Lab 3: Data Cleaning and Preparation

Objectives:

- To be more familiar with Pandas libraries
- To gain more hands-on experience in data cleaning and preparation

[1] More Reviews on Pandas

1.0) 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	61
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	193
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	bodyPa
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	

```
# 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	b
3	150717776	6/26/2015	15.7762	V	61	Male	NaN	71	
4	150721694	7/4/2015	74.8813	L	88	Female	Other	62	
7	150704114	6/14/2015	83.2157	S	53	Male	White	57	
8	150736558	7/16/2015	83.2157	S	98	Male	Black	59	
16	150901411	8/27/2015	83.2157	S	65	Female	White	59	
...	...	...	...	...	...	...	...	...	
334811	150702215	6/27/2015	15.7762	V	51	Female	NaN	53	
334815	151100368	10/28/2015	83.2157	S	85	Female	NaN	57	
334819	150528367	1/13/2015	49.2646	M	85	Female	NaN	57	
334826	150648619	6/17/2015	15.7762	V	52	Female	White	64	
334829	150633526	4/4/2015	49.2646	M	51	Female	NaN	56	

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	b
67072	150533084	5/15/2015	97.9239	M	89	Male	NaN	53	
313846	150521217	4/18/2015	97.9239	M	36	Female	NaN	64	
230135	150857760	8/25/2015	97.9239	M	14	Male	White	64	
141323	151039262	10/11/2015	97.9239	M	39	Female	White	71	
230141	150662453	6/5/2015	97.9239	M	11	Female	White	59	
...	...	...	...	...	...	...	...	...	
122009	151146792	11/15/2015	4.9655	C	2	Female	White	59	
211090	151253201	12/15/2015	4.9655	C	2	Male	White	60	
317625	160106638	12/25/2015	4.9655	C	1	Male	White	55	
33679	151256307	12/20/2015	4.9655	C	9	Female	Black	57	
229596	160148171	12/4/2015	4.9655	C	16	Female	Other	55	

334839 rows × 12 columns

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

	caseNumber	treatmentDate	statWeight	stratum	age	sex	race	diagnosis	b
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	
...	...	...	...	...	...	...	...	...	
334834	150739278	5/31/2015	15.0591	V	7	Male	NaN	59	
334835	150733393	7/11/2015	5.6748	C	3	Female	Black	68	
334836	150819286	7/24/2015	15.7762	V	38	Male	NaN	71	
334837	150823002	8/8/2015	97.9239	M	38	Female	White	59	
334838	150723074	6/20/2015	49.2646	M	5	Female	White	57	

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	b
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	
...	...	...	...	...	...	...	...	...	
334834	150739278	5/31/2015	15.0591	V	7	Male	NaN	59	
334835	150733393	7/11/2015	5.6748	C	3	Female	Black	68	
334836	150819286	7/24/2015	15.7762	V	38	Male	NaN	71	
334837	150823002	8/8/2015	97.9239	M	38	Female	White	59	
334838	150723074	6/20/2015	49.2646	M	5	Female	White	57	

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	b
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	
...	...	...	...	...	...	...	...	...	
334834	150739278	5/31/2015	15.0591	V	7	Male	NaN	59	
334835	150733393	7/11/2015	5.6748	C	3	Female	Black	68	
334836	150819286	7/24/2015	15.7762	V	38	Male	NaN	71	
334837	150823002	8/8/2015	97.9239	M	38	Female	White	59	
334838	150723074	6/20/2015	49.2646	M	5	Female	White	57	

334839 rows × 13 columns

```
# Removing the column and assigning it to a new variable
df.pop('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
```

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	bodyPa
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

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

	caseNumber	treatmentDate	statWeight	stratum	sex	race	diagnosis	bodyPa
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] Pandas Practice

Now that the knowledge about Pandas is still fresh, let's practice!

2.1) **[Question]** Use pandas to generate a *series* of 20 consecutive numbers, starting from 120.

```
import numpy as np
import pandas as pd
num = pd.Series(range(120,140))
print(num)
```

```
0     120
1     121
2     122
3     123
4     124
5     125
6     126
7     127
8     128
9     129
```

```
10    130
11    131
12    132
13    133
14    134
15    135
16    136
17    137
18    138
19    139
dtype: int64
```

2.2) **[Question]** Use pandas to generate a *series* of 20 even numbers, starting from 120.

```
even = pd.Series(range(120,160,2))
print(even)
```

```
0     120
1     122
2     124
3     126
4     128
5     130
6     132
7     134
8     136
9     138
10    140
11    142
12    144
13    146
14    148
15    150
16    152
17    154
18    156
19    158
dtype: int64
```

2.3) **[Question]** Use pandas to generate a *series* of 50 numbers in the Fibonacci sequence.

(Hint: The Fibonacci sequence is the series of numbers where each number is the sum of the two preceding numbers. For example, 0, 1, 1, 2, 3, 5, ...)

```
def fibonacci(n):
  fibo = [0,1]
  while len(fibo)<n:
    fibo.append(sum(fibo[-2:]))
  return fibo[:n]

num = pd.Series(fibonacci(50))
print(num)
```

```
0              0
1              1
2              1
3              2
4              3
5              5
6              8
7             13
8             21
9             34
10            55
11            89
12           144
13           233
14           377
15           610
16           987
17          1597
18          2584
19          4181
20          6765
21         10946
22         17711
23         28657
24         46368
25         75025
```

```
26        121393
27        196418
28        317811
29        514229
30        832040
31       1346269
32       2178309
33       3524578
34       5702887
35       9227465
36      14930352
37      24157817
38      39088169
39      63245986
40     102334155
41     165580141
42     267914296
43     433494437
44     701408733
45    1134903170
46    1836311903
47    2971215073
48    4807526976
49    7778742049
dtype: int64
```

2.4) **[Question]** Use pandas to generate a *series* of 20 random numbers.

```
random = pd.Series(np.random.randn(20))
print(random)
```

```
0     1.289892
1    -2.089178
2    -0.866082
3    -0.957709
4     0.862843
5     0.547167
6    -0.345073
7    -0.138995
8     2.584712
9    -0.657285
10    0.592690
11   -0.016348
12   -0.288702
13   -0.165038
14   -0.235155
15   -0.413447
16    1.556343
17    3.044533
18   -0.672022
19    0.520892
dtype: float64
```

2.5) **[Question]** Use pandas to generate a *series* of 20 random numbers, indexed in alphabetical order.

```
random = pd.Series(np.random.randn(20), index = list('ABCDEFGHIJKLMNOPQRST'))
print(random)
```

```
A    0.094669
B   -0.690833
C    0.138224
D    1.240561
E   -0.687463
F    0.134502
G    0.846827
H    0.271843
I    0.045277
J   -0.008694
K   -1.437961
L   -0.288176
M   -1.418809
N    0.856130
O    0.779236
P   -0.677710
Q    0.118488
R   -0.918571
S    0.546124
T    0.995225
dtype: float64
```

Next, we're going to use a dataframe which has already been created earlier at the beginning of this notebook. Let's view the first 5 rows (by default).

```
df = pd.read_csv('nss15.csv') # uncomment this line if the dataframe has been deleted.
df.head()
```

	caseNumber	treatmentDate	statWeight	stratum	age	sex	race	diagnosis	bodyPa
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	

2.6) **[Question]** Display *the first 12 rows*

```
df.head(12)
```

	caseNumber	treatmentDate	statWeight	stratum	age	sex	race	diagnosis	bodyP
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	
5	150721815	7/2/2015	5.6748	C	1	Female	White	71	
6	150713483	6/8/2015	15.7762	V	25	Male	Black	51	
7	150704114	6/14/2015	83.2157	S	53	Male	White	57	
8	150736558	7/16/2015	83.2157	S	98	Male	Black	59	
9	150734928	7/13/2015	74.8813	L	48	Female	Black	53	
10	150734952	7/4/2015	15.7762	V	20	Male	Black	59	
11	150821622	7/20/2015	83.2157	S	20	Female	White	57	

2.7) **[Question]** Display *the last 7 rows*

```
df.tail(7)
```

	caseNumber	treatmentDate	statWeight	stratum	age	sex	race	diagnosis	b
334832	150747209	7/24/2015	83.2157	S	14	Female	NaN	62	
334833	150747217	7/24/2015	83.2157	S	2	Male	NaN	62	
334834	150739278	5/31/2015	15.0591	V	7	Male	NaN	59	
334835	150733393	7/11/2015	5.6748	C	3	Female	Black	68	
334836	150819286	7/24/2015	15.7762	V	38	Male	NaN	71	
334837	150823002	8/8/2015	97.9239	M	38	Female	White	59	
334838	150723074	6/20/2015	49.2646	M	5	Female	White	57	

2.8) **[Question]** Display the last 5 rows (by default).

```
df.tail(5)
```

	caseNumber	treatmentDate	statWeight	stratum	age	sex	race	diagnosis	b
334834	150739278	5/31/2015	15.0591	V	7	Male	NaN	59	
334835	150733393	7/11/2015	5.6748	C	3	Female	Black	68	
334836	150819286	7/24/2015	15.7762	V	38	Male	NaN	71	
334837	150823002	8/8/2015	97.9239	M	38	Female	White	59	
334838	150723074	6/20/2015	49.2646	M	5	Female	White	57	

2.9) **[Question]** Select the column 'statWeight' and display

```
df['statWeight']
```

```
0         15.7762
1         83.2157
2         74.8813
3         15.7762
4         74.8813
           ...
334834    15.0591
334835     5.6748
334836    15.7762
334837    97.9239
334838    49.2646
Name: statWeight, Length: 334839, dtype: float64
```

2.10) **[Question]** Select the first 20 rows of the column 'statWeight' and display

```
df['statWeight'].head(20)
```

```
0     15.7762
1     83.2157
2     74.8813
3     15.7762
4     74.8813
5      5.6748
6     15.7762
7     83.2157
8     83.2157
9     74.8813
10    15.7762
11    83.2157
12    15.7762
13    15.7762
14    37.6645
15    83.2157
16    83.2157
17     5.6748
18    15.7762
19    97.9239
Name: statWeight, dtype: float64
```

2.11) **[Question]** Select the last 50 rows of the column 'statWeight' and find/compute the following values:

- Minimum
- Maximum
- Average
- Standard Deviation

```
data = df['statWeight'].tail(50)
min = data.min()
max = data.max()
avg = data.mean()
std = data.std()

print("", min, max, avg, std)
```

```
      5.6748 97.9239 45.411078 34.83805532712222
```

2.12) **[Question]** Select the first 25 rows of *two columns* 'statWeight' and 'age', then find/compute the following values for both columns:

- Minimum
- Maximum
- Average
- Standard Deviation

```
data = df[['statWeight', 'age']].head(25)

min1 = data['statWeight'].min()
max1 = data['statWeight'].max()
avg1 = data['statWeight'].mean()
std1 = data['statWeight'].std()

min2 = data['age'].min()
max2 = data['age'].max()
avg2 = data['age'].mean()
std2 = data['age'].std()

print("", min1, max1, avg1, std1)
print("", min2, max2, avg2, std2)
```

```
 5.6748 97.9239 47.033063999999996 34.547734626417984
 1 98 33.84 26.67501952514124
```

2.13) **[Question]** Select only columns that are of the *type integer*

```
inttype = df.select_dtypes(include='int')
print(inttype)
```

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

[334839 rows x 7 columns]
```

2.14) **[Question]** Select only columns that are of the type *string* or *character*

```
objtype = df.select_dtypes(include='object')
print(objtype)
```

```
       treatmentDate stratum     sex   race
0          7/11/2015       V    Male    NaN
1           7/6/2015       S    Male  White
2           8/2/2015       L  Female    NaN
3          6/26/2015       V    Male    NaN
4           7/4/2015       L  Female  Other
...              ...     ...     ...    ...
334834     5/31/2015       V    Male    NaN
334835     7/11/2015       C  Female  Black
334836     7/24/2015       V    Male    NaN
334837      8/8/2015       M  Female  White
334838     6/20/2015       M  Female  White

[334839 rows x 4 columns]
```

2.15) **[Question]** Display only unique values in *the column 'race'*

```
df['race'].unique()
```

```
array([nan, 'White', 'Other', 'Black', 'Asian', 'American Indian'],
      dtype=object)
```

2.16) **[Question]** Display rows with the following conditions:

- Patients are male

- The age ranges from 35 to 60 years old
- Could be of any race

```
df.loc[(df['sex'] == 'Male') & (df['age'] >= 35) & (df['age'] <= 60)]
```

	caseNumber	treatmentDate	statWeight	stratum	age	sex	race	diagnosis	bod
1	150734723	7/6/2015	83.2157	S	36	Male	White	57	
7	150704114	6/14/2015	83.2157	S	53	Male	White	57	
15	150655986	6/6/2015	83.2157	S	36	Male	NaN	59	
27	150913230	9/4/2015	15.7762	V	39	Male	NaN	71	
32	150908859	8/27/2015	37.6645	L	38	Male	Black	53	
...	...	...	...	...	...	...	...	...	
334769	150648575	6/16/2015	15.7762	V	47	Male	White	62	
334779	150612283	6/2/2015	15.7762	V	46	Male	NaN	68	
334800	150648581	6/16/2015	15.7762	V	52	Male	White	64	
334805	150511998	4/20/2015	15.0591	V	55	Male	Black	71	
334836	150819286	7/24/2015	15.7762	V	38	Male	NaN	71	

36406 rows × 12 columns

2.17) **[Question]** Based on your output in 2.16), select only the columns below to display.

- caseNumber
- treatmentDate
- race
- diagnosis
- bodyPart
- product

```
df.loc[(df['sex'] == 'Male') & (df['age'] >= 35) & (df['age'] <= 60), ['caseNumber', 'treatmentDate', 'race', 'diagnosis', 'bodyPart', 'pro
```

	caseNumber	treatmentDate	race	diagnosis	bodyPart	product
1	150734723	7/6/2015	White	57	34	1439
7	150704114	6/14/2015	White	57	30	5040
15	150655986	6/6/2015	NaN	59	82	894
27	150913230	9/4/2015	NaN	71	94	3274
32	150908859	8/27/2015	Black	53	36	5040
...	...	...	...	...	...	...
334769	150648575	6/16/2015	White	62	75	1615
334779	150612283	6/2/2015	NaN	68	85	5041
334800	150648581	6/16/2015	White	64	35	4074
334805	150511998	4/20/2015	Black	71	31	4014
334836	150819286	7/24/2015	NaN	71	79	3250

36406 rows × 6 columns

2.18) **[Question]** Let's change the condition a bit.

- Patients are female
- The age ranges from 5 to 40 years old
- Could be of any race

```
df.loc[(df['sex'] == 'Female') & (df['age'] >= 5) & (df['age'] <= 40)]
```

	caseNumber	treatmentDate	statWeight	stratum	age	sex	race	diagnosis	b
2	150817487	8/2/2015	74.8813	L	20	Female	NaN	71	
11	150821622	7/20/2015	83.2157	S	20	Female	White	57	
13	150666343	6/27/2015	15.7762	V	26	Female	White	62	
24	151029050	9/5/2015	49.2646	M	27	Female	NaN	58	
26	151005691	9/29/2015	74.8813	L	27	Female	Black	64	
...	...	...	...	...	...	...	...	...	
334827	150640832	6/8/2015	15.7762	V	8	Female	NaN	64	
334830	150628863	6/8/2015	15.7762	V	30	Female	White	64	
334832	150747209	7/24/2015	83.2157	S	14	Female	NaN	62	
334837	150823002	8/8/2015	97.9239	M	38	Female	White	59	
334838	150723074	6/20/2015	49.2646	M	5	Female	White	57	

71275 rows × 12 columns

2.19) **[Question]** Likewise, based on your output in 2.18), select only the columns below to display.

- caseNumber
- treatmentDate
- race
- diagnosis
- bodyPart
- product

```
df.loc[(df['sex'] == 'Male') & (df['age'] >= 35) & (df['age'] <= 60), ['caseNumber', 'treatmentDate', 'race', 'diagnosis', 'bodyPart', 'pro
```

	caseNumber	treatmentDate	race	diagnosis	bodyPart	product
1	150734723	7/6/2015	White	57	34	1439
7	150704114	6/14/2015	White	57	30	5040
15	150655986	6/6/2015	NaN	59	82	894
27	150913230	9/4/2015	NaN	71	94	3274
32	150908859	8/27/2015	Black	53	36	5040
...	...	...	...	...	...	...
334769	150648575	6/16/2015	White	62	75	1615
334779	150612283	6/2/2015	NaN	68	85	5041
334800	150648581	6/16/2015	White	64	35	4074
334805	150511998	4/20/2015	Black	71	31	4014
334836	150819286	7/24/2015	NaN	71	79	3250

36406 rows × 6 columns

[3] Data Cleaning and Preparation

.isnull, .dropna, .fillna

3.1) checking

```
# 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
```

```
38.772365226272925
```

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

```
334839
```

3.2) Drop column

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

```
df.head()
```

	caseNumber	treatmentDate	statWeight	stratum	sex	diagnosis	bodyPart	disposit
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	

3.3) Data imputation

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

3.4) 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
age              0
sex              0
race             0
diagnosis        0
bodyPart         0
disposition      0
location         0
product          0
dtype: int64
```

Datetime

3.5) Working with the datetime format

```
import pandas as pd
import numpy as np
df = pd.read_csv('nss15.csv')
```

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

```
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  datetime64[ns]
 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: datetime64[ns](1), float64(1), int64(7), object(3)
memory usage: 30.7+ MB
```

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

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

```
df.head()
```

	caseNumber	treatmentDate	statWeight	stratum	age	sex	race	diagnosis	bodyPa
0	150733174	11/07/15	15.7762	V	5	Male	NaN	57	
1	150734723	06/07/15	83.2157	S	36	Male	White	57	
2	150817487	02/08/15	74.8813	L	20	Female	NaN	71	
3	150717776	26/06/15	15.7762	V	61	Male	NaN	71	
4	150721694	04/07/15	74.8813	L	88	Female	Other	62	

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

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

	caseNumber	treatmentDate	statWeight	stratum	age	sex	race	diagnosis	bodyPa
0	150733174	2015-07-11	15.7762	V	5	Male	NaN	57	
1	150734723	2015-07-06	83.2157	S	36	Male	White	57	
2	150817487	2015-08-02	74.8813	L	20	Female	NaN	71	
3	150717776	2015-06-26	15.7762	V	61	Male	NaN	71	
4	150721694	2015-07-04	74.8813	L	88	Female	Other	62	

Combine Dataframe by .merge and .concat

3.6 Merge

```
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

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

Ans: The result show who return product that each customer has bought and the customer id show who bought the product earlier.

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

707 rows × 22 columns

3.7) 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	
0	1	CA-2016-152156	08/11/2016	11/11/2016	Second Class	CG-12520	Claire Gute	Consumer	United States	H
1	2	CA-2016-152156	08/11/2016	11/11/2016	Second Class	CG-12520	Claire Gute	Consumer	United States	H
2	3	CA-2016-138688	12/06/2016	16/06/2016	Second Class	DV-13045	Darrin Van Huff	Corporate	United States	
3	4	US-2015-108966	11/10/2015	18/10/2015	Standard Class	SO-20335	Sean ODonnell	Consumer	United States	L

4 rows × 23 columns

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

[Question] Briefly describe an information that the result above conveys?

Ans: The customer segment show that there are 3 types of customer which are consumer, corporate, and home office and each of them have different type of shipping, for example first class and standard class.

```
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

[Question] Briefly describe an information that the result above conveys?

Ans: It show the profit ratio of every product the company sell that has variesties of furniture, office services, and technology which each categories have a lot more products.

Pivot and Melt

Pivot

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

State	Ship Mode	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