Lab3-DataModels-Romain_Blanchot

February 5, 2025

1 Lab 3: Data Preparation

CPE232 Data Models

---

2 [1] Reviews on Pandas

1.1) Discover

- methods to explore and understand your DataFrame

[48]:
```
import pandas as pd

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

[49]:
```
# see the shape of the dataframe
print(df.shape)
```

```
(334839, 12)
```

[50]:
```
# 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
```

[51]:
```
# 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
```

[52]:
```
# 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
```

[53]:
```
# 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
```

[54]:
```
# 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

[55]:
```
df['age']
```

[55]:
```
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
```

[56]:
```
df['age'].head()
```

[56]:
```
0     5
1    36
2    20
3    61
4    88
Name: age, dtype: int64
```

[57]:
```
df[['caseNumber', 'age']]
```

[57]:

	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 x 2 columns]

[58]:
```
# select columns based on the data type
df.select_dtypes(include=['number'])
```

[58]:

	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 x 8 columns]

[59]:
```
# select row by .loc
df.loc[0]
```

[59]:
```
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
```

[60]:
```
# select column by .loc
df.loc[:6,'treatmentDate':'diagnosis']
```

[60]:

	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

[61]:
```
df.loc[df['age']>80, ['treatmentDate', 'age']]
```

[61]:

	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 x 2 columns]

[62]:
```
# select row by .iloc
df.iloc[0:5]
```

[62]:
```
   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
```

[63]:
```
# select column by .iloc
df.iloc[:,[0,1,2,3,4]]
```

[63]:
```
        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 x 5 columns]
```

1.3) Filtering the data

[64]:
```
# filter rows based on the condition
df[df['age'] > 50]
```

[64]:
```
        caseNumber treatmentDate  statWeight stratum  age     sex   race  \
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

        diagnosis  bodyPart  disposition  location  product
3              71        35            1         0      611
4              62        75            1         0     1893
7              57        30            1         0     5040
8              59        76            1         1     1807
16             59        83            1         1     1817
…             …         …             …         …        …
334811         53        83            1         1     1426
334815         57        80            4         1     1807
334819         57        79            5         1      676
334826         64        30            1         1     1842
334829         56        92            1         1     1616

[85235 rows x 12 columns]
```

[65]:
```
# filter coloum based on column name
df.filter(like='age')
```

[65]:
```
        age
0         5
1        36
2        20
3        61
4        88
…       …
334834    7
334835    3
334836   38
334837   38
334838    5

[334839 rows x 1 columns]
```

1.4) Sorting

- Sort the DataFrame by its index based on column

[66]:
```
# sort the dataframe based on column name and ascending order
df.sort_values(by='statWeight', ascending=False)
```

[66]:
```
        caseNumber treatmentDate  statWeight stratum  age     sex   race  \
59985    151114128     11/1/2015     97.9239       M   17    Male  Black
239709   150809041      8/2/2015     97.9239       M   34  Female  White
239711   151018719     10/5/2015     97.9239       M   72  Female  Black
239659   151138100    10/25/2015     97.9239       M   24    Male    NaN
239696   150903084     8/25/2015     97.9239       M   75    Male    NaN
```

```
…            …            …          …      …  …      …    …
123586   151201944    11/28/2015      4.9655       C   11  Female  White
123589   151226169     12/8/2015      4.9655       C    5    Male  Black
138289   151132404     11/8/2015      4.9655       C    6  Female  White
138295   151136771    11/14/2015      4.9655       C    5  Female  Black
138235   151150939    11/19/2015      4.9655       C   14  Female  White

        diagnosis  bodyPart  disposition  location  product
59985          64        37            1         1     1842
239709         57        82            1         1      611
239711         59        33            1         0     1871
239659         64        37            1         0     1211
239696         53        79            1         1     1807
…             …       …           …        …       …
123586         56        38            4         1     1669
123589         64        37            1         1     4076
138289         57        76            1         1     1684
138295         59        75            1         0     1807
138235         64        92            1         8     3265

[334839 rows x 12 columns]
```

[67]:
```
# sort the index of the dataframe
df.sort_index()
```

[67]:
```
        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
…              …            …           …      …  …      …    …
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
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
…             …       …           …        …       …
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

[334839 rows x 12 columns]
```

1.5) Add/Remove

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

[68]:
```
# Dropping the column
df.drop(columns=['disposition'])
```

[68]:
```
        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
…               …            …           …      …  …      …    …
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  location  product
0              57        33         9     1267
1              57        34         1     1439
2              71        94         0     3274
3              71        35         0      611
4              62        75         0     1893
…             …         …         …       …
334834         59        76         1     1864
334835         68        85         0     1931
334836         71        79         0     3250
334837         59        82         1      464
334838         57        34         9     3273

[334839 rows x 11 columns]
```

[69]:
```
# Adding column and create into a new column
df.assign(new_column=df['diagnosis'] + df['bodyPart'])
```

[69]:
```
        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
…              …            …          …      …  …     …    …
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  new_column
0              57        33            1         9     1267          90
1              57        34            1         1     1439          91
2              71        94            1         0     3274         165
3              71        35            1         0      611         106
4              62        75            1         0     1893         137
…             …         …           …         …        …           …
334834         59        76            1         1     1864         135
334835         68        85            1         0     1931         153
334836         71        79            1         0     3250         150
334837         59        82            1         1      464         141
334838         57        34            1         9     3273          91

[334839 rows x 13 columns]
```

[70]:
```
# 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

[71]:
```
# replaceing the missing values with a specified value
df.fillna(value=0)
```

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

        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
…             …       …          …        …       …
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

[334839 rows x 11 columns]
```

[72]:
```
# Remove the rows with missing values
df.dropna()
```

[72]:
```
        caseNumber treatmentDate  statWeight stratum     sex   race  \
1        150734723      7/6/2015     83.2157       S    Male  White
4        150721694      7/4/2015     74.8813       L  Female  Other
5        150721815      7/2/2015      5.6748       C  Female  White
6        150713483      6/8/2015     15.7762       V    Male  Black
7        150704114     6/14/2015     83.2157       S    Male  White
…              …           …          …      …     …    …
334830   150628863      6/8/2015     15.7762       V  Female  White
334831   150607637     5/22/2015      5.6748       C  Female  Black
334835   150733393     7/11/2015      5.6748       C  Female  Black
334837   150823002      8/8/2015     97.9239       M  Female  White
334838   150723074     6/20/2015     49.2646       M  Female  White

        diagnosis  bodyPart  disposition  location  product
1              57        34            1         1     1439
4              62        75            1         0     1893
5              71        76            1         1     1715
6              51        33            4         9     1138
7              57        30            1         0     5040
…             …       …          …        …       …
334830         64        79            1         1     1522
334831         59        94            1         0     1616
334835         68        85            1         0     1931
334837         59        82            1         1      464
334838         57        34            1         9     3273

[205014 rows x 11 columns]
```

3 [2] Data Cleaning and Preparation

3.0.1 .isnull, .dropna, .fillna

2.1) checking

[73]:
```
df.columns
```

[73]:
```
Index(['caseNumber', 'treatmentDate', 'statWeight', 'stratum', 'sex', 'race',
       'diagnosis', 'bodyPart', 'disposition', 'location', 'product'],
      dtype='object')
```

[74]:
```
# isnull checking
df.isnull().sum()
```

[74]:
```
caseNumber            0
treatmentDate         0
statWeight            0
stratum               0
sex                   2
race             129825
diagnosis             0
bodyPart              0
disposition           0
location              0
product               0
dtype: int64
```

[75]:
```
# percentage of missing values for the race
df.race.isnull().sum()/df.shape[0]*100
```

[75]:
```
np.float64(38.772365226272925)
```

[76]:
```
df.shape[0]
```

[76]:
```
334839
```

2.2) Drop column

[77]:
```
# remove column by using
df = df.drop(columns=['race'])
```

[78]:
```
df.head()
```

[78]:

	caseNumber	treatmentDate	statWeight	stratum	sex	diagnosis	bodyPart	\
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
```

```
   disposition  location  product
0            1         9     1267
1            1         1     1439
2            1         0     3274
3            1         0      611
4            1         0     1893
```

2.3) Data imputation

[79]:
```
# fillna
df['age'] = df['age'].fillna(df['age'].median())
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
File ~/anaconda3/envs/dataModel-course/lib/python3.12/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[79], line 2
      1 # fillna
----> 2 df['age'] = df['age'].fillna(df['age'].median())

File ~/anaconda3/envs/dataModel-course/lib/python3.12/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 ~/anaconda3/envs/dataModel-course/lib/python3.12/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)

KeyError: 'age'
```

[Q1] From the above cell, Why it showing an error?
Ans: The error occurs because earlier in the code, you executed ages = df.pop('age'), which removes the 'age' column from the DataFrame df. As a result, when you later try to access df['age'], it raises a KeyError since the column no longer exists in the DataFrame.

[Q2] Fix the error from Q1 problem.

[80]:
```
# [Q2]

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

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

df.head()
```

[80]:

	caseNumber	treatmentDate	statWeight	stratum	sex	diagnosis	bodyPart	\
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	

	disposition	location	product	age
0	1	9	1267	5
1	1	1	1439	36
2	1	0	3274	20
3	1	0	611	61
4	1	0	1893	88

2.4) Drop row that have missing value

[81]:
```
# remove column by using .dropna()
df = df.dropna()
```

[82]:
```
df.isnull().sum()
```

[82]:
```
caseNumber       0
treatmentDate    0
statWeight       0
stratum          0
sex              0
diagnosis        0
bodyPart         0
disposition      0
location         0
product          0
age              0
dtype: int64
```

3.0.2 Datetime

2.5) Working with the datetime format

[85]:
```
df["treatmentDate"] = pd.to_datetime(df["treatmentDate"], format="%m/%d/%Y")
df.head()
```

```
/tmp/ipykernel_491719/564257391.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")
```

[85]:

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

[86]:
```
df.info()
df.head()
```

```
<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
```

[86]:

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

[87]:
```
df['Year'] = df['treatmentDate'].dt.year
df.head()
```

```
/tmp/ipykernel_491719/527148567.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
```

[87]:

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

[88]:
```
df['Month'] = df['treatmentDate'].dt.month
df.head()
```

```
/tmp/ipykernel_491719/4039146640.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
```

[88]:

	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	5	2015	7
1	150734723	2015-07-06	83.2157	S	Male	57	34	1	1	1439	36	2015	7
2	150817487	2015-08-02	74.8813	L	Female	71	94	1	0	3274	20	2015	8
3	150717776	2015-06-26	15.7762	V	Male	71	35	1	0	611	61	2015	6
4	150721694	2015-07-04	74.8813	L	Female	62	75	1	0	1893	88	2015	7

[89]:
```
df.head()
```

[89]:

	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	5	2015	7
1	150734723	2015-07-06	83.2157	S	Male	57	34	1	1	1439	36	2015	7
2	150817487	2015-08-02	74.8813	L	Female	71	94	1	0	3274	20	2015	8
3	150717776	2015-06-26	15.7762	V	Male	71	35	1	0	611	61	2015	6
4	150721694	2015-07-04	74.8813	L	Female	62	75	1	0	1893	88	2015	7

	disposition	location	product	age	Year	Month
0	1	9	1267	5	2015	7
1	1	1	1439	36	2015	7
2	1	0	3274	20	2015	8
3	1	0	611	61	2015	6
4	1	0	1893	88	2015	7

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

[90]:
```
df["treatmentDate"] = pd.to_datetime(df["treatmentDate"]).dt.strftime("%d/%m/
  ↪%Y")
df.head()
```

```
/tmp/ipykernel_491719/2138053833.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"]).dt.strftime("%d/%m/%Y")
```

[90]:

	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	5	2015	7
1	1	1	1439	36	2015	7
2	1	0	3274	20	2015	8
3	1	0	611	61	2015	6
4	1	0	1893	88	2015	7

3.0.3 Combine Dataframe by .merge and .concat

2.6 Merge

[91]:
```
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')
```

[93]:
```
superstore_order.merge(superstore_return[superstore_return["Returned"]=="Yes"],
 on="Order ID" ,
 how="inner")\
```

```
 [["Customer ID", "Returned"]]\
 .drop_duplicates()
```

[93]:
```
     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 x 2 columns]
```

[Q2] What does the argument `how="inner"` do?

Ans: The `how="inner"` argument in the `merge` method ensures that only rows with matching values in both DataFrames are included in the result. If there is no match in `superstore_return` for a row in `superstore_order`, that row will be excluded. This effectively performs an intersection based on the specified column, "Order ID".

[Q3] In your opinion, what information that the result above conveys?

Ans: The result shows the list of customers who returned their orders, indicating which customers are likely dissatisfied or have issues with the products. This information can help in analyzing return patterns and improving customer satisfaction.

More merging...

[94]:
```
superstore_order.merge(superstore_return,
 on="Order ID" ,
 how="inner")
```

[94]:
```
     Row ID        Order ID  Order Date   Ship Date       Ship Mode  \
0        19  CA-2014-143336  27/08/2014  01/09/2014    Second Class
1        20  CA-2014-143336  27/08/2014  01/09/2014    Second Class
2        21  CA-2014-143336  27/08/2014  01/09/2014    Second Class
3        56  CA-2016-111682  17/06/2016  18/06/2016     First Class
4        57  CA-2016-111682  17/06/2016  18/06/2016     First Class
..      …              …         …          …              …
702    8870  CA-2017-101805  01/12/2017  06/12/2017  Standard Class
703    8871  CA-2017-101805  01/12/2017  06/12/2017  Standard Class
704    8872  CA-2017-101805  01/12/2017  06/12/2017  Standard Class
705    8873  US-2014-105137  10/10/2014  10/10/2014        Same Day
706    8874  US-2014-105137  10/10/2014  10/10/2014        Same Day
```

```
    Customer ID       Customer Name    Segment        Country           City  \
0      ZD-21925  Zuschuss Donatelli   Consumer  United States  San Francisco
1      ZD-21925  Zuschuss Donatelli   Consumer  United States  San Francisco
2      ZD-21925  Zuschuss Donatelli   Consumer  United States  San Francisco
3      TB-21055     Ted Butterfield   Consumer  United States           Troy
4      TB-21055     Ted Butterfield   Consumer  United States           Troy
..          …                   …          …             …            …
702    SH-19975       Sally Hughsby  Corporate  United States        Seattle
703    SH-19975       Sally Hughsby  Corporate  United States        Seattle
704    SH-19975       Sally Hughsby  Corporate  United States        Seattle
705    RB-19435     Richard Bierner   Consumer  United States       Columbus
706    RB-19435     Richard Bierner   Consumer  United States       Columbus

     … Region       Product ID         Category Sub-Category  \
0    …   West  OFF-AR-10003056  Office Supplies          Art
1    …   West  TEC-PH-10001949       Technology       Phones
2    …   West  OFF-BI-10002215  Office Supplies      Binders
3    …   East  OFF-ST-10000604  Office Supplies      Storage
4    …   East  OFF-PA-10001569  Office Supplies        Paper
..   …    …              …              …          …
702  …   West  OFF-BI-10002003  Office Supplies      Binders
703  …   West  FUR-FU-10000023        Furniture  Furnishings
704  …   West  OFF-ST-10002756  Office Supplies      Storage
705  …   East  TEC-MA-10002694       Technology     Machines
706  …   East  OFF-BI-10002429  Office Supplies      Binders

                                              Product Name    Sales Quantity  \
0                                             Newell 341    8.560        2
1                                 Cisco SPA 501G IP Phone  213.480        3
2              Wilson Jones Hanging View Binder White 1   22.720        4
3                       Home/Office Personal File Carts  208.560        6
4                                           Xerox 232   32.400        5
..                                                 …      …      …
702       Ibico Presentation Index for Binding Systems   15.920        5
703                           Eldon Wave Desk Accessories   70.680       12
704  Tennsco Stur-D-Stor Boltless Shelving 5 Shelve…  541.240        4
705  Hewlett-Packard Deskjet F4180 All-in-One Color…  101.994        2
706                Premier Elliptical Ring Binder Black   18.264        2

     Discount   Profit  Returned
0         0.0   2.4824       Yes
1         0.2  16.0110       Yes
2         0.2   7.3840       Yes
3         0.0  52.1400       Yes
4         0.0  15.5520       Yes
..        …      …       …
702       0.2   5.3730       Yes
```

```
703       0.0  31.0992       Yes
704       0.0   5.4124       Yes
705       0.7 -71.3958       Yes
706       0.7 -13.3936       Yes

[707 rows x 22 columns]
```

2.7) Concatenate

[95]:
```
pd.concat([superstore_order, superstore_people], axis=1, join='inner')
```

[95]:
```
   Row ID        Order ID  Order Date   Ship Date       Ship Mode Customer ID  \
0       1  CA-2016-152156  08/11/2016  11/11/2016    Second Class    CG-12520
1       2  CA-2016-152156  08/11/2016  11/11/2016    Second Class    CG-12520
2       3  CA-2016-138688  12/06/2016  16/06/2016    Second Class    DV-13045
3       4  US-2015-108966  11/10/2015  18/10/2015  Standard Class    SO-20335

     Customer Name    Segment        Country             City  …  \
0      Claire Gute   Consumer  United States        Henderson  …
1      Claire Gute   Consumer  United States        Henderson  …
2  Darrin Van Huff  Corporate  United States      Los Angeles  …
3    Sean ODonnell   Consumer  United States  Fort Lauderdale  …

        Product ID         Category Sub-Category  \
0  FUR-BO-10001798        Furniture    Bookcases
1  FUR-CH-10000454        Furniture       Chairs
2  OFF-LA-10000240  Office Supplies       Labels
3  FUR-TA-10000577        Furniture       Tables

                                     Product Name     Sales Quantity  \
0                  Bush Somerset Collection Bookcase  261.9600        2
1  Hon Deluxe Fabric Upholstered Stacking Chairs …  731.9400        3
2  Self-Adhesive Address Labels for Typewriters b…   14.6200        2
3      Bretford CR4500 Series Slim Rectangular Table  957.5775        5

  Discount    Profit             Person   Region
0     0.00   41.9136      Anna Andreadi     West
1     0.00  219.5820        Chuck Magee     East
2     0.00    6.8714     Kelly Williams  Central
3     0.45 -383.0310  Cassandra Brandow    South

[4 rows x 23 columns]
```

[Q4] What is the difference between `inner` and `outer` on parameter `join` in `pd.concat`?

The `inner` join in `pd.concat` includes only the rows with matching indices from both DataFrames, while the `outer` join includes all rows from both DataFrames, filling in missing values with NaN where there are no matches. Essentially, `inner` provides the intersection, and `outer` provides the union of the DataFrames.

3.0.4 Groupby

[96]:
```
superstore_order.groupby(['Segment','Ship␣
↪Mode'])[['Sales','Quantity','Discount','Profit']].sum()
```

[96]:

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

[Q5] Describe an information that the result above conveys?

Ans: The result summarizes total sales, quantity, discount, and profit for each combination of segment and shipping mode, providing insights into performance across different customer segments and shipping methods.

[97]:
```
superstore_order["Profit Ratio"] = superstore_order["Profit"]/
↪superstore_order["Sales"]
```

[98]:
```
superstore_order.groupby(["Category", "Sub-Category"]).agg(mean_profit_ratio =␣
↪("Profit Ratio", "mean"))
```

[98]:

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
```

[Q6] Describe an information that the result above conveys?

Ans: The result shows the average profit ratio for each category and sub-category, helping to identify which product groups are more profitable.

3.0.5 Pivot and Melt

Pivot

[106]:
```
superstore_order.pivot_table(index="State", columns="Ship Mode", values="Order␣
↪ID", aggfunc="count").fillna(0).head(10)
```

[106]:

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

[107]:
```
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

[108]:
```
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]
```

[Q7] What is the advantage of using `melt`? The advantage of using `melt` is that it transforms a wide-format DataFrame into a long-format DataFrame, making it easier to analyze and visualize data by consolidating multiple columns into key-value pairs. This format is often more suitable for plotting and further data manipulation.

[Q8] From the superstore_order, display the ascending order considering values in the 'Profit' column to group the 'Category'.

[109]:
```
#enter your code
sorted_profit = superstore_order.groupby('Category')['Profit'].sum().
 ↪sort_values(ascending=True)
print(sorted_profit)
```

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

[Q9] 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

[110]:
```
superstore_order['Total Price'] = superstore_order['Sales'] *␣
 ↪superstore_order['Quantity']
total_price_mean = superstore_order.groupby(['Product ID', 'Category'])['Total␣
 ↪Price'].mean()
print(total_price_mean)
```

```
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
```

[Q10] Complete the function to apply `ratio` column that calculates from `First Class` and `Standard Class` columns on `pivot_table_result`

[111]:
```
# [Q10] Complete the function to apply `ratio` column that calculates from␣
↪`First Class` and `Standard Class` columns on `pivot_table_result`

def get_class_ratio(row):

    first_class = row['First Class']

    standard_class = row['Standard Class']

    if standard_class == 0:
        ratio = 0
    else:
        ratio = first_class / standard_class

    return ratio

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

pivot_table_result.head()
```

[111]:

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

[Q11] After complete Q10, What does the `apply` function do? The `apply` function applies a specified function to each row or column of a DataFrame.

[Q12] Create a new column(`short_ratio`) that works the same as Q10 but with `lambda` function

[112]:
```
# [Q12] Create a new column(`short_ratio`) that works the same as Q10 but with␣
↪`lambda` function

pivot_table_result["short_ratio"] = pivot_table_result.apply(lambda row:␣
↪row['First Class'] / row['Standard Class'] if row['Standard Class'] != 0␣
↪else 0, axis=1)

pivot_table_result.head()
```

[112]:

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
```

```
Ship Mode   short_ratio
State
Alabama        0.300000
Arizona        0.341463
Arkansas       0.285714
California     0.302000
Colorado       0.452632
```

[Q13] What is the difference between `apply` and `lambda` function? give 2 examples use case.

The `apply` function is used to apply a function along the axis of a DataFrame (rows or columns), while a `lambda` function is an anonymous function defined with the `lambda` keyword, often used for short, simple operations.

- Example 1: Using `apply` to calculate the mean of a column mean_value = df['column_name'].apply(lambda x: x.mean())
- Example 2: Using `lambda` to filter rows based on a condition filtered_df = df[df['column_name'].apply(lambda x: x > 10)]