Hints and answers

This document contains hints and answers to the practical exercises in

Getting Started with Python Data Analysis

ISBN: 9781785285110.

Chapter 1

Movie recommendation.

Exercise 1 What information can we find in this table? What kind of knowledge can we derive from it?

There is some information we can find in the table:

* "Snow White" movie ranking from men in Canada is less than 3.

* Average ranking of "Snow white" movie is less than 3 from male and greater than 3 from female.

We can learn (a little) about users preferences for a movie. Combined with other data about movies, we could suggest each user other movies, given how much they like the "Snow White" movies. For example, some knowledge we can derive such as:

* Females may like the movie but males may not

*Men in Canada may dislike "Snow White" movie

Exercise 2 Based on the data analysis process in this chapter, try to define the data requirements and analysis steps needed to predict, whether user B likes 'Maleficent' movies or not?

To be able to build a recommendation system, we would need to know a bit more about other movies and possibly other users preferences. We then could apply different techniques. One would be collaborative filtering. A toy implementation could, for a given user and his most liked movies, find other users, who liked the same movies. Then, movies that those people liked, could be recommended to the given user as well.

The concrete steps could look like follows:

- data collection: collect more data on users and movies
- data processing: access database or API, perform basic extract-transform-load (ETL)
- data cleaning: clean the data, bring it in shape for other tasks
- exploratory data analysis: compute sums, averages and other basic statistical measures
- modelling and algorithm: test simple models - like the toy model above - first, gradually refine model and test
- data product: build a pipeline that takes data from the data sources through the refined model to the user and offer recommendations

Chapter 2

Exercise 1 Using array creation function, let's try to create arrays in the following situations:

1. Creating `ndarray` from existing data:

```
>>> import numpy as np
>>> np.array(1)
array(1)
```

```
>>> np.array([1, 2, 3])
array([1, 2, 3])
>>> np.array([1, 2, 3])
array([1, 2, 3])
>>> np.array([range(4), range(2)])
array([[0, 1, 2, 3], [0, 1]], dtype=object)
>>> np.array([range(4) for _ in range(4)])
array([[0, 1, 2, 3],
       [0, 1, 2, 3],
       [0, 1, 2, 3],
       [0, 1, 2, 3]])
>>> np.array([[range(4) for _ in range(3)] for _ in range(2)])
array([[[0, 1, 2, 3],
        [0, 1, 2, 3],
        [0, 1, 2, 3]],
       [[0, 1, 2, 3],
        [0, 1, 2, 3],
        [0, 1, 2, 3]]])
```

2. Initializing ndarray which elements are filled with ones, zeros or a given interval

```
>>> np.ones((3, 4))
array([[ 1.,  1.,  1.,  1.],
       [ 1.,  1.,  1.,  1.],
       [ 1.,  1.,  1.,  1.]])
>>> np.zeros((3, 4))
array([[ 0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.]])
>>> np.arange(0, 100, 8)
array([ 0,  8, 16, 24, 32, 40, 48, 56, 64, 72, 80, 88, 96])
```

3. Loading and saving data from file to an ndarray

```
>>> # Taken from
http://docs.scipy.org/doc/numpy/reference/generated/numpy.save.html
>>> from tempfile import TemporaryFile
>>> outfile = TemporaryFile()
>>> x = np.arange(10)
>>> np.save(outfile, x)
>>> outfile.seek(0) # Only needed here to simulate closing & reopening file
>>> np.load(outfile)
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

Exercise 2 What is the difference between np.dot(a, b) and (a*b)

```
>>> np.dot(np.arange(3), np.arange(3))5
```

```
>>> np.arange(3) * np.arange(3)
array([0, 1, 4])
```

The first computes the dot product, which results in a scalar. The second performs elementwise multiplication. The elementwise multiplication can be turned into a dot product by summing over the elements:

```
>>> (np.arange(3) * np.arange(3)).sum()
5
```

Exercise 3 Consider the vector `[1, 2, 3, 4, 5]`, build a new vector with four consecutive zeros interleaved between each value.

```
>>> v = np.arange(1, 6)
>>> x = np.zeros(v.shape[0] + 4 * 4)
>>> x[::5] = v
>>> x
array([ 1.,  0.,  0.,  0.,  0.,  2.,  0.,  0.,  0.,  0.,  3.,  0.,  0.,
    0.,  0.,  4.,  0.,  0.,  0.,  0.,  5.])
```

Exercise 4 Taking the data example file "chapter2-data.txt" which includes five information of a system log, solves the following tasks:

- Try to build an `ndarray` from the data file
- Statistic frequency of each device type in the built matrix
- List unique OS appear in the data log
- Sort user by `provinceID` and count the number of user in each province

Answer:

- Try to build an `ndarray` from the data file

```
>>> Udata, Ddata, OSdata, Sdata, Pdata = np.loadtxt('\chapter 2\exerci
se\chapter2-data.txt', delimiter='\t', dtype={'names':('userId', 'Device',
'OS',
 'sex', 'provinceID'), 'formats': ('S16', 'S16', 'S16', 'S16', 'i4')},
skiprows=
1, unpack=True)
```

- Statistic frequency of each device type in the built matrix

```
>>> D_unique, D_freq_count = np.unique(Ddata, return_counts=True)
>>> device_freq_count = np.asarray((D_unique, D_freq_count)).T
>>> device_freq_count
array([[b'General_Desktop', b'341'],
       [b'General_Mobile', b'130'],
       [b'General_Tablet', b'29']],
      dtype='|S21')
```

- List unique OS appear in the data log

```
>>> OS_unique = np.unique(OSdata)
>>> OS_unique
array([b'Android', b'Mac OS X', b'MeeGo', b'Nokia Series 40', b'Other',
       b'Symbian OS', b'Ubuntu', b'Windows', b'Windows 7', b'Windows 8',
       b'Windows Phone', b'Windows XP', b'iOS'],
```

```
      dtype='|S16')
```

- Sorting user by `provinceID` and count the number of user in each province

```
//sorted user by provinceID
>>> index = np.argsort(Pdata)
>>> sorted_user = Udata[index]
//counting the number of user in each province
>>> uniq_p, p_num_count = np.unique(Pdata, return_counts=True)
>>> user_of_province = np.asarray((uniq_p, p_num_count)).T
>>> user_of_province
array([[   0,   51],
       [   1,    1],
       [   2,    3],
       [   3,    2],
       [   6,    2],
       [   7,    1],
       [   8,    1],
       [   9,    2],
       [  13,    6],
       [  15,    5],
       [  16,    1],
       [  19,    1],
       [  24,  226],
       [  25,    3],
       [  27,    4],
       [  29,  132],
       [  31,    1],
       [  32,    2],
       [  33,    1],
       [  34,    1],
       [  37,    1],
       [  38,    1],
       [  47,    8],
       [  48,    1],
       [  49,    1],
       [  55,    2],
       [  56,    2],
       [  57,    2],
       [  58,    2],
       [  59,    1],
       [  61,    1],
       [9999,   32]], dtype=int64)
```

Chapter 3

The link https://www.census.gov/2010census/csv/pop_change.csv contains an US census data set. It has 23 columns and one row for each US state as well as a few rows for macro regions like North, South or West.

Exercise 1 Get this data set into a Pandas DataFrame. Hint: Just skip those rows, that do not seem helpful, like comments or description.

```
>>> import pandas as pd
>>> df = pd.read_csv("https://www.census.gov/2010census/csv/pop_change.csv",
                     skiprows=2, index_col=0)
>>> df[df.columns[:3]].head()
                  1910_POPULATION  1920_POPULATION  1930_POPULATION
STATE_OR_REGION
United States            92228531        106021568        123202660
Northeast                25868573         29662053         34427091
Midwest                  29888542         34019792         38594100
South                    29389330         33125803         37857633
West                      7082086          9213920         12323836
```

Exercise 2 While the data set contains change metrics for each decade, we are interested in the population change during the second half of the twentieth century, that is between 1950 and 2000. Which region has seen the biggest and the smallest population growth in this time span? Which US state? Note: There is no single way to come up with the correct answers. Rather than searching for the one way, we would like to encourage you to try various strategies. For example, you can could create a new data frame, that only contains the columns you are interested in, or you could just work with the original one.

```
>>> clip = df[["1950_POPULATION", "2000_POPULATION"]]
>>> clip.head()
                  1950_POPULATION  2000_POPULATION
STATE_OR_REGION
United States           151325798        281421906
Northeast                39477986         53594378
Midwest                  44460762         64392776
South                    47197088        100236820
West                     20189962         63197932
>>> clip["diff"] = df["2000_POPULATION"] - df["1950_POPULATION"]
>>> clip.head()
                  1950_POPULATION  2000_POPULATION       diff
STATE_OR_REGIONUnited States           151325798        281421906  130096108
Northeast
          39477986         53594378   14116392
Midwest                  44460762         64392776   19932014
South                    47197088        100236820   53039732
West                     20189962         63197932   43007970
>>> clip.sort("diff").tail()
                  1950_POPULATION  2000_POPULATION       diff
STATE_OR_REGION
Midwest                  44460762         64392776   19932014
California               10586223         33871648   23285425
West                     20189962         63197932   43007970
South                    47197088        100236820   53039732
United States           151325798        281421906  130096108
>>> clip.sort("diff").head()
```

STATE_OR_REGION	1950_POPULATION	2000_POPULATION	diff
District of Columbia	802178	572059	-230119
West Virginia	2005552	1808344	-197208
North Dakota	619636	642200	22564
South Dakota	652740	754844	102104
Wyoming	290529	493782	203253

Answers: The macro region United States exhibits, unsurprisingly the largest increase, followed by the South macro region. The largest population increase could be observed in California. The smallest (in fact, negative) population growth between 1950 and 2000 has been registered in the Northeast region and in the District of Columbia and West Virginia, respectively.

Exercise 3 Advanced open ended exercise: Find more census data on the internet, not just on the US, but on the world's countries. Try to find GDP data for the same time as well. Try to align this data to explore patterns. How are GDP and population growth related? Are there special cases, like countries with high GDP but low population growth? Or countries with the opposite history?

Hints:

There are numerous data sources with varying level of accessibility. We provide some entry points to authoritative sources below:

- The https://www.census.gov/ collects and publishes data on the US. There are dumps, CSV files and even an API: https://www.census.gov/data/developers/data-sets.html
- Some census data on Europe can be found as zipped CSV files under http://ec.europa.eu/eurostat/web/population-and-housing-census/census-data/database
- The CIA world factbook is a popular general purpose data source on most of the worlds countries. One example page on GDP data can be found here: https://www.cia.gov/library/publications/the-world-factbook/fields/2195.html The complete publication can be found here: https://www.cia.gov/library/publications/resources/the-world-factbook/
- More data sources are listed in this Stack Exchange answer: http://stats.stackexchange.com/questions/27237/what-are-the-most-useful-sources-of-economics-data

Chapter 4

Exercise 1 Name two real or fictional datasets and explain which kind of plot would fit the data best: line plots, bar charts, scatter plots, contour plots or histograms. Name one or two applications, where each of the plot types is common (for example histograms are often used in image editing applications).

- **line plots**: oil price, stock price, exchange rates, in general data that changes over time, non-cyclical data over many periods
- bar **charts**: exports and imports per country, student grades, in general for comparing categorical data
- **scatter plots**: petal length and sepal length of various species of iris flowers, GDP and life expectancy of various countries, in general to show the relationship between two variables.
- **contour plots**: elevation maps, weather maps (isobar, isotherm, isotach, isodrosotherm, ...), in general to visualize a function of two variables in a 2D diagram
- **histograms**: color distribution in images, age distribution for visitors of a museum, in general to estimate the probability distribution of a continous variable

Exercise 2 We only focused on the most common plot types of matplotlib. After a bit of research, can you name a few more plot types, that are available in matplotlib?

More plot types:

- boxplots
- stackplots
- quiver plot
- tricontour
- polar bar charts, polar scatter charts
- radar charts
- 3D variants of the 2D plots, like scatter 3D, wire 3D, ...

Exercise 3 Take one pandas data structure from chapter three and plot the data in a suitable way, then save it as a PNG image to disk.

Here is a short program, that displays the population of a subset of the states over the 20th century:

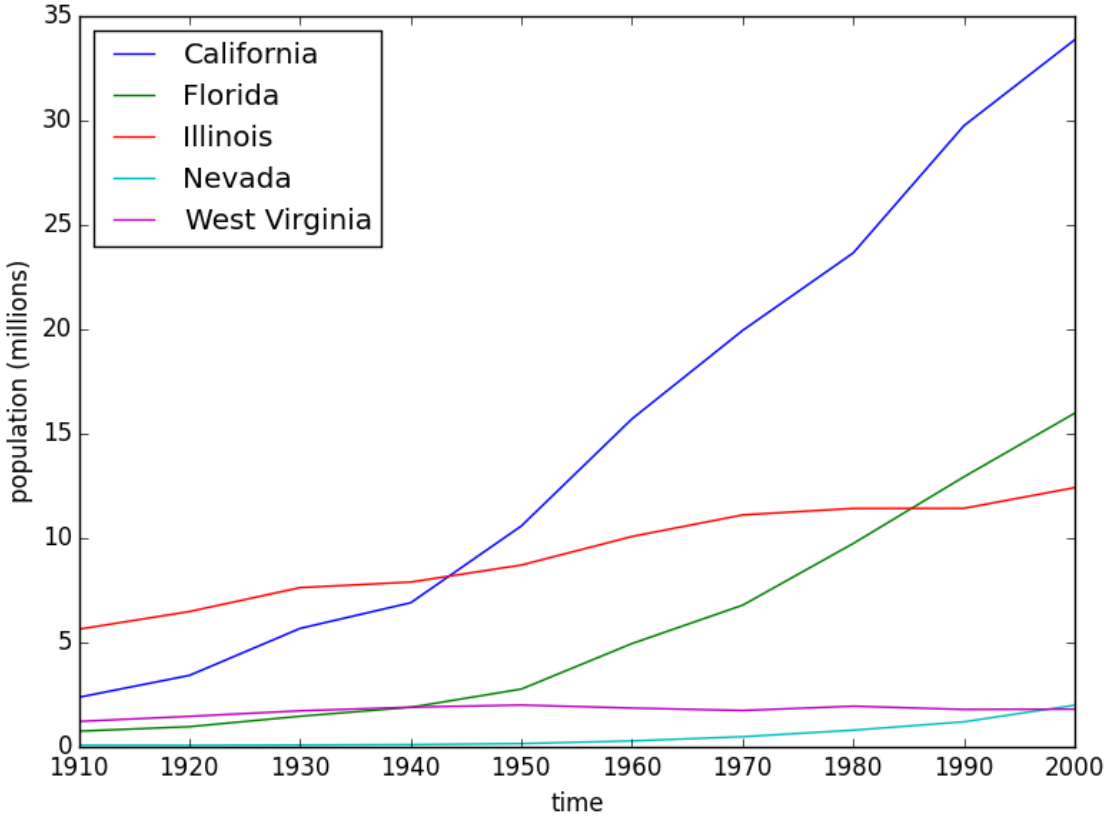
```
#!/usr/bin/env python
import pandas as pd
import matplotlib.pyplot as plt
df = pd.read_csv("https://www.census.gov/2010census/csv/pop_change.csv",
                         skiprows=2, index_col=0)
fig, ax = plt.subplots(1, 1)
df.ix[[
    "California",
    "Florida",
    "Illinois",
    "Nevada",
    "West Virginia",
]].ix[:, :10].T.plot(kind='line', ax=ax)
labels = map(str, range(1910, 2010, 10))
ax.set_xticklabels(labels)

plt.ticklabel_format(style='plain', axis='y')
ax.set_yticklabels(["0", "5", "10", "15", "20", "25", "30", "35"])

plt.xlabel("time")
plt.ylabel("population (millions)")
plt.legend(loc='upper left')

plt.tight_layout()
plt.savefig('chapter4-3.png')
```

The output will look like this:



Chapter 5

1. Find one or two real world examples for data sets, which could - in a sensible way - be assigned to the following groups:
    - Fixed frequency data.
      number of daily visitors in the Louvre
      monthly revenue report for a company
    - Variable frequency data.
      time series of purchases on a point of sales terminal
      any data with missing values, that has not been cleaned
    - Data where frequency is usually measured in seconds.
      requests per seconds in webserver traffic
      counts per second in a Geiger-Müller counter
    - Data where frequency is measured in nanoseconds.
      Large Hadron Collider Logging Service
      (https://cds.cern.ch/record/1000757/files/ab-note-2006-046.pdf)
    - Data, where a TimedeltaIndex would be preferable.

data attached to a countdown of a rocket launch

in general: measurements taken from a fixed starting point

2. Create various fixed frequency ranges:

   - Every minute between 1AM and 2AM on 2000-01-01

```
import pandas as pd pd.date_range(start='2000-01-01T01:00:00', freq='min',
periods=60) DatetimeIndex(['2000-01-01 01:00:00', '2000-01-01 01:01:00',
'2000-01-01 01:02:00', '2000-01-01 01:03:00', '2000-01-01 01:04:00', '2000-
01-01 01:05:00', '2000-01-01 01:06:00', '2000-01-01 01:07:00', '2000-01-01
01:08:00', '2000-01-01 01:09:00', '2000-01-01 01:10:00', '2000-01-01
01:11:00', '2000-01-01 01:12:00', '2000-01-01 01:13:00', '2000-01-01
01:14:00', '2000-01-01 01:15:00', '2000-01-01 01:16:00', '2000-01-01
01:17:00', '2000-01-01 01:18:00', '2000-01-01 01:19:00', '2000-01-01
01:20:00', '2000-01-01 01:21:00', '2000-01-01 01:22:00', '2000-01-01
01:23:00', '2000-01-01 01:24:00', '2000-01-01 01:25:00', '2000-01-01
01:26:00', '2000-01-01 01:27:00', '2000-01-01 01:28:00', '2000-01-01
01:29:00', '2000-01-01 01:30:00', '2000-01-01 01:31:00', '2000-01-01
01:32:00', '2000-01-01 01:33:00', '2000-01-01 01:34:00', '2000-01-01
01:35:00', '2000-01-01 01:36:00', '2000-01-01 01:37:00', '2000-01-01
01:38:00', '2000-01-01 01:39:00', '2000-01-01 01:40:00', '2000-01-01
01:41:00', '2000-01-01 01:42:00', '2000-01-01 01:43:00', '2000-01-01
01:44:00', '2000-01-01 01:45:00', '2000-01-01 01:46:00', '2000-01-01
01:47:00', '2000-01-01 01:48:00', '2000-01-01 01:49:00', '2000-01-01
01:50:00', '2000-01-01 01:51:00', '2000-01-01 01:52:00', '2000-01-01
01:53:00', '2000-01-01 01:54:00', '2000-01-01 01:55:00', '2000-01-01
01:56:00', '2000-01-01 01:57:00', '2000-01-01 01:58:00', '2000-01-01
01:59:00'], dtype='datetime64[ns]', freq='T', tz=None)
```

   - Every two hours for a whole week starting 2000-01-01

```
pd.date_range(start='2000-01-01', end='2000-01-07', freq='2H')
DatetimeIndex(['2000-01-01 00:00:00', '2000-01-01 02:00:00', '2000-01-01
04:00:00', '2000-01-01 06:00:00', '2000-01-01 08:00:00', '2000-01-01
10:00:00', '2000-01-01 12:00:00', '2000-01-01 14:00:00', '2000-01-01
16:00:00', '2000-01-01 18:00:00', '2000-01-01 20:00:00', '2000-01-01
22:00:00', '2000-01-02 00:00:00', '2000-01-02 02:00:00', '2000-01-02
04:00:00', '2000-01-02 06:00:00', '2000-01-02 08:00:00', '2000-01-02
10:00:00', '2000-01-02 12:00:00', '2000-01-02 14:00:00', '2000-01-02
16:00:00', '2000-01-02 18:00:00', '2000-01-02 20:00:00', '2000-01-02
22:00:00', '2000-01-03 00:00:00', '2000-01-03 02:00:00', '2000-01-03
04:00:00', '2000-01-03 06:00:00', '2000-01-03 08:00:00', '2000-01-03
10:00:00', '2000-01-03 12:00:00', '2000-01-03 14:00:00', '2000-01-03
16:00:00', '2000-01-03 18:00:00', '2000-01-03 20:00:00', '2000-01-03
22:00:00', '2000-01-04 00:00:00', '2000-01-04 02:00:00', '2000-01-04
04:00:00', '2000-01-04 06:00:00', '2000-01-04 08:00:00', '2000-01-04
10:00:00', '2000-01-04 12:00:00', '2000-01-04 14:00:00', '2000-01-04
16:00:00', '2000-01-04 18:00:00', '2000-01-04 20:00:00', '2000-01-04
22:00:00', '2000-01-05 00:00:00', '2000-01-05 02:00:00', '2000-01-05
04:00:00', '2000-01-05 06:00:00', '2000-01-05 08:00:00', '2000-01-05
10:00:00', '2000-01-05 12:00:00', '2000-01-05 14:00:00', '2000-01-05
16:00:00', '2000-01-05 18:00:00', '2000-01-05 20:00:00', '2000-01-05
22:00:00', '2000-01-06 00:00:00', '2000-01-06 02:00:00', '2000-01-06
04:00:00', '2000-01-06 06:00:00', '2000-01-06 08:00:00', '2000-01-06
10:00:00', '2000-01-06 12:00:00', '2000-01-06 14:00:00', '2000-01-06
16:00:00', '2000-01-06 18:00:00', '2000-01-06 20:00:00', '2000-01-06
22:00:00', '2000-01-07 00:00:00'], dtype='datetime64[ns]', freq='2H',
tz=None)
```

   - An entry for every Saturday and Sunday during the year 2000

```
reduce(operator.add, [pd.date_range(start='2000', end='2001', freq=freq)
for freq in ['W-SAT', 'W-SUN']])
```

   Or simply use concatenate:

```
>>> pd.date_range(start='2000', end='2001', freq='W-SAT') +
pd.date_range(start='2000', end='2001', freq='W-SUN')
```

- An entry for every Monday of a month, for the years 2000, 2001 and 2002

```
pd.date_range(start='2000', end='2003', freq='W-MON')
DatetimeIndex(['2000-01-03', '2000-01-10', '2000-01-17', '2000-01-24',
'2000-01-31', '2000-02-07', '2000-02-14', '2000-02-21', '2000-02-28',
'2000-03-06', ... '2002-10-28', '2002-11-04', '2002-11-11', '2002-11-
18', '2002-11-25', '2002-12-02', '2002-12-09', '2002-12-16', '2002-12-
23', '2002-12-30'], dtype='datetime64[ns]', length=157, freq='W-MON',
tz=None)
```

Chapter 6

1. Take a data set of your choice and design storage options for it. Consider text files, HDF5, a document database and a data structure store as possible persistent options. Also evaluate, how difficult (by some metric, e.g. how many lines of code) it would be to update or delete a specific item. Which storage type is the easiest to set up? Which storage type supports the most flexible queries?

   Hints: Text files are usually the easiest to set up, but they do come with possible consistency problems. SQL databases have great support for ad-hoc queries.

2. In chapter 3 we saw, that it is possible to create hierarchical indices with Pandas. As an example assume that you have data on each city with more than 1M inhabitant and that we have a two level index, so we can address individual cities, but also whole countries. How would you represent this hierarchical relationship with the various storage options presented in this chapter: text files, HDF5, MongoDB and Redis? What do you believe would be the most convenient to work with in the long run?

   Hints: With textfiles, we could store the countries under one directory and each data set for single city in a separate file. In HDF5, we could use a country group. Since groups are nestabled, we could use a group for each city as well, if we might expect more data sets in the future. Since MongoDB is a document oriented data store, we would add keys for country and city to each document, so we can queries these attributes. In redis, we could define a custom key, e.g. country-city and use this as a key and the data as value. The most suitable format will depend on your application. It is often useful to start with something simple.

Chapter 7

Cleaning: In the section about filtering, we used the Europe Brent Crude Oil Spot Price, which can be found as an Excel document on the internet. Take this Excel spreadsheet and try to convert it into a CSV document that is ready to be imported with Pandas.

Hint: There are many ways to do this. We used a small tool called xls2csv.py and we were able to load the resulting `CSV` file with a helper method:

```
import datetime
import pandas as pd
def convert_date(s)
    parts = s.replace("(", "").replace(")", "").split(",")
```

```
    if len(parts) < 6:
        return datetime.date(1970, 1, 1)
    return datetime.datetime(*[int(p) for p in parts])
df = pd.read_csv("RBRTEd.csv", sep=',', names=["date", "price"],
                 converters={"date": convert_date}).dropna()
```

Take a data set, that is important for your work - or if you do not have any at hand, some data set that interests you and that is available online. Ask one or two questions about the data in advance. Then use cleaning, filtering, grouping and plotting techniques to answer your question.

Hints: If you are stuck, try to look at the documentation: http://pandas.pydata.org/pandas-docs/stable/.

The community around Pandas is very active, maybe you find your problems answered in one of the following posts: http://stackoverflow.com/questions/tagged/pandas.

Chapter 8

3. Are the following problems supervised or unsupervised? Regression or classification problems?
    - Recognizing coins inside a vending machine.
    - This is a classification problem, since the number of coins is usually small. It can be framed both in a supervised and unsupervised fashion, depending on the availability of training data.
    - We want to recognize handwritten digits.
    - This is a classification problem. Most of the time, this will be a supervised problem. The MNIST database of handwritten digits has been one of the popular research problem in the past years.
    - Given a number of facts about people and economy, we want to estimate consumer spending.
    - This is a regression problem, since the output is continuous. Usually, we would start with existing data, so again this would be a supervised problem.
    - Given data about geography, politics and historical events, we want to predict when and where a human right violation will eventually take place.
    - We will most likely start from existing data, so this would a supervised problem. The output could be a probability, so this would belong the class of regression problems.
    - Given sounds of whales and their species, we want to label yet unlabelled whale sound recordings.
    - This can be formulated in a supervised and unsupervised way. In the unsupervised case, we would be interested in detecting clusters (since we know the number of species is limited). It is a classification problem.
4. Lookup one of the first machine learning models and algorithms: The perceptron. Try the perceptron on the Iris data set. Estimate the accuracy of the model. How does the Perceptron compare to the SVC from the chapter?
    - The perceptron is a bit weaker than SVC.