Male	57	33	
1	150734723	2015-07-06	83.2157	S	Male	57	34	
2	150817487	2015-08-02	74.8813	L	Female	71	94	
3	150717776	2015-06-26	15.7762	V	Male	71	35	
4	150721694	2015-07-04	74.8813	L	Female	62	75	

[Q3] Can you change the format to DD/MM/YYYY? Show your work.

In [39]: *# write your code here*

```
df["treatmentDate"] = df["treatmentDate"].dt.strftime("%d/%m/%Y")
df.head()
```

C:\Users\punch\AppData\Local\Temp\ipykernel_15332\366907535.py:2: SettingWithCopyWarning:
Warning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
df["treatmentDate"] = df["treatmentDate"].dt.strftime("%d/%m/%Y")

Out[39]:

	caseNumber	treatmentDate	statWeight	stratum	sex	diagnosis	bodyPart	dis
0	150733174	11/07/2015	15.7762	V	Male	57	33	
1	150734723	06/07/2015	83.2157	S	Male	57	34	
2	150817487	02/08/2015	74.8813	L	Female	71	94	
3	150717776	26/06/2015	15.7762	V	Male	71	35	
4	150721694	04/07/2015	74.8813	L	Female	62	75	

Combine Dataframe by .merge and .concat

2.6 Merge

In [40]: **import** pandas **as** pd

```
superstore_order = pd.read_csv('superstore/superstore_order.csv')
superstore_people = pd.read_csv('superstore/superstore_people.csv')
superstore_return = pd.read_csv('superstore/superstore_return.csv')
```

In [41]:

```
superstore_order.merge(superstore_return[superstore_return["Returned"]=="Yes"],
                        on="Order ID",
                        how="inner")\
[["Customer ID", "Returned"]]\
.drop_duplicates()
```

Out[41]:

	Customer ID	Returned
0	ZD-21925	Yes
3	TB-21055	Yes
10	JS-15685	Yes
13	LC-16885	Yes
20	BS-11755	Yes
...
688	ED-13885	Yes
689	TS-21205	Yes
696	MF-17665	Yes
702	SH-19975	Yes
705	RB-19435	Yes

222 rows × 2 columns

[Q4] What does the argument `how="inner"` do?Ans: `how="inner"` จะรวมเฉพาะข้อมูลที่มีค่าตรงกันในคอลัมน์ที่ใช้เชื่อมจากทั้งสอง DataFrame เท่านั้น

[Q5] In your opinion, what information that the result above conveys?

Ans: ได้ข้อมูลการคืนสินค้าจาก Customer ID ที่คืนแล้วโดยไม่มีข้อมูลซ้ำ

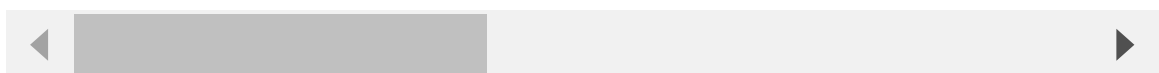
More merging...

```
In [42]: superstore_order.merge(superstore_return,
    on="Order ID" ,
    how="inner")
```

Out[42]:

	Row ID	Order ID	Order Date	Ship Date	Ship Mode	Customer ID	Customer Name	Segment
0	19	CA-2014-143336	27/08/2014	01/09/2014	Second Class	ZD-21925	Zuschuss Donatelli	Consumer
1	20	CA-2014-143336	27/08/2014	01/09/2014	Second Class	ZD-21925	Zuschuss Donatelli	Consumer
2	21	CA-2014-143336	27/08/2014	01/09/2014	Second Class	ZD-21925	Zuschuss Donatelli	Consumer
3	56	CA-2016-111682	17/06/2016	18/06/2016	First Class	TB-21055	Ted Butterfield	Consumer
4	57	CA-2016-111682	17/06/2016	18/06/2016	First Class	TB-21055	Ted Butterfield	Consumer
...
702	8870	CA-2017-101805	01/12/2017	06/12/2017	Standard Class	SH-19975	Sally Hughsby	Corporate
703	8871	CA-2017-101805	01/12/2017	06/12/2017	Standard Class	SH-19975	Sally Hughsby	Corporate
704	8872	CA-2017-101805	01/12/2017	06/12/2017	Standard Class	SH-19975	Sally Hughsby	Corporate
705	8873	US-2014-105137	10/10/2014	10/10/2014	Same Day	RB-19435	Richard Bierner	Consumer
706	8874	US-2014-105137	10/10/2014	10/10/2014	Same Day	RB-19435	Richard Bierner	Consumer

707 rows × 22 columns



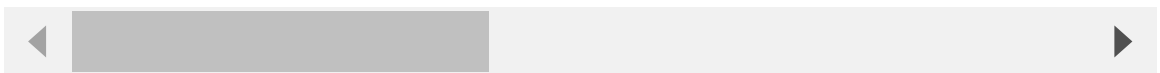
2.7) Concatenate

```
In [43]: pd.concat([superstore_order, superstore_people], axis=1, join='inner')
```

```
Out[43]:
```

	Row ID	Order ID	Order Date	Ship Date	Ship Mode	Customer ID	Customer Name	Segment	Col
0	1	CA-2016-152156	08/11/2016	11/11/2016	Second Class	CG-12520	Claire Gute	Consumer	U
1	2	CA-2016-152156	08/11/2016	11/11/2016	Second Class	CG-12520	Claire Gute	Consumer	U
2	3	CA-2016-138688	12/06/2016	16/06/2016	Second Class	DV-13045	Darrin Van Huff	Corporate	U
3	4	US-2015-108966	11/10/2015	18/10/2015	Standard Class	SO-20335	Sean ODonnell	Consumer	U

4 rows × 23 columns



[Q6] What is the difference between `inner` and `outer` on parameter `join` in `pd.concat` ?

ANS: `inner` : จะเก็บเฉพาะคอลัมน์หรือแถวที่มีอยู่ในทั้ง DataFrame ที่ทำการเชื่อมต่อ ถ้า DataFrame ใดขาดคอลัมน์หรือแถวที่ต้องการ จะไม่ถูกนำมารวมในผลลัพธ์ `outer` : จะเก็บทุกคอลัมน์หรือแถวจาก DataFrame ทั้งหมด และจะเติมค่า NaN ให้ในกรณีที่คอลัมน์หรือแถวใดขาดหายไป ทำการรวมคอลัมน์หรือแถวที่มีจากทุก DataFrame

Groupby

```
In [44]: superstore_order.groupby(['Segment', 'Ship Mode'])[['Sales', 'Quantity', 'Discount']]
```

Out[44]:

		Sales	Quantity	Discount	Profit
Segment	Ship Mode				
Consumer	First Class	138594.9328	2455	110.29	18953.7264
	Same Day	53660.6340	1001	43.85	8555.7193
	Second Class	203605.6822	3489	127.29	24701.9148
	Standard Class	627061.3262	10430	443.05	68864.9892
Corporate	First Class	97720.1209	1670	73.07	12660.2526
	Same Day	41716.5550	366	14.50	1120.9222
	Second Class	130759.9288	2027	71.47	15582.1762
	Standard Class	359359.2109	6203	262.82	49832.6780
Home Office	First Class	76743.8674	924	39.82	11829.8821
	Same Day	20968.5170	343	12.50	3909.3442
	Second Class	77175.1080	1148	37.80	12785.8953
	Standard Class	218325.9795	3595	142.14	27298.5786

[Q7] Describe an information that the result above conveys?

Ans: แสดงข้อมูลการผลรวมของ Sales, Quantity, Discount, Profit ตาม Segment และ Ship Mode

In [45]: `superstore_order["Profit Ratio"] = superstore_order["Profit"]/superstore_order["`In [62]: `superstore_order.groupby(["Category", "Sub-Category"]).agg(mean_profit_ratio = (`

Out[62]:

		mean_profit_ratio
Category	Sub-Category	
Furniture	Bookcases	-0.127756
	Chairs	0.045028
	Furnishings	0.140782
	Tables	-0.147916
Office Supplies	Appliances	-0.145513
	Art	0.251678
	Binders	-0.191641
	Envelopes	0.421913
	Fasteners	0.301157
	Labels	0.429984
	Paper	0.425586
	Storage	0.092382
	Supplies	0.104970
Technology	Accessories	0.219012
	Copiers	0.317826
	Machines	-0.059535
	Phones	0.118926

[Q8] Describe an information that the result above conveys?

Ans: แสดงค่าเฉลี่ย Profit Ratio ในแต่ละ Category และ Sub-Category

Pivot and Melt

Pivot

```
In [46]: superstore_order.pivot_table(index="State", columns="Ship Mode", values="Order I
```

Out[46]:

Ship Mode	First Class	Same Day	Second Class	Standard Class
State				
Alabama	9.0	1.0	18.0	30.0
Arizona	42.0	15.0	22.0	123.0
Arkansas	10.0	2.0	8.0	35.0
California	302.0	106.0	346.0	1000.0
Colorado	43.0	5.0	32.0	95.0
Connecticut	19.0	8.0	11.0	39.0
Delaware	16.0	2.0	13.0	55.0
District of Columbia	0.0	0.0	3.0	7.0
Florida	47.0	25.0	57.0	210.0
Georgia	19.0	15.0	31.0	108.0

```
In [47]: pivot_table_result = superstore_order.pivot_table(index="State", columns="Ship M
print(pivot_table_result)
```

Ship Mode State	First Class	Same Day	Second Class	Standard Class
Alabama	9.0	1.0	18.0	30.0
Arizona	42.0	15.0	22.0	123.0
Arkansas	10.0	2.0	8.0	35.0
California	302.0	106.0	346.0	1000.0
Colorado	43.0	5.0	32.0	95.0
Connecticut	19.0	8.0	11.0	39.0
Delaware	16.0	2.0	13.0	55.0
District of Columbia	0.0	0.0	3.0	7.0
Florida	47.0	25.0	57.0	210.0
Georgia	19.0	15.0	31.0	108.0
Idaho	3.0	0.0	2.0	13.0
Illinois	58.0	24.0	96.0	249.0
Indiana	13.0	3.0	30.0	79.0
Iowa	1.0	1.0	4.0	17.0
Kansas	6.0	1.0	2.0	15.0
Kentucky	12.0	5.0	49.0	62.0
Louisiana	7.0	2.0	14.0	15.0
Maine	0.0	0.0	0.0	5.0
Maryland	18.0	7.0	12.0	63.0
Massachusetts	14.0	4.0	35.0	71.0
Michigan	20.0	16.0	43.0	151.0
Minnesota	9.0	4.0	13.0	59.0
Mississippi	3.0	4.0	7.0	36.0
Missouri	7.0	2.0	20.0	24.0
Montana	1.0	1.0	0.0	13.0
Nebraska	6.0	3.0	6.0	20.0
Nevada	4.0	1.0	12.0	17.0
New Hampshire	2.0	0.0	10.0	13.0
New Jersey	5.0	1.0	20.0	87.0
New Mexico	1.0	0.0	9.0	22.0
New York	155.0	57.0	183.0	606.0
North Carolina	36.0	14.0	40.0	139.0
North Dakota	0.0	0.0	5.0	2.0
Ohio	66.0	47.0	84.0	199.0
Oklahoma	5.0	6.0	7.0	44.0
Oregon	20.0	0.0	15.0	81.0
Pennsylvania	103.0	9.0	78.0	341.0
Rhode Island	16.0	0.0	21.0	16.0
South Carolina	3.0	5.0	18.0	16.0
South Dakota	2.0	0.0	0.0	9.0
Tennessee	21.0	2.0	24.0	118.0
Texas	125.0	37.0	161.0	537.0
Utah	4.0	2.0	19.0	28.0
Vermont	0.0	0.0	1.0	2.0
Virginia	39.0	4.0	33.0	115.0
Washington	56.0	34.0	97.0	265.0
West Virginia	0.0	0.0	0.0	3.0
Wisconsin	12.0	3.0	10.0	66.0
Wyoming	0.0	0.0	0.0	1.0

Melt

```
In [48]: melted_result = pd.melt(pivot_table_result.reset_index(), id_vars=["State"], var
print(melted_result)
```


	State	Ship Mode	Order Count
0	Alabama	First Class	9.0
1	Arizona	First Class	42.0
2	Arkansas	First Class	10.0
3	California	First Class	302.0
4	Colorado	First Class	43.0
..
191	Virginia	Standard Class	115.0
192	Washington	Standard Class	265.0
193	West Virginia	Standard Class	3.0
194	Wisconsin	Standard Class	66.0
195	Wyoming	Standard Class	1.0

[196 rows x 3 columns]

[Q9] What is the advantage of using `melt` ?

ANS : เปลี่ยนรูปแบบข้อมูลจาก wide format เป็น long format ซึ่งช่วยให้การจัดการและการวิเคราะห์ข้อมูลทำได้ง่ายขึ้น

[Q10] From the `superstore_order`, display the ascending order considering values in the 'Profit' column to group the 'Category'.

In [49]: `#enter your code`
`superstore_order.groupby('Category').agg({'Profit': 'sum'}).sort_values(by='Prof`

Out[49]:

	Profit
Category	
Furniture	16858.5619
Office Supplies	105827.0238
Technology	133410.4932

[Q11] Create a new column that calculates the total price (sale*quantity) before discount then group by 'product id' and 'category', then show the mean of the total price

In [52]: `#enter your code here`
`superstore_order['Total Price'] = superstore_order['Sales'] * superstore_order['`
`result = superstore_order.groupby(['Product ID', 'Category'])['Total Price'].mea`
`result`

```
Out[52]: Product ID      Category      Total Price
FUR-BO-10000112  Furniture    7426.566000
FUR-BO-10000330  Furniture    1258.192000
FUR-BO-10000362  Furniture    1726.898000
FUR-BO-10000468  Furniture    426.532400
FUR-BO-10000711  Furniture    3194.100000
...
TEC-PH-10004912  Technology   747.320000
TEC-PH-10004922  Technology   673.249500
TEC-PH-10004924  Technology    57.149333
TEC-PH-10004959  Technology   412.009000
TEC-PH-10004977  Technology   2441.475429
Name: Total Price, Length: 1846, dtype: float64
```

[Q12] Complete the function to apply `ratio` column that calculates from `First Class` and `Standard Class` columns on `pivot_table_result`

```
In [54]: # [Q12] Complete the function to apply `ratio` column that calculates from `First Class` and `Standard Class` columns on `pivot_table_result`

# function to transform the ratio
def get_class_ratio(row):

    # get the first class column
    first_class = row['First Class']

    # get the standard class column
    standard_class = row['Standard Class']

    # calculate the ratio
    ratio = first_class/standard_class

    return ratio

pivot_table_result["ratio"] = pivot_table_result.apply(get_class_ratio, axis=1)

pivot_table_result.head()
```

```
Out[54]: Ship Mode  First Class  Same Day  Second Class  Standard Class  ratio
State
Alabama          9.0          1.0          18.0          30.0  0.300000
Arizona         42.0         15.0          22.0         123.0  0.341463
Arkansas         10.0          2.0           8.0          35.0  0.285714
California       302.0        106.0         346.0        1000.0  0.302000
Colorado         43.0          5.0          32.0          95.0  0.452632
```

[Q13] After complete Q12, What does the `apply` function do?

ANS : ฟังก์ชัน `apply` ใช้ `get_class_ratio` ไปยังแต่ละแถวของ DataFrame

[Q14] Create a new column(`short_ratio`) that works the same as Q12 but with `lambda` function

```
In [56]: # [Q13] Create a new column(`short_ratio`) that works the same as Q11 but with `
pivot_table_result["short_ratio"] = pivot_table_result.apply(lambda row: row['Fi
pivot_table_result.head()
```

```
Out[56]:
```

Ship Mode	First Class	Same Day	Second Class	Standard Class	ratio	short_ratio
Alabama	9.0	1.0	18.0	30.0	0.300000	0.300000
Arizona	42.0	15.0	22.0	123.0	0.341463	0.341463
Arkansas	10.0	2.0	8.0	35.0	0.285714	0.285714
California	302.0	106.0	346.0	1000.0	0.302000	0.302000
Colorado	43.0	5.0	32.0	95.0	0.452632	0.452632

[Q15] What is the difference between using `function` in `apply` and `lambda` function? give 2 examples use case.

ANS :

Regular Function: ใช้ฟังก์ชันมีตรรกะที่ซับซ้อนหรือมีหลายขั้นตอน และต้องการให้ฟังก์ชันนั้นสามารถนำไปใช้ซ้ำในหลายๆครั้ง

Lambda Function: ใช้ฟังก์ชันมีความง่ายและรวดเร็วในการคำนวณเพียงแค่หนึ่งบรรทัด โดยไม่ต้องการการอธิบายหรือการใช้งานซ้ำ

```
In [59]: df.columns
```

```
Out[59]: Index(['caseNumber', 'treatmentDate', 'statWeight', 'stratum', 'sex',
'diagnosis', 'bodyPart', 'disposition', 'location', 'product', 'age',
'Year', 'Month'],
dtype='object')
```

```
In [61]: def calculate_discount(row):
    if row['sex'] == 'Male': # Adjust based on your data
        return row['statWeight'] * 0.2
    elif row['statWeight'] > 100:
        return row['statWeight'] * 0.1
    return 0

df['Discount'] = df.apply(calculate_discount, axis=1)

df.head()
```

Out[61]:

	caseNumber	treatmentDate	statWeight	stratum	sex	diagnosis	bodyPart	dis
--	------------	---------------	------------	---------	-----	-----------	----------	-----

0	150733174	11/07/2015	15.7762	V	Male	57	33	
1	150734723	06/07/2015	83.2157	S	Male	57	34	
2	150817487	02/08/2015	74.8813	L	Female	71	94	
3	150717776	26/06/2015	15.7762	V	Male	71	35	
4	150721694	04/07/2015	74.8813	L	Female	62	75	



In [64]: `df['Discount'] = df.apply(lambda row: row['statWeight'] * 0.2 if row['sex'] == 'Male' else row['statWeight'], axis=1)`
`df.head()`

Out[64]:

	caseNumber	treatmentDate	statWeight	stratum	sex	diagnosis	bodyPart	dis
--	------------	---------------	------------	---------	-----	-----------	----------	-----

0	150733174	11/07/2015	15.7762	V	Male	57	33	
1	150734723	06/07/2015	83.2157	S	Male	57	34	
2	150817487	02/08/2015	74.8813	L	Female	71	94	
3	150717776	26/06/2015	15.7762	V	Male	71	35	
4	150721694	04/07/2015	74.8813	L	Female	62	75	



Punchaya Chanchaoen
65070507236