Looking_at_Data

July 18, 2024

1 Looking at DataFrame Data

1. Run the cell below to import required libraries and create a DataFrame

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

num_rows = 100
colors = ['Red', 'Blue', 'Green']

df = pd.DataFrame( {'color': [colors[random.randint(0,2)] for _ in␣
↪range(num_rows)],
                    'integers': [random.randint(0,15) for _ in range(num_rows)],
                    'floats': [random.random() for _ in range(num_rows)]})
df
```

[1]:
```
    color  integers    floats
0    Blue         8  0.832620
1     Red        11  0.659854
2     Red        11  0.483620
3    Blue         6  0.780785
4    Blue         2  0.691239
..    …       …        …
95    Red         9  0.572455
96    Red        14  0.949298
97  Green        14  0.209833
98  Green        12  0.187528
99    Red         1  0.824390

[100 rows x 3 columns]
```

2. Use the DataFrame `head()` method to view the top five rows. Try giving it a number as an argument to control how many rows are displayed.

[3]:
```
df.head(3)
```

[3]:
```
  color  integers    floats
0  Blue         8  0.832620
1   Red        11  0.659854
2   Red        11  0.483620
```

3. View summary statistics using the DataFrame `describe()` method.

[4]:
```
df.describe()
```

[4]:
```
         integers      floats
count  100.00000  100.000000
mean     7.21000    0.523836
std      4.42284    0.275100
min      0.00000    0.007333
25%      4.00000    0.320259
50%      7.00000    0.513239
75%     11.00000    0.764430
max     15.00000    0.985157
```

4. The `decribe()` method accepts some optional arguments, including 'include' and 'exclude'. By default, `describe()` only shows statistics for columns with numerical data, but if you add the argument `include=np.object`, it will display statistics for columns with string data. Try this.

[5]:
```
df.describe(include=np.object)
```

[5]:
```
        color
count     100
unique      3
top     Green
freq       38
```

5. If you change the argument to `include='all'`, it will display statistics for all columns in the data frame, inserting `NaN` (not a number) when the data type is not appropriate for the statistic. Try viewing statistics for all frames using `describe()`.

[6]:
```
df.describe(include='all')
```

[6]:
```
        color   integers      floats
count     100  100.00000  100.000000
unique      3        NaN         NaN
top     Green        NaN         NaN
freq       38        NaN         NaN
mean      NaN    7.21000    0.523836
std       NaN    4.42284    0.275100
min       NaN    0.00000    0.007333
25%       NaN    4.00000    0.320259
50%       NaN    7.00000    0.513239
```

```
75%        NaN    11.00000    0.764430
max        NaN    15.00000    0.985157
```

1.1 Selecting Data

6. You can select a column using bracket syntax very similar to that used with dictionaries. Put the column name, as a string, in brackets after the DataFrame name. Try this with the column 'color'

[8]:
```
df['color'].head()
```

[8]:
```
0    Blue
1     Red
2     Red
3    Blue
4    Blue
Name: color, dtype: object
```

7. Try selecting the columns 'color' and 'floats' by supplying them as a list of strings in the same bracket syntax.

[9]:
```
df[['color','floats']].head()
```

[9]:

	color	floats
0	Blue	0.832620
1	Red	0.659854
2	Red	0.483620
3	Blue	0.780785
4	Blue	0.691239

8. The bracket syntax in DataFrames is overloaded to select rows as well. Selecting rows uses the syntax we used to select slices in Sequences: a start number, a colon, and an upper bound number. Try selecting three rows from the DataFrame using the slice `10:13`

[14]:
```
df[10:13]
```

[14]:

	color	integers	floats
10	Blue	0	0.207491
11	Blue	4	0.370300
12	Green	0	0.324745

9. Now let's try the `.loc[]` syntax. It also uses bracket syntax, but in this case you will specify both rows and columns to select. Select all of the rows by supplying a lone colon as the first argument, and the column 'color' by supplying it as a second argument (remember that arguments must be separated by a comma).

[16]:
```
df.loc[:, 'color']
```

[16]:
```
0      Blue
1       Red
2       Red
3      Blue
4      Blue
      …
95      Red
96      Red
97    Green
98    Green
99      Red
Name: color, Length: 100, dtype: object
```

10. Now specify a slice, `10:13`, for the first argument and a list of columns, `['color', 'integers']`, as a second, to select **four** rows (the upper bound in `loc[]` is included) and two columns.

[17]:
```
df.loc[10:13,['color','integers']]
```

[17]:
```
    color  integers
10   Blue         0
11   Blue         4
12  Green         0
13    Red         1
```

11. Now try the `iloc[]` syntax. This used the position of rows and columns to determine selection. In this DataFrame, the labels for the rows are the same as their position, so we can use the same slice `10:13` as the first argument. For the second, use the slice `0:2` to select the first two columns. Notice that with `iloc[]`, the upper bound is not inclusive, so you will get three rows and two columns.

[18]:
```
df.iloc[10:13,0:2]
```

[18]:
```
    color  integers
10   Blue         0
11   Blue         4
12  Green         0
```

[ ]: