import numpy as np import pandas as pd

from pandas import Series, DataFrame

Chapter 2 - Data Preparation Basics Segment 1 - Filtering and selecting data Selecting and retrieving data

In [36]:
```
series_obj = Series(np.arange(8), index=["row 1","row 2","row 3","row 4","row 5",
series_obj
```

Out[36]:
```
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
```

In [37]:
```
series_obj['row 7']
```

Out[37]: 6

In [38]:
```
series_obj[[0, 7]]
```

Out[38]:
```
row 1    0
row 8    7
dtype: int32
```

In [39]:
```
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'
DF_obj
```

Out[39]:

	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

In [40]:
```
DF_obj.loc[['row 2', 'row 5'], ['column 5', 'column 2']]
```

Out[40]:

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

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 use the same square brackets when doing data slicing.

How slicing differs, however, is that with slicing you pass in two index values tha 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 between them.

In [41]:
```
series_obj['row 3': 'row 7']
```

Out[41]:
```
row 3    2
row 4    3
row 5    4
row 6    5
row 7    6
dtype: int32
```

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 bisically just a single numerical value. You can use comparison oprations like greater than or less than to return true/false values for all records to indicate how each element compares to a scalar value.

In [42]:
```
DF_obj < .2
```

Out[42]:

	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

Filtering with scalars

In [43]:
```
series_obj[series_obj > 6]
```

Out[43]:
```
row 8    7
dtype: int32
```

Setting values with scalars

In [44]:
```
series_obj['row 1', 'row 5', 'row 8'] = 8
series_obj
```

Out[44]:
```
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
```

Chapter 2 - Data Preparation Basics Segment 2 - Testing missing values Figuring out what data is missing

In [1]:
```
missing = np.nan
series_obj = Series(['row 1', 'row 2', missing, 'row 4', 'row 5', 'row 6', missir
series_obj
```

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

In [2]:
```
series_obj.isnull()
```

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

Filling in for missing values

In [4]:
```
np.random.seed(25)
DF_obj = DataFrame(np.random.rand(36).reshape(6,6))
DF_obj
```

Out[4]:

	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

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

Out[5]:

	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

In [6]:
```
filled_DF = DF_obj.fillna(0)
filled_DF
```

Out[6]:

	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

In [7]:
```
filled_DF = DF_obj.fillna({0 : 0.1, 5:1.25})
filled_DF
```

Out[7]:

	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

In [8]:
```
fill_DF = DF_obj.fillna(method="ffill")
fill_DF
```

Out[8]:

	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

In [9]:
```
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
```

Out[9]:

	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

Counting missing values

In [10]:
```
DF_obj.isnull().sum()
```

Out[10]:
```
0    3
1    0
2    0
3    0
4    0
5    4
dtype: int64
```

Filtering out missing values

In [16]:
```
DF_no_NaN = DF_obj.dropna()
DF_no_NaN
```

Out[16]:

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

In [17]:
```
DF_no_NaN = DF_obj.dropna(axis=1)
DF_no_NaN
```

Out[17]:

	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

Chapter 2 - Data Preparation Basics Segment 3 - Removing duplicates

In [18]:
```
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
```

Out[18]:

	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

In [19]:
```
DF_obj.duplicated()
```

Out[19]:
```
0    False
1     True
2    False
3     True
4    False
5     True
6     True
dtype: bool
```

In [20]:
```
DF_obj.drop_duplicates()
```

Out[20]:

	column 1	column 2	column 3
0	1	a	A
2	2	b	B
4	3	c	C

In [21]:
```
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','D','C']})
DF_obj
```

Out[21]:

	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	D
6	3	c	C

In [22]:
```
DF_obj.drop_duplicates('column 3')
```

Out[22]:

	column 1	column 2	column 3
0	1	a	A
2	2	b	B
4	3	c	C
5	3	c	D

In [ ]:
```
Chapter 2- Data Preparation Basics
segment4 - Concatenating and transforming data
```

In [23]:
```
DF_obj = pd.DataFrame(np.arange(36).reshape(6,6))
```

In [24]:
```
DF_obj
```

Out[24]:

	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

In [26]:
```
DF_obj_2 = pd.DataFrame(np.arange(15).reshape(5,3))
DF_obj_2
```

Out[26]:

	0	1	2
0	0	1	2
1	3	4	5
2	6	7	8
3	9	10	11
4	12	13	14

Concatenating data

In [27]:
```
pd.concat([DF_obj, DF_obj_2], axis=1)
```

Out[27]:

	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

In [28]:
```
pd.concat([DF_obj, DF_obj_2])
```

Out[28]:

	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

In [ ]:
```
Transforming data
Dropping data
```

In [29]:
```
DF_obj.drop([0,2])
```

Out[29]:

	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

In [30]:
```
DF_obj.drop([0,2], axis=1)
```

Out[30]:

	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

Adding data

In [31]:
```
series_obj = Series(np.arange(6))
series_obj.name = "added_variable"
series_obj
```

Out[31]:
```
0    0
1    1
2    2
3    3
4    4
5    5
Name: added_variable, dtype: int32
```

In [32]:
```
variable_added = DataFrame.join(DF_obj, series_obj)
variable_added
```

Out[32]:

	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

In [33]:
```
added_datatable = variable_added.append(variable_added, ignore_index=False)
added_datatable
```

Out[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
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

Sorting data

In [34]:
```
DF_sorted = DF_obj.sort_values(by=(5), ascending=[False])
DF_sorted
```

Out[34]:

	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

In [ ]:
```

```