Selecting_DataFrame_Data

July 19, 2024

1 Selecting Data

1. Run the cell below to setup this notebook

[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.head()
```

[1]:

	color	integers	floats
0	Blue	5	0.267919
1	Green	13	0.832495
2	Green	8	0.248069
3	Blue	4	0.447578
4	Blue	4	0.764026

2. Create a mask named 'red_mask' with `True` for all rows where the color is equal to 'Red'. Remember that the equals operator is a double equals `==`.

[4]:
```
red_mask = (df.loc[:, 'color']=='Red')
```

3. New create a DataFrame named 'red_df' by using the mask with the data frame `df`. You can use the mask in the simple bracket syntax to filter the DataFrame.

[7]:
```
red_df = df[red_mask]
red_df.head()
```

[7]:

	color	integers	floats
5	Red	14	0.580931
8	Red	0	0.991636

```
10   Red          6  0.099813
11   Red          0  0.257800
12   Red         14  0.927558
```

4. Columns have a method, `.unique()`, which will return all unique values in that column. Call this method on the `red_df.color` to confirm that it only contians 'Red' values.

[11]:
```
red_df.color.unique()
```

[11]:
```
array(['Red'], dtype=object)
```

5. Now use the not operator, `~`, in combination with `red_mask`, to create a new DataFrame named 'no_red'. Simply put `~` in front of the mask to negate it.

[22]:
```
no_red = df[~red_mask]
no_red.head()
```

[22]:
```
   color  integers    floats
0   Blue         5  0.267919
1  Green        13  0.832495
2  Green         8  0.248069
3   Blue         4  0.447578
4   Blue         4  0.764026
```

6. Now check the values of `no_red.color` using the `.unique()` method to confirm that there are no 'Red' values.

[23]:
```
no_red.color.unique()
```

[23]:
```
array(['Blue', 'Green'], dtype=object)
```

7. Create a new mask named 'three_mask' for all rows where `integers <= 3`.

[28]:
```
three_mask = (df.loc[:,'integers'] <= 3)
df[three_mask]
```

[28]:
```
    color  integers    floats
7   Green         0  0.361239
8     Red         0  0.991636
9    Blue         0  0.401136
11    Red         0  0.257800
16   Blue         0  0.879748
19  Green         1  0.293997
34    Red         1  0.056561
37   Blue         0  0.216545
41   Blue         3  0.924795
49    Red         3  0.028746
51    Red         2  0.198725
55  Green         2  0.292459
```

```
62   Blue         2  0.316853
63    Red         3  0.992894
69    Red         3  0.760558
72    Red         3  0.427590
77    Red         1  0.839111
78    Red         3  0.701721
79   Blue         2  0.240882
81   Blue         2  0.417201
82  Green         1  0.272181
91  Green         2  0.560110
92  Green         0  0.623469
93    Red         3  0.465715
94  Green         1  0.586170
```

8. Create a new DataFrame named 'mixed_df' containing only rows whose color is 'Red' and whose integer value is equal or less than 3 by using the 'and' operator, `&` between that masks.

[35]:
```
mixed_df = (df.loc[:,'color'] == 'Red') & (df.loc[:,'integers']<=3)
mixed_df = df[mixed_df]
mixed_df
```

[35]:
```
   color  integers    floats
8    Red         0  0.991636
11   Red         0  0.257800
34   Red         1  0.056561
49   Red         3  0.028746
51   Red         2  0.198725
63   Red         3  0.992894
69   Red         3  0.760558
72   Red         3  0.427590
77   Red         1  0.839111
78   Red         3  0.701721
93   Red         3  0.465715
```

9. Now use `.unique()` to check the values of the 'color' column

[34]:
```
mixed_df.color.unique()
```

[34]:
```
array(['Red'], dtype=object)
```

10. Use `.unique()` to check the values of the 'integer' column, confirming that they are all less than or equal to 3.

[36]:
```
mixed_df.integers.unique()
```

[36]:
```
array([0, 1, 3, 2])
```

[ ]:
```

```