21. Pandas - Part 4

October 28, 2022

[ ]:
```
import pandas as pd
```

1 How to select multiple rows and columns from a DataFrame?

[ ]:
```
# read a dataset of UFO reports into a DataFrame
df = pd.read_csv('../data/ufo.csv')
```

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

1.1 `loc`

- `loc` is used to select rows and columns by **label**.

You can pass it: - A single label - A list of labels - A slice of labels - A boolean Series - A colon (which indicates "all labels")

[ ]:
```
# row 0, column City
df.loc[0,'City']
```

[ ]:
```
# row 0, all columns
df.loc[0, :]
```

[ ]:
```
# rows 0 and 1 and 2, all columns
df.loc[[0, 1, 2], :]
```

[ ]:
```
# rows 0 through 2 (inclusive), all columns
df.loc[0:2, :]
```

[ ]:
```
# this implies "all columns", but explicitly stating "all columns" is better
df.loc[0:2]
```

[ ]:
```
# rows 0 through 2 (inclusive), column 'City'
df.loc[0:2, 'City']
```

[ ]:
```
# rows 0 through 2 (inclusive), columns 'City' and 'State'
df.loc[0:2, ['City', 'State']]
```

```
[ ]: # accomplish the same thing using double brackets - but using 'loc' is preferred
df[['City', 'State']]#.head(3)
```

```
[ ]: df.loc[:,['City', 'State']]
```

```
[ ]: df.head(5)
```

```
[ ]: # rows 0 through 2 (inclusive), columns 'City' through 'State' (inclusive)
df.loc[0:2, 'City':'State']
```

```
[ ]: # accomplish the same thing using 'head' and 'drop'
df.head(3).drop('Time', axis=1)
```

```
[ ]: df.City=='Oakland'
```

```
[ ]: # rows in which the 'City' is 'Oakland', column 'State'
df.loc[df.City=='Oakland', 'State']
```

```
[ ]: # accomplish the same thing using "chained indexing" - but using 'loc' is␣
↪preferred
df[df.City=='Oakland'].State
```

1.2 iloc

- `iloc` is used to select rows and columns by **integer position**.

You can pass it:

- A single integer position
- A list of integer positions
- A slice of integer positions
- A colon (which indicates "all integer positions")

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

```
[ ]: # row 0, column City
df.iloc[0,0]
```

```
[ ]: # rows in positions 0 and 1, columns in positions 0 and 3
df.iloc[[0, 1], [0, 3]]
```

```
[ ]: # rows in positions 0 through 2 (exclusive), columns in positions 0 through 4␣
↪(exclusive)
df.iloc[0:2, 0:4]
```

```
[ ]: # rows in positions 0 through 2 (exclusive), all columns
df.iloc[0:2, :]
```

[ ]:
```
# accomplish the same thing
df[0:2]
```

2 How to explore a Series/Column?

[ ]:
```
# read dataset of top-rated IMDb movies into a DataFrame
movies_df = pd.read_csv('../data/imdb_1000.csv')
```

[ ]:
```
movies_df.head()
```

[ ]:
```
# examine the data type of each Series
movies_df.dtypes
```

2.1 Exploring a non-numeric Series

[ ]:
```
# count the non-null values, unique values, and frequency of the most common␣
↪value
movies_df.genre.describe()
```

[ ]:
```
# count how many times each value in the Series occurs
movies_df.genre.value_counts()
```

[ ]:
```
# display percentages instead of raw counts
movies_df.genre.value_counts(normalize=True)
```

[ ]:
```
# 'value_counts' (like many pandas methods) outputs a Series
type(movies_df.genre.value_counts())
```

[ ]:
```
# thus, you can add another Series method on the end
movies_df.genre.value_counts().head()
```

[ ]:
```
# display the unique values in the Series
movies_df.genre.unique()
```

[ ]:
```
# count the number of unique values in the Series
movies_df.genre.nunique()
```

[ ]:
```
# compute a cross-tabulation of two Series
pd.crosstab(movies_df.genre, movies_df.content_rating)
```

2.2 Exploring a numeric Series

```
[ ]: # calculate various summary statistics
movies_df.duration.describe()
```

```
[ ]: # many statistics are implemented as Series methods
movies_df.duration.mean()
```

```
[ ]: # 'value_counts' is primarily useful for categorical data, not numerical data
movies_df.duration.value_counts().head()
```

```
[ ]: # histogram of the 'duration' Series (shows the distribution of a numerical␣
  ↪variable)
movies_df.duration.plot(kind='hist')
```

```
[ ]: # bar plot of the 'value_counts' for the 'genre' Series
movies_df.genre.value_counts().plot(kind='pie')
```

3 How to handle missing values?

```
[ ]: # read a dataset of UFO reports into a DataFrame
# df = pd.read_csv('data/ufo.csv')
```

```
[ ]: df.tail()
```

What does "NaN" mean?

- "NaN" is not a string, rather it's a special value: **`numpy.nan`**.
- It stands for "Not a Number" and indicates a **missing value**.
- **`read_csv`** detects missing values (by default) when reading the file, and replaces them with this special value.

```
[ ]: # 'isnull' returns a DataFrame of booleans (True if missing, False if not␣
  ↪missing)
df.isnull().tail()
```

```
[ ]: # 'nonnull' returns the opposite of 'isnull' (True if not missing, False if␣
  ↪missing)
df.notnull()
```

```
[ ]: # count the number of missing values in each Series
df.isnull().sum()
```

This calculation works because:

1. The **`sum`** method for a DataFrame operates on **`axis=0`** by default (and thus produces column sums).
2. In order to add boolean values, pandas converts **`True`** to **1** and **`False`** to **0**.

[ ]:
```
df.City.isnull()
```

[ ]:
```
# use the 'isnull' Series method to filter the DataFrame rows
df[df.City.isnull()]
```

How to handle missing values depends on the dataset as well as the nature of your analysis. Here are some options:

[ ]:
```
# examine the number of rows and columns
df.shape
```

[ ]:
```
# if 'any' values are missing in a row, then drop that row
df.dropna(how='any').shape
```

[ ]:
```
# 'inplace' parameter for 'dropna' is False by default, thus rows were only␣
↪dropped temporarily
df.shape
```

[ ]:
```
# if 'all' values are missing in a row, then drop that row (none are dropped in␣
↪this case)
df.dropna(how='all').shape
```

[ ]:
```
# if 'any' values are missing in a row (considering only 'City' and 'Shape␣
↪Reported'), then drop that row
df.dropna(subset=['City', 'Shape Reported'], how='any').shape
```

[ ]:
```
# if 'all' values are missing in a row (considering only 'City' and 'Shape␣
↪Reported'), then drop that row
df.dropna(subset=['City', 'Shape Reported'], how='all') # inplace =True
```

[ ]:
```
# 'value_counts' does not include missing values by default
df['Shape Reported'].value_counts()
```

[ ]:
```
df.isna().sum()
```

[ ]:
```
df['Shape Reported']
```

[ ]:
```
# fill in missing values with a specified value
df['Shape Reported'].fillna(value='VARIOUS', inplace=True)
```

[ ]:
```
# confirm that the missing values were filled in
df['Shape Reported'].value_counts()
```

4 index

[ ]:
```
drinks_df = pd.read_csv('../data/drinks.csv')
```

[ ]:
```
drinks_df.head()
```

[ ]:
```
drinks_df.shape
```

[ ]:
```
# every DataFrame has an index (sometimes called the "row labels")
drinks_df.index
```

[ ]:
```
# column names are also stored in a special "index" object
drinks_df.columns
```

[ ]:
```
# neither the index nor the columns are included in the shape
drinks_df.shape
```

[ ]:
```
# identification: index remains with each row when filtering the DataFrame
drinks_df[drinks_df.continent=='South America']
```

[ ]:
```
# selection: select a portion of the DataFrame using the index
drinks_df.loc[23, 'beer_servings']
```

[ ]:
```
drinks_df
```

[ ]:
```
# set an existing column as the index
drinks_df.set_index('country', inplace=True)
```

[ ]:
```
drinks_df.head()
```

[ ]:
```
# 'country' is now the index
drinks_df.index
```

[ ]:
```
# 'country' is no longer a column
drinks_df.columns
```

[ ]:
```
# 'country' data is no longer part of the DataFrame contents
drinks_df.shape
```

[ ]:
```
# country name can now be used for selection
drinks_df.loc['Brazil', 'beer_servings']
```

[ ]:
```
# index name is optional
drinks_df.index.name = None
```

[ ]:
```
drinks_df.head()
```

[ ]:
```
# restore the index name, and move the index back to a column
drinks_df.index.name = 'country'
```

[ ]:
```
drinks_df.head()
```

[ ]:
```
drinks_df.reset_index(inplace=True)
```

[ ]:
```
drinks_df.head()
```

[ ]:
```
# many DataFrame methods output a DataFrame
drinks_df.describe()
```

[ ]:
```
# you can interact with any DataFrame using its index and columns
drinks_df.describe().loc['25%', 'beer_servings']
```

[ ]:
```
# every DataFrame has an index
drinks_df.index
```

[ ]:
```
# every Series also has an index (which carries over from the DataFrame)
drinks_df.continent.head()
```

What is the index used for?

1. identification
2. selection

[ ]:
```
# create a Series containing the population of
# two countries with country name as index values
people = pd.Series([3000000, 85000], index=['Albania', 'Andorra'],␣
↪name='population')
```

[ ]:
```
people
```