Master Data Analysis with Python Solutions

by
Ted Petrou

Contents

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')
```

```
As the xlwt package is no longer maintained, the xlwt engine will be removed in a future
version of pandas. This is the only engine in pandas that supports writing in the xls
format. Install openpyxl and write to an xlsx file instead.

: boolean
    use_inf_as_null had been deprecated and will be removed in a future
    version. Use `use_inf_as_na` instead.


/Users/Ted/miniconda3/lib/python3.8/site-packages/pandas/_config/config.py:630:
FutureWarning: As the xlwt package is no longer maintained, the xlwt engine will be
removed in a future version of pandas. This is the only engine in pandas that supports
writing in the xls format. Install openpyxl and write to an xlsx file instead.
  warnings.warn(d.msg, FutureWarning)
/Users/Ted/miniconda3/lib/python3.8/site-packages/pandas/_config/config.py:630:
FutureWarning:
: boolean
    use_inf_as_null had been deprecated and will be removed in a future
    version. Use `use_inf_as_na` instead.

  warnings.warn(d.msg, FutureWarning)
```

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

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.

[8]:
```
bikes.info(memory_usage='deep')
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50089 entries, 0 to 50088
Data columns (total 11 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   gender             50089 non-null  object
 1   starttime          50089 non-null  object
 2   stoptime           50089 non-null  object
 3   tripduration       50089 non-null  int64
 4   from_station_name  50089 non-null  object
 5   start_capacity     50083 non-null  float64
 6   to_station_name    50089 non-null  object
 7   end_capacity       50077 non-null  float64
 8   temperature        50089 non-null  float64
 9   wind_speed         50089 non-null  float64
 10  events             50089 non-null  object
dtypes: float64(4), int64(1), object(6)
memory usage: 23.0 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?

[9]:
```
movies = pd.read_csv('../data/movie.csv', index_col='director_name')
movies.head(3)
```

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

3 rows × 21 columns

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.

[10]:
```
movies = pd.read_csv('../data/movie.csv')
movies = movies.set_index('title', drop=False)
movies.head(3)
```

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

3 rows × 22 columns

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.

[11]:
```
movies = pd.read_csv('../data/movie.csv', index_col='title')
index = movies.index
index[-10:]
```

[11]:
```
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?

[12]:
```
movies = pd.read_csv('../data/movie.csv', index_col=-5)
movies.head(3)
```

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

3 rows × 21 columns

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)
```

title	director_name	director_fb
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)
```

title	director_name
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)
```

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

3 rows × 21 columns

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]
```

title	actor1	actor2	actor3
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 all of the movies beginning at 'Home Alone' and ending at 'Top Gun'.

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

title	actor1	actor2	actor3
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', :]
```

[10]:
```
year                                                    2012.0
color                                                    Color
content_rating                                           PG-13
duration                                                 164.0
director_name                                  Christopher Nolan
director_fb                                            22000.0
actor1                                             Tom Hardy
actor1_fb                                              27000.0
actor2                                           Christian Bale
actor2_fb                                              23000.0
actor3                                       Joseph Gordon-Levitt
actor3_fb                                              23000.0
gross                                                448130642.0
genres                                           Action|Thriller
num_reviews                                              813.0
num_voted_users                                        1144337
plot_keywords      deception|imprisonment|lawlessness|police offi…
language                                                English
country                                                     USA
budget                                               250000000.0
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'], :]
```

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

1 rows × 21 columns

Exercise 7

Select all columns for the movies 'Tangled' and 'Avatar'.

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

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

2 rows × 21 columns

Alternatively, omit the empty slice.

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

Exercise 8

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

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

title	year	imdb_score
Tangled	2010.0	7.8
Avatar	2009.0	7.9

Exercise 9

What is the data type of the `year` column?

[16]:
```
movie.dtypes.head() # Int64
```

[16]:
```
year              float64
color              object
content_rating     object
duration          float64
director_name      object
dtype: object
```

Exercise 10

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

[17]:
```
# 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):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   year             4810 non-null   float64
 1   color            4897 non-null   object
 2   content_rating   4616 non-null   object
 3   duration         4901 non-null   float64
 4   director_name    4814 non-null   object
 5   director_fb      4814 non-null   float64
 6   actor1           4909 non-null   object
 7   actor1_fb        4909 non-null   float64
 8   actor2           4903 non-null   object
 9   actor2_fb        4903 non-null   float64
 10  actor3           4893 non-null   object
 11  actor3_fb        4893 non-null   float64
 12  gross            4054 non-null   float64
 13  genres           4916 non-null   object
 14  num_reviews      4867 non-null   float64
 15  num_voted_users  4916 non-null   int64
 16  plot_keywords    4764 non-null   object
 17  language         4904 non-null   object
 18  country          4911 non-null   object
 19  budget           4432 non-null   float64
 20  imdb_score       4916 non-null   float64
dtypes: float64(10), int64(1), object(10)
memory usage: 974.0+ KB
```

Exercise 11

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

[18]:
```
# Forrest Gump is not a multiple of 300 away from Tangled
movie.loc['Tangled':'Forrest Gump':300]
```

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

3 rows × 21 columns

2.3 3. Selecting Subsets of Data from DataFrames with `iloc`

Exercise 1

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

[19]:
```
movie.iloc[:, [10, 5, 1]].head(3)
```

title	actor3	director_fb	color
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.

[20]:
```
movie.iloc[[10, 5, 1], :]
```

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

3 rows × 21 columns

Exercise 3

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

[21]:
```
movie.iloc[100:105, 5]
```

[21]:
```
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.

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

[22]:
```
'Rob Cohen'
```

Exercise 5

Return the result of exercise 4 as a DataFrame.

[23]:
```
movie.iloc[[100], [4]]
```

title	director_name
The Fast and the Furious	Rob Cohen

Exercise 6

Select the last 5 rows of the last 5 columns.

[24]:
```
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	6.3
Shanghai Calling	NaN	English	USA	NaN	6.3
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 200 along with every fifth column.

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

title	year	director_fb	actor3	num_voted_users	imdb_score
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.

[26]:
```
movie.iloc[:, 7].head(3)
```

[26]:
```
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.

[27]:
```
rows = [999, 99, 9]
cols = [9, 19]
movie.iloc[rows, cols]
```

title	actor2_fb	budget
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

[28]:
```
import pandas as pd
movie = pd.read_csv('../data/movie.csv', index_col='title')
duration = movie['duration']
duration.head()
```

[28]:
```
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'?

[29]:
```
duration.loc['Titanic']
```

[29]:
```
194.0
```

Exercise 2

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

[30]:
```
duration.iloc[999]
```

[30]:
```
90.0
```

Exercise 3

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

[31]:
```
names = ['Hulk', 'Toy Story', 'Cars']
duration.loc[names]
```

[31]:
```
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'.

[32]:
```
duration.loc['Hulk':'Cars':100]
```

[32]:
```
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.

[33]:
```
duration.iloc[-100::10]
```

[33]:
```
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

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

[34]:
```
bikes = pd.read_csv('../data/bikes.csv')
wind = bikes['wind_speed']
wind.head()
```

[34]:
```
0    12.7
1     6.9
2    16.1
3    16.1
4    17.3
Name: wind_speed, dtype: float64
```

Exercise 6

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

It has a `RangeIndex`

[35]:
```
wind.index
```

[35]:
```
RangeIndex(start=0, stop=50089, step=1)
```

Exercise 7

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

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

[36]:
```
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?

[37]:
```
wind.loc[4:10]
```

[37]:
```
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

[38]:
```
bikes = pd.read_csv('../data/bikes.csv')
bikes.head(3)
```

	gender	starttime	stoptime	tripduration	from_station_name	...	to_station_name	end_capacity	temperature	wind_speed	events
0	Male	2013-06-28 19:01:00	2013-06-28 19:17:00	993	Lake Shore Dr & Monroe St	...	Michigan Ave & Oak St	15.0	73.9	12.7	mostlycloudy
1	Male	2013-06-28 22:53:00	2013-06-28 23:03:00	623	Clinton St & Washington Blvd	...	Wells St & Walton St	19.0	69.1	6.9	partlycloudy
2	Male	2013-06-30 14:43:00	2013-06-30 15:01:00	1040	Sheffield Ave & Kingsbury St	...	Dearborn St & Monroe St	23.0	73.0	16.1	mostlycloudy

3 rows × 11 columns

Exercise 1

Find all the rides with temperature below 0.

[39]:
```
filt = bikes['temperature'] < 0
bikes[filt].head(3)
```

	gender	starttime	stoptime	tripduration	from_station_name	...	to_station_name	end_capacity	temperature	wind_speed	events
1871	Male	2013-12-12 05:13:00	2013-12-12 05:27:00	878	California Ave & North Ave	...	Carpenter St & Huron St	19.0	-2.0	6.9	mostlycloudy
2049	Female	2014-01-23 06:15:00	2014-01-23 06:29:00	828	Stave St & Armitage Ave	...	Ashland Ave & Division St	19.0	-2.0	16.1	partlycloudy
2054	Male	2014-01-23 21:15:00	2014-01-23 21:21:00	351	LaSalle St & Illinois St	...	McClurg Ct & Illinois St	31.0	-0.9	12.7	clear

3 rows × 11 columns

Exercise 2

Find all the rides with wind speed greater than 30.

[40]:
```
filt = bikes['wind_speed'] > 30
bikes[filt].head(3)
```

	gender	starttime	stoptime	tripduration	from_station_name	...	to_station_name	end_capacity	temperature	wind_speed	events
2164	Male	2014-02-20 19:06:00	2014-02-20 19:07:00	66	Pine Grove Ave & Waveland Ave	...	Pine Grove Ave & Waveland Ave	23.0	46.9	31.1	mostlycloudy
2165	Male	2014-02-20 20:47:00	2014-02-20 21:14:00	1605	Millennium Park	...	Clark St & Wrightwood Ave	15.0	39.0	35.7	cloudy
2479	Male	2014-04-01 08:28:00	2014-04-01 08:29:00	82	Desplaines St & Kinzie St	...	Desplaines St & Kinzie St	19.0	33.1	31.1	mostlycloudy

3 rows × 11 columns

Exercise 3

Find all the rides that began from station 'Millennium Park'.

[41]:
```
bikes['from_station_name'] == 'Millennium Park'
bikes[filt].head(3)
```

	gender	starttime	stoptime	tripduration	from_station_name	...	to_station_name	end_capacity	temperature	wind_speed	events
2164	Male	2014-02-20 19:06:00	2014-02-20 19:07:00	66	Pine Grove Ave & Waveland Ave	...	Pine Grove Ave & Waveland Ave	23.0	46.9	31.1	mostlycloudy
2165	Male	2014-02-20 20:47:00	2014-02-20 21:14:00	1605	Millennium Park	...	Clark St & Wrightwood Ave	15.0	39.0	35.7	cloudy
2479	Male	2014-04-01 08:28:00	2014-04-01 08:29:00	82	Desplaines St & Kinzie St	...	Desplaines St & Kinzie St	19.0	33.1	31.1	mostlycloudy

3 rows × 11 columns

Exercise 4

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

[42]:
```
# these are likely missing values.
# The dataset uses -9999 to represent missing values
filt = bikes['wind_speed'] < 0
bikes[filt].head(3)
```

	gender	starttime	stoptime	tripduration	from_station_name	...	to_station_name	end_capacity	temperature	wind_speed	events
22990	Male	2016-03-19 10:08:00	2016-03-19 10:20:00	702	Wood St & Division St	...	Campbell Ave & Fullerton Ave	15.0	42.8	-9999.0	mostlycloudy
27168	Female	2016-06-30 11:47:00	2016-06-30 11:51:00	240	Kimball Ave & Belmont Ave	...	Avers Ave & Belmont Ave	19.0	-9999.0	-9999.0	unknown
28368	Female	2016-07-21 21:02:29	2016-07-21 21:20:28	1079	Sedgwick St & North Ave	...	Ashland Ave & Division St	19.0	73.0	-9999.0	tstorms

3 rows × 11 columns

Exercise 5

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

[43]:
```
filt = bikes['start_capacity'] > 50
bikes[filt].head()
```

	gender	starttime	stoptime	tripduration	from_station_name	...	to_station_name	end_capacity	temperature	wind_speed	events
36122	Male	2017-02-17 17:00:36	2017-02-17 17:23:27	1371	Field Museum	...	Lake Shore Dr & North Blvd	39.0	63.0	10.4	partlycloudy
37617	Male	2017-04-14 18:44:47	2017-04-14 19:00:53	966	Field Museum	...	Burnham Harbor	23.0	63.0	4.6	cloudy
37920	Male	2017-04-22 12:28:51	2017-04-22 12:44:14	923	Field Museum	...	Indiana Ave & Roosevelt Rd	39.0	55.9	12.7	mostlycloudy
37940	Female	2017-04-22 17:12:55	2017-04-22 17:23:44	649	Field Museum	...	Wabash Ave & Roosevelt Rd	23.0	57.9	12.7	partlycloudy
39102	Male	2017-05-21 20:40:10	2017-05-21 20:51:29	679	Field Museum	...	Buckingham Fountain	27.0	51.1	12.7	cloudy

5 rows × 11 columns

Exercise 6

Did any rides happen in temperature over 100 degrees?

[44]:
```
# no, a dataframe with no rows is returned.
filt = bikes['temperature'] > 100
bikes[filt]
```

gender	starttime	stoptime	tripduration	from_station_name	...	to_station_name	end_capacity	temperature	wind_speed	events

0 rows × 11 columns

Read in new data

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

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

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

3 rows × 21 columns

Exercise 7

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

[46]:
```
filt = movie['actor1'] == 'Tom Hanks'
hanks_movies = movie[filt]
hanks_movies.head(3)
```

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

3 rows × 21 columns

He's starred in 24 movies

[47]:
```
hanks_movies.shape
```

[47]:
```
(24, 21)
```

Exercise 8

Select movies with an IMDB score greater than 9.

[48]:
```
filt = movie['imdb_score'] > 9
movie[filt]
```

title	year	color	content_rating	duration	director_name	...	plot_keywords	language	country	budget	imdb_score
The Shawshank Redemption	1994.0	Color	R	142.0	Frank Darabont	...	escape from prison|first person narration|pris...	English	USA	25000000.0	9.3
Towering Inferno	NaN	Color	NaN	65.0	John Blanchard	...	NaN	English	Canada	NaN	9.5
Dekalog	NaN	Color	TV-MA	55.0	NaN	...	meaning of life|moral challenge|morality|searc...	Polish	Poland	NaN	9.1
The Godfather	1972.0	Color	R	175.0	Francis Ford Coppola	...	crime family|mafia|organized crime|patriarch|r...	English	USA	6000000.0	9.2
Kickboxer: Vengeance	2016.0	NaN	NaN	90.0	John Stockwell	...	NaN	NaN	USA	17000000.0	9.1

5 rows × 21 columns

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'.

[49]:
```
def count_rating(rating):
    filt = movie['content_rating'] == rating
    count = len(movie[filt])
    return f'There are {count} movies rated {rating}'
```

[50]:
```
count_rating('R')
```

[50]:
```
'There are 2067 movies rated R'
```

[51]:
```
count_rating('PG-13')
```

[51]:
```
'There are 1411 movies rated PG-13'
```

[52]:
```
count_rating('PG')
```

[52]:
```
'There are 686 movies rated PG'
```

2.6 6. Boolean Selection Multiple Conditions

[53]:
```
import pandas as pd
bikes = pd.read_csv('../data/bikes.csv')
bikes.head(3)
```

	gender	starttime	stoptime	tripduration	from_station_name	...	to_station_name	end_capacity	temperature	wind_speed	events
0	Male	2013-06-28 19:01:00	2013-06-28 19:17:00	993	Lake Shore Dr & Monroe St	...	Michigan Ave & Oak St	15.0	73.9	12.7	mostlycloudy
1	Male	2013-06-28 22:53:00	2013-06-28 23:03:00	623	Clinton St & Washington Blvd	...	Wells St & Walton St	19.0	69.1	6.9	partlycloudy
2	Male	2013-06-30 14:43:00	2013-06-30 15:01:00	1040	Sheffield Ave & Kingsbury St	...	Dearborn St & Monroe St	23.0	73.0	16.1	mostlycloudy

3 rows × 11 columns

Exercise 1

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

[54]:
```
filt1 = bikes['temperature'] >= 0
filt2 = bikes['temperature'] <= 2
filt = filt1 & filt2
bikes[filt].head(3)
```

	gender	starttime	stoptime	tripduration	from_station_name	...	to_station_name	end_capacity	temperature	wind_speed	events
1870	Female	2013-12-11 21:13:00	2013-12-11 21:19:00	347	Southport Ave & Roscoe St	...	Ashland Ave & Grace St	15.0	1.9	10.4	clear
1938	Male	2013-12-23 22:29:00	2013-12-23 22:37:00	455	State St & Pearson St	...	Clark St & Schiller St	19.0	-0.0	15.0	partlycloudy
2039	Male	2014-01-21 08:48:00	2014-01-21 09:13:00	1547	Damen Ave & Pierce Ave	...	Franklin St & Jackson Blvd	31.0	1.9	19.6	partlycloudy

3 rows × 11 columns

Exercise 2

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

[55]:
```
filt1 = bikes['tripduration'] < 100
filt2 = bikes['gender'] == 'Female'
filt = filt1 & filt2
bikes[filt].head(3)
```

	gender	starttime	stoptime	tripduration	from_station_name	...	to_station_name	end_capacity	temperature	wind_speed	events
2485	Female	2014-04-01 15:43:00	2014-04-01 15:44:00	67	Halsted St & Polk St	...	Halsted St & Polk St	19.0	46.0	23.0	partlycloudy
3366	Female	2014-05-17 14:55:00	2014-05-17 14:56:00	62	Damen Ave & Chicago Ave	...	Damen Ave & Chicago Ave	15.0	57.9	11.5	partlycloudy
3629	Female	2014-05-27 07:55:00	2014-05-27 07:56:00	70	Clark St & Schiller St	...	Clark St & Schiller St	19.0	73.0	9.2	cloudy

3 rows × 11 columns

Exercise 3

Find all the rides from 'Daley Center Plaza' to 'Michigan Ave & Washington St'.

[56]:
```
filt1 = bikes['from_station_name'] == 'Daley Center Plaza'
filt2 = bikes['to_station_name'] == 'Michigan Ave & Washington St'
filt = filt1 & filt2
bikes[filt]
```

	gender	starttime	stoptime	tripduration	from_station_name	...	to_station_name	end_capacity	temperature	wind_speed	events
6276	Female	2014-08-04 17:18:00	2014-08-04 17:22:00	214	Daley Center Plaza	...	Michigan Ave & Washington St	43.0	73.0	8.1	cloudy
31038	Female	2016-09-07 07:48:30	2016-09-07 07:53:27	297	Daley Center Plaza	...	Michigan Ave & Washington St	43.0	79.0	10.4	cloudy
46116	Male	2017-09-13 16:29:00	2017-09-13 16:32:42	222	Daley Center Plaza	...	Michigan Ave & Washington St	43.0	72.0	9.2	cloudy

3 rows × 11 columns

Exercise 4

Find all the rides with temperature greater than 90 or trip duration greater than 2000 or wind speed greater than 20.

[57]:
```
filt1 = bikes['temperature'] > 90
filt2 = bikes['tripduration'] > 2000
filt3 = bikes['wind_speed'] > 20
filt = filt1 | filt2 | filt3
bikes[filt].head(3)
```

	gender	starttime	stoptime	tripduration	from_station_name	...	to_station_name	end_capacity	temperature	wind_speed	events
12	Male	2013-07-05 10:02:00	2013-07-05 10:40:00	2263	Jefferson St & Monroe St	...	Jefferson St & Monroe St	19.0	79.0	0.0	partlycloudy
18	Male	2013-07-09 13:12:00	2013-07-09 14:42:00	5396	Canal St & Jackson Blvd	...	Millennium Park	35.0	79.0	13.8	cloudy
40	Female	2013-07-14 14:08:00	2013-07-14 15:53:00	6274	Wabash Ave & Roosevelt Rd	...	Lake Shore Dr & Monroe St	11.0	87.1	8.1	partlycloudy

3 rows × 11 columns

Exercise 5

Invert the condition from exercise 4.

[58]:
```
filt1 = bikes['temperature'] > 90
filt2 = bikes['tripduration'] > 2000
```

```
filt3 = bikes['wind_speed'] > 20
filt = filt1 | filt2 | filt3
bikes[~filt].head(3)
```

	gender	starttime	stoptime	tripduration	from_station_name	...	to_station_name	end_capacity	temperature	wind_speed	events
0	Male	2013-06-28 19:01:00	2013-06-28 19:17:00	993	Lake Shore Dr & Monroe St	...	Michigan Ave & Oak St	15.0	73.9	12.7	mostlycloudy
1	Male	2013-06-28 22:53:00	2013-06-28 23:03:00	623	Clinton St & Washington Blvd	...	Wells St & Walton St	19.0	69.1	6.9	partlycloudy
2	Male	2013-06-30 14:43:00	2013-06-30 15:01:00	1040	Sheffield Ave & Kingsbury St	...	Dearborn St & Monroe St	23.0	73.0	16.1	mostlycloudy

3 rows × 11 columns

Exercise 6

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

[59]:
```
# no
filt1 = bikes['events'] == 'snow'
filt2 = bikes['temperature'] > 40
filt = filt1 & filt2
bikes[filt]
```

gender	starttime	stoptime	tripduration	from_station_name	...	to_station_name	end_capacity	temperature	wind_speed	events

0 rows × 11 columns

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

[60]:
```
import pandas as pd
movie = pd.read_csv('../data/movie.csv', index_col='title')
movie.head(3)
```

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

3 rows × 21 columns

Exercise 7

Select all movies with an IMDB score between 8 and 9.

[61]:
```
filt1 = movie['imdb_score'] >= 8
filt2 = movie['imdb_score'] <= 9
filt = filt1 & filt2
movie[filt].head(3)
```

title	year	color	content_rating	duration	director_name	...	plot_keywords	language	country	budget	imdb_score
The Dark Knight Rises	2012.0	Color	PG-13	164.0	Christopher Nolan	...	deception|imprisonment|lawlessness|police offi...	English	USA	250000000.0	8.5
The Avengers	2012.0	Color	PG-13	173.0	Joss Whedon	...	alien invasion|assassin|battle|iron man|soldier	English	USA	220000000.0	8.1
Captain America: Civil War	2016.0	Color	PG-13	147.0	Anthony Russo	...	based on comic book|knife|marvel cinematic uni...	English	USA	250000000.0	8.2

3 rows × 21 columns

Exercise 8

Select all movies rated 'PG-13' that had IMDB scores between 8 and 9.

[62]:
```
filt1 = movie['imdb_score'] >= 8
filt2 = movie['imdb_score'] <= 9
filt3 = movie['content_rating'] == 'PG-13'
```

```
filt = filt1 & filt2 & filt3
movie[filt].head(3)
```

title	year	color	content_rating	duration	director_name	...	plot_keywords	language	country	budget	imdb_score
The Dark Knight Rises	2012.0	Color	PG-13	164.0	Christopher Nolan	...	deception|imprisonment|lawlessness|police offi...	English	USA	250000000.0	8.5
The Avengers	2012.0	Color	PG-13	173.0	Joss Whedon	...	alien invasion|assassin|battle|iron man|soldier	English	USA	220000000.0	8.1
Captain America: Civil War	2016.0	Color	PG-13	147.0	Anthony Russo	...	based on comic book|knife|marvel cinematic uni...	English	USA	250000000.0	8.2

3 rows × 21 columns

Exercise 9

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

[63]:
```
filt = movie['content_rating'].isin(['R', 'PG-13', 'PG'])
movie[filt].head(3)
```

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

3 rows × 21 columns

Exercise 10

Select movies that are either rated PG-13 or had an IMDB score greater than 7.

[64]:
```
filt1 = movie['content_rating'] == 'PG-13'
filt2 = movie['imdb_score'] > 7
filt = filt1 | filt2
movie[filt].head(3)
```

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

3 rows × 21 columns

Exercise 11

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

[65]:
```
filt1 = movie['actor1_fb'] > 10000
filt2 = movie['actor2_fb'] > 10000
filt3 = movie['actor3_fb'] > 10000
filt = filt1 | filt2 | filt3
movie[filt].head(3)
```

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

3 rows × 21 columns

Exercise 12

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

The following selects non-PG-13 movies with IMDB score 7 or less.

[66]:
```
filt1 = movie['content_rating'] == 'PG-13'
filt2 = movie['imdb_score'] > 7
filt = filt1 | filt2
movie[~filt].head(3)
```

title	year	color	content_rating	duration	director_name	...	plot_keywords	language	country	budget	imdb_score
The Chronicles of Narnia: Prince Caspian	2008.0	Color	PG	150.0	Andrew Adamson	...	brother brother relationship|brother sister re...	English	USA	225000000.0	6.6
Alice in Wonderland	2010.0	Color	PG	108.0	Tim Burton	...	alice in wonderland|mistaking reality for drea...	English	USA	200000000.0	6.5
Oz the Great and Powerful	2013.0	Color	PG	130.0	Sam Raimi	...	circus|magic|magician|oz|witch	English	USA	215000000.0	6.4

3 rows × 21 columns

Exercise 13

Select all movies from the 1970's.

[67]:
```
filt1 = movie['year'] >= 1970
filt2 = movie['year'] <= 1979
filt = filt1 & filt2
movie[filt].head(3)
```

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

3 rows × 21 columns

2.7 7. Boolean Selection More

[68]:
```
import pandas as pd
bikes = pd.read_csv('../data/bikes.csv')
```

Exercise 1

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

[69]:
```
wind = bikes['wind_speed']
wind.head(3)
```

[69]:
```
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...

[70]:
```
filt = wind < 0
wind[filt].head(3)
```

[70]:
```
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.

[71]:
```
filt = wind.between(12, 16)
wind[filt].head(3)
```

[71]:
```
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.

[72]:
```
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.

[73]:
```
import pandas as pd
movie = pd.read_csv('../data/movie.csv', index_col='title')
movie.head(3)
```

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

3 rows × 21 columns

Exercise 4

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

There are none!

[74]:
```
filt = movie['actor2_fb'] > movie['actor2_fb']
movie[filt]
```

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

0 rows × 21 columns

Exercise 5

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

[75]:
```
filt1 = movie['year'] == 2016
filt2 = movie['imdb_score'] < 4
filt = filt1 & filt2
cols = ['year', 'content_rating', 'imdb_score']

movie.loc[filt, cols]
```

title	year	content_rating	imdb_score
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.

[76]:
```
filt = movie['content_rating'].isna()
movie[filt].head(3)
```

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

3 rows × 21 columns

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.

[77]:
```
filt = movie['gross'].isna() & movie['budget'].isna()
cols = ['gross', 'budget']
movie.loc[filt, cols].head(3)
```

title	gross	budget
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.

[78]:
```
def find_missing(df, col1, col2):
    filt = df[col1].isna() & df[col2].isna()
    cols = [col1, col2]

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

[79]:
```
movie_missing = find_missing(movie, 'gross', 'budget')
movie_missing.head(3)
```

title	gross	budget
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 5,000 and 10,000.

[80]:
```
bikes.query('5000 <= tripduration <= 10000').head(3)
```

	gender	starttime	stoptime	tripduration	from_station_name	...	to_station_name	end_capacity	temperature	wind_speed	events
18	Male	2013-07-09 13:12:00	2013-07-09 14:42:00	5396	Canal St & Jackson Blvd	...	Millennium Park	35.0	79.0	13.8	cloudy
40	Female	2013-07-14 14:08:00	2013-07-14 15:53:00	6274	Wabash Ave & Roosevelt Rd	...	Lake Shore Dr & Monroe St	11.0	87.1	8.1	partlycloudy
77	Female	2013-07-21 11:35:00	2013-07-21 13:54:00	8299	State St & 19th St	...	Sheffield Ave & Kingsbury St	15.0	82.9	5.8	mostlycloudy

3 rows × 11 columns

Exercise 2

Use the `query` method to select trip durations between 5,000 and 10,000 when the weather was snow or rain. Retrieve the same data with boolean selection.

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

	gender	starttime	stoptime	tripduration	from_station_name	...	to_station_name	end_capacity	temperature	wind_speed	events
8506	Male	2014-10-04 12:33:00	2014-10-04 14:06:00	5568	Halsted St & Diversey Pkwy	...	Halsted St & Wrightwood Ave	15.0	42.1	17.3	rain
13355	Male	2015-06-15 11:41:00	2015-06-15 13:43:00	7295	Racine Ave & Belmont Ave	...	Racine Ave & Belmont Ave	15.0	75.9	4.6	rain
22155	Male	2016-02-09 10:09:00	2016-02-09 12:28:00	8309	Wabash Ave & Roosevelt Rd	...	Museum Campus	35.0	16.0	16.1	snow

3 rows × 11 columns

[82]:
```
filt1 = bikes['tripduration'].between(5000, 10000)
filt2 = bikes['events'].isin(['snow', 'rain'])
filt = filt1 & filt2
bikes[filt].head(3)
```

	gender	starttime	stoptime	tripduration	from_station_name	...	to_station_name	end_capacity	temperature	wind_speed	events
8506	Male	2014-10-04 12:33:00	2014-10-04 14:06:00	5568	Halsted St & Diversey Pkwy	...	Halsted St & Wrightwood Ave	15.0	42.1	17.3	rain
13355	Male	2015-06-15 11:41:00	2015-06-15 13:43:00	7295	Racine Ave & Belmont Ave	...	Racine Ave & Belmont Ave	15.0	75.9	4.6	rain
22155	Male	2016-02-09 10:09:00	2016-02-09 12:28:00	8309	Wabash Ave & Roosevelt Rd	...	Museum Campus	35.0	16.0	16.1	snow

3 rows × 11 columns

Exercise 3

Use the `query` method to select trip durations between 5,000 and 10,000 when it was snow or rain. Create a list outside of the `query` method to hold the weather and reference that variable with `@` within `query`.

[83]:
```
weather = ["snow", "rain"]
bikes.query('5000 <= tripduration <= 10000 and events in @weather').head(3)
```

	gender	starttime	stoptime	tripduration	from_station_name	...	to_station_name	end_capacity	temperature	wind_speed	events
8506	Male	2014-10-04 12:33:00	2014-10-04 14:06:00	5568	Halsted St & Diversey Pkwy	...	Halsted St & Wrightwood Ave	15.0	42.1	17.3	rain
13355	Male	2015-06-15 11:41:00	2015-06-15 13:43:00	7295	Racine Ave & Belmont Ave	...	Racine Ave & Belmont Ave	15.0	75.9	4.6	rain
22155	Male	2016-02-09 10:09:00	2016-02-09 12:28:00	8309	Wabash Ave & Roosevelt Rd	...	Museum Campus	35.0	16.0	16.1	snow

3 rows × 11 columns

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

[84]:
```
import pandas as pd
pd.set_option('display.max_columns', 50)
movie = pd.read_csv('../data/movie.csv', index_col='title')
movie.head(3)
```

title	year	color	content_rating	duration	director_name	director_fb	actor1	actor1_fb	actor2	actor2_fb	actor3	actor3_fb	gross	genres	num_reviews	num_voted_users	plot_keywords	language	country	budget	imdb_score
Avatar	2009.0	Color	PG-13	178.0	James Cameron	0.0	CCH Pounder	1000.0	Joel David Moore	936.0	Wes Studi	855.0	760505847.0	Action|Adventure|Fantasy|Sci-Fi	723.0	886204	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	563.0	Johnny Depp	40000.0	Orlando Bloom	5000.0	Jack Davenport	1000.0	309404152.0	Action|Adventure|Fantasy	302.0	471220	goddess|marriage ceremony|marriage proposal|pi...	English	USA	300000000.0	7.1
Spectre	2015.0	Color	PG-13	148.0	Sam Mendes	0.0	Christoph Waltz	11000.0	Rory Kinnear	393.0	Stephanie Sigman	161.0	200074175.0	Action|Adventure|Thriller	602.0	275868	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.

[85]:
```
movie.query('actor1_fb + actor2_fb + actor3_fb > 50000').head(3)
```

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

3 rows × 21 columns

Exercise 5

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

[86]:
```
movie.query('num_voted_users < 10 * num_reviews').head(3)
```

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

3 rows × 21 columns

Exercise 6

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

[87]:
```
movie.query('1990 <= year <= 1999 and content_rating == "R" and imdb_score > 8').
↪head(3)
```

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

3 rows × 21 columns

2.9 9. Miscellaneous Subset Selection

[88]:
```
import pandas as pd
bikes = pd.read_csv('../data/bikes.csv')
```

Exercise 1

Provide several example column names that are not possible to select using dot notation.

- `min` or any other pandas method
- `some column` or any name with spaces
- `4` or any other integer
- `3cities` or any string that begins with an integer
- `(1, 2, 3)` - a tuple or any other object that is not a string

Exercise 2

Use the `%time` magic function to compare the performance difference between `loc` and `at` and between `iloc` and `iat`.

[89]:
```
%time bikes.at[35103, 'tripduration']
```

```
CPU times: user 2.45 ms, sys: 316 µs, total: 2.77 ms
Wall time: 2.31 ms
```

[89]:
```
487
```

[90]:
```
%time bikes.loc[35103, 'tripduration']
```

```
CPU times: user 63 µs, sys: 0 ns, total: 63 µs
Wall time: 67 µs
```

[90]:
```
487
```

[91]:
```
%time bikes.iat[35103, 4]
```

```
CPU times: user 125 µs, sys: 1e+03 ns, total: 126 µs
Wall time: 131 µs
```

[91]:
```
'Clinton St & Madison St'
```

[92]:
```
%time bikes.iloc[35103, 4]
```

```
CPU times: user 136 µs, sys: 1e+03 ns, total: 137 µs
Wall time: 140 µs
```

[92]:
```
'Clinton St & Madison St'
```

[93]:
```
# numpy
values = bikes['tripduration'].values
%time values[35103]
```

```
CPU times: user 7 µs, sys: 1e+03 ns, total: 8 µs
Wall time: 11.9 µs
```

[93]: 487

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)
```

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

3 rows × 21 columns

[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()
```

```
/var/folders/0x/whw882fj4qv0czzqzrngxvdh0000gn/T/ipykernel_39000/2575230201.py:4:
FutureWarning: Boolean inputs to the `inclusive` argument are deprecated infavour of
`both` or `neither`.
  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]:
```
0.1625711960943912
```

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 6?

[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 as a Series. 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 10?

[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 the `sum` 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 Toothed 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 Toothed 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

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 the results.

[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 Toothed 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 9

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 10

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
```

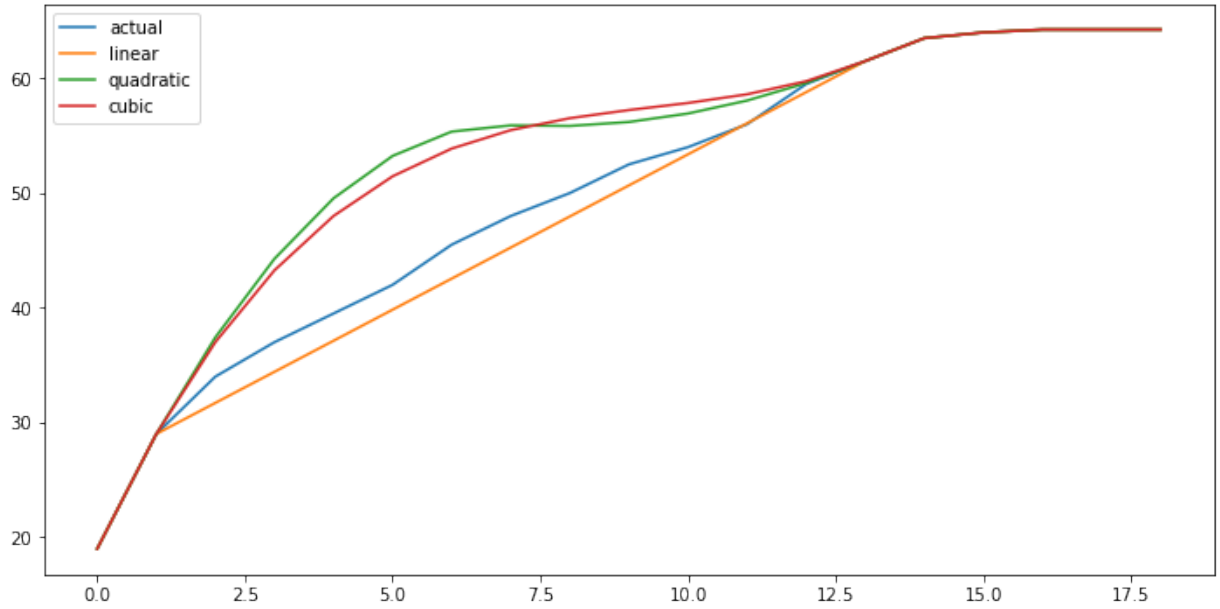
	age	height	height_na
0	0	19.00	19.00
1	1	29.00	29.00
2	2	34.00	NaN
3	3	37.00	NaN
4	4	39.50	NaN
5	5	42.00	NaN
6	6	45.50	NaN
7	7	48.00	NaN
8	8	50.00	NaN
9	9	52.50	NaN
10	10	54.00	NaN
11	11	56.00	NaN
12	12	59.50	NaN
13	13	61.50	61.50
14	14	63.50	63.50
15	15	64.00	64.00
16	16	64.25	64.25
17	17	64.25	64.25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

[42]:
```
hna = girl_height['height_na']
linear = hna.interpolate('linear')
quadratic = hna.interpolate('quadratic')
cubic = hna.interpolate('cubic')
```

Exercise 11

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

[43]:
```
%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)
```



3.3 3. Series Sorting, Ranking, and Uniqueness

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

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

3 rows × 21 columns

Exercise 1

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

[45]:
```
movie['num_reviews'].sort_values(ascending=False).head()
```

[45]:
```
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`.

[46]:
```
movie['actor1'].nunique()
```

[46]:
```
2095
```

[47]:
```
movie['actor2'].nunique()
```

[47]:
```
3030
```

[48]:
```
movie['actor3'].nunique()
```

[48]:
```
3519
```

Exercise 3

Select the `year` column, sort it, and drop any duplicates.

[49]:
```
movie['year'].sort_values().drop_duplicates().head()
```

[49]:
```
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?

[50]:
```
movie['year'].drop_duplicates().sort_values().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
Pandora's Box                                       1929.0
Name: year, dtype: float64
```

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

[51]:
```
%timeit -n 5 movie['year'].sort_values().drop_duplicates().head()
```

```
834 µs ± 49.2 µs per loop (mean ± std. dev. of 7 runs, 5 loops each)
```

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

```
480 µs ± 95 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?

[53]:
```
# yes, they are the same
movie['duration'].rank(ascending=False).sort_values().head(10)
```

[53]:
```
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
```

[54]:
```
movie['duration'].sort_values(ascending=False).head(10)
```

[54]:
```
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 appear first. Output the first 10 values.

[55]:
```
movie['actor1'].sort_values(na_position='first').head(10)
```

[55]:
```
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

[56]:
```
import pandas as pd
emp = pd.read_csv('../data/employee.csv')
movie = pd.read_csv('../data/movie.csv', index_col='title')
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.

[57]:
```
emp['salary'].agg(['min', 'max', 'mean', 'median', 'std'])
```

[57]:
```
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 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.

[58]:
```
imax = emp['salary'].idxmax()
imin = emp['salary'].idxmin()
rows = [imax, imin]
rows
```

[58]:
```
[1732, 1183]
```

[59]:
```
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.

[60]:
```
score = movie['imdb_score']
rows = [score.idxmax(), score.idxmin()]
```

```
rows
```

[60]:
```
['Towering Inferno', 'Justin Bieber: Never Say Never']
```

[61]:
```
movie.loc[rows, :]
```

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

2 rows × 21 columns

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.

[62]:
```
score = movie['imdb_score']
score.agg(['min', 'idxmin', 'max', 'idxmax'])
```

[62]:
```
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` column.

[63]:
```
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

[64]:
```
ac = msft['adjusted_close']
ac.pct_change().nlargest()
```

[64]:
```
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.

[65]:
```
movie['actor1'].sample(n=3, random_state=12345)
```

[65]:
```
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 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.

[66]:
```
emp['title'].head()
```

[66]:
```
0               POLICE SERGEANT
1    ASSISTANT CITY ATTORNEY II
2       SENIOR SLUDGE PROCESSOR
3         SENIOR POLICE OFFICER
4         SENIOR POLICE OFFICER
Name: title, dtype: object
```

[67]:
```
vals = ['POLICE OFFICER', 'SENIOR POLICE OFFICER']
emp['title'].replace(vals, 'POLICE').head()
```

[67]:
```
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

[68]:
```
import pandas as pd
movie = pd.read_csv('../data/movie.csv', index_col='title')
actor1 = movie['actor1'].dropna()
actor1.head(3)
```

[68]:
```
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?

[69]:
```
actor1.value_counts().head()
```

[69]:
```
Robert De Niro    48
Johnny Depp       36
Nicolas Cage      32
J.K. Simmons      29
Matt Damon        29
Name: actor1, dtype: int64
```

[70]:
```
actor1.value_counts().index[0]
```

[70]:
```
'Robert De Niro'
```

Exercise 2

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

[71]:
```
actor1.value_counts(normalize=True).iloc[:100].sum()
```

[71]:
```
0.325117131798737
```

Exercise 3

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

[72]:
```
(actor1.value_counts() == 1).sum()
```

[72]:
```
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.

[73]:
```
filt = actor1.str.count('e') > 3
unique_e = actor1[filt].unique()
unique_e
```

[73]:
```
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)
```

[74]:
```
len(unique_e)
```

[74]:
```
35
```

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

[75]:
```
filt = actor1.str.count('[eE]') > 3
unique_eE = actor1[filt].unique()
unique_eE
```

[75]:
```
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)
```

[76]:
```
len(unique_eE)
```

[76]:
```
45
```

Exercise 5

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

[77]:
```
filt = actor1.str.contains('Johnson')
actor1[filt].drop_duplicates()
```

[77]:
```
title
Miami Vice                                Don Johnson
San Andreas                             Dwayne Johnson
The Boy in the Striped Pajamas        Richard Johnson
The Work and the Glory                   Eric Johnson
The Texas Chainsaw Massacre 2            Bill Johnson
Jimmy and Judy                  Nicole Randall Johnson
Fabled                              R. Brandon Johnson
Name: actor1, dtype: object
```

Exercise 6

How many unique actor 1 names end in 'x'?

[78]:
```
filt = actor1.str.endswith('x')
act_endx = actor1[filt].drop_duplicates()
act_endx
```

[78]:
```
title
Independence Day: Resurgence                        Vivica A. Fox
Total Recall                                            Ronny Cox
The Warrior's Way                                        Tony Cox
Sex Tape                                               James Wilcox
Duplex                                                Justin Theroux
The Pianist                                            Emilia Fox
Suspect Zero                                          Harry Lennix
Jeepers Creepers II                                    Nicki Aycox
Molly                                                 Elaine Hendrix
City of Ghosts                                            Kirk Fox
Dickie Roberts: Former Child Star                     Kevin Grevioux
Alien Zone                                             Bernard Fox
A Nightmare on Elm Street 5: The Dream Child           Lisa Wilcox
The Perfect Wave                                      Rachel Hendrix
Cotton Comes to Harlem                                     Redd Foxx
Name: actor1, dtype: object
```

[79]:
```
len(act_endx)
```

[79]:
```
15
```

Exercise 7

The pandas string methods overlap with the built-in 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.

[80]:
```
python_str = {method for method in dir(str) if method[0] != '_'}
pandas_str = {method for method in dir(actor1.str) if method[0] != '_'}
```

[81]:
```
python_str & pandas_str
```

[81]:
```
{'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'}
```

[82]:
```
python_str - pandas_str
```

[82]:
```
{'expandtabs',
 'format',
 'format_map',
 'isascii',
 'isidentifier',
 'isprintable',
 'maketrans',
 'splitlines'}
```

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

[83]:
```
{'cat',
 'contains',
 'decode',
 'extract',
 'extractall',
 'findall',
 'fullmatch',
 'get',
 'get_dummies',
 'len',
 'match',
 'normalize',
 'pad',
 'repeat',
 'slice',
 'slice_replace',
```

```
 'wrap'}
```

3.6 6. Datetime Series Methods

[84]:
```
bikes = pd.read_csv('../data/bikes.csv', parse_dates=['starttime', 'stoptime'])
start = bikes['starttime']
```

Exercise 1

What percentage of bike rides happen in January?

[85]:
```
# 2.7%
start.dt.month_name().value_counts(normalize=True).round(3)
```

[85]:
```
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
```

[86]:
```
# or
(start.dt.month == 1).mean() * 100
```

[86]:
```
2.7191598953862126
```

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
29    0.017070
34    0.017070
45    0.016950
5     0.016890
11    0.016870
14    0.016870
36    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
46    0.016471
2     0.016471
4     0.016391
```

```
23    0.016331
54    0.016311
57    0.016291
3     0.016251
28    0.016091
59    0.016071
35    0.016071
56    0.016031
0     0.015932
58    0.015912
31    0.015872
50    0.015872
55    0.015852
27    0.015812
9     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.iloc[0]
last = aapl.iloc[-1]
n = aapl_change.size
mean, first, last, n
```

[96]:
```
(0.0012692984527780725, 2.32, 242.46, 5032)
```

[97]:
```
aapl_change.agg(['mean'])
```

[97]:
```
mean    0.001269
Name: AAPL, dtype: float64
```

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.

[98]:
```
first * (mean + 1) ** n
```

[98]:
```
1372.8262535261679
```

[99]:
```
last
```

[99]:
```
242.46
```

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?

[100]:
```
std = aapl_change.std()
z_score_raw = (aapl_change - mean) / std
z_score_raw.head()
```

[100]:
```
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
```

[101]:
```
z_score_raw.max(), z_score_raw.min()
```

[101]:
```
(5.371252085913816, -20.221564477921543)
```

Or with agg.

[102]:
```
z_score_raw.agg(['min', 'max']).round(3)
```

[102]:
```
min   -20.222
max     5.371
Name: AAPL, dtype: float64
```

Exercise 5

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

[103]:
```
aapl_change[z_score_raw == z_score_raw.max()]
```

[103]:
```
date
2008-10-13    0.139188
Name: AAPL, dtype: float64
```

[104]:
```
aapl_change.agg(['max', 'idxmax'])
```

[104]:
```
max                  0.139188
idxmax    2008-10-13 00:00:00
Name: AAPL, dtype: object
```

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.

[105]:
```
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 = round((z_score_abs < 1).mean(), 3)
    pct_within2 = round((z_score_abs < 2).mean().round(3), 3)
    pct_within3 = round((z_score_abs < 3).mean().round(3), 3)

    return pct_within1, pct_within2, pct_within3
```

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

[106]:
```
(0.833, 0.95, 0.98)
```

[107]:
```
stock_pct_finder(stocks['FB'])
```

[107]:
```
(0.804, 0.956, 0.988)
```

[108]:
```
stock_pct_finder(stocks['SLB'])
```

[108]:
```
(0.763, 0.948, 0.988)
```

Exercise 7

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

[109]:
```
filt = (aapl > 100) & (aapl < 120)
filt.sum()
```

[109]:
```
401
```

Exercise 8

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

[110]:
```
((aapl < 50) | (aapl > 150)).sum()
```

[110]:
```
3596
```

Exercise 9

Look up the definition for interquartile range and select all Apple closing prices that are within this range. There are multiple ways to do this. Check the `quantile` method.

[111]:
```
# using the quantile method
q1 = aapl.quantile(.25)
q3 = aapl.quantile(.75)

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

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

[111]:
```
date
2000-03-01    4.06
2000-03-03    3.99
2000-03-06    3.92
2000-03-10    3.92
2000-03-21    4.21
Name: AAPL, dtype: float64
```

[112]:
```
# 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()
```

[112]:
```
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
```

Exercise 10

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

[113]:
```
max_close_date = aapl.idxmax()
max_close_date
```

[113]:
```
Timestamp('2019-10-23 00:00:00')
```

Number of trading days

[114]:
```
aapl.loc[max_close_date:].size
```

[114]:
```
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 64-bit float columns from the `college` DataFrame. How many are there?

[2]:
```
college.select_dtypes('float64').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)
```

title	actor1_fb	actor2_fb	actor3_fb
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()
```

title	actor1_fb	actor2_fb	actor3_fb
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

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
relaffil            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

3 rows × 26 columns

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()
```

```
/var/folders/0x/whw882fj4qv0czzqzrngxvdh0000gn/T/ipykernel_39119/625090225.py:4:
FutureWarning: Dropping of nuisance columns in DataFrame reductions (with
'numeric_only=None') is deprecated; in a future version this will raise TypeError.
Select only valid columns before calling the reduction.
  max_vals = college.max()
```

[42]:
```
city         Zanesville
stabbr               WY
hbcu                1.0
menonly             1.0
womenonly           1.0
```

```
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

3 rows × 26 columns

4.4 4. DataFrame Sorting, Ranking, and Uniqueness

[44]:
```
import pandas as pd
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 frequency of occurrence of all race and sex combinations using the trick from exercise 3. 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()
```

[51]:
```
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
```

Use the college dataset for the remaining exercises

Execute the following cell to read in the college dataset which sets the institution name (`instnm`) as the index.

[53]:
```
college = pd.read_csv('../data/college.csv', index_col='instnm')
college.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

3 rows × 26 columns

Exercise 5

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`.

[54]:
```
cols = ['stabbr', 'satvrmid', 'satmtmid', 'ugds']
college_tx = college[cols].query('stabbr == "TX" and ugds > 20000').dropna()
college_tx
```

instnm	stabbr	satvrmid	satmtmid	ugds
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.

[55]:
```
college_tx.rank(ascending=False)
```

instnm	stabbr	satvrmid	satmtmid	ugds
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 sorting by `ugds`.

[56]:
```
college[['stabbr', 'ugds']].sort_values('ugds', ascending=False) \
       .drop_duplicates(subset='stabbr').head(10)
```

instnm	stabbr	ugds
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.

[57]:
```
col_unique = college.nunique()
col_unique
```

[57]:
```
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.

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

[58]:
```
hbcu            2
menonly         2
womenonly       2
relaffil        2
distanceonly    2
curroper        2
dtype: int64
```

Can extract the index to get just the names.

[59]:
```
col_unique[col_unique == 2].index
```

[59]:
```
Index(['hbcu', 'menonly', 'womenonly', 'relaffil', 'distanceonly', 'curroper'],
dtype='object')
```

4.5 5. DataFrame Structure Methods

[60]:
```
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(3)
```

instnm	city	stabbr	relaffil	satvrmid	satmtmid	ugds
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

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?

[61]:
```
college_all['Verbal Higher'] = college_all['satvrmid'] > college_all['satmtmid']
```

[62]:
```
college_all['Verbal Higher'].mean()
```

[62]:
```
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.

[63]:
```
cols = ['satvrmid', 'satmtmid']
college_all[cols].isna().mean()
```

[63]:
```
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.

[64]:
```
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

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

[65]:
```
0.30574324324324326
```

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

[66]:
```
(sat['satvrmid'] == sat['satmtmid']).mean()
```

[66]:
```
0.08699324324324324
```

Exercise 3

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

[67]:
```
college_all['median all'] = college_all['ugds'].median()
college_all.head()
```

instnm	city	stabbr	relaffil	satvrmid	satmtmid	ugds	Verbal Higher	median all
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.

[68]:
```
college_all = college_all.rename(index={'Texas A & M University-College Station':
 ↪'TAMU'})
college_all.loc['TAMU']
```

[68]:
```
city             College Station
stabbr                        TX
relaffil                       0
satvrmid                   580.0
satmtmid                   615.0
ugds                     46941.0
Verbal Higher              False
median all                 412.5
Name: TAMU, dtype: object
```

Use the City of Houston employee dataset

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

[69]:
```
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 5

Create a new column named `bonus` and insert it right after the salary column equal. Have its value equal to 10% of the salary. Round the bonus to the nearest thousand.

[70]:
```
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

4.6 6. DataFrame Methods More

[71]:
```
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.

[72]:
```
dept_freq = emp['dept'].value_counts(normalize=True)
dept_freq.round(2)
```

[72]:
```
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
```

[73]:
```
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

[74]:
```
dept_freq_top100 = emp_top100['dept'].value_counts(normalize=True)
dept_freq_top100
```

[74]:
```
Other                     0.36
Fire                      0.22
Police                    0.15
Houston Airport System    0.09
Houston Public Works      0.09
Health & Human Services   0.07
Solid Waste Management    0.01
Library                   0.01
Name: dept, dtype: float64
```

[75]:
```
# The police dept makes up 31% of employees
# but only 16% of the top 100 salaries
dept_freq_top100 - dept_freq
```

[75]:
```
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.

[76]:
```
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

[77]:
```
stocks.pct_change().idxmin()
```

[77]:
```
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.

[78]:
```
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.

[79]:
```
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 columns both before and after the assignment.

[80]:
```
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

[81]:
```
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

[82]:
```
bikes.loc[filt, 'tripduration'] =  bikes['tripduration'] * 1.5
```

[83]:
```
bikes.loc[filt, 'tripduration']
```

[83]:
```
22306     292.50
22307    1188.00
22308     805.50
22309     497.25
Name: tripduration, dtype: float64
```

[84]:
```
bikes['tripduration']
```

[84]:
```
0          993.0
1          623.0
2         1040.0
3          667.0
4          130.0
           ...
50084     1625.0
50085      585.0
50086      824.0
50087      178.0
50088      214.0
Name: tripduration, Length: 50089, dtype: float64
```

Exercise 3

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

[85]:
```
bikes.iloc[:2, 5] = 0
bikes.head(2)
```

	gender	starttime	stoptime	tripduration	from_station_name	...	to_station_name	end_capacity	temperature	wind_speed	events
0	Male	2013-06-28 19:01:00	2013-06-28 19:17:00	993.0	Lake Shore Dr & Monroe St	...	Michigan Ave & Oak St	15.0	73.9	12.7	mostlycloudy
1	Male	2013-06-28 22:53:00	2013-06-28 23:03:00	623.0	Clinton St & Washington Blvd	...	Wells St & Walton St	19.0	69.1	6.9	partlycloudy

2 rows × 11 columns

Part V

Data Types

Chapter 5

Solutions

5.1 1. Integer, Float, and Boolean Data types

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

Exercise 1

Find the maximum number of a 16-bit integer using arithmetic operations. 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

Construct a Series that has the nullable integer data type. Make sure it has a mix of integers and missing values.

[4]:
```
s = pd.Series([pd.NA, np.nan, 5, -999], dtype='Int16')
s
```

[4]:
```
0    <NA>
1    <NA>
2       5
3    -999
dtype: Int16
```

Exercise 3

Take a look at the Series below. Change it's data type such that it uses the least amount of memory and preserves the numbers as they are.

[5]:
```
s = pd.Series([1_000, 60_000])
s
```

[5]:
```
0     1000
1    60000
dtype: int64
```

[6]:
```
s.astype('uint16')
```

[6]:
```
0     1000
1    60000
dtype: uint16
```

Exercise 4

Find the precision of a 32-bit float and then create a numpy array with values that have decimal places past that precision.

[7]:
```
np.finfo('float32')
```

[7]:
```
finfo(resolution=1e-06, min=-3.4028235e+38, max=3.4028235e+38, dtype=float32)
```

[8]:
```
np.array([1.00000001, 1.123456789], dtype='float32')
```

[8]:
```
array([1.       , 1.1234568], dtype=float32)
```

Exercise 5

Create a Series of numbers that have decimal places. Use the `astype` method to convert it to an integer and then back to a float. Are the decimals from the original Series preserved?

[9]:
```
# no
s = pd.Series([4.98, -23.123])
s
```

[9]:
```
0     4.980
1   -23.123
dtype: float64
```

[10]:
```
s.astype('int64')
```

[10]:
```
0     4
1   -23
dtype: int64
```

[11]:
```
s.astype('int64').astype('float64')
```

[11]:
```
0     4.0
1   -23.0
dtype: float64
```

Exercise 6

Create a numpy array with 8-bit unsigned integers. Use negative numbers in the construction along with numbers greater than 255. Does the output make sense?

[12]:
```
np.array([-1, 0, 1, 2, 255, 256], dtype='uint8')
```

[12]:
```
array([255,   0,   1,   2, 255,   0], dtype=uint8)
```

Exercise 7

Create a numpy array that has the values 50 and 100 in it, but do so without actually using those two values (or any operations that create them).

[13]:
```
# use the ability of numpy to wrap numbers around to the start of the range.
# there are 256 numbers in the range beginning at 0
np.array([306, 356], dtype='uint8')
```

[13]:
```
array([ 50, 100], dtype=uint8)
```

Exercise 8

Create a numpy array that contains two integers and the numpy nan missing value. Assign it to a variable name and output it to the screen. What data type is it?

[14]:
```
# notice the decimals in the output
a = np.array([99, -88, np.nan])
a
```

[14]:
```
array([ 99., -88.,  nan])
```

[15]:
```
a.dtype
```

[15]:
```
dtype('float64')
```

Exercise 9

Construct a Series from the array created in exercise 8. What data type is it? Construct a new Series with the same array forcing it to be a nullable integer.

[16]:
```
pd.Series(a)
```

[16]:
```
0    99.0
1   -88.0
2     NaN
dtype: float64
```

[17]:
```
pd.Series(a, dtype='Int64')
```

[17]:
```
0      99
1     -88
2    <NA>
dtype: Int64
```

Exercise 10

Construct a Series of 32-bit nullable integers using the data type object itself (and not the string).

[18]:
```
pd.Series([1, 5, pd.NA], dtype=pd.Int32Dtype())
```

[18]:
```
0       1
1       5
2    <NA>
dtype: Int32
```

5.2 2. Object, String, and Categorical Data Types

[19]:
```
import pandas as pd
import numpy as np
```

Exercise 1

Using its constructor, create a Series containing three two-item lists of integers. Then call the `sum` method on the Series. What is returned?

[20]:
```
s = pd.Series([[3, 4], [-9, 99], [5, 59]])
s
```

[20]:
```
0      [3, 4]
1    [-9, 99]
2     [5, 59]
dtype: object
```

A single list with all values together.

[21]:
```
s.sum()
```

[21]:
```
[3, 4, -9, 99, 5, 59]
```

Exercise 2

Use the constructor to create a Series of integers, floats, and booleans. Do not set the `dtype` parameter. What data type is your Series?

[22]:
```
# object
pd.Series([1, 3.2, True])
```

[22]:
```
0       1
1     3.2
2    True
dtype: object
```

Exercise 3

Construct a Series with the same values but force the data type to be a float. Does it work? What happens to the non-float values?

[23]:
```
# Yes, it works. other values become floats. True becomes 1.0
pd.Series([1, 3.2, True], dtype='float64')
```

[23]:
```
0    1.0
1    3.2
2    1.0
dtype: float64
```

Exercise 4

Construct a Series containing three strings and the four missing values `None`, `np.nan`, `pd.NA`, and `pd.NaT` assigning the result to a variable.

[24]:
```
s = pd.Series(['Houston', 'Rockets', 'Basketball', None, np.nan, pd.NA, pd.NaT])
s
```

[24]:
```
0       Houston
1       Rockets
2    Basketball
3          None
4           NaN
5          <NA>
6           NaT
dtype: object
```

[25]:
```
s.astype('string')
```

[25]:
```
0       Houston
1       Rockets
2    Basketball
3          <NA>
4          <NA>
5          <NA>
6          <NA>
dtype: string
```

Exercise 5

Using pandas, count the number of missing values in exercise 4.

[26]:
```
# pandas treats each one as a missing value
s.isna().sum()
```

[26]:
```
4
```

Exercise 6

Convert the Series from exercise 4 to the new string data type. Notice what happens to the missing values.

[27]:
```
# pandas only uses pd.NA for missing values in string columns
# object columns
s.astype('string')
```

[27]:
```
0       Houston
1       Rockets
2    Basketball
3          <NA>
4          <NA>
5          <NA>
6          <NA>
dtype: string
```

Read in the movie dataset

Execute the cell below to read in the first 10 columns of the movie dataset setting the index to be the title.

[28]:
```
pd.set_option('display.max_columns', 100)
movie = pd.read_csv('../data/movie.csv', index_col='title', usecols=range(10))
movie.head(3)
```

title	year	color	content_rating	duration	director_name	director_fb	actor1	actor1_fb	actor2
Avatar	2009.0	Color	PG-13	178.0	James Cameron	0.0	CCH Pounder	1000.0	Joel David Moore
Pirates of the Caribbean: At World's End	2007.0	Color	PG-13	169.0	Gore Verbinski	563.0	Johnny Depp	40000.0	Orlando Bloom
Spectre	2015.0	Color	PG-13	148.0	Sam Mendes	0.0	Christoph Waltz	11000.0	Rory Kinnear

Exercise 8

Which of the columns above are good candidates for the categorical data type?

It can be helpful to use the `value_counts` or `nunique` methods to get more information on the columns.

[29]:
```
movie['color'].value_counts()
```

[29]:
```
Color              4693
Black and White     204
Name: color, dtype: int64
```

[30]:
```
movie['content_rating'].nunique()
```

[30]:
```
18
```

[31]:
```
movie['director_name'].nunique()
```

[31]:
```
2397
```

[32]:
```
movie['actor1'].nunique()
```

[32]:
```
2095
```

[33]:
```
movie['actor2'].nunique()
```

[33]:
```
3030
```

I would make color and content_rating categorical as two have known, limited, and discrete values. Although the year column is discrete, it's not exactly limited as more years of data will come in the future. The

director_name, actor1, and actor2 columns are discrete and do repeat, but the number of unique values is quite substantial and a large percentage of the overall values. I would leave those as objects.

Exercise 9

Select the `content_rating` column as a Series and convert it to categorical. Assign the result to the variable `rating`.

[34]:
```
rating = movie['content_rating'].astype('category')
rating.head(3)
```

[34]:
```
title
Avatar                                        PG-13
Pirates of the Caribbean: At World's End      PG-13
Spectre                                       PG-13
Name: content_rating, dtype: category
Categories (18, object): ['Approved', 'G', 'GP', 'M', …, 'TV-Y', 'TV-Y7', 'Unrated',
'X']
```

Exercise 10

Write an expression that returns the number of categories.

[35]:
```
len(rating.cat.categories)
```

[35]:
```
18
```

Exercise 11

Prove that the `str` accessor still works with categorical columns by making the ratings lowercase.

[36]:
```
rating.str.lower().head()
```

[36]:
```
title
Avatar                                        pg-13
Pirates of the Caribbean: At World's End      pg-13
Spectre                                       pg-13
The Dark Knight Rises                         pg-13
Star Wars: Episode VII - The Force Awakens      NaN
Name: content_rating, dtype: object
```

Exercise 12

Assign the rating 'GGG' as the first value.

[37]:
```
rating = rating.cat.add_categories('GGG')
rating.loc['Avatar'] = 'GGG'
rating.head(3)
```

[37]:
```
title
Avatar                                          GGG
Pirates of the Caribbean: At World's End      PG-13
Spectre                                       PG-13
```

```
Name: content_rating, dtype: category
Categories (19, object): ['Approved', 'G', 'GP', 'M', …, 'TV-Y7', 'Unrated', 'X',
'GGG']
```

Exercise 13

Convert the following Series to integer.

[38]:
```
s = pd.Series(['1', '2'])
```

[39]:
```
s.astype('int64')
```

[39]:
```
0    1
1    2
dtype: int64
```

Exercise 14

Convert the following Series to integer.

[40]:
```
s = pd.Series(['1', '2', 'BAD DATA'])
```

[41]:
```
pd.to_numeric(s, errors='coerce')
```

[41]:
```
0    1.0
1    2.0
2    NaN
dtype: float64
```

Read in the diamonds dataset

Execute the next cell to read in the diamonds dataset.

[42]:
```
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

Exercise 15

Select the `cut` column as a Series and convert it to an ordered categorical. Use the data dictionary from above.

[43]:
```
categories = ['Fair', 'Good', 'Very Good', 'Premium', 'Ideal']
cut_dtype = pd.CategoricalDtype(categories, ordered=True)
cut_cat = diamonds['cut'].astype(cut_dtype)
cut_cat.head()
```

[43]:
```
0      Ideal
1    Premium
2       Good
3    Premium
4       Good
Name: cut, dtype: category
Categories (5, object): ['Fair' < 'Good' < 'Very Good' < 'Premium' < 'Ideal']
```

Exercise 16

By only knowing that `cut_cat` is an ordered categorical, write an expression to get the percentage of diamonds that have the lowest category.

[44]:
```
cut_cat.value_counts(normalize=True).sort_index().iloc[-1]
```

[44]:
```
0.3995365220615499
```

5.3 3. Datetime, Timedelta, and Period Data Types

[45]:
```
import numpy as np, pandas as pd
```

Exercise 1

Create a numpy array of datetimes with year precision for the years 2000, 2010, and 2020. Assign the result to a variable.

[46]:
```
# epoch is 1970
a = np.array([30, 40, 50], dtype='datetime64[Y]')
a
```

[46]:
```
array(['2000', '2010', '2020'], dtype='datetime64[Y]')
```

Exercise 2

Staying in numpy, convert the array created in exercise 1 to a data type with second precision and assign the result to a new variable.

[47]:
```
b = a.astype('datetime64[s]')
b
```

[47]:
```
array(['2000-01-01T00:00:00', '2010-01-01T00:00:00',
       '2020-01-01T00:00:00'], dtype='datetime64[s]')
```

Exercise 3

Staying in numpy, use the `astype` method to return the number of seconds after the epoch for each value from the array created in exercise 2.

[48]:
```
b.astype('int64')
```

[48]:
```
array([ 946684800, 1262304000, 1577836800])
```

Exercise 4

Use the integers from exercise 3 to in the numpy array constructor to get the same result as exercise 2.

[49]:
```
np.array([ 946684800, 1262304000, 1577836800], dtype='datetime64[s]')
```

[49]:
```
array(['2000-01-01T00:00:00', '2010-01-01T00:00:00',
       '2020-01-01T00:00:00'], dtype='datetime64[s]')
```

Exercise 5

Construct a Series of integers for the years 2000, 2010, and 2020. Then convert it to datetime with the `astype` method.

[50]:
```
s = pd.Series([30, 40, 50])
s
```

[50]:
```
0    30
1    40
2    50
dtype: int64
```

[51]:
```
s.astype('datetime64[Y]')
```

[51]:
```
0   2000-01-01
1   2010-01-01
2   2020-01-01
dtype: datetime64[ns]
```

Exercise 6

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

[52]:
```
s = pd.Series([1000000]).astype('datetime64[m]')
s
```

[52]:
```
0   1971-11-26 10:40:00
dtype: datetime64[ns]
```

[53]:
```
# get month as a string
s.dt.month_name()
```

[53]:
```
0    November
dtype: object
```

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

[54]:
```
pd.Timestamp(1000000, unit='m').month_name()
```

[54]:
```
'November'
```

Exercise 7

Construct a datetime Series using strings with precision down to nanoseconds (9 digits after the decimal)

[55]:
```
pd.Series(['2020-01-31 15:45:59.123456789',
           '2020-02-29 15:45:59.123456789'], dtype='datetime64[ns]')
```

[55]:
```
0   2020-01-31 15:45:59.123456789
1   2020-02-29 15:45:59.123456789
dtype: datetime64[ns]
```

Exercise 8

Using only arithmetic operations, find the amount of time 1 million seconds is. Report your answer as 'W days, X hours, Y minutes, Z seconds'.

[56]:
```
num = 1_000_000
seconds_in_day = 24 * 60 * 60
days, seconds_remaining = divmod(num, seconds_in_day)
days, seconds_remaining
```

[56]:
```
(11, 49600)
```

[57]:
```
seconds_in_hour = 60 * 60
hours, seconds_remaining = divmod(seconds_remaining, seconds_in_hour)
hours, seconds_remaining
```

[57]:
```
(13, 2800)
```

[58]:
```
seconds_in_minutes = 60
minutes, seconds = divmod(seconds_remaining, seconds_in_minutes)
minutes, seconds
```

[58]:
```
(46, 40)
```

[59]:
```
f'{days} days, {hours} hours, {minutes} minutes, {seconds} seconds'
```

[59]:
```
'11 days, 13 hours, 46 minutes, 40 seconds'
```

Exercise 9

Verify the results of exercise 8 by creating a pandas timedelta Series.

[60]:
```
pd.Series([1_000_000]).astype('timedelta64[s]')
```

[60]:
```
0   11 days 13:46:40
dtype: timedelta64[ns]
```

The `to_timedelta` function will be covered in the time series part.

[61]:
```
pd.to_timedelta(1_000_000, 's')
```

[61]:
```
Timedelta('11 days 13:46:40')
```

Exercise 10

Construct a Series with the data type period that has the hour 10 a.m. through 12 a.m. as the time period on January 1st for the years 2019, 2020, and 2021.

[62]:
```
pd.Series(['2019-01-01 10', '2020-01-01 10', '2020-01-01 10'], dtype='Period[h]')
```

[62]:
```
0    2019-01-01 10:00
1    2020-01-01 10:00
2    2020-01-01 10:00
dtype: period[H]
```

5.4 4. DataFrame Data Type Conversion

[63]:
```
import pandas as pd, numpy as np
```

Exercise 1

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?

[64]:
```
bikes = pd.read_csv('../data/bikes.csv')
td = bikes['tripduration']
td.head()
```

[64]:
```
0     993
1     623
2    1040
3     667
4     130
Name: tripduration, dtype: int64
```

[65]:
```
td.memory_usage()
```

[65]:
```
400840
```

Find the min and max values

[66]:
```
td.agg(['min', 'max'])
```

[66]:
```
min       60
max    86188
Name: tripduration, dtype: int64
```

Unfortunately an unsigned integer, 'uint16', doesn't quite have enough memory to fit the max.

[67]:
```
np.iinfo('uint16')
```

[67]:
```
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.

[68]:
```
np.iinfo('int32')
```

[68]:
```
iinfo(min=-2147483648, max=2147483647, dtype=int32)
```

[69]:
```
td2 = td.astype('int32')
```

32-bit integers take up half as much space as 64-bit integers. Let's verify this.

[70]:
```
td2.memory_usage(index=False) / td.memory_usage(index=False)
```

[70]:
```
0.5
```

Exercise 2

Read in the diamonds dataset and convert the data types of each column so they use the least amount of memory without losing any information.

[71]:
```
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

[72]:
```
np.finfo('float16')
```

[72]:
```
finfo(resolution=0.001, min=-6.55040e+04, max=6.55040e+04, dtype=float16)
```

[73]:
```
np.iinfo('uint16')
```

[73]:
```
iinfo(min=0, max=65535, dtype=uint16)
```

[74]:
```
diamonds.agg(['min', 'max'])
```

	carat	cut	color	clarity	depth	table	price	x	y	z
min	0.20	Fair	D	I1	43.0	43.0	326	0.00	0.0	0.0
max	5.01	Very Good	J	VVS2	79.0	95.0	18823	10.74	58.9	31.8

[75]:
```
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)

dtype_dict = {'carat': 'float32',
              'cut': cut_dtype,
              'color': color_dtype,
              'clarity': clarity_dtype,
```

```
              'carat': 'float32',
              'depth': 'float32',
              'table': 'float32',
              'price': 'uint16',
              'x': 'float32',
              'y': 'float32',
              'z': 'float32'}

diamonds2 = diamonds.astype(dtype_dict).round(3)
diamonds2.head(3)
```

	carat	cut	color	clarity	depth	table	price	x	y	z
0	0.23	Ideal	E	SI2	61.500000	55.0	326	3.95	3.98	2.43
1	0.21	Premium	E	SI1	59.799999	61.0	326	3.89	3.84	2.31
2	0.23	Good	E	VS1	56.900002	65.0	327	4.05	4.07	2.31

[76]:
```
np.float16(59.1281232)
```

[76]:
```
59.12
```

Part VI

Grouping Data

Chapter 6

Solutions

1. Groupby Aggregation Basics
2. Grouping and Aggregating with Multiple Columns
3. Grouping with Pivot Tables
4. Counting with Crosstabs
5. Alternate Groupby Syntax
6. Custom Aggregation
7. Transform and Filter with Groupby
8. Other Groupby Methods
9. Binning Numeric Columns
10. Miscellaneous Grouping Functionality
11. Create your own Data Analysis

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'))
```

	max_salary
sex	
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()
```

dept	median_salary
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

Exercise 4

Find the number of employees in each department.

It's not necessary to use a groupby.

[5]:
```
emp['dept'].value_counts()
```

[5]:
```
Police                      7573
Fire                        4376
Houston Public Works        4190
Other                       3373
Health & Human Services     1353
Houston Airport System      1216
Parks & Recreation          1152
Library                      563
Solid Waste Management       512
Name: dept, dtype: int64
```

If you do use a groupby, it doesn't matter what column you use, but you must use `size` and not `count` because `count` will not count missing values. It is possible to use the grouping column as the aggregating column.

[6]:
```
emp.groupby('dept').agg(num_employees=('dept', 'size'))
```

dept	num_employees
Fire	4376
Health & Human Services	1353
Houston Airport System	1216
Houston Public Works	4190
Library	563
Other	3373
Parks & Recreation	1152
Police	7573
Solid Waste Management	512

Exercise 5

Find the number of unique titles there are for each department.

[7]:
```
emp.groupby('dept').agg(unique_titles=('title', 'nunique'))
```

dept	unique_titles
Fire	77
Health & Human Services	161
Houston Airport System	137
Houston Public Works	215
Library	66
Other	358
Parks & Recreation	109
Police	145
Solid Waste Management	44

Exercise 6

Find the index of the employee with the maximum salary for each department and then use those index values to select their entire rows from the original DataFrame.

[8]:
```
df = emp.groupby('dept').agg(idx_sal=('salary', 'idxmax'))
df
```

dept	idx_sal
Fire	1732
Health & Human Services	8405
Houston Airport System	3897
Houston Public Works	10704
Library	7564
Other	13338
Parks & Recreation	11679
Police	4413
Solid Waste Management	20244

[9]:
```
idx = df['idx_sal']
emp.loc[idx]
```

	dept	title	hire_date	salary	sex	race
1732	Fire	PHYSICIAN,MD	2014-09-27	342784.0	Male	White
8405	Health & Human Services	CHIEF PHYSICIAN,MD	2017-07-31	186685.0	Female	White
3897	Houston Airport System	AVIATION DIRECTOR	2010-06-01	275000.0	Male	Hispanic
10704	Houston Public Works	PUBLIC WORKS DIRECTOR	2005-08-10	275000.0	Female	White
7564	Library	LIBRARY DIRECTOR	2005-11-07	170000.0	Female	Black
13338	Other	CITY ATTORNEY	2016-05-02	275000.0	Male	Black
11679	Parks & Recreation	PARKS & RECREATION DIRECTOR	2017-07-05	150000.0	Male	White
4413	Police	POLICE CHIEF	2016-11-30	280000.0	Male	Hispanic
20244	Solid Waste Management	SOLID WASTE DIRECTOR	2001-05-14	195000.0	Male	Black

Use the NYC deaths dataset for the remaining exercises

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

[10]:
```
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 7

What year had the most deaths?

[11]:
```
year_deaths = deaths.groupby('year').agg(total=('deaths', 'sum'))
year_deaths
```

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

[12]:
```
year_deaths.agg(['max', 'idxmax'])
```

	total
max	54138
idxmax	2008

Exercise 8

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

[13]:
```
deaths.groupby('race').agg(total=('deaths', 'sum')).sort_values('total',␣
↪ascending=False)
```

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

Exercise 9

Find the total number of deaths by cause and then select the five highest causes.

[14]:
```
deaths.groupby('cause').agg(total=('deaths', 'sum')).nlargest(5, 'total')
```

cause	total
Heart Disease	147551
Cancer	106367
Other	77999
Flu and Pneumonia	18678
Diabetes	13794

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.

[15]:
```
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-level index.

[16]:
```
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.

[17]:
```
emp.groupby(['dept','race', 'sex']).agg(min_salary=('salary', 'min'),
                                        max_salary=('salary', 'max')).head(10)
```

			min_salary	max_salary
dept	**race**	**sex**		
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.

[18]:
```
pd.set_option('display.max_columns', 100)
college = pd.read_csv('../data/college.csv')
college.head(3)
```

	instnm	city	stabbr	hbcu	menonly	womenonly	relaffil	satvrmid	satmtmid	distanceonly	ugds	ugds_white	ugds_black	ugds_hisp	ugds_asian	ugds_aian	ugds_nhpi	ugds_2mor	ugds_nra	ugds_unkn	pptug_ef	curroper	pctpell	pctfloan	ug25abv	md_earn_wne_p10	grad_debt_mdn_supp
0	Alabama A & M University	Normal	AL	1.0	0.0	0.0	0	424.0	420.0	0.0	4206.0	0.0333	0.9353	0.0055	0.0019	0.0024	0.0019	0.0000	0.0059	0.0138	0.0656	1	0.7356	0.8284	0.1049	30300	33888
1	University of Alabama at Birmingham	Birmingham	AL	0.0	0.0	0.0	0	570.0	565.0	0.0	11383.0	0.5922	0.2600	0.0283	0.0518	0.0022	0.0007	0.0368	0.0179	0.0100	0.2607	1	0.3460	0.5214	0.2422	39700	21941.5
2	Amridge University	Montgomery	AL	0.0	0.0	0.0	1	NaN	NaN	1.0	291.0	0.2990	0.4192	0.0069	0.0034	0.0000	0.0000	0.0000	0.0000	0.2715	0.4536	1	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?

[19]:
```
size = college.groupby('city').agg(size=('stabbr', 'size'))
size.head()
```

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

[20]:
```
size.sort_values('size', ascending=False).head()
```

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

Can call `idxmax` directly.

[21]:
```
college.groupby('city').agg(size=('stabbr', 'size')).idxmax()
```

[21]:
```
size    New York
dtype: object
```

Without groupby

Use `value_counts`

[22]:
```
college['city'].value_counts().head()
```

[22]:
```
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.

[23]:
```
filt = college['city'] == 'Houston'
college.loc[filt, 'stabbr'].unique()
```

[23]:
```
array(['TX', 'MO'], dtype=object)
```

Can see exact counts

[24]:
```
college.loc[filt, 'stabbr'].value_counts()
```

[24]:
```
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.

[25]:
```
city_unique_state = college.groupby('city').agg(num_unique_states=('stabbr',␣
↪'nunique'))
city_unique_state.head()
```

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

[26]:
```
city_unique_state.loc['Houston']
```

[26]:
```
num_unique_states    2
Name: Houston, dtype: int64
```

Also with `drop_duplicates`

[27]:
```
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?

[28]:
```
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

Find the largest college from each state. From those colleges, find the difference between the largest and smallest.

[29]:
```
largest_per_state = college.groupby('stabbr').agg(max_ugds=('ugds', 'max'))
largest_per_state.max() - largest_per_state.min()
```

[29]:
```
max_ugds    150956.0
dtype: float64
```

Exercise 7

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

Find the index of the maximum college per state first.

[30]:
```
ugds_idx = college.groupby('stabbr').agg(idx=('ugds', 'idxmax'))
ugds_idx.head()
```

stabbr	idx
AK	60
AL	5
AR	137
AS	4138
AZ	7116

[31]:
```
idx = ugds_idx['idx']
idx.head()
```

[31]:
```
stabbr
AK      60
AL       5
AR     137
AS    4138
AZ    7116
Name: idx, dtype: int64
```

Use the index to select the desired rows.

[32]:
```
college.loc[idx, ['stabbr', 'instnm', 'ugds']].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

Second method - set the index first to be instnm so that you can take advantage of idxmax

[33]:
```
c2 = college.set_index('instnm')
max_indexes = c2.groupby('stabbr').agg(max_ugds_college=('ugds', 'idxmax'),
                                       max_ugds=('ugds', 'max'))
max_indexes.head()
```

stabbr	max_ugds_college	max_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

Third method - Sort the data first, then sse the `first` groupby method to return the first row of each group after sorting.

[34]:
```
college.sort_values('ugds', ascending=False).groupby('stabbr') \
        .agg(max_ugds_college=('instnm', 'first'),
             max_ugds=('ugds', 'first')).head()
```

stabbr	max_ugds_college	max_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

Fourth method - Done previously without grouping

[35]:
```
college.sort_values(['stabbr', 'ugds'], ascending=[True, False]) \
       .drop_duplicates(subset='stabbr')[['stabbr', 'instnm', 'ugds']] \
       .head()
```

	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

Exercise 8

Do distance only schools tend to have more or less student population than non-distance-only schools?

[36]:
```
# They have more
college.groupby('distanceonly').agg(mean_ugds=('ugds', 'mean'))
```

distanceonly	mean_ugds
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?

[37]:
```
# Less
college.groupby('distanceonly').agg(mean_relaffil=('relaffil', 'mean'))
```

distanceonly	mean_relaffil
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?

[38]:
```
rel_oper_mean = college.query('relaffil == 1') \
                       .groupby('stabbr').agg(mean_curroper=('curroper', 'mean')) \
                       .round(2)
rel_oper_mean.head()
```

stabbr	mean_curroper
AK	1.00
AL	0.92
AR	0.94
AZ	0.44
CA	0.59

[39]:
```
rel_oper_mean.sort_values('mean_curroper').head()
```

stabbr	mean_curroper
UT	0.40
AZ	0.44
NV	0.50
CA	0.59
CT	0.65

Exercise 11

Find the top 5 historically black colleges that have the highest undergraduate white percentage (ugds_white)?

[40]:
```
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

[41]:
```
import pandas as pd
flights = pd.read_csv('../data/flights.csv', parse_dates=['date'])
flights.insert(1, 'day_of_week', flights['date'].dt.day_name())
flights.insert(2, 'month', flights['date'].dt.month_name())
flights.head(3)
```

	date	day_of_week	month	airline	origin	...	carrier_delay	weather_delay	nas_delay	security_delay	late_aircraft_delay
0	2018-01-01	Monday	January	UA	LAS	...	0	0	0	0	0
1	2018-01-01	Monday	January	WN	DEN	...	0	0	0	0	0
2	2018-01-01	Monday	January	B6	JFK	...	0	83	8	0	0

3 rows × 16 columns

[42]:
```
flights.shape
```

[42]:
```
(65923, 16)
```

Exercise 1

What is the average carrier delay for each day of the week for each airline? Highlight the worst day of the week for each airline.

[43]:
```
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	3.400000	4.700000	1.800000	3.300000	7.700000	3.000000	4.200000
AA	3.900000	3.700000	3.800000	3.800000	4.900000	4.100000	2.800000
AS	3.000000	1.500000	3.600000	3.700000	2.900000	2.600000	2.100000
B6	7.000000	4.000000	5.500000	5.100000	5.100000	6.200000	4.000000
DL	3.500000	2.600000	3.000000	3.600000	3.900000	3.300000	3.300000
EV	5.200000	8.500000	2.400000	7.200000	0.000000	6.000000	1.700000
F9	7.900000	3.600000	9.500000	4.100000	7.800000	6.600000	1.800000
MQ	2.100000	4.200000	0.000000	5.400000	4.200000	1.500000	2.300000
NK	2.200000	1.500000	5.800000	1.700000	2.200000	2.500000	1.200000
OH	6.300000	4.900000	2.500000	9.200000	19.900000	0.500000	1.500000
OO	6.500000	9.500000	2.000000	4.800000	3.500000	1.900000	10.200000
UA	3.200000	2.400000	3.200000	4.000000	3.700000	3.700000	2.600000
VX	2.600000	1.700000	10.100000	0.300000	2.800000	3.200000	5.200000
WN	5.000000	3.400000	2.200000	2.000000	4.800000	3.500000	3.200000
YV	5.500000	5.600000	5.600000	7.800000	8.600000	1.800000	4.000000
YX	2.200000	4.100000	2.500000	2.100000	2.900000	1.500000	1.400000

You can highlight min and max by chaining style methods.

[44]:
```
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	3.400000	4.700000	1.800000	3.300000	7.700000	3.000000	4.200000
AA	3.900000	3.700000	3.800000	3.800000	4.900000	4.100000	2.800000
AS	3.000000	1.500000	3.600000	3.700000	2.900000	2.600000	2.100000
B6	7.000000	4.000000	5.500000	5.100000	5.100000	6.200000	4.000000
DL	3.500000	2.600000	3.000000	3.600000	3.900000	3.300000	3.300000
EV	5.200000	8.500000	2.400000	7.200000	0.000000	6.000000	1.700000
F9	7.900000	3.600000	9.500000	4.100000	7.800000	6.600000	1.800000
MQ	2.100000	4.200000	0.000000	5.400000	4.200000	1.500000	2.300000
NK	2.200000	1.500000	5.800000	1.700000	2.200000	2.500000	1.200000
OH	6.300000	4.900000	2.500000	9.200000	19.900000	0.500000	1.500000
OO	6.500000	9.500000	2.000000	4.800000	3.500000	1.900000	10.200000
UA	3.200000	2.400000	3.200000	4.000000	3.700000	3.700000	2.600000
VX	2.600000	1.700000	10.100000	0.300000	2.800000	3.200000	5.200000
WN	5.000000	3.400000	2.200000	2.000000	4.800000	3.500000	3.200000
YV	5.500000	5.600000	5.600000	7.800000	8.600000	1.800000	4.000000
YX	2.200000	4.100000	2.500000	2.100000	2.900000	1.500000	1.400000

Exercise 2

Use a pivot table to find the total number of canceled flights for each origin airport and airline. Place the airlines 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.

[45]:
```
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

10 rows × 16 columns

[46]:
```
airline_cancel.agg(['max', 'idxmax'])
```

airline	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

2 rows × 16 columns

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.

[47]:
```
total_dist = flights.pivot_table(index='airline', columns='month',
                                 values='distance', aggfunc='sum')
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

Create a pivot table that shows the number of flights flown for every day of the week for every month.

[48]:
```
flights.pivot_table(index='month', columns='day_of_week', aggfunc='size')
```

day_of_week	Friday	Monday	Saturday	Sunday	Thursday	Tuesday	Wednesday
month							
April	786	943	644	898	818	755	754
August	1006	785	592	776	982	757	963
December	707	840	750	897	759	634	695
February	748	725	544	639	753	716	719
January	673	862	536	696	739	821	838
July	808	936	677	932	765	1012	780
June	1005	792	817	809	842	779	822
March	974	734	705	687	884	676	751
May	766	798	597	726	1058	913	894
November	887	710	624	737	871	707	709
October	773	961	554	711	790	939	894
September	739	742	742	902	761	719	762

Exercise 5

In exercise 4, the months and days of week are ordered alphabetically. It would be better if these values were ordered chronologically. Can you return a result that has both groups in the correct order. Use Monday as the first day of the week.

Convert to ordered categorical first overwriting the original columns.

[49]:
```
months = ['January', 'February', 'March', 'April', 'May', 'June', 'July',
          'August', 'September', 'October', 'November', 'December']
days =  ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday',
         'Sunday']
month_dtype = pd.CategoricalDtype(months, ordered=True)
day_dtype = pd.CategoricalDtype(days, ordered=True)
flights = flights.astype({'month': month_dtype, 'day_of_week': day_dtype})
```

Call the same pivot table and the index and columns will be automatically sorted by their category order.

[50]:
```
flights.pivot_table(index='month', columns='day_of_week', aggfunc='size')
```

day_of_week	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday
month							
January	862	821	838	739	673	536	696
February	725	716	719	753	748	544	639
March	734	676	751	884	974	705	687
April	943	755	754	818	786	644	898
May	798	913	894	1058	766	597	726
June	792	779	822	842	1005	817	809
July	936	1012	780	765	808	677	932
August	785	757	963	982	1006	592	776
September	742	719	762	761	739	742	902
October	961	939	894	790	773	554	711
November	710	707	709	871	887	624	737
December	840	634	695	759	707	750	897

Exercise 6

Create a new column in the flights dataset called `'dep_time_hour'` and set it equal to the hour (this will be an integer 0 through 23) of the flight. Find the average carrier delay for every month and dep_time_hour. Place the month in the columns.

[51]:
```
flights['dep_time_hour'] = flights['dep_time'] // 100
```

[52]:
```
flights.pivot_table(index='dep_time_hour', columns='day_of_week',
                    values='carrier_delay', aggfunc='mean').round(1)
```

day_of_week	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday
dep_time_hour							
0	0.2	3.6	1.2	2.4	6.5	1.7	2.4
1	0.0	0.0	1.1	0.7	0.0	1.7	2.3
2	NaN	NaN	0.0	NaN	NaN	0.0	0.0
4	NaN	NaN	NaN	NaN	0.0	NaN	NaN
5	5.9	1.5	1.8	2.4	4.4	3.0	1.8
6	2.5	2.8	2.6	2.8	5.2	4.0	3.5
7	2.0	3.5	1.8	3.7	3.8	4.1	1.8
8	3.6	3.1	7.2	4.3	2.3	4.0	3.7
9	3.1	2.8	2.5	3.8	3.3	3.4	3.0
10	2.8	2.9	2.1	3.4	3.3	3.2	2.1
11	3.8	3.4	2.0	2.0	2.5	1.6	3.3
12	3.1	4.0	3.1	4.1	3.0	4.2	4.3
13	2.3	3.6	3.1	2.8	4.7	2.6	2.9
14	3.0	4.1	2.3	5.0	1.9	3.6	3.1
15	3.1	3.5	3.0	4.9	3.8	3.7	4.7
16	3.0	3.3	2.5	4.6	3.8	3.8	3.2
17	4.3	3.7	4.9	4.4	3.4	3.2	5.0
18	4.1	4.3	2.9	9.4	2.8	5.3	4.1
19	2.9	2.6	2.9	4.7	6.8	3.3	6.3
20	5.3	4.3	4.2	4.8	4.8	4.1	4.3
21	2.6	6.1	3.4	7.6	6.7	3.4	4.5
22	4.2	7.6	4.5	4.4	6.1	2.5	5.6
23	5.1	2.4	0.9	4.2	10.9	10.3	2.6

Exercise 7

Use both `groupby` and `pivot_table` to compute the average and median distance flown by day of the week.

[53]:
```
flights.groupby('day_of_week').agg(median_dist=('distance', 'median'),
                                   mean_dist=('distance', 'mean')) \
       .style.format('{:,.0f}')
```

day_of_week	median_dist	mean_dist
Monday	912	1,071
Tuesday	888	1,052
Wednesday	868	1,053
Thursday	907	1,066
Friday	868	1,051
Saturday	937	1,107
Sunday	925	1,093

[54]:
```
flights.pivot_table(index='day_of_week', values='distance',
                    aggfunc=['median', 'mean']).style.format('{:,.0f}')
```

	median distance	mean distance
day_of_week		
Monday	912	1,071
Tuesday	888	1,052
Wednesday	868	1,053
Thursday	907	1,066
Friday	868	1,051
Saturday	937	1,107
Sunday	925	1,093

6.4 4. Counting with Crosstabs

[55]:
```
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)
```

	year	age	gender	country	family_history	treatment	work_interfere	no_employees	tech_company	benefits	care_options	wellness_program	seek_help	anonymity	leave	mental_health_consequence	phys_health_consequence	coworkers	supervisor
0	2014	37	Female	United States	No	Yes	Often	6-25	Yes	Yes	Not sure	No	Yes	Yes	Somewhat easy	No	No	Some of them	Yes
1	2014	44	Male	United States	No	No	Rarely	More than 1000	No	Don't know	No	Don't know	Don't know	Don't know	Don't know	Maybe	No	No	No
2	2014	32	Male	Canada	No	No	Rarely	6-25	Yes	No	No	No	No	Don't know	Somewhat difficult	No	No	Yes	Yes

Exercise 1

Do people with a family history of mental illness seek treatment more often than those who do not?

[56]:
```
pd.crosstab(index=mh['family_history'], columns=mh['treatment'])
```

treatment	No	Yes
family_history		
No	414	241
Yes	111	325

[57]:
```
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.

[58]:
```
pd.crosstab(index=mh['benefits'], columns=mh['treatment'])
```

treatment	No	Yes
benefits		
Don't know	225	134
No	142	150
Yes	158	282

[59]:
```
pd.crosstab(index=mh['benefits'], columns=mh['treatment'], normalize='index').round(2)
```

treatment	No	Yes
benefits		
Don't know	0.63	0.37
No	0.49	0.51
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.

[60]:
```
index = [mh['country'], mh['no_employees']]
columns = [mh['benefits'], mh['treatment']]
pd.crosstab(index=index, columns=columns)
```

	benefits	Don't know		No		Yes	
	treatment	No	Yes	No	Yes	No	Yes
country	no_employees						
Australia	1-5	1	0	1	1	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	1	3
	26-100	4	4	2	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	5
France	100-500	1	0	1	0	0	0
	26-100	0	0	3	0	0	0
	500-1000	1	0	0	0	1	0
	6-25	1	0	3	0	0	0
Germany	1-5	0	0	3	3	0	1
	100-500	1	2	1	0	0	0
	26-100	2	4	4	3	1	0
	500-1000	1	0	0	0	1	0
	6-25	5	1	3	2	0	2
	More than 1000	2	1	0	0	0	0
Ireland	1-5	1	0	5	4	0	0
	100-500	1	0	0	1	0	0
	26-100	0	1	2	3	0	0
	500-1000	0	0	0	0	0	1
	6-25	0	0	1	2	0	0
	More than 1000	2	0	1	0	1	1
Netherlands	1-5	2	2	1	2	0	1
	100-500	1	0	0	0	1	1
	26-100	1	0	2	1	0	0
	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	7	13	0	1
	100-500	5	6	2	3	2	1
	26-100	11	5	7	10	3	2
	500-1000	2	3	1	0	1	0
	6-25	11	7	23	13	1	1
	More than 1000	5	4	3	7	0	8
United States	1-5	11	4	16	30	3	8
	100-500	21	14	3	8	23	41
	26-100	41	26	7	6	18	61
	500-1000	8	1	1	1	9	20
	6-25	30	22	14	19	17	25
	More than 1000	35	15	5	2	65	87

[61]:
```
import pandas as pd
```

Exercise 4

Read in the bikes dataset and find the distribution of total trip duration by gender and events. Normalize over all groups. You should be able to answer the question, "From the total of all trip durations, what percent were done by males on a clear day?".

[62]:
```
bikes = pd.read_csv('../data/bikes.csv', parse_dates=['starttime', 'stoptime'])
bikes.head(3)
```

	gender	starttime	stoptime	tripduration	from_station_name	...	to_station_name	end_capacity	temperature	wind_speed	events
0	Male	2013-06-28 19:01:00	2013-06-28 19:17:00	993	Lake Shore Dr & Monroe St	...	Michigan Ave & Oak St	15.0	73.9	12.7	mostlycloudy
1	Male	2013-06-28 22:53:00	2013-06-28 23:03:00	623	Clinton St & Washington Blvd	...	Wells St & Walton St	19.0	69.1	6.9	partlycloudy
2	Male	2013-06-30 14:43:00	2013-06-30 15:01:00	1040	Sheffield Ave & Kingsbury St	...	Dearborn St & Monroe St	23.0	73.0	16.1	mostlycloudy

3 rows × 11 columns

[63]:
```
pd.crosstab(index=bikes['events'], columns=bikes['gender'],
            values=bikes['tripduration'], aggfunc='sum',
            normalize=True, margins=True).round(4) * 100
```

gender / events	Female	Male	All
clear	1.64	4.39	6.03
cloudy	6.09	17.12	23.21
fog	0.06	0.14	0.19
hazy	0.19	0.48	0.67
mostlycloudy	8.67	22.29	30.97
partlycloudy	10.39	23.95	34.34
rain	0.73	2.50	3.23
sleet	0.01	0.02	0.02
snow	0.11	0.66	0.77
tstorms	0.16	0.41	0.56
unknown	0.00	0.01	0.01
All	28.04	71.96	100.00

6.5 5. Alternate Groupby Syntax

Execute the cell below to read in the flights dataset and then use it for the following exercises.

[64]:
```
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

3 rows × 14 columns

Exercise 1

Use a dictionary in the `groupby` `agg` method to calculate the mean, median, min, and max of the air time for every airline.

[65]:
```
flights.groupby('airline').agg({'air_time': ['mean', 'median', 'min', 'max']})
```

	air_time			
	mean	median	min	max
airline				
9E	89.901705	84.0	24.0	224.0
AA	147.940078	126.0	22.0	421.0
AS	185.414344	149.0	38.0	395.0
B6	170.061845	132.0	32.0	428.0
DL	146.034775	125.0	22.0	405.0
EV	57.440252	45.0	35.0	178.0
F9	139.187223	124.0	55.0	327.0
MQ	80.218130	83.0	20.0	164.0
NK	147.173162	133.0	40.0	388.0
OH	66.418699	70.5	28.0	142.0
OO	98.228543	92.0	38.0	266.0
UA	160.476621	138.0	33.0	422.0
VX	178.189189	132.0	38.0	427.0
WN	112.839452	95.0	36.0	344.0
YV	121.506294	130.0	38.0	231.0
YX	93.165300	83.0	21.0	258.0

Exercise 2

Without using the `agg` method calculate the number of unique destinations for each airline.

[66]:
```
flights.groupby('airline')['dest'].nunique()
```

[66]:
```
airline
9E    13
AA    20
AS    18
B6    19
DL    20
EV     8
F9    17
MQ    12
NK    16
OH     9
OO    19
UA    19
VX    12
WN    15
YV    10
```

```
YX    14
Name: dest, dtype: int64
```

Exercise 3

Calculate the mean of every numeric column for each airline and origin without using the `agg` method.

[67]:
```
flights.groupby(['airline', 'origin']).mean().head()
```

airline	origin	dep_time	arr_time	cancelled	air_time	distance	carrier_delay	weather_delay	nas_delay	security_delay	late_aircraft_delay
9E	ATL	729.714286	840.142857	0.000000	105.142857	689.000000	0.000000	0.000000	3.857143	0.0	0.000000
	BOS	1320.029412	1453.970588	0.049020	47.103093	191.558824	2.284314	0.029412	8.607843	0.0	4.176471
	CLT	1254.613636	1461.238636	0.068182	82.926829	554.397727	5.261364	0.000000	4.022727	0.0	4.295455
	DCA	1168.218182	1309.563636	0.018182	44.629630	216.490909	10.945455	0.000000	1.309091	0.0	2.163636
	DFW	1346.068182	1682.136364	0.011364	137.080460	1054.022727	2.840909	0.738636	3.772727	0.0	11.875000

6.6 6. Custom Aggregation

Execute the cell below to read in the flights dataset and then use it for the following exercises.

[68]:
```
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

3 rows × 14 columns

Exercise 1

What are the three airlines with the least number of flights?

[69]:
```
flights['airline'].value_counts().tail(3)
```

[69]:
```
MQ    373
OH    257
EV    171
Name: airline, dtype: int64
```

Exercise 2

For each airline, find the 75th percentile of flight distance. Use a custom aggregation function.

[70]:
```
def per_75(s):
    return s.quantile(.75)
```

[71]:
```
flights.groupby('airline').agg(dist_75=('distance', per_75))
```

airline	dist_75
9E	852.0
AA	1558.0
AS	2402.0
B6	2381.0
DL	1587.0
EV	514.5
F9	1476.0
MQ	612.0
NK	1379.0
OH	500.0
OO	912.0
UA	1635.0
VX	2475.0
WN	1024.0
YV	1075.0
YX	912.0

Exercise 3

For each airline, find out what percentage of its flights leave on a Tuesday. Use a custom aggregation function.

[72]:
```
def tuesday_pct(s):
    return (s.dt.day_name() == 'Tuesday').mean()

flights.groupby('airline').agg(percent_tuesday=('date', tuesday_pct)).round(3) * 100
```

airline	percent_tuesday
9E	14.5
AA	14.6
AS	13.8
B6	13.5
DL	14.4
EV	15.8
F9	12.9
MQ	16.1
NK	12.8
OH	16.7
OO	14.3
UA	14.0
VX	13.3
WN	15.3
YV	13.9
YX	15.0

Exercise 4

Optimize exercise 2 without using a custom aggregation. What is the performance difference?

[73]:
```
flights['airline_cat'] = flights['airline'].astype('category')
flights['is_tuesday'] = flights['date'].dt.day_name() == 'Tuesday'
flights.groupby('airline_cat')['is_tuesday'].mean().round(3) * 100
```

[73]:
```
airline_cat
9E    14.5
AA    14.6
AS    13.8
B6    13.5
DL    14.4
EV    15.8
F9    12.9
MQ    16.1
NK    12.8
OH    16.7
OO    14.3
UA    14.0
VX    13.3
WN    15.3
YV    13.9
YX    15.0
Name: is_tuesday, dtype: float64
```

About 50% improvement

[74]:
```
%timeit -r 1 -n 5 flights.groupby('airline').agg(percent_tuesday=('date', tuesday_pct))
```

```
35.2 ms ± 0 ns per loop (mean ± std. dev. of 1 run, 5 loops each)
```

[75]:
```
%%timeit -r 1 -n 5
flights['is_tuesday'] = flights['date'].dt.day_name() == 'Tuesday'
flights.groupby('airline_cat')['is_tuesday'].mean().round(3) * 100
```

```
UsageError: Line magic function `%%timeit` not found.
```

Exercise 5

The range of salaries per department was calculated using the `min_max` custom function from the beginning of this chapter. Use this same function to calculate the range of distance for each airline. Then calculate this range again without a custom function.

[76]:
```
def min_max(s):
    return s.max() - s.min()
```

[77]:
```
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

[78]:
```
d_max = flights.groupby('airline')['distance'].max()
d_min = flights.groupby('airline')['distance'].min()
d_max - d_min
```

[78]:
```
airline
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
Name: distance, dtype: float64
```

Alternatively, create an entire DataFrame.

[79]:
```
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
```

airline	max_dist	min_dist	dist range
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 6

Which origin airport has the highest percentage of its flights cancelled?

[80]:
```
# no custom aggregation function needed
flights.groupby('origin').agg(pct_cancelled=('cancelled', 'mean')) \
       .nlargest(1, 'pct_cancelled').round(3) * 100
```

origin	pct_cancelled
BOS	3.4

Use the college dataset

Execute the following cell which reads in a few columns from the college dataset, sets the institution name as the index and converts 'stabbr' and 'relaffil' to categorical.

[81]:
```
cols = ['instnm', 'stabbr', 'relaffil', 'satvrmid', 'satmtmid', 'ugds']
college = pd.read_csv('../data/college.csv', usecols=cols,
                      index_col='instnm', dtype={'stabbr': 'category',
                                                 'relaffil': 'category'})
college.head(3)
```

instnm	stabbr	relaffil	satvrmid	satmtmid	ugds
Alabama A & M University	AL	0	424.0	420.0	4206.0
University of Alabama at Birmingham	AL	0	570.0	565.0	11383.0
Amridge University	AL	1	NaN	NaN	291.0

Exercise 7

How many states have more schools with a higher 'satvrmid' than 'satmtmid'? Make sure to not count schools that have missing values for either one.

Make a new DataFrame that drops rows when one of the sat columns is missing.

[82]:
```
col_has_sat = college.dropna(subset=['satvrmid', 'satmtmid']).copy()
col_has_sat.head(3)
```

instnm	stabbr	relaffil	satvrmid	satmtmid	ugds
Alabama A & M University	AL	0	424.0	420.0	4206.0
University of Alabama at Birmingham	AL	0	570.0	565.0	11383.0
University of Alabama in Huntsville	AL	0	595.0	590.0	5451.0

Only a fraction of the schools have both scores.

[83]:
```
len(col_has_sat)
```

[83]: 1184

[84]:
```
len(college)
```

[84]: 7535

Create a new boolean column that determines which score is higher.

[85]:
```
col_has_sat['higher_verbal'] = col_has_sat['satvrmid'] > col_has_sat['satmtmid']
col_has_sat.head(3)
```

instnm	stabbr	relaffil	satvrmid	satmtmid	ugds	higher_verbal
Alabama A & M University	AL	0	424.0	420.0	4206.0	True
University of Alabama at Birmingham	AL	0	570.0	565.0	11383.0	True
University of Alabama in Huntsville	AL	0	595.0	590.0	5451.0	True

Group by state to determine percentage with higher verbal. Only those greater than .5 have more verbal than math.

[86]:
```
avg_verbal_higher = col_has_sat.groupby('stabbr')['higher_verbal'].mean()
avg_verbal_higher.sort_values(ascending=False).head(10)
```

[86]:
```
stabbr
AK    1.000000
VI    1.000000
GA    0.690476
FL    0.684211
OR    0.588235
AL    0.571429
VA    0.564103
MN    0.560000
UT    0.500000
NH    0.500000
Name: higher_verbal, dtype: float64
```

Technically, have to check for ties. No custom function needed.

[87]:
```
(avg_verbal_higher > .5).sum()
```

[87]:
```
8
```

Exercise 8

Create a pivot table that shows the percentage of schools with less than 1,000 students in each state by religious affiliation. Also return the count of schools.

[88]:
```
def less_1k(s):
    return (s < 1_000).mean().round(3) * 100
```

[89]:
```
result = college.pivot_table(index='stabbr', columns='relaffil', values='ugds',
                             aggfunc=[less_1k, 'count'])
result.head(10)
```

	less_1k		count	
relaffil	0	1	0	1
stabbr				
AK	42.9	100.0	7	3
AL	44.4	37.5	71	18
AR	58.8	61.1	68	14
AS	0.0	NaN	1	0
AZ	63.7	77.8	118	8
CA	61.4	27.4	579	76
CO	64.4	28.6	113	4
CT	58.8	23.5	82	7
DC	52.9	0.0	14	4
DE	75.0	0.0	16	3

6.7 7. Transform and Filter with Groupby

Execute the cell below to reread the college dataset and use it for the exercises below.

[90]:
```
import pandas as pd
cols = ['instnm', 'stabbr', 'relaffil', 'satvrmid', 'satmtmid', 'ugds']
college = pd.read_csv('../data/college.csv', usecols=cols, index_col='instnm')
college.head(3)
```

instnm	stabbr	relaffil	satvrmid	satmtmid	ugds
Alabama A & M University	AL	0	424.0	420.0	4206.0
University of Alabama at Birmingham	AL	0	570.0	565.0	11383.0
Amridge University	AL	1	NaN	NaN	291.0

Exercise 1

Filter the college DataFrame for states that have more than 500,000 total undergraduate students. Can you verify your results?

[91]:
```
def filt_500k(sub_df):
    return sub_df['ugds'].sum() > 500_000

college_large = college.groupby('stabbr').filter(filt_500k)
college_large.head()
```

instnm	stabbr	relaffil	satvrmid	satmtmid	ugds
Prince Institute-Southeast	IL	0	NaN	NaN	84.0
Everest College-Phoenix	AZ	1	NaN	NaN	4102.0
Collins College	AZ	0	NaN	NaN	83.0
Empire Beauty School-Paradise Valley	AZ	1	NaN	NaN	25.0
Empire Beauty School-Tucson	AZ	0	NaN	NaN	126.0

[92]:
```
college_large.groupby('stabbr').agg(ugds_total=('ugds', 'sum')) \
             .sort_values('ugds_total', ascending=False).round(-3)
```

stabbr	ugds_total
CA	2304000.0
TX	1277000.0
NY	994000.0
FL	960000.0
PA	605000.0
IL	600000.0
OH	538000.0
AZ	520000.0

Exercise 2

Filter the college DataFrame for states that have a an average undergraduate student population greater than 2,500 and have more than 30 religiously affiliated schools. Can you verify your results?

[93]:
```
def func2(sub_df):
    return sub_df['ugds'].mean() > 2_500 and sub_df['relaffil'].sum() > 30
```

[94]:
```
c2 = college.groupby('stabbr').filter(func2)
c2.head(3)
```

instnm	stabbr	relaffil	satvrmid	satmtmid	ugds
Academy of Art University	CA	0	NaN	NaN	9885.0
ITT Technical Institute-Rancho Cordova	CA	0	NaN	NaN	500.0
Academy of Chinese Culture and Health Sciences	CA	0	NaN	NaN	NaN

[95]:
```
c2.groupby('stabbr').agg(mean_ugds=('ugds', 'mean'),
                         num_relaffil=('relaffil', 'sum'))
```

stabbr	mean_ugds	num_relaffil
CA	3518.308397	164
GA	2642.571429	37
IN	2653.559055	62
MI	2643.016043	48
TX	2998.530516	96
VA	2694.900000	44

Exercise 3

The maximum SAT score for each test is 800. Create a new column in the college dataset that shows each school's percentage of maximum for each SAT score.

No need to use transform here.

[96]:
```
college['pct_max_sat_verbal'] = (college['satvrmid'] / 800).round(3) * 100
college['pct_max_sat_math'] = (college['satmtmid'] / 800).round(3) * 100
college.head(3)
```

instnm	stabbr	relaffil	satvrmid	satmtmid	ugds	pct_max_sat_verbal	pct_max_sat_math
Alabama A & M University	AL	0	424.0	420.0	4206.0	53.0	52.5
University of Alabama at Birmingham	AL	0	570.0	565.0	11383.0	71.2	70.6
Amridge University	AL	1	NaN	NaN	291.0	NaN	NaN

Use the City of Houston dataset

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

[97]:
```
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 4

Filter it so that only position titles with an average salary of 100,000 remain. Can you verify your results?

[98]:
```
high_sal = emp.groupby('title').filter(lambda sub_df: sub_df['salary'].mean() >␣
↪100_000)
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

[99]:
```
high_sal.groupby('title').agg(avg_salary=('salary', 'mean')).min()
```

[99]:
```
avg_salary    100038.0
dtype: float64
```

Exercise 5

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?

[100]:
```
def sal_count(sub_df):
    return sub_df['salary'].mean() > 80000 and len(sub_df) >= 5
```

[101]:
```
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

[102]:
```
high_sal_count.groupby('title').agg(avg_salary=('salary', 'mean'),
                                    size=('salary', 'size')).min()
```

[102]:
```
avg_salary    80153.202222
size              5.000000
dtype: float64
```

Exercise 6

Add a column to the DataFrame that contains the median salary based on department, sex, and race.

[103]:
```
emp['median_drs'] = emp.groupby(['dept', 'sex', 'race'])['salary'].transform('median')
emp.head()
```

	dept	title	hire_date	salary	sex	race	median_drs
0	Police	POLICE SERGEANT	2001-12-03	87545.38	Male	White	73479.00
1	Other	ASSISTANT CITY ATTORNEY II	2010-11-15	82182.00	Male	Hispanic	47445.00
2	Houston Public Works	SENIOR SLUDGE PROCESSOR	2006-01-09	49275.00	Male	Black	38813.00
3	Police	SENIOR POLICE OFFICER	1997-05-27	75942.10	Male	Hispanic	68116.62
4	Police	SENIOR POLICE OFFICER	2006-01-23	69355.26	Male	White	73479.00

Exercise 7

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 sex. 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.

[104]:
```
def pct_max(sub_series):
    return sub_series / sub_series.max()
```

[105]:
```
emp['pct_max_dept_sex'] = emp.groupby(['dept', 'sex'])['salary'].transform(pct_max)
emp.head()
```

	dept	title	hire_date	salary	sex	race	median_drs	pct_max_dept_sex
0	Police	POLICE SERGEANT	2001-12-03	87545.38	Male	White	73479.00	0.312662
1	Other	ASSISTANT CITY ATTORNEY II	2010-11-15	82182.00	Male	Hispanic	47445.00	0.298844
2	Houston Public Works	SENIOR SLUDGE PROCESSOR	2006-01-09	49275.00	Male	Black	38813.00	0.227809
3	Police	SENIOR POLICE OFFICER	1997-05-27	75942.10	Male	Hispanic	68116.62	0.271222
4	Police	SENIOR POLICE OFFICER	2006-01-23	69355.26	Male	White	73479.00	0.247697

[106]:
```
filt = (emp['dept'] == 'Police') & (emp['sex'] == 'Male')
max_sal = emp.loc[filt, 'salary'].max()
max_sal
```

[106]:
```
280000.0
```

[107]:
```
emp.loc[0, 'salary'] / max_sal
```

[107]:
```
0.31266207142857144
```

6.8 8. Other Groupby Methods

Execute the next cell to read in some of the columns from the flights dataset and use it to answer the following exercises.

[108]:
```
import pandas as pd
cols = ['date', 'airline', 'origin', 'dest', 'dep_time', 'arr_time',
       'cancelled', 'air_time', 'distance', 'carrier_delay']
flights = pd.read_csv('../data/flights.csv', parse_dates=['date'], usecols=cols)
flights.head(3)
```

	date	airline	origin	dest	dep_time	arr_time	cancelled	air_time	distance	carrier_delay
0	2018-01-01	UA	LAS	IAH	100	547	0	134.0	1222.0	0
1	2018-01-01	WN	DEN	PHX	515	720	0	91.0	602.0	0
2	2018-01-01	B6	JFK	BOS	550	657	0	39.0	187.0	0

Exercise 1

For each airline, return the first and last row of each group. Use the `nth` groupby method.

[109]:
```
flights.groupby('airline').nth([0, -1]).head(8)
```

airline	date	origin	dest	dep_time	arr_time	cancelled	air_time	distance	carrier_delay
9E	2018-12-31	CLT	JFK	1400	1603	0	80.0	541.0	0
9E	2018-01-01	IAH	ATL	1346	1651	0	86.0	689.0	0
AA	2018-01-01	DFW	DCA	610	959	0	131.0	1192.0	0
AA	2018-12-31	DFW	SFO	2047	2245	0	194.0	1464.0	0
AS	2018-12-31	SEA	DFW	2315	502	0	210.0	1660.0	3
AS	2018-01-01	SEA	SFO	605	816	0	97.0	679.0	0
B6	2018-12-31	PHX	JFK	2234	509	0	233.0	2153.0	0
B6	2018-01-01	JFK	BOS	550	657	0	39.0	187.0	0

Exercise 2

For every origin and destination combination, select the 5000th flight.

Only the combinations that have at least 500 flights will have a returned value.

[110]:
```
flights.groupby(['origin', 'dest']).nth(499)
```

origin	dest	date	airline	dep_time	arr_time	cancelled	air_time	distance	carrier_delay
JFK	**LAX**	2018-11-27	DL	1925	2300	0	325.0	2475.0	0
LAS	**LAX**	2018-12-21	WN	545	655	0	48.0	236.0	0
LAX	**JFK**	2018-11-29	DL	1145	2007	0	269.0	2475.0	0
	LAS	2018-12-25	AA	1955	2107	0	54.0	236.0	0
	SFO	2018-10-15	WN	955	1115	0	56.0	337.0	0
LGA	**ORD**	2018-10-17	UA	1700	1836	0	129.0	733.0	0
ORD	**LGA**	2018-10-14	UA	1300	1615	0	95.0	733.0	0
SFO	**LAX**	2018-10-19	UA	1300	1435	0	58.0	337.0	0

Exercise 3

Find the date of the 10th cancelled flight for each airline.

[111]:
```
flights.query('cancelled == 1').groupby('airline').nth(9)
```

airline	date	origin	dest	dep_time	arr_time	cancelled	air_time	distance	carrier_delay
9E	2018-03-13	JFK	BOS	905	1030	1	NaN	187.0	0
AA	2018-01-04	EWR	PHX	1620	2009	1	NaN	2133.0	0
AS	2018-06-17	EWR	SFO	1725	2054	1	NaN	2565.0	0
B6	2018-01-05	BOS	DFW	731	1105	1	NaN	1562.0	0
DL	2018-01-04	DTW	PHL	1745	1941	1	NaN	453.0	0
EV	2018-10-28	DCA	EWR	600	715	1	NaN	199.0	0
F9	2018-09-15	MSP	DEN	840	951	1	NaN	680.0	0
MQ	2018-04-16	LGA	PHL	1815	1937	1	NaN	96.0	0
NK	2018-02-12	IAH	EWR	630	1042	1	NaN	1400.0	0
OH	2018-09-15	DCA	CLT	1355	1534	1	NaN	331.0	0
OO	2018-03-01	SFO	LAX	1650	1825	1	NaN	337.0	0
UA	2018-01-04	BOS	EWR	800	924	1	NaN	200.0	0
VX	2018-03-20	SFO	LAX	2140	2323	1	NaN	337.0	0
WN	2018-02-27	PHX	LAS	2115	2120	1	NaN	255.0	0
YV	2018-08-29	ATL	IAH	940	1047	1	NaN	689.0	0
YX	2018-01-05	JFK	BOS	1700	1822	1	NaN	187.0	0

Exercise 4

Find the average carrier delay for each origin and destination combination with more than 300 flights.

[112]:
```
delay = flights.groupby(['origin', 'dest'])['carrier_delay'].agg(['size', 'mean']).
↪round(1)
delay.head()
```

origin	dest	size	mean
ATL	**BOS**	304	2.5
	CLT	262	5.0
	DCA	287	2.2
	DEN	215	3.0
	DFW	289	2.3

[113]:
```
delay.query('size > 300').head(10)
```

		size	mean
origin	**dest**		
ATL	**BOS**	304	2.5
	LGA	411	1.5
	MCO	373	4.6
	ORD	319	2.5
BOS	**DCA**	383	3.4
	LGA	416	3.3
	ORD	314	1.8
DCA	**BOS**	348	1.7
	ORD	339	1.2
DEN	**LAX**	345	6.7

Exercise 5

Find the three shortest air times for every airline.

[114]:
```
flights.groupby('airline')['air_time'].nsmallest(3)
```

[114]:
```
airline
9E       32935    24.0
         45541    25.0
         2317     26.0
AA       8900     22.0
         24774    23.0
         43455    23.0
AS       54429    38.0
         55921    38.0
         24645    39.0
B6       9348     32.0
         16214    32.0
         41277    32.0
DL       13270    22.0
         10482    24.0
         22989    26.0
EV       16598    35.0
         17292    35.0
         20431    35.0
F9       10686    55.0
         59371    55.0
         53304    56.0
MQ       15949    20.0
         42097    21.0
         57860    21.0
NK       141      40.0
         2719     40.0
         36361    40.0
OH       2272     28.0
         20572    29.0
         6789     30.0
```

```
OO       31828    38.0
         58185    38.0
         63449    38.0
UA       60729    33.0
         23181    34.0
         28721    34.0
VX       2129     38.0
         3545     39.0
         3723     39.0
WN       23097    36.0
         55993    37.0
         5233     38.0
YV       38847    38.0
         42479    38.0
         43300    38.0
YX       25953    21.0
         50819    24.0
         47151    25.0
Name: air_time, dtype: float64
```

[115]:
```
g = flights.groupby('origin')['air_time'].agg('nlargest')
```

[116]:
```
g
```

[116]:
```
origin
ATL     10644    340.0
        7868     327.0
        15259    325.0
        15345    323.0
        4437     322.0
                 …
SFO     18084    386.0
        18103    348.0
        18079    345.0
        31160    345.0
        21411    344.0
Name: air_time, Length: 100, dtype: float64
```

[117]:
```
g.nsmallest(3)
```

[117]:
```
origin
DEN     28399    235.0
        38792    237.0
        35394    246.0
Name: air_time, dtype: float64
```

6.9 9. Binning Numeric Columns

[118]:
```
import pandas as pd
bikes = pd.read_csv('../data/bikes.csv')
bikes.head(3)
```

	gender	starttime	stoptime	tripduration	from_station_name	...	to_station_name	end_capacity	temperature	wind_speed	events
0	Male	2013-06-28 19:01:00	2013-06-28 19:17:00	993	Lake Shore Dr & Monroe St	...	Michigan Ave & Oak St	15.0	73.9	12.7	mostlycloudy
1	Male	2013-06-28 22:53:00	2013-06-28 23:03:00	623	Clinton St & Washington Blvd	...	Wells St & Walton St	19.0	69.1	6.9	partlycloudy
2	Male	2013-06-30 14:43:00	2013-06-30 15:01:00	1040	Sheffield Ave & Kingsbury St	...	Dearborn St & Monroe St	23.0	73.0	16.1	mostlycloudy

3 rows × 11 columns

Exercise 1

Find the number of rides between trip durations of 0 to 100, 101 to 1000, and 1001 and above.

[119]:
```
max_td = bikes['tripduration'].max()
pd.cut(bikes['tripduration'], [0, 100, 1000, max_td]).value_counts()
```

[119]:
```
(100, 1000]      39669
(1000, 86188]    10178
(0, 100]           242
Name: tripduration, dtype: int64
```

Exercise 2

Cut the trip duration into five equal-sized bins counting the occurence of each. Sort the resulting Series by the index.

[120]:
```
pd.qcut(bikes['tripduration'], 5).value_counts(sort=False)
```

[120]:
```
(59.999, 317.0]      10043
(317.0, 480.0]       10011
(480.0, 682.0]       10024
(682.0, 1007.0]       9997
(1007.0, 86188.0]    10014
Name: tripduration, dtype: int64
```

Exercise 3

Cut trip duration and temperature into five equal-sized bins and count the occurrences using `pd.crosstab`. Do you notice any patterns?

Rides with higher temperature have longer duration and vice versa.

[121]:
```
td_bins = pd.qcut(bikes['tripduration'], 5)
temp_bins = pd.qcut(bikes['temperature'], 5)
pd.crosstab(index=td_bins, columns=temp_bins)
```

temperature	(-9999.001, 48.0]	(48.0, 62.1]	(62.1, 71.1]	(71.1, 78.1]	(78.1, 96.1]
tripduration					
(59.999, 317.0]	2712	2204	1931	1670	1526
(317.0, 480.0]	2412	1947	2068	1927	1657
(480.0, 682.0]	2151	2103	2067	1917	1786
(682.0, 1007.0]	1832	1940	2180	2101	1944
(1007.0, 86188.0]	1458	1754	2328	2331	2143

Exercise 4

Create a pivot table containing the average trip duration by gender and temperature cut into 10 equal-sized bins.

[122]:
```
temp_bins = pd.qcut(bikes['temperature'], 10)
bikes.pivot_table(index=temp_bins, columns='gender',
                  values='tripduration', aggfunc='mean').round(0)
```

gender	Female	Male
temperature		
(-9999.001, 37.0]	797.0	587.0
(37.0, 48.0]	670.0	648.0
(48.0, 55.9]	762.0	622.0
(55.9, 62.1]	789.0	653.0
(62.1, 66.9]	791.0	724.0
(66.9, 71.1]	797.0	706.0
(71.1, 73.9]	844.0	746.0
(73.9, 78.1]	897.0	730.0
(78.1, 82.0]	823.0	725.0
(82.0, 96.1]	906.0	756.0

Exercise 5

The temperature column has a single obviously wrong value. Replace this value with the numpy nan object and then cut the resulting Series into five bins, labeling them 'cold', 'cool', 'mild', 'warm', 'hot'. Choose the boundaries of the bins that make sense for these labels. Then count the occurence of each label and include the missing values.

[123]:
```
# -9999 is wrong
bikes['temperature'].drop_duplicates().sort_values().head()
```

[123]:
```
27168   -9999.0
2064       -8.0
10262      -6.0
2062       -5.1
21774      -4.0
Name: temperature, dtype: float64
```

[124]:
```
import numpy as np
temp = bikes['temperature'].replace(-9999, np.nan)
tmin, tmax = temp.agg(['min', 'max'])
```

```
tmin, tmax
```

[124]:
```
(-8.0, 96.1)
```

[125]:
```
temp_bins = pd.cut(bikes['temperature'], [tmin, 40, 55, 65, 75, tmax],
                   labels=['cold', 'cool', 'mild', 'warm', 'hot'],
                   include_lowest=True)
temp_bins.head()
```

[125]:
```
0    warm
1    warm
2    warm
3    warm
4    warm
Name: temperature, dtype: category
Categories (5, object): ['cold' < 'cool' < 'mild' < 'warm' < 'hot']
```

[126]:
```
temp_bins.value_counts(dropna=False, sort=False)
```

[126]:
```
cold     6433
cool     8483
mild     8571
warm    13302
hot     13299
NaN         1
Name: temperature, dtype: int64
```

6.10 10. Miscellaneous Grouping Functionality

[127]:
```
flights = pd.read_csv('../data/flights.csv')
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

3 rows × 14 columns

Exercise 1

Create a Series of booleans determining if there is a carrier delay of 15 minutes or more. The values should be `False` if under 15 minutes and `True` if 15 minutes or over. Find the average distance flown by each group.

[128]:
```
has_carrier_delay = flights['carrier_delay'] >= 15
has_carrier_delay.head()
```

[128]:
```
0    False
1    False
2    False
3    False
4    False
```

```
Name: carrier_delay, dtype: bool
```

[129]:
```
dist = flights['distance']
dist.groupby(has_carrier_delay).mean()
```

[129]:
```
carrier_delay
False    1067.506608
True     1100.157233
Name: distance, dtype: float64
```

Exercise 2

Create a Series of booleans determining if there is a weather delay of 15 minutes or more. Compute a cross tabulation of this Series with the similar one created above on carrier delay.

[130]:
```
has_weather_delay = flights['weather_delay'] >= 15
```

[131]:
```
pd.crosstab(index=has_carrier_delay, columns=has_weather_delay)
```

weather_delay	False	True
carrier_delay		
False	61902	523
True	3478	20

Exercise 3

Find the total carrier delay by airline and origin as a Series with a multi-level index.

[132]:
```
s = flights.groupby(['airline', 'origin'])['carrier_delay'].sum()
s.head()
```

[132]:
```
airline  origin
9E       ATL        0
         BOS      233
         CLT      463
         DCA      602
         DFW      250
Name: carrier_delay, dtype: int64
```

Exercise 4

Using the Series from Exercise 3, calculate the total carrier delay by airline. Verify the result by calculating it directly from the original DataFrame.

[133]:
```
s.groupby('airline').sum()
```

[133]:
```
airline
9E     4338
AA    65027
AS     8890
B6    20094
DL    43580
```

```
EV      754
F9     6627
MQ     1098
NK     6821
OH     1486
OO    11788
UA    38674
VX     1500
WN    17353
YV     4107
YX     7627
Name: carrier_delay, dtype: int64
```

[134]:
```
# direct verification
flights.groupby('airline')['carrier_delay'].sum()
```

[134]:
```
airline
9E     4338
AA    65027
AS     8890
B6    20094
DL    43580
EV      754
F9     6627
MQ     1098
NK     6821
OH     1486
OO    11788
UA    38674
VX     1500
WN    17353
YV     4107
YX     7627
Name: carrier_delay, dtype: int64
```

Exercise 5

Read in the Sweden deaths dataset found in the covid folder. Place the year column in the index and then calculate the total number of deaths by 10 year age interval per year. Then take this DataFrame and calculate the average deaths per age group group by 5 year time spans

[135]:
```
df = pd.read_csv('../data/covid/sweden_deaths.csv', index_col='year')
df.columns = df.columns.astype('int64')
df.head()
```

year	0	1	2	3	4	...	96	97	98	99	100
1980	671	54	27	29	31	...	370	248	166	89	167
1981	653	40	21	22	18	...	393	281	183	97	144
1982	635	41	27	28	30	...	383	280	178	114	180
1983	646	33	26	23	19	...	439	277	207	133	202
1984	600	40	15	22	11	...	427	313	192	134	221

5 rows × 101 columns

[136]:
```
age_bins = pd.cut(df.columns.astype('int64'),
                  [0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 101],
                  right=False)
deaths_grouped = df.groupby(age_bins, axis=1).sum()
deaths_grouped.head()
```

year	[0, 10)	[10, 20)	[20, 30)	[30, 40)	[40, 50)	[50, 60)	[60, 70)	[70, 80)	[80, 90)	[90, 101)
1980	928	387	893	1489	2524	6516	15432	29001	27542	7085
1981	856	404	796	1398	2360	6142	15308	29200	28351	7219
1982	852	342	762	1460	2316	5917	15011	28707	28009	7294
1983	840	378	801	1319	2301	5592	14505	28521	28756	7777
1984	766	359	806	1361	2323	5381	14123	28665	28800	7898

[137]:
```
# this excludes 1980. first year is 1981
year_bins = pd.cut(deaths_grouped.index,
                   [1980, 1985, 1990, 1995, 2000, 2005, 2010, 2015, 2020])
deaths_grouped.groupby(year_bins).mean()
```

	[0, 10)	[10, 20)	[20, 30)	[30, 40)	[40, 50)	[50, 60)	[60, 70)	[70, 80)	[80, 90)	[90, 101)
(1980, 1985]	832.8	369.4	782.6	1378.8	2327.2	5677.4	14652.8	28872.0	28925.2	7783.4
(1985, 1990]	850.0	356.8	806.8	1243.2	2606.6	4903.0	13272.0	28054.0	32420.8	9608.6
(1990, 1995]	755.0	264.0	671.2	1076.8	2615.2	4745.0	11287.8	26259.4	35024.6	11844.2
(1995, 2000]	465.8	231.2	549.4	917.0	2119.8	5115.4	9670.6	24168.4	35999.4	14546.4
(2000, 2005]	415.8	234.6	530.2	798.4	1856.0	5238.6	9442.8	20645.0	36548.8	17082.6
(2005, 2010]	389.6	235.8	542.6	709.0	1693.4	4471.0	10413.2	18020.6	35769.4	18739.8
(2010, 2015]	368.8	184.8	610.6	706.0	1574.2	3915.8	10539.0	18172.2	32979.8	21381.0
(2015, 2020]	356.2	182.2	622.6	757.6	1415.2	3645.0	9419.6	21104.4	32077.4	22825.6

Part VII

Time Series

Chapter 7

Solutions

7.1 1. Datetime, Timedelta, and Period Objects

[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 1924 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(1_000_000, 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('2021-09-24 12:46:51.793940')
```

[8]:
```
td = dt2 - dt1
td
```

[8]:
```
Timedelta('19059 days 09:50:36.793940')
```

[9]:
```
td.days
```

[9]:
```
19059
```

Exercise 6

Create the Timedelta 84 hours and 17 minutes with both `pd.Timedelta` and `pd.to_timedelta` and verify that they are equal.

[10]:
```
pd.Timedelta('84:17:00') == pd.to_timedelta('84:17:00')
```

[10]:
```
True
```

Exercise 7

Which is larger - 5,206 days or 123,000 hours?

[11]:
```
td1 = pd.to_timedelta(5_206, unit='d')
td2 = pd.to_timedelta(123_000, unit='h')
```

[12]:
```
td1
```

[12]:
```
Timedelta('5206 days 00:00:00')
```

[13]:
```
td2
```

[13]:
```
Timedelta('5125 days 00:00:00')
```

[14]:
```
td1 > td2
```

[14]:
```
True
```

Exercise 8

Take a look at the `pd.Timestamp` docstring. Each component (year, month, day, etc...) is available as a parameter in the constructor. Use the parameters to create a time stamp that has a non-zero value for each component.

[15]:
```
pd.Timestamp(year=2018, month=12, day=5, hour=17, minute=55,
             second=3, microsecond=1534)
```

[15]:
```
Timestamp('2018-12-05 17:55:03.001534')
```

Exercise 9

Convert the given string to a datetime.

[16]:
```
s = 'month=10 year=2021 day=19 hour=6 minute=23'
```

[17]:
```
pd.to_datetime(s, format='month=%m year=%Y day=%d hour=%H minute=%M')
```

[17]:
```
Timestamp('2021-10-19 06:23:00')
```

Exercise 10

How many seconds elapsed from Feb 23, 2018 at 5:45 pm until Dec 14, 2020 at 7:32 am

[18]:
```
ts1 = pd.Timestamp('Feb 23, 2018 5:45 pm')
ts2 = pd.Timestamp('Dec 14, 2020 7:32 pm')
td = ts2 - ts1
td
```

[18]:
```
Timedelta('1025 days 01:47:00')
```

[19]:
```
td.total_seconds()
```

[19]:
```
88566420.0
```

[20]:
```
ts1.weekofyear
```

[20]:
```
8
```

Exercise 11

What day of the year is October 11 on a leap year?

[21]:
```
ts = pd.Timestamp('October 11, 2000')
ts.is_leap_year
```

[21]:
```
True
```

[22]:
```
ts.dayofyear
```

[22]:
```
285
```

Exercise 12

What was the date and time 198 hours and 33 minutes past December 3, 2020 at 5:15 pm

[23]:
```
ts = pd.Timestamp('December 3, 2020 5:15 pm')
td = pd.Timedelta('198:33:00')
```

[24]:
```
ts + td
```

[24]:
```
Timestamp('2020-12-11 23:48:00')
```

Exercise 13

It takes painter A 3 days 14 hours and 38 minutes to paint a house. Painter B takes 9 hours and 56 minutes to paint the same house. How many houses of the same size can painter B paint in the time it takes painter A to paint one.

[25]:
```
td1 = pd.Timedelta('3 days 14:38:00')
td2 = pd.Timedelta('9:56:00')
```

[26]:
```
int(td1 / td2)
```

[26]:
```
8
```

Exercise 14

The following string represents June 3rd, 2020. Convert it to the correct datetime.

[27]:
```
s = '3/6/2020'
```

[28]:
```
pd.to_datetime(s, dayfirst=True)
```

[28]:
```
Timestamp('2020-06-03 00:00:00')
```

Exercise 15

Create a Period object for the entire minute of 2:32 pm on October 11, 2020.

[29]:
```
pd.Period('2020-10-11 2:32 pm')
```

[29]:
```
Period('2020-10-11 14:32', 'T')
```

Exercise 16

The City of Houston employee data was retrieved on June 1, 2019. Can you calculate the exact amount of years of experience and assign as a new column named `experience`?

[30]:
```
emp = pd.read_csv('../data/employee.csv', parse_dates=['hire_date'])
```

One year is approximately 365.25 days.

[31]:
```
pull_date = pd.to_datetime('2019-6-1')
one_year = pd.to_timedelta(365.25, unit='D')
```

[32]:
```
pull_date
```

[32]:
```
Timestamp('2019-06-01 00:00:00')
```

[33]:
```
one_year
```

[33]:
```
Timedelta('365 days 06:00:00')
```

[34]:
```
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	17.492129
1	Other	ASSISTANT CITY ATTORNEY II	2010-11-15	82182.00	Male	Hispanic	8.542094
2	Houston Public Works	SENIOR SLUDGE PROCESSOR	2006-01-09	49275.00	Male	Black	13.390828
3	Police	SENIOR POLICE OFFICER	1997-05-27	75942.10	Male	Hispanic	22.012320
4	Police	SENIOR POLICE OFFICER	2006-01-23	69355.26	Male	White	13.352498

7.2 2. Selecting Time Series Data

Exercise 1

Read in the weather time series dataset and place the date column in the index.

[35]:
```
weather = pd.read_csv('../data/weather.csv', parse_dates=['date'], index_col='date')
weather.head()
```

date	rain	snow	temperature
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

[36]:
```
weather.tail()
```

date	rain	snow	temperature
2018-11-20	No	No	64.0
2018-11-21	No	No	57.0
2018-11-22	No	No	50.0
2018-11-23	No	No	45.0
2018-11-24	No	No	45.0

Exercise 2

Select all of the month of November, 2010

[37]:
```
weather.loc['2010-11']
```

date	rain	snow	temperature
2010-11-01	No	No	63.0
2010-11-02	No	No	57.9
2010-11-03	Yes	No	55.9
2010-11-04	Yes	No	54.0
2010-11-05	Yes	No	61.0
2010-11-06	Yes	No	53.1
2010-11-07	No	No	53.1
2010-11-08	No	No	66.9
2010-11-09	No	No	73.0
2010-11-10	No	No	70.0
2010-11-11	No	No	66.0
2010-11-12	No	No	66.0
2010-11-13	No	No	69.1
2010-11-14	No	No	70.0
2010-11-15	No	No	70.0
2010-11-16	Yes	No	66.9
2010-11-17	Yes	No	64.9
2010-11-18	No	No	62.1
2010-11-19	No	No	61.0
2010-11-20	No	No	69.1
2010-11-21	No	No	69.1
2010-11-22	No	No	73.0
2010-11-23	Yes	No	77.0
2010-11-24	No	No	63.0
2010-11-25	Yes	No	61.0
2010-11-26	Yes	No	66.0
2010-11-27	No	No	54.0
2010-11-28	No	No	52.0
2010-11-29	Yes	No	55.0
2010-11-30	Yes	No	72.0

Exercise 3

Select all of the second quarter of 2017.

[38]:
```
weather.loc['2017Q2']
```

date	rain	snow	temperature
2017-04-01	No	No	75.0
2017-04-02	No	No	69.1
2017-04-03	No	No	75.9
2017-04-04	No	No	82.0
2017-04-05	No	No	78.1
...	...	...	...
2017-06-26	No	No	84.9
2017-06-27	No	No	82.9
2017-06-28	No	No	82.9
2017-06-29	No	No	87.1
2017-06-30	No	No	87.1

91 rows × 3 columns

Exercise 4

Select data from July 1, 2015 to the end of 2016.

[39]:
```
weather.loc['2015Q3':'2016'] # alternatively weather.loc['2015-7':'2016']
```

date	rain	snow	temperature
2015-07-01	No	No	87.1
2015-07-02	No	No	87.1
2015-07-03	No	No	78.1
2015-07-04	No	No	87.1
2015-07-05	No	No	90.0
...	...	...	...
2016-12-27	No	No	68.0
2016-12-28	No	No	60.1
2016-12-29	No	No	63.0
2016-12-30	No	No	48.0
2016-12-31	No	No	50.0

550 rows × 3 columns

Exercise 5

Select just the rain and snow columns from the January 1, 2008 to January 7, 2008.

[40]:
```
weather.loc['2008-1-1':'2008-1-7', ['rain', 'snow']]
```

date	rain	snow
2008-01-01	No	No
2008-01-02	No	Yes
2008-01-03	No	No
2008-01-04	No	No
2008-01-05	No	No
2008-01-06	No	No
2008-01-07	No	No

Exercise 6

What was the temperature on June 11, 2011?

[41]:
```
weather.loc['2011-6-11', 'temperature']
```

[41]:
```
93.9
```

Exercise 7

How many days did it rain during the last three months of 2011?

[42]:
```
weather.loc['2011Q4', 'rain'].value_counts()
```

[42]:
```
No     69
Yes    23
Name: rain, dtype: int64
```

[43]:
```
# to get exact number
(weather.loc['2011Q4', 'rain'] == 'Yes').sum()
```

[43]:
```
23
```

Exercise 8

Which year had more snow days, 2007 or 2012?

[44]:
```
(weather.loc['2007', 'snow'] == 'Yes').sum()
```

[44]:
```
5
```

[45]:
```
(weather.loc['2012', 'snow'] == 'Yes').sum()
```

[45]:
```
2
```

Exercise 9

Select every other Thursday.

[46]:
```
weather.asfreq('2W-THU').head()
```

date	rain	snow	temperature
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 10

Select the first day of each month.

[47]:
```
weather.asfreq('MS').head()
```

date	rain	snow	temperature
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

Exercise 11

Select every other October 1st.

[48]:
```
weather.asfreq('2AS-OCT')
```

date	rain	snow	temperature
2007-10-01	No	No	82.0
2009-10-01	No	No	73.0
2011-10-01	No	No	64.0
2013-10-01	No	No	82.0
2015-10-01	No	No	66.0
2017-10-01	No	No	71.1

Use the temperature dataset for the remaining exercises

Execute the following cell to read in the temperature dataset which sets the datetime column in the index.

[49]:
```
df_temp = pd.read_csv('../data/weather/temperature.csv', parse_dates=['datetime'],
                      index_col='datetime')
df_temp.head()
```

datetime	Seattle	San Francisco	Los Angeles	Las Vegas	Denver	Houston	Chicago	Atlanta	Miami	New York
2013-01-01 00:00:00	2.94	11.50	11.66	7.28	-2.30	8.81	-0.19	1.92	19.41	-1.12
2013-01-01 01:00:00	2.40	10.22	10.67	5.95	-3.23	8.81	0.28	0.60	19.35	-1.69
2013-01-01 02:00:00	1.70	8.02	9.91	5.18	-3.03	8.81	0.33	-0.53	18.99	-1.96
2013-01-01 03:00:00	1.45	7.30	9.33	4.42	-3.67	8.48	0.12	-1.36	18.56	-2.08
2013-01-01 04:00:00	0.95	6.84	8.82	3.62	-5.55	8.34	0.04	-1.44	18.49	-2.32

Exercise 12

Select the temperatures for Houston between 3 and 6 p.m. on July 4, 2014.

[50]:
```
df_temp.loc['2014-7-4 15:00':'2014-7-4 18:00', 'Houston']
```

[50]:
```
datetime
2014-07-04 15:00:00    27.37
2014-07-04 16:00:00    28.85
2014-07-04 17:00:00    30.29
2014-07-04 18:00:00    31.00
Name: Houston, dtype: float64
```

Exercise 13

Upsample the result from the previous exercise so that there are entries every 20 minutes.

[51]:
```
s = df_temp.loc['2014-7-4 15:00':'2014-7-4 18:00', 'Houston'].asfreq('20min')
s
```

[51]:
```
datetime
2014-07-04 15:00:00    27.37
2014-07-04 15:20:00      NaN
2014-07-04 15:40:00      NaN
2014-07-04 16:00:00    28.85
2014-07-04 16:20:00      NaN
2014-07-04 16:40:00      NaN
2014-07-04 17:00:00    30.29
2014-07-04 17:20:00      NaN
2014-07-04 17:40:00      NaN
2014-07-04 18:00:00    31.00
Freq: 20T, Name: Houston, dtype: float64
```

Exercise 14

Linearly interpolate the missing values in the previous exercise to estimate the temperature at 4:40 pm on July 4, 2014.

[52]:
```
s1 = s.asfreq('20min').interpolate()
s1
```

[52]:
```
datetime
2014-07-04 15:00:00    27.370000
2014-07-04 15:20:00    27.863333
```

```
2014-07-04 15:40:00    28.356667
2014-07-04 16:00:00    28.850000
2014-07-04 16:20:00    29.330000
2014-07-04 16:40:00    29.810000
2014-07-04 17:00:00    30.290000
2014-07-04 17:20:00    30.526667
2014-07-04 17:40:00    30.763333
2014-07-04 18:00:00    31.000000
Freq: 20T, Name: Houston, dtype: float64
```

[53]:
```
s1.loc['2014-7-4 4:40 pm']
```

[53]:
```
29.81000000000002
```

7.3 3. Grouping by Time

[54]:
```
import pandas as pd
msft = pd.read_csv('../data/stocks/msft20.csv', parse_dates=['date'], index_col='date')
msft.head(3)
```

date	open	high	low	close	adjusted_close	volume	dividend_amount
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?

[55]:
```
volume = msft.resample('W').agg({'volume':'sum'})
volume.head()
```

date	volume
1999-10-24	262488000
1999-10-31	287399400
1999-11-07	268534000
1999-11-14	541663200
1999-11-21	295783800

[56]:
```
volume.agg(['max', 'idxmax'])
```

	volume
max	879723200
idxmax	2006-05-07 00:00:00

Exercise 2

With help from the `diff` method, find the quarter containing the most number of "up" days. An up day is when the adjusted close of the current day is greater than the previous day.

Use diff to find the difference between the current row and the one directly above it.

[57]:
```
msft['adjusted_close'].diff().head()
```

[57]:
```
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
```

[58]:
```
up_days = msft['adjusted_close'].diff() > 0
up_days.head()
```

[58]:
```
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
```

[59]:
```
up_days.resample('Q').sum().head()
```

[59]:
```
date
1999-12-31    24
2000-03-31    32
2000-06-30    30
2000-09-30    23
2000-12-31    29
Freq: Q-DEC, Name: adjusted_close, dtype: int64
```

[60]:
```
up_days.resample('Q').sum().agg(['max', 'idxmax'])
```

[60]:
```
max                       43
idxmax    2001-12-31 00:00:00
Name: adjusted_close, dtype: object
```

Exercise 3

Find the mean price per year along with the minimum and maximum volume.

[61]:
```
msft.resample('Y').agg({'adjusted_close': 'mean',
                        'volume':['min', 'max']}).head()
```

	adjusted_close	volume	
	mean	min	max
date			
1999-12-31	31.267802	12517600	243819200
2000-12-31	24.601943	15734800	313645800
2001-12-31	20.186950	11701600	209348800
2002-12-31	17.606981	18386000	202307800
2003-12-31	16.917920	12076900	210558300

Exercise 4

Find the mean of each column for every 6 month time period. The first time period should start on the month in the first row.

[62]:
```
msft.resample('6MS').mean().head()
```

	open	high	low	close	adjusted_close	volume	dividend_amount
date							
1999-10-01	99.483626	101.524673	97.742539	99.664739	32.169063	6.881454e+07	0.0
2000-04-01	71.997584	73.081694	70.671567	71.754222	23.160267	7.102180e+07	0.0
2000-10-01	58.145086	59.643325	56.805082	58.132640	18.763582	9.557858e+07	0.0
2001-04-01	65.129725	66.240963	64.089618	65.175279	21.036842	7.723125e+07	0.0
2001-10-01	62.886563	63.912634	62.026427	62.991855	20.332069	6.147958e+07	0.0

Exercise 5

Repeat exercise 4 using a time span of 3 years where the year begins July 1.

[63]:
```
msft.resample('3AS-Jul').mean().head()
```

	open	high	low	close	adjusted_close	volume	dividend_amount
date							
1999-07-01	69.487248	70.805493	68.263427	69.497398	22.431853	7.474444e+07	0.000000
2002-07-01	31.591090	31.945893	31.225483	31.568721	17.448777	6.987057e+07	0.004703
2005-07-01	28.210885	28.479904	27.970386	28.214272	21.107137	6.750704e+07	0.001538
2008-07-01	24.989465	25.292230	24.681177	24.990675	19.746500	6.677171e+07	0.002153
2011-07-01	31.582066	31.855132	31.306446	31.599248	27.001569	4.788299e+07	0.003612

Exercise 6

Repeat exercise five using the `groupby` method instead of `resample`.

[64]:
```
msft.groupby(pd.Grouper(freq='3AS-Jul')).mean().head()
```

date	open	high	low	close	adjusted_close	volume	dividend_amount
1999-07-01	69.487248	70.805493	68.263427	69.497398	22.431853	7.474444e+07	0.000000
2002-07-01	31.591090	31.945893	31.225483	31.568721	17.448777	6.987057e+07	0.004703
2005-07-01	28.210885	28.479904	27.970386	28.214272	21.107137	6.750704e+07	0.001538
2008-07-01	24.989465	25.292230	24.681177	24.990675	19.746500	6.677171e+07	0.002153
2011-07-01	31.582066	31.855132	31.306446	31.599248	27.001569	4.788299e+07	0.003612

Use the temperature dataset for the remaining exercises

Execute the following cell to read in the temperature dataset which sets the datetime column in the index.

[65]:
```
temp = pd.read_csv('../data/weather/temperature.csv',
                   parse_dates=['datetime'], index_col='datetime')
temp.head()
```

datetime	Seattle	San Francisco	Los Angeles	Las Vegas	Denver	Houston	Chicago	Atlanta	Miami	New York
2013-01-01 00:00:00	2.94	11.50	11.66	7.28	-2.30	8.81	-0.19	1.92	19.41	-1.12
2013-01-01 01:00:00	2.40	10.22	10.67	5.95	-3.23	8.81	0.28	0.60	19.35	-1.69
2013-01-01 02:00:00	1.70	8.02	9.91	5.18	-3.03	8.81	0.33	-0.53	18.99	-1.96
2013-01-01 03:00:00	1.45	7.30	9.33	4.42	-3.67	8.48	0.12	-1.36	18.56	-2.08
2013-01-01 04:00:00	0.95	6.84	8.82	3.62	-5.55	8.34	0.04	-1.44	18.49	-2.32

Exercise 7

Find the mean temperature of every city for every 8 hour time period.

[66]:
```
temp.resample('8H').mean().head(6)
```

datetime	Seattle	San Francisco	Los Angeles	Las Vegas	Denver	Houston	Chicago	Atlanta	Miami	New York
2013-01-01 00:00:00	1.33750	7.69875	9.22625	4.89125	-4.74625	8.71875	0.04875	-0.78000	19.00625	-2.28000
2013-01-01 08:00:00	-0.05625	3.97750	5.62500	2.37000	-8.02500	11.24375	1.05500	-0.16750	20.16625	-1.54250
2013-01-01 16:00:00	2.03750	8.42000	11.72125	6.13875	-5.85750	15.62875	-0.13125	10.00375	22.66750	2.59250
2013-01-02 00:00:00	1.45500	7.60250	9.49000	2.94625	-9.14375	18.82625	-5.02750	8.60375	21.32125	3.07500
2013-01-02 08:00:00	-0.62125	5.04500	5.92875	-1.06500	-11.80625	14.46000	-8.68375	8.61375	21.77625	3.61875
2013-01-02 16:00:00	2.26125	8.76750	12.42500	4.89875	-3.93250	11.65875	-5.09125	10.04000	24.38625	2.91250

Exercise 8

Verify that there are 24 rows for each day.

[67]:
```
temp.resample('D').size().head()
```

[67]:
```
datetime
2013-01-01    24
2013-01-02    24
2013-01-03    24
2013-01-04    24
2013-01-05    24
```

```
Freq: D, dtype: int64
```

Get the unique values.

[68]:
```
temp.resample('D').size().unique()
```

[68]:
```
array([24])
```

Exercise 9

For each month, return the maximum temperature amongst all cities.

[69]:
```
# get max temperature for each city
max_temp = temp.resample('M').max()
max_temp.head()
```

datetime	Seattle	San Francisco	Los Angeles	Las Vegas	Denver	Houston	Chicago	Atlanta	Miami	New York
2013-01-31	10.65	16.69	25.54	17.47	17.00	25.57	14.00	22.86	27.36	13.12
2013-02-28	11.55	19.94	26.29	20.66	16.26	25.97	10.95	18.21	28.71	12.63
2013-03-31	18.31	23.22	27.61	29.74	22.49	31.64	14.62	24.51	31.07	14.18
2013-04-30	21.43	30.11	27.94	36.10	25.17	28.07	22.79	28.58	30.42	27.91
2013-05-31	28.72	30.90	32.58	38.18	29.54	32.21	31.67	30.13	32.07	32.84

[70]:
```
# now get maximum amongst all cities
max_temp.max(axis=1).head()
```

[70]:
```
datetime
2013-01-31    27.36
2013-02-28    28.71
2013-03-31    31.64
2013-04-30    36.10
2013-05-31    38.18
Freq: M, dtype: float64
```

Exercise 10

For each month, return the maximum temperature amongst all cities along with the city name where the maximum occurred. Return a two-column DataFrame, where the first column is the maximum temperature, and the second is the city. The index should be the month.

[71]:
```
# use max_temp from exercise 9
max_temp.agg(['max', 'idxmax'], axis=1).head(10)
```

datetime	max	idxmax
2013-01-31	27.36	Miami
2013-02-28	28.71	Miami
2013-03-31	31.64	Houston
2013-04-30	36.1	Las Vegas
2013-05-31	38.18	Las Vegas
2013-06-30	45.49	Las Vegas
2013-07-31	45.26	Las Vegas
2013-08-31	42.54	Las Vegas
2013-09-30	38.8	Los Angeles
2013-10-31	40.6	Denver

7.4 4. Rolling Windows

[72]:
```
temp = pd.read_csv('../data/weather/temperature.csv',
                   parse_dates=['datetime'], index_col='datetime')
temp.head()
```

datetime	Seattle	San Francisco	Los Angeles	Las Vegas	Denver	Houston	Chicago	Atlanta	Miami	New York
2013-01-01 00:00:00	2.94	11.50	11.66	7.28	-2.30	8.81	-0.19	1.92	19.41	-1.12
2013-01-01 01:00:00	2.40	10.22	10.67	5.95	-3.23	8.81	0.28	0.60	19.35	-1.69
2013-01-01 02:00:00	1.70	8.02	9.91	5.18	-3.03	8.81	0.33	-0.53	18.99	-1.96
2013-01-01 03:00:00	1.45	7.30	9.33	4.42	-3.67	8.48	0.12	-1.36	18.56	-2.08
2013-01-01 04:00:00	0.95	6.84	8.82	3.62	-5.55	8.34	0.04	-1.44	18.49	-2.32

Exercise 1

Calculate a 6-hour moving average of temperature. Set the minimum number of rows used in the group to 1.

[73]:
```
temp.rolling('6H', min_periods=1).mean().head()
```

datetime	Seattle	San Francisco	Los Angeles	Las Vegas	Denver	Houston	Chicago	Atlanta	Miami	New York
2013-01-01 00:00:00	2.940000	11.500000	11.660000	7.280000	-2.300000	8.8100	-0.190	1.920000	19.4100	-1.1200
2013-01-01 01:00:00	2.670000	10.860000	11.165000	6.615000	-2.765000	8.8100	0.045	1.260000	19.3800	-1.4050
2013-01-01 02:00:00	2.346667	9.913333	10.746667	6.136667	-2.853333	8.8100	0.140	0.663333	19.2500	-1.5900
2013-01-01 03:00:00	2.122500	9.260000	10.392500	5.707500	-3.057500	8.7275	0.135	0.157500	19.0775	-1.7125
2013-01-01 04:00:00	1.888000	8.776000	10.078000	5.290000	-3.556000	8.6500	0.116	-0.162000	18.9600	-1.8340

Exercise 2

How many observations are there in each 30-day rolling window of time? Use the `count` method because the `size` method is not available.

There are 720 (`24 * 30`) observations per period as there are observations each hour.

[74]:
```
temp.rolling('30D', min_periods=1).count().tail()
```

datetime	Seattle	San Francisco	Los Angeles	Las Vegas	Denver	Houston	Chicago	Atlanta	Miami	New York
2016-12-31 19:00:00	720.0	720.0	720.0	720.0	720.0	720.0	720.0	720.0	720.0	720.0
2016-12-31 20:00:00	720.0	720.0	720.0	720.0	720.0	720.0	720.0	720.0	720.0	720.0
2016-12-31 21:00:00	720.0	720.0	720.0	720.0	720.0	720.0	720.0	720.0	720.0	720.0
2016-12-31 22:00:00	720.0	720.0	720.0	720.0	720.0	720.0	720.0	720.0	720.0	720.0
2016-12-31 23:00:00	720.0	720.0	720.0	720.0	720.0	720.0	720.0	720.0	720.0	720.0

Exercise 3

Calculate the 30-day moving average for Los Angeles and Houston using a 1-row minimum. What percentage of the rows does Houston have a higher temperature?

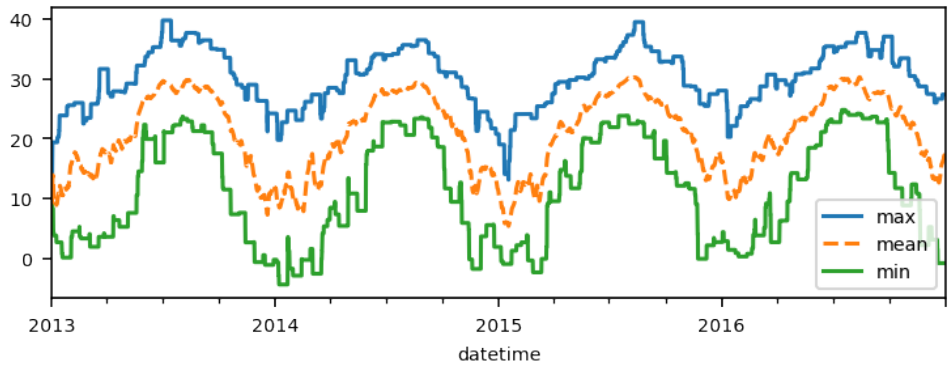
[75]:
```
houston = temp['Houston'].rolling('30D', min_periods=1).mean()
los_angeles = temp['Los Angeles'].rolling('30D', min_periods=1).mean()
(houston > los_angeles).mean()
```

[75]:
```
0.8586584531143052
```

Exercise 4

Calculate the minimum, maximum, and mean temperatures for Houston using a rolling 14-day period and plot the results.

[76]:
```
import matplotlib.pyplot as plt
plt.style.use('../../mdap.mplstyle')
houston = temp['Houston'].rolling('14D', min_periods=1).agg(['max', 'mean', 'min'])
houston.plot(figsize=(6, 2), style=['-', '--', '-'])
```



7.5 5. Grouping by Time and another Column

[77]:
```
import pandas as pd
energy = pd.read_csv('../data/energy_consumption.csv', parse_dates=['date'],
                     index_col='date')
energy.head()
```

date	source	energy (btu)
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

Exercise 1

Find the average energy consumption per sector per 10 year time span beginning from the first year of data. Return the results using both `groupby` and `pivot_table`.

[78]:
```
tg = pd.Grouper(freq='10YS')
energy.groupby(['source', tg]).agg({'energy (btu)':['sum', '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

[79]:
```
energy.pivot_table(index=tg, columns='source',
                   values='energy (btu)', aggfunc='sum').astype('int')
```

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

Use the bikes dataset for the remaining exercises

Execute the following cell to read in the bikes dataset. Note, that it does NOT set the index to be a datetime.

[80]:
```
bikes = pd.read_csv('../data/bikes.csv', parse_dates=['starttime', 'stoptime'])
bikes.head(3)
```

	gender	starttime	stoptime	tripduration	from_station_name	...	to_station_name	end_capacity	temperature	wind_speed	events
0	Male	2013-06-28 19:01:00	2013-06-28 19:17:00	993	Lake Shore Dr & Monroe St	...	Michigan Ave & Oak St	15.0	73.9	12.7	mostlycloudy
1	Male	2013-06-28 22:53:00	2013-06-28 23:03:00	623	Clinton St & Washington Blvd	...	Wells St & Walton St	19.0	69.1	6.9	partlycloudy
2	Male	2013-06-30 14:43:00	2013-06-30 15:01:00	1040	Sheffield Ave & Kingsbury St	...	Dearborn St & Monroe St	23.0	73.0	16.1	mostlycloudy

3 rows × 11 columns

Exercise 2

Filter the data so that it only contains rows from the five most frequent `from_station_name` values. Then find the mean temperature at every station for every quarter. Present the result as a pivot table.

[81]:
```
top5 = bikes['from_station_name'].value_counts().index[:5]
top5
```

[81]:
```
Index(['Clinton St & Washington Blvd', 'Canal St & Adams St',
       'Clinton St & Madison St', 'Canal St & Madison St',
       'Columbus Dr & Randolph St'],
      dtype='object')
```

[82]:
```
df = bikes.query('from_station_name in @top5')
df.head()
```

	gender	starttime	stoptime	tripduration	from_station_name	...	to_station_name	end_capacity	temperature	wind_speed	events
1	Male	2013-06-28 22:53:00	2013-06-28 23:03:00	623	Clinton St & Washington Blvd	...	Wells St & Walton St	19.0	69.1	6.9	partlycloudy
8	Male	2013-07-03 15:21:00	2013-07-03 15:42:00	1300	Clinton St & Washington Blvd	...	Wood St & Division St	15.0	71.1	0.0	cloudy
34	Male	2013-07-12 18:13:00	2013-07-12 18:40:00	1616	Clinton St & Madison St	...	Damen Ave & Pierce Ave	19.0	78.1	10.4	partlycloudy
43	Female	2013-07-15 08:25:00	2013-07-15 08:28:00	185	Clinton St & Madison St	...	Canal St & Jackson Blvd	35.0	82.0	0.0	partlycloudy
60	Male	2013-07-17 10:23:00	2013-07-17 10:40:00	1024	Clinton St & Washington Blvd	...	Larrabee St & Menomonee St	15.0	88.0	5.8	partlycloudy

5 rows × 11 columns

This filters the data from 50k rows down to 3.4k.

[83]:
```
df.shape
```

[83]:
```
(3419, 11)
```

[84]:
```
bikes.shape
```

[84]:
```
(50089, 11)
```

[85]:
```
tg = pd.Grouper(freq='Q', key='starttime')
df.pivot_table(index=tg, columns='from_station_name',
               values='temperature', aggfunc='mean').round(1)
```

from_station_name	Canal St & Adams St	Canal St & Madison St	Clinton St & Madison St	Clinton St & Washington Blvd	Columbus Dr & Randolph St
starttime					
2013-06-30	NaN	NaN	NaN	69.1	NaN
2013-09-30	61.9	75.7	74.8	70.9	73.6
2013-12-31	38.7	45.1	41.2	44.4	43.6
2014-03-31	35.8	27.8	26.7	29.1	26.0
2014-06-30	61.4	64.2	66.1	62.0	65.6
2014-09-30	68.2	70.7	70.9	70.3	75.1
2014-12-31	40.8	42.7	45.8	43.1	44.8
2015-03-31	25.5	24.4	32.6	32.7	24.7
2015-06-30	60.9	63.9	63.9	61.3	66.6
2015-09-30	72.7	72.5	70.6	70.9	75.1
2015-12-31	49.6	49.8	50.1	49.7	54.2
2016-03-31	33.8	42.4	40.1	34.7	36.3
2016-06-30	59.8	66.1	67.4	62.3	67.6
2016-09-30	74.1	77.7	76.3	75.6	75.6
2016-12-31	51.3	49.4	46.2	50.2	47.0
2017-03-31	36.3	36.9	40.5	40.1	37.9
2017-06-30	63.1	66.7	66.5	62.3	67.5
2017-09-30	73.4	74.0	73.4	71.9	76.2
2017-12-31	48.6	50.8	49.7	46.6	53.2

Exercise 3

Find the number of rides per day from each `from_station_name`.

[86]:
```
s = bikes.groupby(['from_station_name', pd.Grouper(freq='D', key='starttime')]).size()
s.tail()
```

[86]:
```
from_station_name             starttime
Woodlawn Ave & Lake Park Ave  2017-02-01    1
                              2017-03-20    1
                              2017-04-25    1
                              2017-06-10    1
Yates Blvd & 75th St          2015-09-18    1
dtype: int64
```

Exercise 4

Reset the `from_station_name` index level from the solution in exercise 3 and then perform a 100 day rolling window of each `from_station_name` calculating the number of rides in this group.

[87]:
```
df = s.reset_index('from_station_name', name='size')
df.head()
```

starttime	from_station_name	size
2016-08-13	2112 W Peterson Ave	1
2016-09-01	2112 W Peterson Ave	1
2016-09-06	2112 W Peterson Ave	1
2016-09-29	2112 W Peterson Ave	1
2016-10-30	2112 W Peterson Ave	1

[88]:
```
df.groupby('from_station_name').rolling('100D')['size'].sum()
```

[88]:
```
from_station_name             starttime
2112 W Peterson Ave           2016-08-13    1.0
                              2016-09-01    2.0
                              2016-09-06    3.0
                              2016-09-29    4.0
                              2016-10-30    5.0
                                            ...
Woodlawn Ave & Lake Park Ave  2017-02-01    1.0
                              2017-03-20    2.0
                              2017-04-25    3.0
                              2017-06-10    3.0
Yates Blvd & 75th St          2015-09-18    1.0
Name: size, Length: 45150, dtype: float64
```

7.6 6. More Time Series Functionality

[89]:
```
import pandas as pd
df = pd.read_csv('../data/stocks/stocks10.csv', parse_dates=['date'],
                 index_col='date')
df.head(3)
```

date	MSFT	AAPL	SLB	AMZN	TSLA	XOM	WMT	T	FB	V
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

Exercise 1

Select the first three trading days of every sixth year.

[90]:
```
df.groupby(pd.Grouper(freq='6AS')).head(3)
```

date	MSFT	AAPL	SLB	AMZN	TSLA	XOM	WMT	T	FB	V
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
2005-01-03	19.47	3.95	23.73	44.52	NaN	33.26	38.41	11.54	NaN	NaN
2005-01-04	19.54	3.99	23.55	42.14	NaN	33.04	38.32	11.41	NaN	NaN
2005-01-05	19.50	4.02	23.62	41.77	NaN	32.87	38.37	11.47	NaN	NaN
2011-01-03	22.56	41.09	66.87	184.22	26.62	56.09	43.74	18.26	NaN	16.50
2011-01-04	22.65	41.30	65.26	185.01	26.67	56.35	43.91	18.38	NaN	16.52
2011-01-05	22.57	41.64	66.03	187.42	26.83	56.20	43.62	18.45	NaN	16.87
2017-01-03	59.50	111.29	76.48	753.67	216.99	81.20	64.37	36.39	116.86	78.05
2017-01-04	59.23	111.16	77.05	757.18	226.99	80.30	64.74	36.18	118.69	78.69
2017-01-05	59.23	111.73	76.93	780.45	226.75	79.11	64.88	36.08	120.67	79.61

Exercise 2

Select the Wednesday in the 19th week of all the leap years.

[91]:
```
filt1 = df.index.isocalendar()['week'] == 19
filt2 = df.index.day_name() == "Wednesday"
filt3 = df.index.is_leap_year
filt = filt1 & filt2 * filt3
df[filt]
```

date	MSFT	AAPL	SLB	AMZN	TSLA	XOM	WMT	T	FB	V
2000-05-10	21.36	3.10	24.94	53.31	NaN	23.90	39.59	17.30	NaN	NaN
2004-05-05	17.13	1.66	21.17	44.26	NaN	28.70	39.93	11.19	NaN	NaN
2008-05-07	22.22	22.76	78.56	73.18	NaN	62.63	43.05	20.50	NaN	20.37
2012-05-09	25.63	70.96	56.58	222.98	30.06	64.51	49.28	22.00	NaN	27.75
2016-05-11	47.58	87.71	65.33	713.23	208.96	77.98	61.40	32.38	119.52	75.60

Exercise 3

Shift all dates in the index up three weeks ending on Sunday.

[92]:
```
df.shift(3, 'W').head(10)
```

date	MSFT	AAPL	SLB	AMZN	TSLA	XOM	WMT	T	FB	V
1999-11-14	29.84	2.32	17.02	82.75	NaN	21.45	38.99	16.78	NaN	NaN
1999-11-14	29.82	2.34	16.65	81.25	NaN	20.89	37.11	17.28	NaN	NaN
1999-11-14	29.33	2.38	16.52	75.94	NaN	20.80	36.94	18.27	NaN	NaN
1999-11-14	29.01	2.43	16.59	71.00	NaN	21.19	38.85	19.79	NaN	NaN
1999-11-14	29.88	2.50	17.21	70.62	NaN	21.47	39.25	20.00	NaN	NaN
1999-11-21	29.82	2.42	17.76	69.12	NaN	21.67	39.20	20.10	NaN	NaN
1999-11-21	29.88	2.50	17.02	66.44	NaN	21.48	39.12	19.24	NaN	NaN
1999-11-21	29.70	2.54	17.23	65.81	NaN	21.16	38.85	19.17	NaN	NaN
1999-11-21	29.61	2.61	17.09	63.06	NaN	21.16	39.03	19.39	NaN	NaN
1999-11-21	29.55	2.75	16.27	64.94	NaN	20.47	40.47	19.43	NaN	NaN

Exercise 4

Create a DatetimeIndex object containing every Friday in the year 2021.

[93]:
```
pd.date_range('2021-1-1', '2021-12-31', freq='W-FRI')
```

[93]:
```
DatetimeIndex(['2021-01-01', '2021-01-08', '2021-01-15', '2021-01-22',
               '2021-01-29', '2021-02-05', '2021-02-12', '2021-02-19',
               '2021-02-26', '2021-03-05', '2021-03-12', '2021-03-19',
               '2021-03-26', '2021-04-02', '2021-04-09', '2021-04-16',
               '2021-04-23', '2021-04-30', '2021-05-07', '2021-05-14',
               '2021-05-21', '2021-05-28', '2021-06-04', '2021-06-11',
               '2021-06-18', '2021-06-25', '2021-07-02', '2021-07-09',
               '2021-07-16', '2021-07-23', '2021-07-30', '2021-08-06',
               '2021-08-13', '2021-08-20', '2021-08-27', '2021-09-03',
               '2021-09-10', '2021-09-17', '2021-09-24', '2021-10-01',
               '2021-10-08', '2021-10-15', '2021-10-22', '2021-10-29',
               '2021-11-05', '2021-11-12', '2021-11-19', '2021-11-26',
               '2021-12-03', '2021-12-10', '2021-12-17', '2021-12-24',
               '2021-12-31'],
              dtype='datetime64[ns]', freq='W-FRI')
```

Exercise 5

Starting from January 1, 1900, create 20 timestamps separated by 57 days each.

[94]:
```
pd.date_range('January 1, 1900', periods=20, freq='57D')
```

[94]:
```
DatetimeIndex(['1900-01-01', '1900-02-27', '1900-04-25', '1900-06-21',
               '1900-08-17', '1900-10-13', '1900-12-09', '1901-02-04',
               '1901-04-02', '1901-05-29', '1901-07-25', '1901-09-20',
               '1901-11-16', '1902-01-12', '1902-03-10', '1902-05-06',
               '1902-07-02', '1902-08-28', '1902-10-24', '1902-12-20'],
              dtype='datetime64[ns]', freq='57D')
```

Exercise 6

Using January 1, 2021 as the end date. Find the previous 15 timestamps separated by 23 month starts

[95]:
```
pd.date_range(end='January 31, 2021', periods=15, freq='23MS')
```

[95]:
```
DatetimeIndex(['1994-03-01', '1996-02-01', '1998-01-01', '1999-12-01',
               '2001-11-01', '2003-10-01', '2005-09-01', '2007-08-01',
               '2009-07-01', '2011-06-01', '2013-05-01', '2015-04-01',
               '2017-03-01', '2019-02-01', '2021-01-01'],
              dtype='datetime64[ns]', freq='23MS')
```

Exercise 7

You are running an experiment beginning at 8:32 a.m. and concluding at 4 p.m. You need to check in on it every 19 minutes. Create a TimedeltaIndex of the check-in times.

[96]:
```
pd.timedelta_range('8:32:00', '16:00:00', freq='19min')
```

[96]:
```
TimedeltaIndex(['0 days 08:32:00', '0 days 08:51:00', '0 days 09:10:00',
                '0 days 09:29:00', '0 days 09:48:00', '0 days 10:07:00',
                '0 days 10:26:00', '0 days 10:45:00', '0 days 11:04:00',
                '0 days 11:23:00', '0 days 11:42:00', '0 days 12:01:00',
                '0 days 12:20:00', '0 days 12:39:00', '0 days 12:58:00',
                '0 days 13:17:00', '0 days 13:36:00', '0 days 13:55:00',
                '0 days 14:14:00', '0 days 14:33:00', '0 days 14:52:00',
                '0 days 15:11:00', '0 days 15:30:00', '0 days 15:49:00'],
               dtype='timedelta64[ns]', freq='19T')
```

Part VIII

Regular Expressions

Chapter 8

Solutions

[1]:
```
import re
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']
def find_pattern(s, pattern, **kwargs):
    filt = s.str.contains(pattern, **kwargs)
    return s[filt]
```

Exercise 1

Find all movies that have 2 consecutive z's in them.

[3]:
```
find_pattern(title, r'zz')
```

[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]:
```
find_pattern(title, r'^9')
```

[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]:
```
find_pattern(title, r'^..b').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]:
```
find_pattern(title, r'M..e$')
```

[6]:
```
704            The Green Mile
1167                    8 Mile
1616                 Like Mike
2122           Moonlight Mile
2486             How She Move
2653                 The Muse
2913    Max Keeble's Big Move
3215    Into the Grizzly Maze
3406               Magic Mike
3696                     Made
4805        The World Is Mine
Name: title, dtype: object
```

Exercise 5

Use a regular expression to find movies that are exactly 6 characters in length.

[7]:
```
find_pattern(title, r'^......$').head()
```

[7]:
```
0      Avatar
41     Cars 2
58     WALL·E
125    Frozen
168    Sahara
Name: title, dtype: object
```

Exercise 6

Complete exercise 5 using a different string-only Series method that does not require 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
```

Exercise 7

Find all movies containing the letter `'q'` ignoring case.

[9]:
```
find_pattern(title, r'q', flags=re.I).head()
```

[9]:
```
12                           Quantum of Solace
73                               Suicide Squad
590    Alvin and the Chipmunks: The Squeakquel
762                             Gangster Squad
839                              The Equalizer
Name: title, dtype: object
```

8.2 2. Quantifiers

[10]:
```
movie = pd.read_csv('../data/movie.csv')
title = movie['title']
```

Exercise 1

Find all movies that have `'z'` as their 15th character.

[11]:
```
find_pattern(title, r'^.{14}z')
```

[11]:
```
2484      American Dreamz
2625    Ramona and Beezus
Name: title, dtype: object
```

Exercise 2

Find all movies that have the word `'Boy'` or `'Boys'` in them followed by a space.

[12]:
```
find_pattern(title, r'Boys? ')
```

[12]:
```
188                       Bad Boys II
1864        Jimmy Neutron: Boy Genius
```

```
2528                    Boys and Girls
2859    The Boy in the Striped Pajamas
2907              The Boys from Brazil
3823                 The Boy Next Door
4166                    Boys Don't Cry
4575      All the Boys Love Mandy Lane
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?

[13]:
```
m40_43 = find_pattern(title, r'^.{40,43}$')
m40_43.head()
```

[13]:
```
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
```

[14]:
```
m40_43.str.len().head()
```

[14]:
```
1     40
4     42
13    42
16    40
18    43
Name: title, dtype: int64
```

[15]:
```
m40_43.str.len().value_counts()
```

[15]:
```
40    15
41    12
42     9
43     9
Name: title, dtype: int64
```

Exercise 4

Find all movies that begin with 'The' and end in 'Movie'.

[16]:
```
find_pattern(title, r'^The.*Movie$')
```

[16]:
```
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

Find all movies that begin with 'The' and end in 'Movie' and have no more than 10 characters between these two words.

[17]:
```
find_pattern(title, r'^The.{,10}Movie$')
```

[17]:
```
319      The Peanuts Movie
569     The Simpsons Movie
759         The Lego Movie
1734     The Rugrats Movie
2162      The Tigger Movie
3296      The Muppet Movie
Name: title, dtype: object
```

Exercise 6

Find all movies that begin with 'The' and end in 'Movie' and have at least 30 characters between these two words.

[18]:
```
find_pattern(title, r'^The.{30,}Movie$')
```

[18]:
```
2645    The Pirates Who Don't Do Anything: A VeggieTal…
Name: title, dtype: object
```

Exercise 7

Find all movies that begin with capital `G` followed by at least one `o`, followed by a `d`.

[19]:
```
find_pattern(title,  r'^Go+d')
```

[19]:
```
98             Godzilla Resurgence
162                  Gods of Egypt
874              Gods and Generals
1688                       Godsend
1886                    Goodfellas
1906                 Good Luck Chuck
2462                       Good Boy!
2551                            Good
2735                    Good Deeds
2787         Good Morning, Vietnam
3037             Good Will Hunting
3197               Good Intentions
3412    Good Night, and Good Luck.
```

```
3509               Good Bye Lenin!
3558               Goddess of Love
3919             Gods and Monsters
4438              God's Not Dead 2
4440                 Godzilla 2000
4649                     Good Kill
4793                     Good Dick
Name: title, dtype: object
```

Exercise 8

Find all movies have either `Free` or `Fee` in them.

[20]:
```
find_pattern(title, r'Fr?ee')
```

[20]:
```
174                      Happy Feet 2
266             Live Free or Die Hard
391                        Happy Feet
856                        Free Birds
1001              Free State of Jones
1507    Mandela: Long Walk to Freedom
1712             The Color of Freedom
1727                      Cry Freedom
2114                  Freedom Writers
2719                         Freedom
3181                      Free Style
3463                        Freeheld
4027                         Freeway
4226                     Freeze Frame
4459             20 Feet from Stardom
Name: title, dtype: object
```

Exercise 9

Find all movies that begin with any five characters followed by a space, followed by a `'t'` not case sensitive.

[21]:
```
find_pattern(title, r'^.{5} t', flags=re.I).head()
```

[21]:
```
106    Alice Through the Looking Glass
107                     Shrek the Third
127               Thor: The Dark World
299          Where the Wild Things Are
384               K-19: The Widowmaker
Name: title, dtype: object
```

8.3 3. Or Conditions and Character Classes

[22]:
```
import pandas as pd
title = pd.read_csv('../data/movie.csv')['title']
title.head()
```

[22]:
```
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 either start with `'C'` or end with `'c'`.

[23]:
```
find_pattern(title, r'^C|c$').head()
```

[23]:
```
26                                Titanic
27             Captain America: Civil War
41                                 Cars 2
86     Captain America: The Winter Soldier
118      Charlie and the Chocolate Factory
Name: title, dtype: object
```

Exercise 2

Find all movies that are 6 or less characters in length or between 30 and 33 characters in length.

[24]:
```
find_pattern(title, r'^.{,6}$|^.{30,33}$').head()
```

[24]:
```
0                             Avatar
37    Transformers: Age of Extinction
41                            Cars 2
53     Transformers: Dark of the Moon
56                             Brave
Name: title, dtype: object
```

Exercise 3

Find all movies that have the word `'and'` followed by the word `'the'`.

[25]:
```
find_pattern(title, r'\band the\b').head()
```

[25]:
```
9                 Harry Potter and the Half-Blood Prince
54     Indiana Jones and the Kingdom of the Crystal S…
64     The Chronicles of Narnia: The Lion, the Witch …
81                          Snow White and the Huntsman
100                            The Fast and the Furious
Name: title, dtype: object
```

Exercise 4

Find all movies that have the word `'and'` followed by two words and then followed by the word `'the'`. In this exercise, words are defined as 1 or more consecutive "word" characters.

[26]:
```
find_pattern(title, r'\band \w+ \w+ the\b')
```

[26]:
```
4400    Down and Out with the Dolls
Name: title, dtype: object
```

Exercise 5

Find all movies that begin with `'The'` followed by the next word that begins with digits.

[27]:
```
find_pattern(title, r'^The [0-9]')
```

[27]:
```
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 6

Find all movies that have three consecutive capital letters in them.

[28]:
```
find_pattern(title, r'[A-Z]{3}').head(10)
```

[28]:
```
4         Star Wars: Episode VII - The Force Awakens
40                                        TRON: Legacy
58                                              WALL·E
140                          Mission: Impossible III
177                                           The BFG
233    Star Wars: Episode III - Revenge of the Sith
340                                 Jurassic Park III
440                                             RED 2
589                          AVP: Alien vs. Predator
725                                               RED
Name: title, dtype: object
```

Exercise 7

Find all movies that begin and end with a capital letter.

[29]:
```
find_pattern(title, r'^[A-Z].*[A-Z]$').head()
```

[29]:
```
46                 World War Z
58                      WALL·E
140    Mission: Impossible III
151            Men in Black II
177                    The BFG
Name: title, dtype: object
```

Exercise 8

Find all the movies that have a digit followed by a comma followed by a digit.

[30]:
```
find_pattern(title, r'[0-9],[0-9]')
```

[30]:
```
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 9

Find all the movies that have either an ampersand or a question mark in them.

[31]:
```
find_pattern(title, r'[&?]').head(15)
```

[31]:
```
129                                      Angels & Demons
145                                Mr. Peabody & Sherman
214                                        Batman & Robin
252                                     Mr. & Mrs. Smith
278                                       Town & Country
347    Percy Jackson & the Olympians: The Lightning T…
425              Cats & Dogs: The Revenge of Kitty Galore
437                                         Lilo & Stitch
512                 The Adventures of Rocky & Bullwinkle
703                                          Marley & Me
715                                          Cats & Dogs
740                                       Starsky & Hutch
747                                 Up Close & Personal
813                      Did You Hear About the Morgans?
850                                         Tango & Cash
Name: title, dtype: object
```

Exercise 10

Which movie has the most ampersands, question marks, and periods in it?

[32]:
```
count = title.str.count(r'[&.?]')
count.head()
```

[32]:
```
0    0
1    0
2    0
3    0
4    0
Name: title, dtype: int64
```

[33]:
```
idx = count.idxmax()
idx
```

[33]:
```
542
```

[34]:
```
title.loc[idx]
```

[34]:
```
'The Man from U.N.C.L.E.'
```

Exercise 11

Find all the movies with exactly three words with each word no more than 6 characters in length. For this exercise, a word is defined as consecutive non-space characters followed by exactly one space (or end of string).

[35]:
```
find_pattern(title, r'^\S{,6} \S{,6} \S{,6}$').head()
```

[35]:
```
14    The Lone Ranger
15       Man of Steel
32         Iron Man 3
43        Toy Story 3
46        World War Z
Name: title, dtype: object
```

Exercise 12

Find all movies that have four consecutive non-word characters.

[36]:
```
find_pattern(title, r'\W{4}')
```

[36]:
```
252                      Mr. & Mrs. Smith
3703    Jekyll and Hyde… Together Again
3828         What the #$*! Do We (K)now!?
4498                  The Helix… Loaded
Name: title, dtype: object
```

Exercise 13

Find all movies that have at least one word that ends in `'ats'`.

[37]:
```
find_pattern(title, r'ats\b')
```

[37]:
```
425     Cats & Dogs: The Revenge of Kitty Galore
715                                    Cats & Dogs
1552                             Cats Don't Dance
1593                 Rugrats in Paris: The Movie
1607                                   Hope Floats
1734                            The Rugrats Movie
1896                              Rugrats Go Wild
2013                  The Men Who Stare at Goats
2084                     Josie and the Pussycats
3001                                     Tomcats
3497                                        Bats
3513                                    Mallrats
Name: title, dtype: object
```

Exercise 14

Find all the movies containing, but not ending in `'tes'`.

[38]:
```
find_pattern(title, r'tes\B')
```

[38]:
```
1140                The Sweetest Thing
1941     The Greatest Game Ever Played
1976        The World's Fastest Indian
2238      The Greatest Story Ever Told
2547                The White Countess
2708        Bathory: Countess of Blood
3169              World's Greatest Dad
3569                       The Greatest
3817        The Greatest Show on Earth
4271      The Greatest Movie Ever Sold
4845    The Past is a Grotesque Animal
Name: title, dtype: object
```

Exercise 15

Find all movies containing a word that is at least 7 lowercase letters in length.

[39]:
```
find_pattern(title, r'\b[a-z]{7,}\b')
```

[39]:
```
2310    Les couloirs du temps: Les visiteurs II
2824              Del 1 - Män som hatar kvinnor
3455                                Les visiteurs
3584                         L'auberge espagnole
3761                             Camping sauvage
3925                           La otra conquista
Name: title, dtype: object
```

Exercise 16

Find all movies that have a word that contains, but does not start with `'Z'`.

[40]:
```
find_pattern(title, r'\BZ')
```

[40]:
```
2127    eXistenZ
Name: title, dtype: object
```

Exercise 17

Find all movies containing a word starting with `'W'` and ending with `'w'`.

[41]:
```
find_pattern(title, r'\bW\w*w\b')
```

[41]:
```
1229                Secret Window
2833    The Widow of Saint-Pierre
Name: title, dtype: object
```

Exercise 18

Count the total number of digit characters between 7 and 9 in all of the movies.

[42]:
```
title.str.count(r'[7-9]').sum()
```

[42]:
```
53
```

Exercise 19

Use the `count` method to count the number of words in each title. Use consecutive word characters as the definition of a word.

[43]:
```
title.str.count(r'\b\w+\b').head()
```

[43]:
```
0    1
1    8
2    1
3    4
4    7
Name: title, dtype: int64
```

8.4 4. Grouping and Capturing

Exercise 1

Find all movies that begin with the exact words `'I'`, `'An'`, or `'How'`, are followed by at least 20 characters, and end in lowercase `'n'` through `'s'`. Make sure there is no warning.

[44]:
```
find_pattern(title, r'^(?:I|An|How)\b.{19,}[n-s]$')
```

[44]:
```
93                        How to Train Your Dragon
215                 How the Grinch Stole Christmas
929                   How to Lose a Guy in 10 Days
2009        I Still Know What You Did Last Summer
2406                        I Love You, Beth Cooper
2457              I Know What You Did Last Summer
2847                     I Love You Phillip Morris
3177    An Alan Smithee Film: Burn Hollywood Burn
Name: title, dtype: object
```

Exercise 2

Find all movies that have a lower case consonant followed by a lowercase vowel followed immediately by a repeat of those same two letters. For example, `'Banana'` repeats `'na'` twice in succession.

[45]:
```
find_pattern(title, r'([bcdfghjklmnpqrstvwxyz][aeiou])\1')
```

```
/var/folders/0x/whw882fj4qv0czzqzrngxvdh0000gn/T/ipykernel_40829/1369938214.py:7:
UserWarning: This pattern has match groups. To actually get the groups, use str.extract.
  filt = s.str.contains(pattern, **kwargs)
```

[45]:
```
2294    Princess Mononoke
2559                Ararat
2969             Madadayo
3196            Partition
```

```
4231              Bananas
Name: title, dtype: object
```

Exercise 3

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 the word.

[46]:
```
title.str.extract(r'^The (\d+)').dropna()
```

	0
212	13
429	6
1354	5
1817	40
1958	33
3567	5
4373	41

Exercise 4

Find all movies that have two separate numbers in them. An example would be, '7 days and 7 nights'.

[47]:
```
find_pattern(title, r'\d+\D+\d+')
```

[47]:
```
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 5

Find all movies that have six or more non-vowel and non-space characters in a row.

[48]:
```
find_pattern(title, r'[^aeiouAEIOU\s]{6,}')
```

[48]:
```
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 6

Extract the very next non-space character after 't' or 'T' for each movie, convert to lowercase, and then return the count of each character.

[49]:
```
# extract always returns a DataFramea
title.str.extract(r'[Tt](\S)').dropna().head()
```

	0
0	a
1	e
2	r
3	h
4	a

[50]:
```
# Select first and only column as a Series with `[0]`.
title.str.extract(r'[Tt](\S)')[0].str.lower().value_counts().head(10)
```

[50]:
```
h    1486
e     275
o     198
i     179
a     164
r     150
t      81
y      69
u      53
s      49
Name: 0, dtype: int64
```

[51]:
```
# there is an `expand` paramter that you can set to False to return a Series
title.str.extract(r'[Tt](\S)', expand=False).head()
```

[51]:
```
0    a
1    e
2    r
3    h
4    a
Name: title, dtype: object
```

Exercise 7

Extract all the words that begin with 'T' or 't' and end in 'e' then find their frequency converting to lowercase first.

[52]:
```
title.str.extractall(r'\b([tT]\w*e)\b')[0].str.lower().value_counts().head(10)
```

[52]:
```
the        1555
time         26
tale         12
true         10
three         8
teenage       7
take          6
there         6
trouble       4
trade         4
Name: 0, dtype: int64
```

Exercise 8

Find all movies containing a 6, 7, and 8 character word.

[53]:
```
find_pattern(title, r'^(?=.*\b\w{6}\b)(?=.*\b\w{7}\b)(?=.*\b\w{8}\b)')
```

[53]:
```
10                      Batman v Superman: Dawn of Justice
101                   The Curious Case of Benjamin Button
193               Harry Potter and the Prisoner of Azkaban
209                     Sherlock Holmes: A Game of Shadows
223                        Charlie's Angels: Full Throttle
1239                          Aliens vs. Predator: Requiem
1736                               Florence Foster Jenkins
1775                  How to Lose Friends & Alienate People
2075                         Deuce Bigalow: European Gigolo
2113                                Silver Linings Playbook
2365    Borat: Cultural Learnings of America for Make …
2711          Dungeons & Dragons: Wrath of the Dragon God
2894                                Johnson Family Vacation
3283     The Haunting in Connecticut 2: Ghosts of Georgia
4110    Marilyn Hotchkiss' Ballroom Dancing and Charm …
4373    The 41-Year-Old Virgin Who Knocked Up Sarah Ma…
Name: title, dtype: object
```

Exercise 9

Find all movies that have a `'q'` that is not followed by a `'u'`.

[54]:
```
find_pattern(title, r'q(?!u)', flags=re.I)
```

[54]:
```
1366                                 John Q
3294           Valley of the Wolves: Iraq
4403                                     Q
4566    Iraq for Sale: The War Profiteers
```

```
Name: title, dtype: object
```

Exercise 10

Find all movies containing an `'s'` that is not preceded by a vowel or a single quote mark. The `'s'` cannot be the first or last letter of the word. Ignore case.

[55]:
```
find_pattern(title, r"^.*(?<![aeiou'])\Bs\B", flags=re.I).head()
```

[55]:
```
29                          Jurassic World
35                    Monsters University
36    Transformers: Revenge of the Fallen
37        Transformers: Age of Extinction
50                         The Great Gatsby
Name: title, dtype: object
```

Exercise 11

Find all movies where the first four characters of the first word repeat at least 10 characters after the fourth character somewhere else in the title. Use a named group.

[56]:
```
find_pattern(title, r'^(?P<first_four>\w{4}).{10,}(?P=first_four)')
```

```
/var/folders/0x/whw882fj4qv0czzqzrngxvdh0000gn/T/ipykernel_40829/1369938214.py:7:
UserWarning: This pattern has match groups. To actually get the groups, use str.extract.
  filt = s.str.contains(pattern, **kwargs)
```

[56]:
```
1690    Hoodwinked Too! Hood vs. Evil
3412       Good Night, and Good Luck.
Name: title, dtype: object
```

Exercise 12

Find all movies that repeat one character three or more times in a row.

[57]:
```
find_pattern(title, r'(.)\1\1').head(10)
```

```
/var/folders/0x/whw882fj4qv0czzqzrngxvdh0000gn/T/ipykernel_40829/1369938214.py:7:
UserWarning: This pattern has match groups. To actually get the groups, use str.extract.
  filt = s.str.contains(pattern, **kwargs)
```

[57]:
```
140                          Mission: Impossible III
233     Star Wars: Episode III - Revenge of the Sith
276                                       10,000 B.C.
340                                Jurassic Park III
474                                    Fantasia 2000
697                            3000 Miles to Graceland
810                                         Rambo III
886                            The Godfather: Part III
972                              Beverly Hills Cop III
1191                       Back to the Future Part III
Name: title, dtype: object
```

Exercise 13

Find all movies where a word of at least four characters in length repeats but has a different starting letter. For example, `'Hocus Pocus'` and `'Kill Bill'` both work.

[58]:
```
find_pattern(title, r'\b(\w)(\w{3,})\b.*\b(?!\1)\w\2\b')
```

```
/var/folders/0x/whw882fj4qv0czzqzrngxvdh0000gn/T/ipykernel_40829/1369938214.py:7:
UserWarning: This pattern has match groups. To actually get the groups, use str.extract.
  filt = s.str.contains(pattern, **kwargs)
```

[58]:
```
846                       Kill Bill: Vol. 1
849                       Kill Bill: Vol. 2
1616                              Like Mike
1650    What's the Worst That Could Happen?
1754                             Hocus Pocus
1831                             Pain & Gain
4461                    Walking and Talking
4602        They Will Have to Kill Us First
Name: title, dtype: object
```

Exercise 14

Extract all characters from the first digit through the last. Output the first 10 non-missing rows.

[59]:
```
title.str.extract(r'(\d.*\d)').dropna().head(10)
```

	0
60	2012
85	47
212	13
258	300
268	80
276	10,000
289	1 2 3
384	19
400	102
474	2000

Exercise 15

Extract all characters from the first digit through the second digit. Output the first 10 non-missing rows.

[60]:
```
title.str.extract(r'(\d.*?\d)').dropna().head(10)
```

	0
60	20
85	47
212	13
258	30
268	80
276	10
289	1 2
384	19
400	10
474	20

Exercise 16

Extract all characters from the first word that ends in `'e'` through the next word that ends in `'e'`. Ignore case.

[61]:
```
title.str.extract(r'(\b\w*?e\b.*?e\b)', flags=re.I).dropna()
```

	0
4	Episode VII - The
9	the Half-Blood Prince
14	The Lone
16	The Chronicles of Narnia: Prince
20	The Hobbit: The
...	...
4863	The Naked Ape
4868	The Image
4876	Hide, Die
4900	The Circle
4902	The Cure

684 rows × 1 columns

8.5 5. Multiline Regex Patterns

[62]:
```
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...

[63]:
```
text = news['text']
```

Exercise 1

Find the number of messages that have at least one line that begins with `'The'` and ends with a period.

[64]:
```
find_pattern(text, r'^The.*\.$', flags=re.M).size
```

[64]:
```
15
```

Exercise 2

Find the messages that do not start with the word `'From'`.

[65]:
```
find_pattern(text, r'^(?!From)')
```

[65]:
```
16     Subject: help for school\nFrom: mcrandall@eagl…
23     Subject: Re: PHILS, NL EAST NOT SO WEAK\nFrom:…
66     Organization: University of Illinois at Chicag…
82     Organization: University of Illinois at Chicag…
96     Subject: Life on Mars???\nFrom: schiewer@pa881…
162    Subject: Re: FORSALE: Men Without Hats- Folk o…
176    Nntp-Posting-Host: surt.ifi.uio.no\nFrom: Thom…
180    Subject: After-Market Cruise Controls: Specifi…
195    Subject: CDs for sale [update]\nFrom: koutd@hi…
213    Organization: University of Illinois at Chicag…
214    Subject: Space FAQ 01/15 - Introduction\nFrom:…
226    Organization: University of Illinois at Chicag…
227    Organization: Ryerson Polytechnical Institute\…
253    Subject: Re: Can't Breathe -- Update\nFrom: RG…
306    Subject: EXPERTS on PENICILLIN…LOOK!\nFrom: …
317    Subject: Re: Non-lethal alternatives to handgu…
389    Organization: University of Illinois at Chicag…
399    Subject: STARGARDTS DISEASE\nFrom: kmcvay@oneb…
Name: text, dtype: object
```

Exercise 3

Extract the first word from the messages that do not start with the word `'From'` and then count the occurrences of each of these words.

[66]:
```
text.str.extract('^(?!From)(\w+)').value_counts()
```

[66]:
```
Subject         11
Organization     6
Nntp             1
dtype: int64
```

Exercise 4

Extract the header of each message. These are the lines at the top that begin with `From`, `Subject`, `Organization`, etc… and assign it to a variable named `header`. Take a look at a few of the individual messages to understand what pattern would work best. Make sure `header` is a Series of strings.

[67]:
```
# assume that two newlines separate the body and header
header = text.str.extract(r'^(.*?)\n\n', flags=re.S, expand=False)
header.head()
```

[67]:
```
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…
Name: text, dtype: object
```

Printing out an example of one of the headers

[68]:
```
print(header.iloc[100])
```

```
From: c23reg@kocrsv01.delcoelect.com (Ron Gaskins)
Subject: Re: Dumbest automotive concepts of all tim
Originator: c23reg@koptsw21
Keywords: Dimmer switch location (repost)
Organization: Delco Electronics Corp.
Lines: 22
```

Exercise 5

Use the`extractall` method to extract two groups from the header. For all lines that do not begin with a space, extract the first word up to but not including the colon. That is the first group. Extract the remaining characters to the right of the colon as the second group. Name the groups `'property'` and `'value'`.

[69]:
```
pattern = r'^(?P<property>\S+): (?P<value>.*)'
df = header.str.extractall(pattern, flags=re.M)
df.head(10)
```

		property	value
	match		
0	0	From	nyeda@cnsvax.uwec.edu (David Nye)
	1	Subject	Re: Post Polio Syndrome Information Needed Ple...
	2	Organization	University of Wisconsin Eau Claire
	3	Lines	21
1	0	From	ndallen@r-node.hub.org (Nigel Allen)
	1	Subject	WACO: Clinton press conference, part 1
	2	Organization	R-node Public Access Unix - 1 416 249 5366
	3	Lines	282
2	0	From	mark@ardsley.business.uwo.ca (Mark Bramwell)
	1	Subject	Re: Cellular Phone (Portable) for sale

The second group occasionally spans across multiple lines and when that occurs that line does not begin with a space. See the example of message header 176. Both `'In-Reply-To'` and `'Organization'` span multiple lines.

[70]:
```
print(header.loc[176])
```

```
Nntp-Posting-Host: surt.ifi.uio.no
From: Thomas Parsli <thomasp@ifi.uio.no>
Subject: Re: Rewording the Second Amendment (ideas)
In-Reply-To: arc@cco.caltech.edu (Aaron Ray Clements)'s message of 21 Apr
        1993 12:34:51 GMT
Organization: Dept. of Informatics, University of Oslo, Norway
        <1993Apr20.083057.16899@ousrvr.oulu.fi>
        <viking.735378520@ponderous.cc.iastate.edu>
        <1993Apr21.091130.17788@ousrvr.oulu.fi>
        <1r3f1bINN3n6@gap.caltech.edu>
Lines: 24
Originator: thomasp@surt.ifi.uio.no
```

Our original pattern did not extract these extra lines.

[71]:
```
df.loc[176]
```

match	property	value
0	Nntp-Posting-Host	surt.ifi.uio.no
1	From	Thomas Parsli <thomasp@ifi.uio.no>
2	Subject	Re: Rewording the Second Amendment (ideas)
3	In-Reply-To	arc@cco.caltech.edu (Aaron Ray Clements)'s mes...
4	Organization	Dept. of Informatics, University of Oslo, Norway
5	Lines	24
6	Originator	thomasp@surt.ifi.uio.no

To take care of this edge case, we use the dotall option so that the second group continues to match across newlines. A positive lookahead is used to match the end of a line, but a negative lookahead is used to make sure there is not a space after the newline.

[72]:
```
pattern = r'^(?P<property>\S+): (?P<value>.*?)(?=$)(?!\n\s)'
df = header.str.extractall(pattern, flags=re.M | re.S)
df.head(10)
```

	match	property	value
0	0	From	nyeda@cnsvax.uwec.edu (David Nye)
	1	Subject	Re: Post Polio Syndrome Information Needed Ple...
	2	Organization	University of Wisconsin Eau Claire
	3	Lines	21
1	0	From	ndallen@r-node.hub.org (Nigel Allen)
	1	Subject	WACO: Clinton press conference, part 1
	2	Organization	R-node Public Access Unix - 1 416 249 5366
	3	Lines	282
2	0	From	mark@ardsley.business.uwo.ca (Mark Bramwell)
	1	Subject	Re: Cellular Phone (Portable) for sale

Exercise 6

Attempt to extract all emails from each message. If you are up for a serious challenge, use the exact specifications for valid email addresses. The solution presented finds the most common types of emails.

[73]:
```
emails = text.str.extractall(r'(\w[\w.-]*?\w@\w[\w.-]*\w)(?<![-._])')[0].str.lower()
emails.head()
```

[73]:
```
   match
0  0                 nyeda@cnsvax.uwec.edu
   1                   keith@actrix.gen.nz
   2                 nyeda@cnsvax.uwec.edu
1  0               ndallen@r-node.hub.org
2  0         mark@ardsley.business.uwo.ca
Name: 0, dtype: object
```

Exercise 7

From each email found, extract the characters after the last dot of each email (usually com, edu, org, etc…), make them lowercase, and count the occurrences of each.

[74]:
```
emails.str.extract('.*\.(.*)$')[0].str.lower().value_counts().head(10)
```

[74]:
```
edu       676
com       360
gov        69
ca         49
org        29
bitnet     13
net        13
uucp       12
au         11
uk         10
Name: 0, dtype: int64
```

Exercise 8

Extract the number of lines of each message as the integer following the word `'Lines:'` in the header. Find the average number of lines per message.

[75]:
```
lines = text.str.extract('Lines: (\d+)', expand=False).astype('float64')
lines.head()
```

[75]:
```
0     21.0
1    282.0
2     19.0
3     12.0
4     62.0
Name: text, dtype: float64
```

[76]:
```
lines.mean()
```

[76]:
```
36.94486215538847
```

Exercise 9

Extract all 10-digit phone numbers.

[77]:
```
# we allow up to two non-digits to separate the first and second set of numbers
# and up to one non-digit between the last two sets of numbers
s = text.str.extractall(r'\b(\d{3}\D{1,2}\d{3}\D?\d{4})\b').dropna()[0]
s.head()
```

[77]:
```
    match
1   0          416 249 5366
    1          202-456-2100
2   0         519) 661-3714
5   0         303) 493-6674
14  0          408 241-9760
Name: 0, dtype: object
```

Can go further and replace all of the characters between the first and second set of numbers with a hyphen.

[78]:
```
s.replace(r'\D{1,2}', '-', regex=True).head()
```

[78]:
```
    match
1   0         416-249-5366
    1         202-456-2100
2   0         519-661-3714
5   0         303-493-6674
14  0         408-241-9760
Name: 0, dtype: object
```

Exercise 10

In this exercise you'll find the most common words in each category. Run the first two cells below to place the category in the index and then extract the body from the message as a Series. Extract all the words between 5 and 12 characters in length. Make them all lowercase and remove the words in the list `remove_words`. Finally, count occurrence of each word by category returning the top 10 per category.

[79]:
```
textc = news.set_index('category')['text']
textc.head(3)
```

[79]:
```
category
sci.med                From: nyeda@cnsvax.uwec.edu (David Nye)\nSubje…
talk.politics.guns     From: ndallen@r-node.hub.org (Nigel Allen)\nSu…
misc.forsale           From: mark@ardsley.business.uwo.ca (Mark Bramw…
Name: text, dtype: object
```

[80]:
```
body = textc.str.extract(r'\n\n(.*)', flags=re.S, expand=False)
body.head(3)
```

[80]:
```
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…
Name: text, dtype: object
```

[81]:
```
remove_words = ['would', 'article', 'there', 'about', 'their', 'other', 'should',␣
↪'could',
                'those', 'these', 'which', 'where', 'writes', 'anyone', 'someone',␣
↪'because']
```

[82]:
```
words = body.str.extractall(r'\b(\w{5,12})\b')[0].str.lower()
words = words[~words.isin(remove_words)]
words.head(10)
```

[82]:
```
category  match
sci.med   0              reply
          1              keith
          2             actrix
          3              keith
          4            stewart
          5             become
          6          interested
          7             through
          8        acquaintance
          9              polio
Name: 0, dtype: object
```

[83]:
```
word_count = words.groupby('category').value_counts()
top_word_count = word_count.groupby('category').head(10)
top_word_count
```

[83]:
```
category            0
misc.forsale        offer          34
                    condition      22
                    asking         20
                    drive          19
                    please         18
                    shipping       17
                    windows        16
                    price          15
                    excellent      14
                    interested     14
rec.autos           think          30
                    drive          23
                    right          22
                    better         19
                    little         18
                    speed          18
                    people         17
                    engine         15
                    power          15
                    without        15
rec.sport.baseball  alomar         39
                    think          37
                    better         33
                    games          28
```

```
                    pitcher        25
                    season         25
                    baseball       24
                    baerga         20
                    league         20
                    first          19
sci.med             people         27
                    patients       23
                    gordon         19
                    think          19
                    banks          17
                    really         17
                    something      17
                    group          15
                    science        15
                    disease        14
sci.space           space         209
                    lunar          89
                    earth          74
                    first          71
                    orbit          64
                    probe          60
                    mission        52
                    planetary      49
                    surface        48
                    system         48
talk.politics.guns  people        104
                    think          47
                    weapons        45
                    believe        44
                    right          38
                    against        36
                    state          36
                    without        36
                    children       34
                    firearms       34
Name: 0, dtype: int64
```

Exercise 11

Find all messages that have have `'From'`, `'Subject'`, `'Organization'`, and `'Lines'` as parts of the header.

[84]:
```
pattern = r'^(?=.*^From)(?=.*^Subject)(?=.*^Organization)(?=.*^Lines)'
find_pattern(header, pattern, flags=re.M | re.S).head()
```

[84]:
```
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…
Name: text, dtype: object
```

8.6 Project - Feature Engineering on the Titanic

[85]:
```
titanic = pd.read_csv('../data/titanic.csv')
titanic.head()
```

	PassengerId	Survived	Pclass	Name	Sex	...	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	...	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	...	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	...	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	...	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	...	0	373450	8.0500	NaN	S

5 rows × 12 columns

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?

[86]:
```
titanic['ticket_first'] = titanic.Ticket.str[0]
titanic.head(3)
```

	PassengerId	Survived	Pclass	Name	Sex	...	Ticket	Fare	Cabin	Embarked	ticket_first
0	1	0	3	Braund, Mr. Owen Harris	male	...	A/5 21171	7.2500	NaN	S	A
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	...	PC 17599	71.2833	C85	C	P
2	3	1	3	Heikkinen, Miss. Laina	female	...	STON/O2. 3101282	7.9250	NaN	S	S

3 rows × 13 columns

[87]:
```
titanic.groupby('ticket_first').agg({'Survived': ['mean', 'sum', 'size']})
```

	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

[88]:
```
# overall survival rate
titanic['Survived'].mean()
```

[88]: 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`.

[89]:
```
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?

[90]:
```
titanic['ticket_last'] = titanic['Ticket'].str[-1]
titanic.groupby('ticket_last').agg({'Survived': ['mean', 'sum', 'size']})
```

	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?

[91]:
```
titanic['name_len'] = titanic['Name'].str.len()
titanic['name_len'].agg(['min', 'max'])
```

[91]:
```
min    12
max    82
Name: name_len, dtype: int64
```

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.

[92]:
```
titanic['name_len_cat'] = pd.cut(titanic['name_len'], bins=[0, 20, 40, 60, 80, 100])
titanic['name_len_cat'].head()
```

[92]:
```
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, right]): [(0, 20] < (20, 40] < (40, 60] < (60, 80] < (80,
100]]
```

[93]:
```
titanic['name_len_cat'].value_counts()
```

[93]:
```
(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?

[94]:
```
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

[95]:
```
names = titanic.sort_values(by='name_len')['Name']
names.head(10)
```

[95]:
```
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
```

[96]:
```
# Names exceed pandas display settings.
# change them with pd.options.display.max_colwidth
# or just print out values
names.tail(10).values
```

[96]:
```
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)
```

[97]:
```
# 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.

[98]:
```
titanic['title'] = titanic['Name'].str.extract(r'(\w+[.])')
titanic['title'].value_counts()
```

[98]:
```
Mr.          517
Miss.        182
Mrs.         125
Master.       40
Dr.            7
Rev.           6
Mlle.          2
Major.         2
Col.           2
Countess.      1
Capt.          1
Ms.            1
Sir.           1
Lady.          1
Mme.           1
Don.           1
Jonkheer.      1
Name: title, dtype: int64
```

Exercise 9

Does the title have good predictive value of survival?

[99]:
```
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

[100]:
```
titanic.pivot_table(index='title', columns='Sex',
                    values='Survived', aggfunc=['mean', 'size'])
```

	mean		size	
Sex	female	male	female	male
title				
Capt.	NaN	0.000000	NaN	1.0
Col.	NaN	0.500000	NaN	2.0
Countess.	1.000000	NaN	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	NaN	1.0	NaN
Major.	NaN	0.500000	NaN	2.0
Master.	NaN	0.575000	NaN	40.0
Miss.	0.697802	NaN	182.0	NaN
Mlle.	1.000000	NaN	2.0	NaN
Mme.	1.000000	NaN	1.0	NaN
Mr.	NaN	0.156673	NaN	517.0
Mrs.	0.792000	NaN	125.0	NaN
Ms.	1.000000	NaN	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

[101]:
```
pattern = r'\w+[.] (\w+)'
titanic['first_name'] = titanic['Name'].str.extract(pattern)
```

To be more precise, we can do this:

[102]:
```
pattern = r'\w+[.][a-z (]+([A-Z]\w+)'
titanic['first_name'] = titanic['Name'].str.extract(pattern)
```

[103]:
```
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

[104]:
```
filt = first_name_ct['Sex'] == 2
first_name_ct[filt].head(10)
```

first_name	Sex
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.

[105]:
```
filt = titanic['first_name'] == 'Albert'
titanic.loc[filt, 'Name']
```

[105]:
```
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.

[106]:
```
titanic['cabin_first'] = titanic.Cabin.str[0].fillna('Missing')
```

Just having a cabin is highly predictive.

[107]:
```
titanic.groupby('cabin_first').agg({'Survived': ['size', 'mean']})#.
 ↪sort_values('size', ascending=False)
```

	Survived	
	size	mean
cabin_first		
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	237000000.0	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

2 rows × 22 columns

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
Steve Buscemi     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()
```

race	Asian	Black	Hispanic	Unknown	White
cause					
Accidents	53.000000	158.000000	154.000000	13.000000	297.000000
Cancer	528.000000	1523.000000	1013.000000	74.000000	3356.000000
Diabetes	47.000000	246.000000	177.000000	20.000000	237.000000
Flu and Pneumonia	66.000000	229.000000	189.000000	20.000000	530.000000
HIV	nan	377.000000	218.000000	6.000000	nan
Heart Disease	496.000000	2121.000000	1327.000000	122.000000	5632.000000
Homicide	nan	267.000000	114.000000	nan	nan
Hypertension	25.000000	137.000000	nan	nan	123.000000
Liver Disease	18.000000	nan	107.000000	nan	nan
Other	221.000000	1163.000000	988.000000	67.000000	1749.000000
Respiratory Diseases	43.000000	137.000000	nan	nan	345.000000
Stroke	56.000000	213.000000	126.000000	7.000000	267.000000
Substance Abuse	nan	163.000000	183.000000	10.000000	261.000000
Suicide	36.000000	nan	nan	5.000000	187.000000

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

5 rows × 14 columns

[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

16 rows × 20 columns

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	64.000000	62.600000
cloudy	61.000000	57.000000
fog	54.000000	52.000000
hazy	60.100000	57.900000
mostlycloudy	72.000000	71.100000
partlycloudy	71.100000	69.100000
rain	59.000000	57.900000
sleet	30.900000	32.000000
snow	28.450000	28.000000
tstorms	74.450000	75.900000
unknown	-9999.000000	50.000000

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 DatFrame 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',
       'Johnny Depp', 'Steve Buscemi', 'Brad Pitt', 'Nicolas Cage',
```

```
       'Will Ferrell', 'Liam Neeson'],
      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
28	PG-13	Liam Neeson

[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						
Brad Pitt	0	0	3	10	20	0
Bruce Willis	0	0	2	15	21	0
Johnny Depp	0	0	7	13	15	1
Liam Neeson	0	1	4	11	15	0
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

5 rows × 12 columns

[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()
```

	status	Delayed	On Time
airline	**airport**		
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 1. Automatic Index Alignment

Exercise 1

Create two Series of integers with no missing values. Make one with 4 values and the other with five. When added together, they result should be a Series with 10 values, 2 of which are missing.

[1]:
```
import pandas as pd
import numpy as np
s1 = pd.Series(index=['a', 'a', 'a', 'b', 'b', 'd'], data=[1, 2, 3, 4, 5, 6])
s2 = pd.Series(index=['a', 'a', 'b', 'c'], data=[1, 2, 3, 4])
s1 + s2
```

[1]:
```
a    2.0
a    3.0
a    3.0
a    4.0
a    4.0
a    5.0
b    7.0
b    8.0
c    NaN
d    NaN
dtype: float64
```

[2]:
```
len(s1 + s2)
```

[2]:
```
10
```

Exercise 2

Create two Series of integers, each with three values, but with a non-identical index. When added together, the result should be a Series with three values.

[3]:
```
s1 = pd.Series(index=['a', 'b', 'c'], data=[1, 2, 3])
s2 = pd.Series(index=['c', 'b', 'a'], data=[4, 8, 3])
s1 + s2
```

[3]:
```
a     4
b    10
c     7
dtype: int64
```

Exercise 3

Add two Series together containing integers resulting in a new Series with all missing values.

[4]:
```
s1 = pd.Series(index=['a', 'b', 'c'], data=[1, 2, 3])
s2 = pd.Series(index=['d', 'e', 'f'], data=[4, 8, 3])
s1 + s2
```

[4]:
```
a   NaN
b   NaN
c   NaN
d   NaN
e   NaN
f   NaN
dtype: float64
```

Exercise 4

You add two Series together, one with four values, and the other with five. Each index label is the same. For instance, all labels for all Series could be `'a'`. How many total values would be in the resulting Series? Answer the question without pandas, and then check your work with it.

[5]:
```
# 20 values - cartesian product 5 * 4
s1 = pd.Series(index=['a', 'a', 'a', 'a', 'a'], data=[1, 2, 3, 4, 5])
s2 = pd.Series(index=['a', 'a', 'a', 'a'], data=[4, 8, 3, 4])
s1 + s2
```

[5]:
```
a     5
a     9
a     4
a     5
a     6
a    10
a     5
a     6
a     7
a    11
a     6
a     7
a     8
a    12
a     7
a     8
a     9
a    13
a     8
a     9
```

```
dtype: int64
```

[6]:
```
len(s1 + s2)
```

[6]:
```
20
```

Exercise 5

Can you determine the shape of the resulting addition between the following two DataFrames before completing the operation?

[7]:
```
df1 = pd.DataFrame(data=np.random.randint(1, 6, (4, 8)),
                   index=['a', 'b', 'b', 'd'],
                   columns=['a', 'b', 'c', 'd',
                            'e', 'f', 'g', 'h'])
df2 = pd.DataFrame(data=np.random.randint(1, 6, (6, 5)),
                   index=['a', 'b', 'b', 'b', 'b', 'c'],
                   columns=['a', 'b', 'c', 'd', 'e'])
display(df1, df2)
```

	a	b	c	d	e	f	g	h
a	3	1	1	3	1	2	1	5
b	2	1	2	1	4	4	3	4
b	2	1	3	4	5	4	5	2
d	5	5	1	5	3	3	4	1

[8]:
```
# 11 by 8
(df1 + df2).shape
```

[8]:
```
(11, 8)
```

Exercise 6

Read in the sample dataset (`sample_data.csv`), placing the name column in the index. Create a new Series with two values and use it to create a new column in the DataFrame. Make it such that only one of the values appears in the resulting DataFrame.

[9]:
```
df = pd.read_csv('../data/sample_data.csv', index_col='name')
s = pd.Series(index=['Dean', 'Thomas'], data=[99, 2])
df['new_values'] = s
df
```

name	state	color	food	age	height	score	new_values
Jane	NY	blue	Steak	30	165	4.6	NaN
Niko	TX	green	Lamb	2	70	8.3	NaN
Aaron	FL	red	Mango	12	120	9.0	NaN
Penelope	AL	white	Apple	4	80	3.3	NaN
Dean	AK	gray	Cheese	32	180	1.8	99.0
Christina	TX	black	Melon	33	172	9.5	NaN
Cornelia	TX	red	Beans	69	150	2.2	NaN

Exercise 7

Read in the x, y, and z columns from the diamonds dataset. Find the mean of each row and subtract that value from each value in the row.

[10]:
```
diamonds = pd.read_csv('../data/diamonds.csv', usecols=['x', 'y', 'z'])
diamonds.head(3)
```

	x	y	z
0	3.95	3.98	2.43
1	3.89	3.84	2.31
2	4.05	4.07	2.31

[11]:
```
diamonds.sub(diamonds.mean(axis=1), axis=0).head(3)
```

	x	y	z
0	0.496667	0.526667	-1.023333
1	0.543333	0.493333	-1.036667
2	0.573333	0.593333	-1.166667

10.2 2. Concatenating Data

Exercise 1

Read in all of the files in the `../data/weather` directory except `city_attributes.csv` as DataFrames placing the `datetime` column in the index. Store the data in a dictionary using the file name (without extension) as the key. Concatenate the DataFrames vertically labeling each DataFrame in the index appropriately.

[12]:
```
from pathlib import Path
weather_data_path = Path('../data/weather')
paths = sorted(weather_data_path.glob('*.csv'))
dfs = {}
for path in paths:
    weather_type = path.stem
    if weather_type == 'city_attributes':
        continue
    dfs[weather_type] = pd.read_csv(path, parse_dates=['datetime'],␣
↪index_col='datetime')
df_weather = pd.concat(dfs, names=['weather_type'])
df_weather.head()
```

		Seattle	San Francisco	Los Angeles	Las Vegas	Denver	Houston	Chicago	Atlanta	Miami	New York
weather_type	**datetime**										
humidity	**2013-01-01 00:00:00**	75.0	58.0	NaN	42.0	46.0	70.0	NaN	55.0	56.0	NaN
	2013-01-01 01:00:00	80.0	62.0	NaN	42.0	50.0	76.0	64.0	64.0	63.0	54.0
	2013-01-01 02:00:00	86.0	70.0	43.0	52.0	58.0	76.0	69.0	NaN	63.0	54.0
	2013-01-01 03:00:00	NaN	70.0	NaN	52.0	53.0	76.0	NaN	86.0	63.0	54.0
	2013-01-01 04:00:00	NaN	70.0	53.0	52.0	53.0	81.0	68.0	80.0	63.0	54.0

[13]:
```
df_weather.tail()
```

		Seattle	San Francisco	Los Angeles	Las Vegas	Denver	Houston	Chicago	Atlanta	Miami	New York
weather_type	datetime										
wind_speed	2016-12-31 19:00:00	4.0	3.0	3.0	1.0	2.0	2.0	6.0	1.0	6.0	4.0
	2016-12-31 20:00:00	3.0	3.0	1.0	1.0	1.0	3.0	7.0	1.0	6.0	3.0
	2016-12-31 21:00:00	3.0	4.0	4.0	1.0	1.0	3.0	7.0	2.0	5.0	7.0
	2016-12-31 22:00:00	4.0	3.0	1.0	1.0	1.0	1.0	5.0	3.0	5.0	4.0
	2016-12-31 23:00:00	4.0	3.0	1.0	1.0	1.0	1.0	8.0	3.0	5.0	5.0

Exercise 2

Use the dictionary created in Exercise 1 to concatenate the DataFrames horizontally.

[14]:
```
df_weather = pd.concat(dfs, names=['weather_type', 'city'], axis=1)
df_weather.head()
```

weather_type	humidity					...	wind_speed				
city	Seattle	San Francisco	Los Angeles	Las Vegas	Denver	...	Houston	Chicago	Atlanta	Miami	New York
datetime											
2013-01-01 00:00:00	75.0	58.0	NaN	42.0	46.0	...	3.0	4.0	0.0	3.0	13.0
2013-01-01 01:00:00	80.0	62.0	NaN	42.0	50.0	...	3.0	3.0	0.0	4.0	11.0
2013-01-01 02:00:00	86.0	70.0	43.0	52.0	58.0	...	4.0	6.0	0.0	4.0	8.0
2013-01-01 03:00:00	NaN	70.0	NaN	52.0	53.0	...	3.0	7.0	0.0	6.0	9.0
2013-01-01 04:00:00	NaN	70.0	53.0	52.0	53.0	...	2.0	7.0	0.0	5.0	8.0

5 rows × 60 columns

Exercise 3

Write a function that accepts the dictionary created from Exercise 1 and a city name. Return a DataFrame of all of the weather metrics for that city for each day. The index will be the datetime and the columns will be each individual weather metric.

[15]:
```
def get_city_weather(dfs, city):
    s_dict = {}
    for weather_type, df in dfs.items():
        s_dict[weather_type] = df[city]
    return pd.concat(s_dict, axis=1)
```

[16]:
```
get_city_weather(dfs, 'Houston').head()
```

	humidity	pressure	temperature	weather_description	wind_direction	wind_speed
datetime						
2013-01-01 00:00:00	70.0	1026.0	8.81	broken clouds	100.0	3.0
2013-01-01 01:00:00	76.0	1026.0	8.81	broken clouds	90.0	3.0
2013-01-01 02:00:00	76.0	1026.0	8.81	overcast clouds	80.0	4.0
2013-01-01 03:00:00	76.0	1026.0	8.48	broken clouds	90.0	3.0
2013-01-01 04:00:00	81.0	1025.0	8.34	sky is clear	90.0	2.0

[17]:
```
get_city_weather(dfs, 'Denver').head()
```

datetime	humidity	pressure	temperature	weather_description	wind_direction	wind_speed
2013-01-01 00:00:00	46.0	1014.0	-2.30	broken clouds	320.0	7.0
2013-01-01 01:00:00	50.0	1014.0	-3.23	broken clouds	340.0	7.0
2013-01-01 02:00:00	58.0	1015.0	-3.03	broken clouds	350.0	4.0
2013-01-01 03:00:00	53.0	1015.0	-3.67	broken clouds	330.0	3.0
2013-01-01 04:00:00	53.0	1015.0	-5.55	broken clouds	320.0	4.0

Exercise 4

Iterate through all of the DataFrames in the dictionary created in Exercise 1 and use the `assign` method to add a column for the `weather_type`. Save the results in a **list** named `dfs_list`.

[18]:
```
dfs_list = [df.assign(weather_type=wt) for wt, df in dfs.items()]
```

[19]:
```
dfs_list[0].head()
```

datetime	Seattle	San Francisco	Los Angeles	Las Vegas	Denver	...	Chicago	Atlanta	Miami	New York	weather_type
2013-01-01 00:00:00	75.0	58.0	NaN	42.0	46.0	...	NaN	55.0	56.0	NaN	humidity
2013-01-01 01:00:00	80.0	62.0	NaN	42.0	50.0	...	64.0	64.0	63.0	54.0	humidity
2013-01-01 02:00:00	86.0	70.0	43.0	52.0	58.0	...	69.0	NaN	63.0	54.0	humidity
2013-01-01 03:00:00	NaN	70.0	NaN	52.0	53.0	...	NaN	86.0	63.0	54.0	humidity
2013-01-01 04:00:00	NaN	70.0	53.0	52.0	53.0	...	68.0	80.0	63.0	54.0	humidity

5 rows × 11 columns

Exercise 5

Concatenate all of the DataFrames in `dfs_list` vertically using the `append` method. Create a single DataFrame.

[20]:
```
df_all = dfs_list[0]
for df in dfs_list[1:]:
    df_all = df_all.append(df)
```

[21]:
```
df_all.tail()
```

datetime	Seattle	San Francisco	Los Angeles	Las Vegas	Denver	...	Chicago	Atlanta	Miami	New York	weather_type
2016-12-31 19:00:00	4.0	3.0	3.0	1.0	2.0	...	6.0	1.0	6.0	4.0	wind_speed
2016-12-31 20:00:00	3.0	3.0	1.0	1.0	1.0	...	7.0	1.0	6.0	3.0	wind_speed
2016-12-31 21:00:00	3.0	4.0	4.0	1.0	1.0	...	7.0	2.0	5.0	7.0	wind_speed
2016-12-31 22:00:00	4.0	3.0	1.0	1.0	1.0	...	5.0	3.0	5.0	4.0	wind_speed
2016-12-31 23:00:00	4.0	3.0	1.0	1.0	1.0	...	8.0	3.0	5.0	5.0	wind_speed

5 rows × 11 columns

Exercise 6

Select the temperature data from the dictionary created in Exercise 1. Find the mean, median, min, and max temperature per month for the city of Houston. Then create a new column named `mean_median_diff` that contains the absolute difference between the mean and median. Also create the column `min_max_diff` that rounds the absolute difference between the min and max temperatures. Use one line of code.

[22]:
```
(dfs['temperature']
    .resample('M')['Houston'].agg(['mean', 'median', 'min', 'max'])
    .assign(mean_median_diff=lambda df: (df['mean'] - df['median']).abs(),
            min_max_diff=lambda df: (df['min'] - df['max']).abs().round(0))
    .head())
```

datetime	mean	median	min	max	mean_median_diff	min_max_diff
2013-01-31	12.885014	11.930	0.10	25.57	0.955014	25.0
2013-02-28	14.872489	15.060	3.30	25.97	0.187511	23.0
2013-03-31	16.011552	16.625	1.90	31.64	0.613448	30.0
2013-04-30	19.001104	19.725	5.24	28.07	0.723896	23.0
2013-05-31	23.581378	24.145	7.57	32.21	0.563622	25.0

10.3 3. Joining DataFrames

[23]:
```
import pandas as pd
CS = 'sqlite:///../data/databases/chinook.db'
tracks = pd.read_sql('tracks', CS)
genres = pd.read_sql('genres', CS)
albums = pd.read_sql('albums', CS)
artists = pd.read_sql('artists', CS)
media_types = pd.read_sql('media_types', CS)
playlist_track = pd.read_sql('playlist_track', CS)
playlists = pd.read_sql('playlists', CS)
invoice_items = pd.read_sql('invoice_items', CS)
invoices = pd.read_sql('invoices', CS)
customers = pd.read_sql('customers', CS)
employees = pd.read_sql('employees', CS)
```

Exercise 1

Find the occurrences of each media type in the tracks table. Use the name of the media type.

[24]:
```
df = tracks.merge(media_types, how='inner',
                  on='MediaTypeId', suffixes=('_track', '_mt'))
df.head(3)
```

	TrackId	Name_track	AlbumId	MediaTypeId	GenreId	Composer	Milliseconds	Bytes	UnitPrice	Name_mt
0	1	For Those About To Rock (We Salute You)	1	1	1	Angus Young, Malcolm Young, Brian Johnson	343719	11170334	0.99	MPEG audio file
1	6	Put The Finger On You	1	1	1	Angus Young, Malcolm Young, Brian Johnson	205662	6713451	0.99	MPEG audio file
2	7	Let's Get It Up	1	1	1	Angus Young, Malcolm Young, Brian Johnson	233926	7636561	0.99	MPEG audio file

[25]:
```
df['Name_mt'].value_counts()
```

[25]:
```
MPEG audio file                3034
Protected AAC audio file        237
Protected MPEG-4 video file     214
AAC audio file                   11
Purchased AAC audio file          7
Name: Name_mt, dtype: int64
```

Alternatively, find the count of the mediatypeid first.

[26]:
```
mt_count = tracks['MediaTypeId'].value_counts()
mt_count
```

[26]:
```
1    3034
2     237
3     214
5      11
4       7
Name: MediaTypeId, dtype: int64
```

DataFrames can be joined to a Series index.

[27]:
```
media_types.merge(mt_count, left_on='MediaTypeId', right_index=True)
```

	MediaTypeId	MediaTypeId_x	Name	MediaTypeId_y
0	1	1	MPEG audio file	3034
1	2	2	Protected AAC audio file	237
2	3	3	Protected MPEG-4 video file	214
3	4	4	Purchased AAC audio file	7
4	5	5	AAC audio file	11

Exercise 2

Are there any playlists that have no tracks? If so, which ones are they? Use `merge` in your solution.

[28]:
```
playlists.head()
```

	PlaylistId	Name
0	1	Music
1	2	Movies
2	3	TV Shows
3	4	Audiobooks
4	5	90's Music

[29]:
```
playlist_track.head()
```

	PlaylistId	TrackId
0	1	3402
1	1	3389
2	1	3390
3	1	3391
4	1	3392

[30]:
```
df = playlists.merge(playlist_track, how='left', on='PlaylistId', indicator=True)
df.head()
```

	PlaylistId	Name	TrackId	_merge
0	1	Music	3402.0	both
1	1	Music	3389.0	both
2	1	Music	3390.0	both
3	1	Music	3391.0	both
4	1	Music	3392.0	both

Check to see if any are left_only

[31]:
```
df['_merge'].value_counts()
```

[31]:
```
both          8715
left_only        4
right_only       0
Name: _merge, dtype: int64
```

Yes, there are four playlists without any tracks. Let's filter for them.

[32]:
```
df.query('_merge == "left_only"')
```

	PlaylistId	Name	TrackId	_merge
3290	2	Movies	NaN	left_only
3504	4	Audiobooks	NaN	left_only
4982	6	Audiobooks	NaN	left_only
4983	7	Movies	NaN	left_only

Exercise 3

Find the number of tracks per playlist. Use the playlist name in the result. Some playlists have the same name. Make sure not to combine them.

[33]:
```
playlists.head()
```

	PlaylistId	Name
0	1	Music
1	2	Movies
2	3	TV Shows
3	4	Audiobooks
4	5	90's Music

[34]:
```
# there shold be 18 playlists in the result
playlists.shape
```

[34]:
```
(18, 2)
```

[35]:
```
playlist_track.head()
```

	PlaylistId	TrackId
0	1	3402
1	1	3389
2	1	3390
3	1	3391
4	1	3392

[36]:
```
df = playlists.merge(playlist_track, how='left', on='PlaylistId')
df.head()
```

	PlaylistId	Name	TrackId
0	1	Music	3402.0
1	1	Music	3389.0
2	1	Music	3390.0
3	1	Music	3391.0
4	1	Music	3392.0

Some playlists have 0 songs, which is why a left join must be done. Use `count` to count non-missing values.

[37]:
```
df.groupby(['PlaylistId', 'Name'])['TrackId'].count().sort_values(ascending=False)
```

[37]:
```
PlaylistId  Name
1           Music                     3290
8           Music                     3290
5           90's Music                1477
10          TV Shows                   213
3           TV Shows                   213
12          Classical                   75
11          Brazilian Music             39
17          Heavy Metal Classic         26
13          Classical 101 - Deep Cuts   25
14          Classical 101 - Next Steps  25
15          Classical 101 - The Basics  25
16          Grunge                      15
9           Music Videos                 1
18          On-The-Go 1                  1
4           Audiobooks                   0
6           Audiobooks                   0
7           Movies                       0
2           Movies                       0
Name: TrackId, dtype: int64
```

Exercise 4

Find the number of invoices per customer. Show the customer id, first name, and last name and count of invoices.

[38]:
```
(invoices.merge(customers, how='inner', on='CustomerId')
         .groupby(['CustomerId', 'FirstName', 'LastName'])).size().head()
```

[38]:
```
CustomerId  FirstName  LastName
1           Luís       Gonçalves      7
2           Leonie     Köhler         7
3           François   Tremblay       7
4           Bjørn      Hansen         7
5           František  Wichterlová    7
dtype: int64
```

[39]:
```
(customers.merge(invoices, how='inner', on='CustomerId')
          .merge(invoice_items, how='inner', on='InvoiceId')
          .groupby(['CustomerId', 'FirstName', 'LastName']).size().head()
)
```

[39]:
```
CustomerId  FirstName  LastName
1           Luís       Gonçalves      38
2           Leonie     Köhler         38
3           François   Tremblay       38
4           Bjørn      Hansen         38
5           František  Wichterlová    38
dtype: int64
```

Exercise 5

How many customers is each employee responsible for.

[40]:
```
(employees.merge(customers, how='left',
                 left_on='EmployeeId', right_on='SupportRepId')
          .value_counts('EmployeeId'))
```

[40]:
```
EmployeeId
3    21
4    20
5    18
1     1
2     1
6     1
7     1
8     1
dtype: int64
```

Exercise 6

Find all of the tracks with the same name as the album title.

[41]:
```
(tracks.merge(albums, how='inner', on='AlbumId')
       .query('Name == Title'))[['TrackId', 'Name', 'Title']].head(10)
```

	TrackId	Name	Title
10	2	Balls to the Wall	Balls to the Wall
12	4	Restless and Wild	Restless and Wild
16	17	Let There Be Rock	Let There Be Rock
99	100	Out Of Exile	Out Of Exile
148	149	Black Sabbath	Black Sabbath
168	169	Body Count	Body Count
183	184	Chemical Wedding	Chemical Wedding
205	206	Prenda Minha	Prenda Minha
236	237	Minha Historia	Minha Historia
288	275	Da Lama Ao Caos	Da Lama Ao Caos

Exercise 7

Find the top 10 tracks by length of song (Milliseconds).

[42]:
```
# don't need to join
tracks.nlargest(10, 'Milliseconds')
```

	TrackId	Name	AlbumId	MediaTypeId	GenreId	Composer	Milliseconds	Bytes	UnitPrice
2819	2820	Occupation / Precipice	227	3	19	None	5286953	1054423946	1.99
3223	3224	Through a Looking Glass	229	3	21	None	5088838	1059546140	1.99
3243	3244	Greetings from Earth, Pt. 1	253	3	20	None	2960293	536824558	1.99
3241	3242	The Man With Nine Lives	253	3	20	None	2956998	577829804	1.99
3226	3227	Battlestar Galactica, Pt. 2	253	3	20	None	2956081	521387924	1.99
3225	3226	Battlestar Galactica, Pt. 1	253	3	20	None	2952702	541359437	1.99
3242	3243	Murder On the Rising Star	253	3	20	None	2935894	551759986	1.99
3227	3228	Battlestar Galactica, Pt. 3	253	3	20	None	2927802	554509033	1.99
3247	3248	Take the Celestra	253	3	20	None	2927677	512381289	1.99
3238	3239	Fire In Space	253	3	20	None	2926593	536784757	1.99

Exercise 8

Are there any genres that do not appear in the tracks table? If so, which ones are they? Use `merge` in your solution.

[43]:
```
df = genres.merge(tracks, how='left', on='GenreId', indicator=True)
df.head()
```

	GenreId	Name_x	TrackId	Name_y	AlbumId	...	Composer	Milliseconds	Bytes	UnitPrice	_merge
0	1	Rock	1	For Those About To Rock (We Salute You)	1	...	Angus Young, Malcolm Young, Brian Johnson	343719	11170334	0.99	both
1	1	Rock	2	Balls to the Wall	2	...	None	342562	5510424	0.99	both
2	1	Rock	3	Fast As a Shark	3	...	F. Baltes, S. Kaufman, U. Dirkscneider & W. Ho...	230619	3990994	0.99	both
3	1	Rock	4	Restless and Wild	3	...	F. Baltes, R.A. Smith-Diesel, S. Kaufman, U. D...	252051	4331779	0.99	both
4	1	Rock	5	Princess of the Dawn	3	...	Deaffy & R.A. Smith-Diesel	375418	6290521	0.99	both

5 rows × 11 columns

Count `_merge` column - No, all genres appear in tracks table.

[44]:
```
df['_merge'].value_counts()
```

[44]:
```
both          3503
left_only        0
right_only       0
Name: _merge, dtype: int64
```

Exercise 9

Count the number of albums per artist. Make sure to include artists that do not have any albums.

[45]:
```
df = artists.merge(albums, on='ArtistId', how='left')
df.head()
```

	ArtistId	Name	AlbumId	Title
0	1	AC/DC	1.0	For Those About To Rock We Salute You
1	1	AC/DC	4.0	Let There Be Rock
2	2	Accept	2.0	Balls to the Wall
3	2	Accept	3.0	Restless and Wild
4	3	Aerosmith	5.0	Big Ones

[46]:
```
# note that some artists have 0 albums
df.groupby('Name')['AlbumId'].count().head(10)
```

[46]:
```
Name
A Cor Do Som
0
AC/DC
2
Aaron Copland & London Symphony Orchestra
1
Aaron Goldberg
1
Academy of St. Martin in the Fields & Sir Neville Marriner
1
Academy of St. Martin in the Fields Chamber Ensemble & Sir Neville Marriner
1
Academy of St. Martin in the Fields, John Birch, Sir Neville Marriner & Sylvia McNair
1
Academy of St. Martin in the Fields, Sir Neville Marriner & Thurston Dart
1
Academy of St. Martin in the Fields, Sir Neville Marriner & William Bennett
0
Accept
2
Name: AlbumId, dtype: int64
```

Exercise 10

Find the cost of each playlist. Include playlists with zero tracks.

[47]:
```
t = tracks[['TrackId', 'UnitPrice']]
df = (playlists.merge(playlist_track, how='left', on='PlaylistId')
```

```
                .merge(t, how='left', on='TrackId'))
df.head()
```

	PlaylistId	Name	TrackId	UnitPrice
0	1	Music	3402.0	0.99
1	1	Music	3389.0	0.99
2	1	Music	3390.0	0.99
3	1	Music	3391.0	0.99
4	1	Music	3392.0	0.99

[48]:
```
df.groupby(['PlaylistId', 'Name'])['UnitPrice'].agg(['count', 'sum'])
```

		count	sum
PlaylistId	**Name**		
1	**Music**	3290	3257.10
2	**Movies**	0	0.00
3	**TV Shows**	213	423.87
4	**Audiobooks**	0	0.00
5	**90's Music**	1477	1462.23
6	**Audiobooks**	0	0.00
7	**Movies**	0	0.00
8	**Music**	3290	3257.10
9	**Music Videos**	1	0.99
10	**TV Shows**	213	423.87
11	**Brazilian Music**	39	38.61
12	**Classical**	75	74.25
13	**Classical 101 - Deep Cuts**	25	24.75
14	**Classical 101 - Next Steps**	25	24.75
15	**Classical 101 - The Basics**	25	24.75
16	**Grunge**	15	14.85
17	**Heavy Metal Classic**	26	25.74
18	**On-The-Go 1**	1	0.99

Exercise 11

Count the total number of times each track was sold and return the top 10 tracks.

[49]:
```
(invoice_items.merge(tracks, how='inner', on='TrackId')
              .groupby('Name')['Quantity'].sum()
              .nlargest(10))
```

[49]:
```
Name
The Trooper                5
Eruption                   4
Hallowed Be Thy Name       4
Sure Know Something        4
The Number Of The Beast    4
Untitled                   4
```

```
2 Minutes To Midnight      3
Blood Brothers             3
Brasil                     3
Can I Play With Madness    3
Name: Quantity, dtype: int64
```

Exercise 12

Create a pivot table with billing country and genre as the index and columns and the number of tracks sold as the values.

[50]:
```
# create simpler tables first with minimum number of columns
i = invoices[['InvoiceId', 'BillingCountry']]
ii = invoice_items[['InvoiceId', 'TrackId']]
t = tracks[['TrackId', 'GenreId']]
df = (i.merge(ii, how='inner', on='InvoiceId')
       .merge(t, how='inner', on='TrackId')
       .merge(genres, how='inner', on='GenreId'))
df.head()
```

	InvoiceId	BillingCountry	TrackId	GenreId	Name
0	1	Germany	2	1	Rock
1	214	Canada	2	1	Rock
2	1	Germany	4	1	Rock
3	2	Norway	6	1	Rock
4	2	Norway	8	1	Rock

[51]:
```
pd.crosstab(index=df['BillingCountry'], columns=df['Name']).head()
```

Name	Alternative	Alternative & Punk	Blues	Bossa Nova	Classical	...	Sci Fi & Fantasy	Science Fiction	Soundtrack	TV Shows	World
BillingCountry											
Argentina	0	9	0	0	0	...	0	0	1	0	0
Australia	0	0	1	0	0	...	0	0	0	0	0
Austria	0	0	0	0	2	...	0	0	0	4	0
Belgium	0	14	0	0	0	...	0	0	0	0	0
Brazil	0	7	6	0	6	...	2	0	4	0	2

5 rows × 24 columns

Exercise 13

Find the name and email of each employee's boss. Make use of the suffix arguments to better label the merged data. Be sure to include employees that don't have bosses. This is called a recursive relationship.

[52]:
```
e = employees[['EmployeeId', 'FirstName', 'LastName', 'ReportsTo']]
boss = employees[['EmployeeId', 'FirstName', 'LastName', 'Email']]
e.merge(boss, how='left', left_on='ReportsTo',
        right_on='EmployeeId', suffixes=('_employee', '_boss'))
```

	EmployeeId_employee	FirstName_employee	LastName_employee	ReportsTo	EmployeeId_boss	FirstName_boss	LastName_boss	Email
0	1	Andrew	Adams	NaN	NaN	NaN	NaN	NaN
1	2	Nancy	Edwards	1.0	1.0	Andrew	Adams	andrew@chinookcorp.com
2	3	Jane	Peacock	2.0	2.0	Nancy	Edwards	nancy@chinookcorp.com
3	4	Margaret	Park	2.0	2.0	Nancy	Edwards	nancy@chinookcorp.com
4	5	Steve	Johnson	2.0	2.0	Nancy	Edwards	nancy@chinookcorp.com
5	6	Michael	Mitchell	1.0	1.0	Andrew	Adams	andrew@chinookcorp.com
6	7	Robert	King	6.0	6.0	Michael	Mitchell	michael@chinookcorp.com
7	8	Laura	Callahan	6.0	6.0	Michael	Mitchell	michael@chinookcorp.com

Exercise 14

Find the average length of tracks for each artist for those with at least 10 tracks. Return five artists with the longest average track length.

[53]:
```
a = artists.rename(columns={'Name': 'ArtistName'})
t = tracks[['AlbumId', 'Milliseconds']]
df = (a.merge(albums, how='inner', on='ArtistId')
       .merge(t, how='inner', on='AlbumId'))
df.head()
```

	ArtistId	ArtistName	AlbumId	Title	Milliseconds
0	1	AC/DC	1	For Those About To Rock We Salute You	343719
1	1	AC/DC	1	For Those About To Rock We Salute You	205662
2	1	AC/DC	1	For Those About To Rock We Salute You	233926
3	1	AC/DC	1	For Those About To Rock We Salute You	210834
4	1	AC/DC	1	For Those About To Rock We Salute You	203102

[54]:
```
df2 = (df.groupby('ArtistName')['Milliseconds'].agg(['count', 'mean'])
         .query('count >= 10')
         .sort_values('mean', ascending=False))
df2['mean'] = df2['mean'] / (60 * 1000)
df2.head()
```

	count	mean
ArtistName		
Battlestar Galactica (Classic)	24	48.759572
Battlestar Galactica	20	46.174409
Heroes	23	43.319035
Lost	92	43.166410
The Office	53	23.562410

Part XI

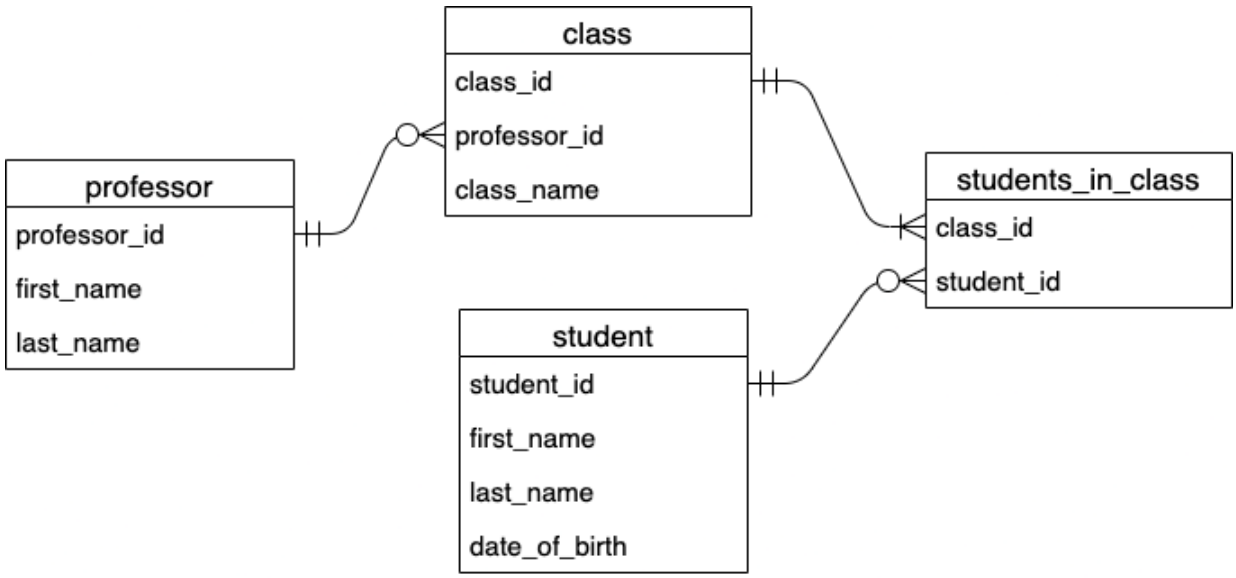
Fundamentals of SQL

Chapter 11

Solutions

1. Intro to Databases and SQL
2. The SELECT Statement
3. GROUP BY and JOIN Clauses

11.1 1. Intro to Databases and SQL



Exercise 1

In words, describe the relationship between the professor and class tables.

Answer - Each class is taught by one and only one professor. Some professors may not be present in the class table. Other professors may appear one or more times in the class table.

Exercise 2

In words, describe the relationship between the class and students_in_class tables.

Answer - Each class has at least one student_id in it. Each class in the students_in_class table is mapped to exactly one row in the class table.

Exercise 3

What is the minimum and maximum number of professors each student can have?

Answer - Since each student is not guaranteed to appear in the students_in_class table, the minimum is 0. There is no maximum as students may appear any number of times in the students_in_class table and all classes are guaranteed to have exactly one professor.

11.2 2. The SELECT Statement

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

Exercise 1

Create a variable called `CS_CHINOOK` and assign it the value of the connection string. Use it to read in all of the columns of the tracks table.

[2]:
```
CS_CHINOOK = 'sqlite:///../data/databases/chinook.db'
sql = """
SELECT *
FROM tracks
"""
pd.read_sql(sql, CS_CHINOOK).head()
```

	TrackId	Name	AlbumId	MediaTypeId	GenreId	Composer	Milliseconds	Bytes	UnitPrice
0	1	For Those About To Rock (We Salute You)	1	1	1	Angus Young, Malcolm Young, Brian Johnson	343719	11170334	0.99
1	2	Balls to the Wall	2	2	1	None	342562	5510424	0.99
2	3	Fast As a Shark	3	2	1	F. Baltes, S. Kaufman, U. Dirkscneider & W. Ho...	230619	3990994	0.99
3	4	Restless and Wild	3	2	1	F. Baltes, R.A. Smith-Diesel, S. Kaufman, U. D...	252051	4331779	0.99
4	5	Princess of the Dawn	3	2	1	Deaffy & R.A. Smith-Diesel	375418	6290521	0.99

Exercise 2

Select the name and composer columns from the tracks table, returning the first five records.

[3]:
```
sql = """
SELECT name, composer
FROM tracks
LIMIT 5
"""
pd.read_sql(sql, CS_CHINOOK)
```

	Name	Composer
0	For Those About To Rock (We Salute You)	Angus Young, Malcolm Young, Brian Johnson
1	Balls to the Wall	None
2	Fast As a Shark	F. Baltes, S. Kaufman, U. Dirkscneider & W. Ho...
3	Restless and Wild	F. Baltes, R.A. Smith-Diesel, S. Kaufman, U. D...
4	Princess of the Dawn	Deaffy & R.A. Smith-Diesel

Exercise 3

Find the number of unique composers and unit prices in the tracks table.

[4]:
```
sql = """
SELECT
    count(distinct composer) as num_unique_composer,
    count(distinct UnitPrice) as num_unique_unitprice
FROM tracks
LIMIT 5
"""
pd.read_sql(sql, CS_CHINOOK)
```

	num_unique_composer	num_unique_unitprice
0	852	2

Exercise 4

Find the unique unit prices in the tracks table.

[5]:
```
sql = """
SELECT DISTINCT unitprice
FROM tracks
"""
pd.read_sql(sql, CS_CHINOOK)
```

	UnitPrice
0	0.99
1	1.99

Exercise 5

Count the total number of records and the number of non-missing values of composer in the tracks table.

[6]:
```
sql = """
SELECT count(*), count(composer)
FROM tracks
"""
pd.read_sql(sql, CS_CHINOOK)
```

	count(*)	count(composer)
0	3503	2525

Exercise 6

Return the first five records in the tracks table where composer is missing.

[7]:
```
sql = """
SELECT *
FROM tracks
WHERE composer IS NULL
LIMIT 5
"""
pd.read_sql(sql, CS_CHINOOK)
```

	TrackId	Name	AlbumId	MediaTypeId	GenreId	Composer	Milliseconds	Bytes	UnitPrice
0	2	Balls to the Wall	2	2	1	None	342562	5510424	0.99
1	63	Desafinado	8	1	2	None	185338	5990473	0.99
2	64	Garota De Ipanema	8	1	2	None	285048	9348428	0.99
3	65	Samba De Uma Nota Só (One Note Samba)	8	1	2	None	137273	4535401	0.99
4	66	Por Causa De Você	8	1	2	None	169900	5536496	0.99

Exercise 7

Filter the tracks table where unit price is 1.99. Return the first five after the 100th.

[8]:
```
sql = """
SELECT *
FROM tracks
WHERE unitprice = 1.99
LIMIT 5 OFFSET 100
"""
pd.read_sql(sql, CS_CHINOOK)
```

	TrackId	Name	AlbumId	MediaTypeId	GenreId	Composer	Milliseconds	Bytes	UnitPrice
0	2919	Born to Run	230	3	19	None	2618619	213772057	1.99
1	2920	Three Minutes	231	3	19	None	2763666	531556853	1.99
2	2921	Exodus (Part 1)	230	3	19	None	2620747	213107744	1.99
3	2922	Live Together, Die Alone, Pt. 1	231	3	21	None	2478041	457364940	1.99
4	2923	Exodus (Part 2) [Season Finale]	230	3	19	None	2605557	208667059	1.99

Exercise 8

Compute the minutes and seconds of each song in the tracks table as separate columns. Return the song name and milliseconds along with the other two columns naming them appropriately.

[9]:
```
sql = """
SELECT name,
       milliseconds,
       Milliseconds / 1000 / 60 AS minutes,
       Milliseconds / 1000 % 60 AS seconds
FROM tracks
"""
pd.read_sql(sql, CS_CHINOOK).head(10)
```

	Name	Milliseconds	minutes	seconds
0	For Those About To Rock (We Salute You)	343719	5	43
1	Balls to the Wall	342562	5	42
2	Fast As a Shark	230619	3	50
3	Restless and Wild	252051	4	12
4	Princess of the Dawn	375418	6	15
5	Put The Finger On You	205662	3	25
6	Let's Get It Up	233926	3	53
7	Inject The Venom	210834	3	30
8	Snowballed	203102	3	23
9	Evil Walks	263497	4	23

Exercise 9

Select all the records between three and four minutes in length from the tracks table.

[10]:
```
sql = """
SELECT *
FROM tracks
WHERE milliseconds between 3 * 60 * 1000 and 4 * 60 * 1000
"""
pd.read_sql(sql, CS_CHINOOK).head()
```

	TrackId	Name	AlbumId	MediaTypeId	GenreId	Composer	Milliseconds	Bytes	UnitPrice
0	3	Fast As a Shark	3	2	1	F. Baltes, S. Kaufman, U. Dirkscneider & W. Ho...	230619	3990994	0.99
1	6	Put The Finger On You	1	1	1	Angus Young, Malcolm Young, Brian Johnson	205662	6713451	0.99
2	7	Let's Get It Up	1	1	1	Angus Young, Malcolm Young, Brian Johnson	233926	7636561	0.99
3	8	Inject The Venom	1	1	1	Angus Young, Malcolm Young, Brian Johnson	210834	6852860	0.99
4	9	Snowballed	1	1	1	Angus Young, Malcolm Young, Brian Johnson	203102	6599424	0.99

Exercise 10

How many records from the tracks table are under 30 seconds in length?

[11]:
```
sql = """
SELECT count(*)
FROM tracks
WHERE milliseconds < 30000
"""
pd.read_sql(sql, CS_CHINOOK)
```

	count(*)
0	8

Exercise 11

How many records from the tracks table are under 30 seconds or more than 10 minutes in length?

[12]:
```
sql = """
SELECT count(*)
FROM tracks
WHERE milliseconds < 30000 or milliseconds > 600000
```

```
"""
pd.read_sql(sql, CS_CHINOOK).head(10)
```

	count(*)
0	268

Exercise 12

Calculate the average unit price for songs greater than 10 minutes in length from the tracks table.

[13]:
```
sql = """
SELECT avg(unitprice)
FROM tracks
WHERE milliseconds > 600000
"""
pd.read_sql(sql, CS_CHINOOK).head(10)
```

	avg(unitprice)
0	1.801538

Exercise 13

Select tracks with TrackId of 10, 100, or 1000.

[14]:
```
sql = """
SELECT *
FROM tracks
WHERE trackid in (10, 100, 1000)
"""
pd.read_sql(sql, CS_CHINOOK)
```

	TrackId	Name	AlbumId	MediaTypeId	GenreId	Composer	Milliseconds	Bytes	UnitPrice
0	10	Evil Walks	1	1	1	Angus Young, Malcolm Young, Brian Johnson	263497	8611245	0.99
1	100	Out Of Exile	11	1	4	Cornell, Commerford, Morello, Wilk	291291	9506571	0.99
2	1000	What If I Do?	80	1	1	Dave Grohl, Taylor Hawkins, Nate Mendel, Chris...	302994	9929799	0.99

Exercise 14

Select all customers from France and Portugal.

[15]:
```
sql = """
SELECT *
FROM customers
WHERE country in ("France", "Portugal")
"""
pd.read_sql(sql, CS_CHINOOK)
```

	CustomerId	FirstName	LastName	Company	Address	...	PostalCode	Phone	Fax	Email	SupportRepId
0	34	João	Fernandes	None	Rua da Assunção 53	...	None	+351 (213) 466-111	None	jfernandes@yahoo.pt	4
1	35	Madalena	Sampaio	None	Rua dos Campeões Europeus de Viena, 4350	...	None	+351 (225) 022-448	None	masampaio@sapo.pt	4
2	39	Camille	Bernard	None	4, Rue Milton	...	75009	+33 01 49 70 65 65	None	camille.bernard@yahoo.fr	4
3	40	Dominique	Lefebvre	None	8, Rue Hanovre	...	75002	+33 01 47 42 71 71	None	dominiquelefebvre@gmail.com	4
4	41	Marc	Dubois	None	11, Place Bellecour	...	69002	+33 04 78 30 30 30	None	marc.dubois@hotmail.com	5
5	42	Wyatt	Girard	None	9, Place Louis Barthou	...	33000	+33 05 56 96 96 96	None	wyatt.girard@yahoo.fr	3
6	43	Isabelle	Mercier	None	68, Rue Jouvence	...	21000	+33 03 80 73 66 99	None	isabelle_mercier@apple.fr	3

7 rows × 13 columns

Exercise 15

Find the top 10 invoices by total.

[16]:
```
sql = """
SELECT *
FROM invoices
ORDER BY total DESC
LIMIT 10
"""
pd.read_sql(sql, CS_CHINOOK)
```

	InvoiceId	CustomerId	InvoiceDate	BillingAddress	BillingCity	BillingState	BillingCountry	BillingPostalCode	Total
0	404	6	2013-11-13 00:00:00	Rilská 3174/6	Prague	None	Czech Republic	14300	25.86
1	299	26	2012-08-05 00:00:00	2211 W Berry Street	Fort Worth	TX	USA	76110	23.86
2	96	45	2010-02-18 00:00:00	Erzsébet krt. 58.	Budapest	None	Hungary	H-1073	21.86
3	194	46	2011-04-28 00:00:00	3 Chatham Street	Dublin	Dublin	Ireland	None	21.86
4	89	7	2010-01-18 00:00:00	Rotenturmstraße 4, 1010 Innere Stadt	Vienne	None	Austria	1010	18.86
5	201	25	2011-05-29 00:00:00	319 N. Frances Street	Madison	WI	USA	53703	18.86
6	88	57	2010-01-13 00:00:00	Calle Lira, 198	Santiago	None	Chile	None	17.91
7	306	5	2012-09-05 00:00:00	Klanova 9/506	Prague	None	Czech Republic	14700	16.86
8	313	43	2012-10-06 00:00:00	68, Rue Jouvence	Dijon	None	France	21000	16.86
9	103	24	2010-03-21 00:00:00	162 E Superior Street	Chicago	IL	USA	60611	15.86

Exercise 16

Sort the invoices table by BillingCountry and within that by Total from greatest to least.

[17]:
```
sql = """
SELECT *
FROM invoices
ORDER BY billingcountry, total DESC
"""
pd.read_sql(sql, CS_CHINOOK).head()
```

	InvoiceId	CustomerId	InvoiceDate	BillingAddress	BillingCity	BillingState	BillingCountry	BillingPostalCode	Total
0	348	56	2013-03-10 00:00:00	307 Macacha Güemes	Buenos Aires	None	Argentina	1106	13.86
1	403	56	2013-11-08 00:00:00	307 Macacha Güemes	Buenos Aires	None	Argentina	1106	8.91
2	164	56	2010-12-17 00:00:00	307 Macacha Güemes	Buenos Aires	None	Argentina	1106	5.94
3	142	56	2010-09-14 00:00:00	307 Macacha Güemes	Buenos Aires	None	Argentina	1106	3.96
4	119	56	2010-06-12 00:00:00	307 Macacha Güemes	Buenos Aires	None	Argentina	1106	1.98

Exercise 17

Find all tracks that have a name beginning or ending in 'X'.

[18]:
```
sql = """
SELECT *
FROM tracks
WHERE name like 'X%' or name like '%X'
"""
pd.read_sql(sql, CS_CHINOOK).head(20)
```

	TrackId	Name	AlbumId	MediaTypeId	GenreId	Composer	Milliseconds	Bytes	UnitPrice
0	52	Man In The Box	7	1	1	Jerry Cantrell, Layne Staley	286641	9310272	0.99
1	159	FX	17	1	3	Tony Iommi, Bill Ward, Geezer Butler, Ozzy Osb...	103157	3331776	0.99
2	361	X-9 2001	32	1	10	None	273920	9310370	0.99
3	977	Xote Dos Milagres	78	1	7	None	269557	8897778	0.99
4	1593	Bonzo's Montreux	128	1	1	John Bonham	258925	8557447	0.99
5	1996	Heart-Shaped Box	163	1	1	Kurt Cobain	281887	9210982	0.99
6	2410	Xanadu	196	1	1	Geddy Lee And Alex Lifeson/Geddy Lee And Neil ...	667428	21753168	0.99
7	2416	The Temples Of Syrinx	196	1	1	Geddy Lee And Alex Lifeson/Geddy Lee And Neil ...	133459	4360163	0.99
8	2642	Twentienth Century Fox	214	1	1	Robby Krieger, Ray Manzarek, John Densmore, Ji...	153913	5069211	0.99
9	2748	Squeeze Box	221	1	1	Pete Townshend	161280	5256508	0.99
10	2850	The Fix	228	3	21	None	2600266	507026323	1.99
11	3473	Take the Box	322	2	9	Luke Smith	199160	3281526	0.99
12	3487	3 Gymnopédies: No.1 - Lent Et Grave, No.3 - Le...	332	2	24	Erik Satie	385506	6458501	0.99

Exercise 18

Find all tracks that have the word 'smith' anywhere in the composer.

[19]:
```
sql = """
SELECT *
FROM tracks
WHERE composer like '%smith%'
"""
pd.read_sql(sql, CS_CHINOOK).head(10)
```

	TrackId	Name	AlbumId	MediaTypeId	GenreId	Composer	Milliseconds	Bytes	UnitPrice
0	4	Restless and Wild	3	2	1	F. Baltes, R.A. Smith-Diesel, S. Kaufman, U. D...	252051	4331779	0.99
1	5	Princess of the Dawn	3	2	1	Deaffy & R.A. Smith-Diesel	375418	6290521	0.99
2	186	Killing Floor	19	1	3	Adrian Smith	269557	8854240	0.99
3	191	Machine Men	19	1	3	Adrian Smith	341655	11138147	0.99
4	1221	2 Minutes To Midnight	95	1	3	Adrian Smith/Bruce Dickinson	337423	5400576	0.99
5	1226	Can I Play With Madness	96	1	3	Adrian Smith/Bruce Dickinson/Steve Harris	282488	4521984	0.99
6	1229	The Evil That Men Do	96	1	3	Adrian Smith/Bruce Dickinson/Steve Harris	325929	5216256	0.99
7	1235	The Wicker Man	97	1	1	Adrian Smith/Bruce Dickinson/Steve Harris	275539	11022464	0.99
8	1241	The Fallen Angel	97	1	1	Adrian Smith/Steve Harris	240718	9629824	0.99
9	1245	Wildest Dreams	98	1	13	Adrian Smith/Steve Harris	232777	9312384	0.99

Exercise 19

Calculate the average bytes per millisecond for all tracks. Make sure to use true division. Round to one decimal place.

[20]:
```
sql = """
SELECT round(avg(bytes / cast(milliseconds as float)), 1) AS avg_bytes_per_ms
FROM tracks
"""
pd.read_sql(sql, CS_CHINOOK).head(10)
```

	avg_bytes_per_ms
0	40.5

Exercise 20

Return the five longest names in the track table.

[21]:
```
sql = """
SELECT name
FROM tracks
ORDER BY length(name) DESC
LIMIT 5
"""
pd.read_sql(sql, CS_CHINOOK)
```

	Name
0	Homecoming / The Death Of St. Jimmy / East 12t...
1	Symphony No. 3 Op. 36 for Orchestra and Sopran...
2	Jesus Of Suburbia / City Of The Damned / I Don...
3	The Nutcracker, Op. 71a, Act II: Scene 14: Pas...
4	Blind Curve: Vocal Under A Bloodlight / Passin...

Exercise 21

Use the SQLite math functions page to calculate the area of a circle with radius of 17.

[22]:
```
sql = """
SELECT pi() * pow(17, 2)
"""
pd.read_sql(sql, CS_CHINOOK)
```

	pi() * pow(17, 2)
0	907.920277

Exercise 22

Count the number of customers that do not have a company name.

[23]:
```
sql = """
SELECT count(*)
FROM customers
WHERE company IS NULL
"""
pd.read_sql(sql, CS_CHINOOK)
```

	count(*)
0	49

11.3 3. GROUP BY and JOIN Clauses

[24]:
```
import pandas as pd
CS_CHINOOK = 'sqlite:///../data/databases/chinook.db'
tracks = pd.read_sql('tracks', CS_CHINOOK)
artists = pd.read_sql('artists', CS_CHINOOK)
genres = pd.read_sql('genres', CS_CHINOOK)
invoices = pd.read_sql('invoices', CS_CHINOOK)
invoice_items = pd.read_sql('invoice_items', CS_CHINOOK)
customers = pd.read_sql('customers', CS_CHINOOK)
employees = pd.read_sql('employees', CS_CHINOOK)
```

[25]:
```
artists.head()
```

	ArtistId	Name
0	1	AC/DC
1	2	Accept
2	3	Aerosmith
3	4	Alanis Morissette
4	5	Alice In Chains

[26]:
```
invoices.head(3)
```

	InvoiceId	CustomerId	InvoiceDate	BillingAddress	BillingCity	BillingState	BillingCountry	BillingPostalCode	Total
0	1	2	2009-01-01	Theodor-Heuss-Straße 34	Stuttgart	None	Germany	70174	1.98
1	2	4	2009-01-02	Ullevålsveien 14	Oslo	None	Norway	0171	3.96
2	3	8	2009-01-03	Grétrystraat 63	Brussels	None	Belgium	1000	5.94

[27]:
```
invoice_items.head(3)
```

	InvoiceLineId	InvoiceId	TrackId	UnitPrice	Quantity
0	1	1	2	0.99	1
1	2	1	4	0.99	1
2	3	2	6	0.99	1

[28]:
```
tracks.head(3)
```

	TrackId	Name	AlbumId	MediaTypeId	GenreId	Composer	Milliseconds	Bytes	UnitPrice
0	1	For Those About To Rock (We Salute You)	1	1	1	Angus Young, Malcolm Young, Brian Johnson	343719	11170334	0.99
1	2	Balls to the Wall	2	2	1	None	342562	5510424	0.99
2	3	Fast As a Shark	3	2	1	F. Baltes, S. Kaufman, U. Dirkscneider & W. Ho...	230619	3990994	0.99

[29]:
```
customers.head(3)
```

	CustomerId	FirstName	LastName	Company	Address	...	PostalCode	Phone	Fax	Email	SupportRepId
0	1	Luís	Gonçalves	Embraer - Empresa Brasileira de Aeronáutica S.A.	Av. Brigadeiro Faria Lima, 2170	...	12227-000	+55 (12) 3923-5555	+55 (12) 3923-5566	luisg@embraer.com.br	3
1	2	Leonie	Köhler	None	Theodor-Heuss-Straße 34	...	70174	+49 0711 2842222	None	leonekohler@surfeu.de	5
2	3	François	Tremblay	None	1498 rue Bélanger	...	H2G 1A7	+1 (514) 721-4711	None	ftremblay@gmail.com	3

3 rows × 13 columns

Exercise 1

Find the grand total of invoices by billingcountry. Order the results by this total from greatest to least.

[30]:
```
sql = """
SELECT billingcountry, sum(total) as total
FROM invoices
GROUP BY billingcountry
ORDER BY total DESC
"""
pd.read_sql(sql, CS_CHINOOK).head(10)
```

	BillingCountry	total
0	USA	523.06
1	Canada	303.96
2	France	195.10
3	Brazil	190.10
4	Germany	156.48
5	United Kingdom	112.86
6	Czech Republic	90.24
7	Portugal	77.24
8	India	75.26
9	Chile	46.62

Exercise 2

Find the count, min, max, and avg milliseconds by genreid in the tracks table. Convert the milliseconds to minutes. Order the results by count descending.

[31]:
```
# note how 1000.0 is used so that the first division is true division
sql = """
SELECT genreid,
       count(milliseconds) as count,
       round(min(milliseconds / 1000.0 / 60), 2) as min_time,
       round(max(milliseconds / 1000.0 / 60), 2) as max_time,
       round(avg(milliseconds / 1000.0 / 60), 2) as avg_time
FROM tracks
GROUP BY genreid
ORDER by count DESC
"""
pd.read_sql(sql, CS_CHINOOK).head(10)
```

	GenreId	count	min_time	max_time	avg_time
0	1	1297	0.02	26.87	4.73
1	7	579	0.55	9.05	3.88
2	3	374	0.70	13.61	5.16
3	4	332	0.08	9.31	3.91
4	2	130	2.11	15.13	4.86
5	19	93	20.63	88.12	35.75
6	6	81	2.25	9.83	4.51
7	24	74	0.86	9.94	4.90
8	21	64	1.88	84.81	42.92
9	14	61	2.12	6.97	3.67

Exercise 3

Using the invoice_items table, count the times each trackid and invoiceid combination appear. This number should be one, as an invoice should not have multiple instances of the same trackid. Can you verify that there is in fact at most one?

[32]:
```
# A simple group by to get the count by trackid and invoiceid
sql = """
SELECT trackid, invoiceid, count(*) as ct
FROM invoice_items
GROUP BY trackid, invoiceid
"""
pd.read_sql(sql, CS_CHINOOK).head()
```

	TrackId	InvoiceId	ct
0	1	108	1
1	2	1	1
2	2	214	1
3	3	319	1
4	4	1	1

[33]:
```
# use having clause to see if any counts are greater than 1
sql = """
SELECT trackid, invoiceid, count(*) as ct
FROM invoice_items
GROUP BY trackid, invoiceid
HAVING ct > 1
"""
pd.read_sql(sql, CS_CHINOOK).head()
```

	TrackId	InvoiceId	ct

Exercise 4

Calculate the total revenue of each track using the invoice_items table. Return the top five tracks by revenue.

[34]:
```
# use having clause to see if any counts are greater than 1
sql = """
SELECT trackid, sum(unitprice) as revenue
FROM invoice_items
GROUP BY trackid
ORDER BY revenue DESC
LIMIT 5
"""
pd.read_sql(sql, CS_CHINOOK)
```

	TrackId	revenue
0	2832	3.98
1	2850	3.98
2	2868	3.98
3	3177	3.98
4	3200	3.98

Exercise 5

Create a table with the track name and genre name (not genreid).

[35]:
```
sql = """
SELECT
    t.name as track_name,
    g.name as genre_name
FROM tracks as t
    LEFT JOIN genres as g ON t.genreid = g.genreid
"""
pd.read_sql(sql, CS_CHINOOK).head()
```

	track_name	genre_name
0	For Those About To Rock (We Salute You)	Rock
1	Balls to the Wall	Rock
2	Fast As a Shark	Rock
3	Restless and Wild	Rock
4	Princess of the Dawn	Rock

Exercise 6

Create a table with the track name, genre name, album title, and artist title. You will need to join four tables together.

[36]:
```
sql = """
SELECT
    t.name as track_name,
    g.name as genre_name,
    a.title as album_title,
    ar.name as artist_title
FROM tracks as t
    LEFT JOIN genres as g ON t.genreid = g.genreid
    LEFT JOIN albums as a ON t.albumid = a.albumid
    LEFT JOIN artists as ar ON a.artistid = ar.artistid
"""
pd.read_sql(sql, CS_CHINOOK).head()
```

	track_name	genre_name	album_title	artist_title
0	For Those About To Rock (We Salute You)	Rock	For Those About To Rock We Salute You	AC/DC
1	Balls to the Wall	Rock	Balls to the Wall	Accept
2	Fast As a Shark	Rock	Restless and Wild	Accept
3	Restless and Wild	Rock	Restless and Wild	Accept
4	Princess of the Dawn	Rock	Restless and Wild	Accept

Exercise 7

For tracks less than two minutes in length, count the occurrence of each media type. Make sure to use the media type name.

[37]:
```
sql = """
SELECT
    mt.name, count(*)
FROM tracks as t
    LEFT JOIN media_types as mt ON t.mediatypeid = mt.mediatypeid
WHERE t.milliseconds < 2 * 60 * 1000
GROUP BY mt.mediatypeid

"""
pd.read_sql(sql, CS_CHINOOK)
```

	Name	count(*)
0	MPEG audio file	85
1	Protected AAC audio file	5
2	Protected MPEG-4 video file	1
3	Purchased AAC audio file	2

[38]:
```

```

Part XII

Visualization with Matplotlib

Chapter 12

Solutions

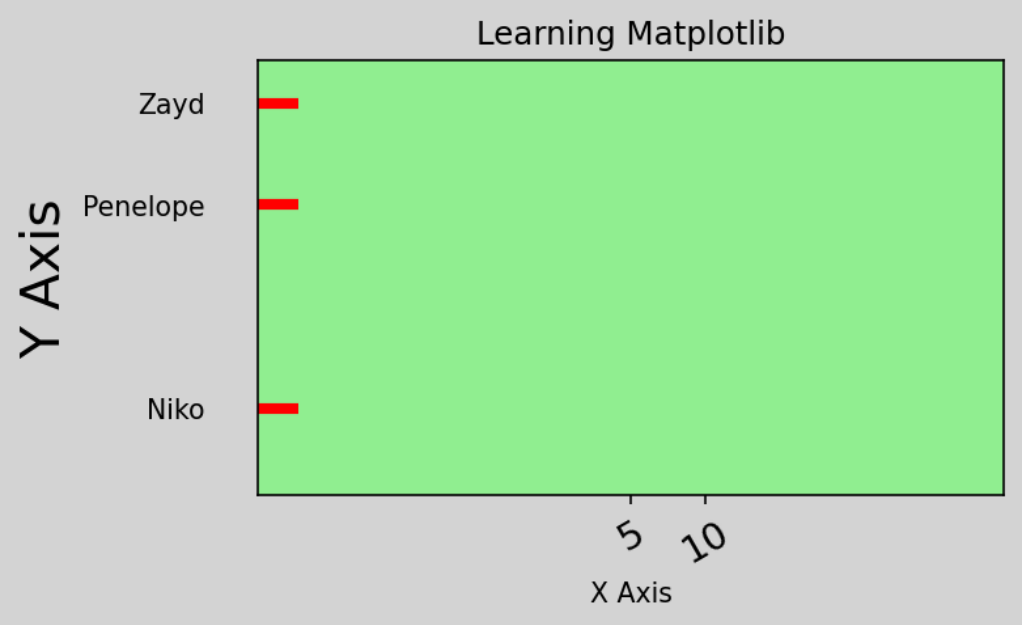
12.1 1. Introduction to matplotlib

[1]:
```
import matplotlib.pyplot as plt
```

Exercise 1

Create a figure with dimensions 5 inches by 3 inches with 147 DPI containing a single axes. Set the facecolor of the figure and the axes. Set a title and labels for the x and y axis. Set three ticks on the y-axis, and two on the x-axis. Give the three ticks on the y-axis a new string label. Change the limits of the x-axis and y-axis so they are larger than the minimum and maximum tick values. Change the size, shape, and color of the y-axis tick lines. Increase the size of the x tick labels and rotate them.

[2]:
```
fig, ax = plt.subplots(figsize=(5, 3), dpi=147)
fig.set_facecolor('lightgray')
ax.set_facecolor('lightgreen')
ax.set_title('Learning Matplotlib')
ax.set_ylabel('Y Axis', size=20)
ax.set_xlabel('X Axis')
ax.set_xticks([5, 10])
ax.set_yticks([-9, 5, 12])
ax.set_yticklabels(['Niko', 'Penelope', 'Zayd'])
ax.set_xlim(-20, 30)
ax.set_ylim(-15, 15)
ax.tick_params(axis='y', direction='in', length=15, width=4, pad=20, color='red')
ax.tick_params(axis='x', labelsize=14, labelrotation=30)
```



12.2 2. Matplotlib Text and Lines

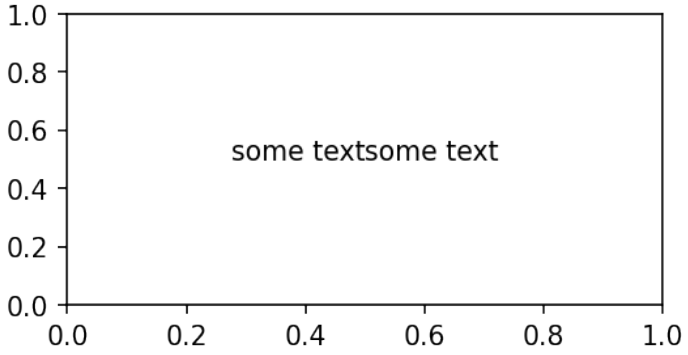
Create a new figure and axes for each exercise.

[3]:
```
import matplotlib.pyplot as plt
```

Exercise 1

Add the same text to the same location, but set the horizontal alignment of each so that they don't overlap.

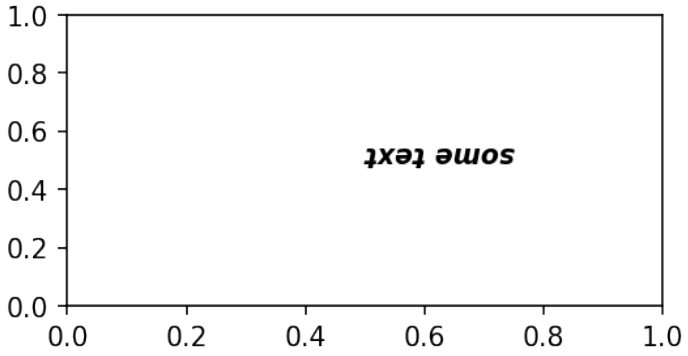
[4]:
```
fig, ax = plt.subplots(figsize=(4, 2), dpi=147)
ax.text(x=.5, y=.5, s='some text', ha='left')
ax.text(x=.5, y=.5, s='some text', ha='right');
```



Exercise 2

Add text so that it is upside down. Use `fontweight` and `fontstyle` to make the text bold and italic.

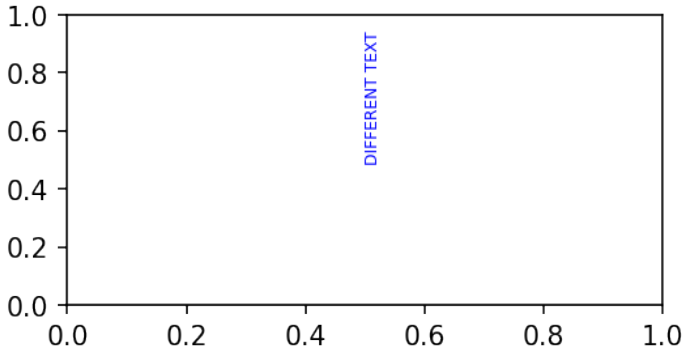
[5]:
```
fig, ax = plt.subplots(figsize=(4, 2), dpi=147)
ax.text(x=.5, y=.5, s='some text', rotation=180, fontweight='bold',␣
↪fontstyle='italic');
```



Exercise 3

Add a simple piece of text using no other parameters other than `x`, `y`, and `s`. Assign the result to a variable and then use the setter methods to set several properties.

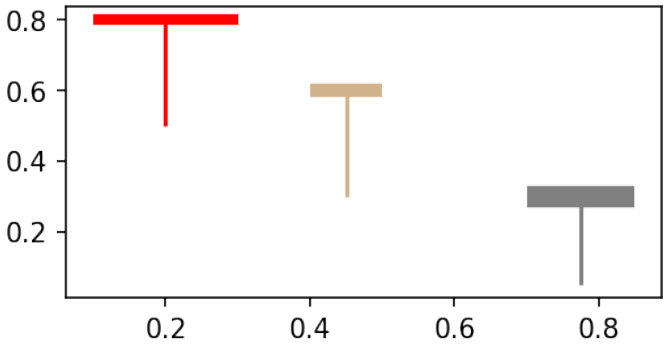
[6]:
```
fig, ax = plt.subplots(figsize=(4, 2), dpi=147)
text = ax.text(x=.5, y=.5, s='some text')
text.set_text('DIFFERENT TEXT')
text.set_rotation(90)
text.set_fontsize(6)
text.set_color('blue');
```



Exercise 4

Create three "T's" using `hlines` and `vlines` in different locations, each with a different color and line width.

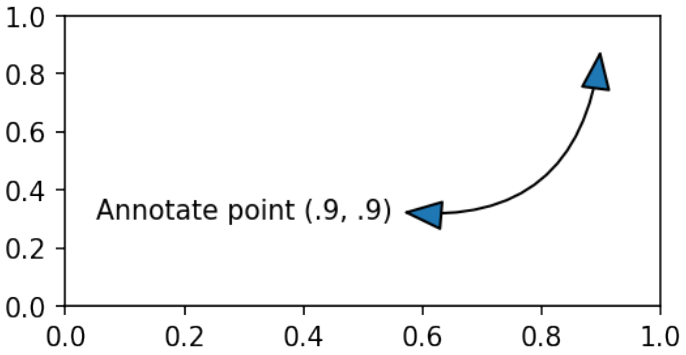
[7]:
```
fig, ax = plt.subplots(figsize=(4, 2), dpi=147)
ax.hlines(y=[.8, .6, .3], xmin=[.1, .4, .7], xmax=[.3, .5, .85],
          colors=['red', 'tan', 'gray'], linewidth=[4, 5, 8])
ax.vlines(x=[.2, .45, .775], ymin=[.5, .3, .05], ymax=[.8, .6, .3],
          colors=['red', 'tan', 'gray'])
```



Exercise 5

Annotate a point with a double-headed arrow.

[8]:
```
fig, ax = plt.subplots(figsize=(4, 2), dpi=147)
arrowprop_dict = {'arrowstyle': '<|-|>, head_length=1.3, head_width=.5',
                  'connectionstyle': 'arc3, rad=.5', 'relpos': (1, .5)}
ax.annotate(s='Annotate point (.9, .9)', xytext=(.3, .3), xy=(.9, .9),
            arrowprops=arrowprop_dict, ha='center');
```



12.3 3. Matplotlib Resolution

Create a new figure and axes for each exercise.

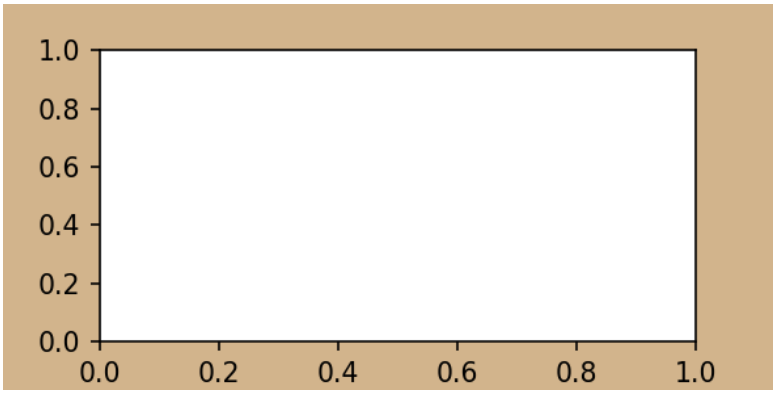
[9]:
```
import matplotlib.pyplot as plt
```

Exercise 1

Find your screen's DPI and verify it by creating a matplotlib figure with that DPI. Measure the figure with a ruler to verify that the figure-inches match the screen-inches. Set the `'bbox_inches'` notebook setting to `None` and then back to `'tight'` after the exercise.

[10]:
```
%config InlineBackend.print_figure_kwargs = {'bbox_inches': None}
```

[11]:
```
fig, ax = plt.subplots(figsize=(4, 2), dpi=147, facecolor='tan')
```



Exercise 2

If you create a figure that has a height of 3 inches and a DPI of 120 and add a line that has a width of 144 points, what fraction of the screen height will the line represent?

[12]:
```
# DPI has nothing to do with this
144 / (3 * 72)
```

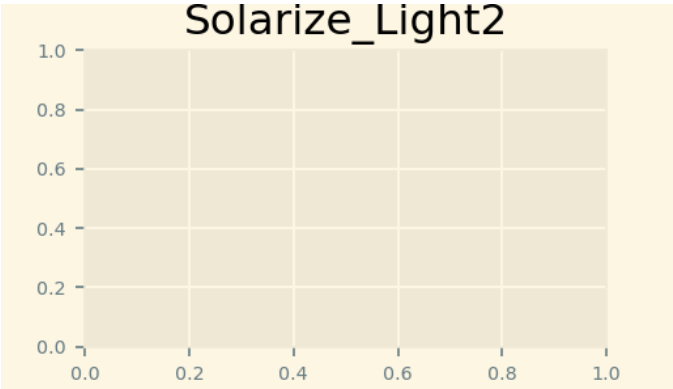
[12]:
```
0.6666666666666666
```

Exercise 3

Iterate through all the available styles creating a figure and axes with each one. Put the name of the style in the axes title.

Only a few styles are show to avoid long output.

[13]:
```
for style in plt.style.available[:3]:
    plt.style.use(['default', '../../mdap.mplstyle', style])
    fig, ax = plt.subplots()
    ax.set_title(style)
```



Exercise 4

Create your own style sheet. You might want to begin by copying one of the pre-made stylesheets.

[14]:
```
```

12.4 4. Matplotlib Patches and Colors

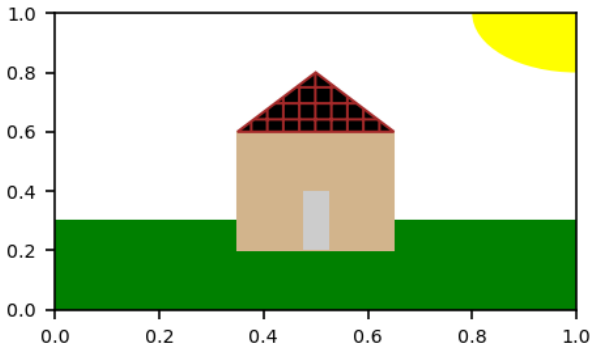
Create a new figure and axes for each exercise.

[15]:
```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import patches
plt.style.use(['default', '../../mdap.mplstyle'])
```

Exercise 1

Create a simple house with a roof, door, yard, and sun shinning overhead with matplotlib patches.

[16]:
```
fig, ax = plt.subplots()
ax.add_patch(patches.Rectangle((0, 0), width=1, height=.3, color='green'))
ax.add_patch(patches.Rectangle((.35, .2), width=.3, height=.4, color='tan'))
ax.add_patch(patches.Polygon([[.35, .6], [.5, .8], [.65, .6]],
                             fc='black', hatch='++', ec='brown'))
ax.add_patch(patches.Rectangle((.475, .2), width=.05, height=.2, fc='.8'))
ax.add_patch(patches.Circle((1, 1), radius=.2, fc='yellow'));
```



Exercise 2

Convert the RGB float triple `(.7, .03, .8)` to a hexadecimal string. Do so without using matplotlib, then verify with one of the functions from the `colors` module.

[17]:
```
hex(round(.7 * 255))
```

[17]:
```
'0xb2'
```

[18]:
```
hex(round(.03 * 255))
```

[18]:
```
'0x8'
```

[19]:
```
hex(round(.8 * 255))
```

[19]:
```
'0xcc'
```

The string should be `'#b208cc'`

[20]:
```
from matplotlib import colors
```

[21]:
```
colors.to_hex((.7, .03, .8))
```

[21]:
```
'#b208cc'
```

Exercise 3

Convert the RGB string `'#7cf2bb'` to an RGB float triple. Do so without using matplotlib, then verify with one of the functions from the `colors` module.

[22]:
```
r = int('7c', base=16) / 255
g = int('f2', base=16) / 255
b = int('bb', base=16) / 255
r, g, b
```

[22]:
```
(0.48627450980392156, 0.9490196078431372, 0.7333333333333333)
```

[23]:
```
colors.to_rgb('#7cf2bb')
```

[23]:
```
(0.48627450980392156, 0.9490196078431372, 0.7333333333333333)
```

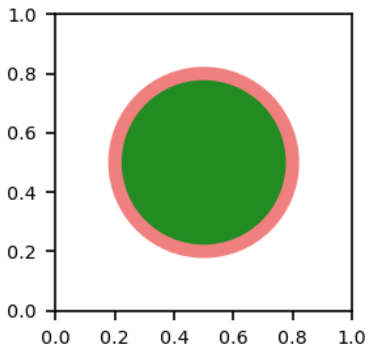
Exercise 4

Create a circle that visually looks like a circle that has edge color of lightcoral and face color of forestgreen. Use the hexadecimal representation for these strings. Increase the edge width so that it is more visible.

[24]:
```
colors.to_hex('lightcoral'), colors.to_hex('forestgreen')
```

[24]:
```
('#f08080', '#228b22')
```

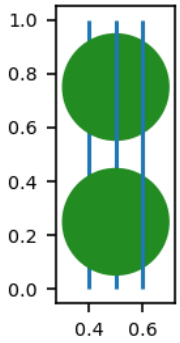
[25]:
```
fig, ax = plt.subplots()
ax.set_aspect('equal')
ax.add_patch(patches.Circle((.5, .5), radius=.3, ec='#f08080', fc='#228b22', lw=5))
```



Exercise 5

Create two circles of the same size vertically next to one another (same x-value, different y-value) without any overlap. Draw three vertical lines that pass through both circles. Place one of these lines underneath both circles, another on top of the top circle and below the bottom circle, and the last line on top of both circles.

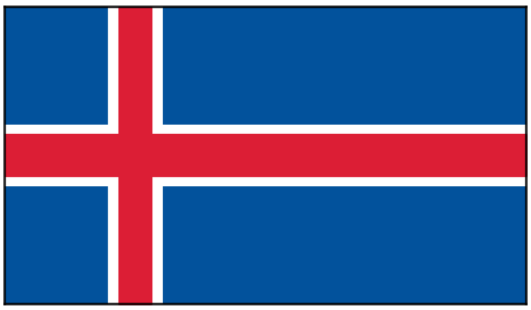
[26]:
```
fig, ax = plt.subplots()
ax.set_aspect('equal')
ax.add_patch(patches.Circle((.5, .75), radius=.2, fc='forestgreen', zorder=1))
ax.add_patch(patches.Circle((.5, .25), radius=.2, fc='forestgreen', zorder=3))
ax.vlines(x=.5, ymin=0, ymax=1, zorder=2)
ax.vlines(x=.4, ymin=0, ymax=1, zorder=0)
ax.vlines(x=.6, ymin=0, ymax=1, zorder=4);
```



Exercise 6

Create the flag of Iceland with matplotlib. Search online for its RGB color values.

[27]:
```
fig, ax = plt.subplots(figsize=(3.5, 2))
ax.set_xticks([])
ax.set_yticks([])
ax.set_xlim(0, 1)
ax.set_ylim(0, 1)
ax.set_facecolor('#02529C')
ax.fill_between(x=[0, 1], y1=.4, y2=.6, color='#FFFFFF')
ax.fill_betweenx(y=[0, 1], x1=.2, x2=.3, color='#FFFFFF')
ax.fill_between(x=[0, 1], y1=.43, y2=.57, color='#DC1E35')
ax.fill_betweenx(y=[0, 1], x1=.22, x2=.28, color='#DC1E35');
```

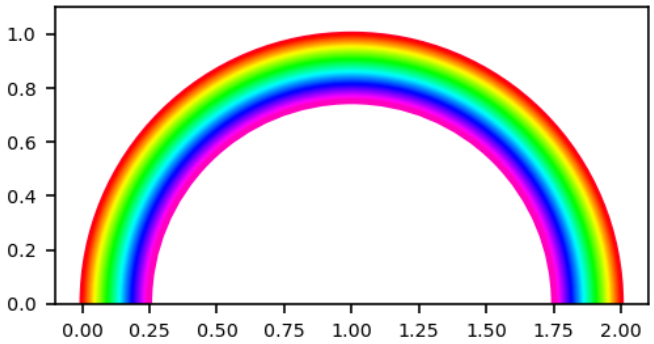


Exercise 7

Create a rainbow. Use a colormap that has the colors of the rainbow.

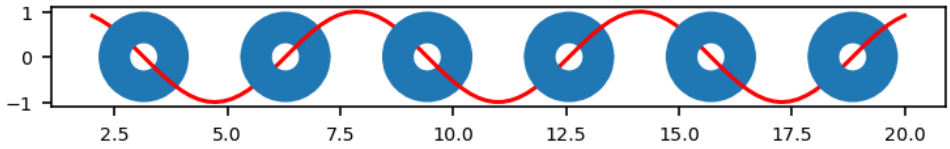
[28]:
```
from matplotlib import cm
import numpy as np
```

```
fig, ax = plt.subplots(figsize=(4, 2))
ax.set_aspect('equal')
ax.set_xlim(-.1, 2.1)
ax.set_ylim(0, 1.1)
for val in np.linspace(0, 1, 50):
    ax.add_patch(patches.Arc((1, 0), width=2 - val / 2, height=2 - val / 2,
                             theta1=0, theta2=180, lw=2, color=cm.gist_rainbow(val)))
```

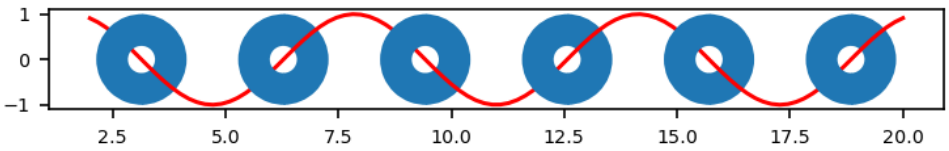


Exercise 8

Recreate the following image in matplotlib.



[29]:
```
fig, ax = plt.subplots(figsize=(6, 1))
ax.set_aspect('equal')
x = np.linspace(2, 20, 100)
y = np.sin(x)
ax.plot(x, y, color='red')
for i in range(1, 7):
    ax.add_patch(patches.Wedge((i * np.pi, 0), r=1, theta1=87, theta2=273,
                               width=.7, zorder=3))
    ax.add_patch(patches.Wedge((i * np.pi, 0), r=1, theta1=270, theta2=90,
                               width=.7, zorder=1))
```



12.5 5. Matplotlib Line Plots

Create a new figure and axes for each exercise.

[30]:
```
import matplotlib.pyplot as plt
plt.style.use('../../mdap.mplstyle')
from matplotlib import patches
import pandas as pd
```

[31]:
```
pd.set_option('display.max_columns', 100)

cols = ['GrLivArea', 'GarageArea', 'TotalBsmtSF',
        'OverallQual', 'Neighborhood', 'SalePrice']
housing = pd.read_csv('../data/housing.csv', usecols=cols)

housing
```

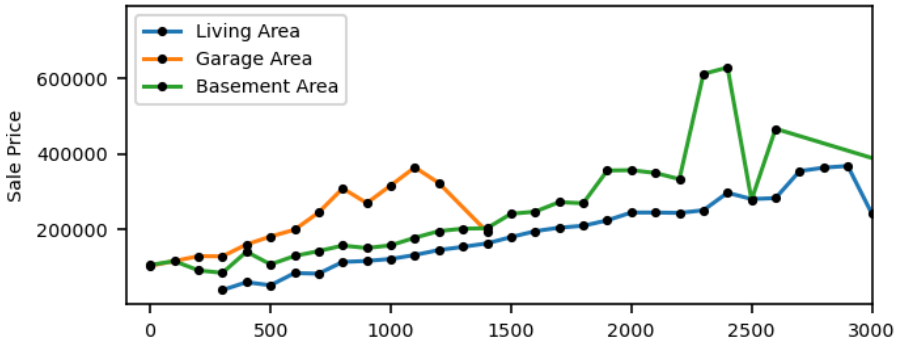
	Neighborhood	OverallQual	TotalBsmtSF	GrLivArea	GarageArea	SalePrice
0	CollgCr	7	856	1710	548	208500
1	Veenker	6	1262	1262	460	181500
2	CollgCr	7	920	1786	608	223500
3	Crawfor	7	756	1717	642	140000
4	NoRidge	8	1145	2198	836	250000
...	...	...	...	...	...	...
1455	Gilbert	6	953	1647	460	175000
1456	NWAmes	6	1542	2073	500	210000
1457	Crawfor	7	1152	2340	252	266500
1458	NAmes	5	1078	1078	240	142125
1459	Edwards	5	1256	1256	276	147500

1460 rows × 6 columns

Exercise 1

Create line plots of the average sale price per every 100 square feet of living area, garage area, and basement area. Place markers at every point and add a legend.

[32]:
```
housing = housing.round({'GrLivArea': -2, 'GarageArea': -2, 'TotalBsmtSF': -2})
price_per_lasf = housing.groupby('GrLivArea').agg(avg_sale_price=('SalePrice', 'mean'))
price_per_gasf = housing.groupby('GarageArea').agg(avg_sale_price=('SalePrice',␣
↪'mean'))
price_per_bsf = housing.groupby('TotalBsmtSF').agg(avg_sale_price=('SalePrice',␣
↪'mean'))
fig, ax = plt.subplots(figsize=(5, 2))
ax.plot(price_per_lasf, marker='.', mec='black', mfc='black', ms=5, label='Living␣
↪Area')
ax.plot(price_per_gasf, marker='.', mec='black', mfc='black', ms=5, label='Garage␣
↪Area')
ax.plot(price_per_bsf, marker='.', mec='black', mfc='black', ms=5, label='Basement␣
↪Area')
ax.set_xlim(-100, 3000)
ax.set_xlabel('Square Feet')
ax.set_ylabel('Sale Price')
ax.legend();
```



Exercise 2

For every neighborhood that has at least 100 homes, find the average sale price by each overall quality between 3 and 8 (inclusive). Plot each neighborhood as a line with overall quality on the x-axis and sale price on the y-axis. Use one of the qualitative colormaps other than the default tab10. Add a legend inside the bounds of the axes that has a frame, a title, and all labels on one row.

[33]:
```
s = housing['Neighborhood'].value_counts()
neigh_ct = s[s >= 100]
neigh_ct
```

[33]:
```
NAmes      225
CollgCr    150
OldTown    113
Edwards    100
Name: Neighborhood, dtype: int64
```

[34]:
```
neigh = neigh_ct.index
h2 = housing.query('Neighborhood in @neigh and 3 <= OverallQual <= 8')
h2.head(3)
```

	Neighborhood	OverallQual	TotalBsmtSF	GrLivArea	GarageArea	SalePrice
0	CollgCr	7	900	1700	500	208500
2	CollgCr	7	900	1800	600	223500
8	OldTown	7	1000	1800	500	129900

[35]:
```
housing
```

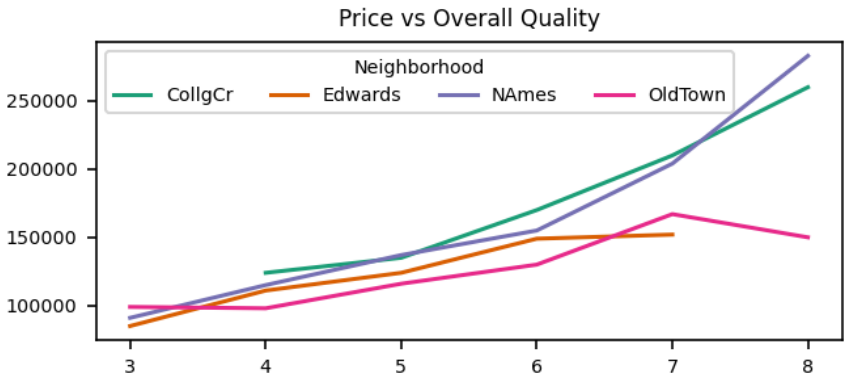
	Neighborhood	OverallQual	TotalBsmtSF	GrLivArea	GarageArea	SalePrice
0	CollgCr	7	900	1700	500	208500
1	Veenker	6	1300	1300	500	181500
2	CollgCr	7	900	1800	600	223500
3	Crawfor	7	800	1700	600	140000
4	NoRidge	8	1100	2200	800	250000
...	...	...	...	...	...	...
1455	Gilbert	6	1000	1600	500	175000
1456	NWAmes	6	1500	2100	500	210000
1457	Crawfor	7	1200	2300	300	266500
1458	NAmes	5	1100	1100	200	142125
1459	Edwards	5	1300	1300	300	147500

1460 rows × 6 columns

[36]:
```
pt = h2.pivot_table(index='OverallQual', columns='Neighborhood',
                    values='SalePrice', aggfunc='mean').round(-3)
pt
```

Neighborhood	CollgCr	Edwards	NAmes	OldTown
OverallQual				
3	NaN	85000.0	91000.0	99000.0
4	124000.0	111000.0	115000.0	98000.0
5	135000.0	124000.0	137000.0	116000.0
6	170000.0	149000.0	155000.0	130000.0
7	210000.0	152000.0	204000.0	167000.0
8	260000.0	NaN	283000.0	150000.0

[37]:
```
fig, ax = plt.subplots(figsize=(5, 2))
ax.set_prop_cycle(color=cm.Dark2(range(10)))
lines = ax.plot(pt)
ax.legend(handles=lines, labels=pt.columns.tolist(), frameon=True,␣
 ↪title='Neighborhood', ncol=4)
ax.set_title('Price vs Overall Quality');
```



12.6 6. Matplotlib Scatter and Bar Plots

Create a new figure and axes for each exercise.

[38]:
```
import matplotlib.pyplot as plt
import pandas as pd
plt.style.use('../../mdap.mplstyle')
```

Exercise 1

Read in the bikes dataset and select 500 rows of data at random. Filter for rides with a trip duration less than the 95th percentile. Remove any rows that have obviously bad data for temperature and wind speed. Make a scatter plot with temperature and trip duration as the x and y variables. Color by gender and size by wind speed using a qualitative color map. Use this tutorial to create two separate legends, one for gender, and the other for wind speed.

[39]:
```
bikes = pd.read_csv('../data/bikes.csv')
bikes.head(3)
```

	gender	starttime	stoptime	tripduration	from_station_name	...	to_station_name	end_capacity	temperature	wind_speed	events
0	Male	2013-06-28 19:01:00	2013-06-28 19:17:00	993	Lake Shore Dr & Monroe St	...	Michigan Ave & Oak St	15.0	73.9	12.7	mostlycloudy
1	Male	2013-06-28 22:53:00	2013-06-28 23:03:00	623	Clinton St & Washington Blvd	...	Wells St & Walton St	19.0	69.1	6.9	partlycloudy
2	Male	2013-06-30 14:43:00	2013-06-30 15:01:00	1040	Sheffield Ave & Kingsbury St	...	Dearborn St & Monroe St	23.0	73.0	16.1	mostlycloudy

3 rows × 11 columns

[40]:
```
bikes2 = bikes.sample(500, random_state=40)
bikes2[['tripduration', 'temperature', 'wind_speed']].describe([.95])
```

	tripduration	temperature	wind_speed
count	500.000000	500.000000	500.000000
mean	696.688000	42.822800	-49.603800
std	442.267262	450.314855	773.790169
min	88.000000	-9999.000000	-9999.000000
50%	569.500000	66.900000	10.400000
95%	1498.400000	84.900000	19.600000
max	3838.000000	93.000000	31.100000

95th percentile of trip duration is about 1500. The minimum for both temperature and wind_speed is -9999.

[41]:
```
bikes3 = bikes2.query('tripduration < 1500 and temperature > -50 and wind_speed > 0').
↪copy()
bikes3.shape
```

[41]:
```
(455, 11)
```

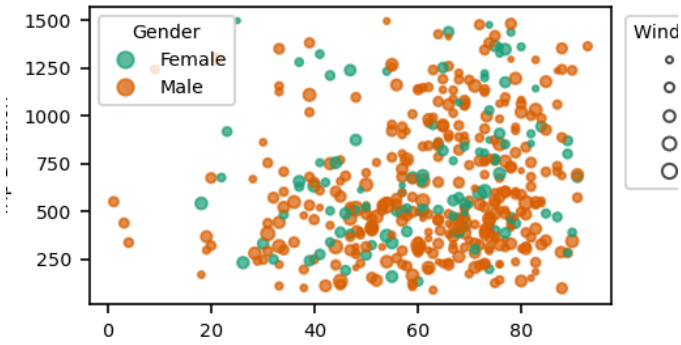
[42]:
```
bikes3['gender'] = bikes2['gender'].astype('category')
bikes3['gender_code'] = bikes3['gender'].cat.codes
gender_categories = bikes3['gender'].cat.categories.tolist()
gender_categories
```

[42]:
```
['Female', 'Male']
```

Wind speed happens to have values that make for good scatter point sizes, so we don't have to transform it.

[43]:
```
fig, ax = plt.subplots()
scatter_obj = ax.scatter('temperature', 'tripduration', s='wind_speed',␣
↪c='gender_code',
```

```
                        vmin=0, vmax=7, data=bikes3, cmap='Dark2', alpha=.7)
handles_colors, labels_colors = scatter_obj.legend_elements('colors')
handles_sizes, labels_sizes = scatter_obj.legend_elements('sizes', num=5, mfc='white')
ax.add_artist(ax.legend(handles=handles_colors, labels=gender_categories,
                        title='Gender', loc='upper left'))
ax.legend(handles_sizes, labels_sizes, title='Wind Speed', bbox_to_anchor=(1.01, 1))
ax.set_xlabel('Temperature')
ax.set_ylabel('Trip Duration');
```



[44]:

Part XIII

Visualization with Pandas and Seaborn

Chapter 13

Solutions

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

[1]:
```
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
Quinnipiac University	1090.0	6503.0	0	57700.0	forestgreen	80.641181
CUNY York College	900.0	7245.0	0	38700.0	forestgreen	85.117566
Manchester University	995.0	1254.0	1	37600.0	violet	35.411862
Creighton University	1183.0	3977.0	1	57100.0	violet	63.063460
Livingstone College	723.0	1301.0	1	24100.0	violet	36.069378

[2]:
```
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'});
```

```
NameError: name 'plt' is not defined
```

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.

[3]:
```
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

[4]:
```
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

[5]:
```
fig, ax = plt.subplots(figsize=(6, 4))
ax.pie('Count', labels='Salary Range', data=df_cat);
```

```
NameError: name 'plt' is not defined
```

13.1 5. Plotting with Pandas

[6]:
```
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 retunrs normally distributed?

[7]:
```
stocks = pd.read_csv('../data/stocks/stocks10.csv', index_col='date',␣
↪parse_dates=['date'])
stocks.head(3)
```

date	MSFT	AAPL	SLB	AMZN	TSLA	XOM	WMT	T	FB	V
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

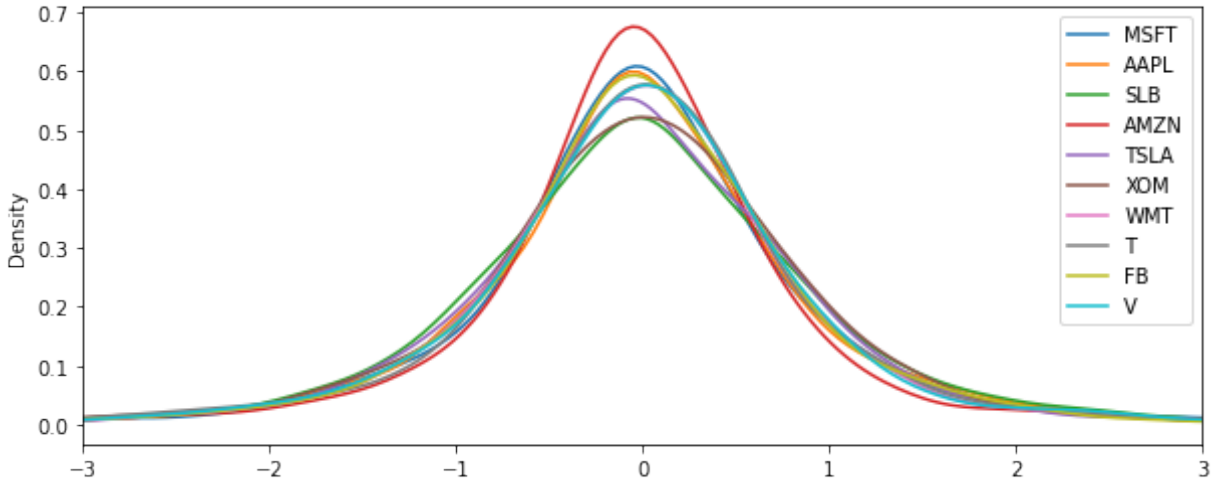
[8]:
```
stock_pct_change = stocks.pct_change()
stock_pct_change.head(3)
```

date	MSFT	AAPL	SLB	AMZN	TSLA	XOM	WMT	T	FB	V
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

[9]:
```
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()
```

```
NameError: name 'np' is not defined
```

[10]:
```
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 are 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.

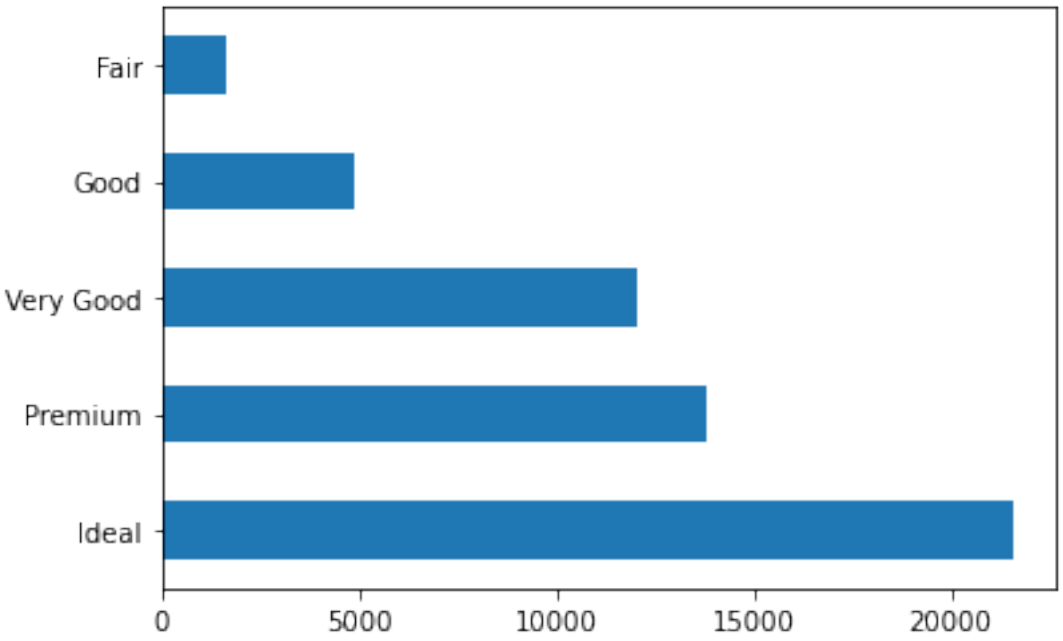
[11]:
```
diamonds = pd.read_csv('../data/diamonds.csv')
diamonds.head(3)
```

	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

[12]:
```
cut_count = diamonds['cut'].value_counts()
cut_count
```

[12]:
```
Ideal        21551
Premium      13791
Very Good    12082
Good          4906
Fair          1610
Name: cut, dtype: int64
```

[13]:
```
cut_count.plot(kind='barh');
```



[14]:
```
plt.style.use(['ggplot'])
```

```
NameError: name 'plt' is not defined
```

Exercise 3

Make a visualization that easily shows the differences in average salary by sex for each department of the employee dataset.

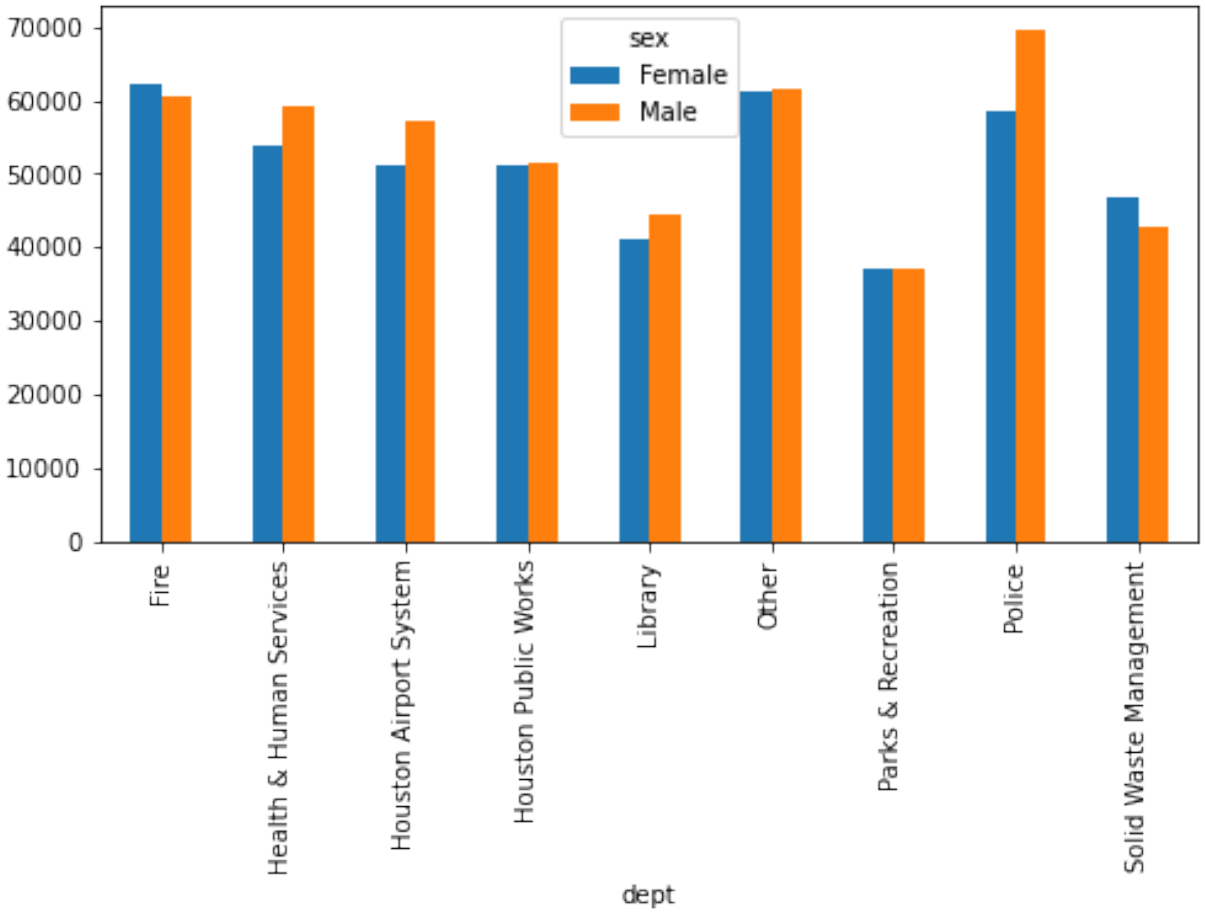
[15]:
```
emp = pd.read_csv('../data/employee.csv', parse_dates=['hire_date'])
emp.head(3)
```

	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

[16]:
```
dept_sex_sal = emp.pivot_table(index='dept', columns='sex', values='salary')
dept_sex_sal.head(3)
```

sex	Female	Male
dept		
Fire	62212.637250	60479.306862
Health & Human Services	53838.310780	59230.425956
Houston Airport System	51099.300226	57278.306598

[17]:
```
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.

[18]:
```
criteria = emp['hire_date'].dt.year >= 2000
```

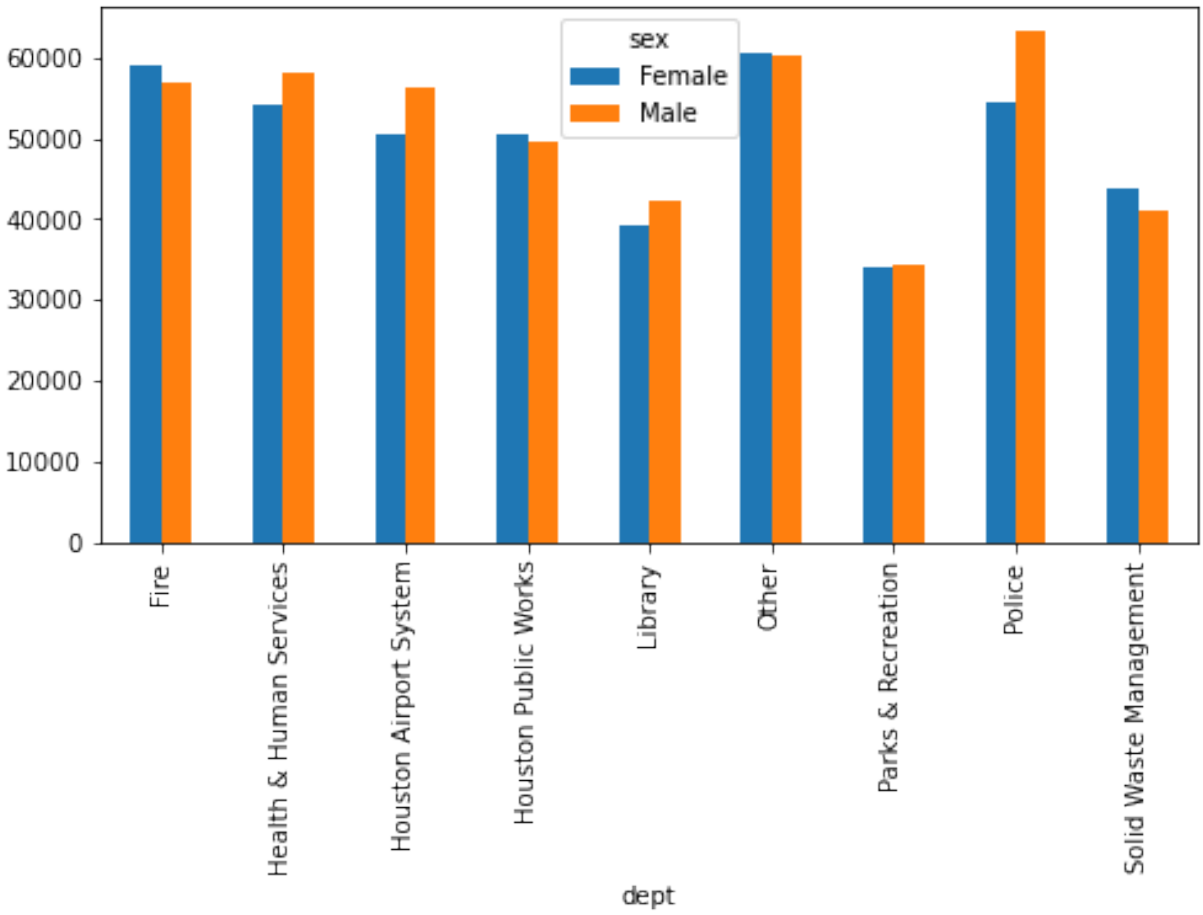
[19]:
```
emp_2000_and_later = emp[criteria]
emp_2000_and_later.head(3)
```

	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

[20]:
```
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)
```

sex	Female	Male
dept		
Fire	59185.247973	57016.700243
Health & Human Services	54267.524020	58113.680000
Houston Airport System	50402.349398	56314.303226

[21]:
```
dept_gender_sal_2000_and_later.plot(kind='bar', figsize=(8, 4));
```



[22]:
```
emp_before_2000 = emp[~criteria]
emp_before_2000.head(3)
```

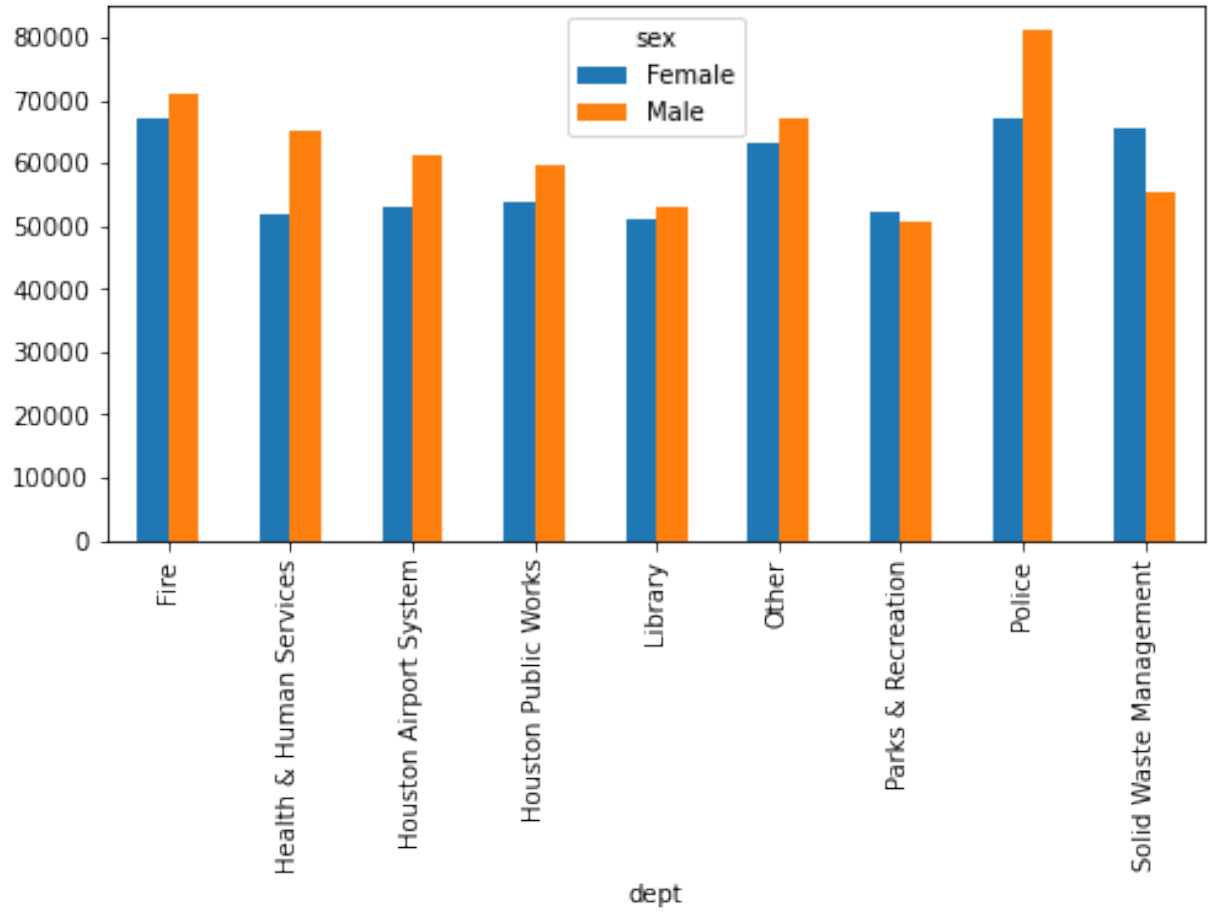
	dept	title	hire_date	salary	sex	race
3	Police	SENIOR POLICE OFFICER	1997-05-27	75942.10	Male	Hispanic
9	Police	SENIOR POLICE OFFICER	1993-08-30	75942.10	Male	Black
17	Police	POLICE COMMANDER	1983-02-07	115821.42	Male	White

[23]:
```
dept_sex_sal_before_2000 =  emp_before_2000.pivot_table(index='dept',
                                                        columns='sex',
                                                        values='salary')
dept_sex_sal_before_2000.head(3)
```

sex	Female	Male
dept		
Fire	67082.785217	71182.220495
Health & Human Services	51989.924731	65284.364561
Houston Airport System	53183.873874	61184.725490

Can drop missing values so that we can compare

[24]:
```
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.

[25]:
```
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

3 rows × 14 columns

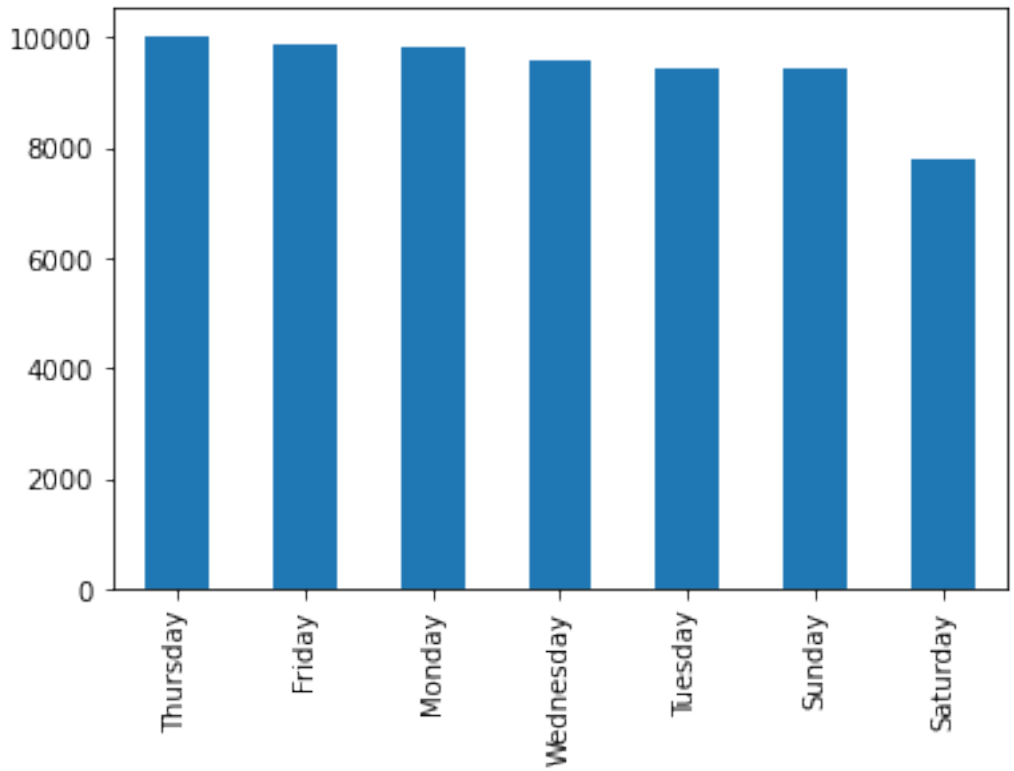
Exercise 5

Use the `flights` data set. Plot the counts of the number of flights per day of week.

[26]:
```
weekday_ct = flights['date'].dt.day_name().value_counts()
weekday_ct
```

[26]:
```
Thursday     10022
Friday        9872
Monday        9828
Wednesday     9581
Tuesday       9428
Sunday        9410
Saturday      7782
Name: date, dtype: int64
```

[27]:
```
weekday_ct.plot(kind='bar');
```

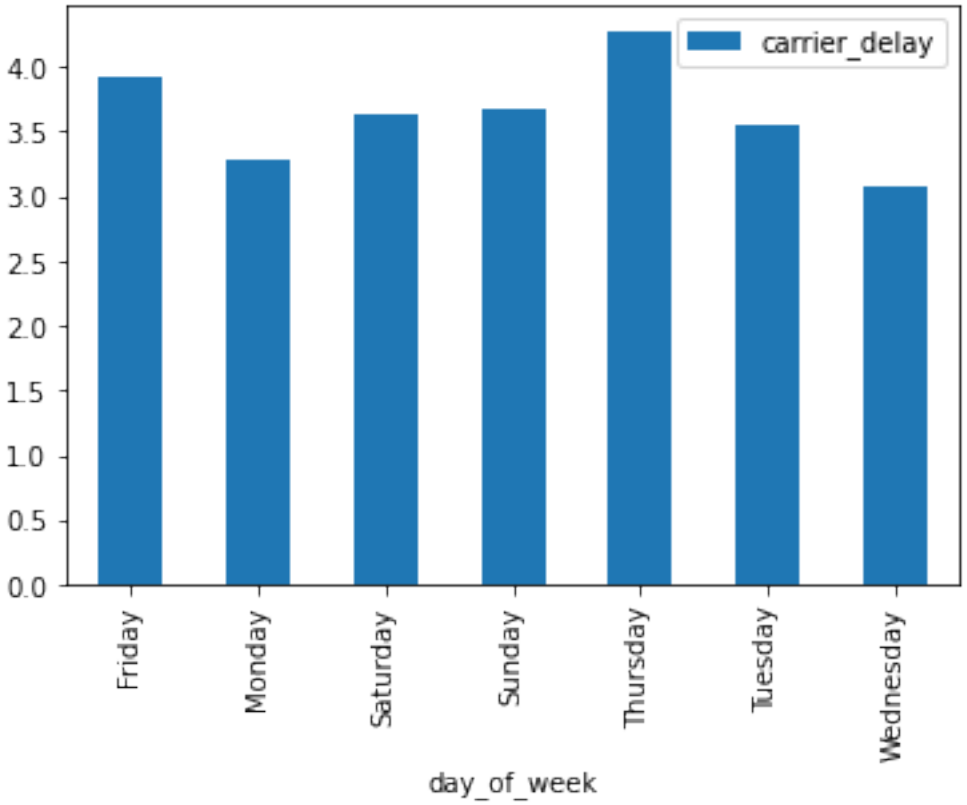


Exercise 6

Plot the average carrier delay per day of week.

[28]:
```
flights['day_of_week'] = flights['date'].dt.day_name()
```

[29]:
```
flights.groupby('day_of_week').agg({'carrier_delay': 'mean'}).plot(kind='bar');
```



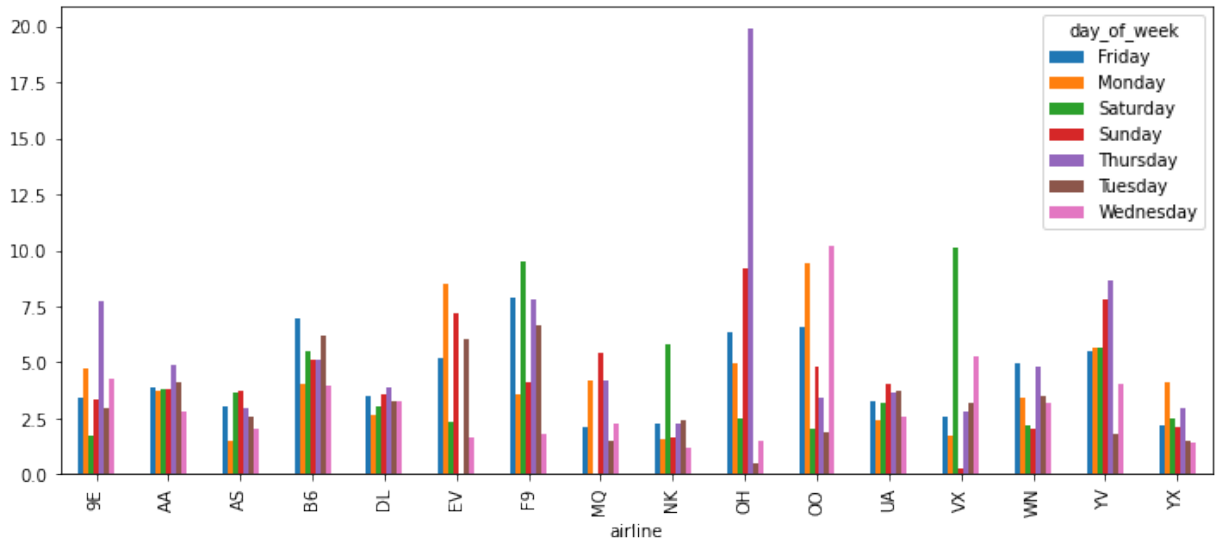
Exercise 7

Plot the average carrier delay per day of week per airline.

[30]:
```
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

[31]:
```
airline_weekday_delay.plot(kind='bar', figsize=(12, 5));
```



13.2 6. Seaborn