

Master Data Analysis with Python Solutions

by
Ted Petrou

© 2019 Ted Petrou All Rights Reserved

Contents

I	Intro to pandas	1
1	Solutions	3
1.1	2. The DataFrame and Series	3
1.2	3. Data Types and Missing Values	4
1.3	4. Setting a meaningful index	6
II	Selecting Subsets of Data	9
2	Solutions	11
2.1	1. Selecting Subsets of Data from DataFrames with just the brackets	11
2.2	2. Selecting Subsets of Data from DataFrames with <code>loc</code>	12
2.3	3. Selecting Subsets of Data from DataFrames with <code>iloc</code>	16
2.4	4. Selecting Subsets of Data from a Series	18
2.5	5. Boolean Selection Single Conditions	21
2.6	6. Boolean Selection Multiple Conditions	25
2.7	7. Boolean Selection More	30
2.8	8. Filtering with the query Method	33
2.9	9. Miscellaneous Subset Selection	36
III	Essential Series Commands	37
3	Solutions	39
3.1	1. Series Attributes and Statistical Methods	39
3.2	2. Series Missing Value Methods	42
3.3	3. Series Sorting, Ranking, and Uniqueness	48
3.4	4. Series Methods More	51
3.5	5. String Series Methods	54
3.6	6. Datetime Series Methods	58
3.7	Project - Testing Normality of Stock Market Returns	60
IV	Essential DataFrame Commands	65
4	Solutions	67
4.1	1. DataFrame Attributes and Methods	67
4.2	2. DataFrame Statistical Methods	67
4.3	3. DataFrame Missing Value Methods	74
4.4	4. DataFrame Sorting, Ranking, and Uniqueness	76
4.5	5. DataFrame Structure Methods	81
4.6	6. DataFrame Methods More	85

4.7 7. Assigning Subsets of Data	87
V Data Types	89
5 Solutions	91
5.1 1. Changing Data Types	91
5.2 2. Categorical Data Type	94
5.3 3. Nullable Integer Data Type	96
VI Grouping Data	97
6 Solutions	99
6.1 1. Groupby Aggregation Basics	99
6.2 2. Grouping and Aggregating with Multiple Columns	101
6.3 3. Grouping with Pivot Tables	110
6.4 4. Counting with Crosstabs	115
6.5 5. Alternate Groupby Syntax	119
6.6 6. Custom Aggregation	119
6.7 9. Transform and Filter with Groupby	123
VII Time Series	129
7 Solutions	131
7.1 1. Datetime and Timedelta	131
7.2 2. Introduction to Time Series	134
7.3 3. Grouping by Time	137
7.4 4. Rolling Windows	140
7.5 5. Grouping by Time and another Column	142
VIII Regular Expressions	147
8 Solutions	149
8.1 1. Introduction to Regular Expressions	149
8.2 2. Quantifiers	151
8.3 3. Or Conditions	153
8.4 4. Character Sets and Grouping	155
8.5 Project - Explore Newsgroups with Regexes	159
8.6 Project - Feature Engineering on the Titanic	168
IX Tidy Data	179
9 Solutions	181
9.1 1. Tidy Data with <code>melt</code>	181
9.2 2. Reshaping by Pivoting	185
9.3 3. Common messy datasets	192

X Joining Data	199
10 Solutions	201
10.1 4. Data Normalization	201
XI Visualization	203
11 Solutions	205
11.1 4. Matplotlib Data Plotting	205
11.2 5. Plotting with Pandas	209
11.3 6. Seaborn	219

Part I

Intro to pandas

Chapter 1

Solutions

1.1 2. The DataFrame and Series

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

      pd.options.display.max_columns = 40
      bikes = pd.read_csv('../data/bikes.csv')
```

Exercise 1

Select the column `events`, the type of weather that was recorded, and assign it to a variable with the same name. Output the first 10 values of it.

```
[2]: events = bikes['events']
      events.head(10)
```

```
[2]: 0    mostlycloudy
      1    partlycloudy
      2    mostlycloudy
      3    mostlycloudy
      4    partlycloudy
      5    mostlycloudy
      6        cloudy
      7        cloudy
      8        cloudy
      9    mostlycloudy
      Name: events, dtype: object
```

Exercise 2

What type of object is `events`?

```
[3]: # it's a Series
      type(events)
```

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

Exercise 3

Select the last two rows of the `bikes` DataFrame and assign it to the variable `bikes_last_2`. What type of object is `bikes_last_2`?

```
[4]: # it's a DataFrame
bikes_last_2 = bikes.tail(2)
type(bikes_last_2)
```

[4]: `pandas.core.frame.DataFrame`

Exercise 4

Use `pd.reset_option('all')` to reset the options to their default values. Test that this worked.

```
[5]: pd.reset_option('all')
```

1.2 3. Data Types and Missing Values

Exercise 1

What type of object is returned from the `dtypes` attribute?

```
[6]: # a Series
type(bikes.dtypes)
```

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

Exercise 2

What type of object is returned from the `shape` attribute?

```
[7]: # a tuple of rows, columns
type(bikes.shape)
```

[7]: `tuple`

Exercise 3

What type of object is returned from the `info` method?

The object `None` is returned. What you see is just output printed to the screen.

```
[8]: info_return = bikes.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50089 entries, 0 to 50088
Data columns (total 19 columns):
trip_id          50089 non-null int64
usertype         50089 non-null object
gender           50089 non-null object
starttime        50089 non-null object
stoptime          50089 non-null object
tripduration     50089 non-null int64
```

```
from_station_name      50089 non-null object
latitude_start         50083 non-null float64
longitude_start        50083 non-null float64
dpcapacity_start       50083 non-null float64
to_station_name        50089 non-null object
latitude_end           50077 non-null float64
longitude_end          50077 non-null float64
dpcapacity_end         50077 non-null float64
temperature            50089 non-null float64
visibility             50089 non-null float64
wind_speed              50089 non-null float64
precipitation           50089 non-null float64
events                 50089 non-null object
dtypes: float64(10), int64(2), object(7)
memory usage: 7.3+ MB
```

```
[9]: type(info_return)
```

```
[9]: NoneType
```

Exercise 4

The memory usage from the `info` method isn't correct when you have objects in your DataFrame. Read the docstrings from it and get the true memory usage.

```
[10]: bikes.info(memory_usage='deep')
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50089 entries, 0 to 50088
Data columns (total 19 columns):
trip_id              50089 non-null int64
usertype             50089 non-null object
gender               50089 non-null object
starttime            50089 non-null object
stoptime              50089 non-null object
tripduration          50089 non-null int64
from_station_name    50089 non-null object
latitude_start        50083 non-null float64
longitude_start       50083 non-null float64
dpcapacity_start      50083 non-null float64
to_station_name       50089 non-null object
latitude_end          50077 non-null float64
longitude_end         50077 non-null float64
dpcapacity_end        50077 non-null float64
temperature           50089 non-null float64
visibility            50089 non-null float64
wind_speed             50089 non-null float64
precipitation          50089 non-null float64
events                50089 non-null object
dtypes: float64(10), int64(2), object(7)
memory usage: 28.9 MB
```

1.3 4. Setting a meaningful index

Exercise 1

Read in the movie dataset and set the index to be something other than movie title. Are there any other good columns to use as an index?

```
[11]: movies = pd.read_csv('../data/movie.csv', index_col='director_name')
movies.head(3)
```

	title	year	color	content_rating	duration	...	plot_keywords	language	country	budget	imdb_score
director_name											
James Cameron	Avatar	2009.0	Color	PG-13	178.0	...	avatar future marine native paraplegic	English	USA	237000000.0	7.9
Gore Verbinski	Pirates of the Caribbean: At World's End	2007.0	Color	PG-13	169.0	...	goddess marriage ceremony marriage proposal pi...	English	USA	300000000.0	7.1
Sam Mendes	Spectre	2015.0	Color	PG-13	148.0	...	bomb espionage sequel spy terrorist	English	UK	245000000.0	6.8

Director name isn't unique. There aren't any other good column names to use as an index.

Exercise 2

Use `set_index` to set the index and keep the column as part of the data. Read the docstrings to find the parameter that controls this functionality.

```
[12]: movies = pd.read_csv('../data/movie.csv')
movies = movies.set_index('title', drop=False)
movies.head(3)
```

	title	year	color	content_rating	duration	...	plot_keywords	language	country	budget	imdb_score
title											
Avatar	Avatar	2009.0	Color	PG-13	178.0	...	avatar future marine native paraplegic	English	USA	237000000.0	7.9
Pirates of the Caribbean: At World's End	Pirates of the Caribbean: At World's End	2007.0	Color	PG-13	169.0	...	goddess marriage ceremony marriage proposal pi...	English	USA	300000000.0	7.1
Spectre	Spectre	2015.0	Color	PG-13	148.0	...	bomb espionage sequel spy terrorist	English	UK	245000000.0	6.8

Exercise 3

Read in the movie DataFrame and set the index as the title column. Assign the index to its own variable and output the last 10 movies titles.

```
[13]: movies = pd.read_csv('../data/movie.csv', index_col='title')
index = movies.index
index[-10:]
```

```
[13]: Index(['Primer', 'Cavite', 'El Mariachi', 'The Mongol King', 'Newlyweds',
       'Signed Sealed Delivered', 'The Following', 'A Plague So Pleasant',
       'Shanghai Calling', 'My Date with Drew'],
       dtype='object', name='title')
```

Exercise 4

Use an integer instead of the column name for `index_col` when reading in the data using `read_csv`. What does it do?

```
[14]: movies = pd.read_csv('../data/movie.csv', index_col=-5)
movies.head(3)
```

		title	year	color	content_rating	duration	...	num_voted_users	language	country	budget	imdb_score
plot_keywords												
avatar future marine native paraplegic	Avatar	2009.0	Color		PG-13	178.0	...	886204	English	USA	237000000.0	7.9
goddess marriage ceremony marriage proposal pirate singapore	Pirates of the Caribbean: At World's End	2007.0	Color		PG-13	169.0	...	471220	English	USA	300000000.0	7.1
bomb espionage sequel spy terrorist	Spectre	2015.0	Color		PG-13	148.0	...	275868	English	UK	245000000.0	6.8

It chooses the column name with that integer location.

Part II

Selecting Subsets of Data

Chapter 2

Solutions

2.1 1. Selecting Subsets of Data from DataFrames with just the brackets

```
[1]: import pandas as pd  
movie = pd.read_csv('../data/movie.csv', index_col='title')
```

Exercise 1

Select the column with the director's name as a Series

```
[2]: movie['director_name'].head(3)
```

```
[2]: title  
Avatar           James Cameron  
Pirates of the Caribbean: At World's End Gore Verbinski  
Spectre          Sam Mendes  
Name: director_name, dtype: object
```

Exercise 2

Select the column with the director's name and number of Facebook likes.

```
[3]: movie[['director_name', 'director_fb']].head(3)
```

	director_name	director_fb
title		
Avatar	James Cameron	0.0
Pirates of the Caribbean: At World's End	Gore Verbinski	563.0
Spectre	Sam Mendes	0.0

Exercise 3

Select a single column as a DataFrame and not a Series

```
[4]: # make a one item list
col = ['director_name']
movie[col].head(3)
```

director_name	
title	
Avatar	James Cameron
Pirates of the Caribbean: At World's End	Gore Verbinski
Spectre	Sam Mendes

2.2 2. Selecting Subsets of Data from DataFrames with loc

```
[5]: movie = pd.read_csv('../data/movie.csv', index_col='title')
movie.head(3)
```

	year	color	content_rating	duration	director_name	...	plot_keywords	language	country	budget	imdb_score
title											
Avatar	2009.0	Color	PG-13	178.0	James Cameron	...	avatar future marine native paraplegic	English	USA	237000000.0	7.9
Pirates of the Caribbean: At World's End	2007.0	Color	PG-13	169.0	Gore Verbinski	...	goddess marriage ceremony marriage proposal pi...	English	USA	300000000.0	7.1
Spectre	2015.0	Color	PG-13	148.0	Sam Mendes	...	bomb espionage sequel spy terrorist	English	UK	245000000.0	6.8

Exercise 1

Select columns `actor1`, `actor2`, and `actor3` for the movies Home Alone and Top Gun

```
[6]: rows = ['Home Alone', 'Top Gun']
cols = ['actor1', 'actor2', 'actor3']
movie.loc[rows, cols]
```

	actor1	actor2	actor3
title			
Home Alone	Macaulay Culkin	Devin Ratray	Catherine O'Hara
Top Gun	Tom Cruise	Tom Skerritt	Adrian Pasdar

Exercise 2

Select columns `actor1`, `actor2`, and `actor3` for the all the movies beginning at Home Alone and ending at Top Gun

```
[7]: cols = ['actor1', 'actor2', 'actor3']
movie.loc['Home Alone':'Top Gun', cols]
```

	actor1	actor2	actor3
title			
Home Alone	Macaulay Culkin	Devin Ratray	Catherine O'Hara
3 Men and a Baby	Tom Selleck	Ted Danson	Steve Guttenberg
Tootsie	Bill Murray	Sydney Pollack	Teri Garr
Top Gun	Tom Cruise	Tom Skerritt	Adrian Pasdar

Exercise 3

Select just the `director_name` column for the movies Home Alone and Top Gun.

```
[8]: rows = ['Home Alone', 'Top Gun']
movie.loc[rows, 'director_name']
```

```
[8]: title
Home Alone      Chris Columbus
Top Gun          Tony Scott
Name: director_name, dtype: object
```

Exercise 4

Repeat exercise 3, but return a DataFrame instead.

```
[9]: rows = ['Home Alone', 'Top Gun']
cols = ['director_name']
movie.loc[rows, cols]
```

director_name	
title	
Home Alone	Chris Columbus
Top Gun	Tony Scott

Exercise 5

Select all columns for the movie ‘The Dark Knight Rises’.

```
[10]: movie.loc['The Dark Knight Rises', :]
```

year	2012
color	Color
content_rating	PG-13
duration	164
director_name	Christopher Nolan
director_fb	22000
actor1	Tom Hardy
actor1_fb	27000
actor2	Christian Bale

```

actor2_fb           23000
actor3                Joseph Gordon-Levitt
actor3_fb            23000
gross              4.48131e+08
genres             Action|Thriller
num_reviews          813
num_voted_users      1144337
plot_keywords    deception|imprisonment|lawlessness|police offi...
language            English
country             USA
budget              2.5e+08
imdb_score            8.5
Name: The Dark Knight Rises, dtype: object

```

Alternatively, don't include the empty slice.

```
[11]: # movie.loc['The Dark Knight Rises']
```

Exercise 6

Repeat exercise 5 but return a DataFrame instead.

```
[12]: movie.loc[['The Dark Knight Rises'], :]
```

	year	color	content_rating	duration	director_name	...	plot_keywords	language	country	budget	imdb_score
title											
The Dark Knight Rises	2012.0	Color	PG-13	164.0	Christopher Nolan	...	deception imprisonment lawlessness police offi...	English	USA	250000000.0	8.5

Exercise 7

Select all columns for the movies ‘Tangled’ and ‘Avatar’.

```
[13]: rows = ['Tangled', 'Avatar']
movie.loc[rows, :]
```

	year	color	content_rating	duration	director_name	...	plot_keywords	language	country	budget	imdb_score
title											
Tangled	2010.0	Color	PG	100.0	Nathan Greno	...	17th century based on fairy tale disney flower...	English	USA	260000000.0	7.8
Avatar	2009.0	Color	PG-13	178.0	James Cameron	...	avatar future marine native paraplegic	English	USA	237000000.0	7.9

Alternatively, omit the empty slice.

```
[14]: # movie.loc[rows]
```

Exercise 7

What year was ‘Tangled’ and ‘Avatar’ made and what was their IMBD scores?

```
[15]: movie.loc[['Tangled', 'Avatar'], ['year', 'imdb_score']]
```

	year	imdb_score
title		
Tangled	2010.0	7.8
Avatar	2009.0	7.9

Exercise 8

Can you tell what the data type of the `year` column is by just looking at its values?

Yes, because it has a decimal value it must be a float. Integers do not have decimals

Exercise 9

Use a single method to output the data type and number of non-missing values of `year`. Is it missing any?

```
[16]: # yes, it's missing many values. 4810 non-missing vs 4916 total
       movie.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 4916 entries, Avatar to My Date with Drew
Data columns (total 21 columns):
year           4810 non-null float64
color          4897 non-null object
content_rating 4616 non-null object
duration       4901 non-null float64
director_name  4814 non-null object
director_fb    4814 non-null float64
actor1         4909 non-null object
actor1_fb      4909 non-null float64
actor2         4903 non-null object
actor2_fb      4903 non-null float64
actor3         4893 non-null object
actor3_fb      4893 non-null float64
gross          4054 non-null float64
genres         4916 non-null object
num_reviews    4867 non-null float64
num_voted_users 4916 non-null int64
plot_keywords   4764 non-null object
language        4904 non-null object
country         4911 non-null object
budget          4432 non-null float64
imdb_score     4916 non-null float64
dtypes: float64(10), int64(1), object(10)
memory usage: 1004.9+ KB
```

Exercise 10

Select every 300th movie between ‘Tangled’ and ‘Forrest Gump’. Why doesn’t ‘Forrest Gump’ appear in the results?

[17]: # Forrest Gump is not a multiple of 300 away from Tangled
`movie.loc['Tangled':'Forrest Gump':300]`

	year	color	content_rating	duration	director_name	...	plot_keywords	language	country	budget	imdb_score
title											
Tangled	2010.0	Color	PG	100.0	Nathan Greno	...	17th century based on fairy tale disney flower...	English	USA	260000000.0	7.8
Cloud Atlas	2012.0	Color	R	172.0	Tom Tykwer	...	composer future letter nonlinear timeline nurs...	English	Germany	102000000.0	7.5
Doom	2005.0	Color	R	113.0	Andrzej Bartkowiak	...	commando unit extra chromosome first person sh...	English	UK	60000000.0	5.2

2.3 3. Selecting Subsets of Data from DataFrames with iloc

Exercise 1

Select the columns with integer location 10, 5, and 1.

[18]: `movie.iloc[:, [10, 5, 1]].head(3)`

	actor3	director_fb	color
title			
	Avatar	Wes Studi	0.0 Color
Pirates of the Caribbean: At World's End	Jack Davenport	563.0	Color
Spectre	Stephanie Sigman	0.0	Color

Exercise 2

Select the rows with integer location 10, 5, and 1.

[19]: `movie.iloc[[10, 5, 1], :]`

	year	color	content_rating	duration	director_name	...	plot_keywords	language	country	budget	imdb_score
title											
Batman v Superman: Dawn of Justice	2016.0	Color	PG-13	183.0	Zack Snyder	...	based on comic book batman sequel to a reboot ...	English	USA	250000000.0	6.9
John Carter	2012.0	Color	PG-13	132.0	Andrew Stanton	...	alien american civil war male nipple mars prin...	English	USA	263700000.0	6.6
Pirates of the Caribbean: At World's End	2007.0	Color	PG-13	169.0	Gore Verbinski	...	goddess marriage ceremony marriage proposal pi...	English	USA	300000000.0	7.1

Exercise 3

Select rows with integer location 100 to 104 along with the column integer location 5.

[20]: `movie.iloc[100:105, 5]`

[20]: title

```
The Fast and the Furious          357.0
The Curious Case of Benjamin Button    21000.0
X-Men: First Class                  905.0
The Hunger Games: Mockingjay - Part 2   508.0
The Sorcerer's Apprentice            226.0
Name: director_fb, dtype: float64
```

Exercise 4

Select the value at row integer location 100 and column integer location 4.

[21]: movie.iloc[100, 4]

[21]: 'Rob Cohen'

Exercise 5

Return the result of exercise 4 as a DataFrame.

[22]: movie.iloc[[100], [4]]

director_name	
title	
The Fast and the Furious	Rob Cohen

Exercise 6

Select the last 5 rows of the last 5 columns.

[23]: movie.iloc[-5:, -5:]

title	plot_keywords	language	country	budget	imdb_score
Signed Sealed Delivered	fraud postal worker prison theft trial	English	Canada	NaN	7.7
The Following	cult fbi hideout prison escape serial killer	English	USA	NaN	7.5
A Plague So Pleasant		NaN	English	USA	1400.0
Shanghai Calling		NaN	English	USA	NaN
My Date with Drew	actress name in title crush date four word tit...	English	USA	1100.0	6.6

Exercise 7

Select every 25th row between rows with integer location 100 and 20 and every fifth column.

[24]: movie.iloc[100:200:25, ::5]

	year	director_fb	actor3	num_voted_users	imdb_score
title					
The Fast and the Furious	2001.0	357.0	Jordana Brewster	272223	6.7
Frozen	2013.0	69.0	Livvy Stubenrauch	421658	7.6
Armageddon	1998.0	0.0	Will Patton	322395	6.6
The Incredible Hulk	2008.0	255.0	William Hurt	326286	6.8

Exercise 8

Select the column with integer location 7 as a Series.

```
[25]: movie.iloc[:, 7].head(3)
```

```
[25]: title
      Avatar                         1000.0
      Pirates of the Caribbean: At World's End 40000.0
      Spectre                         11000.0
      Name: actor1_fb, dtype: float64
```

Exercise 9

Select the rows with integer location 999, 99, and 9 and the columns with integer location 9 and 19.

```
[26]: rows = [999, 99, 9]
       cols = [9, 19]
       movie.iloc[rows, cols]
```

	actor2_fb	budget
	title	
The Iron Giant	631.0	70000000.0
The Hobbit: An Unexpected Journey	972.0	180000000.0
Harry Potter and the Half-Blood Prince	11000.0	250000000.0

2.4 4. Selecting Subsets of Data from a Series

```
[27]: import pandas as pd  
movie = pd.read_csv('../data/movie.csv', index_col='title')  
duration = movie['duration']  
duration.head()
```

```
[27]: title  
      Avatar                      178.0  
      Pirates of the Caribbean: At World's End 169.0  
      Spectre                      148.0  
      The Dark Knight Rises       164.0
```

```
Star Wars: Episode VII - The Force Awakens      NaN
Name: duration, dtype: float64
```

Exercise 1

How long was the movie Titanic?

```
[28]: duration.loc['Titanic']
```

```
[28]: 194.0
```

Exercise 2

How long was the movie at the 999th integer location?

```
[29]: duration.iloc[999]
```

```
[29]: 90.0
```

Exercise 3

Select the duration for the movies Hulk, Toy Story, and Cars.

```
[30]: names = ['Hulk', 'Toy Story', 'Cars']
duration.loc[names]
```

```
[30]: title
Hulk          138.0
Toy Story    74.0
Cars          117.0
Name: duration, dtype: float64
```

Exercise 4

Select the duration for every 100th movies from Hulk to Cars

```
[31]: duration.loc['Hulk':'Cars':100]
```

```
[31]: title
Hulk          138.0
Live Free or Die Hard 129.0
Valkyrie     121.0
Yogi Bear    80.0
Dr. Dolittle 2   87.0
Name: duration, dtype: float64
```

Exercise 5

Select the duration for every 10th movie beginning from the 100th from the end.

```
[32]: duration.iloc[-100::10]
```

[32]: title

Antarctic Edge: 70° South	72.0
A Dog's Breakfast	88.0
Penitentiary	99.0
Peace, Propaganda & the Promised Land	80.0
Supporting Characters	87.0
Bending Steel	92.0
Run, Hide, Die	75.0
Manito	79.0
Breaking Upwards	88.0
Primer	77.0

Name: duration, dtype: float64

Read in the bikes dataset and select the `wind_speed` column by executing the cell below and use it for the rest of exercises. Notice that the index labels are integers, meaning that when you use `loc` you will be using integers.

[33]:

```
bikes = pd.read_csv('../data/bikes.csv')
wind = bikes['wind_speed']
wind.head()
```

[33]:

0	12.7
1	6.9
2	16.1
3	16.1
4	17.3

Name: wind_speed, dtype: float64

Exercise 6

What kind of index does the `wind` Series have?

It has a `RangeIndex`

[34]:

```
wind.index
```

[34]:

```
RangeIndex(start=0, stop=50089, step=1)
```

Exercise 7

From the `wind` Series, select the integer locations 4 through, but not including 10

[35]:

```
wind.iloc[4:10]
```

[35]:

4	17.3
5	17.3
6	15.0
7	5.8
8	0.0
9	12.7

Name: wind_speed, dtype: float64

Exercise 8

Copy and paste your answer to Exercise 7 below but use `loc` instead. Do you get the same result? Why not?

```
[36]: wind.loc[4:10]
```

```
[36]: 4      17.3
5      17.3
6     15.0
7      5.8
8      0.0
9     12.7
10     9.2
Name: wind_speed, dtype: float64
```

This is tricky - the index in this case contains integers and not strings. So the labels themselves are also integers and happen to be the same integers corresponding to integer location. The reason `.iloc` and `.loc` produce different results is that `.loc` always includes the last value when slicing.

2.5 5. Boolean Selection Single Conditions

```
[37]: bikes = pd.read_csv('../data/bikes.csv')
bikes.head(3)
```

	trip_id	usertype	gender	starttime	stoptime	...	temperature	visibility	wind_speed	precipitation	events
0	7147	Subscriber	Male	2013-06-28 19:01:00	2013-06-28 19:17:00	...	73.9	10.0	12.7	-9999.0	mostlycloudy
1	7524	Subscriber	Male	2013-06-28 22:53:00	2013-06-28 23:03:00	...	69.1	10.0	6.9	-9999.0	partlycloudy
2	10927	Subscriber	Male	2013-06-30 14:43:00	2013-06-30 15:01:00	...	73.0	10.0	16.1	-9999.0	mostlycloudy

Exercise 1

Find all the rides with temperature below 0.

```
[38]: filt = bikes['temperature'] < 0
bikes[filt].head(3)
```

	trip_id	usertype	gender	starttime	stoptime	...	temperature	visibility	wind_speed	precipitation	events
1871	1071512	Subscriber	Male	2013-12-05:13:00	2013-12-12 05:27:00	...	-2.0	10.0	6.9	-9999.0	mostlycloudy
2049	1142687	Subscriber	Female	2014-01-06:15:00	2014-01-23 06:29:00	...	-2.0	10.0	16.1	-9999.0	partlycloudy
2054	1144341	Subscriber	Male	2014-01-21:15:00	2014-01-23 21:21:00	...	-0.9	10.0	12.7	-9999.0	clear

Exercise 2

Find all the rides with wind speed greater than 30.

```
[39]: filt = bikes['wind_speed'] > 30
bikes[filt].head(3)
```

	trip_id	usertype	gender	starttime	stoptime	...	temperature	visibility	wind_speed	precipitation	events
2164	1193566	Subscriber	Male	2014-02-20 19:06:00	2014-02-20 19:07:00	...	46.9	10.0	31.1	-9999.0	mostlycloudy
2165	1193760	Subscriber	Male	2014-02-20 20:47:00	2014-02-20 21:14:00	...	39.0	10.0	35.7	-9999.0	cloudy
2479	1320861	Subscriber	Male	2014-04-01 08:28:00	2014-04-01 08:29:00	...	33.1	10.0	31.1	-9999.0	mostlycloudy

Exercise 3

Find all the rides that began from station Millennium Park.

```
[40]: bikes['from_station_name'] == 'Millennium Park'
bikes[filt].head(3)
```

	trip_id	usertype	gender	starttime	stoptime	...	temperature	visibility	wind_speed	precipitation	events
2164	1193566	Subscriber	Male	2014-02-20 19:06:00	2014-02-20 19:07:00	...	46.9	10.0	31.1	-9999.0	mostlycloudy
2165	1193760	Subscriber	Male	2014-02-20 20:47:00	2014-02-20 21:14:00	...	39.0	10.0	35.7	-9999.0	cloudy
2479	1320861	Subscriber	Male	2014-04-01 08:28:00	2014-04-01 08:29:00	...	33.1	10.0	31.1	-9999.0	mostlycloudy

Exercise 4

Find all the rides with wind speed less than 0. How is this possible?

```
[41]: # these are likely missing values.
# The dataset uses -9999 to represent missing values
filt = bikes['wind_speed'] < 0
bikes[filt].head(3)
```

	trip_id	usertype	gender	starttime	stoptime	...	temperature	visibility	wind_speed	precipitation	events
22990	8976097	Subscriber	Male	2016-03-19 10:08:00	2016-03-19 10:20:00	...	42.8	10.0	-9999.0	-9999.00	mostlycloudy
27168	10412628	Subscriber	Female	2016-06-30 11:47:00	2016-06-30 11:51:00	...	-9999.0	-9999.0	-9999.0	-9999.00	unknown
28368	10863258	Subscriber	Female	2016-07-21 21:02:29	2016-07-21 21:20:28	...	73.0	1.0	-9999.0	0.39	tstorms

Exercise 5

Find all the rides where the starting number of bikes at the station (dpcapacity_start) was more than 50.

```
[42]: filt = bikes['dpcapacity_start'] > 50
bikes[filt].head()
```

	trip_id	usertype	gender	starttime	stoptime	...	temperature	visibility	wind_speed	precipitation	events
36122	13228203	Subscriber	Male	2017-02-17 17:00:36	2017-02-17 17:23:27	...	63.0	10.0	10.4	-9999.0	partlycloudy
37617	13648506	Subscriber	Male	2017-04-14 18:44:47	2017-04-14 19:00:53	...	63.0	10.0	4.6	-9999.0	cloudy
37920	13750060	Subscriber	Male	2017-04-22 12:28:51	2017-04-22 12:44:14	...	55.9	10.0	12.7	-9999.0	mostlycloudy
37940	13754184	Subscriber	Female	2017-04-22 17:12:55	2017-04-22 17:23:44	...	57.9	10.0	12.7	-9999.0	partlycloudy
39102	14106813	Subscriber	Male	2017-05-21 20:40:10	2017-05-21 20:51:29	...	51.1	10.0	12.7	-9999.0	cloudy

Exercise 6

Did any rides happen in temperature over 100 degrees.

```
[43]: # no, a dataframe with no rows is returned.
filt = bikes['temperature'] > 100
bikes[filt]
```

	trip_id	usertype	gender	starttime	stoptime	...	temperature	visibility	wind_speed	precipitation	events

Read in the movie dataset by executing the cell below and use it for the following exercises.

```
[44]: import pandas as pd
movie = pd.read_csv('../data/movie.csv', index_col='title')
movie.head(3)
```

	year	color	content_rating	duration	director_name	...	plot_keywords	language	country	budget	imdb_score
title											
Avatar	2009.0	Color	PG-13	178.0	James Cameron	...	avatar future marine native paraplegic	English	USA	237000000.0	7.9
Pirates of the Caribbean: At World's End	2007.0	Color	PG-13	169.0	Gore Verbinski	...	goddess marriage ceremony marriage proposal pi...	English	USA	300000000.0	7.1
Spectre	2015.0	Color	PG-13	148.0	Sam Mendes	...	bomb espionage sequel spy terrorist	English	UK	245000000.0	6.8

Exercise 7

Select all movies that have Tom Hanks as actor1. How many of these movies has he starred in?

```
[45]: filt = movie['actor1'] == 'Tom Hanks'  
hanks_movies = movie[filt]  
hanks_movies.head(3)
```

	year	color	content_rating	duration	director_name	...	plot_keywords	language	country	budget	imdb_score
title											
Toy Story 3	2010.0	Color	G	103.0	Lee Unkrich	...	college day care escape teddy bear toy	English	USA	200000000.0	8.3
The Polar Express	2004.0	Color	G	100.0	Robert Zemeckis	...	boy christmas christmas eve north pole train	English	USA	165000000.0	6.6
Angels & Demons	2009.0	Color	PG-13	146.0	Ron Howard	...	conclave illuminati murder reference to bernin...	English	USA	150000000.0	6.7

He's starred in 24 movies

```
[46]: hanks_movies.shape
```

[46] : (24, 21)

Exercise 8

Select movies with an IMDB score greater than 9.

```
[47]: filt = movie['imdb_score'] > 9  
movie[filt]
```

Exercise 9

Write a function that accepts a single parameter to find the number of movies for a given content rating. Use the function to find the number of movies for ratings ‘R’, ‘PG-13’, and ‘PG’.

```
[48]: def count_rating(rating):
    filt = movie['content_rating'] == rating
    count = len(movie[filt])
    return f'There are {count} movies rated {rating}'
```

```
[49]: count_rating('R')
```

```
[49]: 'There are 2067 movies rated R'
```

```
[50]: count_rating('PG-13')
```

```
[50]: 'There are 1411 movies rated PG-13'
```

```
[51]: count_rating('PG')
```

```
[51]: 'There are 686 movies rated PG'
```

2.6 6. Boolean Selection Multiple Conditions

```
[52]: bikes = pd.read_csv('../data/bikes.csv')
bikes.head(3)
```

	trip_id	usertype	gender	starttime	stoptime	...	temperature	visibility	wind_speed	precipitation	events
0	7147	Subscriber	Male	2013-06-28 19:01:00	2013-06-28 19:17:00	...	73.9	10.0	12.7	-9999.0	mostlycloudy
1	7524	Subscriber	Male	2013-06-28 22:53:00	2013-06-28 23:03:00	...	69.1	10.0	6.9	-9999.0	partlycloudy
2	10927	Subscriber	Male	2013-06-30 14:43:00	2013-06-30 15:01:00	...	73.0	10.0	16.1	-9999.0	mostlycloudy

Exercise 1

Find all the rides where visibility was between 0 and 2.

```
[53]: filt1 = bikes['visibility'] >= 0
filt2 = bikes['visibility'] <= 2
filt = filt1 & filt2
bikes[filt].head(3)
```

	trip_id	usertype	gender	starttime	stoptime	...	temperature	visibility	wind_speed	precipitation	events
689	528940	Subscriber	Male	2013-09-19 10:47:00	2013-09-19 10:53:00	...	69.8	1.8	6.9	0.03	tstorms
972	666814	Subscriber	Female	2013-10-03 06:31:00	2013-10-03 06:51:00	...	64.4	0.5	3.5	-9999.00	fog
973	666837	Subscriber	Male	2013-10-03 06:36:00	2013-10-03 06:50:00	...	64.4	0.5	3.5	-9999.00	fog

Exercise 2

Find all the rides with trip duration less than 100 done by females.

```
[54]: filt1 = bikes['tripduration'] < 100
filt2 = bikes['gender'] == 'Female'
filt = filt1 & filt2
bikes[filt].head(3)
```

	trip_id	usertype	gender	starttime	stoptime	...	temperature	visibility	wind_speed	precipitation	events
2485	1322828	Subscriber	Female	2014-04-01 15:43:00	2014-04-01 15:44:00	...	46.0	10.0	23.0	-9999.0	partlycloudy
3366	1694713	Subscriber	Female	2014-05-17 14:55:00	2014-05-17 14:56:00	...	57.9	10.0	11.5	-9999.0	partlycloudy
3629	1835879	Subscriber	Female	2014-05-27 07:55:00	2014-05-27 07:56:00	...	73.0	10.0	9.2	-9999.0	cloudy

Exercise 3

Find all the rides from ‘Daley Center Plaza’ to ‘Michigan Ave & Washington St’

```
[55]: filt1 = bikes['to_station_name'] == 'Daley Center Plaza'
filt2 = bikes['from_station_name'] == 'Michigan Ave & Washington St'
filt = filt1 & filt2
bikes[filt]
```

	trip_id	usertype	gender	starttime	stoptime	...	temperature	visibility	wind_speed	precipitation	events
25247	9733603	Subscriber	Male	2016-05-26 15:24:00	2016-05-26 15:27:00	...	82.0	10.0	12.7	-9999.0	mostlycloudy

Exercise 4

Find all the rides with temperature greater than 90 or visibility equal to 1 or wind speed greater than 20

```
[56]: filt1 = bikes['temperature'] > 90
filt2 = bikes['visibility'] == 1
filt3 = bikes['wind_speed'] > 20
```

```
filt = filt1 | filt2 | filt3
bikes[filt].head(3)
```

	trip_id	usertype	gender	starttime	stoptime	...	temperature	visibility	wind_speed	precipitation	events
54	69250	Subscriber	Male	2013-07-15:13:00	2013-07-16 15:18:00	...	91.0	10.0	8.1	-9999.0	mostlycloudy
55	69320	Subscriber	Male	2013-07-15:31:00	2013-07-16 15:37:00	...	91.0	10.0	8.1	-9999.0	mostlycloudy
56	69613	Subscriber	Male	2013-07-16:35:00	2013-07-16 16:42:00	...	91.0	10.0	10.4	-9999.0	mostlycloudy

Exercise 5

Invert the condition from exercise 4.

```
[57]: filt1 = bikes['temperature'] > 90
filt2 = bikes['visibility'] == 1
filt3 = bikes['wind_speed'] > 20
filt = filt1 | filt2 | filt3
bikes[~filt].head(3)
```

	trip_id	usertype	gender	starttime	stoptime	...	temperature	visibility	wind_speed	precipitation	events
0	7147	Subscriber	Male	2013-06-19:01:00	2013-06-28 19:17:00	...	73.9	10.0	12.7	-9999.0	mostlycloudy
1	7524	Subscriber	Male	2013-06-22:53:00	2013-06-28 23:03:00	...	69.1	10.0	6.9	-9999.0	partlycloudy
2	10927	Subscriber	Male	2013-06-14:43:00	2013-06-30 15:01:00	...	73.0	10.0	16.1	-9999.0	mostlycloudy

Exercise 6

Are there any rides where the weather event was snow and the temperature was greater than 40?

```
[58]: # no
filt1 = bikes['events'] == 'snow'
filt2 = bikes['temperature'] > 40
filt = filt1 & filt2
bikes[filt]
```

	trip_id	usertype	gender	starttime	stoptime	...	temperature	visibility	wind_speed	precipitation	events
--	---------	----------	--------	-----------	----------	-----	-------------	------------	------------	---------------	--------

Read in the movie dataset by executing the cell below and use it for the following exercises.

```
[59]: import pandas as pd
movie = pd.read_csv('../data/movie.csv', index_col='title')
movie.head(3)
```

	year	color	content_rating	duration	director_name	...	plot_keywords	language	country	budget	imdb_score
title						...					
Avatar	2009.0	Color	PG-13	178.0	James Cameron	...	avatar future marine native paraplegic	English	USA	237000000.0	7.9
Pirates of the Caribbean: At World's End	2007.0	Color	PG-13	169.0	Gore Verbinski	...	goddess marriage ceremony marriage proposal pi...	English	USA	300000000.0	7.1
Spectre	2015.0	Color	PG-13	148.0	Sam Mendes	...	bomb espionage sequel spy terrorist	English	UK	245000000.0	6.8

Exercise 7

Select all movies from the 1970s.

```
[60]: filt1 = movie['year'] >= 1970
filt2 = movie['year'] <= 1979
filt = filt1 & filt2
movie[filt].head(3)
```

	year	color	content_rating	duration	director_name	...	plot_keywords	language	country	budget	imdb_score
title						...					
All That Jazz	1979.0	Color	R	123.0	Bob Fosse	...	dancer editing stand up comedian surgery vomiting	English	USA	NaN	7.8
Superman	1978.0	Color	PG	188.0	Richard Donner	...	1970s clark kent planet superhero year 1978	English	USA	55000000.0	7.3
Solaris	1972.0	Black and White	PG	115.0	Andrei Tarkovsky	...	hallucination ocean psychologist scientist spa...	Russian	Soviet Union	1000000.0	8.1

Exercise 8

Select all movies from the 1970s that had IMDB scores greater than 8.

```
[61]: filt1 = movie['year'] >= 1970
filt2 = movie['year'] <= 1979
filt3 = movie['imdb_score'] > 8
filt = filt1 & filt2 & filt3
movie[filt].head(3)
```

	year	color	content_rating	duration	director_name	...	plot_keywords	language	country	budget	imdb_score
title						...					
Solaris	1972.0	Black and White	PG	115.0	Andrei Tarkovsky	...	hallucination ocean psychologist scientist spa...	Russian	Soviet Union	1000000.0	8.1
Apocalypse Now	1979.0	Color	R	289.0	Francis Ford Coppola	...	army green beret insanity jungle vietnam	English	USA	31500000.0	8.5
The Deer Hunter	1978.0	Color	R	183.0	Michael Cimino	...	escape friend party pittsburgh steelers vietnam	English	UK	15000000.0	8.2

Exercise 9

Select movies that were rated either R, PG-13, or PG.

```
[62]: filt = movie['content_rating'].isin(['R', 'PG-13', 'PG'])
movie[filt].head(3)
```

	year	color	content_rating	duration	director_name	...	plot_keywords	language	country	budget	imdb_score
title						...					
Avatar	2009.0	Color	PG-13	178.0	James Cameron	...	avatar future marine native paraplegic	English	USA	237000000.0	7.9
Pirates of the Caribbean: At World's End	2007.0	Color	PG-13	169.0	Gore Verbinski	...	goddess marriage ceremony marriage proposal pi...	English	USA	300000000.0	7.1
Spectre	2015.0	Color	PG-13	148.0	Sam Mendes	...	bomb espionage sequel spy terrorist	English	UK	245000000.0	6.8

Exercise 10

Select movies that are either rated PG-13 or were made after 2010.

```
[63]: filt1 = movie['content_rating'] == 'PG-13'
filt2 = movie['year'] > 2010
filt = filt1 | filt2
movie[filt].head(3)
```

	year	color	content_rating	duration	director_name	...	plot_keywords	language	country	budget	imdb_score
title						...					
Avatar	2009.0	Color	PG-13	178.0	James Cameron	...	avatar future marine native paraplegic	English	USA	237000000.0	7.9
Pirates of the Caribbean: At World's End	2007.0	Color	PG-13	169.0	Gore Verbinski	...	goddess marriage ceremony marriage proposal pi...	English	USA	300000000.0	7.1
Spectre	2015.0	Color	PG-13	148.0	Sam Mendes	...	bomb espionage sequel spy terrorist	English	UK	245000000.0	6.8

Exercise 11

Find all the movies that have at least one of the three actors with more than 10,000 Facebook likes.

```
[64]: filt1 = movie['actor1_fb'] > 10000
filt2 = movie['actor2_fb'] > 10000
filt3 = movie['actor3_fb'] > 10000
filt = filt1 | filt2 | filt3
movie[filt].head(3)
```

	year	color	content_rating	duration	director_name	...	plot_keywords	language	country	budget	imdb_score
title						...					
Pirates of the Caribbean: At World's End	2007.0	Color	PG-13	169.0	Gore Verbinski	...	goddess marriage ceremony marriage proposal pi...	English	USA	300000000.0	7.1
Spectre	2015.0	Color	PG-13	148.0	Sam Mendes	...	bomb espionage sequel spy terrorist	English	UK	245000000.0	6.8
The Dark Knight Rises	2012.0	Color	PG-13	164.0	Christopher Nolan	...	deception imprisonment lawlessness police offi...	English	USA	250000000.0	8.5

Exercise 12

Invert the condition from exercise 10. In words, what have you selected?

The following selects non-PG-13 movies made in the year 2010 or before.

```
[65]: filt1 = movie['content_rating'] == 'PG-13'
filt2 = movie['year'] > 2010
filt = filt1 | filt2
movie[~filt].head(3)
```

	year	color	content_rating	duration	director_name	...	plot_keywords	language	country	budget	imdb_score
title											
Star Wars: Episode VII - The Force Awakens	NaN	NaN	NaN	NaN	Doug Walker	...	NaN	NaN	NaN	NaN	7.1
Tangled	2010.0	Color	PG	100.0	Nathan Greno	...	17th century based on fairy tale disney flower...	English	USA	260000000.0	7.8
Harry Potter and the Half-Blood Prince	2009.0	Color	PG	153.0	David Yates	...	blood book love potion professor	English	UK	250000000.0	7.5

2.7 7. Boolean Selection More

```
[66]: import pandas as pd
bikes = pd.read_csv('../data/bikes.csv')
```

Exercise 1

Select the wind speed column as a Series and assign it to a variable. Are there any negative wind speeds?

```
[67]: wind = bikes['wind_speed']
wind.head(3)
```

```
[67]: 0    12.7
1    6.9
2    16.1
Name: wind_speed, dtype: float64
```

Yes, there is really strong negative wind! Or maybe its just bad data...

```
[68]: filt = wind < 0
wind[filt].head(3)
```

```
[68]: 22990   -9999.0
27168   -9999.0
28368   -9999.0
Name: wind_speed, dtype: float64
```

Exercise 2

Select all wind speed values between 12 and 16.

```
[69]: filt = wind.between(12, 16)
wind[filt].head(3)
```

```
[69]: 0    12.7
       6    15.0
       9    12.7
Name: wind_speed, dtype: float64
```

Exercise 3

Select the events and gender columns for all trip durations longer than 1,000 seconds.

```
[70]: filt = bikes['tripduration'] > 1000
cols = ['events', 'gender']
bikes.loc[filt, cols].head(3)
```

	events	gender
2	mostlycloudy	Male
8	cloudy	Male
10	mostlycloudy	Male

Read in the movie dataset by executing the cell below and use it for the following exercises.

```
[71]: import pandas as pd
movie = pd.read_csv('../data/movie.csv', index_col='title')
movie.head(3)
```

	year	color	content_rating	duration	director_name	...	plot_keywords	language	country	budget	imdb_score
title											
Avatar	2009.0	Color	PG-13	178.0	James Cameron	...	avatar future marine native paraplegic	English	USA	237000000.0	7.9
Pirates of the Caribbean: At World's End	2007.0	Color	PG-13	169.0	Gore Verbinski	...	goddess marriage ceremony marriage proposal pi...	English	USA	300000000.0	7.1
Spectre	2015.0	Color	PG-13	148.0	Sam Mendes	...	bomb espionage sequel spy terrorist	English	UK	245000000.0	6.8

Exercise 4

Select all the movies such that the Facebook likes for actor 2 are greater than those for actor 1.

There are none!

```
[72]: filt = movie['actor2_fb'] > movie['actor1_fb']
movie[filt]
```

	year	color	content_rating	duration	director_name	...	plot_keywords	language	country	budget	imdb_score
title											

Exercise 5

Select the year, content rating, and IMDB score columns for movies from the year 2016 with IMDB score less than 4.

```
[73]: filt1 = movie['year'] == 2016
filt2 = movie['imdb_score'] < 4
filt = filt1 & filt2
cols = ['year', 'content_rating', 'imdb_score']

movie.loc[filt, cols]
```

	year	content_rating	imdb_score
title			
Fifty Shades of Black	2016.0	R	3.5
Cabin Fever	2016.0	Not Rated	3.7
God's Not Dead 2	2016.0	PG	3.4

Exercise 6

Select all the movies that are missing values for content rating.

```
[74]: filt = movie['content_rating'].isna()
movie[filt].head(3)
```

	year	color	content_rating	duration	director_name	...	plot_keywords	language	country	budget	imdb_score
title											
Star Wars: Episode VII - The Force Awakens	NaN	NaN	NaN	NaN	Doug Walker	...	NaN	NaN	NaN	NaN	7.1
Godzilla Resurgence	2016.0	Color	NaN	120.0	Hideaki Anno	...	blood godzilla monster sequel	Japanese	Japan	NaN	8.2
Harry Potter and the Deathly Hallows: Part II	2011.0	Color	NaN	NaN	Matt Birch	...	NaN	English	UK	NaN	7.5

Exercise 7

Select all the movies that are missing values for both the gross and budget columns. Return just those columns to verify that those values are indeed missing.

```
[75]: filt = movie['gross'].isna() & movie['budget'].isna()
cols = ['gross', 'budget']
movie.loc[filt, cols].head(3)
```

	gross	budget
title		
Star Wars: Episode VII - The Force Awakens	NaN	NaN
The Lovers	NaN	NaN
Godzilla Resurgence	NaN	NaN

Exercise 8

Write a function `find_missing` that has three parameters, `df`, `col1` and `col2` where `df` is a DataFrame and `col1` and `col2` are column names. This function should return all the rows of the DataFrame where `col1` and `col2` are missing. Only return the two columns as well. Answer Exercise 7 with this function.

```
[76]: def find_missing(df, col1, col2):
    filt = df[col1].isna() & df[col2].isna()
    cols = [col1, col2]

    return df.loc[filt, cols]
```

```
[77]: movie_missing = find_missing(movie, 'gross', 'budget')
movie_missing.head(3)
```

		gross	budget
	title		
	Star Wars: Episode VII - The Force Awakens	NaN	NaN
	The Lovers	NaN	NaN
	Godzilla Resurgence	NaN	NaN

2.8 8. Filtering with the query Method

Exercise 1

Use the `query` method to select trip durations between 5000 and 10000.

```
[78]: bikes.query('5000 <= tripduration <= 10000').head(3)
```

	trip_id	usertype	gender	starttime	stoptime	...	temperature	visibility	wind_speed	precipitation	events
18	40924	Subscriber	Male	2013-07-09 13:12:00	2013-07-09 14:42:00	...	79.0	10.0	13.8	0.0	cloudy
40	61401	Subscriber	Female	2013-07-14 14:08:00	2013-07-14 15:53:00	...	87.1	10.0	8.1	-9999.0	partlycloudy
77	87005	Subscriber	Female	2013-07-21 11:35:00	2013-07-21 13:54:00	...	82.9	10.0	5.8	-9999.0	mostlycloudy

Exercise 2

Use the `query` method to select trip durations between 5000 and 10000 when the weather was snow or rain. Retrieve the same data with boolean selection.

```
[79]: bikes.query('5000 <= tripduration <= 10000 and events in ["snow", "rain"]').head(3)
```

	trip_id	usertype	gender	starttime	stoptime	...	temperature	visibility	wind_speed	precipitation	events
8506	3846762	Subscriber	Male	2014-10-04 12:33:00	2014-10-04 14:06:00	...	42.1	8.0	17.3	0.02	rain
13355	5650639	Subscriber	Male	2015-06-15 11:41:00	2015-06-15 13:43:00	...	75.9	10.0	4.6	0.00	rain
22155	8723460	Subscriber	Male	2016-02-09 10:09:00	2016-02-09 12:28:00	...	16.0	4.0	16.1	0.00	snow

```
[80]: filt1 = bikes['tripduration'].between(5000, 10000)
filt2 = bikes['events'].isin(['snow', 'rain'])
filt = filt1 & filt2
bikes[filt].head(3)
```

	trip_id	usertype	gender	starttime	stoptime	...	temperature	visibility	wind_speed	precipitation	events
8506	3846762	Subscriber	Male	2014-10-04 12:33:00	2014-10-04 14:06:00	...	42.1	8.0	17.3	0.02	rain
13355	5650639	Subscriber	Male	2015-06-15 11:41:00	2015-06-15 13:43:00	...	75.9	10.0	4.6	0.00	rain
22155	8723460	Subscriber	Male	2016-02-09 10:09:00	2016-02-09 12:28:00	...	16.0	4.0	16.1	0.00	snow

Exercise 3

Use the `query` method to select trip durations between 5000 and 10000 when it was snow or rain. Create a list outside of the query method to hold the weather and reference that variable with `@weather` within query.

```
[81]: weather = ["snow", "rain"]
bikes.query('5000 <= tripduration <= 10000 and events in @weather').head(3)
```

	trip_id	usertype	gender	starttime	stoptime	...	temperature	visibility	wind_speed	precipitation	events
8506	3846762	Subscriber	Male	2014-10-04 12:33:00	2014-10-04 14:06:00	...	42.1	8.0	17.3	0.02	rain
13355	5650639	Subscriber	Male	2015-06-15 11:41:00	2015-06-15 13:43:00	...	75.9	10.0	4.6	0.00	rain
22155	8723460	Subscriber	Male	2016-02-09 10:09:00	2016-02-09 12:28:00	...	16.0	4.0	16.1	0.00	snow

Read in the movie dataset by executing the cell below and use it for the following exercises.

```
[82]: import pandas as pd
pd.set_option('display.max_columns', 50)
movie = pd.read_csv('../data/movie.csv', index_col='title')
movie.head(3)
```

	year	color	content_rating	duration	director_name	...	plot_keywords	language	country	budget	imdb_score
title						...					
Avatar	2009.0	Color	PG-13	178.0	James Cameron	...	avatar future marine native paraplegic	English	USA	237000000.0	7.9
Pirates of the Caribbean: At World's End	2007.0	Color	PG-13	169.0	Gore Verbinski	...	goddess marriage ceremony marriage proposal pi...	English	USA	300000000.0	7.1
Spectre	2015.0	Color	PG-13	148.0	Sam Mendes	...	bomb espionage sequel spy terrorist	English	UK	245000000.0	6.8

Exercise 4

Use the `query` method to find all movies where the total number of Facebook likes for all three actors is greater than 50,000.

```
[83]: movie.query('actor1_fb + actor2_fb + actor3_fb > 50000').head(3)
```

	year	color	content_rating	duration	director_name	...	plot_keywords	language	country	budget	imdb_score
title						...					
The Dark Knight Rises	2012.0	Color	PG-13	164.0	Christopher Nolan	...	deception imprisonment lawlessness police offi...	English	USA	250000000.0	8.5
Avengers: Age of Ultron	2015.0	Color	PG-13	141.0	Joss Whedon	...	artificial intelligence based on comic book ca...	English	USA	250000000.0	7.5
The Avengers	2012.0	Color	PG-13	173.0	Joss Whedon	...	alien invasion assassin battle iron man soldier	English	USA	220000000.0	8.1

Exercise 5

Select all the movies the number of user voters is less than 10 times the number of reviews.

```
[84]: movie.query('num_voted_users < 10 * num_reviews').head(3)
```

	year	color	content_rating	duration	director_name	...	plot_keywords	language	country	budget	imdb_score
title						...					
Indignation	2016.0	Color	R	110.0	James Schamus	...	based on novel	Hebrew	USA	NaN	7.8
Pete's Dragon	2016.0	Color	PG	102.0	David Lowery	...	NaN	English	USA	65000000.0	7.3
Kicks	2016.0	Color	R	80.0	Justin Tipping	...	NaN	English	USA	NaN	7.8

Exercise 6

Select all the movies made in the 1990s that were rated R with an IMDB score greater than 8.

```
[85]: movie.query('1990 <= year <= 1999 and content_rating == "R" and imdb_score > 8').head(3)
```

	year	color	content_rating	duration	director_name	...	plot_keywords	language	country	budget	imdb_score
title											
Terminator 2: Judgment Day	1991.0	Color	R	153.0	James Cameron	...	future liquid metal multiple cameos sexy woman...	English	USA	102000000.0	8.5
Braveheart	1995.0	Color	R	178.0	Mel Gibson	...	14th century legend revolt scotland tyranny	English	USA	72000000.0	8.4
Saving Private Ryan	1998.0	Color	R	169.0	Steven Spielberg	...	army invasion killed in action normandy soldier	English	USA	70000000.0	8.6

2.9 9. Miscellaneous Subset Selection

Part III

Essential Series Commands

Chapter 3

Solutions

3.1 1. Series Attributes and Statistical Methods

```
[1]: import pandas as pd
import numpy as np
movie = pd.read_csv('../data/movie.csv', index_col='title')
movie.head(3)
```

	year	color	content_rating	duration	director_name	...	plot_keywords	language	country	budget	imdb_score
title											
Avatar	2009.0	Color	PG-13	178.0	James Cameron	...	avatar future marine native paraplegic	English	USA	237000000.0	7.9
Pirates of the Caribbean: At World's End	2007.0	Color	PG-13	169.0	Gore Verbinski	...	goddess marriage ceremony marriage proposal pi...	English	USA	300000000.0	7.1
Spectre	2015.0	Color	PG-13	148.0	Sam Mendes	...	bomb espionage sequel spy terrorist	English	UK	245000000.0	6.8

```
[2]: score = movie['imdb_score']
score.head()
```

```
[2]: title
Avatar                               7.9
Pirates of the Caribbean: At World's End    7.1
Spectre                                6.8
The Dark Knight Rises                  8.5
Star Wars: Episode VII - The Force Awakens   7.1
Name: imdb_score, dtype: float64
```

Exercise 1

What is the data type of `score` and how many values does it contain?

```
[3]: score.dtype
```

```
[3]: dtype('float64')
```

```
[4]: score.size
```

[4]: 4916

Or use the `len` method

[5]: `len(score)`

[5]: 4916

Exercise 2

What is the maximum and minimum score?

[6]: `score.max()`

[6]: 9.5

[7]: `score.min()`

[7]: 1.6

Exercise 3

How many movies have scores greater than 6?

[8]: `(score > 6).sum()`

[8]: 3368

Exercise 4

How many movies have scores greater than 4 and less than 7?

[9]: `filt1 = score > 4
filt2 = score < 7
filt = filt1 & filt2
filt.sum()`

[9]: 3021

Or in one line of code

[10]: `((score > 4) & (score < 7)).sum()`

[10]: 3021

Or use the `between` method.

[11]: `score.between(4, 7, inclusive=False).sum()`

[11]: 3021

Exercise 5

Find the difference between the median and mean of the scores.

```
[12]: score.median() - score.mean()
```

```
[12]: 0.16257119609438497
```

Exercise 6

Add 1 to every value of `score` and then calculate the median.

```
[13]: (score + 1).median()
```

```
[13]: 7.6
```

Exercise 7

Calculate the median of `score` and add 1 to this. Why is this value the same as Exercise 7?

```
[14]: score.median() + 1
```

```
[14]: 7.6
```

Exercise 8

Return a Series that has only scores above the 99.9th percentile

```
[15]: filt = score > score.quantile(.999)  
score[filt]
```

```
[15]: title  
The Shawshank Redemption      9.3  
Towering Inferno              9.5  
Dekalog                      9.1  
The Godfather                  9.2  
Kickboxer: Vengeance          9.1  
Name: imdb_score, dtype: float64
```

Exercise 9

Assign the gross column of the movie dataset to its own variable name. Round it to the nearest million.

```
[16]: gross = movie['gross']  
gross.head()
```

```
[16]: title  
Avatar                         760505847.0  
Pirates of the Caribbean: At World's End 309404152.0  
Spectre                         200074175.0  
The Dark Knight Rises           448130642.0  
Star Wars: Episode VII - The Force Awakens      NaN  
Name: gross, dtype: float64
```

```
[17]: gross.round(-6).head()
```

```
[17]: title
Avatar                      761000000.0
Pirates of the Caribbean: At World's End 309000000.0
Spectre                      200000000.0
The Dark Knight Rises        448000000.0
Star Wars: Episode VII - The Force Awakens      NaN
Name: gross, dtype: float64
```

Exercise 10

Calculate the cumulative sum of the gross Series and then select the 99th integer location.

```
[18]: gross.cumsum().iloc[99]
```

```
[18]: 23119723385.0
```

Exercise 11

Select the first 100 values of the gross Series and then calculate the sum. Does the result match exercise 11.

```
[19]: gross.iloc[:100].sum()
```

```
[19]: 23119723385.0
```

3.2 2. Series Missing Value Methods

```
[20]: import pandas as pd
movie = pd.read_csv('../data/movie.csv', index_col='title')
duration = movie['duration']
actor1_fb = movie['actor1_fb']
actor2_fb = movie['actor2_fb']
actor1_fb.head()
```

```
[20]: title
Avatar                      1000.0
Pirates of the Caribbean: At World's End 40000.0
Spectre                      11000.0
The Dark Knight Rises        27000.0
Star Wars: Episode VII - The Force Awakens     131.0
Name: actor1_fb, dtype: float64
```

```
[21]: actor2_fb.head()
```

```
[21]: title
Avatar                      936.0
Pirates of the Caribbean: At World's End 5000.0
Spectre                      393.0
The Dark Knight Rises        23000.0
Star Wars: Episode VII - The Force Awakens     12.0
Name: actor2_fb, dtype: float64
```

Exercise 1

What percentage of actor 1 Facebook likes are missing?

```
[22]: actor1_fb.isna().mean() * 100
```

```
[22]: 0.14239218877135884
```

Exercise 2

Use the `notna` method to find the number of non-missing values in the actor 1 Facebook like column. Verify this number is the same as the `count` method.

```
[23]: actor2_fb.notna().sum()
```

```
[23]: 4903
```

```
[24]: actor2_fb.count()
```

```
[24]: 4903
```

Exercise 3

Fill the missing values of `actor1_fb` with the maximum of `actor2_fb`. Save this result to variable `actor1_fb_full`

```
[25]: max_fb2 = actor2_fb.max()  
max_fb2
```

```
[25]: 137000.0
```

```
[26]: actor1_fb_full = actor1_fb.fillna(max_fb2)  
actor1_fb_full.head()
```

```
[26]: title  
Avatar 1000.0  
Pirates of the Caribbean: At World's End 40000.0  
Spectre 11000.0  
The Dark Knight Rises 27000.0  
Star Wars: Episode VII - The Force Awakens 131.0  
Name: actor1_fb, dtype: float64
```

Exercise 4

Verify the results of Exercise 3 by selecting just the values of `actor1_fb_full` that were filled by `actor2_fb`.

```
[27]: filt = actor1_fb.isna()  
actor1_fb_full[filt]
```

```
[27]: title  
Pink Ribbons, Inc. 137000.0  
Sex with Strangers 137000.0  
The Harvest/La Cosecha 137000.0
```

```
Ayurveda: Art of Being      137000.0
The Brain That Sings       137000.0
The Blood of My Brother    137000.0
Counting                   137000.0
Name: actor1_fb, dtype: float64
```

[28]: `actor2_fb.max()`

[28]: 137000.0

Exercise 5

Use the `duration` Series and test whether each movie is greater than 100. Assign the resulting Series to `filt`. Then test whether the `duration` Series is less than or equal to 100 and assign it to `filt2`. Call `thesum` method on both of these new Series and add their results together. Why doesn't this result equal the total length of the Series? Shouldn't a value be either greater than 100 or less than or equal to 100?`

[29]: `filt = duration > 100`
`filt2 = duration <= 100`

[30]: `filt.sum() + filt2.sum()`

[30]: 4901

Missing values always get evaluated as `False` for comparisons. You need to add in the missing values to get the total length of the Series.

[31]: `filt.sum() + filt2.sum() + duration.isna().sum()`

[31]: 4916

[32]: `len(duration)`

[32]: 4916

Exercise 6

How many missing values are there in the `year` column.

[33]: `movie['year'].isna().sum()`

[33]: 106

Exercise 7

Select the language column as a Series and assign it to a variable with the same name. Create a variable `filt` that determines whether a language is missing. Create a new Series that fills in all missing languages with 'English' and assign it to the variable `language2`. Output both `language` and `language2` for the movies that originally had missing values.

[34]: `language = movie['language']`
`filt = language.isna()`

```
language2 = language.fillna('English')
```

```
[35]: language[filt]
```

```
[35]: title
Star Wars: Episode VII - The Force Awakens      NaN
10,000 B.C.                                     NaN
Unforgettable                                    NaN
September Dawn                                    NaN
Alpha and Omega 4: The Legend of the Saw Tooothed Cave  NaN
Silent Movie                                     NaN
Love's Abiding Joy                                NaN
Kickboxer: Vengeance                            NaN
A Fine Step                                      NaN
Intolerance: Love's Struggle Throughout the Ages  NaN
The Big Parade                                    NaN
Over the Hill to the Poorhouse                  NaN
Name: language, dtype: object
```

```
[36]: language2[filt]
```

```
[36]: title
Star Wars: Episode VII - The Force Awakens      English
10,000 B.C.                                     English
Unforgettable                                    English
September Dawn                                    English
Alpha and Omega 4: The Legend of the Saw Tooothed Cave  English
Silent Movie                                     English
Love's Abiding Joy                                English
Kickboxer: Vengeance                            English
A Fine Step                                      English
Intolerance: Love's Struggle Throughout the Ages  English
The Big Parade                                    English
Over the Hill to the Poorhouse                  English
Name: language, dtype: object
```

Exercise 7

Repeat exercise 7 without first assigning the language column to a variable. Reference it by using *just the brackets*. Still make a variable `language2` to verify th

```
[37]: filt = movie['language'].isna()
language2 = movie['language'].fillna('English')
language2[filt]
```

```
[37]: title
Star Wars: Episode VII - The Force Awakens      English
10,000 B.C.                                     English
Unforgettable                                    English
September Dawn                                    English
Alpha and Omega 4: The Legend of the Saw Tooothed Cave  English
```

```
Silent Movie           English
Love's Abiding Joy    English
Kickboxer: Vengeance English
A Fine Step           English
Intolerance: Love's Struggle Throughout the Ages English
The Big Parade        English
Over the Hill to the Poorhouse English
Name: language, dtype: object
```

Exercise 8

Select the `gross` column, and drop all missing values from it. Confirm that the new length of the resulting Series is correct.

```
[38]: gd = movie['gross'].dropna()
len(gd)
```

[38]: 4054

```
[39]: movie['gross'].isna().sum()
```

[39]: 862

```
[40]: movie['gross'].isna().sum() + len(gd)
```

[40]: 4916

Exercise 9

Read in `girl_height.csv` as a DataFrame and output all of the data. The average height for each age is found in the `height` column. Assign the `height_na` Series to a variable. Notice that all ages from 2 through 12 are missing, but all other ages have the same value as the `height` column. Use the `interpolate` method to fill in the missing values with method ‘linear’, ‘quadratic’, and ‘cubic’. Save each interpolated Series to a variable with the same name as its method.

```
[41]: girl_height = pd.read_csv('../data/girl_height.csv')
girl_height
```

	age	height	height_na
0	0	19.00	19.00
1	1	29.00	29.00
2	2	34.00	NaN
3	3	37.00	NaN
4	4	39.50	NaN
5	5	42.00	NaN
6	6	45.50	NaN
7	7	48.00	NaN
8	8	50.00	NaN
9	9	52.50	NaN
10	10	54.00	NaN
11	11	56.00	NaN
12	12	59.50	NaN
13	13	61.50	61.50
14	14	63.50	63.50
15	15	64.00	64.00
16	16	64.25	64.25
17	17	64.25	64.25
18	18	64.25	64.25

```
[42]: hna = girl_height['height_na']
```

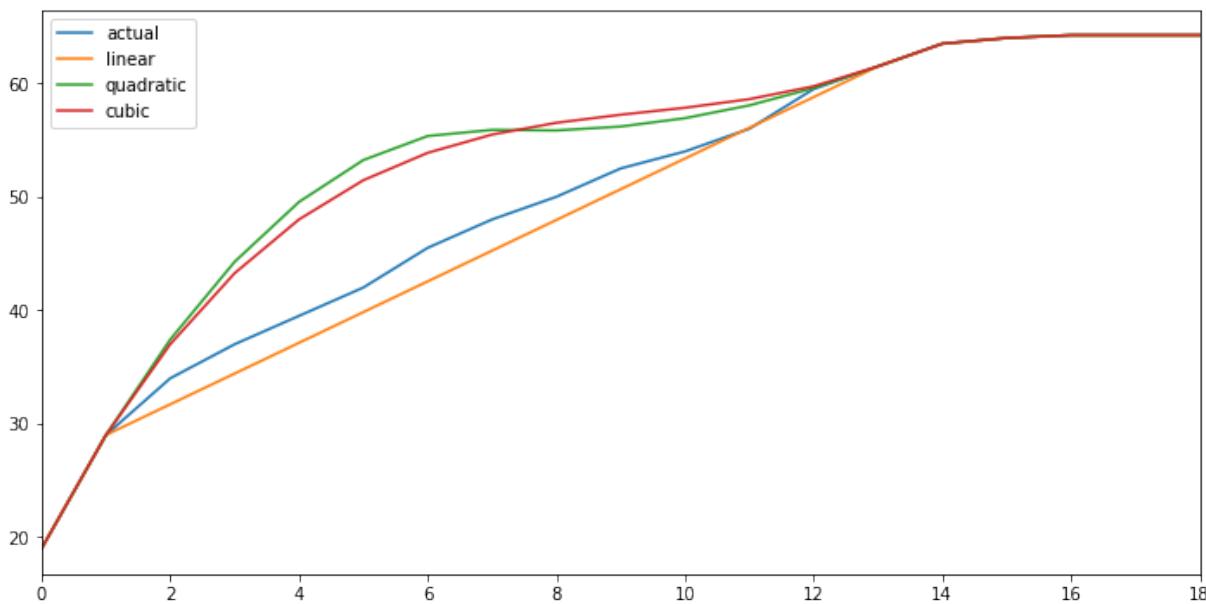
```
[43]: linear = hna.interpolate('linear')
quadratic = hna.interpolate('quadratic')
cubic = hna.interpolate('cubic')
```

Exercise 10

Uncomment and run the commands below to plot each interpolated Series. Which one provides the best estimate for height?

```
[44]: %matplotlib inline
girl_height['height'].plot(figsize=(12, 6), label='actual', legend=True)
linear.plot(label='linear', legend=True)
quadratic.plot(label='quadratic', legend=True)
cubic.plot(label='cubic', legend=True)
```

```
[44]: <matplotlib.axes._subplots.AxesSubplot at 0x114fb7390>
```



3.3 3. Series Sorting, Ranking, and Uniqueness

```
[45]: movie = pd.read_csv('../data/movie.csv', index_col='title')
movie.head(3)
```

	year	color	content_rating	duration	director_name	...	plot_keywords	language	country	budget	imdb_score
title											
Avatar	2009.0	Color	PG-13	178.0	James Cameron	...	avatar future marine native paraplegic	English	USA	237000000.0	7.9
Pirates of the Caribbean: At World's End	2007.0	Color	PG-13	169.0	Gore Verbinski	...	goddess marriage ceremony marriage proposal pi...	English	USA	300000000.0	7.1
Spectre	2015.0	Color	PG-13	148.0	Sam Mendes	...	bomb espionage sequel spy terrorist	English	UK	245000000.0	6.8

Exercise 1

Select the column holding the number of reviews as a Series and sort it from greatest to least.

```
[46]: movie['num_reviews'].sort_values(ascending=False).head()
```

```
[46]: title
The Dark Knight Rises      813.0
Prometheus                  775.0
Django Unchained            765.0
Skyfall                      750.0
Mad Max: Fury Road          739.0
Name: num_reviews, dtype: float64
```

Exercise 2

Find the number of unique actors in each of the actor columns. Do not count missing values. Use three separate calls to `nunique`.

```
[47]: movie['actor1'].nunique()
```

[47]: 2095

```
[48]: movie['actor2'].nunique()
```

[48]: 3030

```
[49]: movie['actor3'].nunique()
```

[49]: 3519

Exercise 3

Select the year column, sort it, and drop any duplicates?

```
[50]: movie['year'].sort_values().drop_duplicates().head()
```

[50]: title

Intolerance: Love's Struggle Throughout the Ages	1916.0
Over the Hill to the Poorhouse	1920.0
The Big Parade	1925.0
Metropolis	1927.0
The Broadway Melody	1929.0

Name: year, dtype: float64

Exercise 4

Get the same result as Exercise 3 by dropping duplicates first and then sort. Which method is faster?

```
[51]: movie['year'].drop_duplicates().sort_values().head()
```

[51]: title

Intolerance: Love's Struggle Throughout the Ages	1916.0
Over the Hill to the Poorhouse	1920.0
The Big Parade	1925.0
Metropolis	1927.0
Pandora's Box	1929.0

Name: year, dtype: float64

It's faster to drop duplicates first and then sort.

```
[52]: %timeit -n 5 movie['year'].sort_values().drop_duplicates().head()
```

1.63 ms ± 57.7 µs per loop (mean ± std. dev. of 7 runs, 5 loops each)

```
[53]: %timeit -n 5 movie['year'].drop_duplicates().sort_values().head()
```

1.07 ms ± 70 µs per loop (mean ± std. dev. of 7 runs, 5 loops each)

Exercise 5

Rank each movie by duration from greatest to least and then sort this ranking from least to greatest. Output the top 10 values. Do you get the same result by sorting the duration from greatest to least?

```
[54]: # yes, they are the same
movie['duration'].rank(ascending=False).sort_values().head(10)
```

```
[54]: title
Trapped           1.0
Carlos            2.0
Blood In, Blood Out    3.0
Heaven's Gate      4.0
The Legend of Suriyothai  5.0
Das Boot          6.0
Apocalypse Now    7.0
The Company        8.0
Gods and Generals  9.0
Gettysburg         10.0
Name: duration, dtype: float64
```

```
[55]: movie['duration'].sort_values(ascending=False).head(10)
```

```
[55]: title
Trapped           511.0
Carlos            334.0
Blood In, Blood Out    330.0
Heaven's Gate      325.0
The Legend of Suriyothai  300.0
Das Boot          293.0
Apocalypse Now    289.0
The Company        286.0
Gods and Generals  280.0
Gettysburg         271.0
Name: duration, dtype: float64
```

Exercise 6

Select actor1 as a Series and sort it from least to greatest, but have missing values show up first. Output the first 10 values.

```
[56]: movie['actor1'].sort_values(na_position='first').head(10)
```

```
[56]: title
Pink Ribbons, Inc.           NaN
Sex with Strangers          NaN
The Harvest/La Cosecha       NaN
Ayurveda: Art of Being      NaN
The Brain That Sings         NaN
The Blood of My Brother      NaN
Counting                     NaN
Get Rich or Die Tryin'       50 Cent
The Good Dinosaur            A.J. Buckley
```

```
Queen of the Damned          Aaliyah
Name: actor1, dtype: object
```

3.4 4. Series Methods More

```
[57]: emp = pd.read_csv('../data/employee.csv')
emp.head()
```

	dept	title	hire_date	salary	sex	race
0	Police	POLICE SERGEANT	2001-12-03	87545.38	Male	White
1	Other	ASSISTANT CITY ATTORNEY II	2010-11-15	82182.00	Male	Hispanic
2	Houston Public Works	SENIOR SLUDGE PROCESSOR	2006-01-09	49275.00	Male	Black
3	Police	SENIOR POLICE OFFICER	1997-05-27	75942.10	Male	Hispanic
4	Police	SENIOR POLICE OFFICER	2006-01-23	69355.26	Male	White

Exercise 1

Find the minimum, maximum, mean, median, and standard deviation of the salary column. Return the result as a Series.

```
[58]: emp['salary'].agg(['min', 'max', 'mean', 'median', 'std'])
```

```
[58]: min      9912.000000
max     342784.000000
mean    58206.761571
median   56956.640000
std     23322.315285
Name: salary, dtype: float64
```

Exercise 2

Use the `idxmax` and `idxmin` methods to find the index of where the maximum and minimum salaries are located in the DataFrame. Then use the `loc` indexer to select both of those rows as a DataFrame.

```
[59]: imax = emp['salary'].idxmax()
imin = emp['salary'].idxmin()
rows = [imax, imin]
rows
```

```
[59]: [1732, 1183]
```

```
[60]: emp.loc[rows, :]
```

	dept	title	hire_date	salary	sex	race
1732	Fire	PHYSICIAN,MD	2014-09-27	342784.0	Male	White
1183	Library	CUSTOMER SERVICE CLERK	2016-01-19	9912.0	Female	Hispanic

Exercise 3

Repeat exercise 3, but do so on the `imdb_score` column from the movie dataset.

```
[61]: score = movie['imdb_score']
rows = [score.idxmax(), score.idxmin()]
rows
```

```
[61]: ['Towering Inferno', 'Justin Bieber: Never Say Never']
```

```
[62]: movie.loc[rows, :]
```

	year	color	content_rating	duration	director_name	...	plot_keywords	language	country	budget	imdb_score
title											
Towering Inferno	NaN	Color	NaN	65.0	John Blanchard	...	NaN	English	Canada	NaN	9.5
Justin Bieber: Never Say Never	2011.0	Color	G	115.0	Jon M. Chu	...	boyhood friend manager plasma tv prodigy star	English	USA	13000000.0	1.6

Exercise 4

The `idxmax` and `idxmin` methods are aggregations as they return a single value. Use the `agg` method to return the min/max `imdb_score` and the label for each score.

```
[63]: score = movie['imdb_score']
score.agg(['min', 'idxmin', 'max', 'idxmax'])
```

```
[63]: min           1.6
idxmin      Justin Bieber: Never Say Never
max           9.5
idxmax        Towering Inferno
Name: imdb_score, dtype: object
```

Exercise 5

Read in 20 years of Microsoft stock data, setting the ‘timestamp’ column as the index. Find the top 5 largest one-day percentage gains in the `adjusted_close`.

```
[64]: msft = pd.read_csv('../data/stocks/msft20.csv')
msft.head(3)
```

	date	open	high	low	close	adjusted_close	volume	dividend_amount
0	1999-10-19	88.250	89.250	85.25	86.313	27.8594	69945600	0.0
1	1999-10-20	91.563	92.375	90.25	92.250	29.7758	88090600	0.0
2	1999-10-21	90.563	93.125	90.50	93.063	30.0381	60801200	0.0

```
[65]: ac = msft['adjusted_close']
ac.pct_change().nlargest()
```

```
[65]: 254      0.195654
2259     0.186043
2288     0.122646
639      0.111181
305      0.105183
Name: adjusted_close, dtype: float64
```

Exercise 6

Randomly sample the `actor1` column as a Series with replacement to select three values. Use random state 12345. Setting a random state ensures that the same random sample is chosen regardless of which machine or version of numpy is being used.

```
[66]: movie['actor1'].sample(n=3, random_state=12345)
```

```
[66]: title
Windtalkers           Nicolas Cage
Mission: Impossible   Tom Cruise
Saw 3D: The Final Chapter   Costas Mandylor
Name: actor1, dtype: object
```

Exercise 7

Select the title column from the employee dataset as a Series. Replace the all occurrences of ‘POLICE OFFICER’ and ‘SENIOR POLICE OFFICER’ with ‘POLICE’. You can use a list as the first argument passed to the `replace` method.

```
[67]: emp['title'].head()
```

```
[67]: 0          POLICE SERGEANT
1      ASSISTANT CITY ATTORNEY II
2      SENIOR SLUDGE PROCESSOR
3      SENIOR POLICE OFFICER
4      SENIOR POLICE OFFICER
Name: title, dtype: object
```

```
[68]: vals = ['POLICE OFFICER', 'SENIOR POLICE OFFICER']
emp['title'].replace(vals, 'POLICE').head()
```

```
[68]: 0          POLICE SERGEANT
1      ASSISTANT CITY ATTORNEY II
2      SENIOR SLUDGE PROCESSOR
3          POLICE
4          POLICE
Name: title, dtype: object
```

3.5 5. String Series Methods

```
[69]: import pandas as pd
movie = pd.read_csv('../data/movie.csv', index_col='title')
actor1 = movie['actor1'].dropna()
actor1.head(3)
```

```
[69]: title
Avatar                               CCH Pounder
Pirates of the Caribbean: At World's End    Johnny Depp
Spectre                                Christoph Waltz
Name: actor1, dtype: object
```

Exercise 1

Which actor 1 has appeared in the most movies? Can you write an expression that returns this actors name as a string?

```
[70]: actor1.value_counts().head()
```

```
[70]: Robert De Niro      48
Johnny Depp            36
Nicolas Cage           32
Denzel Washington     29
J.K. Simmons           29
Name: actor1, dtype: int64
```

```
[71]: actor1.value_counts().index[0]
```

```
[71]: 'Robert De Niro'
```

Exercise 2

What percent of movies have the top 100 most frequent actor 1's appeared in?

```
[72]: actor1.value_counts(normalize=True).iloc[:100].sum()
```

```
[72]: 0.325117131798737
```

Exercise 3

How many actor 1's have appeared in exactly one movie?

```
[73]: (actor1.value_counts() == 1).sum()
```

```
[73]: 1379
```

Exercise 4

How many actor 1's have more than 3 e's in their name? Output a unique array of just these actor names so we can manually verify them.

```
[74]: filt = actor1.str.count('e') > 3
unique_e = actor1[filt].unique()
unique_e
```

```
[74]: array(['Jennifer Lawrence', 'Keanu Reeves', 'Seychelle Gabriel',
       'Jeremy Renner', 'Amber Stevens West', 'Peter Greene',
       'Steven Anthony Lawrence', 'Cedric the Entertainer',
       'Sean Pertwee', 'Xander Berkeley', 'Kathleen Freeman',
       'Pierre Perrier', 'Catherine Deneuve', 'George Kennedy',
       'Leighton Meester', 'Steve Guttenberg', 'Emmanuelle Seigner',
       'Jurnee Smollett-Bell', 'Steve Oedekerk',
       'Johannes Silberschneider', 'Bernadette Peters',
       'Jacqueline McKenzie', 'Dee Bradley Baker', 'Jennifer Freeman',
       'Gene Tierney', 'Roscoe Lee Browne', 'Phoebe Legere',
       'Eric Sheffer Stevens', 'Michael Greyeyes', 'Steven Weber',
       'George Newbern', 'Florence Henderson', 'Michelle Simone Miller',
       'Chemeeka Walker', 'Fereshteh Sadre Orafaiy'], dtype=object)
```

```
[75]: len(unique_e)
```

```
[75]: 35
```

To be more exact and account for both upper and lowercase letters, use the regular expression, [eE].

```
[76]: filt = actor1.str.count('[eE]') > 3
unique_eE = actor1[filt].unique()
unique_eE
```

```
[76]: array(['Jennifer Lawrence', 'Keanu Reeves', 'Seychelle Gabriel',
       'Jeremy Renner', 'Amber Stevens West', 'Rupert Everett',
       'Peter Greene', 'Eileen Brennan', 'Steven Anthony Lawrence',
       'Cedric the Entertainer', 'Sean Pertwee', 'Xander Berkeley',
       'Eddie Redmayne', 'Jennifer Ehle', 'Kathleen Freeman',
       'Alden Ehrenreich', 'Pierre Perrier', 'Catherine Deneuve',
       'George Kennedy', 'Leighton Meester', 'Steve Guttenberg',
       'Ernie Reyes Jr.', 'Emmanuelle Vaugier', 'Emmanuelle Seigner',
       'Jurnee Smollett-Bell', 'Steve Oedekerk',
       'Johannes Silberschneider', 'Bernadette Peters',
       'Jacqueline McKenzie', 'Dee Bradley Baker', 'Jennifer Freeman',
       'Eugenio Derbez', 'Gene Tierney', 'Roscoe Lee Browne',
       'Ed Speleers', 'Joe Estevez', 'Phoebe Legere',
       'Eric Sheffer Stevens', 'Michael Greyeyes', 'Steven Weber',
       'George Newbern', 'Florence Henderson', 'Michelle Simone Miller',
       'Chemeeka Walker', 'Fereshteh Sadre Orafaiy'], dtype=object)
```

```
[77]: len(unique_eE)
```

```
[77]: 45
```

Exercise 5

Get a unique list of all actors that have the name ‘Johnson’ as part of their name.

```
[78]: filt = actor1.str.contains('Johnson')
actor1[filt].unique()
```

```
[78]: array(['Don Johnson', 'Dwayne Johnson', 'Richard Johnson', 'Eric Johnson',
       'Bill Johnson', 'Nicole Randall Johnson', 'R. Brandon Johnson'],
       dtype=object)
```

Exercise 6

How many unique actor 1 names end in ‘x’?

```
[79]: filt = actor1.str.endswith('x')
act_endx = actor1[filt].unique()
act_endx
```

```
[79]: array(['Vivica A. Fox', 'Ronny Cox', 'Tony Cox', 'James Wilcox',
       'Justin Theroux', 'Emilia Fox', 'Harry Lennix', 'Nicki Aycox',
       'Elaine Hendrix', 'Kirk Fox', 'Kevin Grevioux', 'Bernard Fox',
       'Lisa Wilcox', 'Rachel Hendrix', 'Redd Foxx'], dtype=object)
```

```
[80]: len(act_endx)
```

```
[80]: 15
```

Exercise 7

The pandas string methods overlap with the builtin Python string methods. Find all the public method names that are in-common to both. Then find the public methods that are unique to each.

```
[81]: python_str = {method for method in dir(str) if method[0] != '_'}
pandas_str = {method for method in dir(actor1.str) if method[0] != '_'}
```

```
[82]: python_str & pandas_str
```

```
[82]: {'capitalize',
       'casefold',
       'center',
       'count',
       'encode',
       'endswith',
       'find',
       'index',
       'isalnum',
       'isalpha',
       'isdecimal',
       'isdigit',
       'islower',
       'isnumeric',
       'isspace',
       'istitle',
       'isupper',
       'join',
```

```
'ljust',
'lower',
'lstrip',
'partition',
'replace',
'rfind',
'rindex',
'rjust',
'rpartition',
'rsplit',
'rstrip',
'split',
'startswith',
'strip',
'swapcase',
'title',
'translate',
'upper',
'zfill'}
```

```
[83]: python_str - pandas_str
```

```
[83]: {'expandtabs',
'format',
'format_map',
'isascii',
'isidentifier',
'isprintable',
'maketrans',
'splitlines'}
```

```
[84]: pandas_str - python_str
```

```
[84]: {'cat',
'contains',
'decode',
'extract',
'extractall',
'findall',
'get',
'get_dummies',
'len',
'match',
'normalize',
'pad',
'repeat',
'slice',
'slice_replace',
'wrap'}
```

3.6 6. Datetime Series Methods

```
[85]: bikes = pd.read_csv('../data/bikes.csv', parse_dates=['starttime', 'stoptime'])
start = bikes['starttime']
```

Exercise 1

What percentage of bike rides happen in January?

```
[86]: # 2.7%
start.dt.month_name().value_counts(normalize=True).round(3)
```

```
[86]: August      0.137
July        0.132
September   0.130
June         0.121
October     0.111
May          0.086
November    0.071
April        0.064
December    0.045
March        0.044
February    0.030
January      0.027
Name: starttime, dtype: float64
```

Exercise 2

What percentage of bike rides happen on the weekend?

```
[87]: start.dt.weekday.isin([5, 6]).mean()
```

```
[87]: 0.19692946555131866
```

Or like this:

```
[88]: (start.dt.weekday > 4).mean()
```

```
[88]: 0.19692946555131866
```

Exercise 3

What percentage of bike rides happen on the last day of the month?

```
[89]: start.dt.is_month_end.mean()
```

```
[89]: 0.031563816406795904
```

Exercise 4

We would expect that the value of the minutes recorded for each starting ride is approximately random. Can you show some data that confirms or rejects this?

The distribution looks quite uniform:

```
[90]: start.dt.minute.value_counts(normalize=True)
```

```
[90]: 12    0.017968
 6    0.017928
 8    0.017868
18    0.017808
43    0.017629
21    0.017549
10    0.017529
48    0.017509
44    0.017449
15    0.017409
53    0.017349
17    0.017329
37    0.017309
13    0.017289
19    0.017269
33    0.017229
42    0.017229
39    0.017189
24    0.017189
22    0.017110
34    0.017070
29    0.017070
45    0.016950
 5    0.016890
36    0.016870
11    0.016870
14    0.016870
49    0.016850
47    0.016830
30    0.016810
16    0.016710
32    0.016670
38    0.016630
 1    0.016630
40    0.016531
 7    0.016491
 2    0.016471
46    0.016471
 4    0.016391
23    0.016331
54    0.016311
57    0.016291
 3    0.016251
28    0.016091
35    0.016071
59    0.016071
56    0.016031
 0    0.015932
```

```

58    0.015912
50    0.015872
31    0.015872
55    0.015852
9     0.015812
27    0.015812
41    0.015712
20    0.015612
25    0.015512
52    0.015253
51    0.015213
26    0.014973
Name: starttime, dtype: float64

```

Exercise 5

Assign the length of the ride to `ride_length`. Then find the percentage of rides that lasted longer than 30 minutes.

```
[91]: stop = bikes['stoptime']
ride_length = stop - start
```

```
[92]: (ride_length.dt.total_seconds() > (60 * 30)).mean()
```

```
[92]: 0.019625067380063487
```

3.7 Project - Testing Normality of Stock Market Returns

Execute the cells below to read in 20 years of Apple (AAPL) data as a Series and answer the exercises below with it.

```
[93]: stocks = pd.read_csv('../data/stocks/stocks10.csv', index_col='date',
                         parse_dates=['date'])
stocks.head(3)
```

	MSFT	AAPL	SLB	AMZN	TSLA	XOM	WMT	T	FB	V
date										
1999-10-25	29.84	2.32	17.02	82.75	NaN	21.45	38.99	16.78	NaN	NaN
1999-10-26	29.82	2.34	16.65	81.25	NaN	20.89	37.11	17.28	NaN	NaN
1999-10-27	29.33	2.38	16.52	75.94	NaN	20.80	36.94	18.27	NaN	NaN

```
[94]: aapl = stocks['AAPL']
aapl.head()
```

```
[94]: date
1999-10-25    2.32
1999-10-26    2.34
1999-10-27    2.38
1999-10-28    2.43
```

```
1999-10-29    2.50
Name: AAPL, dtype: float64
```

Exercise 1

Use one line of code to find the daily percentage returns of AAPL and drop any missing values. Save the result to `aapl_change`.

```
[95]: aapl_change = aapl.pct_change().dropna()
aapl_change.head(3)
```

```
[95]: date
1999-10-26    0.008621
1999-10-27    0.017094
1999-10-28    0.021008
Name: AAPL, dtype: float64
```

Exercise 2

Find the mean daily return for Apple, the first and last closing prices, and the number of trading days. Store all four of these values into separate variables.

```
[96]: mean = aapl_change.mean()
first = aapl_change.iloc[0]
last = aapl_change.iloc[-1]
n = aapl_change.size
mean, first, last, n
```

```
[96]: (0.0012692984527780723, 0.008620689655172376, -0.002960769800148033, 5032)
```

Exercise 3

If Apple returned its mean percentage return every single day since the first day you have data, what would its last closing price be? Is it the same as the actual last closing price? You need to use all the variables calculated from Exercise 2.

```
[97]: first * (mean + 1) ** n
```

```
[97]: 5.101167707811244
```

```
[98]: last
```

```
[98]: -0.002960769800148033
```

Exercise 4

Find the z-score for the Apple daily returns. Save this to a variable `z_score_raw`. What is the max and minimum score?

```
[99]: std = aapl_change.std()
z_score_raw = (aapl_change - mean) / std
z_score_raw.head()
```

```
[99]: date
1999-10-26    0.286300
1999-10-27    0.616294
1999-10-28    0.768740
1999-10-29    1.072441
1999-11-01   -1.295674
Name: AAPL, dtype: float64
```

```
[100]: z_score_raw.max(), z_score_raw.min()
```

```
[100]: (5.3712520859138, -20.221564477921486)
```

Or with agg.

```
[101]: z_score_raw.agg(['min', 'max']).round(3)
```

```
[101]: min    -20.222
       max     5.371
Name: AAPL, dtype: float64
```

Exercise 5

What percentage did Tesla stock increase when it had its highest maximum raw z-score?

```
[102]: aapl_change[z_score_raw == z_score_raw.max()]
```

```
[102]: date
2008-10-13    0.139188
Name: AAPL, dtype: float64
```

Exercise 6

Create a function that accepts a Series of stock closing prices. Have it return the percentage of prices within 1, 2, and 3 standard deviations from the mean. Use your function to return results for different stocks found in the `stocks` DataFrame.

```
[103]: def stock_pct_finder(close):
    close_change = close.pct_change().dropna()
    mean = close_change.mean()
    std = close_change.std()
    z_score_abs = (close_change - mean).abs() / std

    pct_within1 = (z_score_abs < 1).mean().round(3)
    pct_within2 = (z_score_abs < 2).mean().round(3)
    pct_within3 = (z_score_abs < 3).mean().round(3)

    return pct_within1, pct_within2, pct_within3
```

```
[104]: stock_pct_finder(stocks['AMZN'])
```

```
[104]: (0.833, 0.95, 0.98)
```

```
[105]: stock_pct_finder(stocks['FB'])
```

[105]: (0.804, 0.956, 0.988)

```
[106]: stock_pct_finder(stocks['SLB'])
```

[106]: (0.763, 0.948, 0.988)

Exercise 7

How many days did Apple close above 100 and below 120?

```
[107]: filt = (aapl > 100) & (aapl < 120)
filt.sum()
```

[107]: 401

Exercise 8

How many days did Apple close below 50 or above 150?

```
[108]: ((aapl < 50) | (aapl > 150)).sum()
```

[108]: 4871

Exercise 9

Look up the definition for interquartile range and slice Apple closing prices so it contains just the interquartile range. There are multiple ways to do this. Check the `quantile` method.

```
[109]: # a few ways to do this
n = aapl.size
first_q = n // 4
third_q = n // 4 * 3

aapl.sort_values().iloc[first_q:third_q].head()
```

```
[109]: date
2004-12-03    3.91
2000-03-06    3.92
2000-03-10    3.92
2004-12-07    3.92
2000-03-30    3.92
Name: AAPL, dtype: float64
```

```
[110]: # using the quantile method
q1 = aapl.quantile(.25)
q3 = aapl.quantile(.75)

criteria = (aapl >= q1) & (aapl <= q3)

aapl[criteria].sort_values().head()
```

[110]: date

```
2004-12-03    3.91
2004-12-20    3.91
2004-12-07    3.92
2000-03-06    3.92
2000-03-10    3.92
Name: AAPL, dtype: float64
```

Exercise 10

Find the date of the highest closing price. Find out how many trading days it has been since Apple recorded its highest closing price.

[111]: max_close_date = aapl.idxmax()
max_close_date

[111]: Timestamp('2019-10-23 00:00:00')

Number of trading days

[112]: aapl.loc[max_close_date:].size

[112]: 2

Part IV

Essential DataFrame Commands

Chapter 4

Solutions

```
[1]: import pandas as pd  
college = pd.read_csv('../data/college.csv', index_col='instnm')
```

4.1 1. DataFrame Attributes and Methods

Exercise 1

Select only the float columns from the `college` DataFrame. How many are there?

```
[2]: college.select_dtypes('float').shape
```

```
[2]: (7535, 20)
```

There are 20 float columns

Exercise 2

When you call the `info` method on a DataFrame, one of the very last items that gets outputted is the count of columns for each data type. Can you think of a different combination of pandas operations that would return this as a Series.

```
[3]: college.dtypes.value_counts()
```

```
[3]: float64    20  
object       4  
int64       2  
dtype: int64
```

4.2 2. DataFrame Statistical Methods

```
[4]: import pandas as pd  
movie = pd.read_csv('../data/movie.csv', index_col='title')  
cols = ['actor1_fb', 'actor2_fb', 'actor3_fb']  
actor_fb = movie[cols]  
actor_fb.head(3)
```

	actor1_fb	actor2_fb	actor3_fb
title			
Avatar	1000.0	936.0	855.0
Pirates of the Caribbean: At World's End	40000.0	5000.0	1000.0
Spectre	11000.0	393.0	161.0

Exercise 1

Calculate the mean of each actor Facebook like column. Which actor (1, 2, or 3) has the highest mean?
actor 1 has the highest mean.

```
[5]: actor_fb.mean()
```

```
[5]: actor1_fb      6494.488491
actor2_fb      1621.923516
actor3_fb      631.276313
dtype: float64
```

Exercise 2

The result of exercise 1 is a Series of three values. Can you call a method on this Series to choose the column name with the highest mean Facebook likes.

```
[6]: actor_fb.mean().idxmax()
```

```
[6]: 'actor1_fb'
```

Exercise 3

Calculate the total Facebook likes of all three actors for each movie

```
[7]: actor_fb.sum(axis=1).head()
```

```
[7]: title
Avatar                  2791.0
Pirates of the Caribbean: At World's End  46000.0
Spectre                  11554.0
The Dark Knight Rises    73000.0
Star Wars: Episode VII - The Force Awakens 143.0
dtype: float64
```

Exercise 4

What percentage of movies have more than 10,000 total actor FB likes?

About 30%

```
[8]: (actor_fb.sum(axis='columns') > 10000).mean()
```

```
[8]: 0.2982099267697315
```

Exercise 5

Find the median gross revenue in millions of dollars for the movies that have more than 10,000 total actor FB likes. Do the same for movies with 10,000 or less total actor FB likes.

```
[9]: filt = actor_fb.sum(axis='columns') > 10000
movie.loc[filt, 'gross'].median() / 1e6
```

[9]: 42.3919155

```
[10]: movie.loc[~filt, 'gross'].median() / 1e6
```

[10]: 16.8157525

Exercise 6

From exercise 5, it appears that movies with more than 10,000 total actor FB likes gross 2.5 times as much. This may be due to the fact that newer movies have more actors that are recognized by FB users. Find the median year produced for both groups.

```
[11]: movie.loc[filt, 'year'].median()
```

[11]: 2006.0

```
[12]: movie.loc[~filt, 'year'].median()
```

[12]: 2005.0

Exercise 7

For each movie made in the year 2016, what is the median of the total actor FB likes?

```
[13]: filt = movie['year'] == 2016
actor_fb[filt].sum(axis=1).median()
```

[13]: 3571.5

If the above is too complex, here it is one step at a time.

```
[14]: actor_fb_2016 = actor_fb[filt]
actor_fb_2016.head()
```

	actor1_fb	actor2_fb	actor3_fb
title			
Batman v Superman: Dawn of Justice	15000.0	4000.0	2000.0
Captain America: Civil War	21000.0	19000.0	11000.0
Star Trek Beyond	998.0	119.0	105.0
The Legend of Tarzan	11000.0	10000.0	103.0
X-Men: Apocalypse	34000.0	13000.0	1000.0

```
[15]: actor_fb_2016_total = actor_fb_2016.sum(axis=1)
actor_fb_2016_total.head()
```

```
[15]: title
Batman v Superman: Dawn of Justice      21000.0
Captain America: Civil War              51000.0
Star Trek Beyond                         1222.0
The Legend of Tarzan                   21103.0
X-Men: Apocalypse                      48000.0
dtype: float64
```

```
[16]: actor_fb_2016_total.median()
```

```
[16]: 3571.5
```

Exercise 8

Write a function that has a single parameter, `year`. Have it return the median of the total actor FB likes for the given year. Test your function with the year 2016 and verify the result with Exercise 6.

```
[17]: def median_fb_likes(year):
    filt = movie['year'] == year
    return actor_fb.loc[filt].sum(axis='columns').median()
```

```
[18]: median_fb_likes(2016)
```

```
[18]: 3571.5
```

Exercise 9

Write a loop to print out the year and median total actor FB likes for that year from 1990 to 2016

```
[19]: for year in range(1990, 2017):
    print(year, median_fb_likes(year))
```

```
1990 2017.0
1991 2436.0
1992 2147.5
1993 2018.0
1994 2368.5
1995 2612.0
1996 2692.5
1997 1964.0
1998 2482.0
1999 2595.0
2000 2378.0
2001 2424.0
2002 2146.0
2003 2019.0
2004 2298.0
2005 2072.0
2006 2359.0
```

```
2007 2002.5  
2008 2400.0  
2009 2145.0  
2010 2411.0  
2011 2818.5  
2012 2426.0  
2013 2420.0  
2014 2084.0  
2015 2063.0  
2016 3571.5
```

Use the college dataset with the institution name as the index for the remaining exercises.

```
[20]: college = pd.read_csv('../data/college.csv', index_col='instnm')
```

Exercise 10

Using the **college** dataset, find the number of non-missing values in each column and again for each row.

non-missing for each column

```
[21]: college.count()
```

```
[21]: city                7535  
stabbr               7535  
hbcu                 7164  
menonly              7164  
womenonly             7164  
relafafil            7535  
satvrmid             1185  
satmtmid              1196  
distanceonly          7164  
ugds                 6874  
ugds_white            6874  
ugds_black            6874  
ugds_hisp              6874  
ugds_asian             6874  
ugds_aian              6874  
ugds_nhpi              6874  
ugds_2mor              6874  
ugds_nra              6874  
ugds_unkn              6874  
pptug_ef              6853  
curroper              7535  
pctpell                6849  
pctfloan                6849  
ug25abv                  6718  
md_earn_wne_p10        6413  
grad_debt_mdn_supp      7503  
dtype: int64
```

non-missing count for the rows

```
[22]: college.count(axis='columns').head()
```

```
[22]: instnm
Alabama A & M University          26
University of Alabama at Birmingham 26
Amridge University                  24
University of Alabama in Huntsville 26
Alabama State University            26
dtype: int64
```

Exercise 11

What is the average number of non-missing values for each row?

```
[23]: college.count(axis=1).mean()
```

```
[23]: 22.70763105507631
```

Exercise 12

The UGDS column of the college dataset contains the total undergraduate population. What is the least number of colleges it would take to have have a total of more than 5 million students.

```
[24]: ugds_cumsum = college['ugds'].sort_values(ascending=False).cumsum()
ugds_cumsum.head(10)
```

```
[24]: instnm
University of Phoenix-Arizona          151558.0
Ivy Tech Community College             229215.0
Miami Dade College                    290685.0
Lone Star College System              350605.0
Houston Community College              408689.0
University of Central Florida          460969.0
Liberty University                     510309.0
Texas A & M University-College Station 557250.0
American Public University System       602174.0
Ashford University                     646918.0
Name: ugds, dtype: float64
```

```
[25]: (ugds_cumsum < 5000000).sum() + 1
```

```
[25]: 185
```

It takes the top 185 colleges (by population) to total more than 5 million students. Let's verify the results:

```
[26]: ugds_sort = college['ugds'].sort_values(ascending=False)
```

```
[27]: ugds_sort.iloc[:184].sum()
```

```
[27]: 4989478.0
```

```
[28]: ugds_sort.iloc[:185].sum()
```

```
[28]: 5007289.0
```

Exercise 13

Call the `describe` method, but make it work only for the string columns.

```
[29]: college.describe(include='object')
```

	city	stabbr	md_earn_wne_p10	grad_debt_mdn_supp
count	7535	7535	6413	7503
unique	2514	59	598	2038
top	New York	CA	PrivacySuppressed	PrivacySuppressed
freq	87	773	822	1510

Exercise 14

Call the `max` method, but only return columns that are numeric.

```
[30]: college.max(numeric_only=True)
```

```
[30]: hbcu                  1.0000
menonly                1.0000
womenonly               1.0000
relaffil                1.0000
satvrmid                765.0000
satmtmid                785.0000
distanceonly             1.0000
ugds                   151558.0000
ugds_white              1.0000
ugds_black              1.0000
ugds_hisp                1.0000
ugds_asian               0.9727
ugds_aian                1.0000
ugds_nhpi                0.9983
ugds_2mor                0.5333
ugds_nra                 0.9286
ugds_unkn                0.9027
pptug_ef                 1.0000
curroper                 1.0000
pctpell                  1.0000
pctfloan                 1.0000
ug25abv                  1.0000
dtype: float64
```

4.3 3. DataFrame Missing Value Methods

```
[31]: import pandas as pd
college = pd.read_csv('../data/college.csv', index_col='instnm')
```

Exercise 1

Find the number of missing values for each row.

```
[32]: college.isna().sum(axis='columns').head(10)
```

```
[32]: instnm
Alabama A & M University          0
University of Alabama at Birmingham 0
Amridge University                  2
University of Alabama in Huntsville 0
Alabama State University           0
The University of Alabama          0
Central Alabama Community College  2
Athens State University           2
Auburn University at Montgomery    0
Auburn University                  0
dtype: int64
```

Exercise 2

What percentage of rows have more than 5 missing values?

```
[33]: (college.isna().sum(axis='columns') > 5).mean()
```

```
[33]: 0.09011280690112806
```

Exercise 3

How many total missing values are there in the entire DataFrame?

```
[34]: college.isna().sum().sum()
```

```
[34]: 24808
```

Exercise 4

How many total non-missing values are there in the entire DataFrame?

```
[35]: college.count().sum()
```

```
[35]: 171102
```

Exercise 5

How many rows will be dropped when the `dropna` method is called with its defaults. Calculate this number without calling the `dropna` method.

```
[36]: # calculate the number of rows with at least one missing value
college.isna().sum(axis=1) > 0).sum()
```

[36]: 6364

Exercise 6

Verify the result from exercise 5 by calling the `dropna` method.

```
[37]: len(college) - len(college.dropna())
```

[37]: 6364

Exercise 7

Drop all the rows that are missing the `ugds` column.

```
[38]: college.dropna(subset=['ugds']).head(3)
```

instnm	city	stabbr	hbcu	menonly	womenonly	...	pctpell	pctfloan	ug25abv	md_earn_wne_p10	grad_debt_mdn_supp
Alabama A & M University	Normal	AL	1.0	0.0	0.0	...	0.7356	0.8284	0.1049	30300	33888
University of Alabama at Birmingham	Birmingham	AL	0.0	0.0	0.0	...	0.3460	0.5214	0.2422	39700	21941.5
Amridge University	Montgomery	AL	0.0	0.0	0.0	...	0.6801	0.7795	0.8540	40100	23370

Exercise 8

Drop all columns that have more than 5% of their values missing.

```
[39]: thresh = int(.95 * len(college))
thresh
```

[39]: 7158

```
[40]: college.dropna(axis=1, thresh=thresh).shape
```

[40]: (7535, 9)

17 columns were dropped.

```
[41]: college.shape
```

[41]: (7535, 26)

Exercise 9

Fill in the missing values with the maximum value of each column.

```
[42]: max_vals = college.max()
max_vals.head()
```

```
[42]: city           Zanesville
stabbr          WY
hbcu            1
menonly         1
womenonly       1
dtype: object
```

```
[43]: college.fillna(max_vals).head(3)
```

instnm	city	stabbr	hbcu	menonly	womenonly	...	pctpell	pctfloan	ug25abv	md_earn_wne_p10	grad_debt_mdn_supp
Alabama A & M University	Normal	AL	1.0	0.0	0.0	...	0.7356	0.8284	0.1049	30300	33888
University of Alabama at Birmingham	Birmingham	AL	0.0	0.0	0.0	...	0.3460	0.5214	0.2422	39700	21941.5
Amridge University	Montgomery	AL	0.0	0.0	0.0	...	0.6801	0.7795	0.8540	40100	23370

4.4 4. DataFrame Sorting, Ranking, and Uniqueness

```
[44]: emp = pd.read_csv('../data/employee.csv')
```

Exercise 1

Sort department, race, sex ascending along with salary descending.

```
[45]: emp.sort_values(['dept', 'race', 'sex', 'salary'], ascending=[True, True, True, False]).head()
```

	dept		title	hire_date	salary	sex	race
19946	Fire		PHYSICIAN,MD	2015-06-22	342784.0	Female	Asian
514	Fire	ASSISTANT EMS PHYSICIAN DIRECTOR		2017-08-28	141669.0	Female	Asian
8642	Fire	STAFF PSYCHOLOGIST		2014-12-22	103805.0	Female	Asian
7199	Fire	SENIOR STAFF ANALYST		2003-01-14	97850.0	Female	Asian
430	Fire	ADMINISTRATIVE ASSISTANT		1985-08-19	55432.0	Female	Asian

Exercise 2

How many unique combinations of department and title exist?

```
[46]: len(emp.drop_duplicates(subset=['dept', 'title']))
```

```
[46]: 1312
```

Exercise 3

Since only Series methods have a `unique` method, can you think of a creative way of getting the same result as exercise 2 with the `unique` method?

```
[47]: # concatenate the two columns together to create a Series
dept_title = emp['dept'] + emp['title']
dept_title.head()
```

```
[47]: 0          PolicePOLICE SERGEANT
      1          OtherASSISTANT CITY ATTORNEY II
      2  Houston Public WorksSENIOR SLUDGE PROCESSOR
      3          PoliceSENIOR POLICE OFFICER
      4          PoliceSENIOR POLICE OFFICER
      dtype: object
```

```
[48]: # Now, use the `unique` method.
len(dept_title.unique())
```

```
[48]: 1312
```

```
[49]: # in one line
len((emp['dept'] + emp['title']).unique())
```

```
[49]: 1312
```

Exercise 4

Find the occurrence of all race and sex combinations. For instance, you would return an object that contains the number of ‘Hispanic Males’, ‘Black Females’, etc...

```
[50]: race_sex = emp['race'] + ' - ' + emp['sex']
race_sex.head()
```

```
[50]: 0      White - Male
      1      Hispanic - Male
      2      Black - Male
      3      Hispanic - Male
      4      White - Male
      dtype: object
```

```
[51]: race_sex.value_counts()
```

White - Male	6488
Black - Male	5074
Hispanic - Male	4208
Black - Female	3587
Hispanic - Female	1940
White - Female	1291
Asian - Male	1059
Asian - Female	488
Native American - Male	107
Native American - Female	39

```
dtype: int64
```

```
[52]: # in one line
# normalized to get relative frequency
(emp['race'] + ' - ' + emp['sex']).value_counts(normalize=True).round(3)
```

```
[52]: White - Male          0.267
Black - Male             0.209
Hispanic - Male          0.173
Black - Female            0.148
Hispanic - Female         0.080
White - Female            0.053
Asian - Male              0.044
Asian - Female             0.020
Native American - Male    0.004
Native American - Female   0.002
dtype: float64
```

Exercise 5

Read in the college dataset setting the institution name (`instnm`) as the index. Select the columns `stabbr`, `satvrmid`, `satmtmid` and `ugds` columns for the state of Texas ('TX') that have an undergraduate student population of more than 20,000. Drop any rows with missing values and assign the result to the variable name `college_tx`.

```
[53]: college = pd.read_csv('../data/college.csv', index_col='instnm')
cols = ['stabbr', 'satvrmid', 'satmtmid', 'ugds']
college_tx = college[cols].query('stabbr == "TX" and ugds > 20000').dropna()
college_tx
```

	stabbr	satvrmid	satmtmid	ugds
instnm				
University of Houston	TX	555.0	590.0	31643.0
University of North Texas	TX	545.0	555.0	29758.0
Texas State University	TX	510.0	515.0	32177.0
Texas A & M University-College Station	TX	580.0	615.0	46941.0
The University of Texas at Arlington	TX	494.0	550.0	29616.0
The University of Texas at Austin	TX	630.0	660.0	38914.0
The University of Texas at San Antonio	TX	505.0	535.0	23815.0
Texas Tech University	TX	540.0	560.0	28278.0

Exercise 6

Rank each column from the `college_tx` DataFrame from greatest to least.

```
[54]: college_tx.rank(ascending=False)
```

	stabbr	satvrmid	satmtmid	ugds
instnm				
University of Houston	4.5	3.0	3.0	4.0
University of North Texas	4.5	4.0	5.0	5.0
Texas State University	4.5	6.0	8.0	3.0
Texas A & M University-College Station	4.5	2.0	2.0	1.0
The University of Texas at Arlington	4.5	8.0	6.0	6.0
The University of Texas at Austin	4.5	1.0	1.0	2.0
The University of Texas at San Antonio	4.5	7.0	7.0	8.0
Texas Tech University	4.5	5.0	4.0	7.0

Exercise 7

Using the full college dataset, find the largest school by population for each state. Return only the `stabbr` and `ugds` columns and have it sorted by `ugds`.

```
[55]: college[['stabbr', 'ugds']].sort_values('ugds', ascending=False) \
    .drop_duplicates(subset='stabbr').head(10)
```

	stabbr	ugds
instnm		
University of Phoenix-Arizona	AZ	151558.0
Ivy Tech Community College	IN	77657.0
Miami Dade College	FL	61470.0
Lone Star College System	TX	59920.0
Liberty University	VA	49340.0
American Public University System	WV	44924.0
Ashford University	CA	44744.0
Western Governors University	UT	44499.0
Ohio State University-Main Campus	OH	43733.0
Kaplan University-Davenport Campus	IA	40335.0

Exercise 8

Several of the columns from the college dataset contain binary data (are either 0 or 1). Can you identify the names of these columns?

Start by finding the number of unique values of each column.

```
[56]: col_unique = college.nunique()
col_unique
```

```
[56]: city          2514
       stabbr        59
       hbcu          2
       menonly        2
       womenonly      2
       relaffil        2
       satvrmid     163
       satmtmid     167
       distanceonly      2
       ugds         2932
       ugds_white    4397
       ugds_black     3242
       ugds_hisp      2809
       ugds_asian     1254
       ugds_aian       601
       ugds_nhpi      363
       ugds_2mor      957
       ugds_nra       920
       ugds_unkn     1517
       pptug_ef      3420
       curroper        2
       pctpell       4422
       pctfloan      4155
       ug25abv       4285
       md_earn_wne_p10  598
       grad_debt_mdn_supp 2038
       dtype: int64
```

Do boolean indexing to select just those that are binary.

```
[57]: col_unique[col_unique == 2]
```

```
[57]: hbcu          2
       menonly        2
       womenonly      2
       relaffil        2
       distanceonly      2
       curroper        2
       dtype: int64
```

Can extract the index to get just the names.

```
[58]: col_unique[col_unique == 2].index
```

```
[58]: Index(['hbcu', 'menonly', 'womenonly', 'relaffil', 'distanceonly', 'curroper'],
       dtype='object')
```

4.5 5. DataFrame Structure Methods

```
[59]: import pandas as pd
cols = ['instnm', 'city', 'stabbr', 'relaffil', 'satvrmid', 'satmtmid', 'ugds']
college_all = pd.read_csv('../data/college.csv', index_col='instnm', usecols=cols)
college_all.head()
```

		city	stabbr	relaffil	satvrmid	satmtmid	ugds
	instnm						
	Alabama A & M University	Normal	AL	0	424.0	420.0	4206.0
	University of Alabama at Birmingham	Birmingham	AL	0	570.0	565.0	11383.0
	Amridge University	Montgomery	AL	1	NaN	NaN	291.0
	University of Alabama in Huntsville	Huntsville	AL	0	595.0	590.0	5451.0
	Alabama State University	Montgomery	AL	0	425.0	430.0	4811.0

Exercise 1

Create a new boolean column in the `college_all` DataFrame named ‘Verbal Higher’ that is True for every college that has a higher verbal than math SAT score. Find the mean of this new column. Why does this number look suspiciously low?

```
[60]: college_all['Verbal Higher'] = college_all['satvrmid'] > college_all['satmtmid']

[61]: college_all['Verbal Higher'].mean()

[61]: 0.048042468480424684
```

One reason it is so low is that there are mostly missing values for the SAT columns and the comparison operators return False when comparing missing values. Notice that 84% of the values are missing for both SAT columns.

```
[62]: cols = ['satvrmid', 'satmtmid']
college_all[cols].isna().mean()

[62]: satvrmid    0.842734
satmtmid    0.841274
dtype: float64
```

Exercise 2

Find the real percentage of schools with higher verbal than math SAT scores.

Drop the rows with missing SAT values first.

```
[63]: cols = ['satvrmid', 'satmtmid']
sat = college_all[cols].dropna()
sat.head()
```

	instnm	satvrmid	satmtmid
Alabama A & M University	424.0	420.0	
University of Alabama at Birmingham	570.0	565.0	
University of Alabama in Huntsville	595.0	590.0	
Alabama State University	425.0	430.0	
The University of Alabama	555.0	565.0	

```
[64]: (sat['satvrmid'] > sat['satmtmid']).mean()
```

[64]: 0.30574324324324326

Can also find all those school with equal scores in both subjects.

```
[65]: (sat['satvrmid'] == sat['satmtmid']).mean()
```

[65]: 0.08699324324324324

Exercise 3

Create a new column called ‘median all’ that has every value set to the median population of all the schools.

```
[66]: college_all['median_all'] = college_all['ugds'].median()
       college_all.head()
```

	city	stabbr	relaffil	satvrmid	satmtmid	ugds	Verbal Higher	median all
institnm								
Alabama A & M University	Normal	AL	0	424.0	420.0	4206.0	True	412.5
University of Alabama at Birmingham	Birmingham	AL	0	570.0	565.0	11383.0	True	412.5
Amridge University	Montgomery	AL	1	NaN	NaN	291.0	False	412.5
University of Alabama in Huntsville	Huntsville	AL	0	595.0	590.0	5451.0	True	412.5
Alabama State University	Montgomery	AL	0	425.0	430.0	4811.0	False	412.5

Exercise 4

Rename the row label ‘Texas A & M University-College Station’ to ‘TAMU’. Reassign the result back to college all and then select this row as a Series.

```
[67]: college_all = college_all.rename(index={'Texas A & M University-College Station':  
    ↪ 'TAMU'})  
college_all.loc['TAMU']
```

```
[67]: city          College Station
stabbr           TX
relaffil          0
satvrmid         580
satmtmid         615
ugds            46941
Verbal Higher    False
median all       412.5
Name: TAMU, dtype: object
```

Execute the following cell to read in the City of Houston employee dataset and use it for the remaining problems.

```
[68]: emp = pd.read_csv('../data/employee.csv')
emp.head()
```

	dept	title	hire_date	salary	sex	race
0	Police	POLICE SERGEANT	2001-12-03	87545.38	Male	White
1	Other	ASSISTANT CITY ATTORNEY II	2010-11-15	82182.00	Male	Hispanic
2	Houston Public Works	SENIOR SLUDGE PROCESSOR	2006-01-09	49275.00	Male	Black
3	Police	SENIOR POLICE OFFICER	1997-05-27	75942.10	Male	Hispanic
4	Police	SENIOR POLICE OFFICER	2006-01-23	69355.26	Male	White

Exercise 5

Create a new column `bonus` right after the `salary` column equal to 10% of the salary. Round the bonus to the nearest thousand.

```
[69]: bonus = (emp['salary'] * .1).round(-3)
emp.insert(4, 'bonus', bonus)
emp.head()
```

	dept	title	hire_date	salary	bonus	sex	race
0	Police	POLICE SERGEANT	2001-12-03	87545.38	9000.0	Male	White
1	Other	ASSISTANT CITY ATTORNEY II	2010-11-15	82182.00	8000.0	Male	Hispanic
2	Houston Public Works	SENIOR SLUDGE PROCESSOR	2006-01-09	49275.00	5000.0	Male	Black
3	Police	SENIOR POLICE OFFICER	1997-05-27	75942.10	8000.0	Male	Hispanic
4	Police	SENIOR POLICE OFFICER	2006-01-23	69355.26	7000.0	Male	White

Exercise 6

Read in the college dataset and set `instnm` as the index and assign it to the variable name `college1`. Use the `copy` method to create a new copy of the `college` DataFrame and assign it to variable `college2`. Select all the non-white race columns (`ugds_black` through `ugds_unkn`). Sum the rows of this DataFrame and assign the result to a variable. Now drop all the non-white race columns from the `college2` DataFrame and assign the result to `college3`.

Use the `insert` method to insert a new column to the right of the `ugds_white` column of the `college3` DataFrame. Name this column `ugds_nonwhite`.

```
[70]: college1 = pd.read_csv('../data/college.csv', index_col='instnm')
college2 = college1.copy()
college2_race = college2.loc[:, 'ugds_black':'ugds_unkn']
college2_race.head()
```

instnm	ugds_black	ugds_hisp	ugds_asian	ugds_aian	ugds_nhpi	ugds_2mor	ugds_nra	ugds_unkn
Alabama A & M University	0.9353	0.0055	0.0019	0.0024	0.0019	0.0000	0.0059	0.0138
University of Alabama at Birmingham	0.2600	0.0283	0.0518	0.0022	0.0007	0.0368	0.0179	0.0100
Amridge University	0.4192	0.0069	0.0034	0.0000	0.0000	0.0000	0.0000	0.2715
University of Alabama in Huntsville	0.1255	0.0382	0.0376	0.0143	0.0002	0.0172	0.0332	0.0350
Alabama State University	0.9208	0.0121	0.0019	0.0010	0.0006	0.0098	0.0243	0.0137

```
[71]: non_white = college2_race.sum(axis='columns')
non_white.head()
```

```
[71]: instnm
Alabama A & M University          0.9667
University of Alabama at Birmingham 0.4077
Amridge University                  0.7010
University of Alabama in Huntsville 0.3012
Alabama State University            0.9842
dtype: float64
```

```
[72]: college3 = college2.drop(columns=college2_race.columns)
college3.head(2)
```

instnm	city	stabbr	hbcu	menonly	womenonly	...	pctpell	pctfloan	ug25abv	md_earn_wne_p10	grad_debt_mdn_supp
Alabama A & M University	Normal	AL	1.0	0.0	0.0	...	0.7356	0.8284	0.1049	30300	33888
University of Alabama at Birmingham	Birmingham	AL	0.0	0.0	0.0	...	0.3460	0.5214	0.2422	39700	21941.5

```
[73]: college3.insert(11, 'ugds_nonwhite', non_white)
```

[74]: college3.head(2)

instnm	city	stabbr	hbcu	menonly	womenonly	...	pctpell	pctfloan	ug25abv	md_earn_wne_p10	grad_debt_mdn_supp
Alabama A & M University	Normal	AL	1.0	0.0	0.0	...	0.7356	0.8284	0.1049	30300	33888
University of Alabama at Birmingham	Birmingham	AL	0.0	0.0	0.0	...	0.3460	0.5214	0.2422	39700	21941.5

[75]: # all in one cell

```
college1 = pd.read_csv('../data/college.csv', index_col='instnm')
college2 = college1.copy()
college2_race = college2.loc[:, 'ugds_black':'ugds_unkn']
non_white = college2_race.sum(axis='columns')
college3 = college2.drop(columns=college2_race.columns)
college3.insert(11, 'ugds_nonwhite', non_white)
college3.head(2)
```

instnm	city	stabbr	hbcu	menonly	womenonly	...	pctpell	pctfloan	ug25abv	md_earn_wne_p10	grad_debt_mdn_supp
Alabama A & M University	Normal	AL	1.0	0.0	0.0	...	0.7356	0.8284	0.1049	30300	33888
University of Alabama at Birmingham	Birmingham	AL	0.0	0.0	0.0	...	0.3460	0.5214	0.2422	39700	21941.5

4.6 6. DataFrame Methods More

[76]: import pandas as pd
emp = pd.read_csv('../data/employee.csv')
emp.head(3)

	dept		title	hire_date	salary	sex	race
0	Police	POLICE SERGEANT	2001-12-03	87545.38	Male	White	
1	Other	ASSISTANT CITY ATTORNEY II	2010-11-15	82182.00	Male	Hispanic	
2	Houston Public Works	SENIOR SLUDGE PROCESSOR	2006-01-09	49275.00	Male	Black	

Exercise 1

Find the relative frequency of departments for all employees and then find the relative frequency of departments for the top 100 salaries. Compare the differences.

[77]: dept_freq = emp['dept'].value_counts(normalize=True)
dept_freq.round(2)

```
[77]: Police          0.31
      Fire           0.18
      Houston Public Works 0.17
      Other          0.14
      Health & Human Services 0.06
      Houston Airport System 0.05
      Parks & Recreation   0.05
      Library          0.02
      Solid Waste Management 0.02
Name: dept, dtype: float64
```

```
[78]: emp_top100 = emp.nlargest(100, 'salary')
emp_top100.head()
```

	dept	title	hire_date	salary	sex	race
1732	Fire	PHYSICIAN,MD	2014-09-27	342784.0	Male	White
1975	Fire	PHYSICIAN,MD	2014-09-27	342784.0	Male	Asian
4680	Fire	PHYSICIAN,MD	2015-11-23	342784.0	Male	White
4882	Fire	PHYSICIAN,MD	2017-01-09	342784.0	Male	White
6501	Fire	PHYSICIAN,MD	2016-05-31	342784.0	Male	White

```
[79]: dept_freq_top100 = emp_top100['dept'].value_counts(normalize=True)
dept_freq_top100
```

```
[79]: Other          0.36
      Fire           0.22
      Police         0.15
      Houston Airport System 0.09
      Houston Public Works 0.09
      Health & Human Services 0.07
      Library          0.01
      Solid Waste Management 0.01
Name: dept, dtype: float64
```

```
[80]: # The police dept makes up 31% of employees
      # but only 16% of the top 100 salaries
dept_freq_top100 - dept_freq
```

```
[80]: Fire          0.039977
      Health & Human Services 0.014339
      Houston Airport System 0.039975
      Houston Public Works -0.082371
      Library          -0.013161
      Other            0.221239
      Parks & Recreation    NaN
      Police           -0.161544
      Solid Waste Management -0.011063
Name: dept, dtype: float64
```

Exercise 2

Find the day that each stock had its largest percentage one-day drop in price.

```
[81]: stocks = pd.read_csv('../data/stocks/stocks10.csv', index_col='date',
   ↪parse_dates=['date'])
stocks.head(3)
```

	MSFT	AAPL	SLB	AMZN	TSLA	XOM	WMT	T	FB	V
date										
1999-10-25	29.84	2.32	17.02	82.75	NaN	21.45	38.99	16.78	NaN	NaN
1999-10-26	29.82	2.34	16.65	81.25	NaN	20.89	37.11	17.28	NaN	NaN
1999-10-27	29.33	2.38	16.52	75.94	NaN	20.80	36.94	18.27	NaN	NaN

```
[82]: stocks.pct_change().idxmin()
```

```
[82]: MSFT    2000-04-24
AAPL    2000-09-29
SLB     2008-10-15
AMZN    2001-07-24
TSLA    2012-01-13
XOM     2008-10-15
WMT     2018-02-20
T       2000-12-19
FB      2018-07-26
V       2008-10-15
dtype: datetime64[ns]
```

4.7 7. Assigning Subsets of Data

Use the bikes dataset for all of the following exercises.

```
[83]: import pandas as pd
bikes = pd.read_csv('../data/bikes.csv')
```

Exercise 1

Change the values of `events` to ‘HEAT WAVE’ for all rides where `temperature` is above 95. Verify this by outputting just the `events` and `temperature` columns that meet the condition.

```
[84]: filt = bikes['temperature'] > 95
bikes.loc[filt, 'events'] = 'HEAT WAVE'
bikes.loc[filt, ['temperature', 'events']]
```

	temperature	events
395	96.1	HEAT WAVE
396	96.1	HEAT WAVE
397	96.1	HEAT WAVE

Exercise 2

Increase the trip duration by 50% for all the rides that took place with a wind speed above 40. Output just the trip duration and wind speed before and after the assignment.

```
[85]: filt = bikes['wind_speed'] > 40
cols = ['tripduration', 'wind_speed']
bikes.loc[filt, cols]
```

	tripduration	wind_speed
22306	130	42.6
22307	528	42.6
22308	358	42.6
22309	221	41.4

```
[86]: bikes.loc[filt, 'tripduration'] *= 1.5
bikes.loc[filt, cols]
```

	tripduration	wind_speed
22306	195.0	42.6
22307	792.0	42.6
22308	537.0	42.6
22309	331.5	41.4

Exercise 3

Change the trip duration for the first two rows to 0.

```
[87]: bikes.iloc[:, 5] = 0
bikes.head(2)
```

	trip_id	usertype	gender	starttime	stoptime	...	temperature	visibility	wind_speed	precipitation	events
0	7147	Subscriber	Male	2013-06-28 19:01:00	2013-06-28 19:17:00	...	73.9	10.0	12.7	-9999.0	mostlycloudy
1	7524	Subscriber	Male	2013-06-28 22:53:00	2013-06-28 23:03:00	...	69.1	10.0	6.9	-9999.0	partlycloudy

Part V

Data Types

Chapter 5

Solutions

5.1 1. Changing Data Types

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

Exercise 1

Find the maximum integer of a 16-bit integer. Then verify it with numpy's `iinfo` function.

```
[2]: 2 ** 15 - 1
```

```
[2]: 32767
```

```
[3]: np.iinfo('int16')
```

```
[3]: iinfo(min=-32768, max=32767, dtype=int16)
```

Exercise 2

Read in the bikes data and select the `tripduration` column. Find its data type and then use the `memory_usage` method to find how much memory (in bytes) it is using. Change its data type to the smallest possible type so that no information is lost. What percentage of memory has been saved?

```
[4]: bikes = pd.read_csv('../data/bikes.csv')  
td = bikes['tripduration']  
td.head()
```

```
[4]: 0      993  
1      623  
2     1040  
3      667  
4      130  
Name: tripduration, dtype: int64
```

```
[5]: td.memory_usage()
```

```
[5]: 400840
```

Find the min and max values

```
[6]: td.agg(['min', 'max'])
```

```
[6]: min      60
      max    86188
      Name: tripduration, dtype: int64
```

Unfortunately a uint16 doesn't quite have enough memory to fit the max.

```
[7]: np.iinfo('uint16')
```

```
[7]: iinfo(min=0, max=65535, dtype=uint16)
```

We need to use 32 bits. Although you can use uint32 its probably best to stick with int32 as this is much more common.

```
[8]: np.iinfo('int32')
```

```
[8]: iinfo(min=-2147483648, max=2147483647, dtype=int32)
```

```
[9]: td_32 = td.astype('int32')
      td_32.head()
```

```
[9]: 0      993
      1      623
      2     1040
      3      667
      4      130
      Name: tripduration, dtype: int32
```

```
[10]: td_32.memory_usage()
```

```
[10]: 200484
```

We went from using 64 bits for every value to 32 bits. This should yield a decrease of 50% and we verify this below.

```
[11]: td_32.memory_usage() / td.memory_usage()
```

```
[11]: 0.5001596647041213
```

Exercise 3

Create three different Series. Make them each have a different data type and have a different number of items. Make a fourth Series that has these three Series as the values. Output the fourth Series. Can you make sense of it?

```
[12]: s1 = pd.Series([1, 2])
      s2 = pd.Series([4.3, 2.1, 1.554])
      s3 = pd.Series(['python', 'pandas', 'numpy'])
      s4 = pd.Series([s1, s2, s3])
      s4
```

```
[12]: 0      0      1
      1      2
      dtype: int64
      1      0      4.300
      1      2.100
      2      1.554
      dtype: float64
      2      0      python
      1      pandas
      2      numpy
      dtype: object
      dtype: object
```

It's very hard to decipher what is going on. Series objects are not designed to contain other Series.

Exercise 4

What month is it 1 million minutes after the unix epoch?

```
[13]: s = pd.Series([1000000]).astype('datetime64[m]')
s
```

```
[13]: 0    1971-11-26 10:40:00
      dtype: datetime64[ns]
```

```
[14]: s.dt.month_name()
```

```
[14]: 0    November
      dtype: object
```

In the time series part, you will learn how to do this in a more direct manner.

```
[15]: pd.Timestamp(1000000, unit='m').month_name()
```

```
[15]: 'November'
```

Exercise 5

Convert the following Series to float.

```
[16]: s = pd.Series(['1.9', '43', 'python'])
s
```

```
[16]: 0      1.9
      1      43
      2      python
      dtype: object
```

```
[17]: pd.to_numeric(s, errors='coerce')
```

```
[17]: 0      1.9
      1      43.0
      2      NaN
```

```
dtype: float64
```

Exercise 6

Take a look at the `dpcapacity_start` column from the bikes dataset. It contains the capacity of the bike rack when the ride began. This number should be an integer but it is a float. Why do you think pandas read this in as a float? Do something to the DataFrame as a whole so that you can convert just this column to an integer. Choose the lowest size integer that is possible.

```
[18]: bikes['dpcapacity_start'].head()
```

```
[18]: 0    11.0
      1    31.0
      2    15.0
      3    19.0
      4    19.0
Name: dpcapacity_start, dtype: float64
```

There must be missing values since integers do not contain them.

```
[19]: bikes['dpcapacity_start'].isna().sum()
```

```
[19]: 6
```

The values easily fit within a 8-bit integer.

```
[20]: bikes['dpcapacity_start'].agg(['min', 'max'])
```

```
[20]: min      0.0
      max     55.0
Name: dpcapacity_start, dtype: float64
```

Let's drop the missing values just for that column and then convert it to an `int8`.

```
[21]: bikes2 = bikes.dropna(subset=['dpcapacity_start']) \
                 .astype({'dpcapacity_start': 'int8'})
bikes2['dpcapacity_start'].head()
```

```
[21]: 0    11
      1    31
      2    15
      3    19
      4    19
Name: dpcapacity_start, dtype: int8
```

5.2 2. Categorical Data Type

Execute the cell below to read in the diamonds dataset and use it to answer the following questions.

```
[22]: diamonds = pd.read_csv('../data/diamonds.csv')
diamonds.head(3)
```

	carat	cut	color	clarity	depth	table	price	x	y	z
0	0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2.43
1	0.21	Premium	E	SI1	59.8	61.0	326	3.89	3.84	2.31
2	0.23	Good	E	VS1	56.9	65.0	327	4.05	4.07	2.31

```
[23]: pd.set_option('display.max_colwidth', 100)
pd.read_csv('../data/dictionaries/diamonds_dictionary.csv')
```

Column Name	Description
0 carat	weight of the diamond (0.2--5.01)
1 clarity	a measurement of how clear the diamond is (I1 (worst), SI2, SI1, VS2, VS1, VVS2, VVS1, IF (best))
2 color	diamond colour, from J (worst) to D (best)
3 cut	quality of the cut (Fair, Good, Very Good, Premium, Ideal)
4 depth	total depth percentage = z / mean(x, y) = 2 * z / (x + y) (43--79)
5 price	price in US dollars (\$326--\$18,823)
6 table	width of top of diamond relative to widest point (43--95)
7 x	length in mm (0--10.74)
8 y	width in mm (0--58.9)
9 z	depth in mm (0--31.8)

Exercise 1

Create a new DataFrame `diamonds2` that has the columns `clarity`, `color`, and `cut` as ordered categoricals.

```
[24]: clarity_cats = ['I1', 'SI2', 'SI1', 'VS2', 'VS1', 'VVS2', 'VVS1', 'IF']
clarity_dtype = pd.CategoricalDtype(clarity_cats, ordered=True)

color_cats = ['J', 'I', 'H', 'G', 'F', 'E', 'D']
color_dtype = pd.CategoricalDtype(color_cats, ordered=True)

cut_cats = ['Fair', 'Good', 'Very Good', 'Premium', 'Ideal']
cut_dtype = pd.CategoricalDtype(cut_cats, ordered=True)

diamonds2 = diamonds.astype({'clarity': clarity_dtype, 'color': color_dtype, 'cut': cut_dtype})
diamonds2.head(3)
```

	carat	cut	color	clarity	depth	table	price	x	y	z
0	0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2.43
1	0.21	Premium	E	SI1	59.8	61.0	326	3.89	3.84	2.31
2	0.23	Good	E	VS1	56.9	65.0	327	4.05	4.07	2.31

```
[25]: diamonds2.dtypes
```

```
[25]: carat      float64
cut        category
color      category
clarity    category
depth      float64
table      float64
price      int64
x          float64
y          float64
z          float64
dtype: object
```

Exercise 2

Find the number of occurrences for each cut type sorted by its cut order.

```
[26]: diamonds2['cut'].value_counts().sort_index()
```

```
[26]: Fair        1610
Good       4906
Very Good 12082
Premium    13791
Ideal      21551
Name: cut, dtype: int64
```

5.3 3. Nullable Integer Data Type

Part VI

Grouping Data

Chapter 6

Solutions

6.1 1. Groupby Aggregation Basics

```
[1]: import pandas as pd  
emp = pd.read_csv('../data/employee.csv')  
emp.head(3)
```

	dept	title	hire_date	salary	sex	race
0	Police	POLICE SERGEANT	2001-12-03	87545.38	Male	White
1	Other	ASSISTANT CITY ATTORNEY II	2010-11-15	82182.00	Male	Hispanic
2	Houston Public Works	SENIOR SLUDGE PROCESSOR	2006-01-09	49275.00	Male	Black

Exercise 1

Find the maximum salary for each sex.

```
[2]: emp.groupby('sex').agg(max_salary=('salary', 'max'))
```

sex	max_salary
Female	342784.0
Male	342784.0

Exercise 2

Find the median salary for each department.

```
[3]: emp.groupby('dept').agg(median_salary=('salary', 'median')).head()
```

median_salary	
dept	
Fire	61921.08
Health & Human Services	50773.00
Houston Airport System	44200.00
Houston Public Works	46841.50
Library	34611.00

Exercise 3

Find the average salary for each race. Return a DataFrame with the race as a column.

```
[4]: emp.groupby('race').agg(avg_salary=('salary', 'mean')).round(-3).reset_index()
```

	race	avg_salary
0	Asian	65000.0
1	Black	52000.0
2	Hispanic	55000.0
3	Native American	58000.0
4	White	67000.0

Execute the cell below to read in the NYC deaths dataset and use it to answer the following exercises.

```
[5]: deaths = pd.read_csv('../data/nyc_deaths.csv')
deaths.head(3)
```

	year	cause	sex	race	deaths
0	2007	Accidents	F	Asian	32
1	2007	Accidents	F	Black	87
2	2007	Accidents	F	Hispanic	71

Exercise 4

What year had the most deaths?

```
[6]: year_deaths = deaths.groupby('year').agg(total=('deaths', 'sum'))
year_deaths
```

total	
year	
2007	53996
2008	54138
2009	52820
2010	52505
2011	52726
2012	52420
2013	53387
2014	53006

```
[7]: year_deaths.agg(['max', 'idxmax'])
```

total	
max	54138
idxmax	2008

Exercise 5

Find the total number of deaths by race and sort by most to least.

```
[8]: deaths.groupby('race').agg(total=('deaths', 'sum')).sort_values('total', ↴ ascending=False)
```

total	
race	
White	206487
Black	111116
Hispanic	74802
Asian	26355
Unknown	6238

6.2 2. Grouping and Aggregating with Multiple Columns

Execute the following cell to read in the City of Houston employee data and use it for the first few exercises.

```
[9]: import pandas as pd
emp = pd.read_csv('../data/employee.csv', parse_dates=['hire_date'])
emp.head(3)
```

	dept		title	hire_date	salary	sex	race
0	Police	POLICE SERGEANT	2001-12-03	87545.38	Male	White	
1	Other	ASSISTANT CITY ATTORNEY II	2010-11-15	82182.00	Male	Hispanic	
2	Houston Public Works	SENIOR SLUDGE PROCESSOR	2006-01-09	49275.00	Male	Black	

Exercise 1

For each department and sex find the number of unique position titles, the total number of employees and the average salary. Make sure there is no multi-index for the index or columns.

```
[10]: data = emp.groupby(['dept', 'sex']).agg(num_unique_titles=('title', 'nunique'),
                                             num_employees=('title', 'size'),
                                             avg_salaray=('salary', 'mean')).reset_index()
data.head(10)
```

	dept	sex	num_unique_titles	num_employees	avg_salaray
0	Fire	Female	51	240	62212.637250
1	Fire	Male	54	4136	60479.306862
2	Health & Human Services	Female	136	987	53838.310780
3	Health & Human Services	Male	110	366	59230.425956
4	Houston Airport System	Female	85	443	51099.300226
5	Houston Airport System	Male	113	773	57278.306598
6	Houston Public Works	Female	151	1195	51294.453004
7	Houston Public Works	Male	180	2995	51490.113309
8	Library	Female	55	404	41126.962921
9	Library	Male	44	159	44399.943396

Exercise 2

For each department, race and sex find the min and max and salaries.

```
[11]: emp.groupby(['dept', 'race', 'sex']).agg(min_salary=('salary', 'min'),
                                              max_salary=('salary', 'max')).head(10)
```

dept	race	sex	min_salary max_salary	
Fire	Asian	Female	39104.0	342784.00
		Male	28024.0	342784.00
	Black	Female	16411.0	342784.00
		Male	28024.0	342784.00
	Hispanic	Female	28024.0	89590.02
		Male	26000.0	342784.00
	Native American	Female	48189.7	70181.28
		Male	28024.0	115835.98
	White	Female	16910.0	342784.00
		Male	16515.0	342784.00

Execute the following cell to read in the college dataset and use it for the remaining exercises.

```
[12]: college = pd.read_csv('../data/college.csv')
college.head(3)
```

	instnm	city	stabbr	hbcu	menonly	...	pctpell	pctfloan	ug25abv	md_earn_wne_p10	grad_debt_mdn_supp
0	Alabama A & M University	Normal	AL	1.0	0.0	...	0.7356	0.8284	0.1049	30300	33888
1	University of Alabama at Birmingham	Birmingham	AL	0.0	0.0	...	0.3460	0.5214	0.2422	39700	21941.5
2	Amridge University	Montgomery	AL	0.0	0.0	...	0.6801	0.7795	0.8540	40100	23370

Exercise 3

Which city name appears the most frequently. Do this in two different ways. Do it once with and once without the `groupby` method?

```
[13]: size = college.groupby('city').agg(size=('stabbr', 'size'))
size.head()
```

	size
city	
Aberdeen	3
Abilene	5
Abingdon	2
Abington	1
Ada	3

```
[14]: size.sort_values('size', ascending=False).head()
```

	size
city	
New York	87
Chicago	78
Houston	72
Los Angeles	56
Miami	51

Can also just `size` directly and sort the series.

Without groupby

Use `value_counts`

```
[15]: college['city'].value_counts().head()
```

```
[15]: New York      87
      Chicago       78
      Houston        72
      Los Angeles    56
      Miami          51
      Name: city, dtype: int64
```

Exercise 4

Does the city ‘Houston’ only appear in the state of Texas (abbreviated ‘TX’)?

NO! It also appears in Missouri.

```
[16]: filt = college['city'] == 'Houston'
college.loc[filt, 'stabbr'].unique()
```

```
[16]: array(['TX', 'MO'], dtype=object)
```

Can see exact counts

```
[17]: college.loc[filt, 'stabbr'].value_counts()
```

```
[17]: TX      71
MO      1
Name: stabbr, dtype: int64
```

You can use a groupby and find the number of unique states for each city. This is not very efficient.

```
[18]: city_unique_state = college.groupby('city').agg(num_unique_states=('stabbr', u
    ↴'nunique'))
city_unique_state.head()
```

city	num_unique_states
Aberdeen	2
Abilene	1
Abingdon	1
Abington	1
Ada	2

```
[19]: city_unique_state.loc['Houston']
```

```
[19]: num_unique_states      2
Name: Houston, dtype: int64
```

Also with drop_duplicates

```
[20]: college[['city', 'stabbr']].query('city == "Houston"') \
        .drop_duplicates(subset='stabbr')
```

city	stabbr
3617	Houston TX
5366	Houston MO

Exercise 5

Find the maximum undergraduate population for each state?

```
[21]: college.groupby('stabbr').agg(max_ugds=('ugds', 'max')).head(10)
```

max_ugds	
stabbr	
AK	12865.0
AL	29851.0
AR	21405.0
AS	1276.0
AZ	151558.0
CA	44744.0
CO	25873.0
CT	18016.0
DC	10433.0
DE	18222.0

Exercise 6

Among colleges that have the largest undergrad population for each state, what is the difference between the most and least populous college?

```
[22]: largest_per_state = college.groupby('stabbr').agg(max_ugds=('ugds', 'max'))
largest_per_state.max() - largest_per_state.min()
```

```
[22]: max_ugds    150956.0
      dtype: float64
```

Exercise 7

Find the name and population of the largest college per state.

```
[23]: # set the index first to be instnm so that you can take advantage of idxmax
c2 = college.set_index('instnm')
max_indexes = c2.groupby('stabbr').agg(max_ugds=('ugds', 'max'),
                                         max_ugds_college=('ugds', 'idxmax'))
max_indexes.head(10)
```

stabbr	max_ugds	max_ugds_college
AK	12865.0	University of Alaska Anchorage
AL	29851.0	The University of Alabama
AR	21405.0	University of Arkansas
AS	1276.0	American Samoa Community College
AZ	151558.0	University of Phoenix-Arizona
CA	44744.0	Ashford University
CO	25873.0	University of Colorado Boulder
CT	18016.0	University of Connecticut
DC	10433.0	George Washington University
DE	18222.0	University of Delaware

Another way

Use the `first` groupby method to return the first row of each group after sorting.

```
[24]: cols = ['stabbr', 'instnm', 'ugds']
college_trim = college[cols]

# sort by state then by population descending
college_trim_sort = college_trim.sort_values(['stabbr', 'ugds'], ascending=[True, False])

# group by state and take the first in the group
college_trim_sort.groupby('stabbr').first().head()
```

stabbr	instnm	ugds
AK	University of Alaska Anchorage	12865.0
AL	The University of Alabama	29851.0
AR	University of Arkansas	21405.0
AS	American Samoa Community College	1276.0
AZ	University of Phoenix-Arizona	151558.0

Use `sort_values` with `drop_duplicates`

We've done this in previous notebooks. No grouping.

```
[25]: college_trim.sort_values(['stabbr', 'ugds'], ascending=[True, False]) \
    .drop_duplicates(subset='stabbr').head(10)
```

	stabbr	instnm	ugds
60	AK	University of Alaska Anchorage	12865.0
5	AL	The University of Alabama	29851.0
137	AR	University of Arkansas	21405.0
4138	AS	American Samoa Community College	1276.0
7116	AZ	University of Phoenix-Arizona	151558.0
1299	CA	Ashford University	44744.0
574	CO	University of Colorado Boulder	25873.0
641	CT	University of Connecticut	18016.0
701	DC	George Washington University	10433.0
691	DE	University of Delaware	18222.0

Exercise 8

Do distance only schools tend to have more or less student population than non-distance-only schools?

```
[26]: # They have more
college.groupby('distanceonly').agg(mean_ugds=('ugds', 'mean'))
```

	mean_ugds
distanceonly	
0.0	2334.648135
1.0	6245.743590

Exercise 9

Do distance only schools tend to be more or less religiously affiliated than non-distance-only schools?

```
[27]: # Less
college.groupby('distanceonly').agg(mean_relaffil=('relaffil', 'mean'))
```

	mean_relaffil
distanceonly	
0.0	0.149635
1.0	0.050000

Exercise 10

What state has the lowest percentage of currently operating schools of those that have religious affiliation?

```
[28]: filt = college['relaffil'] == 1
cr = college[filt]
```

```
rel_oper_mean = cr.groupby('stabbr').agg(mean_curroper=('curroper', 'mean'))
rel_oper_mean.head()
```

mean_curroper	
stabbr	
AK	1.000000
AL	0.916667
AR	0.944444
AZ	0.444444
CA	0.585366

[29]: rel_oper_mean.sort_values('mean_curroper').head()

mean_curroper	
stabbr	
UT	0.400000
AZ	0.444444
NV	0.500000
CA	0.585366
CT	0.647059

Exercise 11

Find the top 5 historically black colleges that have the highest undergraduate white percentage (ugds_white)?

[30]: filt = college['hbcu'] == 1
cols = ['instnm', 'ugds_white']
college.loc[filt, cols].sort_values('ugds_white', ascending=False).head()

	instnm	ugds_white
4021	Bluefield State College	0.8437
17	Gadsden State Community College	0.6921
4050	West Virginia State University	0.5816
48	Shelton State Community College	0.5613
55	H Councill Trenholm State Community College	0.3951

6.3 3. Grouping with Pivot Tables

```
[31]: import pandas as pd
flights = pd.read_csv('../data/flights.csv', parse_dates=['date'])
flights['day_of_week'] = flights['date'].dt.day_name()
flights['month'] = flights['date'].dt.month_name()
flights.head(3)
```

	date	airline	origin	dest	dep_time	...	nas_delay	security_delay	late_aircraft_delay	day_of_week	month
0	2018-01-01	UA	LAS	IAH	100	...	0	0	0	Monday	January
1	2018-01-01	WN	DEN	PHX	515	...	0	0	0	Monday	January
2	2018-01-01	B6	JFK	BOS	550	...	8	0	0	Monday	January

```
[32]: flights.shape
```

[32]: (65923, 16)

Exercise 1

What is the carrier departure delay for each day of the week for each airline? Highlight the worst day of the week for each airline.

```
[33]: avg_delay = flights.pivot_table(index='airline', columns='day_of_week',
                                     values='carrier_delay').round(1)
avg_delay.style.highlight_max(axis='columns')
```

	day_of_week	Friday	Monday	Saturday	Sunday	Thursday	Tuesday	Wednesday
airline								
9E	3.4	4.7	1.8	3.3	7.7	3	4.2	
AA	3.9	3.7	3.8	3.8	4.9	4.1	2.8	
AS	3	1.5	3.6	3.7	2.9	2.6	2.1	
B6	7	4	5.5	5.1	5.1	6.2	4	
DL	3.5	2.6	3	3.6	3.9	3.3	3.3	
EV	5.2	8.5	2.4	7.2	0	6	1.7	
F9	7.9	3.6	9.5	4.1	7.8	6.6	1.8	
MQ	2.1	4.2	0	5.4	4.2	1.5	2.3	
NK	2.2	1.5	5.8	1.7	2.2	2.5	1.2	
OH	6.3	4.9	2.5	9.2	19.9	0.5	1.5	
OO	6.5	9.5	2	4.8	3.5	1.9	10.2	
UA	3.2	2.4	3.2	4	3.7	3.7	2.6	
VX	2.6	1.7	10.1	0.3	2.8	3.2	5.2	
WN	5	3.4	2.2	2	4.8	3.5	3.2	
YV	5.5	5.6	5.6	7.8	8.6	1.8	4	
YX	2.2	4.1	2.5	2.1	2.9	1.5	1.4	

You can highlight min and max by chaining style methods.

```
[34]: avg_delay.style.highlight_max(axis='columns') \
    .highlight_min(axis='columns', color='lightblue')
```

day_of_week	Friday	Monday	Saturday	Sunday	Thursday	Tuesday	Wednesday
airline							
9E	3.4	4.7	1.8	3.3	7.7	3	4.2
AA	3.9	3.7	3.8	3.8	4.9	4.1	2.8
AS	3	1.5	3.6	3.7	2.9	2.6	2.1
B6	7	4	5.5	5.1	5.1	6.2	4
DL	3.5	2.6	3	3.6	3.9	3.3	3.3
EV	5.2	8.5	2.4	7.2	0	6	1.7
F9	7.9	3.6	9.5	4.1	7.8	6.6	1.8
MQ	2.1	4.2	0	5.4	4.2	1.5	2.3
NK	2.2	1.5	5.8	1.7	2.2	2.5	1.2
OH	6.3	4.9	2.5	9.2	19.9	0.5	1.5
OO	6.5	9.5	2	4.8	3.5	1.9	10.2
UA	3.2	2.4	3.2	4	3.7	3.7	2.6
VX	2.6	1.7	10.1	0.3	2.8	3.2	5.2
WN	5	3.4	2.2	2	4.8	3.5	3.2
YV	5.5	5.6	5.6	7.8	8.6	1.8	4
YX	2.2	4.1	2.5	2.1	2.9	1.5	1.4

Exercise 2

Use a `pivot_table` to find the total number of canceled flights for each origin airport and airline making sure the airline is in the columns. Use the result to find the origin airport with the most cancelled flights for each airline. Also return this maximum number of cancelled flights.

```
[35]: airline_cancel = flights.pivot_table(index='origin', columns='airline',
                                         values='cancelled', aggfunc='sum', fill_value=0)
airline_cancel.head(10)
```

airline	9E	AA	AS	B6	DL	...	UA	VX	WN	YV	YX
origin											
ATL	0	5	0	2	19	...	2	0	8	4	9
BOS	5	41	1	31	9	...	18	0	2	0	12
CLT	6	33	0	2	1	...	0	0	0	0	11
DCA	1	27	0	3	3	...	2	0	6	0	31
DEN	0	3	1	0	0	...	10	0	9	0	0
DFW	1	33	0	0	1	...	1	0	0	1	7
DTW	1	4	0	3	8	...	0	0	1	1	8
EWR	2	10	6	8	0	...	27	1	1	0	15
IAH	0	7	0	0	1	...	7	0	0	4	4
JFK	10	6	3	17	3	...	0	1	0	0	4

```
[36]: airline_cancel.agg(['max', 'idxmax'])
```

	9E	AA	AS	B6	DL	...	UA	VX	WN	YV	YX
max	10	41	9	31	19	...	27	6	18	4	31
idxmax	JFK	BOS	SEA	BOS	ATL	...	EWR	LAX	LAX	ATL	DCA

Exercise 3

Find the total distance flown for each airline for each month. Highlight the month with the most number of miles flown and use the style `format` method to put commas in the numbers so that they are easier to read.

```
[37]: total_dist = flights.pivot_table(index='airline', columns='month',
                                         values='distance', aggfunc='sum')
```

```
[38]: total_dist.style.format('{:.0f}').highlight_max(axis='columns')
```

month	April	August	December	February	January	July	June	March	May	November	October	September
airline												
9E	54,592	62,216	46,032	51,784	47,230	53,868	50,421	61,460	42,423	42,275	48,106	45,745
AA	1,586,655	1,649,436	1,444,276	1,371,620	1,473,883	1,669,007	1,619,325	1,528,361	1,545,453	1,409,540	1,588,285	1,482,841
AS	454,146	451,512	399,787	201,275	195,553	455,061	496,358	199,288	495,090	391,304	409,479	429,045
B6	352,234	404,458	427,097	348,189	385,517	478,230	443,151	382,666	410,877	384,038	425,712	384,008
DL	1,265,266	1,315,865	1,160,997	997,216	1,017,440	1,396,697	1,292,928	1,215,516	1,253,361	1,100,681	1,214,950	1,173,359
EV	6,847	1,194	3,933	11,854	10,186	927	5,926	4,511	3,569	1,592	2,587	995
F9	117,439	97,777	97,846	97,879	118,067	84,417	116,116	80,444	78,807	110,423	105,833	89,938
MQ	13,060	15,787	14,057	17,539	15,170	20,057	15,310	13,349	13,656	15,559	16,884	14,767
NK	250,683	270,894	232,613	219,678	249,461	273,963	318,648	228,829	261,421	266,838	253,692	235,754
OH	8,280	7,986	8,596	9,911	14,802	6,461	5,808	14,664	5,296	4,948	6,028	7,674
OO	124,729	90,098	123,870	112,342	114,927	87,394	87,014	116,430	130,298	133,669	122,277	113,162
UA	1,211,211	1,350,333	1,184,206	1,023,812	1,055,351	1,353,485	1,332,403	1,101,258	1,303,095	1,125,754	1,251,047	1,272,765
VX	nan	nan	nan	180,621	168,486	nan	nan	244,380	nan	nan	nan	nan
WN	367,918	339,114	327,634	288,393	347,737	350,299	383,858	332,745	347,911	329,873	333,324	288,325
YV	57,510	45,114	73,982	45,261	36,986	59,699	60,166	44,678	51,225	59,978	52,643	36,430
YX	164,271	177,050	141,817	174,891	163,374	179,511	162,816	178,524	164,673	116,173	148,398	160,317

Exercise 4

Use the City of Houston employee dataset for this exercise. You can create pivot tables with multiple columns in the index or the columns by using a list. Create a pivot table with the department as the index and the race and sex as the columns. Calculate the median salary for these cross sections.

```
[39]: emp = pd.read_csv('../data/employee.csv')
emp.pivot_table(index='dept', columns=['race', 'sex'],
                 values='salary', aggfunc='median').round(-3).style.format('{:, .0f}')
```

race	Asian		Black		Hispanic		Native American		White		
	sex	Female	Male	Female	Male	Female	Male	Female	Male	Female	Male
dept											
Fire		52,000	51,000	55,000	61,000	48,000	58,000	59,000	63,000	61,000	63,000
Health & Human Services		59,000	59,000	51,000	53,000	38,000	51,000	37,000	49,000	59,000	62,000
Houston Airport System		35,000	55,000	40,000	43,000	34,000	42,000	44,000	51,000	59,000	62,000
Houston Public Works		67,000	60,000	40,000	39,000	40,000	48,000	41,000	56,000	60,000	59,000
Library		36,000	33,000	33,000	35,000	33,000	36,000	31,000	29,000	44,000	50,000
Other		73,000	61,000	53,000	48,000	45,000	47,000	47,000	50,000	66,000	64,000
Parks & Recreation		29,000	41,000	30,000	30,000	30,000	33,000	30,000	39,000	41,000	39,000
Police		61,000	66,000	50,000	68,000	57,000	68,000	47,000	76,000	63,000	73,000
Solid Waste Management		71,000	45,000	39,000	39,000	45,000	41,000	31,000	nan	57,000	38,000

6.4 4. Counting with Crosstabs

```
[40]: import pandas as pd
pd.options.display.max_columns = 100
pd.options.display.max_colwidth = 200
mh = pd.read_csv('../data/mental_health.csv')
mh.head(3)
```

	timestamp	age	gender	country	state	...	mental_health_interview	phys_health_interview	mental_vs_physical	obs_consequence	comments
0	2014-08-27 11:29:31	37	Female	United States	IL	...	No	Maybe	Yes	No	NaN
1	2014-08-27 11:29:37	44	M	United States	IN	...	No	No	Don't know	No	NaN
2	2014-08-27 11:29:44	32	Male	Canada	NAN	...	Yes	Yes	No	No	NaN

Exercise 1

Do people with a family history of mental illness seek treatment more often than those who do not?

```
[41]: pd.crosstab(index=mh['family_history'], columns=mh['treatment'])
```

	treatment	No	Yes
family_history			
No	427	255	
Yes	116	343	

```
[42]: pd.crosstab(index=mh['family_history'], columns=mh['treatment'], normalize='index').
      round(2)
```

	treatment	No	Yes
family_history			
No	0.63	0.37	
Yes	0.25	0.75	

Yes, there is a large difference. 75% of people with a family history seek treatment vs 37% for those who have not.

Exercise 2

Find the total number and ratio of employees that seek treatment for companies that provide health benefits vs those that do not.

```
[43]: pd.crosstab(index=mh['benefits'], columns=mh['treatment'])
```

	treatment	No	Yes
benefits			
Don't know	231	141	
No	146	158	
Yes	166	299	

```
[44]: pd.crosstab(index=mh['benefits'], columns=mh['treatment'], normalize='index').round(2)
```

	treatment	No	Yes
benefits			
Don't know	0.62	0.38	
No	0.48	0.52	
Yes	0.36	0.64	

Exercise 3

You can provide a list of multiple columns to both the `index` and `columns` parameters of the `crosstab` function. Put country and number of employees in the index and benefits and treatment in the columns. It's probably easier to make separate list variables first.

```
[45]: index = [mh['country'], mh['no_employees']]
columns = [mh['benefits'], mh['treatment']]
pd.crosstab(index=index, columns=columns)
```

	country	no_employees	benefits		Don't know		No	Yes	
			treatment		No	Yes	No	Yes	No
	Australia	1-5	1	0	1	1	0	0	0
		100-500	1	0	1	2	0	2	
		26-100	0	0	1	3	0	0	
		500-1000	1	0	0	0	0	0	
		6-25	0	1	0	3	0	0	
		More than 1000	1	0	0	0	1	1	
	Canada	1-5	1	0	5	5	0	0	
		100-500	2	3	0	0	2	4	
		26-100	4	4	3	1	3	3	
		500-1000	0	0	0	0	0	1	
		6-25	4	2	6	1	3	5	
		More than 1000	0	2	0	0	2	6	
	France	100-500	1	0	1	0	0	0	
		26-100	0	0	3	1	0	0	
		500-1000	1	0	0	0	1	0	

	Netherlands	500-1000	1	0	0	0	0	0	
		6-25	1	1	7	1	0	0	
		More than 1000	0	0	0	0	1	0	
	United Kingdom	1-5	6	3	8	13	0	1	
		100-500	5	6	2	4	2	1	
		26-100	12	5	7	11	3	2	
		500-1000	2	3	1	0	1	0	
		6-25	11	8	23	14	1	1	
		More than 1000	5	6	3	7	0	8	
	United States	1-5	11	4	16	31	3	11	
		100-500	22	15	3	8	23	42	
		26-100	42	27	7	7	21	66	
		500-1000	9	1	1	1	10	20	
		6-25	32	23	16	20	17	26	
		More than 1000	35	15	5	2	68	91	

6.5 5. Alternate Groupby Syntax

6.6 6. Custom Aggregation

Execute the cell below to read in the flights dataset and then use it for the following exercises.

```
[46]: import pandas as pd
flights = pd.read_csv('../data/flights.csv', parse_dates=['date'])
flights.head(3)
```

	date	airline	origin	dest	dep_time	...	carrier_delay	weather_delay	nas_delay	security_delay	late_aircraft_delay
0	2018-01-01	UA	LAS	IAH	100	...	0	0	0	0	0
1	2018-01-01	WN	DEN	PHX	515	...	0	0	0	0	0
2	2018-01-01	B6	JFK	BOS	550	...	0	83	8	0	0

Exercise 1

What are the 3 least common airlines?

```
[47]: flights['airline'].value_counts().tail(3)
```

```
[47]: MQ      373
OH      257
EV      171
Name: airline, dtype: int64
```

Exercise 2

For each airline, find out what percentage of its flights leave on a Tuesday. Use a custom aggregation function.

```
[48]: def tuesday_pct(s):
    return (s.dt.day_name() == 'Tuesday').mean()

flights.groupby('airline').agg(percent_tuesday=('date', tuesday_pct))
```

airline	percent_tuesday
9E	0.144648
AA	0.146195
AS	0.138101
B6	0.134958
DL	0.143544
EV	0.157895
F9	0.128834
MQ	0.160858
NK	0.128437
OH	0.167315
OO	0.143405
UA	0.139875
VX	0.132867
WN	0.152687
YV	0.138546
YX	0.150000

Exercise 3

Redo Exercise 2 without using a custom aggregation Exercise. What is the performance difference?

```
[49]: flights['is_tuesday'] = flights['date'].dt.day_name() == 'Tuesday'  
flights.groupby('airline').agg(percent_tuesday=('is_tuesday', 'mean'))
```

percent_tuesday	
airline	
9E	0.144648
AA	0.146195
AS	0.138101
B6	0.134958
DL	0.143544
EV	0.157895
F9	0.128834
MQ	0.160858
NK	0.128437
OH	0.167315
OO	0.143405
UA	0.139875
VX	0.132867
WN	0.152687
YV	0.138546
YX	0.150000

About 50% improvement

```
[50]: %%timeit -n 1 flights.groupby('airline').agg(percent_tuesday=('date', 'tuesday_pct'))
```

40.7 ms ± 620 µs per loop (mean ± std. dev. of 7 runs, 1 loop each)

```
[51]: %%timeit -n 1
# flights['is_tuesday'] = flights['date'].dt.day_name() == 'Tuesday'
flights.groupby('airline').agg(percent_tuesday='is_tuesday', 'mean'))
```

9.37 ms ± 535 µs per loop (mean ± std. dev. of 7 runs, 1 loop each)

Exercise 4

The range of undergrad populations per state was calculated using the `min_max` custom function from the top of this notebook. Use this same function to calculate the range of distance for each airline. Then calculate this range again without a custom function.

```
[52]: def min_max(s):
    return s.max() - s.min()
```

```
[53]: flights.groupby('airline').agg(dist_range='distance', min_max))
```

airline	dist_range
9E	1297.0
AA	2515.0
AS	2468.0
B6	2520.0
DL	2610.0
EV	876.0
F9	2042.0
MQ	831.0
NK	2166.0
OH	835.0
OO	1405.0
UA	2505.0
VX	2468.0
WN	1940.0
YV	1192.0
YX	1320.0

```
[54]: dist_min_max = flights.groupby('airline').agg(max_dist=('distance', 'max'),
                                              min_dist=('distance', 'min'))
dist_min_max['dist range'] = dist_min_max['max_dist'] - dist_min_max['min_dist']
dist_min_max
```

		max_dist	min_dist	dist range
airline				
	9E	1391.0	94.0	1297.0
	AA	2611.0	96.0	2515.0
	AS	2704.0	236.0	2468.0
	B6	2704.0	184.0	2520.0
	DL	2704.0	94.0	2610.0
	EV	1075.0	199.0	876.0
	F9	2446.0	404.0	2042.0
	MQ	925.0	94.0	831.0
	NK	2402.0	236.0	2166.0
	OH	931.0	96.0	835.0
	OO	1589.0	184.0	1405.0
	UA	2704.0	199.0	2505.0
	VX	2704.0	236.0	2468.0
	WN	2176.0	236.0	1940.0
	YV	1416.0	224.0	1192.0
	YX	1416.0	96.0	1320.0

Exercise 5

For each airline, return the first and last row of each group. Use one of the direct [Groupby](#) methods

```
[55]: flights.groupby('airline').nth([0, -1]).head()
```

	date	origin	dest	dep_time	arr_time	...	weather_delay	nas_delay	security_delay	late_aircraft_delay	is_tuesday
airline											
9E	2018-12-31	CLT	JFK	1400	1603	...	0	0	0	0	False
9E	2018-01-01	IAH	ATL	1346	1651	...	0	0	0	0	False
AA	2018-01-01	DFW	DCA	610	959	...	0	0	0	0	False
AA	2018-12-31	DFW	SFO	2047	2245	...	0	0	0	0	False
AS	2018-12-31	SEA	DFW	2315	502	...	0	3	0	26	False

6.7 9. Transform and Filter with Groupby

Execute the following cell to read in the college dataset and then use it for the following exercises.

```
[56]: pd.options.display.max_columns = 100
college = pd.read_csv('../data/college.csv')
college.head(3)
```

	instnm	city	stabbr	hbcu	menonly	...	pctpell	pctfloan	ug25abv	md_earn_wne_p10	grad_debt_mdn_supp
0	Alabama A & M University	Normal	AL	1.0	0.0	...	0.7356	0.8284	0.1049	30300	33888
1	University of Alabama at Birmingham	Birmingham	AL	0.0	0.0	...	0.3460	0.5214	0.2422	39700	21941.5
2	Amridge University	Montgomery	AL	0.0	0.0	...	0.6801	0.7795	0.8540	40100	23370

Exercise 1

Filter the college DataFrame for states that have more than 500,000 total undergraduate students. Can you verify your results?

```
[57]: college_large = college.groupby('stabbr').filter(lambda sub_df: sub_df['ugds'].sum() > 500000)
college_large.head()
```

	instnm	city	stabbr	hbcu	menonly	...	pctpell	pctfloan	ug25abv	md_earn_wne_p10	grad_debt_mdn_supp
43	Prince Institute-Southeast	Elmhurst	IL	0.0	0.0	...	0.7857	0.9375	0.6569	PrivacySuppressed	20992
68	Everest College-Phoenix	Phoenix	AZ	0.0	0.0	...	0.8291	0.7151	0.6700	28600	9500
69	Collins College	Phoenix	AZ	0.0	0.0	...	0.7205	0.8228	0.4764	25700	47000
70	Empire Beauty School-Paradise Valley	Phoenix	AZ	0.0	0.0	...	0.6349	0.5873	0.4651	17800	9588
71	Empire Beauty School-Tucson	Tucson	AZ	0.0	0.0	...	0.7962	0.6615	0.4229	18200	9833

```
[58]: college_large.groupby('stabbr').agg(ugds_total=('ugds', 'sum')) \
    .sort_values('ugds_total', ascending=False).round(-3)
```

ugds_total		
stabbr		
CA	2304000.0	
TX	1277000.0	
NY	994000.0	
FL	960000.0	
PA	605000.0	
IL	600000.0	
OH	538000.0	
AZ	520000.0	

Execute the following cell to read in the City of Houston employee dataset and then use it for the following exercises.

```
[59]: emp = pd.read_csv('../data/employee.csv')
emp.head()
```

	dept	title	hire_date	salary	sex	race
0	Police	POLICE SERGEANT	2001-12-03	87545.38	Male	White
1	Other	ASSISTANT CITY ATTORNEY II	2010-11-15	82182.00	Male	Hispanic
2	Houston Public Works	SENIOR SLUDGE PROCESSOR	2006-01-09	49275.00	Male	Black
3	Police	SENIOR POLICE OFFICER	1997-05-27	75942.10	Male	Hispanic
4	Police	SENIOR POLICE OFFICER	2006-01-23	69355.26	Male	White

Exercise 2

Filter it so that only position titles with an average salary of 100,000 remain. Can you verify your results?

```
[60]: high_sal = emp.groupby('title').filter(lambda sub_df: sub_df['salary'].mean() > 100000)
high_sal.head()
```

	dept	title	hire_date	salary	sex	race
16	Other	ASSOCIATE JUDGE OF MUNICIPAL COURTS	2005-11-09	107744.00	Male	Hispanic
17	Police	POLICE COMMANDER	1983-02-07	115821.42	Male	White
19	Other	ASSISTANT DIRECTOR (EXECUTIVE LEVEL)	2002-05-28	95783.00	Female	Hispanic
39	Houston Airport System	DEPUTY ASSISTANT DIRECTOR (EXECUTIVE LEV)	2017-08-15	112270.00	Male	Black
48	Fire	ASSISTANT FIRE CHIEF	1994-11-07	115835.98	Male	Hispanic

```
[61]: high_sal.groupby('title').agg(avg_salary=('salary', 'mean')).min()
```

```
[61]: avg_salary    100038.0
      dtype: float64
```

Exercise 3

Filter the employee dataset so that only position titles with at least 5 employees and an average salary of \$80,000 remain. Can you verify the results?

```
[62]: def sal_count(sub_df):
        return sub_df['salary'].mean() > 80000 and len(sub_df) >= 5
```

```
[63]: high_sal_count = emp.groupby('title').filter(sal_count)
high_sal_count.head()
```

	dept	title	hire_date	salary	sex	race
0	Police	POLICE SERGEANT	2001-12-03	87545.38	Male	White
1	Other	ASSISTANT CITY ATTORNEY II	2010-11-15	82182.00	Male	Hispanic
16	Other	ASSOCIATE JUDGE OF MUNICIPAL COURTS	2005-11-09	107744.00	Male	Hispanic
17	Police	POLICE COMMANDER	1983-02-07	115821.42	Male	White
19	Other	ASSISTANT DIRECTOR (EXECUTIVE LEVEL)	2002-05-28	95783.00	Female	Hispanic

```
[64]: high_sal_count.groupby('title').agg(avg_salary=('salary', 'mean'),
                                         size=('salary', 'size')).min()
```

```
[64]: avg_salary    80153.202222
      size         5.000000
      dtype: float64
```

Exercise 4

Add a new column, `pct_max_dept_sex`, to the employee DataFrame that holds the employees percentage of the maximum salary for each department and race. For instance, if a male HPD employee makes 80,000 and the maximum male HPD salary is 120,000 then the value for this employee would be 80,000/120,000 or .666. Verify this value for the first employee.

```
[65]: def pct_max(sub_series):
        return sub_series / sub_series.max()
```

```
[66]: emp['pct_max_dept_sex'] = emp.groupby(['dept', 'sex']).transform(pct_max)
emp.head()
```

	dept	title	hire_date	salary	sex	race	pct_max_dept_sex
0	Police	POLICE SERGEANT	2001-12-03	87545.38	Male	White	0.312662
1	Other	ASSISTANT CITY ATTORNEY II	2010-11-15	82182.00	Male	Hispanic	0.298844
2	Houston Public Works	SENIOR SLUDGE PROCESSOR	2006-01-09	49275.00	Male	Black	0.227809
3	Police	SENIOR POLICE OFFICER	1997-05-27	75942.10	Male	Hispanic	0.271222
4	Police	SENIOR POLICE OFFICER	2006-01-23	69355.26	Male	White	0.247697

```
[67]: filt = (emp['dept'] == 'Police') & (emp['sex'] == 'Male')
max_sal = emp.loc[filt, 'salary'].max()
max_sal
```

```
[67]: 280000.0
```

```
[68]: emp.loc[0, 'salary'] / max_sal
```

```
[68]: 0.31266207142857144
```


Part VII

Time Series

Chapter 7

Solutions

7.1 1. Datetime and Timedelta

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

Exercise 1

What day of the week was Jan 15, 1997?

```
[2]: dt = pd.to_datetime('Jan 15, 1997')
dt.day_name()
```

```
[2]: 'Wednesday'
```

Exercise 2

Was 1925 a leap year?

```
[3]: dt = pd.to_datetime('Jan 1, 1924')
dt.is_leap_year
```

```
[3]: True
```

Exercise 3

What year will it be 1 million hours after the UNIX epoch?

```
[4]: dt = pd.to_datetime(10 ** 6, unit='h')
dt
```

```
[4]: Timestamp('2084-01-29 16:00:00')
```

```
[5]: dt.year
```

```
[5]: 2084
```

Exercise 4

Create the datetime July 20, 1969 at 2:56 a.m. and 15 seconds.

```
[6]: dt = pd.to_datetime('1969-07-20 2:56:15')
dt
```

```
[6]: Timestamp('1969-07-20 02:56:15')
```

Exercise 5

Neil Armstrong stepped on the moon at the time in the last Exercise. How many days have passed since that happened? Use the string ‘today’ when creating your datetime.

```
[7]: dt1 = pd.to_datetime('1969-07-20 2:56:15')
dt2 = pd.to_datetime('today')
dt2
```

```
[7]: Timestamp('2019-11-01 12:13:47.366957')
```

```
[8]: td = dt2 - dt1
td
```

```
[8]: Timedelta('18366 days 09:17:32.366957')
```

```
[9]: td.days
```

```
[9]: 18366
```

Exercise 6

Which is larger - 35 days or 700 hours?

```
[10]: td1 = pd.to_timedelta(35, unit='d')
td2 = pd.to_timedelta(700, unit='h')
```

```
[11]: td1
```

```
[11]: Timedelta('35 days 00:00:00')
```

```
[12]: td2
```

```
[12]: Timedelta('29 days 04:00:00')
```

```
[13]: td1 > td2
```

```
[13]: True
```

Can also use a string for hours

```
[14]: pd.to_timedelta('700 hours')
```

```
[14]: Timedelta('29 days 04:00:00')
```

```
[15]: pd.to_timedelta('700h')
```

```
[15]: Timedelta('29 days 04:00:00')
```

Exercise 7

In a previous notebook, we were told that the employee data was retrieved on Dec 1, 2016. We used the simple calculation `2016 - emp['hire_date'].dt.year` to determine the years of experience. Can you improve upon this method to get the exact amount of years of experience and assign as a new column named `experience`?

```
[16]: emp = pd.read_csv('../data/employee.csv', parse_dates=['hire_date'])
```

```
[17]: pull_date = pd.to_datetime('2016-1-1')
one_year = pd.to_timedelta(1, unit='Y')
```

```
/Users/Ted/miniconda3/envs/minimal_ds/lib/python3.7/site-
packages/pandas/util/_decorators.py:208: FutureWarning: M and Y units are deprecated and
will be removed in a future version.

    return func(*args, **kwargs)
/Users/Ted/miniconda3/envs/minimal_ds/lib/python3.7/site-
packages/pandas/core/tools/timedeltas.py:134: FutureWarning: M and Y units are deprecated
and will be removed in a future version.

    result = Timedelta(r, unit)
```

```
[18]: pull_date
```

```
[18]: Timestamp('2016-01-01 00:00:00')
```

```
[19]: one_year
```

```
[19]: Timedelta('365 days 05:49:12')
```

```
[20]: emp['experience'] = (pull_date - emp['hire_date']) / one_year
emp.head()
```

	dept	title	hire_date	salary	sex	race	experience
0	Police	POLICE SERGEANT	2001-12-03	87545.38	Male	White	14.078318
1	Other	ASSISTANT CITY ATTORNEY II	2010-11-15	82182.00	Male	Hispanic	5.128100
2	Houston Public Works	SENIOR SLUDGE PROCESSOR	2006-01-09	49275.00	Male	Black	9.976933
3	Police	SENIOR POLICE OFFICER	1997-05-27	75942.10	Male	Hispanic	18.598602
4	Police	SENIOR POLICE OFFICER	2006-01-23	69355.26	Male	White	9.938602

7.2 2. Introduction to Time Series

Exercise 1

Read in the weather time series dataset and place the date column in the index.

```
[21]: weather = pd.read_csv('../data/weather.csv', parse_dates=['date'], index_col='date')
weather.head()
```

		rain	snow	temperature
	date			
	2007-01-01	Yes	No	68.0
	2007-01-02	No	No	55.9
	2007-01-03	No	No	62.1
	2007-01-04	No	No	69.1
	2007-01-05	Yes	No	72.0

Exercise 2

What was the temperature on June 11, 2011?

```
[22]: weather.loc['2011-6-11', 'temperature']
```

[22]: 93.9

Exercise 3

How many days did it rain during the last three months of 2011?

```
[23]: weather.loc['2011-10':'2011-12', 'rain'].value_counts()
```

[23]: No 69
Yes 23
Name: rain, dtype: int64

Exercise 4

Which year had more snow days, 2007 or 2012?

```
[24]: weather.loc['2007', 'snow'].value_counts()
```

[24]: No 360
Yes 5
Name: snow, dtype: int64

```
[25]: weather.loc['2012', 'snow'].value_counts()
```

[25]: No 364
Yes 2
Name: snow, dtype: int64

Exercise 5

Select every other thursday

```
[26]: weather.asfreq('2W-THU').head()
```

		rain	snow	temperature
	date			
	2007-01-04	No	No	69.1
	2007-01-18	Yes	Yes	35.1
	2007-02-01	Yes	Yes	34.0
	2007-02-15	No	No	39.9
	2007-03-01	Yes	No	66.9

Exercise 6

Select the first day of each month.

```
[27]: weather.asfreq('MS').head()
```

		rain	snow	temperature
	date			
	2007-01-01	Yes	No	68.0
	2007-02-01	Yes	Yes	34.0
	2007-03-01	Yes	No	66.9
	2007-04-01	Yes	No	77.0
	2007-05-01	No	No	91.9

```
[28]: weather.asfreq('4M')
```

	rain	snow	temperature
date			
2007-01-31	No	No	39.9
2007-05-31	No	No	91.0
2007-09-30	No	No	80.1
2008-01-31	No	No	51.1
2008-05-31	No	No	89.1
2008-09-30	Yes	No	81.0
2009-01-31	No	No	46.9
2009-05-31	Yes	No	88.0
2009-09-30	No	No	75.0
2010-01-31	Yes	Yes	37.9
2010-05-31	Yes	No	88.0
2010-09-30	Yes	No	77.0
2011-01-31	No	No	46.9
2011-05-31	No	No	95.0
2011-09-30	Yes	No	84.0
...
2014-01-31	No	No	53.1
2014-05-31	No	No	82.0
2014-09-30	No	No	75.0
2015-01-31	No	No	48.0
2015-05-31	No	No	89.1
2015-09-30	No	No	82.9
2016-01-31	No	No	70.0
2016-05-31	No	No	84.9
2016-09-30	No	No	82.9
2017-01-31	No	No	68.0
2017-05-31	No	No	86.0
2017-09-30	No	No	77.0
2018-01-31	No	No	46.9
2018-05-31	No	No	88.0
2018-09-30	No	No	81.0

7.3 3. Grouping by Time

```
[29]: import pandas as pd
msft = pd.read_csv('../data/stocks/msft20.csv', parse_dates=['date'], index_col='date')
msft.head(3)
```

	open	high	low	close	adjusted_close	volume	dividend_amount
date							
1999-10-19	88.250	89.250	85.25	86.313	27.8594	69945600	0.0
1999-10-20	91.563	92.375	90.25	92.250	29.7758	88090600	0.0
1999-10-21	90.563	93.125	90.50	93.063	30.0381	60801200	0.0

Exercise 1

In which week did MSFT have the greatest number of its shares (volume) traded?

```
[30]: volume = msft.resample('W').agg({'volume':'sum'})
volume.head()
```

	volume
date	
1999-10-24	262488000
1999-10-31	287399400
1999-11-07	268534000
1999-11-14	541663200
1999-11-21	295783800

```
[31]: volume.agg(['max', 'idxmax'])
```

	volume
max	879723200
idxmax	2006-05-07 00:00:00

Pandas Trick

Turn into a period to get Monday-Sunday date range

```
[32]: msft.resample('W', kind='period').agg({'volume':'sum'}).agg(['max', 'idxmax'])
```

	volume
max	879723200
idxmax	2006-05-01/2006-05-07

Exercise 2

With help from the `diff` method, find the quarter containing the most number of up days.

Use `diff` to find the difference between the current row and the one directly above it.

```
[33]: msft['adjusted_close'].diff().head()
```

```
[33]: date
1999-10-19      NaN
1999-10-20    1.9164
1999-10-21    0.2623
1999-10-22   -0.1210
1999-10-25   -0.0807
Name: adjusted_close, dtype: float64
```

```
[34]: up_days = msft['adjusted_close'].diff() > 0
up_days.head()
```

```
[34]: date
1999-10-19    False
1999-10-20     True
1999-10-21     True
1999-10-22    False
1999-10-25    False
Name: adjusted_close, dtype: bool
```

```
[35]: up_days.resample('Q').sum().head()
```

```
[35]: date
1999-12-31    24.0
2000-03-31    32.0
2000-06-30    30.0
2000-09-30    23.0
2000-12-31    29.0
Freq: Q-DEC, Name: adjusted_close, dtype: float64
```

```
[36]: up_days.resample('Q').sum().agg(['max', 'idxmax'])
```

```
[36]: max            43
idxmax    2001-12-31 00:00:00
Name: adjusted_close, dtype: object
```

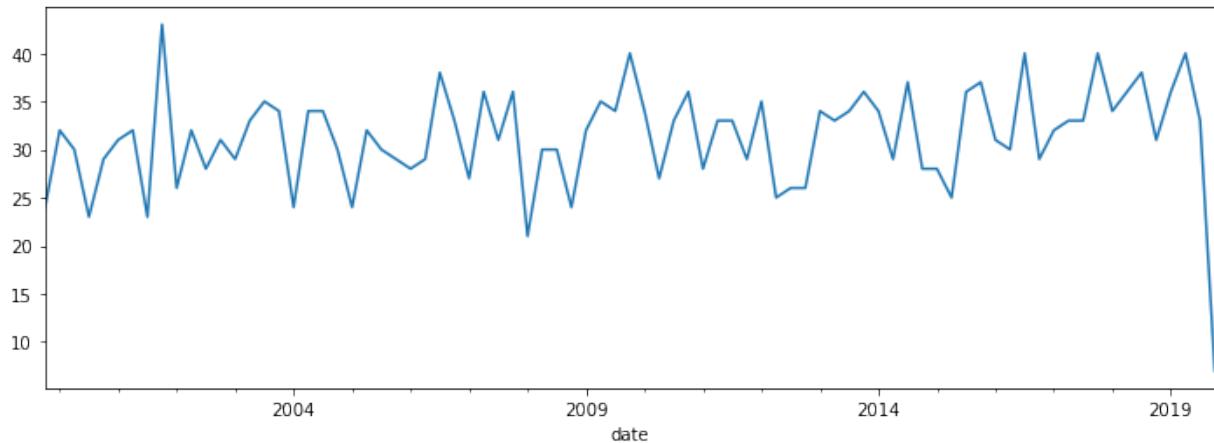
Pandas trick

```
[37]: import matplotlib.pyplot as plt
%matplotlib inline
```

Can visually verify with plot:

```
[38]: up_days.resample('Q', kind='period').sum().plot(figsize=(12,4))
```

```
[38]: <matplotlib.axes._subplots.AxesSubplot at 0x1175b5e10>
```



Exercise 3

Find the mean price per year along with the minimum and maximum volume.

```
[39]: msft.resample('Y', kind='period').agg({'adjusted_close': 'mean',
                                             'volume':['min', 'max']}).head()
```

	date	adjusted_close		volume	
		mean	min	max	
	1999	31.267802	12517600	243819200	
	2000	24.601943	15734800	313645800	
	2001	20.186950	11701600	209348800	
	2002	17.606981	18386000	202307800	
	2003	16.917920	12076900	210558300	

Execute the cell below to read in the employee dataset and use it for the rest of the exercises.

```
[40]: emp = pd.read_csv('../data/employee.csv')
```

Exercise 4

Use the `to_datetime` function to convert the hire date column into datetimes. Reassign this column in the `emp` DataFrame.

```
[41]: emp['hire_date'] = pd.to_datetime(emp['hire_date'])
```

Exercise 5

Without putting `hire_date` into the index, find the mean salary based on `hire_date` over 5 year periods. Also return the number of salaries used in the mean calculation for each period.

```
[42]: emp.resample('5Y', on='hire_date').agg({'salary': ['mean', 'count']})
```

salary		
	mean	count
hire_date		
1968-12-31	NaN	0
1973-12-31	64574.640000	4
1978-12-31	78074.008667	30
1983-12-31	73504.712738	336
1988-12-31	70655.520653	613
1993-12-31	69255.759681	1946
1998-12-31	67875.931733	2424
2003-12-31	63357.963896	2618
2008-12-31	60447.709702	4329
2013-12-31	56581.778488	3453
2018-12-31	48230.686634	7609

7.4 4. Rolling Windows

Exercise 1

Attempt to take a rolling average on salary using a 30 day time span on hire date. Does the error message make sense?

```
[43]: emp = pd.read_csv('../data/employee.csv', parse_dates=['hire_date'])
```

```
[44]: emp.rolling('30D', on='hire_date')
```

`ValueError: hire_date must be monotonic`

Yes, the error message makes sense. If you are going to do a rolling average by an amount of time, then the dates need to be sorted. Monotonic means all increasing or all decreasing.

Exercise 2

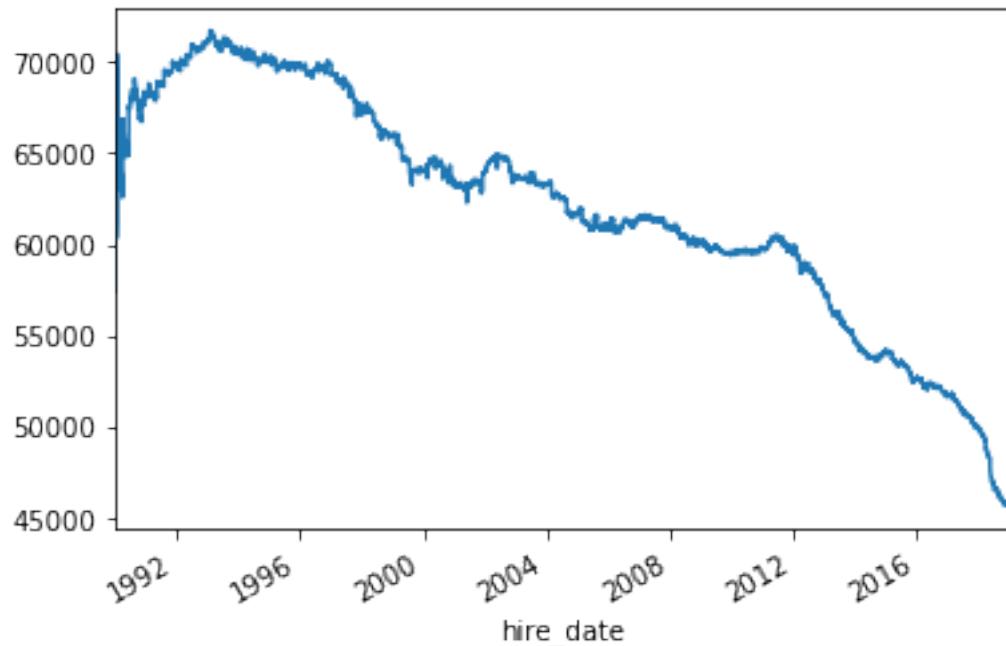
Set hire date as the index and then select the salary column as a Series. Sort the Series by date and drop the missing values. Now select a subset that only has hire dates from 1990 onwards. Then find a 1,000 day rolling average. Finally make a call to the `plot` method. Make sure you inline matplotlib if you did not do it earlier.

```
[45]: %matplotlib inline
```

```
[46]: sal = emp.set_index('hire_date')['salary'].sort_index().dropna()
```

```
[47]: sal['1990':].rolling('1000D').mean().plot()
```

[47]: <matplotlib.axes._subplots.AxesSubplot at 0x116975128>



Exercise 3

Read in the energy consumption dataset. Select just the residential source and plot a 12 month trailing rolling mean of the energy.

```
[48]: energy = pd.read_csv('../data/energy_consumption.csv', parse_dates=['date'],
    ↪index_col='date')
energy.head()
```

	source	energy (btu)
	date	
1973-01-01	residential	1932.187
1973-02-01	residential	1687.255
1973-03-01	residential	1497.067
1973-04-01	residential	1177.661
1973-05-01	residential	1015.008

```
[49]: filt = energy['source'] == 'residential'
res_energy = energy.loc[filt, 'energy (btu)']
res_energy.head()
```

```
[49]: date
1973-01-01    1932.187
1973-02-01    1687.255
1973-03-01    1497.067
1973-04-01    1177.661
1973-05-01    1015.008
```

```
Name: energy (btu), dtype: float64
```

If you try to use an offset alias here, you will get an error because Pandas does not consider a year to be a fixed frequency.

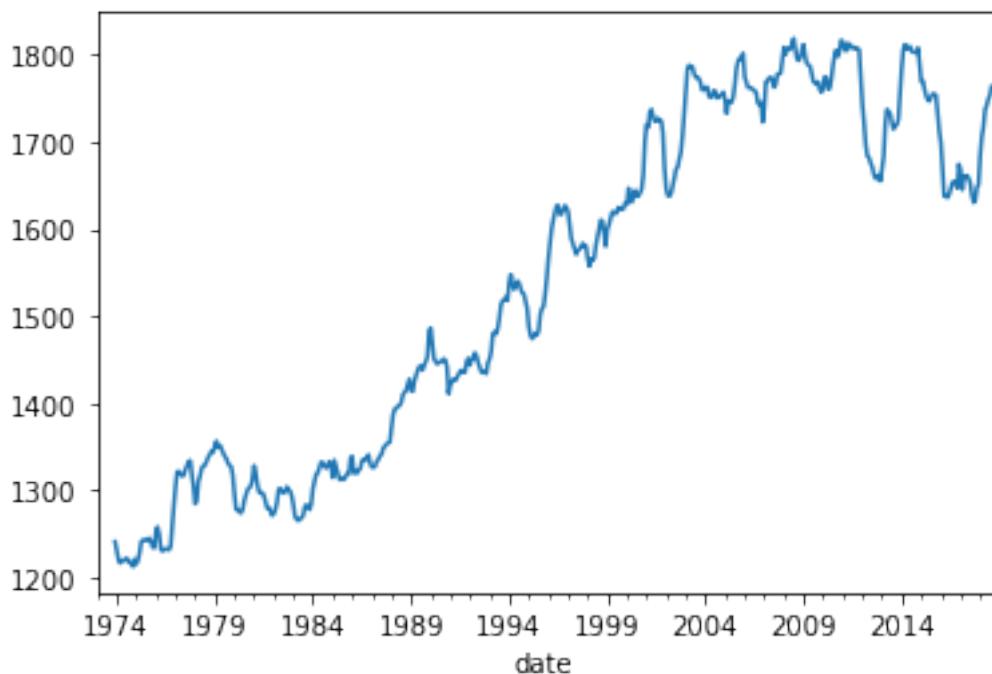
```
[50]: res_energy.rolling('Y').mean()
```

```
ValueError: <YearEnd: month=12> is a non-fixed frequency
```

Instead, you can use the integer 12 (assuming you check that there is data for each month).

```
[51]: res_energy.rolling(12).mean().plot()
```

```
[51]: <matplotlib.axes._subplots.AxesSubplot at 0x116a13fd0>
```



7.5 5. Grouping by Time and another Column

```
[52]: import numpy as np
```

Exercise 1

Read in the energy consumption dataset. Find the average energy consumption per sector per 10 year time span beginning from the first year of data. Return the results as both a groupby and a pivot table. Experiment with adding 'S' to the end of your offset alias. How does this change the results?

```
[53]: energy = pd.read_csv('../data/energy_consumption.csv', parse_dates=['date'],
   ↴index_col='date')
energy.head()
```

	source	energy (btu)
	date	
1973-01-01	residential	1932.187
1973-02-01	residential	1687.255
1973-03-01	residential	1497.067
1973-04-01	residential	1177.661
1973-05-01	residential	1015.008

By default, the offset alias ‘Y’ represents the year end. So the first date for each source is the time span of Jan 1, 1964 - Dec 31, 1973. It is instructive to also find the size of each group to determine how many rows were aggregated.

```
[54]: tg = pd.Grouper(freq='10Y')
energy.groupby(['source', tg]).agg({'energy (btu)':['sum', np.size]}).astype('int')
```

		energy (btu)	
		sum	size
source	date		
commercial	1973-12-31	9544	12
	1983-12-31	103325	120
	1993-12-31	126317	120
	2003-12-31	161011	120
	2013-12-31	179146	120
	2023-12-31	84850	56
industrial	1973-12-31	32622	12
	1983-12-31	309268	120
	1993-12-31	306423	120
	2003-12-31	339772	120
	2013-12-31	314844	120
	2023-12-31	148435	56
residential	1973-12-31	14895	12
	1983-12-31	154427	120
	1993-12-31	169077	120
	2003-12-31	195951	120
	2013-12-31	211653	120
	2023-12-31	96460	56
transportation	1973-12-31	18612	12
	1983-12-31	193850	120
	1993-12-31	216420	120
	2003-12-31	253859	120
	2013-12-31	274295	120
	2023-12-31	129451	56

Appending an ‘S’ to the offset alias string makes pandas treat the first observation as the beginning of the time period. The first group is from Jan 1, 1973 to Dec 31, 1982 and this is likely what you would want.

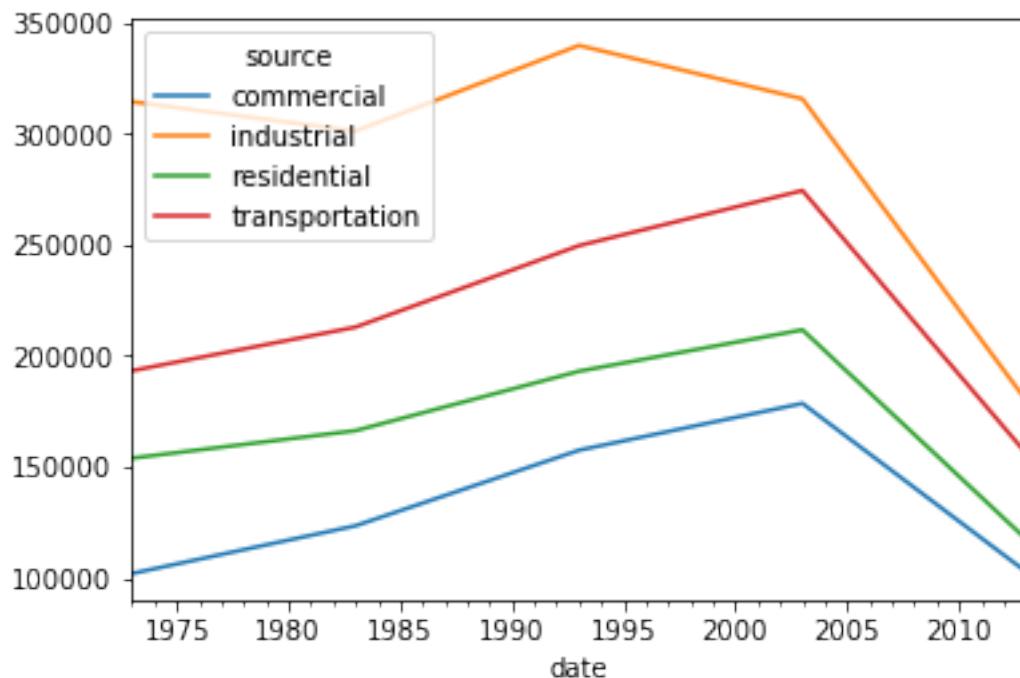
```
[55]: tg = pd.Grouper(freq='10YS')
energy.groupby(['source', tg]).agg({'energy (btu)': ['sum', np.size]}).astype('int')
```

		energy (btu)		
		sum	size	
source	date			
commercial	1973-01-01	101931	120	
	1983-01-01	123438	120	
	1993-01-01	157480	120	
	2003-01-01	178566	120	
	2013-01-01	102779	68	
industrial	1973-01-01	314461	120	
	1983-01-01	301236	120	
	1993-01-01	339833	120	
	2003-01-01	315776	120	
	2013-01-01	180058	68	
residential	1973-01-01	153900	120	
	1983-01-01	166286	120	
	1993-01-01	193051	120	
	2003-01-01	211705	120	
	2013-01-01	117522	68	
transportation	1973-01-01	193286	120	
	1983-01-01	212926	120	
	1993-01-01	249648	120	
	2003-01-01	274455	120	
	2013-01-01	156173	68	

```
[56]: df = energy.pivot_table(index=tg, columns='source',
                           values='energy (btu)', aggfunc='sum').astype('int')
df
```

source	commercial	industrial	residential	transportation
date				
1973-01-01	101931	314461	153900	193286
1983-01-01	123438	301236	166286	212926
1993-01-01	157480	339833	193051	249648
2003-01-01	178566	315776	211705	274455
2013-01-01	102779	180058	117522	156173

```
[57]: df.plot();
```



Part VIII

Regular Expressions

Chapter 8

Solutions

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

8.1 1. Introduction to Regular Expressions

```
[2]: movie = pd.read_csv('../data/movie.csv')  
title = movie['title']
```

Exercise 1

Find all movies that have 2 consecutive z's in them.

```
[3]: filt = title.str.contains('zz')  
title[filt]
```

```
[3]: 416          All That Jazz  
907          The Dukes of Hazzard  
1041         Bedazzled  
2234         Paparazzi  
2524         Hot Fuzz  
2593    The Lizzie McGuire Movie  
3215    Into the Grizzly Maze  
3535        Mystic Pizza  
4399        Blue Like Jazz  
Name: title, dtype: object
```

Exercise 2

Find all movies that begin with 9.

```
[4]: filt = title.str.contains('^9')  
title[filt]
```

```
[4]: 1651          9  
2416        9½ Weeks  
3705    90 Minutes in Heaven
```

```
Name: title, dtype: object
```

Exercise 3

Find all movies that have a b as their third character.

```
[5]: filt = title.str.contains('^..b')
title[filt].head()
```

```
[5]: 22           Robin Hood
  228          RoboCop
   286      Public Enemies
   448          Robots
  494 Babe: Pig in the City
Name: title, dtype: object
```

Exercise 4

Find all movies with a fourth-to-last character of M and a last character of e.

```
[6]: filt = title.str.contains('M..e$')
title[filt].head()
```

```
[6]: 704      The Green Mile
  1167        8 Mile
   1616      Like Mike
   2122  Moonlight Mile
   2486    How She Move
Name: title, dtype: object
```

Exercise 5

Could you use a regular expression to find a movie that was exactly 6 characters in length?

```
[7]: filt = title.str.contains('^.....$')
title[filt].head(10)
```

```
[7]: 0      Avatar
  41    Cars 2
   58    WALL·E
  125    Frozen
  168    Sahara
  292    Eraser
  298    Eragon
  368    Pixels
  426    Jumper
  428    Zodiac
Name: title, dtype: object
```

Exercise 6

What is a more natural way to complete Exercise 5 without a regex?

```
[8]: filt = title.str.len() == 6
title[filt].head(10)
```

```
[8]: 0      Avatar
41     Cars 2
58     WALL·E
125    Frozen
168    Sahara
292    Eraser
298    Eragon
368    Pixels
426    Jumper
428    Zodiac
Name: title, dtype: object
```

8.2 2. Quantifiers

Read in the movie dataset first.

```
[9]: movie = pd.read_csv('../data/movie.csv')
title = movie['title']
```

Exercise 1

Find all movies that have a ‘z’ as their 15th character.

```
[10]: pattern = '^.{14}z'
filt = title.str.contains(pattern)
title[filt]
```

```
[10]: 2484      American Dreamz
2625      Ramona and Beezus
Name: title, dtype: object
```

Exercise 2

Find all movies that have the word ‘Friend’ or ‘Friends’ in them.

```
[11]: pattern = 'Friends?'
filt = title.str.contains(pattern)
title[filt]
```

```
[11]: 1055          My Best Friend's Wedding
1413          Friends with Benefits
1775          How to Lose Friends & Alienate People
2216          My Best Friend's Girl
3116          Seeking a Friend for the End of the World
3495          Friends with Money
4184          We Are Your Friends
4279          Dysfunctional Friends
4670          Mutual Friends
Name: title, dtype: object
```

Exercise 3

Find all movies that have between 40 and 43 characters in them. Can you verify the results with another `str` accessor method?

```
[12]: pattern = '^.{40,43}$'
filt = title.str.contains(pattern)
m40_43 = title[filt]
m40_43.head()
```

```
[12]: 1      Pirates of the Caribbean: At World's End
       4      Star Wars: Episode VII - The Force Awakens
      13      Pirates of the Caribbean: Dead Man's Chest
      16      The Chronicles of Narnia: Prince Caspian
      18      Pirates of the Caribbean: On Stranger Tides
Name: title, dtype: object
```

```
[13]: m40_43.str.len().head()
```

```
[13]: 1      40
       4      42
      13      42
      16      40
      18      43
Name: title, dtype: int64
```

```
[14]: m40_43.str.len().value_counts()
```

```
[14]: 40      15
       41      12
       43       9
       42       9
Name: title, dtype: int64
```

Exercise 4

Find all movies that begin with ‘The’ and end in ‘Movie’

```
[15]: pattern = '^The.*Movie$'
filt = title.str.contains(pattern)
title[filt]
```

```
[15]: 319                  The Peanuts Movie
      561                  The Angry Birds Movie
      569                  The Simpsons Movie
      759                  The Lego Movie
     1586                 The SpongeBob SquarePants Movie
     1734                 The Rugrats Movie
     1895                 The Wild Thornberrys Movie
     2162                 The Tigger Movie
     2593                 The Lizzie McGuire Movie
    2645 The Pirates Who Don't Do Anything: A VeggieTal...
     3296                 The Muppet Movie
```

```
4597                               The Kentucky Fried Movie
Name: title, dtype: object
```

Exercise 5

Create your own Series and make a regular expression that uses the + metacharacter. Is this character necessary?

[]:

8.3 3. Or Conditions

```
[16]: import pandas as pd
title = pd.read_csv('../data/movie.csv')['title']
title.head()
```

```
[16]: 0          Avatar
      1    Pirates of the Caribbean: At World's End
      2          Spectre
      3    The Dark Knight Rises
      4  Star Wars: Episode VII - The Force Awakens
Name: title, dtype: object
```

Exercise 1

Find all movies that begin with ‘The’ followed by the next word that begins with digits.

```
[17]: pattern = '^The [0-9]'
filt = title.str.contains(pattern)
title[filt]
```

```
[17]: 212          The 13th Warrior
      429          The 6th Day
      1354         The 5th Wave
      1817        The 40-Year-Old Virgin
      1958          The 33
      3567        The 5th Quarter
      4373  The 41-Year-Old Virgin Who Knocked Up Sarah Ma...
Name: title, dtype: object
```

Exercise 2

Find all movies that have three consecutive capital letters in them.

```
[18]: pattern = '[A-Z]{3}'
filt = title.str.contains(pattern)
title[filt].head()
```

```
[18]: 4      Star Wars: Episode VII - The Force Awakens
      40          TRON: Legacy
      58          WALL·E
```

```
140           Mission: Impossible III
177           The BFG
Name: title, dtype: object
```

Exercise 3

Find all movies that have begin and end with a capital letter.

```
[19]: pattern = '^[A-Z].*[A-Z]$'
filt = title.str.contains(pattern)
title[filt].head()
```

```
[19]: 46           World War Z
      58           WALL·E
     140          Mission: Impossible III
    151          Men in Black II
    177          The BFG
Name: title, dtype: object
```

Exercise 4

Find all the movies that have a digit followed by a comma followed by a digit.

```
[20]: pattern = r'[0-9],[0-9]'
filt = title.str.contains(pattern)
title[filt].head()
```

```
[20]: 276          10,000 B.C.
3266         Ultramarines: A Warhammer 40,000 Movie
3641         20,000 Leagues Under the Sea
4775         The Beast from 20,000 Fathoms
Name: title, dtype: object
```

Exercise 5

Find all the movies that have either an ampersand or a question mark in them.

```
[21]: pattern = '[&?]'
filt = title.str.contains(pattern)
title[filt].head()
```

```
[21]: 129          Angels & Demons
145          Mr. Peabody & Sherman
214          Batman & Robin
252          Mr. & Mrs. Smith
278          Town & Country
Name: title, dtype: object
```

Exercise 6

Which movie has the most ampersands, question marks, and periods in it?

```
[22]: pattern = '[&.?]'
count = title.str.count(pattern)
count.head()
```

```
[22]: 0    0
      1    0
      2    0
      3    0
      4    0
Name: title, dtype: int64
```

```
[23]: filt = count == count.max()
title[filt]
```

```
[23]: 542    The Man from U.N.C.L.E.
Name: title, dtype: object
```

```
[24]: count.max()
```

```
[24]: 5
```

8.4 4. Character Sets and Grouping

Exercise 1

For all movies that begin with ‘The’ and are followed by the next word that begins with a digit, extract just the digits part of this word.

```
[25]: pattern = r'^The (\d+)'
title.str.extract(pattern).dropna()
```

0
212 13
429 6
1354 5
1817 40
1958 33
3567 5
4373 41

Exercise 2

Find all movies that have two separate numbers in them. An example would be, ‘7 days and 7 nights’.

```
[26]: pattern = r'\d+\D+\d+'
filt = title.str.contains(pattern)
title[filt]
```

```
[26]: 276           10,000 B.C.
      289           The Taking of Pelham 1 2 3
      509           2 Fast 2 Furious
     1043           3:10 to Yuma
     1610           13 Going on 30
    1617   Naked Gun 33 1/3: The Final Insult
    2466           40 Days and 40 Nights
    2646           U2 3D
    3266 Ultramarines: A Warhammer 40,000 Movie
    3308           50/50
    3516           Fahrenheit 9/11
    3576           11:14
    3641 20,000 Leagues Under the Sea
    3934           2:13
    4210 24 7: Twenty Four Seven
    4376 Friday the 13th Part 2
    4532 4 Months, 3 Weeks and 2 Days
    4775 The Beast from 20,000 Fathoms
Name: title, dtype: object
```

Exercise 3

Find all the movies that have 6 or more non-vowel and non-space characters in a row.

```
[27]: pattern = r'[^aeiouAEIOU ]{6,}'
filt = title.str.contains(pattern)
title[filt]
```

```
[27]: 276           10,000 B.C.
      542           The Man from U.N.C.L.E.
     1935           Punch-Drunk Love
     2392           Catch-22
     2480           Brooklyn's Finest
     2507 When Harry Met Sally...
    2912 Tales from the Crypt: Demon Knight
    3266 Ultramarines: A Warhammer 40,000 Movie
    3641 20,000 Leagues Under the Sea
    4775 The Beast from 20,000 Fathoms
Name: title, dtype: object
```

Exercise 4

Extract the very next character after ‘t’ or ‘T’ for each movie.

```
[28]: pattern = r'[Tt](.)'
title.str.extract(pattern).head()
```

0	
0	a
1	e
2	r
3	h
4	a

Exercise 5

What is the most common character after ‘t’ or ‘T’?

```
[29]: pattern = r'[Tt](.)'
letters = title.str.extract(pattern)
letters.head()
```

0	
0	a
1	e
2	r
3	h
4	a

This is a DataFrame. The column name is the integer 0. Let’s select it as a Series.

```
[30]: letter_series = letters[0]
letter_series.head()
```

```
[30]: 0    a
      1    e
      2    r
      3    h
      4    a
Name: 0, dtype: object
```

```
[31]: letter_series.value_counts().head()
```

```
[31]: h    1431
      311
      e    266
      o    183
      i    169
Name: 0, dtype: int64
```

Minor detail here - there is an `expand` parameter than you can set to `False` to return a Series.

```
[32]: pattern = r'[Tt](.)'
letters = title.str.extract(pattern, expand=False)
letters.head()
```

```
[32]: 0      a
      1      e
      2      r
      3      h
      4      a
Name: title, dtype: object
```

```
[33]: letters.value_counts().head()
```

```
[33]: h     1431
      311
      e     266
      o     183
      i     169
Name: title, dtype: int64
```

The above only extracts the character after first appearance of the letter ‘t’. Use the `extractall` string method to get the first characters after each ‘t’.

```
[34]: pattern = r'[Tt](.)'
letters = title.str.extractall(pattern)
letters.head()
```

0		
match		
0	0	a
1	0	e
	1	h
	2	
2	0	r

```
[35]: letters[0].value_counts().head()
```

```
[35]: h     1942
      620
      e     480
      o     354
      i     343
Name: 0, dtype: int64
```

Exercise 6

Extract all the words that begin with ‘T’ or ‘t’ and end in ‘e’ then find their frequency. Research the word boundary special character.

```
[36]: pattern = r'\b([tT]\w*e)\b'  
letters = title.str.extractall(pattern)  
letters[0].str.lower().value_counts()
```

```
[36]: the          1555  
time          26  
tale          12  
true          10  
three          8  
teenage         7  
there          6  
take           6  
trade           4  
trouble          4  
twice           2  
treasure          2  
terrace          2  
timeline          1  
transcendence        1  
triangle          1  
tae             1  
terrible          1  
turbulence         1  
turtle            1  
tree             1  
trance            1  
tease             1  
tide              1  
thunderdome        1  
torture            1  
tape              1  
tadpole            1  
thr3e             1  
tombstone          1  
twelve            1  
triple             1  
temple             1  
torque             1  
throttle            1  
tootsie            1  
trapeze            1  
Name: 0, dtype: int64
```

8.5 Project - Explore Newsgroups with Regexes

```
[37]: news = pd.read_csv('..../data/newsgroups.csv')  
news.head()
```

	category	text
0	sci.med	From: nyeda@cnsvax.uwec.edu (David Nye)\nSubje...
1	talk.politics.guns	From: ndallen@r-node.hub.org (Nigel Allen)\nSu...
2	misc.forsale	From: mark@ardsley.business.uwo.ca (Mark Bramw...
3	misc.forsale	From: zmed16@trc.amoco.com (Michael)\nSubject:...
4	talk.politics.guns	From: fcrary@ucsu.Colorado.EDU (Frank Crary)\n...

Extracting emails

It appears all emails follow the line in the header that begins with ‘From:’. The following captures emails as the sequence of characters that do not have a space, parentheses or greater than or less than signs, or line breaks in them. There must also be an at symbol in the sequence.

```
[38]: pattern = r'\bFrom:.*?([^\s()<]+@[^\s()]+)'
emails = news['text'].str.extract(pattern)
emails.head()
```

	0
0	nyeda@cnsvax.uwec.edu
1	ndallen@r-node.hub.org
2	mark@ardsley.business.uwo.ca
3	zmed16@trc.amoco.com
4	fcrary@ucsu.Colorado.EDU

Extracting the header

It appears that the header begins at the start of the email and continues until it hits an empty line. The following matches all characters (including line breaks) up until two line breaks in a row. This should represent the header. The pattern `[\s\S]*?` represents all characters. The dot special character does not match line breaks.

The `*?` represents a non-greedy match, meaning the pattern will stop after the first match. If the question mark was absent, then it would match until the last two line breaks in a row. That’s called **greedy**.

```
[39]: headers = news['text'].str.extract(r'([\s\S]*?)\n\n')
headers.head()
```

	0
0	From: nyeda@cnsvax.uwec.edu (David Nye)\nSubje...
1	From: ndallen@r-node.hub.org (Nigel Allen)\nSu...
2	From: mark@ardsley.business.uwo.ca (Mark Bramw...
3	From: zmed16@trc.amoco.com (Michael)\nSubject:...
4	From: fcrary@ucsu.Colorado.EDU (Frank Crary)\n...

Example header

```
[40]: print(headers.loc[100, 0])
```

From: c23reg@kocrsv01.delcoelect.com (Ron Gaskins)
 Subject: Re: Dumbest automotive concepts of all time
 Originator: c23reg@koptsw21
 Keywords: Dimmer switch location (repost)
 Organization: Delco Electronics Corp.
 Lines: 22

Finding posts with quotes

The assumption here is that the line begins with a greater than symbol.

```
[41]: filt = news['text'].str.contains(r'\n>')
posts_with_quotes = news.loc[filt, 'text']
print(posts_with_quotes.values[0])
```

From: nyeda@cnsvax.uwec.edu (David Nye)
 Subject: Re: Post Polio Syndrome Information Needed Please !!!
 Organization: University of Wisconsin Eau Claire
 Lines: 21

[reply to keith@actrix.gen.nz (Keith Stewart)]

>My wife has become interested through an acquaintance in Post-Polio
 >Syndrome This apparently is not recognised in New Zealand and different
 >symptoms (eg chest complaints) are treated separately. Does anyone have
 >any information on it

It would help if you (and anyone else asking for medical information on some subject) could ask specific questions, as no one is likely to type in a textbook chapter covering all aspects of the subject. If you are looking for a comprehensive review, ask your local hospital librarian. Most are happy to help with a request of this sort.

Briefly, this is a condition in which patients who have significant residual weakness from childhood polio notice progression of the weakness as they get older. One theory is that the remaining motor neurons have to work harder and so die sooner.

David Nye (nyeda@cnsvax.uwec.edu). Midelfort Clinic, Eau Claire WI
 This is patently absurd; but whoever wishes to become a philosopher must learn not to be frightened by absurdities. -- Bertrand Russell

Counting words per category

We first put the category into the index and extract just the body of the posts (this excludes the header). This returns a DataFrame with a single column with name 0. We select this column in the second line.

```
[42]: body = news.set_index('category')['text'].str.extract(r'[\s\S]*?\n\n([\s\S]+)')
```

```
[42]: category
sci.med      [reply to keith@actrix.gen.nz (Keith Stewart)]...
talk.politics.guns  Here is a press release from the White House.\...
misc.forsale >\n>I hope you realize that for a cellular pho...
misc.forsale \nI have an Alesis HR-16 drum machine for sale...
talk.politics.guns In article <C4tsHu.Ew6@magpie.linknet.com> man...
Name: 0, dtype: object
```

Extract each individual non-quote line

We then use `extractall` to capture a pattern for each individual line. The assumption we make is that the line must begin with a word character.

```
[43]: body_lines = body_series.str.extractall(r'\n+(\w.*)')
```

```
body_lines.head(20)
```

			0
category	match		
sci.med	0	It would help if you (and anyone else asking f...	
	1	some subject) could ask specific questions, as...	
	2	in a textbook chapter covering all aspects of ...	
	3	looking for a comprehensive review, ask your l...	
	4	Most are happy to help with a request of this ...	
	5	Briefly, this is a condition in which patients...	
	6	residual weakness from childhood polio notice ...	
	7	weakness as they get older. One theory is tha...	
	8	neurons have to work harder and so die sooner.	
	9	David Nye (nyeda@cnsvax.uwec.edu). Midelfort ...	
	10	This is patently absurd; but whoever wishes to...	
	11	must learn not to be frightened by absurdities...	
talk.politics.guns	0	Clinton in a question and answer session with ...	
	1	1:36 P.M. EDT	
	2	agents were killed in the line of duty trying ...	
	3	against the Branch Davidian compound, which ha...	
	4	weaponry and ammunition, and placed innocent c...	
	5	Because the BATF operation had failed to meet ...	
	6	standoff ensued.	
	7	reasonable effort to bring this perilous situa...	

Split into individual words

We then split on any non-word character and use `expand=True` to put each word in its own column.

```
[44]: split_words = body_lines[0].str.split(r'\W+', expand=True)
split_words.head()
```

		0	1	2	3	4	...	32	33	34	35	36	
category	match												
sci.med	0	It	would	help	if	you	...	None	None	None	None	None	
	1	some	subject	could	ask	specific	...	None	None	None	None	None	
	2	in	a	textbook	chapter	covering	...	None	None	None	None	None	
	3	looking	for	a	comprehensive	review	...	None	None	None	None	None	
	4	Most	are	happy	to	help	...	None	None	None	None	None	
	5	briefly	this	is	a	condition	in	which	patients	...	None	None	
	6	residual	weakness	from	childhood	polio	noticed	...	None	None	None	None	
	7	weakness	as	they	get	older	One	theory	is	tha	...	None	
	8	neurons	have	to	work	harder	and	so	die	sooner	...	None	
	9	David	Nye	(nyeda@cnsvax.uwec.edu)	Midelfort	...	None	None	None	None	None	None	
	10	This	is	patently	absurd	but	whoever	wishes	to	...	None	None	
	11	must	learn	not	to	be	frightened	by	absurdities	...	None	None	
talk.politics.guns	0	Clinton	in	a	question	and	answer	session	with	...	None	None	
	1	1:36 P.M. EDT											
	2	agents	were	killed	in	the	line	of	duty	trying	...	None	None
	3	against	the	Branch	Davidian	compound	,	which	ha	...	None	None	None
	4	weaponry	and	ammunition	,	and	placed	innocent	c	...	None	None	None
	5	Because	the	BATF	operation	had	failed	to	meet	...	None	None	None
	6	standoff ensued.											
	7	reasonable	effort	to	bring	this	perilous	situa	...	None	None	None	None

Stack words into a single column

Use the stack method to put all the words in a single column. This will put the column names into the index. We also,

```
[45]: stacked_words = split_words.stack().str.lower()
stacked_words.head(30)
```

```
[45]: category    match
sci.med     0      0      it
           1      would
           2      help
           3      if
           4      you
           5      and
           6      anyone
           7      else
           8      asking
           9      for
          10      medical
          11      information
          12      on
1      0      some
           1      subject
           2      could
           3      ask
           4      specific
           5      questions
           6      as
           7      no
           8      one
           9      is
          10      likely
          11      to
          12      type
2      0      in
           1      a
           2      textbook
           3      chapter
dtype: object
```

Remove words less than 7 characters in length

These shorter words won't give us as much information about the topic as the longer ones.

```
[46]: long_word = stacked_words[stacked_words.str.len() >= 7]
long_word.head(20)
```

```
[46]: category    match
sci.med     0      10      medical
           11      information
1      1      subject
```

```

        4      specific
        5      questions
2       2      textbook
        3      chapter
        4      covering
        6      aspects
        9      subject
3       0      looking
        3      comprehensive
        8      hospital
        9      librarian
4       7      request
5       0      briefly
        4      condition
        7      patients
10      significant
6       0      residual
dtype: object

```

Groupby category and count the unique values

You can groupby an index level and the count the values for each group.

```
[47]: category_counts = long_word.groupby('category').value_counts().reset_index()
category_counts.columns = ['category', 'word', 'count']
category_counts.head(10)
```

	category	word	count
0	misc.forsale	condition	17
1	misc.forsale	excellent	11
2	misc.forsale	interested	9
3	misc.forsale	shipping	9
4	misc.forsale	windows	9
5	misc.forsale	printer	8
6	misc.forsale	publish	8
7	misc.forsale	contact	7
8	misc.forsale	software	7
9	misc.forsale	compatible	6

Select top 10 words per category

```
[48]: top10_words = category_counts.groupby('category').head(10)
top10_words.head(20)
```

	category	word	count
0	misc.forsale	condition	17
1	misc.forsale	excellent	11
2	misc.forsale	interested	9
3	misc.forsale	shipping	9
4	misc.forsale	windows	9
5	misc.forsale	printer	8
6	misc.forsale	publish	8
7	misc.forsale	contact	7
8	misc.forsale	software	7
9	misc.forsale	compatible	6
475	rec.autos	someone	13
476	rec.autos	problem	11
477	rec.autos	anything	9
478	rec.autos	without	9
479	rec.autos	driving	8
480	rec.autos	traffic	8
481	rec.autos	because	7
482	rec.autos	between	7
483	rec.autos	information	7
484	rec.autos	business	6

Fix the index

The index values are the old location of the rows. They don't make sense. Let's drop it.

```
[49]: top10_words = top10_words.reset_index(drop=True)
top10_words.head(20)
```

	category	word	count
0	misc.forsale	condition	17
1	misc.forsale	excellent	11
2	misc.forsale	interested	9
3	misc.forsale	shipping	9
4	misc.forsale	windows	9
5	misc.forsale	printer	8
6	misc.forsale	publish	8
7	misc.forsale	contact	7
8	misc.forsale	software	7
9	misc.forsale	compatible	6
10	rec.autos	someone	13
11	rec.autos	problem	11
12	rec.autos	anything	9
13	rec.autos	without	9
14	rec.autos	driving	8
15	rec.autos	traffic	8
16	rec.autos	because	7
17	rec.autos	between	7
18	rec.autos	information	7
19	rec.autos	business	6

Get unique categories for querying

```
[50]: top10_words['category'].unique()
```

```
[50]: array(['misc.forsale', 'rec.autos', 'rec.sport.baseball', 'sci.med',
   'sci.space', 'talk.politics.guns'], dtype=object)
```

Choose a couple categories

```
[51]: filt = top10_words['category'] == 'sci.space'
top10_words[filt]
```

	category	word	count
40	sci.space	telescope	27
41	sci.space	satellite	25
42	sci.space	national	24
43	sci.space	shuttle	22
44	sci.space	vehicle	18
45	sci.space	observatory	16
46	sci.space	because	15
47	sci.space	international	15
48	sci.space	spacecraft	14
49	sci.space	astronomical	13

```
[52]: filt = top10_words['category'] == 'talk.politics.guns'
top10_words[filt]
```

	category	word	count
50	talk.politics.guns	because	27
51	talk.politics.guns	federal	26
52	talk.politics.guns	believe	24
53	talk.politics.guns	against	23
54	talk.politics.guns	weapons	23
55	talk.politics.guns	without	23
56	talk.politics.guns	defense	22
57	talk.politics.guns	firearms	22
58	talk.politics.guns	control	21
59	talk.politics.guns	government	19

8.6 Project - Feature Engineering on the Titanic

Exercise 1

Extract the first character of the `Ticket` column and save it as a new column `ticket_first`. Find the total number of survivors, the total number of passengers, and the percentage of those who survived **by this column**. Next find the total survival rate for the entire dataset. Does this new column help predict who survived?

```
[53]: titanic = pd.read_csv('../data/titanic.csv')
titanic.head()
```

PassengerId	Survived	Pclass	Name	Sex	...	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	Braund, Mr. Owen Harris	male	...	0	A/5 21171	7.2500	NaN	S
1	2	1	Cumings, Mrs. John Bradley (Florence Briggs Th... Heikkinen, Miss. Laina	female	...	0	PC 17599 STON/O2. 3101282	71.2833 7.9250	C85 NaN	C S
2	3	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	...	0	113803	53.1000	C123	S
3	4	0	Allen, Mr. William Henry	male	...	0	373450	8.0500	NaN	S

```
[54]: titanic['ticket_first'] = titanic.Ticket.str[0]
titanic.head()
```

PassengerId	Survived	Pclass	Name	Sex	...	Ticket	Fare	Cabin	Embarked	ticket_first
0	1	0	Braund, Mr. Owen Harris	male	...	A/5 21171	7.2500	NaN	S	A
1	2	1	Cumings, Mrs. John Bradley (Florence Briggs Th... Heikkinen, Miss. Laina	female	...	PC 17599 STON/O2. 3101282	71.2833 7.9250	C85 NaN	C S	P S
2	3	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	...	113803	53.1000	C123	S	1
3	4	0	Allen, Mr. William Henry	male	...	373450	8.0500	NaN	S	3

```
[55]: ticket_first_survival = titanic.groupby('ticket_first').agg({'Survived': ['mean',  
                           'sum', 'size']})
ticket_first_survival
```

	Survived	mean	sum	size
	ticket_first			
1	0.630137	92	146	
2	0.464481	85	183	
3	0.239203	72	301	
4	0.200000	2	10	
5	0.000000	0	3	
6	0.166667	1	6	
7	0.111111	1	9	
8	0.000000	0	2	
9	1.000000	1	1	
A	0.068966	2	29	
C	0.340426	16	47	
F	0.571429	4	7	
L	0.250000	1	4	
P	0.646154	42	65	
S	0.323077	21	65	
W	0.153846	2	13	

```
[56]: # overall survival rate
titanic['Survived'].mean()
```

[56]: 0.3838383838383838

It does look like `ticket_first` has predictive power. 63% of those tickets beginning with ‘1’ survived while versus 24% for ‘3’. Only 2 out of 29 people survived with tickets beginning with ‘A’.

Exercise 2

If you did Exercise 2 correctly, you should see that only 7% of the people with tickets that began with ‘A’ survived. Find the survival rate for all those ‘A’ tickets by `Sex`.

```
[57]: filt = titanic['ticket_first'] == 'A'
ticket_A = titanic[filt]
ticket_A.groupby('Sex').agg({'Survived': ['mean', 'size']})
```

Survived		
	mean	size
Sex		
female	0.000000	2
male	0.074074	27

Exercise 3

Find the survival rate by the last letter of the ticket. Is there any predictive power here?

```
[58]: titanic['ticket_last'] = titanic['Ticket'].str[-1]
ticket_last_survival = titanic.groupby('ticket_last').agg({'Survived': ['mean', 'sum', 'size']})
ticket_last_survival
```

Survived			
	mean	sum	size
ticket_last			
0	0.395062	32	81
1	0.430000	43	100
2	0.364706	31	85
3	0.339286	38	112
4	0.297297	22	74
5	0.392405	31	79
6	0.419355	39	93
7	0.355556	32	90
8	0.453488	39	86
9	0.390805	34	87
E	0.250000	1	4

No predictive power. They are all about equal.

Exercise 4

Find the length of each passengers name and assign to the `name_len` column. What is the minimum and maximum name length?

```
[59]: titanic['name_len'] = titanic['Name'].str.len()
titanic['name_len'].min()
```

```
[59]: 12
```

```
[60]: titanic['name_len'].max()
```

[60]: 82

Exercise 5

Pass the `name_len` column to the `pd.cut` function. Also, pass a list of equal-sized cut points to the `bins` parameter. Assign the resulting Series to the `name_len_cat` column. Find the frequency count of each bin in this column.

```
[61]: titanic['name_len_cat'] = pd.cut(titanic['name_len'], bins=[0, 20, 40, 60, 80, 100])
titanic['name_len_cat'].head()
```

```
[61]: 0      (20, 40]
1      (40, 60]
2      (20, 40]
3      (40, 60]
4      (20, 40]
Name: name_len_cat, dtype: category
Categories (5, interval[int64]): [(0, 20] < (20, 40] < (40, 60] < (60, 80] < (80, 100]]
```

```
[62]: titanic['name_len_cat'].value_counts()
```

```
[62]: (20, 40]    558
(0, 20]     243
(40, 60]     86
(60, 80]      3
(80, 100]     1
Name: name_len_cat, dtype: int64
```

Exercise 6

Is name length a good predictor of survival?

```
[63]: titanic.groupby('name_len_cat').agg({'Survived': ['mean', 'size']})
```

	Survived	
	mean	size
name_len_cat		
(0, 20]	0.230453	243
(20, 40]	0.383513	558
(40, 60]	0.790698	86
(60, 80]	1.000000	3
(80, 100]	1.000000	1

Yes, the longer the name, the higher the survival rate.

Exercise 7

Why do you think people with longer names had a better chance at survival?

Let's output the shortest and longest 10 names

```
[64]: names = titanic.sort_values(by='name_len')['Name']
```

```
[65]: names.head(10)
```

```
[65]: 826      Lam, Mr. Len
  692      Lam, Mr. Ali
   74      Bing, Mr. Lee
  169      Ling, Mr. Lee
  509      Lang, Mr. Fang
  832      Saad, Mr. Amin
  210      Ali, Mr. Ahmed
  694      Weir, Col. John
  108      Rekic, Mr. Tido
  838      Chip, Mr. Chang
Name: Name, dtype: object
```

```
[66]: # Names exceed pandas display settings.
# change them with pd.options.display.max_colwidth
# or just print out values
names.tail(10).values
```

```
[66]: array(['Vander Planke, Mrs. Julius (Emelia Maria Vandemoortele)',
       'Rothes, the Countess. of (Lucy Noel Martha Dyer-Edwards)',
       'Spedden, Mrs. Frederic Oakley (Margaretta Corning Stone)',
       'Turpin, Mrs. William John Robert (Dorothy Ann Wonnacott)',
       'Asplund, Mrs. Carl Oscar (Selma Augusta Emilia Johansson)',
       'Andersson, Mrs. Anders Johan (Alfrida Konstantia Brogren)',
       'Brown, Mrs. Thomas William Solomon (Elizabeth Catherine Ford)',
       'Duff Gordon, Lady. (Lucille Christiana Sutherland) ("Mrs Morgan")',
       'Phillips, Miss. Kate Florence ("Mrs Kate Louise Phillips Marshall")',
       'Penasco y Castellana, Mrs. Victor de Satode (Maria Josefa Perez de Soto y
Vallejo)',

       dtype=object)
```

```
[67]: # temporarily set options in a context manager
with pd.option_context('display.max_colwidth', 100):
    print(names.tail(10))
```

```
18                  Vander Planke, Mrs. Julius (Emelia Maria Vandemoortele)
  759                 Rothes, the Countess. of (Lucy Noel Martha Dyer-Edwards)
   319                 Spedden, Mrs. Frederic Oakley (Margaretta Corning Stone)
    41                 Turpin, Mrs. William John Robert (Dorothy Ann Wonnacott)
    25                 Asplund, Mrs. Carl Oscar (Selma Augusta Emilia Johansson)
   610                 Andersson, Mrs. Anders Johan (Alfrida Konstantia Brogren)
   670                 Brown, Mrs. Thomas William Solomon (Elizabeth Catherine Ford)
   556                 Duff Gordon, Lady. (Lucille Christiana Sutherland) ("Mrs Morgan")
   427                 Phillips, Miss. Kate Florence ("Mrs Kate Louise Phillips Marshall")
  307     Penasco y Castellana, Mrs. Victor de Satode (Maria Josefa Perez de Soto y Vallejo)
Name: Name, dtype: object
```

Looks like all the people with short names are men. All people with long names are females.

Exercise 8

Using the titanic dataset, do your best to extract the title from a person's name. Examples of title are 'Mr.', 'Dr.', 'Miss', etc... Save this to a column called `title`. Find the frequency count of the titles.

```
[68]: titanic['title'] = titanic['Name'].str.extract(r'(\w+[\.])')
titanic['title'].value_counts()
```

```
[68]: Mr.          517
Miss.        182
Mrs.         125
Master.       40
Dr.           7
Rev.           6
Major.         2
Col.           2
Mlle.          2
Ms.            1
Mme.          1
Capt.          1
Sir.           1
Don.           1
Countess.      1
Lady.          1
Jonkheer.      1
Name: title, dtype: int64
```

Exercise 9

Does the title have good predictive value of survival?

```
[69]: titanic.groupby('title').agg({'Survived': ['mean', 'size']})
```

Survived		
	mean	size
title		
Capt.	0.000000	1
Col.	0.500000	2
Countess.	1.000000	1
Don.	0.000000	1
Dr.	0.428571	7
Jonkheer.	0.000000	1
Lady.	1.000000	1
Major.	0.500000	2
Master.	0.575000	40
Miss.	0.697802	182
Mlle.	1.000000	2
Mme.	1.000000	1
Mr.	0.156673	517
Mrs.	0.792000	125
Ms.	1.000000	1
Rev.	0.000000	6
Sir.	1.000000	1

Exercise 10

Create a pivot table of survival by title and sex. Use two aggregation functions, mean and size

```
[70]: titanic.pivot_table(index='title', columns='Sex',
                           values='Survived', aggfunc=['mean', 'size'])
```

Sex	mean		size	
	female	male	female	male
title				
Capt.	NaN	0.000000	NaN	1.0
Col.	NaN	0.500000	NaN	2.0
Countess.	1.000000		1.0	NaN
Don.	NaN	0.000000	NaN	1.0
Dr.	1.000000	0.333333	1.0	6.0
Jonkheer.	NaN	0.000000	NaN	1.0
Lady.	1.000000		1.0	NaN
Major.	NaN	0.500000	NaN	2.0
Master.	NaN	0.575000	NaN	40.0
Miss.	0.697802		182.0	NaN
Mlle.	1.000000		2.0	NaN
Mme.	1.000000		1.0	NaN
Mr.	NaN	0.156673	NaN	517.0
Mrs.	0.792000		125.0	NaN
Ms.	1.000000		1.0	NaN
Rev.	NaN	0.000000	NaN	6.0
Sir.	NaN	1.000000	NaN	1.0

Exercise 11

Attempt to extract the first name of each passenger into the column `first_name`. Are there are males and females with the same first name?

Most can be found like this following the title

```
[71]: pattern = r'\w+[\. ] (\w+)'
titanic['first_name'] = titanic['Name'].str.extract(pattern)
```

To be more precise, we can do this:

```
[72]: pattern = r'\w+[\. ] [a-z ()]+([A-Z]\w+)'
titanic['first_name'] = titanic['Name'].str.extract(pattern)
```

```
[73]: first_name_ct = titanic.groupby('first_name').agg({'Sex': 'nunique'})
first_name_ct.head()
```

Sex	
first_name	
Abraham	1
Achille	1
Ada	1
Adele	1
Adola	1

```
[74]: filt = first_name_ct['Sex'] == 2
first_name_ct[filt].head(10)
```

Sex	
first_name	
Albert	2
Alexander	2
Amin	2
Anders	2
Antoni	2
Benjamin	2
Carl	2
Charles	2
Dickinson	2
Edgar	2

Looks like some female first names are actually in parentheses after their husband/father name.

```
[75]: filt = titanic['first_name'] == 'Albert'
titanic.loc[filt, 'Name']
```

```
[75]: 64                      Stewart, Mr. Albert A
      107                     Moss, Mr. Albert Johan
      323    Caldwell, Mrs. Albert Francis (Sylvia Mae Harb...
      690                      Dick, Mr. Albert Adrian
      781    Dick, Mrs. Albert Adrian (Vera Gillespie)
      817                      Mallet, Mr. Albert
      833    Augustsson, Mr. Albert
Name: Name, dtype: object
```

Exercise 12

The past several exercises have been an exercise in **feature engineering**. Several new features (columns) have been created from existing columns. Come up with your own feature and test it out on survival.

Get first letter of cabin. Use ‘Missing’ if not present.

```
[76]: titanic['cabin_first'] = titanic.Cabin.str[0].fillna('Missing')
```

Just having a cabin is highly predictive.

```
[77]: titanic.groupby('cabin_first').agg({'Survived': ['size', 'mean']}).
      ↪sort_values('size', ascending=False)
```

cabin_first	Survived	
	size	mean
A	15	0.466667
B	47	0.744681
C	59	0.593220
D	33	0.757576
E	32	0.750000
F	13	0.615385
G	4	0.500000
Missing	687	0.299854
T	1	0.000000

Part IX

Tidy Data

Chapter 9

Solutions

9.1 1. Tidy Data with `melt`

```
[1]: import pandas as pd  
import numpy as np  
pd.options.display.max_columns = 40
```

```
[2]: movie = pd.read_csv('../data/movie.csv')  
movie.head(2)
```

	title	year	color	content_rating	duration	...	plot_keywords	language	country	budget	imdb_score
0	Avatar	2009.0	Color	PG-13	178.0	...	avatar future marine native paraplegic	English	USA	2370000000.0	7.9
1	Pirates of the Caribbean: At World's End	2007.0	Color	PG-13	169.0	...	goddess marriage ceremony marriage proposal pi...	English	USA	300000000.0	7.1

Exercise 1

Read in the movie dataset. Select the title column and all of the actor name columns. Restructure the dataset so that there are only three variables - the title of the movie, the actor number (1, 2, or 3), and the actor name. Sort the result by title and output the result.

```
[3]: actor_tidy = movie.melt(id_vars='title',  
                           value_vars=['actor1', 'actor2', 'actor3'],  
                           var_name='Actor Number',  
                           value_name='Actor Name').sort_values('title')  
actor_tidy.head(6)
```

	title	Actor Number	Actor Name
14181	#Horror	actor3	Lydia Hearst
9265	#Horror	actor2	Balthazar Getty
4349	#Horror	actor1	Timothy Hutton
13461	10 Cloverfield Lane	actor3	Sumalee Montano
8545	10 Cloverfield Lane	actor2	John Gallagher Jr.
3629	10 Cloverfield Lane	actor1	Bradley Cooper

Exercise 2

Using the original movie dataset (and keeping its structure), attempt to count the total appearances of each actor in the dataset regardless whether they are 1, 2, or 3. Then repeat this task with your tidy dataset.

We have not covered how to do the first part of the question. It becomes very easy with tidy data.

```
[4]: actor_ct = actor_tidy['Actor Name'].value_counts()
      actor_ct.head()
```

```
[4]: Robert De Niro      53
      Morgan Freeman     43
      Bruce Willis        38
      Matt Damon          37
      Johnny Depp          36
      Name: Actor Name, dtype: int64
```

Advanced Pandas for first part. This is a pandas trick that uses the `add` method to add Series together:

```
[5]: vc1 = movie['actor1'].value_counts()
      vc2 = movie['actor2'].value_counts()
      vc3 = movie['actor3'].value_counts()
```

```
[6]: actor_ct2 = vc1.add(vc2, fill_value=0) \
      .add(vc3, fill_value=0) \
      .astype(int) \
      .sort_values(ascending=False)
      actor_ct2.head()
```

```
[6]: Robert De Niro      53
      Morgan Freeman     43
      Bruce Willis        38
      Matt Damon          37
      Steve Buscemi       36
      dtype: int64
```

Exercise 3

Tidy the dataset in the `tidy/employee_messy1.csv` file. It contains the count of all employees by race and gender.

```
[7]: em = pd.read_csv('../data/tidy/employee_messy1.csv')
em
```

		race	Female	Male
0	Native American		6	5
1	Asian		30	77
2	Black		305	395
3	Hispanic		139	341
4	White		108	557

```
[8]: em.melt(id_vars='race', value_vars=['Female', 'Male'],
           var_name='gender', value_name='count')
```

		race	gender	count
0	Native American	Female		6
1	Asian	Female		30
2	Black	Female		305
3	Hispanic	Female		139
4	White	Female		108
5	Native American	Male		5
6	Asian	Male		77
7	Black	Male		395
8	Hispanic	Male		341
9	White	Male		557

Exercise 4

Tidy the dataset in the tidy/employee_messy2.csv file. It contains the count of all employees by department, race and gender.

```
[9]: em2 = pd.read_csv('../data/tidy/employee_messy2.csv')
em2.head()
```

		dept	gender	Asian	Black	Hispanic	White
0	Health & Human Services	Female		6	43	22	6
1	Health & Human Services	Male		5	10	5	6
2	Houston Fire Department (HFD)	Female		0	7	8	6
3	Houston Fire Department (HFD)	Male		1	62	91	203
4	Houston Police Department-HPD	Female		6	76	35	35

```
[10]: em2.melt(id_vars=['dept', 'gender'],
            var_name='race',
            value_name='count').head()
```

		dept	gender	race	count
0		Health & Human Services	Female	Asian	6
1		Health & Human Services	Male	Asian	5
2		Houston Fire Department (HFD)	Female	Asian	0
3		Houston Fire Department (HFD)	Male	Asian	1
4		Houston Police Department-HPD	Female	Asian	6

Exercise 5

Tidy the dataset in the `tidy/employee_salary_stats.csv` file. Save the tidy dataset to a variable and then select all the median salaries. The select all the median salaries with the original ‘messy’ dataset. Which one is easier to read summary statistics from?

```
[11]: em_stats = pd.read_csv('../data/tidy/employee_salary_stats.csv')
em_stats.head()
```

	race	gender	min	mean	median	max
0	Native American	Female	26125	60238	58855	98536
1	Native American	Male	26125	60305	60347	81239
2	Asian	Female	26125	63226	57227	130416
3	Asian	Male	27914	61033	55461	163228
4	Black	Female	24960	48915	44491	150416

```
[12]: df_tidy = em_stats.melt(id_vars=['race', 'gender'],
                           var_name='Statistic',
                           value_name='Salary Value')
df_tidy.head()
```

	race	gender	Statistic	Salary Value
0	Native American	Female	min	26125
1	Native American	Male	min	26125
2	Asian	Female	min	26125
3	Asian	Male	min	27914
4	Black	Female	min	24960

```
[13]: filt = df_tidy['Statistic'] == 'median'
df_tidy[filt]
```

	race	gender	Statistic	Salary Value
20	Native American	Female	median	58855
21	Native American	Male	median	60347
22	Asian	Female	median	57227
23	Asian	Male	median	55461
24	Black	Female	median	44491
25	Black	Male	median	46486
26	Hispanic	Female	median	43087
27	Hispanic	Male	median	54090
28	White	Female	median	62264
29	White	Male	median	62540

```
[14]: cols = ['race', 'gender', 'median']
em_stats[cols]
```

	race	gender	median
0	Native American	Female	58855
1	Native American	Male	60347
2	Asian	Female	57227
3	Asian	Male	55461
4	Black	Female	44491
5	Black	Male	46486
6	Hispanic	Female	43087
7	Hispanic	Male	54090
8	White	Female	62264
9	White	Male	62540

The messy dataset is probably easier, as it shows all the aggregated statistics for each race and gender in a single row. Aggregated values are generally easier to read as “messy” datasets.

9.2 2. Reshaping by Pivoting

Exercise 1

Read the file `clean_movie1.csv` and then use the `pivot` method to put the country names as the columns. Put the `count` as the new values for the DataFrame.

```
[15]: cm = pd.read_csv('../data/tidy/clean_movie1.csv')
cm.head()
```

	content_rating	country	count
0	G	Australia	2
1	G	Canada	1
2	G	France	5
3	G	Germany	1
4	G	UK	11

```
[16]: cm.pivot(index='content_rating', columns='country', values='count')
```

	country	Australia	Canada	France	Germany	India	Spain	UK	USA
content_rating									
G	2.0	1.0	5.0	1.0	NaN	NaN	11.0	85.0	
Not Rated	NaN	9.0	11.0	4.0	6.0	2.0	10.0	51.0	
PG	11.0	10.0	12.0	11.0	2.0	2.0	70.0	544.0	
PG-13	10.0	25.0	34.0	22.0	2.0	6.0	97.0	1168.0	
R	25.0	64.0	63.0	49.0	3.0	21.0	198.0	1534.0	

```
[17]: # can fill in the missing value with 0 and change type to integer
cm.pivot(index='content_rating', columns='country', values='count').fillna(0).
    ↪astype(int)
```

	country	Australia	Canada	France	Germany	India	Spain	UK	USA
content_rating									
G	2	1	5	1	0	0	11	85	
Not Rated	0	9	11	4	6	2	10	51	
PG	11	10	12	11	2	2	70	544	
PG-13	10	25	34	22	2	6	97	1168	
R	25	64	63	49	3	21	198	1534	

Exercise 2

Read in the NYC deaths dataset and select only males from 2007. Pivot this information so we can more clearly see the breakdown of causes of death by race. Assign the result to a variable.

```
[18]: nyc_deaths = pd.read_csv('../data/nyc_deaths.csv')
nyc_deaths.head()
```

	year	cause	sex	race	deaths
0	2007	Accidents	F	Asian	32
1	2007	Accidents	F	Black	87
2	2007	Accidents	F	Hispanic	71
3	2007	Accidents	F	White	162
4	2007	Accidents	M	Asian	53

```
[19]: male_2007 = nyc_deaths[(nyc_deaths['year'] == 2007) & (nyc_deaths['sex'] == 'M')]
race_death = male_2007.pivot(index='cause', columns='race', values='deaths')
race_death
```

	race	Asian	Black	Hispanic	Unknown	White
cause						
Accidents	53.0	158.0	154.0	13.0	297.0	
Cancer	528.0	1523.0	1013.0	74.0	3356.0	
Diabetes	47.0	246.0	177.0	20.0	237.0	
Flu and Pneumonia	66.0	229.0	189.0	20.0	530.0	
HIV	NaN	377.0	218.0	6.0	NaN	
Heart Disease	496.0	2121.0	1327.0	122.0	5632.0	
Homicide	NaN	267.0	114.0	NaN	NaN	
Hypertension	25.0	137.0	NaN	NaN	123.0	
Liver Disease	18.0	NaN	107.0	NaN	NaN	
Other	221.0	1163.0	988.0	67.0	1749.0	
Respiratory Diseases	43.0	137.0	NaN	NaN	345.0	
Stroke	56.0	213.0	126.0	7.0	267.0	
Substance Abuse	NaN	163.0	183.0	10.0	261.0	
Suicide	36.0	NaN	NaN	5.0	187.0	

Exercise 3

Use the result from Exercise 2 and highlight the leading cause of death for each race. Is it the same for each one?

Cancer is the most common among Asian and Pacific Islanders. Heart Disease for everyone else.

```
[20]: race_death.style.highlight_max()
```

cause	race	Asian	Black	Hispanic	Unknown	White
	Accidents	53	158	154	13	297
	Cancer	528	1523	1013	74	3356
	Diabetes	47	246	177	20	237
	Flu and Pneumonia	66	229	189	20	530
	HIV	nan	377	218	6	nan
	Heart Disease	496	2121	1327	122	5632
	Homicide	nan	267	114	nan	nan
	Hypertension	25	137	nan	nan	123
	Liver Disease	18	nan	107	nan	nan
	Other	221	1163	988	67	1749
	Respiratory Diseases	43	137	nan	nan	345
	Stroke	56	213	126	7	267
	Substance Abuse	nan	163	183	10	261
	Suicide	36	nan	nan	5	187

Exercise 4

Read in the flights dataset. Find the total number of flights from each airline by their origin airport. Hint: When making a pivot table of just frequency, its not necessary to have a `values` column. Save the results to a variable.

```
[21]: flights = pd.read_csv('../data/flights.csv')
flights.head()
```

	date	airline	origin	dest	dep_time	...	carrier_delay	weather_delay	nas_delay	security_delay	late_aircraft_delay
0	2018-01-01	UA	LAS	IAH	100	...	0	0	0	0	0
1	2018-01-01	WN	DEN	PHX	515	...	0	0	0	0	0
2	2018-01-01	B6	JFK	BOS	550	...	0	83	8	0	0
3	2018-01-01	B6	DTW	BOS	600	...	0	0	19	0	0
4	2018-01-01	UA	LAS	EWR	600	...	0	0	0	0	0

```
[22]: airline_origin = flights.pivot_table(index='airline', columns='origin',
                                         aggfunc='size', fill_value=0)
airline_origin
```

origin	ATL	BOS	CLT	DCA	DEN	...	ORD	PHL	PHX	SEA	SFO
airline											
9E	7	102	88	55	0	...	47	14	0	0	0
AA	442	1084	1935	767	393	...	1716	1275	1264	340	492
AS	22	76	0	48	81	...	96	33	93	995	542
B6	113	1053	60	204	37	...	100	92	32	51	161
DL	2789	545	242	369	425	...	360	248	262	710	527
EV	0	0	11	66	0	...	0	0	0	0	0
F9	59	0	28	44	368	...	67	64	40	23	44
MQ	32	8	0	1	0	...	65	36	0	0	0
NK	190	85	0	0	104	...	247	97	21	66	0
OH	43	0	92	47	0	...	0	4	0	0	0
OO	47	17	73	35	60	...	449	22	82	51	143
UA	168	541	31	210	1338	...	1755	187	312	377	1415
VX	0	16	0	2	10	...	8	6	0	32	122
WN	531	70	0	70	866	...	0	134	710	156	382
YV	62	0	31	34	0	...	0	18	81	0	0
YX	176	130	154	607	30	...	258	64	0	0	0

Exercise 5

Highlight the origin airport with the most flights for each airline. Do a few online searches to determine if those airports are hubs for those airlines.

```
[23]: airline_origin.style.highlight_max(axis=1)
```

origin	ATL	BOS	CLT	DCA	DEN	DFW	DTW	EWR	IAH	JFK	LAS	LAX	LGA	MCO	MSP	ORD	PHL	PHX	SEA	SFO
airline																				
9E	7	102	88	55	0	88	117	44	52	285	0	0	56	0	82	47	14	0	0	0
AA	442	1084	1935	767	393	2337	199	262	287	594	559	1152	780	631	270	1716	1275	1264	340	492
AS	22	76	0	48	81	51	21	134	24	155	259	535	0	44	35	96	33	93	995	542
B6	113	1053	60	204	37	32	46	172	0	701	107	220	127	474	34	100	92	32	51	161
DL	2789	545	242	369	425	243	1263	197	93	731	472	920	641	639	1428	360	248	262	710	527
EV	0	0	11	66	0	1	6	59	3	0	0	0	25	0	0	0	0	0	0	0
F9	59	0	28	44	368	19	22	0	21	0	92	42	34	140	34	67	64	40	23	44
MQ	32	8	0	1	0	6	49	12	57	34	0	0	65	0	8	65	36	0	0	0
NK	190	85	0	0	104	291	230	79	211	0	383	219	78	309	154	247	97	21	66	0
OH	43	0	92	47	0	0	51	2	0	5	0	0	8	0	5	0	4	0	0	0
OO	47	17	73	35	60	124	193	71	95	29	58	237	111	0	188	449	22	82	51	143
UA	168	541	31	210	1338	273	61	1420	1430	0	491	887	361	473	152	1755	187	312	377	1415
VX	0	16	0	2	10	0	0	13	0	36	50	127	0	7	0	8	6	0	32	122
WN	531	70	0	70	866	0	110	62	0	0	734	574	106	278	129	0	134	710	156	382
YV	62	0	31	34	0	70	38	0	346	0	0	0	6	0	43	0	18	81	0	0
YX	176	130	154	607	30	153	228	348	178	51	0	0	661	0	162	258	64	0	0	0

American Airlines (AA) has a hub at DFW

Exercise 6

Read in the bikes dataset. For each type of weather event (the `events` column) find the median temperature for males and females.

```
[24]: bikes = pd.read_csv('../data/bikes.csv')
```

```
[25]: bikes.pivot_table(index='events', columns='gender',
                       values='temperature', aggfunc='median').style.
    highlight_max(axis='columns')
```

	gender	Female	Male
	events		
clear	64	62.6	
cloudy	61	57	
fog	54	52	
hazy	60.1	57.9	
mostlycloudy	72	71.1	
partlycloudy	71.1	69.1	
rain	59	57.9	
sleet	30.9	32	
snow	28.45	28	
tstorms	74.45	75.9	
unknown	-9999	50	

Exercise 7

Reshape the movie dataset so that there are two columns, one for all of the actors and one for the content rating of each of their respective movies. Filter this DataFrame so that it contains the top 10 most common actors. Then create a table that displays the number of movies each actor made by content rating. The actor names should be in the index, with the content ratings in the columns, with the counts as the values.

```
[26]: actor_rating = movie.melt(id_vars='content_rating', value_vars=['actor1', 'actor2', 'actor3'],
                                value_name='actor')
actor_rating = actor_rating.drop(columns='variable')
actor_rating.head()
```

	content_rating	actor
0	PG-13	CCH Pounder
1	PG-13	Johnny Depp
2	PG-13	Christoph Waltz
3	PG-13	Tom Hardy
4	NaN	Doug Walker

```
[27]: top10_actors = actor_rating['actor'].value_counts().index[:10]
top10_actors
```

```
[27]: Index(['Robert De Niro', 'Morgan Freeman', 'Bruce Willis', 'Matt Damon',
           'Steve Buscemi', 'Johnny Depp', 'Brad Pitt', 'Nicolas Cage',
           'Bill Murray', 'Will Ferrell'],
           dtype='object')
```

```
[28]: actor_rating_top10 = actor_rating[actor_rating['actor'].isin(top10_actors)]
actor_rating_top10.head()
```

	content_rating	actor
1	PG-13	Johnny Depp
13	PG-13	Johnny Depp
14	PG-13	Johnny Depp
18	PG-13	Johnny Depp
33	PG	Johnny Depp

```
[29]: actor_rating_top10.pivot_table(index='actor', columns='content_rating',  
    aggfunc='size', fill_value=0)
```

	content_rating	G	Not Rated	PG	PG-13	R	X
actor							
Bill Murray	0	0	9	11	11	0	
Brad Pitt	0	0	3	10	20	0	
Bruce Willis	0	0	2	15	21	0	
Johnny Depp	0	0	7	13	15	1	
Matt Damon	1	0	2	18	16	0	
Morgan Freeman	1	0	5	20	17	0	
Nicolas Cage	0	0	5	9	19	0	
Robert De Niro	0	0	4	12	37	0	
Steve Buscemi	3	1	5	12	15	0	
Will Ferrell	1	0	4	17	10	0	

9.3 3. Common messy datasets

Exercise 1

Make the `country_hour_price.csv` dataset tidy by putting all the hour columns into a single column.

```
[30]: df = pd.read_csv('../data/tidy/country_hour_price.csv')
df
```

	ASID	BORDER	HOUR1	HOUR2
0	21	GERMANY	2	3
1	32	FRANCE	2	3
2	99	ITALY	2	3
3	77	USA	4	5
4	66	CANADA	4	5
5	55	MEXICO	4	5
6	44	INDIA	6	7
7	88	CHINA	6	7
8	111	JAPAN	6	7

```
[31]: df_tidy = df.melt(id_vars=['ASID', 'BORDER'],
                      value_vars=['HOUR1', 'HOUR2'],
                      var_name = 'Hour')
df_tidy.head()
```

	ASID	BORDER	Hour	value
0	21	GERMANY	HOUR1	2
1	32	FRANCE	HOUR1	2
2	99	ITALY	HOUR1	2
3	77	USA	HOUR1	4
4	66	CANADA	HOUR1	4

```
[32]: df_tidy.dtypes
```

```
[32]: ASID      int64
BORDER    object
Hour      object
value     int64
dtype: object
```

Exercise 2

If the resulting DataFrame from Exercise 1 has the strings ‘HOUR1’ and ‘HOUR2’ as values in the hour column, then extract just the numerical part of the strings and reassign the result to the hour column.

```
[33]: df_tidy['Hour'] = df_tidy['Hour'].str.extract('(\d)').astype('int')
df_tidy.head()
```

	ASID	BORDER	Hour	value
0	21	GERMANY	1	2
1	32	FRANCE	1	2
2	99	ITALY	1	2
3	77	USA	1	4
4	66	CANADA	1	4

[34]: df_tidy.dtypes

[34]: ASID int64
 BORDER object
 Hour int64
 value int64
 dtype: object

Exercise 3

Tidy the tidy/flights_status.csv dataset.

[35]: fs = pd.read_csv('..../data/tidy/flight_status.csv')
 fs.head()

	airline	status	ATL	DEN	DFW	...	LAX	MSP	ORD	PHX	SFO
0	AA	Delayed	42	57	893	...	202	25	378	170	68
1	AA	On Time	191	162	3113	...	783	122	1118	709	297
2	AS	Delayed	3	4	2	...	23	4	7	5	23
3	AS	On Time	10	46	45	...	239	11	45	59	114
4	B6	Delayed	0	10	5	...	32	0	30	5	33

[36]: fs1 = fs.melt(id_vars=['airline', 'status'], var_name='airport', value_name='count')
 fs1.head(15)

	airline	status	airport	count
0	AA	Delayed	ATL	42
1	AA	On Time	ATL	191
2	AS	Delayed	ATL	3
3	AS	On Time	ATL	10
4	B6	Delayed	ATL	0
5	B6	On Time	ATL	0
6	DL	Delayed	ATL	924
7	DL	On Time	ATL	5696
8	EV	Delayed	ATL	373
9	EV	On Time	ATL	1319
10	F9	Delayed	ATL	41
11	F9	On Time	ATL	107
12	HA	Delayed	ATL	0
13	HA	On Time	ATL	0
14	MQ	Delayed	ATL	23

```
[37]: fs_tidy = fs1.pivot_table(index=['airline', 'airport'], columns='status',
                                values='count', aggfunc='max')
fs_tidy.head()
```

airline	airport	status	
		Delayed	On Time
AA	ATL	42	191
	DEN	57	162
	DFW	893	3113
	IAH	40	156
	LAS	79	295

```
[38]: fs_tidy_final = fs_tidy.reset_index().rename_axis(None, axis='columns')
fs_tidy_final.head(10)
```

	airline	airport	Delayed	On Time
0	AA	ATL	42	191
1	AA	DEN	57	162
2	AA	DFW	893	3113
3	AA	IAH	40	156
4	AA	LAS	79	295
5	AA	LAX	202	783
6	AA	MSP	25	122
7	AA	ORD	378	1118
8	AA	PHX	170	709
9	AA	SFO	68	297

Exercise 4

Tidy the tidy/metrics.csv dataset.

```
[39]: df = pd.read_csv('../data/tidy/metrics.csv')
df
```

	year	metric	Black Male	Black Female	White Male	White Female	Hispanic Male	Hispanic Female
0	2010	income	54885	51058	53941	58671	55344	52187
1	2010	life expectancy	77	72	73	70	79	73
2	2011	income	57073	58494	59718	58239	53891	59914
3	2011	life expectancy	70	76	79	79	77	74
4	2012	income	57028	50207	52610	58672	55844	55632
5	2012	life expectancy	78	72	73	73	75	74

```
[40]: df1 = df.melt(id_vars=['year', 'metric'], var_name='race sex')
df1.head()
```

	year	metric	race sex	value
0	2010	income	Black Male	54885
1	2010	life expectancy	Black Male	77
2	2011	income	Black Male	57073
3	2011	life expectancy	Black Male	70
4	2012	income	Black Male	57028

```
[41]: df2 = df1.pivot_table(index=['year', 'race sex'], columns='metric', values='value',  
                         aggfunc='max')  
df3 = df2.reset_index().rename_axis(None, axis='columns')  
df3.head()
```

	year	race sex	income	life expectancy
0	2010	Black Female	51058	72
1	2010	Black Male	54885	77
2	2010	Hispanic Female	52187	73
3	2010	Hispanic Male	55344	79
4	2010	White Female	58671	70

```
[42]: df3[['race', 'sex']] = df3['race sex'].str.split(expand=True)  
df3.head()
```

	year	race	sex	income	life expectancy	race	sex
0	2010	Black	Female	51058	72	Black	Female
1	2010	Black	Male	54885	77	Black	Male
2	2010	Hispanic	Female	52187	73	Hispanic	Female
3	2010	Hispanic	Male	55344	79	Hispanic	Male
4	2010	White	Female	58671	70	White	Female

```
[43]: df4 = df3.drop(columns='race sex')  
df4.head()
```

	year	income	life expectancy	race	sex
0	2010	51058	72	Black	Female
1	2010	54885	77	Black	Male
2	2010	52187	73	Hispanic	Female
3	2010	55344	79	Hispanic	Male
4	2010	58671	70	White	Female

Part X

Joining Data

Chapter 10

Solutions

10.1 4. Data Normalization

[]:

Part XI

Visualization

Chapter 11

Solutions

```
[1]: import pandas as pd  
import numpy as np  
import matplotlib.pyplot as plt  
%matplotlib inline
```

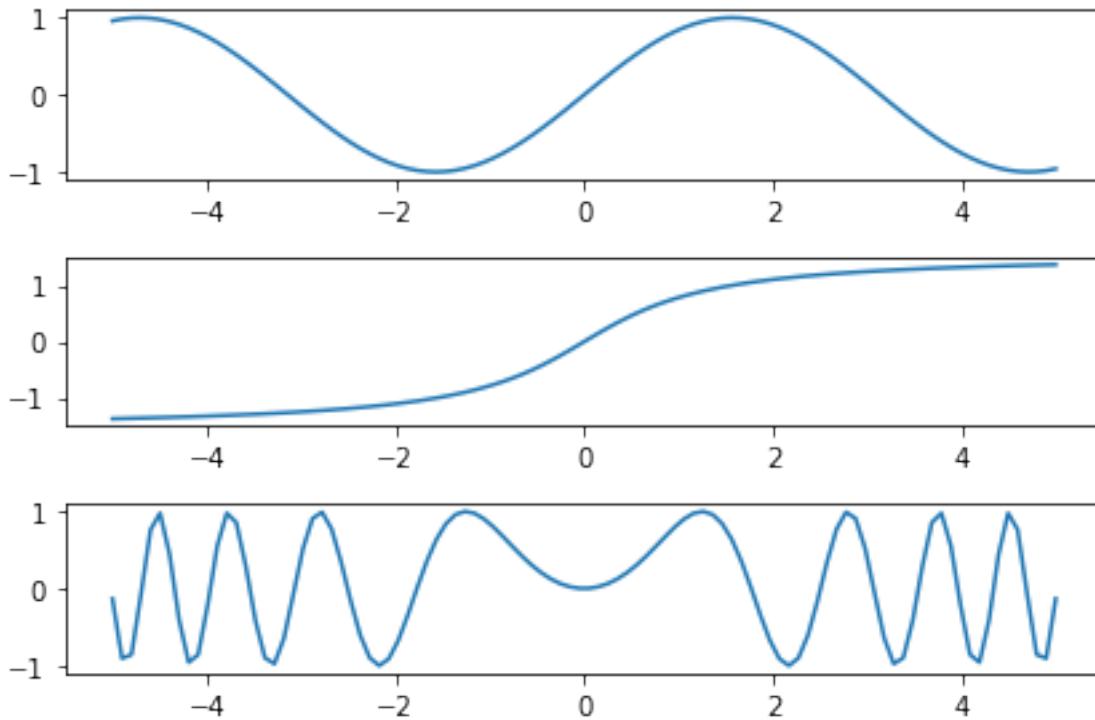
11.1 4. Matplotlib Data Plotting

```
[2]: import pandas as pd  
import numpy as np  
import matplotlib.pyplot as plt  
%matplotlib inline
```

Exercise 1

Create a Figure with three Axes using `plt.subplots`. Use `np.linspace` to create a one dimensional array of data from -5 to 5 of length 100 and store this to `x`. In each of three Axes, plot take some mathematical function of `x` to create `y` values in a line plot. For instance, you can take the square root of `x`.

```
[3]: fig, ax_array = plt.subplots(3, 1)  
ax1 = ax_array[0]  
ax2 = ax_array[1]  
ax3 = ax_array[2]  
x = np.linspace(-5, 5, 100)  
ax1.plot(x, np.sin(x))  
ax2.plot(x, np.arctan(x))  
ax3.plot(x, np.sin(x ** 2))  
fig.tight_layout() # this makes all Axes fit correctly
```



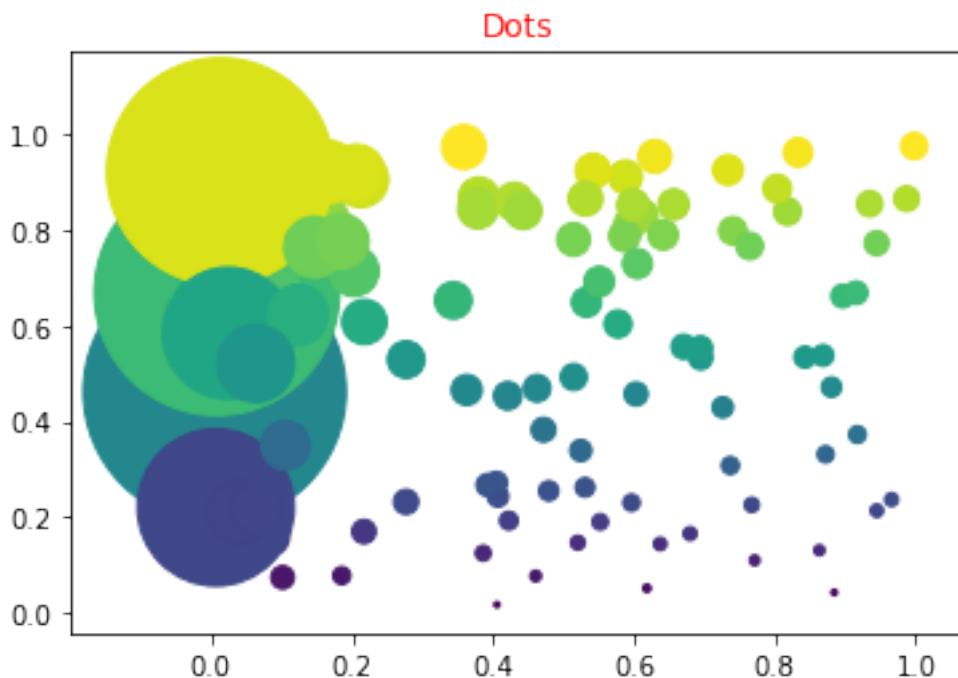
Exercise 2

Use `np.random.rand` to create two arrays, `x` and `y` that are each 100 units in length. Make a scatter plot of the data. Make the size of the markers proportional to the ratio of `y` to `x`. Make the color proportional to `y`. For scatter plots, use the parameter `c` to control the color with a number. The parameter `s` controls the size. Set the title as well.

```
[4]: x = np.random.rand(100)
y = np.random.rand(100)

fig, ax = plt.subplots(figsize=(6, 4))

ax.scatter(x, y, c=y, s=y / x * 100)
ax.set_title('Dots', color='Red');
```



Exercise 3

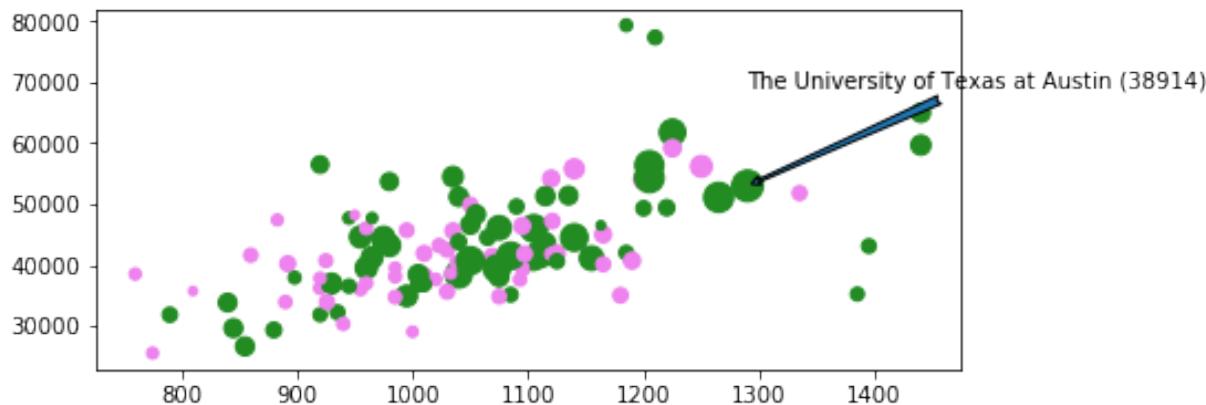
Read in the college dataset and set the index to the institution name. Complete the following tasks:

- Convert the median earnings within 10 years (MD_EARN_WNE_P10) column to numeric
- Create a column for the total SAT score
- Select just the columns for SAT total, UGDS, RELAFFIL and MD_EARN_WNE_P10 into another DataFrame called `college_samp`. Continue with this DataFrame for the rest of the Exercise.
- Drop any rows with missing values
- Randomly sample 10% of the DataFrame and assign it back to itself.
- Call the `map` method on the RELAFFIL column. Pass it a dictionary to convert the values to color names. Assign the result to the column `color`
- Take the square root of the UGDS column and assign it to the column `size`.
- Create a scatterplot of the total SAT scores vs the MD_EARN_WNE_P10 column. Color and size each point with their respective columns.
- Extra Credit: Annotate the school with the largest population as it is done [in this example](#)

```
[5]: college = pd.read_csv("../data/college.csv", index_col='instnm')
college['md_earn_wne_p10'] = pd.to_numeric(college['md_earn_wne_p10'], errors='coerce')
college['sat_total'] = college['satmtmid'] + college['satvrmid']
college_samp = college[['sat_total', 'ugds', 'relaffil', 'md_earn_wne_p10']]
college_samp = college_samp.dropna()
college_samp = college_samp.sample(frac=.1)
college_samp['color'] = college_samp['relaffil'].map({0: 'forestgreen', 1:'violet'})
college_samp['size'] = college_samp['ugds'] ** .5
max_school = college_samp['ugds'].idxmax()
x = college_samp.loc[max_school, 'sat_total']
y = college_samp.loc[max_school, 'md_earn_wne_p10']
college_samp.head()
```

instnm	sat_total	ugds	relaffil	md_earn_wne_p10	color	size
Tufts University	1440.0	5143.0	0	65000.0	forestgreen	71.714713
University of Houston-Clear Lake	1050.0	5072.0	0	46600.0	forestgreen	71.217975
The University of Texas at Austin	1290.0	38914.0	0	53000.0	forestgreen	197.266317
Kettering University	1210.0	1689.0	0	77400.0	forestgreen	41.097445
California Polytechnic State University-San Luis Obispo	1225.0	19177.0	0	61800.0	forestgreen	138.481046

```
[6]: fig, ax = plt.subplots(figsize=(7, 3))
ax.scatter('sat_total', 'md_earn_wne_p10', data=college_samp, c='color', s='size')
max_ugds = int(college_samp['ugds'].max())
school_size = max_school + f' ({max_ugds})'
ax.annotate(school_size, xy=(x, y), xytext=(x, y * 1.3), arrowprops={'arrowstyle': 'fancy'});
```



Exercise 4

Read in the employee dataset and select the `salary` column as a Series, drop the missing values, and assign it to a variable. Read about the `pd.cut` function and create categories that span 25k from 0 to 300k. Save this result as a Series and find the frequency of each category. Then take that result and create a pie chart with labels.

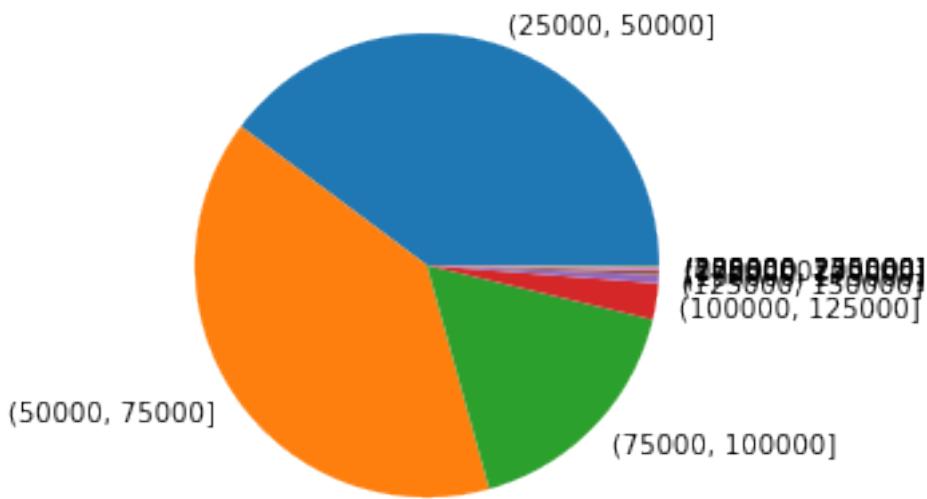
```
[7]: emp = pd.read_csv('../data/employee.csv', parse_dates=['hire_date'])
emp.head()
```

	dept	title	hire_date	salary	sex	race
0	Police	POLICE SERGEANT	2001-12-03	87545.38	Male	White
1	Other	ASSISTANT CITY ATTORNEY II	2010-11-15	82182.00	Male	Hispanic
2	Houston Public Works	SENIOR SLUDGE PROCESSOR	2006-01-09	49275.00	Male	Black
3	Police	SENIOR POLICE OFFICER	1997-05-27	75942.10	Male	Hispanic
4	Police	SENIOR POLICE OFFICER	2006-01-23	69355.26	Male	White

```
[8]: s = pd.cut(emp['salary'], bins=range(0, 300000, 25000))
df_cat = s.value_counts().reset_index()
df_cat.columns = ['Salary Range', 'Count']
df_cat
```

	Salary Range	Count
0	(25000, 50000]	9289
1	(50000, 75000]	9230
2	(75000, 100000]	3950
3	(100000, 125000]	588
4	(125000, 150000]	134
5	(150000, 175000]	72
6	(175000, 200000]	38
7	(0, 25000]	34
8	(200000, 225000]	5
9	(250000, 275000]	4
10	(225000, 250000]	1

```
[9]: fig, ax = plt.subplots(figsize=(6, 4))
ax.pie('Count', labels='Salary Range', data=df_cat);
```



11.2 5. Plotting with Pandas

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

Exercise 1

In this Exercise we will test whether daily returns from stocks are normally distributed. Complete the following tasks:

- * Take the `stocks` DataFrame and call the `pct_change` method to get the daily return percentage and assign it to a variable.
- * Assign the mean and standard deviation of each column (these will return Series) to separate variables.
- * Standardize your columns by subtracting the mean and dividing by the standard deviation. You have now produced a z-score for each daily return.
- * Add a column to this DataFrame called `noise` by calling `np.random.randn` which creates random normal variables.
- * Plot the KDE for each column in your DataFrame. If the stock returns are normal, then the shapes of the curves will all look the same.
- * Limit the xaxis to be between -3 and 3.
- * Are stock returns normally distributed?

```
[11]: stocks = pd.read_csv('../data/stocks/stocks10.csv', index_col='date',
   ↪parse_dates=['date'])
stocks.head(3)
```

	MSFT	AAPL	SLB	AMZN	TSLA	XOM	WMT	T	FB	V
date										
1999-10-25	29.84	2.32	17.02	82.75	NaN	21.45	38.99	16.78	NaN	NaN
1999-10-26	29.82	2.34	16.65	81.25	NaN	20.89	37.11	17.28	NaN	NaN
1999-10-27	29.33	2.38	16.52	75.94	NaN	20.80	36.94	18.27	NaN	NaN

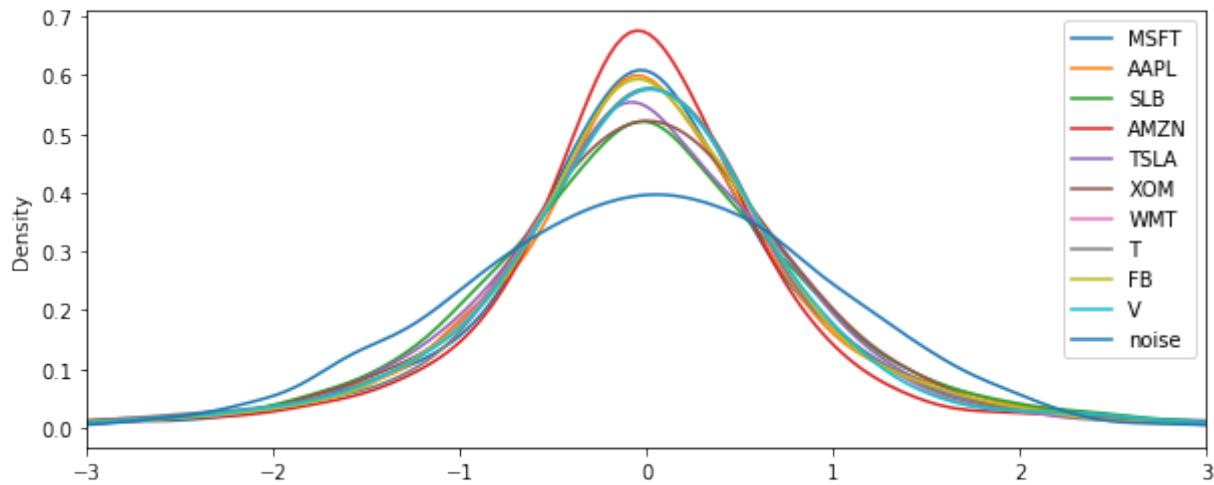
```
[12]: stock_pct_change = stocks.pct_change()
stock_pct_change.head(3)
```

	MSFT	AAPL	SLB	AMZN	TSLA	XOM	WMT	T	FB	V
date										
1999-10-25	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1999-10-26	-0.000670	0.008621	-0.021739	-0.018127	NaN	-0.026107	-0.048217	0.029797	NaN	NaN
1999-10-27	-0.016432	0.017094	-0.007808	-0.065354	NaN	-0.004308	-0.004581	0.057292	NaN	NaN

```
[13]: mean = stock_pct_change.mean()
std = stock_pct_change.std()
stock_normal = (stock_pct_change - mean) / std
stock_normal['noise'] = np.random.randn(len(stocks))
stock_normal.head()
```

	MSFT	AAPL	SLB	AMZN	TSLA	...	WMT	T	FB	V	noise
date											
1999-10-25	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	0.227129
1999-10-26	0.060696	0.286300	0.994959	0.577644	NaN	...	3.231718	1.827176	NaN	NaN	0.638465
1999-10-27	0.885861	0.616294	0.368504	1.992386	NaN	...	0.327185	3.529553	NaN	NaN	0.478651
1999-10-28	0.596791	0.768740	0.173134	1.983325	NaN	...	3.419352	5.133515	NaN	NaN	0.143004
1999-10-29	1.544427	1.072441	1.663113	0.194959	NaN	...	0.663056	0.639230	NaN	NaN	0.572474

```
[14]: stock_normal.plot(kind='kde', xlim=[-3, 3], figsize=(10, 4));
```



No, There is quite a bit more area under the curve between -1 and 1 standard deviations. Also, there is a lot more area outside of 3 standard deviations (can't see this from the graph).

Exercise 2

Use Pandas to plot a horizontal bar plot of diamond cuts.

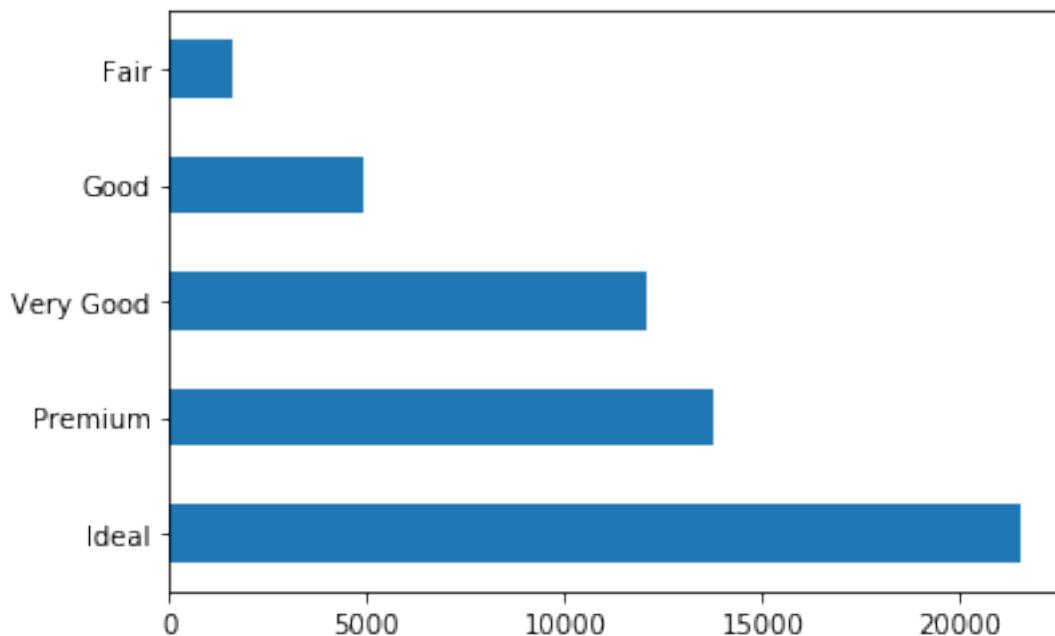
```
[15]: diamonds = pd.read_csv('../data/diamonds.csv')
diamonds.head(3)
```

	carat	cut	color	clarity	depth	table	price	x	y	z
0	0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2.43
1	0.21	Premium	E	SI1	59.8	61.0	326	3.89	3.84	2.31
2	0.23	Good	E	VS1	56.9	65.0	327	4.05	4.07	2.31

```
[16]: cut_count = diamonds['cut'].value_counts()
cut_count
```

```
[16]: Ideal      21551
Premium    13791
Very Good  12082
Good       4906
Fair        1610
Name: cut, dtype: int64
```

```
[17]: cut_count.plot(kind='barh');
```



```
[18]: plt.style.use(['ggplot'])
```

Exercise 3

Make a visualization that easily shows the differences in average salary by sex for each department of the employee dataset.

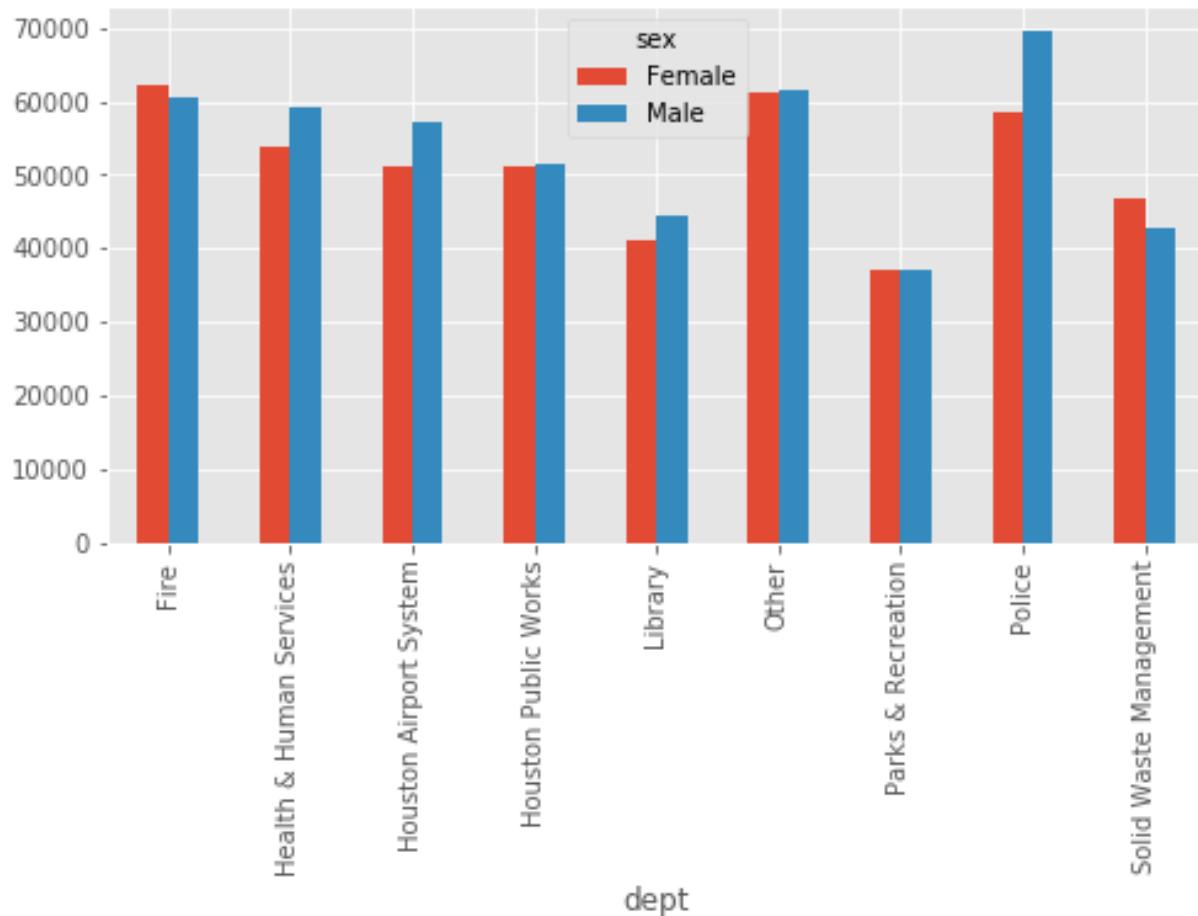
```
[19]: emp = pd.read_csv('../data/employee.csv', parse_dates=['hire_date'])
emp.head(3)
```

	dept	title	hire_date	salary	sex	race
0	Police	POLICE SERGEANT	2001-12-03	87545.38	Male	White
1	Other	ASSISTANT CITY ATTORNEY II	2010-11-15	82182.00	Male	Hispanic
2	Houston Public Works	SENIOR SLUDGE PROCESSOR	2006-01-09	49275.00	Male	Black

```
[20]: dept_sex_sal = emp.pivot_table(index='dept', columns='sex', values='salary')
dept_sex_sal.head(3)
```

dept	sex	Female	Male
Fire	Fire	62212.637250	60479.306862
Health & Human Services	Health & Human Services	53838.310780	59230.425956
Houston Airport System	Houston Airport System	51099.300226	57278.306598

```
[21]: dept_sex_sal.plot(kind='bar', figsize=(8, 4));
```



Exercise 4

Split the employee data into two separate DataFrames. Those who have a hire date after the year 2000 and those who have one before. Make the same plot above for each group.

```
[22]: criteria = emp['hire_date'].dt.year >= 2000
```

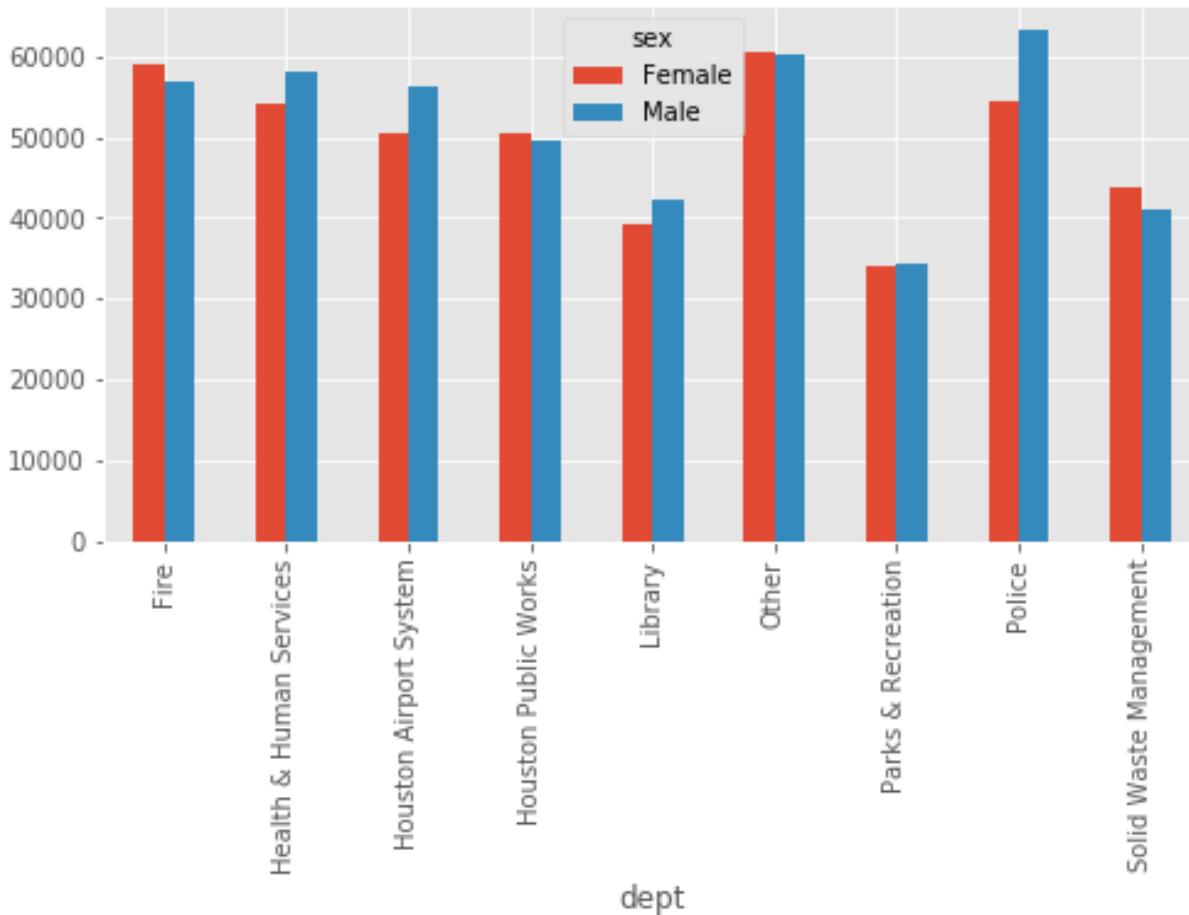
```
[23]: emp_2000_and_later = emp[criteria]
emp_2000_and_later.head(3)
```

	dept		title	hire_date	salary	sex	race
0	Police	POLICE SERGEANT	2001-12-03	87545.38	Male	White	
1	Other	ASSISTANT CITY ATTORNEY II	2010-11-15	82182.00	Male	Hispanic	
2	Houston Public Works	SENIOR SLUDGE PROCESSOR	2006-01-09	49275.00	Male	Black	

```
[24]: dept_gender_sal_2000_and_later = emp_2000_and_later.pivot_table(index='dept',
                                                               columns='sex',
                                                               values='salary')
dept_gender_sal_2000_and_later.head(3)
```

dept	sex	Female	Male
Fire	Female	59185.247973	57016.700243
Health & Human Services	Female	54267.524020	58113.680000
Houston Airport System	Female	50402.349398	56314.303226

```
[25]: dept_gender_sal_2000_and_later.plot(kind='bar', figsize=(8, 4));
```



```
[26]: emp_before_2000 = emp[~criteria]
emp_before_2000.head(3)
```

	dept	title	hire_date	salary	sex	race
3	Police	SENIOR POLICE OFFICER	1997-05-27	75942.10	Male	Hispanic
9	Police	SENIOR POLICE OFFICER	1993-08-30	75942.10	Male	Black
17	Police	POLICE COMMANDER	1983-02-07	115821.42	Male	White

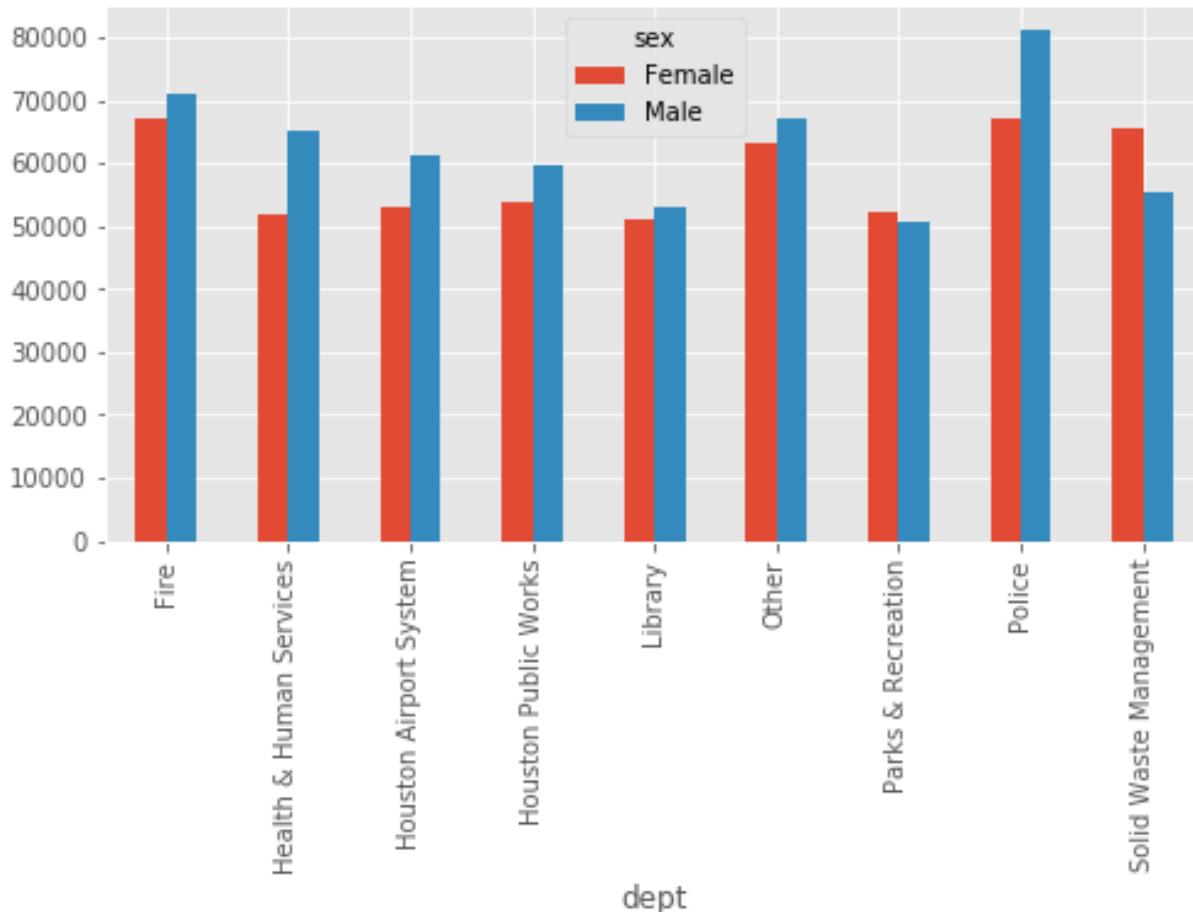
```
[27]: dept_sex_sal_before_2000 = emp_before_2000.pivot_table(index='dept',
                                                               columns='sex',
                                                               values='salary')

dept_sex_sal_before_2000.head(3)
```

dept	sex	Female	Male
Fire	Female	67082.785217	71182.220495
Health & Human Services	Female	51989.924731	65284.364561
Houston Airport System	Female	53183.873874	61184.725490

Can drop missing values so that we can compare

```
[28]: dept_sex_sal_before_2000.dropna().plot(kind='bar', figsize=(8, 4));
```



Execute the next cell to read in the flights dataset and use it for the remaining exercises.

```
[29]: flights = pd.read_csv('../data/flights.csv', parse_dates=['date'])
flights.head(3)
```

	date	airline	origin	dest	dep_time	...	carrier_delay	weather_delay	nas_delay	security_delay	late_aircraft_delay
0	2018-01-01	UA	LAS	IAH	100	...	0	0	0	0	0
1	2018-01-01	WN	DEN	PHX	515	...	0	0	0	0	0
2	2018-01-01	B6	JFK	BOS	550	...	0	83	8	0	0

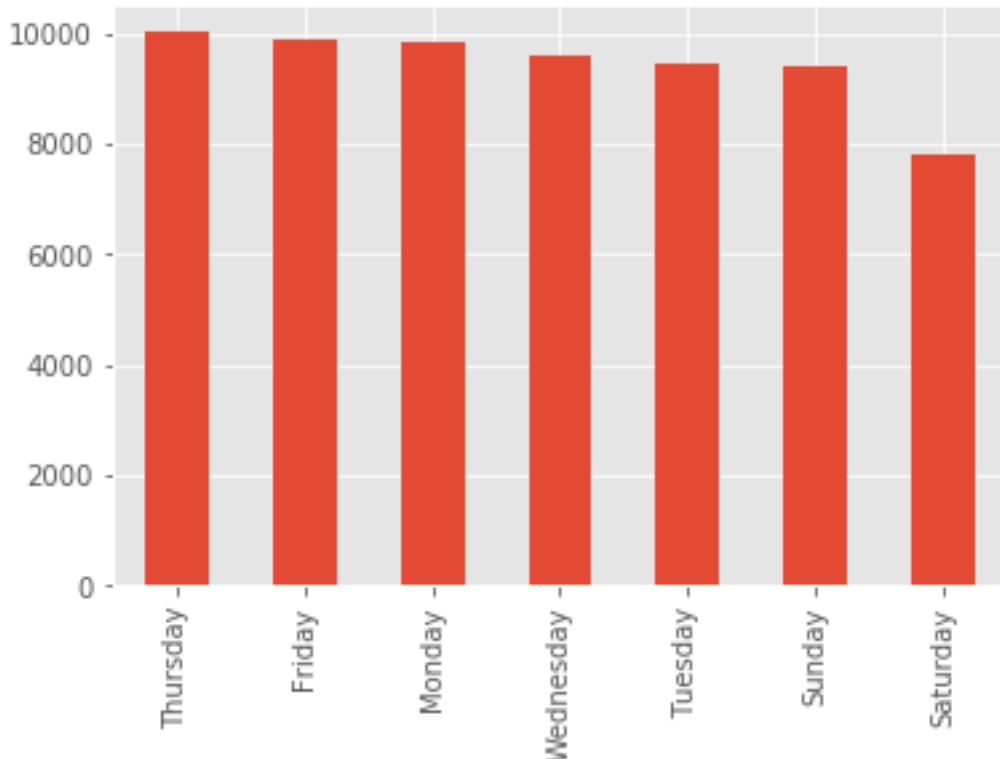
Exercise 5

Use the `flights` data set. Plot the counts of the number of flights per day of week.

```
[30]: weekday_ct = flights['date'].dt.day_name().value_counts()
weekday_ct
```

```
[30]: Thursday      10022
Friday         9872
Monday        9828
Wednesday     9581
Tuesday       9428
Sunday        9410
Saturday      7782
Name: date, dtype: int64
```

```
[31]: weekday_ct.plot(kind='bar');
```

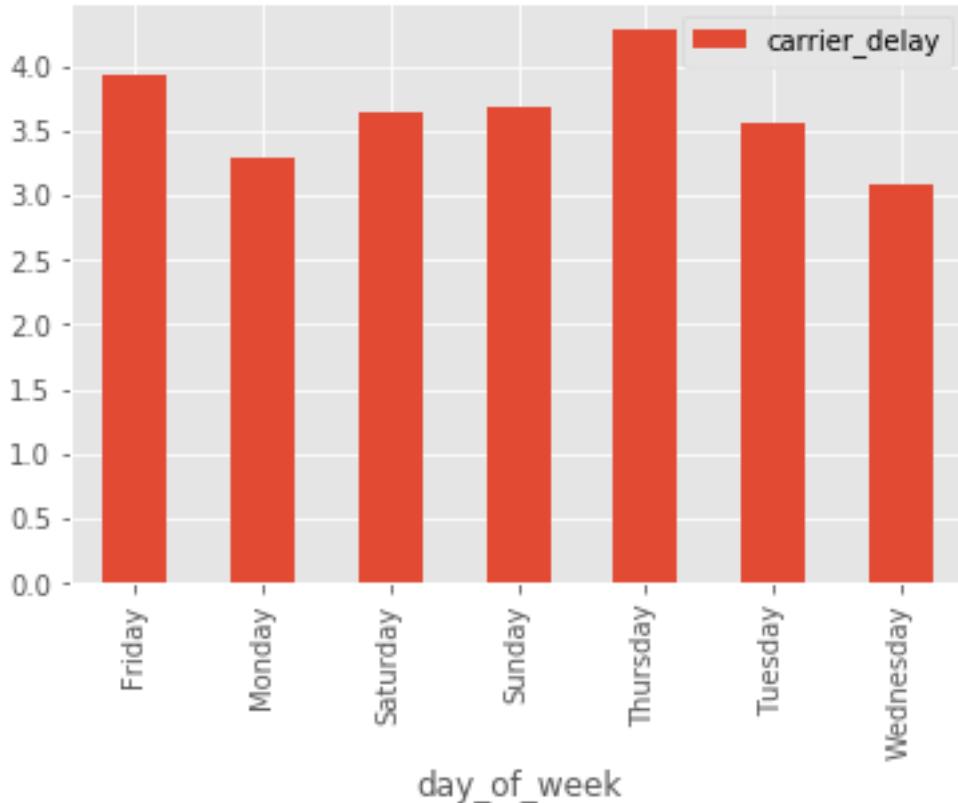


Exercise 6

Plot the average carrier delay per day of week.

```
[32]: flights['day_of_week'] = flights['date'].dt.day_name()
```

```
[33]: flights.groupby('day_of_week').agg({'carrier_delay': 'mean'}).plot(kind='bar');
```



Exercise 7

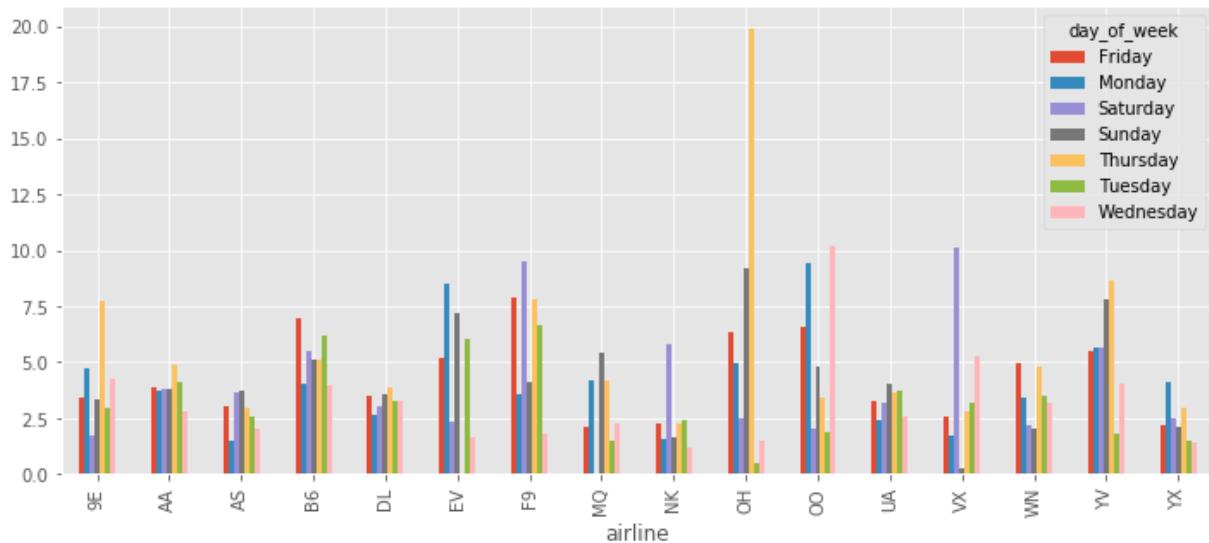
Plot the average carrier delay per day of week per airline.

```
[34]: airline_weekday_delay = flights.pivot_table(index='airline',
                                                 columns='day_of_week',
                                                 values='carrier_delay')

airline_weekday_delay.head(3)
```

day_of_week	Friday	Monday	Saturday	Sunday	Thursday	Tuesday	Wednesday
airline							
9E	3.408805	4.696774	1.755556	3.313725	7.735849	2.993333	4.233918
AA	3.900165	3.697871	3.835072	3.829536	4.883819	4.090094	2.808836
AS	3.000000	1.477407	3.622871	3.706140	2.920949	2.602679	2.053215

```
[35]: airline_weekday_delay.plot(kind='bar', figsize=(12, 5));
```



11.3 6. Seaborn