Lab 3: Data Preparation

CPE232 Data Models

[1] Reviews on Pandas

1.1) Discover

- methods to explore and understand your DataFrame

```
import pandas as pd

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

```
# see the shape of the dataframe
print(df.shape)
```

```
(334839, 12)
```

```
# 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
```

```
# 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
```

```
# 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
```

```
# 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
```

```
# 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

```
df['age']
```

```
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
```

```
df['age'].head()
```

```
0     5
1    36
2    20
3    61
4    88
Name: age, dtype: int64
```

```
df[['caseNumber', 'age']]
```

	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

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

	caseNumber	statWeight	age	diagnosis	bodyPart	disposition	location	product
0	150733174	15.7762	5	57	33	1	9	1267
1	150734723	83.2157	36	57	34	1	1	1439
2	150817487	74.8813	20	71	94	1	0	3274
3	150717776	15.7762	61	71	35	1	0	611
4	150721694	74.8813	88	62	75	1	0	1893
...	...	...	...	...	...	...	...	...
334834	150739278	15.0591	7	59	76	1	1	1864
334835	150733393	5.6748	3	68	85	1	0	1931
334836	150819286	15.7762	38	71	79	1	0	3250
334837	150823002	97.9239	38	59	82	1	1	464
334838	150723074	49.2646	5	57	34	1	9	3273

334839 rows × 8 columns

```
# select row by .loc
df.loc[0]
```

```
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
```

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

	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

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

	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

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

	caseNumber	treatmentDate	statWeight	stratum	age	sex	race	diagnosis	bodyPart	disposition	location	product
0	150733174	7/11/2015	15.7762	V	5	Male	NaN	57	33	1	9	1267
1	150734723	7/6/2015	83.2157	S	36	Male	White	57	34	1	1	1439
2	150817487	8/2/2015	74.8813	L	20	Female	NaN	71	94	1	0	3274
3	150717776	6/26/2015	15.7762	V	61	Male	NaN	71	35	1	0	611
4	150721694	7/4/2015	74.8813	L	88	Female	Other	62	75	1	0	1893

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

	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

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

	caseNumber	treatmentDate	statWeight	stratum	age	sex	race	diagnosis	bodyPart	disposition	location	product
3	150717776	6/26/2015	15.7762	V	61	Male	NaN	71	35	1	0	611
4	150721694	7/4/2015	74.8813	L	88	Female	Other	62	75	1	0	1893
7	150704114	6/14/2015	83.2157	S	53	Male	White	57	30	1	0	5040
8	150736558	7/16/2015	83.2157	S	98	Male	Black	59	76	1	1	1807
16	150901411	8/27/2015	83.2157	S	65	Female	White	59	83	1	1	1817
...	...	...	...	...	...	...	...	...	...	...	...	...
334811	150702215	6/27/2015	15.7762	V	51	Female	NaN	53	83	1	1	1426
334815	151100368	10/28/2015	83.2157	S	85	Female	NaN	57	80	4	1	1807
334819	150528367	1/13/2015	49.2646	M	85	Female	NaN	57	79	5	1	676
334826	150648619	6/17/2015	15.7762	V	52	Female	White	64	30	1	1	1842
334829	150633526	4/4/2015	49.2646	M	51	Female	NaN	56	92	1	1	1616

85235 rows × 12 columns

```
# filter coloum based on column name
df.filter(like='age')
```

	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

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

	caseNumber	treatmentDate	statWeight	stratum	age	sex	race	diagnosis	bodyPart	disposition	location	product
275174	150343700	3/9/2015	97.9239	M	48	Male	NaN	57	93	1	1	281
36	151029422	10/6/2015	97.9239	M	37	Male	White	64	35	1	0	1267
334806	150612491	5/29/2015	97.9239	M	18	Female	White	59	92	1	1	845
334810	150725804	7/8/2015	97.9239	M	33	Female	Black	71	94	1	0	1616
275161	150450816	4/13/2015	97.9239	M	24	Male	White	71	37	1	1	3286
...	...	...	...	...	...	...	...	...	...	...	...	...
44011	160222258	12/29/2015	4.9655	C	2	Female	Other	71	92	1	1	1893
325320	151213065	11/29/2015	4.9655	C	16	Female	White	62	75	1	8	3254
43891	160113865	12/28/2015	4.9655	C	4	Male	White	59	76	1	1	1842
43628	151130111	11/9/2015	4.9655	C	13	Male	Black	53	33	1	0	5011
43523	151139237	11/16/2015	4.9655	C	2	Female	Black	57	80	1	0	679

334839 rows × 12 columns

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

	caseNumber	treatmentDate	statWeight	stratum	age	sex	race	diagnosis	bodyPart	disposition	location	product
0	150733174	7/11/2015	15.7762	V	5	Male	NaN	57	33	1	9	1267
1	150734723	7/6/2015	83.2157	S	36	Male	White	57	34	1	1	1439
2	150817487	8/2/2015	74.8813	L	20	Female	NaN	71	94	1	0	3274
3	150717776	6/26/2015	15.7762	V	61	Male	NaN	71	35	1	0	611
4	150721694	7/4/2015	74.8813	L	88	Female	Other	62	75	1	0	1893
...	...	...	...	...	...	...	...	...	...	...	...	...
334834	150739278	5/31/2015	15.0591	V	7	Male	NaN	59	76	1	1	1864
334835	150733393	7/11/2015	5.6748	C	3	Female	Black	68	85	1	0	1931
334836	150819286	7/24/2015	15.7762	V	38	Male	NaN	71	79	1	0	3250
334837	150823002	8/8/2015	97.9239	M	38	Female	White	59	82	1	1	464
334838	150723074	6/20/2015	49.2646	M	5	Female	White	57	34	1	9	3273

334839 rows × 12 columns

1.5) Add/Remove

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

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

	caseNumber	treatmentDate	statWeight	stratum	age	sex	race	diagnosis	bodyPart	location	product
0	150733174	7/11/2015	15.7762	V	5	Male	NaN	57	33	9	1267
1	150734723	7/6/2015	83.2157	S	36	Male	White	57	34	1	1439
2	150817487	8/2/2015	74.8813	L	20	Female	NaN	71	94	0	3274
3	150717776	6/26/2015	15.7762	V	61	Male	NaN	71	35	0	611
4	150721694	7/4/2015	74.8813	L	88	Female	Other	62	75	0	1893
...	...	...	...	...	...	...	...	...	...	...	...
334834	150739278	5/31/2015	15.0591	V	7	Male	NaN	59	76	1	1864
334835	150733393	7/11/2015	5.6748	C	3	Female	Black	68	85	0	1931
334836	150819286	7/24/2015	15.7762	V	38	Male	NaN	71	79	0	3250
334837	150823002	8/8/2015	97.9239	M	38	Female	White	59	82	1	464
334838	150723074	6/20/2015	49.2646	M	5	Female	White	57	34	9	3273

334839 rows × 11 columns

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

	caseNumber	treatmentDate	statWeight	stratum	age	sex	race	diagnosis	bodyPart	disposition	location	product	new_
0	150733174	7/11/2015	15.7762	V	5	Male	NaN	57	33	1	9	1267	
1	150734723	7/6/2015	83.2157	S	36	Male	White	57	34	1	1	1439	
2	150817487	8/2/2015	74.8813	L	20	Female	NaN	71	94	1	0	3274	
3	150717776	6/26/2015	15.7762	V	61	Male	NaN	71	35	1	0	611	
4	150721694	7/4/2015	74.8813	L	88	Female	Other	62	75	1	0	1893	
...	...	...	...	...	...	...	...	...	...	...	...	...	
334834	150739278	5/31/2015	15.0591	V	7	Male	NaN	59	76	1	1	1864	
334835	150733393	7/11/2015	5.6748	C	3	Female	Black	68	85	1	0	1931	
334836	150819286	7/24/2015	15.7762	V	38	Male	NaN	71	79	1	0	3250	
334837	150823002	8/8/2015	97.9239	M	38	Female	White	59	82	1	1	464	
334838	150723074	6/20/2015	49.2646	M	5	Female	White	57	34	1	9	3273	

334839 rows × 13 columns

```
# 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

```
# replaceing the missing values with a specified value
df.fillna(value=0)
```

	caseNumber	treatmentDate	statWeight	stratum	sex	race	diagnosis	bodyPart	disposition	location	product
0	150733174	7/11/2015	15.7762	V	Male	0	57	33	1	9	1267
1	150734723	7/6/2015	83.2157	S	Male	White	57	34	1	1	1439
2	150817487	8/2/2015	74.8813	L	Female	0	71	94	1	0	3274
3	150717776	6/26/2015	15.7762	V	Male	0	71	35	1	0	611
4	150721694	7/4/2015	74.8813	L	Female	Other	62	75	1	0	1893
...	...	...	...	...	...	...	...	...	...	...	...
334834	150739278	5/31/2015	15.0591	V	Male	0	59	76	1	1	1864
334835	150733393	7/11/2015	5.6748	C	Female	Black	68	85	1	0	1931
334836	150819286	7/24/2015	15.7762	V	Male	0	71	79	1	0	3250
334837	150823002	8/8/2015	97.9239	M	Female	White	59	82	1	1	464
334838	150723074	6/20/2015	49.2646	M	Female	White	57	34	1	9	3273

334839 rows × 11 columns

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

	caseNumber	treatmentDate	statWeight	stratum	sex	race	diagnosis	bodyPart	disposition	location	product
1	150734723	7/6/2015	83.2157	S	Male	White	57	34	1	1	1439
4	150721694	7/4/2015	74.8813	L	Female	Other	62	75	1	0	1893
5	150721815	7/2/2015	5.6748	C	Female	White	71	76	1	1	1715
6	150713483	6/8/2015	15.7762	V	Male	Black	51	33	4	9	1138
7	150704114	6/14/2015	83.2157	S	Male	White	57	30	1	0	5040
...	...	...	...	...	...	...	...	...	...	...	...
334830	150628863	6/8/2015	15.7762	V	Female	White	64	79	1	1	1522
334831	150607637	5/22/2015	5.6748	C	Female	Black	59	94	1	0	1616
334835	150733393	7/11/2015	5.6748	C	Female	Black	68	85	1	0	1931
334837	150823002	8/8/2015	97.9239	M	Female	White	59	82	1	1	464
334838	150723074	6/20/2015	49.2646	M	Female	White	57	34	1	9	3273

205014 rows × 11 columns

[2] Data Cleaning and Preparation

.isnull, .dropna, .fillna

2.1) checking

```
df.columns
```

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

```
# isnull checking
df.isnull().sum()
```

```
caseNumber         0
treatmentDate      0
statWeight         0
```

```
stratum              0
sex                  2
race            129825
diagnosis            0
bodyPart             0
disposition          0
location             0
product              0
dtype: int64
```

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

```
np.float64(38.772365226272925)
```

```
df.shape[0]
```

```
334839
```

2.2) Drop column

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

```
df.head()
```

	caseNumber	treatmentDate	statWeight	stratum	sex	diagnosis	bodyPart	disposition	location	product
0	150733174	7/11/2015	15.7762	V	Male	57	33	1	9	1267
1	150734723	7/6/2015	83.2157	S	Male	57	34	1	1	1439
2	150817487	8/2/2015	74.8813	L	Female	71	94	1	0	3274
3	150717776	6/26/2015	15.7762	V	Male	71	35	1	0	611
4	150721694	7/4/2015	74.8813	L	Female	62	75	1	0	1893

2.3) Data imputation

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

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
File c:\Users\adiso\AppData\Local\Programs\Python\Python313\Lib\site-packages\pandas\core\indexes\base.py:3805, 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.PyObjectHashTable.get_item()

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

KeyError: 'age'

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

KeyError                                  Traceback (most recent call last)
Cell In[32], line 2
      1 # fillna
----> 2 df['age'] = df['age'].fillna(df['age'].median())

File c:\Users\adiso\AppData\Local\Programs\Python\Python313\Lib\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 c:\Users\adiso\AppData\Local\Programs\Python\Python313\Lib\site-packages\pandas\core\indexes\base.py:3812, 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)
```

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

Ans: The error message "KeyError: 'age'" indicates that the DataFrame does not contain a column named 'age'. Upon checking with df.columns, the available columns are:

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

This confirms that the 'age' column is missing from the DataFrame.

[Q2] Fix the error from Q1 problem.

```
# [Q2]

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

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

df.head()
```

	caseNumber	treatmentDate	statWeight	stratum	sex	diagnosis	bodyPart	disposition	location	product	age	Year	Month
0	150733174	11/07/2015	15.7762	V	Male	57	33	1	9	1267	5	2015	7
1	150734723	06/07/2015	83.2157	S	Male	57	34	1	1	1439	36	2015	7
2	150817487	02/08/2015	74.8813	L	Female	71	94	1	0	3274	20	2015	8
3	150717776	26/06/2015	15.7762	V	Male	71	35	1	0	611	61	2015	6
4	150721694	04/07/2015	74.8813	L	Female	62	75	1	0	1893	88	2015	7

2.4) Drop row that have missing value

```
# remove column by using .dropna()
df = df.dropna()
```

```
df.isnull().sum()
```

```
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

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

```
C:\Users\adiso\AppData\Local\Temp\ipykernel_17212\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
  df["treatmentDate"] = pd.to_datetime(df["treatmentDate"], format="%m/%d/%Y")
```

```
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
```

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

```
C:\Users\adiso\AppData\Local\Temp\ipykernel_17212\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
  df['Year'] = df['treatmentDate'].dt.year
```

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

```
C:\Users\adiso\AppData\Local\Temp\ipykernel_17212\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
  df['Month'] = df['treatmentDate'].dt.month
```

```
df.head()
```

	caseNumber	treatmentDate	statWeight	stratum	sex	diagnosis	bodyPart	disposition	location	product	age	Year	Month
0	150733174	2015-07-11	15.7762	V	Male	57	33	1	9	1267	0	2015	7
1	150734723	2015-07-06	83.2157	S	Male	57	34	1	1	1439	0	2015	7
2	150817487	2015-08-02	74.8813	L	Female	71	94	1	0	3274	0	2015	8
3	150717776	2015-06-26	15.7762	V	Male	71	35	1	0	611	0	2015	6
4	150721694	2015-07-04	74.8813	L	Female	62	75	1	0	1893	0	2015	7

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

```
# write your code here
df['treatmentDate'] = pd.to_datetime(df['treatmentDate'])

df['treatmentDate'] = df['treatmentDate'].dt.strftime('%d/%m/%Y')

print(df.head())
```

```
C:\Users\adiso\AppData\Local\Temp\ipykernel_17212\563021845.py:2: 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
  df['treatmentDate'] = pd.to_datetime(df['treatmentDate'])
   caseNumber treatmentDate  statWeight stratum     sex  diagnosis  bodyPart  \
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

   disposition  location  product  age  Year  Month
0            1         9     1267    0  2015      7
1            1         1     1439    0  2015      7
2            1         0     3274    0  2015      8
3            1         0      611    0  2015      6
4            1         0     1893    0  2015      7
C:\Users\adiso\AppData\Local\Temp\ipykernel_17212\563021845.py:4: 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
  df['treatmentDate'] = df['treatmentDate'].dt.strftime('%d/%m/%Y')
```

Combine Dataframe by .merge and .concat

2.6 Merge

```
import pandas as pd

superstore_order = pd.read_csv('superstore_order.csv')
superstore_people = pd.read_csv('superstore_people.csv')
superstore_return = pd.read_csv('superstore_return.csv')


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

	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: argument how="inner" is to ensures that only rows with matching "Order ID" values in both DataFrames are retained and to filtering out unmatched records.

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

Ans: List of Customers Who Returned Orders and also provide Customer ID column shows which customers. the infomation are removing Duplicates it can use to analyze in to find patterns.

More merging...

```
superstore_order.merge(superstore_return,
 on="Order ID" ,
 how="inner")
```

	Row ID	Order ID	Order Date	Ship Date	Ship Mode	Customer ID	Customer Name	Segment	Country	City	...	Region	Product ID	Categor
0	19	CA-2014-143336	27/08/2014	01/09/2014	Second Class	ZD-21925	Zuschuss Donatelli	Consumer	United States	San Francisco	...	West	OFF-AR-10003056	Offic Supplie
1	20	CA-2014-143336	27/08/2014	01/09/2014	Second Class	ZD-21925	Zuschuss Donatelli	Consumer	United States	San Francisco	...	West	TEC-PH-10001949	Technolo
2	21	CA-2014-143336	27/08/2014	01/09/2014	Second Class	ZD-21925	Zuschuss Donatelli	Consumer	United States	San Francisco	...	West	OFF-BI-10002215	Offic Supplie
3	56	CA-2016-111682	17/06/2016	18/06/2016	First Class	TB-21055	Ted Butterfield	Consumer	United States	Troy	...	East	OFF-ST-10000604	Offic Supplie
4	57	CA-2016-111682	17/06/2016	18/06/2016	First Class	TB-21055	Ted Butterfield	Consumer	United States	Troy	...	East	OFF-PA-10001569	Offic Supplie
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
702	8870	CA-2017-101805	01/12/2017	06/12/2017	Standard Class	SH-19975	Sally Hughsby	Corporate	United States	Seattle	...	West	OFF-BI-10002003	Offic Supplie
703	8871	CA-2017-101805	01/12/2017	06/12/2017	Standard Class	SH-19975	Sally Hughsby	Corporate	United States	Seattle	...	West	FUR-FU-10000023	Furnitu
704	8872	CA-2017-101805	01/12/2017	06/12/2017	Standard Class	SH-19975	Sally Hughsby	Corporate	United States	Seattle	...	West	OFF-ST-10002756	Offic Supplie
705	8873	US-2014-105137	10/10/2014	10/10/2014	Same Day	RB-19435	Richard Bierner	Consumer	United States	Columbus	...	East	TEC-MA-10002694	Technolo
706	8874	US-2014-105137	10/10/2014	10/10/2014	Same Day	RB-19435	Richard Bierner	Consumer	United States	Columbus	...	East	OFF-BI-10002429	Offic Supplie

707 rows × 22 columns

2.7) Concatenate

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

	Row ID	Order ID	Order Date	Ship Date	Ship Mode	Customer ID	Customer Name	Segment	Country	City	...	Product ID	Category	Sub-Category
0	1	CA-2016-152156	08/11/2016	11/11/2016	Second Class	CG-12520	Claire Gute	Consumer	United States	Henderson	...	FUR-BO-10001798	Furniture	Bookcases
1	2	CA-2016-152156	08/11/2016	11/11/2016	Second Class	CG-12520	Claire Gute	Consumer	United States	Henderson	...	FUR-CH-10000454	Furniture	Chairs
2	3	CA-2016-138688	12/06/2016	16/06/2016	Second Class	DV-13045	Darrin Van Huff	Corporate	United States	Los Angeles	...	OFF-LA-10000240	Office Supplies	Labels
3	4	US-2015-108966	11/10/2015	18/10/2015	Standard Class	SO-20335	Sean ODonnell	Consumer	United States	Fort Lauderdale	...	FUR-TA-10000577	Furniture	Tables

4 rows × 23 columns

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

Ans: inner: Keeps only indices that appear in the input DataFrames. outer: Keeps all indices that appear in any of the input DataFrames then using NULL to fill missing values.

Groupby

```
superstore_order.groupby(['Segment','Ship Mode'])[['Sales','Quantity','Discount','Profit']].sum()
```

Segment	Ship Mode	Sales	Quantity	Discount	Profit
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: Standard Class shipping is the highest volume shipping method. The profit columns show positive values on all segments and shipping modes

```
superstore_order["Profit Ratio"] = superstore_order["Profit"]/superstore_order["Sales"]
```

```
superstore_order.groupby(["Category", "Sub-Category"]).agg(mean_profit_ratio = ("Profit Ratio", "mean"))
```

Category	Sub-Category	mean_profit_ratio
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: Some sub-categories have negative profit ratios, indicating losses: Bookcases (-0.12756) Tables (-0.14792) Appliances (-0.14551) Binders (-0.19164) Machines (-0.05954)

```
Technology category mostly shows positive profit ratios:
   Copiers having the highest (0.31783)
   Only Machines showing a slight loss (-0.05954)

Office Supplies has several highly profitable sub-categories:
   Labels (0.42998)
   Paper (0.42559)
   Envelopes (0.42191)
   Fasteners (0.30116)
   Art (0.25168)
```

Pivot and Melt

Pivot

```
superstore_order.pivot_table(index="State", columns="Ship Mode", values="Order ID", aggfunc="count").fillna(0).head(10)
```

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

```
pivot_table_result = superstore_order.pivot_table(index="State", columns="Ship Mode", values="Order ID", aggfunc="count").fillna(0)
print(pivot_table_result)
```

```
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
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

```
melted_result = pd.melt(pivot_table_result.reset_index(), id_vars=["State"], var_name="Ship Mode", value_name="Order Count")
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: it helps transform "wide" format data into "long" format data, making it more suitable for certain types of analysis and visualization

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

```
#enter your code
# Group by Category, sum the Profit, and sort in ascending order
```

```
result = superstore_order.groupby('Category')['Profit'].sum().sort_values(ascending=True)
print(result)
```

```
Category
Furniture          16858.5619
Office Supplies   105827.0238
Technology        133410.4932
Name: Profit, dtype: float64
```

[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

```
#enter your code here
# Create a new column for total price before discount
superstore_order['Total_Price'] = superstore_order['Sales'] * superstore_order['Quantity']

# Group by product id and category, then calculate mean of total price
result = superstore_order.groupby(['Product ID', 'Category'])['Total_Price'].mean()
print(result)
```

```
Product ID       Category
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`

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

# 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()
```

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: to perform operations that are more complex than simple vectorized operations and returns a new Series with the results of applying the function to each row/column

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

```
pivot_table_result["short_ratio"] = pivot_table_result.apply(lambda row: row['First Class'] / row['Standard Class'], axis=1)

pivot_table_result.head()
```

Ship Mode	First Class	Same Day	Second Class	Standard Class	ratio	short_ratio
State						
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: Key differences: Regular functions are better for complex logic, multiple operations, or reusable code and documented and are more readable for complex operations Lambda functions are better for one-line operations, anonymous and can't be reused elsewhere in the code

Use cases: Use regular functions when I need documentation, complex logic, or reusability Use lambda functions when the operation is simple and used only once