30_days

February 29, 2024

1 Course: pandas in 30 days

1. What is pandas?
2. How do I read a tabular data file into pandas?
3. How do I select a pandas Series from a DataFrame?
4. Why do some pandas commands end with parentheses (and others don't)?
5. How do I rename columns in a pandas DataFrame?
6. How do I remove columns from a pandas DataFrame?
7. How do I sort a pandas DataFrame or a Series?
8. How do I filter rows of a pandas DataFrame by column value?
9. How do I apply multiple filter criteria to a pandas DataFrame?
10. Your pandas questions answered!
11. How do I use the "axis" parameter in pandas?
12. How do I use string methods in pandas?
13. How do I change the data type of a pandas Series?
14. When should I use a "groupby" in pandas?
15. How do I explore a pandas Series?
16. How do I handle missing values in pandas?
17. What do I need to know about the pandas index? (Part 1)
18. What do I need to know about the pandas index? (Part 2)
19. How do I select multiple rows and columns from a pandas DataFrame?
20. When should I use the "inplace" parameter in pandas?
21. How do I make my pandas DataFrame smaller and faster?
22. How do I use pandas with scikit-learn to create Kaggle submissions?
23. More of your pandas questions answered!
24. How do I create dummy variables in pandas?
25. How do I work with dates and times in pandas?
26. How do I find and remove duplicate rows in pandas?
27. How do I avoid a SettingWithCopyWarning in pandas?
28. How do I change display options in pandas?
29. How do I create a pandas DataFrame from another object?
30. How do I apply a function to a pandas Series or DataFrame?

1.1 Day 1: What is pandas?

- pandas documentation
- pandas installation instructions

- Anaconda distribution of Python
- How to use the Jupyter Notebook (Data School video)
- Python Essentials for Data Scientists (Data School course)

[Back to top]

1.2 Day 2: How do I read a tabular data file into pandas?

[1]:
```
# conventional way to import pandas
import pandas as pd
```

[2]:
```
# read a dataset of Chipotle orders directly from a URL and store the results␣
↪in a DataFrame
orders = pd.read_table('http://bit.ly/chiporders')
```

[3]:
```
# examine the first 5 rows
orders.head()
```

[3]:
```
   order_id  quantity                              item_name  \
0         1         1           Chips and Fresh Tomato Salsa
1         1         1                                   Izze
2         1         1                       Nantucket Nectar
3         1         1  Chips and Tomatillo-Green Chili Salsa
4         2         2                           Chicken Bowl

                                choice_description item_price
0                                              NaN     $2.39
1                                      [Clementine]     $3.39
2                                           [Apple]     $3.39
3                                              NaN     $2.39
4  [Tomatillo-Red Chili Salsa (Hot), [Black Beans…    $16.98
```

Documentation for **read_table**

[4]:
```
# read a dataset of movie reviewers (modifying the default parameter values for␣
↪read_table)
user_cols = ['user_id', 'age', 'gender', 'occupation', 'zip_code']
users = pd.read_table('http://bit.ly/movieusers', sep='|', header=None,␣
↪names=user_cols)
```

[5]:
```
# examine the first 5 rows
users.head()
```

[5]:
```
   user_id  age gender  occupation zip_code
0        1   24      M  technician    85711
1        2   53      F       other    94043
2        3   23      M      writer    32067
3        4   24      M  technician    43537
4        5   33      F       other    15213
```

[Back to top]

1.3 Day 3: How do I select a pandas Series from a DataFrame?

[6]:
```
# read a dataset of UFO reports into a DataFrame
ufo = pd.read_table('http://bit.ly/uforeports', sep=',')
```

[7]:
```
# read_csv is equivalent to read_table, except it assumes a comma separator
ufo = pd.read_csv('http://bit.ly/uforeports')
```

Documentation for **read_csv**

[8]:
```
# examine the first 5 rows
ufo.head()
```

[8]:

	City	Colors Reported	Shape Reported	State	Time
0	Ithaca	NaN	TRIANGLE	NY	6/1/1930 22:00
1	Willingboro	NaN	OTHER	NJ	6/30/1930 20:00
2	Holyoke	NaN	OVAL	CO	2/15/1931 14:00
3	Abilene	NaN	DISK	KS	6/1/1931 13:00
4	New York Worlds Fair	NaN	LIGHT	NY	4/18/1933 19:00

[9]:
```
# select the 'City' Series using bracket notation
ufo['City']

# or equivalently, use dot notation
ufo.City
```

[9]:
```
0                    Ithaca
1                Willingboro
2                    Holyoke
3                    Abilene
4        New York Worlds Fair
                 …
18236             Grant Park
18237            Spirit Lake
18238            Eagle River
18239            Eagle River
18240                   Ybor
Name: City, Length: 18241, dtype: object
```

Bracket notation will always work, whereas **dot notation** has limitations:

- Dot notation doesn't work if there are **spaces** in the Series name
- Dot notation doesn't work if the Series has the same name as a **DataFrame method or attribute** (like 'head' or 'shape')
- Dot notation can't be used to define the name of a **new Series** (see below)

Should you use "dot notation" or "bracket notation" with pandas? (Data School blog post)

[10]:
```
# create a new 'Location' Series (must use bracket notation to define the␣
↪Series name)
ufo['Location'] = ufo.City + ', ' + ufo.State
ufo.head()
```

[10]:

	City	Colors Reported	Shape Reported	State	Time	Location
0	Ithaca	NaN	TRIANGLE	NY	6/1/1930 22:00	Ithaca, NY
1	Willingboro	NaN	OTHER	NJ	6/30/1930 20:00	Willingboro, NJ
2	Holyoke	NaN	OVAL	CO	2/15/1931 14:00	Holyoke, CO
3	Abilene	NaN	DISK	KS	6/1/1931 13:00	Abilene, KS
4	New York Worlds Fair	NaN	LIGHT	NY	4/18/1933 19:00	New York Worlds Fair, NY

[Back to top]

1.4 Day 4: Why do some pandas commands end with parentheses (and others don't)?

[11]:
```
# read a dataset of top-rated IMDb movies into a DataFrame
movies = pd.read_csv('http://bit.ly/imdbratings')
```

Methods end with parentheses, while **attributes** don't:

[12]:
```
# example method: show the first 5 rows
movies.head()
```

[12]:

	star_rating	title	content_rating	genre	duration	actors_list
0	9.3	The Shawshank Redemption	R	Crime	142	[u'Tim Robbins', u'Morgan Freeman', u'Bob Gunt…
1	9.2	The Godfather	R	Crime	175	[u'Marlon Brando', u'Al Pacino', u'James Caan']
2	9.1	The Godfather: Part II	R	Crime	200	[u'Al Pacino', u'Robert De Niro', u'Robert Duv…
3	9.0	The Dark Knight	PG-13	Action	152	[u'Christian Bale', u'Heath Ledger', u'Aaron E…
4	8.9	Pulp Fiction	R	Crime	154	[u'John Travolta', u'Uma Thurman', u'Samuel L…

[13]:
```
# example method: calculate summary statistics
movies.describe()
```

[13]:
```
       star_rating    duration
count   979.000000  979.000000
mean      7.889785  120.979571
std       0.336069   26.218010
min       7.400000   64.000000
25%       7.600000  102.000000
50%       7.800000  117.000000
75%       8.100000  134.000000
max       9.300000  242.000000
```

[14]:
```
# example attribute: number of rows and columns
movies.shape
```

[14]:
```
(979, 6)
```

[15]:
```
# example attribute: data type of each column
movies.dtypes
```

[15]:
```
star_rating       float64
title              object
content_rating     object
genre              object
duration            int64
actors_list        object
dtype: object
```

[16]:
```
# use an optional parameter to the describe method to summarize only 'object'␣
↪columns
movies.describe(include=['object'])
```

[16]:
```
          title content_rating  genre  \
count       979            976    979
unique      975             12     16
top     Dracula              R  Drama
freq          2            460    278

                                                actors_list
count                                                   979
unique                                                  969
top     [u'Daniel Radcliffe', u'Emma Watson', u'Rupert…
freq                                                      6
```

Documentation for **describe**

[Back to top]

1.5 Day 5: How do I rename columns in a pandas DataFrame?

[17]:
```
# read a dataset of UFO reports into a DataFrame
ufo = pd.read_csv('http://bit.ly/uforeports')
```

[18]:
```
# examine the column names
ufo.columns
```

[18]:
```
Index(['City', 'Colors Reported', 'Shape Reported', 'State', 'Time'],
dtype='object')
```

[19]:
```
# rename two of the columns by using the 'rename' method
ufo.rename(columns={'Colors Reported':'Colors_Reported', 'Shape Reported':
 ↪'Shape_Reported'}, inplace=True)
ufo.columns
```

[19]:
```
Index(['City', 'Colors_Reported', 'Shape_Reported', 'State', 'Time'],
dtype='object')
```

Documentation for **rename**

[20]:
```
# replace all of the column names by overwriting the 'columns' attribute
ufo_cols = ['city', 'colors reported', 'shape reported', 'state', 'time']
ufo.columns = ufo_cols
ufo.columns
```

[20]:
```
Index(['city', 'colors reported', 'shape reported', 'state', 'time'],
dtype='object')
```

[21]:
```
# replace the column names during the file reading process by using the 'names'␣
 ↪parameter
ufo = pd.read_csv('http://bit.ly/uforeports', header=0, names=ufo_cols)
ufo.columns
```

[21]:
```
Index(['city', 'colors reported', 'shape reported', 'state', 'time'],
dtype='object')
```

Documentation for **read_csv**

[22]:
```
# replace all spaces with underscores in the column names by using the 'str.
 ↪replace' method
ufo.columns = ufo.columns.str.replace(' ', '_')
ufo.columns
```

[22]:
```
Index(['city', 'colors_reported', 'shape_reported', 'state', 'time'],
dtype='object')
```

Documentation for **str.replace**

[Back to top]

1.6 Day 6: How do I remove columns from a pandas DataFrame?

[23]:
```
# read a dataset of UFO reports into a DataFrame
ufo = pd.read_csv('http://bit.ly/uforeports')
ufo.head()
```

[23]:
```
                  City Colors Reported Shape Reported State              Time
0               Ithaca              NaN       TRIANGLE    NY   6/1/1930 22:00
1          Willingboro              NaN          OTHER    NJ  6/30/1930 20:00
2              Holyoke              NaN           OVAL    CO  2/15/1931 14:00
3              Abilene              NaN           DISK    KS   6/1/1931 13:00
4  New York Worlds Fair              NaN          LIGHT    NY  4/18/1933 19:00
```

[24]:
```
# number of rows and columns
ufo.shape
```

[24]:
```
(18241, 5)
```

[25]:
```
# remove a single column (axis=1 refers to columns)
ufo.drop('Colors Reported', axis=1, inplace=True)
ufo.head()
```

[25]:
```
                  City Shape Reported State              Time
0               Ithaca       TRIANGLE    NY   6/1/1930 22:00
1          Willingboro          OTHER    NJ  6/30/1930 20:00
2              Holyoke           OVAL    CO  2/15/1931 14:00
3              Abilene           DISK    KS   6/1/1931 13:00
4  New York Worlds Fair          LIGHT    NY  4/18/1933 19:00
```

Documentation for **drop**

[26]:
```
# remove multiple columns at once
ufo.drop(['City', 'State'], axis=1, inplace=True)
ufo.head()
```

[26]:
```
  Shape Reported              Time
0       TRIANGLE   6/1/1930 22:00
1          OTHER  6/30/1930 20:00
2           OVAL  2/15/1931 14:00
3           DISK   6/1/1931 13:00
4          LIGHT  4/18/1933 19:00
```

[27]:
```
# remove multiple rows at once (axis=0 refers to rows)
ufo.drop([0, 1], axis=0, inplace=True)
ufo.head()
```

[27]:
```
  Shape Reported              Time
2           OVAL  2/15/1931 14:00
3           DISK   6/1/1931 13:00
```

```
4          LIGHT  4/18/1933 19:00
5           DISK  9/15/1934 15:30
6         CIRCLE   6/15/1935 0:00
```

[28]:
```
# two rows and three columns have been removed
ufo.shape
```

[28]:
```
(18239, 2)
```

[Back to top]

1.7 Day 7: How do I sort a pandas DataFrame or a Series?

[29]:
```
# read a dataset of top-rated IMDb movies into a DataFrame
movies = pd.read_csv('http://bit.ly/imdbratings')
movies.head()
```

[29]:
```
   star_rating                     title content_rating   genre  duration  \
0          9.3  The Shawshank Redemption              R   Crime       142
1          9.2             The Godfather              R   Crime       175
2          9.1    The Godfather: Part II              R   Crime       200
3          9.0           The Dark Knight          PG-13  Action       152
4          8.9              Pulp Fiction              R   Crime       154

                                        actors_list
0  [u'Tim Robbins', u'Morgan Freeman', u'Bob Gunt…
1    [u'Marlon Brando', u'Al Pacino', u'James Caan']
2  [u'Al Pacino', u'Robert De Niro', u'Robert Duv…
3  [u'Christian Bale', u'Heath Ledger', u'Aaron E…
4  [u'John Travolta', u'Uma Thurman', u'Samuel L…
```

Note: None of the sorting methods below affect the underlying data. (In other words, the sorting is temporary).

[30]:
```
# sort the 'title' Series in ascending order (returns a Series)
movies.title.sort_values()
```

[30]:
```
542     (500) Days of Summer
5               12 Angry Men
201         12 Years a Slave
698               127 Hours
110    2001: A Space Odyssey
               …
955         Zero Dark Thirty
677                   Zodiac
615              Zombieland
526                    Zulu
864                   [Rec]
```

```
Name: title, Length: 979, dtype: object
```

[31]:
```
# sort in descending order instead
movies.title.sort_values(ascending=False)
```

[31]:
```
864                    [Rec]
526                     Zulu
615               Zombieland
677                   Zodiac
955           Zero Dark Thirty
                 …
110      2001: A Space Odyssey
698                  127 Hours
201           12 Years a Slave
5                  12 Angry Men
542       (500) Days of Summer
Name: title, Length: 979, dtype: object
```

Documentation for **sort_values** for a **Series**.

[32]:
```
# sort the entire DataFrame by the 'title' Series (returns a DataFrame)
movies.sort_values('title')
```

[32]:
```
     star_rating                  title content_rating      genre  duration  \
542          7.8   (500) Days of Summer          PG-13     Comedy        95
5            8.9          12 Angry Men      NOT RATED      Drama        96
201          8.1       12 Years a Slave              R  Biography       134
698          7.6             127 Hours              R  Adventure        94
110          8.3  2001: A Space Odyssey              G    Mystery       160
..           …                    …            …        …       …
955          7.4       Zero Dark Thirty              R      Drama       157
677          7.7                 Zodiac              R      Crime       157
615          7.7             Zombieland              R     Comedy        88
526          7.8                  Zulu        UNRATED      Drama       138
864          7.5                  [Rec]              R     Horror        78

                                          actors_list
542  [u'Zooey Deschanel', u'Joseph Gordon-Levitt', …
5    [u'Henry Fonda', u'Lee J. Cobb', u'Martin Bals…
201  [u'Chiwetel Ejiofor', u'Michael Kenneth Willia…
698  [u'James Franco', u'Amber Tamblyn', u'Kate Mara']
110  [u'Keir Dullea', u'Gary Lockwood', u'William S…
..                                                 …
955  [u'Jessica Chastain', u'Joel Edgerton', u'Chri…
677  [u'Jake Gyllenhaal', u'Robert Downey Jr.', u'M…
615  [u'Jesse Eisenberg', u'Emma Stone', u'Woody Ha…
526  [u'Stanley Baker', u'Jack Hawkins', u'Ulla Jac…
864  [u'Manuela Velasco', u'Ferran Terraza', u'Jorg…
```

```
[979 rows x 6 columns]
```

[33]:
```
# sort in descending order instead
movies.sort_values('title', ascending=False)
```

[33]:
```
     star_rating                  title content_rating      genre  duration  \
864          7.5                  [Rec]              R     Horror        78
526          7.8                   Zulu        UNRATED      Drama       138
615          7.7             Zombieland              R     Comedy        88
677          7.7                 Zodiac              R      Crime       157
955          7.4      Zero Dark Thirty              R      Drama       157
..           …                        …            …        …         …
110          8.3  2001: A Space Odyssey              G    Mystery       160
698          7.6             127 Hours              R  Adventure        94
201          8.1      12 Years a Slave              R  Biography       134
5            8.9          12 Angry Men      NOT RATED      Drama        96
542          7.8   (500) Days of Summer          PG-13     Comedy        95

                                          actors_list
864  [u'Manuela Velasco', u'Ferran Terraza', u'Jorg…
526  [u'Stanley Baker', u'Jack Hawkins', u'Ulla Jac…
615  [u'Jesse Eisenberg', u'Emma Stone', u'Woody Ha…
677  [u'Jake Gyllenhaal', u'Robert Downey Jr.', u'M…
955  [u'Jessica Chastain', u'Joel Edgerton', u'Chri…
..                                               …
110  [u'Keir Dullea', u'Gary Lockwood', u'William S…
698  [u'James Franco', u'Amber Tamblyn', u'Kate Mara']
201  [u'Chiwetel Ejiofor', u'Michael Kenneth Willia…
5    [u'Henry Fonda', u'Lee J. Cobb', u'Martin Bals…
542  [u'Zooey Deschanel', u'Joseph Gordon-Levitt', …

[979 rows x 6 columns]
```

Documentation for **sort_values** for a **DataFrame**.

[34]:
```
# sort the DataFrame first by 'content_rating', then by 'duration'
movies.sort_values(['content_rating', 'duration'])
```

[34]:
```
     star_rating                             title content_rating  \
713          7.6                The Jungle Book       APPROVED
513          7.8  Invasion of the Body Snatchers       APPROVED
272          8.1                     The Killing       APPROVED
703          7.6                         Dracula       APPROVED
612          7.7              A Hard Day's Night       APPROVED
..           …                                …            …
387          8.0                Midnight Cowboy              X
86           8.4              A Clockwork Orange              X
```

```
187          8.2  Butch Cassidy and the Sundance Kid            NaN
936          7.4                            True Grit            NaN
649          7.7                    Where Eagles Dare            NaN

         genre  duration                                        actors_list
713  Animation        78  [u'Phil Harris', u'Sebastian Cabot', u'Louis P…
513     Horror        80  [u'Kevin McCarthy', u'Dana Wynter', u'Larry Ga…
272      Crime        85  [u'Sterling Hayden', u'Coleen Gray', u'Vince E…
703     Horror        85  [u'Bela Lugosi', u'Helen Chandler', u'David Ma…
612     Comedy        87  [u'John Lennon', u'Paul McCartney', u'George H…
..         …       …                                                 …
387      Drama       113  [u'Dustin Hoffman', u'Jon Voight', u'Sylvia Mi…
86       Crime       136  [u'Malcolm McDowell', u'Patrick Magee', u'Mich…
187  Biography       110  [u'Paul Newman', u'Robert Redford', u'Katharin…
936  Adventure       128    [u'John Wayne', u'Kim Darby', u'Glen Campbell']
649     Action       158  [u'Richard Burton', u'Clint Eastwood', u'Mary …

[979 rows x 6 columns]
```

[Back to top]

1.8 Day 8: How do I filter rows of a pandas DataFrame by column value?

[35]:
```
# read a dataset of top-rated IMDb movies into a DataFrame
movies = pd.read_csv('http://bit.ly/imdbratings')
movies.head()
```

[35]:
```
   star_rating                     title content_rating   genre  duration  \
0          9.3  The Shawshank Redemption              R   Crime       142
1          9.2             The Godfather              R   Crime       175
2          9.1    The Godfather: Part II              R   Crime       200
3          9.0           The Dark Knight          PG-13  Action       152
4          8.9              Pulp Fiction              R   Crime       154

                                        actors_list
0  [u'Tim Robbins', u'Morgan Freeman', u'Bob Gunt…
1    [u'Marlon Brando', u'Al Pacino', u'James Caan']
2  [u'Al Pacino', u'Robert De Niro', u'Robert Duv…
3  [u'Christian Bale', u'Heath Ledger', u'Aaron E…
4  [u'John Travolta', u'Uma Thurman', u'Samuel L…
```

[36]:
```
# examine the number of rows and columns
movies.shape
```

[36]:
```
(979, 6)
```

Goal: Filter the DataFrame rows to only show movies with a 'duration' of at least 200 minutes.

[37]:
```
# create a list in which each element refers to a DataFrame row: True if the␣
↪row satisfies the condition, False otherwise
booleans = []
for length in movies.duration:
    if length >= 200:
        booleans.append(True)
    else:
        booleans.append(False)
```

[38]:
```
# examine the first five list elements
booleans[0:5]
```

[38]:
```
[False, False, True, False, False]
```

[39]:
```
# confirm that the list has the same length as the DataFrame
len(booleans)
```

[39]:
```
979
```

[40]:
```
# convert the list to a Series
is_long = pd.Series(booleans)
is_long.head()
```

[40]:
```
0    False
1    False
2     True
3    False
4    False
dtype: bool
```

Documentation for **Series** constructor

[41]:
```
# use bracket notation with the boolean Series to tell the DataFrame which rows␣
↪to display
movies[is_long]
```

[41]:

	star_rating	title
2	9.1	The Godfather: Part II
7	8.9	The Lord of the Rings: The Return of the King
17	8.7	Seven Samurai
78	8.4	Once Upon a Time in America
85	8.4	Lawrence of Arabia
142	8.3	Lagaan: Once Upon a Time in India
157	8.2	Gone with the Wind
204	8.1	Ben-Hur
445	7.9	The Ten Commandments
476	7.8	Hamlet
630	7.7	Malcolm X

```
767          7.6                It's a Mad, Mad, Mad, Mad World

    content_rating      genre  duration  \
2                R      Crime       200
7            PG-13  Adventure       201
17         UNRATED      Drama       207
78               R      Crime       229
85              PG  Adventure       216
142             PG  Adventure       224
157              G      Drama       238
204              G  Adventure       212
445       APPROVED  Adventure       220
476          PG-13      Drama       242
630          PG-13  Biography       202
767       APPROVED     Action       205

                                          actors_list
2    [u'Al Pacino', u'Robert De Niro', u'Robert Duv…
7    [u'Elijah Wood', u'Viggo Mortensen', u'Ian McK…
17   [u'Toshir\xf4 Mifune', u'Takashi Shimura', u'K…
78   [u'Robert De Niro', u'James Woods', u'Elizabet…
85   [u"Peter O'Toole", u'Alec Guinness', u'Anthony…
142  [u'Aamir Khan', u'Gracy Singh', u'Rachel Shell…
157  [u'Clark Gable', u'Vivien Leigh', u'Thomas Mit…
204  [u'Charlton Heston', u'Jack Hawkins', u'Stephe…
445  [u'Charlton Heston', u'Yul Brynner', u'Anne Ba…
476  [u'Kenneth Branagh', u'Julie Christie', u'Dere…
630  [u'Denzel Washington', u'Angela Bassett', u'De…
767  [u'Spencer Tracy', u'Milton Berle', u'Ethel Me…
```

[42]:
```
# simplify the steps above: no need to write a for loop to create 'is_long'␣
 ↪since pandas will broadcast the comparison
is_long = movies.duration >= 200
movies[is_long]

# or equivalently, write it in one line (no need to create the 'is_long' object)
movies[movies.duration >= 200]
```

[42]:
```
     star_rating                                         title  \
2            9.1                       The Godfather: Part II
7            8.9  The Lord of the Rings: The Return of the King
17           8.7                                Seven Samurai
78           8.4                  Once Upon a Time in America
85           8.4                            Lawrence of Arabia
142          8.3          Lagaan: Once Upon a Time in India
157          8.2                            Gone with the Wind
204          8.1                                      Ben-Hur
```

```
445          7.9                          The Ten Commandments
476          7.8                                        Hamlet
630          7.7                                     Malcolm X
767          7.6               It's a Mad, Mad, Mad, Mad World

    content_rating      genre  duration  \
2                R      Crime       200
7            PG-13  Adventure       201
17         UNRATED      Drama       207
78               R      Crime       229
85              PG  Adventure       216
142             PG  Adventure       224
157              G      Drama       238
204              G  Adventure       212
445       APPROVED  Adventure       220
476          PG-13      Drama       242
630          PG-13  Biography       202
767       APPROVED     Action       205

                                           actors_list
2    [u'Al Pacino', u'Robert De Niro', u'Robert Duv…
7    [u'Elijah Wood', u'Viggo Mortensen', u'Ian McK…
17   [u'Toshir\xf4 Mifune', u'Takashi Shimura', u'K…
78   [u'Robert De Niro', u'James Woods', u'Elizabet…
85   [u"Peter O'Toole", u'Alec Guinness', u'Anthony…
142  [u'Aamir Khan', u'Gracy Singh', u'Rachel Shell…
157  [u'Clark Gable', u'Vivien Leigh', u'Thomas Mit…
204  [u'Charlton Heston', u'Jack Hawkins', u'Stephe…
445  [u'Charlton Heston', u'Yul Brynner', u'Anne Ba…
476  [u'Kenneth Branagh', u'Julie Christie', u'Dere…
630  [u'Denzel Washington', u'Angela Bassett', u'De…
767  [u'Spencer Tracy', u'Milton Berle', u'Ethel Me…
```

[43]:
```
# select the 'genre' Series from the filtered DataFrame
movies[movies.duration >= 200].genre

# or equivalently, use the 'loc' accessor
movies.loc[movies.duration >= 200, 'genre']
```

[43]:
```
2          Crime
7      Adventure
17         Drama
78         Crime
85     Adventure
142    Adventure
157        Drama
204    Adventure
```

```
445    Adventure
476        Drama
630    Biography
767       Action
Name: genre, dtype: object
```

Documentation for `loc`

[Back to top]

1.9 Day 9: How do I apply multiple filter criteria to a pandas DataFrame?

[44]:
```
# read a dataset of top-rated IMDb movies into a DataFrame
movies = pd.read_csv('http://bit.ly/imdbratings')
movies.head()
```

[44]:
```
   star_rating                     title content_rating   genre  duration  \
0          9.3  The Shawshank Redemption              R   Crime       142
1          9.2             The Godfather              R   Crime       175
2          9.1    The Godfather: Part II              R   Crime       200
3          9.0           The Dark Knight          PG-13  Action       152
4          8.9              Pulp Fiction              R   Crime       154

                                        actors_list
0  [u'Tim Robbins', u'Morgan Freeman', u'Bob Gunt…
1    [u'Marlon Brando', u'Al Pacino', u'James Caan']
2  [u'Al Pacino', u'Robert De Niro', u'Robert Duv…
3  [u'Christian Bale', u'Heath Ledger', u'Aaron E…
4  [u'John Travolta', u'Uma Thurman', u'Samuel L…
```

[45]:
```
# filter the DataFrame to only show movies with a 'duration' of at least 200 minutes
movies[movies.duration >= 200]
```

[45]:
```
     star_rating                                      title  \
2            9.1                   The Godfather: Part II
7            8.9  The Lord of the Rings: The Return of the King
17           8.7                               Seven Samurai
78           8.4                Once Upon a Time in America
85           8.4                             Lawrence of Arabia
142          8.3          Lagaan: Once Upon a Time in India
157          8.2                             Gone with the Wind
204          8.1                                      Ben-Hur
445          7.9                        The Ten Commandments
476          7.8                                      Hamlet
630          7.7                                   Malcolm X
767          7.6            It's a Mad, Mad, Mad, Mad World
```

```
    content_rating      genre  duration  \
2                R      Crime       200
7            PG-13  Adventure       201
17         UNRATED      Drama       207
78               R      Crime       229
85              PG  Adventure       216
142             PG  Adventure       224
157              G      Drama       238
204              G  Adventure       212
445       APPROVED  Adventure       220
476          PG-13      Drama       242
630          PG-13  Biography       202
767       APPROVED     Action       205

                                           actors_list
2    [u'Al Pacino', u'Robert De Niro', u'Robert Duv…
7    [u'Elijah Wood', u'Viggo Mortensen', u'Ian McK…
17   [u'Toshir\xf4 Mifune', u'Takashi Shimura', u'K…
78   [u'Robert De Niro', u'James Woods', u'Elizabet…
85   [u"Peter O'Toole", u'Alec Guinness', u'Anthony…
142  [u'Aamir Khan', u'Gracy Singh', u'Rachel Shell…
157  [u'Clark Gable', u'Vivien Leigh', u'Thomas Mit…
204  [u'Charlton Heston', u'Jack Hawkins', u'Stephe…
445  [u'Charlton Heston', u'Yul Brynner', u'Anne Ba…
476  [u'Kenneth Branagh', u'Julie Christie', u'Dere…
630  [u'Denzel Washington', u'Angela Bassett', u'De…
767  [u'Spencer Tracy', u'Milton Berle', u'Ethel Me…
```

Understanding **logical operators:**

- **or**: True if **either side** of the operator is True
- **and**: True only if **both sides** of the operator are True

[46]:
```
# demonstration of the 'or' operator
print(True or True)
print(True or False)
print(False or False)
```

```
True
True
False
```

[47]:
```
# demonstration of the 'and' operator
print(True and True)
print(True and False)
print(False and False)
```

```
True
False
```

```
False
```

Rules for specifying **multiple filter criteria** in pandas:

- use `|` instead of **`or`**
- use `&` instead of **`and`**
- add **parentheses** around each condition to specify evaluation order

Goal: Further filter the DataFrame of long movies (duration >= 200) to only show movies which also have a 'genre' of 'Drama'

[48]:
```
# CORRECT: use the '&' operator to specify that both conditions are required
movies[(movies.duration >=200) & (movies.genre == 'Drama')]
```

[48]:
```
     star_rating              title content_rating  genre  duration  \
17           8.7       Seven Samurai        UNRATED  Drama       207
157          8.2  Gone with the Wind              G  Drama       238
476          7.8              Hamlet          PG-13  Drama       242

                                        actors_list
17   [u'Toshir\xf4 Mifune', u'Takashi Shimura', u'K…
157  [u'Clark Gable', u'Vivien Leigh', u'Thomas Mit…
476  [u'Kenneth Branagh', u'Julie Christie', u'Dere…
```

[49]:
```
# INCORRECT: using the '|' operator would have shown movies that are either␣
↪long or dramas (or both)
movies[(movies.duration >=200) | (movies.genre == 'Drama')]
```

[49]:
```
     star_rating                                          title  \
2            9.1                       The Godfather: Part II
5            8.9                                 12 Angry Men
7            8.9  The Lord of the Rings: The Return of the King
9            8.9                                   Fight Club
13           8.8                                 Forrest Gump
..           …                                              …
958          7.4                         My Sister's Keeper
968          7.4                       The English Patient
970          7.4                              Wonder Boys
972          7.4                           Blue Valentine
973          7.4                    The Cider House Rules

    content_rating      genre  duration  \
2                R      Crime       200
5        NOT RATED      Drama        96
7            PG-13  Adventure       201
9                R      Drama       139
13           PG-13      Drama       142
..             …        …       …
958          PG-13      Drama       109
```

```
968              R      Drama       162
970              R      Drama       107
972          NC-17      Drama       112
973          PG-13      Drama       126

                                           actors_list
2    [u'Al Pacino', u'Robert De Niro', u'Robert Duv…
5    [u'Henry Fonda', u'Lee J. Cobb', u'Martin Bals…
7    [u'Elijah Wood', u'Viggo Mortensen', u'Ian McK…
9    [u'Brad Pitt', u'Edward Norton', u'Helena Bonh…
13     [u'Tom Hanks', u'Robin Wright', u'Gary Sinise']
..                                                 …
958  [u'Cameron Diaz', u'Abigail Breslin', u'Alec B…
968  [u'Ralph Fiennes', u'Juliette Binoche', u'Will…
970  [u'Michael Douglas', u'Tobey Maguire', u'Franc…
972  [u'Ryan Gosling', u'Michelle Williams', u'John…
973  [u'Tobey Maguire', u'Charlize Theron', u'Micha…

[287 rows x 6 columns]
```

Goal: Filter the original DataFrame to show movies with a 'genre' of 'Crime' or 'Drama' or 'Action'

[50]:
```
# use the '|' operator to specify that a row can match any of the three criteria
movies[(movies.genre == 'Crime') | (movies.genre == 'Drama') | (movies.genre ==␣
 ↪'Action')]

# or equivalently, use the 'isin' method
movies[movies.genre.isin(['Crime', 'Drama', 'Action'])]
```

[50]:
```
     star_rating                                              title  \
0            9.3                           The Shawshank Redemption
1            9.2                                      The Godfather
2            9.1                             The Godfather: Part II
3            9.0                                    The Dark Knight
4            8.9                                       Pulp Fiction
..           …                                                  …
970          7.4                                       Wonder Boys
972          7.4                                    Blue Valentine
973          7.4                                The Cider House Rules
976          7.4  Master and Commander: The Far Side of the World
978          7.4                                       Wall Street

    content_rating   genre  duration  \
0                R   Crime       142
1                R   Crime       175
2                R   Crime       200
3            PG-13  Action       152
4                R   Crime       154
```

```
..              …      …       …
970             R   Drama       107
972         NC-17   Drama       112
973         PG-13   Drama       126
976         PG-13  Action       138
978             R   Crime       126

                                                actors_list
0    [u'Tim Robbins', u'Morgan Freeman', u'Bob Gunt…
1      [u'Marlon Brando', u'Al Pacino', u'James Caan']
2    [u'Al Pacino', u'Robert De Niro', u'Robert Duv…
3    [u'Christian Bale', u'Heath Ledger', u'Aaron E…
4    [u'John Travolta', u'Uma Thurman', u'Samuel L…
..                                                 …
970  [u'Michael Douglas', u'Tobey Maguire', u'Franc…
972  [u'Ryan Gosling', u'Michelle Williams', u'John…
973  [u'Tobey Maguire', u'Charlize Theron', u'Micha…
976  [u'Russell Crowe', u'Paul Bettany', u'Billy Bo…
978  [u'Charlie Sheen', u'Michael Douglas', u'Tamar…

[538 rows x 6 columns]
```

Documentation for **isin**

[Back to top]

1.10 Day 10: Your pandas questions answered!

Question: When reading from a file, how do I read in only a subset of the columns?

[51]:
```
# read a dataset of UFO reports into a DataFrame, and check the columns
ufo = pd.read_csv('http://bit.ly/uforeports')
ufo.columns
```

[51]:
```
Index(['City', 'Colors Reported', 'Shape Reported', 'State', 'Time'],
dtype='object')
```

[52]:
```
# specify which columns to include by name
ufo = pd.read_csv('http://bit.ly/uforeports', usecols=['City', 'State'])

# or equivalently, specify columns by position
ufo = pd.read_csv('http://bit.ly/uforeports', usecols=[0, 3])
ufo.columns
```

[52]:
```
Index(['City', 'State'], dtype='object')
```

Question: When reading from a file, how do I read in only a subset of the rows?

[53]:
```
# specify how many rows to read
ufo = pd.read_csv('http://bit.ly/uforeports', nrows=3)
ufo
```

[53]:

	City	Colors Reported	Shape Reported	State	Time
0	Ithaca	NaN	TRIANGLE	NY	6/1/1930 22:00
1	Willingboro	NaN	OTHER	NJ	6/30/1930 20:00
2	Holyoke	NaN	OVAL	CO	2/15/1931 14:00

Documentation for **read_csv**

Question: How do I iterate through a Series?

[54]:
```
# Series are directly iterable (like a list)
for c in ufo.City:
    print(c)
```

```
Ithaca
Willingboro
Holyoke
```

Question: How do I iterate through a DataFrame?

[55]:
```
# various methods are available to iterate through a DataFrame
for index, row in ufo.iterrows():
    print(index, row.City, row.State)
```

```
0 Ithaca NY
1 Willingboro NJ
2 Holyoke CO
```

Documentation for **iterrows**

Question: How do I drop all non-numeric columns from a DataFrame?

[56]:
```
# read a dataset of alcohol consumption into a DataFrame, and check the data␣
↪types
drinks = pd.read_csv('http://bit.ly/drinksbycountry')
drinks.dtypes
```

[56]:
```
country                          object
beer_servings                     int64
spirit_servings                   int64
wine_servings                     int64
total_litres_of_pure_alcohol    float64
continent                        object
dtype: object
```

[57]:
```
# only include numeric columns in the DataFrame
drinks.select_dtypes(include='number').dtypes
```

[57]:
```
beer_servings                    int64
spirit_servings                  int64
wine_servings                    int64
total_litres_of_pure_alcohol   float64
dtype: object
```

Documentation for **select_dtypes**

Question: How do I know whether I should pass an argument as a string or a list?

[58]:
```
# describe all of the numeric columns
drinks.describe()
```

[58]:
```
       beer_servings  spirit_servings  wine_servings  \
count     193.000000       193.000000     193.000000
mean      106.160622        80.994819      49.450777
std       101.143103        88.284312      79.697598
min         0.000000         0.000000       0.000000
25%        20.000000         4.000000       1.000000
50%        76.000000        56.000000       8.000000
75%       188.000000       128.000000      59.000000
max       376.000000       438.000000     370.000000

       total_litres_of_pure_alcohol
count                    193.000000
mean                       4.717098
std                        3.773298
min                        0.000000
25%                        1.300000
50%                        4.200000
75%                        7.200000
max                       14.400000
```

[59]:
```
# pass the string 'all' to describe all columns
drinks.describe(include='all')
```

[59]:
```
            country  beer_servings  spirit_servings  wine_servings  \
count           193     193.000000       193.000000     193.000000
unique          193            NaN              NaN            NaN
top     Afghanistan            NaN              NaN            NaN
freq              1            NaN              NaN            NaN
mean            NaN     106.160622        80.994819      49.450777
std             NaN     101.143103        88.284312      79.697598
min             NaN       0.000000         0.000000       0.000000
25%             NaN      20.000000         4.000000       1.000000
50%             NaN      76.000000        56.000000       8.000000
75%             NaN     188.000000       128.000000      59.000000
max             NaN     376.000000       438.000000     370.000000
```

```
        total_litres_of_pure_alcohol continent
count                     193.000000       193
unique                           NaN         6
top                              NaN    Africa
freq                             NaN        53
mean                        4.717098       NaN
std                         3.773298       NaN
min                         0.000000       NaN
25%                         1.300000       NaN
50%                         4.200000       NaN
75%                         7.200000       NaN
max                        14.400000       NaN
```

[60]:
```
# pass a list of data types to only describe certain types
drinks.describe(include=['object', 'float64'])
```

[60]:
```
             country  total_litres_of_pure_alcohol continent
count            193                    193.000000       193
unique           193                           NaN         6
top     Afghanistan                           NaN    Africa
freq               1                           NaN        53
mean             NaN                      4.717098       NaN
std              NaN                      3.773298       NaN
min              NaN                      0.000000       NaN
25%              NaN                      1.300000       NaN
50%              NaN                      4.200000       NaN
75%              NaN                      7.200000       NaN
max              NaN                     14.400000       NaN
```

[61]:
```
# pass a list even if you only want to describe a single data type
drinks.describe(include=['object'])
```

[61]:
```
             country continent
count            193       193
unique           193         6
top     Afghanistan    Africa
freq               1        53
```

Documentation for **describe**

[Back to top]

1.11 Day 11: How do I use the "axis" parameter in pandas?

[62]:
```
# read a dataset of alcohol consumption into a DataFrame
drinks = pd.read_csv('http://bit.ly/drinksbycountry')
drinks.head()
```

[62]:
```
       country  beer_servings  spirit_servings  wine_servings  \
0  Afghanistan              0                0              0
1      Albania             89              132             54
2      Algeria             25                0             14
3      Andorra            245              138            312
4       Angola            217               57             45

   total_litres_of_pure_alcohol continent
0                           0.0      Asia
1                           4.9    Europe
2                           0.7    Africa
3                          12.4    Europe
4                           5.9    Africa
```

[63]:
```
# drop a column (temporarily)
drinks.drop('continent', axis=1).head()
```

[63]:
```
       country  beer_servings  spirit_servings  wine_servings  \
0  Afghanistan              0                0              0
1      Albania             89              132             54
2      Algeria             25                0             14
3      Andorra            245              138            312
4       Angola            217               57             45

   total_litres_of_pure_alcohol
0                           0.0
1                           4.9
2                           0.7
3                          12.4
4                           5.9
```

Documentation for **drop**

[64]:
```
# drop a row (temporarily)
drinks.drop(2, axis=0).head()
```

[64]:
```
             country  beer_servings  spirit_servings  wine_servings  \
0        Afghanistan              0                0              0
1            Albania             89              132             54
3            Andorra            245              138            312
4             Angola            217               57             45
5  Antigua & Barbuda            102              128             45

   total_litres_of_pure_alcohol      continent
0                           0.0           Asia
1                           4.9         Europe
3                          12.4         Europe
4                           5.9         Africa
```

```
5                          4.9  North America
```

When **referring to rows or columns** with the axis parameter:

- **axis 0** refers to rows
- **axis 1** refers to columns

[65]:
```
# calculate the mean of each numeric column
drinks.mean(numeric_only=True)

# or equivalently, specify the axis explicitly
drinks.mean(numeric_only=True, axis=0)
```

[65]:
```
beer_servings                 106.160622
spirit_servings                80.994819
wine_servings                  49.450777
total_litres_of_pure_alcohol    4.717098
dtype: float64
```

Documentation for **mean**

[66]:
```
# calculate the mean of each row
drinks.mean(numeric_only=True, axis=1)
```

[66]:
```
0        0.000
1       69.975
2        9.925
3      176.850
4       81.225
        …
188    110.925
189     29.000
190      1.525
191     14.375
192     22.675
Length: 193, dtype: float64
```

When performing a **mathematical operation** with the axis parameter:

- **axis 0** means the operation should "move down" (or "aggregate along") the row axis
- **axis 1** means the operation should "move across" (or "aggregate along") the columns axis

[67]:
```
# 'index' is an alias for axis 0
drinks.mean(numeric_only=True, axis='index')
```

[67]:
```
beer_servings                 106.160622
spirit_servings                80.994819
wine_servings                  49.450777
total_litres_of_pure_alcohol    4.717098
dtype: float64
```

[68]:
```
# 'columns' is an alias for axis 1
drinks.mean(numeric_only=True, axis='columns')
```

[68]:
```
0        0.000
1       69.975
2        9.925
3      176.850
4       81.225
        ...
188    110.925
189     29.000
190      1.525
191     14.375
192     22.675
Length: 193, dtype: float64
```

[Back to top]

1.12 Day 12: How do I use string methods in pandas?

[69]:
```
# read a dataset of Chipotle orders into a DataFrame
orders = pd.read_table('http://bit.ly/chiporders')
orders.head()
```

[69]:
```
   order_id  quantity                              item_name  \
0         1         1           Chips and Fresh Tomato Salsa
1         1         1                                   Izze
2         1         1                       Nantucket Nectar
3         1         1  Chips and Tomatillo-Green Chili Salsa
4         2         2                           Chicken Bowl

                                choice_description item_price
0                                              NaN     $2.39
1                                      [Clementine]     $3.39
2                                           [Apple]     $3.39
3                                              NaN     $2.39
4  [Tomatillo-Red Chili Salsa (Hot), [Black Beans…    $16.98
```

[70]:
```
# normal way to access string methods in Python
'hello'.upper()
```

[70]:
```
'HELLO'
```

[71]:
```
# string methods for pandas Series are accessed via 'str'
orders.item_name.str.upper()
```

[71]:
```
0               CHIPS AND FRESH TOMATO SALSA
1                                       IZZE
```

```
2                            NANTUCKET NECTAR
3       CHIPS AND TOMATILLO-GREEN CHILI SALSA
4                                CHICKEN BOWL
                        …
4617                          STEAK BURRITO
4618                          STEAK BURRITO
4619                     CHICKEN SALAD BOWL
4620                     CHICKEN SALAD BOWL
4621                     CHICKEN SALAD BOWL
Name: item_name, Length: 4622, dtype: object
```

Documentation for **str.upper**

[72]:
```
# string method 'contains' checks for a substring and returns a boolean Series
orders.item_name.str.contains('Chicken')
```

[72]:
```
0       False
1       False
2       False
3       False
4        True
        …
4617    False
4618    False
4619     True
4620     True
4621     True
Name: item_name, Length: 4622, dtype: bool
```

Documentation for **str.contains**

[73]:
```
# use the boolean Series to filter the DataFrame
orders[orders.item_name.str.contains('Chicken')]
```

[73]:
```
      order_id  quantity             item_name  \
4            2         2          Chicken Bowl
5            3         1          Chicken Bowl
11           6         1  Chicken Crispy Tacos
12           6         1    Chicken Soft Tacos
13           7         1          Chicken Bowl
…          …       …                    …
4604      1828         1          Chicken Bowl
4615      1832         1    Chicken Soft Tacos
4619      1834         1    Chicken Salad Bowl
4620      1834         1    Chicken Salad Bowl
4621      1834         1    Chicken Salad Bowl

                                choice_description item_price
```

```
4     [Tomatillo-Red Chili Salsa (Hot), [Black Beans…    $16.98
5     [Fresh Tomato Salsa (Mild), [Rice, Cheese, Sou…    $10.98
11    [Roasted Chili Corn Salsa, [Fajita Vegetables,…     $8.75
12    [Roasted Chili Corn Salsa, [Rice, Black Beans,…     $8.75
13    [Fresh Tomato Salsa, [Fajita Vegetables, Rice,…    $11.25
…                                                   …        …
4604  [Fresh Tomato Salsa, [Rice, Black Beans, Chees…     $8.75
4615   [Fresh Tomato Salsa, [Rice, Cheese, Sour Cream]]     $8.75
4619  [Fresh Tomato Salsa, [Fajita Vegetables, Pinto…    $11.25
4620  [Fresh Tomato Salsa, [Fajita Vegetables, Lettu…     $8.75
4621  [Fresh Tomato Salsa, [Fajita Vegetables, Pinto…     $8.75

[1560 rows x 5 columns]
```

[74]:
```
# string methods can be chained together
orders.choice_description.str.replace('[', '').str.replace(']', '')
```

[74]:
```
0                                                   NaN
1                                            Clementine
2                                                 Apple
3                                                   NaN
4       Tomatillo-Red Chili Salsa (Hot), Black Beans, …
                                  …
4617    Fresh Tomato Salsa, Rice, Black Beans, Sour Cr…
4618    Fresh Tomato Salsa, Rice, Sour Cream, Cheese, …
4619    Fresh Tomato Salsa, Fajita Vegetables, Pinto B…
4620       Fresh Tomato Salsa, Fajita Vegetables, Lettuce
4621    Fresh Tomato Salsa, Fajita Vegetables, Pinto B…
Name: choice_description, Length: 4622, dtype: object
```

Documentation for **str.replace**

[75]:
```
# many pandas string methods support regular expressions (regex)
orders.choice_description.str.replace('[\[\]]', '', regex=True)
```

[75]:
```
0                                                   NaN
1                                            Clementine
2                                                 Apple
3                                                   NaN
4       Tomatillo-Red Chili Salsa (Hot), Black Beans, …
                                  …
4617    Fresh Tomato Salsa, Rice, Black Beans, Sour Cr…
4618    Fresh Tomato Salsa, Rice, Sour Cream, Cheese, …
4619    Fresh Tomato Salsa, Fajita Vegetables, Pinto B…
4620       Fresh Tomato Salsa, Fajita Vegetables, Lettuce
4621    Fresh Tomato Salsa, Fajita Vegetables, Pinto B…
Name: choice_description, Length: 4622, dtype: object
```

String handling section of the pandas API reference

[Back to top]

1.13 Day 13: How do I change the data type of a pandas Series?

[76]:
```
# read a dataset of alcohol consumption into a DataFrame
drinks = pd.read_csv('http://bit.ly/drinksbycountry')
drinks.head()
```

[76]:
```
       country  beer_servings  spirit_servings  wine_servings  \
0  Afghanistan              0                0              0
1      Albania             89              132             54
2      Algeria             25                0             14
3      Andorra            245              138            312
4       Angola            217               57             45

   total_litres_of_pure_alcohol continent
0                           0.0      Asia
1                           4.9    Europe
2                           0.7    Africa
3                          12.4    Europe
4                           5.9    Africa
```

[77]:
```
# examine the data type of each Series
drinks.dtypes
```

[77]:
```
country                          object
beer_servings                     int64
spirit_servings                   int64
wine_servings                     int64
total_litres_of_pure_alcohol    float64
continent                        object
dtype: object
```

[78]:
```
# change the data type of an existing Series
drinks['beer_servings'] = drinks.beer_servings.astype(float)
drinks.dtypes
```

[78]:
```
country                          object
beer_servings                   float64
spirit_servings                   int64
wine_servings                     int64
total_litres_of_pure_alcohol    float64
continent                        object
dtype: object
```

Documentation for **astype**

[79]:
```
# alternatively, change the data type of a Series while reading in a file
drinks = pd.read_csv('http://bit.ly/drinksbycountry', dtype={'beer_servings':
 ↪float})
drinks.dtypes
```

[79]:
```
country                          object
beer_servings                   float64
spirit_servings                   int64
wine_servings                     int64
total_litres_of_pure_alcohol    float64
continent                        object
dtype: object
```

Documentation for **read_csv**

[80]:
```
# read a dataset of Chipotle orders into a DataFrame
orders = pd.read_table('http://bit.ly/chiporders')
orders.head()
```

[80]:
```
   order_id  quantity                              item_name  \
0         1         1           Chips and Fresh Tomato Salsa
1         1         1                                   Izze
2         1         1                       Nantucket Nectar
3         1         1  Chips and Tomatillo-Green Chili Salsa
4         2         2                           Chicken Bowl

                                choice_description item_price
0                                              NaN     $2.39
1                                     [Clementine]     $3.39
2                                          [Apple]     $3.39
3                                              NaN     $2.39
4  [Tomatillo-Red Chili Salsa (Hot), [Black Beans…    $16.98
```

[81]:
```
# examine the data type of each Series
orders.dtypes
```

[81]:
```
order_id              int64
quantity              int64
item_name            object
choice_description   object
item_price           object
dtype: object
```

[82]:
```
# convert a string to a number in order to do math
orders.item_price.str.replace('$', '').astype(float).mean()
```

[82]:
```
7.464335785374297
```

[83]:
```
# string method 'contains' checks for a substring and returns a boolean Series
orders.item_name.str.contains('Chicken')
```

[83]:
```
0       False
1       False
2       False
3       False
4        True
        ...
4617    False
4618    False
4619     True
4620     True
4621     True
Name: item_name, Length: 4622, dtype: bool
```

[84]:
```
# convert a boolean Series to an integer (False = 0, True = 1)
orders.item_name.str.contains('Chicken').astype(int)
```

[84]:
```
0       0
1       0
2       0
3       0
4       1
       ..
4617    0
4618    0
4619    1
4620    1
4621    1
Name: item_name, Length: 4622, dtype: int64
```

[Back to top]

1.14 Day 14: When should I use a "groupby" in pandas?

[85]:
```
# read a dataset of alcohol consumption into a DataFrame
drinks = pd.read_csv('http://bit.ly/drinksbycountry')
drinks.head()
```

[85]:

	country	beer_servings	spirit_servings	wine_servings	total_litres_of_pure_alcohol	continent
0	Afghanistan	0	0	0		
1	Albania	89	132	54		
2	Algeria	25	0	14		
3	Andorra	245	138	312		
4	Angola	217	57	45		

```
0                          0.0      Asia
1                          4.9    Europe
2                          0.7    Africa
3                         12.4    Europe
4                          5.9    Africa
```

[86]:
```
# calculate the mean beer servings across the entire dataset
drinks.beer_servings.mean()
```

[86]:
```
106.16062176165804
```

[87]:
```
# calculate the mean beer servings just for countries in Africa
drinks[drinks.continent=='Africa'].beer_servings.mean()
```

[87]:
```
61.471698113207545
```

[88]:
```
# calculate the mean beer servings for each continent
drinks.groupby('continent').beer_servings.mean()
```

[88]:
```
continent
Africa            61.471698
Asia              37.045455
Europe           193.777778
North America    145.434783
Oceania           89.687500
South America    175.083333
Name: beer_servings, dtype: float64
```

Documentation for **groupby**

[89]:
```
# other aggregation functions (such as 'max') can also be used with groupby
drinks.groupby('continent').beer_servings.max()
```

[89]:
```
continent
Africa           376
Asia             247
Europe           361
North America    285
Oceania          306
South America    333
Name: beer_servings, dtype: int64
```

[90]:
```
# multiple aggregation functions can be applied simultaneously
drinks.groupby('continent').beer_servings.agg(['count', 'min', 'max', 'mean'])
```

[90]:
```
               count  min  max        mean
continent
Africa            53    0  376   61.471698
```

```
Asia              44    0  247   37.045455
Europe            45    0  361  193.777778
North America     23    1  285  145.434783
Oceania           16    0  306   89.687500
South America     12   93  333  175.083333
```

Documentation for **agg()**

[91]:
```
# specifying a column to which the aggregation function should be applied is␣
↪not required
drinks.groupby('continent').mean(numeric_only=True)
```

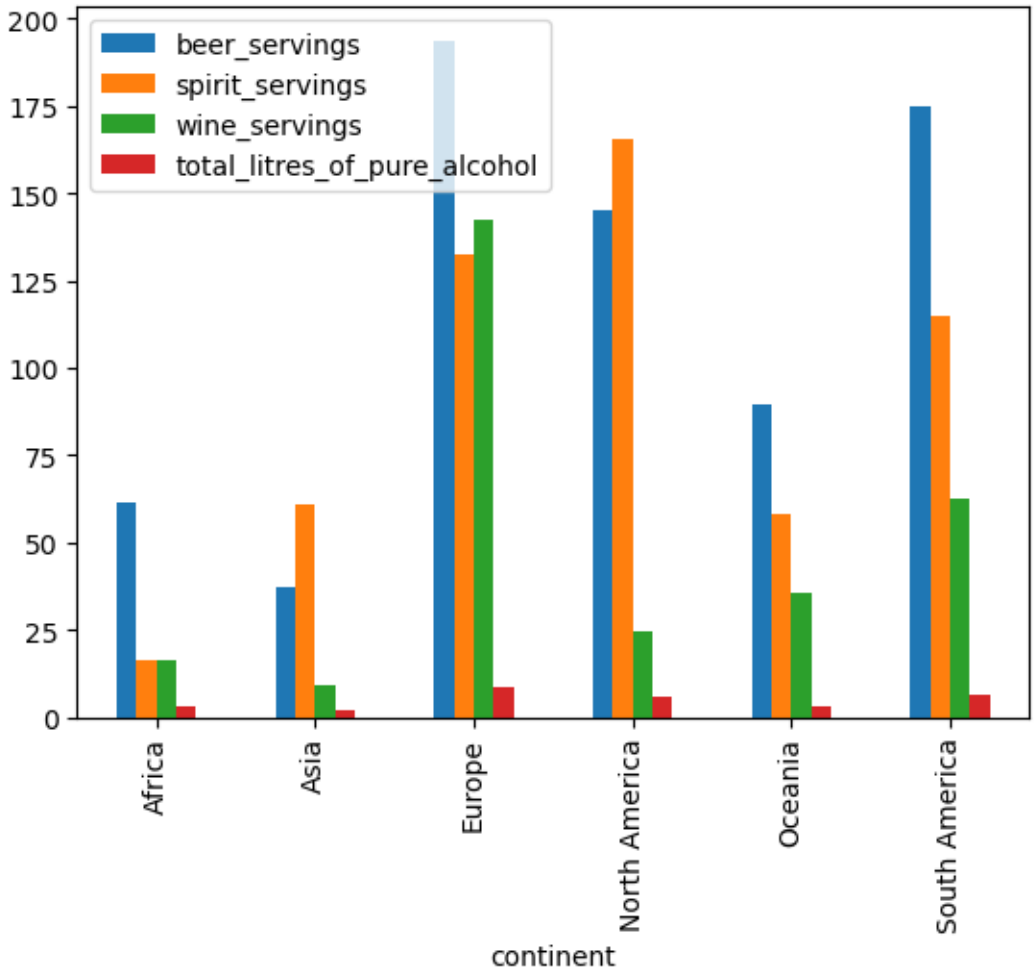
[91]:
```
               beer_servings  spirit_servings  wine_servings  \
continent
Africa             61.471698        16.339623      16.264151
Asia               37.045455        60.840909       9.068182
Europe            193.777778       132.555556     142.222222
North America     145.434783       165.739130      24.521739
Oceania            89.687500        58.437500      35.625000
South America     175.083333       114.750000      62.416667

               total_litres_of_pure_alcohol
continent
Africa                             3.007547
Asia                               2.170455
Europe                             8.617778
North America                      5.995652
Oceania                            3.381250
South America                      6.308333
```

Documentation for **mean**

[92]:
```
# side-by-side bar plot of the DataFrame directly above
drinks.groupby('continent').mean(numeric_only=True).plot(kind='bar')
```

[92]:
```
<Axes: xlabel='continent'>
```



Documentation for `plot`

[Back to top]

1.15 Day 15: How do I explore a pandas Series?

[93]:
```
# read a dataset of top-rated IMDb movies into a DataFrame
movies = pd.read_csv('http://bit.ly/imdbratings')
movies.head()
```

[93]:

	star_rating	title	content_rating	genre	duration	\
0	9.3	The Shawshank Redemption	R	Crime	142	
1	9.2	The Godfather	R	Crime	175	
2	9.1	The Godfather: Part II	R	Crime	200	
3	9.0	The Dark Knight	PG-13	Action	152	
4	8.9	Pulp Fiction	R	Crime	154	

```
                                          actors_list
0  [u'Tim Robbins', u'Morgan Freeman', u'Bob Gunt…
1    [u'Marlon Brando', u'Al Pacino', u'James Caan']
2  [u'Al Pacino', u'Robert De Niro', u'Robert Duv…
3  [u'Christian Bale', u'Heath Ledger', u'Aaron E…
4  [u'John Travolta', u'Uma Thurman', u'Samuel L…
```

[94]:
```
# examine the data type of each Series
movies.dtypes
```

[94]:
```
star_rating       float64
title              object
content_rating     object
genre              object
duration            int64
actors_list        object
dtype: object
```

1.15.1 Exploring a non-numeric Series:

[95]:
```
# count the non-null values, unique values, and frequency of the most common␣
↪value
movies.genre.describe()
```

[95]:
```
count       979
unique       16
top       Drama
freq        278
Name: genre, dtype: object
```

Documentation for **describe**

[96]:
```
# count how many times each value in the Series occurs
movies.genre.value_counts()
```

[96]:
```
genre
Drama        278
Comedy       156
Action       136
Crime        124
Biography     77
Adventure     75
Animation     62
Horror        29
Mystery       16
Western        9
Sci-Fi         5
Thriller       5
```

```
Film-Noir      3
Family         2
History        1
Fantasy        1
Name: count, dtype: int64
```

Documentation for **value_counts**

[97]:
```
# display percentages instead of raw counts
movies.genre.value_counts(normalize=True)
```

[97]:
```
genre
Drama        0.283963
Comedy       0.159346
Action       0.138917
Crime        0.126660
Biography    0.078652
Adventure    0.076609
Animation    0.063330
Horror       0.029622
Mystery      0.016343
Western      0.009193
Sci-Fi       0.005107
Thriller     0.005107
Film-Noir    0.003064
Family       0.002043
History      0.001021
Fantasy      0.001021
Name: proportion, dtype: float64
```

[98]:
```
# 'value_counts' (like many pandas methods) outputs a Series
type(movies.genre.value_counts())
```

[98]:
```
pandas.core.series.Series
```

[99]:
```
# thus, you can add another Series method on the end
movies.genre.value_counts().head()
```

[99]:
```
genre
Drama        278
Comedy       156
Action       136
Crime        124
Biography     77
Name: count, dtype: int64
```

[100]:
```
# display the unique values in the Series
movies.genre.unique()
```

[100]:
```
array(['Crime', 'Action', 'Drama', 'Western', 'Adventure', 'Biography',
       'Comedy', 'Animation', 'Mystery', 'Horror', 'Film-Noir', 'Sci-Fi',
       'History', 'Thriller', 'Family', 'Fantasy'], dtype=object)
```

[101]:
```
# count the number of unique values in the Series
movies.genre.nunique()
```

[101]:
```
16
```

Documentation for **unique** and **nunique**

[102]:
```
# compute a cross-tabulation of two Series
pd.crosstab(movies.genre, movies.content_rating)
```

[102]:

genre	APPROVED	G	GP	NC-17	NOT RATED	PASSED	PG	PG-13	R
Action	3	1	1	0	4	1	11	44	67
Adventure	3	2	0	0	5	1	21	23	17
Animation	3	20	0	0	3	0	25	5	5
Biography	1	2	1	0	1	0	6	29	36
Comedy	9	2	1	1	16	3	23	23	73
Crime	6	0	0	1	7	1	6	4	87
Drama	12	3	0	4	24	1	25	55	143
Family	0	1	0	0	0	0	1	0	0
Fantasy	0	0	0	0	0	0	0	0	1
Film-Noir	1	0	0	0	1	0	0	0	0
History	0	0	0	0	0	0	0	0	0
Horror	2	0	0	1	1	0	1	2	16
Mystery	4	1	0	0	1	0	1	2	6
Sci-Fi	1	0	0	0	0	0	0	1	3
Thriller	1	0	0	0	0	0	1	0	3
Western	1	0	0	0	2	0	2	1	3

genre	TV-MA	UNRATED	X
Action	0	3	0
Adventure	0	2	0
Animation	0	1	0
Biography	0	0	0
Comedy	0	4	1
Crime	0	11	1
Drama	1	9	1
Family	0	0	0
Fantasy	0	0	0
Film-Noir	0	1	0
History	0	1	0
Horror	0	5	1
Mystery	0	1	0

```
Sci-Fi              0        0  0
Thriller            0        0  0
Western             0        0  0
```

Documentation for **crosstab**

1.15.2 Exploring a numeric Series:

[103]:
```
# calculate various summary statistics
movies.duration.describe()
```

[103]:
```
count    979.000000
mean     120.979571
std       26.218010
min       64.000000
25%      102.000000
50%      117.000000
75%      134.000000
max      242.000000
Name: duration, dtype: float64
```

[104]:
```
# many statistics are implemented as Series methods
movies.duration.mean()
```

[104]:
```
120.97957099080695
```

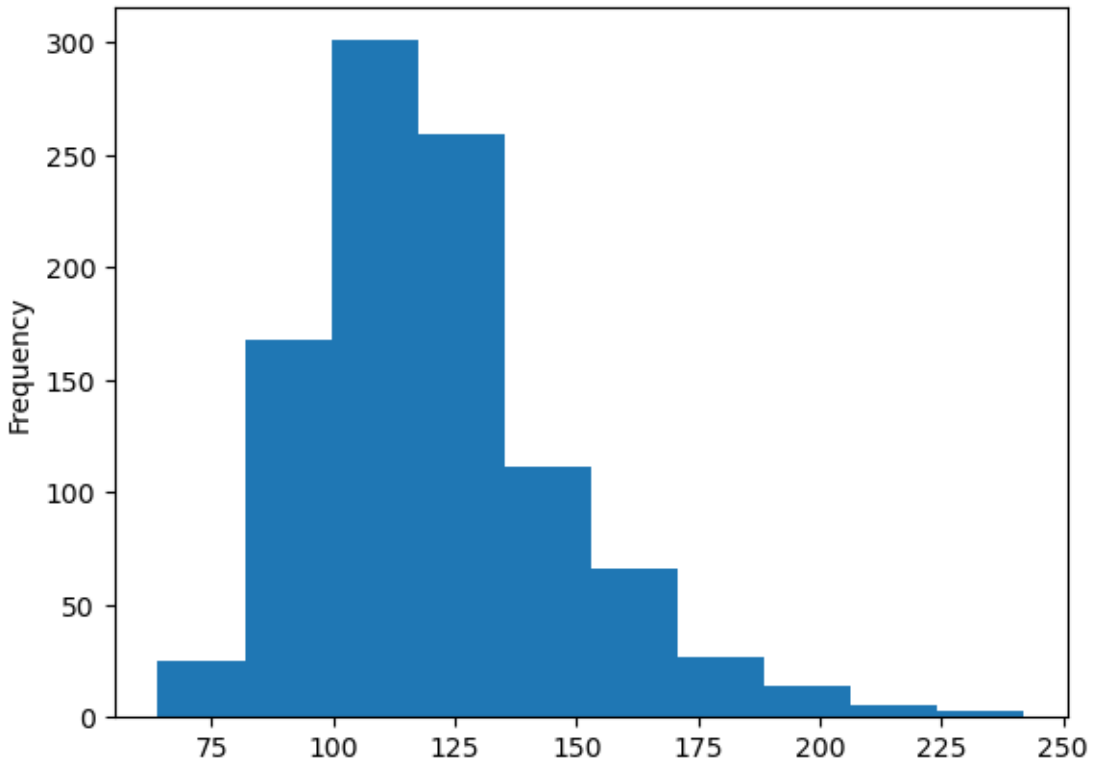
Documentation for **mean**

[105]:
```
# 'value_counts' is primarily useful for categorical data, not numerical data
movies.duration.value_counts()
```

[105]:
```
duration
112    23
113    22
102    20
101    20
129    19
       ..
67      1
195     1
76      1
66      1
205     1
Name: count, Length: 133, dtype: int64
```

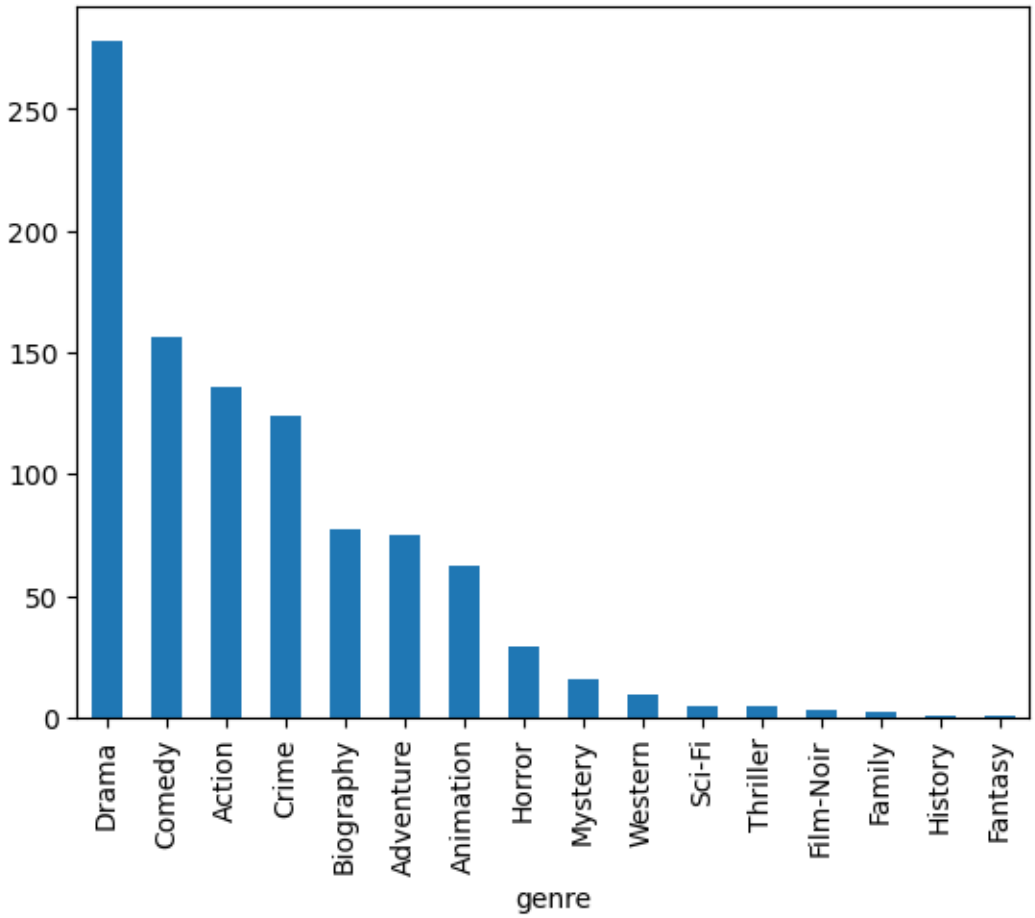
[106]:
```
# histogram of the 'duration' Series (shows the distribution of a numerical␣
↪variable)
movies.duration.plot(kind='hist')
```

[106]: <Axes: ylabel='Frequency'>



[107]:
```
# bar plot of the 'value_counts' for the 'genre' Series
movies.genre.value_counts().plot(kind='bar')
```

[107]: <Axes: xlabel='genre'>



Documentation for `plot`

[Back to top]

1.16 Day 16: How do I handle missing values in pandas?

[108]:
```
# read a dataset of UFO reports into a DataFrame
ufo = pd.read_csv('http://bit.ly/uforeports')
ufo.tail()
```

[108]:

	City	Colors Reported	Shape Reported	State	Time
18236	Grant Park	NaN	TRIANGLE	IL	12/31/2000 23:00
18237	Spirit Lake	NaN	DISK	IA	12/31/2000 23:00
18238	Eagle River	NaN	NaN	WI	12/31/2000 23:45
18239	Eagle River	RED	LIGHT	WI	12/31/2000 23:45
18240	Ybor	NaN	OVAL	FL	12/31/2000 23:59

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.

Documentation for **read_csv**

[109]:
```
# 'isna' returns a DataFrame of booleans (True if missing, False if not missing)
ufo.isna().tail()
```

[109]:
```
        City  Colors Reported  Shape Reported  State   Time
18236  False             True           False  False  False
18237  False             True           False  False  False
18238  False             True            True  False  False
18239  False            False           False  False  False
18240  False             True           False  False  False
```

[110]:
```
# 'notna' returns the opposite of 'isna' (True if not missing, False if missing)
ufo.notna().tail()
```

[110]:
```
        City  Colors Reported  Shape Reported  State  Time
18236  True            False            True   True  True
18237  True            False            True   True  True
18238  True            False           False   True  True
18239  True             True            True   True  True
18240  True            False            True   True  True
```

Documentation for **isna** and **notna**

[111]:
```
# count the number of missing values in each Series
ufo.isna().sum()
```

[111]:
```
City                26
Colors Reported    15359
Shape Reported      2644
State                  0
Time                   0
dtype: int64
```

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**.

[112]:
```
# use the 'isna' Series method to filter the DataFrame rows
ufo[ufo.City.isna()]
```

[112]:
```
      City Colors Reported Shape Reported State              Time
21     NaN             NaN            NaN    LA    8/15/1943 0:00
22     NaN             NaN          LIGHT    LA    8/15/1943 0:00
```

```
204    NaN            NaN           DISK    CA   7/15/1952 12:30
241    NaN           BLUE           DISK    MT    7/4/1953 14:00
613    NaN            NaN           DISK    NV    7/1/1960 12:00
1877   NaN         YELLOW         CIRCLE    AZ    8/15/1969 1:00
2013   NaN            NaN            NaN    NH     8/1/1970 9:30
2546   NaN            NaN       FIREBALL    OH  10/25/1973 23:30
3123   NaN            RED       TRIANGLE    WV  11/25/1975 23:00
4736   NaN            NaN         SPHERE    CA   6/23/1982 23:00
5269   NaN            NaN            NaN    AZ   6/30/1985 21:30
6113   NaN            NaN          OTHER    CO   6/30/1989 22:30
6735   NaN            NaN      FORMATION    TX     4/1/1992 2:00
7208   NaN            NaN         CIRCLE    MI   10/4/1993 17:30
8828   NaN            NaN       TRIANGLE    WA  10/30/1995 21:30
8967   NaN            NaN        VARIOUS    CA   12/8/1995 18:00
9273   NaN            NaN       TRIANGLE    OH     5/1/1996 3:00
9388   NaN            NaN           OVAL    CA   6/12/1996 12:00
9587   NaN            NaN            EGG    FL   8/24/1996 15:00
10399  NaN            NaN       TRIANGLE    IL   6/15/1997 23:00
11625  NaN            NaN         CIRCLE    TX     6/7/1998 7:00
12441  NaN            RED       FIREBALL    WA  10/26/1998 17:58
15767  NaN            NaN      RECTANGLE    NV   1/21/2000 11:30
15812  NaN            NaN          LIGHT    NV     2/2/2000 3:00
16054  NaN          GREEN            NaN    FL    3/11/2000 3:30
16608  NaN            NaN         SPHERE    NY   6/15/2000 15:00
```

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

[113]:
```
# examine the number of rows and columns
ufo.shape
```

[113]:
```
(18241, 5)
```

[114]:
```
# if 'any' values are missing in a row, then drop that row
ufo.dropna(how='any').shape
```

[114]:
```
(2486, 5)
```

Documentation for **dropna**

[115]:
```
# 'inplace' parameter for 'dropna' is False by default, thus rows were only␣
↪dropped temporarily
ufo.shape
```

[115]:
```
(18241, 5)
```

[116]:
```
# if 'all' values are missing in a row, then drop that row (none are dropped in␣
↪this case)
```

```
ufo.dropna(how='all').shape
```

[116]:
```
(18241, 5)
```

[117]:
```
# if 'any' values are missing in a row (considering only 'City' and 'Shape␣
↪Reported'), then drop that row
ufo.dropna(subset=['City', 'Shape Reported'], how='any').shape
```

[117]:
```
(15575, 5)
```

[118]:
```
# if 'all' values are missing in a row (considering only 'City' and 'Shape␣
↪Reported'), then drop that row
ufo.dropna(subset=['City', 'Shape Reported'], how='all').shape
```

[118]:
```
(18237, 5)
```

[119]:
```
# 'value_counts' does not include missing values by default
ufo['Shape Reported'].value_counts()
```

[119]:
```
Shape Reported
LIGHT        2803
DISK         2122
TRIANGLE     1889
OTHER        1402
CIRCLE       1365
SPHERE       1054
FIREBALL     1039
OVAL          845
CIGAR         617
FORMATION     434
VARIOUS       333
RECTANGLE     303
CYLINDER      294
CHEVRON       248
DIAMOND       234
EGG           197
FLASH         188
TEARDROP      119
CONE           60
CROSS          36
DELTA           7
ROUND           2
CRESCENT        2
DOME            1
PYRAMID         1
FLARE           1
HEXAGON         1
```

```
Name: count, dtype: int64
```

[120]:
```
# explicitly include missing values
ufo['Shape Reported'].value_counts(dropna=False)
```

[120]:
```
Shape Reported
LIGHT        2803
NaN          2644
DISK         2122
TRIANGLE     1889
OTHER        1402
CIRCLE       1365
SPHERE       1054
FIREBALL     1039
OVAL          845
CIGAR         617
FORMATION     434
VARIOUS       333
RECTANGLE     303
CYLINDER      294
CHEVRON       248
DIAMOND       234
EGG           197
FLASH         188
TEARDROP      119
CONE           60
CROSS          36
DELTA           7
ROUND           2
CRESCENT        2
DOME            1
PYRAMID         1
FLARE           1
HEXAGON         1
Name: count, dtype: int64
```

Documentation for **value_counts**

[121]:
```
# fill in missing values with a specified value
ufo['Shape Reported'].fillna(value='VARIOUS', inplace=True)
```

Documentation for **fillna**

[122]:
```
# confirm that the missing values were filled in
ufo['Shape Reported'].value_counts()
```

[122]:
```
Shape Reported
VARIOUS      2977
```

```
LIGHT        2803
DISK         2122
TRIANGLE     1889
OTHER        1402
CIRCLE       1365
SPHERE       1054
FIREBALL     1039
OVAL          845
CIGAR         617
FORMATION     434
RECTANGLE     303
CYLINDER      294
CHEVRON       248
DIAMOND       234
EGG           197
FLASH         188
TEARDROP      119
CONE           60
CROSS          36
DELTA           7
ROUND           2
CRESCENT        2
DOME            1
PYRAMID         1
FLARE           1
HEXAGON         1
Name: count, dtype: int64
```

Working with missing data from the pandas user guide

[Back to top]

1.17 Day 17: What do I need to know about the pandas index? (Part 1)

[123]:
```
# read a dataset of alcohol consumption into a DataFrame
drinks = pd.read_csv('http://bit.ly/drinksbycountry')
drinks.head()
```

[123]:

	country	beer_servings	spirit_servings	wine_servings	total_litres_of_pure_alcohol	continent
0	Afghanistan	0	0	0	0.0	Asia
1	Albania	89	132	54	4.9	Europe
2	Algeria	25	0	14		
3	Andorra	245	138	312		
4	Angola	217	57	45		

```
2                          0.7    Africa
3                         12.4    Europe
4                          5.9    Africa
```

[124]:
```
# every DataFrame has an index (sometimes called the "row labels")
drinks.index
```

[124]:
```
RangeIndex(start=0, stop=193, step=1)
```

[125]:
```
# column names are also stored in a special "index" object
drinks.columns
```

[125]:
```
Index(['country', 'beer_servings', 'spirit_servings', 'wine_servings',
       'total_litres_of_pure_alcohol', 'continent'],
      dtype='object')
```

[126]:
```
# neither the index nor the columns are included in the shape
drinks.shape
```

[126]:
```
(193, 6)
```

[127]:
```
# index and columns both default to integers if you don't define them
pd.read_table('http://bit.ly/movieusers', header=None, sep='|').head()
```

[127]:

	0	1	2	3	4
0	1	24	M	technician	85711
1	2	53	F	other	94043
2	3	23	M	writer	32067
3	4	24	M	technician	43537
4	5	33	F	other	15213

What is the index used for?

1. identification
2. selection
3. alignment (covered in the next video)

[128]:
```
# identification: index remains with each row when filtering the DataFrame
drinks[drinks.continent=='South America']
```

[128]:

	country	beer_servings	spirit_servings	wine_servings	\
6	Argentina	193	25	221	
20	Bolivia	167	41	8	
23	Brazil	245	145	16	
35	Chile	130	124	172	
37	Colombia	159	76	3	
52	Ecuador	162	74	3	
72	Guyana	93	302	1	
132	Paraguay	213	117	74	

```
133       Peru            163              160             21
163   Suriname            128              178              7
185    Uruguay            115               35            220
188  Venezuela            333              100              3

     total_litres_of_pure_alcohol      continent
6                             8.3  South America
20                            3.8  South America
23                            7.2  South America
35                            7.6  South America
37                            4.2  South America
52                            4.2  South America
72                            7.1  South America
132                           7.3  South America
133                           6.1  South America
163                           5.6  South America
185                           6.6  South America
188                           7.7  South America
```

[129]:
```
# selection: select a portion of the DataFrame using the index
drinks.loc[23, 'beer_servings']
```

[129]:
```
245
```

Documentation for **loc**

[130]:
```
# set an existing column as the index
drinks.set_index('country', inplace=True)
drinks.head()
```

[130]:
```
             beer_servings  spirit_servings  wine_servings  \
country
Afghanistan              0                0              0
Albania                 89              132             54
Algeria                 25                0             14
Andorra                245              138            312
Angola                 217               57             45

             total_litres_of_pure_alcohol continent
country
Afghanistan                           0.0      Asia
Albania                               4.9    Europe
Algeria                               0.7    Africa
Andorra                              12.4    Europe
Angola                                5.9    Africa
```

Documentation for **set_index**

[131]:
```
# 'country' is now the index
drinks.index
```

[131]:
```
Index(['Afghanistan', 'Albania', 'Algeria', 'Andorra', 'Angola',
       'Antigua & Barbuda', 'Argentina', 'Armenia', 'Australia', 'Austria',
       …
       'Tanzania', 'USA', 'Uruguay', 'Uzbekistan', 'Vanuatu', 'Venezuela',
       'Vietnam', 'Yemen', 'Zambia', 'Zimbabwe'],
      dtype='object', name='country', length=193)
```

[132]:
```
# 'country' is no longer a column
drinks.columns
```

[132]:
```
Index(['beer_servings', 'spirit_servings', 'wine_servings',
       'total_litres_of_pure_alcohol', 'continent'],
      dtype='object')
```

[133]:
```
# 'country' data is no longer part of the DataFrame contents
drinks.shape
```

[133]:
```
(193, 5)
```

[134]:
```
# country name can now be used for selection
drinks.loc['Brazil', 'beer_servings']
```

[134]:
```
245
```

[135]:
```
# index name is optional
drinks.index.name = None
drinks.head()
```

[135]:

	beer_servings	spirit_servings	wine_servings	total_litres_of_pure_alcohol	continent
Afghanistan	0	0	0	0.0	Asia
Albania	89	132	54	4.9	Europe
Algeria	25	0	14	0.7	Africa
Andorra	245	138	312	12.4	Europe
Angola	217	57	45	5.9	Africa

[136]:
```
# restore the index name, and move the index back to a column
drinks.index.name = 'country'
drinks.reset_index(inplace=True)
```

```
drinks.head()
```

[136]:
```
       country  beer_servings  spirit_servings  wine_servings  \
0  Afghanistan              0                0              0
1      Albania             89              132             54
2      Algeria             25                0             14
3      Andorra            245              138            312
4       Angola            217               57             45

   total_litres_of_pure_alcohol continent
0                           0.0      Asia
1                           4.9    Europe
2                           0.7    Africa
3                          12.4    Europe
4                           5.9    Africa
```

Documentation for **reset_index**

[137]:
```
# many DataFrame methods output a DataFrame
drinks.describe()
```

[137]:
```
       beer_servings  spirit_servings  wine_servings  \
count     193.000000       193.000000     193.000000
mean      106.160622        80.994819      49.450777
std       101.143103        88.284312      79.697598
min         0.000000         0.000000       0.000000
25%        20.000000         4.000000       1.000000
50%        76.000000        56.000000       8.000000
75%       188.000000       128.000000      59.000000
max       376.000000       438.000000     370.000000

       total_litres_of_pure_alcohol
count                    193.000000
mean                       4.717098
std                        3.773298
min                        0.000000
25%                        1.300000
50%                        4.200000
75%                        7.200000
max                       14.400000
```

[138]:
```
# you can interact with any DataFrame using its index and columns
drinks.describe().loc['25%', 'beer_servings']
```

[138]:
```
20.0
```

Indexing and selecting data from the pandas user guide

[Back to top]

1.18 Day 18: What do I need to know about the pandas index? (Part 2)

[139]:
```
# read a dataset of alcohol consumption into a DataFrame
drinks = pd.read_csv('http://bit.ly/drinksbycountry')
drinks.head()
```

[139]:

	country	beer_servings	spirit_servings	wine_servings	total_litres_of_pure_alcohol	continent
0	Afghanistan	0	0	0	0.0	Asia
1	Albania	89	132	54	4.9	Europe
2	Algeria	25	0	14	0.7	Africa
3	Andorra	245	138	312	12.4	Europe
4	Angola	217	57	45	5.9	Africa

[140]:
```
# every DataFrame has an index
drinks.index
```

[140]:
```
RangeIndex(start=0, stop=193, step=1)
```

[141]:
```
# every Series also has an index (which carries over from the DataFrame)
drinks.continent.head()
```

[141]:
```
0      Asia
1    Europe
2    Africa
3    Europe
4    Africa
Name: continent, dtype: object
```

[142]:
```
# set 'country' as the index
drinks.set_index('country', inplace=True)
```

Documentation for **set_index**

[143]:
```
# Series index is on the left, values are on the right
drinks.continent.head()
```

[143]:
```
country
Afghanistan      Asia
Albania        Europe
Algeria        Africa
Andorra        Europe
```

```
Angola         Africa
Name: continent, dtype: object
```

[144]:
```
# another example of a Series (output from the 'value_counts' method)
drinks.continent.value_counts()
```

[144]:
```
continent
Africa           53
Europe           45
Asia             44
North America    23
Oceania          16
South America    12
Name: count, dtype: int64
```

Documentation for **value_counts**

[145]:
```
# access the Series index
drinks.continent.value_counts().index
```

[145]:
```
Index(['Africa', 'Europe', 'Asia', 'North America', 'Oceania',
       'South America'],
      dtype='object', name='continent')
```

[146]:
```
# access the Series values
drinks.continent.value_counts().values
```

[146]:
```
array([53, 45, 44, 23, 16, 12])
```

[147]:
```
# elements in a Series can be selected by index (using bracket notation)
drinks.continent.value_counts()['Africa']
```

[147]:
```
53
```

[148]:
```
# any Series can be sorted by its values
drinks.continent.value_counts().sort_values()
```

[148]:
```
continent
South America    12
Oceania          16
North America    23
Asia             44
Europe           45
Africa           53
Name: count, dtype: int64
```

[149]:
```
# any Series can also be sorted by its index
drinks.continent.value_counts().sort_index()
```

[149]:
```
continent
Africa           53
Asia             44
Europe           45
North America    23
Oceania          16
South America    12
Name: count, dtype: int64
```

Documentation for **sort_values** and **sort_index**

What is the index used for?

1. identification (covered in the previous video)
2. selection (covered in the previous video)
3. alignment

[150]:
```
# 'beer_servings' Series contains the average annual beer servings per person
drinks.beer_servings.head()
```

[150]:
```
country
Afghanistan      0
Albania         89
Algeria         25
Andorra        245
Angola         217
Name: beer_servings, dtype: int64
```

[151]:
```
# create a Series containing the population of two countries
people = pd.Series([3000000, 85000], index=['Albania', 'Andorra'],␣
 ↪name='population')
people
```

[151]:
```
Albania    3000000
Andorra      85000
Name: population, dtype: int64
```

Documentation for **Series**

[152]:
```
# calculate the total annual beer servings for each country
(drinks.beer_servings * people).head()
```

[152]:
```
Afghanistan            NaN
Albania        267000000.0
Algeria                NaN
Andorra         20825000.0
Angola                 NaN
dtype: float64
```

- The two Series were **aligned** by their indexes.

- If a value is missing in either Series, the result is marked as **NaN**.
- Alignment enables us to easily work with **incomplete data**.

[153]:
```
# concatenate the 'drinks' DataFrame with the 'population' Series (aligns by␣
↪the index)
pd.concat([drinks, people], axis=1).head()
```

[153]:
```
             beer_servings  spirit_servings  wine_servings  \
Afghanistan              0                0              0
Albania                 89              132             54
Algeria                 25                0             14
Andorra                245              138            312
Angola                 217               57             45

             total_litres_of_pure_alcohol continent  population
Afghanistan                           0.0      Asia         NaN
Albania                               4.9    Europe   3000000.0
Algeria                               0.7    Africa         NaN
Andorra                              12.4    Europe     85000.0
Angola                                5.9    Africa         NaN
```

Documentation for **concat**

Indexing and selecting data from the pandas user guide

[Back to top]

1.19 Day 19: How do I select multiple rows and columns from a pandas DataFrame?

[154]:
```
# read a dataset of UFO reports into a DataFrame
ufo = pd.read_csv('http://bit.ly/uforeports')
ufo.head(3)
```

[154]:
```
          City Colors Reported Shape Reported State              Time
0       Ithaca             NaN       TRIANGLE    NY   6/1/1930 22:00
1  Willingboro             NaN          OTHER    NJ  6/30/1930 20:00
2      Holyoke             NaN           OVAL    CO  2/15/1931 14:00
```

The **loc** accessor 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”)

[155]:
```
# row 0, all columns
ufo.loc[0, :]
```

[155]:
```
City                        Ithaca
Colors Reported                NaN
Shape Reported            TRIANGLE
State                           NY
Time                 6/1/1930 22:00
Name: 0, dtype: object
```

[156]:
```
# rows 0 and 1 and 2, all columns
ufo.loc[[0, 1, 2], :]
```

[156]:

	City	Colors Reported	Shape Reported	State	Time
0	Ithaca	NaN	TRIANGLE	NY	6/1/1930 22:00
1	Willingboro	NaN	OTHER	NJ	6/30/1930 20:00
2	Holyoke	NaN	OVAL	CO	2/15/1931 14:00

[157]:
```
# rows 0 through 2 (inclusive), all columns
ufo.loc[0:2, :]
```

[157]:

	City	Colors Reported	Shape Reported	State	Time
0	Ithaca	NaN	TRIANGLE	NY	6/1/1930 22:00
1	Willingboro	NaN	OTHER	NJ	6/30/1930 20:00
2	Holyoke	NaN	OVAL	CO	2/15/1931 14:00

[158]:
```
# this implies "all columns", but explicitly stating "all columns" is better
ufo.loc[0:2]
```

[158]:

	City	Colors Reported	Shape Reported	State	Time
0	Ithaca	NaN	TRIANGLE	NY	6/1/1930 22:00
1	Willingboro	NaN	OTHER	NJ	6/30/1930 20:00
2	Holyoke	NaN	OVAL	CO	2/15/1931 14:00

[159]:
```
# all rows, column 'City'
ufo.loc[:, 'City']
```

[159]:
```
0                    Ithaca
1                Willingboro
2                    Holyoke
3                    Abilene
4        New York Worlds Fair
                 …
18236             Grant Park
18237            Spirit Lake
18238            Eagle River
18239            Eagle River
18240                   Ybor
Name: City, Length: 18241, dtype: object
```

[160]:
```
# all rows, columns 'City' and 'State'
ufo.loc[:, ['City', 'State']]
```

[160]:
```
                      City State
0                   Ithaca    NY
1              Willingboro    NJ
2                  Holyoke    CO
3                  Abilene    KS
4      New York Worlds Fair    NY
…                      …   …
18236           Grant Park    IL
18237          Spirit Lake    IA
18238          Eagle River    WI
18239          Eagle River    WI
18240                 Ybor    FL

[18241 rows x 2 columns]
```

[161]:
```
# accomplish the same thing - but 'loc' is preferred since it's more explicit
ufo[['City', 'State']]
```

[161]:
```
                      City State
0                   Ithaca    NY
1              Willingboro    NJ
2                  Holyoke    CO
3                  Abilene    KS
4      New York Worlds Fair    NY
…                      …   …
18236           Grant Park    IL
18237          Spirit Lake    IA
18238          Eagle River    WI
18239          Eagle River    WI
18240                 Ybor    FL

[18241 rows x 2 columns]
```

[162]:
```
# all rows, columns 'City' through 'State' (inclusive)
ufo.loc[:, 'City':'State']
```

[162]:
```
                      City Colors Reported Shape Reported State
0                   Ithaca             NaN       TRIANGLE    NY
1              Willingboro             NaN          OTHER    NJ
2                  Holyoke             NaN           OVAL    CO
3                  Abilene             NaN           DISK    KS
4      New York Worlds Fair             NaN          LIGHT    NY
…                      …              …             …   …
18236           Grant Park             NaN       TRIANGLE    IL
```

```
18237           Spirit Lake             NaN           DISK    IA
18238           Eagle River             NaN            NaN    WI
18239           Eagle River             RED          LIGHT    WI
18240                  Ybor             NaN           OVAL    FL

[18241 rows x 4 columns]
```

[163]:
```
# rows 0 through 2 (inclusive), columns 'City' through 'State' (inclusive)
ufo.loc[0:2, 'City':'State']
```

[163]:
```
          City Colors Reported Shape Reported State
0       Ithaca             NaN       TRIANGLE    NY
1  Willingboro             NaN          OTHER    NJ
2      Holyoke             NaN           OVAL    CO
```

[164]:
```
# accomplish the same thing using 'head' and 'drop'
ufo.head(3).drop('Time', axis=1)
```

[164]:
```
          City Colors Reported Shape Reported State
0       Ithaca             NaN       TRIANGLE    NY
1  Willingboro             NaN          OTHER    NJ
2      Holyoke             NaN           OVAL    CO
```

[165]:
```
# rows in which the 'City' is 'Oakland'
ufo[ufo.City=='Oakland']
```

[165]:
```
          City Colors Reported Shape Reported State              Time
1694   Oakland             NaN          CIGAR    CA   7/21/1968 14:00
2144   Oakland             NaN           DISK    CA    8/19/1971 0:00
4686   Oakland             NaN          LIGHT    MD     6/1/1982 0:00
7293   Oakland             NaN          LIGHT    CA   3/28/1994 17:00
8488   Oakland             NaN            NaN    CA   8/10/1995 21:45
8768   Oakland             NaN            NaN    CA  10/10/1995 22:40
10816  Oakland             NaN          LIGHT    OR   10/1/1997 21:30
10948  Oakland             NaN           DISK    CA  11/14/1997 19:55
11045  Oakland             NaN       TRIANGLE    CA   12/10/1997 1:30
12322  Oakland             NaN       FIREBALL    CA   10/9/1998 19:40
12941  Oakland             NaN       CYLINDER    CA   1/23/1999 21:30
16803  Oakland             NaN       TRIANGLE    MD    7/4/2000 23:00
17322  Oakland             NaN       CYLINDER    CA    9/1/2000 21:35
```

[166]:
```
# accomplish the same thing using loc
ufo.loc[ufo.City=='Oakland', :]
```

[166]:
```
          City Colors Reported Shape Reported State              Time
1694   Oakland             NaN          CIGAR    CA   7/21/1968 14:00
2144   Oakland             NaN           DISK    CA    8/19/1971 0:00
4686   Oakland             NaN          LIGHT    MD     6/1/1982 0:00
```

```
7293   Oakland              NaN          LIGHT    CA   3/28/1994 17:00
8488   Oakland              NaN            NaN    CA   8/10/1995 21:45
8768   Oakland              NaN            NaN    CA  10/10/1995 22:40
10816  Oakland              NaN          LIGHT    OR   10/1/1997 21:30
10948  Oakland              NaN           DISK    CA  11/14/1997 19:55
11045  Oakland              NaN       TRIANGLE    CA   12/10/1997 1:30
12322  Oakland              NaN       FIREBALL    CA   10/9/1998 19:40
12941  Oakland              NaN       CYLINDER    CA   1/23/1999 21:30
16803  Oakland              NaN       TRIANGLE    MD    7/4/2000 23:00
17322  Oakland              NaN       CYLINDER    CA    9/1/2000 21:35
```

[167]:
```
# rows in which the 'City' is 'Oakland', column 'State'
ufo.loc[ufo.City=='Oakland', 'State']
```

[167]:
```
1694     CA
2144     CA
4686     MD
7293     CA
8488     CA
8768     CA
10816    OR
10948    CA
11045    CA
12322    CA
12941    CA
16803    MD
17322    CA
Name: State, dtype: object
```

[168]:
```
# accomplish the same thing using "chained indexing" - but 'loc' is preferred␣
 ↪since chained indexing can cause problems
ufo[ufo.City=='Oakland'].State
```

[168]:
```
1694     CA
2144     CA
4686     MD
7293     CA
8488     CA
8768     CA
10816    OR
10948    CA
11045    CA
12322    CA
12941    CA
16803    MD
17322    CA
Name: State, dtype: object
```

The `iloc` accessor 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")

[169]:
```
# all rows, columns in positions 0 and 3
ufo.iloc[:, [0, 3]]
```

[169]:
```
                      City State
0                   Ithaca    NY
1                Willingboro    NJ
2                    Holyoke    CO
3                    Abilene    KS
4      New York Worlds Fair    NY
…                       …   …
18236            Grant Park    IL
18237           Spirit Lake    IA
18238           Eagle River    WI
18239           Eagle River    WI
18240                  Ybor    FL

[18241 rows x 2 columns]
```

[170]:
```
# all rows, columns in positions 0 through 4 (exclusive)
ufo.iloc[:, 0:4]
```

[170]:
```
                      City Colors Reported Shape Reported State
0                   Ithaca             NaN       TRIANGLE    NY
1                Willingboro             NaN          OTHER    NJ
2                    Holyoke             NaN           OVAL    CO
3                    Abilene             NaN           DISK    KS
4      New York Worlds Fair             NaN          LIGHT    NY
…                       …              …             …   …
18236            Grant Park             NaN       TRIANGLE    IL
18237           Spirit Lake             NaN           DISK    IA
18238           Eagle River             NaN            NaN    WI
18239           Eagle River             RED          LIGHT    WI
18240                  Ybor             NaN           OVAL    FL

[18241 rows x 4 columns]
```

[171]:
```
# iloc slices are exclusive of the stopping value, just like range
list(range(0, 4))
```

[171]:
```
[0, 1, 2, 3]
```

[172]:
```
# rows in positions 0 through 3 (exclusive), all columns
ufo.iloc[0:3, :]
```

[172]:

	City	Colors Reported	Shape Reported	State	Time
0	Ithaca	NaN	TRIANGLE	NY	6/1/1930 22:00
1	Willingboro	NaN	OTHER	NJ	6/30/1930 20:00
2	Holyoke	NaN	OVAL	CO	2/15/1931 14:00

[173]:
```
# accomplish the same thing - but 'iloc' is preferred since it's more explicit
ufo[0:3]
```

[173]:

	City	Colors Reported	Shape Reported	State	Time
0	Ithaca	NaN	TRIANGLE	NY	6/1/1930 22:00
1	Willingboro	NaN	OTHER	NJ	6/30/1930 20:00
2	Holyoke	NaN	OVAL	CO	2/15/1931 14:00

Different choices for indexing in the pandas user guide

[174]:
```
# read a dataset of alcohol consumption into a DataFrame and set 'country' as␣
↪the index
drinks = pd.read_csv('http://bit.ly/drinksbycountry', index_col='country')
drinks.head()
```

[174]:

country	beer_servings	spirit_servings	wine_servings	total_litres_of_pure_alcohol	continent
Afghanistan	0	0	0	0.0	Asia
Albania	89	132	54	4.9	Europe
Algeria	25	0	14	0.7	Africa
Andorra	245	138	312	12.4	Europe
Angola	217	57	45	5.9	Africa

[175]:
```
# row with label 'Albania', column in position 0
drinks.loc['Albania', drinks.columns[0]]
```

[175]: 89

[176]:
```
# row in position 1, column with label 'beer_servings'
drinks.iloc[1, drinks.columns.get_loc('beer_servings')]
```

[176]: 89

[177]:
```
# rows 'Albania' through 'Andorra' (inclusive), columns in positions 0 through
↪2 (exclusive)
drinks.loc['Albania':'Andorra', drinks.columns[0:2]]
```

[177]:
```
         beer_servings  spirit_servings
country
Albania             89              132
Algeria             25                0
Andorra            245              138
```

[178]:
```
# rows 0 through 2 (inclusive), columns in positions 0 through 2 (exclusive)
ufo.loc[0:2, ufo.columns[0:2]]
```

[178]:
```
           City Colors Reported
0        Ithaca             NaN
1  Willingboro             NaN
2      Holyoke             NaN
```

Combining positional and label-based indexing in the pandas user guide

[Back to top]

1.20 Day 20: When should I use the "inplace" parameter in pandas?

[179]:
```
# read a dataset of UFO reports into a DataFrame
ufo = pd.read_csv('http://bit.ly/uforeports')
ufo.head()
```

[179]:
```
                   City Colors Reported Shape Reported State              Time
0                Ithaca             NaN       TRIANGLE    NY   6/1/1930 22:00
1           Willingboro             NaN          OTHER    NJ  6/30/1930 20:00
2               Holyoke             NaN           OVAL    CO  2/15/1931 14:00
3               Abilene             NaN           DISK    KS   6/1/1931 13:00
4  New York Worlds Fair             NaN          LIGHT    NY  4/18/1933 19:00
```

[180]:
```
ufo.shape
```

[180]:
```
(18241, 5)
```

[181]:
```
# remove the 'City' column (doesn't affect the DataFrame since inplace=False)
ufo.drop('City', axis=1).head()
```

[181]:
```
  Colors Reported Shape Reported State              Time
0             NaN       TRIANGLE    NY   6/1/1930 22:00
1             NaN          OTHER    NJ  6/30/1930 20:00
2             NaN           OVAL    CO  2/15/1931 14:00
3             NaN           DISK    KS   6/1/1931 13:00
4             NaN          LIGHT    NY  4/18/1933 19:00
```

[182]:
```
# confirm that the 'City' column was not actually removed
ufo.head()
```

[182]:

	City	Colors Reported	Shape Reported	State	Time
0	Ithaca	NaN	TRIANGLE	NY	6/1/1930 22:00
1	Willingboro	NaN	OTHER	NJ	6/30/1930 20:00
2	Holyoke	NaN	OVAL	CO	2/15/1931 14:00
3	Abilene	NaN	DISK	KS	6/1/1931 13:00
4	New York Worlds Fair	NaN	LIGHT	NY	4/18/1933 19:00

Documentation for **drop**

[183]:
```
# remove the 'City' column (does affect the DataFrame since inplace=True)
ufo.drop('City', axis=1, inplace=True)
```

[184]:
```
# confirm that the 'City' column was actually removed
ufo.head()
```

[184]:

	Colors Reported	Shape Reported	State	Time
0	NaN	TRIANGLE	NY	6/1/1930 22:00
1	NaN	OTHER	NJ	6/30/1930 20:00
2	NaN	OVAL	CO	2/15/1931 14:00
3	NaN	DISK	KS	6/1/1931 13:00
4	NaN	LIGHT	NY	4/18/1933 19:00

[185]:
```
# drop a row if any value is missing from that row (doesn't affect the␣
↪DataFrame since inplace=False)
ufo.dropna(how='any').shape
```

[185]: (2490, 4)

[186]:
```
# confirm that no rows were actually removed
ufo.shape
```

[186]: (18241, 4)

Documentation for **dropna**

[187]:
```
# use an assignment statement instead of the 'inplace' parameter
ufo = ufo.set_index('Time')
ufo.tail()
```

[187]:

Time	Colors Reported	Shape Reported	State
12/31/2000 23:00	NaN	TRIANGLE	IL
12/31/2000 23:00	NaN	DISK	IA
12/31/2000 23:45	NaN	NaN	WI
12/31/2000 23:45	RED	LIGHT	WI
12/31/2000 23:59	NaN	OVAL	FL

Documentation for **set_index**

[188]:
```
# fill missing values using "backward fill" strategy (doesn't affect the␣
↪DataFrame since inplace=False)
ufo.bfill().tail()
```

[188]:
```
                 Colors Reported Shape Reported State
Time
12/31/2000 23:00              RED        TRIANGLE    IL
12/31/2000 23:00              RED            DISK    IA
12/31/2000 23:45              RED           LIGHT    WI
12/31/2000 23:45              RED           LIGHT    WI
12/31/2000 23:59              NaN            OVAL    FL
```

[189]:
```
# compare with "forward fill" strategy (doesn't affect the DataFrame since␣
↪inplace=False)
ufo.ffill().tail()
```

[189]:
```
                 Colors Reported Shape Reported State
Time
12/31/2000 23:00              RED        TRIANGLE    IL
12/31/2000 23:00              RED            DISK    IA
12/31/2000 23:45              RED            DISK    WI
12/31/2000 23:45              RED           LIGHT    WI
12/31/2000 23:59              RED            OVAL    FL
```

Documentation for **bfill** and **ffill**

[Back to top]

1.21 Day 21: How do I make my pandas DataFrame smaller and faster?

[190]:
```
# read a dataset of alcohol consumption into a DataFrame
drinks = pd.read_csv('http://bit.ly/drinksbycountry')
drinks.head()
```

[190]:
```
       country  beer_servings  spirit_servings  wine_servings  \
0  Afghanistan              0                0              0
1      Albania             89              132             54
2      Algeria             25                0             14
3      Andorra            245              138            312
4       Angola            217               57             45

   total_litres_of_pure_alcohol continent
0                           0.0      Asia
1                           4.9    Europe
2                           0.7    Africa
3                          12.4    Europe
4                           5.9    Africa
```

[191]:
```
# exact memory usage is unknown because object columns are references elsewhere
drinks.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 193 entries, 0 to 192
Data columns (total 6 columns):
 #   Column                        Non-Null Count  Dtype
---  ------                        --------------  -----
 0   country                       193 non-null    object
 1   beer_servings                 193 non-null    int64
 2   spirit_servings               193 non-null    int64
 3   wine_servings                 193 non-null    int64
 4   total_litres_of_pure_alcohol  193 non-null    float64
 5   continent                     193 non-null    object
dtypes: float64(1), int64(3), object(2)
memory usage: 9.2+ KB
```

[192]:
```
# force pandas to calculate the true memory usage
drinks.info(memory_usage='deep')
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 193 entries, 0 to 192
Data columns (total 6 columns):
 #   Column                        Non-Null Count  Dtype
---  ------                        --------------  -----
 0   country                       193 non-null    object
 1   beer_servings                 193 non-null    int64
 2   spirit_servings               193 non-null    int64
 3   wine_servings                 193 non-null    int64
 4   total_litres_of_pure_alcohol  193 non-null    float64
 5   continent                     193 non-null    object
dtypes: float64(1), int64(3), object(2)
memory usage: 30.5 KB
```

[193]:
```
# calculate the memory usage for each Series (in bytes) without checking the␣
 ↪object references
drinks.memory_usage()
```

[193]:
```
Index                           132
country                        1544
beer_servings                  1544
spirit_servings                1544
wine_servings                  1544
total_litres_of_pure_alcohol   1544
continent                      1544
dtype: int64
```

[194]:
```
# calculate the true memory usage for each Series
drinks.memory_usage(deep=True)
```

[194]:
```
Index                           132
country                       12588
beer_servings                  1544
spirit_servings                1544
wine_servings                  1544
total_litres_of_pure_alcohol   1544
continent                     12332
dtype: int64
```

Documentation for **info** and **memory_usage**

[195]:
```
# use the 'category' data type to store the 'continent' strings as integers
drinks['continent'] = drinks.continent.astype('category')
drinks.dtypes
```

[195]:
```
country                          object
beer_servings                     int64
spirit_servings                   int64
wine_servings                     int64
total_litres_of_pure_alcohol    float64
continent                      category
dtype: object
```

Documentation for **astype**

[196]:
```
# 'continent' Series appears to be unchanged
drinks.continent.head()
```

[196]:
```
0      Asia
1    Europe
2    Africa
3    Europe
4    Africa
Name: continent, dtype: category
Categories (6, object): ['Africa', 'Asia', 'Europe', 'North America', 'Oceania',
'South America']
```

[197]:
```
# strings are now encoded (0 means 'Africa', 1 means 'Asia', 2 means 'Europe',␣
↪etc.)
drinks.continent.cat.codes.head()
```

[197]:
```
0    1
1    2
2    0
3    2
4    0
```

```
dtype: int8
```

[198]:
```
# memory usage has been drastically reduced
drinks.memory_usage(deep=True)
```

[198]:
```
Index                            132
country                        12588
beer_servings                   1544
spirit_servings                 1544
wine_servings                   1544
total_litres_of_pure_alcohol    1544
continent                        756
dtype: int64
```

[199]:
```
# repeat this process for the 'country' Series
drinks['country'] = drinks.country.astype('category')
drinks.memory_usage(deep=True)
```

[199]:
```
Index                            132
country                        17142
beer_servings                   1544
spirit_servings                 1544
wine_servings                   1544
total_litres_of_pure_alcohol    1544
continent                        756
dtype: int64
```

[200]:
```
# memory usage increased because we created 193 categories
drinks.country.cat.categories
```

[200]:
```
Index(['Afghanistan', 'Albania', 'Algeria', 'Andorra', 'Angola',
       'Antigua & Barbuda', 'Argentina', 'Armenia', 'Australia', 'Austria',
       …
       'United Arab Emirates', 'United Kingdom', 'Uruguay', 'Uzbekistan',
       'Vanuatu', 'Venezuela', 'Vietnam', 'Yemen', 'Zambia', 'Zimbabwe'],
      dtype='object', length=193)
```

The **category** data type should only be used with a string Series that has a **small number of possible values**.

[201]:
```
# create a small DataFrame from a dictionary
df = pd.DataFrame({'ID':[100, 101, 102, 103], 'quality':['good', 'very good',␣
 ↪'good', 'excellent']})
df
```

[201]:

	ID	quality
0	100	good
1	101	very good

```
2  102       good
3  103  excellent
```

[202]:
```
# sort the DataFrame by the 'quality' Series (alphabetical order)
df.sort_values('quality')
```

[202]:
```
    ID    quality
3  103  excellent
0  100       good
2  102       good
1  101  very good
```

[203]:
```
# define a logical ordering for the categories
cats = pd.CategoricalDtype(categories=['good', 'very good', 'excellent'],␣
 ↪ordered=True)
df['quality'] = df.quality.astype(cats)
df.quality
```

[203]:
```
0         good
1    very good
2         good
3    excellent
Name: quality, dtype: category
Categories (3, object): ['good' < 'very good' < 'excellent']
```

[204]:
```
# sort the DataFrame by the 'quality' Series (logical order)
df.sort_values('quality')
```

[204]:
```
    ID    quality
0  100       good
2  102       good
1  101  very good
3  103  excellent
```

[205]:
```
# comparison operators work with ordered categories
df.loc[df.quality > 'good', :]
```

[205]:
```
    ID    quality
1  101  very good
3  103  excellent
```

Categorical data from the pandas user guide

Categorical accessor section of the pandas API reference

[Back to top]

1.22 Day 22: How do I use pandas with scikit-learn to create Kaggle submissions?

[206]:
```
# read the training dataset from Kaggle's Titanic competition into a DataFrame
train = pd.read_csv('http://bit.ly/kaggletrain')
train.head()
```

[206]:
```
   PassengerId  Survived  Pclass  \
0            1         0       3
1            2         1       1
2            3         1       3
3            4         1       1
4            5         0       3

                                                Name     Sex   Age  SibSp  \
0                            Braund, Mr. Owen Harris    male  22.0      1
1  Cumings, Mrs. John Bradley (Florence Briggs Th…  female  38.0      1
2                             Heikkinen, Miss. Laina  female  26.0      0
3       Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0      1
4                           Allen, Mr. William Henry    male  35.0      0

   Parch            Ticket     Fare Cabin Embarked
0      0         A/5 21171   7.2500   NaN        S
1      0          PC 17599  71.2833   C85        C
2      0  STON/O2. 3101282   7.9250   NaN        S
3      0            113803  53.1000  C123        S
4      0            373450   8.0500   NaN        S
```

Kaggle's Titanic competition

[207]:
```
# create a feature matrix 'X' by selecting two DataFrame columns
feature_cols = ['Pclass', 'Parch']
X = train.loc[:, feature_cols]
X.shape
```

[207]:
```
(891, 2)
```

[208]:
```
# create a response vector 'y' by selecting a Series
y = train.Survived
y.shape
```

[208]:
```
(891,)
```

Note: There is no need to convert these pandas objects to NumPy arrays.

[209]:
```
# fit a classification model to the training data
from sklearn.linear_model import LogisticRegression
logreg = LogisticRegression()
logreg.fit(X, y)
```

[209]:
```
LogisticRegression()
```

Introduction to Machine Learning with scikit-learn (Data School course)

[210]:
```
# read the testing dataset from Kaggle's Titanic competition into a DataFrame
test = pd.read_csv('http://bit.ly/kaggletest')
test.head()
```

[210]:

	PassengerId	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	892	3	Kelly, Mr. James	male	34.5	0	0	330911	7.8292	NaN	Q
1	893	3	Wilkes, Mrs. James (Ellen Needs)	female	47.0	1	0	363272	7.0000	NaN	S
2	894	2	Myles, Mr. Thomas Francis	male	62.0	0	0	240276	9.6875	NaN	Q
3	895	3	Wirz, Mr. Albert	male	27.0	0	0	315154	8.6625	NaN	S
4	896	3	Hirvonen, Mrs. Alexander (Helga E Lindqvist)	female	22.0	1	1	3101298	12.2875	NaN	S

[211]:
```
# create a feature matrix from the testing data that matches the training data
X_new = test.loc[:, feature_cols]
X_new.shape
```

[211]:
```
(418, 2)
```

[212]:
```
# use the fitted model to make predictions for the testing set observations
new_pred_class = logreg.predict(X_new)
```

[213]:
```
# create a DataFrame of passenger IDs and testing set predictions
pd.DataFrame({'PassengerId':test.PassengerId, 'Survived':new_pred_class})
```

[213]:

	PassengerId	Survived
0	892	0
1	893	0
2	894	0
3	895	0
4	896	0
..	…	…
413	1305	0
414	1306	1
415	1307	0
416	1308	0
417	1309	0

[418 rows x 2 columns]

Documentation for **DataFrame** constructor

[214]:
```
# ensure that PassengerID is the first column by setting it as the index
pd.DataFrame({'PassengerId':test.PassengerId, 'Survived':new_pred_class}).
  ↪set_index('PassengerId')
```

[214]:
```
             Survived
PassengerId
892                 0
893                 0
894                 0
895                 0
896                 0
…                   …
1305                0
1306                1
1307                0
1308                0
1309                0

[418 rows x 1 columns]
```

[215]:
```
# write the DataFrame to a CSV file that can be submitted to Kaggle
pd.DataFrame({'PassengerId':test.PassengerId, 'Survived':new_pred_class}).
  ↪set_index('PassengerId').to_csv('sub.csv')
```

Documentation for **to_csv**

[216]:
```
# save a DataFrame to disk ("pickle it")
train.to_pickle('train.pkl')
```

[217]:
```
# read a pickled object from disk ("unpickle it")
pd.read_pickle('train.pkl').head()
```

[217]:
```
   PassengerId  Survived  Pclass  \
0            1         0       3
1            2         1       1
2            3         1       3
3            4         1       1
4            5         0       3

                                                Name     Sex   Age  SibSp  \
0                            Braund, Mr. Owen Harris    male  22.0      1
1  Cumings, Mrs. John Bradley (Florence Briggs Th…  female  38.0      1
2                             Heikkinen, Miss. Laina  female  26.0      0
3       Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0      1
4                           Allen, Mr. William Henry    male  35.0      0
```

```
   Parch            Ticket     Fare Cabin Embarked
0      0         A/5 21171   7.2500   NaN        S
1      0          PC 17599  71.2833   C85        C
2      0  STON/O2. 3101282   7.9250   NaN        S
3      0            113803  53.1000  C123        S
4      0            373450   8.0500   NaN        S
```

Documentation for **to_pickle** and **read_pickle**

[Back to top]

1.23 Day 23: More of your pandas questions answered!

Question: Could you explain how to read the pandas documentation?

pandas API reference

Question: What is the difference between **ufo.isnull()** and **pd.isnull(ufo)**?

[218]:
```
# read a dataset of UFO reports into a DataFrame
ufo = pd.read_csv('http://bit.ly/uforeports')
ufo.head()
```

[218]:
```
                  City Colors Reported Shape Reported State              Time
0               Ithaca             NaN       TRIANGLE    NY   6/1/1930 22:00
1          Willingboro             NaN          OTHER    NJ  6/30/1930 20:00
2              Holyoke             NaN           OVAL    CO  2/15/1931 14:00
3              Abilene             NaN           DISK    KS   6/1/1931 13:00
4  New York Worlds Fair             NaN          LIGHT    NY  4/18/1933 19:00
```

[219]:
```
# use 'isna' as a top-level function
pd.isna(ufo).head()
```

[219]:
```
    City  Colors Reported  Shape Reported  State   Time
0  False             True           False  False  False
1  False             True           False  False  False
2  False             True           False  False  False
3  False             True           False  False  False
4  False             True           False  False  False
```

[220]:
```
# equivalent: use 'isna' as a DataFrame method
ufo.isna().head()
```

[220]:
```
    City  Colors Reported  Shape Reported  State   Time
0  False             True           False  False  False
1  False             True           False  False  False
2  False             True           False  False  False
3  False             True           False  False  False
4  False             True           False  False  False
```

Documentation for **isna**

Question: Why are DataFrame slices inclusive when using `.loc`, but exclusive when using `.iloc`?

[221]:
```
# label-based slicing is inclusive of the start and stop
ufo.loc[0:4, :]
```

[221]:
```
                  City Colors Reported Shape Reported State              Time
0                Ithaca              NaN       TRIANGLE    NY    6/1/1930 22:00
1           Willingboro              NaN          OTHER    NJ   6/30/1930 20:00
2               Holyoke              NaN           OVAL    CO   2/15/1931 14:00
3               Abilene              NaN           DISK    KS    6/1/1931 13:00
4  New York Worlds Fair              NaN          LIGHT    NY   4/18/1933 19:00
```

[222]:
```
# position-based slicing is inclusive of the start and exclusive of the stop
ufo.iloc[0:4, :]
```

[222]:
```
           City Colors Reported Shape Reported State              Time
0        Ithaca              NaN       TRIANGLE    NY    6/1/1930 22:00
1  Willingboro              NaN          OTHER    NJ   6/30/1930 20:00
2      Holyoke              NaN           OVAL    CO   2/15/1931 14:00
3      Abilene              NaN           DISK    KS    6/1/1931 13:00
```

Documentation for `loc` and `iloc`

[223]:
```
# 'iloc' is simply following NumPy's slicing convention...
ufo.to_numpy()[0:4, :]
```

[223]:
```
array([['Ithaca', nan, 'TRIANGLE', 'NY', '6/1/1930 22:00'],
       ['Willingboro', nan, 'OTHER', 'NJ', '6/30/1930 20:00'],
       ['Holyoke', nan, 'OVAL', 'CO', '2/15/1931 14:00'],
       ['Abilene', nan, 'DISK', 'KS', '6/1/1931 13:00']], dtype=object)
```

[224]:
```
# ...and NumPy is simply following Python's convention
list(range(0, 4))
```

[224]:
```
[0, 1, 2, 3]
```

[225]:
```
# 'loc' is inclusive of the stopping label because you don't necessarily know
  ↪what label will come after it
ufo.loc[0:4, 'City':'State']
```

[225]:
```
                  City Colors Reported Shape Reported State
0                Ithaca              NaN       TRIANGLE    NY
1           Willingboro              NaN          OTHER    NJ
2               Holyoke              NaN           OVAL    CO
3               Abilene              NaN           DISK    KS
4  New York Worlds Fair              NaN          LIGHT    NY
```

Question: How do I randomly sample rows from a DataFrame?

[226]:
```
# sample 3 rows from the DataFrame without replacement
ufo.sample(n=3)
```

[226]:
```
            City Colors Reported Shape Reported State              Time
10938      Sodus              NaN          CIGAR    NY  11/12/1997 0:03
13120   Rockford              NaN            NaN    IL   3/1/1999 22:00
14005  Bay Point              RED          OTHER    CA  7/22/1999 20:00
```

Documentation for **sample**

[227]:
```
# use the 'random_state' parameter for reproducibility
ufo.sample(n=3, random_state=42)
```

[227]:
```
               City Colors Reported Shape Reported State               Time
217    Norridgewock              NaN           DISK    ME   9/15/1952 14:00
12282         Ipava              NaN       TRIANGLE    IL   10/1/1998 21:15
17933     Ellinwood              NaN       FIREBALL    KS  11/13/2000 22:00
```

[228]:
```
# sample 75% of the DataFrame's rows without replacement
train = ufo.sample(frac=0.75, random_state=99)
```

[229]:
```
# store the remaining 25% of the rows in another DataFrame
test = ufo.loc[~ufo.index.isin(train.index), :]
```

Documentation for **isin**

[Back to top]

1.24 Day 24: How do I create dummy variables in pandas?

[230]:
```
# read the training dataset from Kaggle's Titanic competition
train = pd.read_csv('http://bit.ly/kaggletrain')
train.head()
```

[230]:
```
   PassengerId  Survived  Pclass  \
0            1         0       3
1            2         1       1
2            3         1       3
3            4         1       1
4            5         0       3

                                                Name     Sex   Age  SibSp  \
0                            Braund, Mr. Owen Harris    male  22.0      1
1  Cumings, Mrs. John Bradley (Florence Briggs Th…  female  38.0      1
2                             Heikkinen, Miss. Laina  female  26.0      0
3       Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0      1
4                           Allen, Mr. William Henry    male  35.0      0

   Parch            Ticket     Fare Cabin Embarked
```

```
0      0         A/5 21171   7.2500   NaN        S
1      0          PC 17599  71.2833   C85        C
2      0  STON/O2. 3101282   7.9250   NaN        S
3      0            113803  53.1000  C123        S
4      0            373450   8.0500   NaN        S
```

[231]:
```
# create the 'Sex_male' dummy variable using the 'map' method
train['Sex_male'] = train.Sex.map({'female':0, 'male':1})
train.head()
```

[231]:
```
   PassengerId  Survived  Pclass  \
0            1         0       3
1            2         1       1
2            3         1       3
3            4         1       1
4            5         0       3

                                                Name     Sex   Age  SibSp  \
0                            Braund, Mr. Owen Harris    male  22.0      1
1  Cumings, Mrs. John Bradley (Florence Briggs Th…  female  38.0      1
2                             Heikkinen, Miss. Laina  female  26.0      0
3       Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0      1
4                           Allen, Mr. William Henry    male  35.0      0

   Parch            Ticket     Fare Cabin Embarked  Sex_male
0      0         A/5 21171   7.2500   NaN        S         1
1      0          PC 17599  71.2833   C85        C         0
2      0  STON/O2. 3101282   7.9250   NaN        S         0
3      0            113803  53.1000  C123        S         0
4      0            373450   8.0500   NaN        S         1
```

Documentation for **map**

[232]:
```
# alternative: use 'get_dummies' to create one column for every possible value
pd.get_dummies(train.Sex, dtype=int).head()
```

[232]:
```
   female  male
0       0     1
1       1     0
2       1     0
3       1     0
4       0     1
```

Documentation for **get_dummies**

Generally speaking:

- If you have **"K" possible values** for a categorical feature, you only need **"K-1" dummy variables** to capture all of the information about that feature.

- One convention is to **drop the first dummy variable**, which defines that level as the "baseline".

[233]:
```
# drop the first dummy variable ('female') using the 'iloc' method
pd.get_dummies(train.Sex, dtype=int).iloc[:, 1:].head()
```

[233]:
```
   male
0     1
1     0
2     0
3     0
4     1
```

[234]:
```
# add a prefix to identify the source of the dummy variables
pd.get_dummies(train.Sex, prefix='Sex', dtype=int).iloc[:, 1:].head()
```

[234]:
```
   Sex_male
0         1
1         0
2         0
3         0
4         1
```

[235]:
```
# use 'get_dummies' with a feature that has 3 possible values
pd.get_dummies(train.Embarked, prefix='Embarked', dtype=int).head(10)
```

[235]:
```
   Embarked_C  Embarked_Q  Embarked_S
0           0           0           1
1           1           0           0
2           0           0           1
3           0           0           1
4           0           0           1
5           0           1           0
6           0           0           1
7           0           0           1
8           0           0           1
9           1           0           0
```

[236]:
```
# drop the first dummy variable ('C')
pd.get_dummies(train.Embarked, prefix='Embarked', dtype=int).iloc[:, 1:].
  ↪head(10)
```

[236]:
```
   Embarked_Q  Embarked_S
0           0           1
1           0           0
2           0           1
3           0           1
4           0           1
```

```
5           1           0
6           0           1
7           0           1
8           0           1
9           0           0
```

How to translate these values back to the original 'Embarked' value:

- **0, 0** means **C**
- **1, 0** means **Q**
- **0, 1** means **S**

[237]:
```
# save the DataFrame of dummy variables and concatenate them to the original␣
↪DataFrame
embarked_dummies = pd.get_dummies(train.Embarked, prefix='Embarked', dtype=int).
↪iloc[:, 1:]
train = pd.concat([train, embarked_dummies], axis=1)
train.head()
```

[237]:
```
   PassengerId  Survived  Pclass  \
0            1         0       3
1            2         1       1
2            3         1       3
3            4         1       1
4            5         0       3

                                                Name     Sex   Age  SibSp  \
0                            Braund, Mr. Owen Harris    male  22.0      1
1  Cumings, Mrs. John Bradley (Florence Briggs Th…  female  38.0      1
2                             Heikkinen, Miss. Laina  female  26.0      0
3       Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0      1
4                           Allen, Mr. William Henry    male  35.0      0

   Parch            Ticket     Fare Cabin Embarked  Sex_male  Embarked_Q  \
0      0         A/5 21171   7.2500   NaN        S         1           0
1      0          PC 17599  71.2833   C85        C         0           0
2      0  STON/O2. 3101282   7.9250   NaN        S         0           0
3      0            113803  53.1000  C123        S         0           0
4      0            373450   8.0500   NaN        S         1           0

   Embarked_S
0           1
1           0
2           1
3           1
4           1
```

Documentation for **concat**

[238]:
```
# reset the DataFrame
train = pd.read_csv('http://bit.ly/kaggletrain')
train.head()
```

[238]:
```
   PassengerId  Survived  Pclass  \
0            1         0       3
1            2         1       1
2            3         1       3
3            4         1       1
4            5         0       3

                                                Name     Sex   Age  SibSp  \
0                            Braund, Mr. Owen Harris    male  22.0      1
1  Cumings, Mrs. John Bradley (Florence Briggs Th…  female  38.0      1
2                             Heikkinen, Miss. Laina  female  26.0      0
3       Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0      1
4                           Allen, Mr. William Henry    male  35.0      0

   Parch            Ticket     Fare Cabin Embarked
0      0         A/5 21171   7.2500   NaN        S
1      0          PC 17599  71.2833   C85        C
2      0  STON/O2. 3101282   7.9250   NaN        S
3      0            113803  53.1000  C123        S
4      0            373450   8.0500   NaN        S
```

[239]:
```
# pass the DataFrame to 'get_dummies' and specify which columns to dummy (it␣
↪drops the original columns)
pd.get_dummies(train, columns=['Sex', 'Embarked'], dtype=int).head()
```

[239]:
```
   PassengerId  Survived  Pclass  \
0            1         0       3
1            2         1       1
2            3         1       3
3            4         1       1
4            5         0       3

                                                Name   Age  SibSp  Parch  \
0                            Braund, Mr. Owen Harris  22.0      1      0
1  Cumings, Mrs. John Bradley (Florence Briggs Th…  38.0      1      0
2                             Heikkinen, Miss. Laina  26.0      0      0
3       Futrelle, Mrs. Jacques Heath (Lily May Peel)  35.0      1      0
4                           Allen, Mr. William Henry  35.0      0      0

            Ticket     Fare Cabin  Sex_female  Sex_male  Embarked_C  \
0        A/5 21171   7.2500   NaN           0         1           0
1         PC 17599  71.2833   C85           1         0           1
2  STON/O2. 3101282   7.9250   NaN           1         0           0
```

```
3            113803  53.1000  C123           1         0           0
4            373450   8.0500   NaN           0         1           0

   Embarked_Q  Embarked_S
0           0           1
1           0           0
2           0           1
3           0           1
4           0           1
```

[240]:
```
# use the 'drop_first' parameter to drop the first dummy variable for each␣
↪feature
pd.get_dummies(train, columns=['Sex', 'Embarked'], drop_first=True, dtype=int).
↪head()
```

[240]:
```
   PassengerId  Survived  Pclass  \
0            1         0       3
1            2         1       1
2            3         1       3
3            4         1       1
4            5         0       3

                                                Name   Age  SibSp  Parch  \
0                            Braund, Mr. Owen Harris  22.0      1      0
1  Cumings, Mrs. John Bradley (Florence Briggs Th…  38.0      1      0
2                             Heikkinen, Miss. Laina  26.0      0      0
3       Futrelle, Mrs. Jacques Heath (Lily May Peel)  35.0      1      0
4                           Allen, Mr. William Henry  35.0      0      0

             Ticket     Fare Cabin  Sex_male  Embarked_Q  Embarked_S
0         A/5 21171   7.2500   NaN         1           0           1
1          PC 17599  71.2833   C85         0           0           0
2  STON/O2. 3101282   7.9250   NaN         0           0           1
3            113803  53.1000  C123         0           0           1
4            373450   8.0500   NaN         1           0           1
```

Introduction to Machine Learning with scikit-learn (Data School course)

[Back to top]

1.25 Day 25: How do I work with dates and times in pandas?

[241]:
```
# read a dataset of UFO reports into a DataFrame
ufo = pd.read_csv('http://bit.ly/uforeports')
ufo.head()
```

[241]:
```
            City Colors Reported Shape Reported State              Time
0         Ithaca              NaN       TRIANGLE    NY   6/1/1930 22:00
```

```
1           Willingboro              NaN          OTHER    NJ  6/30/1930 20:00
2               Holyoke              NaN           OVAL    CO  2/15/1931 14:00
3               Abilene              NaN           DISK    KS   6/1/1931 13:00
4  New York Worlds Fair              NaN          LIGHT    NY  4/18/1933 19:00
```

[242]:
```
# 'Time' is currently stored as a string
ufo.dtypes
```

[242]:
```
City               object
Colors Reported    object
Shape Reported     object
State              object
Time               object
dtype: object
```

[243]:
```
# hour could be accessed using string slicing, but this approach breaks too␣
↪easily
ufo.Time.str.slice(-5, -3).astype(int).head()
```

[243]:
```
0    22
1    20
2    14
3    13
4    19
Name: Time, dtype: int64
```

[244]:
```
# convert 'Time' to datetime format
ufo['Time'] = pd.to_datetime(ufo.Time)
ufo.head()
```

[244]:
```
                  City Colors Reported Shape Reported State  \
0               Ithaca              NaN       TRIANGLE    NY
1           Willingboro              NaN          OTHER    NJ
2               Holyoke              NaN           OVAL    CO
3               Abilene              NaN           DISK    KS
4  New York Worlds Fair              NaN          LIGHT    NY

                 Time
0 1930-06-01 22:00:00
1 1930-06-30 20:00:00
2 1931-02-15 14:00:00
3 1931-06-01 13:00:00
4 1933-04-18 19:00:00
```

[245]:
```
ufo.dtypes
```

[245]:
```
City                          object
Colors Reported               object
Shape Reported                object
State                         object
Time                  datetime64[ns]
dtype: object
```

Documentation for **to_datetime**

[246]:
```
# convenient Series attributes are now available
ufo.Time.dt.hour.head()
```

[246]:
```
0    22
1    20
2    14
3    13
4    19
Name: Time, dtype: int32
```

[247]:
```
# note that day_name is a method, not an attribute
ufo.Time.dt.day_name().head()
```

[247]:
```
0     Sunday
1     Monday
2     Sunday
3     Monday
4    Tuesday
Name: Time, dtype: object
```

[248]:
```
ufo.Time.dt.weekday.head()
```

[248]:
```
0    6
1    0
2    6
3    0
4    1
Name: Time, dtype: int32
```

[249]:
```
ufo.Time.dt.dayofyear.head()
```

[249]:
```
0    152
1    181
2     46
3    152
4    108
Name: Time, dtype: int32
```

Datetime accessor section of the pandas API reference

[250]:
```
# convert a single string to datetime format (outputs a timestamp object)
ts = pd.to_datetime('1/1/1999')
ts
```

[250]:
```
Timestamp('1999-01-01 00:00:00')
```

[251]:
```
# compare a datetime Series with a timestamp
ufo.loc[ufo.Time >= ts, :].head()
```

[251]:
```
                   City Colors Reported Shape Reported State  \
12832          Loma Rica              NaN           LIGHT    CA
12833            Bauxite              NaN             NaN    AR
12834           Florence              NaN        CYLINDER    SC
12835       Lake Henshaw              NaN           CIGAR    CA
12836  Wilmington Island              NaN           LIGHT    GA

                     Time
12832 1999-01-01 02:30:00
12833 1999-01-01 03:00:00
12834 1999-01-01 14:00:00
12835 1999-01-01 15:00:00
12836 1999-01-01 17:15:00
```

[252]:
```
# perform mathematical operations with timestamps (outputs a timedelta object)
ufo.Time.max() - ufo.Time.min()
```

[252]:
```
Timedelta('25781 days 01:59:00')
```

[253]:
```
# timedelta objects also have attributes you can access
(ufo.Time.max() - ufo.Time.min()).days
```

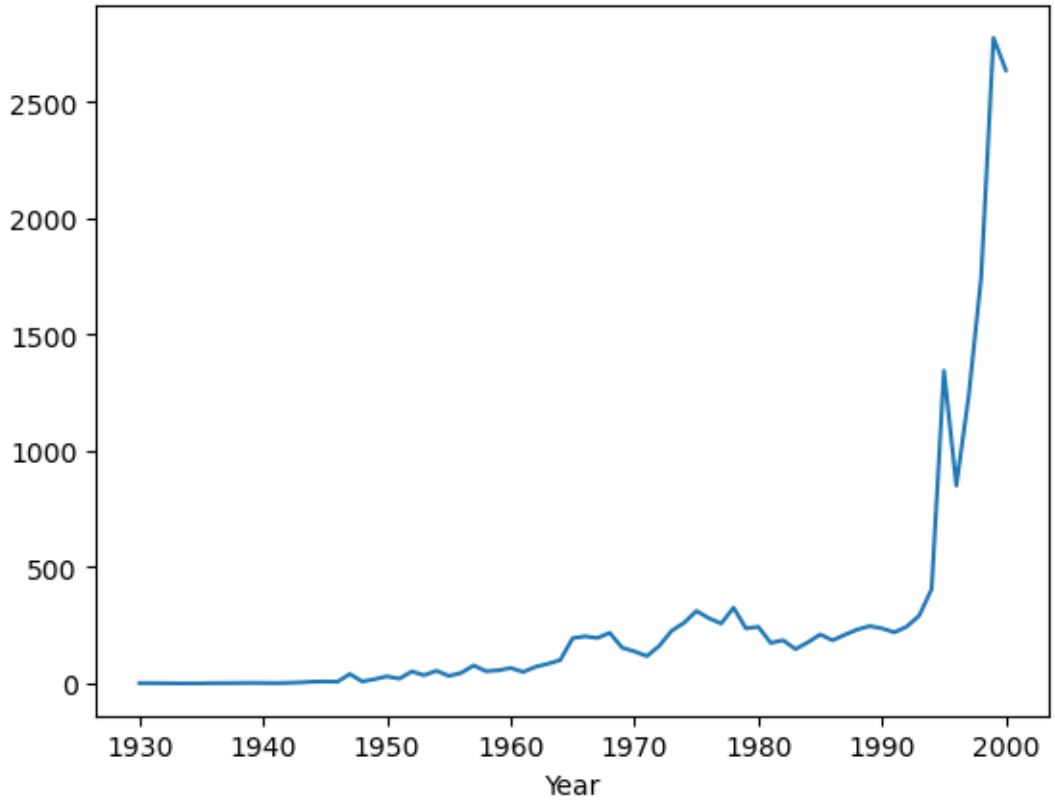
[253]:
```
25781
```

[254]:
```
# count the number of UFO reports per year
ufo['Year'] = ufo.Time.dt.year
ufo.Year.value_counts().sort_index().head()
```

[254]:
```
Year
1930    2
1931    2
1933    1
1934    1
1935    1
Name: count, dtype: int64
```

[255]:
```
# plot the number of UFO reports per year (line plot is the default)
ufo.Year.value_counts().sort_index().plot()
```

[255]: <Axes: xlabel='Year'>



[Back to top]

1.26 Day 26: How do I find and remove duplicate rows in pandas?

[256]:
```
# read a dataset of movie reviewers into a DataFrame
user_cols = ['user_id', 'age', 'gender', 'occupation', 'zip_code']
users = pd.read_table('http://bit.ly/movieusers', sep='|', header=None,␣
↪names=user_cols, index_col='user_id')
users.head()
```

[256]:

user_id	age	gender	occupation	zip_code
1	24	M	technician	85711
2	53	F	other	94043
3	23	M	writer	32067
4	24	M	technician	43537
5	33	F	other	15213

[257]:
```
users.shape
```

[257]: (943, 4)

[258]:
```
# detect duplicate zip codes: True if an item is identical to a previous item
users.zip_code.duplicated()
```

[258]:
```
user_id
1      False
2      False
3      False
4      False
5      False
       …
939    False
940     True
941    False
942    False
943    False
Name: zip_code, Length: 943, dtype: bool
```

[259]:
```
# count the duplicate items (True becomes 1, False becomes 0)
users.zip_code.duplicated().sum()
```

[259]: 148

Documentation for **duplicated** Series method

[260]:
```
# detect duplicate DataFrame rows: True if an entire row is identical to a␣
 ↪previous row
users.duplicated()
```

[260]:
```
user_id
1      False
2      False
3      False
4      False
5      False
       …
939    False
940    False
941    False
942    False
943    False
Length: 943, dtype: bool
```

[261]:
```
# count the duplicate rows
users.duplicated().sum()
```

[261]: 7

Documentation for **duplicated** DataFrame method

Logic for **duplicated**:

- **keep='first'** (default): Mark duplicates as True except for the first occurrence.
- **keep='last'**: Mark duplicates as True except for the last occurrence.
- **keep=False**: Mark all duplicates as True.

[262]:
```
# examine the duplicate rows (ignoring the first occurrence)
users.loc[users.duplicated(keep='first'), :]
```

[262]:

user_id	age	gender	occupation	zip_code
496	21	F	student	55414
572	51	M	educator	20003
621	17	M	student	60402
684	28	M	student	55414
733	44	F	other	60630
805	27	F	other	20009
890	32	M	student	97301

[263]:
```
# examine the duplicate rows (ignoring the last occurrence)
users.loc[users.duplicated(keep='last'), :]
```

[263]:

user_id	age	gender	occupation	zip_code
67	17	M	student	60402
85	51	M	educator	20003
198	21	F	student	55414
350	32	M	student	97301
428	28	M	student	55414
437	27	F	other	20009
460	44	F	other	60630

[264]:
```
# examine the duplicate rows (including all duplicates)
users.loc[users.duplicated(keep=False), :]
```

[264]:

user_id	age	gender	occupation	zip_code
67	17	M	student	60402
85	51	M	educator	20003
198	21	F	student	55414
350	32	M	student	97301
428	28	M	student	55414
437	27	F	other	20009
460	44	F	other	60630
496	21	F	student	55414
572	51	M	educator	20003
621	17	M	student	60402

```
684        28       M    student    55414
733        44       F      other    60630
805        27       F      other    20009
890        32       M    student    97301
```

[265]:
```
# drop the duplicate rows (inplace=False by default)
users.drop_duplicates(keep='first').shape
```

[265]:
```
(936, 4)
```

[266]:
```
users.drop_duplicates(keep='last').shape
```

[266]:
```
(936, 4)
```

[267]:
```
users.drop_duplicates(keep=False).shape
```

[267]:
```
(929, 4)
```

Documentation for **drop_duplicates**

[268]:
```
# only consider a subset of columns when identifying duplicates
users.duplicated(subset=['age', 'zip_code']).sum()
```

[268]:
```
16
```

[269]:
```
users.drop_duplicates(subset=['age', 'zip_code']).shape
```

[269]:
```
(927, 4)
```

[Back to top]

1.27 Day 27: How do I avoid a SettingWithCopyWarning in pandas?

[270]:
```
# read a dataset of top-rated IMDb movies into a DataFrame
movies = pd.read_csv('http://bit.ly/imdbratings')
movies.head()
```

[270]:
```
   star_rating                     title content_rating   genre  duration  \
0          9.3  The Shawshank Redemption              R   Crime       142
1          9.2             The Godfather              R   Crime       175
2          9.1    The Godfather: Part II              R   Crime       200
3          9.0           The Dark Knight          PG-13  Action       152
4          8.9              Pulp Fiction              R   Crime       154

                                        actors_list
0  [u'Tim Robbins', u'Morgan Freeman', u'Bob Gunt…
1    [u'Marlon Brando', u'Al Pacino', u'James Caan']
2  [u'Al Pacino', u'Robert De Niro', u'Robert Duv…
3  [u'Christian Bale', u'Heath Ledger', u'Aaron E…
```

```
4   [u'John Travolta', u'Uma Thurman', u'Samuel L…
```

[271]:
```
# count the missing values in the 'content_rating' Series
movies.content_rating.isna().sum()
```

[271]:
```
3
```

[272]:
```
# examine the DataFrame rows that contain those missing values
movies[movies.content_rating.isna()]
```

[272]:
```
     star_rating                              title content_rating  \
187          8.2  Butch Cassidy and the Sundance Kid            NaN
649          7.7                   Where Eagles Dare            NaN
936          7.4                           True Grit            NaN

          genre  duration                                        actors_list
187  Biography       110  [u'Paul Newman', u'Robert Redford', u'Katharin…
649     Action       158  [u'Richard Burton', u'Clint Eastwood', u'Mary …
936  Adventure       128    [u'John Wayne', u'Kim Darby', u'Glen Campbell']
```

[273]:
```
# examine the unique values in the 'content_rating' Series
movies.content_rating.value_counts()
```

[273]:
```
content_rating
R            460
PG-13        189
PG           123
NOT RATED     65
APPROVED      47
UNRATED       38
G             32
PASSED         7
NC-17          7
X              4
GP             3
TV-MA          1
Name: count, dtype: int64
```

Goal: Mark the 'NOT RATED' values as missing values, represented by 'NaN'.

[274]:
```
# first, locate the relevant rows
movies[movies.content_rating=='NOT RATED']
```

[274]:
```
     star_rating                          title content_rating      genre  \
5            8.9                    12 Angry Men      NOT RATED      Drama
6            8.9  The Good, the Bad and the Ugly      NOT RATED    Western
41           8.5                    Sunset Blvd.      NOT RATED      Drama
63           8.4                               M      NOT RATED      Crime
```

```
66          8.4              Munna Bhai M.B.B.S.      NOT RATED     Comedy
..          …                                …          …        …
665         7.7                           Lolita      NOT RATED      Drama
673         7.7                         Blow-Up       NOT RATED      Drama
763         7.6                           Hunger      NOT RATED  Biography
827         7.5  The Wind That Shakes the Barley      NOT RATED      Drama
899         7.5                      In the Loop      NOT RATED     Comedy

     duration                                          actors_list
5          96  [u'Henry Fonda', u'Lee J. Cobb', u'Martin Bals…
6         161  [u'Clint Eastwood', u'Eli Wallach', u'Lee Van …
41        110  [u'William Holden', u'Gloria Swanson', u'Erich…
63         99  [u'Peter Lorre', u'Ellen Widmann', u'Inge Land…
66        156   [u'Sunil Dutt', u'Sanjay Dutt', u'Arshad Warsi']
..        …                                                 …
665       152  [u'James Mason', u'Shelley Winters', u'Sue Lyon']
673       111  [u'David Hemmings', u'Vanessa Redgrave', u'Sar…
763        96  [u'Stuart Graham', u'Laine Megaw', u'Brian Mil…
827       127  [u'Cillian Murphy', u'Padraic Delaney', u'Liam…
899       106  [u'Tom Hollander', u'Peter Capaldi', u'James G…

[65 rows x 6 columns]
```

[275]:
```
# then, select the 'content_rating' Series from those rows
movies[movies.content_rating=='NOT RATED'].content_rating
```

[275]:
```
5      NOT RATED
6      NOT RATED
41     NOT RATED
63     NOT RATED
66     NOT RATED
         …
665    NOT RATED
673    NOT RATED
763    NOT RATED
827    NOT RATED
899    NOT RATED
Name: content_rating, Length: 65, dtype: object
```

[276]:
```
# finally, replace the 'NOT RATED' values with 'NaN' (imported from NumPy)
import numpy as np
movies[movies.content_rating=='NOT RATED'].content_rating = np.nan
```

```
/var/folders/zn/8p5wr_855bjbd9s6fvn6wd7w0000gn/T/ipykernel_13776/1398416548.py:3
: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  movies[movies.content_rating=='NOT RATED'].content_rating = np.nan
```

Returning a view versus a copy from the pandas user guide

Modern Pandas (Part 1) blog post by Tom Augspurger

Problem: That statement involves two operations, a **__getitem__** and a **__setitem__**. pandas can't guarantee whether the **__getitem__** operation returns a view or a copy of the data.

- If **__getitem__** returns a view of the data, **__setitem__** will affect the 'movies' DataFrame.
- But if **__getitem__** returns a copy of the data, **__setitem__** will not affect the 'movies' DataFrame.

[277]:
```
# the 'content_rating' Series has not changed
movies.content_rating.isna().sum()
```

[277]:
```
3
```

Solution: Use the **loc** accessor, which replaces the 'NOT RATED' values in a single **__setitem__** operation.

[278]:
```
# replace the 'NOT RATED' values with 'NaN' (does not cause a␣
 ↪SettingWithCopyWarning)
movies.loc[movies.content_rating=='NOT RATED', 'content_rating'] = np.nan
```

[279]:
```
# this time, the 'content_rating' Series has changed
movies.content_rating.isna().sum()
```

[279]:
```
68
```

Summary: Use the **loc** accessor any time you are selecting rows and columns in the same statement.

[280]:
```
# create a DataFrame only containing movies with a high 'star_rating'
top_movies = movies.loc[movies.star_rating >= 9, :]
top_movies
```

[280]:
```
   star_rating                     title content_rating   genre  duration  \
0          9.3  The Shawshank Redemption              R   Crime       142
1          9.2             The Godfather              R   Crime       175
2          9.1    The Godfather: Part II              R   Crime       200
3          9.0           The Dark Knight          PG-13  Action       152

                                        actors_list
0  [u'Tim Robbins', u'Morgan Freeman', u'Bob Gunt…
1    [u'Marlon Brando', u'Al Pacino', u'James Caan']
2  [u'Al Pacino', u'Robert De Niro', u'Robert Duv…
3  [u'Christian Bale', u'Heath Ledger', u'Aaron E…
```

Goal: Fix the 'duration' for 'The Shawshank Redemption'.

[281]:
```
# overwrite the relevant cell with the correct duration
top_movies.loc[0, 'duration'] = 150
```

Problem: pandas isn't sure whether 'top_movies' is a view or a copy of 'movies'.

Update from 2024: This is no longer the case.

[282]:
```
# 'top_movies' DataFrame has been updated
top_movies
```

[282]:
```
   star_rating                     title content_rating   genre  duration  \
0          9.3  The Shawshank Redemption              R   Crime       150
1          9.2             The Godfather              R   Crime       175
2          9.1    The Godfather: Part II              R   Crime       200
3          9.0           The Dark Knight          PG-13  Action       152

                                        actors_list
0  [u'Tim Robbins', u'Morgan Freeman', u'Bob Gunt…
1    [u'Marlon Brando', u'Al Pacino', u'James Caan']
2  [u'Al Pacino', u'Robert De Niro', u'Robert Duv…
3  [u'Christian Bale', u'Heath Ledger', u'Aaron E…
```

[283]:
```
# 'movies' DataFrame has not been updated
movies.head(1)
```

[283]:
```
   star_rating                     title content_rating  genre  duration  \
0          9.3  The Shawshank Redemption              R  Crime       142

                                        actors_list
0  [u'Tim Robbins', u'Morgan Freeman', u'Bob Gunt…
```

Solution: Any time you are attempting to create a DataFrame copy, use the `copy` method.

[284]:
```
# explicitly create a copy of 'movies'
top_movies = movies.loc[movies.star_rating >= 9, :].copy()
```

[285]:
```
# pandas now knows that you are updating a copy instead of a view (does not␣
↪cause a SettingWithCopyWarning)
top_movies.loc[0, 'duration'] = 150
```

[286]:
```
# 'top_movies' DataFrame has been updated
top_movies
```

[286]:
```
   star_rating                     title content_rating   genre  duration  \
0          9.3  The Shawshank Redemption              R   Crime       150
1          9.2             The Godfather              R   Crime       175
2          9.1    The Godfather: Part II              R   Crime       200
3          9.0           The Dark Knight          PG-13  Action       152
```

```
                                          actors_list
0  [u'Tim Robbins', u'Morgan Freeman', u'Bob Gunt…
1    [u'Marlon Brando', u'Al Pacino', u'James Caan']
2  [u'Al Pacino', u'Robert De Niro', u'Robert Duv…
3  [u'Christian Bale', u'Heath Ledger', u'Aaron E…
```

[Back to top]

1.28 Day 28: How do I change display options in pandas?

[287]:
```
# read a dataset of alcohol consumption into a DataFrame
drinks = pd.read_csv('http://bit.ly/drinksbycountry')
```

[288]:
```
# only 10 rows will be displayed when printing
drinks
```

[288]:
```
          country  beer_servings  spirit_servings  wine_servings  \
0     Afghanistan              0                0              0
1         Albania             89              132             54
2         Algeria             25                0             14
3         Andorra            245              138            312
4          Angola            217               57             45
..            …              …                …              …
188     Venezuela            333              100              3
189       Vietnam            111                2              1
190         Yemen              6                0              0
191        Zambia             32               19              4
192      Zimbabwe             64               18              4

     total_litres_of_pure_alcohol      continent
0                             0.0           Asia
1                             4.9         Europe
2                             0.7         Africa
3                            12.4         Europe
4                             5.9         Africa
..                            …             …
188                           7.7  South America
189                           2.0           Asia
190                           0.1           Asia
191                           2.5         Africa
192                           4.7         Africa

[193 rows x 6 columns]
```

[289]:
```
# check the current setting for the 'max_rows' option
pd.get_option('display.max_rows')
```

[289]:
```
60
```

Documentation for **get_option**

[290]:
```
# overwrite the current setting so that all rows will be displayed
pd.set_option('display.max_rows', None)
drinks
```

[290]:

	country	beer_servings	spirit_servings	\
0	Afghanistan	0	0	
1	Albania	89	132	
2	Algeria	25	0	
3	Andorra	245	138	
4	Angola	217	57	
5	Antigua & Barbuda	102	128	
6	Argentina	193	25	
7	Armenia	21	179	
8	Australia	261	72	
9	Austria	279	75	
10	Azerbaijan	21	46	
11	Bahamas	122	176	
12	Bahrain	42	63	
13	Bangladesh	0	0	
14	Barbados	143	173	
15	Belarus	142	373	
16	Belgium	295	84	
17	Belize	263	114	
18	Benin	34	4	
19	Bhutan	23	0	
20	Bolivia	167	41	
21	Bosnia-Herzegovina	76	173	
22	Botswana	173	35	
23	Brazil	245	145	
24	Brunei	31	2	
25	Bulgaria	231	252	
26	Burkina Faso	25	7	
27	Burundi	88	0	
28	Cote d'Ivoire	37	1	
29	Cabo Verde	144	56	
30	Cambodia	57	65	
31	Cameroon	147	1	
32	Canada	240	122	
33	Central African Republic	17	2	
34	Chad	15	1	
35	Chile	130	124	
36	China	79	192	
37	Colombia	159	76	
38	Comoros	1	3	
39	Congo	76	1	

40	Cook Islands	0	254
41	Costa Rica	149	87
42	Croatia	230	87
43	Cuba	93	137
44	Cyprus	192	154
45	Czech Republic	361	170
46	North Korea	0	0
47	DR Congo	32	3
48	Denmark	224	81
49	Djibouti	15	44
50	Dominica	52	286
51	Dominican Republic	193	147
52	Ecuador	162	74
53	Egypt	6	4
54	El Salvador	52	69
55	Equatorial Guinea	92	0
56	Eritrea	18	0
57	Estonia	224	194
58	Ethiopia	20	3
59	Fiji	77	35
60	Finland	263	133
61	France	127	151
62	Gabon	347	98
63	Gambia	8	0
64	Georgia	52	100
65	Germany	346	117
66	Ghana	31	3
67	Greece	133	112
68	Grenada	199	438
69	Guatemala	53	69
70	Guinea	9	0
71	Guinea-Bissau	28	31
72	Guyana	93	302
73	Haiti	1	326
74	Honduras	69	98
75	Hungary	234	215
76	Iceland	233	61
77	India	9	114
78	Indonesia	5	1
79	Iran	0	0
80	Iraq	9	3
81	Ireland	313	118
82	Israel	63	69
83	Italy	85	42
84	Jamaica	82	97
85	Japan	77	202
86	Jordan	6	21

87	Kazakhstan	124	246
88	Kenya	58	22
89	Kiribati	21	34
90	Kuwait	0	0
91	Kyrgyzstan	31	97
92	Laos	62	0
93	Latvia	281	216
94	Lebanon	20	55
95	Lesotho	82	29
96	Liberia	19	152
97	Libya	0	0
98	Lithuania	343	244
99	Luxembourg	236	133
100	Madagascar	26	15
101	Malawi	8	11
102	Malaysia	13	4
103	Maldives	0	0
104	Mali	5	1
105	Malta	149	100
106	Marshall Islands	0	0
107	Mauritania	0	0
108	Mauritius	98	31
109	Mexico	238	68
110	Micronesia	62	50
111	Monaco	0	0
112	Mongolia	77	189
113	Montenegro	31	114
114	Morocco	12	6
115	Mozambique	47	18
116	Myanmar	5	1
117	Namibia	376	3
118	Nauru	49	0
119	Nepal	5	6
120	Netherlands	251	88
121	New Zealand	203	79
122	Nicaragua	78	118
123	Niger	3	2
124	Nigeria	42	5
125	Niue	188	200
126	Norway	169	71
127	Oman	22	16
128	Pakistan	0	0
129	Palau	306	63
130	Panama	285	104
131	Papua New Guinea	44	39
132	Paraguay	213	117
133	Peru	163	160

134	Philippines	71	186
135	Poland	343	215
136	Portugal	194	67
137	Qatar	1	42
138	South Korea	140	16
139	Moldova	109	226
140	Romania	297	122
141	Russian Federation	247	326
142	Rwanda	43	2
143	St. Kitts & Nevis	194	205
144	St. Lucia	171	315
145	St. Vincent & the Grenadines	120	221
146	Samoa	105	18
147	San Marino	0	0
148	Sao Tome & Principe	56	38
149	Saudi Arabia	0	5
150	Senegal	9	1
151	Serbia	283	131
152	Seychelles	157	25
153	Sierra Leone	25	3
154	Singapore	60	12
155	Slovakia	196	293
156	Slovenia	270	51
157	Solomon Islands	56	11
158	Somalia	0	0
159	South Africa	225	76
160	Spain	284	157
161	Sri Lanka	16	104
162	Sudan	8	13
163	Suriname	128	178
164	Swaziland	90	2
165	Sweden	152	60
166	Switzerland	185	100
167	Syria	5	35
168	Tajikistan	2	15
169	Thailand	99	258
170	Macedonia	106	27
171	Timor-Leste	1	1
172	Togo	36	2
173	Tonga	36	21
174	Trinidad & Tobago	197	156
175	Tunisia	51	3
176	Turkey	51	22
177	Turkmenistan	19	71
178	Tuvalu	6	41
179	Uganda	45	9
180	Ukraine	206	237

181	United Arab Emirates	16	135
182	United Kingdom	219	126
183	Tanzania	36	6
184	USA	249	158
185	Uruguay	115	35
186	Uzbekistan	25	101
187	Vanuatu	21	18
188	Venezuela	333	100
189	Vietnam	111	2
190	Yemen	6	0
191	Zambia	32	19
192	Zimbabwe	64	18

	wine_servings	total_litres_of_pure_alcohol	continent
0	0	0.0	Asia
1	54	4.9	Europe
2	14	0.7	Africa
3	312	12.4	Europe
4	45	5.9	Africa
5	45	4.9	North America
6	221	8.3	South America
7	11	3.8	Europe
8	212	10.4	Oceania
9	191	9.7	Europe
10	5	1.3	Europe
11	51	6.3	North America
12	7	2.0	Asia
13	0	0.0	Asia
14	36	6.3	North America
15	42	14.4	Europe
16	212	10.5	Europe
17	8	6.8	North America
18	13	1.1	Africa
19	0	0.4	Asia
20	8	3.8	South America
21	8	4.6	Europe
22	35	5.4	Africa
23	16	7.2	South America
24	1	0.6	Asia
25	94	10.3	Europe
26	7	4.3	Africa
27	0	6.3	Africa
28	7	4.0	Africa
29	16	4.0	Africa
30	1	2.2	Asia
31	4	5.8	Africa
32	100	8.2	North America

33	1	1.8	Africa
34	1	0.4	Africa
35	172	7.6	South America
36	8	5.0	Asia
37	3	4.2	South America
38	1	0.1	Africa
39	9	1.7	Africa
40	74	5.9	Oceania
41	11	4.4	North America
42	254	10.2	Europe
43	5	4.2	North America
44	113	8.2	Europe
45	134	11.8	Europe
46	0	0.0	Asia
47	1	2.3	Africa
48	278	10.4	Europe
49	3	1.1	Africa
50	26	6.6	North America
51	9	6.2	North America
52	3	4.2	South America
53	1	0.2	Africa
54	2	2.2	North America
55	233	5.8	Africa
56	0	0.5	Africa
57	59	9.5	Europe
58	0	0.7	Africa
59	1	2.0	Oceania
60	97	10.0	Europe
61	370	11.8	Europe
62	59	8.9	Africa
63	1	2.4	Africa
64	149	5.4	Europe
65	175	11.3	Europe
66	10	1.8	Africa
67	218	8.3	Europe
68	28	11.9	North America
69	2	2.2	North America
70	2	0.2	Africa
71	21	2.5	Africa
72	1	7.1	South America
73	1	5.9	North America
74	2	3.0	North America
75	185	11.3	Europe
76	78	6.6	Europe
77	0	2.2	Asia
78	0	0.1	Asia
79	0	0.0	Asia

80	0	0.2	Asia
81	165	11.4	Europe
82	9	2.5	Asia
83	237	6.5	Europe
84	9	3.4	North America
85	16	7.0	Asia
86	1	0.5	Asia
87	12	6.8	Asia
88	2	1.8	Africa
89	1	1.0	Oceania
90	0	0.0	Asia
91	6	2.4	Asia
92	123	6.2	Asia
93	62	10.5	Europe
94	31	1.9	Asia
95	0	2.8	Africa
96	2	3.1	Africa
97	0	0.0	Africa
98	56	12.9	Europe
99	271	11.4	Europe
100	4	0.8	Africa
101	1	1.5	Africa
102	0	0.3	Asia
103	0	0.0	Asia
104	1	0.6	Africa
105	120	6.6	Europe
106	0	0.0	Oceania
107	0	0.0	Africa
108	18	2.6	Africa
109	5	5.5	North America
110	18	2.3	Oceania
111	0	0.0	Europe
112	8	4.9	Asia
113	128	4.9	Europe
114	10	0.5	Africa
115	5	1.3	Africa
116	0	0.1	Asia
117	1	6.8	Africa
118	8	1.0	Oceania
119	0	0.2	Asia
120	190	9.4	Europe
121	175	9.3	Oceania
122	1	3.5	North America
123	1	0.1	Africa
124	2	9.1	Africa
125	7	7.0	Oceania
126	129	6.7	Europe

127	1	0.7	Asia
128	0	0.0	Asia
129	23	6.9	Oceania
130	18	7.2	North America
131	1	1.5	Oceania
132	74	7.3	South America
133	21	6.1	South America
134	1	4.6	Asia
135	56	10.9	Europe
136	339	11.0	Europe
137	7	0.9	Asia
138	9	9.8	Asia
139	18	6.3	Europe
140	167	10.4	Europe
141	73	11.5	Asia
142	0	6.8	Africa
143	32	7.7	North America
144	71	10.1	North America
145	11	6.3	North America
146	24	2.6	Oceania
147	0	0.0	Europe
148	140	4.2	Africa
149	0	0.1	Asia
150	7	0.3	Africa
151	127	9.6	Europe
152	51	4.1	Africa
153	2	6.7	Africa
154	11	1.5	Asia
155	116	11.4	Europe
156	276	10.6	Europe
157	1	1.2	Oceania
158	0	0.0	Africa
159	81	8.2	Africa
160	112	10.0	Europe
161	0	2.2	Asia
162	0	1.7	Africa
163	7	5.6	South America
164	2	4.7	Africa
165	186	7.2	Europe
166	280	10.2	Europe
167	16	1.0	Asia
168	0	0.3	Asia
169	1	6.4	Asia
170	86	3.9	Europe
171	4	0.1	Asia
172	19	1.3	Africa
173	5	1.1	Oceania

```
174              7                          6.4  North America
175             20                          1.3         Africa
176              7                          1.4           Asia
177             32                          2.2           Asia
178              9                          1.0        Oceania
179              0                          8.3         Africa
180             45                          8.9         Europe
181              5                          2.8           Asia
182            195                         10.4         Europe
183              1                          5.7         Africa
184             84                          8.7  North America
185            220                          6.6  South America
186              8                          2.4           Asia
187             11                          0.9        Oceania
188              3                          7.7  South America
189              1                          2.0           Asia
190              0                          0.1           Asia
191              4                          2.5         Africa
192              4                          4.7         Africa
```

[291]:
```
# reset the 'max_rows' option to its default
pd.reset_option('display.max_rows')
```

Documentation for **set_option** and **reset_option**

[292]:
```
# the 'max_columns' option is similar to 'max_rows'
pd.get_option('display.max_columns')
```

[292]:
```
20
```

[293]:
```
# read the training dataset from Kaggle's Titanic competition into a DataFrame
train = pd.read_csv('http://bit.ly/kaggletrain')
train.head()
```

[293]:
```
   PassengerId  Survived  Pclass  \
0            1         0       3
1            2         1       1
2            3         1       3
3            4         1       1
4            5         0       3

                                                Name     Sex   Age  SibSp  \
0                            Braund, Mr. Owen Harris    male  22.0      1
1  Cumings, Mrs. John Bradley (Florence Briggs Th…  female  38.0      1
2                             Heikkinen, Miss. Laina  female  26.0      0
3       Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0      1
4                           Allen, Mr. William Henry    male  35.0      0
```

```
   Parch            Ticket     Fare Cabin Embarked
0      0         A/5 21171   7.2500   NaN        S
1      0          PC 17599  71.2833   C85        C
2      0  STON/O2. 3101282   7.9250   NaN        S
3      0            113803  53.1000  C123        S
4      0            373450   8.0500   NaN        S
```

[294]:
```
# an ellipsis is displayed in the 'Name' cell of row 1 because of the␣
↪'max_colwidth' option
pd.get_option('display.max_colwidth')
```

[294]:
```
50
```

[295]:
```
# overwrite the current setting so that more characters will be displayed
pd.set_option('display.max_colwidth', 1000)
train.head()
```

[295]:
```
   PassengerId  Survived  Pclass  \
0            1         0       3
1            2         1       1
2            3         1       3
3            4         1       1
4            5         0       3

                                                 Name     Sex   Age  SibSp  \
0                             Braund, Mr. Owen Harris    male  22.0      1
1  Cumings, Mrs. John Bradley (Florence Briggs Thayer)  female  38.0      1
2                              Heikkinen, Miss. Laina  female  26.0      0
3        Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0      1
4                            Allen, Mr. William Henry    male  35.0      0

   Parch            Ticket     Fare Cabin Embarked
0      0         A/5 21171   7.2500   NaN        S
1      0          PC 17599  71.2833   C85        C
2      0  STON/O2. 3101282   7.9250   NaN        S
3      0            113803  53.1000  C123        S
4      0            373450   8.0500   NaN        S
```

[296]:
```
# notice that the 'Fare' column displays 4 digits after the decimal point
pd.get_option('display.precision')
```

[296]:
```
6
```

[297]:
```
# overwrite the 'precision' setting to display 2 digits after the decimal point
pd.set_option('display.precision', 2)
train.head()
```

[297]:
```
   PassengerId  Survived  Pclass  \
0            1         0       3
1            2         1       1
2            3         1       3
3            4         1       1
4            5         0       3

                                                Name     Sex   Age  SibSp  \
0                            Braund, Mr. Owen Harris    male  22.0      1
1  Cumings, Mrs. John Bradley (Florence Briggs Thayer)  female  38.0      1
2                             Heikkinen, Miss. Laina  female  26.0      0
3        Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0      1
4                           Allen, Mr. William Henry    male  35.0      0

   Parch            Ticket   Fare Cabin Embarked
0      0         A/5 21171   7.25   NaN        S
1      0          PC 17599  71.28   C85        C
2      0  STON/O2. 3101282   7.92   NaN        S
3      0            113803  53.10  C123        S
4      0            373450   8.05   NaN        S
```

[298]:
```
# add two meaningless columns to the drinks DataFrame
drinks['x'] = drinks.wine_servings * 1000
drinks['y'] = drinks.total_litres_of_pure_alcohol * 1000
drinks.head()
```

[298]:
```
       country  beer_servings  spirit_servings  wine_servings  \
0  Afghanistan              0                0              0
1      Albania             89              132             54
2      Algeria             25                0             14
3      Andorra            245              138            312
4       Angola            217               57             45

   total_litres_of_pure_alcohol continent       x        y
0                           0.0      Asia       0      0.0
1                           4.9    Europe   54000   4900.0
2                           0.7    Africa   14000    700.0
3                          12.4    Europe  312000  12400.0
4                           5.9    Africa   45000   5900.0
```

[299]:
```
# use a Python format string to specify a comma as the thousands separator
pd.set_option('display.float_format', '{:,}'.format)
drinks.head()
```

[299]:
```
       country  beer_servings  spirit_servings  wine_servings  \
0  Afghanistan              0                0              0
1      Albania             89              132             54
```

```
2      Algeria             25               0             14
3      Andorra            245             138            312
4       Angola            217              57             45

   total_litres_of_pure_alcohol continent       x        y
0                           0.0      Asia       0      0.0
1                           4.9    Europe   54000  4,900.0
2                           0.7    Africa   14000    700.0
3                          12.4    Europe  312000 12,400.0
4                           5.9    Africa   45000  5,900.0
```

[300]:
```
# 'y' was affected (but not 'x') because the 'float_format' option only affects␣
↪floats (not ints)
drinks.dtypes
```

[300]:
```
country                          object
beer_servings                     int64
spirit_servings                   int64
wine_servings                     int64
total_litres_of_pure_alcohol    float64
continent                        object
x                                 int64
y                               float64
dtype: object
```

[301]:
```
# view the option descriptions (including the default and current values)
pd.describe_option()
```

```
compute.use_bottleneck : bool
    Use the bottleneck library to accelerate if it is installed,
    the default is True
    Valid values: False,True
    [default: True] [currently: True]
compute.use_numba : bool
    Use the numba engine option for select operations if it is installed,
    the default is False
    Valid values: False,True
    [default: False] [currently: False]
compute.use_numexpr : bool
    Use the numexpr library to accelerate computation if it is installed,
    the default is True
    Valid values: False,True
    [default: True] [currently: True]
display.chop_threshold : float or None
    if set to a float value, all float values smaller than the given threshold
    will be displayed as exactly 0 by repr and friends.
    [default: None] [currently: None]
display.colheader_justify : 'left'/'right'
```

```
    Controls the justification of column headers. used by DataFrameFormatter.
    [default: right] [currently: right]
display.date_dayfirst : boolean
    When True, prints and parses dates with the day first, eg 20/01/2005
    [default: False] [currently: False]
display.date_yearfirst : boolean
    When True, prints and parses dates with the year first, eg 2005/01/20
    [default: False] [currently: False]
display.encoding : str/unicode
    Defaults to the detected encoding of the console.
    Specifies the encoding to be used for strings returned by to_string,
    these are generally strings meant to be displayed on the console.
    [default: UTF-8] [currently: UTF-8]
display.expand_frame_repr : boolean
    Whether to print out the full DataFrame repr for wide DataFrames across
    multiple lines, `max_columns` is still respected, but the output will
    wrap-around across multiple "pages" if its width exceeds `display.width`.
    [default: True] [currently: True]
display.float_format : callable
    The callable should accept a floating point number and return
    a string with the desired format of the number. This is used
    in some places like SeriesFormatter.
    See formats.format.EngFormatter for an example.
    [default: None] [currently: <built-in method format of str object at
0x1333fc0f0>]
display.html.border : int
    A ``border=value`` attribute is inserted in the ``<table>`` tag
    for the DataFrame HTML repr.
    [default: 1] [currently: 1]
display.html.table_schema : boolean
    Whether to publish a Table Schema representation for frontends
    that support it.
    (default: False)
    [default: False] [currently: False]
display.html.use_mathjax : boolean
    When True, Jupyter notebook will process table contents using MathJax,
    rendering mathematical expressions enclosed by the dollar symbol.
    (default: True)
    [default: True] [currently: True]
display.large_repr : 'truncate'/'info'
    For DataFrames exceeding max_rows/max_cols, the repr (and HTML repr) can
    show a truncated table, or switch to the view from
    df.info() (the behaviour in earlier versions of pandas).
    [default: truncate] [currently: truncate]
display.max_categories : int
    This sets the maximum number of categories pandas should output when
    printing out a `Categorical` or a Series of dtype "category".
    [default: 8] [currently: 8]
```

```
display.max_columns : int
    If max_cols is exceeded, switch to truncate view. Depending on
    `large_repr`, objects are either centrally truncated or printed as
    a summary view. 'None' value means unlimited.

    In case python/IPython is running in a terminal and `large_repr`
    equals 'truncate' this can be set to 0 or None and pandas will auto-detect
    the width of the terminal and print a truncated object which fits
    the screen width. The IPython notebook, IPython qtconsole, or IDLE
    do not run in a terminal and hence it is not possible to do
    correct auto-detection and defaults to 20.
    [default: 20] [currently: 20]
display.max_colwidth : int or None
    The maximum width in characters of a column in the repr of
    a pandas data structure. When the column overflows, a "…"
    placeholder is embedded in the output. A 'None' value means unlimited.
    [default: 50] [currently: 1000]
display.max_dir_items : int
    The number of items that will be added to `dir(…)`. 'None' value means
    unlimited. Because dir is cached, changing this option will not immediately
    affect already existing dataframes until a column is deleted or added.

    This is for instance used to suggest columns from a dataframe to tab
    completion.
    [default: 100] [currently: 100]
display.max_info_columns : int
    max_info_columns is used in DataFrame.info method to decide if
    per column information will be printed.
    [default: 100] [currently: 100]
display.max_info_rows : int or None
    df.info() will usually show null-counts for each column.
    For large frames this can be quite slow. max_info_rows and max_info_cols
    limit this null check only to frames with smaller dimensions than
    specified.
    [default: 1690785] [currently: 1690785]
display.max_rows : int
    If max_rows is exceeded, switch to truncate view. Depending on
    `large_repr`, objects are either centrally truncated or printed as
    a summary view. 'None' value means unlimited.

    In case python/IPython is running in a terminal and `large_repr`
    equals 'truncate' this can be set to 0 and pandas will auto-detect
    the height of the terminal and print a truncated object which fits
    the screen height. The IPython notebook, IPython qtconsole, or
    IDLE do not run in a terminal and hence it is not possible to do
    correct auto-detection.
    [default: 60] [currently: 60]
display.max_seq_items : int or None
```

```
    When pretty-printing a long sequence, no more then `max_seq_items`
    will be printed. If items are omitted, they will be denoted by the
    addition of "…" to the resulting string.

    If set to None, the number of items to be printed is unlimited.
    [default: 100] [currently: 100]
display.memory_usage : bool, string or None
    This specifies if the memory usage of a DataFrame should be displayed when
    df.info() is called. Valid values True,False,'deep'
    [default: True] [currently: True]
display.min_rows : int
    The numbers of rows to show in a truncated view (when `max_rows` is
    exceeded). Ignored when `max_rows` is set to None or 0. When set to
    None, follows the value of `max_rows`.
    [default: 10] [currently: 10]
display.multi_sparse : boolean
    "sparsify" MultiIndex display (don't display repeated
    elements in outer levels within groups)
    [default: True] [currently: True]
display.notebook_repr_html : boolean
    When True, IPython notebook will use html representation for
    pandas objects (if it is available).
    [default: True] [currently: True]
display.pprint_nest_depth : int
    Controls the number of nested levels to process when pretty-printing
    [default: 3] [currently: 3]
display.precision : int
    Floating point output precision in terms of number of places after the
    decimal, for regular formatting as well as scientific notation. Similar
    to ``precision`` in :meth:`numpy.set_printoptions`.
    [default: 6] [currently: 2]
display.show_dimensions : boolean or 'truncate'
    Whether to print out dimensions at the end of DataFrame repr.
    If 'truncate' is specified, only print out the dimensions if the
    frame is truncated (e.g. not display all rows and/or columns)
    [default: truncate] [currently: truncate]
display.unicode.ambiguous_as_wide : boolean
    Whether to use the Unicode East Asian Width to calculate the display text
    width.
    Enabling this may affect to the performance (default: False)
    [default: False] [currently: False]
display.unicode.east_asian_width : boolean
    Whether to use the Unicode East Asian Width to calculate the display text
    width.
    Enabling this may affect to the performance (default: False)
    [default: False] [currently: False]
display.width : int
    Width of the display in characters. In case python/IPython is running in
```

```
    a terminal this can be set to None and pandas will correctly auto-detect
    the width.
    Note that the IPython notebook, IPython qtconsole, or IDLE do not run in a
    terminal and hence it is not possible to correctly detect the width.
    [default: 80] [currently: 80]
future.infer_string Whether to infer sequence of str objects as pyarrow string
dtype, which will be the default in pandas 3.0 (at which point this option will
be deprecated).
    [default: False] [currently: False]
io.excel.ods.reader : string
    The default Excel reader engine for 'ods' files. Available options:
    auto, odf.
    [default: auto] [currently: auto]
io.excel.ods.writer : string
    The default Excel writer engine for 'ods' files. Available options:
    auto, odf.
    [default: auto] [currently: auto]
io.excel.xls.reader : string
    The default Excel reader engine for 'xls' files. Available options:
    auto, xlrd.
    [default: auto] [currently: auto]
io.excel.xlsb.reader : string
    The default Excel reader engine for 'xlsb' files. Available options:
    auto, pyxlsb.
    [default: auto] [currently: auto]
io.excel.xlsm.reader : string
    The default Excel reader engine for 'xlsm' files. Available options:
    auto, xlrd, openpyxl.
    [default: auto] [currently: auto]
io.excel.xlsm.writer : string
    The default Excel writer engine for 'xlsm' files. Available options:
    auto, openpyxl.
    [default: auto] [currently: auto]
io.excel.xlsx.reader : string
    The default Excel reader engine for 'xlsx' files. Available options:
    auto, xlrd, openpyxl.
    [default: auto] [currently: auto]
io.excel.xlsx.writer : string
    The default Excel writer engine for 'xlsx' files. Available options:
    auto, openpyxl, xlsxwriter.
    [default: auto] [currently: auto]
io.hdf.default_format : format
    default format writing format, if None, then
    put will default to 'fixed' and append will default to 'table'
    [default: None] [currently: None]
io.hdf.dropna_table : boolean
    drop ALL nan rows when appending to a table
    [default: False] [currently: False]
```

```
io.parquet.engine : string
    The default parquet reader/writer engine. Available options:
    'auto', 'pyarrow', 'fastparquet', the default is 'auto'
    [default: auto] [currently: auto]
io.sql.engine : string
    The default sql reader/writer engine. Available options:
    'auto', 'sqlalchemy', the default is 'auto'
    [default: auto] [currently: auto]
mode.chained_assignment : string
    Raise an exception, warn, or no action if trying to use chained assignment,
    The default is warn
    [default: warn] [currently: warn]
mode.copy_on_write : bool
    Use new copy-view behaviour using Copy-on-Write. Defaults to False,
    unless overridden by the 'PANDAS_COPY_ON_WRITE' environment variable
    (if set to "1" for True, needs to be set before pandas is imported).
    [default: False] [currently: False]
mode.data_manager : string
    Internal data manager type; can be "block" or "array". Defaults to "block",
    unless overridden by the 'PANDAS_DATA_MANAGER' environment variable (needs
    to be set before pandas is imported).
    [default: block] [currently: block]
mode.sim_interactive : boolean
    Whether to simulate interactive mode for purposes of testing
    [default: False] [currently: False]
mode.string_storage : string
    The default storage for StringDtype. This option is ignored if
    ``future.infer_string`` is set to True.
    [default: python] [currently: python]
mode.use_inf_as_na : boolean
    True means treat None, NaN, INF, -INF as NA (old way),
    False means None and NaN are null, but INF, -INF are not NA
    (new way).

    This option is deprecated in pandas 2.1.0 and will be removed in 3.0.
    [default: False] [currently: False]
    (Deprecated, use `` instead.)
plotting.backend : str
    The plotting backend to use. The default value is "matplotlib", the
    backend provided with pandas. Other backends can be specified by
    providing the name of the module that implements the backend.
    [default: matplotlib] [currently: matplotlib]
plotting.matplotlib.register_converters : bool or 'auto'.
    Whether to register converters with matplotlib's units registry for
    dates, times, datetimes, and Periods. Toggling to False will remove
    the converters, restoring any converters that pandas overwrote.
    [default: auto] [currently: auto]
styler.format.decimal : str
```

```
    The character representation for the decimal separator for floats and
complex.
    [default: .] [currently: .]
styler.format.escape : str, optional
    Whether to escape certain characters according to the given context; html or
latex.
    [default: None] [currently: None]
styler.format.formatter : str, callable, dict, optional
    A formatter object to be used as default within ``Styler.format``.
    [default: None] [currently: None]
styler.format.na_rep : str, optional
    The string representation for values identified as missing.
    [default: None] [currently: None]
styler.format.precision : int
    The precision for floats and complex numbers.
    [default: 6] [currently: 6]
styler.format.thousands : str, optional
    The character representation for thousands separator for floats, int and
complex.
    [default: None] [currently: None]
styler.html.mathjax : bool
    If False will render special CSS classes to table attributes that indicate
Mathjax
    will not be used in Jupyter Notebook.
    [default: True] [currently: True]
styler.latex.environment : str
    The environment to replace ``\begin{table}``. If "longtable" is used results
    in a specific longtable environment format.
    [default: None] [currently: None]
styler.latex.hrules : bool
    Whether to add horizontal rules on top and bottom and below the headers.
    [default: False] [currently: False]
styler.latex.multicol_align : {"r", "c", "l", "naive-l", "naive-r"}
    The specifier for horizontal alignment of sparsified LaTeX multicolumns.
Pipe
    decorators can also be added to non-naive values to draw vertical
    rules, e.g. "\|r" will draw a rule on the left side of right aligned merged
cells.
    [default: r] [currently: r]
styler.latex.multirow_align : {"c", "t", "b"}
    The specifier for vertical alignment of sparsified LaTeX multirows.
    [default: c] [currently: c]
styler.render.encoding : str
    The encoding used for output HTML and LaTeX files.
    [default: utf-8] [currently: utf-8]
styler.render.max_columns : int, optional
    The maximum number of columns that will be rendered. May still be reduced to
    satisfy ``max_elements``, which takes precedence.
```

```
    [default: None] [currently: None]
styler.render.max_elements : int
    The maximum number of data-cell (<td>) elements that will be rendered before
    trimming will occur over columns, rows or both if needed.
    [default: 262144] [currently: 262144]
styler.render.max_rows : int, optional
    The maximum number of rows that will be rendered. May still be reduced to
    satisfy ``max_elements``, which takes precedence.
    [default: None] [currently: None]
styler.render.repr : str
    Determine which output to use in Jupyter Notebook in {"html", "latex"}.
    [default: html] [currently: html]
styler.sparse.columns : bool
    Whether to sparsify the display of hierarchical columns. Setting to False
will
    display each explicit level element in a hierarchical key for each column.
    [default: True] [currently: True]
styler.sparse.index : bool
    Whether to sparsify the display of a hierarchical index. Setting to False
will
    display each explicit level element in a hierarchical key for each row.
    [default: True] [currently: True]
```

[302]:
```
# search for specific options by name
pd.describe_option('rows')
```

```
display.max_info_rows : int or None
    df.info() will usually show null-counts for each column.
    For large frames this can be quite slow. max_info_rows and max_info_cols
    limit this null check only to frames with smaller dimensions than
    specified.
    [default: 1690785] [currently: 1690785]
display.max_rows : int
    If max_rows is exceeded, switch to truncate view. Depending on
    `large_repr`, objects are either centrally truncated or printed as
    a summary view. 'None' value means unlimited.

    In case python/IPython is running in a terminal and `large_repr`
    equals 'truncate' this can be set to 0 and pandas will auto-detect
    the height of the terminal and print a truncated object which fits
    the screen height. The IPython notebook, IPython qtconsole, or
    IDLE do not run in a terminal and hence it is not possible to do
    correct auto-detection.
    [default: 60] [currently: 60]
display.min_rows : int
    The numbers of rows to show in a truncated view (when `max_rows` is
    exceeded). Ignored when `max_rows` is set to None or 0. When set to
    None, follows the value of `max_rows`.
```

```
    [default: 10] [currently: 10]
styler.render.max_rows : int, optional
    The maximum number of rows that will be rendered. May still be reduced to
    satisfy ``max_elements``, which takes precedence.
    [default: None] [currently: None]
```

Documentation for **describe_option**

[303]:
```
# reset all of the options to their default values
pd.reset_option('all')
```

```
/var/folders/zn/8p5wr_855bjbd9s6fvn6wd7w0000gn/T/ipykernel_13776/708164066.py:2:
FutureWarning: use_inf_as_na option is deprecated and will be removed in a
future version. Convert inf values to NaN before operating instead.
  pd.reset_option('all')
```

Options and settings from the pandas user guide

[Back to top]

1.29 Day 29: How do I create a pandas DataFrame from another object?

[304]:
```
# create a DataFrame from a dictionary (keys become column names, values become␣
↪data)
pd.DataFrame({'id':[100, 101, 102], 'color':['red', 'blue', 'red']})
```

[304]:

	id	color
0	100	red
1	101	blue
2	102	red

[305]:
```
# optionally define the index
df = pd.DataFrame({'id':[100, 101, 102], 'color':['red', 'blue', 'red']},␣
↪index=['a', 'b', 'c'])
df
```

[305]:

	id	color
a	100	red
b	101	blue
c	102	red

Documentation for **DataFrame**

[306]:
```
# create a DataFrame from a list of lists (each inner list becomes a row)
pd.DataFrame([[100, 'red'], [101, 'blue'], [102, 'red']], columns=['id',␣
↪'color'])
```

[306]:

	id	color
0	100	red
1	101	blue

```
2  102   red
```

[307]:
```
# create a NumPy array (with shape 4 by 2) and fill it with random numbers␣
↪between 0 and 1
import numpy as np
arr = np.random.rand(4, 2)
arr
```

[307]:
```
array([[0.39533253, 0.58414813],
       [0.1181487 , 0.36323294],
       [0.52904656, 0.20510143],
       [0.94205954, 0.73645436]])
```

Documentation for **np.random.rand**

[308]:
```
# create a DataFrame from the NumPy array
pd.DataFrame(arr, columns=['one', 'two'])
```

[308]:

	one	two
0	0.395333	0.584148
1	0.118149	0.363233
2	0.529047	0.205101
3	0.942060	0.736454

[309]:
```
# create a DataFrame of student IDs (100 through 109) and test scores (random␣
↪integers between 60 and 100)
pd.DataFrame({'student':np.arange(100, 110, 1), 'test':np.random.randint(60,␣
↪101, 10)})
```

[309]:

	student	test
0	100	75
1	101	67
2	102	63
3	103	74
4	104	80
5	105	93
6	106	97
7	107	96
8	108	78
9	109	92

Documentation for **np.arange** and **np.random.randint**

[310]:
```
# 'set_index' can be chained with the DataFrame constructor to select an index
pd.DataFrame({'student':np.arange(100, 110, 1), 'test':np.random.randint(60,␣
↪101, 10)}).set_index('student')
```

[310]:
```
         test
student
100        75
101        82
102        60
103        62
104        91
105        98
106        70
107        88
108        61
109        94
```

Documentation for **set_index**

[311]:
```
# create a new Series using the Series constructor
s = pd.Series(['round', 'square'], index=['c', 'b'], name='shape')
s
```

[311]:
```
c     round
b    square
Name: shape, dtype: object
```

Documentation for **Series**

[312]:
```
# review the existing DataFrame
df
```

[312]:
```
    id color
a  100   red
b  101  blue
c  102   red
```

[313]:
```
# concatenate the DataFrame and the Series (use axis=1 to concatenate columns)
pd.concat([df, s], axis=1)
```

[313]:
```
    id color   shape
a  100   red     NaN
b  101  blue  square
c  102   red   round
```

Notes:

- The Series name became the column name in the DataFrame.
- The Series data was aligned to the DataFrame by its index.
- The 'shape' for row 'a' was marked as a missing value (NaN) because that index was not present in the Series.

Documentation for **concat**

[Back to top]

1.30 Day 30: How do I apply a function to a pandas Series or DataFrame?

[314]:
```
# read the training dataset from Kaggle's Titanic competition into a DataFrame
train = pd.read_csv('http://bit.ly/kaggletrain')
train.head()
```

[314]:
```
   PassengerId  Survived  Pclass  \
0            1         0       3
1            2         1       1
2            3         1       3
3            4         1       1
4            5         0       3

                                                Name     Sex   Age  SibSp  \
0                            Braund, Mr. Owen Harris    male  22.0      1
1  Cumings, Mrs. John Bradley (Florence Briggs Th…  female  38.0      1
2                             Heikkinen, Miss. Laina  female  26.0      0
3       Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0      1
4                           Allen, Mr. William Henry    male  35.0      0

   Parch            Ticket     Fare Cabin Embarked
0      0         A/5 21171   7.2500   NaN        S
1      0          PC 17599  71.2833   C85        C
2      0  STON/O2. 3101282   7.9250   NaN        S
3      0            113803  53.1000  C123        S
4      0            373450   8.0500   NaN        S
```

Goal: Map the existing values of a Series to a different set of values

Method: `map` (Series method)

[315]:
```
# map 'female' to 0 and 'male' to 1
train['Sex_num'] = train.Sex.map({'female':0, 'male':1})
train.loc[0:4, ['Sex', 'Sex_num']]
```

[315]:
```
      Sex  Sex_num
0    male        1
1  female        0
2  female        0
3  female        0
4    male        1
```

Goal: Apply a function to each element in a Series

Method: `apply` (Series method)

Note: **`map`** can be substituted for **`apply`** in many cases, but **`apply`** is more flexible and thus is recommended

[316]:
```
# calculate the length of each string in the 'Name' Series
train['Name_length'] = train.Name.apply(len)
train.loc[0:4, ['Name', 'Name_length']]
```

[316]:
```
                                                Name  Name_length
0                            Braund, Mr. Owen Harris           23
1  Cumings, Mrs. John Bradley (Florence Briggs Th…           51
2                             Heikkinen, Miss. Laina           22
3       Futrelle, Mrs. Jacques Heath (Lily May Peel)           44
4                           Allen, Mr. William Henry           24
```

[317]:
```
# round up each element in the 'Fare' Series to the next integer
import numpy as np
train['Fare_ceil'] = train.Fare.apply(np.ceil)
train.loc[0:4, ['Fare', 'Fare_ceil']]
```

[317]:
```
      Fare  Fare_ceil
0   7.2500        8.0
1  71.2833       72.0
2   7.9250        8.0
3  53.1000       54.0
4   8.0500        9.0
```

[318]:
```
# we want to extract the last name of each person
train.Name.head()
```

[318]:
```
0                              Braund, Mr. Owen Harris
1    Cumings, Mrs. John Bradley (Florence Briggs Th…
2                               Heikkinen, Miss. Laina
3         Futrelle, Mrs. Jacques Heath (Lily May Peel)
4                             Allen, Mr. William Henry
Name: Name, dtype: object
```

[319]:
```
# use a string method to split the 'Name' Series at commas (returns a Series of␣
↪lists)
train.Name.str.split(',').head()
```

[319]:
```
0                            [Braund,  Mr. Owen Harris]
1    [Cumings,  Mrs. John Bradley (Florence Briggs …
2                            [Heikkinen,  Miss. Laina]
3      [Futrelle,  Mrs. Jacques Heath (Lily May Peel)]
4                          [Allen,  Mr. William Henry]
Name: Name, dtype: object
```

[320]:
```
# define a function that returns an element from a list based on position
def get_element(my_list, position):
    return my_list[position]
```

[321]:
```
# apply the 'get_element' function and pass 'position' as a keyword argument
train.Name.str.split(',').apply(get_element, position=0).head()
```

[321]:
```
0       Braund
1      Cumings
2    Heikkinen
3     Futrelle
4        Allen
Name: Name, dtype: object
```

[322]:
```
# alternatively, use a lambda function
train.Name.str.split(',').apply(lambda x: x[0]).head()
```

[322]:
```
0       Braund
1      Cumings
2    Heikkinen
3     Futrelle
4        Allen
Name: Name, dtype: object
```

Goal: Apply a function along either axis of a DataFrame

Method: `apply` (DataFrame method)

[323]:
```
# read a dataset of alcohol consumption into a DataFrame
drinks = pd.read_csv('http://bit.ly/drinksbycountry')
drinks.head()
```

[323]:

	country	beer_servings	spirit_servings	wine_servings	total_litres_of_pure_alcohol	continent
0	Afghanistan	0	0	0	0.0	Asia
1	Albania	89	132	54	4.9	Europe
2	Algeria	25	0	14	0.7	Africa
3	Andorra	245	138	312	12.4	Europe
4	Angola	217	57	45	5.9	Africa

[324]:
```
# select a subset of the DataFrame to work with
drinks.loc[:, 'beer_servings':'wine_servings'].head()
```

[324]:

	beer_servings	spirit_servings	wine_servings
0	0	0	0
1	89	132	54
2	25	0	14

```
3            245             138            312
4            217              57             45
```

[325]:
```
# apply the 'max' function along axis 0 to calculate the maximum value in each column
drinks.loc[:, 'beer_servings':'wine_servings'].apply(max, axis=0)
```

[325]:
```
beer_servings      376
spirit_servings    438
wine_servings      370
dtype: int64
```

[326]:
```
# apply the 'max' function along axis 1 to calculate the maximum value in each row
drinks.loc[:, 'beer_servings':'wine_servings'].apply(max, axis=1).head()
```

[326]:
```
0      0
1    132
2     25
3    312
4    217
dtype: int64
```

[327]:
```
# use 'idxmax' to calculate which column has the maximum value for each row
drinks.loc[:, 'beer_servings':'wine_servings'].idxmax(axis=1).head()
```

[327]:
```
0      beer_servings
1    spirit_servings
2      beer_servings
3      wine_servings
4      beer_servings
dtype: object
```

Goal: Apply a function to every element in a DataFrame

Method: `map` (DataFrame method)

[328]:
```
# convert every DataFrame element into a float
drinks.loc[:, 'beer_servings':'wine_servings'].map(float).head()
```

[328]:
```
   beer_servings  spirit_servings  wine_servings
0            0.0              0.0            0.0
1           89.0            132.0           54.0
2           25.0              0.0           14.0
3          245.0            138.0          312.0
4          217.0             57.0           45.0
```

[329]:
```
# overwrite the existing DataFrame columns (this no longer works in 2024)
drinks.loc[:, 'beer_servings':'wine_servings'] = drinks.loc[:, 'beer_servings':
 ↪'wine_servings'].map(float)
drinks.head()
```

[329]:

	country	beer_servings	spirit_servings	wine_servings	total_litres_of_pure_alcohol	continent
0	Afghanistan	0	0	0	0.0	Asia
1	Albania	89	132	54	4.9	Europe
2	Algeria	25	0	14	0.7	Africa
3	Andorra	245	138	312	12.4	Europe
4	Angola	217	57	45	5.9	Africa

Introduction to Machine Learning with scikit-learn (Data School course)

Tuesday Tips (Data School newsletter)

[Back to top]