Chapter2

March 1, 2022

1 Chapter 2 - Data Preparation Basics

1.1 Segment 1 - Filtering and selecting data

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

from pandas import Series, DataFrame
```

1.1.1 Selecting and retrieving data

You can write an index value in two forms. - Label index or - Integer index

[2]:
```
series_obj = Series(np.arange(8), index=['row 1', 'row 2', 'row 3', 'row 4',␣
↪'row 5', 'row 6','row 7', 'row 8'])
series_obj
```

[2]:
```
row 1    0
row 2    1
row 3    2
row 4    3
row 5    4
row 6    5
row 7    6
row 8    7
dtype: int32
```

[3]:
```
series_obj['row 7']
```

[3]:
```
6
```

[4]:
```
series_obj[[0, 7]]
```

[4]:
```
row 1    0
row 8    7
dtype: int32
```

[5]:
```
np.random.seed(25)
DF_obj = DataFrame(np.random.rand(36).reshape((6,6)),
                   index=['row 1', 'row 2', 'row 3', 'row 4','row 5','row 6'],
                   columns=['column 1','column 2','column 3','column 4','column␣
 ↪5','column 6'])
DF_obj
```

[5]:

	column 1	column 2	column 3	column 4	column 5	column 6
row 1	0.870124	0.582277	0.278839	0.185911	0.411100	0.117376
row 2	0.684969	0.437611	0.556229	0.367080	0.402366	0.113041
row 3	0.447031	0.585445	0.161985	0.520719	0.326051	0.699186
row 4	0.366395	0.836375	0.481343	0.516502	0.383048	0.997541
row 5	0.514244	0.559053	0.034450	0.719930	0.421004	0.436935
row 6	0.281701	0.900274	0.669612	0.456069	0.289804	0.525819

[6]:
```
DF_obj.loc[['row 2', 'row 5'], ['column 5', 'column 2']]
```

[6]:

	column 5	column 2
row 2	0.402366	0.437611
row 5	0.421004	0.559053

1.1.2 Data slicing

You can use slicing to select and return a slice of several values from a data set. Slicing uses index values so you can use the same square brackets when doing data slicing.

How slicing differs, however, is that with slicing you pass in two index values that are separated by a colon. The index value on the left side of the colon should be the first value you want to select. On the right side of the colon, you write the index value for the last value you want to retrieve. When you execute the code, the indexer then simply finds the first record and the last record and returns every record in between them.

[7]:
```
series_obj['row 3': 'row 7']
```

[7]:
```
row 3    2
row 4    3
row 5    4
row 6    5
row 7    6
dtype: int32
```

1.1.3 Comparing with scalars

Now we're going to talk about comparison operators and scalar values. Just in case you don't know that a scalar value is, it's basically just a single numerical value. You can use comparison operators like greater than or less than to return true/false values for all records to indicate how each element compares to a scalar value.

[8]:
```
DF_obj < .2
```

[8]:

	column 1	column 2	column 3	column 4	column 5	column 6
row 1	False	False	False	True	False	True
row 2	False	False	False	False	False	True
row 3	False	False	True	False	False	False
row 4	False	False	False	False	False	False
row 5	False	False	True	False	False	False
row 6	False	False	False	False	False	False

1.1.4 Filtering with scalars

[9]:
```
series_obj[series_obj > 6]
```

[9]:
```
row 8    7
dtype: int32
```

1.1.5 Setting values with scalars

[10]:
```
series_obj['row 1', 'row 5', 'row 8'] = 8
series_obj
```

[10]:
```
row 1    8
row 2    1
row 3    2
row 4    3
row 5    8
row 6    5
row 7    6
row 8    8
dtype: int32
```

Filtering and selecting using Pandas is one of the most fundamental things you'll do in data analysis. Make sure you know how to use indexing to select and retrieve records.

2 Chapter 2 - Data Preparation Basics

2.1 Segment 2 - Treating missing values

[11]:
```
import numpy as np
import pandas as pd

from pandas import Series, DataFrame
```

2.1.1 Figuring out what data is missing

[12]:
```
missing = np.nan
```

```
series_obj = Series(['row 1', 'row 2', missing, 'row 4', 'row 5', 'row 6',␣
 ↪missing, 'row 8'])
series_obj
```

[12]:
```
0    row 1
1    row 2
2      NaN
3    row 4
4    row 5
5    row 6
6      NaN
7    row 8
dtype: object
```

[13]:
```
series_obj.isnull()
```

[13]:
```
0    False
1    False
2     True
3    False
4    False
5    False
6     True
7    False
dtype: bool
```

2.1.2 Filling in for missing values

[14]:
```
np.random.seed(25)
DF_obj = DataFrame(np.random.rand(36).reshape(6,6))
DF_obj
```

[14]:

	0	1	2	3	4	5
0	0.870124	0.582277	0.278839	0.185911	0.411100	0.117376
1	0.684969	0.437611	0.556229	0.367080	0.402366	0.113041
2	0.447031	0.585445	0.161985	0.520719	0.326051	0.699186
3	0.366395	0.836375	0.481343	0.516502	0.383048	0.997541
4	0.514244	0.559053	0.034450	0.719930	0.421004	0.436935
5	0.281701	0.900274	0.669612	0.456069	0.289804	0.525819

[15]:
```
DF_obj.loc[3:5, 0] = missing
DF_obj.loc[1:4, 5] = missing
DF_obj
```

[15]:

	0	1	2	3	4	5
0	0.870124	0.582277	0.278839	0.185911	0.411100	0.117376
1	0.684969	0.437611	0.556229	0.367080	0.402366	NaN

```
2  0.447031  0.585445  0.161985  0.520719  0.326051       NaN
3       NaN  0.836375  0.481343  0.516502  0.383048       NaN
4       NaN  0.559053  0.034450  0.719930  0.421004       NaN
5       NaN  0.900274  0.669612  0.456069  0.289804  0.525819
```

[16]:
```
filled_DF = DF_obj.fillna(0)
filled_DF
```

[16]:
```
          0         1         2         3         4         5
0  0.870124  0.582277  0.278839  0.185911  0.411100  0.117376
1  0.684969  0.437611  0.556229  0.367080  0.402366  0.000000
2  0.447031  0.585445  0.161985  0.520719  0.326051  0.000000
3  0.000000  0.836375  0.481343  0.516502  0.383048  0.000000
4  0.000000  0.559053  0.034450  0.719930  0.421004  0.000000
5  0.000000  0.900274  0.669612  0.456069  0.289804  0.525819
```

[17]:
```
filled_DF = DF_obj.fillna({0: 0.1, 5:1.25})
filled_DF
```

[17]:
```
          0         1         2         3         4         5
0  0.870124  0.582277  0.278839  0.185911  0.411100  0.117376
1  0.684969  0.437611  0.556229  0.367080  0.402366  1.250000
2  0.447031  0.585445  0.161985  0.520719  0.326051  1.250000
3  0.100000  0.836375  0.481343  0.516502  0.383048  1.250000
4  0.100000  0.559053  0.034450  0.719930  0.421004  1.250000
5  0.100000  0.900274  0.669612  0.456069  0.289804  0.525819
```

[18]:
```
fill_DF = DF_obj.fillna(method='ffill')
fill_DF
```

[18]:
```
          0         1         2         3         4         5
0  0.870124  0.582277  0.278839  0.185911  0.411100  0.117376
1  0.684969  0.437611  0.556229  0.367080  0.402366  0.117376
2  0.447031  0.585445  0.161985  0.520719  0.326051  0.117376
3  0.447031  0.836375  0.481343  0.516502  0.383048  0.117376
4  0.447031  0.559053  0.034450  0.719930  0.421004  0.117376
5  0.447031  0.900274  0.669612  0.456069  0.289804  0.525819
```

2.1.3 Counting missing values

[19]:
```
np.random.seed(25)
DF_obj = DataFrame(np.random.rand(36).reshape(6,6))
DF_obj.loc[3:5, 0] = missing
DF_obj.loc[1:4, 5] = missing
DF_obj
```

[19]:

	0	1	2	3	4	5
0	0.870124	0.582277	0.278839	0.185911	0.411100	0.117376
1	0.684969	0.437611	0.556229	0.367080	0.402366	NaN
2	0.447031	0.585445	0.161985	0.520719	0.326051	NaN
3	NaN	0.836375	0.481343	0.516502	0.383048	NaN
4	NaN	0.559053	0.034450	0.719930	0.421004	NaN
5	NaN	0.900274	0.669612	0.456069	0.289804	0.525819

[20]:
```
DF_obj.isnull().sum()
```

[20]:
```
0    3
1    0
2    0
3    0
4    0
5    4
dtype: int64
```

[21]:
```
DF_no_NaN = DF_obj.dropna()
DF_no_NaN
```

[21]:

	0	1	2	3	4	5
0	0.870124	0.582277	0.278839	0.185911	0.4111	0.117376

[22]:
```
DF_no_NaN = DF_obj.dropna(axis=1)
DF_no_NaN
```

[22]:

	1	2	3	4
0	0.582277	0.278839	0.185911	0.411100
1	0.437611	0.556229	0.367080	0.402366
2	0.585445	0.161985	0.520719	0.326051
3	0.836375	0.481343	0.516502	0.383048
4	0.559053	0.034450	0.719930	0.421004
5	0.900274	0.669612	0.456069	0.289804

3 Chapter 2 - Data Preparation Basics

3.1 Segment 3 - Removing duplicates

3.1.1 Removing duplicates

[23]:
```
DF_obj= DataFrame({'column 1':[1,1,2,2,3,3,3],
                   'column 2':['a', 'a','b', 'b', 'c', 'c', 'c'],
                   'column 3':['A', 'A', 'B', 'B', 'C', 'C', 'C']})
DF_obj
```

[23]:

	column 1	column 2	column 3
0	1	a	A

```
1          1          a          A
2          2          b          B
3          2          b          B
4          3          c          C
5          3          c          C
6          3          c          C
```

[24]:
```
DF_obj.duplicated()
```

[24]:
```
0    False
1     True
2    False
3     True
4    False
5     True
6     True
dtype: bool
```

[25]:
```
DF_obj.drop_duplicates()
```

[25]:
```
   column 1 column 2 column 3
0          1          a          A
2          2          b          B
4          3          c          C
```

3.2 Segment 4 - Concatenating and transforming data

[26]:
```
DF_obj = pd.DataFrame(np.arange(36).reshape(6,6))
DF_obj
```

[26]:
```
    0   1   2   3   4   5
0   0   1   2   3   4   5
1   6   7   8   9  10  11
2  12  13  14  15  16  17
3  18  19  20  21  22  23
4  24  25  26  27  28  29
5  30  31  32  33  34  35
```

[27]:
```
DF_obj_2 = pd.DataFrame(np.arange(15).reshape(5,3))
DF_obj_2
```

[27]:
```
    0   1   2
0   0   1   2
1   3   4   5
2   6   7   8
3   9  10  11
4  12  13  14
```

3.2.1 Concatenating data

[28]:
```
pd.concat([DF_obj, DF_obj_2], axis=1)
```

[28]:

	0	1	2	3	4	5	0	1	2
0	0	1	2	3	4	5	0.0	1.0	2.0
1	6	7	8	9	10	11	3.0	4.0	5.0
2	12	13	14	15	16	17	6.0	7.0	8.0
3	18	19	20	21	22	23	9.0	10.0	11.0
4	24	25	26	27	28	29	12.0	13.0	14.0
5	30	31	32	33	34	35	NaN	NaN	NaN

[29]:
```
pd.concat([DF_obj, DF_obj_2])
```

[29]:

	0	1	2	3	4	5
0	0	1	2	3.0	4.0	5.0
1	6	7	8	9.0	10.0	11.0
2	12	13	14	15.0	16.0	17.0
3	18	19	20	21.0	22.0	23.0
4	24	25	26	27.0	28.0	29.0
5	30	31	32	33.0	34.0	35.0
0	0	1	2	NaN	NaN	NaN
1	3	4	5	NaN	NaN	NaN
2	6	7	8	NaN	NaN	NaN
3	9	10	11	NaN	NaN	NaN
4	12	13	14	NaN	NaN	NaN

3.2.2 Transforming data

Dropping data

[30]:
```
DF_obj.drop([0, 2])
```

[30]:

	0	1	2	3	4	5
1	6	7	8	9	10	11
3	18	19	20	21	22	23
4	24	25	26	27	28	29
5	30	31	32	33	34	35

[31]:
```
DF_obj.drop([0, 2], axis=1)
```

[31]:

	1	3	4	5
0	1	3	4	5
1	7	9	10	11
2	13	15	16	17
3	19	21	22	23
4	25	27	28	29
5	31	33	34	35

3.2.3 Adding data

[32]:
```
series_obj = Series(np.arange(6))
series_obj.name = "added_variable"
series_obj
```

[32]:
```
0    0
1    1
2    2
3    3
4    4
5    5
Name: added_variable, dtype: int32
```

[33]:
```
variable_added = DataFrame.join(DF_obj, series_obj)
variable_added
```

[33]:

	0	1	2	3	4	5	added_variable
0	0	1	2	3	4	5	0
1	6	7	8	9	10	11	1
2	12	13	14	15	16	17	2
3	18	19	20	21	22	23	3
4	24	25	26	27	28	29	4
5	30	31	32	33	34	35	5

[34]:
```
added_datatable = variable_added.append(variable_added, ignore_index=False)
added_datatable
```

[34]:

	0	1	2	3	4	5	added_variable
0	0	1	2	3	4	5	0
1	6	7	8	9	10	11	1
2	12	13	14	15	16	17	2
3	18	19	20	21	22	23	3
4	24	25	26	27	28	29	4
5	30	31	32	33	34	35	5
0	0	1	2	3	4	5	0
1	6	7	8	9	10	11	1
2	12	13	14	15	16	17	2
3	18	19	20	21	22	23	3
4	24	25	26	27	28	29	4
5	30	31	32	33	34	35	5

[35]:
```
added_datatable = variable_added.append(variable_added, ignore_index=True)
added_datatable
```

[35]:

	0	1	2	3	4	5	added_variable
0	0	1	2	3	4	5	0
1	6	7	8	9	10	11	1

```
2   12  13  14  15  16  17              2
3   18  19  20  21  22  23              3
4   24  25  26  27  28  29              4
5   30  31  32  33  34  35              5
6    0   1   2   3   4   5              0
7    6   7   8   9  10  11              1
8   12  13  14  15  16  17              2
9   18  19  20  21  22  23              3
10  24  25  26  27  28  29              4
11  30  31  32  33  34  35              5
```

3.2.4 Sorting data

[36]:
```
DF_sorted = DF_obj.sort_values(by=(5), ascending=[False])
DF_sorted
```

[36]:
```
    0   1   2   3   4   5
5  30  31  32  33  34  35
4  24  25  26  27  28  29
3  18  19  20  21  22  23
2  12  13  14  15  16  17
1   6   7   8   9  10  11
0   0   1   2   3   4   5
```

3.3 Segment 5 - Grouping and data aggregation

[37]:
```
address = './Data/mtcars.csv'

cars = pd.read_csv(address)

cars.columns = ['car_names', 'mpg', 'cyl', 'disp', 'hp', 'drat', 'wt', 'qsec',␣
↪'vs', 'am', 'gear', 'carb']
cars.head()
```

[37]:
```
           car_names   mpg  cyl   disp   hp  drat     wt   qsec  vs  am  gear  \
0          Mazda RX4  21.0    6  160.0  110  3.90  2.620  16.46   0   1     4
1      Mazda RX4 Wag  21.0    6  160.0  110  3.90  2.875  17.02   0   1     4
2         Datsun 710  22.8    4  108.0   93  3.85  2.320  18.61   1   1     4
3     Hornet 4 Drive  21.4    6  258.0  110  3.08  3.215  19.44   1   0     3
4  Hornet Sportabout  18.7    8  360.0  175  3.15  3.440  17.02   0   0     3

   carb
0     4
1     4
2     1
3     1
4     2
```

[38]:
```
cars_groups = cars.groupby(cars['cyl'])
cars_groups.mean()
```

[38]:

cyl	mpg	disp	hp	drat	wt	qsec	vs	am	gear	carb
4	26.663636	105.136364	82.636364	4.070909	2.285727	19.137273	0.909091	0.727273	4.090909	1.545455
6	19.742857	183.314286	122.285714	3.585714	3.117143	17.977143	0.571429	0.428571	3.857143	3.428571
8	15.100000	353.100000	209.214286	3.229286	3.999214	16.772143	0.000000	0.142857	3.285714	3.500000

[39]:
```
cars_groups = cars.groupby(cars['am'])
cars_groups.mean()
```

[39]:

am	mpg	cyl	disp	hp	drat	wt	qsec	vs	gear	carb
0	17.147368	6.947368	290.378947	160.263158	3.286316	3.768895	18.183158	0.368421	3.210526	2.736842
1	24.392308	5.076923	143.530769	126.846154	4.050000	2.411000	17.360000	0.538462	4.384615	2.923077

[ ]:
```

```